UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Sistema vestible para medida de biomarcadores

Autor
Álvaro García Ávila

Directores
Almudena Rivadeneyra Torres
Víctor Toral López

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Julio de 2023

Sistema vestible para medida de biomarcadores

Autor
Álvaro García Ávila

Directores
Almudena Rivadeneyra Torres
Víctor Toral López

SISTEMA VESTIBLE PARA LA MEDIDA DE BIOMARCADORES

Álvaro García Ávila

PALABRAS CLAVE: Vestible, Bioseñal, Reprogramabilidad, Flexible, Filtrado de señal, PSoC, Electrocardiograma, Fotopletismografía, Porcentaje de saturación de oxígeno en sangre, SpO_2

RESUMEN

Este Trabajo de Fin de Grado tiene como objetivo el desarrollo de un dispositivo que ofrezca suficiente flexibilidad y reprogramabilidad como para ser utilizado en diversas aplicaciones enfocadas a la adquisición de bioseñales y biomarcadores, es decir, se pretende crear un único dispositivo que mediante modificaciones de *software* y las mínimas posibles de *hardware*, se pueda utilizar en la adquisición de distintos parámetros y señales presentes en el cuerpo humano. En concreto, se va a ejemplificar como dicho dispositivo se puede emplear en la adquisición de la señal cardíaca y en el cálculo del porcentaje del nivel de saturación de oxígeno en sangre, haciendo cambios mínimos al apartado físico. Se pretende que según la aplicación, el dispositivo diseñado pueda ser utilizado en su función de dispositivo vestible. Para lograr estos objetivos, el proyecto está basado en la tecnología SoC (*System on a chip*), la cual permite aglutinar gran cantidad de componentes electrónicos en un mismo empaquetado de reducido tamaño.

Con la intencionalidad de simplificar el uso para un usuario final, se desarrolla una aplicación Android, la cual es capaz de comunicarse con el dispositivo creado vía *Bluetooth Low Energy*, mostrar los resultados obtenidos para la medida concreta que se esté realizando y almacenar estos para su posible consulta a futuro.

WEARABLE SYSTEM FOR BIOSIGNAL MEASUREMENTS

Álvaro García Ávila

KEYWORDS: Wearable, Biosignal, Reprogrammability, Flexible, Signal filtering, PSoC, Electrocardiogram, Photoplethysmography, Percentage of oxygen saturation in blood, SpO_2

ABSTRACT

This Bachelor's Thesis aims to develop a device that offers enough flexibility and reprogrammability to be used in various applications focused on the acquisition of biosignals and biomarkers, that is, it is intended to create a single device that through software modifications and the minimum possible modifications of hardware can be used in the acquisition of different parameters and signals present in the human body. Specifically, it is going to be exemplified how this device can be used in the acquisition of the cardiac signal and in the calculation of the percentage of oxygen saturation level in blood, making minimal changes to the physical section. It is intended that depending on the application, the designed device can be used in its function as a wearable device. To achieve these goals, the project is based on SoC (System on a chip) technology, which allows to agglutinate a large number of electronic components in the same small package.

With the intention of simplifying the use for an end user, an Android application is developed, which is able to communicate with the device created via Bluetooth Low Energy, display the results obtained for the specific measurement being performed and store them for possible future reference.

Dña. **Almudena Rivadeneyra Torres**, Profesora del Área de Electrónica del Departamento de Electrónica y Tecnología de Computadores de la Universidad de Granada.

D. **Víctor Toral López**, Investigador con Cargo a Proyecto del Departamento de Electrónica y Tecnología de Computadores de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***SISTEMA VESTIBLE PARA LA MEDIDA DE BIOMARCADORES***, ha sido realizado bajo su supervisión por **Álvaro García Ávila**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 13 de Julio de 2023 .

Los directores: Almudena Rivadeneyra Torres **Víctor Toral López**

Agradecimientos

Quisiera agradecer a todas aquellas personas sin las cuales no habría sido capaz de sacar este trabajo adelante y que han estado conmigo durante todo este tiempo.

En primer lugar a mi familia, que ha vivido conmigo no sólo la realización de este trabajo si no todas mis etapas estudiantiles, siempre han estado con su cariño y soporte incondicionales.

A mi pareja, que ha sufrido este trabajo como si fuese suyo, siempre apoyándome, mostrando su afecto, pasando momentos inolvidables juntos, y por supuesto leyendo y releyendo mil veces este documento junto a mi hasta encontrar ese sinónimo que teníamos en la punta de la lengua.

A mis tutores Almudena y Víctor por su constante ayuda e interés a lo largo de todo este trabajo y por sus muchos consejos durante todos estos meses.

Y por último a mis compañeros de laboratorio, que aunque siempre ocupados, no dudaron en sacar un hueco para ayudarme o hacer de sujetos de prueba, y sobre todo que hicieron esas largas sesiones en busca de fallos mucho más amenas.

Índice

Índice de Figuras

Índice de Tablas

Acrónimos

A | B | C | E | F | G | H | I | J | L | M | N | P | R | S | T | U | W | X

A

AC *Alternating Current* (Corriente alterna)

ADC *Analog to Digital Converter* (Conversor analógico-digital)

ASIC *Application Specific Integrated Circuit* (Circuito integrado para aplicación específica)

B

Bio-MEMS *Biological MEMS* (Sistemas microelectromecánicos biológicos)

BIO-Z *Bioelectrical impedance* (Impedancia bioeléctrica)

BLE *Bluetooth Low Energy* (Bluetooth de baja energía)

BOMs *Bill of Materials* (Lista de materiales)

C

CMM *Common Mode* (Modo común)

CMRR *Common Mode Rejection Ratio* (Factor de rechazo al modo común)

CNT *Carbon Nanotube* (Nanotubos de carbono)

CPU *Central Processing Unit* (Unidad central de procesamiento)

CRC *Cyclic Redundancy Check* (Verificación de redundancia cíclica)

D

DAC *Digital to Analog Converter* (Conversor digital-analógico)

DAP *Die Attach Paddle* (Paleta de fijación)

DC *Cirect Current* (Corriente continua)

DMA *Direct Memory Access* (Acceso directo a memoria)

DSB *Double-sided Bond* (Enlace de doble cara)

E

ECG *Electrocardiogram* (Electrocardiograma)

EDA *Electronic Design Automation* (Automatización de diseño electrónico)

EEPROM *Electrically Erasable Programmable Read-Only Memory* (Memoria de solo lectura programable y borrable eléctricamente)

EMG *Electromyography* (Electromiografía)

EMI *Electromagnetic Interference* (Interferencia electromagnética)

EOG *Electroocoulogram* (Electrooculograma)

F

FFC *Flexible Flat Cable* (Cable plano flexible)

FPA Filtro Paso Alta

FPB Filtro Paso Baja

G

GPIO *General Purpose Input/Output* (Entrada/Salida de propósito general)

GSR *Galvanic Skin Response* (Respuesta galvánica de la piel)

I

I²C *Inter-Integrated Circuit* (Circuito inter-integrado)

I/O *Input/Output* (Entrada/Salida)

IC *Integrated Circuit* (Circuito integrado)

IDE *Integrated development environment* (Entorno de desarrollo integrado)

IEEE *Institute of Electrical and Electronics Engineers* (Instituto de ingenieros eléctricos y electrónicos)

IIR *Infinite Impulse Response* (Respuesta infinita al impulso)

IoT *Respuesta infinita al impulso* (Internet de las cosas)

IR *Infrared* (Infrarrojo)

J

JTAG *Joint Test Action Group* (Grupo de Acción de Pruebas Conjuntas)

L

LA *Left Atrium* (Aurícula izquierda)

LED *Light Emitting Diode* (Diodo emisor de luz)

LV *Left Ventricle* (Ventrículo izquierdo)

M

Matlab *MATrix LABoratory* (Laboratorio de matrices)

MCU *Microcontroller Unit* (Microcontrolador)

MEMS *Micro-Electromechanical Systems* (Sistemas microelectromecánicos)

MLB *Multi-layer bond* (Enlace multicapa)

N

NASA *National Aeronautics and Space Administration* (Administración Nacional de la Aeronáutica y del Espacio)

NC *No Connection* (Sin conexión)

P

PCB *Printed Circuit Board* (Placa de circuito impreso)

PDMS *Polydimethylsiloxane* (Polidimetilsiloxano)

PET *Polyethylene Terephthalate* (Polietileno Tereftalato)

PI *Polyimide* (Poliimida)

PPG *Photoplethysmography* (Fotopletismografía)

PSoC Programable System on Chip (Sistema programable en chip)

PSRR *Power Supply Rejection Ratio* (Factor de Rechazo a Fuente de Alimentación)

PWM *Pulse Width Modulated* (Modulación por ancho de pulsos)

R

RA *Right Atrium* (Aurícula Derecha)

RAM *Random Access Memory* (Memoria de acceso aleatorio)

RMS *Root Mean Square* (Media cuadrática)

RV *Right Ventricle* (Ventrículo Derecho)

S

SaO_2 *Oxygen Saturation of Arterial Blood* (Saturación arterial de oxígeno)

SMD *Surface Mounting Device* (Dispositivo de montaje en superficie)

SMT *Surface Mount Technology* (Tecnología de montaje en superficie)

SoC *System on a Chip* (Sistema en chip)

SpO_2 *Oxygen Saturation* (Saturación de oxígeno)

SPI *Serial Peripheral Interface* (Interfaz periférica serie)

SPICE *Simulation Program with Integrated Circuits Emphasis* (Programa de simulación con énfasis en circuitos integrados)

SSB *Single-sided bond* (Enlace en una sola cara)

SWD *Serial Wire Debugging* (Depuración de cable serie)

T

THT *Through Hole Technology* (Tecnología de agujeros pasantes)

TIA *Transimpedance amplifier* (Amplificador de transimpedancia)

U

UART *Universal Asynchronous Receiver-Transmitter* (Receptor-transmisor asíncrono universal)

UDBS *Universal Digital Blocks* (Bloque digital universal)

USB *Universal Serial Bus* (Bus serie universal)

UUID *Universally Unique Identifier* (Identificador único universal)

W

WBAN *Wireless Body Area Network* (Red inalámbrica de área corporal)

WBSN *Wireless Body Sensor Network* (Red inalámbrica de sensores corporales)

WMSN *Wireless Medical Sensor Network* (Red inalámbrica de sensores médicos)

X

XML *Extensible Markup Language* (Lenguaje de marcado extensible)

Capítulo 1

Introducción

1.1. Resumen del proyecto

El objetivo de este Trabajo de Fin Grado es el desarrollo de un dispositivo vestible que ofrezca suficiente flexibilidad como para ser utilizado en diversas aplicaciones enfocadas a la adquisición de bioseñales y biomarcadores.

Para lograr este objetivo, se va desarrollar, en primer lugar, un dispositivo de procesamiento central. Este va a consistir en un instrumento reconfigurable y del menor tamaño posible, que se encargue de procesar (muestreo, acondicionamiento de señal, etc) y transmitir las muestras adquiridas. Este último no cuenta con sensorización de por sí, en su lugar dispone de diversos puertos de entrada que se pueden utilizar para la conexión de electrodos o kits de expansión que serán los encargados de aportar los sensores para la adquisición de señales. Dichos puertos están concebidos para lograr comunicación desde y/o hacia los kits que implementan las sensorización y para la alimentación de estos últimos, de manera que todo el sistema dependa de una única entrada de potencia. El procesador usado para este dispositivo estará basado en tecnologías PSoC (*Programable System on Chip*) desarrolladas por Cypress Semiconductor.

Una vez tomadas y tratadas las muestras concretas, estas se envían a un dispositivo móvil que cuenta con una aplicación que posibilita consultar los resultados obtenidos y almacenarlos para futuros estudios.

1.1.1. Objetivos a cumplir

Se puede establecer *grosso modo* que los objetivos a cumplir son los siguientes:

- Desarrollar un dispositivo vestible, de reducido tamaño y gasto energético, que ofrezca la suficiente reconfigurabilidad como para ser utilizado en el procesado y transmisión de diversas bioseñales y biomarcadores.
- Desarrollar el *firmware* y *hardware* necesario para adquirir una o varias medidas haciendo uso del sistema anterior. En este caso, el *hardware* que se tiene que diseñar es el respectivo al acondicionamiento del sensor que se utilice.
- Desarrollar una aplicación en Android que sea capaz de comunicarse con el dispositivo de procesamiento central para almacenar y mostrar a un usuario los resultados de las medidas. Esta aplicación también se utiliza para mantener el control de cuando se produce una adquisición.
- Tomar medidas con todos aquellos sistemas que se creen y comparar los resultados, caso de que sea posible, con los que obtendría un dispositivo comercial para comprobar el correcto diseño de cada uno.

1.2. Motivación

Cuando hablamos de un dispositivo vestible o *wearable*, como también son conocidos, lo estamos haciendo en el contexto de un dispositivo médico, el cual adopta la forma de un accesorio que se puede llevar puesto de manera confortable, idealmente imperceptible para el usuario, y que permite, mediante el uso de diversos sensores, adquirir parámetros fisiológicos del cuerpo humano. Entre estos se incluyen, por mencionar algunos, temperatura corporal, presión sanguínea, ritmo cardíaco, etc. En general, estos aparatos toman medidas las cuales posteriormente se envían a una base de datos para su análisis o directamente a una aplicación software para su consulta en tiempo real [1, 2].

Actualmente, nos hallamos en un contexto de población altamente envejecida, donde las estimaciones apuntan a que el porcentaje de la población considerada de tercera edad va a seguir en aumento en los próximos años, especialmente en los países desarrollados [3]. En España la edad media de la población en el año 2021 fue de 44.7 años [4] y se prevé que para el año 2040 habrá más de 24 millones de personas mayores de 50 años [5]. Este tipo de tendencias hacen que se espere un incremento sustancial sobre el gasto de los sistemas sanitarios [6]. En las Figuras 1.1 y 1.2 se pueden consultar, respectivamente, la edad media de la población en Europa y el Cáucaso por países y en España por comunidades autónomas.

A la población envejecida, hubo que sumar durante también durante un tiempo la situación del virus SARS-CoV-2, popularmente conocido como COVID-19, el cual produjo una situación de colapso sobre los sistemas sanitarios de una gran cantidad de países [7, 8, 9]. En esta situación el haber dispuesto de estos dispositivos podría haber ayudado a reducir la presión sobre los hospitales y con ello potencialmente aumentar la tasa de supervivencia a la enfermedad.

Debido a los motivos anteriores, la posibilidad de ser capaces de monitorizar de manera continuada a personas sin requerir hospitalización, se ha vuelto muy deseable. El uso de dispositivos vestibles puede ayudar a conseguir este requisito, reduciendo con ello el gasto de recursos a la par que se produce un incremento en la calidad de vida del paciente [2, 10, 11]. Por ejemplo, actividades físicas para rehabilitación se pueden realizar desde la comodidad e intimidad del hogar, a la par que se mantiene supervisión de la correcta realización de las mismas [12].

Nótese que aunque este trabajo se centra en el uso de estos dispositivos para aplicaciones médicas (medicina preventiva, rehabilitación, etc), su uso se extiende a muchos otros sectores tanto a nivel privado como profesional. Uno de su empleos muy comunes es la supervisión de actividades deportivas, tanto realizada por deportistas de élite como por aficionados como forma de ocio o salud, lo cual se ha visto popularizado para estos últimos por la aparición de relojes inteligentes y pulseras de actividad asequibles [10, 13, 14, 15]. Otros campos de aplicación son el sector militar donde, por ejemplo, mencionamos el sistema *Land Warrior* que permite desde un centro de control supervisar en tiempo real información como localización GPS (*Global Positioning System*) o el nivel de estrés del soldado [16]. Para finalizar, se hace mención a la utilidad en la exploración espacial, para ilustrarlo mencionamos al sistema *NASA's LifeGuard*, el cual posibilita mantener control de los signos vitales de los astronautas [17]. Se han mencionado sólo algunas de las posibilidades que da el uso de dispositivos vestibles; como se puede ver, estos dispositivos presentan un alto potencial y su uso se puede extender a variedad de campos de aplicación.

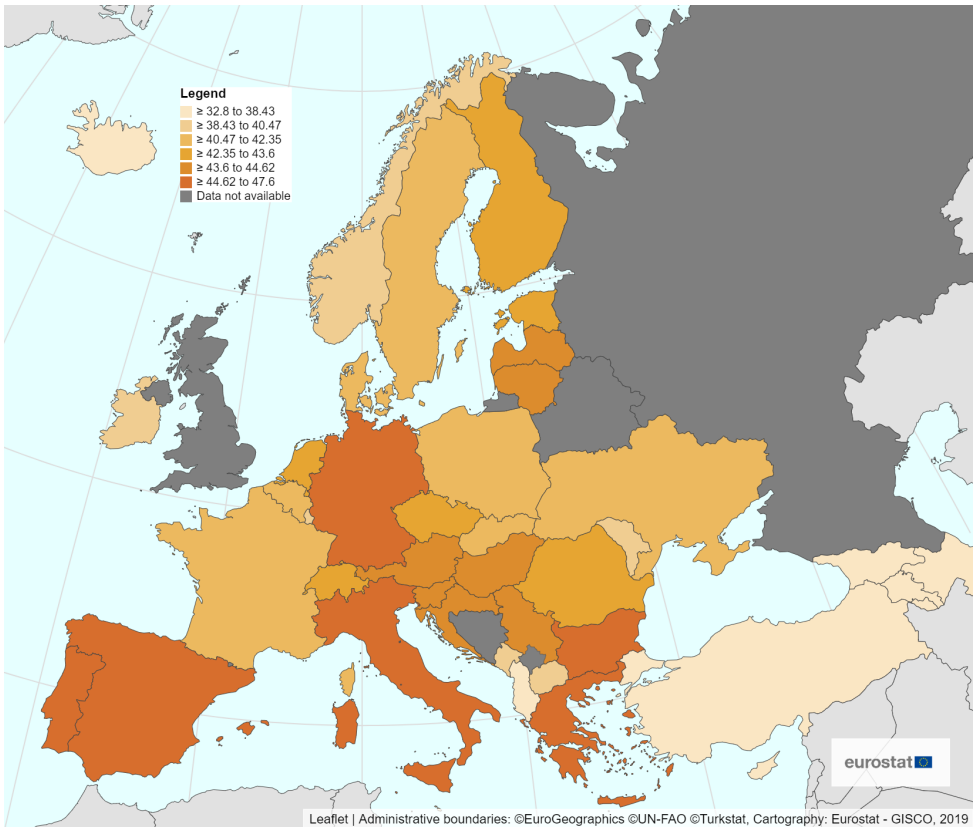


Figura 1.1: Edad media de la población en Europa y el Cáucaso en 2021. Datos disponibles en la página oficial de la Oficina Europea de Estadística [4].

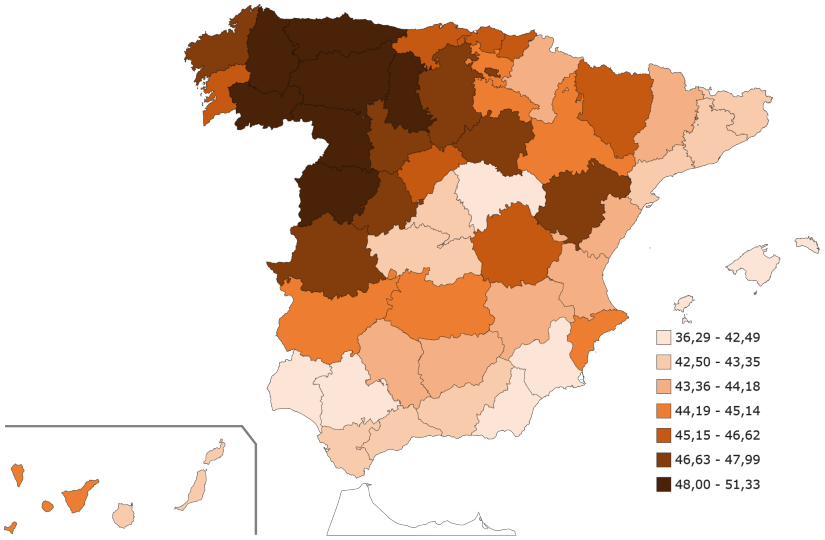


Figura 1.2: Edad media de la población por comunidades autónomas españolas en 2022. Datos disponibles en la página oficial del Instituto Nacional de Estadística (INE) [3].

1.3. Estado del Arte

El objetivo de este apartado es realizar un examen de algunos de los parámetros que se pueden extraer de manera no invasiva mediante un dispositivo vestible, así como que interés puede tener conocer cada uno de ellos y que aplicaciones se les puede dar. También se hace un análisis de los requerimientos que se le pueden exigir a los sensores empleados, tanto a nivel técnico como teniendo en cuenta el factor humano. Para acabar, se realiza un breve vistazo a los sensores que se pueden utilizar.

1.3.1. Campos de aplicación y parámetros extraíbles de manera no invasiva

A la hora de extraer diversos parámetros del cuerpo humano con un dispositivo médico, se pueden usar sensores invasivos y no invasivos. Por invasivos, se entienden aquellos que precisan de penetrar el cuerpo a través de una inyección o incisión, un ejemplo muy claro sería un análisis de sangre. Por otra parte, se tienen los no invasivos, estos permiten la obtención de parámetros sin requerir ningún tipo de incisión, lo que los hace mucho más atractivos para los pacientes [18]. A continuación, se puede consultar una recopilación de algunos de los diversos parámetros que son posibles de extraer de manera no invasiva, así como el interés que puede tener el conocer cada uno de ellos.

Presentamos los siguientes [1, 2, 19, 20]:

- **Temperatura corporal:** Es uno de los valores más elementales que se obtienen por su sencillez y la inmediatez con la que permite detectar anomalías. En situaciones normales, el cuerpo humano se halla en el rango de los 35 ºC a los 38 ºC, valores fuera de este rango permiten detectar de manera temprana hipertermias o hipotermias.
- **Ritmo cardíaco**: Se trata de otro parámetros básico. Se suele tomar en conjunto a otros parámetros, ya que, por ejemplo, un cambio en el ritmo del corazón afecta de igual manera a la presión sanguínea. La extracción de la señal cardíaca se puede usar en la detección de enfermedades de esta índole. Otro uso muy común que se le puede dar a conocer esta información, es supervisar la intensidad de una actividad deportiva. Para visualizar la señal del corazón se puede seguir la técnica conocida como electrocardiograma o ECG (*Electrocardiogram*).
- **Presión sanguínea**: Se trata de la presión arterial de la sangre que circula por el cuerpo. Es un parámetro que se puede ver afectado por obesidad, estrés o la edad. La presión sanguínea de una persona sana se encuentra por debajo del rango 80-120 mmHg (milímetros de mercurio), donde valores inferiores a 120 mmHg indican la sístole del corazón y valores inferiores a 80 mmHg la diástole. Se puede usar para detectar hipertensión. Como ya se ha comentado, es un valor que va intrínsecamente relacionado con el ritmo cardíaco, con lo cual es habitual adquirir ambos parámetros simultáneamente.
- **Ritmo respiratorio**: Es un indicador relativamente obvio de si una persona se encuentra en un buen estado de salud. Se puede ver alterado, por ejemplo, por edad avanzada, obesidad o la intensidad de una actividad deportiva. Cuando se diseña un instrumento para realizar la medida de este parámetro, resulta especialmente relevante que el sistema opere de manera inalámbrica, ya que en caso de el paciente sentir incomodidad, es muy probable que las medidas se vean alteradas.
- **Tasa de sudoración**: Se puede medir con doble intencionalidad, por un lado es posible usar un sensor capaz de medir el índice de pH en el sudor, índices anómalos pueden ser síntoma de enfermedad. Así mismo, se puede medir la cantidad de sudor segregado, ya que en caso de ser este muy elevado, en condiciones de actividad física, puede suponer deshidratación, lo que a su vez lleva a cansancio y fatiga. Esta medida es por tanto especialmente interesante a la hora de supervisar una actividad deportiva. Resulta de utilidad considerar que los sensores se integren en tejidos que estén directamente en contacto con el cuerpo para asegurar buenas mediciones.

- **Respuesta muscular**: Con esto se quiere hacer referencia a los distintos parámetros (movimiento, velocidad, fuerza...) que caracterizan el desplazamiento de una persona cuando esta realiza actividades tales como caminar, trotar o correr o cuando se realiza un movimiento muscular con, por ejemplo, los brazos o incluso los ojos. Estos valores permiten, en resumen, singularizar la locomoción de una persona y se pueden extraer usando acelerómetros o sensores de presión entre otros. La posibilidad de conocer estos datos tienen un rango muy amplio de usos, por ejemplo, en el contexto de una persona en edad avanzada, es muy común que exista el riesgo de caídas. Un dispositivo capaz de detectar que se ha dado esta situación podría ser capaz de dar la voz de alarma. También resulta de especial atractivo en la detección de la enfermedad de Párkinson, ya que esta se ve caracterizada por dificultades motrices, tales como, rigidez en las extremidades o movimientos lentos. Sin embargo, esta técnica no resulta útil sólo en la detección, si no también en determinar si un tratamiento de rehabilitación concreto está resultando el adecuado para un paciente en específico. Técnicas como la electromiografía (EMG) o el electrooculograma (EOG) permiten obtener respectivamente la señal eléctrica de los músculos esqueléticos y del movimiento de los ojos [21, 22].

- **Saturación de oxígeno en sangre**: La hemoglobina es una proteína de la sangre que permite, entre otras cosas, el transporte de oxígeno. Según si esta se encuentra completamente enlazada con oxígeno o no, se la califica respectivamente como hemoglobina oxigenada (HbO_2) y desoxigenada (Hb). Cada uno de estos dos tipos tiene una absorción lumínica diferente, con lo cual basándonos en este parámetro se puede extraer la cantidad de los dos tipos de hemoglobina presente mediante la técnica conocida como fotopletismografía o PPG (*photoplethysmography*). Uno de los usos de conocer este dato es detectar una hipoxemia, bajo nivel de oxígeno en sangre, durante una intervención que requiera de anestesia, ya que en caso de no tratarse este estado puede inducir en el fallecimiento del paciente [23, 24, 25].

Extrayendo de la información anterior, podemos concluir que algunas de la aplicaciones en potencia que tienen estos dispositivos son la asistencia a pacientes que sufran de: asma, diabetes, epilepsia, Alzheimer, obesidad, enfermedad de Párkinson, problemas cardiovasculares, desórdenes del sueño, etc. A esto se le suma la supervisión de deportistas de élite o la de personas que trabajen en entornos potencialmente peligrosos para la salud, así como la recolección general de datos para análisis clínicos y de investigación [1, 10, 24].

1.3.2. Requerimientos, posibles dificultades y consideraciones

A cualquier dispositivo vestible que aspire a tener éxito se le exigen una serie de características, las cuales deben de tener en cuenta tanto requerimientos a nivel técnico (consumo de energía, seguridad de las comunicaciones, etc), como el factor humano (ergonomía, comprensión de los resultados, etc). En cualquier caso, a nivel técnico, los dispositivos que finalmente se utilicen deben de cumplir las siguientes características básicas:

- **Bajo consumo**, lo que prolonga la vida de las baterías y con ello la autonomía del dispositivo [10, 26, 27]. Existen multitud de técnicas para optimizar el gasto energético, para ilustrar mencionamos: planificación de tareas, para que se ejecuten en instantes de tiempo determinados y mantener el procesador en *Sleep Mode* durante el resto del tiempo, o utilizar control dinámico del ciclo y frecuencia del reloj. También puede resultar interesante incluir técnicas para que el usuario o el propio dispositivo sea conocedor del nivel de batería y que de dicha forma se puedan tomar medidas concretas para tratar de prolongar esta en caso de que la recarga no sea posible [28]. Dado que, como veremos a continuación, se favorecen las comunicaciones inalámbricas, es importante optimizar la transmisión de datos para reducir el gasto energético total [10, 27].

- **Comunicación inalámbrica** siempre que sea posible para transmitir la información procesada a, por ejemplo, una aplicación móvil o un servicio en la nube [29]. Las redes de sensores inalámbricos usados para este tipo de aplicaciones pueden recibir diversas designaciones, tales como: WBANs (*Wireless Body Area Networks*) [1], WMSNs (*Wireless Medical Sensor Networks*) [30] o WBSN (*Wireless Body Sensor Network*) [26] según la bibliografía que se use como referencia. Al no existir en la actualidad un estándar de comunicación concreto para este tipo de redes, se pueden emplear multitud de los ya existentes como, por mencionar alguno: WiFi (IEEE 802.11), Bluetooth (IEEE 802.15.1) o ZigBee entre otros [10, 26, 27, 30].

- **Biocompatibilidad**. Debido a que los sensores se van a encontrar en contacto directo con el cuerpo humano, es necesario favorecer materiales que no causen una respuesta negativa por parte de este, induciendo por tanto a posibles lesiones como irritación de la piel, o incluso respuestas inmunes, esto último sobre todo se puede dar en el caso de que los sensores sean implantes [31].
- **Capacidad de postprocesado de datos**. Los sensores extraen señales sin procesar, es necesario que el dispositivo las acondicione y muestre de forma que sea interpretable por una persona de manera gráfica o numérica. Para ello, se tienen que incluir, por ejemplo, algoritmos para realizar filtrado de las señales captadas y posteriormente amplificar el resultado [10, 29, 32]. Adicionalmente, se espera del sistema también que sea capaz de trabajar en tiempo real [32].
- **Robustez** para minimizar dentro de lo posible el mantenimiento [10].
- **Bajo coste** para tratar de generar accesibilidad para la mayor cantidad posible de personas. Esto se relaciona directamente con la característica de la robustez, ya que el disponer de un dispositivo robusto y confiable abarata los costes en reemplazos o reparaciones.

Un componente imprescindible a tener en cuenta es el factor humano. Hay que asegurar que el dispositivo final resulte ergonómico y cómodo de vestir para tratar de evitar el rechazo de su uso por parte de los usuarios [2, 33]. Adicionalmente, el uso de dispositivos poco ergonómicos pueden afectar a los resultados obtenidos [34]. Como consideración adicional, es recomendable diseñar el aparato de manera discreta, con la intención de intentar interferir mínimamente en el día a día de un paciente que necesite de supervisión constante. Como guías generales se pueden tener en mente los principios de diseño universal. Estos son [35]:

1. Igualdad de uso: El dispositivo no debe de suponer un estigma para los usuarios.
2. Flexibilidad de uso: El diseño se acomoda a un rango de preferencias y habilidades personales.
3. Uso simple e intuitivo: El uso del dispositivo es fácil de entender, independientemente de la experiencia o conocimientos del usuario.
4. Información perceptible: El dispositivo comunica la información necesaria de manera efectiva al usuario.
5. Tolerancia de errores: El diseño minimiza riesgos y consecuencias adversas de acciones involuntarias o accidentales.
6. Mínimo esfuerzo físico: El diseño se puede usar de manera efectiva y cómoda con un mínimo de fatiga.
7. Forma y tamaño adecuados: La forma y tamaño deben de ser adecuadas para el uso planteado, independientemente del tamaño corporal, postura o movilidad del usuario.

Debido a la naturaleza de los datos con los que se trabaja en este tipo de aplicaciones, es imprescindible hacer énfasis en unas condiciones adecuadas de seguridad, privacidad y confidencialidad de los datos del usuario [23, 30]. Por ejemplo, hay que disponer de medios para asegurar que la integridad de los datos recibidos desde el lado del paciente, es decir, se tiene que poder detectar si un tercero ha modificado los datos originales durante el proceso de transmisión de los mismos [18].

1.3.3. Sensores

Los sensores empleados pueden ser tanto flexibles como rígidos. En cuanto a los rígidos, suelen estar basados en silicio, debido a lo cual se puede reutilizar técnicas de fabricación en masa ya conocidas, abaratando los costes en tiradas grandes. Entre ellos podemos mencionar los sensores MEMS (Micro-Electromechanical Systems), quienes se pueden dividir a su vez en sensores piezoeléctricos, capacitivos, electromagnéticos y piezoresistivos. Cuando se utilizan para detección de parámetros biológicos, como por ejemplo glucosa, pasan a ser conocidos como Bio-MEMS (Biological MEMS) [36, 37, 38].

En cuanto a los flexibles, presentan ciertas ventajas como: ser más finos y ligeros que sus contraparte, lo que los hace más cómodos de portar, costes reducidos, tanto de fabricación como de gastos energéticos, y en general mayor resistencia a golpes [18, 39]. Debido a estas ventajas, el uso de sensores flexibles resulta preferible cuando sea posible su implementación y es en ellos en los que nos centramos a continuación. Por flexible entendemos un sensor fabricado con materiales que le permiten cierta maleabilidad sin cambiar sus propiedades [39].

Es necesario seleccionar los materiales adecuados de manera que se alcance un compromiso entre el coste de fabricación y el desempeño obtenido. Se emplean dos materiales, por un lado un sustrato sobre el que se imprimirá el material conductor, y por el otro, el propio electrodo. Para la elaboración del sustrato se pueden usar polímeros orgánicos, inorgánicos y sintéticos, mencionamos el polidimetilsiloxano (PDMS), el tereftalato de politileno (PET) y la poliimida (PI). Para los electrodos se pueden escoger nanotubos de carbono (CNT), grafeno o aluminio [40]. Otros materiales a contemplar para el electrodo pueden ser plata u oro, pero estos se suelen descartar debido al alto coste que supone emplear metales preciosos [40, 41]. Los elementos aquí mencionados son solo una pequeña selección de la variedad de materiales que se pueden emplear para la manufacturación del sustrato y del electrodo.

En la literatura se dividen las tecnologías de impresión usadas en dos grupos, la impresión sin contacto (*Non-Contact Printing Technologies*) y con contacto *(Contact Printing Technologies*). Entre las técnicas de fabricación que caen dentro de la categoría de impresión sin contacto, podemos mencionar: *Screen Printing* e *Inkjet Printing*. Respecto a métodos usados en tecnologías con contacto, se citan: *Gravure Printing*, *Gravure-Offset Printing*, *Flexographic Printing*, *Micro-Contact* (uCP) y *Nano-Imprinting* (NI) [39, 41].

1.4. Comparativa entre este trabajo y otros proyectos de vestibles

En la tabla 1.1 se presenta una comparativa entre el diseño con dispositivos vestibles que se ha realizado durante este trabajo y otras alternativas desarrolladas en proyectos similares. Dado que una de las principales bioseñales que se pretenden obtener en este trabajo es la del corazón mediante un ECG, en la tabla comparativa se ha hecho especial énfasis en incluir otros trabajos que también realicen esta medida. La mayoría de los casos estudiados emplean circuitos integrados de aplicación específica o ASIC (*Application Specific Integrated Circuit*), es decir, se diseña una circuitería a medida que únicamente es capaz de procesar señales o parámetros concretos y donde no es posible realizar modificaciones para añadir funcionalidades adicionales.

Cuando en la tabla se indica que un dispositivo es parcialmente flexible, se viene a referir a que como ocurre, por ejemplo, en [48], los sensores que se usan para tomar las medida si que son flexibles, pero el dispositivo incluye también una parte rígida que alberga los circuitos para el procesamiento de la señal y las comunicaciones. En la Figura 1.3, se puede consultar una imagen de dicho proyecto a modo de ejemplo.

La columna "Seco/Húmedo" hace referencia, respectivamente, a si los sensores pueden tomar medidas directamente sobre la piel o si previamente hay que aplicar un gel para reducir la impedancia entre esta y el sensor. Todos los proyectos comparados optan por el empleo de sensores "secos", ya que esto hace el uso del dispositivo mucho más cómodo y atractivo de cara al usuario. El empleo de este tipo de sensores presenta además la ventaja de prevenir posibles reacciones alérgicas que puede causar el uso de geles [50, 51].

La adquisición del ECG se ha validado utilizando parches húmedos fabricados por Ambu, debido a la alta disponibilidad de estos durante el desarrollo del proyecto. Más información al respecto de dichos parches se puede localizar en el apartado Obtención del electrocardiograma (ECG) del Capítulo 4. Por otra parte, para la estimación de la saturación de oxígeno si que se usa un sensor seco. Por este motivo, se ha mencionado en la Tabla 1.1 que el hecho de que el que los sensores sean secos o húmedos depende enteramente de la aplicación concreta.

Flexible	Tipo de vestible	Seco/ Húmedo	Medidas	Reconfigurable	Ref
No	Pulsera	Seco	ECG/PPG/BIO-Z/GSR	No	[42]
No	Pulsera	Seco	ECG/PPG/Temp/ Humedad de la piel	No	[43]
Parcial	Camiseta	Seco	ECG/PPG	No	[44]
Si	Parche	Seco	ECG/PPG/Temp	No	[45]
No	Pulsera	Seco	ECG	No	[46]
Si	Parche	Seco	ECG/GSR/Temp	No	[47]
Parcial	Parche	Seco	EMG	Si	[48]
Si	Lente	Seco	Glucosa	No	[49]
Parcial	Parche	Depende de medida	ECG/SpO_2	Si	Este trabajo

Tabla 1.1: Comparativa entre varios proyectos de dispositivos médicos vestibles.

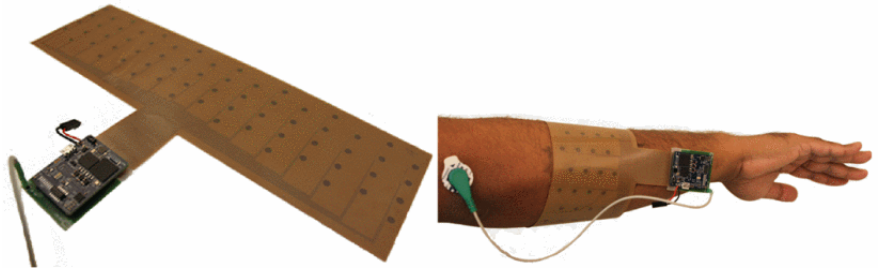


Figura 1.3: Ejemplo de dispositivo vestible parcialmente flexible para medida del EMG [48].

1.5. Introducción a las herramientas software utilizadas

En esta sección se encuentra la información relevante al respecto del *software* que se ha utilizado para el desarrollo del trabajo. Esto incluye a las herramientas usadas para el diseño y simulación de circuitos, así como el desarrollo de *firmware* y de la aplicación Android. Además del programa utilizado para tratar y representar las muestras una vez almacenadas en un ordenador.

1.5.1. Altium Designer

Un *software* EDA (*Electronic Design Automation*) es un programa utilizado para el diseño asistido de circuitería por ordenador. Para el desarrollo de este trabajo se ha escogido **Altium Designer** (en su versión 21.1.1) como *software* sobre el que desarrollar el diseño de la PCB que sintetiza el proyecto.

Altium Designer es un *software* de diseño profesional que posibilita la simulación de circuitos, su representación lógica mediante esquemáticos, el diseño asistido de PCBs y la generación de documentación y archivos de fabricación tales como BOMs (*Bill Of Materials*), modelos 3D de la PCB, archivos Gerber, etc [52].

A la hora de añadir un componente en Altium es necesario crear su símbolo y su *footprint*. El símbolo refleja la funcionalidad de un componente y las conexiones con las que cuenta. Estos se utilizan en la creación de esquemáticos, los cuales vienen a ser una representación lógica de un circuito

electrónico donde se asignan las conexiones entre componentes y se definen los nodos del circuito. Por otra parte, un *footprint* consiste en un patrón que representa los agujeros (si se usa tecnología de agujero pasante) o zonas de cobre expuestas (si se usa montaje superficial) necesarias para soldar el componente a la placa. En los *footprints* se pueden también definir capas de material no conductor que se pueden utilizar, por ejemplo, para añadir siglas que permitan identificar rápidamente los componentes en la PCB. En la Figura 1.4 se puede consultar un ejemplo de estos identificadores.

Los símbolos y los *footprints* se almacenan respectivamente en librerías SchLib y PcbLib, de modo que una vez creado un componente se puede reutilizar en otros proyectos. En la librería SchLib, se pueden asociar a cada símbolo uno o varios *footprints*. Adicionalmente, en la librería PcbLib se puede asociar también modelados en 3D a cada *footprint*, de manera que se pueden obtener modelados orientativos de la PCB previamente a su montaje. Las librerías suelen estar proporcionadas por los propios fabricantes de componentes, aunque también se pueden crear o modificar de manera manual. En este trabajo se han utilizado tanto componentes de librerías disponibles en repositorios públicos, tales como Ultra Librarian [53], como otros creados manualmente a partir de la información disponible en las hojas de datos de cada uno.

Una vez finalizado el esquemático, el circuito planteado de manera lógica se puede exportar al editor de PCBs para su diseño físico. Para ello se exporta, de manera automática, cada símbolo presente en el esquemático como su *footprint* correspondiente. Durante el diseño de la PCB, Altium asiste mostrando que puntos deben de conectarse según se estipuló en el esquemático previo y resaltando posibles errores. De hecho, en circuitos sencillos se puede utilizar la función de Altium para enrutamiento automático de los nodos.

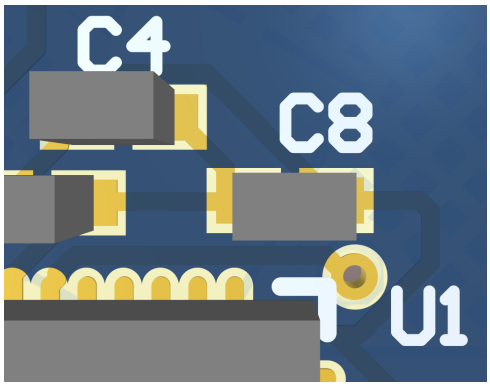


Figura 1.4: Capas con texto para identificar el PSoC 5LP y dos condensadores en la PCB.

1.5.2. LTspice

A la hora de comprobar el correcto funcionamiento de algunos diseños *hardware* se ha utilizado LTspice, el cual viene a ser un *software* gratuito para simulación de circuitos analógicos basado en SPICE (*Simulation Program with Integrated Circuit Emphasis*) [54].

De manera resumida, LTspice permite diseñar el esquema de un circuito electrónico y visualizar las formas de onda y niveles de voltaje y corriente en todos los nodos de este. Entre otras cosas da la posibilidad de simular: barridos de tensión o frecuencia, respuesta en función del tiempo, operación en corriente continua, etc. Las simulaciones se pueden correr usando componentes ideales o añadiendo no idealidades, como, por ejemplo, limitaciones en el ancho de banda de un amplificador o los elementos parásitos en un componente pasivo como puede ser una resistencia, un condensador o una bobina.

El programa permite crear y descargan el modelado de componentes electrónicos reales, de manera que se puedan conseguir simulaciones lo más parecido a la realidad posible. El programa no soporta, eso si, el diseño de PCBs.

1.5.3. MATLAB

MATLAB, cuyo nombre nombre viene a ser la abreviatura de *MATrix LABoratory* o laboratorio de matrices, es un *software* privativo de cálculo numérico que posibilita la manipulación de matrices y la representación de funciones. Esto permite que MATLAB se pueda utilizar para el diseño e implementación de sistemas de control, el procesamiento de señales digitalizadas, *machine learning*, procesamiento de imágenes, etc [55]. Durante la realización de este proyecto se ha utilizado la versión de MATLAB R2018b.

En este trabajo, MATLAB se utilizará para la visualización y el tratamiento de las señales adquiridas desde el dispositivo de procesamiento central. Aunque en la aplicación desarrollada en Android también se harán dichas acciones, resulta más cómodo utilizar Matlab durante las fases experimentales del proyecto. Adicionalmente, permite hacer de manera sencilla pruebas con diversos filtrados previamente a implementarlos.

1.5.4. PSoC Creator

Toda la programación hecha en los dispositivos PSoC usados se ha desarrollado empleando la herramienta PSoC Creator (versión 4.4), en esta sección del capítulo se hace una breve introducción a este programa. PSoC Creator es un entorno de desarrollo integrado o IDE (*Integrated Design Environment*) concebido para la creación y depurado de *firmware* en dispositivos PSoC y FM0+ y creado por Cypress Semiconductor Corporation. Para el desarrollo de *software* con este programa se puede utilizar el lenguaje C, aunque también es posible emplear Verilog para ciertas tareas. [56, 57].

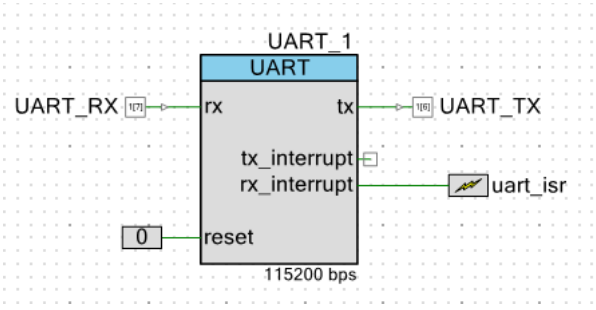
El empleo de esta herramienta facilita la producción de *firmware* gracias a que cuenta con librerías de componentes prefabricados que permiten configurar elementos internos de los PSoCs de manera sencilla, como lo pueden ser temporizadores (*timers*), contadores o multiplexores por poner un ejemplo. Estos componentes se añaden de manera gráfica en esquemáticos que permiten asignar las conexiones entre dichos elementos, los cuales incluyen también a los pines de entrada y salida. Posteriormente, se puede acceder a cada componente usado en el esquemático para configurar sus propiedades. Para clarificar estos conceptos, en la Figura 1.5 se ha incluido un ejemplo de un bloque UART conectado a varios pines en un esquemático y el menú de configuración del mismo.

Una vez creado el esquemático, se puede hacer la programación del procesador mediante código C. Cuando se añade un bloque al esquemático y se compila el proyecto, el programa genera automáticamente una serie de librerías en C que permiten el control de los bloques mediante *firmware*. Para ello, en el código principal se puede hacer llamada a los métodos generados para cada elemento. En la Figura 1.6 se puede consultar una lista con algunos de los métodos que se pueden usar con el bloque UART, la lista completa se puede consultar en [58].

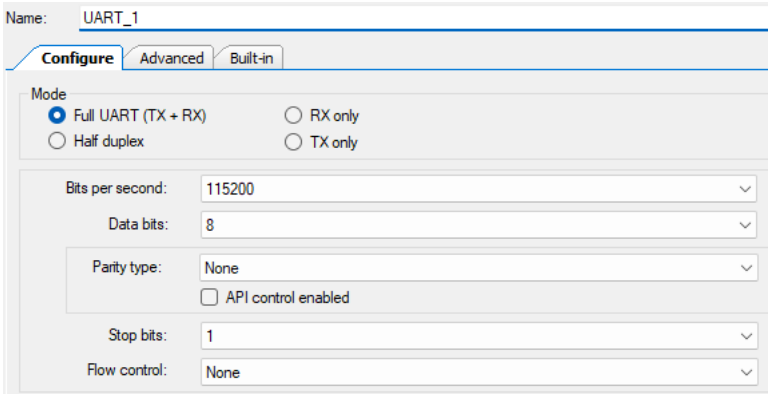
Finalmente, una vez se ha creado un esquemático y el código C necesario, se realiza la asignación de pines en caso de que se usen entradas y salidas, lo cual es habitual en la gran mayoría de aplicaciones. En la Figura 1.7 se muestra un ejemplo genérico del menú de asignación.

Otras características destacables de este programa son:

- El acceso inmediato a documentación extensa, ya que al seleccionar un elemento en el esquemático se proporciona la opción de abrir la hoja de datos del mismo, la cual en muchas ocasiones incluye ejemplos de uso.
- Un registro del número de componentes totales usados y cuantos restantes quedan disponibles según el modelo y familia de PSoC sobre el que se indique que se está trabajando.



(a) Bloque UART con conexiones a pines de entrada y salida.



(b) Configuración del bloque UART.

Figura 1.5: Ejemplo de añadir y configurar componentes en un esquemático de PSoC Creator.

Function	Description
UART_Start()	Initializes and enables the UART operation
UART_Stop()	Disables the UART operation
UART_ReadControlRegister()	Returns the current value of the control register
UART_WriteControlRegister()	Writes an 8-bit value into the control register
UART_EnableRxInt()	Enables the internal interrupt irq
UART_DisableRxInt()	Disables the internal interrupt irq
UART_SetRxInterruptMode()	Configures the RX interrupt sources enabled
UART_ReadRxData()	Returns the data in the RX Data register
UART_ReadRxStatus()	Returns the current state of the status register

Figura 1.6: Listado con algunos de los métodos que se pueden emplear en el control del bloque UART en PSoC Creator. Créditos [58].

