Universidad de Granada

UNIVERSIDAD DE GRANADA
Departamento de Física Teórica y del Cosmos
Área de Física de la Tierra

Evolución, diseño y desarrollo de antenas sísmicas. Las antenas del Gran Sasso, del Vesubio y las nuevas antenas sísmicas portátiles del Instituto Andaluz de Geofísica. Aplicación a zonas tectónicas y volcánicas.

Miguel Abril Martí
Granada, 2007

Editor: Editorial de la Universidad de Granada
Autor: Miguel Abril Martí
D.L.: Gr. 1297 - 2007
ISBN: 978-84-338-4373-9

Universidad de Granada

UNIVERSIDAD DE GRANADA
Departamento de Física Teórica y del Cosmos
Área de Física de la Tierra

Evolución, diseño y desarrollo de antenas sísmicas. Las antenas del Gran Sasso, del Vesubio y las nuevas antenas sísmicas portátiles del Instituto Andaluz de Geofísica. Aplicación a zonas tectónicas y volcánicas.

Directores:

Dr. Gerardo Alguacil de la Blanca

Dr. Jesús Miguel Ibáñez Godoy

Doctorando:

Miguel Abril Martí

Granada, mayo de 2007

Agradecimientos

Los agradecimientos son lo primero que hay en una tesis, y son lo primero (y casi lo único) que la mayoría de la gente se lee. Siempre había pensado que también era lo primero que se escribía, o que por lo menos desde el principio se tenía una idea clara de lo que se iba a poner. Ahora, sin embargo, me doy cuenta de que los agradecimientos son como una despedida, una especie de examen de conciencia de todo el trabajo realizado en el que se pasa revista a la gente que ha estado ahí durante este tiempo. Así que no me voy a extender mucho, no sólo porque no me gustan las despedidas sino, sobre todo, porque me cierran la fotocopiadora.

El trabajo que aquí presento lo he desarrollado durante mi estancia en el Instituto Andaluz de Geofísica de la Universidad de Granada, y es a su gente a la primera a la que quiero mandarle un abrazo. Después de casi nueve años con ellos son más una familia que compañeros de trabajo. En primer lugar quiero darles las gracias a mis directores: a Gerardo, por enseñarme todo lo que sé sobre instrumentación sísmica y por su paciencia mientras aprendía. A Jesús, por ayudarme a entender lo que quieren decir los volcanes a través de los registros de los instrumentos. Por eso y por haber estado pendiente de que no me faltara una beca o un contrato con el que poder comer, ya fuera a través de instituciones españolas, italianas o una mezcla de las dos. Confieso que la mayoría de las veces no sabía quién me pagaba, pero a final de mes siempre había un ingreso en mi cuenta. Pepe también se ha ocupado de buscarme sustento cuando los volcanes no daban más de sí. Gracias. Te debo un litro de helado de yema y aprovecho estas líneas para saludarte, por todas esas veces que no me ha dado tiempo cuando pasabas como una exhalación por el laboratorio. ¿Es un pájaro? ¿Es un avión? No, es… era Pepe. A Antonio Martos y Javi Moreno, no sólo por su gran ayuda con la electrónica e informática, sino sobre todo por hacer del laboratorio un lugar agradable y divertido. A Javi, con el que la ciencia parece fácil. Lo sabe todo, te lo sabe explicar y además sabe darle al balón con la izquierda. A Bea, que me explicó dónde estaba el baño y me enseñó a distinguir las puertas de la rotonda. A mis vecinas de despacho, Mamen y Luisa, por los ratos de charla en las tardes de invierno. A Flor y sus películas raras. A Inma y sus historias. A Fede y su moto limpia. A Mauri y su guacamole. A Daria y sus bicicletas. A Enrique Valenzuela, siempre con una sonrisa aunque le lleves una dieta difícil. A Antoñillo y sus bandas. A Antonio Peregrín (¡lo que sabe ese hombre!). A Antonio Pazos, por sus ánimos. A Beni, Merche, Elena, Paco, Jaime, Daniel, Peña, Teresa, Encarni, Mimoun, Taoufik, Asma, Manolo, José Antonio, Juan de Dios, Araceli,… A todos los que pasaron por allí y se fueron.

Enrique merece un párrafo especial. No sólo porque mi mayor aliciente para finiquitar esta tesis ha sido terminarla antes que él, para poder levantarme en su lectura cuando digan eso de ‘¿algún doctor en la sala quiere hacer alguna pregunta?’. Lo vas a pasar mal, chaval. No sólo por eso, decía, sino sobre todo por ser el mejor amigo de todos los que lo conocen. Incluso en las patadas que da jugando al fútbol se nota su cariño. Compi, seguramente ni tú ni yo ganaremos nunca el Premio Nobel de Física, pero si te lo propusieras tú podrías ganar el de la Paz. Aunque, claro, luego se lo regalarías a alguien.

A la Peña Cocoroco, por estar ahí casi dos décadas después de habernos conocido. El hecho de que algunos de sus miembros no contesten a mis mensajes desde hace años no quiere decir nada, supongo que siguen estando ahí. Será por falta de tiempo. O eso, o que soy un pesado. Entre los que sí responden, un abrazo especial para Nacho y el Peli, que me ayudaron a conseguir la plaza de la que ahora disfruto (Nacho, si llega a salir la bola de lógica difusa me quedo yo solo con las catorce plazas). Y a

Olga, por su ayuda con la plaza y por mucho más, este papel es demasiado pequeño para contarlo todo. Y a Pedro, por sus conversaciones telefónicas, cada vez más escasas debido a este asuntillo que tenía entre manos, pero que ahora podremos retomar.

Durante estos años he tenido mucho contacto con italianos, un pueblo alegre, simpático y hospitalario al que hay que perdonarle incluso su suerte ganando mundiales y su chovinismo a la hora de comer (no, Luisa, los piononos de Santa Fe NO los hacen mejor en Nápoles). Un recuerdo especial para Edoardo, por muchas cosas pero, especialmente, por haberme dado la posibilidad de conocer ese paraíso anclado en el tiempo que es Strómboli. A Marcello, impulsor y co-diseñador de la antena del Vesubio. A Walter, por sus siempre acertados comentarios. A Fabrizio, por su buen hacer y su ayuda con la antena del Gran Sasso (la gira española de Agua Calientes sigue en pie). A Gilberto, el Maestro, que sabe de todo y además es un tío encantador. Espero que algún día entiendas que no estaba loco, que cuando te preguntaba por el Dragon Khan es porque pensaba que habías pasado las vacaciones en Port Aventura, no en Fuerteventura. Gracias también a Norma, Tiziana, Carla, Mario, Danilo, Luciano, Costantino, Pasquale, Rosanna, Francesca,… A todos ellos, ¡Forza Italia!

Ahora he cambiado la geofísica por la astrofísica. De IAG a IAA sólo hay una letra y la instrumentación que hacemos viene a ser igual, siempre que tengas cuidado de no enterrar los telescopios. Sin embargo, el cambio ha sido grande. Muchos de mis nuevos compañeros ni siquiera sabían que estaba escribiendo la tesis, porque lo hacía por las tardes, pero a los que sí estaban al tanto (Raquel, Camino, Cristina, César,…) les doy las gracias por su interés. Y, especialmente, a Luis, por su impulso para que me decidiera a escribirla y su apoyo para que la terminara.

Y, como no, a mi familia. Parafraseando al gran G. A. Martí, puedo decir que '… he tenido mucha suerte en el reparto de familias que se hizo al principio de los tiempos'. Todos me han apoyado de distintas formas para concluir este trabajo. Inés no me ha ayudado con el índice ni la bibliografía, pero se lo agradezco como si lo hubiera hecho porque se ha ofrecido muchas veces. Hermana, no es que no me fiara, es que esto era algo personal entre ella (la tesis) y yo. Mis padres han estado al quite con Miguelito para permitirme sacar más horas y me han dado, además, mucha ayuda moral. Igual que Marián, que siempre ha tenido claro (más que yo) que acabaría siendo doctor. Pero sobre todo quiero mandarle un beso muy especial a Mabel, por ser la mejor madre/tía/hermana del mundo, capaz de sacar tiempo de su tiempo un fin de semana tras otro para quedarse con Miguelito y permitirme darle más vueltas a todo esto.

He dejado para el final a los dos más perjudicados por este trabajo. A Miguelito le debo una gran disculpa por todo el tiempo durante el que no he podido hacerle caso. Y a Patricia, un enorme GRACIAS por saber aguantar tantas tardes y tantos fines de semana de madre soltera, viuda o separada (según su estado de ánimo) mientras yo le daba forma a este trabajo. No podría haberlo hecho sin tu ayuda. Prometo que, a partir de ahora, iremos al cine y seré yo el que bañe a Miguelito (… ¿Cómo? ¿Qué ya se ducha solo? ¿Pero cuántos años llevo con esto?).

Granada, 27 de mayo de 2007

Vaya… ¿Alguien conoce una fotocopiadora de guardia?

Esta tesis doctoral ha sido parcialmente financiada por los siguientes proyectos de investigación:

- *Modello di velocità e di attenuazione al vulcano Stromboli*, 1997. CNR-GNV (Italia) 96.00856.PF62.
- *Studio dell'attenuaziones sismica all'Etna mediante arrays di sismometri a corto periodo*, 1998. CNR-GNV (Italia).
- *UNDERSEIS: UNDERground SEISmic array*. INGV-INFN (Italia).
- Sismicidad volcánica, local y regional del área de las Shetland del Sur y Estrecho de Bransfield (Antártida). ANT98-1111.
- *Study and constraints on intermediate storage, magma uprise and conduits through modelling of strain fields, velocity and attenuation tomography at Mt. Etna*, 2000-2002. GNV (Italia).
- *Progettazione e realizzazione di 8 sub-arrays sismici*, 1999-2001. *Osservatorio Vesuviano* (Italia)-IAG (España).
- *e-Ruption. A satellite telecommunication and Internet-based seismic monitoring system for volcanic eruption forecasting and risk management*. EU. EVR1-2001-00024.
- TOMODEC. Tomografía sísmica de alta resolución de la Isla Decepción (Antártida) y modelización de la fuente sismo-volcánica. REN2001-3833.
- Sismicidad volcánica del Teide: Tomografía de alta resolución usando datos sísmicos activos y pasivos. TOM-TEIDEVS. CGL2004-05744-C04-01.
- SIS-VOLTEDEC: Monitorización sismo-volcánica, estructura superficial y modelo cortical de la Isla Decepción (Antártida). CGL2005-05789-C03-02/ANT.
- *Volcanoes: Understanding subsurface mass movement; VOLUME*. UE, FP6-2004-Global-3-018471.

ÍNDICE

CAPÍTULO 1

INTRODUCCIÓN

Las antenas sísmicas son, como todo instrumento, sistemas concebidos para captar información de un fenómeno físico y procesarla para convertirla a un formato comprensible. La información recogida y ofrecida al usuario una vez procesada nunca va a ser una representación totalmente fiel del fenómeno, sino que se va a ver distorsionada y limitada por la respuesta del propio instrumento. Así pues, desde el punto de vista del diseño resulta de gran interés conocer las características de las señales que van a ser objeto de estudio, puesto que permitirán orientar el proyecto hacia la parte del fenómeno que más interese conocer y definir las prestaciones del sistema para captar la información de la manera más fiel posible. Es lógico, por tanto, que el primer paso en un estudio sobre cualquier tipo de instrumento sea la presentación del fenómeno que se pretende examinar, en este caso el movimiento del suelo. Así, en la sección 1.1 se presentarán brevemente las distintas señales sísmicas con que nos podemos encontrar, prestando especial atención a las de origen volcánico (sección 1.1.1).

En la segunda parte del capítulo (sección 1.2) se introducirán los sistemas de *array* o antena sísmica. Se comenzará presentando los conceptos básicos relativos a estos dispositivos y a las técnicas de análisis de datos asociadas a ellos (secciones 1.2.1 a 1.2.3), para seguir con sus orígenes y aplicaciones (1.2.4). Esta segunda parte concluirá con una revisión de los procesos que intervienen en un instrumento sísmico, en la que, teniendo en cuenta el carácter de esta tesis, se prestará especial atención a los aspectos que deben considerarse en el diseño de dispositivos de antena (sección 1.2.5).

En la tercera parte del capítulo (sección 1.3) se introducirán los tres dispositivos cuyo diseño se describe en este trabajo, definiendo en primer lugar los objetivos que se pretende cubrir con cada uno de ellos (1.3.1), para hacer luego algunos breves comentarios sobre la estructura de cada una de las antenas y los procesos que se contemplarán en su diseño (1.3.2).

Por último, en la sección 1.4 se presentará la estructura del resto de los capítulos que componen esta tesis y de los anexos que se incluyen en el CD adjunto.

1.1. Señales sísmicas

Las señales sísmicas quedan caracterizadas principalmente por tres parámetros: duración, amplitud y contenido espectral. Desde el punto de vista del desarrollo de un instrumento, son sobre todo las dos últimas las que condicionan el diseño. La amplitud, o más bien las amplitudes máximas y mínimas que se pretende registrar, están directamente relacionadas con el rango dinámico del sistema. El contenido espectral, por su parte, condiciona la respuesta en frecuencia de los distintos elementos, que debe ser tal que las señales que se pretende registrar no sufran distorsiones ni atenuaciones. En sistemas digitales, el contenido espectral impone además ciertas condiciones a la frecuencia de muestreo de los sistemas de conversión analógico digital (A/D).

El rango de amplitudes medido en sismología es muy amplio. El ruido de fondo natural, cuya amplitud es altamente dependiente de la frecuencia, representa el límite inferior de amplitudes mensurables, con valores típicos de 1 nm a 1 Hz. El límite superior queda determinado por los mayores desplazamientos registrados, correspondientes a terremotos de gran magnitud, que son del orden de un metro. Estos valores dan como resultado un rango dinámico de 10^9. La figura 1.1 muestra la densidad espectral de amplitud para ondas de terremotos de distinta magnitud.

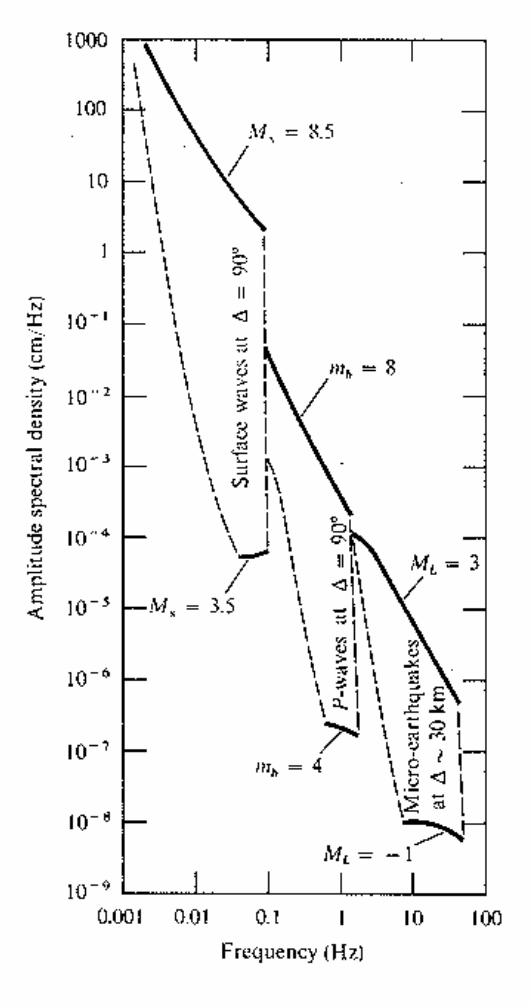


Figura 1.1. *Rangos de la densidad espectral de amplitud para ondas superficiales con magnitudes M_s entre 3.5 y 8.5, para ondas P con magnitudes m_b entre 4 y 8 y para ondas S de microterremotos con magnitudes M_L entre –1 y 3. M_S, m_b y M_L son distintas escalas de magnitud basadas en ondas superficiales de telesismos, ondas internas de telesismos y ondas de sismos locales, respectivamente (de Aki y Richards, 1980).*

El rango de frecuencias de interés en sismología es también muy extenso, como se muestra en la tabla 1.1.

Frecuencia (Hz)	*Tipo de medidas*
0.00001-0.0001	Mareas terrestres
0.0001-0.001	Oscilaciones libres de la Tierra, terremotos
0.001-0.01	Ondas superficiales, terremotos
0.01-0.1	Ondas superficiales, ondas P y S, terremotos con M > 6
0.1-10	Ondas P y S, terremotos con M > 2
10-1000	Ondas P y S, terremotos con M < 2

Tabla 1.1. *Frecuencias típicas generadas por distintas fuentes sísmicas (de Havskov y Alguacil, 2004).*

Los elevados rangos dinámico y de frecuencias de las señales sísmicas hacen imposible conseguir instrumentos capaces de registrar todo el espectro de señales. Es necesario acotar el objeto de estudio al que se va a dedicar el sistema y, una vez hecho esto, el reto es conseguir registrar la mayor parte de las señales de interés con el mayor margen dinámico posible.

Es comprensible que, dado el impacto que producen los terremotos en la actividad humana, haya sido este tipo de señales tectónicas el objetivo clásico de la sismología. Además el rango de frecuencias de estas señales hacía posible su estudio con sistemas de registro analógicos y sensores puramente mecánicos, factibles con la tecnología existente en los primeros años de investigación (ver, por ejemplo, Batlló, 2003). El gran avance que se ha

producido en las últimas décadas en materia de registradores y sensores ha hecho posible acometer otro tipo de estudios centrados en señales de baja o muy baja frecuencia, como las mareas terrestres (Anderson, 1979). Mención aparte merecen las señales de origen volcánico, que por su variedad y por la importancia que tendrán en esta tesis (dos de los tres dispositivos que se presentan tratarán habitualmente con este tipo de señales) se describirán con más detalle en la siguiente sección.

1.1.1. Señales volcánicas

Frente a la relativa homogeneidad de las señales presentes en zonas activas de tipo tectónico, en áreas volcánicas existe una gran variedad de señales. Esta diversidad proviene del hecho de que, además de los fenómenos de ruptura típicamente tectónicos, están implicados fenómenos de transporte de los fluidos presentes en los sistemas volcánicos (agua, magma y gases) (ver, por ejemplo, Ibáñez y Carmona, 2000). Las principales señales sísmicas presentes en regiones volcánicas son los terremotos volcano-tectónicos, los eventos de largo periodo, los eventos híbridos, el trémor y las explosiones. Las figuras 1.2 a 1.6 muestran distintos ejemplos de cada uno de estos tipos de señales, cuyas características se describen brevemente a continuación.

1.1.1.1. Terremotos volcano-tectónicos

Un terremoto volcano-tectónico (figura 1.2) no es más que un terremoto ocurrido en un ambiente volcánico. El origen de estos eventos es el mismo que el de los terremotos tectónicos, la acumulación de esfuerzos en estructuras internas de la corteza que se liberan súbitamente en forma de ondas elásticas. Sin embargo, debido a la fracturación normalmente existente en medios volcánicos, que impide que existan sistemas de grandes fallas, la magnitud suele ser menor que en los terremotos que se producen en zonas tectónicas. En áreas volcánicas no es común registrar terremotos con magnitudes superiores a 4, si bien en algunos casos las intensidades sí pueden ser elevadas debido a la proximidad de la fuente.

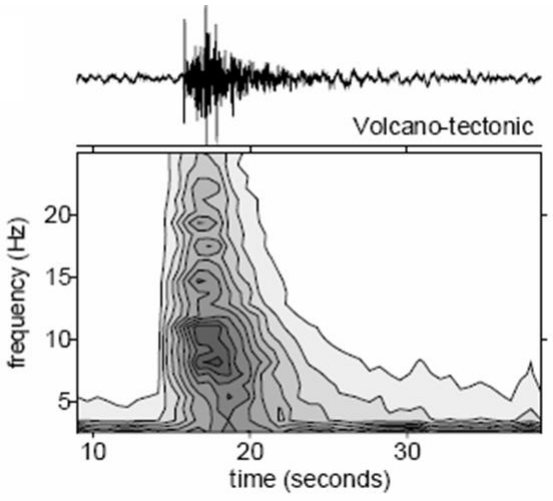


Figura 1.2. *Terremoto volcano-tectónico de la isla Decepción (Antártida). En el espectrograma, que representa el contenido espectral a lo largo del tiempo, se aprecia que existe señal por encima de los 25 Hz (de Ibáñez y Carmona, 2000).*

La señal asociada a un terremoto volcano-tectónico está caracterizada por tener una duración variable, desde unos pocos segundos hasta algunos minutos. Su comienzo suele ser más o menos impulsivo (llegada de la onda P) y es posible identificar la llegada de la onda S, especialmente si se cuenta con sistemas de registro de tres componentes. El contenido espectral de estos eventos es amplio. Es posible observar terremotos cuyos espectros presentan frecuencias superiores a los 30 Hz.

La aparición de los terremotos en regiones volcánicas suele darse en forma de lo que se conoce como enjambres sísmicos, secuencias de numerosos terremotos agrupados en el tiempo, de tamaño similar y compartiendo una misma zona epicentral. Es difícil relacionar estos enjambres con la actividad eruptiva, ya que aunque en algunos casos se ha producido un

brusco aumento de la actividad sísmica antes de una erupción, también existen precedentes de enjambres sísmicos sin una relación aparente con el inicio de la actividad eruptiva, así como erupciones sin aumento significativo de la actividad (Beniot y McNutt, 1996).

1.1.1.2. Eventos de largo periodo

Los eventos de largo periodo o LPs (figura 1.3), también conocidos como eventos de baja frecuencia o LFs, son señales típicas de ambientes volcánicos. Están caracterizados por tener una duración de entre pocos segundos hasta algo más de un minuto, y un contenido espectral muy limitado a unas bandas de frecuencia relativamente estrechas (normalmente entre 0.5 y 5 Hz).

Su forma de onda es muy característica, similar a un huso de tejer. Su comienzo suele ser emergente y no presentan llegadas definidas de ningún tipo de fase, ni P ni S, lo cual las hace bastante difíciles de localizar usando técnicas clásicas. La ocurrencia temporal de los eventos de tipo LP suele ser en forma de enjambre sísmico. Se ha podido observar (Chouet, 1996) que existe una fuerte relación entre la ocurrencia de enjambres de este tipo de eventos y la presencia de erupciones volcánicas.

En la actualidad el modelo de fuente más aceptado para este tipo de eventos está relacionado con la dinámica de los fluidos presentes en un volcán. Los LPs se generarían por resonancias en fracturas cerradas en sus extremos y rellenas de agua o magma con un cierto nivel de gas disuelto en ellas, en los que se produciría un brusco transitorio de presión (Chouet, 1996).

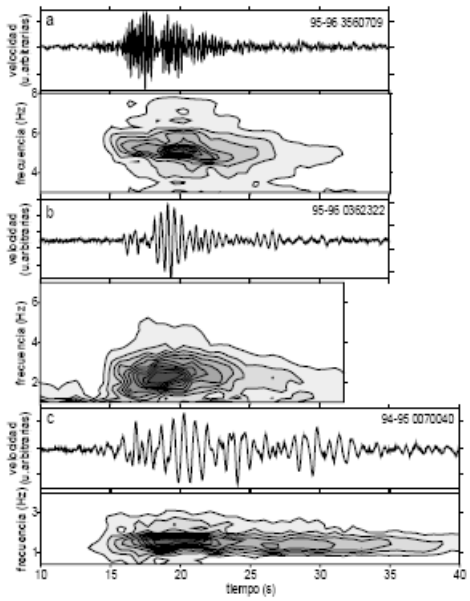


Figura 1.3. *Tres ejemplos de eventos de largo periodo de la Isla Decepción, que permiten apreciar la variedad en formas de onda y contenido espectral de este tipo de eventos (de Ibáñez y Carmona, 2000).*

1.1.1.3. Eventos híbridos

Los eventos híbridos están caracterizados por tener un comienzo con señales de altas frecuencias, normalmente dentro de una amplia banda espectral (hasta más allá de los 10 Hz), seguido de una señal muy similar en forma de onda, duración y contenido espectral a los eventos de largo periodo (figura 1.4). Por regla general la llegada en alta frecuencia de este tipo de eventos presenta ondas P y S claras.

La presencia espacial y temporal de los eventos híbridos es muy similar a la de los eventos de tipo LP, por lo que también aparecen asociadas a episodios pre-eruptivos inminentes.

El modelo de fuente más probable parte de una región fuente consistente en una fractura sellada y sometida a la presión de los fluidos volcánicos. Un aumento de presión llevaría a la ruptura de la zona,

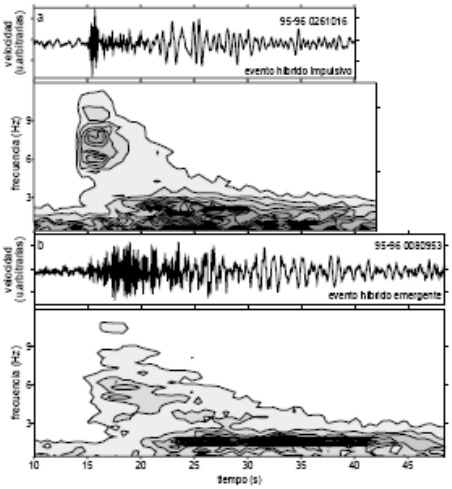


Figura 1.4. *Dos eventos híbridos de la Isla Decepción, en los que se aprecia el contenido en altas frecuencias de la llegada y la similitud de la última parte de la señal con los eventos de tipo LP (de Ibáñez y Carmona, 2000).*

originando un terremoto que produce las señales de alta frecuencia. La fractura se rellenaría de fluido y la resonancia del mismo produciría la señal monocromática en bajas frecuencias según el modelo de los eventos LP.

1.1.1.4. Trémor volcánico

En entornos volcánicos es habitual encontrar este tipo de señales, caracterizadas por mantener una amplitud constante durante un largo periodo de tiempo (entre varios minutos y horas) y presentar un contenido espectral centrado en bandas de frecuencia relativamente estrechas (figura 1.5).

La localización espacial de la fuente del trémor resulta complicada debido a la ausencia de fases identificables. La utilización de varias antenas sísmicas en una misma región ha permitido comenzar a dibujar espacialmente la localización de estas fuentes y su evolución espacial. La ocurrencia temporal del trémor puede darse tanto en la fase pre-eruptiva como en la eruptiva y post-eruptiva.

Se han propuesto múltiples modelos de fuente para el trémor volcánico, pero la ausencia de fases sísmicas que permitan la localización espacial de la fuente y su evolución temporal dificulta mucho la validación de los mismos. Por otra parte, la gran variedad espectral de este tipo de señales implica la necesidad de contar con múltiples modelos de fuente. El tipo de trémor que más se suele registrar presenta frecuencias entre 1 y 6 Hz. Sobre éste es sobre el que se han propuesto más modelos basados, entre otros procesos, en degasificaciones, fluctuaciones del gas o resonancia de conductos. Algunos estudios que integran observaciones de LPs y señales de trémor muestran evidencias de que en ocasiones el trémor puede ser resultado de la suma temporal de LPs (Almendros *et al.*, 1997), en cuyo caso los mecanismos serían los mismos que en este tipo de señales, pero con transitorios de presión continuos en el tiempo (Ibáñez *et al.*, 2000).

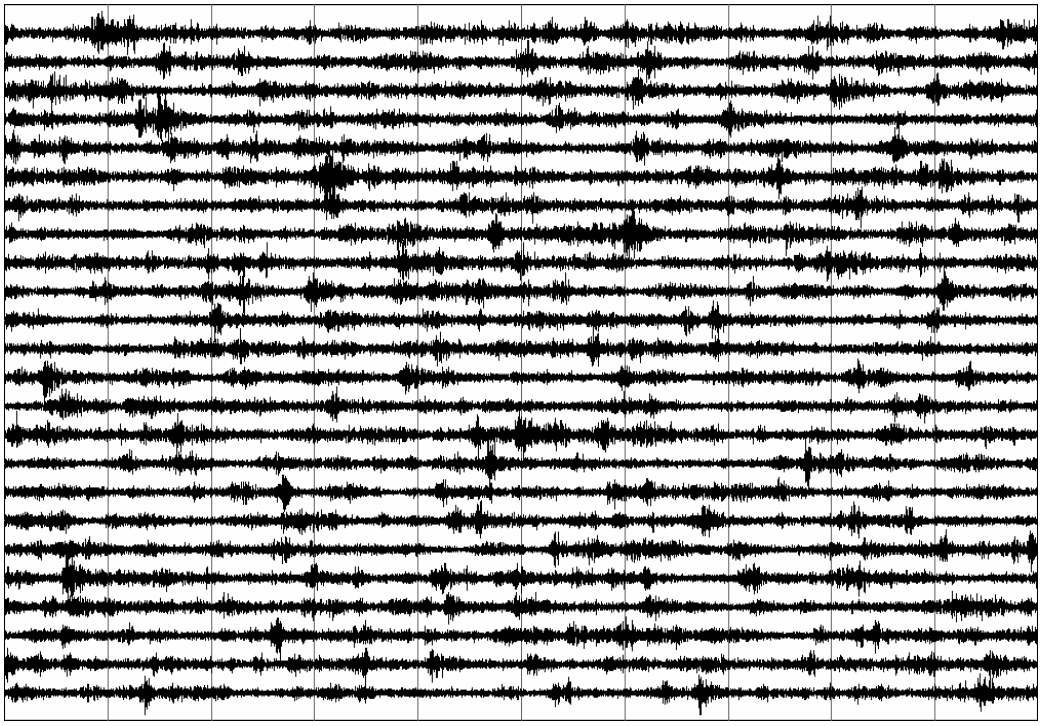


Figura 1.5. *Ejemplo de señal de tipo trémor volcánico registrado en el volcán Etna (Italia). El registro tiene una duración de cuatro horas (de Ibáñez y Carmona, 2000).*

1.1.1.5. Explosiones

Las explosiones son, junto al trémor, las señales más características de un sistema volcánico cuando se encuentra en proceso eruptivo. Por regla general las explosiones tienen en los registros sísmicos al menos dos llegadas diferentes y claras. La primera está asociada con la propagación en forma de ondas internas o superficiales de la explosión. La segunda es la llegada de lo que se conoce como ondas de aire, ondas de choque u ondas sonoras, cuya velocidad de propagación coincide claramente con la del sonido en el aire (340 m/s). Esta velocidad de propagación tan lenta y clara es la forma más fácil de identificar este tipo de eventos sobre los sismogramas cuando no ha sido posible distinguirlos en el momento de su ocurrencia.

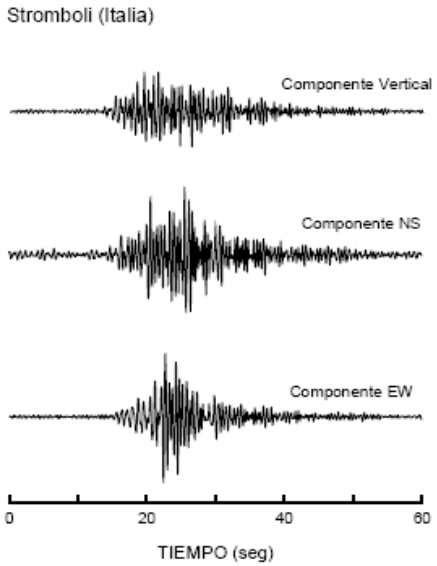


Figura 1.6. *Registro de tres componentes correspondiente a una explosión del volcán Strómboli, también en Italia (de Ibáñez y Carmona, 2000).*

De esta breve introducción a las señales sísmicas puede extraerse alguna conclusión interesante desde el punto de vista del desarrollo de instrumentación sísmica. Entre las características de las señales que condicionan el diseño destaca, como se ha dicho antes, su elevado rango dinámico. En sistemas digitales el principal factor que determina el margen dinámico de un instrumento es la resolución del conversor A/D, que por tanto deberá ser lo mayor posible. Como se verá en este mismo capítulo, existen en la actualidad conversores A/D de alta resolución que ofrecen buenas características a un precio razonable. Sin embargo, la frecuencia de muestreo de estos dispositivos no puede ser demasiado alta por razones que se comentarán más adelante, por lo que es imposible conseguir sistemas que cubran toda la banda de frecuencias de las señales sísmicas recogidas en la tabla 1.1. Las frecuencias más altas quedarán fuera del rango de señales registrables debido a la limitación en la frecuencia de muestreo. Las frecuencias más bajas que el sistema podrá registrar dependerán, sobre todo, del tipo de sensores que se utilicen, que a su vez estará condicionado por diversos factores como el precio, la portabilidad del sistema o el objeto de estudio al que se va a aplicar el instrumento.

1.2. Antenas sísmicas

1.2.1. Definición

Una antena sísmica[1] o *array* sísmico es un dispositivo compuesto por:

a) un grupo numeroso de sensores sísmicos de respuesta conocida dispuestos en un espacio reducido y homogéneo de terreno según una configuración espacial determinada.
b) un sistema de registro sincronizado en el tiempo.

En la sección 1.2.3 se analizará con más detenimiento esta definición. Antes, sin embargo, es necesario introducir el fundamento en el que se basan las técnicas de procesado de datos para este tipo de dispositivos, que constituye uno de los pilares básicos en los que reside su potencia.

1.2.2. Antenas frente a redes sísmicas convencionales

Desde el punto de vista del análisis, lo que define una antena es el procesado colectivo de las formas de onda de todo el conjunto de sensores. Tradicionalmente el registro de señales sísmicas se ha realizado mediante redes sísmicas de varias estaciones, normalmente con enlaces telemétricos, distribuidas en torno a la zona sismogenética. Los métodos de análisis de los registros obtenidos en una red sísmica de este tipo se basan en identificar las distintas fases de los eventos y determinar las diferencias de tiempo entre ellas. Partiendo de estas diferencias se procede a la localización del foco. La máxima precisión en la localización, con los algoritmos tradicionalmente usados basados en técnicas de mínimos cuadrados, se obtiene si el evento se produce dentro de la red.

Las antenas sísmicas se presentan como una alternativa interesante a las redes sísmicas convencionales. En este caso el estudio de los registros no se realiza individualmente, estación por estación, sino simultáneamente y utilizando técnicas de análisis de señales. Al estar las estaciones muy próximas entre sí, las formas de onda registradas en cada una de ellas procedentes de un evento lejano (comparado con las dimensiones de la antena) serán muy similares, excepto por el retraso temporal y el ruido. Las señales procedentes del evento estarán correlacionadas, mientras que el ruido - tanto el del suelo, generado localmente, como el instrumental - no lo estará. Esta característica permite aumentar drásticamente la relación señal-ruido (SNR, *Signal to Noise Ratio*), por ejemplo sumando los registros con los retardos apropiados de manera que la señal procedente del evento quede en fase en todos los canales (figura 1.7). Además del aumento de la SNR las técnicas de *array* permiten determinar la posición de la fuente de señales que no son terremotos propiamente dichos y que no presentan fases definidas, como las asociadas a la dinámica de fluidos presentes en áreas volcánicas que se han presentado al principio del capítulo.

Desde el punto de vista logístico, las ventajas de las antenas frente a las redes sísmicas son notables. En primer lugar, las reducidas dimensiones de estos dispositivos en relación

[1] La denominación más usual en castellano es la de antenas sísmicas, pero se usa también frecuentemente el vocablo inglés *array* sísmico. En este trabajo se usarán los dos términos indistintamente. Otros nombres por los que se conocen estos dispositivos son agregados sísmicos, arreglos sísmicos (término más utilizado en Sudamérica) o, simplemente, dispositivos sísmicos, si bien este último puede llevar a confusión por ser más genérico.

con las de una red sísmica[2] hacen que las antenas sean más fáciles de instalar, lo cual implica una evidente ventaja en el caso de instrumentos concebidos para la intervención rápida en zonas de crisis sísmicas. El mantenimiento también se simplifica mucho frente al de las redes sísmicas convencionales, y al estar toda la instrumentación concentrada en una zona limitada resulta más fácil de vigilar y, en consecuencia, ofrece una mayor seguridad frente al vandalismo.

En el aspecto técnico la reducida separación entre los sensores hace que se puedan utilizar enlaces de alta velocidad, generalmente por cable, lo cual permite mayores resoluciones y frecuencias de muestreo en los sistemas de conversión analógico/digital. Por otra parte, la transmisión es más fiable, ya que se producen menos interferencias y contaminación de las señales. Todo ello redunda en una mayor calidad de los registros.

El resultado que se obtiene al aplicar las técnicas de localización de las antenas sísmicas consta de dos valores: la velocidad aparente con que las ondas atraviesan la superficie de las antenas y el azimut epicentro-antena de la fuente sísmica (figura 1.8). Si las señales registradas presentan fases P y S, con sus lecturas y el trazado del rayo es posible determinar la posición del foco de los eventos. Para las señales en las que no existen fases reconocibles, como las que se presentan habitualmente en áreas volcánicas, con una sola antena sólo se obtiene información acerca de la dirección de la fuente y de la propagación de las ondas, pero no es posible estimar la distancia. Sin embargo, sí se puede determinar la posición de la zona epicentral utilizando dos antenas dispuestas adecuadamente en torno al foco y operando de forma combinada (figura 1.9). Esta determinación de la posición de la fuente de señales sin fases definidas no es posible realizarla utilizando una red sísmica convencional.

No obstante, las redes sísmicas presentan otras características interesantes para su aplicación en entornos volcánicos. En primer lugar, si las estaciones de la red se disponen en torno al volcán, permiten una localización más precisa de los terremotos asociados al proceso volcánico, así como la determinación del mecanismo de fuente de un modo más simple que utilizando *arrays*. Al igual que con las antenas, es posible realizar estudios de polarización de las ondas, siempre que se utilicen estaciones de tres componentes. Y, sobre todo, permiten caracterizar mejor que las antenas si el contenido espectral de la señal está asociado a procesos de fuente o de camino.

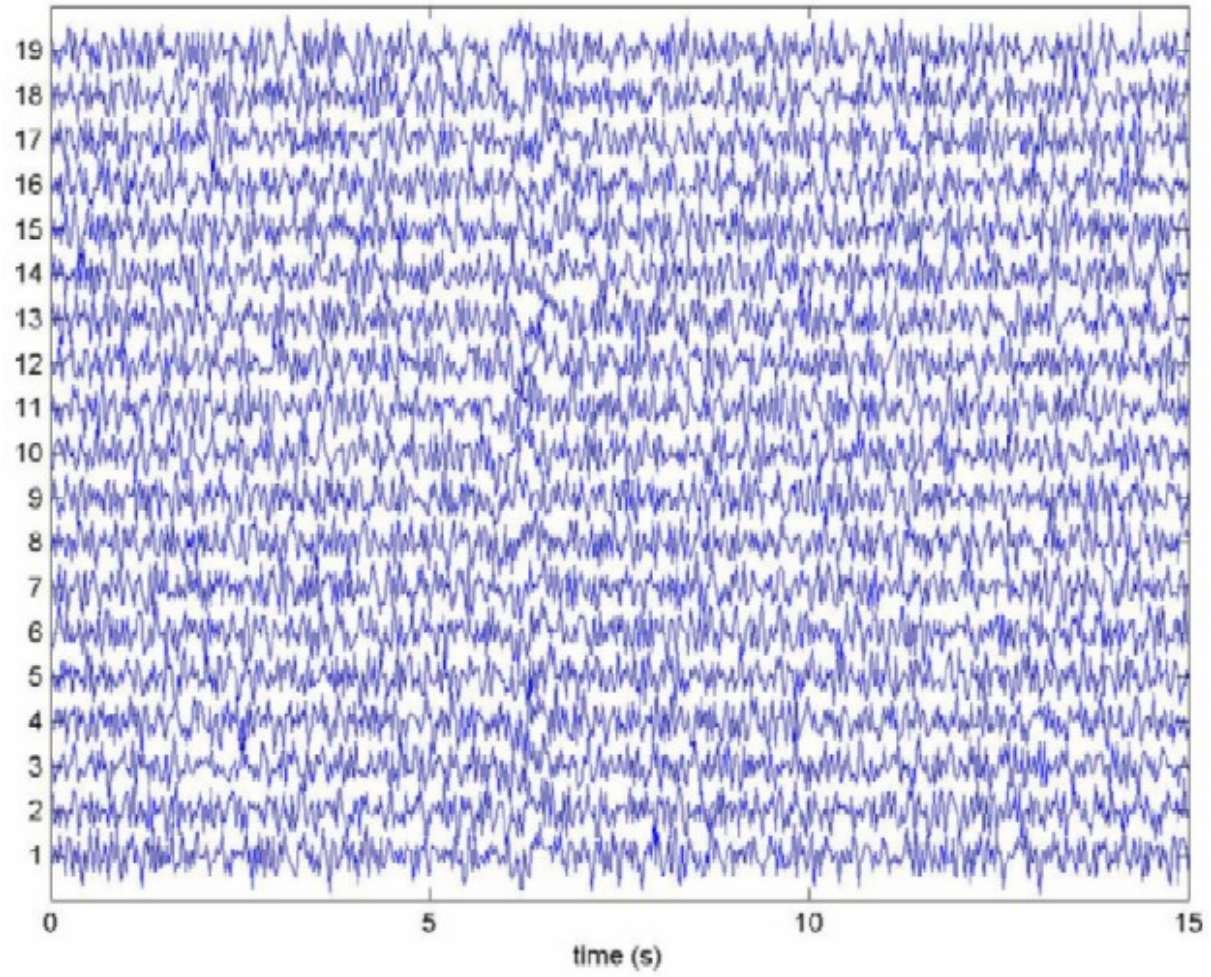
De todo lo dicho se concluye que la configuración idónea para el estudio de la actividad en un área volcánica sería una combinación de una red sísmica dispuesta en torno a la zona activa más dos o más antenas sísmicas (Abril e Ibáñez, 2000). Sin embargo, teniendo en cuenta las dificultades que suele entrañar el despliegue y mantenimiento de una red en este tipo de entornos debido a la orografía, vegetación y condiciones ambientales, en la práctica la solución más adecuada puede consistir en una o dos antenas operando en modo combinado.

Las técnicas de localización de *array* están basadas en la búsqueda de la máxima coherencia de la señal registrada en las estaciones sísmicas de la antena. Esta búsqueda se puede llevar a cabo mediante diversos métodos, tanto en el dominio del tiempo como en el de la frecuencia. En el dominio del tiempo una de las técnicas más comunes es la de correlaciones cruzadas o ZLCC (*Zero Lag Cross Correlation*, ver por ejemplo Frankel *et al.*, 1991, o Del Pezzo *et al.*, 1997). En el dominio de la frecuencia destacan la técnica de clasificación múltiple de señales o MUSIC (Schmidt, 1986; Goldstein y Archuleta, 1987 y 1991), la de producción del haz o *Beam Forming* (LaCoss *et al.*, 1969) o la de análisis de alta resolución del espectro frecuencia-número de onda (Capon, 1969). Los fundamentos de cada uno de estos métodos de análisis quedan fuera del ámbito de esta tesis, por lo que no se van a discutir aquí. Puede consultarse una descripción detallada de todos ellos, así como una

[2] Como se verá más adelante, existen antenas de gran apertura e incluso las propias redes sísmicas locales o regionales pueden utilizarse como antenas para telesismos. Aquí nos referimos a antenas de pequeña o mediana apertura.

discusión sobre las ventajas e inconvenientes que presenta cada uno en el ámbito del análisis de señales sismo volcánicas, en Almendros, 1999.

a)



b)

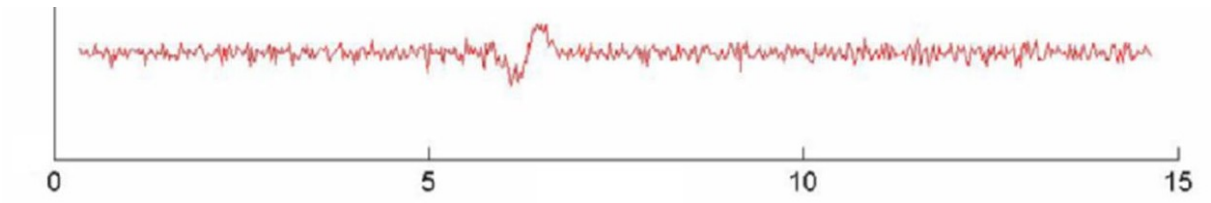


Figura 1.7. *Las antenas sísmicas permiten obtener un sensible aumento en la relación señal-ruido de los registros. La figura muestra un ejemplo de señal sintética, consistente en un pulso al que se le ha superpuesto ruido aleatorio. Las trazas individuales de las estaciones (a) se han obtenido desplazando el pulso con distintos intervalos para simular los registros reales que se obtendrían en una antena. Como puede verse, debido a la baja amplitud del pulso éste queda totalmente enmascarado por el ruido. Sin embargo, si las señales se suman con el retardo adecuado el ruido se cancela, mientras que la señal coherente se suma, lo cual permite discriminar perfectamente el pulso original (b) (Almendros, comunicación personal).*

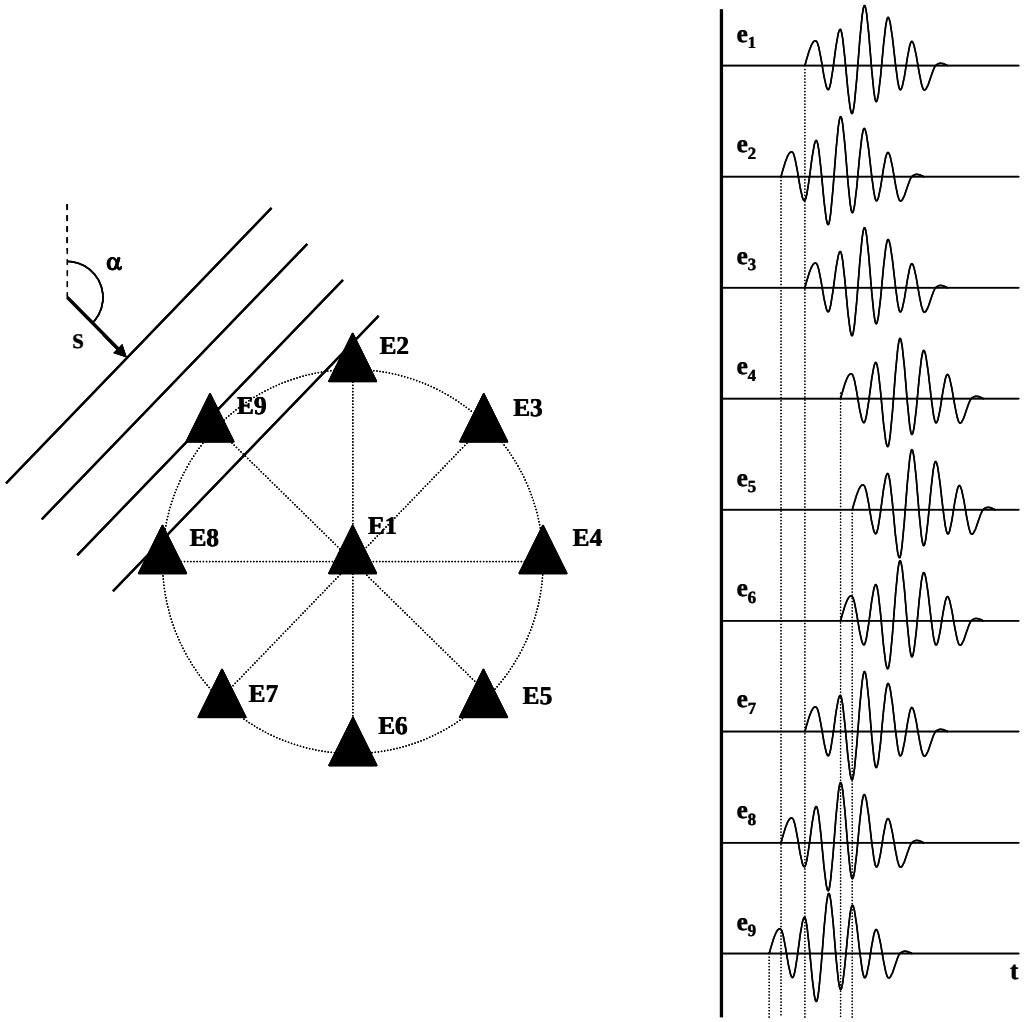


Figura 1.8. *Un frente de onda plano incide sobre una antena circular con nueve estaciones, produciendo registros casi idénticos, salvo por el retardo relativo. Los registros reales presentarán también pequeñas diferencias debidas al ruido no coherente y a heterogeneidades locales. A partir de los retardos entre las trazas las técnicas de análisis suministran normalmente estimaciones de dos parámetros en la solución: el azimut* α *(que por convenio se mide en sentido horario tomando como origen el norte) y la velocidad aparente. En la práctica se trabaja con la lentitud aparente s (de* slowness*), definida como el inverso de la velocidad aparente. La determinación de estos dos parámetros equivale a obtener el vector* $\mathbf{s}$ *que se muestra en la figura. Otros parámetros habitualmente utilizados en los análisis son el vector número de onda* $\mathbf{k}$*, relacionado con la lentitud y con la frecuencia según la expresión* $\mathbf{s} = \mathbf{k}/2\pi f$*, y el parámetro del rayo, definido como el módulo del vector* $\mathbf{s}$*.*

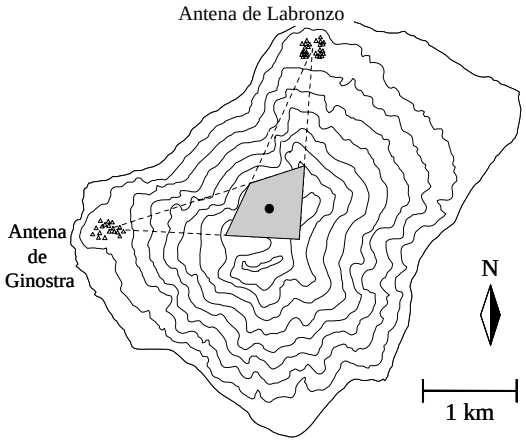


Figura 1.9. *Ejemplo de localización de un evento explosivo y determinación del error mediante el uso combinado de dos antenas sísmicas en la isla de Strómboli (de Almendros, 1999).*

1.2.3. Definición revisada

Una vez presentados los conceptos básicos asociados al análisis de datos mediante técnicas de *array*, podemos intentar acotar la definición presentada antes. Como decíamos, una antena o *array* sísmico es un dispositivo compuesto por…

... un grupo numeroso…

Pero, ¿qué se entiende por 'un grupo numeroso'? En el contexto de la definición propuesta de antena sísmica, 'numeroso' implica una cantidad suficiente de sensores sísmicos como para poder aplicar las técnicas de análisis de datos y obtener información útil. Estrictamente hablando, el número mínimo de sensores sería tres, que dispuestos adecuadamente proporcionarían los retardos en la llegada de la señal necesarios para calcular el azimut y velocidad aparente. Sin embargo, tal configuración cubriría un rango muy corto de frecuencias y presentaría un gran *aliasing* espacial, como se verá a continuación. Por otra parte, en caso de malfuncionamiento o registro de mala calidad de una estación el sistema quedaría inutilizado.

En la práctica se usa un número mucho mayor de sensores. Originalmente, en las grandes antenas fijas como LASA (Green *et al.*, 1965) o NORSAR (Kedrov y Ovtchinnikov, 1990) se utilizaron cientos de ellos, dispuestos en varios subarrays operando coordinadamente. En la actualidad tienden a usarse menos sensores, sobre todo en antenas portátiles concebidas para la intervención rápida en áreas de crisis volcánicas o tectónicas, en las que la facilidad de transporte y despliegue son condiciones necesarias. Es normal utilizar sistemas modulares con un número medio de estaciones, entre ocho y veinte. En caso de necesitar más sensores se pueden usar varios de estos sistemas operando sincronizadamente (ver, por ejemplo, Almendros *et al.*, 1997; Ibañez *et al.*, 1997; Del Pezzo *et al.*, 1998).

…de sensores sísmicos de respuesta conocida…

¿Qué son 'sensores de respuesta conocida'? En principio, cualquier tipo de instrumento sísmico preparado para operar en campo puede utilizarse en una antena y en la práctica se usan tanto sensores de banda ancha como sismómetros de corto periodo y acelerómetros. Sin embargo, como se ha visto antes, las técnicas de procesado se basan en que las señales de interés son casi idénticas en todas las estaciones, salvo por el retardo y el ruido no coherente.

Esto hace fundamental conocer la respuesta de todos los sensores, para poder corregirla y normalizar todas las formas de onda antes del procesado. En la práctica en una antena se suelen usar como mucho dos tipos de sensores distintos, aunque lo más habitual es que todos sean del mismo tipo y con respuesta lo más parecida posible, lo cual permite trabajar con los datos crudos, sin necesidad de corregir la respuesta instrumental.

… dispuestos en un espacio reducido y homogéneo de terreno…

¿Qué es 'un espacio reducido y homogéneo'? Según se ha visto, las técnicas de procesado de datos de antenas sísmicas se basan en que la señal procedente de todos los sensores sísmicos es coherente, mientras que el ruido no lo es. A efectos prácticos, puede decirse que la disposición de los sensores debe ser tal que la señal de todos ellos sea lo más parecida posible. En el caso ideal todas las señales serían idénticas y en el registro se diferenciarían tan solo en los tiempos de llegada. La coherencia de las señales exige que el terreno en el que se disponen los sensores no presente heterogeneidades que producirían diferencias en las formas de onda. En la práctica, la forma más sencilla de asegurar que no existen heterogeneidades es disponer los sensores en un espacio lo suficientemente reducido como para que corresponda a la misma estructura geológica.

Ahora bien, los términos 'reducido' y 'homogéneo' están relacionados con las frecuencias de la señales. Así, una red sísmica local con un diámetro del orden de decenas de kilómetros podría usarse como *array* para telesismos, ya que la forma de onda de los registros sería muy parecida en todas las estaciones. No sería posible darle a la red la misma aplicación para sismos locales o regionales, en los que las frecuencias dominantes son mayores y por tanto también lo es la influencia de las heterogeneidades en la forma de onda registrada.

Por tanto, las dimensiones de la antena están condicionadas por el rango espectral de las señales que se estudian. Si la separación entre estaciones es demasiado grande, aparecerá el fenómeno de *aliasing* espacial, análogo al *aliasing* que se presenta en el dominio del tiempo si la frecuencia de muestreo es demasiado baja. Para evitar este efecto de pérdida de información, la separación entre estaciones debe ser siempre menor que la mitad de la longitud de onda de la señal que se registra. El contenido espectral de las señales también impone un límite inferior al diámetro o apertura total de la antena, que debe ser tan grande al menos como la longitud de onda más larga que nos interese (Asten y Henstridge, 1984). Es decir, debe cumplirse que:

$$2d < \lambda < D \qquad [1.1]$$

siendo d la separación media entre estaciones y D la apertura total de la antena.

Así, para registrar señales en la banda de frecuencia 1-10 Hz en un medio con velocidades aparentes esperadas del orden de 2 km/s las longitudes de onda máximas y mínimas serían:

$$\lambda = v/f \quad \Rightarrow \quad \lambda_{min} = v/f_{max} = 200\ m;\ \lambda_{max} = v/f_{min} = 2000\ m; \qquad [1.2]$$

De modo que la separación entre estaciones debería ser de unos 100 m y la apertura total de, al menos, 2000 m.

Dado que la apertura impuesta por el rango de frecuencias de las señales es un límite inferior, podría caerse en la tentación de adoptar una configuración arbitrariamente grande para aumentar el rango de señales que se registran. Sin embargo, además del aumento en el número de estaciones que un diseño así requeriría, no produciría ningún beneficio en el estudio de las frecuencias mayores, ya que la coherencia de las señales (y por tanto la

similitud entre las formas de onda) suele perderse con espaciados mayores de dos o tres longitudes de onda.

…según una configuración espacial determinada…

¿Qué es 'una configuración espacial determinada'? La definición propuesta de antena sísmica no recomienda una configuración espacial como la más adecuada, ya que ésta depende, entre otros factores, del objetivo que se persiga. Así, si la fuente sísmica puede a priori presentar cualquier azimut, los sensores deberían estar dispuestos según un patrón omnidireccional, como una circunferencia o, si se cuenta con un número suficiente de sensores, varias circunferencias concéntricas. En este último caso es conveniente que las estaciones de circunferencias consecutivas no estén alineadas desde el centro geométrico, para mejorar la cobertura en azimut. Por otra parte el espaciado radial no debe ser uniforme, sino menor entre las circunferencias internas, para ayudar a evitar el *aliasing* espacial.

En muchas ocasiones, especialmente en entornos volcánicos, las fuentes sísmicas objeto de estudio están limitadas a una pequeña región (por ejemplo, el área cratérica en un volcán en erupción). En estos casos se adoptan para los sensores distribuciones espaciales orientadas a optimizar el comportamiento de la antena para las ondas procedentes de esa región, como pueden ser configuraciones lineales, en L o, si se cuenta con mayor número de sensores, semicirculares de diversos tipos (Ferrazzini *et al.*, 1991).

En cualquier caso, a veces es recomendable que la distribución espacial de los sensores no obedezca a configuraciones geométricas perfectas, sino que se mejora la resolución espacial utilizando distribuciones semi aleatorias basadas en modelos geométricos (Abril e Ibáñez, 2000).

La resolución espacial de una antena puede estimarse mediante su función de transferencia, también conocida como patrón de radiación (ver, por ejemplo, Havskov y Alguacil, 2004, pp. 266-269). La figura 1.10 muestra las funciones de transferencia de varias configuraciones espaciales de sensores. La respuesta ideal de una antena presentaría un pico en el origen del plano $\mathbf{k_t}$ **-** $\mathbf{k}$, y ningún otro pico significativo en el resto del plano, lo cual significaría que la potencia del rayo sería máxima para el verdadero valor de k (es decir, $\mathbf{k} = \mathbf{k_t}$) y nula para el resto de valores. Cuanto más acusado sea el pico central, más precisa es la determinación del valor correcto de la lentitud. La aparición en el diagrama de picos distintos del central representa cierto grado de *aliasing* espacial, que puede dar como resultado una estimación ambigua del valor de $\mathbf{k_t}$ (o, equivalentemente, de la lentitud $\mathbf{p_t}$).

… y un sistema de registro sincronizado en el tiempo.

Por último, ¿qué es 'un sistema de registro sincronizado en el tiempo'? Uno de los problemas más complejos en el diseño de estaciones sísmicas ha sido siempre el sistema de tiempo. En una red sísmica convencional no sólo es necesario registrar el movimiento del suelo en distintos puntos, sino también conocer en qué momento se produce ese movimiento para poder relacionar las señales entre sí. Por las especiales características de las antenas sísmicas y de las técnicas de procesado de datos asociadas a ellas, el sistema de temporización adquiere en este tipo de dispositivos una importancia mucho mayor. La reducida separación entre los sensores y el procesado conjunto de todas las formas de onda hace que cualquier indeterminación en el tiempo de registro produzca resultados totalmente distintos de los esperados, como se muestra en la figura 1.11. La sincronización relativa entre las diferentes estaciones de una antena debe ser al menos un orden de magnitud mejor que para estaciones de una red sísmica. Más adelante en este mismo capítulo se darán las pautas para el diseño del sistema de sincronismo en dispositivos de antena.

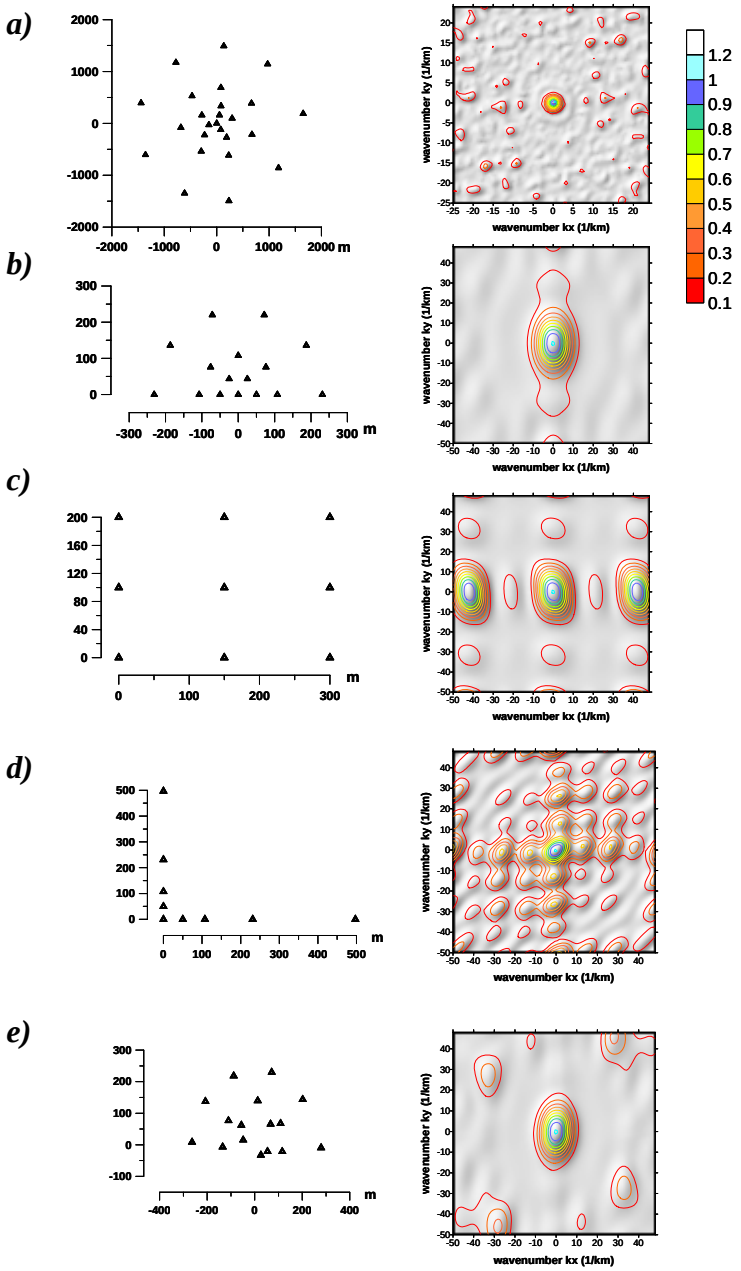


Figura 1.10. *Distintos ejemplos de configuraciones espaciales de sensores en antenas de pequeña apertura y sus funciones de transferencia. Éstas se representan en el plano* $\mathbf{k_t}$-$\mathbf{k}$*, siendo* $\mathbf{k_t}$ *el número de onda verdadero del rayo. Los valores de los ejes pueden transformarse, para una frecuencia dada, a valores de lentitud mediante la expresión* $\mathbf{k} = 2\pi f\mathbf{p}$*. En a) se muestra el* array *ARCES, en Noruega, que consta de 25 estaciones dispuestas según una estructura de circunferencias concéntricas con un diámetro máximo de 3km. Esta antena presenta una respuesta casi ideal en los dos ejes. En b), una antena semicircular en la que aparece el fenómeno de* aliasing *espacial en el eje* ky*, razón por la cual en aplicaciones reales se suele orientar el eje mayor de este tipo de dispositivos hacia la fuente esperada de señal. En c) se muestra una antena rectangular equiespaciada, cuya respuesta presenta un fuerte* aliasing *espacial en los dos ejes, representado en el diagrama por picos fantasma. En d) puede verse una antena biaxial, que presenta una respuesta bastante escarpada pero con un* aliasing *acusado. Esta configuración se utiliza cuando se dispone de pocos sensores y la posición esperada de la fuente es conocida. Por último, en e) se muestra una disposición semi-aleatoria de los sensores, basada en la configuración semicircular de b), que presenta una buena respuesta y una mejora de la resolución a lo largo del eje* ky *respecto a la de b) (de Havskov y Alguacil, 2004).*

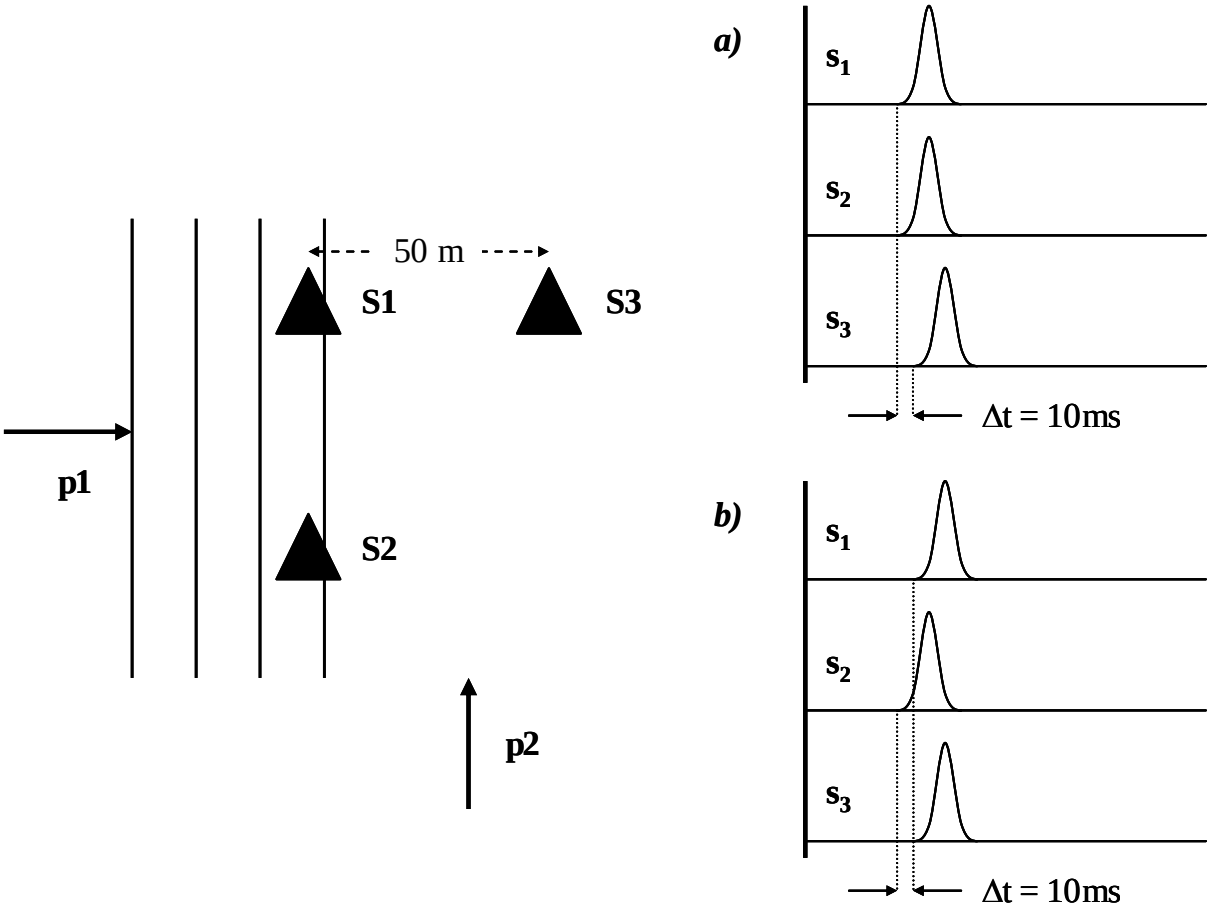


Figura 1.11. *La importancia de conseguir una sincronización precisa en el proceso de muestreo queda patente en esta figura. Entre las estaciones S1 y S3 hay una distancia de 50 m, igual que entre S1 y S2. Suponiendo que el proceso de muestreo está correctamente sincronizado y que la frecuencia de muestreo es de 100 mps, un pulso que llegara a la antena con velocidad aparente de 5km/s y azimut de 90º (*$\mathbf{p_1}$*) se registraría como se muestra en a). El pulso tarda 10 ms en recorrer la distancia que separa S1 de S3, lo cual equivale a un periodo de muestreo. Por tanto, una indeterminación de tan solo una muestra en la estación S1 podría hacer que el registro fuera el de la figura b), con lo que el sistema interpretaría que el pulso llega antes a S2 que a S1 y S3 y le asignaría el vector* $\mathbf{p_2}$ *con azimut 0º.*

1.2.4. Origen y aplicaciones

Aunque las ventajas de utilizar un gran número de sensores sísmicos en un espacio reducido de terreno ya habían sido aprovechadas antes en técnicas de prospección sísmica (Klipsch, 1936, McDermott, 1937), su aplicación a la sismología data de finales de los años cincuenta y se debe a un motivo fundamentalmente político. En 1958, en plena guerra fría, un grupo de expertos recomendó llevar a cabo una renovación de las estaciones sísmicas de todo el mundo para aumentar sus prestaciones, con el objetivo de detectar violaciones del acuerdo de suspensión de pruebas nucleares. A partir de entonces los gobiernos, sobre todo los del Reino Unido y EEUU, aumentaron los presupuestos destinados a la investigación y desarrollo de dispositivos de *array*. En los años siguientes se instalaron cinco antenas en EEUU y otras en

Escocia, Canadá, Australia, India y Brasil. En España se instaló con el mismo fin el *array* de Sonseca (Toledo) (Martínez Solares, 1995).

La detección y localización de explosiones nucleares no representaba una tarea fácil. Por una parte porque, como es lógico, no era posible instalar los dispositivos en suelo enemigo, por lo que las fuentes eran lejanas y, como consecuencia, las señales débiles y enmascaradas en el ruido de fondo. Por otra porque muchas veces las señales estaban superpuestas a terremotos reales, ya que las grandes potencias, conscientes de la vigilancia a que estaban sometidas por el enemigo, solían preparar los ensayos en zonas sísmicas y llevarlos a cabo cuando se producía un sismo de magnitud considerable. De esta forma dificultaban la detección y localización de las pruebas (Evernden, 1976). En la actualidad el organismo encargado de controlar el cumplimiento del tratado de prohibición de ensayos nucleares (CTBT, *Comprehensive Nuclear-Test-Ban Treaty*, *http://pws.ctbto.org*) se sirve de una red global de monitorización conocida como IMS (*International Monitoring System*) que está considerada como la mayor y más avanzada del mundo en tiempo real y que está constituida principalmente por *array*s sísmicos.

Aprovechando la experiencia adquirida en la vigilancia para el cumplimiento de los tratados nucleares, en los años sesenta se empezaron a diseñar antenas que podían utilizarse en aplicaciones más generales. Los primeros dispositivos eran fijos y de gran apertura, como LASA (*Large Aperture Seismic Array*), construido en Montana (EEUU) entre 1964 y 1965, (Green *et al.*, 1965) o NORSAR (*Norwegian Seismic Array*), instalado en el sur de Noruega en 1971 (Kedrov y Ovtchinnikov, 1990). Además de seguir cumpliendo la misión para la que fueron originalmente concebidas las antenas sísmicas, las de gran apertura han proporcionado importantes avances en el conocimiento de la estructura interna de la Tierra. La capacidad de realzar la parte coherente de los registros frente al ruido permite extraer de los sismogramas fases muy débiles causadas por discontinuidades de las capas internas (Song y Richards, 1996, Vidale *et al.*, 2000, Poupinet y Kennett, 2004). Las técnicas de *array* aplicadas a los registros de antenas de gran apertura permiten no sólo obtener información de fases conocidas pero difíciles de detectar, sino que incluso se llegan a detectar fases no observadas antes, como la PKJKP (Julian *et al.*, 1972), que demostró el carácter rígido del núcleo interno.

Más recientemente se han utilizado antenas con aperturas menores para reducir los efectos de inhomogeneidades laterales (Mikkeltveit, 1985). En la actualidad, la posibilidad de contar con sensores cada vez más pequeños, sistemas de adquisición de datos de mejores prestaciones y ordenadores más potentes y de menor consumo ha hecho que las aplicaciones de las antenas sísmicas no estén limitadas a la monitorización de la sismicidad a nivel global, sino que constituyen una poderosa herramienta para el estudio de la sismicidad regional y local. Como ejemplo puede consultarse Ibáñez *et al.*, 1997, en el que los autores se sirvieron de una antena de pequeña apertura para registrar la actividad sísmica entre la península antártica y las islas Shetland del Sur, y demostrar la actividad asociada a la zona de subducción. En este caso la instalación y mantenimiento de una red sísmica convencional no resultaba viable debido a las condiciones ambientales y logísticas.

Según se vio antes, dos de las principales ventajas que ofrecen las antenas sísmicas frente a las redes sísmicas tradicionales son un aumento de la SNR y la capacidad para tratar con señales sin fases definidas. La primera permite detectar eventos distantes que de otra forma quedarían enmascarados por el ruido de fondo (Frankel, 1994). La segunda tiene particular interés en el estudio de áreas volcánicas, en las que además las redes sísmicas convencionales normalmente son complicadas de instalar y de mantener. Estos estudios, habitualmente realizados utilizando antenas sísmicas portátiles de pequeña apertura que se despliegan durante campañas de recogida de datos para su posterior interpretación, han permitido profundizar en áreas como la naturaleza de las fuentes sismo-volcánicas, modelos de fracturas o estructuras volcánicas (ver, por ejemplo, Chouet *et al.*, 1998; Del Pezzo *et al.*, 1997). Las técnicas de procesado de datos permiten la localización de señales sin llegadas

claras o el seguimiento temporal de fuentes de señales casi continuas como el trémor volcánico (Almendros *et al.*, 1997).

Un ejemplo de la versatilidad de las antenas sísmicas en zonas volcánicas lo constituye el volcán Kilauea (Hawai), en el que diversos experimentos con distintas antenas y configuraciones han servido para estudiar, entre otras cosas, la estructura interna del sistema volcánico, la composición del trémor y la localización y características de las fuentes de señales de eventos de largo periodo y terremotos, utilizando una o varias antenas operando sincronizadamente. Además estos estudios se han llevado a cabo en distintas bandas de frecuencias, desde corto periodo hasta ultra largo periodo (Ferrazzini *et al.*, 1991; Ferrazzini y Aki, 1992; Goldstein y Chouet, 1994). Otros volcanes que han sido estudiados de manera exhaustiva utilizando antenas sísmicas son el Strómboli (Chouet *et al.*, 1997; Del Pezzo *et al.*, 1998; Saccorotti *et al.*, 1998), Etna (Del Pezzo *et al.*, 2000) y Vesubio (Del Pezzo *et al.*, 1999, Saccorotti *et al.*, 2001b), todos en Italia. Teide en España (Del Pezzo *et al.*, 1997; Almendros *et al.*, 2000), Masaya en Nicaragua (Metaxian *et al.*, 1997), o la isla volcánica de Decepción en la Antártida (Ibáñez *et al.*, 1997; Almendros *et al.*, 1997; Ibáñez *et al.*, 2000). En todos estos volcanes se han utilizado diversas configuraciones y número de antenas sísmicas con el fin de estudiar la estructura volcánica y las características de la fuente sismo-volcánica.

Además se han utilizado antenas sísmicas para realizar, entre otros, estudios sobre propagación de ondas cerca de la fuente sísmica (Gupta *et al.*, 1990; Iida *et al.*, 1990; Niazi y Bozorgnia, 1991), efectos de sitio y propagación de ondas cerca del receptor (Dainty y Toksoz, 1990; Al-Shukri *et al.*, 1995; Aoi *et al.*, 1997), localización de heterogeneidades mediante el estudio de las ondas coda (Spudich y Botswick, 1987; Wagner y Langston, 1992, Del Pezzo *et al.*, 1997), determinación de la estructura superficial mediante el estudio de microtemblores (Kagawa *et al.*, 1996) o caracterización de la polarización de las ondas (Jurkevics, 1988).

Rost y Thomas, 2002, ofrecen una excelente introducción a los estudios mediante técnicas de antena sísmica. También puede consultarse Almendros, 1999, en la que se presta especial atención a las aplicaciones en entornos volcánicos.

1.2.5. Introducción al diseño de antenas

La figura 1.12 muestra un diagrama de bloques funcional de una antena sísmica. En realidad también podría ser el de una estación perteneciente a una red sísmica o el de un sistema de registro concebido para operar independientemente, ya que los procesos que debe controlar son básicamente los mismos: sensado del movimiento del suelo, acondicionamiento de la señal recogida, conversión analógico/digital (A/D), preprocesado y grabación de los datos y, en ocasiones, transmisión de los mismos a una estación remota. Además debe gestionar la transmisión de la señal analógica o de los datos digitales[3] entre los puntos de sensado y el sistema central y la temporización de todo el proceso, e incorporar un subsistema de alimentación para todos los módulos.

Aunque el diagrama de bloques funcional presenta pocas diferencias respecto al de una estación de una o tres componentes, éstas son más acusadas en la implementación física de los distintos subsistemas. Por una parte, el elevado número de canales que debe gestionar un dispositivo de antena y la distribución de los sensores a distancias lejanas (comparadas con las de los sistemas de una o tres componentes) hace que cobre una gran importancia la transmisión de señal o datos digitales desde los puntos de sensado hasta el sistema central de control. Por otra parte debe prestarse especial atención al diseño del subsistema de temporización, ya que de su precisión y fiabilidad depende en buena parte la correcta

[3] La transmisión de la señal de los sensores hasta el módulo central se realiza de forma analógica o digital, dependiendo de en qué momento se lleve a cabo el proceso de digitalización.

sincronización de los datos, necesaria para poder aplicar las técnicas de procesado de datos propias de estos dispositivos. La figura 1.13 muestra un diagrama de bloques de una posible implementación de los distintos módulos, en concreto la utilizada en las antenas portátiles de dieciséis bits del Instituto Andaluz de Geofísica. En el capítulo 5 se describirán con más detalle estos sistemas, así como las nuevas antenas portátiles que los sustituirán.

A continuación se describen algunos aspectos que conviene tener en cuenta desde el punto de vista del diseño de dispositivos de antena, relacionándolos con cada uno de los procesos mostrados en la figura 1.12.

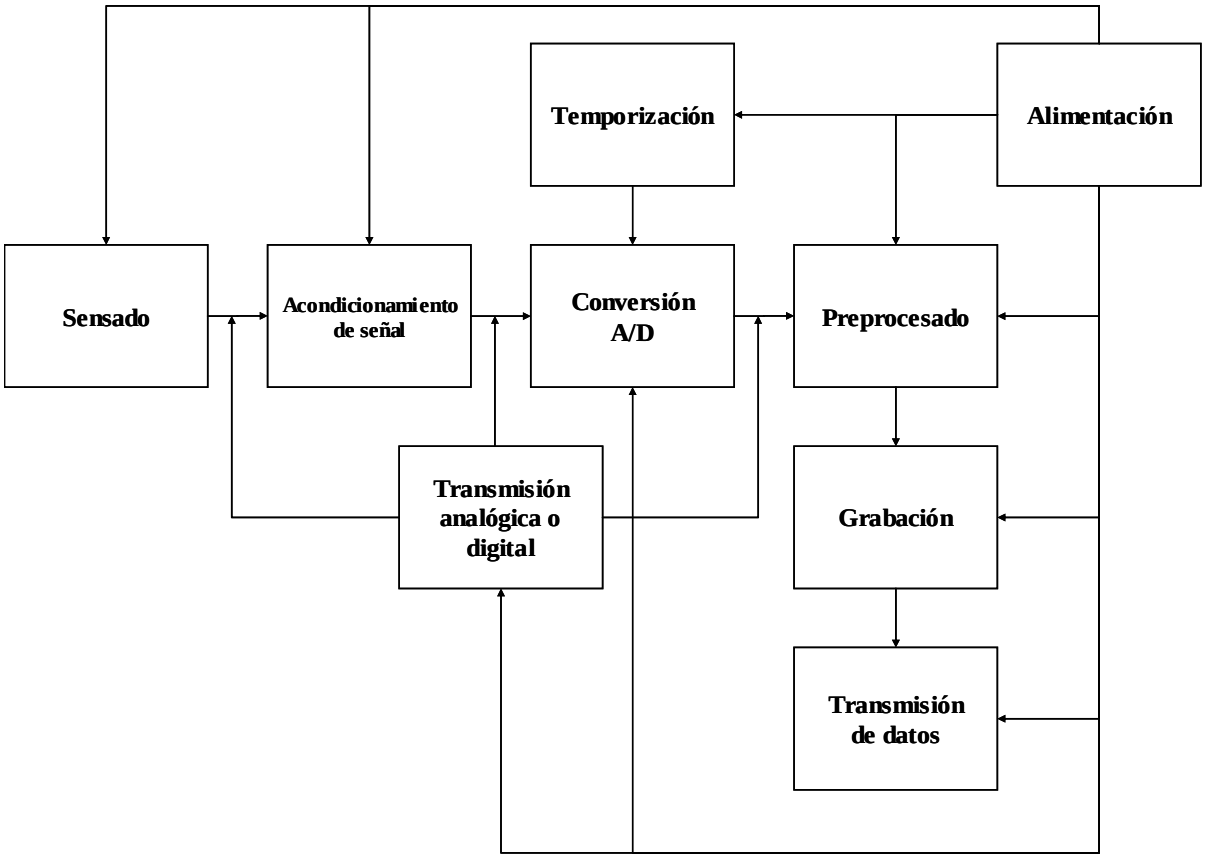


Figura 1.12. *Diagrama de bloques funcional de una antena sísmica (explicaciones en el texto).*

1.2.5.1. Sensado

Como se apuntó anteriormente, prácticamente cualquier instrumento preparado para operar en campo puede adaptarse para trabajar en una antena. En la actualidad se utilizan tanto sensores de velocidad de corto periodo o banda ancha como acelerómetros, si bien el uso de unos u otros está relacionado con las dimensiones del *array* y con el objeto de estudio para el que se ha diseñado. En el contexto que nos ocupa, el de las antenas de pequeña apertura para estudio de la sismicidad volcánica o tectónica local y regional se suelen utilizar sensores de corto periodo de una o tres componentes. En ocasiones se ocupa alguno de los puntos de sensado con sensores de banda ancha, como se verá en la antena del Gran Sasso.

En las antenas portátiles es habitual usar sensores de 4.5 Hz, cuyas principales ventajas son su reducido tamaño, peso y precio. En función de las frecuencias de interés pueden usarse directamente o con ecualizadores para extender su respuesta a 1 s. Sea cual sea la opción elegida, antes de la utilización de los sensores debe hacerse una selección para que

todos presenten una respuesta lo más uniforme posible. En caso de sensores ecualizados conviene llevar a cabo, si se dispone de tiempo, un ajuste individual de la ecualización de todas las unidades.

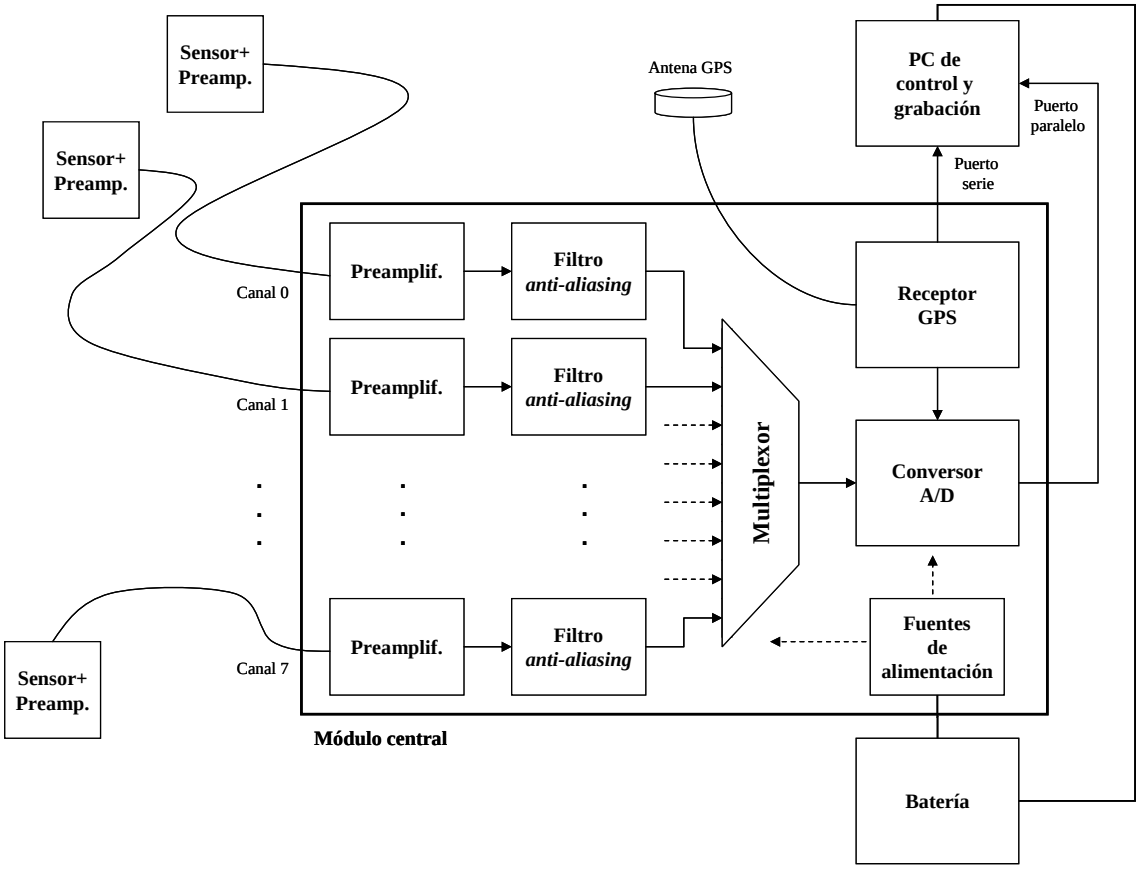


Figura 1.13. *Diagrama de bloques de los arrays de 16 bits. La transmisión de la señal de los sensores es analógica hasta el módulo central. El acondicionamiento de señal consta de preamplificado, filtrado* anti-aliasing *y multiplexado de la señal analógica para conectar sucesivamente cada uno de los ocho canales al conversor A/D. El PC se ocupa del preprocesado (básicamente un algoritmo de detección de eventos STA/LTA y conversión del formato de los datos), grabación de los ficheros en un disco duro local y control de todos los procesos. La temporización se consigue mediante un receptor GPS. Todo el sistema está alimentado con una única batería a través de distintos circuitos de fuente.*

1.2.5.2. Acondicionamiento de señal

En sistemas de conversión A/D se denomina acondicionamiento de señal al conjunto de procesos que se realizan sobre la señal analógica para ponerla en condiciones óptimas para su digitalización. Normalmente el acondicionamiento consta, al menos, de una etapa de filtrado *anti-aliasing*. El teorema de Nyquist-Shannon o del muestreo establece que la frecuencia de muestreo de un sistema de conversión A/D debe ser, al menos, el doble que la máxima frecuencia que se quiere convertir (frecuencia de Nyquist). En caso contrario aparece el fenómeno de *aliasing*, por el cual se produce una pérdida de información de la señal, puesto que al reconstruirla a partir de las muestras digitales adquiridas se obtiene una señal de frecuencia menor. El efecto más perjudicial del *aliasing* no es la pérdida de información de las señales de alta frecuencia, sino que éstas hacen aportaciones falsas de energía a señales con frecuencias menores que la de Nyquist, lo cual produce una distorsión en el espectro de la

señal (figura 1.14). Para evitar este efecto, antes de digitalizar las señales analógicas éstas se filtran paso baja para quedarse con las componentes de frecuencias menores que la de Nyquist. Lo más usual es utilizar para ello filtros paso baja de Butterworth, puesto que presentan una respuesta más plana que otros tipos de filtro estándar en toda su banda de paso. Se suelen realizar con amplificadores operacionales de bajo ruido, y es habitual diseñarlos de octavo orden, ya que este diseño presenta el mejor compromiso entre la pendiente de la respuesta y la facilidad de implementación.

Normalmente la señal de salida de los sensores es demasiado débil como para digitalizarla directamente sin perder rango dinámico en el proceso, por lo que es necesario amplificarla. Lo usual es realizar esta amplificación antes del filtrado *anti-aliasing*, generalmente utilizando circuitos amplificadores de una o varias etapas basadas en operacionales de bajo ruido. La amplificación es especialmente importante cuando la señal de los sensores se transmite de forma analógica hasta un sistema central, ya que de otro modo la atenuación y el ruido inducido en el cable disminuirían la calidad de la misma.

En sistemas de adquisición de datos que utilizan un solo conversor A/D para digitalizar varios canales, después de la amplificación y filtrado *anti-aliasing* de la señal de salida del sensor, es necesario realizar un multiplexado analógico. El multiplexor selecciona sucesivamente y una por una la señal de todos los canales para que el conversor la digitalice. El cambio de canal se produce cuando el multiplexor recibe la señal de que la conversión ha terminado. En este caso, por tanto, el muestreo de la señal de todos los canales no se produce simultáneamente, sino con un retardo distinto para cada canal. En estaciones pertenecientes a redes sísmicas convencionales este retardo no suele ser importante, pero en antenas sísmicas sí, puesto que los datos tienen que estar correlacionados entre sí. Por tanto, en caso de optar por diseños multicanal con un solo conversor A/D, el retardo de conversión de cada canal debe conocerse exactamente y ser corregido cuando se procesan los datos.

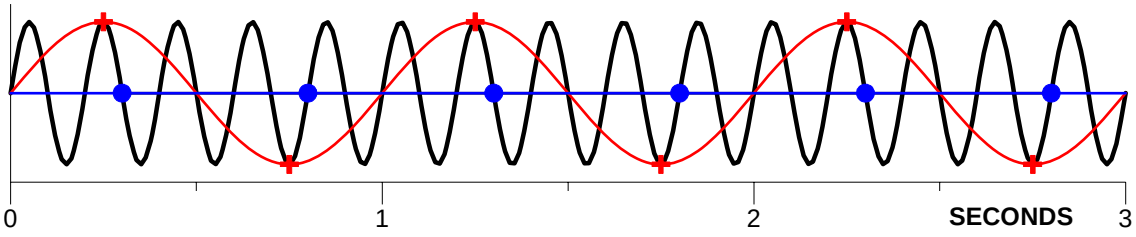


Figura 1.14. *El* aliasing *se produce cuando una señal es muestreada con una frecuencia de muestreo demasiado baja. En la figura una señal de 5Hz se digitaliza con una frecuencia de muestreo de 2Hz. Dependiendo de cuándo se tomen las muestras, la señal de salida puede interpretarse como una línea recta (puntos azules) o como una señal senoidal de 1Hz (línea roja) (de Havskov y Alguacil, 2004).*

1.2.5.3. Transmisión de la señal de los sensores

Los sistemas de *array* tienen que gestionar la adquisición de datos de numerosos sensores situados a distancias considerables, por lo que el proceso de transmisión cobra una gran importancia en este tipo de dispositivos. En antenas de gran apertura o redes sísmicas utilizadas como antenas para sismos lejanos la transmisión suele ser telemétrica, dado que las estaciones habitualmente están separadas por distancias de decenas o centenares de kilómetros. En antenas de pequeña apertura como las que se describen en este trabajo las distancias son mucho menores y la transmisión se realiza prácticamente siempre mediante cables. Existen dos posibilidades: transmitir de forma analógica la señal de los sensores hasta un punto central en el que se digitalizan, o bien digitalizar la señal de cada sensor en el mismo punto de sensado y transmitirla digitalmente hasta el módulo central de control.

Tradicionalmente se ha utilizado la transmisión analógica y digitalización de las señales en el módulo central con un solo conversor A/D, como se muestra en la figura 1.13. Sin embargo, en los últimos años el desarrollo de conversores A/D baratos y de buenas prestaciones, así como el desarrollo de circuitos integrados para diversos estándares de comunicación digital de alta velocidad, ha hecho que cada vez se use más la digitalización local de las señales de los sensores para su posterior transmisión digital. Esta presenta las ventajas, frente a la analógica, de una menor atenuación y una mayor inmunidad al ruido electromagnético, especialmente si se adoptan estándares que utilizan transmisión diferencial mediante líneas balanceadas, como el RS-485 o el RS-422.

En la actualidad es tecnológicamente posible utilizar transmisión digital inalámbrica de bajo consumo en distancias cortas como las que se requieren en una antena de pequeña apertura. Sin embargo, una implementación sin cables implica la necesidad de utilizar alimentaciones independientes en cada punto de recogida de datos. Teniendo en cuenta que para transmitir telemétricamente los datos hacen falta, además del sensor y el propio transmisor, un digitalizador y un pequeño ordenador o controlador digital en cada punto, el consumo aumenta y obliga a utilizar un número elevado de baterías de mediano tamaño. El consumo es, de momento, el problema más importante a resolver, pero teniendo en cuenta la evolución de los ordenadores industriales y registradores (Havskov y Alguacil, 2004) es muy probable que la transmisión por cable desaparezca en futuros diseños de antenas portátiles.

1.2.5.4. Conversión A/D

Existen muchos tipos de conversores A/D en el mercado, entre los cuales los más utilizados tradicionalmente han sido el de tipo flash, el de rampa, el de doble rampa y el de aproximaciones sucesivas (ver, por ejemplo, Austerlitz, 1991, pp. 50-56). Los dispositivos basados en estas técnicas de conversión ofrecen una resolución máxima de 16 bits, ya que la tecnología actual no permite ajustar y mantener los distintos componentes del conversor con las precisiones necesarias para obtener mejores resoluciones. Además, el propio ruido térmico originado en los componentes, contactos y conductores limita la resolución que se puede obtener mediante estas técnicas.

Como se vio al principio del capítulo, las señales sísmicas tectónicas y volcánicas se caracterizan por tener un amplio rango dinámico, por lo que si se quiere diseñar un sistema con capacidad para registrar al menos una parte importante de las señales no es suficiente con utilizar conversores de 16 bits. Una de las primeras técnicas utilizadas para aumentar el rango dinámico de los conversores A/D fue usar un amplificador de ganancia programable como etapa previa al propio conversor. El dispositivo opera disminuyendo la ganancia del amplificador cuando la señal de entrada supera un nivel prefijado y viceversa. El amplificador tiene programadas varias ganancias, y el valor usado en cada muestreo se guarda como información junto al dato para poder recomponer la señal en la etapa de procesado. De esta forma, conmutando entre las distintas ganancias, se pueden obtener rangos dinámicos de más de 140 dB. El principal problema de este tipo de conversores es que la resolución baja con la ganancia utilizada, de modo que no se pueden recuperar señales pequeñas en presencia de otras de gran amplitud. Por otra parte, muchos de los conversores que utilizan esta técnica dan errores transitorios cuando se producen los cambios de ganancia.

Sin embargo, es posible aumentar la resolución de los conversores A/D, dentro de un cierto límite impuesto por el rango de frecuencias, aplicando la técnica de sobremuestreo. Este método consiste en utilizar una frecuencia de muestreo mayor que la deseada para luego filtrar paso baja la señal obtenida y volver a muestrear con una frecuencia menor. De esta forma los errores de cuantización de las muestras individuales en la secuencia sobremuestreada son promediados con los de muestras vecinas por el filtro paso baja, y en

consecuencia las muestras promediadas tienen mayor precisión y por tanto mayor rango dinámico (Havskov y Alguacil, 2004, pp. 95-98).

Una técnica parecida es la que utilizan los conversores sigma-delta, que actualmente han copado prácticamente todo el mercado de conversores de alta resolución frente a los de ganancia programable citados antes. El fundamento de operación de este tipo de conversores se basa en digitalizar la entrada con una resolución muy baja (generalmente un solo bit) para hacer una estimación del nivel de señal, sumar el error de cuantización a la señal original, hacer una nueva estimación y así sucesivamente. Este proceso continúa indefinidamente, realizándose un seguimiento continuo de la señal de entrada, razón por la cual los diseños basados en este tipo de conversores exigen utilizar uno por cada canal analógico[4]. El valor final de la muestra digital se obtiene promediando un gran número de estimaciones. Los conversores sigma-delta utilizan generalmente un conversor de un solo bit muestreando a frecuencias del orden de 200 kHz, que luego se reduce mediante un filtro digital paso baja a menos de 200 Hz. Mediante esta técnica es posible alcanzar hasta 24 bits de resolución, si bien en banda de frecuencias limitada. La resolución efectiva[5] desciende cuando se aumenta la frecuencia de muestreo. Como se verá en el próximo capítulo, los modelos comerciales de bajo coste con resolución nominal de 24 bits no alcanzan esa resolución efectiva ni siquiera con la mínima frecuencia de muestreo a la que pueden programarse. Aún así, los conversores de técnica sigma-delta ofrecen un notable beneficio frente las técnicas tradicionales de conversión y actualmente son la opción elegida en casi todos los instrumentos sísmicos de alta resolución. Una descripción más detallada de la técnica de muestreo sigma-delta puede consultarse en Havskov y Alguacil, 2004, pp. 98-103, o en Austerlitz, 1991, pp. 56-58.

Otro de los factores relacionados con el proceso de conversión A/D que deben tenerse en cuenta en el diseño de un dispositivo de antena es la frecuencia de muestreo. Como en todos los sistemas digitales, el teorema de Nyquist-Shannon impone un valor mínimo igual al doble de la máxima frecuencia de la señal que se desea muestrear. Sin embargo, en las antenas sísmicas la frecuencia de muestreo debe cumplir además otra condición relacionada con las dimensiones del dispositivo, puesto que si el frente de onda viaja a través de la antena en menos tiempo del que se invierte en tomar dos muestras consecutivas (es decir, el periodo de muestreo) no será posible obtener ninguna información de los tiempos de llegada relativos. Por tanto, la condición que debe cumplirse es:

$$T << D/v \qquad [1.3]$$

siendo T el periodo de muestreo, D la apertura del array en la dirección del rayo y v la velocidad aparente. Por ejemplo, para una antena de 1 km de apertura en la que la máxima velocidad aparente esperada sea de 20 km/s, el periodo de muestreo debería ser mucho menor de 0.05 segundos. Es decir, la frecuencia de muestreo debería ser bastante mayor de 20 mps. En antenas de pequeña apertura como las que se describen en este capítulo la frecuencia de muestreo suele estar entre las 50 y las 200 mps. En sistemas que usen conversores de técnica sigma-delta es necesario llegar a un compromiso entre la frecuencia de muestreo y la resolución efectiva, ya que como se ha visto ésta disminuye al aumentar la primera.

[4] Como se verá en el capítulo 7, esto no es estrictamente cierto para los conversores sigma-delta de última generación.

[5] Por resolución efectiva se entiende el número de bits libres de ruido, que en la práctica constituyen los bits de los que se puede obtener información real de la señal. La resolución efectiva en los conversores sigma-delta es casi siempre menor que la nominal, como demuestra el hecho de que cortocircuitando las entradas analógicas la señal digital de salida no es cero.

1.2.5.5. Preprocesado y grabación

Hasta hace pocos años la capacidad de los medios de almacenamiento masivos y la velocidad de los sistemas informáticos condicionaban estos dos procesos en el diseño de cualquier instrumento sísmico digital. El elevado precio, tamaño y consumo de los discos duros, cintas magnéticas y otros medios de almacenamiento, unidos a su relativamente poca capacidad, hacían impensable considerar el registro continuo de los datos. En el caso de dispositivos de antena esta limitación era más acusada, puesto que el volumen de datos se multiplica por el número de canales operativos. La solución que se adoptaba era grabar sólo los datos que aparentemente presentaban interés, para lo cual el sistema de registro se dotaba de un algoritmo de disparo (normalmente de tipo STA/LTA, ver Lee y Stewart, 1981) que discriminaba los eventos sobre el nivel normal de ruido sísmico. La limitación no provenía tan solo de la capacidad de los medios de almacenamiento, sino también de la velocidad a que eran capaz de operar los equipos informáticos, que normalmente no era suficiente como para seguir gestionando el programa de adquisición de datos a la vez que la grabación.

Actualmente la situación es muy diferente. Los discos duros ofrecen capacidades y velocidades de acceso impensables hace pocos años, y la velocidad de los microprocesadores y tarjetas de periféricos también ha aumentado espectacularmente. Esto hace que en pocos sistemas modernos se siga implementando la detección de eventos como única opción de registro, optándose casi siempre por el registro continuo o por una solución conjunta, en la que se almacenan todos los datos y también se implementa un algoritmo de disparo para almacenar ficheros de evento. Estas soluciones permiten tener acceso a mucha información que antes se perdía. En este sentido se han visto beneficiados especialmente los estudios de áreas volcánicas en las que, según se ha visto al principio del capítulo, se presentan señales como el trémor, sin fases y con una distribución de energía prácticamente constante en el tiempo, para las que es muy difícil diseñar e implementar algoritmos de detección efectivos.

En cualquier caso, la detección de eventos puede implementarse a posteriori, durante el análisis de los datos registrados. Del mismo modo, los parámetros de localización (azimut y lentitud) tampoco se suelen calcular en tiempo real, sino durante la etapa del procesado de datos. Así las cosas, el preprocesado en antenas de pequeña apertura, especialmente en portátiles, suele limitarse a la organización de los datos en ficheros con un formato predefinido, que además de los propios datos deben incluir la información necesaria para recomponer las señales sin ningún tipo de ambigüedad. Esta información adicional debe constar, al menos, de los parámetros de adquisición (ganancia, frecuencia de muestreo, resolución,…) y el tiempo de adquisición de la primera muestra, a partir del cual se pueden calcular los de las demás. En algunos sistemas se incluye además información complementaria, como versiones de los programas utilizados, número de muestras del fichero o valor máximo de las muestras en cada uno de los canales.

A veces el preprocesado incluye la compresión de los datos, lo cual resulta especialmente útil en sistemas con transmisión telemétrica o telefónica, en los que el ancho de banda y/o el tiempo disponible de transmisión limitan el volumen de datos que se puede transmitir.

El medio de almacenamiento más usual son los discos duros, que con las capacidades actuales proporcionan espacio para semanas o meses de registro continuo, incluso en antenas con un número elevado de canales y resoluciones y frecuencias de muestreo altas. Se suelen emplear discos duros de 2.5’’, del tipo de los usados en ordenadores portátiles, que además de su reducido peso y tamaño ofrecen la ventaja de requerir alimentación única. El mayor problema que presentan es que, debido a que contienen mecanismos y partes móviles, son bastante sensibles a golpes y pueden dar problemas de funcionamiento en entornos hostiles. Este inconveniente es especialmente acusado en antenas portátiles, en las que los desplazamientos son habituales y las condiciones ambientales muchas veces adversas.

Una opción atractiva son las memorias semiconductoras, por ejemplo las de tipo FLASH, que ofrecen muy buenas prestaciones a la par que un diseño sin partes móviles, lo cual elimina los problemas descritos para los discos duros. Las memorias FLASH parecen la opción más adecuada para diseños futuros, si bien de momento siguen siendo demasiado caras como para resultar rentables como medio de almacenamiento único. Como se verá en los capítulos 3 y 7, sí que puede resultar interesante utilizar memorias semiconductoras de pequeña capacidad como medio almacenamiento de información crítica (sistema operativo, programas de arranque,…), utilizando para la grabación de los datos discos duros remotos de gran capacidad.

1.2.5.6. Temporización

Ya se ha apuntado antes en este capítulo la importancia que cobra en las antenas sísmicas el sistema de temporización y sincronismo de los datos. Para lograr las precisiones necesarias en la sincronización del proceso de adquisición de los datos de los distintos canales es necesario contar con una señal de referencia de precisión, que en los diseños modernos casi siempre es suministrada por un receptor GPS.

Aunque el sistema GPS fue originalmente concebido con fines militares, en la actualidad se benefician del mismo un amplio abanico de aplicaciones civiles, entre las que destacan por su importancia la navegación aérea, terrestre y marítima. En el ámbito científico, y más concretamente en el campo de la instrumentación, el GPS supuso una verdadera revolución, ya que proporciona una posibilidad sencilla, barata y fiable de obtener localizaciones precisas de equipos remotos y de conseguir datos sincronizados en redes de instrumentos que deban operar coordinadamente, como es el caso de las antenas sísmicas. Hasta la aparición del GPS en la década de los ochenta, la parte más crítica en el diseño de un instrumento para la adquisición de datos sísmicos era precisamente el subsistema de tiempo. En equipos destinados a operar en entornos remotos la solución solía basarse en receptores de señales horarias transmitidas por radio. En España el Observatorio Naval de San Fernando transmitía en esa época señales horarias en onda corta durante una media hora diaria. Sin embargo, debido a las especiales condiciones de propagación en esa banda, en muchos observatorios situados en el territorio nacional la recepción era defectuosa o nula. Así, en el Observatorio de Cartuja, ante la imposibilidad de usar la señal de San Fernando se optó por emisoras situadas en otros países de Europa como la suiza HBG, la alemana DCF, la inglesa MSF o incluso emisoras de radiodifusión como *Radio France Inter* (Alguacil, 2003). Otra opción que se utilizó durante un tiempo fue la señal procedente de estaciones Omega, un sistema de ayuda a la navegación que, aunque no transmitía señales de tiempo, suprimía la portadora en ciclos sincronizados con el tiempo universal, lo cual permitía mantener un reloj en sincronismo. El principal inconveniente de todos los sistemas basados en la recepción de señales de radio era la generalmente débil señal de recepción, cuya calidad dependía además del emplazamiento del equipo. A estos problemas conocidos había que sumarle otros imprevistos, como el cambio de frecuencia de la señal, que podía dar al traste con un diseño realizado para adaptarse a unas características muy concretas de la misma. Aún hoy en día se mantienen varios de estos sistemas basados en tierra, como las señales de tiempo DCF77[6], MSF[7], WWVB[8] o los sistemas de navegación LORAN[9], sustituto de Omega, y CHAYKA[10]. Todos estos sistemas transmiten señales codificadas de radio en baja frecuencia, entre 60 y 110 kHz, y pese a la mejora de sus características respecto a las de los sistemas originales,

[6] www.dcf77.com/index.htm
[7] www.npl.co.uk/time/
[8] tf.nist.gov/stations/wwvb.htm
[9] www.navcen.uscg.gov/loran/Default.htm
[10] en.wikipedia.org/wiki/CHAYKA

siguen presentando importantes desventajas frente a los basados en satélites: su cobertura es limitada, son muy dependientes de las condiciones atmosféricas y su precisión varía en función de la localización geográfica. En el campo de la instrumentación geofísica las ventajas del GPS son tan evidentes que sólo en casos muy concretos, en los que el uso de este tipo de receptores se vea dificultado por circunstancias particulares, se suele contemplar la utilización de otro tipo de soluciones. De los tres dispositivos que se presentan en este trabajo se utilizaron receptores GPS en la antena del Vesubio y en las portátiles del Instituto Andaluz de Geofísica. La antena del Gran Sasso, por su parte, representa una de esas raras excepciones en las que las especiales características del diseño obligan a adoptar otra solución.

Como se verá en el próximo capítulo, los receptores GPS orientados a su integración en sistemas autónomos o embebidos proporcionan una señal de un pulso por segundo (en adelante PPS), cuyo flanco inicial está sincronizado con el tiempo universal con precisiones mejores que 1 μs. A partir de ella se sintetizan las señales necesarias para garantizar la sincronización del proceso de muestreo en todos los puntos de adquisición, para lo cual se pueden utilizar varias estrategias. Dos de ellas se muestran en la figura 1.15. Estas técnicas son las que se han utilizado en el diseño del sistema de sincronismo de las antenas del Gran Sasso (a) y de las antenas portátiles del Vesubio y del IAG (b), con ciertas particularidades que se describirán en los capítulos 3, 4 y 5, respectivamente.

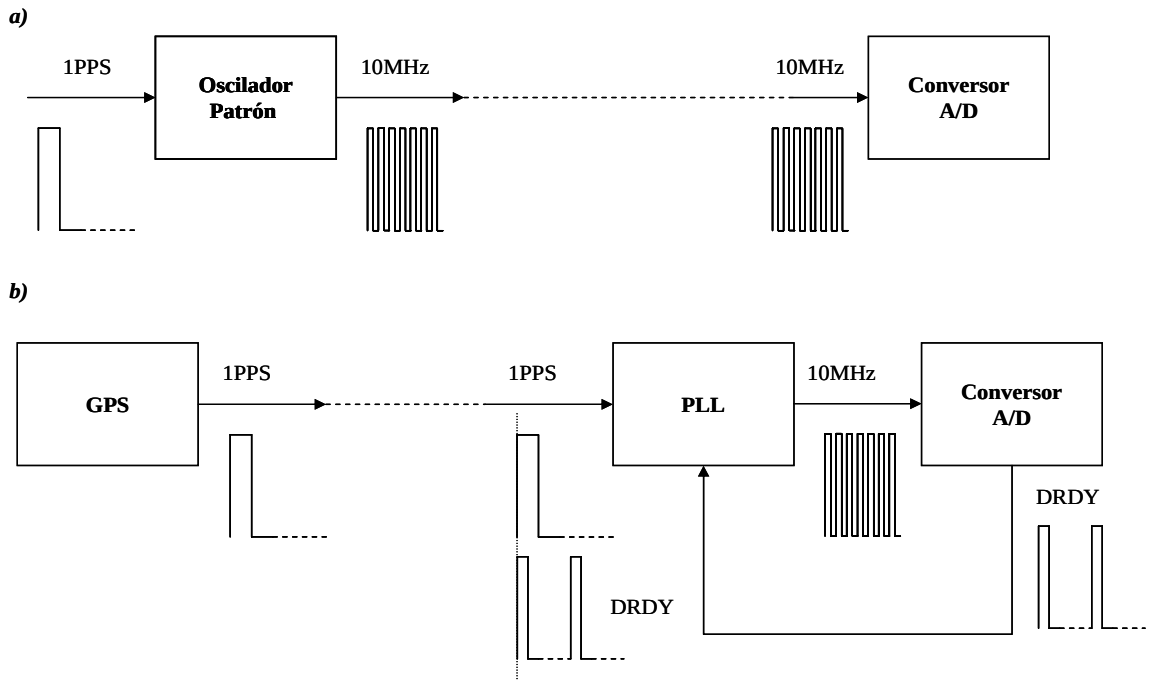


Figura 1.15. *Técnicas de sincronización utilizadas en la antena del Gran Sasso (a) y en las portátiles del Vesubio e IAG (explicaciones en el texto).*

En a) se genera, a partir de la señal de PPS, la señal de reloj de alta frecuencia que los conversores A/D usan como base de tiempo para su operación. En la figura esta frecuencia es de 10 MHz, aunque más adelante se verá que en nuestras aplicaciones es algo menor por razones de diseño. Esta señal, generada en un único oscilador patrón de precisión, se transmite a todos los puntos de adquisición, de modo que se garantiza que todos los conversores A/D del sistema operarán a la misma velocidad y por tanto su frecuencia de muestreo será exactamente la misma. Por tanto, para asegurar el sincronismo de las muestras sólo será necesario realizar una sincronización inicial de todos los conversores.

En la figura 1.15.b) lo que se transmite a los puntos de adquisición es la señal de PPS del receptor GPS. A partir de ella se genera localmente la señal de 10 MHz, para lo cual se utilizan circuitos de enganche de fase (PLL, ver capítulo 2). En nuestra aplicación los

circuitos PLL intentan mantener en fase la señal de fin de conversión de dato (DRDY) y la señal de PPS de referencia, incrementando o disminuyendo la frecuencia de la señal de 10 MHz cuando se producen desfases.

Existen otras posibles soluciones al problema del sincronismo. Se podría, por ejemplo, sobremuestrear las señales analógicas en cada punto de adquisición utilizando osciladores no sincronizados. Las muestras se enviarían en tiempo real a una unidad central de registro, donde se les pondría la marca de tiempo y se promediarían mediante un filtrado digital. Esta técnica facilita el diseño del sistema de sincronismo en sí, ya que no hay que transmitir señales de precisión, pero a cambio complica el preprocesado de señal. Además es necesario conocer los retrasos de transmisión con una precisión mejor que un periodo de muestreo.

En sistemas que utilizan telemetría analógica el sistema de sincronismo se simplifica notablemente, ya que no es necesario transmitir señales de tiempo de precisión ni generarlas localmente a partir de marcas de referencia. Esta circunstancia se aprovechará en las antenas portátiles del IAG que se presentarán en el capítulo 5.

1.2.5.7. Alimentación

Teniendo en cuenta que el despliegue de las antenas portátiles suele ser temporal, lo normal es que la alimentación se proporcione a partir de baterías que se cambian cuando se descargan. La autonomía de estos sistemas depende del consumo y del tipo y capacidad de las baterías que se utilizan, pudiendo ir desde algunos días hasta varias semanas. Por esta razón uno de los factores críticos en el diseño de dispositivos portátiles es el consumo, que debe mantenerse tan bajo como sea posible para aumentar la autonomía. Hay que partir de la premisa de que una sola batería debe alimentar todo el sistema, que generalmente consta de un módulo central y varios módulos periféricos, como pueden ser los digitalizadores en cada punto de adquisición y el módulo de GPS. Generalmente la tensión de la batería se manda por cable a los distintos módulos, en cada uno de los cuales se monta un circuito de fuente para convertir esta tensión a la necesaria para alimentar el circuito correspondiente. El circuito de fuente debe ser capaz de aceptar un amplio margen de tensión a la entrada, puesto que ésta puede variar mucho según el estado de la batería. En módulos con consumos relativamente altos se suelen utilizar circuitos de fuente conmutados (por ejemplo el LM2576) ya que su rendimiento es mucho mayor que el de los circuitos lineales (entre los cuales la familia 78XX es una opción típica). A cambio las fuentes conmutadas son más susceptibles de presentar rizado en la tensión de salida, por lo que hay que tener la precaución de filtrarla adecuadamente.

En instalaciones permanentes las baterías se suelen dotar de un sistema de carga mediante paneles solares o generadores eólicos, intercalando entre ellos y la batería un regulador de carga (Havskov y Alguacil, 2004, pp. 204-207). El regulador cumple varias funciones. En primer lugar, evita la descarga de la batería a través del panel solar cuando no hay suficiente iluminación. En segundo, evita que las baterías se sobrecarguen, para lo cual el regulador deja de cargar cuando la tensión de la batería llega a un nivel prefijado. En dispositivos más sofisticados, el sistema que se alimenta (en nuestro caso, el instrumento sísmico) se desconecta de la batería cuando el voltaje de ésta baja hasta un valor mínimo predeterminado. Así se consigue, por una parte, alargar la vida de la batería, que se reduce a causa de las descargas profundas. Por otra, se evitan los funcionamientos erráticos del instrumento que suelen producirse cuando la tensión baja hasta el límite de operación de los distintos módulos.

En otros sistemas con emplazamientos fijos se cuenta con una infraestructura que permite alimentar el instrumento directamente de la red eléctrica, lo cual facilita el diseño del sistema de alimentación. En estos casos, como se verá en el capítulo de la antena del Gran Sasso, el consumo deja de ser un factor crítico en el diseño.

1.2.5.8. Transmisión de datos

En las antenas sísmicas portátiles los datos se suelen almacenar localmente y no es usual transmitirlos a otro punto para su proceso. La razón es que, como se ha visto, la instalación de estos dispositivos suele ser temporal y orientada a la recogida de datos durante un periodo breve de tiempo para el estudio de un área sísmica activa, por lo que generalmente no resulta rentable instalar un sistema de telemetría. Sobre todo teniendo en cuenta que el sistema de alimentación normalmente está basado en baterías sin cargadores y por tanto se requiere la presencia de personal de mantenimiento cada cierto tiempo. Así, lo más sencillo es prescindir del sistema de telemetría en tiempo real e incluir entre las tareas de mantenimiento la recogida de los datos registrados en el periodo entre cada cambio de baterías.

En instalaciones permanentes se puede dotar al instrumento de un sistema de telemetría que, además de hacer innecesarias las continuas visitas de mantenimiento para la recogida de datos, permite realizar su procesado en tiempo real o casi real. Aparte de los transmisores de radio en VHF o UHF tradicionalmente usados (cuyo ancho de banda normalmente no va a ser suficiente para transmitir el volumen de datos de un *array*), en la actualidad existen otras tecnologías como la transmisión digital por espectro disperso[11] (*spread spectrum*). Esta técnica se basa en la utilización de una banda ancha de frecuencias que comprende diferentes canales de transmisión. De esta forma, aunque la potencia de transmisión es similar a la de los transmisores tradicionales de banda estrecha, la densidad espectral de potencia (medida en W/Hz) es mucho menor. Ahí reside una de las mayores ventajas de esta tecnología, ya que la baja densidad espectral permite ocupar la misma banda que otras transmisiones de banda estrecha sin sufrir ni producir interferencias. Por esta razón su uso es libre hasta una potencia nominal de 100 mW y no es necesario solicitar licencias para el uso de la banda de frecuencias, como ocurre con las transmisiones en UHF o VHF.

El uso de una banda ancha de transmisión se basa en la utilización alternativa de los distintos canales definidos en la banda. La conmutación entre canales se realiza a la vez en transmisor y receptor, según un algoritmo pseudo aleatorio predefinido y de forma transparente al usuario[12]. El algoritmo incluye el cambio de canal en caso de que la recepción en alguno de ellos no sea buena, lo cual asegura la integridad de la transmisión con una pérdida mínima de las prestaciones del enlace.

La implementación de la transmisión mediante la técnica de espectro disperso es sencilla. Existen transmisores en formato radio-módem que sólo deben conectarse al puerto serie de un PC y suministrarles el flujo de datos en serie, de manera similar a como se haría si los datos se enviaran en formato RS-232 a través de cable. La gestión de errores, intercambio de tramas de control y selección del canal más adecuado para transmitir en cada momento los realizan transmisor y receptor de modo automático y transparente al usuario. Pese a que las velocidades que se pueden conseguir con esta técnica son altas (existen varios modelos de radio-módems en el mercado con velocidades de hasta 115 Kbaudios), en la práctica para enlaces largos y en condiciones reales no se suelen alcanzar. Incluso con las velocidades máximas, el volumen de datos generado por una antena sísmica en registro continuo suele ser demasiado grande como para usar transmisión serie. La solución puede ser una tarjeta de red *ethernet* con transmisores de espectro disperso, con la que se pueden conseguir velocidades de

[11] En realidad el fundamento de este sistema de transmisión no es reciente. Curiosamente fue una actriz de Hollywood, Hedy Lamarr, la que lo inventó y patentó en 1942, con la intención de desarrollar un sistema de guiado de torpedos por radio que resultara difícil de interferir. Su implementación en equipos electrónicos tuvo que esperar hasta que la tecnología evolucionó lo suficiente como para fabricar los sistemas necesarios, en la década de los 60.

[12] Esta técnica se conoce como *frequency hopping* o salto de frecuencias. Existen otras formas de implementar la comunicación mediante espectro disperso, pero su fundamento es parecido.

enlace similares a las de redes por cable (10 Mbaudios en condiciones óptimas). En este caso las prestaciones del PC de control deben ser suficientes como para admitir un sistema operativo multitarea que gestione la interfaz *ethernet*. Los enlaces pueden establecerse punto a punto o multipunto, con lo que es posible implementar una red LAN con varios *arrays* o cualquier otro tipo de instrumentos como nodos.

El principal inconveniente que presenta la tecnología de espectro disperso es que, debido a su alta frecuencia (en Europa se utiliza la banda de 2.4 GHz) es imprescindible que exista visión directa entre el transmisor y el receptor. Por otra parte, las distancias que son capaces de cubrir estos enlaces con la potencia permitida de 100 mW no son excesivamente grandes (algunos km). Para distancias mayores es necesario utilizar repetidores o antenas de ganancia.

En casos muy concretos en los que la infraestructura lo permita, es posible implementar la comunicación entre nuestra antena sísmica y el centro de proceso a través de cable, mediante una red LAN o conexión directa a Internet. Teniendo en cuenta que los dispositivos de antena suelen situarse en entornos aislados (entre otras cosas, para minimizar en los registros el ruido originado por la actividad humana) no es la opción más corriente, pero sin duda sí la más adecuada para permitir el acceso a los datos desde cualquier ordenador con conexión a Internet.

Otras posibles soluciones al problema de la transmisión de datos son la línea telefónica, convencional o GSM, y los enlaces vía satélite. El tiempo de transmisión con todas estas opciones resulta bastante caro, por lo que no se implementan como enlaces continuos sino suministrando determinados paquetes de datos a petición del usuario. Para ello se suele enviar periódicamente y de forma automática un resumen de la actividad, consistente por ejemplo en los datos comprimidos de una estación significativa. A partir del fichero resumen el operador puede discernir los eventos de interés y solicitar los datos completos correspondientes al periodo de tiempo que le interesa.

En el capítulo 7 se volverá sobre algunos de los temas presentados en esta sección, orientándolos a la implementación de posibles mejoras en los dispositivos que se describen en esta memoria.

1.3. Nuestros diseños

1.3.1. Objetivos

La presente tesis se centra en el diseño y desarrollo de tres dispositivos de antena sísmica de pequeña apertura. El primero de ellos será un *array* fijo y subterráneo, cuyo emplazamiento serán los LNGS (*Laboratori Nazionali del Gran Sasso)*, en el macizo del Gran Sasso (Italia). El segundo será un array portátil, aunque dispondrá de un emplazamiento casi permanente en el volcán Vesubio, también en Italia. El tercero será, asimismo, una antena portátil, en esta ocasión para el Instituto Andaluz de Geofísica perteneciente a la Universidad de Granada (España). Su utilización se restringirá a campañas de recogida de datos en áreas sísmicamente activas, por lo que no contará con un emplazamiento permanente.

A continuación se presentan brevemente los tres dispositivos, así como los objetivos que se pretende cubrir con su construcción.

1.3.1.1. La antena del Gran Sasso

Desde 1986 operan en la región central de Italia los Laboratorios Nacionales del Gran Sasso (*Laboratori Nazionali del Gran Sasso,* en adelante LNGS), unas instalaciones pertenecientes al Instituto Italiano de Física Nuclear, cuyo principal objetivo son los estudios sobre la física de partículas subatómicas y astrofísica. Los laboratorios están situados en un túnel subterráneo a unos 120 km al este de Roma, construido a partir de un túnel de la autopista que une las ciudades de L'Aquila y Teramo. El escudo natural que forman los 1400 m de roca que hay por encima de los laboratorios los convierte en un emplazamiento ideal para llevar a cabo estudios en distintos campos de la física de partículas. Desde el punto de vista de la geofísica, los LNGS presentan varias características interesantes.

En primer lugar, están situados muy cerca de uno de los mayores sistemas de fallas activas de los Apeninos centrales. Aunque actualmente la actividad es moderada, la región ha experimentado terremotos destructores en el pasado, como el de 1703 (de magnitud 7 aproximadamente) o el de Avezzano de 1915 (Amoruso *et al.*, 1998).

En segundo, el hecho de que los laboratorios estén situados bajo tierra hace que las características de las señales sísmicas sean especiales, como se verá en el capítulo 3, produciendo unos registros muy limpios y con llegadas de las distintas fases claras y bien definidas.

Por último, desde el punto de vista logístico los LNGS conforman un entorno especialmente apropiado para la instalación de instrumentación de cualquier tipo, ya que la infraestructura existente permite contar con facilidades que normalmente no son directamente accesibles. Así, un aspecto que en los diseños de antenas portátiles suele ser crítico como el sistema de alimentación se ve facilitado en este caso por la disponibilidad de una red de corriente alterna de potencia en todos los puntos de adquisición. Del mismo modo, la dificultad para gestionar receptores GPS por el carácter subterráneo de los laboratorios se ve subsanada por la disponibilidad de una señal de PPS procedente de un patrón atómico de tiempo operativo en los laboratorios.

Estas características convierten a los LNGS en el emplazamiento ideal para un sistema de antena sísmica de pequeña apertura, que contribuirá al conocimiento de una importante área sismogenética. El principal objetivo de estudio al que se orientará la antena son los fenómenos de propagación de ondas y procesos de fuente de los microterremotos que constituyen, en la actualidad, la actividad sísmica dominante en esta región. No obstante, el dispositivo permitirá realizar, como se ha visto en la sección de aplicaciones, un amplio abanico de estudios aplicados a la sismicidad local y regional, tanto basados exclusivamente en los registros de la antena como en conjunción con las estaciones de la red sísmica nacional operativas en la región. Por otra parte, la posibilidad de incorporar sensores de banda ancha en alguno de los puntos de adquisición extendería el ámbito de estudio a señales sísmicas lejanas.

1.3.1.2. La antena del Vesubio

Quizás la erupción volcánica más popularmente conocida de la historia, por sus consecuencias y por la impronta arqueológica que dejó en las ciudades de Pompeya y Herculano, sea la del Vesubio del año 79 d.C. Sin embargo, como se verá en el capítulo 4, ésta erupción no ha sido sino uno de los muchos episodios plinianos y subplinianos que se han podido constatar en este volcán a partir de muestras geológicas. La última erupción importante tuvo lugar en 1944, pero desde entonces el sistema volcánico no ha dejado de presentar una actividad sísmica moderada representada por algunos cientos de terremotos al año. La región también presenta actividad geotermal y fenómenos de deformación del suelo, entre los cuales destacan los que se produjeron en el área de *Campi Flegrei* en los años 1969-72 y 1982-84 (Zollo *et al*., 2006).

Su agitada historia eruptiva y el alto índice de urbanización del área metropolitana de Nápoles convierten al Vesubio en uno de los volcanes con mayor índice de riesgo del mundo hoy en día. Como consecuencia, es también uno de los más estudiados y monitorizados. La red sísmica gestionada por el *Osservatorio Vesuviano* incluye numerosas estaciones de corto periodo, banda ancha y acelerómetros, además de contar con personal y presupuestos suficientes como para garantizar un mantenimiento adecuado. Sin embargo, hasta el momento no se disponía de dispositivos de antena densos y de pequeña apertura, los cuales, como se ha comentado previamente, suministran en áreas volcánicas una valiosa información inasequible para las redes sísmicas convencionales. El dispositivo que se presenta en este trabajo pretende cubrir ese hueco en la instrumentación dedicada a la vigilancia de la actividad sísmica de origen volcánico, aunque sin renunciar al estudio de la sismicidad tectónica local y regional. El sistema, tal y como se presenta en esta memoria, puede definirse como un módulo de antena sísmica de alta densidad y con registro local de los datos. Sin embargo este diseño se engloba dentro de otro más amplio, que contempla la construcción de un dispositivo gemelo para determinar la posición y evolución temporal del trémor y otras señales volcánicas mediante la técnica del cruce de rayos. Para dotar al conjunto de capacidad de localización en tiempo real, se plantea la posibilidad de implementar transmisión telemétrica de los datos desde cada una de las antenas hasta el centro de control de la red sísmica, situada en la sede del *Osservatorio Vesuviano*, en la ciudad de Nápoles.

Teniendo en cuenta el objetivo que se persigue, se podría decir que el emplazamiento del dispositivo será permanente o casi permanente. Sin embargo, se decidió orientar el diseño hacia un sistema portátil, ya que el *Osservatorio Vesuviano*, además de la vigilancia del área del Vesubio, participa activamente en la monitorización y control de otras áreas volcánicas activas de Italia como las islas de Vulcano o Strómboli. De esta forma se posibilita un despliegue rápido del dispositivo en caso de aumento de actividad en alguna de esas regiones, como la que se produjo en Strómboli en diciembre de 2002 y primeros meses de 2003 (ver, por ejemplo, D'Auria *et al.*, 2006, Esposito *et al.*, 2006).

1.3.1.3. Las antenas portátiles del IAG

El Instituto Andaluz de Geofísica y Prevención de Desastres Sísmicos (IAGPDS o, más abreviadamente, IAG), de la Universidad de Granada, es en la actualidad la principal institución en España en el estudio de la actividad sísmica de la región sur del país. Aunque el IAG fue fundado como tal en el año 1989, en el área de la sismología local y regional puede considerarse heredero directo del Observatorio de Cartuja, institución fundada a principios del siglo pasado por los padres jesuitas y que mantuvo, desde sus inicios, esta disciplina como uno de los objetivos principales de sus esfuerzos investigadores y técnicos. A principios de los años 90 el IAG incorporó la volcanología como nueva línea de investigación. Poco tiempo después la necesidad de contar con instrumentación propia se hizo evidente, por lo que se establecieron colaboraciones con instituciones externas (Departamento de Volcanología del Museo de Ciencias Naturales del CSIC y *Osservatorio Vesuviano* de Nápoles (Italia)) para desarrollarla. El resultado fue la construcción de unos módulos de antena sísmica portátiles de dieciséis bits y ocho canales de adquisición sincronizados mediante GPS, que hasta hace poco tiempo han constituido el principal argumento instrumental del IAG para la obtención de datos propios en áreas volcánicas activas. Estos módulos (a los que en adelante llamaremos por brevedad módulos de dieciséis bits) se han utilizado con éxito en numerosas campañas de adquisición de datos, en volcanes como el Etna (ver, por ejemplo, Del Pezzo *et al.*, 2000), Strómboli (Del Pezzo *et al.*, 1998) y Vesubio (Del Pezzo *et al.*, 1999), en Italia, Teide, en España (Del Pezzo *et al.*, 1997, Almendros *et al.*, 2000), Isla Decepción, en la Antártica (Almendros *et al.*, 1997, Ibáñez *et al.*, 2000), Copahue, en Argentina (Ibáñez *et al.*, 2007) o Volcán de Fuego de Colima, en México. También se han empleado para recoger datos en

áreas tectónicas como en el caso de la crisis sísmica de Iznájar (Carmona *et al.*, 2002; Fernández, 2007, pp. 36-41), en la provincia de Granada.

Pese al excelente comportamiento de los módulos de dieciséis bits, los más de quince años que han estado operativos han pasado factura. Por una parte, en forma de degradación y deterioro de los distintos elementos, lógica en cualquier instrumento y mucho más en aquellos que operan normalmente bajo condiciones ambientales adversas, como es el caso. Por otra, debido a la rápida evolución de equipos informáticos y electrónicos, que ha hecho que las prestaciones de los módulos de dieciséis bits hayan quedado comparativamente mermadas respecto a otros instrumentos más modernos, en particular en parámetros como la resolución de los conversores A/D o el registro continuo de los datos.

El desarrollo de las nuevas antenas del IAG se planteó, por tanto, como sustitución de los módulos de dieciséis bits. Como se verá en el capítulo 5, el principal objetivo que se perseguía desde el punto de vista del diseño era conseguir aprovechar la experiencia y elementos desarrollados para la antena del Vesubio, si bien introduciendo ciertas simplificaciones orientadas a conservar la filosofía y flexibilidad de configuración de los antiguos módulos de dieciséis bits.

1.3.2. Realización física

La realización física de un sistema de antena sísmica que integre los distintos procesos descritos puede realizarse de distintas formas (Ortiz *et al.*, 2001, pp. 299-304; Abril e Ibáñez, 2000). A priori, la filosofía más sencilla consistiría en utilizar muchas estaciones autónomas, dispuestas en un espacio reducido y todas ellas con un sistema de sincronismo temporal (normalmente un receptor GPS). Este sistema haría innecesario el tendido de cables entre las distintas estaciones y no precisaría un módulo central de control. Sin embargo, no sería posible tener información en tiempo real, además de requerir una elevada inversión en sistemas de registro y de sincronismo temporal, y precisar de tantas fuentes de alimentación independientes como estaciones operativas. Esta solución no se utiliza en el diseño de dispositivos de antena de pequeña apertura, aunque es una configuración que puede adoptarse para adaptar instrumentos autónomos de registro a su uso como *array*.

El otro extremo sería utilizar una única unidad de registro con todos los sensores conectados a ella y un solo sistema de tiempo local. Esta configuración permite tener información en tiempo real, pero requiere gran capacidad de almacenamiento y velocidades de comunicación y cálculo elevadas. Representa una posible solución para dispositivos fijos en los que no es fundamental la flexibilidad en la configuración espacial de los sensores, dado que ésta se establece en la etapa de diseño y no se modifica posteriormente. En este trabajo se ha adoptado esta filosofía de diseño para las antenas del Gran Sasso y el Vesubio.

Por último, se pueden adoptar soluciones intermedias entre las dos descritas. Por ejemplo, se podría disponer de varias unidades de registro, cada una con capacidad para varios canales y dotadas de un sistema de tiempo absoluto de precisión. De esta forma los sistemas de adquisición no precisan elevadas prestaciones y se aumenta la flexibilidad del conjunto. Esta opción es una buena solución para dispositivos portátiles, en los que la configuración espacial de los sensores no es fija sino que depende de cada aplicación. Con módulos de diez canales se podría, por ejemplo, implementar varias antenas en torno a un cráter activo para utilizar la técnica del cruce de rayos, o bien disponer de un único array denso circular cuando el azimut preferente no es conocido. Esta filosofía es la que se adoptó en las antenas portátiles de dieciséis bits y también en las nuevas antenas portátiles del IAG que se han diseñado en este trabajo para sustituirlas.

Por otra parte, conviene en este punto hacer una anotación en relación con lo descrito en la sección 1.2.3, en la que se presentaron todos los procesos que pueden intervenir en un

instrumento sísmico. Esta secuencia de procesos se implementa físicamente como una serie de subsistemas que buscan adaptarse a los requerimientos de diseño de cada aplicación particular. Las soluciones adoptadas para los dispositivos que se presentan en esta tesis prescinden, en ocasiones, de subsistemas específicos para ejecutar algunos de estos procesos. Así, mientras los procesos de conversión A/D, transmisión de datos, grabación de los mismos, temporización y alimentación de los distintos módulos, ocuparán una parte central en los siguientes capítulos, otros procesos descritos en la sección 1.2.3 no se discutirán, o sólo se hará parcialmente, debido a las siguientes razones:

Sensado.- Los sensores que se emplean en las antenas del Gran Sasso y Vesubio son modelos comerciales y sólo precisan las precauciones de instalación propias de este tipo de dispositivos (Havskov y Alguacil, 2004, pp. 188-190), que quedan fuera del ámbito de esta tesis. Las antenas portátiles del IAG suelen utilizar sensores de 4.5 Hz ecualizados para extender su respuesta a 1Hz, pero el diseño de los circuitos necesarios para conseguirlo no forma parte de este trabajo, por lo que tan solo se dará una breve descripción de estos sensores, como parte de los antiguos módulos de dieciséis bits, en el capítulo 5. Una explicación más detallada de la técnica y circuito de ecualización puede consultarse en Havskov y Alguacil, 2004, pp. 49-51, o en Ortiz *et al.*, 2001, pp. 246-254).

Acondicionamiento de señal.- Los conversores A/D basados en la técnica sigma-delta de muestreo incorporan, como se ha visto, un filtro digital que hace las veces de filtro paso baja, por lo que no es necesario implementar además un filtro *anti-aliasing* analógico (Havskov y Alguacil, 2004, pp. 104-108). Por otra parte, el conversor AD7710 que se ha utilizado en los tres diseños lleva implementado un amplificador de ganancia programable, por lo que tampoco es necesario diseñar un preamplificador analógico adicional. Por último, como ya se ha dicho, los conversores sigma-delta realizan un seguimiento continuo de la señal de entrada y por tanto en los sistemas que los utilizan es necesario (generalmente) utilizar un conversor A/D por cada canal de entrada, lo cual elimina la necesidad de multiplexar las señales analógicas.

Como se verá, la etapa de acondicionamiento de señal se limitará a adaptar la señal de salida de los sensores a la entrada de los conversores mediante resistencias de amortiguamiento y a la implementación de filtros pasivos RC para el filtrado de radiofrecuencias.

Preprocesado.- Las tres antenas que se describen operarán con registro continuo, por lo que no será necesario implementar un algoritmo de detección de eventos. En lo relativo al preprocesado, la principal tarea que se realizará es la conversión del formato de los datos crudos (tal como se obtienen tras el muestreo) y de los parámetros de adquisición al definido para los ficheros de datos.

Transmisión de ficheros de datos.- El diseño de las antenas portátiles del Vesubio y del IAG contempla la grabación local (es decir, en la propia antena) de los ficheros de datos. No se describe en este trabajo la implementación de un sistema de telemetría para la transmisión de los ficheros, aunque en el capítulo 7 se discuten algunos factores que habría que tener en cuenta para llevarla a cabo.

En la antena del Gran Sasso los datos se almacenan en el disco duro del servidor, pero en este caso sí está previsto permitir el acceso remoto a ellos. Dado que el servidor es un PC de sobremesa con un sistema operativo de tipo *Windows* y conexión a Internet, no es necesario transmitir los ficheros de datos sino simplemente compartir los correspondientes directorios y permitir el acceso mediante contraseña al personal autorizado.

1.4. Estructura de la tesis

En este capítulo se han presentado los fundamentos básicos de los sistemas de antena sísmica, necesarios para abordar la descripción de los tres dispositivos que se han diseñado y desarrollado en este trabajo. En el próximo capítulo se introducirán la metodología y herramientas utilizadas para realizar los programas de aplicación y para diseñar las tarjetas de circuito impreso de los distintos subsistemas de cada uno de los *arrays*. Asimismo, se presentarán en el capítulo 2 algunos de los componentes y circuitos utilizados en los tres desarrollos: el conversor A/D, los microcontroladores PIC, las tarjetas de PC industrial, los receptores GPS y los circuitos PLL.

El bloque más importante de la tesis se concentra en los capítulos 3, 4 y 5, en los que se describen las antenas del Gran Sasso, Vesubio e IAG, respectivamente. Esta descripción se abordará tanto desde el punto de vista del diseño e integración de los distintos elementos *hardware* como de la programación de los correspondientes controladores. Las secciones finales de los tres capítulos se ocuparán de aspectos prácticos del montaje y operación del sistema.

En el capítulo 6 se describirán las aplicaciones y se discutirán los principales resultados obtenidos a partir de datos registrados con las antenas. Para ello se analizarán los métodos y conclusiones de varios artículos y presentaciones en congresos de distintos autores, prestando especial atención a los aspectos relativos a la operación de los sistemas, por encima de los resultados científicos de las publicaciones.

En el capítulo 7 se apuntarán algunas posibles líneas de trabajo futuro. Teniendo en cuenta el carácter técnico de esta tesis, éstas se centrarán sobre todo en posibles mejoras de los sistemas, basadas en la utilización de componentes más modernos y de mejores prestaciones que los empleados o en ideas concebidas a partir de la experiencia en la operación de los tres dispositivos.

En el capítulo 8 se ofrecerán las conclusiones a las que se ha llegado tras la redacción de esta memoria. Por último, en los anexos que se encuentran en el CD adjunto se presentarán los listados de todos los programas desarrollados específicamente para los tres sistemas, con comentarios para su fácil comprensión. Además se incluyen los ficheros de circuito impreso de las tarjetas diseñadas, junto con algunos protocolos de transmisión de datos y otra documentación que no se ha incluido en esta memoria por brevedad.

En el CD adjunto se incluyen también una copia electrónica de esta memoria de tesis y una parte de la bibliografía utilizada para facilitar su consulta. El contenido completo del CD se detalla al final de esta memoria.

CAPÍTULO 2

MATERIALES Y MÉTODOS

El diseño y desarrollo de un dispositivo complejo como es una antena sísmica implica tareas diversas, desde el diseño de circuitos, fabricación de placas de circuito impreso y desarrollo de programas hasta las pruebas conjuntas de material y programas o montaje de los sistemas.

En este capítulo se presentarán brevemente algunos de los métodos y materiales utilizados. Se comenzará describiendo la metodología seguida en el diseño y fabricación de circuitos impresos (sección 2.1), para continuar con el procedimiento de desarrollo de los programas de los distintos controladores de las tres antenas (2.2). La tercera parte del capítulo se centrará en introducir algunos componentes y elementos que juegan un papel importante en los tres sistemas: el conversor analógico/digital AD7710 (sección 2.3.1), los microcontroladores de la familia PIC (2.3.2), las tarjetas de PC industrial (2.3.3), los receptores GPS (2.3.4) y los circuitos PLL (2.3.5). La introducción en esta memoria de un capítulo dedicado a materiales y métodos ofrecía la posibilidad de incluir en él toda la información técnica relevante relativa a todos los dispositivos y componentes que se iban a utilizar. Sin embargo, consideramos que el resultado habría sido una sucesión abrumadora de datos técnicos, muchos de los cuales no tiene sentido proporcionar si no se conocen antes las necesidades y características concretas de la aplicación en la que se van a integrar. Por esta razón se ha preferido plantear la sección 2.3 como una introducción descriptiva de los materiales utilizados, que permita situarlos dentro del contexto de los sistemas que se van a desarrollar. No debe extrañar, por tanto, que en los siguientes capítulos se vuelvan a incluir secciones de materiales, en las que se describirán con más detalle características relevantes para aplicaciones determinadas de alguno de los dispositivos ya presentados aquí.

2.1. Métodos de diseño de circuitos

El proceso seguido para el diseño y desarrollo de circuitos fue el siguiente:

1. Realización de un esquema de bloques del circuito.

2. Selección de los componentes y dispositivos a utilizar.

3. Realización de un esquema del circuito. Para ello se utilizó el programa de captura de esquemas *Orcad/SDTIII* funcionando sobre un ordenador personal. De los distintos módulos de que consta el programa, se utilizaron:
   - *DRAFT*, para la captura de esquemas en sí.
   - *LIBEDIT*, para editar y añadir componentes a las librerías originales del programa.
   - *CLEANUP*, para la detección de errores en los esquemas.
   - *PARTLIST*, para la generación de un listado de los componentes del circuito.

4. Realización de una placa de prueba. En algunos casos, antes de pasar a la realización del circuito impreso se probó una parte o la totalidad del circuito en placas perforadas, con el objeto de realizar una comprobación rápida de su funcionamiento.

5. Trazado del circuito impreso. Para ello se utilizaron los programas *TANGO* y

PROTEL EasyEdit. Las placas se diseñaron a una sola cara siempre que fue posible, y a doble cara en caso contrario. El posicionado de las pistas se realizó manualmente, partiendo de los circuitos integrados y procurando que hiciesen el mínimo recorrido.

6. Realización de un prototipo. Una vez diseñado y revisado el circuito impreso, se sacaron copias en papel vegetal y se insolaron, con luz solar, placas fotosensibilizadas de simple o doble cara. A continuación se taladraron las placas y se montaron y soldaron los componentes.

7. Verificación del correcto funcionamiento del prototipo. Para ello se utilizaron las técnicas y equipos usuales en laboratorios de electrónica (generadores de señal, fuentes de alimentación, osciloscopios, frecuencímetros,...).

8. En el caso de placas pequeñas y cuya complejidad y número no fuesen excesivos, todas las unidades se realizaron manualmente, siguiendo los pasos descritos en el punto 6. La fabricación de las placas más complejas o numerosas se encargó a una empresa especializada, siendo posteriormente montados y soldados los componentes en el IAG y – en el caso de la antena del Gran Sasso - en los laboratorios de electrónica de la Universidad de L'Aquila.

En la antena del Gran Sasso la práctica totalidad de las tarjetas (salvo las adquiridas comercialmente, como PCs o tarjetas de red) se diseñó y fabricó a través del proceso descrito. En las antenas del Vesubio e IAG éste se siguió para el desarrollo de tarjetas de prueba, prototipos y algunas de las tarjetas definitivas, pero otras fueron diseñadas y fabricadas por una empresa especializada a la que se le suministró el esquema eléctrico del circuito y los requerimientos físicos principales (dimensiones, tipo de conectores, asignación de pines,…).

La figura 2.1 muestra una pantalla del programa de diseño PCB *PROTEL EasyEdit* con una de las placas que se desarrollaron para la antena del Gran Sasso, concretamente la tarjeta de generación de la señal de reloj en las estaciones. Como puede verse, se trata de un diseño a doble cara. La superior contiene los componentes y un plano de tierra, mientras que las pistas de conexionado entre los componentes se sitúan en la cara inferior. Los componentes se dispusieron en la tarjeta tratando de minimizar la longitud necesaria de las líneas de conexión. Algunas de las celdas correspondientes a los componentes necesarios, tanto en éste como en el resto de los circuitos, no estaban incluidas en las librerías del programa (en la figura 2.1, los dos tipos de conectores y el *trimmer* C1), por lo que hubo que diseñarlas expresamente. Para ello se utilizó el editor de símbolos del programa, partiendo de las medidas físicas de la huella de cada componente. Como puede verse, las pistas se trazaron con una anchura considerable (prácticamente la máxima permitida con el ruteado propuesto), con el fin de facilitar la fabricación de prototipos mediante las técnicas descritas más arriba. Debido al gran número de tarjetas que había que utilizar (una por estación) el diseño que se muestra es uno de los que, una vez probados los prototipos, se mandó a una empresa especializada para la fabricación industrial en serie. Para ello fue necesario generar los ficheros necesarios en los formatos adecuados (Gerber y NC Drill).

Dado que la disposición de componentes y trazado de pistas no resultan necesarios para comprender el funcionamiento de los distintos circuitos, por brevedad no se han incluido en esta memoria los diseños de los circuitos impresos, sino sólo sus diagramas eléctricos. Los ficheros en formato gráfico de los circuitos impresos de las tres antenas pueden verse en los anexos I.B, II.B y III.B, incluidos en el CD adjunto.

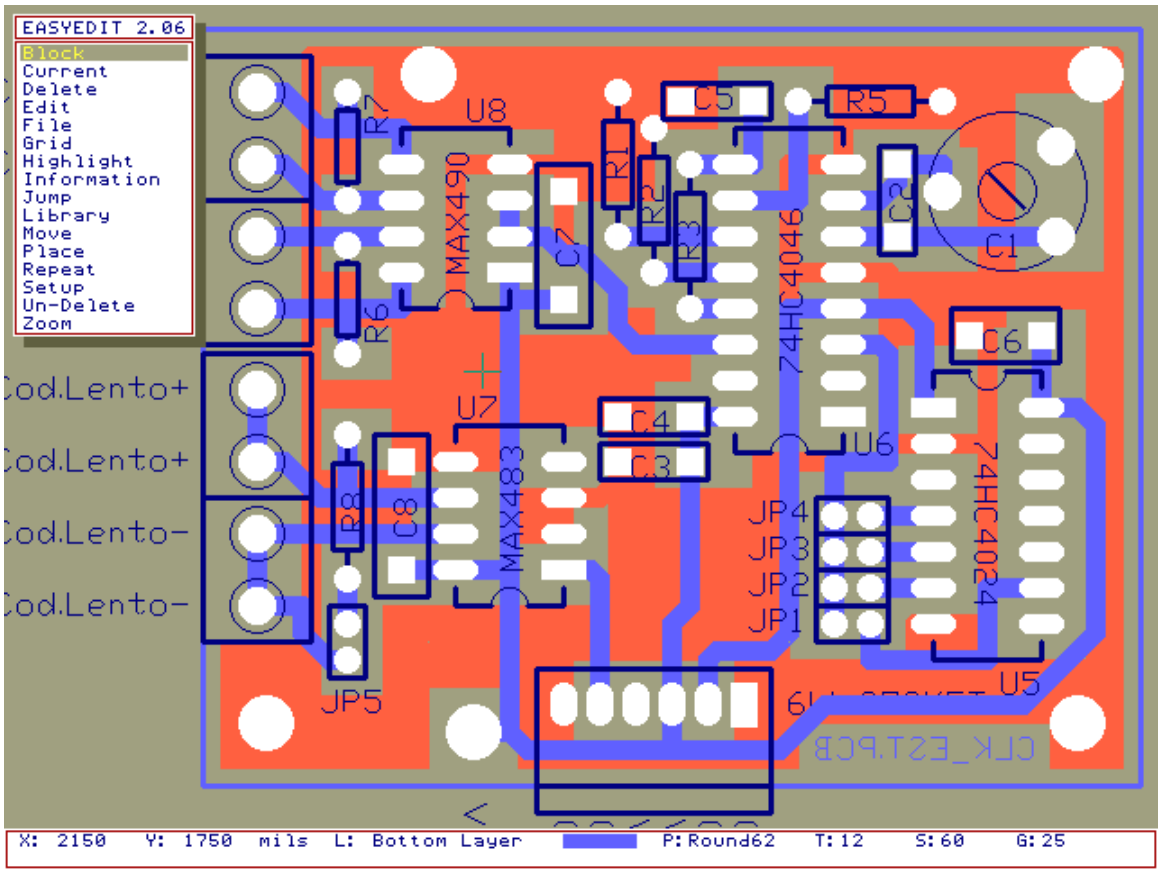


Figura 2.1. *Pantalla del programa de diseño de circuitos impresos Protel EasyEdit con una de las placas diseñadas para la antena del Gran Sasso. En la esquina inferior derecha del diseño aparece el nombre del fichero (CLK_EST.PCB), que se incluye en cada tarjeta con el objeto de facilitar su identificación para posteriores modificaciones. El rótulo está invertido porque se grabará en la cara inferior de la tarjeta, lo cual además ayuda a distinguir y orientar los fotolitos superior e inferior en el proceso de fabricación manual.*

2.2. Desarrollo de los programas

El funcionamiento de una antena sísmica requiere la operación coordinada de varios programas que controlen cada uno de los módulos de que consta el sistema. En los tres dispositivos que se describen se han utilizado tres tipos de controladores para realizar distintas funciones:

- Microcontroladores de la familia PIC para funciones de temporización en la antena del Gran Sasso y para temporización, adquisición de datos y control de las comunicaciones en los otros dos sistemas.

- PCs industriales para adquisición de datos y control de las comunicaciones en la antena del Gran Sasso y para el control central de los sistemas del Vesubio e IAG.

- Un PC convencional para el control central de la antena del Gran Sasso.

Los programas de los microcontroladores PIC se desarrollaron utilizando el entorno de programación y depuración MPLAB. Pese a que existen en el mercado varios compiladores cruzados de lenguajes de alto nivel (C, Pascal,…) con buenas características para este tipo de

microcontroladores, se prefirió programarlos directamente en lenguaje ensamblador, ya que el código que se genera de esta forma es más eficiente. La secuencia normal de programación de los PICs puede resumirse en los siguientes pasos:

1. Realización de un diagrama de flujo del programa.

2. Escritura, compilación y depuración del programa en un PC convencional desde el entorno de desarrollo MPLAB, que proporciona las herramientas necesarias para la depuración y simulación de programas de microcontroladores de la familia PIC con una interfaz gráfica tipo *Windows*.

3. Grabación del programa depurado en el microcontrolador y prueba del mismo en la placa de circuito impreso. Para la escritura del programa en memoria se utilizó la grabadora suministrada en el kit de desarrollo *PICStart+*. En el caso del PIC modelo 16F84, que tiene memoria FLASH, se puede llevar a cabo la escritura del programa sin necesidad de borrar la memoria. No ocurre lo mismo en los modelos con memoria EPROM regrabable (PIC16C73-JW y PIC16C74-JW), en los que antes de grabar una nueva versión del programa es necesario borrar la memoria mediante exposición a luz ultravioleta.

4. Los dos pasos anteriores se repitieron hasta conseguir la versión definitiva del programa. En los microcontroladores con memoria EPROM existen versiones grabables una sola vez (OTP, *One Time Programmable*), más baratas, pensadas para reducir costes en producciones industriales. En nuestro caso no se usaron, ya que el número de unidades que se tenían que grabar no era suficiente como para suponer un ahorro apreciable y las versiones regrabables ofrecen la ventaja, frente a las OTP, de ser susceptibles a futuras modificaciones.

En la antena del Gran Sasso los PCs industriales forman nodos intermedios de una red y no necesitan una interacción directa por parte del usuario. Por esta razón se eligió como sistema operativo el MSDOS, que consume menos recursos que *Windows* y resulta más adecuado para la gestión de procesos de entrada/salida por puertos paralelo y serie y para la programación de funciones de temporización y sincronización. En el caso de las antenas del Vesubio y del IAG sí es necesaria una interacción con los programas del PC por parte del usuario, a la hora de volcar datos o cambiar parámetros de operación. Sin embargo, también en estos casos se prefirió la robustez del MSDOS, aunque ello implicara una interfaz de usuario menos atractiva e intuitiva que la que se podría haber realizado en un entorno *Windows*.

Todos los programas para los PCs industriales se realizaron en lenguaje C, utilizando el compilador de *Borland C++* versión 3.1. El proceso de desarrollo de estos programas consta de los siguientes pasos:

1. Realización de un diagrama de flujo.

2. Compilación y depuración de cada uno de los programas en un PC convencional.

3. Prueba simultánea de las versiones preliminares en los equipos definitivos. En la antena del Gran Sasso el desarrollo de los programas de las estaciones y PCs nodales se llevó a cabo simultáneamente, dado que ambos tienen que funcionar de un modo coordinado. En esta antena el arranque de los PCs industriales se realiza desde discos de estado sólido (memorias FLASH o EPROM). Durante la fase de desarrollo de los programas se utilizaron memorias FLASH, por su mayor facilidad y velocidad de

programación. Las pruebas se realizaron primero utilizando una sola estación conectada vía serie a un PC nodal, conectando éste a su vez a un servidor a través de una red *ethernet 10Base2* (cable coaxial). Posteriormente se incrementó el número de estaciones, hasta llegar al montaje de una línea serie completa con cuatro estaciones y un PC nodal. Una vez depurados totalmente ambos programas se grabaron las versiones definitivas, junto a los programas de arranque y configuración, en memorias EPROM que constituyen el medio definitivo de almacenamiento.

Para el programa de control de la antena del Gran Sasso se utilizó el compilador *Borland C++ Builder* versión 3 en lugar del *Borland C++* bajo MSDOS, puesto que se prefería dotarlo de una interfaz de usuario tipo *Windows*. Dado que la plataforma donde debía correr era un PC convencional, el desarrollo de este programa fue más sencillo que en los otros casos. La escritura y depuración se llevó a cabo en un PC compatible, analizando los valores de las variables de entrada/salida que se usarían para interactuar con el resto de los módulos del sistema. Una vez depurado el programa se realizaron pruebas coordinadas con los programas de los PCs nodales y estaciones.

2.3. Materiales

2.3.1. El conversor analógico/digital AD7710

El núcleo de los módulos de adquisición de datos de los tres sistemas que se describen es el conversor analógico/digital AD7710, de la empresa *Analog Devices* (figura 2.2). En el momento de iniciar el primero de los proyectos que se presentan en esta memoria, la disponibilidad en el mercado de conversores de alta resolución y bajo coste era escasa. En la actualidad la situación ha cambiado y existen varios modelos de características iguales o mejores que el AD7710, en particular en lo referente a la resolución efectiva. En el capítulo 7 se presentarán algunos de estos conversores y se discutirá el incremento de prestaciones que implicaría su uso en los dispositivos que se han desarrollado.

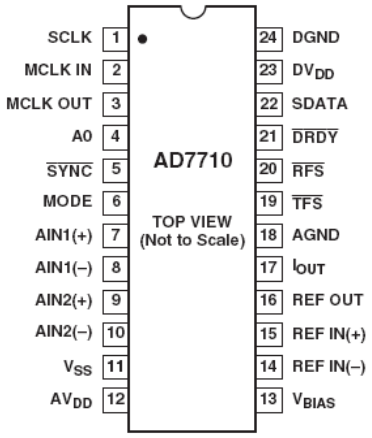


Figura 2.2. *Disposición y nombre de los pines en el AD7710 (de Analog Devices, 1991).*

Las características principales del AD7710 son (Analog Devices, 1991):

- Técnica de conversión sigma-delta.
- 24 bits de resolución nominal sin pérdida de códigos.
- Salida de datos en serie.
- Compacto (Encapsulado cerámico de veinticuatro patillas).
- Bajo coste.

La señal analógica de entrada se aplica a un módulo amplificador de ganancia programable basado en un modulador analógico, cuya salida es procesada por un filtro digital integrado en el chip. La frecuencia del primer nodo de este filtro puede ser programada a través del registro de control, permitiendo así el ajuste de la frecuencia de corte y tiempo de establecimiento del mismo.

Hay varias características del AD7710 que merece la pena destacar, ya que resultan muy interesantes para nuestras aplicaciones:

- Es ideal para el uso en sistemas basados en microcontroladores o DSPs, ya que permite la configuración de sus parámetros de funcionamiento (canal de entrada, ganancia, polaridad de la señal, frecuencia de muestreo,...) mediante programa, a través de su puerto serie bidireccional.

- El usuario tiene completo control sobre el procedimiento de calibración, ya que el AD7710 permite la posibilidad de lectura y escritura en el registro de calibración, ofreciendo distintas opciones para la misma. Los efectos de deriva con la temperatura pueden corregirse con la opción de auto-calibración, que elimina los errores de cero y de fondo total de escala.

- El módulo de ganancia programable permite al AD7710 tomar las señales de entrada directamente de los sensores, haciendo innecesario cualquier otro sistema para el acondicionamiento de la señal.

- Al ser un dispositivo de tecnología CMOS, la disipación de potencia es bastante baja (unos 25 mW). Además ofrece la posibilidad - también programable a través del registro de control- de funcionamiento en modo de bajo consumo, que reduce éste a sólo 7mW en estado de espera. Como se verá en los siguientes capítulos, el consumo no es un factor condicionante en la antena del Gran Sasso, pero sí lo es en dispositivos portátiles como la antena del Vesubio y las del IAG.

A continuación se ofrece una breve explicación del funcionamiento de este conversor, centrándonos en los aspectos que deberán tenerse en cuenta en el diseño de los subsistemas que controlen su operación. Para una descripción más detallada, pueden consultarse las hojas de características (Analog Devices, 1991).

2.3.1.1. Descripción de las funciones de los pines

El AD7710 se presenta en varios encapsulados para montaje pasante o superficial (PDIP, CERDIP, SOIC). La disposición de los pines es la misma en todos ellos y se muestra en la figura 2.2. A continuación se describen las funciones de las señales más importantes y la configuración que se le asignará en nuestras aplicaciones:

Nombre	*Función*
MCLKIN MCLK OUT	El reloj maestro del sistema puede generarse mediante un cristal, que debe conectarse entre los pines MCLKIN y MCLK OUT. Otra opción, que será la elegida en nuestras aplicaciones, es utilizar una señal externa, que debe conectarse a MCLKIN dejando MCLK OUT desconectada. La frecuencia nominal para este reloj maestro es de 10 MHz. La variación de la frecuencia del reloj maestro afecta a todos los parámetros relacionados con la velocidad de operación del dispositivo, entre ellos la frecuencia de muestreo. Como se verá en los próximos capítulos, esto puede aprovecharse para conseguir la sincronización del muestreo en distintos conversores.

MODE	Entrada lógica que se utiliza para seleccionar el modo de operación de entre dos posibilidades: reloj interno (*self clocking*) o externo (*external clocking*). En nuestras aplicaciones se utilizará el modo de reloj externo, para lo cual este pin debe estar a nivel bajo.
SCLK	En el modo de reloj externo este pin es una entrada lógica, por la que el usuario debe proporcionar una secuencia de pulsos para temporizar las operaciones de escritura de parámetros y lectura de datos.
AIN1(+) AIN1(-) AIN2(+) AIN2(-)	Entradas analógicas diferenciales. El AD7710 tiene dos canales de adquisición, pero, teniendo en cuenta que la técnica de muestreo sigma-delta exige un seguimiento continuo de la señal, los dos canales sólo se pueden usar a la vez en aplicaciones de muy baja frecuencia de muestreo. En nuestro caso se utilizará siempre el canal 1.
SDATA	Entrada/salida de datos serie. En función del estado del pin A0, se accederá a uno de los tres registros del AD7710 (control, calibración o datos). Es posible leer el contenido de los tres registros, pero la escritura sólo está permitida en los de control y calibración.
A0	Entrada lógica que selecciona el destino de las operaciones de lectura o escritura. Con A0 a nivel bajo, el acceso es al registro de control. Con A0 en alto se accede al registro de datos o al de calibración.
/TFS /RFS	/TFS (*Transmit Frame Sinchronization*) y /RFS (*Receive Frame Synchronization*) son entradas lógicas que se activan en nivel bajo y que se usan para escribir y leer, respectivamente, datos en los registros del AD7710.
/DRDY	Salida lógica que se activa en nivel bajo. El flanco descendente indica que en el registro de datos hay un nuevo dato listo para ser leído. La señal no se desactivará (es decir, no retornará al nivel alto) hasta que se haya completado la lectura del dato. Esta señal tendrá una gran importancia en nuestras aplicaciones, puesto que los programas la tomarán como referencia de que los conversores han completado el proceso de muestreo. Como se verá más adelante, el objetivo del proceso de sincronización será conseguir que las señales DRDY de todos los módulos de adquisición se activen simultáneamente.
/SYNC	Entrada lógica que permite la sincronización de los filtros digitales cuando se utilizan varios conversores AD7710.
AV_{DD} AGND V_{SS}	Alimentación analógica positiva (AV_{DD}), negativa (V_{SS}) y tierra (AGND). AV_{DD} debe estar comprendida entre +5V y +10V, y V_{SS} entre 0 y -5V. En nuestras aplicaciones se usará alimentación simétrica (AV_{DD} = +5V y V_{SS} = -5V), para permitir la conversión de señales negativas.
DV_{DD} DGND	Alimentación digital positiva (DV_{DD}) y tierra (DGND). En nuestras aplicaciones se utilizará la misma fuente de +5V para AV_{DD} y DV_{DD}, pero para minimizar el ruido inducido desde la alimentación analógica a la digital, los circuitos impresos se rutearán de forma que sólo exista un punto común a ambas. Además se usarán condensadores de desacoplo independientes para cada una de las alimentaciones en todos los conversores de las tarjetas de adquisición, situándolos lo más próximos posible a los integrados.

REF IN(+) REF IN(-) REF OUT	La tensión entre los pines REF IN(+) y REF IN(-) se toma como referencia para la conversión, y puede tener valores entre 1 y 5V. Esta tensión puede suministrarse mediante un integrado específico (por ejemplo, el AD580 o el AD680) o desde el pin REF OUT, que proporciona una referencia de 2.5V compensada en temperatura. Para utilizar la referencia de REF OUT debe conectarse este pin a REF IN(+) y REF IN(-) a AGND. Esta será la opción elegida en nuestras aplicaciones. Teniendo en cuenta que la alimentación del integrado será simétrica, con esta configuración el fondo de escala de la entrada analógica será de ± 2.5V.
V_{BIAS}	Tensión de polarización de entrada. El valor ideal es el punto medio entre AV_{DD} y V_{SS}, por lo que en nuestro caso se fijará a AGND.

2.3.1.2. Registro de control

Los parámetros de operación del AD7710 se programan mediante la escritura de una palabra de 24 bits en el registro de control. Esta palabra de control, cuyo formato se muestra en la figura 2.3, contiene toda la información necesaria para la configuración del conversor, como el modo de operación, los valores de la ganancia, la frecuencia de muestreo, la resolución nominal o el canal analógico usado.

MSB

MD2	MD1	MD0	G2	G1	G0	CH	PD	WL	IO	BO	B/U
FS11	FS10	FS9	FS8	FS7	FS6	FS5	FS4	FS3	FS2	FS1	FS0

LSB

Figura 2.3. *Formato de la palabra de control que se utiliza para la configuración del AD7710. Las abreviaturas corresponden a:*

MD2, MD1, MD0: modo de operación
G2, G1, G0: ganancia
CH: canal analógico
PD: selección de modo de bajo consumo
WL: longitud de palabra de datos (16 o 24 bits)
IO: habilitar/deshabilitar corriente de compensación
*BO: habilitar/deshabilitar corriente de comprobación de sensores (*Burn-Out current*)*
B/U: selección de modo bipolar o unipolar
FS11,…, FS0: selección de la frecuencia del primer nodo del filtro digital, que determina la frecuencia de muestreo.

Como puede verse en la figura 2.3, los bits más significativos de la palabra de control son MD2, MD1 y MD0, cuya función es elegir el modo de operación. En modo normal (MD2 = MDl = MD0 = 0) el conversor comienza a muestrear directamente con los valores introducidos en el registro de control para el resto de parámetros. En los otros modos de operación el dispositivo comienza realizando calibraciones de cero y de fondo total de escala, para minimizar los errores de offset y de ganancia. Los distintos modos difieren en los valores usados para realizar estas calibraciones. En nuestras aplicaciones se inicializarán los conversores en el modo de auto calibración (*self-calibration*), en el que la calibración de cero se realiza con las dos entradas cortocircuitadas (es decir, AIN(+) = AIN(-) = V_{BIAS}) y para la calibración de fondo total de escala se toma como referencia la tensión V_{REF}. En este modo el

conversor retorna automáticamente a modo normal una vez ha realizado las calibraciones, por lo que no es necesario reprogramarlo.

El AD7710 ofrece la posibilidad, a través del modo de calibración en segundo plano (*background calibration*), de realizar calibraciones continuas que intercala con el proceso de muestreo. De esta forma se minimizan los errores debido a variaciones en la tensión de alimentación y se eliminan los efectos de deriva de temperatura o de deriva temporal de los errores de cero y fondo total de escala. Sin embargo, en este modo se produce una reducción en un factor 6 de la frecuencia de muestreo, por lo que no es recomendable para nuestras aplicaciones.

La ganancia se introduce mediante los bits G2 a G0, y puede tomar valores correspondientes a potencias de dos, entre 1 y 128. Como se ha comentado anteriormente, en los conversores que utilizan la técnica sigma-delta la ganancia y la frecuencia de muestreo influyen en el valor de la resolución efectiva, por lo que deben seleccionarse con precaución. La tabla 2.1 muestra la relación entre estos tres parámetros (resolución efectiva, frecuencia de muestreo y ganancia) para el AD7710.

First Notch of Filter and O/P Data Rate	−3 dB Frequency	Effective Resolution* (Bits)							
		Gain of 1	Gain of 2	Gain of 4	Gain of 8	Gain of 16	Gain of 32	Gain of 64	Gain of 128
10 Hz	2.62 Hz	22.5	21.5	21.5	21	20.5	19.5	18.5	17.5
25 Hz	6.55 Hz	21.5	21	21	20	19.5	18.5	17.5	16.5
30 Hz	7.86 Hz	21	21	20.5	20	19.5	18.5	17.5	16.5
50 Hz	13.1 Hz	20	20	20	19.5	19	18.5	17.5	16.5
60 Hz	15.72 Hz	20	20	20	19.5	19	18	17	16
100 Hz	26.2 Hz	18.5	18.5	18.5	18.5	18	17.5	17	16
250 Hz	65.5 Hz	15	15	15.5	15.5	15.5	15.5	15	14.5
500 Hz	131 Hz	13	13	13	13	13	12.5	12.5	12.5
1 kHz	262 Hz	10.5	10.5	11	11	11	10.5	10	10

Tabla 2.1. *Relación entre la resolución efectiva, la ganancia y la frecuencia de muestreo – determinada por el primer nodo del filtro digital – en el AD7710 (de Analog Devices, 1991).*

Los bits CH, WL y B/U seleccionan, respectivamente, el canal analógico de entrada, la longitud de palabra de datos o resolución nominal (16 o 24 bits) y la polaridad de la señal (unipolar o bipolar). En nuestras aplicaciones se utilizará siempre el canal 1 en modo bipolar y con resolución de 24 bits.

El bit PD permite habilitar el modo de bajo consumo, en el que éste se reduce notablemente mientras el conversor está en estado de espera (*standby*). Esta posibilidad resulta útil sobre todo en aplicaciones de baja frecuencia de muestreo.

Los bits IO y BO habilitan la corriente de compensación y la corriente de comprobación de sensores (*burn-out current*), características pensadas para su uso con termopares y otros sensores, y que no se utilizarán en nuestras aplicaciones.

Por último, los bits FS11 a FS0 determinan la frecuencia del primer nodo del filtro digital, que es equivalente a la frecuencia de muestreo, y cuyo valor se obtiene de la siguiente expresión:

$$f_m \equiv \frac{f_{CLKIN}}{512 \times código} \qquad [2.1]$$

siendo f_m la frecuencia de muestreo, f_{CLKIN} la frecuencia del reloj maestro y *código* el valor decimal correspondiente al código binario introducido mediante los bits FS0 a FS11. Como se verá en el próximo capítulo, este procedimiento de programación de la frecuencia de muestreo impondrá ciertas condiciones al valor de la frecuencia de reloj maestro que habrá que utilizar en nuestras aplicaciones.

2.3.1.3. Interfaz digital

El puerto de comunicación serie del AD7710 proporciona una interfaz flexible entre el conversor y sistemas de control basados en microprocesadores, microcontroladores o procesadores digitales de señales.

Hay dos modos de operación posibles, optimizados para diferentes tipos de interfaces en los que el AD7710 puede actuar bien como maestro (modo de reloj interno o *self-clocking*) o como esclavo (modo de reloj externo o *external clocking*). En el primer caso es el propio conversor el que proporciona la secuencia de pulsos serie para sincronizar la transmisión. En el modo de reloj externo, que es el que se utilizará en nuestras aplicaciones, la sincronización corre a cargo de un controlador externo que suministra la secuencia de pulsos serie.

La figura 2.4 muestra un diagrama de tiempos de la secuencia que debe seguirse en la operación de lectura del registro de datos en el modo de reloj externo. La señal /DRDY es la que indica que el proceso de muestreo ha concluido y que existe un nuevo dato, listo para ser leído, en el registro de datos. Cuando esta señal se activa el microcontrolador debe poner A0 a nivel alto, para seleccionar el registro de datos, y /RFS a nivel bajo para iniciar el proceso de lectura. El flanco descendente de /RFS pone el bit más significativo de la palabra de datos en la línea serie. El resto de los bits de datos se colocan en la salida serie mediante pulsos de la línea SCLK. El flanco ascendente del último pulso pone el bit menos significativo de la palabra de datos en la línea serie, y el siguiente flanco descendente provoca la desactivación de la señal /DRDY. El flanco ascendente de /DRDY a su vez desconecta la salida de datos serie y deja la línea en alta impedancia.

La operación de escritura de la palabra de configuración en el registro de control se realiza de modo similar. Las diferencias principales son que en este caso la señal /DRDY no interviene en el proceso, que la señal que activa la escritura no es /RFS sino /TFS y que, como es lógico, la línea SDATA no se comporta como salida sino como entrada de datos serie.

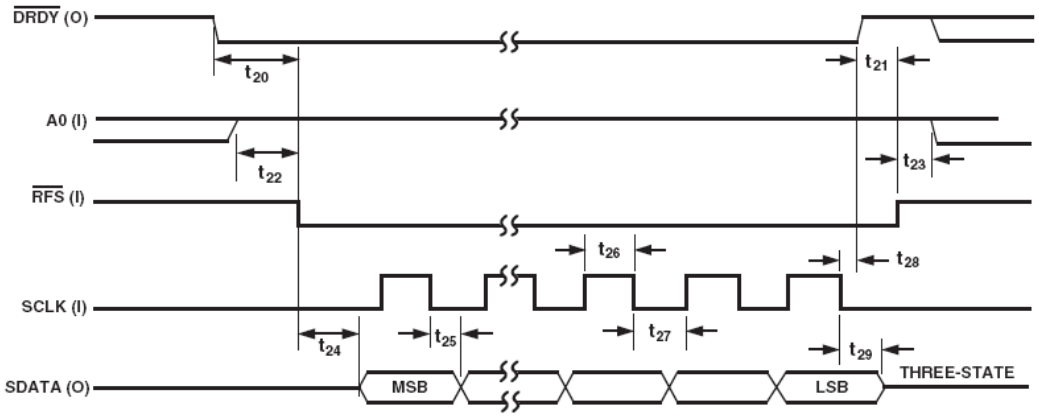


Figura 2.4. *Diagrama de tiempos para la operación de lectura del registro de datos en el modo de reloj externo (de Analog Devices, 1991).*

2.3.2. Los microcontroladores de la familia PIC

La familia de microcontroladores PIC, de la empresa *Arizona Microchip Technologies*, está constituida por una gran cantidad de modelos de microcontroladores que se pueden agrupar en tres gamas (Campos et al, 1998). En la actualidad esta familia copa una parte importante del mercado mundial en el campo del diseño de aplicaciones para sistemas embebidos. Su éxito se basa en una combinación de factores (velocidad, precio, facilidad de uso, información abundante, herramientas de apoyo,…) que en conjunto ofrecen una imagen de sencillez y utilidad que hace de estos microcontroladores una opción muy atractiva para los diseñadores. Si bien otras familias pueden ser más eficaces en aplicaciones concretas, en las más usuales la elección de una versión adecuada de PIC puede ofrecer una solución óptima.

2.3.2.1. Características

Las características más relevantes de los PIC son:

1. La arquitectura del procesador sigue el modelo Harvard. La CPU se conecta de forma independiente y a través de buses distintos con la memoria de instrucciones y con la de datos, lo cual permite accesos simultáneos a las dos memorias. La utilización de buses distintos posibilita, además, que las instrucciones tengan una longitud distinta a los 8 bits de las palabras de datos.

2. Se aplica la técnica de segmentación (*pipeline*) en la ejecución de las instrucciones. La segmentación permite al procesador realizar al mismo tiempo la ejecución de una instrucción y la búsqueda del código de la siguiente. De esta forma cada instrucción se puede ejecutar en un ciclo de instrucción (equivalente a cuatro ciclos de reloj), excepto las instrucciones de salto, que ocupan dos ciclos debido a que hasta que no se completa la bifurcación no es posible conocer la dirección de la siguiente instrucción. Esta característica facilita el cálculo del tiempo de ejecución de los programas, lo cual permite realizar funciones de temporización de forma precisa y sencilla (como ejemplo, ver un bucle de retardo en la figura 2.5). La arquitectura Harvard y la técnica de segmentación son los principales recursos en los que se sostiene el elevado rendimiento de esta familia de microprocesadores, mejorando dos características esenciales: la velocidad de ejecución y la eficiencia en la compactación del código.

3. El formato de todas las instrucciones tiene la misma longitud. Todas las instrucciones de los microcontroladores de la gama baja tiene una longitud de 12 bits. Las de la gama media tienen 14 bits y las de la gama alta más de 14. Esta característica es muy ventajosa en la optimización de la memoria de instrucciones y facilita enormemente el desarrollo de ensambladores y compiladores.

4. Procesador RISC (*Reduced Instruction Set Computer*). El juego de instrucciones reducido facilita el aprendizaje y reduce notablemente el tiempo de desarrollo de un proyecto. Los modelos de la gama baja disponen de un repertorio de 33 instrucciones, los de la gama media de 35 y los de la alta de unas 60.

5. Arquitectura basada en un banco de registros. Todos los objetos del sistema (puertos de entrada/salida, temporizadores, posiciones de memoria,…) están implementados físicamente como registros.

6. Diversidad de modelos de microcontroladores con prestaciones y recursos diferentes. La gran variedad de modelos de PIC permite que el usuario pueda seleccionar el más conveniente para su proyecto. Por otra parte, la arquitectura interna y el repertorio de instrucciones comunes en cada gama hace que sea sencillo cambiar de modelo cuando se ha aprendido a usar uno de ellos.

7. Herramientas de soporte potentes y económicas. La empresa Microchip ha creado una serie de herramientas de ayuda al desarrollo del *hardware* y *software* de los proyectos de aplicación. Entre ellas pueden citarse el ensamblador MPASM, el simulador software MPSIM, el compilador de lenguaje C (MP-C), el programador universal PRO MATE, el emulador universal PIC MASTER, la herramienta para desarrollo de lógica difusa FUZZY TECH-MP o el entorno de desarrollo integrado MPLAB. Además, debido al uso extendido de los PICs, hay otras muchas empresas y particulares que se han dedicado al desarrollo de herramientas de ayuda. Así, es posible adquirir comercialmente e incluso descargar gratuitamente gran número de programadores, simuladores *software*, emuladores en tiempo real, ensambladores, compiladores C, intérpretes y compiladores BASIC.

8. Documentación abundante. Desde la página web de la empresa Microchip[13] pueden descargarse, además de las hojas de características de todos sus modelos de microcontroladores, numerosos manuales de referencia, ejemplos y notas de aplicación ordenados según distintos criterios orientados a facilitar la búsqueda (tipo de función que se desea implementar, modelo de microcontrolador, tipo de aplicación, módulos internos que se van a utilizar,…). Por otra parte, el uso generalizado de este tipo de dispositivos en los últimos años hace que haya muchas otras empresas y usuarios particulares que ofrecen abundante documentación técnica en publicaciones especializadas e Internet.

2.3.2.2. Las tres gamas

La filosofía que la empresa fabricante ha seguido para los PICs ha sido la de ofrecer un gran catálogo de modelos cuyas características se hacen progresivamente más potentes. Así, hay disponibles microcontroladores sencillos y baratos concebidos para aplicaciones simples y otros complejos y más costosos para las que requieren mayores prestaciones, de modo que la familia completa es capaz de cubrir las necesidades de un amplio abanico de proyectos. Según esta filosofía, una de las decisiones más importantes de un proyecto basado en este tipo de microcontroladores es la selección del modelo adecuado, que debe reunir las características más acordes a las funciones que se le asignen, sin sobredimensionar sus prestaciones. Algunos parámetros que deben tenerse en cuenta a la hora de tomar esta decisión son el tipo y capacidad de las memorias (de datos y de programa), el número de líneas de entrada/salida (E/S), funciones auxiliares disponibles y tipo de encapsulado. Todas las versiones están construidas alrededor de una arquitectura común, un repertorio mínimo de instrucciones y un conjunto de opciones muy interesantes, como el bajo consumo y el amplio margen del voltaje de alimentación.

Los modelos de PIC actualmente existentes en el mercado pueden agruparse en tres gamas:

Gama baja.- Actualmente existen dos tendencias en la fabricación de microcontroladores: la arquitectura cerrada y la arquitectura abierta. Los modelos de arquitectura cerrada se

[13] www.microchip.com

construyen con una determinada CPU, cierta capacidad de memoria de datos e instrucciones, un número de líneas de E/S y un conjunto de recursos auxiliares muy concreto, por lo que no admiten variaciones ni ampliaciones. Los modelos de arquitectura abierta se caracterizan porque, además de disponer de una estructura interna determinada, pueden emplear sus líneas de E/S para sacar al exterior los buses de datos, direcciones y control, con lo que se posibilita la ampliación de la memoria y las líneas de E/S con circuitos integrados externos.

Los microcontroladores de la familia PIC pertenecientes a la gama baja son de arquitectura cerrada y se caracterizan por tener poca memoria de programa (entre 512 y 2 kpalabras de 12 bits) y de datos (entre 25 y 73 bytes), un solo temporizador (TMR0), un repertorio de 33 instrucciones y un número de líneas de E/S comprendido entre 12 y 20. El voltaje de alimentación admite un valor muy flexible comprendido entre 2 y 6.25V, lo cual posibilita – teniendo en cuenta su bajo consumo – el funcionamiento mediante pilas corrientes.

Presentan algunas características comunes a las gamas superiores, como un temporizador tipo perro guardián (*watchdog*), líneas de E/S de alta corriente, posibilidad de protección del código o modo de reposo para ahorrar consumo. Dos restricciones importantes de los modelos de esta gama son que no admiten interrupciones y que la pila sólo dispone de dos niveles, lo que implica que no se pueden encadenar más de dos subrutinas.

Gama media.- Los modelos de la gama media son también microcontroladores de arquitectura cerrada. A las prestaciones de los modelos de la gama baja les añaden otras como la gestión de interrupciones, convertidores A/D, comparadores de magnitudes analógicas, puertos serie y diversos temporizadores y módulos asociados a ellos. Hay modelos con memoria FLASH, que hace innecesaria la exposición a rayos ultravioleta para el borrado de memoria que precisan las memorias EPROM tradicionales. Algunos modelos disponen de versiones grabables una sola vez (OTP), que resultan mucho más económicas para la grabación de series numerosas.

Gama alta.- Los dispositivos de la gama alta son microcontroladores de arquitectura abierta, que por tanto pueden expansionarse al permitir el acceso a los buses de datos, direcciones y control. Así se pueden configurar sistemas similares a los que utilizan los microprocesadores convencionales, siendo capaces de ampliar la configuración interna del PIC añadiendo nuevos dispositivos de memoria y E/S externas. Esta capacidad de expansión hace que estos dispositivos tengan un elevado número de pines, entre 40 y 44. Admiten interrupciones, poseen puerto serie, varios temporizadores y mayores capacidades de memoria, que alcanza las 8 kpalabras de instrucciones y 454 bytes de datos.

En las tres antenas sísmicas cuyo diseño se describe en esta memoria se han utilizado tres modelos distintos de PIC, todos pertenecientes a la gama media. El PIC 16C74 se utilizó en funciones de temporización en las antenas del Gran Sasso y del Vesubio, mientras que los modelos 16F84 y 16C73 se utilizaron, respectivamente, en funciones de control del conversor A/D y en funciones de temporización en las antenas del Vesubio y del IAG. En todos los casos los modelos utilizados pueden sustituirse por versiones compatibles más modernas (normalmente con el mismo nombre y sufijos ‘A’ o ‘B’). En los capítulos relativos al diseño de cada una de las antenas se describirán con más detalle algunas de las características de estos modelos y los módulos y funciones auxiliares que se utilizaron en la programación.

```
Retardo1ms
            MOVLW     .99
            MOVWF     Contador
BucleRet1ms
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            DECFSZ    Contador, F
            GOTO      BucleRet1ms
            NOP
            NOP
            NOP
            NOP
            NOP
            RETURN
```

Figura 2.5. *Programación de un bucle de retardo en código ensamblador de los microcontroladores PIC. Las instrucciones NOP no realizan ninguna acción, tan solo consumen tiempo. Todas las instrucciones consumen un ciclo de instrucción, salvo las de salto, que consumen dos ciclos. En la figura, las instrucciones de salto son DECFSZ, GOTO y RETURN. Hay que tener en cuenta también la instrucción CALL del programa principal, mediante la cual se accedería al bucle de retardo. Cada ciclo de instrucción equivale a cuatro ciclos de reloj, por lo que con un cristal de 4MHz el ciclo de instrucción vale:*

$$t_{ciclo} = 4 \times (1/4MHz) = 1\ \mu s/ciclo$$

Para crear un retardo de 1ms harán falta, por tanto:

$$1ms / t_{ciclo} = 1ms / (1\mu s/ciclo) = 1000\ ciclos$$

En el código de la figura se genera un bucle de 10 ciclos, que se ejecuta 99 veces. Cuando sale del último ciclo se ejecuta el número necesario de instrucciones NOP para que el número total de ciclos de instrucción sea exactamente el necesario. El número total de ciclos se puede calcular como:
4 ciclos (instrucción CALL + asignación de valor a la variable 'Contador') +
98x10 ciclos (bucles en los que 'Contador' no es cero y que vuelven a 'BucleRet1ms') +
9 ciclos (último bucle, en el que 'Contador' se hace cero) +
7 ciclos (5 NOPs finales + 2 ciclos del RETURN)
Total: 1000 ciclos

2.3.3. Las tarjetas de PC industrial

En la antena del Gran Sasso la mayor carga de trabajo del sistema se asignó a PCs de tipo industrial, que realizan casi todas las funciones relativas a la adquisición y transmisión de datos. En las otras dos antenas la mayoría de estas funciones son desempeñadas por microcontroladores PIC, mientras que los PCs industriales se ocupan básicamente de la centralización y grabación de los datos y de la gestión de la interfaz de usuario.

En el caso de la antena del Gran Sasso, los PCs industriales elegidos fueron modelos *AR-B1474*, de la empresa *Acrosser*[14]. Las principales características de estas placas son:

- Procesador Intel 486 o compatible, con distintas opciones de velocidad.
- Memoria RAM configurable desde 1 MB a 32 MB.
- Controladores y puertos:
  - Un controlador HDD.
  - Un controlador de disco flexible (3.5" o 5.25").
  - Dos puertos serie con búffer de entrada-salida (UART 16C550).
  - Un puerto paralelo bidireccional.
  - Dos controladores de interrupciones compatibles con el 8259.
- Temporizador tipo perro guardián (*watchdog*) con temporización programable.
- Opcionalmente se ofrece con zócalos para disco de estado sólido (EPROM, SRAM o FLASH EPROM).

De entre las opciones posibles se eligieron:

- Procesador 486DX4 a 100 MHz para las estaciones y a 133 MHz para los PCs nodales.
- Memoria de 4 MB. Aunque no es necesario más de 1MB (ya que el programa de adquisición no gestiona la memoria extendida) la disponibilidad en el mercado de los módulos de 4 MB en el momento de su adquisición era mayor que la de los de 1 MB, mientras que el precio era parecido.
- Se eligieron, como medios de almacenamiento de ficheros y programas, discos de estado sólido en lugar de discos duros. Como se verá en el capítulo correspondiente a la antena del Gran Sasso, las tarjetas de PC no necesitan grabar en disco, ya que los datos que adquieren son transmitidos, primero por comunicación serie y luego por red ethernet, a los servidores. La única función del disco es la de almacenar los programas de adquisición y los ficheros de arranque y configuración de los PCs industriales, por lo que resulta más práctico y rentable usar discos de estado sólido que discos duros. Al no tener partes móviles el comportamiento de los discos de estado sólido es mucho más fiable en entornos hostiles que el de los discos duros.

De entre las distintas opciones disponibles para los discos de estado sólido (EPROM, SRAM y FLASH EPROM) se utilizaron EPROM para grabar los programas definitivos, por su menor precio. Durante el desarrollo de los programas se utilizaron pastillas de FLASH EPROM, por su mayor facilidad y rapidez de borrado y regrabado.

Los continuos y rápidos avances en el campo de la informática hicieron posible la elección, para las antenas del Vesubio y del IAG (cuyo diseño fue posterior al de la antena del Gran Sasso), de un modelo de placa de PC industrial con prestaciones sensiblemente superiores a las del modelo descrito, a la par que con un consumo y tamaño menores y un

[14] www.acrosser.com

precio parecido. El modelo utilizado fue el *Cool RoadRunner II*, de la empresa *Lippert*[15], cuyas características principales son:

- Procesador Geode GX1 (compatible con Pentium) a 200 o 300 MHz.
- Memoria SDRAM de 64 MB, ampliable hasta 512 MB.
- Memoria caché de nivel 1 de 16 kB. Es posible también instalar memoria caché de nivel 2.
- Controladores y puertos:
  - Un controlador EIDE.
  - Un controlador de vídeo SVGA.
  - Un controlador de disco flexible.
  - Dos puertos serie.
  - Un puerto paralelo bidireccional.
  - Dos puertos PS/2 (para teclado y ratón).
  - Dos puertos USB.
  - Un puerto de infrarrojos IrDA.
  - Controlador de red *ethernet 10/100 BaseT* Intel 82559.
  - Controlador de sonido con entrada y salida de línea y entrada de micrófono.
- Dos temporizadores *watchdog* independientes.
- Zócalo para memoria Compact Flash.
- Posibilidad de expansión mediante zócalos PC/104.
- Formato de placa tipo PC/104 (95.9 x 115.6mm).
- Bajo consumo de potencia (6W).
- Alimentación única (+5V). Todas las tensiones necesarias se generan internamente.
- No necesita ventilador.

La compacidad y bajo consumo de esta tarjeta la convierten en una opción idónea para su uso en sistemas embebidos. Por otra parte, la capacidad de operar sin refrigeración activa supone una mejora importante frente al modelo de *Acrosser*, ya que la ausencia de ventilador en el microprocesador ahorra consumo, reduce el nivel de ruido electromagnético y aumenta la fiabilidad.

Además de las ventajas que proporcionan las prestaciones de esta placa en la operación final del sistema, algunas de sus características aportan mayor funcionalidad en la etapa de desarrollo. Así, por ejemplo, el hecho de que el PC industrial de *Lippert* contara con controladores de red *ethernet* y VGA integradas en la propia placa hizo innecesario el uso de tarjetas adicionales con estas funciones, que en el caso del PC de *Acrosser* debían interconectarse mediante un *backplane* pasivo. La alimentación única a +5V también supone una mejora tanto en la etapa de desarrollo como en la operación final, ya que simplifica el diseño de la fuente de alimentación o reduce costes si se utiliza un modelo de fuente comercial.

2.3.4. Los receptores GPS

2.3.4.1. El sistema GPS

GPS es el acrónimo de *Global Positioning System* o Sistema de Posicionamiento Global, un sistema de navegación basado en satélites operado y mantenido por el Departamento de

[15] www.lippert-at.com

Defensa de los Estados Unidos. Consta de una constelación de 24 satélites que proporcionan datos de precisión de tiempo y espaciales en tres dimensiones. Las trayectorias de los satélites están diseñadas de forma que se garantiza una cobertura continua en toda la superficie terrestre.

Los receptores GPS son capaces de determinar una posición precisa por medio de cálculos de la distancia a los satélites de la constelación GPS cuya señal se reciba dentro de unos parámetros de calidad. Para poder determinar la localización espacial bidimensional es necesario recibir correctamente la señal de al menos tres satélites. El cálculo de la posición tridimensional, que incluye también la altitud, requiere como mínimo la señal de cuatro. Los receptores GPS pueden proporcionar además información precisa del tiempo, velocidad y rumbo, lo cual resulta de gran utilidad en aplicaciones de navegación.

El GPS diferencial (DGPS) es un modo de navegación que proporciona una precisión aún mayor que el GPS convencional. Esta técnica se basa en correcciones de error transmitidas por un receptor GPS situado en una localización conocida. Este receptor, denominado estación base o de referencia, calcula el error en los datos de posición enviados por los satélites y genera correcciones que transmite a otros receptores, que se denominan estaciones móviles y que pueden estar a distancias de hasta 100 km de la estación base. El GPS diferencial elimina prácticamente los errores en las medidas de posición de los satélites y permite un cálculo de la posición del receptor altamente precisa.

2.3.4.2. Receptores

Hoy en día los receptores GPS son artículos de uso común. Los más demandados son los receptores personales, que ofrecen unas características muy concretas orientadas principalmente a la localización de objetos y personas y a la navegación de vehículos. Generalmente cuentan con interfaces gráficas de usuario complejas que incluyen pantallas de cristal líquido para representar la información y con dispositivos de entrada de datos que permiten realizar la configuración del dispositivo de forma sencilla e intuitiva. Estos receptores permiten, entre otras funciones, conocer en tiempo real la posición, velocidad y aceleración, marcar puntos de paso o grabar los trayectos realizados. Los modelos orientados a la navegación de vehículos permiten además cargar mapas digitales de carreteras y ciudades a través de distintos tipos de puertos de comunicaciones (USB, IrDA, RS-232,…). Es posible introducir el origen y el destino de un trayecto, a partir de los cuales el sistema calcula la ruta más adecuada e informa al usuario en tiempo real sobre la posición actual y el camino a seguir. En caso de que el usuario se desvíe de la ruta originalmente calculada por cualquier motivo, el sistema la recalcula de modo automático. La figura 2.6.a) muestra uno de estos receptores.

En el campo de la instrumentación geofísica, y en general en el de los sistemas embebidos de adquisición de datos, se usa otro tipo de receptores. No interesa una interfaz gráfica de usuario, ya que los receptores GPS irán integrados en un sistema mayor que incluirá, en caso de ser necesaria, la correspondiente interfaz gráfica. Por otra parte, teniendo en cuenta que en general los sistemas tendrán un emplazamiento fijo, tampoco es necesario que los receptores GPS sean capaces de calcular variables cinemáticas (velocidad y aceleración) ni de determinar rutas o establecer puntos de paso. Sin embargo sí es importante, al contrario que en los receptores personales orientados a navegación, que el dispositivo cuente con señales de tiempo de precisión accesibles al usuario, que se utilizarán como referencia para lograr la adquisición sincronizada de los datos.

La interfaz en este tipo de receptores suele constar de uno o varios puertos serie, normalmente de tipo RS-232 o RS-485, que facilitan el control desde PCs o sistemas diseñados específicamente basados en microcontroladores. La comunicación se puede realizar a través de distintos protocolos, tanto binarios como ASCII (TSIP, TAIP, NMEA 0183,…)

(ver, por ejemplo, Trimble Navigation Limited, 1999). Estos protocolos permiten realizar de un modo simple una configuración exhaustiva de gran cantidad de parámetros de operación, como los relativos al puerto de comunicaciones o al formato de los datos. Además se le pueden enviar al receptor comandos desde el sistema de control para ejecutar, por ejemplo, autochequeos o reinicios del sistema ante comportamientos defectuosos. Los datos que el receptor envía al sistema de control pueden ser relativos al estado del receptor (*status*, número de satélites que se reciben, calidad de cada una de las señales,…) o a la propia información de tiempo y localización espacial.

a)

b)

c)

Figura 2.6. *Fotos de receptores GPS: a) Dispositivo PDA de la marca Acer, modelo n35, preparado para operar como receptor GPS personal. Como puede verse, cuenta con antena GPS integrada y soporte para montar en el interior de un vehículo. La interfaz de usuario está basada en una pantalla táctil de cristal líquido. En b) se muestra el receptor utilizado en la antena del Vesubio, modelo Lassen SK II de la marca Trimble. Este receptor precisa una tarjeta para regulación de la tensión de alimentación e interfaz para comunicación serie, más una antena GPS externa. En c), el receptor utilizado en las antenas portátiles del IAG, modelo HVS 35 de la marca Garmin, que integra en un solo encapsulado el receptor propiamente dicho, el circuito de fuente, la antena GPS y la interfaz para comunicación RS-232, por lo que la conexión al sistema de adquisición es inmediata. La imagen muestra varios de estos dispositivos durante las pruebas finales de las nuevas antenas portátiles del IAG.*

Además de la información de tiempo real, en nuestro caso resulta de especial importancia el patrón de tiempo de precisión que ofrece este tipo de receptores. Éste consiste generalmente en una señal de un pulso por segundo, cuyo flanco inicial está sincronizado con el tiempo universal con una precisión mejor que 1 μs. Los receptores de este tipo suelen ofrecer además la posibilidad de usar GPS diferencial, para lo cual cuentan con un segundo puerto serie a través del cual se introducen las correcciones necesarias en este tipo de técnicas.

La figura 2.6 muestra dos modelos de receptores GPS para uso en sistemas embebidos, concretamente los usados en las antenas del Vesubio (figura 2.6.b) y del IAG (figura 2.6.c). El primero es el modelo *Lassen SK II,* de la marca *Trimble*. El segundo es el GPS 35 HVS, de la marca *Garmin.* Como puede verse, el *Trimble* se reduce a una tarjeta con conectores para la antena y la interfaz serie. Para integrarlo en el sistema total será necesario diseñar otra tarjeta con el circuito de alimentación y una interfaz inteligente, que en nuestro caso estará basada en un microcontrolador PIC. El *Garmin HVS 35*, sin embargo, se presenta en un encapsulado hermético que incluye la antena y el circuito de fuente, del que sale un cable con las líneas de datos y alimentación. La integración en el sistema total se reduce al montaje del conector adecuado para establecer comunicación con el subsistema de control. Las características de cada uno de estos receptores y los criterios de selección empleados se describirán con más detalle en los capítulos correspondientes a la antena del Vesubio y a las antenas portátiles del IAG, cuando se haya presentado la problemática y la estructura general de estos dispositivos.

2.3.5. Los circuitos PLL

PLL es el acrónimo de *Phase Locked Loop* o lazo de enganche de fase[16]. El objetivo de este tipo de circuitos es realizar el control de la frecuencia y fase de un oscilador mediante una señal de referencia externa. Los circuitos PLL tuvieron un papel fundamental en el desarrollo de los receptores de radio homodinos o sincrodinos, que se desarrollaron en 1932 como alternativa a los receptores superheterodinos, de buenas prestaciones pero bastante complejos debido al gran número de etapas sintonizadas que necesitaban. Este es el primer uso documentado de un PLL, pero por problemas de índole tecnológica su uso no se generalizó hasta la década de los 60, cuando aparecieron los circuitos integrados. En la actualidad los circuitos PLL se usan profusamente en sistemas de telecomunicaciones, ordenadores y radio. Entre sus aplicaciones más frecuentes se pueden citar la demodulación de señales de FM, el desarrollo de sintetizadores de frecuencia para receptores y transmisores sintonizados digitalmente, la recuperación de señales con baja relación señal/ruido o el diseño de multiplicadores de frecuencia en microprocesadores.

2.3.5.1. Componentes

La figura 2.7 muestra el diagrama de bloques de un PLL básico. Como puede verse, es un sistema realimentado que consta de tres bloques: un comparador de fase, un filtro paso baja y un oscilador controlado por tensión (VCO). El detector de fase es un dispositivo que compara las dos señales de entrada, generando una salida dependiente de su diferencia de fase. Esta señal de error, una vez filtrada en el filtro paso baja, ataca el VCO de forma que la frecuencia de su señal de salida tiende a desviarse hacia la de la señal de referencia f_{IN} para compensar la diferencia de fase inicial. De este modo la frecuencia y la fase de la señal de salida se igualarán con las de la señal de referencia, lo cual intuitivamente se conoce como enganchar las dos señales, de donde proviene el nombre de este tipo de circuitos. El dispositivo será

[16] Existe gran cantidad de información en línea sobre este tipo de circuitos. Ver, por ejemplo, Rabinovich (2006), Nash (1994) o consultar las páginas web:
en.wikipedia.org/wiki/Phase-locked_loop
www.ifent.org/lecciones/PLL/

capaz de enganchar las dos señales dentro de un cierto rango de frecuencias de la señal de entrada, que se conoce como margen de captura. Una vez enganchadas, el rango de frecuencias sobre las que el PLL puede mantener el enganche se denomina margen de enganche o de seguimiento, que será siempre mayor o igual que el margen de captura.

Un circuito PLL basado en el diagrama de bloques de la figura 2.7.a) daría como salida una señal con la misma frecuencia de la señal de referencia f_{IN}. La configuración más comúnmente utilizada se muestra en la figura 2.7.b), en la que se introduce un contador o divisor de frecuencia en el lazo de realimentación. De esta forma la frecuencia de la señal de salida queda multiplicada por el mismo factor por el que se divide en el contador. Éste factor puede ser fijo o seleccionable, por ejemplo mediante una serie de *jumpers* o microinterruptores en la tarjeta de circuito impreso, como se verá en el próximo capítulo.

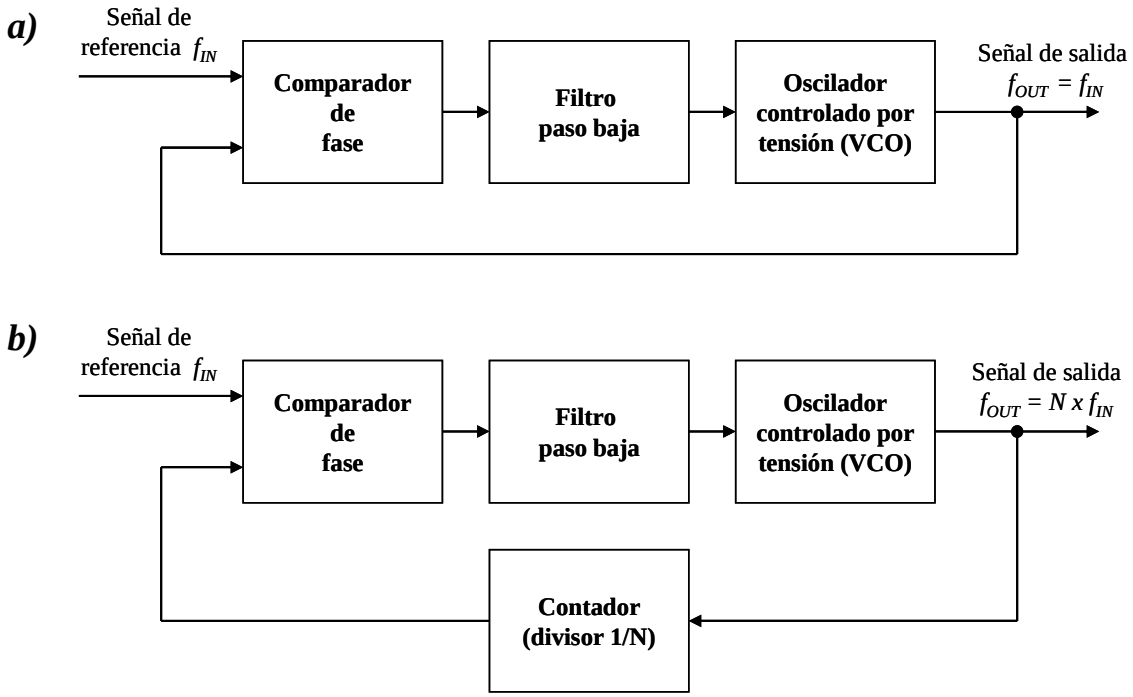


Figura 2.7. *Diagrama de bloques de un circuito PLL. En a) se muestra la configuración básica. En b) la más comúnmente utilizada, que incluye un divisor de frecuencia en el lazo de realimentación para que la frecuencia de la señal de salida sea múltiplo de la de la señal de referencia.*

A continuación de describen brevemente cada uno de los bloques de los circuitos PLL, centrándonos sobre todo en las configuraciones utilizadas en los dispositivos que se presentan en este trabajo.

Comparador de fase.- Tradicionalmente se han utilizado dos tipos básicos de comparadores de fase, comúnmente conocidos como Tipo I y Tipo II. El tipo I (figura 2.8.a) consiste básicamente en una puerta OR exclusiva, que por tanto da un nivel alto de salida cuando sus entradas tienen niveles lógicos distintos. El tipo II (figura 2.8.b) es sensible sólo a la diferencia de tiempo entre los flancos de las dos señales que se comparan. El circuito genera pulsos positivos o negativos dependiendo de si las transiciones de la señal de referencia ocurren antes o después que las de la señal de salida del VCO, respectivamente. La duración de estos pulsos es igual al tiempo entre los respectivos flancos. El circuito de salida absorbe o inyecta corriente en función del tipo de pulsos, quedándose en circuito abierto cuando no detecta pulsos de ningún tipo.

El comparador de Tipo II presenta algunas ventajas frente al de Tipo I. En primer lugar, su comportamiento es completamente independiente del *duty-cycle* de las señales de entrada, cosa que no ocurría en el de Tipo I. En segundo lugar, los pulsos de salida desaparecen completamente cuando las dos señales están enganchadas, lo cual quiere decir que no hay rizado en la salida. El detector de Tipo I, por el contrario, genera constantemente una señal de salida, que aunque es suavizada por el filtro paso baja, normalmente está presente en mayor o menor medida en la salida en forma de rizado. Teniendo en cuenta que la salida del detector de fase de Tipo II se queda en circuito abierto cuando las dos señales están en fase, el condensador del filtro paso baja se comporta como un dispositivo de almacenamiento de carga, manteniendo la tensión que produce la frecuencia correcta a la salida del VCO. Si la señal de referencia varía de frecuencia, el detector de fase genera una serie de pulsos cortos que cargan o descargan el condensador hasta el valor de tensión necesario para enganchar de nuevo el VCO.

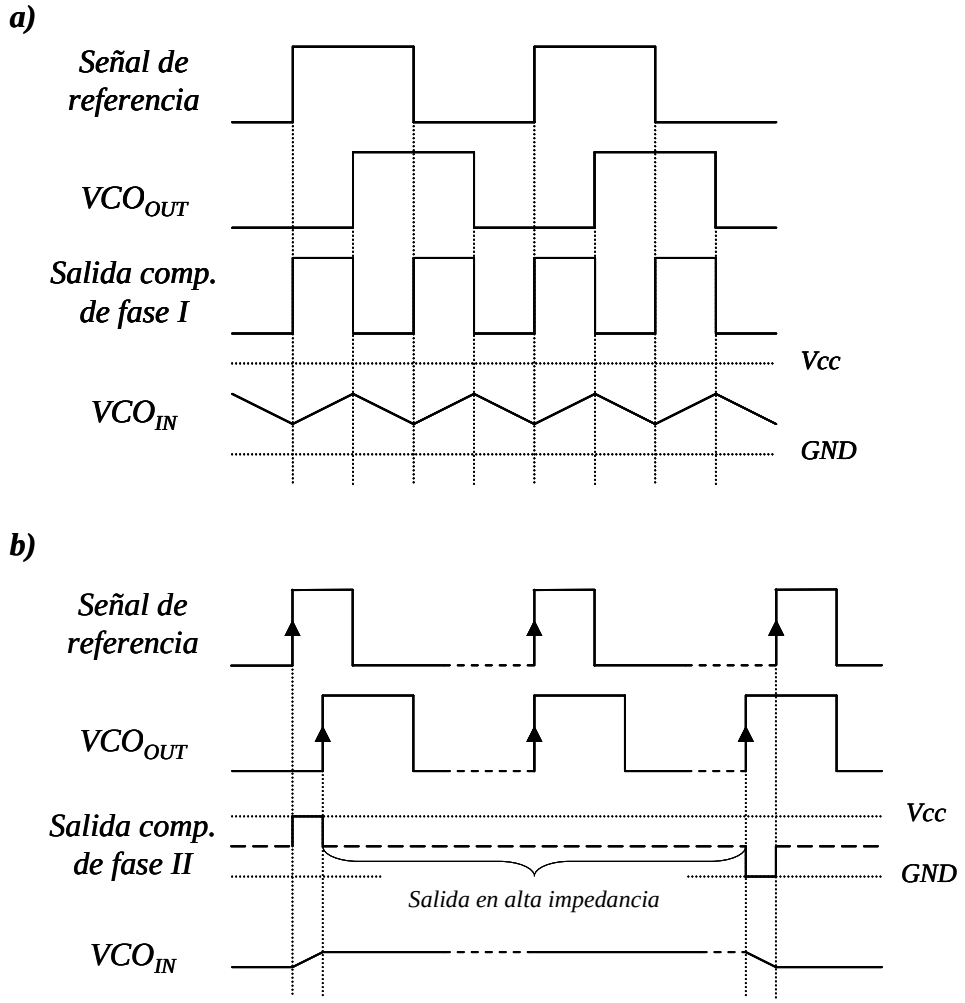


Figura 2.8. *Diagramas de tiempo de los comparadores de fase Tipo I (a) y Tipo II (b) (explicaciones en el texto).*

Filtro paso baja.- El filtro paso baja cumple dos funciones. En primer lugar, elimina el ruido y cualquier componente de alta frecuencia de la salida del detector de fase, tales como armónicos de las señales de entrada. En segundo, es el bloque más importante en la determinación de las características dinámicas del lazo, como rango de captura, respuesta en frecuencia y respuesta transitoria. Cuando las dos señales están enganchadas, el filtro sólo

limita la velocidad del sistema para seguir los cambios de la señal de entrada. Por otra parte, el filtro proporciona al PLL lo que a veces se denomina memoria a corto plazo, asegurando un rescate rápido de la señal si el sistema se sale del enganche debido a algún ruido transitorio.

Los filtros más comúnmente usados en circuitos PLL se muestran en la figura 2.9. En nuestras aplicaciones será suficiente con el primero de ellos, un filtro paso baja pasivo tipo RC.

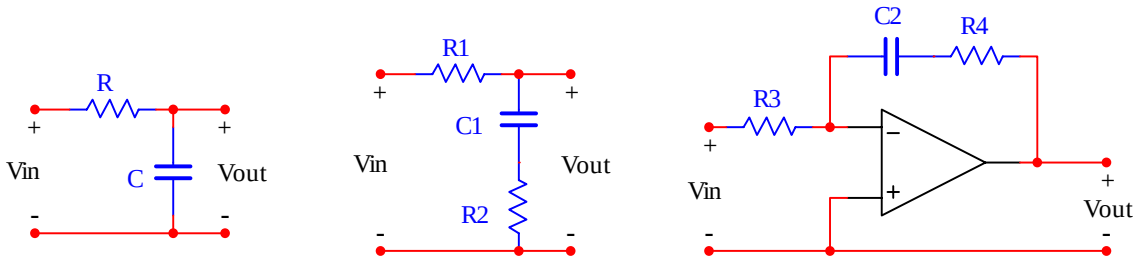


Figura 2.9. *Filtros paso baja comúnmente utilizados en circuitos PLL.*

Oscilador controlado por tensión.- Existen muchos tipos de VCOs, pero los más comunes son los astables o de relajación y los osciladores senoidales sintonizados por diodos varicap. Aquí nos centraremos en estos últimos, ya que son los que se utilizarán en nuestras aplicaciones.

Los diodos varicap, también llamados diodos varactores o diodos de capacidad variable, son diodos que presentan una capacidad dependiente de la tensión inversa. La capacidad de estos dispositivos es debida a las propiedades dieléctricas de la zona de deplexión que se crea al polarizarlos con tensiones inversas. Al variar el valor de la tensión se modifica la anchura de la zona de deplexión y por tanto varía también la capacidad efectiva del varactor.

En realidad casi todos los diodos se comportan como condensadores variables cuando se polarizan en inversa, siempre que se trabaje por debajo de los valores de ruptura. Los diodos varactores no son más que diodos específicamente diseñados para conseguir un amplio rango de valores de la capacidad operando dentro de una zona de trabajo y un intervalo de frecuencias bien conocidos y caracterizados.

Los osciladores sintonizados mediante diodos varicap pueden sintetizarse a partir de diversas configuraciones de osciladores LC, como los conocidos *Hartley* o *Colpitts* (figura 2.10). El problema principal en este tipo de circuitos es su baja estabilidad. Cualquier pequeño cambio en la tensión de entrada, incluidos los producidos por ruido o pulsos espúreos, se ve reflejado en la frecuencia de la señal de salida. Para circuitos que requieran alta estabilidad y bajo ruido se introduce un cristal de cuarzo, que sustituye al circuito resonante LC. El cristal sólo permite pequeñas variaciones en torno a la frecuencia central de oscilación, por lo que el aumento de estabilidad se consigue a cambio de una disminución del rango de frecuencias de salida o rango de sintonía. Este tipo de dispositivos se conoce como osciladores a cristal controlados por tensión o VCXOs y, como se verá en los próximos capítulos, serán los utilizados en nuestras aplicaciones. El rango de sintonía limitado no supondrá un problema, puesto que en todos los casos la señal de referencia del PLL será muy estable y no exigirá cambios de frecuencia grandes al VCXO.

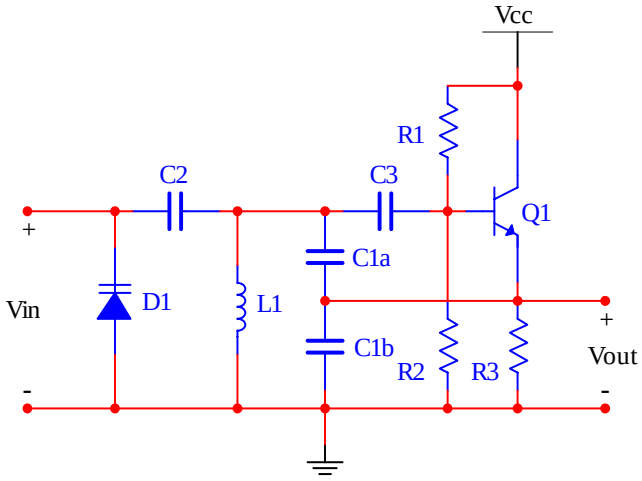


Figura 2.10. *Ejemplo de un oscilador controlado por tensión basado en un oscilador de tipo Colpitts. La frecuencia central queda determinada por el circuito tanque LC formado por L1 y C1 (capacidad formada por los condensadores C1a y C1b, que constituyen un divisor de tensión alterna). La relación entre C1a y C1 determina la amplitud de la señal de realimentación. La variación de frecuencia de salida se obtiene a partir de los cambios de capacidad del diodo varactor D1 con la tensión de entrada (modificado de Rabinovich, 2006).*

2.3.5.2. Circuitos PLL en nuestros diseños

En el presente trabajo se utilizan circuitos PLL en varias aplicaciones, siempre relacionadas con el subsistema de tiempo de las distintas antenas. En el capítulo 1 ya se presentó brevemente el fundamento de las técnicas de sincronización utilizadas en los dispositivos que se describen en esta tesis (ver sección 1.2.5.6 y figura 1.11). Como se recordará, la sincronización del proceso de adquisición de datos en la antena del Gran Sasso se basaba en que todos los conversores A/D utilizaban una base de tiempo común, lo cual garantizaba que todos ellos operasen exactamente a la misma velocidad. En la figura 1.11.a) la señal de 10 MHz que los conversores usan como base de tiempo se genera en un oscilador de precisión al que denominamos oscilador patrón. En realidad la figura es una simplificación del sistema real, que resulta algo más complejo. Como se verá en el próximo capítulo, para evitar los problemas inherentes a la transmisión de señales de alta frecuencia en entornos ruidosos, la señal que se genera en el oscilador patrón y que luego se transmite a todos los puntos de adquisición de datos no es la de 10MHz, sino otra de frecuencia menor. Esta señal se utilizará como señal de referencia en cada punto de adquisición para generar la señal de 10 MHz mediante circuitos PLL. En este caso se utilizaron circuitos PLL comerciales 74HC4046 (Philips Semiconductors, 1997), una versión de la familia HC del conocido 4046 que cuenta, entre otras características, con un tercer comparador de fase que se suma a los de Tipo I y Tipo II incluidos en el integrado original. El 4046 incluye en el propio integrado los bloques correspondientes al comparador de fase y al VCO, por lo que los únicos elementos externos que necesita para operar con la configuración básica de PLL de la figura 2.7.a son los componentes del filtro paso baja. Dado que en nuestro caso el objetivo del circuito era obtener una señal de frecuencia múltiplo de la de referencia, se utilizó el esquema de la figura 2.7.b, por lo que también se introdujo en el circuito un contador binario para dividir la frecuencia de salida del VCO antes de atacar el comparador de fase.

La señal de referencia de estos circuitos PLL se sintetiza en el oscilador patrón, utilizando a su vez otro circuito de este tipo. Como se verá en el próximo capítulo, en este caso se toma como referencia una señal de un pulso por segundo de precisión, generada por un patrón atómico de los laboratorios donde opera la antena. Al contrario que en el caso anterior, en éste no se utiliza un integrado específico como el 4046, debido sobre todo (aunque no únicamente) a que la diferencia de las frecuencias de la señal de salida y la de referencia es demasiado grande. En lugar de eso, para realizar la comparación de fase de las dos señales y el control del VCXO se diseñó una tarjeta inteligente basada en un microcontrolador de la familia PIC, que además se ocupa de otras labores relacionadas con la temporización y sincronización de los datos. Tanto el filtro paso baja como el VCXO, que es

del tipo sintonizado mediante diodos varactores descrito anteriormente, se implementaron como circuitos externos con componentes discretos.

Como se describió en 1.2.5.6 y se muestra en la figura 1.11.b), en las antenas del Vesubio y del IAG también se utilizaron circuitos PLL, en este caso para generar en cada punto de adquisición la señal de reloj de los conversores A/D, tomando como referencia la señal de un pulso por segundo generada por los receptores GPS. Al igual que en el caso anterior, la operación de cada PLL es controlada por una tarjeta inteligente basada en un PIC. La diferencia más notable con respecto a los PLLs de la antena del Gran Sasso es que en estos se utilizaron como VCXOs módulos integrados comerciales, en lugar de circuitos diseñados específicamente e implementados con componentes discretos.

Los VCXOs integrados presentan varias ventajas frente a los discretos. En primer lugar, las especificaciones están garantizadas de fábrica, con lo que no es necesario realizar pruebas previas de calibración y operación. En segundo, ofrecen generalmente márgenes de captura y de ajuste mayores que los que se pueden obtener con componentes discretos. Por último, la estabilidad frente a variaciones de la temperatura también es mayor.

El principal inconveniente que plantea la opción de los módulos VCXO comerciales es su disponibilidad. No se pueden conseguir en los grandes suministradores de componentes electrónicos (RS, Farnell, Digi-Key), ya que normalmente no aparecen en los catálogos, salvo algunos dispositivos con valores estándar de frecuencia de uso común. Generalmente el usuario elige las características que desea para el modelo y encarga una partida más o menos numerosa a una empresa especializada, que corta los cristales de cuarzo a medida y fabrica los osciladores. Esto hace que sean relativamente caros – aunque el precio suele reducirse cuando aumenta el número de unidades del pedido – y que los plazos de entrega sean largos (semanas o, más probablemente, algunos meses). Por otra parte, las empresas sólo suelen estar interesadas en servir pedidos de un número de unidades superior a un mínimo, por lo que la adquisición de estos dispositivos es complicada y la elección de esta opción difícilmente justificable en el caso de aplicaciones con pocos circuitos PLL. Por esta razón se descartó su uso en la antena del Gran Sasso, en la que sólo era necesaria una tarjeta de PLL con VCXO externo. Las antenas del Vesubio y del IAG, sin embargo, precisan un circuito PLL en cada punto de adquisición de datos, por lo que el número total de VCXOs, incluyendo unidades de repuesto, debería rondar las cuarenta unidades. Una vez consultadas distintas empresas sobre la viabilidad del pedido, éste se realizó a la casa *Champion Technologies* (actualmente MTRON[17]), y consistió en cuarenta unidades del modelo K1525CDM-9.8304, cuyas principales características se describirán más adelante.

[17] www.mtron.com

CAPÍTULO 3

LA ANTENA DEL GRAN SASSO

La descripción detallada de las tres antenas sísmicas que se han diseñado y desarrollado abarcará otros tantos capítulos, que se han incluido en esta memoria por orden cronológico. Así pues, la primera que se presenta es la antena del Gran Sasso, cuya primera fase de diseño se inició en el año 1996 y concluyó en torno a 1998, aunque la instalación de las sucesivas líneas de adquisición y transmisión de datos, así como la adaptación e integración de programas y sistemas, continuó de forma intermitente durante distintos periodos posteriormente.

El capítulo comienza con una introducción (3.1), en la que se presentan de una forma muy concisa los laboratorios que constituyen el emplazamiento de este *array*. En esta sección se hace además un repaso a la actividad sísmica reciente en la región y se citan otros sistemas de instrumentación geofísica utilizados en los laboratorios, tanto para realizar pruebas temporales de la respuesta de sitio previas al desarrollo de la antena como para la medida de deformaciones.

A continuación se describen los objetivos tecnológicos y científicos que se pretende cubrir con este dispositivo, y se citan las especificaciones técnicas inicialmente requeridas para el mismo (3.2). En la siguiente sección (3.3) se comentan algunos de los parámetros de diseño condicionados por las características de los laboratorios que servirán de emplazamiento a la antena, se justifican los cambios realizados en sus características respecto a las especificaciones iniciales y se determinan sus prestaciones definitivas.

Las secciones siguientes se ocupan de la descripción del dispositivo, tanto de la instrumentación como de los programas desarrollados específicamente. En primer lugar se da una visión general (3.4), en la que se describe brevemente la estructura del dispositivo y todos sus subsistemas, prestando especial atención al sistema de temporización desarrollado para asegurar la sincronización de los datos. Las siguientes secciones incluyen una descripción más detallada de los distintos módulos desde un punto de vista *hardware* (3.5) y *software* (3.6). Por último se ofrece una visión general del procedimiento de instalación de programas y equipos, incluyendo imágenes de los distintos elementos montados en su emplazamiento final (sección 3.7).

3.1. Introducción

El emplazamiento de esta primera antena sísmica son las instalaciones subterráneas de los Laboratorios Nacionales del Gran Sasso (*Laboratori Nazionali del Gran Sasso*, en adelante LNGS[18]), dependientes del *Istituto Nazionale di Fisica Nucleare* (INFN) italiano.

La idea de dotar al INFN de un gran laboratorio subterráneo para el estudio de física nuclear y astrofísica surgió en 1980. Su realización se llevó a cabo en un tiempo relativamente breve, teniendo en cuenta la complejidad técnica y organizativa del proyecto. En febrero de 1982 el Parlamento Italiano aprobó la iniciativa, asignando una primera partida presupuestaria para la preparación de las obras, y a comienzos de 1985 el INFN ya estaba iniciando la preparación de los primeros experimentos que se iban a llevar a cabo en las instalaciones subterráneas. En la realización de los mismos tomaron parte grupos de investigación de Francia, República Federal de Alemania, Israel, Unión Soviética, Estados Unidos, Canadá, Japón, China y Brasil, a través de algunas de las universidades e instituciones más

[18] www.lngs.infn.it

prestigiosas del mundo, como la Universidad de Harvard, los Institutos Tecnológicos de Massachussets y de California, los Laboratorios Bell, el Instituto Max Planck, el Centro de Investigación de Saclay, el Instituto Weizmann de Rehovot o el CERN.

Los LNGS están situados en el macizo del Gran Sasso, una región sísmicamente activa situada en la cordillera de los Apeninos, a unos 120 km al este de Roma (figura 3.1). Construidos en la proximidad de uno de los túneles de autopista más grandes de Europa, están protegidos por una cubierta de casi 1400 m de roca calcáreo-dolomítica que los protege de las radiaciones cósmicas. Este gigantesco escudo natural crea unas condiciones ideales para la investigación en distintos campos de la física, ya que los niveles medidos de radiactividad natural son diez veces inferiores a los usuales. En el programa experimental, que toca muchos campos de interés dentro de la física de partículas elementales, astrofísica y geofísica, colaboran prestigiosas universidades e instituciones de todo el mundo.

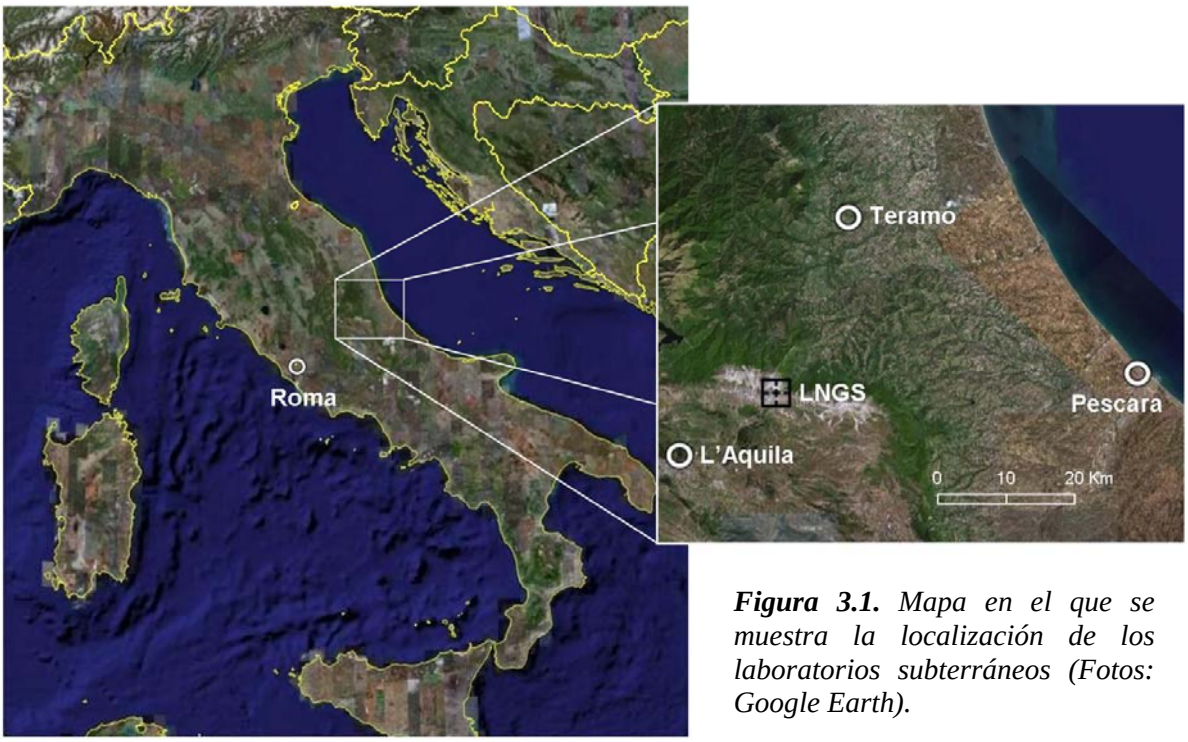


Figura 3.1. *Mapa en el que se muestra la localización de los laboratorios subterráneos (Fotos: Google Earth).*

Además de los laboratorios subterráneos donde se llevan a cabo los distintos experimentos, los LNGS cuentan con unas instalaciones exteriores que constituyen el centro de operaciones de la actividad científica y técnica que se realiza bajo tierra. Situadas en las proximidades de Assergi, incluyen el centro de tratamiento de datos, oficinas administrativas y de dirección y salas de reuniones y estudio, en un edificio que ocupa una superficie de 3250 m^2 y un volumen de 11200 m^3.

El enlace entre el centro de tratamiento de datos y los laboratorios subterráneos (situados a cerca de 7 km. de distancia) se realiza mediante cable óptico formado por cien fibras monomodo de baja atenuación y gran ancho de banda. Las funciones del enlace comprenden canales telefónicos y de transmisión de datos, canales de alta velocidad para diversos experimentos, montaje de redes locales tipo ethernet y conexiones con la red geográfica INFNet. Esta red conecta, a su vez, los LNGS con las universidades y centros de investigación más importantes del mundo.

Para construir los laboratorios subterráneos se aprovecharon los dos túneles excavados para la autopista entre las ciudades de L'Aquila y Teramo, de más de 10 km de longitud. Entrando por el extremo que viene de L'Aquila, el espesor de roca sobre el túnel aumenta, llegando a alcanzar los 1494m (correspondientes al monte Aquila) en el km 6,250. En este

punto están situados los laboratorios subterráneos, que comprenden tres salas experimentales de unos 100 m de largo, 18 de ancho y 20 de alto, conectadas por una red de galerías, conductos y salidas de emergencia (figura 3.2). Además existe una red de enlace con la autopista, destinada al tráfico de vehículos ligeros y pesados para el transporte de materiales.

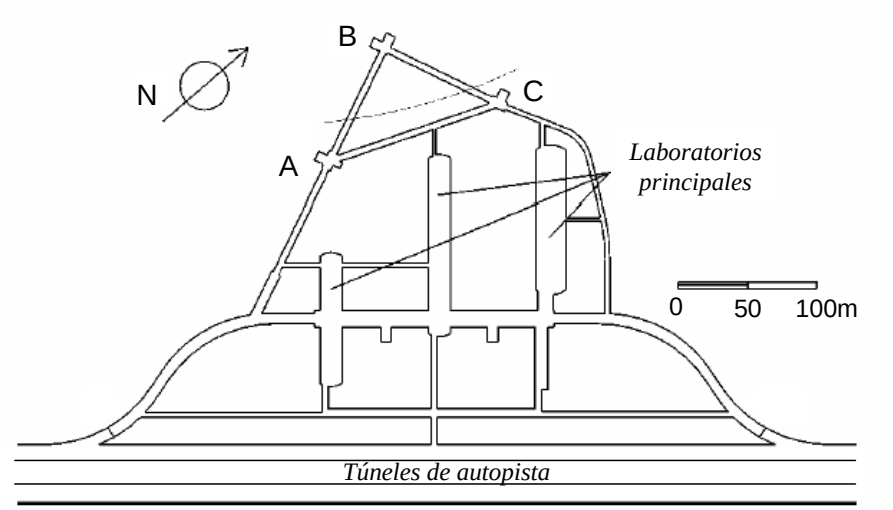


Figura 3.2. *Plano de las instalaciones subterráneas de los LNGS. Pueden apreciarse los tres laboratorios principales y el túnel de la autopista que se aprovechó para los accesos, además de la red de galerías de servicio y emergencia que enlaza todas las instalaciones. Los puntos A, B y C muestran la situación de las tres estaciones de medida del interferómetro láser. La línea de puntos entre ellos representa la situación aproximada de la falla que cruza los dos brazos del instrumento (modificada de Crescentini et al., 1997).*

Debido a la existencia de grandes cantidades de agua en el macizo del Gran Sasso, la temperatura ambiente en los túneles es de 6-7 ºC, con una humedad cercana al 100%. Por esta razón las salas principales y las galerías de enlace están aisladas e impermeabilizadas para obtener una climatización óptima.

La ventilación de los túneles se consigue mediante un sistema de reciclaje de aire formado por una red de tuberías y galerías, que intercambia con el exterior un flujo de 30000 m^3 de aire por hora. Un potente sistema de acondicionamiento garantiza además la habitabilidad de los laboratorios y el correcto funcionamiento de los instrumentos científicos.

Cada una de las salas está atravesada por corredores y puentes dispuestos de forma que se consiga el mayor aprovechamiento posible del espacio. Aparte de utilizarse para el acceso del personal científico y técnico a los distintos módulos, por estos corredores circulan los cables de distribución de la energía eléctrica, las tuberías para el agua y otros fluidos y las líneas de comunicaciones. Existe además una sala de servicios que alberga instalaciones necesarias para la fibra óptica, los transformadores eléctricos, los primeros auxilios y otros servicios indispensables en unos laboratorios de estas dimensiones.

En la actualidad la actividad sísmica de los Apeninos centrales, y en particular de la región del Gran Sasso, es relativamente baja en comparación con otras áreas activas de Europa. La actividad más reseñable recientemente detectada consistió en tres enjambres sísmicos en agosto de 1992, junio de 1994 y octubre de 1996, siendo el mayor terremoto registrado de magnitud $M_L = 4.2$. Sin embargo, en épocas históricas la actividad en el área ha sido bastante importante, como se muestra en la tabla 3.1. Entre los eventos mostrados se cuentan algunos terremotos destructores, como los de 1703 o de 1915, ambos de magnitud cercana a 7. En concreto el de 1915, conocido como terremoto de Avezzano, produjo más de quince mil víctimas y cuantiosos daños materiales (Amoruso *et al.*, 1998). Actualmente, el

nivel medio de actividad en la región es de aproximadamente un microterremoto de magnitud por encima de $M_L = 1$ al día. Además de las características sismotectónicas de la región, la infraestructura existente en los LNGS los convierte en un excelente lugar para realizar estudios relacionados con la propagación de ondas en medios complejos, fuente sísmica y desarrollo de técnicas de alta sensibilidad para la detección de precursores de terremotos. La antena sísmica cuyo diseño y desarrollo se presenta en este capítulo constituye un instrumento de altas prestaciones capaz de detectar e identificar los principales parámetros de la sismicidad regional.

Las especiales condiciones de los LNGS hacen que los registros sísmicos obtenidos tengan unas características particulares. En 1993 se realizaron algunos experimentos preliminares para estudiar la respuesta de sitio de los laboratorios. Se dispuso temporalmente una antena en L, formada por 17 estaciones sísmicas digitales de corto periodo y tres componentes espaciadas 600 m., dando una extensión espacial total de 10.5 km (De Luca *et al.*, 1998). La figura 3.3.a) muestra los sismogramas registrados en varias de las estaciones de la antena, para un terremoto ocurrido el 21 de febrero del 93. La figura 3.3.b) muestra el mismo terremoto tal y como fue registrado en algunas de las estaciones de la red sísmica regional, situadas en las proximidades de los LNGS pero fuera de las instalaciones subterráneas. Hay dos características interesantes en la respuesta del dispositivo: una gran homogeneidad en la respuesta espectral y, para las ondas S, una disminución media de las amplitudes con respecto a los datos registrados en la superficie, dentro de la banda 1-8 Hz. Como puede verse en la figura 3.4, esta disminución llega a ser de un factor de 4 en las componentes horizontales y de más de 2 en la vertical.

Año	*Latitud*	*Longitud*	*Magnitud*
349	42.10.12	13.22.48	No determinada
1461	42.19.12	13.33.00	6.2
1461	42.18.00	13.33.00	No determinada
1639	42.37.48	13.16.12	5.5
1639	42.39.00	13.15.00	6.0
1703	42.25.48	13.18.00	6.6
1706	42.04.48	14.04.48	6.6
1762	42.18.00	13.34.48	5.5
1786	42.19.12	13.22.12	5.4
1904	42.06.00	13.19.12	5.6
1904	42.58.48	13.15.00	4.8
1915	42.58.48	13.39.00	6.9
1984	42.40.12	14.03.00	5.9
1984	42.43.12	14.03.00	5.5

Tabla 3.1. *Principal actividad sísmica histórica en la región del Gran Sasso.*

Las estaciones cuyos registros aparecen en la figura 3.3 pertenecen a la antigua red sísmica regional, que estuvo operativa hasta el año 2002, aproximadamente (ver, por ejemplo, De Luca *et al.*, 2000). En la actualidad no existe una red regional como tal, sino que las estaciones de registro que operan en el área pertenecen a la red sísmica nacional, gestionada por el INGV (*Istituto Nazionale di Geofisica e Vulcanologia*). La figura 3.5 muestra un mapa con la distribución de estaciones en la región central de Italia. La red cuenta con estaciones de corto periodo, banda ancha y acelerómetros con cobertura densa en todo el territorio nacional, y se complementa con algunas otras redes asociadas, como la gestionada por el *Osservatorio Vesuviano* en la región de Campania, que se describirá en el capítulo 4.

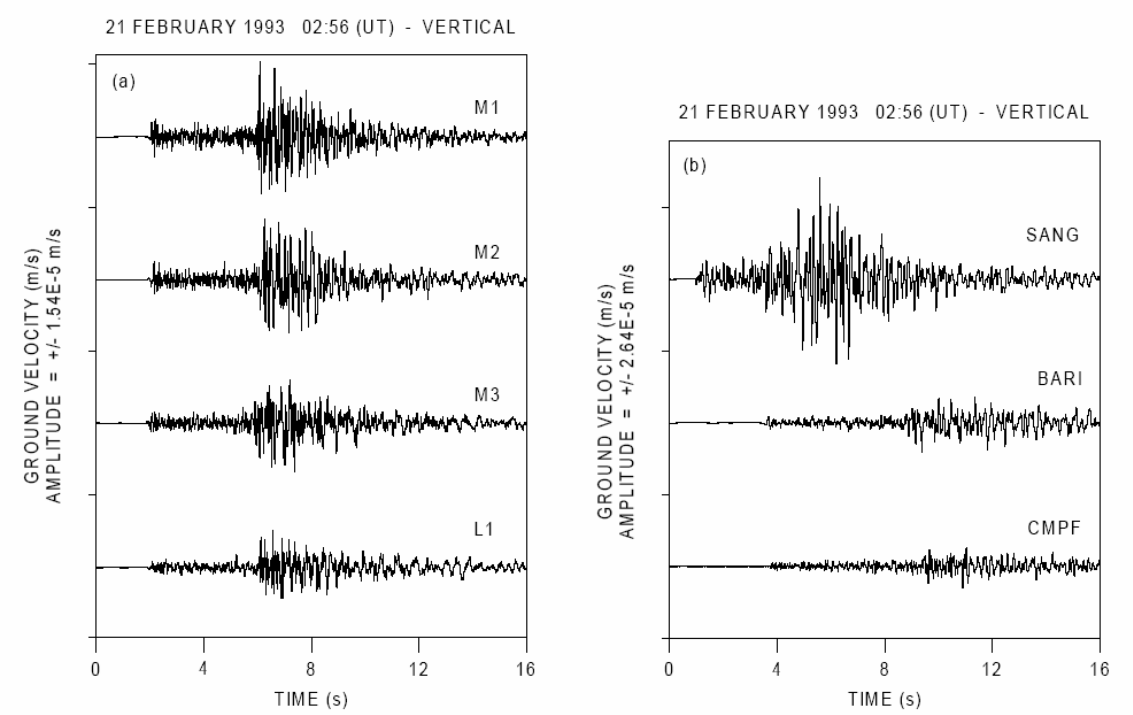


Figura 3.3. *Registros de un mismo terremoto en algunas de las estaciones del* array *temporal (a) y en cuatro estaciones de la red sísmica regional (b) (de De Luca et al., 1998).*

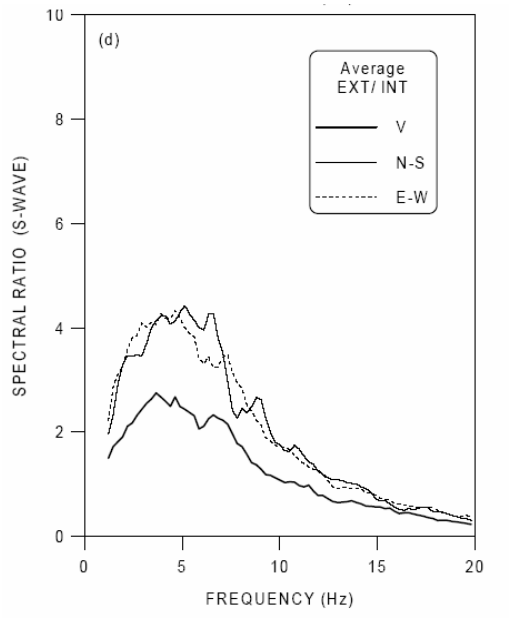


Figura 3.4. *Relación de respuesta para las ondas S entre las estaciones de la red sísmica regional y el* array *en L instalado temporalmente en los laboratorios subterráneos (de De Luca et al., 1998).*

Por otra parte, en los LNGS también se lleva a cabo la monitorización de deformaciones mediante el uso de un interferómetro láser de alta sensibilidad (Amoruso *et al.*, 1997; Crescentini *et al.*, 1997). Como puede verse en la figura 3.2, el instrumento mide los desplazamientos relativos entre tres estaciones situadas en los vértices A, B y C de un triángulo rectángulo, cuyos catetos cruzan una falla de actividad moderada. Se trata de un interferómetro de Michelson de alta sensibilidad ($\Delta L/L \approx 10^{-12}$), alto rango dinámico y gran fiabilidad. Su resolución nominal es de 3pε, aunque la capacidad real de detección de señales geofísicas está limitada por el ruido presente en su emplazamiento, originado principalmente por la actividad humana y por fluctuaciones de la presión atmosférica. El instrumento,

operativo desde 1994 con diversas configuraciones, ha obtenido resultados importantes entre los que destacan el descubrimiento de terremotos lentos[19] de pequeña amplitud cerca del macizo del Gran Sasso (Crescentini *et al.*, 1999; Amoruso *et al.*, 2002). Los terremotos lentos son uno de los mecanismos propuestos para solucionar el denominado 'problema de déficit de deslizamiento sísmico', según el cual los terremotos detectados sólo aportan energía para justificar una fracción del movimiento observado y predicho de las placas tectónicas. Los deslizamientos lentos aportarían una parte de este déficit sin radiar ondas sísmicas y, en consecuencia, sin haber sido observados hasta la fecha por redes de sismómetros. En la actualidad el estudio de terremotos lentos mediante el interferómetro de los LNGS se engloba dentro del proyecto GIGS.

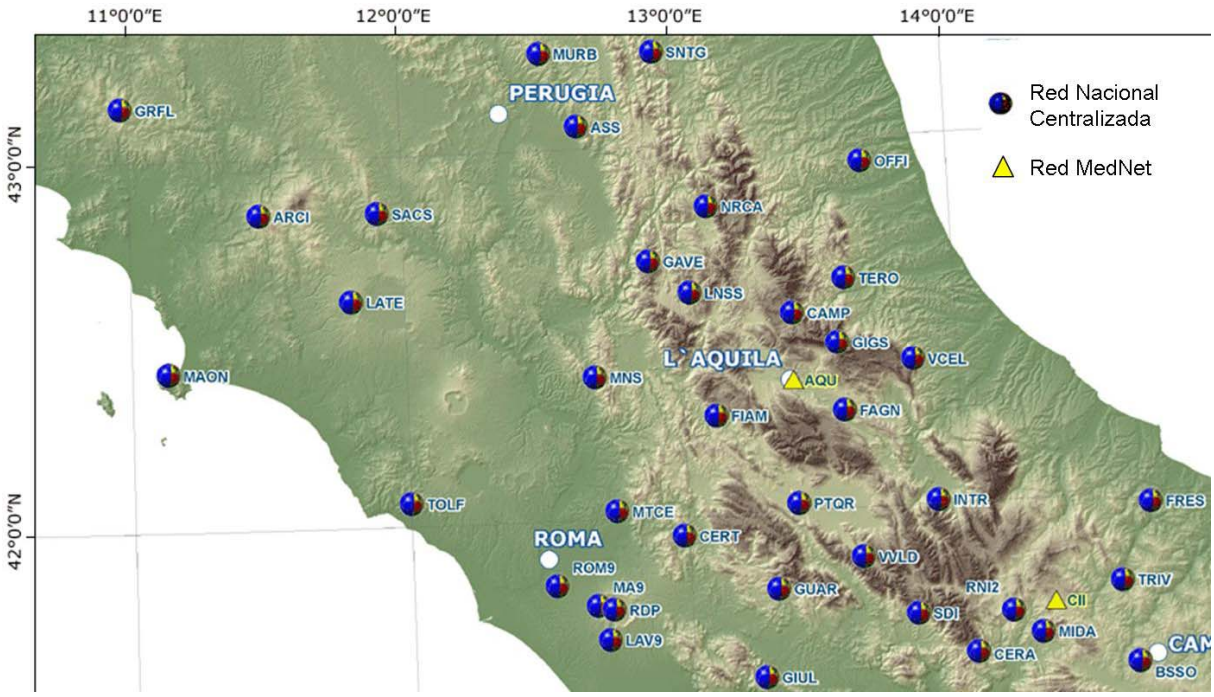


Figura 3.5. *Mapa de las estaciones de la red sísmica nacional gestionada por el INGV en la región central de Italia. Además aparecen dos estaciones de la red de banda ancha MedNet (modificado del sitio web del INGV en Roma[20]).*

Otros proyectos actualmente operativos en los LNGS relacionados con la geofísica son ERMES[21] y TELLUS.

ERMES (*Environmental Radioactivity Monitoring for Earth Sciences*) se centra en el estudio de la relación entre las variaciones de radón en depósitos de agua subterránea y las tensiones en las rocas, así como en la investigación de propiedades de transporte relacionadas con la geoquímica del agua subterránea.

El objetivo principal del proyecto TELLUS es la monitorización continua de deformaciones en los LNGS con el fin de detectar episodios asísmicos lentos de liberación de

[19] Los terremotos lentos son eventos que producen registros similares a los de terremotos normales, excepto por el hecho de que la escala de tiempos del proceso de ruptura es considerablemente mayor (Sacks *et al.*, 1978). Han sido observados en Japón y California como cambios de tensión prácticamente exponenciales con duraciones de entre una hora y tres días. En otros casos se manifiestan en los sismogramas como comportamientos espectrales anómalos de largo periodo, presentando duraciones menores (desde unos pocos segundos hasta varios minutos).

[20] www.ingv.it/~roma

[21] www.fis.uniroma3.it/~plastino/ERMES

esfuerzos asociados a la preparación de terremotos. Este proyecto pretende caracterizar los terremotos desde una perspectiva que engloba distintos fenómenos y según la cual éstos deben describirse mediante teorías capaces de explicar las causas de su génesis, dinámica, reología y física de su preparación, ocurrencia, fases sísmicas y relajación postsísmica. Los procesos asociados a la sismicidad incluyen emisiones electromagnéticas, acústicas y de gas que perturban el medio circundante y que pueden alcanzar largas distancias, hasta la ionosfera y magnetosfera terrestre. Así, TELLUS no se limita a la monitorización local de deformaciones, sino que forma parte de un ambicioso proyecto que pretende registrar posibles perturbaciones e inestabilidades en el espacio cercano a la Tierra como consecuencia de dichos procesos. Para ello se ha propuesto la misión espacial ESPERIA (*Earthquake investigation by Satellite and Physics of the Environment Related to the Ionosphere and Atmosphere*), basada en un microsatélite de órbita baja desde el cual se registrarían parámetros relacionados con flujos de partículas cargadas, plasma ionosférico y campos electromagnéticos en las bandas de ULF, ELF, VLF y HF.

Todos estos proyectos relacionados con la geofísica aprovechan las especiales condiciones de los LNGS, no sólo por la infraestructura existente, que facilita la realización física de los experimentos, sino sobre todo por la situación privilegiada de los laboratorios, próxima a un sistema de fallas en una zona sismogenética de actividad moderada.

3.2. Objetivos y especificaciones técnicas iniciales

Los objetivos científicos que se pretende conseguir con el instrumento que se ha diseñado son contribuir a un mejor conocimiento de una región potencialmente muy sismogenética de Italia y llevar a cabo estudios detallados de los procesos físicos que originan las rupturas sísmicas en el área. La instalación del dispositivo permitirá además realizar un estudio experimental de los fenómenos de propagación de ondas en un medio heterogéneo, que resultarán de gran interés en la evaluación de riesgos sísmicos en áreas de geología compleja. El emplazamiento subterráneo de los LNGS crea unas condiciones privilegiadas para este tipo de estudios. Como se ha visto en las figuras 3.3 y 3.4, la comparación entre las señales registradas en las estaciones en superficie y las de la antena subterránea instalada temporalmente en el experimento previo de 1993 evidencian que los efectos de superficie libre, y sobre todo la resonancia debida a capas superficiales, afectan significativamente al perfil espectral, particularmente para eventos a distancias menores de 50 km. La antena que se presenta en este capítulo permitirá comparar los espectros de ondas Coda de las estaciones en superficie y los de las estaciones subterráneas, lo cual posibilitará el estudio de los procesos de *scattering* que tienen lugar en la región.

Según se vio en el capítulo de introducción, uno de los primeros pasos en el diseño de un dispositivo de *array* es la elección de su configuración espacial, descrita por los parámetros de apertura, separación media entre estaciones y disposición espacial de las mismas. En este caso, sin embargo, tanto la apertura total de la antena como la disposición espacial de las estaciones están condicionadas por la estructura de los laboratorios subterráneos[22]. Así, la apertura está limitada a la extensión del sistema de túneles, y viene a ser de unos 400 x 600 m, mientras que la disposición de los sensores será aproximadamente semicircular para cubrir con la mayor resolución posible toda la superficie de los laboratorios. La separación media entre los sensores de corto periodo será de unos 90 m. Esta distribución espacial permite resolver longitudes de onda en el rango 180-500 m, correspondientes a

[22] El proyecto inicial contemplaba la instalación de estaciones a distintas profundidades, con lo que se conseguiría una cobertura tridimensional del campo de ondas. Sin embargo, hasta el momento este proyecto se ha desechado por motivos técnicos y, sobre todo, económicos.

velocidades de fase de 0.2-10 km/seg.

El diseño inicialmente previsto para el dispositivo consistía en una antena con 21 puntos de adquisición, cada uno de ellos equipado con un sensor de corto periodo de tres componentes. Por tanto, el número total de canales que originalmente debía gestionar el sistema era de 63. El muestreo se realizaría a una frecuencia de 200 muestras por segundo (mps) con una resolución de 24 bits nominales.

Como se verá a continuación, algunas de estas especificaciones se modificaron tras la discusión de varias cuestiones relacionadas con la viabilidad del proyecto y aplicaciones posteriores del dispositivo.

3.3. Consideraciones de diseño

Como primer paso en el desarrollo del dispositivo, se plantearon algunas consideraciones generales de diseño en base a las características de los LNGS y a los requerimientos técnicos del sistema.

En lo que respecta a las características de los LNGS, el primer punto a tener en cuenta fueron las especiales condiciones ambientales. Si bien la temperatura en los laboratorios se mantiene prácticamente constante (6-7 ºC), y por tanto no requiere tomar medidas especiales para el desarrollo de la instrumentación, el alto nivel de humedad (cercano al 100%) sí que crea un entorno hostil e hizo necesaria la elección de cajas y conectores con el grado de estanqueidad y aislamiento adecuados.

Otra de las características de los LNGS que había que considerar era el alto nivel de contaminación electromagnética (EMI) existente, originada por el funcionamiento de los distintos equipos operativos en los numerosos experimentos que se llevan a cabo en los laboratorios. Esto nos hizo decidir por la utilización de transmisión serie digital en lugar de analógica, escogiéndose una interfaz de transmisión diferencial (RS-485). Además se eligieron, siempre que la velocidad de transmisión lo permitía, dispositivos para la transmisión-recepción serie con limitación en la rapidez de respuesta (*slew-rate*). Estos dispositivos permiten menor velocidad pero, a cambio, ofrecen un comportamiento mejor frente a interferencias electromagnéticas y reflexiones en los cables, garantizando por tanto un menor número de errores en la comunicación (Maxim Integrated Products, 1996). Por otra parte, también se siguieron todas las recomendaciones de los fabricantes para conseguir una comunicación serie fiable, (Goldie, 1992 y 1996, Nelson, 1995), como son el uso de resistencias de fin de línea, conexionado de la malla del cable para minimizar las interferencias, limitación de la longitud de las líneas en función de la velocidad de transmisión o evitar el uso de ciertas topologías del bus.

Una de las grandes ventajas que ofrecen los LNGS para el desarrollo de un sistema como el que nos ocupa es que es posible contar con una infraestructura de la que se carece en muchos proyectos de este tipo. Esto hace que problemas como el consumo -determinante en diseños de *arrays* portátiles o en enclaves aislados (Ortiz *et al.*, 1994, Abril e Ibáñez, 2000)- no sean condicionantes en nuestro caso, ya que es posible alimentar toda la instrumentación directamente de la red eléctrica (220V-50Hz) disponible en los laboratorios. Este factor hizo que se optara por la utilización de tarjetas de PC industrial para el control de la adquisición de datos y transmisión serie en las estaciones, en lugar de microcontroladores de bajo consumo como habría sido recomendable en otros casos. Las tarjetas de PC aportan mayor sencillez y potencia de programación, compatibilidad con otros sistemas de adquisición y flexibilidad en el código con un coste relativamente bajo.

Actualmente la mayoría de los sistemas embebidos para el estudio de variables físicas que necesitan una sincronización precisa de los datos - entre los que se cuentan las antenas sísmicas - utilizan como señales de referencia de tiempo las suministradas por receptores GPS

(ver, por ej., Abril e Ibáñez, 2000). Existen en el mercado múltiples modelos que proporcionan señales de tiempo de alta precisión a un precio muy asequible. Las interfaces de usuario son simples y permiten la comunicación con el receptor a varias velocidades y usando distintos protocolos, lo cual simplifica mucho la programación de estos dispositivos. Por otra parte, cada vez se consiguen sistemas de menor consumo, por lo que esta opción suele ser la más adecuada en sistemas portátiles y/o con alimentación autónoma. Sin embargo, la utilización de este tipo de receptores está condicionada a la necesidad de contar con un punto cercano con buena visibilidad GPS, es decir, que garantice una buena recepción de forma permanente o casi permanente de la señal de al menos tres satélites de los veinticuatro que constituyen la constelación GPS. La localización subterránea de los laboratorios del Gran Sasso planteaba, por tanto, un problema para la gestión de este tipo de receptores, razón por la cual se decidió no utilizarlos y aprovechar, de nuevo, las ventajas que proporcionaba la infraestructura existente de otros experimentos. En este caso se decidió emplear, como señal de referencia de tiempo para la sincronización de la antena, alguno de los patrones de tiempo atómicos utilizados por varios de los experimentos que se llevan a cabo en los LNGS (GALLEX, MACRO, LVD,...). Sin embargo, los patrones atómicos no proporcionan una señal de tiempo real, sino sólo una de un pulso por segundo, por lo que fue necesario diseñar el sistema de sincronización a partir de ésta. Como se verá más adelante, éste se basa en un dispositivo (oscilador patrón) que, tomando como entrada la señal de un pulso por segundo procedente del patrón atómico, genera una señal de tiempo real y la envía codificada a cada uno de los puntos de adquisición.

Por lo que se refiere a los requerimientos técnicos del sistema, la primera prioridad era conseguir que el margen dinámico del dispositivo fuera lo suficientemente alto como para permitir el registro de las señales sísmicas en la mayor parte posible de su rango de variación. Como se ha discutido en los capítulos anteriores, en sistemas digitales esto requiere la utilización de conversores A/D (CADs) de alta resolución basados en la técnica de conversión sigma-delta. El funcionamiento de este tipo de dispositivos se fundamenta en un conversor de un solo bit operando en seguimiento continuo de la señal (método de sobremuestreo), aplicando luego a los datos un procesado digital. De esta forma se pueden alcanzar hasta veinticuatro bits de resolución, si bien sólo en banda limitada (Ortiz, 1994).

En segundo lugar, se pedía una frecuencia de muestreo de 200 mps. Sin embargo, hay varias razones que hicieron aconsejable fijar una frecuencia de muestreo menor:

La más importante es que, como se vio en el capítulo anterior (ver tabla 2.1), en los CADs que usan la técnica sigma-delta la resolución efectiva disminuye al aumentar la frecuencia de muestreo y la ganancia, de modo que esto impone una limitación práctica a la hora de escoger los parámetros de funcionamiento. Por otra parte, una frecuencia de muestreo alta conservando a su vez una resolución alta da como resultado un aumento del volumen de datos que se deben transmitir, lo que conlleva la necesidad de usar velocidades de comunicación serie mayores. Sin embargo, cuando se utiliza el estándar para comunicación serie RS-485 es necesario llegar a un compromiso entre las longitudes de las líneas y la velocidad de transmisión, y con las longitudes de línea consideradas en nuestro caso no era recomendable usar velocidades estándar por encima de los 115.2kbps.

Hay que tener en cuenta que el valor de 200 mps requerido inicialmente para la frecuencia de muestreo no respondía a la necesidad de aumentar el ancho de banda de los canales[23], sino a obtener una buena resolución temporal. Por tanto, las prestaciones de la antena no se ven mermadas al reducir la frecuencia de muestreo, siempre que el sistema de temporización del dispositivo sea lo suficientemente preciso como para garantizar la sincronización de las muestras adquiridas en los distintos puntos de sensado.

Por todas estas razones se decidió fijar la frecuencia de muestreo en 100 mps. No

[23] El valor de la frecuencia de Nyquist para una frecuencia de muestreo de 100 mps es de 50 Hz, lo cual resulta más que suficiente para el tipo de señales que se pretende registrar con esta antena.

obstante, en el diseño del dispositivo se tuvo siempre presente la posibilidad de implementar en el futuro el muestreo a 200 mps, teniendo especial cuidado en que el diseño del *hardware* no impusiera ninguna limitación en este sentido. De esta forma el muestreo a 200 mps sólo requeriría la realización de cambios menores en los programas de los distintos módulos. Como se verá en el capítulo 7, en caso de implementar esta modificación se podrían configurar los conversores A/D para que su resolución nominal fuera de 16 bits en lugar de los 24 con los que se digitaliza la señal a 100 mps. Otra opción seria continuar operando con la resolución nominal de 24 bits, pero transmitiendo tan solo los 16 más significativos, ya que a 200mps el byte menos significativo no refleja cambios reales de la señal sino que corresponde a ruido electrónico (ver tabla 2.1). De esta forma se reduciría el volumen de datos a transmitir, que como se ha visto es uno de los problemas asociados a la utilización de frecuencias de muestreo elevadas.