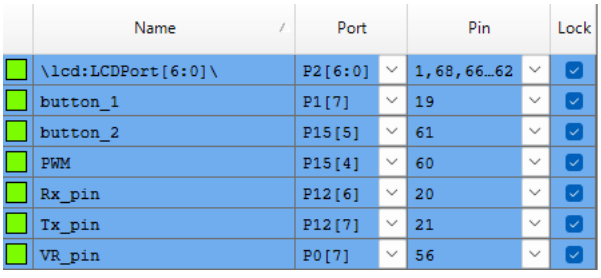


Figura 1.7: Ejemplo genérico del menú de asignación para los pines de entrada/salida en un PSoC mediante PSoC Creator.

1.5.5. Android Studio

Para el diseño de una aplicación en Android se ha utilizado el *software* Android Studio (versión 2022.1.1). Este es el entorno de desarrollo integrado (IDE) oficial para la creación de aplicaciones en dispositivos Android en el lenguaje Java o Kotlin, aunque se incluye también compatibilidad para desarrollo en el lenguaje C++. Entre las funciones que incluye este IDE se encuentran [59]:

- La posibilidad de realizar la gestión de la apariencia de la aplicación de manera gráfica. Esto permite añadir de manera rápida diversas características a la aplicación, tales como: botones, vista de imágenes, diversidad de *widgets*, etc y configurar de manera rápida el apartado visual de estos elementos y sus funciones. Todas las propiedades de los elementos editados quedan almacenadas en un archivo en formato XML (*Extensible Markup Language*) y se pueden modificar también mediante desde él en caso de resultar más cómodo para el desarrollador.
- Emulador que permite hacer pruebas sobre la app en estado de desarrollo en cualquier versión de Android. Esto también incluye la posibilidad de modificar los tamaños de las pantallas que muestran la aplicación, para asegurar que el apartado gráfico de la misma es correcto entre dispositivos de diversos tamaños.
- En caso de desearse hacer pruebas en un entorno real sobre un dispositivo físico, se incluye la posibilidad de instalar las apps en desarrollo en dicho dispositivo mediante conexión USB. Esto también da acceso a herramientas de depuración desde una terminal y a la consulta de un registro de los eventos ocurridos en el dispositivo.

Los proyectos en Android Studio se organizan mediante un sistema de ficheros que separa estos en las carpetas [59]:

- **manifests**: Aquí se incluye el archivo "AndroidManifest.xml". Este último se trata de la raíz fuente del proyecto y por tanto en el se declaran: el nombre del paquete de la aplicación, los componentes de la app, lo cual incluye actividades, servicios, receptores de emisiones y proveedores de contenido y los permisos que la aplicación precisa para su funcionamiento (estos varían según la versión de Android) [60].
- **java**: Incluye el código fuente Java o Kotlin.
- **res**: Alberga el resto de recursos, como puden venir a ser: diseños XML, cadenas de IU e imágenes de mapa de bits.

1.6. Estructura del proyecto

En esta sección se puede consultar la organización general que siguen los capítulos de este documento, así como un breve resumen de que información concreta de trata en cada uno de ellos.

- **Capítulo 1: Introducción.** En este capítulo se puede encontrar una descripción en términos generales del trabajo desarrollado y las razones que motivan el interés en seguir trabajando en dispositivos médicos vestibles. También se puede consultar un resumen del estado del arte de la tecnología actual, una comparativa entre este trabajo y otros similares, las herramientas *software* usada para el desarrollo del *firmware* y *hardware* creado y, para acabar, que organización temporal se ha seguido para la elaboración del proyecto.
- **Capítulo 2: Especificaciones del dispositivo de procesamiento central.** En este capítulo se tratan los objetivos que se plantean cumplir con el dispositivo de procesamiento central, así como que tareas se desea que realice, que componentes electrónicos serán necesarios para poder realizar dichas tareas y el proceso de selección de estos últimos.
- **Capítulo 3: Diseño hardware del dispositivo de procesamiento central.** En esta sección del trabajo se hace un diseño físico, mediante tecnología PCB, del circuito planteado en el capítulo anterior. Entre la información aportada se encuentra: las consideraciones hechas para cada componente electrónico como, por ejemplo, que componentes pasivos necesitan de manera externa para funcionar correctamente o como se les proporciona alimentación, las conexiones entre dichos componentes y finalmente el enrutado y colocación de todos los elementos en la PCB.

- **Capítulo 4: Diseño firmware y hardware para la obtención de adquisiciones.** Este capítulo trata el diseño de aplicaciones específicas usando el dispositivo desarrollado en los dos capítulos anteriores. Se cubre tanto el desarrollo *firmware*, como el *hardware* externo que sea necesario añadir.
- **Capítulo 5: Diseño de aplicación compatible en Android.** En esta parte se puede consultar la aplicación Android desarrollada para comunicarse mediante Bluetooth Low Energy con el dispositivo creado y posteriormente representar y almacenar los valores transmitidos desde este.
- **Capítulo 6: Resultados y adquisiciones.** Aquí se trata la información respectiva a las bioseñales que se han medido en personas reales utilizando el dispositivo desarrollado en este proyecto.
- **Capítulo 7: Conclusiones y lineas de trabajo futuras.** En este último capítulo se tratan los objetivos logrados durante la realización del proyecto y que líneas de trabajo futuras se plantean.

Se incluyen también varios anexos con información complementaria al trabajo, como puede ser: el código fuente de las aplicaciones desarrolladas o fotografías del sistema físico.

Capítulo 2

Especificaciones del dispositivo de procesamiento central

Este capítulo se trata de establecer que objetivos se pretenden cumplir con el dispositivo de procesamiento central y que componentes electrónicos son necesarios para ello. El capítulo se distribuye de la siguiente forma:

- En la sección Síntesis de las capacidades del dispositivo se proyectan los objetivos a cumplir con el dispositivo, así como que requerimientos se exigen basándonos en lo que se planteó en el Capítulo 1: Introducción. Esta información se complementa con los esquemas de alto nivel que ilustra la lógica del sistema en el apartado Arquitectura lógica del sistema.
- En el apartado Placas de circuito impreso (PCB) se puede encontrar un breve resumen el estado del arte de la tecnología PCB, acompañado de la elección del tipo más adecuado para este proyecto.
- En la sección Selección de componentes se mencionan los componentes electrónicos necesarios, el fundamento teórico de cada uno y el proceso de selección de un modelo concreto para cada caso. El listado final de los componentes seleccionados se encuentra de manera resumida en el apartado Listado de componentes.

2.1. Síntesis de las capacidades del dispositivo

Se plantea un dispositivo empleado como centro de recepción, procesamiento y transmisión de la información adquirida desde sensores. El objetivo de este aparato es plantear una plataforma reprogramable que, con los mínimos cambios posibles, se pueda reconfigurar para diversas aplicaciones de adquisición de biomarcadores y señales que puedan resultar de utilidad en aplicaciones médicas. Este enfoque presenta la ventaja de que cada vez que se desee añadir capacidades para adquirir un nuevo parámetro, no sea necesario realizar un rediseño completo del *hardware*, a diferencia de lo que ocurre, por ejemplo, en los sistemas ASIC, con lo cual se consiguen ahorrar costes y se acortan tiempos de desarrollo de cara a trabajos futuros.

El dispositivo en sí no cuenta capacidad de sensorización, en su lugar se hace uso de cuatro puertos de entrada para conectar, mediante cableado, los sensores usados para la adquisición de muestras. Por un lado, el procesador encargado del tratamiento de la información recibe por los puertos la señal (analógica o digital) proveniente de los sensores, y por el otro, también se usan para proporcionar alimentación a dichos sensores caso de que sea necesario. Hay que tener en cuenta que aunque el dispositivo principal sólo tiene que ser reconfigurado por *software* para cada nueva aplicación, si que es cierto que el *hardware* necesario para la adquisición de cada señal o valor si que será diferente. Sin embargo, las capacidades de estos puertos permiten que se puedan diseñar kits de expansión, de manera que solo hay que construir el *hardware* específico para la muestra concreta sin tener que alterar el sistema completo, el cual se podría seguir usando para otras aplicaciones con no más que cambiando el kit o sensor conectado a los puertos y cargando en este dispositivo central el nuevo *firmware*.

Se dispone de *switches* digitales para que, por *software*, sea posible mantener el control sobre los puertos que reciben alimentación. Uno de los *switches* controla al mismo tiempo el encendido o apagado del amplificador diferencial que incluye el sistema.

Como ventaja adicional, el hecho de que las adquisiciones se realicen de manera externa desde kits de expansión, permite personalizar las soluciones de cara a cada usuario concreto. Por ejemplo, si se disponen de kits para la adquisición del nivel de saturación de oxigeno en sangre y la temperatura corporal, en caso de que el usuario no desee adquirir una de estas medidas se puede prescindir de dicho *hardware* de manera sencilla, de forma que cada usuario obtenga únicamente los parámetros que le resulten de interés.

Sin embargo, también se plantea el inconveniente de que dado que el dispositivo tiene que estar preparado para adquisiciones que pueden variar en gran medida en, por poner un ejemplo, los recursos que necesitan para el procesamiento de la señal, este dispositivo quedará sobredimensionado en muchas situaciones de uso. Esto es, que el dispositivo dispone de muchos más recursos de los que en realidad se necesitan.

Como mencionado, la segunda funcionalidad de este aparato es la transmisión de la información una vez procesada. Se busca que esta se envíe a un ordenador o dispositivo móvil mediante comunicación Bluetooth de baja energía (*Bluetooth Low Energy* o BLE) o UART, de forma que pueda ser consultada y/o almacenada de manera cómoda por el usuario.

2.1.1. Otros requerimientos

Basándonos en los requerimientos que se estudiaban en el apartado Requerimientos, posibles dificultades y consideraciones del Capítulo 1: Introducción, se van a plantear los requisitos que hay que cumplir a la hora de diseñar el dispositivo central descrito teniendo en cuenta que se trata de un dispositivo vestible.

- Dado que el usuario final debe de llevar este dispositivo puesto, se deben de logran el tamaño y peso más bajos posibles.
- Relacionado con el punto anterior, la alimentación se debe de proporcionar mediante baterías ligeras y de pequeño tamaño para poder facilitar la portabilidad. Por ello, el consumo de potencia se tiene que optimizar con la intención de prolongar la autonomía. Idealmente, el dispositivo debe de ser autoconsciente del nivel restante de batería y avisar al usuario en caso de ser este bajo.
- Resulta adecuado que el sistema acepte fuentes de alimentación externas que abarquen un rango de valores de voltaje e intensidad en lugar de requerir niveles de entrada fijos, esto se hace con la intención de dar mayor flexibilidad al usuario final.
- Dado que el dispositivo opera en las inmediaciones del cuerpo humano, se hace especialmente necesario evitar que queden expuestas partes metálicas que puedan suponer daños al usuario o al aparato.

2.2. Arquitectura lógica del sistema

En este apartado nos centramos en plantear un diagrama de bloques que ejemplifique el funcionamiento a rasgos generales del dispositivo de transmisión y procesamiento central. En la Figura 2.1 se puede consultar el diagrama de bloques del sistema completo y en la 2.2 el del sistema de alimentación de manera más detallada.

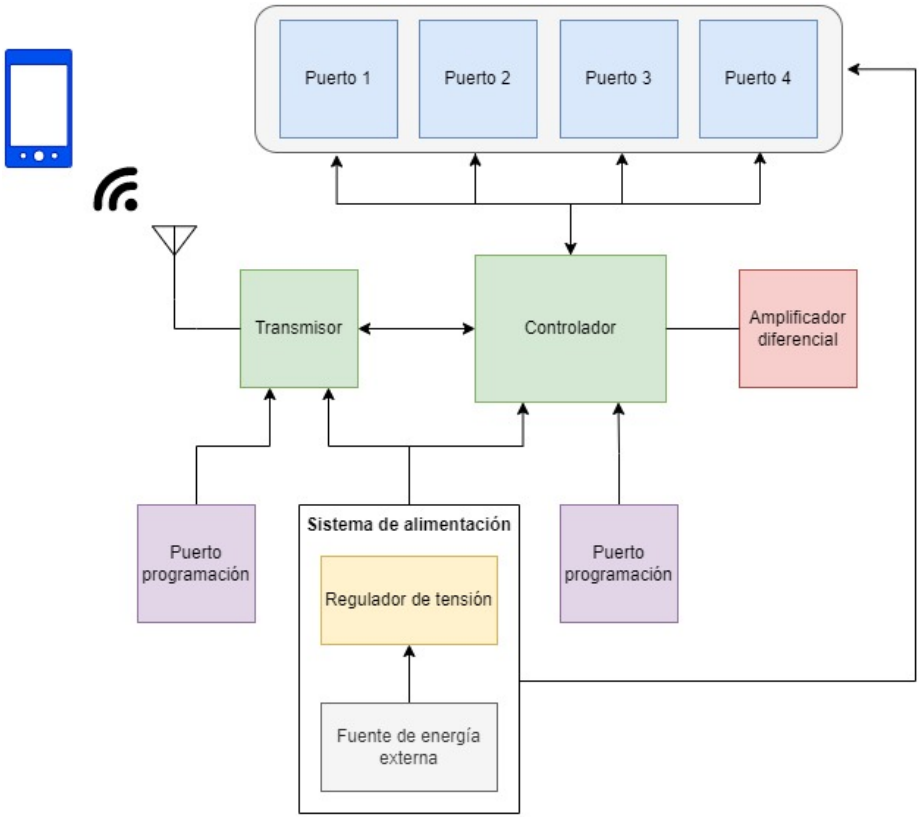


Figura 2.1: Diagrama de bloques del sistema central de procesamiento y transmisión.

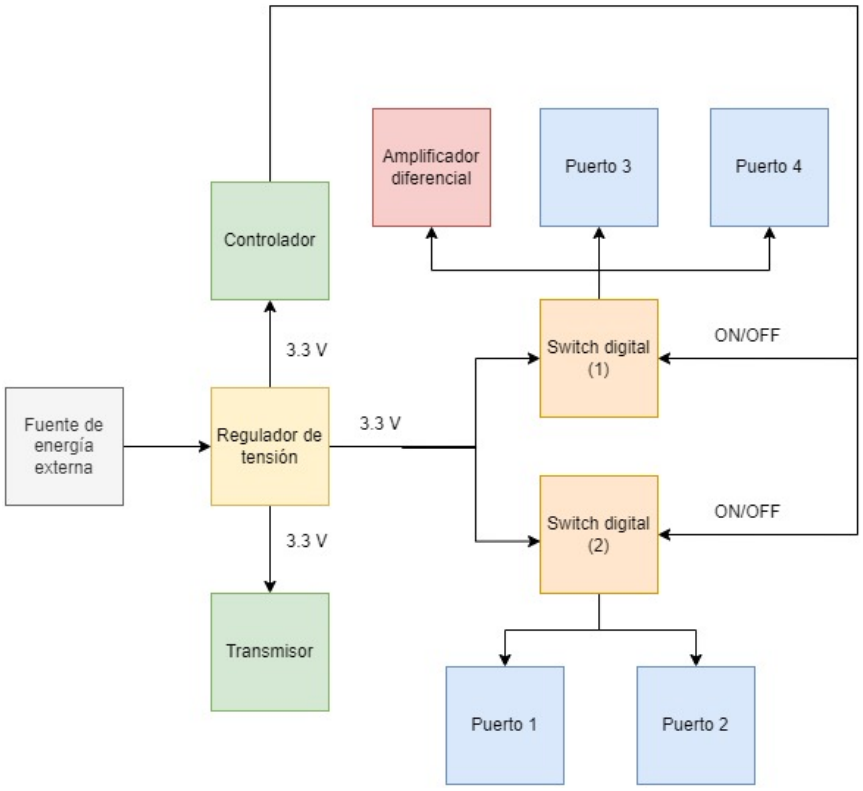


Figura 2.2: Diagrama de bloques de la alimentación del sistema central y los componentes conectados en los puertos.

2.3. Placas de circuito impreso (PCB)

A día de hoy, las placas de circuito impreso (PCB o *Printed Circuit Board*) son la tecnología predilecta para el montaje de dispositivos electrónicos empleados en electrónica de consumo (ordenadores personales, dispositivos móviles, equipos de sonido...), equipamiento médico, componentes para la industria automovilística, aeroespacial y marítima, etc [61, 62]. Debido a ser este el estándar en la elaboración de dispositivos electrónicos, es el método seleccionado para construir el sistema descrito en este trabajo.

Una PCB consiste en una lámina, fabricada habitualmente en fibra de cristal aunque se pueden utilizar otros materiales, sobre la cual se graban o imprimen pistas o trazas planas de cobre que permiten conectar eléctricamente los componentes que formen el circuito, tales como, condensadores, resistencias, bobinas, ICs (*Integrated Circuits*), etc. En resumidas cuentas, una PCB es un soporte físico sobre el que instalar e interconectar componentes electrónicos [61, 62].

Para soldar los componentes anteriormente mencionados a la placa se pueden seguir dos técnicas: THT (*Through Hole Technology*) o SMT (*Surface Mount Technology*) [62]:

- **THT**: La tecnología de agujero pasante consiste en taladrar agujeros en la PCB a través de los cuales se pasan las patillas de los componentes para posteriormente soldarlos. Aunque esta técnica presenta un menor coste ensamblaje, tiene el importante detrimento de reducir la densidad de componentes, ya que al atravesar las patillas de estos la placa por ambos extremos, no se pueden aprovechar ambas caras de la placa para colocar elementos ni láminas interiores para lanzar pistas. Esto acaba derivando en dificultades para miniaturizar la circuitería, con lo cual no es ideal para aplicaciones como los dispositivos vestibles.

- **SMT**: La tecnología de montaje superficial permite soldar un componente sobre una de las caras de la placa sin afectar a la cara contraria, esto permite utilizar ambos lados de la placa para colocar componentes. Para ello, en lugar de emplear agujeros, se usan vías sobre las que se sueldan los elementos. Esta técnica permite un uso más óptimo del espacio ya que no inhabilita el uso del resto de capas de la PCB, lo cual permite obtener dispositivos de menor tamaño. Los componentes que se sueldan siguiendo esta tecnología se conocen como SMD (*Surface Mount Device*).

Las PCBs se pueden clasificar en tres tipos según cuantas capas de material conductor (cobre) incluyan [62]:

- **SSB** (*Single-sided bond*): Sólo una de las caras de la PCB contiene cobre, la otra cara está formada por un material aislante y no puede albergar componentes.

- **DSB** (*Double-sided bond*): Ambas caras de la PCB pueden tener cobre y, por tanto, componentes. Existe una capa de material aislante entre las dos caras de la lámina. Para conectar eléctricamente elementos que se encuentre en caras contrarias de la placa se emplean vías, estas consisten en perforaciones que conectan ambos lados de la PCB mediante un camino con cobre.

- **MLB** (*Multi-layer bond*): Se utilizan más de dos capas con cobre, esto es, entre las capas superior e inferior pueden existir otras que también incluyan material conductor. Esta técnica ayuda a maximizar la densidad de los componentes, ya que se pueden usar las capas internas para que viajen pista de cobre que, en los casos anteriores, tendrían que ocupar área de la superficie de las caras externas que ahora se puede aprovechar para colocar componentes eléctricos.

En la Figura 2.3 se muestra la sección transversal de una PCB *multi-layer*, donde se pueden apreciar las vías usadas para interconectar las diferentes capas.

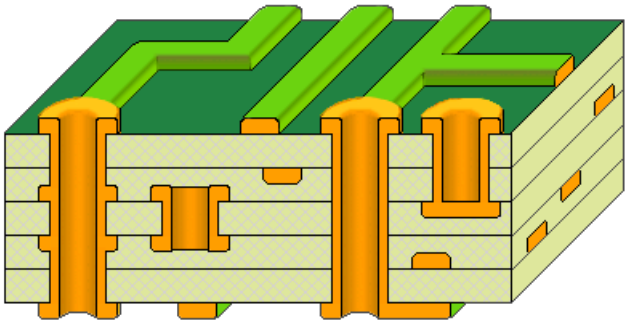


Figura 2.3: Sección transversal de las capas de una PCB [63].

2.3.1. Elección de PCB para este trabajo

Vistas las dos técnicas de ensamblaje existente, **en este trabajo se favorece el uso de la tecnología SMT** al ser el tamaño del dispositivo final una característica crítica, tal y como visto en el apartado Otros requerimientos de este capítulo. Por tanto, los componentes que se elijan serán SMD. Respecto a la cantidad de capas con material conductor que van a ser necesarias, una **PCB tipo DSB resulta suficiente** para condensar los componentes eléctricos en un área aceptable, sin tener que incurrir en el incremento de coste que supone una tipo *multi-layer*.

2.4. Selección de componentes

En este apartado se puede encontrar la información relativa a los modelos concretos de componentes seleccionados para construir el diseño planteado en los diagramas de bloques de las Figuras 2.1 y 2.2.

2.4.1. PSoC

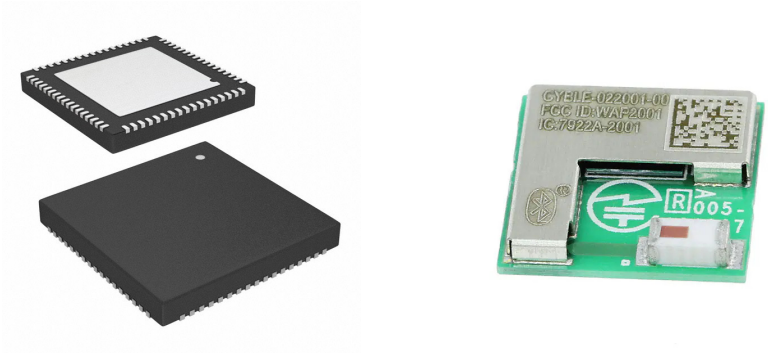
Un SoC (*System on Chip*) se puede definir como la técnica de aglutinar los módulos necesarios para un sistema electrónico en un único circuito integrado o chip. De esta forma CPUs, RAM, convertidores analógico-digitales (ADC) y digitales-analógicos (DAC), memorias EEPROM, amplificadores operacionales... se encuentran todos integrados dentro del mismo chip. El uso de un SoC añade adicionalmente la ventaja de que al poder incluir varios componentes, tales como amplificadores operaciones, dentro del propio empaquetado, se consigue minimizar el espacio ocupado por el circuito resultante debido a reducirse la necesidad de añadir componentes externos. Estos dispositivos suelen tener una mayor potencia de cálculo que los microcontroladores (MCU) tradicionales y por lo tanto son más adecuados para aplicaciones complejas. De hecho, en ocasiones un único SoC puede incluir en su interior varios microcontroladores [64].

Para el sistema desarrollado en este trabajo se ha optado por escoger soluciones basadas en PSoC (*Programable System on Chip*) desarrolladas por Cypress Semiconductor para el procesamiento y transmisión de las señales provenientes de los sensores. PSoC es la designación comercial usada por Cypress para referirse a la familia de SoCs que esta empresa desarrolla. Estos dispositivos cuentan con el atractivo de tratarse de un SoC cuyas entradas y salidas, así como componentes electrónicos tales como amplificadores, convertidores de corriente a voltge, ADCs, etc que pueda incluir, se pueden reconfigurar por *software*, aportando con ello una enorme flexibilidad a la cantidad de aplicaciones que pueden abarcar y minimizando el *hardware* externo a utilizar.

Como protocolo de comunicación inalámbrica para transmitir los datos se emplea Bluetooth de baja energía (BLE o *Bluetooth Low Energy*), también en ocasiones conocido como *Bluetooth Smart*, el cual está especialmente enfocado a su uso en dispositivos alimentados por pilas o baterías en aplicaciones de IoT (*Internet Of Things*). Se selecciona este protocolo debido a su excelente ratio de energía consumida por bit transmitido [65, 66]. Para la transmisión se precisa de que o bien el PSoC incluya de manera nativa comunicación inalámbrica, o que en su defecto, se añada otro

módulo adicional que si incluya esta capacidad, de manera que asista a modo de transmisor.

Para el tratamiento de las señales se usa el **PSoC 5LP CY8C5868LTI-LP039** [67], al no disponer este de comunicación inalámbrica por sí solo, se usa el módulo **CYBLE-022001-00** [68], también de Cypress, para poder utilizar comunicación Bluetooth de baja energía. En la Figura 2.4 se encuentran imágenes del empaquetado te estos dos integrados.



(a) PSoC 5LP CY8C5868LTI-LP039. [70]

(b) CYBLE-022001-00 [71]

Figura 2.4: Empaquetado de los PSoCs usados.

2.4.1.1. Módulo de procesamiento

Al emplearse el PSoC para el control del sistema y el procesamiento de las señales, se precisa de que este cuente con una CPU con capacidad de cálculo adecuada para implementar los algoritmos para el tratamiento de las señales, sea capaz de trabajar en tiempo real en caso de ser necesario, pueda hacer uso de manera efectiva de interrupciones, ofrezca flexibilidad en la asignación de pines de I/O e incluya suficientes módulos internos (DACs, ADCs, amplificadores, etc) como para minimizar el *hardware* que se tiene que añadir externamente. La familia PSoC 5LP CY8C58LP [67] cumple con estos requerimientos, habiéndose escogido en concreto el modelo **PSoC 5LP CY8C5868LTI-LP039**, cuyas especificaciones se pueden consultar en la Tabla 2.1. La diferencia con el resto de modelos de la misma familia es mínima, en concreto, esta radica en el número total de pines disponibles y cuantos se pueden asignar para cada funcionalidad (GPIO, UART, etc).

Un factor a considerar ha sido el hecho de que se pretende controlar cuatro puertos de entrada, los cuales se pueden tratar de canales analógicos o digitales según el kit de expansión usado en cada momento. Para ello, se hace necesario contar con una cantidad adecuada de pines de I/O disponibles, tanto analógicos como digitales. El PSoC presenta disponibles 38 pines GPIO que se pueden conectar directamente a los periféricos analógicos o digitales. El disponer de acceso directo a componentes analógicos, tales como amplificadores, pueden permitir, por ejemplo, amplificar una señal diferencial recibida como entada sin tener que muestrear previamente ambas componentes de la señal. Aunque el fabricante recomienda ciertos puertos GPIO como más adecuados según el periférico al que se desee acceder, en la práctica se puede elegir casi cualquier GPIO según resulte más conveniente.

También se incluyen otros periféricos analógicos que aunque puedan no utilizarse en ciertas aplicaciones finales, si que pueden resultar útiles durante las pruebas con prototipos. Por ejemplo, para comprobar el correcto funcionamiento y/o configuración de un ADC, se pueden utilizar un DAC interno para introducir señales en un entorno controlado.

Respecto a los protocolos de comunicación disponibles, nos resulta de interés principalmente el hecho de que se dispone de comunicación I^2C, utilizado habitualmente para la comunicación entre dispositivos, y de UART, ya que este último será el usado para la comunicación el módulo Bluetooth.

Para esta familias de PSoCs fabricadas por Cypress, el propio fabricante pone a disposición de los desarrolladores la herramienta PSoC Creator, la cual permite desarrollar *firmware* para el PSoC mediante programación en el lenguaje C y con la asistencia de herramientas gráficas que posibilitan configurar de manera sencilla los varios periféricos del sistema. Este *software* se tratará con mayor profundidad en apartados futuros. Una ventajas muy destacable de los PSoC es el acceso que da Cypress a extensa documentación y ejemplos de uso, lo que permite agilizar el desarrollo de proyectos.

Para reprogramar el dispositivo se puede hacer uso de las interfaces de programación JTAG (*Joint Test Access Group*) o SWD (*Serial Wire Debugging*). Como se tratará en más detalle en el Capítulo 3, se ha planteado utilizar JTAG para este fin.

2.4.1.2. Módulo de transmisión

El **CYBLE-022001-00** [68] es un modulo basado en el PSoC 4 Bluetooth LE el cual está enfocado a proporcionar comunicación Bluetooth Low Energy, en concreto Bluetooth 5.1, y con una velocidad de conexión de 1 MByte/s a los sistemas que así lo requieran. Para la transmisión y recepción emplea el modelo de antena en chip 2450AT18B100 [69] de Johanson Technology, la cual opera en el rango de frecuencia de 2.4 a 2.5 GHz. En la Tabla 2.2 se han incluido las características del módulo.

El módulo dispone de periféricos internos, tales como ADCs, TIMERs, contadores o PWM. Adicionalmente, también cuenta con comunicación I^2C, SPI y UART, usaremos este último protocolo para comunicar el módulo BLE con el PSoC 5LP CY8C5868LTI-LP039.

Al igual que el resto de PSoCs desarrollados por Infineon, el PSoC 4 BLE se puede programar haciendo uso de la herramienta oficial PSoC Creator. Para reprogramar este componente en concreto sólo se puede utilizar la interfaz SWD.

	PSoC 5LP CY8C5868LTI-LP039 [67]
CPU	32-bit Arm Cortex-M3
Velocidad CPU	Hasta 80 MHz
SRAM	64 kB
Flash	256 kB
EEPROM	2 kB
Comunicación inalámbrica	No
Comunicación UART	Si
Comunicación I^2C	Si
Comunicación SPI	Si
CapSense	Si (1)
Nº de GPIO	38
Nº de SIO	8
Nº de ADC	1 canal de 20 bits (Delta-Sigma) 2 canales de 12 bits (SAR)
Nº de DAC	4 canales de 8 bits
Nº de UDBS	24 canales (2)
Nº de USBIO	2
Nº de TIMERs	4 de 16 bits
Nº de Amplificadores Operacionales	4
Nº de Comparadores	4
Rango Temp Ambiente	-40 - 85 ºC
Rango Voltaje Alimentación	1.71 - 5.5 V
Consumo de corriente según modo de funcionamiento	Activo 1.9 - 25.5 mA Sleep 1.9 - 3.6 μA Hibernación 0.2 μA - 21 mA (3)
Empaquetado	68-QFN (68 pines)

Tabla 2.1: Especificaciones técnicas del PSoC 5LP CY8C5868LTI-LP039.

(1) CapSense consiste en un sistema integrado en el PSoC que permite medir capacitancias en aplicaciones tales como botones táctiles o detectores de proximidad. Se puede utilizar con cualquier pin GPIO.

(2) Los pines UDBs (*Universal Digital Blocks*) se pueden utilizar para comunicaciones (UART, SPI e I^2C), timers, contadores, verificación de redundancia cíclica (CRC), generadores de secuen-

cias pseudo aleatorias (PRS) y PWM (*Pulse Width Modulator*).

(3) Los rangos de consumo de corriente son orientativos. Estos se ven influenciados por la frecuencia de funcionamiento del procesador, la temperatura y, en los modos *sleep* e hibernación, también debido a que bloques del sistema se mantienen en uso. Por ejemplo, en el modo *sleep* se puede apagar la CPU pero mantener un comparador en funcionamiento. El rango de consumo en modo activo se proporciona asumiendo que únicamente la CPU se encuentra en uso, caso de usarse otros bloques del PSoC el consumo de corriente se puede incrementar.

	CYBLE-022001-00[68]
CPU	32-bits Arm Cortex-M0
Velocidad CPU	Hasta 48 MHz
SRAM	16 kB
Flash	128 kB
EEPROM	-
Comunicación inalámbirca	Bluetooth 5.1 (BLE)
Tipo de antena	Antena en chip (2.4 – 2.5 GHz)
Velocidad de conexión	1 MByte/s
Comunicación UART	Si
Comunicación I^2C	Si
Comunicación SPI	Si
Nº de GPIO	16
Nº de SIO	-
Nº de ADC	1
Nº de DAC	2
Nº de SCB	2 (1)
Nº de TCPWM	4 de 16 bits (2)
Rango Temp Ambiente	-40 - 85 ºC
Rango Voltaje Alimentación	1.9 - 5.5 V
Consumo de corriente según modo de funcionamiento	Activo 1.7 - 13.4 mA Deep-Sleep 1.5 μA Hibernación 150 nA
Empaquetado	21-pad SMT

Tabla 2.2: Especificaciones técnicas del módulo CYBLE-022001-00.

(1). Los SCBs (*Serial Communication Blocks*) hacen referencia a un bloque *hardware* que se puede configurar para ser utilizado en comunicación I^2C, SPI, UART, o EZI2C.

(2). Los TCPWMs son bloques que se pueden configurar para ser utilizados a modo de *timer*, contador o generador de PWM.

2.4.2. Regulador de tensión

Se va a alimentar desde una fuente de alimentación externa, por ejemplo una batería portátil, al PSoC 5LP CY8C5868LTI-LP039, al módulo CYBLE-022001-00, al resto de componentes del dispositivo central y a los sensores que se conecten a cada uno de los cuatro puertos de entrada, a este fin se usa un regulador de tensión. Estos dispositivos pueden recibir como entrada un cierto rango de nivel de tensión y en su lugar devuelven un valor de voltaje constante deseado, lo que permite estabilizar el voltaje del circuito. Algunos modelos sólo devuelven valores de tensión configurados en fábrica, mientras que otros se pueden ajustar para conseguir valores concretos. Se han barajado varios modelos de regulador de tensión de Texas Instruments, se puede encontrar una comparativa entre algunos de ellos en la Tabla 2.3.

A la hora de elegir un regulador para esta situación, el primer parámetro que se busca es que acepte un rango amplio de voltajes de entrada (V_{IN}), para de dicha forma dar más flexibilidad a como se puede alimentar el circuito. Adicionalmente, tiene que poder dar de corriente de salida (I_{OUT}) del orden de amperios; esto se hace con la intención de utilizar un único regulador para alimentar simultáneamente todos los componentes en lugar de tener que emplear varios reguladores de baja corriente, ya que de dicha forma se optimiza la cantidad de componentes usados y con ello el espacio ocupado. Finalmente, se debe de buscar un dispositivo que cumpliendo los anteriores requerimientos, tenga la menor corriente I_Q, también llamada corriente de reposo o *quiescent current*. Este parámetro consiste en la corriente que se sigue consumiendo aún cuando el integrado no tiene una carga conectada, pero el regulador si que se encuentra activado. Valores bajos de I_Q permiten prolongar la autonomía de dispositivos portátiles o vestibles [72].

De la comparativa entre los varios modelos se ha concluido que el dispositivo que mejor se adapta a los requerimientos buscados es el **LP8340** [73]. Este dispositivo tiene una versión que proporciona valores de tensión fijo y otra que permite una salida de tensión ajustable, dentro de ellos se ha seleccionado la variante que proporciona un **valor fijo de 3.3 V**. Cabe destacar que adicionalmente existe una variación del LP8340 según el rango de temperatura en el cual se puede operar. Ya que se planea que el dispositivo final no tenga que funcionar en situaciones térmicas extremas, se escoge la variante **LP8340C**, la cual puede operar en el rango 0-125 ºC.

El modelo seleccionado tiene un voltaje de *dropout* (V_{IN}-V_{OUT}) típico de 420 mV, 800 mV en el peor caso posible. Este parámetro viene a significar cual es la mínima diferencia que puede existir entre las tensiones de entrada y de salida de modo que el regulador aun consigue obtener el nivel de tensión de salida deseado. Es importante conocer este parámetro, ya que de manera efectiva acota por abajo el voltaje externo con el cual se puede alimentar el circuito, es decir, aunque el integrado se puede alimentar según el fabricante entre los 2.7 y los 10 V, debido a que queremos regular a 3.3 V, en el mejor caso posible habría que alimentar el sistema con como mínimo 3.72 V, 4.1 V en el peor caso posible [74]. Aun así, el rango de voltaje que se puede usar para la alimentación del regulador es suficiente como para proporcionar cierta flexibilidad al usuario final a la hora de alimentar el dispositivo.

Modelo	**LP8340** [73]	**TPS72733DSER** [75]	**TPS7A03** [76]	**TPS785-Q1** [77]	**TLV767-Q1** [78]
V_{IN} (V)	2.7 - 10	2 - 5.5	1.5 - 6	1.7 - 6	2.5 - 16
V_{OUT} (V)	3.3 (Fijo)	3.3 (Fijo)	3.3 (Fijo)	3.3 (Fijo)	3.3 (Fijo)
I_{OUT} Máx (A)	1	0.25	0.2	1	1
I_Q Típica (μA)	19	7.9	0.2	25	60
I_Q Máx (μA)	50	- (2)	-	-	95
V_{IN}-V_{OUT} Típico (mV) (1)	420	162.5	-	-	940
V_{IN}-V_{OUT} Máx (mV)	800	200	270	350	1500

Tabla 2.3: Comparativa de reguladores de tensión.

(1) Los voltajes de *dropout* tanto típicos como máximos que se proporcionan en la tabla 2.3 se dan asumiendo que el regulador está proporcionando la máxima corriente de salida (I_{OUT}) posible, o en su defecto, el mayor valor dado por el fabricante en la documentación, al tiempo que se selecciona un voltaje de salida de 3.3 V.

(2) Algunos parámetros no se han podido incluir en la tabla comparativa ya que no están proporcionados por el fabricante en la documentación oficial.

2.4.3. Load Switch

Se desea disponer de un método que permita, mediante *software*, mantener el control sobre que puertos se están alimentando desde el regulador de tensión. A este fin, se emplea un *load switch* o interruptor de carga, el cual consiste en un circuito integrado que permite desde uno de sus pines, llamado de *enable*, controlar abertura o cierre de una línea. Según si el nivel de tensión que haya en este pin se considera alta o baja, el switch permitirá el paso de corriente o por el contrario lo cortará. Normalmente, una tensión alta en el pin *enable* supone que el *switch* cierre el circuito, aunque hay modelos que operan de manera inversa. A pesar de que el funcionamiento de estos dispositivos pueda ser similar al del un relé, hay que tener en cuenta que no tienen piezas móviles, ya que se basan en transistores para abrir o cortar el paso de corriente. En resumen, el uso de este dispositivo permitirá mantener un mayor control sobre el consumo y los puertos de entrada a la PCB desde el PSoC, activando o desactivando la alimentación hacia ellos.

Cuando el *switch* cierre el circuito, esto es, permita el paso de corriente, en él se disipará un cierto valor de potencia. Esto se puede representar como que el *switch* se trata de una resistencia muy pequeña de valor R_{ON}, se busca por tanto que esta resistencia equivalente sea lo más pequeña posible para optimizar el consumo de potencia.

Al igual que ocurría durante la selección del regulador de tensión (Regulador de tensión), hay que tener en cuenta el valor de la corriente de reposo (I_Q). Adicionalmente, también resulta relevante la corriente de apagado o *shutdown current* (I_{SD}); esta determina la potencia que el *switch* sigue consumiendo aún cuando está apagado desde el pin de *enable* [79].

Dado que se ha elegido un regulador de tensión que puede proporcionar un máximo de 1 A de corriente (I_{OUT}), el *load switch* debe de al menos ser capaz de soportar este mismo nivel de intensidad. Además, dado que se proporciona un voltaje de 3.3 V, bastará con que el dispositivo sea compatible con este valor de tensión de entrada (V_{IN}). En la Tabla 2.4 se puede consultar una comparativa entre varios *load switches*. Se ha escogido el modelo **TCK106AG** debido a ser el que presenta la corrientes parásitas más pequeñas.

Modelo	**TCK106AG** [80]	**TPS22908** [81]	**TPS22934** [82]	**ADP198** [83]
V_{IN} (V)	1.1 - 5.5	1 - 3.6	1.5 - 3.6	1.65 - 6.5
I_{OUT} Máx (A) (1)	1	1	1	1
I_Q Típica (nA)	110	190	3500	2500
I_Q Máx (nA)	230	1000	20000	-
I_{SD} Típica (nA)	14	120	2500	1100
I_{SD} Máx (nA)	1000	1000	5000	2000
R_{ON} Típica (mΩ)	42.0	28.2	63.0	90.0
R_{ON} Máx (mΩ)	68	32.1	77	-

Tabla 2.4: Comparativa de reguladores de tensión.

(1) Dado que se pretende trabajar a un valor de tensión de 3.3 V, los valores de I_{OUT} y R_{ON} que aparecen en la taba 2.4, son los proporcionados por el fabricante para esta tensión. En caso de que no se indiquen los parámetros para este valor de voltaje, se usan los que se si que se proporcionen para el valor de tensión más cercano a 3.3 V.

2.4.4. Amplificador de instrumentación

Un amplificador diferencial es un dispositivo que realiza la operación de amplificar, por un cierto valor de ganancia G, la resta de dos señales de entrada (V_1 y V_2). Este comportamiento se puede describir mediante la siguiente ecuación [84]:

$$V_{Out} = G(V_2 - V_1) \tag{2.1}$$

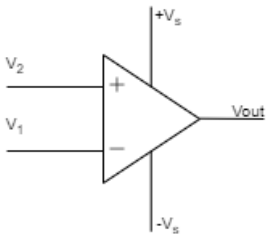


Figura 2.5: Modelo genérico de un amplificador de instrumentación.

Este dispositivo se emplea en este proyecto para la amplificación de señales diferenciales, tales como la señal del corazón obtenida durante un ECG. Una señal diferencial se caracteriza por desdoblarse en dos señales con polaridad opuesta, la señal que contiene la información se obtiene a partir de la resta de estas dos. Esto presenta la ventaja de poder eliminar el ruido en modo común, esto es, si asumimos que a las dos señales que viajan por conductores distintos se ven afectadas por exactamente la misma señal de ruido externo, al resta ambas el valor de ruido de las señales se elimina mutuamente. [85].

A la hora de elegir un amplificador de instrumentación, resulta ideal disponer de los valores más elevados posibles de CMRR (relación de rechazo al modo común), PSRR (relación de rechazo a la variación de la alimentación) e impedancia de entrada. Así mismo, es preferible tener la menor impedancia de salida posible.

Idealmente estos amplificadores devolverían un valor 0 a la salida en caso de estar en modo común, es decir, si se da que $V_1 = V_2$. En la práctica esto no ocurre; el CMRR es un parámetro que representa como de ideal es la respuesta del amplificador en el modo común, cuanto mayor sea dicho parámetros, más cercana a 0 es la salida en dicho modo. Otro parámetro estrechamente relacionado es el voltaje de corrección o de *offset* (V_{OS}), este se define como la tensión que hay que aplicar en los terminales de entrada de modo que la salida sea 0. Esto implica que si, por ejemplo, fuésemos a conectar ambas entradas del dispositivo a tierra, a la salida seguiría existiendo cierto nivel de tensión distinto de 0, este nivel de tensión es el voltaje de *offset*. Adicionalmente, debido a la degradación de los dispositivos electrónicos, este parámetro cambia con el tiempo [84, 86]. Variaciones en el voltaje de alimentación debido a una mala regulación alteran también la salida, el PSRR es un parámetro que se usa para especificar la habilidad del dispositivo para suprimir estas variaciones. Tanto el CMRR como el PSRR decaen a medida que aumenta la frecuencia y son también dependientes del nivel de ganancia G que se emplee.

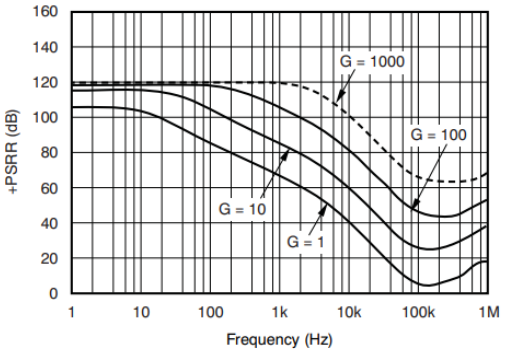
Respecto a las impedancias de entrada (Z_{In}) y de la salida (Z_{Out}), son respectivamente la impedancia que se ve desde los pines de entrada hacia el dispositivo y la impedancia que se ve desde la salida hacia el dispositivo. Debido a los componentes parásitos que albergan los dispositivos no ideales, ambos valores son dependientes también de la frecuencia [84].

Los amplificadores ven limitado el rango de voltaje que pueden proporcionar como salida, en caso de querer obtener un valor de tensión superior al límite, la salida se satura. Estos valores máximos para la amplificación de tensión positiva y negativa están fijados por el voltaje de alimentación ($+V_s$ y $-V_s$) y por las limitaciones físicas del propio dispositivo. Hay que seleccionar un dispositivo de manera que el nivel de amplificación deseado quede dentro del rango que permite el amplificador. En la práctica, el valor de la amplificación máxima es algo inferior a la tensión $+V_s$, y de igual manera ocurre con el mínimo valor posible y $-V_s$.

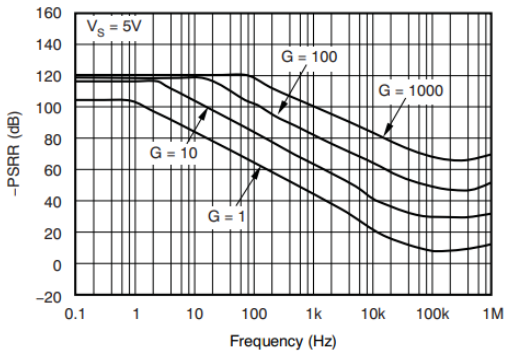
En la tabla 2.5 se pueden consultar la información de algunos amplificadores de instrumentación barajados. Finalmente, se ha escogido el **INA333** debido a que presenta un nivel de *offset* muy bajo y un buen valor de I_Q. Dado que las señales que se extraen del cuerpo humano tienen frecuencias bajas, para este trabajo el comportamiento del amplificador a altas frecuencias no resulta tan relevante. En la Figura 2.6 se pueden encontrar gráficas de la evolución del PSRR y el CMRR en función de la ganancia y de la frecuencia para este modelo de amplificador.

Modelo	**INA333** [87]	**INA317** [88]	**AD8235** [89]
Voltaje de alimentación (V)	1.8 - 5.5	1.8 - 5.5	1.8 - 5
Ganancia Máx	1000	1000	200
V_{OS} Típica (μV)	±10	±10	-
V_{OS} Máx (μV)	±25	±75	±2500
CMRR Típica (dB)	G=1 → 90 G=1000 → 115	G=1 → 90 G=1000 → 115	G=5 → 94 G=200 → 110
Z_{In} (GΩ || pF)	100 || 3	100 || 3	440 || 1.6 m.d 110 || 6.2 m.c
I_Q Típica (μA)	50	50	30
I_Q Máx (μA)	75	75	40

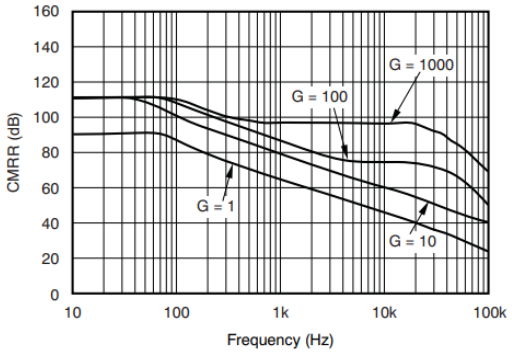
Tabla 2.5: Comparativa de amplificadores de instrumentación.



(a) PSRR (alimentación positiva).



(b) PSRR (alimentación negativa).



(c) CMRR.

Figura 2.6: Gráficas del CMRR y PSRR en dB en función de la frecuencia del INA333 [87].

2.4.5. Componentes pasivos

Por componentes pasivos aquellos que no suministran potencia al circuito, pero que en cambio si que pueden disiparla o almacenarla. En esta categoría entran componentes tales como resistencias, inductores o condensadores. En este trabajo se van a hacer uso de:

- Resistores
- Condensadores
- Núcleos de ferrita

En la 2.6 se pueden consultar los modelos usados y la cantidad necesaria de cada uno. En el Capítulo 3: Diseño hardware del dispositivo de procesamiento central, en el cual se trata el propio diseño *hardware* con más profundidad, se puede encontrar la información detalla del motivo de la elección de valores para estos componentes, la cantidad necesaria y cual es el uso de cada uno de ellos.

2.4.5.1. Núcleo de ferrita

Un núcleo de ferrita, o filtro de ferrita, es un componente electrónico pasivo que se utiliza para suprimir ruido electromagnético (EMI) de alta frecuencia, es decir, actúa a modo de filtro paso baja.

De manera ideal estos componentes no afectan a señales de baja de frecuencia, en la realidad debido a que tendrán un cierto valor de resistencia parásita, si que causan una cierta pérdida de potencia. En la práctica, se pueden tratar de manera similar a una bobina, aunque en este caso el parámetro más relevante es la impedancia que desarrollan a una cierta frecuencia, en lugar de su valor de inductancia. Por tanto, a la hora de seleccionar un modelo hay que considerar que se tenga la menor impedancia posible a la frecuencia de trabajo y la mayor posible a la frecuencia que se busca filtrar, dicho de otra forma, que filtrando los componentes frecuenciales deseados, cause la menor disipación de potencia posible [90].

2.4.5.2. Condensador de bypass

Un concepto muy relevante es el del condensador de *bypass*. Tal y como se tratará en la parte respectiva a cada IC en el Capítulo 3, resulta habitual añadir estos condensadores de manera externa al empaquetado de algunos integrados y se emplean con doble finalidad [91, 92]. Por un lado, dado que los condensadores se "oponen" a cambios bruscos de voltaje, el primero de sus usos es suprimir fluctuaciones de la fuente de alimentación debido a cambios repentinos de corriente. De esta forma, si se produce una reducción en el nivel de tensión, el condensador se descarga para compensar dicho desnivel y si se produce un incremento, el condensador se carga.

La segunda funcionalidad de estos componentes es eliminar parte del ruido de alta frecuencia proveniente de la alimentación. Esto ocurre debido a que en alta frecuencia, un condensador se ve como una impedancia de pequeño valor, lo que ayuda a atenuar la señal de ruido que llega a la carga ya que la mayoría se disipa en el condensador. En la ecuación 2.2 se muestra el cálculo de la impedancia en un condensador ideal según la frecuencia. Hay que tener en cuenta que en la realidad si se supera la frecuencia de resonancia del condensador, la impedancia comienza a incrementarse en lugar de disminuir debido a que el condensador pasa a tener comportamiento inductivo en lugar de capacitivo. Por tanto, es importante tener en consideración que banda de frecuencia se pretende filtrar y las propias características del condensador. Además, también hay que considerar que la impedancia no cambia, en la práctica, de manera perfectamente lineal.

$$Z_C = \frac{1}{j2\pi fC} \tag{2.2}$$

En la Figura 2.7 se muestra un circuito que ejemplifica el funcionamiento de un condensador de *bypass*.

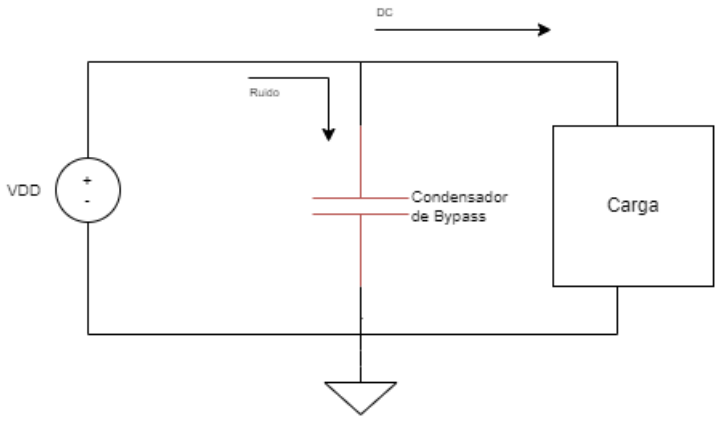


Figura 2.7: Ilustración del funcionamiento de un condensador de *bypass*.

2.4.6. Puertos de I/O

Se necesita elegir el conector que se va a utilizar como puerto de entrada para conectar la placa central y los sensores externos, así como puertos que permiten acceder a los pines de programación del PSoC 5LP CY8C5868LTI-LP039 y del CYBLE-022001-00 con la intención de poder reprogramar el *firmware* de ambos.

- **Puerto de programación para PSoC 5LP CY8C5868LTI-LP039**: El PSoC 5LP se puede programar mediante interfaz JTAG o SWD [67]. Se plantea usar la interfaz JTAG, para lo cual cuenta con 7 pines a los que hay que tener acceso, por este motivo se ha escogido una tira de pines macho tipo SMD con 2 filas de 5 conectores cada una. El modelo concreto escogido es el **M50-3600542** [94]. Aunque se tengan conectores sobrantes, se consigue aprovechar mejor el espacio disponible en la PCB de esta forma que si se usase una tira de 7 pines distribuidos en una única fila. Además, el empleo de este tipo de tira de pines facilita también uso del MiniProg3 [95], el cual es el dispositivo que se va a emplear para reprogramar ambos PSoC.

- **Puerto de programación para CYBLE-022001-00**: Este módulo se puede programar únicamente mediante interfaz SWD, con lo cual se tienen 5 pines de programación [68]. Bastará con emplear para acceder a los pines de programación una fila de 5 pines macho SMD. El modelo es el **M52-040000P0545** [96].

- **Conectores para sensores y kits de expansión**: Para la conexión con los sensores o kits de expansión se ha elegido el conector **84953-4** [93]. Cuenta con 4 conexiones, de las cuales 2 se emplean para alimentación (Vin y GND) y los otros dos se puede utilizar para recepción de información, nótese que según el sensor o kit de expansión conectado puede ser que sean necesarios ambas conexiones de comunicación o sólo una de los dos. En total se emplean 4 de estos conectores.

  Estas entradas están concebidas para ser usadas con cables planos flexibles o FFC (*Flexible Flat Cable*). En la Figura 2.8, se puede consultar un ejemplo de este tipo de cables. Los usados para este trabajo son de un ancho de cuatro pistas, de manera que coincidan con el tamaño de los conectores.

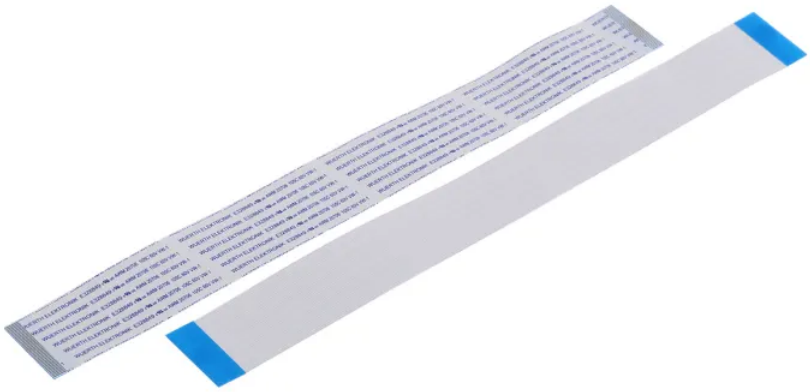


Figura 2.8: Ejemplo de un cable plano flexible [97].

2.5. Listado de componentes

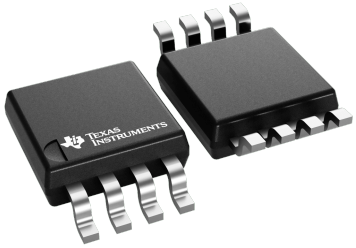
A modo de resumen, se puede consultar la Tabla 2.6, la cual incluye los componentes utilizados, los modelos escogidos y la cantidad necesaria. Respecto a los condensadores, resistencia y núcleos de ferrita, la información detallada a su respecto se puede encontrar en el Capítulo 3. En las Figuras 2.9, 2.10 y 2.11 se pueden hallar imágenes orientativas de los distintos componentes.

Componente	Modelo	Fabricante	Cantidad
Procesador	PSoC 5LP CY8C5868LTI-LP039	Cypress Semiconductor	1
Transmisor BLE	CYBLE-022001-00	Cypress Semiconductor	1
Regulador de tensión	LP8340C (3.3 V)	Texas Instruments	1
Load switch	TCK106AG	Toshiba	2
Amplificador de instrumentación	INA333	Texas Instruments	1
Conectores para sensores/kits	84953-4	TE Connectivity	4
Puerto programación PSoC 5LP	M50-3600542	Harwin	1
Puerto programación módulo BLE	M52-040000P0545	Harwin	1
Condensadores	C0402C104J4RACTU	KEMET	9
	CC0402KRX5R6BB105	YAGEO	4
Resistencias	ERA2AED102X	Panasonic	2
	ERA-2AEB104X	Panasonic	1
Núcleo de ferrita	MMZ1005Y301CT000	TDK	2

Tabla 2.6: Resumen componentes necesarios para dispositivo de procesamiento central.

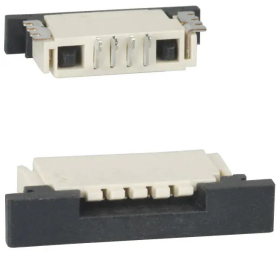
(a) LP8340C (3.3 V) [98].

(b) TCK106AG [99].



(c) INA333 [100].

Figura 2.9: ICs.



(a) 84953-4 [101].

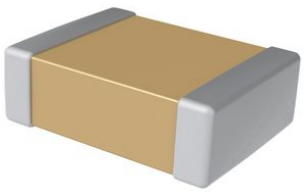
(b) M52-040000P0545 [102].

(c) M50-3600542 [103].

Figura 2.10: Puertos de I/O y pines macho para reprogramación de los PSoCs.

(a) MMZ1005Y301CT000 [104].

(b) ERA2AED102X y ERA-2AEB104X [105, 106].



(c) C0402C104J4RACTU y CC0402KRX5R6BB105 [107].

Figura 2.11: Componentes pasivos.

Capítulo 3

Diseño hardware del dispositivo de procesamiento central

Habiéndose establecido en el capítulo anterior que objetivos se pretenden lograr con el dispositivo de procesamiento central y para ello que componentes electrónicos son necesarios, en este capítulo se pasa a describir el diseño físico del dispositivo sobre la PCB. Como ya se mencionó en el Capítulo 1: Introducción, el diseño *hardware* se ha realizado haciendo uso del *software* EDA: Altium Designer. El capítulo se encuentra organizado de la siguiente manera:

- Los componentes individuales diseñados en Altium se pueden encontrar en la sección Símbolos y footprints de los IC y los componentes pasivos.
- La configuración, y por configuración nos referimos a, por ejemplo, añadir elementos externos pasivos, así como la conexión de los conectores de cada componente individual, se ha tratado en el apartado Configuración, conexión y alimentación de los ICs.
- Respecto al diseño de PCB al que finalmente se ha llegado, se puede localizar en Diseño y análisis de la PCB, donde también se justifican algunas decisiones de diseño tomadas.

3.1. Símbolos y footprints de los IC y los componentes pasivos

En este apartado se pueden consultar los símbolos usados para cada componentes en el esquemático y los *footprints* necesarios par el correcto ensamblaje de los elementos a la PCB. Todos los símbolos se pueden consultar en las Tablas 3.1 y 3.2, por otro lado, en las Tablas 3.3 y 3.4 se encuentran los respectivos *footprints*. Nótese que en el caso del PSoC 5LP CY8C5868LTI-LP039 existen dos entrada en la Tabla 3.1 debido a que su símbolo se divide también en dos en el esquemático. A pesar de esto, Altium reconoce que estos dos símbolos forma parte del mismo componente y están únicamente desdoblados en el esquemático por comodidad hacia el desarrollador, con lo cual existe un único *footprint* asociado a ellos.

Respecto a los condensadores y resistores, como es lógico, los mismos tipos de componentes usan el mismo símbolo y solo se modifica el valor asociado. Para el núcleo de ferrita, se ha usado el símbolo de un inductor. Dado que los condensadores, resistencias y núcleo de ferrita usan el mismo empaquetado, el 0402, el *footprint* es también el mismo para todos estos pasivos.

3.1.1. Símbolos

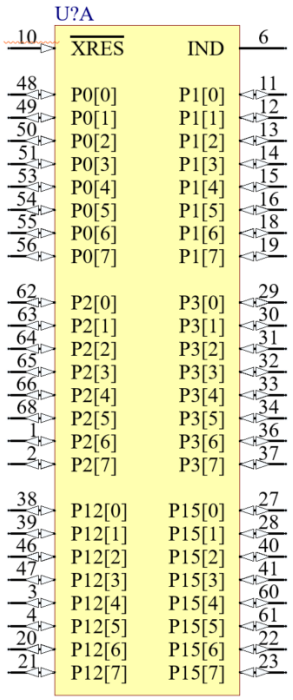
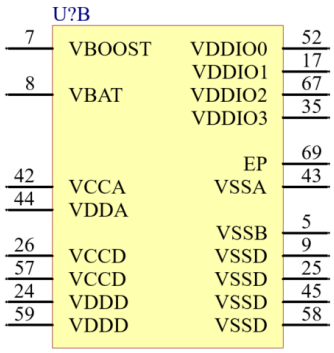
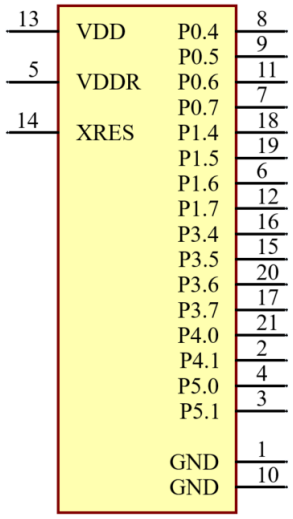
Componente	Símbolo
PSoC 5LP CY8C5868LTI-LP039 (Parte A)	
PSoC 5LP CY8C5868LTI-LP039 (Parte B)	
CYBLE-022001-00	

Tabla 3.1: Símbolos de los ICs y elementos pasivos usados en la PCB del dispositivo de procesamiento central - Parte 1 de 2.

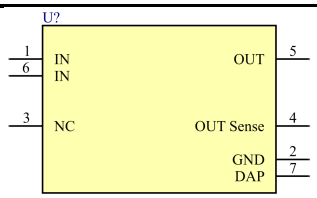
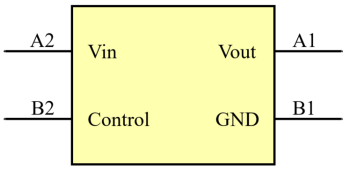
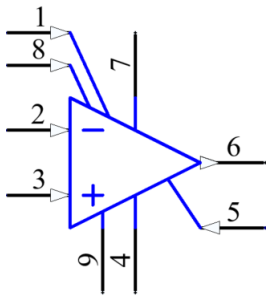
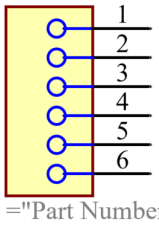
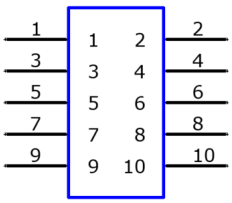
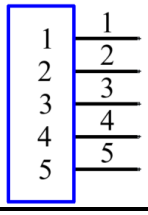
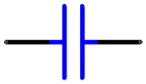
Componente	Símbolo
LP8340C	
TCK106AG	
INA333	
84953-4	
M50-3600542	
M52-040000P0545	
C0402C104J4RACTU CC0402KRX5R6BB105	
ERA2AED102X 2AEB104X	
MMZ1005Y301CT000	

Tabla 3.2: Símbolos de los ICs y elementos pasivos usados en la PCB del dispositivo de procesamiento central - Parte 2 de 2.

3.1.2. Footprints

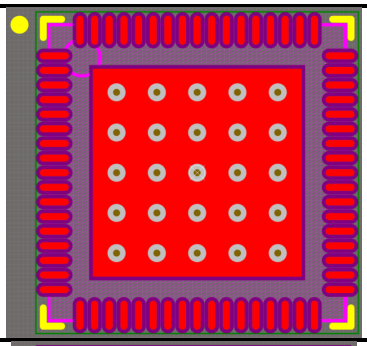
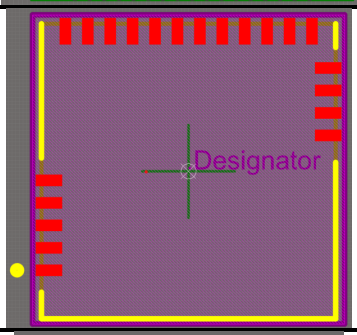
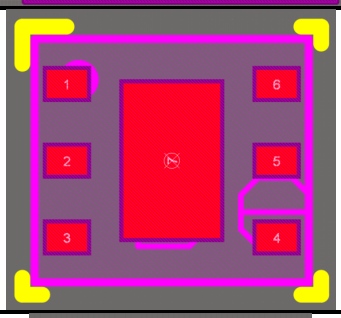
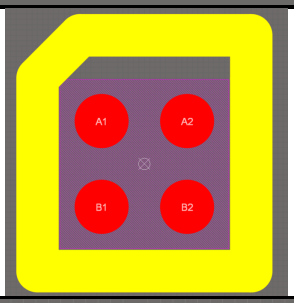
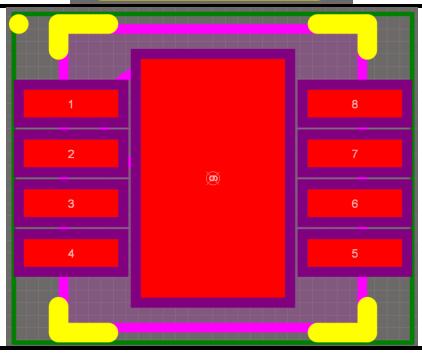
Componente	Símbolo
PSoC 5LP CY8C5868LTI-LP039	
CYBLE-022001-00	
LP8340C	
TCK106AG	
INA333	

Tabla 3.3: *Footprints* de los ICs y elementos pasivos usados en la PCB del dispositivo de procesamiento central - Parte 1 de 2.

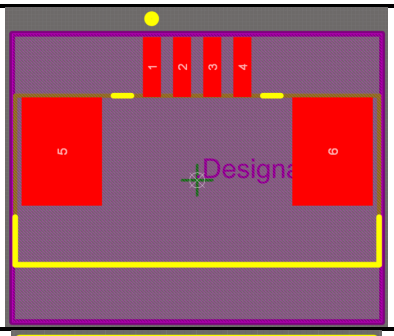
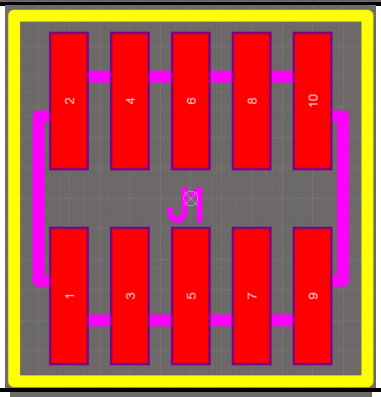
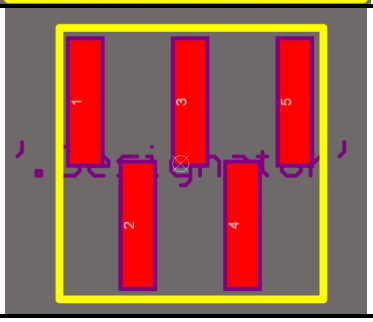
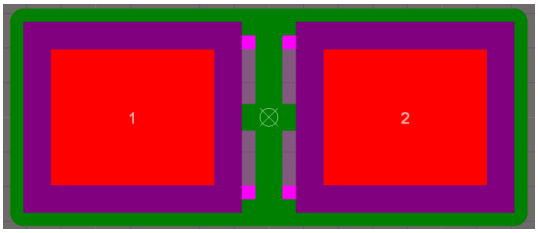
Componente	Símbolo
84953-4	
M50-3600542	
M52-040000P0545	
C0402C104J4RACTU CC0402KRX5R6BB105 ERA2AED102X ERA-2AEB104X MMZ1005Y301CT000	

Tabla 3.4: *Footprints* de los ICs y elementos pasivos usados en la PCB del dispositivo de procesamiento central - Parte 2 de 2.

3.2. Configuración, conexión y alimentación de los ICs

Es habitual que para el correcto funcionamiento de un circuito integrado (IC) sea necesario añadir componentes pasivos externos, un caso muy habitual consiste en añadir condensadores de *bypass*. En esta sección se tratan los elementos pasivos requeridos por el fabricante que hay que añadir de manera externa para el correcto funcionamiento de los integrados seleccionados en el Capítulo 2: Especificaciones del dispositivo de procesamiento central, así como los nodos a los que se encuentran conectados las patillas de cada uno de los componentes usados.

En la página 49, se puede consultar el esquemático completo diseñado en Altium Designer. En los subapartados siguientes se realiza una disección de dicho esquemático.

3.2.1. PSoC 5LP CY8C5868LTI-LP039

Como se exponía en la Tabla 2.1, el PSoC 5LP se puede alimentar con un rango de voltaje que va de los 1.71 a los 5.5 V. Por otra parte, el PSoC 5LP no tiene por qué tener un mismo valor de alimentación para: el núcleo o *core* y los componentes digitales, los analógicos y los pines de entrada/salida. A continuación se tratan las tensiones de alimentación con mayor detalle [67]:

- En la documentación se utiliza el distintivo VDDD para identificar al voltaje que alimenta a todos los periféricos digitales y al *core*. Esta tensión se debe de encontrar en el rango de alimentación mencionado anteriormente.
- Los pines de entrada/salida (I/O) se dividen en grupos que pueden ser alimentados a distintos voltajes, de manera que un único PSoC puede suministrar distintos niveles de tensión a componentes externos. Los voltajes a los que se alimenta cada grupo de pines I/O se identifican como VDDIO0, VDDIO1, VDDIO2 y VDDIO3. Su nivel mínimo y máximo de tensión es también de 1.71 y 5.5 V.
- El voltaje de alimentación de los periféricos analógicos puede ser diferente del usado para los periféricos digitales, el *core* y los pines I/O y se identifica como VDDA. Se tiene que cumplir que VDDA sea mayor o igual que las tensiones de VDDD y VDDIOX.
- Otros valores de tensión son VCCD, que viene a ser el voltaje de entrada/salida del regulador digital y VCCA que es el del analógico. Como consideraciones adicionales a la hora de diseñar la PCB, el fabricante recomienda que la pista que conecte los dos pines de VCCD sea lo más corta posible.

Dado que usa un único regulador de tensión de 3.3 V para alimentar todos los dispositivos presentes en la PCB, todos los niveles de tensión vistos se encontrarán a este valor. Para suministrar de manera correcta la tensión al PSoC, se tienen que añadir varios condensadores de *bypass* como se muestra en la Figura 3.2 extraída de la documentación del PSoC 5LP oficial proporcionada por Cypress.

En la Figura 3.3 se encuentra la configuración de la alimentación hecha en Altium y creada a partir de la información proporcionada en la Figura 3.2. Dado que como ya se ha dicho, todos los niveles de tensión para alimentar los componentes del PSoC están al mismo valor y provienen de la salida del mismo regulador de tensión, se han conectado todos los condensadores provenientes de VDDD requeridos en la Figura 3.2 en paralelo. Aunque todos los condensadores están conectados a VDDD, se han usado el componente de Alitum marcado como J2 para que sea más sencillo identificar las distintas entradas aunque en realidad sean el mismo nodo. Exactamente de igual manera que se hace con las entradas, se hace con las tierras usando J1.

En la Figura 3.3 se muestran únicamente los pines del PSoC 5LP referidos a la alimentación, la conexión de los pines respectivos a los puertos de entrada y salida se puede encontrar en la Figura 3.4. Se procede a identificar cada uno de los nodos o redes conectados a los pines del PSoC:

- OutX_Y se refieren a los puertos de entrada desde los que se reciben información de los sensores, donde "X" identifica al puerto e "Y" a cada uno de los pines dentro de dicho puerto. Basándonos en la Figura 3.1, se ha tratado de hacer las asignaciones de los pines GPIO de entrada de manera que las pistas que se lancen hacia los puertos de entrada a la PCB sean lo más cortas posibles. En la subsección Asignación de pines se puede encontrar más información al respecto de las asignaciones hechas.
- EN1 y EN2 se usan para controlar el estado ON/OFF de los *switches* TCK106AG. EN1 permite controlar la alimentación hacia los puertos 3 y 4 y el amplificador diferencial y EN2 hacia los puertos 1 y 2. Identificamos el *switch* controlado por EN1 como el *swtich* 1 y el controlado por EN2 como el *switch* 2.
- /XRES, TMS, TCK, TDO y TDI son los pines usados por la interfaz JTAG para reprogramar el *firmware*. Aunque aquí sólo se mencionan 5 pines, en el apartado Puertos de I/O se comentaba que se necesitan 7 para la programación mediante JTAG. Esto se debe a que los dos pines que faltan son de alimentación..
- UART_TX y UART_RX comunican el PSoC 5LP con el módulo BLE CYBLE-022001-00 mediante protocolo UART.
- INA_p e INA_n son las entradas del amplificador de instrumentación. Respectivamente, la primera se refiere a la entrada positiva y la segunda a la negativa. Por otro lado, INA_out recibe la salida obtenida desde dicho amplificador.
- CMM se usa para poder medir el modo común del amplificador diferencial.
- TP1 y TP2 están conectados a *pads* que no se encuentran soldados a ningún componente, es decir, en principio estos pines se dejan en abierto. Se han añadido para poder ser utilizados en pruebas que permitan comprobar el correcto estado del PSoC o de la PCB en caso de que sea necesario.

3.2.1.1. Asignación de pines

Para la elección de los pines a los que se deben de conectar las entradas, se ha utilizado el esquemático proporcionado en la Figura 3.1, el cual muestra la asignación de pines para los modelos de la familia PSoC 5LP que usan el empaquetado QFN de 68 pines.

Aunque en principio se puede usar cualquier puerto de pines GPIO para acceder tanto a la parte digital como analógica del PSoC, si que es cierto que el fabricante recomienda utilizar ciertos puertos según la funcionalidad prevista con la intención de optimizar el rendimiento. Por ejemplo, en la familia PSoC 5LP se recomienda el uso de los sets de pines P0[7:0], P3[7:0] y P4[7:0] para aplicaciones analógicas [108].

Se ha planteado disponer del protocolo de comunicación I^2C en caso de que sea necesario para la transmisión hacia o la recepción desde los kits de expansión. Al igual que ocurría en el caso anterior, se puede utilizar cualquier par de GPIOs para comunicación mediante este protocolo, sin embargo, hay pines que resultan más adecuados. En concreto, se recomienda el uso de las parejas P12[4] y P12[5] o P12[0] y P12[1]. El motivo de preferirse usar uno de estos dos dúos, es el hecho de que permiten enviar una señal de "despertar" a una dirección en la que se encuentre un dispositivo operando modo de bajo consumo o *sleep*, acción que no se puede hacer desde GPIOs diferentes a los mencionados [67]. El disponer de la posibilidad de usar bajo consumo resulta ventajoso desde el punto de vista de la optimización del consumo.

En la Tabla 3.5 se pueden consultar las lista completa de la asignación de pines. Esta lista se puede consultar a modo de recomendación a la hora de desarrollar *firmware* y conectar dispositivos externos al dispositivo central. La columna "Uso recomendado" indica el uso que el fabricante aconseja para cada pin, como se puede leer en la tabla, no se ha seleccionado ninguno de los pines recomendados para uso analógico. El motivo viene a ser que la localización de dichos pines no resultaba adecuada para el enrutado de las pistas en la PCB. Esto no implica aún así que los módulos analógicos del PSoC 5LP no se puedan usar de manera efectiva.

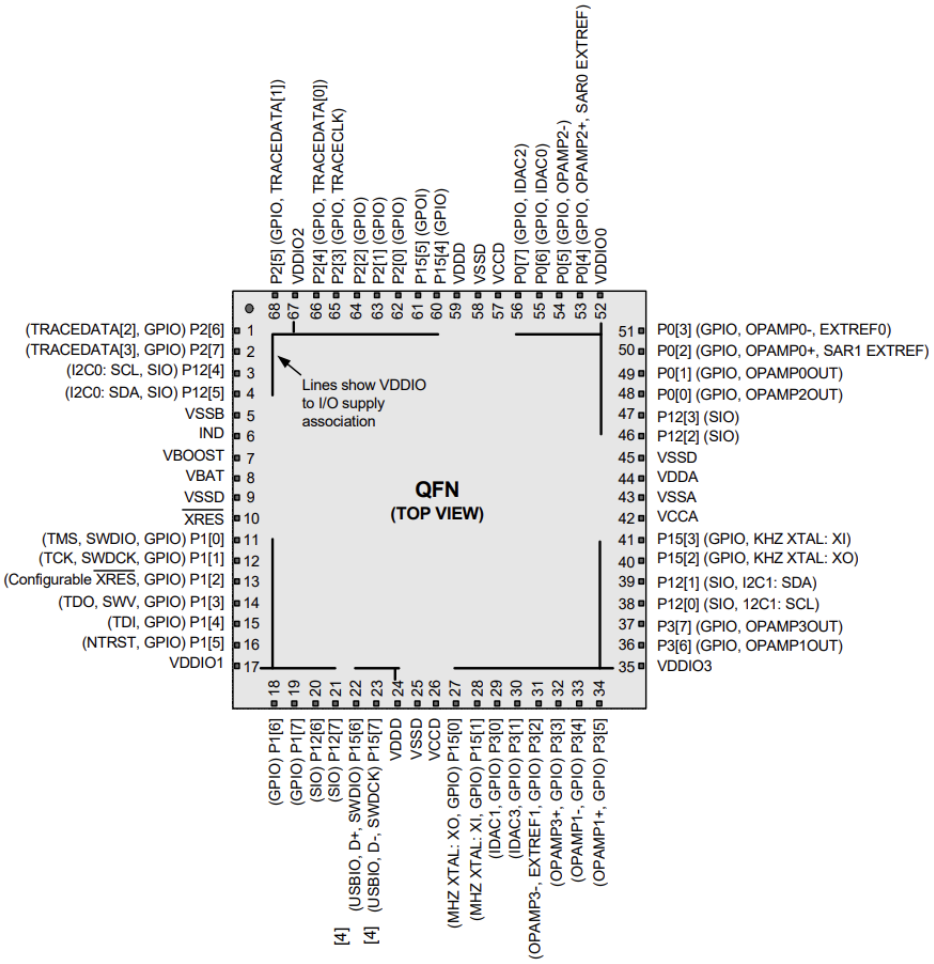


Figura 3.1: Mapa de pines del PSoC 5LP CY8C5868LTI-LP039 usando el empaquetado QFN-68 [67].

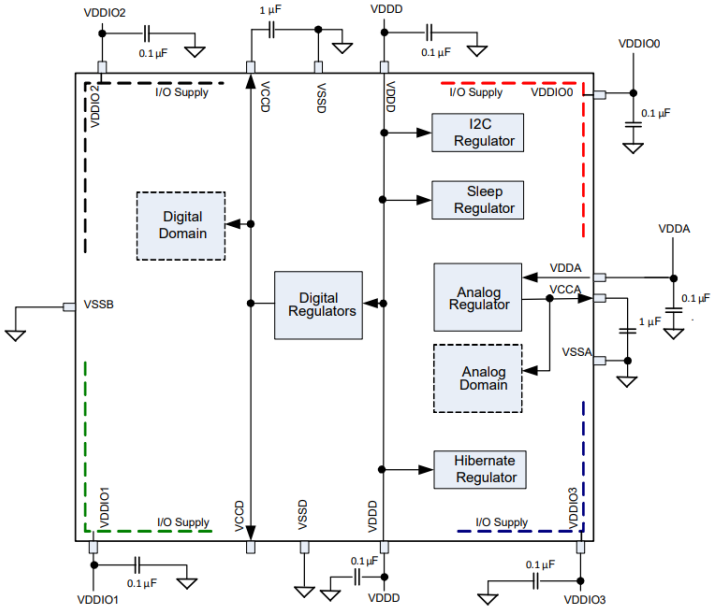


Figura 3.2: Esquema de la configuración de la alimentación del PSoC 5LP CY8C5868LTI-LP039 [67].

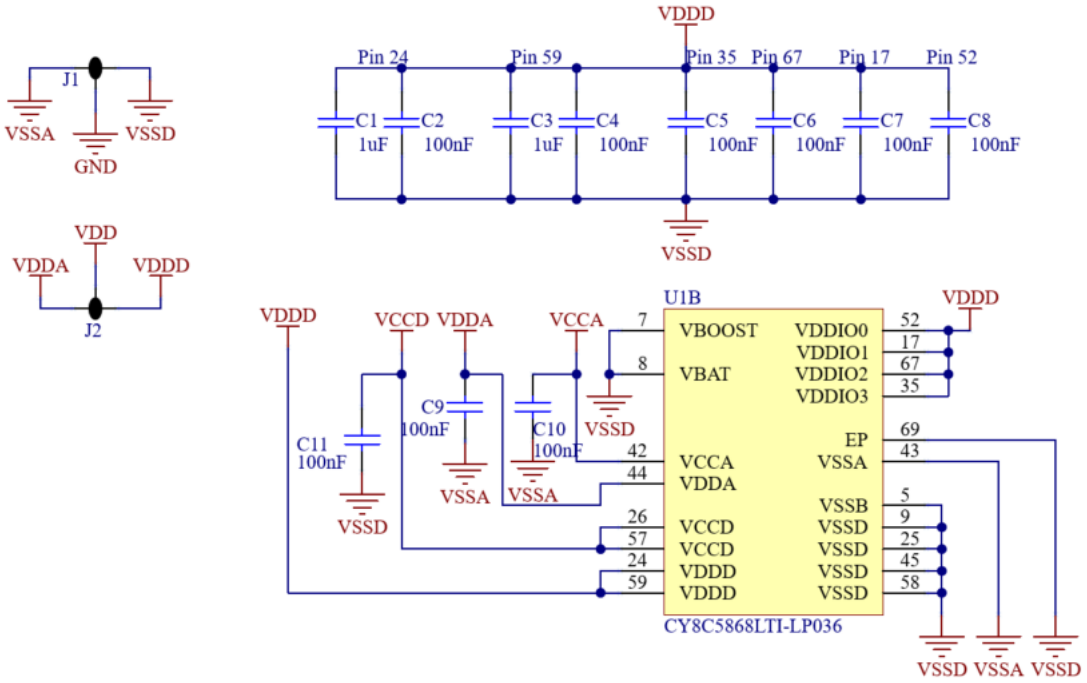


Figura 3.3: Esquema de la configuración de la alimentación del PSoC 5LP CY8C5868LTI-LP039 implementado en Altium.

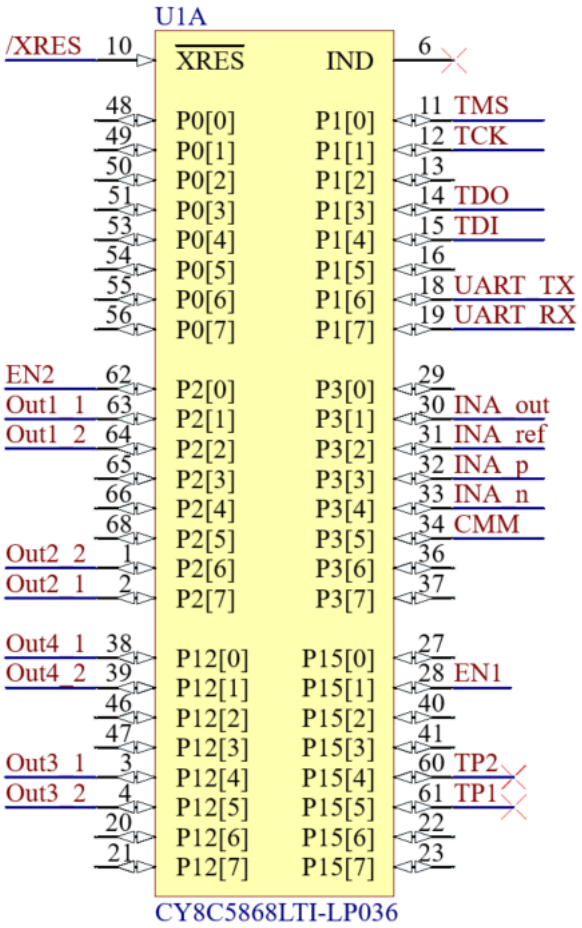


Figura 3.4: Esquema de la conexión de los pines de entrada y salida del PSoC 5LP CY8C5868LTI-LP039 implementado en Altium.

	Pines	Uso recomendado
Puerto 1	P2[1] (63)	Digital
	P2[2] (64)	
Puerto 2	P2[6] (1)	Digital
	P2[7] (2)	
Puerto 3	P12[4] (3)	Digital e I^2C
	P12[5] (4)	
Puerto 4	P12[0] (38)	Digital e I^2C
	P12[1] (39)	

Tabla 3.5: Listado de la asignación de pines hecha en el PSoC 5LP CY8C5868LTI-LP039. y el uso recomendado para cada puerto.

	Pin	Puertos controlados
Switch 1 (ON/OFF)	P15[1] (28)	3 y 4
Switch 2 (ON/OFF)	P2[0] (62)	1 y 2

Tabla 3.6: Listado de la asignación de pines hecha en el PSoC 5LP CY8C5868LTI-LP039 para el control de la alimentación de los puertos mediante los *switches* digitales.

3.2.2. CYBLE-022001-00

El módulo BLE [68] dispone de dos entradas de alimentación. Por un lado se tiene VDD, quién suministra la tensión para la alimentación de los periféricos analógicos y digitales. Por el otro, se encuentra VDDR, que suministra a la comunicación radio. Para asegurar el correcto funcionamiento del módulo, se debe de cumplir que la alimentación de VDD sea mayor o igual que la de VDDR y el rizado de la alimentación no sea de más de 100 mV. Dado que como ya se ha visto, se recurre a un único regulador de tensión de 3.3 V, ambos valores son iguales. La supresión del rizado se consigue desde el propio regulador y de los condensadores de *bypass* a la entrada y salida de este.

Para la alimentación, se hace necesario añadir también varios componentes pasivos de manera externa. En primer lugar, se deberían de añadir varios condensadores de *bypass*, sin embargo, dado que estos capacitores son los mismos que ya se emplearon para el PSoC 5LP CY8C5868LTI-LP039, no será necesario volver a añadirlos, ya que los nodos VDD y VDDR son el mismo punto que los nodos VDDD, VDDA y VDDIOX en el caso del PSoC 5LP. En la Figura 3.3 se puede consultar la colocación de estos componentes pasivos en el esquemático.

Adicionalmente, se añaden también dos núcleos de ferrita a la entrada de los pines de alimentación, tal y como mostrado en la Figura 3.3, donde aparecen identificados como L1 y L2. El fabricante recomienda colocar los núcleos de ferrita lo más cercano posible a los pines, además proporciona como sugerencia que este elemento tenga un valor de 330 Ω a una frecuencia de 100 MHz. En el momento de diseñar la PCB, se seleccionó el modelo MMZ1005Y301CT000 [109], el cual proporciona 300 Ω a 100 MHz, debido a la disponibilidad del *stock* de los distribuidores durante el diseño de la PCB.

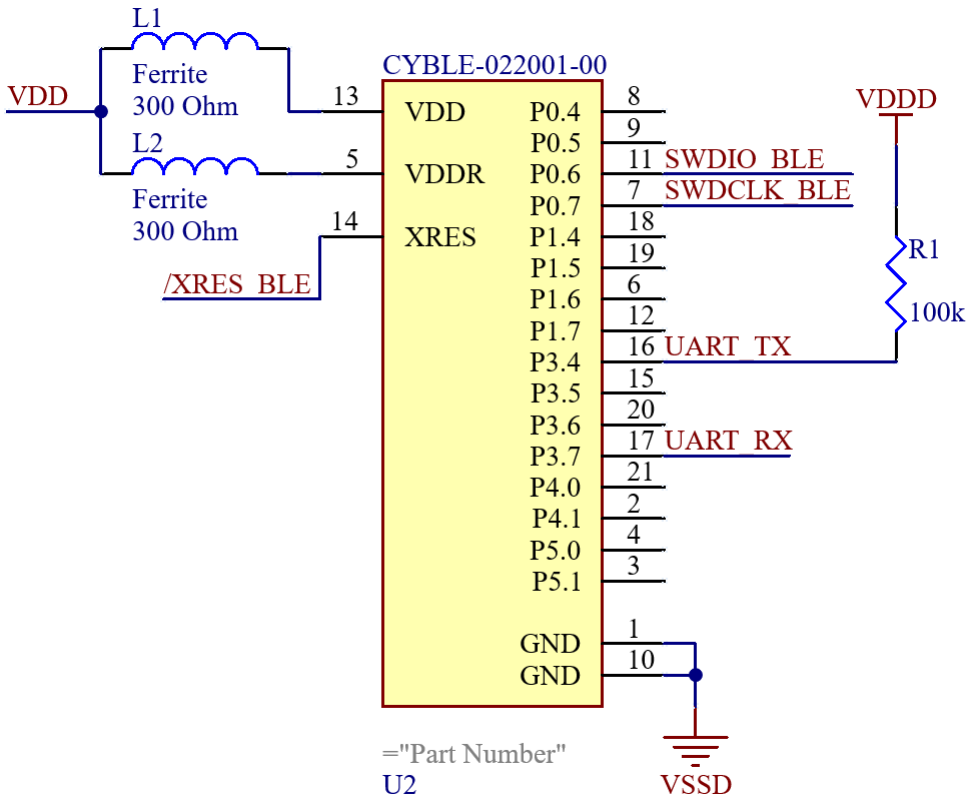


Figura 3.5: Esquema de la conexión de los pines de entrada y salida del CYBLE-022001-00 implementado en Altium.

3.2.2.1. Consideraciones respecto de la antena

Para maximizar el desempeño en radio frecuencia, el fabricante recomienda situar la antena del CYBLE-022001-00 en uno de los extremos de la PCB y que, de manera adicional, el área inmediatamente por debajo de la antena quede despejado de pistas que conecten con tierra o que transporten señales, ya que en caso contrario se limita la propagación de la señal [68, 110]. En la Figura 3.6 se muestran posibles posicionamientos del módulo en una PCB según el fabricante.

En la PCB no ha sido posible cumplir la segunda condición, ya que se ha preferido condensar los componentes en el menor área posible. Se ha tomado esta decisión, ya que se espera que el dispositivo a el cual se va a transmitir la información de manera inalámbrica se encuentre en las inmediaciones del circuito, con lo cual se ha considerado más importante que el dispositivo final sea más pequeño en lugar de que pueda transmitir a mayor distancia.

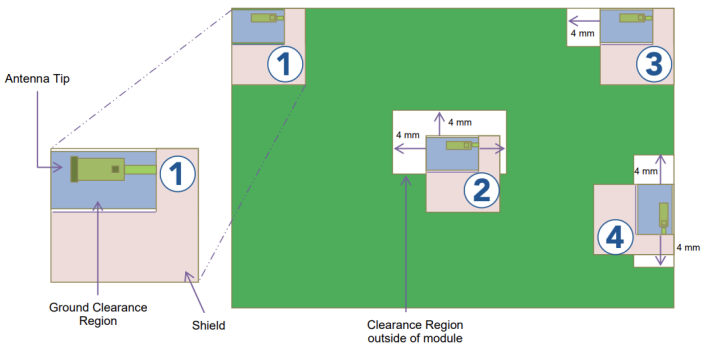
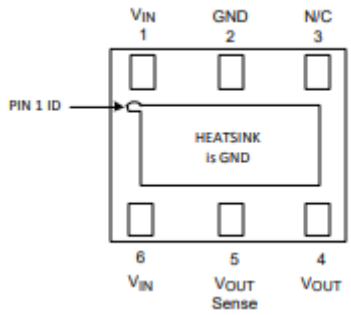


Figura 3.6: Recomendaciones para la colocación del módulo CYBLE-022001-00 en una PCB según el fabricante [110].

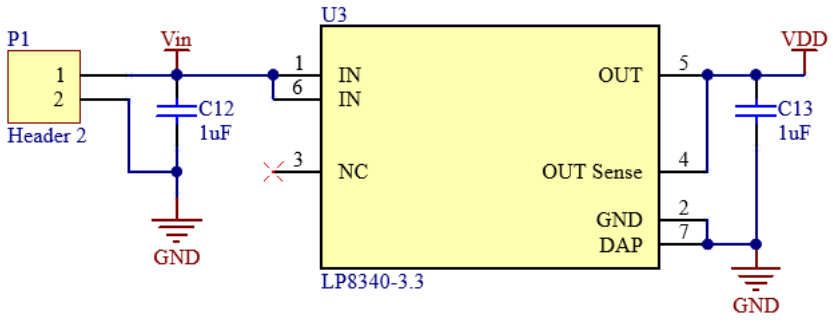
3.2.3. LP8340C

Para el adecuado funcionamiento del regulador de tensión, el fabricante aconseja utilizar dos condensadores de *bypass* que se encuentren en el rango de valores de 1 a 10 μF, consiguiéndose el nivel de tensión más estable en caso de que ambos sean de 10 μF [73]. Se ha optado por usar dos de 1 μF ya que se encuentran dentro del rango de valores sugerido y permiten reutilizar el mismo modelo de condensador que ya se emplearon para el *bypass* de la alimentación de los PSoCs.

Dado que se ha escogido la versión del LP8340C que proporciona un valor prefijado de tensión usando el empaquetado NGD0006A, se tienen que cortocircuitar las patillas 4 y 5 por un lado, y las patillas 1 y 6 por el otro. La patilla 3, indicada como NC (*No Connection*), se deja al aire debido a que no se usa en este integrado. La patilla 7, la cual aparece referida en la Figura 3.7b como DAP (*Die Attach Paddle*) se conecta a tierra, su objetivo es servir como disipador de calor del integrado.



(a) Mapa de pines del empaquetado NGD0006A [73].



(b) Esquemático de Altium.

Figura 3.7: Diagrama de configuración del regulador de tensión LP8340C (salida fija de 3.3 V).

3.2.3.1. Entrada de alimentación externa

Como resulta lógico, se necesita un punto de entrada para la fuente de alimentación externa y que vaya hacia el regulador. Aunque en el Capítulo 2: Especificaciones del dispositivo de procesamiento central se mencionaba que se prefería el uso de dispositivos SMD se han dispuesto dos agujeros pasantes para la entrada de alimentación.

Aunque esta solución pueda no resultar adecuada para un producto final. Sin embargo, sí que es cierto que para el prototipo que se implementa en esta trabajo, este tipo de entrada resulta útil a la hora de hacer series de pruebas con variedad de fuentes de alimentación, como lo podrían ser fuentes de laboratorio. El en esquemático se ha usado el símbolo de la Figura 3.8 para representar esta entrada, su *footprint* no se ha incluido en la Tablas 3.3 y 3.4, ya que se han utilizado agujeros pasantes genéricos disponibles en Altium.

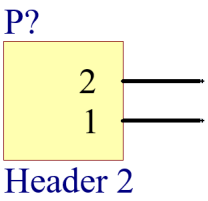


Figura 3.8: Esquemático de los agujeros pasantes usados para la entrada de alimentación externa.

3.2.4. INA333

Se busca una ganancia de aproximadamente 50 V/V. El valor más cercano que se puede alcanzar usando resistencias de valor comercial añadiendo los mínimos componentes es 51 V/V, para lo cual se necesita una resistencia 2 kΩ, este valor se puede extraer a partir de la expresión 3.1. Se ha optado por dividir el resistor en 2 de 1 kΩ, de manera que se tenga un punto en el cual sea posible medir el modo común. En la Figura 3.10 este punto se ha referido como CMM y es al que está conectado el *test point*, también llamado CMM, mencionado en el apartado anterior PSoC 5LP CY8C5868LTI-LP039.

El motivo de seleccionar este nivel de amplificación es el hecho de que las señales extraídas del cuerpo humano tienden a valores de amplitud relativamente pequeños. Por ejemplo, en el caso de la señal del corazón se esperan amplitudes del rango de los μV o de apenas unos pocos mV [111]. A la hora de amplificar, hay que tener en cuenta también que la señal debe de ser posteriormente muestreada por un convertidor analógico-digital. Estos últimos tienen su resolución comprendida entre un rango de valores máximo y mínimo, hay que amplificar teniendo presente que no se pueden superar este rango si se desea que la señal captada sea recuperada correctamente. La información relativa a la configuración del ADC del PSoC y los conceptos relacionados se puede encontrar de manera más detallada en el Capítulo 4: Diseño firmware y hardware para la obtención de adquisiciones.

En la Figura 3.10 se puede consultar el empaquetado usado para el INA333, así como un esquemático de las conexiones de sus patillas. Se procede a clarificar cada una de estas conexiones, a modo ilustrativo, se ha incluido también un esquema simplificado del interior del INA333 proporcionado por el fabricante y el cual se puede consultar en la Figura 3.9 [87].