En tercer lugar, se aumentó el número de canales a implementar en el sistema, de los 63 inicialmente considerados (correspondientes a 21 puntos de adquisición de tres componentes) a 72 (24 puntos de tres componentes). La razón para hacer esto fue que, como se verá en la descripción general de la antena, con la configuración elegida para el dispositivo la incorporación de tres nuevos puntos era inmediata e incluso hacía más simétrico el sistema: con 21 puntos, tres de los PCs intermedios se tendrían que ocupar del control de líneas serie con tres estaciones y otros tres de líneas de cuatro estaciones, mientras que con 24 puntos de adquisición todas las líneas serie serían de cuatro estaciones. Aunque en la primera etapa de desarrollo el sistema se dotaría de los 21 sensores triaxiales de corto periodo considerados inicialmente, los tres puntos de adquisición adicionales se podrían dotar en un futuro de sensores de banda ancha. De este modo se proporcionarían nuevas prestaciones al dispositivo, ya que, además de permitir el estudio de fenómenos de propagación de ondas y procesos de fuente de los microterremotos de la región, con la incorporación de los sensores de banda ancha se facilitaría el estudio de terremotos mayores a escala global.

Teniendo en cuenta las modificaciones comentadas, las características definitivas para esta antena serán:

- 72 canales de adquisición
- Frecuencia de muestreo de 100 mps
- 24 bits nominales de resolución
- Ganancia configurable entre 1 y 128
- Sincronización de datos mediante patrón atómico de tiempo

3.4. Descripción general del sistema

3.4.1. Estructura del dispositivo

La estructura general del dispositivo se muestra en la figura 3.6. Como puede verse, cada una de las veinticuatro estaciones de que se compone el sistema en su configuración máxima controla la operación de un sensor de tres componentes. La electrónica de la estación consta de una tarjeta de conversión analógico/digital y una placa de PC industrial que se ocupará del control de la adquisición de los datos y la transmisión serie de los mismos, además de otras tarjetas cuyas funciones se describirán más adelante.

Las estaciones estarán conectadas, vía serie, a otros PCs industriales que actuarán como concentradores de datos y enlace con los servidores. Se usarán seis de estos PCs intermedios (que llamaremos PCs nodales), de modo que cada uno de ellos se ocupará del control de cuatro estaciones. El estándar elegido para la transmisión serie es el RS-485.

Los seis PCs nodales estarán, por último, conectados a través de una red de área local *ethernet* a uno o varios servidores, que se ocuparán de la grabación ordenada de los datos en su disco duro y de un preprocesado y análisis de los mismos. Esta red de área local puede conectarse a su vez a Internet, para permitir el acceso remoto a los usuarios del sistema a ficheros de datos y programas de los servidores y PCs nodales.

Ya se ha comentado antes la importancia de la sincronización temporal de los datos en dispositivos de antena sísmica. Como puede verse en la figura 3.6 y se explicará más adelante, las señales de tiempo necesarias para la sincronización del sistema se generarán en un oscilador de precisión (oscilador patrón) y luego se transmitirán a cada una de las estaciones, usando también transmisión serie basada en el estándar RS-485.

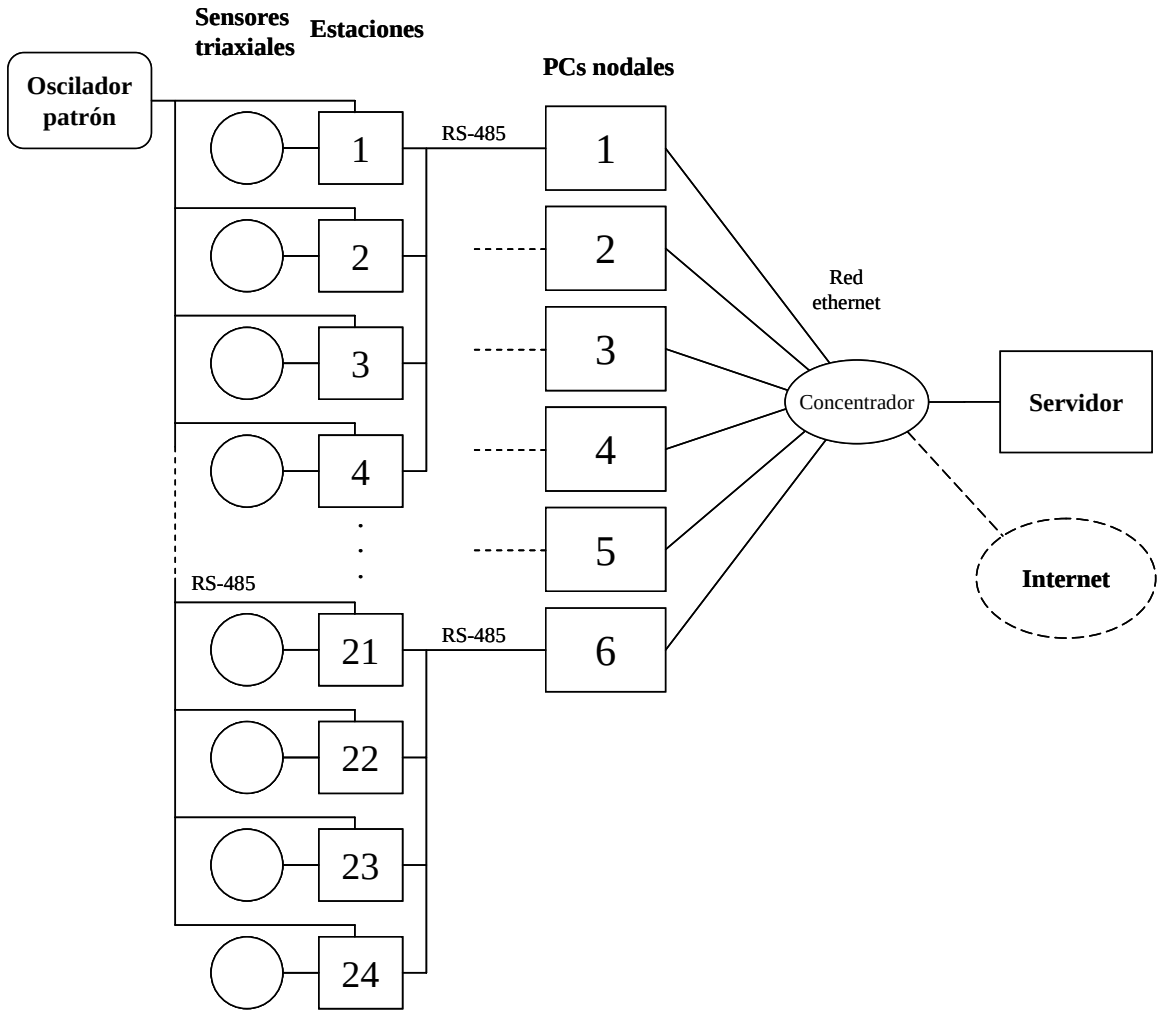


Figura 3.6. *Estructura general del sistema.*

3.4.2. Descripción general del funcionamiento

La primera etapa en el proceso de registro de las señales es la adquisición de los datos, que es gestionada por los controladores de las estaciones. Estos controlan la operación de los CADs y van formando, con las muestras adquiridas, tramas de datos de longitud fija correspondientes a un segundo de adquisición. Además de los datos adquiridos, en las tramas se introduce información relativa al tiempo de adquisición y otras características de los mismos. Mientras no son solicitadas por el PC nodal correspondiente, las tramas son temporalmente grabadas por el controlador de cada estación en un búffer de datos circular.

Cada PC nodal se ocupa del control de la transmisión de datos de las cuatro estaciones conectadas a su línea serie. Para ello interroga sucesivamente a cada una de ellas, que le responden enviando una trama de datos (si tienen alguna en el búffer), o alguna trama de error si se ha producido algún fallo en la operación de la estación. El protocolo de comunicación serie y el formato de las tramas se discutirá más detalladamente en la próxima sección. Los datos procedentes de los seis PCs nodales se almacenan en el disco duro del servidor, desde el que también se pueden ejecutar los programas de cambio de parámetros y consulta del estado del sistema, así como otras utilidades de tratamiento y preprocesado de los datos utilizando técnicas de análisis de array.

3.4.3. Protocolo de comunicación serie

La comunicación serie entre las estaciones y los PCs nodales es bidireccional, y se basa en el intercambio de tramas de distintos tipos (tabla 3.2).

Durante el funcionamiento normal de la antena, cada una de las estaciones muestrea a la frecuencia programada y forma tramas de datos correspondientes a un segundo de muestreo, que introduce en un búffer de datos. Si durante el funcionamiento del programa se ha detectado algún tipo de error en la adquisición, se forma una trama informativa del tipo de error.

Cada PC nodal controla la comunicación serie con las cuatro estaciones de su línea interrogando, por turno, a cada una de ellas mediante el envío de una trama de petición de trama. Si la estación interrogada tiene alguna trama de datos en el búffer de datos, o alguna trama de error preparada, toma el control de la línea serie y la manda, dando prioridad a las tramas de error frente a las de datos. Cuando el PC nodal recibe la trama, comprueba su validez y, en caso de ser correcta, gestiona la información recibida: si la trama era de datos graba éstos en el correspondiente fichero del servidor. Si era de error, se ocupa de su tratamiento según el algoritmo programado. Luego pasa a interrogar a la siguiente estación de la línea, y así sucesivamente.

Si la recepción no ha sido buena y la trama recibida por el PC nodal no es válida, éste manda una trama de petición de reenvío y vuelve a quedar a la escucha. La estación que recibe la petición recupera la última trama enviada y la vuelve a mandar.

En caso de que la estación interrogada no tenga ninguna trama (ya sea de datos o de error) para enviar en el momento de recibir la petición, no contesta a ésta. El PC nodal esperará un tiempo prestablecido y pasará a interrogar a la siguiente estación de la línea.

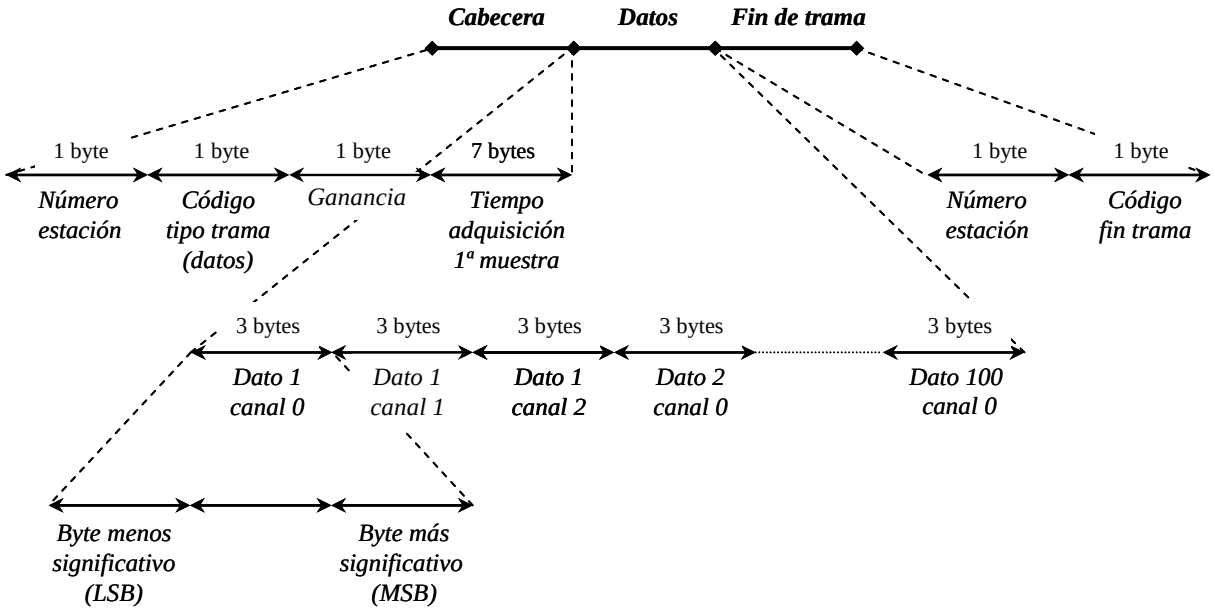
TIPOS DE TRAMA	
A. Del PC nodal a las estaciones	
	Trama de petición de trama
	Trama de petición de reenvío
	Trama de cambio de ganancia
	Trama de resincronización del reloj *software* local
	Trama de desconexión remota de una estación
	Trama de reconexión remota de una estación
B. De las estaciones al PC nodal	
	Trama de datos
	Trama de error de saturación del búffer de datos
	Trama de error de inicialización de los CADs
	Trama de error de sincronización del reloj *software* local

Tabla 3.2. *Tipos de trama en la comunicación serie.*

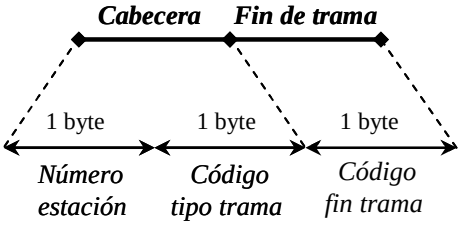
Como puede verse en la tabla 3.2, además de las tramas de petición de trama y de petición de reenvío, el PC nodal puede enviar otras tramas de control a las estaciones, ya sea para cambiar la ganancia de cualquiera de ellas, sincronizar el reloj *software* local o para desconectarlas o reconectarlas a la línea serie.

La figura 3.7 muestra el formato de las distintas tramas utilizadas en la comunicación serie. Como puede verse, todas ellas constan de una cabecera y un fin de trama, más la información correspondiente a los propios datos en el caso de tramas de datos.

a)



b)



c)

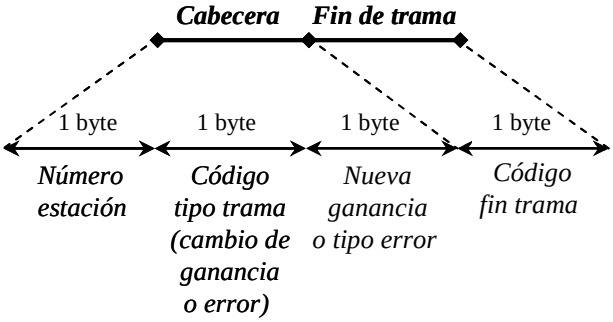


Figura 3.7. *Formato de las tramas intercambiadas en la transmisión serie. Las tramas de datos, que ocupan un total de 912 bytes, son las más largas (a). Las demás tramas sólo ocupan tres bytes (b), salvo las de cambio de ganancia y las de error, que necesitan uno más para especificar la nueva ganancia o el tipo de error, respectivamente (c).*

La cabecera de trama consiste, como mínimo, en un byte identificativo del número de estación hacia la que va dirigida o de la que procede (según el sentido de la comunicación), más un byte identificativo del tipo de trama. Las tramas de cambio de ganancia y de error necesitan un byte más que especifique la nueva ganancia o el tipo de error, respectivamente. Las tramas de datos, por su parte, incluyen, además de los dos bytes genéricos de cabecera, siete bytes correspondientes al tiempo de adquisición del primer dato de la trama, más un byte correspondiente a la ganancia de adquisición.

En todas las tramas, salvo las de datos, el fin de trama consiste en un solo carácter fijo. En las tramas de datos se incluye, además, el número de estación de procedencia. Esta información, que ya había sido incluida en la cabecera, será utilizada por el PC nodal para la verificación de la correcta recepción de la trama y confirmación de su procedencia.

3.4.4. Sincronización

Como ya se ha comentado, la obtención de resultados satisfactorios mediante la aplicación de técnicas de procesado de datos de *array* exige que los datos estén sincronizados con la suficiente precisión. En el capítulo 1 se hizo una introducción a esta cuestión, así como una breve descripción de las técnicas de sincronización empleadas en las tres antenas que se describen en esta memoria (figura 1.11). Como se recordará, en el caso del Gran Sasso el fundamento del proceso de sincronización era que todos los conversores A/D del sistema utilizaban la misma referencia temporal de 10MHz (figura 1.11.a) para garantizar que su velocidad de operación fuera idéntica y por tanto también lo fuese su frecuencia de muestreo. Por otra parte, en el capítulo 2 se presentaron los circuitos PLL y se situaron dentro del contexto de los tres dispositivos que se han diseñado, para lo cual fue necesario introducir algunos aspectos relativos a sus respectivos sistemas de sincronización. Sin embargo, tanto en el capítulo 1 como en el 2 las descripciones tenían un carácter introductorio, y por tanto muy simplificado. En los capítulos dedicados a cada uno de los *arrays* se describirán sus correspondientes sistemas de sincronización con más detalle, comenzando aquí con el de la antena del Gran Sasso.

Para conseguir la sincronización de los datos es necesario contar con alguna referencia temporal precisa, que en el caso de la antena del Gran Sasso consiste en dos señales generadas en el oscilador patrón y cuyas características se presentarán en la siguiente sección (3.4.4.1). A partir de ellas las estaciones abordan la tarea de sincronización de los datos, para lo cual, como es lógico, lo primero que debe hacerse es sincronizar el proceso de muestreo en todos los puntos de adquisición. La técnica utilizada para ello se describe en la sección 3.4.4.2. Una vez garantizada la sincronización en el muestreo, es necesario introducir información de tiempo real en las muestras. En sistemas dotados de receptores GPS la información de tiempo real se actualiza continuamente, ya sea de modo automático o a demanda del sistema de control, y se suele introducir como cabecera antes de cada bloque de datos. En la antena del Gran Sasso esta información también se inserta en forma de cabeceras en los paquetes de datos, si bien la dificultad de gestión de receptores GPS hizo aconsejable utilizar otras técnicas, que se describirán en la sección 3.4.4.3. La figura 3.8 muestra un diagrama de bloques de las tres etapas implicadas en el proceso de sincronización.

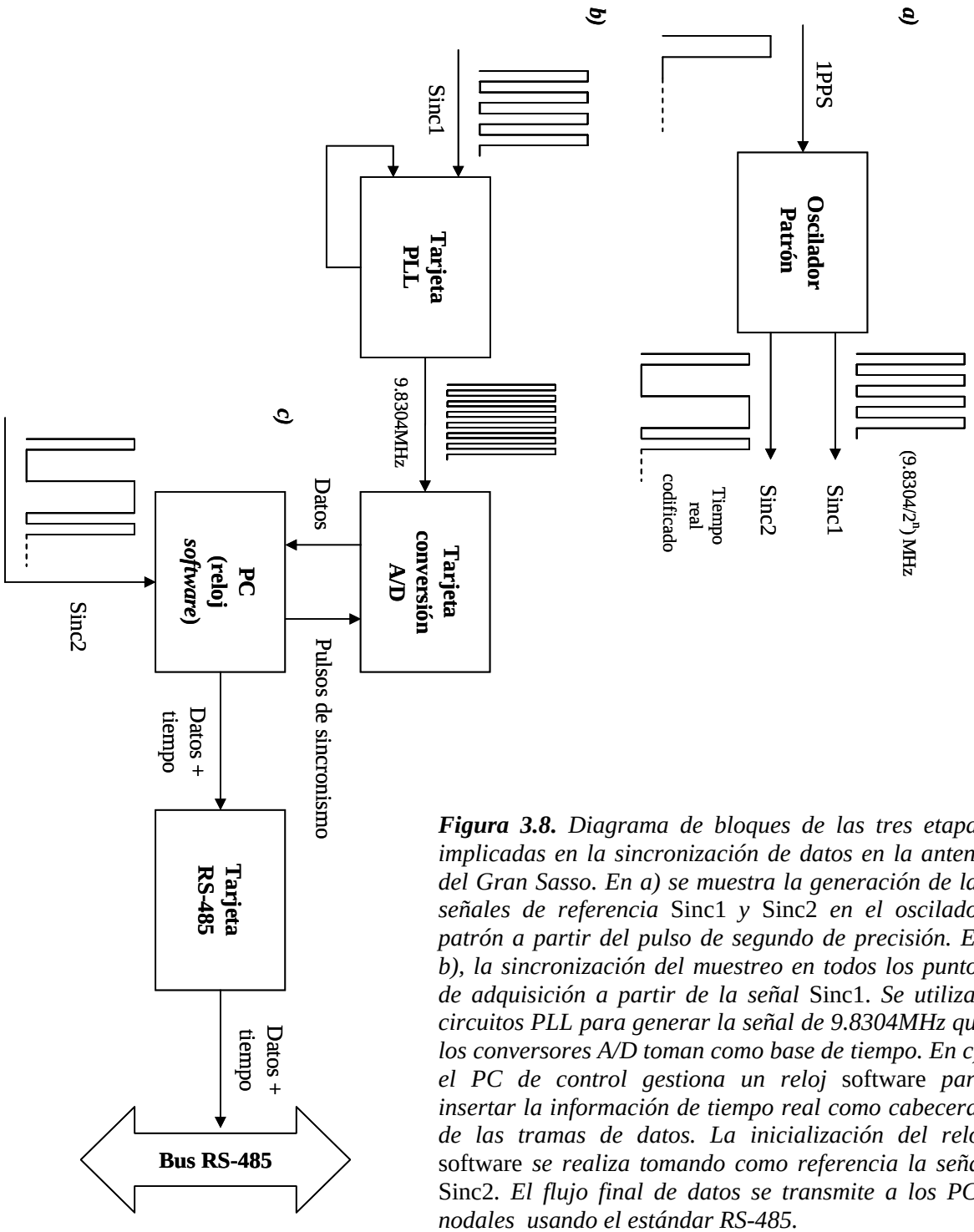


Figura 3.8. *Diagrama de bloques de las tres etapas implicadas en la sincronización de datos en la antena del Gran Sasso. En a) se muestra la generación de las señales de referencia* Sinc1 *y* Sinc2 *en el oscilador patrón a partir del pulso de segundo de precisión. En b), la sincronización del muestreo en todos los puntos de adquisición a partir de la señal* Sinc1. *Se utilizan circuitos PLL para generar la señal de 9.8304MHz que los conversores A/D toman como base de tiempo. En c), el PC de control gestiona un reloj* software *para insertar la información de tiempo real como cabeceras de las tramas de datos. La inicialización del reloj* software *se realiza tomando como referencia la señal* Sinc2. *El flujo final de datos se transmite a los PCs nodales usando el estándar RS-485.*

3.4.4.1. Señales de referencia

Las señales de referencia requeridas por las estaciones para conseguir la sincronización de los datos en la antena del Gran Sasso son generadas por el oscilador patrón. Éste necesita, a su vez, una referencia de precisión para la síntesis de las señales, que en nuestro caso es la señal de un pulso por segundo procedente de uno de los patrones atómicos operativos en los laboratorios.

Las señales de referencia generadas por el oscilador patrón son dos (figura 3.8). La primera de ellas, a la que denominaremos por simplicidad *Sinc1*, tiene cuatro posibles valores de frecuencia, correspondientes a una frecuencia nominal de 9.8304 MHz dividido por potencias de dos ($9.8304\text{MHz}/2^n$, con n = 3, 4, 5 o 6). Como se verá más adelante, la selección de uno u otro valor se lleva a cabo por *hardware*, mediante unos *jumpers* situados en una de las tarjetas del oscilador patrón. La elección del valor 9.8304 MHz se justificará en la próxima sección.

La segunda señal de referencia generada por el oscilador patrón (en adelante, *Sinc2*) consiste en una secuencia de ceros y unos que codifican la información de tiempo real según el formato que puede verse en la figura 3.9. Este formato es similar a algunos usados para la transmisión por radio de señales horarias, como los códigos DCF o IRIG (U.S. Military Standards, 2004).

3.4.4.2. Sincronización de la adquisición de datos

La sincronización del proceso de muestreo en todos los puntos de adquisición de la antena queda garantizada mediante dos factores:

1. Todos los conversores A/D usan la misma señal de reloj maestro, lo cual asegura que operan a la misma velocidad y con la misma frecuencia de muestreo.
2. Se realiza una sincronización inicial de los conversores A/D para que el primer muestreo se realice de manera simultánea en todos los puntos de adquisición.

A continuación se comentan más detalladamente ambos factores:

1. Señal de reloj maestro de los CADs.- Como se vio en el capítulo anterior, la señal de reloj maestro usada por los conversores AD7710 se introduce a través de su pin 2 (MCLK IN) (Analog Devices, 1991). Esta señal es usada por los CADs como base de tiempo en todos los procesos que éstos ejecutan, por lo que el uso de una señal común a todos ellos garantiza la misma velocidad de operación para todo el conjunto. Si además se realiza una sincronización inicial de todos los CADs, como se verá en la próxima sección, se consigue que la adquisición de datos se realice de modo simultáneo en todos los puntos de muestreo.

Un primer factor a tener en cuenta es la frecuencia necesaria para esta señal de reloj maestro, cuyo valor nominal es de 10 MHz. Este, sin embargo, no es un valor recomendable en nuestro caso[24], ya que no permite configurar los conversores a la frecuencia de muestreo deseada (100 mps) debido al procedimiento utilizado para la programación de los parámetros de funcionamiento. Como se vio en el capítulo anterior, dicho procedimiento se basa en la escritura de una palabra de veinticuatro bits en el registro de control, cuyos doce últimos bits corresponden a un código para la frecuencia de muestreo, que se obtiene según la siguiente expresión:

$$código \equiv \frac{f_{CLKIN}}{512 \times f_m} \qquad [3.1]$$

siendo f_m la frecuencia de muestreo a la que se pretende programar el dispositivo y f_{CLKIN} la frecuencia de la señal de reloj maestro. Con una frecuencia para esta señal de 10 MHz, y para conseguir una frecuencia de muestreo de 100 Hz, el código que habría que introducir en el registro de control sería:

[24] En los capítulos anteriores nos hemos referido a esta señal como ‘señal de 10MHz’ o ‘de casi 10MHz’ por simplicidad. Aquí se determina y justifica su valor exacto de frecuencia.

$$código = \frac{10MHz}{512 \times 100Hz} = 195.3125 \qquad [3.2]$$

Como se observa, el resultado no es un valor entero, con lo cual el registro de control no se puede programar exactamente a esa frecuencia de muestreo. Habría que elegir el valor entero más próximo, esto es, 195, lo que nos daría una frecuencia de muestreo de:

$$f_m = \frac{10MHz}{512 \times 195} = 100.16 \qquad [3.3]$$

Esta variación no es admisible, por lo que se decidió usar una señal de referencia temporal para los CADs de frecuencia distinta a 10 MHz, y tal que el valor resultante para el código fuera un número entero. Convenía además que el valor buscado fuera lo más próximo posible a 10 MHz, para que el comportamiento de los conversores no se alejara mucho de las características especificadas. La última condición que debía cumplir la frecuencia elegida es que fuera de uso común en sistemas electrónicos, para garantizar la disponibilidad de componentes como cristales o circuitos osciladores. La opción elegida fue f_{CLKIN} = 9.8304 MHz, valor con el que, como se puede comprobar, el código resultante para la programación de la frecuencia de muestreo es 192. El código también resulta un entero (96) para f_m = 200 Hz, con lo cual la elección de f_{CLKIN} = 9.8304 MHz deja abierta la posibilidad de utilizar el sistema con esa frecuencia de muestreo.

En primera instancia se pensó en generar la señal de reloj maestro de los CADs en el oscilador patrón y transmitirla, usando el estándar RS-485, a todas las estaciones. Sin embargo, la distancia entre el oscilador patrón y las estaciones más lejanas es demasiado grande como para poder asegurar una transmisión fiable a una frecuencia de casi 10 MHz. Por esta razón se decidió generar en el oscilador patrón una señal de frecuencia menor que se transmitiría a todas las estaciones (señal *Sinc1*). En cada estación se generará a partir de la señal recibida, mediante un circuito PLL, la definitiva de 9.8304MHz (figura 3.8). Como se verá en la sección correspondiente al diseño del oscilador patrón, el valor de la frecuencia de la señal *Sinc1* se puede seleccionar entre cuatro posibles, correspondientes a los 9.8304 MHz divididos por potencias de dos (9.8304MHz / 2^n, con n = 3, 4, 5 o 6). Por su parte, los circuitos PLL de las estaciones son capaces de generar la señal de 9.8304 MHz a partir de cualquiera de esos valores, para lo cual sólo hay que seleccionar la entrada adecuada cambiando la posición de un *jumper* en la placa correspondiente. En el montaje definitivo del sistema, la frecuencia de la señal *Sinc1* que generará el oscilador patrón se determinará en función de la calidad de recepción en las estaciones de cada una de las frecuencias posibles.

2. Sincronización de los CADs.- Como se vio en el capítulo anterior, el pin número 7 (señal /SYNC) de los AD7710s es una entrada lógica que permite la sincronización en aplicaciones en las que se utilizan varios de estos CADs. En nuestro sistema esta posibilidad se usará para la sincronización inicial de todos los CADs, así como para posteriores sincronizaciones periódicas[25]. Para ello los controladores de las estaciones utilizan como señal de referencia la hora suministrada por los relojes *software* locales (ver sección 3.4.4.3), generando, al iniciar

[25] La primera idea que surge al conocer la existencia del pin /SYNC es conectar directamente la señal de pulso de segundo del patrón atómico a este pin en todos los conversores del sistema, con lo cual éstos se mantendrían constantemente sincronizados. De esta forma se simplificaría notablemente todo el sistema de sincronismo, ya que no sería necesario utilizar la misma señal de reloj maestro en todos los CADs. Sin embargo, cada pulso en el pin /SYNC produce un reseteo de los filtros digitales del conversor, lo cual provoca que se pierdan varias muestras, por lo que en la práctica la sincronización a través del pin /SYNC sólo debe realizarse de forma esporádica.

el programa y luego a una hora fija cada día, un pulso de sincronismo a través de la línea de salida conectada a los pines /SYNC de la placa de CADs.

3.4.4.3. Información de tiempo real

Cada una de las estaciones del sistema gestiona, a través del programa de adquisición, un reloj *software* de tiempo real que se incrementa con cada nueva adquisición de un dato. La sincronización en el proceso de muestreo comentada en la sección anterior garantiza que los relojes *software* de las distintas estaciones estarán a su vez sincronizados.

El tiempo real de este reloj *software* se incluye como información en las tramas de datos, y no se pierde hasta el último paso del procesado en el servidor. Esta información se introduce, como puede verse en la figura 3.7, en la cabecera de cada trama de datos, y consiste en siete bytes correspondientes al año, día juliano, hora, minuto, segundo y fracción de segundo de adquisición del primer dato de la trama. De esta forma es posible conocer el tiempo real de adquisición de cualquier dato en cualquier punto del sistema (estaciones, PCs nodales o servidores), independientemente de factores como retardos en la transmisión serie o en la grabación en los ficheros de datos.

La inicialización de los relojes *software* se lleva a cabo mediante la señal *Sinc2* generada en el oscilador patrón. La transmisión de esta señal se realiza mediante codificación por anchura de pulsos, a razón de un bit por segundo, invirtiendo un minuto completo en la transmisión de cada cadena. Como puede observarse en la figura 3.9, sólo hay dos tipos de marcas, correspondientes al cero y al uno lógico. No es necesario introducir, como en algunos códigos similares (por ejemplo los IRIG, ver U.S. Military Standards, 2004), una marca especial de principio de código, ya que la detección puede hacerse por *software*. Los controladores de las estaciones detectarán que está empezando un nuevo minuto cuando identifiquen la cadena formada por un mínimo de 27 ceros seguidos de 4 unos, y en el siguiente flanco ascendente actualizarán sus relojes *software* locales con la información recibida. Esta serie de ceros y unos puede usarse como principio de minuto porque no ofrece posibilidad de confusión, ya que el propio código de tiempo no puede generar una secuencia tan larga de ceros.

Además de asegurar que todos los relojes *software* locales están inicialmente sincronizados entre sí, esta señal sirve para sincronizarlos con el tiempo universal, ya que los flancos ascendentes de los pulsos del código están sincronizados con los flancos ascendentes de los pulsos por segundo del patrón atómico que el oscilador patrón usa como entrada.

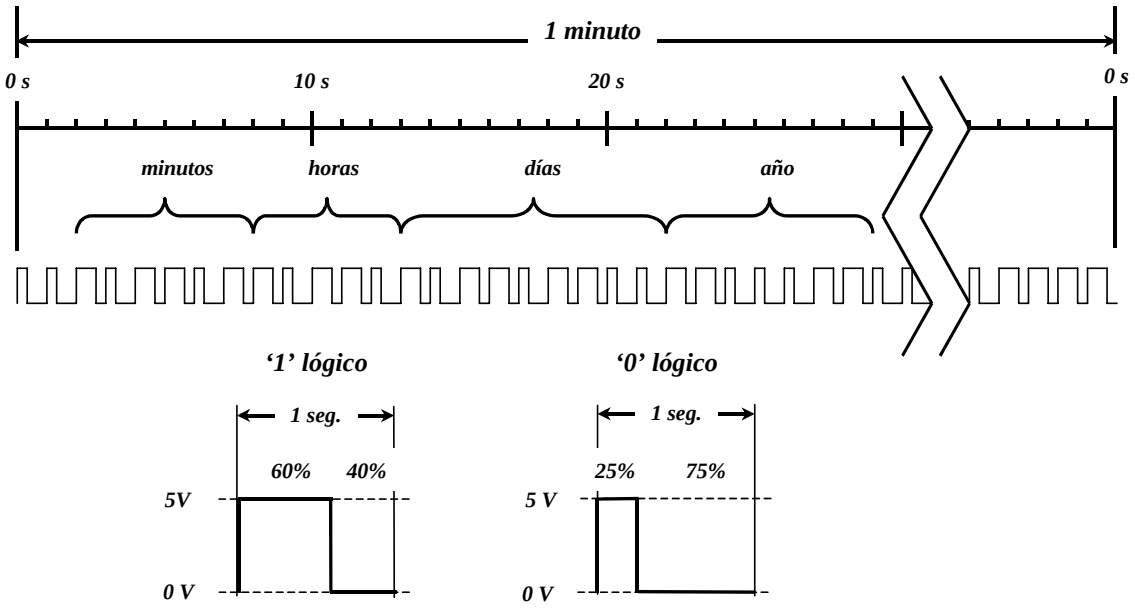


Figura 3.9. *Formato del código utilizado para la transmisión de la señal de tiempo real generada por el oscilador patrón (Sinc2). Los dos estados lógicos (0 y 1) se codifican mediante distintas anchuras de los pulsos de segundo. La información recibida por los controladores de las estaciones (minuto, hora, día juliano y año) es actualizada en el flanco ascendente del segundo cero del siguiente minuto.*

3.5. Desarrollo de la instrumentación

En esta sección se discutirán algunos aspectos relativos al diseño y funcionamiento de los circuitos que se han diseñado específicamente para esta antena sísmica, agrupados según el subsistema en el que desarrollan su función: estaciones, transmisión serie, PCs nodales u oscilador patrón. No se incluyen en esta descripción otros elementos como el servidor o dispositivos asociados a la comunicación *ethernet*, como el concentrador o las tarjetas de red, ya que estos son modelos comerciales en los que no se ha realizado ninguna modificación ni desarrollo desde el punto de vista *hardware*.

3.5.1. Estaciones

Según puede verse en el diagrama de bloques de la figura 3.10, las estaciones se componen básicamente de un sensor de tres componentes, una placa de conversores analógico/digital y una placa de PC industrial para el control de la adquisición y transmisión de los datos. A estas tarjetas se les conectan otras tres a través de distintos puertos de entrada/salida:

1. Una tarjeta para generación de la señal de reloj de los conversores A/D a partir de las señales de referencia sintetizadas por el oscilador patrón.
2. Una tarjeta de interfaz RS-232/RS-485, conectada al puerto serie COM1 del PC, para la comunicación con los PCs nodales usando el estándar RS-485.
3. Una placa de identificación de la estación dentro de la línea serie, conectada al puerto serie COM2 del PC.

Tanto los sensores como la placa de PC industrial son modelos que se adquieren comercialmente, por lo que en lo que respecta a las estaciones los circuitos desarrollados expresamente para el dispositivo que se presenta fueron tres: la placa de conversión analógico/digital, la de conversión de la señal de reloj y la placa generadora del código de estación. La tarjeta de interfaz RS-232/RS-485 se considera como parte del subsistema de comunicación serie, por lo que se describirá en la sección 3.5.2.

3.5.1.1. Placa de conversión analógico/digital

La placa de conversión analógico/digital toma como entradas las tres señales correspondientes a las tres componentes de cada sensor. Es controlada a través del puerto paralelo por el PC de cada estación, que obtiene la salida de los CADs como un código serie de veinticuatro bits. La figura 3.11 muestra el diagrama eléctrico de esta tarjeta, que consiste básicamente en tres conversores modelo AD7710, con la interfaz necesaria para controlar el funcionamiento de los mismos a través del puerto paralelo de la tarjeta de PC industrial. Para su diseño se partió de una tarjeta de adquisición de una sola componente que ya había sido probada en el Instituto Andaluz de Geofísica, con las correspondientes modificaciones para su adaptación a tres componentes.

Aparte de la utilización de líneas adicionales para gestionar la lectura de datos de los tres canales, el cambio más significativo respecto a la placa de una sola componente es el implementado para realizar la comprobación de que hay un nuevo dato listo para ser leído. Como se vio en el capítulo anterior, la señal /DRDY de los CADs se activa cuando el registro de datos se ha actualizado con un nuevo dato (Analog Devices, 1991). En la placa de una componente bastaba con consultar esta señal y proceder a la lectura del dato cuando se hubiera activado, o conectarla directamente a una línea de solicitud de interrupción *hardware* y gestionar la lectura en la rutina de servicio de la interrupción. En el caso de la placa de tres componentes, sin embargo, hay que asegurarse de que las señales /DRDY de los tres CADs están activas antes de realizar la lectura, para lo cual se implementó una puerta NAND con transistores. Las entradas de dicha puerta son las tres señales /DRDY de los CADs, y la salida la señal que se utilizará para la comprobación de dato listo. A la hora de la programación hay que tener en cuenta, por tanto, que al ser una puerta NAND esta señal se activa en alto, al contrario que las señales /DRDY originales[26].

3.5.1.2. Placa de generación de la señal de reloj

Como se comentó al hablar de la sincronización del sistema, el oscilador patrón genera dos señales, a las que denominamos *Sinc1* y *Sinc2*, que son transmitidas a cada una de las estaciones del sistema. *Sinc1* era la señal de referencia de $9.8304\text{MHz}/2^n$ (n = 3, 4, 5 o 6), que en cada estación debe utilizarse para generar la de 9.8304MHz necesaria para el funcionamiento de los CADs. *Sinc2* contenía la información de tiempo real codificada según el formato que se vio en la figura 3.9. La placa cuyo esquema se muestra en la figura 3.12 se ocupa de la gestión de las dos señales en las estaciones.

Dado que tanto *Sinc1* como *Sinc2* se transmiten usando el protocolo RS-485, el primer paso es convertirlas a niveles TTL. Para ello se utilizan transceptores de la empresa MAXIM.

[26] Teniendo en cuenta que en nuestra aplicación los tres conversores A/D de cada tarjeta están sincronizados no sería estrictamente necesario el uso de la puerta NAND. Bastaría con consultar la señal /DRDY de uno de los conversores A/D, puesto que el sistema de sincronización garantiza que el cambio se va a producir simultáneamente en los tres. Sin embargo, la implementación de una puerta lógica para obtener una señal de dato listo general resulta conceptualmente más correcta y útil en otras aplicaciones que usen esta tarjeta en las que los conversores no estén sincronizados.

Los modelos utilizados son, en concreto, el MAX490 para la señal *Sinc1* y el MAX483 para *Sinc2*. Hay que hacer notar que en el primer caso el MAX490 actúa como un repetidor, mientras que en el segundo el MAX483 está configurado como un nodo de línea (Maxim Integrated Products, 1996). El MAX490 se implementó como repetidor con el objeto de reducir la distancia de transmisión punto a punto y de ese modo asegurar una recepción fiable en todas las estaciones, pese a la relativamente alta frecuencia de la señal transmitida.

Una vez convertida a niveles TTL, la señal *Sinc1* se toma como referencia para generar la de 9.8304MHz que servirá de reloj maestro a los CADs. Para ello se utiliza, como puede verse en la figura, un circuito PLL, formado por un 74HC4046 (operando con el segundo de los tres comparadores de fase disponibles) y un contador binario 74HC4024. Ambos integrados se escogieron de la serie HC porque los CMOS convencionales no garantizaban un funcionamiento correcto a la frecuencia a la que deben trabajar en este caso.

En lo que respecta a la señal *Sinc2*, el procesado en la tarjeta que nos ocupa se reduce a la conversión de niveles descrita. La señal de salida del transceptor MAX483 se conecta directamente a una entrada del PC industrial que, como se verá en las secciones relativas a los programas, la utilizará como referencia temporal en el proceso de sincronización de los datos.

3.5.1.3. Placa de código de estación

Dentro de cada una de las líneas serie establecidas para la comunicación de datos, cada estación debe tener un número particular que permita al PC nodal correspondiente identificarla. Para desempeñar esta función se ha diseñado una pequeña placa que se conecta al segundo puerto serie de la tarjeta de PC industrial de cada estación. Se eligió el segundo puerto serie porque no tenía ninguna función asignada, a diferencia del puerto paralelo, que tenía casi todas sus líneas dedicadas al control de la placa de conversión analógico/digital.

Como puede verse en la figura 3.13, el código identificativo se introduce mediante tres microinterruptores que ponen en alto o en bajo las líneas correspondientes del puerto serie. Las tensiones positiva y negativa necesarias para la operación de la placa se consiguen poniendo a cero o uno, respectivamente, las líneas DTR y RTS del puerto serie. De esta forma no es necesario suministrar alimentación externa a la tarjeta. Además de los microinterruptores para la introducción del código identificativo, la placa incluye un pulsador que pone en alto o bajo otra de las líneas del puerto serie, y que se usará para salir del programa de adquisición.

Estas mismas placas se usarán en los PCs nodales, para asignarles números identificativos dentro de todo el sistema.

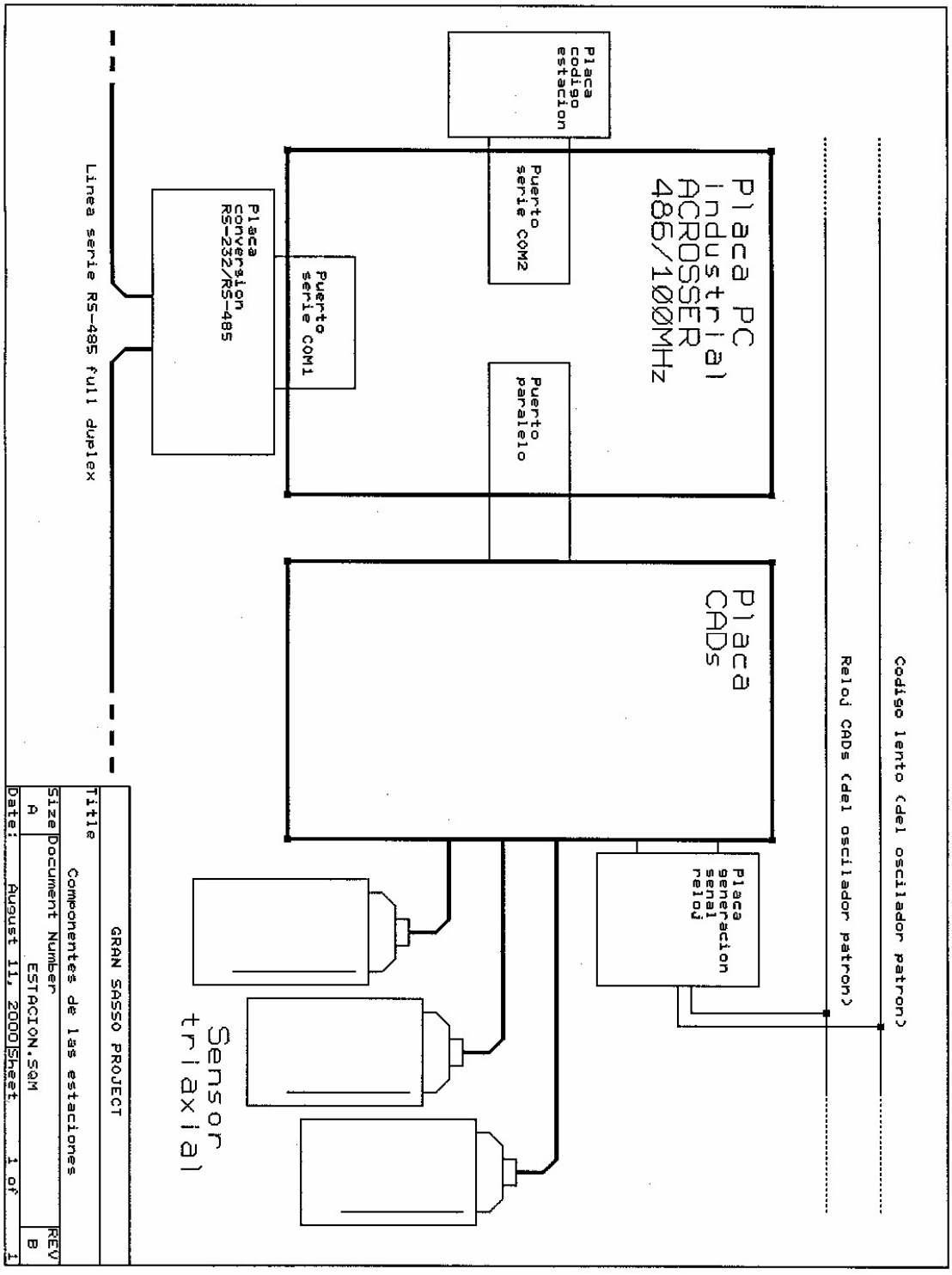


Figura 3.10. *Diagrama de bloques de las estaciones.*

a)

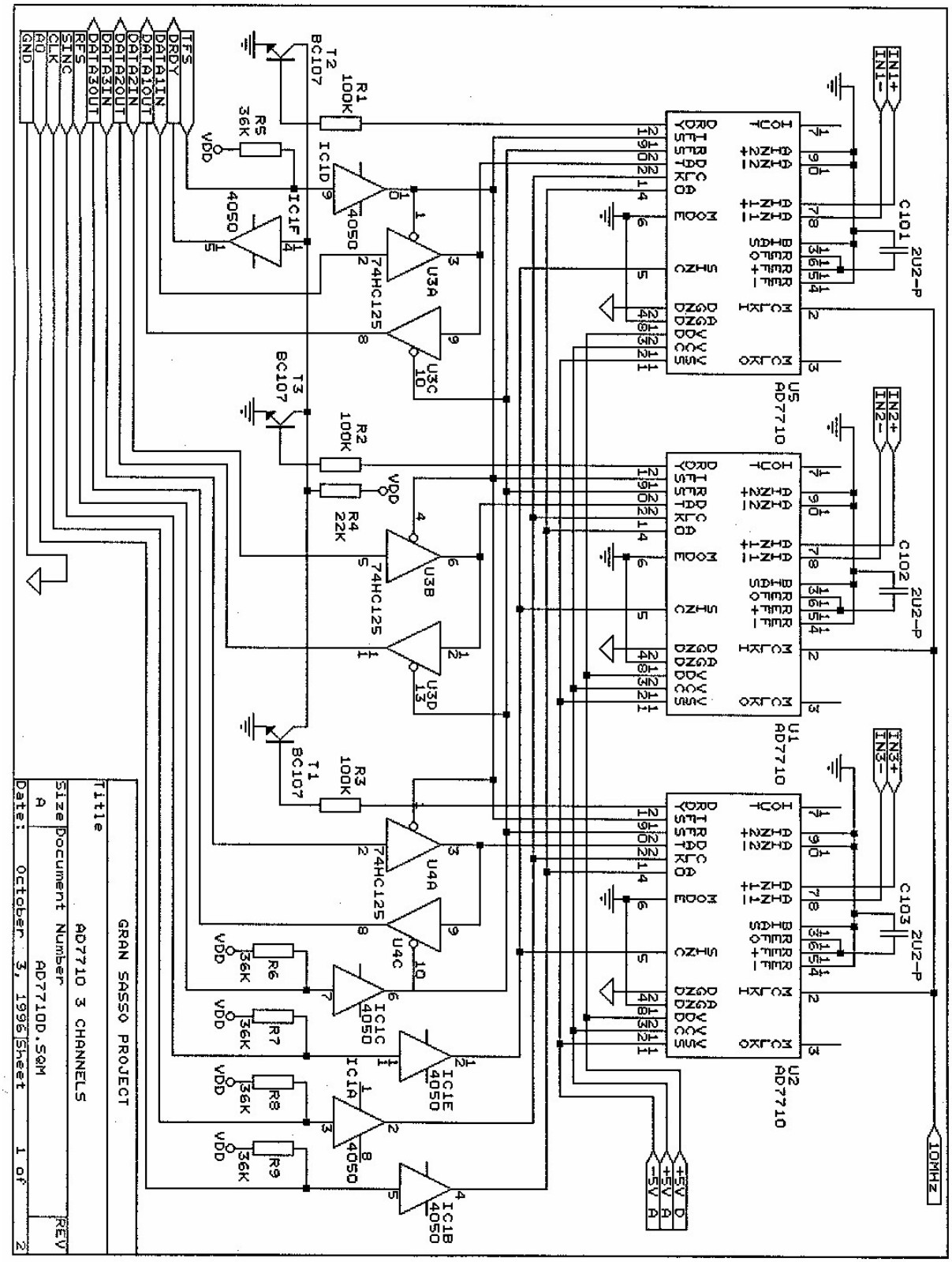
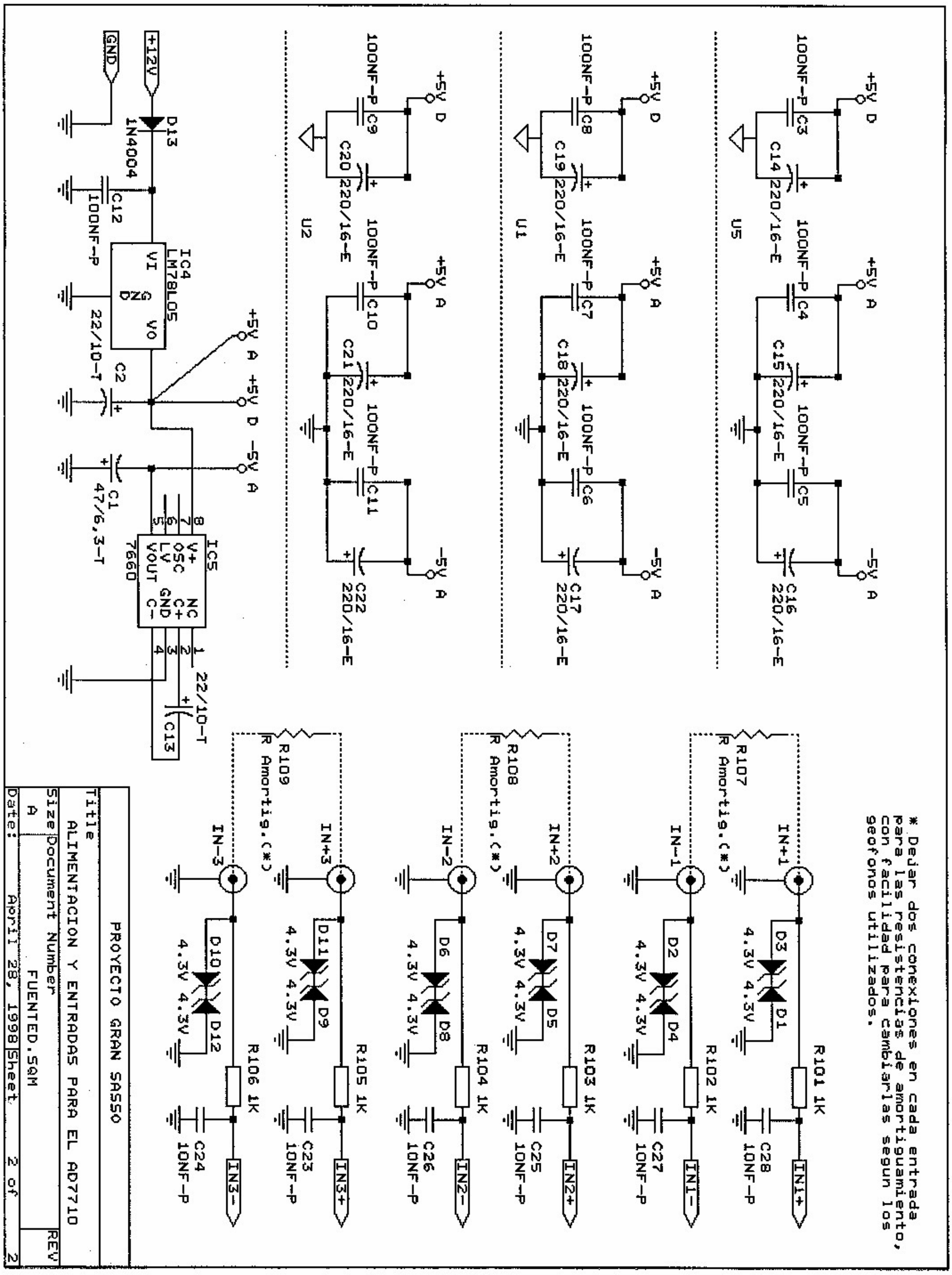


Figura 3.11. *Esquemas eléctricos de la placa de conversores analógico/digital. En a) se muestra la interfaz de control de los conversores a través del puerto paralelo del PC. En b) (página siguiente), los circuitos de alimentación y filtrado.*

3.11.b)


PROYECTO GRAN SASSO
Title
ALIMENTACION Y ENTRADAS PARA EL AD7710
Size A
Document Number
FUENTED.SGM
REV
Date: April 28, 1998
Sheet 2 of 2
* Dejar dos conexiones en cada entrada para las resistencias de amortiguamiento, con facilidad para cambiarlas segun los geofonos utilizados.
+12V
GND
D13
1N4004
C12
100NF-P
IC4
LM78L05
VI
G N D
VO
22/10-T
C2
+5V A
+5V D
-5V A
C1
47/6,3-T
IC5
7660
V+
OSC
LV
VOUT
NC
C+
GND
C-
22/10-T
C13
U1
U2
U5
220/16-E
R Amortig.(*)
R107
R108
R109
IN+1
IN-1
IN+2
IN-2
IN+3
IN-3
4.3V
R101 1K
R102 1K
R103 1K
R104 1K
R105 1K
R106 1K
10NF-P
IN1+
IN1-
IN2+
IN2-
IN3+
IN3-

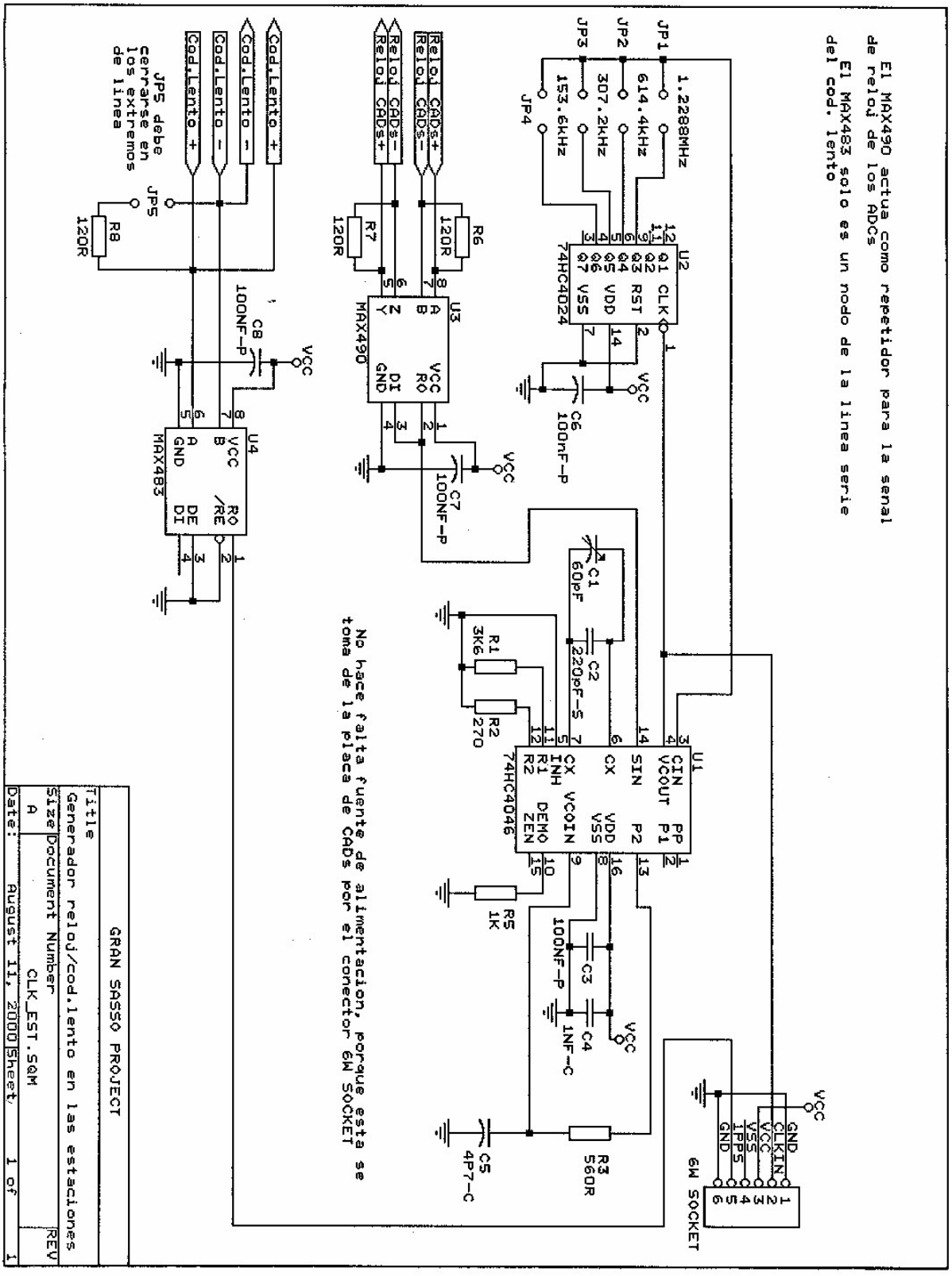


Figura 3.12. *Esquema eléctrico de la placa de generación de las señales de reloj en las estaciones.*

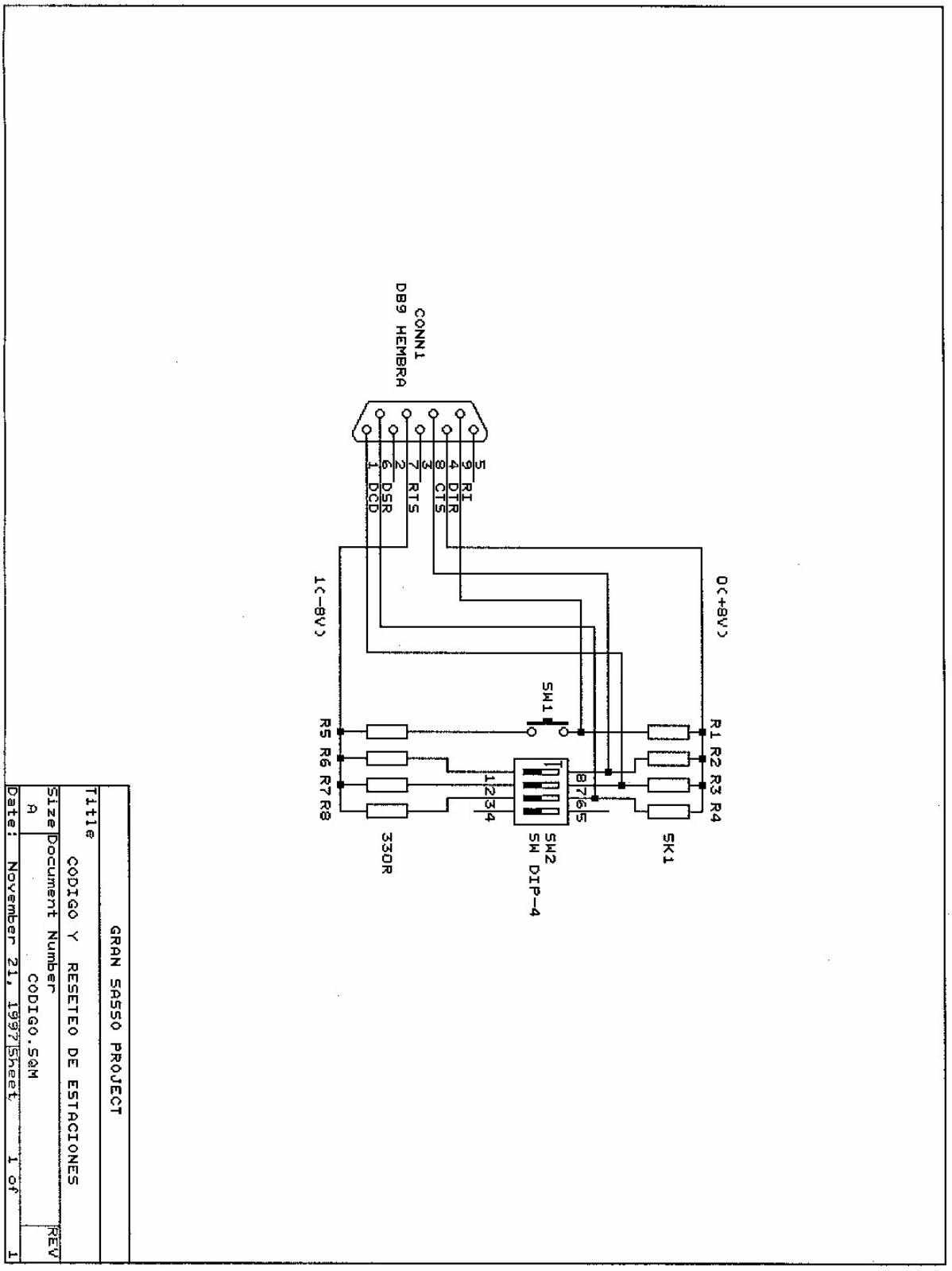


Figura 3.13. *Esquema eléctrico del circuito generador del código identificativo de las estaciones y PCs nodales.*

3.5.2. Interfaz serie

3.5.2.1. Placa de conversión RS-232/RS-485

Los dos puertos serie de las tarjetas de PC industrial utilizadas son de tipo RS-232, por lo que para conectarlas a las líneas de comunicación serie es necesario realizar una conversión de los niveles lógicos. Con este fin se han diseñado tarjetas adaptadoras RS-232/RS-485, que realizan en primer lugar la conversión de niveles RS-232 a TTL, y a continuación transforman los niveles TTL a RS-485.

Para la primera conversión se ha utilizado un integrado MAX232A, que es compatible pin a pin con el estándar MAX232 y permite una mayor velocidad de transmisión (hasta 200kbps, frente a los 120kbps del MAX232). En cuanto al montaje, sólo difiere del MAX232 en el valor de los condensadores necesarios.

Para la conversión de niveles TTL a RS-485 se ha utilizado el MAX489, un integrado de la familia de transceptores para RS-485 de MAXIM (Maxim Integrated Products, 1996) con las siguientes características:

- Realiza la transmisión-recepción en modo *full-duplex*.
- Presenta limitación en la velocidad de respuesta (*slew-rate*) de la señal, lo cual limita la frecuencia de operación a 250 kbps.
- Tiene señales de control para la conmutación entre los estados de transmisión y recepción.

En la figura 3.14 se muestra el esquema eléctrico de las placas de conversión RS-232/RS-485. Como puede verse, las líneas de transmisión y recepción de datos del puerto serie se convierten a niveles TTL a través del MAX232A, y luego a niveles RS-485 mediante el MAX489. Para el control de la transmisión-recepción se ha utilizado una sola señal (DTR del puerto serie), ya que la habilitación-deshabilitación de la transmisión o recepción se llevará a cabo alternativamente. Las señales de control de transmisión-recepción del MAX489 deben tener niveles TTL, por lo que la línea DTR se ha pasado también previamente por el MAX232A.

Por otro lado, pueden verse también las terminaciones de línea necesarias para evitar reflexiones y garantizar una buena transmisión (Goldie, 1996). Estas terminaciones únicamente deben utilizarse en los nodos inicial y final de cada línea serie, por lo que se han dispuesto los *jumpers* JP1 a JP4 para habilitarlas sólo en caso necesario.

3.5.3. PCs nodales

Como puede verse en el diagrama de bloques de la figura 3.15, los componentes principales de los PCs nodales son la propia placa de PC industrial y la placa de red. Ambos dispositivos se adquieren comercialmente, por lo que los únicos componentes de diseño propio en esta etapa del sistema fueron las placas de conversión RS-232/RS-485 y las placas de código, ambas ya descritas anteriormente.

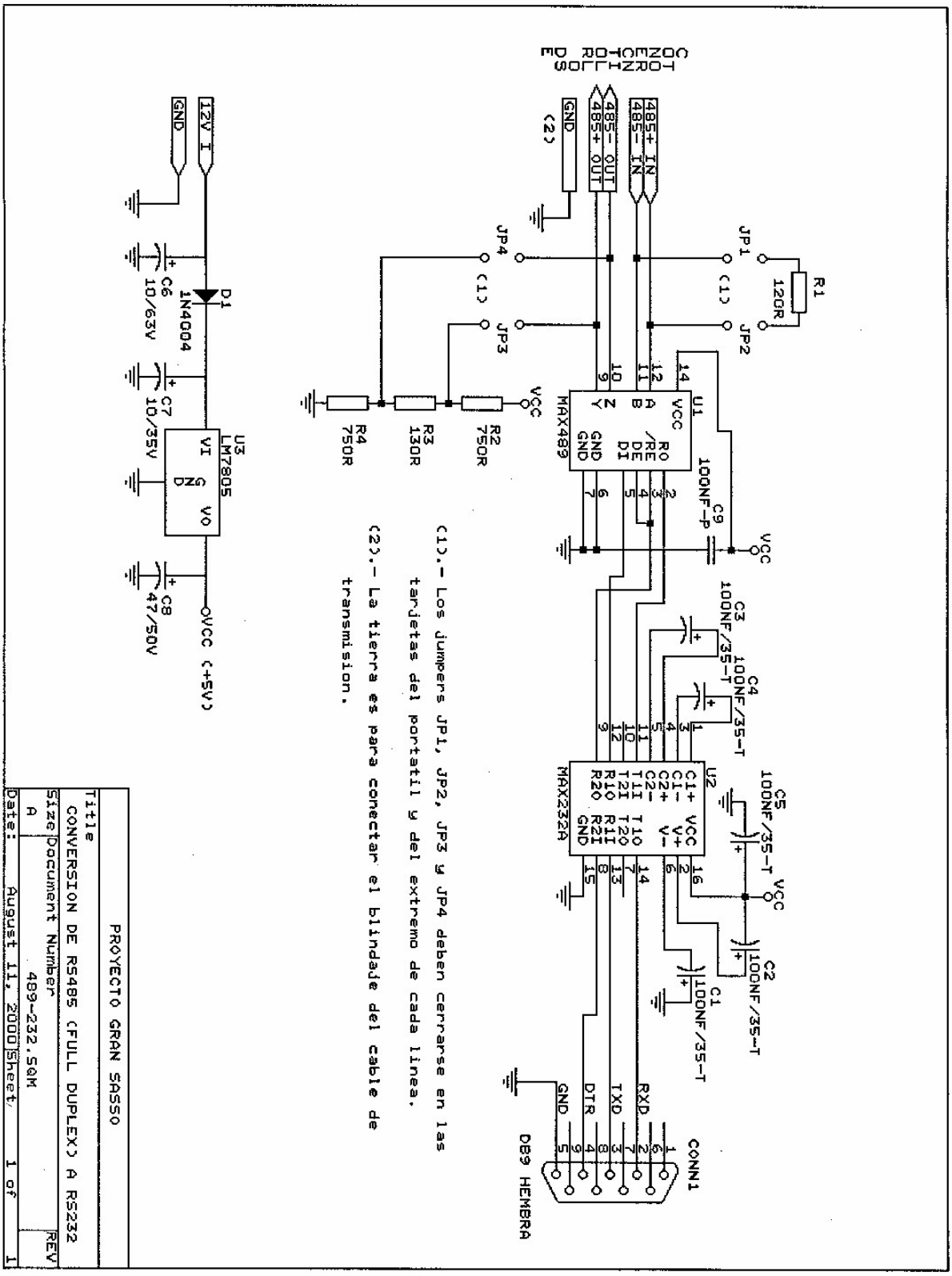


Figura 3.14. *Esquema eléctrico del circuito convertidor de niveles RS-232 a RS-485.*

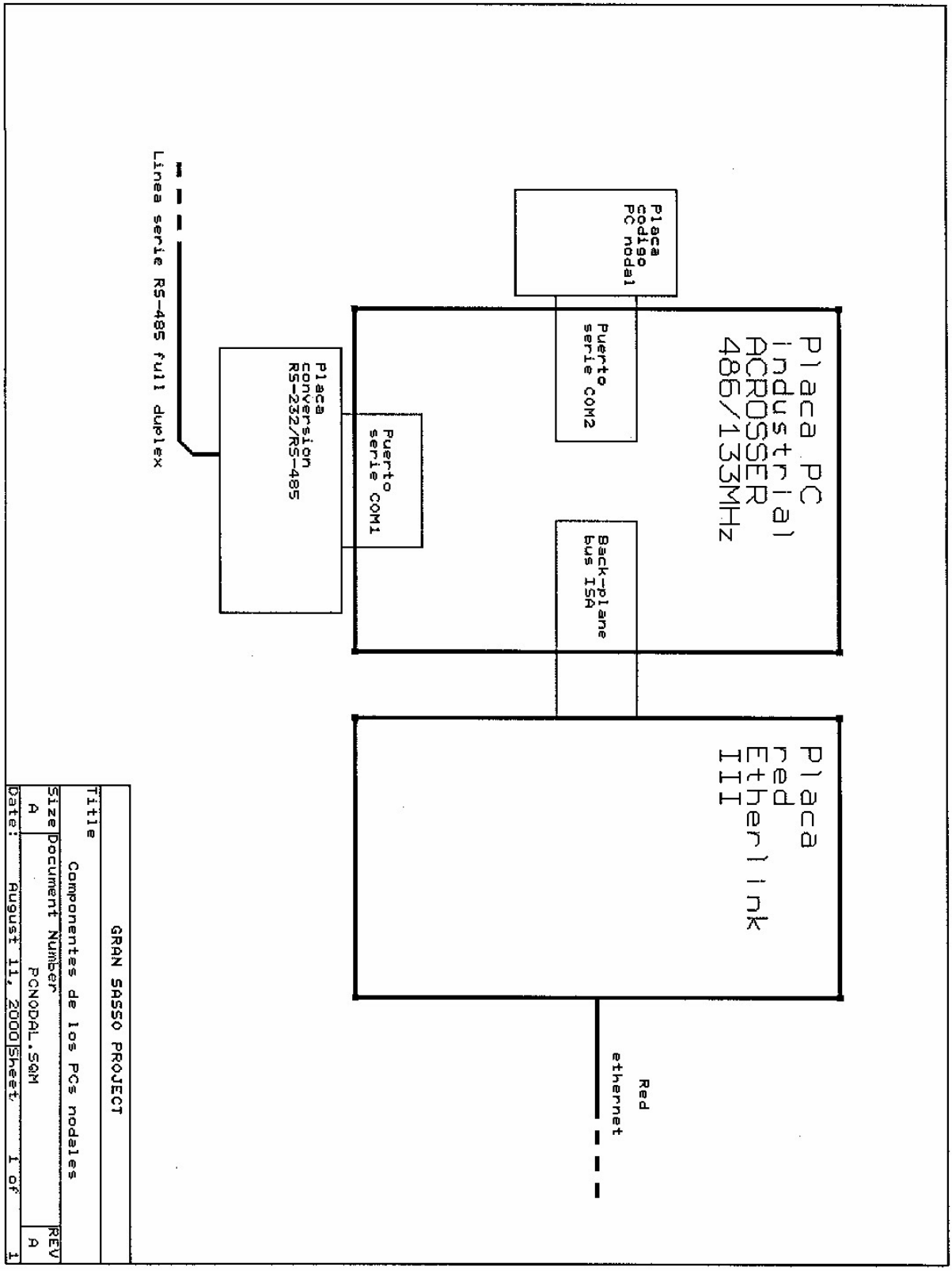


Figura 3.15. *Diagrama de bloques de los PCs nodales.*

3.5.4. Oscilador patrón

Para generar las dos señales *Sinc1* y *Sinc2* comentadas en el apartado de sincronización del sistema se diseñaron tres placas distintas, correspondientes a la interfaz serie, al oscilador a cristal controlado por tensión (VCXO) y al circuito del microcontrolador PIC16C74 (figura 3.16). Para la fuente de alimentación de todos los componentes del oscilador patrón se eligió un modelo comercial de entrada 220V AC y salida 12V DC, implementando luego en cada una de las placas los reguladores lineales necesarios para convertir los 12V a las tensiones de alimentación requeridas (normalmente 5V).

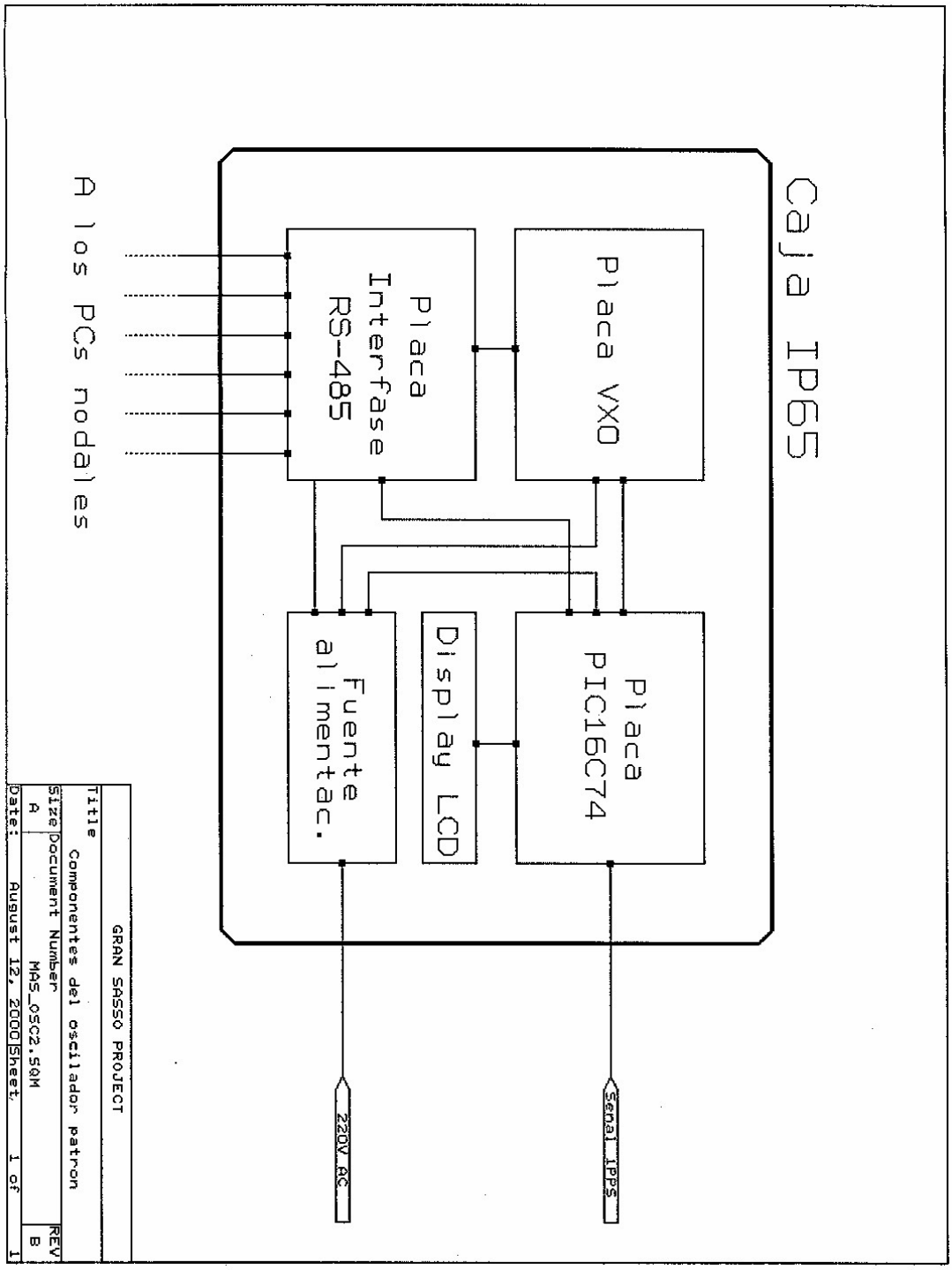


Figura 3.16. *Diagrama de bloques del oscilador patrón.*

Como puede verse en las figuras 3.8 y 3.16, el oscilador patrón toma como entrada la señal de un pulso por segundo del reloj atómico y da como salidas las señales de sincronización *Sinc1* y *Sinc2*, que se transmiten a todas las estaciones del sistema. Para evitar tener que realizar tendidos adicionales de cable, en primera instancia las señales se transmiten a los seis PCs nodales, cada uno de los cuales las hace llegar a las estaciones correspondientes a sus líneas serie siguiendo el mismo trazado que las líneas de datos.

El diseño se realizó en tres placas distintas con el fin de ocupar el mínimo espacio posible. Por otra parte, también interesaba desarrollar el circuito del PIC16C74 como una placa independiente que pudiera ser utilizada en futuros diseños con este microcontrolador.

Las placas se hicieron en formato rectangular de pequeño tamaño (8 x 8.5cm), con el objeto de poder montar una encima de otra y ocupar un espacio mínimo. De esta forma todos los componentes del módulo, incluida la fuente de alimentación y el display, se pueden montar en una caja de pequeñas dimensiones.

3.5.4.1. El microcontrolador PIC16C74

En el capítulo de materiales y métodos se dio una visión general de la familia de microcontroladores PIC, utilizados en varios de los subsistemas de las tres antenas que se describen en esta memoria. Aquí se hará una revisión más detallada de las características más relevantes del modelo elegido como núcleo del oscilador patrón, el PIC16C74.

1. Características_generales.- El PIC16C74 es un microcontrolador de la gama media de la familia (ver sección 2.3.2.2), cuyas principales características son:

- Tecnología EPROM CMOS de alta velocidad y bajo consumo.
- Programable en código ensamblador propio de 35 instrucciones.
- Longitud de las instrucciones: 14 bits
- Memoria de programa: 4K (x 14)
- Memoria de datos: 192 bytes
- Velocidad de operación: reloj de entrada de hasta 20 MHz, que da un ciclo de instrucción de 200 ns.
- Posibilidad de programación con protección de código.
- Modo SLEEP para ahorro de consumo.
- Amplio margen de tensión de alimentación (2.5V-6.0V).
- Temporizador tipo perro guardián (*watchdog timer*) con su propio oscilador RC para una operación más fiable.
- 33 pines de entrada/salida
- Puerto paralelo de 8 bits, con señales externas de control /RD, /WR y /CS.
- Puerto serie síncrono (SSP) con SPI e I2C.
- Transmisor-receptor universal síncrono-asíncrono (USART).
- 3 módulos controladores de tiempo, configurables en modos de contador (síncrono o asíncrono) y timer.
- 2 módulos CCP (Captura/Comparación/PWM) que funcionan en conjunción con los módulos controladores de tiempo.
- Capacidad de gestión de interrupciones, con 12 fuentes generadoras de las mismas.
- Conversor analógico-digital de ocho bits multicanal.
- Se presenta en encapsulados cerámicos de 40 o 44 pines.

Las características detalladas de este modelo pueden consultarse en las hojas de características (Microchip Technology Inc., 1997), accesible desde el propio sitio web del fabricante[27]. Aquí se hará una breve presentación de las características que afectan al diseño de las tarjetas y a la programación del microcontrolador para nuestra aplicación concreta. Así, en la siguiente sección se describe la organización de la memoria en el PIC16C74, con la que conviene estar familiarizado para entender la metodología de programación de este tipo de microcontroladores y la estructura del programa de control del oscilador patrón. Por otra parte, teniendo en cuenta que la principal tarea que debe desempeñar este subsistema (la generación de las señales de referencia *Sinc1* y *Sinc2)* se fundamenta en los módulos de Captura/Comparación/PWM (CCP) del PIC16C74 y que la operación de éstos está basada en los módulos controladores de tiempo, también se explicará el funcionamiento de ambos.

2. Organización de la memoria.- Como todos los microcontroladores de la familia PIC, el 16C74 usa una arquitectura Harvard, en la que se utilizan memorias distintas para el programa y para los datos, así como distintos buses para acceder a cada una de ellas. Esto implica un aumento del ancho de banda en relación con la arquitectura tradicional de Von Neumann, en la que tanto programas como datos se guardan en la misma memoria y a los que se accede a través del mismo bus. La utilización de buses distintos para los datos y el programa permite, además, que las instrucciones tengan una longitud distinta a los 8 bits de las palabras de datos. En este caso, la longitud de las instrucciones es de 14 bits, y permite que todos los códigos de instrucción sean de una sola palabra. El bus de acceso a la memoria de programa de 14 bits hace posible la búsqueda de una instrucción en cada ciclo. Esto, en conjunción con una técnica de procesamiento en canal (*pipeline*) de dos etapas, que solapa la búsqueda de una instrucción con la ejecución de la anterior, permite que todas las instrucciones se ejecuten en un solo ciclo, excepto las instrucciones de salto que requieren dos.

Memoria de programa.- La familia de los PIC16C7x tiene un contador de programa de 13 bits, capaz de direccionar un espacio de memoria de 8K x 14. En todos los dispositivos que cuentan con menos de 8K de memoria de programa (como el PIC16C74), intentar acceder a una posición de memoria por encima de la físicamente implementada provocará un acceso cíclico a las posiciones existentes.

Para la gestión de las interrupciones se utiliza un solo vector de interrupción, situado en la posición 0004h de la memoria de programa. Por su parte, el vector de reset se encuentra en la posición 0000h.

La figura 3.17 muestra la organización de la memoria de programa en el PIC16C74.

Memoria de datos.- La memoria de datos en la familia de los PIC16C7x está particionada en varios bancos que contienen los registros de propósito general y los de funciones especiales, que son registros usados por la CPU y los módulos periféricos para controlar la operación del dispositivo. La selección del banco al que se desea acceder se realiza a través de los bits RP1 y RP0 del registro STATUS.

Cada banco tiene 128 bytes, cuyas posiciones inferiores están reservadas para los registros de función especial. Es posible acceder a algunos de los registros de este tipo más utilizados desde más de un banco, para aumentar de este modo la velocidad de acceso y reducir el tamaño del código. Por encima de los registros de función especial se encuentran los registros de propósito general, implementados como RAM estática.

Los dispositivos de la familia PIC16C7x con tamaño máximo de memoria de datos cuentan con cuatro bancos, de los que 368 bytes corresponden a registros de propósito general y el resto a registros de función especial.

[27] www.microchip.com

El PIC16C74, por su parte, cuenta con dos bancos de memoria de datos, de los que 192 bytes pueden ser utilizados como registros de propósito general. El resto corresponde a registros de función especial.

La figura 3.18 muestra la organización de la memoria de datos en el PIC16C74, incluyendo todos los registros de función especial.

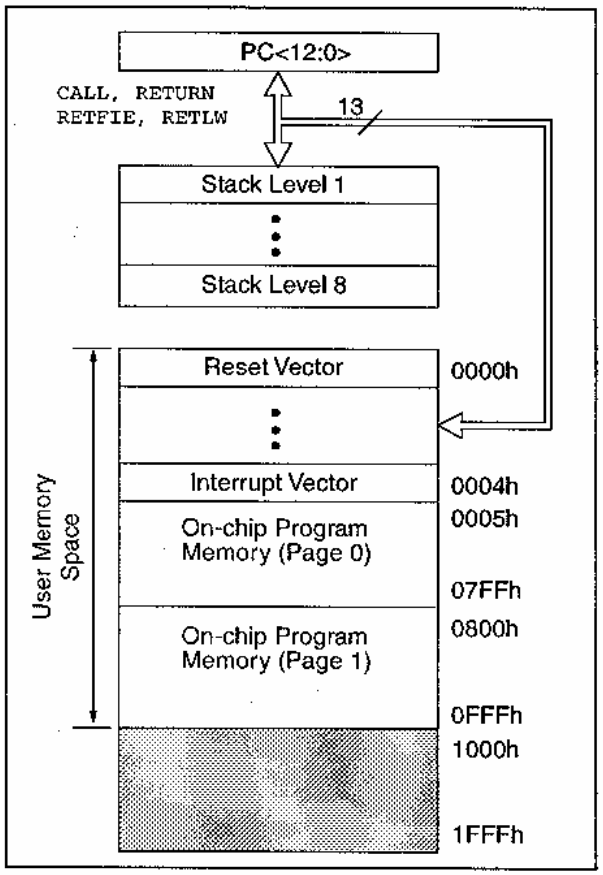


Figura 3.17. *Mapa de la memoria de programa y pila en el PIC16C74 (de Microchip Technology Inc., 1997).*

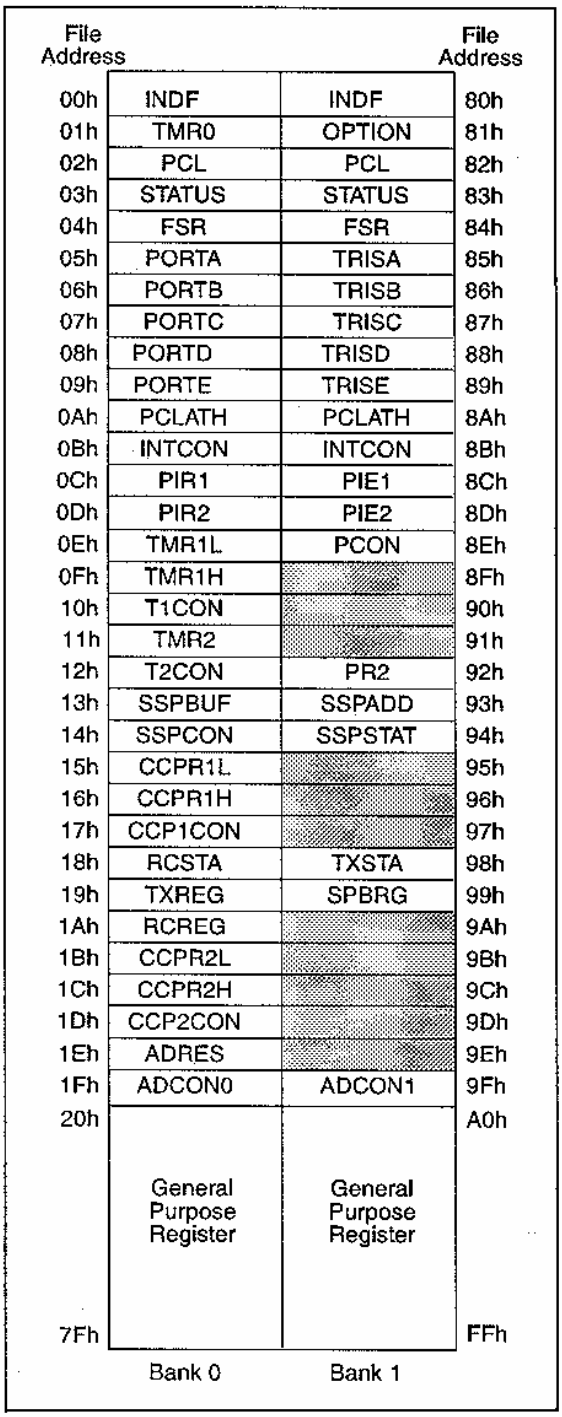


Figura 3.18 (derecha). *Mapa de la memoria de datos y registros de función especial en el PIC16C74. Los espacios sombreados representan posiciones de memoria no implementadas (de Microchip Technology Inc., 1997).*

3. Módulos controladores de tiempo.- El PIC16C74 tiene tres módulos controladores de tiempo, con distintas características:

El módulo 0 es simplemente un contador de desbordamiento de 8 bits, que puede funcionar con una fuente externa de reloj o con el reloj interno del sistema ($f_{OSC}/4$). El módulo 1 puede funcionar en modo de timer o de contador, en ambos casos de 16 bits. La fuente de reloj puede ser el reloj interno del sistema, un reloj externo o un cristal externo. Por último, el módulo 2 es un timer de 8 bits que cuenta con un registro de periodo (PR2), orientado al uso con el modo PWM de los módulos CCP.

Los tres módulos cuentan con distintas opciones de prescalador (*prescaler*) y/o postescalador (*postscaler*), que permiten programar los incrementos de los contadores con distintas relaciones de la frecuencia de entrada. Cualquiera de los tres módulos puede ser programado para generar una interrupción cuando se produzca un evento, como un desbordamiento del contador.

4. Módulos de Captura/Comparación/PWM (CCP).- El PIC16C74 tiene dos módulos CCP, cada uno de los cuales puede ser programado en tres modos distintos: modo de captura, modo de comparación y modo de PWM (modulación por anchura de pulsos). Los módulos CCP operan en conjunción con los módulos controladores de tiempo, según se muestra en la siguiente tabla:

Modo CCP	*Timer*
Captura	Timer1
Comparación	Timer1
PWM	Timer2

El modo de captura toma el valor de 16 bits del par de registros TMR1H:TMR1L y lo introduce en el par de registros CCPRxH:CCPRxL ('x' corresponde al número de módulo CCP que estemos utilizando). La captura del valor puede producirse con un flanco ascendente, un flanco descendente, uno de cada cuatro o uno de cada dieciséis flancos ascendentes de la señal introducida a través del pin CCPx. La elección del tipo de captura se realiza a través del registro CCPxCON.

El modo de comparación compara el contenido del par de registros TMR1H:TMR1L con el de los registros CCPRxH:CCPRxL. Cuando ambos valores coinciden, se pueden programar varias opciones: generar una interrupción, poniendo además el pin CCPx en un estado determinado (alto o bajo), resetear el TMR1 (módulo CCP1), o resetear el TMR1 y empezar una conversión A/D (módulo CCP2). La elección de una u otra opción se realiza a través del registro CCPxCON.

El modo PWM compara el registro TMR2 con un registro de *duty-cycle* de 10 bits, formado por el registro CCPRxH y los bits 4 y 5 del registro CCPRxL. También compara el registro TMR2 con un registro de 8 bits donde se introduce el periodo deseado para la señal PWM (registro PR2). Cuando el contenido de TMR2 iguala el del registro de *duty-cycle*, el pin CCPx se pone a cero, mientras que cuando iguala el de PR2, CCPx se pone a uno. En este último caso, además, TMR2 se pone a 00h, y se puede generar una interrupción si se ha programado previamente. De este modo, jugando con los valores introducidos en el registro de *duty-cycle* y en el PR2 se puede conseguir una señal modulada en anchura de pulsos con la frecuencia y *duty-cycle* deseados, y con una resolución de hasta 10 bits.

3.5.4.2. Placa del microcontrolador PIC16C74

La primera tarea que debe realizar el PIC es el control de la frecuencia de la señal de reloj de los CADs (*Sinc1*). Para ello, se utiliza el módulo CCP1 programado en modo captura, en conjunción con el timer 1. En este modo, el PIC cuenta el número de pulsos que entran por el pin RC0 durante el intervalo de tiempo determinado por un número prefijado de pulsos a través del pin RC2. En función de la diferencia entre el número contado de pulsos y el correcto el PIC actúa sobre el VCXO. Para ello utiliza la salida del segundo módulo CCP, que debe programarse en el modo PWM (anchura modulada de pulsos). Para poder operar de esta forma, las conexiones deben ser como se muestran en la figura 3.19: la salida del VCXO a la entrada del módulo CCP1 (pin RC0), la salida del módulo CCP2 (pin RC1) a la entrada del VCXO, y la entrada de pulsos de referencia del módulo CCP1 (pin RC2) a la señal de un pulso por segundo del reloj atómico que se utiliza como referencia de tiempo.

La segunda tarea del PIC es gestionar el reloj de tiempo real del sistema, para lo cual usa también como referencia la señal de un pulso por segundo del reloj atómico, y mantiene un reloj *software* que transmite codificado a todas las estaciones (señal *Sinc2*). Para ello, la salida del código lento de tiempo real (pin RC3) está conectada a la placa de interfaz serie RS-485.

Además de estas conexiones, la placa del PIC implementa una interfaz para display alfanumérico, que es controlado a través de ocho bits de datos (pines RD0-RD7) y tres de control (RA1-RA3). La alimentación del display también se toma de esta misma placa.

Los pulsadores SW2, SW3 y SW4 se usarán para tareas de configuración y sincronización manual del módulo, como se describirá en la sección 3.7.3, mientras que SW1 es el pulsador de reinicio del microcontrolador.

3.5.4.3. VCXO

La figura 3.20 muestra el esquema eléctrico del circuito diseñado para operar como VCXO del oscilador patrón. Como puede verse, es un circuito oscilador por inversores (4049), en el que la regulación por tensión se consigue a través de dos diodos varactores modelo BA102. La frecuencia de la señal que se transmitirá a las estaciones se selecciona mediante uno de los *jumpers* JP1 a JP4, que selecciona una de las salidas del divisor de frecuencia 4024. Sin embargo, la señal que se conecta al PIC16C74 y que se utilizará para corregir la frecuencia del VCXO se toma siempre de la salida Q1 del 4024, por lo que estará centrada en torno a la mitad de la frecuencia del cristal XT1 (2.4576 / 2 = 1.2288 MHz).

El microinterruptor SW1A permite realizar un ajuste del oscilador, para lo cual debe ponerse en la posición de ‘ajuste’, que fija la tensión de entrada a la mitad de la máxima (2.5 V). A continuación se actúa sobre el *trimmer* C7, hasta que la frecuencia de salida del oscilador coincida con la del cristal (2.4576 MHz). Después, para funcionamiento normal, SW1A debe volver a ponerse en la posición de ‘lazo cerrado’.

Para obtener el margen de ajuste del VCXO, primero debe realizarse un ajuste como se ha indicado. Luego hay que poner SW1A en posición de ‘lazo cerrado’ y fijar la tensión de entrada a 0 V y a 5 V para obtener la mínima y máxima frecuencia de oscilación, respectivamente. Los resultados obtenidos fueron:

V_{IN}	f_{OUT}
0 V	2.457546 MHz
2.5 V (tensión de ajuste)	2.457600 MHz
5 V	2.457620 MHz

Por lo que el margen de ajuste vale:

$$M.A. = ((2.457620 - 2.457546) / 2.4576) \times 10^6 = 30.1 \text{ ppm} \qquad [3.4]$$

Como se comentó en las consideraciones de diseño, una de las características del entorno (LNGS) es que su temperatura se mantiene prácticamente constante. Por tanto, no es necesario que el control del oscilador compense las derivas de frecuencia de origen térmico, y es suficiente con el margen de ajuste conseguido.

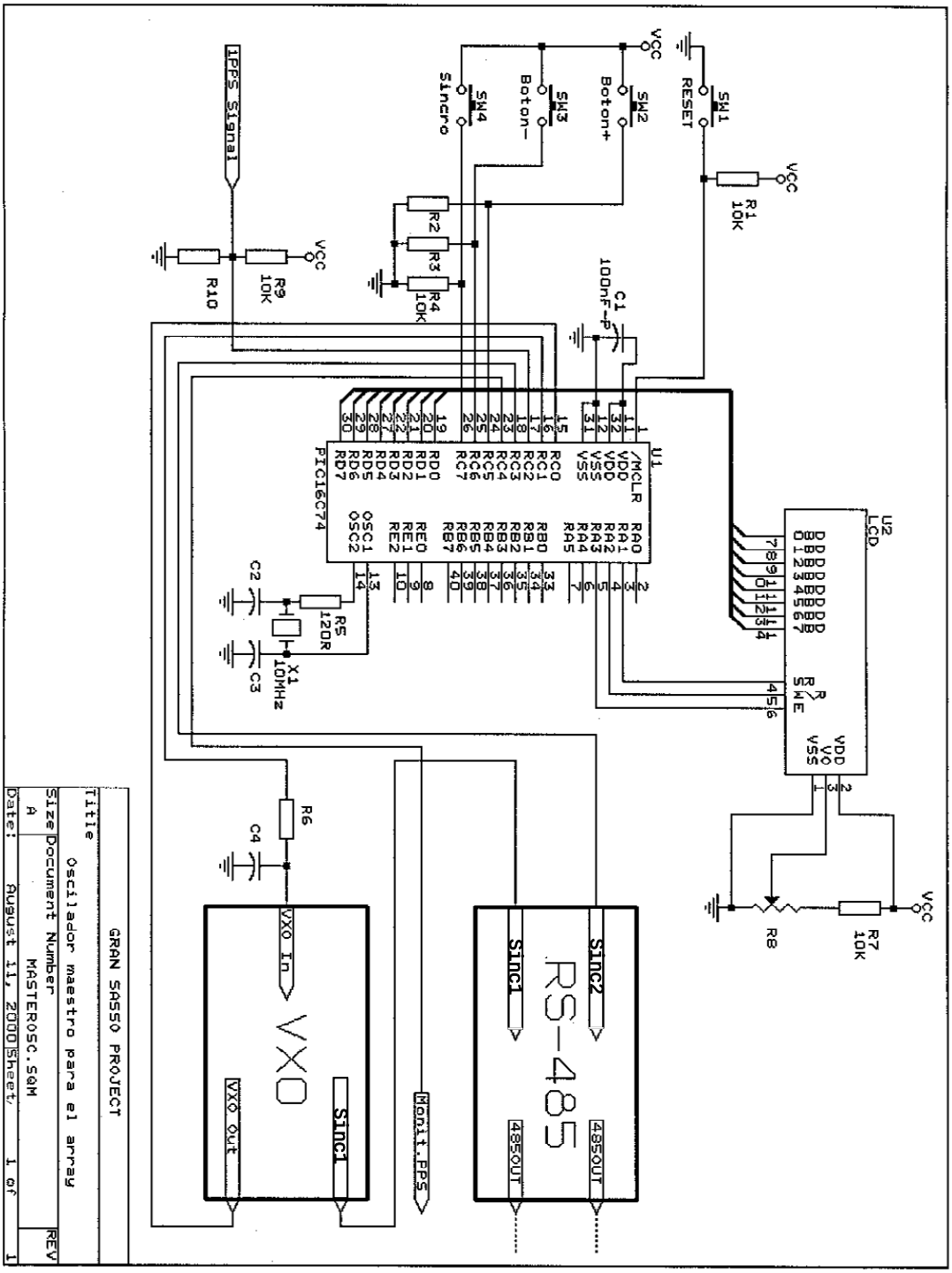


Figura 3.19. *Esquema eléctrico del circuito del PIC16C74, que controla la operación del oscilador patrón.*

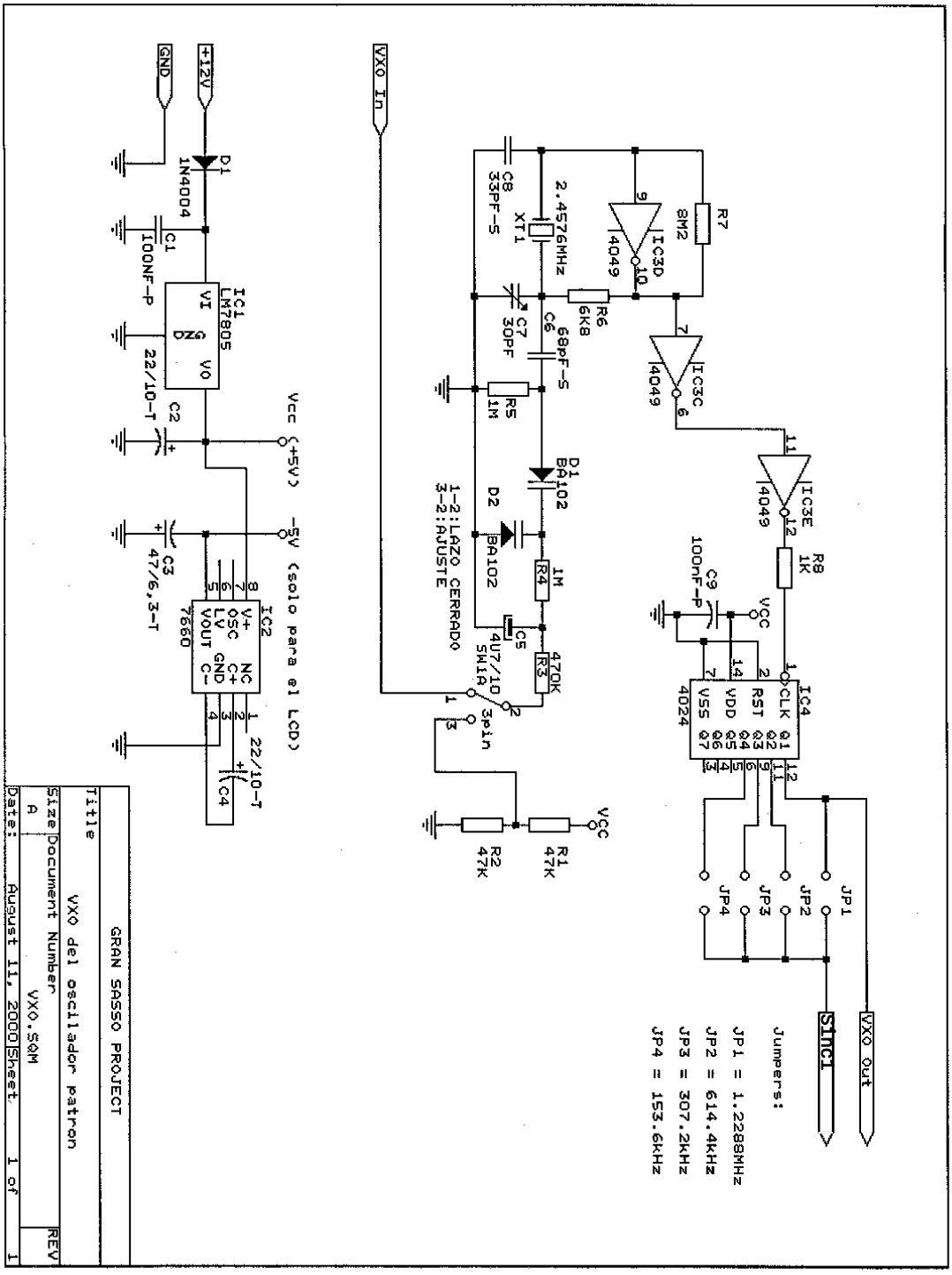


Figura 3.20. *Esquema eléctrico del circuito oscilador a cristal controlado por tensión (VCXO) del oscilador maestro.*

3.5.4.4. Interfaz serie

El circuito diseñado para operar como interfaz serie con las estaciones se muestra en la figura 3.21. Se trata, en realidad, de seis interfaces distintas, una para cada una de las líneas serie del sistema. Cada una de las interfaces consiste en dos transceptores de RS-485, uno para la transmisión de la señal *Sinc1* y el otro para *Sinc2*. Previamente ambas señales se buferizan mediante un 4050 para asegurar una buena calidad a la entrada de los transceptores de RS-485. La señal *Sinc2*, al ser de baja frecuencia, permite utilizar transceptores con limitación del *slew-rate*, pero no sucede lo mismo con *Sinc1*. En ambos casos la comunicación sólo tiene que ser unidireccional, por lo que no hay necesidad de utilizar integrados para modo *full-duplex* como ocurría en la transmisión de datos. Según estas consideraciones, los modelos utilizados fueron el MAX485 para la transmisión de la señal de reloj de los CADs (*Sinc1*) y el MAX483 para la señal de reloj codificada (*Sinc2*) (ver Maxim Integrated Products, 1996).

3.6. Programación de los distintos módulos

La programación de los controladores de los distintos módulos que se han presentado se realizó siguiendo los procedimientos descritos en el capítulo 2. En este dispositivo fueron cuatro los programas desarrollados específicamente:

- ESTACION.C: programa de adquisición y transmisión serie de datos, que corre de modo continuo en las estaciones.

- PC_NODAL.C: programa de control de la transmisión serie y grabación en disco remoto de los datos, que corre de modo continuo en los PCs nodales.

- CONTROL.C: Programa de control del sistema (cambio de parámetros de operación, consulta del estado del sistema, visualización de los ficheros de datos) que es ejecutado en el servidor cuando el usuario lo solicita.

- OSCILAD.ASM: Programa de generación de las señales de temporización y sincronización, que corre de modo continuo en el oscilador patrón.

Aparte del programa de control del sistema desarrollado específicamente (CONTROL.C), el servidor puede utilizar diversos programas y utilidades comerciales o de diseño propio, dependiendo del tipo de análisis y procesado que se desee llevar a cabo con los datos. Las versiones de los programas que se presentan en esta memoria realizan la grabación de los datos en el servidor en ficheros circulares (es decir, ficheros de longitud fija que se sobrescriben cuando se llega al final) cuya estructura se describirá más adelante. En ese caso se asume que el servidor está corriendo de forma continua y en tiempo real un programa de detección de eventos basado, por ejemplo, en un algoritmo STA/LTA (Lee y Stewart, 1981). La detección se realizaría a partir de los ficheros circulares, si bien no sería necesario tener en cuenta los datos de todas las estaciones operativas, sino sólo los correspondientes a algunas estaciones significativas, cuyo número y localización serían seleccionables a través del programa de control del sistema. En caso de producirse un disparo, se grabarían en ficheros de evento los datos de los ficheros circulares correspondientes a todas las estaciones operativas en el momento del mismo, empezando un tiempo antes (preevento) y finalizando un tiempo después (postevento) del propio disparo.

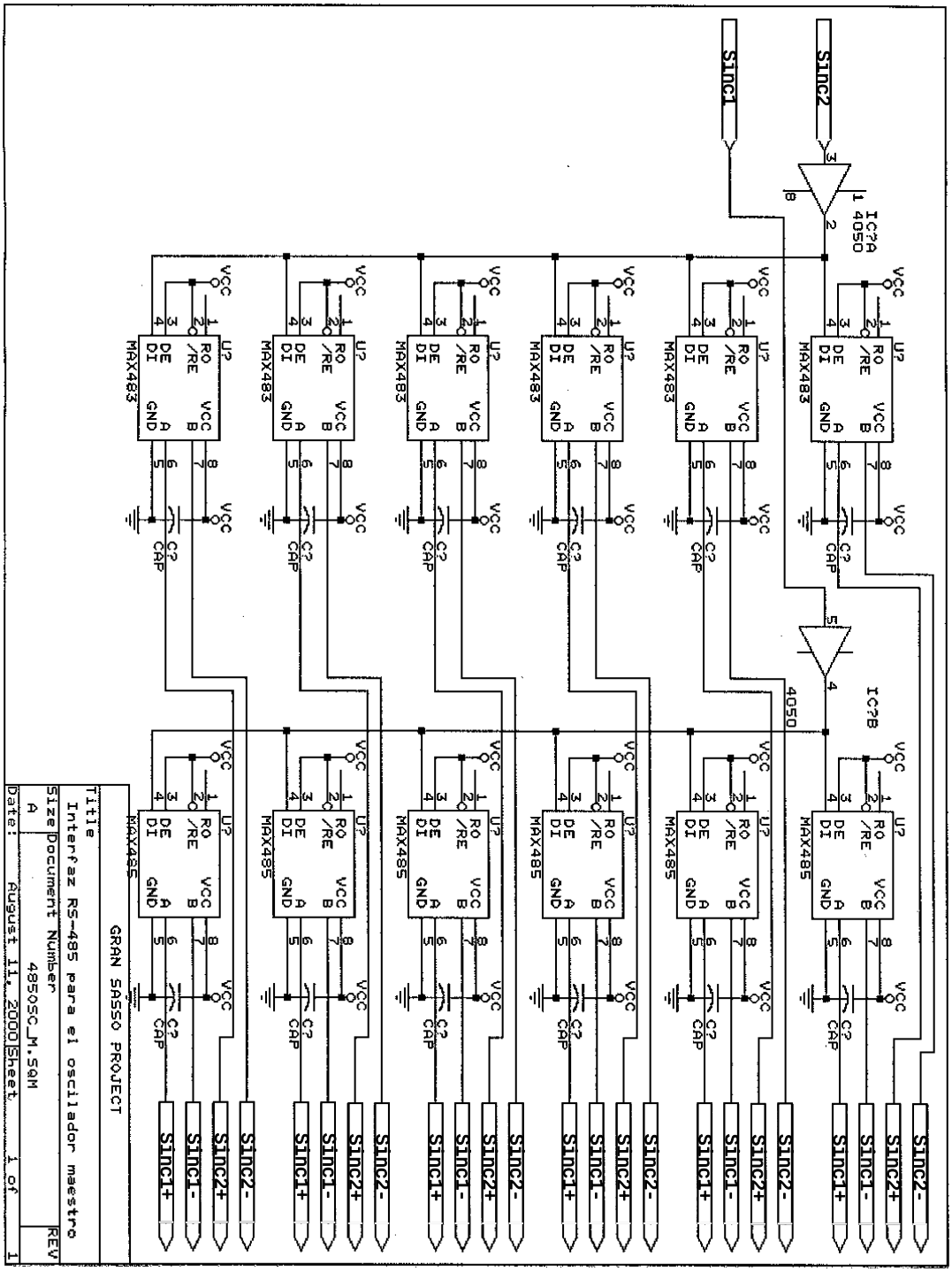


Figura 3.21. *Esquema eléctrico de la tarjeta de interfaz serie RS-485 del oscilador patrón.*

La continua evolución en los sistemas informáticos, tanto en términos de capacidad de los medios de almacenamiento como en velocidad de los procesadores y memorias, hace cada vez más factible la grabación continua de datos, incluso en dispositivos con elevado número de canales y resolución alta como el que se presenta en este capítulo. En lugar de llevar a cabo la detección de eventos en tiempo real para conservar únicamente los ficheros de evento, la tendencia actual es grabar en dispositivos de almacenamiento masivo la totalidad de los datos. La detección de eventos se puede realizar a posteriori, tomando los datos de los ficheros grabados, o en tiempo real, en cuyo caso los datos que disparan el algoritmo de detección se guardan por duplicado, en ficheros de evento y en los ficheros de continuo. La grabación continua permite realizar estudios de fenómenos asociados a cambios lentos de la señal sísmica, lo cual resulta particularmente interesante si en el futuro se equipa el dispositivo con sensores de banda ancha o muy ancha, según se comentó en las consideraciones de diseño. Los cambios necesarios en los programas que se presentan para conseguir la grabación continua serían mínimos, orientados principalmente a nombrar los ficheros de datos según una nomenclatura predeterminada que debería incluir la fecha y hora de adquisición de los datos y un código identificador del número de estación en la que se adquirieron.

A continuación se describen la estructura y funcionamiento de cada uno de los programas desarrollados específicamente para esta antena. Los listados completos se encuentran en el anexo I.A, incluido en el CD adjunto.

3.6.1. Programa para las estaciones (ESTACION.C)

El programa ESTACION.C debe ocuparse de varias tareas:

- Control de los CADs, que incluye su inicialización y la gestión de la adquisición de los datos.

- Formación de tramas de datos y envío de las mismas.

- Gestión del reloj *software* local. Esto incluye la inicialización, chequeo y, si fuera necesario, resincronización del mismo, usando para ello la señal de tiempo real que se recibe codificada del oscilador patrón (*Sinc2*).

- Comprobación del correcto funcionamiento del proceso y envío de tramas de error en caso de producirse alguno.

Para conseguir estos objetivos, el programa se planteó según los diagramas de flujo que se muestran en la figura 3.22. Como puede verse en el primero de ellos, la gestión de la adquisición de los datos se realiza a través de la interrupción IRQ7, para lo cual la señal DRDY procedente de la placa de los CADs se conecta a la entrada de solicitud de dicha interrupción, que se encuentra en el pin 10 del puerto paralelo (señal /ACK, ver figura 3.11 y tabla 3.3). Además de la adquisición de los datos, en la rutina de servicio de la interrupción IRQ7 se incrementan las fracciones de segundo del reloj *software* de la estación. En caso de que este incremento provoque a su vez un incremento en los segundos (lo que significa que la muestra adquirida es la primera de una nueva trama de datos), se introduce toda la información de tiempo real del reloj *software* (año, día juliano, hora, minuto, segundo y fracción) como cabecera de la nueva trama.

a)

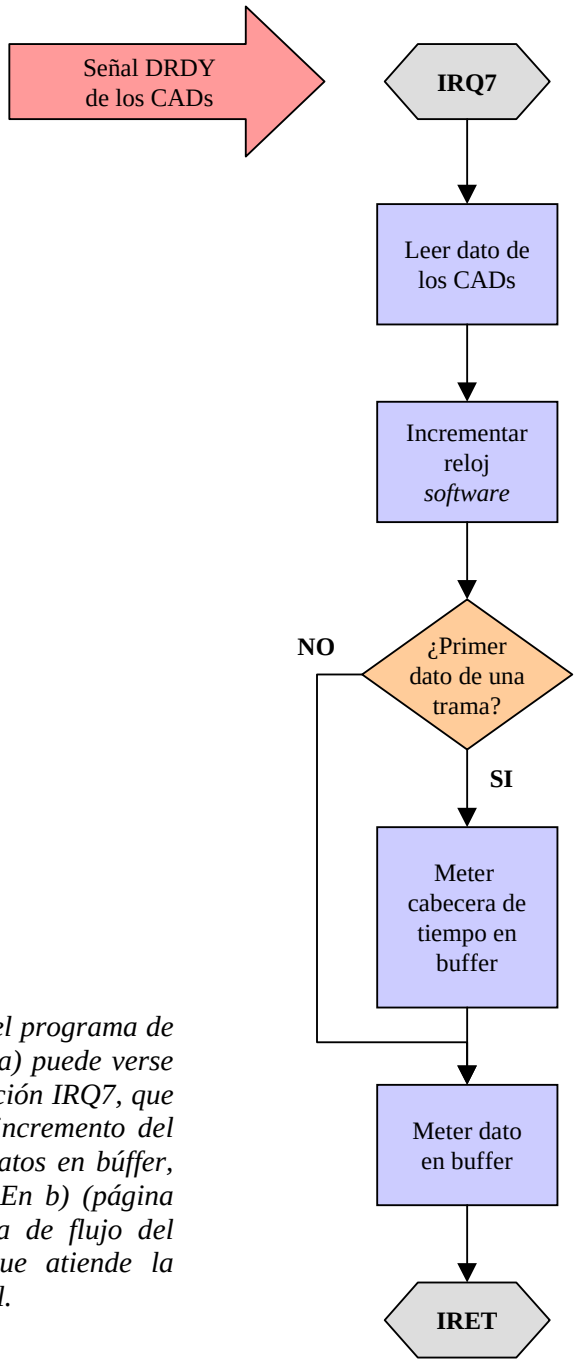
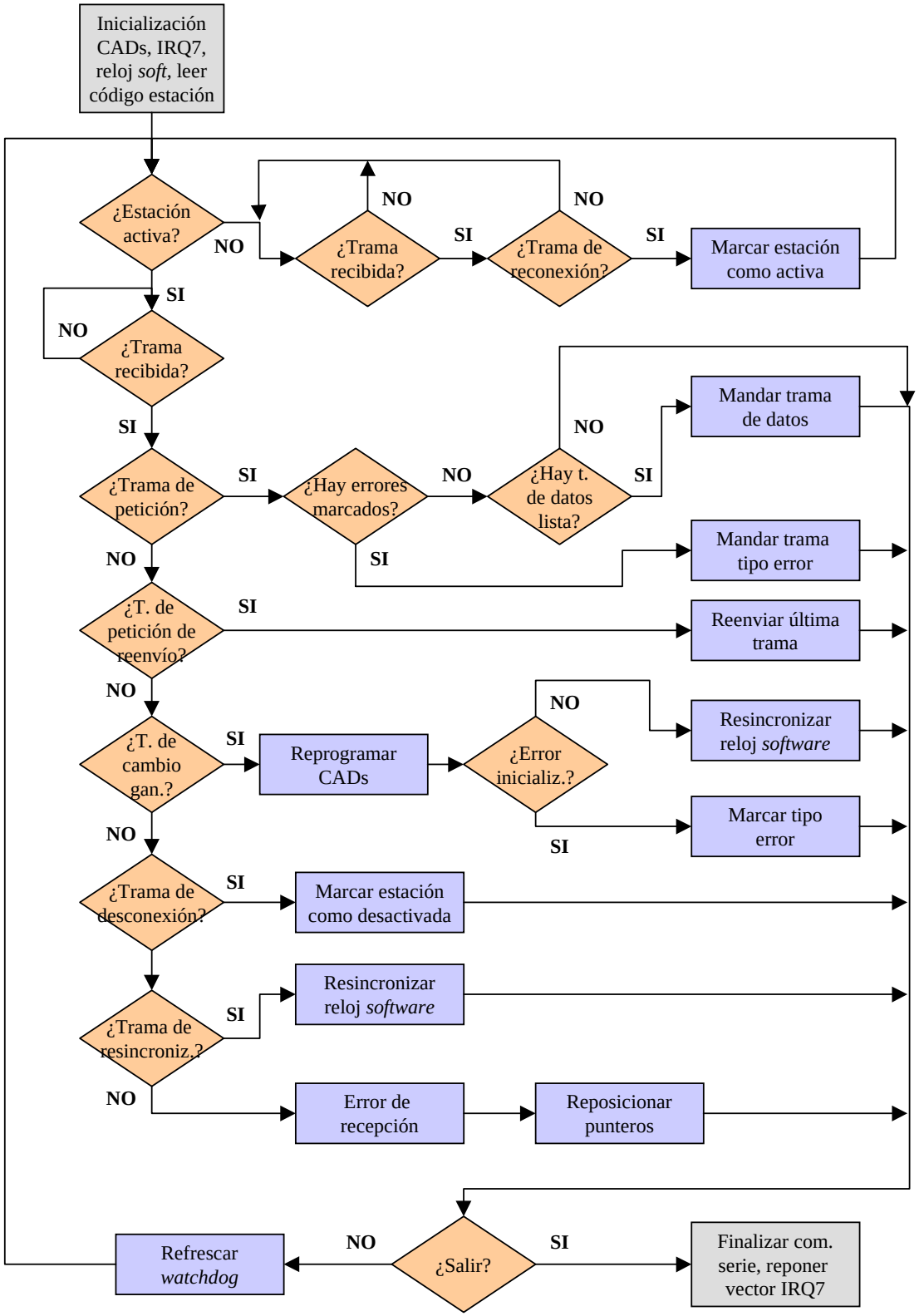


Figura 3.22. *Diagramas de flujo del programa de las estaciones (ESTACION.C). En a) puede verse el diagrama de flujo de la interrupción IRQ7, que se ocupa de la lectura de datos, incremento del reloj* software *y escritura de los datos en búffer, formando tramas de un segundo. En b) (página siguiente) se muestra el diagrama de flujo del bucle principal del programa, que atiende la comunicación serie con el PC nodal.*

3.22.b)


Inicialización CADs, IRQ7, reloj *soft*, leer código estación
¿Estación activa?
NO
¿Trama recibida?
NO
SI
¿Trama de reconexión?
NO
SI
Marcar estación como activa
SI
NO
¿Trama recibida?
SI
¿Trama de petición?
SI
¿Hay errores marcados?
NO
¿Hay t. de datos lista?
NO
SI
Mandar trama de datos
SI
Mandar trama tipo error
NO
¿T. de petición de reenvío?
SI
Reenviar última trama
NO
¿T. de cambio gan.?
SI
Reprogramar CADs
¿Error inicializ.?
NO
Resincronizar reloj *software*
SI
Marcar tipo error
NO
¿Trama de desconexión?
SI
Marcar estación como desactivada
¿Trama de resincroniz.?
SI
Resincronizar reloj *software*
NO
Error de recepción
Reposicionar punteros
¿Salir?
NO
SI
Refrescar *watchdog*
Finalizar com. serie, reponer vector IRQ7

Nombre señal	*Pin puerto paralelo*	*Nombre señal puerto paralelo*
SCLK	17	/SELECT IN
A0	14	/AUTO FD XT
/RFS	16	/INIT
/TFS	3	DATA 1
/SYNC	4	DATA 2
DRDY	10	/ACK
D1IN	2	DATA 0
D2IN	2	DATA 0
D3IN	2	DATA 0
D1OUT	11	BUSY
D2OUT	12	PE
D3OUT	13	SELECT
GND	18-25	GROUND
Sinc2	15	/ERROR

Tabla 3.3. *Correspondencia entre las señales de la placa de conversión A/D y las del puerto paralelo utilizadas para el control de la adquisición de datos. Las señales de entrada de datos de los tres canales (D1IN, D2IN y D3IN) son controladas por la misma señal del puerto paralelo, dado que los tres conversores de cada placa se van a programar siempre con los mismos parámetros. Por tanto estas tres señales tienen que estar cortocircuitadas en el conector o en la propia placa de CADs.*

Para la programación de la IRQ7 deben llevarse a cabo los procedimientos normales en la programación de interrupciones *hardware*, es decir:

- Programación de una rutina de gestión de interrupción, que debe incluir el código que se desea ejecutar más las sentencias que informen a los chips controladores de interrupciones del final de la rutina.

- Habilitación de la interrupción en el registro de máscara de interrupciones y desvío del vector correspondiente al iniciar el programa, para posibilitar que al producirse una interrupción se ejecute el código programado.

- Reposición del vector de interrupción original al salir del programa, para restaurar la situación inicial.

Además de estos procedimientos generales en la gestión de las interrupciones, en el caso de la IRQ7 al iniciar el programa también debe habilitarse ésta en el correspondiente registro del puerto paralelo (Austerlitz, 1991).

Aparte de la inicialización de las interrupciones, el programa debe realizar otras tareas antes de entrar en el bucle principal, como son (figura 3.22.b):

- Inicialización de los CADs, que comprende la sincronización, autocalibración y escritura de los parámetros de operación en el registro de control. Todo esto se lleva a cabo según los diagramas de tiempo y otras indicaciones de las hojas de características de los CADs (Analog Devices, 1991).

- Inicialización del puerto serie.

- Inicialización del reloj *software* local, a partir de la señal *Sinc2* transmitida por el oscilador patrón.

- Lectura del código de estación. Como ya se comentó en el desarrollo de la instrumentación, el código se introduce a través de tres microinterruptores implementados en una pequeña placa conectada al segundo puerto serie del PC industrial de cada estación.

- Realización de una primera lectura de datos introduciendo las funciones correspondientes explícitamente. Cuando la adquisición de datos se realiza con estos CADs a través de interrupciones, como en nuestro caso, es necesario hacer esto después de haber inicializado los CADs y la interrupción, para hacer que la señal DRDY se desactive y al volver a activarse dispare la interrupción.

Una vez realizadas las inicializaciones, el programa ESTACION.C debe ocuparse básicamente de la gestión de la comunicación serie con el PC nodal. Como puede verse en la figura 3.22.b, siempre que la estación esté activa el programa se mantiene a la espera hasta que recibe una trama del PC nodal. Cuando esto ocurre, se pasa a la identificación de la trama y a la correspondiente actuación según el tipo de trama de que se trate.

Si la identificación es negativa, y no se reconoce ninguno de los posibles tipos de trama, se asume que ha habido un error de recepción. En este caso se reposicionan los punteros del búffer de entrada, pasando por alto el error y esperando a la siguiente trama. Esto se hace así porque la gran mayoría de las tramas que se van a recibir serán tramas de petición, por lo que normalmente no va a suponer ningún problema el no atenderlas en caso de recepción incorrecta. En ese caso, el envío de datos simplemente se aplazaría hasta la siguiente recepción de una trama de petición. Por otra parte, en el poco probable caso de que la trama incorrectamente recibida fuera otra distinta de una trama de petición, el efecto de despreciarla sería que el comando que se pretendía ejecutar (cambio de ganancia, desconexión, resincronización del reloj *software* o reconexión) no se ejecutaría, teniendo en todo caso el usuario conocimiento de ello a través del programa de control del sistema.

Como puede verse en el diagrama de flujo, antes de entrar en la espera de recepción de trama, hay una comprobación del estado (activo o no) de la estación. En un estado no activo, la estación permanece atenta a las tramas que recibe, pero sólo para detectar una posible trama de reconexión que la devolvería al estado normal. En caso de no ser este tipo de trama, el programa no realiza ninguna acción y, por tanto, para el PC nodal es como si no existiera. Este estado inactivo se ha introducido para permitir la posibilidad de desconectar una estación por mal funcionamiento o por cualquier otra causa, sin necesidad de tener que desplazarse hasta la misma y apagarla físicamente. Para realizar la desconexión remota de una estación hay que introducir un comando de desconexión desde el programa de control del sistema. Del mismo modo, para devolver la estación a su estado activo debe introducirse un comando de reconexión. Como se verá en el programa del PC nodal, una estación también puede pasar al estado inactivo automáticamente, cuando el PC nodal no reciba contestación a las peticiones de trama después de un número de intentos consecutivos prefijado.

Por último, cada vez que se ejecuta el bucle principal se consulta el estado de la línea RI del segundo puerto serie que, como ya se vio en la sección 3.5.1.3, está conectada al nivel alto de RS-232 a través de un pulsador. De este modo, cuando éste ha sido presionado el programa sale del bucle principal y realiza las correspondientes operaciones (reposición de los vectores de interrupción, finalización de la comunicación serie) antes de salir al sistema operativo. Si no se detecta que el pulsador se ha presionado, se vuelve al principio del bucle principal del programa, refrescando antes el *watchdog* para evitar que el sistema se reinicie. El pulsador se utiliza básicamente durante la etapa de desarrollo y depuración de los

programas en laboratorio, ya que durante la operación normal del sistema éste se controlará remotamente a través de Internet y no será habitual acceder físicamente a los laboratorios subterráneos donde está instalado.

El refresco del *watchdog* se realiza escribiendo en el registro de *watchdog* un código correspondiente al periodo de refresco seleccionado, que puede tomar valores entre 6 y 42 segundos. Si transcurre un tiempo igual al periodo seleccionado sin refrescar el timer del *watchdog*, se asume que el programa ha quedado fuera de control, y el timer generará una señal de reset del sistema. En el modelo de placas de PC industrial utilizado existe también la opción de programar el *watchdog* para que genere una señal de petición de interrupción IRQ15 en lugar de una de reset, y realizar así las operaciones necesarias en la rutina de servicio de la interrupción.

3.6.2. Programa de los PCs nodales (PC_NODAL.C)

El programa PC_NODAL.C es responsable del control de la comunicación serie con las cuatro estaciones conectadas a su línea, así como de la grabación de los datos vía red. Para ello realiza las siguientes tareas:

- Interrogación de cada una de las estaciones mediante el envío de tramas de petición de trama.

- Grabación de las tramas de datos procedentes de cada estación en su correspondiente fichero de red.

- Lectura periódica del fichero de comandos, que es modificado por el programa de control del sistema que se ejecuta en el servidor. En caso de que haya alguna orden en el mismo, el programa PC_NODAL debe gestionar el envío de la trama de control correspondiente a la estación o estaciones indicadas en el fichero.

- Actuación en caso de producirse algún error de transmisión o recibir alguna trama de error de una de las estaciones. Esta actuación consiste en la escritura del error producido en el fichero de estado del sistema y, en algunos casos, en el envío de una trama de control a la estación correspondiente. Las tramas intercambiadas y el efecto producido en las estaciones se muestran en la tabla 3.4.

Con el objeto de evitar conflictos de acceso, la escritura y lectura de ficheros debe hacerse con funciones de gestión de ficheros en red. Estas funciones bloquean un fichero cuando se está accediendo a él, de modo que si otro programa intenta leer o escribir en el mismo fichero a la vez se encontrará con que tiene el acceso restringido (Tischer y Jennrich, 1996). Las funciones de red permiten distintas posibilidades y niveles de acceso y protección. Se puede hacer, por ejemplo, que el programa que escribe bloquee el fichero para escritura, pero permita el acceso a otro programa para lectura; o que no se bloquee el fichero entero, sino sólo determinados campos del mismo, permitiéndose el acceso total al resto del fichero. En nuestro caso, sin embargo, se ha utilizado el modo de bloqueo total: cuando un programa accede a un fichero, ya sea para lectura o escritura, impide a cualquier otro programa todo tipo de acceso. Este permanecerá a la espera hasta que el primero libere el fichero, y entonces pasará a realizar su acceso para lectura o escritura. El programa de detección de eventos deberá utilizar las mismas funciones (u otras que gestionen los bloqueos de modo análogo) para acceder a los ficheros circulares de datos. Esto, sin embargo, no es necesario en el caso del programa de control del sistema, ya que los ficheros de datos a los que tendrá acceso serán

ficheros de evento, a los que ningún otro programa necesitará acceder al mismo tiempo.

	Error producido	*Trama recibida por el PC nodal*	*¿Escritura en fichero de estado?*	*Trama enviada a la estación*	*Efecto en la estación*
Errores en PC nodal	Trama incompleta	------	No	T. de petición de reenvío	Reenvío de la última trama
	Formato trama incorrecto	------	No	T. de petición de reenvío	Reenvío de la última trama
	TimeOut (Nº < límite)	------	No	------	------
	TimeOut (Nº > límite)	------	Sí	Trama de desconexión	Paso a estado inactivo
Errores en estaciones	Error inicializ. CADs	Trama error inicializ.CADs	Sí	------	------
	Error saturac. búffer	Trama error saturac. búffer	Sí	------	------
	Error sincr. reloj *software*	T. error sincr. reloj *software*	Sí	------	------

Tabla 3.4. *Tramas de error intercambiadas entre PC nodal y estaciones y efecto en estas últimas.*

La figura 3.23 muestra un diagrama de flujo del programa PC_NODAL.C. El primer paso que debe realizar son las inicializaciones correspondientes. En este caso, además de la inicialización del puerto serie y la lectura del número de PC nodal a través de la tarjeta de códigos, que ya se comentaron al hablar del programa ESTACION.C, el programa debe abrir los ficheros de datos correspondientes a las cuatro estaciones de su línea serie.

Con objeto de mantener un orden en el programa, la gestión de un error no se realiza inmediatamente después de haberlo detectado, sino en puntos fijos del programa. Para ello se utilizan indicadores o flags de error, que se activan al detectar el error y se consultan y desactivan (si es necesario) en las funciones del programa encargadas de la gestión de errores. Así, como puede verse en el diagrama de flujo, lo primero que se hace dentro del bucle principal es comprobar si hay algún flag de desconexión o de comandos activado, para enviar la trama correspondiente a la estación implicada. Si ninguno de esos flags está activado se comprueba, de la misma manera, si es necesario pedir un reenvío de trama a la última estación interrogada: se consulta el correspondiente flag de petición de reenvío, que se habría activado en la anterior pasada por el bucle principal si se hubiera detectado un error de recepción. Si no es necesario pedir reenvío de trama, se continúa la operación normal, enviando una trama de petición de trama a la siguiente estación de la línea.

Una vez enviada la trama de petición de trama, el programa queda a la espera de respuesta. Si después de un tiempo prefijado no la ha recibido, activa el flag de *time out* y continúa el programa, incrementando antes un contador de *time outs* consecutivos. Si este contador sobrepasa un límite prefijado, se activa otro flag que provocará el envío de una trama de desconexión a la estación correspondiente, ya que después de ese tiempo sin recibir respuesta se asume funcionamiento incorrecto de dicha estación. A partir de este momento el PC nodal dejará de enviar tramas a la estación desconectada, hasta que se solicite un intento de reconexión desde el programa de control del servidor.

Si se ha recibido respuesta antes de producirse el *time out*, el programa pasa a analizar la longitud y estructura de la trama, activando los flags de error correspondientes en caso de detectar algún fallo.

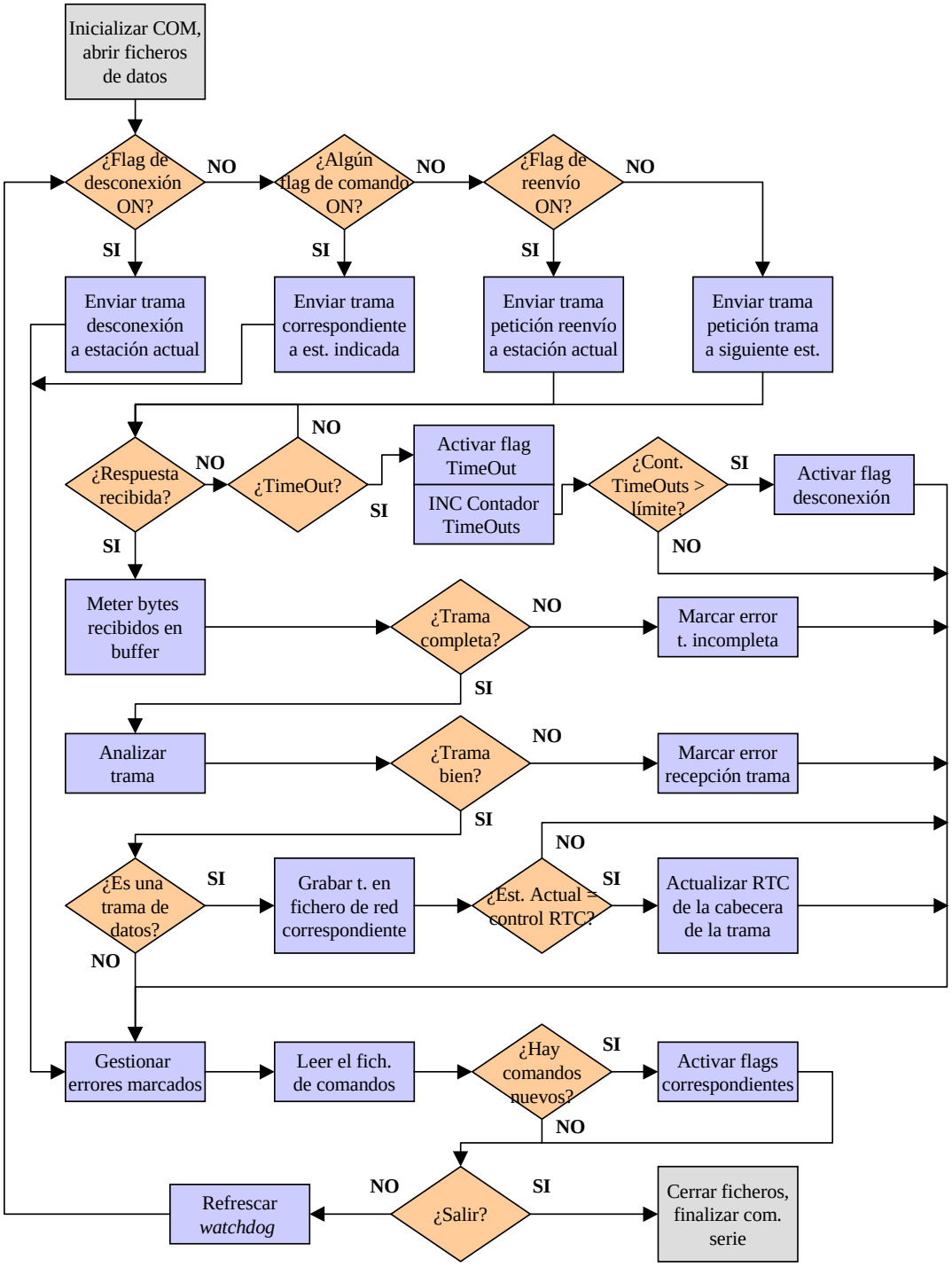


Figura 3.23. *Diagrama de flujo del programa de los PCs nodales (PC_NODAL.C). La mayor parte del código se ocupa del control del intercambio de tramas de datos, de control y de error con las estaciones. Además hay que gestionar otras tareas como la escritura de datos en los ficheros de red y la lectura del fichero de comandos.*

En el caso de que la trama recibida sea de datos y no se haya producido ningún error de recepción, la trama se graba en el fichero correspondiente. Aunque en el diagrama de flujo esto aparece, por simplicidad, como una sola tarea, en el programa no se realiza así. La grabación no se realiza trama a trama conforme se van recibiendo, sino que en primera instancia las tramas se van guardando en un búffer de memoria de un minuto de duración máxima. Cada PC nodal debe gestionar cuatro de estos búffers, uno por cada estación de su línea serie. La descarga en cada uno de los ficheros se produce en momentos distintos, según se muestra en la tabla 3.5. Como puede verse, se ha procurado separar lo máximo posible en el tiempo las grabaciones sucesivas de un mismo PC nodal para minimizar los conflictos de tiempo en la grabación en red. La temporización de este proceso se realiza a partir del reloj de tiempo real (RTC) de los PCs nodales. Como se ve en el diagrama de flujo, la actualización de este reloj se realiza a partir de las cabeceras de tiempo de las tramas de datos de una de las cuatro estaciones de la línea serie. Inicialmente esta estación es la número 0 de cada línea serie, pero si dicha estación pasa a estado inactivo, el control del RTC del PC nodal pasará a la siguiente de la línea (estación número 1), y así sucesivamente. Aunque la precisión de este reloj no es muy alta –está influido por retrasos en la transmisión serie y posibles errores en la transmisión-, y no podría utilizarse en procesos de sincronización, sí es más que suficiente para la temporización del proceso de grabación en red explicado. La utilización de este reloj *software* no implica, por supuesto, una pérdida de precisión en los datos ni requiere ninguna manipulación posterior de los mismos en la etapa de análisis, ya que las cabeceras de tiempo de las tramas de datos proporcionan en todo momento la información precisa del tiempo de adquisición de cada muestra.

Después de este proceso de recepción e identificación de tramas se pasa a la gestión de los posibles errores marcados. Para ello se consultan todos los flags de error y, en función de su estado, se procede según se vio en la tabla 3.4. Como se ha explicado antes, en caso de ser necesaria una petición de reenvío o una desconexión remota no se realiza aquí, sino que se activa otro flag para mandar la trama correspondiente al principio de la siguiente pasada por el bucle principal.

PC nodal	*Estación*	*Segundos grabación*		*PC nodal*	*Estación*	*Segundos grabación*
0	0	0 y 30		3	0	3 y 33
0	1	6 y 36		3	1	9 y 39
0	2	12 y 42		3	2	15 y 45
0	3	18 y 48		3	3	21 y 51
1	0	1 y 31		4	0	4 y 34
1	1	7 y 37		4	1	10 y 40
1	2	13 y 43		4	2	16 y 46
1	3	19 y 49		4	3	22 y 52
2	0	2 y 32		5	0	5 y 35
2	1	8 y 38		5	1	11 y 41
2	2	14 y 44		5	2	17 y 47
2	3	20 y 50		5	3	23 y 53

Tabla 3.5. *Temporización de la escritura de datos en los ficheros de red. Cada PC nodal graba dos veces por minuto en cada uno de los ficheros de datos, en momentos distintos para evitar conflictos de tiempo.*

Tras la gestión de los posibles errores se realiza una lectura del fichero de comandos, para ver si se ha introducido algún comando nuevo desde la última lectura. Como se verá al

hablar del programa de control de los servidores, cada uno de los PCs nodales tiene asignado un fichero de comandos distinto que se borra cada vez que es leído. Por tanto, lo único que debe hacer el PC nodal en este punto es comprobar si su fichero de comandos existe. En caso afirmativo lo lee, prepara la trama o tramas correspondientes y activa los flags necesarios para enviarlas en la siguiente pasada por el bucle principal.

Por último, e igual que en el programa ESTACION.C, se comprueba el pulsador de salida y, en caso de que haya sido presionado, se realizan las tareas previas a la salida del programa (finalización de la comunicación serie, cierre de los ficheros de datos) y se sale al sistema operativo. Si el pulsador no se ha presionado se vuelve al inicio del bucle principal del programa refrescando previamente el *watchdog.*

3.6.3. Programa de control para los servidores (CONTROL.C)

Dado que el sistema operativo de los servidores será alguno de la familia *Windows*, el programa de control se ha realizado bajo este entorno, usando el compilador *Borland C++ Builder* versión 3. La filosofía del programa en este caso es distinta que la de los programas bajo MS-DOS de las estaciones y PCs nodales, ya que se trata de un programa gestionado por eventos y sin necesidad de utilizar funciones de tiempo real. La comunicación con los programas de los PCs nodales se realiza a través de ficheros de cuatro tipos:

- Fichero de estado del sistema (GRANSASS.LOG): Es un fichero único, en el que cada PC nodal escribe los eventos significativos en el funcionamiento tanto de las estaciones conectadas a su línea serie como del suyo propio. Estos eventos son, entre otros, arranques de las estaciones y PCs nodales, sincronizaciones del reloj *software*, cambios de ganancia solicitados por los PCs nodales y errores en el funcionamiento de los programas. Este fichero puede ser consultado en cualquier momento desde el programa CONTROL.

- Ficheros de comandos (COMMANDS.PCx, con x = número de PC nodal): son seis ficheros distintos, uno para cada PC nodal. Los comandos se introducen a través del programa CONTROL, y los ficheros son consultados cada cierto tiempo por los PCs nodales. Cada vez que uno de los ficheros es leído se borra, de modo que cada uno de los seis COMMANDS.PCx sólo existe cuando hay algún comando pendiente de ser ejecutado por el PC nodal correspondiente.

- Ficheros de datos (NODxESTy.DAT, con *x* = número de PC nodal e *y* = número de estación): son ficheros binarios de formato circular, con una longitud correspondiente a una hora de registro. Como se vio en la descripción del programa de los PCs nodales, éstos escriben tramas de un segundo de duración, con la información relativa al tiempo y ganancia de adquisición como cabecera de cada trama. El formato de los ficheros puede verse en la figura 3.24.

- Fichero de estado de las estaciones (STATIONS.BIN): es un fichero binario, único para todo el sistema, en el que se describe el estado actual de cada una de las estaciones. Los tres posibles estados son:

  - Desactivada: puede ser tanto el estado inactivo descrito en el programa de las estaciones como físicamente apagada.
  - Registrando: teniendo en cuenta la alta densidad de estaciones de la antena y su proximidad espacial, no suele ser necesario que todas las estaciones operativas

tomen parte en el proceso de detección de eventos. Para evitar la necesidad de procesar un elevado volumen de datos, lo más lógico sería que el programa de detección sólo analizara en tiempo real los procedentes de un cierto número de estaciones configurables por programa, pero en caso de detectar un evento grabara los datos de todas las estaciones operativas. El estado 'registrando' hace referencia a las estaciones operativas y funcionando normalmente, pero que no toman parte en la detección de eventos.

- Detectando: la estación está registrando y tomando parte en la detección de eventos.

La principal finalidad del fichero de estado de las estaciones STATIONS.BIN es la de comunicarse con el programa de detección de eventos. Aunque ese programa no se describe aquí, hay que hacer notar que la primera tarea que debe realizar es la de consultar el fichero STATIONS.BIN para ver qué estaciones debe tener en cuenta en el algoritmo de detección de eventos. El fichero STATIONS.BIN puede ser modificado tanto por el propio servidor (cuando el usuario cambie las estaciones seleccionadas para participar en la detección de eventos) como por los PCs nodales (cuando alguna de las estaciones pase a estado inactivo o vuelva a conectarse a la línea serie después de haber estado inactiva).

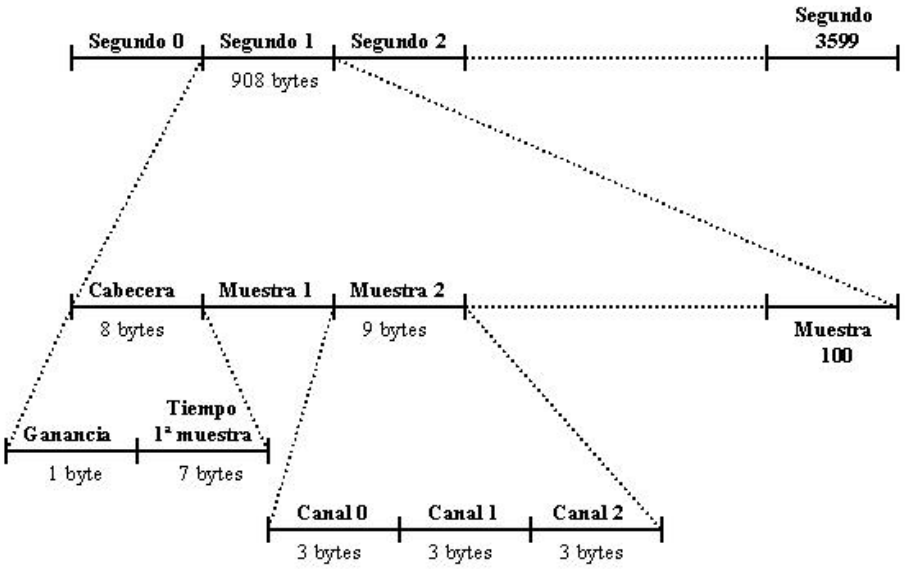


Figura 3.24. *Formato de los ficheros circulares de datos. Cada fichero comprende una hora de registro, con 908 bytes de información por segundo. La longitud de los ficheros es, por tanto, de 3268800 bytes. Los 7 bytes de la cabecera de tiempo que preceden a los datos de cada segundo corresponden al año, día juliano (2 bytes), hora, minuto, segundo y fracción de segundo de adquisición de la primera muestra.*

En la figura 3.25 se muestra la interfaz gráfica principal del programa CONTROL. Las funciones que se pueden ejecutar desde esta pantalla son:

- Botón '*View plan of stations*': muestra un plano de los LNGS, con indicación de la posición de las estaciones, PCs nodales y sala de control (figura 3.26).
- Botón '*Check state of stations*': lee el contenido del fichero de estado de las estaciones (STATIONS.BIN) y presenta la información en pantalla (figura 3.27).

- Botón '*Check system status*': presenta el contenido del fichero de estado del sistema (GRANSASS.LOG) en pantalla (figura 3.28).
- Botón '*Input commands*': abre un nuevo formulario con los posibles comandos que se pueden enviar a las estaciones, así como un menú de selección de estaciones (figura 3.29). Cuando se presiona algún comando se abre otro formulario que pide confirmación y, en su caso, los parámetros necesarios para el comando seleccionado. En la figura 3.30 se muestra como ejemplo el formulario que se abre en el caso de un comando de cambio de ganancia.
- Botón '*Demultiplex data files*': el programa de detección de eventos graba un fichero por cada evento registrado. El formato de estos ficheros es el mismo que el de los ficheros circulares comentado anteriormente (figura 3.24), salvo por el hecho de que su longitud no es fija, puesto que depende de la duración del evento. Por tanto, para poder trabajar con los datos es necesario separar antes la información de cada uno de los tres canales y la de los parámetros de adquisición. Con esta opción del programa CONTROL se realiza esta separación o demultiplexado de la información. Se toma como entrada uno de los ficheros de evento existentes en el directorio de datos seleccionado y se obtienen como salida tres ficheros binarios de datos (uno por canal) y un fichero ASCII de cabecera con el tiempo y parámetros de adquisición de los datos. La figura 3.30 muestra el formulario abierto al seleccionar la opción de demultiplexado de los ficheros de datos. En el cuadro de texto de la pantalla se muestra el contenido del fichero ASCII de cabecera. La nomenclatura utilizada para nombrar los ficheros de evento y los de salida demultiplexados puede verse en la tabla 3.6.
- Botón '*Help*': Muestra en pantalla el fichero ASCII de ayuda CONTROL.HLP.
- Botón '*Quit*': Sale del programa. La salida del programa se puede realizar también mediante el botón de cerrar ventana de la pantalla principal.

El anexo I.A.3 incluido en el CD adjunto muestra el fichero fuente principal de la aplicación (I.A.3.a), así como los ficheros fuente de los subprogramas asociados a los distintos formularios que se van abriendo a medida que se seleccionan distintas opciones en la interfaz gráfica. Así, el listado de I.A.3.b corresponde al código fuente del formulario principal, mientras que el resto (I.A.3.c a I.A.3.h) son los listados correspondientes a los formularios de selección de los distintos comandos.

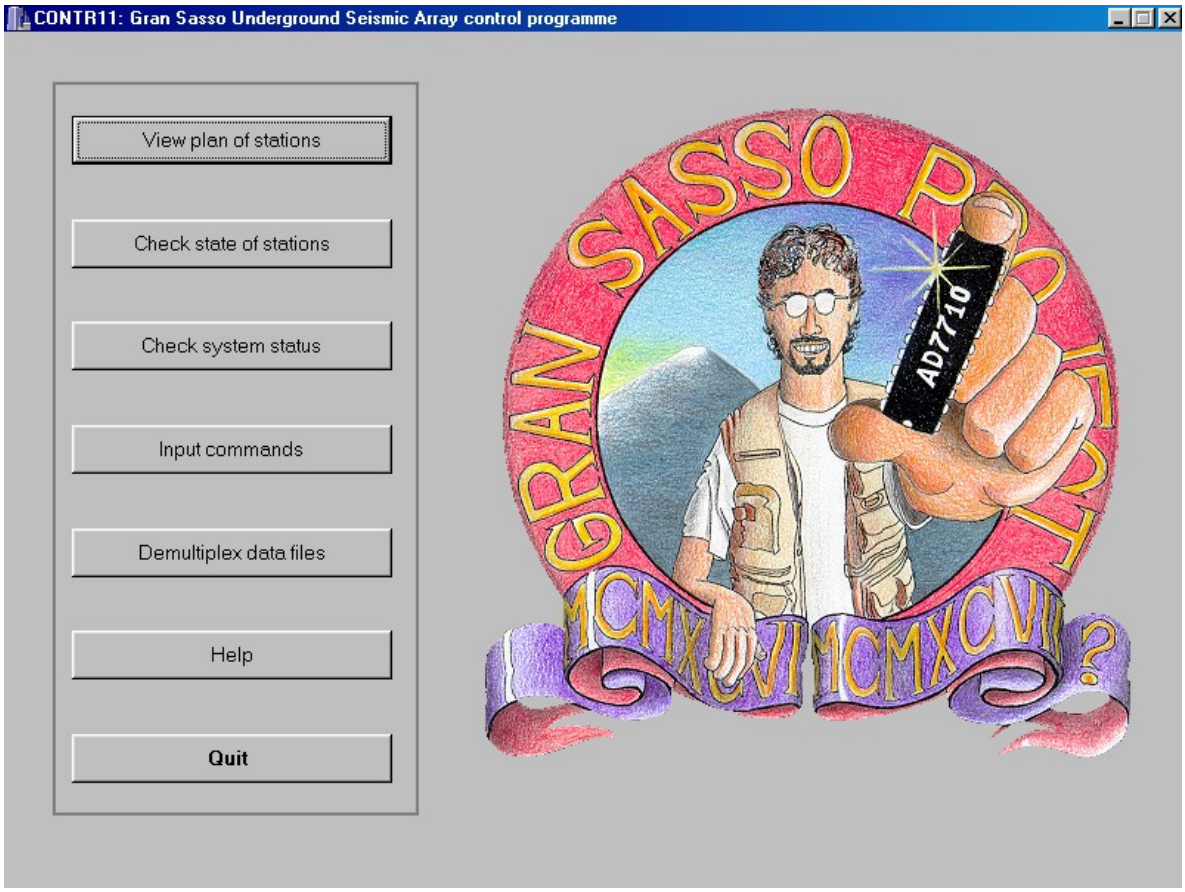


Figura 3.25. *Pantalla principal del programa CONTROL..*

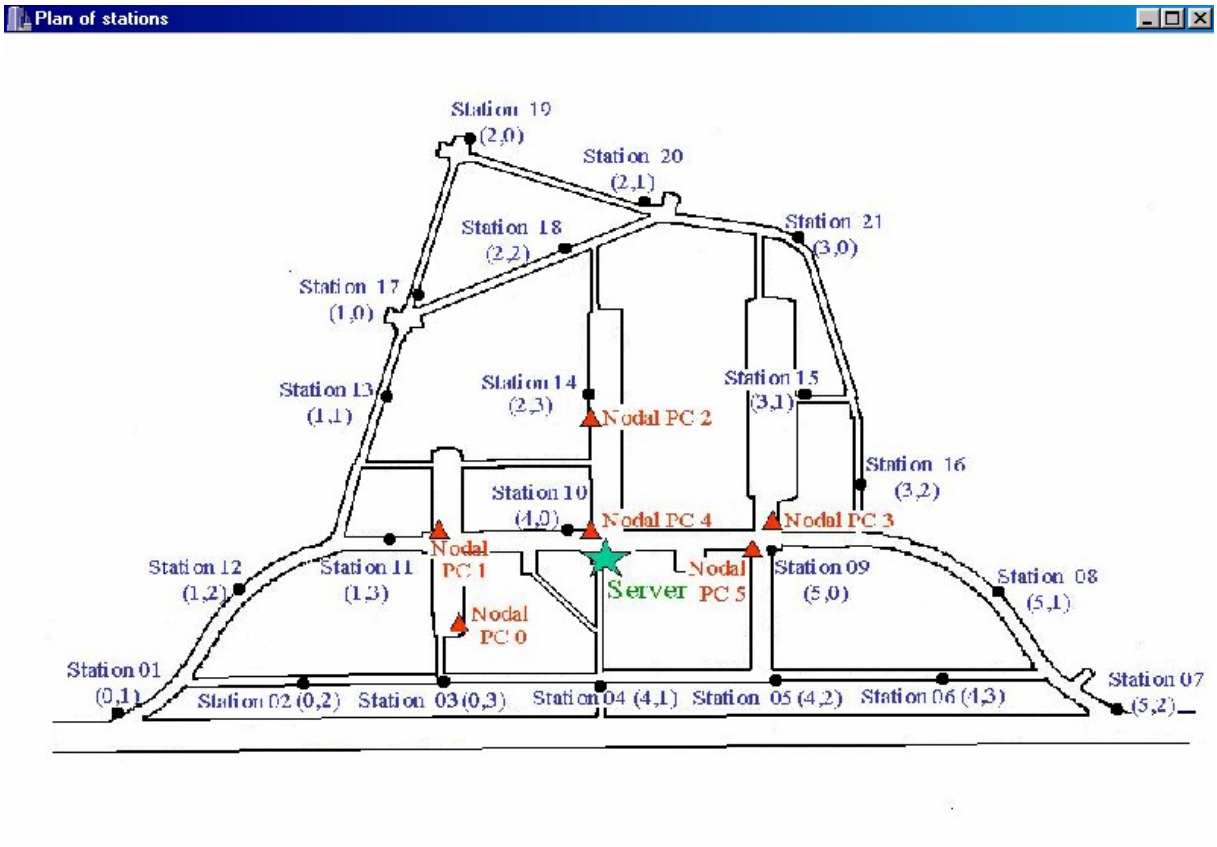


Figura 3.26. *Pantalla de visualización del plano de las estaciones.*

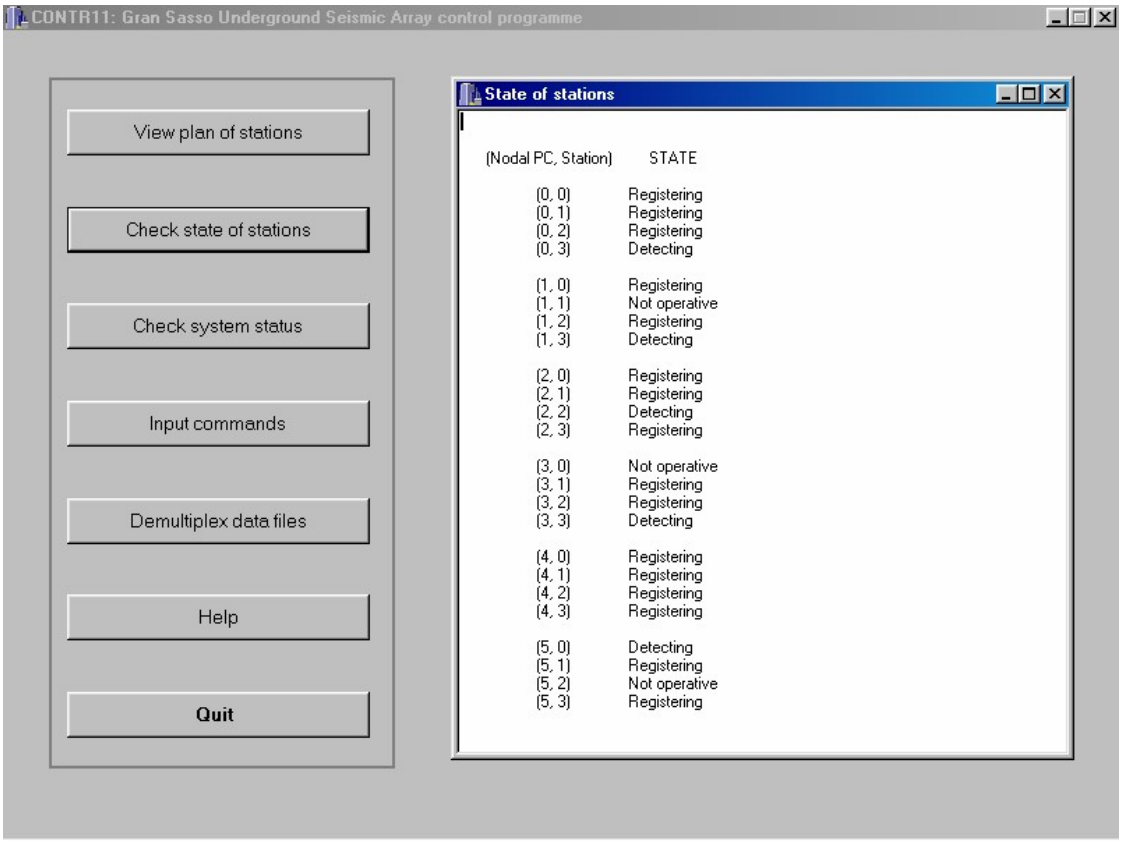


Figura 3.27. *Pantalla de consulta del estado de las estaciones.*

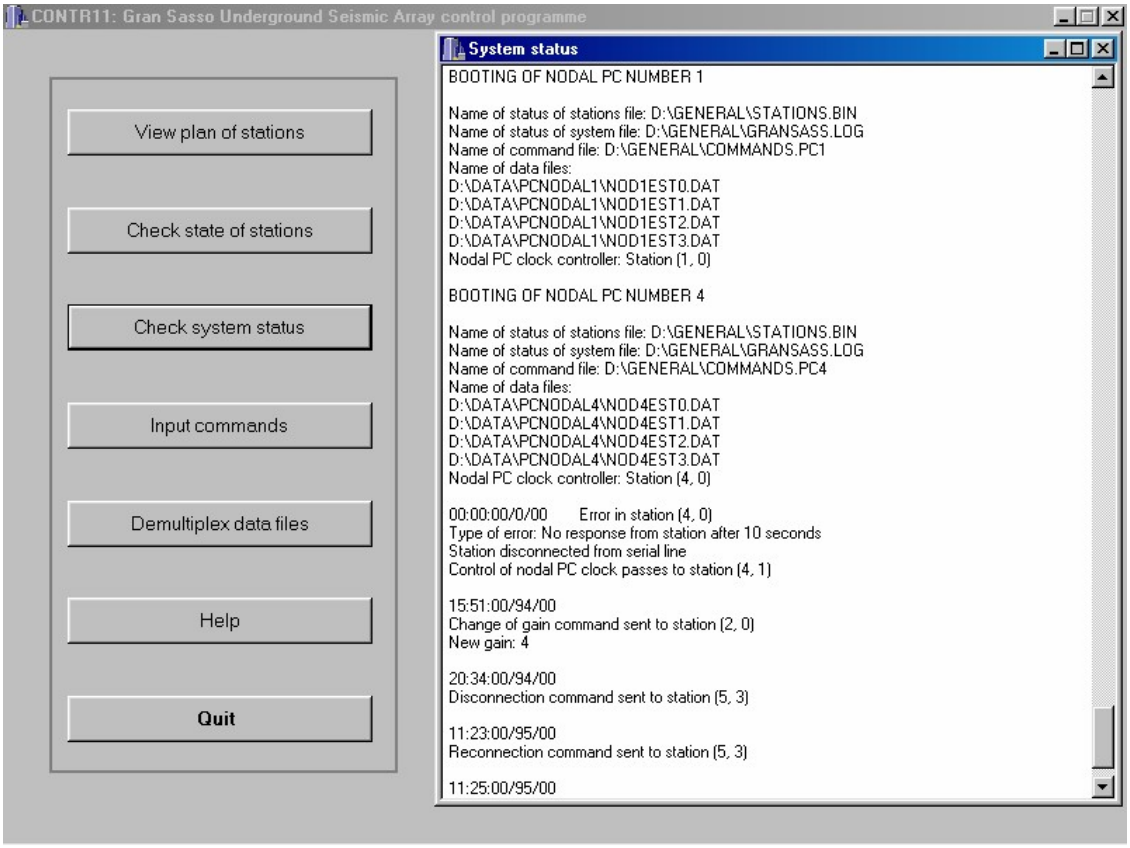


Figura 3.28. *Pantalla de consulta del estado del sistema.*

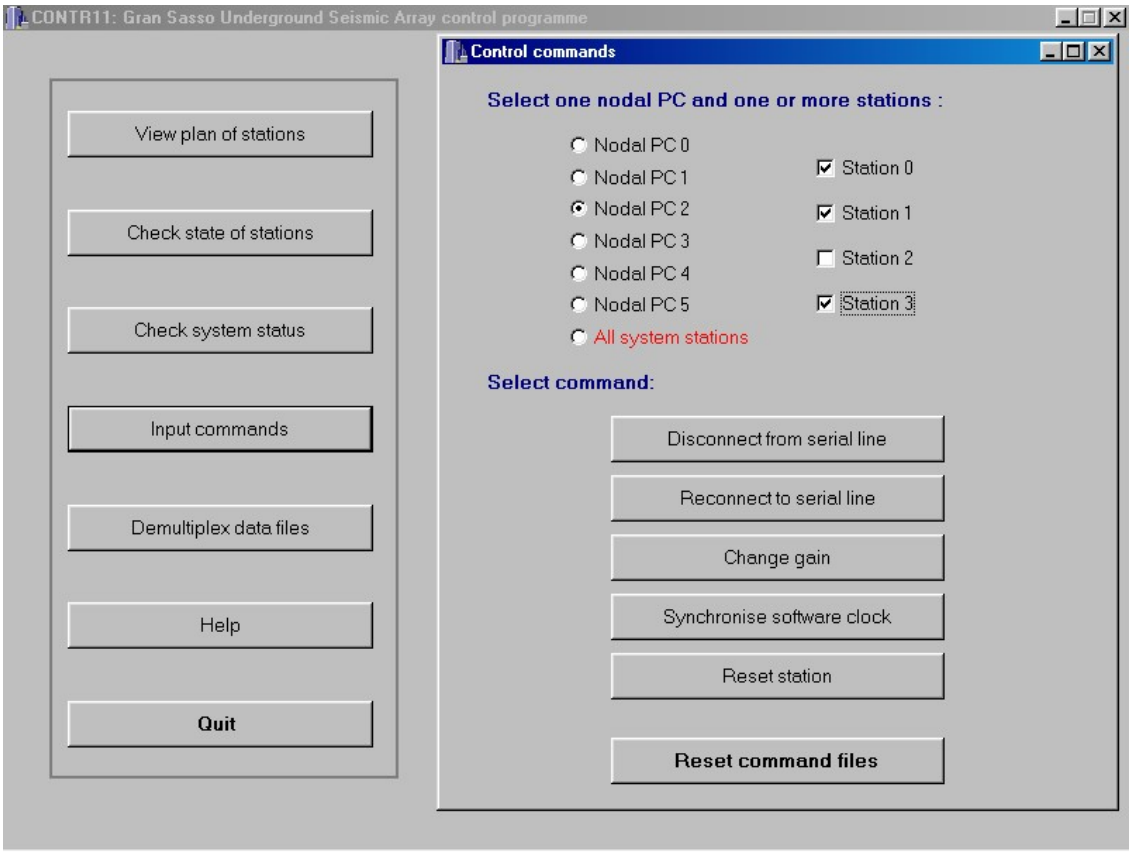


Figura 3.29. *Pantalla de introducción de comandos.*

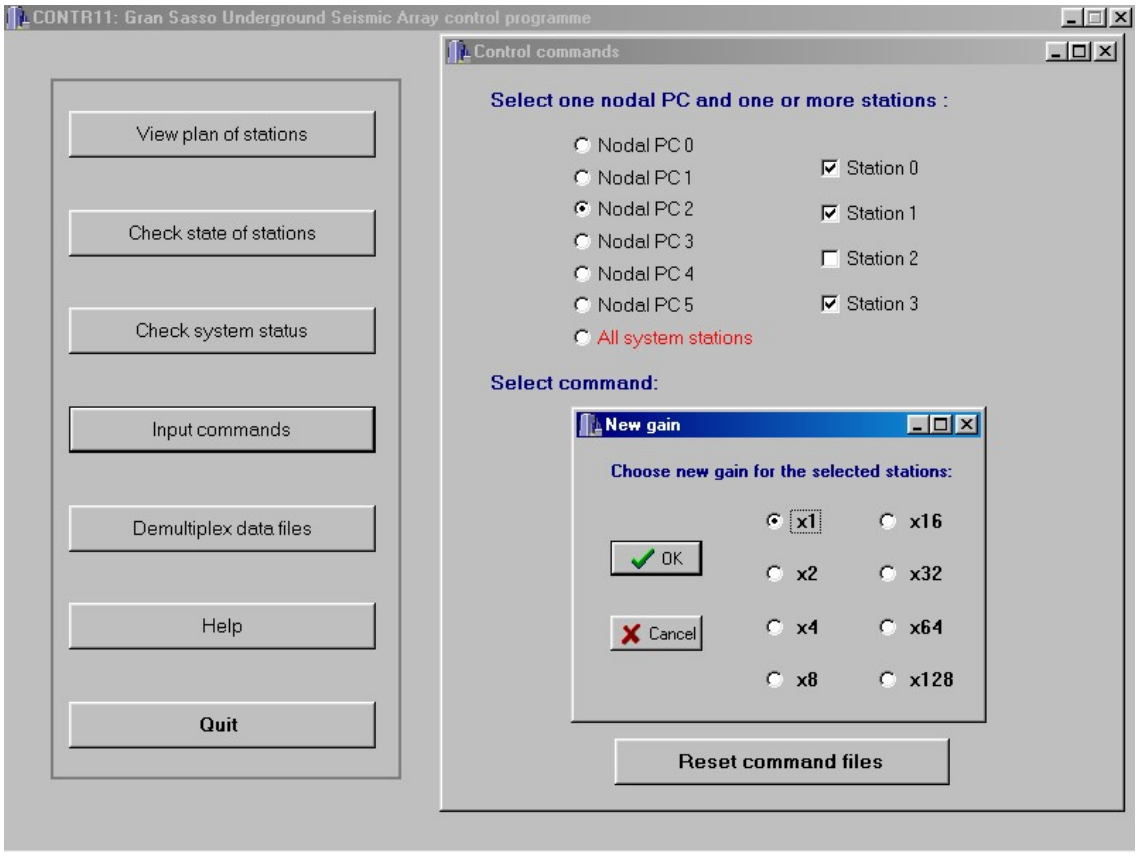


Figura 3.30. *Pantalla de introducción de comando de cambio de ganancia.*

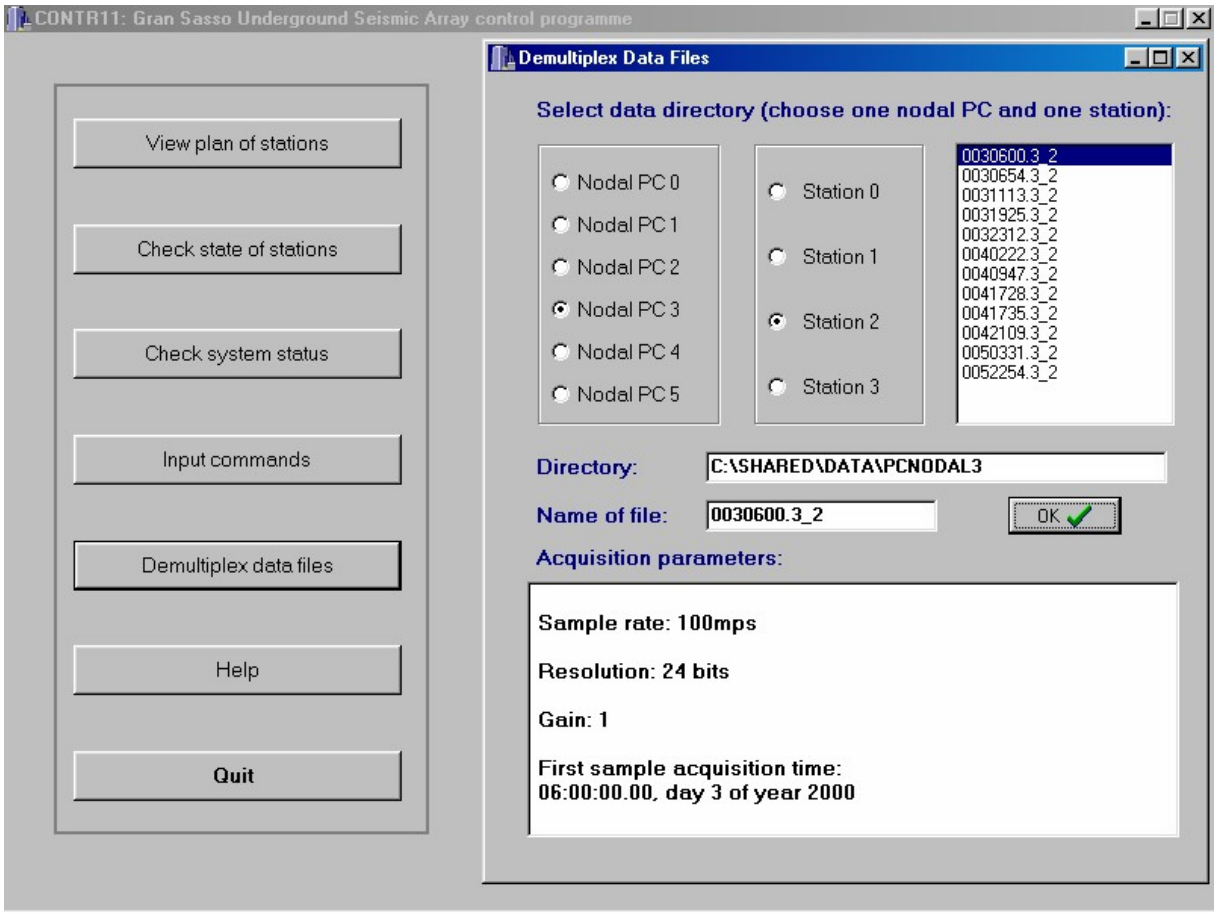


Figura 3.31. *Pantalla de demultiplexado de los ficheros de evento.*

Nomenclatura ficheros de datos: DDDHHMM.N_E *siendo:*
DDD*: Día juliano (tres dígitos)*
HH*: Hora de inicio (dos dígitos)*
MM*: Minuto de inicio(dos dígitos)*
N: *Número de PC nodal (0, 1,..., 5)*
E: *Número de estación (0, 1, 2 o 3)*

Nomenclatura ficheros demultiplexados: *mismo nombre que los ficheros de datos, con extensiones (N y E tienen los mismos significados que antes):*
.NEz*: componente Z de la señal (primer canal de adquisición)*
.NEn*: componente N-S de la señal (segundo canal de adquisición)*
.NEe*: componente E-W de la señal (tercer canal de adquisición)*
.NEh*: fichero ASCII de cabecera*

Tabla 3.6. *Nomenclatura utilizada para nombrar los ficheros de datos y los demultiplexados. Al demultiplexar se obtienen tres ficheros binarios de datos y uno ASCII de cabecera, con los parámetros de adquisición y tiempo de registro de la primera muestra del fichero.*

3.6.4. Programa del oscilador patrón para la generación de las señales de sincronización (OSCILAD.ASM)

El programa del oscilador patrón es el encargado de controlar la generación de las dos señales de sincronización necesarias para el sistema. Como ya se ha comentado anteriormente, estas señales son dos:

- Señal *Sinc1*, de frecuencia 9.8304MHz/2^n (n = 3, 4, 5 o 6), para la síntesis en las estaciones de la señal de reloj de 9.8304 MHz de los CADs.
- Señal *Sinc2* de tiempo real codificada.

Según se explicó al hablar de la tarjeta del PIC16C74, la operación del programa del oscilador patrón se basa en el primer módulo Captura/Comparación/PWM (CCP1) del PIC16C74, que funciona en conjunción con el timer1 del microcontrolador. En modo de captura, el par de registros CCPR1H:CCPR1L captura el valor de 16 bits del registro TMR1 cuando ocurre un evento en el pin RC2/CCP1. Se pueden utilizar como eventos todos los flancos ascendentes de la señal conectada a RC2/CCP1, todos los flancos descendentes, uno de cada cuatro flancos ascendentes o uno de cada dieciséis flancos ascendentes. La selección de uno u otro tipo de evento se realiza programando los bits 3 a 0 del registro CCP1CON (Microchip Technology Inc., 1997). Cuando se produce un evento se realiza una captura, y además se activa el flag de solicitud de interrupción (bit 2 del registro PIR1), lo que permite realizar las operaciones necesarias con el valor capturado en la rutina de gestión de la interrupción.

La señal que se usará en nuestro caso como generadora de eventos es la señal de un pulso por segundo del patrón atómico. Sin embargo, dado que en el momento de plantear el programa del oscilador patrón no se conocían con detalle las características de dicha señal se dejó la posibilidad de utilizar como evento tanto el flanco ascendente como el descendente. La selección de uno u otro flanco se realiza a través de los pulsadores de configuración del oscilador patrón, como se verá al hablar de la operación del sistema.

Como puede verse en el diagrama de flujo de la figura 3.32.b, el primer paso que debe realizar el programa es la inicialización de los distintos módulos:

- Inicialización de todos los puertos de entrada/salida según la función que vayan a realizar.
- Módulo Timer1 en modo contador síncrono, para poder operar con la señal de salida del VCXO como entrada a través del pin RC0.
- Módulo CCP1 en modo captura. Inicialmente se programa con el flanco descendente de la señal de RC2 (entrada de un pulso por segundo) como evento de captura, pero como se ha dicho el flanco de disparo se puede cambiar luego mediante los pulsadores de configuración.
- Módulo CCP2 en modo PWM, para actuar sobre el VCXO a través del pin RC1. Inicialmente el módulo PWM se programa con un *duty-cycle* del 50%, que se irá variando luego para corregir las desviaciones del VCXO de su frecuencia central.
- Inicializaciones para el control del display LCD.

La mayor parte de las tareas del programa se realizan en el bucle principal, mientras que en la rutina de gestión de interrupción sólo se ejecutan las tareas críticas en el tiempo. Hay que hacer notar que en el PIC16C74 sólo hay un vector de interrupción, al que el contador de programa accede al producirse cualquiera de ellas. Por tanto, el primer paso en la programación de la rutina de servicio debe ser siempre la identificación de la fuente generadora de la interrupción.

En nuestro caso hay dos fuentes distintas de interrupción permitidas:

- La interrupción del módulo CCP1, que se dispara con los pulsos de segundo del patrón atómico.
- La interrupción del timer1, que se dispara cada vez que hay desbordamiento del timer1. El registro del timer1 es de 16 bits, de forma que usando esta interrupción podemos contar el número de vueltas que da y calcular el número total de pulsos en un intervalo de tiempo.

Para identificar si la interrupción que se ha producido es alguna de estas dos se consulta el estado de los bits CCP1IF y TMR1IF en el registro PIR1 (flags de interrupción del módulo CCP1 y del Timer1, respectivamente). Si ninguno de los bits está activado se abandona la rutina de servicio de interrupción, como se muestra en la figura 3.32.a.

a)

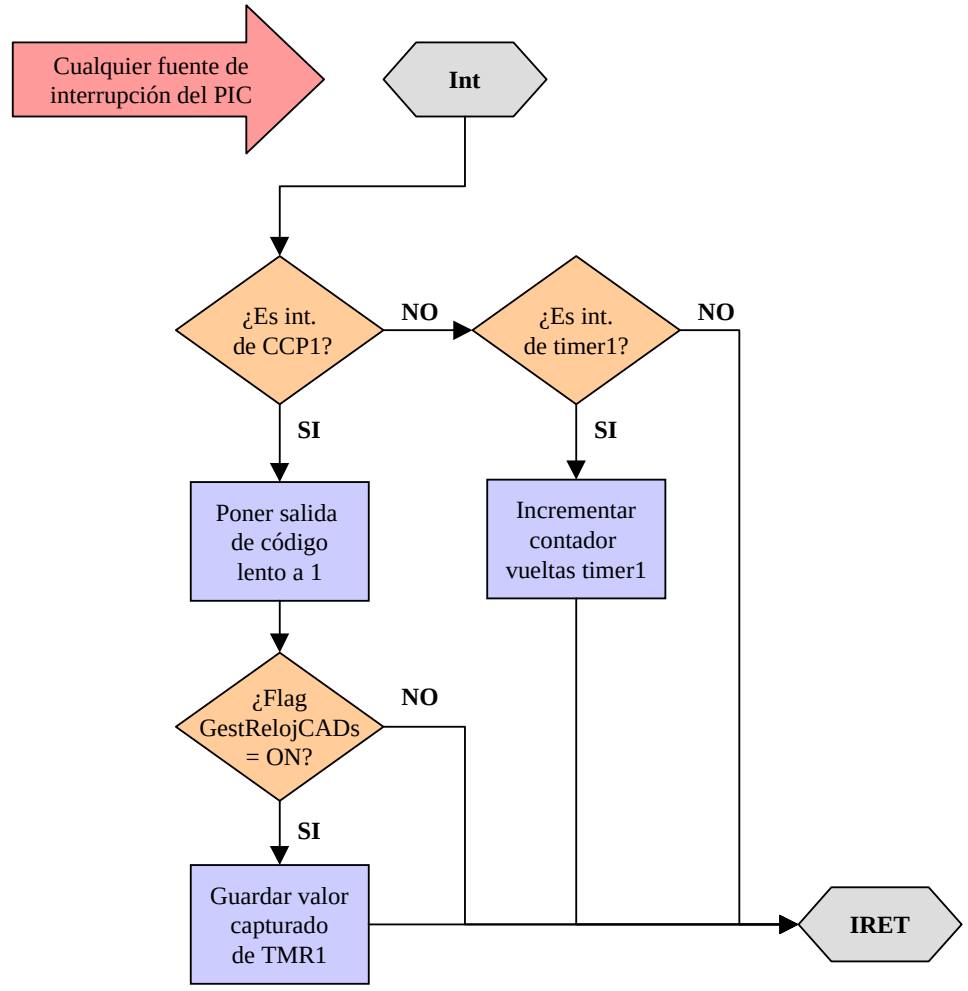
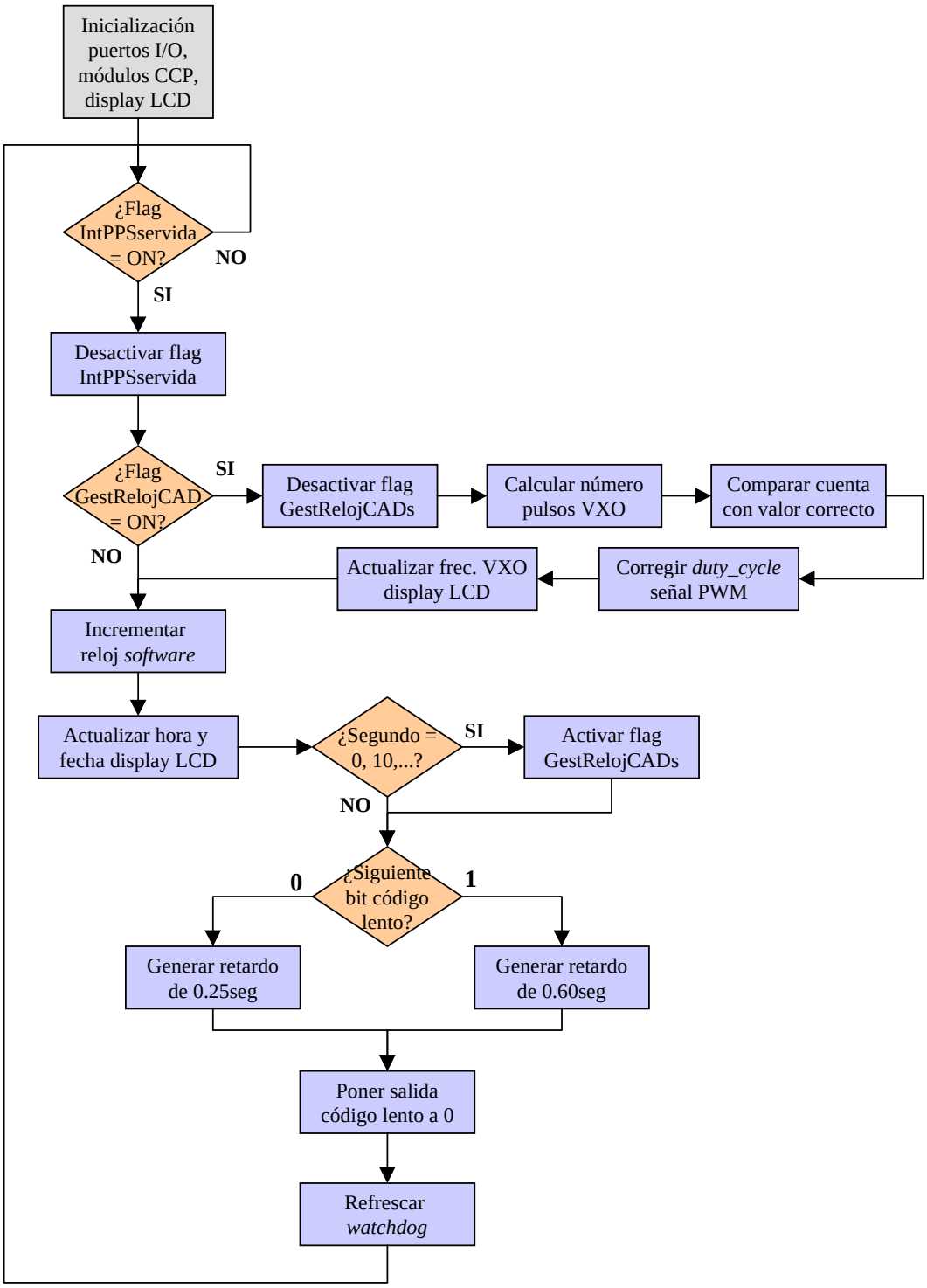


Figura 3.32. *Diagrama de flujo del programa del oscilador patrón. En a) se muestra la rutina de servicio de las interrupciones de CCP1 y timer1. En b) (página siguiente), el bucle principal del programa.*

3.32.b)



Dado que el flanco de subida del código lento tiene que estar sincronizado con el flanco del pulso de segundo del patrón atómico, la tarea más crítica en el tiempo en la interrupción del módulo CCP1 es la de poner la salida del código lento de tiempo real (señal *Sinc2*) a uno. Según puede verse en el diagrama de flujo de la figura 3.32.a, ésta es la única tarea que se realiza siempre en la interrupción de CCP1. La otra tarea de esta interrupción, la lectura del valor capturado del timer1, no se realiza siempre, sino sólo una de cada diez

interrupciones. De este modo se busca que el ajuste de la frecuencia de la señal del VCXO se realice de una forma suave. Para saber cuándo se debe leer el valor capturado se consulta un flag de 'Gestionar reloj CADs', que se activa, en la función que incrementa el reloj de tiempo real en el bucle principal, cada diez segundos. Este flag indica que la siguiente interrupción debe realizar las tareas de gestión del reloj de los CADs (es decir, de la señal *Sinc1*), por lo que si el flag se activa en los segundos 0, 10, 20,..., el reloj de los CADs se gestionará en los segundos 1, 11, 21,...

Una vez ejecutadas estas funciones críticas en el tiempo se abandona la rutina de servicio, activando previamente un flag de 'Interrupción de PPS servida', para que en el siguiente paso por el bucle principal se sirvan el resto de tareas.

En cuanto a la interrupción del Timer1, la única tarea que debe realizarse en la rutina de servicio es la de incrementar el contador de vueltas que ha dado el timer.

Las tareas que se deben realizar en el bucle principal del programa después de cada interrupción de CCP1 comprenden todas las de gestión del reloj de tiempo real y su transformación a código lento para ser transmitido a las estaciones como *Sinc2*, así como el resto de tareas relativas al control de la frecuencia de la señal de reloj de los CADs o *Sinc1*.

Como puede verse en la figura 3.32.b, entre las tareas de gestión del código lento están:

- Incrementar el reloj *software* de tiempo real.
- Activar el flag de 'Gestionar reloj CADs' cada diez segundos.
- Seleccionar el siguiente bit que debe transmitirse del código lento.
- En función del valor de ese bit, 0 o 1, generar un retardo de 0.25 o 0.60 segundos, respectivamente.
- Volver a poner a 0 el pin RC3 (señal *Sinc2*), que se había puesto a 1 en la rutina de servicio de la interrupción.

En cuanto al resto de tareas relativas al control de la señal *Sinc1*, también se ejecutan sólo cada diez segundos, para lo cual se consulta de nuevo el flag de 'Gestionar reloj CADs'. Las tareas que se realizan en caso de estar el flag activado son:

- Desactivar flag 'Gestionar reloj CADs'.
- Calcular número de ciclos del VCXO en el último intervalo de diez segundos. Para ello se utiliza el valor del registro del timer1 capturado en la última interrupción de CCP1 y el valor del contador de vueltas leído en la última interrupción del timer1.
- Comparar el número de ciclos calculado con el número de ciclos correcto del VCXO cuando está oscilando en su frecuencia central (2.4576MHz).
- Actuar sobre el VCXO en función de la diferencia entre los dos valores. El ajuste se realiza modificando el *duty-cycle* de la señal PWM del módulo CCP2, para lo cual se varía el valor del registro CCPR2L. Con el objeto de conseguir un ajuste suave el registro sólo se incrementa o decrementa en una unidad cada vez, independientemente de la magnitud de la diferencia entre los dos valores.

Además de las tareas de la generación de las dos señales de sincronización propiamente dichas, el programa debe ocuparse continuamente de la gestión del display LCD, para proporcionar en todo momento información al usuario sobre el funcionamiento del oscilador patrón. Durante la operación normal, el display muestra la hora y fecha y la frecuencia actual del VCXO. Cuando se pulsan los botones de sincronización o de selección de flanco del pulso de segundo, el display muestra mensajes orientativos para realizar la función seleccionada.

Como se comentó en la descripción del PIC16C74, la memoria de programa de este microcontrolador es de 4 kB. Sin embargo, a la hora de organizar el programa se ha procurado

utilizar sólo los dos primeros kB, ya que los saltos entre los dos primeros y los dos últimos KB requieren algunas instrucciones adicionales que implican mayor complejidad y, sobre todo, menor claridad en el código. La disposición del código dentro de la memoria de programa se muestra en la tabla 3.7. El listado completo del programa del oscilador patrón, así como las librerías desarrolladas para incluir constantes, nemotécnicos de registros y otras definiciones, se muestran en el anexo I.A.4 del CD adjunto.

Dirección de memoria	*Código*
00h (Vector de reset)	Salto a etiqueta 'Inicio'
04h (Vector de interrupción)	Rutina de servicio de la interrupción
30h	Definición de las funciones y macros
600h	Tablas para mensajes del display LCD
750h	Etiqueta 'Inicio', donde se ubica el programa principal

Tabla 3.7. *Organización del programa OSCILAD.ASM en la memoria de programa del microcontrolador PIC16C74.*

3.7. Instalación y operación del sistema

3.7.1. Instalación del equipo en los LNGS

La figura 3.33 muestra la disposición de las estaciones, PCs nodales y sala de control dentro de los LNGS. La situación de los sensores es aproximadamente semicircular, aunque viene determinada por la estructura de los túneles y galerías. Como puede verse, sólo hay veintiún puntos indicados, correspondientes a los veintiún sensores de corto periodo. Los tres puntos restantes corresponderían a sensores de banda ancha, cuya disposición se elegiría de modo que formaran entre ellos un triángulo con los vértices lo más alejados posible, teniendo en cuenta que los tres sensores deben conectarse a las líneas serie que quedan con tres estaciones.

Originalmente los puntos de adquisición estaban numerados correlativamente desde 1 hasta 21. A esta numeración absoluta se le ha agregado otra, más útil a efectos de programación y control del sistema, que incluye el número de PC nodal que controla dicho punto de adquisición y el número de estación dentro de esa línea serie.

La figura muestra la localización inicialmente prevista para los PCs nodales, que coincide con la de las primeras estaciones de cada línea serie. Sin embargo, finalmente se optó por instalar todos los PCs nodales dentro de la sala de control. Este cambio de configuración obedeció sobre todo a motivos puramente logísticos: es más sencillo acceder al equipo para tareas de mantenimiento dentro de la sala de control que si se sitúa en los contenedores de los pasillos externos. Por otra parte, la sala de control ofrece unas condiciones ambientales (en particular, en lo referente a la humedad) más adecuadas para el funcionamiento de los equipos. La elección de esta configuración implicaba una longitud algo mayor de las líneas serie RS-485, pero en todos los casos se mantenían dentro de los valores recomendados para la velocidad de transmisión utilizada.

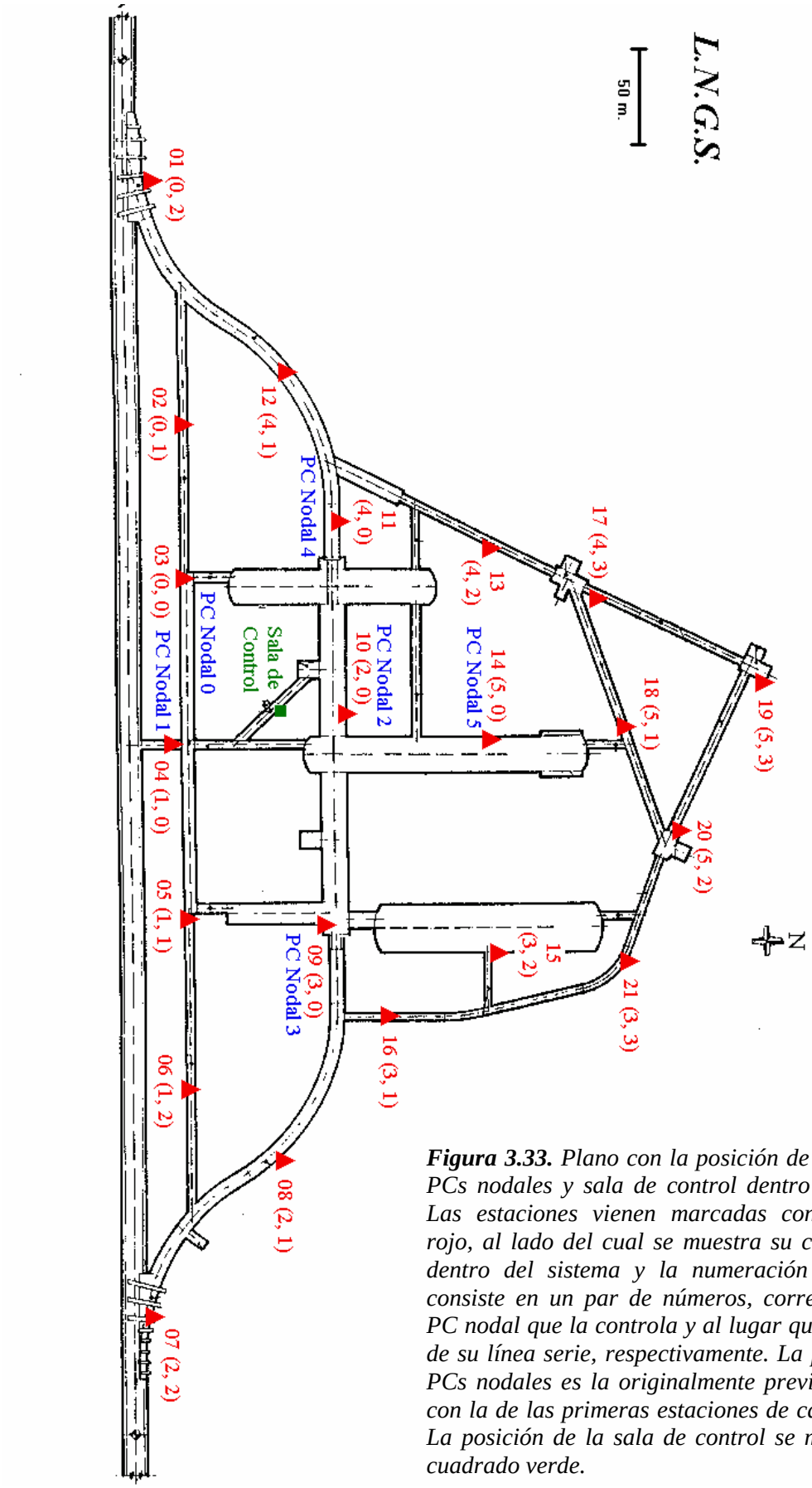


Figura 3.33. *Plano con la posición de las estaciones, PCs nodales y sala de control dentro de los* LNGS. *Las estaciones vienen marcadas con un triángulo rojo, al lado del cual se muestra su código absoluto dentro del sistema y la numeración relativa. Esta consiste en un par de números, correspondientes al PC nodal que la controla y al lugar que ocupa dentro de su línea serie, respectivamente. La posición de los PCs nodales es la originalmente prevista, y coincide con la de las primeras estaciones de cada línea serie. La posición de la sala de control se muestra con un cuadrado verde.*

Asimismo, puede verse en la figura 3.33 la localización de la sala de control, acondicionada dentro de un contenedor industrial proporcionado por los LNGS. La figura 3.34 muestra el interior del mismo, en el que pueden verse el oscilador patrón, los PCs nodales, el servidor y el concentrador para la red *ethernet*. Además de conectar los seis PCs nodales con el servidor, el concentrador se utiliza para comunicar todo el sistema con la red de alta velocidad de los LNGS y con el exterior a través de Internet. De esta forma el control del dispositivo puede realizarse de forma remota, normalmente desde las instalaciones del departamento de física de la universidad de L'Aquila y del *Parco Scientifico e Tecnologico d'Abruzzo*, cuyo personal se encarga actualmente de la gestión del dispositivo y que también fue responsable de su instalación en los laboratorios.

Las figuras 3.35 a 3.37 muestran la disposición de la electrónica y material adicional instalado en las cajas de PCs nodales y estaciones, indicando la posición de los componentes principales. Como puede verse, la alimentación de la electrónica se realiza mediante cajas comerciales con fuente de alimentación y *back-plane* de tres ranuras ISA (Acrosser modelo AR-IPC3SP). La placa de PC está conectada directamente a una de las ranuras del *back-plane*, mientras que el resto de las placas se alimenta a través de la salida de 12 V de la fuente. La alimentación de la fuente se toma del punto de red de 220 V-50 Hz. Toda la infraestructura eléctrica necesaria en los puntos de adquisición y en la sala de control, así como el tendido del cableado, corrió a cargo de los LNGS. Los cables de transmisión serie de los datos y de las señales de reloj se introducen en los contenedores a través de pasamuros para evitar la pérdida de estanqueidad de los mismos.

Figura 3.34. *Fotografía del interior de la sala de control. A la izquierda de la imagen puede verse el oscilador patrón. En primer plano, el teclado y monitor del servidor, y sobre él los PCs nodales fijados a la pared. Detrás del monitor puede verse el concentrador para la red ethernet.*

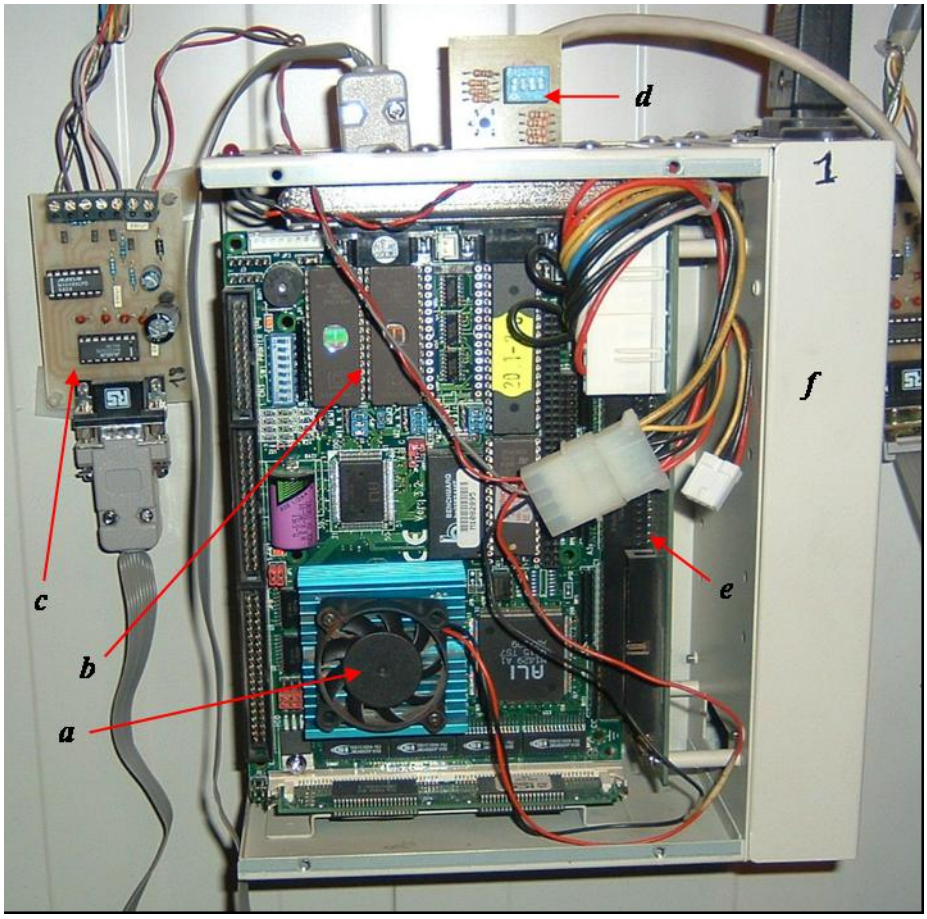


Figura 3.35. *Fotografía de uno de los PCs nodales abierto. En primer plano puede verse la tarjeta de PC industrial, en la que destacan el disipador y ventilador del microprocesador (a) y las memorias EPROM para el arranque del sistema (b). La tarjeta de conversión RS-485/RS-232 (c) está fijada a la pared y conectada a uno de los puertos serie del PC mediante un cable plano. El segundo puerto serie se utiliza para leer el código de identificación del PC nodal, que se introduce mediante los micro interruptores de la tarjeta de código (d). La placa de red ethernet no se ve, puesto que queda detrás de la tarjeta de PC. Ambas comparten las señales de control y datos a través de un* back-plane *de tres ranuras ISA (e), del que obtienen también las tensiones de alimentación necesarias. La tarjeta de código se alimenta desde el propio puerto serie, y el ventilador y la tarjeta RS-485/RS-232, a través de los conectores externos de la fuente, que está situada en la base de la caja (f).*

3.7.2. Instalación de los programas

El medio de almacenamiento de los programas es distinto en cada uno de los tres niveles del dispositivo (estaciones, PCs nodales y servidor), por lo que también lo son los respectivos procedimientos de instalación. Así, el programa de adquisición de las estaciones ESTACION.EXE debe grabarse en memorias EPROM que se insertan en los zócalos para discos de estado sólido de los PCs industriales. Además del programa ESTACION.EXE, en la memoria de cada estación deben grabarse los ficheros del sistema operativo necesarios para el funcionamiento. La versión de MS-DOS usada en las estaciones y PCs nodales es la 6.22.

Una vez grabadas las memorias e insertadas en los zócalos correspondientes de las placas de PC, éstas deben configurarse para arrancar desde disco de estado sólido. Para ello debe modificarse la posición de algunos *jumpers* en las tarjetas, que depende del tipo y

capacidad de las memorias utilizadas. Tanto para la grabación de las memorias como para la configuración de los *jumpers* se siguieron las instrucciones del manual de usuario de las tarjetas de PC.

En el caso de los PCs nodales las EPROMs deben contener, además de los ficheros de arranque propiamente dichos, los ficheros de red necesarios para comunicarse con los servidores. Se ha utilizado la red de Microsoft para trabajo en grupo de Windows 3.11, aislando los ficheros necesarios para el funcionamiento bajo MSDOS versión 6.22 (Martos, A., comunicación personal).

El programa de control PC_NODAL.EXE, sin embargo, no se graba en la memoria EPROM de los PCs nodales. Se encuentra en el disco duro del servidor, de modo que cuando cada PC nodal arranca y establece conexión de red lo carga desde ahí. De este modo, además de ahorrar espacio en la EPROM, se permite la posibilidad de realizar cualquier modificación en el programa PC_NODAL y rearrancar el sistema sin necesidad de tener que regrabar todas las EPROMs.

En cuanto al servidor, antes de arrancar el sistema se debe crear la estructura de directorios y programas que se muestra en la figura 3.38. Para ello se ha creado un CD-ROM de instalación que, a través de un programa de proceso por lotes (INSTALAR.BAT), crea las carpetas y graba los programas de adquisición, control y tratamiento de datos necesarios.

Figura 3.36. *Aspecto interno de uno de los contenedores de las estaciones, en el que pueden apreciarse la base de red con toma de tierra (a), los interruptores diferenciales de protección (b) y un termostato (c) para mantener constante la temperatura interna. En el contenedor (d) está alojada la electrónica de adquisición y en (e) la fuente de alimentación y el PC industrial, iguales a los del PC nodal mostrados en la figura 3.35.*

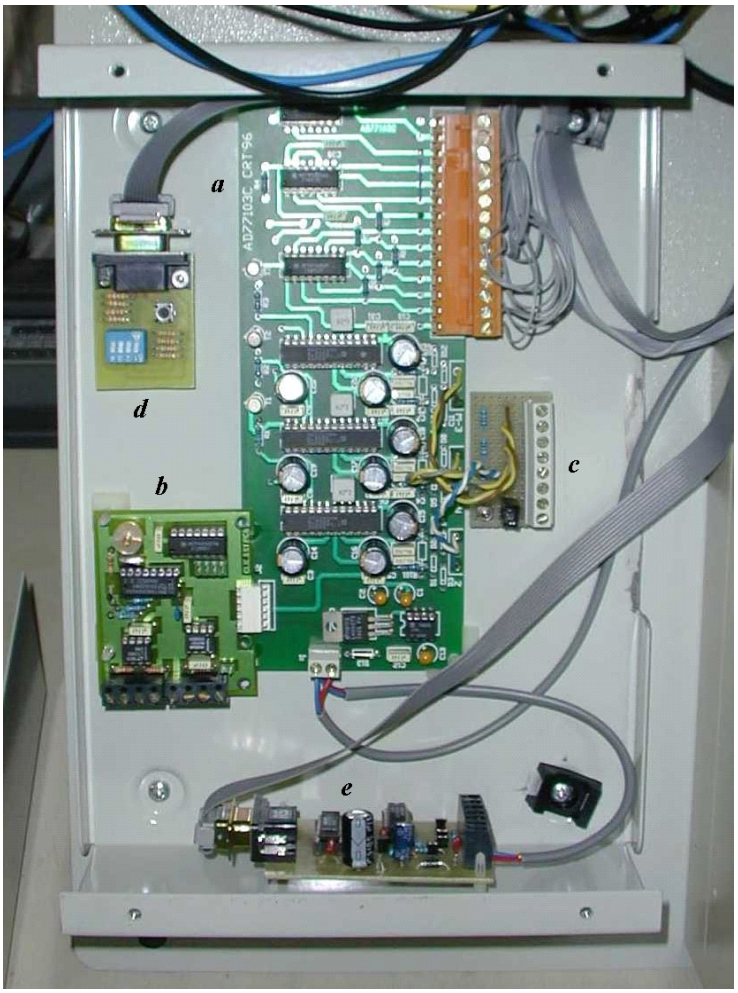


Figura 3.37. *Imagen del interior de la caja con la electrónica de adquisición. Puede verse la placa de conversores analógico/digital (a), conectada al puerto paralelo del PC industrial mediante un conector de tornillos. A ella se conectan la placa de generación de las señales de reloj (b) y una pequeña placa para la adaptación de la salida de los geófonos (c). También pueden verse las placas de código de estación (d) y la de conversión de niveles RS-232/RS-485 (e), conectadas a los dos puertos serie del PC a través de sendos cables planos.*

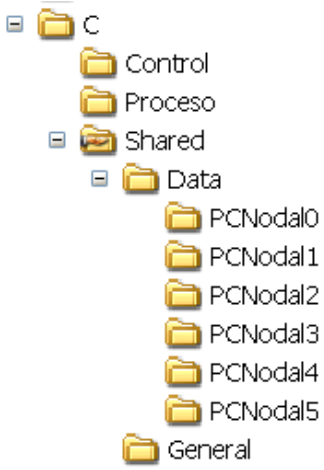


Figura 3.38. *Estructura de directorios del servidor. El directorio* SHARED *debe compartirse con acceso total. Los programas a los que deben tener acceso todos los PCs nodales se graban en* GENERAL, *mientras que los ficheros de datos están ordenados por subdirectorios. Los otros directorios contienen los programas de control y proceso, a los que sólo tiene que acceder el propio servidor.*

Como se ha comentado previamente, con las versiones de los programas presentadas en esta memoria, en las que la grabación de datos se realiza en ficheros circulares, debe operar continuamente un programa de detección de eventos, configurado con las estaciones de detección deseadas y con el directorio de grabación de ficheros de evento correspondiente. Este programa podría ejecutarse desde el propio servidor o desde otro PC externo, que accedería remotamente a los datos de los ficheros circulares a través de Internet[28].

En lo que respecta al oscilador patrón, la instalación se reduce a grabar el programa definitivo en el microcontrolador PIC16C74 e insertarlo en el zócalo correspondiente. Una vez hecho esto, lo único que hay que hacer es seleccionar la frecuencia de la señal de reloj de los CADs que se transmitirá a las estaciones. Para ello, como se vio en la sección 3.5.4.3, hay que cerrar uno de los cuatro *jumpers* de la placa del VCXO del oscilador patrón, teniendo en cuenta que la frecuencia seleccionada sea la misma que en las placas generadoras de la señal de reloj de las estaciones.

3.7.3. Puesta en marcha

Cada uno de los elementos del sistema (estaciones, PCs nodales, oscilador patrón, servidor y PC de procesado) debe configurarse para ejecutar el programa correspondiente al arrancar. En el caso de los PCs nodales y estaciones, al estar controlados por placas de PC bajo MSDOS, la configuración de arranque consiste en hacer una llamada al programa correspondiente al final del fichero AUTOEXEC.BAT. Como se vio en las secciones correspondientes al desarrollo de los programas, las placas de PC industrial tienen implementado un temporizador tipo perro guardián (*watchdog*), que se ha habilitado en ambos programas para evitar pérdidas de control. Cuando el programa pasa un determinado tiempo sin refrescar el *watchdog* se provoca un reinicio y se vuelve a entrar en el programa de adquisición.

El PC de procesado también se podría configurar para ejecutar el programa de detección de eventos al arrancar, aunque no es estrictamente necesario porque éste se puede lanzar manualmente desde la sala de control. En cuanto al servidor, en principio no debe ejecutar ningún programa, sino simplemente tener habilitada la tarjeta de red y la configuración apropiada para permitir a los PCs nodales escribir en su disco duro y al PC de procesado leer de él.

En cuanto al oscilador patrón, el programa se ejecuta directamente al conectar la alimentación, ya que se ha grabado a partir de la dirección del vector de reset de la memoria de programa del PIC, donde el contador de programa se posiciona automáticamente al arrancar. Sin embargo, las variables de tiempo al inicio del programa están a cero, por lo que antes de la puesta en marcha del resto del sistema es necesario realizar una inicialización del reloj de tiempo real. Esta inicialización es manual hasta el segundo, y se realiza mediante los pulsadores de configuración que pueden verse en la figura 3.39. Es importante tener en cuenta que, puesto que el código lento que se manda a las estaciones es un anuncio del minuto siguiente, el reloj del oscilador patrón debe ir siempre un minuto adelantado sobre la hora real.

Los pasos a seguir para la inicialización del oscilador patrón son:

- Conectar la alimentación y activar el interruptor de ON/OFF. El display LCD mostrará unos mensajes informativos, entre los cuales está el flanco de segundo de la señal de 1PPS que se usa en el programa (figura 3.40.b). A continuación el display mostrará los mensajes normales de operación, esto es, la hora y fecha y la

[28] En lo sucesivo y para simplificar las siguientes explicaciones, se considerará que el programa de detección de eventos se ejecuta en un ordenador distinto del servidor, que estará situado también en la sala de control y al que se denominará ‘PC de procesado’.

frecuencia actual del VCXO. Podrá apreciarse que la hora y fechas iniciales del reloj son las 00:00:00 del día 1 del año 99. Es probable, asimismo, que el VCXO muestre el mensaje *Unlocked* (fuera de lazo) durante los primeros segundos de operación (figura 3.40.c).

- Si el flanco de la señal de 1PPS coincide con el que se ha mostrado en el display al iniciar el programa, se puede pasar directamente a la inicialización de las variables de tiempo pulsando el botón '*Synchronization*', tras lo cual el display mostrará el mensaje de la figura 3.40.d. En caso contrario hay que seleccionar el otro tipo de flanco, para lo cual hay que presionar el pulsador de '*1PPS slope selection*' (el display mostrará el mensaje de la figura 3.40.f) y realizar la selección con el botón '+'. Para salir de la función, hay que pulsar de nuevo el botón '*Synchronization*'. Después de la selección de flanco, automáticamente se entrará en la de inicialización de las variables de tiempo (figura 3.40.d).
- La selección del año debe realizarse con los botones '+' y '-', pulsando de nuevo el botón '*Synchronization'* cuando el año sea el correcto.
- Hay que realizar la misma operación para la selección del día juliano (figura 3.40.e), hora, minuto y segundo. Una vez hecho esto, hay que esperar a que el segundo sea el anterior al correcto y pulsar de nuevo el botón de '*Synchronization*'. El microcontrolador entrará en el programa y esperará hasta el siguiente pulso de segundo para incrementar el reloj.

El PIC16C74 tiene, igual que las placas de PC, un timer tipo *watchdog*, que provoca el rearranque del oscilador patrón después de una pérdida de control. Aunque el hecho de que la sincronización sea manual provocaría que la hora que tomara el reloj del oscilador fuera falsa al reiniciar, la señal del VCXO se seguiría generando correctamente y posibilitaría el funcionamiento del sistema.

La puesta en marcha de los distintos elementos del sistema no se puede hacer en cualquier orden, dado que:

- Las tarjetas de conversión A/D precisan la señal de referencia *Sinc1* para operar. Por otra parte, lo primero que hacen los programas de las estaciones al arrancar es sincronizar sus relojes *software* de tiempo real con el del oscilador patrón mediante la señal *Sinc2*. Por tanto, el oscilador patrón debe estar operativo antes que las estaciones.
- Cuando los PCs nodales están un tiempo programado sin recibir respuesta de una estación, la hacen pasar a estado inactivo. Por tanto, los PCs nodales deben encenderse después que las estaciones.
- Los PCs nodales intentan abrir los distintos ficheros en el disco del servidor al arrancar, por lo que éste debe encenderse antes.
- El PC de procesado accede a los ficheros circulares de datos del servidor para llevar a cabo la detección de eventos, por lo que es necesario que éste último esté encendido cuando el primero ejecuta el programa de detección.

Según todo esto, el orden correcto para la puesta en marcha del sistema es:

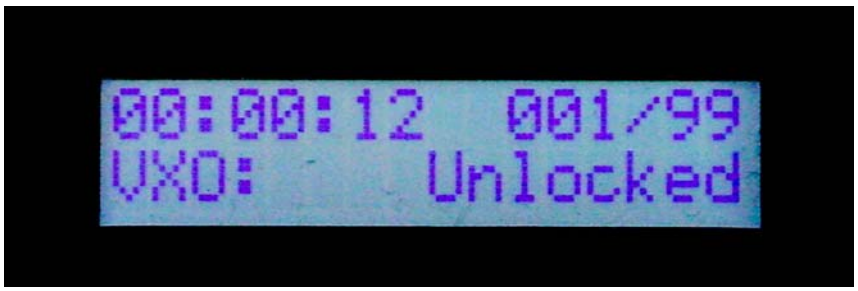
1. Encender servidor.
2. Encender oscilador patrón.
3. Inicializar oscilador patrón según los pasos indicados anteriormente.
4. Encender estaciones.
5. Encender PCs nodales.
6. Lanzar programa de detección en el PC de procesado.

Figura 3.39. *Vistas externa e interna del oscilador patrón. Pueden apreciarse el display LCD, los interruptores de retroiluminación y alimentación general, los cuatro pulsadores de control (sincronización, selección de pulso/-, + y reset) y los pasamuros metálicos para evitar la pérdida de estanqueidad. Abajo puede verse la fuente de alimentación y las tres placas diseñadas para la aplicación, montadas una sobre otra para ahorrar espacio. La situada en el nivel superior es la interfaz RS-485.*

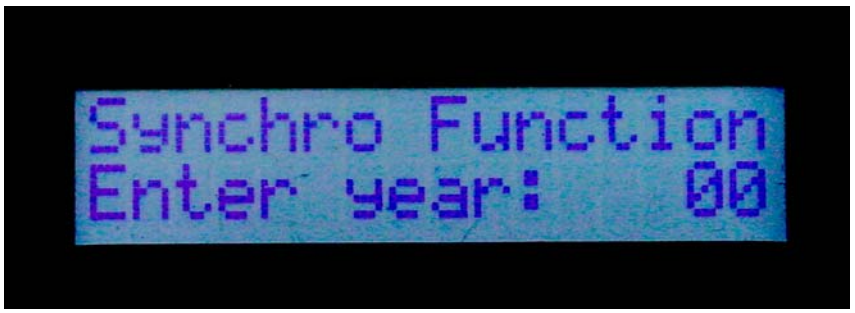
a)

b)

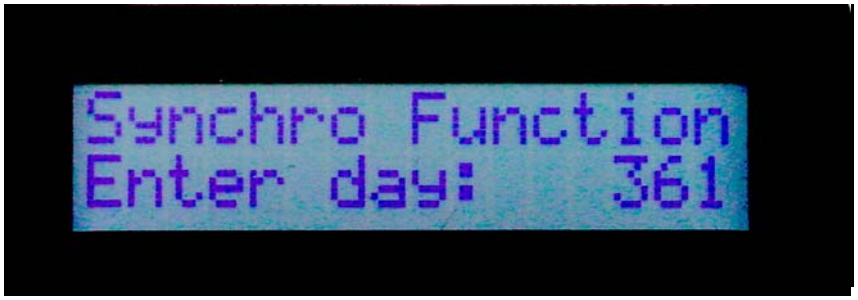
c)



d)



e)



f)

Figura 3.40. *Mensajes del display de cristal líquido del oscilador patrón.*

CAPÍTULO 4

LA ANTENA DEL VESUBIO

La segunda antena sísmica que se describe en esta memoria es la que se diseñó y desarrolló entre los años 1999 y 2002 para el volcán Vesubio, en colaboración con el *Osservatorio Vesuviano*[29] de Nápoles (Italia). El desarrollo de este *array* se solapó, durante los años 2001 y 2002, con el de las últimas antenas que se presentan en este trabajo, las portátiles para el Instituto Andaluz de Geofísica, que se describirán en el próximo capítulo.

El capítulo comienza con una introducción (4.1) en la que se realiza una breve revisión de la actividad histórica del Vesubio y se presentan los sistemas de vigilancia sísmica y geodésica actualmente operativos. Una vez caracterizado el contexto dentro del cual operará la antena, se plantean los objetivos que se pretende cubrir con el dispositivo y se especifican sus características técnicas (4.2). La parte que cubre la descripción de la antena comprende tres secciones, la primera de las cuales (4.3) presenta de manera general su estructura y funcionamiento, incluyendo la descripción de los formatos utilizados en la transmisión y grabación de los datos en el módulo de memoria. La segunda (4.4) realiza una descripción de cada uno de los módulos desde el punto de vista *hardware*, mientras que la tercera (4.5) revisa la estructura de los programas diseñados específicamente para este dispositivo. Por último, la sección 4.6 da las pautas de operación del sistema desde el punto de vista del usuario y muestra fotografías y figuras del dispositivo terminado.

4.1. Introducción

El Vesubio es un estrato volcán de dimensiones medias que alcanza una altura máxima de 1281 metros sobre el nivel del mar y que está situado en la región italiana de Campania, junto a la ciudad de Nápoles (figura 4.1). Está formado por dos estructuras claramente diferenciadas: el monte Somma, que corresponde a la estructura más antigua del sistema volcánico, y el volcán Vesubio propiamente dicho, que creció dentro de la caldera formada tras el colapso de la parte superior del monte Somma. La caldera, con forma de elipse de casi 5 km en su eje mayor, es el resultado de diversos episodios de colapso asociados a erupciones plinianas, la última de las cuales tuvo lugar en el año 79 d.C. (Cioni *et al.*, 1999). El volcán Vesubio propiamente dicho se formó en los periodos subsiguientes de actividad persistente de baja energía y a conducto abierto, con episodios puntuales de colapso de la cima. El último periodo de actividad está comprendido entre las erupciones de 1631 y 1944 (Andronico *et al.*, 1995; Cioni *et al.*, 1999, Arrighi *et al.*, 2001).

Aunque el inicio de la actividad volcánica en el área del Somma-Vesubio se remonta al menos a 400.000 años (edad de algunas lavas encontradas en perforaciones profundas), la historia del sistema Somma-Vesubio como tal comienza hace unos 25.000 años. En sus orígenes la actividad era principalmente efusiva con episodios esporádicos explosivos de baja energía. Este tipo de actividad se mantuvo hasta hace unos 19.000 años, determinando la formación de la estructura volcánica del Somma, cuyo probable perfil puede verse en la figura 4.2. La parte septentrional de este edificio está aún hoy bien conservada y corresponde a lo que actualmente se conoce por monte Somma.

Con la primera erupción pliniana de *Pomici di Base*, que tuvo lugar hace 18.300 años, comenzó el colapso del aparato volcánico del Somma y la formación de la caldera, en cuyo

[29] www.ov.ingv.it

interior las sucesivas fases de actividad volcánica dieron lugar a la formación del volcán más joven, el Vesubio. Desde sus orígenes la actividad de éste ha estado caracterizada por presentar una gran variabilidad, tanto en el tipo de erupciones como en la composición química de los magmas expulsados. A grandes rasgos esta variabilidad puede resumirse como una alternancia entre periodos de volcanismo a conducto abierto y largos periodos a conducto cerrado, con ausencia de actividad, seguidos de grandes erupciones plinianas o subplinianas. Los periodos a conducto abierto se caracterizan por intervalos de actividad estromboliana persistente, frecuentes efusiones lávicas y esporádicas, pero más devastadoras, erupciones mixtas tanto explosivas como efusivas. La figura 4.3 muestra un esquema temporal de las erupciones más importantes conocidas.

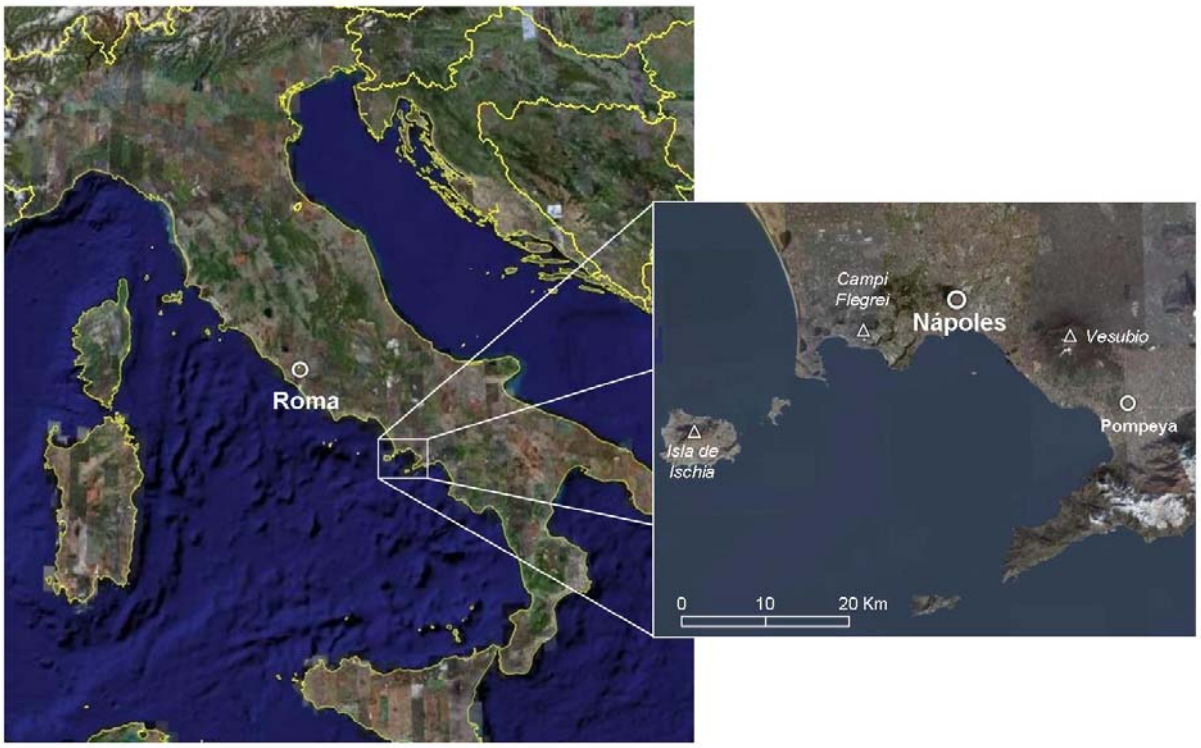


Figura 4.1. *Mapa en el que se muestra la localización del Vesubio y las áreas volcánicas de su entorno próximo (Campi Flegrei y la Isla de Ischia) (Fotos: Google Earth).*

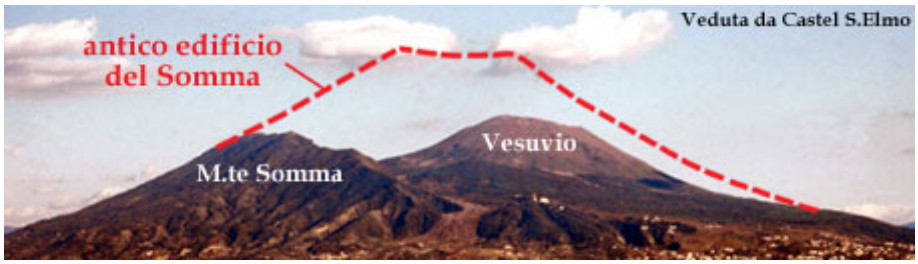


Figura 4.2. *Reconstrucción del probable perfil del antiguo volcán Somma. Como puede verse, la parte septentrional de la antigua estructura permanece en pie y es lo que hoy en día se conoce como monte Somma (tomado de la página web del* Osservatorio Vesuviano*, basado en Cioni et al, 1999).*

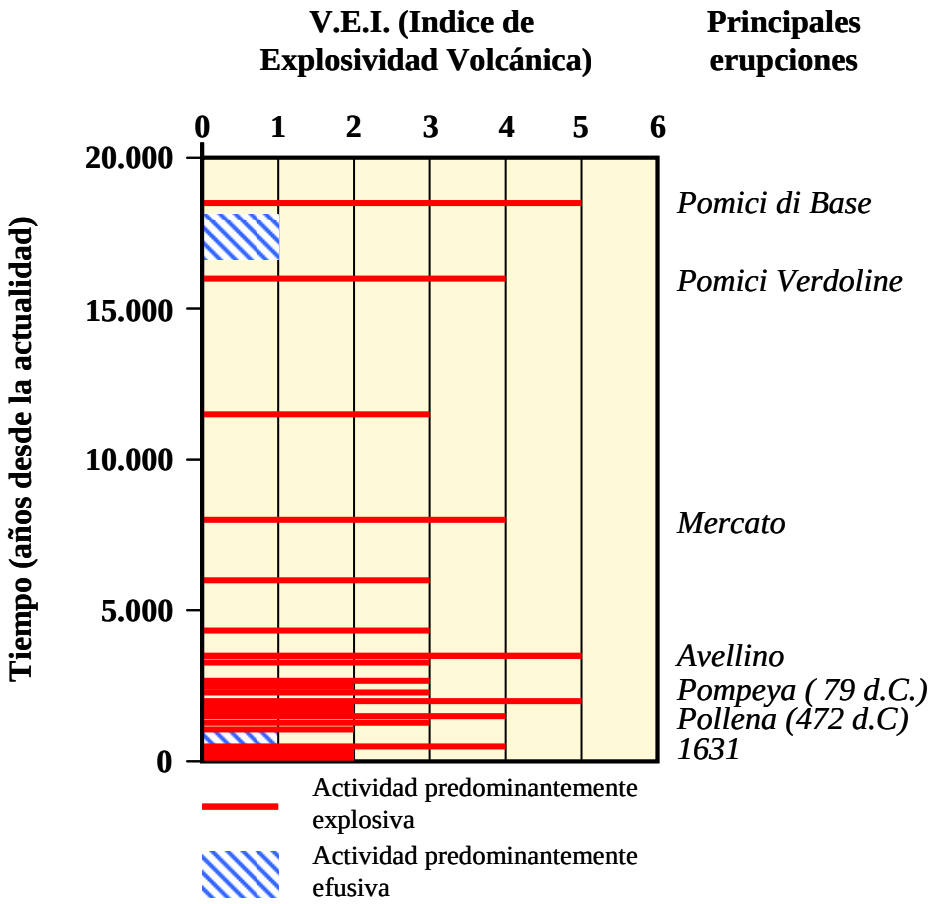


Figura 4.3. *Cuadro temporal esquemático de la actividad del Vesubio. En la escala vertical se presenta el tiempo hasta la actualidad, en la horizontal el índice de explosividad volcánica (VEI,* Volcanic Explosivity Index*), un índice ideado para clasificar la explosividad de una erupción basándose en una serie de parámetros observables en el curso de la misma o que es posible calcular mediante el estudio de los productos eruptivos (Newhall y Self, 1982). Puede observarse en la figura la alternancia entre fases de actividad predominantemente explosiva y otras fundamentalmente efusivas. Asimismo, se observa que la erupción más conocida del Vesubio, la del año 79 d.C., que devastó las poblaciones de Pompeya, Ercolano y Stabia (Zeilinga de Boer y Sanders, 2002, pp. 74-107), fue sólo una de las numerosas erupciones plinianas y subplinianas conocidas en la historia eruptiva del volcán (modificado de la página web del* Osservatorio Vesuviano*).*

La última erupción del Vesubio tuvo lugar en 1944. Actualmente el sistema volcánico está caracterizado por la presencia de un sistema hidrotermal que alimenta un campo de fumarolas en el interior del cráter y que produce una sismicidad moderada, representada por algunos cientos de terremotos al año. Tan solo los mayores de estos eventos son sentidos por la población residente en el área.

Los eventos sísmicos se localizan en el área del cráter, con profundidades hipocentrales menores de 6 km, y con magnitudes raramente mayores de 3.0 (M_{Dmax} = 3.6). De sus características espectrales, mecanismos focales y formas de onda se concluye que el mecanismo de fuente corresponde a fenómenos de fracturación de rocas. Los eventos se consideran volcano-tectónicos y no directamente asociados al movimiento de magma.

Antes de la erupción de 1944 el Vesubio se encontraba en condiciones de conducto abierto, y presentaba actividad intracratérica casi permanente salpicada por frecuentes

episodios eruptivos. Según diversas fuentes históricas, durante ese periodo, que duró cerca de trescientos años, la actividad sísmica fue intensa y las mayores erupciones fueron precedidas de enjambres de terremotos sentidos por la población. Por desgracia no se dispone de datos científicos correspondientes a esta época, dado que no existía instrumentación adecuada para llevar a cabo una observación rigurosa, aunque sí se realizaban medidas de forma experimental.

Los primeros sistemas de monitorización instrumental de la sismicidad en el Vesubio datan de la segunda mitad del siglo XIX, cuando con el nacimiento del *Osservatorio Vesuviano* (en adelante OV) el volcán, muy activo en ese periodo, se convirtió en un laboratorio natural para la experimentación de instrumentación sismométrica. De esta forma el Vesubio se convirtió en uno de los primeros volcanes del mundo, tal vez el primero, en registrar la actividad sísmica utilizando este tipo de instrumentación. Sin embargo, el desarrollo de un sistema moderno de monitorización no se inició hasta los años setenta del siglo XX, cuando se instalaron las primeras estaciones de la red sísmica, que desde entonces ha experimentado una rápida progresión hasta alcanzar la configuración actual.

En la actualidad, la red sísmica del OV está constituida por 31 estaciones sísmicas analógicas de corto periodo (20 de una componente y 11 de tres) y 3 estaciones digitales de banda ancha y tres componentes, con transmisión continua de los datos al centro de adquisición (Martini *et al.*, 2004). La red está proyectada para la monitorización de las áreas volcánicas activas de la región de Campania (Vesubio, *Campi Flegrei* y la isla de Ischia, ver figura 4.1). Proporciona, asimismo, información relativa a la sismicidad a escala regional en colaboración con la Red Sísmica Nacional italiana. La recepción de datos está centralizada en la sede del OV en Nápoles. Como puede verse en la figura 4.4, la geometría de la red prevé la mayor densidad en torno al Vesubio (12 estaciones, de las cuales 7 son de componente vertical y 5 de tres componentes) y a *Campi Flegrei* (9 estaciones, de ellas 4 de componente vertical y 5 de tres componentes). La isla de Ischia se monitoriza mediante 3 estaciones (2 de componente vertical y una de tres componentes), mientras otras 7 estaciones (3 de componente vertical y 4 de tres componentes) están distribuidas a escala regional.

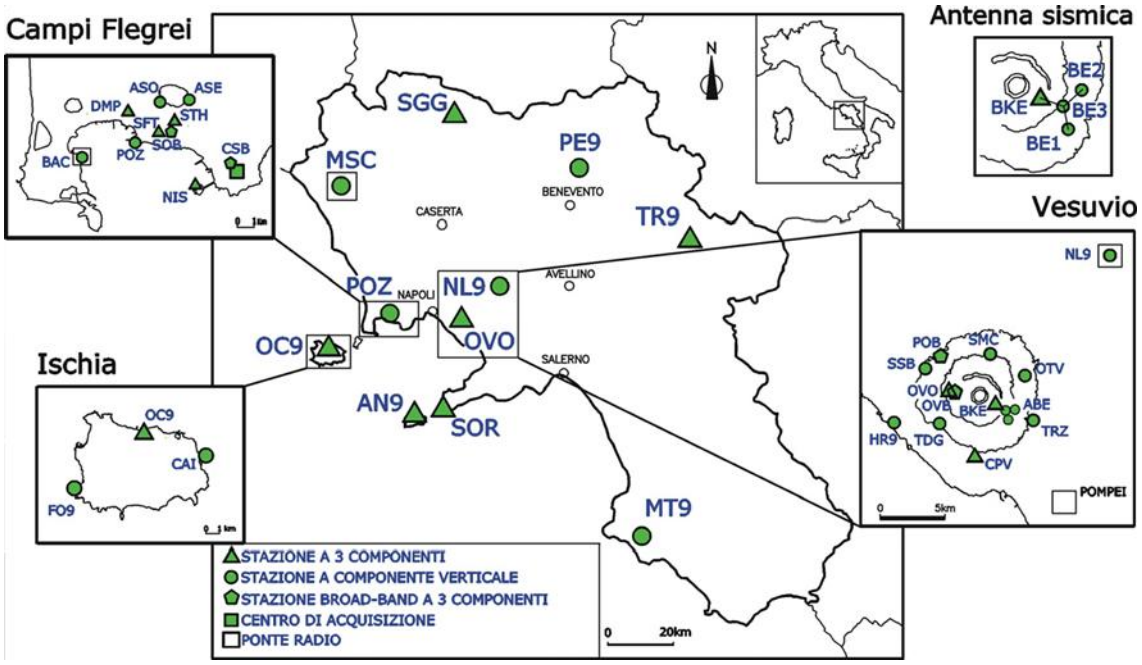


Figura 4.4. *Mapa con la localización de las estaciones de la red sísmica del OV (de la página web del Osservatorio Vesuviano).*

La distribución espacial de las estaciones se planificó teniendo en cuenta los algoritmos de cálculo hipocentral utilizados, garantizando una eficaz cobertura azimutal de las áreas que se monitorizan. Otro factor que condiciona de manera importante la geometría de la red es el alto nivel de ruido cultural existente en la zona, originado por la alta densidad de población del área. Con la distribución que se muestra en la figura 4.4 se garantizan localizaciones fiables de eventos locales con magnitud mínima $M \geq 1.0$.

Las estaciones analógicas están equipadas con sismómetros de corto periodo (T = 1s). Las de componente vertical, con sensores Mark L4-C o Geotech S13 y las de tres componentes, con Mark L4-3D o con tres sensores Geotech S13. La transmisión de datos se realiza normalmente mediante telemetría UHF, salvo en algunas estaciones en las que se utiliza línea telefónica dedicada.

Las estaciones digitales de banda ancha utilizan sensores de tres componentes Guralp CMG-40T cuya respuesta en frecuencia se extiende desde los 60 s hasta los 50 Hz. Los datos se digitalizan a 100 Hz mediante sistemas de adquisición basados en estaciones de la marca Kinemetrics, modelo K2. La transmisión se realiza en modo continuo a 9600 baudios mediante telemetría UHF, aunque actualmente se están instalando radio-modems digitales SATEL con canalización a 12.5 kHz.

Todas las estaciones se alimentan de la red eléctrica o a través de paneles solares de 75 W. El sistema de alimentación utiliza carga baterías de 3 A y baterías de plomo de 70 Ah, que garantizan una autonomía de 3-4 días en caso de fallo de la corriente eléctrica o falta de carga de los paneles solares.

La recepción de las señales de todas las estaciones se centraliza en el llamado centro de adquisición. Una vez allí, las señales son digitalizadas (en el caso de las analógicas) y procesadas por una serie de módulos de *software* en cascada, que realizan la conversión de formato y la transmisión vía Internet hasta el centro de vigilancia del OV. Además de los datos de las estaciones del Vesubio, *Campi Flegrei* y la isla de Ischia, desde mayo de 2003 se gestiona la vigilancia del volcán Strómboli, utilizando los datos que se reciben en modo continuo de la red sísmica de banda ancha instalada en la isla a raíz del episodio eruptivo que tuvo lugar entre los meses de diciembre de 2002 y julio de 2003 (ver, por ejemplo, D'Auria *et al.*, 2006, Esposito *et al.*, 2006). Entre unas y otras, en el centro de vigilancia se reciben los datos de 40 estaciones sísmicas, que entre monocomponentes y triaxiales suponen un total de 94 canales.

En el centro de vigilancia los datos se visualizan en una batería de monitores (figura 4.5) y son procesados en tiempo real por un sistema automático que realiza el análisis espectral y de polarización del ruido sísmico y calcula la localización hipocentral de los eventos.

Además de la red descrita para vigilancia sísmica, el OV gestiona varias redes de monitorización de otras variables geofísicas y geoquímicas, cuyo fin es la identificación y caracterización en tiempo real o casi real de variaciones de distintos parámetros físico-químicos que podrían constituir fenómenos precursores de una reactivación de la actividad eruptiva. En la actualidad hay operativas redes para la monitorización continua de las deformaciones del suelo y del campo gravimétrico, y redes geoquímicas para el control de la temperatura y composición química de las emisiones de gas del suelo y de las fumarolas. Por otra parte, periódicamente se realizan campañas de medida de determinados parámetros geofísicos y geoquímicos. Los datos procedentes tanto de los instrumentos en monitorización continua como de las campañas de medida son analizados por sistemas automáticos y controlados por investigadores de distintos campos.

Para la vigilancia de las deformaciones del suelo se aplican tanto técnicas clásicas (nivelación, EDM (*Electronic Distance Measurement*), inclinometría, gravimetría, mareometría) como otras más innovadoras basadas en recepción de datos por satélite (GPS, SAR). En el área del Vesubio el sistema de monitorización de deformaciones está operativo

desde principios de los años setenta. Ha ido evolucionando en el tiempo, tanto con la actualización de los sistemas ya existentes como con la incorporación de nuevos instrumentos para la vigilancia. Durante todo ese periodo no se ha registrado ningún fenómeno deformativo significativo en esta zona. Tan solo en algunas áreas de extensión limitada se han observado fenómenos de subsidencia que probablemente no están asociados a la dinámica del volcán. Así, en la zona del cráter se observa una subsidencia de unos 0.5 cm/año, que seguramente esté ligada a fenómenos de compactación causados por las pendientes acusadas.

Figura 4.5. *Fotografía de la sala de vigilancia de la red sísmica del OV. Cada ordenador controla cuatro monitores, en cada uno de los cuales se puede monitorizar en tiempo real la señal de una estación de la red. Este sistema sustituye a los registradores analógicos de tambor, con la ventaja de que desde los ordenadores además se puede acceder a programas de localización y procesado.*

No puede decirse lo mismo de las otras áreas volcánicas activas de Campania. Así, en Ischia, el sistema de monitorización de deformaciones, operativo también desde los años setenta, ha evidenciado una subsidencia significativa de la zona centro-meridional de la isla, que en el periodo 1990-2001 ha alcanzado un nivel de cerca de 7 cm. Durante el mismo periodo se ha observado un descenso del nivel del suelo de cerca de 14 cm en el sector noroeste de la isla.

Mucho más espectacular resulta el fenómeno del bradisismo de *Campi Flegrei*, cuyos episodios más significativos tuvieron lugar en los periodos 1969-72 y 1982-84, en los que en la zona costera de Pozzuoli se registraron elevaciones del nivel del suelo de cerca de 1.70 y 1.80m, respectivamente (Zollo *et al*., 2006). Los dos episodios fueron seguidos por fenómenos de descenso del suelo mucho más lentos que en la fase de elevación, de cerca de 20 cm en el primer caso y de cerca de 90 en el segundo. Esta fase de descenso, iniciada en 1985 y que continúa en la actualidad, se ha visto interrumpida esporádicamente por breves episodios de elevación, ocurridos sobre todo entre los años 1989 y 2000, con variaciones máximas de cerca de 5 cm (figura 4.6).

Otro de los factores que hay que tener en cuenta en la vigilancia de un entorno volcánico es la geoquímica de gases. En el periodo que precede a una erupción volcánica se

experimentan generalmente fuertes variaciones de las características físico-químicas de los depósitos freáticos subterráneos. En el caso del Vesubio se prevén posibles aumentos de temperatura de los gases fumarólicos, variaciones químicas hacia composiciones típicamente magmáticas (aparición de SO_2, que en la actualidad está ausente, condiciones redox más oxidantes, disminución del CH_4 y del H_2,…) y aumento del flujo de CO_2 en áreas de emisión difusa. La vigilancia geoquímica en el Vesubio está orientada al descubrimiento de estos procesos, para lo cual se realiza una monitorización continua y mediante campañas periódicas del área del cráter y de los depósitos subterráneos de agua.

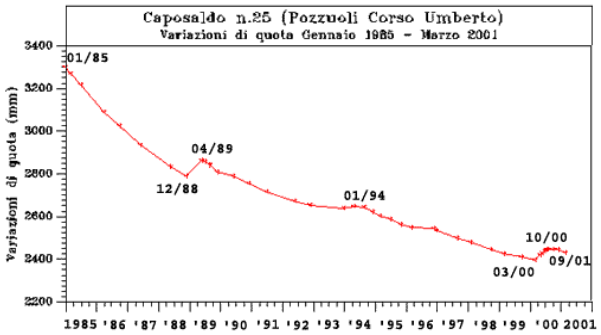


Figura 4.6. *Variaciones del nivel del suelo asociados al bradisismo de* Campi Flegrei *en uno de los puntos de máxima deformación, entre los años 1985 y 2001 (de la página web del* Osservatorio Vesuviano*).*

En la actualidad la actividad hidrotermal asociada a actividad volcánica en el complejo Somma-Vesubio puede considerarse relativamente baja. Las manifestaciones más importantes de esta actividad son emisiones fumarólicas débiles acompañadas de emisión difusa de CO_2 en el área del cráter; la presencia, en el flanco meridional del volcán, de depósitos de agua subterránea con contenidos anómalos en CO_2 y las emisiones submarinas de gases a lo largo de la costa meridional. Los registros de temperatura del campo fumarólico evidencian una continua disminución de ésta desde 1944, lo cual sugiere que el sistema hidrotermal puede estar en realidad actuando como sistema refrigerante del aparato volcánico.

En conjunto puede decirse que en los últimos años los resultados de los análisis de los distintos parámetros que intervienen en la vigilancia no evidencian variaciones significativas en el estado de actividad del volcán. Pese a ello y a que las características de la actividad en la actualidad no hacen pensar en una reactivación inminente, si se tienen en cuenta los precedentes eruptivos descritos y el hecho de que la zona de Nápoles y sus poblaciones circundantes es una de las más pobladas del sur de Italia, el Vesubio puede considerarse actualmente como uno de los volcanes con un índice de riesgo más altos del mundo.

4.2. Objetivos y especificaciones técnicas

4.2.1. Objetivos

Tradicionalmente las áreas situadas en el entorno de zonas volcánicas activas han sido muy atractivas para la población local, que en situaciones de necesidad antepone la existencia de recursos al evidente riesgo que una erupción implica. En comunidades agrícolas esos recursos están representados por la fertilidad de las tierras abonadas con los productos eruptivos. Aunque en la actualidad no puede considerarse que la actividad sobre la que se sustenta la economía de la zona metropolitana de Nápoles y sus poblaciones adyacentes sea la agricultura (desde el punto de vista económico tienen más importancia otras actividades como el turismo, el comercio marítimo o la industria alimentaria), los núcleos de población históricamente ligados a ella no han dejado de crecer. La falta de previsión y control urbanístico ha

provocado que muchos de los terrenos situados en la misma falda del volcán estén urbanizados, y gran parte de la población que habita en el área se encuentre en zonas de alto riesgo (figura 4.7). Dada la proximidad de los centros habitados y los precedentes eruptivos conocidos, la posibilidad de una erupción constituye una amenaza real que tendría una importantísima repercusión desde el punto de vista económico y humano.

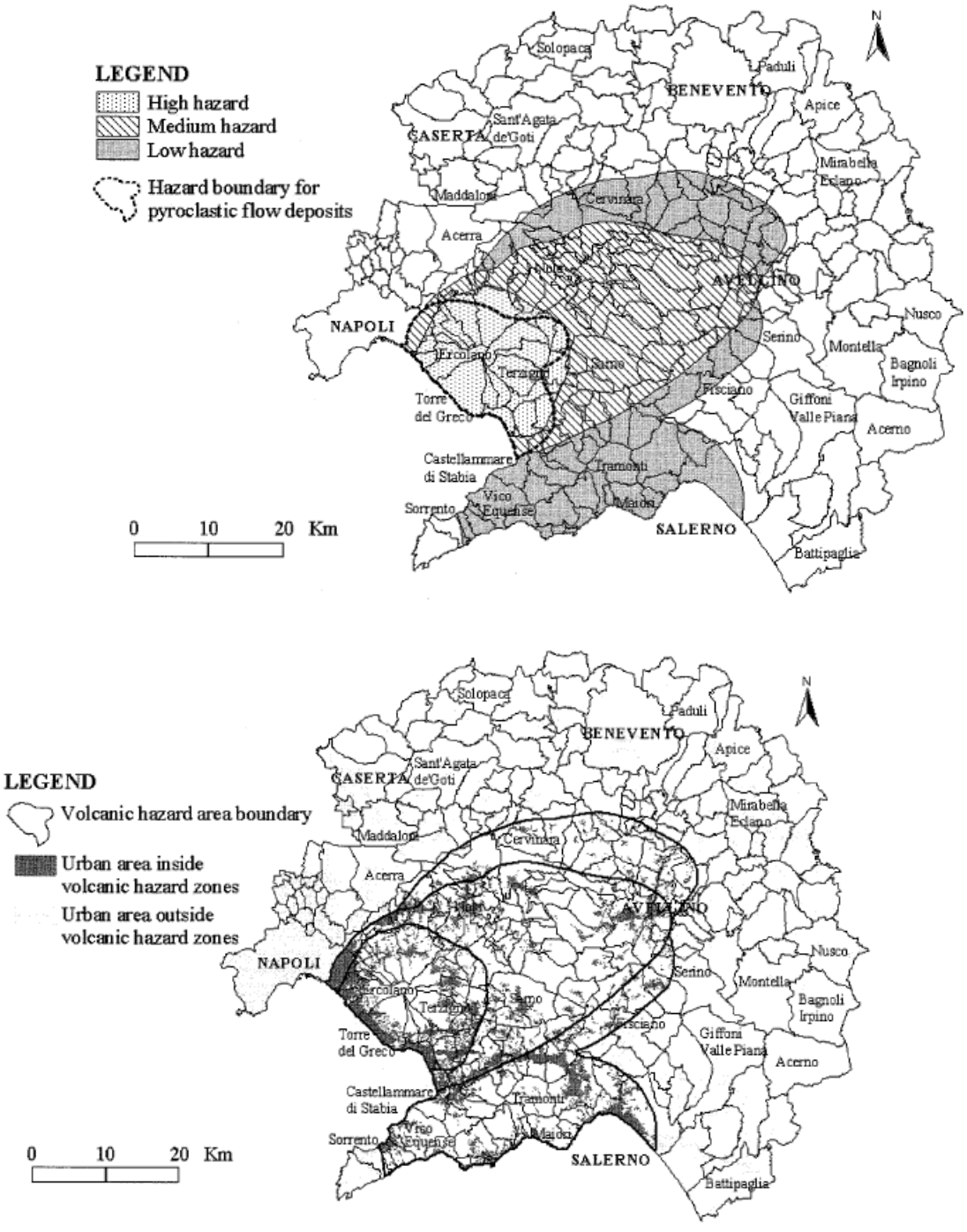


Figura 4.7. *Mapas de peligrosidad volcánica (a) y de riesgo (b) del área del Vesubio. Para obtener el mapa de riesgo se ha superpuesto información de densidad de población sobre el mapa de peligrosidad de a). Pese a no ser la más extensa, el área de máxima peligrosidad es, con mucha diferencia, la más poblada (de Lirer et al., 2001).*

Ante esta situación, resulta indiscutible la necesidad de contar con recursos orientados a la predicción, prevención y actuación en caso de un hipotético aumento de la actividad eruptiva. Si bien la seguridad de la población implica que toda la cadena de transmisión de información - desde la detección de los primeros indicios de aumento de actividad hasta la decisión de llevar a cabo una evacuación en caso de crisis severa – debería hacerse de la forma más rápida posible, también hay que tener en cuenta que las pérdidas económicas originadas por una actuación desproporcionada o precipitada son muy importantes y pueden resultar nefastas para una región. Por tanto, la gestión de una posible crisis debe llevarse a cabo con extremada precaución, lo cual engloba aspectos diversos, desde la adquisición e interpretación de los datos hasta la coordinación entre los distintos organismos responsables o la interacción con los medios de comunicación. Desde el punto de vista que nos ocupa, se hace necesario que las redes de monitorización proporcionen datos precisos y actualizados, de forma que la toma de decisiones se lleve a cabo a partir de una base fiable.

La conciencia de la amenaza que supone una posible erupción hace que los fondos destinados a las redes de vigilancia, orientados tanto a la contratación de personal cualificado como a la instalación de nuevos equipos y mantenimiento de los ya existentes, no sean escasos. Como se ha visto en la introducción, el sistema de vigilancia del Vesubio consta de varias redes de monitorización para el estudio de distintos parámetros físico-químicos cuya variación podría estar relacionada con una reactivación del sistema volcánico. En el caso de la sismología, en concreto, la red cuenta con varias subredes bien planificadas, con configuraciones espaciales adecuadas, cobertura densa de las zonas activas y tecnología moderna. Todo ello hace que en la actualidad el Vesubio sea probablemente el volcán más estudiado y conocido del mundo. En este contexto, resulta lógico preguntarse por la necesidad de dotar a la red sísmica de nuevos equipos como la antena que se presenta aquí. Teniendo en cuenta la gran cantidad de datos procedentes de los sistemas de monitorización ya operativos, ¿es necesario o, al menos, resulta de utilidad real una antena con las características de la que se describe?

La respuesta es que sí. Como se vio en el capítulo de introducción, la funcionalidad de una antena sísmica se basa en que los datos, adquiridos de forma sincronizada en un entorno espacial relativamente reducido, son coherentes entre sí, lo cual permite realizar análisis distintos a los que se llevan a cabo en redes sísmicas convencionales. La información que proporcionarán los datos de la antena sísmica que se describe resultará, por tanto, complementaria a la proporcionada por la red sísmica actualmente operativa. El uso de una antena resulta especialmente interesante para obtener información de señales sísmicas sin fases definidas (trémor volcánico, eventos de largo periodo), especialmente si las señales son de baja amplitud como las que actualmente se registran en el Vesubio, ya que en muchos casos éstas son irresolubles para redes sísmicas convencionales.

Ahora bien, también se apuntó en el capítulo de introducción que la principal limitación en el uso de antenas sísmicas para el análisis de señales volcánicas es que no proporcionan información – al menos, directamente – sobre la distancia a la fuente, sino sólo sobre la dirección de incidencia de las ondas sísmicas. Para conseguir una localización precisa es necesario utilizar técnicas indirectas como la del cruce de rayos, que mediante el empleo de dos antenas situadas adecuadamente en torno a la fuente sísmica permite calcular su posición como el punto de cruce de las direcciones de incidencia determinadas por cada uno de los *arrays*. Este es el objetivo que se plantea en el caso que nos ocupa, ya que el proyecto dentro del cual se engloba el diseño y desarrollo de esta antena pretende que sean dos los dispositivos que operen simultáneamente en torno a la región activa. Así, en la primera fase, que correspondería al trabajo que se presenta en este capítulo, se desarrollaría el primer *array*. Una vez probado en laboratorio y luego en condiciones de trabajo reales se realizaría físicamente otra antena con las mismas características o con eventuales cambios de diseño planteados a partir de la experiencia con la primera. En su configuración final, las dos antenas

se situarían en torno al cráter, con una disposición espacial tal que mediante la técnica del cruce de rayos se pudiera calcular la posición de la fuente sísmica. Los datos de las dos antenas se transmitirían por telemetría al centro de adquisición de datos y de ahí a la sala de monitorización, en la que se mostraría la posición de la fuente en tiempo casi real[30] y se programarían alertas en caso de una migración de la fuente, que podría ser una señal de reactivación del sistema volcánico.

A las ventajas propias de las antenas sísmicas hay que añadir el diseño modular y portátil que se ha adoptado en este dispositivo. Aunque en principio el emplazamiento de las antenas será casi permanente para realizar un seguimiento de la actividad del Vesubio, su diseño portátil permitirá realizar un despliegue rápido en caso de crisis sísmica en cualquier otro área volcánica o tectónica. El sur de Italia es la zona con mayor actividad volcánica de Europa, ya que además de las áreas de la región de Campania (Vesubio, *Campi Flegrei*, las islas de Ischia y Prócida y el volcán *RoccaMonfina*) en Sicilia se encuentra el volcán más activo de Europa (el Etna) y en las islas Eolias los volcanes Strómboli y Vulcano. El Strómboli, en concreto, además de su actividad característica que ha servido para bautizar este tipo de comportamiento en otros volcanes (actividad estromboliana), presenta cada cierto tiempo episodios de tipo efusivo o explosivo que pueden resultar peligrosos para la población de la isla, como el sucedido entre diciembre de 2002 y julio de 2003 (D'Auria *et al.*, 2006, Esposito *et al.*, 2006). El OV es directamente responsable de la vigilancia de varias de estas áreas activas y colabora en el estudio y monitorización de otras, por lo que una antena sísmica portátil resulta un instrumento de gran utilidad para la gestión de crisis en su entorno geográfico próximo.

4.2.2. Consideraciones de diseño

Considerando todas las circunstancias descritas, queda claro que las características técnicas requeridas para esta antena son muy distintas a las de la del Gran Sasso. Concretamente, la portabilidad de que se quiere dotar al dispositivo condiciona en gran manera muchos de los parámetros de su diseño.

Así, será de gran importancia conseguir el menor consumo de potencia posible para cada uno de los módulos, puesto que no se tendrá disponible una red de alimentación de corriente alterna, como sucedía en los laboratorios del Gran Sasso. La alimentación se proporcionará mediante baterías, que en el emplazamiento inicial del Vesubio estarán alimentadas a través de paneles solares y reguladores de carga. En la mayor parte de las campañas de intervención rápida en las que se utilicen las antenas probablemente no se contará con paneles solares, por lo que la autonomía del equipo se verá limitada por la capacidad de las baterías que se utilicen. Es, por tanto, fundamental conseguir un consumo de potencia lo más ajustado posible en todos los módulos del sistema. Esta es la razón principal de haber elegido, para el desempeño de la práctica totalidad de tareas que deben realizar los distintos módulos, microcontroladores de bajo consumo de la familia PIC, ya presentados en el capítulo de materiales y métodos. Los PCs industriales de la antena del Gran Sasso proporcionaban gran potencia de programación y la posibilidad de implementar protocolos complejos de comunicaciones, pero a costa de un consumo de potencia que resulta inadmisible en este caso.

La portabilidad de la antena también exige que todos los módulos sean fácilmente transportables, lo cual requiere que su tamaño y peso sean reducidos. Por otra parte, hay que tener en cuenta que en general un sistema portátil tendrá que operar a la intemperie, por lo que las cajas y conectores deben garantizar, según estándares bien definidos, un cierto grado de

[30] Debido al gran volumen de datos y a la complejidad del procesado de las técnicas de *array*, el análisis en tiempo real resulta casi imposible actualmente.

estanqueidad y protección bajo condiciones ambientales adversas. Por ello se eligieron cajas con protección IP65 para los módulos de adquisición y de reloj, y conectores de tipo militar para todas las conexiones exteriores. Además de un buen aislamiento frente a condiciones ambientales adversas, las cajas elegidas proporcionan comodidad para el transporte, ya que son de pequeño tamaño y fácilmente apilables. Los dieciséis módulos de adquisición de que consta el sistema en su configuración máxima, más el módulo de reloj, pueden apilarse y empaquetarse en una sola caja de tamaño medio, que puede facturarse si es necesario viajar en avión. El módulo central constituye la parte más delicada del sistema, por lo que es recomendable su transporte como equipaje de mano. Por esta razón se eligió como contenedor una caja de la serie Peli[31], que también proporciona un buen aislamiento (protección IP67) a la par que comodidad en los desplazamientos por su bajo peso y su diseño con forma de maleta con asa abatible. Más adelante, en las secciones correspondientes a la operación del sistema, se volverá sobre este tema y se mostrarán imágenes de los distintos módulos en las que se pueden apreciar los contenedores y conectores utilizados.

Mención aparte merece el tema de los cables. Teniendo en cuenta el gran número de estaciones de una antena sísmica como la que se describe y la separación espacial entre ellas, las longitudes de cable necesarias para conectar los distintos módulos son del orden de varios kilómetros (ver ejemplo en figura 4.8). Eso convierte a los cables, en general, en el elemento más pesado del sistema, y es la causa de que existan dos filosofías distintas cuando se pretende realizar un despliegue rápido de un dispositivo portátil. La primera posibilidad consiste en no trasladar los cables, sino adquirirlos en el punto donde se va a desplegar la antena, lo cual implica, en primer lugar, que tiene que haber alguien que los suministre en la vecindad del área en crisis (cuestión ésta que no resulta trivial en caso de entornos aislados o en países en vías de desarrollo). Por otra parte, también supone un trabajo adicional, ya que antes de instalar la antena en el emplazamiento elegido es necesario montar los conectores en los cables correspondientes, lo cual es sinónimo de retraso en la obtención de los primeros datos, que pueden ser de vital importancia. La decisión de no transportar los cables condiciona además la elección del tipo de conectores. Los de tipo militar, con alto grado de aislamiento y contactos fiables, son complicados de montar incluso en laboratorio, tanto más en condiciones de trabajo en campo. Por tanto, si se decide no transportar los cables es recomendable que los conectores sean sencillos de montar, aunque eso implique peor calidad de los contactos y menor aislamiento y estanqueidad en los módulos. Y, como en la etapa de planificación de un sistema siempre conviene ponerse en el peor caso, siguiendo esta filosofía existe la posibilidad de olvidarse los conectores en el punto de origen, con lo cual la antena quedaría totalmente inoperativa.

La segunda opción consiste en transportar los cables junto con todo el equipo, lo cual permite utilizar conectores fiables que garantizan buenos contactos y un aislamiento adecuado de los módulos. Además de eso, la principal ventaja de tener los cables hechos es que permiten realizar una conexión inmediata del dispositivo, reduciéndose al mínimo el tiempo de despliegue. El principal inconveniente es, lógicamente, que se reduce la portabilidad.

Como se verá en el próximo capítulo, en las nuevas antenas portátiles del IAG se ha elegido la primera opción, manteniendo la filosofía utilizada en los antiguos módulos de *array* de dieciséis bits que se utilizaron en numerosas campañas antes de desarrollar los nuevos dispositivos. Para evitar en lo posible una pérdida de aislamiento por culpa de los conectores de bajo coste, en las nuevas antenas la conexión se hace en una caja estanca en la que los cables se introducen a través de pasamuros.

En la antena del Vesubio, sin embargo, se ha elegido la segunda opción, la de fabricar los cables con conectores militares y transportarlos con el resto de los módulos de la antena. Se prefirió primar la calidad de contactos y estanqueidad de los módulos frente a la

[31] www.peli.com

portabilidad del dispositivo debido, principalmente, a que el emplazamiento en el Vesubio se supone casi permanente. Las intervenciones en áreas de crisis no sólo serán casos puntuales, sino que en general estarán centradas en áreas próximas (Etna, Strómboli, *Campi Flegrei*,…) a las que el personal y la instrumentación se desplazarán por carretera y/o barco, medios en los que no resulta tan importante reducir peso y volumen como cuando se utiliza transporte aéreo. Por otra parte, la estructura modular elegida para esta antena y la transmisión a través de una sola línea serie de los datos de cada rama hace que la longitud (y, por tanto, el peso) de los cables se reduzca notablemente (ver figura 4.8).

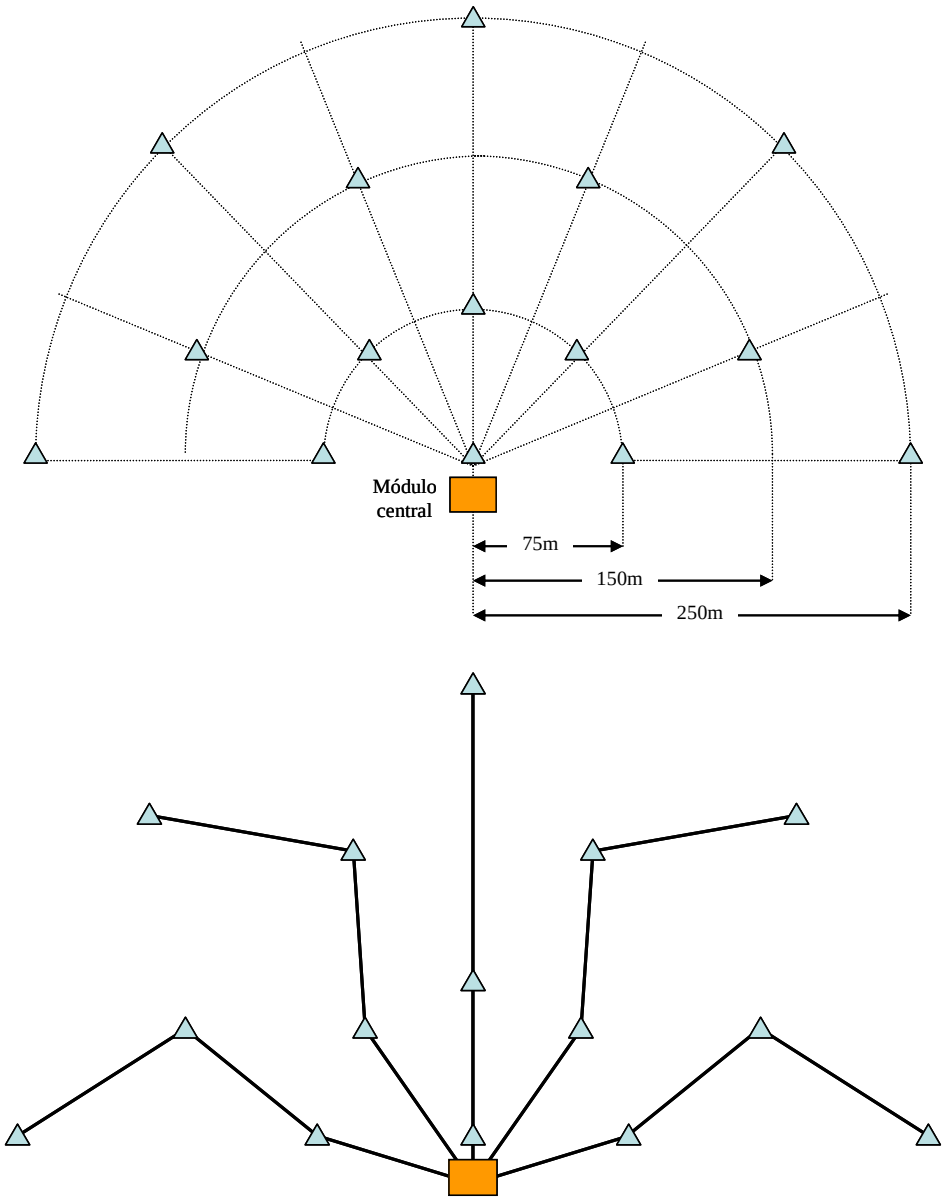


Figura 4.8. *Ejemplo de la disposición de sensores en una antena semicircular de 500 m de apertura. La longitud de cable necesaria para conectar todos los puntos al módulo central mediante líneas independientes sería de 5 x 75m + 4 x 150m + 5 x 250m = 2225m. El peso dependería del tipo de cable utilizado: para uno de cuatro pares trenzados sin apantallar podría ser de unos 70kg, que subirían a casi 90 si el cable fuera apantallado. La longitud necesaria de cable, y con ello el peso, se reduce sensiblemente si los sensores o puntos de adquisición se conectan a través de líneas serie, como en la figura de abajo. Mediante sencillos cálculos trigonométricos puede comprobarse que con la configuración dibujada la longitud de cable necesaria sería de unos 1400m, con un peso de entre 45 y 55kg aproximadamente, dependiendo del tipo de cable.*

Otra diferencia fundamental respecto a la antena del Gran Sasso es que en este caso sí es posible utilizar receptores GPS, ya que no se presentan los problemas de recepción de señal de los satélites que había en los LNGS. Los modelos de receptores GPS orientados a la implantación en sistemas embebidos, como el que se ha utilizado en esta aplicación (*Trimble Lassen SK II*), proporcionan, además de una señal de precisión de un pulso por segundo (PPS) que puede ser utilizada como referencia para conseguir una adquisición de datos sincronizada en todos los puntos de muestreo, información sobre el tiempo absoluto. Esto hace innecesario el desarrollo de un subsistema como el oscilador patrón del Gran Sasso, cuya principal función era precisamente generar una señal de tiempo real absoluto y transmitirla a las estaciones para que estas introdujeran la información de tiempo en las tramas de datos. En este caso el subsistema de tiempo (llamado aquí módulo de reloj) se reduce prácticamente a una interfaz, basada en un microcontrolador de la familia PIC, entre el módulo central y el receptor GPS.

4.2.3. Especificaciones técnicas y características

Las especificaciones requeridas inicialmente para la antena fueron:

- Posibilidad de obtener distribuciones espaciales densas utilizando un solo módulo central, lo cual implica que éste debe ser capaz de gestionar un elevado número de canales de adquisición. Como se verá en el próximo capítulo, esta filosofía es distinta a la que se usó en las nuevas antenas portátiles del IAG, en las que se prefería utilizar módulos de *array* con menos estaciones, con la consiguiente simplificación de diseño que ello implica. En ese caso, para montar una antena densa es necesario utilizar varios módulos independientes operando sincronizadamente.
- Frecuencia de muestreo configurable. Uno de los requerimientos ineludibles era llegar a las 200 mps, aunque fuera a costa de reducir el número de canales operativos.
- Alta resolución.
- Ganancia de adquisición programable.
- Sincronización de datos mediante GPS.
- Diseño portátil.
- Bajo consumo.

Tras una discusión inicial de estas especificaciones con miembros del personal técnico y científico del *Osservatorio Vesuviano*, las características fijadas para la antena fueron:

- Un máximo de 48 canales de adquisición, con una estructura de 16 módulos de adquisición que gestionan tres canales cada uno. Esta configuración permite operar con un sensor triaxial o con un sensor vertical y dos horizontales en cada módulo, orientados según los tres ejes del espacio para configurar una estación de tres componentes. También se pueden conectar tres sensores verticales a cada módulo, obteniendo configuraciones mixtas (vertical-3D) más densas. Como se verá en su momento, los canales operativos en cada configuración se habilitan mediante *software* para optimizar los recursos del sistema.
- Se decidió utilizar el conversor analógico/digital AD7710, para aprovechar la experiencia adquirida con este CAD en la antena del Gran Sasso. Por tanto, la resolución de esta antena será también de 24 bits nominales. Como ya se comentó en capítulos anteriores, la resolución efectiva de este tipo de conversores se reduce cuando aumentan la frecuencia de muestreo y la ganancia (ver tabla 2.1 para el AD7710).
- Los valores seleccionables de ganancia son todos los que permite el AD7710 (1, 2, 4, 8, 16, 32, 64 o 128).

- Frecuencia de muestreo de 50, 100 o 200 mps. En el caso de utilizar 200 mps, el número de canales operativos se reduce a 24, puesto que el volumen de datos procedentes de los 48 canales es demasiado elevado y no es posible su transmisión. En el capítulo 7 se discutirá una posible solución para aumentar el número de canales a 200 mps.
- La sincronización de los datos se gestionará mediante un módulo específico (módulo de reloj) basado en un microcontrolador PIC, que controlará un receptor GPS marca *Trimble,* modelo *Lassen SK II.*
- Bajo consumo. Como ya se ha dicho, la decisión más importante en este sentido es que todos los módulos del sistema estarán basados en microcontroladores de bajo consumo de la familia PIC. Además la elección de otros componentes del sistema, como el receptor GPS o la placa de PC industrial también estuvieron condicionadas en gran parte por este factor.

4.3. Descripción general del sistema

4.3.1. Estructura del dispositivo

La figura 4.9 muestra un diagrama de bloques del dispositivo. Como puede verse, el sistema se compone de un módulo central, un módulo de reloj y un máximo de 16 módulos de adquisición.

Los módulos de adquisición, que constan de una tarjeta de conversión A/D y de un circuito PLL, se conectan al módulo central a través de cuatro líneas serie basadas en el estándar RS-485. Cada línea serie admite un máximo de cuatro módulos de adquisición, cada uno de los cuales controla tres canales de datos. Por tanto el sistema puede gestionar hasta 48 canales de adquisición, con una frecuencia de muestreo de 50 o 100 mps. También es posible configurar el sistema a 200 mps, reduciéndose en este caso el número máximo de canales a la mitad.

El módulo central consta de un PC industrial de bajo consumo, un módulo serie/paralelo (en lo sucesivo, SerPar) y un módulo de memoria. El módulo SerPar controla la transmisión de datos desde los módulos de adquisición, así como la comunicación con el módulo de reloj. Tanto los datos de los módulos de adquisición como la información de tiempo –actualizada cada segundo- del módulo de reloj son escritos por SerPar en el módulo de memoria. Éste se utiliza como un búffer que permite al PC dedicarse a otras tareas distintas de la lectura de datos y acceder al mismo sólo una vez cada cierto tiempo, que varía en función de varios parámetros (número de módulos de adquisición operativos, frecuencia de muestreo y número de tarjetas de memoria utilizadas).

El control de los distintos procesos en los diferentes módulos del sistema se lleva a cabo mediante microcontroladores de la familia PIC, salvo el control central del proceso, tarea que corre a cargo de un PC industrial modelo *Lippert Cool Roadrunner II*, cuyas características ya se comentaron en el capítulo de materiales.

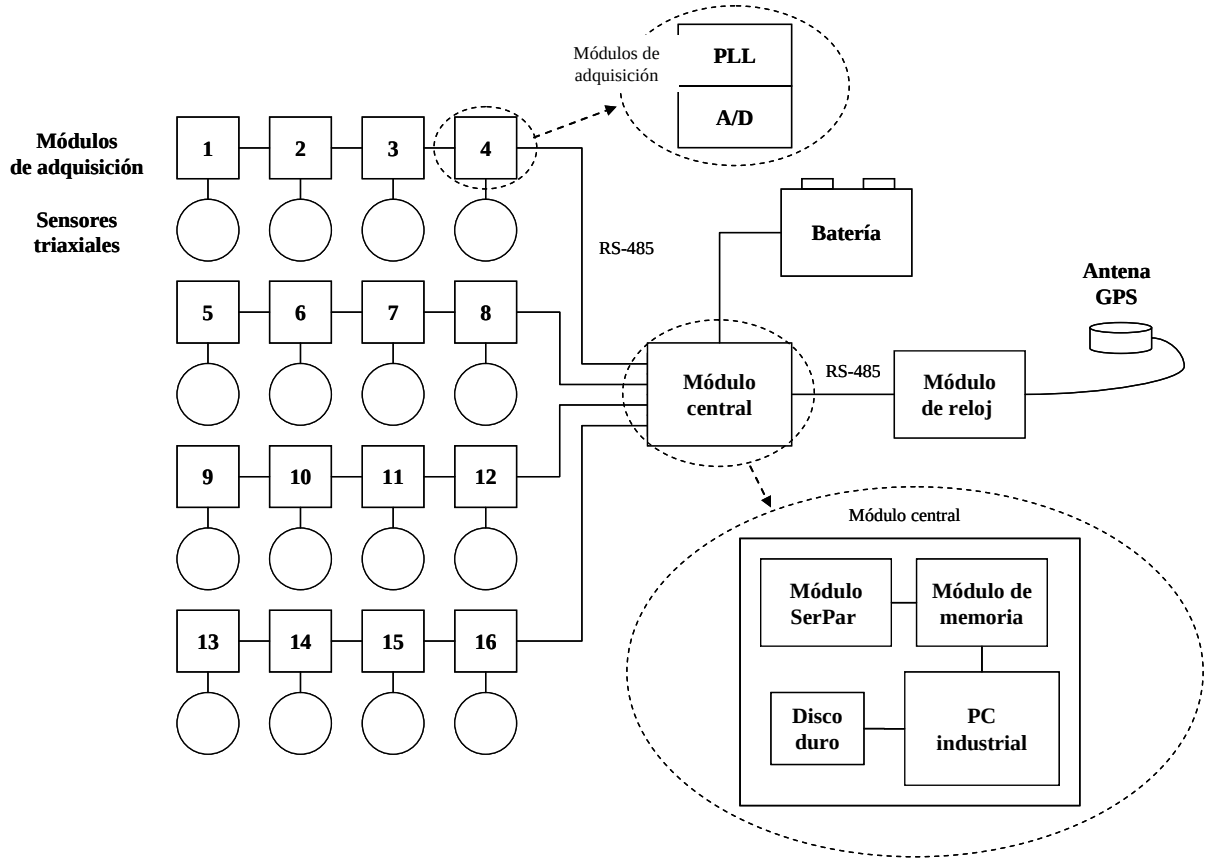


Figura 4.9. *Diagrama de bloques del dispositivo.*

4.3.2. Descripción general del funcionamiento

La adquisición de los datos en esta antena se realiza en los módulos de adquisición, cada uno de los cuales gestiona tres canales. Los sensores, que pueden ser de una componente o triaxiales, se conectan directamente a los módulos de adquisición a través de conectores de tipo militar.

El sistema de sincronización de los datos difiere del utilizado en el Gran Sasso. Mientras allí eran necesarias dos señales de tiempo que se generaban en el oscilador patrón, en este caso los módulos de adquisición toman directamente como referencia los pulsos de segundo del receptor GPS. A partir de ellos los módulos de adquisición generan, mediante los circuitos PLL locales, la señal de reloj para los conversores A/D. Igual que en la antena del Gran Sasso, en ésta se ha elegido el estándar RS-485 tanto para la transmisión de la señal de PPS como para la del resto de señales de control y datos, ya que en todos los casos la comunicación es multipunto y a distancias de cientos de metros.

Una de las ventajas de utilizar PCs industriales como controladores de la adquisición y transmisión de datos, como se hacía en el Gran Sasso, es que los PCs no tienen ningún problema en gestionar grandes búffers de memoria en los que se pueden almacenar varios segundos de datos. Gracias a ello se podían implementar protocolos complejos de comunicación, que incluían comprobación del formato de los paquetes en el módulo central, reenvíos de trama en el caso de recepción defectuosa e intercambio de comandos y tramas de control. En la antena del Vesubio, sin embargo, esto no será posible, ya que la memoria de datos de los PICs es reducida. Aunque podrían implementarse pastillas externas de memoria para la gestión de búffers, resulta más simple y más acorde con la filosofía de programación de los PICs transmitir los datos en tiempo real. Cada módulo de adquisición envía el dato muestreado justo después de adquirirlo, siendo el módulo SerPar el responsable del control de

la transmisión y de la escritura ordenada en memoria de los datos de todos los módulos de adquisición.

La transmisión de datos entre los módulos de adquisición y el módulo central se realiza de forma síncrona. Los módulos de adquisición toman como referencia para la sincronización de la transmisión los pulsos generados con este fin por el módulo SerPar, a los que en adelante nos referiremos como pulsos de transmisión. El módulo SerPar envía los pulsos de transmisión a la vez a las cuatro líneas serie, y éstos son recibidos por todos los módulos de adquisición conectados a las mismas. Cada uno de los módulos de adquisición llevará un control del número de pulsos recibidos, y transmitirá sus datos cuando el valor del contador de pulsos se encuentre entre dos valores prefijados y dependientes de la posición que ocupe el módulo dentro de su línea serie. De esta forma el tiempo asignado para la transmisión de los datos adquiridos se divide en cuatro fracciones o *slots*, en cada uno de los cuales uno de los módulos de adquisición de cada una de las líneas serie toma el control para enviar su dato. El módulo SerPar lee en paralelo los datos procedentes de las cuatro líneas serie y los escribe en el módulo de memoria en el mismo orden en que los recibe, como se verá más detalladamente en las próximas secciones.

El módulo de memoria se utiliza como un búffer de datos, en el que el módulo SerPar escribe y del que el PC industrial lee. Para controlar el proceso de lectura/escritura la memoria se divide en dos bancos (A y B) a los que no se permite el acceso simultáneo del PIC de SerPar y del PC. Para evitar conflictos el control de la lectura/escritura es gestionado unívocamente por el PIC de SerPar, que mientras está escribiendo en uno de los bancos lo bloquea y sólo permite al PC acceder al otro. El control del acceso a uno u otro banco se realiza mediante una única señal (*Bank Select A/B*), que el PIC sube o baja en función del banco en el que está escribiendo.

Hasta ahora se ha descrito, a grandes rasgos, el funcionamiento normal del sistema, en el que los procesos implicados son la adquisición, sincronización, transmisión y escritura/lectura de los datos en el módulo de memoria. Ahora bien, antes de que el dispositivo opere normalmente es necesario comunicar los parámetros de operación seleccionados por el usuario a los distintos módulos del sistema. Esta comunicación parte del PC industrial, con el que el usuario interacciona desde un PC externo mediante un programa de acceso remoto. Los parámetros de operación se almacenan en un fichero de configuración ASCII. Como se verá posteriormente, cuando el fichero se ha editado y guardado con los valores deseados de los distintos parámetros, basta con reiniciar el sistema. Al recibir alimentación los distintos módulos quedan inicialmente en un estado de espera hasta que reciben los parámetros según una secuencia de arranque, en la que el PC envía primero la información al módulo SerPar y luego éste transmite a los módulos de adquisición. En ambos casos se utiliza comunicación síncrona con protocolos propios que, por brevedad, no se han incluido en esta memoria sino en el anexo II.C.1, dentro del CD adjunto. Una vez inicializados los distintos módulos, la transmisión de datos desde los módulos de adquisición al módulo SerPar se realiza de modo síncrono, con el formato que se describe en la sección 4.3.3. La sección 4.3.4 se ocupa del procedimiento de escritura de los datos en el módulo de memoria y, por último, en el apartado 4.3.5 se compara brevemente el proceso de sincronización de los datos con el utilizado en la antena del Gran Sasso. La descripción detallada del algoritmo de sincronización se deja para más adelante, cuando ya se haya presentado la estructura de los distintos módulos desde el punto de vista *hardware*.

4.3.3. Transmisión de datos

Durante la operación normal del sistema (es decir, una vez inicializados los distintos módulos), el módulo SerPar se ocupa del control de la transmisión de datos desde los módulos de adquisición operativos y desde el módulo de reloj. La primera se realiza de modo síncrono, generando el módulo SerPar una serie de pulsos de transmisión que se utilizan para sincronizar el proceso. La comunicación con el módulo de reloj se realiza de modo asíncrono, utilizando los controladores USART implementados en los microcontroladores PIC16C74A de SerPar y del módulo de reloj.

La figura 4.10 muestra el protocolo utilizado en la comunicación con los módulos de adquisición (se ha considerado, por simplicidad, que la frecuencia de muestreo es de 100 mps). Como puede verse, el proceso completo de transmisión de datos se repite en cada periodo de muestreo. Este intervalo se divide en un primer periodo de 1 ms y otro de 9 ms. Durante el primer ms SerPar permanece a la espera, dando tiempo a que los módulos de adquisición concluyan el proceso de muestreo (los circuitos PLL de cada módulo de adquisición garantizan que este proceso no ocupa más tiempo). Durante los 9 ms siguientes, SerPar genera una serie de pulsos de transmisión, que envía a la vez a todas las líneas serie. Los módulos de adquisición responden poniendo la línea de datos en alto o en bajo, en función del valor del siguiente bit de su cadena de datos. SerPar realiza la lectura de los bits de las cuatro líneas serie en paralelo, y cada vez que termina la recepción de un byte completo lo escribe en el módulo de memoria. El número total de pulsos de transmisión que el módulo SerPar genera es de 288, correspondientes a cuatro módulos de adquisición en cada línea serie, que envían nueve bytes de datos cada uno. El estándar RS-485 no permite que más de un nodo de una línea serie transmita simultáneamente, por lo que se establece un orden de transmisión basado en la posición[32] que ocupa cada módulo en la línea. Como se verá más adelante, la posición de cada uno de los módulos puede tomar un valor entre 0 y 3, que se introduce mediante un microinterruptor situado en las placas de conversión A/D. Así, el módulo 0 de cada línea serie transmitirá sus datos al recibir los pulsos de transmisión 1 a 72; el número 1, con los pulsos 73 a 144, y así sucesivamente.

Esta secuencia se repite con cada intervalo de muestreo (10 ms). La única diferencia se produce en el primero de cada segundo (es decir, justo después de recibirse un pulso de segundo). En ese caso SerPar, en lugar de permanecer a la espera durante el primer ms, aprovecha ese tiempo para comunicarse con el módulo de reloj y solicitarle los parámetros de status y tiempo GPS. El módulo de reloj envía toda la información durante ese primer ms. Los microcontroladores de ambos módulos utilizan para esta comunicación los controladores USART que tienen implementados, con un formato asíncrono de ocho bits de datos, uno de inicio, uno de parada y sin paridad, y a una velocidad de transmisión de 156.3 kbps.

4.3.4. Escritura en memoria

Según se ha visto, SerPar realiza la lectura de los bits de las cuatro líneas serie en paralelo, y cada vez que termina la recepción de un byte completo lo escribe en el módulo de memoria (figura 4.10.b). La escritura se realiza mediante señales de control específicas y siguiendo los diagramas de tiempo de las pastillas de memoria (Sony, 2000).

La principal ventaja de esta escritura en memoria en tiempo real es que simplifica el programa del PIC de SerPar, ya que éste no se tiene que ocupar de ordenar los datos recibidos antes de escribirlos. El principal inconveniente es que los datos quedan en el módulo de

[32] El término ‘posición’ debe entenderse como posición lógica, no el lugar que ocupa físicamente el módulo en la línea serie.

memoria en un orden dependiente de la configuración física de la antena (es decir, del número de módulos de adquisición operativos y de su posición en cada línea).

4.10. a)

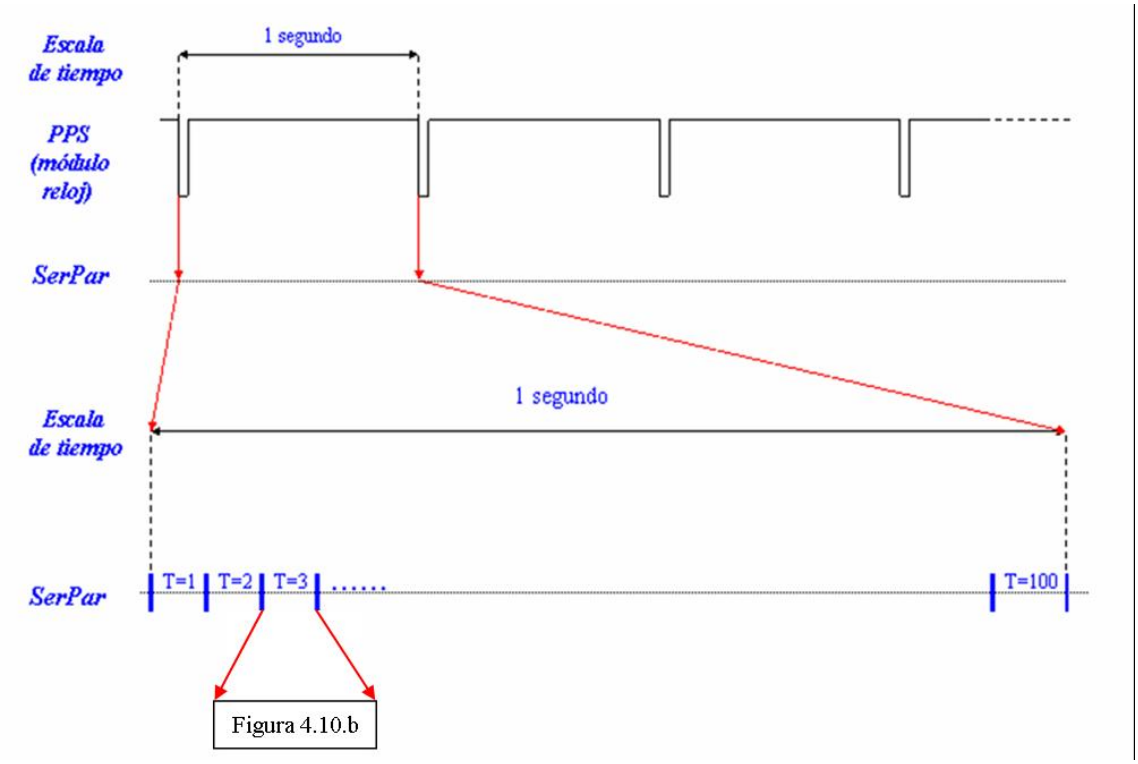


Figura 4.10.b

4.10. b)

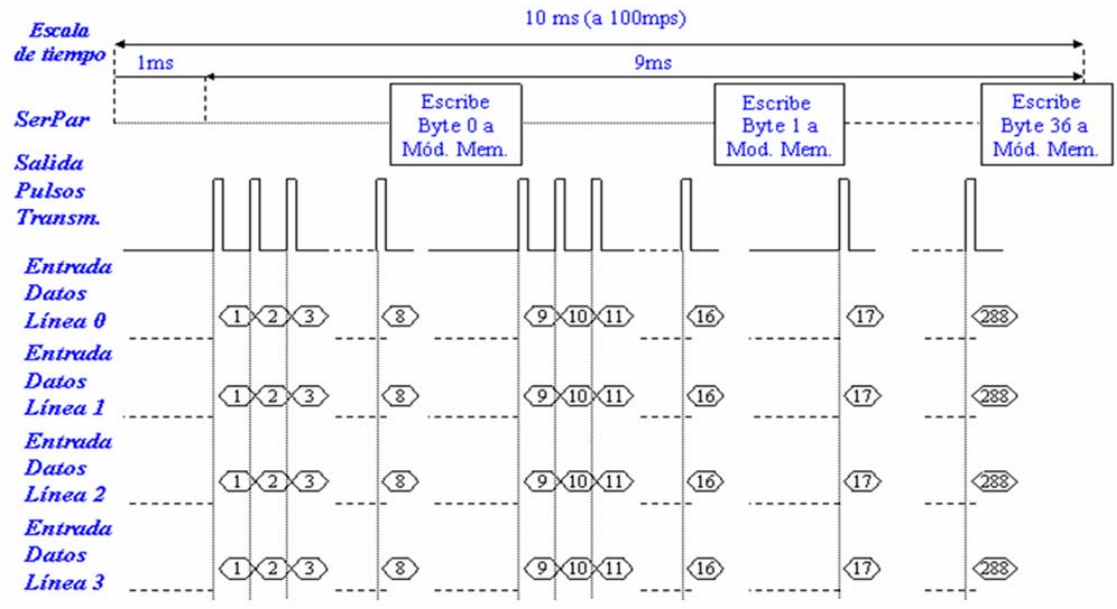


Figura 4.10. *Formato de la transmisión de datos entre el módulo SerPar y los módulos de adquisición.*

A la hora de plantear el programa se contempló la posibilidad de escribir los datos ordenados, para lo cual el PIC debería gestionar un búffer en el que almacenar los bytes transmitidos por los módulos de adquisición y escribirlos, una vez recibidos todos, siguiendo un orden prefijado e independiente de la configuración física de la antena[33].

Finalmente se decidió implementar la primera opción, debido a que el programa del PIC de SerPar debe encargarse del control y coordinación de varias tareas críticas (transmisión de datos, escritura en memoria, comunicación con el módulo de reloj, interacción con el PC), de modo que se prefirió no sobrecargarlo y dejar la labor de reordenar los datos al PC. Para ello el PC debe conocer la configuración física de la antena, información a la que puede acceder a través del fichero de configuración.

El formato de los datos en cada banco de memoria se muestra en la siguiente tabla:

Cabecera parámetros configuración
Cabecera GPS
Datos
Cabecera GPS
Datos
Cabecera GPS
Datos
...
Cabecera GPS
Datos
Bytes no usados

Como puede verse, al final de cada banco aparece un campo correspondiente a 'bytes no usados'. La razón de no utilizar todos los bytes del banco es que siempre se guardan en cada uno de ellos los datos correspondientes a segundos enteros. Aunque eso supone una pequeña pérdida de capacidad, se prefirió evitar que el último segundo quedara dividido entre dos bancos para tener un control mayor de los paquetes, ya que de esta forma los diferentes campos empiezan siempre en la misma posición de memoria.

El formato detallado de cada uno de los campos puede consultarse en el anexo II.C.2, en el CD adjunto.

4.3.5. Sincronización

Hay dos diferencias fundamentales entre el sistema de sincronización utilizado en la antena del Gran Sasso y el que nos ocupa.

En primer lugar, en esta aplicación se cuenta con la posibilidad de usar un receptor GPS, que no sólo proporciona una señal de PPS de precisión – como el patrón atómico del Gran Sasso –, sino también información de tiempo real. Esto evita la necesidad de desarrollar un subsistema complejo como el oscilador patrón del Gran Sasso.

En segundo lugar, como ya se apuntó en la descripción general del sistema, ahora los conversores A/D de los distintos módulos no trabajan con una señal de reloj común, sino que los circuitos PLL de cada módulo de adquisición la generan localmente usando como entrada

[33] Se ha dicho antes que una de las limitaciones de los PICs era el tamaño de la memoria de datos, que impedía gestionar grandes búffers sin utilizar pastillas de memoria adicionales. Sin embargo, el búffer necesario para reordenar los datos antes de escribirlos en el módulo de memoria sólo necesitaría tener capacidad para una muestra, y sí sería suficiente con la memoria de datos del PIC.

la señal de PPS del GPS. Los circuitos PLL garantizan que la frecuencia de todas estas señales locales es idéntica, y por tanto también lo es la velocidad de operación de los CADs. La principal ventaja de este sistema es que hace innecesaria la transmisión de señales de alta frecuencia, sujetas a pérdidas y distorsiones especialmente en entornos ruidosos. Además de la dificultad añadida en la gestión de estas señales, para aumentar la fiabilidad en su transmisión es recomendable utilizar cables apantallados, más caros y pesados, con los inconvenientes que eso implica en el caso de un dispositivo portátil como el que nos ocupa. Con el sistema utilizado en la antena que se describe en este capítulo, la única señal necesaria para la sincronización de la adquisición de datos es la de PPS del receptor GPS, que por su baja frecuencia no requiere precauciones especiales para su transmisión.

La descripción detallada del algoritmo seguido para la generación de la señal de reloj con la que operan los conversores A/D se realizará más adelante, cuando se comenten los programas desarrollados para los controladores de los distintos módulos de que consta el sistema.

4.4. Desarrollo de la instrumentación

En esta sección se discutirán los aspectos más relevantes relativos al diseño de los circuitos que se han desarrollado específicamente para esta antena, empezando por los módulos de adquisición (tarjeta de conversión A/D y tarjeta PLL) y siguiendo con las tarjetas del módulo central (módulo SerPar y módulo de memoria), módulo de reloj y PC del módulo central. Aunque el PC industrial es un modelo comercial en el que no se ha realizado ninguna modificación, se describirá la tarjeta diseñada para montar la placa de PC, que incluye fuentes, fusibles y otros elementos accesorios.

Antes de describir los circuitos se incluye un apartado de materiales en el que se presentarán los microcontroladores utilizados en los distintos módulos, el receptor GPS del módulo de reloj y los módulos VCXO usados en las tarjetas PLL, así como los criterios adoptados para la selección de cada uno de estos elementos.

En el dispositivo que se describe en este capítulo se utilizaron tres modelos de microcontroladores de la familia PIC: el PIC16C74 en el módulo central y los 16C73 y 16F84 en los módulos de adquisición. Se presentan en esta sección los dos últimos modelos, ya que las características y criterios de selección del PIC16C74 ya se discutieron en el capítulo anterior. Tampoco se incluye en este apartado el PC industrial del módulo central, dado que éste también se presentó en el capítulo de materiales y métodos.

4.4.1. Materiales

4.4.1.1. Los PIC 16C73 y 16F84

La positiva experiencia en la programación del PIC del oscilador patrón de la antena del Gran Sasso fue uno de los factores que llevó a centrar el diseño de los distintos módulos del sistema del Vesubio en microcontroladores de esta familia. Aunque las características del PIC16C74 eran suficientes para desempeñar el papel central en todos los módulos, sólo se mantuvo este modelo en el caso de SerPar y del módulo de reloj. Como se dijo en el capítulo de materiales y métodos al presentar esta familia de microcontroladores, una de sus ventajas es que la arquitectura común y características parecidas, sobre todo dentro de la misma gama, permiten una migración entre modelos de una forma sencilla y sin necesidad de realizar cambios importantes en el entorno de trabajo o en la filosofía de programación. Una vez asimilados los

procesos de programación, simulación en PC, escritura en el dispositivo y pruebas sobre placa para un modelo, el cambio a otro es casi inmediato y la curva de aprendizaje muy rápida.

Los programas de las tarjetas de PLL y del módulo de reloj eran, a priori, los que más analogías presentaban con el del oscilador patrón del Gran Sasso, en el sentido de que tenían que ocuparse de la sincronización de procesos tomando como entradas señales de tiempo de precisión. Para el módulo de reloj era necesario que el modelo elegido contara con puerto serie USART, ya que la comunicación con el módulo SerPar se planteó así desde un principio. Aunque en el Gran Sasso esa característica no se utilizó, el PIC16C74 utilizado allí sí contaba con USART, por lo que se decidió emplear el mismo modelo. En cuanto a los módulos de PLL, era necesario que el modelo de PIC elegido contara con *timers* y módulos de Captura/Comparación/PWM (CCP), ya que el control de la frecuencia del VCXO se iba a plantear de manera parecida a como se había realizado en el Gran Sasso. No hacían falta, sin embargo, tantas líneas de entrada/salida como se utilizaron allí, por lo que se decidió sustituir el modelo 16C74 por el 16C73. Este puede considerarse como un hermano pequeño del 16C74, ya que presenta características muy parecidas (incluso las hojas de características son comunes a los dos modelos) pero menos líneas de entrada/salida y menos prestaciones en algunos otros módulos que no se usan en nuestra aplicación. La capacidad de las memorias de programa y de datos son las mismas que en el PIC16C74 (4 Kpalabras de 14 bits y 192 bytes, respectivamente).

Por tanto, las características más destacadas del PIC16C73 son las mismas que las del PIC16C74 (ver sección 3.5.4.1), salvo las citadas líneas de entrada/salida, que se reducen de 33 a 22, y algunas variaciones en otras características que no son importantes en nuestra aplicación (menor número de líneas de entrada para el conversor A/D integrado, una fuente menos de interrupción, no tiene puerto paralelo esclavo). Como es lógico por la reducción de pines de entrada/salida, el encapsulado también varía, pasando de las cuarenta patas del CERDIP usado en el Gran Sasso a las veintiocho del PIC16C73 elegido para los módulos PLL.

En el caso del módulo SerPar, los requerimientos más importantes que debía cumplir el microcontrolador eran contar con un número elevado de pines de entrada/salida y con módulo de comunicación serie USART. La primera característica era necesaria porque debía gestionar las interfaces de comunicación con los diversos módulos del sistema (interfaces de comunicación RS-485 con los módulos de adquisición de las cuatro líneas serie, interfaz de comunicación con el PC, señales de control del módulo de memoria), lo cual requiere la utilización de más de veinte líneas de entrada/salida. En cuanto al puerto serie USART, era necesario para implementar la comunicación con el módulo de reloj. Teniendo en cuenta que el ya conocido PIC16C74 cumplía ambos requisitos y que ya se tenía experiencia en su programación, resultaba la elección más lógica.

En cuanto a los módulos de adquisición, la elección del modelo de microcontrolador estuvo condicionada, principalmente, porque se partía de un diseño previo de placa de conversión A/D en la que se utilizaba el PIC16F84 (Ortiz, comunicación personal, parcialmente reproducido en Ortiz *et al.*, 2001, p. 82). Como se verá en la próxima sección, sobre ese diseño se realizaron algunas modificaciones orientadas, sobre todo, a adaptar la interfaz serie al protocolo de comunicación previsto. El modelo de PIC, sin embargo, se mantuvo, dado que se había mostrado eficaz en aplicaciones parecidas con el diseño previo de placa. Las características principales de este microcontrolador se muestran a continuación (Microchip Technology Inc., 1998):

- Es un microcontrolador de la gama media de la familia PIC (ver sección 2.3.2.2).
- Tecnología Flash/EEPROM CMOS de alta velocidad y bajo consumo.
- Programable en código ensamblador propio de 35 instrucciones.
- Longitud de las instrucciones: 14 bits

- Memoria de programa: 1K (x 14bits), memoria Flash de 1000 ciclos de escritura/borrado.
- Memoria de datos: 68 bytes (RAM) + 64 bytes (EEPROM).
- Velocidad de operación: reloj de entrada de hasta 10 MHz, que da un ciclo de instrucción de 400 ns.
- Posibilidad de programación con protección de código.
- Modo SLEEP para ahorro de consumo.
- Amplio margen de tensión de alimentación (2.0V-6.0V).
- *Watchdog timer* (WDT) con su propio oscilador RC integrado para una operación más fiable.
- 13 pines de entrada/salida.
- Un módulo controlador de tiempo, configurable en modo de contador y timer.
- Capacidad de gestión de interrupciones, con 4 fuentes generadoras de las mismas.
- Conversor analógico-digital de ocho bits multicanal.
- Se presenta en encapsulados PDIP y SOIC de 18 pines.

4.4.1.2. El receptor GPS Trimble Lassen SK II

En lo que respecta al receptor GPS, se decidió utilizar uno de la marca *Trimble* para aprovechar la experiencia de proyectos anteriores, en los que se habían probado y utilizado varios modelos de esta casa (*SVeeSix-CM2* y *CM-3*, *Ace II*, *Acutime II*). La experiencia acumulada en estos receptores no sólo implicaba un conocimiento de la interfaz *hardware*, sino también del lenguaje de comunicación TSIP (*Trimble Standard Interface Protocol*), específico de esta marca. El TSIP es un potente protocolo binario que permite un control preciso de la configuración del receptor GPS para conseguir un comportamiento óptimo en aplicaciones muy diversas. Soporta más de cuarenta comandos de control más sus correspondientes paquetes de respuesta, lo cual permite realizar una configuración específica para cada aplicación concreta (Trimble Navigation Limited, 1999).

Por otra parte, al haber utilizado ya otros modelos de la misma familia se contaba con funciones de configuración y comunicación en alto y bajo nivel, programadas previamente para otras aplicaciones y que se podían adaptar fácilmente a los nuevos programas. El receptor elegido fue el *Lassen SK II*, que además de su facilidad de integración tiene un consumo muy reducido comparado con los otros modelos probados de la misma gama (475mW, frente a 1.25W del CM2, 1.15W del CM3, 875mW del Ace II y 800mW del Acutime II), características que lo convierten en una opción ideal para sistemas embebidos (Ortiz *et al.*, 2001, p. 331). La tensión de alimentación es de +5V ± 5%, lo cual hace necesario utilizar un circuito de fuente si se alimenta el receptor con baterías, como en nuestro caso, en el que para ahorrar consumo se usó un regulador conmutado modelo LM2576-5. Las principales características del receptor GPS se muestran a continuación:

- Capacidad de seguimiento de señales débiles que permite el posicionamiento y el uso de las señales de tiempo en entornos con baja visibilidad GPS.
- Utiliza un procesador de señales (DSP) de ocho canales con microprocesador de 32 bits, lo cual posibilita arranques en frío extremadamente rápidos (típicamente menos de dos minutos).
- Tensión de alimentación: +5V ± 5%.
- Consumo: 95mA@5V (475mW).
- Reloj de tiempo real alimentado por batería de litio de 3.6V.
- Dos puertos serie con niveles TTL.
- Posibilidad de comunicación mediante tres protocolos (TSIP, TAIP, NMEA).

- Capacidad para utilizar la técnica de GPS diferencial.
- Proporciona una señal de un pulso de segundo (PPS) para aplicaciones de precisión. Los pulsos son positivos de 10μs de anchura, y su flanco inicial está sincronizado con el tiempo UTC con una precisión de 100 ns.

4.4.1.2. El VCXO

El componente principal de los circuitos PLL, aparte del PIC de control, es un oscilador a cristal controlado por tensión (VCXO). Como se verá en la próxima sección, la tarjeta PLL permite seleccionar entre dos opciones para este dispositivo. La primera es un circuito oscilador sintonizado mediante diodos varactores, análogo al que se utilizó en el oscilador patrón del Gran Sasso. La segunda opción consiste en un módulo VCXO comercial. En la aplicación que nos ocupa se decidió utilizar la segunda opción, que presenta las ventajas que se comentaron cuando se introdujeron los circuitos PLL en el capítulo 2 (sección 2.3.5.2). El modelo utilizado fue el K1525CDM-9.8304, de la empresa *Champion Technologies* (actualmente MTRON[34]), cuyas principales características se muestran a continuación:

- Frecuencia central: 9.8304 MHz
- Margen de ajuste: ± 80 / ± 120 ppm
- Tipo de salida: HCMOS / TTL
- Temperatura de operación: -40 / +85 ºC
- Tensión de alimentación: +5 V ± 10 %
- Tensión de ajuste: +0.5 / +4.5 V
- Corriente de entrada: 26 mA (máx.)
- Corriente de salida: ±16 mA (máx.)
- Estabilidad en frecuencia: ± 50 ppm
- Envejecimiento (máx.): ± 3 ppm el primer año, ± 1 ppm los siguientes.
- Simetría (*duty cycle*): 40% / 60%
- Encapsulado metálico, compatible con DIP14.

Hay dos características que merece la pena resaltar. En primer lugar, el margen de ajuste resulta hasta ocho veces mayor que el medido en el VCXO del oscilador patrón del Gran Sasso (ver sección 3.5.4.3), lo cual permite realizar el ajuste con precisión y aprovechando toda la resolución del módulo PWM del PIC (ver sección 4.5.6). En segundo, el margen de temperatura de operación también es amplio, entre -40ºC y +85ºC. En el Gran Sasso no se hicieron pruebas de funcionamiento del VCXO a distintas temperaturas, dado que las condiciones ambientales del emplazamiento de la antena se caracterizaban por su estabilidad. Sin embargo, en este caso sí resulta importante que las características estén garantizadas para un margen amplio de temperatura, puesto que la antena debe estar preparada para operar a la intemperie y, a causa de su diseño portátil, en entornos diversos.

4.4.2. Módulos de adquisición

Los módulos de adquisición constan de dos tarjetas. La primera corresponde al circuito de adquisición analógico/digital, mientras que la segunda implementa un circuito PLL que se utiliza para sincronizar la adquisición de datos en todos los puntos. El desarrollo de ambas tarjetas, que en conjunto se denominaron SEISAD18, así como de los programas de los correspondientes microcontroladores, se llevó a cabo con la idea de hacerlos de dominio público (Alguacil *et al.*, 2002). De esta forma se pretendía ofrecer a cualquier institución o

[34] www.mtron.com

particular interesado la posibilidad de contar con un sistema de adquisición de datos sísmicos de altas prestaciones y bajo coste. Abril *et al.*, 2003, ofrecen una descripción detallada de las posibilidades del sistema SEISAD18, del que puede verse un diagrama de bloques en la figura 4.11. Las figuras 4.12 y 4.13 muestran los esquemas eléctricos de los dos circuitos.

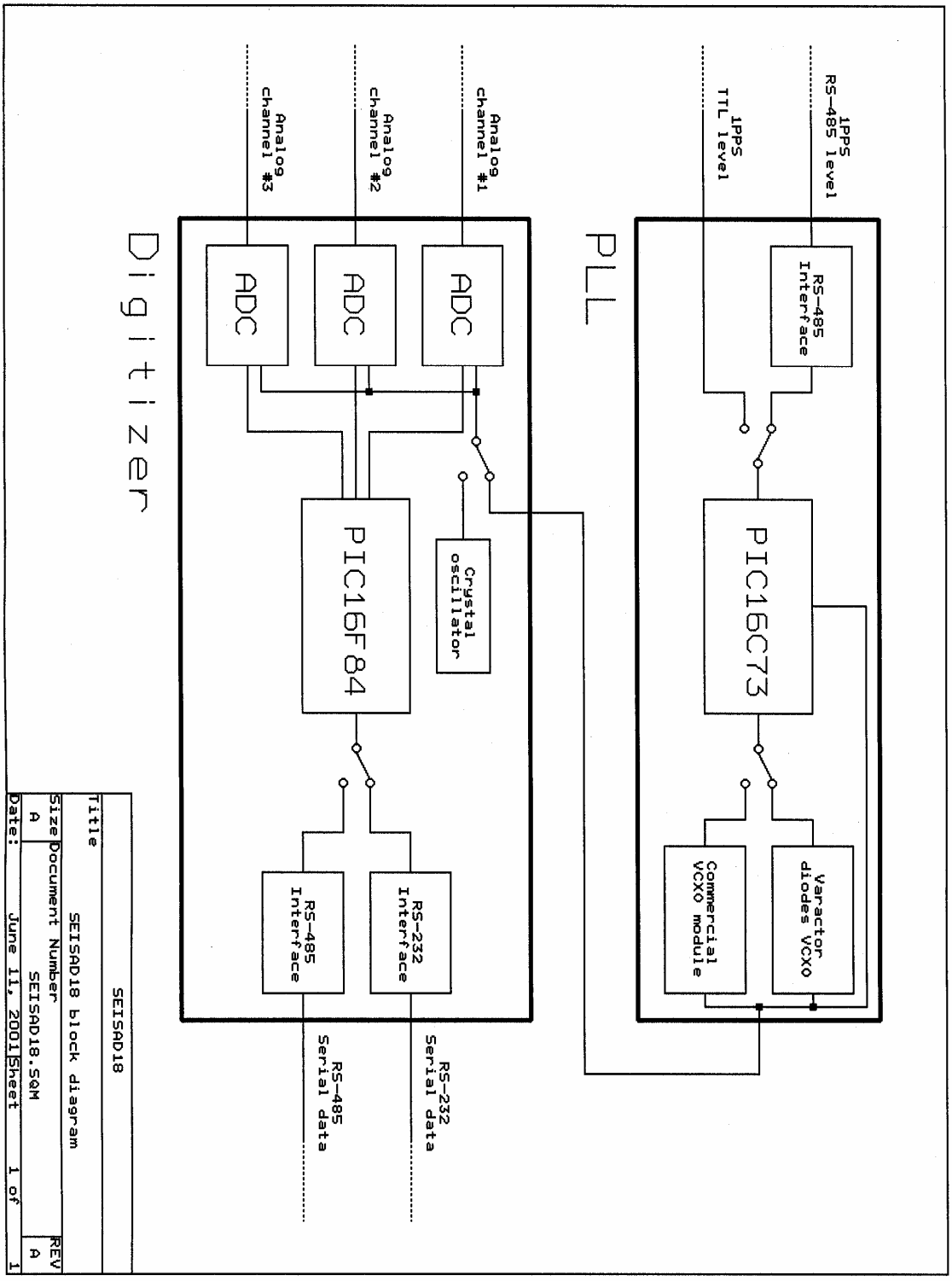


Figura 4.11. *Diagrama de bloques del conjunto placa de conversión A/D + circuito PLL.*

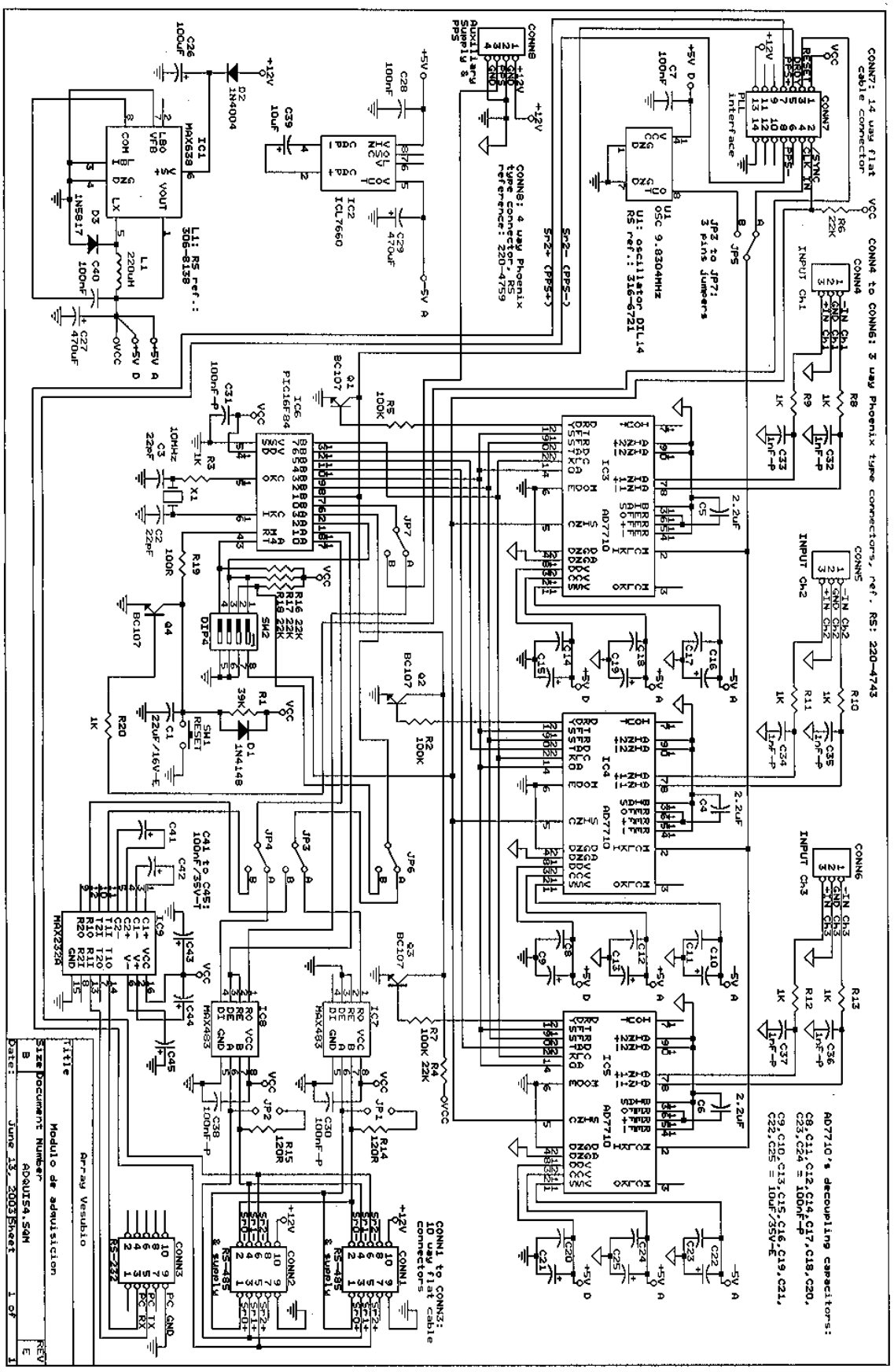


Figura 4.12. *Esquema eléctrico de la placa de conversión A/D.*

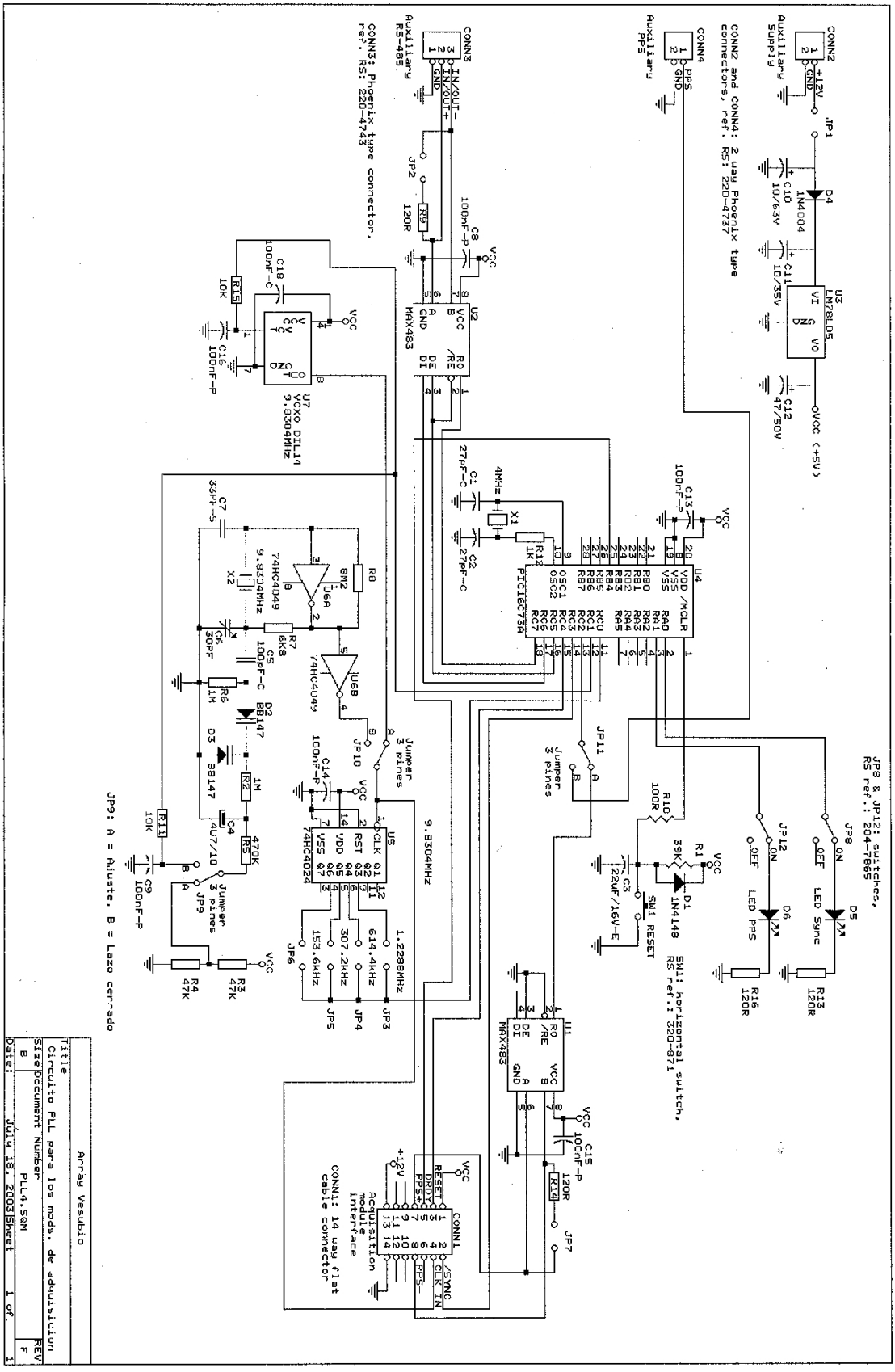


Figura 4.13. *Esquema eléctrico del circuito PLL.*

Como puede verse, la adquisición de los datos se realiza usando tres conversores A/D modelo AD7710 controlados por un microcontrolador PIC16F84. Como se ha dicho en la sección anterior, para el circuito de conversión A/D se partía de un diseño previo, en el que se hicieron las modificaciones necesarias para adaptarlo a nuestra aplicación. Los cambios afectaron principalmente al circuito de fuente (se sustituyó el regulador lineal del circuito original por una fuente conmutada para ahorrar consumo) y a la interfaz serie. Ésta se modificó para adaptarse al protocolo de comunicación diseñado para nuestra aplicación y descrito previamente (sección 4.3.3 y anexo II.C.1), implementando, como se puede ver en la figura 4.12, dos transceptores RS-485 en configuraciones distintas. Así, IC7 está configurado como receptor, puesto que los pines de conmutación entre transmisión/recepción (RE y DE) están en nivel bajo. IC8, sin embargo, puede configurarse como receptor o transmisor, puesto que la señal de conmutación entre los dos estados es controlada por el PIC. Los dos transceptores de la placa de conversión A/D, junto al integrado U1 del circuito PLL (figura 4.13) conforman tres interfaces o canales distintos de comunicación RS-485, que en los conectores CONN1 y CONN2 de la figura 4.12 aparecen marcados con los nombres Ser0, Ser1 y Ser2, y cuyas funciones se describen en la próxima sección.

Además de las interfaces para comunicación RS-485, en la placa de conversión A/D se implementó un transceptor de RS-232 para comunicación de datos usando este estándar. Si bien esta interfaz no se usa en nuestra aplicación, aporta flexibilidad a las tarjetas permitiendo su uso en futuros sistemas. Y es que la filosofía de diseño del conjunto placa A/D + PLL va más allá de este proyecto, ya que está orientada a permitir el uso de las dos tarjetas en distintas configuraciones, adaptándose a las necesidades concretas de cada aplicación (ver Abril *et al.*, 2003). El diagrama de bloques de la figura 4.11 muestra algunas de las posibles opciones de configuración, que se seleccionan mediante los enlaces de tipo *jumper* que se muestran en las figuras 4.12 y 4.13. Así, la interfaz RS-232 se seleccionaría poniendo los *jumpers* JP3, JP4 y JP6 de la figura 4.12 en la posición B.

También es posible elegir la fuente de reloj para los conversores A/D. En aplicaciones en las que se usa una sola tarjeta A/D puede ser suficiente con un oscilador a cristal que se monta en la propia tarjeta, pero cuando se usan varias tarjetas A/D y se pretende que la adquisición de los datos esté sincronizada en todas ellas es necesario tomar la señal del circuito PLL. El PLL usa como referencia para generar la señal de reloj una señal de un pulso por segundo (PPS), que normalmente se tomará de un receptor GPS, aunque podría usarse cualquier otra fuente lo suficientemente precisa. En aplicaciones donde el receptor GPS esté próximo la tarjeta PLL puede usar directamente la señal de PPS, pero en otros casos será necesario transmitirla por RS-485 y habilitar, por tanto, la interfaz implementada en dicha placa (integrado U1).

En la tarjeta PLL también se pueden seleccionar varias opciones, la más importante de las cuales es el circuito VCXO. Como se comentó en la sección anterior, es posible elegir entre un circuito realizado con componentes discretos o un módulo VCXO monolítico comercial. En el primer caso el diseño es el mismo que en el circuito equivalente para el oscilador patrón de la antena del Gran Sasso, con el cambio del valor de frecuencia del cristal de cuarzo, que en lugar de 2.4576 MHz pasa a ser de 9.8304 MHz. Según se explicó en la sección 3.4.4.2, dicho valor es el más adecuado para la operación de los conversores A/D porque permite programarlos a frecuencias de muestreo exactas de 50, 100 y 200 muestras por segundo. El aumento de la frecuencia del cristal hizo necesario sustituir los inversores 4049 del circuito del Gran Sasso por un modelo equivalente de la serie HC. Por otra parte, los diodos varactores (responsables en último término de la variación de frecuencia de la señal de salida en respuesta a variaciones de la tensión de entrada) utilizados en el Gran Sasso (BA102) eran difíciles de conseguir cuando se diseñó la placa PLL, por lo que se decidió utilizar otro modelo de características parecidas y con mayor disponibilidad (BB147).

En cualquier caso, y como ya se ha indicado antes, en la aplicación que nos ocupa se optó por utilizar circuitos VCXO comerciales (modelo K1525CDM-9.8304, de la empresa *Champion Technologies*), cuyas prestaciones son mejores y están garantizadas por el fabricante. El procedimiento seguido por los circuitos PLL para lograr la sincronización del proceso de adquisición de datos en todos los módulos de adquisición se describirá en la sección 4.5.6.

Según todo lo visto aquí y en la descripción general del sistema, en la configuración que nos ocupa la posición de los jumpers en las tarjetas A/D y PLL debe ser:

Tarjeta A/D (Figura 4.12)	
Nombre	*Posición*
JP1	Deben cerrarse (ON) en las tarjetas situadas en el extremo de cada línea serie para habilitar las resistencias de fin de línea RS-485
JP2	
JP3	A
JP4	A
JP5	A
JP6	A
JP7	A
SW2 (1/8)	OFF
SW2 (2/7)	OFF
SW2 (3/6)	Bit más significativo del código identificativo de cada módulo A/D dentro de la línea serie
SW2 (4/5)	Bit menos significativo del código identificativo de cada módulo A/D dentro de la línea serie

Tarjeta PLL (Figura 4.13)	
Nombre	*Posición*
JP1	OFF
JP2	OFF
JP3	OFF
JP4	OFF
JP5	OFF
JP6	OFF
JP7	Debe cerrarse (ON) en las tarjetas situadas en el extremo de cada línea serie.
JP8	ON si se quiere habilitar el LED D5, OFF en caso contrario
JP9	No afecta al funcionamiento del circuito mientras JP10 esté en la posición A
JP10	A
JP11	A
JP12	ON si se quiere habilitar el LED D6, OFF en caso contrario

4.4.3. Módulo serie/paralelo (SerPar)

La figura 4.14 muestra el esquema eléctrico del módulo serie/paralelo. Este circuito es básicamente un conjunto de interfaces diseñado para comunicar entre sí el resto de los módulos del sistema. Además implementa un circuito de fuente que proporciona alimentación tanto a la propia placa como a las tarjetas del módulo de memoria.

El responsable del control de éste módulo es un microcontrolador PIC16C74A. La comunicación con los distintos módulos se realiza a través de las siguientes interfaces:

- Comunicación con el módulo de reloj.- Interfaz RS-485 formada por los integrados U6, U7 y U8. Los dos primeros están configurados como receptores, para los PPS y datos del GPS, respectivamente. U8 está configurado como transmisor, para el envío de caracteres de control al GPS. La comunicación se realiza a una velocidad de 156.3 kbps, utilizando el puerto serie USART implementado en el PIC.
- Comunicación con los módulos de adquisición.- Hay tres interfaces distintas, llamadas Ser0, Ser1 y Ser2:
  - Ser0: formada por los integrados U2 a U5. Están configurados permanentemente como transmisores, y se usarán para enviar los pulsos de transmisión.
  - Ser1: interfaz semi-dúplex formada por los integrados U9 a U12. Se usará para comunicar a los módulos de adquisición los parámetros de operación y, después, para la recepción de los datos de las cuatro líneas serie. La conmutación entre los estados de recepción y transmisión se realiza mediante la señal RB1 del PIC16C74A.
  - Ser2: formada por los integrados U14 a U17, configurados permanentemente como transmisores, que se usarán para enviar la señal de PPS a los circuitos PLL.
- Comunicación con el módulo de memoria.- Las señales utilizadas por el PIC del módulo SerPar para la escritura en memoria se comparten con las tarjetas del módulo de memoria a través de un *back-plane* de formato Eurocard. La asignación de las señales en el *back-plane* se muestra en la tabla 4.1.
- Comunicación con el PC.- La interfaz para la comunicación entre el PC y el módulo SerPar se reduce a tres señales. Éstas están conectadas a través del *back-plane* a la placa de control del módulo de memoria, desde donde el PC accede a ellas a través de un conector para puerto paralelo. Las tres señales son:
  - *IN Configuration* y *OUT Configuration*.- Son las señales utilizadas por el PC para enviar al módulo SerPar los parámetros de configuración del sistema al iniciar la operación, según el formato descrito en el anexo II.C.1.
  - *System reset*.- El PC activa esta señal para realizar un reinicio de todo el sistema (ver sección 4.5.2).

El hecho de que se haya definido SerPar como una colección de interfaces de comunicación no debe restar importancia a la labor que desempeña. Antes bien, éste módulo juega un papel central en el control del sistema, ya que la comunicación con el resto de los subsistemas implica la realización de tareas complejas, como la sincronización de la comunicación con los módulos de adquisición, la gestión simultánea de los datos de las cuatro líneas serie o la escritura de los mismos en el módulo de memoria, intercalándolos con la información de tiempo procedente del módulo de reloj. Teniendo en cuenta que, una vez inicializado el sistema, la intervención del PC es mínima (gracias sobre todo a la descarga de trabajo que le proporciona el búffer de datos del módulo de memoria), el módulo SerPar puede considerarse el verdadero corazón del dispositivo. La descripción del programa del PIC que controla este módulo se realizará en la sección 4.5.4, pero las referencias a SerPar serán continuas en el resto de las secciones debido a su activa participación en casi todos los procesos implicados en la operación del sistema.

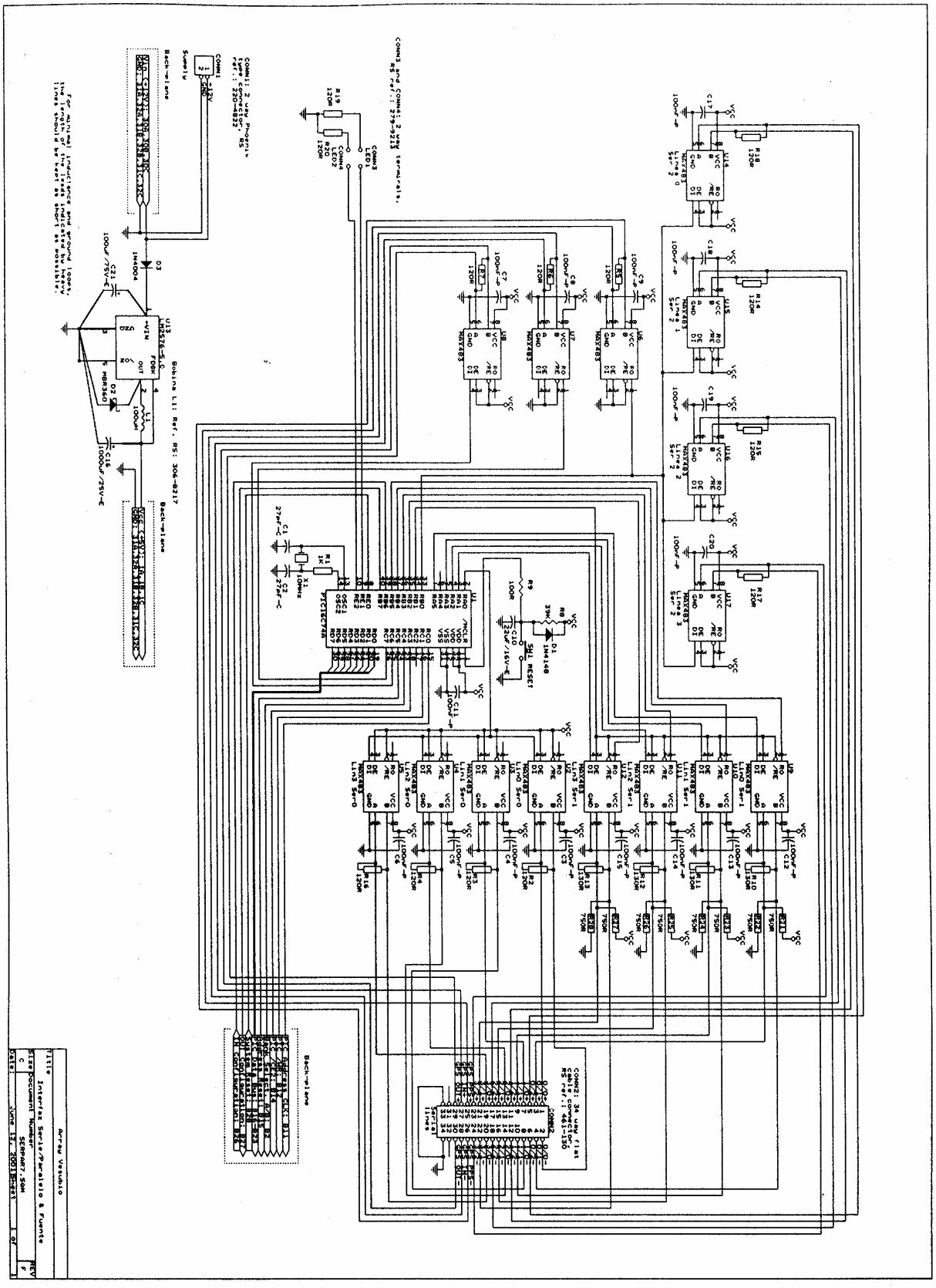


Figura 4.14. *Esquema eléctrico de la interfaz Serie/Paralelo (SerPar) del módulo central.*

Pin conector back-plane (Fila / Nº pin)	*Señal*
A1	Vcc (+5V)
A2	/WR Bank A
A3	-------
A4	/CE2 Bank A
A5-A21	Bank A Address bus
A22-A29	Bank A Data bus
A30	+12V
A31-A32	GND
B1	Vcc (+5V)
B2	Bank selection signal A/B
B3	A Bank A (Row selection signal)
B4	B Bank A (Row selection signal)
B5	C Bank A (Row selection signal)
B6	D Bank A (Row selection signal)
B7	A Bank B (Row selection signal)
B8	B Bank B (Row selection signal)
B9	C Bank B (Row selection signal)
B10	D Bank B (Row selection signal)
B11	PIC Address CLK
B12	PIC /WR
B13	-------
B14	PIC /CE2
B15	Address Reset
B16-B23	PIC Data bus
B24-B25	-------
B26	IN Configuration
B27	OUT Configuration
B28	System Reset
B29-B30	+12V
B31-B32	GND
C1	Vcc (+5V)
C2	/WR Bank B
C3	-------
C4	/CE2 Bank B
C5-C21	Bank B Address bus
C22-C29	Bank B Data bus
C30	+12V
C31-C32	GND

Tabla 4.1. *Asignación de señales en el* back-plane *del módulo central.*

4.4.4. Módulo de memoria

El módulo de memoria se compone de una tarjeta de control de memoria (figura 4.15) y una o varias (hasta un máximo de cuatro) tarjetas de expansión de memoria (figura 4.16). Como en el caso de los módulos de adquisición, cada tarjeta de expansión de memoria debe tener un número identificativo para permitir a la tarjeta de control distinguirla del resto. Dicho código se introduce a través del conjunto de microinterruptores SW1 a SW4 de cada tarjeta de expansión. Hay un microinterruptor para seleccionar cada pastilla de memoria, por lo que se deben activar un total de ocho en cada placa. Así, en la primera tarjeta de memoria deben activarse los microinterruptores 1 a 4 de SW2 y SW4, dejando el resto (5 a 8 de SW2 y SW4 y 1 a 8 de SW1 y SW3) desactivados (posición OFF). Si se usa más de una tarjeta de memoria, en la segunda deben activarse los microinterruptores 5 a 8 de SW2 y SW4, dejando el resto desactivados. En la tercera se activarán los números 1 a 4 de SW1 y SW3 y en la cuarta los números 5 a 8 de SW1 y SW3. En todas las configuraciones del sistema es necesario el uso de al menos una tarjeta de memoria, incluso cuando se trabaja en modo TEST (ver sección 4.6.5). Como se verá allí, para seleccionar el modo TEST se introduce el valor 0 en el parámetro 'Número de tarjetas de memoria' del fichero de configuración, pero es sólo una manera de comunicarle al sistema que debe arrancar en ese modo de operación. El número de tarjetas de memoria operativas puede ser 1, 2 o 4. Con objeto de simplificar el cálculo de los parámetros de escritura en memoria, el programa del módulo SerPar no permite el uso de 3 tarjetas.

Tanto la tarjeta de control como las de expansión de memoria se conectan al *back-plane* del módulo central, a través del cual toman la alimentación (del circuito de fuente del módulo SerPar) y comparten señales de control y datos.

Como se apuntó ya en la descripción general del sistema, la memoria está organizada en dos bancos a los que el PIC del módulo SerPar y el PC acceden alternativamente. El control de acceso a uno u otro banco lo realiza el PIC de SerPar mediante la señal *Bank Select A/B*: cuando ésta vale 0, el PIC accede al banco A y el PC al B, y viceversa. La tarjeta de control de memoria está diseñada de modo que el PIC de SerPar sólo puede escribir, mientras que los accesos del PC son siempre para lectura.

El acceso a las memorias es secuencial. Tanto el PC como el PIC de SerPar usan líneas únicas de control del bus de direcciones (*PC Address CLK* y *PIC Address CLK*, respectivamente), por las que mandan pulsos de incremento que activan varios contadores en cascada (en la figura 4.15, U3 y U4 para el banco A, U5 y U6 para el B). Los bits menos significativos de la salida de los contadores forman los buses de direcciones de los dos bancos, mientras que los cuatro bits más significativos sirven para seleccionar la fila (o pastilla de memoria). La puesta a cero de los buses de direcciones de los dos bancos es realizada por el PIC de SerPar reseteando simultáneamente todos los contadores (señal *Address Reset*). Esto lo realiza justo antes de cambiar el estado de la señal *Bank Select A/B*, de modo que cuando el PC tiene acceso a un nuevo banco el bus de direcciones se encuentra en la posición inicial.

La lógica de la tarjeta de control de memoria debe permitir un acceso compartido por PIC y PC de los dos bancos de memoria. Para conmutar entre uno y otro banco se utilizan demultiplexores 74HC157 (U1 y U2) que redirigen las señales de control de memoria del PIC y del PC a cada uno de los bancos en función del estado de la señal *Bank Select A/B.*

El PIC de SerPar y el PC tienen que compartir también los buses de datos de los dos bancos de memoria. Para ello se utilizan búffers triestado (U8 a U11) activados por la señal de selección de banco *Bank Select A/B* o por su invertida. Así, cuando el PIC tiene el control del banco A (*Bank Select A/B* a 0) la salida del integrado U8 queda en alta impedancia, permitiendo al PIC tomar el control del bus de datos del banco A a través de U10. Al mismo tiempo la salida de U11 también queda en alta impedancia, permitiendo al PC leer el

contenido del bus de datos del banco B a través de U9. Análogamente ocurre cuando el PIC tiene el control del banco B, permitiendo al PC leer el banco A.

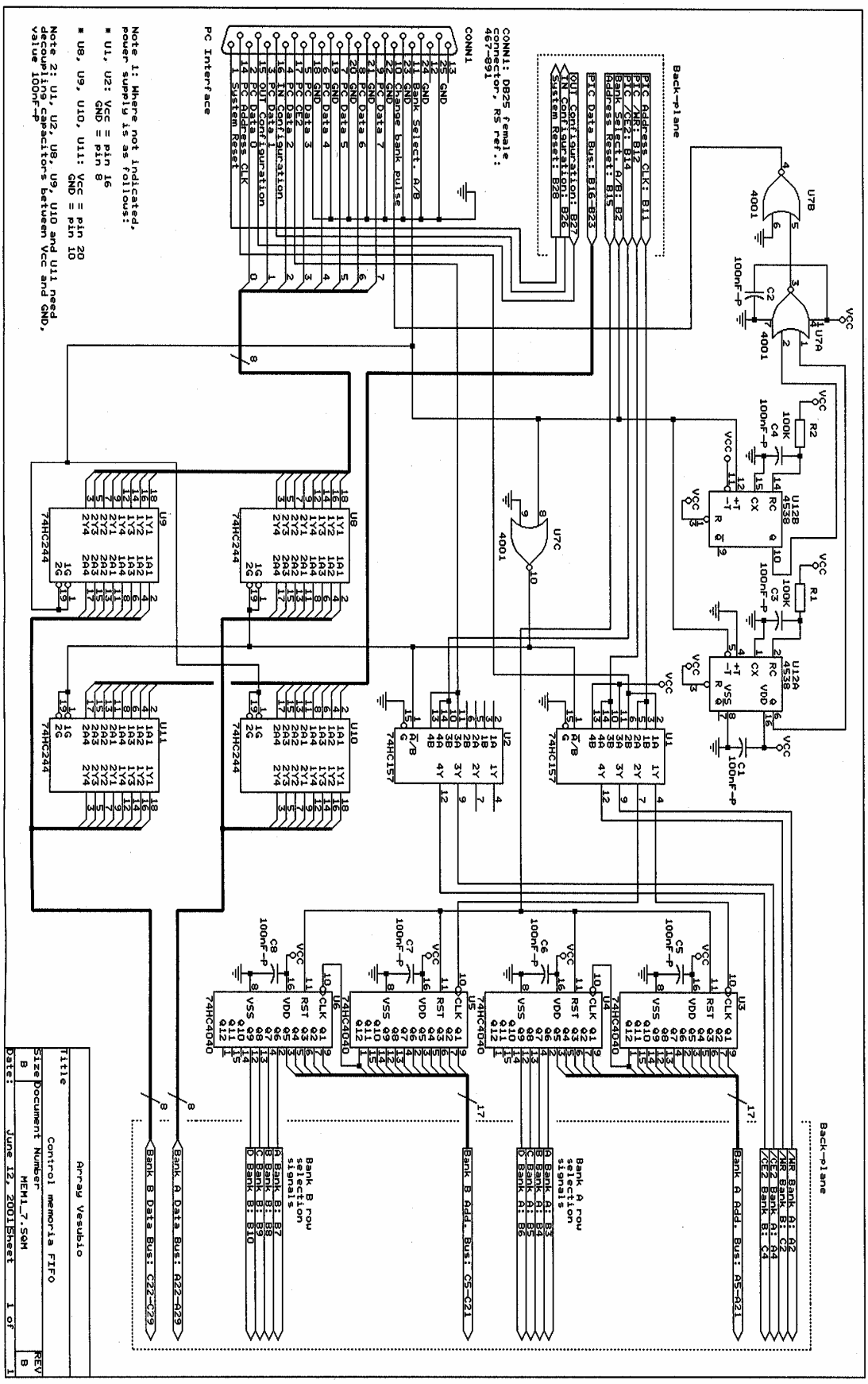


Figura 4.15. *Esquema eléctrico de la placa de control de memoria.*

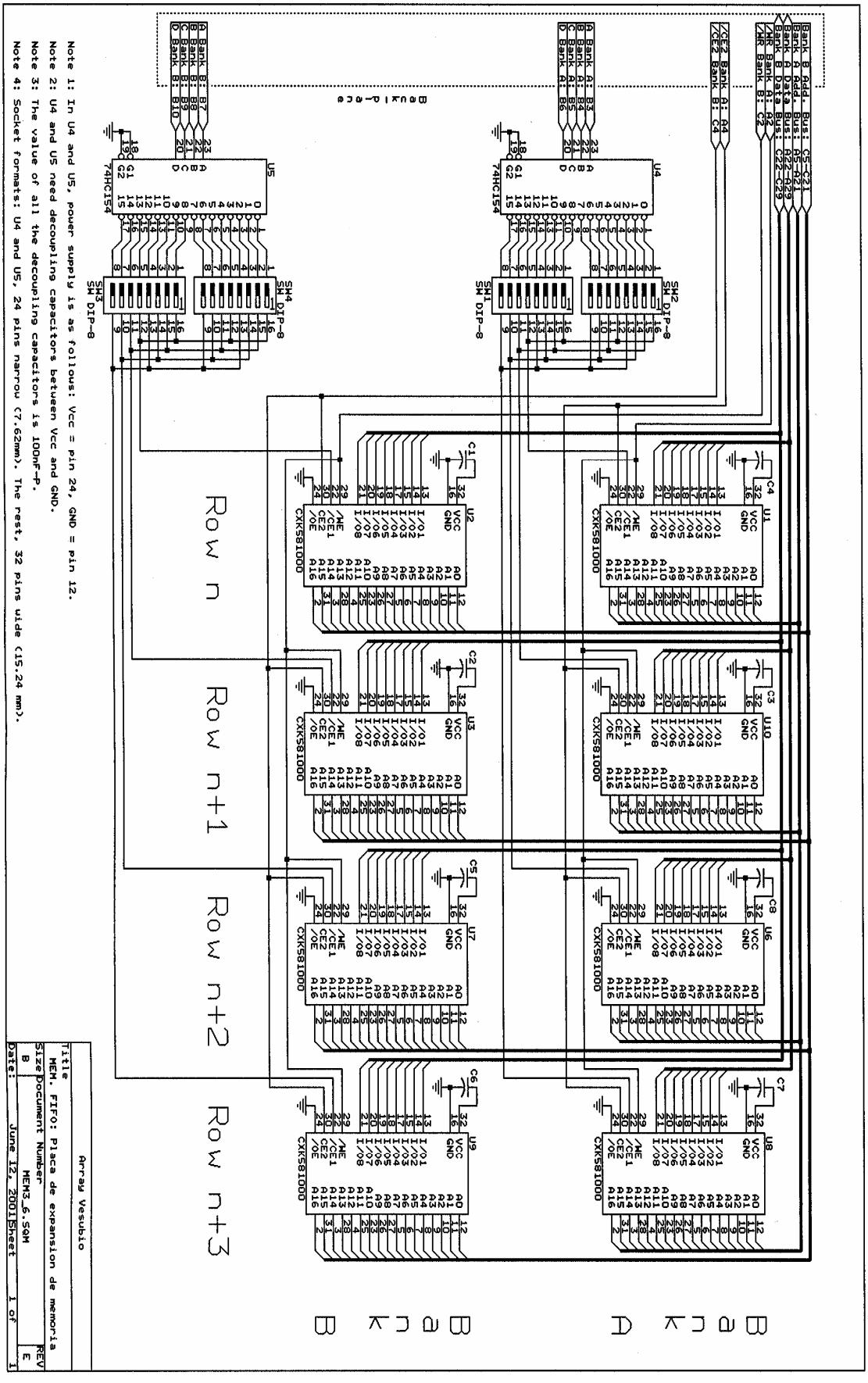


Figura 4.16. *Esquema eléctrico de las placas de expansión de memoria.*

La tarjeta de control de memoria gestiona un número variable (entre una y cuatro) de tarjetas de expansión de memoria. Cada una de estas tarjetas tiene ocho pastillas de memoria de 128 kBytes, por lo que con el número máximo de tarjetas se pueden obtener hasta 4 MBytes (2MB/banco).

El módulo de memoria permite crear un búffer de datos para el PC, que sólo necesitará realizar la lectura al detectar que el PIC ha terminado de escribir datos en un banco. La capacidad del búffer (en términos de número de segundos de datos) varía en función de los siguientes parámetros:

- Número de tarjetas de memoria utilizadas.
- Número de módulos de adquisición operativos.
- Frecuencia de muestreo utilizada en los módulos de adquisición.

La capacidad mínima del búffer es de 36 segundos/banco, que corresponde al número máximo de módulos A/D (16) operando con una frecuencia de muestreo de 100 mps y con una sola tarjeta de expansión de memoria. La capacidad máxima corresponde a un solo módulo de adquisición operando a 50 mps con cuatro tarjetas de memoria, y es de 4536 segundos.

4.4.5. Módulo de reloj

Como se ha comentado previamente, las señales de tiempo necesarias para la sincronización de todo el sistema se obtienen de un receptor GPS *Trimble Lassen SK II*, que es controlado por un PIC16C74A. El circuito de control del GPS e interfaz de datos se muestra en la figura 4.17. La comunicación con el receptor GPS se realiza mediante tres líneas de entrada/salida del PIC, implementando por *software* una interfaz serie asíncrona a 9600 bps. El intercambio de datos y señales de control con el módulo SerPar se realiza, igual que en aquél, mediante el puerto serie USART implementado en el microcontrolador, a una velocidad de 156.3 kbps.

Tanto el programa del PIC del módulo SerPar como el de los PIC de los circuitos PLL necesitan en todo momento una entrada de un pulso por segundo (PPS). El módulo de reloj es el encargado de generar esta señal, tomando como referencia los pulsos de segundo del receptor GPS. Sin embargo, el receptor GPS utilizado sólo genera estos pulsos cuando está recibiendo señal de más de tres satélites, por lo que una de las primeras tareas que debe realizar el PIC es cambiar la configuración del receptor para que genere pulsos de segundo en cualquier situación. De esta forma el módulo de reloj generará continuamente la salida de PPS durante la operación normal del sistema, y tan solo dejará de hacerlo cuando detecte una petición de reset del sistema procedente del módulo SerPar. En ese caso inhibirá la salida de PPS durante treinta segundos, lo cual será interpretado por los distintos módulos como una señal de reset (ver sección 4.5.2).

La placa del módulo de reloj implementa una interfaz RS-485, ya que tanto los PPS como los datos de entrada y de salida se transmiten usando este estándar. Esto permite colocar el módulo de reloj a distancias largas del módulo central, lo que posibilita la elección de emplazamientos resguardados para éste último sin estar condicionados a que tengan buena visibilidad GPS. A diferencia de la interfaz usada en el módulo SerPar, en este caso todos los transceptores MAX483 pueden ser configurados como transmisores o receptores, para aumentar la flexibilidad del diseño y permitir el uso del mismo circuito en otras aplicaciones.

El receptor GPS utilizado se alimenta a +5V, al igual que el resto de la electrónica. Para conseguirlos se ha implementado un circuito conmutado de fuente LM2576, que obtiene la entrada de alimentación de la batería que alimenta a todo el sistema, a través de uno de los pares trenzados del cable que conecta el módulo de reloj con el central.

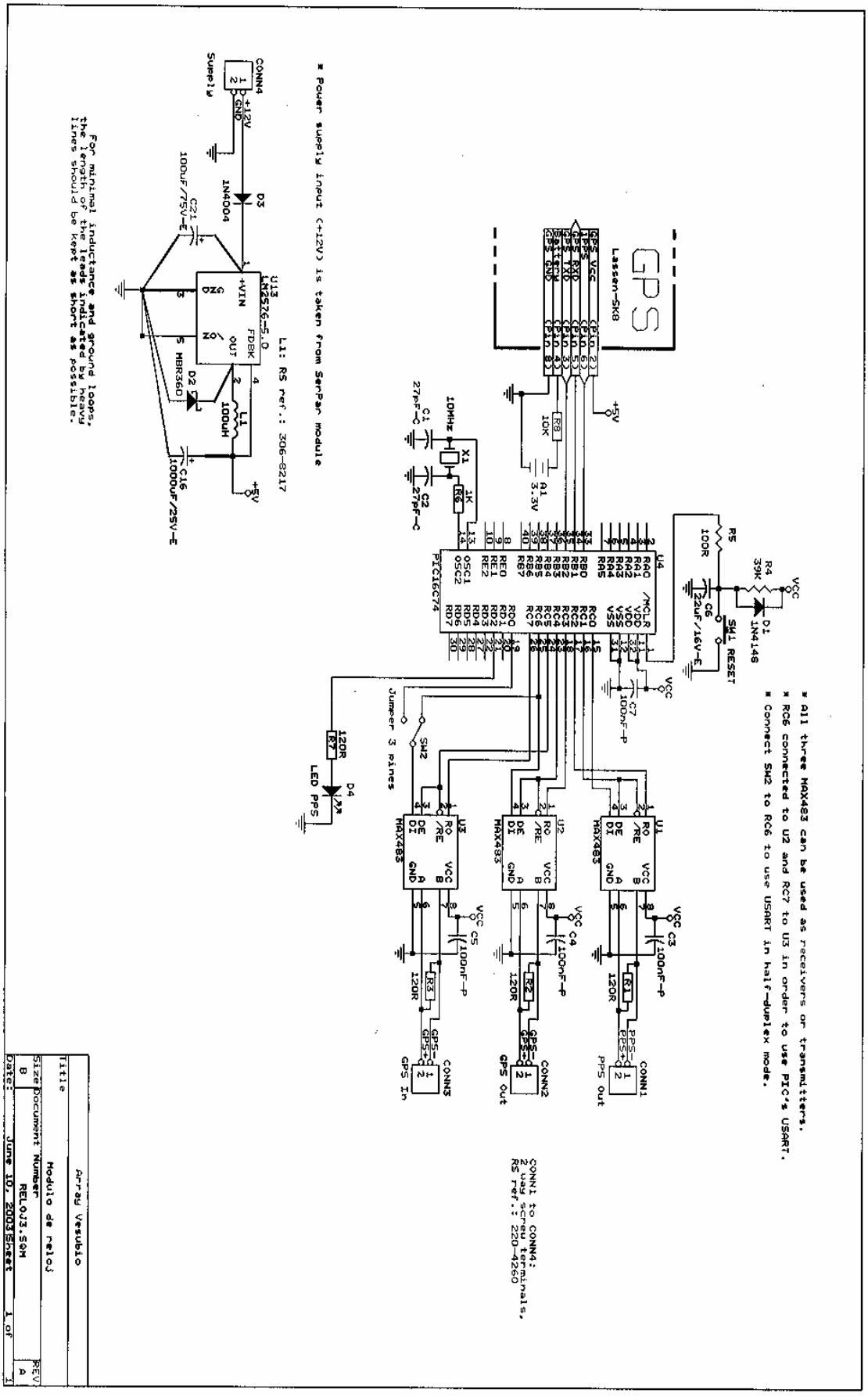


Figura 4.17. *Esquema eléctrico del módulo de reloj.*

4.4.6. Tarjeta de PC

El modelo de PC elegido es el *Lippert Cool Roadrunner II*, cuyas características ya se comentaron en el capítulo de materiales.

La tarjeta de PC se ha montado, junto con el disco duro, en una placa diseñada al efecto, que incluye una fuente de alimentación conmutada para ambos (figura 4.18). La fuente toma la tensión de entrada de la batería que alimenta todo el sistema y genera los +5V necesarios para alimentar la tarjeta de PC y el disco duro. En el esquema de la figura 4.18 se muestran además los fusibles de protección a la entrada y salida de la fuente conmutada y las conexiones del interruptor y LED de alimentación y del pulsador de STOP, que también se incluyen en la placa. El botón de STOP se utiliza para salir del programa de adquisición, aunque también podría usarse, por ejemplo, para seleccionar distintas opciones en un menú de arranque.

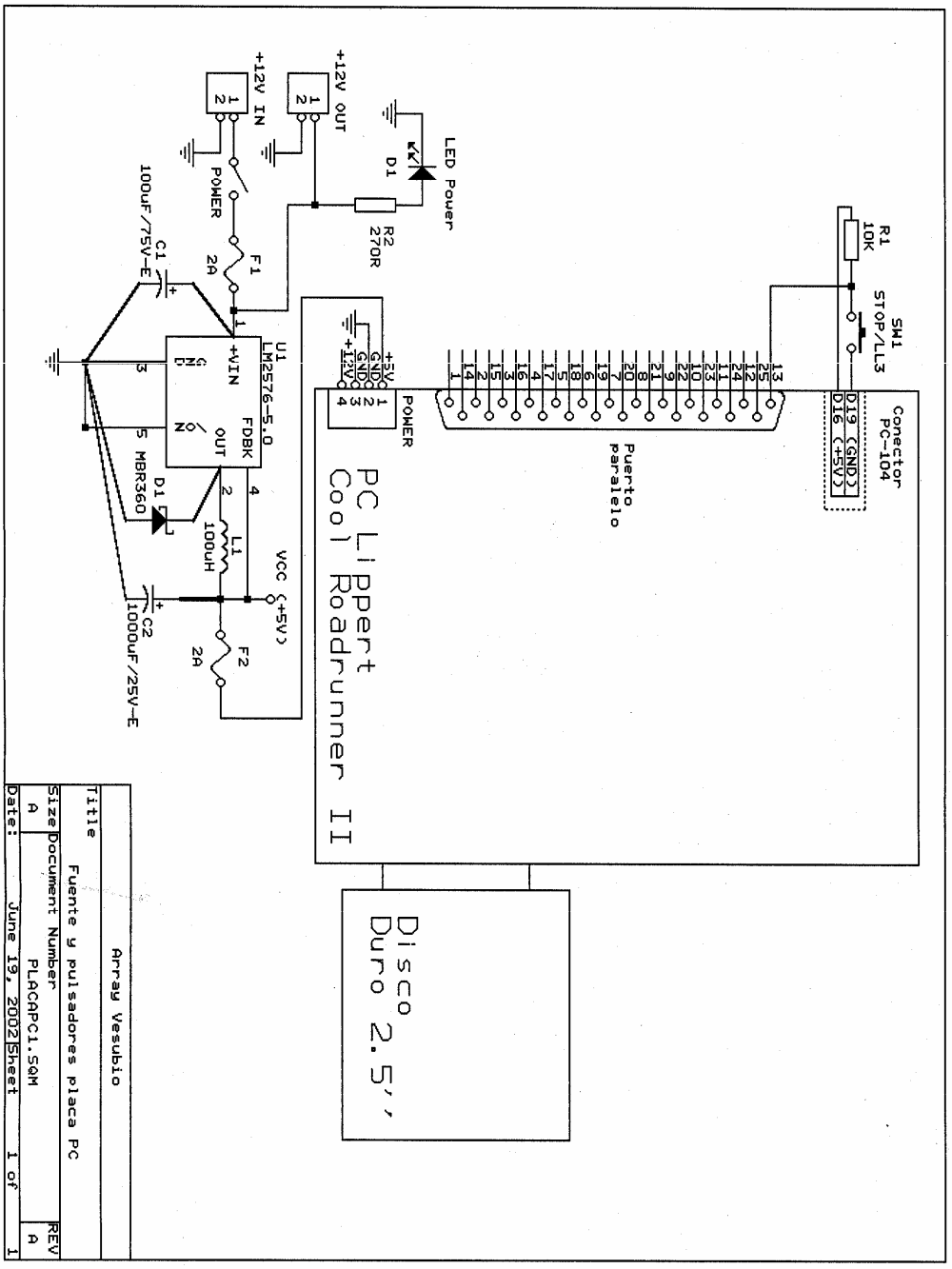


Figura 4.18. *Esquema eléctrico del circuito de fuente y conexiones para la placa de PC industrial.*

4.5. Programas

Para el funcionamiento del sistema son necesarios cinco programas distintos: uno en el PC de control y otro en cada una de las tarjetas que usan microcontroladores PIC (módulo SerPar, tarjetas de adquisición y de PLL de los módulos de adquisición y tarjeta de control del módulo de reloj). Todos los que se ejecutan en microcontroladores PIC se han programado en el lenguaje ensamblador propio de esta familia. El del PC, en lenguaje C. En la versión que se presenta aquí el del PC no es un único programa, sino varios distintos que realizan diversas funciones, si bien su adaptación como subrutinas de un programa general resultaría sencilla.

Las versiones operativas en la versión que se presenta del sistema son:

Módulo	Programa(s)	Comentarios
PC	P_INIC2.C VES2.CFG RDMEM2.CFG RDMEM2.C P_RESET.C FICHVES2.C	P_INIC2 inicializa el sistema con los parámetros del fichero de texto VES2.CFG. RDMEM2 lee el módulo de memoria y escribe los datos en ficheros con los parámetros del fichero de texto RDMEM2.CFG. P_RESET manda un comando de reinicio del sistema. FICHVES2 es el programa de demultiplexado de los ficheros de datos.
SerPar	SERPAR10.ASM	
A/D	ADQUIS6.ASM	
PLL	PLL6.ASM	
Reloj	RELOJ5.ASM	

En la próxima sección se describe brevemente el funcionamiento conjunto de los programas. La sección 4.5.2 explica el procedimiento de reset del sistema y en las secciones 4.5.3 a 4.5.7 se realiza una descripción más detallada de cada uno de los programas. Los listados completos de todos ellos pueden consultarse en el anexo II.A, incluido en el CD adjunto.

4.5.1. Funcionamiento coordinado de los programas

Cuando el sistema arranca la operación de los distintos módulos se inicia de modo escalonado. Al suministrar alimentación al sistema los distintos programas quedan en un estado de espera hasta que se le comunican los parámetros desde el módulo central:

- El programa del módulo SerPar espera la transmisión de los parámetros de operación del sistema desde el PC, según el formato descrito en el anexo II.C.1.a.

- Los módulos de adquisición esperan la transmisión de los parámetros de operación desde el módulo SerPar, según el formato que se describe en el anexo II.C.1.b.

- Los módulos PLL no empiezan a controlar la frecuencia del reloj de los conversores A/D hasta que no ha comenzado la adquisición de datos. Para ello consultan permanentemente la señal de muestreo de los conversores (señal DRDY) y no entran en la función de control del VCXO hasta que ésta no está activa.

- El programa del módulo de reloj espera a que el módulo SerPar inicie la comunicación para entrar en modo normal de operación, en el que actualiza la información de status y tiempo GPS cada segundo. Sin embargo, desde el principio y en todo momento genera una salida de pulsos de segundo, ya que estos son necesarios para el funcionamiento de varios módulos del sistema (SerPar y PLLs).

Una vez el PC ha comunicado los parámetros al módulo SerPar y éste a los módulos de adquisición, el sistema entra en funcionamiento normal: los módulos de adquisición programan los conversores con la frecuencia de muestreo y ganancia recibidas, los circuitos PLL comienzan el control de la señal de reloj y el módulo SerPar realiza el control de todo el proceso (transmisión de datos desde los módulos de adquisición, petición de información de tiempo y estado del GPS al módulo de reloj, grabación de los datos en el módulo de memoria, comprobación y gestión –en su caso- de una petición de reset del sistema desde el PC). La comunicación entre el módulo SerPar y los módulos de reloj y de adquisición se realiza de acuerdo con los formatos descritos en la sección 4.3.3. Como también se explicó anteriormente, el módulo SerPar realiza la escritura de los datos en el módulo de memoria de forma continua y alternativamente en cada uno de los dos bancos de memoria. Cuando uno de los bancos está lleno el módulo SerPar selecciona el otro y escribe en él, sin comprobar si los datos han sido leídos o no. Es, por tanto, función del programa de lectura del PC asegurarse de que los datos de cada banco han sido leídos y guardados antes de que el módulo SerPar los sobrescriba. El PC puede realizar esta tarea de dos formas:

- Por *polling* o consulta de la línea de selección de banco A/B (‘*Bank select A/B’*). Esta señal es controlada por el microcontrolador PIC del módulo SerPar. El PC no tiene control sobre ella, de modo que es el módulo SerPar quien permite el acceso del PC a uno u otro banco. Además el PIC pone todas las señales de control de las memorias del banco al que accede el PC en modo de lectura, salvo la señal CE2, de forma que el PC puede realizar la lectura simplemente conmutando el estado lógico de esta señal. El programa del PC que se presenta aquí (RDMEM2.C) realiza la lectura de esta forma.

- Por interrupciones. Cuando el PIC del módulo SerPar cambia la señal *Bank select A/B* se genera un pulso positivo en el pin 10 del puerto paralelo (señal /ACK, ver figura 4.15), que genera una llamada a la interrupción *hardware* IRQ7 si ésta está habilitada. Por tanto se puede gestionar la lectura del módulo de memoria siguiendo el procedimiento normal para la programación de interrupciones *hardware*. Hay que tener en cuenta que, por tratarse de la interrupción del puerto paralelo IRQ7, además de usar el procedimiento general es necesario habilitarla poniendo a 1 el bit 4 del tercer registro del puerto paralelo (ver, por ejemplo, Austerlitz, 1991, p. 117).

4.5.2. Reset del sistema

Existe la posibilidad de reiniciar los distintos programas del sistema, para lo cual el PC debe enviar un pulso negativo a través de la línea ‘*System reset’* (señal /STROBE del puerto paralelo, accesible a través del bit 0 del tercer registro del puerto). La generación de un pulso de estas características está implementada en el programa P_RESET.C.

Tras detectar el PIC del módulo SerPar esta señal envía un comando de reset al módulo de reloj, después de lo cual salta al inicio del programa para comenzar él mismo la secuencia de inicialización.

Cuando el PIC del módulo de reloj recibe el comando de reset del módulo SerPar deja de generar la salida de PPS durante treinta segundos. Los PICs de las placas PLL de los módulos de adquisición están programados para reiniciar la operación cuando detecten un periodo de más de diez segundos sin señal de PPS. Antes de saltar al inicio del programa el PIC de cada tarjeta PLL enviará un pulso de reset al de la correspondiente tarjeta de conversión A/D. De esta forma se consigue que todos los módulos del sistema (salvo el PC y el módulo de reloj) inicien la operación desde el principio de los respectivos programas. Esto

resulta útil, por ejemplo, en caso de que se desee cambiar los parámetros de operación del sistema sin apagarlo físicamente. Hay que recalcar que, salvo en el caso de las tarjetas A/D de los módulos de adquisición, no se trata de un reinicio *hardware*, sino simplemente un salto a las posiciones de memoria de inicio de los programas.

4.5.3 Programas del PC

Las tareas principales que realiza el PC de control del sistema son:

- Comunicación de los parámetros de operación al módulo SerPar.
- Lectura del módulo de memoria.
- Escritura de los datos en ficheros.
- Generación de un comando de reset del sistema cuando lo solicita el usuario.

La primera y la cuarta tareas sólo se realizan en momentos puntuales (al arrancar el sistema y cuando el usuario ejecuta el programa de reset, respectivamente). La única tarea crítica en el tiempo es la lectura de datos del módulo de memoria, ya que el PC debe realizarla antes de que el PIC de SerPar sobrescriba el banco correspondiente. Ni esta tarea ni la escritura de los datos en ficheros suponen una carga excesiva para el PC, por lo que el tiempo de CPU sobrante podría dedicarse a otras tareas de preprocesado de los datos, comunicación con otros sistemas de registro o gestión de un sistema de telemetría.

Para llevar a cabo las cuatro tareas se han creado tres programas distintos, que se podrían adaptar fácilmente como funciones de un único programa. Los programas son P_INIC2.C (inicialización del sistema), RDMEM2.C (lectura del módulo de memoria y escritura de los datos en ficheros) y P_RESET.C (generación de un pulso de reset).

4.5.3.1. Programa de inicialización del sistema (P_INIC2.C)

El diagrama de flujo de la figura 4.19 muestra el proceso seguido por el programa P_INIC2 para comunicar los parámetros de operación del sistema al módulo SerPar. Como puede verse, el programa lee los parámetros del fichero ASCII VES2.CFG. Estos parámetros son cuatro: frecuencia de muestreo, ganancia, configuración física de la antena y número de tarjetas de memoria usadas.

Tras la lectura de los parámetros el programa intenta establecer comunicación con el módulo SerPar, para lo cual envía pulsos negativos de 250 ms de anchura a través de la línea *InConfiguration.* Después de cada pulso consulta la línea de salida de configuración (*OutConfiguration)* para comprobar si SerPar ha generado un pulso de respuesta. En caso negativo continúa mandando pulsos hasta que obtiene respuesta, momento en el que pasa a comunicar los parámetros a SerPar usando el protocolo descrito en el anexo II.C.1.a.

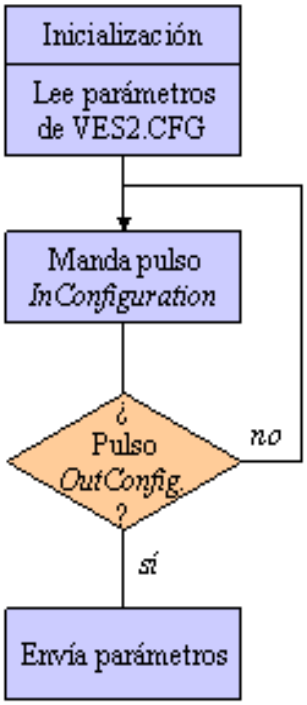


Figura 4.19. *Diagrama de flujo del programa de inicialización del sistema P_INIC2.C.*

4.5.3.2. Programa de lectura de datos y escritura en ficheros (RDMEM2.C)

La figura 4.20 muestra el diagrama de flujo del programa de lectura del búffer de datos del módulo de memoria y escritura de los datos en ficheros. Como puede verse, tras una inicialización en la que se asigna a las variables y señales de lectura los valores necesarios, el programa lee del fichero de configuración (RDMEM2.CFG) los parámetros de los ficheros de salida. Estos consisten en el tipo de fichero (archivo con todos los datos y/o sólo con cabeceras GPS) y la duración de éstos. En la sección 4.6.7 se dará una descripción más detallada de estos dos parámetros.

Antes de realizar la lectura del primer banco de memoria, el PC tiene que esperar a que el módulo SerPar haya terminado de escribir los datos en el mismo, para lo cual consulta continuamente la señal de selección de banco A/B *(Bank select A/B)*. Para un mejor aprovechamiento del tiempo, se podrían programar tareas de procesado o transmisión de datos durante este tiempo de espera, o bien realizar la lectura del módulo de memoria mediante interrupciones. Cuando el programa detecta que la señal A/B ha cambiado, lee los cuatro primeros bytes del banco de memoria recién escrito por el PIC del módulo SerPar. Estos cuatro bytes corresponden a la configuración del sistema (ver sección 4.3.4 y anexo II.C.2), y a partir de ellos el programa calcula la longitud de cada banco de memoria y el número de bytes de datos. El programa además discrimina, en función del número de tarjetas de memoria que lee de la cabecera, si el sistema está operando en modo normal o en modo TEST, lo cual determina el nombre de los nuevos ficheros. Una vez abiertos los ficheros, el programa lee el resto del banco de memoria, escribe los datos y actualiza la información de pantalla.

El programa sigue realizando el mismo proceso mientras no se solicita la salida, mediante la tecla ‘Q’ (si hay un teclado conectado al PC interno de control) o mediante el pulsador de STOP/LL3 del módulo central. Como puede verse en la figura 4.20, la única diferencia entre la lectura del primer banco y la del resto es que en el primero se abren directamente los ficheros de datos de salida. En el resto se realiza una comprobación, antes de leer el banco, de

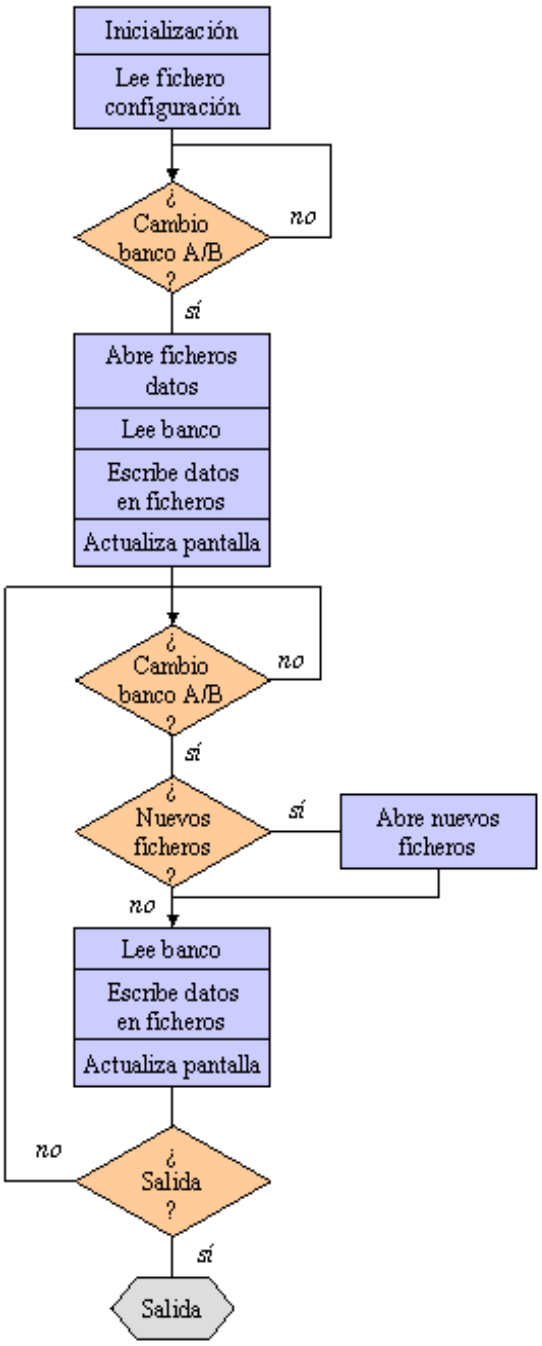


Figura 4.20. *Diagrama de flujo del programa de lectura de memoria (RDMEM2.C).*

si es necesario cerrar los ficheros de datos actuales y abrir otros nuevos. Para ello se usan criterios distintos, según el tipo de fichero de salida seleccionado a través del fichero de configuración RDMEM2.CFG.

Toda la información relevante relativa al funcionamiento del programa RDMEM2 se escribe en el fichero RDMEM2.LOG.

4.5.3.3. Programa de reset del sistema (P_RESET)

El programa P_RESET activa la señal de reset *software* del sistema (*System Reset*), para lo cual genera un pulso negativo de tres segundos de duración. Este pulso es detectado por el módulo SerPar, que transmite al módulo de reloj el comando de reset antes de saltar al inicio de su propio programa.

4.5.4. Programa del módulo serie/paralelo (SERPAR10.ASM)

El módulo serie/paralelo (SerPar) es el responsable del control central de los procesos que realizan el resto de los módulos del sistema. Las principales funciones que desempeña son las siguientes:

- Comunicación con el PC para conocer los parámetros de operación del sistema.
- Comunicación de los parámetros de operación a los módulos de adquisición.
- Comunicación continua con el módulo de reloj para conocer la información actualizada de tiempo real (status GPS y tiempo GPS).
- Control de la transmisión de datos desde los módulos de adquisición mediante la generación de pulsos de sincronismo.
- Escritura de los datos e información de tiempo en el módulo de memoria.
- Gestión de los comandos de reset del sistema.

La figura 4.21 muestra los diagramas de flujo del programa principal y de las interrupciones utilizadas en el mismo. Las fuentes de interrupción habilitadas son dos: la señal de un pulso por segundo (PPS) del módulo de reloj, que dispara la interrupción externa del PIC, y la saturación del timer 1, que se usa para temporizar los distintos procesos en el programa principal. En las rutinas de servicio de ambas interrupciones[35] se activan sendos flags definidos en el programa (*Flag PPS* en el caso de la interrupción externa y *Flag Tmr1* en la interrupción de saturación del timer 1), cuyo estado se consultará dentro del programa principal cuando sea necesario. Es decir, pese a que se utilizan rutinas de interrupción, en último término se utiliza la técnica de consulta o *polling* para conocer si éstas se han servido. En el caso que nos ocupa, por tanto, podría haberse simplificado la programación renunciando al uso de las rutinas de interrupción y consultando simplemente en el programa principal el estado de los flags de interrupción internos del PIC. Sin embargo, se ha preferido hacerlo así para mantener la estructura usada en el resto de los programas del sistema, en los que en las rutinas de servicio de las interrupciones se realizan tareas críticas en el tiempo aparte de la activación de flags.

[35] En los microcontroladores PIC de la gama media todas las interrupciones se gestionan mediante una única rutina de servicio. El primer paso dentro de la misma debe ser la consulta de los flags internos de interrupción, para conocer cuál ha sido la fuente de interrupción que ha provocado el salto a la rutina de servicio y acceder al código programado para cada caso. Técnicamente, por tanto, no existen distintas rutinas, pero se ha preferido llamarlo así en el texto por simplicidad.

a)

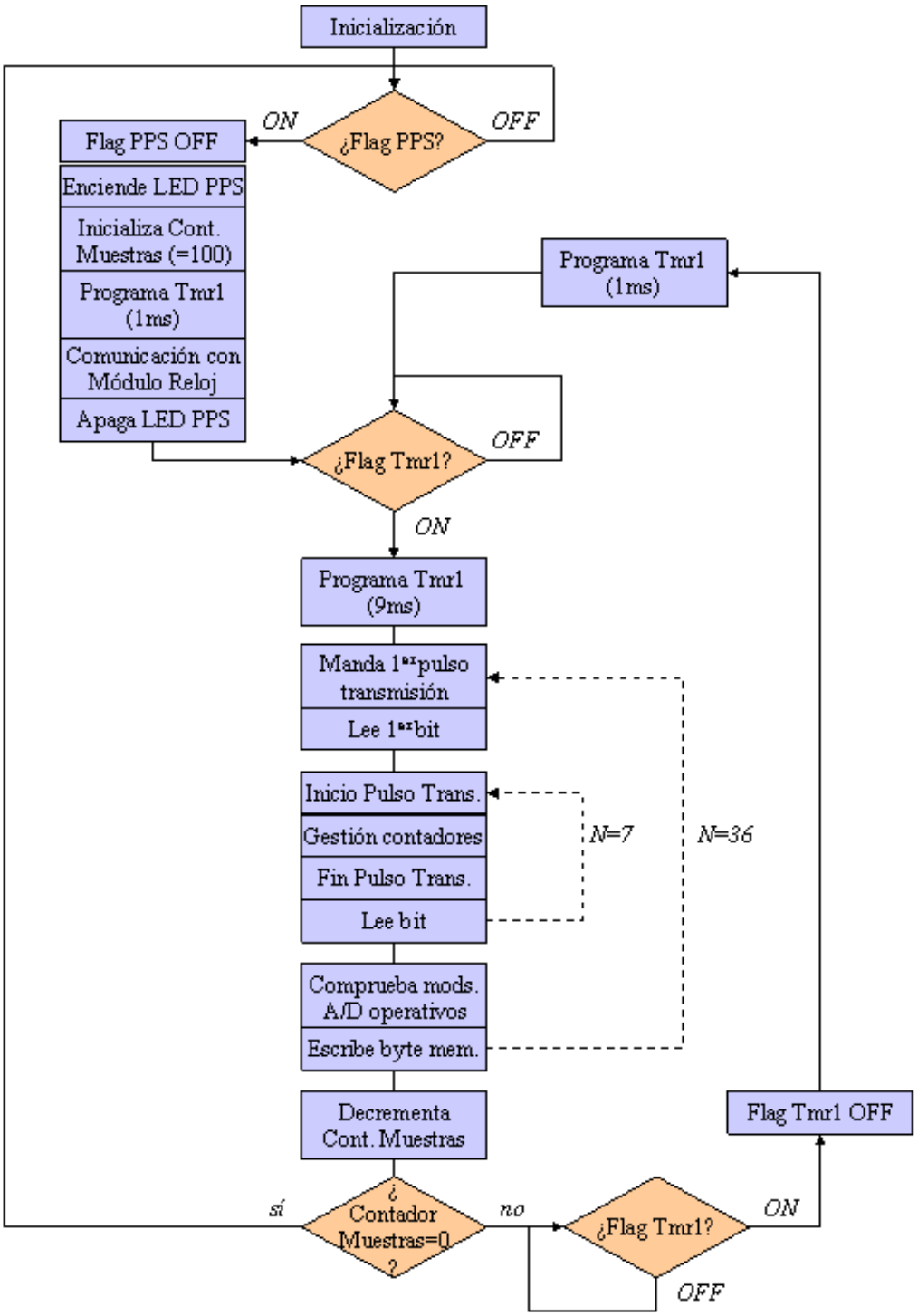
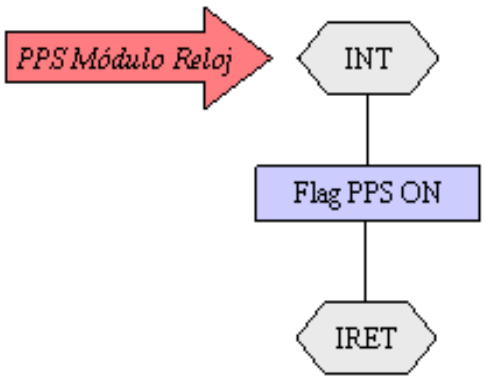
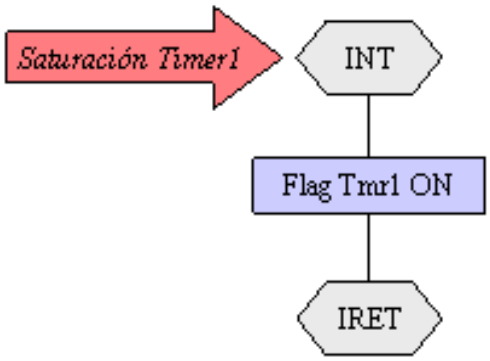


Figura 4.21. *Diagramas de flujo del programa del módulo Serie/Paralelo (SERPAR10.ASM). En esta página, el programa principal (a). En la siguiente, la rutina de servicio de la interrupción externa, disparada por los PPS del módulo de reloj (b) y la de saturación del timer 1 (c).*

4.21.b)



4.21.c)



El programa principal comienza realizando las inicializaciones normales en la programación de los microcontroladores PIC (configuración de puertos de entrada/salida, programación de interrupciones, configuración del puerto USART,...). Además, dentro de la fase de inicialización también se realiza la comunicación con el PC para recibir los parámetros de operación y la transmisión de éstos a los módulos de adquisición.

El proceso de comunicación con los módulos de adquisición y con el módulo de reloj se realiza según el formato descrito en la sección 4.3.3. Como puede verse en la figura 4.21.a, con objeto de sincronizar este proceso el programa SERPAR10 comienza esperando un PPS del módulo de reloj, para lo cual consulta continuamente el flag de PPS que se activa en la rutina de servicio de la interrupción externa. Cuando detecta que éste ha sido activado lo desactiva, enciende el LED de PPS e inicializa el contador de muestras con la frecuencia de muestreo operativa (en el diagrama de flujo se ha tomado el valor 100 por simplicidad). A continuación programa el timer 1 para que produzca una interrupción después de 1 milisegundo, que es el tiempo asignado para la comunicación con el módulo de reloj. Esta comunicación se realiza mediante el puerto serie USART del PIC. Una vez recibida la información de tiempo y status del GPS del módulo de reloj, el programa apaga el LED de PPS y pasa a esperar el fin del milisegundo programado en el timer 1. Para ello, igual que se hacía antes para la detección del PPS, consulta continuamente el flag de timer 1, que se activa en la rutina de servicio de interrupción.

Tras el primer ms, el programa pasa a controlar el proceso de comunicación con los módulos de adquisición. Para ello comienza programando el timer 1 para que se produzca una interrupción después de un tiempo igual al periodo de muestreo menos un milisegundo (nueve milisegundos para frecuencia de muestreo de 100 mps). Este es el tiempo asignado para la transmisión de todos los datos correspondientes a una muestra tomada por cada uno de los módulos de adquisición operativos. La comunicación de datos se realiza de modo síncrono. El programa SERPAR10 genera pulsos de transmisión que envía a las cuatro líneas serie. Cuando los módulos de adquisición detectan estos pulsos ponen en la línea de datos el bit correspondiente, desplazando a continuación los datos para preparar el siguiente bit (ver sección 4.3.3). En el caso del primer bit de cada byte, el programa genera el flanco ascendente inicial del pulso de transmisión, crea el retardo correspondiente y genera el pulso descendente final, tras lo cual pasa a leer el bit. En los restantes siete bits de cada byte el tiempo correspondiente al retardo entre el flanco inicial y el final del pulso de transmisión se aprovecha para realizar otras tareas, como la gestión de contadores y flags descriptivos del estado de la transmisión. La lectura de cada uno de los bits se realiza en paralelo, por las cuatro líneas de datos correspondientes a cada una de las líneas serie. De este modo, tras leer el octavo bit de cada byte, el programa SERPAR10 tendrá preparados cuatro bytes, uno por

cada línea serie, para escribir en el módulo de memoria. Antes de hacerlo, sin embargo, comprueba la máscara de módulos de adquisición operativos que le fue comunicada por el PC al principio del programa. De este modo sólo escribe en memoria los bytes que corresponden a módulos de adquisición operativos. Este proceso de lectura en paralelo se repite un total de 36 veces, tantas como bytes debe enviar cada una de las líneas serie (4 módulos de adquisición x 3 canales/módulo x 3 bytes/dato). El sistema descrito de lectura hace que los bytes de datos de cada uno de los módulos que ocupan igual posición en las respectivas líneas serie se vayan intercalando en el módulo de memoria, como se describe en el anexo II.C.2. El programa de lectura del PC (RDMEM2) escribe los datos en ficheros en el mismo orden en que los lee, de modo que antes de procesar o visualizar los datos es necesario separar los correspondientes a los distintos canales, para lo cual se utiliza el programa de demultiplexado FICHVES2 (ver sección 4.6.8).

4.5.5. Programa de las tarjetas de conversión A/D de los módulos de adquisición (ADQUIS6.ASM)

El programa de las tarjetas de conversión analógico/digital de los módulos de adquisición es el responsable del control de los conversores A/D, así como de la transmisión de los datos al módulo SerPar. Las principales funciones que realiza son:

- Comunicación inicial con el módulo SerPar para la recepción de los parámetros de operación (frecuencia de muestreo y ganancia).
- Programación de los conversores A/D con los parámetros de operación recibidos.
- Control de los conversores A/D para la lectura de datos.
- Transmisión síncrona de los datos al módulo SerPar.

La figura 4.22 muestra los diagramas de flujo de la rutina de servicio de la interrupción externa (a) y del programa principal (b).

La única fuente de interrupción habilitada en este programa es la externa, conectada a la entrada de la línea de datos Ser1, por la que se recibirá la secuencia de inicialización. Esta secuencia contiene los valores de ganancia y frecuencia de muestreo con los que deben programarse los conversores A/D y, como se describe en el anexo II.C.1.b, consiste simplemente en un número de pulsos comprendido entre 11 y 34. La rutina de servicio de la interrupción sólo se utiliza en el proceso de recepción de estos parámetros.

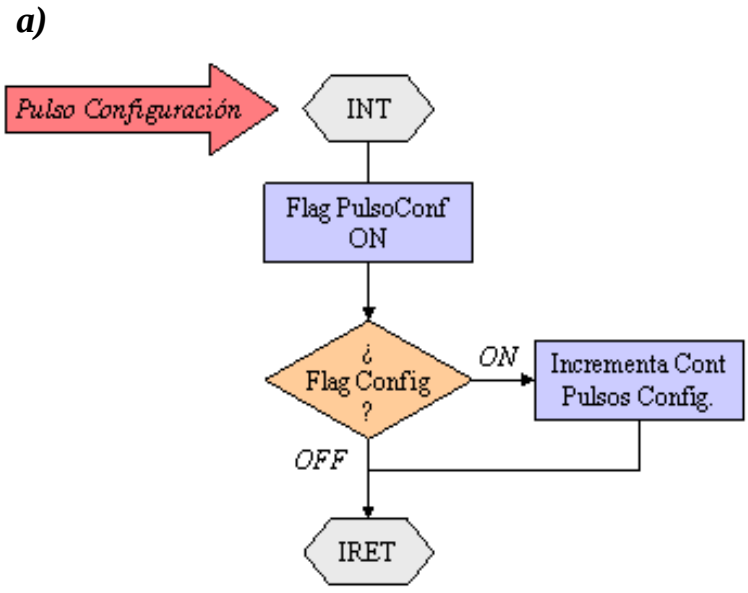
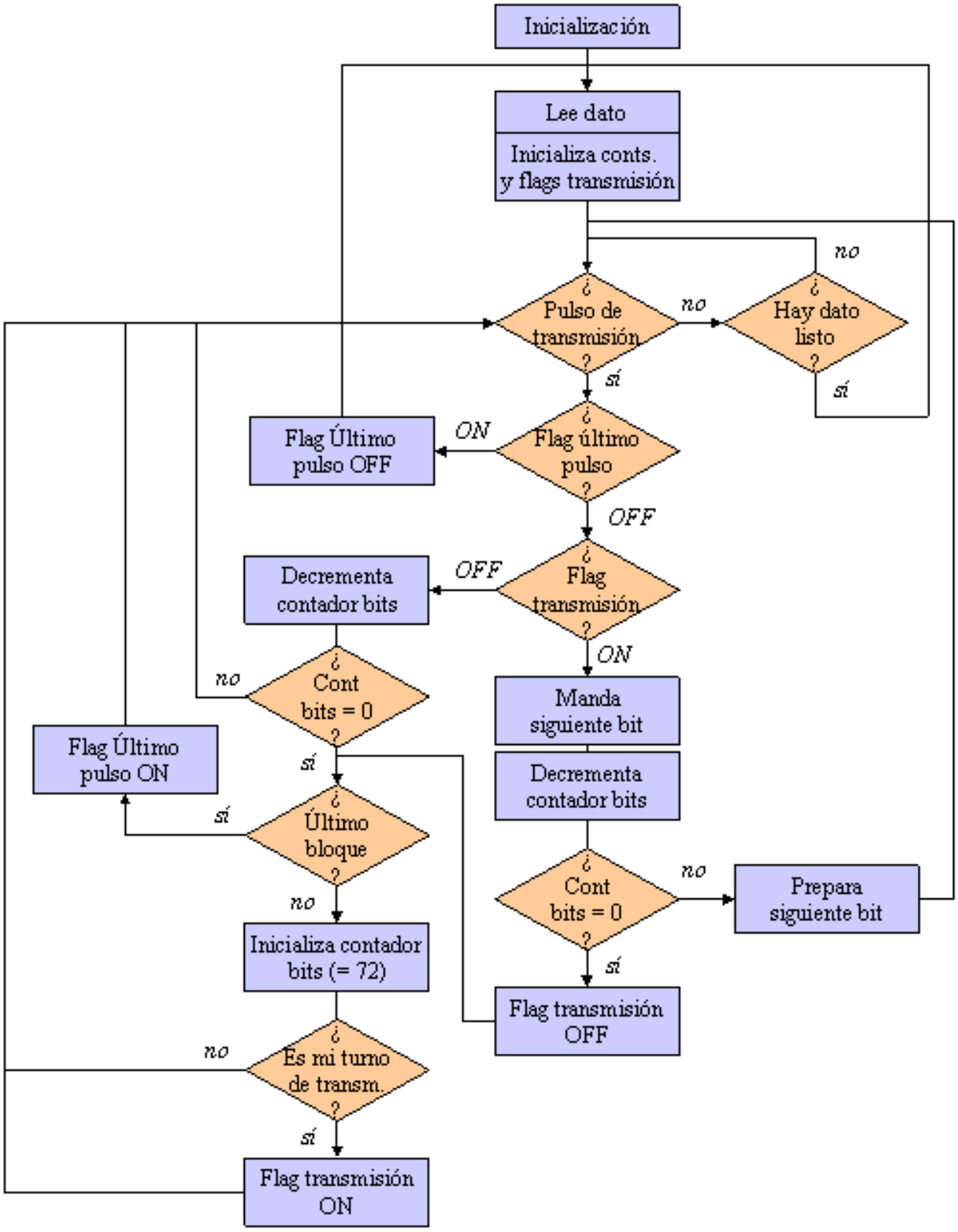


Figura 4.22. *Diagramas de flujo del programa de las placas de conversión A/D de los módulos de adquisición: rutina de servicio de la interrupción externa (a) y programa principal (b, página siguiente).*

4.22. b)


Inicialización
Lee dato
Inicializa conts. y flags transmisión
¿ Pulso de transmisión ?
no
¿ Hay dato listo ?
no
sí
¿ Flag último pulso ?
ON
Flag Último pulso OFF
OFF
¿ Flag transmisión ?
OFF
Decrementa contador bits
ON
¿ Cont bits = 0 ?
no
Manda siguiente bit
Flag Último pulso ON
sí
Decrementa contador bits
¿ Último bloque ?
sí
¿ Cont bits = 0 ?
no
Prepara siguiente bit
no
Inicializa contador bits (= 72)
sí
Flag transmisión OFF
¿ Es mi turno de transm. ?
no
sí
Flag transmisión ON

Cuando el programa del PIC arranca, queda en un estado de espera de la secuencia de configuración y activa un flag informativo de dicho estado (*Flag Config)*. Al recibirse el primer pulso de configuración se produce una interrupción, y en la rutina de servicio se incrementa un contador de pulsos, siempre que el *Flag Config* esté activado. Además, en la rutina también se activa un flag para informar al programa principal de que se ha detectado un pulso de configuración (*Flag PulsoConf*), como puede verse en la figura 4.22.a. De este modo, al recibir el primer pulso, el programa principal inicia un bucle de retardo para saber cuándo debe dar por finalizada la etapa de recepción de la secuencia de configuración, momento en el que desactiva el *Flag Config*. En función del número de pulsos detectados en ese intervalo de tiempo, conocerá los parámetros de operación, según la codificación descrita.

El programa principal comienza con las funciones de inicialización necesarias. En este caso, además de las normales en la programación de los microcontroladores PIC (configuración de los puertos de entrada/salida, programación de las interrupciones,...) hay que realizar otras tareas específicas, como son la comunicación con el módulo SerPar para la recepción de la secuencia de inicialización o la programación de los conversores A/D con los parámetros recibidos. Otra tarea que debe realizarse al principio del programa es la lectura del código de tarjeta. Cada una de las cuatro líneas serie admite un máximo de cuatro módulos de adquisición, y a efectos de programación es necesario que cada uno de los módulos conozca qué posición ocupa dentro de la línea. Esta información se introduce mediante dos microinterruptores conectados a sendos pines de entrada/salida del PIC (ver sección 4.4.2), cuyo estado se consulta al principio del programa.

Tras las tareas de inicialización, el programa pasa a la lectura de datos. Tanto las funciones de inicialización de los conversores como las de lectura de datos se han implementado siguiendo los diagramas de tiempo incluidos en las hojas de características de los conversores (ver sección 2.3.1.3). El PIC de la tarjeta de adquisición no tiene en cuenta ninguna señal de sincronismo para realizar la lectura de datos. Como se verá en la próxima sección, son las tarjetas PLL de los módulos de adquisición las responsables de asegurar que el proceso de muestreo está sincronizado en todos los puntos.

Una vez adquirida la muestra, el PIC inicializa los contadores y flags necesarios para el proceso de transmisión de los datos, que se realizará de modo síncrono según el protocolo descrito anteriormente. El programa de los módulos de adquisición llevará un control de los pulsos de transmisión recibidos y enviará, cuando calcule que es su turno de transmisión, un bit por cada pulso recibido.

Según se explicó en la sección 4.3.3, el número de pulsos de transmisión que el módulo SerPar genera y envía es de 288 (suponiendo una frecuencia de muestreo de 100mps), correspondientes a:

4 módulos/línea serie x 3 canales/módulo x 3 bytes/canal x 8bits/byte = 288

En la práctica, el módulo SerPar envía 289 pulsos. El pulso adicional se utiliza para que los programas de todas las tarjetas A/D vuelvan a la lectura de datos y para que la última que estaba transmitiendo deje la línea serie en alta impedancia.

Para llevar el control del número de pulsos, en lugar de contar la secuencia completa, el programa de las tarjetas A/D cuenta cuatro bloques de 72 pulsos. Este es el número de bits que debe enviar cada uno de los módulos de adquisición, por lo que a efectos de programación es más cómodo hacerlo así. El programa gestiona, por tanto, dos contadores distintos, uno para el bloque de pulsos y otro para el número de pulso (o de bit enviado) dentro del bloque actual (*Contador bits* en el diagrama de flujo de la figura 4.22.b). Cada vez que termina un bloque decrementa el contador de bloques, comprueba si es su turno de transmitir e inicializa el contador de bits a 72.

La secuencia de control completa puede verse en la figura 4.22.b. Tras la lectura de datos y la inicialización de contadores y flags, el programa pasa a un bucle de espera de pulso de transmisión. En este bucle se consulta continuamente la señal de entrada de los pulsos de transmisión, hasta que se ha detectado el flanco inicial ascendente y el flanco final descendente. Dentro del bucle se consulta además la señal de 'dato preparado' (*DRDY*) de los conversores A/D, saltándose a la rutina de lectura de datos si ésta se activa antes de recibir el pulso de transmisión. De esta forma en el caso de que, por cualquier causa, se reciban menos de los 289 pulsos programados, se vuelve a iniciar el proceso completo con el nuevo dato muestreado y se evita que el programa se quede en el bucle de espera de los pulsos.

Cuando se detecta el pulso de transmisión, el programa comprueba – siempre que el pulso no sea el último de la secuencia – si está en turno de transmitir, para lo cual consulta el estado de un flag (*Flag Transmisión*). En caso de que no esté activado, se limita a decrementar el contador de bits y, si éste se hace cero, comprobar si se trataba del último bloque de 72 pulsos. En caso de que *Flag Transmisión* estuviera activado, antes de decrementar el contador enviaría el siguiente bit por la línea de datos. Si el contador de bits se hace cero al decrementarlo, pasaría a comprobar si era el último bloque de pulsos, desactivando antes el *Flag Transmisión.* En caso de que no fuera el último pulso del bloque, prepararía el siguiente bit para su transmisión y volvería a la espera de un pulso de transmisión. Durante todo el tiempo que una tarjeta está transmitiendo toma el control de la línea serie. Para ello pone el transceptor RS-485 en estado de transmisión a través de una señal de control (bit 1 del puerto A del PIC, ver figura 4.12). Cuando termina de transmitir desactiva esa señal para poner el transceptor en estado de recepción y dejar la línea serie en alta impedancia.

Cada vez que termina un bloque de pulsos se comprueba si era el último. En caso afirmativo, se activaría un flag (*Flag Último pulso*) para pasar a la espera del pulso 289. Por el contrario, si no se tratara del último bloque de pulsos, se inicializaría el contador de pulsos a 72 y se comprobaría si es nuestro turno de transmisión, activando en caso afirmativo el *Flag Transmisión.* En cualquier caso, se pasaría a la espera de un nuevo pulso de transmisión. La salida del proceso de espera de pulsos y transmisión de datos se realiza cuando se detecta un pulso y el *Flag Último Pulso* se encuentra activado. En ese caso se pasa a esperar la activación de la señal DRDY y a la lectura del nuevo dato.

4.5.6. Programa de las tarjetas PLL de los módulos de adquisición (PLL6.ASM)

Las tarjetas PLL tienen como principal tarea la de sincronizar el proceso de adquisición de datos en las tarjetas de conversión A/D de los distintos módulos de adquisición. Como se discutió en el capítulo 3, la técnica de muestreo empleada por los conversores AD7710 no permite sincronizarlos mediante una señal externa (ver nota 12, referente a la señal /SYNC). Por tanto, para conseguir que varios conversores operen sincronizadamente una de las soluciones es modificar la frecuencia de la señal de reloj maestro con la que trabajan, lo cual produce un cambio en su velocidad de operación y con ello una variación de la frecuencia de muestreo real. Con el fin de tener control sobre la señal de reloj maestro, en lugar de utilizar en cada tarjeta A/D un cristal para generar esta señal se emplea un oscilador a cristal controlado por tensión (VCXO), cuya frecuencia de salida varía en torno al valor central de 9.8304MHz. La modificación de la frecuencia del VCXO para sincronizar el proceso de muestreo es realizada por los PICs de los circuitos PLL. Para conseguir esta sincronización es necesario utilizar como base una referencia de tiempo precisa y común en todos los módulos de adquisición, que en nuestro caso es la señal de un pulso por segundo (PPS) generada por el módulo de reloj a partir de los pulsos de segundo del receptor GPS. El cálculo de las variaciones necesarias en la frecuencia del VCXO se realiza midiendo el tiempo transcurrido

entre cada PPS y el primer pulso DRDY de la placa de conversión A/D[36]. El PIC del circuito PLL trata de mantener este tiempo fijo e igual a 200μs (ver figura 4.23), para lo cual si éste es mayor actuará sobre el VCXO para incrementar su frecuencia de salida, y de esta forma aumentar ligeramente la frecuencia de muestreo de los conversores A/D. Del mismo modo, si el tiempo transcurrido entre un PPS y la primera señal DRDY de ese segundo es menor de 200μs, el PIC actúa sobre el VCXO para disminuir la frecuencia de salida y ralentizar así el proceso de muestreo en los conversores.

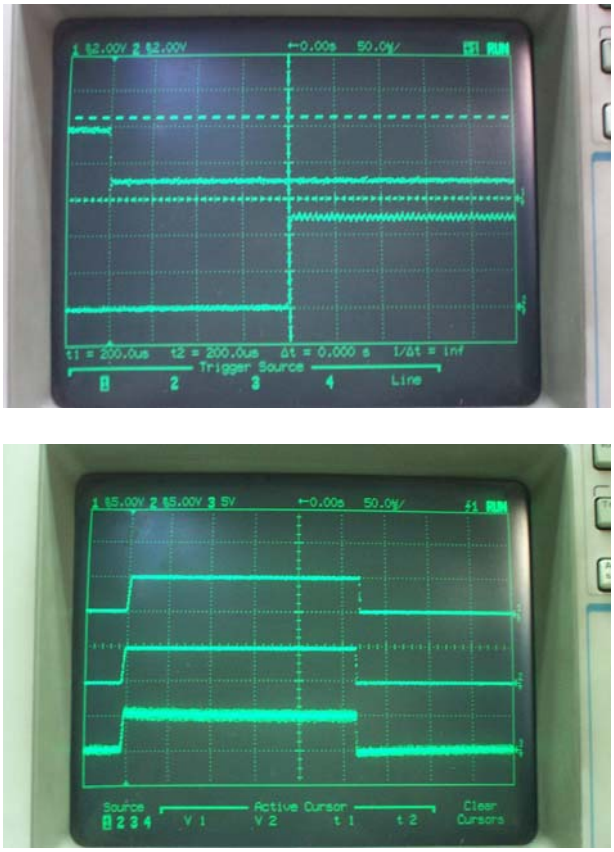


Figura 4.23. *Imágenes de osciloscopio de las señales de sincronización de la adquisición. En la imagen superior, diferencia de tiempo entre el flanco inicial descendente de la señal PPS (canal 1) y la de adquisición de datos (DRDY, canal 2) en uno de los módulos de adquisición. En la imagen inferior, señales DRDY de tres módulos de adquisición distintos. La imagen da idea de la sincronización conseguida en el proceso de muestreo mediante el uso de circuitos PLL.*
(Escala de tiempos: 50μs/div en ambas imágenes).

Según esto, el diagrama de flujo del programa de las tarjetas PLL podría ser el que se muestra en la figura 4.24.a. La gestión del VCXO se realiza mediante una de las interrupciones del PIC, que se dispara con el flanco descendente inicial de la señal de PPS. Como puede verse, la rutina inicializaría un timer y esperaría a que la señal de dato listo de los conversores A/D (DRDY) se activase. En ese momento detendría el timer y lo leería, comparando el valor leído con el correcto (200μs). Si el valor leído resultara mayor que el correcto sería señal de que los conversores A/D operan demasiado lentos, por lo que se actuaría sobre el VCXO para aumentar la frecuencia de salida, y viceversa.

[36] Como se describió en el capítulo 2, durante la operación normal de los conversores A/D, la señal DRDY se activa cuando hay un nuevo dato listo para ser leído y se desactiva cuando se ha completado la lectura. Por tanto, es esta señal la que se usa en los programas como referencia de que los conversores han completado el proceso de muestreo, y el objetivo del sistema de sincronización será conseguir que las señales DRDY de todos los módulos de adquisición se activen simultáneamente.

a)

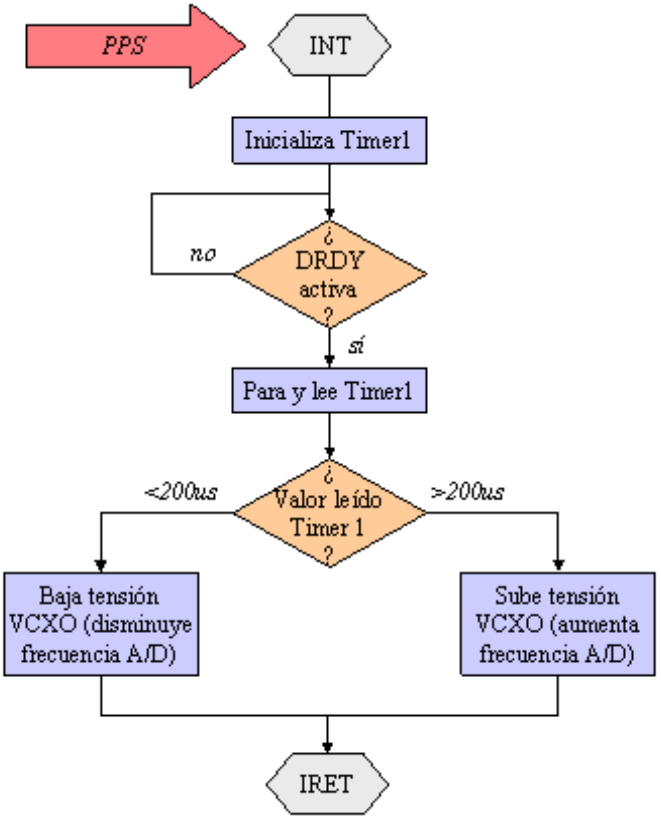


Figura 4.24. *Diagramas de flujo del programa de los circuitos PLL de los módulos de adquisición. En a), diagrama simplificado de la interrupción de PPS. En b) y c) (páginas siguientes) se muestra el diagrama completo de esta interrupción y el del programa principal, respectivamente.*

En la práctica, sin embargo, la implementación en el programa del control del VCXO es algo más complicada debido a varias razones:

- Cuando la situación se aproxima a la ideal (señal DRDY variando muy lentamente en torno a los 200μs desde el PPS) las medidas de tiempo varían muy poco entre un segundo y el siguiente, y las diferencias pueden ser menores que la precisión que es posible alcanzar con el PIC. Por tanto, la lectura del timer y el control del VCXO no se realizan cada segundo, sino en intervalos de diez segundos, con objeto de minimizar el error en la medida.
- El control del VCXO se realiza a través de una salida del PIC codificada mediante anchura modulada de pulsos (PWM), que se pasa luego por un filtro paso baja (ver figura 4.13). El programa debe, por tanto, inicializar y controlar el módulo correspondiente de salida PWM del PIC.
- La variación de la frecuencia del VCXO debe ser suave, para evitar cambios bruscos en la frecuencia de muestreo e inestabilidades en el lazo de realimentación del PLL. Para conseguirlo el PIC no compara simplemente la diferencia de tiempo PPS-DRDY medida con la correcta (200μs), sino que también la compara con la medida anterior y establece si el sistema tiende al equilibrio. En caso de que sea así, no actúa sobre el VCXO.
- Teniendo en cuenta el formato de la comunicación de datos entre el módulo SerPar y los módulos de adquisición (ver sección 4.3.3), es necesario garantizar que el proceso de lectura de un dato no se va a producir fuera del primer milisegundo de cada segundo. El PIC del circuito PLL debe vigilar que esto no ocurra y anticiparse a ello, para lo cual establece unos valores mínimo y máximo permitidos para la diferencia PPS-DRDY (20 y 500μs, respectivamente). Cuando alguno de ellos se supera envía un pulso de sincronismo

a los conversores A/D, que sincroniza los tres conversores de la placa y centra de nuevo la señal DRDY a unos 200μs del PPS.

- Deben gestionarse LEDs testigos de la operación del conjunto PLL + placa de adquisición A/D. Uno de ellos monitorizará los PPS, el otro los pulsos de sincronismo enviados a la placa de conversores. Éste último puede dar una idea al usuario de si un módulo de adquisición está funcionando bien, ya que si se enciende demasiado a menudo es señal de que el PLL no es capaz de mantener la primera señal DRDY dentro del primer ms únicamente actuando sobre el VCXO, sino que necesita enviar continuamente pulsos de sincronismo a los conversores A/D.

La figura 4.24.b muestra un diagrama de flujo más completo en el que se incorporan estas tareas. Como puede verse, la rutina de servicio de la interrupción comienza encendiendo el LED de PPS y activando el Flag de PPS recibido, que se utilizará para la gestión de los comandos de reset en el programa principal. A continuación comprueba si el *Flag HayDRDY* está activado, y sólo continúa la rutina de la interrupción en caso afirmativo. Este flag se activa al principio del programa principal, cuando se detecta que la placa de conversores A/D ha comenzado a operar normalmente.

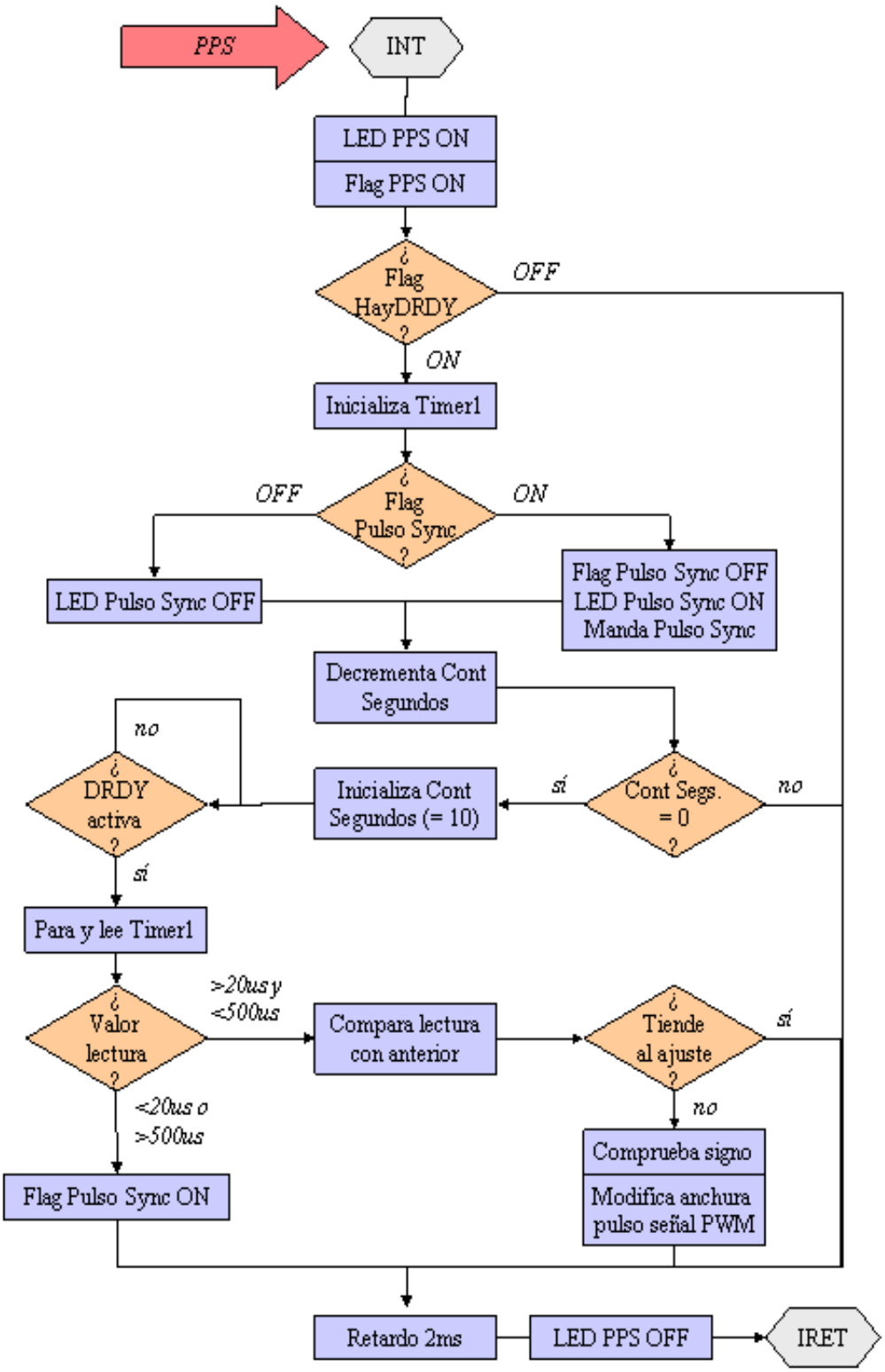
El siguiente paso es la programación del timer 1, que se configura de modo que el registro principal del timer se incrementa con cada ciclo de instrucción del PIC (modo *timer*). Al introducir los valores iniciales en este registro hay que tener en cuenta, por tanto, el número de ciclos de instrucción perdidos desde que se recibe el PPS hasta que se ha llegado a este punto de la rutina de servicio.

Antes de realizar el control de la frecuencia de salida del VCXO, el programa comprueba el estado del *Flag Pulso Sync*. En caso de que esté activado envía un pulso de sincronismo a la placa de conversores y enciende el LED correspondiente. Este pulso hace que los tres conversores de la placa sincronicen su operación. Empíricamente puede verse que cuando se realiza una sincronización siguiendo esta secuencia, la primera señal DRDY del siguiente segundo se produce a unos 200μs del flanco inicial del PPS. Esta es la razón de haber elegido este valor como referencia para el ajuste del VCXO.

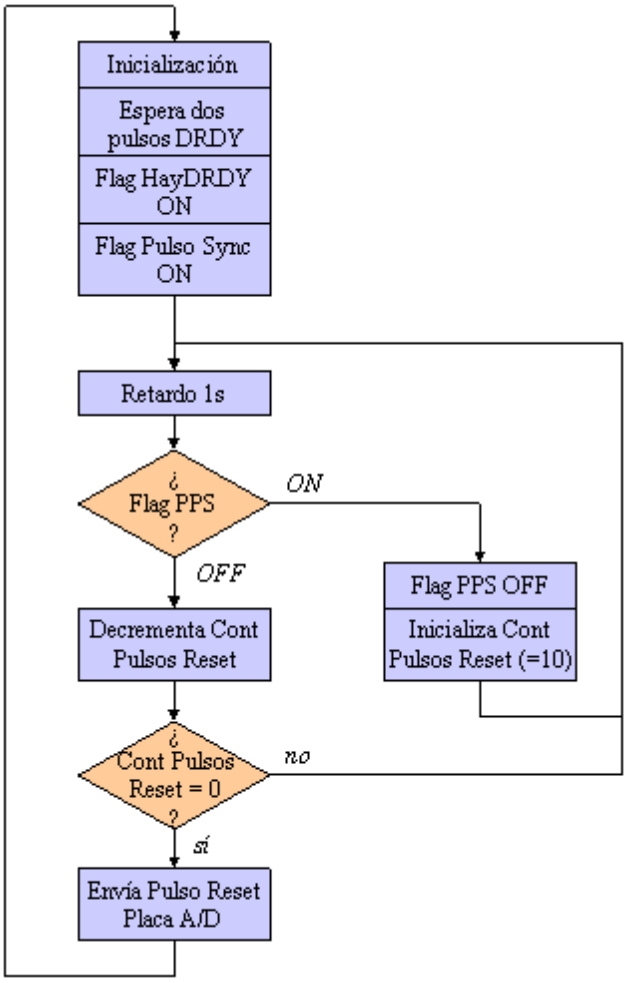
Como ya se ha comentado antes, el control del VCXO se realiza en intervalos de diez segundos, para lo cual el programa gestiona un contador que se decrementa cada vez que se sirve la interrupción de PPS. Cuando el contador llega a cero se vuelve a inicializar a diez y se entra en las funciones de control del VCXO. El primer paso en las mismas es esperar hasta que se activa la señal DRDY de los conversores, tras lo cual se detiene el timer 1 y se lee su registro. Si el valor leído es menor de 20μs o mayor de 500μs es necesario resincronizar los conversores A/D, para evitar que se llegue a la situación crítica en la que el muestreo queda fuera del intervalo de 1 ms que tiene asignado. En ese caso se activaría el *Flag Pulso Sync*, para que el pulso de sincronismo se mande a la placa de conversión A/D sincronizado con el siguiente PPS y para encender el LED correspondiente.

Si el valor leído del timer 1 estaba dentro de los límites permitidos, se compara con la lectura anterior. De este modo se puede ver si la tensión de entrada actual en el VCXO es suficiente para realizar un ajuste de la frecuencia de muestreo de los conversores o es necesario modificarla más. Si de esta comparación se deduce que el sistema tiende al ajuste (es decir, que la señal DRDY se está desplazando progresivamente hacia el valor central de 200μs desde el PPS) no se realiza ninguna modificación en la tensión de entrada del VCXO. Si, por el contrario, DRDY tiende a separarse del valor central, se modificará la tensión de entrada del VCXO para compensar esta tendencia. Este ajuste se realiza aumentando o disminuyendo, en función del signo de la diferencia entre el valor leído del timer 1 y el correcto (200μs), la anchura del pulso de la salida PWM.

4.24.b)


PPS
INT
LED PPS ON
Flag PPS ON
¿ Flag HayDRDY ?
OFF
ON
Inicializa Timer1
¿ Flag Pulso Sync ?
OFF
ON
LED Pulso Sync OFF
Flag Pulso Sync OFF
LED Pulso Sync ON
Manda Pulso Sync
Decrementa Cont Segundos
no
¿ DRDY activa ?
Inicializa Cont Segundos (= 10)
sí
¿ Cont Segs. = 0 ?
no
sí
Para y lee Timer1
¿ Valor lectura ?
>20us y <500us
Compara lectura con anterior
¿ Tiende al ajuste ?
sí
<20us o >500us
no
Comprueba signo
Modifica anchura pulso señal PWM
Flag Pulso Sync ON
Retardo 2ms
LED PPS OFF
IRET

4.24. c)



Toda la gestión de control de la frecuencia del VCXO se realiza en la rutina de tratamiento de la interrupción. El programa principal, en este caso, se reduce a la gestión del comando de reset del sistema. Como se verá en la descripción del programa del módulo de reloj, éste genera PPS de salida en cualquier situación, salvo cuando recibe un comando de reset desde SerPar. En ese caso inhibe la salida de PPS durante treinta segundos, de forma que si el programa de las tarjetas PLL permanece más de diez segundos sin haber detectado ningún PPS interpreta que se debe a un comando de reset del sistema. Como puede verse en la figura 4.24.c, para realizar esta comprobación el programa principal genera retardos, mediante bucles de instrucciones NOP, de un segundo de duración. Si después de cada retardo el flag de PPS (que se activa en la interrupción de PPS) no está activado, decrementa un contador. Si éste llega a cero interpreta que se trata de un comando de reset, por lo que manda un pulso de reset al PIC de la placa de conversores A/D y salta al inicio del propio programa.

Las funciones de inicialización previas al bucle principal consisten en la configuración de los puertos de entrada/salida y de las interrupciones, así como de los módulos CCP (Captura/Comparación/PWM). Uno de estos módulos se configura en modo PWM, para realizar el control del VCXO como se ha explicado previamente. El otro se configura en modo captura, para poder gestionar la interrupción producida por el PPS como una interrupción externa.

Como se muestra en la figura 4.24.c, el programa no entra en el bucle principal hasta que no ha detectado dos pulsos DRDY, señal de que el programa de la placa de conversión A/D ha recibido la secuencia de inicialización del módulo SerPar, ha programado los conversores A/D con los parámetros recibidos y ha comenzado a operar normalmente. Cuando detecta los pulsos DRDY activa el *Flag HayDRDY*, para que la rutina de interrupción comience el control del VCXO, y el *Flag Pulso Sync*. Esto hará que en la siguiente interrupción de PPS se genere un pulso de sincronismo inicial de los conversores y el control del VCXO se comience en una situación lo más próxima posible a la del equilibrio (distancia PPS-DRDY igual a 200μs).

4.5.7. Programa del módulo de reloj (RELOJ5.ASM)

El módulo de reloj es el encargado de generar la señal de PPS necesaria para el funcionamiento de todo el sistema, para lo cual controla un receptor GPS *Trimble Lassen SK II*. También utiliza el receptor GPS para actualizar la información de tiempo real y comunicársela al módulo SerPar. Las principales tareas que debe realizar son, por tanto:

- Inicialización y control del receptor GPS.
- Generación de la señal de salida de PPS tomando como entrada los pulsos de segundo del GPS.
- Comunicación de la información de tiempo real y status del GPS al módulo SerPar.
- Inhibición de la salida de PPS durante treinta segundos en caso de recibir una petición de reset del sistema desde el módulo SerPar.

La figura 4.25 muestra varios diagramas de flujo del programa RELOJ5.ASM. Como puede verse, se han habilitado tres interrupciones distintas (figuras 4.25.b, c y d) para gestionar tareas específicas, mientras que el programa principal (4.25.a) se ocupa de otras más genéricas.

La primera tarea que se realiza en el programa principal es la inicialización, tanto de las líneas de entrada/salida como de las interrupciones, el receptor GPS y los parámetros del puerto serie USART.

Las tres interrupciones habilitadas en el programa son la externa, disparada por la señal de un pulso por segundo del receptor GPS, la de recepción de datos en el puerto USART, que se usará para detectar cuándo quiere establecer comunicación el módulo SerPar, y la del timer 1, que se utilizará para determinar la longitud de los pulsos PPS de salida.

La inicialización del receptor GPS se realiza mediante los comandos 25h y 35h del protocolo TSIP (Trimble Navigation Limited, 1999), que realizan un reset *software* y establecen los valores de las opciones de entrada/salida, respectivamente. En el caso del comando 35h es importante configurar el receptor para que genere pulsos de segundo en cualquier situación, ya que este modelo está configurado por defecto para hacerlo sólo cuando recibe señal de más de tres satélites. En nuestro caso es fundamental la presencia continua de PPS, ya que los programas del módulo SerPar y de los PLLs basan su operación en esta señal. Para conseguir que el receptor genere pulsos de segundo en cualquier situación es necesario poner a uno el bit 5 del segundo byte de datos del comando 35h.

En cuanto a la inicialización de los parámetros del puerto serie USART, éstos son los mismos que en el caso del módulo SerPar: 8 bits de datos, uno de inicio, uno de parada, sin bit de paridad y velocidad de transmisión de 156.3 kbps.

Después de la inicialización se entra en el bucle principal del programa. Lo primero que se realiza en el mismo es una sincronización con el programa del módulo SerPar, para lo cual se espera el siguiente pulso de segundo del GPS. Esta espera se realiza consultando continuamente el estado del *Flag PPS*, que es activado en la rutina de servicio de la interrupción externa. Cuando se detecta que este flag se ha activado, se vuelve a desactivar y se continúa el programa.

Durante el funcionamiento normal, el programa espera a que el módulo SerPar solicite el envío de datos, para lo cual consulta el *Flag SerPar espera datos* hasta que la interrupción de carácter recibido lo active. Una vez activado, comprueba si ha llegado un nuevo PPS durante el tiempo de espera. Si esto ha sucedido quiere decir que los datos de tiempo y status GPS no están actualizados, por lo que se desactiva el *Flag DataOK*. A continuación, y en función del estado de este flag, el programa envía a SerPar los datos de status y tiempo GPS o una secuencia de doce caracteres iguales (31h = ASCII ‘1’) que codifican el error. Aunque, por simplicidad, no se ha incluido en el diagrama de flujo, hay otra situación en la que no se

envía la información real de tiempo y status, sino otra secuencia de doce caracteres iguales. Corresponde a la primera transmisión después del arranque, y la secuencia en este caso es de doce caracteres 30h (ASCII '0'). La transmisión de esta cadena se realiza con el objeto de que el módulo SerPar sea capaz de detectar un rearranque del módulo de reloj, ya sea por fallo de alimentación o porque ha actuado el *watchdog.* El resto de secuencias especiales que pueden aparecer en lugar de la información de status y tiempo GPS no corresponden a tramas enviadas por el módulo de reloj, sino a errores de transmisión en el módulo SerPar (ver sección 4.3.4 y anexo II.C.2).

Después de transmitir la información de status y tiempo GPS (o, en su caso, la secuencia alternativa) el PIC del módulo de reloj realiza una petición de status y de tiempo al receptor GPS. La comunicación con el GPS se realiza mediante una interfaz serie asíncrona a 9600 bps implementada por *software.*

Como se comentó al describir el formato de la transmisión de datos (sección 4.3.3), la comunicación entre SerPar y el módulo de reloj se realiza durante el primer ms de cada segundo. Para evitar que el programa del módulo de reloj desatienda esta tarea, en las funciones de comunicación con el GPS se realiza una comprobación continua del flag de PPS. Si se activa, señal de que un pulso de segundo ha llegado antes que la respuesta del GPS, el programa salta a gestionar la comunicación con SerPar, poniendo antes el *Flag DataOK* a OFF para transmitir el error.

La interrupción externa se dispara con la señal de pulso de segundo del GPS. Como puede verse en el diagrama de flujo de la figura 4.25.b, lo primero que se realiza en la rutina es el refresco del *watchdog timer.* Se ha elegido esta rutina para hacerlo en lugar del programa principal porque la generación de la salida de PPS, que se realiza aquí, es la función más crítica del módulo de reloj, puesto que sin esta señal los otros módulos del sistema no podrían operar.

El resto de las tareas desarrolladas en esta rutina depende del estado del flag de reset. Como se verá posteriormente, este flag se activa en la interrupción de carácter recibido cuando SerPar envía un comando de reset del sistema. La respuesta del módulo de reloj a un carácter de reset es inhibir la salida de PPS durante treinta segundos. Así pues, si el flag de reset está activado, la rutina de interrupción externa se limita a decrementar el contador de pulsos de reset y a deshabilitar el flag de reset cuando el contador se hace cero.

En el caso más normal de que el flag de reset no se encuentre activo, la función de la rutina es gestionar la salida de PPS. Para ello genera el flanco inicial descendente del PPS de salida (ya que los pulsos de salida son negativos) y enciende el LED de PPS. La generación del flanco inicial es la primera tarea que se realiza, para que el PPS de salida tenga el menor retraso posible respecto al pulso de segundo del GPS. Teniendo en cuenta el tiempo que se invierte en el salto al vector de interrupciones en el PIC, así como la llamada a la función de gestión de la interrupción externa, este retraso es de diez ciclos de instrucción, que en nuestro caso, en que el PIC opera con cristal de 10 MHz, corresponden a 4μs.

Después de generar el flanco inicial del PPS de salida y de encender el LED de PPS, la rutina de gestión de la interrupción programa el timer 1 del PIC, que se utilizará para determinar cuándo termina el PPS de salida. La anchura programada para el PPS de salida depende del status del GPS: cuando la última información de status recibida corresponde a una recepción de señal de los satélites óptima (cabecera 00h 00h según el protocolo TSIP) el pulso tendrá una anchura de 10 ms. En caso de que el status sea cualquier otro, el PPS de salida se generará con una anchura de 20 ms. De esta forma los distintos módulos del sistema pueden conocer fácilmente el estado del receptor. Esto también resulta útil para pruebas en laboratorio, ya que permite estimar de una forma rápida la calidad de la señal GPS cuando se monitorizan los PPS con un osciloscopio.

a)

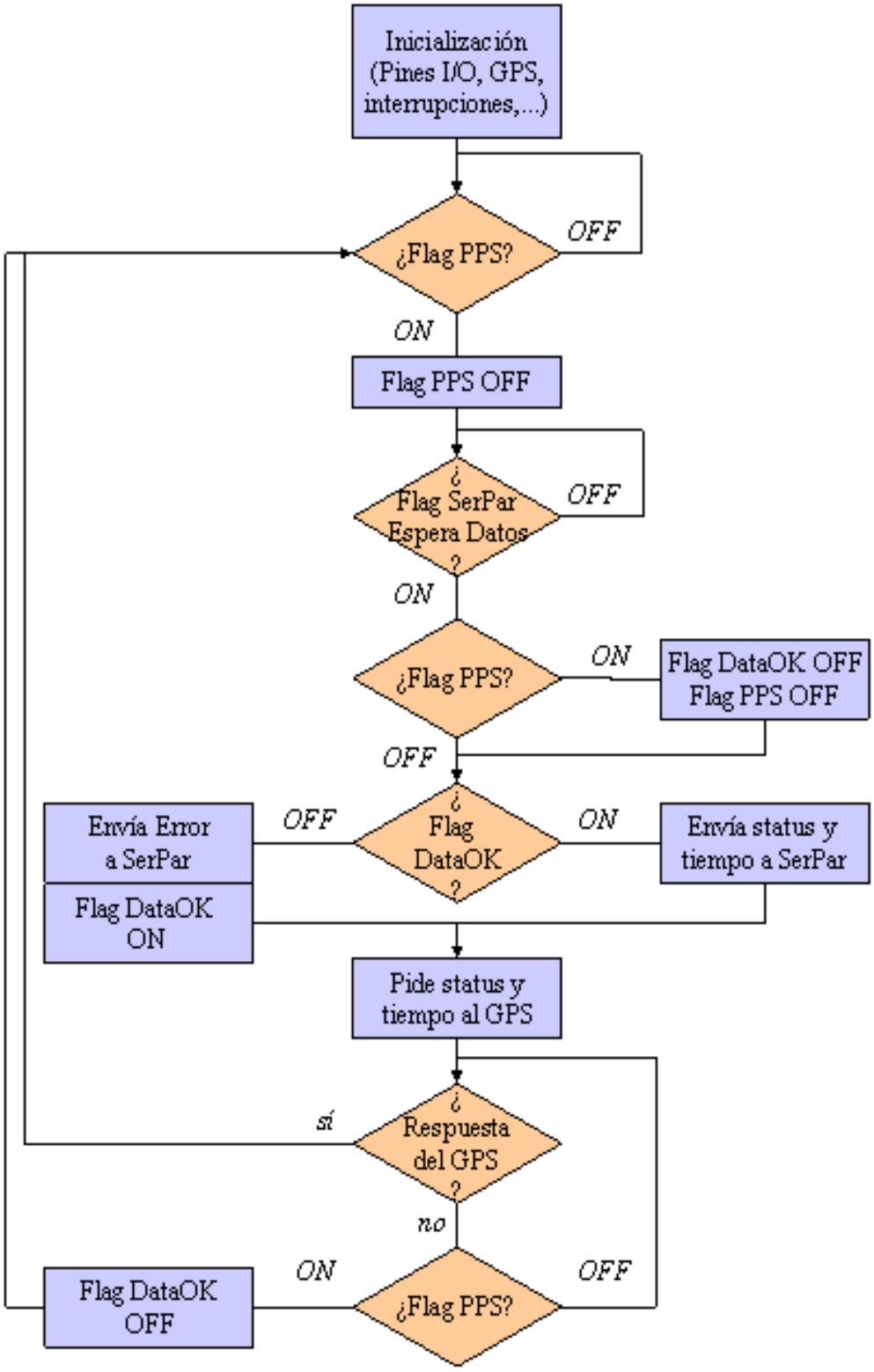
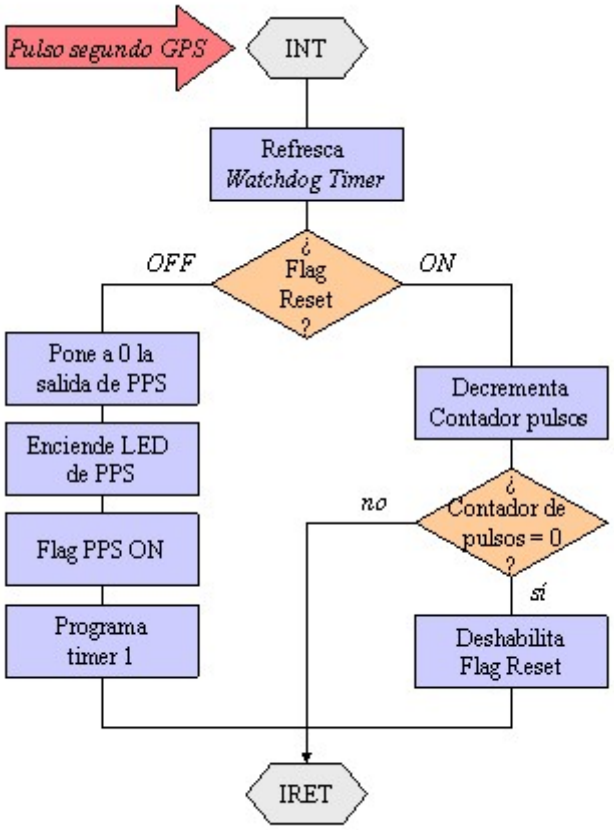
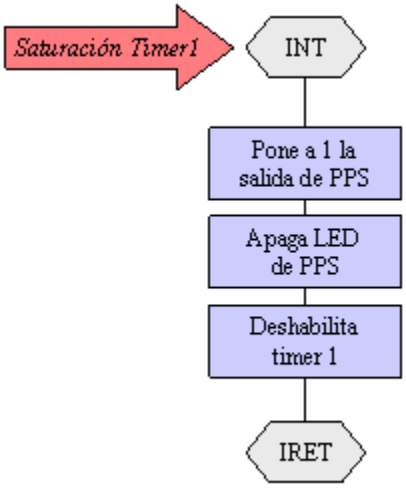


Figura 4.25. *Diagramas de flujo del programa del módulo de reloj (RELOJ5.ASM). En a), programa principal. En b), c) y d) (páginas siguientes), rutinas de servicio de las interrupciones externa, de saturación del timer 1 y de recepción de datos por el puerto USART, respectivamente.*

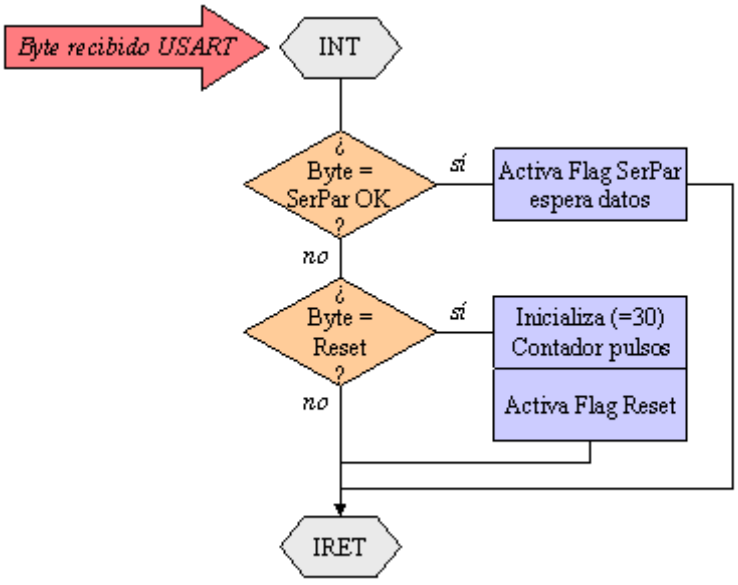
4.25.b)



4.25.c)



4.25.d)



La interrupción del timer se dispara cuando los registros del timer 1 se saturan. Como puede verse en el diagrama de flujo de la figura 4.25.c, en la rutina de gestión de esta interrupción se genera el flanco ascendente final de la señal de salida de PPS, se apaga el LED de PPS y se deshabilita el timer 1, con el objeto de que no se produzcan más interrupciones de este tipo antes de que llegue el siguiente PPS del GPS.

Por último, en la rutina de servicio de interrupción de dato recibido (figura 4.25.d) se identifica el carácter recibido y se activan los flags correspondientes: el flag de SerPar a la espera de datos, si el carácter recibido desde SerPar es de petición de datos, o el de reset si es un comando de reset del sistema. En este último caso además se inicializa a treinta el contador de pulsos durante los que se inhibirán los PPS de salida.

Aunque la función principal de esta rutina permite la identificación de errores de comunicación con el módulo SerPar (en concreto, errores de trama y de saturación del búffer de entrada), esta posibilidad no se ha implementado en el programa por simplicidad. De esta forma, si se produjera alguno de estos errores, el resultado sería que el módulo de reloj no enviaría ninguna información al módulo SerPar hasta el segundo siguiente, quedando registrado este hecho mediante la cabecera 34h (‘error de time out’) que SerPar escribiría en el módulo de memoria (ver formato de las cabeceras GPS en memoria en la sección 4.3.4 y anexo II.C.2).

4.6. Operación del sistema

4.6.1. Realización física del sistema: conectores, cajas y cables

Las figuras 4.26 a 4.32 muestran imágenes de los distintos módulos de la antena. En la figura 4.26 puede verse el contenedor utilizado para el módulo central, una maleta plástica con protección IP67. Dentro de la misma se ha dispuesto el *back-plane*, al que se conectan las placas del módulo central (placa de SerPar y tarjetas de control y de expansión de memoria), y una caja que alberga el PC, el disco duro y el circuito de fuente para ambos (figuras 4.27 y 4.28).

Todos los conectores externos son de tipo militar, para proporcionar un buen aislamiento y garantizar contactos fiables. El módulo central tiene seis conectores externos, de los cuales cuatro son para las líneas serie, uno para el módulo de reloj y otro para la batería (figura 4.29). Para evitar confusiones a la hora de conectar los distintos módulos, los conectores para las líneas serie y para el módulo de reloj (que tienen todos diez vías) se han elegido distintos: hembras los primeros y macho el del módulo de reloj. La conexión entre los conectores militares y la placa de SerPar se ha realizado mediante un conector IDC de 34 vías en la propia placa.

Los módulos de adquisición se han montado en cajas de aluminio con protección IP65 (figuras 4.30 y 4.31). Cada módulo tiene tres conectores en uno de los lados de la caja, uno en el panel frontal y otro en el posterior. Los tres primeros son para la conexión de los sensores. El frontal y el posterior son para la conexión del módulo a la línea serie. Ambos conectores son idénticos, por lo que la línea de entrada y la de salida se pueden conectar indistintamente a uno u otro. Las conexiones entre los conectores militares y las placas de los módulos de adquisición se han realizado mediante conectores IDC de diez vías. Dado que las señales son diferenciales con niveles RS-485, se ha utilizado para ello cable *Twist & Flat* (pares trenzados con secciones planas intercaladas para el montaje de los conectores IDC).

Para el módulo de reloj se ha utilizado también una caja con protección IP65 (figura 4.32). En este caso, sin embargo, dado que la antena GPS se iba a montar en el interior de la caja, esta no se ha escogido metálica sino de PVC con el objeto de permitir la correcta recepción de la señal. Además se ha elegido un modelo de caja con tapa transparente, para

permitir realizar una comprobación rápida de que el módulo de reloj está generando pulsos de segundo, mediante el LED implementado en la placa.

Los cables de transmisión serie y de comunicación con el módulo de reloj son de cuatro pares trenzados y 120Ω de impedancia característica. Por facilidad para el mantenimiento las conexiones en los módulos se realizaron de forma que todos los cables de transmisión fueran pin a pin. A la hora de definir las conexiones no se siguió ninguna regla en cuanto a los colores de los cables, si bien sí se tuvo en cuenta que las señales diferenciales debían transmitirse por un mismo par trenzado.

La alimentación de los distintos módulos de la antena se toma de una única batería situada junto al módulo central, desde donde se transmite hasta el módulo de reloj y los de adquisición. Se utilizaron los dos cables del mismo par trenzado, unidos entre sí, para enviar los 12V de la batería, y la malla metálica del cable se usó como tierra. De este modo se reduce la resistencia del cable y se disminuyen las caídas de tensión en el mismo. Esto es particularmente importante cuando se definen líneas serie con cuatro módulos de adquisición, ya que si la longitud del cable es muy grande es posible que las caídas de tensión no sean despreciables, en particular en el primer tramo en el que se suma el consumo de todos los módulos conectados a la línea serie.

Figura 4.26. *Fotografía del módulo central. A la izquierda se muestra el contenedor del* back-plane *para la tarjeta SerPar y las placas del módulo de memoria. A la derecha, la caja que contiene el PC, el disco duro y la fuente de alimentación para ambos. Pueden verse el cable plano que conecta la tarjeta SerPar con las distintas líneas serie a través de un conector IDC de 34 vías, así como el que conecta la tarjeta de control del módulo de memoria con el puerto paralelo del PC industrial, al que se accede a través de un conector D25 situado en la parte posterior de la caja del PC (figura 4.27). A la derecha de la fotografía se ve dicha caja, de color negro, con los pulsadores de reset del PC y salida del programa y el interruptor y LED de alimentación. En el lateral derecho, por la parte externa de la maleta, aparecen los cables de transmisión serie de datos, el cable de comunicación con el módulo de reloj y el cable de alimentación, conectados a los correspondientes conectores militares.*

Figura 4.27. *Fotografía del panel posterior de la caja del PC industrial, con los conectores D9 y D25 para acceder al puerto serie COM1 y al puerto paralelo del PC, respectivamente. La alimentación del PC y disco duro se toma de la batería externa a través del conector marcado como 'POWER IN'. El conector 'POWER OUT' se utiliza para llevar la tensión de la batería hasta la tarjeta SerPar. La fuente conmutada de dicha tarjeta toma esta tensión como entrada y ofrece la tensión regulada de salida (+5V) a todas las tarjetas del módulo de memoria a través del* back-plane.

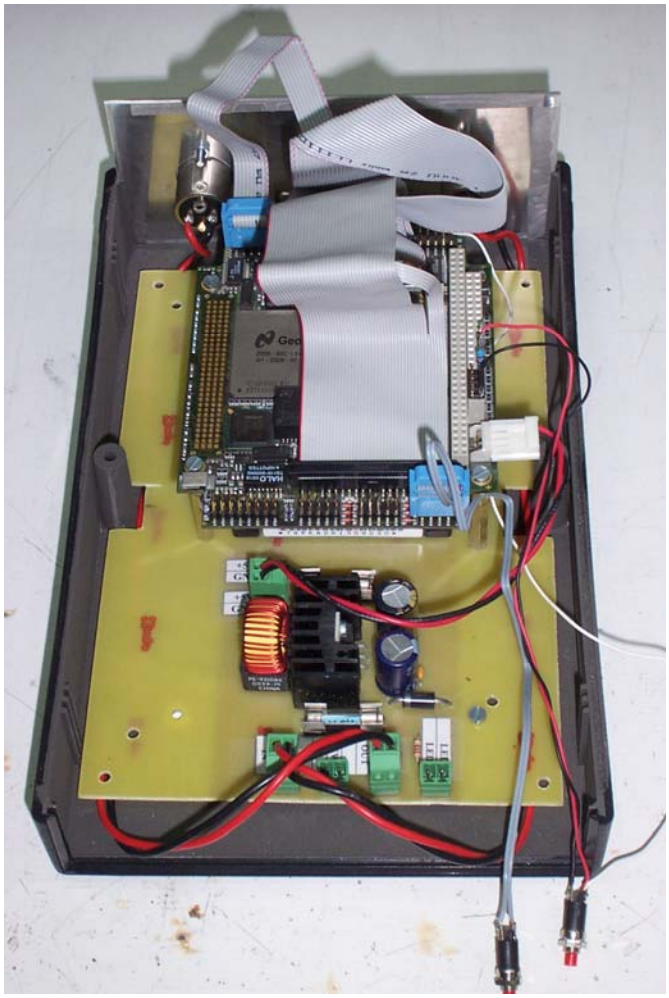


Figura 4.28. *Fotografía de la caja del PC abierta, que permite ver la disposición de la tarjeta de PC industrial y del circuito de fuente en la placa base. Los conectores verdes del borde inferior de la placa sirven para realizar las conexiones del LED e interruptor de alimentación y de la tensión de la batería. Ésta última se toma como entrada para el circuito de fuente y también se saca al conector externo 'POWER OUT' visto en la figura anterior. El otro conector verde, situado cerca del centro de la placa, corresponde a los +5V regulados de salida de la fuente, que se usan para alimentar el PC y el disco duro. Éste no se aprecia en la imagen porque queda debajo del PC.*

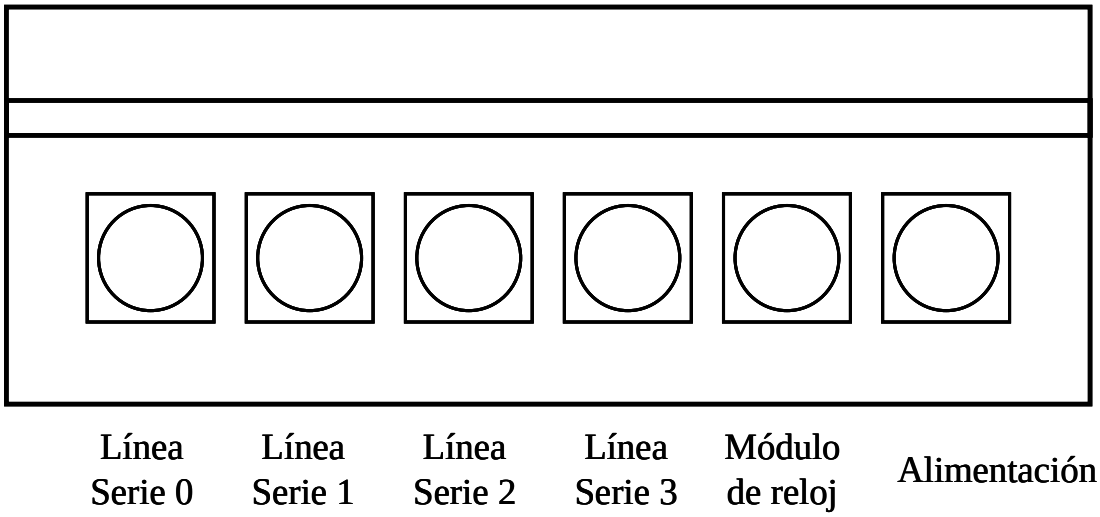


Figura 4.29. *Esquema de los conectores militares del módulo central y sus correspondientes funciones.*

Figura 4.30. *Fotografía de un módulo de adquisición, con la placa de conversión A/D en primer término y la tarjeta PLL debajo. En el exterior de la caja pueden verse los tres conectores para los sensores, en el lateral izquierdo, y los dos conectores para la conexión del módulo a la línea serie.*

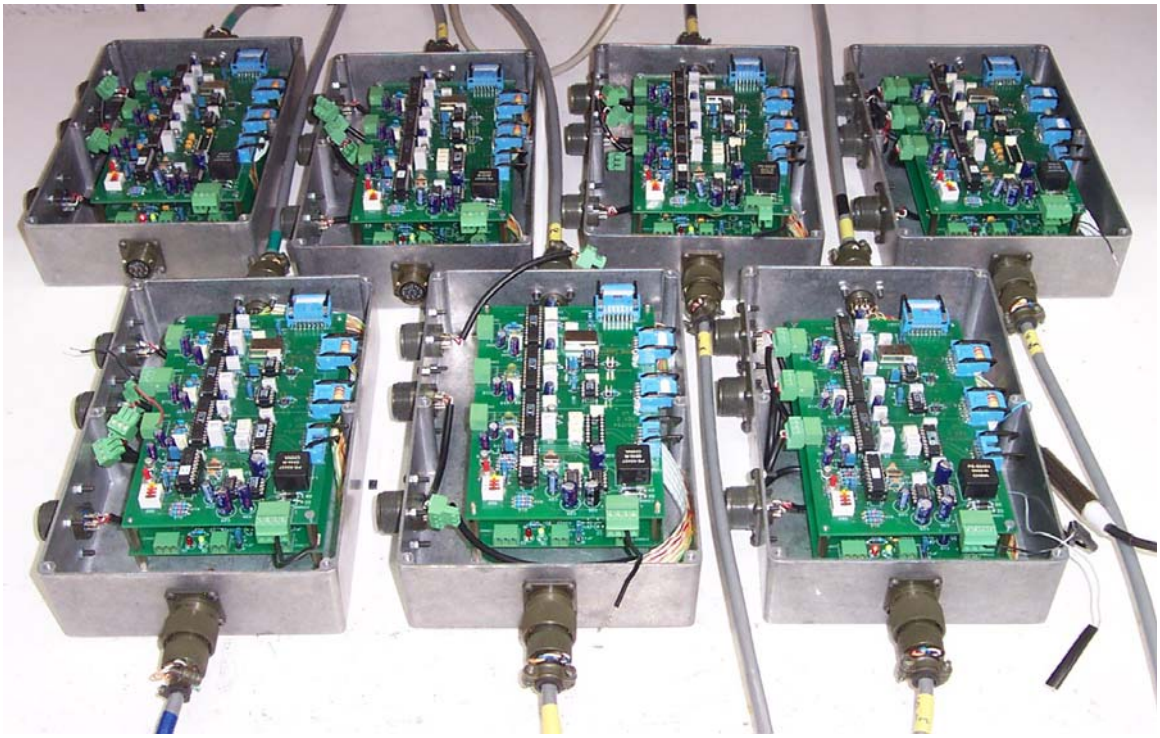


Figura 4.31. *Siete módulos de adquisición interconectados durante las pruebas del sistema en laboratorio.*

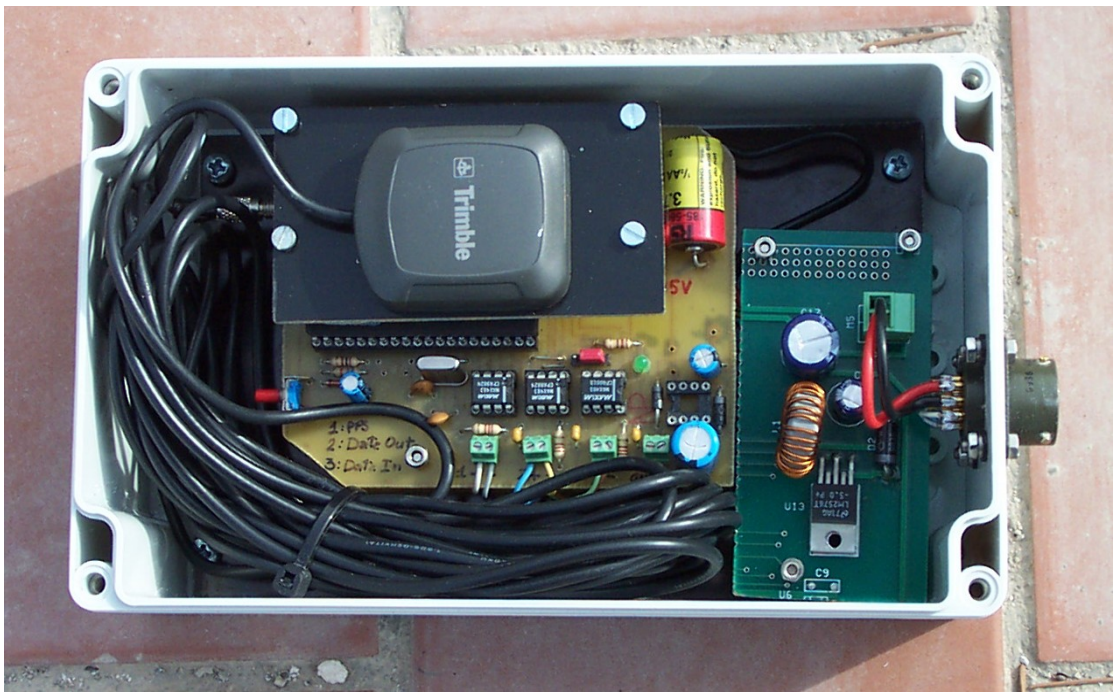


Figura 4.32. *Fotografía del módulo de reloj, en la que se aprecian la antena GPS, la placa de interfaz y la de fuente. El microcontrolador PIC16C74A y el receptor GPS quedan bajo la antena GPS, que se deja dentro de la caja, fijada mediante tornillos, para aumentar la estanqueidad del conjunto.*

Para evitar reflexiones en las líneas serie basadas en el estándar RS-485 es recomendable adaptar cada línea con una resistencia del valor de la impedancia característica del cable (Goldie, 1992 y 1996). En nuestro caso se ha montado, en cada una de las interfaces RS-485 de los módulos de adquisición, una resistencia de 120Ω, que debe conectarse a la línea cerrando un *jumper* en caso de que dicho módulo ocupe el extremo de la línea serie (ver sección 4.4.2).

Para realizar la conexión física de todos los elementos del sistema simplemente hay que utilizar los cables correspondientes que, como se ha dicho, no presentan posibilidad de confusión por usar conectores distintos según la función del cable. Para realizar la conexión de los módulos de adquisición a las líneas serie se parte del módulo central, teniendo en cuenta que el conector más alejado del de alimentación corresponde a la línea serie 0 y el más cercano a la línea serie 3 (ver figura 4.29). Hay que asegurarse de asignar códigos de identificación o posición lógica distintos a los módulos de adquisición de la misma línea serie (sección 4.4.2). Lo usual es asignar números crecientes en función de la distancia al módulo central, pero puede usarse cualquier criterio que se considere oportuno. Como se verá en las próximas secciones, también es necesario verificar que la asignación física de códigos se corresponde con la configuración habilitada por *software* en el fichero de configuración VES2.CFG.

4.6.2. Arranque del sistema

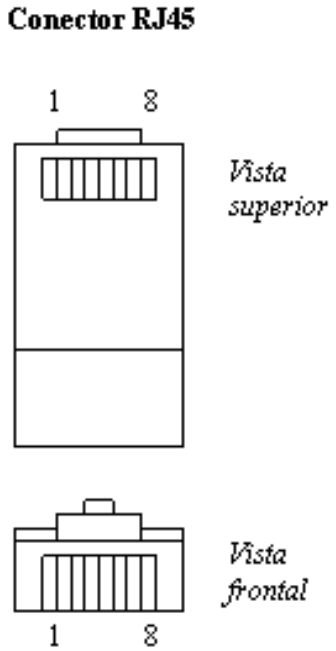
Para iniciar la operación del sistema debe seguirse la siguiente secuencia:

1. Poner los *jumpers* y microinterruptores de todas las placas de adquisición y PLL en las posiciones descritas en el apartado 4.4.2, incluyendo los códigos de identificación de las estaciones en cada línea serie.
2. Activar los microinterruptores de identificación de las tarjetas de memoria, según lo explicado en la sección 4.4.4, y conectarlas al *back-plane* del módulo central. También es necesario modificar el parámetro correspondiente en el fichero de configuración VES2.CFG (ver sección 4.6.4).
3. Conectar los distintos módulos entre sí, según lo descrito en el apartado 4.6.1.
4. Conectar una batería al conector de alimentación del módulo central.
5. Poner el interruptor de alimentación en la posición ON. El sistema comenzará a adquirir y grabar datos con los parámetros especificados en los ficheros de configuración VES2.CFG y RDMEM2.CFG. Para verificar que el sistema ha iniciado la operación correctamente puede comprobarse la secuencia inicial del LED del módulo SerPar, que se describe en el anexo II.C.1.c.
6. Si se desea modificar el fichero de configuración, comprobar el funcionamiento de los distintos canales o realizar cualquier otra tarea de mantenimiento puede realizarse una conexión remota con un PC portátil a través del conector de red según se explica en la próxima sección. Alternativamente, puede conectarse al PC un monitor, teclado y ratón utilizando los cables de expansión suministrados con la placa. Para realizar una comprobación de los canales el sistema debe arrancarse en modo TEST, como se explica en la sección 4.6.5. La modificación de los parámetros de operación, el volcado de datos y la selección del formato de los ficheros de salida se explican en las secciones 4.6.4, 4.6.6 y 4.6.7, respectivamente.

4.6.3. Conexión remota con un PC portátil

Aunque es posible conectar los periféricos usuales al PC de control (monitor, teclado, ratón, disquetera,...) y utilizarlo como un ordenador de sobremesa, normalmente las condiciones de trabajo no permitirán hacerlo. Resulta más sencillo trabajar desde un PC portátil con tarjeta de red *ethernet*, para lo cual es necesario establecer conexión siguiendo los pasos que a continuación se describen:

1. Conectar el PC portátil y el PC interno de control mediante un cable de transmisión de datos de cuatro pares trenzados, cuyas conexiones se muestran en la figura 4.33.
2. El PC de control carga, al arrancar, un programa servidor para conexión remota (*VNC Server*[37]). Desde el PC portátil debe ejecutarse la versión para cliente del mismo programa (*VNC Viewer*).
3. Una vez se ha establecido la conexión, aparecerá en el PC portátil una ventana con el escritorio del PC de control (o la ventana activa en el mismo en ese momento). Se podrá acceder a cualquier icono o realizar cualquier acción mediante el ratón y el teclado del portátil, si bien algunas teclas de control no funcionarán (ej.: tecla del menú de inicio de *Windows*).
4. Para interrumpir la comunicación basta con cerrar la ventana de conexión. Sin embargo, si se desea apagar el PC interno, antes de interrumpir la conexión es necesario realizar remotamente los pasos normales en W98. Es decir, seleccionar el menú 'Inicio' y, dentro de él, la opción 'Apagar sistema'. Una vez hecho esto la comunicación se perderá, con lo cual la pantalla de conexión desaparecerá, y habrá que esperar unos diez segundos antes de apagar el interruptor de alimentación.



Nombre	*Pin con. 1*	*Color*	*Pin con. 2*	*Nombre*
TX+ (BI_DA+)	1	Blanco/ Naranja	3	**RX+** (BI_DB+)
TX- (BI_DA-)	2	Naranja	6	**RX-** (BI_DB-)
RX+ (BI_DB+)	3	Blanco/ Verde	1	**TX+** (BI_DA+)
(BI_DC+)	4	Azul	7	(BI_DD+)
(BI_DC-)	5	Blanco/ Azul	8	(BI_DD-)
RX- (BI_DB-)	6	Verde	2	**TX-** (BI_DA-)
(BI_DD+)	7	Blanco/ Marrón	4	(BI_DC+)
(BI_DD-)	8	Marrón	5	BI_DC-)

Figura 4.33. *Conexiones del cable de red para acceso remoto al PC de control. Deben usarse dos conectores RJ45 macho y cable de cuatro pares trenzados. Los colores indicados son los normalmente utilizados en cables de transmisión de datos para redes ethernet.*

[37] www.realvnc.com

4.6.4. Modificación de los parámetros de operación

El valor de los principales parámetros de operación del sistema se selecciona en el fichero de texto VES2.CFG, al que puede accederse mediante la conexión remota descrita en la sección anterior. Los cuatro parámetros seleccionables son:

- Frecuencia de muestreo. Los valores posibles son 50, 100 o 200 mps.
- Ganancia. Los valores posibles son 1, 2, 4, 8, 16, 32, 64 o 128.
- Configuración física del array: es necesario comunicarle al módulo SerPar cuántos módulos de adquisición están operativos y qué posiciones ocupan en cada línea serie. Para ello se le envía una secuencia de dieciséis bits, en la que un 1 representa que el correspondiente módulo se encuentra operativo y un 0 que no lo está. El orden en el que los módulos están representados en la cadena del fichero de configuración es el siguiente:

  L0M0/L1M0/L2M0/L3M0/L0M1/... /L3M3

  siendo L = Número de línea serie
  M = Número de módulo A/D dentro de la línea serie

- Número de tarjetas de memoria usadas. Los valores posibles son 0, 1, 2 o 4. Como se describe en la próxima sección, si se selecciona el valor 0, el tamaño de los bancos de memoria se fija en 10 segundos, independientemente del valor del resto de parámetros de operación.

Una vez seleccionados los valores deseados, debe reiniciarse el sistema. Para ello se debe salir del sistema operativo del PC interno de control desde el menú 'Inicio' a través de la conexión remota, como se ha explicado en la sección anterior. Posteriormente se debe apagar y volver a encender el interruptor de alimentación del módulo central.

4.6.5. Arranque en modo TEST

El modo TEST permite realizar una comprobación rápida del correcto funcionamiento de los distintos canales. Para arrancar en este modo simplemente hay que poner a 0 el valor del número de tarjetas en el fichero de configuración VES2.CFG (aunque debe haber al menos una tarjeta de memoria físicamente conectada) y reiniciar el sistema. Esto hará que el número de segundos de cada banco de memoria se fije a diez, independientemente del valor del resto de los parámetros de operación. Cada vez que se lea un banco de memoria la señal se mostrará en pantalla, como puede verse en la figura 4.34. La pantalla muestra además información sobre los parámetros de muestreo y sobre los módulos de adquisición operativos. El módulo de adquisición y el canal cuya señal se muestra pueden seleccionarse con las teclas 'M' y 'C', respectivamente. Además puede realizarse un zoom de la señal con la tecla 'S', que conmutará entre dos posibles escalas. El fichero de texto RDMEM2.HLP muestra una ayuda sobre el manejo del programa RDMEM2. Las incidencias durante el funcionamiento se escriben en el fichero RDMEM2.LOG, tanto si se opera en modo TEST como en modo normal.

El resto de la información que muestra la pantalla del programa RDMEM2 es el estado en que se encuentra (esperando acceso a alguno de los dos bancos o mostrando datos), el tiempo GPS del primer dato mostrado, el estado del receptor GPS y el nombre del fichero en el que se han escrito los últimos datos monitorizados. Cuando opera en modo TEST, el

programa graba los datos en ficheros de diez segundos cuyos nombres son TEST*.DAT y su formato el mismo que el de los ficheros reales de datos (ver sección 4.6.7).

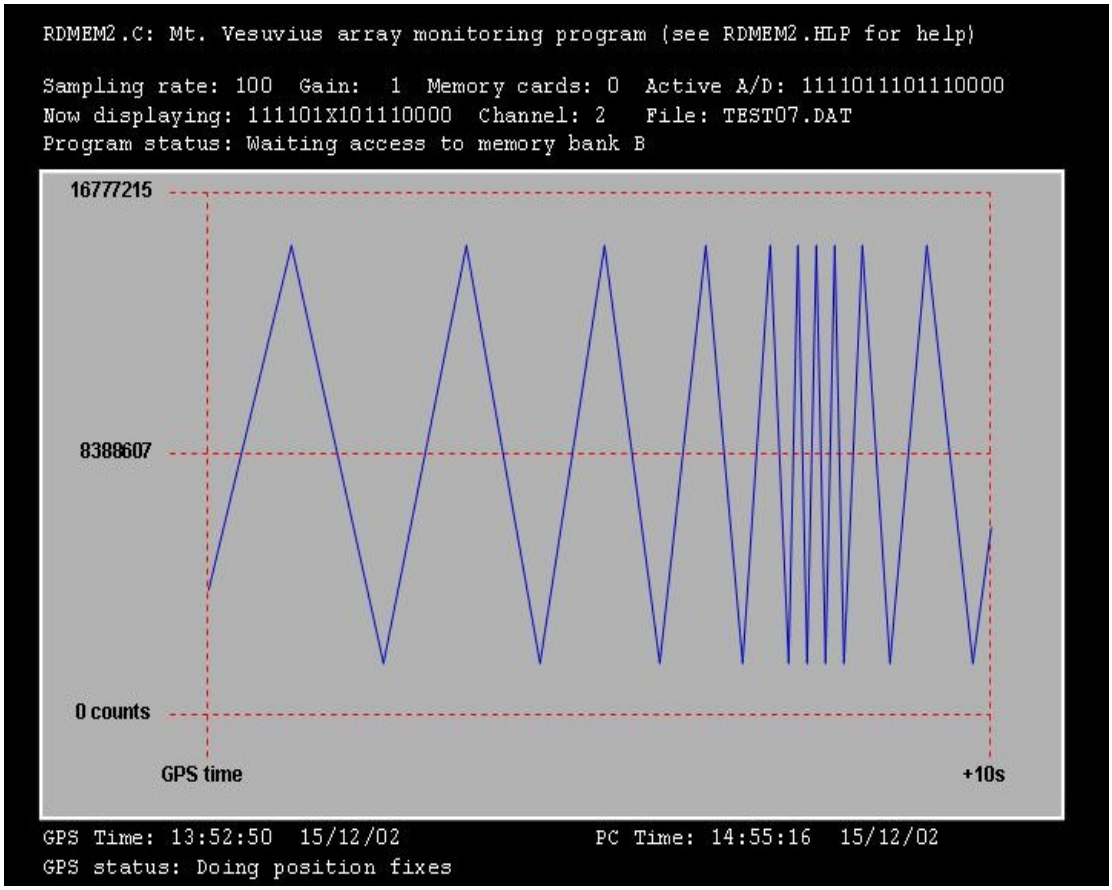


Figura 4.34. *Pantalla del programa de adquisición RDMEM2 en modo TEST, en la que se muestra una señal triangular de frecuencia variable utilizada durante las pruebas del sistema en laboratorio.*

4.6.6. Volcado de datos

Durante la operación normal del sistema, los datos se grabarán en el disco duro interno. Para acceder a ellos se puede realizar una conexión con un PC portátil a través de un cable de red *ethernet*, como se ha descrito en la sección 4.6.3. En este caso no es necesario, sin embargo, utilizar el programa VNC, ya que no hace falta visualizar el escritorio del PC de control. Se puede acceder directamente al disco duro interno mediante el icono 'Mis sitios de red' o 'Mi PC' del PC portátil. Es necesario, por supuesto, que el disco duro interno (o los directorios a los que queramos acceder) esté compartido para lectura. Si esto no fuera así, podría compartirse accediendo al 'Explorador de Windows' del PC interno de control mediante la conexión con el programa VNC.

4.6.7. Formato de los ficheros de datos

El programa de lectura de datos del PC de control (RDMEM2) permite seleccionar el tipo y duración de los ficheros de salida. Para hacerlo debe editarse el fichero de configuración

RDMEM2.CFG. Cada opción se activa o desactiva escribiendo las palabras ON u OFF, respectivamente, al final de la línea correspondiente.

Los datos pueden escribirse en dos posibles tipos de ficheros: de datos completos o de cabeceras GPS, que tendrán las extensiones .DAT o .GPS, respectivamente (a partir de ahora nos referiremos a ellos como ficheros DAT o ficheros GPS). Es posible seleccionar uno de los dos tipos de ficheros, los dos o ninguno de ellos, en cuyo caso no se generará ningún fichero de salida, salvo el RDMEM2.LOG. Éste es un fichero de texto en el que se describe el funcionamiento del sistema, y que se genera independientemente del tipo y extensión seleccionada para los ficheros de datos.

Los ficheros DAT son copias de uno o más bancos de memoria (el número de bancos viene determinado por el valor del parámetro 'Fichs1Hora', que se explica más adelante). El formato de los ficheros, por tanto, puede deducirse de la estructura de los bancos de memoria descrita en la sección 4.3.4, teniendo en cuenta que los bytes no usados no se escriben en los ficheros de datos. La figura 4.35 muestra la estructura de un fichero de datos completo. El formato de cada uno de los bloques es el explicado en la sección 4.3.4 y el anexo II.C.2.

Los ficheros GPS permiten realizar una comprobación rápida del estado del receptor GPS y de la información de tiempo durante el proceso de adquisición de datos. Estos ficheros tan solo incluyen las cabeceras de banco con los parámetros de configuración y las cabeceras GPS de cada segundo. Además, antes de cada cabecera GPS (salvo en el caso de la primera) se escriben los últimos cuatro bytes de datos del segundo anterior. Esto se hace simplemente para facilitar la visualización de las cabeceras GPS con programas de edición de ficheros binarios. La estructura de los ficheros GPS es la misma que la de los ficheros DAT (figura 4.35), si bien hay que tener en cuenta que en este caso los paquetes de datos no son los descritos en el anexo II.C.2, sino simplemente los últimos cuatro bytes de datos de cada segundo.

La nomenclatura de los ficheros DAT y GPS es la misma, salvo su extensión:

MMDDHHmm.EEE

siendo:

MM: Mes.
DD: Día.
HH: Hora.
mm: minuto.
EEE: extensión (DAT o GPS).

La fecha y hora indicadas por el nombre del fichero corresponden a la primera muestra del mismo. El segundo de adquisición de la primera muestra puede conocerse mediante las cabeceras GPS del fichero de datos. Hay que tener en cuenta que, según se explica en el anexo II.C.2, las cabeceras de tiempo corresponden a los datos adquiridos durante el segundo anterior, por lo que el tiempo de adquisición correspondiente a la primera muestra de cada fichero sería el indicado por la segunda cabecera GPS. Por otra parte, también es importante indicar que la información de tiempo de las cabeceras sólo es significativa hasta el segundo. La parte de la fracción de segundo corresponde al instante en el que el receptor GPS actualizó la información de tiempo, y no debe tenerse en cuenta. La fracción de segundo de adquisición del primer dato de cada segundo es siempre 200 ± 25us, ya que los módulos PLL fijan la diferencia de tiempo entre la señal de PPS del GPS y la primera muestra a ese valor.

El parámetro 'Fichs1Hora' permite seleccionar la longitud de los ficheros de datos. Si se pone a OFF los ficheros son cortos, de menos de 1.44MB (salvo cuando se usen cuatro tarjetas de memoria, ya que en ese caso la capacidad de cada banco de memoria es mayor que

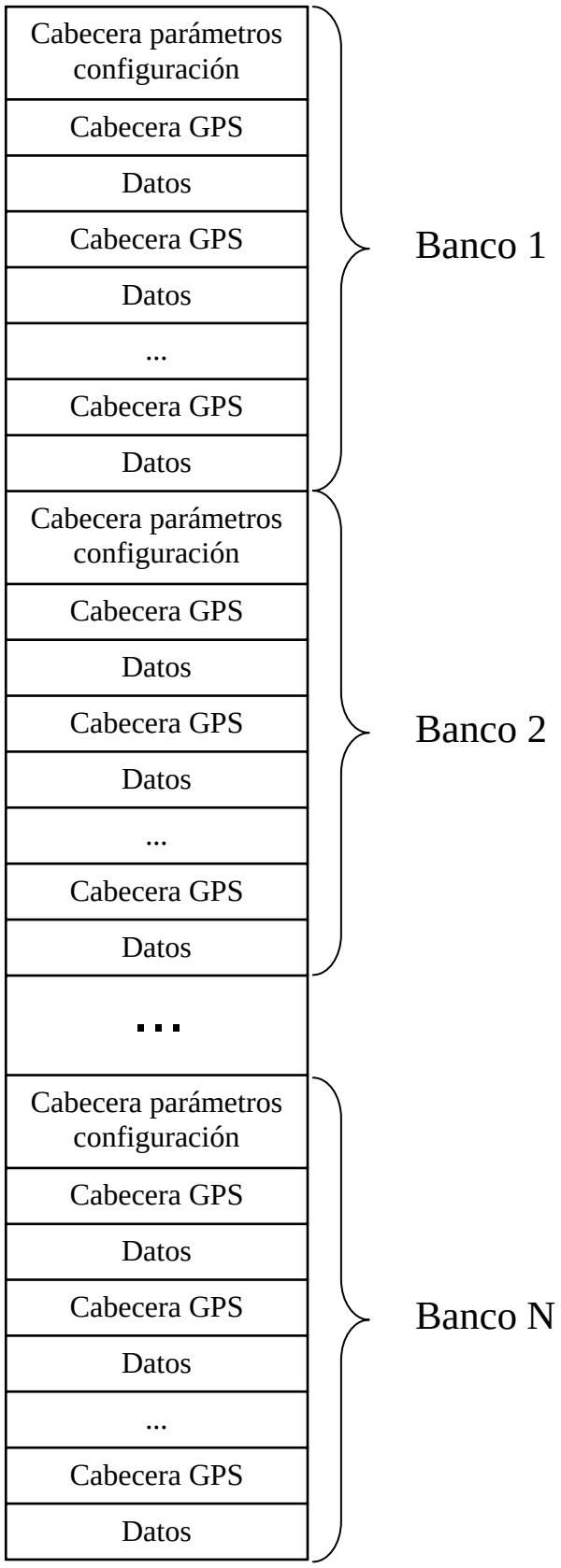


Figura 4.35. *Formato de los ficheros de datos (.DAT) y de cabeceras GPS (.GPS). Explicaciones en el texto.*

ésa), con el objeto de poder gestionarlos mediante disquetes. Esto puede resultar particularmente útil cuando se está realizando alguna prueba de funcionamiento del sistema. El tamaño exacto de los ficheros depende de los parámetros de operación del sistema.

Si el parámetro 'Fichs1Hora' se pone a ON, los ficheros de datos tendrán una duración de una hora. Esta duración no es exacta, sólo aproximada, debido a que los bancos de memoria se leen y se escriben completos en los ficheros de datos. Para saber en qué fichero debe grabar los datos, el programa RDMEM2 sólo comprueba la información horaria del primer segundo de cada banco, por lo que es probable que haya datos del comienzo de una hora en el fichero cuyo nombre corresponde a la hora anterior. Lo importante, sin embargo, es que cada bloque de un segundo de datos lleva asociada la información sobre el tiempo de adquisición de la primera muestra del bloque (Figura 4.35).

Cuando el sistema se inicia en modo TEST (ver sección 4.6.5) el parámetro 'Fichs1Hora' no se tendrá en cuenta. Independientemente de su estado en el fichero RDMEM2.CFG y de los parámetros de operación del sistema, se grabarán ficheros de diez segundos de duración, con la misma estructura de la figura 4.35 y cuyos nombres serán TEST*.DAT y/o TEST*.GPS.

4.6.8. Demultiplexado de los ficheros DAT

Según la estructura descrita en el punto anterior para los ficheros DAT, estos contienen toda la información relativa al tiempo de adquisición, a los parámetros de operación del sistema y a los datos de los diferentes canales operativos. Antes de poder visualizar o trabajar con los datos es necesario un preprocesado que elimine las cabeceras

de tiempo y de parámetros de operación y que demultiplexe los datos de los diferentes canales. Para ello se utiliza el programa FICHVES2, que toma como entrada un fichero DAT y da como salida los ficheros binarios de datos correspondientes a los canales que se deseen visualizar.

Al ejecutar el programa FICHVES2 se pedirá el nombre del fichero DAT de entrada. Una vez introducido, el programa leerá los parámetros de operación del sistema de la primera cabecera y los mostrará en pantalla. En función de ellos realizará el demultiplexado de los datos, que se lleva a cabo en dos pasos. En primer lugar se separan los datos correspondientes a cada uno de los módulos A/D, generando tantos ficheros de salida como módulos haya operativos. La nomenclatura de estos ficheros es:

MMDDHHmm.LA

donde MM, DD, HH y mm tienen los mismos significados que antes, y L y A los siguientes:

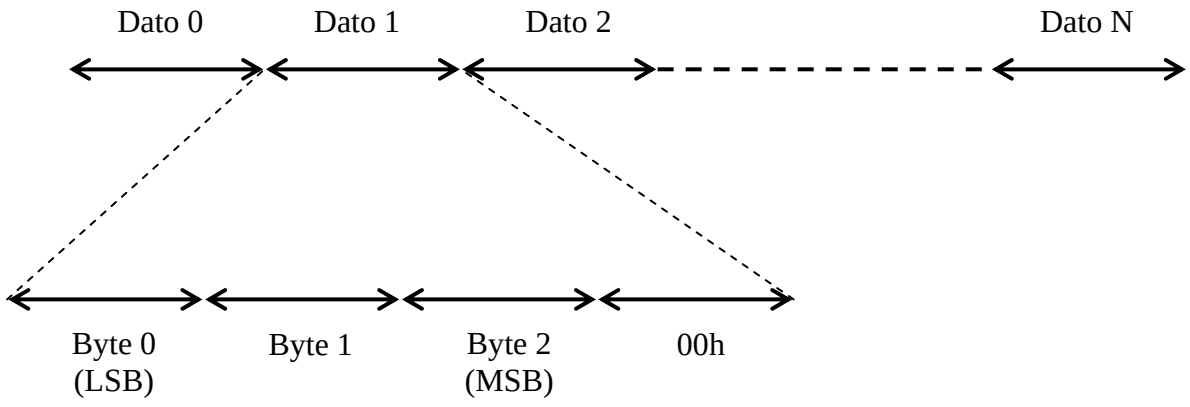
L: Número de línea serie (0, 1, 2 o 3).
A: Número de módulo de adquisición dentro de la línea serie (0, 1, 2 o 3).

El segundo paso en el demultiplexado de los datos es el que se realiza a partir de estos ficheros intermedios, para obtener los ficheros de datos de cada canal. Con el objeto de evitar tener un número elevado de ficheros finales, el programa pide confirmación sobre cuáles de los ficheros intermedios se desea demultiplexar. Cada vez que se selecciona uno de ellos, el programa genera tres ficheros más de salida, cuya nomenclatura es:

MMDDHHmm.LAC

donde MM, DD, HH, mm, L y A tienen los mismos significados que antes, y C es el número de canal (0, 1 o 2) del módulo de adquisición A de la línea serie L.

El formato de los ficheros finales de datos es el siguiente:



Como se muestra, los ficheros finales generados por el programa FICHVES2 son una sucesión de datos de un mismo canal, sin ninguna cabecera ni información adicional. Cada dato está compuesto por tres bytes, comenzando por el menos significativo (LSB), más un cuarto byte con todos los bits a 0. Este byte es añadido por el programa FICHVES2 para facilitar la visualización del fichero con programas de tratamiento de datos, que normalmente aceptan formatos de cuatro bytes por dato.

CAPÍTULO 5

LAS ANTENAS PORTÁTILES DEL IAG

La tercera y última antena sísmica que se describe en esta memoria es la que se diseñó y desarrolló entre los años 2002 y 2004 para el Instituto Andaluz de Geofísica (IAG), de la Universidad de Granada. En realidad no se puede hablar de una sola antena, sino de varios módulos (en concreto, catorce) diseñados para operar independientemente, pero que gracias a la base de tiempo común que proporcionan los receptores GPS, pueden utilizarse como un único sistema de registro. Como se verá, estas antenas pueden considerarse como una simplificación de la del Vesubio, con cuyo diseño comparten aspectos comunes tanto desde el punto de vista *hardware* como de programación. El objetivo principal que se pretendía cubrir con el desarrollo de estos dispositivos era sustituir otros módulos de *array* portátiles de dieciséis bits que se habían venido utilizando en el IAG desde principios de los años noventa.

El capítulo comienza con una introducción (sección 5.1) en la que, a diferencia de los capítulos anteriores, no se profundizará en la actividad sísmica histórica en el área geográfica próxima al IAG. La razón es que las antenas que se presentan en este capítulo están concebidas como dispositivos portátiles que, salvo en casos muy concretos, no se dedicarán al estudio de la actividad local. Si bien la antena del Vesubio también se diseñó y desarrolló como dispositivo portátil, su objetivo fundamental era el seguimiento de la actividad del volcán, por lo que iba a contar con un emplazamiento casi permanente en la falda del mismo. No es éste el caso de las antenas que aquí se presentan, ya que la orientación actual de la actividad científica en el IAG hace que el uso previsto para estos módulos sea de prácticamente una campaña de registro de datos anual en áreas volcánicas activas, y por tanto lejos de la región geográficamente próxima al IAG.

La introducción está dividida en dos partes. La primera (5.1.1) hace un breve repaso de la historia del IAG y de la que puede considerarse como su institución precursora, el Observatorio de Cartuja, centrándose en la vertiente de la instrumentación sísmica. Quizás este repaso histórico no era necesario para el desarrollo de esta tesis, pero debido a los lazos sentimentales que me unen a esta institución me he permitido la licencia de hacerlo como homenaje a todos aquellos que, con precariedad de medios y recursos, sentaron las bases del desarrollo instrumental en los campos de la astronomía y sismología en nuestro país.

La segunda parte de la introducción (5.1.2) describe de una manera general la estructura y funcionamiento de los módulos de dieciséis bits a los que los sistemas que se presentan en este capítulo sustituirán. Teniendo en cuenta que se pretende adoptar la filosofía de operación de los antiguos módulos, el conocimiento de sus características y limitaciones servirá como referencia para establecer los requerimientos técnicos de los nuevos dispositivos.

El resto del capítulo seguirá una estructura prácticamente idéntica a la de los dos anteriores: después de la introducción se plantean los objetivos que se pretende cubrir con el dispositivo y se especifican sus características técnicas (5.2). A continuación se presenta de una manera general la estructura y funcionamiento de las antenas (5.3), para dar luego una descripción detallada de las mismas desde el punto de vista *hardware* (5.4) y *software* (5.5). Por último, se incluye una sección relativa a la operación de los sistemas desde el punto de vista del usuario, en la que también se muestran imágenes de los dispositivos terminados (5.6).

5.1. Introducción

5.1.1. El Observatorio de Cartuja

El Observatorio de Cartuja (figura 5.1), situado en el campus universitario del mismo nombre de la ciudad de Granada, fue fundado en 1902 por los Padres Jesuitas de la Facultad de Teología de Cartuja. Prácticamente desde el comienzo de su andadura se estructuró en tres secciones: astronómica, geodinámica y meteorológica, si bien la geodinámica se redujo en principio a estudios de sismología, debido a que los cables de alta tensión y los tranvías de la capital hacían ineficaz la instalación de instrumentación para la medida de fenómenos relacionados con el magnetismo terrestre y las corrientes telúricas (Espinar, 2003). El emplazamiento elegido resulta ideal para el estudio de las tres disciplinas, como describe la revista *El Mundo Científico* en su número de agosto de 1903:'…*No podía elegirse mejor campo de observación para fundar en él el nuevo establecimiento. Si el cielo de Granada es espléndido como pocos, la tierra trepida aún allí con el eco de terribles sacudidas, y en el seno de su atmósfera se elabora la riqueza ó la miseria de una dilatada comarca, de uno de los oasis más preciados de los desiertos ibéricos'*.

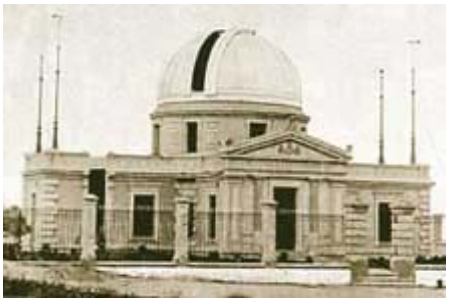


Figura 5.1. *Fotografía del edificio original del Observatorio de Cartuja. Posteriormente ha sufrido diversas modificaciones y ampliaciones, aunque se mantiene la fachada original.*

Levantado en una colina en las afueras de la ciudad de Granada, pronto se mostró como un lugar excelente para las observaciones astronómicas, con un 60% de noches despejadas que permitían observaciones astronómicas regulares. Durante su funcionamiento como observatorio astronómico se consiguieron hitos importantes, como la puesta en marcha del primer fotómetro fotoeléctrico de España, desarrollado en el propio Observatorio. Asimismo, la institución adquirió desde muy pronto gran renombre en el campo de la astrofísica solar (Vives, 2003).

Sin embargo, a partir de 1960 la contaminación lumínica provocada por la gran expansión geográfica de la vecina ciudad de Granada empezó a dificultar las observaciones nocturnas, por lo que se hizo necesario buscar un emplazamiento alternativo para dotar al centro de una estación astronómica con mejores condiciones para la observación. Se eligió la colina denominada "Mojón (o Mohón) del Trigo", en Sierra Nevada, a 2.700 metros de altura, cercana al Parador de Turismo de Sierra Nevada y de fácil acceso por su proximidad a la carretera.

En esta etapa se llevó a cabo una labor investigadora significativa, en colaboración con instituciones extranjeras como el Observatorio de Greenwich (Inglaterra) o el de la Universidad de Georgetown, Washington (USA). La labor desarrollada en el Observatorio, tanto desde el punto de vista científico como técnico, sentó una sólida base sobre la que se sustentó el nacimiento de la astrofísica moderna en España. Sin embargo, con el desarrollo que a partir de los años setenta experimentó esta disciplina en nuestro país, las relativamente modestas instalaciones del Observatorio de Cartuja dejaron de ser competitivas frente a los grandes observatorios internacionales que comenzaron a funcionar, como el Observatorio de Calar Alto, del Centro Astronómico Hispano Alemán, o los de Tenerife y de Las Palmas de Gran Canaria. Asimismo, se crearon importantes institutos de astrofísica como el de Canarias o el de Andalucía. Este último puso en funcionamiento, en el año 1981, el Observatorio de Sierra Nevada, en un emplazamiento próximo al del Mojón del Trigo y con equipamiento e

instrumentación de mejores prestaciones. El Observatorio de Sierra Nevada podría, por tanto, considerarse como un relevo actualizado de las antiguas instalaciones del Observatorio de Cartuja en el esfuerzo investigador en el campo de la astrofísica en Andalucía. Ante esta situación, la sección de astronomía del Observatorio de Cartuja dejó de tener sentido, y actualmente esa línea se ha abandonado de forma definitiva.

Los estudios y recolección de datos meteorológicos se han mantenido hasta la actualidad, constituyéndose el Observatorio de Cartuja en el único centro en España que ha mantenido de forma continuada desde principios del siglo XX la recogida de datos de la instrumentación meteorológica.

La originalmente conocida como sección de geodinámica también se ha mantenido hasta la fecha, sumándose a la sismología una nueva área de estudio en 1994, cuando quedó constituida el área de prospección[38]. Originalmente ésta empezó realizando labores de geofísica aplicada a la arqueología mediante métodos eléctricos y magnéticos. Posteriormente amplió su campo de acción al reconocimiento del subsuelo en general, mediante el empleo de métodos electromagnéticos y sísmicos de alta resolución. En la actualidad sus actividades se extienden a obras civiles, arqueología y patrimonio, recursos naturales, medio ambiente, geología e ingeniería sísmica, para lo cual se aplican técnicas de georrádar, prospección sísmica, prospección magnética y prospección eléctrica en corriente continua. En todos estos métodos de geofísica de alta resolución es fundamental un sistema de coordenadas fiable y que permita un replanteo de precisión, para lo cual se aplican técnicas de posicionamiento GPS. Para la aplicación de todos estos métodos el área de prospección cuenta con un moderno equipo constituido por un georrádar con antenas para distintas frecuencias, un sismógrafo para prospección sísmica, un magnetómetro de protones, un magnetómetro de potasio, un resistivímetro para prospección eléctrica y un equipo GPS bifrecuencia, capaz de dar la posición de un punto con errores menores de 5 cm.

Sin embargo, el campo de mayor interés dentro del contexto de este trabajo es el de la sismología. Desde la fundación del Observatorio ésta fue una de las disciplinas a las que se dedicó más esfuerzo, dados los precedentes sísmicos del área de Granada. No en vano, menos de veinte años antes de la fundación del Observatorio, concretamente en 1884, se había producido el terremoto de Alhama[39], que había dejado cientos de víctimas mortales y cuantiosos daños materiales.

El año 1971 podría considerarse como el punto de partida de la etapa moderna del Observatorio de Cartuja. En esa fecha la Orden de los Jesuitas, fundadora y gestora hasta entonces de la institución, cedió el control a la Universidad de Granada, primero de forma temporal y luego, al terminar el período estipulado inicialmente (veinticinco años), de modo definitivo[40].

A finales de esa década el único personal científico con que contaba el Observatorio era un reducido grupo de geofísicos, puesto que los astrofísicos que habían estado realizando su labor investigadora allí ya se habían incorporado al Instituto de Astrofísica de Andalucía, del CSIC, o a departamentos de la Universidad de Granada. En cuanto a la instrumentación sísmica, sólo había operativo un sismógrafo de tres componentes y registro fotográfico (Alguacil, 2003). Desde entonces, y gracias al esfuerzo constante en el diseño, desarrollo y mantenimiento de los equipos, la cantidad y calidad de la instrumentación controlada por el Observatorio de Cartuja (hoy sede del IAG[41]) fue creciendo hasta constituir, actualmente, una

---

[38] www.ugr.es/~geofisic/

[39] www.ugr.es/~iag/divulgacion/div_i.html

[40] Una descripción detallada de las actividades investigadoras y técnicas del Observatorio de Cartuja en el campo de la sismología durante la etapa anterior a 1971 puede consultarse en *Batlló, 2003*.

[41] www.ugr.es/~iag/

de las redes más modernas y completas de Europa[42]. La figura 5.2 muestra un plano de la red sísmica regional del IAG, que consta de estaciones de corto periodo, banda ancha y acelerómetros con telemetría por radio o interrogación automática vía internet o teléfono GSM. Además de la red regional, en el IAG se han desarrollado diversos tipos de estaciones y antenas portátiles, como los que se van a describir en este capítulo, que han sido utilizados en distintas campañas en entornos sísmicos y volcánicos en el sur de España, Islas Canarias y Azores, Italia y la Antártida, entre otros.

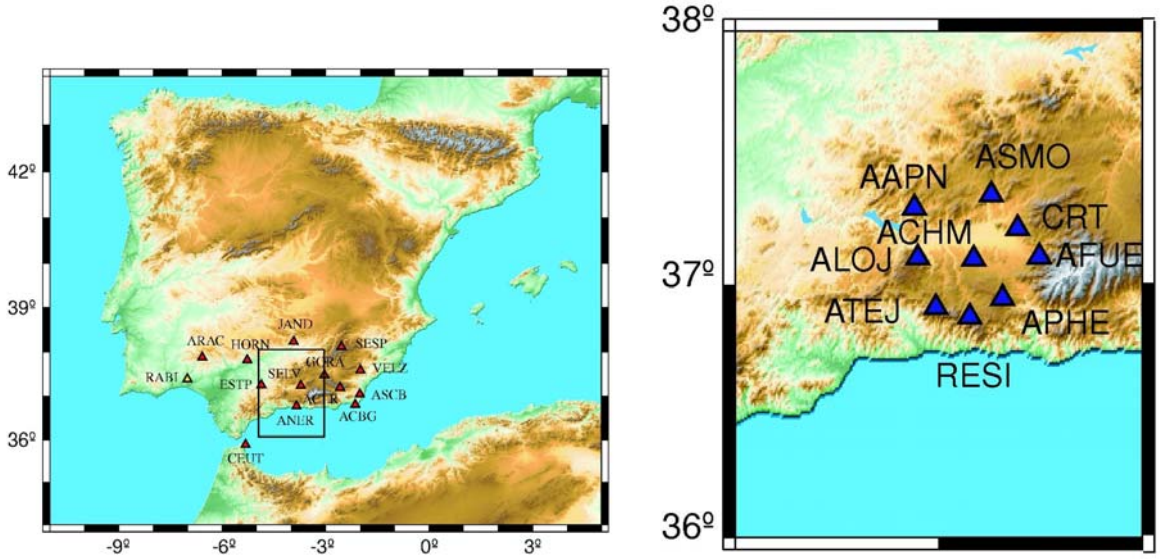


Figura 5.2. *Mapa de estaciones sísmicas del IAG. Los triángulos azules representan las estaciones de corto periodo que conforman la red de microsismicidad. Los rojos son las estaciones de banda ancha. Algunas de ellas se complementan con acelerómetros que forman la red de movimiento fuerte. A la derecha se muestra, ampliada, el área correspondiente a la Depresión de Granada, que como puede verse se cubre con una mayor densidad de estaciones. La central de registro está situada en el Observatorio de Cartuja, cuya localización coincide con la de la estación CRT (de Morales et al., 2007).*

Dentro de esta nueva etapa del Observatorio de Cartuja tuvo una gran importancia la serie sísmica que sacudió Granada en el año 1979, con numerosos terremotos sentidos y que originó una gran alarma social (Vidal *et al.*, 1981, Vidal y De Miguel, 1983). Esta serie puso de manifiesto las carencias en instrumentación del momento. La necesidad de suministrar información precisa y actualizada después de cada evento sentido ponía en evidencia el sistema utilizado hasta entonces, que requería realizar varios procesos en las bandas fotográficas (revelado, fijado y secado) antes de poder tomar medidas en los registros para hacer una estimación de las características de la señal. Así, se hizo patente la necesidad de contar con un sistema de registro visual en tiempo real.

La alarma social creada por la serie sísmica del 79 provocó, entre otras cosas, la concienciación de las autoridades en la necesidad de contar con personal y equipos adecuados, lo cual derivó en un aumento de los presupuestos asignados al Observatorio de Cartuja, que hasta el momento habían sido prácticamente nulos. Esta circunstancia permitió acometer el diseño y desarrollo de las primeras estaciones de la red sísmica regional, además de adecuar distintas instalaciones del observatorio para ampliar y acondicionar las zonas de trabajo. En 1981 se finaliza el prototipo de estación sísmica telemétrica, que se instala en la

[42] www.ugr.es/~iag/red.html

Sierra de Albuñuelas bajo la denominación de Pico Herrero (PHE). Durante los meses siguientes se construyen otras cuatro estaciones, y a finales de 1982 ya están instaladas y operativas, además de la citada PHE, las de SMO, LOJ y TEJ, que conforman la primera red sísmica radiotelemétrica de microsismicidad de la Península Ibérica (Alguacil, 1986).

A principios de los noventa también se comienza a trabajar en otra línea de desarrollo instrumental, la de la sismicidad volcánica. Con ocasión de un proyecto europeo sobre volcanes-laboratorio, que incluye el Teide, se decide utilizar antenas sísmicas, para cuyo diseño y desarrollo se colabora con el grupo de volcanología del Museo Nacional de Ciencias Naturales (profesor Ramón Ortiz). Los dispositivos resultantes se utilizan profusamente en los siguientes años, sobre todo en campañas de campo en distintas áreas volcánicas, pero también para el seguimiento de crisis sísmicas de origen tectónico (Carmona *et al.*, 2002). Estos módulos de *array* (a los que a partir de ahora se denominará 'módulos de dieciséis bits') tienen gran importancia en el marco de este trabajo, ya que pueden considerarse como los precursores de las antenas portátiles que se presentan en este capítulo. Los módulos de dieciséis bits se describirán con detalle en las próximas secciones como introducción a los dispositivos que se han diseñado para sustituirlos.

Desde que se decidió adoptar la sismología volcánica como una de las principales líneas de investigación y desarrollo instrumental del IAG ésta no se ha abandonado. Especial atención se ha prestado al seguimiento de la actividad en la Isla Decepción, en la Antártida, en la que se han llevado a cabo campañas de recogida de datos con continuidad. Aparte del seguimiento continuo de la actividad en esta isla, se realizaron campañas puntuales en otras áreas activas como los volcanes Strómboli, Etna y Vesubio, todos en Italia, el Volcán de Fuego de Colima, en México, el volcán Copahue, en Argentina, las islas Azores o el volcán Teide, en las Canarias.

Hasta el año 2002 el grueso de la instrumentación utilizada en las campañas de campo lo constituían los módulos de dieciséis bits. A partir de entonces las nuevas antenas tomaron el relevo, constituyendo, por ejemplo, el principal equipamiento instrumental en el proyecto TOMODEC[43]. El objetivo de este proyecto era la realización de una tomografía de velocidad de la Isla Decepción, para lo cual se utilizaron doce de los nuevos módulos operando de forma sincronizada. Además de los módulos de *array* (antiguos o nuevos), en diversas campañas se ha utilizado otra instrumentación desarrollada en el IAG, como sistemas de adquisición de datos de banda ancha con grabación local o estaciones telemétricas. Una descripción general de estos sistemas, así como de la historia del desarrollo de la instrumentación propia en lo que aquí hemos denominado 'etapa moderna' del Observatorio de Cartuja puede consultarse en Alguacil, 2003.

5.1.2. Los módulos de array de dieciséis bits

5.1.2.1. Hardware

Los módulos de *array* de dieciséis bits, precursores de los sistemas que se describen en este capítulo, se desarrollaron a principios de los años noventa para su uso en campañas sísmicas en entornos volcánicos. La figura 5.3.a muestra un diagrama de bloques de estos módulos, cuyo núcleo consiste en un sistema de adquisición de datos de ocho canales y 16 bits, basado en el conversor A/D CS5016 (Cirrus Logic, 2005). La figura 5.3.b muestra dos imágenes de uno de los sistemas. Como puede verse, en el módulo central se monta toda la electrónica del

[43] TOMODEC: TOMOgrafía sísmica de alta resolución de la isla DECepción. Proy. TOMODEC-REN2001-3833 (www.utm.csic.es/proyecto.asp?id=%7BCD8D2A9E-387B-44B0-84C5-C05FFF9743A4%7D).

sistema de adquisición, que consta de una tarjeta de conversión A/D, dos placas para el acondicionamiento de las señales analógicas antes de su digitalización y una tarjeta de interfaz para el receptor GPS. El acondicionamiento de la señal consta de preamplificado y filtrado *antialiasing*. El control de todo el proceso corre a cargo de un PC portátil, que se comunica con el módulo de adquisición a través del puerto paralelo. La temporización del proceso de muestreo se lleva a cabo mediante un receptor GPS, que se comunica con el PC por medio de una tarjeta de interfaz RS-232. Los módulos pueden operar indistintamente con geófonos MARK L4-C de 1 Hz o con pequeños geófonos de 4.5 Hz, con la respuesta extendida electrónicamente hasta 1s (figura 5.4). Estos últimos tienen una respuesta prácticamente equivalente a la de los MARK L4-C con un sensible ahorro en coste y peso, lo cual supone una ventaja importante en campañas de campo. Para más detalles sobre la técnica y circuito de ecualización pueden consultarse Havskov y Alguacil, 2004, pp. 49-51, u Ortiz *et al.*, 2001, pp. 246-254.

A continuación se describen brevemente los principales componentes de estos módulos:

Interfaz GPS.- La figura 5.5 muestra el esquema eléctrico del circuito de interfaz entre el receptor GPS y el PC de control. El modelo de tarjeta receptora utilizado originalmente era el *SVeeSix-CM2,* de la marca Trimble. Con el paso del tiempo esta familia de receptores ha ido dando otros modelos compatibles de mejores características, como el *CM3*, *Ace I, II* y *III* o *Lassen SK II,* que también pueden utilizarse en estos sistemas con pocos o ningún cambio.

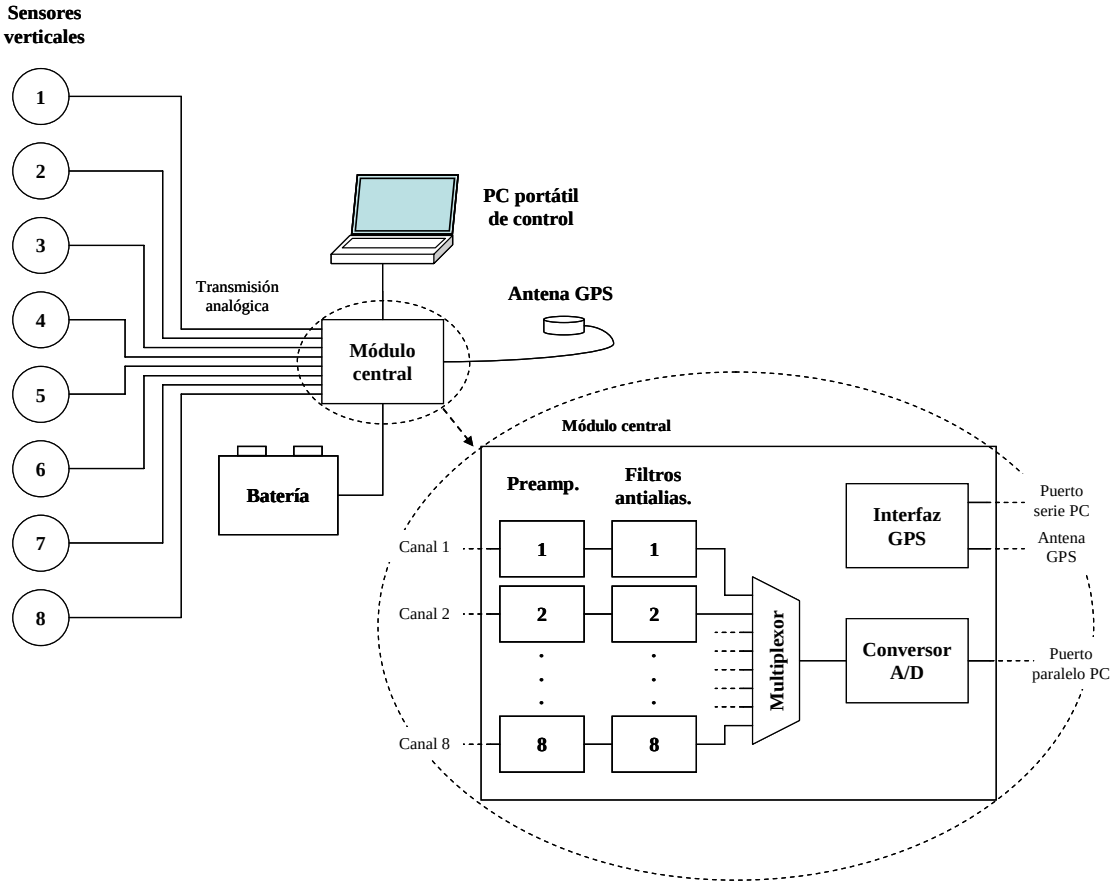
La comunicación con el receptor GPS se debe realizar utilizando el estándar RS-232. Los niveles lógicos del puerto serie del receptor son TTL, por lo que deben convertirse a los permitidos por el estándar RS-232. Para ello se emplea un transceptor modelo MAX232 (U2 en la figura 5.5). Además de la conversión de niveles, en la tarjeta de interfaz se genera una salida de pulso de segundo sincronizado con los pulsos del GPS pero de mayor longitud que éstos, para lo cual se utiliza un monostable 4538 (U3).

En la versión de los módulos de dieciséis bits que se presenta aquí no existe una tarjeta específica de alimentación, sino que en cada tarjeta se implementan los integrados necesarios para alimentar el correspondiente circuito a partir de la tensión de la batería externa. En la tarjeta de interfaz GPS se utilizó una fuente conmutada modelo LM2575-5 (U1). Tanto en este caso como en el resto de las tarjetas se incluye, a la entrada del circuito de alimentación, un diodo en serie para proteger la circuitería en caso de conexión invertida de la batería. Aparte del circuito de alimentación propiamente dicho, en la tarjeta de interfaz GPS se utiliza una batería de litio (BAT1 en el esquema) para la alimentación del reloj de tiempo real del receptor GPS cuando no hay alimentación externa.

Tarjeta de acondicionamiento de señal.- La figura 5.6 muestra el esquema eléctrico del circuito de acondicionamiento de señal de cada uno de los ocho canales analógicos. El circuito cumple dos funciones, la preamplificación de la señal analógica y el filtrado *antialiasing*.

La preamplificación se realiza en el primer bloque, que a su vez consta de dos etapas (integrados IC1A e IC1B). La primera es una etapa amplificadora diferencial de ganancia unidad, en la que se han incorporado filtros pasivos paso baja en las entradas analógicas para el filtrado de radiofrecuencias. La existencia en el entorno de despliegue de los *arrays* de señales de comunicaciones de radio es una de las principales fuentes potenciales de ruido en este tipo de dispositivos, ya que los cables de los sensores actúan como antenas para esas frecuencias.

a)



b)

Figura 5.3. *Diagrama de bloques de los módulos de dieciséis bits (a). Los ocho sensores verticales pueden sustituirse por cualquier combinación de sensores de una o tres componentes, hasta ocupar el máximo de ocho canales que es capaz de gestionar cada módulo. En b) se muestran dos imágenes de uno de los módulos. A la derecha, el módulo cerrado, en el que se aprecian las fichas de conexión de los ocho canales de adquisición de datos y de la batería (más dos fichas de reserva). Detrás del módulo puede verse la antena GPS. No se muestra el PC portátil que controla el sistema. A la izquierda puede verse el módulo abierto. En primer término aparecen los conectores de cable serie, paralelo y de la antena GPS. Además se distinguen, dentro de la caja, la tarjeta de interfaz GPS (a la izquierda) y una de las placas de acondicionamiento de señal analógica, Debajo de ésta quedan la otra tarjeta de acondicionamiento y la tarjeta de conversión A/D.*

La segunda etapa del preamplificador (integrado IC1B) es un amplificador de ganancia variable. La ganancia se controla mediante una red de resistencias R-2R a la salida del operacional. Las distintas resistencias de la red se habilitan o deshabilitan mediante el conmutador SW1, con lo que se consiguen ganancias entre 1 y 128.

El segundo bloque del circuito de acondicionamiento de señal es un filtro *antialiasing*, cuya frecuencia de corte es de 49.1 Hz. Es un filtro paso baja de Butterworth y de octavo orden, que como puede verse en la figura 5.6 se ha implementado mediante cuatro etapas de segundo orden en configuración de Sallen y Key[44], con divisores de tensión en el lazo de realimentación para variar la ganancia en banda de paso. La ganancia total, producto de la ganancia de las cuatro etapas, es de 6.84 con los valores de componentes indicados.

Con el fin de mantener la relación señal-ruido lo mayor posible y un consumo reducido, se utilizaron amplificadores operacionales de bajo ruido y bajo consumo. Para minimizar el número de integrados y ahorrar espacio en las placas se utilizaron dispositivos dobles (TL072) en los preamplificadores y cuádruples (TL064) en el filtro.

Es necesario uno de estos circuitos de acondicionamiento de señal para cada uno de los ocho canales analógicos. Con el objeto de no aumentar excesivamente el tamaño de las placas la implementación física se hizo en tarjetas de cuatro circuitos, por lo que cada módulo necesita dos tarjetas (ver figura 5.3.b).

Para poder trabajar con señales bipolares la tarjeta de acondicionamiento de señal opera con alimentación simétrica. La tensión positiva (+8V) se obtiene mediante un regulador lineal 7808, tomando como entrada la tensión de la batería. La tensión negativa de alimentación (-8V) se consigue mediante un inversor 7660, tomando como entrada la tensión de salida del 7808.

Tarjeta de conversión A/D.- El conversor A/D alrededor del cual se construye esta tarjeta es el CS5016, de la empresa *Cirrus Logic*[45], cuyas principales características se muestran a continuación (Cirrus Logic, 2005):

- Resolución de dieciséis bits reales, sin pérdida de códigos.
- Utiliza la técnica de aproximaciones sucesivas.
- Posibilidad de seleccionar el tipo de salida de datos: en serie, en paralelo (16 bits) o mixta (dos lecturas de ocho bits).
- Baja distorsión.
- Bajo consumo (150 mW).
- Sistema de autocalibración, que permite mantener la precisión en el tiempo y ante variaciones de las condiciones ambientales.
- Admite entradas unipolares o bipolares. El tipo de entrada es seleccionable por *hardware.*

El tiempo de conversión para este modelo es de 16.25 μs, lo cual redunda en la posibilidad de muestrear a frecuencias de hasta 50 kmps. Esta característica, unida al hecho de que la técnica de muestreo que utiliza no exige un seguimiento continuo de la señal de entrada (como ocurre en los conversores sigma-delta), permite utilizar un solo conversor para los ocho canales analógicos a una frecuencia de 200 mps. De esta forma se consigue un considerable ahorro en el coste y una notable simplificación en el circuito. La figura 5.7 muestra un esquema eléctrico del circuito usado en los módulos de dieciséis bits, en el que se utiliza un

[44] Existe abundante información en Internet sobre este tipo de filtros. Ver, por ejemplo, Texas Instruments 1999, o la página www.ecircuitcenter.com/Circuits/opsalkey1/opsalkey1.htm.
[45] www.cirrus.com/en/

multiplexor analógico de ocho a uno, modelo 4051 (IC6 en la figura) para conmutar entre las entradas analógicas de los distintos canales. El control del multiplexor se realiza a través de tres líneas del puerto paralelo.

Como puede verse en la figura 5.7, el control del proceso de muestreo se realiza a través del puerto paralelo, adaptándose las señales mediante búffers inversores 4049. La limitación del número de señales de control y datos del puerto paralelo (especialmente en los modelos de PC originalmente utilizados para el control de estos módulos, que sólo permitían usar el puerto en modo SPP[46] (*Standard Parallel Port*) se salva realizando la lectura de los datos mediante multiplexación temporal. Es decir, en lugar de leer en paralelo los dos bytes de datos, la lectura se realiza a través del registro correspondiente al byte menos significativo. A su vez éste se multiplexa usando un integrado específico (4519, IC4 en la figura 5.7), controlado por una de las líneas del puerto paralelo, con lo cual la lectura del dato completo se realizará en cuatro pasos, correspondientes cada uno a un *nibble*.

Al igual que en la tarjeta de acondicionamiento de señal, en ésta la alimentación debe ser simétrica para poder trabajar con señales bipolares. En este caso se utiliza un regulador lineal 7805 para obtener la tensión positiva para las etapas analógica y digital (+5V), y el mismo integrado que en la tarjeta de acondicionamiento (7660) para la tensión negativa (-5V).

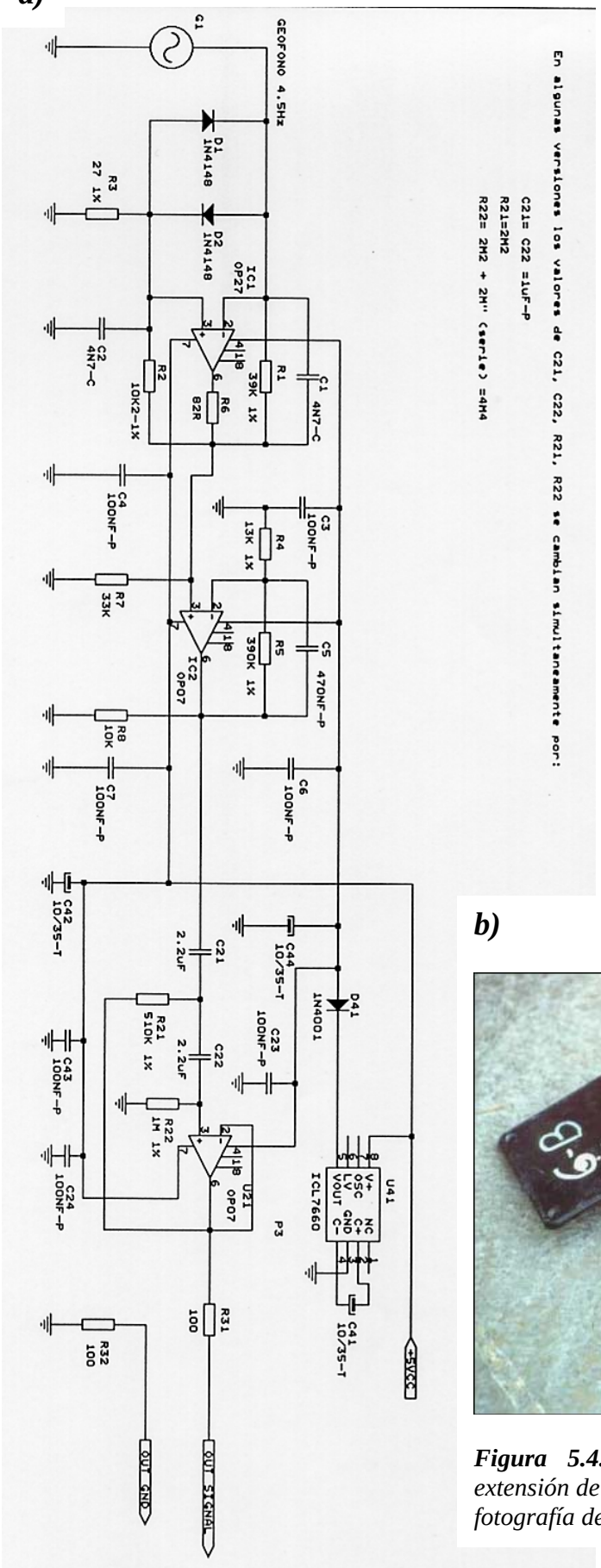
5.1.2.2. Operación del sistema

Una de las mayores diferencias entre estos módulos y los que se han presentado en los capítulos anteriores, así como los que se describirán más adelante en éste, es que el control no es distribuido. Tanto en las antenas del Gran Sasso como en la del Vesubio y en las nuevas antenas portátiles del IAG, las distintas tareas que debe desempeñar cada sistema se dividen en varias secciones cuyo control se asigna a algún tipo de dispositivo (microcontroladores PIC o tarjetas de PC) con determinada capacidad de procesamiento de la información. Las tareas desempeñadas por cada módulo y los programas diseñados para realizarlas son más o menos complejos dependiendo de las características de cada sistema y de las soluciones elegidas en la etapa de diseño. En el caso de los antiguos módulos de dieciséis bits, sin embargo, el único subsistema con capacidad de proceso es el PC portátil, que debe soportar por tanto toda la carga de programación. Por otra parte, el sistema operativo que utilizaban los primeros PCs encargados del control de los módulos era el MS-DOS, por lo que el programa se diseñó para operar en ese entorno, realizando todas las tareas (lectura de los parámetros de inicialización, programación de los parámetros del conversor A/D, lectura de los datos, grabación en disco, comunicación con el receptor GPS,…) de forma secuencial o por interrupciones en un único programa (REDSIS.EXE). El único proceso que queda fuera del mismo es la interacción con el usuario para la modificación de los parámetros de operación del sistema, tarea que se realiza mediante otro programa (CONFIRED.EXE). Éste debe ejecutarse antes de lanzar el programa de adquisición en caso de que se desee cambiar alguno de los parámetros.