- Las patillas 4 y 7 (V^+ y V^-) establecen los límites superior e inferior que puede tomar la señal de salida. Estas están conectadas a, respectivamente, la salida del *load switch* 1, el cual se utiliza para abrir o cerrar la alimentación a 3.3 V proveniente del regulador de tensión, y tierra. En el apartado Amplificador de instrumentación nos referíamos a estos voltajes como $+V_s$ y $-V_s$.
- Los pines 2 y 3, denominados como V_{IN+} y V_{IN-}, son los puntos de entrada de las dos componentes de las señales diferenciales a amplificar. Estas señales se introducen en el PSoC 5LP directamente desde los puertos de entrada a la PCB y se redirigen a los pines del PSoC conectados al INA333.
- Entre los pines 1 y 8 se conecta el resistor que ajusta la ganancia del amplificador.
- El punto REF (pin 5) se usa para la referencia de entrada, la cual permite establecer un *offset* para la señal de salida. Por como se configura el ADC del PSoC 5LP (*rail-to-rail*), este no tiene resolución negativa, es decir, no es capaz de muestrear valores negativos de tensión. Por tanto, el desplazamiento vertical de la señal que se añade a través del pin REF, permite que si sea posible muestrear valores negativos de señales como podría ser la del corazón.

$$G = 1 + \frac{100k\Omega}{R_G} \tag{3.1}$$

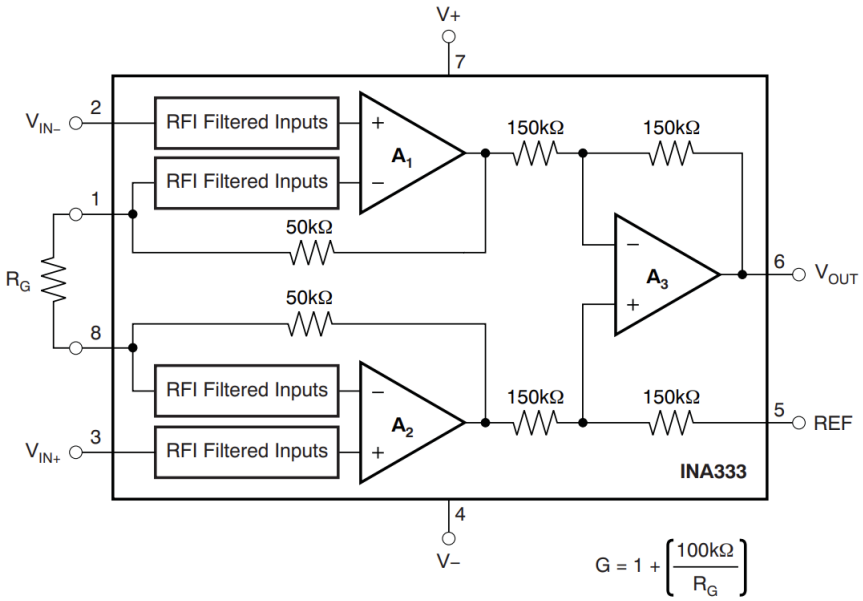
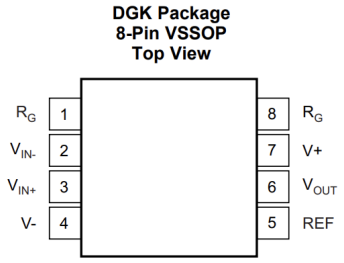
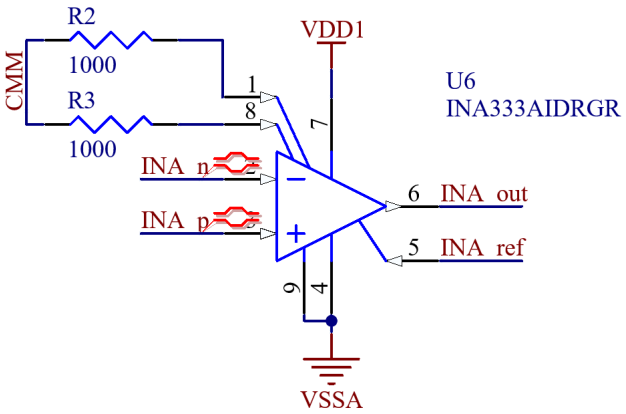


Figura 3.9: Esquema simplificado del interior del INA333 [87].



(a) Mapa de pines del empaquetado DGK [87].



(b) Esquemático de Altium.

Figura 3.10: Diagrama de configuración del amplificador de instrumentación INA333.

3.2.5. TCK106AG

Tal y como ya se ha establecido, el interruptor de carga se utiliza para activar o desactivar la alimentación que va hacia los puertos de entrada a la PCB. Se utilizan dos TCK106AG, controlando cada uno de ellos la alimentación de dos puertos. Adicionalmente, el *switch* controlado desde la entrada EN1 habilita y deshabilita también el funcionamiento del amplificador de instrumentación. Esta información se puede consultar en los diagramas de bloques de la sección Arquitectura lógica del sistema del Capítulo 2.

El pin B2, etiquetado como EN1 y EN2, se encuentra conectado a dos pines del PSoC 5LP. En caso de que estos pines se establezcan a un valor de tensión considerado como alto, esto es una tensión de 0.9 V o más, el *switch* permite el paso de corriente. Por otra parte, VDD viene a ser el nodo de salida del regulador de tensión, mientras que en VDDX está al voltaje que proporciona dicho regulador (en caso de que ENX = HIGH), menos la tensión que se disipe debido a la resistencia presentada por el propio *switch*.

Aunque el fabricante asegura que es dispositivo el capaz de operan con normalidad sin necesitar de un condensador de *bypass* [80], si que recomienda también usar uno en caso de que sea posible en el pin A2. Dado que este pin se encuentra conectado a la salida del regulador y este último ya cuenta con un pin de *bypass* en esta, no se hace necesario añadir otro.

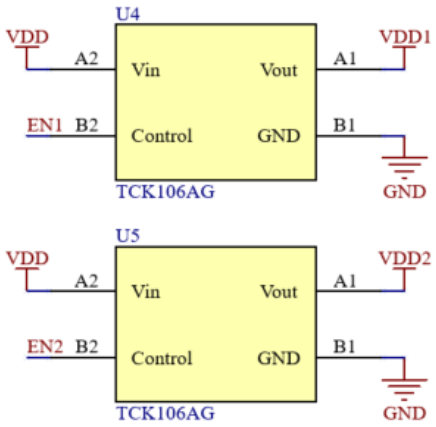


Figura 3.11: Esquema de conexiones de los *load switch* que controlan la alimentación hacia los puertos de entrada de la PCB [80].

3.2.6. Puertos de I/O y de programación

Los puertos están configurados de manera de que dos de sus pines estén conectados a GPIOs del PSoC 5LP, mientras que los otros dos se usan para alimentación (en caso de que sea necesario) de los kits de expansión o los sensores conectados al dispositivo central de procesamiento. Los voltajes de alimentación provienen de la salidas de los *load switch*. Los puertos sólo tienen 4 posibles conectores, los conectores 5 y 6 están cortocircuitados a tierra y en el modelo físico se usan para la soldadura a la PCB.

Respecto a los pines usados para la programación del PSoC 5LP, dado que se tienen más pines de los necesarios para la programación mediante interfaz JTAG. Lo pines sobrantes se han cortocircuitado a tierra tal y como se indica en la documentación del MiniProg3 [95], el cual se utiliza para reprogramar ambos PSoCs.

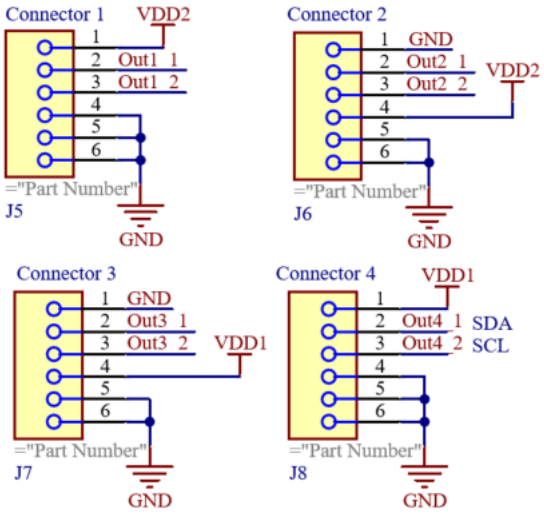
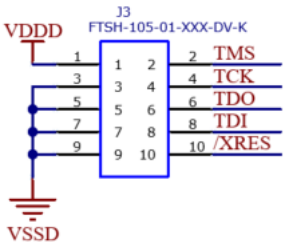
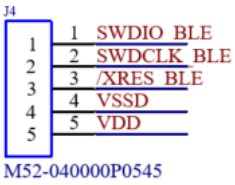


Figura 3.12: Esquema de conexiones de los puertos de entrada a la PCB [93].

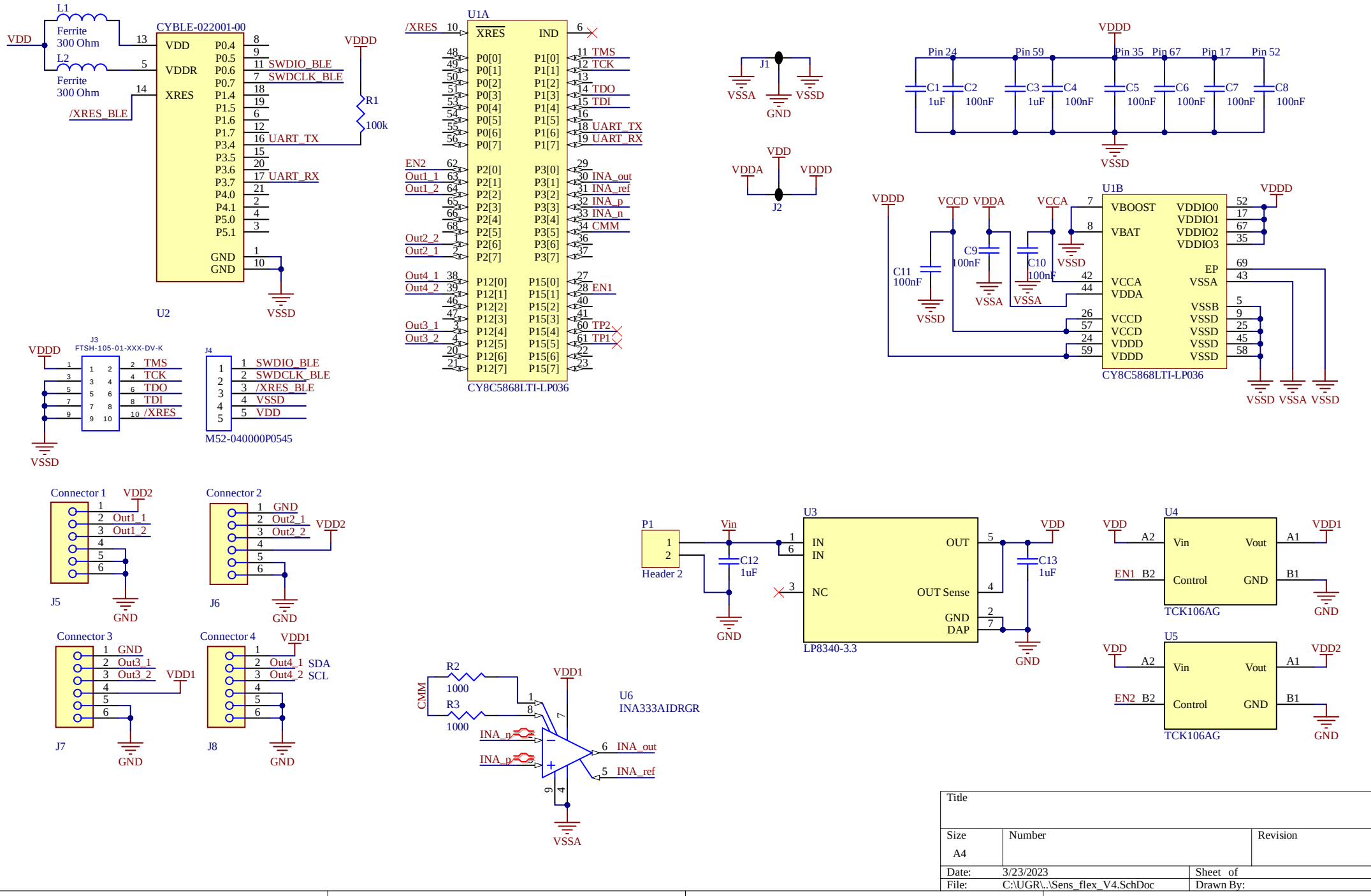


(a) Esquema de conexiones del puerto de programación del PSoC 5LP CY8C5868LTI-LP039 [94].



(b) Esquema de conexiones del puerto de programación del módulo CYBLE-022001-00. [96].

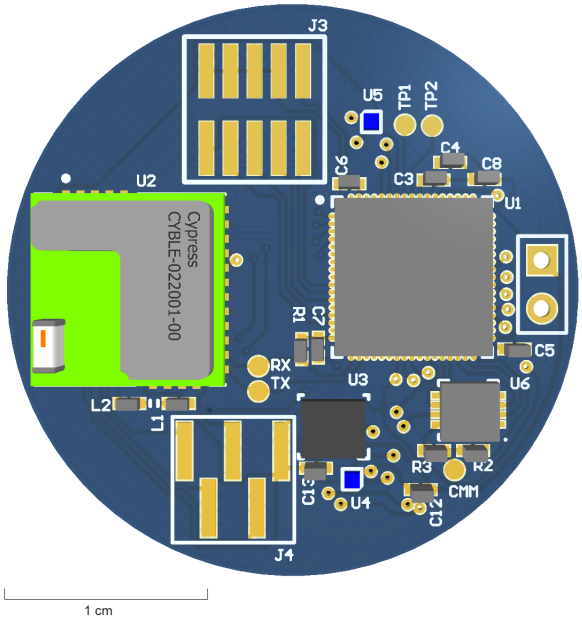
Figura 3.13: Conexiones de los puertos de programación.


L1
Ferrite
300 Ohm
L2
Ferrite
300 Ohm
VDD
/XRES_BLE
CYBLE-022001-00
VDD
VDDR
XRES
P0.4
P0.5
P0.6
P0.7
P1.4
P1.5
P1.6
P1.7
P3.4
P3.5
P3.6
P3.7
P4.0
P4.1
P5.0
P5.1
GND
GND
SWDIO_BLE
SWDCLK_BLE
UART_TX
UART_RX
VDDD
R1
100k
U2
VSSD
U1A
/XRES
XRES
IND
P0[0]
P0[1]
P0[2]
P0[3]
P0[4]
P0[5]
P0[6]
P0[7]
P1[0]
P1[1]
P1[2]
P1[3]
P1[4]
P1[5]
P1[6]
P1[7]
TMS
TCK
TDO
TDI
UART_TX
UART_RX
EN2
Out1_1
Out1_2
P2[0]
P2[1]
P2[2]
P2[3]
P2[4]
P2[5]
P2[6]
P2[7]
P3[0]
P3[1]
P3[2]
P3[3]
P3[4]
P3[5]
P3[6]
P3[7]
INA_out
INA_ref
INA_p
INA_n
CMM
Out2_2
Out2_1
Out4_1
Out4_2
Out3_1
Out3_2
P12[0]
P12[1]
P12[2]
P12[3]
P12[4]
P12[5]
P12[6]
P12[7]
P15[0]
P15[1]
P15[2]
P15[3]
P15[4]
P15[5]
P15[6]
P15[7]
EN1
TP2
TP1
CY8C5868LTI-LP036
J1
VSSA
VSSD
GND
VDD
VDDA
VDDD
J2
Pin 24
Pin 59
Pin 35
Pin 67
Pin 17
Pin 52
C1
1uF
C2
100nF
C3
1uF
C4
100nF
C5
100nF
C6
100nF
C7
100nF
C8
100nF
VSSD
U1B
VDDD
VCCD
VDDA
VCCA
VBOOST
VBAT
VCCA
VDDA
VCCD
VCCD
VDDD
VDDD
VDDIO0
VDDIO1
VDDIO2
VDDIO3
EP
VSSA
VSSB
VSSD
VSSD
VSSD
VSSD
C9
100nF
C10
100nF
C11
100nF
VSSD
VSSA
VSSA
VSSD
VSSA
VSSD
CY8C5868LTI-LP036
J3
FTSH-105-01-XXX-DV-K
VDDD
TMS
TCK
TDO
TDI
/XRES
VSSD
J4
SWDIO_BLE
SWDCLK_BLE
/XRES_BLE
VSSD
VDD
M52-040000P0545
Connector 1
VDD2
Out1_1
Out1_2
J5
GND
Connector 2
GND
Out2_1
Out2_2
VDD2
J6
GND
Connector 3
GND
Out3_1
Out3_2
VDD1
J7
GND
Connector 4
VDD1
Out4_1
Out4_2
SDA
SCL
J8
GND
P1
Header 2
Vin
C12
1uF
GND
U3
IN
IN
NC
OUT
OUT Sense
GND
DAP
LP8340-3.3
VDD
C13
1uF
GND
U4
VDD
A2
Vin
Vout
A1
VDD1
EN1
B2
Control
GND
B1
GND
TCK106AG
U5
VDD
A2
Vin
Vout
A1
VDD2
EN2
B2
Control
GND
B1
GND
TCK106AG
R2
1000
R3
1000
CMM
VDD1
U6
INA333AIDRGR
INA_n
INA_p
INA_out
INA_ref
VSSA
Title
Size
A4
Number
Revision
Date:
3/23/2023
Sheet of
File:
C:\UGR\..\Sens_flex_V4.SchDoc
Drawn By:

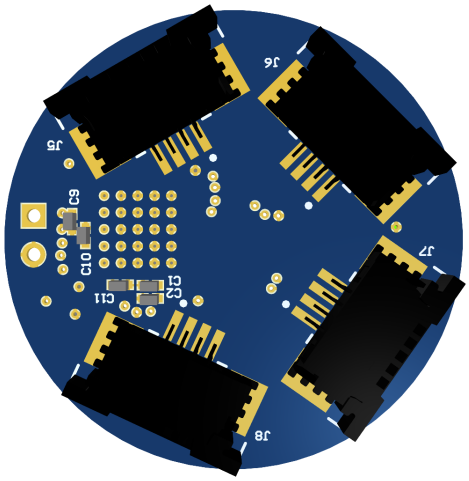
3.3. Diseño y análisis de la PCB

En la sección Placas de circuito impreso (PCB) del Capítulo 2: Especificaciones del dispositivo de procesamiento central se estableció que para este proyecto el tipo de PCB que resulta más adecuada es una DSB, es decir, se pueden colocar componentes en ambas caras de la placa. En esta sección se trata la PCB que finalmente se ha diseñado aglutinando todos los componentes anteriormente tratados. En la Figura 3.14 se pueden consultar imágenes del diseño final de la PCB desde varios puntos de vista.

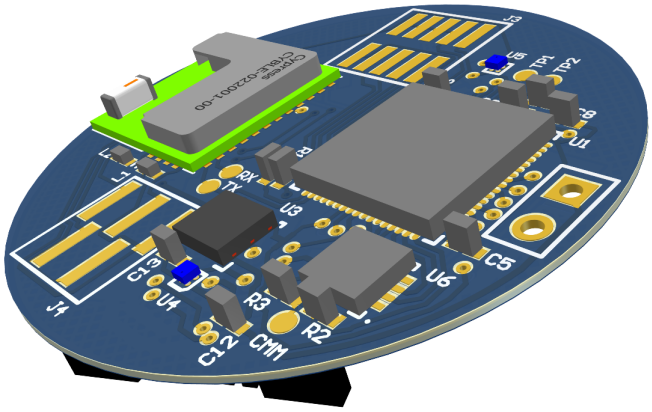
Se ha seleccionado una forma circular para la PCB y se le ha tenido que dar un **diámetro de 30 mm** para poder posicionar y enrutar todos los elementos. En el Anexo F se pueden visualizar imágenes tomadas de la PCB física.



(a) Vista desde arriba.



(b) Vista desde abajo.

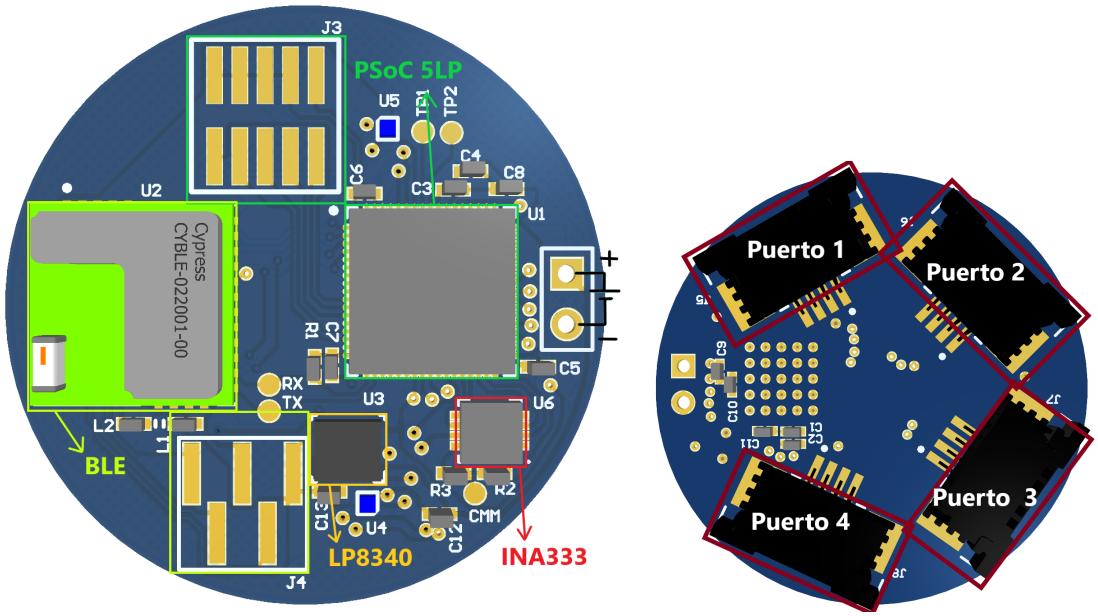


(c) Vista de perfil

Figura 3.14: Modelo 3D de la PCB que forma el dispositivo de procesamiento central vista desde varios ángulos.

3.3.1. Mapa de componentes

En la Figura 3.15 se han vuelto a incluir las imágenes ya mostradas en la Figura 3.14 habiéndose resaltado los componentes principales del circuito. El módulo Bluetooth se ha colocado en un extremo de la placa tal y como se recomendaba en [110]. El componente J4 es la zona de metal expuesto donde se deben de soldar los pines macho SMD para la programación del módulo BLE. Exactamente lo mismo es J3 para con el PSoC 5LP.



(a) Vista desde arriba.

(b) Vista desde abajo.

Figura 3.15: PCB del dispositivo de procesamiento central con las partes que lo componen resaltadas.

3.3.2. Conexiones y planos de referencia

En este apartado se busca tratar y clarificar las consideraciones seguidas para la conexión de los elementos en la PCB.

3.3.2.1. Planos de referencia

En la Figura 3.16 se pueden consultar un mapa de la zonas de metal conductor que la PCB posee en sus caras superior e inferior. La parte inferior se ha configurado como plano de tierra, es decir, el área queda cubierta por una capa conductora de referencia 0 V. También se utiliza para trazar las entradas de los puertos de entrada hacia el plano superior, ya que dichas entradas se han colocado en la parte inferior de la PCB para maximizar el área aprovechada.

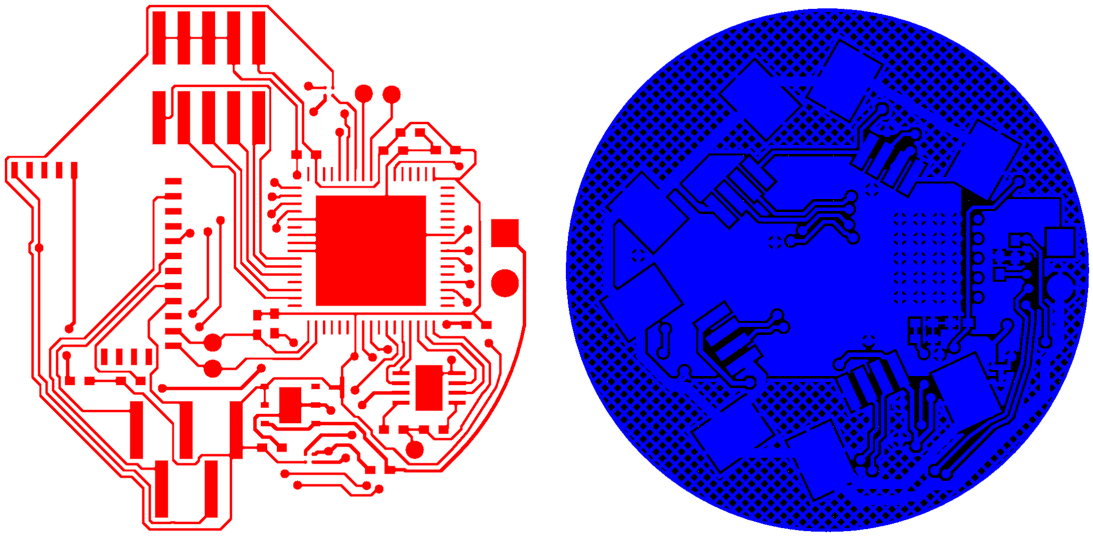
El motivo de usar el área inferior completa como plano plano de tierra, es el hecho de que ayuda a reducir las EMI. Esto ocurre debido a que la región plana actúa a modo de escudo electromagnético para con las pistas del área opuesta de la placa. De manera simultanea, el plano de tierra se aprovecha como disipador de calor [112]. Este hecho ya se comentaba también en la sección LP8340C de este capítulo.

Para un mejor desempeño del plano de tierra, hay que evitar tanto como sea posible el que por esta cara de la PCB transcurran pistas o *vias* que transporten señales. En el diseño final, con la intención de alcanzar la máxima miniaturización posible, ha sido necesario pasar algunas algunas pistas entre el plano de tierra. Se ha procurado hacer estas trazas tan cortas como se ha podido y que ocupen un área mínima.

También se recomienda separar los planos de tierra de dispositivos analógicos y digitales para reducir la transmisión de ruido [112]. En la Figura 3.16b se puede ver una región compuesta por una rejilla y otra sólida. El área solida corresponde a la conexión a tierra de los elementos digitales; por otro lado, a la rejilla se conecta la referencia de tierra de los elementos analógicos.

Un par de consideraciones respecto del área superior son:

- El PSoC 5LP tiene metal expuesto en su parte inferior. La documentación [67] recomienda conectar este área a tierra para logra un mejor comportamiento mecánico, térmico y eléctrico. Dado que los planos de tierra se encuentran en la otra cara de la PCB, se ha establecido una matriz de *vias* para lograr la conexión. En la Tabla 3.3 se puede consultar este hecho más claramente.
- Ya que ha sido necesario lanzar pistas por debajo de la antena, se ha procurado acercar estas lo máximo posible hacia el centro de la placa para evitar en la medida de lo posible las interferencias desde y hacia la antena.



(a) Conexiones plano superior.

(b) Conexiones plano inferior.

Figura 3.16: Conexiones de la PCB en los planos superior e inferior.

3.3.2.2. Trazas

A la hora de trazar las pistas para conectar los componentes, es habitual que estas tengan que realizar giros. Para empezar, en todos los casos se recomienda evitar ángulos agudos. En el caso de diseños de alta frecuencia, con alta frecuencia haciendo referencia a magnitudes de varios MHz o de GHz, es importante evitar giros de 90^{o} con la intención de reducir la EMI generada. Dado que este diseño no va a trabajar con frecuencias de estos rangos, los ángulos rectos resultan aceptables caso de que no sea posible otra alternativa [113]. Teniendo en cuenta estas consideraciones, en todo el diseño se han favorecido los giros de 45^{o}. Dichos ángulo se usa también con la intención de tratar de acortar las pistas tanto como sea posible para tratar de reducir los efectos de fuentes externas de ruido, ya que al fin y al cabo las pistas conductoras acaban actuando como antenas que captan señales del exterior [112].

Hay que seleccionar también el ancho de pista. Para este diseño se han utilizado varios anchos de valor estándar que a continuación se mencionan:

- Para las pistas por la que circula voltaje de alimentación desde la fuentes externa hacia el regulador, se ha empleado un ancho de 0.254 mm. Esto incluye a las pistas que se usan para las conexiones a tierra. Es habitual que la pistas que conducen alimentación sean más

anchas debido a la mayor corriente que portan con la intención de reducir la impedancia del conductor. En cualquier caso, dado que el dispositivo trabaja con corriente pequeñas, siendo en la intensidad máxima en el caso más extremo de 1 A, con el ancho elegido es suficiente.

- En la Figura 3.16b se puede ver que dentro de la sección del plano de tierra que está formada por una rejilla, se han creado también trazas conectando las tierras de los puertos de entrada. Estas pistas tienen un ancho de 0.548 mm.
- Para todo el resto de pistas utilizadas en el circuito se ha utilizado un ancho de 0.2 mm.

Hay que tener en cuenta que en este diseño cuestiones como la resistencia o impedancia de las trazas no resultan de un aspecto crítico debido a que estas son en general cortas y que no se está trabajando a altas frecuencias.

3.3.2.3. Vías

La *vías* usadas para la transmisión entre las dos caras de la PCB se han diseñado con un diámetro total de 0.524 mm, esto incluye toda el área con material conductor y no sólo el propio agujero. El agujero en sí tiene un diámetro de 0.254 mm.

3.3.3. Puntos de prueba

Se han añadido varios puntos de prueba, o como es habitual denominarlos *test points*. Estos consisten en zonas de metal expuesto conectadas a nodos de interés con la intención de ser utilizadas en la medida de valores de tensión o señales. Dicho de otra forma, estos puntos ayudan a verificar el correcto funcionamiento del circuito y facilitan la tarea de detectar o diagnosticar errores en caso de que se produzcan. En total se han añadido 5 *test points*.

- **TP1 y TP2**: Dado que el PSoC 5LP cuenta con varios pines GPIO que han quedado sin utilizar, se han conectado estos puntos de prueba a, respectivamente, los pines 61 y 60 del mismo. El motivo es poder utilizarlos para comprobar el correcto funcionamiento del *firmware* o del *hardware* en caso de que fuese necesario. Si, por poner un ejemplo, se quisiera comprobar la salida del amplificador de instrumentación, se puede modificar mediante *firmware* el pin que se usa para sacar esta señal y obtenerla en uno de estos dos *test points*.

  El hecho de contar con TP1 y TP2 conectados a pines libres del PSoC 5LP, ayuda a reducir la cantidad de puntos de prueba totales que hay que añadir en la PCB, debido a que los buses que existen en el interior del propio integrado permiten redirigir señales entre pines. Esto permite que en vez de añadir un *tests point* a cada nodo conectado al PSoC, se puede en su lugar redirigir la salida del punto de interés hacia uno de los dos puntos de prueba, siempre y cuando este punto de interés incluya un pin GPIO del PSoC.
- **UART_TX y UART_RX**: Se emplean para poder captar los paquetes que se transmiten mediante comunicación UART entre los dos PSoCs.
- **CMM**: Su aplicación es poder medir el modo común del amplificador de instrumentación en caso de que se den problemas para polarizarlo correctamente.

Capítulo 4

Diseño firmware y hardware para la obtención de adquisiciones

El objetivo de este capítulo es tratar el diseño del *software* con para la placa central y el *hardware* externo que sea necesario añadir para poder adquirir medidas de cuerpo humano. Se recuerda que para la implementación de *firmware* sobre los PSoCs se usa el programa PSoC Creator. El capítulo se ha dividido en las siguientes secciones:

- En Obtención del electrocardiograma (ECG) se puede consultar toda la información relativa a la obtención de la señal del corazón mediante un electrocardiograma, ello incluye un resumen teórico de esta técnica y el desarrollo *firmware* para implementarla mediante el dispositivo de procesamiento central expuesto en los capítulos anteriores.
- Obtención del nivel de saturación de oxigeno alberga la misma información que la sección anterior, pero relativa a la adquisición del nivel de saturación de oxígeno en sangre mediante técnicas de fotopletismografía.

4.1. Obtención del electrocardiograma (ECG)

En esta sección se trata el *firmware* desarrollado para obtener un electrocardiograma o ECG (*electrocardiogram*) mediante el sistema construido a partir del PSoC 5LP CY8C5868LTI-LP039, así como un repaso a los conceptos relacionados con la obtención de la señal del corazón. Para la adquisición de esta medida se usarán sensores comerciales directamente conectados a los puertos de entrada de la placa principal, sin que sea necesario desarrollar un kit de expansión con su correspondiente circuitería.

4.1.1. Fundamento teórico del ECG

El corazón está compuesto por tejido cardíaco, a su vez formado por células, conocidas como cardiomiocitos, capaces de polarizarse y despolarizarse en cada ciclo cardíaco para causar las contracciones que permiten el bombeo de sangre por el cuerpo. La señal eléctrica generada durante este proceso se puede captar mediante sensores colocados en la superficie del pecho o de las extremidades, el ECG consiste en la obtención y representación lineal del voltaje en función del tiempo de la actividad eléctrica cardíaca captada mediante dichos sensores [111, 114]. La señal cardíaca a menudo se puede encontrar referida como señal PQRST, en la Figura 4.2 se puede encontrar una representación de esta.

4.1.1.1. Funcionamiento del corazón y la señal cardíaca

El corazón está compuesto por cuatro cámaras, siendo las dos superiores conocidas como aurículas y las dos inferiores ventrículos [114]. En la Figura 4.1 se pueden consultar una imagen de la fisiología del corazón, donde RA (*right atrium*) y RV (*right ventricle*) son respectivamente la aurícula y ventrículo derechos. LA (*left atrium*) y LV (*left ventricle*) son la aurícula y ventrículo izquierdo.

Respecto de la señal cardíaca, esta se puede interpretar de la siguiente forma [111, 114]:

- La onda P es el resultado de la despolarización de la aurícula.
- El complejo QRS es el correspondiente de la despolarización ventricular y marca el inicio de la contracción ventricular.
- La onda T se produce debido a la repolarización del ventrículo. El intervalo QT se mide desde el principio del complejo QRS hasta el final de la onda T.
- Al final del ciclo cardíaco se produce la señal U, cuyo origen aún no se tiene del todo claro, aunque existen varias teorías entre las que apuntan a que es debida a un potencial de repolarización.

A partir de la amplitud de las señales anteriores y de la duración de algunos segmentos de interés se puede realizar el diagnóstico de multitud de anomalías. Entre los segmentos que resulta de interés estudiar se tienen [111]:

- El intervalo temporal de los puntos P a Q corresponde con la trasmisión de impulsos eléctricos a través del nodo sinoauricular o nodo SA (*Sinoatrial node*), dicho de otra forma, el intervalo PQ se corresponde con el tiempo trascurrido desde la despolarización de la aurícula hasta el comienzo de la despolarización ventricular. Esto va desde el comienzo de P hasta el inicio del complejo QRS.
- El intervalo QT se mide desde el final del complejo QRS, en el punto identificado como J, hasta el final de la onda T.
- El segmento ST se toma desde el final del complejo QRS hasta el comienzo de la onda T.
- El segmento de tiempo que va desde R al siguiente R, llamado intervalo RR, representa un ciclo cardíaco y se emplea para el cálculo del ritmo cardíaco.

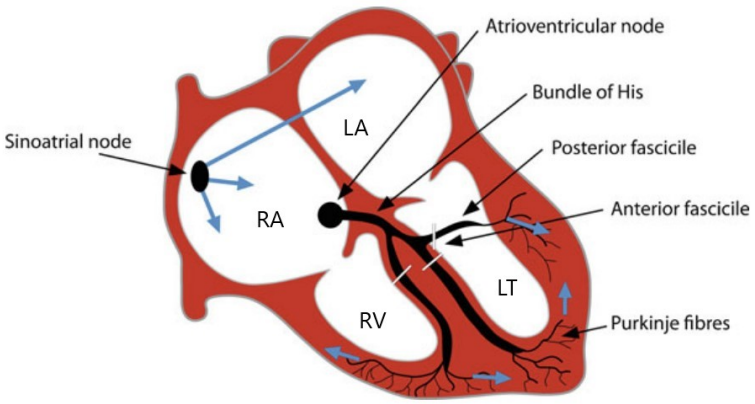


Figura 4.1: Cavidades del corazón y conducción eléctrica a través de él [111].

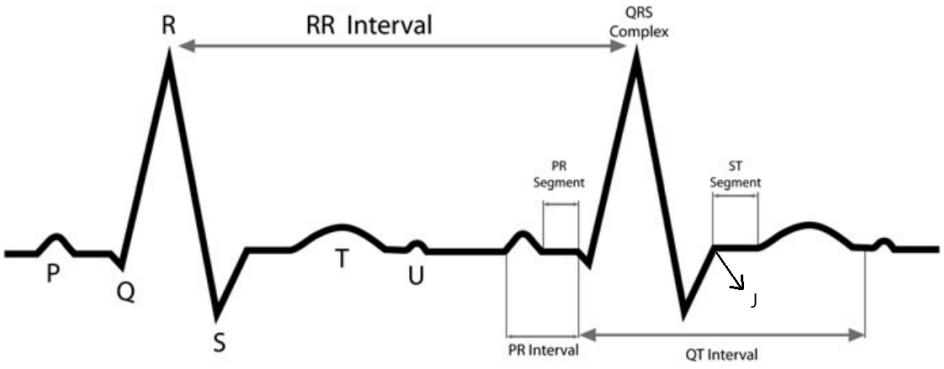


Figura 4.2: Señal cardíaca con los intervalos y puntos de interés resaltados. El eje vertical representa voltaje, mientras que el horizontal es el tiempo [111].

4.1.1.2. Adquisición de la señal cardíaca

La instrumentación usada para la adquisición de un ECG debe de trabajar a frecuencias bajas, en [114] se recomienda un ancho de banda de 0.5 a 150 Hz, aunque en la bibliografía se pueden encontrar varios rangos mencionados como adecuados para el filtrado paso banda de la señal cardíaca. Por ejemplo, en [115] se recomienda un filtrado paso banda en el rango de 0.1 a 100 Hz o uno paso baja con frecuencia de corte 30 Hz entre otros. Este filtrado es necesario para lidiar con multitud de interferencias provenientes de otros equipos médicos que se pueden encontrar presentes en la inmediaciones del paciente, las propias interferencias generadas por el resto de músculos, el ruido de la alimentación (50 o 60 Hz), ruido causado por un mal contacto entre la piel y los sensores, etc [111, 115].

La señal del corazón se adquiere de manera diferencial entre dos puntos del cuerpo. De forma habitual, la adquisición ocurre en el brazo derecho (RA o *right arm*) e izquierdo (LA o *left arm*) y la pierna izquierda (LL o *left leg*). Estas posiciones se conocen como el triángulo de Einthoven. En la pierna derecha se añade un electrodo que actúa a modo de referencia para reducir el ruido de la medida. A continuación, se exponen los pares que representan las tres posibles combinaciones para extraer la señal del corazón de manera diferencial [114]:

$$\mathrm{I} = \mathrm{V_{LA}} - \mathrm{V_{RA}}$$

$$\mathrm{II} = \mathrm{V_{LL}} - \mathrm{V_{RA}}$$

$$\mathrm{III} = \mathrm{V_{LL}} - \mathrm{V_{LA}}$$

Estas medidas están relacionadas entre ellas y de hecho se tiene que cumplir la condición de que II = I + III [114].

Existen varios métodos para la adquisición del ECG, siendo una de uso muy extendido la conocida como ECG de 12 derivaciones (o "12-lead ECG") [116], la cual en adición a los sensores presentes en el triángulo de Einthoven añade también varios puntos de adquisición en las costillas. Para este trabajo se utiliza la técnica de 3 derivaciones ("3-lead"), el motivo de esta elección es el hecho de que permite la obtención de la señal del corazón con no más de tres puntos de adquisición, a diferencia de la de 12 derivaciones que precisa de 12. Además, esta última necesita de personal especializado y resulta muy aparatosa para una medida a obtener desde un dispositivo portable.

En la técnica de 3 derivaciones, como su nombre indica, se usan 3 electrodos. Dos de ellos son para la adquisición de la señal diferencial y el tercero es la referencia. La medida de la señal diferencial se realiza en las muñecas derecha (RA) e izquierda (LA) y la referencia se coloca en el tobillo derecho (RL). En resumen, se utiliza el primer par del triángulo de Einthoven. También es posible acercar los puntos de medida al torso, tal y como se muestra en la Figura 4.4. En este último caso, la señal diferencial se toma por debajo de las clavículas derecha e izquierda y la referencia se coloca en el extremo inferior izquierdo de las costillas (LL).

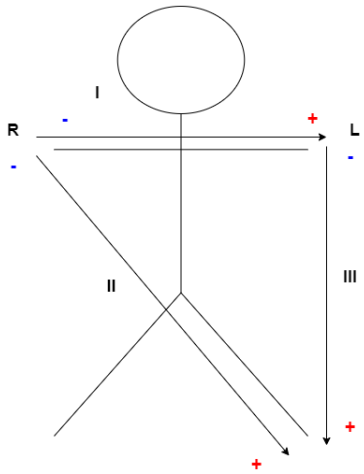


Figura 4.3: Triángulo de Einthoven. Se puede ver que las polaridades coinciden con la propagación de la señal eléctrica proveniente del Nódulo sinoauricular mostrado en la Figura 4.1.

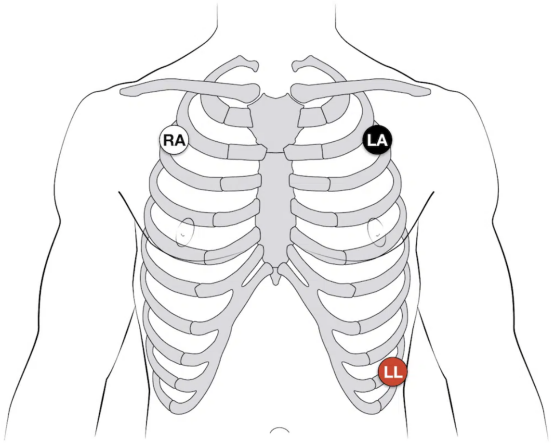


Figura 4.4: Colocación de electrodos en el torso para la realización de ECG según la técnica de 3 derivaciones [117].

4.1.2. Sistema para la adquisición del ECG

En este apartado se trata el *firmware* concreto desarrollado sobre el PSoC 5LP que permite la obtención de la señal cardíaca haciendo uso de la técnica de 3 derivaciones. Se ha planteado un sistema que adquiera la señal del corazón durante, aproximadamente, 1 minuto.

La PCB que incluye al dispositivo de procesamiento central ya dispone de todo el *hardware* que pudiera resultar necesario para el tratamiento de la señal cardíaca, por tanto, únicamente será necesario conectar a los puertos de entrada del dispositivo los electrodos para la toma de las señales provenientes del corazón. Usando los componentes disponibles en el interior del propio PSoC 5LP y el amplificador de instrumentación INA333, se ha planteado el diagrama de bloques de la Figura 4.5 que representa el flujo para el tratamiento de la señal diferencial.

La señal extraída del cuerpo se ve afectada por varias fuentes de ruido [118, 119]:

- Ruido de baja frecuencia conocido como *baseline wander* (BW) y que viene a estar causado de manera habitual por el movimiento y la respiración de la persona sobre la que se están adquiriendo las medidas. Este ruido se suele encontrar en las frecuencias de 0.15 Hz a 0.3 Hz.
- Ruido de 50 o 60 Hz proveniente de la alimentación, el cual tiene un ancho de banda de aproximadamente 1 Hz.

- Ruido de alta frecuencia causado por equipos electrónicos cercanos.

Profundizando en el diagrama de la Figura 4.5, el procesamiento sigue el siguiente orden:

1. Desde un puerto de entrada se reciben las dos componentes de la señal diferencial captadas desde dos electrodos. Estas se pasan por dos amplificadores de ganancia programables (PGA) [121], disponibles en el interior del PSoC, en caso de que sea necesario añadir amplificación extra previa al INA333. En principio, se mantienen a ganancia 1, es decir, salvo que se indique lo contrario no amplifican la señal previamente al amplificador de instrumentación.

2. Se redirigen ambas componentes de la señal cardíaca desde el PSoC 5LP hacia el INA333 colocado de manera externa. Aquí, a la componente proveniente del PGA1 se le resta la que sale del PGA2, el resultado se amplifica y se le añade un cierto desplazamiento vertical u *offset*. El motivo de añadir este *offset* es que, como se verá a continuación, el ADC usado no tienen resolución para valores de tensión negativos, con lo cual se añade el desplazamiento de manera que todos los valores de tensión muestreados sean positivos.

3. La señal amplificada se devuelve al PSoC 5LP, donde se ha usado un seguidor de tensión para desacoplar impedancias.

4. El ADC Delta-Sigma del PSoC muestrea la señal y el resultado se pasa por un filtro paso baja para eliminar el ruido proveniente de componentes frecuenciales no deseadas.

5. Para acabar, las muestras de la señal se envían al módulo Bluetooth Low Energy mediante comunicación UART, y desde este hacía un dispositivo móvil con una aplicación compatible para reconstruir las muestras, mostrarlas al usuario y almacenarlas. Dado que la máxima cantidad de bits que se pueden transmitir mediante el bloque UART son 9 y las muestras que se están tomando son de 16 bits, para cada muestra se separan sus 8 bits más y menos significativos y se envían por separado. Más tarde, en la aplicación Android, ambas partes se volverán a unir para reconstruir la información.

   Adicionalmente, esta comunicación se usa también para recibir ordenes desde la aplicación. En el estado actual, desde la app se envía la orden para comenzar a adquirir muestras.

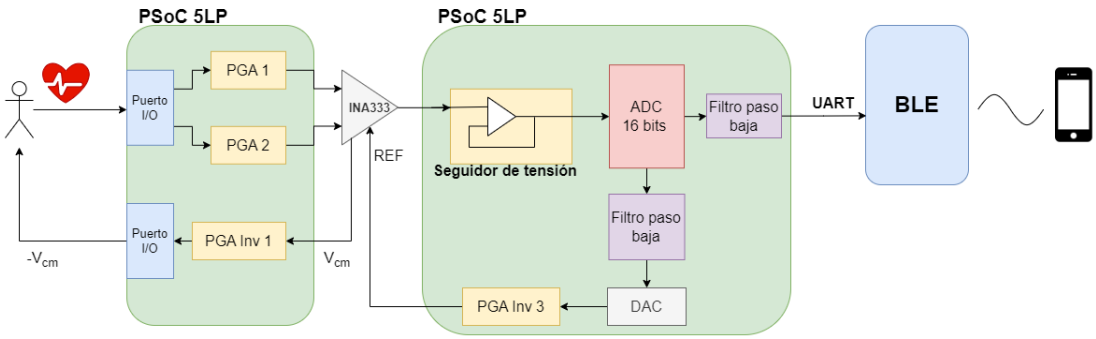


Figura 4.5: Esquema lógico seguido para la obtención y transmisión de las señales del ECG mediante el dispositivo de procesamiento central.

El INA333 tiene una entrada de referencia (REF) que posibilita añadir un *offset* a la salida. Dado que los componentes que se tiene el PSoC 5LP no son capaces de operar con valores de tensión negativos, se añade este desplazamiento vertical con el pretexto de asegurar que todos los valores que lleguen al ADC sean positivos.