[46] Puede consultarse información sobre los distintos modos de operación del puerto paralelo en las siguientes direcciones:

www.fapo.com/1284int.htm
es.wikipedia.org/wiki/IEEE_1284

a)



b)

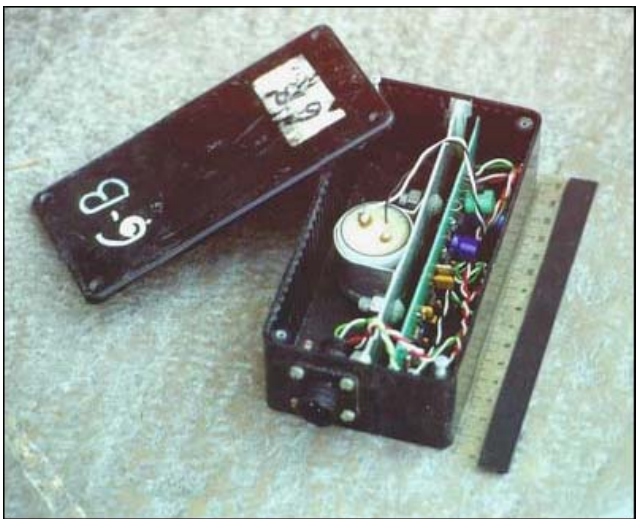


Figura 5.4. *Esquema eléctrico del circuito de extensión de respuesta de los sensores de 4.5 Hz (a) y fotografía de uno de los sensores (b).*

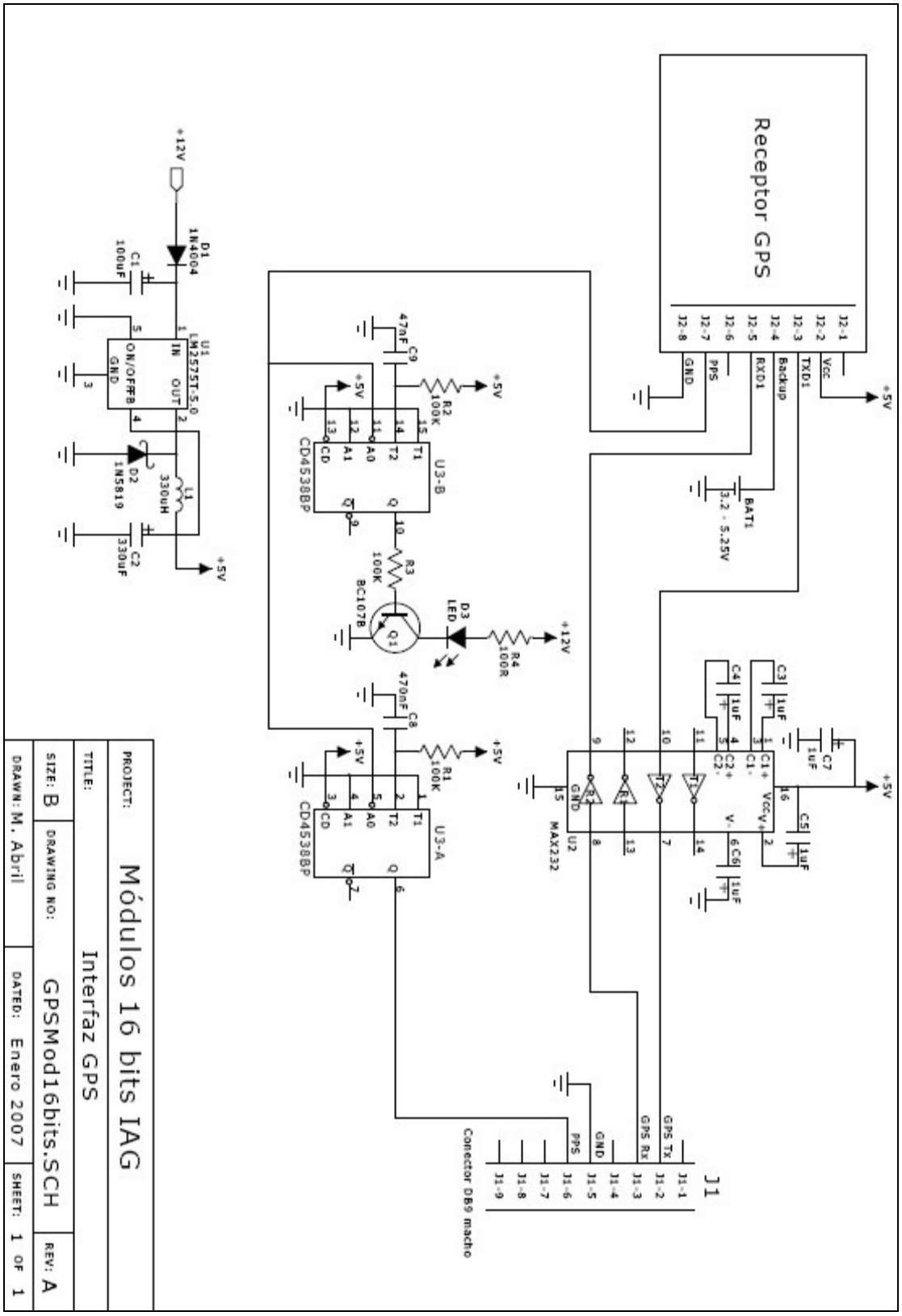


Figura 5.5. *Esquema eléctrico del circuito de interfaz GPS de los módulos de dieciséis bits.*

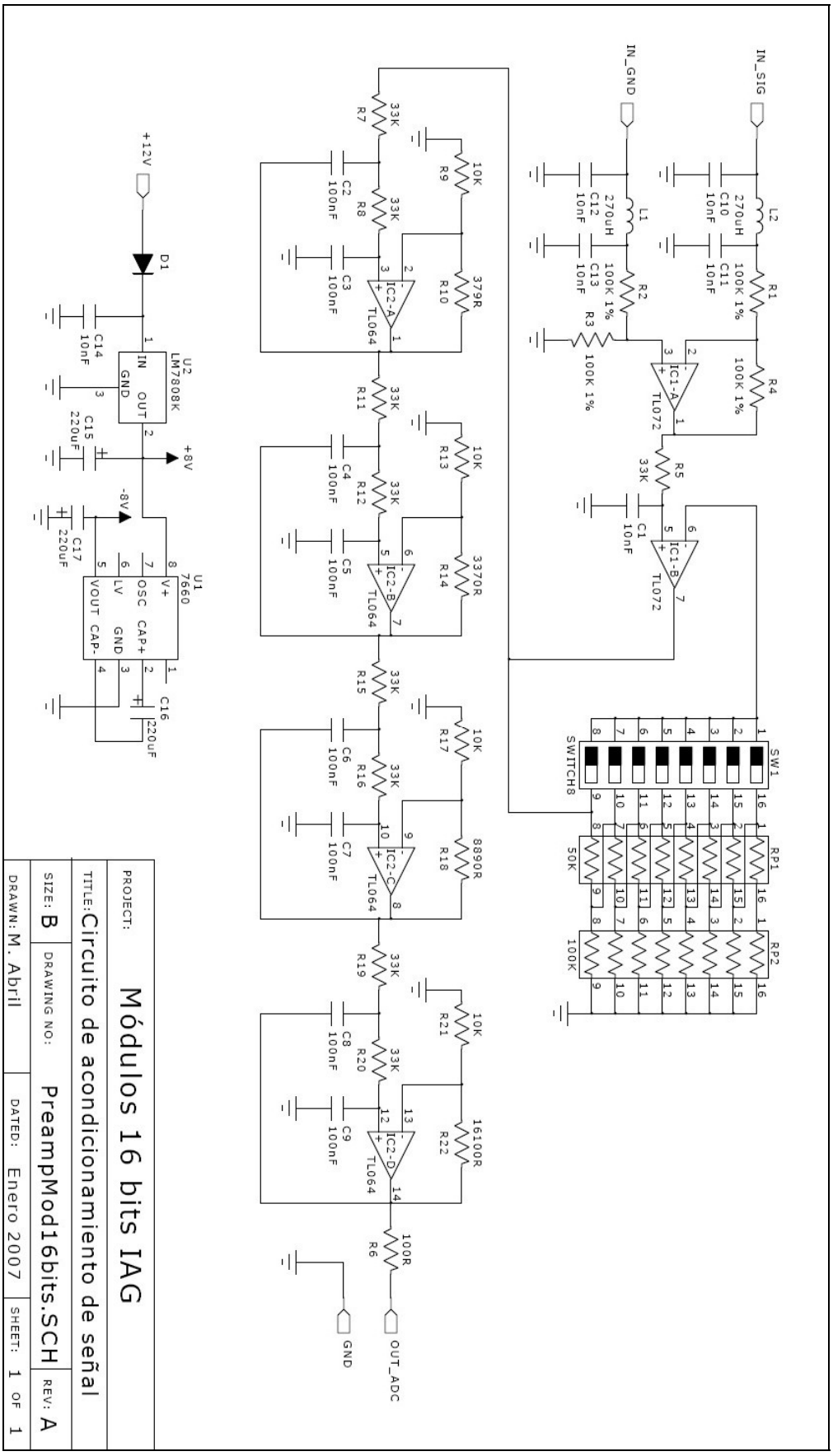


Figura 5.6. *Esquema eléctrico del circuito de acondicionamiento de señal (preamplificadores y filtros antialiasing) de los módulos de dieciséis bits.*

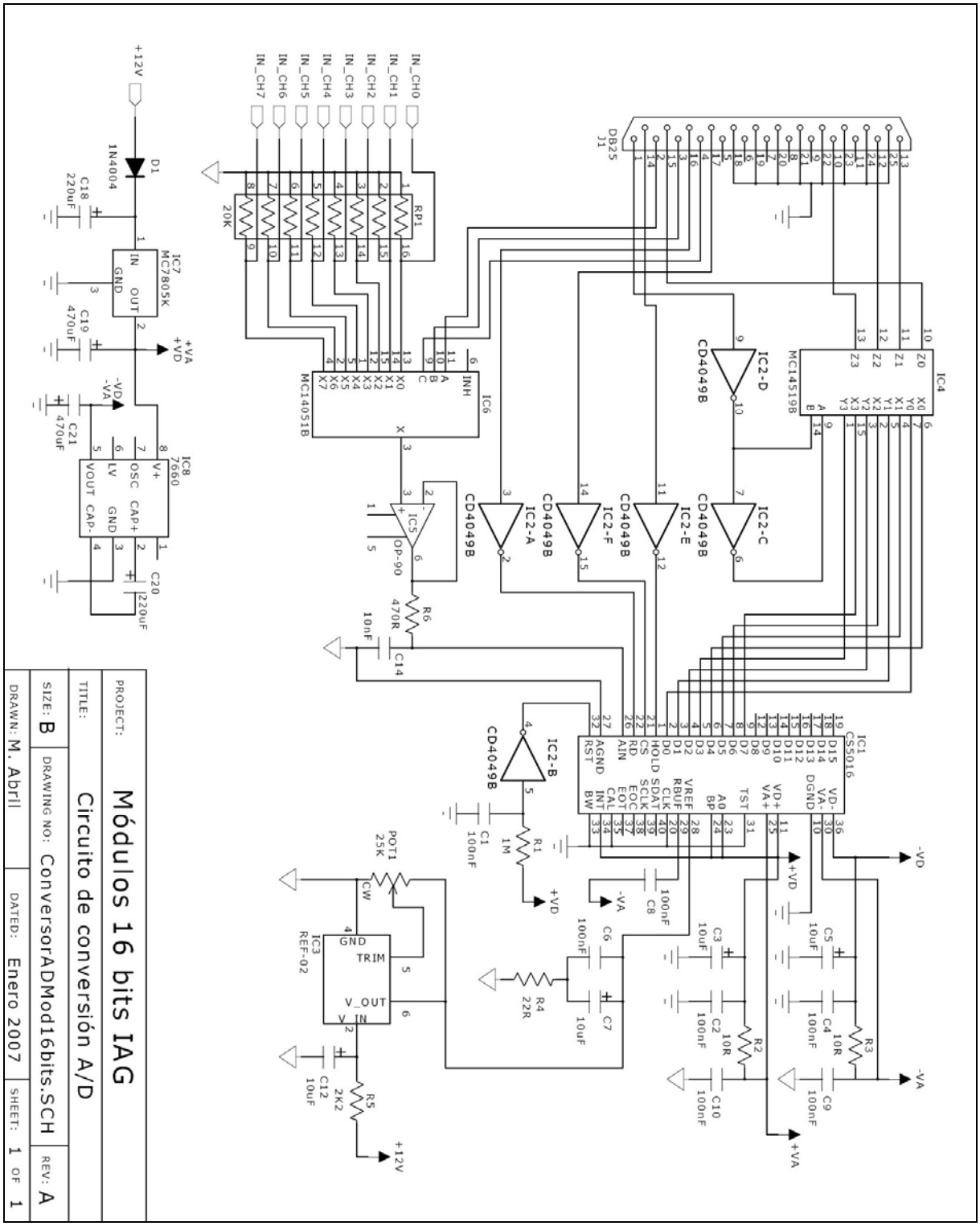


Figura 5.7. *Esquema eléctrico de la placa del conversor A/D CS5016.*

Así, desde el punto de vista del usuario, la operación de los módulos debe iniciarse ejecutando el programa CONFIRED, cuya interfaz de usuario se muestra en la figura 5.8. Como puede verse, la interfaz consiste en una pantalla MS-DOS en la que se presentan los valores y/o nombres de todos los parámetros de operación. Es posible moverse a través de los diferentes campos mediante las teclas de cursor y modificar cualquiera de ellos introduciendo mediante el teclado hasta tres caracteres alfanuméricos por campo (salvo el relativo al directorio de grabación de eventos, que admite más para permitir la introducción de la ruta completa). Una vez modificados los campos y confirmada la salida, el programa CONFIRED graba los nuevos valores en el fichero de configuración REDSIS.CFG, al que el programa de adquisición REDSIS.EXE accederá al iniciar la operación para conocer los parámetros de operación. Dado que el fichero de configuración REDSIS.CFG se crea en formato ASCII, también es posible modificar los parámetros usando cualquier editor de textos. Los campos modificables del fichero REDSIS.CFG están relacionados con tres aspectos: parámetros de operación del sistema, constantes del algoritmo de disparo y nomenclatura de los canales y ficheros de datos. Los distintos parámetros no se describen aquí por brevedad, pero pueden consultarse en el anexo III.C.1.a, incluido en el CD adjunto.

Además de la detección de eventos mediante el algoritmo de disparo STA/LTA (Lee y Stewart, 1981), es posible realizar registros a intervalos regulares. Esta modalidad, denominada registro por ventana programable, se configura a través de una serie de parámetros que no se encuentran en el fichero genérico de configuración REDSIS.CFG, sino en un fichero específico en formato ASCII denominado DISPARO.TXT. Para programar los registros por ventana dicho archivo debe encontrarse en el mismo directorio que el programa de adquisición REDSIS en el momento de lanzar éste. En caso contrario no se programarán ventanas y en pantalla aparecerá el mensaje ‘Sin Ventana’. La información que debe incluirse en el fichero DISPARO.TXT consiste en cinco líneas:

ESTACION SISMICA. CONFIGURACIÓN C.S.I.C. Spain

numero de canales	8	Mascara Canal 1	0
mps	200	Mascara Canal 2	0
PreEvento	16	Mascara Canal 3	0
PostEvento	32	Mascara Canal 4	0
Estaciones para disparo	3	Mascara Canal 5	1
TSta	2	Mascara Canal 6	1
TLta	16	Mascara Canal 7	0
KDispa	1	Mascara Canal 8	1
Kfiltro	1	Nombre Canal 1	-Z-
Algoritmo	1	Nombre Canal 2	N-S
Segundos Registro	64	Nombre Canal 3	E-W
Sincronismo Reloj	1	Nombre Canal 4	ZN-
Nombre Archivo	1	Nombre Canal 5	ZS-
		Nombre Canal 6	ZW-
		Nombre Canal 7	ZE-
		Nombre Canal 8	ZSW
Actualizado 09-12-1994 12:30:30		Nombre Estación Registro	A
		Directorio SALIDA	C:\SISMOS\

Figura 5.8. *Interfaz de usuario del programa de configuración de los módulos de dieciséis bits CONFIRED.EXE. Los valores de los parámetros corresponden a los utilizados en uno de los equipos de la campaña del Teide de 1994*

- Hora de inicio del registro.
- Minuto de inicio del registro.
- Segundo de inicio del registro.
- Intervalo entre registros, en segundos.
- Número de registros que se realizarán.

Una vez configurado el sistema a través del programa CONFIRED (o mediante la edición directa del fichero de configuración REDSIS.CFG) y programados, en su caso, los registros por ventana, debe lanzarse el programa de adquisición REDSIS.EXE desde la línea de comandos de MS-DOS. La interfaz de usuario del programa REDSIS tiene el aspecto que se muestra en la figura 5.9. Como puede verse, la pantalla queda dividida en tres partes. En la inferior se muestra, además de algunos de los parámetros de operación descritos en el anexo III.C.1.a, información adicional como la fecha y hora de inicio del programa, el número de eventos detectados y el espacio libre en disco. El área intermedia se reserva para la visualización gráfica en tiempo real de la señal correspondiente al canal seleccionado, mientras que en la parte superior se muestran la identificación del equipo (en este caso ARRAY A), la fuente de reloj seleccionada (PC, GPS o marcas externas, según se describe en el parámetro de configuración 'Sincronismo Reloj', en el anexo III.C.1.a), la hora y fecha y la fase actual del programa. Hay siete posibles fases o estados (iniciación, sincronismo, detección, disparo, postevento, desbordamiento y transferencia), que se describen en el anexo III.C.1.b.

Los sistemas están preparados para operar continuamente mientras la tensión de la batería lo permita. Los paneles solares o aerogeneradores para proporcionar alimentación continua no evitarían las visitas periódicas para acceder a los datos registrados y realizar otras operaciones de mantenimiento, por lo que no se suelen instalar en campañas cortas. Los intervalos de tiempo entre las visitas de mantenimiento varían en función de la capacidad de las baterías. Teniendo en cuenta que durante las campañas de adquisición de datos conviene llevar un control lo más actualizado posible de la actividad y eso implica acceder a los datos con frecuencia, en la práctica se suelen utilizar baterías de coche, de capacidad media (en torno a los 75Ah), que resultan relativamente cómodas de transportar y precisan de poco tiempo de carga.

El acceso a los datos registrados se puede realizar de dos formas. La primera consiste en cambiar el PC portátil con los ficheros de datos por otro con el directorio de datos vacío. Esta opción resulta muy cómoda, ya que limita el tiempo de recogida de datos al necesario para desconectar los distintos subsistemas del PC usado y reconectarlos al nuevo. Se ahorra, por tanto, mucho tiempo de trabajo en campo, lo cual resulta recomendable en cualquier caso y fundamental si se trata de un entorno peligroso. Sin embargo, esta opción presenta el inconveniente de que hay que contar con al menos dos portátiles por cada uno de los módulos que se tengan operativos. Además del consiguiente aumento de coste, ello conlleva el riesgo de utilizar por equivocación configuraciones distintas para el mismo array o intercambiar los PCs de los distintos módulos, con la consiguiente desorganización de los datos.

La segunda opción consiste en transferir los datos grabados por el PC portátil de control a otro portátil. La conexión entre ambos ordenadores se realiza a través de puerto serie o paralelo, utilizando el programa Laplink bajo MS-DOS. El principal inconveniente de esta solución es la baja velocidad de transferencia, especialmente si se utiliza la conexión a través del puerto serie, que implica un tiempo considerable en el proceso de volcado de datos. Como es lógico este tiempo aumenta con el número de eventos grabados en disco, que a su vez depende de la actividad, los parámetros programados para el algoritmo de disparo y el periodo durante el que el sistema ha estado funcionando desde el último volcado de datos

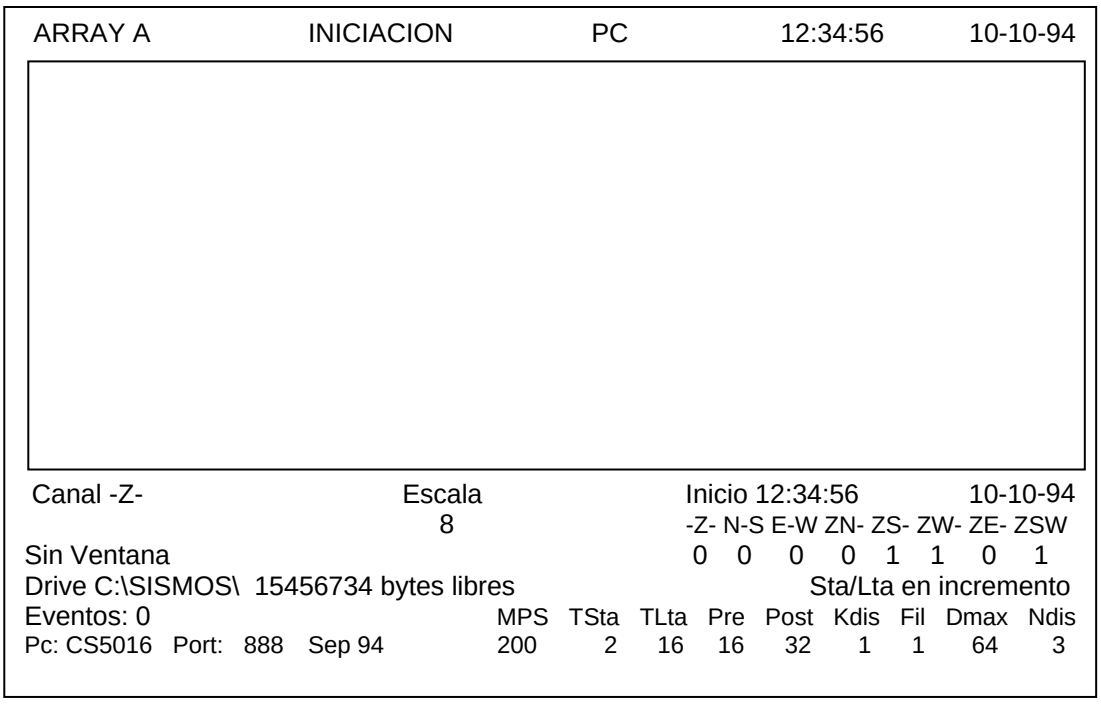


Figura 5.9. *Interfaz de usuario del programa de adquisición REDSIS.EXE.*

5.2. Objetivos y especificaciones técnicas

El objetivo principal que se pretendía cubrir con las antenas que se presentan en este capítulo era la sustitución de los antiguos módulos de dieciséis bits por otros con mejores prestaciones pero que siguieran la misma filosofía, que ya había sido llevada a la práctica con resultados muy satisfactorios en numerosas campañas de campo.

Esta filosofía se basa en utilizar módulos con pocos canales de adquisición diseñados para operar independientemente. Como se ha descrito en las secciones anteriores, cada una de las antenas gestiona un receptor GPS, de forma que, pese a su capacidad de operar como módulos independientes, la base de tiempo común proporcionada por los receptores GPS permite realizar análisis conjuntos de los datos de todas las antenas.

Por otra parte, otro de los aspectos que pueden considerarse como inherentes a la filosofía de los antiguos módulos de dieciséis bits es la disposición de la electrónica, que se concentra en un módulo central. En dichos módulos el encargado de muestrear la señal de todos los canales era un único conversor A/D. Como es lógico, éste estaba situado en el módulo central, por lo que la transmisión de las señales de los distintos sensores era analógica. En las nuevas antenas se utilizarán conversores sigma-delta, que realizan un seguimiento continuo de la señal y por tanto exigen un conversor por cada canal. Pese a ello se mantendrá la misma disposición, situando todos los conversores A/D en el módulo central y conservando la transmisión analógica de las señales de los sensores.

También se mantiene en estas antenas la filosofía de los módulos de dieciséis bits en lo referente a los cables y conectores: para la mayor parte de las conexiones se utilizarán conectores de bajo coste y fácilmente manipulables en campo, en lugar de conectores de tipo militar. Así se aumenta la portabilidad de los dispositivos, ya que no es necesario transportar los cables si es posible conseguirlos en alguna localidad próxima al área de despliegue. Por supuesto, en el caso de campañas en entornos remotos en los que resulta difícil conseguir

cables adecuados, o bien en situaciones de emergencia en las que no es posible hacerlo en un tiempo razonable, los cables se pueden transportar junto con el resto del equipo.

Las principales ventajas de esta filosofía son:

- La capacidad de los sistemas de operar independientemente proporciona una gran flexibilidad al conjunto. En función de la aplicación es posible disponer los distintos dispositivos en un área reducida para obtener una única antena de alta densidad, o distribuirlos en un área extensa para, por ejemplo, rodear la zona donde se origina la señal sísmica y localizar la fuente mediante la técnica del cruce de rayos.
- Aumenta la tolerancia a fallos del conjunto, ya que el malfuncionamiento de un sistema no afecta a los demás. En un sistema único como el del Vesubio, el fallo en alguno de los subsistemas críticos (módulo SerPar, módulo de memoria, PC de control,…) implicaría la pérdida total de los datos de todo el dispositivo.
- El hecho de que la electrónica esté centralizada en un solo módulo facilita el transporte e instalación.
- Como se verá en breve, el menor número de canales que debe gestionar cada antena permite simplificar notablemente el diseño.

Sin embargo, esta filosofía también presenta inconvenientes frente a la de un sistema único con mayor número de canales, como la seguida para el diseño de la antena del Vesubio:

- El consumo total aumenta, ya que los elementos que más consumen del diseño (PC industrial y receptor GPS) se multiplican. Si en el Vesubio sólo eran necesarios un PC y un GPS, ahora hay que contar con uno por cada sistema. Sin embargo, aunque el consumo global aumenta, también es cierto que el consumo de cada sistema individual es menor que el de la antena del Vesubio. La consecuencia práctica es que para una configuración con los mismos canales e igual autonomía que en la antena del Vesubio hará falta más potencia total, pero al utilizarse más baterías (una por sistema) éstas podrán ser de menor capacidad (y por tanto, de menor peso).
- Del mismo modo, el precio total aumenta debido al mayor número de elementos que antes eran únicos. Aparte de los PCs y los receptores GPS, hay que contar con las maletas para la electrónica de los módulos centrales y otros elementos relacionados con el montaje. No obstante, este aumento de precio queda parcialmente compensado por la reducción en otros elementos de montaje y electrónica como consecuencia de concentrar la instrumentación de cada sistema en un mismo contenedor. Así, por ejemplo, se reduce el número de circuitos PLL respecto al utilizado en la antena del Vesubio (en la que era necesario un PLL por cada módulo de adquisición) y se eliminan las cajas para los módulos de adquisición.
- La transmisión analógica de las señales impuesta al optar por situar los conversores A/D en el módulo central implica una disminución en la relación señal/ruido (SNR), lo cual puede imponer un límite a la longitud de los cables y obliga a utilizar amplificadores en determinados sensores y cable apantallado en entornos ruidosos.

5.2.1. Consideraciones de diseño

La idea de sustituir los módulos de dieciséis bits por otros basados en conversores sigma-delta surgió mientras el proyecto de la antena del Vesubio estaba aún vigente. Se decidió buscar un sistema que sirviera de sustituto a los antiguos módulos con una mejora notable de sus características, pero aprovechando la experiencia y resultados obtenidos en el sistema del Vesubio para que el tiempo de diseño y desarrollo fuera lo menor posible. El planteamiento de sistemas independientes con pocos canales permitía prescindir de algunos de los módulos específicos de la antena del Vesubio, como la tarjeta SerPar y las tarjetas del módulo de memoria. Por otra parte, al concentrar todas las tarjetas de conversión A/D en el módulo central era posible reducir el número de tarjetas PLL (bastaba con una que gestionase la señal de reloj para todos los conversores A/D). Por tanto, el diseño del sistema que nos ocupa podía considerarse como una simplificación o subconjunto de la antena del Vesubio, puesto que prácticamente todos los circuitos necesarios en éste ya estaban diseñados en aquél, pero no todos los circuitos diseñados en aquél eran necesarios en éste.

Los nuevos sistemas debían solventar las principales limitaciones de los antiguos módulos de dieciséis bits:

- *Número de canales.*- Pese a que interesaba que el número de canales por cada módulo se mantuviera reducido para poder prescindir de módulos *hardware* específicos, convenía que éste fuera algo mayor que el de los antiguos sistemas. Los ocho canales que éstos gestionaban hacían que en ocasiones la distribución espacial de los sensores asociados a un solo módulo de array fuera demasiado poco densa, en especial si se utilizaban estaciones de tres componentes. Por tanto, se decidió aumentar el número de canales hasta doce.
- *Resolución.*- El conversor A/D CS5016 utilizado en los antiguos módulos tenía una resolución nominal de dieciséis bits. Como ya se ha apuntado antes, el alto margen dinámico de las señales sísmicas, y especialmente de aquellas presentes en entornos volcánicos, hace recomendable que la resolución de los conversores A/D sea lo mayor posible para permitir el registro de las señales en todo su rango de variación. Igual que en los casos ya descritos de las antenas del Gran Sasso y el Vesubio, se decidió utilizar el conversor AD7710.
- *Muestreo simultáneo.*- Como se ha explicado en la introducción, los antiguos módulos de dieciséis bits utilizaban un solo conversor A/D para muestrear los ocho canales de adquisición, lo cual exigía realizar un multiplexado en el tiempo y hacía imposible muestrear simultáneamente todas las señales. La corrección del retraso de las muestras de cada canal debía hacerse por *software* mediante interpolación, durante el procesado de los datos. Sin embargo los AD7710, como todos los conversores que utilizan la técnica sigma-delta, realizan un seguimiento continuo de la señal. Por tanto resulta obligatorio utilizar un conversor por cada canal, lo cual permite obtener simultáneamente las muestras de todas las señales.
- *Registro continuo.*- Los programas de los antiguos módulos implementaban un algoritmo de detección de eventos STA/LTA y sólo grababan en disco los datos correspondientes a los eventos detectados. Cuando se desarrollaron dichos módulos la grabación continua resultaba poco asequible, no sólo por el elevado precio de los medios de almacenamiento masivo, sino también en términos de velocidad de los microprocesadores y periféricos. Actualmente la situación es muy distinta, y en un sistema como el que se describe es factible registrar de modo continuo los datos de todos los canales, lo cual resulta especialmente interesante

para señales de origen volcánico como el trémor, que puede ser difícil de detectar por un algoritmo automático y que suele tener una duración prolongada[47].

- *Volcado de datos.*- Como se ha visto en la introducción, en los módulos de dieciséis bits el control de todo el proceso corría a cargo de un PC portátil, que grababa los datos en ficheros de evento en su disco duro. La recogida de datos consistía en conectar el PC a otro portátil a través del puerto paralelo o serie y volcar todos los ficheros de uno a otro. En función del número de eventos detectados y grabados (que dependía a su vez del nivel de actividad y del tiempo que el sistema hubiera estado funcionando) este proceso podía durar entre algunos minutos y más de una hora. En el mejor de los casos esta tarea podía hacerse muy pesada, en especial si el personal tenía que ocuparse del mantenimiento de varias antenas. En el peor, podría resultar peligrosa si la actividad volcánica alcanza niveles de riesgo. Por tanto, uno de los requerimientos que se les exigía a los nuevos sistemas era que el procedimiento de recogida de datos se llevara a cabo de algún modo más eficiente. Como se verá más adelante, la decisión que se tomó fue utilizar discos USB externos para el almacenamiento de los datos. De este modo no es necesario volcar los datos, sino que basta con cambiar el disco externo con los datos adquiridos por otro vacío.
- *Consumo.*- El consumo de la electrónica de los antiguos módulos no es excesivo, pero se ve aumentado drásticamente por la necesidad de contar con un PC portátil para el control de todo el proceso. Aunque los requerimientos de este PC no son exigentes, su consumo de potencia sobrepasa con mucho el de los PCs de tipo industrial diseñados específicamente para sistemas embebidos. Por otra parte, la tendencia actual de la informática, que ofrece equipos cada vez más potentes pero con un mayor consumo, hacía que en los últimos años de operación de los módulos de dieciséis bits no fuera fácil encontrar portátiles adecuados para sustituir a los que se iban deteriorando por el uso. En los nuevos sistemas se usará como controlador del proceso el PC industrial *Lippert Cool Roadrunner II*, ya presentado en capítulos anteriores.
- *Operación en condiciones ambientales adversas.*- Los PCs portátiles no están optimizados para operar en condiciones ambientales adversas como las que se suelen dar en las campañas de campo. Por otra parte, las cajas y conectores de los módulos antiguos tampoco eran los adecuados para este tipo de ambientes, lo cual hacía necesario utilizar protecciones adicionales como bolsas de plástico y/o tiendas de campaña. Las nuevas antenas utilizarán conectores militares para algunas de las conexiones. Para otras se utilizarán conectores del mismo tipo que en las antiguas, pero las conexiones no se realizarán en el exterior de las cajas, sino dentro de ellas. Dichas cajas cumplirán normas estándar de protección (IP65), y para evitar pérdida de estanqueidad los cables se pasarán a través de pasamuros aislantes. En cuanto al PC, la utilización de un PC monotarjeta de tipo industrial en lugar de un portátil mejora la operatividad del sistema en las condiciones habituales de operación.

[47] Con los antiguos módulos es posible registrar de modo 'pseudo-continuo', programándolos en modo de disparo por ventana y asignando a los ficheros de datos la duración máxima (tres minutos). Sin embargo, los ficheros de datos quedan separados por el intervalo de tiempo que el sistema necesita para la grabación (aproximadamente treinta segundos), por lo que la proporción de tiempo grabado es de sólo el 85%, con el inconveniente añadido de que al ser la duración de los ficheros tan corta la probabilidad de que los eventos o señales de interés queden interrumpidos es muy alta.

5.2.2. Especificaciones técnicas y características

Teniendo en cuenta todo lo que se ha expuesto hasta ahora, se fijaron las siguientes especificaciones técnicas para los nuevos sistemas:

- 12 canales de adquisición por módulo.
- 24 bits nominales de resolución.
- Ganancia configurable. Como se verá más adelante, con objeto de simplificar el proceso de inicialización, los posibles valores se reducen a una ganancia baja (1), una media (4) y otra alta (16).
- En una primera fase la frecuencia de muestreo será de 100 mps. Como en el caso del Gran Sasso, el diseño del *hardware* debe permitir el muestreo a 200 mps, si bien el desarrollo de los programas se dejará para el futuro.
- Sincronización de datos mediante GPS. En este caso el modelo elegido será el *HVS 35*, de la marca *Garmin*. En el apartado de materiales se justificará esta elección.
- Bajo consumo. La utilización de las placas y dispositivos diseñados o seleccionados para la antena del Vesubio, en el que éste era también un criterio de diseño, garantiza un consumo reducido.
- Portabilidad. Esta característica se debe potenciar aún más que en la antena del Vesubio, ya que la utilización casi exclusiva de los nuevos módulos serán las campañas de adquisición de datos en campo, en la gran mayoría de los casos en entornos remotos.

Conviene reiterar que la idea en torno a la cual se centra el diseño de estos sistemas es la de aprovechar la experiencia y resultados obtenidos en la antena del Vesubio, para obtener un sistema de características similares pero bastante simplificado. Esta simplificación se traduce principalmente en la eliminación en el diseño de varios módulos *hardware* (SerPar, módulo de memoria, módulo de reloj, reducción del número de tarjetas PLL). Por otra parte, al aprovechar los resultados de la antena del Vesubio tanto desde el punto de vista *hardware* (casi todas las tarjetas son las mismas que se habían usado allí) como *software* (los programas de los microcontroladores PIC son los de la antena del Vesubio con algunas adaptaciones) se pretendía optimizar el tiempo de diseño y desarrollo.

5.3. Descripción general del sistema

5.3.1. Estructura del dispositivo y descripción general del funcionamiento

La figura 5.10 muestra un diagrama de bloques de los nuevos sistemas portátiles. Como puede verse, la estructura es parecida a la de la antena del Vesubio, con las diferencias principales de que toda la electrónica se concentra en la misma caja y de que faltan algunos módulos que se utilizaban allí (módulo de memoria, módulo SerPar, módulo de reloj). Así, para la adquisición se utilizan cuatro tarjetas de conversión A/D tipo SEISAD18, que gestionan tres canales cada una. Por tanto, el sistema admite un máximo de doce canales. La sincronización de la adquisición se consigue mediante una tarjeta PLL que genera, a partir de la señal de PPS del GPS, la señal de reloj que los conversores A/D toman como referencia para su operación. Al estar las cuatro placas de conversión A/D juntas, sólo es necesaria una de estas tarjetas en lugar de una por cada placa de conversión A/D, como ocurría en la antena del Vesubio. Igual que allí, las cuatro tarjetas SEISAD18 se conectan mediante una línea serie basada en el estándar RS-485. En este caso la proximidad de las tarjetas no obligaba a utilizar este estándar, pero sí el hecho de que todas las tarjetas deben conectarse a la misma línea serie, lo

que impide el uso de de un estándar punto a punto como el RS-232. Dado que los puertos serie del PC son de tipo RS-232, para la conexión debe utilizarse una placa de conversión de niveles RS-485/RS-232, que se ha diseñado específicamente. A diferencia del sistema del Vesubio, la transmisión de datos de las tarjetas A/D se realizará en este caso de modo asíncrono, como se explicará en la sección 5.3.3.

El PC industrial se ocupa del control de todo el proceso. Se ha utilizado el mismo modelo que en la antena del Vesubio (*Lippert Cool Roadrunner II*). Además de leer los datos de las cuatro tarjetas A/D por el primer puerto serie, el PC gestiona la comunicación con el receptor GPS a través del segundo, para poner las marcas de tiempo en los paquetes de datos según el formato que se describirá más adelante. Se ha utilizado un receptor GPS modelo *Garmin HVS 35*, que admite conexión directa al puerto serie sin necesidad de utilizar un circuito de interfaz como ocurría en el sistema del Vesubio.

Los datos se graban en ficheros en el disco duro interno. Cada cierto tiempo (seis horas en las versiones que se presentan de los programas) los ficheros de datos se vuelcan al disco duro externo a través de uno de los puertos USB del PC. De esta forma el tiempo que debe invertirse en el proceso de recogida de datos se reduce sensiblemente, ya que sólo será necesario volcar, como máximo, las últimas seis horas de datos del disco duro interno al disco USB. Además de usarse como almacenamiento temporal de los datos, el disco duro interno se utiliza como soporte del sistema operativo. Como se verá más adelante el programa del PC y los ficheros de configuración se almacenarán en los discos USB, para no tener que acceder al disco duro interno si se necesita hacer alguna modificación.

La alimentación de todo el sistema es proporcionada por una batería de 12 V. Una tarjeta de alimentación diseñada específicamente distribuye esta tensión a todos los módulos que la necesitan (GPS, placas de conversión A/D, sensores). Además utiliza un circuito conmutado de fuente LM2576-5 para obtener la tensión regulada de 5V necesaria para el resto de la electrónica (PC, disco duro interno y disco duro USB[48]).

La conexión de los sensores se realiza en una caja distinta de la de la electrónica, para evitar en la medida de lo posible la manipulación de ésta con la consiguiente pérdida de estanqueidad. Los cables de los sensores se pasan a través de las paredes de la caja supletoria usando pasamuros aislantes y las conexiones se realizan internamente. La caja de conexión de los sensores se conecta, a su vez, a la de la electrónica mediante una única manguera que lleva las señales de todos los sensores y la alimentación de éstos. Esta conexión se realiza mediante un conector de tipo militar.

En las siguientes secciones se describen con más detalle los principales aspectos relativos al funcionamiento general del sistema. En la sección 5.3.2 se comenta el formato utilizado para la transmisión de los parámetros de configuración desde el PC a las tarjetas de conversión A/D. En 5.3.3 se describe el protocolo usado para transmitir los datos desde las tarjetas A/D al PC. Por último, la sección 5.3.4 se centra en el procedimiento utilizado para sincronizar los datos de las cuatro tarjetas A/D. Teniendo en cuenta que dicho procedimiento es el mismo que en la antena del Vesubio, esta sección se limita a comentar las pequeñas diferencias entre ambos sistemas.

[48] Aunque el estándar USB permite alimentar los dispositivos periféricos a través del propio puerto, la intensidad que son capaces de suministrar los puertos USB del PC industrial no es suficiente para los discos duros, por lo que hay que utilizar una fuente externa.

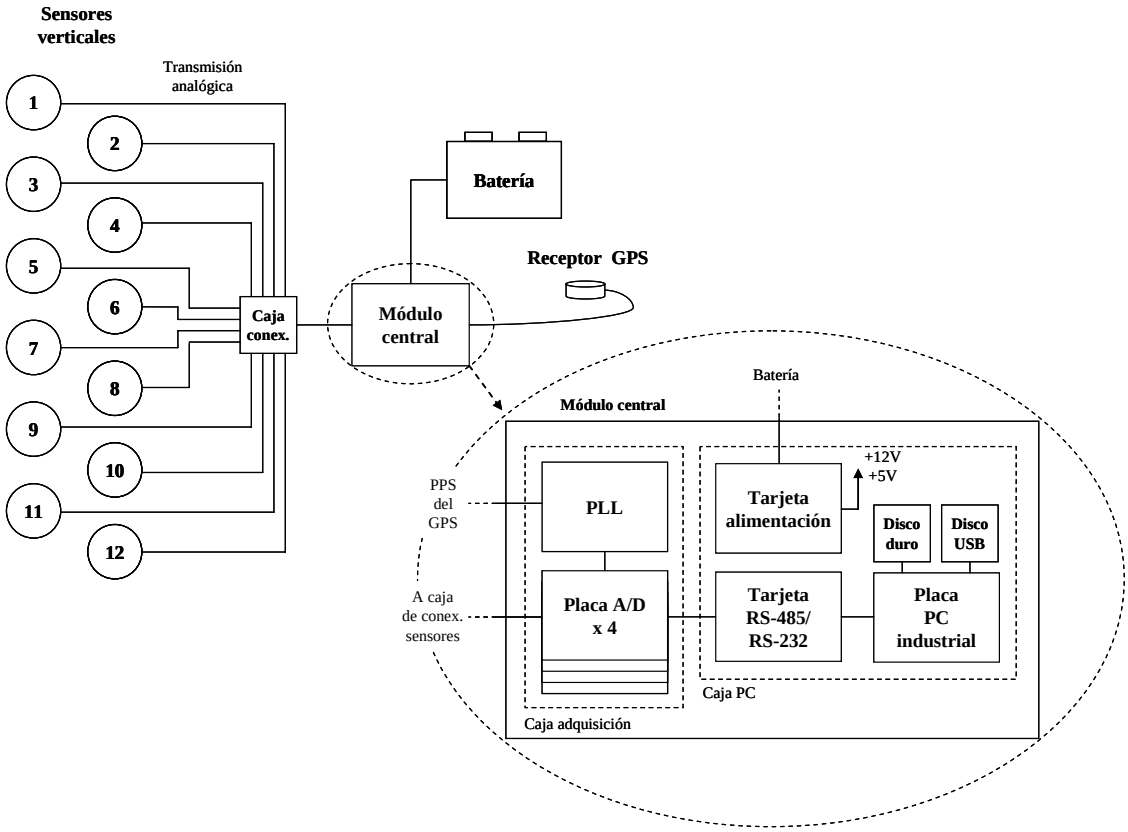


Figura 5.10. *Diagrama de bloques de las nuevas antenas portátiles del IAG. Las placas de conversión A/D y el PLL se encuentran en una caja distinta a la del PC industrial. Ésta última además contiene el disco duro interno, la placa de alimentación y la placa de adaptación RS-485/RS-232. Ambas cajas van dentro de una maleta estanca con protección IP-67, en la que se han montado los conectores necesarios para realizar la conexión del GPS, alimentación y sensores. La conexión de los sensores se realiza a través de otra caja externa, que se conecta a la maleta mediante una única manguera con un conector de tipo militar (ver figuras 5.19.c y d). Como en el caso de los antiguos módulos, los doce sensores verticales se pueden sustituir por cualquier combinación de sensores de una o tres componentes.*

5.3.2. Secuencias de inicialización

El formato para la transmisión de los parámetros de operación entre el PC y las tarjetas SEISAD18 se describe en Abril *et al.*, 2003. En esencia es análogo al utilizado en el sistema del Vesubio para la comunicación de los parámetros desde el módulo SerPar a los módulos de adquisición. Es decir, la información se codifica simplemente como una cadena de pulsos, siendo el número de éstos el que determina la combinación de valores de los distintos parámetros. Hay, sin embargo, algunas diferencias reseñables.

En primer lugar, en este caso, dado que la comunicación de datos se va a realizar de forma asíncrona, además de la frecuencia de muestreo y la ganancia es necesario transmitir la velocidad de transmisión serie (*baud rate*), para que los PICs configuren sus puertos serie[49].

[49] Como se vio cuando se presentaron las características de este modelo de microcontrolador, los PIC16F84 no tienen puerto serie USART. Según se verá más adelante éste debe implementarse por *software*, generando los microcontroladores los retardos de transmisión adecuados dependiendo de la velocidad programada, que por tanto deben conocer previamente. Así pues, el término 'configuración del puerto serie' se utiliza por simplicidad, refiriéndose a la asignación de valores a las constantes de retardo.

El número posible de combinaciones totales con estos tres parámetros (frecuencia de muestreo, ganancia y velocidad del puerto serie) es mucho mayor que en el caso del Vesubio (ver sección 4.3.3.2), por lo que con objeto de simplificar el proceso de inicialización sólo se codifican las combinaciones más utilizadas en la práctica. En Abril *et al.*, 2003, se explican con detalle los criterios seguidos para la selección de las posibles configuraciones, que se muestran en la tabla 5.1[50].

En segundo lugar, el PC genera los pulsos de forma distinta a como lo hacía el módulo SerPar. En aquel caso el PIC realizaba una gestión de los correspondientes pines de entrada/salida a bajo nivel, poniéndolos a nivel alto, temporizando mediante un bucle de retardo y devolviéndolos luego a nivel bajo. En este caso, y dado que la única línea de comunicación entre el PC y las tarjetas A/D es el puerto serie, resulta más cómodo generar los pulsos mediante el envío de caracteres. Puesto que la transmisión de los parámetros no es una tarea crítica en el tiempo, se realiza de forma relativamente lenta para garantizar la correcta identificación de la secuencia. Así, el PC inicializa el puerto serie a 9600 baudios con un bit de inicio, ocho de datos y uno de parada. Luego, por cada pulso de la secuencia, envía un carácter 00h, que al tener los ocho bits de datos y el de comienzo a cero puede interpretarse como un pulso de duración (ver figura 5.11):

$$1/9600bps \times 9bits = 0.937ms \qquad [5.1]$$

Es decir, aproximadamente 1 ms, como en el caso de la antena del Vesubio. Una vez enviada la secuencia el PC reconfigura el puerto serie a la velocidad adecuada.

En cuanto al procedimiento utilizado por los PICs para la identificación de la secuencia de parámetros, éste es exactamente igual que en la antena del Vesubio: al recibir el primer pulso se programa un *timer* con un valor mayor que el máximo tiempo que puede durar la secuencia, y mientras el *timer* está activo se incrementa un contador cada vez que se detecta un nuevo pulso. Si al final del tiempo asignado por el *timer* a la secuencia de inicialización el número de pulsos es menor de once, se interpreta que la secuencia recibida no correspondía a la enviada por el PC, sino que ha habido un error, por lo que se vuelve a la espera del primer pulso. Si el número de pulsos es mayor o igual a once, los valores de los parámetros de operación se asignan según la codificación mostrada en la tabla 5.1.

5.3.3. Transmisión de datos

A diferencia del sistema del Vesubio, en este caso la transmisión de datos se realiza de modo asíncrono. El formato de transmisión es el definido por el estándar RS-232 (ver figura 5.11), con un bit de comienzo, ocho de datos, uno de parada y sin bit de paridad. De esta forma es posible gestionar la línea serie a la que se conectan las cuatro tarjetas de conversión A/D directamente a través del puerto serie del PC, usando como interfaz únicamente la tarjeta de adaptación de niveles RS-485/RS-232. No hay necesidad de utilizar un módulo específico como SerPar, una de cuyas funciones era gestionar la comunicación síncrona con los módulos de adquisición mediante la generación de pulsos de transmisión y la lectura en paralelo de los datos de las cuatro líneas serie.

[50] Aunque en la versión que se presentará de los programas los valores de la frecuencia de muestreo y velocidad del puerto serie serán fijas (100 mps y 115200 bps, respectivamente), y por tanto bastaría con transmitir la ganancia, se ha preferido respetar el formato ideado para las tarjetas SEISAD18 por compatibilidad de las funciones de inicialización con futuras versiones.

Igual que se hacía en el sistema del Vesubio, la transmisión de datos se realiza en tiempo real. Los datos se transmiten antes de adquirir la siguiente muestra, de modo que no es necesario que los PICs de las tarjetas de conversión A/D los almacenen temporalmente. Ya se apuntó anteriormente que el estándar RS-485 no permite a más de un nodo transmitir simultáneamente. En el sistema del Vesubio era el módulo SerPar el que gestionaba los turnos de transmisión, para lo cual generaba una serie de pulsos de transmisión que cada módulo contaba para saber cuándo debía tomar el control de la línea serie. Como se verá más adelante, en este caso la división del tiempo se realiza en cada tarjeta de conversión. Para ello el PIC programa un temporizador con intervalos de tiempo distintos en función del número de identificación o posición lógica de la tarjeta en la línea serie y de la frecuencia de muestreo a la que operan los conversores.

Número de pulsos	*Velocidad de transmisión*	*Frec. de muestreo*	*Ganancia*
11	9600	25	1
12	9600	25	4
13	9600	25	16
14	19200	25	1
15	19200	25	4
16	19200	25	16
17	9600	50	1
18	9600	50	4
19	9600	50	16
20	19200	50	1
21	19200	50	4
22	19200	50	16
23	115200	50	1
24	115200	50	4
25	115200	50	16
26	19200	100	1
27	19200	100	4
28	19200	100	16
29	115200	100	1
30	115200	100	4
31	115200	100	16
32	115200	200	1
33	115200	200	4
34	115200	200	16

Tabla 5.1. *Las posibles combinaciones de parámetros que el PC comunica a las tarjetas de adquisición A/D se codifican mediante secuencias de distinto número de pulsos. Como en el caso del sistema del Vesubio, el número mínimo de pulsos es de once, para evitar que uno o varios (hasta diez) pulsos espúreos puedan disparar el proceso de inicialización en los PICs.*

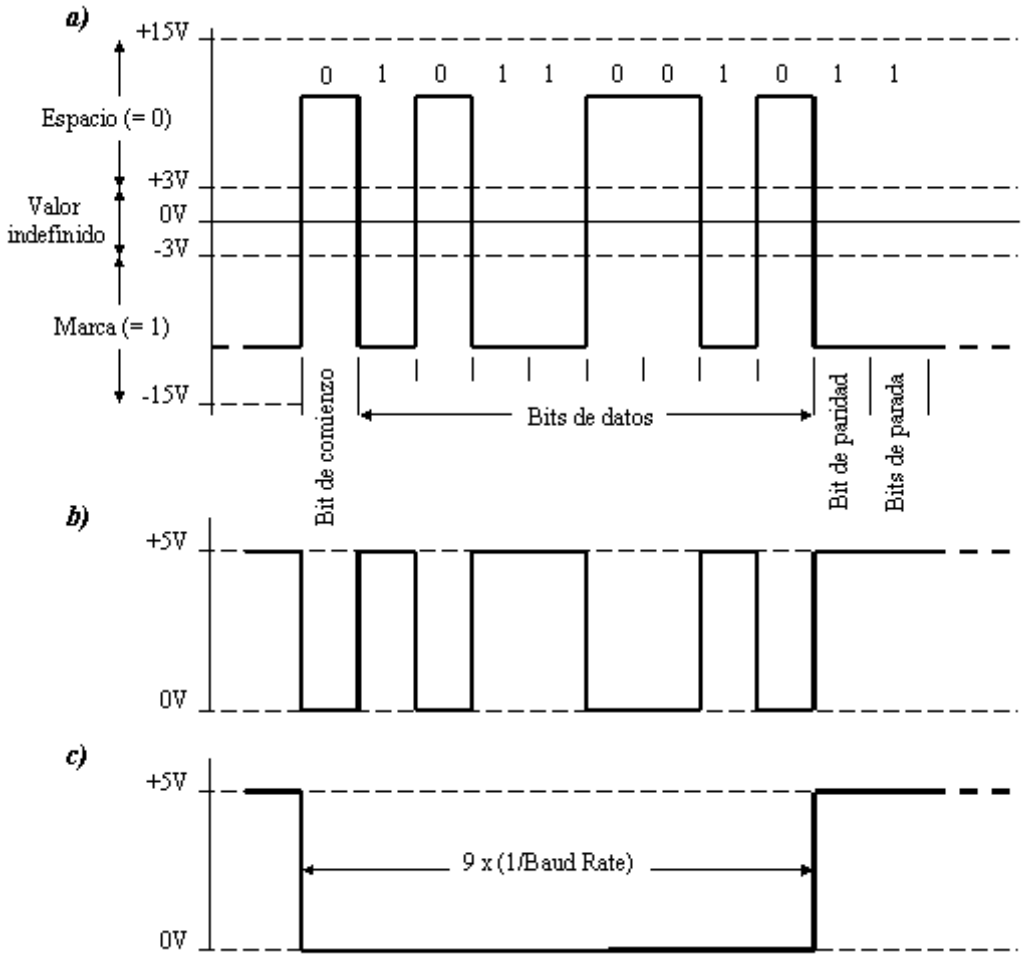


Figura 5.11. *El estándar de transmisión RS-232 admite niveles entre -15V y +15V. Como muestra la figura a) utiliza lógica inversa, puesto que el cero lógico (o espacio) queda caracterizado por tensiones positivas (entre +3 y +15V), mientras que el 1 lógico (o marca) puede ser cualquier valor entre -3V y -15V. El rango intermedio (±3V) no se identifica con ninguno de los dos estados. El formato de los datos consiste en un bit a 0 inicial que sirve para sincronizar los relojes de transmisión y recepción (bit de comienzo), siete u ocho bits de datos, un bit opcional de paridad para la detección de errores y un periodo correspondiente a un bit, un bit y medio o dos bits en los que la línea debe estar a nivel uno lógico (bits de parada), y que por tanto corresponde al mínimo tiempo de separación entre dos datos consecutivos. En la figura b) puede verse el mismo dato del ejemplo a) en niveles TTL, mientras que en c) se muestra la forma del PC de generar los pulsos de inicialización, mediante el envío de caracteres 00h (carácter NULL en código ASCII) a través del puerto serie.*

5.3.4. Sincronización

El proceso de sincronización se realiza de la misma forma que en la antena del Vesubio. La tarjeta PLL mide la diferencia entre cada pulso de segundo del GPS y el primer pulso de la señal DRDY de los conversores A/D, y en función de ésta actúa sobre el VCXO para generar la señal de reloj para los conversores A/D (ver sección 4.5.6). La diferencia más notable respecto al sistema del Vesubio es que una única tarjeta PLL debe generar la señal para cuatro tarjetas A/D. Sin embargo, la señal DRDY se toma de una sola tarjeta, a la que por convenio se le asigna la posición cero, y con ella se realiza el cálculo para modificar la frecuencia del VCXO. La señal de salida de éste se ofrece a las cuatro placas de conversión A/D, para lo cual hay que configurar los *jumpers* y conectar las tarjetas como se describe en la sección 5.4.2.

5.4. Desarrollo de la instrumentación

Teniendo en cuenta que las tarjetas de conversión A/D y de PLL son las mismas que se utilizaron en el Vesubio, en lo que respecta a tarjetas diseñadas específicamente para esta aplicación tan solo falta por presentar el circuito de adaptación RS-485/RS-232 y la placa de fuente. No obstante, se incluirá un apartado relativo a las tarjetas de conversión A/D y PLL, si bien únicamente para describir las diferencias en las conexiones y posiciones de los *jumpers* de configuración respecto a las usadas en la antena del Vesubio. En cuanto a los materiales, el único que supone una novedad respecto a los ya introducidos en capítulos anteriores es el receptor GPS, cuyas características y criterios de selección se presentan a continuación.

5.4.1. Materiales

5.4.1.1. El receptor GPS Garmin HVS 35

En el capítulo anterior se comentó que el principal criterio de selección del modelo de receptor GPS que se utilizó en la antena del Vesubio fue la experiencia previa que se tenía con receptores de esta marca. Esta experiencia implicaba un conocimiento tanto de la interfaz *hardware* como del protocolo de comunicación TSIP, así como la ventaja añadida de que se contaba con funciones en alto y bajo nivel para la gestión de este tipo de receptores.

¿Por qué, entonces, se decidió en este proyecto abandonar la línea seguida durante varios años y cambiar de marca?

La decisión tuvo su origen en la concepción inicial de los nuevos módulos. Como se explicará más detalladamente en la sección 5.5, el concepto original para estos sistemas era distinto a lo que finalmente se desarrolló, especialmente en lo que respecta al *software* de control del PC industrial. La primera idea surgió tras mantener contactos con el profesor Havskov, de la Universidad de Bergen, durante su estancia de un año en el IAG. El profesor Havskov es uno de los impulsores y creadores del paquete de software SEIS*, que incluye un sistema genérico y configurable de adquisición de datos (SEISLOG, Utheim y Havskov, 1999, Utheim *et al.*, 2001), un paquete de programas de procesado de datos sísmicos (SEISAN, Havskov y Ottemöller, 1999) y otro para la integración de sistemas de adquisición en redes sísmicas (SEISNET, Ottemöller y Havskov, 1999). De hecho, el conjunto tarjeta de adquisición + PLL responde a la misma nomenclatura (SEISAD18), ya que aunque se desarrolló inicialmente para el sistema del Vesubio, se adaptó para permitir su uso en otras aplicaciones en colaboración con el profesor Havskov (Abril *et al.*, 2003).

El concepto original para las antenas portátiles del IAG se centraba, en lo que a *software* se refiere, en un PC corriendo *Seislog* bajo *Windows* , que debía gestionar a través de uno de sus puertos serie los datos transmitidos por las cuatro tarjetas de conversión A/D según el formato establecido en Abril *et al.*, 2003. Asimismo, el PC debía gestionar los datos del receptor GPS a través del segundo puerto serie. Teniendo en cuenta que la programación de los *drivers* para la gestión de los datos desde el PC correría a cargo del personal de la Universidad de Bergen, se tuvo en cuenta su recomendación en lo referente al modelo de GPS. Ellos aconsejaban el uso del *Garmin 35*, puesto que ya habían programado los *drivers* para *Seislog* de otros digitizadores con ese GPS. Así, del mismo modo que en la antena del Vesubio se intentó aprovechar la experiencia propia en receptores de la marca *Trimble*, en este caso se procuró facilitar el desarrollo de la aplicación respetando las preferencias del equipo que se iba a encargar de una parte importante de la programación.

Sin embargo, no fue sólo la recomendación del personal encargado de la programación de los *drivers* lo que nos hizo decidir por este modelo. Una vez considerado su uso, se analizaron sus características y se descubrió que algunas de ellas se adaptaban perfectamente a la filosofía de los nuevos sistemas.

Así, la integración de toda la electrónica del receptor en un solo módulo hacía innecesario el desarrollo de un circuito de interfaz inteligente como el módulo de reloj de la antena del Vesubio. Por otra parte, el amplio margen de alimentación (6-40VDC) permitía alimentar el receptor directamente con la tensión de la batería, lo cual evitaba tener que utilizar un circuito de fuente con su correspondiente caja y conectores. Todo ello se ajustaba a los criterios de simplificación y optimización del tiempo de desarrollo buscados en este diseño.

La simplicidad que aporta este modelo no sólo es aplicable al *hardware*, sino también al *software.* Por una parte, la comunicación entre el PC y el receptor GPS se realiza a través del puerto serie y utilizando código ASCII en lugar de un código binario específico como el TSIP, lo cual facilita la visualización de los datos mediante editores de texto convencionales. Por otra, la configuración se puede realizar de una forma extremadamente simple. El receptor soporta el protocolo NMEA (ver, por ejemplo, Garmin Corporation, 2000), que se puede utilizar en la aplicación para implementar una función de inicialización de los parámetros de operación. Sin embargo, resulta mucho más fácil conectar el receptor a un PC con *Windows* y, a través de un sencillo programa de configuración, seleccionar la velocidad de comunicación, habilitar la señal de PPS y especificar los paquetes de datos que debe transmitir. Una vez hecho esto, el receptor memoriza la configuración, por lo que el programa de adquisición no debe preocuparse de la gestión del GPS, ya que éste envía periódicamente los paquetes habilitados. Los datos quedarán en el búffer de recepción del puerto serie hasta que sean gestionados por la aplicación.

El *Garmin HVS 35* presenta también algunos inconvenientes frente al receptor utilizado en el Vesubio. En primer lugar, es sensiblemente más caro que aquél, si bien la diferencia de precio queda parcialmente compensada por el ahorro en los componentes que ahora no se usan (interfaz con PIC, fuente de alimentación, caja, conectores).

En lo que respecta al consumo, el modelo usado aquí también sale desfavorecido frente al del Vesubio. El consumo típico del *Garmin* es de 870 mW (1W como máximo) frente a los 475 mW del *Trimble Lassen SK II*, aunque en este caso para calcular el consumo global habría que sumarle el del circuito de interfaz y fuente.

Por otra parte, el estándar RS-232 que utiliza el receptor *Garmin* no garantiza comunicaciones fiables más allá de unos pocos metros, razón por la cual no conviene aumentar la longitud del cable suministrado con el mismo (5 metros). Esto limita el emplazamiento de los sistemas a sitios con buena visibilidad GPS, en contra de lo que ocurría en el sistema del Vesubio, en el que el estándar RS-485 utilizado proporcionaba la posibilidad de elegir un sitio resguardado para el módulo central manteniendo la antena en el exterior.

En lo referente al *software*, la configuración del receptor mediante el programa específico aporta sencillez, pero también le resta potencia de programación respecto a la que ofrecía el protocolo TSIP, si bien en esta aplicación las opciones a las que se tiene acceso desde el programa de configuración son suficientes. En caso de necesitar configurar parámetros más específicos, se podrían implementar funciones de control en el programa de adquisición usando el protocolo NMEA, aunque para ello habría que reprogramar las funciones basadas en el protocolo TSIP ya utilizadas en otros proyectos.

Por último, el código ASCII hace más intuitiva la interpretación de los datos, pero también es menos eficiente en la codificación de la información, lo cual hace que los paquetes de datos sean más largos. Esto, unido al hecho de que el envío de los paquetes no se realice a demanda, sino a un ritmo determinado por el receptor, hace que se produzcan más errores de integridad de los datos que con el receptor utilizado en el sistema del Vesubio. Como se verá más adelante, estas contingencias deben resolverse en el programa de adquisición.

Las principales características del receptor GPS *Garmin HVS 35* se muestran a continuación (Garmin Corporation, 2000):

- Utiliza un receptor de señales de altas prestaciones, con capacidad de búsqueda y seguimiento de la señal de doce satélites simultáneamente, lo cual permite arranques rápidos (45 segundos sin efemérides, cinco minutos sin ningún dato).
- Tensión de alimentación: +6V / +40VDC no regulados.
- Consumo: 870 mW (típico), 1 W (máximo).
- Reloj de tiempo real y memoria interna alimentados por batería de litio de 3 V. La batería se recarga mientras el receptor está alimentado.
- Dos puertos serie RS-232.
- Soporta el protocolo de comunicación NMEA.
- Capacidad de GPS diferencial.
- Precisión de la medida de posición: 15 metros RMS (GPS no diferencial), 5 metros RMS (GPS diferencial).
- Proporciona una señal de un pulso de segundo (PPS) para aplicaciones de precisión. Los pulsos son positivos, de anchura configurable entre 20 y 980 ms, y su flanco inicial está sincronizado con el tiempo UTC con una precisión de $\pm 1\mu s$.
- Proporciona una salida de fase en formato binario.
- Memoria de programa de tecnología FLASH. El *software* es actualizable a través del puerto serie.
- Diseño integrado de receptor y antena, ideal para ahorrar espacio.
- Encapsulado estanco, idóneo para operar en condiciones ambientales adversas.
- Temperatura de operación: -30 ºC / +85 ºC.

5.4.2. Las tarjetas de conversión A/D y PLL

Las tarjetas de conversión A/D y PLL son las desarrolladas para la antena del Vesubio (SEISAD18), cuyos esquemas eléctricos pueden verse en las figuras 4.12 y 4.13. Las posiciones de los *jumpers* y microinterruptores de las tarjetas A/D son las mismas que se especificaron en la sección 4.4.2, salvo la del *jumper* JP7, que ahora debe estar en la posición B. En cuanto a la tarjeta PLL, la única modificación respecto a las descritas allí es que el *jumper* JP11 debe ponerse también en la posición B. Dada la proximidad del receptor GPS a la placa PLL no es necesario utilizar el estándar RS-485 para la transmisión de la señal de PPS, sino que ésta, de niveles TTL, se conecta directamente al conector CONN4 de la placa PLL. Poniendo el *jumper* JP11 en posición B se selecciona la línea procedente de este

conector y se desconecta del PIC la salida del transceptor de RS-485 U1.

Además de este cambio hay que hacer alguna modificación más en las conexiones. Como se comentó en la sección 5.3.4, la principal diferencia en el proceso de sincronización en este sistema respecto al del Vesubio es que ahora el PLL ofrece la señal de salida del VCXO a cuatro tarjetas, en lugar de a una. La señal DRDY, sin embargo, se toma de una sola tarjeta, y con ella se realiza el cálculo para modificar la frecuencia del VCXO. Las conexiones entre placas se deben realizar como se muestra en la figura 5.12. Al final del capítulo se mostrarán imágenes de los sistemas terminados en los que podrán apreciarse la disposición de las placas y conexiones (ver, por ejemplo, la figura 5.19.b). Los conectores CONN7 de la tarjeta A/D y CONN1 del PLL se eligieron de tipo IDC y se situaron en la misma posición en las placas, para permitir su interconexión mediante cable plano minimizando su longitud. Las señales DRDY de las tarjetas segunda a cuarta deben interrumpirse, para lo cual se debe cortar la línea correspondiente del cable plano.

Por otra parte, a diferencia de la antena del Vesubio, el programa de los PICs de las tarjetas de adquisición necesita la señal de PPS para programar los turnos de transmisión. Como muestran las figuras 5.12 y 5.19.b, la conexión de la señal de PPS a las tarjetas A/D se realiza a través de los conectores CONN8 y situando, según se ha indicado antes, el *jumper* JP7 en la posición B.

5.4.3. Tarjeta de conversión RS-485/RS-232

Como se ha indicado anteriormente, la conexión de la línea serie RS-485 al puerto serie del PC debe realizarse a través de un circuito de adaptación de niveles RS-485/RS-232 (figura 5.13). No es posible usar la tarjeta diseñada con el mismo fin para la antena del Gran Sasso (ver sección 3.5.2.1), puesto que allí se utilizaba una línea para conmutar entre recepción y transmisión. En el caso que nos ocupa la tarjeta debe ser transparente para el PC. Es decir, los dos canales (transmisión y recepción) deben estar permanentemente habilitados sin necesidad de utilizar señales de control, de modo que el PC pueda gestionar el puerto serie como en una aplicación de comunicación serie RS-232 convencional. De este modo se establecerá entre las tarjetas A/D y el PC una comunicación serie *full-duplex*, en la que el PC verá una línea de transmisión (PC TX) y otra de recepción (PC RX). Las tarjetas de conversión A/D verán las líneas con las funciones contrarias. El PC tendrá el control permanente de la línea PC TX, que las tarjetas A/D se limitarán a consultar durante la fase de transmisión de la secuencia de configuración, para lo cual sus transceptores RS-485 están permanentemente en modo de recepción (integrado IC7 de la figura 4.12).

En cuanto a la línea PC RX, será de recepción permanente para el PC, mientras que las tarjetas A/D distribuirán el tiempo de transmisión mediante turnos (ver sección 5.3.3), para lo cual conmutarán sus transceptores RS-485 entre transmisión y recepción (integrado IC8 de la figura 4.12).

El fundamento de esta tarjeta es el mismo que el comentado para la del Gran Sasso: la conversión de niveles se hace primero desde RS-485 a TTL y luego de TTL a RS-232. Para el segundo paso se utiliza el mismo integrado que en la tarjeta del Gran Sasso, un MAX232A. Para el primero no se puede utilizar el MAX489 como allí, ya que éste tenía señales de control para la conmutación entre los estados de transmisión y recepción. En lugar de ése se usa el MAX488, que permite el uso simultáneo de las líneas RX y TX sin utilizar señales de control. Se han utilizado transceptores MAX488 con limitación en la velocidad de respuesta (*slew-rate*), que aumentan la inmunidad al ruido electromagnético. Sin embargo, dada la reducida longitud de la línea serie éstos se podrían sustituir por su equivalente sin limitación de *slew-rate* (MAX490), compatible pin a pin con el 488 y normalmente más fácil de encontrar.

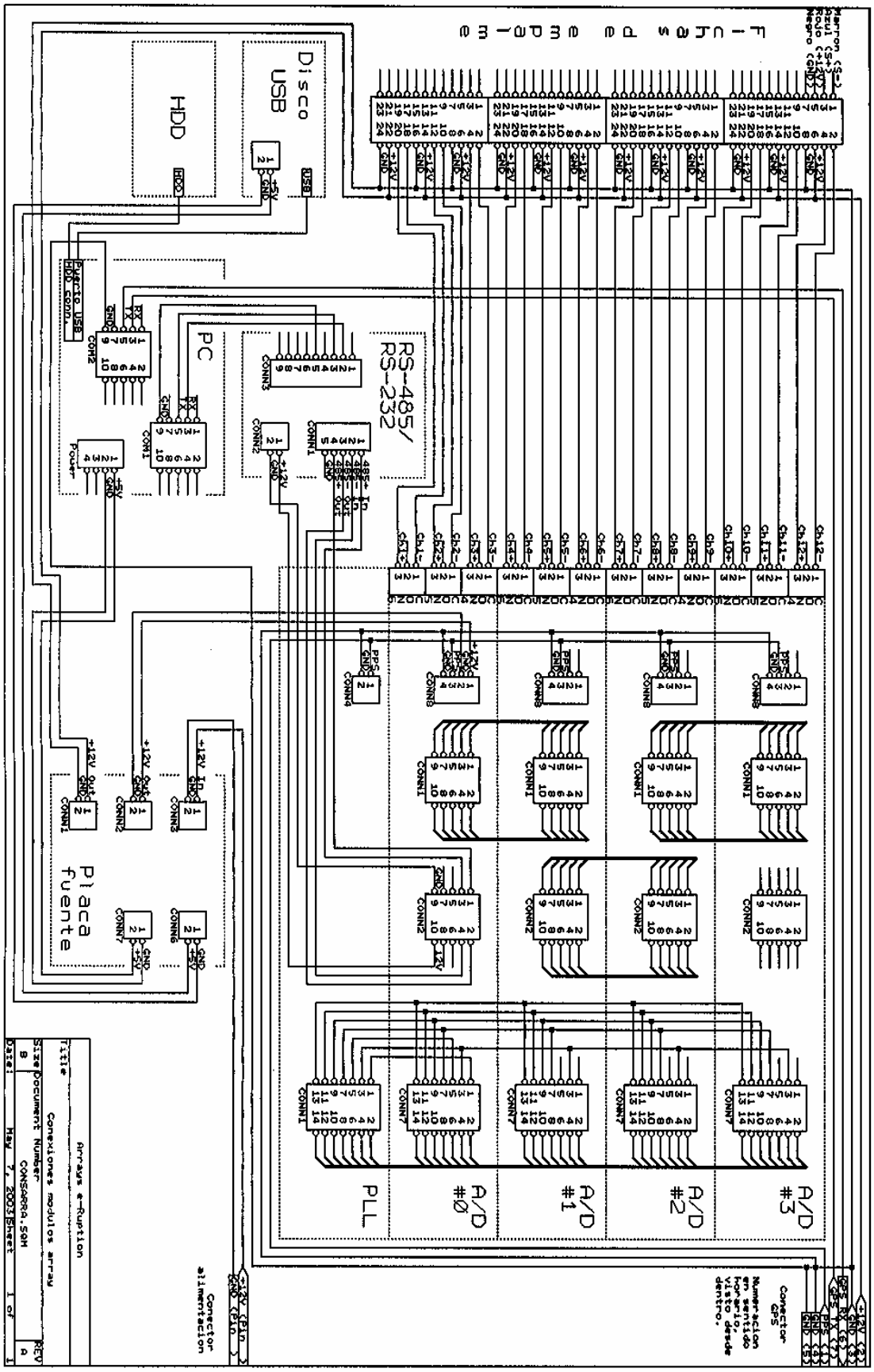


Figura 5.12. *Esquema de las conexiones en las nuevas antenas portátiles.*

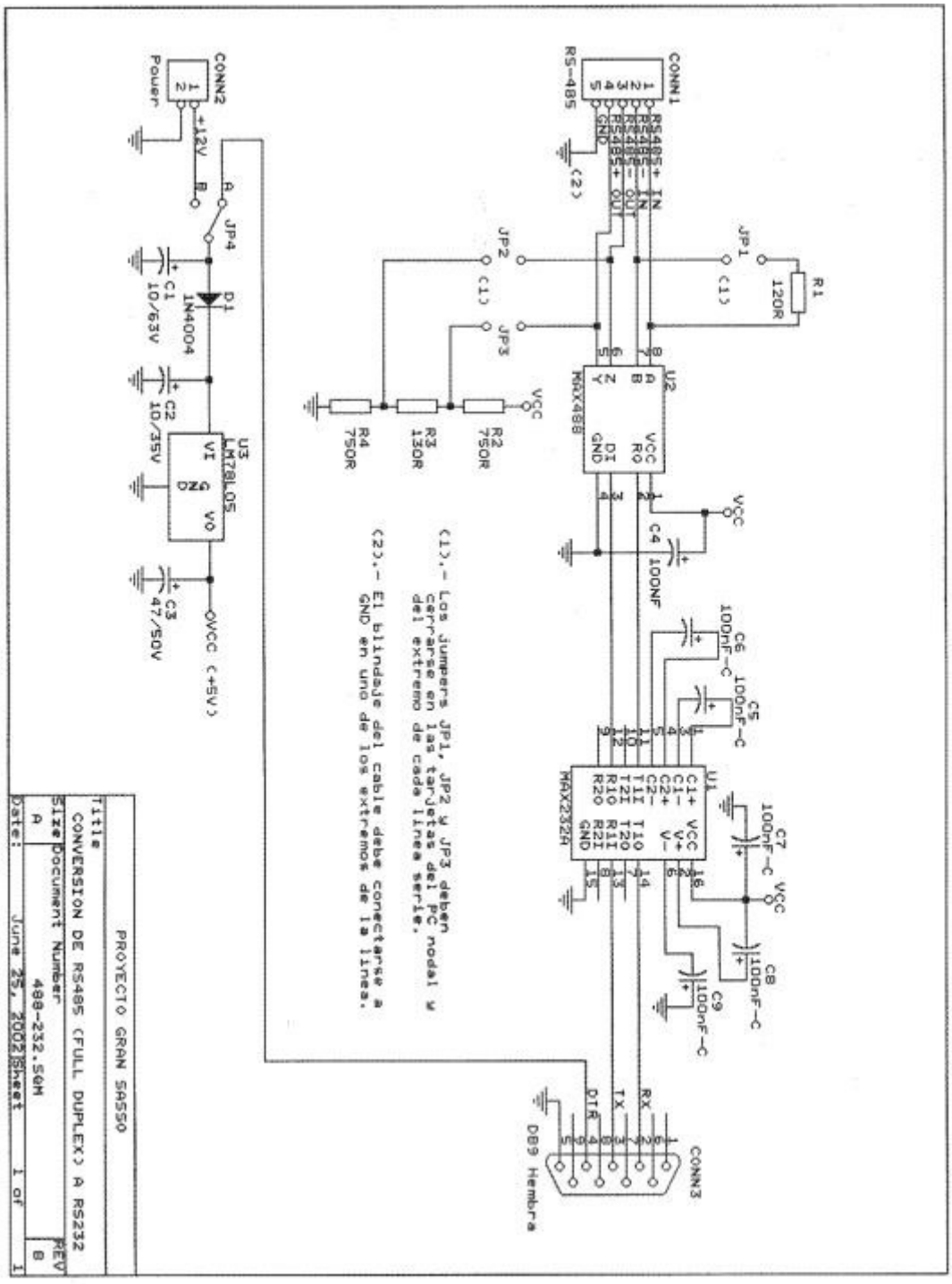


Figura 5.13. *Esquema eléctrico de las tarjetas de adaptación de niveles RS-485/RS-232.*

5.4.4. Circuito de fuente

La figura 5.14 muestra el esquema eléctrico de la placa de fuente. Como puede verse, el circuito es el mismo que en la tarjeta de PC del sistema del Vesubio (ver sección 4.4.6), y está basado en un regulador de tensión conmutado LM2576-5, con salida de tensión de 5 V y capaz de suministrar hasta 3 A de corriente.

La diferencia más importante respecto al circuito del Vesubio es que se han montado varios conectores para permitir la conexión de un LED y un interruptor de encendido, así como para facilitar la distribución de las tensiones de entrada y de salida. Desde esta placa se distribuye la tensión de la batería al receptor GPS, tarjetas de conversión A/D y caja de conexión de los sensores, y la tensión regulada de salida de la fuente al PC (que a su vez alimenta el disco duro interno a través del cable de interfaz) y disco duro externo USB.

Como en el caso de la placa del Vesubio, se han incluido varios elementos de protección (fusibles de entrada y de salida y un diodo para evitar daños en caso de conexión a la batería externa con polaridad inversa).

5.5. Programas

De acuerdo con la filosofía de simplificación que preside el diseño de los sistemas que se presentan en este capítulo, en este caso son sólo tres los programas que deben funcionar coordinadamente: uno en el PC, otro en el PIC de la tarjeta de conversión A/D y otro en el de la tarjeta PLL. Como en los dispositivos descritos anteriormente, el programa del PC se realizó en lenguaje C y los de los PICs en el lenguaje ensamblador propio de estos microcontroladores. Hay que hacer notar, sin embargo, que sólo en el caso del programa del PC (ARRAYS8.C) se partió de cero, ya que en los otros dos se aprovechó parte o todo el código desarrollado para los programas del Vesubio.

Como se ha comentado en la sección 5.4.1.1, originalmente se pensó en utilizar para la lectura de datos en el PC el paquete de *software Seislog* bajo *Windows*. En la próxima sección (5.5.1) se hace una breve introducción a este *software*, centrándonos en los problemas que dio en nuestro sistema y que nos llevaron a rediseñar la estrategia de lectura de datos en el PC. Se pasó a un concepto más parecido al del sistema del Vesubio, es decir, un programa bajo MSDOS diseñado específicamente para nuestra aplicación (ARRAYS8.C), que se describe en la sección 5.5.2. La sección 5.5.3, por último, presenta el programa desarrollado para el PIC de las tarjetas de conversión A/D (SEISAD_3.ASM). El programa del PIC de las tarjetas PLL es el mismo que en el sistema del Vesubio, por lo que ya se describió en el capítulo anterior (sección 4.5.6).

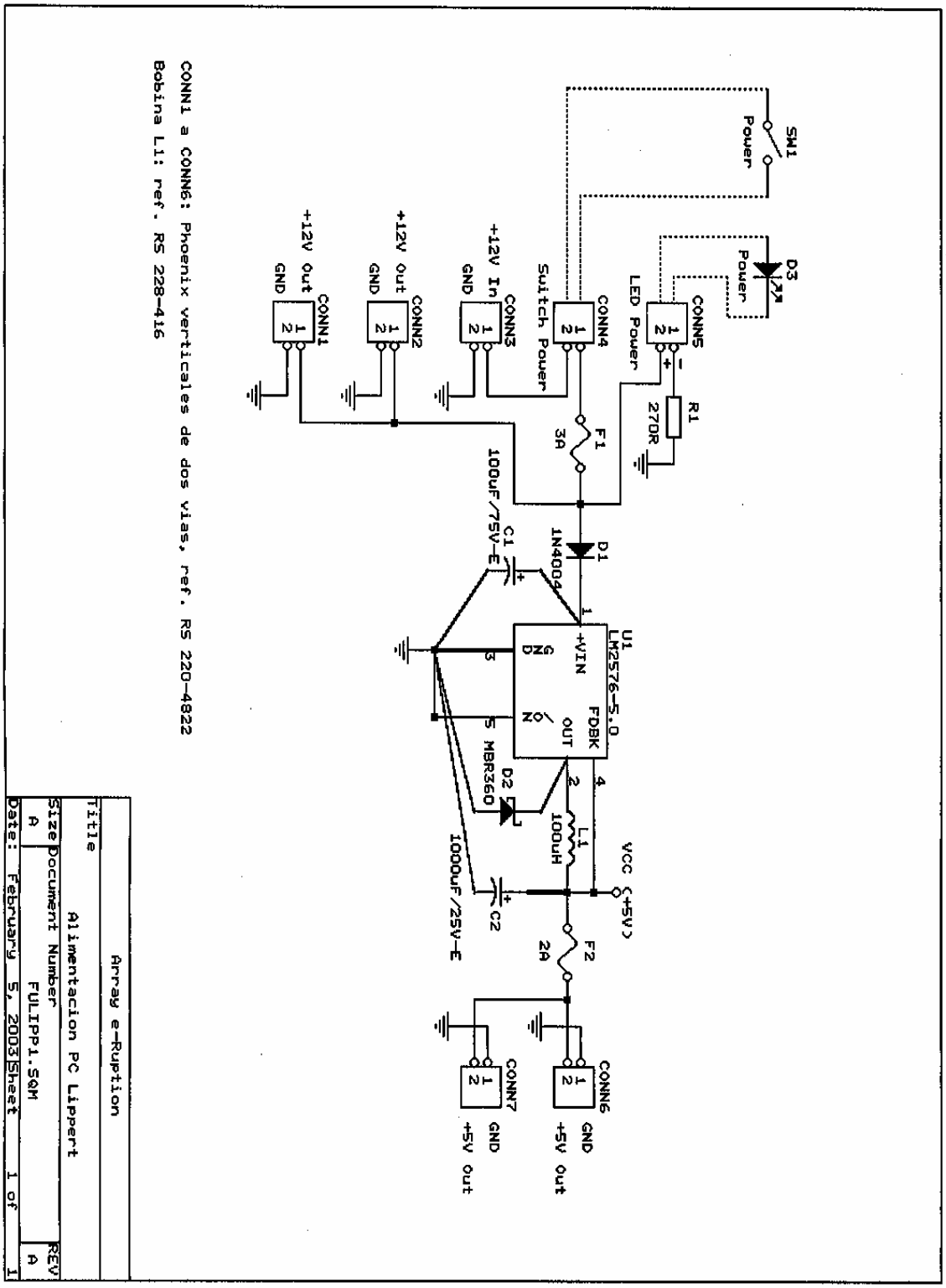


Figura 5.14. *Esquema eléctrico de la placa de fuente.*

5.5.1. Seislog

Seislog es un potente *software* de libre dominio[51] para adquisición de datos orientado a sistemas de adquisición de datos sísmicos (Utheim y Havskov, 1999). La versión para QNX[52] ha sido probada y ha estado operativa en diversos sistemas durante varios años (Utheim *et al.*, 2001). Sin embargo, hoy en día hay muchas aplicaciones en las que no es necesario usar un sistema operativo de tiempo real, puesto que se han desarrollado numerosos digitizadores que incorporan sistemas propios de gestión de tiempo. Por ello, y teniendo en cuenta la gran difusión de las diferentes versiones de *Windows,* los autores decidieron desarrollar una versión simplificada de *Seislog* para este sistema operativo. En la actualidad también hay disponible una versión simplificada para PDAs basada en el sistema operativo *Pocket PC.*

Inicialmente la versión para *Windows* se ideó para su uso con digitizadores GBV de bajo coste, pero las sucesivas actualizaciones fueron soportando muchos otros modelos. Una de las ventajas de *Seislog* es que su código es público y abierto, lo cual facilita la programación de nuevos *drivers* por parte de terceras personas. Las versiones actuales admiten numerosos digitizadores desarrollados por empresas, universidades o particulares.

5.5.1.1. Principio de operación

Un sistema de adquisición de datos basado en *Seislog* consiste en uno o varios digitizadores conectados a un PC a través de sus puertos serie. El programa también admite entradas desde ficheros. Se pueden conectar al PC un máximo de 50 digitizadores, cada uno de los cuales puede contar con uno o varios canales, hasta un máximo de 64, si bien el número total de canales que el sistema puede gestionar no puede sobrepasar los 208. Los distintos digitizadores pueden configurarse con parámetros diferentes (entre ellos la frecuencia de muestreo) y no necesitan estar sincronizados entre ellos.

Las aplicaciones más habituales contemplan que los digitizadores envíen bloques de datos con marcas de tiempo, para lo cual son ellos los que deben gestionar el sistema de sincronismo. Sin embargo, si ese no es el caso, la temporización también puede realizarse en el PC a partir del reloj del sistema. De esta forma se obtendrá una temporización menos precisa, aunque el reloj del sistema puede a su vez sincronizarse con un receptor GPS para aumentar la precisión. Esta fue la opción elegida para nuestra aplicación, dado que las tarjetas SEISAD18 no ofrecen la posibilidad de controlar un receptor GPS ni un sistema de tiempo de precisión alternativo.

Cada canal físico puede configurarse como uno o varios canales lógicos, en los que es posible asignar valores distintos a diversos parámetros, como los relativos al disparo del algoritmo STA/LTA y grabación en disco. Los datos de cada canal físico se almacenan temporalmente en búffers circulares (*ring buffers*), en los que se llevan a cabo cálculos correspondientes al algoritmo STA/LTA y se declaran detecciones de eventos individualizadas. Hay tantos *ring buffers* como canales lógicos. Por tanto, los datos procedentes de un mismo canal físico pueden pasar a varios *ring buffers*, participando en el proceso de detección de eventos con distinto peso en función de los valores asignados a los

---

[51] Las versiones de Seislog para QNX y para *Windows* , así como el *software* para el análisis de datos sísmicos SEISAN, están disponibles gratuitamente para su uso no comercial en la página web del Departamento de Ciencias de la Tierra de la Universidad de Bergen (www.geo.uib.no/seismo/software/software.html).

[52] QNX es un sistema operativo de tiempo real con capacidad de temporización precisa de procesos. Ofrece características multi-usuario, lo cual le proporciona una gran flexibilidad para integración en redes y para su uso remoto a través de puerto serie e Internet. Su pequeño tamaño (la versión básica cabe en un disquete de 1.44MB) lo hace especialmente adecuado para sistemas embebidos. En la página web www.qnx.com pueden consultarse sus características principales y descargarse versiones de evaluación.

parámetros en ese canal lógico. Esto resulta particularmente útil a la hora de calibrar un sistema, puesto que permite utilizar los mismos datos simultáneamente con distintas configuraciones para analizar qué combinación de valores se adapta mejor a los criterios de detección propuestos. Se puede habilitar independientemente la escritura en disco de cada uno de los canales lógicos. La escritura se realiza en ficheros que, por analogía con los *ring buffers*, se denominan simplemente ficheros de *ring buffer* o *ring buffers* de disco.

Teniendo en cuenta que SEISAN constituye la herramienta principal de inspección y procesado de datos, las capacidades de *Seislog* en ese sentido son limitadas. Básicamente consisten en dos tipos de ventanas de visualización de datos en tiempo real, en las que es posible también monitorizar numerosos parámetros de operación. La visualización y el procesado de los ficheros, tanto los de evento como los de *ring buffer*, corren a cargo de SEISAN.

Uno de los objetivos que *Seislog* persigue es que los sistemas que lo usen sean capaces de recuperarse frente a caídas de alimentación globales o fallos de elementos individuales. Así, si el digitizador deja de enviar datos por un fallo en la alimentación o cualquier otra causa, el programa sigue funcionando y vuelve a gestionar los datos del digitizador cuando éste se recupera. Una interrupción de este tipo provocaría que los ficheros *ring buffer* actuales se cerraran, iniciándose nuevos ficheros al recuperarse el flujo de datos del digitizador. De este modo quedaría garantizada la integridad y continuidad de los datos dentro de un mismo fichero.

El programa también corrige las diferencias de tiempo entre el reloj del sistema y la información de tiempo de los bloques de datos que la lleven incorporada. En la práctica esto se traduce en que el sistema puede gestionar, dentro de un límite, los retrasos producidos en la transmisión de los datos desde los digitizadores.

La figura 5.15 muestra varias ventanas de configuración y monitorización de la versión para *Windows* del programa *Seislog*.

5.5.1.2. Seislog y SEISAD18

La filosofía del sistema *Seislog* podría concretarse en las siguientes características:

- Robustez, representada por la capacidad de recuperación frente a fallos.
- Flexibilidad en el *hardware*: permite el uso de numerosos digitizadores y la incorporación de nuevos modelos, para lo cual sólo es necesario programar el *driver* específico sin tocar el núcleo del programa.
- Flexibilidad en el *software*: la configuración de canales de un dispositivo permite asignar valores a numerosos parámetros de operación, e incluso grabar los mismos datos con distintas configuraciones.
- Requerimientos *hardware* razonables: pese a ser un programa *Windows*, no es exigente en los recursos *hardware* que utiliza. De este modo se posibilita su uso en sistemas embebidos con PCs de bajo consumo y prestaciones medias.
- Simplicidad en las comunicaciones: el digitizador tan solo debe gestionar un puerto serie para proporcionar un flujo continuo de datos con un formato prefijado.

A priori esta filosofía se adaptaba muy bien al concepto inicialmente concebido para las nuevas antenas portátiles. Este paralelismo no era casual, puesto que cuando las tarjetas de conversión A/D de la antena del Vesubio se modificaron para permitir su uso en otras aplicaciones (dando lugar al sistema SEISAD18) y, sobre todo, cuando se definió el formato de la transmisión de los datos (Abril et al, 2003), se hizo teniendo en mente su integración en un sistema basado en *Seislog*.

a)

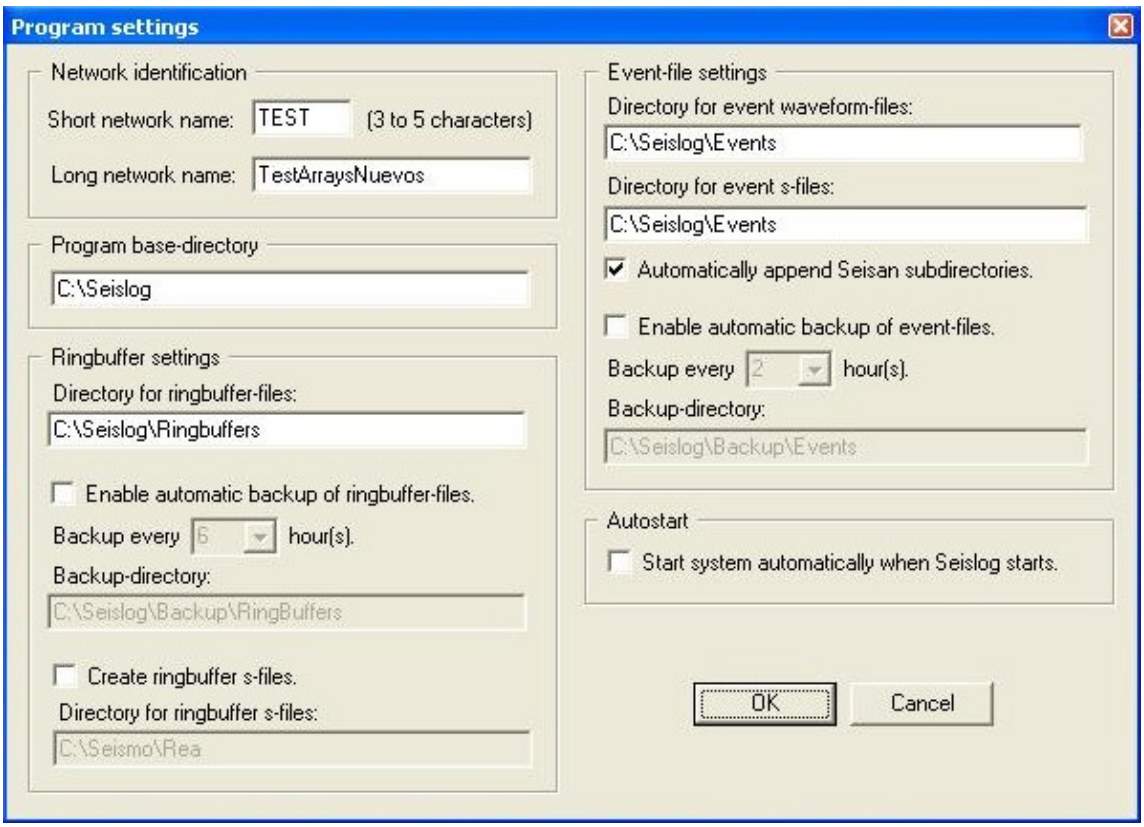
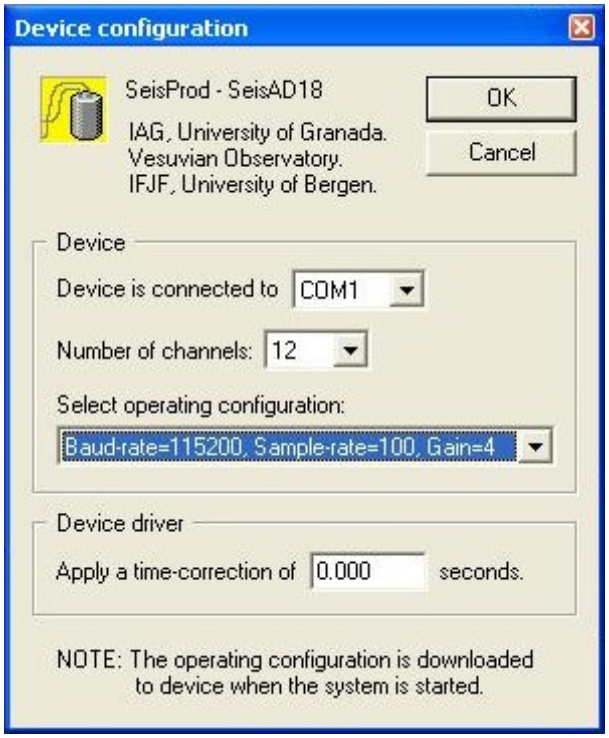


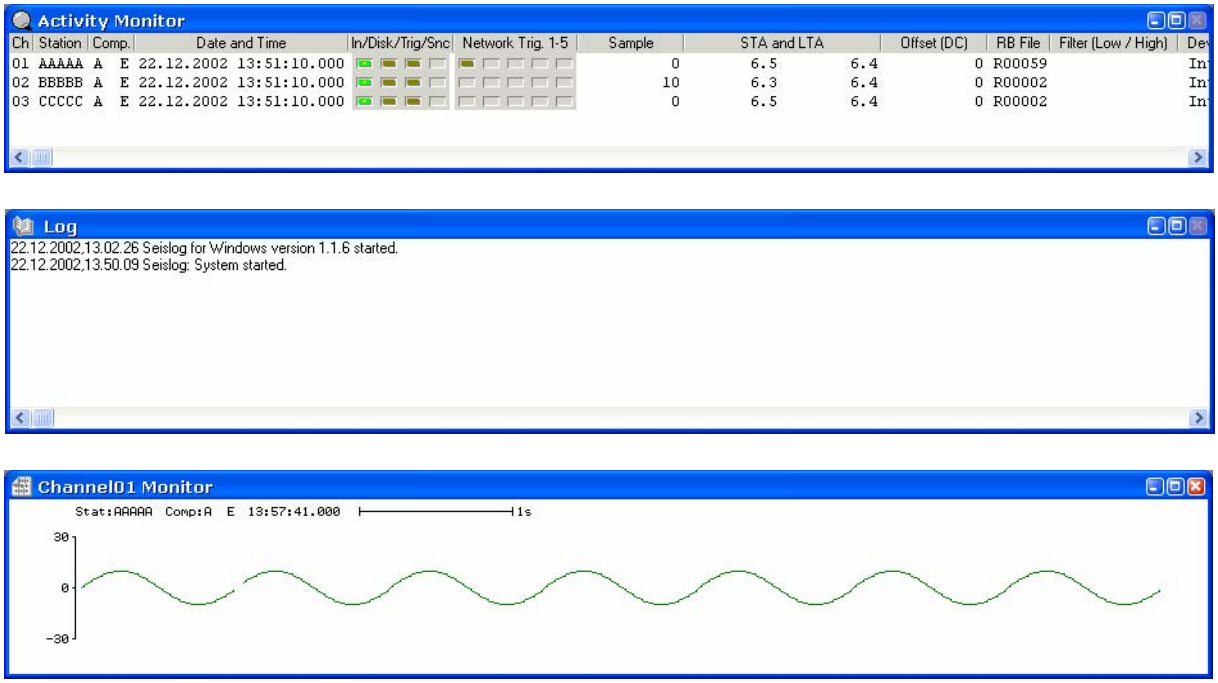
Figura 5.15. *Pantallas del programa Seislog para Windows. En a) pantalla de configuración general del programa, en la que se pueden especificar los directorios de trabajo y de grabación de los* ringbuffers *y de los ficheros de evento, así como dos nombres (normal y abreviado) para la red que se configura. En b) (página siguiente) se muestran las pantallas de configuración para añadir un nuevo dispositivo. Al seleccionar uno de ellos (en este caso las tarjetas SEISAD18) se abre una nueva ventana para introducir los parámetros de operación. En nuestro caso se han seleccionado las opciones necesarias para operar con las nuevas antenas sísmicas: 12 canales (es decir, cuatro tarjetas SEISAD18), frecuencia de muestreo de 100 mps y transmisión de datos a través del puerto serie COM1 con una velocidad de 115200 bps. En c) (página siguiente) se muestran las ventanas de monitorización durante el funcionamiento normal del sistema. La primera es el monitor de actividad, que permite conocer en tiempo real y para cada canal cuándo se graban los datos a disco, en qué momento se dispara el algoritmo de detección de eventos y el estado de recepción de la señal. Esta información se suministra mediante indicadores tipo LED. El monitor de actividad también proporciona, mediante indicadores alfanuméricos, información sobre el número de* ringbuffer *actual y varios parámetros del algoritmo de detección de eventos. La segunda ventana de la figura c) muestra el fichero LOG del sistema, en el que se registran las incidencias más significativas durante su funcionamiento, como inicio y fin de operación, errores (de formato de los datos, temporización,…) o disparos del algoritmo de detección. La tercera ventana muestra la señal de un canal en tiempo real. Los distintos canales se visualizan independientemente, de modo que es posible habilitar las ventanas correspondientes a todos o sólo a algunos de ellos.*

5.15.b)


Add new device
This Wizard will help you add a new device .
NOTE: Before attempting to add a new device, make sure the device is connected to the PC, and powered on.
Please select a device:
Kinemetrics - K2
M.Mariotti - SADC10/18/20/30
Measurement Computing - CIO/PCI DAS6402/16
Nanometrics - HRD24
Nanometrics - RD3/RD6
SeisProd - SEISAD18
Then next step is to configure the device.
Press the 'Configure' button to continue.
Cancel
Configure


Device configuration
SeisProd - SeisAD18
IAG, University of Granada.
Vesuvian Observatory.
IFJF, University of Bergen.
OK
Cancel
Device
Device is connected to COM1
Number of channels: 12
Select operating configuration:
Baud-rate=115200, Sample-rate=100, Gain=4
Device driver
Apply a time-correction of 0.000 seconds.
NOTE: The operating configuration is downloaded to device when the system is started.

5.15.c)


Activity Monitor
Ch Station Comp. Date and Time In/Disk/Trig/Snc Network Trig. 1-5 Sample STA and LTA Offset (DC) RB File Filter (Low / High)
01 AAAAA A E 22.12.2002 13:51:10.000 0 6.5 6.4 0 R00059
02 BBBBB A E 22.12.2002 13:51:10.000 10 6.3 6.4 0 R00002
03 CCCCC A E 22.12.2002 13:51:10.000 0 6.5 6.4 0 R00002
Log
22.12.2002,13.02.26 Seislog for Windows version 1.1.6 started.
22.12.2002,13.50.09 Seislog: System started.
Channel01 Monitor
Stat:AAAAA Comp:A E 13:57:41.000
1s
30
0
-30

Como ya se ha apuntado anteriormente, la programación del *driver* para las tarjetas SEISAD18 fue realizada por personal de la Universidad de Bergen, al que se le suministró como entrada el formato de la secuencia de datos que las tarjetas generarían. Gracias a su experiencia en este tipo de tareas, el *driver* estuvo preparado e integrado en una actualización de *Seislog* en un tiempo asombrosamente corto. A partir de entonces se comenzaron las pruebas. Primero con una sola tarjeta SEISAD18 y, tras los resultados satisfactorios, con la estructura completa diseñada para las antenas. Es decir, cuatro tarjetas de conversión A/D y una de PLL, conectadas al PC industrial a través del puerto serie mediante la tarjeta de conversión RS-485/RS-232. Las pruebas resultaron aparentemente satisfactorias, salvo por un detalle: no era posible, como se pretendía, grabar los datos directamente en el disco USB, por lo que debía volverse al sistema de volcado de datos utilizado en los módulos antiguos. Es decir, para acceder a los ficheros de datos era necesario comunicarse con el PC interno desde otro PC y volcar los datos al disco USB externo o al disco duro del PC externo.

El bautismo de fuego de los nuevos sistemas se produjo en la campaña de las Islas Azores del año 2003 (Abril *et al.*, 2004). El comportamiento de los equipos fue bueno, aunque se presentaron los problemas propios de las campañas de campo unidos a los atribuibles a la falta de experiencia en unos equipos nuevos. Una de las cosas que quedó clara después de la campaña fue la necesidad de resolver el problema del volcado de datos. Pese a que con los nuevos sistemas éste se realizaba a través de una interfaz *ethernet*, mucho más rápida que los puertos serie o paralelo de los módulos antiguos, el mayor número de canales y el registro continuo implicaban un volumen de datos muy superior, por lo que el tiempo de volcado se mantenía aproximadamente igual. El personal de mantenimiento debía invertir en esta tarea normalmente más de una hora por cada antena, lo cual resultaba totalmente inaceptable en unos equipos de nuevo diseño.

Hasta que no se empezaron a analizar los datos en profundidad no aparecieron los errores verdaderamente graves, consistentes en pérdidas de paquetes de datos en los ficheros de *ring buffer*. La gravedad de las pérdidas se veía incrementada por dos factores:

- Los errores se daban aparentemente de forma aleatoria, tanto en el tiempo como en la distribución por canales.
- Al no tener los paquetes de datos cabecera de tiempo, resultaban indistinguibles dentro del fichero. En la práctica esto se traducía en que resultaba imposible saber qué paquete era el que faltaba para poder sustituirlo o marcarlo de alguna forma.

Se realizaron una serie de pruebas para determinar la causa de los errores. Los resultados a los que se llegaron fueron:

- Los errores no eran introducidos por las tarjetas SEISAD18. Para comprobarlo se hicieron pruebas de transmisión con un programa específicamente diseñado, en el que los datos se almacenaban en un búffer de memoria para independizar la prueba del proceso de grabación en disco. Al salir del programa todo el búffer se volcaba a un fichero, cuyo formato se comprobaba, con especial atención a los bytes de número de paquete incluidos en la cadena de datos de las tarjetas SEISAD18 (ver formato en Abril *et al.*, 2003), para poder garantizar que no se perdían paquetes completos de datos.
- Los errores disminuían con el número de canales habilitados o, más exactamente, con el número de canales lógicos con escritura a disco activada. Pero incluso con el mínimo (una sola tarjeta SEISAD18, o sea, tres canales) se producían algunos.
- Los errores disminuían al hacerlo la frecuencia de muestreo programada para los conversores A/D.

- Los errores disminuían al aumentar las prestaciones del PC en el que corría *Seislog*.

La conclusión que parecía inferirse de estos resultados era que los errores estaban relacionados con la grabación de los datos en disco. El PC aparentemente no tenía tiempo para grabar los datos correspondientes a todos los canales, como sugería el hecho de que la frecuencia con que se producían las pérdidas aumentara al hacerlo el flujo de datos y disminuyera al aumentar las prestaciones del sistema. Ante esto, cabía abordar el problema desde varias ópticas:

- Disminución del flujo de datos.- No era viable, puesto que no se quería disminuir el número de canales de las antenas. Un cambio de formato para eliminar algunas de las cabeceras reservadas para uso futuro (ver formato Abril *et al.*, 2003) habría reducido algo el volumen de datos, pero no lo suficiente como para solucionar el problema.
- Utilizar un PC con mejores prestaciones.- Esta opción tampoco llegó a considerarse seriamente, ya que no se quería aumentar el consumo del sistema. Por otra parte, no habría resuelto de manera definitiva el problema, ya que algunas de las pruebas para determinar la relación de los fallos con las prestaciones del PC se hicieron con PCs de sobremesa cuyas características eran mejores que las disponibles en PCs de tipo industrial, y los fallos, aunque en número mucho menor, se siguieron produciendo.
- Modificar el *driver* de *Seislog*.- El personal de la Universidad de Bergen hizo varias versiones del *driver*, cambiando sobre todo la estrategia de escritura de los datos en disco, pero el problema no se llegó a solucionar.
- Enviar las tramas de tiempo con los datos desde la tarjeta A/D.- Probablemente no habría resuelto el problema de la grabación en disco, pero habría permitido localizar el momento en el que se producían los fallos y sustituir los paquetes perdidos por alguna marca de error. Se planteó hacerlo, aunque no como solución definitiva sino como prueba para determinar los errores. Sin embargo, para ello las tarjetas SEISAD18 (al menos una de las cuatro) tendrían que gestionar un receptor GPS para poner las marcas de tiempo en los paquetes de datos. El principal impedimento era que todos los puertos de entrada/salida del PIC de las tarjetas A/D están ocupados en funciones relacionadas con el control de los conversores A/D y transmisión de datos (ver figura 4.12), por lo que no quedan líneas libres para gestionar el receptor GPS. La utilización de un PIC con más puertos de entrada/salida (por ejemplo, el ya conocido 16C73) obligaría a rediseñar completamente la placa, por lo que esta opción no llegó a considerarse como factible dentro de este proyecto. Sí resulta una línea de trabajo interesante para el futuro, puesto que, como se dijo en la introducción a *Seislog*, el funcionamiento del programa está optimizado para aplicaciones en las que los digitizadores envían la información de tiempo incorporada con los paquetes de datos.
- Utilizar la versión de *Seislog* para QNX.- Al ser un sistema operativo para tiempo real, QNX permite un mayor control en la gestión de los recursos y probablemente resultaría más adecuado que *Windows* para nuestra aplicación. Sin embargo, presentaba varios inconvenientes. En primer lugar, la falta de experiencia en este sistema operativo provocaría retrasos en la puesta en marcha del sistema. En segundo, había que programar desde cero el *driver* para las tarjetas, si bien esta labor correría a cargo del personal de la Universidad de Bergen. Y por último, el mayor inconveniente era que el QNX no gestionaba los puertos USB, lo cual implicaba que no se podría utilizar el disco externo y habría que resignarse a continuar con el sistema de volcado de datos de los módulos antiguos.
- Programa específico para MSDOS.- Finalmente se decidió realizar un programa específico para MSDOS, que debía gestionar el flujo de datos por uno de los puertos

serie y controlar el receptor GPS por el otro. La robustez de este sistema operativo dio como resultado un programa fiable y simple. Su interfaz de usuario era menos amigable que la de *Seislog*, y no ofrecía sus posibilidades de monitorización de señales y parámetros ni implementaba ningún algoritmo de detección de eventos. Sin embargo, esto no era necesario en nuestra aplicación, en la que se pretendía un sistema de registro continuo (por tanto, sin necesidad de algoritmo de detección) para un sistema embebido (por tanto, sin posibilidades directas ni necesidad estricta de visualización de señales y parámetros). La experiencia en programación bajo MSDOS permitió optimizar el tiempo de desarrollo y prueba del programa, que se describe en la próxima sección.

No obstante, la opción de un sistema basado en *Seislog* no se abandonó completamente, sino que se contempla la posibilidad de retomarlo como trabajo futuro. Como se verá en el capítulo 7, el proyecto abarcaría varios aspectos, como la gestión del receptor GPS desde la tarjeta de conversión A/D, para lo cual habría que rediseñar ésta en torno a un modelo de PIC con más líneas de entrada/salida. Opcionalmente, se podría considerar la posibilidad de cambiar también el conversor A/D por otro de mejores prestaciones. En función de los resultados de las nuevas tarjetas con la versión de *Seislog* para *Windows*, se consideraría la posibilidad de utilizar la versión QNX, siempre teniendo en cuenta que la grabación de los datos debería hacerse directamente sobre disco USB.

5.5.2. Programa para el PC (ARRAYS8.C)

La figura 5.16 muestra el diagrama de flujo del programa realizado para el PC. Como se ha comentado en la sección anterior, es un programa relativamente simple, orientado a sustituir las funcionalidades básicas de *Seislog* en entorno MSDOS. Las principales tareas que realiza son la gestión de los dos puertos serie, comprobación del formato e integridad de los datos procedentes de las tarjetas de conversión A/D y del receptor GPS y escritura de éstos en ficheros de una hora de duración.

Las únicas interrupciones habilitadas son las de los puertos serie, IRQ4 e IRQ3, que se disparan cuando se recibe un nuevo carácter por el puerto serie correspondiente. El diagrama de flujo de las rutinas de servicio de ambas presenta la misma estructura (figura 5.16.a). Las principales tareas que se realizan en ellas son la escritura del carácter recibido en un búffer circular y la gestión de sus punteros. El carácter recibido se escribe en la posición actual del búffer circular correspondiente (COM1 o COM2) y el puntero de lectura se incrementa. Al hacerlo se comprueba si se ha llegado al final del búffer, en cuyo caso se reposiciona el puntero en la posición inicial.

Como se muestra en la figura 5.16.b, la primera tarea que se realiza en el programa principal es la comprobación de la correcta recepción de los pulsos de segundo del GPS. Esta señal es necesaria para la sincronización de la comunicación con las tarjetas, por lo que en caso de que los pulsos de segundo no se recibieran, se abortaría la ejecución del resto del programa. Como puede verse en el diagrama de flujo, la salida no se produciría inmediatamente, sino que se esperaría a que el fichero de incidencias (fichero LOG) estuviera abierto para registrar la causa que la produjo.

En esta aplicación la detección de los pulsos de segundo no se hace por interrupción, sino por consulta a la línea correspondiente (*polling*). Se prefirió esta opción porque el proceso de transmisión de datos desde las tarjetas de conversión A/D está sincronizado con los pulsos de segundo, por lo que se puede anticipar cuándo se van a recibir éstos. En estas condiciones, la gestión por *polling* era más simple y facilitaba la depuración del programa.

a)

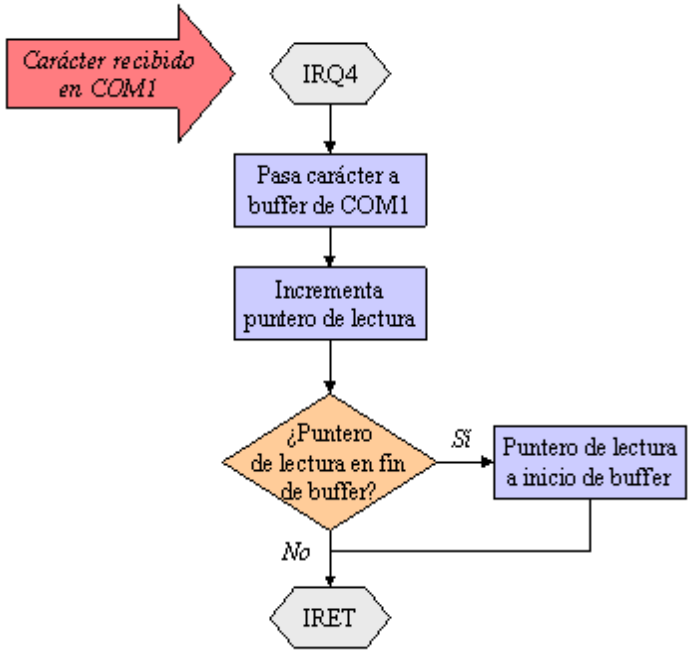


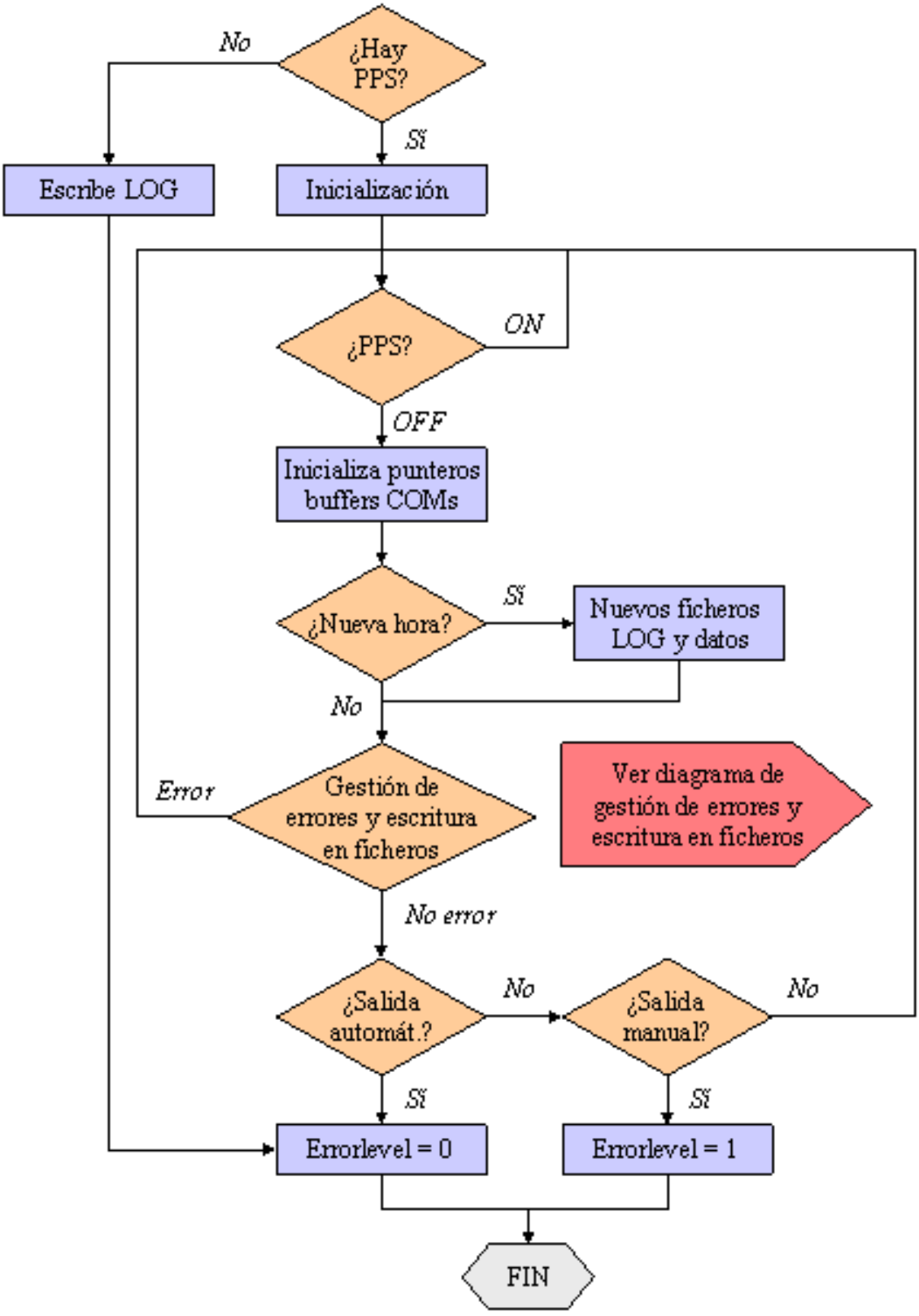
Figura 5.16. *Diagrama de flujo del programa del PC. En a) se muestra la rutina de servicio de la interrupción de carácter recibido en el puerto serie COM1. No se muestra la rutina equivalente para COM2, ya que su estructura es idéntica. En las páginas siguientes puede verse el diagrama de flujo del programa principal, que por claridad se ha dividido en dos: la estructura básica se muestra en b), mientras que en c) se recoge el procedimiento de gestión de errores y escritura de los datos y/o errores en ficheros.*

En caso de que los pulsos de segundo se reciban correctamente, el programa pasa a realizar las inicializaciones necesarias. En este caso son:

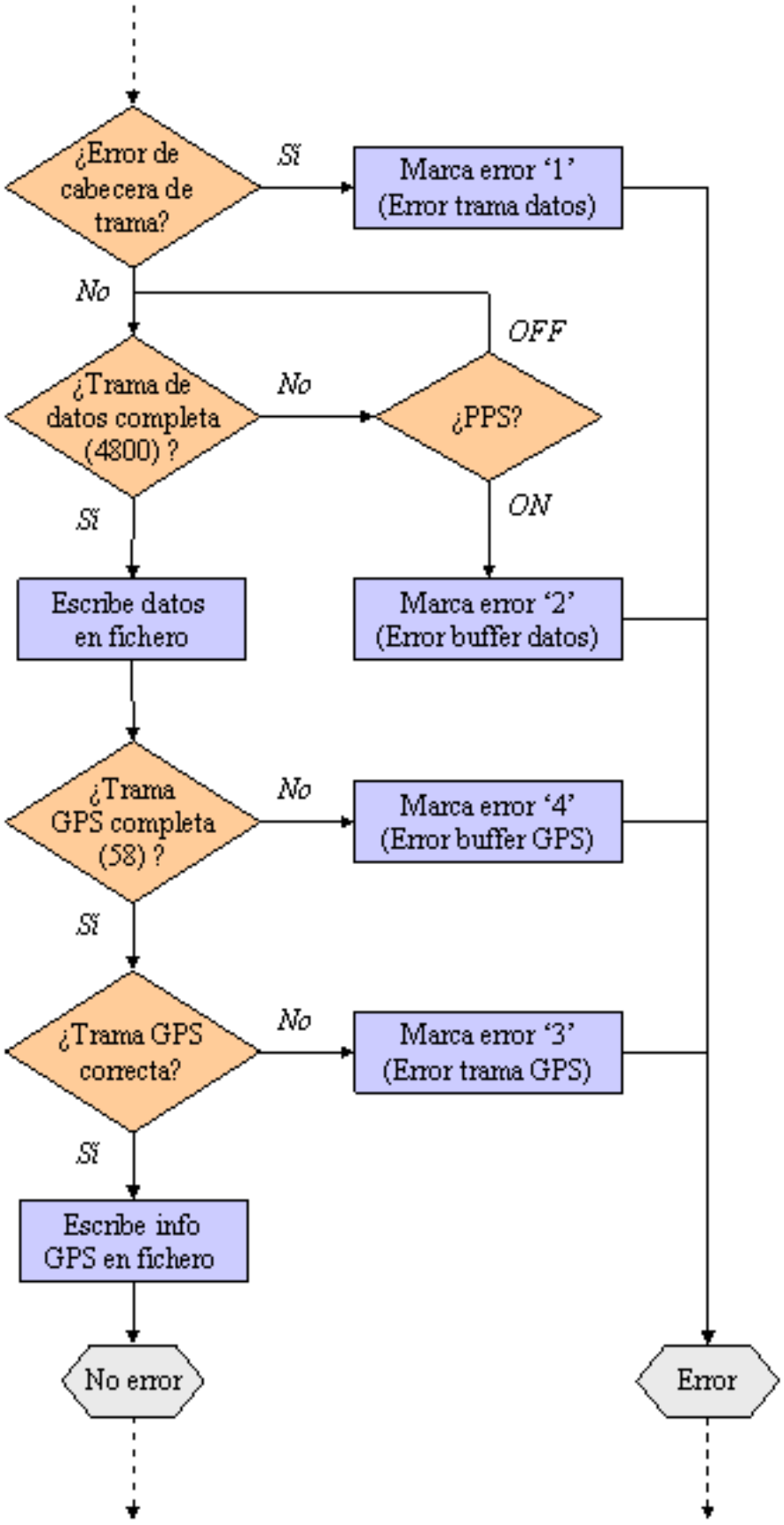
- Configuración de los puertos serie.- El primero (COM1) se inicializa para la comunicación con las tarjetas SEISAD18 a 115200 bps, con un bit de inicio, ocho de datos, uno de parada y sin bit de paridad. El segundo (COM2), que se utilizará para la comunicación con el GPS, se configura con los mismos valores para todos los parámetros, excepto la velocidad, que en este caso es de 9600 bps.
- Sincronización de la hora del PC con el GPS.
- Lectura del fichero de configuración SEISAD18.CFG.- Aunque en la versión que se presenta del sistema la frecuencia de muestreo y la resolución de los conversores AD7710 es fija, ambas se han incluido como parámetros en el fichero de configuración para facilitar su implementación en futuras versiones de los programas. Los otros dos parámetros del fichero son la ganancia, que puede tomar los valores 1, 4 o 16, y el código de sistema, consistente en dos caracteres alfanuméricos, y que aparecerá en la extensión de los ficheros de cada antena.
- Apertura de los primeros ficheros LOG y de datos, cuyos nombres se definen a partir de la información de tiempo tomada del GPS y del código de sistema leído del fichero de configuración.
- Envío de la secuencia de inicialización a las tarjetas de conversión A/D, con el formato descrito en la sección 5.3.2.

Una vez ejecutadas las funciones de inicialización, se entra en el bucle principal del programa. Éste se ejecutará una vez por segundo, para lo cual la condición de acceso al bucle es la recepción de un nuevo pulso de segundo. La detección del flanco inicial se realiza por *polling* a la línea correspondiente.

5.16.b)


No
¿Hay PPS?
Sí
Escribe LOG
Inicialización
¿PPS?
ON
OFF
Inicializa punteros buffers COMs
¿Nueva hora?
Sí
Nuevos ficheros LOG y datos
No
Error
Gestión de errores y escritura en ficheros
Ver diagrama de gestión de errores y escritura en ficheros
No error
¿Salida automát.?
No
¿Salida manual?
No
Sí
Sí
Errorlevel = 0
Errorlevel = 1
FIN

5.16.c)


¿Error de cabecera de trama?
Sí
Marca error '1' (Error trama datos)
No
OFF
¿Trama de datos completa (4800) ?
No
¿PPS?
Sí
ON
Escribe datos en fichero
Marca error '2' (Error buffer datos)
¿Trama GPS completa (58) ?
No
Marca error '4' (Error buffer GPS)
Sí
¿Trama GPS correcta?
No
Marca error '3' (Error trama GPS)
Sí
Escribe info GPS en fichero
No error
Error

A continuación se ponen a cero los punteros a los búffers circulares de recepción serie. Esta inicialización se realiza para que, en caso de que haya algún error, éste no se traslade de un segundo a otro. Poniendo todos los punteros a cero justo después de detectar el nuevo pulso de segundo se consigue que en cada bucle de segundo se gestionen exclusivamente los datos recibidos durante ese periodo de tiempo. Si quedara algún dato en cualquiera de los dos búffers, éste se descartaría, marcando el error correspondiente según se explica más adelante.

Después de la inicialización de los punteros se comprueba el valor de las variables de tiempo actuales y, en caso de que correspondan a una nueva hora se cierran los ficheros de datos y LOG actuales y se abren otros nuevos.

A continuación se realizan las comprobaciones de formato e integridad de los datos, tanto de los recibidos de las tarjetas SEISAD18 como del receptor GPS. En primer lugar se comprueba que el formato de la cabecera de la trama de datos[53] es correcto. Empíricamente se ha comprobado que casi todos los errores de formato que se producen consisten en un intercambio en el orden de las tramas procedentes de las distintas tarjetas, por lo que por simplicidad el algoritmo de comprobación del formato se reduce al análisis de los tres primeros bytes, que deben corresponder a la cabecera de trama de la primera tarjeta (55 55 55h). En caso de error, se sustituye la trama de datos por una marca predeterminada, consistente en diez caracteres iguales. Se han elegido como marcadores de los distintos errores los caracteres ASCII ‘1’, ‘2’, ‘3’ y ‘4’, para facilitar la identificación del error cuando se visualizan los ficheros con editores de texto. Por tanto, si se produce un error como el descrito (‘error de trama de datos’) se sustituye toda la trama por diez caracteres ‘1’ y se vuelve al inicio del bucle principal (es decir, a la espera de un nuevo pulso de segundo). Los diferentes tipos de errores se describirán con más detalle en la sección 5.6.9.

Si el formato de la cabecera de trama es correcto se pasa a la espera del resto de la trama. Para ello se comprueba el valor del puntero de lectura del búffer circular de recepción de COM1, y se permanece en el bucle de espera hasta que éste toma el valor correspondiente a una trama completa (4800). Sin embargo, un bucle simple de espera provocaría una parada del programa en este punto si se dejaran de recibir datos. Para evitarlo se consulta continuamente la línea PPS y, en caso de que se detecte un nuevo pulso antes de haber recibido todos los datos de la trama, se vuelve al inicio del bucle principal marcando previamente el error como ‘2’ (es decir, sustituyendo toda la trama de datos por diez caracteres ‘2’).

En caso de que la recepción de la trama haya sido completa, ésta se graba en el fichero de datos. La velocidad de la interfaz USB[54] no permite la grabación directa de los datos en el disco externo, por lo que ésta se realiza en el disco duro interno. Como se describirá más adelante, los ficheros se vuelcan cada seis horas al disco externo para agilizar el proceso de recogida de datos por parte del personal de mantenimiento.

Teniendo en cuenta que las tarjetas SEISAD18 transmiten los datos a medida que los adquieren, la recepción completa de la trama de datos en el PC implica que ha transcurrido prácticamente todo el tiempo asignado al bucle principal. En este punto se procede a la comprobación del formato e integridad de las tramas GPS, tareas que se realizan al final del bucle con objeto de dejar el máximo tiempo posible al receptor GPS para enviar la información de tiempo al PC.

Como se muestra en la figura, en este caso la primera comprobación es la de trama completa, dando en caso de que no se cumpla la condición de la sentencia condicional un

[53] Por simplicidad se denominarán simplemente ‘tramas de datos’ a las secuencias de datos procedentes de las tarjetas SEISAD18, mientras que los datos enviados por el GPS se nombrarán como ‘tramas GPS’.

[54] El PC *Lippert Cool Roadrunner II* soporta el estándar USB 1.1. El estándar 2.0 incorpora un modo de alta velocidad con velocidades de transferencia de hasta 480 Mbps, que sería suficiente para grabar los datos directamente en el disco externo.

error de búffer GPS, que se marca con diez caracteres ‘4’. A continuación se comprueba el formato de la trama, marcando el error de trama en caso de que se produzca con diez caracteres ‘3’. A diferencia de los errores en las tramas de datos, en los que todos los datos del segundo se sustituían por los códigos de error, ahora sólo es la información del GPS la que se elimina. Por tanto, en el fichero aparecerán los datos sísmicos pero sin información de tiempo ni status GPS, que los programas de procesado tendrán que tomar del segundo siguiente o del anterior.

Pese a dejar el mayor intervalo de tiempo posible dentro del bucle principal antes de comprobar la trama GPS, en las primeras pruebas de los programas se verificaba que en determinadas ocasiones ésta no se había recibido completa en el momento de realizar la comprobación. Monitorizando las tramas GPS a través de un osciloscopio se pudo ver que en algunos casos la trama terminaba de recibirse después de la llegada del siguiente pulso de segundo. El caso extremo de estos retardos de transmisión se daba en ocasiones en las que entre dos pulsos de segundo no se recibía trama de tiempo alguna, para recibir dos en el siguiente segundo. Con objeto de minimizar los errores causados por estos retardos se implementaron dos modificaciones en el algoritmo de detección de trama. En primer lugar, como puede verse en la figura, la condición para considerar que la trama está completa no es que se hayan recibido todos los caracteres que la forman (74), sino sólo aquellos que contienen la información que se va a utilizar (los 59 primeros de la trama). En segundo, se comprueba si se han recibido dos tramas en el mismo segundo, y en caso de que sea así se queda con la última. De esta forma se evita asignar la información de tiempo falsa a una trama de datos.

Una vez comprobada la integridad y formato de la trama de información GPS, ésta se escribe en el fichero de datos. Aunque no se ha incluido en el diagrama de flujo por simplicidad, cuando se escriben los datos y la información GPS en el fichero de datos los búffers circulares se gestionan de modo análogo al descrito en las rutinas de servicio de las interrupciones. La detección de que hay datos en el búffer circular listos para ser escritos se realiza comprobando los punteros de lectura y escritura del búffer correspondiente. En caso de que las posiciones a las que apuntan sean distintas, se escribe en el fichero el byte contenido en la posición actual del puntero de escritura y se incrementa éste, tantas veces como sean necesarias para que ambos punteros apunten a la misma posición. Como se hacía en las rutinas de servicio de las interrupciones, cada vez que se incrementa cualquiera de los dos punteros se comprueba si ha llegado al fin del búffer, en cuyo caso se reposiciona éste asignándole el valor correspondiente a la posición inicial del búffer.

La última tarea que se muestra en la figura 5.16.b es la comprobación de las condiciones de salida del programa. Se han definido dos tipos de salida: manual y automática. La salida manual se produce cuando se presiona el pulsador de salida o una tecla, en caso de tener conectado un teclado. La salida automática se produce cada seis horas, para realizar el volcado de los ficheros de datos desde el disco duro interno al disco USB. Como se verá más adelante, la operación del sistema se gestiona desde un fichero de proceso por lotes, para lo cual la salida automática y la manual devuelven códigos de error distintos ((ERRORLEVEL 0 y 1, respectivamente) interpretables por el sistema operativo.

5.5.3. Programa de las tarjetas de conversión A/D (SEISAD_3.ASM)

El programa de las tarjetas de conversión A/D debe realizar las mismas funciones que el equivalente de la antena del Vesubio (ver sección 4.5.5), con alguna diferencia. A efectos de programación, la más importante es que ahora la transmisión de datos se realiza de forma asíncrona. En la medida de lo posible se utilizaron las funciones del programa del Vesubio, lo cual redundó en una optimización del tiempo de programación y prueba. Así, se aprovecharon

gran parte de las funciones de inicialización y control de los conversores A/D e identificación de la secuencia de parámetros de configuración desarrollados anteriormente. La parte nueva de programación se centró principalmente en la implementación del puerto serie asíncrono, funciones de transmisión de datos y reorganización del programa principal que, como puede verse en el diagrama de flujo de la figura 5.17, queda notablemente simplificado.

Hay dos interrupciones habilitadas: la externa, disparada por la señal de pulsos de segundo, y la interrupción de saturación del *timer 0*, que se utiliza para generar distintos retardos. Como se muestra en 5.17.a, la rutina de servicio de la interrupción externa se reduce a activar el flag de PPS recibido, que se utilizará en el programa principal para comprobar la llegada de un nuevo pulso de segundo. Por su parte, la rutina de servicio correspondiente a la interrupción de saturación del *timer 0* decrementa los contadores de retardo y comprueba si éstos se hacen cero, en cuyo caso se activa un flag de fin del retardo programado (figura 5.17.b). Los contadores se inicializan en el programa principal con los valores necesarios para generar el retardo de la duración deseada. Cuando ha transcurrido el tiempo programado, se desactiva la interrupción del *timer 0* para que deje de servirse hasta una nueva inicialización de los contadores. Los retardos generados mediante esta técnica presentan la ventaja respecto a los bucles de retardo de que el programa se puede dedicar a otras tareas, ya que el control del tiempo transcurrido lo realiza el *timer* en segundo plano contando los ciclos de instrucción consumidos.

El programa principal (figura 5.17.c) comienza con las inicializaciones, que en este caso consisten en:

- Configuración de puertos de entrada/salida.
- Inicialización de variables (contadores, variables de *checksum* y número de bloque,…).
- Configuración de interrupciones.
- Lectura de la secuencia de parámetros de configuración que envía el PC y cálculo de las constantes para inicialización de los CADs.
- Programación de los CADs con los valores calculados.
- Lectura del código de tarjeta y, en función del mismo, asignación del carácter que se usará como cabecera de las tramas de datos.

Como puede verse, las inicializaciones son muy parecidas a las del programa del Vesubio. La diferencia más importante es la lectura de la secuencia de parámetros de configuración que envía el PC. Aunque, como se dijo en la sección 5.3.2, el procedimiento utilizado para la identificación de la secuencia es prácticamente el mismo, en este caso la detección de los pulsos de la secuencia no se puede realizar mediante interrupciones, ya que el pin RB0 asignado a la interrupción externa está conectado a la señal PPS. La identificación de los pulsos se realizará, por tanto, mediante *polling* a la línea correspondiente.

Tras las inicializaciones se entra en el bucle principal del programa. En esta ocasión la condición de entrada es que los conversores tengan un dato listo, para lo cual el PIC consulta continuamente la señal DRDY. Cuando ésta se activa, se consulta el *Flag PPS* que, como se ha dicho antes, se activa en la rutina de servicio de la interrupción externa. Si *Flag PPS* está activado quiere decir que la muestra que se va a leer es la primera del segundo, por lo que se incrementa la variable de número de bloque, que se enviará con los datos para permitir el control de los paquetes de datos en recepción. Además se desactiva el *Flag PPS* y se activa el flag indicador de que se trata de la primera muestra del segundo (*Flag Muestra1*).

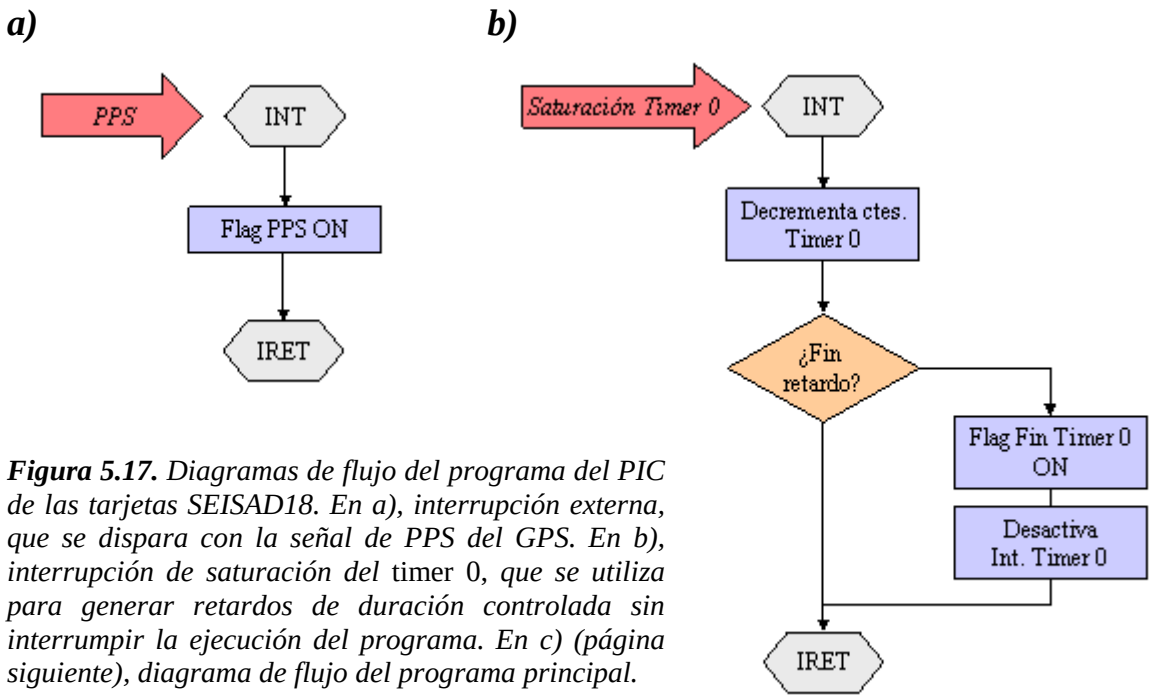


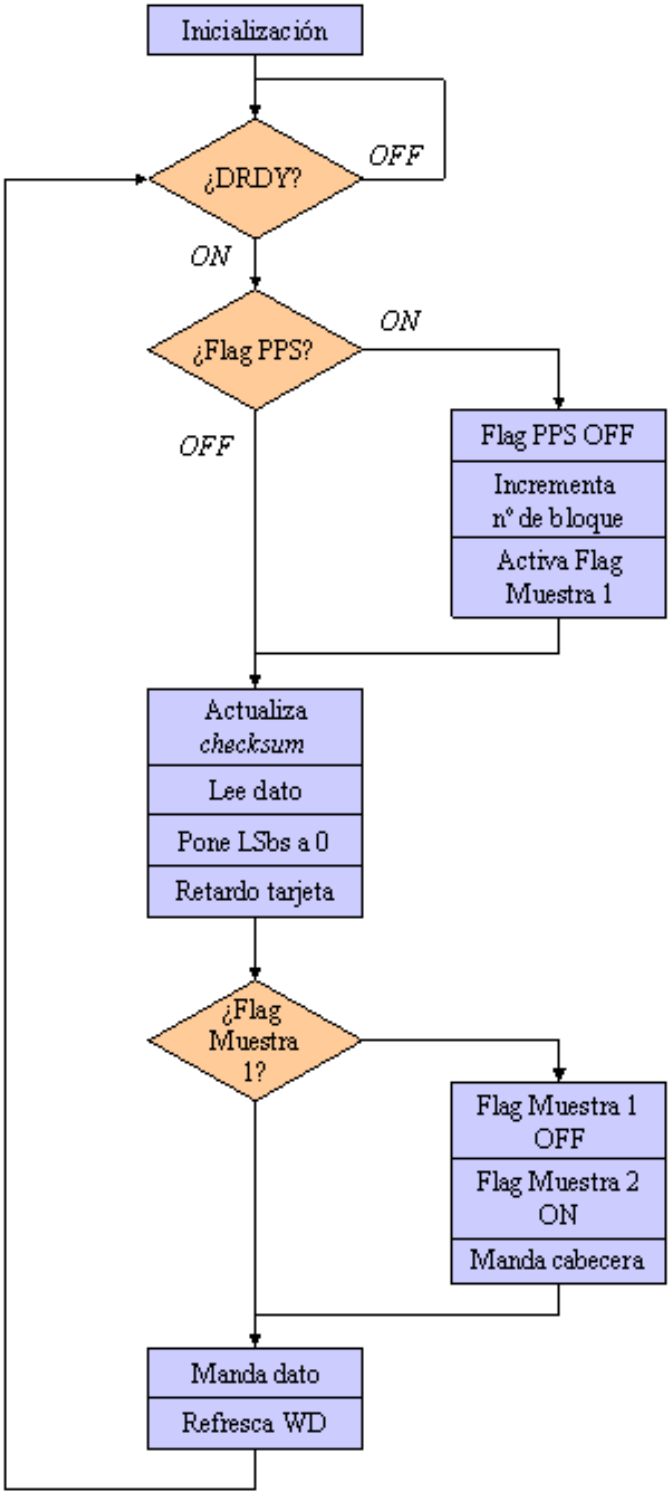
Figura 5.17. *Diagramas de flujo del programa del PIC de las tarjetas SEISAD18. En a), interrupción externa, que se dispara con la señal de PPS del GPS. En b), interrupción de saturación del* timer 0, *que se utiliza para generar retardos de duración controlada sin interrumpir la ejecución del programa. En c) (página siguiente), diagrama de flujo del programa principal.*

Para la lectura de los datos se utiliza la misma función que en el programa del Vesubio. Antes de realizar la lectura se actualizan las variables de *checksum*, que se enviarán con los datos para detectar posibles errores de transmisión. El programa utiliza tres variables de *checksum* que, por simplicidad en la programación, se gestionan de forma independiente, correspondiendo cada una de ellas a la suma de los bytes primero, segundo y tercero de todos los datos de un segundo. En la función de gestión del *checksum* se comprueba si el *Flag Muestra 1* está activado, en cuyo caso las tres variables se ponen a cero.

Como se muestra en el diagrama de flujo, después de la función de lectura los bits menos significativos de cada dato se ponen a cero, con el objeto de evitar confusiones con las cabeceras de trama de segundo. Cuando se definió el formato de las tramas de datos (Abril *et al.*, 2003) se eligieron como cabeceras de segundo caracteres cuyos bits menos significativos valían uno (55 55 55h, 77 77 77h, 99 99 99h y BB BB BBh para las tarjetas 0, 1, 2 y 3, respectivamente). Así, poniendo el bit menos significativo de cada dato a cero se garantiza que ninguna muestra tomará esas combinaciones de valores, lo cual puede ser útil si se pretende identificar las cabeceras en caso de pérdida de control del flujo de datos. Esta puesta a cero no implica una pérdida de información ya que, como se vio al presentar el conversor AD7710, la resolución efectiva es menor que los 24 bits nominales, por lo que los bits menos significativos no corresponden a variaciones reales de la señal sino a ruido de tipo electrónico.

La temporización para la asignación de turnos de transmisión en las distintas tarjetas A/D se realiza en las propias tarjetas. Para ello se genera un retardo programando el *timer 0* como se indicó anteriormente. El intervalo de tiempo depende de la posición lógica de la tarjeta en la línea serie o, lo que es lo mismo, del valor del código de tarjeta leído al principio del programa. Transcurrido el intervalo programado el PIC toma el control de la línea serie, para lo cual pone a uno la señal de control del transceptor RS-485 para pasarlo a modo de transmisión.

5.17.c)


Inicialización
¿DRDY?
OFF
ON
¿Flag PPS?
ON
OFF
Flag PPS OFF
Incrementa nº de bloque
Activa Flag Muestra 1
Actualiza checksum
Lee dato
Pone LSbs a 0
Retardo tarjeta
¿Flag Muestra 1?
Flag Muestra 1 OFF
Flag Muestra 2 ON
Manda cabecera
Manda dato
Refresca WD

La primera tarea que se realiza dentro de las funciones de transmisión de los datos es identificar la posición de la trama en la que se encuentra el programa. Como se muestra en la figura, se comienza comprobando el estado del *Flag Muestra 1*. En caso de que esté activado se envía la cabecera correspondiente a la muestra 1 del segundo, desactivando antes el *Flag Muestra 1* y activando el correspondiente a la segunda muestra del segundo (*Flag Muestra 2*). De este modo en la siguiente pasada por el bucle principal se mandará la cabecera correspondiente a la segunda muestra. En el diagrama de flujo de la figura sólo se ha incluido, por simplicidad, la rama correspondiente a la identificación de la primera muestra del segundo, pero en realidad el proceso es análogo para las cinco primeras muestras, cuyas cabeceras presentan formatos distintos (Abril *et al.*, 2003).

Tras el envío de la cabecera se pasa a la transmisión del dato propiamente dicho, después de lo cual se vuelve a dejar la línea serie en alta impedancia poniendo el transceptor RS-485 en modo de recepción. Por último se refresca el *watchdog timer* y se vuelve al inicio del bucle principal, a la espera de que los conversores tengan un nuevo dato listo.

5.6. Operación del sistema

5.6.1. Realización física del sistema: conectores, cajas y cables

Inicialmente, y de acuerdo con la filosofía adoptada para estos sistemas, se pensó en concentrar todos los elementos de cada antena en una sola caja. Sin embargo, el hecho de que los sensores se conecten al sistema de forma independiente (en lugar de a través de líneas serie comunes como en la antena del Vesubio) implicaba un problema de diseño: eran necesarios doce conectores, uno por cada canal, que en principio debían situarse en la maleta de la electrónica, a los que había que añadir los conectores de alimentación, GPS y comunicación externa. El elevado precio de los conectores de buenas prestaciones incrementaba sensiblemente el coste del sistema, por lo que se contempló la posibilidad de usar pasamuros estancos para los cables procedentes de los sensores. No obstante, esta solución tenía el problema de que implicaba un tiempo elevado de manipulación en el interior de la maleta cada vez que hubiera que realizar las conexiones, en condiciones frecuentemente adversas (humedad, viento, lluvia) que podrían afectar a la electrónica. Por otra parte, tanto la solución de los conectores militares como la de los pasamuros hacía necesario practicar numerosos agujeros en la maleta, lo cual no resultaba recomendable por distintas razones (disminución de estanqueidad, pérdida de rigidez, posibilidad de que los conectores situados en la parte inferior sufriesen golpes y se deteriorasen durante los desplazamientos). Por tanto, la solución que se adoptó finalmente fue utilizar una caja supletoria para la conexión de los sensores. Esta caja sería de menor tamaño, dado que no tendría que albergar ninguna electrónica, sino tan solo una placa de circuito impreso para el interconexionado entre los sensores y la maleta central. En la maleta la conexión se realizaría mediante un único conector militar, mientras que en la caja supletoria habría trece pasamuros (doce para la entrada de los cables de los doce sensores y uno de salida hacia la maleta) con la consiguiente disminución de costes respecto a la solución de un conector militar por cada canal. Esta solución además hacía innecesaria la manipulación interior de la maleta de electrónica en campo, ya que la conexión de los cables se realizaría en la caja supletoria, pudiendo permanecer cerrada la maleta durante todo el proceso de instalación (salvo, como se verá más adelante, durante la comprobación del funcionamiento mediante el sistema de LEDs de diagnóstico).

Las figuras 5.18 y 5.19 muestran varias fotografías de uno de los sistemas. En 5.18 se muestran los tres módulos que lo componen: la maleta que contiene la electrónica, la caja en la que se realizan las conexiones de los sensores y el receptor GPS. Como puede verse, la maleta utilizada como contenedor para la electrónica es del mismo tipo que la usada en la

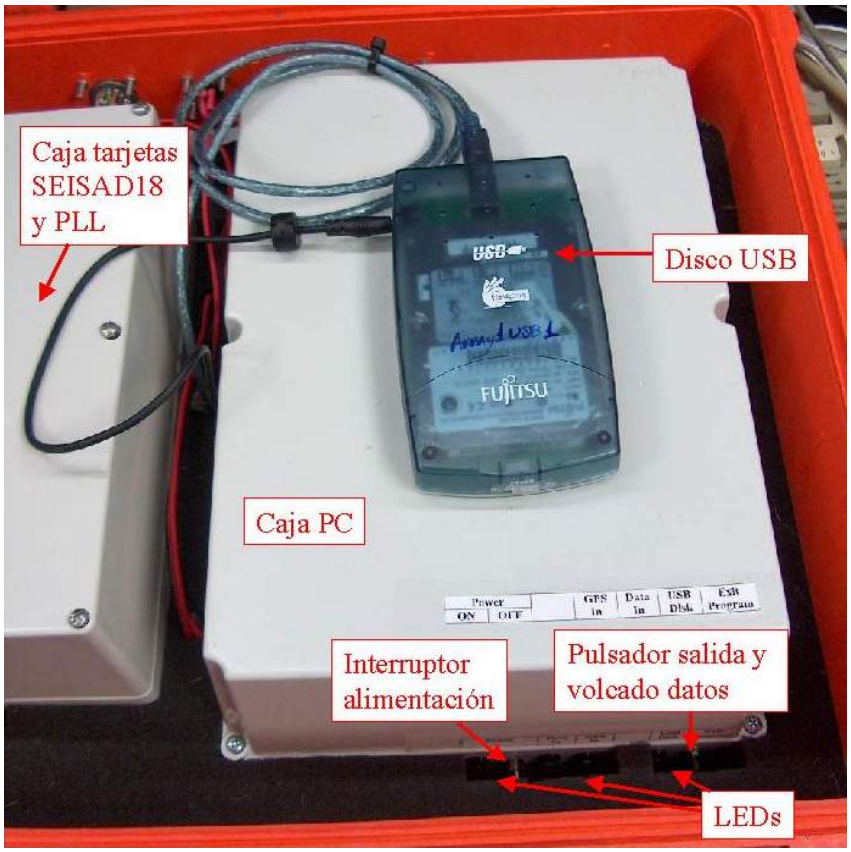
antena del Vesubio, que proporciona un buen aislamiento (protección IP-67) y resulta cómoda de transportar. Con objeto de mantener la estanqueidad, se utilizaron conectores militares para realizar las conexiones a los distintos elementos externos (ver figura 5.19.d). Estos son tres: las baterías o fuente externa de alimentación, el receptor GPS y la caja de sensores. El conector *ethernet* que se muestra en la figura no se utiliza en la versión del sistema que se describe aquí, pero se mantiene para permitir la posibilidad de acceder al disco duro interno o de comunicarse con otros módulos en futuras versiones de los programas basadas en *Windows*.

Las figuras 5.19.a y b muestran el interior de la maleta con las cajas de la electrónica cerradas y abiertas, respectivamente. En *a* pueden verse el disco USB y los interruptores, pulsadores y LEDs cuyo funcionamiento se describirá en la sección 5.6.4. En *b* se muestran las distintas tarjetas y componentes que aparecen en el diagrama de bloques de la figura 5.10 y cuyo funcionamiento se ha descrito anteriormente. En la caja del PC pueden verse, además de la propia tarjeta de PC industrial, el disco duro interno, la placa de fuente y la tarjeta de adaptación RS-485/RS-232. En la otra caja, la tarjeta PLL y las cuatro tarjetas de conversión A/D SEISAD18.

La figura 5.19.c muestra el interior de una de las cajas de conexión de los sensores, en la que se aprecian los pasamuros estancos y las fichas utilizadas para la conexión de los distintos cables, así como la manguera que va a la maleta a través de otro pasamuros.

Figura 5.18. *Fotografía de una de las nuevas antenas portátiles del IAG durante la campaña TOMODEC, en la isla Decepción (Antártida). En el centro se puede ver el módulo central abierto, en el que se distingue el disco externo USB y las dos cajas de la electrónica. El receptor GPS queda a la derecha, dentro del círculo rojo. A la izquierda del módulo central puede verse la caja de conexiones, de la que parten los distintos cables hacia los sensores. También se observan en la imagen las baterías, las bobinas de cable utilizadas y la tienda de campaña, que se usa como protección del personal durante las labores de mantenimiento y de los equipos durante su operación.*

a)



b)

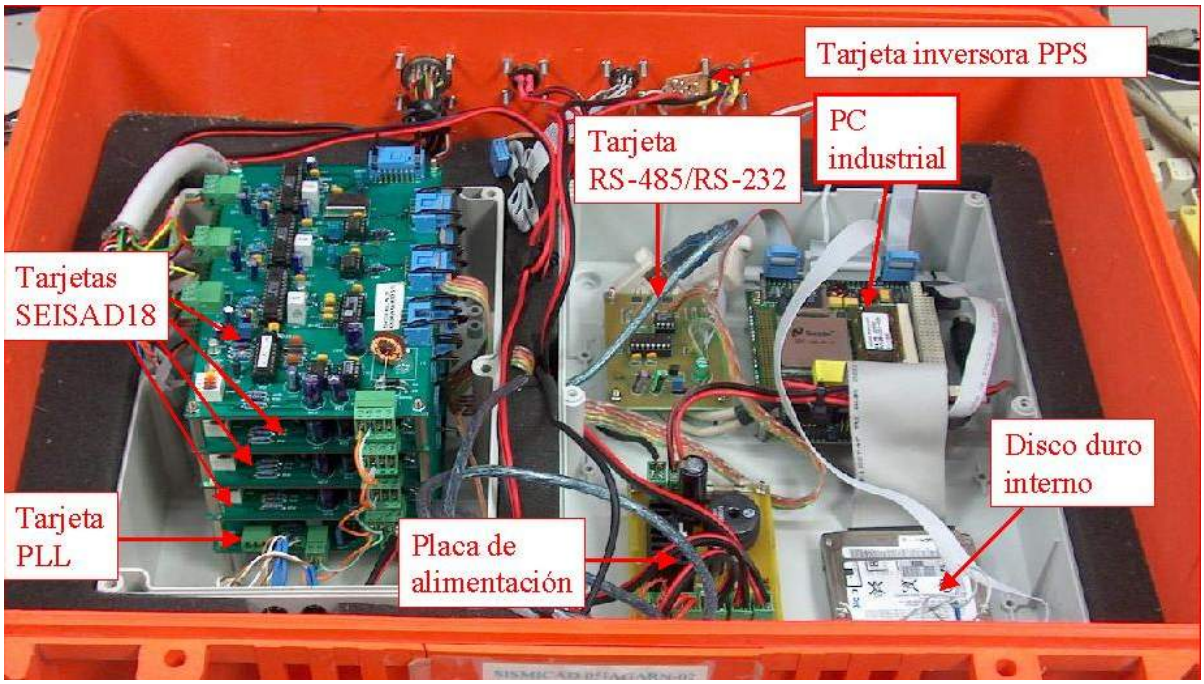
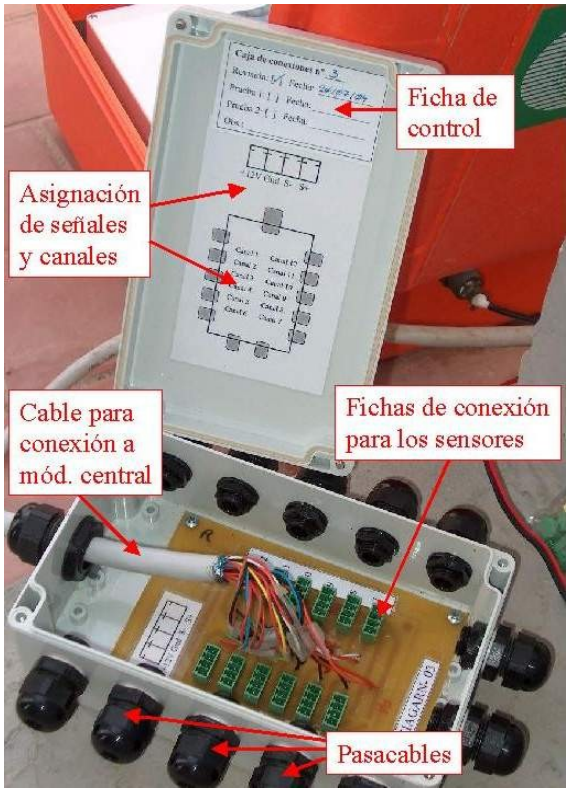
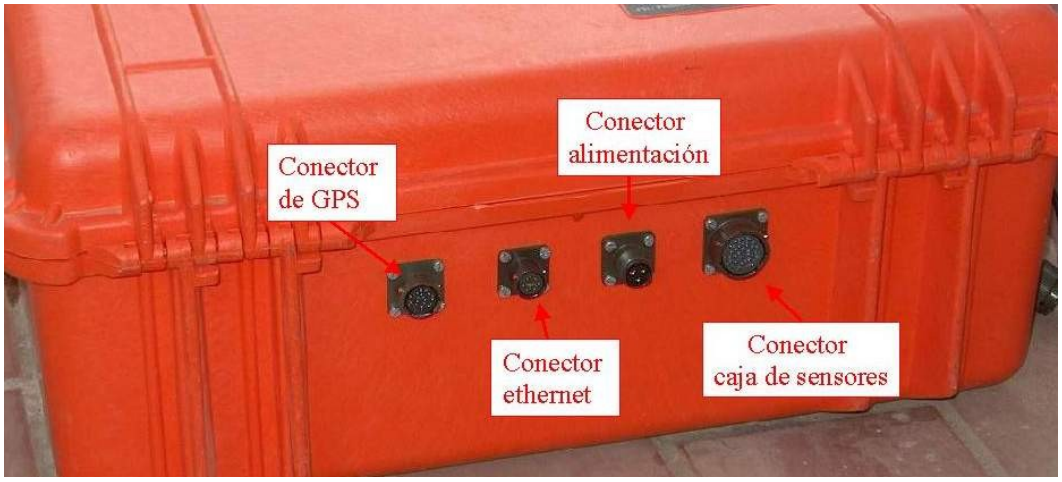


Figura 5.19. *Fotografías del módulo central con las cajas internas cerradas (a) y abiertas (b), caja de conexión de los sensores (c) y conectores militares del módulo central (d), en las que se especifica la ubicación de los componentes principales (c y d, en página siguiente).*

5.19.c)



5.19.d)



5.6.2. Implementación de los programas

En la sección 5.5 ya se han descrito los programas de los distintos módulos. A continuación se comentan algunos aspectos relativos a la implementación de los mismos en el sistema.

5.6.2.1. Sistema operativo

El sistema operativo instalado en el PC de control del sistema es el *Windows 98*. Dado que el programa de adquisición ARRAYS8 opera bajo MSDOS, el sistema está configurado para arrancar en modo MSDOS. En general no será necesario utilizar *Windows* en el funcionamiento normal del programa, aunque es posible que haya que hacerlo en ocasiones

puntuales (por ejemplo, para lanzar algún programa de apoyo que corra bajo *Windows*, como el de configuración de GPS). En ese caso deben seguirse las instrucciones del apartado 5.6.6.

El principal problema que planteaba el usar MS-DOS era que, igual que ocurre con QNX, este sistema operativo no gestiona los puertos USB. Sin embargo, a diferencia de aquel, es posible encontrar *drivers* de control de los puertos USB en Internet, muchos de ellos distribuidos como *freeware*. De los cuatro *drivers* que se probaron se escogió el *MHairu* (RTD Embedded Technologies, 2004) por su funcionamiento robusto, simplicidad de instalación y por suministrar documentación adecuada para su utilización. La instalación del *driver* en los sistemas se limita a la inserción de las siguientes líneas en el fichero de arranque CONFIG.SYS:

```
device=c:\mhairu\usbaspi1.sys
device=c:\mhairu\di1000dd.sys
```

Lógicamente, también es necesario asegurarse de que los ficheros especificados están en el directorio correcto.

La única limitación en el uso del disco externo USB bajo MS-DOS con este *driver* es que el sistema no detecta los dispositivos conectados 'en caliente'. Es decir, no es posible, como en *Windows,* conectar el disco externo mientras el equipo está funcionando. En nuestra aplicación, sin embargo, esta limitación no implica una pérdida significativa de operatividad, ya que normalmente el cambio de discos USB se realiza durante las labores de mantenimiento de los sistemas, en los que estos deben apagarse para sustituir las baterías.

5.6.2.2. Arranque del sistema

Los sistemas están configurados para comenzar a adquirir datos cuando se les suministra alimentación. En condiciones normales la verificación de la correcta operación se realiza a través de una serie de LEDs de colores, cuya secuencia de encendido durante el funcionamiento normal se describirá más adelante. También es posible conectar al PC de control un monitor, teclado y ratón para tareas de mantenimiento o resolución de problemas, para lo cual deben utilizarse los cables de expansión suministrados con los PCs industriales.

El procedimiento seguido en el arranque para lanzar el programa de adquisición se muestra en la figura 5.20. Como puede verse, el fichero de arranque AUTOEXEC.BAT hace una llamada, después de las inicializaciones habituales, a otro fichero de proceso por lotes, el ARRAYS.BAT. Tanto este fichero como todos a los que se llama desde él se encuentran en la unidad de disco externo USB. De este modo es posible hacer modificaciones en cualquiera de estos programas sin necesidad de acceder al disco duro interno. A cambio, esto exige ser cuidadosos con los cambios que se realicen en cualquiera de los programas o ficheros, ya que hay que tener en cuenta que dichas modificaciones deben realizarse en todos los discos USB que se utilicen en los distintos sistemas.

Como se muestra en la figura 5.20, antes de llamar al fichero de proceso por lotes ARRAYS.BAT, se lanza un programa llamado CHKBOOT.EXE, que no es más que un testigo de arranques del sistema. Cada vez que el sistema arranca se escribe la hora y fecha en un fichero LOG (CHKBOOT.LOG), lo cual permite seguir la pista de eventuales reinicios fortuitos, debidos por ejemplo a fallos en la alimentación.

Ya en el fichero ARRAYS.BAT, se lanza el programa INICARR2.EXE, cuya función es enviar a las tarjetas de adquisición SEISAD18 la secuencia con los parámetros de funcionamiento cuyo formato se describió en la sección 5.3.2. Estos parámetros son leídos del fichero ASCII de configuración SEISAD18.CFG, que se encuentra también en los discos externos USB y al que, por tanto, puede accederse fácilmente para modificar dichos parámetros.

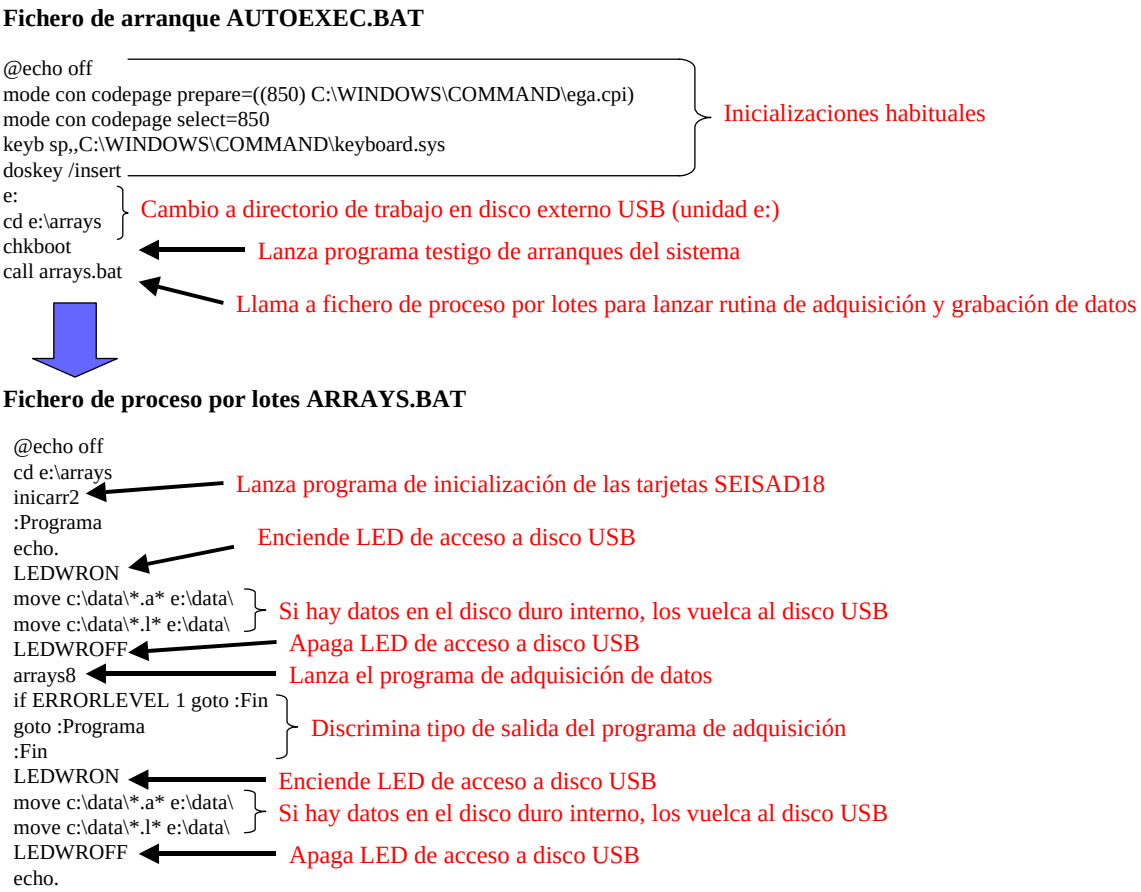


Figura 5.20. *Ficheros de arranque del sistema.*

Como puede verse en la figura 5.20, antes de lanzar el programa de adquisición ARRAYS8.EXE existen en el fichero ARRAYS.BAT varias líneas, cuya función es volcar, en caso de que existieran, los datos grabados en el disco duro interno por el programa de adquisición al disco USB. Durante el volcado de los datos se mantiene encendido el LED de acceso al disco USB.

Una vez lanzado el programa de adquisición, éste se ocupa principalmente de tres tareas: la atención de los puertos serie a los que están conectadas las líneas de datos de las tarjetas SEISAD18 (COM1) y GPS (COM2), la verificación de la integridad y formato de los datos y la grabación de estos (en caso de que sean correctos) o la inserción en los ficheros de un código de error (en caso de que no lo sean). Los ficheros de datos tienen una duración de una hora y se graban en el directorio C:\DATA del disco duro interno. La nomenclatura y formato de estos ficheros se describirán en la sección 5.6.8.

Según se explicó en la sección 5.5.2, el programa de adquisición del PC permite dos salidas distintas. La salida manual se produce cuando se presiona el pulsador de salida del programa (marcado como *Exit program* en la figura 5.19). En ese caso el programa pasa al sistema operativo un código de salida (*errorlevel 1*), que el fichero de proceso por lotes ARRAYS.BAT detecta y que hace que se salte al final del proceso, volcando previamente los datos que haya en el disco duro interno.

El segundo tipo de salida del programa de adquisición se produce cada seis horas, concretamente a las 00:00h, 06:00h, 12:00h y 18:00h de cada día. En estas salidas prefijadas

el programa pasa al sistema operativo un código de salida distinto al anterior (*errorlevel 0*), que permite al fichero de proceso por lotes ARRAYS.BAT distinguirlas de las salidas manuales. Como se aprecia en la figura 5.20, en este tipo de salidas el fichero de proceso por lotes vuelve a lanzar el programa de adquisición, volcando antes los datos del disco duro interno al disco USB. De esta forma se facilitan las operaciones de mantenimiento, ya que tras funcionamiento continuo del sistema durante varios días o semanas, el usuario sólo tiene que realizar una salida manual del programa y esperar a que se vuelquen al disco USB los últimos ficheros de datos. Como máximo, seis ficheros de una hora, que suponen un tiempo de volcado máximo de unos tres minutos, durante los cuales se detiene la adquisición de datos.

Este tipo de salida es equivalente a la que se produce cuando no se detecta señal de PPS del GPS. La primera comprobación que realiza el programa de adquisición del PC es la correcta recepción de la señal de PPS, ya que ésta es necesaria para la sincronización de los datos. Si no se detectan PPS en un intervalo de cinco segundos el programa sale al sistema operativo, pasándole el mismo código de salida que en las salidas prefijadas, de modo que el fichero de proceso por lotes ARRAYS.BAT entra en un bucle en el que sigue lanzando el programa de adquisición hasta que se detecte señal de PPS.

5.6.3. Puesta en marcha

El sistema está diseñado para permitir la operación en campo con la máxima simplicidad. Los pasos necesarios para su puesta en marcha son los siguientes:

1. Conexión del GPS.- El GPS se conecta al sistema a través del conector militar correspondiente (ver figura 5.19.d). El receptor debe situarse en el punto con mejor visibilidad GPS que permita el cable, cuya longitud es de cinco metros.
2. Conexión de los sensores.- La conexión de los sensores al sistema se realiza introduciendo los cables de los mismos a través de los pasamuros de la caja de conexiones (ver figura 5.19.c) y utilizando conectores de tornillo de cuatro vías. La asignación de señales y el orden de los canales en la caja de conexión puede verse en la figura 5.21. Para facilitar la instalación y mantenimiento las conexiones también están especificadas en la tapa de cada una de las cajas y en la placa de circuito impreso situada en su interior.
3. Suministro de alimentación al sistema.- Una vez alimentado el sistema con una o varias baterías de 12V se debe activar el interruptor de alimentación.
4. Comprobación del funcionamiento.- La verificación de la correcta operación del equipo se realiza mediante un sistema de LEDs que se describe en la siguiente sección. Es recomendable, además, grabar un fichero de prueba para comprobar el correcto funcionamiento de todos los canales. El fichero de prueba se puede visualizar en campo conectando el disco USB a un ordenador portátil y utilizando alguno de los programas de visualización (ver sección 5.6.10).
5. Volcado de datos.- La operación de volcado de datos se reduce a presionar el pulsador *'Exit program'* y esperar a que se apague el LED *'USB disk',* tras lo cual se puede apagar el interruptor de alimentación, cambiar el disco USB y repetir el proceso desde el paso 3 si se desea dejar el sistema funcionando de nuevo. Como se ha comentado en la sección precedente, el fichero de proceso por lotes ARRAYS.BAT transfiere automáticamente al disco USB, cada seis horas, los ficheros de datos que el programa de adquisición graba en el disco duro interno. De este modo el tiempo invertido en la operación de volcado manual se reduce considerablemente, limitándose al necesario para transferir, como máximo, las últimas seis horas de datos. Este tiempo es de aproximadamente tres minutos.

6. Es importante recordar que el disco USB se alimenta externamente, por lo que además de conectar el cable de alimentación, debe situarse el conmutador de alimentación que se encuentra en la parte trasera del propio disco en la posición *OFF*. De lo contrario el sistema no podría acceder al disco o lo haría de forma intermitente. Resulta especialmente importante realizar esta comprobación después de visualizar los ficheros de datos en otro PC, ya que en la mayoría de los casos el conmutador de alimentación debe situarse en la posición *ON* para acceder al disco desde otro ordenador y se corre el peligro de dejarlo así al volver a conectarlo al sistema.

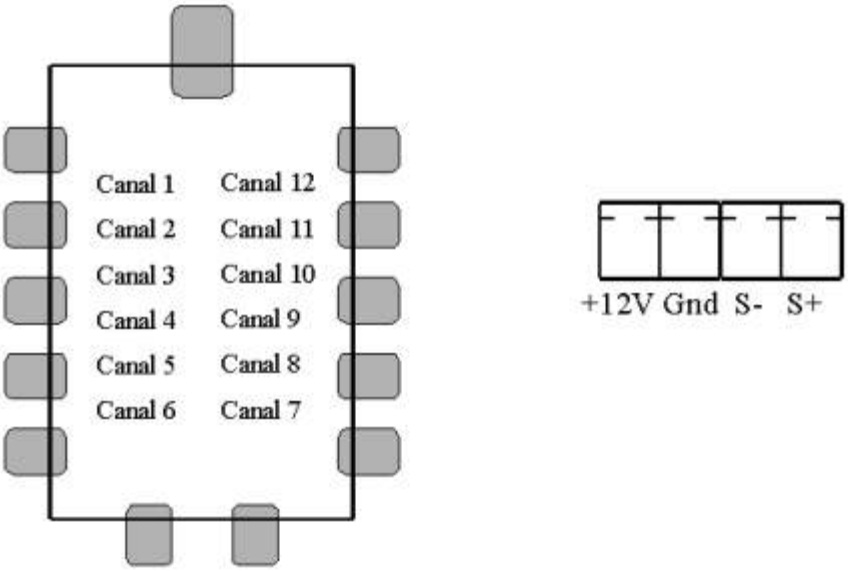


Figura 5.21. *Asignación de canales y señales en la caja de conexión de los sensores.*

5.6.4. Sistema de LEDs

El sistema de verificación del correcto funcionamiento de los módulos consiste en un conjunto de seis LEDs de colores, cuya disposición se muestra en la figura 5.22 y que se describe a continuación:

En la caja que contiene el PC:

- LED de alimentación (color rojo, rotulado como *'Power'*).- Permanece encendido mientras haya una fuente de alimentación conectada al sistema (normalmente una o varias baterías) y el interruptor de alimentación esté en posición *ON*. Si en estas condiciones este LED está apagado lo más probable es que el fusible de entrada de la fuente de alimentación esté fundido (ver figura 5.14).
- LED de recepción de datos GPS (color verde, rotulado como *'GPS In'*).- Se enciende cuando se está recibiendo y procesando correctamente la información de tiempo del GPS. En funcionamiento normal se encenderá durante un segundo de cada cuatro, lo cual no quiere decir que el programa de adquisición procese las tramas del GPS cada cuatro segundos. La gestión de estas tramas se realiza cada segundo, pero el LED se enciende con una frecuencia menor simplemente para ahorrar consumo. Por otra parte, el programa de adquisición procesa las tramas del GPS inmediatamente después que las de datos de las tarjetas de adquisición, por lo que si el LED *'GPS In'* no se enciende no quiere decir necesariamente que no se reciban tramas del GPS, sino más

probablemente que el formato de las tramas de datos no es correcto y el programa detecta el error antes de procesar las de GPS.

- LED de recepción de datos de las tarjetas de adquisición SEISAD18 (color amarillo, rotulado como *'Data In'*).- Se enciende cuando se están recibiendo y procesando correctamente los datos de las tarjetas de adquisición. Del mismo modo que ocurre con el LED *'GPS In'*, éste se enciende uno de cada cuatro segundos con objeto de ahorrar consumo, pese a que la recepción y procesado de los datos de las tarjetas de adquisición se realiza de forma continua.
- LED de acceso al disco USB (color rojo, rotulado como *'USB Disk'*).- Se enciende cuando se vuelcan datos desde el disco duro interno al disco USB.

En la caja que contiene las tarjetas SEISAD18 y PLL:

- LED de pulso de segundo (color verde, rotulado como *'PPS'*).- Parpadea cuando el PIC de la tarjeta PLL detecta la presencia de la señal de pulso de segundo del receptor GPS. Los receptores *Garmin HVS-35* utilizados en los sistemas comienzan a generar la señal de PPS cuando reciben buena señal de al menos tres satélites de la constelación GPS. Una vez que han comenzado a generarla no dejarán de hacerlo incluso en el caso de que se pierda la recepción de la señal de alguno o todos los satélites. Por tanto, una vez que el LED *PPS* comienza a parpadear no debe dejar de hacerlo a no ser que se interrumpa la alimentación del receptor GPS o de la placa PLL. En condiciones normales y en los emplazamientos habituales de los sistemas los receptores GPS tardarán poco tiempo (normalmente menos de un minuto) en comenzar a generar la señal de PPS. En el caso de que el sistema arranque por primera vez en un emplazamiento puede invertir más tiempo (varios minutos). Por otra parte, cuando el LED parpadea sólo permanece encendido el tiempo que dura el pulso (100 ms), por lo que puede resultar difícil de apreciar, especialmente en ambientes iluminados.
- LED de sincronismo de la tarjeta PLL (color amarillo, rotulado como *'Sync'*).- La tarjeta PLL se encarga de sincronizar la adquisición de datos en las cuatro tarjetas SEISAD18, para lo cual gestiona un oscilador a cristal controlado por tensión (VCXO). Como se explicó en la sección 4.5.6, el PIC de la tarjeta PLL está programado para realizar un ajuste suave de la frecuencia del VCXO, pero dependiendo del desajuste inicial del mismo en algunos casos deberá realizar ajustes más bruscos enviando pulsos de sincronismo a las placas de adquisición. Cada vez que el PLL genera uno de estos pulsos el LED *Sync* se enciende durante un segundo. Además, un pulso de sincronismo provoca la pérdida de algunos datos en las placas de adquisición, lo cual producirá un error de datos en el segundo en que se generó el pulso. Dicho error aparecerá en el fichero LOG, así como en el correspondiente fichero de datos, en el que los programas de visualización representarán el segundo completo como una línea de datos a cero. Por otra parte, como se ha dicho antes, un error en los datos hace que el sistema no llegue a procesar la trama del GPS, por lo que el LED *GPS In* sufrirá un retardo de un segundo respecto al LED *Data In* con cada nuevo pulso de sincronismo (ej.: si los LEDs *GPS In* y *Data In* se encendían simultáneamente en el arranque del programa de adquisición, después del primer pulso de sincronismo lo harán uno inmediatamente después que el otro). El número de pulsos de sincronismo que genera el PIC de la tarjeta PLL no es fijo, sino que depende del desajuste inicial del VCXO. Puede ir desde cero, si el VCXO está muy bien ajustado, hasta más de diez durante los primeros minutos de operación del sistema. También es posible que se genere algún pulso de sincronismo cuando cambian

bruscamente las condiciones ambientales (temperatura, humedad, presión,...) bajo las que opera el sistema.

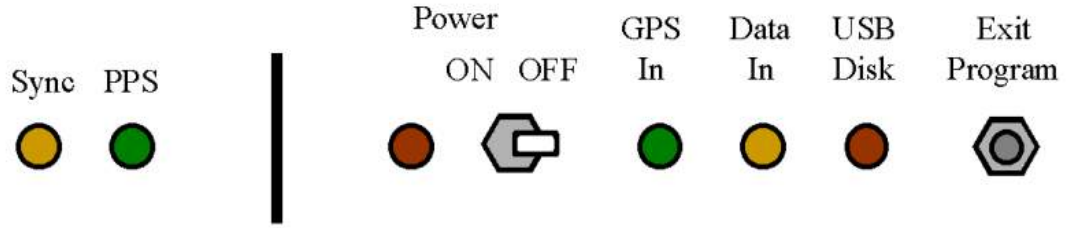


Figura 5.22. *Sistema de LEDs y pulsadores. A la izquierda de la línea, los LEDs de la caja de tarjetas de adquisición SEISAD18 y PLL. A la derecha, los de la caja del PC, junto con el interruptor de alimentación y el pulsador de salida de programa.*

La secuencia de encendido de los LEDs al arrancar el sistema, cuando el funcionamiento es normal, es la que se describe a continuación:

1. Al activar el interruptor de alimentación el LED *Power* debe encenderse y permanecer así hasta que el sistema se apague o la batería se consuma.
2. Los LEDs *GPS In, Data In* y *USB Disk* se encienden con intensidad media y permanecen así durante unos segundos. Este encendido temporal corresponde al arranque del PC.
3. Unos segundos después de iniciarse el arranque (normalmente menos de treinta), el LED *PPS* empezará a parpadear, señal de que el GPS ha comenzado a generar pulsos de segundo.
4. El LED *USB Disk* se encenderá un momento, mientras se comprueba que no existen datos en el disco duro interno. En caso de que sí existieran, dicho LED permanecería encendido durante el tiempo de volcado de estos datos al disco USB.
5. El sistema entrará en el programa de adquisición ARRAYS8, y los LEDs *GPS In* y *Data In* comenzarán a parpadear (un segundo encendidos y tres apagados).
6. Durante los primeros minutos de operación el LED *Sync* se encenderá esporádicamente, provocando retardos en el encendido del LED *GPS In* con respecto al de *Data In.*
7. Durante los procesos de volcado de datos automático (cada seis horas) o manual los LEDs *Data In* y *GPS In* se apagarán y el LED *USB Disk* permanecerá encendido mientras dura el volcado de datos al disco USB.

Además de servir para comprobar el correcto funcionamiento del equipo, este sistema de LEDs permite realizar un diagnóstico en caso de funcionamiento anormal.

5.6.5. Fichero de configuración

Los parámetros de operación del sistema pueden modificarse en el fichero de configuración SEISAD18.CFG. Para evitar la necesidad de acceder al disco duro interno el fichero de configuración se encuentra en los discos USB, de modo que se puede modificar con cualquier editor ASCII desde otro ordenador con puerto USB.

Como ya se ha comentado, las versiones operativas de los programas de los microcontroladores y PC fijan la resolución en 24 bits y la frecuencia de muestreo en 100 mps. Por tanto, en estas versiones los parámetros que se pueden modificar se limitan a la ganancia y al código del sistema. La ganancia permite tres valores (1, 4 o 16). El código de

sistema consta de dos caracteres alfanuméricos que sirven de identificación a cada módulo de *array*. Estos caracteres aparecerán en el nombre de los ficheros de datos y de incidencias, como se verá en la sección 5.6.8.

5.6.6. Arranque en modo *Windows*

Como ya se ha comentado con anterioridad, el sistema operativo instalado en el disco duro interno es el *Windows 98*, pero los módulos están configurados para arrancar en modo MSDOS. Para ello se ha modificado el parámetro 'BootGUI' del fichero de sistema MSDOS.SYS, cambiando su valor de 1 a 0. Si se necesita cargar *W98* aparentemente debería bastar con volver a poner el valor original de dicho parámetro y rearrancar el sistema, o bien teclear simplemente 'Win' desde la línea de comandos de MSDOS. En la práctica, sin embargo, al hacer cualquiera de las dos cosas el sistema se bloquea, debido a que el *driver* del puerto USB cargado para arrancar en modo MSDOS crea conflictos con el de *W98*. Por tanto, antes de lanzar *W98* hay que deshabilitar dicho *driver*. Así, la forma más sencilla de arrancar en modo *Windows* consta de los siguientes pasos:

1. Conectar teclado, ratón y monitor al PC industrial, utilizando los cables de expansión suministrados con los PCs *Lippert.*
2. Deshabilitar el *driver* del puerto USB para modo MSDOS. Para ello es necesario editar el fichero CONFIG.SYS del disco duro interno y comentar las líneas relativas a dicho *driver* utilizando el comando REM del MSDOS. Así, las líneas

   ```
   device=c:\mhairu\usbaspi1.sys
   device=c:\mhairu\di1000dd.sys
   ```

   quedarán como

   ```
   REM device=c:\mhairu\usbaspi1.sys
   REM device=c:\mhairu\di1000dd.sys
   ```

3. Reiniciar el sistema (apagando y encendiendo el interruptor de alimentación o mediante las teclas Ctrl+Alt+Del) para volver a cargar el fichero CONFIG.SYS.
4. Teclear 'Win' desde la línea de comandos.

Siguiendo estos pasos no es necesario modificar el parámetro 'BootGUI' del MSDOS.SYS, por lo que al reiniciar el sistema éste arrancará de nuevo en modo MSDOS. Una vez realizadas las operaciones necesarias en modo *Windows* es importante volver a dejar como estaban las líneas modificadas en el CONFIG.SYS, ya que de otro modo el sistema no detectaría el disco USB bajo MSDOS.

5.6.7. Configuración de los receptores GPS

Antes de utilizarse por primera vez, los receptores GPS deben configurarse con los parámetros de operación adecuados. Esta inicialización sólo debe realizarse una vez, por lo que lo más cómodo es hacerlo desde un PC convencional o portátil. Si se necesita hacerlo desde los propios sistemas éstos deben arrancarse en modo *Windows*, siguiendo los pasos descritos en el apartado anterior.

La configuración se realiza mediante un programa específico (GPSCFG.EXE) que se puede encontrar fácilmente en Internet (por ejemplo, en la página del Instituto de Física de la Tierra de la Universidad de Bergen[55]). La conexión entre PC y receptor GPS se realiza a través de uno de los puertos serie. Desde la interfaz del programa se debe especificar el puerto que se va a utilizar y seleccionar la opción de establecer conexión del menú de comunicaciones. El programa detecta automáticamente la velocidad de transmisión actual del receptor GPS y establece la comunicación con el mismo. Las opciones y valores de los distintos parámetros se seleccionan a través de una ventana del menú de configuración. Para nuestra aplicación se deben escoger las siguientes opciones:

- *Velocidad de comunicación:* 9600bps
- *Frases a transmitir por el receptor:* sólo el paquete de tiempo $GPRMC (ver formato en Garmin Corporation, 2000). El resto de las frases deben estar deshabilitadas.
- *Pulso por segundo:* habilitado

Tras seleccionar estas opciones es necesario enviar la configuración al receptor GPS, para lo cual se debe utilizar la correspondiente opción (*'Upload Configuration'*) del menú de configuración. Sin embargo, el receptor GPS no graba la nueva configuración hasta que no se reinicia. La forma más sencilla de hacerlo es interrumpir la alimentación durante unos segundos. Una vez reiniciado el receptor, es aconsejable comprobar que la configuración se ha grabado correctamente. Para ello debe restablecerse la comunicación y seleccionar la opción *'Download Configuration'* del menú de configuración. Si el proceso de cambio de parámetros se ha realizado correctamente deben aparecer en pantalla los nuevos valores y opciones seleccionadas.

5.6.8. Nomenclatura y formato de los ficheros de datos

Durante la operación del sistema se generan dos tipos de ficheros: los de datos y los ficheros LOG, en los que se anotan las incidencias más importantes durante el funcionamiento. El nombre de los ficheros responde a la siguiente nomenclatura:

MMDDHHmm.FSS, siendo:

MM: mes
DD: día
HH: hora
mm: minuto
F: código de tipo de fichero (A para los ficheros de datos, L para los ficheros LOG)
SS: código de sistema

El programa de adquisición ARRAYS8 escribe directamente los datos en los ficheros en el mismo orden en que los va recibiendo. Por tanto, los ficheros consisten en una sucesión de tramas de datos, cuyo formato se describe en Abril *et al.*, 2003, con la salvedad de que los bytes que según dicho documento deben contener la información de tiempo GPS están a cero. La información de tiempo GPS la inserta el programa ARRAYS8 al final de cada segundo de datos, después de leerla directamente del receptor GPS. Cada trama de datos procedente de una tarjeta SEISAD18 consta de doce bytes, de los cuales los nueve últimos corresponden al

[55] http://pcom3.geo.uib.no/Healy-2005/equipment/gps/garmin_gps-35-hvs/gps-35.html

dato propiamente dicho y los tres primeros a una cabecera que contiene distintos paquetes de información. En la figura 5.23 se muestra un fragmento de un fichero de datos, tal y como puede verse con un editor de ficheros binarios (*WinHex*[56]). La parte central muestra los datos en hexadecimal, mientras que la derecha lo hace en código ASCII. El fragmento incluye los últimos bytes del primer segundo de datos y los primeros del segundo segundo. Los paquetes más significativos se han marcado con las letras *a* a *f*, y su significado se describe a continuación:

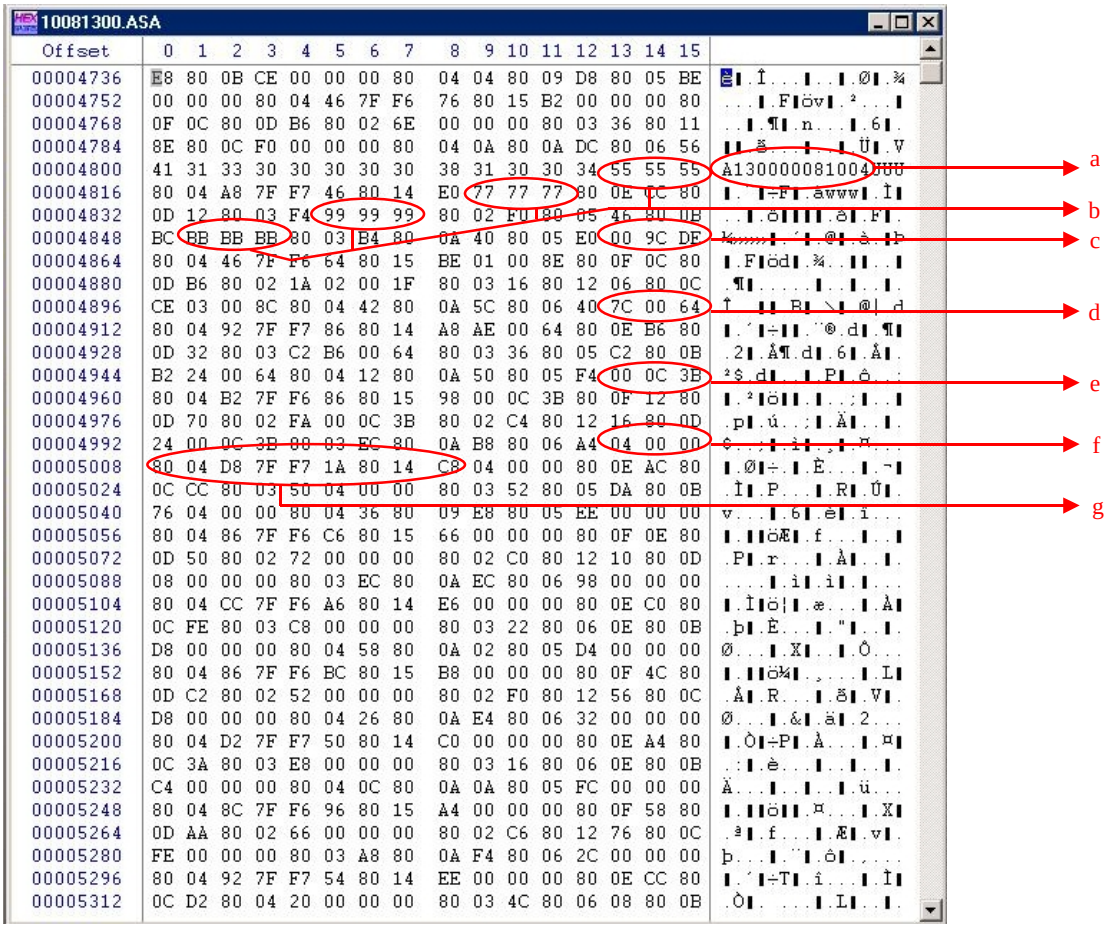


Figura 5.23. *Fragmento de un fichero de datos en el que se especifican los principales paquetes de datos y cabeceras.*

Información GPS.- La letra *a* corresponde a la información recibida, en código ASCII, del receptor GPS. Esta información se adjunta como último paquete de un fichero de datos, por lo que la de la figura corresponde al primer segundo del fichero. El paquete de GPS puede dividirse en tres campos:

- Información sobre el estado del receptor GPS: consiste en un solo byte, que puede tomar únicamente dos valores: 'A' o 'V'. 'A' corresponde a una recepción buena de la señal de los satélites, que asegura que la información de tiempo (incluyendo la señal

[56] En la página web http://www.x-ways.net/winhex/index-e.html puede consultarse información y descargar versiones de evaluación del editor hexadecimal WinHex.

de pulso de segundo) tiene la máxima precisión, mientras que 'V' implica una recepción defectuosa de la señal que conlleva una pérdida de precisión en las señales de tiempo.
- Hora GPS: consiste en seis bytes, correspondientes a la hora, minuto y segundo. En el ejemplo, las 13:00:00.
- Fecha: seis bytes, correspondientes al día, mes y año. En el ejemplo, ocho de octubre de 2004.

Nótese que, dado que la información presentada en el ejemplo corresponde al primer segundo de datos del fichero, los campos mes, día, hora y minuto coinciden con los que aparecen en el nombre del fichero (10081300.ASA). La extensión del nombre del fichero (ASA) implica que el código elegido para el sistema, introducido a través del fichero de configuración SEISAD18.CFG, es 'SA'.

Cabeceras de segundo.- El programa de los microcontroladores PIC de las tarjetas SEISAD18 marca con una cabecera específica los primeros datos adquiridos después de la detección de un pulso de segundo. Estas cabeceras son las que aparecen marcadas con la letra *b* en la figura 5.23 (tres bytes 55h, 77h, 99h o BBh para las tarjetas 0, 1, 2 y 3 del sistema, respectivamente). Se han elegido para las tres cabeceras de segundo valores cuyo último bit es uno, con el objeto de asegurarse de que no puedan confundirse con ningún dato. Para ello, según se vio en la sección 5.5.3, el programa de los microcontroladores pone a cero, antes de enviar cada dato, su bit menos significativo. Con esto no se pierde información, ya que la resolución efectiva de los conversores es menor que la nominal y los bits menos significativos no corresponden a información real sino a ruido electrónico del dispositivo.

Número de tarjeta y checksum.- La letra *c* de la figura 5.23 corresponde a la cabecera del segundo dato de la primera tarjeta SEISAD18. Por simplicidad sólo se ha señalado ésta, pero las cabeceras de los segundos datos de las otras tres tarjetas responden al mismo formato y se encuentran en las posiciones 4873, 4885 y 4897, respectivamente.

El primer byte de esta cabecera corresponde al número de tarjeta, teniendo en cuenta que la numeración empieza en 00h y termina en 03h. El segundo y tercer bytes son los dos primeros, de un total de tres, del código de *checksum*, un sistema que garantiza la integridad de los datos durante el proceso de transmisión. En este caso, el programa de los PICs de las tarjetas SEISAD18 calcula el *checksum* sumando por separado los bytes primero, segundo y tercero de cada dato. Los tres bytes que se obtienen como resultado se envían, de modo que el programa de recepción puede volver a realizar la operación y comprobar si el resultado que obtiene es el mismo. El programa ARRAYS8 no lo hace, porque empíricamente se ha verificado que no se producen apenas errores de este tipo (lo cual resulta lógico si se tiene en cuenta que la transmisión de datos entre tarjetas se realiza por cables de pocos centímetros de longitud), pero la implementación de esta comprobación sería sencilla.

Frecuencia de muestreo.- El primer byte de la cabecera marcada con la letra *d* en la figura 5.23 corresponde al tercer byte del *checksum*. Los otros dos bytes son la frecuencia de muestreo. En el ejemplo su valor es 0064h, que equivale a 100 en decimal.

Byte de sincronismo con GPS y bytes de número de bloque de datos.- Las tarjetas SEISAD18 pueden operar con o sin la presencia de señal de PPS, por lo que se ha reservado un byte (el primero de la cabecera *e* de la figura 5.23) para especificar si los datos están sincronizados (valor 00h) o no (valor 01h). Sin embargo, con la versión de programa actualmente operativa (SEISAD3.ASM) en los PICs de las tarjetas SEISAD18 es necesaria la presencia de señal de PPS, por lo que este byte siempre estará a 00h.

Los dos últimos bytes de la cabecera *e* corresponden al número de bloque de segundo (en el ejemplo, 0C3Bh, que equivale a 3131 en decimal). El programa de los PICs incrementa un contador de dos bytes cada vez que detecta un pulso de segundo, y transmite con los datos el valor de dicho contador, lo cual permite detectar la pérdida de bloques completos de datos.

Ganancia.- El primer byte de la cabecera *f* corresponde a la ganancia, que puede tomar los valores 1, 4 o 16 (1, 4 o 10 en hexadecimal). Los dos últimos bytes de esta cabecera, así como todos los del resto de las cabeceras de datos, no se usan y su valor es 00h. Estos bytes quedan reservados para la transmisión de otros parámetros en futuras versiones de los programas (ej.: parámetros de calibración de los sensores o información de tiempo GPS).

Dato.- La etiqueta *g* de la figura 5.23 no corresponde a una cabecera, sino a un dato propiamente dicho (en este caso, el quinto dato del segundo 13:00:01 de la primera tarjeta SEISAD18). Los tres primeros bytes del paquete corresponden al canal 0 de la tarjeta, los bytes cuarto al sexto, al canal 1 y los bytes séptimo a noveno, al canal 2. El primer bit de cada grupo corresponde al bit más significativo de la muestra, por lo que en el ejemplo los valores de los datos serían:

Canal 0: 8004D8h = 8389848 cuentas
Canal 1: 7FF71Ah = 8386330 cuentas
Canal 2: 8014C8h = 8393928 cuentas

Teniendo en cuenta que la resolución nominal de los conversores es de 24 bits, y que por tanto las muestras pueden tomar valores de entre 0 y 2^{24}-1 (es decir, 16777215) cuentas, puede verse que el valor de los datos de los tres canales corresponde aproximadamente al centro de escala de los conversores. Por otra parte, también puede apreciarse que los tres valores son pares, debido al valor cero del bit menos significativo de cada muestra comentado al hablar de las cabeceras de segundo.

5.6.9. Códigos de error

Según se comentó en la sección 5.5.2, el algoritmo implementado en el programa ARRAYS8 para gestionar la recepción de datos de las tarjetas SEISAD18 y del GPS consiste básicamente en comprobar los bytes recibidos por los dos puertos serie en el intervalo entre dos pulsos de segundo. Si al recibir el segundo pulso de segundo el número de bytes recibidos es correcto y su formato también, los escribe en el fichero correspondiente. Si hay algún error, ya sea en el número de bytes recibidos o en su formato, sustituye la información por un código de error. La forma en que el programa ARRAYS8 gestionaba los distintos errores ya se presentó en la sección 5.5.2, y a continuación se completa aquella descripción con la información más interesante desde el punto de vista del usuario de los sistemas. Los posibles errores son:

Error de formato de trama de datos.- Si el formato de la trama recibida no es correcto, se sustituyen todos los datos (incluyendo el paquete de información GPS) por diez caracteres '1'. Por simplicidad en la programación, el programa ARRAYS8 sólo comprueba la primera cabecera de segundo de la primera tarjeta (caracteres 55h 55h 55h), ya que empíricamente se ha comprobado que prácticamente todos los errores de formato provocan un error en dicha cabecera.

Error de búffer de datos.- Teniendo en cuenta que con las versiones actualmente operativas de los programas la frecuencia de muestreo está fijada en 100 mps, el número total de bytes de datos (es decir, sin contar los de la información GPS) en un segundo debe ser de:

$$100 mps \times 12 \frac{bytes}{muestra} \times 4 tarjetas = 4800 bytes \qquad [5.2]$$

El programa ARRAYS8 comprueba, cuando detecta un nuevo pulso de segundo, el número total de bytes recibidos por el primer puerto serie, y si ese número no coincide con el correcto sustituye todos los datos (incluyendo el paquete de información GPS) por diez caracteres '2'.

Error de formato de trama GPS.- Si el formato de la trama de información GPS no es correcto se sustituye dicha trama por diez caracteres '3'. En este caso, por tanto, los datos de las tarjetas SEISAD18 correspondientes al segundo actual estarán presentes en el fichero, y sólo faltará la información GPS.

Por simplicidad sólo se verifica que el primer carácter de la trama es correcto ($), ya que, como se explicó en la sección 5.6.7, los receptores GPS se configuran para que envíen únicamente tramas de tiempo ($GPRMC), y por tanto no existe posibilidad de confusión con otro tipo de tramas. El formato de la trama de tiempo $GPRMC puede consultarse en Garmin Corporation, 2000.

Error de búffer GPS.- Se produce cuando, al recibir un nuevo pulso de segundo, el búffer de recepción del puerto serie COM2 no contiene los caracteres necesarios para poner la marca de tiempo al final del bloque de datos. En ese caso se sustituye en el fichero la trama de información GPS por diez caracteres '4'.

Este es un error relativamente frecuente, ya que el receptor GPS no siempre manda la trama de tiempo $GPRMC completa dentro del segundo correspondiente. La trama $GPRMC consta de un total de 74 caracteres, de los cuales el programa ARRAYS8 se queda con 13, según se ha visto en la descripción del formato de los ficheros de datos. Para minimizar el número de errores, la condición de 'no error de búffer' no es que se haya recibido, al llegar el nuevo pulso de segundo, la trama $GPRMC completa, sino sólo los 59 primeros bytes, que son los que contienen toda la información necesaria para poner la marca de tiempo en el fichero.

5.6.10. Visualización de los datos

La visualización y procesado de los datos puede realizarse usando programas específicos o con sistemas de análisis ya existentes, en cuyo caso es necesario contar con los programas de transformación de formatos de los ficheros de datos.

Se han desarrollado dos programas de adaptación de formatos (Almendros, comunicación personal): el ARRSASEI.EXE convierte los ficheros generados por los sistemas de array SEISAD18 a ficheros compatibles con el *software* de análisis de datos sísmicos SEISAN. El ARRSASIS, por su parte, transforma los ficheros de datos a otros que pueden visualizarse y procesarse con el programa BANDAS, muy utilizado para el análisis de los ficheros de datos de los antiguos módulos de dieciséis bits.

Por otra parte, también se ha desarrollado el programa CHKARR4.EXE para visualizar los ficheros de datos generados por los sistemas de array de una manera rápida, sin necesidad de transformarlos previamente. Resulta, por tanto, especialmente útil para realizar una comprobación en campo del funcionamiento de todos los canales de adquisición. Existe una copia de este programa en el directorio de datos de cada uno de los discos USB, de modo que se puede ejecutar para realizar las pruebas de canales sin necesidad de transferir ficheros entre varios directorios. Se ejecuta desde la línea de comandos de MSDOS, por lo que se puede utilizar tanto desde los propios sistemas - cuando se tiene un teclado y monitor conectados - como desde otro PC. Es posible ejecutar en este programa algunos comandos para facilitar la visualización de los datos (compresión de los ejes horizontal y vertical, cambio de canales, salto de pantallas,...), que pueden consultarse presionando la tecla 'a' desde la ventana principal del programa.

5.6.11. Programas de prueba

Con el objeto de posibilitar la verificación del correcto funcionamiento de los distintos componentes del sistema, se han realizado algunos programas de prueba. Todos ellos se pueden ejecutar desde la línea de comandos de MSDOS, para lo cual es necesario conectar previamente al sistema un teclado y monitor. Los programas son los siguientes:

- P_BOTON.EXE.- Prueba el pulsador de salida del programa y volcado de datos (*Exit program'*). Al ejecutarlo aparecerá en pantalla una vez por segundo el texto 'No pulsado', cambiando a 'PULSADO' cuando se presiona el pulsador.

- P_PPS.EXE.- Verifica la correcta recepción de los pulsos de segundo del GPS en el PC. Si la recepción es correcta parpadearán el mensaje 'PPS' en la pantalla y los LEDs *GPS In, Data In* y *USB Disk.*

- LEDSON y LEDSOFF.EXE.- Comprueban el funcionamiento de los LEDs conectados al puerto paralelo (*GPS In, Data In* y *USB Disk*). El primero los enciende y el segundo los apaga.

- P_COMRX2.EXE.- Comprueba la recepción de caracteres por los puertos serie. Permite verificar el formato y número de datos recibidos, tanto procedentes del receptor GPS como de las tarjetas SEISAD18. Para comprobar la información recibida del GPS debe seleccionarse el puerto serie COM2 y velocidad de transmisión 9600 bps. La selección de parámetros se realiza siguiendo las instrucciones que aparecen en pantalla al ejecutar el programa. Para realizar la comprobación es suficiente con esperar unos segundos antes de salir del programa pulsando cualquier tecla. El programa grabará los datos recibidos en el fichero DATA.BIN, en el directorio raíz de la unidad D: (definida como disco virtual). El GPS manda los datos en formato ASCII, por lo que se puede comprobar la información recibida con el editor de MSDOS desde la propia línea de comandos. Dado que los receptores GPS están configurados para enviar únicamente paquetes de tiempo ($GPRMC), el fichero debe consistir en una sucesión de tantos paquetes de este tipo como segundos haya estado funcionando el programa. Para comprobar los datos recibidos de las tarjetas SEISAD18 debe seleccionarse el puerto serie COM1 y velocidad de transmisión 115200 bps. El formato del fichero recibido debe coincidir con el descrito en la sección 5.6.8. Hay que hacer notar que, para poder realizar esta comprobación, las tarjetas SEISAD18 deben haber sido inicializadas previamente. Si no se modifica nada del procedimiento de

arranque del sistema esta inicialización se realiza automáticamente mediante el programa INICARR2.EXE, incluido dentro del fichero de proceso por lotes ARRAYS.BAT, según se describió en la sección 5.6.2.2.

El listado completo de estos programas se incluye en el anexo III.A, incluido en el CD adjunto.

CAPÍTULO 6

APLICACIONES Y RESULTADOS

En este capítulo se describirán las aplicaciones y se discutirán los principales resultados obtenidos a partir de datos registrados con las antenas descritas en este trabajo. Las fuentes son principalmente artículos de varios autores publicados en revistas especializadas y presentaciones en congresos. En la discusión se ha prestado especial atención a los aspectos relativos a la operación de los sistemas, a veces por encima de los resultados científicos aportados por las publicaciones.

En el momento de la redacción de esta memoria de tesis la antena del Vesubio aún no se encontraba operativa, por lo que no se incluye una sección dedicada a ella. Así pues, el capítulo constará de dos secciones, en las que se describirán los resultados relativos a la antena del Gran Sasso (6.1) y a las antenas portátiles del IAG (6.2). Dentro de esta última hay a su vez varios apartados, dedicados a los principales entornos volcánicos en los que se han utilizado los nuevos sistemas: la isla de São Miguel, en las Azores (6.2.1), el Teide, en Tenerife (6.2.2), la Isla Decepción, en la Antártida (6.2.3), el volcán Copahue, en Argentina (6.2.4) y el Volcán de Fuego de Colima, en México (6.2.5).

El fundamento de las técnicas de procesado de datos de antenas sísmicas está explicado en el capítulo 1. Una descripción detallada de los distintos métodos de análisis queda fuera del ámbito de esta tesis, pero puede consultarse en la bibliografía que se referencia al citar las técnicas usadas en cada aplicación. Para obtener una visión global pueden consultarse Rost y Thomas (2002) o Almendros (1999), que ofrece un interesante compendio de los principales métodos de análisis y una comparativa de los mismos para su aplicación en áreas volcánicas.

6.1. La antena del Gran Sasso

Como se ha visto en los capítulos anteriores, las antenas del Gran Sasso y del Vesubio gestionan un número elevado de canales y pueden considerarse dispositivos complejos en comparación con las antenas portátiles del IAG. Durante la etapa de desarrollo una de las consecuencias de esta mayor complejidad de diseño fue la imposibilidad de realizar pruebas de laboratorio con la configuración máxima del sistema. La filosofía de módulos independientes con capacidad para un número relativamente reducido de canales adoptada en las antenas portátiles del IAG facilitó la realización de pruebas en su configuración completa, y por tanto en condiciones mucho más próximas a las que se encontrarían en aplicaciones reales. Sin embargo, en el caso de la antena del Gran Sasso la configuración máxima en las pruebas de laboratorio constaba de cuatro estaciones conectadas a una misma línea serie más el correspondiente PC nodal, conectado a su vez a un servidor a través de una red de área local. Era lógico, por tanto, que una vez instalada la antena en su emplazamiento definitivo, y antes de intentar aprovechar los registros para cubrir determinados objetivos científicos, se realizaran ensayos orientados a calibrar la calidad de los datos y las prestaciones reales del dispositivo. Debido a dificultades fundamentalmente económicas y logísticas la instalación se llevó a cabo de forma escalonada y durante un periodo largo de tiempo. A medida que se instalaban las sucesivas líneas serie se fueron realizando distintas pruebas para verificar el correcto funcionamiento del sistema, pero hasta hace unos meses no se publicaron resultados con configuraciones próximas a la inicialmente prevista. Así, Saccorotti *et al.*, 2006, estudiaron las características y prestaciones del *array*, mediante el análisis de una selección de terremotos registrados tanto por la antena como por la Red Sísmica Nacional Central (RSNC)

italiana. La configuración del array en el momento de registrar los datos que se analizan en el artículo era la que se muestra en la figura 6.1, consistente en trece estaciones operativas de las veintiuna previstas, con una apertura de unos 550 m y una separación media entre los sensores de unos 90 m.

El primer paso en el análisis fue determinar el patrón de radiación para la configuración utilizada. Como puede verse en la figura 6.2, el diagrama presenta un pico central bastante acusado, indicando una respuesta adecuada (ver sección 1.2.3), aunque con una marcada orientación NNW-SSE, que es consecuencia de la distribución espacial en la dirección ENE-WSW de los sensores. Esto implica que la resolución de la antena será menor para ondas que se propagan a lo largo de esa dirección determinada.

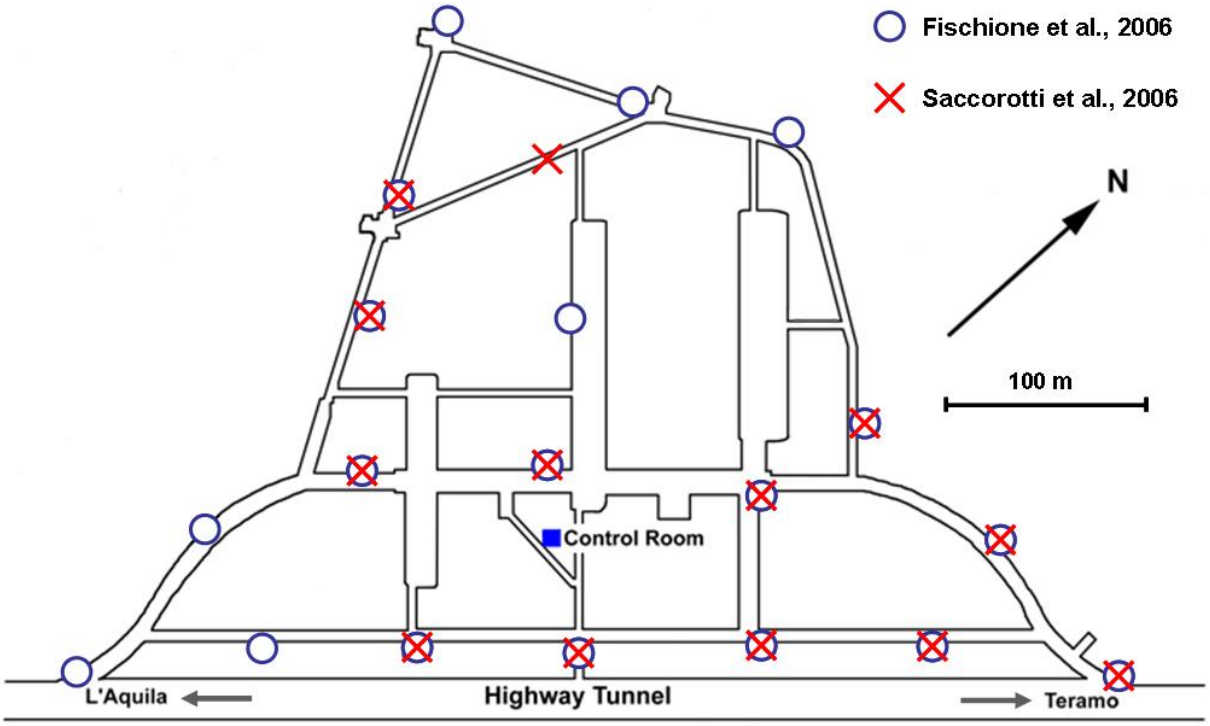


Figura 6.1. *Distribución espacial de los sensores en los artículos citados en el texto. Con cruces rojas se marcan las estaciones operativas en la configuración de Saccorotti et al., 2006. Con círculos azules, las de Fischione et al., 2006, y Tronca et al., 2007 (modificado de Fischione et al., 2006).*

También puede verse en la figura 6.2 que aparecen picos de *aliasing* para valores del número de onda *k* del orden de 20 ciclos/km, que corresponden a longitudes de onda de unos 300 m. Las frecuencias de Nyquist para las ondas P y S se pueden calcular a partir de este valor y los límites inferiores de velocidades aparentes, que corresponden a ángulos de incidencia con la vertical de 90º (es decir, ondas incidentes en el plano horizontal del *array*) y que por tanto coinciden con las velocidades de las ondas P y S en el medio. En esta zona el valor aceptado para dichas velocidades es de 4 km/s y 2.3 km/s, respectivamente, lo cual da frecuencias de Nyquist de unos 13 Hz (ondas P) y 6 Hz (ondas S).

La segunda parte del análisis se centró en el estudio de la coherencia de la señal. Como se ha apuntado anteriormente, al aplicar cualquier método de análisis de datos de *array* se asume que la señal registrada en cada uno de los puntos de muestreo de la antena es la misma, excepto por el retardo asociado con la propagación a través del dispositivo. Para poder aceptar esta premisa es necesario que la señal mantenga de forma significativa la coherencia en todos los sensores. El ruido sísmico, por el contrario, debe mantener niveles bajos de coherencia. Los autores tomaron un conjunto de registros conteniendo llegadas de ondas P y S con diferentes SNRs y calcularon la coherencia promedio de la antena en función de la frecuencia para ambos tipos de ondas, siguiendo el método descrito en Saccorotti y Del Pezzo, 2000. Sus conclusiones fueron que, en general, las ondas P mantienen correlaciones significativas en el rango de frecuencias entre 1 y 10 Hz, mientras que la coherencia de las ondas S se mantiene dentro de valores aceptables entre 1 y 6 Hz. Del mismo modo, se realizaron análisis similares para registros de ruido con distintas ventanas y a distintas horas del día y de la noche para todas las parejas de estaciones independientes. Tan solo se registraron valores significativos de coherencia para frecuencias menores de 1 Hz, lo cual se interpretó como ruido asociado a la actividad del mar. Para el resto de las bandas de frecuencia los niveles de correlación satisfacen totalmente la hipótesis de ruido sísmico no correlacionado espacialmente requerida por los métodos de procesado para antenas sísmicas.

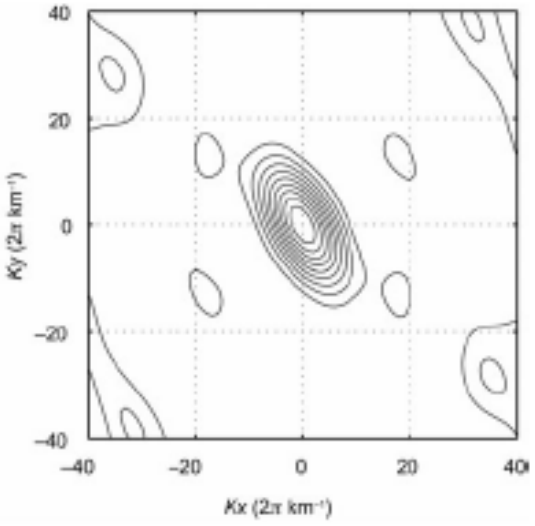


Figura 6.2. *Patrón de radiación de antena para la configuración espacial de trece estaciones mostrada en la figura 1 (de Saccorotti et al., 2006).*

Una vez determinados el patrón de radiación y los niveles de coherencia, se pasó al análisis de señales reales. Para el cálculo de los parámetros de propagación (azimut y lentitud aparente) se utilizó la técnica MUSIC (de *MUltiple SIgnal Classification technique*, ver Schmidt, 1986; Goldstein y Archuleta, 1987 y 1991), que ofrece mejores resultados que otros métodos para señales con bajas SNRs.

Se seleccionaron 26 terremotos registrados por la antena entre los meses de noviembre de 2002 y septiembre de 2003, con magnitudes de entre 2.3 y 3.5 y distancias epicentrales de entre 20 y 140 km.

La figura 6.3 muestra los registros de uno de estos terremotos, ocurrido el 31 de agosto de 2003, junto a los espectros de lentitud obtenidos. La figura 6.4 muestra las formas de onda obtenidas mediante el apilamiento de los registros de las distintas estaciones, en las que puede apreciarse el gran incremento en la SNR que se consigue.

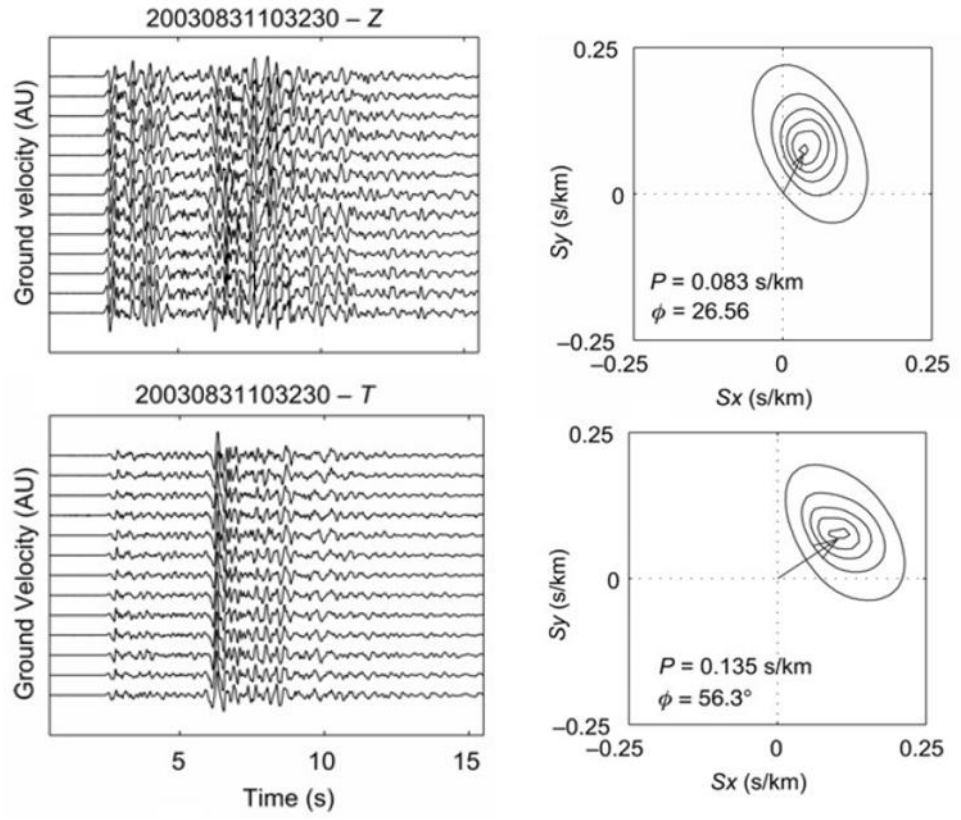


Figura 6.3. *Registros de un terremoto tomados con la antena del Gran Sasso. Se muestran los registros verticales (arriba) y transversales de las trece estaciones operativas en el momento del terremoto, en agosto de 2003. A la derecha se muestran los espectros de lentitud obtenidos mediante técnicas de antena, especificándose en cada caso los valores calculados para el azimut ϕ y parámetro del rayo* |P| *(de Saccorotti et al., 2006).*

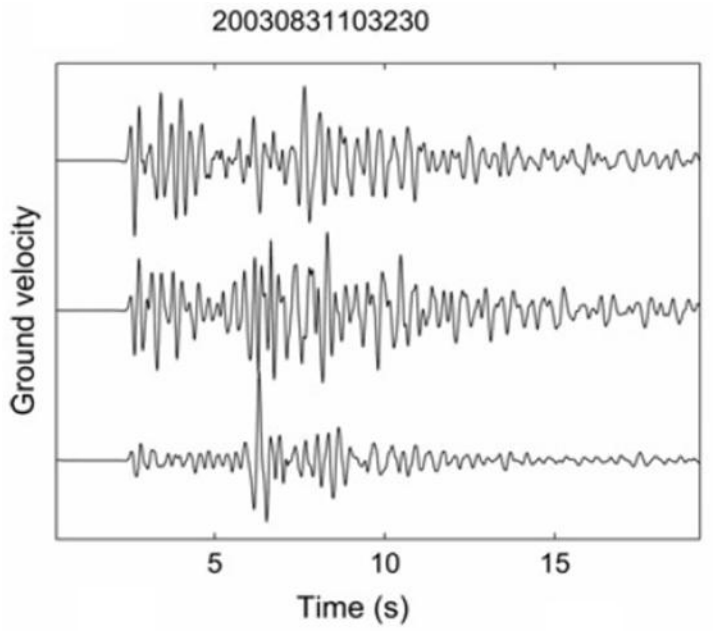


Figura 6.4. *Formas de onda vertical (arriba) y horizontales obtenidas por apilamiento de los registros de las trece estaciones para el mismo terremoto de la figura 6.3. Se observa el gran incremento en la SNR obtenido (de Saccorotti et al., 2006).*

Las prestaciones de la antena en la detección y localización de señales asociadas a la sismicidad local se hacen aún más evidentes en señales con valores especialmente bajos de la SNR. Un ejemplo es el terremoto de la figura 6.5, ocurrido el 27 de diciembre de 2002 y que fue registrado por el *array* con valores muy pequeños de amplitud, haciendo imposible llevar a cabo una estimación visual de la llegada de la onda P. Sin embargo, un análisis multicanal proporciona estimaciones fiables de la lentitud, que permiten llevar a cabo una localización del evento.

Como puede verse en la figura, tanto la potencia del espectro de lentitud como la coherencia multicanal antes de la llegada de la onda P son muy bajas, lo que confirma que el ruido de fondo no está correlacionado entre los distintos sensores. La llegada de la onda P origina un incremento en ambos parámetros, lo cual puede ser utilizado como herramienta en sistemas automáticos de detección en tiempo real.

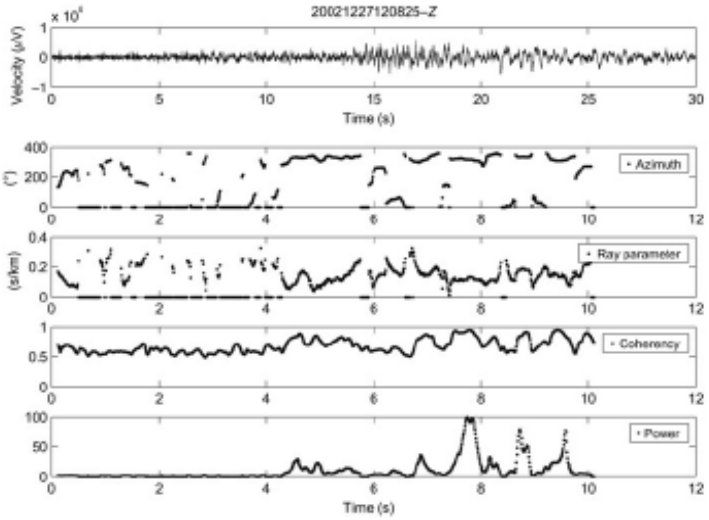


Figura 6.5. *Evolución temporal de los parámetros de propagación para el terremoto de magnitud M_d=2 de la figura, ocurrido el 27 de diciembre de 2002 con distancia epicentral de unos 80 km. De arriba abajo, sismograma vertical, azimut de propagación, parámetro del rayo, coherencia multicanal y potencia del espectro de lentitud. En los dos últimos parámetros se observa un súbito incremento en torno a los 4 s, correspondiente a la llegada de la onda P (de Saccorotti et al., 2006).*

Entre otros análisis realizados por los autores a todo el conjunto de eventos seleccionados se llevó a cabo una relocalización de los mismos para verificar que los resultados eran coherentes con las localizaciones de la RSNC. Las estimaciones de los epicentros se realizaron a partir de los azimuts obtenidos, calculando las distancias multiplicando las diferencias S-P por un factor empírico $\gamma = 7.5$. Los errores en las localizaciones se determinaron teniendo en cuenta las incertidumbres en los cálculos del azimut y considerando una variación de γ de ± 0.5. Los resultados fueron que en todos los casos los epicentros tomados del catálogo de la RSNC estaban incluidos dentro de los intervalos de confianza de las localizaciones obtenidas a partir de los datos de la antena. La figura 6.6 muestra las discrepancias entre ambos conjuntos de localizaciones en función de la distancia epicentral y del parámetro del rayo, tanto para las ondas P como para las S. Se observa que las discrepancias en las localizaciones con ondas S raramente superan los 10 km, y son mucho más consistentes que las obtenidas a partir de datos de ondas P. Esto puede parecer sorprendente, ya que lo normal sería que un análisis de lentitud basado en ondas S diera resultados menos fiables, originados por efectos de sitio en los registros y por orientaciones imprecisas de los sensores. Sin embargo, este factor queda compensado por el aumento de la SNR asociado a las ondas S, generalmente de mayor amplitud que las P, y a la pérdida de precisión en las medidas de lentitud cuando la velocidad aparente aumenta. Este último hecho queda ilustrado en la figura 6.6, en la que puede observarse que los mayores errores en las localizaciones están asociados a ondas P que se propagan con valores bajos del parámetro del rayo.

Teniendo en cuenta estas limitaciones, las localizaciones obtenidas pueden considerarse bastante satisfactorias y, lo que es más importante en el contexto de esta tesis, totalmente coherentes con los resultados obtenidos independientemente a partir de datos de la

RSNC. Por otra parte, teniendo en cuenta que la configuración operativa cuando se tomaron los datos constaba, como se ha dicho, de sólo trece estaciones, los autores vaticinan resultados más precisos en las medidas cuando el número de sensores aumente. Esta mejora irá asociada a dos factores: por una parte, el aumento en la apertura de la antena incrementará la resolución, especialmente para ondas con velocidades aparentes altas. Por otra, el mayor número de estaciones reducirá la contribución del ruido. Este incremento del número de estaciones no se limita a la instalación de la totalidad de las previstas inicialmente, puesto que en la actualidad se están excavando nuevos túneles en los LNGS que permitirán extender la geometría del *array* más allá de los límites impuestos a día de hoy por la estructura de los laboratorios.

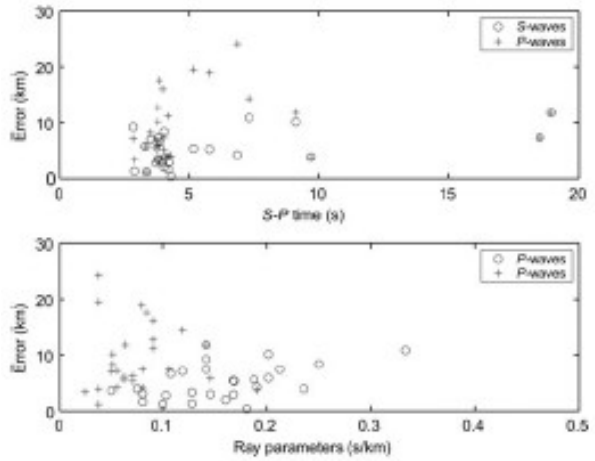


Figura 6.6. *Discrepancias entre las localizaciones realizadas por Saccorotti et al., 2006, con datos del array del Gran Sasso, y las de la RSNC. Las diferencias se muestran en función de la distancia epicentral, expresada en términos de tiempo S-P (arriba) y del parámetro del rayo, en ambos casos tanto para las ondas S como para las P (de Saccorotti et al., 2006).*

Del mismo modo, Fischione *et al.* (2006) realizaron análisis similares con datos del periodo enero 2005 – febrero 2006 registrados por el *array* con la configuración espacial de 19 estaciones que se muestra en la figura 6.1. La figura 6.7 muestra un evento registrado en febrero de 2006, con M = 2.8 y distancia epicentral de 20 km. Este estudio se centró en la caracterización de los parámetros de las señales registradas con la antena para, en función de ellos, desarrollar un sistema de detección automática de eventos.

Para ello los autores comenzaron realizando un análisis detallado del ruido de fondo generando espectrogramas de veinte días de duración y estudiando la correlación espacial del ruido en función de la frecuencia. Los resultados, en acuerdo con los comentados de Saccorotti *et al.*, 2006, se aplicaron a la determinación de la banda de frecuencias para el filtrado previo al análisis de los datos, que se fijó en 2-8 Hz.

Posteriormente se determinó de forma empírica una escala de magnitud-duración para la antena, según la expresión:

$$M = B\log_{10}(d) + A \qquad [6.1]$$

Para la estimación de los parámetros A y B se partió de un conjunto de unos cincuenta eventos registrados por el dispositivo y cuyas magnitudes se conocían previamente (figura 6.8). Los eventos, con valores de magnitud de entre 1 y 5, se tomaron del boletín del INGV. También de forma empírica se determinó el parámetro γ para la conversión del tiempo S-P en distancia epicentral (figura 6.9).

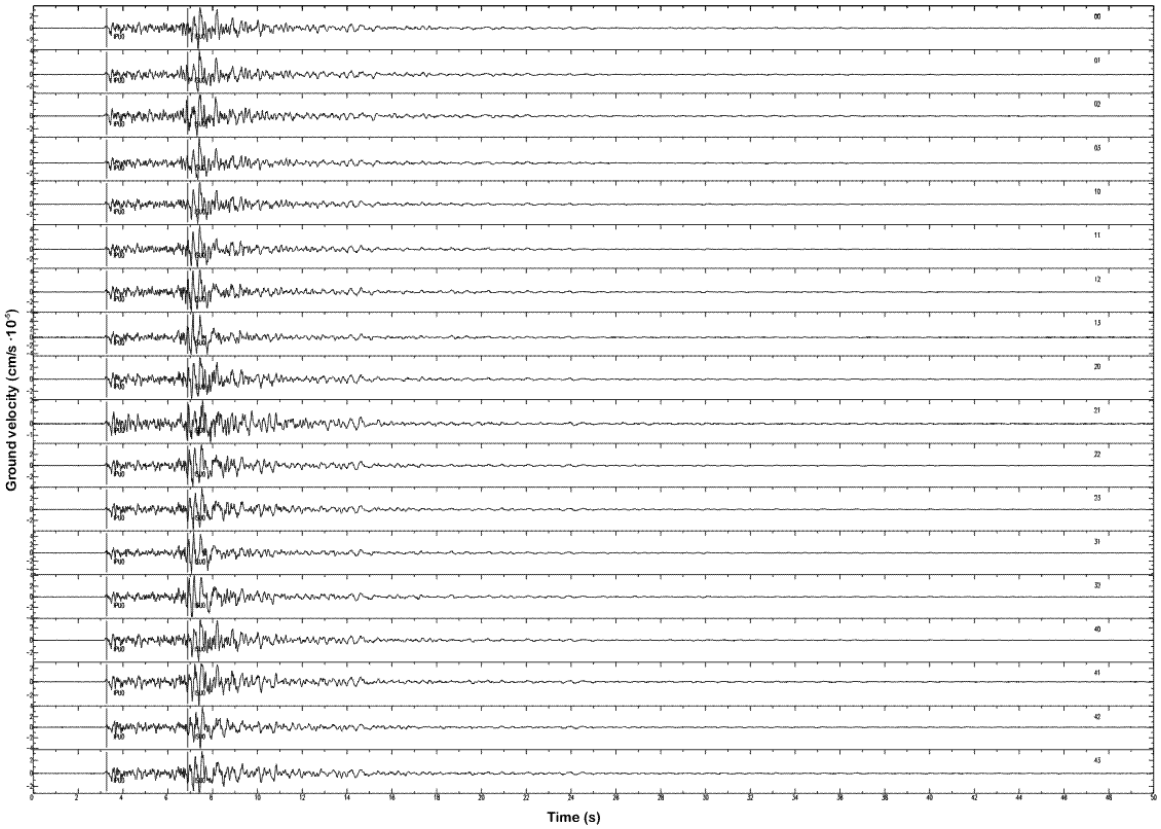


Figura 6.7. *Registros verticales de la antena del Gran Sasso correspondientes a un terremoto de M = 2.8 y distancia epicentral de 20 km ocurrido el 19 de febrero de 2006, en los que se observa una elevada SNR (de Fischione et al., 2006).*

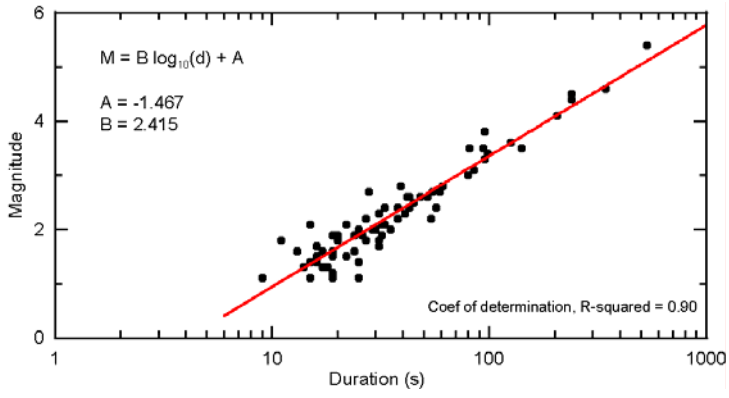


Figura 6.8. *Estimación empírica de los parámetros A y B para la fórmula de cálculo de la magnitud (de Fischione et al., 2006).*

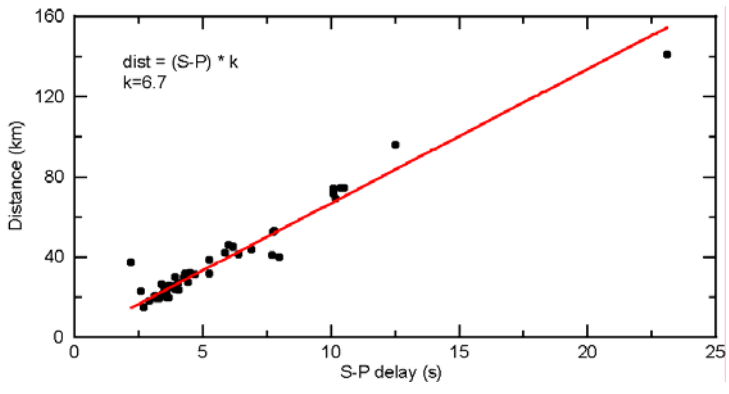


Figura 6.9. *Estimación empírica del parámetro γ para la determinación de la distancia epicentral a partir de la diferencia de tiempos S-P (de Fischione et al., 2006).*

Todos estos resultados se utilizaron para definir los parámetros necesarios para implementar un sistema de detección automática de eventos, cuya estructura básica se muestra en la figura 6.10. Como puede verse, se parte de los ficheros de datos almacenados en el servidor de la sala de control según el proceso descrito en el capítulo 3, al que se denomina en la figura *Server PC 01*. El programa de detección automática corre en un segundo servidor remoto (*Server PC 02*) desde el que accede periódicamente a través de Internet al primer servidor para descargar los últimos ficheros de datos de todas las estaciones operativas. Este segundo servidor convierte los datos a formato SAC y calcula, mediante la técnica MUSIC, el espectro de lentitud, el azimut de propagación y los parámetros de correlación. El algoritmo de detección automática se basa en los resultados descritos de Saccorotti *et al.*, 2006, según los cuales estos parámetros aumentaban significativamente en presencia de señales sísmicas coherentes (figura 6.5). Así, se establece un umbral de disparo para la potencia del espectro de lentitud de la señal, almacenando como ficheros de evento aquellos datos que superen dicho umbral y descartando el resto. Posteriormente se realiza un procesado y análisis manual de los datos, en el que se lleva a cabo una selección más cuidadosa de los eventos, se marcan las fases P y S y se vuelve a aplicar, con parámetros más restrictivos, el algoritmo MUSIC para obtener los parámetros epicentrales y generar un mapa de localizaciones.

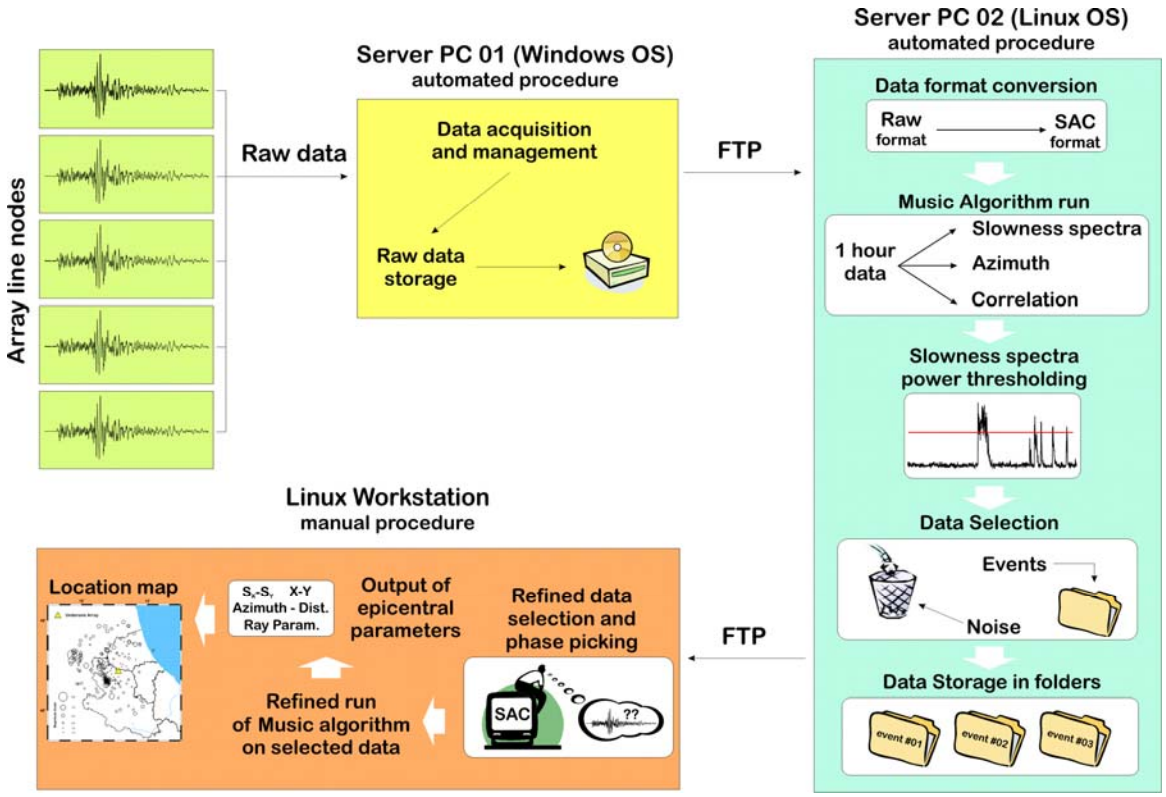


Figura 6.10. *Diagrama de bloques del sistema de análisis de datos implementado en la antena del Gran Sasso. Los dos primeros bloques corresponden a la parte descrita en esta tesis. La detección automática de eventos, almacenamiento y procedimiento refinado de análisis fueron implementados por Fischione et al., 2006, de donde se ha tomado la figura.*

Entre otras conclusiones extraídas por los autores destaca el hecho de que las características de la antena, tanto en lo referente a sus prestaciones como a su emplazamiento privilegiado en los LNGS, han permitido disminuir el umbral de detección de eventos hasta una magnitud de entre 1 y 1.5 en el rango de distancias epicentrales de 20-50 km. Los registros, incluso los correspondientes a menores magnitudes, mantienen una elevada SNR que permite su localización. Durante el periodo enero-noviembre de 2005 el algoritmo de detección automática descrito detectó unos 460 eventos, cantidad comparable a los registrados por la red sísmica regional gestionada por el SSN (*Servizio Sismico Nazionale*) en el periodo durante el cual estuvo operativa (1992-2000, ver De Luca *et al*., 2000).

El mismo equipo ha realizado un estudio sobre la sismicidad reciente en la región central de Italia a partir de datos del *array* (Tronca *et al*., 2007) con la misma configuración de 19 estaciones de Fischione *et al*., 2006 (figura 6.1). La figura 6.11 muestra la sismicidad regional registrada por la antena durante el periodo agosto 2005-febrero 2006. La distribución de terremotos resulta consistente con la de los eventos registrados por la red sísmica regional del SSN durante su periodo operativo.

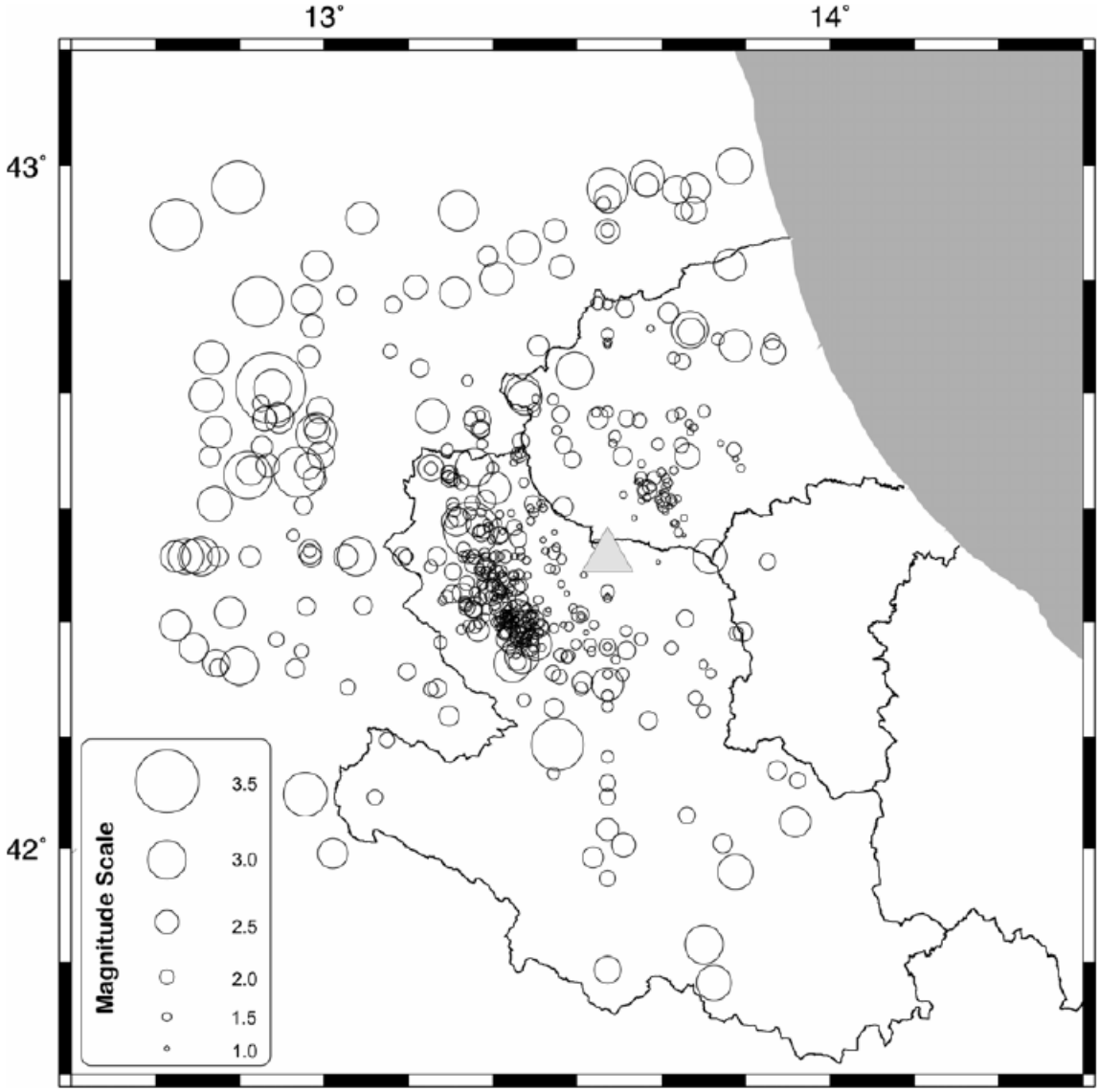


Figura 6.11. *Mapa de localización de los eventos sísmicos regionales registrados por la antena del Gran Sasso en el periodo agosto 2005-febrero 2006 (de Tronca et al., 2007).*

Los autores realizaron una serie de análisis a los datos registrados para continuar calibrando las prestaciones de la antena. Entre ellos, aplicaron el procesado descrito antes (figura 6.10) a un conjunto de datos sintéticos y compararon los resultados con los obtenidos a partir de datos reales para cuantificar los errores de localización cometidos por el método (figura 6.12). Como era previsible, la diferencia se hace mayor para valores menores de la lentitud aparente, debido a que para ángulos de incidencia acusados el retardo de fase entre los elementos de la antena se hace muy pequeño y por tanto difícil de medir con precisión. Este resultado está en concordancia con los obtenidos por Saccorotti *et al.*, 2006 (figura 6.6).

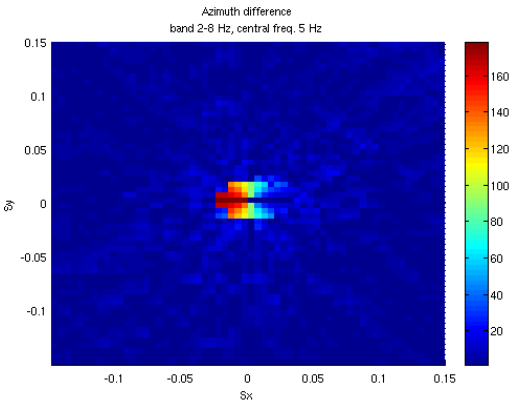


Figura 6.12. *Discrepancias entre el azimut calculado para señales sintéticas y para señales reales en función del vector lentitud aparente. Las mayores diferencias se producen para valores altos de la velocidad aparente, en concordancia con los resultados de Saccorotti et al., 2006 (de Tronca et al., 2007).*

En términos cuantitativos, los autores calcularon que para lentitudes aparentes mayores de unos 0.03 s/km (que corresponden a velocidades aparentes del orden de 30 km/s) los errores en azimut son de aproximadamente 20º, reduciéndose a unos 5º-10º para velocidades aparentes de 10 km/s. Para los valores típicos de la sismicidad regional (profundidades de 0-20 km y distancias epicentrales menores de 100-140 km), las velocidades aparentes esperadas en éste área para las ondas P son del orden de 5-8 km/s. Por tanto, resulta aceptable asignar un error azimutal de 10º para la mayoría de los eventos. Una excepción la constituyen los terremotos cuya profundidad es comparable o mayor que la distancia epicentral, en cuyo caso el elevado ángulo de incidencia de las ondas implica una pérdida significativa de resolución en las medidas de lentitud aparente. En estos casos, se puede realizar una estimación más consistente del valor de la lentitud aparente aplicando los análisis a las fases S, cuyas velocidades de propagación son más lentas.

Aparte de estas discrepancias, los autores descubrieron errores sistemáticos entre los azimuts calculados mediante técnicas de *array* a partir de los datos de la antena y los que se esperarían a partir de las localizaciones del catálogo de la Red Sísmica Centralizada. Las características de estos errores descartan que sean debidos tanto a incertidumbres en las localizaciones del catálogo como a errores del instrumento o de las técnicas de procesado, por lo que los autores los atribuyeron a efectos de propagación debidos a variaciones laterales de velocidad. Esta circunstancia evidencia la necesidad de contar con un modelo de la estructura local adecuado cuando se afronta la localización de terremotos regionales mediante técnicas de *array*, extremo éste que ya se ha apuntado en otros trabajos en distintos entornos (ver, por ejemplo, Lin y Roecker, 1996; Xu *et al.*, 1996). Tronca *et al.*, 2007, definieron de forma empírica un modelo simple que permite en la mayoría de los casos predecir y corregir estas desviaciones dentro de un cierto margen de error. No obstante, para mejorar los resultados es recomendable disponer de un modelo 3-D que permita incrementar la precisión y fiabilidad de las predicciones de las trayectorias de los rayos. Gracias al aumento de cobertura que la Red Nacional Centralizada ha experimentado en los últimos años en la región central de Italia, en un futuro próximo se dispondrá de completas bases de datos de tiempos de viaje que permitirán generar imágenes tomográficas de la estructura superficial e interna de la región.

Según todo lo visto, puede concluirse que la antena del Gran Sasso constituye una herramienta única para monitorizar la actividad sísmica en los Apeninos Centrales, una de las áreas con mayor potencial sismogenético de Italia. Su emplazamiento subterráneo ha permitido reducir el umbral de detección de eventos a M=1.0 para distancias epicentrales de hasta 40-50 km, manteniendo una elevada SNR incluso para los eventos de magnitudes menores. El registro continuo de sismogramas multicanal de bajo ruido podría ayudar además a revelar la existencia de otras señales de origen tectónico, como las relacionadas con terremotos lentos de las que ya se han descubierto casos en el área (Crescentini *et al*., 1999) o episodios de trémor no volcánicos profundos de los que se tiene noticia en los últimos años en diversas regiones sísmicamente activas del mundo (Obara, 2002; Rogers y Dragert, 2003; La Rocca *et al*., 2005; Nadeau y Dolenc, 2005).

6.2. Las nuevas antenas portátiles del IAG

La fase de diseño y prueba de las nuevas antenas portátiles del IAG concluyó a principios de 2003. Los primeros módulos estuvieron listos unos meses después y desde entonces se han utilizado profusamente en diversos entornos volcánicos con resultados muy satisfactorios. En conjunto puede decirse que los nuevos sistemas se han mostrado como una herramienta potente y resolutiva que permite identificar, analizar, clasificar y relacionar con procesos de fuente señales que por otra vía serían muy difíciles de detectar y tratar. Como se verá a continuación, han resultado especialmente útiles para el tratamiento de eventos de tipo LP y VT, explosiones volcánicas y, sobre todo, señales de tipo trémor, que en algunos casos no se habían detectado antes en las regiones que se han estudiado. La compacidad y facilidad de mantenimiento y operación de los nuevos módulos se han revelado como características muy ventajosas durante las campañas de campo, y gracias a la alta resolución y el carácter continuo de los registros se ha facilitado notablemente la fase de procesado y mejorado los resultados.

A continuación se describen algunas de las campañas en las que se han utilizado, resaltando los aspectos más importantes relacionados con su despliegue y operación y los resultados que se han obtenido a partir del análisis de los datos.

6.2.1. Isla de São Miguel

El archipiélago de las Azores está situado en el Océano Atlántico, en el punto de unión entre las placas Norteamericana, Africana y Eurasiática (figura 6.13). Consta de nueve islas habitadas, de las cuales la de São Miguel es la de mayor extensión, y de varios islotes de menor tamaño. Diversos estudios geotérmicos (Gandino *et al.*, 1985), de deformación (Jonsson *et al.*, 1999) y gravimétricos (Montesinos *et al.*, 1999) indican una gran complejidad de la estructura superficial e interna de la isla de São Miguel, que se caracteriza por la presencia de sistemas de fracturas con direcciones preferentes NW-SE y E-W, calderas como las de Fogo y Furnas en la intersección de estos alineamientos tectónicos y fuentes termales y fumarolas distribuidas a lo largo de los principales sistemas de fallas. Los complejos volcánicos activos más importantes de la isla son Furnas, Sete Cidades y Fogo. Furnas, el volcán más joven, y Fogo, se formaron en grandes erupciones pumíticas (Moore, 1990). Las últimas erupciones se remontan a los años 1563 y 1640.

En la actualidad se registran emisiones difusas de CO_2 y existe actividad hidrotermal dentro del cráter de Furnas y en el flanco norte del de Fogo. Desde principio de los años ochenta operan en São Miguel dos plantas de energía geotérmica asociadas al campo geotérmico de Ribeira Grande. La actividad sísmica en el área es intensa, con más de 3000 terremotos registrados y localizados al año. Los eventos están asociados tanto a los principales alineamientos tectónicos a escala regional como a enjambres sísmicos originados por la actividad del sistema geotermal (ver, por ejemplo, Dawson *et al.*, 1985).

Entre los meses de abril y julio de 2003, dentro del marco del proyecto europeo *e-Ruption*[57], se instaló en la isla una red sísmica entre la que se contaban tres antenas sísmicas implementadas con los nuevos sistemas portátiles del IAG, si bien no con la configuración descrita en esta memoria. La electrónica y estructura de los módulos utilizados allí era básicamente la misma que la que se ha presentado, pero con diferencias en la distribución interna de los distintos subsistemas, en los contenedores y conectores utilizados y en el programa de adquisición del PC (Abril *et al.*, 2004). La localización geográfica de las antenas

[57] *e-Ruption. A satellite telecommunication and Internet-based seismic monitoring system for volcanic eruption forecasting and risk management* (www.e-ruption.info/main.htm).

y del resto de las estaciones de la red sísmica, así como la disposición geométrica de los sensores en los *arrays*, se muestra en la figura 6.14.

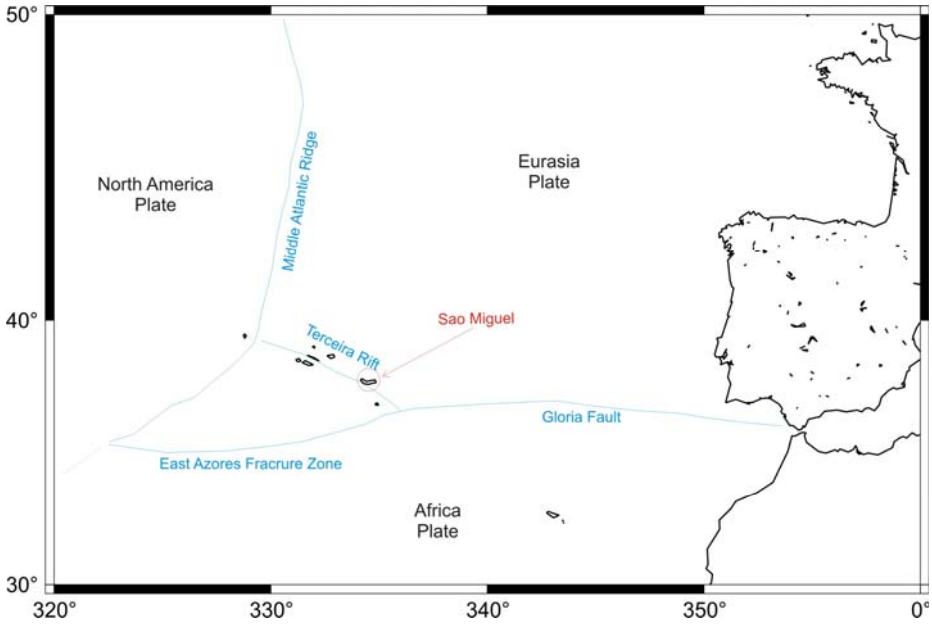
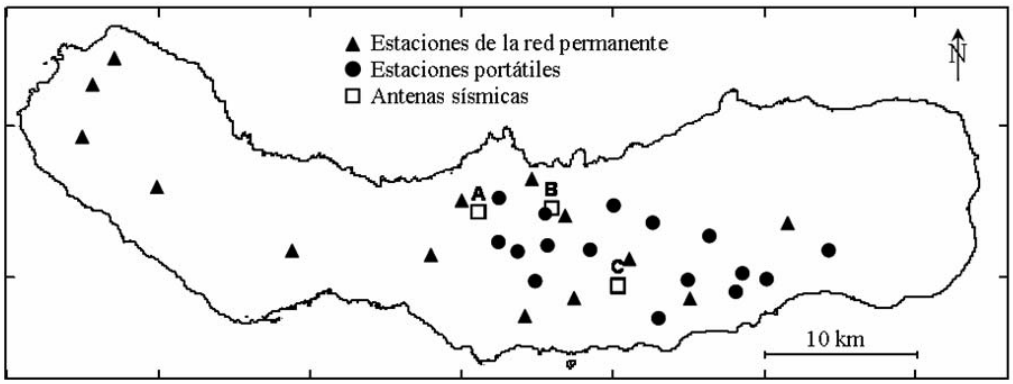


Figura 6.13 (derecha). *Situación geográfica del archipiélago de las Azores y de la isla de São Miguel, en el área de unión entre las placas Eurasiática, Norteamericana y Africana.*

a)



b)

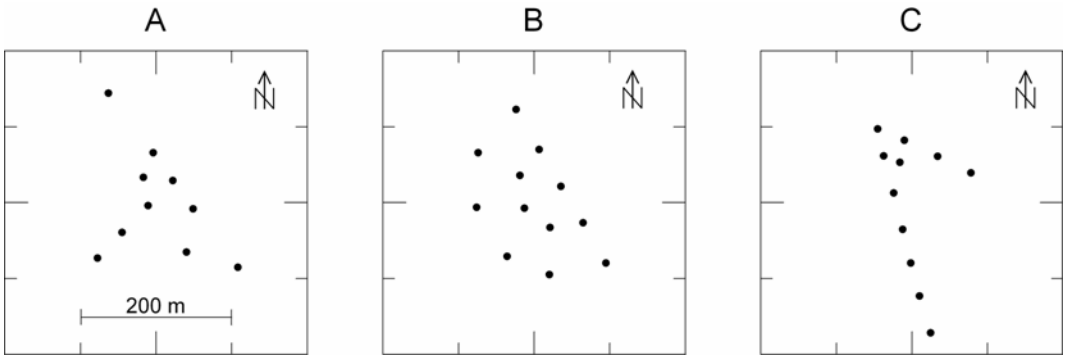


Figura 6.14. *Distribución de estaciones en la isla de São Miguel durante los meses de abril a julio de 2003 (a). En b) se muestra la configuración espacial de los sensores en las tres antenas sísmicas. Todos los sensores eran verticales, salvo uno de tres componentes en la antena A. Los verticales eran modelos Mark L28 con preamplificador y respuesta extendida a 1 Hz (ver sección 5.1.2.1). El de tres componentes se implementó mediante un sensor vertical y dos horizontales, todos modelos Mark L4 sin preamplificador.*

A pesar de los problemas surgidos con los ficheros de datos ya descritos en el capítulo anterior (sección 5.5.1.2), que llevaron a rediseñar la estrategia de adquisición de datos en el PC, los registros obtenidos en esta campaña resultaron de gran utilidad en dos aspectos (Ibáñez, comunicación personal):

1. Los eventos incluidos en ficheros en los que no se había producido la pérdida de ningún paquete de datos sirvieron para comprobar la validez del sistema de sincronismo utilizado en los nuevos módulos. Ésta era la primera ocasión en que las nuevas antenas se probaban en condiciones reales (de hecho, el montaje de los módulos utilizados y otros que se llevaron como repuesto se terminó poco antes del inicio de la campaña), por lo que resultaba fundamental realizar ésta y otras comprobaciones antes de dar el visto bueno a la fabricación del resto de los módulos. Como puede verse en la figura 6.15, la localización de los eventos con datos procedentes de las tres antenas resultó totalmente consistente con las localizaciones realizadas con los datos del resto de las estaciones de la red, tanto en el caso de eventos con epicentros externos (*a*) como internos (*b*) a la red sísmica. De estos resultados fue posible extraer dos conclusiones: por una parte, la calidad de los datos de los nuevos módulos (siempre en condiciones previas a los errores de falta de paquetes), tanto en sincronismo como en resolución. Por otra parte, la corroboración de que el método utilizado para la localización de los eventos, basado en la técnica de correlación cruzada ZLCC (Frankel, 1994, Del Pezzo *et al.*, 1997, Almendros *et al.*, 1999) era válido y podía, por tanto, aplicarse posteriormente a otros estudios.
2. Como se verá a continuación, el análisis de los datos de ruido sísmico registrados con las antenas llevó al descubrimiento de una señal coherente, probablemente de tipo trémor volcánico, que además podría haber sido la precursora de un enjambre de eventos que se produjo durante la campaña.

a) ***b)***

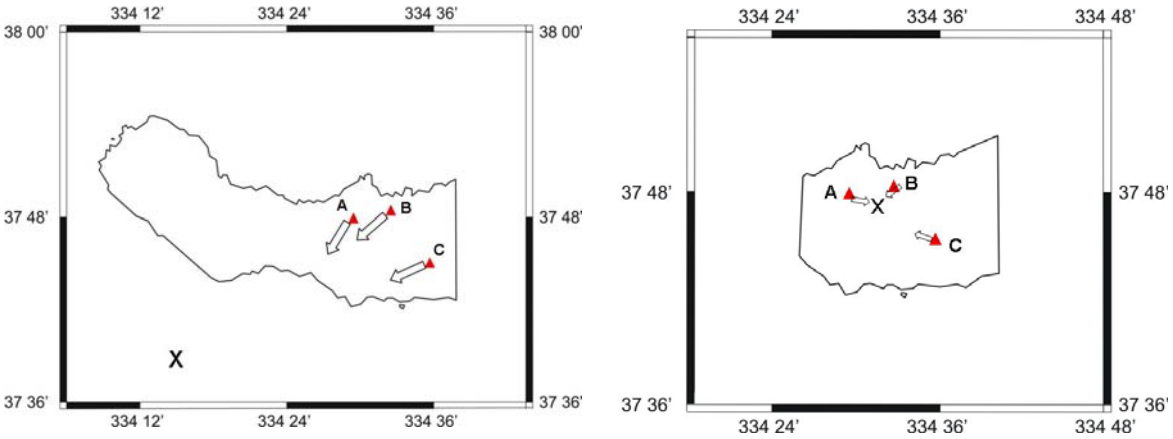


Figura 6.15. *Dos ejemplos de eventos ocurridos el 5 y 6 de julio de 2003 en el exterior (a) y en el interior (b) de la red sísmica. Las flechas corresponden al azimut obtenido con cada uno de los arrays. Las cruces indican la posición del epicentro, calculada aplicando la técnica de inversión de los tiempos de llegada a datos de la red sísmica (excluyendo las antenas). Como puede verse, los arrays detectan la dirección de llegada de las ondas, apuntando en todos los casos hacia el epicentro con un error en el azimut en torno a ±10º, lo cual es suficiente para validar los datos y la técnica de procesamiento utilizada.*

Tras la validación de los datos y métodos mediante la comparación de los resultados de las localizaciones de la red y de las antenas, se trabajó principalmente con datos de la antena C, debido a la calidad de sus registros y a la proximidad de la fuente sísmica. Durante el tiempo que duró el experimento se localizaron más de 1000 terremotos, gran parte de los cuales se produjeron en forma de un enjambre el día 26 de abril de 2003. Con el fin de seleccionar la banda de frecuencia de filtrado de los registros se estudió el espectro del ruido sísmico en varias bandas, y se aplicaron las técnicas de procesado ZLCC y MUSIC para calcular los parámetros de propagación y correlación. Las características del ruido cambiaron drásticamente a partir de las primeras horas del 23 de abril. Hasta entonces los parámetros de propagación y correlación indicaban la presencia de una señal coherente únicamente para frecuencias menores de 1.5 Hz, fenómeno típicamente asociado en islas oceánicas al ruido microsísmico de origen marino (ver, por ejemplo, Almendros *et al.*, 2000). A partir de ese momento, y hasta la ocurrencia del enjambre sísmico del día 26 de abril, el valor del coeficiente de correlación aumentó en diversas bandas y los parámetros de propagación se orientaron hacia la posición aproximada de los epicentros de los terremotos correspondientes al enjambre (figura 6.16). Estos resultados fueron corroborados por estudios de polarización, realizados con datos de las estaciones de tres componentes de la red sísmica. Los valores de azimut y lentitud aparente resultantes de la aplicación del método ZLCC a la banda 0.5-4 Hz se muestran en la figura 6.17 en forma de diagramas de barras. En todos los casos se aprecia un pico en torno a los 180º de azimut, que corresponde al ruido de origen oceánico. Pero además se observan otros picos que corresponden a señales coherentes, como el que surge en torno a valores de 130º de azimut con lentitudes aparentes del orden de 0.5 s/km, que implica la presencia en el campo de ondas de componentes no superficiales. Este pico aparece en las últimas horas del día 23 de abril, dos días antes la crisis sísmica, y desaparece algunos días después. En los diagramas correspondientes al 5 de mayo de la figura 6.17 ya no hay evidencia ni de éste ni de otros picos, lo cual indica que la señal no presenta una dirección preferente de incidencia. Teniendo en cuenta que el azimut de los terremotos del enjambre del día 26 es aproximadamente el mismo que el del pico descrito, este hecho podría interpretarse como la aparición de una señal de trémor volcánico precursora de los eventos.

Existen otros indicios, como el aumento en la emisión de CO_2 difuso los días posteriores a la crisis sísmica, que permiten aventurar hipótesis sobre el desarrollo del proceso (Ibáñez, comunicación personal). Éste podría haber tenido su origen en la emisión de gases magmáticos a una cierta profundidad, que al ascender habrían producido la señal de trémor de los días 23 y 24 de abril. Posteriormente los gases habrían lubrificado en su ascenso un sistema de fallas preexistentes, provocando tensiones y rupturas, lo cual habría dado origen al enjambre de terremotos del día 26. Los gases habrían continuado ascendiendo hasta niveles superficiales, provocando el aumento en el CO_2 difuso registrado entre los días 28 al 30 de abril. Como se verá a continuación, esta explicación es parecida a la que otros autores han dado para el aumento de la actividad en el Teide en el mes de mayo de 2004.

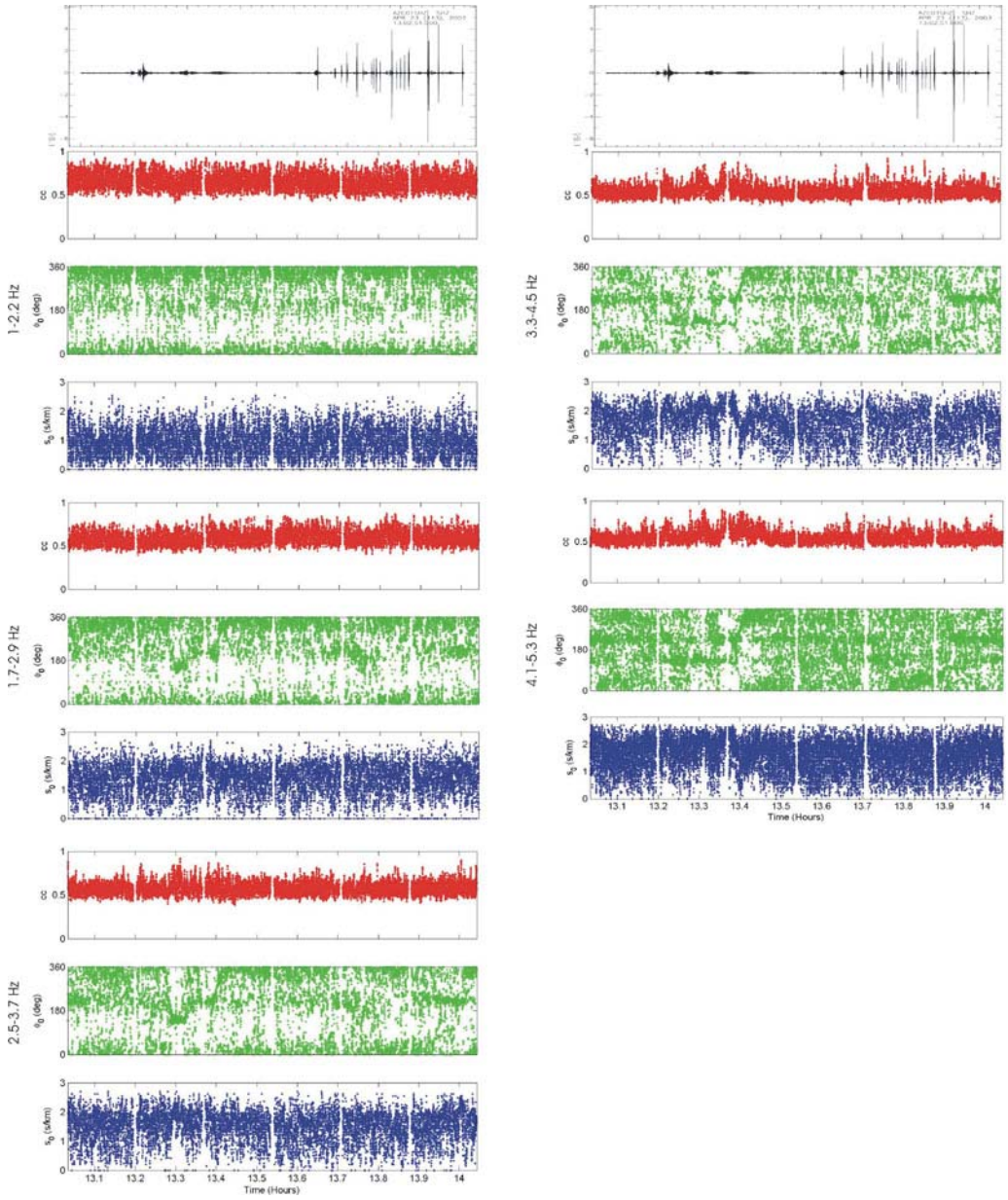


Figura 6.16.- *Resultados del método MUSIC en distintas bandas de frecuencia aplicados a un fichero de una hora de duración del día 23 de abril de 2003. En rojo se muestra el coeficiente de correlación, en verde el azimut y en azul el parámetro del rayo. Puede observarse que el coeficiente de correlación no baja en ningún caso de 0.5 y que aparecen picos en el azimut en torno a 230º y 130º, éste último visible en las frecuencias más altas.*

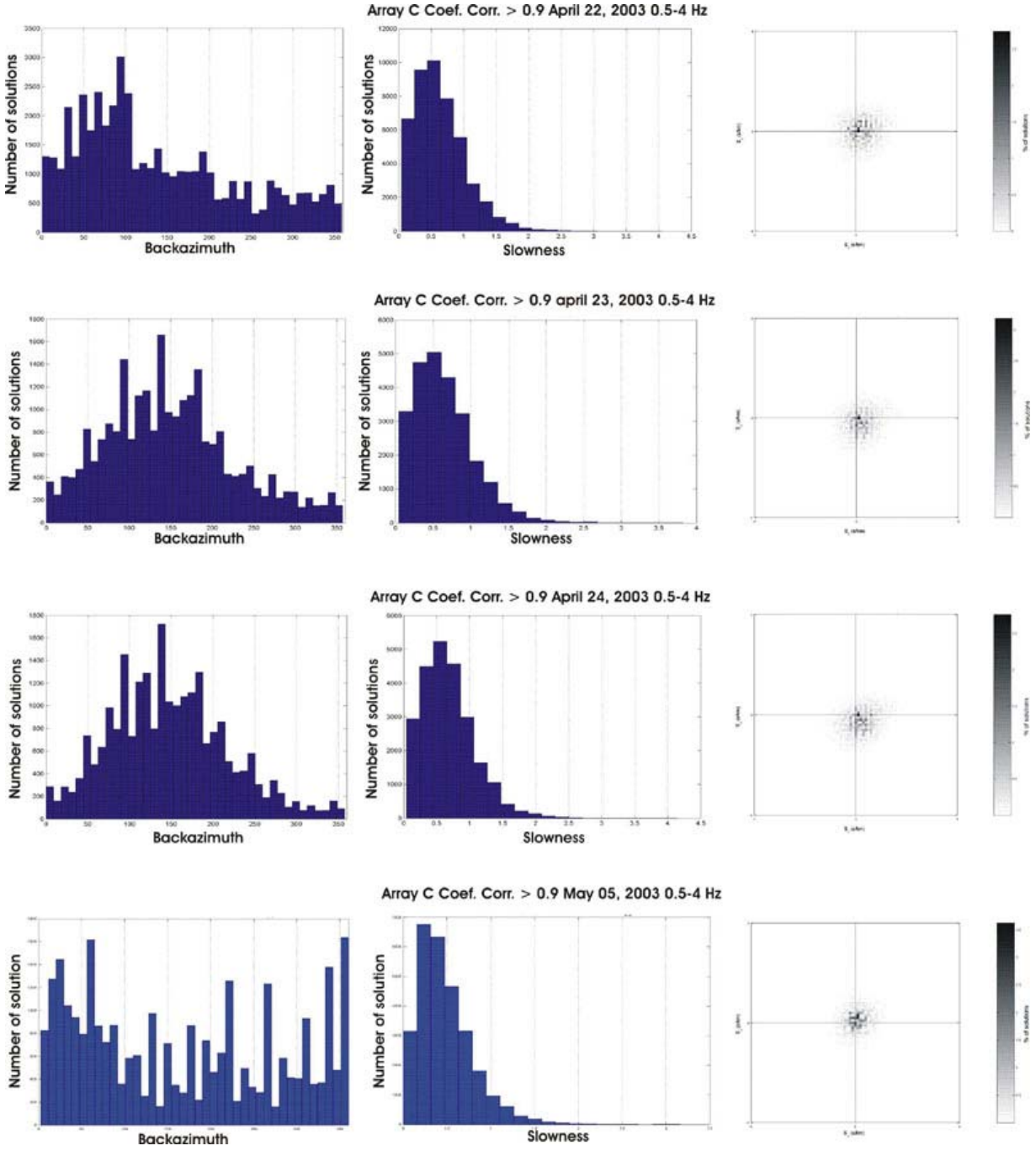


Figura 6.17.- *Diagramas de barras representando el número de soluciones con determinados valores de azimut y parámetro de rayo en distintas fechas. El pico a 130º aparece el 23 de abril y se mantiene el 24, pero no se aprecia en el diagrama del 5 de mayo. A la derecha se muestran los correspondientes mapas de distribución de azimuts y parámetros de rayo en el plano de velocidad aparente.*

6.2.2. El Teide

El complejo del Teide – Pico Viejo – Cañadas es uno de los sistemas volcánicos más importantes de las Islas Canarias. En los últimos 300 años se tiene noticia de seis episodios efusivos, el último de los cuales, la erupción de Chinyero, data de 1909. Asimismo existen evidencias de erupciones explosivas, como la subpliniana ocurrida hace unos 2000 años en Montaña Blanca, en el flanco SE del Teide (Ablay *et al.*, 1995). La existencia de áreas densamente pobladas en el edificio volcánico convierte la región en una de las más vulnerables del archipiélago canario.

Todos los trabajos realizados en los últimos años para estudiar la estructura y dinámica del volcán, no sólo desde el punto de vista de la sismología (Del Pezzo *et al.*, 1997; Canas *et al.*, 1998; Canales *et al.*, 2000; Almendros *et al.*, 2000 y 2004b), sino a partir de datos de campañas gravimétricas y geodéticas (Araña *et al.*, 2000; Yu *et al.*, 2000; Fernández *et al.*, 2003), magnetotelúricas (Ortiz *et al.*, 1986, Pous *et al.*, 2002), análisis geoquímicos (Hernández *et al.*, 2000 y 2004) y magnéticos (Blanco, 1997; Araña *et al.*, 2000) habían concluido que el sistema se encontraba en una fase de reposo. Sin embargo, en 2004 se detectó un incremento de la actividad sísmica, con algunos eventos lo suficientemente energéticos ($M > 3$) como para ser sentidos en poblaciones cercanas. El aumento en el número de eventos vino además acompañado por una migración en la posición de las fuentes, desde una localización preferencial en el mar, al SE de la isla, hacia otra situada en las proximidades del Teide, al NW del cráter.

En mayo de 2004, como respuesta al incremento de actividad detectado, el IAG llevó a cabo una campaña de recogida de datos sísmicos, en la que se utilizaron tres antenas que se instalaron en la caldera de Las Cañadas (figura 6.18) y que se implementaron mediante los nuevos sistemas descritos en este trabajo. Cada uno de los *arrays* se dotó de nueve sensores verticales y uno de tres componentes, todos ellos modelos *Mark L28* con la respuesta extendida electrónicamente a 1 Hz (ver sección 5.1.2.1). Almendros *et al.*, 2005a y 2007, analizaron los datos procedentes de esta campaña. Debido a dificultades de mantenimiento, durante el último tramo de la misma las antenas operaron de forma intermitente, por lo que los dos trabajos se centran en la primera parte, que por otra parte fue la más activa.

Los datos registrados revelan la presencia de varias señales distintas (figura 6.19). En primer lugar, una de baja frecuencia (por debajo de 1 Hz) asociada a ruido microsísmico de origen oceánico (Almendros *et al.*, 2000). Superpuesta a esta señal de fondo, los datos contienen algunos eventos de largo periodo (LPs), generalmente relacionados con la presencia de fluidos volcánicos o hidrotermales (ver, por ejemplo, Chouet, 1996). Además se registraron varios cientos de pequeños terremotos volcano-tectónicos (VT), con duraciones cortas, contenidos de alta frecuencia y ancho espectro. Almendros *et al.*, 2005a, llevaron a cabo su localización mediante las antenas sísmicas PV y DH, obteniendo resultados consistentes con las localizaciones realizadas a partir de datos de la red regional del IGN. Esto resulta especialmente importante en el contexto de esta tesis, puesto que debido a los problemas surgidos en la campaña de las Azores, que obligaron a cambiar la filosofía del sistema de adquisición (ver sección 5.5.1.2), ésta era la primera campaña en la que los nuevos equipos se utilizaban en la versión que se ha presentado aquí. Así, la compatibilidad en los resultados de las localizaciones de los VTs mediante la red regional y mediante las antenas sirve como confirmación de que el sistema de adquisición de datos, y más concretamente el procedimiento utilizado para la sincronización del proceso de muestreo, proporciona los resultados esperados en condiciones reales de operación.

No obstante, lo más destacable en el muestrario de señales registradas durante la campaña de 2004 fue la aparición de un trémor volcánico, que constituye una observación única en el volcán Teide. Este tipo de señales no se había detectado antes en ninguna campaña anterior, incluyendo las que el propio IAG había llevado a cabo con instrumentación parecida

(los módulos de *array* de dieciséis bits) en las mismas áreas en los años 1994, 2000-01 y 2003-04 (Almendros *et al.*, 2000).

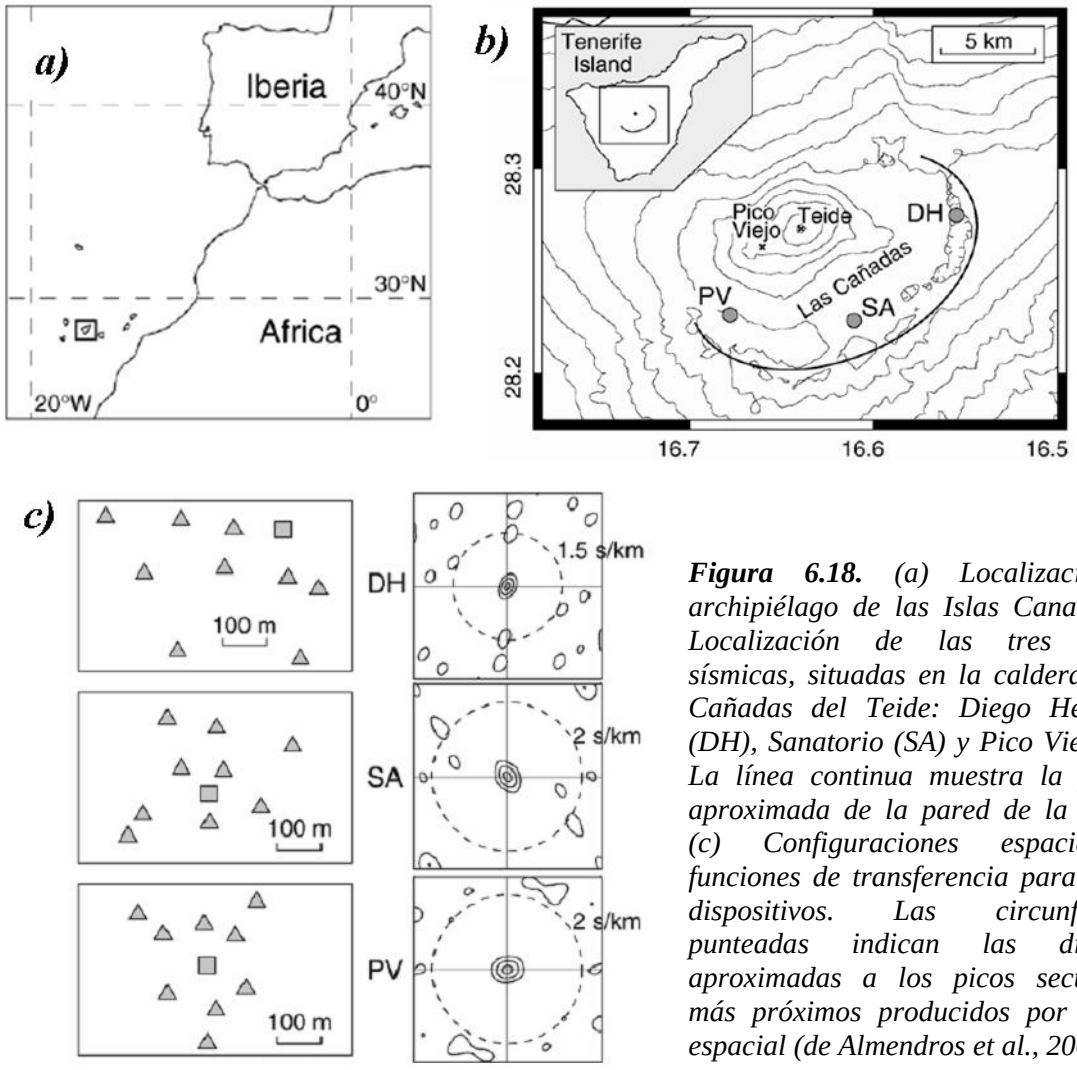


Figura 6.18. *(a) Localización del archipiélago de las Islas Canarias. (b) Localización de las tres antenas sísmicas, situadas en la caldera de Las Cañadas del Teide: Diego Hernández (DH), Sanatorio (SA) y Pico Viejo (PV). La línea continua muestra la posición aproximada de la pared de la caldera. (c) Configuraciones espaciales y funciones de transferencia para los tres dispositivos. Las circunferencias punteadas indican las distancias aproximadas a los picos secundarios más próximos producidos por aliasing espacial (de Almendros et al., 2007).*

La figura 6.20 muestra el inicio de la señal del trémor en la antena DH, la única en la que se observó puesto que las otras dos no estuvieron operativas hasta algunos días después. El comienzo se observa como una señal casi monocromática emergente del ruido oceánico de baja frecuencia. Su frecuencia central aumenta lentamente con el tiempo, estabilizándose unas seis horas después en unos 4-5 Hz, con ligeras variaciones posteriores. Las antenas SA y PV registraron señales similares desde el principio de su operación, lo cual descarta que se deba a un efecto de sitio en el emplazamiento de DH. La duración del trémor fue de al menos varias semanas, aunque no se puede establecer con precisión ya que aún se seguía registrando cuando los *arrays* se retiraron al finalizar la campaña. El trémor observado no sólo plantea una cuestión intrigante desde el punto de vista científico, sino muy interesante en el aspecto técnico, puesto que su amplitud extremadamente baja constituye un reto para calibrar la capacidad de detección de los nuevos sistemas. Almendros *et al.*, 2007, utilizaron la técnica de correlación cruzada ZLCC (Frankel, 1994, Del Pezzo *et al.*, 1997, Almendros *et al.*, 1999) para determinar las lentitudes aparentes y azimuts de propagación en las señales registradas. La figura 6.21 muestra la máxima correlación cruzada promedio para cada una de las antenas en el periodo analizado. Como puede verse, antes del comienzo de la señal de trémor la correlación es baja, con valores medios en torno a 0.3, lo cual indica que no existían fuentes coherentes de señal en el campo de ondas. Esto demuestra, entre otras cosas, que el ruido

electrónico generado en los nuevos sistemas no tiene efecto en los resultados calculados mediante la técnica ZLCC. Una vez que la señal de trémor está presente, la correlación cruzada aumenta para dar valores de pico de hasta 0.7, si bien con una variación oscilatoria de periodo 24 h que está claramente relacionada con el ciclo día/noche. Los máximos y mínimos en la correlación coinciden con las horas nocturnas y diurnas, respectivamente. Los autores atribuyen este hecho a que, debido a la baja amplitud de la señal de trémor, las antenas tienen más dificultades en su identificación durante las horas diurnas a causa del mayor nivel de ruido cultural.

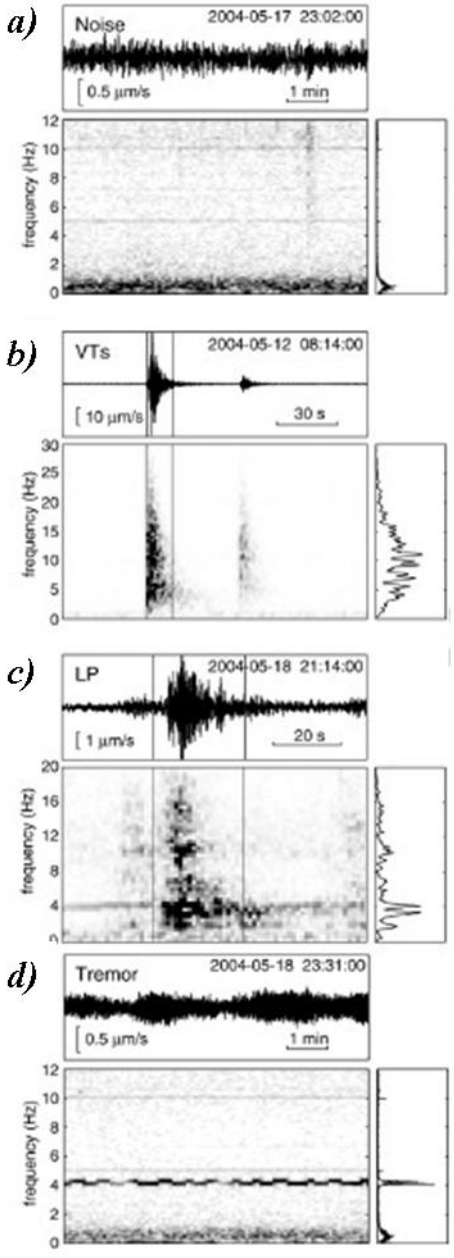


Figura 6.19 (izquierda). *Sismogramas verticales, espectrogramas y espectros para el ruido de origen oceánico (a), terremotos VT (b), eventos LP (c) y trémor volcánico (d) registrados en el Teide durante la campaña de 2004 (de Almendros et al., 2007).*

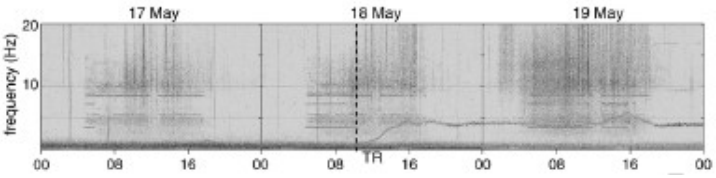


Figura 6.20. *Espectrograma de tres días de duración (entre el 17 y el 19 de mayo de 2004) de una de las estaciones verticales del array DH. La línea de puntos marca el inicio de la señal de trémor (de Almendros et al., 2007).*

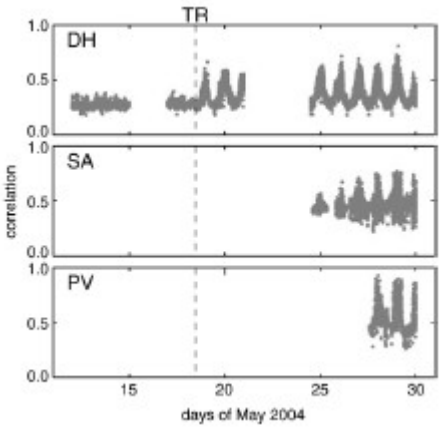


Figura 6.21. *Representación de la correlación cruzada promedio máxima para cada una de las tres antenas sísmicas. La línea de puntos marca el inicio de la señal de trémor (de Almendros et al., 2007)*

El análisis detallado de los datos reveló que los parámetros de propagación del trémor no eran constantes en el tiempo para ninguna de las tres antenas. La figura 6.22 muestra un ejemplo de la evolución del azimut y lentitud aparente para la antena DH. La 6.23 muestra el resultado de los análisis para un periodo simultáneo de señal en las tres antenas.

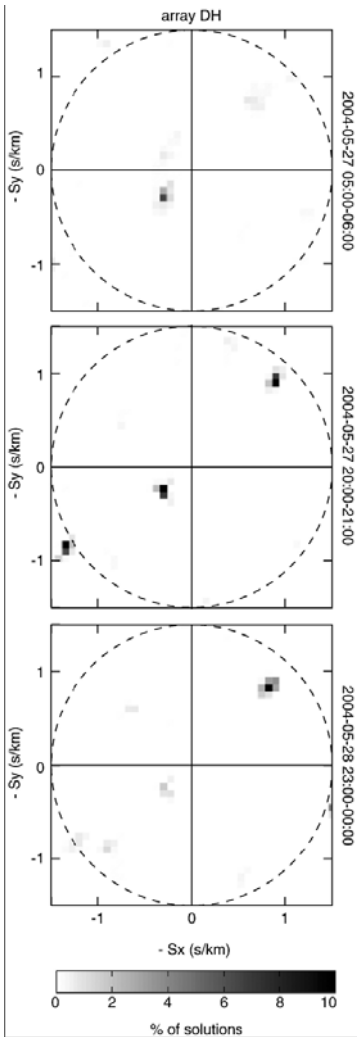
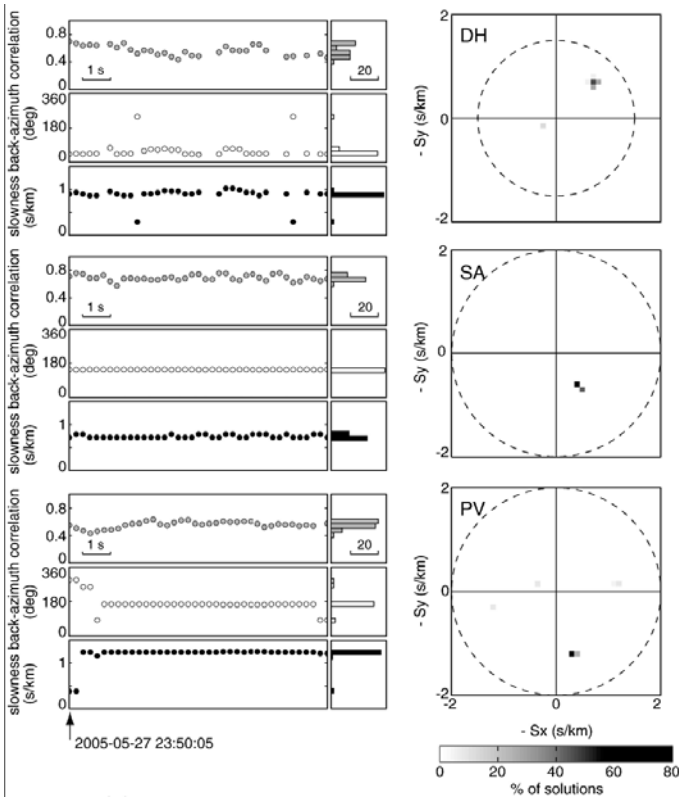
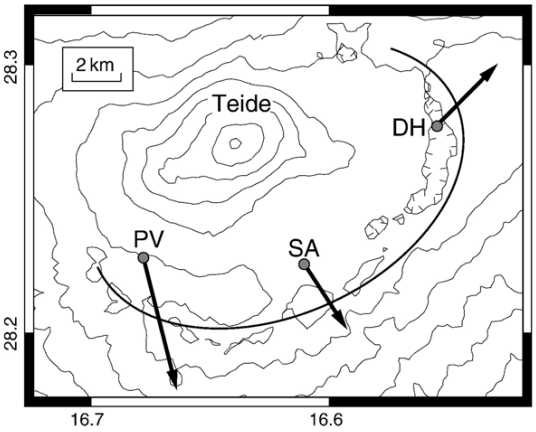


Figura 6.22 (derecha). *Ejemplos de histogramas de una hora de duración para la lentitud aparente en la antena DH. El círculo punteado representa la máxima lentitud aparente permitida por la antena, por lo que el máximo exterior de la figura central no es representativo. Puede observarse una evolución de la posición de los máximos, probablemente asociada a la desactivación de una fuente y activación de otra (de Almendros et al., 2007)*

Figura 6.23 (abajo). *Resultados obtenidos mediante la técnica ZLCC para las tres antenas y 10 s de datos simultáneos. A la izquierda, histogramas de la correlación, azimut y lentitud aparente para las antenas DH, SA y PV, respectivamente. En el centro, representación de los vectores de lentitud en el plano de lentitud aparente. A la derecha, mapa de la caldera de Las Cañadas en el que se representan los vectores de lentitud aparente promedio (de Almendros et al., 2007).*





Los altos valores de correlación y la estabilidad de las soluciones indican la presencia de fuentes sísmicas continuas y coherentes. Sin embargo, los parámetros de propagación en las distintas antenas son incompatibles con un modelo de una sola fuente e impiden la localización mediante la técnica del cruce de rayos u otro método de análisis conjunto. Todo ello lleva a pensar, más bien, en la presencia simultánea de múltiples fuentes de trémor. Cada antena recogería las aportaciones de varias fuentes que permanecerían activas durante un tiempo, y las soluciones del análisis estarían relacionadas con la fuente más energética o más próxima al *array* en ese preciso momento.

El modelo propuesto por los autores para explicar la reactivación de la actividad implica una intrusión profunda de magma, que produjo un enjambre de terremotos VT y provocó emisión de gases responsables de la generación de los eventos LP. Los gases se fueron desplazando hacia el exterior, lubrificando a su paso sistemas de fallas preexistentes que indujeron la ocurrencia de más terremotos VT, lo cual explicaría la migración de la posición preferente de las fuentes observada en este tipo de eventos. Por último, el flujo de gas habría interactuado con el acuífero superficial, originando la señal de trémor (figura 6.24).

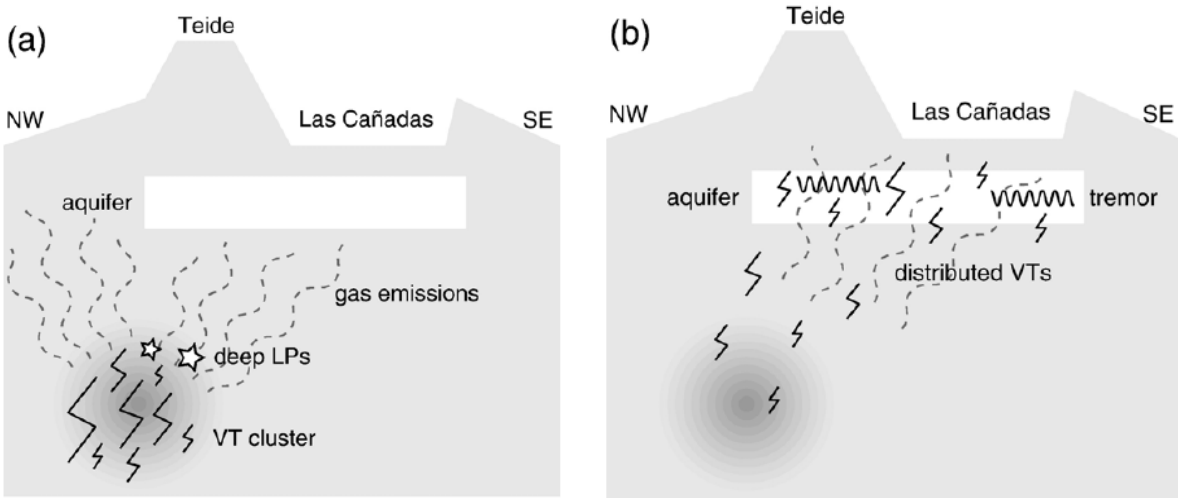


Figura 6.24. *Interpretación de los autores de los eventos sísmicos ocurridos en el entorno del Teide durante el mes de mayo de 2004. En a) se muestra la situación anterior al 18 de mayo, en la que una inyección profunda de magma habría tenido lugar por debajo del edificio volcánico. Según este modelo, la inyección magmática habría producido el primer enjambre de terremotos VT y una emisión de gas que habría estimulado la ocurrencia de los eventos LP. En b), la situación después del 18 de mayo, en la que la presión de fluidos habría lubrificado el sistema de fallas preexistentes induciendo a su paso más eventos VT. El flujo de gas, al interactuar con el acuífero superficial, habría sido el responsable de la generación de la señal de trémor (de Almendros et al., 2007).*

6.2.3. Decepción

La isla Decepción (62º 43' S, 60º57' W) es un complejo volcánico del archipiélago de las Shetland del Sur, situado cerca de la Península Antártica. La isla se formó en diversos episodios eruptivos ocurridos desde el Cuaternario hasta la actualidad. La inundación de la caldera volcánica originó una amplia bahía interior (Port Foster), de entre 8 y 10 km de diámetro, lo cual le da una característica forma de herradura (figura 6.25). Las erupciones más recientes se produjeron en los años 1967, 1969 y 1970 y están bien documentadas, dado que durante esos periodos la isla contaba con bases científicas de diversos países. En la actualidad existe actividad sísmica de baja magnitud, consistente básicamente en terremotos volcano-tectónicos, eventos de largo periodo y trémor volcánico, además de eventos puramente tectónicos asociados a la sismicidad regional. La mayor parte de la sismicidad superficial es debida a la actividad hidrotermal del sistema volcánico (Ibáñez *et al.*, 2000). Esta situación se ve salpicada por periodos de mayor actividad, como los que tuvieron lugar en los años 1993 y 1999, en los que se producen un elevado número de terremotos volcano-tectónicos que pueden llegar a ser relativamente energéticos, y que se han relacionado con procesos de intrusión de fluidos en el sistema volcánico.

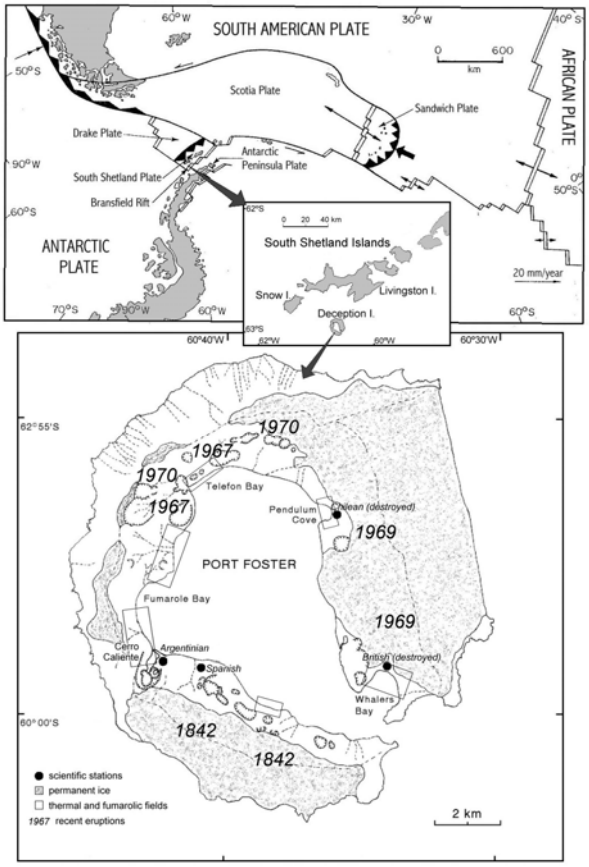


Figura 6.25. *Mapa de la Isla Decepción, en el que se indica su posición, próxima a la Península Antártica, y la localización de las erupciones más recientes (de Zandomeneghi et al., 2005).*

La actividad sísmica en Isla Decepción se viene monitorizando desde principios de los años noventa (ver, por ejemplo, Almendros *et al*, 1997; Alguacil *et al.*, 1999; Ibáñez *et al.*, 2000, 2003a y 2003b; Saccorotti *et al.*, 2001a; Martínez-Arévalo *et al.*, 2003; Benítez *et al.*, 2006). Desde entonces el IAG ha llevado a cabo, prácticamente sin interrupción, campañas anuales de recogida de datos, cuya duración suele ser de entre tres y cuatro meses correspondientes al periodo de mejores condiciones climatológicas (verano austral). Para ello se han utilizado distintos tipos de instrumentación, incluyendo sistemas de corto periodo con registro local o transmisión telemétrica de los datos, estaciones de banda ancha con registro local y antenas sísmicas. Hasta el año 2003 éstas se basaban en los antiguos módulos de dieciséis bits que se presentaron en la introducción del capítulo 5. A partir de entonces los nuevos sistemas portátiles tomaron el relevo.

En la campaña 2003-04 se utilizó la primera versión de los sistemas, que difería de la presentada en este trabajo sobre todo en el tipo de contenedores y conectores utilizado (Abril *et al.*, 2004). En su versión definitiva, las antenas se instalaron por primera vez en la campaña 2004-05, en la que se llevó a cabo un ambicioso experimento encaminado a la determinación de la estructura tridimensional de la isla mediante tomografía de velocidad de alta resolución

(proyecto TOMODEC). Con este fin se dispusieron numerosas estaciones sísmicas, tanto terrestres como de fondo oceánico (OBSs, *Ocean Bottom Seismometers*) y se realizaron más de cinco mil disparos con cañones de aire comprimido para registrar las ondas sísmicas producidas. El objetivo, la determinación de la estructura de velocidad de las ondas P en la corteza superior del volcán y el área circundante, ayudará a comprender el origen de la isla y su entorno regional.

La instrumentación utilizada en TOMODEC consistió en una red sísmica de registro continuo, formada por 130 estaciones terrestres y 13 OBSs (figura 6.26). La instrumentación terrestre incluía 21 sistemas de banda ancha, con frecuencia de muestreo de 125 mps y 24 bits de resolución nominal, estaciones de corto periodo y tres *arrays* densos de al menos ocho sensores de corto periodo (uno de ellos de tres componentes). Para implementar las tres antenas, así como el grueso de las estaciones de corto periodo[58], se utilizaron doce de los nuevos sistemas portátiles descritos en esta memoria. Los otros dos módulos se dejaron como reserva.

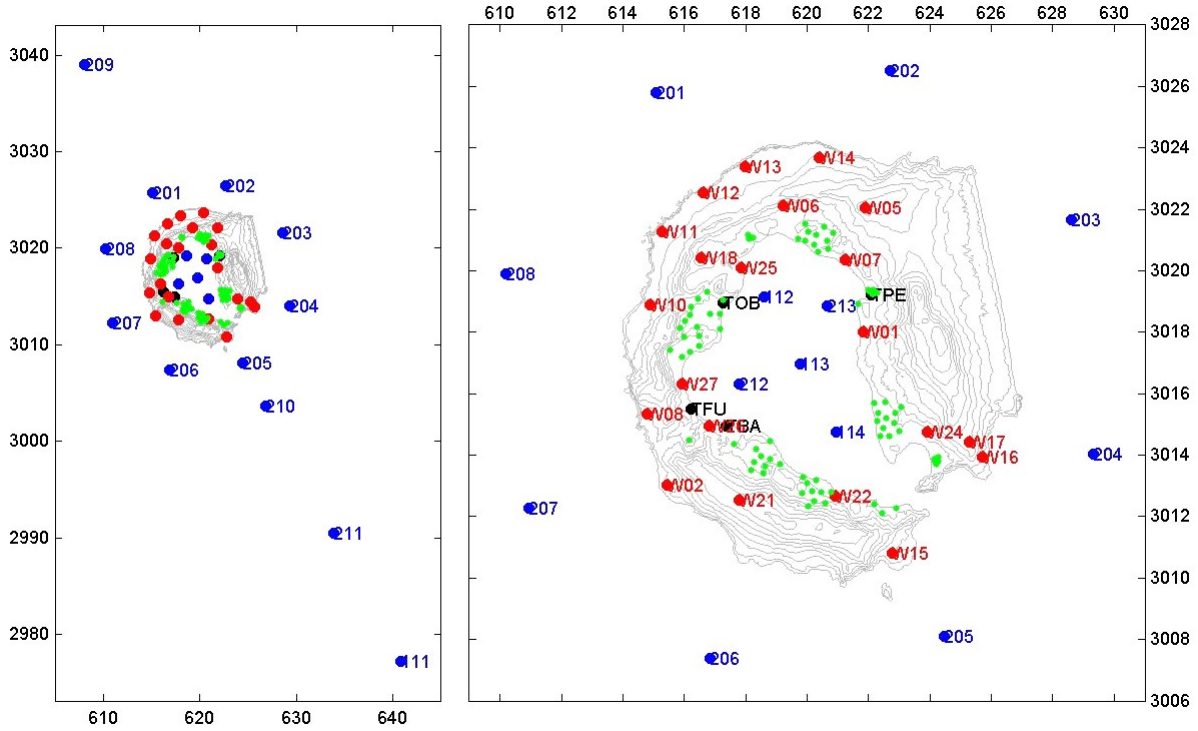


Figura 6.26. *Distribución de las estaciones sísmicas para la campaña TOMODEC. A la izquierda se aprecian todos los OBSs, marcados con puntos azules. La ampliación de la derecha muestra con más detalle las estaciones de banda ancha (puntos rojos) y las estaciones de corto periodo (puntos verdes), implementadas mediante los nuevos sistemas portátiles descritos en este trabajo (de Zandomeneghi et al. 2005).*

La utilización de un número tan alto de sensores e instrumentos requirió de un gran despliegue logístico. Más de treinta personas de diversas instituciones, organizadas en varios grupos de tierra y uno en el Buque de Investigación Oceanográfica (BIO) Hespérides, trabajaron de modo coordinado en el despliegue y mantenimiento de los instrumentos y en la

[58] La diferencia en este caso entre estaciones independientes y nodos de un *array* no está muy clara, ya que el fundamento de la tomografía consiste en distribuir en la superficie objeto de estudio gran número de sensores en configuraciones densas. La densidad de sensores, entre otros factores, determinará la resolución final de la tomografía.

verificación previa de los datos (figura 6.27). Las fuentes de ondas sísmicas necesarias para este tipo de experimentos provenían de disparos de cañones de aire comprimido situados en el BIO Hespérides. Durante los diez días que duró la campaña el buque efectuó más de 5200 disparos siguiendo un patrón reticular en el interior de la isla y una configuración anular en la parte exterior (figura 6.28).

El volumen total de datos registrados asciende a unos 120 GB, que están actualmente en proceso de análisis. No obstante, ya se han obtenido algunos resultados preliminares (Zandomeneghi *et al.*, 2005), a partir de una selección de datos consistente en la primera llegada de la onda P de un conjunto de 1100 disparos, efectuados cuando estaban operativas las primeras 50 estaciones. La figura 6.29 muestra un ejemplo de registros de los sucesivos disparos tomados con una de ellas. A partir de los registros y de un modelo unidimensional de velocidad definido inicialmente se utilizó un método de inversión de los tiempos de viaje de las ondas P (Toomey *et al.*, 1994) para obtener los resultados que se muestran en la figura 6.30. Estos resultados preliminares indican que los datos resuelven satisfactoriamente variaciones de velocidad de las ondas, que están relacionadas con la estructura volcánica. En el contexto de esta tesis los registros de esta campaña son particularmente importantes, ya que en este caso no es posible realizar, como en el *array* del Gran Sasso (Saccorotti *et al*, 2006), una comparación entre la localización de eventos con la antena y con una red sísmica regional operativa, puesto que ésta no existe. Sin embargo, el hecho de que la posición de la fuente sísmica sea conocida (el buque Hespérides) permite realizar una estimación de la calidad de los datos y en particular del sistema de sincronismo entre los distintos canales.

Aparte de la realización de la tomografía de velocidad, los datos de la campaña TOMODEC se están aprovechando para otros estudios. Así, Almendros y Abella, 2006, utilizaron un conjunto seleccionado de registros para calibrar el método RelSE de localización mediante antenas sísmicas (Almendros *et al.*, 2004a). Esta técnica constituye una mejora de la de correlación cruzada promedio (Del Pezzo *et al.*, 1997) para multipletes[59]. Proporciona una estimación precisa de la diferencia de lentitud aparente y azimut de propagación para terremotos con formas de onda similares. Se seleccionaron datos de dos antenas (G y J) y dos grupos de disparos distribuidos en forma de cruz, centrados a distancias de unos 7 y 12 km de las antenas sísmicas (figura 6.31). Al ser la posición de las fuentes conocida es posible investigar la capacidad del método para reproducir los parámetros de propagación. Como se muestra en la figura 6.32, la posición relativa de las soluciones en el plano de lentitud aparente está mucho mejor definida que con la técnica de correlación cruzada promedio y resulta consistente con la posición real de las explosiones. La aplicación de este método al conjunto total de los datos y el análisis detallado de los resultados proporcionará importantes indicios sobre la estructura de velocidad del medio.

Por otra parte, Almendros y Luzón, 2006, y Almendros *et al*, 2006, también utilizaron datos de la campaña TOMODEC para determinar la estructura superficial de la isla mediante la autocorrelación espacial de ruido sísmico. Para ello escogieron registros largos de ruido (5 horas) en seis de las antenas instaladas durante el experimento (figura 6.33). Utilizaron para el análisis el método de autocorrelación espacial (SPAC), propuesto por Aki (1957) y mejorado recientemente por otros autores. Esta técnica se basa en calcular correlaciones a distintas frecuencias para todos los posibles pares de estaciones. Con esos datos se ajusta una curva teórica (curva de dispersión) que proporciona información sobre la velocidad en función de la frecuencia (figura 6.34). A partir de la curva de dispersión se utilizan los programas de Hermann con tres capas sobre un semiespacio para determinar la estructura local de velocidad de la onda S. Los resultados obtenidos se muestran en la figura 6.35. Como puede verse, se presentan capas de características similares, pero con sensibles diferencias incluso entre puntos cercanos, lo cual confirma la heterogeneidad del medio superficial. Esta misma técnica

[59] Se denomina multiplete a un conjunto de terremotos con formas de onda similares.

se aplicó en el año 2006 para determinar la estructura superficial de la isla de La Palma (España), si bien los datos registrados están aún en proceso de análisis (Almendros, comunicación personal).

Figura 6.27. *Distintas imágenes de la campaña TOMODEC. En a), el Buque de Investigación Oceanográfica (BIO) Hespérides, desde el que se efectuaron los disparos y se llevó a cabo la instalación y recogida de los sensores de fondo oceánico (OBSs). En b) traslado de uno de los grupos a tierra desde el barco. Los traslados se realizaban en lanchas neumáticas, cubriendo los equipos con bolsas de plástico para evitar salpicaduras. En c), uno de los nuevos sistemas portátiles en operación, en el que se aprecian los cables de los sensores saliendo de la caja de conexiones y el cable único hacia el módulo central. La alimentación se suministra mediante dos baterías en paralelo, para aumentar la autonomía.*

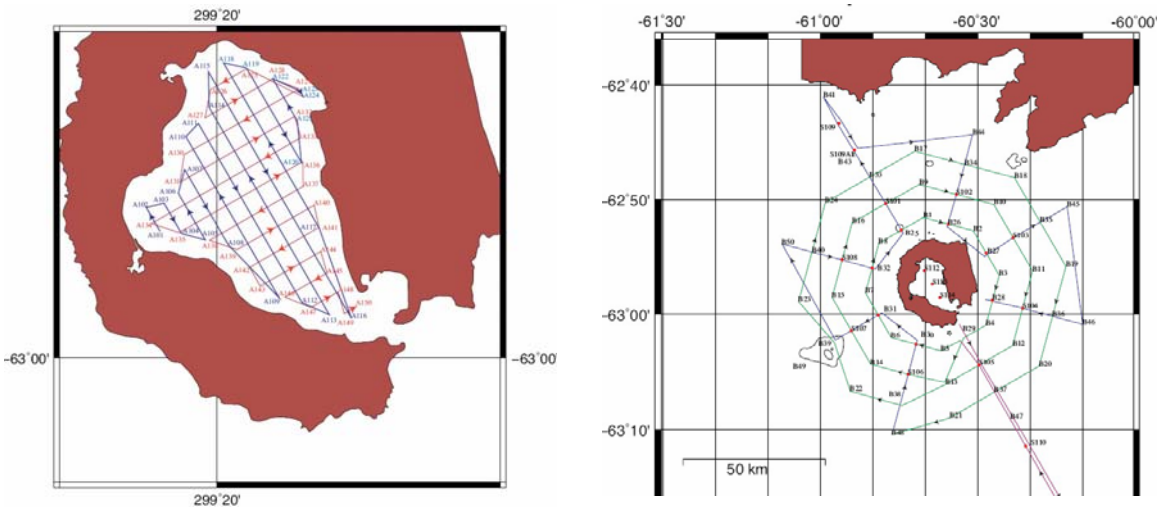


Figura 6.28. *Ruta seguida por el BIO Hespérides durante la campaña TOMODEC. Los disparos se realizaban con intervalos de un minuto en la ruta interior (izquierda) y de dos minutos en el exterior (derecha) (de Zandomeneghi et al., 2005).*

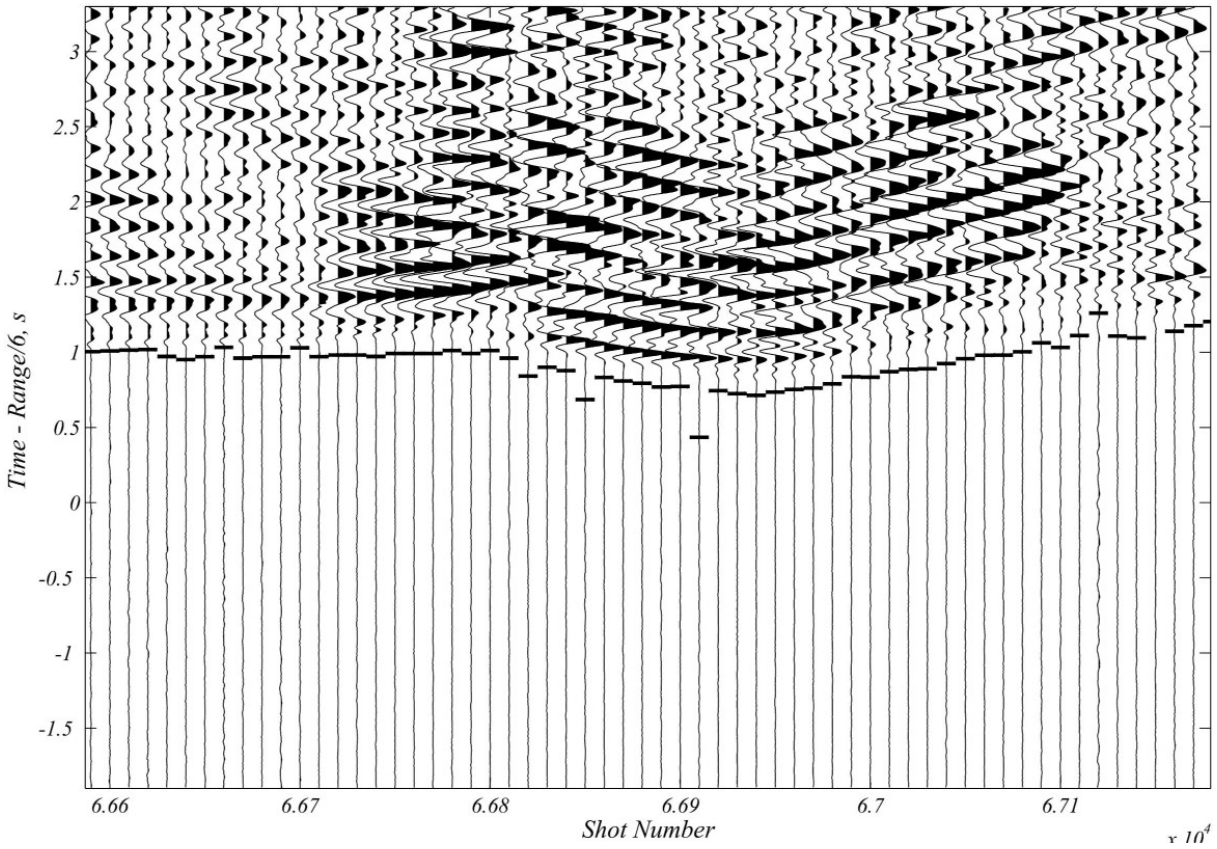


Figura 6.29. *Ejemplo de registros para una estación de tierra y un conjunto de disparos. Los tiempos de llegada de la onda P están marcados por un algoritmo automático. La aplicación del procedimiento automático a los datos de las 50 estaciones operativas produjo unas 34000 lecturas que se utilizaron como entrada del algoritmo de inversión (de Zandomeneghi et al., 2005).*

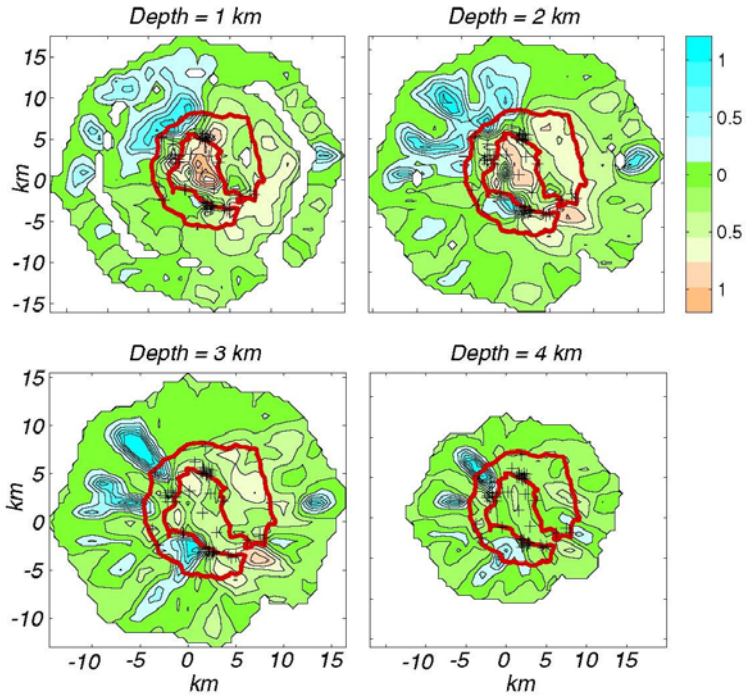


Figura 6.30. *Imagen de la estructura de velocidad de la Isla Decepción, calculada a partir del conjunto preliminar de datos descrito en el texto. La figura muestra las variaciones de la velocidad de las ondas P (en km/s) a distintas profundidades, respecto a las del modelo unidimensional definido inicialmente. El trazo rojo representa el contorno de la isla. Destacan las bajas velocidades en el centro de la caldera y el área de altas velocidades en el sector noroeste (de Zandomeeghi et al., 2005).*

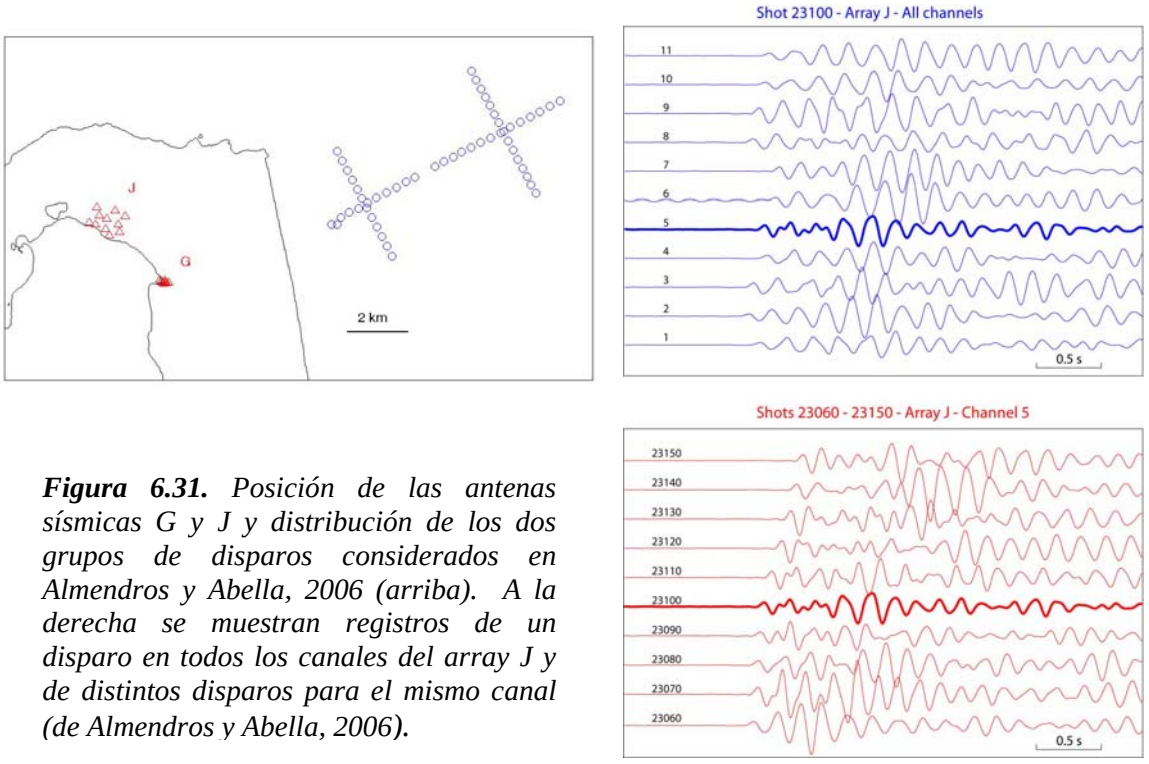


Figura 6.31. *Posición de las antenas sísmicas G y J y distribución de los dos grupos de disparos considerados en Almendros y Abella, 2006 (arriba). A la derecha se muestran registros de un disparo en todos los canales del array J y de distintos disparos para el mismo canal (de Almendros y Abella, 2006).*

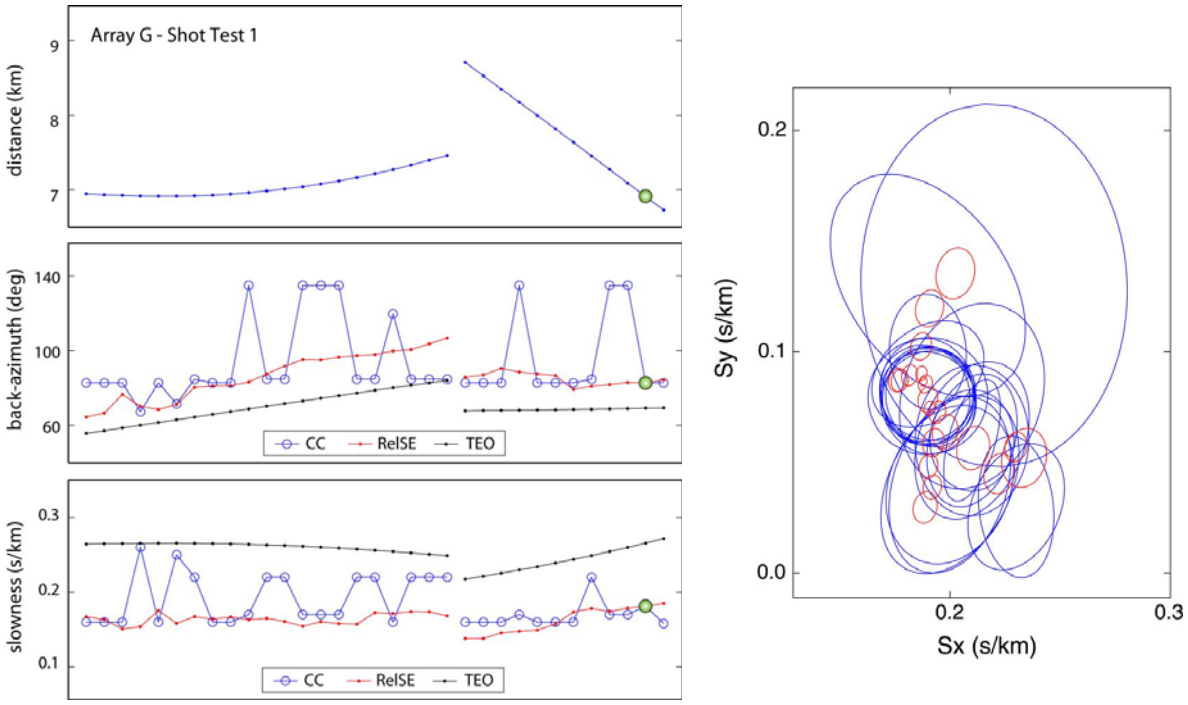


Figura 6.32. *Representación de las soluciones calculadas a partir del conjunto seleccionado de datos mediante las técnicas de correlación cruzada promedio (CC) y RelSE. TEO representa la solución teórica, determinada por la posición del buque Hespérides. A la derecha se representan las soluciones en el plano de lentitud aparente. Con líneas azules se muestran los resultados del método CC y con rojas los de RelSE, en los que, como puede verse, la indeterminación es mucho menor (de Almendros y Abella, 2006).*

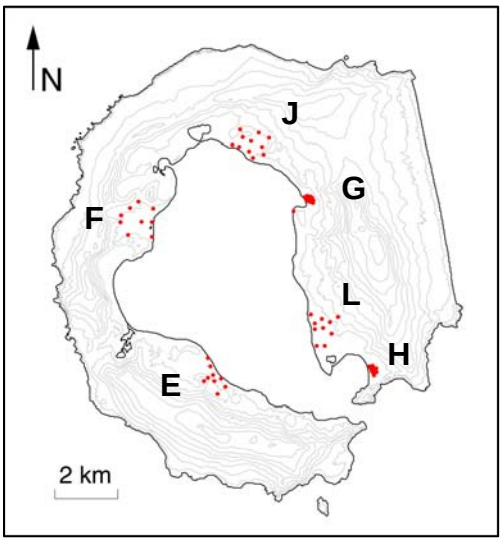


Figura 6.33. *Emplazamiento de las seis antenas sísmicas cuyos datos se utilizaron en Almendros et al., 2006, y en Almendros y Luzón, 2006, de donde se ha tomado la figura.*

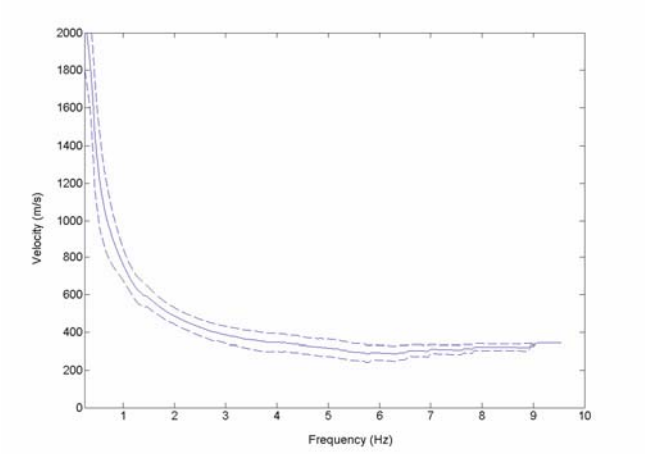


Figura 6.34. *Curva de dispersión obtenida a partir de los registros de ruido para el array G. Se obtuvieron curvas equivalentes para cada una de las antenas consideradas (de Almendros y Luzón, 2006).*

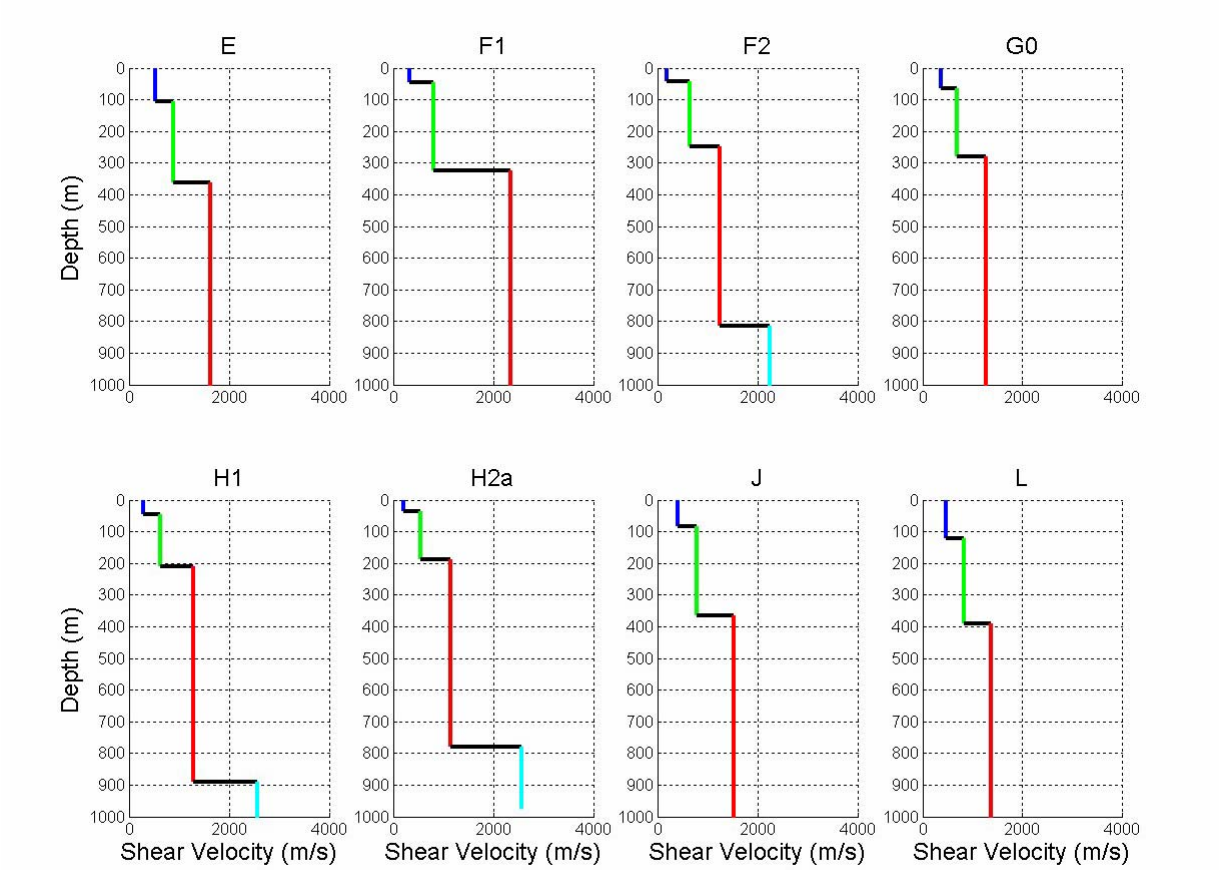


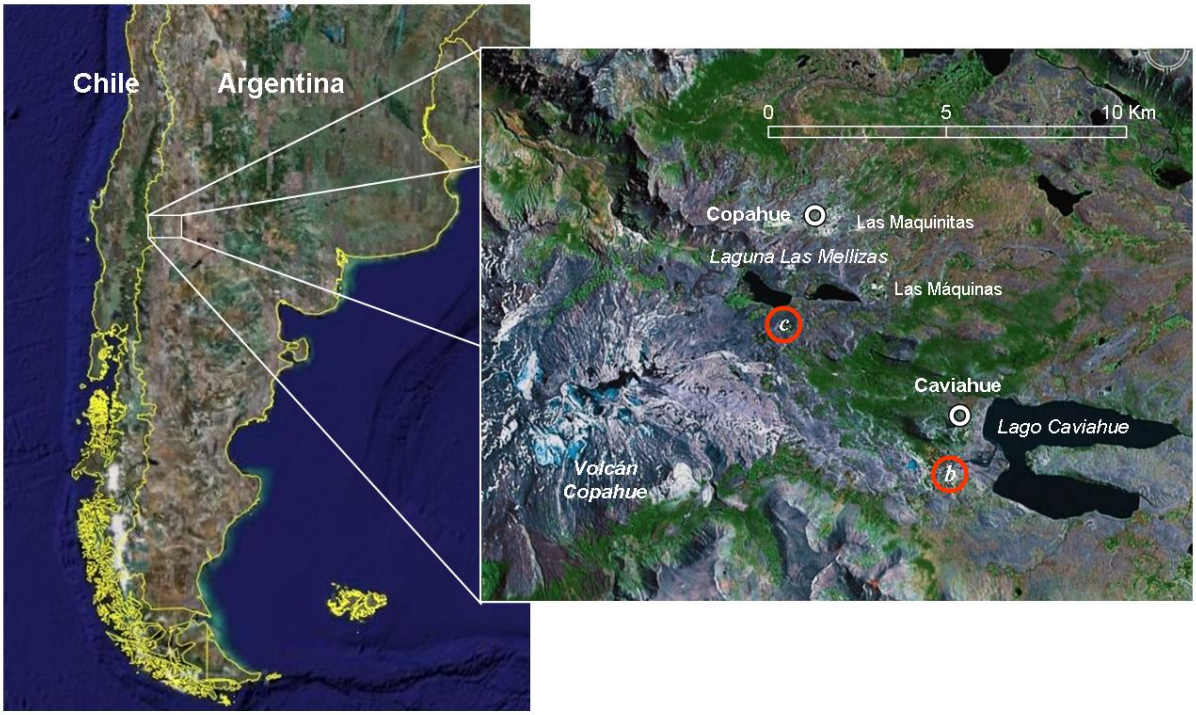
Figura 6.35. *Estructura superficial (hasta 1 km) de velocidad para las ondas S obtenida en los seis puntos considerados, algunos de los cuales admiten dos soluciones (de Almendros et al., 2006).*

Los nuevos sistemas se han seguido utilizando para la monitorización de la actividad en la isla en las últimas campañas (2005-06 y 2006-07), en las cuales se registraron fundamentalmente eventos de largo periodo (LPs), señales de tipo trémor volcánico y terremotos tectónicos regionales. Del análisis de la actividad se puede concluir que el volcán se encuentra en un estado de reposo y que no ha habido ningún síntoma de reactivación durante esos periodos (Serrano *et al.*, 2006).

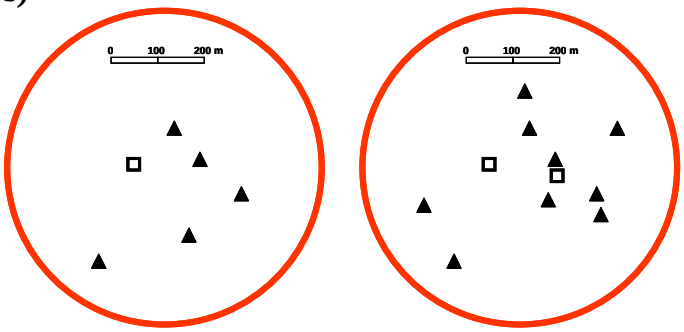
6.2.4. Volcán Copahue

El Copahue es un estratovolcán activo de tipo andesítico/basáltico-andesítico, que alcanza una altura máxima de 3001 m. Está situado en la frontera entre Chile y Argentina (figura 6.36), en la zona de transición entre los Andes Centrales y los Patagónicos (33.3º - 46º), sobre el borde occidental de la caldera de Agrio (Linares *et al.*, 1999). En la cumbre del volcán existen nueve cráteres alineados en la dirección 60º N, de los cuales el único que presenta actividad en la actualidad es el más oriental. Dentro del mismo se ha formado un lago de agua ácida (pH entre 0.3 y 0.8) de unos 125 m de diámetro, cuya superficie se encuentra cubierta por sulfuro líquido (Varekamp *et al.*, 2001; Caselli *et al.*, 2005).

a)



b)



c)

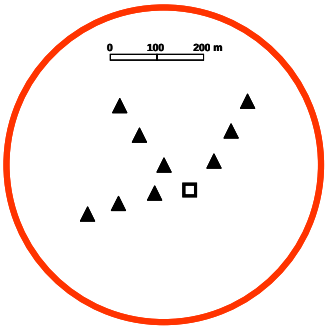


Figura 6.36. *Mapa del entorno del volcán Copahue (a), en el que se indica la posición de las localidades de Copahue y Caviahue, los campos geotermales de Las Máquinas y Las Maquinitas y la posición de las antenas sísmicas. En b) se muestra la distribución original de sensores para el array del Lago Caviahue (izquierda) y la configuración final, más densa (derecha). Los triángulos representan sensores verticales y los cuadrados estaciones de tres componentes. En c), configuración de la antena de Laguna Las Mellizas, implementada mediante uno de los nuevos sistemas portátiles (fotos: Google Earth).*

El Copahue es uno de los volcanes más activos de Argentina. Se tiene noticias de erupciones históricas desde el siglo XVIII, y en los últimos 100 años se conocen al menos 7 erupciones, caracterizadas por una moderada actividad explosiva (erupciones freáticas). Recientemente destacan dos ciclos eruptivos. El primero comenzó en julio de 1992 y terminó en septiembre de 1995 (Delpino y Bermúdez, 1995) y se caracterizó por una intensa actividad sísmica con algunos terremotos sentidos acompañando a la actividad explosiva. El segundo ciclo ocurrió entre julio y octubre de 2000, y está considerado como el proceso eruptivo más intenso del siglo XX en este volcán. Comenzó con emisión de lapilli, cenizas y bombas, algunas de las cuales fueron lanzadas a más de 8 km del área cratérica. Se observaron también flujos piroclásticos a lo largo de la ladera. Durante este ciclo el régimen explosivo evolucionó desde tipo freático a tipo estromboliano, aunque se siguieron observando explosiones moderadas con emisión de cenizas durante unas semanas. Igual que en la erupción del 92, la actividad explosiva fue acompañada de una intensa actividad sísmica con algunos terremotos sentidos.

La estructura superior de la región sobre la que se asienta el volcán ha sido estudiada mediante técnicas magnetotelúricas (Mamani *et al.*, 2000) y exploraciones gravimétricas. Los resultados podrían indicar la posición de una cámara magmática superficial localizada en la corteza superior. El sistema activo actual, el volcán Copahue, está situado a unos 15 km al oeste de esta anomalía.

Durante los meses de diciembre de 2003 a mayo de 2004, y posteriormente entre noviembre de 2004 y enero de 2005, se dispuso una antena sísmica en el área de influencia del volcán (Ibáñez *et al*., 2007). El *array* se montó usando uno de los antiguos módulos de dieciséis bits descritos en el capítulo anterior, y estaba compuesto por cinco sensores verticales y uno de tres componentes, todos basados en geófonos Mark L-25 con la respuesta extendida electrónicamente a 1 Hz (ver sección 5.1.2.1). La antena se montó en la proximidad de la ciudad de Caviahue (figura 6.36.a), en un área llana equidistante entre el cráter y el campo geotermal de Copahue. En la actualidad se ha aumentado la densidad de esta antena agregando uno de los nuevos sistemas portátiles, con la disposición de sensores que se muestra en la figura 6.36.b. Asimismo, se ha completado la configuración con otro de los nuevos sistemas, que se ha situado en una localización próxima a Laguna de Las Mellizas con una distribución geométrica en forma de flecha (figura 6.36.c).

Para realizar un análisis sistemático de los registros Ibáñez *et al.* (2007) aplicaron el método de correlación cruzada ZLCC (Frankel, 1994, Del Pezzo *et al.*, 1997, Almendros *et al.*, 1999) al conjunto de datos registrados, previamente filtrados en varias bandas de frecuencias (1-4, 4-8 y 8-12 Hz). Como ya se ha comentado anteriormente, el valor del coeficiente de correlación calculado con esta técnica aumenta con la coherencia de las señales, lo cual permite identificar la llegada de fases coherentes a través de las estaciones de la antena. La figura 6.37 muestra un ejemplo de un evento VT, en el que puede verse que los máximos en el coeficiente de correlación se alcanzan en la llegada de la onda P. Los autores seleccionaron todas las soluciones con un coeficiente de correlación mayor que 0.7, ya que empíricamente se observó que este valor sólo se alcanzaba con la primera llegada de las fases P. Este criterio se combinó con el estudio del contenido espectral de las señales para obtener un conjunto de 42 eventos VT, a los que se les calcularon los parámetros de propagación (azimut y lentitud aparente). Para llevar a cabo la localización de las fuentes mediante el método del trazado inverso del rayo se precisaba de un modelo de velocidad que no existe para el volcán Copahue, por lo que se decidió extrapolar a este área otros obtenidos para volcanes de características similares. Se consideraron dos posibles modelos, uno obtenido por Ibáñez *et al*. (2003b) para la Isla Decepción y otro por Almendros *et al.* (2000 y 2007) para el volcán Teide. La principal diferencia entre ambos es la existencia de una capa de baja velocidad superficial en el modelo de Isla Decepción. Las diferencias entre los resultados obtenidos a partir de los dos modelos no eran grandes, aunque aumentaban con la distancia

evento-receptor. Finalmente se asumió como válido el modelo de Isla Decepción, basándose en la similitud de los materiales geológicos en superficie en este entorno y el de Copahue.

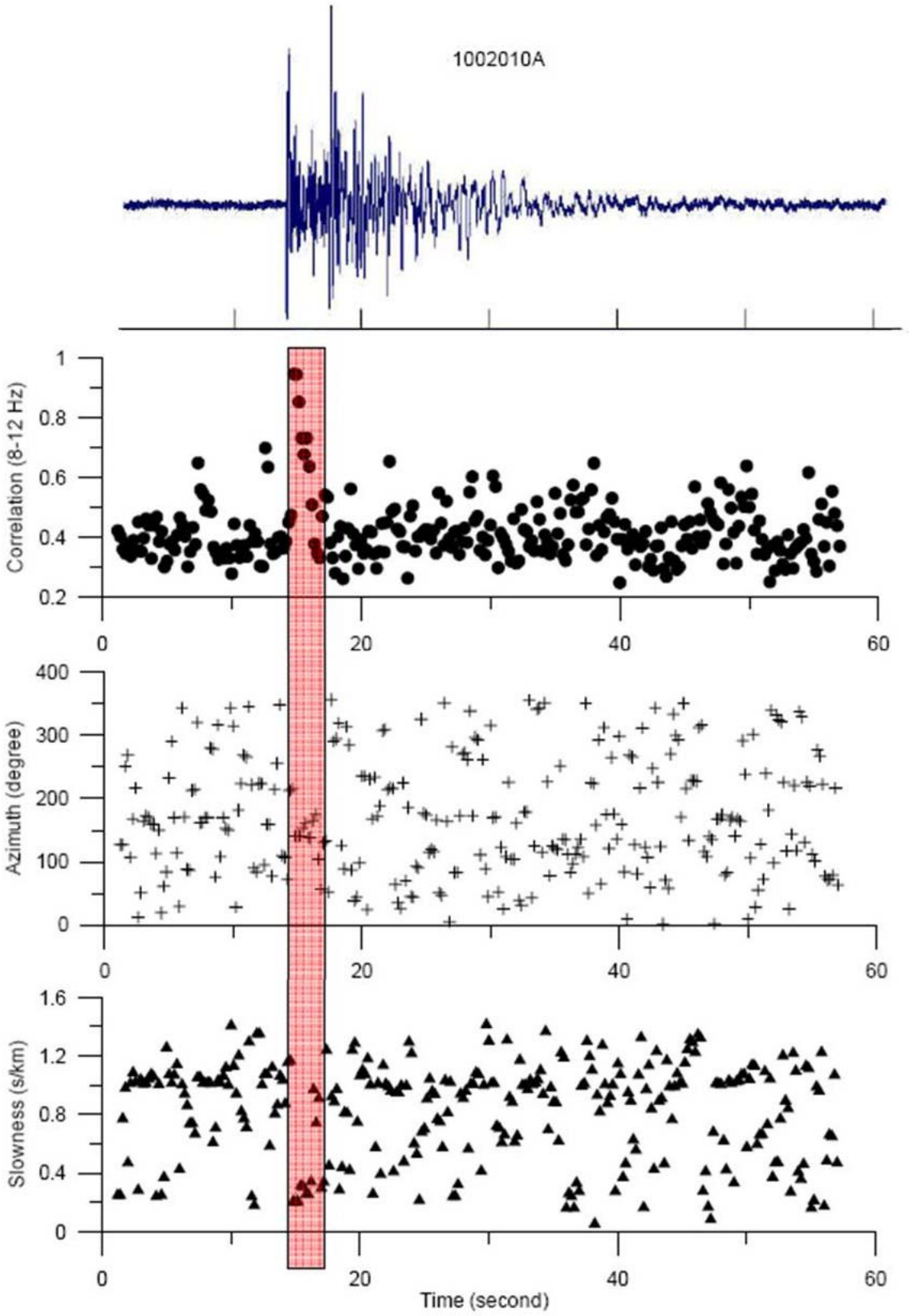


Figura 6.37. *Ejemplo de un terremoto VT ocurrido en el área del volcán de Copahue. Se observa un súbito incremento en el valor del coeficiente de correlación, que toma valores próximos a 1 con la llegada de la onda P. Simultáneamente los parámetros de propagación se ordenan, indicando la dirección de incidencia de los paquetes coherentes de ondas (modificado de Ibáñez et al., 2007).*

Una vez definido el modelo se realizó la localización de las fuentes para los eventos seleccionados, mediante el método del trazado inverso del rayo a partir del tiempo S-P estimado. La figura 6.38 muestra un mapa epicentral de los terremotos VT, en el que se observan dos grupos de eventos principales. El primero se sitúa al sur del lago Caviahue, muy cerca de la antena, y se caracterizó por la baja magnitud de los eventos y sus pequeñas profundidades focales, entre 0.6 y 2.8 km. Los autores interpretaron esta sismicidad de tipo VT como la consecuencia de la activación de fracturas de pequeñas dimensiones (algunas decenas de metros). Este tipo de actividad puede explicarse en ciertas condiciones a partir de la lubrificación de micro-fallas por fluidos a presiones altas (Chouet, 2003) que, como se ha visto anteriormente en este mismo capítulo, favorece los procesos de fractura. Posiblemente los fluidos implicados en esta lubrificación no sean magmáticos, sino agua contenida en la estructura geológica (un acuífero caliente).

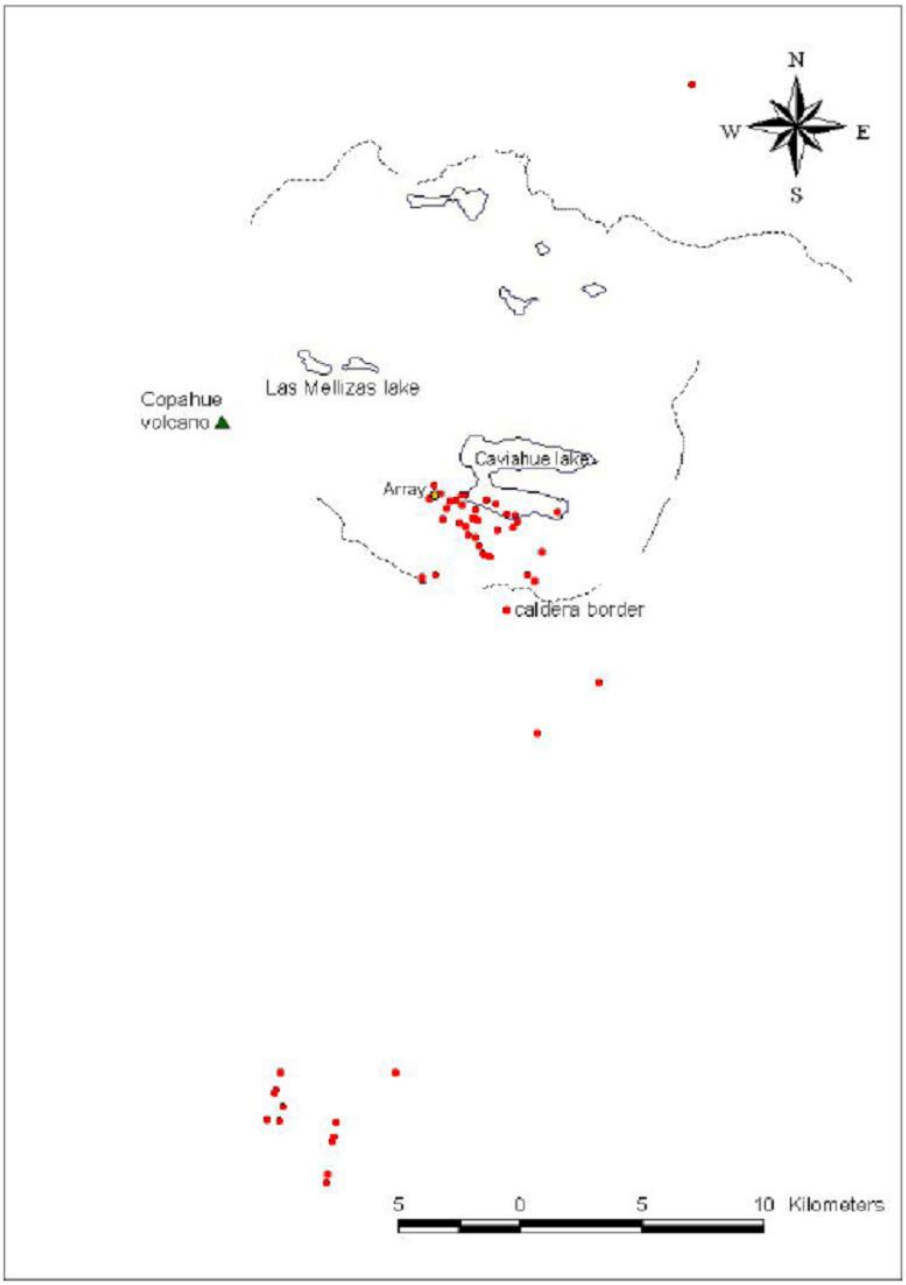


Figura 6.38. *Mapa de localización de los eventos VT utilizando el modelo de velocidad de la Isla Decepción (de Ibáñez et al., 2007).*

El segundo grupo de eventos se produjo a aproximadamente 1.5 km al sur del borde de la caldera, en un área en la que existen varios volcanes monogenéticos. Los eventos se caracterizaron por ser algo más energéticos (magnitudes entre 0.4 y 1.8) y presentar profundidades focales mayores que en el primer caso (unos 4.0 km). Esta sismicidad se asemeja más al régimen de volcanes como el Vesubio (Del Pezzo *et al.*, 2004).

El resto de la sismicidad se alinea en la dirección N-E y se da a profundidades mayores, de hasta 25 km, y podría estar relacionada con el sistema de fracturas asociado al borde oriental de la caldera.

Además de esta selección de eventos VT, se analizaron distintas muestras de ruido sísmico de fondo para estudiar su coherencia. Los parámetros del algoritmo STA/LTA del sistema de adquisición se fijaron para conseguir una elevada sensibilidad[60], de forma que el contenido de muchos de los ficheros de evento fuera simplemente ruido de fondo. Al aplicar la técnica de ZLCC a estos datos en la banda de 1-4 Hz se observó que los valores del coeficiente de correlación estaban distribuidos en un amplio margen, entre mínimos de alrededor de 0.3 y máximos de 0.95 (figura 6.39). Sin embargo, la distribución de azimuts con coeficiente de correlación mayor que 0.7 presenta un pico en un intervalo estrecho, entre 300º N y 40º N, con pocas soluciones fuera de dicho intervalo. En la distribución de la lentitud se observa el mismo patrón, con un pico en torno a 1 s/km, compatible con la propagación de ondas superficiales. Estos resultados indican la presencia de llegadas coherentes dentro del ruido de fondo no coherente en la banda de frecuencias entre 1 y 4 Hz.

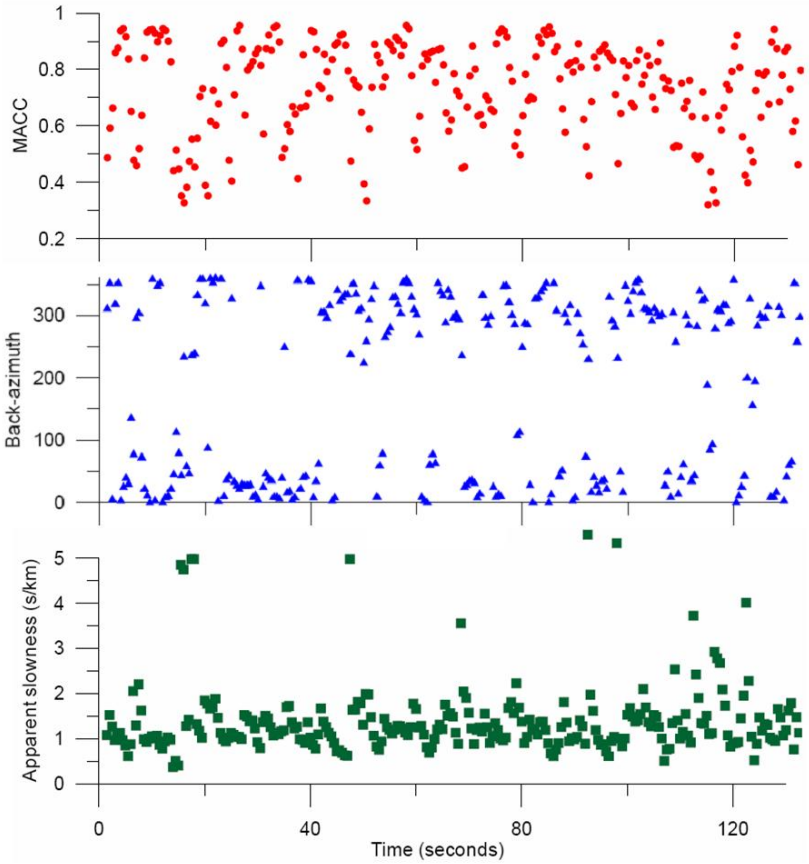


Figura 6.39. *Un ejemplo de las soluciones de* array *para una muestra de ruido sísmico de fondo. Arriba se muestra el coeficiente de correlación, en el que se observan numerosas soluciones con valores por encima de 0.8, indicando la llegada de fases coherentes. El azimut (centro) y lentitud aparente (abajo) están distribuidos en un intervalo estrecho que determina los valores preferentes de incidencia (de Ibáñez et al., 2007).*

[60] Hay varias formas de conseguirlo. Una de ellas, la que se utilizó en esta campaña, es reducir el valor de la constante de disparo (parámetro *Kdispa*, ver anexo III.C.1.a), y el número de estaciones para disparo, que en esta aplicación se fijó en 2. De esta forma el sistema declara un evento cuando el cociente STA/LTA es mayor que el umbral de disparo en sólo dos estaciones, condición que en la mayoría de los entornos y condiciones ambientales se cumple sin necesidad de estar registrando señales muy energéticas.

Para confirmar estos resultados se aumentó el umbral del coeficiente de correlación a 0.85 y se descartaron soluciones con valores altos pero aisladas en el tiempo, con lo cual el número de llegadas correlacionadas puede considerarse una estimación fiable del número de paquetes de onda coherentes presentes en el ruido de fondo. La figura 6.40 muestra los valores de azimut obtenidos mediante este procedimiento. En la figura 6.41 se han superpuesto estos resultados sobre el mapa del área. Puede observarse que el azimut apunta hacia el norte de la antena, en dirección a la población de Copahue y las zonas conocidas como Las Máquinas y Las Maquinitas, en las que existen emisiones visibles de gas a alta y baja energía (CO_2, H_2O y otros), varios lagos de lodo caliente, fuentes naturales de agua caliente, alteraciones hidrotermales de las rocas y otras manifestaciones geotermales. Por todo ello los autores interpretan la señal de fondo registrada por la antena como un trémor asociado a los procesos termales.

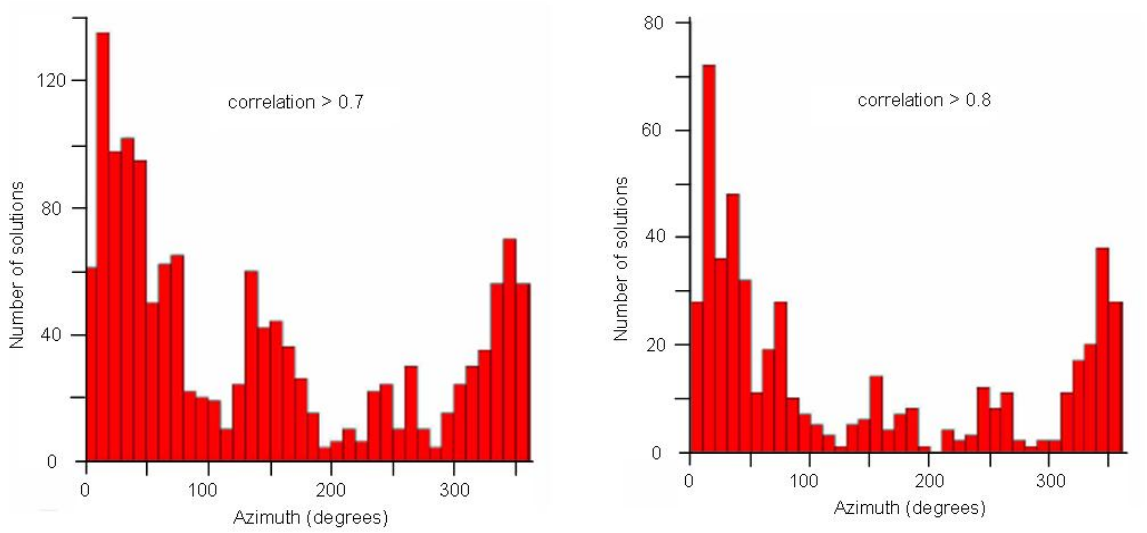


Figura 6.40. *Soluciones de* array *(azimut) para valores del coeficiente de correlación mayores que 0.7 (izquierda) y 0.8. Se observa que la mayoría de las soluciones están comprendidas en el intervalo 300º-40º (de Ibáñez et al., 2007).*

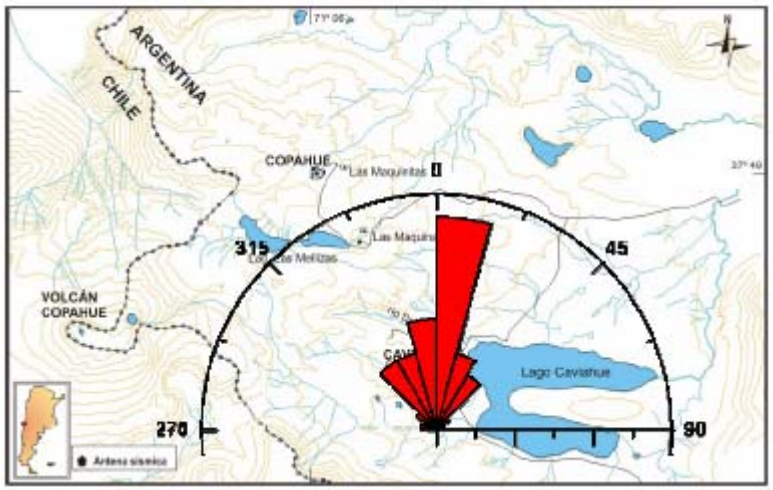


Figura 6.41. *Mapa del área próxima al volcán Copahue, en el que se han superpuesto la soluciones mejor correlacionadas para la señal de trémor volcánico (de Ibáñez et al., 2007).*

En el contexto de esta tesis, el aspecto más destacable de este estudio es la información que las antenas han sido capaces de suministrar. Con una sola antena ha sido posible localizar la actividad de tipo VT y detectar e identificar la fuente de una señal de trémor que probablemente habría pasado inadvertida a una red sísmica convencional. Pero además es posible sacar conclusiones sobre las señales que no se han detectado. En muchos

casos, las características de la sismicidad en un volcán permiten establecer sistemas seguros de alerta temprana. Sin embargo, para ser capaces de detectar cambios en las propiedades de la fuente o de medir la evolución de la dinámica del sistema se requiere conocer la actividad sísmica real durante los periodos de reposo. Este nivel de actividad de fondo varía de un volcán a otro. Así, mientras el Vesubio presenta un bajo nivel de eventos VT, pocos LP (unos dos al año) y ausencia de trémor (Bianco *et al.*, 2005), en la Isla Decepción se producen numerosos episodios de tipo LP y trémor, pero existe un bajo nivel de terremotos VT (ver, por ejemplo, Ibáñez *et al*., 2000).

En el caso que nos ocupa, la falta de evidencias de señales procedentes del volcán propiamente dicho induce a pensar que el sistema se halla en un periodo de reposo, lo cual resulta de gran interés para poder definir criterios orientados a una potencial detección de una reactivación del sistema. Cualquier cambio significativo en el nivel o características de sismicidad podría indicar posibles variaciones de la dinámica del sistema volcánico.

Esta campaña ha sido el primer estudio sistemático de la región volcánica del Copahue en periodo de reposo. La incorporación de uno de los nuevos sistemas portátiles en una localización próxima al lago Las Mellizas (figura 6.36) permitirá en breve obtener registros continuos de mayor resolución de las señales presentes en el área, mientras que el aumento de la densidad de la antena próxima al lago Caviahue incrementará la resolución espacial de sus registros. La posición relativa de ambos *arrays* permitirá realizar localizaciones conjuntas tanto de las señales actualmente presentes en el área (lo cual servirá para confirmar el origen de la señal de trémor detectada) como de las hipotéticas señales procedentes del volcán Copahue que probablemente se producirían en caso de una reactivación del sistema.

6.2.5. Volcán de Colima

El Volcán de Colima, también conocido como Volcán de Fuego o Zapotlán, es un estrato-volcán andesítico que alcanza una altura de casi 4 km sobre el nivel del mar. Está situado en una región tectónica extremadamente compleja (19.512º N, 103.617º W), donde interactúan las placas litosféricas Norteamericana, Pacífica, de Cocos y de Rivera (figura 6.42). Forma, junto con el volcán pleistoceno Nevado de Colima, el Complejo Volcánico de Colima, situado en el extremo oeste del Cinturón Volcánico Mexicano.

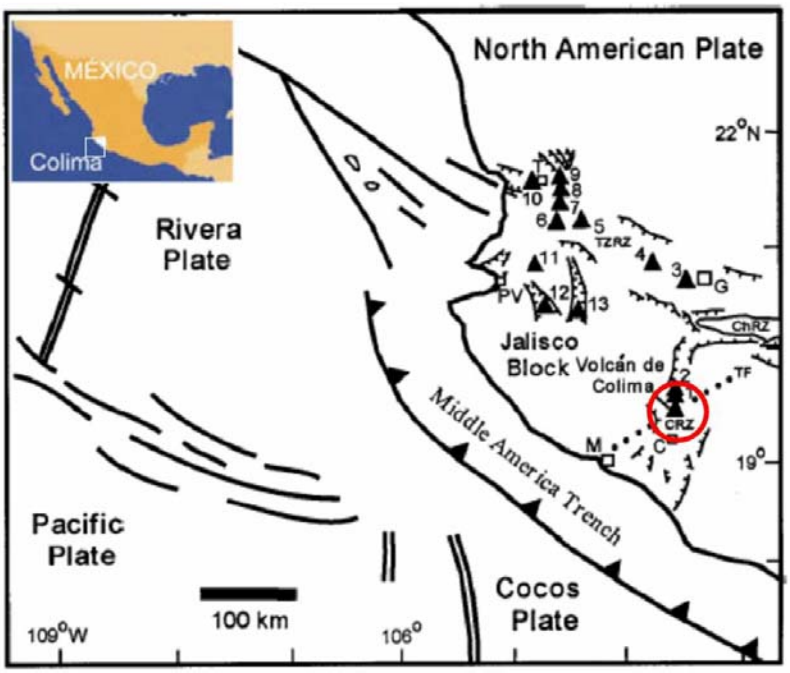


Figura 6.42. *Mapa de localización del Volcán de Colima (circulo rojo), en el que se muestran las cuatro placas tectónicas que interactúan en la región y otros sistemas volcánicos que forman parte de la fracción occidental del Cinturón Volcánico Mexicano (de Del Pezzo et al., 2007).*

El Colima es el volcán más activo de México. Desde 1560, fecha a la que se remonta la primera actividad documentada, se han producido 25 erupciones, de las cuales al menos seis han sido de gran magnitud e intensidad. Su actividad está caracterizada por presentar un amplio abanico de tipos eruptivos, desde crecimiento de domos y flujos lávicos acompañados de frecuentes avalanchas hasta intensas explosiones piroclásticas con expulsión y depósito de cenizas a distancias de cientos de kilómetros. Aunque en la actualidad una parte del área de riesgo es parque nacional, varias ciudades y pueblos, así como numerosos centros agrícolas e industriales e importantes líneas de comunicación quedan dentro del área vulnerable.

La fase de reactivación más reciente comenzó a finales de 1997, con un súbito incremento de la actividad sísmica, y se desarrolló en dos etapas. La primera se inició con la deformación del edificio volcánico durante unos doce meses y culminó con la producción de un flujo lávico de unos 80 días de duración (Navarro-Ochoa *et al*., 2002). La actividad se convirtió en explosiva durante los siguientes dos años (Zobin *et al*., 2002a y 2002b), registrándose en 1999 la mayor explosión desde 1913, que generó una pluma volcánica de 10 km de altura (Saucedo *et al*., 2002). Después de un periodo de relativa baja actividad, en mayo de 2001 se inició la segunda etapa del proceso de reactivación, con la generación de un nuevo domo de lava en el cráter. A principios de 2002 comenzaron los flujos de bloques de lava acompañados de algunos flujos piroclásticos y desprendimientos. Esta actividad culminó a mediados de 2003 con dos grandes explosiones, la segunda de las cuales formó una columna de cenizas de 3 km de altura acompañada de una serie de flujos piroclásticos de hasta 2.5 km que cubrieron prácticamente todo el volcán. La secuencia de explosiones de 2003 formó un nuevo cráter en la cumbre de 200 m de largo y 30 de profundidad. Basándose en el volumen

de material expulsado y en la altura de la columna, se le asignó al proceso eruptivo un VEI (Newhall y Self, 1982) de 1-2.

La actividad sísmica asociada a la reactivación de 1997 comenzó antes de la actividad volcánica superficial, y consistió en unos 600 microterremotos VT con magnitudes entre 0.5 y 2.7 distribuidos en cinco enjambres (Domínguez *et al*., 2001, Zobin *et al*., 2002b y 2002c). Actualmente la actividad sísmica asociada al régimen eruptivo comprende dos tipos de eventos, unos producidos por caídas de rocas y otros por explosiones (Zobin *et al*., 2006).

Desde finales del año 2005 hasta mediados de 2006 se llevó a cabo una campaña de adquisición de datos sísmicos en el Volcán de Colima, en la que participaron el *Osservatorio Vesuviano*, el Observatorio Vulcanológico de la Universidad de Colima[61] y el IAG (Del Pezzo *et al*., 2007). El experimento se enmarcó dentro de un proyecto para el estudio de volcanes explosivos de alto riesgo[62]. El Volcán de Colima fue seleccionado tanto por las características y el nivel de su actividad como por las facilidades logísticas que ofrece para el desarrollo de una campaña de varios meses de duración.

La instrumentación desplegada por el *Osservatorio Vesuviano* consistió en cuatro estaciones *Lennartz MARSlite* con sincronización mediante GPS y muestreo a 62.5 mps. Las estaciones estaban equipadas con sensores de banda ancha *Lennartz LE-3D/20s*. El IAG, por su parte, montó una antena sísmica basada en uno de los nuevos módulos que se han presentado en esta tesis. El *array* constaba de nueve estaciones verticales y una de tres componentes, todas ellas equipadas con sensores *Mark L25* de 4.5 Hz con la respuesta extendida electrónicamente a 1 Hz. Además de la instrumentación desplegada específicamente para esta campaña, se contaba con las estaciones fijas de la Red Sísmica de Colima, gestionada por el Observatorio Vulcanológico de la Universidad de Colima. El desarrollo de esta red comenzó en el año 1989, y actualmente cuenta con nueve estaciones distribuidas alrededor del volcán y otras doce a distancias de unos 35-40 km. Todas ellas transmiten los datos telemétricamente a un centro de recepción y procesamiento. La figura 6.43 muestra la configuración de la red desplegada durante la campaña. La distribución de sensores en la antena del IAG puede verse en la figura 6.44.

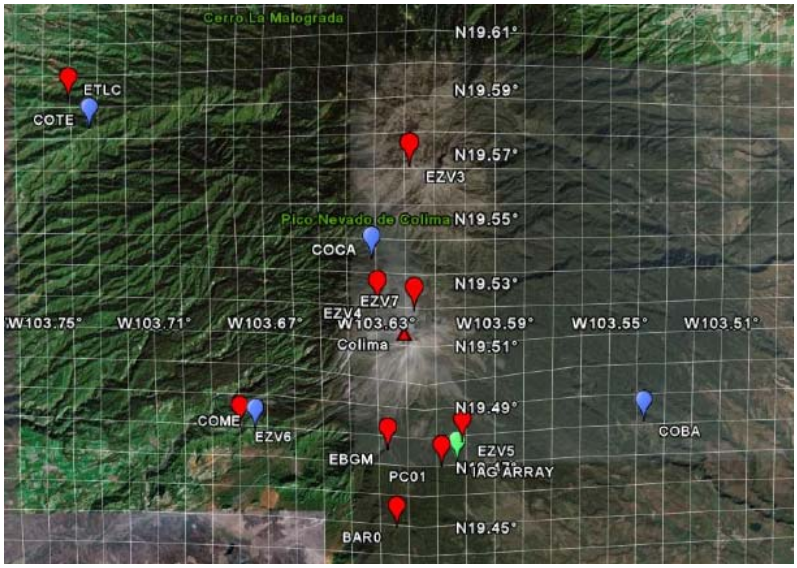


Figura 6.43. *Distribución de la instrumentación desplegada durante la campaña de 2005-06 en el Volcán de Colima. En rojo, estaciones de la Red Sísmica de Colima. En azul, las del* Osservatorio Vesuviano. *La posición de la antena sísmica del IAG se marca con un punto verde (de Del Pezzo et al., 2007).*

[61] www.ucol.mx/volcan/
[62] Proyecto FIRB (*Fondo per gli Investimenti della Ricerca di Base*) número 2-13-3-46-23.

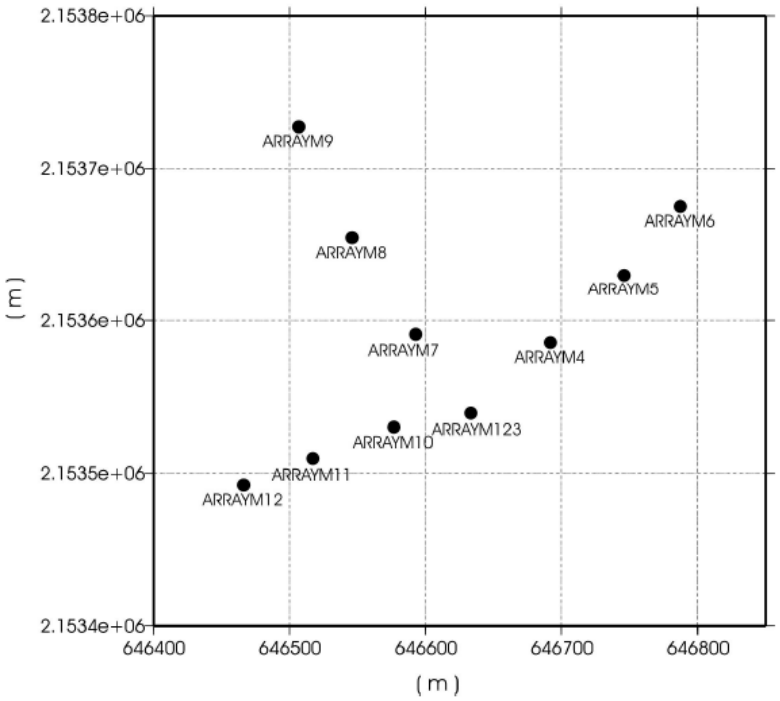


Figura 6.44. *Configuración geométrica de la antena sísmica en la campaña del Volcán de Colima de 2005-06. El punto central (ARRAYM123) sirve de emplazamiento para el módulo de control y la estación de tres componentes (de Del Pezzo et al., 2007).*

La red completa de instrumentación estuvo operativa desde noviembre de 2005 hasta principios de mayo de 2006. La antena sísmica funcionó además durante distintos periodos previos y posteriores a la campaña con distintas configuraciones, y en la actualidad sigue estando operativa como sistema de monitorización continua de la actividad del volcán. El experimento produjo una gran cantidad de datos de calidad, que actualmente están siendo analizados. Una primera inspección de los mismos indica que están compuestos básicamente de trémor volcánico de baja energía, eventos de largo periodo (LPs) y volcano-tectónicos (VTs). También se aprecian formas de onda generadas por desprendimientos de rocas y erupciones explosivas. El registro continuo de los nuevos sistemas permitirá además realizar análisis de ruido sísmico con los ficheros o fragmentos de ficheros que no presenten otras señales más energéticas. Para este tipo de estudios las nuevas antenas aportan una sensible ventaja respecto a los antiguos módulos de dieciséis bits, en los que – como se ha visto en la sección anterior – era necesario configurar el algoritmo de disparo con los valores necesarios para poder tomar muestras de ruido sin perder la capacidad de detección de otras señales más energéticas.

Hasta el momento se ha realizado un análisis preliminar de los datos correspondientes a explosiones, que ha proporcionado resultados interesantes (Ibáñez, comunicación personal). La figura 6.45 muestra un ejemplo de registro de este tipo de eventos, junto al espectrograma correspondiente. La explosión propiamente dicha corresponde a la línea *c*. Sin embargo, quizás lo más reseñable en el registro sea el paquete de ondas de baja frecuencia previo (línea *b*), que aparece en todos los eventos explosivos registrados con la antena. Antes se aprecia en el registro, aunque no en el espectrograma, un pequeño incremento de la amplitud de la señal sobre el nivel de ruido de fondo (línea *a*). Este incremento no se observa en todas las explosiones registradas.

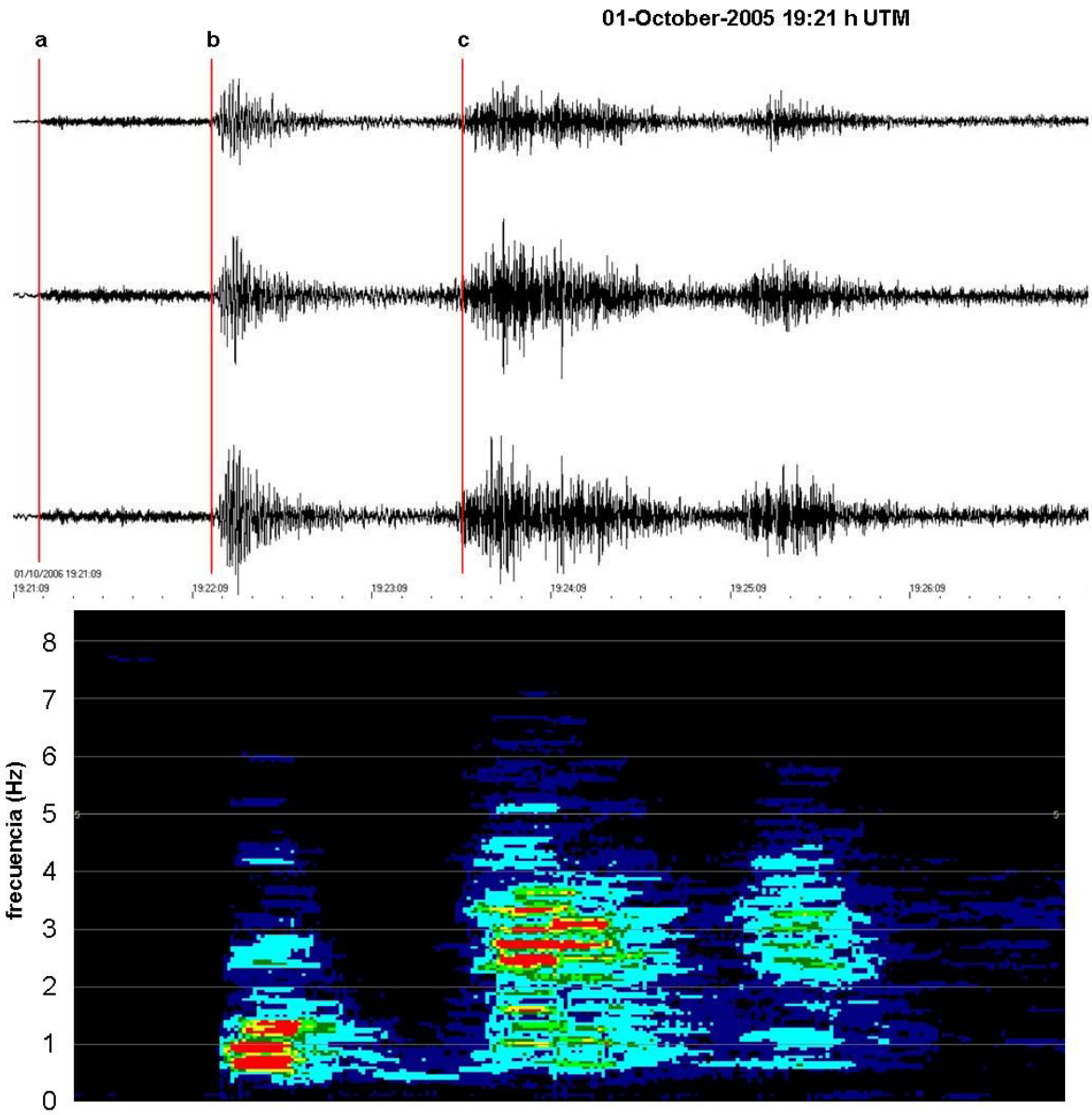


Figura 6.45. *Evento explosivo ocurrido en octubre de 2005. Arriba se muestran los registros vertical y horizontales de la estación de tres componentes de la antena. Abajo, el correspondiente espectrograma, en el que se aprecia el contenido en bajas frecuencias del primer paquete de ondas y en frecuencias mayores en el segundo grupo.*

La figura 6.46 muestra las soluciones de *array* para el mismo evento. Puede apreciarse cómo, coincidiendo con el incremento del nivel de la señal respecto al ruido de fondo (*a*), se produce un aumento progresivo de la correlación. Además los valores de los parámetros de propagación se van ordenando, pasando la lentitud aparente de valores correspondientes a ángulos de incidencia acusados (en torno a 0.4 s/km), que indican una posición profunda de la fuente, a otros compatibles con ondas superficiales (en torno a 1 s/km). La baja energía de esta primera fase hace que la variación en los distintos parámetros, especialmente en el azimut, no sea tan evidente como cuando llega el paquete de ondas de baja frecuencia (*b*). En este punto puede observarse claramente cómo los valores de azimut se distribuyen en un intervalo estrecho en torno a 300º-30º, correspondiente a la posición del cráter respecto a la antena. La lentitud aparente mantiene valores correspondientes a incidencias superficiales de las ondas y el coeficiente de correlación experimenta un súbito incremento hasta valores

próximos a 1, lo cual indica la llegada de paquetes de ondas coherentes. Posteriormente se mantienen valores altos, aunque algo menores, de la correlación, hasta la llegada del paquete de ondas correspondiente a la explosión propiamente dicha (*c*), en la que se produce un ligero incremento. Durante el tiempo que dura la explosión se mantiene la ordenación en los parámetros de propagación. Esta ordenación se hace más evidente en la figura 6.47, que muestra las soluciones de *array* para la hora completa de registro en la que se produjo la explosión de las figuras 6.45 y 6.46. Resulta claramente identificable, un poco antes de la mitad del registro, el cambio experimentado en los valores del coeficiente de correlación y en los parámetros de propagación.

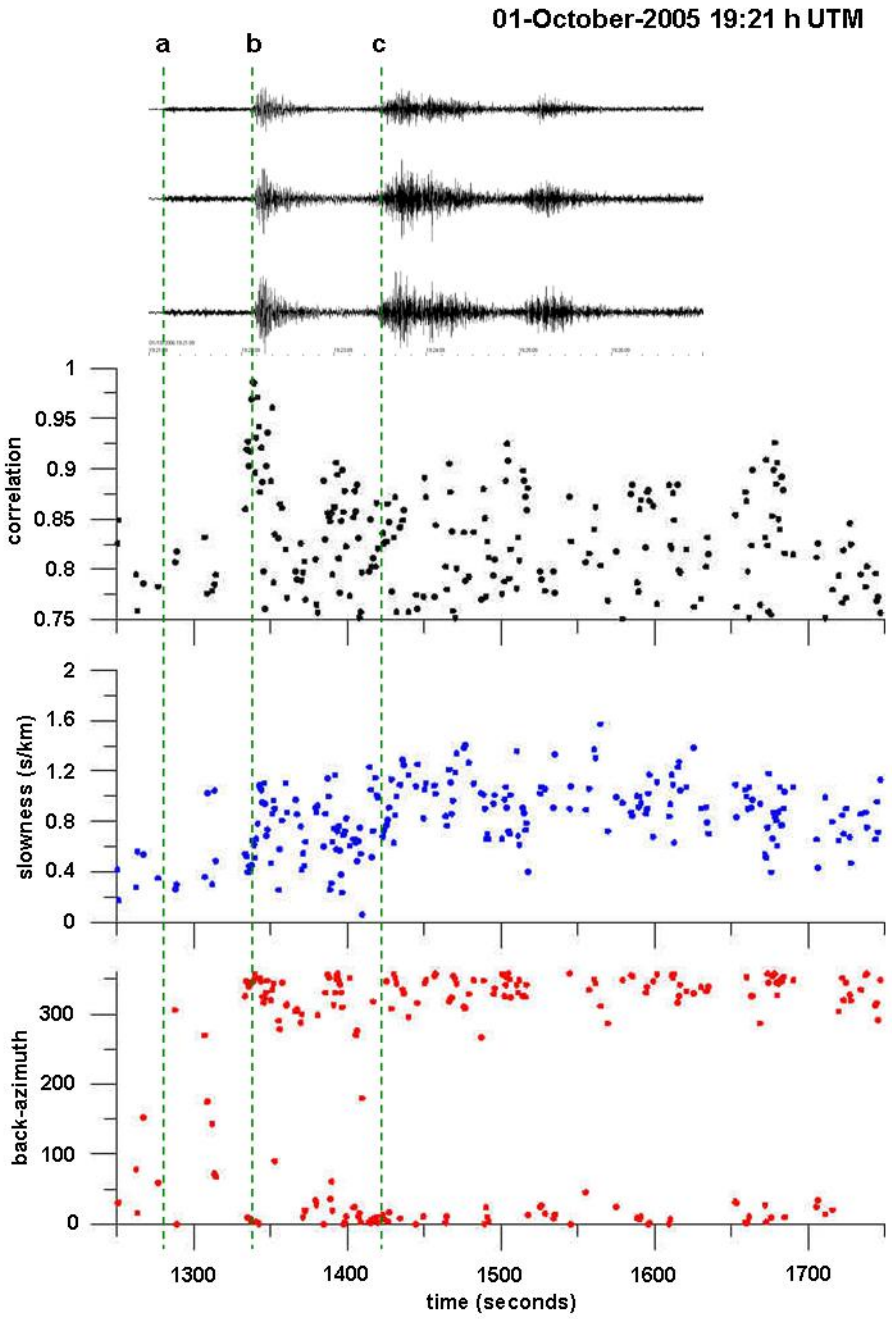


Figura 6.46. *Soluciones de array para la explosión de la figura 6.45 (explicaciones en el texto).*

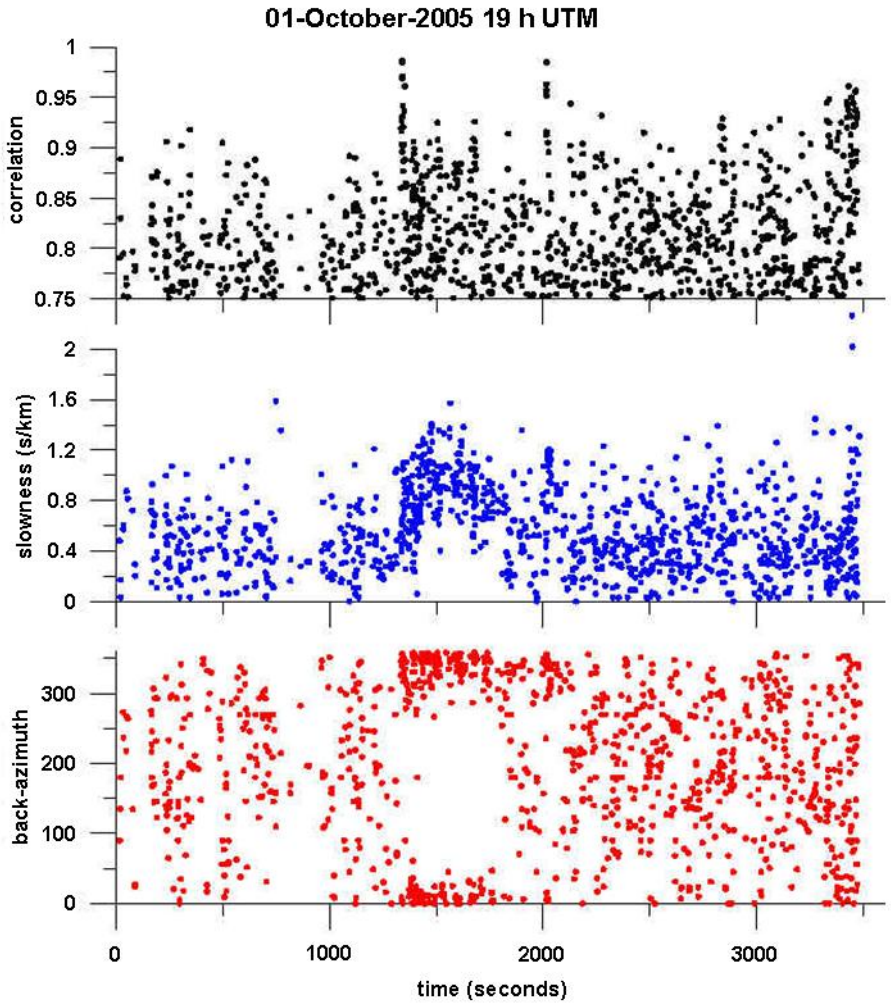


Figura 6.47. *Soluciones de array para una hora de registro. Puede apreciarse la ordenación en los parámetros de propagación y el brusco aumento en el coeficiente de correlación que se producen cuando tiene lugar la explosión de las figuras 6.45 y 6.46.*

Todo ello permite interpretar los eventos explosivos como una secuencia de tres procesos. En primer lugar, el crecimiento de una burbuja de gas que se manifestaría como el aumento del nivel de señal sobre el ruido de fondo observable en los registros (línea *a* en las figuras 6.45 y 6.46). El hecho de que esta parte de la señal no se observe en todos los registros puede ser debido simplemente a que en algunos casos el proceso no es lo suficientemente energético como para ser detectado por la antena. El origen del paquete de ondas de baja frecuencia (línea *b*) podría ser la ascensión de la burbuja a través del conducto, lo cual explicaría el aumento en la lentitud aparente desde valores correspondientes a posiciones relativamente profundas de la fuente (en torno a 1-1.5 km) hasta la superficie, siempre con azimuts orientados en dirección al cráter. La última parte del proceso consistiría en la explosión propiamente dicha, que se produciría cuando la presión diferencial entre el interior y el exterior de la burbuja rebasara las condiciones de equilibrio, y que crearía el paquete de ondas de alta frecuencia (línea *c*).

Paralelamente a estos análisis mediante técnicas de *array* se están realizando estudios de polarización a partir de los registros de la estación de tres componentes de la antena y de otras estaciones de la red sísmica que se desplegó durante la campaña. Los resultados

obtenidos son totalmente coherentes con los que se han descrito y podrían servir como confirmación del modelo propuesto.

Frente a otros entornos volcánicos descritos en esta tesis, el Volcán de Colima ofrece la gran ventaja de que se trata de un sistema muy activo con unas condiciones inmejorables para la adquisición de datos sísmicos, lo cual lo convierte en una inagotable fuente de señales. Como se ha comentado al describir las características del volcán, éste presenta una gran variedad de tipos eruptivos, incluyendo explosiones freáticas, efusiones de grandes bloques de lava y eventos explosivos de distintas magnitudes. Durante los meses que duró esta campaña la actividad consistía en pequeñas erupciones vulcanianas, que se producían a un ritmo de entre dos y tres al día, lo cual permitió la recolección de gran número de eventos de este tipo. El análisis preliminar mostrado aquí sobre datos correspondientes a explosiones, que ha permitido intuir la dinámica de estos procesos, no es sino una pequeña parte de los estudios que se realizarán y que permitirán conocer con más detalle la estructura y comportamiento del sistema volcánico.

CAPÍTULO 7

TRABAJO FUTURO

En este capítulo se discuten las posibles líneas de trabajo futuro relacionadas con los instrumentos que se han descrito. Estas líneas están orientadas principalmente a mejorar las características de los dispositivos que se han presentado, basándose sobre todo en la experiencia acumulada durante el tiempo en el que éstos han estado operativos. En otros casos la posibilidad de mejoras en el rendimiento está relacionada con la aparición de dispositivos electrónicos de mejores prestaciones que los existentes en el momento en que se plantearon los diseños.

En primer lugar se discutirán modificaciones aplicables en común a varios dispositivos (sección 7.1), para comentar posteriormente, en las secciones 7.2 a 7.4, algunos posibles cambios en las antenas del Gran Sasso, Vesubio y antenas portátiles del IAG, respectivamente. Por último se discutirán brevemente algunas ideas para futuros diseños (7.5).

7.1. Modificaciones comunes a varios sistemas

7.1.1. Nuevos conversores A/D de alta resolución

7.1.1.1. Criterios de selección y análisis

En el momento de plantear el primero de los diseños que se han presentado en este trabajo la oferta de conversores A/D de alta resolución y bajo coste era muy limitada. Posteriormente, la antena del Vesubio y las portátiles del IAG se concibieron a partir de la experiencia del *array* del Gran Sasso, cuyo comportamiento era muy satisfactorio en lo referente al sistema de muestreo, por lo que no se planteó un cambio de conversor A/D. En la actualidad la oferta de estos dispositivos es mucho más amplia, con varios modelos de diversos fabricantes que ofrecen características parecidas o mejores que las del AD7710. A continuación se presentan algunos de ellos y se analizan, en cada caso, las ventajas reales que se conseguirían con el uso del nuevo conversor. Hay que tener en cuenta, sin embargo, que en general la sustitución del conversor A/D no sería inmediata, sino que implicaría el diseño y desarrollo de nuevas tarjetas y programas. Por otra parte, si el procedimiento de control del nuevo conversor permite realizar tarjetas con las mismas interfaces de transmisión de datos y temporización que las basadas en el AD7710, el cambio se limitaría a sustituir éstas por aquellas, pero en caso contrario la modificación afectaría a otros subsistemas como las tarjetas PLL y/o interfaces de transmisión de datos. Según todo esto, la decisión de utilizar un modelo nuevo de conversor debería tomarse teniendo en cuenta el esfuerzo necesario en términos económicos, técnicos y de tiempo y analizando si las ventajas que se conseguirían merecen dicho esfuerzo.

Para la realización de este estudio se han consultado las características de unos ochenta conversores A/D de varios fabricantes. Todos ellos utilizan la técnica de conversión sigma-delta o delta-sigma y ofrecen una resolución nominal de 24 bits.

Hay que tener en cuenta que un análisis detallado de las características de todos los conversores consultados sería muy largo y tedioso y queda fuera del ámbito de esta tesis. Aquí se ha realizado tan solo un breve estudio, tomando como principal parámetro de análisis la resolución efectiva en función de la frecuencia de muestreo. Para poder comparar este parámetro en distintos conversores se ha tomado siempre el valor correspondiente a las condiciones en las que opera el AD7710 en los diseños descritos:

- Frecuencia de muestreo de 100 y 200 mps.
- Fondo de escala total de ±2.5V.
- Ganancia unidad en el amplificador de ganancia programable (PGA), en caso de que el dispositivo disponga de él.

Como se vio en el capítulo 2, en estas condiciones la resolución efectiva para el AD7710 vale unos 18.5 bits a 100 mps y unos 16 a 200 mps (tabla 2.1)[63]. En las hojas de características de los distintos conversores la resolución efectiva viene dada de diversas formas (bits efectivos, rango dinámico en dB, ruido rms de salida en μV o en ppm (partes por millón)), que para poder hacer comparaciones se ha pasado en todos los casos a bits efectivos.

Otros parámetros que se han considerado aquí, pero que deberían tenerse en cuenta de una forma más sistemática si se plantea seriamente la posibilidad de sustitución del AD7710, son:

- *Encapsulado*.- El montaje superficial (SMT, de *Surface Mount Technology*) presenta varias ventajas sobre el tradicional de tipo pasante, como la mayor densidad de integración, con la consiguiente disminución del área de los circuitos impresos y de la longitud de las pistas, o la reducción de los efectos parásitos en los componentes. La mayoría de los conversores analizados se presentan en encapsulados para este tipo de montaje, y sólo algunos ofrecen además la opción de encapsulados DIP para montaje pasante. Dentro de los encapsulados SMD (*Surface Mount Device*) hay algunos de mayor tamaño como los SOIC o PLCC, cuya manipulación resulta relativamente simple. Sin embargo, el reducido tamaño y alta densidad de pines de otros, como los tipos TSSOP o MQFP, hacen que su montaje e incluso la fabricación de tarjetas de circuito impreso mediante las técnicas habitualmente disponibles en laboratorios de electrónica se complique bastante.
- *Precio*.- Aunque todos los integrados analizados pueden considerarse de bajo coste, las diferencias entre unos modelos y otros no son despreciables, sobre todo teniendo en cuenta que en general en un sistema de antena son necesarios tantos conversores como canales. Como ejemplo de estas diferencias pueden citarse el ADS1210 de *Burr-Brown*, que puede conseguirse por algo más de 10 €, frente al AD1555 de *Analog Devices*, que cuesta unos 80, a los que hay que sumar otros 25 correspondientes al filtro digital integrado con el que opera (AD1556).
- *Disponibilidad*.- En los últimos años la adquisición de dispositivos y componentes electrónicos se ha facilitado mucho en nuestro país gracias, sobre todo, a la creación de grandes compañías como RS[64], Farnell[65] o Digi-Key[66], que suministran una amplia gama de componentes a precios razonables y sin necesidad de sobrepasar un número mínimo de unidades. No obstante, si se desea utilizar un modelo no catalogado por ninguno de estos suministradores puede resultar un problema, ya que algunos fabricantes no tienen en nuestro país otros distribuidores para ventas a minoristas y sólo ofrecen series de un número elevado de unidades, lo cual complica sobre todo la realización de prototipos para pruebas.

[63] En caso de que la resolución efectiva no se especifique para esas condiciones, se ha tomado el valor correspondiente a las más próximas posibles. Así, muchos conversores no permiten el muestreo a las frecuencias exactas de 100 y 200 Hz, o no tabulan el valor de resolución efectiva a dichas frecuencias. Entre ellos el propio AD7710, que en la tabla 2.1 no muestra el valor para 200 mps sino para 250 mps. El valor citado en toda la tesis de unos 16 bits efectivos a 200 mps se obtiene por interpolación.

[64] www.amidata.es

[65] es.farnell.com

[66] es.digikey.com

- *Rango de entrada*.- El rango de las entradas analógicas del AD7710, con las configuraciones descritas en este trabajo, es de ±2.5V, lo cual permite la conexión directa de los sensores al conversor. La mayoría de los nuevos conversores analizados tienen alimentación única, generalmente de +5V, lo que implica que generalmente no pueden gestionar entradas bipolares. Muchos de ellos pueden configurarse con entradas diferenciales con rangos de ±2.5V, pero siempre que ninguno de los dos terminales diferenciales esté a un nivel más negativo que la tierra analógica. En la práctica esto se traduce en la necesidad de diseñar una etapa previa para adaptar los niveles de señal, con los inconvenientes de consumo, ruido y electrónica adicional que ello implica.

7.1.1.2. Conversores sigma-delta monolíticos monocanal

A priori la elección más adecuada para sustituir el AD7710 en los sistemas descritos podría ser un integrado del mismo fabricante y familia (AD771x), que incluye varios conversores de características similares (AD7711, AD7711A, AD7712, AD7713 y AD7714)[67]. Algunos de ellos son incluso compatibles pin a pin con el AD7710, por lo que su sustitución sería inmediata. Sin embargo, ninguno ofrece un incremento significativo en la resolución efectiva para las frecuencias de interés, por lo que su uso no aportaría ninguna mejora destacable. Tampoco se obtienen mejoras significativas con otros conversores analizados, en unos casos porque las resoluciones efectivas son iguales o peores que en el AD7710 y en otros porque las frecuencias de muestreo son fijas y/o demasiado bajas para nuestras aplicaciones. Entre ellos podemos citar los modelos AD7718, 19, 83, 87, 89 y 91 de *Analog Devices*; el ADS1271 de *Burr-Brown (Texas Instruments)* y los LTC2400, 01, 02, 04, 08, 10, 11, 12, 13, 14, 15 y 18 de *Linear Technology*.

Un sustituto casi directo para los AD7710 en los sistemas descritos en este trabajo podría ser el HI7190 o el HI7191, que son básicamente el mismo integrado fabricado por empresas diferentes (*Harris Semiconductors* e *Intersil*, respectivamente). Las señales digitales de control son prácticamente las mismas que en el AD7710, incluyendo la señal /SYNC para sincronizar la operación de varios conversores. Como además las especificaciones vienen dadas para frecuencias de cristal de 10 MHz, igual que en el AD7710, el sistema de sincronismo podría estar basado en los mismos principios expuestos para los sistemas descritos. Es decir, en el control de una señal de referencia común de 9.8304 MHz que garantice la misma velocidad de operación de todos los conversores, junto con el envío de pulsos a través del pin /SYNC que aseguren que el muestreo está inicialmente sincronizado. Una última analogía de estos conversores con el AD7710 es que admiten alimentación simétrica de ±5 V y una referencia de +2.5 V, permitiendo entradas analógicas bipolares con el mismo rango que en los diseños descritos (±2.5 V). Así las cosas, la sustitución del AD7710 por alguno de los dos conversores citados se reduciría a realizar una pequeña placa de adaptación de las señales y a cambios menores en los programas, sobre todo orientados a la escritura de parámetros de operación y lectura de los datos. El beneficio que se obtendría en la resolución efectiva sería de unos 1.5 bits a 100 mps y casi 2.5 a 200 mps.

Aún mayor sería el incremento en la resolución efectiva con el ADS1210 de *Burr-Brown*, que alcanza hasta 22.5 bits efectivos a 100 mps y casi 22 a 200 mps, aunque para obtener estos valores es necesario mantener fijo el valor de la ganancia del amplificador programable.

[67] Por brevedad, y debido al gran número de conversores consultados, no se incluyen las referencias a las correspondientes hojas de características. Todas pueden obtenerse de los sitios web de los correspondientes fabricantes.

7.1.1.3. Moduladores analógicos y filtros digitales integrados

Otra opción disponible actualmente en el mercado son conversores sigma-delta que se implementan mediante dos integrados distintos. El primero de ellos corresponde al modulador analógico, mientras que el segundo implementa el filtro digital. La salida del primer integrado es una secuencia de ceros y unos de alta frecuencia, que es decimada por el filtro digital para dar una salida de 24 bits en serie. A este tipo de conversores pertenecen, por ejemplo, los integrados CS5322 (modulador analógico) y CS5320-21-23 (filtros digitales)[68], de *Crystal (Cirrus Logic)*, o la pareja AD1555/AD1556, de *Analog Devices*. Con estos dispositivos se obtiene una mejora significativa en la resolución efectiva, que es de unos 21 bits a 250 mps en ambos casos. En la opción de *Analog Devices* la mínima frecuencia de muestreo es de 250 mps, lo cual aumentaría el volumen de datos por tarjeta y haría necesario replantear también el sistema de transmisión. Para utilizar cualquiera de estos modelos sería necesario rediseñar la tarjeta de conversión, que se complicaría bastante por el hecho de que cada conversor consta de dos integrados (por tanto, serían necesarios seis integrados para una tarjeta de tres componentes) con un número de pines relativamente elevado (entre 28 y 44) y encapsulado de montaje superficial, en algunos casos de alta densidad.

Estas parejas de moduladores/filtros digitales se suelen utilizar conjuntamente, e incluso es normal que ambos integrados cuenten con hojas de características comunes. Otros fabricantes ofrecen moduladores analógicos independientes, como el ADS1201 de *Burr-Brown*, pensado para implementar el filtrado digital mediante un sistema de microprocesador o DSP, pero al que probablemente también se le podría aplicar alguno de los filtros digitales integrados anteriormente citados.

7.1.1.4. Conversores multicanal de alta resolución

Al presentar los conversores sigma-delta en el capítulo 2 se dijo que esta técnica de muestreo implicaba la necesidad de realizar un seguimiento continuo de la señal, salvo para frecuencias de muestreo muy bajas. Esto no es estrictamente cierto para algunos conversores de nueva generación, que mediante la aplicación de una frecuencia de muestreo primaria[69] mucho mayor que en los primeros modelos y utilizando filtros digitales con nuevas arquitecturas consiguen que cada muestra sea completamente independiente de la anterior. Este tipo de dispositivos aparece en la literatura como conversores sin tiempo de latencia ('*No Latency*'), de conmutación rápida entre canales ('*Fast Channel Switching*') o con tiempo de establecimiento del filtro digital de un único ciclo de conversión ('*Single Cycle Settling Filter*'). Esta tecnología da como resultado, entre otras cosas, una representación digital mucho más fiel de la señal analógica para entradas de alta frecuencia, incluso ante escalones de tensión, que con los conversores sigma-delta tradicionales se veían suavizados por efecto del filtrado digital. A este tipo de conversores pertenece, por ejemplo, la familia LTC24xx de *Linear Technology*, cuyo modelo más destacado es el LTC2440, que ofrece una resolución efectiva de 22.9 bits a 125 mps y 22.4 bits a 250 mps. No obstante, el principal beneficio que aporta la tecnología '*No Latency*' es la posibilidad de desarrollar conversores con capacidad de seguimiento continuo y simultáneo de varios canales analógicos. A este grupo de conversores multicanal pertenecen algunos de la misma familia LTC24xx, como el LTC2402 y LTC2412 (2 canales), LTC2404 (4 canales), LTC2408 y LTC2414 (8 canales) o el LTC2418 (16 canales). Sin embargo, ninguno de ellos resulta adecuado para nuestras aplicaciones, ya que en general están optimizados para frecuencias de muestreo más bajas y

[68] Una implementación de estación sísmica digital usando esta familia de conversores puede consultarse en Pazos, 2004.

[69] Por frecuencia de muestreo primaria se conoce la que se aplica en el modulador analógico, previo al filtrado digital.

su resolución efectiva se reduce mucho en el rango de los cientos de Hz. En el polo opuesto se encuentran los modelos ADS1251, 52, 53 y 54 de *Burr-Brown*, cuya resolución efectiva se mantiene prácticamente constante, en unos 19 bits, hasta frecuencias muy altas (hasta 40 kHz en el caso del ADS1252). A estas frecuencias este valor resulta muy interesante, pero en torno a los cientos de Hz no supone un incremento destacable respecto al AD7710. Otros conversores multicanal analizados que no ofrecen mejoras reseñables, ya sea por la frecuencia de muestreo limitada o por su reducida resolución efectiva a las frecuencias de interés, son los modelos ADS1232, 34, 40, 41, 42 y 43, de *Burr-Brown* o los CS5522, 24, 28, 40, 41 y 50 de *Crystal*.

Sí presentan características destacables otros modelos con capacidad multicanal, como los de la familia AD773x de *Analog Devices*. El AD7738, por ejemplo, ofrece una resolución efectiva de 21.4 bits a 372 mps y permite muestrear hasta 8 canales en modo común o 4 diferenciales. Cada uno de los canales se selecciona en instantes diferentes mediante un multiplexor implementado en el propio integrado, por lo que la frecuencia de muestreo debe dividirse entre el número de canales operativos. Así, para conseguir un sistema con cuatro canales diferenciales a 250 mps el conversor debería configurarse con una frecuencia de 1000 mps, lo cual reduce la resolución efectiva a 20.6 bits. Lógicamente, con esta solución el muestreo no sería simultáneo en todos los canales, por lo que habría que replantear el sistema de sincronismo y/o tener en cuenta el retardo correspondiente durante el procesado de los datos. En general la filosofía de un sistema basado en este tipo de conversores es completamente distinta a la descrita, por lo que es una opción a considerar sobre todo para nuevos diseños, pero no factible como mejora para los presentados en este trabajo. Entre otras dificultades para la utilización del AD7738, cabe destacar que su implementación no resulta sencilla por el tipo de encapsulado (TSSOP de 28 pines). A esta misma familia pertenecen también el AD7739, cuyas diferencias con el AD7738 son mínimas, y el AD7734 y AD7732, cuyas características son muy parecidas y se diferencian sobre todo en el número de canales analógicos que son capaces de gestionar. Por otra parte, el AD7731, también de *Analog Devices*, gestiona un máximo de tres canales diferenciales, pero la resolución efectiva es menor que en el caso del AD7738 y el máximo fondo de escala que admite es de ±1.28 V. No obstante, podría ser una opción interesante como conversor monocanal, puesto que mantiene una resolución efectiva de 21 bits a 200 mps y se presenta, además de en encapsulados SMD de alta y baja densidad, en encapsulados convencionales DIP24.

Otros conversores multicanal con los que a priori se obtendrían buenos resultados son los modelos ADS1255, 56 y 58 de *Burr-Brown*, que se distinguen básicamente en el número de canales analógicos que admiten (dos, ocho y dieciséis, respectivamente, o la mitad si se usan entradas diferenciales). Con los dos primeros es posible conseguir hasta 23.5 bits efectivos a 100 mps. Como en todos los conversores multicanal, la frecuencia de muestreo debe dividirse entre todos los canales operativos, aunque en este caso la resolución efectiva se mantiene por encima de los 22 bits para frecuencias de muestreo de algunos centenares de Hz (22.3 bits a 500 mps, 21.7 a 1000 mps). Así, con un solo conversor ADS1256 programado a 600 Hz se podría implementar un módulo de tres canales muestreando a 200 mps con más de 22 bits de resolución efectiva, lo cual supone una mejora teórica de unos seis bits respecto al AD7710. El ADS1258, por su parte, mantiene resoluciones efectivas de 21.6 bits a casi 2000 mps, lo cual permitiría en teoría implementar módulos de ocho canales diferenciales con frecuencias de más de 200 mps por canal. Como en el caso de los AD773x citados anteriormente, con todos estos modelos habría que plantear soluciones efectivas para el sistema de sincronismo y superar las dificultades impuestas por los encapsulados SMD de alta densidad.

Hay que destacar que no todos los conversores multicanal de alta resolución actualmente en el mercado implementan la tecnología '*No latency*'. Así, los modelos ADS1216, 17 y 18 de *Burr-Brown*, que admiten un máximo de ocho canales analógicos,

ofrecen un modo de operación con tiempo de establecimiento del filtro en un único ciclo de conversión, pero hay que analizar esta característica para comprender las verdaderas prestaciones del dispositivo. En este modo de operación es posible elegir el filtro digital de entre tres opciones. Sólo en una de ellas, el denominado filtro rápido, el tiempo de establecimiento es de un único ciclo de conversión, pero a cambio ofrece peores características de ruido que los otros dos filtros, lo cual redunda en peores resoluciones efectivas. El tercer filtro ofrece, por el contrario, mejores características de ruido y resoluciones efectivas máximas, pero su tiempo de establecimiento es de tres ciclos de conversión, lo cual implica que no se puede realizar una multiplexación de canales después de cada muestreo. Existe también un modo *auto*, en el que el conversor descarta la primera muestra después de la conmutación entre los distintos canales, y utiliza los filtros digitales más rápidos para obtener las dos siguientes muestras. Así, hasta la cuarta muestra después de cada conmutación no se utiliza el filtro que proporciona la máxima resolución efectiva. En la práctica, por tanto, para implementar un sistema multicanal con este tipo de conversores es necesario sacrificar resolución efectiva, ya sea por la elección de frecuencias de muestreo mayores que las necesarias o por la utilización de filtros digitales rápidos con peores características de ruido.

El mismo problema en la gestión de varios canales presentan los modelos CS5532 y CS5534 de *Cirrus Logic*, que admiten dos y cuatro canales analógicos, respectivamente. El tiempo de establecimiento del filtro digital para las frecuencias de interés es el correspondiente a varios periodos de muestreo (entre uno y algo más de tres, dependiendo del modo de operación), por lo que no es factible su uso como sistema multicanal con multiplexación de canales en cada muestra. Sin embargo, la resolución efectiva que se consigue en estos modelos es muy buena (23.6 bits a 120 mps, 20.2 bits a 240 mps), por lo que son opciones muy interesantes para la digitalización de un único canal analógico. Como en otros casos, su implementación en prototipos se ve dificultada por los encapsulados SMD de alta densidad en los que se presentan.

7.1.1.5. Otras aproximaciones al problema

Un concepto interesante con vistas a nuevos proyectos son los llamados microconversores, que implementan en un solo integrado uno o varios conversores A/D de alta resolución más un microcontrolador, lo cual permite una mayor integración en las tarjetas. A este grupo pertenecen, por ejemplo, los modelos ADuC824, 834, 845 y 847 de *Analog Devices*. Sin embargo, de momento las prestaciones de los conversores A/D que implementan no alcanzan las de algunos de los modelos comentados, ya sea por el valor limitado de la frecuencia de muestreo o por la resolución efectiva que ofrecen.

Muchos de los conversores A/D catalogados como de alta resolución y bajo coste son modelos optimizados para el muestreo de señales de audio (ejs.: familias AD183x y 193x de *Analog Devices*, CS42xx, 534x y 536x de *Cirrus Logic*, PCM180x de *Burr-Brown*, modelos UDA1342, 45 y 61 de *NXP Semiconductors*). Ofrecen en algunos casos resoluciones efectivas del orden de las de los modelos comentados (hasta 20 bits), pero frecuencias de muestreo mucho mayores (desde algunos kHz hasta centenares de kHz). En teoría sería posible adoptar esta frecuencia como un sobremuestreo inicial (ver, por ej., Havskov y Alguacil, 2004, pp. 95-97), para mediante técnicas de procesado digital quedarse con una secuencia de datos de salida de mayor resolución efectiva y con la frecuencia adecuada para nuestras aplicaciones. Sin embargo, al estar optimizados para señales de audio estos dispositivos suelen implementar filtros paso alta internos para eliminar la componente continua de las señales, filtros que en la mayoría de los casos también eliminan o atenúan las frecuencias de interés en sismología. Pero incluso aunque no fuera así, para obtener una secuencia de datos de alta resolución y frecuencia de cientos de Hz a partir de la señal muestreada de alta frecuencia sería necesario

el uso de dispositivos de altas prestaciones para el procesado de datos, como DSPs, lo cual encarecería y haría mucho más complejo el diseño. El esfuerzo necesario en términos de coste, complejidad y tiempo de desarrollo no merece la pena, sobre todo teniendo en cuenta las prestaciones que ofrecen algunos de los conversores presentados anteriormente, cuyas características se ajustan mucho mejor a nuestras necesidades.

7.1.2. Aumento de la frecuencia de muestreo

En las consideraciones de diseño de la antena del Gran Sasso (sección 3.3) se comentó que la frecuencia de muestreo inicialmente requerida para el dispositivo era de 200 mps, que finalmente se dejaron en 100. Esta decisión se justificó en la reducción en la resolución efectiva al aumentar la frecuencia de muestreo en los conversores que usan la técnica sigma-delta, y que concretamente en el AD7710 la limitaban a unos 16 bits para una frecuencia de muestreo de 200 mps.

Como se ha descrito en la sección anterior, actualmente existen conversores que mantienen una resolución efectiva alta para frecuencias de muestreo iguales o superiores a 200 mps, por lo que en caso de optar por alguno de estos dispositivos en el futuro lo más razonable sería permitir el muestreo a esta frecuencia. Pero incluso en los sistemas basados en el AD7710 descritos en este trabajo es posible que, dependiendo de los objetivos científicos que se planteen en cada aplicación, interese muestrear a 200 mps asumiendo la pérdida de resolución efectiva que ello implica. Como se ha apuntado ya en los capítulos de las antenas del Gran Sasso y las portátiles del IAG, esta posibilidad se tuvo en cuenta durante el diseño del *hardware*, por lo que los cambios se limitarían a modificaciones en los programas. Con el fin de reducir el volumen de datos a transmitir, lo más lógico sería que los controladores de adquisición (PCs en el Gran Sasso y PICs en las antenas portátiles) descartaran tras el muestreo el byte menos significativo de cada dato, pues a 200 mps no contiene información real sino sólo ruido electrónico. Otra opción sería configurar los conversores AD7710 con una resolución de 16 bits, lo cual obligaría a realizar algunos cambios en las rutinas de control de los conversores.

En cuanto a la antena del Vesubio, el muestreo a 200 mps era un requerimiento obligatorio de diseño, por lo que se implementó en su momento. Sin embargo, el volumen de datos generado a esa frecuencia de muestreo obligó a imponer una limitación en el número máximo de canales operativos, concretamente a 24 en lugar de los 48 disponibles a 50 o 100 mps. Por tanto, en este caso también resultaría interesante operar con 16 bits (ya sea configurando los conversores con esa resolución nominal o descartando el byte menos significativo), puesto que así se podría utilizar un número mayor de canales.

7.1.3. Tarjetas de PC industrial

Del mismo modo que actualmente existen conversores A/D con características superiores a las del AD7710, los PCs industriales han experimentado continuas mejoras en velocidad y prestaciones. Por citar algunos ejemplos, se pueden consultar las páginas web de las dos casas cuyos modelos se han utilizado en esta tesis, *Acrosser*[70] y *Lippert*[71], en las que aparecen varias tarjetas cuyas características son notablemente mejores que las de las utilizadas. Sin embargo, el aumento de las prestaciones de los PCs normalmente redunda en un mayor consumo, por lo que no tiene mucho sentido su sustitución a no ser que con ello se consiguiera una sensible mejora de las características del sistema en algún otro sentido. Éste

[70] www.acrosser.com
[71] www.lippert-at.com

podría ser el caso, por ejemplo, de la antena del Vesubio, si se decide instalar un sistema de telemetría para la transmisión de los datos (ver sección 7.3.1) o de las antenas portátiles del IAG, si se sustituyen los discos USB 1.1 por otros de tipo 2.0 que posibiliten la grabación directa de los datos (ver sección 7.4.3).

7.2. Modificaciones aplicables a la antena del Gran Sasso

7.2.1. Inicialización del oscilador patrón

La modificación más importante en la antena del Gran Sasso está relacionada con el sistema de sincronismo, concretamente con la inicialización del oscilador patrón. La concepción original del dispositivo contemplaba que el control del mismo se realizaría desde la sala de control situada en los LNGS. En base a esto, se decidió implementar la inicialización del oscilador patrón de forma manual, como se explicó en la sección 3.7.3. En la práctica, sin embargo, el sistema se controla remotamente a través de Internet, ya que el personal que gestiona el dispositivo y se encarga del procesado de los datos trabaja en las instalaciones de la Universidad de L'Aquila y del *Parco Scientifico e Tecnologico d'Abruzzo*, separadas varios kilómetros de los LNGS. Desde allí no se tiene acceso a información sobre el estado del oscilador patrón, por lo que es arriesgado realizar una inicialización de los relojes *software* de las estaciones sin estar seguros de que el reloj del oscilador patrón está en hora. Un pulso espúreo en la entrada de pulsos de segundo o un fallo de alimentación podría desincronizar el reloj del oscilador patrón, y una inicialización de los relojes *software* de las estaciones en esas circunstancias implicaría que se introduciría información de tiempo falsa en las tramas de datos. Aunque la experiencia indica que eso no suele ocurrir, en la práctica, por seguridad, sólo se inicializan los relojes *software* cuando se ha comprobado que la información de tiempo del reloj del oscilador patrón es correcta. O, lo que es lo mismo, cuando el personal responsable del dispositivo se desplaza a la sala del control de los LNGS para cualquier otra tarea de mantenimiento.

La solución más sencilla consistiría en mantener el servidor constantemente sincronizado con el tiempo universal. Existen diversos programas que permiten realizar este tipo de sincronizaciones periódicamente a través de Internet (ej.: Tardis 2000[72]), aunque con *Windows XP* y versiones posteriores no son necesarios, puesto que el propio sistema operativo lleva a cabo estas sincronizaciones de forma automática a partir de la información de tiempo del servidor horario seleccionado por el usuario. El oscilador patrón realizaría a su vez sincronizaciones periódicas con el servidor, mediante rutinas diseñadas al efecto que se añadirían al programa descrito en la sección 3.6.4. Como en el caso de la introducción manual de la información de tiempo comentada en 3.7.3, el servidor sólo suministraría información de tiempo real hasta el segundo, puesto que la inicialización de precisión (fracciones de segundo) la realizaría el oscilador patrón tomando como referencia la entrada de pulsos de segundo del patrón atómico. Se dispondría, por tanto, de un segundo completo para la transmisión de la información de tiempo real (año, mes, día, hora, minuto y segundo), con lo que sería posible utilizar uno de los puertos serie del PC a baja velocidad (por ejemplo, 9600 bps).

[72] www.kaska.demon.co.uk/download.htm

7.2.2. Aumento del número de estaciones

Como se vio en la sección 6.1, el análisis de los primeros datos de la antena llevado a cabo por Saccorotti *et al.*, 2006, obtuvo resultados bastante satisfactorios. No obstante, los autores vaticinaban un comportamiento mucho mejor del *array* si su configuración espacial se ampliaba, cosa que sucederá en un futuro próximo cuando se completen y acondicionen las nuevas galerías que se están excavando en los LNGS. Desde el punto de vista técnico la incorporación de estaciones al sistema no debería ser problemática. La estructura de tres niveles (estaciones, PCs nodales y servidores) con la que se diseñó el dispositivo permitirá la conexión de las nuevas estaciones a líneas serie adicionales, que con la incorporación de los correspondientes PCs nodales mantendrían la autonomía funcional del resto de las líneas serie. El número de PCs nodales que se podrían interconectar sólo estaría limitado por factores como el ancho de banda de la red local y la capacidad de gestión de datos del servidor, ninguno de los cuales impone, con las tecnologías actuales, limitaciones prácticas preocupantes.

7.3. Modificaciones aplicables a la antena del Vesubio

7.3.1. Transmisión telemétrica de los datos

Según se comentó en el capítulo 4, la estructura final prevista para la antena del Vesubio consiste en un par de dispositivos gemelos dispuestos en torno al volcán para localizar, mediante la técnica de cruce de rayos, las señales registradas. Para ello sería necesario que ambos dispositivos transmitieran sus datos hasta el centro de vigilancia del *Osservatorio Vesuviano* para su análisis y procesado. Por tanto, la modificación más inmediata en la antena del Vesubio sería la implementación de un sistema de telemetría. En el capítulo 1 ya se hizo una aproximación al problema. Como se comentó allí, la comunicación se podría implementar mediante transmisores y receptores digitales en las bandas de VHF o UHF o mediante dispositivos que utilicen la técnica de espectro disperso (ver sección 1.2.5.8).

La tabla 7.1 muestra algunos ejemplos de ambos tipos de transmisores. El volumen de datos para la configuración máxima de la antena del Vesubio (48 canales a 100 mps) sería de:

$$48 canales \times 100 mps \times 3 \frac{bytes}{muestra} \times 8 \frac{bits}{byte} = 115200 bps \qquad [7.1]$$

Por tanto, los transmisores y receptores de VHF y UHF de la tabla 7.1, que permiten velocidades máximas de 19200 baudios para canalizaciones de 25 kHz, son claramente insuficientes.

Los modelos 8 y 10, que utilizan la técnica de espectro disperso, estarían aparentemente en el límite de capacidad. Sin embargo, hay que tener en cuenta que su velocidad viene dada en baudios, por lo que la velocidad real de transmisión de información sería sensiblemente inferior y dependería del formato de transmisión utilizado. Por otra parte, la tabla indica las velocidades máximas teóricas, pero en la práctica éstas disminuyen en función de la distancia y de la contaminación del espectro en esa banda de frecuencias, que en una zona metropolitana como la ciudad de Nápoles probablemente sería considerable. Quizás aplicando a los datos algún algoritmo de compresión eficaz (lo cual sería recomendable en cualquier caso) sería posible utilizar estos modelos de radioenlaces. En último caso, además de la compresión se podría realizar una selección de los datos, transmitiendo por ejemplo sólo las componentes verticales de todos los nodos y las horizontales de algunos significativos

para realizar las localizaciones. El resto de los datos se mantendrían en el disco duro del módulo central y su volcado requeriría la presencia periódica de personal de mantenimiento.

La opción más adecuada de entre los modelos mostrados en la tabla 7.1 sería la número 11, que permite hasta 10 Mbaudios. Pese a la disminución de este valor en condiciones reales, se antoja más que suficiente para el volumen de datos generado por la antena. La distancia a cubrir por el enlace sería de entre 10 y 20 km, dependiendo del emplazamiento final del dispositivo (ver figura 4.1), lo cual, teniendo en cuenta la posible contaminación de la banda de frecuencias utilizada, haría necesario el uso de antenas directivas. Una solución probablemente más eficaz sería que el punto de recepción de los datos no se situara en la sede actual del *Osservatorio Vesuviano*, en la ciudad de Nápoles, sino en la antigua sede situada mucho más cerca del volcán. Entre los dos edificios operan varios enlaces telemétricos de banda ancha que transmiten hasta la sede actual los datos de varias de las estaciones de la red sísmica y que podrían aprovecharse también para los datos de la antena.

Nº	*Fabricante y modelo*	*Banda*	*Ancho canal (kHz)*	*Velocidad (baudios)*	*Potencia de salida (W)*	*Potencia consumida (W)*
1	Earth Data TR2400 TR	UHF	12.5 25.0	4800 9600	0.1-0.5	5.5 T 1.4 R
2	Nanometrics TX-2 T	VHF UHF	25.0	9600	2.0	6.6 T
3	Nanometrics RX-2 R	VHF UHF	25.0	9600		0.9 R
4	Wood and Douglas	VHF UHF	12.5 25.0	4800 9600	0.1-0.5	3.2 T 0.9 R
5	Esteem 192C TR	UHF	12.5 25.0	9600 19200	2	18 T 4.8 R
6	Adaptive Broadband, MDS4710 TR	VHF UHF	12.5	9600	0.1-0.5	24 T 1.5 R
7	Dataradio T-96SR	VHF UHF 900MHz	12.5 25.0	9600 19200	1.0-5.0	
8	Data-Linc SRM 6000 SS	900MHz	——	115k	0.1-1.0	7.8 T 1.2 R
9	Dataradio Integra H SS	900MHz	——	19200	0.1-1.0	7.8 T 2.6 R
10	FreeWave DGR-115 SS	900MHz	——	115k	1.0	7.2 T 1.2 R
11	WiLan vip 110-24 SS	2400MHz	——	10M	0.001- 0.1	5.0

Tabla 7.1. *Algunos ejemplos de radio enlaces digitales con aplicación a la sismología. Las abreviaturas corresponden a transmisor (T), receptor (R) y espectro disperso (SS). Todos los equipos se alimentan a 12V. La velocidad de transmisión mostrada para los modelos 8, 10 y 11 es la máxima que se puede obtener teóricamente, difícil de conseguir en condiciones reales y/o con distancias largas (modificado de Havskov y Alguacil, 2004).*

7.4. Modificaciones aplicables a las antenas portátiles del IAG

Teniendo en cuenta que nuestra experiencia en la operación de las nuevas antenas del IAG es mucho más amplia que en los otros dos sistemas, es lógico que sea en éstas en las que han surgido más ideas orientadas a la mejora de sus características. Entre ellas destacamos las siguientes:

7.4.1. Conexión de periféricos a la placa de PC

El conjunto de LEDs implementado en las antenas del IAG (sección 5.6.4) se ha revelado como un sistema de diagnóstico de bajo coste y consumo reducido, que permite verificar de forma intuitiva el correcto funcionamiento de los sistemas. También permite diagnosticar, hasta cierto punto, las causas de un posible malfuncionamiento, si bien para ello es necesario tener un conocimiento mínimo del procedimiento de arranque y operación del dispositivo. En caso de problemas el personal no especializado suele sentirse más cómodo si tiene acceso a toda la información a través de un monitor y puede interactuar con el programa mediante un teclado o ratón. Como se vio en el capítulo 5, es posible conectar al PC éstos y otros periféricos usuales mediante los cables de expansión suministrados con el mismo, si bien con el montaje presentado en esta memoria para hacerlo es necesario abrir la caja del PC y conocer dónde se conecta cada cable (o disponer de la documentación oportuna). Por tanto, una mejora inmediata sería hacer los conectores más accesibles, conectando los cables de expansión de forma permanente y situando los conectores, por ejemplo, en la tapa de la caja del PC (ver figura 5.19). Otra opción sería acceder al PC interno desde un PC portátil, mediante conexión a través del puerto *ethernet.* Como se vio anteriormente, este puerto es accesible desde el exterior de la maleta (figura 5.19.d), por lo que para permitir el acceso remoto desde otro PC sólo habría que configurar el sistema para que arrancara como en la antena del Vesubio.

7.4.2. Uso de tarjetas de memoria para almacenamiento temporal de datos

En la experiencia con los nuevos módulos los elementos que se han mostrado más frágiles han sido los discos duros internos. Debido principalmente a que contienen mecanismos formados por piezas móviles, su uso en entornos hostiles no ofrece las garantías que serían deseables. También son propensos a sufrir averías durante los desplazamientos, hasta tal punto que la experiencia aconseja que, aunque haya disponible un medio de transporte para el envío de material pesado (por ejemplo, por barco en el caso de campañas antárticas), los discos duros sean transportados por el propio personal como equipaje de mano. Aún así, son los elementos que más fallos dan, por lo que es necesario contar con, al menos, uno de repuesto para cada sistema.

Figura 7.1. *Imagen de una tarjeta de memoria Compact Flash de 256MB. Existen dos tipos de tarjetas, denominados Tipo I y Tipo II, que se diferencian en el grosor, de 3.3 y 5mm, respectivamente.*

La solución más inmediata a estos problemas sería sustituir los discos duros por algún tipo de soporte de almacenamiento masivo de estado sólido como memorias Flash, entre las cuales la

mejor opción por su disponibilidad y relación capacidad-precio son las tarjetas Compact Flash (figura 7.1). Esta posibilidad ya se planteó en su momento, pero el elevado precio de este tipo de memorias la hizo inviable económicamente. Sin embargo, el rápido desarrollo que han experimentado en los últimos tiempos y la gran demanda que tienen para sistemas de fotografía y grabación digital han hecho que actualmente sean bastante más asequibles. Esto, unido al hecho de que la capacidad necesaria no sería demasiado elevada (con la estructura de los sistemas y la estrategia de grabación en disco mostradas en este trabajo sólo tendrían que almacenar el sistema operativo y, como máximo, seis horas de datos) hace que actualmente este tipo de memorias constituyan una opción real a considerar.

Las tarjetas de PC *Lippert Cool Roadrunner II* utilizadas en las antenas portátiles del IAG cuentan con un zócalo para memorias de este tipo, por lo que su adaptación sería inmediata. Para otras placas de PC que no permiten la conexión directa es posible fabricar una tarjeta de interfaz IDE-Compact Flash. Las señales de control para este tipo de memorias son las mismas que para un disco duro IDE, por lo que la tarjeta de interfaz consiste básicamente en un adaptador entre los dos tipos de conectores (figura 7.2). La mayor dificultad estriba en la realización del circuito impreso y soldadura del conector de la tarjeta Compact Flash, ya que la separación entre sus pines es relativamente reducida (0.05 pulgadas).

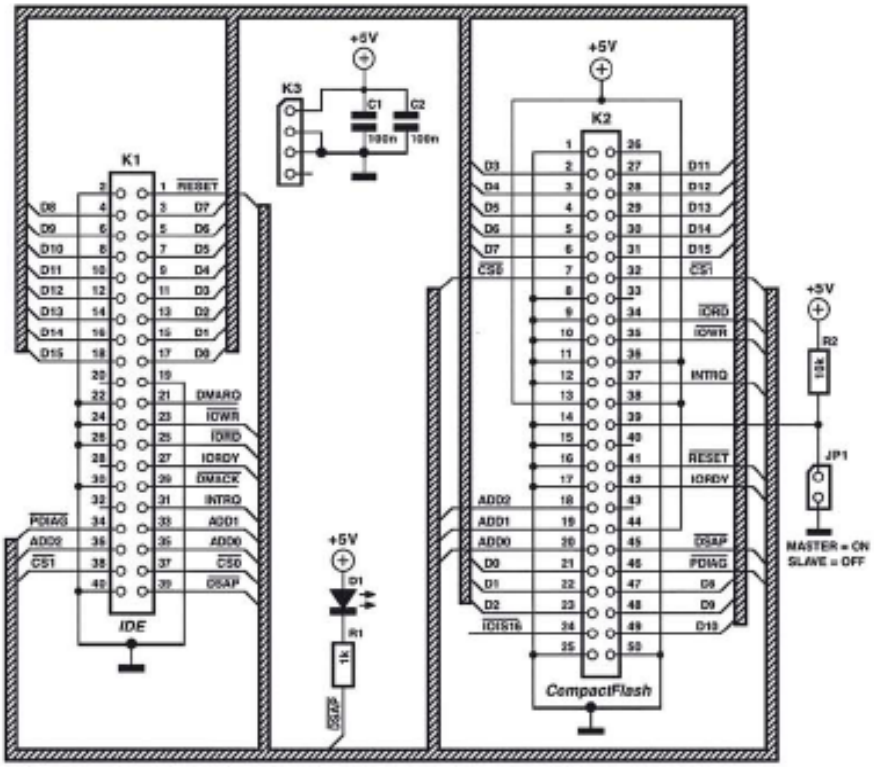


Figura 7.2. *Esquema eléctrico de la tarjeta de interfaz IDE-Compact Flash. El LED D1 se utiliza para monitorizar la lectura y escritura en la tarjeta de memoria. El jumper JP1 permite configurar la memoria como disco maestro (OFF) o esclavo (ON) (Goossens, 2003).*

Actualmente no existe ninguna dificultad en encontrar tarjetas Compact Flash de hasta 4 GB en cualquier establecimiento de fotografía y vídeo digital, y de hasta 12 GB en distribuidores especializados. En nuestro caso, sin embargo, no se necesitaría tanta capacidad. Las seis horas de datos requerirían, según el formato de ficheros descrito en el capítulo 5, unos 100 MB:

$$6horas \times 3600 \frac{s}{hora} \times (4tarjetas \times 100mps \times 12 \frac{bytes}{muestra} + 12bytesGPS) = 103939200bytes \qquad [7.2]$$

El mayor volumen de la memoria lo ocuparía el sistema operativo. Como se describió en el capítulo 5, el programa del PC se ejecuta bajo MSDOS, por lo que se podría limitar la instalación a este sistema operativo para reducir la capacidad de disco necesaria. Pero incluso con la configuración actual de los sistemas, con el sistema operativo *Windows 98*, probablemente sería suficiente con una tarjeta de 2GB, lo cual hoy en día no supone un desembolso excesivo.

Las tarjetas Flash en sustitución de los discos duros no sólo ofrecen la ventaja de su mayor robustez y fiabilidad, sino también de un ahorro considerable de espacio y, sobre todo, de consumo, que se reduce hasta apenas el 5% del de las unidades de disco duro convencionales.

7.4.3. Discos USB de nueva generación

El uso de discos USB para almacenamiento de los datos se ha revelado como una solución casi ideal, puesto que resuelve de una forma sencilla, rápida y práctica el problema del volcado de los antiguos módulos. La limitación en velocidad de estos dispositivos obligó a adoptar la solución descrita en el capítulo 5, consistente en grabar los datos temporalmente en el disco duro interno para luego volcarlos, cada seis horas, al disco USB, deteniendo durante el periodo de tiempo de volcado (unos tres minutos) la adquisición de datos.

La utilización de discos USB de nueva generación proporcionaría una alternativa más eficiente, puesto que su mayor velocidad de transferencia de datos (480 Mbps en el estándar 2.0 frente a los 12 Mbps que garantiza el estándar 1.1 utilizado en los sistemas) permitiría que la grabación se realizara directamente sobre este soporte en lugar de tener que utilizar el disco duro interno como búffer temporal.

El mayor problema que implica esta modificación es que el desembolso económico sería elevado, ya que además de los propios discos duros USB habría que cambiar las tarjetas de PC industrial por otras que soportaran el estándar USB 2.0. En caso de que las nuevas tarjetas fueran capaces de arrancar desde el disco duro USB se podría prescindir del disco duro interno o memorias FLASH, lo que supondría una notable simplificación y ahorro de costes.

7.4.4. Seislog

Cuando en el capítulo 5 se presentaron los programas de las nuevas antenas sísmicas del IAG se comentó que la concepción original de estos sistemas se basaba en la adopción de *Seislog* como programa de adquisición de datos en el PC. Finalmente, debido a los problemas que surgieron y que ya se han comentado, esta opción se abandonó y se decidió realizar un programa específico bajo MSDOS que implementara de modo muy simplificado las funcionalidades básicas de *Seislog*.

No obstante, la opción de *Seislog* no se descartó definitivamente, ya que las características de este programa y del paquete SeisXX del que forma parte lo convertían en un núcleo muy adecuado para sustentar la parte *software* de los nuevos módulos.

La implementación de *Seislog* en los nuevos sistemas se podría orientar desde dos líneas de trabajo distintas. La primera sería una continuación de los intentos de descubrir los fallos que se diseñaron en su momento. Implicaría la colaboración con el personal de la Universidad de Bergen (que se encargaría de realizar modificaciones en el *driver* para *Seislog* de las tarjetas SEISAD18) para encontrar la relación entre los errores en los ficheros de datos y distintos parámetros de operación de los sistemas (frecuencia de muestreo, número de canales, prestaciones del PC, estrategias de grabación a disco,…). La segunda línea partiría de la realización de una tarjeta de adquisición de prueba que gestionara, además del conversor A/D, un receptor GPS para insertar las marcas de tiempo como cabecera de los datos. No sería necesario el desarrollo de nuevos *drivers* para *Seislog* por parte de la Universidad de Bergen, puesto que bastaría con adoptar en las nuevas tarjetas el formato de datos de salida establecido para las tarjetas SEISAD18 (Abril *et al.*, 2003). El diseño de una tarjeta de este tipo en torno al conversor AD7710 no debería implicar demasiadas dificultades, teniendo en cuenta que el funcionamiento del mismo es bien conocido. Alternativamente, o bien en una etapa posterior, se podría testear *Seislog* con otras tarjetas de prueba con gestión de GPS y que implementaran alguno de los conversores A/D presentados en la sección 7.1.1, lo cual serviría para entender sus beneficios reales y calibrar la posibilidad de cambio de las tarjetas de conversión de todos los sistemas.

7.5. Algunas ideas para el futuro

Además de aprovechar las prestaciones, impensables hace unos años, de los nuevos conversores A/D y tarjetas de PC, las próximas generaciones de antenas sísmicas podrían incorporar algunas novedades, como son:

7.5.1. Reducción del consumo

A lo largo de toda esta tesis se ha comentado en numerosas ocasiones la problemática del consumo en sistemas embebidos como los que se han presentado. En dispositivos permanentes o casi permanentes este aspecto puede no ser tan importante como en el caso de los *arrays* portátiles. Como se ha visto, en la antena del Gran Sasso se contaba con una red de tensión alterna que suministraba potencia de manera ininterrumpida. En el caso del Vesubio, se podría montar en su emplazamiento casi permanente en la falda del volcán algún tipo de sistema de alimentación (por ejemplo, a base de paneles solares o aerogeneradores) para eliminar o reducir al mínimo las operaciones de mantenimiento relacionadas con el consumo. Sin embargo, en sistemas estrictamente portátiles como los nuevos módulos del IAG, el consumo es el principal factor limitador de su autonomía, puesto que hoy en día la capacidad de los medios de almacenamiento no impone ninguna restricción en ese sentido.

En el diseño de las antenas del Vesubio y del IAG el consumo ha sido un factor determinante. En ambos casos su reducción se ha buscado sobre todo en la etapa de selección de los distintos elementos que se iban a utilizar. Sin embargo, existen otras estrategias para obtener valores menores de consumo, como puede ser el diseño de módulos de operación intermitente. Es decir, que algunos de los subsistemas – idealmente, los de mayor consumo – se diseñarían para operar durante breves intervalos de tiempo, entrando en cuanto fuera posible en modo de espera o *standby*.

En nuestro caso el ejemplo más claro es el de los receptores GPS. Pese al cada vez menor consumo que necesitan, en un sistema similar a las antenas portátiles del IAG éste puede ser equivalente al del resto de la electrónica. El diseño de un módulo de reloj

intermitente podría ser parecido al que se muestra en la figura 7.3. El elemento central sería una tarjeta inteligente con un microcontrolador que controlase la frecuencia de salida de un oscilador de precisión basado en un VCXO o TCXO[73], de forma parecida a como se ha visto en el oscilador patrón del Gran Sasso o en las tarjetas PLL de los sistemas del Vesubio e IAG. Esta tarjeta generaría las señales de salida necesarias para la sincronización de la adquisición de datos, sincronizándose a su vez con el receptor GPS cada cierto tiempo. Para ello tendría una línea de control que conectaría o desconectaría la fuente de alimentación del GPS, o conmutaría entre estado activo o de espera si el receptor cuenta con esa posibilidad.

Los discos duros son otros de los mayores responsables del aumento del consumo de un sistema. Aunque, como se ha indicado en la sección 7.4.2, en determinadas aplicaciones de futuros instrumentos probablemente puedan sustituirse por memorias de estado sólido, cuando se requieran grandes capacidades de almacenamiento seguirán siendo necesarios. En estos casos se podría diseñar un sistema que, a semejanza del descrito para los receptores GPS, permitiera desconectar totalmente el disco durante periodos de tiempo relativamente largos durante los cuales los datos se almacenarían en memorias Flash o similares.

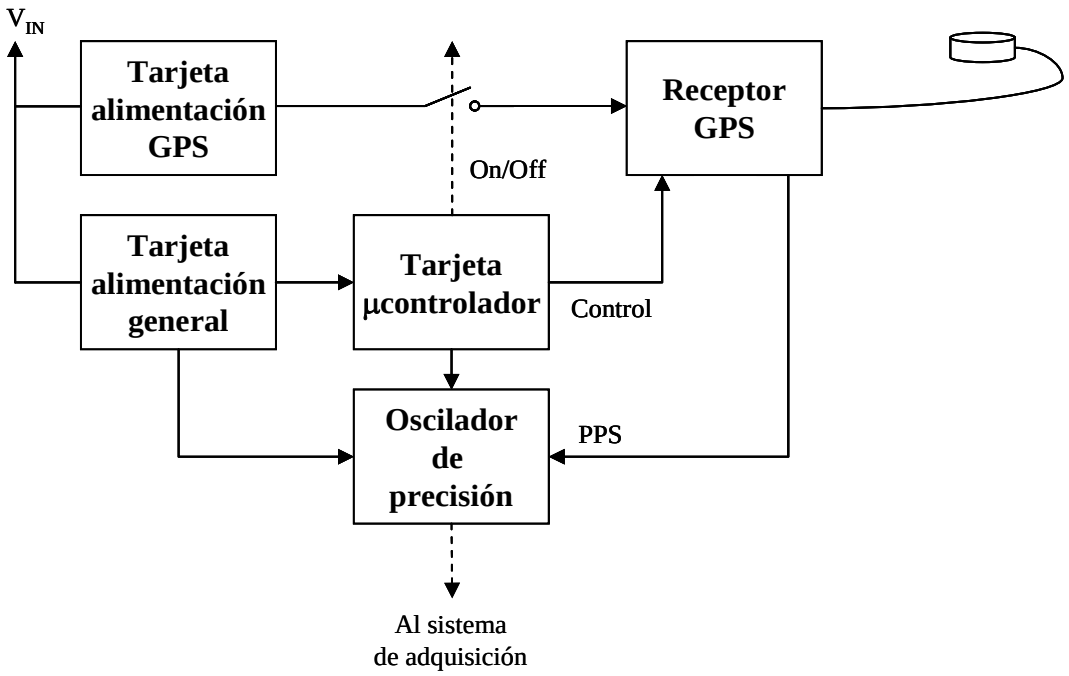


Figura 7.3. *Diagrama de bloques de un módulo de reloj de bajo consumo basado en la desconexión temporal del receptor GPS. El microcontrolador establece la duración de los intervalos de desconexión, durante los cuales las señales de sincronización del sistema de adquisición son generadas por el oscilador de precisión. Éste se resincroniza con el receptor GPS cuando el microcontrolador activa su alimentación y comprueba que el estado de recepción de la señal de los satélites, y por tanto la calidad del pulso de segundo, es óptimo.*

[73] TCXO: Oscilador a cristal compensado en temperatura. Como su propio nombre indica, son osciladores de precisión capaces de compensar las derivas producidas por cambios de la temperatura.

7.5.2. Procesado de datos en tiempo real

Debido al elevado volumen de datos y a la complejidad de las técnicas de procesado asociados a los dispositivos de *array*, uno de los aspectos que no se han podido resolver de momento es el procesado en tiempo real. Teniendo en cuenta el gran desarrollo actual de los equipos informáticos, no es impensable que se consiga en un futuro próximo.

Pese a que en general las antenas sólo suministran información sobre el azimut de la onda incidente y su velocidad aparente (que en general puede convertirse en ángulo de incidencia), existen técnicas como la del cruce de rayos o la de correlación cruzada promedio considerando frentes de onda circulares (Almendros *et al.*, 1999) que bajo determinadas circunstancias permiten calcular también la distancia a la fuente y con ello localizarla en el sentido clásico del término. Como se vio en la introducción, existen determinadas señales sísmicas, sobre todo de origen volcánico, que son difícilmente detectables (y mucho menos localizables) mediante redes sísmicas clásicas pero sí por dispositivos de antena. El procesado en tiempo real de este tipo de señales se convertiría en una valiosa herramienta para la vigilancia de regiones volcánicas, puesto que permitiría introducir en sistemas de vigilancia parámetros asociados a fenómenos no contemplados hasta ahora, como la migración de la fuente sísmica de señales de tipo trémor volcánico.

El algoritmo de localización en tiempo real podría inspirarse en el de detección automática implementado en la antena del Gran Sasso (sección 6.1). Así, podría constar de una primera fase de detección de señal coherente, utilizando como parámetro de control la potencia del espectro de lentitud o la coherencia multicanal. En caso de superarse un umbral prefijado en el parámetro de control se pasaría a una segunda fase, en la que se determinarían los parámetros de propagación de la señal (azimut y lentitud) durante el tiempo en el que ésta se mantuviese por encima del umbral de disparo.

7.5.3. *Arrays* sin cables

Desde el punto de vista logístico los cables se cuentan entre los elementos potencialmente más problemáticos de una antena sísmica, especialmente en el caso de dispositivos portátiles. Como se discutió en el capítulo 4, existen desventajas tanto si los cables se transportan con el resto de los módulos como si se decide adquirirlos en la región de despliegue del dispositivo. Incrementan drásticamente el precio – sobre todo si se utilizan conectores de tipo militar para todas las conexiones –, el peso y el volumen del equipo, lo cual puede suponer un verdadero problema si deben transportarse por vía aérea. Una vez en el área de despliegue el tendido de cables suele ser la tarea más complicada, especialmente con orografías escarpadas y/o vegetación densa, hasta el punto de que a veces condiciona la elección del emplazamiento de la antena. Y una vez operativo el dispositivo son susceptibles a cortes y roturas, ya sea por accidentes, vandalismo o a causa de la fauna autóctona (especialmente conejos y otros roedores, que encuentran un inexplicable placer en interrumpir líneas de transmisión de datos sísmicos). Un *array* sin cables eliminaría todos estos problemas.

Esta idea no es, ni mucho menos, nueva. De hecho, seguramente todos los que han sufrido las incomodidades de una campaña con *arrays* en un entorno medianamente hostil se lo habrán planteado alguna vez. Sin embargo, las dificultades técnicas para llevar a cabo un diseño de este tipo han sido, hasta hace pocos años, demasiadas como para plantearse desarrollarlo con un coste razonable. Actualmente la situación ha cambiado, y el proyecto es mucho más viable gracias, sobre todo, a la gran oferta y disponibilidad en el mercado de módulos transmisores y receptores para sistemas de control remoto y redes inalámbricas de ordenadores. En el diseño de una antena sísmica sin cables se cuenta con la ventaja, respecto a la transmisión de datos vista en la sección 7.3.1, de que las distancias que los enlaces deben

cubrir van a ser relativamente cortas y de que en general va a haber visión directa entre los puntos, por lo que se podría elegir entre una amplia gama de transmisores y receptores comerciales. Entre ellos los basados en técnicas de espectro disperso que, como se vio en el capítulo 1, permiten una comunicación fiable y presentan una alta inmunidad al ruido.

También es posible encontrar transmisores de FM de bajo consumo y coste reducido (unos diez euros) que cubren distancias de unos 250 m en condiciones óptimas, y receptores de varios canales que permitirían configurar una red en estrella para la recepción de datos de los distintos nodos (ver, por ejemplo, las familias RRFQ1 y 2 de la empresa RFSolutions[74]). Estos transmisores admiten un flujo continuo de datos a través de una línea serie, que transmiten con velocidades en el aire de hasta 9600 baudios. Con esta velocidad se podría transmitir el flujo de datos correspondiente a tres canales y 100 mps:

$$3canales \times 100mps \times 3\frac{bytes}{muestra} \times 8\frac{bits}{byte} = 7200bps \qquad [7.3]$$

En condiciones reales y contando con los bits de control necesarios, probablemente los 9600 bps quedarían un poco justos. No obstante, con un coste algo superior (pero todavía menor de 50 euros) se pueden encontrar módulos que permiten velocidades mayores, del orden de 40000 baudios y con los mismos rangos de operación, unos 250m (ver, por ejemplo, los modelos RF650 para bandas de 433 y 868 MHz del mismo fabricante).

Además de los dispositivos con entrada serie, existe actualmente una gran variedad de módulos diseñados para la conexión de ordenadores mediante redes inalámbricas. Estos dispositivos permiten velocidades mucho mayores que los anteriores, y normalmente se controlan a través de los puertos USB o e*thernet*. En una antena sísmica sin cables la necesidad de mantener reducido el consumo haría inviable el uso de PCs en cada punto de adquisición de datos. Sin embargo, en la actualidad esto no supone ningún problema, pues existen microcontroladores que implementan módulos de control para estos puertos. En la familia de los PICs podrían utilizarse, por ejemplo, los modelos PIC18F2455, 18F2550, 18F4455 o 18F4550 con controlador USB, o la familia PIC18F97J60 para gestión de un puerto *ethernet*.

El planteamiento de un dispositivo de antena sin cables podría ser el de la figura 7.4. El sistema estaría formado por módulos de tres canales, cada uno con un microcontrolador que se encargaría del control de los conversores y proporcionaría el flujo de datos a un transmisor, que mandaría los datos al nodo central. En caso de usar sensores triaxiales el módulo controlador se situaría junto al sensor, con lo cual se eliminarían los cables completamente. Si se utilizaran sensores de una sola componente sería necesario tender cable desde el módulo controlador hasta dos de los tres sensores, por lo que no se eliminarían totalmente los cables pero se reduciría notablemente su longitud.

Quizás el mayor problema asociado a una antena sísmica sin cables sea el de la alimentación de los distintos módulos. Como es lógico, la propia filosofía de prescindir de los cables descarta la posibilidad de alimentar todo el sistema con una única batería, por lo que cada nodo necesitaría contar con una fuente de alimentación independiente. Éstas podrían consistir en pequeñas baterías o incluso en pilas de alta capacidad, si se consigue que el consumo de los nodos (determinado principalmente por el del sistema de telemetría) sea extremadamente bajo. En el caso de sistemas permanentes sería necesario instalar un pequeño panel solar u otro sistema de carga en cada nodo.

[74] www.rfsolutions.co.uk

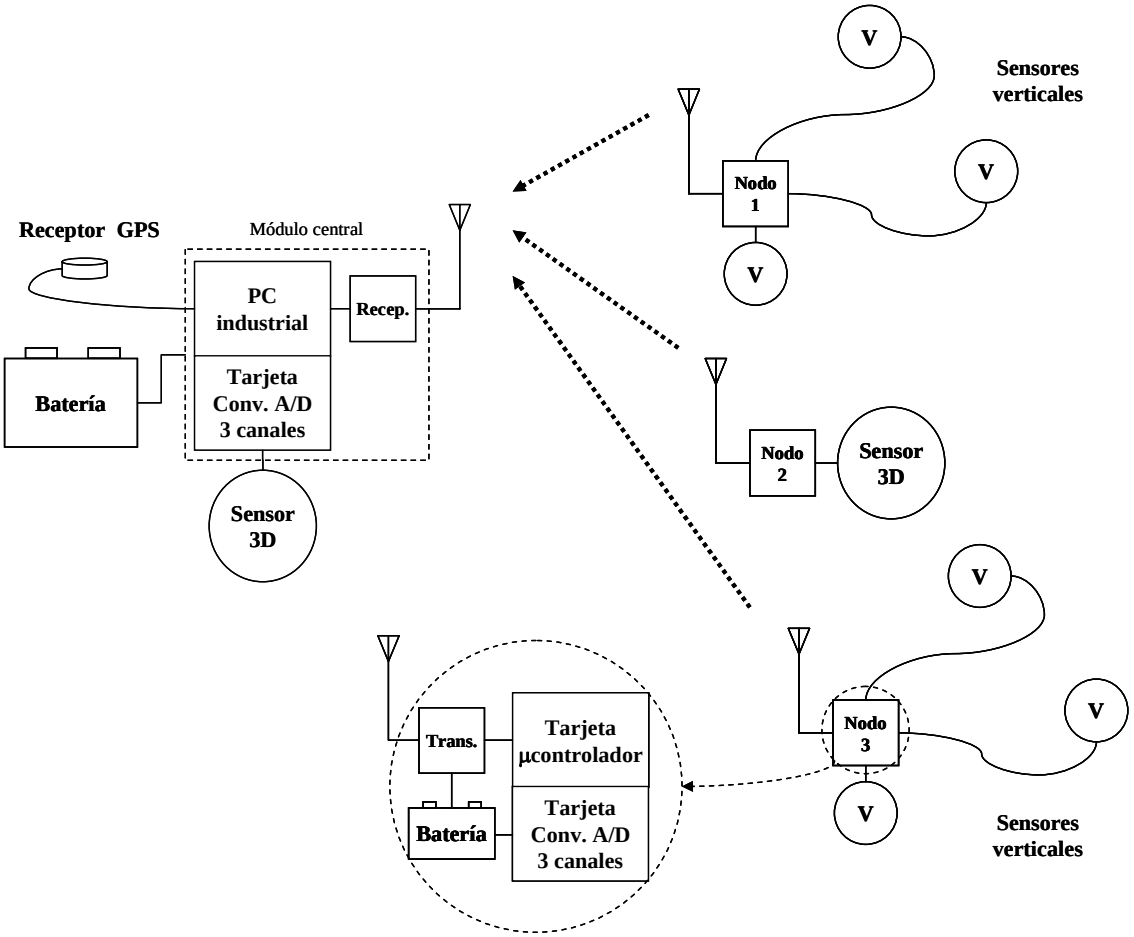


Figura 7.4.- *Diagrama de bloques de una posible implementación de antena sísmica sin cables. La estructura constaría de varios nodos de tres componentes equipados con transmisores de baja potencia, que podrían dotarse de sensores de una o tres componentes. El control de los conversores A/D y transmisores se realizaría mediante microcontroladores de bajo consumo. El módulo central, basado en un PC industrial de bajo consumo, gestionaría la recepción y grabación en disco de los datos de los diversos nodos y podría a su vez controlar otra tarjeta de conversión de tres canales. La alimentación se proporcionaría mediante varias baterías pequeñas, cuya capacidad debería calcularse en función del consumo de los nodos y módulo central.*

CAPÍTULO 8

CONCLUSIONES

Las antenas o *arrays* sísmicos constituyen un instrumento potente y resolutivo para el estudio de la estructura interior de la Tierra y de los procesos de fuente y propagación de la señal sísmica. En determinadas circunstancias, las antenas y las técnicas de procesado de datos asociadas a ellas, basadas en la búsqueda de la máxima coherencia de la señal, presentan grandes ventajas frente a las redes sísmicas convencionales. Entre ellas cabe destacar el aumento en la relación señal/ruido de los registros, la capacidad para suministrar información a partir de señales sin fases identificables, como las habitualmente presentes en entornos volcánicos (trémor, eventos de tipo LP,…) o las ventajas de tipo logístico derivadas del reducido espacio necesario para su despliegue.

La oferta comercial de antenas sísmicas de pequeña apertura es prácticamente inexistente. Es posible implementarlas a partir de instrumentos autónomos, pero a costa de un precio desproporcionado y muy poca operatividad. El diseño de dispositivos como los que se han presentado en este trabajo ha permitido obtener instrumentación de altas prestaciones a un precio asequible. El hecho de que los diseños sean de dominio público permitirá que cualquier institución aproveche el trabajo realizado para obtener instrumentación sísmica de calidad con un coste asumible.

El recorte en el precio no es la única ventaja de diseñar instrumentación propia. También es importante la facilidad para el mantenimiento que supone, ya que tanto la parte *hardware* como los programas son conocidos y fácilmente accesibles por el personal técnico. Para ello es importante la realización de una documentación técnica detallada, que es uno de los objetivos que se ha cubierto con este trabajo. Por otra parte, el hecho de contar con diseños propios hace posible la adaptación de los mismos a las necesidades específicas de aplicaciones concretas.

En esta tesis se ha presentado el diseño y desarrollo de tres dispositivos de antena, que constituyen instrumentos de altas prestaciones para el estudio de la sismicidad volcánica o tectónica de áreas activas. Los tres dispositivos comparten características comunes, como el control distribuido o la conversión analógico/digital de alta resolución, representada por el conversor AD7710 en el que se basan los correspondientes subsistemas de conversión. Sin embargo, las diferentes circunstancias que concurrían en cada uno de los casos dieron como resultado la adopción de distintas soluciones de diseño.

Así, en la antena del Gran Sasso fueron dos los factores más determinantes en el diseño. Por una parte, el emplazamiento subterráneo de los LNGS. Por otra, el carácter permanente de la antena y la infraestructura existente en los laboratorios. En la práctica, el primer factor se traducía en la imposibilidad de utilizar receptores GPS como base del sistema de sincronismo. El segundo factor permitía obviar las restricciones de consumo habitualmente impuestas en sistemas portátiles de antena, lo cual posibilitó el uso de PCs industriales como controladores de los distintos procesos.

Teniendo en cuenta estas circunstancias, se adoptó una estructura de control distribuido en tres niveles (estaciones, PCs nodales y servidores). Una de las mayores ventajas de la estructura utilizada es que es abierta, en el sentido de que el sistema es fácilmente ampliable mediante la incorporación de más estaciones conectadas a nuevas líneas serie controladas por los correspondientes PCs nodales. En la práctica las posibilidades de ampliación sólo están limitadas por el espacio físico disponible, que será ampliado en los próximos años con la construcción de nuevos túneles en los LNGS. No obstante, la capacidad de ampliación del dispositivo respecto a la estructura descrita no se restringe a los dos

primeros niveles, sino que también es posible añadir nuevos servidores y conectarlos al servidor original a través de Internet. De hecho, esta posibilidad ya ha sido implementada en la actualidad por los responsables de mantenimiento de la antena, que de esta forma pueden llevar a cabo el control del sistema y el análisis de los datos de forma remota.

La imposibilidad de utilizar receptores GPS se solventó mediante el diseño y desarrollo de un oscilador patrón, que a partir de una señal de precisión suministrada por los LNGS genera las señales necesarias para la sincronización de todos los procesos.

Desde el punto de vista científico, los LNGS constituyen un entorno privilegiado para la operación de un dispositivo de antena como el que se ha presentado, puesto que están situados en la proximidad de un sistema de fallas de actividad moderada. Esto permitirá realizar estudios *in situ* de los procesos físicos que originan las rupturas sísmicas en el área. Por otra parte, el emplazamiento subterráneo del dispositivo permite, mediante comparación de los registros con los de las estaciones de superficie, obtener información que no es posible conseguir en otros instrumentos similares. Esta información resulta especialmente interesante para estudios de los efectos de sitio, estructura de velocidad en el área y los procesos de *scattering* que tienen lugar en la región.

Las circunstancias presentes en la antena del Vesubio eran bien distintas. Se pretendía construir un dispositivo con un emplazamiento casi permanente en la falda del volcán, orientado a la monitorización continua de su actividad. Sin embargo, el dispositivo también debía ofrecer la posibilidad de ser trasladado fácilmente para su despliegue en otras áreas activas, principalmente del entorno próximo, como los volcanes Etna y Strómboli.

Así pues, en este caso el requerimiento más condicionante a la hora de plantear el diseño fue el de la portabilidad, que entre otros factores llevaba aparejada la necesidad de reducir el consumo total. Con el objeto de minimizarlo, se optó por un diseño basado en microcontroladores de la familia PIC. Al igual que en el Gran Sasso, se decidió digitalizar las señales de los sensores en cada punto de adquisición y utilizar transmisión digital hasta el módulo central, en el que un PC industrial de bajo consumo se ocuparía de la grabación local de los datos. El elevado número de canales que debía gestionar el sistema (48) no permitía al PC controlar directamente todo el proceso, por lo que se decidió diseñar y desarrollar dos subsistemas *hardware* (módulo Serie Paralelo y módulo de control de memoria) cuyas funciones serían preprocesar y buferizar los datos recibidos para liberar de carga al PC.

En esta ocasión no había impedimentos técnicos para utilizar receptores GPS, lo cual simplificó notablemente el diseño del subsistema de sincronismo respecto al del Gran Sasso. Para hacer más flexible la disposición de los distintos subsistemas, el de sincronismo (llamado aquí módulo de reloj) se planteó como un módulo separado del módulo central, con el que se comunicaría mediante transmisión serie asíncrona.

La antena del Vesubio permitirá profundizar en el conocimiento de la estructura volcánica, aportando a la red sísmica del OV las ventajas inherentes a los dispositivos de antena. Teniendo en cuenta que la red del OV no cuenta con más dispositivos de este tipo, la aportación del sistema presentado resulta muy interesante, puesto que posibilitará la detección de señales sin llegadas claras de fases o con bajas SNRs, que probablemente pasarían desapercibidas al resto de las estaciones. En su configuración final, consistente en dos antenas gemelas dispuestas en torno al cráter con transmisión telemétrica de los datos, el dispositivo permitiría llevar a cabo una localización de las fuentes de señal e incorporar esta información a los procedimientos de alerta sísmica del OV. Por otra parte, el carácter portátil de la antena permitirá la intervención rápida en casos de crisis sísmicas en áreas del entorno.

En el caso de las nuevas antenas portátiles del IAG, el objetivo era muy claro: la sustitución de los módulos portátiles de dieciséis bits, que se habían venido utilizando desde principios de los años noventa en numerosas campañas de recogida de datos.

Con este objetivo en mente, el primer paso fue llevar a cabo una revisión de las características principales de los antiguos módulos, orientada a conocer su funcionamiento y limitaciones para establecer las prestaciones y requerimientos de los nuevos sistemas. El diseño y desarrollo de éstos se estructuró en torno a dos premisas. En primer lugar, debía mantenerse la filosofía de operación de los antiguos módulos. En segundo, debían aprovecharse al máximo los resultados y componentes desarrollados para la antena del Vesubio, con el objeto de minimizar el tiempo de desarrollo y puesta en marcha. El resultado fueron los instrumentos que se han descrito en este trabajo, cuya estructura consiste en sistemas independientes pero con capacidad para operar sincronizadamente gracias a la base de tiempo común que proporcionan los receptores GPS. A semejanza de los antiguos módulos, el número de canales no es elevado (doce) y la transmisión de las señales es analógica, realizándose la digitalización en el módulo central. En cuanto a la segunda premisa, los elementos más importantes de los nuevos sistemas (tarjetas A/D, PLLs, tarjeta de PC) son los mismos que en la antena del Vesubio. Gracias a la simplificación funcional que implica la gestión de menos canales, no fue necesario utilizar algunos de los subsistemas diseñados para aquella (SerPar, módulo de memoria, módulo de reloj), por lo que desde el punto de vista estructural las nuevas antenas portátiles pueden considerarse como una versión reducida de la antena del Vesubio.

Desde su puesta en marcha los nuevos sistemas se han utilizado en varias campañas en distintos entornos volcánicos (Azores, Teide, Isla Decepción, Copahue, Colima, …), constituyéndose como un relevo eficaz de los antiguos módulos. Las mejoras en las características, tanto desde el punto de vista logístico (volcado de datos más rápido, mantenimiento más sencillo, menor consumo) como de prestaciones (mayor resolución, mayor número de canales, registro continuo) están suministrando datos de calidad con los que, en los próximos años, se contribuirá al conocimiento de las estructuras volcánicas, fenómenos de propagación de las ondas o régimen de actividad en las regiones de estudio.

Con objeto de calibrar las prestaciones reales de los dispositivos presentados, se han analizado los resultados obtenidos por diversos autores a partir de datos registrados con las antenas del Gran Sasso y con los nuevos sistemas portátiles del IAG.

En el caso del Gran Sasso, las principales conclusiones fueron que el dispositivo permite detectar llegadas de frentes de onda coherentes de forma rápida y fiable, incluso con SNRs muy bajas. El emplazamiento subterráneo de la antena fija el umbral de detección de eventos en torno a M = 1.0, lo cual supone una sensible reducción respecto al de la Red Sísmica Nacional Centralizada italiana. Los autores aplicaron distintos análisis a un conjunto de terremotos con distancias epicentrales de entre 20 y 140 km para calcular los parámetros de propagación. Posteriormente, localizaron los eventos usando los azimuts calculados para las ondas P y S, junto con las diferencias de tiempo S-P de los registros. Los resultados se compararon con los de la Red Sísmica Nacional italiana, obteniendo que para el caso de las ondas S las discrepancias entre las dos localizaciones nunca superaron los 10 km. Para las ondas P se obtuvieron errores mayores, lo cual probablemente se debe a que la pequeña apertura de la antena no permite una evaluación precisa de ondas que se propagan con velocidades aparentes altas. En conjunto estos resultados pueden calificarse como muy satisfactorios. No obstante, se prevé que mejorarían si se aumentara la densidad y apertura de la antena, lo cual podría conseguirse en los próximos años, cuando se finalice la construcción de nuevas galerías en los LNGS.

En el caso de las antenas portátiles del IAG, se contrastaron resultados obtenidos por diversos autores a partir de datos adquiridos en varios entornos volcánicos. La primera campaña en la que se emplearon los nuevos módulos fue la de 2003 en la Isla de São Miguel, en las Azores. En el contexto de esta tesis la importancia de este experimento residía en que era la primera prueba en campo de los equipos, en la que éstos debían demostrar su

funcionalidad y en la que a partir de la experiencia del personal de mantenimiento y del análisis de los datos se plantearían posibles cambios en la estructura y operación de los sistemas. Las modificaciones que se realizaron tras la campaña, concernientes principalmente a las cajas y conectores utilizados, al sistema de volcado de datos y a la estrategia de adquisición de datos del PC de control, dieron como resultado la versión definitiva de los módulos. Desde el punto de vista científico, lo más destacable fue la detección de un enjambre de eventos VT y, sobre todo, de una señal de tipo trémor cuyos parámetros de correlación y propagación llevan a pensar que podría tratarse de un precursor del enjambre.

En el Teide se desplegaron tres antenas, como respuesta al incremento de la actividad sísmica que se había detectado unos meses antes. El comportamiento de los nuevos sistemas fue muy satisfactorio, registrando algunos eventos de tipo LP y cientos de terremotos VT, cuya localización mediante las antenas resultó consistente con las realizadas a partir de datos de la red regional del IGN. Sin embargo, lo más destacable de esta campaña fue la aparición de un trémor volcánico, puesto que hasta entonces no se había detectado en la región este tipo de señales. El hecho de que los nuevos sistemas fueran capaces de detectarla, pese a su extremadamente baja amplitud, supuso una confirmación de sus prestaciones en condiciones reales de operación.

En Isla Decepción los sistemas se han estado utilizando desde la campaña 2003-2004 para el seguimiento de la actividad del volcán. La aplicación más ambiciosa se realizó en la campaña 2004-05, en la que entre otros equipos se desplegaron doce de los nuevos sistemas portátiles para realizar una tomografía de velocidad de la isla (campaña TOMODEC). Aunque los datos obtenidos se encuentran actualmente en proceso de análisis, se han realizado algunos estudios preliminares en los que se concluye que los registros resuelven satisfactoriamente variaciones de velocidad de las ondas sísmicas, relacionadas con las heterogeneidades de la estructura volcánica. Asimismo, los datos de TOMODEC se han utilizado para otros estudios, como calibración de técnicas de localización relativa de alta precisión (RelSE) o determinación de la estructura superficial de la isla a partir de registros de ruido sísmico.

En el volcán Copahue se desplegó uno de los antiguos módulos de dieciséis bits, que posteriormente se completó con dos de los nuevos sistemas. Durante el periodo que duró este experimento, el volcán, uno de los más activos de Argentina, se encontraba en periodo de reposo. La caracterización de la sismicidad de fondo durante estos periodos juega un papel fundamental en el desarrollo de sistemas de alerta temprana, ya que una reactivación del sistema volcánico vendría probablemente precedida por un cambio en la actividad sísmica. Teniendo en cuenta que el régimen de sismicidad en reposo varía de un volcán a otro, resulta vital conocer esta actividad de fondo para ser capaces de detectar posibles cambios en las propiedades de la fuente o medir la evolución de la dinámica del sistema. En el caso del Copahue se detectaron dos enjambres de eventos VT de baja energía y una señal de trémor volcánico, que tras el análisis de sus parámetros de propagación se interpretó como consecuencia de la actividad de dos campos geotermales de la región. Sin embargo, no se detectaron señales procedentes del volcán propiamente dicho, lo cual a raíz de este estudio puede considerarse una característica del periodo de reposo de este sistema. El aumento de la densidad de la antena utilizada en este experimento permitirá aumentar la resolución espacial de sus registros, mientras que la incorporación de una segunda antena hará posible realizar localizaciones conjuntas, tanto de las señales actualmente presentes en el área (en particular el trémor con origen geotermal) como de las hipotéticas señales procedentes del volcán que probablemente se producirían en caso de una reactivación del sistema.

La última aplicación de las nuevas antenas portátiles descrita en este trabajo ha sido la del Volcán de Fuego de Colima. Al igual que el Copahue, se trata de un volcán muy activo, pero a diferencia de éste se encontraba en erupción durante el periodo que duró la campaña. El Volcán de Colima se caracteriza por presentar un amplio abanico de tipos eruptivos. En el periodo del experimento la actividad era de tipo vulcaniano, con una media de entre dos y tres

explosiones al día, lo cual permitió recoger una gran cantidad de eventos de este tipo. Además se detectaron eventos de tipo LP y VT y señales de tipo trémor volcánico de baja energía, datos que en su conjunto están siendo analizados en la actualidad. En esta memoria se ha presentado un análisis preliminar de eventos de tipo explosivo. La aplicación de técnicas de procesado de *array* permitió caracterizar la evolución de sus parámetros de correlación y propagación e interpretar los registros asociados a este tipo de eventos como la sucesión de tres procesos bien diferenciados.

Como último capítulo en esta memoria se ha incluido un conjunto de mejoras aplicables a las tres antenas. Algunas de estas modificaciones han surgido a partir de la experiencia con los nuevos equipos, otras se centran en la sustitución de algunos componentes utilizados por otros de mejores características. En proyectos técnicos de cierta envergadura es inevitable que algunos de los componentes se queden obsoletos, puesto que en el tiempo que transcurre desde su elección hasta su implementación final se desarrollan modelos de prestaciones superiores. En el contexto de los sistemas de adquisición de datos para instrumentación sísmica tiene especial importancia la evolución de los conversores A/D de alta resolución, por lo que dentro de este último capítulo se le ha prestado una atención preferente.

Como conclusión final, puede decirse que los tres sistemas diseñados y desarrollados en este trabajo constituyen un conjunto instrumental de altas prestaciones para el estudio de distintas áreas sísmicas tectónicas y volcánicas durante los próximos años. Gracias a su diseño modular, constituyen además una base estructural adecuada para la incorporación de distintas mejoras técnicas, lo que les augura un futuro largo y prolífico en el campo de la adquisición de datos sísmicos de alta resolución.

BIBLIOGRAFÍA

BIBLIOGRAFÍA

1. Publicaciones y presentaciones en congresos

ABLAY, G. J., ERNST, G. G. J., MARTÍ, J., SPARKS, R. S. J. (1995). *The – 2 ka subplinian eruption of Montaña Blanca, Tenerife.* Bull. Volcanol., 57 (5), pp. 337-355.

ABRIL, M., IBÁÑEZ, J. M. (2000). *Uso de antenas sísmicas en ambientes volcánicos*. En *Curso Internacional de Volcanología y Geofísica Volcánica, Edición 2000,* pp. 283-295. Editado por M. Astiz y A. García. Servicio de Publicaciones del Cabildo de Lanzarote.

ABRIL, M., MARTINI, M., HAVSKOV, J., ALGUACIL, G., MORENO, J. (2003). *SEISAD18, 18 bit digitizer. Development and user manual*. Instituto Andaluz de Geofísica, 21 pp. Incluido en el CD adjunto[75].

ABRIL, M., ALGUACIL, G., IBÁÑEZ, J. M. (2004). *Nuevas antenas sísmicas portátiles*. 4ª Asamblea Hispano Portuguesa de Geodesia y Geofísica. Figueira da Foz (Portugal). Póster. Incluido en el CD adjunto.

AKI, K. (1957). *Space and time spectra of stationary stochastic waves, with special reference to microtremors.* Bull. Earthquake Res. Inst. Tokyo Univ., 35, pp. 415-456.

AKI, K., RICHARDS, P. G. (1980). *Quantitative Seismology: Theory and Methods.* W. H. Freeman and Company, San Francisco.

ALGUACIL, G. (1986). *Los instrumentos de una Red Sísmica Local Telemétrica para Microterremotos. La Red Sísmica de la Universidad de Granada.* Tesis Doctoral. Universidad de Granada.

ALGUACIL, G., ALMENDROS, J., DEL PEZZO, E., GARCÍA, A., IBÁÑEZ, J. M., LA ROCCA, M., MORALES, J., ORTIZ, R. (1999). *Observations of volcanic earthquakes and tremor at Deception island – Antarctica*. Annali di Geofisica, 42, pp. 417-436.

ALGUACIL, G., HAVSKOV, J., ABRIL, M., MARTINI, M., MORENO, J., MARTOS, A. (2002). *Equipos de dominio público para adquisición de datos sísmicos.* III Asamblea Hispano-Portuguesa de Geodesia y Geofísica. Universidad de Valencia-Instituto Geográfico Nacional. Presentación oral. Incluida en el CD adjunto.

ALGUACIL, G. (2003). *Etapas del desarrollo instrumental en la época universitaria del Observatorio de Cartuja*. En *Historia del Observatorio de Cartuja, 1902-2002. Nuevas investigaciones.* Editado por M. Espinar, J. A. Esquivel y J. A. Peña. Ayuntamiento de Granada. [CD]. Disponible en www.ugr.es/~iag/ins/incd.html

ALMENDROS, J.; IBÁÑEZ, J. M., ALGUACIL, G., DEL PEZZO, E., ORTIZ, R. (1997). *Array tracking of the volcanic tremor source at Deception Island, Antarctica.* Geophys. Res. Lett., 24, pp. 3069-3072.

[75] Para facilitar la consulta, las presentaciones en congresos y algunos otros documentos no publicados o de difícil acceso se incluyen en el CD adjunto a esta memoria.

ALMENDROS, J. (1999). *Análisis de señales sismo-volcánica mediante técnicas de array.* Tesis doctoral. Universidad de Granada. 301 pp.

ALMENDROS, J., IBÁÑEZ, J. M., ALGUACIL, G., DEL PEZZO, E. (1999). *Array analysis using circular-wave-front geometry: an application to locate the nearby seismo-volcanic source.* Geophys. J. Int., 136, pp. 159-170.

ALMENDROS, J., IBÁÑEZ, J. M., ALGUACIL, G., MORALES, J., DEL PEZZO, E., LA ROCCA, M., ORTIZ, R., ARAÑA, V., BLANCO, M. J. (2000). *A double seismic antenna experiment at Teide Volcano: existence of local seismicity and lack of evidences of volcanic tremor.* J. Volcanol. Geotherm. Res., 103 (1-4), pp. 439-462.

ALMENDROS, J., CARMONA, E., IBÁÑEZ, J. M. (2004a). *Precise determination of the relative wave propagation parameters of similar events using a small-aperture seismic array*. J. Geophys. Res., 109, B11308, doi: 10.1029/2003JB002930.

ALMENDROS, J., LUZÓN, F., POSADAS, A. (2004b). *Microtremor analyses at Teide volcano (Canary Islands, Spain): assessment of natural frequencies of vibration using time-dependent horizontal-to-vertical spectral ratios*. Pure Appl. Geophys., 161 (7), pp. 1579-1596.

ALMENDROS, J., ZANDOMENEGHI, D., CARMONA, E., IBÁÑEZ, J. M., BLANCO, M. J., ABELLA, R. (2005a). *Array analyses of seismic activity during May 2004 at Teide Volcano, Canary Islands, Spain.* 2005 General Assembly of the European Geophysical Union (EGU), Viena (Austria). Póster. Incluido en el CD adjunto.

ALMENDROS, J., ZANDOMENEGHI, D., SACCOROTTI, G., BARCLAY, A., IBÁÑEZ, J. M. (2005b). *Seismic Tomography of Central Sâo Miguel, Azores, Portugal.* 2005 Fall Meeting of the American Geophysical Union (AGU), San Francisco (EEUU). Póster. Incluido en el CD adjunto.

ALMENDROS, J., ABELLA, R. (2006). *Estimación precisa del vector lentitud aparente relativo mediante antenas sísmicas: un test del método RelSE con fuente sísmicas activas.* VII Simposio Español de Estudios Polares, Granada (España). Póster. Incluido en el CD adjunto.

ALMENDROS, J., LUZÓN, F. (2006). *Estructura superficial de velocidad del volcán Isla Decepción (Antártida) mediante la autocorrelación espacial de ruido sísmico.* VII Simposio Español de Estudios Polares, Granada (España). Póster. Incluido en el CD adjunto.

ALMENDROS, J., LUZÓN, F., IBÁÑEZ, J. M. (2006). *Shallow structure of Deception Island volcano, Antarctica, using the spatial autocorrelation method on a dense set of seismic arrays.* 2006 General Assembly of the European Geophysical Union (EGU), Viena (Austria). Póster. Incluido en el CD adjunto.

ALMENDROS, J., IBÁÑEZ, J. M., CARMONA, E., ZANDOMENEGHI, D. (2007). *Array analyses of volcanic earthquakes and tremor recorded at Las Cañadas caldera (Tenerife Island, Spain) during the 2004 seismic activation of Teide volcano.* J. Volcanol. Geoth. Res., 160, pp. 285-299.

AL-SHUKRI, H., PAVLIS, G. L., VERNON III F. L. (1995). *Site Effect Observations from Broadband Arrays*. Bull. Seism. Soc. Am. 85, pp. 1758-1769.

AMORUSO, A., CRESCENTINI, L., DE LUCA, G., SCARPA, R., ABRIL, M., CIRELLA, A. (1997). *Underground earth strain and seismic radiation measurements with a laser interferometer and a dense small-aperture seismic array*. Annali di Geofisica, 40, pp. 995-1005.

AMORUSO, A., CRESCENTINI, L., SCARPA, R. (1998). *Inversion of source parameters from near- and far-field observations: An application to the 1915 Fucino earthquake, central Apennines, Italy.* J. Geophys. Res., 103, pp. 29989-29999.

AMORUSO, A., CRESCENTINI, L., MORELLI, A., SCARPA, R. (2002). *Slow rupture of an aseismic fault in a seismogenic region of central Italy*. Geophys. Res. Lett., 29, p. 2219, doi: 10.1029/2002GL016027.

ANALOG DEVICES (1991): *LC^{2}MOS Signal Conditioning ADC AD7710*. Hojas de características del AD7710 de *Analog Devices*. Disponible en www.analog.com.

ANDERSON, D. L. (1979). *Earth Tides and Ocean Tides*. Science, 204, pp. 1074-1075, doi: 10.1126/science.204.4397.1074-a.

ANDRONICO, D., CALDERONI, G., CIONI, R., SBRANA, A., SULPIZIO R., SANTACROCE, R. (1995). *Geological map of Somma-Vesuvius Volcano*. Per. Mineral, 64-8, pp. 77-78.

AOI, S., IWATA, T., FUJIWARA, H., IRIKURA, K. (1997). *Boundary Shape Waveform Inversion for Two-Dimensional Basin Structure Using Three-Component Array Data of Plane Incident Wave with an Arbitrary Azimuth*. Bull. Seism. Soc. Am., 87, pp. 222-233.

ARAÑA, V., CAMACHO, A. G., GARCÍA, A., MONTESINOS, F. G., BLANCO, I., VIEIRA, R., FELPETO, A. (2000). *Internal structure of Tenerife (Canary Islands) based on gravity, aeromagnetic and volcanological data*. J. Volcan. Geotherm. Res., 103 (1-4), pp. 43-64.

ARRIGHI, S., PRINCIPE, C., ROSI, M. (2001). *Violent Strombolian Eruptions at Vesuvius During post-1631 Activity*. Bull. Volcanol., 63, pp. 126-150.

ASTEN, M. W., HENSTRIDGE, J. D. (1984). *Array estimators and the use of microseism for reconnaissance of sedimentary basins*. Geophysics, 49, pp. 1828-1837.

AUSTERLITZ, H. (1991): *Data Acquisition Techniques Using Personal Computers*. Academic Press, San Diego, CA.

BATLLÓ, J. (2003). *Los sismógrafos del Observatorio de Cartuja.* En *Historia del Observatorio de Cartuja, 1902-2002. Nuevas investigaciones.* Editado por M. Espinar, J. A. Esquivel y J. A. Peña. Ayuntamiento de Granada. [CD]. Disponible en www.ugr.es/~iag/ins/incd.html

BENIOT, J. P., MCNUTT, R. (1996). *Global volcanic earthquake swarm database and preliminary analysis of volcanic earthquake swarm duration*. Annali di Geofisica, 39, pp. 221-230.

BENÍTEZ, C., RAMÍREZ, J., SEGURA J. C., IBÁÑEZ J. M., ALMENDROS J., GARCÍA-YEGUAS A., CORTÉS, G. (2006). *Continuous HMM-based seismic event classification at Deception Island, Antarctica*. IEEE Transactions On Geoscience And Remote Sensing, Vol. 44, Nº 12, pp. 40-49.

BIANCO, F., CUSANO, P., PETROSINO, S., CASTELLANO, M., BUONOCUNTO, C., CAPELLO, M., DEL PEZZO, E. (2005). *Small-aperture array for seismic monitoring of Mt. Vesuvius*. Seism. Res. Lett., 76, 3, pp. 345-355.

BLANCO, I. (1997). *Análisis e interpretación de las anomalías magnéticas de tres calderas volcánicas: Decepción (Shetland del Sur, Antártida), Furnas (San Miguel, Azores) y Las Cañadas del Teide (Tenerife, Canarias).* Tesis doctoral. Universidad Complutense de Madrid.

CAMPOS, M. F., CASTAÑEDA, R., CONTRERAS, A. C. (1998). *Implementación de un sistema de desarrollo utilizando los microcontroladores PIC*. Centro Universitario de Ciencias Exactas e Ingeniería, Universidad de Guadalajara. Disponible en: www.abcdatos.com/tutoriales/tutorial/l9787.html

CANALES, J. P., DAÑOBEITIA, J. J., WATTS, A. B. (2000). *Wide-angle seismic constraints on the internal structure of Tenerife, Canary Islands*. J. Volcan. Geotherm. Res., 103 (1-4), pp. 65-81.

CANAS, J. A., UGALDE, A., PUJADES, L. G., CARRACEDO, J. C., SOLER, V., BLANCO, M. J. (1998). *Intrinsic and scattering seismic wave attenuation in the Canary Islands*. J. Geophys. Res., 103 (B7), pp. 15037-15050.

CAPON, J. (1969). *High-resolution frequency-wavenumber spectrum analysis*. Proc. IEEE, 57, pp. 1408-1418.

CARMONA, E., ESQUIVEL, J. A., ESTÉVEZ, G., IBÁÑEZ, J.M. (2002). *Aproximación a la caracterización de las series sísmicas utilizando métodos no paramétricos*. Simpósium 'Cien años de sismología en Granada'. [CD]. Universidad de Granada.

CASELLI, A. T., AGUSTO M. R., FAZIO, A. (2005). *Cambios térmicos y geoquímicos del lago cratérico del volcán Copahue (Neuquén): posibles variaciones cíclicas del sistema volcánico*. XVI Congreso Geológico Argentino, La Plata. Tomo I, pp. 751-756.

CHOUET, B. A. (1996). *Long-period volcano seismicity: its source and use in eruption forecasting*, Nature, 380, pp. 309-316.

CHOUET, B. A., SACCOROTTI, G., MARTINI, M., DAWSON, P., DE LUCA, G., MILANA, G., SCARPA, R. (1997). *Source and path effects in the wave fields of tremor and explosions at Stromboli Volcano, Italy.* Journ. Geophys. Res., 102, pp. 15129-15150.

CHOUET, B. A., DE LUCA, G., MILANA, G., DAWSON, P., MARTINI, M., SCARPA, R. (1998). *Shallow Velocity Structure of Stromboli Volcano, Italy, derived from Small-Aperture Array Measurementes of Strombolian Tremor.* Bull. Seism. Soc. Am., 88, pp. 653-666.

CHOUET, B. (2003). *Volcano seismology*. Pure Appl. Geophys., 160 (3-4), pp. 739-788.

CIONI, R., SANTACROCE, R., SBRANA, A. (1999). *Pyroclastic Deposits as a Guide for Reconstructing the Multi-Stage Evolution of the Somma-Vesuvius Caldera*. Bull. Volcanol., 60, pp. 207-222.

CIRRUS LOGIC (2005). *CS5012A, CS5014, CS5016: 16-, 14-, & 12-bit Self-calibrating A/D Converters*. Hojas de características de los conversores CS5012A, CS5014 y CS5016 de *Cirrus Logic*. Disponible en: www.cirrus.com/en/pubs/proDatasheet/CS5012A-14-16_F9.pdf.

CRESCENTINI, L., AMORUSO, A., FIOCCO, G., VISCONTI, G. (1997). *Installation of a high sensitivity laser strainmeter in a tunnel in central Italy*. Review of Scientific Instruments, 68, pp. 3206-3210.

CRESCENTINI, L., AMORUSO, A., SCARPA, R. (1999). *Constraints on slow earthquakes dynamics from a swarm in Central Italy.* Science, 286, pp. 2132-2134.

DAINTY, A. M., TOKSOZ, M. N. (1990). *Array analysis of seismic scattering*. Bull. Seism. Soc. Am., 80, pp. 2242-2260.

D'AURIA, L., GIUDICEPIETRO, F., MARTINI, M., PELUSO, R. (2006). *Seismological insight into kinematics of the 5 April 2003 vulcanian explosion at Stromboli volcano (Southern Italy)*. Gephys. Res.Letters, 33, L08308, doi: 1029/2006GL026018.

DAWSON, P. B., DA SILVA, A. R., IYER, H. M., EVANS, J. R. (1985). *Seismic study of the Agua de Pau geothermal prospect, Sâo Miguel, Azores.* Geoth. Resour. Council Transac., Vol. 9, parte II, pp. 401-406.

DE LUCA, G., DEL PEZZO, E., DI LUCCIO, F., MARGHERITI, I., MILANA, G., SCARPA, R. (1998). *Site response study in Abruzzo (central Italy): underground array versus surface stations*. J. Seismol., 2, pp. 223-236.

DE LUCA, G., SCARPA, R., FILIPPI, L., GORINI, A., MARCUCCI, S., MARSAN, P., MILANA, G., ZAMBONELLI, E. (2000). *A detailed analysis of two seismic sequences in Abruzzo, Central Appenines, Italy.* J. Seismol., 4, pp. 1-21.

DEL PEZZO, E., LA ROCCA, M., IBÁÑEZ, J.M. (1997). *Observations of high-frequency scattered waves using dense arrays at Teide volcano.* Bull. Seism. Soc. Am., 87, pp. 1637-1647.

DEL PEZZO, E., LA ROCCA, M., PETROSINO, S., GROZEA, B., MARITATO, L., SACCOROTTI, G., SIMINI, M., IBÁÑEZ, J. M., ALGUACIL, G., CARMONA, E., ABRIL, M., ALMENDROS, J., ORTIZ, R., GARCÍA, A., PINGÜE, F. y ESPOSITO, T. (1998). *Twin digital short period seismic array experiment at Stromboli Volcano. Open file report* número 1/1998 del *Osservatorio Vesuviano*. Disponible en www.ov.ingv.it.

DEL PEZZO, E., BIANCO, F., CASTELLANO, M., PETROSINO, S., PINGUE, F., CAPELLO, M., ESPOSITO, T., AUGUSTI, V., SACCOROTTI, G., LA ROCCA, M., MARESCA, R., GALLUZO, D., CIRILLO, A., GROZEA. B., IBÁÑEZ, J. M., CARMONA, E., ALGUACIL, G. (1999). *A seismic array on Mt. Vesuvius. Open file report* número 1/1999 del *Osservatorio Vesuviano*. Disponible en www.ov.ingv.it.

DEL PEZZO, E., CASTELLANO, M., CAPELLO, M., GIUDICEPIETRO, F., LA ROCCA, M., MARTINI, M., PETROSINO, S., SACCOROTTI, G., IBÁÑEZ, J. M., ABRIL, M., ALMENDROS, J., CARMONA, E., MARTÍNEZ-ARÉVALO, C., VÍLCHEZ, J., PRIVITERA, E., ALPARONE, S., DI GRAZIA, G., GRESTA, S. (2000). *A Double Seismic Array Experiment on Mt. Etna. Open file report* número 2/2000 del *Osservatorio Vesuviano*. Disponible en www.ov.ingv.it.

DEL PEZZO, E., BIANCO, F., SACCOROTTI, G. (2004). *Seismic source dynamics at Vesuvius volcano, Italy*. J. Vol. Geo. Res., 133, pp. 23-39.

DEL PEZZO, E., BIANCO, F., SACCOROTTI, G., LA ROCCA, M., GALLUZZO, D., NISII, V., PETROSINO, S., CUSANO, P., ZACCARELLI, L., DAMIANO, N., TRAMELLI, A., DE ASTIS, G., BRETÓN, M., OROZCO-ROJAS, J., NAVARRO, C., IBÁÑEZ, J. M., OCAÑA, E., GARCÍA, A. (2007). *Seismic survey of Colima Volcano (México), November 2005-May 2006. Open file report* número 7/2007 del *Osservatorio Vesuviano*. Disponible en www.ov.ingv.it.

DELPINO, D., BERMUDEZ, A. (1995). *Eruptions of pyroclastic sulphur at crater lake of Copahue volcano, Argentina*. International Union of Geodesy and Geophysics, General Assembly, Nº 21, Abstracts, p. B410.

DOMÍNGUEZ, T., ZOBIN, V. M., REYES-DÁVILA, G. A. (2001). *The fracturing in volcanic edifice before an eruption: the June-July 1998 high-frequency earthquake swarm at Volcán de Colima, México*. J. Volcanol. Geotherm. Res., 105, pp. 65-75.

ESPINAR, M. (2003). *Fundación del Observatorio de Cartuja. Primeros años de funcionamiento (1902-1906)*. En *Historia del Observatorio de Cartuja, 1902-2002. Nuevas investigaciones.* Editado por M. Espinar, J. A. Esquivel y J. A. Peña. Ayuntamiento de Granada. [CD]. Disponible en www.ugr.es/~iag/ins/incd.html

ESPOSITO, A.M., GIUDICEPIETRO, F., SCARPETTA, S., D'AURIA, L., MARINARO, M., MARTINI, M. (2006). *Automatic discrimination among landslide, explosion-quake and microtremor seismic signals at Stromboli volcano using Neural Networks*. Bull. Seismol. Soc. Am. (BSSA), 96 (N. 4A), pp. 1230–1240, doi: 10.1785/0120050097.

EVERNDEN, J. F. (1976). *Study of seismological evasion Part III: evaluation of evasion posibilities using codas of large earthquakes*. Bull. Seism. Soc. Am., 66, pp. 549-592.

FERNÁNDEZ, J., YU, T. T., RODRÍGUEZ VELASCO, G., GONZÁLEZ MATESANZ, F. J., ROMERO, R., RODRÍGUEZ, G., QUIRÓS, R., DALDA, A., APARICIO, A., BLANCO, M. J. (2003). *New geodetic monitoring system in the volcanic island of Tenerife, Canaries, Spain;*

combination of InSAR and GPS techniques. J. Volcan. Geotherm. Res., 124 (3-4), pp. 241-253.

FERNÁNDEZ, F. (2007). *Sismicidad, reología y estructura térmica de la corteza en el Arco de Gibraltar*. Tesis doctoral. Universidad de Granada.

FERRAZZINI, V., AKI K., CHOUET B. A. (1991). *Characteristics of seismic waves composing Hawaiian volcanic tremor and gas-piston eventes observed by a near-source array*. J. Geophys. Res., 96, pp. 6199-6209.

FERRAZZINI, V., AKI, K. (1992). *Preliminary results from a field experiment on volcanic events at Kilauea using an array of digital seismographs.* En *Volcanic Seismology,* editado por K.Aki, P. Gasparini y R. Scarpa. Springer-Verlag, Berlin, pp. 168-189.

FISCHIONE, C., TRONCA, F., SACCOROTTI, G., SCARPA, R. (2006). *Recent seismicity in Central Italy as observed by the Gran Sasso UNDERground SEISmic array*. 2006 General Assembly of the European Geophysical Union (EGU), Viena (Austria). Póster. Incluido en el CD adjunto.

FRANKEL, A., HOUGH, S., FRIBERG, P., BUSBY, R. (1991). *Observations of Loma Prieta aftershocks from a dense array in Sunnyvale, California*. Bull. Seism. Soc. Am., 81, pp. 1900-1922.

FRANKEL, A. (1994). *Dense Array Recordings in the San Bernardino Valley of Landers-Big Bear Aftershocks: Basin Surface Waves, Moho Reflections, and Three-Dimensional Simulations*. Bull. Seism. Soc. Am. 84, pp. 613-624.

GANDINO, A., GUIDI, M., MERLO, C., METE, L., ROSSI, R., ZAN, L. (1985). *Preliminary model of the Ribeira grande geothermal field (Azores Island).* Geothermics,14, pp. 91-105.

GARMIN CORPORATION (2000). *GPS 35 LP smart antenna technical specification.* Manual de usuario del receptor GPS Garmin 35. Disponible en www.garmin.com

GOLDIE, J. (1992). *Failsafe Biasing of Differential Buses*. Nota de Aplicación 847 de *National Semiconductor*. Disponible en www.national.com/an/AN/AN-847.pdf

GOLDIE, J. (1996). *Ten Ways to Bulletproof RS-485 Interfaces*. Nota de Aplicación 1057 de *National Semiconductor*. Disponible en www.national.com/an/AN/AN-1057.pdf

GOLDSTEIN, P., ARCHULETA, R. J. (1987). *Array analysis of seismic signals*. Geophys. Res. Lett., 14, pp. 13-16.

GOLDSTEIN, P., ARCHULETA, R. J. (1991). *Deterministic frequency-wavenumber methods and direct measurements of rupture propagation during earthquakes using a dense array: theory and methods.* J. Geophys. Res., 96, pp. 6173-6185.

GOLDSTEIN, P., CHOUET, B. A. (1994). *Array measurements and modeling of sources of shallow volcanic tremor at Kilauea Volcano, Hawaii*. Journ. Geophys. Res., 99, pp. 2637-2652.

GOOSENS, P. (2002). *Controlador de CompactFlash para Bus IDE*. Revista Elektor, 265, pp. 68-75, junio 2002.

GREEN, P., FROSCH, R., ROMNEY, C. (1965). *Principles of an experimental Large Aperture Seismic Array (LASA).* Proc. IEEE, 53, pp. 1821-1833.

GUPTA, I. N., LYNNES, C. S., MCELFRESH, T. W., WAGNER, R. A. (1990). *F-K analysis of NORESS Array and single-station data to identify sources of near-receiver and near-source scattering.* Bull. Seism. Soc. Am. 80, pp. 2227-2241.

HAVSKOV, J., OTTEMÖLLER, L. (1999). *SEISAN earthquake analysis software.* Seism. Res. Lett., 70, pp. 532-534.

HAVSKOV, J., ALGUACIL, G. (2004). *Instrumentation in Earthquake Seismology.* Springer.

HERNÁNDEZ, P. A., PÉREZ, N. M., SALAZAR, J., SATO, M., NOTSU, K., WAKITA, H. (2000). *Soil gas CO_2, CH_4 and H_2 distribution in and around Las Cañadas caldera, Tenerife, Canary Islands, Spain*. J. Volcan. Geotherm. Res., 103 (1-4), 425-438.

HERNÁNDEZ, P. A., PÉREZ, N. M., SALAZAR, J., FERRELL, R., ÁLVAREZ, C. (2004). *Soil volatile mercury, boron and ammonium distribution at Las Cañadas Caldera, Tenerife, Canary Islands, Spain*. Appl. Geochem., 19 (6), pp. 819-834.

IBÁÑEZ, J. M., MORALES, J., ALGUACIL, G., ALMENDROS, J., ORTIZ, R., DEL PEZZO, E. (1997). *Intermediate-focus earthquakes under South Shetland Islands (Antarctica).* Geophys. Res. Lett., 24, pp. 531-534.

IBÁÑEZ, J. M., CARMONA, E. (2000). *Sismicidad volcánica.* En *Curso Internacional de Volcanología y Geofísica Volcánica, Edición 2000,* pp. 269-282. Editado por M. Astiz y A. García. Cabildo de Lanzarote.

IBÁÑEZ, J. M., DEL PEZZO, E., ALMENDROS, J., LA ROCCA, M., ALGUACIL, G., ORTIZ, R., GARCÍA, A. (2000). *Seismovolcanic signals at Deception Island volcano, Antarctica: wave field analysis and source modeling.* Journ. Geophys. Res., 105, B6, pp. 13905-13931.

IBÁÑEZ, J. M., ALMENDROS, J., CARMONA, E., MARTÍNEZ-ARÉVALO, C., ABRIL, M. (2003a). *The recent seismo-volcanic activity at Deception Island volcano*. Deep-Sea Research, Part II. En el volumen especial *ERUPT: Ecosystem studies of an enclosed bay within Deception Island, Antarctica*. Editado por K. Smith., 50, pp. 1611-1629.

IBÁÑEZ, J. M., CARMONA, E., ALMENDROS, J., SACCOROTTI, G., DEL PEZZO, E., ABRIL, M., ORTIZ, R. (2003b). *The 1998-1999 seismic series at Deception Island volcano, Antarctica*. Journ. Vol. Geo. Res., 128, pp. 65-88.

IBÁÑEZ, J. M., DEL PEZZO, E., BENGOA, C., CASELLI, A. T., BADI, G. A., ALMENDROS, J. (2007). *Volcanic tremor and local earthquakes at Copahue volcano, Southern Andes, Argentina*. Journ. Vol. Geo. Res. (en prensa).

IIDA, M., MIYATAKE, T., SHIMAZAKI, K. (1990). *Relationship between strong-motion array parameters and the accuracy of source inversion, and physical waves.* Bull. Seism. Soc. Am., 80, pp. 1533-1552.

JONSSON, S., ALVES, M. M., SIGMUNDSSON, F. (1999). *Low rates of deformation of the Furnas and Fogo Volcanoes, Sâo Miguel, Azores, observed with the Global Positioning System, 1993-1997*. J. Volcanol. Geotherm. Res., 92, pp. 83-94.

JULIAN, B., DAVIES, D., SHEPPARD, R. (1972). *PKJKP*. Nature, 235, pp. 317-318.

JURKEVICS, A. (1988). *Polarization analysis of three-component array data*. Bull. Seism. Soc. Am., 78, pp. 1725-1743.

KAGAWA, T., VIDAL, F., ALGUACIL, G., IBÁÑEZ, J. M., MORALES, J., PEÑA, J.A., MARTÍN, J. B., ALMENDROS, J., NAVARRO, M., SEO, K., HORIKE, M., SAMANO, T., IWATA, T., TSURUGI, M., KURITA, K., HATAYAMA, K. (1996). *Microtremor array observation in the Granada Basin, Southern Spain*. En *Homenaje en honor al Profesor Fernando de Miguel Martínez*, pp. 287-304. Servicio de Publicaciones de la Universidad de Granada.

KEDROV, O. K., OVTCHINNIKOV, V. M. (1990). *An online analysis system for three-component seismic data: method and preliminary results*. Bull. Seism. Soc. Am., 80, pp. 2053-2071.

KLIPSCH, P. W. (1936). *Some aspects of multiple recording in seismic prospecting*. Geophysics, 1, pp. 365-377.

LA ROCCA, M., MCCAUSLAND, W., GALLUZZO, D., MALONE, S., SACCOROTTI, G., DEL PEZZO, E. (2005). *Array measurements of deep tremor signals in the Cascadia subduction zone*. Geophys. Res. Lett., 32, doi: 10.1029/2005GL023974.

LACOSS, R. T., KELLY, E. J., TOKSOZ, M. N. (1969). *Estimation of seismic noise structure using arrays*. Geophysics, 34, pp. 21-38.

LEE, W.H. Y STEWART, S.W. (1981). *Principles and applications of microearthquake networks*. Academic Press, Nueva York.

LIN, C.-H., ROECKER S. W. (1996). *P-Wave Backazimuth Anomalies Observed by a Small-Aperture Seismic Array at Pinyon Flat, Southern California: Implications for Structure and Source Location*. Bull. Seism. Soc. Am., 86, pp. 470-476.

LINARES, E., OSTERA, H. A., MAS, L. (1999). Cronología Potasio-Argón del complejo efusivo Copahue-Caviahue, provincia de Neuquén. Revista de la Asociación Geológica Argentina, Vol. 54, Nº 3, pp. 240-247.

LIRER L., PETROSINO P., ALBERICO I., POSTIGLIONE I. (2001). *Long term volcanic hazard forecast based on Somma-Vesuvio past eruptive activity*. Bull. Volcanol., 68, pp. 145-163.

MAMANI, M. J., BORZOTTA, E., VENENCIA, J. E., MAIDANA, A., MOYANO, C. E., CASTIGLIONE, B. (2000). *Electric structure of the Copahue Volcano (Neuquén Province, Argentina)*

from magnetotelluric soundings: 1D and 2D modellings. Journ. South Am. Earth Sciences, 13, pp. 147-156.

MARTÍNEZ-AREVALO, C., BIANCO, F., IBÁÑEZ, J. M., DEL PEZZO, E. (2003). *Shallow seismic attenuation and shear waves splitting in the short period range of Deception Island volcano (Antarctica)*. Journ. Vol. Geo. Res., 128, pp. 89-113.

MARTÍNEZ SOLARES, J. M. (1995). *Métodos de localización epicentral mediante el procesamiento de señales de dispositivos sísmicos*. En *Redes sísmicas regionales*, pp. 113-144 (Monografía 11). Editado por J. Mezcua. Instituto Geográfico Nacional.

MARTINI, M., AUGER, E., BORRIELLO, G., BUONOCUNTO, C., CAPELLO, M., CAPUTO, A., CASTELLANO, M., CUSANO, P., D'AURIA, L., DE CESARE, W., ESPOSITO, A., GIUDICEPIETRO, F., LO BASCIO, D., ORAZI, M., PELUSO, R., PETROSINO, S., RICCIOLINO, P., SCARPATO, G., TALARICO, G. (2005). *Sismología*. En *Rendiconto di sorveglianza 2004*. Disponible en www.ov.ingv.it

MAXIM INTEGRATED PRODUCTS (1996). *Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers*. Hojas de características de los transceptores MAX48x/49x de *Maxim Integrated Products*. Disponible en www.maxim-ic.com

MCDERMOTT, E. E. (1937). *Use of multiple seismometers*. Petroleum Engineer, 8, pp. 135-136.

METAXIAN, J. P., LESAGE, P., DOREL, J. (1997). *Permanent tremor of Masaya volcano, Nicaragua: wave field analysis and source location.* Journ. Geophys. Res., 102, pp. 22529-22545.

MICROCHIP TECHNOLOGY INC. (1997). *8-Bit CMOS Microcontrollers with A/D Converter*. Hojas de características de los microcontroladores de la familia PIC16C7X de *Microchip Technology*. Disponible en www.microchip.com

MICROCHIP TECHNOLOGY INC. (1998). *18-pin Flash/EEPROM 8-Bit Microcontrollers*. Hojas de características de los microcontroladores de la familia PIC16F8X de *Microchip Technology*. Disponible en www.microchip.com

MIKKELTVEIT, S. (1985). *A new regional array in Norway: design, work and results from analysis of data from a provisional installation.* En *The Vela Program, a Twenty-Five Review of Basic Research*, editado por U. A. Kerr. Defence Advanced Research Project Agency, pp. 546-553.

MONTESINOS, F. G., CAMACHO, A. G., VIEIRA, R. (1999). *Analysis of gravimetric anomalies in Furnas Caldera (Sâo Miguel).* J. Volcanol. Geotherm. Res, 92, pp. 67-81.

MOORE, R. B. (1990). *Volcanic geology and eruption frequency, Sâo Miguel, Azores*. Bull. Volcanol., 52, pp. 602-614.

MORALES, J., ALGUACIL, G., MARTÍN, J. B., MARTOS, A. (2007). *The Instituto Andaluz de Geofísica – Universidad de Granada seismic network in Southern Spain*. Orfeus Electronic Newsletters, Vol. 7, Nº 2, p. 1.

NADEAU, R. M., DOLENC, D. (2005). *Non volcanic tremors deep beneath the San Andreas fault*. Science, 307, p. 389.

NASH, G. (1994). *Phase Locked Loop Design Fundamentals*. Nota de Aplicación AN535 de *Motorola*. Disponible en: www.datasheetcatalog.net/es/datasheets_pdf/A/N/5/3/AN535.shtml

NAVARRO-OCHOA, C., GAVILANES-RUIZ, J. C., CORTÉS-CORTÉS, A. (2002). *Movement and emplacement of lava flows at Volcán de Colima, Mexico: November 1998-February 1999*. J. Volcanol. Geotherm. Res., 117, pp. 155-167.

NELSON, T. (1995). *The practical limits of RS-485*. Nota de Aplicación 979 de *National Semiconductor*. Disponible en www.national.com/an/AN/AN-979.pdf

NEWHALL, C. G., SELF, S. (1982). *Volcanic Explosivity Index (VEI): An estimate of explosive magnitude for historical volcanism*. J. Geophys. Res., 87, pp. 1231-1238.

NIAZI, M., BOZORGNIA, Y. (1991). *Behavior of near-source peak horizontal and vertical ground motions over SMART-1 Array, Taiwan*. Bull. Seism. Soc. Am., 80, pp. 2053-2071.

OBARA, K. (2002). *Nonvolcanic deep tremor associated with subduction in Southwest Japan*. Science, 296, pp. 1679-1681.

ORTIZ, R., ARAÑA, V., ASTIZ, M., GARCÍA, A. (1986). *Magnetotelluric study of the Teide (Tenerife) and Timanfaya (Lanzarote) volcanic areas*. J. Volcan. Geotherm. Res., 30 (3-4), pp. 357-377.

ORTIZ, R. (1994). *Sistemas de Conversión Analógica Digital*. En *Instrumentación en Volcanología I*, pp. 55-82. Editado por R. Ortiz. CSIC.

ORTIZ, R., ALGUACIL, G., DEL PEZZO, E., OLMEDILLAS, J. C. (1994). *Array modular de ocho canales*, pp. 63-84. En *Instrumentación en Volcanología II*. Editado por R. Ortiz. CSIC.

ORTIZ, R., GARCÍA, A., ASTIZ, M. (2001). *Instrumentación en Volcanología*. Cabildo de Lanzarote.

OTTEMÖLLER, L., HAVSKOV, J. (1999). *SEISNET: A general purpose virtual seismic network*. Seism. Res. Lett., 70, pp. 522-528.

PALMER, P., FINK. S. (1997). *Interfacing PICmicro MCUs to an LCD Module*. Nota de Aplicación AN587 de *Microchip Technology Inc*. Disponible en : ww1.microchip.com/downloads/en/AppNotes/00587b.pdf

PAZOS, A. (2004). *Estación sísmica digital. Tratamiento digital de señales*. Tesis doctoral. Universidad de Cádiz. 204 pp.

PHILIPS SEMICONDUCTORS (1997). *74HC/HCT4046A: phase-locked-loop with VCO*. Hojas de características del 74HC4046. Disponible en: www.nxp.com/pip/74HC4046AN.html

POUPINET, G., KENNETT, B. L. N. (2004). *On the observations of high frequency PKiKP and its coda in Australia.* Physics of the Earth and Planetary Interiors, 146, pp. 497-511.

POUS, J., HEISE, W., SCHNEGG, P. A., MUÑOZ, G., MARTI, J., SORIANO, C. (2002). *Magnetotelluric study of the Las Cañadas Caldera (Tenerife, Canary Islands); structural and hydrogeological implications*. Earth Planet. Sci. Lett., 204, pp. 249-263.

RABINOVICH, D. (2006). *Teoría de los lazos enganchados en fase (PLL)*. En *Electrónica Aplicada III*. Universidad Tecnológica Nacional, Córdoba, Argentina. Disponible en la página web:
www.profesores.frc.utn.edu.ar/electronica/ElectronicaAplicadaIII/Aplicada/Cap02RedesPLL.pdf

ROGERS, G., DRAGERT, H. (2003). *Episodic tremor and slip on Cascadia subduction zone: the chatter of silent slip.* Science, 300, pp. 1942-1943.

ROST, S., THOMAS, C. (2002). *Array Seismology: Methods and Applications.* Rev. of Geophys., 40, pp. 1-27.

RTD EMBEDDED TECHNOLOGIES (2004). *Using USB Mass Storage Devices in ROM-DOS.* Nota de Aplicación de *RTD Embedded Technologies, Inc*. Disponible en :
www.rtd.com/NEW_appnote/SWM-640000018%20rev%20A.pdf

SACCOROTTI, G., CHOUET, B., MARTINI, M., SCARPA, R. (1998). *Bayesian statistics applied to the location of the source of Stromboli volcano, Italy.* Bull. Seism. Soc. Am., 88, pp. 1099-1111.

SACCOROTTI, G., DEL PEZZO, E. (2000). *A probabilistic approach to the inversion of data from a seismic array and its application to volcanic signals.* Geophys. J. Int., 143, pp. 249-261.

SACCOROTTI, G., ALMENDROS, J., CARMONA, E., IBÁÑEZ, J. M., DEL PEZZO, E. (2001a). *Slowness anomalies from two dense seismic arrays at Deception Island, Antarctica*. Bull. Seism. Soc. Am., 91, pp. 561-571.

SACCOROTTI, G., MARESCA, R., DEL PEZZO, E. (2001b). *Array analysis of seismic noise at Mt. Vesuvius volcano, Italy.* J. Volcan. Geotherm. Res., 110, pp. 79-100.

SACCOROTTI, G., DI LIETO, B., TRONCA, F., FISCHIONE, C., SCARPA, R., MUSCENTE, R. (2006). *Performances of the UNDERground SEISmic array for the analysis of seismicity in Central Italy.* Annals Geophys., 49 (4/5), pp. 1055-1071.

SACKS, I. S., LINDE, A. T., SUYEHIRO, S., SNOKE, J. (1978). *Slow earthquakes and stress distribution*. Nature, 275, pp. 599-602.

SANTA CRUZ, O. M. (2006). *Osciladores LC*. En *Electrónica Aplicada III*. Universidad Tecnológica Nacional, Córdoba, Argentina. Disponible en la página web:
www.profesores.frc.utn.edu.ar/electronica/ElectronicaAplicadaIII/Aplicada/Cap01Osciladores2parte.pdf

SAUCEDO, R., MACÍAS, J. L., BURSIK, M. I., MORA, J. C., GAVILANES, J. C., CORTÉS, A. (2002). *Emplacement of pyroclastic flows during the 1998-1999 eruption of Volcán de Colima, México.* En *Volcán de Colima, México, and its activity in 1997-2000*. Editado por J. F. Luhr, V. M. Zobin y Y. A. Taran. J. Volcanol. Geotherm. Res., 117, pp. 129-153.

SCHMIDT, R. O. (1986). *Multiple emitter location and signal parameter estimation.* IEEE Trans. Antennas Propag., 34, pp. 276-280.

SERRANO, I., MARTÍN, J. B., JIMÉNEZ, A., ROSSI, N., ALMENDROS, J. (2006). *Actividad sísmica en Isla Decepción durante la campaña 2005-2006.* VII Simposio Español de Estudios Polares, Granada (España). Póster. Incluido en el CD adjunto.

SONG, X., RICHARDS, P. G. (1996). *Seismological evidence for differential rotation of the Earth's inner core.* Nature, 382, pp. 221-224.

SONY (2000). *CXK581000ATM/AYM/AM/AP: 131072-word x 8-bit High Speed CMOS Static RAM*. Hojas de características de las memorias CXK581000 de Sony. Disponible en: www.datasheetarchive.com/search.php?q=cxk581000&sType=part&ExactDS=Starts

SPUDICH, P., BOSTWICK, T. (1997). *Studies of the seismic coda using an earthquake cluster as a deeply buried seismograph array*. Journ. Geophys. Res., 92, pp. 10526-10546.

TEXAS INSTRUMENTS (1999). *Analysis of the Sallen-Key Architecture. Application report* de *Texas Instruments*. Disponible en focus.ti.com/lit/an/sloa024b/sloa024b.pdf.

TISCHER, M., JENNRICH, B. (1996). *PC Interno 5, Programación de Sistemas*. Marcombo, Barcelona.

TOOMEY, D. R., SOLOMON, S., PURDY, G. M. (1994). *Tomographic imaging of the shallow crustal structure of the East Pacific Rise at 9º30'N.* J. Geophys. Res., 99, B12, pp. 24135-24157.

TRIMBLE NAVIGATION LIMITED (1999). *Lassen SK II GPS System Designer Reference Manual.* Manual del receptor GPS *Lassen SK II*. Disponible en www.trimble.com

TRONCA, F., FISCHIONE, C., SACCOROTTI, G., SCARPA, R. (2007). *Recent seismicity in Central Italy as observed by the UNDERSEIS seismic array*. Bull. Seism. Soc. Am. (en prensa).

U.S. MILITARY STANDARDS. TELECOMMUNICATIONS AND TIMING GROUP (2004). *IRIG Serial Time Code Formats*. Publicado por el Secretariado del *Range Commanders Council, U.S. Army White Sands Missile Range, New Mexico 88002-5110.* Disponible en www.jcte.jcs.mil/RCC/manuals/200-04/index.html

UTHEIM, T., HAVSKOV, J. (1999). *The Seislog data acquisition system, Version 8.0, manual and software.* Institute of Solid Earth Physics, University of Bergen, Norway. Disponible en www.geo.uib.no/seismo/software/software.html

UTHEIM, T., HAVSKOV, J., NATVIK, Y. (2001). *Seislog data acquisition systems.* Seism. Res. Lett., 72, pp. 77-79.

VAREKAMP, J., OUIMETTE, A., HERMÁN, E., BERMÚDEZ, A., DELPINO, D. (2001). *Hydrotermal element fluxes from Copahue, Argentina: a "beehive" volcano in turmoil*. Geology, Vol. 29, Nº 11, pp. 1059-1062.

VIDAL, F., DE MIGUEL, F., ALGUACIL, G., GUIRAO, J. M. (1981). *Características de la secuencia sísmica granadina del año 1979.* IV Asamblea Nacional de Geodesia y Geofísica. Instituto Geográfico Nacional, Vol. I, pp. 423-428.

VIDAL, F., DE MIGUEL, F. (1983). *Datos macrosísmicos de los terremotos sentidos en la Depresión de Granada durante el año 1979.* Publicaciones del Observatorio de Cartuja, report 3, 77 pp.