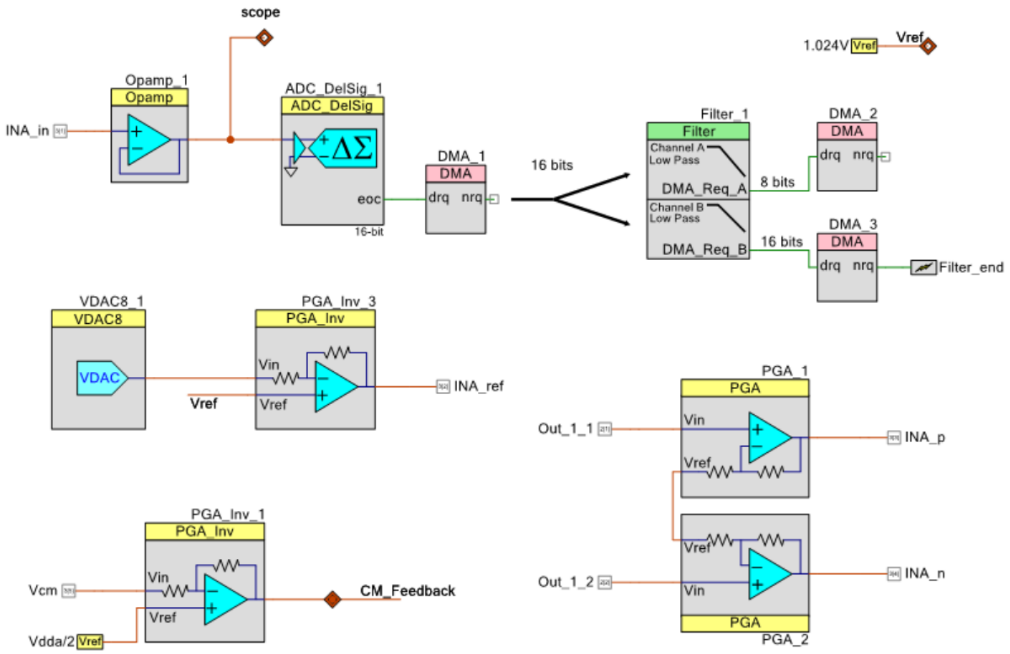
El algoritmo también añade un sistema de realimentación a través del electrodo de referencia para tratar de minimizar el efecto del ruido de modo común del amplificador. El objetivo de dicha realimentación es introducir en el cuerpo la señal de ruido con la amplitud invertida, de este modo al obtener la salida del amplificador el ruido de modo común se suma a él mismo pero con amplitud inversa, quedando teóricamente suprimido. Esta sección sigue la siguiente lógica:

1. Se dispone de un pin de entrada (Vcm) al PSoC 5LP, a través del cual se capta la señal de ruido de modo común del INA333. Esta señal se localiza en el nodo existente entre los dos resistores que ajustan la ganancia del amplificador de instrumentación, como ya se trató en el Capítulo 3.

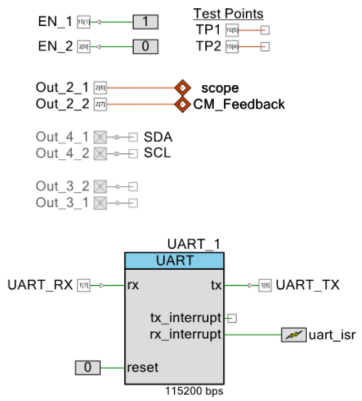
2. La señal de ruido se pasa por un amplificador inversor de ganancia 1, de modo que se obtiene la señal de ruido común invertida.

3. La señal obtenida en el amplificador se introduce en el cuerpo mediante el electrodo de referencia.

En la Figura 4.6 se muestran los esquemáticos desarrollados en PSoC Creator para la programación del PSoC 5LP que implementan la lógica descrita en la Figura 4.5. Se procede en los subapartados a continuación al estudio de la configuración individual de los bloques más relevantes que componen el esquemático. En el Anexo B: Configuración de los módulos en PSoC Creator se puede consultar la pestaña de configuración de cada bloque en PSoC Creator. Por otra parte, en el Anexo Firmware usado en el PSoC 5LP se ha incluido el código C que se ha programado en el PSoC y que funciona en conjunto con los elementos añadidos como bloques gráficamente.



(a) Página 1.



(b) Página 2.

Figura 4.6: Esquemático de PSoC Creator para la interconexión de los componentes que implementan la adquisición y tratamiento de la señal de corazón mediante un ECG.

4.1.2.1. Configuración: ADC Delta-sigma

Se ha configurado el conversor analógico-digital para que muestre a una velocidad de 2006 muestras/segundo. Según el teorema de muestreo de Nyquist-Shannon, para que una señal analógica se puede digitalizar recuperando toda la información, se debe de muestrear a una frecuencia de al menos el doble del máximo componente en frecuencia de dicha señal. En la práctica, debido al ruido y al empleo de filtros no ideales, se puede usar una frecuencia de muestreo de 5 o 10 veces más de la frecuencia de la señal analógica.

El ADC no se puede configurar con menos de una frecuencia de 2006 muestras/segundo. Para evitar sobremuestreo, previamente a enviar la señal al módulo BLE, se hace un diezmado de las muestras eliminando del vector que las almacena una de cada dos, con lo cual en la práctica la señal recuperada va a estar muestreada a una frecuencia de aproximadamente 1000 muestras/segundo. Es relevante no sobremuestrear las señales ya que se incrementa el espacio necesario para almacenar las muestras y los tiempos de transmisión. El rango de las frecuencias de muestreo que se pueden seleccionar está directamente relacionado con el número de bits de la resolución de ADC. En caso de incrementar el número de bits se podría reducir el frecuencia de muestreo sin tener que eliminar *a posteriori* una de cada dos muestras. Sin embargo, la resolución extra que se consigue no es realmente necesaria y por contra incrementa también los tiempos de transmisión y el espacio ocupado por la señal digitalizada.

Adicionalmente, el ADC se ha configurado de manera *rail-to-rail*, esto quiere decir que el rango de valores que puede muestrear se encuentra comprendido entre sus niveles de alimentación, en este caso de 0 a 3.3 V. Sin embargo, no es habitual que los ADCs lleguen hasta los valores máximos y mínimos teóricamente posibles. En su lugar, en muchas ocasiones no son capaces de muestrear estos valores tan extremos. Como se puede consultar en la Figura B.1, PSoC Creator indica que en la práctica el ADC sólo es capaz de muestrear valores que se encuentren aproximadamente 250 mV por debajo del valor de alimentación y 100 mV por encima del mínimo.

La resolución del ADC es de 16 bits, con lo cual se pueden tener $2^{16} - 1$ valores distintos de tensión medidos sin contar el cero. Resolviendo por tanto cual es la tensión mínima que causa un incremento en 1 bit del valor digitalizado, que viene a ser lo mismo que la sensibilidad del ADC, se obtiene:

$$Sensibilidad\ ADC = \frac{V_{max} - V_{min}}{2^n - 1} = \frac{3.3 - 0}{2^{16} - 1} = 50.35477 * 10^{-6} V = 50.35477\ \mu V \tag{4.1}$$

Para acabar, el modo de conversión continuo que se puede leer en la Figura B.1, viene a referirse a que el ADC se configura para muestrear una única señal de entrada de manera constante desde el momento en que se indica al ADC que comience a operar.

A la salida del conversor se ha colocado un bloque conocido como DMA (*Direct Memory Access*). Este se utiliza para transmitir datos desde o hacia la memoria RAM, los componentes internos del PSoC como lo son el propio ADC o registros. En este caso se utiliza para enviar los bytes de salida del conversor hacia el filtro digital.

4.1.2.2. Configuración: Filtros digitales

Se utilizan dos filtrados digitales. Ambos reciben como entrada la señal digitalizada a la salida del ADC, uno de ellos se emplea para la realimentación a través de la referencia y el otro para eliminar ruido de alta frecuencia de la señal que contiene al ECG previamente a transmitirla a través del módulo BLE. Identificamos respectivamente estos filtros como canal A y canal B. A continuación, se trata la configuración de cada uno, esta información también se puede consultar en las Figura B.2.

- **Filtro A**: Se configura como un filtro paso baja de Butterworth de orden 1, con una frecuencia de corte de 0.5 Hz. La salida de este filtro se envía al conversor digital-analógico de 8 bits, "VDAC8_1", de la Figura 4.6.

- **Filtro B**: Se trata de un filtro paso baja de Bessel de orden 4 y con frecuencia de corte de 35 Hz. La salida de este filtro se transmite mediante comunicación UART al módulo BLE para su envío.

A la salida del filtro se colocan dos DMAs, los cuales tienen el propósito de:

- **DMA2**: Es el encargado de mover las muestras filtradas extraídas del canal A del filtro al conversor digital-analógico. Este valor es el que indica el nivel de tensión DC a generar en el convertidor.
- **DMA3**: Se emplea en enviar las muestras a memoria RAM para que se almacenen en vectores desde los que poder operar con ellas. Este DMA tiene una interrupción asociada en el punto "nrq", de forma que cuando envíe el número indicado de bytes, en este caso 4000 bytes, se genera una interrupción. Esto da lugar a que se llene un vector de 2000 posiciones de 2 bytes cada una.

4.1.2.3. Configuración: UART

Como ya se ha comentado, para poder transmitir la información de cada muestra tomada en dos bloques distintos, se ha configurado que el UART transmita grupos de 8 bits con una velocidad de 115200 bits/segundo. Dado que el mismo bloque se emplean para la transmisión y recepción de información, se ha seleccionado el modo de operación *Full UART*. En caso de usarse el modo *Half Duplex* también se podrían realizar ambas operaciones aunque con menos recursos disponibles, con lo cual habría que transmitir la información en paquetes de 4 bits [58].

La entrada del bloque UART "rx_interrupt" permite que salte una interrupción cuando se reciban instrucción desde la aplicación. La pestaña de configuración del bloque UART se puede encontrar también en la Figura B.3.

4.1.3. Electrodos usados para la adquisición de la señal

Los parches que se han utilizado para la adquisición de la señal en bruto del ECG son el modelo "BlueSensor L" fabricados por Ambu. Estos sensores de un único uso cuentan con gel conductor para la toma adecuada de la señal y conectores que permiten llevar esta hacia el dispositivo de procesamiento. En cuanto a las dimensiones, el área de contacto con la piel tiene un diámetro de aproximadamente 55 mm [120].

Figura 4.7: Sensor BlueSensor L usado para la adquisición del ECG [120].

4.2. Obtención del nivel de saturación de oxigeno

Se ha desarrollado un kit de expansión para la obtención del nivel de oxígeno en sangre mediante técnicas de fotopletismografía. En esta parte del capítulo se estudia que es la fotopletismografía y se trata el diseño y la construcción de un pulsioxímetro que permita la adquisición de la saturación de oxígeno en sangre y del ritmo cardíaco. Aunque en el apartado Campos de aplicación y parámetros extraíbles de manera no invasiva del Capítulo 1 ya se hizo una breve mención de la fotopletismografía o PPG (photoplethysmography), en esta sección se tratan en más profundidad los conceptos relacionados con esta técnica no invasiva.

4.2.1. Fundamento teórico del la fotopletismografía

Cuando se transmite una luz a través de un tejido blando del cuerpo humano, una parte se absorberá, otra se verá reflejada y el restante atravesará la zona incidida. El PPG consiste en, mediante un fotodetector, medir la intensidad de luz obtenida, a partir de este valor se puede extraer, por ejemplo, la saturación de oxigeno en sangre o el ritmo cardíaco que son los parámetros que nos resultan de interés en este trabajo. Según la zona del cuerpo donde se vaya a tomar la medida, se busca adquirir o bien la cantidad reflejada o bien la que se ha transmitido a través del tejido. En el caso de que la medida se tome en la punta de los dedos o el lóbulo de la oreja se adquiere la transmisión, mientras que en el resto de partes del cuerpo se mide la cantidad reflejada. En la Figura 4.8 se muestra como se colocarían el transmisor y el receptor según si se adquiere la transmisión o la reflexión. En este proyecto se tiene especial interés en usar técnicas para la adquisición de la reflexión, dado que se pretende tomar la medida en las muñecas, tal y como suelen hacer los dispositivos de actividad comerciales. Para la transmisión se suelen emplear LEDs (*light emitting diodes*) y para la recepción fotodiodos [24].

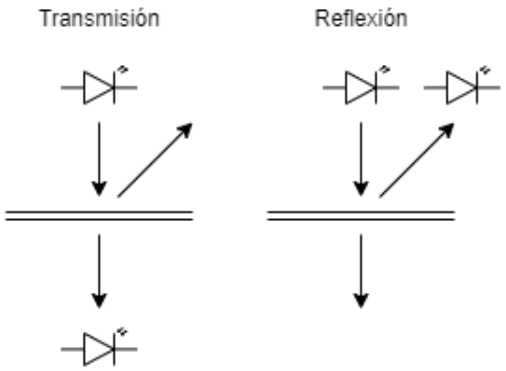


Figura 4.8: Colocación del fotodiodo usado a medo de receptor según si se quiere medir la cantidad de luz transmitida o reflejada.

Aquella longitud de onda para la cual la intensidad del LED es máxima se identifica como la longitud de onda de pico (λ_p). Según el parámetro del cuerpo a medir se seleccionan una o varias longitudes de onda diferentes. En los apartados siguientes se tratan los pormenores de las técnicas para adquisición del ritmo cardíaco y de la saturación de oxigeno.

4.2.1.1. Medida de la saturación de oxigeno en sangre

Cuando se desea conocer el nivel de saturación de oxígeno en sangre se utiliza la técnica conocida como pulsioximetría. En este método se emplean dos LEDs, uno de ellos emitiendo a 660 nm (luz roja) y otro a 940 nm (infrarrojo) y se va alternando de manera secuencial el uso de uno u otro [24], esto viene a referirse a que se tiene que producir una multiplexación entre las dos luces incidentes, ya que no es posible obtener medidas correctas con ambas en funcionamiento de manera simultanea. La utilización de dos longitudes de onda se debe a que tal y como se muestra en la Figura 4.9, la absorción lumínica de la sangre varía según si se encuentra oxigenada o no. La hemoglobina oxigenada (HbO_2) tiene una menor absorción de luz para longitudes de onda correspondientes a luz roja, mientras que es mayor para longitudes de onda del rango de los infrarrojos.

Por otra parte, la propiedades de absorción lumínica de la hemoglobina desoxigenada (Hb) se comportan de manera inversa. El distinto comportamiento respecto a la absorción lumínica de ambos tipos de hemoglobina se puede aprovechar en el cálculo de la cantidad de HbO_2 o Hb presentes [24].

La variación en las propiedades de absorción de ambas clases de hemoglobina, hace que las medidas oscilen durante la sístole y la diástole del corazón, con lo cual se dice que la medida tiene una componente de alterna (AC) y otra de continua (DC). La componente DC de la medida se deberá a la presencia de músculo, huesos, grasa, etc y aunque es relativamente constante, sufre pequeñas variaciones debido a, por poner un ejemplo, la propia respiración del paciente entre otras razones. El componente AC se produce a causa de la sangre que bombea a través de las arterias durante la sístole, ya que el incremento en el volumen de sangre modifica la absorción lumínica. La fase de subida de la componente AC se produce cuando se da la sístole, mientras que la fase de bajada es causada por la diástole [24, 122]. En la Figura 4.10 se puede encontrar un gráfico que muestra la variación en la absorción lumínica según el bombeo del corazón, mientras que en la Figura 4.11 se haya un ejemplo teórico de la señal obtenida para una de las luces, donde se pueden apreciar las componentes DC y AC.

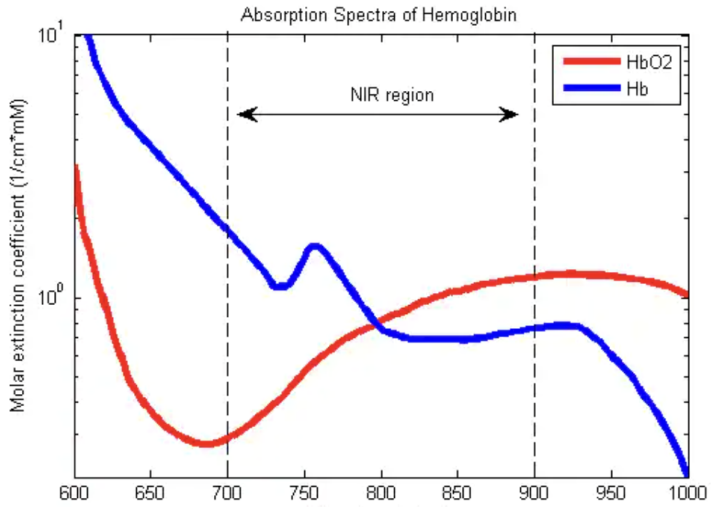


Figura 4.9: Gráfico de la absorción espectral de la hemoglobina en el caso de sangre oxigenada (rojo) y desoxigenada (azul). Créditos: [124].

Para el cálculo del porcentaje del nivel de saturación de oxígeno arterial (SaO_2) se utiliza la ecuación 4.2, donde $[HbO_2]$ es la concentración de hemoglobina oxigenada y $[HHb]$ es la concentración de la desoxigenada. Sin embargo, este método precisas de análisis realizados en laboratorio [24].

$$SaO_2 = \frac{[HbO_2]}{[HbO_2] + [HHb]} \tag{4.2}$$

Alternativamente, se puede utilizar el índice de absorbancia (R), el cual también se puede encontrar referido en la bibliografía como el "ratio de ratios" y cuya expresión se encuentra en la ecuación 4.3. PPG_{DC} hace referencia a la componente continua y PPG_{AC} a la de alterna. Adicionalmente, *Red* es la onda de luz roja e IR (*infrared*) la infrarroja. Posteriormente, se debe de generar una curva de calibración para obtener la relación entre el valor R del índice de absorbancia y el porcentaje de oxígeno [123, 125]. Es posible generar una curva de calibración propia a partir de muestras obtenidas de pacientes sanos para corresponder el valor de R con un porcentaje de SpO_2, sin embargo, también existe una curva de calibración estándar definida mediante la expresión 4.4 [24, 123]. Hay que tener en cuenta que la ecuación 4.4 ayuda a calcular la saturación de oxígeno periférico (SpO_2), la cual es una cuantificación del nivel total de oxigenación [123].

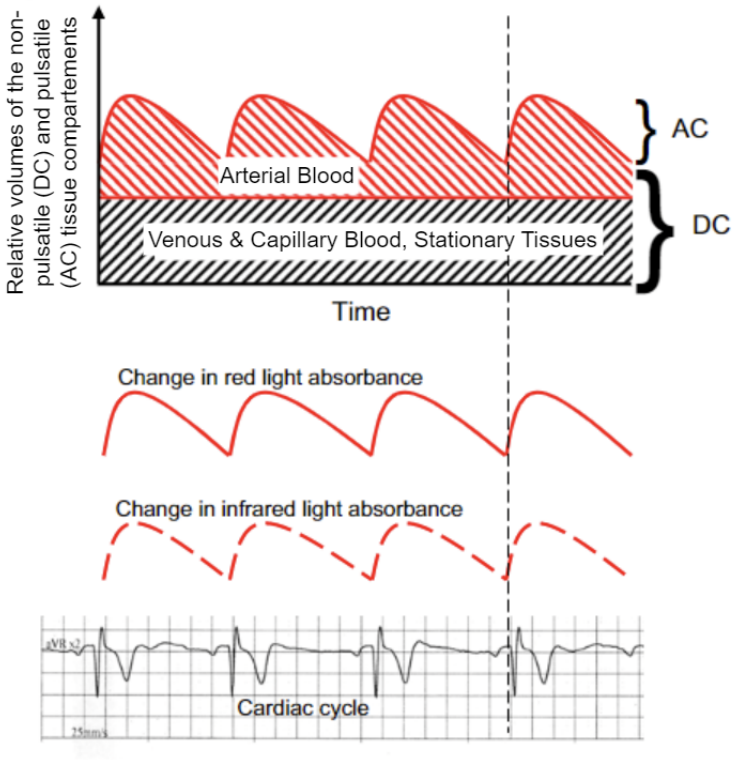


Figura 4.10: Cambios en la absorción lumínica debido a la sístole y diástole del corazón. Créditos: [122].

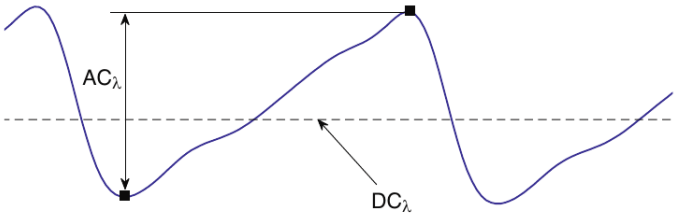


Figura 4.11: Ejemplo de la señal obtenida tras realizar un PPG. Se obtendrían dos de esta señales, una para la frecuencias de luz roja y otra para infrarrojos. Créditos: [24].

En este trabajo se ha optado por emplear la curva estándar, debido a la inviabilidad de realizar un estudio y posterior calibración con pacientes reales. En la Figura 4.12 se ha representado la curva de calibración a partir de la expresión 4.4.

$$R = (\frac{PPG_{AC,Red}}{PPG_{DC,Red}})/(\frac{PPG_{AC,IR}}{PPG_{DC,IR}}) \tag{4.3}$$

$$SpO_2 = 110 - 25R \tag{4.4}$$

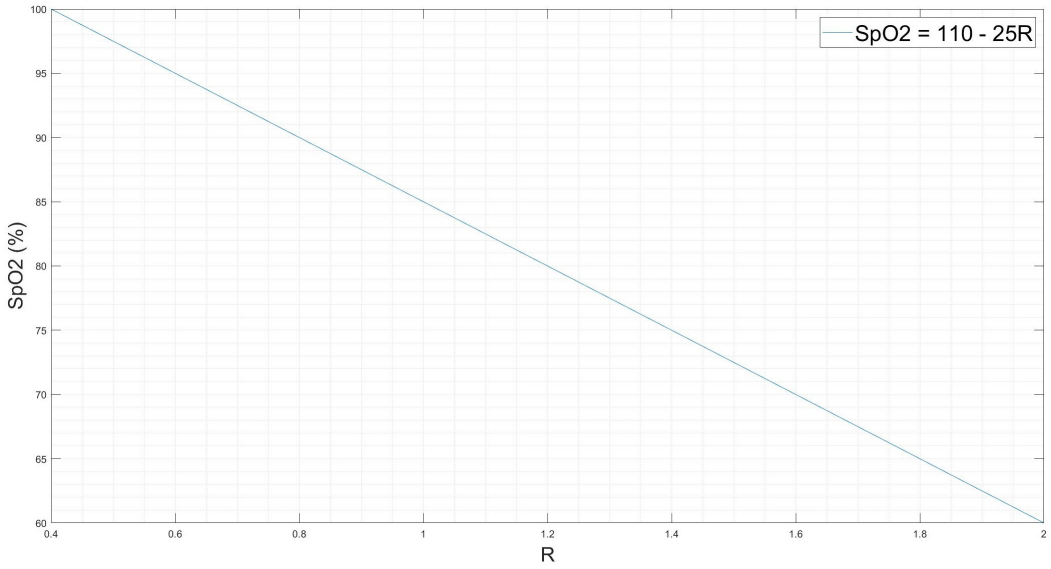


Figura 4.12: Curva de calibración estándar de la pulsioximetría, la cual representa el porcentaje de SpO_2 en función de el índice de absorbancia (R).

4.2.1.2. Medida del ritmo cardíaco a partir del SpO_2

En caso de que se busque calcular el ritmo cardíaco mediante técnicas pletismográficas, se suele utilizar un LED que emita luz a una longitud de onda de pico de alrededor de unos 540 nm (luz verde). En este caso de estudio, ya que ya se está obteniendo la sístole y diástole del corazón en la señal mostrada en la Figura 4.10, se podría reutilizar una de las señales de alterna obtenidas o bien mediante la luz roja o bien mediante la infrarroja y contar el periodo entre picos de la misma polaridad. En [125] se recomienda usar la señal extraída de la luz infrarroja en caso de que se desee aprovechar esta técnica para el cálculo simultaneo del ritmo cardíaco.

4.2.2. Sistema para la adquisición del nivel de SpO_2

En función de los conceptos teóricos tratados en el apartado Fundamento teórico del la fotopletismografía de este capítulo, se va a diseñar un sistema que permita el cálculo del nivel de saturación de oxígeno periférico (SpO_2).

Para la construcción del pulsioxímetro se necesitan dos subsistema. Por una parte, se tiene uno que controla la multiplexación de las luces incidentes mediante el control del apagado y encendido de los LEDs. Por otra parte, se necesita otro que incluya la recepción de la luz reflejada, y que posteriormente haga el filtrado y muestreo de las señales producidas desde los fotodiodos usados como receptores. Hay que tener en cuenta que los fotodiodos actúan como generadores de corriente cuando tienen luz incidente, por este motivo, se necesita hacer una conversión de intensidad a voltaje (I-V) antes de empezar a poder tratar la señal. A modo de conversor se emplea un amplificador de transimpedancia (TIA).

En la bibliografía [24, 126] se puede encontrar recomendado el uso de un sistema de filtrado analógico. Este consiste en, a continuación del TIA, separar las componentes continua y alterna de la señal de la pulsioximetría mediante el uso de dos filtros analógicos tipo Sallen-Key de segundo orden. En primer lugar se coloca un filtro paso baja (FPB) para eliminar componentes de alta frecuencia no deseados, se debe de elegir una frecuencia de corte para el filtro de manera que no se elimine el ancho de banda que contiene la señal AC. El filtrado de componentes de alta frecuencia ayuda a obtener un mejor muestreo de la señal en el ADC, ya que en caso de que dichas frecuencias no se filtrasen, habría que incrementar la frecuencia de muestreo de manera que se siguiera cumpliendo el teorema de Nyquist. La componente de DC se puede muestrear sin previamente filtrar la componente de AC, esto se debe a que esta última es muy pequeña en comparación a la parte continua de la señal. El segundo filtro es un paso alta (FPA) que se emplea para desacoplar la

componente de DC de la de AC. Posteriormente a aislar la parte AC de la señal, se le hace una amplificación de manera que la señal alcance, idealmente, todo el rango del ADC. De esta manera se puede obtener el muestreo más preciso posible. El hecho de obtener las dos componentes de la señal por separado es que, como se puede consultar en la expresión 4.3, es necesario conocer ambas para el cálculo del ratio de ratios R.

En este trabajo se descarta esta solución debido a que, para este caso concreto, no se hace un uso adecuado de los recursos de los que se disponen en el PSoC 5LP. En su lugar, se ha optado por un diseño basado en la información disponible en [119], el cual hace uso del TIA y los filtros internos del propio PSoC, con lo cual se reduce en gran medida el *hardware* externo a añadir. En la Figura 4.13 se ha incluido un diagrama de bloques de la lógica que sigue el sistema completo incluyendo emisor y receptor.

En lugar de separar las componentes AC y DC mediante filtros analógicos, la parte continua de la señal se calcula como el promedio de las muestras extraídas por el ADC en un cierto intervalo, y la parte alterna como la media cuadrática o RMS (*Root Mean Square*) de la señal restándole la parte DC [125]. En la expresiones 4.5 y 4.6 se pueden consultar las formulas para el cálculo de ambos parámetros, donde $m[i]$ hace referencia al vector de muestras extraído por el ADC y N al número total de muestras.

$$PPG_{DC} = \frac{1}{N}\sum_{i=0}^{N} m[i] \tag{4.5}$$

$$PPG_{AC} = \sqrt{\frac{1}{N}\sum_{i=0}^{N}(m[i] - PPG_{DC})^2} \tag{4.6}$$

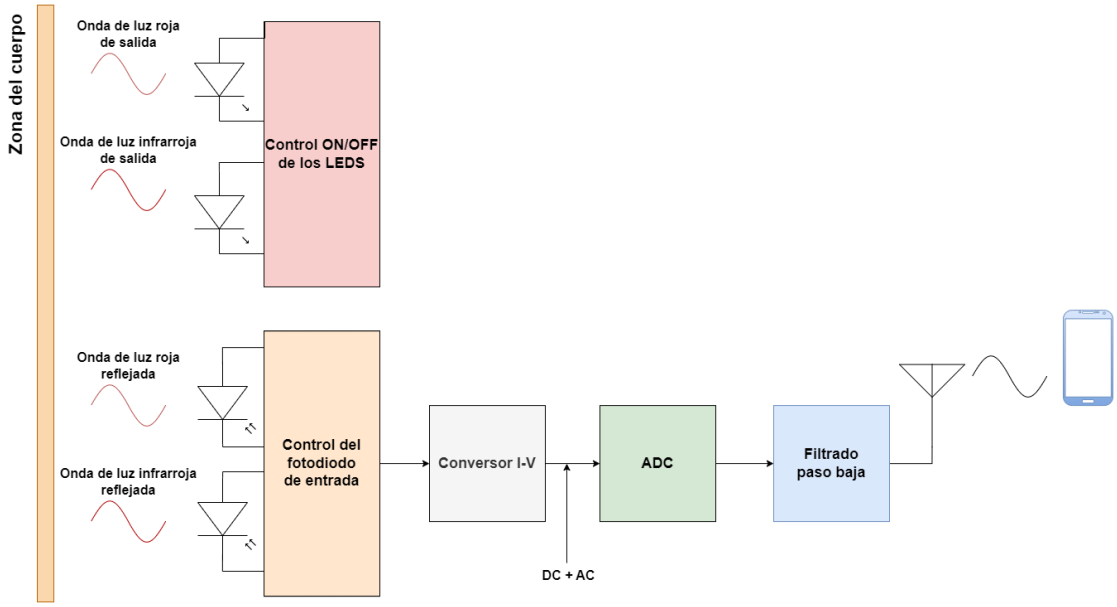


Figura 4.13: Diagrama de bloques del pulsioxímetro. Imagen creada a partir de la información disponible en [126].

Como se ha tratado en el capítulo Capítulo 2: Especificaciones del dispositivo de procesamiento central, el dispositivo diseñado en este proyecto no incluye los LEDs y y fotodiodos necesarios para realizar la medida de la saturación de oxígeno. Por este motivo, se conecta a los puertos de entrada el módulo **ILE-BI01-GGRIRICBD-SC201**, basado en el chip **SFH7072**.

Los únicos elementos analógicos que hay que añadir de manera externa a la PCB son el propio módulo que contiene a los fotodiodos y los LEDs y las resistencias de polarización de estos últimos.

Todo el resto de componentes, tales como el conversor de corriente a voltaje o el filtro, se pueden implementar con los componentes presentes en el interior del PSoC 5LP. En la Figura 4.14 se ha representado un diagrama con la conexión de los elementos externos a la PCB. Como se puede ver, se necesitan emplear dos puertos de entrada, uno para la alimentación y control sobre los LEDs y el otro para la entrada de las corrientes del fotodiodo. Normalmente se suele emplear un fotodiodo con longitud máxima de detección igual a la longitud de onda de pico de uno de los LEDs, con lo cual se tiene un fotodiodo por cada LED. Como se trata en la subsección SFH7072, el módulo usado cuenta con un fotodiodo de banda ancha, con lo cual para este caso concreto sólo se emplea un único fotodiodo.

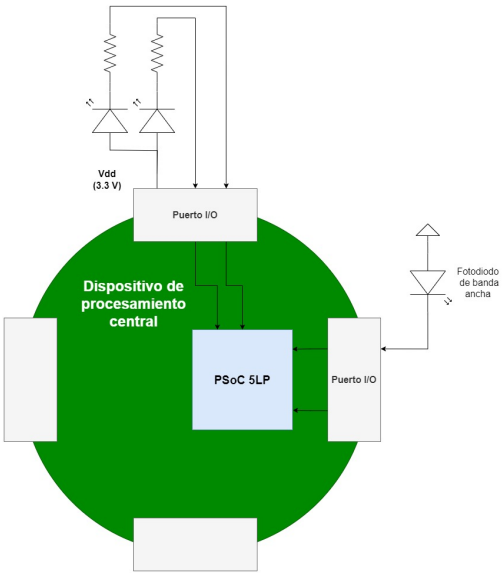


Figura 4.14: Diagrama de la conexión del *hardware* externo necesario para pulsioximetría a la PCB del dispositivo de procesamiento central.

La Figura 4.15 incluye los esquemas creados en PSoC Creator para el desarrollo del *firmware* que realiza la medida del porcentaje de oxígeno en sangre. De manera genera, se sigue la siguiente lógica para la recepción de la corriente proveniente del fotodiodo de banda ancha:

1. Desde una entrada analógica del PSoC 5LP, se introduce la corriente generada en el fotodiodo al conversor de corriente a voltaje.

2. Se muestrea el nivel de tensión mediante un conversor Delta-Sigma con una resolución de 16 bits a una frecuencia de 2006 muestras/segundo.

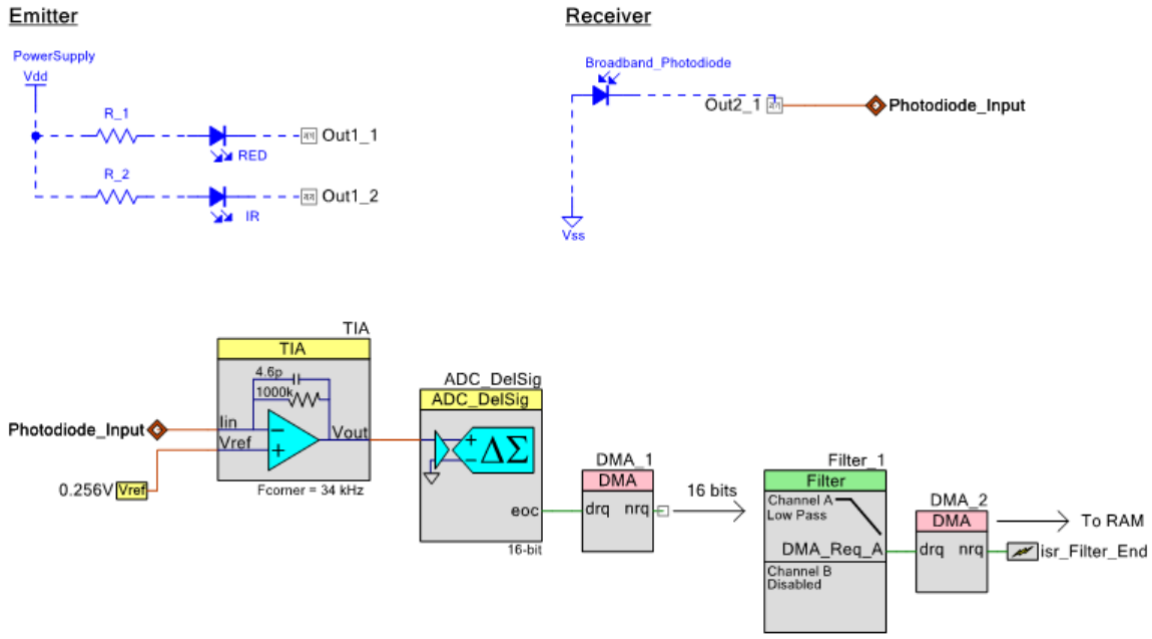
3. Las muestras digitalizadas en el ADC se envían a un filtro digital y se aplica sobre ellas un filtrado paso baja para eliminar componentes frecuenciales indeseadas.

4. Las muestras extraídas a la salida del filtro, se envía a la memoria RAM, donde se calcula el ratio de absorción R y a partir de él se obtiene el nivel de oxígeno en sangre. Este último valor se envía mediante comunicación UART al módulo BLE o al puerto serie de un ordenador.

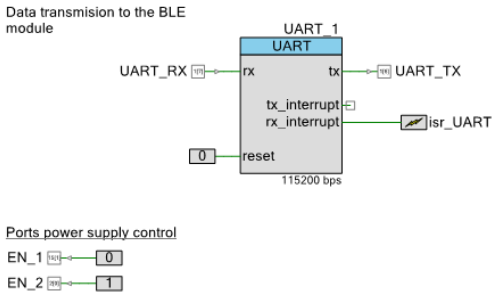
La configuración individual de cada bloque se ha incluido en el Anexo Configuración de los módulos en PSoC Creator. Por otra parte, el código C desarrollado para esta aplicación se puede localizar en el Anexo Firmware usado en el PSoC 5LP.

Cuando el sistema recibe el carácter “a” de manera externa desde el UART, inicia la adquisición y transmisión de resultados de manera constante, no deteniéndose hasta que reciba el carácter “s”. El sistema calcula un valor de SpO_2 cada 8 segundos, ya que se adquieren mediciones durante 4 segundos con cada uno de los LEDs antes de calcular el porcentaje de saturación.

Para el control de que LED se encuentra encendido en cada momento, se emplean dos pines configurados como digitales. En caso de que se quiera encender un LED, el pin se establece a nivel de tensión bajo, con lo cual el LED queda polarizado al existir suficiente diferencia de tensión entre su ánodo y cátodo. Si por otra parte se quiere apagar, se configura el pin a nivel de tensión alto, con lo cual se despolariza y no emite luz al ser la diferencia de tensión entre sus patillas insuficiente. Adicionalmente, es necesario incluir resistencias de polarización para cada LED, dado que de lo contrarío, estos se dañarían al intentar alimentarlos. En la subsección SFH7072 se ha tratado este asunto.



(a) Página 1.



(b) Página 2.

Figura 4.15: Esquemático de PSoC Creator para la interconexión de los componentes que implementan la adquisición del nivel de saturación de oxígeno en sangre. Nótese que los elementos resaltados en azul como los resistores o los diodos son sólo representativos para un mejor entendimiento del sistema y no se encuentran en el interior del PSoC 5LP.

4.2.2.1. SFH7072

El SFH7072 es un chip que incluye todos los LEDs y fotodiodos necesarios para la adquisición del nivel de oxigeno en sangre y el ritmo cardíaco. Para ello, el integrado cuenta con dos fotodiodos, uno de banda ancha (530-940 nm) y otro con una sensibilidad máxima para longitudes de 635 nm. También incluye tres LEDs, los cuales emiten a longitudes de luz verde (530 nm), roja (655 nm) e infrarroja (940 nm) [127].

Se utilizan los LEDs rojo e IR y el fotodiodo de banda ancha, ya que su rango de operación incluye las dos longitudes de onda de interés. En la Figura 4.16 se puede consultar la corriente generada por dicho fotodiodo según la intensidad y la longitud de onda de la luz incidente. De manera típica, si la luz incidente es roja la corriente generada tiene una intensidad de 0.6 μA, mientras que si esta es IR la intensidad es de 1.1 μA.

El LED rojo necesita una diferencia de tensión de 2.1 V entre su ánodo y cátodo para estar polarizado y es capaz de resistir hasta 40 mA de corriente, mientras que el infrarrojo necesita una diferencia 1.3 V y resiste hasta 60 mA. De la hoja técnica del módulo [127] se extrae que las corrientes que atraviesan a los LEDs para el comportamiento típico, son de 20 mA. Se ha tratado de elegir resistencias de polarización que fuercen una corriente de entorno a este valor. Dado que durante las pruebas que se ha realizado se ha alimentado al SFH7072 con una fuente de 5 V, cada uno de los LEDs **se polariza mediante una resistencia de 220** Ω en serie, como se ha representado en la Figura 4.15. Esto proporciona una corriente algo menor a 20 mA, con lo cual los LEDs lucirán con menor intensidad, aunque es más que suficiente para adquirir las medidas.

Para el prototipo construido en este trabajado, el integrado SFH7072 se encuentra montado sobre el módulo ILE-BI01-GGRIRICBD-SC201. Este último no es más que un soporte que permite acceder a los pines del SFH7072 mediante pines macho-macho o macho-hembra. En la Figura 4.17 se ha incluido una imagen de este módulo.

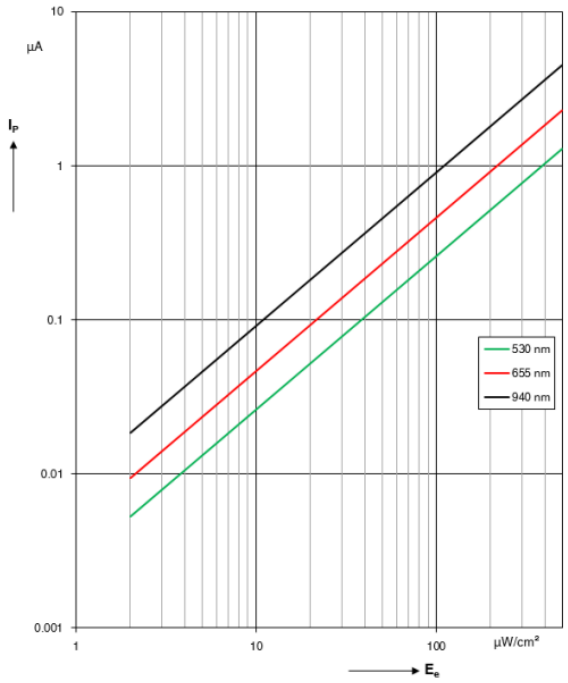


Figura 4.16: Corrientes generadas por el fotodiodo de banda ancha del SFH7072 según la intensidad y longitud de onda de la luz incidente. Créditos: [127].

Figura 4.17: Módulo ILE-BI01-GGRIRICBD-SC201, en el centro de él se encuentra el chip SFH7072. Créditos: [128].

4.2.2.2. Configuración: Conversor V-I

Para la conversión de la corriente generada en el fotodiodo a una señal de tensión, se una un amplificador de transimpedancia o TIA (*Transimpedance Amplifier*). Se tiene una resistencia (R_f) que es la que permite el control de la transformación a voltaje mediante la relación:

$$V_{out} = I_f * R_f \tag{4.7}$$

Donde I_f representa la corriente emitida desde el fotodiodo (fotocorriente) cuando este recibe luz incidente en un momento determinado y cuya polaridad depende de la orientación del fotodiodo. Hay que tener en cuenta que los fotodiodos reales no se comportan como fuentes de corriente ideales, si no que tienen una capacitancia parásita de valor C_t como se muestra en la Figura 4.18. Es importante compensar este elemento reactivo para logra un correcto funcionamiento del TIA, para lo cual se añade un condensador (C_f) en paralelo a la resistencia R_f. En caso de no seleccionar el valor del condensador de compensación de manera adecuada, se puede dar el caso de que el circuito oscile o sea más ruidoso [24, 129].

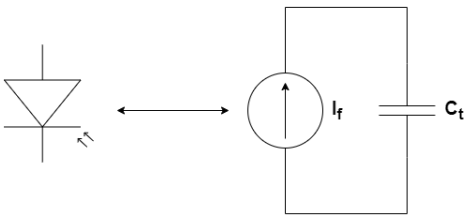


Figura 4.18: Sistema equivalente de un fotodiodo real.

En la Figura 4.19 se muestra el diagrama de Bode ganancia del amplificador operacional en lazo abierto (curva A_{VOL}), es decir, cuando sólo se encuentra el amplificador sin más componentes presentes y el del sistema de realimentación con su función de transferencia invertida ($1/\beta$), donde este último es el sistema formado por el diodo, su capacitor parásito C_f y la resistencia R_f. El valor $1/\beta$ se mantiene aproximadamente constante a valor 1 a bajas frecuencias hasta que alcanza la "frecuencia de esquina" (f_{corner} o f_F), a partir de cuyo instante tiene un crecimiento en ganancia de 20 dB/dec. En la ecuación 4.8 se puede consultar la función de transferencia del sistema de realimentación [129].

$$\beta(s) = \frac{X_{Ct}}{Z_F + X_{Ct}} = \frac{1}{1 + sR_fC_t} \tag{4.8}$$

Cuanto mayor sea el ritmo al que las dos curvas se aproximen una a la otra, mayor será la inestabilidad del sistema. En el caso representado en la Figura 4.19, ambas curvas se aproximan a una velocidad de 40 dB/dec, lo que hace que el sistema sea inestable. El condensador de compensación (C_f) permite aplanar la respuesta de $1/\beta$, haciendo que el sistema se estabilice.

Se consigue una compensación óptima si el cero que añade C_f coincide con la frecuencia de corte entre las dos curvas (f_i o *intercept frequency*). Caso de que se use un condensador más grande de lo necesario, se dice que el sistema se encuentra sobrecompensado, en cuyo caso este sigue siendo estable, pero se reduce el ancho de banda del sistema. En la Figura 4.20 en caso de añadirse un condensador para estabilizar el sistema. Para el cálculo del valor del condensador óptimo, se puede recurrir a la expresión 4.9 [129]:

$$C_f = \frac{1 + \sqrt{1 + 8\pi R_f C_i f_{GBWP}}}{4\pi R_f f_{GBWP}} \tag{4.9}$$

Donde f_{GBWP} es el ancho de banda del amplificador operacional en ganancia unitaria, dicho de otra forma, la frecuencia a la que la respuesta en ganancia del amplificador sufre una caída de -3 dB. Por otra parte, C_i representa la capacidad parásita del fotodiodo (C_t) en paralelo con la capacidad de entrada del amplificador operacional [130].

$$C_i = C_t + C_{in(AmpOp)} \tag{4.10}$$

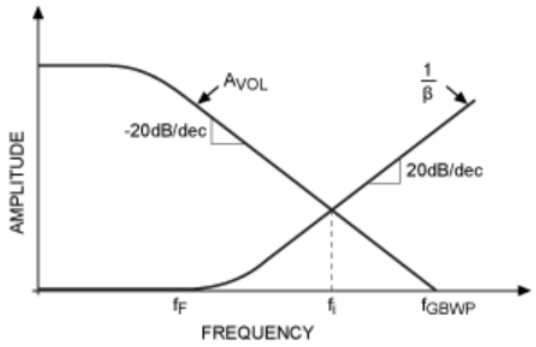


Figura 4.19: Diagrama de Bode de un amplificador de transimpedancia genérico sin condensador C_f que asegure la estabilidad del circuito. Créditos: [129].

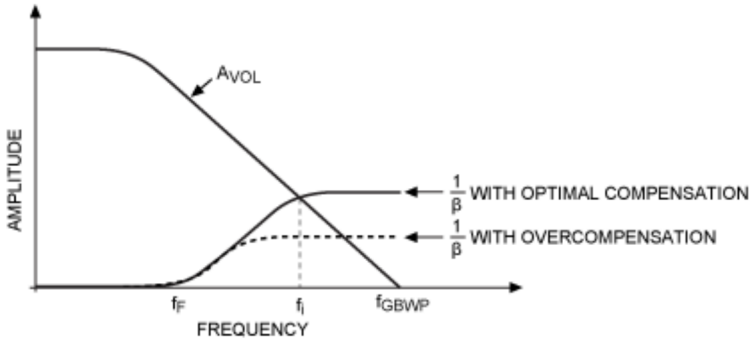


Figura 4.20: Diagrama de Bode de un amplificador de transimpedancia genérico con condensador de compensación (C_f). Créditos: [129].

En la Figura 4.21 se muestra la configuración de un amplificador de transimpedancia genérico. Nótese que también es posible utilizar este circuito invirtiendo la orientación del fotodiodo. En este caso se ha elegido esta orientación ya que el ADC utilizado sólo tiene resolución para valores positivos de tensión.

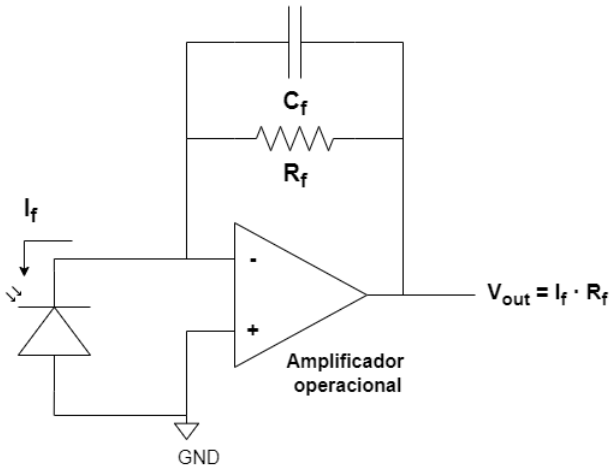


Figura 4.21: Esquema de un amplificador de transimpedancia genérico. El fotodiodo se encuentra orientado de manera que la tensión de salida sea positiva.

Dado que las frecuencias de interés en este caso son relativamente pequeñas, el ancho de banda no es un factor limitante, con lo cual no resulta crítico que el sistema se sobrecompense. Este factor, unido al hecho de que no se conoce el ancho de banda del amplificador integrado en el interior del PSoC 5LP, ha hecho que se elija el mayor valor de C_f disponible en el TIA, el cual viene a ser **4.6 pF**. Se ha escogido también el mayor valor de resistor R_f posible con la intención de cubrir el máximo rango del ADC, en este caso **1 MΩ** [131]. Con lo cual, en caso de que se recibiese la corriente proveniente del fotodiodo IR o rojo, en función del valor máximo de corriente que genera cada uno, la tensión máxima del TIA sería para ambos casos:

$$V_{out,IR} = I_f * R_f = 1.1\,\mu A * 10^6\,\Omega = 1.1\,V$$

$$V_{out,RED} = I_f * R_f = 0.6\,\mu A * 10^6\,\Omega = 0.6\,V \tag{4.11}$$

Mediante la entrada de referencia del bloque del TIA se ha introducido un *offset* de 0.256 V, ya que el ADC no tiene en resolución para valores de tensión cercanos a los límites de alimentación. Añadiendo este desplazamiento nos aseguramos de que todo el rango de valores pueda ser muestreado.