VIDALE, J. E., DODGE, D. A., EARLE, P. S. (2000). *Slow differential rotation of the Earth's inner core indicated by temporal changes in scattering.* Nature, 405, pp. 445-448.

VIVES, T. (2003). *El Observatorio Astronómico de Cartuja*. En *Historia del Observatorio de Cartuja, 1902-2002. Nuevas investigaciones.* Editado por M. Espinar, J. A. Esquivel y J. A. Peña. Ayuntamiento de Granada. [CD]. Disponible en: www.ugr.es/~iag/ins/incd.html

WAGNER, G. S., LANGSTON, C. A. (1992). *Body-to-surface-wave scattered energy in teleseismic coda observed at the NORESS seismic array*. Bull. Seism. Soc. Am., 82, pp. 2126-2138.

XU, Z., SCHWARTZ, S., LAY, T. (1996). *Seismic Wave-Field Observations at a Dense, Small-Aperture Array Located on a Landslide in the Santa Cruz Mountains, California*. Bull. Seism. Soc. Am., 86, pp. 655-669.

YU, T. T., FERNANDEZ, J., TSENG, C. L., SEVILLA, M. J., ARAÑA, V. (2000). *Sensitivity test of the geodetic network in Las Cañadas Caldera, Tenerife, for volcano monitoring*. J. Volcan. Geotherm. Res., 103, pp. 393-407.

ZANDOMENEGHI, D., BARCLAY, A. H., ALMENDROS, J., BEN ZVI, T., WILCOCK, E., IBÁÑEZ, J. M., TOMODEC WORKING GROUP (2005). *Preliminary seismic tomography of Deception Island volcano, South Shetland Islands, Antarctica.* 2005 Fall Meeting of the American Geophysical Union (AGU), San Francisco (EEUU). Póster. Incluido en el CD adjunto.

ZEILINGA DE BOER, J., SANDERS, D. T. (2002). *Volcanoes in Human History*. Princeton University Press, Princeton.

ZOBIN, V. M., GONZÁLEZ-AMÉZCUA, M., REYES-DÁVILA, G. A. (2002a). *Seismotectonic deformation of the volcanic edifice prior to the 1998 lava eruption of Volcán de Colima, México*. Bull. Volcanol., 64, pp. 349-355.

ZOBIN, V. M., LUHR, J. F., TARAN, Y. A., BRETÓN, M., CORTÉS, A., DE LA CRUZ-REYNA, S., DOMÍNGUEZ, T., GALINDO, I., GAVILANES, J. C., NUÑEZ, J. J., NAVARRO, C., RAMIREZ, J. J., REYES, G. A., BURSIK, M., VELASCO, J. E., ALATORRE, E., SANTIAGO, H. (2002b). *Overview of the 1997-2000 activity of Volcán de Colima, México.* J. Volcanol. Geotherm. Res., 117, pp. 1-19.

ZOBIN, V. M., GONZÁLEZ-AMÉZCUA, M., REYES-DÁVILA, G. A., DOMÍNGUEZ, T., CERDA-CHACÓN, J., CHÁVEZ-ÁLVAREZ, M. J. (2002c). *The comparative characteristics of the 1997-1998 seismic swarms preceding the November 1998 eruption of Volcán de Colima, México*. J. Volcanol. Geotherm. Res., 117, pp. 47-60.

ZOBIN, V. M., NAVARRO-OCHOA, C. J., REYES-DÁVILA, G. A. (2006). *Seismic quantification of the explosions that destroyed the dome of Volcán de Colima, México, in July-August 2003*. Bull. Volcanol., 69 (2), pp. 141-147.

ZOLLO, A., CAPUANO, P., CORCIULO, M. (editores) (2006). *Geophysical Exploration of the Campi Flegrei (southern Italy) caldera interiors: Data, Methods and Results*. Doppiavoce Editore, Napoli. 225 pp.

2. Páginas web consultadas[76]

ACROSSER, fabricante de PCs industriales:
www.acrosser.com

ARIZONA MICROCHIP TECHNOLOGIES, empresa fabricante de los microcontroladores PIC:
www.microchip.com

CHAMPION TECHNOLOGIES (actualmente MTRON), fabricante de osciladores a cristal controlados por tensión (VCXO):
www.mtron.com

CHAYKA, sistema de radionavegación:
en.wikipedia.org/wiki/CHAYKA

CIRRUS LOGIC, fabricante de circuitos integrados:
www.cirrus.com/en/

CTBT, *Comprehensive Nuclear-Test-Ban Treaty:*
pws.ctbto.org

DCF77, señal horaria de radio:
www.dcf77.com/index.htm

DIGI-KEY, suministrador de componentes electrónicos:
es.digikey.com

E-RUPTION PROJECT, *A satellite telecommunication and Internet-based seismic monitoring system for volcanic eruption forecasting and risk management*:
www.e-ruption.info/main.htm

ERMES, *Environmental Radioactivity Monitoring for Earth Sciences*:
www.fis.uniroma3.it/~plastino/ERMES

FARNELL, suministrador de componentes electrónicos:
es.farnell.com

FILTROS SALLEN Y KEY :
www.ecircuitcenter.com/Circuits/opsalkey1/opsalkey1.htm

IAG, INSTITUTO ANDALUZ DE GEOFÍSICA:
www.ugr.es/~iag/
www.ugr.es/~geofisic/ (Área de prospección)
www.ugr.es/~iag/red.html (Red Sísmica de Andalucía)
www.ugr.es/~iag/divulgacion/div_i.html (Terremoto de Alhama de 1884)

INGV, *Istituto Nazionale di Geofisica e Vulcanologia*, sede de Roma:
www.ingv.it/~roma

[76] Sólo se incluyen las páginas web a las que se ha hecho referencia en la memoria, siempre que la consulta no haya implicado la descarga de ningún documento con título, autor(es) y fecha de publicación (ej.: hojas de características de componentes electrónicos). En ese caso, la referencia se incluye en la sección anterior.

LIPPERT, fabricante de PCs industriales:
www.lippert-at.com

LNGS, *Laboratori Nazionali del Gran Sasso*:
www.lngs.infn.it

LORAN, sistema de radionavegación:
www.navcen.uscg.gov/loran/Default.htm

MSF, señal horaria de radio:
www.npl.co.uk/time/

OBSERVATORIO VULCANOLÓGICO DE LA UNIVERSIDAD DE COLIMA:
www.ucol.mx/volcan/

OSSERVATORIO VESUVIANO:
www.ov.ingv.it

PELI, fabricante de cajas y contenedores:
www.peli.com

PLLS, información general:
en.wikipedia.org/wiki/Phase-locked_loop
www.ifent.org/lecciones/PLL/

PUERTO PARALELO, información general sobre modos de operación y estándar IEEE-1284-1994:
www.fapo.com/1284int.htm
es.wikipedia.org/wiki/IEEE_1284

QNX, sistema operativo para aplicaciones en tiempo real:
www.qnx.com

RFSOLUTIONS, enlaces telemétricos de baja potencia:
www.rfsolutions.co.uk

RS AMIDATA, suministrador de componentes electrónicos:
www.amidata.es

TARDIS, página de descarga del programa de sincronización Tardis 2000:
www.kaska.demon.co.uk/download.htm

TOMODEC, proyecto para la TOMOgrafía sísmica de alta resolución de la isla DECepción:
www.utm.csic.es/proyecto.asp?id=%7BCD8D2A9E-387B-44B0-84C5-C05FFF9743A4%7D

UNIVERSIDAD DE BERGEN, sección de geofísica, página de descarga de programas:
www.geo.uib.no/seismo/software/software.html

UNIVERSIDAD DE BERGEN, descarga del programa GPSCFG.EXE para configuración del receptor GPS Garmin 35:
pcom3.geo.uib.no/Healy-2005/equipment/gps/garmin_gps-35-hvs/gps-35.html

VNC, programa de control remoto:
www.realvnc.com

WINHEX, editor hexadecimal de ficheros:
www.x-ways.net/winhex/index-e.html

WWVB, señal horaria de radio:
tf.nist.gov/stations/wwvb.htm

ANEXOS

ANEXO I

ANTENA DEL GRAN SASSO

Anexo I.A:

Programas de la antena del Gran Sasso

ANEXO I.A.1.- PROGRAMA DE LAS ESTACIONES

I.A.1.a. FICHERO DE CABECERA PARA LA GESTIÓN DEL PUERTO SERIE

Fichero:MISERIE.H

/* Declaración de constantes y funciones para la gestión de los puertos serie. Las funciones de gestión del puerto serie del PC no se han realizado en este proyecto, sino que se ha utilizado una librería que ya se había probado en otros programas de adquisición de datos del Instituto Andaluz de Geofísica. El fichero de cabecera original, SERIE.H, se ha modificado para incluir los prototipos de las funciones de gestión del interfaz serie RS-485, que sí se han realizado en el presente trabajo.

En nuestra aplicación no se han utilizado todas las funciones declaradas en este fichero. El cuerpo de las funciones usadas se encuentra en el fichero fuente del programa ESTACION (anexo I.A.1.b).*/

```
/*************************************************************************
  Macros
*************************************************************************/
// ESTADOS POSIBLES EN LA COMUNICACIÓN
#define INICIAL         0
#define ENVIANDO        1
#define RECIBIENDO      2

// PUERTOS DE COMUNICACIÓN
#define COM1            0
#define COM2            1
#define COM3            2
#define COM4            3

// DEFINICIONES PARA LA CONFIGURACIÓN DEL PUERTO
#define NINGUNA         0       // Definiciones de paridad
#define IMPAR           1
#define PAR             2

#define LON7            0       // Definiciones de longitud de carácter
#define LON8            1

#define STOP1           0       // Definiciones de bits de stop
#define STOP2           1

#define BAUD110         0       // Definiciones de velocidad
#define BAUD300         1
#define BAUD600         2
#define BAUD1200        3
#define BAUD1800        4
#define BAUD2400        5
#define BAUD3600        6
#define BAUD4800        7
#define BAUD9600        8
```

```
#define BAUD19200       9
#define BAUD38400       10
#define BAUD57600       11
#define BAUD115200      12

/*************************************************************************
```

Declaración de las funciones

```
*************************************************************************/
// FUNCIONES PARA LA GESTIÓN DE LOS PUERTOS SERIE DEL PC
void Ruptura_RS(int Puerto);
int DatoDisponible(int Puerto);
void VaciarBuffer (int Puerto);
void CargarTimer (int Puerto,int NumTicks);
int ErrorTimeOut (int Puerto);
int RecibirCaracter(int Puerto, char *Ch);
void RecibirBloque (int Puerto, char *Buffer, int NumCar);
int RecibirBloqueT (int Puerto, char *Buffer, int NumCar, int NumTicks);
void RecibirString (int Puerto, char *Buffer);
int RecibirStringT (int Puerto, char *Buffer, int NumTicks);
void MandarCaracter (int Puerto, char Param);
void MandarBloque(int Puerto, char *Buffer, int NumCar);
void InicializarComunicacionSerie (int Puerto,int Velocidad,int Paridad, int BitsDatos,int BitsStop);
int ConfigurarPuertoSerie (int Puerto,int Velocidad,int Paridad,int BitsDatos, int BitsStop);
void FinalizarComunicacionSerie(void);

// FUNCIONES PARA  COMUNICACION POR RS485
void pone_DE_0(int Puerto);
void pone_DE_1(int Puerto);
void EsperaFinEnvio(int Puerto);
```

I.A.1.b. PROGRAMA DE LAS ESTACIONES

Fichero: ESTACION.C

/* Programa definitivo para las estaciones. Incluye todas las funciones creadas expresamente para la aplicación, más las necesarias para la gestión del puerto serie tomadas de la librería SERIE.C. Las funciones de control de los CADs AD7710 se han tomado de otros programas operativos (Ortiz, R., comunicación personal), con las modificaciones necesarias para adaptarlas a nuestra aplicación. El programa principal se encuentra al final. */

```
/*************************************************************************
  Macros
*************************************************************************/
#define kfrec                   19200   // 9.8304MHz/512: cte. para programar frec. muestreo
#define TRUE                    1
#define FALSE                   0
#define MAL                     1
#define OK                      0
#define portpar                 888
#define portpar1                portpar+1
#define portpar2                portpar+2
#define NUM_PORTS               4
#define TRAMAS_BUFFER           60
#define BYTES_TRAMADATOS        908
#define Tamano_Buffer_RS        TRAMAS_BUFFER * BYTES_TRAMADATOS
#define Tamano_Buf_Recepcion    48      // Para que quepan 18 tramas de peticion
#define NUMERO_ESTACIONES       4
#define PUERTO_WD               0x214
#define  TIEMPO_WD              0x87    // 0x81, 0x82, ...0x87 = 6, 12, ...42 segundos

// CODIGOS DE TRAMA
#define TIPO_TRAMA_DATOS                0xC3
#define TIPO_TRAMA_ERROR                0x3C
#define TIPO_TRAMA_PETICION             0x2B
#define TIPO_TRAMA_CAMBIOGANANCIA       0xB2
#define TIPO_TRAMA_REENVIO              0xD4
#define TIPO_TRAMA_DESCONEXION          0xE5
#define TIPO_TRAMA_RECONEXION           0x5E
#define TIPO_TRAMA_SINCRONIZACION       0xF6
#define ERROR_INIC_ADCs                 0xA0
#define ERROR_SATURACION_BUFFER         0xB1
#define ERROR_SINCRONIZACION            0xD3
#define FIN_TRAMA                       0xAA
#define ERROR_RECEPCION_TRAMA           0xF8
#define DESCONEXION_TIMEOUT             0x27

// COMANDOS para los CADs AD7710
#define ESPERA_888      7       // SDATA=1 TFS=1 SINC=1
#define LEER_888        7       // SDATA=1 TFS=1 SINC=1
#define ESCR_888_1      5       // SDATA=1 TFS=0 SINC=1
#define ESCR_888_0      4       // SDATA=0 TFS=0 SINC=1
#define SINCRO_888      3       // SDATA=1 TFS=1 SINC=0
```

```
#define ESPERA_890          28      // A0=1 RFS=1 SCK=0 IRQEN=1
#define LEER_D_890          24      // A0=1 RFS=0 SCK=0 IRQEN=1
#define LEER_D_890_C        16      // A0=1 RFS=0 SCK=1 IRQEN=1
#define ESCR_D_890          28      // A0=1 RFS=1 SCK=0 IRQEN=1
#define ESCR_D_890_C        20      // A0=1 RFS=1 SCK=1 IRQEN=1
#define LEER_R_890_P        30      // A0=0 RFS=1 SCK=0 IRQEN=1
#define LEER_R_890          26      // A0=0 RFS=0 SCK=0 IRQEN=1
#define LEER_R_890_C        18      // A0=0 RFS=0 SCK=1 IRQEN=1
#define ESCR_R_890          30      // A0=0 RFS=1 SCK=0 IRQEN=1
#define ESCR_R_890_C        22      // A0=0 RFS=1 SCK=1 IRQEN=1

#include <miserie.h>
#include <dos.h>
#include <stdio.h>
#include <conio.h>

/*****************************************************************************
  Declaración de variables
*****************************************************************************/
unsigned char CodigoTarjeta[NUMERO_ESTACIONES] = { 0x11, 0x22, 0x44, 0x88 };

// Variables para la inicialización de los CADs
int n_bits = 24;
unsigned long adc[3];
int modo, gain = 0, via, power, com_current, burn_current, bipolar, mps, bits, fac_frecuencia;
unsigned long dato;
long PalCtrolEscrita, PalCtrolLeida[3];      // Para ver si hay error de inicialización de los CADs

// Variables para la lectura de datos y formación de tramas
int ContBytesTramaDatos = 0;
unsigned char TramasListas = 0;
union UNlongCUATROchar
{
        long largo;
        unsigned char ch[4];
}adcs[3];

// Variables para la lectura del código de tarjeta
unsigned char MiCodigoTarjeta;

// Variables para la identificación de tramas recibidas
unsigned char CaracterAIdentificar;
struct semaforos {
        unsigned RecibiendoTrama:       1;
        unsigned Cabecera:              1;
        unsigned CambioGanancia:        1;
        unsigned EsperandoFinTrama:     1;
} Semaf = { FALSE, FALSE, FALSE, FALSE };

// Flags, que se activan ante diversos eventos
struct {
        unsigned EstacionActiva:        1;
        unsigned TramaLista:            1;
        unsigned NuevaGanancia:         1;
        unsigned MandarTrama:           1;
```

```
        unsigned ErrorRecepcionTrama:   1;
        unsigned ErrorSatBuffer:        1;
        unsigned ErrorInicADCs:         1;
        unsigned MandarErrorADCs:       1;
        unsigned TramaIdentificada:     1;
        unsigned Reenvio:               1;
        // Flags para la sincronización con el RTC del oscilador patrón
        unsigned ErrorSincronizacion:   1;
        unsigned Muestra1PulsoSinc1:    1;
        unsigned Sincronizando:         1;
        unsigned Esperando27_0s:        1;
        unsigned Esperando4_1s:         1;
        unsigned RecibiendoCodigo:      1;
        unsigned ActualizarVars:        1;
        unsigned SincronizacionOK:      1;
        unsigned Fin:                   1;
} Flag =
{ TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,
FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE };

// Variables para la gestión de los puertos serie
struct INS8250
{
        int thr;        // Registro de retención de transmisión
        int rbr;        // Registro de retención de recepción
        int ier;        // Registro de activación de interrupción
        int lcr;        // Registro de control de línea
        int mcr;        // Registro de control del módem
        int lsr;        // Registro de la línea de estado
        int msr;        // Registro de estado del módem
} RS232[NUM_PORTS] =
{
        { 0x3f8,0x3f8,0x3f9,0x3fb,0x3fc,0x3fd,0x3fe },
        { 0x2f8,0x2f8,0x2f9,0x2fb,0x2fc,0x2fd,0x2fe },
        { 0x3e8,0x3e8,0x3e9,0x3eb,0x3ec,0x3ed,0x3ee },
        { 0x2e8,0x2e8,0x2e9,0x2eb,0x2ec,0x2ed,0x2ee }
};
unsigned char Error_RS[NUM_PORTS];
unsigned char Buffer_RS[Tamano_Buffer_RS];
unsigned int Princ_Buf_RS,Fin_Buf_RS;
int TimeOut[NUM_PORTS],ErrorTOut[NUM_PORTS];
int PuertoActivo[NUM_PORTS] = { FALSE, FALSE, FALSE, FALSE };

// Variables para la gestión de las interrupciones
int RelojActivo = 0, IRQ4Activo = 0, IRQ3Activo = 0;
void interrupt (*GuardaIntIRQ7)();
void interrupt (*GuardaIntIRQ4)();      // Guarda vector de interrupción antiguo
void interrupt (*GuardaIntIRQ3)();      // Guarda vector de interrupción antiguo
void interrupt (*GuardaIntReloj)();     // Guarda vector del timer

// Variables para la recepción de las tramas de control
unsigned char Error_Recepcion;
unsigned char Buffer_Recepcion[Tamano_Buf_Recepcion];   // Para meter las tramas de petición o cambio
int Princ_Buf_Recepcion = 0, Fin_Buf_Recepcion = 0;     //de parámetros conforme lleguen. Es un buffer
                                                        //pequeño, porque no va a ser necesario
                                                        //almacenar más de una trama en él.
```

```
// Variables para la gestión del reloj software de tiempo real
unsigned int i;
unsigned char fraccion, segundo, minuto, hora, year;
union UNintDOSchar {
        unsigned int    total;
        unsigned char   mitad[2];
        } dia;
unsigned int dias_year;     // Para los años bisiestos

// Variables para la gestión del reenvío de tramas y de los cambios de ganancia
int InicioDatos = 0;
unsigned char UltimaTramaEnviada;
int InicioTramaActual;

// Variables para la sincronización con el RTC del oscilador patrón
unsigned int Mascara;
unsigned char ContCeros, ContUnos;
unsigned char MinutoCL, HoraCL, YearCL;
unsigned int DiaCL;
unsigned int ContBitsCL, ContSegundosCL;

/*************************************************************************
```

Declaración de las funciones

```
*************************************************************************/
// Funciones para la gestión de los CADs:
void sincronismo(void);
void escribe_registro(void);
void lee_registro(void);
int ErrorInicADCs(void);
void LeerGanancia(void);
void iniciacion(void);
void leer_dato(void);

// Funciones para la gestión de interrupciones y rutinas de servicio de las mismas
void interrupt Reloj ();
void interrupt Trat_IRQ3();
void interrupt Trat_IRQ4(void);
void interrupt Trat_IRQ7(void);
void InicializarIRQ7(void);
void ReponerVectoresIRQ7(void);

// Funciones para la inicialización y tratamiento del puerto serie:
void InicializarComunicacionSerie ();
int ConfigurarPuertoSerie ();
void FinalizarComunicacionSerie();

// Funciones para la recepción y transmisión de tramas:
int AnalizarCaracter(void);
void LeerTrama(void);
void DesactivarDE(int Com_RS);
void ActivarDE(int Com_RS);
void MandarCaracter (int Com_RS, char Param);
void MandarTramaErrInic(void);
void MandarTramaErrSat(void);
void MandarTramaErrSinc(void);
```

```
void MandarTramaDatos(void);
void ReenviarUltimaTrama(void);

// Funciones para la gestión de errores y de tramas recibidas:
int Error(void);
void GestionErrores(void);
void CambiarGanancia(void);

// Funciones para la gestión del reloj software:
void InicializarRelojSoftware(void);
void SincronizarRelojSoftware(void);
void LeerCodigoLento(void);

// Otras funciones:
void RefrescaWD(void);
void EsperaActivacion(void);
void LeerCodigoTarjeta(void);
unsigned char Salir(void);

/*****************************************************************************
  Definición de las funciones
*****************************************************************************/
/*-----------------------------------------------------------------------------------------------------------------
  FUNCIONES PARA LA GESTIÓN DE LOS CADS
-----------------------------------------------------------------------------------------------------------------*/
void sincronismo(void)
{
        outportb(portpar,  SINCRO_888);
        outportb(portpar,  ESPERA_888);
        outportb(portpar2, ESPERA_890);
}

void escribe_registro(void)
{
int i;
long sale, muestra;

        dato = (long) modo;
        dato <<= 3;
        dato |= (long) gain;
        dato <<= 1;
        dato |= (long) via;
        dato <<= 1;
        dato |= (long) power;
        dato <<= 1;
        dato |= (long) bits;
        dato <<= 1;
        dato |= (long) com_current;
        dato <<= 1;
        dato |= (long) burn_current;
        dato <<= 1;
        dato |= (long) bipolar;
        dato <<= 12;
        dato |= (long) fac_frecuencia;
        PalCtrolEscrita = dato;          // Para ver si hay error de inicialización
```

```
	muestra = 1;
	muestra <<= 23;
	outportb(portpar2, ESCR_R_890);
	for(i = 0; i < 24; i++)
	{
		outportb(portpar2, ESCR_R_890);
		sale = dato & muestra;
		if(sale == 0)
		{
			outport(portpar, ESCR_888_0);
		}
		else
		{
			outport(portpar, ESCR_888_1);
		}
		muestra >>= 1;
		outportb(portpar2, ESCR_R_890_C);
	}
	outportb(portpar2, ESCR_R_890);
	outportb(portpar,  ESPERA_888);
	outportb(portpar2, ESPERA_890);
}

void lee_registro(void)
{
int i;
unsigned char entra[3];
unsigned long dato_leido[3];

	for (i=0; i<3; i++) dato_leido[i] = 0;
	outportb(portpar,  LEER_888);
	outportb(portpar2, LEER_R_890_P);
	for(i = 0; i < 24; i++)
	{
		outportb(portpar2, LEER_R_890);
		entra[0] = inportb(portpar1) & 0x80;		// Canal 1
		entra[1] = inportb(portpar1) & 0x20;		// Canal 2
		entra[2] = inportb(portpar1) & 0x10;		// Canal 3
		outportb(portpar2, LEER_R_890_C);
		dato_leido[0] <<= 1;
		if (entra[0] == 0) dato_leido[0] |= 1;		// El bit del puerto paralelo que
		dato_leido[1] <<= 1;				//corresponde al canal 1 está invertido
		if (entra[1] != 0) dato_leido[1] |= 1;
		dato_leido[2] <<= 1;
		if (entra[2] != 0) dato_leido[2] |= 1;
	}
	outportb(portpar,  ESPERA_888);
	outportb(portpar2, ESPERA_890);
	for (i=0; i<3; i++) PalCtrolLeida[i] = dato_leido[i];	// Para ver si hay error de inicialización
}

int ErrorInicADCs(void)
{
unsigned char i;
	for (i=0; i<3; i++)
	{
```

```
                if (PalCtrolEscrita != PalCtrolLeida[i]) return(TRUE);
        }
        return(FALSE);
}


/* Cuando se quiere reiniciar una estación con la misma ganancia se guarda ésta en el fichero GAIN.BIN y se
sale del programa. El watchdog reinicia el PC y se vuelve a lanzar el programa de adquisición. Este busca el
fichero GAIN.BIN y, si existe, lee la ganancia de él. Si no existe, se toma el valor por defecto (gain = 0 →
ganancia = 1). */
void LeerGanancia(void)
{
        FILE *FichGain;
        if (( FichGain = fopen("C:\GAIN.BIN", "rb")) == NULL)
        {
                gain = 0;
                printf("\n No se puede abrir C:\\GAIN.BIN para lectura");
                printf("\n Se toma gain = 0");
        }
        else
        {
                gain = getc(FichGain);
                printf("\n Ganancia leída: %i", gain);
                fclose(FichGain);
        }
}

void iniciacion(void)
{
        modo = 0;
        via = 0;
        power = 0;
        com_current = 0;
        burn_current = 0;
        bipolar = 0;
        mps = 100;
        switch(n_bits)
        {
                case 16:
                        bits = 0;
                        break;
                case 24:
                        bits = 1;
                        break;
                default:
                        bits = 0;
        }
        fac_frecuencia = kfrec/mps;
        escribe_registro();
        delay(500);
        lee_registro();
        if (ErrorInicADCs()) Flag.ErrorInicADCs = TRUE;
}
```

```
void leer_dato(void)
{
int i;
long data1, data2, data3;
unsigned char entra;

        /* Se sincroniza con el flanco ascendente del pulso de segundo del código lento. Primero se espera a
        que la entrada esté a cero y luego a que se vuelva a poner a uno. */
        do
        {
        }
        while(inportb(portpar1) & 0x08);
        do
        {
        }
        while(!(inportb(portpar1) & 0x08));
        fraccion = 0;
        Flag.Sincronizando = TRUE;
        Flag.Esperando27_0s = TRUE;
        ContCeros = ContUnos = 0;

        data1 = 0;
        data2 = 0;
        data3 = 0;
        outportb(portpar,  LEER_888);
        outportb(portpar2, LEER_D_890);
        for(i = 0; i < n_bits; i++)
        {
                outportb(portpar2, LEER_D_890);
                data1 <<= 1;
                data2 <<= 1;
                data3 <<= 1;
                entra = inportb(portpar1);
                outportb(portpar2, LEER_D_890_C);
                if((entra & 0x80) == 0)
                {
                        data1 |= 1;
                }
                if((entra & 0x20) != 0)
                {
                        data2 |= 1;
                }
                if((entra & 0x10) != 0)
                {
                        data3 |= 1;
                }
        }
        outportb(portpar,  ESPERA_888);
        outportb(portpar2, ESPERA_890);
        adc[0] = data1;
        adc[1] = data2;
        adc[2] = data3;
}
```

```
/*-------------------------------------------------------------------------------------------------------------------
  FUNCIONES PARA LA GESTIÓN DE INTERRUPCIONES Y RUTINAS DE SERVICIO
-------------------------------------------------------------------------------------------------------------------*/

//  TRATAMIENTO DE LA INTERRUPCIÓN DEL TIMER PARA LA GESTIÓN DEL TIME-OUT.
void interrupt Reloj ()
        {
         static int i;
         for (i=0; i<NUM_PORTS; i++)
                {
                        if (TimeOut[i]) TimeOut[i]--;
                        if (!TimeOut[i]) ErrorTOut[i] = TRUE;
                }
                //(*GuardaIntReloj)();
        }

/*  RUTINA DE TRATAMIENTO DE LA INTERRUPCION SERIE. Cuando llega un carácter al puerto serie se genera una interrupción que hace que se salte a esta rutina. Aquí se toma el carácter recibido y se introduce en el buffer */
void interrupt Trat_IRQ4(void)
{
static int Puerto;
        enable();
        Puerto = COM1;
        Error_Recepcion |= (inportb(RS232[Puerto].lsr) & 0x1e);
        Buffer_Recepcion[Fin_Buf_Recepcion] = (unsigned char) inportb (RS232[Puerto].rbr);
        Fin_Buf_Recepcion = (Fin_Buf_Recepcion +1) % Tamano_Buf_Recepcion;
        outportb(0x20, 0x20);
}

void interrupt Trat_IRQ3(void)
        {
         static int Puerto;
         enable();
         /*if (DatoDisponible (COM2)) Puerto = COM2; else Puerto = COM4;*/
         Puerto = COM2;
         Error_RS[Puerto] |= inportb(RS232[Puerto].lsr) & 0x1e;
         Buffer_RS[Fin_Buf_RS] = inportb (RS232[Puerto].rbr);
         Fin_Buf_RS = (Fin_Buf_RS + 1) % Tamano_Buffer_RS;
         TimeOut[Puerto] = 0;
/*  disable(); */
         outportb (0x20,0x20); /* Pone a cero la bandera de interrupción */
         outportb(0xA0, 0x20);
        }


/*  FUNCIONES PARA EL CONTROL DE LA INTERRUPCION DE TOMA DE DATOS (IRQ7, INTERRUPCION DEL PUERTO PARALELO LPT1) */

//  RUTINA DE SERVICIO DE LA INTERRUPCIÓN IRQ7. Se ocupa de seis tareas (ver números).
 void interrupt Trat_IRQ7(void)
{
        static unsigned char i, j, k;
        static long data1, data2, data3;
        static unsigned char entra;
```

```
// 1.- Si corresponde, sincroniza con el flanco de segundo:
if (Flag.Muestra1PulsoSinc1)
{
        //El flag no se deshabilita aquí porque hace falta más abajo (en la gestión del reloj
        //s*oftware.*).
        disable();          // Mientras espera el primer pulso de segundo, deshabilita las
        do                  //deshabilita las interrupciones para que no se aniden.
        {
        }
        while(inportb(portpar1) & 0x08);
        do
        {
        }
        while(!(inportb(portpar1) & 0x08));
        outportb(portpar, inportb(portpar) | 0x80);    // Pone el bit 7 del primer registro del puerto
                                                       //paralelo (pin 9 del conector DB25) a 1,
                                                       //para ver si sincroniza bien.
        // Pone a 0 todos los datos perdidos mientras esperábamos el flanco de PPS e inicia una
        //nueva trama:
        Fin_Buf_RS = 0;
        Princ_Buf_RS = 0;
        ContBytesTramaDatos = 0;
        enable();
}

// 2.- Lee el dato:
//enable();
data1 = 0;
data2 = 0;
data3 = 0;
outportb(portpar,  LEER_888);
outportb(portpar2, LEER_D_890);
for(i = 0; i < n_bits; i++)
{
        outportb(portpar2, LEER_D_890);
        data1 <<= 1;
        data2 <<= 1;
        data3 <<= 1;
        entra = inportb(portpar1);
        outportb(portpar2, LEER_D_890_C);
        if((entra & 0x80) == 0)
        {
                data1 |= 1;
        }
        if((entra & 0x20) != 0)
        {
                data2 |= 1;
        }
        if((entra & 0x10) != 0)
        {
                data3 |= 1;
        }
}
outportb(portpar,  ESPERA_888);
outportb(portpar2, ESPERA_890);
```

```
adcs[0].largo = data1;
adcs[1].largo = data2;
adcs[2].largo = data3;

// 3.- Gestiona el reloj software:
/*   Si es la primera muestra del primer flanco de una nueva sincronización, pone las centésimas
(variable 'fraccion') a 0. En caso contrario, incrementa el reloj software. */
if (Flag.Muestra1PulsoSinc1)
        Flag.Muestra1PulsoSinc1 = FALSE;
        fraccion = 0;
        ContCeros = 0;
        ContUnos = 0;
}
else
{
        if (++fraccion == mps)
        {
                fraccion = 0;
                if (++segundo == 60)
                {
                        segundo = 0;
                        if (++minuto == 60)
                        {
                                minuto = 0;
                                if (++hora == 24)
                                {
                                        hora = 0;
                                        if (++dia.total > dias_year)
                                        {
                                                dia.total = 0;
                                                if(++year >= 100) year -= 100;
                                                if ((year % 4) == 0) dias_year = 366;
                                        else dias_year = 365;
                                        }
                                }
                        }
                }
        }
}

// 4.- Mete los datos (y la ganancia y cabecera temporal, si es inicio de trama) en el buffer:
if (ContBytesTramaDatos == 0)
{
        InicioTramaActual = Fin_Buf_RS;  // Si hay cambio de ganancia, se volverán a inicializar
                                         //los CADs y Fin_Buf_RS volverá a esta posición, con
                                         //lo que sólo se pierde un trozo de trama.
        //Mete la ganancia en el buffer:
        Buffer_RS[Fin_Buf_RS] = gain;
        Fin_Buf_RS = (Fin_Buf_RS + 1) % Tamano_Buffer_RS;
        ContBytesTramaDatos++;
        //Mete la cabecera temporal en el buffer:
        Buffer_RS[Fin_Buf_RS] = fraccion;
        Fin_Buf_RS = (Fin_Buf_RS + 1) % Tamano_Buffer_RS;
        Buffer_RS[Fin_Buf_RS] = segundo;
        Fin_Buf_RS = (Fin_Buf_RS + 1) % Tamano_Buffer_RS;
        Buffer_RS[Fin_Buf_RS] = minuto;
```

```
                Fin_Buf_RS = (Fin_Buf_RS + 1) % Tamano_Buffer_RS;
                Buffer_RS[Fin_Buf_RS] = hora;
                Fin_Buf_RS = (Fin_Buf_RS + 1) % Tamano_Buffer_RS;
                Buffer_RS[Fin_Buf_RS] = dia.mitad[0];
                Fin_Buf_RS = (Fin_Buf_RS + 1) % Tamano_Buffer_RS;
                Buffer_RS[Fin_Buf_RS] = dia.mitad[1];
                Fin_Buf_RS = (Fin_Buf_RS + 1) % Tamano_Buffer_RS;
                Buffer_RS[Fin_Buf_RS] = year;
                Fin_Buf_RS = (Fin_Buf_RS + 1) % Tamano_Buffer_RS;
                ContBytesTramaDatos += 7;
        }
        //Mete los nueve bytes de datos en el buffer:
        for (j=0; j<3; j++)
        {
                for (k=0; k<3; k++)
                {
                        Buffer_RS[Fin_Buf_RS] = adcs[j].ch[k];
                        Fin_Buf_RS = (Fin_Buf_RS + 1) % Tamano_Buffer_RS;
                        ContBytesTramaDatos++;
                }
        }
        if (ContBytesTramaDatos == BYTES_TRAMADATOS)
        {
                ContBytesTramaDatos = 0;
                TramasListas++;
                if (TramasListas >= TRAMAS_BUFFER)
                {
                        Flag.ErrorSatBuffer = TRUE;
                }
                else
                {
                        Flag.TramaLista = TRUE;
                }
        }

        // 5.- Si está sincronizando, lee el codigo lento:
        /* Cuando est  sincronizando lee la señal de código lento en las centésimas 15 (como comprobación,
        siempre debe estar alta) y 50 (si es un 0 debe estar baja y si es un 1, alta).*/
        if(Flag.Sincronizando) LeerCodigoLento();

        // 6.- Refresca el watchdog
        RefrescaWD();

        outportb(0x20, 0x20);       // Comando de fin de interrupción, enviado a los dos controladores de
        outportb(0xA0, 0x20);       //interrupciones.
}

void InicializarIRQ7(void)
{
        disable();
        GuardaIntIRQ7 = getvect(0x0F);                  // Desvía el vector de interrupción
        setvect(0x0F, Trat_IRQ7);
        outportb(portpar2, inportb(portpar2) | 0x10); // Habilita IRQ7 en el registro 2 del puerto paralelo
        outportb(0x21, inportb(0x21) & 0x7F);         // Habilita IRQ7
        enable();
}
```

```
void ReponerVectoresIRQ7(void)
{
        disable();
        setvect(0x0F, GuardaIntIRQ7);
        outportb(portpar2, inportb(portpar2) & 0xEF);       // Pone a 0 el bit 4 del  tercer registro del
                                                            //puerto paralelo (IRQ EN),  para deshabilitar
                                                            //la interrupción.
        outportb(0x21, inportb(0x21)|0x80);                 // Deshabilita IRQ7
        enable();
}

/*-------------------------------------------------------------------------------------------------------------------
  FUNCIONES PARA LA INICIALIZACIÓN Y TRATAMIENTO DEL PUERTO SERIE
-------------------------------------------------------------------------------------------------------------------*/

//   INICIALIZACIÓN DEL PUERTO SERIE.
void InicializarComunicacionSerie (Puerto,Velocidad,Paridad,BitsDatos,BitsStop)

int Puerto,Velocidad,Paridad,BitsDatos,BitsStop;
{
        int IRQ;
        int BasePuerto, i;

        if (Puerto == 0 || Puerto == 2)         IRQ = 0;
        else       IRQ = 1;

        // Configura el puerto hasta que el registro de estado de línea esté a cero
        while      (ConfigurarPuertoSerie (Puerto,Velocidad,Paridad,BitsStop,BitsDatos));

        if (IRQ == 0)
        {
                if (!IRQ4Activo)
                {
                        disable();
                        GuardaIntIRQ4 = getvect (0x0c-IRQ);
                        setvect (0x0c-IRQ, Trat_IRQ4);
                        enable();
                        IRQ4Activo = TRUE;
                }
        }
        else
        {
                if (!IRQ3Activo)
                {
                         disable();
                         GuardaIntIRQ3 = getvect (0x0c-IRQ);
                         setvect (0x0c-IRQ, Trat_IRQ3);
                         enable();
                         IRQ3Activo = TRUE;
                }
        }
        if (!RelojActivo)
        {
                disable();
```

```
                GuardaIntReloj = getvect (0x1C);
                setvect (0x1C, Reloj);
                enable();
                RelojActivo = TRUE;
        }
        disable();
        if (IRQ == 0) outportb (0x21,inportb(0x21) & 0xef);
        else outportb (0x21,inportb(0x21) & 0xf7);

        outportb (RS232[Puerto].ier,0x01);
        outportb (RS232[Puerto].mcr,0x08);

        // Limpia posibles interrupciones pendientes leyendo los registros
        BasePuerto = peek(0x40,Puerto << 1);
        for (i=0; i<6; i++) inportb(BasePuerto+i);
        Princ_Buf_RS = 0;
        Fin_Buf_RS = 0;
        TimeOut[Puerto] = 0;
        enable();
        PuertoActivo[Puerto] = TRUE;
}

/*   INICIALIZACIÓN DEL PUERTO SERIE. Se programan directamente los registros   de la pastilla, para
poder lograr una mayor velocidad. */
int ConfigurarPuertoSerie (Puerto,Velocidad,Paridad,BitsStop,BitsDatos)

int Puerto,Velocidad,Paridad,BitsStop,BitsDatos;
{
        int BasePuerto;
        unsigned char Control;
        union REGS rin,rout;
        char hecho;
        // Registros divisores de frecuencia
        static unsigned char
        DLL[13] =
        {
                0x17,0x80,0xc0,0x60,0x40,0x30,0x20,0x10,0x0c,0x06,0x03,0x02,0x01
        },
        DML[13] =
        {
                0x04,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
        },
        // Para definición de los bits de datos en el registro de control
        DATA[2] =
        {
        // 7 bits, 8 bits
                 0x02,0x03
        },
        // Para definición de los bits de stop en el registro de control
        STOP[2] =
        {
        // 1 bit, 2 bits
                0x00,0x04
        },
        // Para definición de la paridad en el registro de control
        PARITY[3] =
```

```
        {
        // NINGUNA, IMPAR, PAR
                0x00, 0x08,  0x18
        };

        BasePuerto = peek(0x40,Puerto << 1);
        outportb (BasePuerto+3,0x80);
        outportb (BasePuerto+1,DML[Velocidad]);
        outportb (BasePuerto,DLL[Velocidad]);
        Control = DATA[BitsDatos] | STOP[BitsStop] | PARITY[Paridad];
        outportb (BasePuerto+3,Control);
        outportb (BasePuerto+1,0);

        // Comprueba si se ha inicializado correctamente el puerto
        rin.x.dx = Puerto;
        rin.h.ah = 3;
        int86 (0x14,&rin,&rout);

        // Devuelve TRUE si no detecta error en el estado de línea y FALSE si hay error
        hecho= (rout.h.ah & 0x1C);
        return ((int) hecho);
}

/*   FINALIZA LA COMUNICACION SERIE. Se restauran los vectores y se inhiben las interrupciones serie.
Finaliza la comunicaci¢n serie de TODOS LOS PUERTOS.*/
void FinalizarComunicacionSerie()
{
        int Com_RS;

        // Esperar hasta que la comunicación se ha completado
        for (Com_RS=0; Com_RS<NUM_PORTS; Com_RS++)
                if (PuertoActivo[Com_RS])
                        while ((inportb (RS232[Com_RS].lsr) & 0x20) != 0x20);
        delay (500);
        outportb (0x21,inportb(0x21) | 0x18);
        for (Com_RS=0; Com_RS<NUM_PORTS; Com_RS++)
        {
                if (PuertoActivo[Com_RS])
                {
                        outportb (RS232[Com_RS].lcr,inportb(RS232[Com_RS].lcr) & 0x7f);
                        outportb (RS232[Com_RS].ier,0);
                        outportb (RS232[Com_RS].mcr,0);
                }
                PuertoActivo[Com_RS] = FALSE;
        }

        if (IRQ4Activo)
        {
                disable();
                setvect (0x0c,GuardaIntIRQ4);
                enable();
        }

        if (IRQ3Activo)
        {
                disable();
```

```
                setvect (0x0c-1,GuardaIntIRQ3);
                enable();
        }

        if (RelojActivo)
        {
                disable();
                setvect (0x1c,GuardaIntReloj);
                enable();
        }
}

/*-------------------------------------------------------------------------------------------------------------
  FUNCIONES PARA LA RECEPCIÓN Y TRANSMISIÓN DE TRAMAS
-------------------------------------------------------------------------------------------------------------*/

/*  IDENTIFICACIÓN DE LA TRAMA RECIBIDA. Se toma el siguiente carácter del buffer y se activan los
semáforos correspondientes según su posición en la trama. */

void LeerTrama(void)
{
        // Se leen los caracteres que se reciben hasta que se identifica una trama o hay error de recepción
        do
        {
                if (Princ_Buf_Recepcion != Fin_Buf_Recepcion)
                {
                        if (AnalizarCaracter() == MAL) Flag.ErrorRecepcionTrama = TRUE;
                }
        }
        while ((!Flag.TramaIdentificada)&&(!Flag.ErrorRecepcionTrama)&&(!Salir()));
        /* Solo sale de aquí cuando identifica una trama completa o detecta un error en la recepción, siempre
        y cuando la estación esté activa. */

        if (Flag.TramaIdentificada) Flag.TramaIdentificada = FALSE;
        else    // Tiene que ser un error en la recepción
        {
                Flag.ErrorRecepcionTrama =      FALSE;
                //printf("\nError de recepción de trama");
                Fin_Buf_Recepcion++;                    //Ignora la trama mala
                Princ_Buf_Recepcion++;
                Semaf.Cabecera          =       FALSE; // Pone a FALSE todos los semáforos de
                Semaf.RecibiendoTrama   =       FALSE; //transmisión y los flags que podrían haberse
                Semaf.CambioGanancia    =       FALSE; //modificado al identificar caracteres de una
                Semaf.EsperandoFinTrama =       FALSE; //trama mala.
                Flag.NuevaGanancia      =       FALSE;
                Flag.Reenvio            =       FALSE;
                Flag.MandarTrama        =       FALSE;
                Flag.TramaIdentificada  =       FALSE;
        }
}

int AnalizarCaracter(void)
{
        CaracterAIdentificar = Buffer_Recepcion[Princ_Buf_Recepcion];
        Princ_Buf_Recepcion = (Princ_Buf_Recepcion + 1) % Tamano_Buf_Recepcion;
```

```
//¨¿Es mi código de tarjeta? ¿Es un carácter para otra tarjeta?
if (!Semaf.RecibiendoTrama)
{
        if (CaracterAIdentificar ==  MiCodigoTarjeta)
        {
                Semaf.RecibiendoTrama   = TRUE;
                Semaf.Cabecera          = TRUE;
        }
        return(OK);     // Devuelve OK sea o no mi código de tarjeta. Es decir, asumimos que, si
                        //no lo es, es un carácter para otra tarjeta, aunque no se comprueba cual.
}

// ¿Es el segundo carácter de la trama?
if (Semaf.Cabecera)
{
        Semaf.Cabecera                  = FALSE;
        Semaf.EsperandoFinTrama         = TRUE;
        switch(CaracterAIdentificar)
        {
                case TIPO_TRAMA_PETICION:
                        Flag.MandarTrama = TRUE;
                        return(OK);

                case TIPO_TRAMA_CAMBIOGANANCIA:
                        Semaf.CambioGanancia      = TRUE;
                        Semaf.EsperandoFinTrama  = FALSE;       // Las tramas de cambio de
                        return(OK);                             //ganancia son las únicas con
                                                                //4 caracteres.
                case TIPO_TRAMA_REENVIO:
                        Flag.Reenvio = TRUE;
                        return(OK);

                case TIPO_TRAMA_DESCONEXION:
                        Flag.EstacionActiva = FALSE;
                        return(OK);

                case TIPO_TRAMA_RECONEXION:
                        Flag.EstacionActiva = TRUE;
                        return(OK);

                case TIPO_TRAMA_SINCRONIZACION:
                        if(!Flag.Sincronizando)
                        SincronizarRelojSoftware();
                        return(OK);

                default:
                        return(MAL);
        }
}

// ¿Es el tercer carácter de una trama de cambio de ganancia? (nueva ganancia)
if (Semaf.CambioGanancia)
{
        Semaf.CambioGanancia    = FALSE;
        if (CaracterAIdentificar != gain)     // Sólo se van a reinicializar los CADs si la ganancia
        {                                     //ganancia recibida es distinta de la actual.
```

```
                        gain  = CaracterAIdentificar;
                        Flag.NuevaGanancia      = TRUE;   // Se pone a FALSE en el programa ppal.
                }
                //else printf("\nGanancia recibida igual a la actual: %i %i", gain, CaracterAIdentificar);
                Semaf.EsperandoFinTrama = TRUE;
                return(OK);
        }

        // ¿Es el último carácter de la trama?
        if (Semaf.EsperandoFinTrama)
        {
                if (CaracterAIdentificar == FIN_TRAMA)
                {
                        Semaf.RecibiendoTrama   = FALSE;
                        Flag.TramaIdentificada = TRUE;
                        return(OK);
                }
                return(MAL);      // Si no es el carácter de fin de trama, hay error.
        }

        // Si no ha entrado en ninguno de los 'if', hay error.
        return(MAL);
}

void DesactivarDE(int Com_RS)
{
        while ((inportb (RS232[Com_RS].lsr) & 0x60) != 0x60);// Esperamos hasta que los registros de
                                                // retención y de desplazamiento de transmisión están vacíos.
        outportb(RS232[Com_RS].mcr, inportb(RS232[Com_RS].mcr) | 0x01);// Pone a 1 la señal 'Data
                                                //Terminal Ready' de COM1, que es la que controla DE
}

void ActivarDE(int Com_RS)
{
        outportb(RS232[Com_RS].mcr, inportb(RS232[Com_RS].mcr) & 0xFE);// Pone a 0 la señal 'Data
                                                //Terminal Ready' de COM1, que es la que controla DE.
}

// Envía un carácter al puerto RS232.
void MandarCaracter (int Com_RS, char Param)
{
        while ((inportb (RS232[Com_RS].lsr) & 0x20) != 0x20);
        outportb (RS232[Com_RS].thr,Param);
        Error_RS[Com_RS] = 0;
}

void MandarTramaErrInic(void)
{
        ActivarDE(COM1);
        MandarCaracter(COM1, MiCodigoTarjeta);
        MandarCaracter(COM1, TIPO_TRAMA_ERROR);
        MandarCaracter(COM1, ERROR_INIC_ADCs);
        MandarCaracter(COM1, FIN_TRAMA);
        UltimaTramaEnviada = ERROR_INIC_ADCs;          // Por si hay que reenviarla
        DesactivarDE(COM1);
}
```

```
void MandarTramaErrSat(void)
{
	ActivarDE(COM1);
	MandarCaracter(COM1, MiCodigoTarjeta);
	MandarCaracter(COM1, TIPO_TRAMA_ERROR);
	MandarCaracter(COM1, ERROR_SATURACION_BUFFER);
	MandarCaracter(COM1, FIN_TRAMA);
	DesactivarDE(COM1);
	UltimaTramaEnviada = ERROR_SATURACION_BUFFER;  // Por si hay que reenviarla
}

void MandarTramaErrSinc(void)
{
	ActivarDE(COM1);
	MandarCaracter(COM1, MiCodigoTarjeta);
	MandarCaracter(COM1, TIPO_TRAMA_ERROR);
	MandarCaracter(COM1, ERROR_SINCRONIZACION);
	MandarCaracter(COM1, FIN_TRAMA);
	DesactivarDE(COM1);
	UltimaTramaEnviada = ERROR_SINCRONIZACION;	// Por si hay que reenviarla
}

void MandarTramaDatos(void)
{
	unsigned int i;

	ActivarDE(COM1);
	MandarCaracter(COM1, MiCodigoTarjeta);
	MandarCaracter(COM1, TIPO_TRAMA_DATOS);
	InicioDatos = Princ_Buf_RS;	// Por si hay que reenviarla
	for (i=0; i<BYTES_TRAMADATOS; i++)
	{
		MandarCaracter(COM1, Buffer_RS[Princ_Buf_RS]);
		//if(Princ_Buf_RS == InicioDatos +1) printf("%u ", Buffer_RS[Princ_Buf_RS]);
		Princ_Buf_RS = (Princ_Buf_RS + 1) % Tamano_Buffer_RS;
	}
	MandarCaracter(COM1, MiCodigoTarjeta);
	MandarCaracter(COM1, FIN_TRAMA);
	DesactivarDE(COM1);
	UltimaTramaEnviada = TIPO_TRAMA_DATOS;
	TramasListas --;
	if(TramasListas == 0) Flag.TramaLista = FALSE;
}

void ReenviarUltimaTrama(void)
{
	switch (UltimaTramaEnviada)	// Se comprueba el tipo de la última trama enviada y se vuelve a
					//mandar. No se tocan los punteros porque se dejaron en su sitio
					//al mandar la trama por primera vez.
	{
		case ERROR_INIC_ADCs:
			MandarTramaErrInic();
			break;
```

```
                case ERROR_SATURACION_BUFFER:
                        MandarTramaErrSat();
                        break;

                case ERROR_SINCRONIZACION:
                        MandarTramaErrSinc();
                        break;

                case TIPO_TRAMA_DATOS:
                        Princ_Buf_RS = InicioDatos;
                        TramasListas++;         //Para que no dé error de saturación del buffer
                        MandarTramaDatos();
                        break;
        }
}

/*-------------------------------------------------------------------------------------------------------------
  FUNCIONES PARA LA GESTIÓN DE ERRORES Y DE TRAMAS RECIBIDAS
-------------------------------------------------------------------------------------------------------------*/
int Error(void)
{
        if ((Flag.MandarErrorADCs)||(Flag.ErrorInicADCs)||(Flag.ErrorSatBuffer)
                ||(Flag.ErrorSincronizacion)||(Flag.Reenvio))
                return(TRUE);
        else
                return(FALSE);
}

void GestionErrores(void)
{
        /* Si hay error de inicialización de los CADs no se comunica inmediatamente, sino que se marca
        para hacerlo en la próxima pasada. Así se evita que el PC nodal llegue al TimeOut si el fallo ha sido
        en el arranque del programa.*/
        if (Flag.MandarErrorADCs)       // En caso de varios errores simultáneos éste se sirve primero,
        {                               //porque va a originar la desconexión.
                Flag.MandarErrorADCs = FALSE;
                MandarTramaErrInic();
        }
        if (Flag.ErrorInicADCs)         // En caso de varios errores simultáneos éste se sirve primero,
        {                               //porque va a originar la desconexión.
                Flag.ErrorInicADCs = FALSE;
                Flag.MandarErrorADCs = TRUE;
        }
        else
        {
                if (Flag.ErrorSatBuffer)
                {
                        Flag.ErrorSatBuffer = FALSE;
                        //ContBytesTramaDatos = 0;
                        TramasListas = 0;
                        //InicioDatos = 0
                        Flag.TramaLista = FALSE;
                        //Princ_Buf_RS = Fin_Buf_RS = 0;
                        MandarTramaErrSat();
                }
                else    // Hay que reenviar la última trama
```

```
                {
                        if (Flag.Reenvio)
                        {
                                Flag.Reenvio = FALSE;
                                ReenviarUltimaTrama();
                        }
                        else
                        {
                                if (Flag.ErrorSincronizacion)
                                {
                                        Flag.ErrorSincronizacion = FALSE;
                                        MandarTramaErrSinc();
                                }
                        }
                }
        }
}

void CambiarGanancia(void)
{
        SincronizarRelojSoftware();         // Se sale y arranca con la nueva ganancia
}

void InicializarRelojSoftware(void)         // Activa los flags de inicio de la sincronización y espera a que
                                            //ésta termine.
{
        printf("\n Inicializando reloj software...");
        Flag.Muestra1PulsoSinc1  = TRUE;
        Flag.Sincronizando       = TRUE;
        Flag.Esperando27_0s      = TRUE;
        do
        {
        }
        while(Flag.Sincronizando);
        printf("\n Fin de la inicialización del reloj software");
        printf("\n Hora y fecha adquiridas: %02i:%02i:%02i.%02i del día %i del año %i\n",
                                                hora, minuto, segundo, fraccion, dia.total, year );
}

void SincronizarRelojSoftware(void)
{
        /* Para sincronizar el reloj software se reinicia el PC con la ganancia actual. Para ello se guarda la
        ganancia en el fichero GAIN.BIN y se sale del programa, dejando actuar el WD. Teniendo en cuenta
        que los PCs de las estaciones arrancan desde EPROM y no tienen disco duro, el fichero GAIN.BIN
        se escribe en disco RAM. */
        FILE *FichGain;
        if (( FichGain = fopen("C:\GAIN.BIN", "wb")) == NULL)
        {
                printf("\n No se puede abrir C:\\GAIN.BIN para escritura");
        }
        else
        {
                putc(gain, FichGain);
                printf("\n Ganancia escrita en el fichero C:\\GAIN.BIN: %i", gain);
                fclose(FichGain);
        }
```

```
        Flag.Fin = TRUE;// Para salir y sincronizar
}
```

/* LECTURA DEL CODIGO LENTO DE TIEMPO REAL.- El formato del código lento es:

Seg.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Bit	0	0	m	m	m	m	m	m	h	h	h	h	H	d	d	d	d	d	d	d
Seg.	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
Bit	d	d	a	a	a	a	a	a	a	0	0	0	0	0	0	0	0	0	0	0
Seg.	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
Bit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

siendo:
m = cód. de minuto / h = cód. de hora / d = cód. de dia / a = cód. de año

En cada una de las variables se recibe primero el bit menos significativo y el más significativo el último.

Para decodificar la hora se empieza esperando una cadena de 27 o más ceros seguida de cuatro unos y otros dos ceros. Luego se toman los bits de minuto (m), hora (h), día juliano (d) y año (a). Por último se espera de nuevo la cadena de 27 ceros y cuatro unos para actualizar las variables de tiempo en el flanco del segundo cero del siguiente minuto. Por eso el RTC del oscilador patrón tiene que estar adelantado un minuto.*/

```
void LeerCodigoLento(void)
{
        if (fraccion == 15)
        {
                if(!(inportb(portpar1) & 0x08))
                {
                        Flag.ErrorSincronizacion  = TRUE;
                        // Cuando hay error se ponen todos los flags a FALSE para que no entre en
                        //ninguno de los siguientes 'if'.
                        Flag.Sincronizando        = FALSE;
                        Flag.Muestra1PulsoSinc1   = FALSE;
                        Flag.Sincronizando        = FALSE;
                        Flag.Esperando27_0s       = FALSE;
                        Flag.Esperando4_1s        = FALSE;
                        Flag.RecibiendoCodigo     = FALSE;
                        Flag.ActualizarVars       = FALSE;
                        Flag.SincronizacionOK     = FALSE;
                }
        }
        if (fraccion == 50)
        {
                if(++ContSegundosCL == 180)      // Si pasan más de 3 minutos sin sincronizar, error
                {
                        Flag.ErrorSincronizacion  = TRUE;
                        // Cuando hay error se ponen todos los flags a FALSE para que no entre en
                        //ninguno de los siguientes 'if'
                        Flag.Sincronizando        = FALSE;
                        Flag.Muestra1PulsoSinc1   = FALSE;
                        Flag.Sincronizando        = FALSE;
                        Flag.Esperando27_0s       = FALSE;
                        Flag.Esperando4_1s        = FALSE;
                        Flag.RecibiendoCodigo     = FALSE;
                        Flag.ActualizarVars       = FALSE;
```

```
		Flag.SincronizacionOK	= FALSE;
	}
	if(Flag.Esperando27_0s)
	{
		if(!(inportb(portpar1) & 0x08))		// Si es 0 se aumenta el contador de 0s
		{
			if (++ContCeros >= 27)
			{
				Flag.Esperando27_0s	= FALSE;
				Flag.Esperando4_1s	= TRUE;
			}
		}
		else ContCeros = 0;		// Si es 1 se resetea el contador de 0s
	}
	else	// Si ya ha detectado la cadena de 27 o más ceros...
	{
		if(Flag.Esperando4_1s)
		{
			// Si es 0 no se hace nada (no tiene que ser error porque el número de
			//ceros seguidos puede ser mayor de 27)
			if(inportb(portpar1) & 0x08)		// Si es uno se aumenta el contador de
			{					//1s y se comprueba si ha llegado a 4
				if(++ContUnos == 4)
				{
					Flag.Esperando4_1s = FALSE;
					Flag.RecibiendoCodigo = TRUE;
					ContBitsCL = 0;
				}
			}
		}
		else	// Si ya ha detectado la cadena de cuatro unos...
		{
			if(Flag.RecibiendoCodigo)
			{
				// El incremento del ContBitsCL lo ponemos después del switch,
				//para que entre en el 'case 0:'
				switch(ContBitsCL)
				{
					case 0:	// El reloj *software* se actualiza en segundo 0
						//del siguiente minuto
						if (Flag.ActualizarVars)
						{
							Flag.ActualizarVars = FALSE;
							Flag.Sincronizando = FALSE;
							//Ahora acaba la sincronizaci¢n
							Flag.SincronizacionOK = TRUE;
							segundo = 0;
							minuto = MinutoCL;
							hora = HoraCL;
							dia.total = DiaCL;
							year = YearCL;
							if ((YearCL % 4) == 0)
								dias_year = 366;
							else
								dias_year = 365;
						}
```

```
case 1:  // En los segundos 0 y 1 el código debe ser 0
         if(inportb(portpar1) & 0x08)
         {
                  Flag.ErrorSincronizacion = TRUE;
                  // Cuando hay error se ponen todos
                  //los flags a FALSE para que no entre
                  //en ninguno de los siguientes 'if'
                  Flag.Sincronizando        = FALSE;
                  Flag.Muestra1PulsoSinc1   = FALSE;
                  Flag.Sincronizando        = FALSE;
                  Flag.Esperando27_0s       = FALSE;
                  Flag.Esperando4_1s        = FALSE;
                  Flag.RecibiendoCodigo     = FALSE;
                  Flag.ActualizarVars       = FALSE;
                  Flag.SincronizacionOK     = FALSE;
         }
         break;

case 2:  // LSBit del minuto
         Mascara = 0x01;
         MinutoCL = 0;
case 3:
case 4:
case 5:
case 6:
case 7:  // MSBit del minuto
         if (inportb(portpar1) & 0x08)
                  MinutoCL |= Mascara;
         Mascara <<= 1;   // Desplaza la máscara para
                          //el siguiente segundo.
         break;

case 8:  // LSBit de la hora
         Mascara = 0x01;
         HoraCL = 0;
case 9:
case 10:
case 11:
case 12: // MSBit de la hora
         if (inportb(portpar1) & 0x08)
                  HoraCL |= Mascara;
         Mascara <<= 1;
         break;

case 13:// LSBit del d¡a
         Mascara = 0x01;
         DiaCL = 0;
case 14:
case 15:
case 16:
case 17:
case 18:
case 19:
case 20:
case 21: // MSBit del d¡a
         if (inportb(portpar1) & 0x08)
```

```
                                        DiaCL |= Mascara;
                                Mascara <<= 1;
                                break;

                        case 22: // LSBit del a¤o
                                Mascara = 0x01;
                                YearCL = 0;
                        case 23:
                        case 24:
                        case 25:
                        case 26:
                        case 27:
                        case 28: // MSBit del a¤o
                                if (inportb(portpar1) & 0x08)
                                        YearCL |= Mascara;
                                Mascara <<= 1;
                                break;
                        case 29:
                                // El bit 29 tiene que ser un cero
                                if (inportb(portpar1) & 0x08)
                                {
                                        Flag.ErrorSincronizacion = TRUE;
                                        // Cuando hay error se ponen todos
                                        //los flags a FALSE para que no entre
                                        //en ninguno de los siguientes 'if'
                                        Flag.Sincronizando      = FALSE;
                                        Flag.Muestra1PulsoSinc1 = FALSE;
                                        Flag.Sincronizando      = FALSE;
                                        Flag.Esperando27_0s     = FALSE;
                                        Flag.Esperando4_1s      = FALSE;
                                        Flag.RecibiendoCodigo   = FALSE;
                                        Flag.ActualizarVars     = FALSE;
                                        Flag.SincronizacionOK   = FALSE;
                                }
                                else
                                {
                                        // Pone los flags para que empiece de
                                        //nuevo a contar ceros.
                                        Flag.RecibiendoCodigo   = FALSE;
                                        Flag.ActualizarVars     = TRUE;
                                        Flag.Esperando27_0s     = TRUE;
                                        ContCeros = 1;
                                        ContUnos = 0;
                                }
                                break;
                        }       // Fin del switch
                        ++ContBitsCL;
                }
        }
    }
}
```

```
/*-------------------------------------------------------------------------------------------------------------------
  OTRAS FUNCIONES
-------------------------------------------------------------------------------------------------------------------*/
void RefrescaWD(void)
{
        outportb(PUERTO_WD, TIEMPO_WD);
}


/*  ESPERA DE ACTIVACIÓN. Si la estación no está activa, se queda esperando aquí, identificando las
tramas que le llegan (sin tratamiento de errores) y sin mandar ninguna respuesta hasta que recibe una trama de
reconexión, que la devuelve a su  funcionamiento normal). */
void EsperaActivacion(void)
{
        do
        {
                if (Princ_Buf_Recepcion != Fin_Buf_Recepcion) AnalizarCaracter();
        }
        while(((!Flag.EstacionActiva)||(!Flag.TramaIdentificada))&&(!Salir()));

        // Cuando se identifica una trama de reconexión, se reinicializan todos los semáforos, flags y
        //punteros de los dos buffers para volver al funcionamiento normal.
        disable();          // Esto se hace deshabilitando las interrupciones para que la de toma de datos no
                            //nos cambie los punteros.
        Semaf.RecibiendoTrama           = FALSE;
        Semaf.Cabecera                  = FALSE;
        Semaf.CambioGanancia            = FALSE;
        Semaf.EsperandoFinTrama         = FALSE;
        Flag.TramaIdentificada          = FALSE;
        Flag.TramaLista                 = FALSE;
        Flag.NuevaGanancia              = FALSE;
        Flag.MandarTrama                = FALSE;
        Flag.ErrorRecepcionTrama        = FALSE;
        Flag.ErrorSatBuffer             = FALSE;
        Flag.Reenvio                    = FALSE;
        Fin_Buf_RS              = Princ_Buf_RS          = 0;
        Princ_Buf_Recepcion     = Fin_Buf_Recepcion     = 0;
        TramasListas            = 0;
        ContBytesTramaDatos     = 0;
        enable();
}

void LeerCodigoTarjeta(void)
{
        unsigned char registro, MiEstacion;

        outportb(RS232[COM2].mcr, 0x01);        // Saca un 1(+8V) por el bit 0 (línea DTR) y un 0 (-8V)
                                                //por el bit 1 (línea RTS) de COM2.
        delay(500);         // Para que se estabilice la tensión de entrada
        registro = ~(inportb(RS232[COM2].msr));
        MiEstacion = (((registro & 0x10) >> 4) | ((registro & 0x80) >> 6) | ((registro & 0x20) >> 3));
        printf("\nEstaci¢n número %i", MiEstacion);
        MiCodigoTarjeta = CodigoTarjeta[MiEstacion];
        printf("\nCódigo de tarjeta: 0x%X\n", MiCodigoTarjeta);
}
```

```
unsigned char Salir(void)
{
	FILE	*FichGain;

	outportb(RS232[COM2].mcr, 0x01);		// Saca un 1(+8V) por el bit 0 (línea DTR) y un 0(-8V)
							//por el bit 1 (línea RTS) de COM2.
	if((!(inportb(RS232[COM2].msr) & 0x40))||(Flag.Fin))
	{
		if (!Flag.Fin)	// Si se ha pulsado el botón, borra el fichero C:\GAIN.BIN (si existe) para
		{			//que no vuelva a arrancar el programa	.
			if (( FichGain = fopen("C:\GAIN.BIN", "rb")) == NULL)
			{
				printf("\n Función Salir(): No existe C:\\GAIN.BIN ");
			}
			else
			{
				fclose(FichGain);
				remove("C:\GAIN.BIN");
			}
		}
		return(TRUE);
	}
	else return(FALSE);
}
```

```
/*-------------------------------------------------------------------------------------------------------------------
  PROGRAMA PRINCIPAL
-------------------------------------------------------------------------------------------------------------------*/
void main(void)
{
        clrscr();
        printf ("\n Programa ESTACION\n");          // Los printf son para monitorizar el funcionamiento
                                                    //durante la etapa de desarrollo
        InicializarComunicacionSerie(COM1,BAUD115200,NINGUNA,LON8,STOP1);
        LeerCodigoTarjeta();
        LeerGanancia();
        iniciacion();
        sincronismo();
        DesactivarDE(COM1);     // Inicialmente se deshabilita la transmisión y se habilita la recepción, para
                                //las tramas de petición de trama.
        InicializarIRQ7();
        InicializarRelojSoftware();
        do
        {
                if (!Flag.EstacionActiva) EsperaActivacion();
                LeerTrama();    // Lee hasta que recibe una trama o hay un error de recepción, que ignora
                if (Flag.MandarTrama)
                {
                        Flag.MandarTrama = FALSE;
                        if (Error()) GestionErrores();          // Se le da preferencia a la gestión de errores
                        else                                    //frente al envío de datos.
                        {
                                if (Flag.TramaLista) MandarTramaDatos();
                        }
                }
                else
                {
                        if (Flag.NuevaGanancia)
                        {
                                Flag.NuevaGanancia = FALSE;
                                CambiarGanancia();
                        }
                }
        }
        while(!Salir());
        ReponerVectoresIRQ7();
        FinalizarComunicacionSerie();
        DesactivarDE(COM1);     // Para dejar la línea serie en alta impedancia
}
```

ANEXO I.A.2.- PROGRAMA DE LOS PCs NODALES

Fichero: PC_NODAL.C

/* Programa para los PCs nodales. Las funciones para la gestión del puerto serie son las mismas que en el programa de las estaciones y no se han incluido por brevedad. Tampoco se incluye en el listado la librería de gestión de ficheros con *file-locking* (NETFILEC.C), que se tomó de Tischer y Jennrich, 1996. El programa principal se encuentra al final */

```
/*****************************************************************************
  Macros
*****************************************************************************/
// DEFINICIONES GENERALES
#define MAL                     1
#define OK                      0
#define NUM_PORTS               4
#define NUMERO_ESTACIONES       4
#define NUM_TICKS               2       // Tiempo de espera de respuesta de la estación
#define NUM_TICKS_2             10      // Tiempo de espera al último carácter de la trama
                                        //desde la recepción del primero
#define BYTES_CTROL             4       // Cód. tarjeta inicial + tipo trama + cód. Tarjeta final +
                                        //fin de trama
#define BYTES_DATOS             908     // Bytes de ganancia + cabecera temporal + datos
#define BYTES_TRAMA_DATOS       BYTES_CTROL+BYTES_DATOS
#define BYTES_TRAMA_ERROR       4
#define TRAMAS_BUFFER           35
#define TAMANO_FICHERO          120     // Tamaño en minutos de los ficheros de datos
#define TRAMAS_FICHERO          TAMANO_FICHERO * 60
#define Tamano_Buffer_RS        TRAMAS_BUFFER * BYTES_TRAMA_DATOS
#define INT_TIMER               18      // Núm. interrupciones por segundo del timer (aprox.)
#define TIEMPO_DESCONEXION      600     // Segundos de espera para desconexión de estaciones
#define Tamano_Buffer_Datos     60*BYTES_DATOS
#define PUERTO_WD               0x214
#define TIEMPO_WD               0x87 // 0x81, 0x82, ...0x87 = 6, 12, ...42 segundos

// POSIBLES ESTADOS DE LAS ESTACIONES
#define  NO_OPERATIVA           '0'
#define  REGISTRANDO            '1'
#define  DETECTANDO             '2'

// CODIGOS DE TRAMA
#define TIPO_TRAMA_DATOS                0xC3
#define TIPO_TRAMA_ERROR                0x3C
#define TIPO_TRAMA_PETICION             0x2B
#define TIPO_TRAMA_CAMBIOGANANCIA       0xB2
#define TIPO_TRAMA_REENVIO              0xD4
#define TIPO_TRAMA_DESCONEXION          0xE5
#define TIPO_TRAMA_RECONEXION           0x5E
#define TIPO_TRAMA_SINCRONIZACION       0xF6
#define ERROR_INIC_ADCs                 0xA0
#define ERROR_SATURACION_BUFFER         0xB1
#define ERROR_SINCRONIZACION            0xD3
```

```
#define ERROR_RECEPCION_TRAMA       0xF8
#define DESCONEXION_TIMEOUT         0x27
#define FIN_TRAMA                   0xAA

#include <miserie.h>
#include <dir.h>
#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
#include <process.h>
#include "netfilec.c"

/*************************************************************************
  Declaración de variables
*************************************************************************/
unsigned char CodigoTarjeta[NUMERO_ESTACIONES] ={ 0x11, 0x22, 0x44, 0x88 };
unsigned char CaracterAIdentificar;
unsigned int ContadorBytesTrama, ContadorTramasEnFichero[NUMERO_ESTACIONES];
int EstacionActual = 0;
int i;

// Semáforos de recepción: se van activando cuando se identifica un carácter de la trama recibida, para poder
reconocer las tramas completas.
struct semaforos {
        unsigned RecibiendoTrama:       1;
        unsigned Cabecera:              1;
        unsigned EsperandoError:        1;
        unsigned RecibiendoDatos:       1;
        unsigned EsperandoCodTarjeta:   1;
        unsigned EsperandoFinTrama:     1;
        unsigned TramaIdentificada:     1;
} Semaf =
{ FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE };

// Banderas, que se activan en respuesta a distintos eventos.
struct {
        unsigned GrabarTramaBuffDatos:  1;
        unsigned TramaDatos:            1;
        unsigned TramaError:            1;
        unsigned Error:                 1;
        unsigned EscribirError:         1;
        unsigned PedirReenvio:          1;
        unsigned Timer2:                1;
        unsigned Fin:                   1;
} Flag =
{ FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE };

// Banderas de error, que sirven para identificar el tipo de error
struct {
        unsigned TimeOut:               1;
        unsigned SaturacionBuffer:      1;
        unsigned TramaIncompleta:       1;
        unsigned InicializacionADCs:    1;
        unsigned Sincronizacion:        1;
```

```
        unsigned RecepcionTrama:        1;
 }      FlagError =
{ FALSE, FALSE, FALSE, FALSE, FALSE, FALSE };

// Direcciones de los registros de las puertas serie; tomado de 'SERIE.C'.
struct INS8250
        {
         int thr;           // Registro de retención de transmisión
         int rbr;           // Registro de retención de recepción
         int ier;           // Registro de activación de interrupción
         int fcr;           // Registro de control de la FIFO, para la UART 16C550 (no venía en 'SERIE.C')
         int lcr;           // Registro de control de línea
         int mcr;           // Registro de control del módem
         int lsr;           // Registro de la línea de estado
         int msr;           // Registro de estado del módem
        } RS232[NUM_PORTS] =
                {
                 { 0x3f8,0x3f8,0x3f9,0x3fa,0x3fb,0x3fc,0x3fd,0x3fe },
                 { 0x2f8,0x2f8,0x2f9,0x2fa,0x2fb,0x2fc,0x2fd,0x2fe },
                 { 0x3e8,0x3e8,0x3e9,0x3ea,0x3eb,0x3ec,0x3ed,0x3ee },
                 { 0x2e8,0x2e8,0x2e9,0x2ea,0x2eb,0x2ec,0x2ed,0x2ee }
                };

// Variables para la gestión de los puertos serie; tomados de 'serie.h'
unsigned char Buffer_RS[Tamano_Buffer_RS];
int Princ_Buf_RS, Fin_Buf_RS, PtrIdentificar = 0;     // Punteros
int TimeOut[NUM_PORTS],ErrorTOut[NUM_PORTS];
int PuertoActivo[NUM_PORTS] = { FALSE, FALSE, FALSE, FALSE };
int RelojActivo = 0;
void interrupt (*GuardaIntReloj)();                     // Guarda vector del timer

// Variables para la gestión del segundo timer
int TimeOut2[NUM_PORTS], ErrorTOut2[NUM_PORTS];

// Variables para la gestión convencional de ficheros
FILE* fichDatos[NUMERO_ESTACIONES];

// Variables para la gestión de ficheros con file-locking
NFILE ArchvEstaciones, ArchvDatos[NUMERO_ESTACIONES], ArchvComandos, ArchvLog;
char NomFichComandos[50], NomFichEstaciones[50], NomFichLog[50], NomFichDatos[4][50];
char Texto[100];

// Variables para la escritura de errores en el fichero LOG
int PosBufferInicioTrama;
int IdentificadorError[20], EstacionError[20];
unsigned int ErroresAEscribir = 0;

// Variables para el reloj software
unsigned char segundo, minuto, hora, year, segundo_anterior, minuto_anterior, hora_anterior;
union UNintDOSchar{
        unsigned int    total;
        unsigned char   mitad[2];
        } dia;
char ControladorReloj = 0;          // Estación controladora inicial del reloj software

// Variables para la gestión del número de estaciones activas
```

```
char EstacionActiva[NUMERO_ESTACIONES] = { TRUE, TRUE, TRUE, TRUE };
unsigned char NumEstacionesActivas = NUMERO_ESTACIONES;
unsigned int ContadorTimeOut[NUMERO_ESTACIONES];

// Variables para la gestión de los buffers de datos
unsigned char huge BufferDatos[NUMERO_ESTACIONES][Tamano_Buffer_Datos];
int Princ_Buf_Datos[NUMERO_ESTACIONES], Fin_Buf_Datos[NUMERO_ESTACIONES];    // Punteros
int EstacionGrabacion = 0;

// Otras variables:
int NumeroNodal;
unsigned int ContBytes = 0;

/*****************************************************************************
  Declaración de las funciones
*****************************************************************************/
// Funciones para la gestión del timer y del puerto serie:
void interrupt Reloj ();
void CargarTimer (int Com_RS,int NumTicks);
int ErrorTimeOut (int Com_RS);
void CargarTimer2 (int Com_RS,int NumTicks);
int ErrorTimeOut2 (int Com_RS);

// Funciones para la inicialización y tratamiento del puerto serie:
void InicializarComunicacionSerie ();
int ConfigurarPuertoSerie ();
void FinalizarComunicacionSerie();

// Funciones para la recepción y transmisión de tramas:
int DatoDisponible(int Com_RS);
void MandarCaracter (int Com_RS, char Param);
void MeterTramaBuffRecepcion(void);
int AnalizarCaracter(void);
void AnalizarTrama(void);
void DesactivarDE(int Com_RS);
void ActivarDE(int Com_RS);
void MandarTramaGanancia(char num_estacion, char gain);
void MandarTramaPeticion(void);
void MandarTramaDesconexion(unsigned char num_estacion);
void MandarTramaReconexion(unsigned char num_estacion);
void DesconectarEstacion(unsigned char Est);
void ConectarEstacion(unsigned char Est);
void MandarTramaSinc(unsigned char Est);

// Funciones para la gestión de errores:
void GestionaErrores(void);

// Funciones para la gestión de ficheros con file-locking
void AbrirFicheros(void);
void AbrirFichDatos(int Estacion);
void CerrarFichDatos(int Estacion);
void GrabarTramaBuffDatos(int Estacion);
void GrabarBufferFichero(int Estacion);
unsigned char TocaGrabarBufferFichero(void);
void NombrarFicheros(void);
```

```
void AbrirFichLog(int RecLength);
void EscribirLineaFichLog(char Texto[100]);
void CerrarFichLog(void);
void AbrirFichEstaciones(void);
void CerrarFichEstaciones(void);
void MarcarFichEstaciones(void);
int AbrirFichComandos(void);
void CerrarFichComandos(void);
void LeeFichComandos(void);
void AtiendeComandos(void);
int FModus( int Acceso, int Protec );

// Otras funciones:
void RefrescaWD(void);
void CompruebaControladorReloj(void);
unsigned char Salir(void);
void LeerCodigoTarjeta(void);
void InicializarPunteros(void);

/*************************************************************************
  Definición de las funciones
*************************************************************************/
/*-------------------------------------------------------------------------------------------------------------
  FUNCIONES PARA LA GESTIÓN DEL TIMER Y DEL PUERTO SERIE
-------------------------------------------------------------------------------------------------------------*/
void interrupt Reloj ()       // A la rutina de servicio de la interrupción del timer usada es ESTACION.C se le
{                             //ha añadido el código necesario para gestionar un segundo contador de TimeOuts.
        static int i;

        for (i=0; i<NUM_PORTS; i++)
        {
                if (TimeOut[i]) TimeOut[i]--;
                if (!TimeOut[i]) ErrorTOut[i] = TRUE;
                if (TimeOut2[i]) TimeOut2[i]--;
                if (!TimeOut2[i]) ErrorTOut2[i] = TRUE;
        }
        //(*GuardaIntReloj)();
}

void CargarTimer (int Com_RS,int NumTicks)
{
        TimeOut[Com_RS] = NumTicks;
        ErrorTOut[Com_RS] = FALSE;
}

int ErrorTimeOut (int Com_RS)
{
        return (ErrorTOut[Com_RS]);
}

void CargarTimer2 (int Com_RS,int NumTicks)
{
        Flag.Timer2 = TRUE;             // Bandera de Timer2 activado
        TimeOut2[Com_RS] = NumTicks;// Número de ticks que se espera al último carácter de la trama
        ErrorTOut2[Com_RS] = FALSE;
```

```
}

int ErrorTimeOut2 (int Com_RS)
{
        if ((Flag.Timer2)&&(ErrorTOut2[Com_RS]))        // Sólo da error si se ha activado previamente
        {

                ErrorTOut2[Com_RS] = FALSE;
                Flag.Timer2 = FALSE;    // Cuando hay error desactiva el Timer2 para que no dé más
                Flag.Error = TRUE;
                FlagError.TramaIncompleta = TRUE;
                return(TRUE);
        }
        return (FALSE);
}

/* FUNCIONES PARA LA INICIALIZACIÓN Y TRATAMIENTO DEL PUERTO SERIE: Las funciones
'InicializarComunicacionSerie', 'ConfigurarPuertoSerie' y 'FinalizarComunicacionSerie' son las mismas que
se utilizaron en el programa ESTACION.C. No se incluyen por brevedad, pero deben añadirse en la
compilación del programa. */

/*----------------------------------------------------------------------------------------------------------
  FUNCIONES PARA LA RECEPCIÓN Y TRANSMISIÓN DE TRAMAS
----------------------------------------------------------------------------------------------------------*/
/* RECEPCION E INTERPRETACION DE TRAMAS: Las tramas recibidas se meten en el buffer de
recepción. Luego se identifican mediante la función 'AnalizarTrama()', que toma uno a uno los caracteres y
activa los correspondientes semáforos. Además actualiza el reloj *software* cuando toca.*/

int DatoDisponible(int Com_RS)    // Devuelve TRUE si hay dato en el registro de recepción y FALSE en
{                                 //caso contrario.
         if((inportb(RS232[Com_RS].lsr) & 0x01) == 0)
                return FALSE;
        else
                return TRUE;
}

void MandarCaracter (int Com_RS, char Param)
{
        while ((inportb (RS232[Com_RS].lsr) & 0x20) != 0x20);
         outportb (RS232[Com_RS].thr,Param);
}

void MeterTramaBuffRecepcion(void)
{
        unsigned char TipoTrama;
        unsigned int BytesTrama;

        CargarTimer2(COM1, NUM_TICKS_2);    // Tiempo máximo que esperaremos al último carácter
                                            //de la trama a partir de recibir el primero
        PosBufferInicioTrama = Fin_Buf_RS;
        do
        {
                if(DatoDisponible(COM1))
                {
                        ContBytes++;
                        Buffer_RS[Fin_Buf_RS] = (unsigned char) inportb (RS232[COM1].rbr);
```

```
                        if (ContBytes == 2)
                        {
                                TipoTrama = Buffer_RS[Fin_Buf_RS];
                                if (TipoTrama == TIPO_TRAMA_ERROR)
                                        BytesTrama = BYTES_TRAMA_ERROR;
                                else
                                        BytesTrama = BYTES_TRAMA_DATOS;
                        }
                        Fin_Buf_RS = (Fin_Buf_RS + 1) % Tamano_Buffer_RS;
                }
        }
        while ((ContBytes < BytesTrama)&&(!ErrorTimeOut2 (COM1)));
        ContBytes = 0;
}

int AnalizarCaracter(void)
{
        CaracterAIdentificar = Buffer_RS[PtrIdentificar];
        PtrIdentificar = (PtrIdentificar + 1) % Tamano_Buffer_RS;

        // ¿Primer byte de la trama?
        if (!Semaf.RecibiendoTrama)
        {
                if(CaracterAIdentificar == CodigoTarjeta[EstacionActual])
                {
                        Semaf.RecibiendoTrama =  TRUE;
                        Semaf.Cabecera =        TRUE;
                        ContadorBytesTrama = 1;
                        return(OK);
                }
                else return(MAL);
        }

        // ¿2º byte de la trama?
        if (Semaf.Cabecera)
        {
                Semaf.Cabecera   = FALSE;
                switch (CaracterAIdentificar)
                {
                        case TIPO_TRAMA_DATOS:
                                Flag.TramaDatos = TRUE;          // Para actualizar el reloj software al
                                                                 //final de la trama.
                                Semaf.RecibiendoDatos = TRUE;
                                ContadorBytesTrama++;
                                return(OK);

                        case TIPO_TRAMA_ERROR:
                                Flag.TramaError = TRUE;
                                Semaf.EsperandoError = TRUE;
                                return(OK);

                        default:
                                return(MAL);
                }
        }
```

```
// ¿Es un byte de datos de una trama de datos?
if (Semaf.RecibiendoDatos == TRUE)
{
        ContadorBytesTrama++;
        if (ContadorBytesTrama == BYTES_DATOS + 2)
        {
                Semaf.RecibiendoDatos =     FALSE;
                Semaf.EsperandoCodTarjeta =  TRUE;
        }
        return(OK);
}

// ¿3er byte de una trama de error?
if (Semaf.EsperandoError)
{
        Semaf.EsperandoError = FALSE;
        Semaf.EsperandoFinTrama = TRUE;
        switch(CaracterAIdentificar)
        {
                case ERROR_SATURACION_BUFFER:
                        FlagError.SaturacionBuffer = TRUE;
                        return(OK);

                case ERROR_INIC_ADCs:
                        FlagError.InicializacionADCs = TRUE;
                        return(OK);

                case ERROR_SINCRONIZACION:
                        FlagError.Sincronizacion = TRUE;
                        return(OK);

                default:
                        return(MAL);
        }
}

// ¿Penúltimo byte de una trama de datos?
if (Semaf.EsperandoCodTarjeta == TRUE)
{
        Semaf.EsperandoCodTarjeta = FALSE;
        if (CaracterAIdentificar == CodigoTarjeta[EstacionActual])
        {
                Semaf.EsperandoFinTrama = TRUE;
                ContadorBytesTrama++;
                return(OK);
        }
        else return(MAL);
}

// ¿Último byte de la trama?
if (Semaf.EsperandoFinTrama)
{
        Semaf.EsperandoFinTrama = FALSE;
        Flag.Timer2 = FALSE;     // Para deshabilitar el Timer2, ya que la trama ha llegado entera
        if (CaracterAIdentificar == FIN_TRAMA)
        {
```

```
                        Semaf.RecibiendoTrama = FALSE;
                        Semaf.TramaIdentificada = TRUE;

                        // Si la trama ha sido de datos se pone a cero el contador de TimeOuts de esa
                        //estación, para evitar su desconexión.
                        if (Flag.TramaDatos)
                        {
                                Flag.GrabarTramaBuffDatos = TRUE;
                                ContadorBytesTrama++;
                                Flag.TramaDatos = FALSE;
                                ContadorTimeOut[EstacionActual] = 0;

                                // Si además la estación actual es la responsable del reloj software, éste se
                                //actualiza (se hace aqu¡ para estar seguros de que la trama ha llegado
                                //bien).
                                if(EstacionActual == ControladorReloj)
                                {
                                        segundo_anterior = segundo;
                                        minuto_anterior  = minuto;
                                        hora_anterior    = hora;
                                        segundo          = Buffer_RS[PosBufferInicioTrama + 4];
                                        minuto           = Buffer_RS[PosBufferInicioTrama + 5];
                                        hora             = Buffer_RS[PosBufferInicioTrama + 6];
                                        dia.mitad[0]     = Buffer_RS[PosBufferInicioTrama + 7];
                                        dia.mitad[1]     = Buffer_RS[PosBufferInicioTrama + 8];
                                        year             = Buffer_RS[PosBufferInicioTrama + 9];
                                }
                        }
                        // Si ha sido una trama de error, se vuelve a su inicio para no grabarla en el buffer
                        //de datos.
                        if (Flag.TramaError)
                        {
                                Flag.TramaError = FALSE;
                                Flag.Error = TRUE;          // Para entrar en la gestión de errores
                                PtrIdentificar = PosBufferInicioTrama;      // Se posicionan los punteros
                                Fin_Buf_RS     = PosBufferInicioTrama;      //al  principio de la trama,
                        }                                                   //para sobrescribirla.
                        return(OK);
                }
                else return(MAL);
        }
        return(MAL);      // Si no se cumple ninguno de los 'if', algo va mal
}

void AnalizarTrama(void)
{
        do
        {
                if (AnalizarCaracter() == MAL)
                {
                        Flag.Error = TRUE;
                        FlagError.RecepcionTrama = TRUE;
                }
        }
        while ((!Semaf.TramaIdentificada)&&(!FlagError.RecepcionTrama));
        Semaf.TramaIdentificada = FALSE;
```

```
}

/* ENVIO DE TRAMAS Y FUNCIONES QUE LO USAN:*/
void DesactivarDE(int Com_RS)    // Deshabilita transmisión y habilita recepción en la tarjeta RS-485/232
{
        while ((inportb (RS232[Com_RS].lsr) & 0x60) != 0x60);       // Esperamos hasta que los registros
                                                                    //de retención y de desplazamiento de
                                                                    //transmisión están vacíos.
        outportb(RS232[Com_RS].mcr, inportb(RS232[Com_RS].mcr) | 0x01);// Pone a 1 la señal 'Data
                                                                    //Terminal Ready' de COM1, que es
                                                                    //la que controla DE.
}

void ActivarDE(int Com_RS)        // Habilita transmisión y deshabilita recepción en la tarjeta RS-485/232
{
        outportb(RS232[Com_RS].mcr, inportb(RS232[Com_RS].mcr) & 0xFE);
 }

void MandarTramaGanancia(char num_estacion, char gain)
{
        ActivarDE(COM1);
        MandarCaracter(COM1, CodigoTarjeta[num_estacion]);
        MandarCaracter(COM1, TIPO_TRAMA_CAMBIOGANANCIA);
        MandarCaracter(COM1, gain);
        MandarCaracter(COM1, FIN_TRAMA);
        DesactivarDE(COM1);
}

void MandarTramaPeticion(void)
{
        if (Flag.PedirReenvio == TRUE)
        {
                Flag.PedirReenvio               = FALSE;
                PtrIdentificar                  = PosBufferInicioTrama;  // Inicializa los punteros al
                Fin_Buf_RS                      = PosBufferInicioTrama;  //principio de la trama mala.
                Semaf.RecibiendoTrama           = FALSE;                 // Pone a FALSE todos los
                Semaf.Cabecera                  = FALSE;                 //semáforos de recepción.
                Semaf.RecibiendoDatos           = FALSE;
                Semaf.EsperandoCodTarjeta       = FALSE;
                Semaf.EsperandoFinTrama         = FALSE;

                // Manda la trama de petición de reenvío
                ActivarDE(COM1);
                MandarCaracter(COM1, CodigoTarjeta[EstacionActual]);
                MandarCaracter(COM1, TIPO_TRAMA_REENVIO);
                MandarCaracter(COM1, FIN_TRAMA);
                DesactivarDE(COM1);
        }
        else
        {
                do      // Si la estación actual no está activa, pasa a la siguiente:
                {
                        EstacionActual ++;
                        if (EstacionActual == NUMERO_ESTACIONES) EstacionActual = 0;
                }
                while (!EstacionActiva[EstacionActual]);
```

```
                ActivarDE(COM1);
                MandarCaracter(COM1, CodigoTarjeta[EstacionActual]);
                MandarCaracter(COM1, TIPO_TRAMA_PETICION);
                MandarCaracter(COM1, FIN_TRAMA);
                DesactivarDE(COM1);
        }
        CargarTimer(COM1, NUM_TICKS);       // Carga el timer con el nº de ticks de espera de
                                            //respuesta
        while((!DatoDisponible(COM1))&&(!ErrorTimeOut(COM1)));      // Espera hasta que recibe
                                                                    //una trama o hay error de
                                                                    //TimeOut
        if (ErrorTimeOut(COM1))
        {
                Flag.Error =    TRUE;
                FlagError.TimeOut = TRUE;
        }
}

void MandarTramaDesconexion(unsigned char num_estacion)
{
        ActivarDE(COM1);
        MandarCaracter(COM1, CodigoTarjeta[num_estacion]);
        MandarCaracter(COM1, TIPO_TRAMA_DESCONEXION);
        MandarCaracter(COM1, FIN_TRAMA);
        DesactivarDE(COM1);
}

void MandarTramaReconexion(unsigned char num_estacion)
{
        ActivarDE(COM1);
        MandarCaracter(COM1, CodigoTarjeta[num_estacion]);
        MandarCaracter(COM1, TIPO_TRAMA_RECONEXION);
        MandarCaracter(COM1, FIN_TRAMA);
        DesactivarDE(COM1);
}

void DesconectarEstacion(unsigned char Est)
{
        int Estado;

        EstacionActiva[Est] = FALSE;
        AbrirFichEstaciones();
        NetSeek(&ArchvEstaciones, (NumeroNodal*4)+Est);
        Estado = NO_OPERATIVA;
        NetWrite(&ArchvEstaciones, &Estado);
        CerrarFichEstaciones();
        if (--NumEstacionesActivas == 0)
        {
                sprintf(Texto," Nodal PC %i: No active stations => Exit programme", NumeroNodal);
                EscribirLineaFichLog(Texto);
                Flag.Fin = TRUE;
        }
        MandarTramaDesconexion(Est);
}
```

```
void ConectarEstacion(unsigned char Est)
{
	EstacionActiva[Est] = TRUE;
	NumEstacionesActivas++;
	MandarTramaReconexion(Est);
}

void MandarTramaSinc(unsigned char Est)
{
	ActivarDE(COM1);
	MandarCaracter(COM1, CodigoTarjeta[Est]);
	MandarCaracter(COM1, TIPO_TRAMA_SINCRONIZACION);
	MandarCaracter(COM1, FIN_TRAMA);
	DesactivarDE(COM1);
}

/*-------------------------------------------------------------------------------------------------------------------
  FUNCIONES PARA LA GESTIÓN DE ERRORES
-------------------------------------------------------------------------------------------------------------------*/
void GestionaErrores(void)
{
	if (Flag.Error)
	{
		Flag.Error = FALSE;
		Flag.EscribirError = TRUE;		// Todos los errores se escriben en el fichero LOG
		if(FlagError.TimeOut)			//salvo los de TimeOut.
		{
			FlagError.TimeOut = FALSE;
			Flag.EscribirError = FALSE;

			// A los TIEMPO_DESCONEXION segundos sin respuesta de una estación ésta se
			//desconecta:
			if (++ContadorTimeOut[EstacionActual] >=
				((INT_TIMER*TIEMPO_DESCONEXION)/
				(NUM_TICKS*NumEstacionesActivas)))
			{
				Flag.EscribirError = TRUE;
				DesconectarEstacion(EstacionActual);
				ErroresAEscribir++;
				EstacionError[ErroresAEscribir] = EstacionActual;
				IdentificadorError[ErroresAEscribir] = DESCONEXION_TIMEOUT;
			}
		}
		if(FlagError.TramaIncompleta)	// Si hay recepción de trama incompleta, la trata como
		{					//un error de recepción
			FlagError.TramaIncompleta = FALSE;
			FlagError.RecepcionTrama = TRUE;		// Para entrar en el siguiente 'if'
		}
		if(FlagError.RecepcionTrama)
		{
			FlagError.RecepcionTrama = FALSE;
			ErroresAEscribir++;
			Flag.PedirReenvio = TRUE;
			EstacionError[ErroresAEscribir] = EstacionActual;
			IdentificadorError[ErroresAEscribir] = ERROR_RECEPCION_TRAMA;
		}
```

```
if(FlagError.SaturacionBuffer)
{
        FlagError.SaturacionBuffer = FALSE;
        ErroresAEscribir++;
        EstacionError[ErroresAEscribir] = EstacionActual;
        IdentificadorError[ErroresAEscribir] = ERROR_SATURACION_BUFFER;
}
if(FlagError.InicializacionADCs)
{
        FlagError.InicializacionADCs = FALSE;
        ErroresAEscribir++;
        EstacionError[ErroresAEscribir] = EstacionActual;
        IdentificadorError[ErroresAEscribir] = ERROR_INIC_ADCs;
        DesconectarEstacion(EstacionActual);        // Si hay error de inicialización, se
                                                    //desconecta la estación.
}
if(FlagError.Sincronizacion)
{
        FlagError.Sincronizacion = FALSE;
        ErroresAEscribir++;
        EstacionError[ErroresAEscribir] = EstacionActual;
        IdentificadorError[ErroresAEscribir] = ERROR_SINCRONIZACION;
}
// Escribe el error en el fichero de estado
if (Flag.EscribirError)
{
        for (i=ErroresAEscribir; i>0; i--)
        {
                sprintf( Texto, "\n %02i:%02i:%02i/%i/%02i Error in station (%i, %i)",
                        hora, minuto, segundo, dia.total, year,
                        NumeroNodal, EstacionError[i]);
                EscribirLineaFichLog(Texto);
                EscribirLineaFichLog(" Type of error:");
                switch (IdentificadorError[i])
                {
                        case DESCONEXION_TIMEOUT:
                                sprintf( Texto, " No response from station after %i
                                        seconds", TIEMPO_DESCONEXION);
                                EscribirLineaFichLog(Texto);
                                EscribirLineaFichLog(" Station disconnected from
                                        serial line");
                                break;

                        case ERROR_SATURACION_BUFFER:
                                EscribirLineaFichLog(" Data buffer overflow\n");
                                break;

                        case ERROR_INIC_ADCs:
                                EscribirLineaFichLog(" ADCs initialization error\n");
                                break;

                        case ERROR_SINCRONIZACION:
                                EscribirLineaFichLog(" Software clock synchronization
                                        error\n");
                                break;
```

```
                                        default:
                                                EscribirLineaFichLog(" Error in treatment of errors\n");
                                }
                        }
                        Flag.EscribirError = FALSE;
                        ErroresAEscribir = 0;
                }
        }
}

/*-------------------------------------------------------------------------------------------------------------------
  FUNCIONES PARA LA GESTIÓN DE FICHEROS
-------------------------------------------------------------------------------------------------------------------*/
void AbrirFicheros(void)
{
        unsigned char  i;

        // El fichero de comandos se nombra aquí para no tener que hacerlo más, pero no se intenta abrir ya
        //que de momento probablemente no exista. Se crea sólo cuando se introducen comandos desde el
        //servidor.
        sprintf( NomFichComandos,"D:\\GENERAL\\COMMANDS.PC%i", NumeroNodal);

        // Abre los archivos de datos, nombrados en la función 'NombrarFicheros'
        for (i=0; i<4; i++)
        {
                /* Esta primera apertura es sin file-locking, para comprobar que el fichero puede crearse. No
                debe haber colisión, porque el programa de detección (que es el único que podría intentar
                acceder a la vez) no puede haber seleccionado estos ficheros para detección, ya que hasta
                ahora estas estaciones estaban marcadas como desconectadas en el fichero de estado de
                estaciones.*/
                if (( fichDatos[i] = fopen( NomFichDatos[i], "wb")) == NULL)
                {
                        printf("\n Unable to open file %s\n", NomFichDatos[i]);
                        sprintf( Texto," Unable to open file %s ", NomFichDatos[i]);
                        EscribirLineaFichLog(Texto);
                        Flag.Fin = TRUE;
                }
        }
        if ((fichDatos[0] != NULL)&&(fichDatos[1] != NULL)&&(fichDatos[2] != NULL)
         &&(fichDatos[3] != NULL))
        {
                EscribirLineaFichLog(" Definition and opening of files correct ");
                sprintf(Texto, " Nodal PC clock controller: Station (%i, %i)\n", NumeroNodal,
                ControladorReloj);
                EscribirLineaFichLog(Texto);
        }
        for (i=0; i<4; i++) fclose(fichDatos[i]);       // Se dejan cerrados para que el servidor 2 pueda
                                                        //acceder a ellos.
}

void GrabarTramaBuffDatos(int Estacion)
{
        Princ_Buf_RS += 2;          // Sumamos 2 para descartar el código de tarjeta inicial y el carácter de
                                    //tipo de trama
        for(i = Princ_Buf_RS; i<Princ_Buf_RS + BYTES_DATOS; i++)
        {
```

```
                BufferDatos[Estacion][Fin_Buf_Datos[Estacion]] = Buffer_RS[i];
                Fin_Buf_Datos[Estacion]++;
        }
        Princ_Buf_RS = (Princ_Buf_RS + BYTES_DATOS + 2) % Tamano_Buffer_RS;
        // Sumamos 2 para descartar el código de tarjeta final y el carácter de fin de trama
}

//   CONSULTA DEL RELOJ para ver si toca grabar algún buffer de datos en el fichero correspondiente.
unsigned char TocaGrabarBufferFichero(void)
{
        unsigned char Temp;

        if (segundo >= 30) Temp = segundo-30;       // Para que descargue cada buffer dos veces por minuto
        else Temp = segundo;                        //(segs. 0 y 30, 1 y 31, 2 y 32,...)
        if ((Temp%6)==NumeroNodal)
        {
                EstacionGrabacion = Temp/6;
                // Sólo se devuelve TRUE si, además de ser el segundo correspondiente, hay más de cinco
                //segundos de datos en el buffer
                if (Fin_Buf_Datos[EstacionGrabacion] - Princ_Buf_Datos[EstacionGrabacion] >
                        5*BYTES_DATOS);
                {
                        return(TRUE);
                }
        }
        return(FALSE);
}

void NombrarFicheros(void)
{
        strcpy(NomFichLog, "D:\\GENERAL\\GRANSASS.LOG");   // Lo primero que hacemos es
                                                           //es nombrar el LOG para poder
                                                           //escribir en él
        sprintf(Texto, "\n BOOTING OF NODAL PC NUMBER %u \n", NumeroNodal);
        EscribirLineaFichLog(Texto);

        strcpy(NomFichEstaciones, "D:\\GENERAL\\STATIONS.BIN");
        sprintf( Texto, " Name of status of stations file: %s", NomFichEstaciones);
        EscribirLineaFichLog(Texto);

        sprintf( Texto, " Name of status of system file: %s", NomFichLog);
        EscribirLineaFichLog(Texto);

        sprintf( NomFichComandos,"D:\\GENERAL\\COMMANDS.PC%i", NumeroNodal);
        sprintf( Texto, " Name of command file: %s", NomFichComandos);
        EscribirLineaFichLog(Texto);

        EscribirLineaFichLog(" Name of data files: ");
        for (i=0; i<4; i++)
        {
                sprintf(NomFichDatos[i], "D:\\DATA\\PCNODAL%i\\NOD%iEST%i.DAT",
                        NumeroNodal, NumeroNodal, i);
                sprintf( Texto, " %s ", NomFichDatos[i]);
                EscribirLineaFichLog(Texto);
        }
}
```

```
void CerrarFichLog(void)
{
	NetClose(&ArchvLog);
}

void AbrirFichLog(int RecLength)  // Apertura en modo escritura para añadir (se abre para escritura,
                                  //posicionando el puntero al final del archivo, con protección total y
                                  //tamaño de record de 70 (caracteres por línea)
{
	do
	{
		NetReset( NomFichLog, FModus( 2, 2), RecLength, &ArchvLog);
		if( NetError == NE_FileNotFound)
			NetRewrite( NomFichLog, FModus( 2, 2), RecLength, &ArchvLog);
	}
	while ((!(Is_NetWriteOk( &ArchvLog ))) && (!kbhit()));
	NetSeekEnd(&ArchvLog);
}

void EscribirLineaFichLog(char Texto[100])
{
	char Cadena[101];
	char i;

	AbrirFichLog(strlen(Texto)+1);
	sprintf(Cadena,"%s\n", Texto);
	NetWrite(&ArchvLog, &Cadena);
	CerrarFichLog();
}

void AbrirFichEstaciones(void)	// Apertura en modo escritura, con protección total y tamaño de record = 1
{
		NetReset( NomFichEstaciones, FModus( 3, 2), 1, &ArchvEstaciones);
		if( NetError == NE_FileNotFound)
		{
			sprintf(Texto, "\n File STATIONS.BIN does not exist or is not present in
			C:\SHARED\GENERAL\n Nodal PC %i quits programme", NumeroNodal);

			exit(0);
		}
}

void CerrarFichEstaciones(void)
{
	NetClose( &ArchvEstaciones );
}

void MarcarFichEstaciones(void)
{
	int Estado[4];
	int i;

	AbrirFichEstaciones();
	NetSeek(&ArchvEstaciones, NumeroNodal*4);
	for (i=0; i<4; i++)
```

```
        {
                Estado[i] = REGISTRANDO;
                NetWrite(&ArchvEstaciones, &Estado[i]);
        }
        CerrarFichEstaciones();
}

void AbrirFichDatos(int Estacion)  // Apertura en modo escritura, con protección total y tamaño de record = 1
{                                  //record = BYTES_DATOS
        do
        {
                NetReset( NomFichDatos[Estacion], FModus( 3, 2), BYTES_DATOS,
                &ArchvDatos[Estacion]);
                if( NetError == NE_FileNotFound)
                        NetRewrite( NomFichDatos[Estacion], FModus( 3, 2), BYTES_DATOS,
                        &ArchvDatos[Estacion]);
        }
        while (!(Is_NetWriteOk( &ArchvDatos[Estacion] )));
}

void CerrarFichDatos(int Estacion)
{
        NetClose( &ArchvDatos[Estacion]);
}

void GrabarBufferFichero(int Estacion)
{
        int NumTramas;

        AbrirFichDatos(Estacion);
        // La grabación no se puede hacer de una sola vez, porque hay que ir comprobando si se ha llegado al
        //fin del fichero y eso no se puede hacer si los bloques no tienen una longitud fija.
        // El número de tramas es un número entero, porque ya se ha comprobado que las tramas estaban
        //bien.
        NumTramas = (Fin_Buf_Datos[Estacion] - Princ_Buf_Datos[Estacion])/BYTES_DATOS;
        for (i=0; i<NumTramas; i++)
        {
                NetSeek(&ArchvDatos[Estacion], ContadorTramasEnFichero[Estacion]);
                NetWrite(&ArchvDatos[Estacion], &BufferDatos[Estacion][Princ_Buf_Datos[Estacion]]);
                Princ_Buf_Datos[Estacion] += BYTES_DATOS;
                ContadorTramasEnFichero[Estacion]++;
                if (ContadorTramasEnFichero[Estacion] == TRAMAS_FICHERO)
                {
                        ContadorTramasEnFichero[Estacion] = 0;
                        NetSeek(&ArchvDatos[Estacion], 0);
                }
        }
        Princ_Buf_Datos[Estacion] = Fin_Buf_Datos[Estacion] = 0;
        CerrarFichDatos(Estacion);
}

int AbrirFichComandos(void)        // Apertura en modo lectura, con protección total y tamaño de record = 1.
                                   // Devuelve TRUE si se ha abierto, FALSE si no (p.ej., si no existe el
                                   //fichero).
{
        NetReset( NomFichComandos, FModus( 1, 2), 1, &ArchvComandos);
```

```
        // El fichero de comandos no existe hasta que se ha escrito algún comando desde el servidor; por eso
        //no se gestiona el error NE_FileNotFound.
        if( NetError == NE_AccessDenied)
        {
                sprintf(Texto, "\n Access denied to command file %s ", NomFichComandos);
                EscribirLineaFichLog(Texto);
                EscribirLineaFichLog(" Waiting until access is permitted");
                do
                {
                        NetReset( NomFichComandos, FModus( 1, 2), 1, &ArchvComandos);
                }
                while (!(Is_NetReadOk( &ArchvComandos )));
        }
        if (Is_NetOpen(&ArchvComandos)) return(TRUE);
        else return(FALSE);
}

void CerrarFichComandos(void)
{
        NetClose(&ArchvComandos);
}

void AtiendeComandos(void)).
{
        if (minuto != minuto_anterior) LeeFichComandos();
}

void LeeFichComandos(void)
{
        unsigned char Estacion, Comando, Parametro;
        long int Longitud;

        if (AbrirFichComandos())            // Si existe se abre y se leen los comandos
        {
                Longitud = NetSeekEnd(&ArchvComandos);          // Calcula la longitud del fichero
                if(Longitud%3 != 0) EscribirLineaFichLog(" Error in command file ");
                else
                {
                        NetSeek(&ArchvComandos, 0L);            // Vuelve al principio del fichero
                        for (i=0; i<Longitud/3; i++)
                        {
                                NetRead(&ArchvComandos, &Estacion);
                                NetRead(&ArchvComandos, &Comando);
                                NetRead(&ArchvComandos, &Parametro);
                                switch(Comando)
                                {
                                        case TIPO_TRAMA_CAMBIOGANANCIA:
                                                MandarTramaGanancia(Estacion, Parametro);
                                                sprintf(Texto,"\n Change of gain command sent to
                                                        station(%i, %i)", NumeroNodal, Estacion);
                                                EscribirLineaFichLog(Texto);
                                                sprintf(Texto," New gain: %i", Parametro);
                                                EscribirLineaFichLog(Texto);
                                                break;

                                        case TIPO_TRAMA_DESCONEXION:
```

```
                                        DesconectarEstacion(Estacion);
                                        sprintf(Texto," Disconnection command sent to station
                                                (%i, %i)", NumeroNodal, Estacion);
                                        EscribirLineaFichLog(Texto);
                                        break;

                                case TIPO_TRAMA_RECONEXION:
                                        ConectarEstacion(Estacion);
                                        sprintf(Texto," Reconnection command sent to station
                                                (%i, %i)", NumeroNodal, Estacion);
                                        EscribirLineaFichLog(Texto);
                                        break;

                                case TIPO_TRAMA_SINCRONIZACION:
                                        MandarTramaSinc(Estacion);
                                        sprintf(Texto," Software clock synchronization
                                                command sent to station (%i, %i)",
                                                NumeroNodal, Estacion);
                                        EscribirLineaFichLog(Texto);
                                        break;

                                default:
                                        sprintf(Texto," Error in command file %s",
                                                NomFichComandos);
                                        EscribirLineaFichLog(Texto);
                                }
                        }
                }
                NetClose(&ArchvComandos);       // Una vez leído, el fichero de comandos se borra
                remove(NomFichComandos);
                sprintf(Texto," Command file %s erased", NomFichComandos);
                EscribirLineaFichLog(Texto);
        }
}

/* CREACIÓN DEL MODO DE ARCHIVO DE TIPO DE ACCESO Y LOCKING: La función Fmodus toma
como entrada el tipo de acceso y nivel de protección de archivo, y da como salida el modo de archivo.*/
int FModus( int Acceso, int Protec )
{
        static unsigned char TipoAcceso[ 3 ] = { FM_R, FM_W, FM_RW };
        static unsigned char TipoProtec[ 5 ] = { SM_COMP, SM_RW, SM_R, SM_W, SM_NO };

        return TipoAcceso[ Acceso-1 ] | TipoProtec[ Protec-1 ];
}

/*----------------------------------------------------------------------------------------------------------------
  OTRAS FUNCIONES
----------------------------------------------------------------------------------------------------------------*/
void RefrescaWD(void)
{
        outportb(PUERTO_WD, TIEMPO_WD);
}

void CompruebaControladorReloj(void)
{
        if (!EstacionActiva[ControladorReloj])
```

```
	{
		do
		{
			ControladorReloj ++;	// Si la estación que se ocupa del reloj no está activa,
						//pasa el control a la siguiente.
			if (ControladorReloj == NUMERO_ESTACIONES) ControladorReloj = 0;
		}
		while ((!EstacionActiva[ControladorReloj])&&(NumEstacionesActivas != 0));
		// La última condición es para que no se quede esperando aquí si no hay ninguna estación
		//activa
		if (NumEstacionesActivas != 0)
		{
			sprintf(Texto," Control of nodal PC clock passes to station (%i, %i)\n",
				NumeroNodal, ControladorReloj);
			EscribirLineaFichLog(Texto);
		}
	}
}

unsigned char Salir(void)		// Se sale del programa cuando se aprieta el pulsador de COM2, cuando
					//no hay estaciones activas (ver función 'DesconectarEstacion') o cuando
					//no se ha podido abrir algún fichero al principio.
{
	outportb(RS232[COM2].mcr, 0x01);	// Saca un 1(+8V) por la línea DTR y un 0(-8V) por la
						//línea RTS de COM2.
	if((!(inportb(RS232[COM2].msr) & 0x40))||(Flag.Fin == TRUE)) return(TRUE);
	else return(FALSE);
}

void LeerCodigoTarjeta(void)
{
	unsigned char registro;

	outportb(RS232[COM2].mcr, 0x01);	// Saca un 1(+8V) por el bit 0 (línea DTR) y un 0(-8V)
						//por el bit 1 (línea RTS)
	delay(500);	// Para que se estabilice la tensión de entrada
	registro = ~(inportb(RS232[COM2].msr));
	NumeroNodal = (((registro & 0x10) >> 4) | ((registro & 0x80) >> 6) | ((registro & 0x20) >> 3));
	printf("\nPC nodal número %i", NumeroNodal);
}

void InicializarPunteros(void)
{
	unsigned char i;

	Princ_Buf_RS	= 0;
	Fin_Buf_RS	= 0;
	PtrIdentificar	= 0;
	for (i=0; i<4; i++)
	{
		Princ_Buf_Datos[i]	= 0;
		Fin_Buf_Datos[i]	= 0;
	}
}
```

```
/*-------------------------------------------------------------------------------------------------------------------
  PROGRAMA PRINCIPAL
-------------------------------------------------------------------------------------------------------------------*/
void main(void)
{
	clrscr();
	printf("\nPrograma PC_NODAL");
	InicializarComunicacionSerie(COM1,BAUD115200,NINGUNA,LON8,STOP1);
	LeerCodigoTarjeta();
	NombrarFicheros();
	AbrirFicheros();
	MarcarFichEstaciones();
	InicializarPunteros();
	do
	{
		MandarTramaPeticion();   // Manda la trama y espera a recibir respuesta o a que haya error
					 //de Time Out

		// Sólo se llama a 'MeterTramaBuffRecepcion' si no ha habido error de Time Out:
		if (!FlagError.TimeOut)
		{
			MeterTramaBuffRecepcion();
			// Sólo se llama a 'AnalizarTrama()' si ha llegado entera al buffer:
			if (!FlagError.TramaIncompleta) AnalizarTrama();
		}

		// Graba las tramas en un buffer y, en el segundo correspondiente, descarga éste en el
		//fichero de red:
		if (Flag.GrabarTramaBuffDatos)
		{
			Flag.GrabarTramaBuffDatos = FALSE;
			GrabarTramaBuffDatos(EstacionActual);
		}
		if (TocaGrabarBufferFichero())
		{
			GrabarBufferFichero(EstacionGrabacion);
		}

		AtiendeComandos();
		GestionaErrores();
		CompruebaControladorReloj();
		RefrescaWD();
	}
	while(!Salir());
	FinalizarComunicacionSerie();
}
```

ANEXO I.A.3.- PROGRAMA DE CONTROL DEL SISTEMA

/* Este anexo incluye los ficheros fuente del programa de control del sistema CONTROL.CPP, que ha sido realizado utilizando el compilador C++ Builder (versión 3), de Borland. Por brevedad, se han incluido sólo los ficheros fuente en los que se ha introducido código propio, dejando fuera los ficheros generados automáticamente por el compilador y a los que no se ha añadido código alguno (gestión de formularios de visualización del plano y de los ficheros de estado de las estaciones y del sistema). La única excepción a esto es el fichero fuente principal del proyecto (anexo I.A.3.a), que sí se ha incluido por considerarse de interés. */

I.A.3.a. FICHERO FUENTE PRINCIPAL DE LA APLICACIÓN

Fichero: CONTROL.CPP

/* Funciones de inicialización de los formularios creados en la aplicación y gestión de excepciones. Este fichero es creado por el compilador C++ Builder al abrir un nuevo proyecto, y actualizado automáticamente cada vez que se crea un nuevo formulario.*/

```
#include <vcl.h>
#pragma hdrstop
USERES("Control.res");
USEFORM("Principal.cpp", FormPpal);
USEFORM("Comandos.cpp", FormComandos);
USEFORM("NewGain.cpp", FormNewGain);
USEFORM("Disconnect.cpp", FormDisconnect);
USEFORM("Reconnect.cpp", FormReconnect);
USEFORM("Synchro.cpp", FormSynchro);
USEFORM("SystemStatus.cpp", FormSystemStatus);
USEFORM("Plano.cpp", FormPlano);
USEFORM("Stations.cpp", FormStations);
USEFORM("Data2.cpp", FormData2);
//---------------------------------------------------------------------------

WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
   try
   {
      Application->Initialize();
      Application->CreateForm(__classid(TFormPpal), &FormPpal);
      Application->CreateForm(__classid(TFormComandos), &FormComandos);
      Application->CreateForm(__classid(TFormNewGain), &FormNewGain);
      Application->CreateForm(__classid(TFormDisconnect), &FormDisconnect);
      Application->CreateForm(__classid(TFormReconnect), &FormReconnect);
      Application->CreateForm(__classid(TFormSynchro), &FormSynchro);
      Application->CreateForm(__classid(TFormSystemStatus), &FormSystemStatus);
      Application->CreateForm(__classid(TFormPlano), &FormPlano);
      Application->CreateForm(__classid(TFormStations), &FormStations);
```

```
      Application->CreateForm(__classid(TFormData2), &FormData2);
      Application->Run();
   }
   catch (Exception &exception)
   {
      Application->ShowException(&exception);
   }
   return 0;
}
```

I.A.3.b FICHERO FUENTE DEL FORMULARIO PRINCIPAL

Fichero: PRINCIPAL.CPP

/* Funciones de gestión de los eventos asociados a los componentes del formulario principal de la aplicación (figura 3.25 de la memoria). */

```
/*****************************************************************************
  Macros y ficheros de cabecera
*****************************************************************************/
#include <vcl.h>
#include <stdio.h>
#include <process.h>
#pragma hdrstop

#include "Principal.h"
#include "Comandos.h"
#include "SystemStatus.h"
#include "Plano.h"
#include "Stations.h"
#include "Data2.h"

// Posibles estados de las estaciones:
#define NOT_OPERATIVE       '0';
#define REGISTERING         '1';
#define DETECTING           '2';

#pragma package(smart_init)
#pragma resource "*.dfm"

TFormPpal *FormPpal;

/*****************************************************************************
  Definición de variables
*****************************************************************************/
unsigned char PCnodal[6], Estacion[4];

/*****************************************************************************
  Funciones
*****************************************************************************/
/*  COMPONENTES UTILIZADOS:
   FormPpal: TFormPpal.- Formulario principal.
   Button1: TButton.- Botón para visualización del plano de estaciones ('View plan of stations')
   Button2: TButton.- Botón para visualización del fichero de estado de las estaciones ('Check state of
stations').
   Button3: TButton.- Botón para visualización del fichero de estado del sistema ('Check system status').
   Button4: TButton.- Botón para introducción de comandos de control ('Input commands').
   Button5: TButton.- Botón para demultiplexado de ficheros de datos ('Demultiplex data files').
   Button6: TButton.- Botón para visualización del fichero de ayuda ('Help').
   Button7: TButton.- Botón de salida del programa ('Quit').
   Image1: TImage.- Visualizador del fichero gráfico de logotipo.
```

```
   Shape1: TShape.- Marco para los botones.
*/
//---------------------------------------------------------------------------
__fastcall TFormPpal::TFormPpal(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
```

/* GESTIÓN DEL EVENTO OnClick DEL COMPONENTE Button1. Al hacer click sobre *Button1* se abre el formulario *FormPlano* y se visualiza un plano con la situación de estaciones, PCs nodales y sala de control (figura 6.5).

```
    COMPONENTES UTILIZADOS en FormPlano:
    FormPlano: TFormPlano.- Formulario.
    Image1: TImage.- Visualizador del fichero gráfico
.*/
void __fastcall TFormPpal::Button1Click(TObject *Sender)
{
   FormPlano->Show();      // No hace falta abrir el fichero al abrir el formulario porque se hace en tiempo de
                           //diseño (propiedad 'Picture').
}
//---------------------------------------------------------------------------
```

/* GESTIÓN DEL EVENTO OnClick DEL COMPONENTE Button2. Al hacer click sobre *Button2* se abre el formulario *FormStations* y se presenta en pantalla la información del fichero de estado de las estaciones STATIONS.BIN (figura 6.6).

```
   COMPONENTES UTILIZADOS en FormStations:
   FormStations: TFormStations.- Formulario.
   Memo1: TMemo.- Cuadro de visualización del fichero de estado de las estaciones.
 */
void __fastcall TFormPpal::Button2Click(TObject *Sender)
{
    FILE    *fichEstaciones, *fichTextoEstaciones;
    const int NumNodales = 6, NumEstaciones = 4;
    int Estado;

    // Comprueba que el fichero STATIONS.BIN existe, para lo cual intenta abrirlo para lectura.
    if ((fichEstaciones = fopen("STATIONS.BIN", "rb")) == NULL)
    {
       MessageBeep(MB_OK);
       Application->MessageBox("File STATIONS.BIN does not exist in working directory",
          "CONTR11 Message", MB_ICONEXCLAMATION);
    }
    else
    {
       // Abre un fichero de texto para presentar la información del fichero binario STATIONS.BIN.
       if ((fichTextoEstaciones = fopen("STATIONS.TMP", "w")) == NULL)
       {
          MessageBeep(MB_OK);
          Application->MessageBox("Unable to open file STATIONS.TMP for writing",
             "CONTR11 Message", MB_ICONEXCLAMATION);
       }
       else
       {
```

```
        fprintf(fichTextoEstaciones,"\n\n     (Nodal PC, Station)        STATE\n");

        for (int i=0; i<NumNodales; i++)
        {
           for (int j=0; j<NumEstaciones; j++)
           {
              Estado = getc(fichEstaciones);
              switch (Estado)
              {
                 case '0':
                    fprintf(fichTextoEstaciones,
                          "\n              (%i, %i)          Not operative", i, j);
                    break;
                 case '1':
                    fprintf(fichTextoEstaciones,
                          "\n              (%i, %i)          Registering", i, j);
                    break;
                 case '2':
                    fprintf(fichTextoEstaciones,
                          "\n              (%i, %i)          Detecting", i, j);
                    break;
                 default:
                    fprintf(fichTextoEstaciones,
                          "\n            Error in file STATIONS.BIN");
              }
           }
           fprintf(fichTextoEstaciones,"\n");
        }
        fclose(fichTextoEstaciones);
        fclose(fichEstaciones);
        FormStations->Memo1->Lines->LoadFromFile("STATIONS.TMP");    // Carga el fichero temporal
        FormStations->ShowModal();                                   //de texto en Memo1 y abre
                                                                     //el formulario.
        remove("STATIONS.TMP");          // Borra el fichero temporal de texto (.TMP), deja el binario de
                                         //estaciones (.BIN).
     }
   }
}
//---------------------------------------------------------------------------
```

/* GESTIÓN DEL EVENTO OnClick DEL COMPONENTE Button3. Al hacer click sobre *Button3* se abre el formulario *FormSystemStatus* y se visualiza el fichero de estado del sistema 'GRANSASS.LOG' (figura 6.7).

```
  COMPONENTES UTILIZADOS en  FormSystemStatus:
  FormSystemStatus: TFormSystemStatus.- Formulario.
  Memo1: TMemo.- Cuadro de visualización del fichero de estado del sistema.
*/
void __fastcall TFormPpal::Button3Click(TObject *Sender)
{
   FILE *Fich;

   // Comprueba que el fichero GRANSASS.LOG existe, para lo cual intenta abrirlo para lectura.
   if ((Fich = fopen("GRANSASS.LOG", "r")) == NULL)
   {
      MessageBeep(MB_OK);
      Application->MessageBox("File GRANSASS.LOG does not exist in working directory",
```

```
        "CONTR11 Message", MB_ICONEXCLAMATION);
   }
   else
   {
      fclose(Fich);
      FormSystemStatus->Memo1->Lines->LoadFromFile("GRANSASS.LOG");
      FormSystemStatus->ShowModal();
   }
}

//---------------------------------------------------------------------------
/* GESTIÓN DEL EVENTO OnClick DEL COMPONENTE Button4. Al hacer click sobre Button4 se abre el
formulario de comandos FormComandos (figura 6.8 y apéndice III.C).
*/
void __fastcall TFormPpal::Button4Click(TObject *Sender)
{
   // Al abrir el formulario de comandos está seleccionado, por defecto, el PC nodal 0 y ninguna estación
   PCnodal[0] = TRUE;
   PCnodal[1] = FALSE;
   PCnodal[2] = FALSE;
   PCnodal[3] = FALSE;
   PCnodal[4] = FALSE;
   PCnodal[5] = FALSE;
   Estacion[0] = FALSE;
   Estacion[1] = FALSE;
   Estacion[2] = FALSE;
   Estacion[3] = FALSE;

   FormComandos->ShowModal(); // Abre el formulario de comandos
}
//---------------------------------------------------------------------------

/* GESTIÓN DEL EVENTO OnClick DEL COMPONENTE Button5. Al hacer click sobre Button5 se abre el
formulario de demultiplexión de los ficheros de datos FormData2 (figura 6.10 y apéndice III.G).
*/
void __fastcall TFormPpal::Button5Click(TObject *Sender)
{
  FormData2->ShowModal();
}
//---------------------------------------------------------------------------

/* GESTIÓN DEL EVENTO OnClick DEL COMPONENTE Button7. Al hacer click sobre Button7 se cierra
el formulario principal y se sale del programa.
*/
void __fastcall TFormPpal::Button7Click(TObject *Sender)
{
   Close();
}
//---------------------------------------------------------------------------
```

I.A.3.c. FICHERO FUENTE DEL FORMULARIO DE COMANDOS

Fichero: COMANDOS.CPP

/* Funciones de gestión de los eventos asociados a los componentes del formulario de comandos (figura 3.29). */

```
/*****************************************************************************
  Macros y ficheros de cabecera
*****************************************************************************/
#include <vcl.h>
#include <stdio.h>
#pragma hdrstop

#include "Comandos.h"
#include "NewGain.h"
#include "Disconnect.h"
#include "Reconnect.h"
#include "Synchro.h"

#define COMANDO_CAMBIOGANANCIA      0xB2
#define COMANDO_DESCONEXION         0xE5
#define COMANDO_RECONEXION          0x5E
#define COMANDO_SINCRONIZACION      0xF6

#pragma package(smart_init)
#pragma resource "*.dfm"
TFormComandos *FormComandos;

/*****************************************************************************
  Definición de variables
*****************************************************************************/
extern unsigned char PCnodal[6], Estacion[4];
extern int NumeroPCnodal;
unsigned char TipoComando, Parametro;

/*****************************************************************************
  Funciones
*****************************************************************************/
/*  COMPONENTES UTILIZADOS:
  FormComandos: TFormComandos.- Formulario.
  Label1 y Label2: TLabel.- Etiquetas de texto estático.
  RadioGroup1: TRadioGroup.- Grupo de botones de selección del número de PC nodal.
  CheckBox1 a CheckBox4: TCheckBox.- Casillas de selección de estaciones 0 a 3.
  Button1: TButton.- Botón para introducción de comando de desconexión de la línea serie ('Disconnect from
serial line').
  Button2: TButton.- Botón para introducción de comando de reconexión a la línea serie ('Reconnect to serial
line').
  Button3: TButton.- Botón para introducción de comando de cambio de ganancia ('Change of gain').
  Button4: TButton.- Botón para introducción de comando de sincronización del reloj software ('Synchronise
software clock').
```

```
   Button5: TButton.- Botón para anulación de todos los comandos ('Reset command files')
   Button6: TButton.- Botón para introducción de comando de reinicialización de estación ('Reset station').
*/

__fastcall TFormComandos::TFormComandos(TComponent* Owner)
   : TForm(Owner)
{
}
//---------------------------------------------------------------------------

/* SELECCIÓN DE ESTACIONES A LAS QUE SE ENVIARÁN LOS COMANDOS DE CONTROL. Los números de PC nodal y de estaciones seleccionadas se toman de los componentes RadioGroup1 y CheckButton 1 a 4. Si el botón 6 de Radiogroup1 está seleccionado se activan los flags correspondientes a todos los PCs nodales y estaciones del sistema. */
void SeleccionaEstaciones(void)
{
   unsigned char i;

   {
      for(i=0; i<=5; i++)
      {
         PCnodal[i] = FALSE;
      }
      for(i=0; i<=3; i++)
      {
         Estacion[i] = FALSE;
      }
   }

   NumeroPCnodal = FormComandos->RadioGroup1->ItemIndex;
   if (NumeroPCnodal != 6)
   {
      PCnodal[NumeroPCnodal] = TRUE;
      if(FormComandos->CheckBox1->Checked) Estacion[0] = TRUE;
      if(FormComandos->CheckBox2->Checked) Estacion[1] = TRUE;
      if(FormComandos->CheckBox3->Checked) Estacion[2] = TRUE;
      if(FormComandos->CheckBox4->Checked) Estacion[3] = TRUE;
   }
   else
   {
      for(i=0; i<=5; i++)
      {
         PCnodal[i] = TRUE;
      }
      for(i=0; i<=3; i++)
      {
         Estacion[i] = TRUE;
      }
   }
}
//---------------------------------------------------------------------------

/* GESTIÓN DEL EVENTO OnClick DEL COMPONENTE Button1. Al hacer click sobre Button1 se abre el formulario FormDisconnect (apéndice III.D), en el que se pide confirmación del comando de desconexión.
*/
void __fastcall TFormComandos::Button1Click(TObject *Sender)
```

```
{
   TipoComando = COMANDO_DESCONEXION;
   Parametro = 0;

   SeleccionaEstaciones();
   FormDisconnect->ShowModal();
}
//---------------------------------------------------------------------------
```

/* GESTIÓN DEL EVENTO OnClick DEL COMPONENTE Button2. Al hacer click sobre *Button2* se abre el formulario *FormReconnect* (apéndice III.E), en el que se pide confirmación del comando de reconexión.
*/

```
void __fastcall TFormComandos::Button2Click(TObject *Sender)
{
   TipoComando = COMANDO_RECONEXION;
   Parametro = 0;

   SeleccionaEstaciones();
   FormReconnect->ShowModal();
}
//---------------------------------------------------------------------------
```

/* GESTIÓN DEL EVENTO OnClick DEL COMPONENTE Button3. Al hacer click sobre *Button3* se abre el formulario *FormNewGain* (figura 6.9 y apéndice III.F), en el que se debe introducir la nueva ganancia y se pide confirmación.
*/

```
void __fastcall TFormComandos::Button3Click(TObject *Sender)
{
   TipoComando = COMANDO_CAMBIOGANANCIA;
   // El parámetro es la nueva ganancia, pero no se inicializa porque se hará en el nuevo formulario

   SeleccionaEstaciones();
   FormNewGain->ShowModal();
}
//---------------------------------------------------------------------------
```

/* GESTIÓN DEL EVENTO OnClick DEL COMPONENTE Button4. Al hacer click sobre *Button4* se abre el formulario *FormSynchro* (apéndice III.G), en el que se pide confirmación sobre el comando de sincronización del reloj *software*.
*/

```
void __fastcall TFormComandos::Button4Click(TObject *Sender)
{
   TipoComando = COMANDO_SINCRONIZACION;
   Parametro = 0;

   SeleccionaEstaciones();
   FormSynchro->ShowModal();
}
//---------------------------------------------------------------------------
```

/* GESTIÓN DEL EVENTO OnClick DEL COMPONENTE Button5. Al hacer click sobre *Button5* se borran todos los ficheros de comandos existentes, con lo cual se anulan los comandos introducidos hasta el momento.
*/

```
void __fastcall TFormComandos::Button5Click(TObject *Sender)
{
   unsigned char i;
```

```
   char NombreFichero[13];
   FILE *FicheroComandos;

   // La anulación de comandos consiste simplemente en borrar los ficheros de comandos que existan. Para
   //ello se intentan abrir para lectura (para que no se creen nuevos) y, si existen, se borran.
   for (i=0; i<=5; i++)
   {
      sprintf(NombreFichero,"COMMANDS.PC%u", i);
      if ((FicheroComandos = fopen(NombreFichero, "r")) != NULL)
      {
         fclose(FicheroComandos);
         remove(NombreFichero);
      }
   }
}
//---------------------------------------------------------------------------
```

/* GESTIÓN DEL EVENTO OnClick DEL COMPONENTE RadioGroup1. Al hacer click sobre cualquiera de los botones de *RadioGroup1* se desactiva la propiedad *Checked* de todos los cuadros de selección de estación, para que no aparezcan marcados.

```
*/
void __fastcall TFormComandos::RadioGroup1Click(TObject *Sender)
{
   TFormComandos::CheckBox1->Checked = FALSE;
   TFormComandos::CheckBox2->Checked = FALSE;
   TFormComandos::CheckBox3->Checked = FALSE;
   TFormComandos::CheckBox4->Checked = FALSE;
}
//---------------------------------------------------------------------------
```

I.A.3.d. FICHERO FUENTE DEL FORMULARIO DE CONFIRMACIÓN DEL COMANDO DE DESCONEXIÓN.

Fichero: DISCONNECT.CPP

/* Funciones de gestión de los eventos asociados a los componentes del formulario de confirmación del comando de desconexión. */

```
/*****************************************************************************
  Macros y ficheros de cabecera
*****************************************************************************/
#include <vcl.h>
#include <fstream>
#pragma hdrstop

#include "Disconnect.h"

#pragma package(smart_init)
#pragma resource "*.dfm"
TFormDisconnect *FormDisconnect;

/*****************************************************************************
  Definición de variables
*****************************************************************************/
extern unsigned char PCnodal[6], Estacion[4];
extern unsigned char TipoComando, Parametro;

/*****************************************************************************
  Funciones
*****************************************************************************/
/*  COMPONENTES UTILIZADOS:
   FormDisconnect: TFormDisconnect.- Formulario.
   Label1 y Label2: TLabel.- Etiquetas de texto estático.
   BitBtn1: TBitBtn.- Botón de confirmación del comando ('OK').
   BitBtn2: TBitBtn.- Botón de cancelación del comando ('Cancel').
*/

//---------------------------------------------------------------------------
__fastcall TFormDisconnect::TFormDisconnect(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------

/* GESTIÓN DEL EVENTO OnClick DEL COMPONENTE BitBtn1. Al hacer click sobre BitBtn1 se escribe el comando de desconexión de la línea serie en el fichero de comandos correspondiente a las estaciones seleccionadas.
*/
void __fastcall TFormDisconnect::BitBtn1Click(TObject *Sender)
{
    unsigned char i, j;
```

```
   FILE *FicheroSalida;
   char NombreFichero[13];

   for (i=0; i<=5; i++)
   {
      if (PCnodal[i])
      {
         for (j=0; j<=3; j++)
         {
            if (Estacion[j])
            {
               sprintf(NombreFichero,"COMMANDS.PC%i", i);
               if ((FicheroSalida = fopen(NombreFichero, "a")) == NULL)
                  fprintf(stderr, "Unable to open command file %s\n", NombreFichero);
               fseek(FicheroSalida, 0, SEEK_END);
               putc(j, FicheroSalida);
               putc(TipoComando, FicheroSalida);
               putc(Parametro, FicheroSalida);
               fclose(FicheroSalida);
            }
         }
      }
   }
}
```

I.A.3.e. FICHERO FUENTE DEL FORMULARIO DE CONFIRMACIÓN DEL COMANDO DE RECONEXIÓN.

Fichero: RECONNECT.CPP

/* Funciones de gestión de los eventos asociados a los componentes del formulario de confirmación del comando de reconexión. */

```
/*****************************************************************************
  Macros y ficheros de cabecera
*****************************************************************************/
#include <vcl.h>
#include <fstream>
#pragma hdrstop

#include "Reconnect.h"

#pragma package(smart_init)
#pragma resource "*.dfm"
TFormReconnect *FormReconnect;

/*****************************************************************************
  Definición de variables
*****************************************************************************/
extern unsigned char PCnodal[6], Estacion[4];
extern unsigned char TipoComando, Parametro;

/*****************************************************************************
  Funciones
*****************************************************************************/
/*  COMPONENTES UTILIZADOS:
   FormReconnect: TFormReconnect.- Formulario.
   Label1 y Label2: TLabel.- Etiquetas de texto estático.
   BitBtn1: TBitBtn.- Botón de confirmación del comando (‘OK’).
   BitBtn2: TBitBtn.- Botón de cancelación del comando (‘Cancel’).
*/
__fastcall TFormReconnect::TFormReconnect(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------

/* GESTIÓN DEL EVENTO OnClick DEL COMPONENTE BitBtn1. Al hacer click sobre BitBtn1 se escribe el comando de reconexión a la línea serie en el fichero de comandos correspondiente a las estaciones seleccionadas.
*/
void __fastcall TFormReconnect::BitBtn1Click(TObject *Sender)
{
    unsigned char i, j;
    FILE *FicheroSalida;
    char NombreFichero[13];
```

```
   for (i=0; i<=5; i++)
   {
      if (PCnodal[i])
      {
         for (j=0; j<=3; j++)
         {
            if (Estacion[j])
            {
               sprintf(NombreFichero,"COMMANDS.PC%i", i);
               if ((FicheroSalida = fopen(NombreFichero, "a")) == NULL)
                  fprintf(stderr, "Unable to open command file %s\n", NombreFichero);
               fseek(FicheroSalida, 0, SEEK_END);
               putc(j, FicheroSalida);
               putc(TipoComando, FicheroSalida);
               putc(Parametro, FicheroSalida);
               fclose(FicheroSalida);
            }
         }
      }
   }
}
//---------------------------------------------------------------------------
```

I.A.3.f. FICHERO FUENTE DEL FORMULARIO DE NUEVA GANANCIA.

Fichero: NEWGAIN.CPP

/* Funciones de gestión de los eventos asociados a los componentes del formulario de confirmación de la nueva ganancia (figura 3.30). */

```
/*****************************************************************************
  Macros y ficheros de cabecera
*****************************************************************************/
#include <vcl.h>
#include <fstream>
#include <stdio.h>
#pragma hdrstop

#include "NewGain.h"
#include "Comandos.h"

#pragma package(smart_init)
#pragma resource "*.dfm"
TFormNewGain *FormNewGain;

/*****************************************************************************
  Declaración de variables
*****************************************************************************/
extern unsigned char PCnodal[6], Estacion[4];
extern unsigned char TipoComando, Parametro;

/*****************************************************************************
  Funciones
*****************************************************************************/
/*  COMPONENTES UTILIZADOS:
   FormNewGain: TFormNewGain.- Formulario.
   Label1: TLabel.- Etiqueta de texto estático.
   RadioGroup1: TRadioGroup.- Grupo de botones de selección del número de PC nodal.
   BitBtn1: TBitBtn.- Botón de confirmación del comando (‘OK’).
   BitBtn2: TBitBtn.- Botón de cancelación del comando (‘Cancel’).
*/
__fastcall TFormNewGain::TFormNewGain(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------

/* GESTIÓN DEL EVENTO OnClick DEL COMPONENTE BitBtn1. Al hacer click sobre BitBtn1 se escribe el valor de la propiedad ‘ItemIndex’ del componente ‘RadioGroup1’ (cuyo valor está entre 0 y el número de botones menos uno) en el fichero de comandos correspondiente a las estaciones seleccionadas. Los PCs nodales interpretan la nueva ganancia como dos elevado a ese valor.
*/
void __fastcall TFormNewGain::BitBtn1Click(TObject *Sender)
{
    unsigned char i, j;
    FILE *FicheroSalida;
```

```
   char NombreFichero[13];

   Parametro = FormNewGain->RadioGroup1->ItemIndex;
   for (i=0; i<=5; i++)
   {
      if (PCnodal[i])
      {
         for (j=0; j<=3; j++)
         {
            if (Estacion[j])
            {
               sprintf(NombreFichero,"COMMANDS.PC%i", i);
               if ((FicheroSalida = fopen(NombreFichero, "a")) == NULL)
                  fprintf(stderr, "Unable to open command file %s\n", NombreFichero);
               fseek(FicheroSalida, 0, SEEK_END);
               putc(j, FicheroSalida);
               putc(TipoComando, FicheroSalida);
               putc(Parametro, FicheroSalida);
               fclose(FicheroSalida);
            }
         }
      }
   }
}
//---------------------------------------------------------------------------
```

I.A.3.g. FICHERO FUENTE DEL FORMULARIO DE CONFIRMACIÓN DEL COMANDO DE SINCRONIZACIÓN DEL RELOJ *SOFTWARE*.

Fichero: SYNCHRO.CPP

/* Funciones de gestión de los eventos asociados a los componentes del formulario de confirmación del comando de sincronización del reloj *software*.*/

```
/*****************************************************************************
  Macros y ficheros de cabecera
*****************************************************************************/
#include <vcl.h>
#include <fstream>
#pragma hdrstop

#include "Synchro.h"

#pragma package(smart_init)
#pragma resource "*.dfm"
TFormSynchro *FormSynchro;

/*****************************************************************************
  Declaración de variables
*****************************************************************************/
extern unsigned char PCnodal[6], Estacion[4];
extern unsigned char TipoComando, Parametro;

/*****************************************************************************
  Funciones
*****************************************************************************/
/*  COMPONENTES UTILIZADOS:
   FormSynchro: TFormSynchro.- Formulario.
   Label1 y Label2: TLabel.- Etiquetas de texto estático.
   BitBtn1: TBitBtn.- Botón de confirmación del comando ('OK').
   BitBtn2: TBitBtn.- Botón de cancelación del comando ('Cancel').
*/
__fastcall TFormSynchro::TFormSynchro(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
```

/* GESTIÓN DEL EVENTO OnClick DEL COMPONENTE BitBtn1. Al hacer click sobre *BitBtn1* se escribe el comando de sincronización del reloj *software* en el fichero de comandos correspondiente a las estaciones seleccionadas.
*/

```
void __fastcall TFormSynchro::BitBtn1Click(TObject *Sender)
{
    unsigned char i, j;
    FILE *FicheroSalida;
    char NombreFichero[13];
```

```
   for (i=0; i<=5; i++)
   {
      if (PCnodal[i])
      {
         for (j=0; j<=3; j++)
         {
            if (Estacion[j])
            {
               sprintf(NombreFichero,"COMMANDS.PC%i", i);
               if ((FicheroSalida = fopen(NombreFichero, "a")) == NULL)
                  fprintf(stderr, "Unable to open command file %s\n", NombreFichero);
               fseek(FicheroSalida, 0, SEEK_END);
               putc(j, FicheroSalida);
               putc(TipoComando, FicheroSalida);
               putc(Parametro, FicheroSalida);
               fclose(FicheroSalida);
            }
         }
      }
   }
}
```

I.A.3.h. FICHERO FUENTE DEL FORMULARIO DE DEMULTIPLEXIÓN DE LOS FICHEROS DE DATOS.

Fichero: DATA2.CPP

/* Funciones de gestión de los eventos asociados a los componentes del formulario de demultiplexión de los ficheros de datos (figura 3.31). */

```
/*****************************************************************************
  Macros y ficheros de cabecera
*****************************************************************************/
#define    BYTES_TRAMA    908

#include <vcl.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <ctype.h>
#include <process.h>
#pragma hdrstop

#include "Data2.h"

#pragma package(smart_init)
#pragma resource "*.dfm"
TFormData2 *FormData2;

/*****************************************************************************
  Declaración de variables
*****************************************************************************/
char NomDirectorio[30];
char NomFichDatos [14];
char Mascara[14];
unsigned char fraccion, segundo, minuto, hora, year;
unsigned int dia;
unsigned char ganancia;
unsigned int ContadorBytes;
int car, i, j;
int NumeroPCnodal, NumeroEstacion;

/*****************************************************************************
  Funciones
*****************************************************************************/
/*  COMPONENTES UTILIZADOS:
  FormData2: TFormData2.- Formulario.
  Label1 a Label4: TLabel.- Etiquetas de texto estático.
  RadioGroup1: TRadioGroup.- Grupo de botones de selección del número de PC nodal, para determinar el directorio de datos.
  RadioGroup2: TRadioGroup.- Grupo de botones de selección del número de estación, para determinar la máscara de selección de ficheros.
  Edit1: TEdit.- Ventana de edición para la visualización de la máscara de ficheros y del nombre del fichero
```

```
seleccionado.
  Edit2: TEdit.- Ventana de edición para la visualización del directorio seleccionado.
  Memo1: TMemo.- Cuadro de visualización de los parámetros de adquisición del fichero demultiplexado.
  FileListBox1: TFileListBox.- Cuadro de visualización de lista de ficheros del directorio seleccionado.
  BitBtn1: TBitBtn.- Botón de confirmación del fichero seleccionado (‘OK’).
*/
/* COMENTARIOS GENERALES:
   - Las ventanas de edición Edit1 y Edit2, que muestran el nombre del directorio y del fichero de datos que se
demultiplexa, deben tener la propiedad ReadOnly a TRUE, para que el fichero sólo se pueda seleccionar a
través de los botones.
   - Del mismo modo el cuadro Memo1, donde se muestran los parámetros de adquisición del fichero
demultiplexado, debe tener ReadOnly a TRUE, para evitar la posibilidad de modificar el fichero de cabecera
desde el programa.
   - Los valores por defecto de todos los elementos se asignan en la función DirectorioPorDefecto, que se
asigna al evento OnActivate del formulario.
*/

__fastcall TFormData2::TFormData2(TComponent* Owner)
   : TForm(Owner)
{
}
//---------------------------------------------------------------------------

/* GESTIÓN DEL EVENTO OnClick DE LOS COMPONENTES RadioGroup1 y RadioGroup2. Al
seleccionarse un PC nodal y una estación se determina el directorio de datos y la extensión del fichero de
datos correspondientes, que se muestran por las ventanas de edición Edit2 y Edit1, respectivamente. Además
en el componente FileListBox1 se muestran todos los ficheros del directorio seleccionado con esa extensión.
Esta función debe asociarse en tiempo de diseño al componente RadioGroup2, ya que por defecto sólo está
asociada a RadioGroup1.
*/
void __fastcall TFormData2::RadioGroup1Click(TObject *Sender)
{
   NumeroPCnodal = RadioGroup1->ItemIndex;
   NumeroEstacion = RadioGroup2->ItemIndex;
   sprintf(NomDirectorio, "C:\\SHARED\\DATA\\PCNODAL%i", NumeroPCnodal);
   Edit2->Text = NomDirectorio;
   sprintf(Mascara, "*.%i_%i", NumeroPCnodal, NumeroEstacion);
   FileListBox1->Mask = Mascara;
}
//---------------------------------------------------------------------------


/* GESTIÓN DEL EVENTO OnChange DEL COMPONENTE Edit2. Se hace coincidir el directorio cuyos
ficheros se muestran en FileListBox1 con el mostrado por la ventana de edición Edit2.
*/
void __fastcall TFormData2::Edit2Change(TObject *Sender)
{
   FileListBox1->Directory = Edit2->Text;
}
//---------------------------------------------------------------------------

/* GESTIÓN DEL EVENTO OnChange DEL COMPONENTE FileListBox1. Al seleccionar un fichero en
FileListBox1 se visualiza el nombre en la ventana de edición Edit1.
*/
void __fastcall TFormData2::FileListBox1Change(TObject *Sender)
{
```

```
   FileListBox1->FileEdit = Edit1;
}
//---------------------------------------------------------------------------
```

/* VALORES POR DEFECTO. Con esta función se toma por defecto el directorio de datos del nodal 0 y la máscara de ficheros del PC nodal 0 y estación 0 cuando se pulsa el botón '*Demultiplex data files*' del formulario principal. Para ello hay que asignar esta función al evento *OnActivate* del formulario *FormData2*.
*/

```
void __fastcall TFormData2::DirectorioPorDefecto(TObject *Sender)
{
   sprintf(NomDirectorio, "C:\\SHARED\\DATA\\PCnodal0\\");

   // Nombre del directorio al FileListBox1.
   FileListBox1->Directory = NomDirectorio;

   // Máscara inicial de ficheros: *.0_0
   FileListBox1->Mask = "*.0_0";

   // Nombre del directorio al cuadro de edición
   Edit2->Text = FormData2->FileListBox1->Directory;

   // Por defecto, marcados PC nodal 0 y estación 0
   RadioGroup1->ItemIndex = 0;
   RadioGroup2->ItemIndex = 0;
}
```

/* GESTIÓN DEL EVENTO OnClick DEL COMPONENTE BitBtn1.- Al pulsar el botón *BitBtn1* se realiza la demultiplexión del fichero seleccionado, sacándose por el cuadro *Memo1* el contenido del fichero de cabecera con los parámetros de adquisición o, en caso de que haya habido algún error, un mensaje informativo.
*/

```
void IncrementaContadorBytes(void)
{
   ContadorBytes++;
   if (ContadorBytes == BYTES_TRAMA) ContadorBytes = 0;
}

void __fastcall TFormData2::BitBtn1Click(TObject *Sender)
{
   FILE *fichDatos, *salida_hdr, *salida_data[3];
   char String[50], NomFich[10];
   char NombreFichero [14];
   char i;

   // LEE EL NOMBRE DEL FICHERO DE DATOS DE ENTRADA DE Edit1...
   sprintf(NombreFichero,"%s", Edit1->Text);

   //...SE QUEDA CON EL NOMBRE SIN EXTENSIÓN PARA NOMBRAR EL RESTO DE FICHEROS...
   i=0;
   while ((NombreFichero[i] != '.')&& (i<8))
   {
      NomFich[i] = NombreFichero[i];
      i++;
   }
   NomFich[i] = 0;
```

```
//...ABRE EL FICHERO DE DATOS DE ENTRADA...
if ((fichDatos = fopen(NombreFichero, "rb")) == NULL)
{
   Memo1->Lines->Text = "Unable to open data file";
}
else
{

// ...ABRE LOS FICHEROS DE DATOS DE SALIDA...
sprintf(NombreFichero, "%s.%u%uZ", NomFich, NumeroPCnodal, NumeroEstacion); // Canal Z
if ((salida_data[0] = fopen(NombreFichero, "wb")) == NULL)
{
   sprintf(String," Unable to open output data file %s", NombreFichero);
   Memo1->Lines->Text = String;
}

sprintf(NombreFichero, "%s.%u%uN", NomFich, NumeroPCnodal, NumeroEstacion); // Canal N-S
if ((salida_data[1] = fopen(NombreFichero, "wb")) == NULL)
{
   sprintf(String," Unable to open output data file %s", NombreFichero);
   Memo1->Lines->Text = String;
}
sprintf(NombreFichero, "%s.%u%uE", NomFich, NumeroPCnodal, NumeroEstacion); // Canal E-W
if ((salida_data[2] = fopen(NombreFichero, "wb")) == NULL)
{
   sprintf(String," Unable to open output data file %s", NombreFichero);
   Memo1->Lines->Text = String;
}

// ...ABRE EL FICHERO DE CABECERA...
sprintf(NombreFichero, "%s.HDR", NomFich);
if ((salida_hdr = fopen(NombreFichero, "w")) == NULL)
{
   sprintf(String," Unable to open data file %s", NombreFichero);
   Memo1->Lines->Text = String;
}

// ...LEE LA CABECERA DEL FICHERO DE ENTRADA...
fread(&ganancia, 1, 1, fichDatos);
fread(&fraccion, 1, 1, fichDatos);
fread(&segundo, 1, 1, fichDatos);
fread(&minuto, 1, 1, fichDatos);
fread(&hora, 1, 1, fichDatos);
fread(&dia, 2, 1, fichDatos);
fread(&year, 1, 1, fichDatos);
rewind(fichDatos);        // Vuelve al principio

//...ESCRIBE LA INFORMACIÓN EN EL FICHERO DE CABECERA...
fprintf(salida_hdr, "\n Sample rate: 100mps");
fprintf(salida_hdr, "\n\n Resolution: 24 bits");
fprintf(salida_hdr, "\n\n Gain: %u", (int)pow(2, ganancia));
fprintf(salida_hdr, "\n\n First sample acquisition time:");
fprintf(salida_hdr, "\n %02u:%02u:%02u.00, day %u of year 2%03u\n\n", hora, minuto, segundo, dia,
   year);

//...DEMULTIPLEXA LAS TRAZAS HASTA EL FIN DEL FICHERO DE ENTRADA...
```

```
   ContadorBytes = 0;
   do
   {
      //...DESCARTANDO LOS 8 BYTES DE CABECERA DE CADA TRAMA...
      if (ContadorBytes == 0)
      {
         for (i=0; i<8; i++)
         {
            if(!feof(fichDatos))
            {
               car = getc(fichDatos);
               IncrementaContadorBytes();
            }
         }
      }

      //...Y SEPARANDO LAS TRAZAS...
      for (i=0; i<3; i++)// Tres canales
      {
         for (j=0; j<3; j++)  // Tres bytes por dato
         {
            if(!feof(fichDatos))
            {
               car = getc(fichDatos);
               putc(car, salida_data[i]);
               IncrementaContadorBytes();
            }
         }
         putc(0x00, salida_data[i]);// Pone un byte a cero para que cada dato quede como entero largo (4 bytes)
      }
   }
   while((!feof(fichDatos))&&(!kbhit()));

    //...CIERRA LOS FICHEROS...
   fclose(fichDatos);
   fclose(salida_hdr);
   for (i=0; i<3; i++) fclose(salida_data[i]);

   //...Y SACA EL FICHERO DE CABECERA POR Memo1
   Memo1->Lines->LoadFromFile(NombreFichero);
}
```

ANEXO I.A.4.- PROGRAMAS DEL OSCILADOR PATRÓN

I.A.4.a. DIRECCIONES DE LOS VECTORES Y REGISTROS DE FUNCIÓN ESPECIAL DEL PIC16C74

Fichero:REGSC74.H

```
; Definición de vectores, registros de función especial, bits y otros para el PIC16C74. Con el objeto
;de ahorrar espacio, se han incluido sólo los registros de función especial y bits utilizados en
;nuestra aplicación.
;
NOLIST
;*****************************************************************************
;Vectores
;*****************************************************************************
RESET_V         EQU     0x0000  ; Dirección vector RESET
ISR_V           EQU     0x0004  ; Dirección vector interrupciones
;*****************************************************************************
;Registros de función especial
;*****************************************************************************
OPTION_R        EQU  81
PCL             EQU  02
STATUS          EQU  03
PORTA           EQU  05
TRISA           EQU  85
PORTB           EQU  06
TRISB           EQU  86
PORTC           EQU  07
TRISC           EQU  87
PORTD           EQU  08
TRISD           EQU  88
PORTE           EQU  09
TRISE           EQU  89
PCLATH          EQU  0A
INTCON          EQU  0B
PIR1            EQU  0C
PIE1            EQU  8C
PIR2            EQU  0D
PIE2            EQU  8D
TMR1L           EQU  0E
TMR1H           EQU  0F
T1CON           EQU  10
TMR2            EQU  11
T2CON           EQU  12
PR2             EQU  92
CCPR1L          EQU  15
CCPR1H          EQU  16
CCP1CON         EQU  17
CCPR2L          EQU  1B
CCPR2H          EQU  1C
CCP2CON         EQU  1D
```

```
ADRES           EQU  1E
ADCON1          EQU  9F
;
;*******************************************************************************
;Definiciones de bits
;*******************************************************************************
; Registro STATUS:
RP0             EQU     5
Z               EQU     2
DC              EQU     1
C               EQU     0
;
; Registro INTCON:
GIE             EQU     7
PEIE            EQU     6
T0IF            EQU     2
INTF            EQU     1
RBIF            EQU     0
;
; Registro PIR1:
CCP1IF          EQU     2
TMR1IF          EQU     0
;
; Registro PIE1:
CCP1IE          EQU     2
TMR1IE          EQU     0
;
; Registro OPTION:
RBPU            EQU     7
;
; Registro TRISC
T1CKI           EQU     0       ; Control del bit 0 del puerto C (o entrada del timer1)
CCP2            EQU     1       ; Control del bit 1 del puerto C (o entrada/salida del módulo CCP2)
CCP1            EQU     2       ; Control del bit 2 del puerto C (o entrada/salida del módulo CCP1)
;
; Registro T1CON:
TMR1ON          EQU     0
;
; Registro T2CON:
TMR2ON          EQU     2
T2CKPS1         EQU     1
T2CKPS0         EQU     0
;
;*******************************************************************************
; Otras definiciones
;*******************************************************************************
; Bits de control de destino
W               EQU     0       ; El destino del comando es W
F               EQU     1       ; El destino del comando es el registro
;
; FALSE y TRUE
FALSE           EQU     0
TRUE            EQU     1

LIST
```

I.A.4.b. DEFINICIONES PARA EL PROGRAMA DEL OSCILADOR PATRÓN

Fichero: RTC.H

```
;-------------------------------------------------------------------------------
;
; Fichero de cabecera para el programa del oscilador patrón. Incluye el valor de algunas constantes
;usadas para el cálculo de la corrección de la señal de los CADs y del RTC. Además se han tomado
;las definiciones necesarias para la inicialización del display de la nota de aplicación AN587 (ver
;Palmer y Fink, 1997), para no tener que incluir toda la librería que se definía allí.
;
NOLIST
;*******************************************************************************
;Asignación de pines
;*******************************************************************************
CodLento        EQU   D'3'      ; Salida del código lento: Pin 3 de PORTC
VerPPS          EQU   D'4'      ; Salida de monitorización de las interrupciones de PPS: pin 4
                                ;de PORTC
LCD_DATA        EQU   PORTD     ; Puertos y pines para el control del LCD
LCD_DATA_TRIS   EQU   TRISD
LCD_CNTL        EQU   PORTA
E               EQU   3         ; LCD Enable control line
R_W             EQU   2         ; LCD Read/Write control line
RS              EQU   1         ; LCD Register Select control line

SINC_CNTL       EQU   PORTC     ; Puerto para el control de la sincronización del RTC
SINC_CFG        EQU   TRISC     ; Registro de configuración del puerto
BotSinc         EQU   7         ; Botón de sincronización
BotMas          EQU   6         ; Botón de aumento de la variable seleccionada
BotMenos        EQU   5         ;  "   " disminución "   "        "
;
;*******************************************************************************
;Posición de los mensajes en el LCD
;*******************************************************************************
; LCD en operación normal:     LCD en la función de sincronización:
; ----------------      ----------------
; 17:08:23  362/99      Synchro Function
; VXO: 1228800.0Hz      Enter year:   99
; ----------------      ----------------
;
PosTexto1       EQU   B'10000000'   ; Posición de los mensajes (fila 1)
PosTexto2       EQU   B'11000000'   ; Posición de los mensajes (fila 2)
PosHora         EQU   B'10000000'   ; Posición de las horas en el display
PosMinuto       EQU   B'10000011'   ; Idem, minutos
PosSegundo      EQU   B'10000110'   ; Idem, segundos
PosDia          EQU   B'10001010'   ; Idem, días
PosYear         EQU   B'10001110'   ; Idem, año
PosSinc2        EQU   B'11001110'   ; Pos. para sincroniz. variables de 2 dígitos
PosSinc3        EQU   B'11001101'   ; Idem, 3 dígitos (día)
PosFrec         EQU   B'11000101'   ; Pos. de la frecuencia del VXO
;
```

```
;*******************************************************************************
;Constantes
;*******************************************************************************
Centesimas0             EQU     D'25'           ; Anchura, en centesimas de segundo, de los pulsos de segundo
Centesimas1             EQU     D'60'           ;correspondientes a un 0 y un 1, respectivamente.
;
; CALCULO DE LAS CTES. PARA CREAR UN RETARDO DE 0.01s
; Para crear un retardo de 0.01s es necesario que se cumpla:
;
;       0.01s = Nc * Tc                 (1)
;
; siendo Nc el número de ciclos de instrucción y Tc su periodo:
;
;       Tc = To / CteCristal, con To = Periodo máximo (en nuestro caso, el periodo para un cristal de fo = 1MHz, porque no pensamos ponerle uno más lento). Como To = 4*(1/fo) = 4/1000000 (ya que cada ciclo de instrucción son 4 ciclosde reloj), tenemos
;
;       Tc = 4/(1000000*CteCristal)             (2)
;
;  Por otra parte, Nc la calculamos a partir de la función Centesimas, en el programa OSCILAD.ASM.
;  Tal y como está pensada, la función invierte:
;       Bucle1  10 ciclos en cada salto a 'Bucle1'
;               12 ciclos en cada salto a 'Bucle0'
;       Bucle0  8 ciclos
;
; Esto equivale a 10 ciclos en cada Bucle1 y 10 en cada Bucle0. Además hay dos ciclos al entrar en ;la función y tres más al salir de ella, pero son despreciables y no los vamos a tener en cuenta. Por ;tanto, como el Bucle0 se ejecuta 'CteCristal' veces y el Bucle1 'CteCristal*B1' veces, queda:
;
;       Nc = 10*CteCristal + 10*CteCristal*B1 = 10*CteCristal(1+B1)
;
; Sustituyendo (3) y (2) en (1) nos queda:
;
;       0.01s = 10*(1+B1)*(4/1000000), B1=249 independientemente del valor de CteCristal. Por ;tanto, hay que definir:

B1              EQU             D'249'
;CteCristal     EQU             D'1'            ; Cristal de 1MHz (comentar los no usados)
;CteCristal     EQU             D'5'            ; Cristal de 5MHz
CteCristal      EQU             D'10'           ; Cristal de 10MHz
;CteCristal     EQU             D'15'           ; Cristal de 15.56MHz
;CteCristal     EQU             D'20'           ; Cristal de 20MHz
;
;*******************************************************************************
; Definiciones necesarias para las funciones primitivas del display LCD (tomadas de la nota de ;aplicación AN587 (Palmer y Fink, 1997))
;*******************************************************************************
Dev_Freq                EQU     D'10000000'             ; Device Frequency is 4 MHz
LCD_INIT_DELAY          EQU     (HIGH ((( Dev_Freq / 4 ) * D'46' / D'10000' ) / 3 ) ) + 1

MSD             EQU     0x033           ; Temporary register, Holds Most Significant Digit of BIN to BCD
                                        ;conversion
LSD             EQU     0x034           ;Temporary register, Holds Least Significant Digit of BIN to BCD
                                        ;conversion
```

```
; LCD Module commands
;
DISP_ON         EQU   0x00C     ; Display on
DISP_ON_C       EQU   0x00E     ; Display on, Cursor on
DISP_ON_B       EQU   0x00F     ; Display on, Cursor on, Blink cursor
DISP_OFF        EQU   0x008     ; Display off
CLR_DISP        EQU   0x001     ; Clear the Display
ENTRY_INC       EQU   0x006
ENTRY_INC_S     EQU   0x007
ENTRY_DEC       EQU   0x004
ENTRY_DEC_S     EQU   0x005
DD_RAM_ADDR     EQU   0x080     ; Least Significant 7-bit are for address
DD_RAM_UL       EQU   0x080     ; Upper Left coner of the Display

LIST
```

I.A.4.c. PROGRAMA DEL OSCILADOR PATRÓN

Fichero: OSCILAD.ASM

```
;
; Versión definitiva del programa del oscilador patrón. En esta versión se han reducido el número
;de macros respecto a la versión anterior para ahorrar espacio de programa (y poder así incluirlo
;todo en la primera página (2kB) de memoria). Para ello se han sustituido algunas de las macros
;definidas antes por funciones, incluyendo, en caso de ser necesaria, una pequeña macro para
;pasarle las variables.

        LIST P=16C74A

        INCLUDE <REGSC74.h>
        INCLUDE <RTC.H>

;*****************************************************************************
; Variables
;*****************************************************************************
PosVar          EQU     0x20
;
; Variables de tiempo:
Segundo         EQU     PosVar+0
Minuto          EQU     PosVar+1
Hora            EQU     PosVar+2
LSBDia          EQU     PosVar+3
MSBDia          EQU     PosVar+4
Year            EQU     PosVar+5
DespMinuto      EQU     PosVar+6
DespHora        EQU     PosVar+7
DespLSBDia      EQU     PosVar+8
DespMSBDia      EQU     PosVar+.9
DespYear        EQU     PosVar+.10
LSBDiasYear     EQU     PosVar+.11
;
; Variables para crear los retardos del código lento y de las esperas:
Retardo         EQU     PosVar+.12
indice0         EQU     PosVar+.13
indice1         EQU     PosVar+.14
ContTiempo      EQU     PosVar+.15
;
; Variables para el display LCD:
; Para las primitivas de control del LCD, de la nota de aplicación AN587:
CHAR            EQU     PosVar+.16
TEMP            EQU     PosVar+.17
; Para las funciones de control del LCD:
Decimas         EQU     PosVar+.18
Unidades        EQU     PosVar+.19
Decenas         EQU     PosVar+.20
Centenas        EQU     PosVar+.21
UnsMillar       EQU     PosVar+.22
DecsMillar      EQU     PosVar+.23
```

```
CentsMillar     EQU     PosVar+.24
Millones        EQU     PosVar+.25
HoraLCD         EQU     PosVar+.26
MinutoLCD       EQU     PosVar+.27
Offset          EQU     PosVar+.28      ; Para la lectura de tablas
Limite          EQU     PosVar+.29      ; Para fijar el límite de cada variable
;
; Variables para gestionar las interrupciones del Timer 1:
ContVueltas     EQU     PosVar+.30
RecContVueltas  EQU     PosVar+.31
;
; Variables para el paso de parámetros a las funciones
CteDM           EQU     PosVar+.32
CteUM           EQU     PosVar+.33
CteC            EQU     PosVar+.34
CteD1           EQU     PosVar+.35
CteU            EQU     PosVar+.36
CteD2           EQU     PosVar+.37
Pos             EQU     PosVar+.38
Var             EQU     PosVar+.39
;
; Otras variables:
Temporal        EQU     PosVar+.40
Flag            EQU     PosVar+.41      ; Para definir indicadores (flags)
IntPPS          EQU     0               ; Bit 0 de Flag: Int. de PPS
IntADCs         EQU     1               ; Bit 1: Gestión ADCs CLK
Flanco          EQU     2               ; Bit 2: Flanco de disparo de PPS
Testigo         EQU     3               ; Bit 3: Testigo de interrupciones
;
;*********************************************************************************
; Funciones y Macros
;*********************************************************************************
                ORG             0x030
;-------------------------------------------------------------------------------------------------------------
; Primitivas para el LCD (sacadas de la nota de aplicación AN587)
;-------------------------------------------------------------------------------------------------------------
;
; SEND_CHAR - Sends character contained in register W to LCD.
; This routine sends the entire character to the PORT .
; The data is transmitted on the PORT<7:0> pins.
;
SEND_CHAR
                BCF             STATUS, RP0     ; Banco 0
                MOVWF           CHAR            ; Character to be sent is in W
                CALL            BUSY_CHECK      ; Wait for LCD to be ready
                BCF             LCD_CNTL, R_W   ; Set LCD in read mode
                BSF             LCD_CNTL, RS    ; Set LCD in data mode
                BSF             LCD_CNTL, E     ; toggle E for LCD
                MOVF            CHAR, w
                MOVWF           LCD_DATA        ; Send data to LCD (puesto aqui para crear el retardo
                BCF             LCD_CNTL, E     ;necesario cuando usemos cristales de frecuencia alta)
                RETURN
```

```
; SEND_CMD - Sends command contained in register W to LCD.
; This routine sends the entire character to the PORT .
; The data is transmitted on the PORT<7:0> pins.
;
SEND_CMD
                BCF             STATUS, RP0             ; Banco 0
                MOVWF           CHAR                    ; Command to be sent is in W
                CALL            BUSY_CHECK              ; Wait for LCD to be ready
                BCF             LCD_CNTL, R_W           ; Set LCD in read mode
                BCF             LCD_CNTL, RS            ; Set LCD in command mode
                BSF             LCD_CNTL, E             ; toggle E for LCD
                MOVF            CHAR, w
                MOVWF           LCD_DATA                ; Send data to LCD (puesto aqui para crear el retardo
                BCF             LCD_CNTL, E             ;necesario cuando usemos cristales de frecuencia alta)
                RETURN
;
; BUSY_CHECK -This routine checks the busy flag, returns when not busy.
; Affects:
;    TEMP - Returned with busy/address
;
BUSY_CHECK
                BSF             STATUS,RP0              ; Select Register page 1 (Banco 1)
                MOVLW           0xFF                    ; Set port_D for input
                MOVWF           LCD_DATA_TRIS
                BCF             STATUS, RP0             ; Select Register page 0
                BCF             LCD_CNTL, RS            ; Set LCD for command mode
                BSF             LCD_CNTL, R_W           ; Setup to read busy flag
                BSF             LCD_CNTL, E             ; Set E high
                NOP                                     ; Para crear el retardo necesario cuando usemos cristales
                                                        ;de frecuencia alta (p.ej.10MHz)
                BCF             LCD_CNTL, E             ; Set E low
                MOVF            LCD_DATA, w             ; Read busy flag, DDram address
                MOVWF           TEMP
                BTFSC           TEMP, 7                 ; Check busy flag, high=busy
                GOTO            BUSY_CHECK
                BCF             LCD_CNTL, R_W
                BSF             STATUS, RP0             ; Select Register page 1 (Banco 1)
                MOVLW           0x00
                MOVWF           LCD_DATA_TRIS           ; Set port_D for output
                BCF             STATUS, RP0             ; Select Register page 0 (Banco 0)
                RETURN
;
; IniciaLCD - Función para la inicialización del display LCD.
;
IniciaLCD
                CLRF            STATUS                  ; Do initialization (Bank 0)
                ;CLRF           INTCON
                ;CLRF           PIR1
                BSF             STATUS, RP0             ; Bank 1
                MOVLW           0x00                    ; The LCD module does not like to work w/ weak pullups
                MOVWF           OPTION_R
                ;CLRF           PIE1                    ; Disable all peripheral interrupts
                MOVLW           0xFF
                MOVWF           ADCON1                  ; Port A is Digital.;
                BCF             STATUS, RP0             ; Bank 0
                ;CLRF           PORTA                   ; ALL PORT output should output Low.
```

```
                ;CLRF       PORTB
                ;CLRF       PORTC
                ;CLRF       PORTD
                ;CLRF       PORTE
                BCF         T1CON, TMR1ON       ; Timer 1 is NOT incrementing
;
                BSF         STATUS, RP0         ; Select Bank 1
                CLRF        TRISA               ; RA5 -  0 outputs
                ;MOVLW      0xF0
                ;MOVWF      TRISB               ; RB7 - 4 inputs, RB3 - 0 outputs
                ;CLRF       TRISC               ; RC Port are outputs
                ;BSF        TRISC, T1OSO        ; RC0 needs to be input for the oscillator to function
                CLRF        TRISD               ; RD Port are outputs
                CLRF        TRISE               ; RE Port are outputs
                ;BSF        PIE1, TMR1IE        ; Enable TMR1 Interrupt
                BSF         OPTION_R, RBPU      ; Disable PORTB pull-ups
                BCF         STATUS, RP0         ; Select Bank 0
;
; Initilize the LCD Display Module
;
                CLRF        LCD_CNTL            ; ALL PORT output should output Low.

DISPLAY_INIT
                MOVLW       0x038               ; Command for 8-bit interface
                MOVWF       LCD_DATA
                BSF         LCD_CNTL, E
                NOP
                BCF         LCD_CNTL, E
;
; This routine takes the calculated times that the delay loop needs to be executed, based on the LCD_INIT_DELAY
;EQUate that includes the frequency of operation. These uses registers before they are needed to store the time.
;
LCD_DELAY
                MOVLW       LCD_INIT_DELAY  ;
                MOVWF       MSD                 ; Use MSD and LSD Registers to Initilize LCD
                CLRF        LSD
LOOP2
                DECFSZ      LSD, F              ; Delay time = MSD * ((3 * 256) + 3) * Tcy
                GOTO        LOOP2
                DECFSZ      MSD, F
END_LCD_DELAY
                GOTO        LOOP2

;Command sequence for 2 lines of 5x7 characters
CMD_SEQ
                MOVLW       0X038
                MOVWF       LCD_DATA            ; This code for both 4-bit and 8-bit modes
                BSF         LCD_CNTL, E
                NOP
                BCF         LCD_CNTL, E
;
; Busy Flag should be valid after this point
;
                MOVLW       DISP_ON
                CALL        SEND_CMD
                MOVLW       CLR_DISP
```

```
                CALL    SEND_CMD
                MOVLW   ENTRY_INC
                CALL    SEND_CMD
                MOVLW   DD_RAM_ADDR
                CALL    SEND_CMD

                RETURN
;-----------------------------------------------------------------------------------------------------------
; Funciones para el LCD, desarrolladas a partir de las primitivas
;-----------------------------------------------------------------------------------------------------------
;
; Bin2ASC - Función para pasar las variables de tiempo a ASCII, para poder sacarlas por el display LCD. La variable
;a convertir debe haber sido previamente introducida en 'Unidades'. Esta función la pasa primero a tres dígitos BCD
;(Centenas, Decenas y Unidades) y luego les suma 48, para convertirlos en ASCII.
;
Bin2ASC
                BCF     STATUS, RP0    ; Banco 0
                MOVLW   .10
                SUBWF   Unidades, W
                BTFSS   STATUS, C
                GOTO    RestaCentenas
                MOVWF   Unidades
                INCF    Decenas, F
                GOTO    Bin2ASC
RestaCentenas
                MOVLW   .10
                SUBWF   Decenas, W
                BTFSS   STATUS, C
                GOTO    BCD2ASCII
                MOVWF   Decenas
                INCF    Centenas, F
                GOTO    RestaCentenas
BCD2ASCII
                MOVLW   .48
                ADDWF   Centenas, F
                ADDWF   Decenas, F
                ADDWF   Unidades, F
                RETURN
;
; WrLCDChar - Escribe el carácter 'Caracter' en la posición 'Posicion' del  LCD.
; Se ha dejado como macro, porque ocupa más espacio pasándola a función.
;
WrLCDChar       MACRO   Posicion, Caracter
                MOVLW   Posicion
                CALL    SEND_CMD
                MOVLW   Caracter
                CALL    SEND_CHAR
                ENDM
;
; WrLCD2 - Escribe la variable de dos dígitos (segundo, minuto, hora, year) indicada por 'Var' en el display, en la
;posición indicada por la variable 'Pos'. Para introducir ambas se usa la macro IntroPosVar.
;
IntroPosVar     MACRO   Posicion, Variable
                MOVLW   Posicion
                MOVWF   Pos
                MOVF    Variable, W
```

```
                MOVWF   Var
                ENDM
;
WrLCD2
                CLRF    Centenas        ; Inicializamos las variables del  display
                CLRF    Decenas
                MOVF    Var, W
                MOVWF   Unidades
                CALL    Bin2ASC
                BCF     STATUS, RP0     ; Banco 0
                MOVF    Pos, W
                CALL    SEND_CMD
                MOVF    Decenas, W
                CALL    SEND_CHAR
                MOVF    Unidades, W
                CALL    SEND_CHAR
                RETURN
;
; WrLCD3 - Igual que WrLCD2 para variables de tres dígitos.
; Para la introducción de los parámetros se usa la macro IntroPosVar definida más arriba.
;
WrLCD3
                BCF     STATUS, RP0     ; Banco 0
                CLRF    Centenas
                CLRF    Decenas
                MOVF    Var, W
                MOVWF   Unidades
                CALL    Bin2ASC
                MOVF    Pos, W
                CALL    SEND_CMD
                MOVF    Centenas, W
                CALL    SEND_CHAR
                MOVF    Decenas, W
                CALL    SEND_CHAR
                MOVF    Unidades, W
                CALL    SEND_CHAR
                RETURN
;
; WrLCDDia - Escribe el día en la posición indicada por 'Pos' del display LCD. Al usar dos variables para el día no
;sirve la función 'WrLCD3' definida antes.
; Para introducir la posición se usa la macro 'IntroPos'.
;
IntroPos        MACRO   Posicion
                MOVLW   Posicion
                MOVWF   Pos
                ENDM
;
WrLCDDia
                BCF     STATUS, RP0             ; Banco 0
                CLRF    Centenas
                CLRF    Decenas
                CLRF    Unidades
                DECFSZ  MSBDia, W
                GOTO    Seguir
                MOVLW   .2
                MOVWF   Centenas
```

```
                MOVLW   .5
                MOVWF   Decenas
                MOVLW   .6
                MOVWF   Unidades
Seguir
                MOVF    LSBDia, W
                ADDWF   Unidades, F
                CALL    Bin2ASC
                MOVF    Pos, W
                CALL    SEND_CMD
                MOVF    Centenas, W
                CALL    SEND_CHAR
                MOVF    Decenas, W
                CALL    SEND_CHAR
                MOVF    Unidades, W
                CALL    SEND_CHAR
                RETURN
;
; WrLCDText - Escribe en la posición 'Posicion' del LCD el texto almacenado en 'Tabla'.
;
WrLCDText       MACRO   Posicion, Tabla
                LOCAL   LeeTabla, FinWrLCDText
                MOVLW   Posicion
                CALL    SEND_CMD
                BCF     STATUS, RP0         ; Banco 0
                MOVLW   HIGH Tabla          ; Mete los 5 bits más significativos de la dirección de la
                MOVWF   PCLATH              ;tabla en los bits <13:8> del contador de programa.
                CLRF    Offset
LeeTabla
                MOVF    Offset, W
                CALL    Tabla
                ANDLW   0FFh
                BTFSC   STATUS, Z           ; Si el valor que se ha metido en W es cero, se ha llegado
                GOTO    FinWrLCDText        ;se ha llegado al fin de la tabla y se sale de la macro.
                CALL    SEND_CHAR
                INCF    Offset, F           ; Apunta al siguiente carácter
                GOTO    LeeTabla
FinWrLCDText
                ENDM
;
; WrLCDFrec - Escribe la frecuencia del VXO en el display. La posición debe introducirse previamente en la variable
;'Pos', y los dígitos de frecuencia deben estar ya convertidos a ASCII, en las variables DecsMillar, UnsMillar, Centenas,
;Decenas, Unidades y Decimas.
;
WrLCDFrec
                MOVF    Pos, W
                CALL    SEND_CMD
                MOVF    Millones, W
                CALL    SEND_CHAR
                MOVF    CentsMillar, W
                CALL    SEND_CHAR
                MOVF    DecsMillar, W
                CALL    SEND_CHAR
                MOVF    UnsMillar, W
                CALL    SEND_CHAR
                MOVF    Centenas, W
```

```
                CALL    SEND_CHAR
                MOVF    Decenas, W
                CALL    SEND_CHAR
                MOVF    Unidades, W
                CALL    SEND_CHAR
                MOVLW   .46
                CALL    SEND_CHAR
                MOVF    Decimas, W
                CALL    SEND_CHAR
                RETURN
;
;-------------------------------------------------------------------------------------------------------------
; Funciones y macros para la selección y generación del retardo
;-------------------------------------------------------------------------------------------------------------
;
; AsignaRet - Si 'Segundo' es mayor o igual que 'cte1' y menor que 'cte2', asigna 'retardo' a la variable global 'Retardo'
;
AsignaRet       MACRO   cte1, cte2, retardo
                LOCAL   FinMacro
                MOVLW   cte1
                SUBWF   Segundo, W
                BTFSS   STATUS, C
                GOTO    FinMacro
                MOVLW   cte2
                SUBWF   Segundo, W
                BTFSC   STATUS, C
                GOTO    FinMacro
                MOVLW   retardo
                MOVWF   Retardo
FinMacro
                ENDM
;
; BitTest - Si 'Segundo' es mayor o igual que 'cte1' y menor que 'cte2' mira el bit menos significativo de 'variable' (y
;deja 'variable' desplazada para la siguiente llamada). Además, asigna el valor de dicho bit a 'Retardo'.
;
BitTest         MACRO   cte1, cte2, variable
                LOCAL   FinMacro, Ret0
                MOVLW   cte1
                SUBWF   Segundo, W
                BTFSS   STATUS, C
                GOTO    FinMacro
                MOVLW   cte2
                SUBWF   Segundo, W
                BTFSC   STATUS, C
                GOTO    FinMacro
                RRF     variable, F
                BTFSS   STATUS, C
                GOTO    Ret0            ; Retardo 0
                MOVLW   D'1'            ; Retardo 1
                MOVWF   Retardo
                GOTO    FinMacro
Ret0
                CLRF    Retardo
FinMacro
                ENDM
```

```
; Centesima - Genera un retardo de una centésima de segundo, en función de las constantes 'CteCristal' y 'B1'. El
;cálculo de los valores de estas constantes está hecho en la librería RTC.H.
;
Centesima
                BCF             STATUS, RP0             ; Banco 0
                MOVLW           CteCristal
                MOVWF           indice0
Bucle0
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                MOVLW           B1
                MOVWF           indice1
Bucle1
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                DECFSZ          indice1, F
                GOTO            Bucle1
                DECFSZ          indice0, F
                GOTO            Bucle0
                RETURN
;
; Delay - Macro que crea un retardo de duración 'Centesimas'
;
Delay           MACRO           Centesimas
                LOCAL           OtraCentesima
                BCF             STATUS, RP0     ; Banco 0
                MOVLW           Centesimas
                MOVWF           ContTiempo
OtraCentesima
                CALL            Centesima
                DECFSZ          ContTiempo, F
                GOTO            OtraCentesima
                ENDM
;
;-------------------------------------------------------------------------------------------------------------------
; Funciones para la sincronización del RTC
;-------------------------------------------------------------------------------------------------------------------
;
; Sinc - Macro para introducir el valor de las variables de tiempo en el PIC. Esto se realizará mediante los pulsadores
;BotSinc, BotMas y BotMenos.
;
Sinc            MACRO           variable
                LOCAL           ChkHour, ChkMinSec, ChkBotones, Espera1, ChkBotMenos, WrLCD, Espera2,
                                ChkBotSinc

                ; Determinación del límite superior de la variable
                BCF             STATUS, RP0             ; Banco 0
```

```
                MOVLW   variable
                SUBLW   Year
                BTFSS   STATUS, Z
                GOTO    ChkHour
                MOVLW   .100            ; El año sólo puede llegar a 99
                MOVWF   Limite
                GOTO    ChkBotones
ChkHour
                MOVLW   variable
                SUBLW   Hora
                BTFSS   STATUS, Z
                GOTO    ChkMinSec
                MOVLW   .24             ; La hora sólo puede llegar a 23
                MOVWF   Limite
                GOTO    ChkBotones
ChkMinSec
                MOVLW   .60             ; Si no es ninguna de las otras, tiene que ser segundo o
                MOVWF   Limite          ;minuto. En cualquier caso, el máximo es 59.
ChkBotones
                BTFSS   SINC_CNTL, BotMas
                GOTO    ChkBotMenos
                INCF    variable, F     ; Incrementa la variable y, si ha llegado al límite, la pone
                MOVF    variable, W     ;a 0
                SUBWF   Limite, W
                BTFSC   STATUS, Z
                CLRF    variable
                GOTO    WrLCD
ChkBotMenos
                BTFSS   SINC_CNTL, BotMenos
                GOTO    ChkBotSinc
                DECF    variable, F     ; Decrementa la variable y, si es menor que cero, la pone
                MOVF    variable, W     ;a límite-1
                SUBWF   Limite, W
                BTFSC   STATUS, C
                GOTO    WrLCD
                MOVFW   Limite
                MOVWF   variable
                DECF    variable, F     ; Hay que meter el límite-1
WrLCD
                IntroPosVar PosSinc2, variable
                CALL    WrLCD2
                Delay   .20             ; Espera 20 centesimas entre cada inc. (o dec.) de la
                                        ;variable que se esté actualizando
ChkBotSinc
                BTFSS   SINC_CNTL, BotSinc
                GOTO    ChkBotones
                ENDM
;
; SincDia - Al implicar dos variables (MSBDia y LSBDia) la sincronización del día no se puede hacer con la misma
;macro 'Sinc' y es necesaria una función específica.
;
IncremDia
                BCF     STATUS, RP0     ; Banco 0
                MOVF    LSBDia, W       ; 365 → MSBDia=1, LSBDia=109. Si LSBDia es
                SUBLW   .109            ;distinto de 109, o si vale 109 pero MSBDia vale 0,
```

```
                BTFSS       STATUS, Z           ;simplemente se incrementa LSBDia. En caso contrario
                GOTO        Incrementar         ;(que corresponde al día 365) se  inicializan ambas
                DECFSZ      MSBDia, W           ;variables.
                GOTO        Incrementar
                CLRF        MSBDia
                CLRF        LSBDia
Incrementar
                INCF        LSBDia, F           ; Si LSBDia se desborda (poniéndose a 0) hay que poner
                MOVF        LSBDia, F           ;MSBDia a 1.
                BTFSC       STATUS, Z
                INCF        MSBDia, F
                RETURN

DecremDia
                BCF         STATUS, RP0         ; Banco 0
                MOVF        LSBDia, F           ; Si LSBDia no es cero, simplemente se decrementa.
                BTFSS       STATUS, Z           ;Si es cero, se mira MSBDia: si vale uno se pone a cero
                GOTO        Decrementar         ;y se decrementa LSBDia, y si vale cero se pone a uno
                MOVF        MSBDia, F           ;y se carga LSBDia con el valor 109 (para representar
                BTFSS       STATUS, Z           ;365).
                GOTO        MSBDia_0
                MOVLW       .1
                MOVWF       MSBDia
                MOVLW       .109
                MOVWF       LSBDia
                RETURN
MSBDia_0
                CLRF        MSBDia
Decrementar
                DECFSZ      LSBDia, F           ; Si al final del proceso los valores son 0 para las dos
                RETURN                          ; variables, hay que volver al principio de la función (ya
                MOVF        MSBDia, F           ;que no existe el día 0).
                BTFSC       STATUS, Z
                GOTO        DecremDia
                RETURN

SincDia
ChkBotones
                BCF         STATUS, RP0         ; Banco 0
                BTFSS       SINC_CNTL, BotMas
                GOTO        ChkBotMenos
                CALL        IncremDia
                GOTO        WrLCD
ChkBotMenos
                BTFSS       SINC_CNTL, BotMenos
                GOTO        ChkBotSinc
                CALL        DecremDia
WrLCD
                IntroPos    PosSinc3
                CALL        WrLCDDia
                Delay       .20                 ; Espera 20 centesimas entre cada incremento o
                                                ;decremento
ChkBotSinc
                BTFSS       SINC_CNTL, BotSinc
                GOTO        ChkBotones
                RETURN
```

```
;
; Sincronizacion - Función para introducir los valores de las variables de tiempo, a través de los pulsadores BotSinc,
;BotMas y BotMenos. Es llamada en el bucle principal.
;
Sincronizacion
        BCF         STATUS, RP0         ; Banco 0
        BTFSS       SINC_CNTL, BotSinc
        RETURN
        BSF         STATUS, RP0         ; Banco 1
        BCF         PIE1, CCP1IE        ; Deshabilita la interrupcion de CCP1
        BCF         STATUS, RP0         ; Banco 0

        WrLCDText   PosTexto1, MensSincro
        WrLCDText   PosTexto2, MensSincA
        IntroPosVar PosSinc2, Year
        CALL        WrLCD2              ; Visualiza el valor inicial
        Delay       .100                ; Entre cada dos comprobaciones del botón de sinc. debe
                                        ;haber un retardo, para que cada pulsado no se interprete
                                        ;como varios.
        Sinc        Year
        CALL        CheckBisiesto       ; Comprueba si el año es bisiesto para guardar el número
                                        ;de días
        WrLCDText   PosTexto2, MensSincD
        IntroPos    PosSinc3
        CALL        WrLCDDia
        Delay       .100
        CALL        SincDia

        WrLCDText   PosTexto2, MensSincH
        IntroPosVar PosSinc2, Hora
        CALL        WrLCD2
        Delay       .100
        Sinc        Hora

        WrLCDText   PosTexto2, MensSincM
        IntroPosVar PosSinc2, Minuto
        CALL        WrLCD2
        Delay       .100
        Sinc        Minuto

        WrLCDText   PosTexto2, MensSincS
        IntroPosVar PosSinc2, Segundo
        CALL        WrLCD2
        Delay       .100
        Sinc        Segundo

        WrLCDText   PosTexto1, MensInicioA
        WrLCDText   PosTexto2, MensInicioB
        IntroPosVar PosHora, Hora
        CALL        WrLCD2
        IntroPosVar PosMinuto, Minuto
        CALL        WrLCD2
        IntroPosVar PosSegundo, Segundo
        CALL        WrLCD2
        IntroPos PosDia
        CALL        WrLCDDia
```

```
                IntroPosVar     PosYear, Year
                CALL            WrLCD2

                BSF             STATUS, RP0             ; Banco 1
                BSF             PIE1, CCP1IE            ; Habilita la interrupcion de CCP1
                BCF             STATUS, RP0             ; Banco 0

                RETURN
;
; EligeFlanco - Función para elegir el flanco (ascendente o descendente) de la señal de PPS que disparará la
;interrupción. Para ello hace polling al botón '-', y si se ha pulsado espera que se pulse el botón '+' (para cambiar de
;elección) o 'Synchro' (para salir, sincronizando previamente el RTC).
;
EligeFlanco
                BCF             STATUS, RP0             ; Banco 0
                BTFSS           SINC_CNTL, BotMenos
                RETURN

                BSF             STATUS, RP0             ; Banco 1
                BCF             PIE1, CCP1IE            ; Deshabilita la interrupcion de CCP1 (PPS)
                BCF             STATUS, RP0             ; Banco 0
                WrLCDText       PosTexto1, MensFlanco
                WrLCDText       PosTexto2, MensInstrucc
ChkBotones1
                Delay           .50
                BTFSS           SINC_CNTL, BotMas
                GOTO            ChkBotSinc1
                BTFSS           Flag, Flanco
                GOTO            PoneFlancoAsc
                BCF             Flag, Flanco
                WrLCDText       PosTexto1, MensFlancoDesc
                GOTO            ChkBotSinc1
PoneFlancoAsc
                BSF             Flag, Flanco
                WrLCDText       PosTexto1, MensFlancoAsc
ChkBotSinc1
                BTFSS           SINC_CNTL, BotSinc
                GOTO            ChkBotones1

                MOVLW           0x05            ; Si 'Flanco' ha quedado a 1, pulso ascendente (0x05). En otro
                BTFSS           Flag, Flanco    ;caso, descendente (0x04).
                MOVLW           0x04
                MOVWF           CCP1CON
                                                ; El último pulsado de BotSinc sirve también para entrar en
                                                ;'Sincronizacion', donde además se habilita la ;interrupción.
                RETURN
;
; Configuracion - Función para configurar el sistema. Primero comprueba si se quiere elegir el flanco de los PPS y
;luego si se desea resincronizar el RTC. La frecuencia de la señal del reloj de los ADCs se elegirá por hardware,
;mediante jumpers.
;
Configuracion
                CALL            EligeFlanco
                CALL            Sincronizacion
                RETURN
```

```
; Presentacion - Saca los mensajes iniciales por el display LCD.
;
Presentacion
                WrLCDText       PosTexto1, MensPres1A
                WrLCDText       PosTexto2, MensPres1B
                Delay           .200
                WrLCDText       PosTexto1, MensPres2A
                WrLCDText       PosTexto2, MensPres2B
                Delay           .200
                WrLCDText       PosTexto1, MensFlancoDesc
                WrLCDText       PosTexto2, FilaEnBlanco
                Delay           .200
                WrLCDText       PosTexto1, MensInicioA
                WrLCDText       PosTexto2, MensInicioB
                RETURN
;
;-------------------------------------------------------------------------------------------------------------
; Funciones para la gestión del código lento
;-------------------------------------------------------------------------------------------------------------
;
; GestCodLento - Función principal de la gestión del código lento, que llama a las demás.
;
GestCodLento
                CALL            IncRTC          ; Incrementa el Reloj de Tiempo Real.
                CALL            CheckSegundo
                CALL            Display         ; Saca la hora por el display LCD.
                MOVLW           D'1'            ; Si es el segundo 1, lee la hora y la pone en variables
                SUBWF           Segundo, W      ;temporales para poder operar con ellas.
                BTFSC           STATUS, Z
                CALL            LeerRTC
                CALL            SeleccRet       ; Selecciona el retardo para el codigo de tiempo real.
                CALL            Espera          ; Espera el tiempo correspondiente al retardo
                                                ;seleccionado.
                RETURN
;
; IncRTC - Incrementa el reloj software de tiempo real.
;
IncRTC
                INCF            Segundo, F      ; Incrementa el segundo
                MOVLW           D'60'
                SUBWF           Segundo, W
                BTFSS           STATUS, Z
                GOTO            FinIncRTC

                CLRF            Segundo         ; Si es necesario, incrementa el minuto
                INCF            Minuto, F
                MOVLW           D'60'
                SUBWF           Minuto, W
                BTFSS           STATUS, Z
                GOTO            FinIncRTC

                CLRF            Minuto          ; Si es necesario, incrementa la hora
                INCF            Hora, F
                MOVLW           D'24'
                SUBWF           Hora, W
                BTFSS           STATUS, Z
```

```
                GOTO        FinIncRTC

                CLRF        Hora            ; Si es necesario, incrementa el día
                INCFSZ      LSBDia, F
                GOTO        Seguir1
                INCF        MSBDia, F
Seguir1
                MOVF        LSBDiasYear, W  ; Cambia si el año es bisiesto
                SUBWF       LSBDia, W
                BTFSS       STATUS, Z
                GOTO        FinIncRTC
                BTFSS       MSBDia, 0
                GOTO        FinIncRTC

                CLRF        MSBDia          ; Si es necesario, incrementa el año
                MOVLW       D'1'
                MOVWF       LSBDia
                INCF        Year, F
                MOVLW       D'100'
                SUBWF       Year, W
                BTFSC       STATUS, Z
                CLRF        Year
                CALL        CheckBisiesto   ; Comprueba si el año es bisiesto para saber el número
                                            ;de días.
FinIncRTC
                RETURN
;
; CheckBisiesto - Comprueba si el año es bisiesto y asigna el valor correspondiente a la variable 'LSBDiasYear.'
;
CheckBisiesto
                MOVF        Year, W         ; Para disparar los flags de STATUS
                BTFSC       STATUS, Z
                GOTO        NoBisiesto
                MOVWF       Temporal        ; Se mueve a otra variable para poder operar con ella.
Again
                MOVLW       .4
                SUBWF       Temporal, F     ; F-W = Temporal-4 → Temporal
                BTFSC       STATUS, Z
                GOTO        Bisiesto
                BTFSC       STATUS, C       ; C=0: negativo, C=1: positivo o cero.
                GOTO        Again
NoBisiesto
                MOVLW       .110
                GOTO        FinCheckBis
Bisiesto
                MOVLW       .111
FinCheckBis
                MOVWF       LSBDiasYear
                RETURN
;
; CheckSegundo - En los segundos acabados en 0 activa un flag, para que en la siguiente interrupción de PPS se
;gestione el reloj de los CADs.
;
CheckSegundo
                MOVF        Segundo, W
                MOVWF       Temporal
```

```
                MOVF    Temporal, F
                BTFSC   STATUS, Z
                GOTO    ActivaFlag
OtraVez
                MOVLW   .10
                SUBWF   Temporal, F
                BTFSC   STATUS, Z
                GOTO    ActivaFlag
                BTFSC   STATUS, C
                GOTO    OtraVez
                GOTO    FinCheckSegundo
ActivaFlag
                BSF     Flag, IntADCs
FinCheckSegundo
                RETURN
;
; Display - Saca hora y fecha por el display LCD.
;
Display
                MOVF    HoraLCD, W      ; Sólo actualiza las horas en el LCD si son distintas a
                SUBWF   Hora, W         ;las que hay; en ese caso también saca el día y año.
                BTFSC   STATUS, Z
                GOTO    ChkMinuto
                IntroPosVar PosHora, Hora
                CALL    WrLCD2
                IntroPos PosDia
                CALL    WrLCDDia
                IntroPosVar PosYear, Year
                CALL    WrLCD2
                MOVF    Hora, W
                MOVWF   HoraLCD         ; Actualiza la HoraLCD
ChkMinuto
                MOVF    MinutoLCD, W    ; Sólo actualiza los minutos en el LCD si son distintos a
                SUBWF   Minuto, W       ;los que hay
                BTFSC   STATUS, Z
                GOTO    SacaSegundo
                IntroPosVar PosMinuto, Minuto
                CALL    WrLCD2
                MOVF    Minuto, W
                MOVWF   MinutoLCD       ; Actualiza el MinutoLCD
SacaSegundo
                IntroPosVar PosSegundo, Segundo
                CALL    WrLCD2

                RETURN
;
; LeerRTC - Lee el RTC para generar el código lento.
;
LeerRTC
                BCF     STATUS, RP0 ; Banco 0
                MOVF    Minuto, W   ; Mete las variables de hora y fecha en variables temporales para
                MOVWF   DespMinuto  ;poder operar con ellas luego.
                MOVF    Hora, W
                MOVWF   DespHora
                MOVF    LSBDia, W
                MOVWF   DespLSBDia
```

```
                MOVF        MSBDia, W
                MOVWF       DespMSBDia
                MOVF        Year, W
                MOVWF       DespYear
                RETURN
;
; SeleccRet - Selecciona el retardo en el bit del código lento, según el número de segundo del reloj software. El formato
;del código es:
;
```

Seg.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Bit	0	0	m	m	m	m	m	m	h	h	h	h	h	d	d	d	d	d	d	d
Seg.	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
Bit	d	d	a	a	a	a	a	a	a	0	0	0	0	0	0	0	0	0	0	0
Seg.	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
Bit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

```
; siendo:
; m = cód. de minuto / h = cód. de hora / d = cód. de dia / a = cód. de año
;
SeleccRet
                BCF         STATUS, RP0         ; Banco 0

                ; Si 56 <= Segundo < 60, Retardo1:
                AsignaRet   D'56', D'60', 1

                ; Si 29 <= Segundo < 56, Retardo0:
                AsignaRet   D'29', D'56', 0

                ;  Si 2 <= Segundo < 29, Retardo segun código de tiempo real:
                BitTest     D'22', D'29', DespYear
                BitTest     D'21', D'22', DespMSBDia
                BitTest     D'13', D'21', DespLSBDia
                BitTest     D'08', D'13', DespHora
                BitTest     D'02', D'08', DespMinuto

                ; Si  0 <= Segundo < 2, Retardo0:
                AsignaRet   D'0', D'2', 0

                RETURN
;
; Espera - Función de espera para crear el código lento.
;
Espera
                MOVF        Retardo, F          ; Dependiendo del valor (0 o 1) de 'Retardo', mete en
                BTFSC       STATUS, Z           ;'ContTiempo' el valor 'Centesimas0' o 'Centesimas1'.
                GOTO        Retardo0
                MOVLW       Centesimas1
                GOTO        IniCont
Retardo0
                MOVLW       Centesimas0
IniCont
                MOVWF       ContTiempo          ; Inicializa el contador de tiempo
BucleEspera
                CALL        Centesima           ; Llama a 'Centesima' tantas veces como indique
                DECFSZ      ContTiempo, F       ;'ContTiempo'.
```

```
                GOTO        BucleEspera
                BCF         PORTC, CodLento     ; Después del retardo vuelve a poner la salida
                                                ;del código lento a 0
                RETURN
;
;-------------------------------------------------------------------------------------------------------------
; Funciones para la gestión del reloj de los conversores A/D
;-------------------------------------------------------------------------------------------------------------
;
; IntroFrecIni - Macro para introducir la frecuencia inicial en las variables 'Millones', 'CentsMillar', ..., 'Unidades'.
;
IntroFrecIni    MACRO       Cte1, Cte2, Cte3, Cte4, Cte5, Cte6, Cte7, Cte8
                MOVLW       Cte1
                MOVWF       Millones
                MOVLW       Cte2
                MOVWF       CentsMillar
                MOVLW       Cte3
                MOVWF       DecsMillar
                MOVLW       Cte4
                MOVWF       UnsMillar
                MOVLW       Cte5
                MOVWF       Centenas
                MOVLW       Cte6
                MOVWF       Decenas
                MOVLW       Cte7
                MOVWF       Unidades
                MOVLW       Cte8
                MOVWF       Decimas
                ENDM
;
; IntroCtes - Macro para introducir valores en las variables CteUM, CteC, CteD1, CteU y CteD2.
;
IntroCtes       MACRO       CteUnsMillar, CteCents, CteDecs, CteUns, CteDecimas
                MOVLW       CteUnsMillar
                MOVWF       CteUM
                MOVLW       CteCents
                MOVWF       CteC
                MOVLW       CteDecs
                MOVWF       CteD1
                MOVLW       CteUns
                MOVWF       CteU
                MOVLW       CteDecimas
                MOVWF       CteD2
                ENDM
;
; GestADCsCLK - Función principal para la gestión del  reloj de los conversores A/D.
;
GestADCsCLK
                BCF         STATUS, RP0         ; Banco 0
                BTFSS       Flag, IntADCs       ; Si el flag no se ha activado en 'IncRTC', no se hace
                GOTO        FinGestADCsCLK      ;nada.
                BCF         Flag, IntADCs       ; Se desactiva el flag
                BCF         STATUS, RP0         ; Banco 0
                MOVLW       .187                ; Si el Timer 1 no ha dado 187 vueltas, es porque el VXO
                SUBWF       RecContVueltas, W   ;está fuera de lazo
                BTFSS       STATUS, Z
```

```
                GOTO        Unlocked
                CALL        CorrigePWM
                IntroFrecIni .1, .2, .2, .5, .5, .2, .3, .2  ; Frecuencia correspondiente a 187 vueltas justas
                BTFSS       CCPR1H, 7
                GOTO        CompruebaBit14
                IntroCtes   .3, .2, .7, .6, .8               ; Se comprueba cada bit del contador y,si procede, se
                CALL        SumaFrecBCD                      ;suma el valor correspondiente.
CompruebaBit14
                BTFSS       CCPR1H, 6
                GOTO        CompruebaBit13
                IntroCtes   .1, .6, .3, .8, .4
                CALL        SumaFrecBCD
CompruebaBit13
                BTFSS       CCPR1H, 5
                GOTO        CompruebaBit12
                IntroCtes   .0, .8, .1, .9, .2
                CALL        SumaFrecBCD
CompruebaBit12
                BTFSS       CCPR1H, 4
                GOTO        CompruebaBit11
                IntroCtes   .0, .4, .0, .9, .6
                CALL        SumaFrecBCD
CompruebaBit11
                BTFSS       CCPR1H, 3
                GOTO        CompruebaBit10
                IntroCtes   .0, .2, .0, .4, .8
                CALL        SumaFrecBCD
CompruebaBit10
                BTFSS       CCPR1H, 2
                GOTO        CompruebaBit9
                IntroCtes   .0, .1, .0, .2, .4
                CALL        SumaFrecBCD
CompruebaBit9
                BTFSS       CCPR1H, 1
                GOTO        CompruebaBit8
                IntroCtes   .0, .0, .5, .1, .2
                CALL        SumaFrecBCD
CompruebaBit8
                BTFSS       CCPR1H, 0
                GOTO        CompruebaBit7
                IntroCtes   .0, .0, .2, .5, .6
                CALL        SumaFrecBCD
CompruebaBit7
                BTFSS       CCPR1L, 7
                GOTO        CompruebaBit6
                IntroCtes   .0, .0, .1, .2, .8
                CALL        SumaFrecBCD
CompruebaBit6
                BTFSS       CCPR1L, 6
                GOTO        CompruebaBit5
                IntroCtes   .0, .0, .0, .6, .4
                CALL        SumaFrecBCD
CompruebaBit5
                BTFSS       CCPR1L, 5
                GOTO        CompruebaBit4
                IntroCtes   .0, .0, .0, .3, .2
```

```
                CALL        SumaFrecBCD
CompruebaBit4
                BTFSS       CCPR1L, 4
                GOTO        CompruebaBit3
                IntroCtes   .0, .0, .0, .1, .6
                CALL        SumaFrecBCD
CompruebaBit3
                BTFSS       CCPR1L, 3
                GOTO        CompruebaBit2
                IntroCtes   .0, .0, .0, .0, .8
                CALL        SumaFrecBCD
CompruebaBit2
                BTFSS       CCPR1L, 2
                GOTO        CompruebaBit1
                IntroCtes   .0, .0, .0, .0, .4
                CALL        SumaFrecBCD
CompruebaBit1
                BTFSS       CCPR1L, 1
                GOTO        CompruebaBit0
                IntroCtes   .0, .0, .0, .0, .2
                CALL        SumaFrecBCD
CompruebaBit0
                BTFSS       CCPR1L, 0
                GOTO        BCD2ASC
                IntroCtes   .0, .0, .0, .0, .1
                CALL        SumaFrecBCD
BCD2ASC
                MOVLW       .48
                ADDWF       Decimas, F
                ADDWF       Unidades, F
                ADDWF       Decenas, F
                ADDWF       Centenas, F
                ADDWF       UnsMillar, F
                ADDWF       DecsMillar, F
                ADDWF       CentsMillar, F
                ADDWF       Millones, F

                WrLCDText   PosTexto2, MensInicioB
                MOVLW       PosFrec
                MOVWF       Pos
                CALL        WrLCDFrec
                GOTO        FinGestADCsCLK
Unlocked
                ;CLRF       Decimas
                ;CLRF       Unidades
                ;CLRF       Decenas
                ;CLRF       Centenas
                ;CLRF       UnsMillar
                ;CLRF       DecsMillar
                WrLCDText   PosTexto2, MensUnlocked

FinGestADCsCLK
                RETURN
```

```
; CorrigePWM - Compara la cuenta capturada del Timer 1 con la correcta, y corrige el duty-cycle de la señal PWM en
;consecuencia.
;
CorrigePWM
                BCF             STATUS, RP0             ; Banco 0
                MOVLW           .128                    ; Resta el MSB del timer del valor correcto (128); si es
                SUBWF           CCPR1H, W               ;cero, pasa a comprobar el LSB; si no, mira si el nuevo
                BTFSC           STATUS, Z               ;valor es mayor o menor que el correcto y corrige el
                GOTO            VerByteBajo             ;duty-cycle de la señal PWM en consecuencia.
                BTFSS           STATUS, C
                GOTO            SubirPWM
                GOTO            BajarPWM
VerByteBajo
                CLRF            W
                SUBWF           CCPR1L, W
                BTFSC           STATUS, Z
                GOTO            FinCorrigePWM
                BTFSS           STATUS, C
                GOTO            SubirPWM
                GOTO            BajarPWM
SubirPWM
                INCF            CCPR2L, F
                GOTO            FinCorrigePWM
BajarPWM
                DECF            CCPR2L, F
FinCorrigePWM
                RETURN
;
; SumaFrecBCD - Suma los valores de las variables CteUM, CteC, CteD1, CteU y CteD2 a las variables UnsMillar,
;Centenas, Decenas, Unidades y Decimas, respectivamente. No es necesario considerar más allá de las unidades de
;millar, porque sólo van a cambiar las cinco cifras menos significativas de la frecuencia. Para la introducción de los
;valores en las variables se usa la macro 'IntroCtes'.
;
SumaFrecBCD
                BCF             STATUS, RP0             ; Banco 0
                MOVF            CteD2, W
                ADDWF           Decimas, F
                MOVLW           .10
                SUBWF           Decimas, W
                BTFSS           STATUS, C
                GOTO            SumaUns
                MOVWF           Decimas
                INCF            Unidades, F
SumaUns
                MOVF            CteU, W
                ADDWF           Unidades, F
                MOVLW           .10
                SUBWF           Unidades, W
                BTFSS           STATUS, C
                GOTO            SumaDecs
                MOVWF           Unidades
                INCF            Decenas, F
SumaDecs
                MOVF            CteD1, W
                ADDWF           Decenas, F
                MOVLW           .10
```

```
                SUBWF       Decenas, W
                BTFSS       STATUS, C
                GOTO        SumaCents
                MOVWF       Decenas
                INCF        Centenas, F

SumaCents
                MOVF        CteC, W
                ADDWF       Centenas, F
                MOVLW       .10
                SUBWF       Centenas, W
                BTFSS       STATUS, C
                GOTO        SumaUnsMillar
                MOVWF       Centenas
                INCF        UnsMillar, F

SumaUnsMillar
                MOVF        CteUM, W
                ADDWF       UnsMillar, F
                MOVLW       .10
                SUBWF       UnsMillar, W
                BTFSS       STATUS, C
                GOTO        FinSumaFrecBCD
                MOVWF       UnsMillar
                INCF        DecsMillar, F

FinSumaFrecBCD
                RETURN


;
;-------------------------------------------------------------------------------------------------------------------
; Funciones para la monitorización de las interrupciones en el PIC
;-------------------------------------------------------------------------------------------------------------------
;
; CambiaPin - Cambia de estado, con cada interrupción de PPS, el bit 4 del puerto C (pin 23 del PIC16C74A). Esto
;permite comprobar si la interrupción se está sirviendo correctamente.
;
CambiaPin
                BTFSC       Flag, Testigo
                GOTO        SacarUno
                GOTO        SacarCero
SacarUno
                BCF         Flag, Testigo       ; Saca uno por el pin del PuertoC y pone la variable de
                BSF         PORTC, VerPPS       ;flag a 0, para la próxima pasada.
                RETURN
SacarCero
                BSF         Flag, Testigo       ; Saca cero por el pin del PuertoC y pone la variable de
                BCF         PORTC, VerPPS       ;flag a 1, para la próxima pasada
                RETURN
```

```
;*****************************************************************************
; Tablas
;*****************************************************************************
;-----------------------------------------------------------------------------------------------------------
; Tablas para sacar mensajes por el LCD
;-----------------------------------------------------------------------------------------------------------
;
                ORG             0x600

MensPres1A                                              ; Mensajes de presentación:
                ADDWF           PCL, F
                DT              "Gran Sasso      ", 0   ; Las tablas deben acabar siempre con 0

MensPres1B
                ADDWF           PCL, F
                DT              "seismic array   ", 0

MensPres2A
                ADDWF           PCL, F
                DT              "IAG-Univ.Granada", 0

MensPres2B
                ADDWF           PCL, F
                DT              "& Univ. L'Aquila", 0

MensInicioA
                ADDWF           PCL, F
                DT              "00:00:00  001/99", 0

MensInicioB
                ADDWF           PCL, F
                DT              "VXO:       0.0Hz", 0

MensFlanco                                              ; Mensaje de elección del flanco de los PPS
                ADDWF           PCL, F
                DT              "Choose PPS slope", 0

MensInstrucc                                            ; Instrucciones para la selección de flanco
                ADDWF           PCL, F
                DT              "Select with +   ", 0

MensFlancoAsc                                            Mensaje de flanco ascendente
                ADDWF           PCL, F
                DT              "PPS rising slope", 0

MensFlancoDesc                                          ; Mensaje de flanco descendente
                ADDWF           PCL, F
                DT              "PPS desc. slope ", 0

MensSincro                                              ; Mensaje de función de sincronización
                ADDWF           PCL, F
                DT              "Synchro Function", 0
```

```
MensSincA                                       ; Mensaje de sincronización del año
                ADDWF       PCL, F
                DT          "Enter year:    ", 0

MensSincD                                       ; Idem, día
                ADDWF       PCL, F
                DT          "Enter day:     ", 0

MensSincH                                       ; Idem, hora
                ADDWF       PCL, F
                DT          "Enter hour:    ", 0

MensSincM                                       ; Idem, minuto
                ADDWF       PCL, F
                DT          "Enter minute:  ", 0

MensSincS                                       ; Idem, segundo
                ADDWF       PCL, F
                DT          "Enter second:  ", 0

FilaEnBlanco                                    ; Fila en blanco
                ADDWF       PCL, F
                DT          "               ", 0

MensUnlocked                                    ; Mensaje de fuera de lazo
                ADDWF       PCL, F
                DT          "VXO:   Unlocked", 0
;
;
;*****************************************************************************
; Código de programa
;*****************************************************************************
;-----------------------------------------------------------------------------------------------------------
; Rutina de servicio de la interrupción
;-----------------------------------------------------------------------------------------------------------
;
                ORG         ISR_V               ; Vector de interrupcion

                BCF         STATUS, RP0         ; Banco 0
                BTFSC       PIR1, CCP1IF        ; Comprueba si es la int. de CCP1
                CALL        GestIntPPS
                BCF         STATUS, RP0         ; Banco 0
                BTFSC       PIR1, TMR1IF        ; Comprueba si es la int. de Timer1
                CALL        GestIntTmr1
                BCF         INTCON, RBIF        ; Limpia los flags de todas las int. salvo las de TMR1
                BCF         INTCON, INTF        ;y CCP1, que se limpian en las funciones de gestión
                BCF         INTCON, T0IF        ;correspondientes (‘GestIntPPS’ y ‘GestIntTmr1’).
                MOVLW       0x05
                ANDWF       PIR1, F
                CLRF        PIR2
                RETFIE
```

```
; GestIntPPS.- Función de gestión de la interrupción de CCP1, que es disparada por la señal de referencia de un pulso
;por segundo. En esta función sólo se realizan las tareas críticas en el tiempo, como son la puesta en alto de la salida de
;tiempo real codificada (código lento) y la inicialización, si procede, del Timer 1. El resto de las tareas se realiza en el
;bucle principal del programa.
;
GestIntPPS                                          ; Banco 0 habilitado antes
                BSF             PORTC, CodLento     ; Pone a 1 la salida del Código Lento
                BCF             PIR1, CCP1IF        ; Limpia el flag de interrupción
                BTFSS           Flag, IntADCs       ; Comprueba si toca gestión del Tmr1
                GOTO            FinGestIntPPS
                BCF             T1CON, TMR1ON       ; Deshabilita el Timer1
                CLRF            TMR1H               ; Inicializa el timer1 a 4, que es el número de pulsos que
                MOVLW           .4                  ;se pierden mientras la interrupción está deshabilitada.
                MOVWF           TMR1L
                BTFSC           PIR1, TMR1IF        ; Antes de leer el contador de vueltas hay que ver si es
                INCF            ContVueltas, F      ;necesario incrementarlo de nuevo.
                BCF             PIR1, TMR1IF
                MOVF            ContVueltas, W      ; Hay que poner a 0 el contador de vueltas, por lo que
                MOVWF           RecContVueltas      ;lo que antes se pasa el valor a otra variable temporal para
                CLRF            ContVueltas         ;poder operar con ella.
                BSF             T1CON, TMR1ON       ; Habilita el Timer1
FinGestIntPPS
                BSF             Flag, IntPPS        ; Marca para que entre en el resto de tareas en el bucle
                                                    ;principal.
                RETURN
;
; GestIntTmr1.- Función de gestión de la interrupción del Timer 1, que se dispara cada vez que hay desbordamiento
;del timer 1. La función simplemente incrementa un contador de vueltas.
;
GestIntTmr1                                         ; Banco 0 habilitado antes.
                BCF             PIR1, TMR1IF        ; Limpia el flag de interrupción
                INCF            ContVueltas, F      ; Incrementa contador de vueltas
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; Programa principal
;-------------------------------------------------------------------------------------------------------------------
;
                ORG             RESET_V
                GOTO            Inicio
;
                ORG             0x750
Inicio
;
; Inicializaciones para el módulo de reloj de los CADs ----------------------------------------------------
;
                                        ; TIMER1 EN MODO CONTADOR SÍNCRONO
                BCF             STATUS, RP0         ; Banco 0
                CLRF            PORTC               ; Inicializa PORTC limpiando la salida
                MOVLW           0x03                ; Configuración del Timer1:
                MOVWF           T1CON               ; (T1CON<0>=1): Timer1 habilitado
                                                    ; (T1CON<1>=1): Modo contador
                                                    ; (T1CON<2>=0): Modo síncrono
                                                    ; (T1CON<3>=0): Oscilador deshabilitado
                                                    ;(entrada por pin T1CKI)
                                                    ; (T1CON<5:4>=01): Prescaler = 1:1
```

```
                BSF             STATUS, RP0             ; Banco 1
                BSF             TRISC, T1CKI            ; Pin RC0/T1CKI como entrada
                BCF             PIE1, TMR1IE            ; Deshabilita la int. del Timer1
                BCF             STATUS, RP0             ; Banco 0
                CLRF            TMR1H                   ; Inicializa el Timer1 a 0
                CLRF            TMR1L
                CLRF            ContVueltas

                                ; MÓDULO CCP1 EN MODO CAPTURA: Hecho luego, al
                                ;inicializar las interrupciones

                                ; MÓDULO CCP2 EN MODO PWM:
                BSF             STATUS, RP0             ; Banco 1
                MOVLW           D'254'                  ; Fija el periodo de PWM a 9.766kHz
                MOVWF           PR2
                BCF             STATUS, RP0             ; Banco 0
                MOVLW           0x80                    ; Inicializa el duty-cycle del PWM al 50%
                MOVWF           CCPR2L                  ;(CCPRL + CCP2CON<5:4>), y el módulo CCP2 a
modo
                MOVLW           0x0C                    ;PWM (haciendo CCP2CON<3:0> = 11xx).
                MOVWF           CCP2CON
                BCF             T2CON, T2CKPS1          ; Prescaler del Timer2 a 1:1
                BCF             T2CON, T2CKPS0
                BSF             T2CON, TMR2ON           ; Habilita el Timer2
                BSF             STATUS, RP0             ; Banco 1
                BCF             TRISC, CCP2             ; Pin RC1/CCP2 como salida

;
; Inicializaciones para la monitorización de los PPS ----------------------------------------------------------
;
                BSF             STATUS, RP0             ; Banco 1
                BCF             TRISC, VerPPS           ; Pin 4 de PORTC como salida (indicador de
                                                        ;interrupciones)
                BCF             STATUS, RP0             ; Banco 0
                BCF             Flag, Testigo           ; Inicializa el flag de Int. de PPS
;
; Inicializaciones para el display LCD -------------------------------------------------------------------------
;
                CLRF            HoraLCD
                CLRF            MinutoLCD
                CALL            IniciaLCD
;
; Inicializaciones para la sincronización del RTC --------------------------------------------------------------
;
                BCF             STATUS, RP0             ; Banco 0
                CLRF            SINC_CNTL               ; Limpia los latches
                BSF             STATUS, RP0             ; Banco 1
                BSF             SINC_CFG, BotSinc       ; Pin de sincronización → entrada
                BSF             SINC_CFG, BotMas        ; Pin conectado al Botón+ → entrada
                BSF             SINC_CFG, BotMenos      ; "     "     "       Botón- →    "
;
; Inicializaciones para el RTC y el código lento ---------------------------------------------------------------
;
                BCF             STATUS, RP0             ; Banco 0
                CLRF            Segundo                 ; Inicializa RTC a las 00:00:00 del día 1 del año 2000
                CLRF            Minuto
```

```
                CLRF        Hora
                CLRF        MSBDia
                MOVLW       .1
                MOVWF       LSBDia
                CLRF        Year
                CLRF        Retardo

                BCF         STATUS, RP0         ; Banco 0
                CLRF        PIR1                ; Pone a 0 todos los flags de interrupción
                CLRF        PIR2

                BSF         STATUS, RP0         ; Banco 1
                BSF         TRISC, CCP1         ; Pin RC2/CCP1 como entrada
                BCF         Flag, IntPPS        ; Pone a 0 el flag de int.

                BCF         STATUS, RP0         ; Banco 0
                MOVLW       0x04                ; Interrupcion con cada flanco DESCENDENTE del pin
                MOVWF       CCP1CON             ;RC2/CCP1
                BSF         INTCON, PEIE        ; Habilita interrupciones periféricos

                BSF         STATUS, RP0         ; Banco 1
                BSF         PIE1, CCP1IE        ; Habilita la interrupcion de CCP1
                BCF         TRISC, CodLento     ; Pin 3 de PORTC como salida (código lento)

                BSF         PIE1, TMR1IE        ; Habilita la interrupción de Timer1
                BCF         STATUS, RP0         ; Banco 0 (lo dejo habilitado para el bucle ppal.)
                BCF         INTCON, GIE         ; Deshabilita las interrupciones
                CALL        Presentacion        ; Saca los mensajes iniciales por el display
                BSF         T1CON, TMR1ON       ; Habilita el timer1
                BSF         INTCON, GIE         ; Habilita las interrupciones
;
; Bucle principal ------------------------------------------------------------------------------------------
;
bucle
                CALL        Configuracion       ; Hace polling al botón de sincro. y al de elección de
                                                ;flanco.
                BTFSS       Flag, IntPPS        ; Si se ha producido una int de PPS realiza las tareas de
                GOTO        bucle               ; monitorización, gestión del reloj de los CADs y código
                BCF         Flag, IntPPS        ; lento. Las tareas críticas en el tiempo se han realizado
                CALL        CambiaPin           ;en la rutina de servicio de la interrupción.
                CALL        GestADCsCLK
                CALL        GestCodLento
                GOTO        bucle
                END
```

Anexo I.B:

Ficheros de circuito impreso de la antena del Gran Sasso

ANEXO I.B: FICHEROS DE CIRCUITO IMPRESO DE LA ANTENA DEL GRAN SASSO

a)

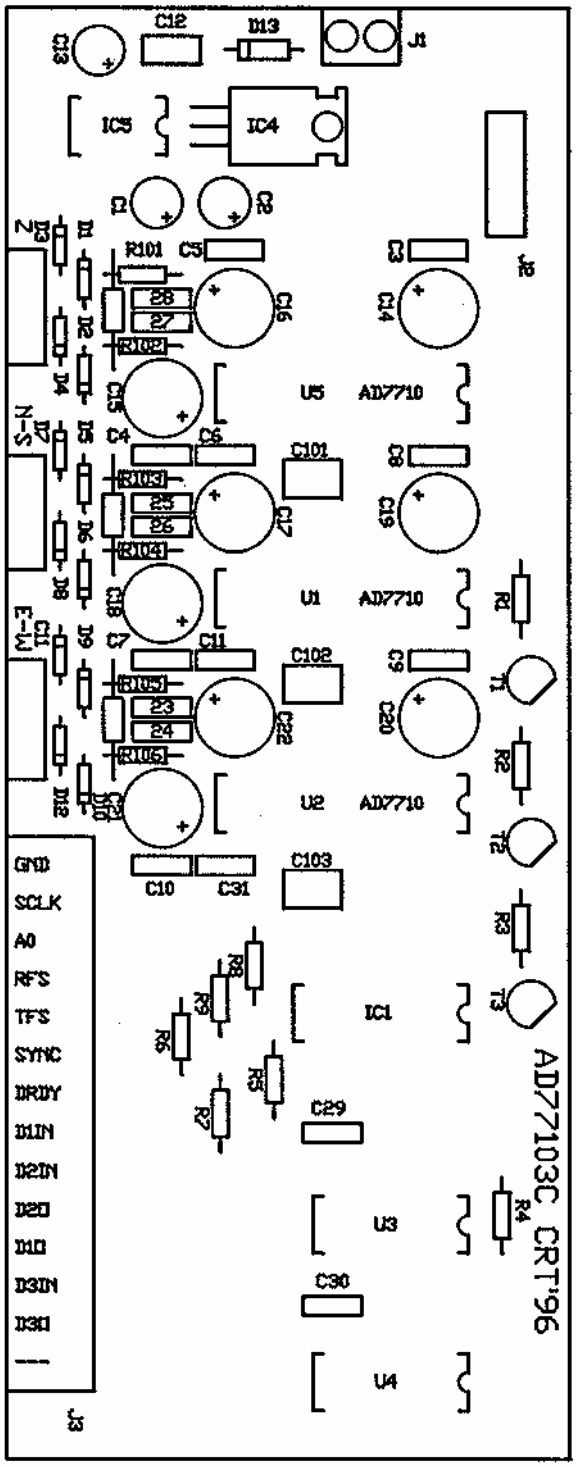
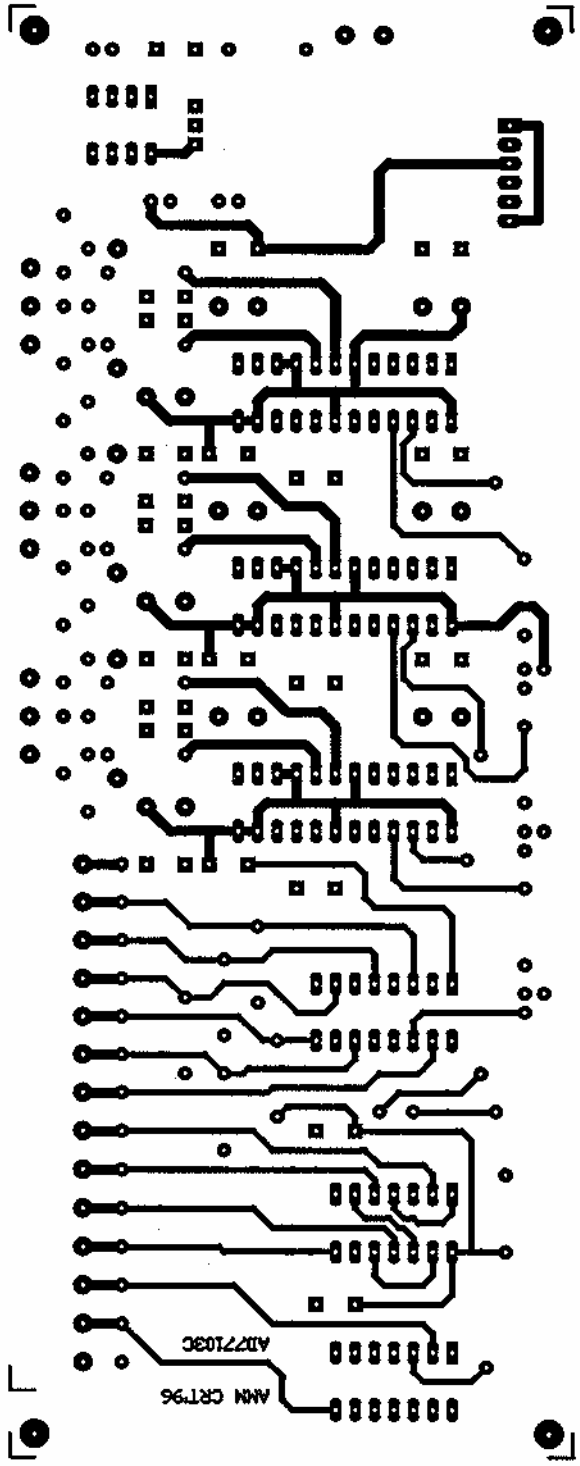
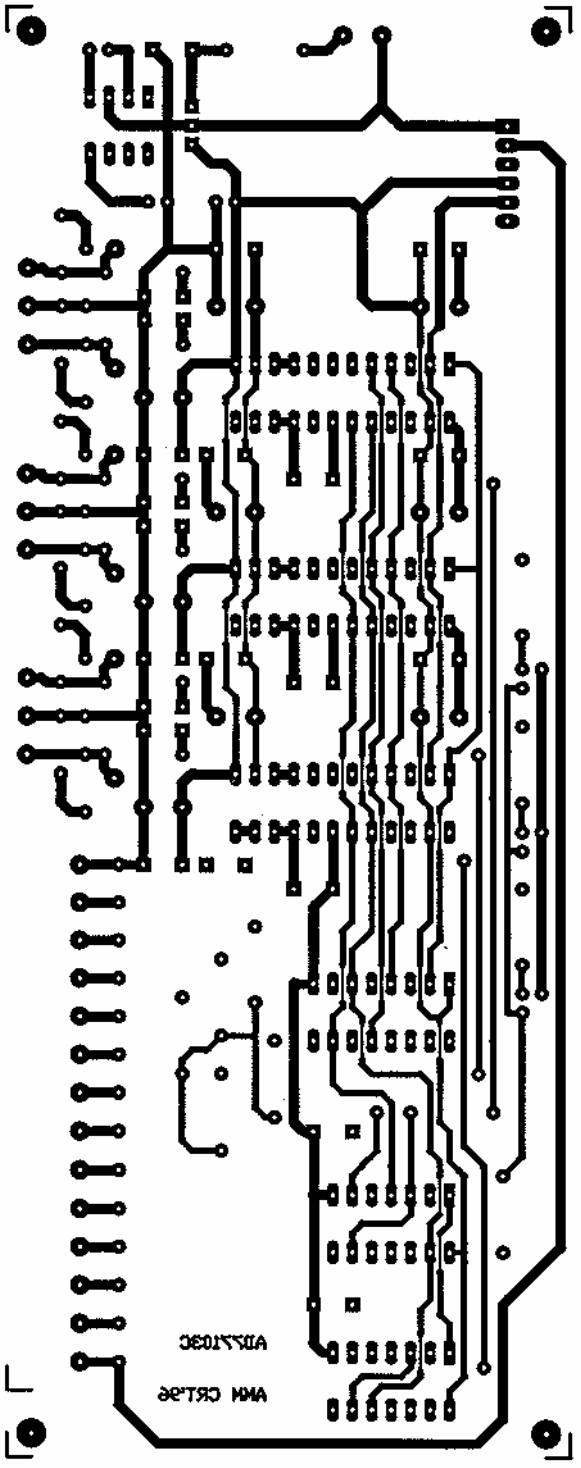


Figura I.B.1. *Tarjeta de conversión analógico/digital de la antena del Gran Sasso, correspondiente al esquema eléctrico de la figura 3.11 de la memoria: distribución de componentes (a), cara superior (b) y cara inferior (c) (diseño de la tarjeta: Antonio Martos Moreno).*

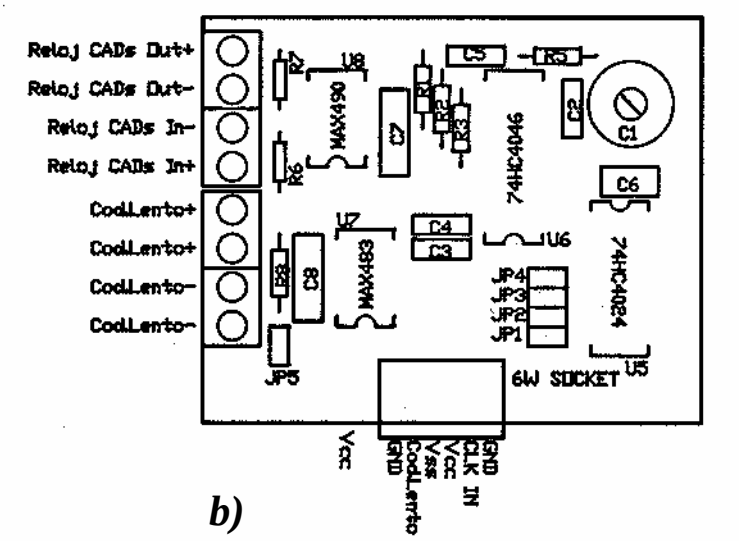
I.B.1.b)



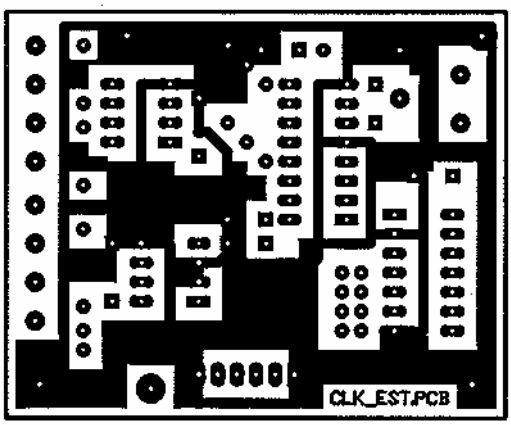
I.B.1.c)



a)



b)



c)

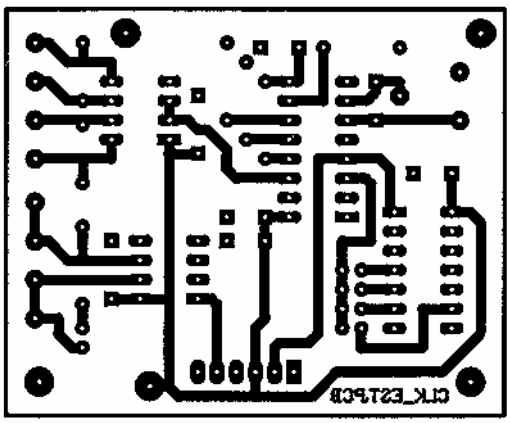


Figura I.B.2. *Tarjeta de generación de las señales de reloj en las estaciones (esquema eléctrico en la figura 3.12 de la memoria de tesis): distribución de componentes (a), cara superior –plano de tierra- (b) y cara inferior (c).*

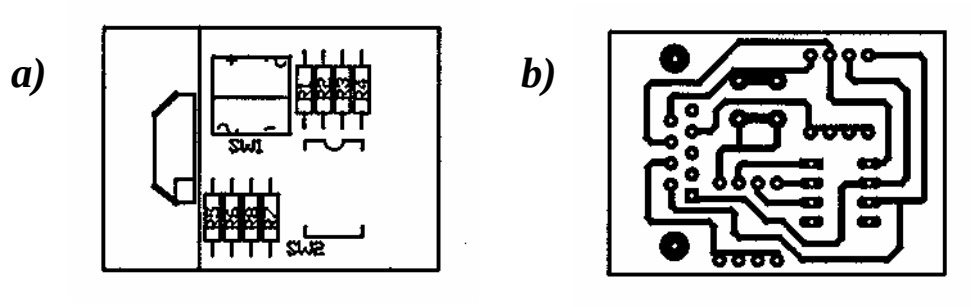


Figura I.B.3. *Placa de generación del código identificativo de las estaciones y PCs nodales (esquema en figura 3.13 de la memoria): distribución de componentes (a) y capa inferior (b).*

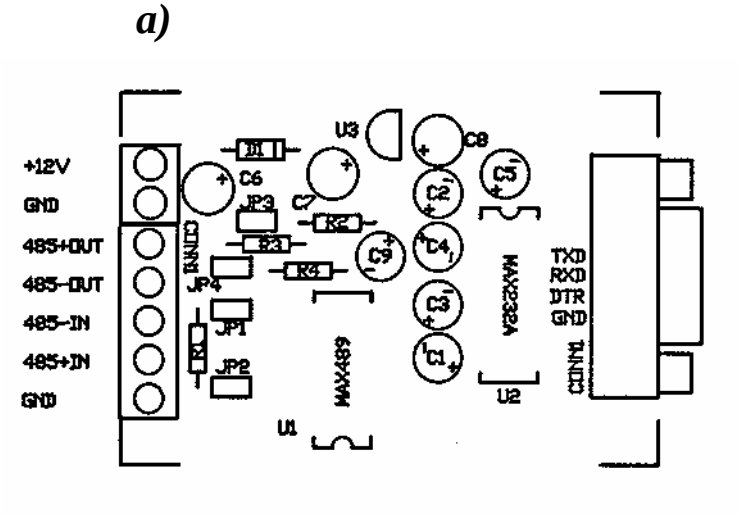
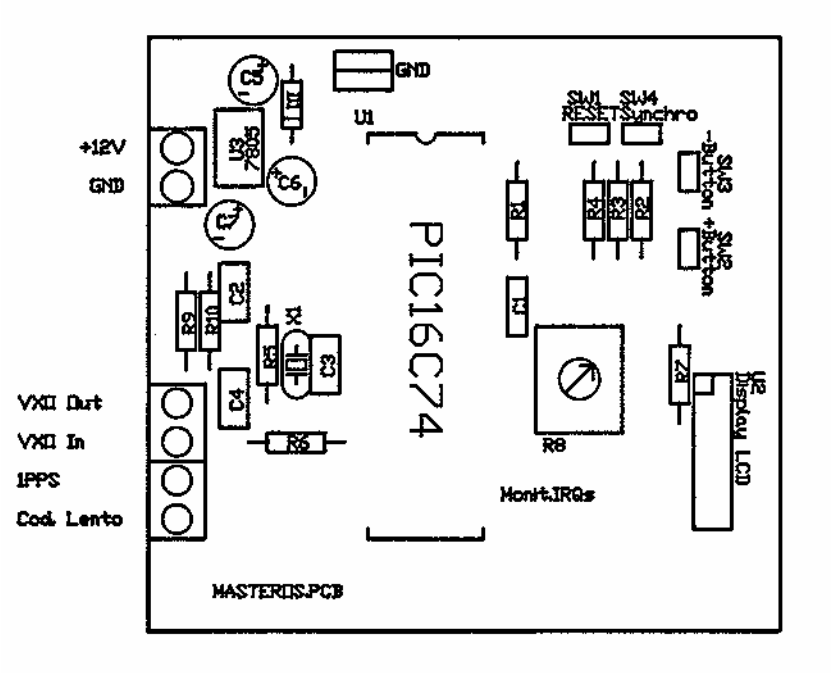


Figura I.B.4. *Tarjeta de conversión de niveles RS-232 / RS-485 (esquema en la figura 3.14 de la memoria): distribución de componentes (a) y cara inferior (b).*

a)



b)

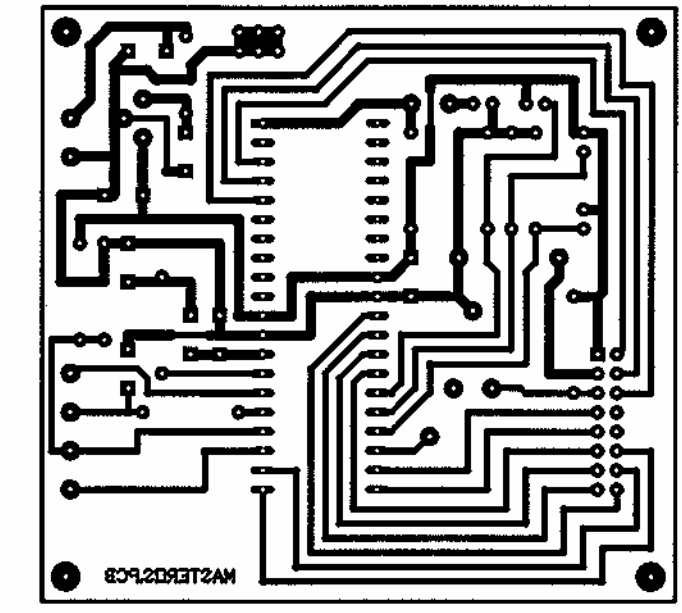
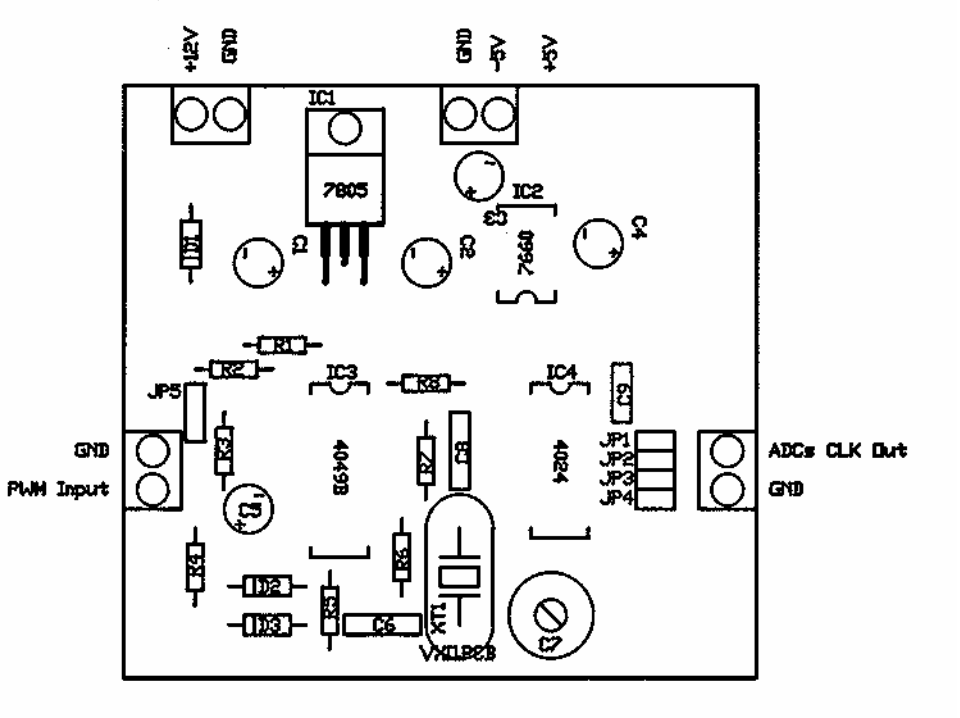


Figura I.B.5. *Placa del microcontrolador PIC16C74 del oscilador patrón (esquema en figura 3.19 de la memoria): distribución de componentes (a) y cara inferior (b).*

a)



b)

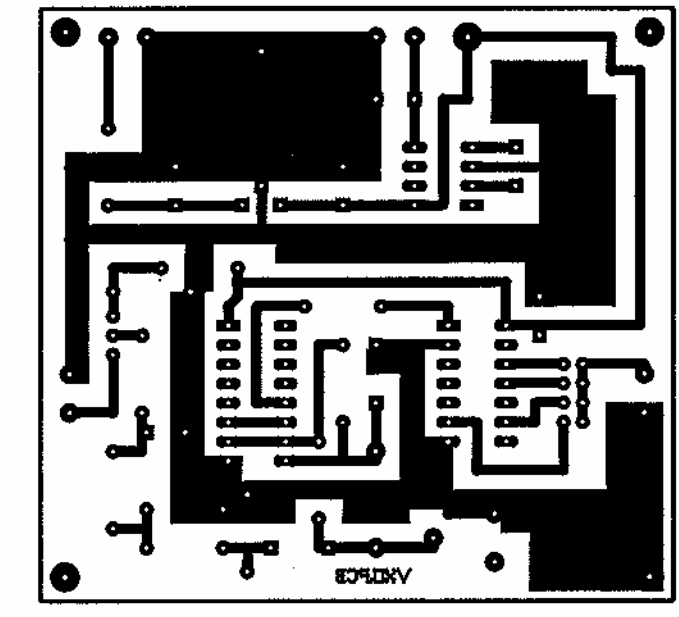


Figura I.B.6. *Tarjeta del VCXO del oscilador patrón (esquema en figura 3.20 de la memoria): distribución de componentes (a) y capa inferior (b).*

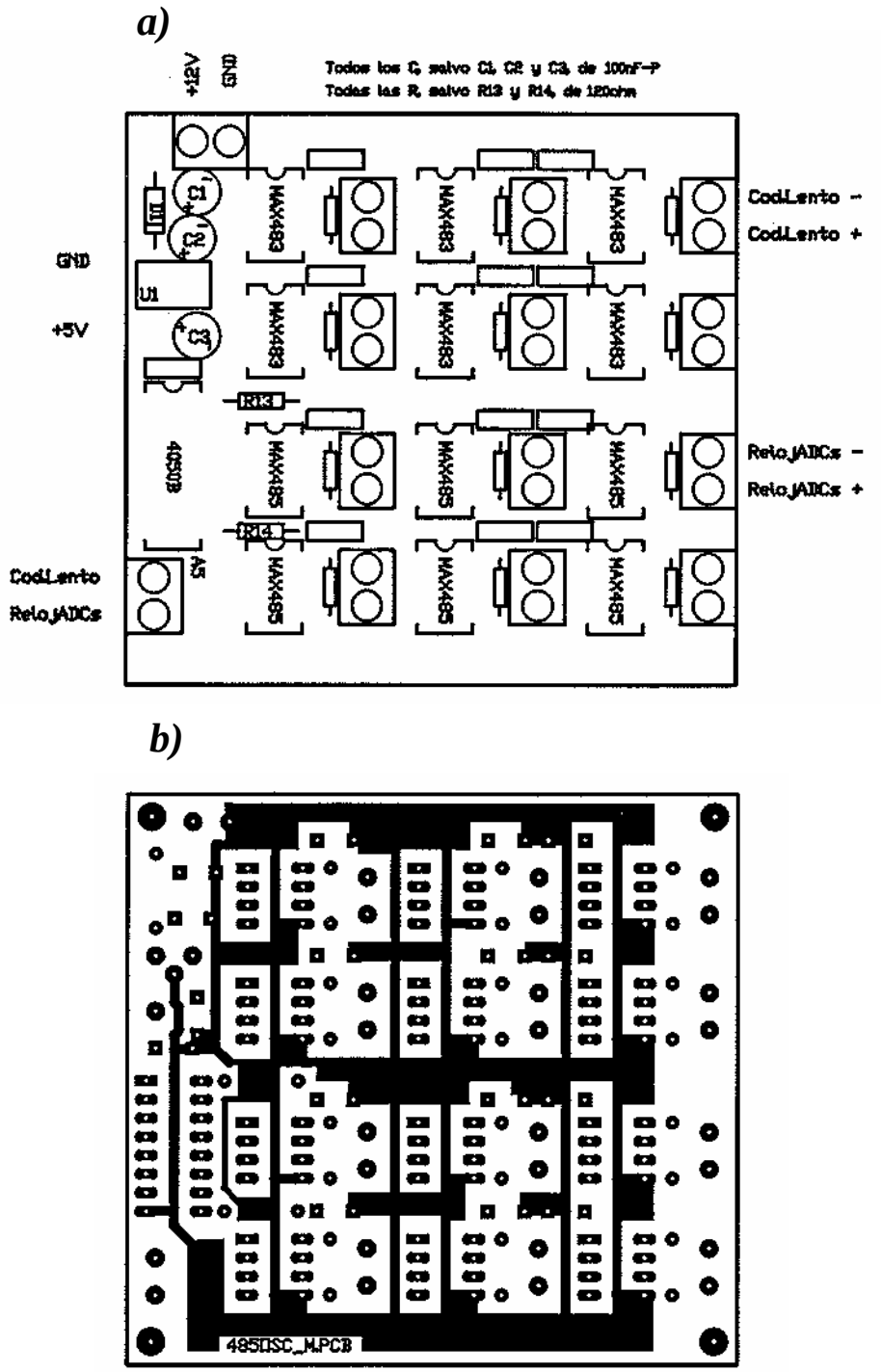
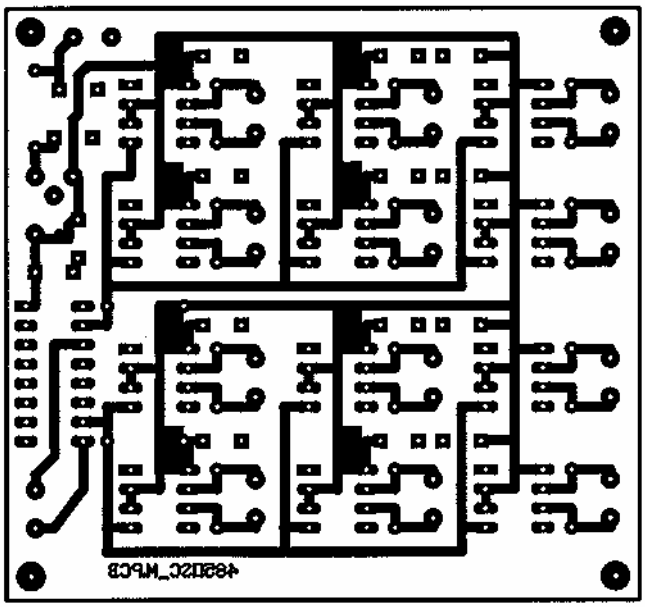


Figura I.B.7. *Placa de interfaz RS-485 del oscilador patrón (esquema en figura 3.21 de la memoria): distribución de componentes (a), cara superior –plano de tierra- (b) y cara inferior (c, página siguiente).*

I.B.7.c)



ANEXO II

ANTENA DEL VESUBIO

Anexo II.A:

Programas de la antena del Vesubio

ANEXO II.A.1.- PROGRAMAS Y FICHEROS DE CONFIGURACIÓN DEL PC DE LA ANTENA DEL VESUBIO

II.A.1.a. FICHERO DE CONFIGURACIÓN PRINCIPAL

Fichero: VES2.CFG

/* Fichero de configuración principal de la antena del Vesubio, al que accede el programa de inicialización P_INIC2 (anexo II.A.1.b) para conocer los valores de los parámetros de operación. */

```
/*************************************************************************
  Fichero
*************************************************************************/
; VES2.CFG: Fichero de configuración del array del Vesubio. Es usado por el programa P_INIC2.C. Puede
;insertarse cualquier número de líneas de comentario, siempre que el primer carácter sea ';'.
;
; FRECUENCIA DE MUESTREO. Los valores posibles son 50, 100 o 200 mps.
;
Frecuencia: 100
;
; GANANCIA. Los valores posibles son 1, 2, 4, 8, 16, 32, 64 o 128.
;
Ganancia: 1
;
; CONFIGURACIÓN FÍSICA DEL ARRAY: cada bit representa un módulo A/D. Si está a 0 significa que
;dicho módulo no se encuentra operativo. Si vale 1, sí lo está.
; El orden en el que los módulos están representados es el siguiente:
; L0M0/L1M0/L2M0/L3M0/L0M1/... /L3M3
; siendo L = Número de línea serie
;       M = Número de módulo A/D dentro de la línea serie
; Ej.: Si se usan los primeros dos módulos en las cuatro líneas serie:
; ConfLinSer: 1111111100000000
;          0123012301230123
;
ConfLinSer: 1111011101110000
;
; NÚMERO DE TARJETAS DE MEMORIA USADAS: Los valores posibles son 0, 1, 2 o 4.
;El valor 0 fija el tamaño de los bancos de memoria en 10 segundos, independientemente del resto de
;parámetros de operación.
;
NumTarjsMem: 2
;
; Fin fichero VES2.CFG
```

II.A.1.b. PROGRAMA DE INICIALIZACIÓN DEL SISTEMA

Fichero: P_INIC2.C

/* Programa de inicialización de la antena del Vesubio. El valor de los parámetros de operación se lee del fichero de configuación VES2.CFG (anexo II.A.1.a). El programa principal se encuentra al final. */

```
/*************************************************************************
  Macros y ficheros de cabecera
*************************************************************************/
#include <dos.h>
#include <io.h>
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <ctype.h>
#include <string.h>

#define  FALSE          0
#define  TRUE           !FALSE
#define BusDatos        0x378     // Primer puerto paralelo
#define Control1        BusDatos+1
#define Control2        BusDatos+2
#define  BITS_INIC      30        // Número de bits de la secuencia de inicialización

/*************************************************************************
  Declaración de variables
*************************************************************************/
FILE *FichCfg;
unsigned char Cadena[40], ConfLinSer[20];
unsigned int Frec, Ganancia, NumTarjsMem;
int i;
unsigned long PalabraCtrol;

/*************************************************************************
  Declaración de funciones
*************************************************************************/
void InConfig_0(void);
void InConfig_1(void);
void SysReset_0(void);
void SysReset_1(void);
void Inicializacion(void);
void LeeConfig(void);
int PulsOutConfig(void);
void MandaConfig(void);
```

```
/*************************************************************************
  Definición de las funciones
*************************************************************************/

/* Pone la señal 'IN Configuration' a 0 (el nombre de las señales de configuración es el del módulo SerPar,
por lo que 'IN Configuration' es una salida para el PC y 'OUT Configuration' una entrada) */
void InConfig_0(void)
{
	outportb(Control2, inportb(Control2) & 0xFB);
}

// Pone la señal 'IN Configuration' a 1
void InConfig_1(void)
{
	outportb(Control2, inportb(Control2) | 0x04);
}

// Pone la señal 'SysReset' a 0 (está invertida, luego hay que sacar un 1)
void SysReset_0(void)
{
	outportb(Control2, inportb(Control2) | 0x01);
}

// Pone la señal 'SysReset' a 1 (está invertida, luego hay que sacar un 0)
void SysReset_1(void)
{
	outportb(Control2, inportb(Control2) & 0xFE);
}

void Inicializacion(void)
{
	if (( FichCfg = fopen("VES2.CFG", "r")) == NULL)
	{
		printf("\n No se puede abrir el fichero de configuración VES2.CFG para lectura");
		exit(0);
	}

	SysReset_1();	// Desactiva la línea de reset.
	InConfig_1();	// La línea /INIT del puerto paralelo está a 1 por defecto, pero por si acaso la subo
}

void LeeConfig(void)
{
	unsigned char CambioConfLinSer;

	// Lee parámetros del fichero VES2.CFG:
	do
	{
		fscanf(FichCfg, "%s", &Cadena);
		// Si es un comentario salta de línea:
		if (Cadena[0] == ';')		fscanf(FichCfg, "\n");
		else
		{
			if (!feof(FichCfg))
			{
```

```
                                // Si no es un comentario, lee el par metro correspondiente:
                                if (!strcmp(Cadena,"Ganancia:")) fscanf(FichCfg, "%u", &Ganancia);
                                if (!strcmp(Cadena,"Frecuencia:")) fscanf(FichCfg, "%u", &Frec);
                                if (!strcmp(Cadena,"ConfLinSer:")) fscanf(FichCfg, "%s", &ConfLinSer);
                                if (!strcmp(Cadena,"NumTarjsMem:")) fscanf(FichCfg, "%u",
                                    &NumTarjsMem);
                        }
                }
        }
while (!feof(FichCfg));

// Comprueba que no hay errores en los valores introducidos:
if ((Frec != 50) && (Frec != 100) && (Frec != 200))
{
        printf("\n\n Error en la frecuencia de muestreo introducida en el fichero VES2.CFG.");
        printf("\n No se ha enviado secuencia de configuración al módulo SerPar.\n");
        fclose(FichCfg);
        exit(0);
}
if ((Ganancia != 1) && (Ganancia != 2) && (Ganancia != 4) && (Ganancia != 8) &&
(Ganancia != 16) && (Ganancia != 32) && (Ganancia != 64) && (Ganancia != 128))
{
        printf("\n\n Error en la ganancia introducida en el fichero de configuración VES2.CFG.");
        printf("\n No se ha enviado secuencia de configuración al módulo SerPar.\n");
        fclose(FichCfg);
        exit(0);
}
for (i=0; i<16; i++)
{
        if ((ConfLinSer[i] != '0') && (ConfLinSer[i] != '1'))
        {
                printf("\n\n Error en la configuración de las líneas serie introducida en
                VES2.CFG.");
                printf("\n No se ha enviado secuencia de configuración al módulo SerPar.\n");
                fclose(FichCfg);
                exit(0);
        }
}
if ((NumTarjsMem != 0) && (NumTarjsMem != 1) && (NumTarjsMem != 2) && (NumTarjsMem
!= 4))
{
        printf("\n\n Error en el número de tarjetas de memoria introducido en el fichero
        VES2.CFG.");
        printf("\n No se ha enviado secuencia de configuración al módulo SerPar.\n");
        fclose(FichCfg);
        exit(0);
}
printf("\n\n Frecuencia leída: %u", Frec);
printf("\n Ganancia leída: %u", Ganancia);
printf("\n Máscara de módulos A/D operativos leída: %s", ConfLinSer);
printf("\n Número de tarjetas de memoria leído: %u", NumTarjsMem);
if(Frec == 200)
{
        CambioConfLinSer = FALSE;
        for (i=8; i<16; i++)
        {
```

```
                        if (ConfLinSer[i] != '0')
                        {
                                ConfLinSer[i] = '0';
                                CambioConfLinSer = TRUE;
                        }
                }
                if (CambioConfLinSer)
                {
                        printf("\n\n Para frecuencia de muestreo = 200mps sólo se registran los datos de
                        los módulos 0 y 1 de cada línea serie.");
                        printf("\n La m scara de módulos A/D operativos se ha modificado en
                        consecuencia:");
                        printf("\n Nueva m scara de módulos A/D operativos: %s", ConfLinSer);
                }
        }

/* Forma la palabra de control ordenando los bytes tal y como se enviarán. El formato es:
FFFG GGGG GGGL LLLL LLLL LLLL LLLM MM00, siendo:
F=Bits de frecuencia de muestreo: 001=50mps, 010=100mps, 100=200mps
G=Bits de ganancia: 0x01=1, 0x02=2, 0x04=4,... 0x80=128
L=Bits de configuración de las líneas serie, con el formato que se explica en los comentarios del fichero
VES2.CFG.
M=Número de tarjetas de memoria usadas: 001=1, 010=2, 100=4
Ej.: Frec = 100mps, Ganancia = 8, operativos los módulos 1 de la línea serie 0 y 2 de la línea serie 3, una
tarjeta de memoria:
Palabra de control: 0100 0001 0000 0001 0000 0100 0000 0100 */

        switch(Frec)
        {
                case(50):
                        PalabraCtrol = 0x01;
                        break;
                case(100):
                        PalabraCtrol = 0x02;
                        break;
                case(200):
                        PalabraCtrol = 0x04;
        }

        PalabraCtrol = ((PalabraCtrol<<8)|Ganancia);
        for (i=0; i<=15; i++)
        {
                PalabraCtrol <<= 1;
                if(ConfLinSer[i] == '1') PalabraCtrol |= 0x01;
        }
        PalabraCtrol = ((PalabraCtrol<<3)|NumTarjsMem);
        PalabraCtrol <<= 2;// Al final desplaza la palabra dos posiciones para que el MSb de la variable ya
                           //sea de información.
        printf("\n Palabra de control: 0x%08lX \n", PalabraCtrol);
}

/* Detecta los pulsos de configuración enviados por SerPar a través de la línea 'OUT Configuration'. Devuelve
TRUE cuando detecta un pulso (flanco ascendente), FALSE en caso contrario.*/
int PulsOutConfig(void)
{
        if (inportb(Control1) & 0x08) return(TRUE);
```

```
        else return(FALSE);
}

void MandaConfig(void)
{
        /* Manda pulsos negativos de 250ms para comunicar al módulo SerPar que se va a enviar la
        configuración. No deja de enviar pulsos hasta detectar que SerPar ha puesto la línea a 1, indicando
        que está preparado y que va a empezar a mandar pulsos de sincronismo.*/
        delay(500);
        do
        {
                InConfig_0();
                delay(250);
                InConfig_1();
                delay(500);
        }
        while(!PulsOutConfig()&&(!kbhit()));// Espera a que la línea esté a 1


        for(i=BITS_INIC; i>0; i--)
        {
                // Espera la recepción de un flanco positivo de SerPar:
                do
                {
                }
                while(PulsOutConfig()&&(!kbhit()));// Espera a que la línea esté a 0
                do
                {
                }
                while(!PulsOutConfig()&&(!kbhit()));// Espera a que la línea esté a 1

                // Manda el bit correspondiente de la palabra de control:
                if (PalabraCtrol & 0x80000000) InConfig_1();
                else InConfig_0();

                if (PalabraCtrol & 0x80000000) printf("1 ");
                else printf("0 ");

                // Desplaza la palabra de control para el siguiente bit:
                PalabraCtrol <<= 1;
        }
}
```

```
/*****************************************************************************
```