4.2.2.3. Configuración: ADC Delta-Sigma

Para el muestreo de los datos se ha utilizado un conversor analógico-digital Delta-Sigma, manteniendo la misma configuración que la empleada para la adquisición del ECG en el apartado Sistema para la adquisición del ECG de este mismo capítulo. Referirse a este apartado para consultar los pormenores de la configuración. Hay que tener en cuenta que nuevamente se ha sobremuestreado la señal, debido a que en esta ocasión lo que se va a transmitir mediante comunicación Bluetooth es únicamente el resultado final (R), no es necesario emplear ningún diezmado de los datos. A la salida del ADC se ha colocado un DMA para transmitir los bytes obtenidos al filtro digital.

4.2.2.4. Configuración: Filtro paso baja

En [24] se recomienda que la frecuencia de corte del filtrado paso baja se encuentre en el rango de los 10 a los 40 Hz. Se ha escogido un filtro Butterworth de orden 4 con frecuencia de corte de 10 Hz.

La salida del filtro se envía a la memoria RAM mediante un DMA para hacer los cálculos correspondientes a partir de las muestras. Además, cada vez que el filtro envía una cierta cantidad de bits, se activa una interrupción. Conociendo la frecuencia de muestreo se puede extraer que tiempo ha transcurrido a partir de la cantidad de bytes transmitidos. Se utiliza este procedimiento para

que al transcurrir 4 segundos se permuten los LEDs encendido y apagado, estando en funcionamiento el sistema 8 segundos en total aproximadamente. Se ha configurado que la interrupción se produzca cada vez que se envíen 4000 bytes, teniendo en cuenta que el ADC obtienen las muestras con una precisión de 16 bits (2 bytes) a una frecuencia de 2000 muestras/segundo, esto supone que la interrupción se produce aproximadamente cada 1 segundo.

4.2.2.5. Compensación de la fluctuación en las medidas

Varias medidas seguidas en la misma zona del dedo en un instante temporal dado, pueden dar como resultado un valor de SpO_2 que oscila entorno a un valor central, lo cual se puede deber a factores tales como ruido externo, el artefacto del movimiento o incluso la presión ejercida sobre el propio sensor [132]. Para compensar este hecho, se añade a la salida una realimentación de los resultados previos utilizando la siguiente expresión:

$$R = 0,8 * R_{Actual} + 0,2 * R_{Anterior} \tag{4.12}$$

Donde R_{Actual} es el ratio de absorción que se ha obtenido en el instante temporal actual y $R_{Anterior}$ es el ratio que se obtuvo en la medida previa. Esto hace que hasta que no se han calculado varios valores de R, la medida no se estabilice.

Capítulo 5

Diseño de aplicación compatible en Android

Se ha diseñado una aplicación para *smartphones* basados en el sistema operativo Android. El objetivo de esta es controlar el funcionamiento de la parte física de proyecto, en concreto cuando se toman una nueva adquisición. Además de mostrar los resultados obtenidos y almacenarlos para poder ser consultados en el futuro u operar con ellos desde otro *software* como lo puede ser MATLAB o incluso Python. La aplicación en sí está programada en el lenguaje Java.

La aplicación esta enfocada para su uso en la adquisición de la señal del corazón mediante un ECG, con lo cual todos los métodos diseñados están enfocados a esta medida en específico. En este capítulo se puede encontrar la información relevante a dicha app, habiéndose organizado esta de la siguiente manera:

- El diagrama se flujo que sigue la aplicación se ha comentado en Diagrama de flujo de la aplicación.
- En la sección Actividades y apartado gráfico se trata como se presenta la aplicación de manera gráfica al usuario, las actividades Android que se han incluido y el manifiesto de la app.
- En Código Java se discute el código fuente de la aplicación. Aquí se puede encontrar en que archivos se ha dividido el código java y un resumen de algunos de los métodos más importantes. En este capítulo se han incluido únicamente algunas secciones del código Java, en el Anexo D: Código fuente de la aplicación Android se puede encontrar el código fuente completo.

5.1. Diagrama de flujo de la aplicación

En la Figura 5.1 se puede consultar el diagrama de flujo seguido por la aplicación. En primer lugar, se verifica que el dispositivo sobre el que está instalada la app es compatible con el Bluetooth LE y que el usuario ha concedido todos los permisos necesarios para el funcionamiento de la misma.

Cuando se presione el botón para acceder a la acción de conectarse a un dispositivo, se mostrará una lista con todos los equipos con conexión BLE presentes al alcance. Hay que tener en cuenta, que sólo es posible establecer conexión con el dispositivo físico desarrollado en este trabajo. No se logrará habilitar una comunicación caso de tratar de usar la app con un sistema distinto.

Cuando se consigue la comunicación, se le indica al usuario mediante un mensaje en el parte superior de su pantalla. En este caso se muestra la ventana donde se pueden solicitar adquisiciones y consultar la última recibida. Si no se solicita una adquisición, el sistema se encuentra a la espera en esta pantalla. Cuando se solicite una medida, el sistema no mostrará el resultado hasta que haya pasado todo tiempo de adquisición configurado en el dispositivo físico.

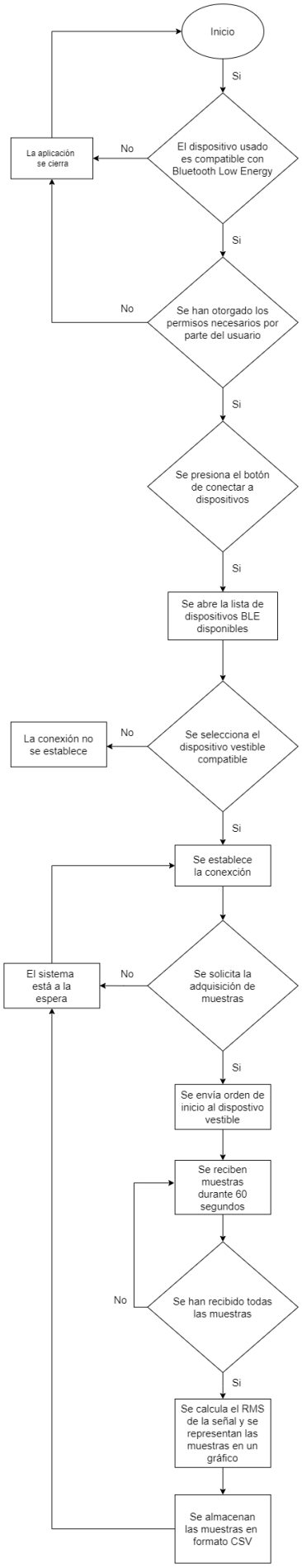


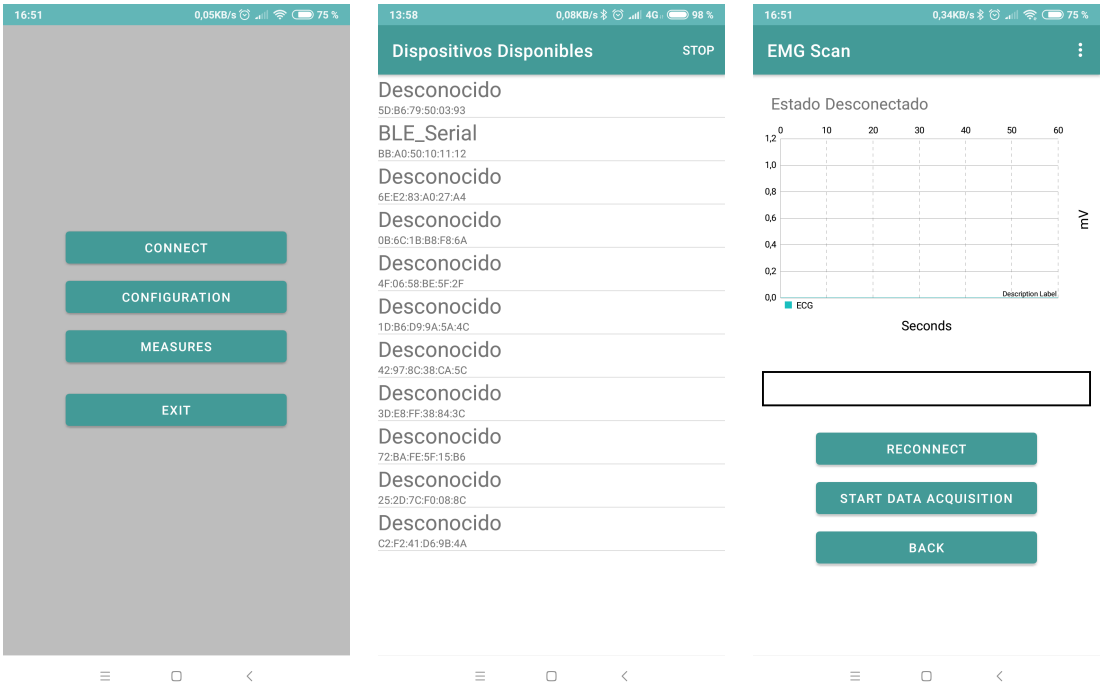
Figura 5.1: Diagrama de flujo de la aplicación desarrollada en Android.

5.2. Actividades y apartado gráfico

En este apartado se pueden consultar las distintas actividades que se han diseñado para la aplicación. En Android las acciones que el usuario puede realizar con una aplicación se dividen en diversas ventanas, que pueden o bien ocupar la pantalla completa o bien permanecer flotantes sobre otra. Podríamos definir en este contexto una actividad como la ventana sobre la cual la aplicación dibuja una interfaz gráfica para un funcionalidad concreta. Por poner un ejemplo, la actividad principal es la ventana que el usuario visualiza cuando accede por primera vez a la aplicación, pudiéndose tratar esto del menú principal o similar [133].

Se han diseñado tres actividades, la primera es el menú principal de la aplicación, el cual permite acceder al resto de actividades o cerrar la app. La segunda se utiliza para escanear los dispositivos disponibles y solicitar conexión al indicado. Para acabar, se tiene una ventana que permite visualizar los resultados provenientes dispositivo vestible y ordenar que se tomen nuevas medidas. Estas tres se pueden consultar en la Figura 5.2.

Para configurar el apartado gráfico de la aplicación, y con esto nos venimos a referir a la colocación de pulsadores y otros elementos así como el aspecto visual de estos, se emplean archivos de configuración en formato XML. El qué y cómo se deben de mostrar estas pestañas se ha desarrollado de manera gráfica, y la información relativa a cada caso ha quedado almacenada en tres archivos XML generados automáticamente por Android Studio. En caso de desearse, también se pueden editar directamente el texto de estos archivos para modificar aspecto visuales de la app. A continuación, se muestra como ejemplo el texto descriptivo XML generado para configurar el menú principal, caso de que se desee se pueden encontrar todos los demás en los anexos del trabajo.



(a) Menú principal.

(b) Listado de dispositivos Bluetooth a los que solicitar conexión. El dispositivo denominado "BLE_Serial" es el desarrollado en este trabajo.

(c) Actividad que muestra los resultados obtenidos en el dispositivo vestible y que permite solicitar la toma de nuevas medidas.

Figura 5.2: Menús y actividades de la aplicación diseñada con Android Studio

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/
    apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@color/divider"
        tools:context=".presentation.MainActivity">


        <Button
                android:id="@+id/main_menu_button_1"
                android:layout_width="250dp"
                android:layout_height="wrap_content"
                android:layout_marginTop="228dp"
                android:onClick="conectar"
                android:text="@string/connect"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintHorizontal_bias="0.497"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent" />

        <Button
                android:id="@+id/main_menu_button_2"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_marginTop="8dp"
                android:onClick="configuracion"
                android:text="@string/config"
                app:layout_constraintEnd_toEndOf="@+id/main_menu_button_1"
                app:layout_constraintHorizontal_bias="0.0"
                app:layout_constraintStart_toStartOf="@+id/main_menu_button_1"
                app:layout_constraintTop_toBottomOf="@+id/main_menu_button_1" />

        <Button
                android:id="@+id/main_menu_button_3"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_marginTop="8dp"
                android:layout_marginBottom="46dp"
                android:onClick="data"
                android:text="@string/extract_measures"
                app:layout_constraintBottom_toTopOf="@+id/main_menu_button_4"
                app:layout_constraintEnd_toEndOf="@+id/main_menu_button_2"
                app:layout_constraintHorizontal_bias="0.0"
                app:layout_constraintStart_toStartOf="@+id/main_menu_button_2"
                app:layout_constraintTop_toBottomOf="@+id/main_menu_button_2" />

        <Button
                android:id="@+id/main_menu_button_4"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_marginTop="24dp"
                android:onClick="salir"
                android:text="@string/button_exit"
                app:layout_constraintEnd_toEndOf="@+id/main_menu_button_3"
                app:layout_constraintStart_toStartOf="@+id/main_menu_button_3"
                app:layout_constraintTop_toBottomOf="@+id/main_menu_button_3" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Listing 5.1: Texto descriptivo XML para diseño de la actividad del menú principal.

5.2.1. Manifiesto

Cómo ya se describió en la sección Android Studio del Capítulo 1, las aplicaciones en Android incluyen un manifiesto, llamado "AndroidManifest.xml", necesario para declarar los permisos, las actividades, las funcionalidades *software* y *hardware* necesarias, etc.

Hay que tener en cuenta, que según la versión de Android sobre la que se instale la aplicación, la app se puede ver sometida a diferentes restricciones y por tanto, cambian los permisos a solicitar. Principalmente, se producen cambios de seguridad significativos a partir de Android 11 (nivel de API 30) respecto de versiones anteriores del SO. Se ha intentado que la aplicación sea, en la medida de lo posible, compatible con versiones anteriores y posteriores a Android 11 [134, 135]. A continuación, se muestra el manifiesto de la aplicación creada:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:tools="http://schemas.android.com/tools"
        xmlns:android="http://schemas.android.com/apk/res/android">

        <uses-permission android:name="android.permission.BLUETOOTH"
        android:maxSdkVersion="30"/>
        <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"
        android:maxSdkVersion="30"/>
        <uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
        <uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
        <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
        <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
        <uses-permission android:name="android.permission.BLUETOOTH" android:maxSdkVersion="30"
            />
        <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
        <uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>

        <application
                android:allowBackup="true"
                android:icon="@mipmap/ic_launcher"
                android:label="@string/app_name"
                android:roundIcon="@mipmap/ic_launcher_round"
                android:supportsRtl="true"
                android:theme="@style/Theme.FisoApp">

                <service
                        android:name=".services.BluetoothLeService"
                        android:enabled="true"
                        android:exported="true" />
                <activity
                        android:name=".presentation.ScanActivity"
                        android:label="Dispositivos Disponibles"/>
                <activity
                        android:name=".presentation.ConnectedActivity"
                        android:label="EMG Scan"/>
                <activity
                        android:name=".presentation.MainActivity"
                        android:exported="true"
                        android:label="@string/app_name"
                        android:theme="@style/Theme.FisoApp.NoActionBar">
                        <intent-filter>
                                <action android:name="android.intent.action.MAIN" />
                                <category android:name="android.intent.category.LAUNCHER" />
                        </intent-filter>
                </activity>
        </application>
</manifest>
```

Listing 5.2: Manifiesto de la aplicación.

5.3. Código Java

El código fuente de la aplicación, escrito en el lenguaje Java, se ha dividido en varios ficheros. En esta sección se va a tratar que funcionalidad implementa cada uno de ellos.

- Acquisition.java
- GattAttributes.java
- ConnectedActivity.java
- MainActivity.java
- ScanActivity.java
- BluetoothLeService.java

"GattAttributes.java" guarda los UUID (*Universally Unique Identifier*) o identificador único universal) del servicio del dispositivo vestible para acceder fácilmente a ellos. Por otra parte, "BluetoothLeService" es un servicio que gestiona la comunicación Bluetooth LE. Los archivos "ConnectedActivity.java", "MainActivity.java" y "ScanActivity.java" sintetizan las tres actividades representadas en la Figura 5.2. En los siguientes subapartados se estudian estos tres con más detalle.

5.3.1. MainActivity.java

Este archivo se utiliza para configurar la actividad del menú principal. Para empezar, hay que asegurarse de que el dispositivo móvil usado puede utilizar la tecnología de red de área Bluetooth Low Energy. En caso de que no sea así, se notifica este hecho al usuario y la aplicación se cierra, ya que no sería posible establecer comunicación con el dispositivo de procesamiento central.

Caso de que el *smartphone* usado sí sea compatible con la comunicación BLE, se incluyen métodos para configurar los botones del menú principal de la aplicación. Dichos pulsadores, se emplean para: acceder a un menú de opciones (en la versión actual de la app dicho menú aún no se encuentra desarrollado), conectarse al dispositivo de procesamiento central, acceder directamente a la pantalla que mostraría las medidas, o para cerrar la aplicación.

```
1  public class MainActivity extends AppCompatActivity {
2
3          private BluetoothAdapter mBluetoothAdapter;
4          private static final int REQUEST_ENABLE_BT = 1;
5
6          @Override
7          protected void onCreate(Bundle savedInstanceState) {
8                  super.onCreate(savedInstanceState);
9                  setContentView(R.layout.activity_main);
10
11                 // Determine whether BLE is supported on the device.
12                 // Then you can selectively disable BLE-related features.
13                 if (!getPackageManager().hasSystemFeature(PackageManager.FEATURE_BLUETOOTH_LE))
                       {
14                        Toast.makeText(this, R.string.ble_not_supported, Toast.LENGTH_SHORT).
                              show();
15                        finish();
16                 }
17
18                 // Initializes a Bluetooth adapter. For API level 18 and above, get a reference
                       to
19                 // BluetoothAdapter through BluetoothManager.
20                 final BluetoothManager bluetoothManager =
21                 (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
22                 mBluetoothAdapter = bluetoothManager.getAdapter();
23
24                 // Checks if Bluetooth is supported on the device.
25                 if (mBluetoothAdapter == null) {
26                         Toast.makeText(this, R.string.ble_not_supported, Toast.LENGTH_SHORT).
                              show();
27                         finish();
28                 }
29
30         }
```

Listing 5.3: Método de MainActivity.java para comprobar si el dispostivo es compatible con Bluetooth Low Energy.

```
1  public void conectar(View view){
2          // Access to the connect to BLE devices menu
3
4          Intent intent = new Intent(this, ScanActivity.class);
5          startActivity(intent);
6          }
```

Listing 5.4: Método de MainActivity.java detectar si se ha activado el pulsador para acceder al menú de conexión con el dispositivo de procesamiento central.

5.3.2. Acquisition.java

En este archivo se han definido los métodos para el tratamiento de los paquetes de bytes recibidos que contienen las muestras adquiridas desde el dispositivo de procesamiento central. A parte de declararse los vectores donde se va a almacenar dichas muestras, se han creado métodos para:

- Almacenar los paquetes de bytes que se reciben en un único vector para su posterior tratamiento. El dispositivo de procesamiento central no envía todas las muestras que adquiere de una sola vez en un único paquete, si no que los va enviando en grupos de 4000 bytes hasta que se tiene un total de 120000 bytes, lo que supone aproximadamente 1 minuto de adquisición para el ECG debido a la frecuencia de muestreo. Se utiliza un método que va almacenando los grupos enviados en un solo vector. Hay que tener en cuenta que se tiene que guardar hasta que posición se ha llenado hasta el momento el vector, dado que de lo contrario se destruirían las muestras a medida que se adquiere la medida.
- Unificar las muestras de 1 bytes en muestras de 2 bytes. Como ya se vió en el Capítulo 4, las muestras se adquieren con una resolución de 16 bits (2 bytes), pero se tienen que separar en dos de 8 bits cada una debido a las limitaciones de la comunicación UART usada en el interior del dispositivo de procesamiento central. El objetivo de este método es reconstruir las muestras recibidas.
- Calcular el RMS de las señales, dado que para medidas tales como el ECG resulta de especial interés conocer este parámetro.
- Almacenar las muestras reconstruidas con 16 bits en un archivo en formato CSV (*Comma-separated values*) para que se pueda trabajar con ellas desde otro programa si así se desea. El formato CSV facilita el acceso a las muestras desde MATLAB, Python o incluso Excel. El nombre que se le asigna al archivo CSV que se genera, está formado por una cadena de texto de entrada a lo cual se le suma la fecha y hora en la cual se generó el fichero.
- Se incluyen varios métodos cuya utilidad es poder actualizar el valor de los parámetros globales del interior del archivo o extraer el contenido de estos.

A continuación, se muestran los métodos definidos para la reconstrucción de la muestras con 16 bits, el proceso de almacenar estas en formato CSV, el cálculo del RMS y el guardar los grupos de 4000 bytes enviados desde el módulo BLE en un único vector:

```
public void saveAcquisition(short[] data, String fileName) throws IOException {
        // Saves the data received in a .csv file

        // CSV file name
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy_MM_dd_HH_mm_ss");
        LocalDateTime now = LocalDateTime.now();
        String file_path = Environment.getExternalStoragePublicDirectory(Environment.
            DIRECTORY_DOWNLOADS)
        + "/" + fileName + dtf.format(now) + ".csv";
        File file = new File(file_path);

        CSVWriter writer;

        // If the file already exists -> Appends data
        if (file.exists() && !file.isDirectory()){
                FileWriter mFileWriter = new FileWriter(file_path, true);
                writer = new CSVWriter(mFileWriter, CSVWriter.DEFAULT_SEPARATOR,
                CSVWriter.NO_QUOTE_CHARACTER);
        }
        // If the file does not exists -> Creates it and adds data
        else {
                writer = new CSVWriter(new FileWriter(file_path), CSVWriter.DEFAULT_SEPARATOR,
                CSVWriter.NO_QUOTE_CHARACTER);
        }

        // Conversion: short[] -> String[]
        String[] data_string = new String[data.length];
        for (int i = 0; i < data.length; i++){
                data_string[i] = "" + (data[i] & 0xFFFF);
}

// Save data
```

```
writer.writeNext(data_string);
writer.close();
}


public double calculateRMS(short[] data){
        // Calculate the RMS from a data array

        double rms = 0;

        for (int i = 0; i < data.length; i++) {
                rms += data[i] * data[i];
        }
        rms = Math.sqrt(rms / data.length);
        return rms;
}


public void addPacket(byte[] Packet){
        // Stores the 1 byte data received in a single array

        int lastIndex = getLastIndex();
        byte[] data = getData();

        // Copy array "Packet" into array "data"
        System.arraycopy(Packet,0, data, lastIndex, Packet.length);
        lastIndex += Packet.length;
        if (lastIndex >= length_mData){
                lastIndex = 0;
                updateReadyToPlot(true); // The buffer is full
        }
        setData(data);
        setLastIndex(lastIndex);
}


public void adaptSamples(byte[] Samples){
        // Adapts samples from 8 bits to 16 bits

        short[] data16 = new short[Samples.length/2];

        for(int i = 0; i<(Samples.length/2); i++){
                data16[i] = (short) ((Samples[2*i] << 8) | Samples[2*i+1] & 0xFF);
        }

        setConvertedData(data16);
}
```

Listing 5.5: Métodos del fichero Acquisition.java.

5.3.3. ConnectedActivity.java

En este archivo se localiza el código que maneja la actividad que permite solicitar la adquisición al circuito y mostrar los resultados de la misma. De manera aclaratoria, esta es la actividad que se muestra en la Figura 5.2c. Aquí se implementa, por ejemplo, el leer el estado del pulsador, de manera que si se activa por parte del usuario se llame a los métodos adecuados para adquirir una nueva medida desde el dispositivo vestible.

El siguiente método de utiliza para el control de la comunicación. Cuando se cumple la última condición quiere decir que hay datos disponibles para transmitir. Lo que se hace en dicho caso es, en primer lugar, almacenar las muestras en un vector de tamaño igual que el número de muestras totales de 8 bits a transmitir desde el dispositivo vestible. En este caso la longitud es de 120000 posiciones debido a que el circuito adquiere muestras de 16 bits a una frecuencia de muestreo de 2000 muestras/segundo durante 60 segundos, teniendo en cuenta que cada muestra de 16 bits se divide en 2 de 8 bits.

Cuando el vector de 120000 posiciones se llena, se activa un *flag* que indica que las muestras están listas para ser almacenadas y representadas en la aplicación. Para ello, previamente se produce la reconstrucción de las muestras a 16 bits.

```
// Handles various events fired by the Service.
// ACTION_GATT_CONNECTED: connected to a GATT server.
// ACTION_GATT_DISCONNECTED: disconnected from a GATT server.
// ACTION_GATT_SERVICES_DISCOVERED: discovered GATT services.
// ACTION_DATA_AVAILABLE: received data from the device. This can be a result of read or
// notification operations.
private final BroadcastReceiver mGattUpdateReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
                final String action = intent.getAction();
                if (BluetoothLeService.ACTION_GATT_CONNECTED.equals(action)) {
                        mConnected = true;
                        updateConnectionState(R.string.connected);
                        ImageView view = findViewById(R.id.conn_image);
                        view.setVisibility(View.VISIBLE);
                        invalidateOptionsMenu();
                } else if (BluetoothLeService.ACTION_GATT_DISCONNECTED.equals(action)) {
                        mConnected = false;
                        updateConnectionState(R.string.disconnected);
                        ImageView view = findViewById(R.id.conn_image);
                        view.setVisibility(View.INVISIBLE);
                        invalidateOptionsMenu();
                } else if (BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED.equals(action)) {
                        // Show all the supported services and characteristics on the user
                             interface.
                        discoverGattServices(mBluetoothLeService.getSupportedGattServices());

                } else if (BluetoothLeService.ACTION_DATA_AVAILABLE.equals(action)) {
                        // Go here if there is info ready to be stored

                        byte[] data = intent.getByteArrayExtra(BluetoothLeService.EXTRA_DATA);
                        mAcquisition.addPacket(data);

                        // Plot and save the data if the buffer is full
                        if (mAcquisition.isReadyToPlot()){
                                mAcquisition.updateReadyToPlot(false);
                                mAcquisition.adaptSamples(mAcquisition.getData());
                                updateGraphics();
                                saveData();
                        }
                }
        }
};
```

Listing 5.6: Método para el control de la comunicación en la actividad.

A continuación, se pueden consultar el código para las funciones "saveData()" y "updateGraphics()" usadas para, respectivamente, almacenar las muestras reconvertidas a 16 bits en un fichero en formato CSV y para mostrar en pantalla la representación de dichas muestras y el valor RMS de la señal.

Hay que tener en cuenta que hay que el valor de amplitud que se obtiene para cada muestra, es un número que puede ir de 0 a 2^{16} - 1, siendo este el número de posibles valores distintos que puede ver el ADC. En el método "setData()" se hace un conversión al valor de voltaje que representa cada muestra. Para ello, se sigue la siguiente expresión:

$$Amplitud\ (mV) = (\frac{Amplitud_{ADC} * V^{+}}{2^{N} - 1} + V^{-}) * 1000 \tag{5.1}$$

Donde:

- $Amplitud_{ADC}$ es el valor digitalizado directamente extraído del ADC.
- V^{+} y V^{-} son, respectivamente, el límite superior e inferior del ADC.
- N son los bits de resolución del ADC.

Al acabar se multiplica por 1000 para hacer la conversión de voltios a milivoltios. Esta expresión también se puede utilizar en caso de que la resolución del ADC no esté comprendida en un rango que empiece en 0 V, con no más que adaptar V^{+}. **Nótese que esta conversión a valores de tensión no se aplica a las muestras que se almacenan en formato CSV**.

```
private void saveData() {
        // Save the plotted data in a CSV file

        short[] short_data = mAcquisition.getConvertedData();
        String file_name = "DataPointsStorage";

        try {
                mAcquisition.saveAcquisition(short_data, file_name);
        } catch (IOException e) {
                throw new RuntimeException(e);
        }
}
```

Listing 5.7: Método que permite llamar a la función para almacenar las muestras en un fichero CSV.

```
private void updateGraphics(){
        // Displays the RMS value in mV and plot the signal

        double RMS = mAcquisition.calculateRMS(mAcquisition.getConvertedData());

        // Adapt RMS to mV
        double ADC_resolution = 16;     // ADC resolution bits
        double ADC_top_limit = 3.3;     // Max ADC voltage level
        double ADC_bottom_limit = 0;    // Minimum ADC voltage level

        double mRMS = (((RMS * ADC_top_limit) / (Math.pow(2, ADC_resolution) - 1) +
        ADC_bottom_limit) * 1000);

        // Displays mRMS value in screen
        String mRMS_display = "RMS: " + mRMS + " mV";
        rms_value_display = findViewById(R.id.rms_value_display);
        rms_value_display.setText(mRMS_display);

        if (mRMS>2){
                rms_value_display.setTextColor(Color.RED);
        }
        else {
                rms_value_display.setTextColor(Color.GREEN);
        }

        // Update mlineChart with the data from mAcquisition
        renderData();
}

public void renderData(){
        // Configures the axis for the LineChart before plotting and then
        // calls the function to plot

        // Horizontal axis configuration
        XAxis xAxis = mLineChart.getXAxis();
        xAxis.enableGridDashedLine(10f, 10f, 0f);
        xAxis.setAxisMaximum(60f); // Represent 60 seconds of the signal
        xAxis.setAxisMinimum(0f);

        // Vertical axis configuration
        YAxis leftAxis = mLineChart.getAxisLeft();
        leftAxis.removeAllLimitLines();
        leftAxis.setDrawZeroLine(false);
        leftAxis.setAxisMinimum(0f);

        mLineChart.getAxisRight().setEnabled(false);
        setData();
}

private void setData() {
        // Plot the received 2 bytes data

        short[] short_data = mAcquisition.getConvertedData();
        int data_array_length = short_data.length;

        // Conversion: Byte[] -> int[]
        int[] int_data = new int[data_array_length];
        for (int i = 0; i < data_array_length; i++){
                int_data[i] = short_data[i] & 0xFFFF;
        }

        // Conversion: int[] -> float[]
        float[] float_data = new float[data_array_length];
        for (int i = 0; i < data_array_length; i++){
                float_data[i] = (float) int_data[i];
        }

```

```
68          // Conversion from samples to milivolts
69          double ADC_resolution = 16;     // ADC resolution bits
70          double ADC_top_limit = 3.3;     // Max ADC voltage level
71          double ADC_bottom_limit = 0;    // Minimum ADC voltage level
72
73          float[] mV_data = new float[data_array_length];
74          for (int i = 0; i < data_array_length; i++){
75                  mV_data[i] = (float) (((float_data[i] * ADC_top_limit) /
76                  (Math.pow(2, ADC_resolution) - 1) + ADC_bottom_limit) * 1000);
77          }
78
79          // Add values to entry list
80          ArrayList<Entry> data_points = new ArrayList<>();
81          float j = 0;
82          for (int i = 0; i < data_array_length; i++){
83                  j += 0.001;  // Seconds per sample
84                  data_points.add(new Entry(j, mV_data[i]));
85          }
86
87          LineDataSet plot1;
88          plot1 = new LineDataSet(data_points, "ECG");
89          LineData data = new LineData(plot1);
90          plot1.setDrawCircles(false);
91          plot1.setLineWidth(0f);
92          plot1.setColor(Color.argb(255,25,190,190));
93          mLineChart.setData(data);
94  }
```

Listing 5.8: Métodos para representar la señal recibida del ECG a partir de las muestras de 16 bts y su valor valor RMS.

Capítulo 6

Resultados y adquisiciones

En este capítulo se tratan las medidas que se han conseguido obtener usando el dispositivo de procesamiento central junto con los diseños hechos en el Capítulo 4: Diseño firmware y hardware para la obtención de adquisiciones. En la sección Adquisiciones ECG se pueden consultar los resultados conseguidos mediante el electrocardiograma, mientras que en Adquisiciones $SpO2$ se pueden consultar las medidas del porcentaje de saturación de oxígeno.

6.1. Adquisiciones ECG

En esta sección se trata las señales medidas del ECG, así como el postprocesado que se ha hecho en Matlab. Hay que tener en cuenta que las muestras tomadas tienen un nivel de ruido considerable superpuesto a ellas, debido a que al tomarse todas las medidas en uno de los laboratorios de la Universidad de Granada existían multitud de equipos electrónicos, presentes tanto la propia habitación como en laboratorios contiguos, lo que añade una cantidad de EMI considerable a los resultados.

En la Figura 6.1 se puede consultar la señal que se extrae directamente del CSV generado por la aplicación Android. Tal y como se muestra en la representación temporal de la señal, se tiene superpuesta una señal sinusoide de 50 Hz de amplitud considerable. Esto se puede encontrar también en el espectro de la señal, el cual se representa en la Figura 6.4. Para eliminar dicho ruido se diseñal un filtro de *Notch*, también llamado filtro ranura, con la intención de eliminar únicamente esta banda de frecuencia particular.

En primer lugar se ha intentado conseguir el filtrado mediante un filtro de 2^{o} orden, cuyo resultado se puede consultar en la Figura 6.2. Al considerarse este como insuficiente, se ha probado también a realizar el filtrado haciendo uso de un filtro de 6^{o} orden. En la Figura 6.3 se pueden consultar los resultados, donde además se han resaltado, para este caso, los puntos de interés de la señal cardíaca que ya se mencionaron en la sección Fundamento teórico del ECG del capítulo Diseño firmware y hardware para la obtención de adquisiciones. Los espectros de frecuencia para los tres casos mencionados se pueden visualizar en las Figuras 6.4, 6.5 y 6.6.

Ambos Notch se han diseñado como filtros IIR, ya que de dicha forma se consigue una mayor atenuación con un menor orden del filtro. Adicionalmente, se ha configurado que el ancho de banda de la banda suprimida sea de 1 Hz. Los coeficientes de los dos filtros se han obtenido utilizando funcionalidades integradas de Matlab.

El ritmo normal del corazón en reposo se encuentra en el rango de 60 a 100 pulsaciones por minuto, también llamadas BPM (*Beats Per Minute*). Resulta sencillo extraer este valor a partir de la señal del ECG. Para ello, se comienza extrayendo los instantes en los que se dan el primer y último pico R de la señal cardíaca. A continuación, se restan ambos valores de tiempo, teniendo en cuenta que ambos se encuentran en unidades de segundos, el resultado de esta operación se le divide a la cantidad de picos R detectados. El resultado nos proporciona las pulsaciones por segundo, con lo cual solo hay que multiplicar por 60 segundos para conocer el ritmo cardíaco en unidades de pulsaciones/minuto.

$$BPM = \frac{N^{\circ}\ de\ picos\ R}{Instante\ primer\ pico - Instante\ último\ pico} * 60 \tag{6.1}$$

Por poner un ejemplo, se ha calculado que la señal cardíaca representada en la Figura 6.3 tiene una frecuencia de aproximadamente 76 pulsaciones/minuto.

En la sección Script de Matlab se puede consultar el *script* empleado para el tratamiento de las muestras y el cálculo del BPM.

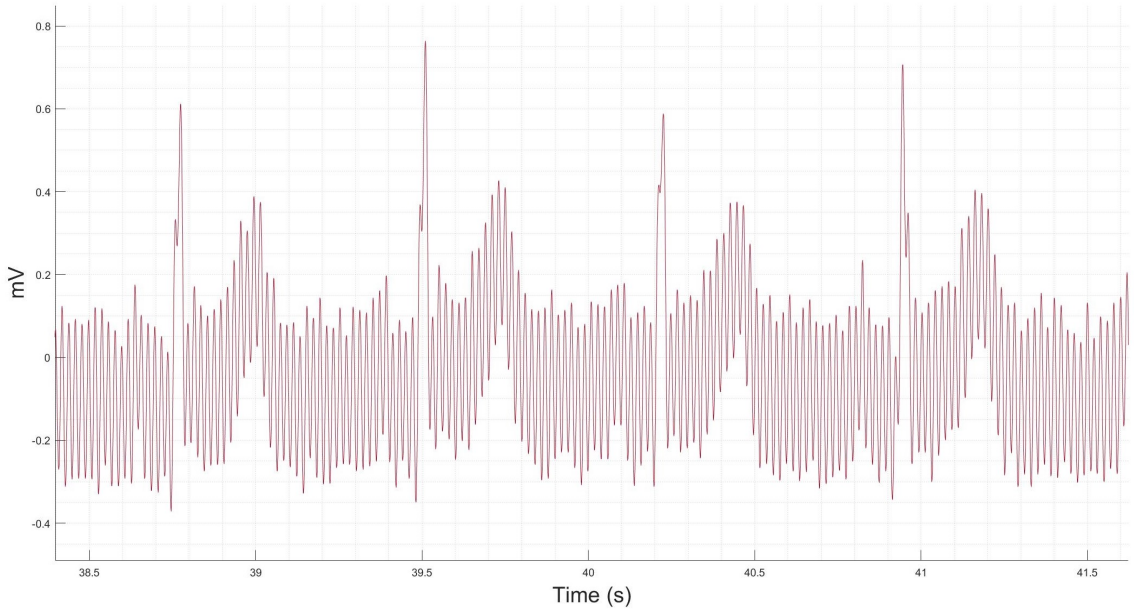


Figura 6.1: Espectro de la señal del ECG sin filtrado.

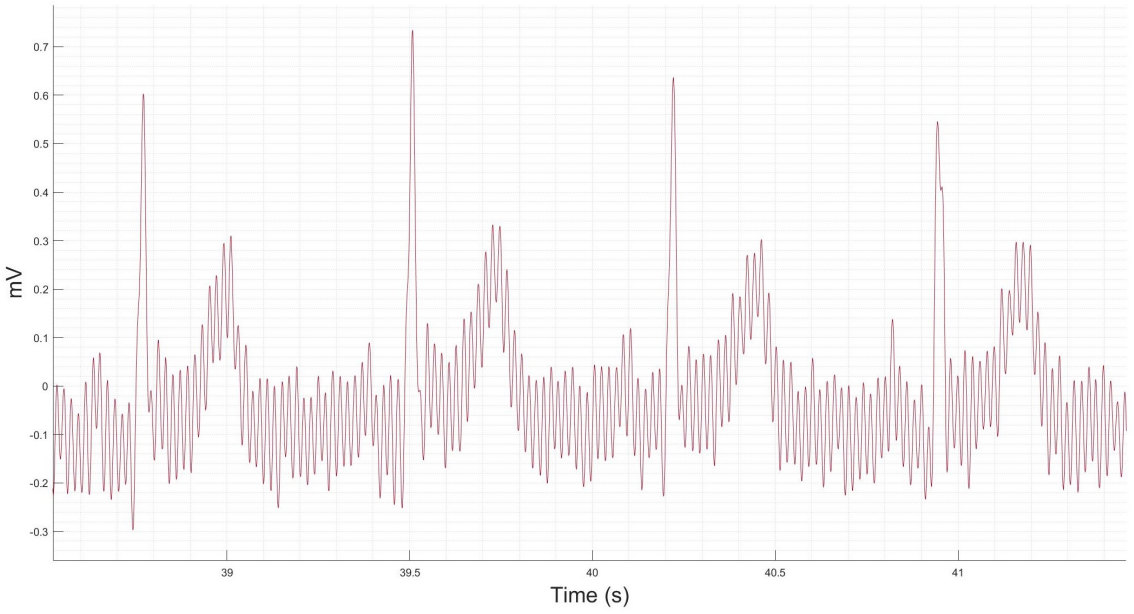


Figura 6.2: Señal del corazón aplicando un filtro de Notch a 50 Hz de 2º orden.

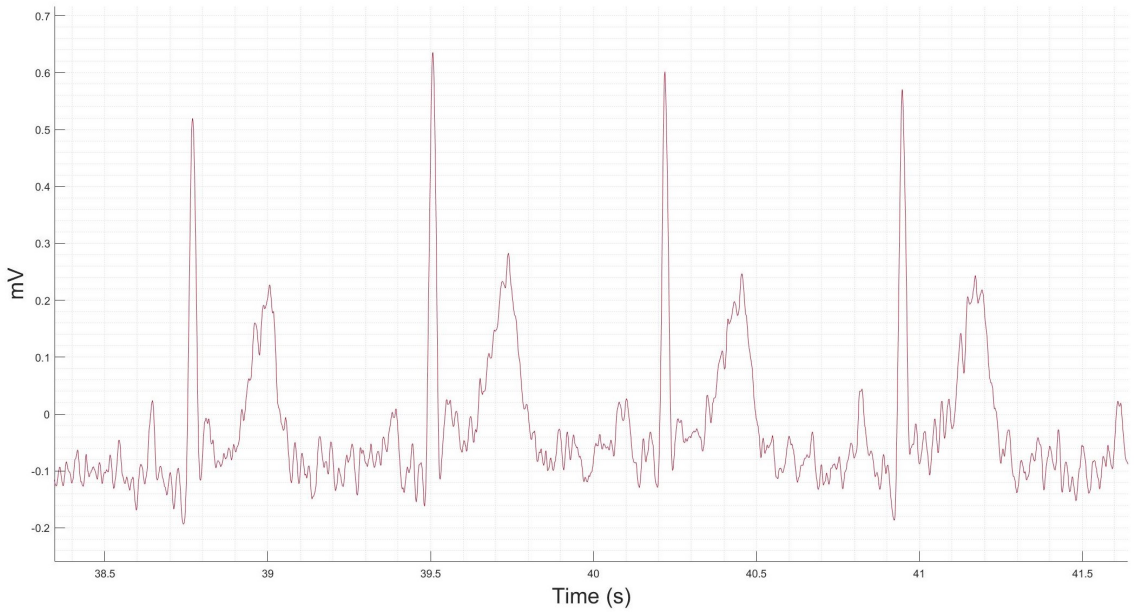


Figura 6.3: Señal del corazón aplicando un filtro de Notch a 50 Hz de 6º orden, donde adicionalmente se ha resaltado el complejo PQRST.

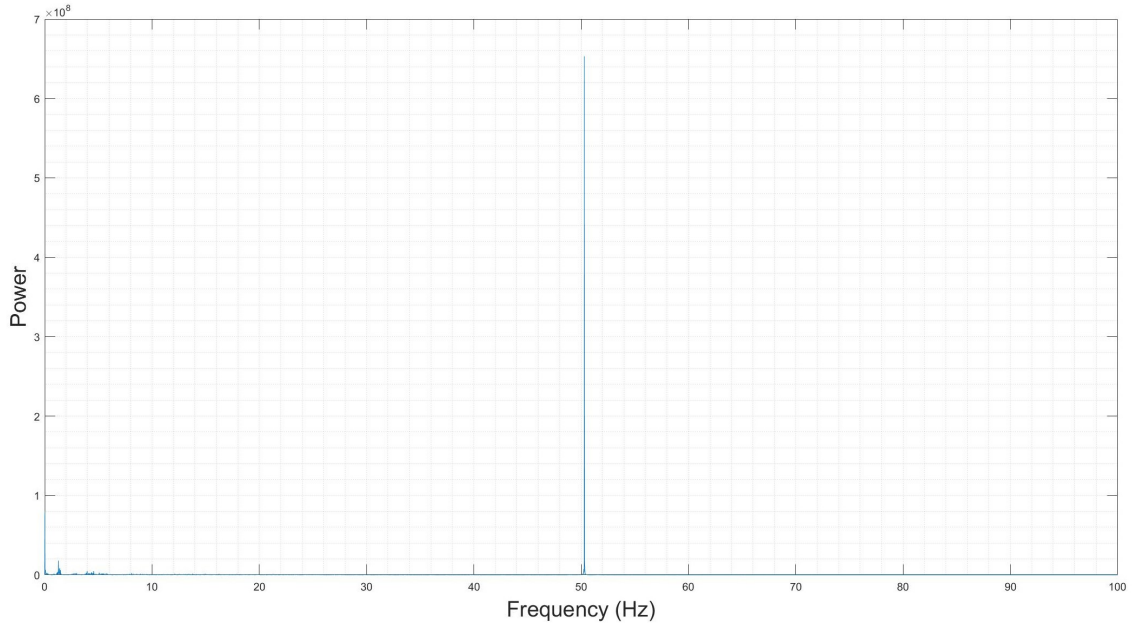


Figura 6.4: Espectro de la señal del ECG sin filtrado.

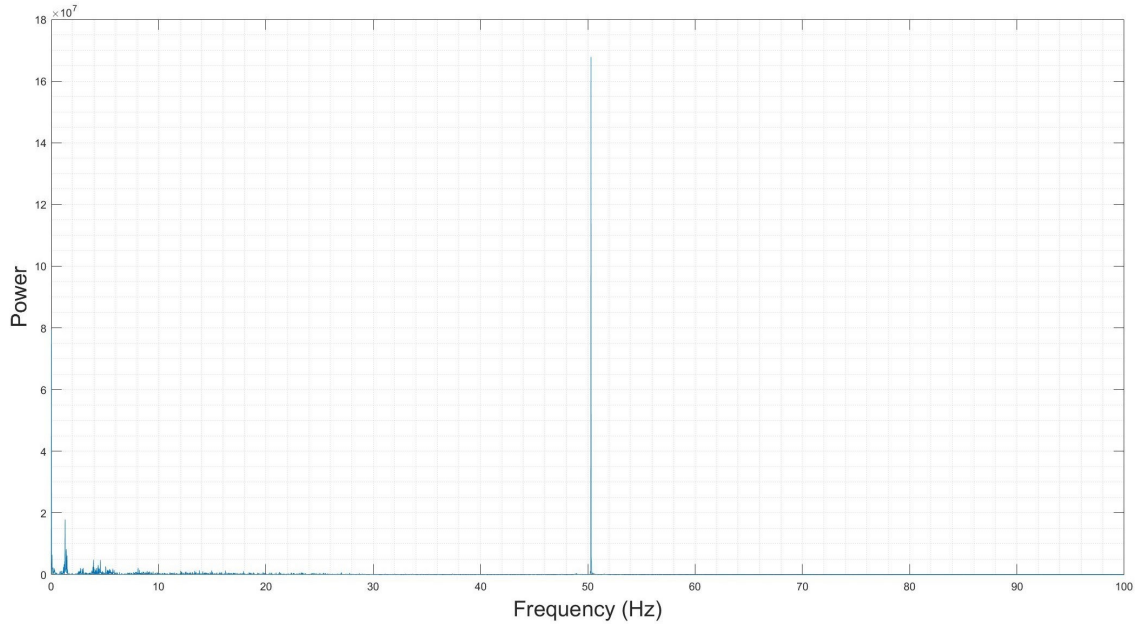


Figura 6.5: Espectro de la señal del ECG con filtro de Notch de 2º orden.

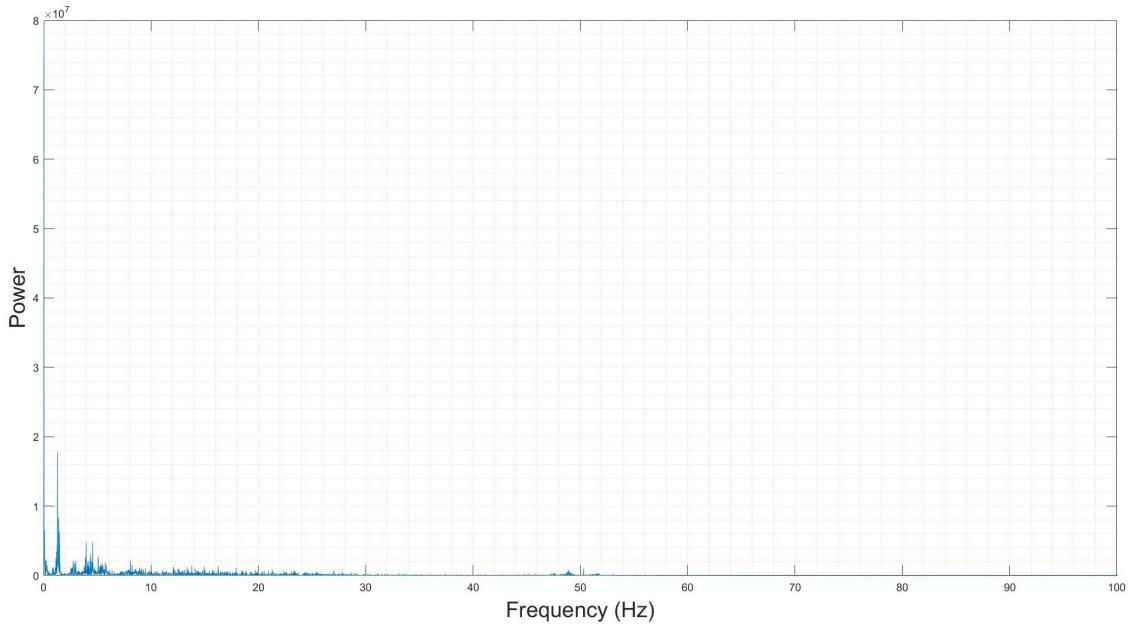


Figura 6.6: Espectro de la señal del ECG con filtro de Notch de 6º orden.

6.1.1. Script de Matlab

En esta sección se puede consultar el *script* de Matlab empleado para la representación gráfica de la señal cardíaca y su filtrado de la banda de 50 Hz, así como el cálculo del ritmo cardíaco. Nótese que en el *script* presente en este apartado se ha suprimido el código Matlab respectivo a la propia graficación de la señales al no considerarse relevante, el código completo se puede consultar en el Anexo E.

Para el diseño del filtro de orden 2, se emplea la función "iirnotch()", que devuelve directamente como resultado los coeficientes de un filtro IIR de Notch centrado a la frecuencia indicada. En

cambio, para la obtención de los coeficientes del filtro de 6º se ha utilizado la herramienta integrada de Matlab: "Filter Designer" [136].

Los valores de amplitud se representan en función del tiempo en un rango de 0 a 60 segundos. Dado que se usa una frecuencia de muestreo de 1000 m/s, se representa una muestra cada 0.001 segundos. Para el trazado de la amplitud de las muestras, hay que hacer la conversión a valores de tensión que también se realizó en la aplicación de Android del Capítulo 5.

En cuanto al cálculo de las pulsaciones por minuto, se utiliza la función de Matlab *findpeaks()*, habiéndose configurado un valor mínimo de voltaje para ser considerado como pico y una distancia mínima entre dichos picos de modo que no se contabilicen falsos puntos R.