Programa principal

```
*****************************************************************************/
void main(void)
{
        clrscr();
        printf("\n Programa P_INIC2.C.");
        printf("\n\n Manda los parámetros de configuración al módulo SerPar.");
        printf("\n Los parámetros se leerán del fichero VES2.CFG.");

        Inicializacion();
        LeeConfig();
        MandaConfig();
        fclose(FichCfg);
}
```

II.A.1.c. FICHERO DE CONFIGURACIÓN PARA LA LECTURA DEL MÓDULO DE MEMORIA

Fichero: RDMEM2.CFG

/* Fichero de configuración para el programa RDMEM2.C de lectura del módulo de memoria (anexo II.A.1.d). */

```
/*****************************************************************************
  Fichero
*****************************************************************************/
;
; RDMEM2.CFG: Configuration file for program RDMEM2.C. It selects the output files size and type.
; This file can include any number of comment lines, provided that they are preceded by a ';' character.
; The options are enabled with ON and disabled with OFF.
;
; 1. Selection of output file type. DAT files include all data and GPS headers, while GPS files only include
;GPS headers.
;
DAT: ON
GPS: ON
;
; 2. Selection of output file size. With 'Fichs1Hora = ON' the approximate size of the data files will be one
;hour (the exact size depends on the array's configuration). With 'Fichs1Hora = OFF' the files size will be
;lower than 1.44MB (except when using four memory cards, which make every memory bank bigger than
;1.44MB), in order to make the use of diskettes possible.
;       If the array has been configured with a number of memory cards equal to zero, this parameter will be
;ignored, and the file size will be set to ten seconds (see file VES2.CFG or document ManualArrayVes.DOC
;for details).
;
Fichs1Hora: OFF
;
; End of file RDMEM2.CFG
```

II.A.1.d. PROGRAMA DE LECTURA DEL MÓDULO DE MEMORIA

Fichero: RDMEM2.C

/* Programa para la lectura del módulo de memoria de la antena del Vesubio y escritura de los datos en ficheros. Los parámetros de lectura se toman del fichero de configuración RDMEM2.CFG (anexo II.A.1.c). */

```
/*************************************************************************
```

Macros y ficheros de cabecera

```
*************************************************************************/
/* Programa RDMEM2.C: Programa de lectura del módulo de memoria del array del Vesubio. Incorpora una
interfaz gráfica para monitorizar las señales de los distintos canales. La señal sólo se muestra si se ha
seleccionado el modo TEST (bancos de memoria de 10 s), para lo cual hay que poner a 0 el número de bancos
de memoria (ver fichero VES2.CFG o documento ManualArrayVes.DOC para más detalles). RDMEM2.C lee
la configuración del fichero RDMEM2.CFG y escribe las incidencias en el fichero RDMEM2.LOG. Para
obtener ayuda, consultar el fichero de texto RDMEM2.HLP.*/
#include <dos.h>
#include <io.h>
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <string.h>
#include <graphics.h>
#include <ctype.h>

#define  FALSE          0
#define  TRUE           !FALSE
#define BusDatos        0x378    // Primer puerto paralelo
#define Control1        BusDatos+1
#define Control2        BusDatos+2
#define BancoA          1 // Banco A: Señal de selección de banco a 1
#define BancoB          0 // Banco B: Señal de selección de banco a 0

/*************************************************************************
```

Declaración de variables

```
*************************************************************************/
FILE *FichDatos, *FichGPS, *FichLog;
unsigned char Banco, Dato, BufHdrCfg[4], DatoGPS[12];
unsigned long i;
unsigned char Ganancia, Frec, NumTarjsMem;
unsigned long MascaraModsAD, MascaraDisplay, Mascara, GrabaMascara;
unsigned char NumModsAD;
unsigned int NumBytesSeg, NumSegsBanco;
unsigned long NumBytesBanco;
struct  time t;
struct date d;
unsigned char NomFich[50], NombreFicheroDatos[50], NombreFicheroGPS[50];
unsigned char HoraAnterior;
unsigned char TamanoFich;// Número de bancos de memoria que caben en cada fichero. Sólo se usa para la
                         //opción de ficheros cortos.
unsigned char NumBancosEnFich;
```

```
unsigned char NumFichsTest = 0;
struct {
        WrFichTotal:    1;
        WrFichGPS:      1;
        Fichs1Hora:     1;
        Zoom:           1;
        Espera1erBanco: 1;
        Fin:            1;
} Flag = { FALSE, FALSE, FALSE, FALSE, TRUE, FALSE };

// Variables para la representación gráfica, tomadas de P_GRAPH1.C:
int gdriver = DETECT, gmode, errorcode;
int SignalX, SignalY;
int xmax, ymax, x0Signal, y0Signal, xfSignal, yfSignal, xMedioSignal, yMedioSignal;
int PosXCursor, PosYCursor;
int yLinea1, yLinea2, yLinea3, yLinea4, yLinea5, yLinea6, yLinea7;
unsigned char Texto[100];
unsigned char SegundoAnterior;
unsigned char ModADDisplay[9];
union CuatroBytesUnLong
{
        unsigned char Byte[4];
        unsigned long Total;
} CanalDisplay;
unsigned char PrimerPunto;// Booleana, definir flags si hay m s
unsigned char CanalADibujar;
unsigned char TeclaPulsada;
unsigned int CteDisplay;

// Variables para la conversión del tiempo GPS a UTC, tomadas de GPS2UTC2.C:
union CuatroBytesUnFloat
{
        unsigned char Byte[4];
        float Total;
} SegSemana, Offset;
union DosBytesUnFloat
{
        unsigned char Byte[4];
        unsigned int Total;
} NumSemana;
unsigned int IntOffset;// Parte entera del offset
unsigned long IntSegSemana;// Parte entera de SegSemana
float FracSegSemana;// Parte fraccionaria de Segundo Semana
unsigned int IntSegundoUTC, yearUTC;
float SegundoUTC;
unsigned char MinutoUTC, HoraUTC, DiaUTC, MesUTC, YearUTC;
long Temporal;
```

```
/*****************************************************************************
```

Declaración de funciones

```
*****************************************************************************/
// Primitivas de control de las señales de lectura
void AddCLK_1(void);
void AddCLK_0(void);
void CE1_1(void);
void CE1_0(void);
void CE2_1(void);
void CE2_0(void);
unsigned char LeeBankSelect(void);

// Resto de funciones
void InfoPrograma(void);
void LeeFichCfg(void);
void Inicializacion(void);
unsigned char CheckPulsSTOP(void);
void CheckTeclado(void);
void EsperaCambioBanco(void);
void NombraNuevosFichs(void);
void CheckDataDir(void);
void AbreFichLOG();
void AbreFicheros(void);
void CierraFicheros(void);
void LeeParamsCfg(void);
void CalcNumBytesBanco(void);
unsigned char NuevosFichs(void);
void LeeBanco(void);
void SMCW(unsigned char registro, unsigned char dato);// Para el WD

// Funciones para la representación gráfica, tomadas de P_GRAPH1.C:
void IniciaModoGrafico(void);
void movetoxy(int x, int y);
void DibujaPantalla(void);
void SacaReloj(void);
void MuestraParametros(void);
void DisplayDato(unsigned long Dato);
void BorraVentana(int color);
void SacaDatosGPS(void);
void GPS2UTC(void);
void PantallaNormal(void);
void PantallaZoom(void);
```

```
/*************************************************************************
  Definición de las funciones
*************************************************************************/
// Pone la señal 'PC Address CLK' a 1
void AddCLK_1(void)// La señal /AUTO FEED XT del puerto paralelo está invertida
{
        outportb(Control2, inportb(Control2) & 0xFD);
}

// Pone la señal 'PC Address CLK' a 0
void AddCLK_0(void)
{
        outportb(Control2, inportb(Control2) | 0x02);
}

// Pone la señal 'PC /CE1' a 1
void CE1_1(void)
{
        outportb(Control2, inportb(Control2) | 0x04);
}

// Pone la señal 'PC /CE1' a 0
void CE1_0(void)
{
        outportb(Control2, inportb(Control2) & 0xFB);
}

// Pone la señal 'PC CE2' a 1
void CE2_1(void)// La señal /SELECT IN del puerto paralelo está invertida
{
        outportb(Control2, inportb(Control2) & 0xF7);
 }

// Pone la señal 'PC CE2' a 0
void CE2_0(void)// La señal /SELECT IN del puerto paralelo está invertida,
{
        outportb(Control2, inportb(Control2) | 0x08);
}

/* Lee la señal de selección de banco A/B (controlada por el PIC del módulo SerPar). Si dicha señal vale 0, el
PIC escribe en el banco A y el PC lee del B, y viceversa. El pin /ACK del puerto paralelo se lee invertido, de
modo que cuando se lee un 1 la señal A/B est  a 0, y el PC tiene acceso al banco B. */
unsigned char LeeBankSelect(void)
{
        if ((inportb(Control1) & 0x80) == 0x80) return(BancoB);
        else return(BancoA);
}

void InfoPrograma(void)
{
        clrscr();

        fprintf(FichLog, " Program RDMEM2.C: Reading of memory module of Vesuvius Array.");
        fprintf(FichLog, "\n\n Info and selection of output files in RDMEM2.CFG.");
        fprintf(FichLog, "\n Help in file RDMEM2.HLP.\n");
```

```
}

void LeeFichCfg(void)
{
	FILE *FichCfg;
	unsigned char Cadena[40], CadenaCtrol[10];

	if (( FichCfg = fopen("RDMEM2.CFG", "r")) == NULL)
	{
		fprintf(FichLog, "\n Unable to open configuration file RDMEM2.CFG.\n");
		exit(0);
	}

	do
	{
		fscanf(FichCfg, "%s", &Cadena);
		// Si es un comentario salta de línea:
		if (Cadena[0] == ';') fscanf(FichCfg, "\n");
		else
		{
			if (!feof(FichCfg))
			{
				// Si no es un comentario, lee el par metro correspondiente:
				if (!strcmp(Cadena,"DAT:"))
				{
					fscanf(FichCfg, "%s", &CadenaCtrol);
					if (!strcmp(CadenaCtrol,"ON")) Flag.WrFichTotal = TRUE;
				}

				if (!strcmp(Cadena,"GPS:"))
				{
					fscanf(FichCfg, "%s", &CadenaCtrol);
					if (!strcmp(CadenaCtrol,"ON")) Flag.WrFichGPS = TRUE;
				}

				if (!strcmp(Cadena,"Fichs1Hora:"))
				{
					fscanf(FichCfg, "%s", &CadenaCtrol);
					if (!strcmp(CadenaCtrol,"ON")) Flag.Fichs1Hora = TRUE;
				}
			}
		}
	}
	while (!feof(FichCfg));

	if((!Flag.WrFichTotal) && (!Flag.WrFichGPS))
	{
		fprintf(FichLog, "\n WARNING!: No output files selected");
		fprintf(FichLog, "\n Edit RDMEM2.CFG to change selection\n");
	}
}

void Inicializacion(void)
{
	outportb(Control2, inportb(Control2) | 0x20);// Pone el bit DIR a 1 para permitir la lectura del
                                                 //registro de datos
```

```
	CE2_1();// Inicialmente pone 'CE2' a 1 y 'AddCLK' a 1
	AddCLK_1();
	CteDisplay = 55924;// 16777216cuentas/300pixels (valor de pantalla normal)
}

// Si la señal 'Select' está a 0 el botón ha sido pulsado y se devuelve TRUE, y viceversa. Además se pone el
flag de fin a TRUE, para utilizar lo que más interese en cada caso.
unsigned char CheckPulsSTOP(void)
{
	if (inportb(Control1) & 0x10)
	{
		return(FALSE);// Flag.Fin no se pone a FALSE aquí porque podría interferir en la función
				//'CheckTeclado()'
	}
	else
	{
		Flag.Fin = TRUE;
		return(TRUE);
	}
}

void EsperaCambioBanco(void)
{
	/* La señal de selección de banco es controlada por el PIC. Si vale 0, el PIC escribe en el banco A y
	el PC lee del B, y viceversa */
	Banco = LeeBankSelect();// Asigna a 'Banco' el valor 1 (banco A) o 0 (banco B)
	if (Banco == BancoA)
	{
		fprintf(FichLog, "\n Waiting access to Bank B...");
		movetoxy(0, yLinea5);
		printf("Program status: Waiting access to memory bank B");
		do
		{
			Banco = LeeBankSelect();
			SacaReloj();
			CheckTeclado();
			CheckPulsSTOP();
		}
		while((Banco == BancoA)&&(!Flag.Fin));// Si estamos en el banco A, hay que esperar a
							//tener acceso al banco B
	}
	else
	{
		fprintf(FichLog, "\n Waiting access to Bank A...");
		movetoxy(0, yLinea5);
		printf("Program status: Waiting access to memory bank A");
		do
		{
			Banco = LeeBankSelect();
			SacaReloj();
			CheckTeclado();
			CheckPulsSTOP();
		}
		while((Banco == BancoB)&&(!Flag.Fin));// Si estamos en el banco B, hay que esperar a
							//tener acceso al banco A
	}
```

```
	if (Flag.Espera1erBanco) Flag.Espera1erBanco = FALSE;
}

void NombraNuevosFichs(void)
{
	if (NumTarjsMem == 0)
	{
		sprintf(NomFich, "..\\DATOS\\TEST%02u", NumFichsTest);
		NumFichsTest++;
	}
	else sprintf(NomFich, "..\\DATOS\\%02u%02u%02u%02u",d.da_mon,d.da_day, t.ti_hour, t.ti_min);

	if (Flag.WrFichTotal)
	{
		sprintf(NombreFicheroDatos, "%s.DAT", NomFich);
		fprintf(FichLog, "\n New data file: %s", NombreFicheroDatos);
		movetoxy(46, yLinea4);
		printf("Current file:%s", NombreFicheroDatos);
	}
	if (Flag.WrFichGPS)
	{
		sprintf(NombreFicheroGPS, "%s.GPS", NomFich);
		fprintf(FichLog, "\n New GPS header file: %s", NombreFicheroGPS);
		if (!Flag.WrFichTotal)// Si no están habilitados los ficheros DAT se saca el nombre del GPS
		{
			movetoxy(46, yLinea4);
			printf("Current file:%s", NombreFicheroGPS);
		}
	}
	// Si no están habilitados los ficheros DAT ni los GPS se escribe una marca en pantalla
	if ((!Flag.WrFichTotal) && (!Flag.WrFichGPS))
	{
		movetoxy(46, yLinea4);
		printf("Current file: No files enabled!    ");
	}
}

void BorraFichsTEST(void)
{
	unsigned char NumFichTEST, MasFichsTEST;

	// Cuando se va a hacer un TEST de bancos rápidos se borran los anteriores ficheros con nombre
	//TESTxx.DAT y TESTxx.GPS, para no confundirlos con los nuevos.
	if(NumTarjsMem == 0)
	{
		fprintf(FichLog, "\n Erasing previous TEST files:");
		NumFichTEST = 0;
		do
		{
			sprintf(NombreFicheroDatos, "..\\DATOS\\TEST%02u.DAT", NumFichTEST);
			sprintf(NombreFicheroGPS, "..\\DATOS\\TEST%02u.GPS", NumFichTEST);
			MasFichsTEST = FALSE;// Por defecto a FALSE
			if (remove(NombreFicheroDatos) == 0)
			{
				MasFichsTEST = TRUE;
				fprintf(FichLog, " File %s erased\n", NombreFicheroDatos);
```

```
                        }
                        else
                        {
                                MasFichsTEST = FALSE;
                                fprintf(FichLog, " File %s not found\n", NombreFicheroDatos);
                        }

                        if (remove(NombreFicheroGPS) == 0)
                        {
                                MasFichsTEST = TRUE;
                                fprintf(FichLog, " File %s erased\n", NombreFicheroGPS);
                        }
                        else
                        {
                                MasFichsTEST = FALSE;
                                fprintf(FichLog, " File %s not found\n", NombreFicheroDatos);
                        }
                        NumFichTEST++;
                }
                while (MasFichsTEST);// Mientras se sigan encontrando ficheros TEST
        }
}

void AbreFicheros(void)
{
        if (Flag.WrFichTotal)
        {
                if (( FichDatos = fopen(NombreFicheroDatos, "ab")) == NULL)
                {
                        fprintf(FichLog, "\n Unable to open file %s to append data",
                        NombreFicheroDatos);
                        exit(0);
                }
        }
        if (Flag.WrFichGPS)
        {
                if (( FichGPS = fopen(NombreFicheroGPS, "ab")) == NULL)
                {
                        fprintf(FichLog, "\n Unable to open file %s to append data", NombreFicheroGPS);
                        exit(0);
                }
        }
}

void CierraFicheros(void)
{
        if (Flag.WrFichTotal) fclose(FichDatos);
        if (Flag.WrFichGPS) fclose(FichGPS);
}
```

```
/* Lee los cuatro primeros bytes del banco de memoria, correspondientes a los parámetros de configuración,
pero no los escribe en disco porque todavía no sabemos el tipo de ficheros que se van a escribir (antes hay que
saber si el número de tarjetas de memoria es 0). */
void LeeParamsCfg(void)
{
	CE2_1();// Lee el dato
	Dato = inportb(BusDatos);
	BufHdrCfg[0] = Dato;// Se meten en un búffer para escribirlo luego en el fichero de datos
	MascaraModsAD = Dato;
	CE2_0();
	AddCLK_0();// Incrementa la dirección de memoria
	AddCLK_1();

	CE2_1();
	Dato = inportb(BusDatos);
	BufHdrCfg[1] = Dato;
	MascaraModsAD = ((MascaraModsAD << 8) | Dato);
	CE2_0();
	AddCLK_0();
	AddCLK_1();

	CE2_1();
	Dato = inportb(BusDatos);
	BufHdrCfg[2] = Dato;
	Ganancia = Dato;
	CE2_0();
	AddCLK_0();
	AddCLK_1();

	CE2_1();
	Dato = inportb(BusDatos);
	BufHdrCfg[3] = Dato;
	CE2_0();
	AddCLK_0();
	AddCLK_1();
	NumTarjsMem = Dato & 0x07;
	Dato >>= 3;
	switch(Dato)
	{
		case 1:
			Frec = 50;
			break;
		case 2:
			Frec = 100;
			break;
		case 4:
			Frec = 200;
			break;
		default:
			Frec = 0;
	}
```

```
        // Si la opción de escribir ficheros de una hora no está activada se escribirán ficheros cortos. Si es
        posible, menores de 1.44MB, para que quepan en disquetes. Teniendo en cuenta que cada tarjeta de
        memoria tiene 512kB por banco, esto es posible cuando se usan una o dos tarjetas, pero no cuando se
        usan cuatro. Por otra parte, si el número de tarjetas de memoria es 0 se tomarán sólo los diez
        primeros segundos de cada banco y se escribirán ficheros de esa longitud, independientemente de si
        Flag.Fichs1Hora está o no activado. */
        switch(NumTarjsMem)
        {
                case 0:
                        TamanoFich = 1;// Prueba de bancos cortos: 1 banco por fichero
                        Flag.Fichs1Hora = FALSE;// Si el número de tarjetas de memoria es 0 se escriben
                        break;                          //ficheros de diez segundos.
                case 1:
                        TamanoFich = 2;// 2 bancos por fichero
                        break;
                case 2:
                        TamanoFich = 1;// 1 banco por fichero
                        break;
                case 4:
                        TamanoFich = 1;// 1 banco por fichero, ficheros de 2MB aprox.
                        break;
                default:
                        fprintf(FichLog, "\n Incorrect number of memory cards: %u", NumTarjsMem);
                        fprintf(FichLog, "\n Exit program");
                        CierraFicheros();
                        exit(0);
        }
}

void CalcNumBytesBanco(void)
{
        unsigned int i;

        // Calcula el número de módulos A/D contando los unos de MascaraModsAD
        Mascara = 0x0001;
        NumModsAD = 0;
        // Para Frec = 200 sólo se tienen en cuenta los módulos 0 y 1 de cada línea serie:
        if (Frec == 200)
        {
                Mascara <<= 8;
                for (i=0; i<8; i++)
                {
                        if (MascaraModsAD & Mascara) NumModsAD++;
                        Mascara <<= 1;
                }
        }
        else
        {
                for (i=0; i<16; i++)
                {
                        if (MascaraModsAD & Mascara) NumModsAD++;
                        Mascara <<= 1;
                }
        }
```

```
	/* El número de segundos tiene que coincidir con el calculado por el PIC de SerPar, por lo que se
	hace igual que allí (programa SERPAR6.ASM): se calcula para una tarjeta de memoria y 50mps.
	Luego se multiplica (desplazando a la izquierda) y/o divide (desplazando a la derecha) en caso de
	que se use m s de una tarjeta de memoria o una frecuencia mayor de 50mps, respectivamente.*/
	if (NumTarjsMem == 0)// P_RDMEM6, prueba para bancos de 10 segundos:
	{
		NumSegsBanco = 10;
	}
	else
	{
		NumBytesSeg = NumModsAD*9*50+12;
		NumSegsBanco = 524288/NumBytesSeg;
		switch (NumTarjsMem)
		{
			case 1:
				break;
			case 2:
				NumSegsBanco <<= 1;
				break;
			case 4:
				NumSegsBanco <<= 2;
		}
		switch (Frec)
		{
			case 50:
				break;
			case 100:
				NumSegsBanco >>= 1;
				break;
			case 200:
				NumSegsBanco >>= 2;
		}
	}

	NumBytesSeg = NumModsAD*9*Frec+12;
	// En cada banco de memoria se guarda el máximo número posible de segundos ENTEROS, por lo
	//que no se usarán todos los bytes.
	NumBytesBanco = (long)NumSegsBanco*NumBytesSeg+4;
}

void LeeBanco(void)
{
	unsigned int NumSegundo, i, j, k, m;
	unsigned char UltimosBytesDatos[4];

	fprintf(FichLog, "\n %2d:%02d:%02d.%02d %d/%d/%d",
			t.ti_hour, t.ti_min, t.ti_sec, t.ti_hund, d.da_day, d.da_mon, d.da_year);
	fprintf(FichLog, " Reading memory bank...\n");
	movetoxy(0, yLinea5);
	printf("Program status: Reading memory bank                    ");

	// 1. Posiciona el cursor en el origen del recuadro que hemos reservado para monitorizar la señal:
	if (NumTarjsMem == 0)// Si es distinto de 0 no se refresca la pantalla, porque no se pinta la señal.
	{
		if (Flag.Zoom) PantallaZoom();// Borra y vuelve a pintar el recuadro (en general bastaría
```

```
        else PantallaNormal();// 'BorraVentana()', pero cuando hay saturación se queda marcada.
}
setlinestyle(0, 1, 1);
PosXCursor =    x0Signal;
PosYCursor =    yfSignal;
moveto(PosXCursor, PosYCursor);
PrimerPunto = TRUE;// Para que pinte desde el segundo punto, que es el primero de verdad.

for(NumSegundo=0; NumSegundo<NumSegsBanco; NumSegundo++)
{
        /* 1. Antes de escribir la cabecera GPS en el fichero GPS escribimos los cuatro últimos
        bytes de datos escritos en el fichero DAT, para que las cabeceras GPS queden alineadas.
        Esto se hace en todos los segundos salvo el primero, en el que ya se han escrito los 4 bytes
        de parámetros de configuración: */
        if (NumSegundo != 0)
        {
                for(i=0; i<4; i++)
                {
                        if (Flag.WrFichGPS) fprintf(FichGPS, "%c", UltimosBytesDatos[i]);
                }
        }

        // 2. Lee la cabecera GPS:
        for(i=0; i<12; i++)
        {
                CE2_1();
                Dato = inportb(BusDatos);// Lee el dato
                CE2_0();
                if (Flag.WrFichTotal) fprintf(FichDatos,"%c",Dato);//Escribe el dato en VES.DAT
                if (Flag.WrFichGPS) fprintf(FichGPS, "%c", Dato);
                AddCLK_0();// Incrementa la dirección de memoria
                AddCLK_1();
                // Si es la cabecera GPS del segundo segundo (que corresponde al primer segundo
                //de datos) se mete en un búffer para sacar luego los datos por pantalla.
                if (NumSegundo == 1) DatoGPS[i] = Dato;
        }

        // 3. Lee los datos
        for(i=0; i<Frec; i++)
        {
                Mascara = 0x8000;
                for(j=0; j<4; j++)// 4 módulos por línea serie (lectura en paralelo)
                {
                        // Al principio de cada bloque de 4 módulos A/D se graba el estado actual
                        //de la máscara, que nos hará falta para cada uno de los nueve bytes.
                        GrabaMascara = Mascara;
                        for(k=0; k<9; k++)// 9 bytes por dato
                        {
                                Mascara = GrabaMascara;
                                for(m=0; m<4; m++)// Lectura en paralelo
                                {
                                        if (Mascara & MascaraModsAD)
                                        {
                                                CE2_1();
                                                Dato = inportb(BusDatos);// Lee el dato
                                                CE2_0();
```

```
                                                // Escribe el dato en fichero DAT:
                                                 if (Flag.WrFichTotal) fprintf(FichDatos,
                                                 "%c", Dato);
                                                 // Actualiza el buffer con los últimos datos
                                                 //escritos en DAT
                                                UltimosBytesDatos[0]=UltimosBytesDatos[1];
                                                UltimosBytesDatos[1]=UltimosBytesDatos[2];
                                                UltimosBytesDatos[2]=UltimosBytesDatos[3];
                                                UltimosBytesDatos[3]=Dato;
                                                AddCLK_0();// Incrementa la dir. de memoria
                                                AddCLK_1();
                                        }
                                        if (Mascara & MascaraDisplay)
                                        {
                                                ModADDisplay[k] = Dato;
                                        }
                                        Mascara >>= 1;
                                }
                        }
                }
                // Con 50mps se dibujan todos los datos, pero con 100mps sólo uno de cada dos y
                //con 200mps uno de cada cuatro (si no, no caben):
                 if ((i%(Frec/50)) == 0)
                 {
                        CanalDisplay.Total = 0;
                        switch(CanalADibujar)
                        {
                                case 0:
                                        CanalDisplay.Byte[0] = ModADDisplay[2];
                                        CanalDisplay.Byte[1] = ModADDisplay[1];
                                        CanalDisplay.Byte[2] = ModADDisplay[0];
                                        break;
                                case 1:
                                        CanalDisplay.Byte[0] = ModADDisplay[5];
                                        CanalDisplay.Byte[1] = ModADDisplay[4];
                                        CanalDisplay.Byte[2] = ModADDisplay[3];
                                        break;
                                case 2:
                                        CanalDisplay.Byte[0] = ModADDisplay[8];
                                        CanalDisplay.Byte[1] = ModADDisplay[7];
                                        CanalDisplay.Byte[2] = ModADDisplay[6];
                                        break;
                                default:// Por defecto saca el canal 0:
                                        CanalDisplay.Byte[0] = ModADDisplay[2];
                                        CanalDisplay.Byte[1] = ModADDisplay[1];
                                        CanalDisplay.Byte[2] = ModADDisplay[0];
                        }
                        // Sólo se muestra la señal si estamos en la opción de test del
                        //módulo de memoria (NumTarjsMem = 0):
                        if(NumTarjsMem == 0) DisplayDato(CanalDisplay.Total);
                }
        }
    }
}
```

```
unsigned char NuevosFichs(void)
{
	gettime(&t);
	getdate(&d);
	if (Flag.Fichs1Hora)
	{
		if(t.ti_hour == HoraAnterior) return(FALSE);
		else
		{
			HoraAnterior = t.ti_hour;
			return(TRUE);
		}
	}
	else
	{
		NumBancosEnFich++;
		if (NumBancosEnFich == TamanoFich)
		{
			NumBancosEnFich = 0;
			return(TRUE);
		}
		else return(FALSE);
	}
}

// Función para escritura de registros en el chip super I/O del Lippert. Se usa para programar el WD (ver pág.
//10 del manual del Lippert).
void SMCW(unsigned char registro, unsigned char dato)
{
	outportb(0x370, 0x55);// Modo configuración
	outportb(0x370, registro);
	outportb(0x371, dato);
	outportb(0x370, 0xAA);// Fin del modo configuración
}

void InfoParametros(void)
{
	fprintf(FichLog, "\n\n PARAMETERS READ FROM MEMORY MODULE:");
	fprintf(FichLog, "\n A/D modules mask: 0x%lX", MascaraModsAD);
	fprintf(FichLog, "\n Number of A/D modules: %u", NumModsAD);
	if (Frec == 200) fprintf(FichLog, "\n (see RDMEM2.CFG for limits on number of modules with
	200sps)");
	fprintf(FichLog, "\n Gain: %u", Ganancia);
	if (Frec != 0) fprintf(FichLog, "\n Sampling rate: %umps", Frec);
	else
	{
		fprintf(FichLog, "\n Incorrect samplig rate read from memory module");
		//exit(0);
	}
	fprintf(FichLog, "\n Number of memory cards: %u", NumTarjsMem);
	if (NumTarjsMem == 0) fprintf(FichLog, "\n (Short banks test selected, see RDMEM2.CFG for
	details)");
	fprintf(FichLog, "\n Number of bytes per second: %u", NumBytesSeg);
	fprintf(FichLog, "\n Number of seconds per memory bank: %u", NumSegsBanco);
	fprintf(FichLog, "\n Number of bytes per memory bank: %lu\n", NumBytesBanco);
}
```

```
void WrHdrCfgFichs(void)
{
	unsigned char i;

	for (i=0; i<4; i++)
	{
		if (Flag.WrFichTotal) fprintf(FichDatos, "%c", BufHdrCfg[i]);
		if (Flag.WrFichGPS) fprintf(FichGPS, "%c", BufHdrCfg[i]);
	}
}

void IniciaModoGrafico(void)
{
	/* initialize graphics and local variables */
	initgraph(&gdriver, &gmode, "");

	/* read result of initialization */
	errorcode = graphresult();
	if (errorcode != grOk)  /* an error occurred */
	{
		fprintf(FichLog, "Graphics error: %s\n", grapherrormsg(errorcode));
		exit(1); /* terminate with an error code */
	}
}

void DibujaPantalla(void)
{
	// Inicialización de coordenadas:
	xmax = getmaxx();
	ymax = getmaxy();
	x0Signal = 100;
	xfSignal = 600;
	y0Signal = 100;
	yfSignal = 400;
	xMedioSignal = (x0Signal + xfSignal)/2;
	yMedioSignal = (y0Signal + yfSignal)/2;
	settextjustify(LEFT_TEXT, CENTER_TEXT);
	yLinea1 = 0;
	yLinea2 = 1;
	yLinea3 = 2;
	yLinea4 = 3;
	yLinea5 = 4;
	yLinea6 = 28;
	yLinea7 = 29;

	// Pantalla gráfica (cuadro, líneas de referencia y valores de referencia)
	PantallaNormal();

	// Texto inicial (con funciones outtextxy no se puede refrescar una variable porque no borra los
	//caracteres anteriores):
	movetoxy(0, yLinea1);
	printf("RDMEM2.C: Mt. Vesuvius array monitoring program (see RDMEM2.HLP for help)");
	movetoxy(0, yLinea3);
	printf("Sampling rate:	Gain:	Memory cards:	Active A/D: ");
	movetoxy(0, yLinea4);
```

```
        printf("Now displaying:          Channel:    Current file:");
        movetoxy(0, yLinea5);
        printf("Program status: ");
        movetoxy(0, yLinea6);
        printf("GPS time:                PC time:");
        movetoxy(0, yLinea7);
        printf("GPS status: ");
}

// El gotoxy() no funciona bien. Función tomada del ejemplo de int86():
void movetoxy(int x, int y)
{
        union REGS regs;

        regs.h.ah = 2;  /* set cursor position */
        regs.h.dh = y;
        regs.h.dl = x;
        regs.h.bh = 0;  /* video page 0 */
        int86(0x10, &regs, &regs);
}

void SacaReloj(void)
{
        gettime(&t);
        getdate(&d);
        if (t.ti_sec != SegundoAnterior)
        {
                movetoxy(38, yLinea6);
                printf("PC time: %02u:%02u:%02u  %02u/%02u/%02u", t.ti_hour, t.ti_min, t.ti_sec,
                d.da_day, d.da_mon, d.da_year);
                SegundoAnterior = t.ti_sec;
        }
}

void MuestraParametros(void)
{
        movetoxy(0, yLinea3);
        printf("Sampling rate: %3usps  Gain: %3u  Memory cards: %u  Active A/D: ", Frec, Ganancia,
        NumTarjsMem);
        // Saca la máscara de módulos A/D en binario, que es más intuitivo:
        Mascara = 0x8000;
        for(i=0; i<16; i++)
        {
                if (MascaraModsAD & Mascara) printf("1");
                else printf("0");
                Mascara >>= 1;
        }
        // Inicializa la máscara de display (que indica qué módulo A/D se está monitorizando) con la primera
        //estación activa:
        Mascara = 0x8000;
        i = 0;
        while((!(MascaraModsAD & Mascara)) && (i<16))
        {
                Mascara >>= 1;
                i++;
        }
```

```
        MascaraDisplay = Mascara;

        // Muestra el módulo A/D cuya señal se va a monitorizar inicialmente (lo señala con una X)
        movetoxy(0, yLinea4);
        printf("Now displaying: ");
        Mascara = 0x8000;
        for(i=0; i<16; i++)
        {
                if (MascaraModsAD & Mascara)
                {
                        if (Mascara & MascaraDisplay) printf("X");
                        else printf("1");
                }
                else printf("0");
                Mascara >>= 1;
        }
        CanalADibujar = 0;
        printf("  Channel: 0");

        movetoxy(0, yLinea5);
        printf("Program status:                  ");
        movetoxy(0, yLinea6);
        printf("GPS time:                      PC time:");
        movetoxy(0, yLinea7);
        printf("GPS status:                  ");

        // Si el número de tarjetas de memoria no es 0, se saca un mensaje diciendo que la señal no se va a
        //mostrar.
        if (NumTarjsMem != 0)
        {
                setfillstyle(1, LIGHTGRAY);
                bar(x0Signal+1, y0Signal+1, xfSignal-1, yfSignal-1);
                setcolor(BLUE);
                settextjustify(CENTER_TEXT, CENTER_TEXT);
                outtextxy(xMedioSignal, yMedioSignal, " Data display only in TEST mode!");
        }
}

// Comprueba si existe el directorio de datos y, en caso contrario, sale dando un aviso:
void CheckDataDir(void)
{
        if (( FichDatos = fopen("..\\DATOS\\PRUEBA.DAT", "wb")) == NULL)
        {
                printf("\n Directory ..\\DATA does not exist!\n");
                fprintf(FichLog, "\n Directory ..\\DATA does not exist!\n");
                exit(0);
        }
        else
        {
                fclose(FichDatos);
                remove("..\\DATOS\\PRUEBA.DAT");
        }
}
```

```
void AbreFichLOG(void)
{
	if (( FichLog = fopen("RDMEM2.LOG", "w")) == NULL)
	{
		printf("\n Unable to open file RDMEM2.LOG.\n");
		exit(0);
	}
}

void DisplayDato(unsigned long Dato)
{
	long CalculaPosYCursor;// Para los cálculos de la pantalla de zoom no se puede usar directamente
				//'PosYCursor' porque los números son grandes
	PosXCursor++;
	if (Flag.Zoom)// Si estamos en zoom le restamos el origen
	{
		if (Dato >= 8387583) Dato -= 8387583;
		else Dato = 0;
	}
	CalculaPosYCursor = yfSignal - Dato/CteDisplay;
	if (CalculaPosYCursor > yfSignal) PosYCursor = yfSignal;
	else
	{
		if (CalculaPosYCursor < y0Signal) PosYCursor = y0Signal;
		else PosYCursor = CalculaPosYCursor;
	}
	if(PrimerPunto)
	{
		PrimerPunto = FALSE;
		moveto(PosXCursor, PosYCursor);
	}
	else
	{
		if (Flag.Zoom) setcolor(LIGHTCYAN);
		else setcolor(BLUE);
		lineto(PosXCursor, PosYCursor);
	}
}

void BorraVentana(int color)
{
	setfillstyle(1, color);
	bar(x0Signal+1, y0Signal+1, xfSignal-1, yfSignal-1);
	if (Flag.Zoom) setcolor(LIGHTBLUE);
	else setcolor(RED);
	setlinestyle(1, 1, 1);
	line(x0Signal+1, yMedioSignal, xfSignal, yMedioSignal);
}

void CheckTeclado(void)
{
	if(kbhit())
	{
		TeclaPulsada = tolower(getch());
		switch(TeclaPulsada)
		{
```

```
case 'c': // Cambio de canal
        if (!Flag.Espera1erBanco)// Sólo se acepta la tecla si ya se han leído los
                                        //parámetros de operación
        {
                CanalADibujar++;
                if (CanalADibujar == 3) CanalADibujar = 0;
                movetoxy(43, yLinea4);
                printf("%u", CanalADibujar);
        }
        break;
case 'q': // Salir
        Flag.Fin = TRUE;
        break;
case 's': // Cambio de escala (conmuta entre normal y zoom)
        if (Flag.Zoom)
        {
                Flag.Zoom = FALSE;
                if (NumTarjsMem == 0) PantallaNormal();
                CteDisplay = 55924;// 16777216cuentas/300pixels
        }
        else
        {
                Flag.Zoom = TRUE;
                if (NumTarjsMem == 0) PantallaZoom();
                CteDisplay = 7;// (16777216/8192)cuentas/300pixels
        }
        break;
case 'm': // Cambio de módulo A/D
        if (!Flag.Espera1erBanco)// Sólo se acepta la tecla si ya se han leído los
                                        //parámetros de operación
        {
                Mascara = MascaraDisplay;
                do
                {
                        Mascara >>= 1;
                }
                while((!(MascaraModsAD & Mascara)) && (Mascara != 0));
                // Si no había más estaciones activas (Mascara = 0), se empieza
                //la comprobación desde el principio:
                if(Mascara == 0)
                {
                        Mascara = 0x8000;
                        i = 0;
                        while((!(MascaraModsAD & Mascara)) && (i<16))
                        {
                                Mascara >>= 1;
                                i++;
                        }
                }
                // Se actualiza la máscara de display con la posición calculada:
                MascaraDisplay = Mascara;
                // Muestra el módulo A/D cuya señal se va a monitorizar (lo
               //señala con una X)
                movetoxy(0, yLinea4);
                printf("Now displaying: ");
                Mascara = 0x8000;
```

```
                                        for(i=0; i<16; i++)
                                        {
                                                if (MascaraModsAD & Mascara)
                                                {
                                                        if (Mascara & MascaraDisplay) printf("X");
                                                        else printf("1");
                                                }
                                                else printf("0");
                                                Mascara >>= 1;
                                        }
                                        CanalADibujar = 0;
                                        printf("  Channel: 0");
                                }
                                break;
                } // Fin del 'switch'
        }
}

void SacaDatosGPS(void)
{
        // 1. Procesa el status GPS:
        movetoxy(0, yLinea7);
        printf("GPS status:");

        // Byte 0:
        switch(DatoGPS[0])
        {
                // A. Posibles valores del byte 0 del paquete 0x46 del protocolo TSIP:
                case 0x00:
                        printf("Doing position fixes");
                        break;
                case 0x01:
                        printf(" Don't have GPS time yet");
                        break;
                case 0x02:
                        printf(" Need initialization");
                        break;
                case 0x03:
                        printf(" PDOP is too high");
                        break;
                case 0x08:
                        printf(" No usable satellites");
                        break;
                case 0x09:
                        printf(" Only 1 usable satellite");
                        break;
                case 0x0A:
                        printf(" Only 2 usable satellites");
                        break;
                case 0x0B:
                        printf(" Only 3 usable satellites");
                        break;
                case 0x0C:
                        printf(" The chosen satellite is useless");
                        break;
                // B. Otros valores de control, introducidos para el array del Vesubio:
```

```
		case 0x30:
			printf(" First data from clock module");
			break;
		case 0x31:
			printf(" No data from clock module");
			break;
		case 0x32:
			printf(" SerPar PIC overflow error!");
			break;
		case 0x33:
			printf(" SerPar PIC data frame error!");
			break;
		case 0x34:
			printf(" Time out error!");
			break;
}// Fin del 'switch'
printf("            ");// Para limpiar posibles restos de mensajes anteriores


// Byte 1:
if ((DatoGPS[1] & 0x01) != 0x00) printf(" / No battery back-up");
if ((DatoGPS[1] & 0x10) != 0x00) printf(" / Antenna feed line fault");

// 2. Procesa la hora GPS:
movetoxy(0, yLinea6);
printf("GPS time:");

// Sólo se procesan los datos de tiempo cuando hay al menos un satélite:
if ((DatoGPS[0] != 0x00)&&(DatoGPS[0] != 0x09)&&(DatoGPS[0] != 0x0A)&&(DatoGPS[0] !=
0x0B)) printf(" --                    ");
else// Procesa los datos de tiempo del GPS (código tomado de GPS2UTC2.C):
{
		// Convierte los datos leídos al formato necesario para usar el código del programa
		//GPS2UTC2:
		SegSemana.Byte[0] = DatoGPS[5];
		SegSemana.Byte[1] = DatoGPS[4];
		SegSemana.Byte[2] = DatoGPS[3];
		SegSemana.Byte[3] = DatoGPS[2];
		NumSemana.Byte[0] = DatoGPS[7];
		NumSemana.Byte[1] = DatoGPS[6];
		Offset.Byte[0] = DatoGPS[11];
		Offset.Byte[1] = DatoGPS[10];
		Offset.Byte[2] = DatoGPS[9];
		Offset.Byte[3] = DatoGPS[8];

		IntSegSemana = SegSemana.Total;// Se queda con la parte entera para los cálculos
		FracSegSemana = SegSemana.Total-IntSegSemana;

		// Antes de hacer la conversión se le resta el offset a los segundos GPS:
		IntOffset = Offset.Total;// Pasa el offset a entero
		for (i=0; i<IntOffset; i++)
		{
			if (IntSegSemana == 0)
			{
				IntSegSemana = 604799;
				NumSemana.Total--;
```

```
                        }
                        else IntSegSemana--;
                }

                GPS2UTC();
                SegundoUTC = IntSegundoUTC + FracSegSemana;

                // No se usa la parte fraccionaria del segundo, porque la primera muestra se toma 200us
                //después del flanco de segundo (.0002 segundos):
                printf(" %02u:%02u:%02u.0002 %2u/%02u/%04u", HoraUTC, MinutoUTC,
                IntSegundoUTC, DiaUTC, MesUTC, yearUTC);
        }
}

void GPS2UTC(void)
{
        unsigned int DiasYearActual, Contador;
        unsigned long Dias, DiasSin1980;
        // Para que el primer mes sea el 1 (no el 0) definimos un array de trece elementos:
        unsigned char DiasMes[13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
        unsigned char Bisiesto, YearsNormales, YearsBisiestos;

        // TRANSFORMA LA INFORMACIÓN DE TIEMPO GPS A FORMATO UTC
        Dias = NumSemana.Total*7 + IntSegSemana/86400 + 1;

        if (Dias < 361)// Año 1980
        {
                yearUTC = 1980;
                DiasMes[2] = 29;// 1980 fue bisiesto
                Dias += 5;// El tiempo GPS empieza a contar a las 00:00:00 del 6 de enero de 1980, por lo
                        //que sumamos los cinco días transcurridos.
                Contador = 0;// Cálculo del mes actual
                Temporal = 0;
                do
                {
                        Contador++;
                        Temporal += DiasMes[Contador];
                }
                while(Temporal<Dias);
                MesUTC = Contador;

                Contador = 0;// Cálculo del día actual
                Temporal = 0;
                for(Contador = 1; Contador < MesUTC; Contador++)
                {
                        Temporal += DiasMes[Contador];
                }
                DiaUTC = Dias - Temporal;

                // Cálculo de la hora actual
                Temporal = IntSegSemana % 86400;// Segundos del día actual
                HoraUTC = Temporal/3600;
                Temporal = Temporal % 3600;
                MinutoUTC = Temporal/60;
                IntSegundoUTC = Temporal % 60;
        }
```

```
else// Año distinto de 1980
{
	DiasSin1980 = Dias -361;

	YearsBisiestos = 0;// Cálculo de los años bisiestos y no bisiestos
	YearsNormales = 0;
	Contador = 0;
	do
	{
		Contador++;
		if (Contador % 4 == 0)
		{
			Bisiesto = TRUE;
			YearsBisiestos++;
		}
		else
		{
			Bisiesto = FALSE;
			YearsNormales++;
		}
		Temporal = YearsNormales*365 + YearsBisiestos*366;
	}
	while(Temporal < DiasSin1980);

	if(Bisiesto)
	{
		YearsBisiestos--;// Restamos 1 porque queremos años enteros.
		DiasMes[2] = 29;
	}
	else
	{
		YearsNormales--;
		DiasMes[2] = 28;
	}

	YearUTC = 81 + YearsBisiestos + YearsNormales;// Cálculo del año actual
	if (YearUTC > 99) YearUTC -= 100;
	if ((NumSemana.Total > 1042) || ((NumSemana.Total == 1042) && (IntSegSemana >=
	518400)))
	{
		yearUTC = YearUTC + 2000;
	}
	else yearUTC = YearUTC + 1900;

	Contador = 0;// Cálculo del mes actual
	Temporal = 0;
	DiasYearActual = DiasSin1980 - YearsNormales*365 - YearsBisiestos*366;
	do
	{
		Contador++;
		Temporal += DiasMes[Contador];
	}
	while(Temporal<DiasYearActual);
	MesUTC = Contador;
```

```
                Contador = 0;// Cálculo del día actual
                Temporal = 0;
                for(Contador = 1; Contador < MesUTC; Contador++)
                {
                        Temporal += DiasMes[Contador];
                }
                DiaUTC = DiasYearActual - Temporal;

                // Cálculo de la hora actual
                Temporal = IntSegSemana % 86400;// Segundos del día actual
                HoraUTC = Temporal/3600;
                Temporal = Temporal % 3600;
                MinutoUTC = Temporal/60;
                IntSegundoUTC = Temporal % 60;
        }
}

void PantallaNormal(void)
{
        setfillstyle(1, LIGHTGRAY);
        //bar(x0Signal+1, y0Signal+1, xfSignal-1, yfSignal-1);
        bar(0, x0Signal-15, xmax, yfSignal+40);

        // Marco:
        setcolor(WHITE);
        setlinestyle(0, 1, 2);
        rectangle(0, x0Signal-15, xmax, yfSignal+40);

        // Líneas de referencia:
        setcolor(RED);
        setlinestyle(1, 1, 1);
        line(x0Signal, y0Signal, x0Signal, yfSignal+20);
        line(xfSignal, y0Signal, xfSignal, yfSignal+20);
        line(x0Signal-20, y0Signal, xfSignal, y0Signal);
        line(x0Signal-20, yfSignal, xfSignal, yfSignal);
        line(x0Signal-20, yMedioSignal, xfSignal, yMedioSignal);

        // Valores de referencia:
        setcolor(BLACK);
        settextjustify(LEFT_TEXT, CENTER_TEXT);
        outtextxy(10, y0Signal, "16777215");
        outtextxy(10, yMedioSignal, " 8388607");
        outtextxy(10, yfSignal, "0 counts");
        settextjustify(CENTER_TEXT, TOP_TEXT);
        outtextxy(x0Signal, yfSignal+25, "GPS time");
        outtextxy(xfSignal, yfSignal+25, "+10s");

        BorraVentana(LIGHTGRAY);
}

void PantallaZoom(void)
{
        setfillstyle(1, BLUE);
        //bar(x0Signal+1, y0Signal+1, xfSignal-1, yfSignal-1);
        bar(0, x0Signal-15, xmax, yfSignal+40);
```

```
        // Marco:
        setcolor(WHITE);
        setlinestyle(0, 1, 2);
        rectangle(0, x0Signal-15, xmax, yfSignal+40);

        // Líneas de referencia:
        setcolor(LIGHTBLUE);
        setlinestyle(1, 1, 1);
        line(x0Signal, y0Signal, x0Signal, yfSignal+20);
        line(xfSignal, y0Signal, xfSignal, yfSignal+20);
        line(x0Signal-20, y0Signal, xfSignal, y0Signal);
        line(x0Signal-20, yfSignal, xfSignal, yfSignal);
        line(x0Signal-20, yMedioSignal, xfSignal, yMedioSignal);

        // Valores de referencia:
        setcolor(WHITE);
        settextjustify(LEFT_TEXT, CENTER_TEXT);
        outtextxy(10, y0Signal, " 8389631");
        outtextxy(10, yMedioSignal, " 8388607");
        outtextxy(10, yfSignal, " 8387583");

        settextjustify(CENTER_TEXT, TOP_TEXT);
        outtextxy(x0Signal, yfSignal+25, "GPS time");
        outtextxy(xfSignal, yfSignal+25, "+10s");

        BorraVentana(BLUE);
}
```

```
/*****************************************************************************
  Programa principal
*****************************************************************************/
void main(void)
{
        AbreFichLOG();
        InfoPrograma();
        CheckDataDir();
        LeeFichCfg();
        Inicializacion();
        IniciaModoGrafico();
        DibujaPantalla();
        EsperaCambioBanco();
        if (!Flag.Fin)
        {
                LeeParamsCfg();
                CalcNumBytesBanco();
                InfoParametros();
                MuestraParametros();
                BorraFichsTEST();
                gettime(&t);
                HoraAnterior = t.ti_hour;
                getdate(&d);
                NombraNuevosFichs();
                AbreFicheros();
                WrHdrCfgFichs();
                LeeBanco();
                CierraFicheros();
                SacaDatosGPS();
        }
        do
        {
                EsperaCambioBanco();// Mientras espera saca el reloj y comprueba el teclado
                if(!Flag.Fin)
                {
                        if (NuevosFichs()) NombraNuevosFichs();
                        AbreFicheros();
                        LeeParamsCfg();
                        WrHdrCfgFichs();
                        LeeBanco();
                        CierraFicheros();
                        SacaDatosGPS();
                }
        }
        while(!Flag.Fin);
        closegraph();
        fclose(FichLog);
}
```

II.A.1.e. PROGRAMA DE REINICIO DEL SISTEMA

Fichero: P_RESET.C

/* Programa de reinicio de todos los módulos de la antena del Vesubio, según la secuencia descrita en la sección 4.5.2 de la memoria de tesis. El comando de reinicio consiste en un único pulso negativo, de tres segundos de duración, que se envía a través de la línea 'SysReset', correspondiente a la señal /STROBE (pin 1 del puerto paralelo) del PC. */

```
/*************************************************************************
  Macros y ficheros de cabecera
*************************************************************************/
#include <dos.h>
#include <io.h>
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <ctype.h>
#include <string.h>

#define  FALSE          0
#define  TRUE           !FALSE
#define BusDatos        0x378    // Primer puerto paralelo
#define Control1        BusDatos+1
#define Control2        BusDatos+2

/*************************************************************************
  Declaración de funciones
*************************************************************************/
void InConfig_0(void);
void InConfig_1(void);
void SysReset_0(void);
void SysReset_1(void);

/*************************************************************************
  Definición de las funciones
*************************************************************************/
/* Pone la señal 'IN Configuration' a 0 (el nombre de las señales de configuración es el del módulo SerPar, por lo que 'IN Configuration' es una salida para el PC y 'OUT Configuration' una entrada) */
void InConfig_0(void)
{
        outportb(Control2, inportb(Control2) & 0xFB);
}

// Pone la señal 'IN Configuration' a 1
void InConfig_1(void)
{
        outportb(Control2, inportb(Control2) | 0x04);
}
```

```
// Pone la señal SysReset a 0 (está invertida, luego hay que sacar un 1)
void SysReset_0(void)
{
        outportb(Control2, inportb(Control2) | 0x01);
}

// Pone la señal SysReset a 1 (está invertida, luego hay que sacar un 0)
void SysReset_1(void)
{
        outportb(Control2, inportb(Control2) & 0xFE);
}

/*****************************************************************************
```

Programa principal

```
*****************************************************************************/
void main(void)
{
        clrscr();
        printf("\n Programa P_RESET.C.");
        printf("\n\n Manda un comando de reset al módulo SerPar. ");
        printf("\n El comando consiste en un pulso negativo de 3 segundos por el pin 1 del puerto paralelo.");

        SysReset_0();
        delay(3000);
        SysReset_1();
        printf("\n\n Comando de reset enviado. \n");
}
```

II.A.1.f. PROGRAMA DE DEMULTIPLEXADO DE LOS FICHEROS DE DATOS

Fichero:FICHVES2.C

/* Programa para demultiplexar los ficheros de datos de la antena del Vesubio. Realiza el proceso en dos pasos. En el primero lee el fichero introducido desde la línea de comandos y genera tantos ficheros de salida como módulos A/D haya operativos. En el segundo demultiplexa los datos de cada uno de esos ficheros y los escribe en otros tres, correspondientes a los tres canales de cada módulo A/D. Los datos de los ficheros finales tienen formato de enteros largos (cuatro bytes por dato), adecuado para ser visualizado directamente con programas de procesado como DADISP. El cálculo del número de módulos operativos y demás parámetros necesarios se hace a partir de la información de la cabecera de configuración. */

```
/*************************************************************************
  Macros y ficheros de cabecera
*************************************************************************/
#include <dos.h>
#include <io.h>
#include <ctype.h>
#include <stdio.h>
#include <conio.h>
#include <process.h>

#define  FALSE          0
#define  TRUE           !FALSE
#define BusDatos        0x378    // Primer puerto paralelo
#define Control1        BusDatos+1
#define Control2        BusDatos+2
#define BancoA          1 // Banco A: Señal de selección de banco a 1
#define BancoB          0 // Banco B: Señal de selección de banco a 0

/*************************************************************************
  Declaración de variables
*************************************************************************/
FILE *FichEntrada, *FichSalida[4][4];
unsigned char NombreFichEntrada[15];// Nombre completo fichero entrada
unsigned char NomFich[10];// Nombre de los ficheros sin extensión
unsigned char NombreFichSalida[15];// Nombre completo ficheros salida
unsigned char Dato, NumModsAD, Ganancia, Frecuencia, NumTarjsMem;
unsigned int NumSegsBanco;
unsigned long MascaraModsAD;
long offset;
struct
{
        unsigned ModAD[4][4];
        unsigned DemFich[4][4];
        unsigned DemTodosFichs;
        unsigned TeclaBuena;
        unsigned FinFichero;
```

```
        unsigned PrimeraPasada;
}Flag;

union
{
        unsigned char car[4];
        long largo;
} dato;

/*************************************************************************
  Declaración de funciones
*************************************************************************/
void PideNombreFich(void);
void AbreFichEntrada(void);
void LeeParamsCfg(void);
void CheckModsAD(void);
void CalcSegsBanco(void);
void AbreFichsSalida(void);
void EscribeDatos(void);
void SeparaCanales(void);
void CierraFichs(void);

/*************************************************************************
  Definición de funciones
*************************************************************************/
void PideNombreFich(void)
{
        unsigned char i;

        printf("\n Introducir nombre del fichero de entrada: \n");
        scanf("%s", NombreFichEntrada);
        // printf("\n Nombre fichero entrada: %s", NombreFichEntrada);
        i = 0;
        do
        {
                NomFich[i] = NombreFichEntrada[i];
                i++;
        }
        while ((i<8) && (NombreFichEntrada[i] != '.'));
}

void AbreFichEntrada(void)
{
        if ((FichEntrada = fopen(NombreFichEntrada, "rb")) == NULL)
        {
                printf("\n No se puede abrir el fichero de datos %s para lectura", NombreFichEntrada);
                exit(0);
        }
}
```

```
void LeeParamsCfg()
{
	unsigned char i;

	printf("\n Leyendo parámetros de configuración del fichero %s: \n", NombreFichEntrada);

	MascaraModsAD = getc(FichEntrada);
	MascaraModsAD <<= 8;

	Dato = getc(FichEntrada);
	MascaraModsAD |= Dato;
	printf("\n M scara de módulos A/D operativos: 0x%lX", MascaraModsAD);

	Ganancia = getc(FichEntrada);
	if ((Ganancia != 1) && (Ganancia != 2) && (Ganancia != 4) && (Ganancia != 8) &&
	(Ganancia != 16) && (Ganancia != 32) &&(Ganancia != 64) && (Ganancia != 128))
	{
		printf("\n Valor incorrecto de la ganancia.\n");
		sound(1000);
		delay(1000);
		nosound();
		fclose(FichEntrada);
		exit(0);
	}
	else printf("\n Ganancia: %u", Ganancia);

	Dato = getc(FichEntrada);
	Frecuencia = Dato >> 3;
	switch(Frecuencia)
	{
		case 0x04:
			Frecuencia = 200;
			break;
		case 0x02:
			Frecuencia = 100;
			break;
		case 0x01:
			Frecuencia = 50;
			break;
		default:
			printf("\n Valor incorrecto de la frecuencia de muestreo.\n");
			sound(1000);
			delay(1000);
			nosound();
			fclose(FichEntrada);
			exit(0);
	}
	printf("\n Frecuencia de muestreo: %umps", Frecuencia);
	if (Frecuencia == 200)
	{
		MascaraModsAD &= 0xFF00;
		printf("\n Para frecuencia de muestreo 200mps sólo se usan los módulos 0 y 1 de cada línea
		serie.");
		printf("\n Nueva máscara de módulos A/D operativos: 0x%lX", MascaraModsAD);
	}
```

```
	NumTarjsMem = Dato & 0x07;
	if ((NumTarjsMem != 0) && (NumTarjsMem != 1) && (NumTarjsMem != 2) &&
	(NumTarjsMem != 4))
	{
		printf("\n Valor incorrecto del número de tarjetas de memoria usadas.\n");
		sound(1000);
		delay(1000);
		nosound();
		fclose(FichEntrada);
		exit(0);
	}
	else printf("\n Número de tarjetas de memoria usadas: %u\n", NumTarjsMem);
}

// Comprueba qué módulos AD están operativos y activa los flags correspondientes
void CheckModsAD(void)
{
	unsigned char i, j;

	NumModsAD = 0;
	for(i=0; i<4; i++)
	{
		for(j=0; j<4; j++)
		{
			Flag.ModAD[j][i] = FALSE;// Limpia el flag
			if (MascaraModsAD & 0x8000)
			{
				Flag.ModAD[j][i] = TRUE;
				NumModsAD++;
			}
			MascaraModsAD <<= 1;
			//printf("\n Ser %u Mod %u: ", j, i);
			//if (Flag.ModAD[j][i]) printf ("X");
		}
	}
}

void CalcSegsBanco(void)
{
	/* El número de segundos tiene que coincidir con el calculado por el PIC de SerPar, por lo que se
	hace igual que allí (programa SERPAR6.ASM): se calcula para una tarjeta de memoria y 50mps.
	Luego se multiplica (desplazando a la izquierda) y/o divide (desplazando a la derecha) en caso de
	que se use más de una tarjeta de memoria o una frecuencia mayor de 50mps, respectivamente.*/
	NumSegsBanco = 524288/(NumModsAD*9*50+12);
	switch (NumTarjsMem)
	{
		case 1:
			break;
		case 2:
			NumSegsBanco <<= 1;
			break;
		case 4:
			NumSegsBanco <<= 2;
	}
```

```
	switch (Frecuencia)
	{
		case 50:
			break;
		case 100:
			NumSegsBanco >>= 1;
			break;
		case 200:
			NumSegsBanco >>= 2;
	}
	// Si el número de tarjetas de memoria leído es igual a 0 el número de segundos es 10 (modo TEST)
	//independientemente del resto de parámetros:
	if (NumTarjsMem == 0) NumSegsBanco = 10;

	printf("\n Número de segundos por banco de memoria: %u", NumSegsBanco);
}

void AbreFichsSalida(void)
{
	unsigned char i, j;

	if(NumModsAD >12)// Si hay más de 12 módulos A/D operativos se dan dos pasadas
	{
		for(i=0; i<4; i++)
		{
			if (Flag.PrimeraPasada)	// En la primera pasada se abren los ficheros de las dos
				{			//primeras líneas serie
				for(j=0; j<2; j++)
				{
					if (Flag.ModAD[j][i])
					{
					// Nomenclatura: *.XY = Datos del módulo Y de la línea serie X
						sprintf(NombreFichSalida,"%s.%u%u", NomFich, j, i);
						printf("\n %s", NombreFichSalida);
						{
							if (( FichSalida[j][i] =
							fopen( NombreFichSalida, "wb")) == NULL)
							{
								printf("\n No se puede abrir el fichero
								de datos %s para escritura",
								NombreFichSalida);
								CierraFichs();
								exit(0);
							}
						}
					}
				}
			}
			else// En la segunda pasada se abren los ficheros de las dos últimas líneas serie
			{
				for(j=2; j<4; j++)
				{
					if (Flag.ModAD[j][i])
					{
						sprintf(NombreFichSalida,"%s.%u%u", NomFich, j, i);
```

```
                                                printf("\n %s", NombreFichSalida);
                                                {
                                                        if (( FichSalida[j][i] =
                                                        fopen( NombreFichSalida, "wb")) == NULL)
                                                        {
                                                                printf("\n No se puede abrir el fichero
                                                                de datos %s para escritura",
                                                                NombreFichSalida);
                                                                CierraFichs();
                                                                exit(0);
                                                        }
                                                }
                                        }
                                }
                        }
                }
        }

        else// Hay menos de doce módulos A/D operativos (basta con una pasada)
        {
                for(i=0; i<4; i++)
                {
                        for(j=0; j<4; j++)
                        {
                                if (Flag.ModAD[j][i])
                                {
                                        // Nomenclatura: *.XY = Datos del módulo Y de la línea serie X
                                        sprintf(NombreFichSalida,"%s.%u%u", NomFich, j, i);
                                        printf("\n %s", NombreFichSalida);
                                        {
                                                if (( FichSalida[j][i] = fopen( NombreFichSalida,
                                                "wb")) == NULL)
                                                {
                                                        printf("\n No se puede abrir el fichero de datos
                                                        %s para escritura", NombreFichSalida);
                                                        CierraFichs();
                                                        exit(0);
                                                }
                                        }
                                }
                        }
                }
        }
}

void EscribeDatos(void)
{
        unsigned char i, j;  // i, j: líneas serie y módulos A/D. El resto, por orden de aparición
        unsigned int k, m, n;

        fseek(FichEntrada, 0, SEEK_SET);// Se posiciona al inicio del fichero
        do
        {
                if (!feof(FichEntrada))
                {
                        for(i=0; i<4; i++)// Lee y descarta las cabeceras de banco
```

```
                        {
                                if (!feof(FichEntrada))      Dato = getc(FichEntrada);
                        }
                        for(m=0; m<NumSegsBanco; m++)
                        {
                                for(i=0; i<12; i++)// Lee y descarta las cabeceras GPS
                                {
                                        if (!feof(FichEntrada)) Dato = getc(FichEntrada);
                                }

                                for(n=0; n<Frecuencia; n++)// 'Frecuencia' muestras cada segundo
                                {
                                        for(i=0; i<4; i++) // 4 módulos en cada línea serie (lectura en
                                        {                  //serie)
                                                for(k=0; k<9; k++)// 9 bytes en cada muestra
                                                {
                                                        for(j=0; j<4; j++) // 4 líneas serie (lectura en
                                                        {                  //paralelo)
                                                                /* El dato se lee siempre que el módulo
                                                                A/D esté activo, pero sólo se escribe si
                                                                el número de módulos es menor de 12,
                                                                o si es mayor y estamos en la pasada
                                                                que toca (primera pasada para las dos
                                                                primeras líneas serie, segunda para las
                                                                otras dos).*/
                                                                 if (Flag.ModAD[j][i])
                                                                 {
                                                                        if (!feof(FichEntrada))
                                                                        {
                                                                        Dato = getc(FichEntrada);
                                                                        if ((NumModsAD <= 12) ||
                                                                        ((NumModsAD > 12) &&
                                                                        (Flag.PrimeraPasada) &&
                                                                        (j < 2)) ||((NumModsAD >
                                                                        12) &&(Flag.PrimeraPasada)
                                                                        && (j >= 2)))
                                                                                {
                                                                                putc(Dato,
                                                                                FichSalida[j][i]);
                                                                                }
                                                                        }
                                                                 }
                                                        }
                                                }
                                        }
                                }
                        }
                }
        }
        while((!feof(FichEntrada))&&(!kbhit()));
}

void CierraFichs(void)
{
        unsigned char i, j;
```

```
	fclose(FichEntrada);

	for(i=0; i<4; i++)
	{
		for(j=0; j<4; j++)
		{
			fclose(FichSalida[j][i]);
		}
	}
}

void SeparaCanales(void)
{
	unsigned char i, j, k, m, Tecla;

	printf("\n\n Pulsar cualquier tecla para continuar...");
	getch();
	clrscr();

	// 1. Se seleccionan por teclado los ficheros que se demultiplexarán:
	Flag.DemTodosFichs = FALSE;
	for(i=0; i<4; i++)
	{
		for(j=0; j<4; j++)
		{
			if ((Flag.ModAD[j][i]) && (!Flag.DemTodosFichs))
			{
				// Nomenclatura: *.XY = Datos del módulo Y de la línea serie X
				do
				{
					printf("\n ¿Separar trazas del fichero %s.%u%u? [Sí/No/Todos]",
					NomFich, j, i);
					Tecla = toupper(getche());
					Flag.TeclaBuena = TRUE;//Por defecto la tecla pulsada es buena
					switch(Tecla)
					{
						case 'T':
							Flag.DemTodosFichs = TRUE;
							break;
						case 'S':
							Flag.DemFich[j][i] = TRUE;
							break;
						case 'N':
							Flag.DemFich[j][i] = FALSE;
							break;
						default:
							Flag.TeclaBuena = FALSE;
							sound(1000);
							delay(500);
							nosound();
							clrscr();
					}
				}
				while(!Flag.TeclaBuena);
			}
		}
```

```
}

// 2. Si se han seleccionado todos los ficheros con la tecla 'T', se activan los flags de demultiplexado
//individuales:
if(Flag.DemTodosFichs)
{
	for(i=0; i<4; i++)
	{
		for(j=0; j<4; j++)
		{
			if (Flag.ModAD[j][i]) Flag.DemFich[j][i] = TRUE;
		}
	}
}

// 3. Abre los ficheros de entrada seleccionados. Cuando termina de separar las trazas de un fichero
//cierra el mismo y los de salida, para limitar el número de ficheros abiertos a la vez a cuatro:
for(i=0; i<4; i++)
{
	for(j=0; j<4; j++)
	{
		if (Flag.DemFich[j][i])
		{
			sprintf(NombreFichEntrada,"%s.%u%u", NomFich, j, i);
			printf("\n Abriendo fichero de datos: %s", NombreFichEntrada);
			{
				if (( FichEntrada = fopen( NombreFichEntrada, "rb")) ==
				NULL)
				{
					printf("\n No se puede abrir el fichero de datos %s para
					lectura", NombreFichEntrada);
					exit(0);
				}
			}
			for(k=0; k<3; k++)
			{
				sprintf(NombreFichSalida, "%s%u", NombreFichEntrada, k);
				printf("\n Abriendo fichero de salida: %s", NombreFichSalida);
				// Se usan los tres primeros handlers de salida definidos antes:
				if (( FichSalida[0][k] = fopen( NombreFichSalida, "wb")) ==
				NULL)
				{
					printf("\n No se puede abrir el fichero de datos %s para
					escritura", NombreFichSalida);
					CierraFichs();
					exit(0);
				}
			}

				// 4. Demultiplexa los tres canales de cada fichero:
				Flag.FinFichero = FALSE;
				do
				{
					for(k=0; k<3; k++)// 3 canales
					{
```

```
                                        dato.largo = 0;     // Convertimos siempre los datos a
                                                            //enteros largos.
                                        for(m=3; m>0; m--)// 3 bytes por dato
                                        {
                                                if (!feof(FichEntrada)) dato.car[m-1] =
                                                getc(FichEntrada);
                                                else Flag.FinFichero = TRUE;
                                        }
                                        // Restamos la mitad del fondo de escala (2 elevado a
                                        //23) para que esté centrado en 0
                                        offset = 8388610;
                                        dato.largo - = offset;
                                        //Hay que darle la vuelta a los bytes, porque el DADISP
                                        //lee primero los menos significativos:
                                        if (!Flag.FinFichero) for(m=0; m<4; m++)
                                        putc(dato.car[m], FichSalida[0][k]);
                                }
                        }
                        while(!Flag.FinFichero);

                        // 5. Cierra todos los ficheros:
                        CierraFichs();
                }
        }
}
}
```

```
/*****************************************************************************
```

Programa principal

```
*****************************************************************************/
void main(void)
{
        clrscr();
        PideNombreFich();
        AbreFichEntrada();
        LeeParamsCfg();
        CheckModsAD();// Comprueba qué móds. A/D están operativos y activa los flags correspondientes.
        CalcSegsBanco();

        // Si hay doce o menos módulos A/D operativos se puede hacer todo el proceso de una vez. En caso
        //contrario hay que dar dos pasadas al fichero, ya que no se pueden tener tantos ficheros abiertos
        //simultáneamente.
        if(NumModsAD <= 12)
        {
                AbreFichsSalida();
                EscribeDatos();
        }
        else
        {
                Flag.PrimeraPasada = TRUE;
                AbreFichsSalida();
                EscribeDatos();
                CierraFichs();
                Flag.PrimeraPasada = FALSE;
                AbreFichEntrada();
                AbreFichsSalida();
                EscribeDatos();
        }
        CierraFichs();
        SeparaCanales();
}
```

ANEXO II.A.2.- PROGRAMAS DEL MÓDULO SERIE/PARALELO

II.A.2.a. DEFINICIONES PARA EL PROGRAMA DEL MÓDULO SERPAR

Fichero: SERPAR.H

```
;
; Fichero de cabecera para el programa del módulo SerPar. Incluye las equivalencias de
;los nombres asignados a los pines de entrada/salida, más algunas otras definiciones
;necesarias.
;
        NOLIST
;*****************************************************************************
; Asignación de pines
;*****************************************************************************
; CONEXIONES PARA INTERFAZ CON LÍNEAS SERIE DE DATOS:
InPulsDRDY0   EQU   5       ; Pin 5 de PORTB
InData0       EQU   5       ; Pin 5 de PORTB
InData1       EQU   4       ; Pin 4 de PORTB
InData2       EQU   3       ; Pin 3 de PORTB
InData3       EQU   2       ; Pin 2 de PORTB
OutRecTrans   EQU   1       ; Pin 1 de PORTB
OutPulsTrans  EQU   0       ; Pin 0 de PORTA
OutPulsInic0  EQU   5       ; Pin 5 de PORTA
OutPulsInic1  EQU   3       ; Pin 3 de PORTA
OutPulsInic2  EQU   2       ; Pin 2 de PORTA
OutPulsInic3  EQU   1       ; Pin 1 de PORTA
OutPulsSPOK_0 EQU   5       ; Pin 5 de PORTA
;
; CONEXIONES PARA INTERFAZ CON MÓDULO DE MEMORIA:
OutAddCLK     EQU   0       ; Pin 0 de PORTC
OutWE         EQU   1       ; Pin 1 de PORTC
OutCE1        EQU   2       ; Pin 2 de PORTC
OutCE2        EQU   3       ; Pin 3 de PORTC
OutBankSelect EQU   4       ; Pin 4 de PORTC
OutAddReset   EQU   5       ; Pin 5 de PORTC
DataBus       EQU   PORTD
;
; CONEXIONES PARA INTERFAZ CON PC:
InConfig      EQU   6       ; Pin 6 de PORTB
OutConfig     EQU   7       ; Pin 7 de PORTB
InReset       EQU   0       ; Pin 0 de PORTE
;
; OTRAS CONEXIONES:
LED1          EQU   1       ; Pin 1 de PORTE
LED2          EQU   2       ; Pin 2 de PORTE
;
; CARACTERES PERMITIDOS EN LA COMUNICACIÓN SERIE CON EL MÓDULO DE RELOJ:
SPWaitData    EQU   .65     ; Carácter 'SerPar preparado para recibir datos GPS' ('A')
SysReset      EQU   .90     ; Comando de Reset del sistema
;
        LIST
```

II.A.2.b. PROGRAMA DEL MÓDULO SERIE/PARALELO

Fichero: SERPAR10.ASM

```
;
; Programa del PIC16C74 del módulo Serie/Paralelo (SerPar). La definición de los
;registros está en el fichero de cabecera RegsC74.h (anexo I.A.4.a).

        LIST P=16C74A

        INCLUDE <REGSC74.h>
        INCLUDE <RTC.H>

;*****************************************************************************
; Variables
;*****************************************************************************
W_TEMP          EQU     0x20            ; Para salvar W y STATUS en la ISR
STATUS_TEMP     EQU     0x21

PosVar          EQU     0x22            ; Posición de inicio de las variables

ContBits        EQU     PosVar
ContBytes       EQU     PosVar+.1
ContModsAD      EQU     PosVar+.2
ContBitsCfg     EQU     PosVar+.3
LSBContSegBnk   EQU     PosVar+.4
MSBContSegBnk   EQU     PosVar+.5
ContPulsosExtra EQU     PosVar+.6
ContPulsInic    EQU     PosVar+.7
ContMuestras    EQU     PosVar+.8
Dato0           EQU     PosVar+.9
Dato1           EQU     PosVar+.10
Dato2           EQU     PosVar+.11
Dato3           EQU     PosVar+.12
Temp1           EQU     PosVar+.13
Temp2           EQU     PosVar+.14
Temp3           EQU     PosVar+.15
mps             EQU     PosVar+.16
Frec            EQU     PosVar+.17
Ganancia        EQU     PosVar+.18
DatoIn          EQU     PosVar+.19
DatoOut         EQU     PosVar+.20
Status0         EQU     PosVar+.21
Status1         EQU     PosVar+.22
Time0           EQU     PosVar+.23
Time1           EQU     PosVar+.24
Time2           EQU     PosVar+.25
Time3           EQU     PosVar+.26
Time4           EQU     PosVar+.27
Time5           EQU     PosVar+.28
Time6           EQU     PosVar+.29
Time7           EQU     PosVar+.30
Time8           EQU     PosVar+.31
Time9           EQU     PosVar+.32
NumTarjsMem     EQU     PosVar+.33
NumModsAD       EQU     PosVar+.34
LSBSegsBanco    EQU     PosVar+.35
MSBSegsBanco    EQU     PosVar+.36
```

```
LSBLinSer       EQU     PosVar+.37
MSBLinSer       EQU     PosVar+.38
ModsAD0y1       EQU     PosVar+.39      ; Representan el estado de los móds. A/D de todas
ModsAD2y3       EQU     PosVar+.40      ;las líneas serie.
HdrCfg0         EQU     PosVar+.41
HdrCfg1         EQU     PosVar+.42
HdrCfg2         EQU     PosVar+.43
HdrCfg3         EQU     PosVar+.44
Flag            EQU     PosVar+.45      ; Para definir indicadores (flags) generales
Escritura       EQU     0               ; Bit 0 de Flag
FinDatos        EQU     1               ; Bit 1
BancoA          EQU     2               ; Bit 2
IntTmr1         EQU     3               ; Bit 3: se ha producido una int. del Tmr1
PPSRecibido     EQU     4               ; Bit 4: se ha producido una interrupción de PPS
Chk_WrMem       EQU     5               ; Bit 5: hay que comprobar qué datos se escriben en
                                        ;memoria
LED2On          EQU     6               ; Bit 6: el LED 2 está encendido
FlagWrMem       EQU     PosVar+.46      ; Para definir indicadores de escritura en memoria
LS0             EQU     0
LS1             EQU     1
LS2             EQU     2
LS3             EQU     3
;
;*****************************************************************************
; Funciones y macros
;*****************************************************************************
        ORG     0x030
;
;-----------------------------------------------------------------------------------------------------------------------
; 1. FUNCIONES DE INICIALIZACIÓN Y CONFIGURACIÓN:
;-----------------------------------------------------------------------------------------------------------------------
;-----------------------------------------------------------------------------------------------------------------------
; Inicializacion.- Configuración de los pines de I/O e inicialización de variables.
;-----------------------------------------------------------------------------------------------------------------------
Inicializacion
                ; Si no se hace esto los puertos A y E no funcionan:
                ; Configura los puertos A y E como I/O digitales
                BSF     STATUS, RP0     ; Banco 1
                MOVLW           0x07
                MOVWF           ADCON1
                ; Configura el puerto A
                BCF     STATUS, RP0     ; Banco 0
                CLRF    PORTA
                BSF     STATUS, RP0     ; Banco 1
                CLRF    TRISA           ; RA<5:0> salidas
                ; Configura el puerto B
                BCF     STATUS, RP0     ; Banco 0
                CLRF    PORTB
                BSF     STATUS, RP0     ; Banco 1
                MOVLW           0x7D    ; RB1 y RB7 salidas, el resto entradas (01111101 = 0x7D)
                MOVWF           TRISB

                BCF     OPTION_R, RBPU          ; Activa pull-ups
                ; Configura el puerto C
                BCF     STATUS, RP0     ; Banco 0
                CLRF    PORTC
                BSF     STATUS, RP0     ; Banco 1
                MOVLW           0xC0    ; Pines <0:5> salidas; pines 6 y 7, USART (hay que ponerlos
                MOVWF           TRISC   ;a 1, así como el bit 7 de RCSTA (se hace más abajo)).
```

```
; Configura el puerto D como I/O de propósito general (no como puerto paralelo
; esclavo)
BSF     STATUS, RP0    ; Banco 1
BCF     TRISE, PSPMODE         ; …aunque ya está a 0 por defecto
; Configura el puerto D
BCF     STATUS, RP0    ; Banco 0
CLRF    PORTD
BSF     STATUS, RP0    ; Banco 1
CLRF    TRISD          ; Todos los pines como salidas
; Configura el puerto E
BCF     STATUS, RP0    ; Banco 0
CLRF    PORTE
BSF     STATUS, RP0    ; Banco 1
MOVLW           0x01    ; RE0 entrada, el resto (RE1 y RE2) salidas
MOVWF           TRISE

; Limpia los Flags:
BCF     STATUS, RP0    ; Banco 0
CLRF    Flag

; Valores iniciales de los puertos I/O usados en la lectura de datos
BCF     STATUS, RP0    ; Banco 0
BCF     PORTB, OutRecTrans      ; Ser1 de todas las líneas serie en recepción
BCF     PORTA, OutPulsInic0     ; Líneas de pulsos de inicialización a 0 (pulsos
BCF     PORTA, OutPulsInic1     ;positivos)
BCF     PORTA, OutPulsInic2
BCF     PORTA, OutPulsInic3
BCF     PORTA, OutPulsTrans     ; Línea de pulsos de transmisión a 0 (pulsos
                                ;positivos)
; Configuración parámetros USART:
BSF     STATUS, RP0    ; Banco 1
CLRF    SPBRG          ; Máxima velocidad posible (156.3kbps con cristal de 10MHz
                       ;en modo de baja velocidad).
MOVLW           0x20    ; Registro de transmisión: transmisión habilitada, 8 bits
MOVWF           TXSTA ;de datos en transmisión, modo asíncrono, baja velocidad.
BCF     STATUS, RP0    ; Banco 0
MOVLW           0x90    ; Registro de recepción: puerto serie habilitado, 8 bits
MOVWF           RCSTA ;de datos en recepción, recepción habilitada.

; Inicializa las variables para la lectura de datos
CLRF    Dato0
CLRF    Dato1
CLRF    Dato2
CLRF    Dato3
CLRF    FlagWrMem
MOVLW           .7              ; Se inicializa a 7, y no a 8, porque es un contador de
MOVWF           ContBits        ;desplazamientos, no de lecturas.
MOVLW           .9              ; 9 bytes por dato
MOVWF           ContBytes
MOVLW           .4              ; En principio, 4 módulos A/D por línea serie. Si
MOVWF           ContModsAD      ;cuando se lean los parámetros la frecuencia de
                                ;muestreo es 200mps, se cambiará a 2 (función
                                ;'LeeConfig').
; Valores iniciales de los puertos I/O usados en la escritura en memoria
BCF     PORTC, OutBankSelect    ; Selecciona banco A de memoria (0=A, 1=B)
BSF     PORTC, OutCE2           ; Señal CE2 a 1 (va a estar siempre a 1)
BSF     PORTC, OutWE            ; Señal WE a 1 (controla la escritura, activa en bajo)
BSF     PORTC, OutAddCLK        ; Señal de incremento de direcciones a 1 (los pulsos
                                ;son negativos)
```

```
                ; Inicializa las variables para la escritura de datos
                BSF     PORTC, OutAddReset      ; Principio de pulso de reset de direcciones
                BSF     Flag, BancoA    ; Pone a TRUE el flag de banco A
                NOP                     ; El contador de segundos por banco no puede inicializarse
                NOP                     ;hasta que no se reciba la configuración, ya que hay que
                NOP                     ;calcular el valor de inicialización a partir del número de
                NOP                     ;módulos A/D, número de tarjetas de memoria, frecuencia de
                NOP                     ;muestreo y ganancia. Esto se hace en la función
                NOP                     ;'CalcSegsBanco'.
                NOP
                NOP
                BCF     PORTC, OutAddReset      ; Fin del pulso de reset de direcciones

                ; Valores iniciales de los puertos I/O usados en la lectura de la configuración y LEDs de
                ;prueba:
                BCF     PORTB, OutConfig
                BCF     PORTE, LED1
                BCF     PORTE, LED2

                ; Configuración de las interrupciones (habilita la int. externa INT/RB0 y la de
                ;desbordamiento del Tmr1):
                BCF     STATUS,RP0      ; Banco 0
                CLRF    INTCON          ; Pone a 0 todos los flags de interrupción y deshabilita
                                        ;todas las interrupciones
                CLRF    PIR1
                CLRF    PIR2
                BSF     STATUS, RP0     ; Banco 1
                CLRF    PIE1
                CLRF    PIE2
                                        ; Interrupción externa (PPS):
                BSF     STATUS, RP0     ; Banco 1
                BCF     OPTION_R, INTEDG        ; Selecciona flanco descendente para la interrupción
                                                ;externa
                BCF     STATUS, RP0     ; Banco 0
                BSF     INTCON, INTE    ; Habilita interrupción externa

                                        ; Interrupción del Tmr1:
                BSF     STATUS, RP0     ; Banco 1
                BSF     PIE1, TMR1IE    ; Habilita interrupción de timer 1
                BCF     STATUS, RP0     ; Banco 0
                BSF     INTCON, PEIE    ; Habilita interrupciones de periféricos

                BSF     INTCON, GIE     ; Habilita todas las interrupciones no enmascaradas

                RETURN
;
;-----------------------------------------------------------------------------------------------------------------
; Retardo1ms.- Genera un retardo de aproximadamente 1ms (con cristal de 10MHz).
; Número de ciclos = 1ms/(0.4us/ciclo) = 2500ciclos. Con 10 ciclos por bucle:
; Número de bucles = 2500ciclos/(10ciclos/bucle) = 250bucles
;-----------------------------------------------------------------------------------------------------------------
Retardo1ms
                MOVLW           .250
                MOVWF           Temp1
BucleRet
                NOP
                NOP
                NOP
                NOP
                NOP
```

```
                NOP
                NOP
                DECFSZ          Temp1, F
                GOTO    BucleRet
                RETURN
;-------------------------------------------------------------------------------------------------------------------
; Retardo250ms.- Genera un retardo de aproximadamente 250ms (con cristal de 10MHz).
; Número de ciclos = 1ms/(0.4us/ciclo) = 2500ciclos. Con 10 ciclos por bucle:
; Número de bucles = 2500ciclos/(10ciclos/bucle) = 250bucles
; ¡OJO! No usar Temp1 como contador, lo está usando Retardo1ms
;-------------------------------------------------------------------------------------------------------------------
Retardo250ms
                MOVLW           .250
                MOVWF           Temp2
Mas_ms
                CALL    Retardo1ms
                DECFSZ          Temp2, F
                GOTO    Mas_ms
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; RetSegs.- Macro que genera un retardo de 'NumSegs' segundos, contando PPSs. Se consigue un retardo
;aproximado, ya que depende de cuándo llega el primer pulso de segundo.
;-------------------------------------------------------------------------------------------------------------------
RetSegs         MACRO           NumSegs
                LOCAL WaitPPS
                MOVLW           NumSegs
                MOVWF           Temp3           ; ¡OJO! No usar Temp2 ni Temp1, las está usando
                                                ; Retardo250ms
                BCF     Flag, PPSRecibido
WaitPPS
                BTFSS   Flag, PPSRecibido
                GOTO    WaitPPS
                BCF     Flag, PPSRecibido
                CALL    Retardo250ms
                BCF     PORTE, LED1
                DECFSZ          Temp3, F
                GOTO    WaitPPS
                ENDM
;
;-------------------------------------------------------------------------------------------------------------------
; RetPulso.- Genera un retardo de 9 ciclos, incluidas las instrucciones CALL y RETURN. Se han tomado
;9 ciclos para obtener pulsos de 4us (10 ciclos), contando un ciclo correspondiente a la instrucción BSF (o
;BCF).
;-------------------------------------------------------------------------------------------------------------------
RetPulso
                NOP
                NOP
                NOP
                NOP
                NOP
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; PulsoSinc.- Manda un pulso de sincronismo para la comunicación con el PC. Los pulsos son positivos
;con una duración de 500ms(bajo)+500ms(alto)+500ms(bajo).
;-------------------------------------------------------------------------------------------------------------------
PulsoSinc
                CALL    Retardo250ms    ; Retardo de 0.5s, para estar seguros de que el PC está
                CALL    Retardo250ms    ;esperando.
```

```
                BSF     PORTB, OutConfig        ; Manda un pulso positivo de 1ms
                CALL    Retardo250ms
                CALL    Retardo250ms
                BCF     PORTB, OutConfig
                CALL    Retardo250ms    ; Otro retardo de 2ms, para estar seguros de que al PC le
                CALL    Retardo250ms    ;ha dado tiempo a mandar el bit de configuración.
                RETURN
;
;-----------------------------------------------------------------------------------------------------------------------
; LeeConfig.- Lee la configuración del PC. Inicialmente espera hasta que recibe el primer pulso del PC
;por la línea 'In Configuration'. Cuando detecta el primer pulso manda pulsos de petición, a los que el PC
;responde a razón de un bit por pulso.
; Devuelve en W: 0 si no ha habido error, 1 si hay error en la frecuencia de muestreo, 2 si hay error en la
;ganancia.
;-----------------------------------------------------------------------------------------------------------------------
LeeConfig
                ; 1. Espera primer pulso del PC, anunciando que va a enviar la configuración:
                BTFSS   PORTB, InConfig         ; Espera a que la línea esté a 1
                GOTO    LeeConfig
Espera0
                BTFSC   PORTB, InConfig         ; Espera a que la línea esté a 0
                GOTO    Espera0

                ; 2. Después de 250ms manda un primer pulso de sincronismo de 1s, para que el PC
                ;sepa que está a la espera.
                CALL    Retardo250ms
                BSF     PORTB, OutConfig; Manda un pulso positivo de 1s
                CALL    Retardo250ms
                CALL    Retardo250ms
                CALL    Retardo250ms
                CALL    Retardo250ms
                BCF     PORTB, OutConfig
                CALL    Retardo250ms    ; Retardo para que la separación entre todos los pulsos sea
                CALL    Retardo250ms    ;de 1s (estos dos más los dos iniciales de 'PulsoSinc')

                ; 3. Inicializa variables:
                CLRF    Frec
                CLRF    Ganancia
                CLRF    ModsAD0y1
                CLRF    ModsAD2y3
                CLRF    NumTarjsMem

                ; 4. Manda pulsos positivos de sincronismo:
                MOVLW           .3              ; Inicializa el contador de bits para leer los 3 bits de
                MOVWF           ContBitsCfg     ;frecuencia de muestreo.
MasBitsFrec
                CALL    PulsoSinc

                ; 5. Lee los bits de frecuencia de muestreo y los mete en 'Frec'
                BCF     STATUS, C       ; Por defecto, bit C a 0
                BTFSC   PORTB, InConfig         ; Se puede hacer así porque esta orden no afecta al
                                                ; bit C
                BSF     STATUS, C               ; Si el bit leído es 1 pone el bit C a 1
                RLF     Frec, F                 ; Desplaza 'Frec' a la izquierda metiendo el bit C en
                DECFSZ          ContBitsCfg, F  ;el LSb
                GOTO    MasBitsFrec

                ; 6. Manda pulsos positivos de sincronismo:
                MOVLW           .8              ; Inicializa el contador de bits para leer los 8 bits de
                MOVWF           ContBitsCfg     ;ganancia.
```

```
MasBitsGain
                CALL    PulsoSinc

                ; 7. Lee los bits de ganancia y los mete en 'Ganancia'
                BCF     STATUS, C       ; Por defecto, bit C a 0
                BTFSC   PORTB, InConfig         ; Se puede hacer así porque esta orden no afecta al
                                                ; bit C
                BSF     STATUS, C               ; Si el bit leído es 1 pone el bit C a 1
                RLF     Ganancia, F             ; Desplaza 'Ganancia' a la izquierda metiendo el bit
                DECFSZ          ContBitsCfg, F  ;C en el LSb
                GOTO    MasBitsGain

                ; 8. Manda pulsos positivos de sincronismo:
                MOVLW           .8              ; Inicializa contador de bits para leer los 8 primeros
                MOVWF           ContBitsCfg     ;bits de distribución de estaciones en las líneas serie.
MasBitsAD0y1
                CALL    PulsoSinc

                ; 9. Lee los ocho primeros bits de distribución de módulos A/D en las líneas serie
                ;y los mete en 'ModsAD0y1':
                BCF     STATUS, C       ; Por defecto, bit C a 0
                BTFSC   PORTB, InConfig         ; Se puede hacer así porque esta orden no afecta al
                                                ; bit C
                BSF     STATUS, C               ; Si el bit leído es 1 pone el bit C a 1
                RLF     ModsAD0y1, F            ; Desplaza 'ModsAD0y1' a la izquierda metiendo el
                DECFSZ          ContBitsCfg, F  ; bit C en el LSb
                GOTO    MasBitsAD0y1

                ; 10. Manda pulsos positivos de sincronismo:
                MOVLW           .8              ; Inicializa el cont. de bits para leer los 8 primeros
                MOVWF           ContBitsCfg     ;bits de distribución de estaciones en las líneas serie.
MasBitsAD2y3
                CALL    PulsoSinc

                ; 11. Lee los ocho últimos bits de distribución de módulos A/D en las líneas serie
                ;y los mete en 'ModsAD2y3':
                BCF     STATUS, C       ; Por defecto, bit C a 0
                BTFSC   PORTB, InConfig         ; Se puede hacer así porque esta orden no afecta al
                                                ; bit C
                BSF     STATUS, C               ; Si el bit leído es 1 pone el bit C a 1
                RLF     ModsAD2y3, F            ; Desplaza 'ModsAD2y3' a la izquierda metiendo el
                DECFSZ          ContBitsCfg, F  ; bit C en el LSb
                GOTO    MasBitsAD2y3

                ; 12. Manda pulsos positivos de sincronismo:
                MOVLW           .3              ; Inicializa el contador de bits para leer los tres bits
                MOVWF           ContBitsCfg     ;correspondientes al número de tarjetas de memoria.
MasBitsTarjsMem
                CALL    PulsoSinc

                ; 13. Lee los tres bits correspondientes al número de tarjetas de memoria y los
                ;mete en 'NumTarjsMem':
                BCF     STATUS, C       ; Por defecto, bit C a 0
                BTFSC   PORTB, InConfig         ; Se puede hacer así porque esta orden no afecta al
                                                ; bit C
                BSF     STATUS, C               ; Si el bit leído es 1 pone el bit C a 1
                RLF     NumTarjsMem, F          ; Desplaza 'NumTarjsMem' a la izquierda metiendo
                DECFSZ          ContBitsCfg, F  ; el bit C en el LSb
                GOTO    MasBitsTarjsMem
```

```
; 14. Sale comprobando si hay algún error. Además asigna a la variable 'mps' el número
;de muestras por segundo. Para otros cálculos se usará este valor o el de la variable
;'Frec', según cual resulte más cómodo en cada caso.
MOVLW   .50             ; Frec = 0x01 => mps = 50
MOVWF   mps
MOVLW   0x01
SUBWF   Frec, W
BTFSC   STATUS, Z
GOTO    CompruebaGain
MOVLW   .100            ; Frec = 0x02 => mps = 100
MOVWF   mps
MOVLW   0x02
SUBWF   Frec, W
BTFSC   STATUS, Z
GOTO    CompruebaGain
MOVLW   .200            ; Frec = 0x04 => mps = 200
MOVWF   mps
MOVLW   .2              ; Si la frecuencia es de 200mps, sólo se podrán
MOVWF   ContModsAD      ;conectar dos módulos A/D por línea serie.
MOVLW   0x04
SUBWF   Frec, W
BTFSC   STATUS, Z
GOTO    CompruebaGain
CLRF    mps             ; Si hay error, mps queda a 0 y ContModsAD a 4
MOVLW   .4              ; (valor por defecto)
MOVWF   ContModsAD
RETLW   .1              ; Error en la frecuencia de muestreo

CompruebaGain
MOVLW   0x01
SUBWF   Ganancia, W
BTFSC   STATUS, Z
RETLW   .0              ; No hay error
MOVLW   0x02
SUBWF   Ganancia, W
BTFSC   STATUS, Z
RETLW   .0              ; No hay error
MOVLW   0x04
SUBWF   Ganancia, W
BTFSC   STATUS, Z
RETLW   .0              ; No hay error
MOVLW   0x08
SUBWF   Ganancia, W
BTFSC   STATUS, Z
RETLW   .0              ; No hay error
MOVLW   0x10
SUBWF   Ganancia, W
BTFSC   STATUS, Z
RETLW   .0              ; No hay error
MOVLW   0x20
SUBWF   Ganancia, W
BTFSC   STATUS, Z
RETLW   .0              ; No hay error
MOVLW   0x40
SUBWF   Ganancia, W
BTFSC   STATUS, Z
RETLW   .0              ; No hay error
MOVLW   0x80
SUBWF   Ganancia, W
BTFSC   STATUS, Z
```

```
                RETLW.2             ; Error en ganancia

                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; MandaConfig.- Manda los parámetros de configuración a los módulos de adquisición. Los parámetros
;se han leído del PC y están almacenados en las variables 'Frec' y 'Ganancia' con la siguiente
;codificación:
;       PARÁMETRO   VARIABLE    POSIBLES VALORES (CORRESPONDENCIA)
;       Frec. muestreo  Frec    0x01(50mps), 0x02(100mps), 0x04(200mps)
;       Ganancia    Ganancia    0x01(1), 0x02(2), 0x04(4), 0x08(8), 0x10(16), 0x20(32),
;                               0x40(64), 0x80(128)
; La transmisión de los parámetros a los módulos de adquisición se realiza a través de pulsos por
;Ser0, según el siguiente código:
;       11 pulsos   50mps, ganancia 1
;       12 pulsos   50mps, ganancia 2
;       13 pulsos   50mps, ganancia 4
;       14 pulsos   50mps, ganancia 8
;       15 pulsos   50mps, ganancia 16
;       16 pulsos   50mps, ganancia 32
;       17 pulsos   50mps, ganancia 64
;       18 pulsos   50mps, ganancia 128
;       19 pulsos   100mps, ganancia 1
;       20 pulsos   100mps, ganancia 2
;       21 pulsos   100mps, ganancia 4
;       22 pulsos   100mps, ganancia 8
;       23 pulsos   100mps, ganancia 16
;       24 pulsos   100mps, ganancia 32
;       25 pulsos   100mps, ganancia 64
;       26 pulsos   100mps, ganancia 128
;       27 pulsos   200mps, ganancia 1
;       28 pulsos   200mps, ganancia 2
;       29 pulsos   200mps, ganancia 4
;       30 pulsos   200mps, ganancia 8
;       31 pulsos   200mps, ganancia 16
;       32 pulsos   200mps, ganancia 32
;       33 pulsos   200mps, ganancia 64
;       34 pulsos   200mps, ganancia 128
;-------------------------------------------------------------------------------------------------------------------
MandaConfig
                ; 1. Inicializa variables:
                CLRF    ContPulsInic    ; Resetea el contador de pulsos de inicialización
                MOVF    Frec, W         ; Mueve los parámetros a variables temporales
                MOVWF       Temp1
                MOVF    Ganancia, W
                MOVWF       Temp2

                ; 2. Calcula el número de pulsos que se enviarán a los módulos de adquisición a partir
                ;del valor de las variables 'Frec' y 'Ganancia':
CheckFrec
                RRF     Temp1, F        ; Añade 8 pulsos al contador por cada rotación de 'Frec' que
                BTFSC   STATUS, C       ;deje el bit C a 0. Es decir, añade 8 pulsos si la frecuencia de
                GOTO    CheckGain       ;muestreo es 100mps o 16 pulsos si es 200mps.
                MOVLW       .8              ;Luego salta a la comprobación de la ganancia. Si la
                ADDWF       ContPulsInic, F ;frecuencia de muestreo es 50mps, salta ahí
                                            ; directamente.
                GOTO    CheckFrec
CheckGain
                INCF    ContPulsInic, F ; Añade un pulso por cada desplazamiento de 'Ganancia'.
                RRF     Temp2, F
```

```
                BTFSS  STATUS, C
                GOTO   CheckGain

                ; 3. Suma 10 pulsos al valor calculado, para que la cadena mínima sea de 11:
                MOVLW          .10
                ADDWF          ContPulsInic, F

                ; 4. Envía el número de pulsos (positivos) calculados en el punto anterior:
EnviaPulso
                BSF     PORTB, OutRecTrans     ; Ser1 de todas la líneas serie en transmisión
                BSF     PORTA, OutPulsInic0    ; Manda un pulso positivo de 2ms (1 en alto y 1 en
                BSF     PORTA, OutPulsInic1    ;bajo) por cada una de las líneas serie.
                BSF     PORTA, OutPulsInic2
                BSF     PORTA, OutPulsInic3
                CALL    Retardo1ms
                BCF     PORTA, OutPulsInic0
                BCF     PORTA, OutPulsInic1
                BCF     PORTA, OutPulsInic2
                BCF     PORTA, OutPulsInic3
                CALL    Retardo1ms
                DECFSZ          ContPulsInic, F
                GOTO    EnviaPulso
                RetSegs .2                     ; Retardo largo (entre 1 y 2 segundos) para
                                               ;asegurarse de que los módulos de adquisición han
                                               ;salido de la función de lectura de parámetros.
                BCF     PORTB, OutRecTrans     ; Ser1 de todas la líneas serie en recepción

                RETURN
;
;-----------------------------------------------------------------------------------------------------------------------
; 2. FUNCIONES PARA EL CONTROL DE LA TRANSMISIÓN DE DATOS:
;-----------------------------------------------------------------------------------------------------------------------
;-----------------------------------------------------------------------------------------------------------------------
; ProgramaTmr1.- Programa el timer 1 para que produzca una interrupción después de un periodo de
;tiempo igual al periodo de muestreo menos 1ms. Teniendo en cuenta que el Tmr1 se incrementa con cada
;ciclo de instrucción, las constantes de inicialización deben ser (para cristal de 10MHz):
; 200mps => 4ms => 4ms/(0.4us/ciclo) = 10000ciclos
; Habría que inicializar TMR1H:TMR1L a 65536-10000 = 55536, pero para ser precisos hay que tener en
;cuenta las instrucciones previas a la activación del timer:
; 7 en el prog. ppal (incluyendo el CALL a esta función) + 22 en esta función + 2 de salto a la ISR + 20 en
;la propia ISR = 51 ciclos.
; Por tanto, para 200mps queda => 55536+51 = 55576 = 0xD923
; 100mps => 9ms => 9ms/(0.4ms/ciclo) = 22500ciclos => Hay que inicializar a:
; 65536-(22500-51) = 43087 = 0xA84F
; 50mps => 19ms => 19ms/(0.4ms/ciclo) = 47500ciclos => Hay que inicializar a:
; 65536-(47500-51) = 18087 = 0x46A7
;-----------------------------------------------------------------------------------------------------------------------
ProgramaTmr1
                BCF     Flag, IntTmr1     ; Limpia el flag, por si hay interrupciones pendientes
                BSF     STATUS, RP0       ; Banco 1
                BCF     PIE1, TMR1IE      ; Deshabilita la interrupción de Tmr1
                BCF     STATUS, RP0       ; Banco 0
                BCF     T1CON, TMR1IE             ; Deshabilita el Tmr1
                MOVF    Frec, W           ; Comprueba la frecuencia de muestreo para elegir las
                MOVWF           Temp1     ;constantes
                RRF     Temp1, F
                BTFSS   STATUS, C
                GOTO    mps100
                MOVLW           0x46      ; 50mps (Frec = 0x01)
                MOVWF           TMR1H
```

```
                MOVLW       0xA7
                MOVWF       TMR1L
                NOP                 ; Usamos NOPs para conseguir que se invierta el mismo
                NOP                 ;número de instrucciones en los tres casos
                NOP
                NOP
                GOTO   ActivaTmr1
mps100
                RRF     Temp1, F
                BTFSS   STATUS, C
                GOTO   mps200
                MOVLW       0xA8    ; 100mps (Frec = 0x02)
                MOVWF       TMR1H
                MOVLW       0x4F
                MOVWF       TMR1L
                GOTO   ActivaTmr1
mps200
                MOVLW       0xD9    ; 200mps (Frec = 0x04), si no es ninguna de las anteriores se
                MOVWF       TMR1H ;asume que son 200mps
                MOVLW       0x23
                MOVWF       TMR1L
                NOP
ActivaTmr1
                MOVLW       0x01    ; 0x01 => PreScaler a 1:1, oscilador deshabilitado, Tmr1
                MOVWF       T1CON ;habilitado en modo timer (reloj interno)
                BSF     STATUS, RP0    ; Banco 1
                BSF     PIE1, TMR1IE   ; Habilita la interrupción de Tmr1
                BCF     STATUS, RP0    ; Banco 0
                RETURN
;
;-----------------------------------------------------------------------------------------------------------------------
; ProgTmr1_1ms.- Programa el timer 1 para que produzca una interrupción después de 1ms. Teniendo en
;cuenta que el Tmr1 se incrementa con cada ciclo de instrucción, las constantes de inicialización deben
;ser (para cristal de 10MHz):
; 1ms/(0.4us/ciclo) = 2500ciclos, por lo que habría que inicializar TMR1H:TMR1L a:
; 65536-2500 = 63036
; Para ser precisos hay que tener en cuenta las instrucciones previas a la activación del timer:
; 10 en el prog. ppal (incluyendo el CALL a esta función) + 11 en esta función + 2 de salto a la ISR + 20
;en la propia ISR = 43 ciclos, y habría que inicializar a 63036+43 = 63079 = 0xF667
;-----------------------------------------------------------------------------------------------------------------------
ProgTmr1_1ms
                BCF     Flag, IntTmr1     ; Limpia el flag, por si hay interrupciones pendientes
                BSF     STATUS, RP0    ; Banco 1
                BCF     PIE1, TMR1IE   ; Deshabilita la interrupción de Tmr1
                BCF     STATUS, RP0    ; Banco 0
                BCF     T1CON, TMR1ON          ; Deshabilita el Tmr1
                MOVLW       0xF6
                MOVWF       TMR1H
                MOVLW       0x67
                MOVWF       TMR1L
                MOVLW       0x01    ; 0x01 => PreScaler a 1:1, oscilador deshabilitado, Tmr1
                MOVWF       T1CON ;habilitado en modo timer (reloj interno)
                BSF     STATUS, RP0    ; Banco 1
                BSF     PIE1, TMR1IE   ; Habilita la interrupción de Tmr1
                BCF     STATUS, RP0    ; Banco 0
                RETURN
```

```
;-------------------------------------------------------------------------------------------------------------------------
; 3. FUNCIONES PARA LA COMUNICACIÓN CON  LOS MÓDULOS DE RELOJ Y DE MEMORIA
;-------------------------------------------------------------------------------------------------------------------------
;-------------------------------------------------------------------------------------------------------------------------
; EnviaDatoSer.- Envía  por el puerto serie el byte cargado en la variable 'DatoOut'. Antes de hacerlo se
;asegura de que el registro de transmisión (TXREG) está vacío, consultando el estado del bit TXIF y
;esperando en caso de que no esté activado.
;-------------------------------------------------------------------------------------------------------------------------
EnviaDatoSer
                ; Para evitar que se quede esperando aquí, si se ha cumplido el tiempo programado en el
                ;timer 1 pasa de la transmisión serie y sigue el programa:
                BTFSC  Flag, IntTmr1
                RETURN

                BTFSS  PIR1, TXIF
                GOTO   EnviaDatoSer
                MOVF   DatoOut, W
                MOVWF          TXREG
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------------
; LeeData.- Lee doce bytes (correspondientes a un paquete de status y uno de tiempo) del puerto serie y
;los mete en las variables 'Status0', 'Status1', 'Time0', 'Time1',... 'Time9'.
; Devuelve en W: 0 si no ha habido error, 1 si ha habido error en la transmisión serie. Además, en este
;último caso, pone todas las variables de datos GPS a 0x32 (ASCII = '2') si ha sido un error de
;desbordamiento, a 0x33 (ASCII = '3') si ha sido un error de trama o a 0x34 (ASCII = '4') si ha habido
;time out del Timer1.
;-------------------------------------------------------------------------------------------------------------------------
LeeData
                CALL    LeeDatoSer      ; Con cada byte leído comprueba si ha habido error de trama.
                SUBLW.2                 ;El error de desbordamiento sólo se comprueba en el último
                BTFSC  STATUS, Z        ;byte, ya que el bit de error OERR (RCSTA<1>) permanece a
                GOTO   ErrTrama         ;1 hasta que se resetea el receptor. El error de time out de
                MOVF   DatoIn, W        ;Tmr1 también se comprueba sólo en el último byte.
                MOVWF          Status0

                CALL    LeeDatoSer
                SUBLW.2
                BTFSC  STATUS, Z
                GOTO   ErrTrama
                MOVF   DatoIn, W
                MOVWF          Status1

                CALL    LeeDatoSer
                SUBLW.2
                BTFSC  STATUS, Z
                GOTO   ErrTrama
                MOVF   DatoIn, W
                MOVWF          Time0

                CALL    LeeDatoSer
                SUBLW.2
                BTFSC  STATUS, Z
                GOTO   ErrTrama
                MOVF   DatoIn, W
                MOVWF          Time1

                CALL    LeeDatoSer
                SUBLW.2
                BTFSC  STATUS, Z
```

```
GOTO   ErrTrama
MOVF   DatoIn, W
MOVWF          Time2

CALL   LeeDatoSer
SUBLW.2
BTFSC  STATUS, Z
GOTO   ErrTrama
MOVF   DatoIn, W
MOVWF          Time3

CALL   LeeDatoSer
SUBLW.2
BTFSC  STATUS, Z
GOTO   ErrTrama
MOVF   DatoIn, W
MOVWF          Time4

CALL   LeeDatoSer
SUBLW.2
BTFSC  STATUS, Z
GOTO   ErrTrama
MOVF   DatoIn, W
MOVWF          Time5

CALL   LeeDatoSer
SUBLW.2
BTFSC  STATUS, Z
GOTO   ErrTrama
MOVF   DatoIn, W
MOVWF          Time6

CALL   LeeDatoSer
SUBLW.2
BTFSC  STATUS, Z
GOTO   ErrTrama
MOVF   DatoIn, W
MOVWF          Time7

CALL   LeeDatoSer
SUBLW.2
BTFSC  STATUS, Z
GOTO   ErrTrama
MOVF   DatoIn, W
MOVWF          Time8

CALL   LeeDatoSer       ; En el último byte comprueba también el error de
MOVWF          Temp1    ;desbordamiento y el de time out del Tmr1
SUBLW.1
BTFSC  STATUS, Z
GOTO   ErrDesb
MOVF   Temp1, W
SUBLW.2
BTFSC  STATUS, Z
GOTO   ErrTrama
MOVF   Temp1, W
SUBLW.3
BTFSC  STATUS, Z
GOTO   ErrTmr1
MOVF   DatoIn, W
```

```
                MOVWF   Time9

                RETLW.0
ErrDesb
                MOVF  RCREG, W      ; Cuando hay error de desbordamiento lee dos veces el
                MOVF  RCREG, W      ;registro RCREG para limpiarlo y después limpia el flag de
                BCF   RCSTA, CREN   ;error OERR (RCSTA<1>) reseteando el receptor. Además
                NOP                 ;mete el carácter 0x32 ('2') en todos los datos, para tener
                BSF   RCSTA, CREN   ;conocimiento del error.
                MOVLW   0x32
                GOTO  FinError
ErrTrama
                MOVLW   0x33    ; Si ha habido error de trama, mete el carácter 0x33 ('3')
                GOTO  FinError      ;en todos los datos.
ErrTmr1
                MOVLW   0x34    ; Si ha habido error de time out del timer 1, mete el carácter
FinError                            ;0x34 ('4') en todos los datos.
                MOVWF   Status0
                MOVWF   Status1
                MOVWF   Time0
                MOVWF   Time1
                MOVWF   Time2
                MOVWF   Time3
                MOVWF   Time4
                MOVWF   Time5
                MOVWF   Time6
                MOVWF   Time7
                MOVWF   Time8
                MOVWF   Time9
                RETLW.1
;
;-------------------------------------------------------------------------------------------------------------------------
; LeeDatoSer.- Lee un byte del puerto serie y lo escribe en la variable 'DatoIn'. Antes de leer el puerto
;serie, comprueba si hay algún error. Si lo hay se sale de la función con un código de error en W. Si no lo
;hay, espera hasta que llega algún byte al registro de recepción (RCREG), consultando para ello el estado
;del bit RCIF.
; Devuelve en W:
;  0 si se ha leído algún dato (no ha habido error).
;  1 si hay error de desbordamiento del registro de recepción (bit OERR).
;  2 si hay error de trama: detectado bit de STOP igual a 0 (bit FERR).
; 3 si hay error de time out: no se ha recibido respuesta después del tiempo establecido para la
;comunicación serie con el módulo de reloj.
;-------------------------------------------------------------------------------------------------------------------------
LeeDatoSer
                BTFSC  RCSTA, OERR
                RETLW.1             ; Error de desbordamiento

                BTFSC  RCSTA, FERR
                RETLW.2             ; Error de trama

EsperaDato
                ; Para evitar que se quede esperando aquí, si se ha cumplido el tiempo programado en el
                ;timer 1 deja de atender la transmisión serie y sigue el programa:
                BTFSC  Flag, IntTmr1
                RETLW.3
                BTFSS  PIR1, RCIF
                GOTO   EsperaDato
                MOVF   RCREG, W
                MOVWF   DatoIn
                RETLW.0
```

```
;
;-------------------------------------------------------------------------------------------------------------------
; LimpiaBufRX.- Limpia el búffer de recepción, para estar seguros de que no hay ningún dato pendiente
;de la petición anterior.
; Devuelve en W: 1 si había error de saturación del buffer, 0 en caso contrario.
;-------------------------------------------------------------------------------------------------------------------
LimpiaBufRX
                BTFSS  PIR1, RCIF       ; Lee datos del búffer hasta que el flag se desactiva.
                GOTO   LimpiaOERR
                MOVF   RCREG, W
                GOTO   LimpiaBufRX
LimpiaOERR                              ; Comprueba si hay error de saturación del búffer y, en caso
                BTFSS  RCSTA, OERR      ;afirmativo, lo limpia reseteando el receptor. No lee RCREG
                RETLW.0                 ;porque lo acabamos de limpiar
                BCF    RCSTA, CREN
                NOP
                BSF    RCSTA, CREN
                RETLW.1
;
;-------------------------------------------------------------------------------------------------------------------
; WrHdrCfgMem.- Escribe la cabecera con los parámetros de configuración en memoria. El formato de
;dicha cabecera es el comentado en la función 'DefHdrCfg'.
;-------------------------------------------------------------------------------------------------------------------
WrHdrCfgMem
                MOVF   HdrCfg0, W
                CALL   WrDatoMem
                MOVF   HdrCfg1, W
                CALL   WrDatoMem
                MOVF   HdrCfg2, W
                CALL   WrDatoMem
                MOVF   HdrCfg3, W
                CALL   WrDatoMem
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; WrDataMem.- Escribe los datos 'Status0', 'Status1', 'Time0', 'Time1',... 'Time9' en memoria.
;-------------------------------------------------------------------------------------------------------------------
WrDataMem
                MOVF   Status0, W
                CALL   WrDatoMem
                MOVF   Status1, W
                CALL   WrDatoMem
                MOVF   Time0, W
                CALL   WrDatoMem
                MOVF   Time1, W
                CALL   WrDatoMem
                MOVF   Time2, W
                CALL   WrDatoMem
                MOVF   Time3, W
                CALL   WrDatoMem
                MOVF   Time4, W
                CALL   WrDatoMem
                MOVF   Time5, W
                CALL   WrDatoMem
                MOVF   Time6, W
                CALL   WrDatoMem
                MOVF   Time7, W
                CALL   WrDatoMem
                MOVF   Time8, W
                CALL   WrDatoMem
```

```
                MOVF  Time9, W
                CALL  WrDatoMem
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; WrDatoMem.- Escribe el contenido de W en la memoria y manda un pulso para que se incremente la
;dirección de memoria.
;-------------------------------------------------------------------------------------------------------------------
WrDatoMem       ; CE2 va a estar siempre a 1 y /CE1 se activa por hardware, de modo que la escritura se
                ;controla mediante /WE (ver hojas de características de las memorias)
                BCF     PORTC, OutWE
                MOVWF           DataBus
                BSF     PORTC, OutWE
                BCF     PORTC, OutAddCLK        ; Pulso de incremento de la dirección de memoria
                NOP
                NOP
                BSF     PORTC, OutAddCLK
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; PideDataReloj.- Pide datos al módulo de reloj, espera la respuesta y la escribe en memoria.
;-------------------------------------------------------------------------------------------------------------------
PideDataReloj
                CALL    LimpiaBufRX     ; Antes de pedir los datos limpia el búffer de recepción
                MOVLW           SPWaitData      ; Manda un carácter de petición de datos al módulo
de reloj
                MOVWF           DatoOut
                CALL    EnviaDatoSer
                CALL    LeeData         ; Espera respuesta

                DECFSZ          LSBContSegBnk, F; Comprueba el contador de segundos por banco
                GOTO    Wr_Data                 ;(LSB y MSB) para ver si el banco está lleno. Si el
                MOVF    MSBContSegBnk, F        ;LSB es distinto de cero simplemente se decrementa,
                BTFSC   STATUS, Z               ;si es cero se comprueba el MSB. Si éste es distinto
                GOTO    Cambia_Banco            ;de cero, se decrementa y se pasa a la escritura de
                DECF    MSBContSegBnk, F        ;datos. Si el MSB también es cero, antes de escribir
                GOTO    Wr_Data                 ;los datos se cambia de banco y se inicializa el
Cambia_Banco                                    ;contador.
                CALL    CambiaBanco     ; Cambia de banco y actualiza el contador de datos en banco
                MOVF    LSBSegsBanco, W         ; Inicializa el contador de segundos por banco con el
                MOVWF           LSBContSegBnk;valor calculado en la función 'CalcSegsBanco'.
                MOVF    MSBSegsBanco, W
                MOVWF           MSBContSegBnk
                CALL    WrHdrCfgMem     ; Cada vez que cambia de banco escribe los parámetros de
Wr_Data                                 ;configuración en memoria
                CALL    WrDataMem       ; Escribe los datos en memoria
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; CambiaBanco.- Manda un pulso de reset de direcciones y cambia de banco.
; Devuelve en W: 0 si cambia a banco A, 1 si cambia a banco B.
;-------------------------------------------------------------------------------------------------------------------
CambiaBanco
                BSF     PORTC, OutAddReset      ; Principio de pulso de reset de direcciones
                NOP
                NOP
                BCF     PORTC, OutAddReset      ; Fin del pulso de reset de direcciones
                BTFSS   Flag, BancoA            ; Cambia de banco
                GOTO    CambioABancoA
                BCF     Flag, BancoA            ; Flag de banco A a FALSE
```

```
                BSF     PORTC, OutBankSelect   ; Selecciona banco B
                RETLW.1
CambioABancoA
                BSF     Flag, BancoA           ; Flag de banco A a TRUE
                BCF     PORTC, OutBankSelect   ; Selecciona banco A
                RETLW.0
;
;-------------------------------------------------------------------------------------------------------------------
; DefHdrCfg.- Define los bytes de cabecera de configuración. Esta cabecera se incluirá al inicio de cada
;banco de memoria, para que el PC conozca en todo momento los parámetros de configuración
;operativos. El formato de la cabecera es:
; Byte 0 (variable HdrCfg0): MSB de la máscara de estaciones operativas.
; Byte 1 (variable HdrCfg0): LSB de la máscara de estaciones operativas.
; Byte 2 (variable HdrCfg0): Ganancia.
; Byte 3 (variable HdrCfg0): 00FFFMMM, siendo:
;       FFF: frecuencia de muestreo (0x001 = 50, 0x010 = 100, 0x100 = 200)
;       MMM: número de tarjetas de memoria (0x001 = 1, 0x010 = 2, 0x100 = 4)
;-------------------------------------------------------------------------------------------------------------------
DefHdrCfg
                MOVF        ModsAD0y1, W
                MOVWF       HdrCfg0
                MOVF        ModsAD2y3, W
                MOVWF       HdrCfg1
                MOVF        Ganancia, W
                MOVWF       HdrCfg2
                MOVF        Frec, W
                MOVWF       HdrCfg3
                MOVF        NumTarjsMem, W
                MOVWF       Temp1
                SWAPF       Temp1, F
                RLF     Temp1, F
                BCF     STATUS, C
                RLF     Temp1, F
                RLF     HdrCfg3, F
                RLF     Temp1, F
                RLF     HdrCfg3, F
                RLF     Temp1, F
                RLF     HdrCfg3, F
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; CalcNumModsAD.- Calcula el número de módulos de adquisición operativos a partir de la
;configuración física del array, leída en la secuencia de configuración (variables ModsAD0y1 y
;ModsAD2y3).
;-------------------------------------------------------------------------------------------------------------------
CalcNumModsAD
                CLRF    NumModsAD     ; Limpia el número de módulos de adquisición
                MOVF    ModsAD0y1, W          ; Mueve las máscaras de módulos operativos a
                MOVWF       MSBLinSer         ;variables temporales, para poder desplazarlas sin
                MOVF    ModsAD2y3, W          ;perderlas.
                MOVWF       LSBLinSer
                MOVLW       .16       ; Inicializa el contador de bucles a 16.
                MOVWF       Temp1
                MOVLW       .200      ; Si la frecuencia de muestreo es 200mps, el valor del
                SUBWFmps, W           ;contador de bucles se cambia a 8, ya que sólo se tienen en
                BTFSS   STATUS, Z     ;cuenta los módulos 0 y 1 de cada línea serie.
                GOTO    Sigue
                MOVLW       .8
                MOVWF       Temp1
Sigue
```

```
                BCF     STATUS, C
                RLF     LSBLinSer, F    ; Comprueba el número de estaciones operativas, para lo cual
                RLF     MSBLinSer, F    ;desplaza las variables de las líneas serie e incremente
                BTFSC   STATUS, C       ;NumModsAD cada vez que encuentra un bit a 1.
                INCF    NumModsAD, F
                DECFSZ          Temp1, F
                GOTO    Sigue
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; CalcSegsBanco.- Calcula el número de segundos que caben en cada banco de memoria, en función del
;número de tarjetas de memoria usadas, del número de módulos de adquisición y de la frecuencia de
;muestreo. El cálculo se realiza según la siguiente expresión:
; S = Nmem x 524288 / (12 + SR x 9 x Nad), siendo:
;       Nmem = Número de tarjetas de memoria
;       SR = Frecuencia de muestreo
;       Nad = Número de módulos de adquisición
; Para no tener que asignar valores a todas las combinaciones de estos parámetros se asume que la
;frecuencia de muestreo es 50mps y que usamos una sola tarjeta de memoria. Si esto no es cierto luego se
;divide ese valor por dos o por cuatro (en caso de que la frecuencia sea de 100 o 200 mps
;respectivamente) y/o se multiplica por dos o por cuatro (en caso de que usemos dos o cuatro tarjetas de
;memoria, respectivamente). Eso hace que, de momento, no se puedan usar tres tarjetas de memoria.
;Tanto las multiplicaciones como las divisiones se realizan mediante desplazamientos, haciendo antes las
;multiplicaciones para no perder los bits menos significativos. Con este procedimiento se obtiene un
;redondeo por abajo del número de segundos, por lo que en ningún caso se pierden datos.
; Devuelve en W: 1 si ha habido error en alguno de los parámetros leídos (número de módulos A/D,
número de tarjetas de memoria, frecuencia de muestreo), 0 en caso contrario.
;-------------------------------------------------------------------------------------------------------------------
CalcSegsBanco
                ; Lo primero que se hace es comprobar si el número de tarjetas de memoria es 0, en
                ;cuyo caso se fija el valor del número de segundos por banco a 10 (prueba rápida de los
                ;bancos de memoria) y se sale de la ;función:
                MOVLW           0x00            ; Por defecto se ponen los valores para la prueba
                MOVWF           MSBSegsBanco    ;rápida de bancos de memoria: 10 segundos = 0xA.
                MOVLW           0x0A            ;Si no es el caso se cambiarán después.
                MOVWF           LSBSegsBanco
                MOVF    NumTarjsMem, W
                MOVWF           Temp1
                SUBLW.0                 ; Si el número de tarjetas de memoria es 0, inicializa el
                BTFSC   STATUS, Z       ;contador de segundos con el valor seleccionado (10
                GOTO    InicContSegBnk  ;segundos). En caso contrario sigue con el cálculo del
                                        ;número de segundos por banco.

                MOVF    NumModsAD, W
                MOVWF           Temp1

                DECFSZ          Temp1, F
                GOTO    ModsAD_2
                MOVLW           0x04            ; 1 tarjeta de memoria, 1 módulo A/D, 50mps:
                MOVWF           MSBSegsBanco    ; 1134 segundos = 0x46E
                MOVLW           0x6E
                MOVWF           LSBSegsBanco
                GOTO    ChkTarjsMem
ModsAD_2
                DECFSZ          Temp1, F
                GOTO    ModsAD_3
                MOVLW           0x02            ; 1 tarjeta de memoria, 2 módulos A/D, 50mps:
                MOVWF           MSBSegsBanco    ; 574 segundos = 0x23E
                MOVLW           0x3E
                MOVWF           LSBSegsBanco
```

```
                GOTO   ChkTarjsMem
ModsAD_3
                DECFSZ         Temp1, F
                GOTO   ModsAD_4
                MOVLW          0x01            ; 1 tarjeta de memoria, 3 módulos A/D, 50mps:
                MOVWF          MSBSegsBanco    ; 384 segundos = 0x180
                MOVLW          0x80
                MOVWF          LSBSegsBanco
                GOTO   ChkTarjsMem
ModsAD_4
                DECFSZ         Temp1, F
                GOTO   ModsAD_5
                MOVLW          0x01            ; 1 tarjeta de memoria, 4 módulos A/D, 50mps:
                MOVWF          MSBSegsBanco    ; 289 segundos = 0x121
                MOVLW          0x21
                MOVWF          LSBSegsBanco
                GOTO   ChkTarjsMem
ModsAD_5
                DECFSZ         Temp1, F
                GOTO   ModsAD_6
                MOVLW          0x00            ; 1 tarjeta de memoria, 5 módulos A/D, 50mps:
                MOVWF          MSBSegsBanco    ; 231 segundos = 0xE7
                MOVLW          0xE7
                MOVWF          LSBSegsBanco
                GOTO   ChkTarjsMem
ModsAD_6
                DECFSZ         Temp1, F
                GOTO   ModsAD_7
                MOVLW          0x00            ; 1 tarjeta de memoria, 6 módulos A/D, 50mps:
                MOVWF          MSBSegsBanco    ; 193 segundos = 0xC1
                MOVLW          0xC1
                MOVWF          LSBSegsBanco
                GOTO   ChkTarjsMem
ModsAD_7
                DECFSZ         Temp1, F
                GOTO   ModsAD_8
                MOVLW          0x00            ; 1 tarjeta de memoria, 7 módulos A/D, 50mps:
                MOVWF          MSBSegsBanco    ; 165 segundos = 0xA5
                MOVLW          0xA5
                MOVWF          LSBSegsBanco
                GOTO   ChkTarjsMem
ModsAD_8
                DECFSZ         Temp1, F
                GOTO   ModsAD_9
                MOVLW          0x00            ; 1 tarjeta de memoria, 8 módulos A/D, 50mps:
                MOVWF          MSBSegsBanco    ; 145 segundos = 0x91
                MOVLW          0x91
                MOVWF          LSBSegsBanco
                GOTO   ChkTarjsMem
ModsAD_9
                DECFSZ         Temp1, F
                GOTO   ModsAD_10
                MOVLW          0x00            ; 1 tarjeta de memoria, 9 módulos A/D, 50mps:
                MOVWF          MSBSegsBanco    ; 129 segundos = 0x81
                MOVLW          0x81
                MOVWF          LSBSegsBanco
                GOTO   ChkTarjsMem
ModsAD_10
                DECFSZ         Temp1, F
                GOTO   ModsAD_11
```

```
                MOVLW       0x00            ; 1 tarjeta de memoria, 10 módulos A/D, 50mps:
                MOVWF       MSBSegsBanco    ; 116 segundos = 0x74
                MOVLW       0x74
                MOVWF       LSBSegsBanco
                GOTO  ChkTarjsMem
ModsAD_11
                DECFSZ      Temp1, F
                GOTO  ModsAD_12
                MOVLW       0x00            ; 1 tarjeta de memoria, 11 módulos A/D, 50mps:
                MOVWF       MSBSegsBanco    ; 105 segundos = 0x69
                MOVLW       0x69
                MOVWF       LSBSegsBanco
                GOTO  ChkTarjsMem
ModsAD_12
                DECFSZ      Temp1, F
                GOTO  ModsAD_13
                MOVLW       0x00            ; 1 tarjeta de memoria, 12 módulos A/D, 50mps:
                MOVWF       MSBSegsBanco    ; 96 segundos = 0x60
                MOVLW       0x60
                MOVWF       LSBSegsBanco
                GOTO  ChkTarjsMem
ModsAD_13
                DECFSZ      Temp1, F
                GOTO  ModsAD_14
                MOVLW       0x00            ; 1 tarjeta de memoria, 13 módulos A/D, 50mps:
                MOVWF       MSBSegsBanco    ; 89 segundos = 0x59
                MOVLW       0x59
                MOVWF       LSBSegsBanco
                GOTO  ChkTarjsMem
ModsAD_14
                DECFSZ      Temp1, F
                GOTO  ModsAD_15
                MOVLW       0x00            ; 1 tarjeta de memoria, 14 módulos A/D, 50mps:
                MOVWF       MSBSegsBanco    ; 83 segundos = 0x53
                MOVLW       0x53
                MOVWF       LSBSegsBanco
                GOTO  ChkTarjsMem
ModsAD_15
                DECFSZ      Temp1, F
                GOTO  ModsAD_16
                MOVLW       0x00            ; 1 tarjeta de memoria, 15 módulos A/D, 50mps:
                MOVWF       MSBSegsBanco    ; 77 segundos = 0x4D
                MOVLW       0x4D
                MOVWF       LSBSegsBanco
                GOTO  ChkTarjsMem
ModsAD_16
                DECFSZ      Temp1, F
                RETLW.1                     ; Error en el número de módulos A/D
                MOVLW       0x00            ; 1 tarjeta de memoria, 16 módulos A/D, 50mps:
                MOVWF       MSBSegsBanco    ; 72 segundos = 0x48
                MOVLW       0x48
                MOVWF       LSBSegsBanco
ChkTarjsMem
                MOVF  NumTarjsMem, W
                MOVWF       Temp1
                SUBLW.1             ; Si el número de tarjetas de memoria es 1, no se toca
                BTFSC  STATUS, Z    ;número de segundos y se pasa a comprobar la frecuencia
                GOTO  ChkFrec
                MOVF  Temp1, W
                SUBLW.2
```

```
                BTFSS   STATUS, Z
                GOTO    TarjsMem_4
                BCF     STATUS, C               ; Si el número de tarjetas es 2, multiplica el núm. de
                RLF     LSBSegsBanco, F         ;segundos por 2 (un desplazamiento a la izquierda)
                RLF     MSBSegsBanco, F
                GOTO    ChkFrec
TarjsMem_4
                MOVF    Temp1, W
                SUBLW.4
                BTFSS   STATUS, Z
                RETLW.1                 ; Error en el número de tarjetas de memoria
                BCF     STATUS, C               ; Si el número de tarjetas es 4, multiplica el núm. de
                RLF     LSBSegsBanco, F         ;segundos por 4 (dos desplazamientos a la izquierda)
                RLF     MSBSegsBanco, F
                BCF     STATUS, C
                RLF     LSBSegsBanco, F
                RLF     MSBSegsBanco, F
ChkFrec
                MOVF    mps, W
                MOVWF           Temp1
                SUBLW.50
                BTFSC   STATUS, Z
                GOTO    InicContSegBnk  ; Si está todo bien, se inicializa el contador de segundos por
                                        ;banco de memoria con el valor calculado. Si la frecuencia de
                MOVF    Temp1, W        ;muestreo es distinta de 50, sigue con el cálculo.
                SUBLW.100
                BTFSS   STATUS, Z
                GOTO    Frec200
                BCF     STATUS, C
                RRF     MSBSegsBanco, F         ; Si la frecuencia de muestreo es 100, divide el núm.
                RRF     LSBSegsBanco, F         ;de segundos por dos (un desplazamiento a la
                GOTO    InicContSegBnk          ;derecha. Es posible que se pierda un segundo, si el
Frec200                                         ;LSb era 1, pero no vale la pena hacer cálculos más
                MOVF    Temp1, W                ;complejos.
                SUBLW.200
                BTFSS   STATUS, Z
                RETLW.1
                BCF     STATUS, C
                RRF     MSBSegsBanco, F         ; Si la frecuencia de muestreo es 200, divide el núm.
                RRF     LSBSegsBanco, F         ;segundos por cuatro (dos desplazamientos a la
                BCF     STATUS, C               ;derecha)
                RRF     MSBSegsBanco, F
                RRF     LSBSegsBanco, F
InicContSegBnk
                MOVF    LSBSegsBanco, W         ; Inicializa el contador de segundos por banco con el
                MOVWF           LSBContSegBnk;valor calculado.
                MOVF    MSBSegsBanco, W
                MOVWF           MSBContSegBnk
                INCF    LSBContSegBnk, F        ; La primera vez que se inicializa el contador hay
                BTFSC   STATUS, Z               ;que sumarle uno, ya que el decremento del contador
                INCF    MSBContSegBnk, F        ;en la función 'PideDataReloj' se hace antes de
                                                ;escribir los datos en memoria.
                RETLW.0
```

```
;
;-------------------------------------------------------------------------------------------------------------------
; ChkWrMem.- Desplaza la máscara de distribución de estaciones para ver si los datos que se van a leer
;tienen que grabarse en memoria o no. El resultado se guarda en los flags 'FlagWrMem'.
;-------------------------------------------------------------------------------------------------------------------
ChkWrMem
                BCF     FlagWrMem, LS0          ; Por defecto todos los flags a FALSE (no se
                BCF     FlagWrMem, LS1          ;escribe ningún dato en memoria).
                BCF     FlagWrMem, LS2
                BCF     FlagWrMem, LS3
                BCF     STATUS, C
                RLF     LSBLinSer, F
                RLF     MSBLinSer, F
                BTFSC   STATUS, C
                BSF     FlagWrMem,LS0
                RLF     LSBLinSer, F
                RLF     MSBLinSer, F
                BTFSC   STATUS, C
                BSF     FlagWrMem, LS1
                RLF     LSBLinSer, F
                RLF     MSBLinSer, F
                BTFSC   STATUS, C
                BSF     FlagWrMem, LS2
                RLF     LSBLinSer, F
                RLF     MSBLinSer, F
                BTFSC   STATUS, C
                BSF     FlagWrMem, LS3
                RETURN
;
;*****************************************************************************
; Programa principal e interrupciones
;*****************************************************************************
;-------------------------------------------------------------------------------------------------------------------
; Rutina de servicio de la interrupción
;-------------------------------------------------------------------------------------------------------------------
                ORG     ISR_V           ; Vector de interrupción

                MOVWF           W_TEMP
                SWAPF STATUS, W
                BCF     STATUS, RP0     ; Banco 0
                MOVWF           STATUS_TEMP

                BCF     STATUS, RP0     ; Banco 0
                BTFSS   PIR1, TMR1IF    ; Si es la interrupción del Timer 1, sigue. Si no lo es,
                GOTO    ChkINT          ;comprueba si es la interrupción externa INT/RB0
                BCF     PIR1, TMR1IF    ; Limpia el flag de interrupción
                BSF     Flag, IntTmr1   ; Activa el flag de interrupción de Tmr1
                GOTO    Pop
ChkINT
                BTFSS   INTCON, INTF    ; Si es la interrupción externa, sigue.
                GOTO    Pop
                BCF     INTCON, INTF
                BSF     Flag, PPSRecibido
                BSF     PORTE, LED1
Pop
                SWAPF STATUS_TEMP, W    ; Recupera W y STATUS
                MOVWF           STATUS
                SWAPF W_TEMP, F
                SWAPF W_TEMP, W
                RETFIE
```

```
;
;-------------------------------------------------------------------------------------------------------------------
; Programa principal
;-------------------------------------------------------------------------------------------------------------------
                ORG     RESET_V
                GOTO    Inicio
;
                ORG     0x750
Inicio
;
; Inicializaciones
*************************************************************************************
;
                ; 1. Inicialización de los puertos I/O y variables:
                CALL    Inicializacion

                ; 2. Retardo largo (60s) para evitar que interprete el arranque del PC como el inicio de la
                ;secuencia de configuración:
                RetSegs .60

                ; 3. Configuración de los módulos A/D y de las variables de memoria a partir de la
                ;secuencia de configuración enviada por el PC:
                CALL    LeeConfig
                BCF     PORTB, OutConfig        ; Deja a 0 la línea de configuración
                CALL    MandaConfig
                CALL    DefHdrCfg       ; Define los bytes de cabecera de configuración
                CALL    CalcNumModsAD
                CALL    CalcSegsBanco   ; Calcula el número de segundos que caben en cada banco en
                                        ;función del número de tarjetas de memoria, número de
                                        ;módulos de adquisición y frecuencia de muestreo.

                ; 4. Retardo largo (10s) para dejar tiempo a que los PICs de adquisición lean la
                ;secuencia de pulsos de configuración e inicialicen los CADs, y para que el PIC de los
                ;PLLs mande el pulso de sincronismo:
                RetSegs .10

                ; 5. Escribe en memoria la primera cabecera con los parámetros de configuración:
                CALL    WrHdrCfgMem
;
; Bucle principal
*************************************************************************************
;
                ; 1. Espera un flanco de segundo, para sincronizar el proceso de transmisión de datos:
                BCF     Flag, PPSRecibido       ; El flag se activa en la ISR
EsperaPPS
                BTFSS   Flag, PPSRecibido       ; No se desactiva el Flag de PPSRecibido porque se
                GOTO    EsperaPPS               ;hace luego, cuando se hayan recibido los datos de
                BSF     PORTE, LED1             ;tiempo del módulo de reloj.

                ; 2. Inicializa el contador de muestras (se hace aquí, y no en la ISR, para evitar que
                ;algún pulso espúreo lo haga):
                MOVF    mps, W
                MOVWF           ContMuestras
```

```
                ; 3. Espera 1ms para que los módulos de adquisición acaben de leer el dato. Ese tiempo
                ;se aprovecha para inicializar la máscara de estaciones activas y para comprobar si el
                ;dato que se va a leer debe grabarse en memoria. Si se acaba de detectar un PPS también
                ;se aprovecha ese ms para comunicarse con el módulo de reloj y grabar los datos de
                ;tiempo en memoria.
Espera1ms
                CALL    ProgTmr1_1ms            ; Programa el timer 1 a 1ms
                MOVF    ModsAD0y1, W            ; Aprovecha el tiempo muerto para inicializar la
                MOVWF           MSBLinSer       ;máscara de estaciones activas.
                MOVF    ModsAD2y3, W
                MOVWF           LSBLinSer
                CALL    ChkWrMem
                BTFSS   Flag, PPSRecibido       ; Cuando recibe un PPS comprueba la línea de
                GOTO    WaitIntTmr1A            ;petición de reset del PC. En caso de que esté
                BCF     Flag, PPSRecibido       ;activada (en bajo) manda un carácter de reset al
                BTFSC   PORTE, InReset          ;módulo de reloj y vuelve al inicio del programa,
                GOTO    PideDatosReloj          ;para recibir la nueva secuencia de configuración.
                MOVLW           SysReset        ;En caso de que la línea de reset no esté activada
                MOVWF           DatoOut         ;pide datos al módulo de reloj y los guarda en
                CALL    EnviaDatoSer            ;memoria.
                GOTO    Inicio
PideDatosReloj
                BSF     PORTE, LED2             ; Enciende el LED2
                CALL    PideDataReloj
                BCF     PORTE, LED2             ; Apaga el LED2
                BCF     PORTE, LED1             ; Apaga el LED1
WaitIntTmr1A
                BTFSS   Flag, IntTmr1           ; Espera el final del ms programado en el Tmr1
                GOTO    WaitIntTmr1A
                BCF     Flag, IntTmr1
                BCF     T1CON, TMR1ON           ; Deshabilita el Tmr1

                ; 4. Programa el timer 1 para controlar cuando puede pedir los sucesivos datos. El timer
                ;se programa para que genere una interrupción después del tiempo correspondiente al
                ;periodo de muestreo seleccionado menos 1ms (retardo inicial que ya hemos generado
                ; antes).
                CALL    ProgramaTmr1

                ; 5. Manda el primer pulso de transmisión. Los pulsos deben tener 40 ciclos como
                ;mínimo, para que al programa de adquisición le dé tiempo a hacer todo lo que debe.
Bit0
                BSF     PORTA, OutPulsTrans
                CALL    RetPulso
                CALL    RetPulso
                BCF     PORTA, OutPulsTrans
                CALL    RetPulso

                ; 6. Lee el primer bit del byte de las cuatro líneas serie:
                BCF     Dato0, 0        ; Por defecto todos los bits leídos valen 0
                BCF     Dato1, 0
                BCF     Dato2, 0
                BCF     Dato3, 0
                BTFSC   PORTB, InData0
                BSF     Dato0, 0
                BTFSC   PORTB, InData1
                BSF     Dato1, 0
                BTFSC   PORTB, InData2
                BSF     Dato2, 0
                BTFSC   PORTB, InData3
                BSF     Dato3, 0
```

```
                ; 7. Inicio pulso de transmisión (flanco ascendente):
DesplazaBit
                BSF     PORTA, OutPulsTrans

                ; 8. Desplazamiento de los datos:
                RLF     Dato0, F
                RLF     Dato1, F
                RLF     Dato2, F
                RLF     Dato3, F

                ; 9. Gestión de contadores y flags: en lugar de un solo contador para los 27 bytes se usan
                ;dos, para contar nueve bytes y cuatro módulos A/D (dos si la frec. de muestreo es 200
                ;mps):
                DECFSZ          ContBits, F
                GOTO    FinPulsTrans
                MOVLW           .7
                MOVWF           ContBits
                BSF     Flag, Escritura
                DECFSZ          ContBytes, F
                GOTO    FinPulsTrans
                MOVLW           .9              ; Si el byte recibido corresponde a un nuevo módulo
                MOVWF           ContBytes       ;A/D se rota la máscara para ver si hay que escribir
                BSF     Flag, Chk_WrMem         ;en memoria. Todavía no se pueden chequear los
                                                ;flags de escritura en memoria porque nos
                                                ;quedaríamos sin escribir un byte.
                DECFSZ          ContModsAD, F   ; Si ya se han leído los datos de todos los módulos de
                GOTO    FinPulsTrans            ;cada línea serie, se inicializa el contador y se activa
                BSF     Flag, FinDatos          ;el flag de fin de datos.
                MOVLW           .4              ; Por defecto el contador se inicializa a 4 (valor para
                MOVWF           ContModsAD      ;50 y 100mps). Luego se comprueba la frecuencia de
                MOVLW           .200            ;muestreo y, si es 200mps, el valor del contador se
                SUBWFmps, W                     ;cambia a 2.
                BTFSS   STATUS, Z
                GOTO    FinPulsTrans
                MOVLW           .2
                MOVWF           ContModsAD
FinPulsTrans
                ; 10. Fin del pulso de transmisión:
                CALL    RetPulso
                BCF     PORTA, OutPulsTrans     ; Fin pulso transmisión

                ; 11. Lectura del siguiente bit:
                CALL    RetPulso
                BCF     Dato0, 0        ; Por defecto todos los bits leídos valen 0
                BCF     Dato1, 0
                BCF     Dato2, 0
                BCF     Dato3, 0
                BTFSC   PORTB, InData0
                BSF     Dato0, 0
                BTFSC   PORTB, InData1
                BSF     Dato1, 0
                BTFSC   PORTB, InData2
                BSF     Dato2, 0
                BTFSC   PORTB, InData3
                BSF     Dato3, 0

                ; 12. Si toca, escritura en memoria. Si no, vuelta a lectura de bits.
                BTFSS   Flag, Escritura
                GOTO    DesplazaBit
```

```
                BCF     Flag, Escritura
                BTFSS   FlagWrMem,LS0
                GOTO    ChkLinSer1
                MOVF    Dato0, W
                CALL    WrDatoMem
ChkLinSer1
                BTFSS   FlagWrMem, LS1
                GOTO    ChkLinSer2
                MOVF    Dato1, W
                CALL    WrDatoMem
ChkLinSer2
                BTFSS   FlagWrMem, LS2
                GOTO    ChkLinSer3
                MOVF    Dato2, W
                CALL    WrDatoMem
ChkLinSer3
                BTFSS   FlagWrMem, LS3
                GOTO    FinWrMem
                MOVF    Dato3, W
                CALL    WrDatoMem
FinWrMem
                BTFSC   Flag, Chk_WrMem         ; Después de la escritura en memoria se chequean y
                CALL    ChkWrMem                ;actualizan los flags de escritura en memoria.
                BCF     Flag, Chk_WrMem
                BTFSS   Flag, FinDatos
                GOTO    Bit0
                BCF     Flag, FinDatos

                ; 13. Manda un pulso de fin de transmisión, para que el último módulo de adquisición
                ;conmute Ser1 a recepción:
                BSF     PORTA, OutPulsTrans
                CALL    RetPulso
                CALL    RetPulso
                BCF     PORTA, OutPulsTrans
                CALL    RetPulso

                ; 14. Si era la última muestra del segundo, vuelve a la espera de PPS. Si no, espera la
                ;interrupción del Tmr1 y vuelve a la gestión de la transmisión de datos.
                DECFSZ          ContMuestras, F ; Si era la última muestra del segundo no se espera a
                GOTO    WaitIntTmr1B            ;la int. del timer 1, porque si ésta se produjera
                                                ;sistemáticamente después de la llegada del PPS
                                                ;provocaría una desincronización de todo el proceso.
                BCF     T1CON, TMR1ON           ; Deshabilita el Tmr1
                BCF     Flag, PPSRecibido       ; Antes de volver a la espera de PPS limpia el flag
                BCF     INTCON, INTF            ; Habilita int. externa, limpiando antes el flag
                GOTO    EsperaPPS
WaitIntTmr1B
                BTFSS   Flag, IntTmr1
                GOTO    WaitIntTmr1B
                BCF     Flag, IntTmr1
                BCF     T1CON, TMR1ON           ; Deshabilita el Tmr1
                GOTO    Espera1ms

                END
```

ANEXO II.A.3.- PROGRAMA DE LOS PIC16F84 DE LAS TARJETAS A/D DE LOS MÓDULOS DE ADQUISICIÓN

II.A.3.a. DIRECCIONES DE LOS VECTORES Y REGISTROS DE FUNCIÓN ESPECIAL DEL PIC16F84

Fichero: C84_REG.H

```
; Definición de vectores, registros de función especial, bits y otros para el PIC16F84.
;
        NOLIST
;*****************************************************************************
;Vectores
;*****************************************************************************
RESET_V     EQU     0x0000              ; Address of RESET Vector
ISR_V       EQU     0x0004              ; Address of Interrupt Vector
PMEM_END    EQU     0x03FF              ; Last address in Program Memory
;
;*****************************************************************************
;Registros de función especial
;*****************************************************************************
INDF        EQU     00
TMR0        EQU     01
OPTION_R    EQU     81
PCL         EQU     02
STATUS      EQU     03
FSR         EQU     04
PORTA       EQU     05
Port_A      EQU     05      ; Por compatibilidad con otros programas
TRISA       EQU     85
PORTB       EQU     06
Port_B      EQU     06      ; Por compatibilidad con otros programas
TRISB       EQU     86
EEDATA      EQU     08
EECON1      EQU     88
EEADR       EQU     09
TRISE       EQU     89
PCLATH      EQU     0A
INTCON      EQU     0B
;
;*****************************************************************************
;Definiciones de bits
;*****************************************************************************
; STATUS register (Address 03/83)
;
IRP         EQU     7
RP1         EQU     6
RP0         EQU     5
TO          EQU     4
PD          EQU     3
Z           EQU     2
DC          EQU     1
C           EQU     0
CARRY       EQU     0       ; Por compatibilidad con otros programas
;
```

```
; OPTION register (Address 81)
;
RBPU            EQU     7
INTEDG          EQU     6
T0CS            EQU     5
T0SE            EQU     4
PSA             EQU     3
PS2             EQU     2
PS1             EQU     1
PS0             EQU     0
;
; INTCON register (Address 0B/8B)
;
GIE             EQU     7
EEIE            EQU     6
T0IE            EQU     5
INTE            EQU     4
RBIE            EQU     3
T0IF            EQU     2
INTF            EQU     1
RBIF            EQU     0
;
;*****************************************************************************
; Otras definiciones
;*****************************************************************************
W               EQU     0
F               EQU     1
Temp            EQU     0       ; Por compatibilidad con otros programas
Same            EQU     1
;
FALSE           EQU     0
TRUE            EQU     1


        LIST
```

II.A.3.b. DEFINICIONES PARA EL PROGRAMA DE LAS TARJETAS A/D DE LOS MÓDULOS DE ADQUISICIÓN

Fichero: ADQUISIC.H

```
;
; Fichero de cabecera para el programa de los PIC16F84 de las tarjetas de conversión
;A/D de los módulos de adquisición. Incluye las equivalencias de los nombres asignados
;a los pines de entrada/salida para el control de los conversores A/D y transmisión de
;datos, más algunas otras definiciones necesarias.
;
        NOLIST
;*****************************************************************************
; Asignación de pines
;*****************************************************************************
; ASIGNACIÓN DE PINES PARA CONTROL DEL AD7710 Y PUERTO SERIE:
; Puerto A:
CONTT           EQU     0
DX              EQU     1
CONTR           EQU     2
GAIN2           EQU     2
GAIN1           EQU     3
GAIN0           EQU     4
; Puerto B:
DR              EQU     0
DRDY            EQU     1
RFS             EQU     2
TFS             EQU     3
A0              EQU     3
DAT1            EQU     4
DAT2            EQU     5
DAT3            EQU     6
CLK             EQU     7

WL              EQU     7       ; Longitud datos 16 o 24 bits
;
; ASIGNACIÓN DE PINES PARA LAS LÍNEAS DE DATOS Y OTRAS:
InPulsConf      EQU     0               ; Bit 0 del puerto B (INT)
InPulsSerParOK  EQU     0               ; Bit 0 del puerto B (INT)
InPulsDRDY      EQU     0               ; Bit 0 del puerto B (INT)
INT             EQU     0               ; Bit 0 del puerto B (INT)
OutPulsDRDY     EQU     0               ; Bit 0 del puerto A
OutData         EQU     0               ; Bit 0 del puerto A
OutRecTrans     EQU     1               ; Bit 1 del puerto A
InPulsTrans     EQU     3               ; Bit 3 del puerto A
MSBitID         EQU     2               ; Bit 2 del puerto A
LSBitID         EQU     4               ; Bit 4 del puerto A
;
; CONSTANTES PARA DEFINIR LA VELOCIDAD DEL PUERTO SERIE:
;CLK  3.579
;4800=.58  9600=.26  19200=.11  38400=.3+nop
;
BAUD            EQU     .26
BAUDR           EQU     .13

        LIST
```

II.A.3.c. PROGRAMA DE LAS TARJETAS DE CONVERSIÓN A/D

Fichero: ADQUIS6.ASM

```
;
; Programa del PIC16F84 de las tarjetas de conversión A/D de los módulos de
;adquisición.

        LIST P=16F84

        INCLUDE <C84_reg.h>
        INCLUDE <adquisic.h>

;*****************************************************************************
; Variables
;*****************************************************************************
PosVar          EQU     0x30            ; Posición de inicio de las variables
Count1          EQU     PosVar
DlyCnt          EQU     PosVar+.1
ADC1            EQU     PosVar+.2
ADC2            EQU     PosVar+.3
ADC3            EQU     PosVar+.4
ADC11           EQU     PosVar+.5
ADC12           EQU     PosVar+.6
ADC13           EQU     PosVar+.7
ADC21           EQU     PosVar+.8
ADC22           EQU     PosVar+.9
ADC23           EQU     PosVar+.10
ADC31           EQU     PosVar+.11
ADC32           EQU     PosVar+.12
ADC33           EQU     PosVar+.13
ContBits        EQU     PosVar+.14      ; Contador de bits enviados
NumTarjeta      EQU     PosVar+.15
Ganancia        EQU     PosVar+.16
Frec            EQU     PosVar+.17
Temp1           EQU     PosVar+.18
Temp2           EQU     PosVar+.19
Temp3           EQU     PosVar+.20
ContPulsConf    EQU     PosVar+.21
ContPulsDRDY    EQU     PosVar+.22
ContBloqPulsTr  EQU     PosVar+.23
ContPulsADC     EQU     PosVar+.24
TurnoTransm     EQU     PosVar+.25
Flag            EQU     PosVar+.26
BitAEnviar      EQU     0               ; Valor del siguiente bit que se enviará
SerParOK        EQU     1
Config          EQU     2               ; Estado de espera de la secuencia de configuración
FinTransm       EQU     3               ; Se ha terminado la transmisión de bits de datos
Transmite       EQU     4               ; Tiene el turno de transmitir datos
LastPulsTrans   EQU     5               ; A la espera del último pulso de transmisión.
;
```

```
;*****************************************************************************
; Funciones y macros
;*****************************************************************************
        ORG     0x100
;
;-----------------------------------------------------------------------------------------------------------------------
; LeeConfig.- Lee la secuencia de configuración enviada por el módulo SerPar. La lectura se reduce a una
;cuenta del número de pulsos de la secuencia, que se interpretan según el siguiente código:
;       11 pulso        50mps, ganancia 1
;       12 pulsos       50mps, ganancia 2
;       13 pulsos       50mps, ganancia 4
;       14 pulsos       50mps, ganancia 8
;       15 pulsos       50mps, ganancia 16
;       16 pulsos       50mps, ganancia 32
;       17 pulsos       50mps, ganancia 64
;       18 pulsos       50mps, ganancia 128
;       19 pulsos       100mps, ganancia 1
;       20 pulsos       100mps, ganancia 2
;       21 pulsos       100mps, ganancia 4
;       22 pulsos       100mps, ganancia 8
;       23 pulsos       100mps, ganancia 16
;       24 pulsos       100mps, ganancia 32
;       25 pulsos       100mps, ganancia 64
;       26 pulsos       100mps, ganancia 128
;       27 pulsos       200mps, ganancia 1
;       28 pulsos       200mps, ganancia 2
;       29 pulsos       200mps, ganancia 4
;       30 pulsos       200mps, ganancia 8
;       31 pulsos       200mps, ganancia 16
;       32 pulsos       200mps, ganancia 32
;       33 pulsos       200mps, ganancia 64
;       34 pulsos       200mps, ganancia 128
;
; Los pulsos son positivos con una duración de 2ms (1 en alto y 1 en bajo). Para saber cuándo se ha
;acabado la secuencia se establece un tiempo máximo mediante un bucle, que se lanza al recibir el primer
;pulso de la secuencia.
; Devuelve: la frecuencia de muestreo en 'Frec' y la ganancia en la variable 'Ganancia'. En el caso de la
;frecuencia de muestreo se usan, por sencillez para la programación posterior de los conversores, los
;siguientes códigos:
;       50mps => Frec = 0x01, 100mps => Frec = 0x02, 200mps => Frec = 0x04.
; Devuelve en W: 0 si todo ha ido bien, 1 si ha habido error (número incorrecto de pulsos).
;-----------------------------------------------------------------------------------------------------------------------
LeeConfig
                ; 1. Inicializa variables:
                BCF     Flag, SerParOK
                BSF     Flag, Config        ; Activa el flag de configuración para que la ISR lo sepa
                CLRF    ContPulsConf        ; Limpia el contador de pulsos de configuración
                CLRF    Temp1               ; Como contadores del bucle de espera usamos variables
                CLRF    Temp2               ;temporales

                ; 2. Espera el primer pulso comprobando el Flag de SerParOK. Este Flag se activa
                ;cuando se recibe un pulso por la línea de interrupción externa (en la rutina de
                ;servicio de la interrupción), y se desactiva al final del bucle de espera de la secuencia
                ;de configuración (punto 3 de esta misma función) o al enviar un pulso de DRDY.
EsperaConfig
                BTFSS   Flag, SerParOK
                GOTO    EsperaConfig
```

```
                ; 3. Bucle para controlar el tiempo máximo de espera. El tiempo de espera es:
                ;       Nº ciclos = NºBuclesTemp2 x (3xNºBuclesTemp1+3) = 197376ciclos (con
                ;       NºBuclesTemp1 = NºBuclesTemp2 = 256).
                ; Con cristal de 10MHz TBucle = 197376 x 0.4us/ciclo = 78950.4us, suficiente para
                ;detectar el número máximo de pulsos (34) de 2ms (1ms alto+1ms bajo).
BucleConfig
                DECFSZ      Temp1, F        ; Bucle de espera
                GOTO   BucleConfig
                DECFSZ      Temp2, F
                GOTO   BucleConfig

                BCF     Flag, Config    ; Deja los flags de configuración y de pulsos de SerPar
                BCF     Flag, SerParOK  ;desactivados.

                ; 4. Comprobación inicial del número de pulsos recibidos. Si este número es menor de
                ;once, sale devolviendo en W el código de error.
                MOVF   ContPulsConf, W      ; Mueve el contador de pulsos a una variable
                MOVWF       Temp1           ;temporal para poder desplazarla sin perderla.
                MOVLW       .10             ; En Temp1 se queda el número de pulsos recibidos
                SUBWFTemp1, F               ;menos diez
                BTFSS  STATUS, C
                RETLW.1                     ; Error en el número de pulsos
                BTFSC  STATUS, Z
                RETLW.1                     ; Error en el número de pulsos

                ; 5. Calcula la frecuencia de muestreo a partir del número de pulsos recibido:
                CLRF    Frec            ; Inicializa variables
                CLRF    Ganancia
                MOVLW       .8          ; Para calcular la frecuencia vemos el número de veces que se
                SUBWFTemp1, W           ;puede restar 8 al número de pulsos recibido manteniéndose
                BTFSC  STATUS, C        ;el resultado mayor o igual que cero. Al ser mayor o igual (y
                BTFSC  STATUS, Z        ;no estrictamente igual) tenemos que consultar el bit Z
                GOTO   mps50            ;además del C (si se invierte el orden de los términos, da
                MOVWF       Temp1       ;problemas).
                MOVLW       .8
                SUBWF       Temp1, W
                BTFSC       STATUS, C
                BTFSC       STATUS, Z
                GOTO        mps100
                MOVWF       Temp1
                MOVLW       .8
                SUBWF       Temp1, W
                BTFSC       STATUS, C
                BTFSC       STATUS, Z
                GOTO        mps200
                RETLW       .1              ; Error de configuración.
mps200
                MOVLW       0x04            ; 200mps => Frec = 00000100b. Usamos estos
                MOVWF       Frec            ;códigos por sencillez a la hora de programar los
                GOTO   CheckGain            ;conversores.
mps100
                MOVLW       0x02            ; 100mps => Frec = 00000010b
                MOVWF       Frec
                GOTO   CheckGain
mps50
                MOVLW       0x01            ; 50mps => Frec = 00000001b
                MOVWF       Frec
                GOTO   CheckGain
ErrorFrec
                RETLW.1
```

```
                ; 6. Calcula la ganancia a partir del número de pulsos recibido:
CheckGain
                DECFSZ      Temp1, F    ; Una vez restado el número de pulsos
                GOTO        Gain2       ;correspondiente a la frecuencia de muestreo (0, 8 o
                MOVLW       .1          ;16), en Temp1 queda el número correspondiente a
                GOTO   WrGain           ;la ganancia: 1 para ganancia 1, 2 para ganancia 2, 3
Gain2                                   ;para ganancia 4,... 8 para ganancia 128. Por tanto
                DECFSZ      Temp1, F    ;ahora simplemente hay que decrementar Temp1
                GOTO   Gain4            ;hasta llegar a 0.
                MOVLW       .2
                GOTO   WrGain
Gain4
                DECFSZ      Temp1, F
                GOTO   Gain8
                MOVLW       .4
                GOTO   WrGain
Gain8
                DECFSZ      Temp1, F
                GOTO   Gain16
                MOVLW       .8
                GOTO   WrGain
Gain16
                DECFSZ      Temp1, F
                GOTO   Gain32
                MOVLW       .16
                GOTO   WrGain
Gain32
                DECFSZ      Temp1, F
                GOTO   Gain64
                MOVLW       .32
                GOTO   WrGain
Gain64
                DECFSZ      Temp1, F
                GOTO   Gain128
                MOVLW       .64
                GOTO   WrGain
Gain128
                DECFSZ      Temp1, F
                RETLW.1                 ; Si todavía quedan pulsos, error de configuración.
                MOVLW       .128
WrGain
                MOVWF       Ganancia
                RETLW.0
```

```
;
;-----------------------------------------------------------------------------------------------------------------------
; PalCtrol.- Calcula los bytes de control con los que se programarán los conversores, a partir del valor de
;las variables 'Frec' y 'Ganancia'. La palabra de control está formada por 24bits con el siguiente formato
;(ver hojas de características del AD7710 para más detalles):
;MD2 MD1 MD0 G2 G1 G0 CH PD WL IO BO B/U FS11 FS10 FS9 FS8 FS7 FS6 FS5 FS4 FS3 FS2
;FS1 FS0
; En nuestro caso se usarán tres variables (ADC1, ADC2 y ADC3) para almacenar la información.
; Devuelve:     Bits MD2 MD1... PD  en la variable ADC1
;               Bits WL  IO ... FS8 en la variable ADC2
;               Bits FS7 FS6... FS0 en la variable ADC3
; Devuelve en W: 0 si todo ha ido bien, 1 si ha habido error en la frecuencia leída.
;-----------------------------------------------------------------------------------------------------------------------
PalCtrol
                ; 1. Inicialización de variables:
                CLRF    ADC1
                CLRF    ADC2
                CLRF    ADC3

                ; 2. Escritura de ADC1 en función de la ganancia:
                MOVF    Ganancia, W     ; Movemos 'Ganancia' a una variable temporal para poderla
                MOVWF       Temp1       ;desplazar sin perderla.
MasGain
                RRF     Temp1, F        ; Formato de la variable 'Ganancia': 0x01 (ganancia = 1),
                BTFSC   STATUS, C       ;0x02 (ganancia = 2), 0x80 (ganancia = 128).
                GOTO    FinGain         ; Para convertirlo a los bits G2 G1 G0 incrementamos éstos
                INCF    ADC1, F         ;tantas veces como desplacemos 'Ganancia'.
                GOTO    MasGain
FinGain
                BCF     STATUS, C       ; Desplazamos a la izquierda dos veces ADC1 introduciendo
                RLF     ADC1, F         ;ceros. Luego ponemos el bit 5 a 1, con lo que ADC1 queda
                RLF     ADC1, F         ;001GGG00b (modo autocalibración, ganancia configurable,
                BSF     ADC1, 5         ;canal 1, operación normal).

                ; 3. Escritura de ADC2 y ADC3 en función de la frecuencia de muestreo:
                RRF     Frec, F
                BTFSC   STATUS, C
                GOTO    Frec50mps
                RRF     Frec, F
                BTFSC   STATUS, C
                GOTO    Frec100mps
                RRF     Frec, F
                BTFSC   STATUS, C
                GOTO    Frec200mps
                RETLW.1                 ; Error en la frecuencia de muestreo.
Frec50mps
                MOVLW       0x81        ; El primer nibble de ADC2 siempre es 0x8 (selecciona 24
                MOVWF       ADC2        ;bits de datos, corrientes de compensación y burn-out off
                MOVLW       0x80        ;y entrada bipolar). El resto de los bits de ADC2 y ADC3
                MOVWF       ADC3        ;se calculan con la expresión:
                RETLW.0                 ;       code = f_CLKIN/(512 x frec.muestreo),
Frec100mps                              ;       con f_CLKIN = 9.8304MHz
                MOVLW       0x80
                MOVWF       ADC2
                MOVLW       0xC0
                MOVWF       ADC3
                RETLW.0
Frec200mps
                MOVLW       0x80
                MOVWF       ADC2
```

```
                MOVLW       0x60
                MOVWF       ADC3
                RETLW.0
;
;-------------------------------------------------------------------------------------------------------------------
; Inicializacion.- Configuración de los pines de I/O y programación de los parámetros de adquisición de
;los conversores A/D.
;-------------------------------------------------------------------------------------------------------------------
Inicializacion
                ; 1. Configuración de los pines I/O y WD:
                BCF     STATUS,RP0      ; Banco 0
                CLRF    PORTA
                CLRF    PORTB
                BSF     STATUS, RP0     ; Banco 1
                MOVLW       0x1C        ; 0x1C = 00011100b: RA2, RA3 y RA4 entradas, el resto
                MOVWF       TRISA       ;salidas
                MOVLW       0x73        ; 0x73 = 01110011b: RB0, RB1, RB4, RB5 y RB6 entradas,
                MOVWF       TRISB       ; el resto salidas
                MOVLW       0x0F                ; Prescaler del WD a 1:128, activa pull-ups puerto B
                MOVWF       OPTION_R            ;para que RB0/INT se quede en alto cuando Ser1 se
                                                ;pone en transmisión
                BCF     STATUS,RP0      ; Banco 0
                BCF     PORTA, OutRecTrans      ; Inicialmente, Ser1 en recepción, con línea de salida
                BSF     PORTA, OutData          ;de datos a 1

                ; 2. Configuración de la interrupción externa INT/RB0:
                BCF     STATUS,RP0      ; Banco 0
                CLRF    INTCON          ; Pone a 0 todos los flags de interrupción
                BSF     STATUS, RP0     ; Banco 1
                                        ; Interrupción externa (PPS):
                BSF     OPTION_R, INTEDG        ; Selecciona flanco ascendente para la interrupción
                                                ;externa
                BCF     STATUS, RP0     ; Banco 0
                BSF     INTCON, INTE    ; Habilita interrupción externa
                BSF     INTCON, GIE     ; Habilita todas las interrupciones no enmascaradas

                ; 3. Espera la secuencia de configuración del módulo SerPar
WaitConfig
                CALL    LeeConfig
                SUBLW.1
                BTFSC   STATUS, Z       ; Si el número de pulsos no era correcto, vuelve a la
                GOTO    WaitConfig      ;espera de otra secuencia.
                BCF     INTCON, INTE    ; Después de leer la configuración deshabilita la interrupción
                                        ;externa, que no se va a usar más.
                ; 4. Retardo de 3 segundos antes de inicializar los CADs, para estar seguros de
                ;que se han reseteado bien:
                CALL    Retardo1s
                CALL    Retardo1s
                CALL    Retardo1s

                ; 5. Programación de los parámetros de operación en el registro de control de los
                ;AD7710s:
                CALL    PalCtrol        ; Asigna valores a ADC1, ADC2 y ADC3 en función de la
                                        ;frec. de muestreo y ganancia leídas en 'LeeConfig'.
                MOVLW       .24         ; Para llevar la cuenta del número de bits transmitidos
                MOVWF       ContPulsADC
                BSF     STATUS, RP0     ; Banco 1
                MOVLW       0x03        ; Pines 0 y 1 del puerto B como entradas. El resto, salidas
                MOVWF       TRISB       ;(cambia los pines de datos a salidas para poder escribir la
                                        ;palabra de control).
```

```
                BCF     STATUS, RP0     ; Banco 0
                CALL    Delay
                BCF     PORTB, CLK      ; Baja SCLK
                BCF     PORTB, TFS      ; Baja TFS (escritura)
                BSF     PORTB, RFS      ; Sube RFS
                BCF     PORTB, A0       ; Baja A0 (acceso al reg. de control)
                CALL    Delay
prog1
                RLF     ADC3, F         ; Desplaza las tres variables un lugar a la izquierda. El
                RLF     ADC2, F         ;último bit se queda en CARRY
                RLF     ADC1, F
                BTFSS   STATUS, C       ; Pone a 0 o 1 los tres pines de datos, en función del
                BCF     PORTB, DAT1     ;valor del último bit desplazado
                BTFSS   STATUS, C
                BCF     PORTB, DAT2
                BTFSS   STATUS, C
                BCF     PORTB, DAT3
                BTFSC   STATUS, C
                BSF     PORTB, DAT1
                BTFSC   STATUS, C
                BSF     PORTB, DAT2
                BTFSC   STATUS, C
                BSF     PORTB, DAT3
                CALL    Delay
                BSF     PORTB, CLK      ; Realiza la escritura subiendo y bajando CLK.
                CALL    Delay
                BCF     PORTB, CLK
                CALL    Delay
                DECFSZ          ContPulsADC, F ;Si ha escrito todos los bits, pone todas las salidas a
                GOTO    prog1                  ;y vuelve a configurar los pines de datos como
                BSF     PORTB, DAT1            ;entradas. Si quedaba alguno, vuelve a prog1
                BSF     PORTB, DAT2
                BSF     PORTB, DAT3
                CALL    Delay
                BSF     STATUS, RP0     ; Banco 1
                MOVLW   0x73            ; Vuelve a configurar los pines de datos como entradas, para
                MOVWF           TRISB   ;la lectura de datos.
                BCF     STATUS, RP0     ; Banco 0
                CALL    Delay
                BSF     PORTB, TFS      ; Sube TFS
                BSF     PORTB, RFS      ; Sube RFS
                BSF     PORTB, A0       ; Sube A0
                CLRWDT                  ; Refresca el WD
                CALL    Delay
                CALL    Delay

                ; 6. Inicialización de variables:
                CLRF    ContBits
                BCF     STATUS, C
                CLRF    ContPulsConf
                CLRF    Flag
                CLRF    ContBloqPulsTr

                RETURN
```

```
;
;-------------------------------------------------------------------------------------------------------------------
; Delay2 y Delay.- Funciones para introducir retardos. Aunque están pensadas para puerto serie, se
;aprovechan también para otras funciones.
;-------------------------------------------------------------------------------------------------------------------
Delay2
                MOVLW       BAUDR
                MOVWF       DlyCnt
                GOTO        redo
Delay
                MOVLW       BAUD
                MOVWF       DlyCnt
redo
                DECFSZ      DlyCnt,Same
                GOTO        redo
                NOP
                RETLW   0
;
;-------------------------------------------------------------------------------------------------------------------
; LeeNumTarj.- Lee el código de tarjeta de los switches. El código se lee invertido, para que un switch a
;OFF corresponda a un 0 y uno a ON a un 1.
; Devuelve en W: el código de tarjeta, leído del par de pines MSBitID/LSBitID
;-------------------------------------------------------------------------------------------------------------------
LeeNumTarj
                BTFSS       PORTA, MSBitID
                GOTO        Nums2o3
                BTFSC       PORTA, LSBitID
                RETLW       0
                RETLW       1
Nums2o3
                BTFSC       PORTA, LSBitID
                RETLW       2
                RETLW       3
;
;-------------------------------------------------------------------------------------------------------------------
; Muestreo.- Función para la lectura de datos. Espera a que DRDY esté activada, lee el dato y vuelve a
;esperar a que se desactive DRDY.
;-------------------------------------------------------------------------------------------------------------------
Muestreo
                BCF         STATUS, RP0     ; Banco 0
                BCF         PORTB, CLK      ; Baja CLK
                MOVLW       .24             ; Mete 24 en el contador
                MOVWF       ContPulsADC
PollDRDY
                BTFSS   PORTB, DRDY  ; Polling a DRDY
                GOTO    PollDRDY
                NOP
                BSF     PORTB, A0       ; Sube A0 (acceso al registro de datos)
                NOP
                BCF     PORTB, RFS      ; Baja RFS (lectura)
                NOP
LeeBit
                NOP
                BCF     STATUS, C       ; Mete el nuevo bit en los bytes de datos. Para ello chequea
                RLF     ADC11, F        ;los puertos de datos y pone el nuevo bit a 0 o a 1 en función
                RLF     ADC12, F        ;de su estado. Esto lo repite 24 veces para los tres canales.
                RLF     ADC13, F
                BTFSC   PORTB, DAT1
                BSF     ADC11, 0
```

```
                BCF     STATUS, C
                RLF     ADC21, F
                RLF     ADC22, F
                RLF     ADC23, F
                BTFSC   PORTB, DAT2
                BSF     ADC21, 0

                BCF     STATUS, C
                RLF     ADC31, F
                RLF     ADC32, F
                RLF     ADC33, F
                BTFSC   PORTB, DAT3
                BSF     ADC31, 0

                BSF     PORTB, CLK      ; Sube y baja CLK para acceder al nuevo bit
                NOP
                NOP
                BCF     PORTB, CLK
                NOP
                NOP
                DECFSZ          ContPulsADC, F ; Si ha leído todos los bits, sale. Si falta alguno,
                GOTO    LeeBit                 ;vuelve a la rutina de lectura.
                NOP
                NOP
                BSF     PORTB, RFS      ; Sube RFS (fin de lectura)
PollDRDY_2
                BTFSC   PORTB, DRDY     ; Espera a que la señal DRDY se desactive
                GOTO    PollDRDY_2
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; Retardo1ms.- Genera un retardo de aproximadamente 1ms (con cristal de 10MHz).
; Número de ciclos = 1ms/(0.4us/ciclo) = 2500ciclos. Con 10 ciclos por bucle:
; Número de bucles = 2500ciclos/(10ciclos/bucle) = 250bucles
;-------------------------------------------------------------------------------------------------------------------
Retardo1ms
                MOVLW           .250
                MOVWF           Temp1
BucleRet
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                DECFSZ          Temp1, F
                GOTO    BucleRet
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; Retardo1s.- Genera un retardo de aproximadamente 1s (con cristal de 10MHz). Llama a la función
;Retardo1ms 1000veces.
;-------------------------------------------------------------------------------------------------------------------
Retardo1s
                MOVLW           .4
                MOVWF           Temp2   ; No se puede usar la variable Temp1 como contador, porque
InicTemp3                               ;es la que usa Retardo1ms.
                MOVLW           .250
                MOVWF           Temp3
```

```
MasRet1ms
                CALL        Retardo1ms
                DECFSZ      Temp3, F
                GOTO        MasRet1ms
                DECFSZ      Temp2, F
                GOTO        InicTemp3
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; RetPulso.- Genera un retardo de 9 ciclos, incluidas las instrucciones CALL y RETURN. Se han tomado
;9 ciclos para obtener pulsos de 4us (10 ciclos), contando un ciclo correspondiente a la instrucción BSF (o
;BCF).
;-------------------------------------------------------------------------------------------------------------------
RetPulso
                NOP
                NOP
                NOP
                NOP
                NOP
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; BytesMuestra.- Función de test para asignar valores conocidos a las variables de datos, y así poder
;comprobar la transmisión.
;-------------------------------------------------------------------------------------------------------------------
BytesMuestra
                MOVF   NumTarjeta, W
                MOVWF       Temp1
                MOVF   Temp1, F
                BTFSS  STATUS, Z
                GOTO   CheckTarj1
                MOVLW       .49             ; Tarjeta 0 manda '1', '2', '3',... '9'
                MOVWF       ADC13           ; .49 = '1', .50 = '2',...
                MOVLW       .50
                MOVWF       ADC12
                MOVLW       .51
                MOVWF       ADC11
                MOVLW       .52
                MOVWF       ADC23
                MOVLW       .53
                MOVWF       ADC22
                MOVLW       .54
                MOVWF       ADC21
                MOVLW       .55
                MOVWF       ADC33
                MOVLW       .56
                MOVWF       ADC32
                MOVLW       .57
                MOVWF       ADC31
                RETURN
CheckTarj1
                DECFSZ      Temp1, F
                GOTO   CheckTarj2
                MOVLW       .97             ; Tarjeta 1 manda 'a', 'b', 'c',... 'i'
                MOVWF       ADC13           ; .97 = 'a', .98 = 'b',...
                MOVLW       .98
                MOVWF       ADC12
                MOVLW       .99
                MOVWF       ADC11
                MOVLW       .100
```

```
                MOVWF   ADC23
                MOVLW   .101
                MOVWF   ADC22
                MOVLW   .102
                MOVWF   ADC21
                MOVLW   .103
                MOVWF   ADC33
                MOVLW   .104
                MOVWF   ADC32
                MOVLW   .105
                MOVWF   ADC31
                RETURN
CheckTarj2
                MOVLW   .               ; Tarjetas 2 o 3 mandan 'A', 'B', 'C',... 'I'
                MOVWF   ADC13           ; .65 = 'A', .66 = 'B',...
                MOVLW   .65
                MOVWF   ADC12
                MOVLW   .66
                MOVWF   ADC11
                MOVLW   .67
                MOVWF   ADC23
                MOVLW   .68
                MOVWF   ADC22
                MOVLW   .69
                MOVWF   ADC21
                MOVLW   .70
                MOVWF   ADC33
                MOVLW   .71
                MOVWF   ADC32
                MOVLW   .72
                MOVWF   ADC31
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; BytesMuestra2.- Función de test para asignar valores conocidos a las variables de datos, y así poder
;comprobar la transmisión. Se diferencia de 'BytesMuestra' en que asigna valores fijos,
;independientemente del número de tarjeta.
;-------------------------------------------------------------------------------------------------------------------
BytesMuestra2
                MOVLW   .49
                MOVWF   ADC13           ; .48 = '0', .49 = '1',...
                MOVLW   .50
                MOVWF   ADC12
                MOVLW   .51
                MOVWF   ADC11
                MOVLW   .52
                MOVWF   ADC23
                MOVLW   .53
                MOVWF   ADC22
                MOVLW   .54
                MOVWF   ADC21
                MOVLW   .55
                MOVWF   ADC33
                MOVLW   .56
                MOVWF   ADC32
                MOVLW   .57
                MOVWF   ADC31
                RETURN
```

```
;*******************************************************************************
; Programa principal e interrupciones
;*******************************************************************************
;-------------------------------------------------------------------------------------------------------------------------
; Rutina de servicio de la interrupción
;-------------------------------------------------------------------------------------------------------------------------
                ORG     ISR_V           ; Vector de interrupción

                BCF     STATUS, RP0     ; Banco 0
                BTFSS   INTCON, INTF    ; Comprueba si es la interrupción INT/RB0
                GOTO    FinISR          ; Si no lo es, se sale
                BSF     Flag, SerParOK  ; Si lo es, activa el flag de pulsos de SerPar. Si además nos
                BTFSC   Flag, Config    ;encontramos en la fase de espera de la secuencia de
                INCF    ContPulsConf, F ;configuración, incrementa el contador de pulsos de
EsperaFinPuls                           ;configuración
                BTFSS   PORTB, INT      ; Antes de salir de la rutina se asegura de que el pulso ha
                GOTO    EsperaFinPuls   ;terminado y la línea ha quedado a 1
FinISR
                MOVLW   0x10            ; 00010000b = 0x10: limpia todos los flags, habilita
                MOVWF   INTCON          ;la interrupción INT/RB0, y deshabilita las demás.
                RETFIE                  ;No pone el bit GIE a 1 porque eso lo hace la orden
                                        ;RETFIE.
;-------------------------------------------------------------------------------------------------------------------------
; Programa principal
;-------------------------------------------------------------------------------------------------------------------------
                ORG     RESET_V
                GOTO    Inicio
;
                ORG     0x300
;
; Inicializaciones
*********************************************************************************
;
Inicio
                ; 1. Lectura de la secuencia de configuración, cálculo de los parámetros de muestreo a
                ;partir de ella e inicialización de los AD7710s:
                CALL    Inicializacion

                ; 2. Lectura del código de tarjeta:
                CALL    LeeNumTarj      ; Devuelve el número de tarjeta en W y lo pasa a
                MOVWF   NumTarjeta      ;NumTarjeta
;
; Bucle de lectura
*********************************************************************************
;
Lectura
                ; 1. Lectura del dato:
                CALL    Muestreo

                ; Test de transmisión: envía datos fijos y conocidos
                ;CALL   BytesMuestra2

                ; 2. Inicialización del contador de bits, contador de pulsos de transmisión y del turno de
                ;transmisión en función del número de tarjeta. Para poder gestionar luego el turno de
                ;transmisión con simples desplazamientos, el valor que se asigna a 'TurnoTransm' es:
                ;   TurnoTransm = 0x00 para NumTarjeta = 0
                ;   TurnoTransm = 0x01 para NumTarjeta = 1
                ;   TurnoTransm = 0x02 para NumTarjeta = 2
                ;   TurnoTransm = 0x04 para NumTarjeta = 3
                MOVLW   .72
```

```
                MOVWF   ContBits
                MOVLW   .4              ; Por defecto el contador de bloques de pulsos de
                MOVWF   ContBloqPulsTr  ;transmisión se inicializa a 4, pero si la frecuencia de
                MOVLW   0x04            ;muestreo es de 200mps (variable 'Frec' a 0x04) se
                SUBWF Frec, W           ;inicializa a 2.
                BTFSS  STATUS, Z
                GOTO  InicTurnoTrans
                MOVLW   .2
                MOVWF   ContBloqPulsTr
InicTurnoTrans
                CLRF   TurnoTransm      ; Por defecto, valores para la tarjeta 0: Flag
                BSF    Flag, Transmite  ;'Transmite' a TRUE, TurnoTransm = 0x00
                MOVF  NumTarjeta, W
                MOVWF   Temp1
                MOVF  Temp1, F          ; Para disparar el bit Z si es 0
                BTFSC  STATUS, Z        ; Es la tarjeta 0, luego se queda con los valores por
                GOTO  Prepara1erBit     ;defecto
                BCF    Flag, Transmite  ; No es la tarjeta 0, luego pone el flag
                BSF    STATUS, C        ;'Transmite' a FALSE y TurnoTransm a uno de los
DesplazaTurno                           ;siguientes valores:
                RLF    TurnoTransm, F   ; TurnoTransm = 0x01 para la tarjeta 1
                DECFSZ         Temp1, F ; TurnoTransm = 0x02 para la tarjeta 2
                GOTO  DesplazaTurno     ; TurnoTransm = 0x04 para la tarjeta 3

                ; 3. Preparación del primer bit que se enviará:
Prepara1erBit
                BCF    Flag, BitAEnviar ; Flag a 0 por defecto
                BCF    STATUS, C        ; Limpia bit de CARRY
                RLF    ADC31, F         ; Los bytes se envían por canales (canal1, 2 y 3),
                RLF    ADC32, F         ;empezando por el MSB. Es decir, el orden es:
                RLF    ADC33, F         ; ADC13, ADC12, ADC11, ADC23, ADC22,
                RLF    ADC21, F         ; ADC21, ADC33, ADC32, ADC31.
                RLF    ADC22, F
                RLF    ADC23, F
                RLF    ADC11, F
                RLF    ADC12, F
                RLF    ADC13, F
                BTFSC  STATUS, C
                BSF    Flag, BitAEnviar

                ; 4. Si es la tarjeta 0, pone Ser1 en transmisión:
                BTFSC  Flag, Transmite
                BSF    PORTA, OutRecTrans
;
; Bucle de transmisión (sin llamadas a funciones, para optimizar en tiempo)
********************************************************************************
;
                ; 1. Espera pulso positivo de transmisión. Además de consultar la línea de pulsos de
                ;transmisión, se hace polling a la de DRDY. De esta forma se evita que el programa se
                ;quede colgado si dejan de recibirse pulsos de transmisión.
EsperaPulsTrans
                BTFSC  PORTB, DRDY      ; Polling a DRDY
                GOTO  Lectura
                BTFSC  PORTA, InPulsTrans ; Espera a que la línea esté a 0
                GOTO  EsperaPulsTrans
Espera_Uno
                BTFSC  PORTB, DRDY      ; Polling a DRDY
                GOTO  Lectura
                BTFSS  PORTA, InPulsTrans ; Espera a que la línea esté a 1
                GOTO  Espera_Uno
```

```
                ; 2. Decrementa el contador total de pulsos de transmisión. Cuando se hace cero vuelve
                ;a la lectura de datos.
                BTFSS   Flag, LastPulsTrans
                GOTO    CheckTurnoTrans
                BCF     Flag, LastPulsTrans
                BCF     PORTA, OutRecTrans      ; Después de recibir el último pulso de transmisión
                GOTO    Lectura                 ;Ser1 se deja en recepción (es necesario hacerlo
                                                ;aquí para la última tarjeta de la línea)

                ; 3. Comprueba si está en turno de transmitir y, en caso contrario, decrementa el
                ;contador de bits. Si este se hace 0, comprueba si ha llegado su turno de transmitir y
                ;activa, en caso afirmativo, el flag correspondiente.
CheckTurnoTrans
                BTFSC   Flag, Transmite
                GOTO    MandaBit
                BCF     PORTA, OutRecTrans      ; Ser1 en recepción. Se hace aquí para que el
                                                ;último bit se transmita durante un pulso de
                                                ;transmisión completo
                DECFSZ          ContBits, F     ; Si no es su turno de transmitir, decrementa el
                GOTO    EsperaPulsTrans         ;contador de pulsos y vuelve a la espera de pulso
GestBloqPulsTr

                DECFSZ          ContBloqPulsTr, F       ; Si no era el último bloque de pulsos de
                GOTO    OtroBloqPulsTr                  ;transmisión, empieza un nuevo bloque.
                BSF     Flag, LastPulsTrans     ; Si era el último bloque de pulsos de transmisión
                GOTO    EsperaPulsTrans         ;pasa a la espera del pulso final.
OtroBloqPulsTr
                MOVLW           .72             ; Cada vez que acaba un bloque de 72 pulsos,
                MOVWF           ContBits        ;comprueba si le toca transmitir.
                BCF     STATUS, C
                RRF     TurnoTransm, F          ; Si le toca transmitir, pone el flag 'Transmite' a
                BTFSC   STATUS, C               ;TRUE. Todavía no pone Ser1 en transmisión, para
                BSF     Flag, Transmite         ;conseguir que la conmutación entre recepción y
                GOTO    EsperaPulsTrans         ;transmisión sea sincronizada en todos los módulos
                                                ;de adquisición.

                ; 4. Manda el bit y gestiona contador de bits
MandaBit
                BSF     PORTA, OutRecTrans      ; Pone Ser1 en transmisión
                BTFSS   Flag, BitAEnviar
                BCF     PORTA, OutData
                BTFSC   Flag, BitAEnviar
                BSF     PORTA, OutData

                ; 5. Decrementa el contador de bits enviados:
                DECFSZ          ContBits, F
                GOTO    PreparaBit
                BCF     Flag, Transmite         ; Aunque sea el último bit, Ser1 no se deja
                GOTO    GestBloqPulsTr          ;todavía en recepción, para transmitirlo durante
                                                ;un pulso de transmisión completo
                ; 6. Prepara el siguiente bit:
PreparaBit
                BCF     Flag, BitAEnviar        ; Flag a 0 por defecto
                BCF     STATUS, C               ; Limpia bit de CARRY
                RLF     ADC31, F                ; Los bytes se envían por canales (canal1, 2 y 3),
                RLF     ADC32, F                ;empezando por el MSB. Es decir, el orden es:
                RLF     ADC33, F                ; ADC13, ADC12, ADC11, ADC23, ADC22,
                RLF     ADC21, F                ; ADC21, ADC33, ADC32, ADC31.
                RLF     ADC22, F
```

```
RLF     ADC23, F
RLF     ADC11, F
RLF     ADC12, F
RLF     ADC13, F
BTFSC   STATUS, C
BSF     Flag, BitAEnviar
GOTO    EsperaPulsTrans

END
```

ANEXO II.A.4.- PROGRAMA DE LOS PIC16C73 DE LAS TARJETAS PLL DE LOS MÓDULOS DE ADQUISICIÓN

II.A.4.a. DEFINICIONES PARA EL PROGRAMA DE LAS TARJETAS PLL DE LOS MÓDULOS DE ADQUISICIÓN

Fichero: PLL.H

```
;
; Fichero de cabecera para el programa de los PIC16C73 de las tarjetas PLL de los
;módulos de adquisición. Incluye las equivalencias de los nombres asignados a los pines
;de entrada/salida más algunas otras definiciones necesarias.
;
        NOLIST
;*****************************************************************************
; Asignación de pines
;*****************************************************************************
LEDUnlock       EQU     0       ; Pin 0 de PORTA, salida al LED 'Unlocked'
LEDSync         EQU     0       ; Pin 0 de PORTA, otro nombre más acorde con el uso que se le da
LEDTest         EQU     1       ; Pin 1 de PORTA, salida al LED 'Test'
LEDPPS          EQU     1       ; Pin 1 de PORTA, otro nombre más acorde con el uso que se le da
VerPPS          EQU     3       ; Pin 3 de PORTA, monitorización de las interrupciones de PPS
DRDY            EQU     4       ; Pin 4 de PORTB, señal DRDY de los CADs
Sync            EQU     3       ; Pin 3 de PORTC, señal /SYNC de los CADs
ResetAD         EQU     4       ; Pin 4 de PORTC, señal de Reset de la placa A/D
CtrolAux485     EQU     5       ; Pin 5 de PORTC, control transm./recep. del canal auxiliar RS-485
OutAux485       EQU     6       ; Pin 6 de PORTC, salida del canal auxiliar RS-485
InAux485        EQU     7       ; Pin 7 de PORTC, entrada del canal auxiliar RS-485

        LIST
```

II.A.4.b. PROGRAMA DE LAS TARJETAS PLL

Fichero: PLL6.ASM

```
;
; Programa del PIC16C73 de las tarjetas de conversión A/D de los módulos de
;adquisición. Los registros en este modelo son los mismos que en el 16C74 y están
;definidos en el fichero REGSC74.H (anexo I.A.4.a).

        LIST P=16C73A

        #define  FlancoPPSDesc          ; Definir para flanco de PPS descendente, comentar para
                                        ;ascendente
        INCLUDE <REGSC74.H>             ; El PIC16C73 tiene los mismos registros que el 16C74
        INCLUDE <PLL.H>

;*****************************************************************************
; Variables
;*****************************************************************************
PosVar          EQU     0x20            ; Posición de inicio de las variables
MinMSB          EQU     PosVar          ; Variables para el cálculo de la diferencia entre el retardo
MinLSB          EQU     PosVar+.1       ;PPS-DRDY y el valor correcto de ese retardo (200us =
SusMSB          EQU     PosVar+.2       ;500 ciclos de instrucción)
SusLSB          EQU     PosVar+.3
DifMSB          EQU     PosVar+.4
DifLSB          EQU     PosVar+.5
DifAntMSB       EQU     PosVar+.6
DifAntLSB       EQU     PosVar+.7
Timer1H         EQU     PosVar+.8       ; Para simular distintos valores capturados del timer 1
Timer1L         EQU     PosVar+.9
ContCheckDifs   EQU     PosVar+.10
ContReset       EQU     PosVar+.11      ; Para la gestión del reset
Temp1           EQU     PosVar+.12
Temp2           EQU     PosVar+.13
Temp3           EQU     PosVar+.14
Flag            EQU     PosVar+.15      ; Para definir indicadores (flags)
HayDRDY         EQU     0               ; Bit 0 de Flag: Los AD7710s han empezado a adquirir datos
PulsSync        EQU     1               ; Bit 1 de Flag: La siguiente int. de PPS debe generar un
                                        ;pulso de sincronismo para los AD7710s
Signo           EQU     2               ; Bit 2 de Flag: Signo de la diferencia 500-Tmr1 (0 =
                                        ;positivo, 1 = negativo)
DifCero         EQU     3               ; Bit 3 de Flag: Se activa cuando la diferencia 500-Tmr1 = 0
PPS             EQU     4               ; Bit 4 de Flag: Se ha recibido un PPS
;
;*****************************************************************************
; Funciones y macros
;*****************************************************************************
                ORG     0x030
;-----------------------------------------------------------------------------------------------------------------------
; Inicializacion.- Inicialización de los pines I/O, variables, timers, módulos CCP e interrupciones
;-----------------------------------------------------------------------------------------------------------------------
Inicializacion
                ; 1. Inicialización de pines I/O:
                BSF     STATUS, RP0     ; Banco 1
                MOVLW           0x07    ; Configura los bits del puerto A como I/O digitales
                MOVWF           ADCON1
                BCF     STATUS, RP0     ; Banco 0
```

```
CLRF    PORTA           ; Inicializa PORTA, PORTB y PORTC limpiando las salidas
CLRF    PORTB
CLRF    PORTC
BSF     STATUS, RP0     ; Banco 1
MOVLW   0x04    ; PORTA: 0x04 = 00000100b: pin 2 entrada, el resto salidas
MOVWF   TRISA
MOVLW   0x10    ; PORTB: 0x10 = 00010000b: pin 4 entrada, el resto salidas
MOVWF   TRISB
MOVLW   0x85    ; PORTC: 0x85 = 10000101b: pines 0, 2 y 7 entradas, el
MOVWF   TRISC   ;resto salidas
BCF     STATUS, RP0     ; Banco 0

; 2. Timer 1 en modo timer:
CLRF    T1CON           ; Conf. del Timer1: 0x00: Timer1 inicialmente deshabilitado,
                        ;modo timer, oscilador deshabilitado, prescaler a 1:1

; 3. Módulo CCP2 en modo PWM:
BSF     STATUS, RP0     ; Banco 1
MOVLW   0xFF    ; Fija el periodo de PWM a 9.766kHz (con cristal de 10MHz)
MOVWF   PR2     ;o 3.906kHz (con cristal de 4MHz)
BCF     STATUS, RP0     ; Banco 0
MOVLW   0x80    ; Inicializa duty-cycle del PWM (CCPR2L + CP2CON<5:4>)
MOVWF   CCPR2L;al 50% y el módulo CCP2 a modo PWM (haciendo
MOVLW   0x0C    ; CCP2CON<3:0> = 11xx)
MOVWF   CCP2CON
MOVLW   0x04    ; Habilita el Timer2 con prescaler y postscaler a 1:1
MOVWF   T2CON

; 4. Inicialización de las interrupciones y módulo CCP1 en modo captura:
CLRF    PIR1            ; Pone a 0 todos los flags de interrupción
CLRF    PIR2

; Por defecto, flanco ascendente para los PPS (Garmin 35). Si se desea flanco
;descendente, hay que habilitar la directiva #define correspondiente al principio
;del programa.
MOVLW   0x05    ; Módulo CCP1 en modo captura, interrupción con cada
ifdef   FlancoPPSDesc   ;flanco ASCENDENTE por defecto, DESCENDENTE si
MOVLW   0x04    ; 'FlancoPPSDesc' está definido al principio del programa.
endif
MOVWF   CCP1CON
BSF     INTCON, PEIE    ; Habilita interrupciones periféricos
BSF     STATUS, RP0     ; Banco 1
BSF     PIE1, CCP1IE    ; Habilita la interrupción de CCP1
BCF     STATUS, RP0     ; Banco 0 (se queda habilitado a la salida de la función)
BSF     INTCON, GIE     ; Habilita las interrupciones

; 5. Inicialización de variables:
CLRF    Flag            ; Inicializa los flags
CLRF    MinMSB          ; Limpia todas las variables para el cálculo de la diferencia
CLRF    MinLSB
CLRF    SusMSB
CLRF    SusLSB
CLRF    DifMSB
CLRF    DifLSB
CLRF    DifAntMSB
CLRF    DifAntLSB
MOVLW   .10             ;Inicializa el contador de reset a 10 (para que se
MOVWF   ContReset       ;produzca un reset de la placa A/D después de 10
                        ;segs. sin recibir PPS)
MOVLW   .10             ; La primera captura del Timer 1 y modificación del
```

```
                MOVWF       ContCheckDifs   ;VCXO se lleva a cabo a los diez segundos.
                BSF     PORTC, Sync     ; Los pulsos Sync son negativos (si no se hace esto, no se
                                        ;permite al PIC del módulo de adquisición inicializar los
                                        ;CADs).
                ; 6. Antes de permitir el control del VCXO se espera a detectar dos DRDYs. De esta
                ;forma nos aseguramos de que el control del VCXO no empieza antes que la adquisición
                ;de datos:
WaitDRDY_0A
                BTFSC  PORTB, DRDY
                GOTO   WaitDRDY_0A
WaitDRDY_1A
                BTFSS  PORTB, DRDY
                GOTO   WaitDRDY_1A
WaitDRDY_0B
                BTFSC  PORTB, DRDY
                GOTO   WaitDRDY_0B
WaitDRDY_1B
                BTFSS  PORTB, DRDY
                GOTO   WaitDRDY_1B
WaitDRDY_0C
                BTFSC  PORTB, DRDY
                GOTO   WaitDRDY_0C

                BSF     Flag, HayDRDY ; Activa el Flag de DRDY, para que la ISR empiece a realizar
                BSF     Flag, PulsSync  ;las tareas que tiene asignadas, y el Flag de PulsSync, para
                                        ;que la ISR del siguiente PPS genere un pulso de sincronismo
                                        ;de los CADs.
                RETURN
;
;-----------------------------------------------------------------------------------------------------------------------
; GestIntPPS.- Función de gestión de la interrupción de CCP1, que es disparada por la señal de un pulso
;por segundo (PPS). En primer lugar programa el timer 1 en modo timer, para ver el tiempo que tarda en
;activarse la señal DRDY de los CADs. Luego comprueba si hay que enviar un pulso de sincronización a
;los CADs y, en caso afirmativo, lo hace. Después entra en un bucle de espera, hasta que detecta que la
;señal DRDY de los CADs se ha activado. En función del tiempo que ha tardado, modifica la tensión de
;entrada del VCXO. Para que el ajuste sea suave y minimizar errores en el cálculo de la diferencia, toda
;esta gestión del VCXO sólo se lleva a cabo una vez cada diez segundos.
;-----------------------------------------------------------------------------------------------------------------------
GestIntPPS                              ; Banco 0 habilitado antes
                BCF     PIR1, CCP1IF    ; Limpia el flag de interrupción
                BSF     Flag, PPS       ; Para la gestión del reset del programa principal
                BTFSS   Flag, HayDRDY ; Sólo ejecuta el resto de la función si se ha detectado que los
                RETURN                  ;CADs han empezado a adquirir datos (hay señal DRDY)
                BCF     T1CON, TMR1ON       ; Deshabilita el timer1
                CLRF    TMR1H           ; Inicializa los registros del timer1 a 18, que es el número de
                MOVLW       .18         ;ciclos de instrucción que se pierden desde que se detecta el
                MOVWF       TMR1L ;pulso de segundo hasta que se habilita el timer1 (contando
                                        ;con 3 ciclos antes de saltar a la ISR)
                BSF     T1CON, TMR1ON       ; Habilita el timer1

                BCF     PORTA, LEDSync      ; Apaga, si estaba encendido, el LED de pulso /Sync
                BTFSS   Flag, PulsSync      ; Genera el pulso /Sync (si el flag correspondiente
                GOTO    ChkDRDY             ;está activado)
                BCF     Flag, PulsSync      ; Limpia el flag de enviar pulso /Sync
                BSF     PORTA, LEDSync
                BCF     PORTC, Sync         ; Genera un pulso de sincronización negativo de 4us
                NOP                         ;de duración (con cristal de 10MHz).
                NOP
                NOP
                NOP
```

```
                NOP
                NOP
                NOP
                NOP
                NOP
                BSF     PORTC, Sync
                GOTO    SigueInt

                ; Cada segundo comprueba si DRDY se va a cruzar con PPS. En caso afirmativo activa
                ;el flag de PulsSync, deshabilita el timer 1, inicializa el contador de segundos y las
                ;variables de diferencia y baja la frecuencia del VCXO.
ChkDRDY
                BTFSS   PORTB, DRDY
                GOTO    SigueInt
                BSF     Flag, PulsSync
                BCF     T1CON, TMR1ON           ; Deshabilita el timer 1
                MOVLW           .10
                MOVWF           ContCheckDifs
                CLRF    DifMSB
                CLRF    DifLSB
                GOTO    BajarPWM

SigueInt
                DECFSZ          ContCheckDifs, F; Las diferencias se calculan sólo cada 10 segundos
                RETURN
                MOVLW           .10
                MOVWF           ContCheckDifs

EsperaDRDY_1
                BTFSS   PORTB, DRDY
                GOTO    EsperaDRDY_1
                BCF     T1CON, TMR1ON           ; Deshabilita el timer 1
                CALL    Diferencia      ; Calcula la diferencia entre el valor correcto (200us = 500
                                        ;ciclos de instrucción) y el capturado en TMR1H:TMR1L.
                CALL    ChkSync         ; Comprueba si hay que resincronizar los conversores. Esto
                SUBLW.1                 ;se hace si la diferencia es mayor de 300us (distancia
                BTFSC   STATUS, Z       ;PPS-DRDY mayor de 500us). En caso de que sea así se
                BSF     Flag, PulsSync  ;activa el Flag de pulso /Sync para generar un pulso en el
                                        ;siguiente PPS, pero no se reinicializa el PWM porque se
                                        ;podría entrar en un bucle Sync-Reinicializar PWM.
                MOVF    DifAntMSB, W    ; Compara el valor absoluto de la diferencia con el de
                SUBWF   DifMSB, W       ;la diferencia anterior.
                BTFSC   STATUS, Z       ; Si los dos MSBs son iguales, compara los LSBs
                GOTO    CheckLSBs
                BTFSS   STATUS, C       ; Si la diferencia actual es menor que la anterior, no se hace
                GOTO    FinGestIntPPS   ;nada (nos estamos acercando al valor correcto).
                GOTO    CheckSigno      ; Si la diferencia actual es mayor o igual que la anterior
                                        ;se comprueba el signo de la actual para ver si hay que
                                        ;subir o bajar el PWM.
CheckLSBs
                MOVF    DifAntLSB, W    ; Compara el valor absoluto de la diferencia con el de la
                SUBWF   DifLSB, W       ;diferencia anterior.
                BTFSC   STATUS, C
                GOTO    CheckSigno      ; Si la diferencia actual es mayor o igual que la anterior se
                                        ;comprueba el signo de la actual para ver si hay que subir o
                                        ;bajar el PWM
                GOTO    FinGestIntPPS   ; Si la diferencia actual es menor que la anterior, no se
                                        ;modifica el VCXO (nos estamos acercando al valor
                                        ;correcto).
```

```
CheckSigno
                BTFSC  Flag, DifCero     ; Si la diferencia actual es cero, no se hace nada (¡estamos
                GOTO   FinGestIntPPS     ;en la frecuencia buena y con el retardo bueno!)
                BTFSC  Flag, Signo
                GOTO   SubirPWM
BajarPWM
                CALL   BajaPWM
                GOTO   FinGestIntPPS
SubirPWM                                 ; Signo negativo => 500 < Tmr1 => Hay que aumentar PWM
                CALL   SubePWM
FinGestIntPPS
                MOVF   DifMSB, W                ; Antes de salir, pasamos la diferencia recién
                MOVWF          DifAntMSB        ; calculada a las variables de diferencia anterior, para
                MOVF   DifLSB, W                ;la próxima vez.
                MOVWF          DifAntLSB
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------------
; ChkSync.- Comprueba si hay que resincronizar los conversores. Esto se hace si la diferencia entre la
;posición buena de la señal DRDY y la real es mayor de 300us (que equivale a una distancia PPS-DRDY
;mayor de 500us). La diferencia entre la posición buena y la real se ha calculado antes en la función
;'Diferencia', y está en las variables DifLSB y DifMSB.
; Devuelve en W: 0 si el valor absoluto de la diferencia entre la posición buena de DRDY y la medida es
;menor que 300us (750 ciclos de instrucción = 0x2EE), 1 si es mayor.
;-------------------------------------------------------------------------------------------------------------------------
ChkSync
                MOVLW          0x02
                SUBWF DifMSB, W
                BTFSS  STATUS, C
                RETLW .0                ; Si el MSB de la diferencia es menor que 2, está bien.
                BTFSS  STATUS, Z
                RETLW          .1       ; Si el MSB de la diferencia es mayor que 2, está desajustado.
                MOVLW          0xEE     ; Si es igual, hay que comprobar el LSB de la diferencia.
                SUBWF DifLSB, W
                BTFSS  STATUS, C
                RETLW .0                ; Si el MSB es igual a 2 y el LSB es menor que 0xEE, está
                                        ;bien.
                RETLW .1                ; Si el MSB es igual a 2 y el LSB es mayor o igual que 0xEE,
                                        ;está desajustado.
;
;-------------------------------------------------------------------------------------------------------------------------
; SubePWM.- Sube el duty-cycle de la señal PWM, a través de los bits CCP2X:CCP2Y de CCP2CON
;(LSbs) y del registro CCPR2L.
;-------------------------------------------------------------------------------------------------------------------------
SubePWM
                BTFSC  CCP2CON, CCP2Y   ; Comprueba el estado del LSb del PWM (bit
                                        ; CCP2Y)
                GOTO   IncCCP2X         ; Si es 1, hay que incrementar el siguiente bit
                BSF    CCP2CON, CCP2Y   ; Si es 0 basta con ponerlo a 1, no es necesario
                RETURN                  ;incrementar el registro CCPR2L
IncCCP2X
                BCF    CCP2CON, CCP2Y   ; Pone el LSb a 0
                BTFSC  CCP2CON, CCP2X   ; Comprueba el estado del 2º bit del PWM (bit
                                        ; CCP2X)
                GOTO   IncCCPR2L        ; Si el 2º bit es 1, hay que incrementar el registro
                                        ; CCPR2L
                BSF    CCP2CON, CCP2X   ; Si el 2º bit es 0 basta con ponerlo a 1, no es
                RETURN                  ;necesario incrementar el registro CCPR2L
```

```
IncCCPR2L
                BCF     CCP2CON, CCP2X      ; CCP2X:CCP2Y = 11, por lo que se ponen los dos a
                INCF    CCPR2L, F           ;0 y se incrementa el registro CCPR2L
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; BajaPWM.- Baja el duty-cycle de la señal PWM, a través de los bits CCP2X:CCP2Y de CCP2CON
;(LSbs) y del registro CCPR2L.
;-------------------------------------------------------------------------------------------------------------------
BajaPWM
                BTFSS   CCP2CON, CCP2Y      ; Comprueba el estado del LSb del PWM (bit
                                            ; CCP2Y)
                GOTO    DecCCP2X            ; Si es 0, hay que decrementar el siguiente bit
                BCF     CCP2CON, CCP2Y      ; Si es 1 basta con ponerlo a 0, no es necesario
                RETURN                      ;decrementar el registro CCPR2L
DecCCP2X
                BSF     CCP2CON, CCP2Y      ; Pone el LSb a 1
                BTFSS   CCP2CON, CCP2X      ; Comprueba el estado del 2º bit del PWM (bit
                                            ; CCP2X)
                GOTO    DecCCPR2L           ; Si el 2º bit es 0, hay que decrementar el registro
                                            ; CCPR2L
                BCF     CCP2CON, CCP2X      ; Si el 2º bit es 1 basta con ponerlo a 0, no es
                RETURN                      ;necesario decrementar el registro CCPR2L
DecCCPR2L
                BSF     CCP2CON, CCP2X      ; CCP2X:CCP2Y = 00, por lo que se ponen los dos a
                DECF    CCPR2L, F           ;1 y se decrementa el registro CCPR2L
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; Diferencia.- Calcula la diferencia entre el retardo correcto (entre las señales de PPS y DRDY) y el
;medido. El retardo medido es el capturado en el par de registros del timer 1 (TMR1H:TMR1L). El
;retardo correcto son 500 ciclos de instrucción (200us).
; Para el cálculo usa las variables MinMSB, MinLSB, SusMSB, SusLSB, DifMSB, DifLSB (Minuendo,
;sustraendo y diferencia). Como salida da el valor absoluto de la diferencia en las variables DifMSB y
;DifLSB y el signo en el flag de signo: 0 si la diferencia es positiva o cero (es decir, el valor medido es
;menor o igual que el correcto) y 1 si es negativa (valor medido mayor que el correcto).
;-------------------------------------------------------------------------------------------------------------------
Diferencia
                MOVLW       0x01            ; La cuenta correcta (500ciclos = 0x1F4) se mete en
                MOVWF       MinMSB          ;el minuendo
                MOVLW       0xF4
                MOVWF       MinLSB
                MOVF    TMR1H, W            ; El valor capturado del timer 1 se mete en el
                MOVWF       SusMSB          ; sustraendo
                MOVF    TMR1L, W
                MOVWF       SusLSB

                BCF     Flag, DifCero       ; Por defecto, la diferencia es distinta de cero y de
                BCF     Flag, Signo         ;signo positivo (flags de cero y de signo a 0)
Resta
                MOVF    SusLSB, W
                SUBWF   MinLSB, W
                MOVWF       DifLSB
                BTFSC   STATUS, C           ; Si el resultado es negativo, se le suma 1 al MSB del
                GOTO    SigueResta          ;sustraendo
                MOVLW       .1
                ADDWF       SusMSB, F
SigueResta
                MOVF    SusMSB, W
                SUBWF   MinMSB, W
```

```
                MOVWF   DifMSB
                BTFSS   STATUS, C
                GOTO    DifNeg
                BTFSS   STATUS, Z
                RETURN                  ; Antes de salir, comprueba si la diferencia es 0 (tanto el
                MOVF    DifLSB, F       ;MSB recién calculado como el LSB calculado antes). En
                BTFSS   STATUS, Z       ;caso afirmativo, activa el flag correspondiente.
                RETURN
                BSF     Flag, DifCero   ; Si la diferencia es 0, se activa el flag DifCero antes de salir
                RETURN
DifNeg
                BSF     Flag, Signo     ; Si llega aquí, la diferencia es negativa => se pone a 1 el flag
                                        ;de signo y se hace la resta al revés (valor capturado del
                                        ;timer 1 - 500)
                MOVF    TMR1H, W
                MOVWF   MinMSB
                MOVF    TMR1L, W
                MOVWF   MinLSB
                MOVLW   0x01
                MOVWF   SusMSB
                MOVLW   0xF4
                MOVWF   SusLSB

                GOTO    Resta
;
;-------------------------------------------------------------------------------------------------------------------
; GestReset.- Gestiona el reseteo del PIC de la placa de conversores. Cuando pasan diez segundos sin
;recibirse pulsos de segundo se manda un pulso de reset a la placa de conversores y se salta al inicio del
;programa.
; Devuelve en W: 1 si se ha cumplido el time out y se ha enviado un pulso de reset a la placa o placas de
;conversores, 0 en caso contrario.
;-------------------------------------------------------------------------------------------------------------------
GestReset
                CALL    Retardo1s
                BTFSS   Flag, PPS
                GOTO    DecCont
                BCF     Flag, PPS               ; Si se ha recibido un PPS se reinicializa el contador
                MOVLW   .10                     ;de reset
                MOVWF   ContReset
                RETLW.0
DecCont
                DECFSZ  ContReset, F    ; Si el contador de reset se hace cero se envía un
                RETLW.0                 ;pulso de reset a las placas de A/D
                BSF     PORTC, ResetAD
                CALL    Retardo1s
                BCF     PORTC, ResetAD
                RETLW.1
;
;-------------------------------------------------------------------------------------------------------------------
; Retardo1ms.- Genera un retardo de aproximadamente 1ms (con cristal de 10MHz).
; Número de ciclos = 1ms/(0.4us/ciclo) = 2500ciclos. Con 10 ciclos por bucle:
; Número de bucles = 2500ciclos/(10ciclos/bucle) = 250bucles
;-------------------------------------------------------------------------------------------------------------------
Retardo1ms
                MOVLW   .250
                MOVWF   Temp1
BucleRet
                NOP
                NOP
                NOP
```

```
                NOP
                NOP
                NOP
                NOP
                DECFSZ          Temp1, F
                GOTO   BucleRet
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; Retardo1s.- Genera un retardo de aproximadamente 1s (con cristal de 10MHz).
; Hay que llamar a la función Retardo1ms 1000veces. No se puede usar la variable Temp1 como
;contador, porque es la que usa Retardo1ms.
;-------------------------------------------------------------------------------------------------------------------
Retardo1s
                MOVLW           .4
                MOVWF           Temp2
InicTemp3
                MOVLW           .250
                MOVWF           Temp3
MasRet1ms
                CALL    Retardo1ms
                DECFSZ          Temp3, F
                GOTO   MasRet1ms
                DECFSZ          Temp2, F
                GOTO   InicTemp3
                RETURN
```

```
;
;*****************************************************************************
; Programa principal e interrupciones
;*****************************************************************************
;-----------------------------------------------------------------------------------------------------------------------
; Rutina de servicio de la interrupción
;-----------------------------------------------------------------------------------------------------------------------
                ORG     ISR_V           ; Vector de interrupción

                BCF     STATUS, RP0     ; Banco 0
                BTFSS   PIR1, CCP1IF    ; Comprueba si es la interrupción de CCP1 (PPS)
                GOTO    FinISR
                BSF     PORTA, LEDPPS           ; Enciende el LED de PPS
                CALL    GestIntPPS
                CALL    Retardo1ms      ; Retardo de 2ms, para que pueda verse el LED de PPS
                CALL    Retardo1ms
                BCF     PORTA, LEDPPS           ; Apaga el LED de PPS
FinISR
                BCF     INTCON, RBIF    ; Limpia los flags de todas las interrupciones salvo la de
                BCF     INTCON, INTF    ;CCP1, que se limpia en la función 'GestIntPPS'
                BCF     INTCON, T0IF
                MOVLW           0x04
                ANDWF           PIR1, F
                CLRF    PIR2
                RETFIE
;
;-----------------------------------------------------------------------------------------------------------------------
; Programa principal
;-----------------------------------------------------------------------------------------------------------------------
                ORG     RESET_V
                GOTO    Inicio
;
                ORG     0x750
Inicio
                ; 1. Inicialización de puertos de I/O y variables y programación de interrupciones,
                ;timers y módulos CCP. El programa no sale de esta función hasta que detecta dos
                ;señales DRDY.
                CALL    Inicializacion

                ; 2. Bucle principal, en el que sólo se comprueba el tiempo que ha transcurrido sin PPS
                ;para ver si hay que mandar un carácter de reset a la placa de adquisición. El control del
                ;VCXO se realiza en la ISR (función 'GestIntPPS').
bucle
                CALL    GestReset
                SUBLW.1                 ; Si GestReset devuelve un 1 hay un comando de reset del
                BTFSC   STATUS, Z       ;sistema, por lo que se salta al inicio del programa después de
                GOTO    Inicio          ;enviar un pulso de reset a la placa de conversores A/D.
                GOTO    bucle
                END
```

ANEXO II.A.5.- PROGRAMA DEL PIC16C74 DEL MÓDULO DE RELOJ DE LA ANTENA DEL VESUBIO

II.A.5.a. DEFINICIONES PARA EL PROGRAMA DEL MÓDULO DE RELOJ

Fichero: RELOJ.H

```
;
; Fichero de cabecera para el programa del PIC16C74 del módulo de reloj. Incluye las
;equivalencias de los nombres asignados a los pines de entrada/salida más algunas otras
;definiciones necesarias.
;
        NOLIST
;*****************************************************************************
; Asignación de pines
;*****************************************************************************
; ASIGNACIÓN DE PINES PARA LAS LÍNEAS DE CONTROL DEL GPS Y COMUNICACIÓN
;CON SERPAR:
; Puerto B:
InPPS           EQU     0       ; Entrada de PPS del GPS
TXaGPS          EQU     1       ; Salida de datos al GPS
RXdeGPS         EQU     2       ; Entrada de datos del GPS
; Puerto C:
OutPPS          EQU     0       ; Salida de PPS
LEDPPS          EQU     2       ; Salida al LED de PPS
CtrolRT0        EQU     1       ; Salida de control recepción/transmisión canal 0 RS-485
CtrolRT1        EQU     4       ; Salida de control recepción/transmisión canal 1 RS-485
CtrolRT2        EQU     5       ; Salida de control recepción/transmisión canal 2 RS-485
;
; CARACTERES PERMITIDOS EN LA COMUNICACIÓN SERIE CON SERPAR:
SPWaitData      EQU     .65     ; Carácter 'SerPar preparado para recibir datos GPS' ('A')
SysReset        EQU     .90     ; Comando de Reset del sistema
;
; CONSTANTES PARA EL RETARDO DE LA COMUNICACIÓN SERIE CON EL GPS:
CteRetSer       EQU     .82     ; Para 9600bps y cristal de 10MHz

        LIST
```

II.A.5.b. PROGRAMA DEL MÓDULO DE RELOJ

Fichero: RELOJ5.ASM

```
;
; Programa del PIC16C74 del módulo de reloj. Los registros de este modelo están
;definidos en el fichero REGSC74.H (anexo I.A.4.a).

        LIST P=16C74A

        INCLUDE <REGSC74.H>
        INCLUDE <RELOJ.H>

;*****************************************************************************
; Variables
;*****************************************************************************
PosVar          EQU     0x20            ; Posición de inicio de las variables
Temp1           EQU     PosVar          ; Temp1 y Temp2: variables temporales que se usan para
Temp2           EQU     PosVar+.1       ;distintas tareas (¡no usarlas en rutinas de interrupción!).
ContRetardo     EQU     PosVar+.2       ; Contador para la generación de retardos
ContReset       EQU     PosVar+.3       ; Contador de segundos durante los cuales se inhiben los
                                        ;PPS en caso de reset
RegTransm       EQU     PosVar+.4       ; Registro de transmisión para la comunicación con el GPS
ContBitsTX      EQU     PosVar+.5       ; Contador de bits transmitidos al GPS
DatoInSP        EQU     PosVar+.6       ; Dato recibido del módulo SerPar
DatoOutSP       EQU     PosVar+.7       ; Dato que se enviará al módulo SerPar
DatoInGPS       EQU     PosVar+.8       ; Dato recibido del GPS
DatoOutGPS      EQU     PosVar+.9       ; Dato que se enviará al GPS
Status0         EQU     PosVar+.10      ; Byte 0 del paquete de status del GPS
Status1         EQU     PosVar+.11      ; Byte 1 del paquete de status del GPS
Time0           EQU     PosVar+.12      ; Byte 0 del paquete de tiempo del GPS
Time1           EQU     PosVar+.13      ; Byte 1 del paquete de tiempo del GPS
Time2           EQU     PosVar+.14      ; Byte 2 del paquete de tiempo del GPS
Time3           EQU     PosVar+.15      ; Byte 3 del paquete de tiempo del GPS
Time4           EQU     PosVar+.16      ; Byte 4 del paquete de tiempo del GPS
Time5           EQU     PosVar+.17      ; Byte 5 del paquete de tiempo del GPS
Time6           EQU     PosVar+.18      ; Byte 6 del paquete de tiempo del GPS
Time7           EQU     PosVar+.19      ; Byte 7 del paquete de tiempo del GPS
Time8           EQU     PosVar+.20      ; Byte 8 del paquete de tiempo del GPS
Time9           EQU     PosVar+.21      ; Byte 9 del paquete de tiempo del GPS

Flag            EQU     PosVar+.22      ; Para definir flags y otras variables booleanas
SerParWait      EQU     0               ; Bit 0 de Flag: SerPar a la espera de datos
Reset           EQU     1               ; Bit 1 de Flag: SerPar ha mandado una orden de Reset
Paridad         EQU     2               ; Bit 2 de Flag: Bit de paridad
DataOK          EQU     3               ; Bit 3 de Flag: Se han recibido los paquetes de status
                                        ;y tiempo antes del siguiente PPS
PPS             EQU     4               ; Bit 4 de Flag: PPS recibido
PrimeraVez      EQU     5               ; Bit 5 de Flag: Primer dato enviado, para saber cuando actúa
                                        ;el WD
PPSLargo        EQU     6               ; Hay que enviar un PPS de 20 ms  en lugar de uno de 10ms.
;
```

```
;*****************************************************************************
; Funciones y macros
;*****************************************************************************
                ORG     0x100
;
;-----------------------------------------------------------------------------------------------------------------------
; Inicializacion.- Configuración de los pines de I/O e inicialización de variables.
;-----------------------------------------------------------------------------------------------------------------------
Inicializacion
                ; 1. Inicialización del WD con prescaler para obtener el periodo máximo (2.3 seg):
                BSF     STATUS, RP0             ; Banco 1
                CLRWDT
                BSF     OPTION_R, PS2           ; Prescaler a 1:128 (<PS2:PS0> = 111)
                BSF     OPTION_R, PS1
                BSF     OPTION_R, PS0
                BSF     OPTION_R, PSA           ; Prescaler asignado al WD

                ; 2. Configuración de los puertos I/O:
                BSF     STATUS, RP0             ; Banco 1
                MOVLW           0x07            ; Configura los puertos A y E como I/O digitales
                MOVWF           ADCON1          ; Si no se hace esto los puertos A y E no funcionan
                BCF     STATUS, RP0             ; Banco 0
                CLRF    PORTB                   ; Configura el puerto B
                BSF     STATUS, RP0             ; Banco 1
                MOVLW           0x05            ; RB0 y RB2 entradas, el resto salidas (0x05 =
                MOVWF           TRISB           ; = 00000101b)
                BCF     STATUS, RP0             ; Banco 0
                CLRF    PORTC                   ; Configura el puerto C
                BSF     STATUS, RP0             ; Banco 1
                MOVLW           0xCC            ; RC2 y RC3 entradas, RC6 y RC7 USART (bits a
                MOVWF           TRISC           ;1), el resto salidas (0xCC = 11001100b)
                BCF     TRISE, PSPMODE          ; Configura el puerto D como I/O de propósito
                                                ;general, no como puerto paralelo esclavo
                BCF     STATUS, RP0             ; Banco 0
                CLRF    PORTD                   ; Configura puerto D
                BSF     STATUS, RP0             ; Banco 1
                CLRF    TRISD                   ; Todos los pines como salidas

                ; 3. Inicialización de variables y configuración transceptores RS-485 como transmisores
                ;o receptores:
                BCF     STATUS, RP0             ; Banco 0
                CLRF    Flag                    ; Todos los flags a 0
                BSF     Flag, PrimeraVez        ; Activa el flag de primera vez para saber cuándo se
                                                ;resetea
                BSF     PORTC, CtrolRT0         ; Canal 0 RS-485 en transmisión (PPS)
                BSF     PORTC, CtrolRT1         ; Canal 1 RS-485 en transmisión (GPS Out)
                BCF     PORTC, CtrolRT2         ; Canal 2 RS-485 en recepción (GPS In)
                BSF     PORTC, OutPPS           ; Los PPS de salida serán negativos, por lo que
                                                ;inicialmente ponemos la salida a 1
                BSF     PORTB, TXaGPS           ; Salida de datos al GPS a 0 inicialmente

                ; 4. Configuración parámetros USART:
                BSF     STATUS, RP0             ; Banco 1
                CLRF    SPBRG                   ; Máxima velocidad posible (156.3kbps con cristal
                                                ;de 10 MHz en modo de baja velocidad).
                MOVLW           0x20            ; Registro de transmisión: transmisión habilitada, 8
                MOVWF           TXSTA           ;bits de datos en transmisión, modo asíncrono, baja
                                                ;velocidad.
                BCF     STATUS, RP0             ; Banco 0
                MOVLW           0x90            ; Registro de recepción: puerto serie habilitado, 8
```

```
MOVWF     RCSTA      ;bits de datos en recepción, recepción habilitada.

; 5. Inicialización del GPS, mediante los comandos 0x25 (reset software) y 0x35
;(opciones de entrada/salida):
MOVLW     0x10       ; Byte <DLE> = 0x10
MOVWF     DatoOutGPS
CALL   EnviaDatoGPS
MOVLW     0x25       ; Byte <id> = 0x25
MOVWF     DatoOutGPS
CALL   EnviaDatoGPS
MOVLW     0x10       ; Byte <DLE> = 0x10
MOVWF     DatoOutGPS
CALL   EnviaDatoGPS
MOVLW     0x03       ; Byte <ETX> = 0x03
MOVWF     DatoOutGPS
CALL   EnviaDatoGPS

CALL   Retardo250ms  ; Retardo de tres segundos entre los dos comandos,
CALL   Retardo250ms  ;para dar tiempo al receptor a que se reinicie después
CALL   Retardo250ms  ;del comando 0x25 antes de enviarle el 0x35
CALL   Retardo250ms
CLRWDT
CALL   Retardo250ms
CALL   Retardo250ms
CALL   Retardo250ms
CALL   Retardo250ms
CLRWDT
CALL   Retardo250ms
CALL   Retardo250ms
CALL   Retardo250ms
CALL   Retardo250ms
CLRWDT

MOVLW     0x10       ; Byte <DLE> = 0x10
MOVWF     DatoOutGPS
CALL   EnviaDatoGPS
MOVLW     0x35       ; Byte <id> = 0x35
MOVWF     DatoOutGPS
CALL   EnviaDatoGPS
MOVLW     0x02       ; Byte 0 de datos
MOVWF     DatoOutGPS
CALL   EnviaDatoGPS
MOVLW     0x00       ; Byte 1 de datos
MOVWF     DatoOutGPS
CALL   EnviaDatoGPS
MOVLW     0x27       ; Byte 2 de datos: el bit 5 está a 1 para que el GPS dé
MOVWF     DatoOutGPS ;PPS por defecto
CALL   EnviaDatoGPS
MOVLW     0x00       ; Byte 3 de datos
MOVWF     DatoOutGPS
CALL   EnviaDatoGPS
MOVLW     0x10       ; Byte <DLE> = 0x10
MOVWF     DatoOutGPS
CALL   EnviaDatoGPS
MOVLW     0x03       ; Byte <ETX> = 0x03
MOVWF     DatoOutGPS
CALL   EnviaDatoGPS
```

```
                ; 6. Inicialización de las interrupciones:
                CLRF    INTCON              ; Pone a 0 todos los flags de interrupción y
                CLRF    PIR1                ;deshabilita todas las interrupciones
                CLRF    PIR2
                BSF     STATUS, RP0         ; Banco 1
                CLRF    PIE1
                CLRF    PIE2
                                    ; Interrupción externa (PPS):
                BSF     OPTION_R, INTEDG    ; Selecciona flanco ascendente para el PPS
                BCF     STATUS, RP0         ; Banco 0
                BSF     INTCON, INTE        ; Habilita interrupción externa

                                    ; Interrupción USART (carácter recibido):
                BSF     STATUS, RP0         ; Banco 1
                BSF     PIE1, RCIE          ; Habilita interrupción USART para recepción
                BCF     STATUS, RP0         ; Banco 0

                                    ; Interrupción del Tmr1:
                BSF     STATUS, RP0         ; Banco 1
                BSF     PIE1, TMR1IE        ; Habilita interrupción de timer 1
                BCF     STATUS, RP0         ; Banco 0
                BSF     INTCON, PEIE        ; Habilita todas las int. de periféricos no
                                            ;enmascaradas

                BSF     INTCON, GIE ; Habilita todas las interrupciones no enmascaradas

                RETURN

;-------------------------------------------------------------------------------------------------------------------
; Retardo250ms.- Genera un retardo de aproximadamente 250ms (con cristal de 10MHz).
; Número de ciclos = 1ms/(0.4us/ciclo) = 2500ciclos. Con 10 ciclos por bucle:
; Número de bucles = 2500ciclos/(10ciclos/bucle) = 250bucles
;-------------------------------------------------------------------------------------------------------------------
Retardo250ms
                MOVLW       .250            ; No se puede usar Temp1, lo está usando
                MOVWF       Temp2           ;Retardo1ms
Mas_ms
                CALL    Retardo1ms
                DECFSZ      Temp2, F
                GOTO    Mas_ms
                RETURN

;-------------------------------------------------------------------------------------------------------------------
; MandarDatos.- Comprueba el flag 'DataOK' y, si está activado, manda los datos de status y tiempo
;(variables Status0, Status1, Time0, Time1,... Time9) al módulo SerPar a través del puerto serie. Si el flag
;no está activado, se mandan doce bytes fijos (carácter 0x31 = '1'). También se comprueba el flag
;'PrimeraVez' y, en caso de que esté activado (lo cual correspondería a un reseteo del PIC, sea por un
;reinicio de todo el sistema o porque ha actuado el WD) se mandan doce caracteres 0x30 ('0').
; Devuelve en W: 0 si había datos listos, 1 si no había (se han enviado doce 0x31), 2 si era la primera vez
;que se mandaban datos (se han enviado doce caracteres 0x30).
;-------------------------------------------------------------------------------------------------------------------
MandarDatos
                BTFSS   Flag, PrimeraVez
                GOTO    MandarDatos1
                BCF     Flag, PrimeraVez
                MOVLW       .12
                MOVWF       Temp1
                MOVLW       0x30
                MOVWF       DatoOutSP
```

```
Mas0x30
        CALL    EnviaDatoSer
        DECFSZ          Temp1, F
        GOTO    Mas0x30
        RETLW.2

MandarDatos1
        BTFSS   Flag, DataOK
        GOTO    Manda0x31
        BCF     Flag, DataOK                    ; Desactiva el flag

        MOVF    Status0, W                      ; Manda los doce bytes de datos
        MOVWF           DatoOutSP
        CALL    EnviaDatoSer
        MOVF    Status1, W
        MOVWF           DatoOutSP
        CALL    EnviaDatoSer
        MOVF    Time0, W
        MOVWF           DatoOutSP
        CALL    EnviaDatoSer
        MOVF    Time1, W
        MOVWF           DatoOutSP
        CALL    EnviaDatoSer
        MOVF    Time2, W
        MOVWF           DatoOutSP
        CALL    EnviaDatoSer
        MOVF    Time3, W
        MOVWF           DatoOutSP
        CALL    EnviaDatoSer
        MOVF    Time4, W
        MOVWF           DatoOutSP
        CALL    EnviaDatoSer
        MOVF    Time5, W
        MOVWF           DatoOutSP
        CALL    EnviaDatoSer
        MOVF    Time6, W
        MOVWF           DatoOutSP
        CALL    EnviaDatoSer
        MOVF    Time7, W
        MOVWF           DatoOutSP
        CALL    EnviaDatoSer
        MOVF    Time8, W
        MOVWF           DatoOutSP
        CALL    EnviaDatoSer
        MOVF    Time9, W
        MOVWF           DatoOutSP
        CALL    EnviaDatoSer
        RETLW.0
Manda0x31
        MOVLW           .12
        MOVWF           Temp1
        MOVLW           0x31
        MOVWF           DatoOutSP
Mas0x31
        CALL    EnviaDatoSer
        DECFSZ          Temp1, F
        GOTO    Mas0x31
        RETLW.1
```

```
;
;-------------------------------------------------------------------------------------------------------------------
; EnviaDatoSer.- Envía  por el puerto serie el byte cargado en la variable 'DatoOutSP'. Antes de hacerlo se
;asegura de que el registro de transmisión (TXREG) está vacío, consultando el estado del bit TXIF y
;esperando en caso de que no esté activado.
;-------------------------------------------------------------------------------------------------------------------
EnviaDatoSer
                BTFSS  PIR1, TXIF
                GOTO   EnviaDatoSer
                MOVF   DatoOutSP, W
                MOVWF          TXREG
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; SendPetStatus.- Manda un paquete de petición de status al GPS (paquete 0x26).
;-------------------------------------------------------------------------------------------------------------------
SendPetStatus
                MOVLW          0x10            ; Byte <DLE> = 0x10
                MOVWF          DatoOutGPS
                CALL   EnviaDatoGPS
                MOVLW          0x26            ; Byte <id> = 0x26
                MOVWF          DatoOutGPS
                CALL   EnviaDatoGPS
                MOVLW          0x10            ; Byte <DLE> = 0x10
                MOVWF          DatoOutGPS
                CALL   EnviaDatoGPS
                MOVLW          0x03            ; Byte <ETX> = 0x03
                MOVWF          DatoOutGPS
                CALL   EnviaDatoGPS
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; EnviaDatoGPS.- Envía el byte 'DatoOutGPS' por el pin I/O usado como TX del puerto serie para
;comunicación con el GPS. La transmisión se realiza con 1 bit de start, 8 de datos, 1 de paridad (impar) y
;1 de stop, a la velocidad determinada por la constante 'CteRetSer' (ver RELOJ.H).
;-------------------------------------------------------------------------------------------------------------------
EnviaDatoGPS
                BCF    STATUS,RP0              ; Banco 0
                MOVF   DatoOutGPS, W           ; Calcula el bit de paridad (paridad impar)
                CALL   ChkParidad
                BCF    PORTB, TXaGPS           ; Bit de START a 0
                MOVF   DatoOutGPS, W           ; Mueve el byte que se enviará al registro de
                MOVWF          RegTransm       ;transmisión
                MOVLW          .8              ; Inicializa el contador de bits transmitidos
                MOVWF          ContBitsTX
                NOP                            ; Ciclo que faltaba en el bit de START
OtroBit
                CALL   DelaySer                ; El primer Delay es del bit de START
                RRF    RegTransm, F            ; Envía primero el LSB
                BTFSC  STATUS, C
                BSF    PORTB, TXaGPS
                BTFSS  STATUS, C
                BCF    PORTB, TXaGPS
                DECFSZ         ContBitsTX, F
                GOTO   OtroBit
                CALL   DelaySer                ; Delay del último bit de datos
                NOP                            ; Faltan tres ciclos
                NOP
                NOP
```

```
                BTFSS  Flag, Paridad            ; Testea el bit de paridad
                BCF    PORTB, TXaGPS            ; Pone la salida a 0 o a 1 en función del valor del bit
                BTFSC  Flag, Paridad            ;de paridad
                BSF    PORTB, TXaGPS
                CALL   DelaySer
                NOP                             ; Faltan 7 ciclos
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP

                BSF    PORTB, TXaGPS            ; Bit de STOP a 1
                CALL   DelaySer                 ; Delay del bit de STOP
                NOP                             ; Faltan 10 ciclos
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                RETURN
;
;-----------------------------------------------------------------------------------------------------------------------
; ChkParidad.- Calcula el bit de paridad (con paridad impar) del byte introducido en W. El resultado se
;mete en el bit 'Paridad' de la variable 'Flag'.
;-----------------------------------------------------------------------------------------------------------------------
ChkParidad
                BSF    Flag, Paridad            ; Por defecto, el bit de paridad vale 1
                MOVWF          Temp1            ; Temp1 = Byte cuyo bit de paridad queremos
                MOVLW          .8               ;calcular
                MOVWF          Temp2            ; Temp2 = Contador de bits
ChkOtroBit
                RRF    Temp1, F
                BTFSS  STATUS, C
                GOTO   DecContBits              ; Si el bit examinado es 0, no se hace nada
                BTFSS  Flag, Paridad            ; Si el bit examinado es 1, se cambia el estado del bit
                GOTO   BitPar1                  ;de paridad
                BCF    Flag, Paridad
                GOTO   DecContBits
BitPar1
                BSF    Flag, Paridad
DecContBits
                DECFSZ         Temp2, F
                GOTO   ChkOtroBit
                RETURN
;
;-----------------------------------------------------------------------------------------------------------------------
; SendPetTiempo.- Manda un paquete de petición de tiempo al GPS (paquete 0x21).
;-----------------------------------------------------------------------------------------------------------------------
SendPetTiempo
                MOVLW          0x10             ; Byte <DLE> = 0x10
                MOVWF          DatoOutGPS
                CALL   EnviaDatoGPS
                MOVLW          0x21             ; Byte <id> = 0x21
                MOVWF          DatoOutGPS
```

```
                CALL    EnviaDatoGPS
                MOVLW   0x10            ; Byte <DLE> = 0x10
                MOVWF   DatoOutGPS
                CALL    EnviaDatoGPS
                MOVLW   0x03            ; Byte <ETX> = 0x03
                MOVWF   DatoOutGPS
                CALL    EnviaDatoGPS
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; WaitStatPack.- Espera hasta que recibe un paquete de status. Está permanentemente comprobando,
;además, el flag de PPS, de modo que si se activa éste antes de recibirse el paquete de status, sale de la
;función. En caso de recibirse el paquete de status, los dos bytes de datos del mismo se guardan en las
;variables 'Status0' y 'Status1'.
; Devuelve en W: 0 si ha recibido el paquete de status, 1 si se ha activado antes el flag de PPS.
;-------------------------------------------------------------------------------------------------------------------
WaitStatPack
                BTFSC   Flag, PPS               ; Comprueba el flag de PPS
                RETLW.1
                BTFSC   PORTB, RXdeGPS          ; Comprueba la línea de entrada de datos, en busca
                GOTO    WaitStatPack            ;del comienzo del bit de START
                CALL    LeeDatoGPS              ; El dato leído está en 'DatoInGPS'
                MOVLW   0x10                    ; Si el dato es el byte <DLE> (= 0x10) sigue, en caso
                SUBWF   DatoInGPS, W            ;contrario vuelve al principio
                BTFSS   STATUS, Z
                GOTO    WaitStatPack
WaitStatID
                BTFSC   Flag, PPS               ; Entre cada dos bytes comprueba el flag de PPS
                RETLW.1
                BTFSC   PORTB, RXdeGPS
                GOTO    WaitStatID
                CALL    LeeDatoGPS              ; El dato leído está en 'DatoInGPS'
                MOVLW   0x46                    ; Si el dato es el byte <id> del paquete de statusx46)
                SUBWF   DatoInGPS, W            ;sigue, en caso contrario vuelve al principio
                BTFSS   STATUS, Z
                GOTO    WaitStatPack
WaitStatus0
                BTFSC   Flag, PPS
                RETLW.1
                BTFSC   PORTB, RXdeGPS
                GOTO    WaitStatus0
                CALL    LeeDatoGPS
                MOVF    DatoInGPS, W
                MOVWF   Status0                 ; Pasa el dato leído a 'Status0'
                BCF     Flag, PPSLargo          ; Por defecto, PPS normal (10ms).
                BTFSS   STATUS, Z               ; Si el primer byte de status es distinto de 0x00,
                BSF     Flag, PPSLargo          ; activa el flag de PPS largo (20ms).

                MOVLW   0x10                    ; Si el byte de datos es <DLE> (0x10), tiene que
                SUBWF   Status0, W              ;venir repetido ('data stuffing', ver manual del GPS).
                BTFSS   STATUS, Z               ;En caso contrario es un error de formato y
                                                ;volvemos al principio.
                GOTO    WaitStatus1             ; Si el byte de datos no es <DLE>, seguimos.
                CALL    WaitStuff               ; WaitStuff devuelve 0 si el byte recibido era
                SUBLW.0                         ; <DLE>, 1 en otro caso
                BTFSS   STATUS, Z
                GOTO    WaitStatPack
WaitStatus1
                BTFSC   Flag, PPS
                RETLW.1
```

```
                BTFSC  PORTB, RXdeGPS
                GOTO   WaitStatus1
                CALL   LeeDatoGPS
                MOVF   DatoInGPS, W
                MOVWF          Status1          ; Pasa el dato leído a 'Status1'
                MOVLW          0x10             ; Si el byte de datos es <DLE> hacemos la misma
                SUBWF Status1, W                ;comprobación que para Status0. En caso contrario,
                BTFSS  STATUS, Z                ;seguimos.
                GOTO   WaitStatDLE
                CALL   WaitStuff                ; WaitStuff devuelve 0 si el byte recibido era
                SUBLW.0                         ; <DLE>, 1 en otro caso
                BTFSS  STATUS, Z
                GOTO   WaitStatPack
WaitStatDLE
                BTFSC  Flag, PPS
                RETLW.1
                BTFSC  PORTB, RXdeGPS
                GOTO   WaitStatDLE
                CALL   LeeDatoGPS
                MOVLW          0x10             ; Si el dato es el bit <DLE> (= 0x10) sigue, en caso
                SUBWF DatoInGPS, W              ;contrario vuelve al principio
                BTFSS  STATUS, Z
                GOTO   WaitStatPack
WaitStatETX
                BTFSC  Flag, PPS
                RETLW.1
                BTFSC  PORTB, RXdeGPS
                GOTO   WaitStatETX
                CALL   LeeDatoGPS
                MOVLW          0x03             ; Si el dato es el bit <ETX> (= 0x03) sigue, en caso
                SUBWF DatoInGPS, W              ;contrario vuelve al principio
                BTFSS  STATUS, Z
                GOTO   WaitStatPack
                RETLW.0                         ; Se ha recibido el paquete de status completo
;
;-------------------------------------------------------------------------------------------------------------------------
; LeeDatoGPS.- Lee un dato serie por la línea RXdeGPS (RB2). En el programa se llama a esta función
cuando ya se ha detectado el bit de START, por lo que no se comprueba aquí el estado de la línea de
entrada. El dato leído se mete en la variable 'DatoInGPS'.
;-------------------------------------------------------------------------------------------------------------------------
LeeDatoGPS
                MOVLW          .8
                MOVWF          Temp1            ; Temp1 = Contador de bits
                CLRF   DatoInGPS
                CALL   DelayHalfBit     ; Delay1/2bit = Retraso de medio bit, para que la lectura
LeeOtroBit                              ;se realice aproximadamente en el punto medio de cada bit
                CALL   Delay1Bit                ; Delay1Bit = Retraso de un bit
                BCF    STATUS, C
                RRF    DatoInGPS, F
                BTFSC  PORTB, RXdeGPS
                BSF    DatoInGPS, 7             ; Se recibe primero el LSbit, por lo que se pone a 1
                DECFSZ         Temp1, F         ;(si la línea serie está en alto) el MSbit y luego se
                GOTO   LeeOtroBit               ;rota.
                CALL   Delay1Bit
                BCF    STATUS, C
                BCF    Flag, Paridad            ; Aunque no usamos para nada el bit de paridad, lo
                BTFSC  PORTB, RXdeGPS           ;metemos en el bit 'Paridad' de Flag
                BSF    Flag, Paridad
                CALL   Delay1Bit
```

```
WaitBitSTOP
                BTFSS  PORTB, RXdeGPS       ; Antes de salir, comprobamos que la línea está a 1
                GOTO   WaitBitSTOP          ;(bit de STOP)
                RETURN
;
;-----------------------------------------------------------------------------------------------------------------------
; WaitStuff.- Comprueba si los datos con valor <DLE> (0x10) vienen repetidos ('stuffed'). Para ello lee un
;carácter y comprueba si es <DLE> (0x10). A la vez que hace polling a la línea serie comprueba si ha
;llegado un nuevo PPS, en cuyo caso se sale.
; Devuelve en W: 0 si detecta un carácter y lo identifica como <DLE>, 1 en caso contrario (no lo
;identifica como <DLE> o llega un PPS antes de detectar un carácter).
;-----------------------------------------------------------------------------------------------------------------------
WaitStuff
                BTFSC  Flag, PPS
                RETLW.1                     ; Ha llegado un PPS antes que el bit de START
                BTFSC  PORTB, RXdeGPS
                GOTO   WaitStuff
                CALL   LeeDatoGPS
                MOVLW          0x10
                SUBWFDatoInGPS, W
                BTFSS  STATUS, Z
                RETLW.1                     ; El byte leído no era <DLE>
                RETLW.0                     ; El byte leído era <DLE>
;
;-----------------------------------------------------------------------------------------------------------------------
; WaitTimePack.- Espera hasta que recibe un paquete de tiempo. Está permanentemente comprobando,
;además, el flag de PPS o de SerParOK, de modo que si se activa éste antes de recibirse el paquete de
;tiempo, sale de la función. En caso de recibirse el paquete de tiempo, los diez bytes de datos del mismo
;se guardan en las variables 'Time0' a 'Time9'.
; Devuelve en W: 0 si ha recibido el paquete de tiempo, 1 si se ha activado antes el flag de PPS.
;-----------------------------------------------------------------------------------------------------------------------
WaitTimePack
                BTFSC  Flag, PPS            ; Comprueba el flag de PPS
                RETLW.1
                BTFSC  PORTB, RXdeGPS       ; Comprueba la línea de entrada de datos, en busca
                GOTO   WaitTimePack         ;del comienzo del bit de START
                CALL   LeeDatoGPS           ; El dato leído está en 'DatoInGPS'
                MOVLW          0x10         ; Si el dato es el byte <DLE> (= 0x10) sigue, en caso
                SUBWFDatoInGPS, W           ;contrario vuelve al principio
                BTFSS  STATUS, Z
                GOTO   WaitTimePack
WaitTimeID
                BTFSC  Flag, PPS            ; Entre cada dos bytes comprueba el flag de PPS
                RETLW.1
                BTFSC  PORTB, RXdeGPS
                GOTO   WaitTimeID
                CALL   LeeDatoGPS           ; El dato leído está en 'DatoInGPS'
                MOVLW          0x41         ; Si el dato es el byte <id> del paquete de tiempo
                SUBWFDatoInGPS, W           ;(0x41) sigue, en caso contrario vuelve al principio
                BTFSS  STATUS, Z
                GOTO   WaitTimePack
WaitTime0
                BTFSC  Flag, PPS
                RETLW.1
                BTFSC  PORTB, RXdeGPS
                GOTO   WaitTime0
                CALL   LeeDatoGPS
                MOVF   DatoInGPS, W
                MOVWF          Time0        ; Pasa el dato leído a 'Status0'
                MOVLW          0x10         ; Si el byte de datos es <DLE> (0x10), tiene que
```

```
                SUBWF   Time0, W        ;venir repetido ('data stuffing', ver manual del GPS).
                BTFSS   STATUS, Z       ;En caso contrario es un error de formato y
                                        ;volvemos al principio.
                GOTO    WaitTime1       ; Si el byte de datos no es <DLE>, seguimos.
                CALL    WaitStuff       ; WaitStuff devuelve 1 si el byte recibido era
                SUBLW.0                 ; <DLE>, 0 en otro caso
                BTFSS   STATUS, Z
                GOTO    WaitTimePack
WaitTime1                               ; Repetimos el proceso para cada uno de los diez
                BTFSC   Flag, PPS       ;bytes de datos del proceso.
                RETLW.1
                BTFSC   PORTB, RXdeGPS
                GOTO    WaitTime1
                CALL    LeeDatoGPS
                MOVF    DatoInGPS, W
                MOVWF           Time1
                MOVLW           0x10
                SUBWF   Time1, W
                BTFSS   STATUS, Z
                GOTO    WaitTime2
                CALL    WaitStuff
                SUBLW.0
                BTFSS   STATUS, Z
                GOTO    WaitTimePack
WaitTime2
                BTFSC   Flag, PPS
                RETLW.1
                BTFSC   PORTB, RXdeGPS
                GOTO    WaitTime2
                CALL    LeeDatoGPS
                MOVF    DatoInGPS, W
                MOVWF           Time2
                MOVLW           0x10
                SUBWF   Time2, W
                BTFSS   STATUS, Z
                GOTO    WaitTime3
                CALL    WaitStuff
                SUBLW.0
                BTFSS   STATUS, Z
                GOTO    WaitTimePack
WaitTime3
                BTFSC   Flag, PPS
                RETLW.1
                BTFSC   PORTB, RXdeGPS
                GOTO    WaitTime3
                CALL    LeeDatoGPS
                MOVF    DatoInGPS, W
                MOVWF           Time3
                MOVLW           0x10
                SUBWF   Time3, W
                BTFSS   STATUS, Z
                GOTO    WaitTime4
                CALL    WaitStuff
                SUBLW.0
                BTFSS   STATUS, Z
                GOTO    WaitTimePack
WaitTime4
                BTFSC   Flag, PPS
                RETLW.1
                BTFSC   PORTB, RXdeGPS
```

```
                GOTO   WaitTime4
                CALL   LeeDatoGPS
                MOVF   DatoInGPS, W
                MOVWF          Time4
                MOVLW          0x10
                SUBWFTime4, W
                BTFSS  STATUS, Z
                GOTO   WaitTime5
                CALL   WaitStuff
                SUBLW.0
                BTFSS  STATUS, Z
                GOTO   WaitTimePack
WaitTime5
                BTFSC  Flag, PPS
                RETLW.1
                BTFSC  PORTB, RXdeGPS
                GOTO   WaitTime5
                CALL   LeeDatoGPS
                MOVF   DatoInGPS, W
                MOVWF          Time5
                MOVLW          0x10
                SUBWFTime5, W
                BTFSS  STATUS, Z
                GOTO   WaitTime6
                CALL   WaitStuff
                SUBLW.0
                BTFSS  STATUS, Z
                GOTO   WaitTimePack
WaitTime6
                BTFSC  Flag, PPS
                RETLW.1
                BTFSC  PORTB, RXdeGPS
                GOTO   WaitTime6
                CALL   LeeDatoGPS
                MOVF   DatoInGPS, W
                MOVWF          Time6
                MOVLW          0x10
                SUBWFTime6, W
                BTFSS  STATUS, Z
                GOTO   WaitTime7
                CALL   WaitStuff
                SUBLW.0
                BTFSS  STATUS, Z
                GOTO   WaitTimePack
WaitTime7
                BTFSC  Flag, PPS
                RETLW.1
                BTFSC  PORTB, RXdeGPS
                GOTO   WaitTime7
                CALL   LeeDatoGPS
                MOVF   DatoInGPS, W
                MOVWF          Time7
                MOVLW          0x10
                SUBWFTime7, W
                BTFSS  STATUS, Z
                GOTO   WaitTime8
                CALL   WaitStuff
                SUBLW.0
                BTFSS  STATUS, Z
                GOTO   WaitTimePack
```

```
WaitTime8
            BTFSC  Flag, PPS
            RETLW.1
            BTFSC  PORTB, RXdeGPS
            GOTO   WaitTime8
            CALL   LeeDatoGPS
            MOVF   DatoInGPS, W
            MOVWF        Time8
            MOVLW        0x10
            SUBWFTime8, W
            BTFSS  STATUS, Z
            GOTO   WaitTime9
            CALL   WaitStuff
            SUBLW.0
            BTFSS  STATUS, Z
            GOTO   WaitTimePack
WaitTime9
            BTFSC  Flag, PPS
            RETLW.1
            BTFSC  PORTB, RXdeGPS
            GOTO   WaitTime9
            CALL   LeeDatoGPS
            MOVF   DatoInGPS, W
            MOVWF        Time9
            MOVLW        0x10
            SUBWFTime9, W
            BTFSS  STATUS, Z
            GOTO   WaitTimeDLE
            CALL   WaitStuff
            SUBLW.0
            BTFSS  STATUS, Z
            GOTO   WaitTimePack
WaitTimeDLE
            BTFSC  Flag, PPS
            RETLW.1
            BTFSC  PORTB, RXdeGPS
            GOTO   WaitTimeDLE
            CALL   LeeDatoGPS
            MOVLW        0x10          ; Si el dato es el bit <DLE> (0x10) sigue, en caso
            SUBWFDatoInGPS, W          ;contrario vuelve al principio
            BTFSS  STATUS, Z
            GOTO   WaitTimePack
WaitETX
            BTFSC  Flag, PPS
            RETLW.1
            BTFSC  PORTB, RXdeGPS
            GOTO   WaitETX
            CALL   LeeDatoGPS
            MOVLW        0x03          ; Si el dato es el bit <ETX> (0x03) sigue, en caso
            SUBWFDatoInGPS, W          ;contrario vuelve al principio
            BTFSS  STATUS, Z
            GOTO   WaitTimePack
            RETLW.0                    ; Se ha recibido el paquete de tiempo completo
```

```
;
;-------------------------------------------------------------------------------------------------------------------
; Delay1Bit.- Genera un retardo de un bit para la recepción serie. Para velocidad de transmisión de
;9600bps, con cristal de 10MHz:
;       Número de ciclos = tBit/tciclo = (1/9600)/(4/10MHz) = 260
; En la función Delay1Bit, número de ciclos = 3*Nbucles+4 (sin contar el CALL). Además, dentro de la
;función 'LeeDatoGPS' hay 11ciclos por bit, por lo que el valor inicial que hay que darle a ContRetardo
;es:
;       260 = 3*NBucles+4+11 -> NBucles = 82.
;-------------------------------------------------------------------------------------------------------------------
Delay1Bit
                MOVLW       .82
                MOVWF       ContRetardo
SigueDelay1Bit
                DECFSZ      ContRetardo, F
                GOTO   SigueDelay1Bit
                NOP
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; DelayHalfBit.- Genera un retardo de medio bit para la recepción serie. Para velocidad de transmisión de
;9600bps, con cristal de 10MHz:
;       Número de ciclos = (tBit/2)/tciclo = (1/2*9600)/(4/10MHz) = 130
; En la función DelayHalfBit, número de ciclos = 3*Nbucles+4 (sin contar el CALL). Además, desde que
;se detecta el bit de START pasan un mínimo de 9 ciclos de reloj, por lo que el valor inicial que hay que
;darle a ContRetardo es:
;       130 = 3*NBucles+4+9 -> NBucles = 39.
;-------------------------------------------------------------------------------------------------------------------
DelayHalfBit
                MOVLW       .39
                MOVWF       ContRetardo
SigueDelayHalf
                DECFSZ      ContRetardo, F
                GOTO   SigueDelayHalf
                NOP
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; DelaySer.- Genera el retardo necesario para la transmisión serie. Para velocidad de transmisión de
;9600bps:
; Para cristal de 3.579525MHz, Número de ciclos = tBit/tciclo = (1/9600)/(4/3.579525MHz) = 93
; Para cristal de 10MHz, Número de ciclos = tBit/tciclo = (1/9600)/(4/10MHz) = 260
; En la función DelaySer, número de ciclos = 3*Nbucles+4
; Además, dentro de la función 'EnviarByte' hay 11ciclos por bit, por lo que el valor que hay que darle a
;CteRetardo es:
; Cristal de 3.579525MHz: 93 = 3*NBucles+4+11 -> NBucles = 26.
; Cristal de 10MHz:      260 = 3*NBucles+4+11 -> NBucles = 82.
; Estos valores se asignan a CteRetSer en el fichero RELOJ.H.
;-------------------------------------------------------------------------------------------------------------------
DelaySer
                MOVLW       CteRetSer
                MOVWF       ContRetardo
Sigue
                DECFSZ      ContRetardo, F
                GOTO   Sigue
                NOP
                RETURN
```

```
;
;-------------------------------------------------------------------------------------------------------------------
; Retardo1ms.- Genera un retardo de aproximadamente 1ms (con cristal de 10MHz) entre cada dos pulsos
;de petición de parámetros, para estar seguros de que el PC está esperando.
; Número de ciclos = 1ms/(0.4us/ciclo) = 2500ciclos. Con 10 ciclos por bucle:
; Número de bucles = 2500ciclos/(10ciclos/bucle) = 250bucles
;-------------------------------------------------------------------------------------------------------------------
Retardo1ms
                MOVLW       .250
                MOVWF       Temp1
BucleRet
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                DECFSZ      Temp1, F
                GOTO   BucleRet
                RETURN
```

```
;*******************************************************************
; Programa principal e interrupciones
;*******************************************************************
;-------------------------------------------------------------------------------------------------------------------
; Rutina de servicio de la interrupción
;-------------------------------------------------------------------------------------------------------------------
                ORG     ISR_V                   ; Vector de interrupcion

                BCF     STATUS, RP0             ; Banco 0
                BTFSC   INTCON, INTF            ; Comprueba si es la interrupción externa (PPS)
                CALL    GestIntPPS
                BTFSC   PIR1, RCIF              ; Comprueba si es la int. de USART (carácter
                CALL    GestIntUSART            ;recibido)
                BTFSC   PIR1, TMR1IF            ; Comprueba si es la int. del timer 1
                CALL    GestIntTmr1
                BCF     INTCON, RBIF            ; Antes de salir limpia todos los flags de
                BCF     INTCON, INTF            ;interrupción, independientemente de cual se haya
                BCF     INTCON, T0IF            ;servido
                CLRF    PIR1
                CLRF    PIR2
                RETFIE
;
;-------------------------------------------------------------------------------------------------------------------
; GestIntPPS.- Función para la gestión de la interrupción externa, disparada por los PPS del GPS. Alarga
;el pulso de segundo y enciende el LED.
; Devuelve en W:
; 0 si se ha realizado la gestión normal del PPS (alargar el pulso de segundo y encender el LED).
; 1 si se está gestionando un comando de reset (se está inhibiendo la salida de PPS).
; 2 si se ha terminado la gestión de un comando de reset.
;-------------------------------------------------------------------------------------------------------------------
GestIntPPS
                CLRWDT                          ; Si no se reciben PPS, se resetea.
                BTFSC   Flag, Reset             ; Comprueba el flag de reset y, si está activado,
                GOTO    GestReset               ;inhibe la salida de PPS durante treinta segundos
                BCF     PORTC, OutPPS           ; Flanco descendente inicial de la salida PPS
                BSF     PORTD, LEDPPS           ; Enciende el LED de PPS
                BSF     Flag, PPS               ; Activa el flag de PPS recibido
                CALL    ProgramaTmr1            ; Programa el timer con la duración del pulso
                RETLW.0
GestReset
                DECFSZ          ContReset, F    ; El contador de reset se inicializa al recibir el
                RETLW.1                         ;carácter de reset del sistema (función
                BCF     Flag, Reset             ; 'GestIntUSART').
                RETLW.2
;
;-------------------------------------------------------------------------------------------------------------------
; ProgramaTmr1.- Programa el timer 1 para que produzca una interrupción después de 10ms. Teniendo en
;cuenta que el Tmr1 se incrementa con cada ciclo de instrucción, las constantes de inicialización deben
;ser (para cristal de 10MHz):
; 10ms => 10ms/(0.4us/ciclo) = 25000ciclos => hay que inicializar TMR1H:TMR1L a 65536-25000 =
;40536 (0x9E58).
; Para pulsos largos (20 ms):
; 20ms => 20ms/(0.4us/ciclo) = 50000ciclos => hay que inicializar TMR1H:TMR1L a 65536-50000 =
;15536 (0x3CB0).
; En este caso no tenemos en cuenta las instrucciones previas a la activación del timer, porque no es
;necesario ser muy precisos.
;-------------------------------------------------------------------------------------------------------------------
ProgramaTmr1
                BSF     STATUS, RP0             ; Banco 1
                BCF     PIE1, TMR1IE            ; Deshabilita la interrupción de Tmr1
```

```
                BCF     STATUS, RP0             ; Banco 0
                BCF     T1CON, TMR1IE           ; Deshabilita el Tmr1

                BTFSS   Flag, PPSLargo
                GOTO    PPSNormal
                MOVLW           0x3C            ; 15536 = 0x3CB0
                MOVWF           TMR1H
                MOVLW           0xB0
                MOVWF           TMR1L
                GOTO    ActivaTmr1
PPSNormal
                MOVLW           0x9E            ; 40536 = 0x9E58
                MOVWF           TMR1H
                MOVLW           0x58
                MOVWF           TMR1L
ActivaTmr1
                MOVLW           0x01            ; 0x01 => PreScaler a 1:1, oscilador deshabilitado,
                MOVWF           T1CON           ;timer 1 habilitado en modo timer (reloj interno).
                BSF     STATUS, RP0             ; Banco 1
                BSF     PIE1, TMR1IE            ; Habilita la interrupción de Tmr1
                BCF     STATUS, RP0             ; Banco 0
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------------
; GestIntUSART.- Función para la gestión de la interrupción del puerto serie, que se dispara cuando hay
;un carácter en el registro de recepción RCREG. La función lee el carácter recibido y lo guarda en
;'DatoInSP'. Además activa el flag correspondiente ('SerParWait' si era el carácter indicador de que SerPar
;está a la espera de datos, o 'Reset' si era el carácter de reset del sistema). Si no es ninguno de los
;caracteres permitidos sale con un código de error en W.
; Devuelve en W:
; 0 si no ha habido error (carácter leído e identificado correctamente)
; 1 si ha habido error de identificación (carácter leído correctamente pero no identificado)
; 2 si ha habido error de trama (bit de STOP a 0)
; 3 si ha habido error de desbordamiento del buffer de recepción
;-------------------------------------------------------------------------------------------------------------------------
GestIntUSART
                BCF     STATUS, RP0             ; Banco 0
                BTFSS   RCSTA, FERR             ; Comprueba el bit de error de trama
                GOTO    CheckOERR
                MOVF    RCREG, W        ; Lee RCREG para limpiar el error, descarta el byte leído
                RETLW.2                         ; Error de trama
CheckOERR
                BTFSS   RCSTA, OERR             ; Comprueba el bit de error de desbordamiento
                GOTO    NoError
                BCF     RCSTA, CREN             ; Limpia el flag de error reseteando el receptor
                NOP
                BSF     RCSTA, CREN
                RETLW.3                         ; Error de desbordamiento del búffer de recepción
NoError
                MOVF    RCREG, W                ; Lee el byte recibido
                MOVWF           DatoInSP        ; Si es el byte 'SerParOK', manda la trama
                SUBLWSPWaitData                 ;correspondiente al módulo SerPar. Si no,
                BTFSS   STATUS, Z               ;comprueba si es el carácter de Reset.
                GOTO    CheckReset
                BSF     Flag, SerParWait
                RETLW.0                         ; No ha habido error
CheckReset
                MOVF    DatoInSP, W
                SUBLWSysReset
                BTFSS   STATUS, Z
```

```
                RETLW.1                 ; Error, no es ningún carácter permitido
                BSF     Flag, Reset     ; Si es el carácter de reset del sistema, activamos el
                MOVLW       .30         ;flag correspondiente e inicializamos el contador al
                MOVWF       ContReset   ;número de segundos durante los que se inhibirá la
                                        ;salida de PPS.
                RETLW.0                 ; No ha habido error
;
;-------------------------------------------------------------------------------------------------------------------
; GestIntTmr1.- Función para la gestión de la interrupción de saturación del timer 1. Deshabilita el timer
;1, apaga el LED y pone a 1 la salida de PPS.
;-------------------------------------------------------------------------------------------------------------------
GestIntTmr1
                BSF     PORTC, OutPPS   ; Flanco ascendente final de la salida PPS
                BCF     PORTD, LEDPPS   ; Apaga el LED de PPS
                BCF     T1CON, TMR1IE   ; Deshabilita el Tmr1, para evitar que produzca más
                RETURN                  ;interrupciones antes del siguiente PPS.
```

```
;
;-------------------------------------------------------------------------------------------------------------------
; Programa principal
;-------------------------------------------------------------------------------------------------------------------
                ORG     RESET_V
                GOTO    Inicio
;
                ORG     0x750
Inicio
;
; Inicializaciones
*************************************************************************************
                ; Inicializaciones: pines I/O, variables, interrupciones, paquetes de inicio al GPS
                CALL    Inicializacion
;
; Bucle de lectura
*************************************************************************************
                ; 1. Bucles de espera: se espera primero un PPS y luego un carácter de petición de datos
                ;de SerPar. Si llegan dos PPS seguidos antes de un carácter de petición, los datos dejan
                ;de ser válidos. Si lo que llega es un carácter de reset, la gestión se realiza
                ;exclusivamente por interrupciones.
EsperaPPS
                BTFSS   Flag, PPS               ; Espera un PPS. El flag de PPS se activa en la
                GOTO    EsperaPPS               ;interrupción externa.
                BCF     Flag, PPS
EsperaSerParOK
                BTFSS   Flag, SerParWait        ; Espera un carácter de petición de datos de SerPar.
                GOTO    EsperaSerParOK          ; El flag de SerParWait se activa en la interrupción
                BCF     Flag, SerParWait        ;de recepción de datos, cuando se recibe un carácter
                                                ;de 'SerPar a la espera de datos'.
                BTFSC   Flag, PPS               ; Si el siguiente PPS llega antes que el carácter de
                BCF     Flag, DataOK            ;petición de datos, los datos dejan de ser válidos.
                BTFSC   Flag, PPS
                BCF     Flag, PPS

                ; 2. Manda la cadena de datos al módulo SerPar:
                CALL    MandarDatos

                ; 3. Manda paquete de petición de status al GPS y espera la respuesta:
                CALL    SendPetStatus           ; Manda petición de status al GPS
                CALL    WaitStatPack            ; Espera paquete de status, comprobando además el
                                                ;flag de PPS
                SUBLW.0                         ; Si 'WaitStatPack' devuelve 0 en w, tenemos el
                BTFSS   STATUS, Z               ;paquete de status y pedimos el de tiempo. Si
                GOTO    EsperaPPS               ;devuelve 1, el siguiente PPS ha llegado antes que la
                                                ;respuesta del GPS, por lo que se enviarán bytes a 0.

                ; 4. Manda paquete de petición de tiempo al GPS y espera la respuesta:
                CALL    SendPetTiempo           ; Manda petición de tiempo al GPS
                CALL    WaitTimePack            ; Espera paquete de tiempo, comprobando además el
                                                ;flag de PPS
                SUBLW.0                         ; Si 'WaitTimePack' devuelve 0 en w, tenemos el
                BTFSC   STATUS, Z               ;paquete de tiempo y activamos el flag de datos
                BSF     Flag, DataOK            ;listos. Si devuelve 1, el siguiente PPS ha llegado
                GOTO    EsperaPPS               ;antes que la respuesta del GPS, por lo que se
                                                ;enviarán bytes a 0
                END
```

Anexo II.B:

Ficheros de circuito impreso de la antena del Vesubio

ANEXO II.B: FICHEROS DE CIRCUITO IMPRESO DE LA ANTENA DEL VESUBIO

a)

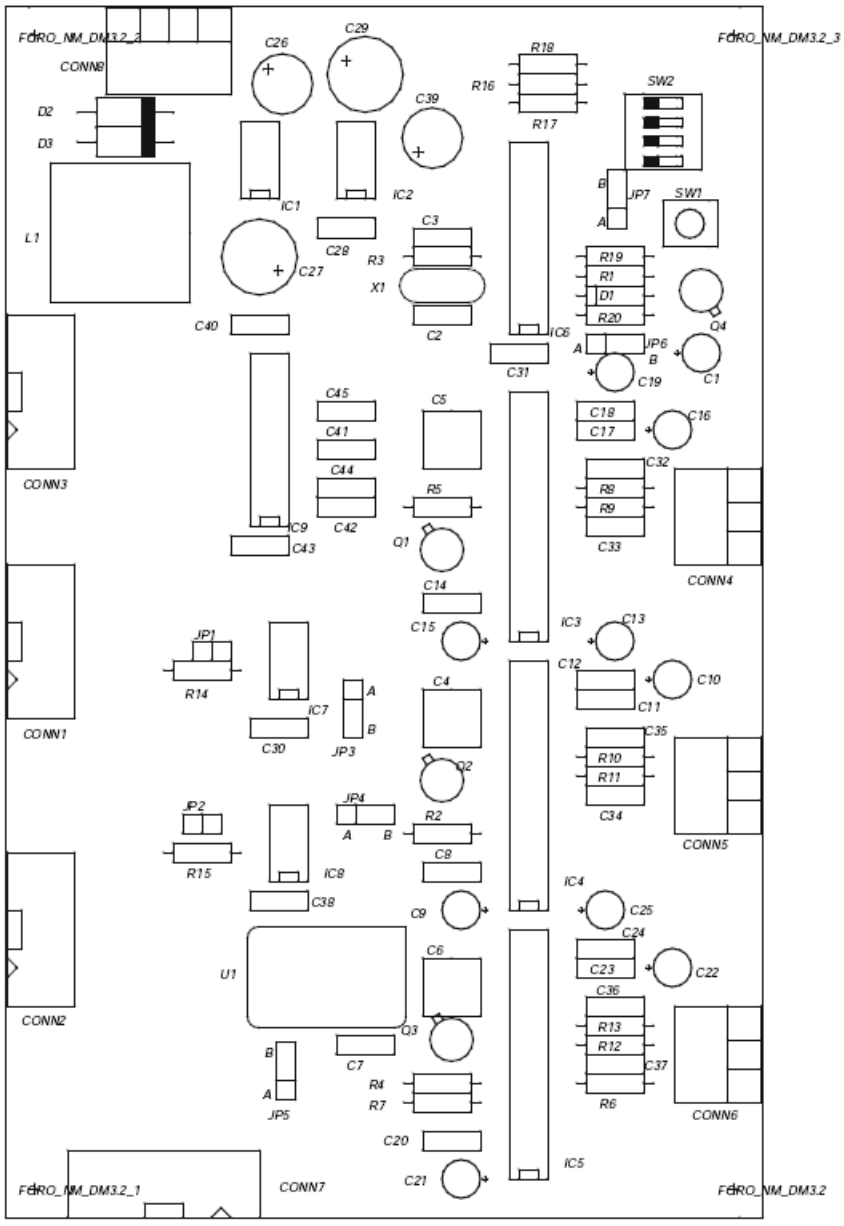
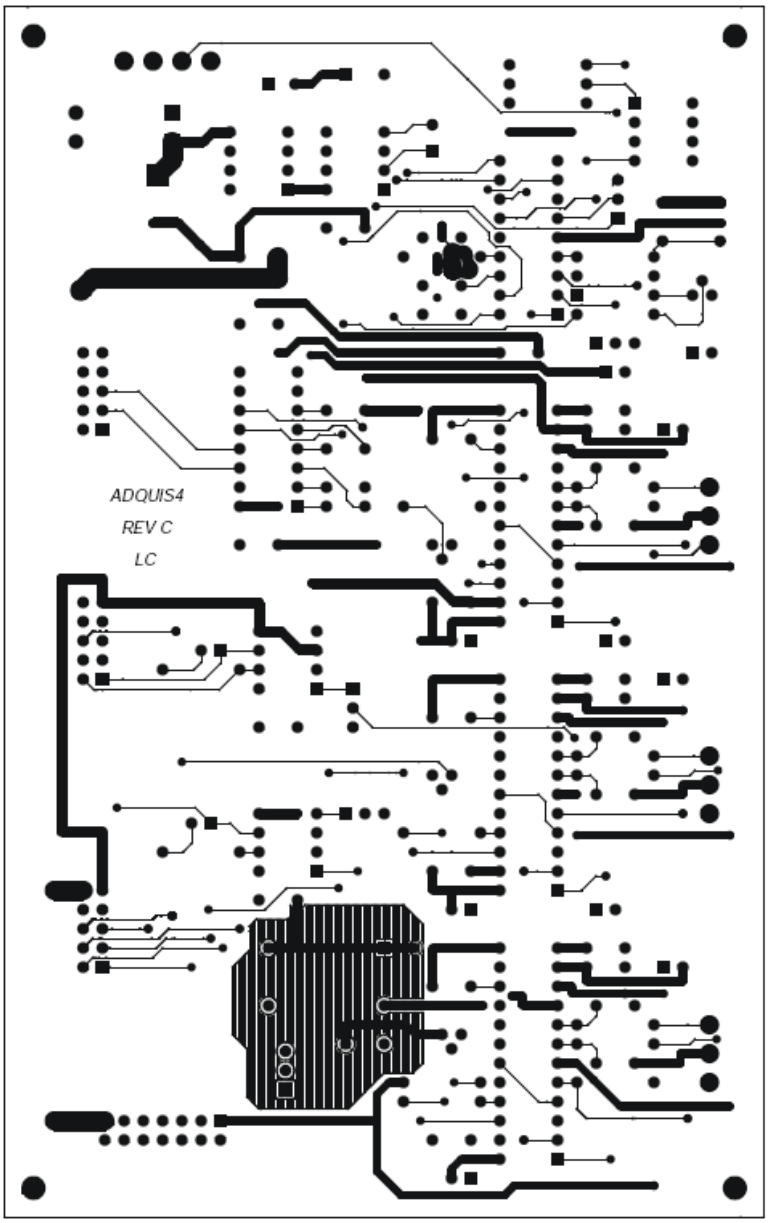
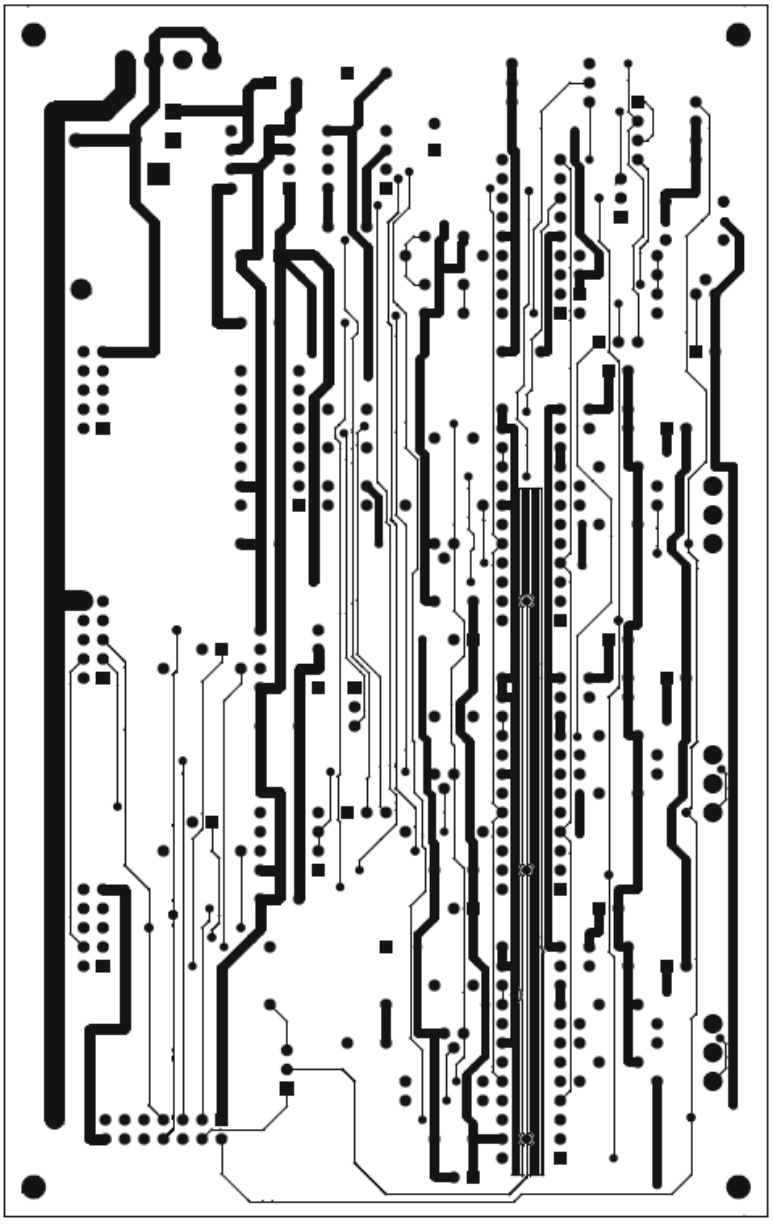


Figura II.1. *Placa de conversión A/D de la antena del Vesubio (esquema en figura 4.12 de la memoria): distribución de componentes (a), cara superior (b) y cara inferior (c). El diseño de esta tarjeta, así como la del circuito PLL y las de los módulos de memoria y SerPar, fue realizado por una empresa especializada a partir de los esquemas eléctricos de las figuras 4.12 a 4.16 de la memoria.*

II.1.b)



II.1.c)



a)

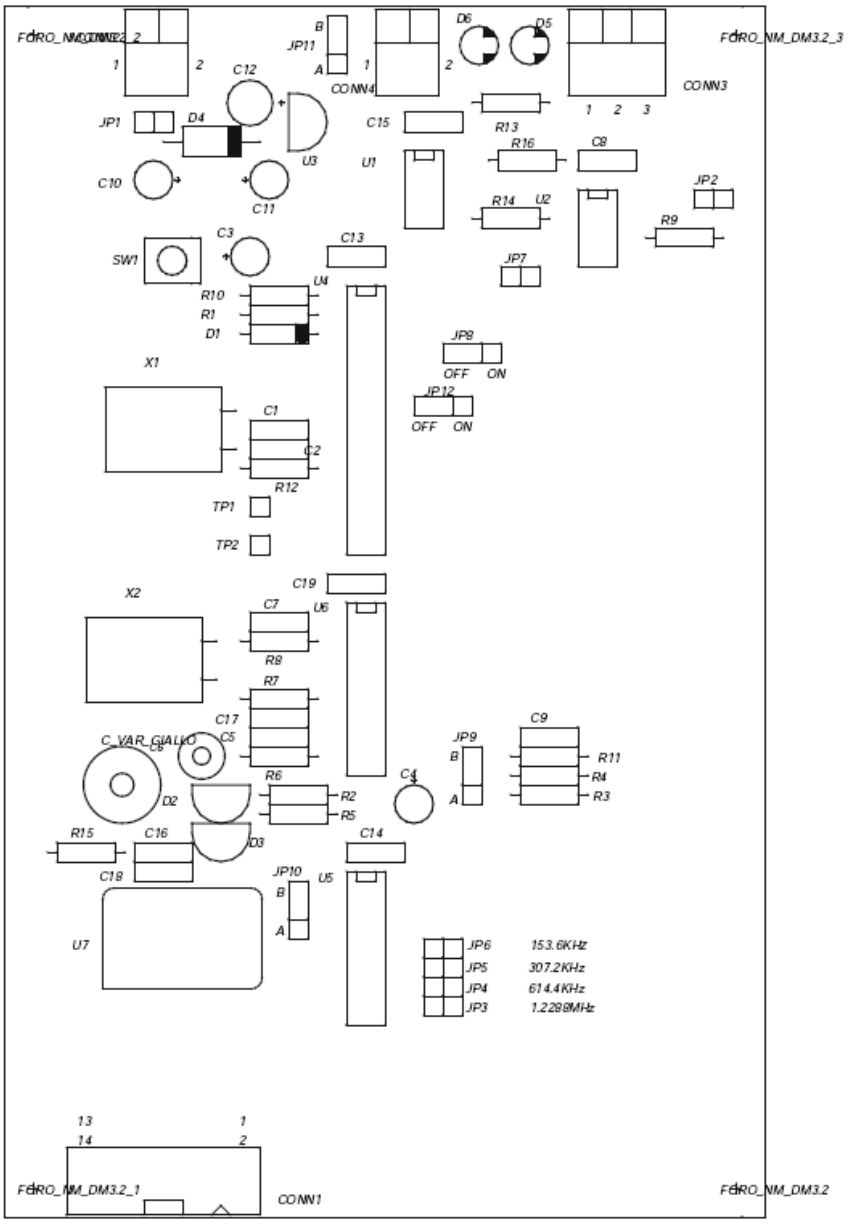
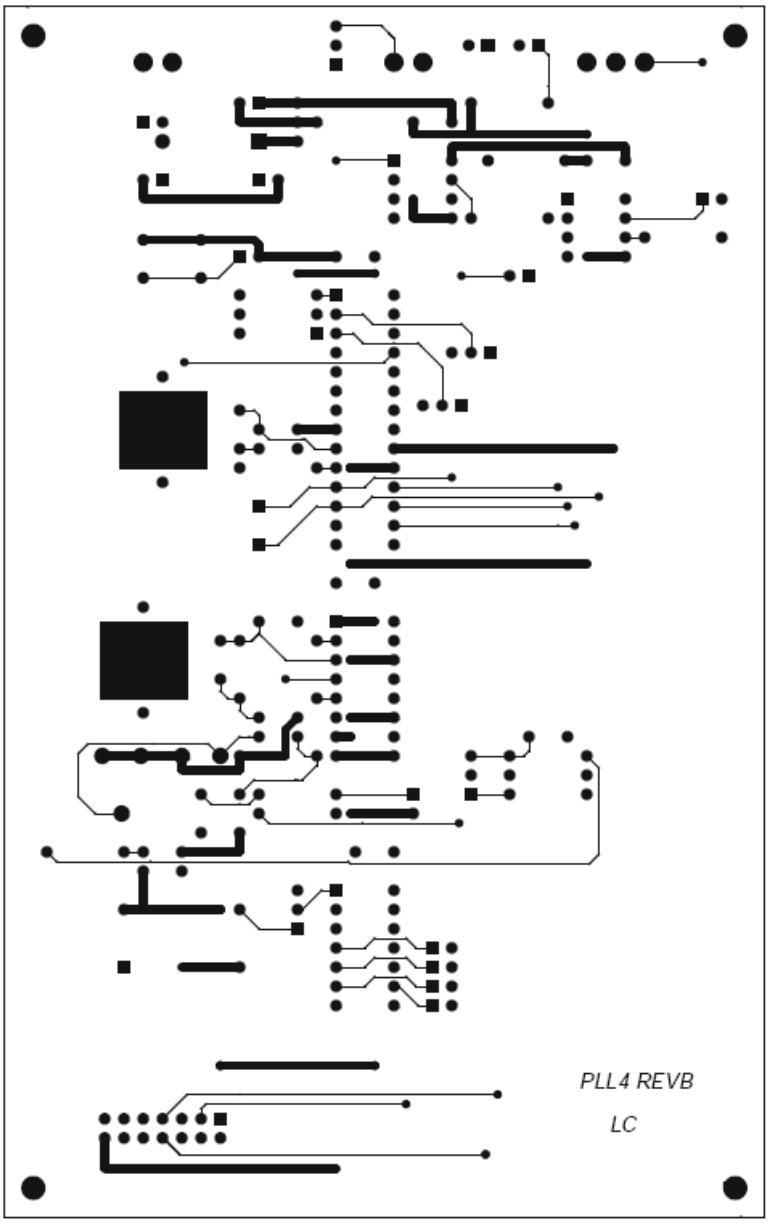
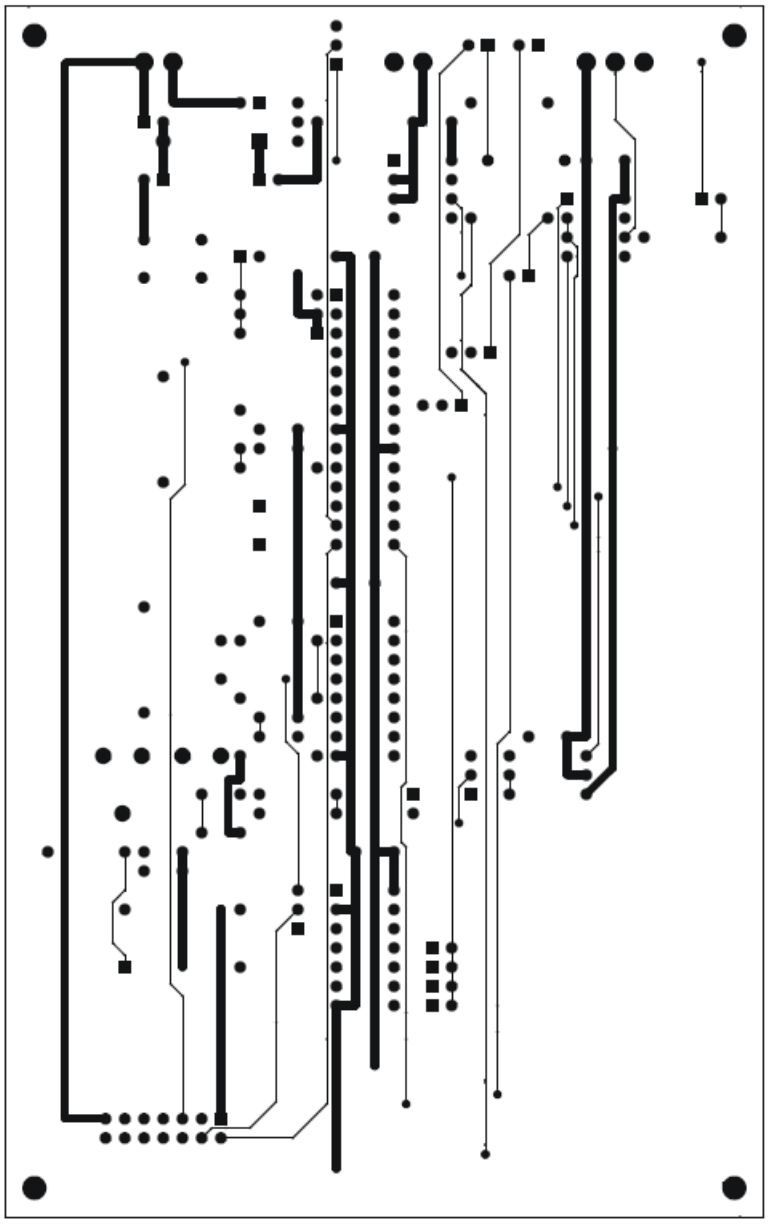


Figura II.2. *Placa del circuito PLL de la antena del Vesubio (esquema en figura 4.13 de la memoria): distribución de componentes (a), cara superior (b) y cara inferior (c).*

II.2.b)



II.2.c)



a)

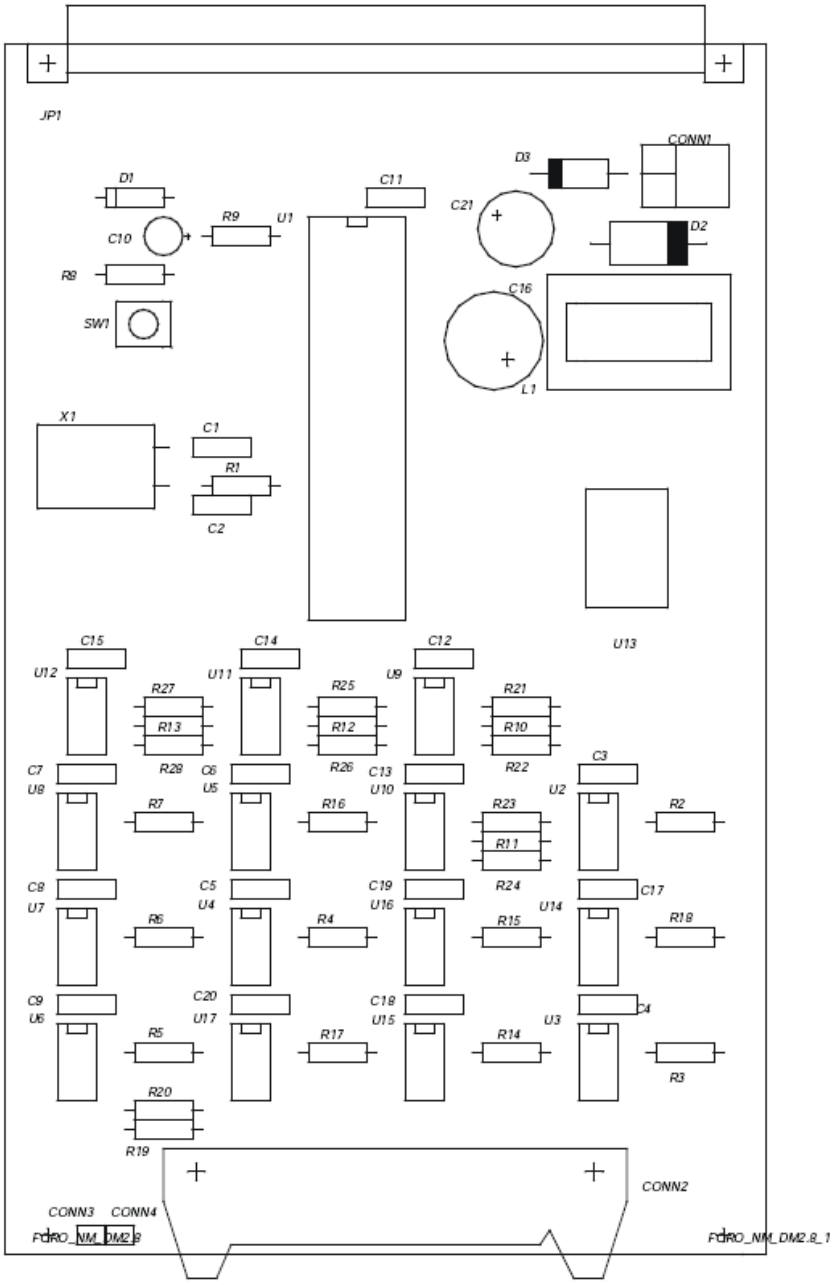
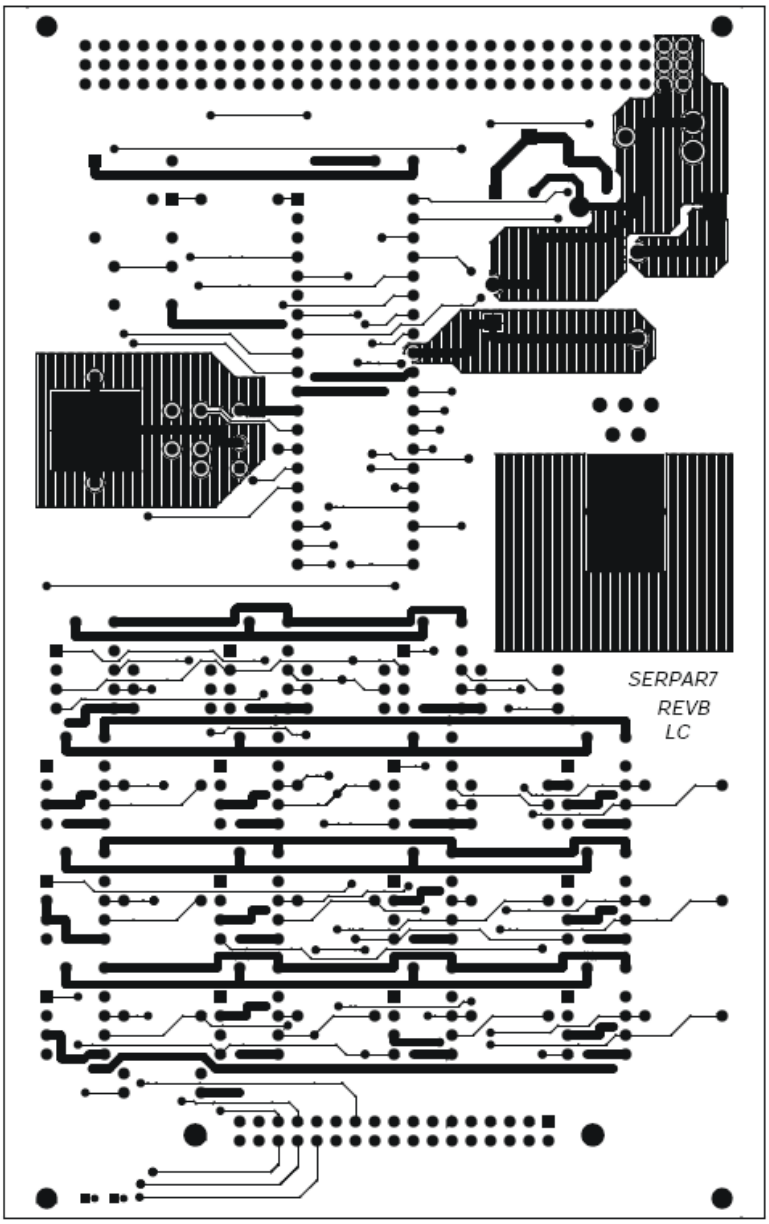
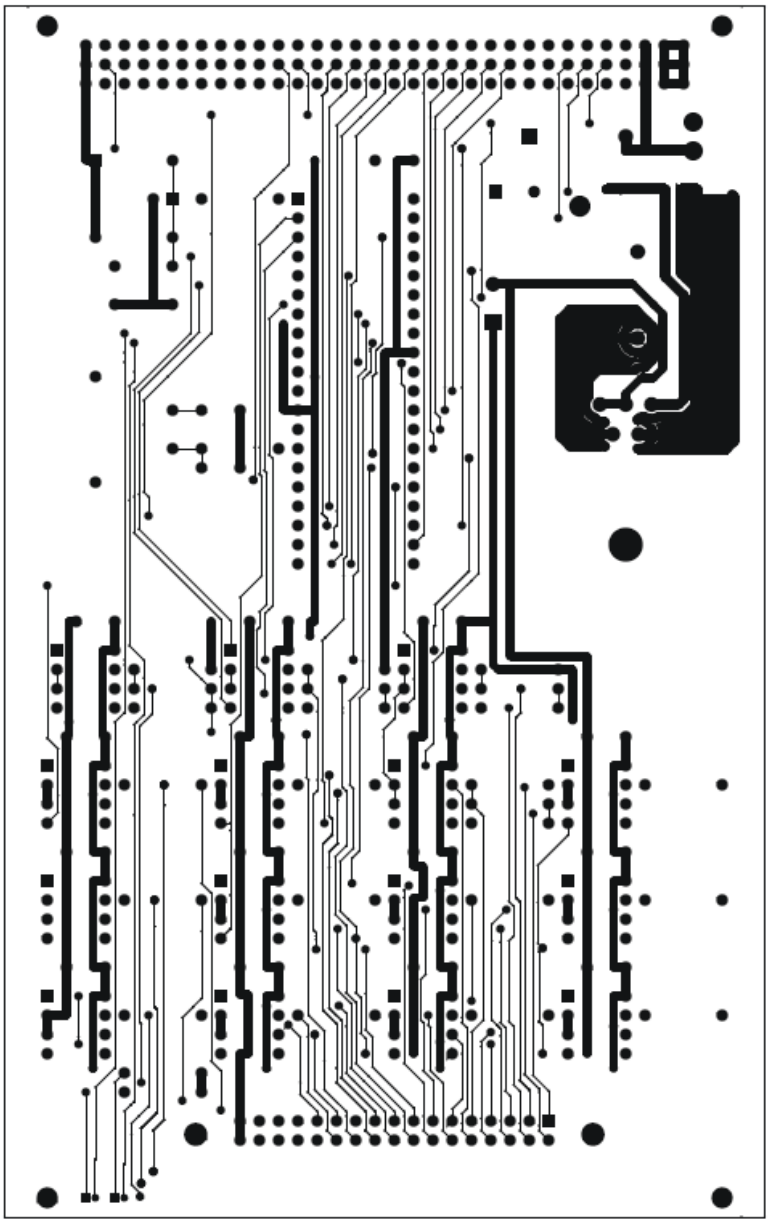


Figura II.3. *Placa de la interfaz serie/paralelo (SerPar) de la antena del Vesubio (esquema en figura 4.14 de la memoria): distribución de componentes (a), cara superior (b) y cara inferior (c).*

II.3.b)



II.3.c)



a)

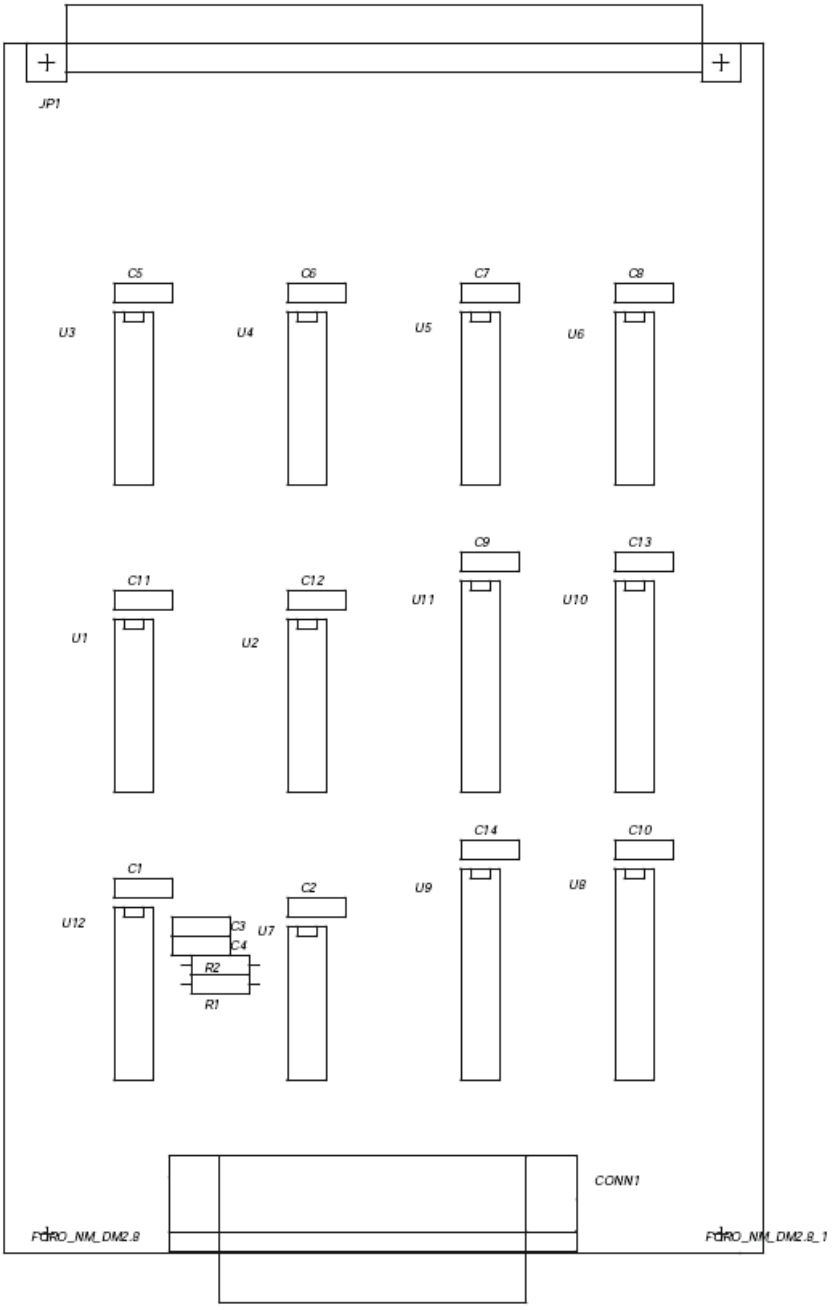
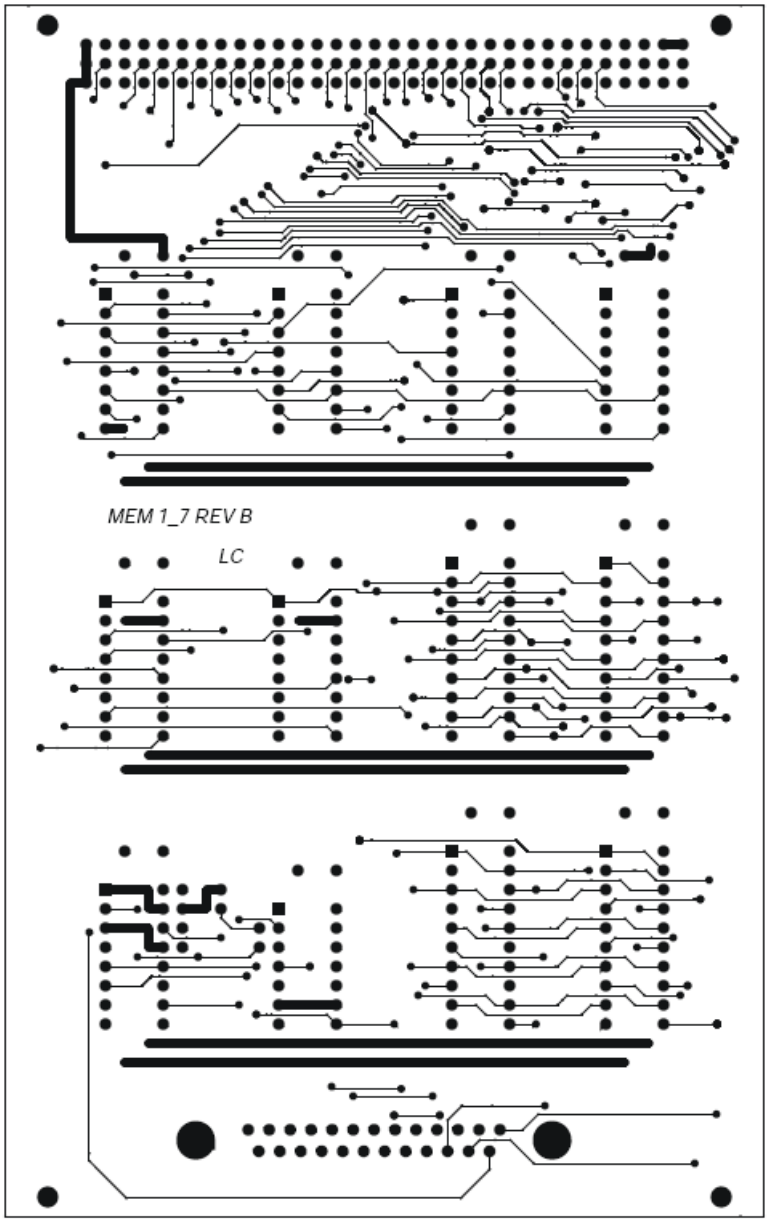
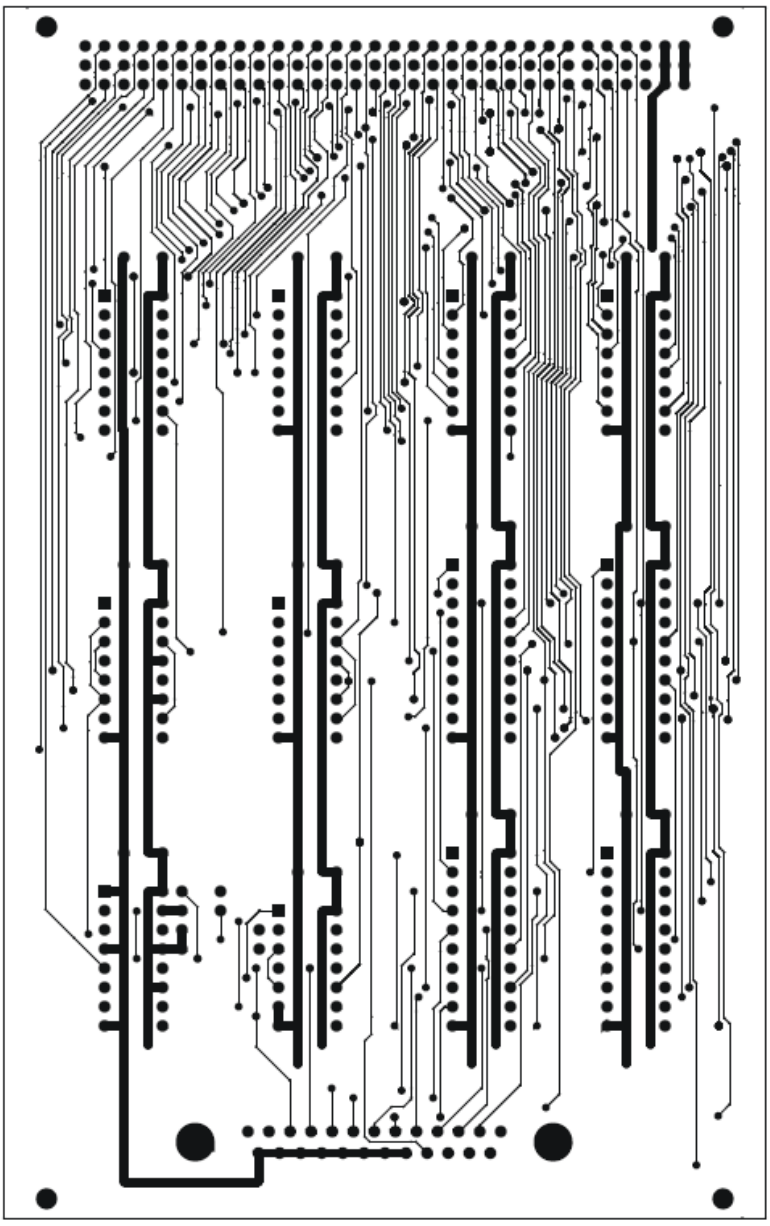


Figura II.4. *Placa de control del módulo de memoria (esquema en figura 4.15 de la memoria): distribución de componentes (a), cara superior (b) y cara inferior (c).*

II.4.b)



II.4.c)



a)

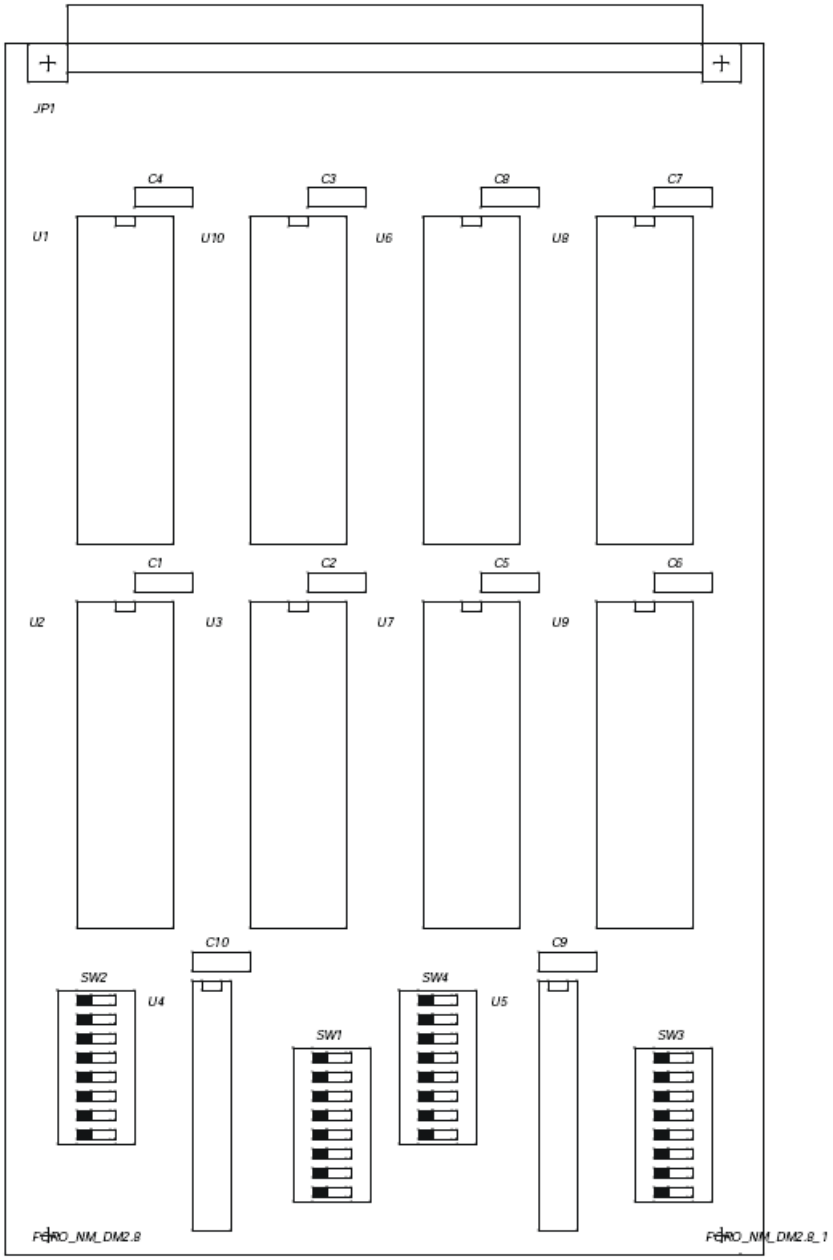
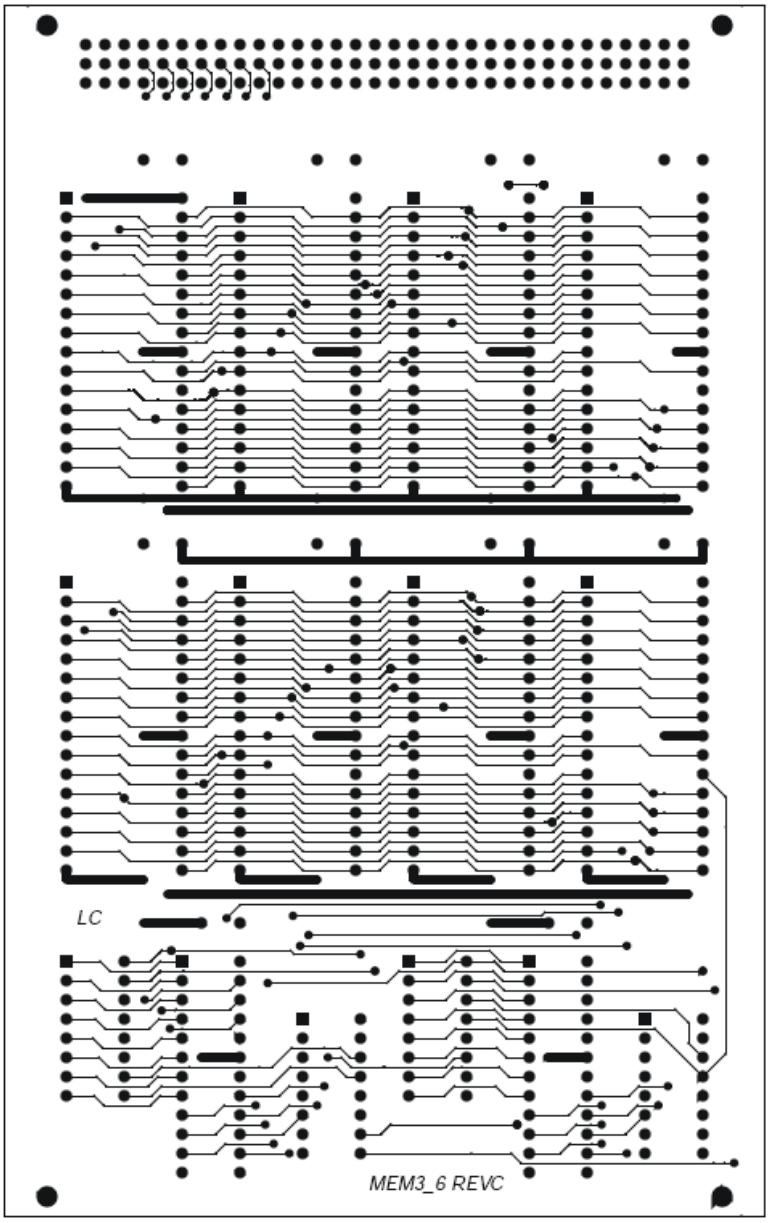
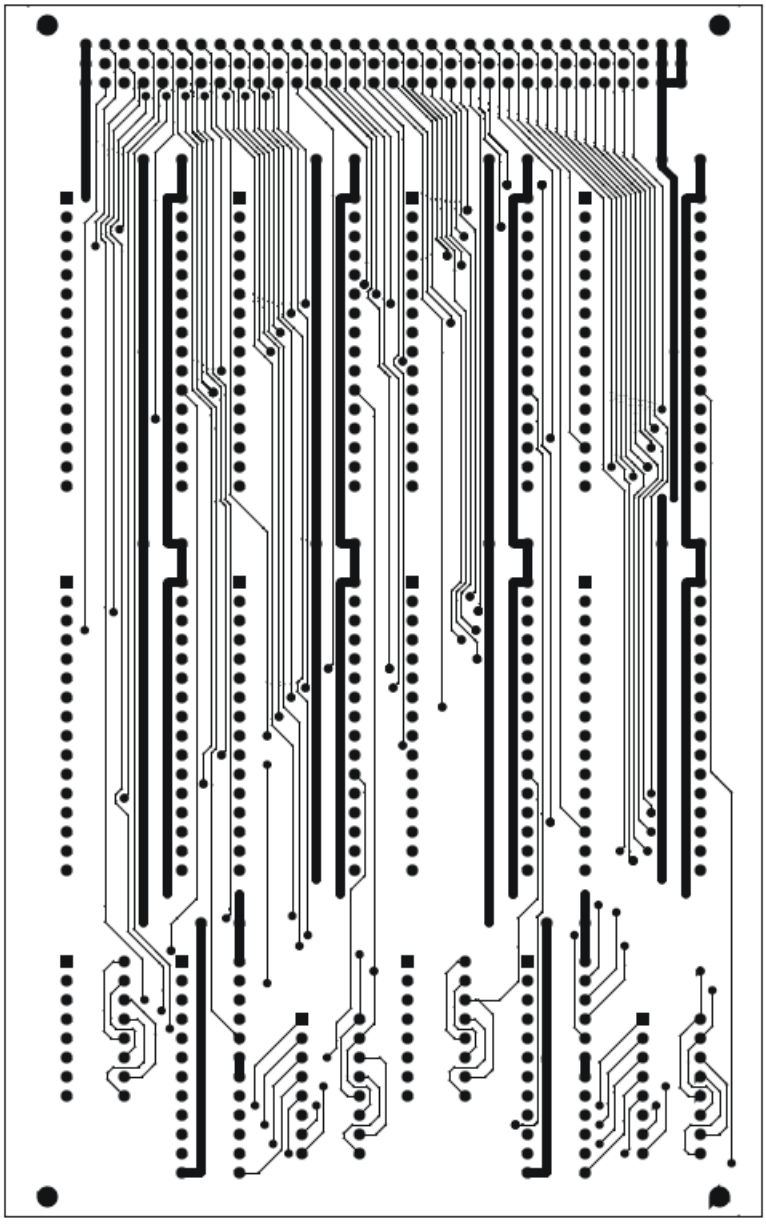


Figura II.5. *Tarjeta de expansión de memoria (esquema en figura 4.16 de la memoria de tesis): distribución de componentes (a), cara superior (b) y cara inferior (c).*

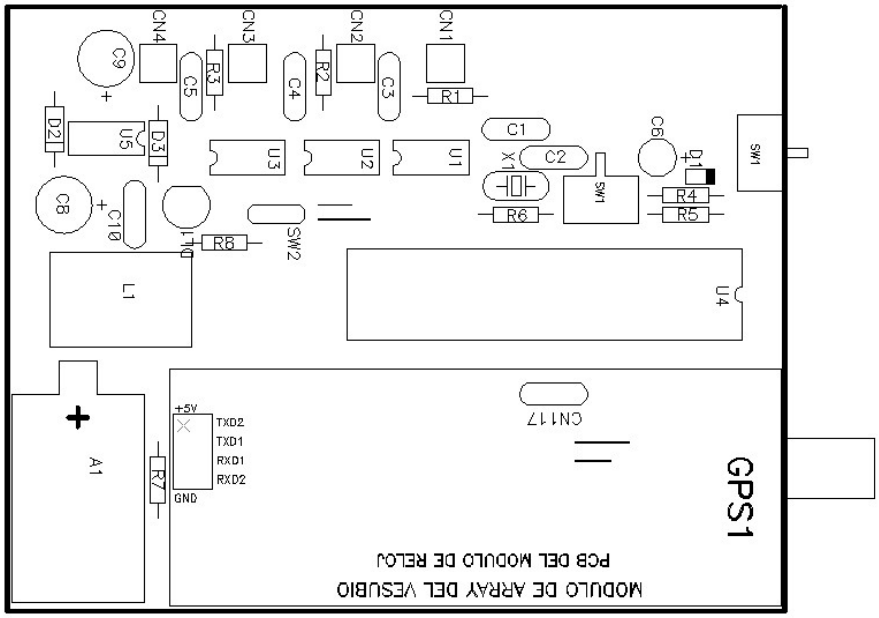
II.5.b)



II.5.c)



a)



b)

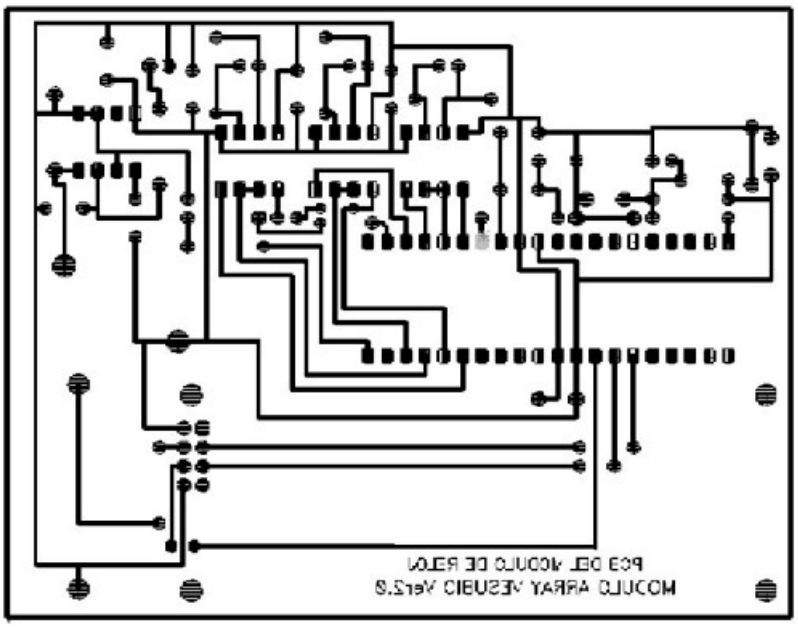


Figura II.6. *Tarjeta de control del módulo de reloj (esquema en figura 4.16 de la memoria de tesis): distribución de componentes (a) y cara inferior (b) (diseño de la tarjeta: Javier Moreno Peláez).*

Anexo II.C:

Otros anexos relacionados con la antena del Vesubio

ANEXO II.C.1.- SECUENCIAS DE INICIALIZACIÓN EN LA ANTENA DEL VESUBIO

II.C.1.a. PROTOCOLO DE TRANSMISIÓN DE LOS PARÁMETROS DE OPERACIÓN ENTRE EL PC Y EL MÓDULO SERPAR

El protocolo usado para la transmisión de los parámetros de operación desde el PC al módulo SerPar se muestra en la figura II.7. Para esta comunicación se utilizan las líneas *IN Configuration* y *OUT Configuration*, correspondientes a las siguientes señales de control:

Señal	*SerPar I/O*	*Señal PIC SerPar*	*Pin PIC SerPar*	*P. Par. I/O*	*Señal P. Paralelo*	*Pin P. Paralelo*	*Back-plane*
IN Conf.	I	RB6	39	O	/INIT	16	B26
OUTConf	O	RB7	40	I	/ERROR	15	B27

Como puede verse en la figura, cuando el programa del PC arranca envía continuamente pulsos negativos de 250 ms de duración por la línea *IN Configuration*, para comunicar al módulo SerPar que está preparado para enviar los datos. Cuando el módulo SerPar detecta uno de estos pulsos manda un pulso positivo de un segundo por la línea *OUT Configuration,* para comunicar al PC que ha detectado la señal. El PC se prepara para mandar datos y el módulo SerPar, después de un retardo de un segundo, empieza a mandar pulsos positivos de 500 ms de duración para sincronizar la transmisión de los datos. Cada vez que el PC detecta un pulso pone la línea *IN Configuration* a 0 o 1, en función del valor del siguiente bit de la cadena de configuración. Dicha cadena es generada por el programa del PC a partir de los parámetros de operación que lee del fichero de texto VES2.CFG. La cadena está formada por un total de 30 bits, con el siguiente formato:

Nº bit	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Tipo bit	F2	F1	F0	G7	G6	G5	G4	G3	G2	G1	G0	L00	L10	L20
Nº bit	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**
Tipo bit	L30	L01	L11	L21	L31	L02	L12	L22	L32	L03	L13	L23	L33	M2
Nº bit	**28**	**29**												
Tipo bit	M1	M0												

El significado de los bits es el siguiente:

- **F2/F1/F0**: Frecuencia de muestreo (001 = 50mps, 010 = 100mps, 100 = 200mps)
- **G7/G6/.../G0**: Ganancia (01h para ganancia 1, 02h para ganancia 2, 04h para ganancia 4, 08h para ganancia 8, 10h para ganancia 16, 20h para ganancia 32, 40h para ganancia 64, 80h para ganancia 128).
- **L00/L10/.../L33**: Configuración física del array. Cada uno de los bits Lxy expresa que la estación y de la línea serie x está operativa (si vale 1) o no operativa (si vale 0).
- **M2/M1/M0**: Número de tarjetas de memoria (001 = 1 tarjeta, 010 = 2, 100 = 4).

Como puede verse, esta codificación utiliza mayor número de bits del estrictamente necesario. Los campos primero, segundo y cuarto se podrían codificar utilizando dos, tres y dos bits, respectivamente, para un total de siete bits en lugar de los catorce que se emplean. Sin embargo, se ha preferido codificar la información según este formato para simplificar el algoritmo de reconocimiento en el PIC, que de este modo se puede realizar mediante simples instrucciones de desplazamiento. No supone un perjuicio utilizar más bits de los necesarios, ya que la comunicación de parámetros no es una tarea crítica en el tiempo y sólo debe realizarse una vez, en el arranque del sistema.

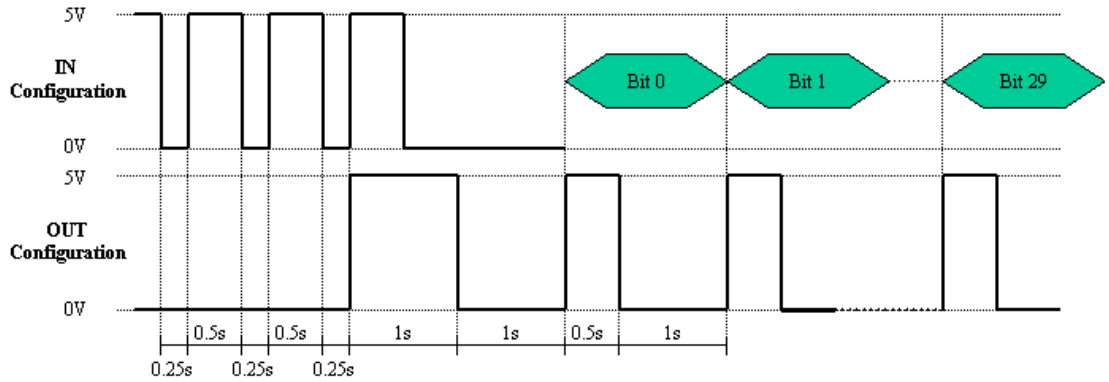


Figura II.7. *Protocolo de la comunicación para el intercambio de los parámetros de configuración entre el PC y el módulo serie/paralelo.*

II.C.1.b. PROTOCOLO DE TRANSMISIÓN DE LOS PARÁMETROS DE CONFIGURACIÓN ENTRE EL MÓDULO SERPAR Y LOS MÓDULOS DE ADQUISICIÓN

Los parámetros que los módulos de adquisición deben conocer antes de comenzar a adquirir datos se reducen a la frecuencia de muestreo y la ganancia, por lo que no es necesario transmitirles la configuración física del *array* ni el número de tarjetas de memoria utilizadas. La transmisión de la información se realiza a través del canal Ser1 (ver sección 4.4.3) de las cuatro líneas serie. Las líneas de control utilizadas en el módulo SerPar y en los módulos de adquisición son:

Señal	*SerPar I/O*	*Puertos PIC SerPar*	*Pines PIC SerPar*	*Módulo Adquis. I/O*	*Puerto PIC Adquis.*	*Pin PIC Adquis.*
Pulsos Inic.	O	RA5, RA3, RA2, RA1	7, 5, 4, 3	I	RB0	6

La codificación de la información se reduce a una secuencia de distinto número de pulsos positivos de duración constante (1ms). El número de pulsos varía en función de la frecuencia de muestreo y ganancia seleccionadas, según la tabla que se muestra a continuación:

Nº pulsos	11	12	13	14	15	16	17	18	19	20	21	22
Frec. (mps)	50	50	50	50	50	50	50	50	100	100	100	100
Ganancia	1	2	4	8	16	32	64	128	1	2	4	8
Nº pulsos	23	24	25	26	27	28	29	30	31	32	33	34
Frec. (mps)	100	100	100	100	200	200	200	200	200	200	200	200
Ganancia	16	32	64	128	1	2	4	8	16	32	64	128

Al igual que en el caso del módulo SerPar, el programa de los módulos de adquisición espera, al arrancar, la secuencia de pulsos de inicialización. Una vez recibe el primer pulso activa un *timer* para saber cuándo tiene que dar por concluida la fase de inicialización, y mientras ésta dura incrementa un contador cada vez que recibe un pulso. Cuando finaliza la recepción de la secuencia de pulsos el PIC programa los conversores con los parámetros correspondientes. Como puede verse en la tabla, el número mínimo de pulsos de la secuencia es once, para evitar que algún pulso espúreo pueda ser interpretado como una secuencia por los módulos de adquisición. Así, en caso de que el número de pulsos detectados al final del periodo de inicialización por alguno de los módulos de adquisición fuera menor de once, éste lo interpretaría como una cadena falsa y volvería a la espera de la secuencia.

II.C.1.c. SECUENCIA DE INICIALIZACIÓN EN EL LED DEL MÓDULO SERPAR

Con el objeto de poder verificar que el arranque del sistema ha sido correcto, el LED verde del módulo SerPar se utiliza como testigo del proceso, iluminándose de acuerdo con el siguiente patrón:

- Una vez ha detectado los pulsos de segundo procedentes del módulo de reloj, el módulo SerPar retarda unos segundos el comienzo del programa. Esto se hace para evitar que hipotéticos cambios de tensión en las líneas de comunicación con el PC durante el arranque de éste sean interpretados por SerPar como el inicio de la cadena de comunicación de los parámetros de operación. Durante estos segundos el LED verde parpadeará con pulsos 'largos' de 250 ms de duración.
- Durante la espera a que el PC comience la transmisión de la secuencia de parámetros y durante el tiempo que dure ésta, el LED verde permanecerá encendido.
- Cuando termina el intercambio de parámetros hay un nuevo bucle de espera con pulsos 'largos', después del cual se inicia el funcionamiento normal. A partir de entonces y mientras el sistema se encuentre en operación normal, el LED verde monitorizará los PPS parpadeando con pulsos 'cortos' de 1 ms aproximadamente.
- En caso de que se envíe un comando de reset desde el PC el módulo de reloj dejará de dar PPS durante treinta segundos, por lo que el LED verde de SerPar también dejará de parpadear. Una vez reanudados los PPS el LED verde volverá a la secuencia inicial, ya que todos los módulos del sistema deben haber reiniciado sus programas (ver sección 4.5.2).

Por tanto, la secuencia normal de arranque en el LED verde de SerPar consiste en:

1. Diez pulsos 'largos' (bucle de espera inicial).
2. LED encendido (espera de arranque del PC + intercambio de parámetros) durante un mínimo de 45 segundos (tiempo del intercambio de parámetros).
3. Once pulsos 'largos' (segundo bucle de espera).
4. Pulsos 'cortos' mientras el sistema opere normalmente.
5. LED apagado durante treinta segundos como respuesta a un comando de reset, con posterior retorno a la secuencia inicial.

ANEXO II.C.2.- FORMATO DE LOS DATOS EN EL MÓDULO DE MEMORIA DE LA ANTENA DEL VESUBIO

El formato de los datos en cada banco de memoria, descrito en la sección 4.3.4, puede verse en la siguiente tabla:

Cabecera parámetros configuración
Cabecera GPS
Datos
Cabecera GPS
Datos
Cabecera GPS
Datos
...
Cabecera GPS
Datos
Bytes no usados

A continuación se describen los formatos de cada uno de estos paquetes.

A. Cabecera de parámetros de configuración:

Los cuatro primeros bytes de cada banco de memoria corresponden a los parámetros de configuración con los que opera el *array*, según el siguiente formato:

Máscara de módulos A/D operativos (MSB)
Máscara de módulos A/D operativos (LSB)
Ganancia
Frecuencia de muestreo + número de tarjetas de memoria usadas

A.1: Máscara de módulos A/D operativos.- Ocupa dos bytes con el siguiente formato:

MS bit															*LS bit*
L0 M0	L1 M0	L2 M0	L3 M0	L0 M1	L1 M1	L2 M1	L3 M1	L0 M2	L1 M2	L2 M2	L3 M2	L0 M3	L1 M3	L2 M3	L3 M3

siendo LxMy el bit que representa el estado del módulo A/D número ‘y’ de la línea serie ‘x’. Los valores 0 y 1 indican que el correspondiente módulo se encuentra en estado no operativo u operativo, respectivamente.

A.2. Ganancia.- Ocupa un byte, que toma como valores posibles 1, 2, 4, 8, 16, 32, 64 o 128.

A.3. Frecuencia de muestreo + número de tarjetas de memoria usadas.- Ocupa un byte, con el siguiente formato:

MS bit							*LS bit*
X	X	F	F	F	M	M	M

siendo:

- X: bits no usados.
- F: frecuencia de muestreo. Puede tomar los siguientes valores:
  - FFF = 100: Frecuencia de muestreo = 200mps.
  - FFF = 010: Frecuencia de muestreo = 100mps.
  - FFF = 001: Frecuencia de muestreo = 50mps.
- M: Número de tarjetas de memoria usadas. Puede tomar los siguientes valores:
  - MMM = 100: 4 tarjetas de memoria.
  - MMM = 010: 2 tarjetas de memoria.
  - MMM = 001: 1 tarjeta de memoria.

B. Cabecera GPS:

Ocupa doce bytes, con el siguiente formato:

Status GPS
Tiempo GPS

B.1. Status GPS.- Ocupa dos bytes, con el formato correspondiente al paquete 0x46 del protocolo TSIP (*Trimble Standard Interface Protocol*, ver por ejemplo Trimble Navigation Limited, 1999).

*B.2. Tiempo GPS***.-** Ocupa diez bytes, con el formato correspondiente al paquete 41h del protocolo TSIP. Este tiempo corresponde a los datos muestreados en el segundo anterior. La información correspondiente a la fracción de segundo no es significativa. Para conocer el tiempo exacto de cada muestra simplemente debe tenerse en cuenta que el primer dato de cada segundo se toma 200 μs después del segundo exacto.

*B.3. Códigos especiales***.-** Cuando se da una situación especial en el tratamiento de la información del módulo de reloj la cabecera GPS no tiene la información descrita en los dos puntos anteriores, sino una secuencia de doce caracteres iguales. Estos caracteres son introducidos por el programa del módulo SerPar para posibilitar la identificación de la situación anómala. Para evitar confusiones se han elegido caracteres distintos de los de las cabeceras válidas de status GPS (ver punto B.1), y fácilmente identificables cuando se consultan los datos con un editor ASCII. Los valores que pueden tomar dichos caracteres y su significado son:

- 30h (ASCII ‘0’): Primer dato del módulo de reloj. Cuando se recibe es señal de que el módulo de reloj acaba de empezar a mandar datos, ya sea por arranque de todo el sistema o sólo del propio módulo de reloj por actuación del *watchdog timer*.
- 31h (ASCII ‘1’): No hay datos GPS. El PIC del módulo de reloj no tenía datos listos al recibir la petición de SerPar.
- 32h (ASCII ‘2’): Error de desbordamiento. Se ha producido un desbordamiento del búffer de recepción serie del PIC del módulo SerPar.
- 33h (ASCII ‘3’): Error de trama. El PIC del módulo SerPar ha detectado un bit de STOP a 0.
- 34h (ASCII ‘4’): Error de *time out*. El módulo SerPar no ha completado la comunicación serie con el módulo de reloj en el tiempo asignado para ello (primer ms de cada segundo).

C. Datos:

Cada paquete de datos entre dos cabeceras GPS corresponde a un segundo de muestreo. El número de bytes por paquete depende del número de módulos A/D operativos y de la frecuencia de muestreo utilizada. En caso, por ejemplo, de una configuración con ocho módulos A/D operando a una frecuencia de muestreo de 100mps, el número de bytes en cada paquete sería:

$$9\frac{bytes}{muestra \cdot módulo} \cdot 100\frac{muestras}{segundo} \cdot 8módulos = 7200bytes$$

Como se ha dicho antes, el orden de los datos en memoria depende de la configuración física del *array*. Los bytes correspondientes a módulos A/D con el mismo número pero pertenecientes a líneas serie distintas son leídos por SerPar en paralelo, por lo que quedarán intercalados en la memoria. Así, para una configuración en que sólo se usaran los módulos 0 de las cuatro líneas serie, los datos quedarían en el siguiente orden:

B0D0M0L0
B0D0M0L1
B0D0M0L2
B0D0M0L3
B1D0M0L0
B1D0M0L1
B1D0M0L2
B1D0M0L3
B2D0M0L0
...
B8D0M0L3
B0D1M0L0
B0D1M0L1
...
B8D99M0L3

siendo BwDxMyLz el byte ‘w’ del dato ‘x’ del módulo ‘y’ de la línea serie ‘z’.

De modo análogo, cuando los módulos A/D pertenecen a la misma línea serie, los datos aparecen consecutivos. Así, para una configuración en que sólo se usara la línea serie 0, con los cuatro módulos operativos, los datos quedarían en el siguiente orden:

B0D0M0L0
B1D0M0L0
B2D0M0L0
B3D0M0L0
B4D0M0L0
B5D0M0L0
B6D0M0L0
B7D0M0L0
B8D0M0L0
B0D0M1L0
B1D0M1L0
...
B8D0M3L0
B0D1M0L0
...
B8D99M3L0

Cuando la configuración incluye varios módulos A/D de distintas líneas serie, los datos quedan como una combinación de los dos casos anteriores. Por ejemplo, si se están utilizando los módulos 0 y 1 de la línea serie 0 y el módulo 0 de la línea serie 1, quedaría:

B0D0M0L0
B0D0M0L1
B1D0M0L0
B1D0M0L1
B2D0M0L0
B2D0M0L1
B3D0M0L0
B3D0M0L1
B4D0M0L0
B4D0M0L1
B5D0M0L0
B5D0M0L1
B6D0M0L0
B6D0M0L1
B7D0M0L0
B7D0M0L1
B8D0M0L0
B8D0M0L1
B0D0M1L0
B1D0M1L0
B2D0M1L0
B3D0M1L0

B4D0M1L0
B5D0M1L0
B6D0M1L0
B7D0M1L0
B8D0M1L0
B0D1M0L0
B0D1M0L1
...
B8D99M1L0

D. Bytes no usados:

Según el formato descrito, el número de bytes que el módulo SerPar escribe en cada banco de memoria será de:

$$N^{o}\,bytes\,/_{Banco} = (9 \cdot N^{o}\,ModsAD \cdot Frec + 12) \cdot NumSegs + 4$$

El resto de los bytes de cada banco no se utiliza. Aunque eso supone una pequeña pérdida de capacidad, se prefirió hacerlo así para evitar que el último segundo de datos quedara dividido entre los dos bancos. De esta forma cada banco almacena segundos completos de datos, lo cual permite un control mayor de los paquetes ya que los diferentes campos empiezan siempre en la misma posición de memoria.

El número de segundos depende del número de tarjetas de memoria usado, de la frecuencia de muestreo y del número de módulos de adquisición operativos. Como ejemplo, para una frecuencia de muestreo de 100 mps, 2 tarjetas de memoria y cuatro módulos A/D operativos, el número de segundos por banco de memoria sería 289, con lo cual el número de bytes escritos por el módulo SerPar en cada banco, según la expresión anterior, sería 1043872. Los 4704 bytes restantes, hasta el total de 1048576 de cada banco, no se usarían.

ANEXO III

ANTENAS PORTÁTILES DEL IAG

Anexo III.A:

Programas de las antenas portátiles del IAG

ANEXO III.A.1.- PROGRAMAS DE LOS PCs DE CONTROL DE LAS ANTENAS PORTÁTILES DEL IAG

III.A.1.a. FICHERO DE CONFIGURACIÓN DE LAS ANTENAS PORTÁTILES DEL IAG

Fichero: SEISAD18.CFG

/* Fichero de configuración de la antena del Vesubio, al que accede el programa de inicialización INICARR2 (anexo III.A.1.b) para conocer los valores de los parámetros de operación. */

/*
Fichero
***/**

FrecMuestreo 100
Resolución 24
Ganancia 4
CódigoArray SA

La única condición en el fichero de configuración es que las cuatro primeras líneas tengan el formato mostrado. Después puede insertarse cualquier texto de información (como éste) sin ningún formato predefinido.

En esta versión de los sistemas los valores de frecuencia de muestreo y resolución son fijos (100 mps y 24 bits). Sin embargo, se han incluido los correspondientes parámetros en este fichero de configuración para permitir, en futuras versiones, valores de 50 y 200 mps y resolución de 16 bits. En lo que respecta a la ganancia, los posibles valores son 1, 4 o 16.

El código de array debe estar formado por uno o dos caracteres alfanuméricos, que aparecerán en las extensiones de los ficheros de datos y LOG.

Si el fichero SEISAD18.CFG no se encuentra en el directorio de trabajo, su formato es incorrecto o el valor de alguno de los parámetros no se encuentra entre los especificados, se tomarán los siguientes valores por defecto:

FrecMuestreo 100
Resolución 24
Ganancia 1
CódigoArray 00

III.A.1.b. PROGRAMA DE INICIALIZACIÓN DE LAS TARJETAS CONTROL DEL SISTEMA

Fichero: INICARR2.C

/* Programa de inicialización de las antenas portátiles del IAG. El valor de los parámetros de operación se lee del fichero de configuración SEISAD18.CFG (anexo III.A.1.a). Las funciones para la transmisión serie son las del fichero MISERIE.H (anexo I.A.1.a). El programa principal se encuentra al final. */

```
/*************************************************************************
  Macros y ficheros de cabecera
*************************************************************************/
#define  TRUE                    1
#define  FALSE                   0
#define  MAL                     1
#define  OK                      0
#define  NUM_PORTS               4
#define  portpar                 888
#define  portpar1                portpar + 1
#define  portpar2                portpar + 2
#define  portpar3                portpar + 7
#define  Tamano_Buf_Recepcion1 50000
#define  Tamano_Buf_Recepcion2 500

#include <miserie.h>
#include <dos.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>

/*************************************************************************
  Declaración de variables
*************************************************************************/
unsigned int FrecMuestreo, Resolucion, Ganancia;
FILE *FichCfg, *FichLog;
// Flags booleanos
struct
{
        unsigned FichCfgOK:                                     1;
} Flag =
{ FALSE };

/*************************************************************************
  Declaración de funciones
*************************************************************************/
void AbreFichLOG(void);
void LeeFichCfg(void);
void InicSEISAD18(void);
```

```
/*****************************************************************************
```

Definición de las funciones

```
*****************************************************************************/
void AbreFichLOG(void)
{
	if ((FichLog = fopen("InicArr2.LOG", "w")) == NULL)
	{
		printf("\n No se puede abrir el fichero InicArr2.LOG");
		exit(0);
	}
}

void LeeFichCfg(void)
{
	int i;
	char Texto[20];

	Flag.FichCfgOK = TRUE;// En principio el fichero de configuración existe y es correcto
	if ((FichCfg = fopen("SEISAD18.CFG", "r")) == NULL)
	{
		printf("\n No se puede abrir el fichero SEISAD18.CFG");
		Flag.FichCfgOK = FALSE;
	}
	else
	{
		fscanf(FichCfg,"%s %u", &Texto, &FrecMuestreo);
		// De momento, sólo es posible frecuencia 100:
		//if ((FrecMuestreo != 50)&&(FrecMuestreo != 100)&&(FrecMuestreo != 200))
		if (FrecMuestreo != 100) Flag.FichCfgOK = FALSE;

		if (Flag.FichCfgOK)// Sólo comprueba el parámetro siguiente si el anterior era correcto
		{
			fscanf(FichCfg,"%s %u", &Texto, &Resolucion);
			// De momento, sólo es posible resolución de 24 bits:
			//if ((Resolucion != 16)&&(Resolucion != 24)) Flag.FichConfOK = FALSE;
			if (Resolucion != 24) Flag.FichCfgOK = FALSE;
		}

		if (Flag.FichCfgOK)
		{
			fscanf(FichCfg,"%s %u", &Texto, &Ganancia);
			if ((Ganancia != 1)&&(Ganancia != 4)&&(Ganancia != 16)) Flag.FichCfgOK =
			FALSE;
		}
	}

	// Si ha encontrado algún fallo en el fichero o no ha podido abrirlo, toma los parámetros por defecto.
	if (!Flag.FichCfgOK)
	{
		FrecMuestreo   = 100;
		Resolucion     = 24;
		Ganancia       = 1;
	}
```

```
	// Escribe las incidencias en el LOG
	if (!Flag.FichCfgOK)
	{
		fprintf(FichLog,"\n El fichero SEISAD18.CFG no existe o su formato es incorrecto");
		fprintf(FichLog,"\n Se continúa el programa con los par metros por defecto:");
		fprintf(FichLog,"\n Frecuencia de muestreo: 100mps\n Resolución: 24bits\n Ganancia:
		1\n");
		printf("\n El fichero SEISAD18.CFG no existe o su formato es incorrecto");
		printf("\n Se continúa el programa con los par metros por defecto:");
		printf("\n Frecuencia de muestreo: 100mps\n Resolución: 24bits\n Ganancia: 1");
	}
	else
	{
		fprintf(FichLog,"\n Par metros de operación leídos del fichero SEISAD18.CFG:");
		fprintf(FichLog,"\n Frecuencia de muestreo: %umps\n Resolución: %ubits\n Ganancia:
		%u", FrecMuestreo, Resolucion, Ganancia);
		printf("\n Par metros de operación leídos del fichero SEISAD18.CFG:");
		printf("\n Frecuencia de muestreo: %umps\n Resolución: %ubits\n Ganancia: %u",
		FrecMuestreo, Resolucion, Ganancia);
	}
}

void InicSEISAD18(void)
{
	unsigned char i, NumPulsos;
	// 115200bps, 100mps y ganancia 1, 4 o 16 (29 pulsos para ganancia 1, 30 para ganancia 4 y 31 para
	//ganancia 16) (de momento la frecuencia de muestreo y resolución no son configurables):
	switch (Ganancia)
	{
		case 1:
			NumPulsos = 29;
			break;
		case 4:
			NumPulsos = 30;
			break;
		case 16:
			NumPulsos = 31;
			break;
		default:
			NumPulsos = 29;// Por defecto, ganancia 1
	}
	for (i=0; i<NumPulsos; i++)
	{
		MandarCaracter(COM1, 0x00);
	}
	fprintf(FichLog, "\n Tarjetas SEISAD18 inicializadas a 115200bps, 100mps, ganancia %u",
	Ganancia);
	printf("\n Tarjetas SEISAD18 inicializadas a 115200bps, 100mps, ganancia %u", Ganancia);
}
```

```
/*****************************************************************************
```

Programa principal

```
*****************************************************************************/
void main(void)
{
        clrscr();
        AbreFichLOG();
        printf(" Programa InicArr2.C\n");
        fprintf(FichLog," Programa InicArr2.C\n");
        InicializarComunicacionSerie(COM1,BAUD115200,NINGUNA,LON8,STOP1);
        LeeFichCfg();
        InicSEISAD18();
        FinalizarComunicacionSerie();
        fclose(FichLog);
        fclose(FichCfg);
        delay(4000);    // Para que el programa de adquisición ARRAYS8 no reciba los primeros segundos
                        //de datos, que dan errores de formato.
}
```

III.A.1.c. PROGRAMA DE ADQUISICIÓN DE DATOS DE LOS PCs DE CONTROL DE LAS ANTENAS PORTÁTILES DEL IAG

Fichero: ARRAYS8.C

/* Programa de los PCs industriales de las nuevas antenas portátiles del IAG. Las funciones para la transmisión serie son las del fichero MISERIE.H (anexo I.A.1.a). El programa principal se encuentra al final. */

```
/*************************************************************************
  Macros
*************************************************************************/
#define  TRUE                    1
#define  FALSE                   0
#define  MAL                     1
#define  OK                      0
#define  NUM_PORTS               4
#define  portpar                 888
#define  portpar1                portpar + 1
#define  portpar2                portpar + 2
#define  portpar3                portpar + 7
#define  Tamano_Buf_Recepcion1 50000
#define  Tamano_Buf_Recepcion2 500

#include <miserie.h>
#include <dos.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>

/*************************************************************************
  Declaración de variables
*************************************************************************/
unsigned int i;
FILE *FichDatos, *FichLog, *FichCfg;
unsigned int FrecMuestreo, Resolucion, Ganancia;
unsigned char CodigoArr[3];
unsigned long int ContBytesEscritos1, ContBytesEscritos2, ContProvisional1;
unsigned long int EspacioLibre;
unsigned char ContIndicadorDatos = 0, ContIndicadorGPS = 0;
unsigned char ContLEDDatos = 0, ContLEDGPS = 0;
unsigned char Hora, Minuto, Segundo, Dia, Mes;
unsigned int Year;
unsigned char HoraFich, MinutoFich, SegundoFich, DiaFich, MesFich;
unsigned int YearFich;
unsigned char TipoError;
fpos_t filepos;
unsigned int ContPPS;
```

```
// Flags booleanos
struct
{
        unsigned HayPPS:            1;
        unsigned SincGPS:           1;
        unsigned ErrorTramaDatos:   1;
        unsigned ErrorBufferDatos:  1;
        unsigned ErrorBufferGPS:    1;
        unsigned ErrorTramaGPS:     1;
        unsigned SalidaManual:      1;
        unsigned DatosADiscoUSB:    1;
        unsigned FichCfgOK:         1;
} Flag =
{ FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE };

/*****************************************************************************
  Declaración de funciones
*****************************************************************************/
void LeeFichCfg(void);
void AbreFicheroLOG(void);
void AbreFichDatos(void);
void CambiaIndicadorDatos(void);
void CambiaIndicadorGPS(void);
unsigned char PPS_1(void);
void CompruebaPPS_Polling(void);
void TomaHoraPC(void);
void TomaFechaPC(void);
void SincronizaHoraPC(void);
void InicSEISAD18(void);
unsigned char SalidaManual(void);

/*****************************************************************************
  Definición de las funciones
*****************************************************************************/
void LeeFichCfg(void)
{
        int i;
        char Texto[20];

        Flag.FichCfgOK = TRUE;// En principio el fichero de configuración existe y es correcto
        if ((FichCfg = fopen("SEISAD18.CFG", "r")) == NULL)
        {
                printf("\n No se puede abrir el fichero SEISAD18.CFG");
                Flag.FichCfgOK = FALSE;
        }
        else
        {
                fscanf(FichCfg,"%s %u", &Texto, &FrecMuestreo);
                // De momento, sólo es posible frecuencia 100:
                //if ((FrecMuestreo != 50)&&(FrecMuestreo != 100)&&(FrecMuestreo != 200))
                if (FrecMuestreo != 100) Flag.FichCfgOK = FALSE;

                if (Flag.FichCfgOK)// Sólo comprueba el parámetro siguiente si el anterior era correcto
                {
```

```
                        fscanf(FichCfg,"%s %u", &Texto, &Resolucion);
                        // De momento, sólo es posible resolución de 24 bits:
                        //if ((Resolucion != 16)&&(Resolucion != 24)) Flag.FichConfOK = FALSE;
                        if (Resolucion != 24) Flag.FichCfgOK = FALSE;
                }

                if (Flag.FichCfgOK)
                {
                        fscanf(FichCfg,"%s %u", &Texto, &Ganancia);
                        if ((Ganancia != 1)&&(Ganancia != 4)&&(Ganancia != 16)) Flag.FichCfgOK =
                        FALSE;
                }

                if (Flag.FichCfgOK)
                {
                        fscanf(FichCfg,"%s %s", &Texto, &CodigoArr);
                        // No se hace ninguna comprobación del código del array, porque en principio
                        //sirven cualquier pareja de caracteres
                }
        }

        // Si ha encontrado algún fallo en el fichero o no ha podido abrirlo, toma los parámetros por defecto.
        if (!Flag.FichCfgOK)
        {
                FrecMuestreo   = 100;
                Resolucion     = 24;
                Ganancia       = 1;
                sprintf(CodigoArr,"00");   // Para que nombre los ficheros de datos como A00 y los LOG
                                           //como L00
        }
}

void AbreFicheroLOG(void)
{
        char NombreFichero[25];

        sprintf(NombreFichero,"C:\\DATA\\%02u%02u%02u%02u.L%s", MesFich, DiaFich, HoraFich,
        MinutoFich, CodigoArr);

        // Abre el fichero LOG:
        if ((FichLog = fopen(NombreFichero, "w")) == NULL)
        {
                printf("\n No se puede abrir el fichero %s", NombreFichero);
                exit(0);
        }
        // Escribe la información de lo que ha pasado hasta ahora, consultando los distintos flags:
        else
        {
                printf("\n Fichero %s abierto para escritura", NombreFichero);
                fprintf(FichLog, " Programa Arrays8.C\n");

                if (Flag.SalidaManual)
                {
                        fprintf(FichLog, "\n %02u:%02u:%02u: Salida manual del programa", Hora,
                        Minuto, Segundo);
```

```
                        printf("\n %02u:%02u:%02u: Salida manual del programa", Hora, Minuto,
                        Segundo);
                        FinalizarComunicacionSerie();
                        exit(1);  // Código de salida 1, para no volver a lanzar el programa de adquisición
                                  //desde el fichero ARRAYS.BAT
                }
                if (!Flag.HayPPS)
                {
                        fprintf(FichLog, "\n No hay señal de PPS, saliendo del programa...");
                        printf("\n No hay señal de PPS, saliendo del programa...");
                        FinalizarComunicacionSerie();
                        exit(0);
                }
                if (!Flag.SincGPS)
                {
                        fprintf(FichLog,"\n No se recibe hora buena del GPS, saliendo del programa...\n");
                        printf("\n No se recibe hora buena del GPS, saliendo del programa...\n");
                        FinalizarComunicacionSerie();
                        exit(0);
                }
                if (!Flag.FichCfgOK)
                {
                        fprintf(FichLog,"\n El fichero SEISAD18.CFG no existe o su formato es
                        incorrecto");
                        fprintf(FichLog,"\n Se continúa el programa con los par metros por defecto:");
                        fprintf(FichLog,"\n Frecuencia de muestreo: 100mps\n Resolución: 24bits\n
                        Ganancia: 1\n Código de array: 00");
                        printf("\n El fichero SEISAD18.CFG no existe o su formato es incorrecto.");
                        printf("\n Se continúa el programa con los par metros por defecto:");
                        printf("\n Frecuencia de muestreo: 100mps\n Resolución: 24bits\n Ganancia: 1\n
                        Código de array: 00");
                }
                else
                {
                        fprintf(FichLog,"\n Parámetros de operación, leídos del fichero
                        SEISAD18.CFG:");
                        fprintf(FichLog,"\n Frecuencia de muestreo: %ums\n Resolución: %ubits\n
                        Ganancia: %u\n Código de array: %s", FrecMuestreo, Resolucion, Ganancia,
                        CodigoArr);
                }
                fprintf(FichLog, "\n Hora y fecha de inicio: %02u:%02u:%02u del %u del %u de %u",
                HoraFich, MinutoFich, SegundoFich, DiaFich, MesFich, YearFich);
        }
}

void AbreFichDatos(void)
{
        char NombreFichero[25];

        sprintf(NombreFichero,"C:\\DATA\\%02u%02u%02u%02u.A%s", MesFich, DiaFich, HoraFich,
        MinutoFich, CodigoArr);

        if ((FichDatos = fopen(NombreFichero, "wb")) == NULL)
        {
                clrscr();
                printf("\n Unable to open file %s", NombreFichero);
```

```
                sound(1000);
                delay(2000);
                nosound();
                exit(0);
        }
        else printf("\n Fichero %s abierto para escritura", NombreFichero);
}

void CambiaIndicadorDatos(void)
{
        // Indicador en pantalla de que se están recibiendo datos de las tarjetas SEISAD18: texto 'DATA'
        //moviéndose de derecha a izquierda
        if(++ContIndicadorDatos > 9) ContIndicadorDatos = 0;
        gotoxy(1,11);
        switch(ContIndicadorDatos)
        {
                case 0:
                        printf("      ");
                        break;
                case 1:
                        printf("     D");
                        break;
                case 2:
                        printf("    DA");
                        break;
                case 3:
                        printf("   DAT");
                        break;
                case 4:
                        printf("  DATA");
                        break;
                case 5:
                        printf(" DATA ");
                        break;
                case 6:
                        printf("DATA  ");
                        break;
                case 7:
                        printf("ATA   ");
                        break;
                case 8:
                        printf("TA    ");
                        break;
                case 9:
                        printf("A     ");
        }
        // Indicador con LED de que se están recibiendo datos de las tarjetas SEISAD18: el LED controlado
        //por el bit DATA6 del puerto paralelo (pin 8 en conector DB25 o pin 15 en conector IDC 26) se
        //enciende uno de cada cuatro segundos:
        if ((++ContLEDDatos % 4) == 0) outportb(portpar, (inportb(portpar)|0x40));
        else outportb(portpar, (inportb(portpar)&0xBF));
}
```

```
void CambiaIndicadorGPS(void)
{
	// Indicador en pantalla de que se están recibiendo datos del GPS: texto 'GPS' moviéndose de derecha
	//a izquierda
	if(++ContIndicadorGPS > 9) ContIndicadorGPS = 0;
	gotoxy(1,12);
	switch(ContIndicadorGPS)
	{
		case 0:
			printf("      ");
			break;
		case 1:
			printf("     G");
			break;
		case 2:
			printf("    GP");
			break;
		case 3:
			printf("   GPS");
			break;
		case 4:
			printf("  GPS ");
			break;
		case 5:
			printf(" GPS  ");
			break;
		case 6:
			printf("GPS   ");
			break;
		case 7:
			printf("PS    ");
			break;
		case 8:
			printf("S     ");
			break;
		case 9:
			printf("      ");
	}
	// Indicador con LED de que se están recibiendo datos del GPS: el LED controlado por el bit DATA7
	//del puerto paralelo (pin 9 en conector DB25 o pin 17 en conector IDC 26) se enciende uno de cada
	//cuatro segundos:
	if ((++ContLEDGPS % 4) == 0) outportb(portpar, (inportb(portpar)|0x80));
	else outportb(portpar, (inportb(portpar)&0x7F));
}

unsigned char PPS_1(void)
{
	if (((inportb(portpar1)) & 0x40) == 0x40)     return(TRUE);
	else return(FALSE);
}

void CompruebaPPS_Polling(void)
{
	struct  time t;
```

```
	unsigned char ContSegundos, SegundoAnterior;

	printf("\n Comprobando PPS...");
	ContSegundos = 0;
	gettime(&t);
	SegundoAnterior = t.ti_sec;
	do
	{
		Flag.HayPPS = FALSE;
		// Espera a que la señal de PPS esté a 1...
		do
		{
			SegundoAnterior = t.ti_sec;
			gettime(&t);
			if (SegundoAnterior != t.ti_sec) ContSegundos++;
			SalidaManual();// Comprueba si se pulsa alguna tecla o el pulsador de salida
		}
		while((!PPS_1()) && (ContSegundos < 5) && (!Flag.SalidaManual));
		//...luego a que esté a 0...
		do
		{
			SegundoAnterior = t.ti_sec;
			gettime(&t);
			if (SegundoAnterior != t.ti_sec) ContSegundos++;
			SalidaManual();
		}
		while((PPS_1()) && (ContSegundos < 5) && (!Flag.SalidaManual));
		//... y a que vuelva a 1 de nuevo
		do
		{
			SegundoAnterior = t.ti_sec;
			gettime(&t);
			if (SegundoAnterior != t.ti_sec) ContSegundos++;
			SalidaManual();
		}
		while((!PPS_1()) && (ContSegundos < 5) && (!Flag.SalidaManual));
		if((ContSegundos < 5) && (!Flag.SalidaManual)) Flag.HayPPS = TRUE;
	}
	while((ContSegundos < 5) && (!Flag.HayPPS) && (!Flag.SalidaManual));
}

void TomaHoraPC(void)
{
	struct time t;

	gettime(&t);
	Hora = t.ti_hour;
	Minuto = t.ti_min;
	Segundo = t.ti_sec;
}

void TomaFechaPC(void)
{
	struct date d;

	getdate(&d);
```

```
	Year = d.da_year;
	Dia = d.da_day;
	Mes = d.da_mon;
}

void SincronizaHoraPC(void)
{
	unsigned char ContErrores = 0, HoraBuena, ErrorGPS;
	struct time TiempoGPS;
	struct date FechaGPS;

	do
	{
		// No se escribe nada en el LOG porque todavía no está abierto
		printf("\n Sincronizando el reloj del PC con el GPS...");
		// Espera flanco descendente PPS
		do
		{
		}
		while((PPS_1()) && (!kbhit()));
		// Inicializa buffers de COM2 (GPS)
		Princ_Buf_Recepcion2 = Fin_Buf_Recepcion2 = 0;
		// Espera a que haya en el búffer toda la información que necesitamos (hasta el byte 59)
		do
		{
		}
		while((Fin_Buf_Recepcion2 < 59) && (!kbhit()));
		// Comprueba que es una trama de tiempo ($GPRMC) y que la posición es válida ('A')
		ErrorGPS = FALSE;
		if ((Buffer_Recepcion2[0] != '$') || (Buffer_Recepcion2[1] != 'G') ||
		(Buffer_Recepcion2[2] != 'P') || (Buffer_Recepcion2[3] != 'R') ||
		(Buffer_Recepcion2[4] != 'M') || (Buffer_Recepcion2[5] != 'C') ||
		(Buffer_Recepcion2[14] != 'A'))
		{
			ErrorGPS = TRUE;
			ContErrores++;
		}
		else
		{
			ErrorGPS = FALSE;
			HoraBuena = FALSE;
			// Mete la información del tiempo GPS en las variables que se usarán para
			//sincronizar el reloj del PC
			TiempoGPS.ti_hund = 0;
			TiempoGPS.ti_hour = (((Buffer_Recepcion2[7]-'0')*10)+
			(Buffer_Recepcion2[8]-'0'));
			TiempoGPS.ti_min = (((Buffer_Recepcion2[9]-'0')*10)+
			(Buffer_Recepcion2[10]-'0'));
			TiempoGPS.ti_sec = (((Buffer_Recepcion2[11]-'0')*10)+
			(Buffer_Recepcion2[12]-'0'));
			FechaGPS.da_day = (((Buffer_Recepcion2[53]-'0')*10)+
			(Buffer_Recepcion2[54]-'0'));
			FechaGPS.da_mon = (((Buffer_Recepcion2[55]-'0')*10)+
			(Buffer_Recepcion2[56]-'0'));
			FechaGPS.da_year = (2000+((Buffer_Recepcion2[57]-'0')*10)+
			(Buffer_Recepcion2[58]-'0'));
```

```
                        // También mete la información GPS en las variables del reloj del programa (el
                        //reloj del PC no se usa para nada)
                        Hora = TiempoGPS.ti_hour;
                        Minuto = TiempoGPS.ti_min;
                        Segundo = TiempoGPS.ti_sec;
                        Dia = FechaGPS.da_day;
                        Mes = FechaGPS.da_mon;
                        Year = FechaGPS.da_year;
                }
                // Si el segundo es 59 no se actualiza el reloj, porque incrementarlo supone incrementar el
                //minuto y puede que la hora, día, mes y año.
                if (TiempoGPS.ti_sec != 59)
                {
                        TiempoGPS.ti_sec++;
                        HoraBuena = TRUE;
                        // Espera el siguiente PPS (flanco descendente) para sincronizar el reloj del PC
                        do
                        {
                        }
                        while(!PPS_1());
                        do
                        {
                        }
                        while(PPS_1());
                        settime(&TiempoGPS);
                        setdate(&FechaGPS);
                }
        }
        while((ErrorGPS) && (ContErrores < 5) && (!HoraBuena));
        // No se puede escribir nada en el LOG porque todavía no está abierto
        if (ContErrores == 5) Flag.SincGPS = FALSE;
        else Flag.SincGPS = TRUE;
}

void InicSEISAD18(void)
{
        unsigned char i, NumPulsos;
        // 115200bps, 100mps y ganancia 1, 4 o 16 (29 pulsos para ganancia 1, 30 para ganancia 4 y 31 para
        //ganancia 16) (de momento la frecuencia de muestreo y resolución no son configurables):
        switch (Ganancia)
        {
                case 1:
                        NumPulsos = 29;
                        break;
                case 4:
                        NumPulsos = 30;
                        break;
                case 16:
                        NumPulsos = 31;
                        break;
                default:
                        NumPulsos = 29;// Por defecto, ganancia 1
        }
        for (i=0; i<NumPulsos; i++)
        {
                MandarCaracter(COM1, 0x00);
```

```
	}
	fprintf(FichLog, "\n Tarjetas SEISAD18 inicializadas a 115200bps, 100mps, ganancia %u",
	Ganancia);
	printf("\n Tarjetas SEISAD18 inicializadas a 115200bps, 100mps, ganancia %u", Ganancia);
}

void IncVarsFich(void)
{
	SegundoFich++;
	if (SegundoFich == 60)
	{
		SegundoFich = 0;
		MinutoFich++;
		if (MinutoFich == 60)
		{
			MinutoFich = 0;
			HoraFich++;
			if (HoraFich == 24)
			{
				HoraFich = 0;
				DiaFich++;
				switch (MesFich)
				{
					// Meses de 31 días (salvo diciembre):
					case 1:
					case 3:
					case 5:
					case 7:
					case 8:
					case 10:
						if (DiaFich == 32)
						{
							DiaFich = 1;
							MesFich++;
						}
						break;
					// Diciembre:
					case 12:
						if (DiaFich == 32)
						{
							DiaFich = 1;
							MesFich = 1;
							YearFich++;
						}
						break;
					// Meses de 30 días:
					case 4:
					case 6:
					case 9:
					case 11:
						if (DiaFich == 31)
						{
							DiaFich = 1;
							MesFich++;
						}
						break;
```

```
                                        // Febrero:
                                        case 2:
                                                if (YearFich % 4 == 0) // Años bisiestos
                                                {
                                                        if (DiaFich == 30)
                                                        {
                                                                DiaFich = 1;
                                                                MesFich++;
                                                        }
                                                }
                                                else // Años no bisiestos
                                                {
                                                        if (DiaFich == 29)
                                                        {
                                                                DiaFich = 1;
                                                                MesFich++;
                                                        }
                                                }
                                        }
                                }
                        }
                }
        }
}

void ActualizaVarsFich(void)
{
        SegundoFich = Segundo;
        MinutoFich = Minuto;
        HoraFich = Hora;
        DiaFich = Dia;
        MesFich = Mes;
        YearFich = Year;
}

unsigned char SalidaManual(void)
{
        // Si se pulsa una tecla o el botón de salida (conectado entre la señal INIT (pin 16 de conector DB25
        //o 6 de conector IDC26) y GND) devuelve TRUE (la señal INIT está normalmente a 1)
        if ((kbhit()) || (((inportb(portpar2)) & 0x04) != 0x04))
        {
                if (kbhit()) getch();// Para limpiar el búffer de teclado
                Flag.SalidaManual = TRUE;
                // No se escribe nada en el LOG porque puede que todavía no esté abierto
                return(TRUE);
        }
        return(FALSE);
}
```

```
/*************************************************************************
  Programa principal
*************************************************************************/
void main(void)
{
        clrscr();
        printf(" Programa Arrays8.C\n");
        // Si hay señal de PPS se sincroniza el reloj del PC con el GPS y se prosigue. En caso contrario se
        //sale del programa, indicándolo antes en el fichero LOG (función 'AbreFicheroLOG()').
        CompruebaPPS_Polling();
        if(Flag.HayPPS)
        {
                InicializarComunicacionSerie(COM1,BAUD115200,NINGUNA,LON8,STOP1);
                InicializarComunicacionSerie(COM2,BAUD9600,NINGUNA,LON8,STOP1);
                Princ_Buf_Recepcion1 = Fin_Buf_Recepcion1 = ContBytesEscritos1 = 0;
                Princ_Buf_Recepcion2 = Fin_Buf_Recepcion2 = ContBytesEscritos2 = 0;
                EspacioLibre = 33500000;
                SincronizaHoraPC();     // Tiene que hacerse después de inicializar la comunicación serie
                                        // en COM2
        }
        // Inicializa las variables con las que se nombrar n los ficheros a partir de las del reloj del programa,
        //que a su vez se ha sincronizado con el GPS
        ActualizaVarsFich();
        // Lee los parámetros de operación del fichero de configuración
        LeeFichCfg();
        // Abre los primeros ficheros LOG y de datos
        AbreFicheroLOG();
        AbreFichDatos();
        // Inicializa las tarjetas SEISAD18 con los par metros leídos o por defecto
        InicSEISAD18();

        // Antes de entrar en el bucle principal espera a que el PPS esté a 1, para asegurarse de que todo el
        //proceso está sincronizado desde el primer segundo
        do
        {
        }
        while(!PPS_1());

        ContPPS = 0;

        // Bucle principal
        do
        {
                // Espera flanco descendente PPS
                do
                {
                }
                while(PPS_1());
                // Inicializa búffers
                Princ_Buf_Recepcion1 = Fin_Buf_Recepcion1 = 0;
                Princ_Buf_Recepcion2 = Fin_Buf_Recepcion2 = 0;
                ContPPS++;
                // Si empieza una hora, se abren nuevos ficheros LOG y de datos:
                if ((Minuto == 59) && (Segundo == 59))
                {
```

```
            fprintf(FichLog, "\n Contador PPS: %u", ContPPS);
            printf("\n Contador PPS: %u", ContPPS);
            ContPPS = 0;

            fclose(FichLog);
            fclose(FichDatos);
            // Incrementa en un segundo las variables con las que se nombrarán los nuevos
            //ficheros (para que el primer segundo de cada fichero sea el 0 del minuto 0)
            ActualizaVarsFich();
            IncVarsFich();
            AbreFicheroLOG();
            AbreFichDatos();
}
// Espera a que haya algún carácter en el búffer de COM1 (datos)
do
{
}
while((Fin_Buf_Recepcion1 < 3) && (!kbhit()));
CambiaIndicadorDatos();
Flag.ErrorTramaDatos = FALSE;
Flag.ErrorBufferDatos = FALSE;
// Antes de analizar los datos recibidos, guarda la posición actual del puntero al fichero por
//si hay que volver a ella, así como el contador de bytes de datos escritos
fgetpos(FichDatos, &filepos);
ContProvisional1 = ContBytesEscritos1;

// Error de trama de datos (codigo de error: '1')
if ((Buffer_Recepcion1[0] != 0x55) || (Buffer_Recepcion1[1] != 0x55) ||
(Buffer_Recepcion1[2] != 0x55))
{
            Flag.ErrorTramaDatos = TRUE;
            TipoError = '1';
            for(i=0; i<10; i++)
            {
                        fputc(TipoError, FichDatos);
                        ContBytesEscritos1++;
            }
            //TomaHoraPC();
            fprintf(FichLog, "\n %02u:%02u:%02u Error trama datos: '%c' '%c' '%c'", Hora,
            Minuto, Segundo, Buffer_Recepcion1[0], Buffer_Recepcion1[1],
            Buffer_Recepcion1[2]);
}
// En caso de error en el formato de datos, espera a que la señal de PPS vuelva a 1, para
//evitar que vuelva a entrar en la comprobación de la cabecera mientras está a 0.
do
{
}
while(!PPS_1());

// Si no hay error de trama espera hasta que se reciben todos los datos por COM1 o hasta
//que llega un nuevo PPS
if (!Flag.ErrorTramaDatos)
{
            do
            {
                        if(Fin_Buf_Recepcion1 != Princ_Buf_Recepcion1)
```

```
                {
                        fputc(Buffer_Recepcion1[Princ_Buf_Recepcion1], FichDatos);
                        if (++Princ_Buf_Recepcion1 == Tamano_Buf_Recepcion1)
                        Princ_Buf_Recepcion1 = 0;
                        ContBytesEscritos1++;
                }
        }
        while((Princ_Buf_Recepcion1 < 4800) && (PPS_1()));

        // Error de buffer de datos (codigo de error: '2')
        if(Princ_Buf_Recepcion1 < 4800)
        {
                Flag.ErrorBufferDatos = TRUE;
                TipoError = '2';
                // Se vuelve a la posición y al valor del contador de bytes que se habían
                //guardado antes:
                fsetpos(FichDatos, &filepos);
                ContBytesEscritos1 = ContProvisional1;
                for(i=0; i<10; i++)
                {
                        fputc(TipoError, FichDatos);
                        ContBytesEscritos1++;
                }
                //TomaHoraPC();
                fprintf(FichLog, "\n %02u:%02u:%02u Error buffer datos (Pos. Buffer:
                %u)", Hora, Minuto, Segundo, Princ_Buf_Recepcion1);
        }
}
// Si no ha habido error de trama de datos ni error de buffer de datos, comprueba si están
//los bytes que necesitamos (los 59 primeros de la trama)
if ((!Flag.ErrorTramaDatos) && (!Flag.ErrorBufferDatos))
{
        Flag.ErrorBufferGPS = FALSE;
        Flag.ErrorTramaGPS = FALSE;
        if (Fin_Buf_Recepcion2 < 58)
        {
                // Error de buffer gps (codigo de error: '4')
                Flag.ErrorBufferGPS = TRUE;
                TipoError = '4';
                for(i=0; i<10; i++)
                {
                        fputc(TipoError, FichDatos);
                        ContBytesEscritos2++;
                }
                //TomaHoraPC();
                fprintf(FichLog, "\n %02u:%02u:%02u Error buffer GPS (Pos. Buffer:
                %u)", Hora, Minuto, Segundo, Fin_Buf_Recepcion2);
        }
        else CambiaIndicadorGPS();

        // Si no hay error de búffer, comprueba el primer carácter del búffer de COM2
        if (!Flag.ErrorBufferGPS)
        {
                // ERROR DE TRAMA GPS (CODIGO DE ERROR: '3')
                if ((Buffer_Recepcion2[0]) != 0x24)
                {
```

```
                    Flag.ErrorTramaGPS = TRUE;
                    TipoError = '3';
                    for(i=0; i<10; i++)
                    {
                        fputc(TipoError, FichDatos);
                        ContBytesEscritos2++;
                        //TomaHoraPC();
                    }
                    fprintf(FichLog, "\n %02u:%02u:%02u Error trama GPS:
                    0x%02X '%c'", Hora, Minuto, Segundo, Buffer_Recepcion2[0],
                    Buffer_Recepcion2[0]);
                }
            }
            if ((!Flag.ErrorBufferGPS) && (!Flag.ErrorTramaGPS))
            {
                // Para evitar los errores que se producen si el GPS envía dos tramas de
                //tiempo en un segundo (si hay dos tramas en el mismo segundo, se queda
                //con la última):
                if (Fin_Buf_Recepcion2 > 130)
                {
                    // Status GPS ('A': posición válida, 'V': warning)
                    fputc(Buffer_Recepcion2[86], FichDatos);
                    // Hora[0]
                    fputc(Buffer_Recepcion2[79], FichDatos);
                    // Hora[1]
                    fputc(Buffer_Recepcion2[80], FichDatos);
                    // Minuto[0]
                    fputc(Buffer_Recepcion2[81], FichDatos);
                    // Minuto[1]
                    fputc(Buffer_Recepcion2[82], FichDatos);
                    // Segundo[0]
                    fputc(Buffer_Recepcion2[83], FichDatos);
                    // Segundo[1]
                    fputc(Buffer_Recepcion2[84], FichDatos);
                    // Día[0]
                    fputc(Buffer_Recepcion2[125], FichDatos);
                    // Día[1]
                    fputc(Buffer_Recepcion2[126], FichDatos);
                    // Mes[0]
                    fputc(Buffer_Recepcion2[127], FichDatos);
                    // Mes[1]
                    fputc(Buffer_Recepcion2[128], FichDatos);
                    // Año[0]
                    fputc(Buffer_Recepcion2[129], FichDatos);
                    // Año[1]
                    fputc(Buffer_Recepcion2[130], FichDatos);
                    // Actualiza el reloj con la información del GPS:
                    Year = 2000+((Buffer_Recepcion2[129]-'0')*10)+
                    (Buffer_Recepcion2[130]-'0');
                    Mes = ((Buffer_Recepcion2[127]-'0')*10)+
                    (Buffer_Recepcion2[128]-'0');
                    Dia = ((Buffer_Recepcion2[125]-'0')*10)+
                    (Buffer_Recepcion2[126]-'0');
                    Hora = ((Buffer_Recepcion2[79]-'0')*10)+
                    (Buffer_Recepcion2[80]-'0');
```

```
                                        Minuto = ((Buffer_Recepcion2[81]-'0')*10)+
                                        (Buffer_Recepcion2[82]-'0');
                                        Segundo = ((Buffer_Recepcion2[83]-'0')*10)+
                                        (Buffer_Recepcion2[84]-'0');
                                }
                                else
                                {
                                        // Status GPS ('A': posición v lida, 'V': warning)
                                        fputc(Buffer_Recepcion2[14], FichDatos);
                                        // Hora[0]
                                        fputc(Buffer_Recepcion2[7], FichDatos);
                                        // Hora[1]
                                        fputc(Buffer_Recepcion2[8], FichDatos);
                                        // Minuto[0]
                                        fputc(Buffer_Recepcion2[9], FichDatos);
                                        // Minuto[1]
                                        fputc(Buffer_Recepcion2[10], FichDatos);
                                        // Segundo[0]
                                        fputc(Buffer_Recepcion2[11], FichDatos);
                                        // Segundo[1]
                                        fputc(Buffer_Recepcion2[12], FichDatos);
                                        // Día[0]
                                        fputc(Buffer_Recepcion2[53], FichDatos);
                                        // Día[1]
                                        fputc(Buffer_Recepcion2[54], FichDatos);
                                        // Mes[0]
                                        fputc(Buffer_Recepcion2[55], FichDatos);
                                        // Mes[1]
                                        fputc(Buffer_Recepcion2[56], FichDatos);
                                        // Año[0]
                                        fputc(Buffer_Recepcion2[57], FichDatos);
                                        // Año[1]
                                        fputc(Buffer_Recepcion2[58], FichDatos);
                                        // Actualiza el reloj con la información del GPS:
                                        Year = 2000+((Buffer_Recepcion2[57]-'0')*10)+
                                        (Buffer_Recepcion2[58]-'0');
                                        Mes = ((Buffer_Recepcion2[55]-'0')*10)+
                                        (Buffer_Recepcion2[56]-'0');
                                        Dia = ((Buffer_Recepcion2[53]-'0')*10)+
                                        (Buffer_Recepcion2[54]-'0');
                                        Hora = ((Buffer_Recepcion2[7]-'0')*10)+
                                        (Buffer_Recepcion2[8]-'0');
                                        Minuto = ((Buffer_Recepcion2[9]-'0')*10)+
                                        (Buffer_Recepcion2[10]-'0');
                                        Segundo = ((Buffer_Recepcion2[11]-'0')*10)+
                                        (Buffer_Recepcion2[12]-'0');
                                }
                                ContBytesEscritos2 += 13;
                                // Cada seis horas sale del programa para volcar los ficheros grabados
                                //al disco USB:
                                if((Minuto == 59) && (Segundo == 59) && (Hora % 6 == 5))
                                Flag.DatosADiscoUSB = TRUE;
                        }
                }
        }
while ((!SalidaManual()) && (!Flag.DatosADiscoUSB));
```

```
        FinalizarComunicacionSerie();
        fprintf(FichLog, "\n Hora y fecha de salida del programa: %02u:%02u:%02u del %u del %u de %u",
        Hora, Minuto, Segundo, Dia, Mes, Year);
        fprintf(FichLog,"\n Contador PPS: %u", ContPPS);
        printf("\n Contador PPS: %u", ContPPS);
        fclose(FichDatos);
        fclose(FichLog);
        // Apaga los LEDs de recepción de datos y recepción de GPS
        outportb(portpar, (inportb(portpar)&0xBF));
        outportb(portpar, (inportb(portpar)&0x7F));
        printf("\n\n");
        // Para gestionar ejecución continua mediante un fichero BAT: si se ha pulsado una tecla o el botón
        //de salida, dará ERRORLEVEL 1, en caso contrario ERRORLEVEL 0
        if (Flag.SalidaManual)
        {
                Flag.SalidaManual = FALSE;
                fprintf(FichLog, "\n %02u:%02u:%02u: Salida manual del programa", Hora, Minuto,
                Segundo);
                printf("\n %02u:%02u:%02u: Salida manual del programa", Hora, Minuto, Segundo);
                exit(1);  // Código de salida 1, para no volver a lanzar el programa de adquisición desde el
                        //fichero ARRAYS.BAT
        }
        else exit(0);
}
```

III.A.1.d. PROGRAMAS DE ENCENDIDO Y APAGADO DEL LED DE ESCRITURA

Ficheros: LEDWRON.C y LEDWROFF.C

/* Programas para encender y apagar el LED de escritura en disco USB. Su usan en el fichero de arranque ARRAYS.BAT, para monitorizar el volcado de datos desde el disco duro interno al disco USB. */

```
/*************************************************************************
  Programa LEDWrOn.C
*************************************************************************/
#define portpar    888

#include <dos.h>

void main(void)
{
        // Pone a 1 el bit DATA5 del puerto paralelo
        outportb(portpar, (inportb(portpar)|0x20));
}
```

```
/*************************************************************************
  Programa LEDWrOff.C
*************************************************************************/
#define portpar    888

#include <dos.h>

void main(void)
{
        // Pone a 0 el bit DATA5 del puerto paralelo
        outportb(portpar, (inportb(portpar)&0xDF));
}
```

III.A.1.e. PROGRAMAS DE PRUEBA DE LOS DISTINTOS COMPONENTES DE LAS ANTENAS PORTÁTILES DEL IAG

Ficheros: P_BOTON.C, P_PPS.C, LEDSON.C, LEDSOFF.C y P_COMRX2.C

/* Programas de test de los distintos componentes de las nuevas antenas portátiles del IAG:

P_BOTON.C prueba el pulsador de salida del programa y volcado de datos (*Exit program'*). Al ejecutarlo aparecerá en pantalla una vez por segundo el texto 'No pulsado', cambiando a 'PULSADO' cuando se presiona el pulsador.

P_PPS.C verifica la correcta recepción de los pulsos de segundo del GPS en el PC. Si la recepción es correcta parpadearán el mensaje 'PPS' en la pantalla y los LEDs *GPS In*, *Data In* y *USB Disk.*

LEDSON.C y LEDSOFF.C comprueban el funcionamiento de los LEDs conectados al puerto paralelo (*GPS In, Data In* y *USB Disk*). El primero los enciende y el segundo los apaga.

P_COMRX2.C comprueba la recepción de caracteres por los puertos serie. Permite verificar el formato y número de datos recibidos, tanto procedentes del receptor GPS como de las tarjetas SEISAD18 (ver sección 5.6.11 para más detalles).

```
/*************************************************************************
  Programa P_BOTON.C
*************************************************************************/
#define  TRUE          1
#define  FALSE         0
#define  portpar       888
#define  portpar1      portpar + 1
#define  portpar2      portpar + 2

#include <dos.h>
#include <stdio.h>
#include <conio.h>

unsigned char Init(void)
{
        if (((inportb(portpar2)) & 0x04) == 0x04) return(TRUE);
        else return(FALSE);
}

void main(void)
{
        clrscr();
        printf(" Programa P_BOTON.C\n");
        do
        {
                if (Init()) printf("\n No pulsado");
                else printf("\n PULSADO");
                delay(1000);
        }
        while(!kbhit());
}
```

```
/*****************************************************************************
  Programa P_PPS.C
*****************************************************************************/
#define  TRUE                    1
#define  FALSE                   0
#define  portpar                 888
#define  portpar1                portpar + 1
#define  portpar2                portpar + 2
#define  portpar3                portpar + 7

#include <dos.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

unsigned char PPS_1(void);
void TestPPS(void);
unsigned char SalidaManual(void);

unsigned char PPS_1(void)
{
        if (((inportb(portpar1)) & 0x40) == 0x40) return(TRUE);
        else return(FALSE);
}

void TestPPS(void)
{
        do
        {
                do
                {
                }
                while((!PPS_1()) && (!SalidaManual()));
                gotoxy(1,10);
                printf("PPS");
                // Enciende los tres LEDs:
                // Pone a 1 el bit DATA5 del puerto paralelo
                outportb(portpar, (inportb(portpar)|0x20));
                // Pone a 1 el bit DATA6 del puerto paralelo
                outportb(portpar, (inportb(portpar)|0x40));
                // Pone a 1 el bit DATA7 del puerto paralelo
                outportb(portpar, (inportb(portpar)|0x80));
                do
                {
                }
                while((PPS_1()) && (!SalidaManual()));
                gotoxy(1,10);
                printf("   ");
                // Apaga los tres LEDs:
                // Pone a 0 el bit DATA5 del puerto paralelo
                outportb(portpar, (inportb(portpar)&0xDF));
                // Pone a 0 el bit DATA6 del puerto paralelo
                outportb(portpar, (inportb(portpar)&0xBF));
                // Pone a 0 el bit DATA7 del puerto paralelo
```

```
                outportb(portpar, (inportb(portpar)&0x7F));
        }
        while(!SalidaManual());
}

unsigned char SalidaManual(void)
{
        // Si se pulsa una tecla o el botón de salida (conectado entre la señal INIT (pin 16 de conector DB25
        //o 6 de conector IDC26) y GND) devuelve TRUE (la señal INIT está normalmente a 1)
        if ((kbhit()) || (((inportb(portpar2)) & 0x04) != 0x04))
        {
                if (kbhit()) getch();           // Para limpiar el búffer de teclado
                return(TRUE);
        }
        return(FALSE);
}

void main(void)
{
        clrscr();
        printf(" Programa P_PPS.C: comprueba la recepción de la señal de un pulso por segundo\ndel
        GPS.\n");
        printf("\n Si la recepción es correcta, el mensaje 'PPS' en pantalla y los tres LEDs\nconectados al
        puerto paralelo del PC parpadearán.\n");
        printf(" Pulsar cualquier tecla o el botón de salida para terminar.\n");
        TestPPS();
}

/*****************************************************************************
```

Programa LEDsOn.C

```
*****************************************************************************/
#define portpar    888
#define portpar1 portpar + 1
#define portpar2 portpar + 2
#define portpar3 portpar + 7

#include <dos.h>

void main(void)
{
        // Pone a 1 el bit DATA5 del puerto paralelo
        outportb(portpar, (inportb(portpar)|0x20));

        // Pone a 1 el bit DATA6 del puerto paralelo
        outportb(portpar, (inportb(portpar)|0x40));

        // Pone a 1 el bit DATA7 del puerto paralelo
        outportb(portpar, (inportb(portpar)|0x80));
}
```

```
/*****************************************************************************
  Programa LEDsOff.C
*****************************************************************************/
#define portpar    888
#define portpar1 portpar + 1
#define portpar2 portpar + 2
#define portpar3 portpar + 7

#include <dos.h>

void main(void)
{
        // Pone a 0 el bit DATA5 del puerto paralelo
        outportb(portpar, (inportb(portpar)&0xDF));

        // Pone a 0 el bit DATA6 del puerto paralelo
        outportb(portpar, (inportb(portpar)&0xBF));

        // Pone a 0 el bit DATA7 del puerto paralelo
        outportb(portpar, (inportb(portpar)&0x7F));
}
```

```
/*****************************************************************************
  Programa P_COMRx2.C
*****************************************************************************/
#define  TRUE                   1
#define  FALSE                  0
#define  Tamano_Buf_Recepcion   50000

#include <miserie.h>
#include <dos.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

unsigned int i;
unsigned char Temporal, NumMods, Frecuencia, Ganancia;
int BaudRate, PuertoSerie;
unsigned char ValorCorrecto;
unsigned char CheckSum1, CheckSum2, CheckSum3;
unsigned char CheckSumFinal1, CheckSumFinal2, CheckSumFinal3;
unsigned char ByteRecibido;
FILE *FichDatos;
unsigned long int ContBytesEscritos = 0;
unsigned long int EspacioLibre;

void Presentacion(void);
void PideParametros(void);
int Confirmacion(void);
void PantallaRecepcion(void);
void AbreFichDatos(void);
```

```
void Presentacion(void)
{
	clrscr();
	printf("\n Program P_COMRX2.C");
	printf("\n\n Data reception through serial port \n\n");
	printf("\n Data will be written to file D:\DATA.BIN");
	printf("\n D: must be defined as a virtual disk drive");
	printf("\n\n Press 'Q' to quit, any other key to continue\n");
	if (tolower(getch()) == 'q')
	{
		clrscr();
		exit(0);
	}
	delay(500);
}

void PideParametros(void)
{
	// Pide número de puerto serie:
	do
	{
		clrscr();
		printf(" Input serial port number [1, 2]: ");
		Temporal = getch();
		ValorCorrecto = TRUE;// Por defecto, el valor introducido es correcto
		switch (Temporal)
		{
			case '1':
				PuertoSerie = COM1;// Constantes definidas en 'miserie.h'
				break;
			case '2':
				PuertoSerie = COM2;
				break;
			default:
				ValorCorrecto = FALSE;
				printf(" Illegal value, try again");
				delay(2000);
		}
	}
	while (!ValorCorrecto);
	printf("\n\n Serial Port = COM%u", Temporal-48);// 48 = Cód. ASCII del 0
	delay(1000);

	// Pide velocidad de transmisión:
	do
	{
		clrscr();
		printf(" Input serial port baud rate:\n\n 0.........9600bps\n 1.........19200bps\n
		2.........115200bps\n");
		Temporal = getch();
		switch (Temporal)
		{
			case '0':
				BaudRate = BAUD9600;// Constantes definidas en 'miserie.h'
				printf("\n Serial Port Baud Rate = 9600bps");
				break;
```

```
                        case '1':
                                BaudRate = BAUD19200;
                                printf("\n Serial Port Baud Rate = 19200bps");
                                break;
                        case '2':
                                BaudRate = BAUD115200;
                                printf("\n Serial Port Baud Rate = 115200bps");
                                break;
                        default:
                                BaudRate = 0;
                                printf("\n Illegal value, try again");
                }
                delay(1000);
        }
        while (BaudRate == 0);
}

int Confirmacion(void)
{
        clrscr();
        printf("\n Selected parameters are: \n");
        switch (PuertoSerie)
        {
                case COM1:
                        printf("\n Serial Port: COM1");
                        break;
                case COM2:
                        printf("\n Serial Port: COM2");
                        break;
                case COM3:
                        printf("\n Serial Port: COM3");
                        break;
                case COM4:
                        printf("\n Serial Port: COM4");
        }
        switch (BaudRate)
        {
                case BAUD9600:
                        printf("\n Serial Port Baud Rate = 9600bps");
                        break;
                case BAUD19200:
                        printf("\n Serial Port Baud Rate = 19200bps");
                        break;
                case BAUD115200:
                        printf("\n Serial Port Baud Rate = 115200bps");
        }
        printf("\n\n\n Press [Y] to accept these parameters or any other key to input new ones: ");

        if (toupper(getche()) == 'Y') return(OK);
        else return(MAL);
}
```

```
void PantallaRecepcion(void)
{
	clrscr();
	switch (PuertoSerie)
	{
		case COM1:
			printf("\n\n COM1");
			break;
		case COM2:
			printf("\n\n COM2");
			break;
		case COM3:
			printf("\n\n COM3");
			break;
		case COM4:
			printf("\n\n COM4");
	}
	switch (BaudRate)
	{
		case BAUD9600:
			printf("\n 9600bps");
			break;
		case BAUD19200:
			printf("\n 19200bps");
			break;
		case BAUD115200:
			printf("\n 115200bps");
	}
	printf("\n\n Receiving data, wait until buffer is full or press any key to quit");
}

void AbreFichDatos(void)
{
	if ((FichDatos = fopen("D:\DATA.BIN", "wb")) == NULL)
	{
		clrscr();
		printf("\n Unable to open file D:\DATA.BIN");
		printf("\n D: must have been defined as a virtual disk drive");
		sound(1000);
		delay(2000);
		nosound();
		exit(0);
	}
	else printf("\n File D:\DATA.BIN opened for writting");
}
```

```
void main(void)
{
	Presentacion();
	do
	{
		PideParametros();
	}
	while (Confirmacion() != OK);
	AbreFichDatos();
	InicializarComunicacionSerie(PuertoSerie,BaudRate,NINGUNA,LON8,STOP1);
	PantallaRecepcion();
	Princ_Buf_Recepcion = Fin_Buf_Recepcion = ContBytesEscritos = 0;
	EspacioLibre = 32000000; //Debe ser menor o igual que el tamaño del disco virtual D (31MB =
				//= 32505856 bytes)
	do
	{
		if(Fin_Buf_Recepcion != Princ_Buf_Recepcion)
		{
			fputc(Buffer_Recepcion[Princ_Buf_Recepcion], FichDatos);
			if (++Princ_Buf_Recepcion==Tamano_Buf_Recepcion) Princ_Buf_Recepcion= 0;
			ContBytesEscritos++;
		}
	}
	while((!kbhit()) && (ContBytesEscritos < EspacioLibre));
	fclose(FichDatos);
	FinalizarComunicacionSerie();
	clrscr();
	printf("\n Data have been written to file D:\DATA.BIN\n\n");
}
```

III.A.1.f. FICHERO DE CONFIGURACIÓN PARA EL PROGRAMA DE VISUALIZACIÓN DE LOS FICHEROS DE DATOS DE LAS ANTENAS PORTÁTILES DEL IAG

Fichero: CHKARR4.CFG

/* Fichero de configuración para el programa de visualización de los ficheros de datos CHKARR4.C (anexo III.A.1.g). */

```
/*************************************************************************
  Fichero
*************************************************************************/
Resolucion        24
ColorFondo        1
ColorLíneasRef    9
ColorLíneaCent    9
ColorSeñal        14
ColorMarco        15
ColorUnidadesY    2
Color0            10
ColorTitulo       15
ColorTexto        7
```

La única condición en el fichero de configuración es que las primeras líneas tengan el formato mostrado. Después puede insertarse cualquier texto de información (como éste) sin ningún formato predefinido ni limitación en la extensión.

En esta versión de los sistemas el valor de la resolución es fijo (24 bits), pero se ha incluido como parámetro para permitir, en futuras versiones, representar datos de 16 bits de resolución.

Los códigos correspondientes a los posibles colores son:

0 NEGRO
1 AZUL
2 VERDE
3 CYAN
4 ROJO
5 MAGENTA
6 MARRN
7 GRIS CLARO
8 GRIS OSCURO
9 AZUL CLARO
10 VERDE CLARO
11 CYAN CLARO
12 ROJO CLARO
13 MAGENTA CLARO
14 AMARILLO
15 BLANCO

Si este fichero (CHKARR4.CFG) no se encuentra en el directorio de trabajo, su formato es incorrecto o el valor de alguno de los parámetros no se encuentra entre los especificados, se tomarán los siguientes valores por defecto:

Resolución	24
ColorFondo	1
ColorLíneasRef	9
ColorLíneaCent	11
ColorSeñal	14
ColorMarco	15
ColorUnidadesY	2
Color0	10
ColorTitulo	15
ColorTexto	7

III.A.1.g. PROGRAMA DE VISUALIZACIÓN DE LOS FICHEROS DE DATOS DE LAS ANTENAS PORTÁTILES DEL IAG

Fichero: CHKARR4.C

/* Programa para la visualización de los ficheros registrados con las nuevas antenas portátiles del IAG. Los colores de la representación gráfica se pueden cambiar en el fichero ASCII de configuración CHKARR4.CFG (anexo III.A.1.f). El canal que se muestra y la escala de representación en los ejes X y Y se seleccionan durante la visualización de los datos. Existe una ayuda sobre los distintos comandos, a la que se accede pulsando la tecla 'A' durante la operación del programa. */

```
/*************************************************************************
  Macros y ficheros de cabecera
*************************************************************************/
#include <dos.h>
#include <ctype.h>
#include <stdio.h>
#include <sys\stat.h>
#include <stdlib.h>
#include <time.h>
#include <fcntl.h>
#include <alloc.h>
#include <conio.h>
#include <io.h>
#include <string.h>
#include <graphics.h>
#include <dir.h>
#include <process.h>
#include <math.h>

#define FALSE   0
#define TRUE    !FALSE

/*************************************************************************
  Declaración de variables
*************************************************************************/
int ColorMax, XMax, YMax, XesqIA, YesqIA, XesqDD, YesqDD, XTextoVar;
int Ygnd, Y1, Y2, Y3, YSatMas, Y4, Y5, Y6, YSatMenos, YTime;
int XMinSignal, XMaxSignal, YMinSignal, YMaxSignal;
int espacio;
int ColorMarco, ColorFondo, ColorLineasRef, ColorLinea0, ColorUnidadesY, Color0,
        ColorTitulo, ColorTexto, ColorParametros, ColorSignal;
int Resolucion;
char Texto[80], CadenaParam[5];
long Pixel;                     // Número de divisiones del ADC por pixel en pantalla
unsigned int XActual, YActual;
unsigned long CeroADCs;
unsigned long adc[3];
FILE *FichDatos;
unsigned char NombreFichDatos[15];
```

```
unsigned int i;
unsigned char ChkStr[10];
unsigned char LongInfoTiempo;
unsigned char Gain, Frec;          // Variables para la lectura de los parámetros, directamente leídas del
                                   //fichero de datos, no del de configuración.
unsigned long int offset = 0;
unsigned int NumPantalla;
unsigned int YInicial[700];        // Para almacenar la posición en pantalla y en fichero cuando hay un salto
unsigned long PosFichero[700];     //atrás
unsigned char Muestra[700];
unsigned char FlagDatoOK[700];
unsigned char FlagTiempoOK[700];
unsigned char NumMuestra, NumCanal;
unsigned char EscalaX, FactorEscalaX, EscalaY;
unsigned int FactorEscalaY;
unsigned long FondoEscala;
unsigned long NumTotalMuestras;

fpos_t filepos;

struct Indicador
{
        unsigned Fin:           1;
        unsigned FichConfOK:    1;
        unsigned PantallaNueva: 1;
        unsigned DatoNegativo:  1;
        unsigned DatoOK:        1;
        unsigned TiempoOK:      1;
        unsigned TeclaBuena:    1;
} Flag;

/*************************************************************************
  Declaración de funciones
*************************************************************************/
// Funciones gráficas
void ModoGrafico(void);
void DibujarPantalla(void);
void ReponerPantalla(void);
void PantallaAyuda(void);
void PintaSignal(void);
void PintaDataError(void);
// Otras funciones
void InicVars(void);
void Iniciacion(void);
void LeeDato(void);
void TeclaPulsada(void);
void LeerConfiguracion(void);
void ConvierteAPixels(void);
void LeeTiempo(void);
unsigned char CabeceraSegundo(void);
unsigned char ErrorTramaDatos(void);
unsigned char ErrorBufferDatos(void);
void LeerCabecera(void);
void PoneTexto(void);
void PoneValoresEjeY(void);
```

```
void ValoresPantallaInicial(void);
void ValoresPantallaActual(void);

/*****************************************************************************
   Definición de las funciones
*****************************************************************************/
void ModoGrafico(void)
{
int gdriver = DETECT, gmode, errorcode;
        initgraph(&gdriver, &gmode, "");
        errorcode = graphresult(); /* si hay error */
        if (errorcode != grOk)
        {
                printf("Error al establecer el modo grafico: %s\n",
                                grapherrormsg(errorcode));
                printf("Pulsar una tecla y aborta:");
                getch();
                exit(1);
        }
        ColorMax = getmaxcolor();
        XMax     = getmaxx();
        YMax     = getmaxy();
}

void DibujarPantalla(void)
{
        float ValorY[4];// Para los rótulos del eje Y
        char i;

        // Dibuja un rect ngulo con v,rtices esqIA (esquina Izquierda Arriba) y esqDD (esquina Derecha
        //Debajo)
        XesqIA = XMax/10;
        YesqIA = YMax/8;
        XesqDD = XMax-5;
        YesqDD = YMax-5;
        setcolor(ColorMarco);
        setlinestyle(0,1,THICK_WIDTH);
        rectangle(XesqIA, YesqIA, XesqDD, YesqDD);
        setfillstyle(SOLID_FILL, ColorFondo);
        floodfill(XesqIA+10, YesqIA+10, ColorMarco);

        /* Dibuja las líneas de referencia, en las posiciones Ygnd (punto central de la representación, que se
        corresponde con el 0), YSatMas (límite por arriba, correpondiente a la saturación positiva de la
        señal), YSatMenos (ídem por abajo), e Y1+, Y2+, Y3+, Y1-, Y2-, Y3- (seis puntos intermedios) */
        Ygnd = (YesqIA + YesqDD)/2;
        espacio = 50;
        Y1 = Ygnd - espacio;
        Y2 = Ygnd - 2*espacio;
        Y3 = Ygnd - 3*espacio;
        YSatMas = Ygnd - 4*espacio;
        Y4 = Ygnd + espacio;
        Y5 = Ygnd + 2*espacio;
        Y6 = Ygnd + 3*espacio;
        YSatMenos = Ygnd + 4*espacio;
        // Límites de la pantalla gráfica para dibujar luego la señal
```

```
        XMinSignal = XesqIA+3;          // Hay que sumar o restar 3 porque la línea del marco es gruesa
        XMaxSignal = XesqDD-3;
        YMinSignal = YesqIA+3;
        YMaxSignal = YesqDD-3;

        YTime = YesqIA-15;              // Altura a la que se sacarán las marcas de tiempo:

        // Dibuja las líneas de referencia
        setlinestyle(0,1,NORM_WIDTH);
        //setcolor(ColorLinea0);
        setcolor(ColorLineasRef);
        line(XesqIA+2, Ygnd, XesqDD-2, Ygnd);
        setcolor(ColorLineasRef);
        line(XesqIA+2, Y1, XesqDD-2, Y1);
        line(XesqIA+2, Y2, XesqDD-2, Y2);
        line(XesqIA+2, Y3, XesqDD-2, Y3);
        line(XesqIA+2, Y4, XesqDD-2, Y4);
        line(XesqIA+2, Y5, XesqDD-2, Y5);
        line(XesqIA+2, Y6, XesqDD-2, Y6);
        line(XesqIA+2, YSatMas, XesqDD-2, YSatMas);
        line(XesqIA+2, YSatMenos, XesqDD-2, YSatMenos);

        PoneValoresEjeY();

        // Escribe los textos de presentación y de información de parámetros
        setcolor(ColorTitulo);
        outtextxy(XesqIA, 5, "Programa ChkArr4: visualización de datos de los arrays SEISAD18");
        setcolor(ColorTexto);
        outtextxy(XesqIA, 20, "Pulsar 'A' para ver la pantalla de ayuda, 'ESC' para salir");
        sprintf(Texto,"Fichero: %s  Muestreo: %ums  Gan.: %u", NombreFichDatos, Frec, Gain);
        outtextxy(XesqIA, 30, Texto);
        // Desde XTextoVar hasta el final de la pantalla es el espacio reservado para el texto variable
        //(número de canal, escala en los ejes x e y)
        XTextoVar = XesqDD-160;
        PoneTexto();
        setcolor(ColorTexto);
}

// Borra la señal de la pantalla, para lo cual machaca el fondo actual y vuelve a pintar las líneas de referencia y
//el marco
void ReponerPantalla(void)
{
        // Tapa los mensajes de tiempo de la pantalla anterior
        setfillstyle(SOLID_FILL, BLACK);
        bar(0,YesqIA-20, XMax, YesqIA);
        // Nuevo fondo de pantalla
        setfillstyle(SOLID_FILL, ColorFondo);
        bar(XesqIA, YesqIA, XesqDD, YesqDD);
        // Nuevo marco
        setcolor(ColorMarco);
        setlinestyle(0,1,THICK_WIDTH);
        rectangle(XesqIA, YesqIA, XesqDD, YesqDD);
        // Dibuja las líneas de referencia
        setlinestyle(0,1,NORM_WIDTH);
        setcolor(ColorLinea0);
        line(XesqIA+2, Ygnd, XesqDD-2, Ygnd);
```

```
        setcolor(ColorLineasRef);
        line(XesqIA+2, Y1, XesqDD-2, Y1);
        line(XesqIA+2, Y2, XesqDD-2, Y2);
        line(XesqIA+2, Y3, XesqDD-2, Y3);
        line(XesqIA+2, Y4, XesqDD-2, Y4);
        line(XesqIA+2, Y5, XesqDD-2, Y5);
        line(XesqIA+2, Y6, XesqDD-2, Y6);
        line(XesqIA+2, YSatMas, XesqDD-2, YSatMas);
        line(XesqIA+2, YSatMenos, XesqDD-2, YSatMenos);
}

void PantallaAyuda(void)
{
        // Tapa los mensajes de tiempo de la pantalla anterior
        setfillstyle(SOLID_FILL, BLACK);
        bar(0,YesqIA-20, XMax, YesqIA);
        // Nuevo fondo de pantalla
        setfillstyle(SOLID_FILL, ColorFondo);
        bar(XesqIA, YesqIA, XesqDD, YesqDD);
        // Nuevo marco
        setcolor(ColorMarco);
        setlinestyle(0,1,THICK_WIDTH);
        rectangle(XesqIA, YesqIA, XesqDD, YesqDD);
        // Tapa los valores anteriores del eje Y
        setfillstyle(SOLID_FILL, BLACK);
        bar(0,Ygnd-3,XesqIA-3,Ygnd+3);
        bar(0,YSatMas-3,XesqIA-3,YSatMas+3);
        bar(0,YSatMenos-3,XesqIA-3,YSatMenos+3);
        // Saca los mensajes de ayuda:
        setcolor(ColorTitulo);
        outtextxy(XesqIA+10, YesqIA + 10, "ChkArr4: programa para la visualización de datos de los
        arrays de");
        outtextxy(XesqIA+10, YesqIA + 20, "tarjetas SEISAD18");
        setcolor(ColorTexto);
        outtextxy(XesqIA+10, YesqIA + 40, "Teclas de operación:");
        outtextxy(XesqIA+10, YesqIA + 60, "'+': muestra la siguiente pantalla de datos correspondientes al
        canal");
        outtextxy(XesqIA+50, YesqIA + 70, "seleccionado");
        outtextxy(XesqIA+10, YesqIA + 80, "'-': muestra la pantalla de datos anterior");
        outtextxy(XesqIA+10, YesqIA + 90, "'i': muestra la pantalla de datos inicial");
        outtextxy(XesqIA+10, YesqIA + 100, "'c': cambia el canal que se visualiza");
        outtextxy(XesqIA+10, YesqIA + 110, "'x': cambia la escala del eje X");
        outtextxy(XesqIA+10, YesqIA + 120, "'y': cambia la escala del eje Y");
        outtextxy(XesqIA+10, YesqIA + 130, "'a': muestra esta pantalla de ayuda");
        outtextxy(XesqIA+10, YesqIA + 140, "'ESC ': sale del programa");
        outtextxy(XesqIA+10, YesqIA + 160, "En la esquina superior derecha se muestran, en rojo, los
        siguientes");
        outtextxy(XesqIA+10, YesqIA + 170, "par metros:");
        outtextxy(XesqIA+10, YesqIA + 190, "Ch: canal actual (valores entre 1 y 12)");
        outtextxy(XesqIA+10, YesqIA + 200, "Sx: escala actual del eje X (valores entre 1 y 4)");
        outtextxy(XesqIA+10, YesqIA + 210, "Sy: escala actual del eje Y (valores entre 1 y 13). La escala
        1");
        outtextxy(XesqIA+42, YesqIA + 220, "muestra todo el fondo de escala de los conversores (+/-
        2.5V).");
        outtextxy(XesqIA+42, YesqIA + 230, "Cada vez que se incrementa este parámetro se aumenta al
        doble la");
```

```
        outtextxy(XesqIA+42, YesqIA + 240, "escala. La representación siempre se centra en 0V, por lo
        que");
        outtextxy(XesqIA+42, YesqIA + 250, "para valores altos de Sy puede dejar de verse la señal si
        ésta");
        outtextxy(XesqIA+42, YesqIA + 260, "presenta offset.");
        outtextxy(XesqIA+10, YesqIA + 280, "Pulsar cualquier tecla para volver a la pantalla de
        visualización de");
        outtextxy(XesqIA+10, YesqIA + 290, "datos");
        getch();
        // Repone los valores en el eje Y
        PoneValoresEjeY();
        ValoresPantallaActual();
}

void PintaSignal(void)
{
        if (Flag.PantallaNueva)
        {
                Flag.PantallaNueva = FALSE;
                ReponerPantalla();
                XActual = XMinSignal;
                moveto(XActual, YActual);
                putpixel(XActual, YActual, ColorSignal);
        }
        else
        {
                if ((NumTotalMuestras%FactorEscalaX) == 0)
                {
                        if (XActual++ == XMaxSignal) Flag.PantallaNueva = TRUE;
                        setcolor(ColorSignal);// Por si acaso
                        lineto(XActual, YActual);
                }
        }
}

void PintaDataError(void)
{
        setcolor(LIGHTRED);
        // Sólo se saca el texto 'Data error' si la escala de las X no está comprimida. En caso contrario,
        //simplemente se pinta la línea vertical de color rojo:
        if (FactorEscalaX == 1)     outtextxy(XActual-35,YTime,"Data error");
        moveto(XActual, YActual);
        //setcolor(ColorLineasRef);
        line(XActual, YesqIA+2, XActual, YesqDD-2);
}

void InicVars(void)
{
        NumCanal = 0;// Inicialmente se muestra el canal 0
        EscalaX = 0;
        FactorEscalaX = pow(2, EscalaX);
        EscalaY = 0;
        FactorEscalaY = pow(2, EscalaY);
}
```

```
void Iniciacion(void)
{
	// Cálculo del factor de escala
	FondoEscala = pow(2, Resolucion);
	// El fondo de escala de los ADCs corresponde a las dos líneas de saturación de la pantalla
	Pixel = FondoEscala/((long)(YSatMenos-YSatMas)*FactorEscalaY);

	// Iniciación de la posición del puntero gráfico
	XActual = XMinSignal;
	YActual = Ygnd;
	moveto(XActual, YActual);

	Flag.DatoNegativo = FALSE;

	if (Resolucion == 24) CeroADCs = pow(2,23);
	else CeroADCs = pow(2,15);

	// Inicializa los valores relevantes de la primera pantalla, por si se vuelve a ella
	NumPantalla = 0;
	LeeDato();
	ConvierteAPixels();
	YInicial[NumPantalla] = YActual;
	PosFichero[NumPantalla] = 0;
	offset = 0;
	Muestra[NumPantalla] = 0;
	FlagDatoOK[NumPantalla] = TRUE;
	FlagTiempoOK[NumPantalla] = TRUE;
	rewind(FichDatos);
}

void LeeDato(void)
{
	unsigned char i, j, NumTarjeta;

	adc[0] = adc[1] = adc[2] = 0;
	// En función del canal seleccionado, lee los datos de una de las cuatro tarjetas
	switch(NumCanal)
	{
		case 0:
		case 1:
		case 2:
			NumTarjeta = 0;
			break;
		case 3:
		case 4:
		case 5:
			NumTarjeta = 1;
			break;
		case 6:
		case 7:
		case 8:
			NumTarjeta = 2;
			break;
		case 9:
		case 10:
		case 11:
```

```
                        NumTarjeta = 3;
        }
        // Se salta los tres bytes de cabecera y nueve de datos de las tarjetas que no interesan
        for (i=0; i<NumTarjeta+1; i++)
        {
                fseek(FichDatos, 3, SEEK_CUR);   // Se salta los tres bytes de cabecera
                for(j=0; j<9; j++) ChkStr[j] = getc(FichDatos);
        }
        // Se leen los tres canales de la tarjeta seleccionada
        adc[0] = (((long)ChkStr[0]<<16) | ((long)ChkStr[1]<<8) | (long)ChkStr[2]);
        adc[1] = (((long)ChkStr[3]<<16) | ((long)ChkStr[4]<<8) | (long)ChkStr[5]);
        adc[2] = (((long)ChkStr[6]<<16) | ((long)ChkStr[7]<<8) | (long)ChkStr[8]);
        fseek(FichDatos, (3-NumTarjeta)*12, SEEK_CUR);  // Se posiciona al principio de la siguiente
                                                        //muestra
        offset +=48;
}

void TeclaPulsada(void)
{
        char Tecla;

        do
        {
                Flag.TeclaBuena = TRUE;// En principio, la tecla es una de las correctas
                Tecla = tolower(getch());
                switch(Tecla)
                {
                        case '+':
                                NumPantalla++;
                                // Guarda toda la información relevante, por si hay que recuperar la
                                //pantalla
                                YInicial[NumPantalla] = YActual;
                                PosFichero[NumPantalla] = offset;
                                Muestra[NumPantalla] = NumMuestra;
                                if (Flag.DatoOK) FlagDatoOK[NumPantalla] = TRUE;
                                else FlagDatoOK[NumPantalla] = FALSE;
                                if (Flag.TiempoOK) FlagTiempoOK[NumPantalla] = TRUE;
                                else FlagTiempoOK[NumPantalla] = FALSE;
                                break;
                        case '-':  // Tecla '-': salta una pantalla hacia atrás
                                if (NumPantalla > 0) NumPantalla--;
                                YActual = YInicial[NumPantalla];
                                offset = PosFichero[NumPantalla];
                                NumMuestra = Muestra[NumPantalla];
                                if (FlagDatoOK[NumPantalla]) Flag.DatoOK = TRUE;
                                else Flag.DatoOK = FALSE;
                                if (FlagTiempoOK[NumPantalla]) Flag.TiempoOK = TRUE;
                                else Flag.TiempoOK = FALSE;
                                fseek(FichDatos, PosFichero[NumPantalla], SEEK_SET);
                                break;
                        case 'i':
                                ValoresPantallaInicial();
                                break;
                        case 'c':
                                if (++NumCanal == 12) NumCanal = 0;
                                ValoresPantallaInicial();
```

```
                                PoneTexto();
                                break;
                        case 'x':
                                if (++EscalaX == 4) EscalaX = 0;
                                FactorEscalaX = pow(2, EscalaX);
                                ValoresPantallaActual();
                                PoneTexto();
                                break;
                        case 'y':
                                if (++EscalaY == 13) EscalaY = 0;
                                FactorEscalaY = pow(2, EscalaY);
                                Pixel = FondoEscala/((long)(YSatMenos-YSatMas)*FactorEscalaY);
                                ValoresPantallaActual();
                                PoneTexto();
                                PoneValoresEjeY();
                                break;
                        case 'a':
                                PantallaAyuda();
                                break;
                        case 0x1B:      // Tecla 'ESC': sale del programa
                                Flag.Fin = TRUE;
                                break;
                        default:  // Resto de teclas: pita
                                sound(1000);
                                delay(200);
                                nosound();
                                Flag.TeclaBuena = FALSE;
                }
        }
        while (!Flag.TeclaBuena);
}

void LeerConfiguracion(void)
{
        FILE *FichConfig;
        int i, j;
        int Parametro[15];
        char Nombre[15][20];

        Flag.FichConfOK = TRUE;// En principio, el fichero es bueno
        if ((FichConfig = fopen("CHKARR4.CFG", "rt")) == NULL)
        {
                Flag.FichConfOK = FALSE;
                printf("\n El fichero CHKARR4.CFG no existe.");
                printf("\n Se continúa el programa con los parámetros por defecto.");
        }
        else
        {
                for (i=0; i < 10; i++)
                {
                        fscanf(FichConfig,"%s %i", &Nombre[i], &Parametro[i]);
                }
                fclose(FichConfig);

                if (Flag.FichConfOK)    // Sólo comprueba el parámetro siguiente si el anterior era
                {                       //correcto
```

```
                Resolucion = Parametro[0];
                if ((Resolucion != 16)&&(Resolucion != 24)) Flag.FichConfOK = FALSE;
        }

        if (Flag.FichConfOK)    // Los colores los comprobamos con las variables Parámetro[i],
        {                       //para hacerlo más rápido
                for(i=1; i<9; i++)
                {
                        for(j=0; j<16; j++)
                        {
                                if (Parametro[i] != j) Flag.FichConfOK = FALSE;
                                else
                                {
                                        Flag.FichConfOK = TRUE;
                                        break;
                                }
                        }
                        if (!Flag.FichConfOK) break;    // En cuanto encontremos que uno de
                }                                       //los colores está mal nos salimos.
        }

        if(Flag.FichConfOK)
        {
                ColorFondo      = Parametro[1];
                ColorLineasRef  = Parametro[2];
                ColorLinea0     = Parametro[3];
                ColorSignal     = Parametro[4];
                ColorMarco      = Parametro[5];
                ColorUnidadesY  = Parametro[6];
                Color0          = Parametro[7];
                ColorTitulo     = Parametro[8];
                ColorTexto      = Parametro[9];
        }
    }

    // Si ha encontrado algún fallo en el fichero o no ha podido abrirlo, toma los parámetros por defecto
    if (!Flag.FichConfOK)
    {
        Resolucion      = 24;
        ColorFondo      = BLUE;
        ColorLineasRef  = LIGHTBLUE;
        ColorLinea0     = LIGHTCYAN;
        ColorSignal     = YELLOW;
        ColorMarco      = WHITE;
        ColorUnidadesY  = GREEN;
        Color0          = LIGHTGREEN;
        ColorTitulo     = WHITE;
        ColorTexto      = LIGHTGRAY;
    }
}
```

```
void ConvierteAPixels(void)
{
	unsigned long dato;

	// Para los canales 0, 3, 6 y 9 hay que tomar adc[0], para los canales 1, 4, 7 y 10, adc[1], y para los
	//canales 2, 5, 8 y 11, adc[2].
	if (adc[NumCanal%3] >= CeroADCs)
	{
		dato = adc[NumCanal%3]-CeroADCs;
		Flag.DatoNegativo = FALSE;
	}
	else
	{
		dato = CeroADCs-adc[NumCanal%3];
		Flag.DatoNegativo = TRUE;
	}

	YActual = dato/Pixel;
	if (YActual > ((YMaxSignal-YMinSignal)/2))
	{
		if (Flag.DatoNegativo) YActual = YMaxSignal;
		else YActual = YMinSignal;
	}
	else
	{
		if (Flag.DatoNegativo) YActual = Ygnd+YActual;
		else YActual = Ygnd-YActual;
	}
}

void LeeTiempo(void)
{
	unsigned char HoraMSB, HoraLSB, MinutoMSB, MinutoLSB, SegundoMSB, SegundoLSB;

	// Guarda la posición actual del puntero al fichero para volver después
	fgetpos(FichDatos, &filepos);
	// Mueve el puntero a la posición de la información de tiempo GPS
	fseek(FichDatos, 4800, SEEK_CUR);
	ChkStr[0] = getc(FichDatos);
	if(((ChkStr[0]) == 'A') || ((ChkStr[0]) == 'V'))
	{
		Flag.TiempoOK  = TRUE;
		HoraMSB        = getc(FichDatos);
		HoraLSB        = getc(FichDatos);
		MinutoMSB      = getc(FichDatos);
		MinutoLSB      = getc(FichDatos);
		SegundoMSB     = getc(FichDatos);
		SegundoLSB     = getc(FichDatos);
		sprintf(CadenaParam,"%c%c:%c%c:%c%c", HoraMSB, HoraLSB, MinutoMSB,
		MinutoLSB, SegundoMSB, SegundoLSB);
		if ((ChkStr[0]) == 'A') setcolor(ColorTexto);
		else setcolor(LIGHTRED);		// Si la posición del GPS no es buena (carácter 'V'),
						//la hora se pone en rojo.
		// Si la escala de las X está comprimida, sólo se saca el tiempo cada cinco segundos:
		if (FactorEscalaX == 1) outtextxy(XActual-30,YTime,CadenaParam);
		else
```

```
                {
                        if ((FactorEscalaX == 2) || (FactorEscalaX == 4))
                        {
                                if (((SegundoLSB-'0')%5) == 0) outtextxy(XActual-30,
                                YTime,CadenaParam);
                        }
                        // Para factor de escala 8 sólo se saca el tiempo cada diez segundos:
                        else if (SegundoLSB == '0') outtextxy(XActual-30,YTime,CadenaParam);
                }
        }
        else
        {
                Flag.TiempoOK = FALSE;
                if (FactorEscalaX == 1)
                {
                        setcolor(LIGHTRED);
                        outtextxy(XActual-25,YTime,"No time");
                }
        }
        if ((FactorEscalaX != 1) && (((SegundoLSB-'0')%5) == 0)) setcolor(BLACK);
        else setcolor(ColorLineasRef);
        line(XActual, YesqIA+2, XActual, YesqDD-2);
        fsetpos(FichDatos, &filepos);// Se vuelve a la posición guardada del fichero
        moveto(XActual, YActual);// Se vuelve a la posición correcta de la pantalla
}

unsigned char CabeceraSegundo(void)
{
        ChkStr[0] = getc(FichDatos);
        ChkStr[1] = getc(FichDatos);
        ChkStr[2] = getc(FichDatos);
        fseek(FichDatos, -3, SEEK_CUR); // Deja el puntero donde estaba
        if((ChkStr[0] == 0x55) && (ChkStr[1] == 0x55) && (ChkStr[2] == 0x55)) return(TRUE);
        else return(FALSE);
}

unsigned char ErrorTramaDatos(void)
{
        unsigned char i;

        for (i=0; i<10; i++)
        {
                ChkStr[i] = getc(FichDatos);
                if (ChkStr[i] != '1')
                {
                        fseek(FichDatos, -(i+1), SEEK_CUR);     // Si no es error de trama de datos,
                        return(FALSE);                          //deja el puntero donde estaba antes
                }                                               //salir.
        }
        offset += 10;
        return(TRUE);   // Si es error de trama de datos, el puntero se queda al final
}
```

```
unsigned char ErrorBufferDatos(void)
{
	unsigned char i;

	for (i=0; i<10; i++)
	{
		ChkStr[i] = getc(FichDatos);
		if (ChkStr[i] != '2')
		{
			fseek(FichDatos, -(i+1), SEEK_CUR);	// Si no es error de buffer de datos,
			return(FALSE);				//deja el puntero donde estaba antes
		 }						//de salir.
	}
	offset += 10;
	return(TRUE);// Si es error de buffer de datos, el puntero se queda al final
}

void LeerCabecera(void)
{
	fseek(FichDatos, 98, SEEK_SET);
	Frec = getc(FichDatos);
	fseek(FichDatos, 192, SEEK_SET);
	Gain = getc(FichDatos);
	rewind(FichDatos);		// Deja el puntero en el inicio del fichero
}

void PoneTexto(void)
{
	// Tapa los mensajes anteriores
	setfillstyle(SOLID_FILL, BLACK);
	bar(XTextoVar,30,XMax,40);
	sprintf(Texto,"Ch: %u Sx: %u Sy: %u", NumCanal+1, EscalaX+1, EscalaY+1);
	setcolor(LIGHTRED);
	outtextxy(XTextoVar, 30, Texto);
}

void PoneValoresEjeY(void)
{
	// Tapa los valores anteriores
	setfillstyle(SOLID_FILL, BLACK);
	bar(0,Ygnd-3,XesqIA-3,Ygnd+3);
	bar(0,YSatMas-3,XesqIA-3,YSatMas+3);
	bar(0,YSatMenos-3,XesqIA-3,YSatMenos+3);

	setcolor(ColorUnidadesY);

	switch (EscalaY)
	{
		case 0:
			outtextxy(0, Ygnd-3,"     0V");// Siempre restamos 3 para centrarlo en la línea
			outtextxy(0, YSatMas-3," + 2.5V");
			outtextxy(0, YSatMenos-3," - 2.5V");
			break;
		case 1:
			outtextxy(0, Ygnd-3,"     0V");
			outtextxy(0, YSatMas-3,"+ 1.25V");
```

```
            outtextxy(0, YSatMenos-3,"- 1.25V");
            break;
case 2:
            outtextxy(0, Ygnd-3,"     0V");
            outtextxy(0, YSatMas-3,"+ 0.63V");
            outtextxy(0, YSatMenos-3,"- 0.63V");
            break;
case 3:
            outtextxy(0, Ygnd-3,"     0V");
            outtextxy(0, YSatMas-3,"+ 0.31V");
            outtextxy(0, YSatMenos-3,"- 0.31V");
            break;
case 4:
            outtextxy(0, Ygnd-3,"     0V");
            outtextxy(0, YSatMas-3,"+ 0.16V");
            outtextxy(0, YSatMenos-3,"- 0.16V");
            break;
case 5:
            outtextxy(0, Ygnd-3,"     0V");
            outtextxy(0, YSatMas-3," + 80mV");
            outtextxy(0, YSatMenos-3," - 80mV");
            break;
case 6:
            outtextxy(0, Ygnd-3,"     0V");
            outtextxy(0, YSatMas-3," + 40mV");
            outtextxy(0, YSatMenos-3," -40mV");
            break;
case 7:
            outtextxy(0, Ygnd-3,"     0V");
            outtextxy(0, YSatMas-3," + 20mV");
            outtextxy(0, YSatMenos-3," - 20mV");
            break;
case 8:
            outtextxy(0, Ygnd-3,"     0V");
            outtextxy(0, YSatMas-3," + 10mV");
            outtextxy(0, YSatMenos-3,"- 10mV");
            break;
case 9:
            outtextxy(0, Ygnd-3,"     0V");
            outtextxy(0, YSatMas-3,"  + 5mV");
            outtextxy(0, YSatMenos-3,"  - 5mV");
            break;
case 10:
            outtextxy(0, Ygnd-3,"     0V");
            outtextxy(0, YSatMas-3,"+ 2.5mV");
            outtextxy(0, YSatMenos-3,"- 2.5mV");
            break;
case 11:
            outtextxy(0, Ygnd-3,"     0V");
            outtextxy(0, YSatMas-3,"+1.25mV");
            outtextxy(0, YSatMenos-3,"-1.25mV");
            break;
case 12:
            outtextxy(0, Ygnd-3,"     0V");
            outtextxy(0, YSatMas-3,"+0.63mV");
            outtextxy(0, YSatMenos-3,"-0.63mV");
```

```
        }
}

void ValoresPantallaInicial(void)
{
        NumPantalla = 0;
        YActual = YInicial[NumPantalla];
        offset = PosFichero[NumPantalla];
        NumMuestra = Muestra[NumPantalla];
        if (FlagDatoOK[NumPantalla]) Flag.DatoOK = TRUE;
        else Flag.DatoOK = FALSE;
        if (FlagTiempoOK[NumPantalla]) Flag.TiempoOK = TRUE;
        else Flag.TiempoOK = FALSE;
        rewind(FichDatos);
}

void ValoresPantallaActual(void)
{
        YActual = YInicial[NumPantalla];
        offset = PosFichero[NumPantalla];
        NumMuestra = Muestra[NumPantalla];
        if (FlagDatoOK[NumPantalla]) Flag.DatoOK = TRUE;
        else Flag.DatoOK = FALSE;
        if (FlagTiempoOK[NumPantalla]) Flag.TiempoOK = TRUE;
        else Flag.TiempoOK = FALSE;
        fseek(FichDatos, PosFichero[NumPantalla], SEEK_SET);
}
```

```
/*****************************************************************************
  Programa principal
*****************************************************************************/
void main(int argc, char *argv[])
{
	clrscr();

	// Comprueba si la sintaxis es correcta y si existe el fichero de datos
	if (argc != 2)
	{
		printf("\n La sintaxis correcta es:\n\n ChkArr4 NombreFicheroDatos\n");
		exit(0);
	}
	strcpy(NombreFichDatos, argv[1]);

	if ((FichDatos = fopen( NombreFichDatos, "rb")) == NULL)
	{
		printf("\n No se puede abrir el fichero de datos %s", NombreFichDatos);
		exit(0);
	}

	LeerConfiguracion();
	LeerCabecera();
	ModoGrafico();
	InicVars();
	DibujarPantalla();
	Iniciacion();
	while(!Flag.Fin)
	{
		if(CabeceraSegundo())
		{
			Flag.DatoOK = TRUE;
			LeeTiempo();
		}
		else
		{
			if((ErrorTramaDatos()) || (ErrorBufferDatos()))
			{
				Flag.DatoOK = FALSE;
				PintaDataError();
			}
		}
		// Lee cien datos (o escribe cien ceros, si ha habido error de datos)
		NumMuestra = 0;
		do
		{
			if (Flag.PantallaNueva) TeclaPulsada();
			if (Flag.DatoOK) LeeDato();
			else adc[0] = adc[1] = adc[2] = 0x800000;// Mitad del fondo de escala
			ConvierteAPixels();
			PintaSignal();
			NumMuestra++;
			NumTotalMuestras++;
		}
		while(NumMuestra < Frec);
```

```
                // Después de la última muestra del segundo hay que saltarse el tiempo GPS (o la trama de
                //error, en su caso)
                if (Flag.DatoOK)
                {
                        if (Flag.TiempoOK) LongInfoTiempo = 13;
                        else LongInfoTiempo = 10;
                        fseek(FichDatos, LongInfoTiempo, SEEK_CUR);
                        offset += LongInfoTiempo;
                }
                // Comprueba si ha llegado al final del fichero
                ChkStr[0] = getc(FichDatos);
                if (!feof(FichDatos)) fseek(FichDatos, -1, SEEK_CUR);// Deja el puntero donde estaba
                else
                {
                        Flag.Fin = TRUE;
                        getch();
                }
        }
        closegraph();
        fclose(FichDatos);
}
```

ANEXO III.A.2.- PROGRAMA DE LOS PIC16F84 DE LAS TARJETAS A/D DE LAS ANTENAS PORTÁTILES DEL IAG

III.A.2.a. DEFINICIONES

Fichero: SEISAD18.H

```
;
; Fichero de cabecera para el programa de los PIC16F84 de las tarjetas de conversión
;A/D de las nuevas antenas portátiles del IAG. Incluye las equivalencias de los nombres
;asignados a los pines de entrada/salida para el control de los conversores A/D y
;transmisión de datos, más algunas otras definiciones necesarias.
;
        NOLIST
;*****************************************************************************
; Asignación de pines
;*****************************************************************************
; ASIGNACIÓN DE PINES PARA CONTROL DEL AD7710 Y PUERTO SERIE:
; Puerto A:
CONTT           EQU     0
DX              EQU     1
CONTR           EQU     2
GAIN2           EQU     2
GAIN1           EQU     3
GAIN0           EQU     4
; Puerto B :
DR              EQU     0
DRDY            EQU     1
RFS             EQU     2
TFS             EQU     3
A0              EQU     3
DAT1            EQU     4
DAT2            EQU     5
DAT3            EQU     6
CLK             EQU     7
;
WL              EQU     7       ; Longitud datos 16 o 24 bits
;
; ASIGNACIÓN DE PINES PARA LAS LÍNEAS DE DATOS Y OTRAS:
; Puerto A:
OutData         EQU     0       ; Bit 0: salida serie de datos
OutRecTrans     EQU     1       ; Bit 1: salida habilitación transmisión/recepción RS-485
OutSync         EQU     2       ; Bit 2: salida pulso de sincronización conversores AD7710
InPulsInic      EQU     3       ; Bit 3: entrada de pulsos de inicialización
MSBitID         EQU     2       ; Bit 2: entrada MSbit de identificación de tarjeta
LSBitID         EQU     4       ; Bit 4: entrada LSbit de identificación de tarjeta

        LIST
```

III.A.2.b. PROGRAMA DE LAS TARJETAS DE CONVERSIÓN A/D

Fichero: SEISAD_3.ASM

```
;
; Programa del PIC16F84 de las tarjetas de conversión A/D de los módulos de
;adquisición. Los registros están definidos en el fichero de cabecera C84_REG.H (anexo
;II.A.3.a).

        LIST P=16F84

        ;#define         PruebaTrans     ; Definir para pruebas de transmisión (cadenas constantes)
        #define         FlancoPPSDesc   ; Definir para flanco de PPS descendente, comentar para
                                        ;flanco ascendente
        INCLUDE <C84_reg.h>             ; Registros del PIC16F84
        INCLUDE <SEISAD18.h>            ; Pines I/O y otras definiciones para la adquisición

;*****************************************************************************
; Variables
;*****************************************************************************
PosVar          EQU     0x0C            ; Posición de inicio de las variables
Temp1           EQU     PosVar
Count1          EQU     PosVar+.1
ADC1            EQU     PosVar+.2
ADC2            EQU     PosVar+.3
ADC3            EQU     PosVar+.4
ADC11           EQU     PosVar+.5
ADC12           EQU     PosVar+.6
ADC13           EQU     PosVar+.7
ADC21           EQU     PosVar+.8
ADC22           EQU     PosVar+.9
ADC23           EQU     PosVar+.10
ADC31           EQU     PosVar+.11
ADC32           EQU     PosVar+.12
ADC33           EQU     PosVar+.13
ByteAEnviar     EQU     PosVar+.14
ContBitsTX      EQU     PosVar+.15
RegTransm       EQU     PosVar+.16
ContRetardo     EQU     PosVar+.17
HdrSegundo      EQU     PosVar+.18
NumTarjeta      EQU     PosVar+.19
NumMaxTarj      EQU     PosVar+.20
CheckSum1       EQU     PosVar+.21
CheckSum2       EQU     PosVar+.22
CheckSum3       EQU     PosVar+.23
ChkSumTotal1    EQU     PosVar+.24
ChkSumTotal2    EQU     PosVar+.25
ChkSumTotal3    EQU     PosVar+.26
Frecuencia      EQU     PosVar+.27
Ganancia        EQU     PosVar+.28
SincGPS         EQU     PosVar+.29
ContMuestras    EQU     PosVar+.30
ContPulsInic    EQU     PosVar+.31
MSBContTmr0     EQU     PosVar+.32
LSBContTmr0     EQU     PosVar+.33
CteRetSer       EQU     PosVar+.34
CteMSBTmr0      EQU     PosVar+.35
CteLSBTmr0      EQU     PosVar+.36
```

```
MSBBloque       EQU     PosVar+.37
LSBBloque       EQU     PosVar+.38

Flag            EQU     PosVar+.39
Muestra1        EQU     0               ; Bit 0 de Flag: primera muestra del segundo
Muestra2        EQU     1               ; Bit 1 de Flag: segunda muestra del segundo
Muestra3        EQU     2               ; Bit 2 de Flag: tercera muestra del segundo
Muestra4        EQU     3               ; Bit 3 de Flag: cuarta muestra del segundo
Muestra5        EQU     4               ; Bit 4 de Flag: quinta muestra del segundo
FinTmr0         EQU     5               ; Bit 5 de Flag: fin del intervalo programado con el timer 0
PPS             EQU     6               ; Bit 6 de Flag: PPS recibido
;
;*****************************************************************************
; Funciones y macros
;*****************************************************************************
        ORG     0x050
;-------------------------------------------------------------------------------------------------------------------------
; Inicializacion.- Configuración de los pines de I/O y programación de los parámetros de adquisición de
;los conversores A/D.
;-------------------------------------------------------------------------------------------------------------------------
Inicializacion
                ; 1. Configuración de los pines I/O y WD:
                BCF     STATUS,RP0              ; Banco 0
                CLRF    PORTA
                CLRF    PORTB
                BSF     STATUS, RP0             ; Banco 1
                MOVLW           0x1C            ; 0x1C = 00011100b: RA2, RA3 y RA4 entradas, el
                MOVWF           TRISA           ;resto salidas
                MOVLW           0x73            ; 0x73 = 01110011b: RB0, RB1, RB4, RB5 y RB6
                MOVWF           TRISB           ;entradas, el resto salidas
                MOVLW           0x0F            ; Asigna prescaler al WD con valor 1:128, activa
                MOVWF           OPTION_R        ;pull-ups del puerto B
                BCF     STATUS,RP0              ; Banco 0

                ; 2. Lectura de la secuencia de inicialización y cálculo de las constantes para la
                ;inicialización de los AD7710s:
                CALL    WaitInic
                CALL    CalculaCtes

                ; 3. Inicialización de variables:
                CLRF    CheckSum1
                CLRF    CheckSum2
                CLRF    CheckSum3
                CLRF    ChkSumTotal1
                CLRF    ChkSumTotal2
                CLRF    ChkSumTotal3
                CLRF    MSBBloque
                CLRF    LSBBloque

                MOVLW           .0              ; Byte de sincronismo con GPS a 0 (hay PPS del
                MOVWF           SincGPS         ;GPS)
                CLRF    Flag
                MOVF    Frecuencia, W           ; Inicializa el contador de muestras con el valor de la
                MOVWF           ContMuestras    ;frecuencia de muestreo, calculado antes en
                                                ;'CalculaCtes'
                BCF     PORTA, OutRecTrans      ; Línea serie en recepción, por si se usa RS-485
                BSF     PORTA, OutData          ; Línea serie a 1 (estándar RS-232)

                ; 4. Inicialización de la interrupción externa para la gestión de los PPS
                CLRF    INTCON                  ; Pone a 0 todos los flags de interrupción
```

```
        BSF     STATUS, RP0             ; Banco 1
        ; Por defecto, flanco ascendente para los PPS (Garmin 35). Si se desea flanco
        ;descendente, hay que habilitar la directiva #define correspondiente al principio
        ;del programa.
        BSF     OPTION_R, INTEDG        ; Selecciona flanco ascendente para la interrupción
                                        ;externa
        ifdef   FlancoPPSDesc
        BCF     OPTION_R, INTEDG        ; Selecciona flanco descendente para la interrupción
        Endif                           ;externa
        BCF     STATUS, RP0             ; Banco 0
        BSF     INTCON, INTE            ; Habilita interrupción externa
        BSF     INTCON, GIE             ; Habilita todas las interrupciones no enmascaradas


        ; 5. Programación de los parámetros de operación de los AD7710s en el registro de
        ;control
        MOVLW           .24             ; Mete 24 en Count1, para llevar la cuenta del
número de
        MOVWF           Count1          ;número de bits transmitidos
        BSF     STATUS, RP0             ; Banco 1
        MOVLW           0x03            ; Cambia los pines de datos a salidas para escribir la
        MOVWF           TRISB           ;palabra de control
        BCF     STATUS, RP0             ; Banco 0
        NOP
        BCF     PORTB, CLK              ; Baja CLK
        BCF     PORTB, TFS              ; Baja TFS
        BSF     PORTB, RFS              ; Sube RFS
        BCF     PORTB, A0               ; Baja A0 (acceso al registro de control)
        NOP
Prog1
        RLF     ADC3, F                 ; Desplaza las tres variables un lugar a la izquierda,
        RLF     ADC2, F                 ;en el bit C de STATUS se queda el último bit.
        RLF     ADC1, F
        BTFSS   STATUS, C               ; Pone en los tres pines de datos el valor del bit
        BCF     PORTB, DAT1             ;rotado
        BTFSS   STATUS, C
        BCF     PORTB, DAT2
        BTFSS   STATUS, C
        BCF     PORTB, DAT3
        BTFSC   STATUS, C
        BSF     PORTB, DAT1
        BTFSC   STATUS, C
        BSF     PORTB, DAT2
        BTFSC   STATUS, C
        BSF     PORTB, DAT3
        NOP
        BSF     PORTB, CLK              ; Realiza la escritura: sube CLK
        NOP
        BCF     PORTB, CLK
        NOP
        DECFSZ          Count1, F       ; Cuenta los bits que ha escrito. Si queda alguno,
        GOTO    Prog1                   ;vuelve a Prog1. Si ha acabado, vuelve a configurar
        BSF     PORTB, DAT1             ;los pines de datos como entradas.
        BSF     PORTB, DAT2
        BSF     PORTB, DAT3
        NOP
        BSF     STATUS, RP0             ; Banco 1
        MOVLW           0x73            ; Pines de datos como entradas
        MOVWF           TRISB
        BCF     STATUS, RP0             ; Banco 0
```

```
                NOP
                BSF     PORTB, TFS          ; Sube TFS
                BSF     PORTB, RFS          ; Sube RFS
                BSF     PORTB, A0           ; Sube A0 (acceso al registro de datos)
                CLRWDT                      ; Refresca el WD
                NOP
                NOP
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; adc.- Función para la lectura de datos (Ortiz, R., comunicación personal). Espera a que DRDY esté
;activada, lee el dato y vuelve a esperar a que se desactive DRDY.
;-------------------------------------------------------------------------------------------------------------------
adc
                BCF     PORTB, CLK          ; Baja CLK
                MOVLW       .24             ; Inicializa el contador a 24, para llevar la cuenta del
                MOVWF       Count1          ;número de bits transmitidos
adc0
                BTFSS   PORTB, DRDY         ; Polling a DRDY
                GOTO    adc0
                NOP
                BSF     PORTB, A0           ; Sube A0 (acceso a registro de datos)
                NOP
                BCF     PORTB, RFS          ; Baja RFS (lectura)
                NOP
adc1
                NOP
                BCF     STATUS, C           ; Mete el nuevo bit en los bytes de datos. Para ello
                RLF     ADC11, F            ;desplaza un lugar los tres bytes y pone el bit 0 a 1 si
                RLF     ADC12, F            ;es necesario. Esto lo repite 24 veces para cada uno
                RLF     ADC13, F            ;de los tres canales.
                BTFSC   PORTB, DAT1
                BSF     ADC11, 0

                BCF     STATUS, C
                RLF     ADC21, F
                RLF     ADC22, F
                RLF     ADC23, F
                BTFSC   PORTB, DAT2
                BSF     ADC21, 0

                BCF     STATUS, C
                RLF     ADC31, F
                RLF     ADC32, F
                RLF     ADC33, F
                BTFSC   PORTB, DAT3
                BSF     ADC31, 0

                BSF     PORTB, CLK          ; Sube y baja CLK con el retardo necesario, para
                NOP                         ;acceder al nuevo bit.
                NOP
                BCF     PORTB, CLK
                NOP
                NOP
                DECFSZ      Count1, F       ; Cuenta los bits leídos. Si falta alguno, vuelve a
                GOTO    adc1                ;adc1. Si ha terminado, sigue.
                NOP
                NOP
                BSF     PORTB, RFS          ; Sube RFS (fin de lectura)
```

```
adc2
                BTFSC  PORTB, DRDY          ; No sale hasta que DRDY no ha vuelto a bajar, para
                GOTO   adc2                 ;evitar que entre otra vez en la rutina de lectura.
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; DelaySer.- Genera el retardo necesario para la transmisión serie. El número de ciclos de instrucción
;necesario para cada una de las velocidades de transmisión (BR) se calcula:
; Número de ciclos = tBit/tciclo = (1/BR)/(4/10MHz) (para cristal de 10MHz). Así obtenemos:
;       BR=9600bps:     Número de ciclos = 260
;       BR=19200bps:    Número de ciclos = 130
;       BR=115200bps:   Número de ciclos = 22
; En la función 'DelaySer', Número de ciclos = 3*Nbucles+4
; Además, dentro de la función 'EnviarByte' hay 12 ciclos por bit, por lo que el valor que hay que darle a
CteRetSer es:
;       BR=9600bps:     260 = 3*NBucles+4+12 -> NBucles = 81
;       BR=19200bps:    130 = 3*NBucles+4+12 -> NBucles = 38
;       BR=115200bps:   22 = 3*NBucles+4+12 -> NBucles = 2
;-------------------------------------------------------------------------------------------------------------------
DelaySer
                MOVF   CteRetSer, W
                MOVWF          ContRetardo
Sigue
                DECFSZ         ContRetardo, F
                GOTO   Sigue
                NOP                         ; Tercer ciclo del último bucle
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; EnviarByte.- Envía el byte 'ByteAEnviar' por el pin I/O usado como salida de datos del puerto serie. La
;transmisión se realiza con 1 bit de start, 8 de datos y 1 de stop, a la velocidad definida por la función de
;retardo 'DelaySer'. La función 'EnviarByte' está pensada para que se inviertan exactamente el mismo
;número de ciclos de instrucción (12) en cada uno de los bits (start, stop y datos), tanto si valen 0 como 1.
;-------------------------------------------------------------------------------------------------------------------
EnviarByte
                BCF    STATUS,RP0           ; Banco 0
                BCF    PORTA, OutData       ; Bit de START a 0
                MOVF   ByteAEnviar, W       ; Mueve el byte que se enviará al registro de
                MOVWF          RegTransm    ;transmisión
                MOVLW          .8           ; Inicializa el contador de bits transmitidos
                MOVWF          ContBitsTX
                NOP                         ; Ciclo que faltaba en el bit de START
OtroBit
                CALL   DelaySer             ; El primer Delay es del bit de START
                RRF    RegTransm, F         ; Envía primero el LSB
                BTFSC  STATUS, C
                GOTO   SacaUno
                NOP
                BCF    PORTA, OutData
                GOTO   Cont
SacaUno
                BSF    PORTA, OutData
                NOP
                NOP
Cont
                DECFSZ         ContBitsTX, F
                GOTO   OtroBit
                CALL   DelaySer             ; Delay del último bit de datos
                NOP                         ; Ciclos que faltan
                NOP
```

```
                NOP
                NOP
                NOP
                BSF     PORTA, OutData          ; Bit de STOP a 1
                CALL    DelaySer                ; Delay del bit de STOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; MandaHdr1.- Manda la cabecera de comienzo de segundo (tres caracteres iguales, que dependen del
;número de tarjeta).
;-------------------------------------------------------------------------------------------------------------------
MandaHdr1
                MOVF    HdrSegundo, W
                MOVWF           ByteAEnviar
                CALL    EnviarByte
                CALL    EnviarByte
                CALL    EnviarByte
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; MandaHdr2.- Manda la cabecera de la segunda muestra del segundo (número de tarjeta + 2 primeros
;bytes de checksum).
;-------------------------------------------------------------------------------------------------------------------
MandaHdr2
                MOVF    NumTarjeta, W
                MOVWF           ByteAEnviar
                CALL    EnviarByte
                MOVF    ChkSumTotal1, W
                MOVWF           ByteAEnviar
                CALL    EnviarByte
                MOVF    ChkSumTotal2, W
                MOVWF           ByteAEnviar
                CALL    EnviarByte
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; MandaHdr3.- Manda la cabecera de la tercera muestra del segundo (último byte de checksum + 2 bytes
;de frecuencia de muestreo).
;-------------------------------------------------------------------------------------------------------------------
MandaHdr3
                MOVF    ChkSumTotal3, W
                MOVWF           ByteAEnviar
                CALL    EnviarByte
                CLRF    ByteAEnviar             ; MSByte de la frecuencia de muestreo, en nuestro
                CALL    EnviarByte              ;caso siempre es 0 (ya que la máxima frec. son  200
                MOVF    Frecuencia, W           ;mps)
                MOVWF           ByteAEnviar
                CALL    EnviarByte
                RETURN
```

```
;
;-------------------------------------------------------------------------------------------------------------------
; MandaHdr4.- Manda la cabecera de la cuarta muestra del segundo (código de sincronismo con GPS + 2
;bytes de bloque de datos).
;-------------------------------------------------------------------------------------------------------------------
MandaHdr4
                MOVF   SincGPS, W
                MOVWF          ByteAEnviar
                CALL    EnviarByte
                MOVF   MSBBloque, W
                MOVWF          ByteAEnviar
                CALL    EnviarByte
                MOVF   LSBBloque, W
                MOVWF          ByteAEnviar
                CALL    EnviarByte
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; MandaHdr5.- Manda la cabecera de la quinta muestra del segundo (ganancia + 2 bytes a 0x00)
;-------------------------------------------------------------------------------------------------------------------
MandaHdr5
                MOVF   Ganancia, W
                MOVWF          ByteAEnviar
                CALL    EnviarByte
                CLRF    ByteAEnviar
                CALL    EnviarByte
                CALL    EnviarByte
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; Manda0s.- Manda la cabecera del resto de las muestras (0x00, 0x00, 0x00).
;-------------------------------------------------------------------------------------------------------------------
Manda0s
                CLRF    ByteAEnviar
                CALL    EnviarByte
                CALL    EnviarByte
                CALL    EnviarByte
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; MandaDato.- Manda los nueve bytes de datos. Los bytes se envían por canales (canal1, 2 y 3),
;empezando por el MSB. Es decir, el orden es: ADC13, ADC12, ADC11, ADC23, ADC22, ADC21,
;ADC33, ADC32, ADC31.
;-------------------------------------------------------------------------------------------------------------------
MandaDato
                MOVF   ADC13, W
                MOVWF          ByteAEnviar
                CALL    EnviarByte
                MOVF   ADC12, W
                MOVWF          ByteAEnviar
                CALL    EnviarByte
                MOVF   ADC11, W
                MOVWF          ByteAEnviar
                CALL    EnviarByte
                MOVF   ADC23, W
                MOVWF          ByteAEnviar
                CALL    EnviarByte
                MOVF   ADC22, W
                MOVWF          ByteAEnviar
                CALL    EnviarByte
```

```
                MOVF   ADC21, W
                MOVWF          ByteAEnviar
                CALL    EnviarByte
                MOVF   ADC33, W
                MOVWF          ByteAEnviar
                CALL    EnviarByte
                MOVF   ADC32, W
                MOVWF          ByteAEnviar
                CALL    EnviarByte
                MOVF   ADC31, W
                MOVWF          ByteAEnviar
                CALL    EnviarByte

                RETURN
;
;-------------------------------------------------------------------------------------------------------------------------
; LeeNumTarj.- Lee el código de tarjeta de los switches. El código se lee invertido, para que un switch a
;OFF corresponda a un 0 y uno a ON a un 1.
; Devuelve en W: el código de tarjeta, leído del par de pines MSBitID/LSBitID
;-------------------------------------------------------------------------------------------------------------------------
LeeNumTarj
                BTFSS  PORTA, MSBitID
                GOTO   Nums2o3
                BTFSC  PORTA, LSBitID
                RETLW 0
                RETLW 1
Nums2o3
                BTFSC  PORTA, LSBitID
                RETLW 2
                RETLW 3
;
;-------------------------------------------------------------------------------------------------------------------------
; CharHdr.- Asigna un carácter de cabecera en función del número de tarjeta. Los caracteres son 0x55
;para la tarjeta 0, 0x77 para la 1, 0x99 para la 2 y 0xBB para la 3.
; Devuelve en W: 0 si no ha habido error, 1 si ha habido error en la identificación del número de tarjeta.
;-------------------------------------------------------------------------------------------------------------------------
CharHdr
                CLRF    HdrSegundo
                MOVF   NumTarjeta, W
                MOVWF          Temp1
                INCF    Temp1, F
                DECFSZ         Temp1, F
                GOTO   NoEsTarj0
                MOVLW          0x55            ; Tarjeta 0: carácter de cabecera de segundo = 0x55
                GOTO   FinCharHdr
NoEsTarj0
                DECFSZ         Temp1, F
                GOTO   NoEsTarj1
                MOVLW          0x77            ; Tarjeta 1: carácter de cabecera de segundo = 0x77
                GOTO   FinCharHdr
NoEsTarj1
                DECFSZ         Temp1, F
                GOTO   NoEsTarj2
                MOVLW          0x99            ; Tarjeta 2: carácter de cabecera de segundo = 0x99
                GOTO   FinCharHdr
NoEsTarj2
                DECFSZ         Temp1, F
                RETLW.1                        ; Error, el número de tarjeta no es ningún valor
                                               ;permitido
                MOVLW          0xBB            ; Tarjeta 3: carácter de cabecera de segundo = 0xBB
```

```
FinCharHdr
                MOVWF       HdrSegundo
                RETLW.0
;
;-------------------------------------------------------------------------------------------------------------------
; Checksum.- Actualiza las variables de checksum.
; Devuelve en W: 1 si se han actualizado las variables totales (en la última muestra de cada segundo), 0 si
;sólo se han actualizado las variables parciales (cualquier muestra, salvo la  primera de cada segundo).
;-------------------------------------------------------------------------------------------------------------------
Checksum
                MOVF  ADC13, W          ; En CheckSum1 se suman los bytes más
                ADDWF       CheckSum1, F    ;significativos de las muestras de los tres canales
                MOVF  ADC23, W
                ADDWF       CheckSum1, F
                MOVF  ADC33, W
                ADDWF       CheckSum1, F
                MOVF  ADC12, W          ; En CheckSum2 se suman los bytes intermedios de
                ADDWF       CheckSum2, F    ;las muestras de los tres canales
                MOVF  ADC22, W
                ADDWF       CheckSum2, F
                MOVF  ADC32, W
                ADDWF       CheckSum2, F
                MOVF  ADC11, W          ; En CheckSum3 se suman los bytes menos
                ADDWF       CheckSum3, F    ;significativos de las muestras de los tres canales
                MOVF  ADC21, W
                ADDWF       CheckSum3, F
                MOVF  ADC31, W
                ADDWF       CheckSum3, F

                BTFSS  Flag, Muestra1
                RETLW.0
                MOVF  CheckSum1, W      ; Si es la última muestra del segundo (el flag de
                MOVWF       ChkSumTotal1    ;primera muestra ya se ha activado en 'MandaDato')
                MOVF  CheckSum2, W      ;pone a cero las variables de checksum, pasando
                MOVWF       ChkSumTotal2    ;antes su valor a las variables que se enviarán.
                MOVF  CheckSum3, W
                MOVWF       ChkSumTotal3
                CLRF  CheckSum1
                CLRF  CheckSum2
                CLRF  CheckSum3
                RETLW.1
;
;-------------------------------------------------------------------------------------------------------------------
; WaitInic.- Lee la secuencia de pulsos de inicialización.
;-------------------------------------------------------------------------------------------------------------------
                ; 1. Antes de iniciar la lectura de la cadena espera a que transcurra un intervalo (definido
                ;por la función 'ProgTmr0Inic') sin que se reciban pulsos. Esto se hace para estar
                ;seguros de que el inicio de la lectura no ha coincidido con la mitad de una cadena, ya
                ;que el driver de SEISLOG manda una cadena cada segundo mientras no recibe datos.
WaitInic
                CLRF  ContPulsInic      ; Programa el Tmr0 y espera a que se active el
                BCF   Flag, FinTmr0     ;flag de fin de timer 0 (en la ISR). Si, cuando
                CALL  ProgTmr0Inic      ;esto ocurre, el contador de pulsos de
HayCadena?                              ;inicialización continúa a 0, pasa a la lectura de
                BTFSC PORTA, InPulsInic ;la cadena de inicialización del segundo siguiente.
                GOTO  NoHayPulso        ; Si, por el contrario, se ha recibido algún pulso,
                INCF  ContPulsInic, F   ;se vuelve al principio de la función para repetir
WaitFinPuls                             ;el proceso.
                BTFSS PORTA, InPulsInic
                GOTO  WaitFinPuls
```

```
NoHayPulso
                BTFSS  Flag, FinTmr0
                GOTO   HayCadena?
                BCF    Flag, FinTmr0          ; Si el contador de pulsos de inicialización es
                MOVF   ContPulsInic, F        ;cero pasa a la lectura de la cadena, poniendo
                BTFSS  STATUS, Z              ;antes a FALSE el Flag de Fin de Tmr0
                GOTO   WaitInic

                ; 2. Lee la cadena de pulsos:
WaitPuls1
                CLRWDT
                BTFSC  PORTA, InPulsInic      ; Espera el primer pulso
                GOTO   WaitPuls1
                INCF   ContPulsInic, F
                CALL   ProgTmr0Inic
WaitFinPuls1                                  ; Espera el final del primer pulso
                BTFSS  PORTA, InPulsInic
                GOTO   WaitFinPuls1

WaitFinTmr0
                CLRWDT
                BTFSC  Flag, FinTmr0          ; Espera el final de la secuencia de inicialización,
                RETURN                        ;comprobando el flag correspondiente (se activa en
                BTFSC  PORTA, InPulsInic      ;la ISR, cuando el timer 0 se ha desbordado un
                GOTO   WaitFinTmr0            ;número prefijado de veces)
                INCF   ContPulsInic, F
WaitFinPulso
                BTFSS  PORTA, InPulsInic      ; Aquí no se refresca el WD, de modo que si no se
                GOTO   WaitFinPulso           ;detecta el fin de algún pulso hay reset
                GOTO   WaitFinTmr0
;
;-------------------------------------------------------------------------------------------------------------------------
; ProgTmr0Inic.- En la espera de la secuencia de inicialización el timer 0 debe programarase, desde que se
;ha recibido el primer pulso de inicialización, para que espere un intervalo de tiempo mayor que la
;duración máxima de la secuencia de inicialización y menor de un segundo (ya que el driver de SEISLOG
;envía la secuencia cada segundo mientras no recibe datos). El número máximo de pulsos de la secuencia
;es 34 (ver formato en documento SEISAD18.DOC), que a la velocidad de transmisión mínima (9600bps)
;suponen:
;       34pulsos x 1/9600bps x 10bits/pulso = 35.4ms = 35400us
; El timer 0 es de 8 bits. Tiene posibilidad de usar un postscaler, pero no lo utilizamos porque está
asignado al watchdog. Por tanto, se desbordará cada 256 ciclos de instrucción, que equivalen a:
;       256 x 0.4us/ciclo (con cristal de 10MHz) = 102.4us, y hará falta un número de vueltas igual a:
;       Nº vueltas Tmr0 = 35400us / 102.4us/vuelta = 345.7
; Es necesario un contador de dos bytes, inicializando el MSB a 2 (como mínimo).
;-------------------------------------------------------------------------------------------------------------------------
ProgTmr0Inic
                MOVLW       .4                ; Inicializa las ctes. con las que se programará el
                MOVWF       CteMSBTmr0        ;número de vueltas del Tmr0 a 1024 (MSB = .4,
                CLRF   CteLSBTmr0             ;LSB = 0).
                CALL   ProgTmr0
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------------
; ProgTmr0.- Programa los bytes más y menos significativos del contador de vueltas del Tmr0 con las
;constantes CteMSBTmr0 y CteLSBTmr0, respectivamente.
;-------------------------------------------------------------------------------------------------------------------------
ProgTmr0
                MOVF   CteMSBTmr0, W
                MOVWF       MSBContTmr0
                MOVF   CteLSBTmr0, W
```

```
                MOVWF   LSBContTmr0
                BCF     INTCON, GIE             ; Deshabilita todas las interrupciones
                BSF     STATUS, RP0             ; Banco 1
                BCF     OPTION_R, T0CS          ; Timer 0 en modo timer
                BCF     STATUS, RP0             ; Banco 0
                BCF     INTCON, T0IF            ; Habilita interrupción de timer 0, limpiando antes el
                BSF     INTCON, T0IE            ;flag por si se había activado y poniendo el registro
                CLRF    TMR0                    ;del timer a 0 para que empiece a contar desde ahora.
                BSF     INTCON, GIE             ; Habilita todas las interrupciones no enmascaradas

                RETURN
;
;-------------------------------------------------------------------------------------------------------------------------
; CalculaCtes.- Calcula las constantes para la inicialización de los conversores AD7710 y para la
;transmisión de datos a partir del número de pulsos recibidos en la secuencia de inicialización, que debe
;encontrarse en la variable ContPulsInic.
; Devuelve en W: 0 si no hay error, 1 si el valor de la constante ContPulsInic no era ninguno de los
;permitidos (ver fichero SEISAD18.DOC).
;-------------------------------------------------------------------------------------------------------------------------
CalculaCtes
                MOVF    ContPulsInic, W
                MOVWF   Temp1
                MOVLW   .10
                SUBWF   Temp1, F
                BTFSC   STATUS, Z
                RETLW   .1
                BTFSS   STATUS, C
                RETLW   .1                      ; Error: el número de pulsos leídos es menor de 10

                DECFSZ  Temp1, F
                GOTO    Puls12
                CALL    Ganancia1               ; 11 pulsos: ganancia 1, 25mps, 9600bps
                CALL    Frec25
                CALL    BR9600
                CALL    CR_25_9600              ; Asigna las ctes. de retardo para que las distintas
                                                ;tarjetas no tengan conflictos en la transmisión.
                MOVLW   .3                      ; Con estos parámetros el número máximo de tarjetas
                MOVWF   NumMaxTarj              ;que pueden transmitir datos es 3 (ver
                RETLW   .0                      ;SEISAD18.DOC).
Puls12
                DECFSZ  Temp1, F
                GOTO    Puls13
                CALL    Ganancia4               ; 12 pulsos: ganancia 4, 25mps, 9600bps
                CALL    Frec25
                CALL    BR9600
                CALL    CR_25_9600
                MOVLW   .3
                MOVWF   NumMaxTarj
                RETLW   .0
Puls13
                DECFSZ  Temp1, F
                GOTO    Puls14
                CALL    Ganancia16              ; 13 pulsos: ganancia 16, 25mps, 9600bps
                CALL    Frec25
                CALL    BR9600
                CALL    CR_25_9600
                MOVLW   .3
                MOVWF   NumMaxTarj
                RETLW   .0
Puls14
```

```
                DECFSZ      Temp1, F
                GOTO  Puls15
                CALL  Ganancia1         ; 14 pulsos: ganancia 1, 25mps, 19200bps
                CALL  Frec25
                CALL  BR19200
                CALL  CR_25_19200
                MOVLW       .4
                MOVWF       NumMaxTarj
                RETLW.0
Puls15
                DECFSZ      Temp1, F
                GOTO  Puls16
                CALL  Ganancia4         ; 15 pulsos: ganancia 4, 25mps, 19200bps
                CALL  Frec25
                CALL  BR19200
                CALL  CR_25_19200
                MOVLW       .4
                MOVWF       NumMaxTarj
                RETLW.0
Puls16
                DECFSZ      Temp1, F
                GOTO  Puls17
                CALL  Ganancia16        ; 16 pulsos: ganancia 16, 25mps, 19200bps
                CALL  Frec25
                CALL  BR19200
                CALL  CR_25_19200
                MOVLW       .4
                MOVWF       NumMaxTarj
                RETLW.0
Puls17
                DECFSZ      Temp1, F
                GOTO  Puls18
                CALL  Ganancia1         ; 17 pulsos: ganancia 1, 50mps, 9600bps
                CALL  Frec50
                CALL  BR9600
                ; Para 50_9600 sólo se permite una tarjeta, por lo que no es necesario retardo. Lo que sí
                ;hay que hacer es asegurarse de que el número de tarjeta es el 0 y, en caso contrario,
                ;salir.
                MOVLW       .1
                MOVWF       NumMaxTarj
                RETLW.0
Puls18
                DECFSZ      Temp1, F
                GOTO  Puls19
                CALL  Ganancia4         ; 18 pulsos: ganancia 4, 50mps, 9600bps
                CALL  Frec50
                CALL  BR9600
                MOVLW       .1
                MOVWF       NumMaxTarj
                RETLW.0
Puls19
                DECFSZ      Temp1, F
                GOTO  Puls20
                CALL  Ganancia16        ; 19 pulsos: ganancia 16, 50mps, 9600bps
                CALL  Frec50
                CALL  BR9600
                MOVLW       .1
                MOVWF       NumMaxTarj
                RETLW.0
```

```
Puls20
        DECFSZ      Temp1, F
        GOTO  Puls21
        CALL  Ganancia1          ; 20 pulsos: ganancia 1, 50mps, 19200bps
        CALL  Frec50
        CALL  BR19200
        CALL  CR_50_19200
        MOVLW       .3
        MOVWF       NumMaxTarj
        RETLW.0
Puls21
        DECFSZ      Temp1, F
        GOTO  Puls22
        CALL  Ganancia4          ; 21 pulsos: ganancia 4, 50mps, 19200bps
        CALL  Frec50
        CALL  BR19200
        CALL  CR_50_19200
        MOVLW       .3
        MOVWF       NumMaxTarj
        RETLW.0
Puls22
        DECFSZ      Temp1, F
        GOTO  Puls23
        CALL  Ganancia16         ; 22 pulsos: ganancia 16, 50mps, 19200bps
        CALL  Frec50
        CALL  BR19200
        CALL  CR_50_19200
        MOVLW       .3
        MOVWF       NumMaxTarj
        RETLW.0
Puls23
        DECFSZ      Temp1, F
        GOTO  Puls24
        CALL  Ganancia1          ; 23 pulsos: ganancia 1, 50mps, 115200bps
        CALL  Frec50
        CALL  BR115200
        CALL  CR_50_115200
        MOVLW       .4
        MOVWF       NumMaxTarj
        RETLW.0
Puls24
        DECFSZ      Temp1, F
        GOTO  Puls25
        CALL  Ganancia4          ; 24 pulsos: ganancia 4, 50mps, 115200bps
        CALL  Frec50
        CALL  BR115200
        CALL  CR_50_115200
        MOVLW       .4
        MOVWF       NumMaxTarj
        RETLW.0
Puls25
        DECFSZ      Temp1, F
        GOTO  Puls26
        CALL  Ganancia16         ; 25 pulsos: ganancia 16, 50mps, 115200bps
        CALL  Frec50
        CALL  BR115200
        CALL  CR_50_115200
        MOVLW       .4
        MOVWF       NumMaxTarj
        RETLW.0
```

```
Puls26
        DECFSZ      Temp1, F
        GOTO  Puls27
        CALL  Ganancia1         ; 26 pulsos: ganancia 1, 100mps, 19200bps
        CALL  Frec100
        CALL  BR19200
        ; Para 100_19200 sólo se permite una tarjeta, por lo que no es necesario retardo. Lo que
        ;sí hay que hacer es asegurarse de que el número de tarjeta es el 0 y, en caso contrario,
        ;salir.
        MOVLW       .1
        MOVWF       NumMaxTarj
        RETLW.0
Puls27
        DECFSZ      Temp1, F
        GOTO  Puls28
        CALL  Ganancia4         ; 27 pulsos: ganancia 4, 100mps, 19200bps
        CALL  Frec100
        CALL  BR19200
        MOVLW       .1
        MOVWF       NumMaxTarj
        RETLW.0
Puls28
        DECFSZ      Temp1, F
        GOTO  Puls29
        CALL  Ganancia16        ; 28 pulsos: ganancia 16, 100mps, 19200bps
        CALL  Frec100
        CALL  BR19200
        MOVLW       .1
        MOVWF       NumMaxTarj
        RETLW.0
Puls29
        DECFSZ      Temp1, F
        GOTO  Puls30
        CALL  Ganancia1         ; 29 pulsos: ganancia 1, 100mps, 115200bps
        CALL  Frec100
        CALL  BR115200
        CALL  CR_100_115200
        MOVLW       .4
        MOVWF       NumMaxTarj
        RETLW.0
Puls30
        DECFSZ      Temp1, F
        GOTO  Puls31
        CALL  Ganancia4         ; 30 pulsos: ganancia 4, 100mps, 115200bps
        CALL  Frec100
        CALL  BR115200
        CALL  CR_100_115200
        MOVLW       .4
        MOVWF       NumMaxTarj
        RETLW.0
Puls31
        DECFSZ      Temp1, F
        GOTO  Puls32
        CALL  Ganancia16        ; 31 pulsos: ganancia 16, 100mps, 115200bps
        CALL  Frec100
        CALL  BR115200
        CALL  CR_100_115200
        MOVLW       .4
        MOVWF       NumMaxTarj
        RETLW.0
```

```
Puls32
                DECFSZ      Temp1, F
                GOTO    Puls33
                CALL    Ganancia1           ; 32 pulsos: ganancia 1, 200mps, 115200bps
                CALL    Frec200
                CALL    BR115200
                CALL    CR_200_115200
                MOVLW       .3
                MOVWF       NumMaxTarj
                RETLW.0
Puls33
                DECFSZ      Temp1, F
                GOTO    Puls34
                CALL    Ganancia4           ; 33 pulsos: ganancia 4, 200mps, 115200bps
                CALL    Frec200
                CALL    BR115200
                CALL    CR_200_115200
                MOVLW       .3
                MOVWF       NumMaxTarj
                RETLW.0
Puls34
                DECFSZ      Temp1, F
                RETLW.1                     ; Error: el número de pulsos leídos es mayor de 34
                CALL    Ganancia16          ; 34 pulsos: ganancia 16, 200mps, 115200bps
                CALL    Frec200
                CALL    BR115200
                CALL    CR_200_115200
                MOVLW       .3
                MOVWF       NumMaxTarj
                RETLW.0
;
;-------------------------------------------------------------------------------------------------------------------------
; GananciaXXX, FrecXXX y BRXXXXXX.- Funciones para asignar las constantes. Las constantes se
;guardan en las variables ADC1, ADC2 y ADC3 (constantes de los conversores AD7710) y CteRetSer
;(constante para la velocidad de transmisión). El valor de dichas variables se calcula de acuerdo con el
;siguiente formato:
; ADC1 = 001ggg00, siendo ggg el código para programar la ganancia (ver datasheet del AD7710).
; ADC2:ADC3 = 1000ffff:ffffffff, siendo ff...ff el código para programar la frecuencia de muestreo.
; CteRetSer: ver función 'DelaySer'
;-------------------------------------------------------------------------------------------------------------------------
Ganancia1
                MOVLW       0x20
                MOVWF       ADC1
                MOVLW       .1              ; 'Ganancia' se transmitirá en las cabeceras
                MOVWF       Ganancia
                RETURN
;
Ganancia4
                MOVLW       0x28
                MOVWF       ADC1
                MOVLW       .4
                MOVWF       Ganancia
                RETURN
;
Ganancia16
                MOVLW       0x30
                MOVWF       ADC1
                MOVLW       .16
                MOVWF       Ganancia
                RETURN
```

```
;
Frec25
                MOVLW       0x83
                MOVWF       ADC2
                MOVLW       0x00
                MOVWF       ADC3
                MOVLW       .25             ; 'Frecuencia' se transmitirá en las cabeceras
                MOVWF       Frecuencia
                RETURN
;
Frec50
                MOVLW       0x81
                MOVWF       ADC2
                MOVLW       0x80
                MOVWF       ADC3
                MOVLW       .50
                MOVWF       Frecuencia
                RETURN
;
Frec100
                MOVLW       0x80
                MOVWF       ADC2
                MOVLW       0xC0
                MOVWF       ADC3
                MOVLW       .100
                MOVWF       Frecuencia
                RETURN
;
Frec200
                MOVLW       0x80
                MOVWF       ADC2
                MOVLW       0x60
                MOVWF       ADC3
                MOVLW       .200
                MOVWF       Frecuencia
                RETURN
;
BR9600
                MOVLW       .81
                MOVWF       CteRetSer
                RETURN
;
BR19200
                MOVLW       .38
                MOVWF       CteRetSer
                RETURN
;
BR115200
                MOVLW       .2
                MOVWF       CteRetSer
                RETURN
```

```
;
;-------------------------------------------------------------------------------------------------------------------
; CRXXX_XXXXXX: Funciones para asignar las constantes con las que hay que programar los registros
;del timer 0 para generar el retardo adecuado en la transmisión de datos. Para el cálculo se toma el
;periodo de muestreo menos un ms (asignado a la lectura del dato, con un margen amplio) y se divide
;entre el número máximo de tarjetas que se pueden usar con el baud rate seleccionado. Así, para 50mps y
;19200bps:
;       Ret = (1/50-1ms)/3tarjetas = 6.333ms = 15833ciclos de instrucción = 256x127 (aprox.).
; Teniendo en cuenta que el timer 0 se desborda cada 256 ciclos de instrucción, hay que programar las
ctes. del retardo a (ver rutina de servicio de la interrupción (ISR)):
;       CteMSBTmr0 = 1 (valor mínimo)
;       CteLSBTmr0 = 127
; En el cálculo se ha tenido en cuenta que el número máximo de tarjetas a 50mps y 19200bps es 3 (ver
;documento SEISAD18.DOC).
; Tanto en este caso como en los demás CteMSBTmr0 toma el valor mínimo (0), con lo que no influye
;en la generación del retardo. Sin embargo, no se puede quitar de la ISR porque se usa en la función de
;espera de la secuencia de inicialización (WaitInic).
;-------------------------------------------------------------------------------------------------------------------
CR_25_9600
                MOVLW       .1
                MOVWF       CteMSBTmr0
                MOVLW       .127
                MOVWF       CteLSBTmr0
                RETURN
;
CR_25_19200
                MOVLW       .1
                MOVWF       CteMSBTmr0
                MOVLW       .95
                MOVWF       CteLSBTmr0
                RETURN
;
CR_50_19200
                MOVLW       .1
                MOVWF       CteMSBTmr0
                MOVLW       .61
                MOVWF       CteLSBTmr0
                RETURN
;
CR_50_115200
                MOVLW       .1
                MOVWF       CteMSBTmr0
                MOVLW       .46
                MOVWF       CteLSBTmr0
                RETURN
;
CR_100_115200
                MOVLW       .1
                MOVWF       CteMSBTmr0
                MOVLW       .22
                MOVWF       CteLSBTmr0
                RETURN
;
CR_200_115200
                MOVLW       .1
                MOVWF       CteMSBTmr0
                MOVLW       .13
                MOVWF       CteLSBTmr0
                RETURN
```

```
;
;-------------------------------------------------------------------------------------------------------------------
; SyncADCs.- Manda un pulso de sincronismo a los tres conversores AD7710 de la placa. Este pulso sólo
;será efectivo cuando el switch 1-8 de SW2 esté a ON (configuración de estación de tres componentes,
;ver SEISAD18.DOC). En el resto de las configuraciones, el encargado de enviar el pulso de
;sincronización será el PIC del circuito PLL.
;-------------------------------------------------------------------------------------------------------------------
SyncADCs
                BSF     STATUS, RP0             ; Banco 1
                BCF     TRISA, OutSync          ; Configura el pin 2 del puerto A como salida
                BCF     STATUS,RP0              ; Banco 0
                BCF     PORTA, OutSync          ; Pulso negativo de sincronismo de 8us de duración
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                BSF     PORTA, OutSync
                BSF     STATUS, RP0             ; Banco 1
                BSF     TRISA, OutSync          ; Deja el pin 2 del puerto A como entrada
                BCF     STATUS,RP0              ; Banco 0
                RETURN
;
;-------------------------------------------------------------------------------------------------------------------
; RetTarj.- Genera un retardo variable, en función del número de tarjeta y de la frecuencia de muestreo. El
;retardo se usa para asegurar que cada tarjeta manda los datos en momentos distintos.
;-------------------------------------------------------------------------------------------------------------------
RetTarj
                MOVF    NumTarjeta, W
                BTFSC   STATUS, Z               ; Si es la tarjeta 0, no hay retardo.
                RETURN
                MOVWF           Temp1           ; Número de tarjeta en Temp1
                BCF     Flag, FinTmr0
ProgRetTmr0
                CALL    ProgTmr0
BucleRetTarj
                BTFSS   Flag, FinTmr0
                GOTO    BucleRetTarj
                BCF     Flag, FinTmr0
                DECFSZ          Temp1, F        ; Decrementa el número de tarjeta para ver si hay
                GOTO    ProgRetTmr0             ;que esperar más.
                RETURN
```

```
;
;-------------------------------------------------------------------------------------------------------------------
; ErrNumTarj.- Genera una señal cuadrada de cuatro segundos de periodo (dos segundos en alto y dos en
;bajo), para poder monitorizar el error.
; Esta función no tiene RETURN porque es un bucle infinito.
;-------------------------------------------------------------------------------------------------------------------
ErrNumTarj
                ; Deja la línea serie en alta impedancia para no interferir en la comunicación de datos
                ;del resto de las tarjetas:
                BCF     PORTA, OutRecTrans
                MOVLW           .76             ; Ctes. de retardo para dos segundos (2s =
                MOVWF           CteMSBTmr0      ;= 5000000ciclos = 76x256x256)
                CLRF    CteLSBTmr0
BucleRetErr
                BCF     Flag, FinTmr0
                CALL    ProgTmr0
                BSF     PORTA, OutData
BucleRetErr1
                CLRWDT
                BTFSS   Flag, FinTmr0
                GOTO    BucleRetErr1
                BCF     Flag, FinTmr0
                CALL    ProgTmr0
                BCF     PORTA, OutData
BucleRetErr0
                CLRWDT
                BTFSS   Flag, FinTmr0
                GOTO    BucleRetErr0
                GOTO    BucleRetErr
```

```
;*******************************************************************************
; Programa principal e interrupciones
;*******************************************************************************
;-------------------------------------------------------------------------------------------------------------------------
; Rutina de servicio de la interrupción
;-------------------------------------------------------------------------------------------------------------------------
                ORG     ISR_V                   ; Vector de interrupción

                BCF     STATUS, RP0             ; Banco 0
                BTFSS   INTCON, T0IF            ; Si es la interrupción del Timer 0, sigue. Si no lo es,
                GOTO    ChkINT                  ;comprueba si es la interrupción externa.
                BCF     INTCON, T0IF            ; Limpia el flag de interrupción
                DECFSZ          LSBContTmr0, F; Decrementa el byte menos significativo del
                GOTO    ChkINT                  ;contador de vueltas del Tmr0 y, si no es cero, sale.
                DECFSZ          MSBContTmr0, F; Si es cero decrementa a su vez el byte más
                GOTO    ChkINT                  ;significativo del contador y si no es cero sale. En
                BSF     Flag, FinTmr0           ;caso de que sea cero activa el flag de fin de Tmr 0 y
                BCF     INTCON, T0IE            ;deshabilita la interrupción de timer 0.
ChkINT
                BTFSS   INTCON, INTF
                RETFIE
                BCF     INTCON, INTF            ; Limpia el flag de interrupción
                BSF     Flag, PPS               ; Activa el flag de PPS recibido
                RETFIE
;
;-------------------------------------------------------------------------------------------------------------------------
; Programa principal
;-------------------------------------------------------------------------------------------------------------------------
                ORG     RESET_V
;
                GOTO    Inicio
;
                ORG     0x350
; Inicializaciones
********************************************************************************
;
Inicio
                ; 1. Inicialización:
                CALL    Inicializacion

                ; 2. Lectura del código de tarjeta:
                CALL    LeeNumTarj              ; Devuelve el número de tarjeta en W y lo pasa a
                MOVWF           NumTarjeta      ;NumTarjeta
                MOVF    NumMaxTarj, W           ; Si el número de tarjeta menos el número máximo
                SUBWF   NumTarjeta, W           ;de tarjetas para la configuración recibida es mayor o
                BTFSC   STATUS, C               ;igual que 0, hay error. En caso contrario, continúa.
                CALL    ErrNumTarj

                ; 3. Asigna el carácter de la cabecera de segundo en función del número de tarjeta:
                CALL    CharHdr

                ; 4. Espera un flanco de segundo antes de sincronizar los conversores AD7710:
WaitPPS_1
                BTFSS   Flag, PPS
                GOTO    WaitPPS_1
                BCF     Flag, PPS
                CALL    SyncADCs

                ; 5. Espera inicialmente otro PPS para que la primera muestra que se envía coincida con
                ;la primera de un segundo:
```

```
WaitPPS_2
                BTFSS   Flag, PPS               ; No se deshabilita el Flag, PPS porque se hace en
                GOTO    WaitPPS_2               ;ChkPPS.
;
; Bucle principal
;*********************************************************************************
;
                ; 1. Comprueba si ha llegado un PPS y, en su caso, marca la próxima muestra como la
                ;primera del segundo e incrementa el número de bloque. La comprobación del PPS hay
                ;que hacerla una vez que se ha activado DRDY, para estar seguros de que se realiza a la
                ;vez en todas las tarjetas.
ChkPPS
                BTFSS   PORTB, DRDY             ; Polling a DRDY
                GOTO    ChkPPS
                BTFSS   Flag, PPS
                GOTO    Lectura                 ; Si no ha llegado un PPS, pasa directamente a la
                BCF     Flag, PPS               ;lectura del dato.
                BSF     Flag, Muestra1          ; Activa el flag de muestra 1, para que se mande la
                                                ;cabecera de inicio de segundo (ver formato en
                                                ;SEISAD18.DOC)
                INCFSZ  LSBBloque, F            ; Incrementa el número de bloque, que permite
                GOTO    Lectura                 ;comprobar en recepción si se ha perdido algún dato
                INCF    MSBBloque, F            ;en la transmisión.

                ; 2. Actualiza las variables de checksum. Se hace aquí para saber ya si es la primera
                ;muestra del segundo, y poder así actualizar las variables definitivas que se enviarán:
Lectura
                CALL    Checksum

                ; 3. Lee el dato:
                CALL    adc
                BCF     ADC11, 0                ; Pone a 0 los LSbs de los LSBs de los datos, para
                BCF     ADC21, 0                ;evitar que puedan confundirse con alguna cabecera.
                BCF     ADC31, 0

                ifdef   PruebaTrans             ; Prueba de transmisión: asigna valores constantes a
                MOVLW           0x31            ;las variables de datos (caracteres ASCII '1' a '9').
                MOVWF           ADC11
                MOVLW           0x32
                MOVWF           ADC12
                MOVLW           0x33
                MOVWF           ADC13
                MOVLW           0x34
                MOVWF           ADC21
                MOVLW           0x35
                MOVWF           ADC22
                MOVLW           0x36
                MOVWF           ADC23
                MOVLW           0x37
                MOVWF           ADC31
                MOVLW           0x38
                MOVWF           ADC32
                MOVLW           0x39
                MOVWF           ADC33
                endif

                ; 4. Genera un retardo variable, en función del número de tarjeta:
                CALL    RetTarj
```

```
                ; 5.Mientras dura la transmisión deshabilita la int. de PPS, para que no interfiera:
                BCF     INTCON, INTE  ; Deshabilita interrupción externa

                ; 6. toma el control de la línea serie poniendo el MAX483 en transmisión:
                BSF     PORTA, OutRecTrans

                ; 7. Manda la cabecera de tres bytes (ver formato en SEISAD18.DOC):
                BTFSS   Flag, Muestra1
                GOTO    ChkMuestra2
                BCF     Flag, Muestra1
                BSF     Flag, Muestra2
                CALL    MandaHdr1
                GOTO    MandarDato
ChkMuestra2
                BTFSS   Flag, Muestra2
                GOTO    ChkMuestra3
                BCF     Flag, Muestra2
                BSF     Flag, Muestra3
                CALL    MandaHdr2
                GOTO    MandarDato
ChkMuestra3
                BTFSS   Flag, Muestra3
                GOTO    ChkMuestra4
                BCF     Flag, Muestra3
                BSF     Flag, Muestra4
                CALL    MandaHdr3
                GOTO    MandarDato
ChkMuestra4
                BTFSS   Flag, Muestra4
                GOTO    ChkMuestra5
                BCF     Flag, Muestra4
                BSF     Flag, Muestra5
                CALL    MandaHdr4
                GOTO    MandarDato
ChkMuestra5
                BTFSS   Flag, Muestra5
                GOTO    MandaCeros
                BCF     Flag, Muestra5
                CALL    MandaHdr5
                GOTO    MandarDato
MandaCeros
                CALL    Manda0s

                ; 8. Manda el dato:
MandarDato
                CALL    MandaDato

                ; 9. Deja la línea serie en alta impedancia poniendo el MAX483 en recepción:
                BCF     PORTA, OutRecTrans

                ; 10. Vuelve a habilitar la interrupción externa cuando termina de mandar datos:
                BSF     INTCON, INTE  ; Habilita interrupción externa

                ; 11. Refresca el watchdog:
                CLRWDT

                ; 12. Vuelve a la comprobación de PPS:
                GOTO    ChkPPS

                END
```

Anexo III.B:

Ficheros de circuito impreso de las antenas portátiles del IAG

ANEXO III.B: FICHEROS DE CIRCUITO IMPRESO DE LAS ANTENAS PORTÁTILES DEL IAG

a)

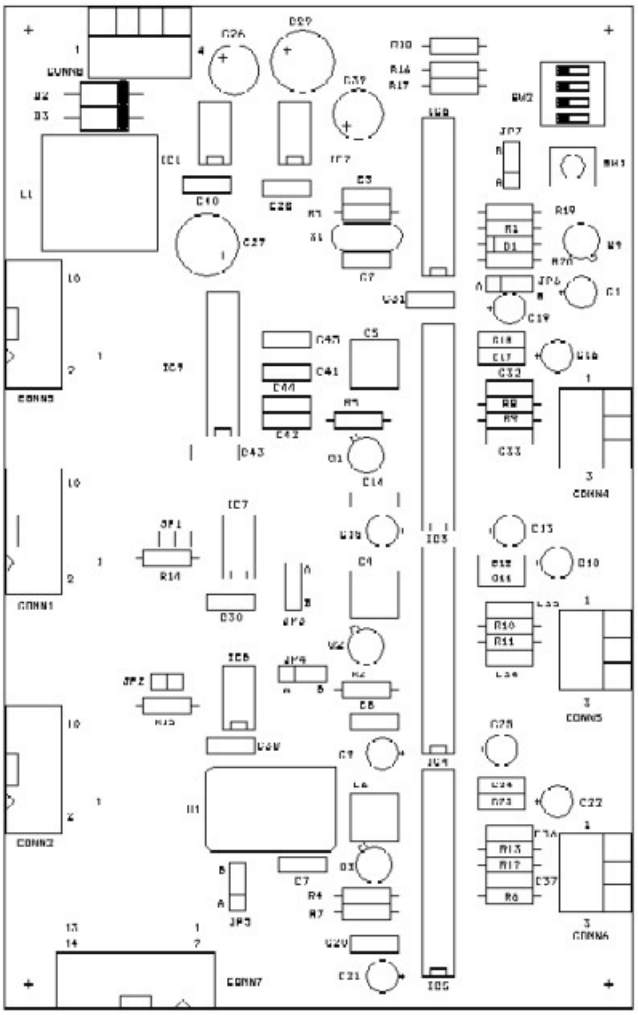
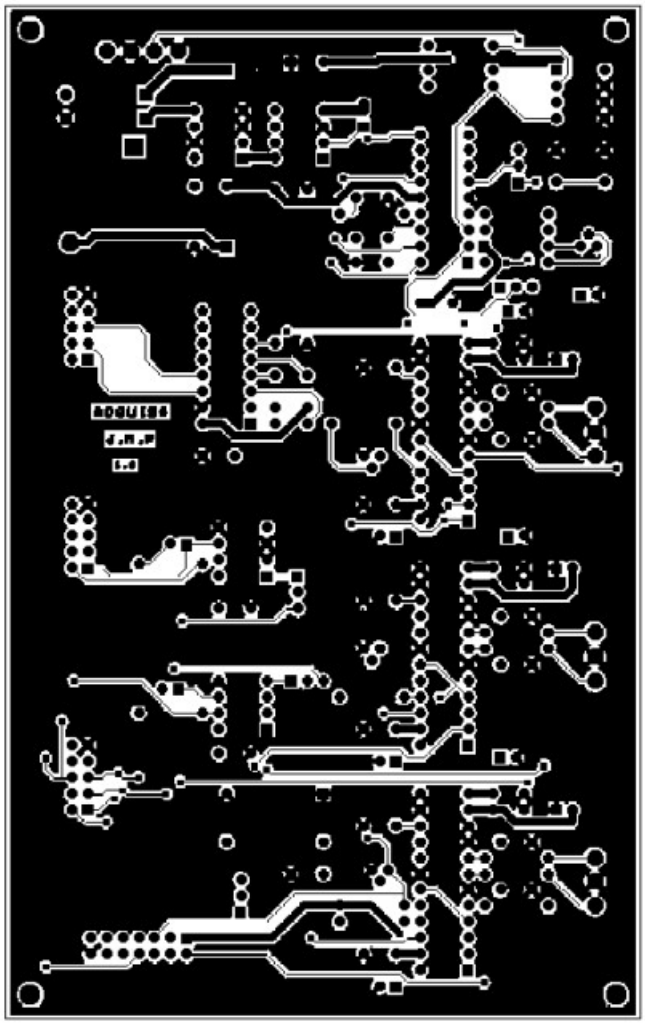
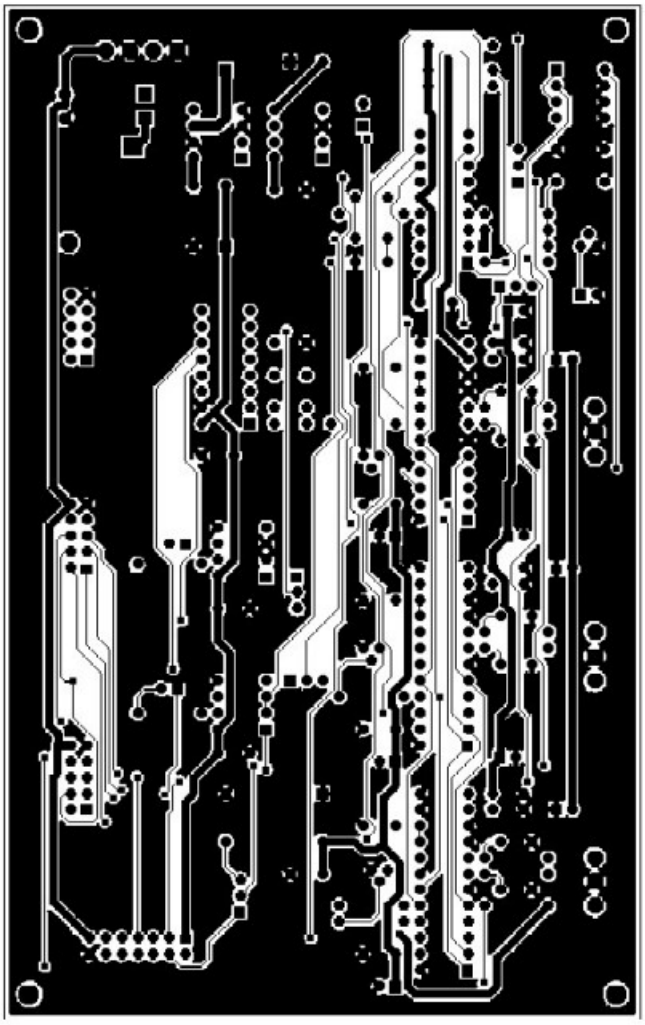


Figura III.1. *Placa de conversión A/D de las antenas portátiles del IAG. Aunque los esquemas eléctricos de esta tarjeta y de la de PLL son los mismos que en la antena del Vesubio (figuras 4.12 y 4.13, respectivamente), en el momento de fabricar las placas no se tenía acceso a los ficheros de fabricación. Por esta razón ambas tarjetas se rediseñaron a partir de los esquemas eléctricos y las placas fabricadas para la antena del Vesubio. Aquí se muestran la distribución de componentes (a), cara superior (b) y cara inferior (c) de la tarjeta de conversión A/D (diseño de la tarjeta: Javier Moreno Peláez).*

III.1.b)



III.1.c)



a)

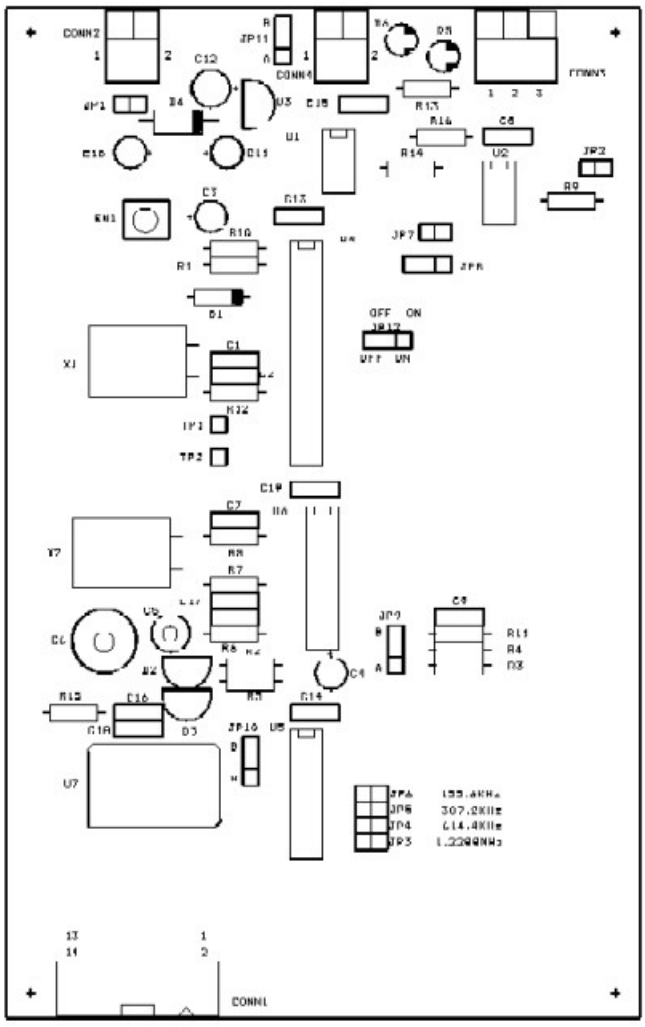
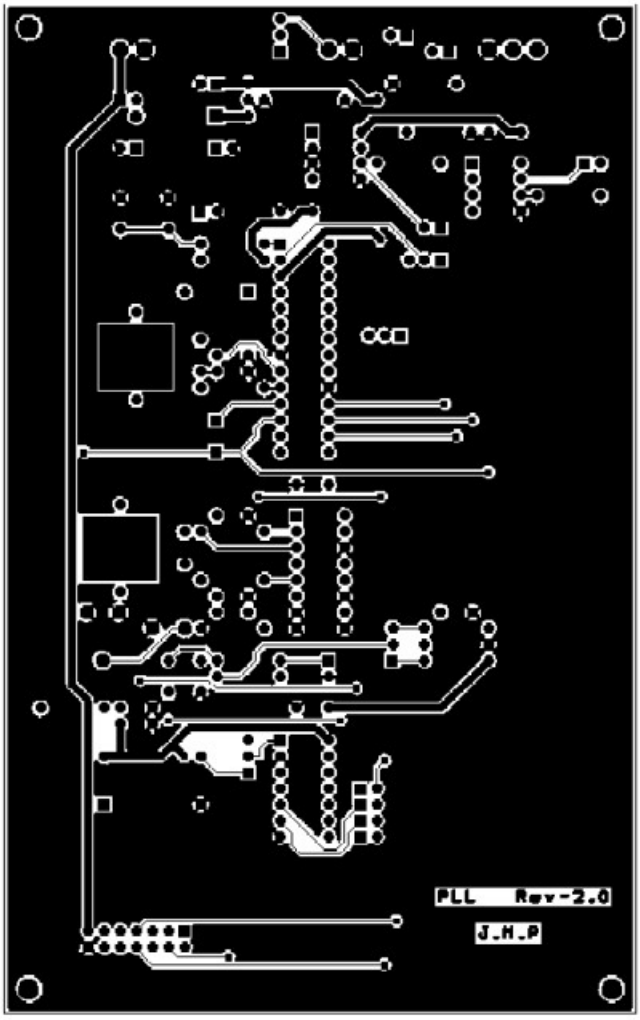
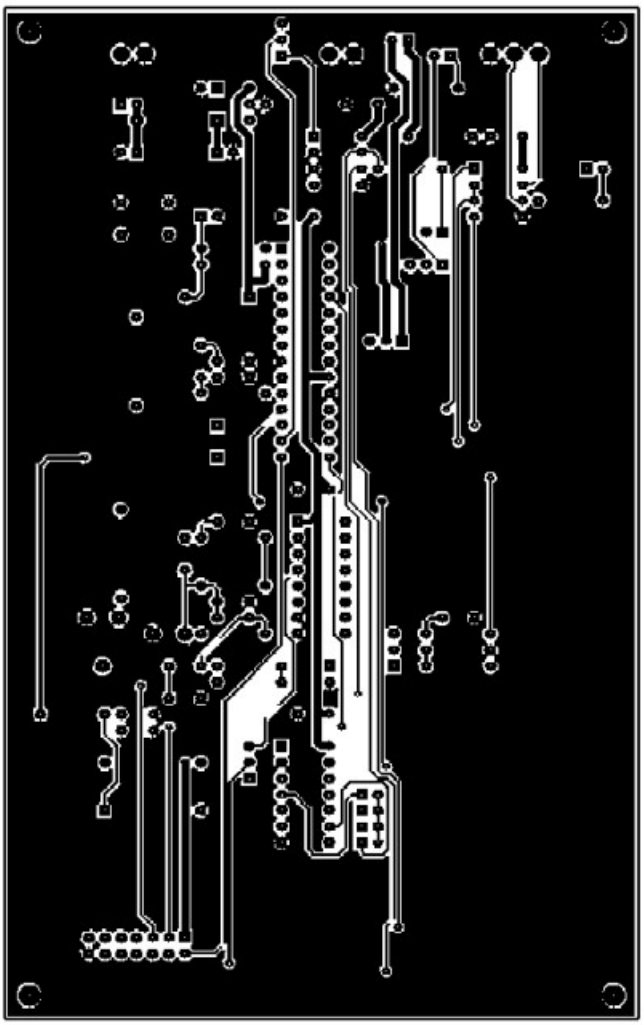


Figura III.2. *Placa del circuito PLL de las antenas portátiles del IAG (esquema en figura 4.13 de la memoria): distribución de componentes (a), cara superior (b) y cara inferior (c) (diseño de la tarjeta: Javier Moreno Peláez).*

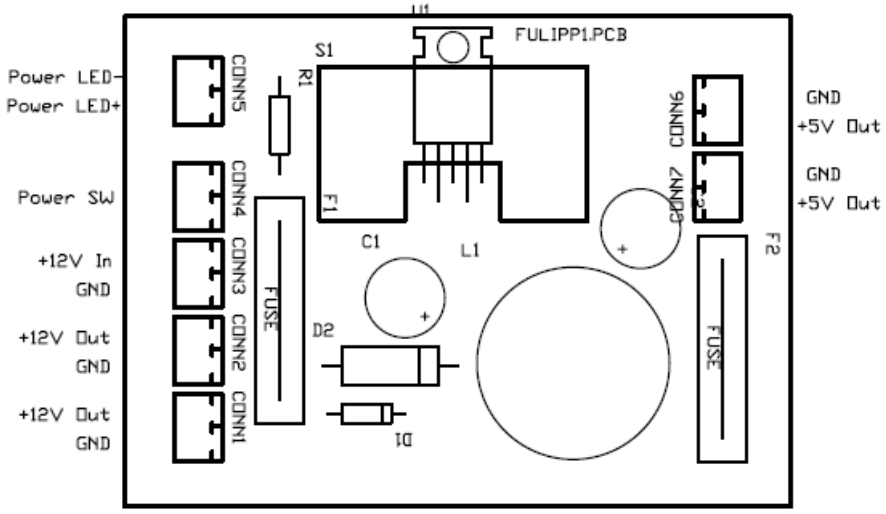
III.2.b)



III.2.c)



a)



b)

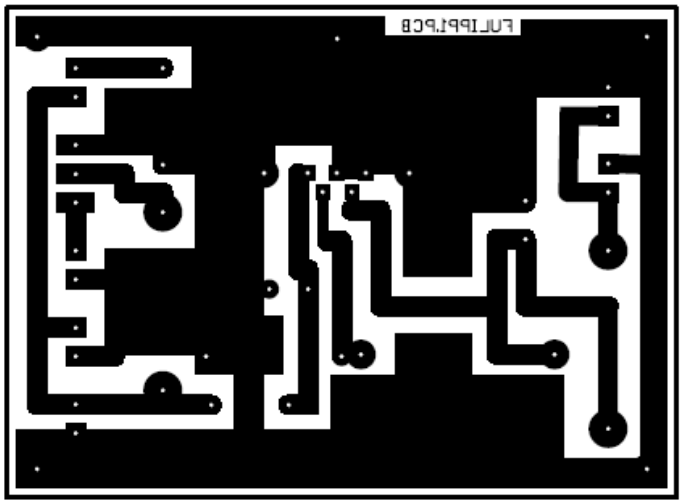


Figura III.3. *Placa de la fuente de alimentación de las antenas portátiles del IAG (esquema en figura 5.14): distribución de componentes (a) y cara inferior (b).*

Anexo III.C:

Otros anexos relacionados con las antenas portátiles del IAG

ANEXO III.C.1.- RELATIVO A LA OPERACIÓN DE LOS MÓDULOS DE DIECISÉIS BITS

III.C.1.a. PARÁMETROS DEL FICHERO DE CONFIGURACIÓN DE LOS MÓDULOS DE DIECISÉIS BITS (REDSIS.CFG)

El fichero de configuración REDSIS.CFG es modificable desde el programa REDSIS.EXE o mediante edición directa con cualquier editor de ficheros ASCII. Contiene parámetros relacionados con tres aspectos: operación del sistema, constantes del algoritmo de disparo y nomenclatura de los canales y ficheros de datos. La figura 5.8 de la memoria de tesis muestra la pantalla del programa de configuración CONFIRED.EXE, en el que puede verse un listado de estos parámetros.

Los parámetros de operación del sistema son tres:

- *Número de canales*: aunque el número máximo es ocho, es posible seleccionar un número menor para no consumir recursos del sistema en caso de tener menos sensores operativos.
- *Frecuencia de muestreo* (mps en la figura 5.8): la frecuencia de muestreo es configurable, con valores de entre 20 y 200 mps para un PC con frecuencia de reloj mayor de 16MHz[1].
- *Sincronismo reloj*: este parámetro indica el modo de sincronismo que se utiliza. Admite valores entre 0 y 2, que corresponden a sincronismo usando el reloj del PC (0), receptor GPS (1) o marcas externas de minuto (2). Cuando el programa de adquisición REDSIS arranca muestra en la parte superior de la pantalla el tipo de sincronismo utilizado.

Los parámetros relativos al algoritmo de disparo son:

- *Algoritmo*: admite los valores 0 y 1. El algoritmo de disparo es siempre de tipo STA/LTA (ver, por ejemplo, Havskov y Alguacil, 2004, pp. 128-132), pero el cálculo de los valores de STA y LTA puede realizarse por incremento (0) o por amplitud (1).
- *Preevento*: longitud en segundos de la memoria anterior al disparo. Admite valores entre 1 y 60.
- *Postevento*: tiempo en segundos que el equipo continúa registrando una vez que el algoritmo de detección da por terminado el evento. Igual que el preevento, admite valores entre 1 y 60 segundos. La duración total de un evento será la suma del preevento, el postevento y el tiempo durante el que el algoritmo de disparo está activo.
- *Estaciones para disparo*: para evitar que una estación ruidosa pueda provocar disparos falsos en toda la red se define este parámetro. El programa de adquisición no considera un disparo como válido hasta que el algoritmo no se ha activado en al menos el número de estaciones indicado por este parámetro, cuyo valor debe estar comprendido entre 1 y el número de

[1] La frecuencia de reloj de los PCs que se utilizaban originalmente para el control de estos sistemas podía imponer límites a la frecuencia de muestreo efectiva. Así, con un PC de menos de 10 MHz no era posible muestrear a 200 mps utilizando todos los canales. Para poder utilizar la configuración máxima (200 mps y ocho canales) era necesario usar un PC de al menos 16 MHz. Lógicamente este parámetro no impone ninguna limitación a la frecuencia de muestreo cuando se usan PCs de control más modernos.

canales operativos. Es necesario que haya un número de canales activos (ver parámetro 'Máscara canal' más adelante) igual o superior al número de estaciones para disparo, ya que de otro modo el programa nunca interpretaría un disparo como válido en toda la red y el sistema no se dispararía.

- *TSTA*: longitud en segundos del intervalo STA. Admite valores entre 1 y 20.
- *TLTA*: longitud en segundos del intervalo LTA. Admite valores entre 1 y 128.
- *Kdispa*: la condición de disparo en un algoritmo STA/LTA es que el cociente STA/LTA sea mayor que una constante de disparo K. En nuestro caso K se obtiene como $K = 2^{Kdispa}$. El parámetro *Kdispa* admite valores entre 1 y 4.
- *Kfiltro*: el algoritmo de adquisición dispone de un filtro digital paso baja con frecuencia de corte programable según la siguiente expresión:

$$f_c = \frac{f_{Nyquist}}{2^{Kfiltro}} = \frac{f_{muestreo}}{2^{Kfiltro+1}}$$

Este filtro digital no afecta a los datos registrados, sólo se utiliza a efectos de cálculo del algoritmo de disparo.

- *Segundos registro*: este parámetro limita la duración máxima de un evento. Su valor está dado en segundos. Tiene que ser mayor que la suma de los parámetros *Preevento* y *Postevento* y menor de 200 segundos.
- *Máscara canal*: como puede verse en la figura 5.8, existe un parámetro de máscara de canal por cada uno de los ocho canales del equipo. Este parámetro admite los valores 0 y 1, y sirve para activar (1) o inhibir (0) la posibilidad de que un canal pueda provocar un disparo del sistema.

En lo que respecta a los parámetros relativos a la nomenclatura de los canales y ficheros de datos, son los siguientes:

- *Directorio*: determina el directorio en el que los ficheros de evento serán archivados. Es importante comprobar que el directorio programado se ha creado previamente, ya que de otro modo los datos no se grabarán.
- *Nombre archivo*: el programa de adquisición REDSIS.EXE genera, cada vez que el algoritmo de disparo se activa, dos ficheros con el mismo nombre y extensiones distintas (DTS y SAD). El fichero SAD contiene, en formato binario, los datos correspondientes a todos los canales operativos en el momento de la adquisición. El DTS es un fichero de cabecera ASCII en el que se especifican los valores de los principales parámetros de adquisición y otra información relativa a los datos del correspondiente fichero SAD (ver figura III.4). El parámetro *Nombre archivo* del fichero de configuración permite seleccionar una de las dos posibles nomenclaturas de los ficheros SAD y DTS. El valor 0 hace que los ficheros se nombren con día y mes, mientras que el valor 1 hace que se utilice el día juliano, lo cual deja un carácter libre para identificador del equipo[2]. Las nomenclaturas completas para los ficheros son:

[2] La extensión máxima del nombre de los ficheros es la establecida por MS-DOS: ocho caracteres alfanuméricos para el nombre y tres para la extensión.

Con ‘Nombre Archivo’ = 0:

MMDDHHmm, con extensiones DTS o SAD, siendo:

MM: mes
DD: día
HH: hora
mm: minuto

Con ‘Nombre Archivo’ = 1:

JJJHHMMN, extensiones DTS o SAD, siendo:

JJJ: día juliano
HH: hora
MM: minuto
N: identificador del equipo

- *Nombre canal*: hay ocho parámetros de este tipo, correspondientes al nombre de cada uno de los posibles canales. Admite una cadena de hasta tres caracteres alfanuméricos, que se utilizará como identificador del canal correspondiente.
- *Nombre estación registro*: como se ha visto antes, cuando se trabaja con varios módulos de *array* y con el nombre de archivo en formato de día juliano, uno de los caracteres del nombre de los ficheros se puede utilizar para identificar el equipo. El carácter identificador, que se introduce en este campo, permite distinguir la procedencia de los datos con solo leer el nombre del fichero, lo cual resulta de gran utilidad en caso de tener el mismo evento registrado en varios módulos.

La figura 5.8 de la memoria muestra los valores de los parámetros utilizados en una aplicación real, durante una campaña de recogida de datos en el volcán Teide en el año 94.

III.C.1.b. POSIBLES ESTADOS DEL SISTEMA EN EL PROGRAMA REDSIS.EXE

Los módulos de dieciséis bits pueden pasar por distintas fases durante su operación. Estas fases o estados, que se muestran en la parte superior de la pantalla de monitorización (figura 5.9 de la memoria), son las siguientes:

- *Iniciación*: al arrancar el programa y durante un tiempo igual al preevento se inicializa el valor del LTA. Esta inicialización se realiza para evitar la posibilidad de que los valores iniciales arbitrarios de las variables STA y LTA puedan provocar un disparo en el arranque del programa.
- *Sincronismo*: en el caso de usar sincronismo con receptor GPS, la primera acción que realiza el programa es adquirir el tiempo GPS y sincronizar el contador de interrupciones con los flancos de segundo del GPS. Si transcurren cinco segundos sin que el programa detecte la presencia de la señal de pulsos de segundo, se abortaría la ejecución indicando que hay un error en el sistema GPS.

```
8   trazas
12404   muestras por traza
tomadas el
97-09-11
a las
11:57:48.00
 200  Hz muestreo
48 Hz 8 polos antialiasing
en las siguientes estaciones
BZ   muestra mayor   720
BNS   muestra mayor   1030
BEW   muestra mayor   1353
B1   muestra mayor   35
B2   muestra mayor   457
B3   muestra mayor   11383
B4   muestra mayor   3328
B5   muestra mayor   1792
----------------------------------------
Información complementaria
  B Identificacion Instrumento
CS5016 Serie Instrumento
Software May 95
MPS TSta TLta Pre Post Kdis Fil Dmax
200   2   16   16  32   1    1   150
Sta/Lta en incremento
```

Figura III.4.- *Contenido de un fichero ASCII de cabecera (2541157B.DTS), tal como se vería con un editor de textos convencional. Como puede observarse, el fichero ofrece información completa sobre los datos muestreados (incluyendo la fecha y hora de la primera muestra, el número de canales, la frecuencia de muestreo utilizada y los valores máximos de cada canal), la configuración hardware del equipo y los parámetros del algoritmo de disparo. Dado que el fichero corresponde a una campaña en la que se utilizaron conjuntamente varios módulos, se adoptó la nomenclatura basada en el día juliano para poder identificar los ficheros procedentes de cada equipo (en este caso, el B).*

- *Detectando*: después de la fase de sincronismo se pasa a la de detección, en la que el programa va adquiriendo las muestras del conversor A/D y las va acumulando en un búffer circular, de longitud igual al preevento programado. Además calcula, para cada canal activado para disparo, el cociente STA/LTA. En el momento en que un canal sobrepasa este umbral activa su condición de disparo y comprueba si hay más canales con la condición de disparo activa. Si esto ocurre en un número de canales igual o mayor que el valor de la variable *Estaciones para disparo* se activa el registro de la señal actual, entrando en la fase *Disparo*.

  Durante la fase *Detectando* el usuario puede obtener, a través de varias teclas, diversa información de las señales conectadas a cada canal:

  - *Tecla [F2]*: permite visualizar el nivel de los promedios STA y LTA del canal seleccionado.
  - *Tecla [F3]*: monitoriza las estaciones que han provocado un disparo en un momento determinado.
  - *Tecla [F4]*: permite acceder a la información de las ventanas temporales de registro si éstas han sido programadas. En caso de que no se hayan programado o de que el registro programado por ventana haya finalizado también se obtienen mensajes informativos.
  - *Tecla [Barra espaciadora]*: Provoca el inicio de un registro de duración igual a la duración máxima establecida en el archivo de configuración.
  - *Teclas de número ([1] a [8])*: cada una de las teclas de número hace que se visualice en pantalla, en tiempo real, la señal del canal correspondiente.
  - *Teclas [-] y [+]*: permiten modificar, en potencias de dos, la escala de visualización de la señal que se está adquiriendo.

- *Disparo*: en esta fase el programa va registrando en memoria las muestras del conversor y calcula el valor del intervalo STA. En el momento en que el valor del STA sea el mismo que antes del disparo se da por finalizado el evento y se pasa a la fase *Postevento*.
- *Postevento*: a partir del momento en que se entra en esta fase se registran en memoria los segundos de registro previamente configurados. La longitud total del evento es variable y está acotada entre los siguientes valores:

  Preevento + postevento < Duración del evento < Duración máxima

  Si se alcanza la duración máxima se entra en la fase *Desbordamiento*.

- *Desbordamiento*: al llegar a esta fase se deja de registrar en memoria RAM la señal y se almacena todo el registro que hay en la memoria en disco duro o disquete, entrando en la fase *Transferencia*.
- *Transferencia*: durante este estado el programa dedica todos los recursos del ordenador a almacenar el registro realizado en el dispositivo elegido, para lo cual desactiva la interrupción de muestreo. Al finalizar la grabación del registro el programa vuelve a la fase de iniciación.

Contenido del CD-ROM adjunto

Por brevedad los anexos no se han incluido en esta memoria, sino en formato digital en el CD adjunto. Los anexos comprenden los listados completos de los programas descritos en los capítulos 3 a 5, los ficheros gráficos de las tarjetas de circuito impreso y otra información referente a la operación de las tres antenas presentadas en esta tesis. Debido a su gran extensión, para facilitar su identificación se han nombrado con cuatro elementos, que se describen a continuación:

1. Número romano correspondiente a la antena (I: Gran Sasso, II: Vesubio, III: IAG).
2. Letra mayúscula correspondiente al contenido del anexo (A: programas, B: ficheros de circuito impreso, C: otros).
3. Número correspondiente al módulo que se describe (ej.: en la antena del Gran Sasso, 1: estaciones, 2: PCs nodales, 3: servidor, 4: oscilador patrón).
4. Letra minúscula para distinguir los distintos anexos correspondientes a los mismos módulos.

Así, como ejemplo, el anexo que contiene los listados de los programas del oscilador patrón de la antena del Gran Sasso sería el I.A.4, que contendría tantos apartados como ficheros se listen. En este caso, tres (a, b y c).

Además de los anexos, el CD-ROM incluye el documento correspondiente a esta memoria, en formato PDF, así como las presentaciones en congresos y otros documentos de difícil acceso a los que se ha hecho referencia en la tesis. En caso de que un documento esté incluido en el CD, este extremo se indica al final de la correspondiente referencia, en el apartado de bibliografía.