```
%% Script that plots and filters the ECG data from a CSV file

clc; clear vars; close all; format long;

%% Read data from CSV file
filename = 'Name of the CSV file';
filedir = strcat('AdquisicionesECG/',filename);
filedir = strcat(filedir,'.csv');
data = csvread(filedir);

fs = 1000;  % Sampling frequency

%% Spectrum of the raw signal
y = fft(data');           % Fast Fourier transform
n = length(data);         % Number of samples
f = (0:n-1)*(fs/n);       % Frequency range
power = abs(y).^2/n;      % Power of the DFT
power(1,1) = 0;

%% Choose filter
n = 3;
switch n
        case 1
                disp('No filter')
        case 2  % 50 Hz filter (2 order)
                f = 50;    % Frequency to filter
                BW = 1;     % Bandwidth
                [num,den] = iirnotch(f/(fs/2),BW/(fs/2));
                fvtool(num,den,'Fs',fs);
                data = filter(num,den,data);
        case 3  % 50 Hz filter (6 order)
                % Filter coeficients
                num =   [0.987512174869590, -5.63519056682043, ...
                13.6815175263959, -18.0667531080674, ...
                13.6815175263959, -5.63519056682043, ...
                0.987512174869590];
                den = [1, -5.68254876635053, 13.7387485793903, ...
                -18.0664564757185, 13.6241305276252, ...
                -5.58812899963934, 0.975180295515611];
                data = filter(num,den,data);
        otherwise
                disp('No filter')
end
```

Listing 6.1: Scritp de Matlab para representación, filtrado rechazo banda de la señal cardíaca y cálculo del ritmo cardíaco. (Parte 1 de 2).

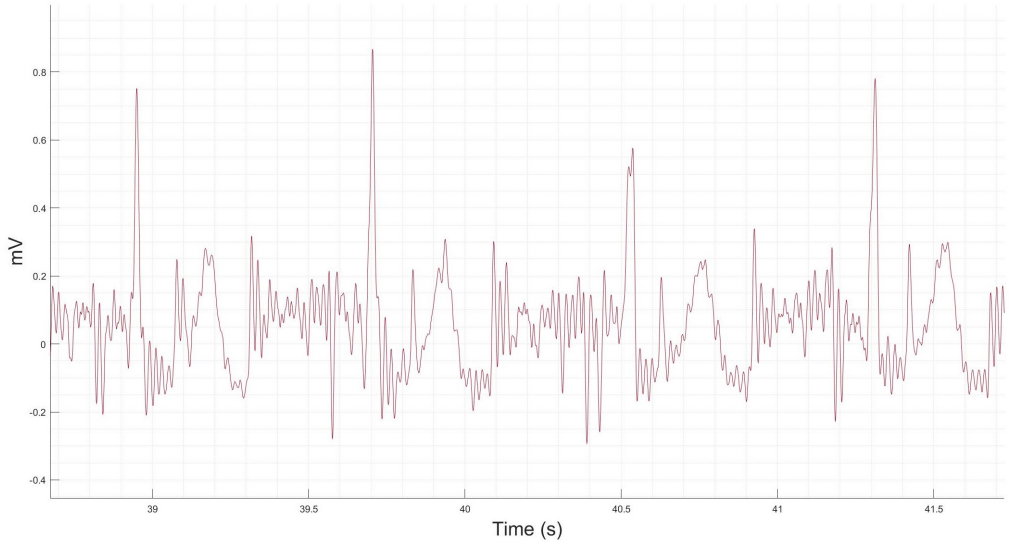
```
%% Spectrum of the filtered signal
y = fft(data');
n = length(data);        % Number of samples
f = (0:n-1)*(fs/n);      % Frequency range
power = abs(y).^2/n;     % Power of the DFT
power(1,1) = 0;

%% RMS and mean
% ADC
ADC_resolution = 16;  % Bits of the ADC
ADC_top_limit = 3.3;
ADC_bottom_limit = 0;

mV_data = zeros(1, length(data));

for i = 1:1:length(data)
mV_data(i) = ((data(i) * ADC_top_limit) / (2^ADC_resolution - 1) +
    ...
ADC_bottom_limit) * 1000/51;
end

fprintf('Mean: %f mV\n', mean(mV_data));
fprintf('RMS: %f mV\n', rms(mV_data));

mV_data = mV_data - mean(mV_data); % Remove offset

%% Calculate heart rate (BPM)
min_amp = 0.4;     % Minimun amplitud (0.4 mV)
min_dis = 30e-3;   % Minimun distante between peaks (30 ms)

[pks,locs] = findpeaks(mV_data, fs, 'MinPeakHeight', min_amp,...
'MinPeakDistance', min_dis);
BPM = length(locs) / (locs(end)- locs(1)) * 60;  % Beats per minute
fprintf('BPM: %f \n', BPM);
```

Listing 6.2: Scritp de Matlab para representación, filtrado rechazo banda de la señal cardíaca y cálculo del ritmo cardíaco. (Parte 2 de 2).

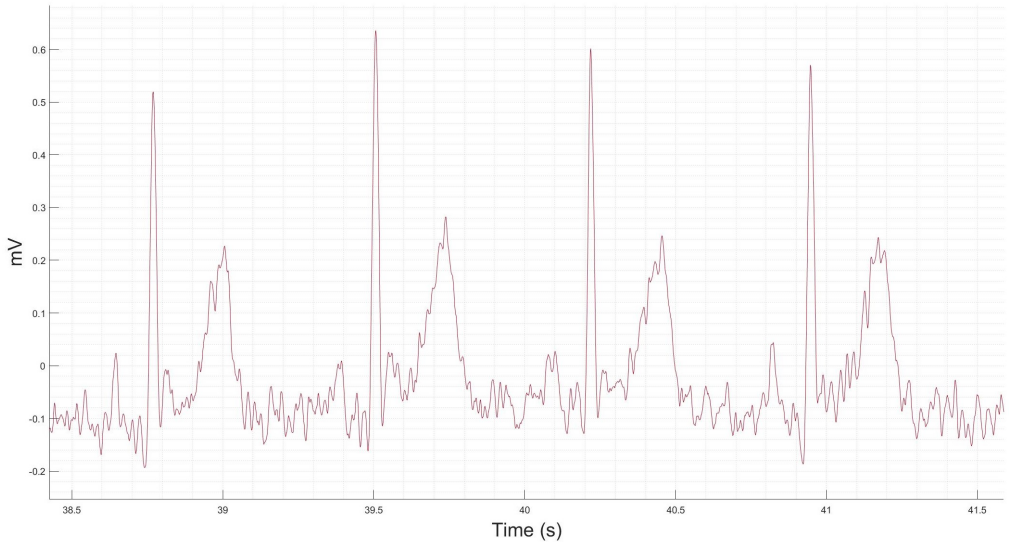
6.1.2. Recopilación de medidas

Se ha adquirido la medidas de la señal del corazón a varios voluntarios. Dado que, como ya se ha comentado, las muestras se tomaron en uno de los laboratorios de la Universidad de Granada, con la intención de preservar la intimidad de los voluntarios se optó por situar los electrodos en las muñecas derecha e izquierda y en la pierna derecha, en lugar de colocarlos en el torso como también se sugirió como posible en el Capítulo 4: Diseño firmware y hardware para la obtención de adquisiciones. A todos los resultados se les ha aplicado un filtrado ranura de 6º orden antes de ser representados. Las señales extraídas para estos casos se pueden consultar en las Figuras 6.7, 6.8 y 6.9.

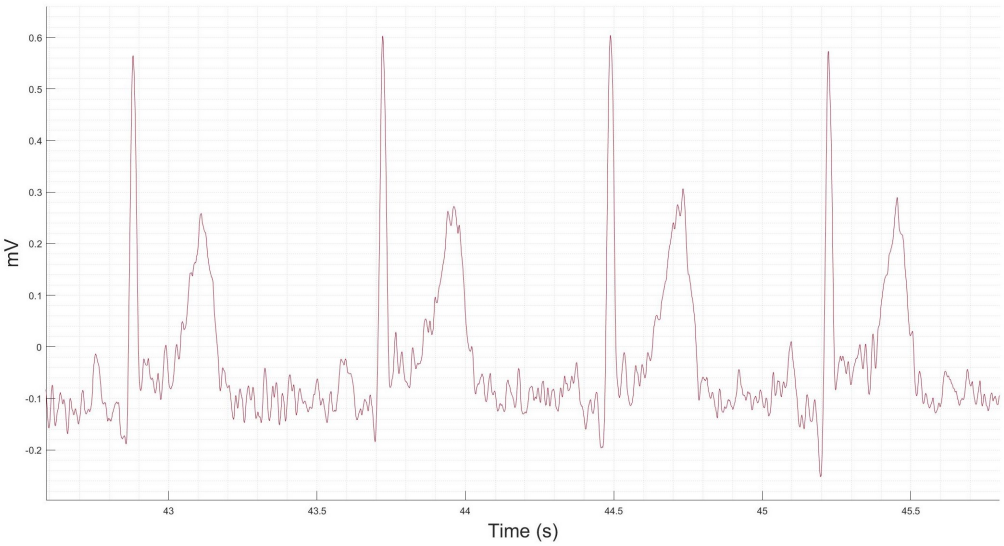
Se puede apreciar una cantidad de ruido considerable en los resultados extraídos del voluntario 3 en la Figura 6.9 en comparación al resto de medidas. Esto se puede deber a una mala colocación de los electrodos. Por ejemplo, se puede haber dado el caso de que al colocarlos estos quedasen flojos, con lo cual no hubo un contacto adecuado entre la piel y el electrodo, incrementándose con ello el ruido.



(a) Medida 1.

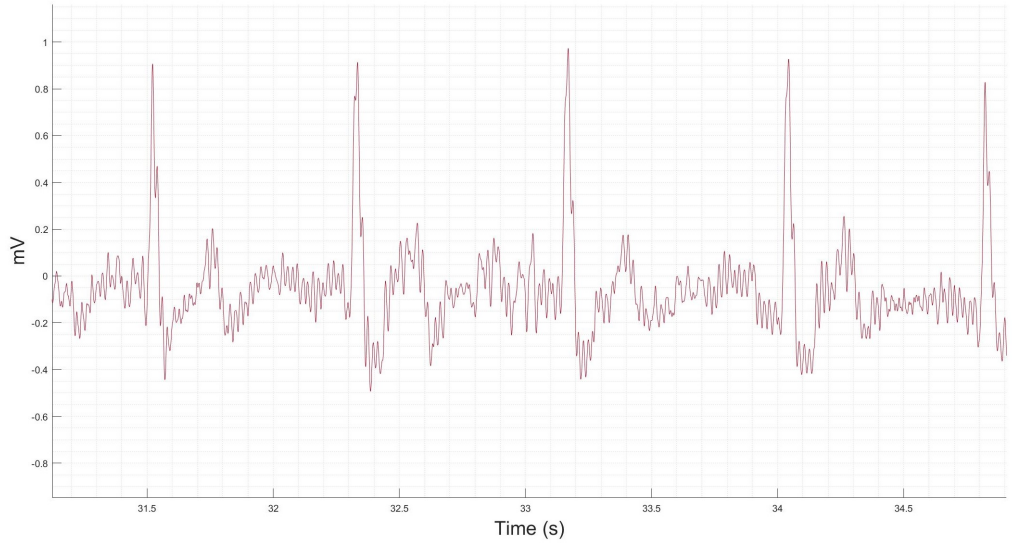


(b) Medida 2

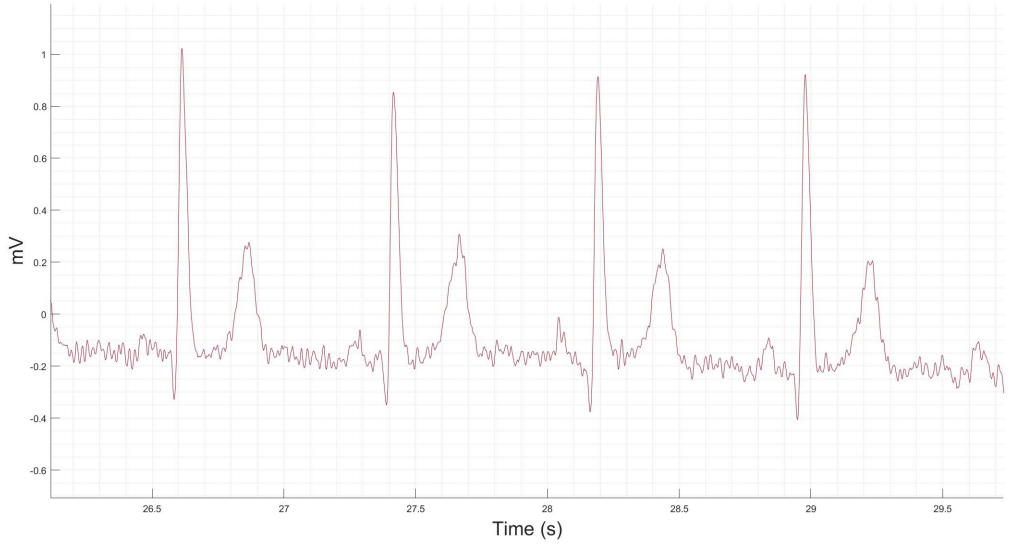


(c) Medida 3

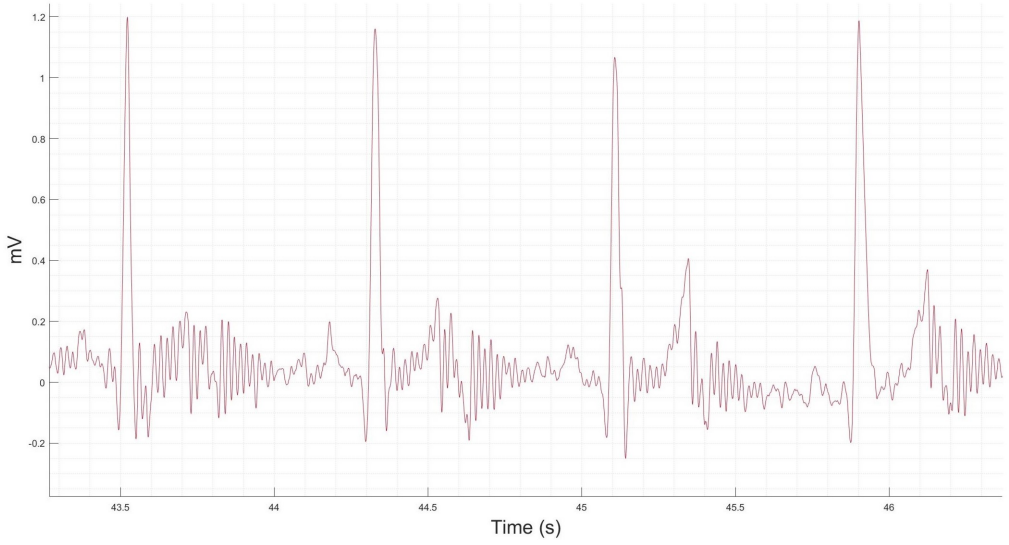
Figura 6.7: Señal del ECG extraída de la persona 1.



(a) Medida 1.

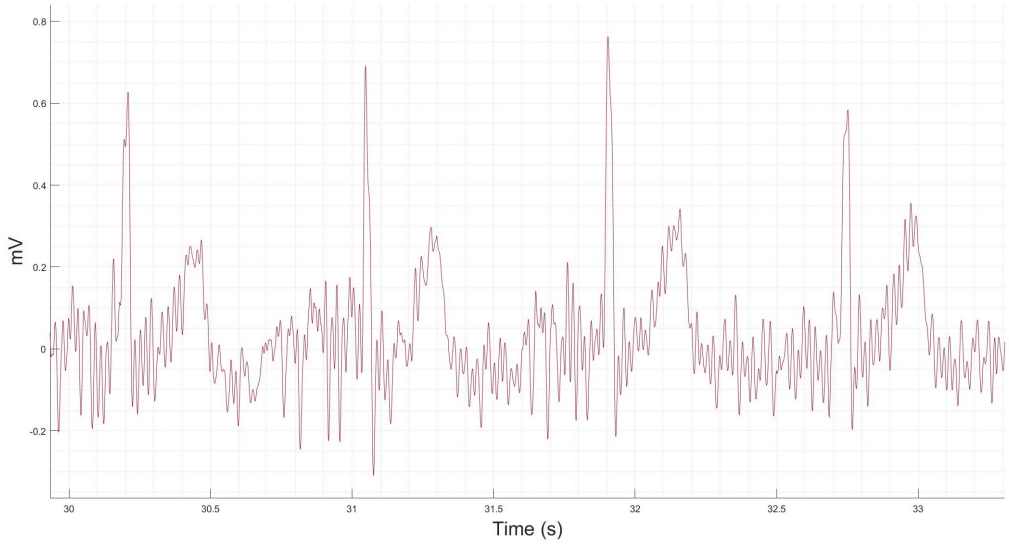


(b) Medida 2

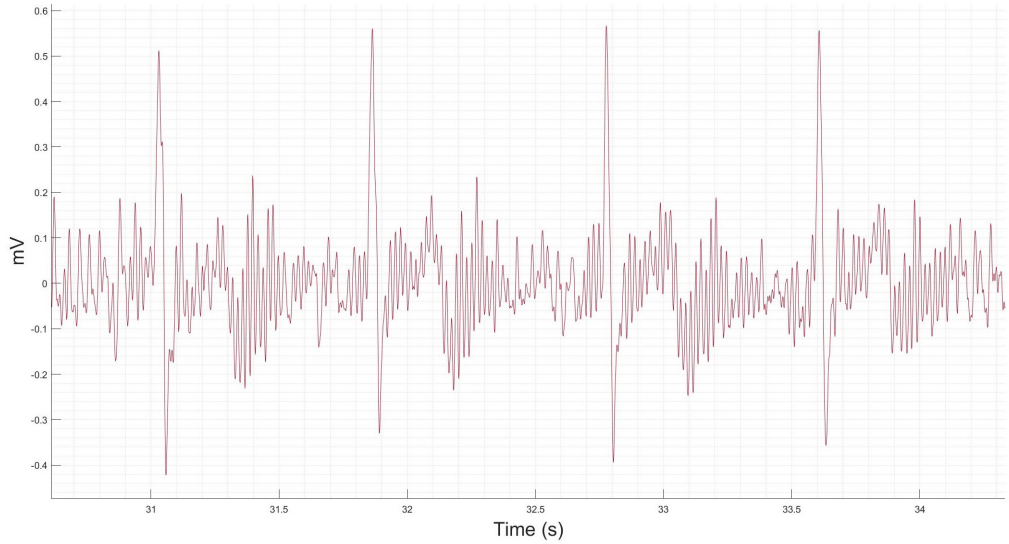


(c) Medida 3

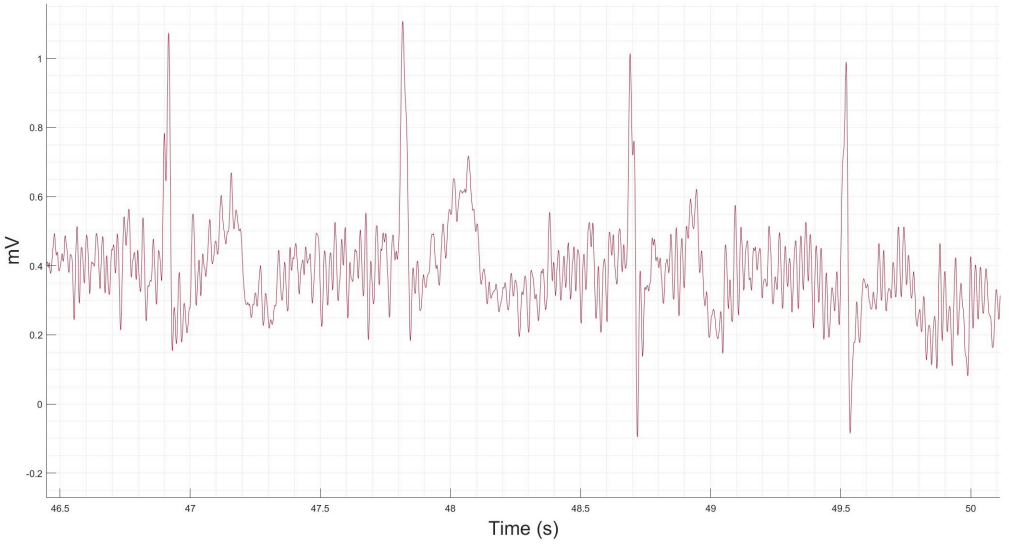
Figura 6.8: Señal del ECG extraída de la persona 2.



(a) Medida 1.



(b) Medida 2



(c) Medida 3

Figura 6.9: Señal del ECG extraída de la persona 3.

6.2. Adquisiciones $SpO2$

Para dar certeza de que los valores de $SpO2$ que se han adquirido con el dispositivo fabricando son correctos, se han los valores conseguidos para una misma persona utilizando, por un lado el dispositivo de este proyecto, y por el un pulsioxímetro comercial. Este último calcula el porcentaje de $SpO2$ usando la intensidad de luz transmitida en lugar de la reflejada, aún así este hecho debería de afectar al resultado final.

La adquisición con el dispositivo fabricado en el proyecto se ha tomado en la falange distal, es decir, en la punta del dedo de la mano, en un punto cercano a la articulación. Se ha comprobado experimentalmente que en esta área es donde se han conseguido los resultados más consistentes. En la Figura 6.10 se han representado dos gráficas correspondientes al porcentaje de oxígeno obtenido con ambos dispositivos. En el peor de los casos posibles existe una diferencia del 2 % en la medida que adquieren ambos dispositivos.

Hay que tener en cuenta ninguno de los dos dispositivos muestra decimales en el resultado final, con lo cual esto puede ser fuente de discrepancia.

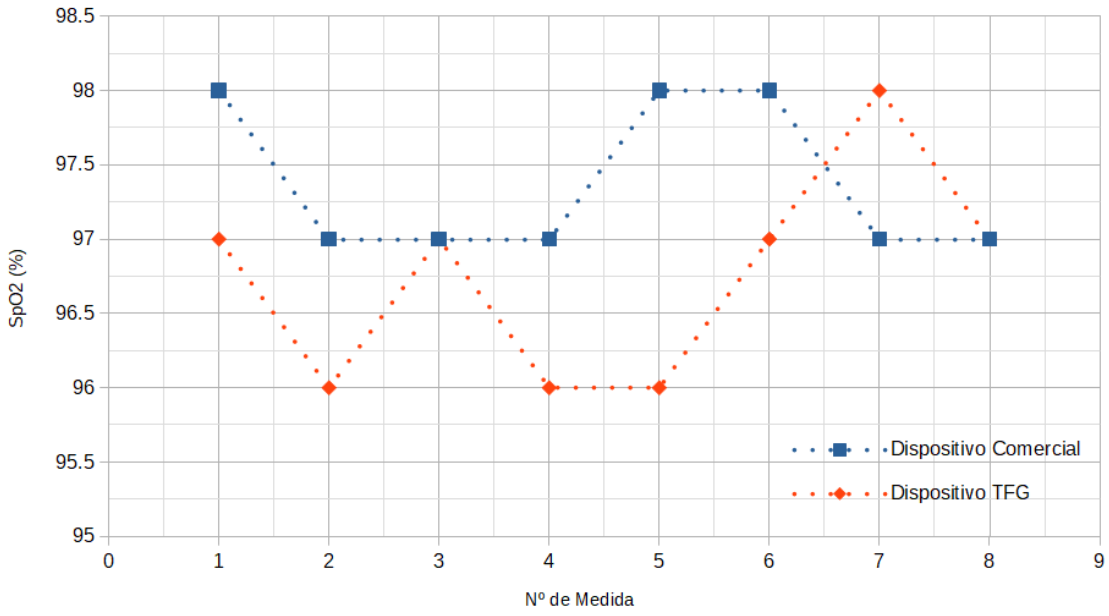


Figura 6.10: fig:Resultados del porcentaje de SpO_2 obtenidos con dispositivo fabricado para este trabajo y el comercial.

Capítulo 7

Conclusiones y lineas de trabajo futuras

En este capítulo final se tratan en primer lugar los objetivos que se han logrado cumplir a lo largo del desarrollo del proyecto (Objetivos logrados). A continuación, se puede consultar que líneas de trabajo futuras se plantean para posteriores versiones de este mismo trabajo (Líneas de trabajo futuras).

7.1. Objetivos logrados

En esta sección se tratan que objetivos de los que se plantearon al comienzo de este documento. Por tanto, se han logrado:

- Desarrollar un dispositivo de procesamiento central de reducido consumo y tamaño, el cual resulta relativamente fácil de reconfigurar para diversas aplicaciones relacionadas con la extracción de medidas de cuerpo humano de manera no invasiva.
- Creado tanto *firmware* para la programación del procesador como el *hardware* externo necesario para adquirir tanto la señal cardíaca mediante el método de tres derivaciones y el porcentaje de saturación de oxígeno.
- Diseñar una aplicación Android compatible con el dispositivo de procesamiento central, la cual permite visualizar y almacenar los resultados extraídos del ECG.
- Desarrollar *script* de Matlab para gráficar y añadir postprocesado adicional a las muestras de la señal cardíaca almacenadas desde la aplicación Android.
- Adquirir de manera exitosa la señal cardíaca en varios voluntarios.
- Adquirir de manera exitosa el $SpO2$, habiéndose verificado los resultados mediante el uso de un dispositivo comercial.

7.2. Líneas de trabajo futuras

Debido a la extensión y a la multitud de partes que componen este Trabajo de Fin de Grado, existe varias mejoras que se pueden plantear sobre las distintas partes desarrolladas. Este subapartado trata de hacer autocrítica y plantear unas líneas de trabajo claras para posibles futuras iteraciones del proyecto.

7.2.1. Dispositivo de procesamiento central

- **Entrada alternativa para la alimentación**. Por el momento, la entrada de alimentación a la PCB está formada por dos pines macho, lo cual resulta conveniente de cara a realizar pruebas sobre un prototipo en laboratorio. De cara a un dispositivo final que fuese a ser utilizado por un usuario, sería conveniente sustituir o al menos dar también como posibilidad el uso de un puerto USB como entrada de alimentación.

- **Retrazado de las pistas que transcurren por debajo de la antena**. Sería interesante revaluar si realmente resulta posible trazar las pistas que conectan las distintas partes eliminando completamente o al menos reduciendo la cantidad de las que pasan por debajo de la antena. Esto se hace con la intención de intentar mejorar la propagación de la señal.

- **Sistema para la evaluación del nivel de batería**. En caso de operar mediante el uso de baterías, se debería de añadir un circuito que permita la evaluación del tiempo restante de alimentación de la fuente de alimentación externa. Dicha información debe de ser leída en el PSoC 5LP y posteriormente transmitida hacia la aplicación de manera que el usuario pueda visualizarla.

- **Encapsulado para el dispositivo central**. Al momento de finalizar este proyecto, la circuitería del dispositivo de procesamiento central se encuentra expuesta. Hay que diseñar un encapsulado adecuado para el mismo, el cual lo proteja de factores externos y que permita seguir haciendo uso de los puertos de entrada y salida.

- **Unificación de librerías de componentes en Altium**. Aunque no es parte del propio diseño final, de cara a futuras iteraciones del dispositivo resultaría conveniente unificar todos los símbolos y *footprints* de Altium Designer en una única librería. Esto se puede hacer por conveniencia de cara al desarrollador.

7.2.2. Aplicación de Android

- **Solucionar problemas con los permisos**. Como ya se comentó en el capítulo correspondiente al desarrollo de la aplicación, en versiones más recientes de Android se han producido cambios significativos en la solicitud de permisos, con la intencionalidad de hacer el uso de aplicaciones más seguro de cara al usuario final. Se ha tratado de que la aplicación creada en este trabajo sea compatible con la mayor cantidad de versiones de Android posibles, lo que en ocasiones puede ocasionar conflictos con los permisos de app. Habría que tratar de corregir dichos errores o escoger si hacer una migración completa de la aplicación a versiones de Android iguales o posteriores a Android 11 (nivel de API 30).

- **Añadir compatibilidad con otras medidas**. Actualmente, la aplicación sólo es compatible con la extracción de la señal cardíaca. Habría que añadir funcionalidades extra de manera que se pueda usar con otras medidas que se planteen adquirir mediante el dispositivo de procesamiento central como, por ejemplo, el porcentaje de saturación de oxígeno (SpO_2).

- **Añadir menú de opciones y personalización**. Sería necesario incluir un menú de opciones, de manera que el usuario pueda personalizar la app y adecuarla a su uso concreto. Por ejemplo, sería posible agregar un listado que permita seleccionar que medida se va a tomar en el dispositivo físico, modificar el idioma, etc.

- **Cambios en la apariencia**. Se pueden plantear cambios en el apartado gráfico de la aplicación de manera que resulte más agradable para el usuario.

7.2.3. Medida del SpO_2

- **PCB para el kit del expansión**. La circuitería externa añadida para la adquisición del SpO_2 se construido sobre una placa de prototipaje, debido a que el tiempo que se tardaría en recibir desde fábrica una PCB que lo incluyese sería demasiado extenso respecto del tiempo que se ha otorgado a este proyecto. En futuras iteraciones hay que diseñar dicha PCB, de manera que el dispositivo se pueda usar para estas medidas de manera externa a una situación de laboratorio.

- **Pulsera para portar los dispositivos**. El diseño de un artilugio similar a una pulsera o muñequera sobre la que incrustar el dispositivo de procesamiento central y el *hardware* adicional necesario para la medida del oxígeno, permitiría portar el dispositivo de manera cómoda, haciendo por tanto esta configuración verdaderamente un dispositivo vestible.
- **Escoger si recibir el valor de SpO_2 o las muestras completas**. Por el momento, el *firmware* para la adquisición del porcentaje de oxígeno en sangre únicamente devuelve dicho valor. Se podría añadir el poder escoger si mantener este comportamiento o en su lugar recibir los vectores de muestras completos a partir de los que se calcula el ratio de absorción R, de manera que se pudiera hacer postprocesado en un *software* como, por ejemplo, Matlab al igual que ocurre con la señal del ECG en caso de que así se desease.

Bibliografía

[1] Sabban, A. (2018). *Wearable communication systems and antennas for commercial, sport and medical applications*. Institute of Physics Publishing.

[2] Silverman, B. G., Jain, A., Ichalkaranje, A., & Jain, L. C. (Eds.). (2005). *Intelligent Paradigms for Healthcare Enterprises: Systems thinking*. Springer. https://doi.org/10.1007/b99809

[3] *Población residente en España a 1 de enero, por sexo, edad y año.* Instituto Nacional de Estadística. https://www.ine.es/jaxiT3/Datos.htm?t=36643#!tabs-grafico. Accedido: Enero 2023.

[4] *Population structure indicators at national level.* Eurostat. https://ec.europa.eu/eurostat/databrowser/view/DEMO_PJANIND__custom_4297410/default/maplang=en. Accedido: Enero 2023.

[5] *Población residente en España a 1 de enero, por edad y año (2040).* Instituto Nacional de Estadística. https://www.ine.es/jaxiT3/Datos.htm?t=36643#!tabs-grafico. Accedido: Enero 2023.

[6] Lloyd-Sherlock, P. (2000). Population ageing in developed and developing regions: implications for health policy. *Social Science & Medicine (1982), 51(6)*, 887–895. https://doi.org/10.1016/s0277-9536(00)00068-x

[7] Al Mutair, A., Layqah, L., Alhassan, B., Alkhalifah, S., Almossabeh, M., AlSaleh, T., Al-Sulaiman, Z., Alatiyyah, Z., Almusalami, E. M., Al-Jamea, L. H., Woodman, A., Hajissa, K., Alhumaid, S., & Rabaan, A. A. (2022). Estimated cost of treating hospitalized COVID-19 patients in Saudi Arabia. *Scientific Reports*, 12(1), 21487. https://doi.org/10.1038/s41598-022-26042-z

[8] Trentini, F., Marziano, V., Guzzetta, G., Tirani, M., Cereda, D., Poletti, P., Piccarreta, R., Barone, A., Preziosi, G., Arduini, F., Della Valle, P. G., Zanella, A., Grosso, F., Del Castillo, G., Castrofino, A., Grasselli, G., Melegaro, A., Piatti, A., Andreassi, A., ... Merler, S. (2022). Pressure on the health-care system and intensive care utilization during the COVID-19 outbreak in the Lombardy region of Italy: A retrospective observational study in 43,538 hospitalized patients. *American Journal of Epidemiology, 191*(1), 137–146. https://doi.org/10.1093/aje/kwab252

[9] Goschin, Z., & Dimian, G. C. (2021). Healthcare under pressure: modelling COVID-19 fatalities with multiscale geographically weighted regressions. Kybernetes. *The International Journal of Cybernetics, Systems and Management Sciences*(ahead-of-print). https://doi.org/10.1108/k-07-2021-0548

[10] Dey, N., Ashour, A. S., Fong, S. J., & Bhatt, C. (2019). *Wearable and Implantable Medical Devices: Applications and challenges*. Academic Press.

[11] Navarro, J., Vidaña-Vila, E., Alsina-Pagès, R., & Hervás, M. (2018). Real-time distributed architecture for remote acoustic elderly monitoring in residential-scale ambient Assisted Living scenarios. *Sensors (Basel, Switzerland), 18*(8), 2492. https://doi.org/10.3390/s18082492

[12] Groat, D., Kwon, H. J., Grando, M. A., Cook, C. B., & Thompson, B. (2018). Comparing real-time self-tracking and device-recorded exercise data in subjects with type 1 diabetes. *Applied Clinical Informatics*, (4), 919–926. urlhttps://doi.org/10.1055/s-0038-1676458

[13] Klepin, K., Wing, D., Higgins, M., Nichols, J., & Godino, J. G. (2019). Validity of cardiorespiratory fitness measured with Fitbit compared to V˙O2max. *Medicine and Science in Sports and Exercise, 51*(11), 2251–2256. https://doi.org/10.1249/mss.0000000000002041

[14] Mühlen, J. M., Stang, J., Lykke Skovgaard, E., Judice, P. B., Molina-Garcia, P., Johnston, W., Sardinha, L. B., Ortega, F. B., Caulfield, B., Bloch, W., Cheng, S., Ekelund, U., Brønd, J. C., Grøntved, A., & Schumann, M. (2021). Recommendations for determining the validity of consumer wearable heart rate devices: expert statement and checklist of the INTERLIVE Network. *British Journal of Sports Medicine, 55*(14), 767–779. https://doi.org/10.1136/bjsports-2020-103148

[15] James, D. A., & Petrone, N. (2016). *Sensors and wearable technologies in sport: Technologies, trends and approaches for implementation (1st ed.)*. Springer.

[16] Zieniewicz, M. J., Johnson, D. C., Wong, C., & Flatt, J. D. (2002). The evolution of Army wearable computers. *IEEE Pervasive Computing,1*(4), 30–40. https://doi.org/10.1109/mprv.2002.1158276

[17] *LifeGuard: Wireless physiological monitor*. NASA. https://www.nasa.gov/centers/ames/research/technology-onepagers/life-guard.html. Accedido: Diciembre 2022

[18] Chandra, S. (2017). *Wearable Sensors: Applications, design and implementation* (S. Mukhopadhyay & T. Islam, Eds.). Institute of Physics Publishing.

[19] Mukhopadhyay, S. C. (2015). *Wearable electronics sensors: For safe and healthy living*. Springer International Publishing.

[20] Khan, Y., Ostfeld, A. E., Lochner, C. M., Pierre, A., & Arias, A. C. (2016). Monitoring of vital signs with flexible and wearable medical devices. *Advanced Materials (Deerfield Beach, Fla.), 28*(22), 4373–4395. https://doi.org/10.1002/adma.201504366

[21] Latifoğlu, F., Esas, M. & Demirci, E. (2020). Diagnosis of attention-deficit hyperactivity disorder using EOG signals: a new approach. Biomedical Engineering / Biomedizinische Technik, 65(2), 149-164. https://doi.org/10.1515/bmt-2019-0027

[22] Chowdhury, R. H., Reaz, M. B. I., Ali, M. A. B. M., Bakar, A. A. A., Chellappan, K., & Chang, T. G. (2013). Surface electromyography signal processing and classification techniques. Sensors (Basel, Switzerland), 13(9), 12431–12466. https://doi.org/10.3390/s130912431

[23] Krishnan, S. (2021). *Biomedical Signal Analysis for Connected Healthcare*. Academic Press.

[24] Kyriacou, P. A., & Allen, J. (Eds.). (2021). *Photoplethysmography: Technology, Signal Analysis and Applications*. Academic Press.

[25] Duun, S., Haahr, R. G., Hansen, O., Birkelund, K., & Thomsen, E. V. (2010). High quantum efficiency annular backside silicon photodiodes for reflectance pulse oximetry in wearable wireless body sensors. *Journal of Micromechanics and Microengineering: Structures, Devices, and Systems, 20*(7), 075020. https://doi.org/10.1088/0960-1317/20/7/075020

[26] Hao, Y., & Foster, R. (2008). Wireless body sensor networks for health-monitoring applications. *Physiological Measurement, 29*(11), R27-56. https://doi.org/10.1088/0967-3334/29/11/R01

[27] Panescu, D. (2008). Wireless communication systems for implantable medical devices. *IEEE Engineering in Medicine and Biology Magazine: The Quarterly Magazine of the Engineering in Medicine & Biology Society, 27*(2), 96–101. https://doi.org/10.1109/EMB.2008.915488

[28] W. Tesema, B. Da Silva, W. Jimma and J. Stiens, "Power Saving Techniques for Wearable Devices in Medical Applications," *IECON 2022 – 48th Annual Conference of the IEEE Industrial Electronics Society*, 2022, pp. 1-8, doi: 10.1109/IECON49645.2022.9968977

[29] Raad, H. (2021). Fundamentals of IoT and wearable technology design. Wiley-Blackwell.

[30] Kumar, P., & Lee, H.-J. (2012). Security issues in healthcare applications using wireless medical sensor networks: a survey. *Sensors (Basel, Switzerland), 12*(1), 55–91. https://doi.org/10.3390/s120100055

[31] Wang, L., Lou, Z., Jiang, K., & Shen, G. (2019). Bio-multifunctional smart wearable sensors for medical devices. *Advanced Intelligent Systems (Weinheim an Der Bergstrasse, Germany), 1*(5), 1900040. https://doi.org/10.1002/aisy.201900040

[32] Webster, J. G. (2009). *Medical Instrumentation: Application and Design* (J. G. Webster, Ed.; 4th ed.). John Wiley & Sons.

[33] International Conference on Advances in Human Factors and Wearable Technologies (2017. Los Angeles, California). (2017). *Advances in human factors in wearable technologies and game design: Proceedings of the AHFE 2017 international conference on advances in human factors and wearable technologies, July 17-21, 2017, the Westin Bonaventure hotel, Los Angeles, California, USA* (T. Ahram & C. Falcao, Eds.; 1st ed.). Springer International Publishing.

[34] Leonhardt, S., Falck, T., & Mahonen, P. (Eds.). (2007). *4th international workshop on wearable and implantable body sensor network (bsn 2007)*. Springer.

[35] Steinfeld, E., & Maisel, J. (2012). *Universal Design*. Wiley.

[36] Khoshnoud, F., & de Silva, C. W. (2012). Recent advances in MEMS sensor technology – biomedical applications. *IEEE Instrumentation & Measurement Magazine, 15*(1), 8–14. https://doi.org/10.1109/mim.2012.6145254

[37] Bogue, R. (2007). MEMS sensors: past, present and future. *Sensor Review, 27*(1), 7–13. https://doi.org/10.1108/02602280710729068

[38] Grayson, A. C. R., Shawgo, R. S., Johnson, A. M., Flynn, N. T., Li, Y., Cima, M. J., & Langer, R. (2004). A BioMEMS review: MEMS technology for physiologically integrated devices. *Proceedings of the IEEE. Institute of Electrical and Electronics Engineers, 92*(1), 6–21. https://doi.org/10.1109/jproc.2003.820534

[39] Nag, A., Mukhopadhyay, S. C., & Kosel, J. (2019). *Printed flexible sensors: Fabrication, characterization and implementation* (1st ed.). Springer Nature.

[40] Mukhopadhyay, S. C., & Nag, A. (2021). *Printed and Flexible Sensor Technology: Fabrication and applications* (S. Mukhopadhyay & A. Nag, Eds.). Institute of Physics Publishing. https://doi.org/10.1088/978-0-7503-3439-6

[41] Khan, S., Lorenzelli, L., & Dahiya, R. S. (2015). Technologies for printing sensors and electronics over large flexible substrates: A review. *IEEE Sensors Journal, 15*(6), 3164–3185. https://doi.org/10.1109/jsen.2014.2375203

[42] M. Konijnenburg et al., "28.4 A battery-powered efficient multi-sensor acquisition system with simultaneous ECG, BIO-Z, GSR, and PPG," *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, San Francisco, CA, USA, 2016, pp. 480-481, doi: 10.1109/ISSCC.2016.7418116.

[43] V. Randazzo, E. Pasero and S. Navaretti, "VITAL-ECG: A portable wearable hospital,"2018 IEEE Sensors Applications Symposium (SAS), Seoul, Korea (South), 2018, pp. 1-6, doi: 10.1109/SAS.2018.8336776.

[44] F. A. Ferreira Marques, D. M. D. Ribeiro, M. F. M. Colunas and J. P. Silva Cunha, .ª real time, wearable ECG and blood pressure monitoring system," *6th Iberian Conference on Information Systems and Technologies (CISTI 2011)*, Chaves, Portugal, 2011, pp. 1-4.

[45] Zaman, M. S., & Morshed, B. I. (2021). A low-power portable scanner for body-worn Wireless Resistive Analog Passive (WRAP) sensors for mHealth applications. *Measurement: Journal of the International Measurement Confederation, 177*(109214), 109214. https://doi.org/10.1016/j.measurement.2021.109214

[46] V. Randazzo, J. Ferretti and E. Pasero, ECG WATCH: a real time wireless wearable ECG, *2019 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*, Istanbul, Turkey, 2019, pp. 1-6, doi: 10.1109/MeMeA.2019.8802210.

[47] B. M. G. Rosa and G. Z. Yang, A Flexible Wearable Device for Measurement of Cardiac, Electrodermal, and Motion Parameters in Mental Healthcare Applications, in *IEEE Journal of Biomedical and Health Informatics*, vol. 23, no. 6, pp. 2276-2285, Nov. 2019, doi: 10.1109/JBHI.2019.2938311.

[48] A. Moin et al., An EMG Gesture Recognition System with Flexible High-Density Sensors and Brain-Inspired High-Dimensional Classifier, *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, Florence, Italy, 2018, pp. 1-5, doi: 10.1109/ISCAS.2018.8351613.

[49] Park, J., Kim, J., Kim, S.-Y., Cheong, W. H., Jang, J., Park, Y.-G., Na, K., Kim, Y.-T., Heo, J. H., Lee, C. Y., Lee, J. H., Bien, F., & Park, J.-U. (2018). Soft, smart contact lenses with integrations of wireless circuits, glucose sensors, and displays. *Science Advances, 4*(1), eaap9841. https://doi.org/10.1126/sciadv.aap9841

[50] Taguchi, N., Taguchi, S., Ishizuki, S., & Ito, H. (2020). Contact dermatitis associated with the Bispectral indexTM sensor: a case report. *JA Clinical Reports, 6*(1), 87. https://doi.org/10.1186/s40981-020-00393-w

[51] Xiao, X., Wu, G., Zhou, H., Qian, K., & Hu, J. (2017). Preparation and property evaluation of conductive hydrogel using poly (vinyl alcohol)/polyethylene glycol/graphene oxide for human electrocardiogram acquisition. *Polymers, 9*(7), 259. https://doi.org/10.3390/polym9070259

[52] *Altium Designer Documentation*. Altium. https://www.altium.com/documentation/altium-designer. Accedido: Febrero 2023.

[53] *Ultra Librarian main page*. Ultra Librarian. https://www.ultralibrarian.com

[54] *LTspice - All things LTspice*. Analog Devices. https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html. Accedido: Abril 2023.

[55] *MATLAB*. MathWorks. https://es.mathworks.com/products/matlab.html. Accedido: Abril 2023.

[56] Cypress, *PSoC Creator User Guide*, Agosto 2022, https://www.infineon.com/dgdl/Infineon-PSoC_Creator_User_Guide-UserManual-v14_00-EN.pdf?fileId=8ac78c8c7ddc01d7017ddd0a1f485c96

[57] *PSoC™ Creator*. Infineon. https://www.infineon.com/cms/en/design-support/tools/sdk/psoc-software/psoc-creator/. Accedido: Marzo 2023.

[58] Cypress, *Universal Asynchronous Receiver Transmitter (UART)*, https://www.infineon.com/dgdl/Infineon-Component_UART_V2.50-Software%20Module%20Datasheets-v02_05-EN.pdf?fileId=8ac78c8c7d0d8da4017d0e7fcaf71181

[59] *Introducción a Android Studio*. Developers Android. https://developer.android.com/studio/intro?hl=es-419. Accedido: Abril 2023.

[60] *Descripción general del manifiesto de una app*. Developers Android. https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419. Accedido: Abril 2023.

[61] Ho, W., & Ji, P. (2007). *Optimal Production Planning for PCB Assembly*. Springer.

[62] Bhunia, S., & Tehranipoor, M. (2018). *Hardware security: A hands-on learning approach*. Morgan Kaufmann.

[63] *RMP354 Multilayer Press*. Fortex. http://www.fortex.co.uk/product/rmp13-multilayer-pcb-press/. Accedido: Febrero 2023.

[64] Ben Abdallah, A. (2017). *Advanced multicore systems-on-chip: Architecture, on-chip network, design* (1st ed.). Springer.

[65] M. Siekkinen, M. Hiienkari, J. K. Nurminen and J. Nieminen, "How low energy is bluetooth low energy? Comparative measurements with ZigBee/802.15.4,"2012 *IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, Paris, France, 2012, pp. 232-237, doi: 10.1109/WCNCW.2012.6215496.

[66] Garcia-Espinosa, E., Longoria-Gandara, O., Pegueros-Lepe, I., & Veloz-Guerrero, A. (2018). Power consumption analysis of Bluetooth low energy commercial products and their implications for IoT applications. *Electronics, 7*(12), 386. https://doi.org/10.3390/electronics7120386

[67] Infineon Technologies AG, *PSoC® 5LP: CY8C58LP Family Datasheet*, Julio 2012 [Revisado Noviembre 2019], https://www.infineon.com/dgdl/Infineon-PSoC_5LP_CY8C58LP_Family_Datasheet_Programmable_System-on-Chip_(PSoC_)-DataSheet-v15_00-EN.pdf?fileId=8ac78c8c7d0d8da4017d0ec547013ab9

[68] Infineon Technologies AG, *CYBLE-022001-00, EZ-BLE™ Creator Module*, Febrero 2015 [Revisado Marzo 2021], https://www.mouser.es/datasheet/2/196/CYPR_S_A0011804390_1-3004892.pdf

[69] Johanson Technology, *2450AT18B100E Datasheet Ver 2.0*, Agosto 2016, https://www.farnell.com/datasheets/2173873.pdf

[70] *CY8C5868LTI-LP039*. Digi-Key. https://www.digikey.es/es/products/detail/infineon-technologies/CY8C5868LTI-LP039/3829540?s=N4IgTCBcDaIMYE8AccCsSBsSA2AXAlgLTYAOADAMwCcIAugL5A. Accedido: Febrero 2023.

[71] *CYBLE-022001-00*. Digi-Key. https://www.digikey.es/es/products/detail/cypress-semiconductor-corp/CYBLE-022001-00/5355485. Accedido: Febrero 2023.

[72] *Op Amps and the Importance of Low Quiescent Current*. Analog Devices. https://www.analog.com/en/design-notes/op-amps-and-the-importance-of-low-quiescent-current.html. Accedido: Febrero 2023.

[73] Texas Instruments, *LP8340 Low Dropout, Low IQ, 1.0A CMOS Linear Regulator datasheet*, Febrero 2003 [Revisado Abril 2013], https://www.ti.com/lit/ds/symlink/lp8340.pdf?ts=1675941098092&ref_url=https%253A%252F%252Fwww.google.com%252F

[74] Day, M. (2006). *Understanding low drop out (LDO) regulators*. https://www.ti.com/lit/ml/slup239a/slup239a.pdf?ts=1675877207745&ref_url=https%253A%252F%252Fwww.google.com%252F#:~:text=The%20dropout%20voltage%20is%20the,element%20equals%20the%20output%20voltage.

[75] Texas Instruments, *TPS727 250-mA, Ultralow IQ, Fast Transient Response, RF Low-Dropout Linear Regulator datasheet*, Junio 2009 [Revisado 2015], https://www.ti.com/lit/ds/symlink/tps727.pdf?HQS=dis-mous-null-mousermode-dsf-pf-null-wwe&ts=1675948763451&ref_url=https%253A%252F%252Fwww.mouser.fr%252F

[76] Texas Instruments, *TPS7A03 Nanopower IQ, 200-nA, 200-mA, Low-Dropout Voltage Regulator With Fast Transient Response*, Julio 2019 [Revisado Septiembre 2022], https://www.ti.com/lit/ds/symlink/tps7a03.pdf?ts=1675963418340&ref_url=https%253A%252F%252Fwww.google.com%252F

[77] Texas Instruments, *PS785-Q1 Automotive, 1-A, High-PSRR Low-Dropout Voltage Regulator With High Accuracy and Enable datasheet*, Enero 2021 [Revisado Enero 2022], https://www.ti.com/lit/ds/symlink/tps785-q1.pdf?ts=1675964620233&ref_url=https%253A%252F%252Fwww.google.com%252F

[78] Texas Instruments, *TLV767-Q1 1-A, 16-V Linear Voltage Regulator datasheet*, Abril 2020 [Revisado Diciembre 2020], https://www.ti.com/lit/ds/symlink/tlv767-q1.pdf?ts=1675925981144

[79] Benjamin Mak. (2014). *Basics of Load Switches*. https://www.ti.com/lit/an/slva652a/slva652a.pdf?ts=1676062198335&ref_url=https%253A%252F%252Fwww.google.com%252F

[80] Toshiba, *TCK106AG, TCK107AG, TCK108AG*, Octubre 2015, https://www.mouser.es/datasheet/2/408/TCK106AG_datasheet_en_20151005-1916439.pdf

[81] Texas Instruments, *TPS22908 3.6-V, 1-A, 28-mΩ On-Resistance Load Switch with Controlled Rise Time*, Julio 2012 [Revisado Abril 2015], https://www.ti.com/lit/ds/symlink/tps22908.pdf?ts=1676118396318&ref_url=https%253A%252F%252Fwww.ti.com%252Fpower-management%252Fpower-switches%252Fload-switches%252Fproducts.html

[82] Texas Instruments, *Ultra-Small, Low-Input-Voltage, Low rON Load Switch With Hysteresis Control Input*, Agosto 2010 [Revisado Abril 2015], https://www.ti.com/lit/ds/symlink/tps22934.pdf?ts=1676119728496&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FTPS22934

[83] Analog Devices, *Logic Controlled, 1 A, High-Side Load Switch with Reverse Current Blocking*, Octubre 2011 [Revisado Agosto 2013], https://www.analog.com/media/en/technical-documentation/data-sheets/ADP198.pdf

[84] Granda Miguel, M., & Mediavilla Bolado, E. (2015). *Instrumentación electrónica: transductores y acondicionadores de señal*. Editorial de la Universidad de Cantabria.

[85] *¿Qué son los pares diferenciales y las señales diferenciales?*. Altium. https://resources.altium.com/es/p/what-are-differential-pairs-and-differential-signals. Accedido: Febrero 2023.

[86] Jim Karki, *Understanding Operational Amplifier Specifications*, Enero 2018 [Revisado Julio 2021], https://www.ti.com/lit/an/sloa011b/sloa011b.pdf?ts=1676382850724&ref_url=https%253A%252F%252Fwww.google.com%252F

[87] Texas Instruments, *INA333 Micro-Power (50 μA), Zero-Drift, Rail-to-Rail Out Instrumentation Amplifier*, Julio 2008 [Revisado Diciembre 2015], https://www.ti.com/lit/ds/symlink/ina333.pdf?ts=1676393490906&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FINA333

[88] Texas Instruments, *INA317 Micro-Power (50 μA), Zero-Drift, Rail-to-Rail-Out Instrumentation Amplifier*, Noviembre 2017, https://www.ti.com/lit/ds/symlink/ina317.pdf?ts=1676410511904&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FINA317

[89] Analog Devices, *40 μA Micropower Instrumentation Amplifier in WLCSP Package*, Agosto 2009 [Revisado Septiembre 2016], https://www.analog.com/media/en/technical-documentation/data-sheets/AD8235.pdf

[90] Daniel Nordwood. *The Use and Benefits of Ferrite Beads in Gate Drive Circuits*. Diciembre 2021. https://www.ti.com/lit/an/sluaai2/sluaai2.pdf

[91] Rohm Semiconductor, *Impedance Characteristics of Bypass Capacitor*, Septiembre 2020, https://fscdn.rohm.com/en/products/databook/applinote/common/bypass_capacitor_impedance_characteristics_an-e.pdf

[92] Bowick, C., Ajluni, C., & Blyler, J. (2007). *RF Circuit Design* (2nd ed.). Newnes.

[93] TE Connectivity, *FPC Connector Family*, Mayo 2006, https://www.te.com/commerce/DocumentDelivery/DDEController?Action=srchrtrv&DocNm=1-1773440-6&DocType=Data+Sheet&DocLang=English&DocFormat=pdf&PartCntxt=84953-4

[94] Harwin, *M50-360xx42 Customer Information Sheet*, https://www.mouser.es/datasheet/2/181/M50_360-1133637.pdf

[95] Cypress, *PSoC® MiniProg3 Program and Debug Kit Guide*, https://www.infineon.com/dgdl/Infineon-CY8CKIT-002_MiniProg3_Kit_Guide-UserManual-v01_00-EN.pdf?fileId=8ac78c8c7d0d8da4017d0eec544077af

[96] Harwin, *M52-040000Pxx45*, https://www.mouser.es/datasheet/2/181/M52-040000PXX45-1273472.pdf

[97] *Wurth Elektronik 0.5mm 40 Way FFC Ribbon Cable, White Sheath, 20.5 mm Width, 200mm Length*. RS Components. https://export.rsdelivers.com/es/product/wurth-elektronik/687640200002/wurth-elektronik-05mm-40-way-ffc-ribbon-cable-205/7635596. Accedido: Mayo 2023.

[98] *LP8340ILDX-ADJ/NOPB*. Mouser Electronics. https://www.mouser.es/ProductDetail/Texas-Instruments/LP8340ILDX-ADJ-NOPB?qs=8uBHJDVwVqwyikeCAqBNUA%3D%3D. Accedido: Febrero 2023.

[99] *TCK106AG,LF*. Digi-Key. https://www.digikey.es/es/products/detail/toshiba-semiconductor-and-storage/TCK106AG-LF/5409283. Accedido Febrero 2023.

[100] *INA333*. Texas Instruments. https://www.ti.com/product/INA333?keyMatch=INA333&tisearch=search-everything&usecase=GPN. Accedido: Febrero 2023.

[101] *84953-4*. Digi-Key. https://www.digikey.es/es/products/detail/te-connectivity-amp-connectors/84953-4/1122095. Accedido: Febrero 2023.

[102] *M52-040000P0545*. Farnell. https://es.farnell.com/harwin/m52-040000p0545/macho-vertical-1fila-5-v-as/dp/1099569. Accedido: Febrero 2023.

[103] *M50-3600542*. Farnell. https://es.farnell.com/harwin/m50-3600542/conector-macho-smt-r-a-1-27-mm/dp/1022310?ost=m50-3600542. Accedido: Febrero 2023.

[104] *MMZ1005Y301CT000*. Farnell. https://es.farnell.com/tdk/mmz1005y301c/inductor-0-38ohm-250ma-0402/dp/1669677?st=MMZ1005Y301CT000. Accedido: Febrero 2023.

[105] *ERA2AED102X*. Farnell. https://es.farnell.com/panasonic/era2aed102x/res-1k-0-5-0-063w-0402-pel-c-metal/dp/2324716?st=era2aed102x. Accedido: Febrero 2023.

[106] *ERA-2AEB104X*. Farnell. https://es.farnell.com/panasonic/era-2aeb104x/sistor-metal-film-100kohm-63mw/dp/2394316?st=era-2aeb104x. Accedido: Febrero 2023.

[107] *C0402C104J4RACTU*. Farnell. https://es.farnell.com/kemet/c0402c104j4ractu/conden-0-1-f-16v-5-x7r-0402/dp/2429353?st=C0402C104J4RACTU. Accedido: Febrero 2023.

[108] Cypress, *PSoC®3 and PSoC 5LP – Pin Selection for Analog Designs*, https://www.infineon.com/dgdl/Infineon-AN58304_PSoC_3_and_PSoC_5LP_Pin_Selection_for_Analog_Designs-ApplicationNotes-v09_00-EN.pdf?fileId=8ac78c8c7cdc391c017d072e43f252d4

[109] TDK, *MMZ Series MMZ1005 Type*, https://www.farnell.com/datasheets/84898.pdf

[110] Cypress, *Getting Started with EZ-BLE™ Creator Modules*, https://www.infineon.com/dgdl/Infineon-AN96841_GETTING_STARTED_WITH_EZ-BLE_CREATOR_MODULES-ApplicationNotes-v10_00-EN.pdf?fileId=8ac78c8c7cdc391c017d073ff52c637b&utm_source=cypress&utm_medium=referral&utm_campaign=202110_globe_en_all_integration-application_note

[111] Gacek, A., & Pedrycz, W. (Eds.). (2014). *ECG signal processing, classification and interpretation: A comprehensive framework of computational intelligence* (2012th ed.). Springer.

[112] N. L. Eastman, "Considerations for mixed analog/digital PCB design" Wescon/96, Anaheim, CA, USA, 1996, pp. 297-301, doi: 10.1109/WESCON.1996.554004.

[113] S. Muralikrishna and S. Sathyamurthy, "An overview of digital circuit design and PCB design guidelines - An EMC perspective" *2008 10th International Conference on Electromagnetic Interference & Compatibility*, Bangalore, India, 2008, pp. 567-573.

[114] Bronzino, J. D., & Peterson, D. R. (2018). *Biomedical Engineering Fundamentals* (2nd ed.). CRC Press.

[115] Kaplan Berkaya, S., Uysal, A. K., Sora Gunal, E., Ergin, S., Gunal, S., & Gulmezoglu, M. B. (2018). A survey on ECG analysis. *Biomedical Signal Processing and Control*, 43, 216–235. https://doi.org/10.1016/j.bspc.2018.03.003

[116] Crawford, Jacqui, & Doherty, L. (2012). *Practical aspects of ECG recording*. M&K Update.

[117] *ECG Lead positioning*. Life in the fastline. https://litfl.com/ecg-lead-positioning/. Accedido: Abril 2023.

[118] Castillo, E., Morales, D. P., Botella, G., García, A., Parrilla, L., & Palma, A. J. (2013). Efficient wavelet-based ECG processing for single-lead FHR extraction. *Digital Signal Processing, 23*(6), 1897–1909. https://doi.org/10.1016/j.dsp.2013.07.010

[119] Toral, V., García, A., Romero, F. J., Morales, D. P., Castillo, E., Parrilla, L., Gómez-Campos, F. M., Morillas, A., & Sánchez, A. (2019). Wearable system for biosignal acquisition and monitoring based on reconfigurable technologies. *Sensors (Basel, Switzerland)*, 19(7), 1590. https://doi.org/10.3390/s19071590

[120] *BlueSensor L*. Ambu®. https://www.ambu.es/cardiologia/electrodos-ecg/producto/ambu-bluesensor-l. Accedido: Mayo 2023.

[121] Infineon, *Programmable Gain Amplifier (PGA)*, Noviembre 2017, https://www.infineon.com/dgdl/Infineon-Component_Programmable_Gain_Amplifier_(PGA)_V2.0-Software%20Module%20Datasheets-v02_00-EN.pdf?fileId=8ac78c8c7d0d8da4017d0e8c76dd1b2a

[122] Chan, E. D., Chan, M. M., & Chan, M. M. (2013). Pulse oximetry: understanding its basic principles facilitates appreciation of its limitations. *Respiratory Medicine*, 107(6), 789–799. https://doi.org/10.1016/j.rmed.2013.02.004

[123] Tsiakaka O, Gosselin B, Feruglio S. Source-Detector Spectral Pairing-Related Inaccuracies in Pulse Oximetry: Evaluation of the Wavelength Shift. *Sensors* (Basel). 2020 Jun 10;20(11):3302. doi: 10.3390/s20113302. PMID: 32532116; PMCID: PMC7309008.

[124] *Diseñar un oxímetro de pulso de bajo costo usando componentes de venta*. DigiKey. https://www.digikey.com/es/articles/design-a-low-cost-pulse-oximeter-using-off-the-shelf-components. Accedido: Abril 2023.

[125] Petersen, C. L., Chen, T. P., Ansermino, J. M., & Dumont, G. A. (2013). Design and evaluation of a low-cost smartphone pulse oximeter. *Sensors (Basel, Switzerland)*, 13(12), 16882–16893. https://doi.org/10.3390/s131216882

[126] G. Di, X. Tang and W. Liu, .ª Reflectance Pulse Oximeter Design Using the MSP430F149," *2007 IEEE/ICME International Conference on Complex Medical Engineering*, Beijing, China, 2007, pp. 1081-1084, doi: 10.1109/ICCME.2007.4381907.

[127] BIOFY, *SFH7072 Sensor Version 1.2*, Julio 2020, https://docs.rs-online.com/5112/A700000007243436.pdf

[128] *Intelligent LED Solutions BIOFY Sensor Version 1.0 for SFH 7072*. RS Components. https://uk.rs-online.com/web/p/sensor-development-tools/1757444. Accedido: Abril 2023.

[129] Akshay Bhat, *Stabilize Your Transimpedance Amplifier*, Febrero 2012, https://pdfserv.maximintegrated.com/en/an/TUT5129.pdf

[130] *How to Design Stable Transimpedance Amplifiers for Automotive and Medical System*. DigiKey. https://www.digikey.es/es/articles/how-to-design-stable-transimpedance-amplifiers-automotive-medical-systems. Accedido: Abril 2023.

[131] Cypress, *Trans-Impedance Amplifier (TIA)*, Noviembre 2011, https://www.infineon.com/dgdl/Infineon-Component_TIA_V1.80-Software%20Module%20Datasheets-v02_00-EN.pdf?fileId=8ac78c8c7d0d8da4017d0e7e89831085

[132] Banik, P. P., Hossain, S., Kwon, T.-H., Kim, H., & Kim, K.-D. (2020). Development of a wearable reflection-type pulse oximeter system to acquire clean PPG signals and measure pulse rate and SpO2 with and without finger motion. *Electronics, 9*(11), 1905. https://doi.org/10.3390/electronics9111905

[133] *Introducción a las actividades*. Developers Android. https://developer.android.com/guide/components/activities/intro-activities?hl=es-419#:~:text=An%20activity%20provides%20the%20window,one%20screen%20in%20an%20app. Accedido: Mayo 2023.

[134] *Introducción a las actividades*. Developers Android. https://developer.android.com/google/play/requirements/target-sdk?hl=es-419. Accedido: Mayo 2023.

[135] *Permisos en Android*. Developers Android. https://developer.android.com/guide/topics/permissions/overview?hl=es-419. Accedido: Mayo 2023.

[136] *Introducción a Filter Designer*. MathWorks. https://es.mathworks.com/help/signal/ug/introduction-to-filter-designer.html. Accedido: Mayo 2023.

[137] *The electronic part search engine*. Octopart. https://octopart.com. Accedido: Mayo 2023.

Apéndice A

Costes del proyecto

En este anexo se recopila el coste monetario del proyecto, esto incluye tanto el precio estimado de los componentes individuales, lo cual se puede consultar en el apartado Componentes electrónicos, como el coste total incluyendo a la PCB y el ensamblaje de la circuitería en ella, así como otros gastos en Total.

A.1. Componentes electrónicos

Para la estimación del coste de los componentes individuales se ha utilizado la herramienta en línea para generación de listas de materiales o BOMs (*Bill of materials*) disponible en la web "Octopart" [137]. Esta no es más que una lista de todos los componentes necesarios, modelos en específico y cantidad necesaria de cada uno, para la construcción de un circuito electrónico, añadiendo una estimación del coste de cada uno. La lista de materiales para este trabajo se puede consultar en la Tabla A.1.

Los gastos estimados en este apartado se refieren a únicamente los componentes electrónicos para la fabricación de un único dispositivo de procesamiento central diseñado en los capítulos 3 y 4. A dichos costes habrá que añadir los gastos fabricación de la PCB y ensamblaje de los componentes en esta, así como los posteriores costes de envío.

A.2. Total

La PCB se ha encargado al fabricante PCBWay, habiéndose solicitado un total de 3 PCBs más el respectivo ensamblaje de los componentes en cada una de ellas y las tasas de envío.

En la Tabla A.2 se proporcionan los gastos de fabricación de la propia PCB sin componentes y el posterior precio de compra y ensamblaje de estos. Dado que el fabricante índica la factura en dolares estadounidenses (USD), se puede consultar los costes tanto en esta moneda, como en euros (EUR). Hay que tener en cuenta que la conversión entre monedas se ha realizado a Mayo de 2023 y puede cambiar en el futuro, al igual que ocurre con el precio de componentes y fabricación indicados en este trabajo.

Los gastos totales se encuentran resumidos en la Tabla A.3, donde se incluyen al margen de los propios coste de fabricación, el resto de tasas. Es importante tener en cuenta que los costes se ven influenciados a la alta debido a que al ser un tirada pequeña para prototipaje, el precio de cada dispositivo individual se incrementa. En caso de tiradas a gran escala para, por ejemplo, comercialización, el precio unitario se reduciría de manera considerable.

Compoenente	Modelo	Descripción	Unidades	Coste/unidad (EUR)	Coste total (EUR)
Procesador	PSoC 5LP CY8C5868LTI-LP039	MCU 32-Bit PSoC ARM Cortex	1	24,5400	24,5400
Transmisor BLE	CYBLE-022001-00	Módulo PSoC Bluetooth	1	10,6337	10,6337
Regulador de tensión	LP8340CLD-3.3/NOPB	IC, Reg, Ldo Cmos, 1A, 3.3V	1	1,5000	1,5000
Load switch	TCK106AG	Soporte de fuente de alimentación, fijo, 1 canal, CMOS	2	0,4400	0,8800
Amplificador de instrumentación	INA333AIDRGT	Amp instr, bajo consumo, Zero-drift, 8-SON	1	5,8852	5,8852
Conectores cables planos	84953-4	Conector hembra, 4 entradas	4	0,2750	1,1000
Puertos de programación	M50-3600542	Conector macho SMD, 10 pines, 2x5	1	1,3300	1,3300
	M52-040000P0545	Conector macho SMD, 5 pines, 1x5	1	0.8342	0.8342
Condensadores	C0402C104J4RACTU	Condensador cerámico, 0.1 uF, 16 V, ± 5 %, 0402	9	0,0940	0.8460
	CC0402KRX5R6BB105	Condensador cerámico, 1 uF, 16 V, ± 20 %, 0402	4	0,0917	0,3667
Resistencias	ERA-2AEB104X	100 kOhm, 0.1 %, 0.063W, 0402	1	0,3850	0,3850
	ERA2AED102X	1 kOhm, 0.5 %, 0.063W, 0402	2	0,2070	0,4140
Núcleo de ferrita	MMZ1005Y301CT000	300 kOhm, 100 MHz	2	0,0110	0,0220
				Total (EUR)	**48,74**

Tabla A.1: Costes de los componentes electrónicos individuales para la fabricación de uno de los dispositivos de procesamiento central desarrollados en este trabajo.

	PCB sin componentes		**Componentes y ensamblaje**	
Moneda	**Coste/unidad**	**Coste total**	**Coste/unidad**	**Coste total**
USD	10,36	20,72	137,70	413,10
EUR	9,52	19,03	126,51	379,54

Tabla A.2: Costes de fabricación de la PCB y el ensamblaje de los componentes en ella.

Gasto	**USD**	**EUR**
PCB + Componentes y Ensamblaje	433,82	398,44
Envío	46,65	42,85
Tarifa bancaria	20,57	18,89
Descuento	-30	-27,55
Total	**471,04**	**432,62**

Tabla A.3: Costes totales de las 3 PCBs encargadas.

Apéndice B

Configuración de los módulos en PSoC Creator

En este anexo se incluye la configuración gráfica que se ha hecho en PSoC Creator para los componentes presentes en el PSoC 5LP tanto para la adquisición del ECG como del PPG.

B.1. Configuración para el ECG

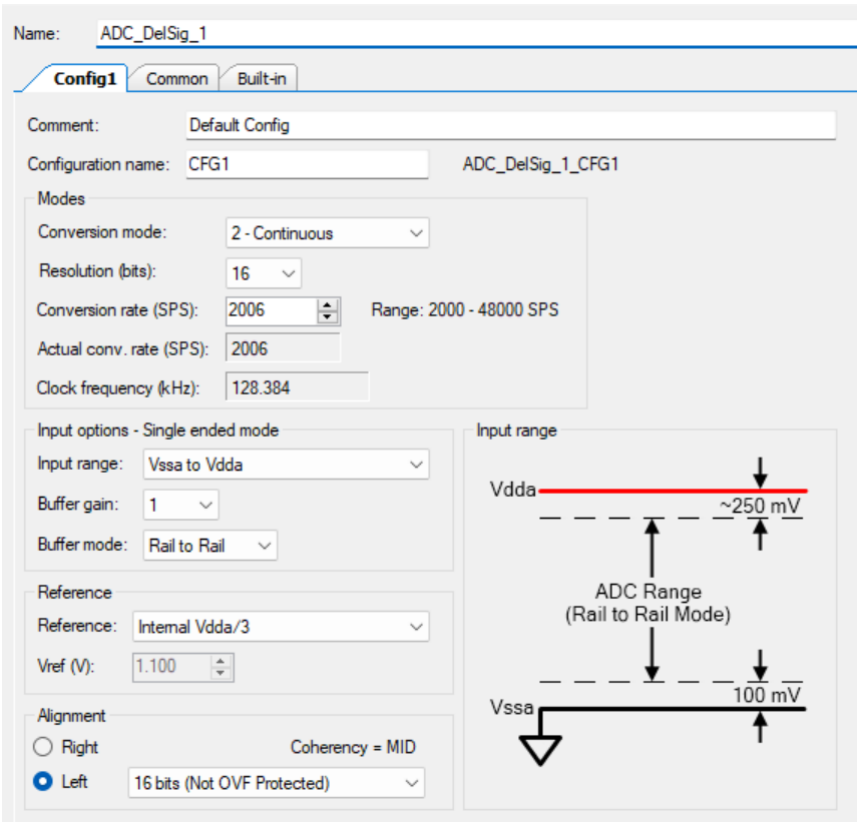
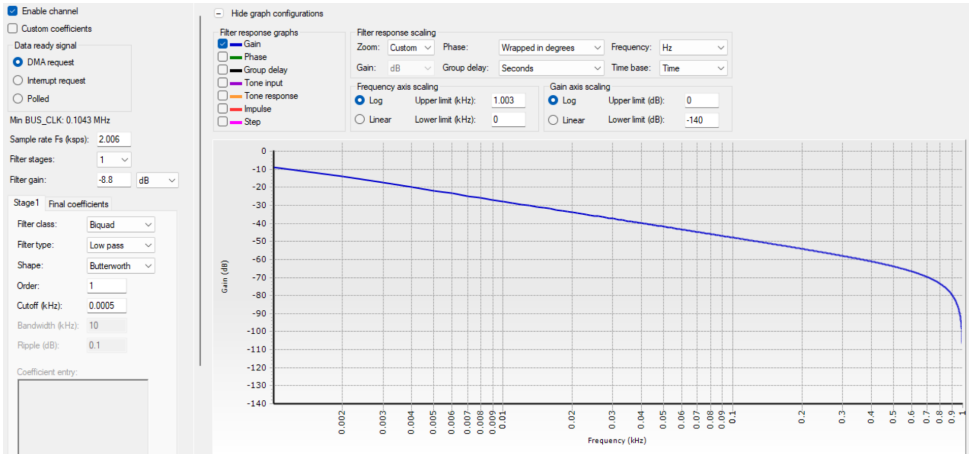
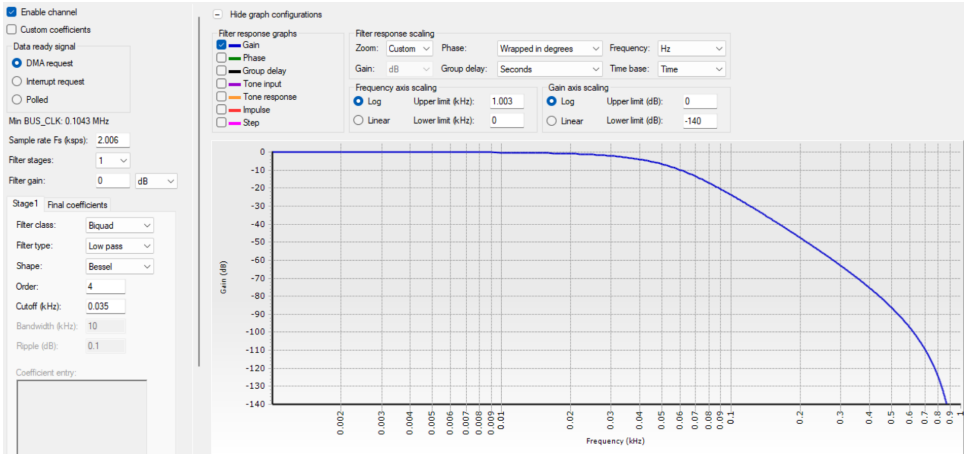


Figura B.1: Configuración del ADC delta-sigma para la adquisición del ECG en PSoC Creator.



(a) Canal A.



(b) Canal B.

Figura B.2: Configuración de los filtros digitales usados en el tratamiento de las señales del ECG.

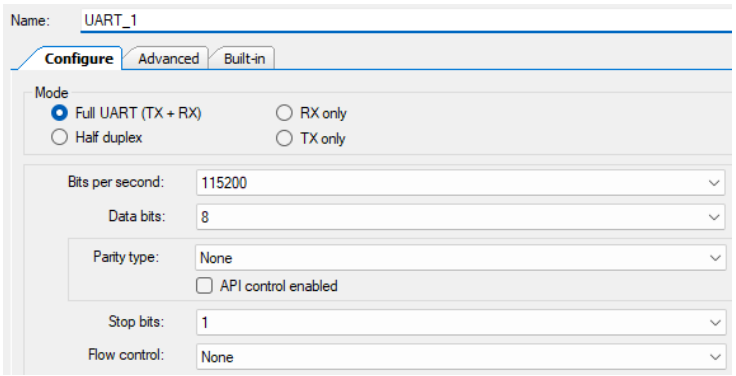


Figura B.3: Configuración del bloque UART para la adquisición del ECG en PSoC Creator.

B.2. Configuración para el PPG

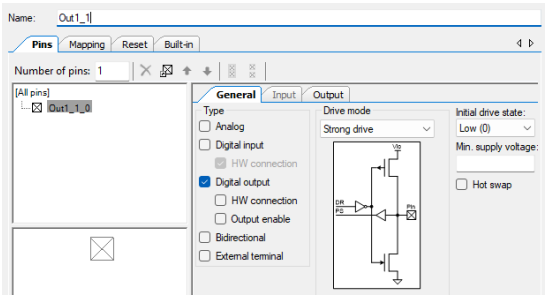


Figura B.4: Configuración de los pines digitales empleados para el control del encendido y apagado de los LEDs.

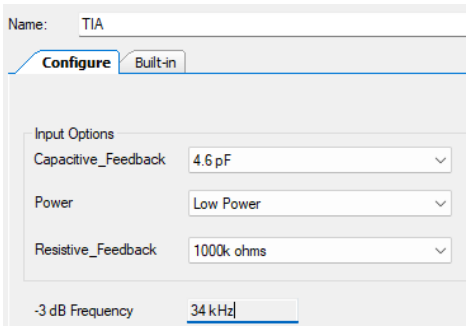


Figura B.5: Configuración del bloque TIA.

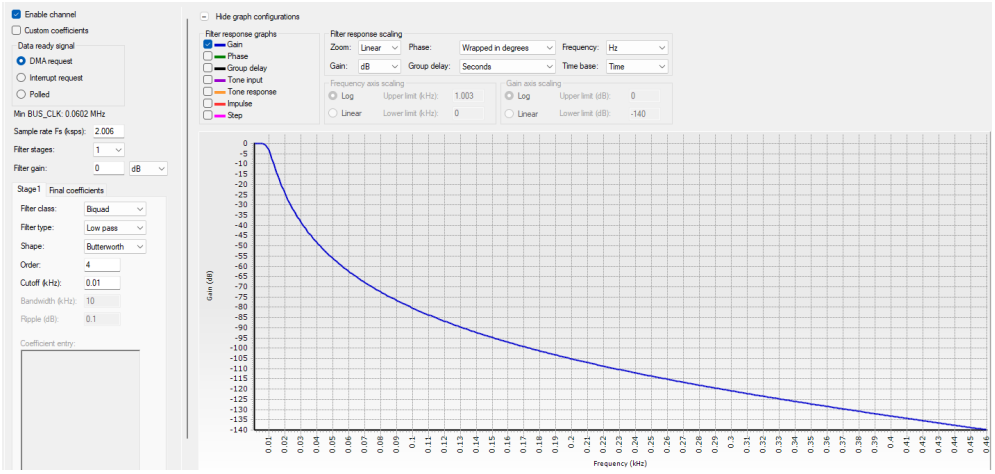


Figura B.6: Configuración del filtro digital paso baja empleado para la adquisición del nivel de oxígeno en sangre.

Apéndice C

Firmware usado en el PSoC 5LP

En este anexo se puede encontrar el *firmware* desarrollado en el lenguaje C para el control del PSoC 5LP tanto para la adquisición del ECG como del porcentaje de SpO_2.

C.1. Firmware: ECG

```
#include "project.h"

volatile uint16 out_array[2000];
volatile uint16 out_array2[2000];

volatile uint8 DMA_3_Chan;
volatile uint8 uart_array [4002];
volatile uint8 send_flag = 0;
volatile uint8 Rec_flac = 0;
volatile uint8 j = 0;


void DMA_1_Startup(void){
        /* Defines for DMA_1 */
        #define DMA_1_BYTES_PER_BURST 4
        #define DMA_1_REQUEST_PER_BURST 1
        #define DMA_1_SRC_BASE (CYDEV_PERIPH_BASE)
        #define DMA_1_DST_BASE (CYDEV_PERIPH_BASE)

        /* Variable declarations for DMA_1 */
        uint8 DMA_1_Chan;
        uint8 DMA_1_TD[2];

        /* DMA Configuration for DMA_1 */
        DMA_1_Chan = DMA_1_DmaInitialize(DMA_1_BYTES_PER_BURST, DMA_1_REQUEST_PER_BURST,
        HI16(DMA_1_SRC_BASE), HI16(DMA_1_DST_BASE));
        DMA_1_TD[0] = CyDmaTdAllocate();
        DMA_1_TD[1] = CyDmaTdAllocate();
        CyDmaTdSetConfiguration(DMA_1_TD[0], 2, DMA_1_TD[1], CY_DMA_TD_AUTO_EXEC_NEXT);
        CyDmaTdSetConfiguration(DMA_1_TD[1], 2, DMA_1_TD[0], 0);
        CyDmaTdSetAddress(DMA_1_TD[0], LO16((uint32)ADC_DelSig_1_DEC_SAMP_PTR), LO16((uint32)
            Filter_1_STAGEA_PTR));
        CyDmaTdSetAddress(DMA_1_TD[1], LO16((uint32)ADC_DelSig_1_DEC_SAMP_PTR), LO16((uint32)
            Filter_1_STAGEB_PTR));
        CyDmaChSetInitialTd(DMA_1_Chan, DMA_1_TD[0]);
        CyDmaChEnable(DMA_1_Chan, 1);
}


void DMA_2_Startup(void){
        /* Defines for DMA_2 */
        #define DMA_2_BYTES_PER_BURST 1
        #define DMA_2_REQUEST_PER_BURST 1
        #define DMA_2_SRC_BASE (CYDEV_PERIPH_BASE)
        #define DMA_2_DST_BASE (CYDEV_PERIPH_BASE)

        /* Variable declarations for DMA_2 */
        uint8 DMA_2_Chan;
        uint8 DMA_2_TD[1];

        /* DMA Configuration for DMA_2 */
        DMA_2_Chan = DMA_2_DmaInitialize(DMA_2_BYTES_PER_BURST, DMA_2_REQUEST_PER_BURST,
        HI16(DMA_2_SRC_BASE), HI16(DMA_2_DST_BASE));
        DMA_2_TD[0] = CyDmaTdAllocate();
        CyDmaTdSetConfiguration(DMA_2_TD[0], 1, CY_DMA_END_CHAIN_TD, 0);
```

```
54          CyDmaTdSetAddress(DMA_2_TD[0], LO16((uint32)Filter_1_HOLDAM_PTR), LO16((uint32)
                VDAC8_1_Data_PTR));
55          CyDmaChSetInitialTd(DMA_2_Chan, DMA_2_TD[0]);
56          CyDmaChEnable(DMA_2_Chan, 1);
57  }
58
59
60  void DMA_3_Startup(void){
61          /* Defines for DMA_3 */
62          #define DMA_3_BYTES_PER_BURST 2
63          #define DMA_3_REQUEST_PER_BURST 1
64          #define DMA_3_SRC_BASE (CYDEV_PERIPH_BASE)
65          #define DMA_3_DST_BASE (CYDEV_SRAM_BASE)
66
67          /* Variable declarations for DMA_3 */
68          uint8 DMA_3_TD[2];
69
70          /* DMA Configuration for DMA_3 */
71          DMA_3_Chan = DMA_3_DmaInitialize(DMA_3_BYTES_PER_BURST, DMA_3_REQUEST_PER_BURST,
72          HI16(DMA_3_SRC_BASE), HI16(DMA_3_DST_BASE));
73          DMA_3_TD[0] = CyDmaTdAllocate();
74          DMA_3_TD[1] = CyDmaTdAllocate();
75
76          CyDmaTdSetConfiguration(DMA_3_TD[0], 4000, DMA_3_TD[1], CY_DMA_TD_INC_DST_ADR |
                DMA_3__TD_TERMOUT_EN);
77          CyDmaTdSetConfiguration(DMA_3_TD[1], 4000, DMA_3_TD[0], CY_DMA_TD_INC_DST_ADR |
                DMA_3__TD_TERMOUT_EN);
78
79          CyDmaTdSetAddress(DMA_3_TD[0], LO16((uint32)Filter_1_HOLDB_PTR), LO16((uint32)&
                out_array[0]));
80          CyDmaTdSetAddress(DMA_3_TD[1], LO16((uint32)Filter_1_HOLDB_PTR), LO16((uint32)&
                out_array2[0]));
81
82          CyDmaChSetInitialTd(DMA_3_Chan, DMA_3_TD[0]);
83  }
84
85
86  CY_ISR(Filter_isr){
87          /*
88          Adapt the 16 bits to 8bits and send the 5000 samples of the array by the UART
89          */
90
91          if(Rec_flac == 0){  // Takes 1 of each 2 samples (fs/2) and adapt to 8 bits
92                  for (int i=0; i<2000; i+=2){
93
94                          uart_array[i] = out_array[i]>> 8 ;
95                          uart_array[i+1] = (uint8) out_array[i];
96                  }
97                  Rec_flac = 1;
98          }
99          else{  // Takes 1 of each 2 samples (fs/2) and adapt to 8 bits
100                 for (int i=0; i<2000; i+=2){
101
102                         uart_array[2000 + i] = out_array2[i]>> 8 ;
103                         uart_array[2000 + i+1] = (uint8) out_array2[i];
104                 }
105                 Rec_flac = 0;
106                 send_flag = 1;
107                 j++;
108         }
109         if(j == 30){  // Each iteration represent 2 seconds of the signal
110                 CyDmaChDisable(DMA_3_Chan);
111                 j = 0;
112         }
113 }
114
115
116 CY_ISR(Uart_input){
117         /*
118         Adapt the 16 bits to 8bits and send the 5000 samples of the array by the UART
119         */
120
121         char rec;
122         rec = UART_1_GetByte();
123
124         switch (rec){
125                 case 'a':
126                 Filter_1_ClearInterruptSource();
127                 CyDmaChEnable(DMA_3_Chan,1);
128                 break;
129         }
130
131         UART_1_ReadRxStatus();
132         UART_1_ClearRxBuffer();
133 }
134
135
```

```
136 int main(void){
137
138         CyGlobalIntEnable; /* Enable global interrupts. */
139
140         ADC_DelSig_1_Start();
141         PGA_1_Start();
142         PGA_2_Start();
143         Opamp_1_Start();
144         PGA_Inv_1_Start();
145         PGA_Inv_3_Start();
146         VDAC8_1_Start();
147         Filter_1_Start();
148         UART_1_Start();
149
150         Filter_1_SetCoherency(Filter_1_CHANNEL_A,Filter_1_KEY_MID);
151         Filter_1_SetCoherency(Filter_1_CHANNEL_B,Filter_1_KEY_MID);
152
153         DMA_1_Startup();
154         DMA_2_Startup();
155         DMA_3_Startup();
156
157         Filter_end_StartEx(Filter_isr);
158         uart_isr_StartEx(Uart_input);
159
160         ADC_DelSig_1_StartConvert();
161
162         for(;;){
163                 if(send_flag == 1){
164                         //Envio 2000 muestras de 16bits a (Fs/2)
165                         uart_array[4000]= 10;
166                         uart_array[4001]= 13;
167                         UART_1_PutArray(uart_array, (uint16)4000);
168                         send_flag = 0;
169                 }
170         }
171 }
```

Listing C.1: Código fuente C para implementación del ECG

C.2. Firmware: SpO_2

Se hace necesario hacer un anotación respecto del comportamiento que sigue la interrupción que se activa cuando el filtro ha enviado 4000 bytes, denominada como *CY_ISR(FilterDone)* en el código fuente:

Para empezar, se utilizan dos vectores distintos para almacenar las muestras filtradas: *samples1[]* y *samples2[]*, de una longitud de 2000 posiciones cada uno. El motivo de que los vectores sean de este tamaño, es el hecho de que se está muestreado con una resolución de 16 bits a una frecuencia de 2000 muestras/segundo, lo que implica que cada uno de estos vectores es capaz de almacenar 1 segundo de adquisición. Se han configurado adicionalmente otros dos vectores, *RED_samples[]* e *IR_samples[]*, de una longitud de 8000 posiciones cada uno, esto implica que en total se guardan 4 segundos de señal en los cuales se encontraba en operación el LED rojo y otros 4 en los que se utilizaba el infrarrojo.

Se comienza almacenando el primer segundo de adquisición en el vector *samples1[]*, cuando este se ha llenado, se empiezan a copiar sus muestras en los vectores *RED_samples[]* o *IR_samples[]* según corresponda, y se indica que el siguiente segundo de señal se grabe sobre el vector *samples2[]*. De esta manera se consigue dar mayor fluidez al programa, ya que mientras un conjunto de muestras se está copiando en el vector intermedio (*samples1[]* o *samples2[]*), el conjunto anterior se copia al vector final correspondiente (*RED_samples[]* o *IR_samples[]*) que se usará para las operaciones del cálculo del ratio de absorción (R). La variable *last_index* tiene el objetivo de almacenar hasta que posición se han rellenado los dos vectores finales, de manera que no se sobrescriban las muestras más antiguas.

La variable *count* se utiliza para saber cuantos segundos de adquisición han transcurrido, cada vez que se suma 1 a esta variable es que ha pasado 1 segundo. Adicionalmente, si su valor es par, las muestras se copian en *samples1[]*, mientras que si es impar se copian en *samples2[]*. El valor del contador también se utiliza para saber si las muestras se tienen que copiar en *RED_samples[]* o en *IR_samples[]*. Durante los primeros 4 segundos se almacenan muestras provenientes del LED

rojo y durante los 4 siguientes del infrarrojo, cada conjunto en su vector correspondiente.

Al transcurrir 4 segundos se apaga el LED rojo y se enciende el infrarrojo. Cuando transcurren 8, se hace el cálculo del ratio de absorción, se vuelve a intercambiar que LEDs están encendido y apagado y se resetean las variables de manera que se tome una nueva adquisición de 8 segundos automáticamente.

```
/*
This program allows the acquisition of the SpO2 using 2 LEDs and 1
photodiode. The result is send through UART communication.
*/

#include "project.h"
#include <math.h>
#include <stdbool.h>

#define LED_ON 0u
#define LED_OFF 1u
#define LENGTH_ALL_SAMPLES 2000        // Length of the array send from DMA2
#define LENGTH_SEPARATED_SAMPLES 8000  // Length of the complete array of samples for one of the LEDs
#define A 110                          // Coeficients used to calculate R
#define B 25

volatile uint16 samples1[LENGTH_ALL_SAMPLES];
volatile uint16 samples2[LENGTH_ALL_SAMPLES];

bool send_flag = false;           // Control if the value of R is ready
bool acquisition_active = false;  // Control if the adquisition of values is ON

uint16 RED_samples[LENGTH_SEPARATED_SAMPLES]; // Stores samples only from the RED LED
uint16 IR_samples[LENGTH_SEPARATED_SAMPLES];  // Stores samples only from the IR LED
uint16 R = 0;                                 // Ratio of absorbance
uint16 count = 0;                             // Count how many times 2000 samples has been stored

float last_measure = 0;  // Measure from the last adquisition

// Variables declaration for DMA_1 and DMA_2
uint8 DMA_1_Chan;
uint8 DMA_1_TD[1];
uint8 DMA_2_Chan;
uint8 DMA_2_TD[2];


void DMA_1_Init(){
        /* Defines for DMA_1 */

        #define DMA_1_BYTES_PER_BURST 2
        #define DMA_1_REQUEST_PER_BURST 1
        #define DMA_1_SRC_BASE (CYDEV_PERIPH_BASE)
        #define DMA_1_DST_BASE (CYDEV_PERIPH_BASE)

        /* DMA Configuration for DMA_1 */
        DMA_1_Chan = DMA_1_DmaInitialize(DMA_1_BYTES_PER_BURST, DMA_1_REQUEST_PER_BURST,
        HI16(DMA_1_SRC_BASE), HI16(DMA_1_DST_BASE));
        DMA_1_TD[0] = CyDmaTdAllocate();
        CyDmaTdSetConfiguration(DMA_1_TD[0], 2, DMA_1_TD[0], 0);
        CyDmaTdSetAddress(DMA_1_TD[0], LO16((uint32)ADC_DelSig_DEC_SAMP_PTR), LO16((uint32)
            Filter_1_STAGEA_PTR));
        CyDmaChSetInitialTd(DMA_1_Chan, DMA_1_TD[0]);
        CyDmaChEnable(DMA_1_Chan, 1);
}


void DMA_2_Init(){
        /* Defines for DMA_2 */

        #define DMA_2_BYTES_PER_BURST 2
        #define DMA_2_REQUEST_PER_BURST 1
        #define DMA_2_SRC_BASE (CYDEV_PERIPH_BASE)
        #define DMA_2_DST_BASE (CYDEV_SRAM_BASE)

        /* DMA Configuration for DMA_2 */
        DMA_2_Chan = DMA_2_DmaInitialize(DMA_2_BYTES_PER_BURST, DMA_2_REQUEST_PER_BURST,
        HI16(DMA_2_SRC_BASE), HI16(DMA_2_DST_BASE));
        DMA_2_TD[0] = CyDmaTdAllocate();
        DMA_2_TD[1] = CyDmaTdAllocate();
        CyDmaTdSetConfiguration(DMA_2_TD[0], 4000, DMA_2_TD[1], DMA_2__TD_TERMOUT_EN |
            CY_DMA_TD_INC_DST_ADR);
        CyDmaTdSetConfiguration(DMA_2_TD[1], 4000, DMA_2_TD[0], DMA_2__TD_TERMOUT_EN |
            CY_DMA_TD_INC_DST_ADR);
        CyDmaTdSetAddress(DMA_2_TD[0], LO16((uint32)Filter_1_HOLDA_PTR), LO16((uint32)&samples1
            [0]));
```

```
72          CyDmaTdSetAddress(DMA_2_TD[1], LO16((uint32)Filter_1_HOLDA_PTR), LO16((uint32)&samples2
                [0]));
73          CyDmaChSetInitialTd(DMA_2_Chan, DMA_2_TD[0]);
74  }
75
76
77  float calculateMean (uint16 samples_array[]){
78          /* Calculates the mean of the samples vector
79
80          Inputs ->
81          - Array of samples for one light
82          */
83
84          uint32 sum = 0;
85
86          for(int i = 0; i < LENGTH_SEPARATED_SAMPLES; i++){
87                  sum += samples_array[i];
88          }
89
90          return (float) sum/LENGTH_SEPARATED_SAMPLES;
91  }
92
93
94  float calculateRMS (uint16 samples_array[], float DC){
95          /* Calculates the RMS of the signal
96
97          Inputs ->
98          - Array of samples from the filter
99          - Mean of the samples
100         */
101
102         uint32 sum = 0;
103
104         for(int i = 0; i < LENGTH_SEPARATED_SAMPLES; i++){
105                 sum += pow(samples_array[i] - DC, 2);
106         }
107
108         return (float) sqrt(sum/LENGTH_SEPARATED_SAMPLES);
109 }
110
111
112 float calculateOxygen(){
113         /* Calculate the percentaje of oxigen in blood using the coef R (ratio of ratios) */
114
115         float DC_RED, DC_IR, AC_RED, AC_IR;
116
117         DC_RED = calculateMean(RED_samples);
118         DC_IR = calculateMean(IR_samples);
119
120         AC_RED = calculateRMS(RED_samples, DC_RED);
121         AC_IR = calculateRMS(IR_samples, DC_IR);
122
123         return A - (B * (AC_RED/DC_RED) / (AC_IR/DC_IR));
124 }
125
126
127 CY_ISR(UARTInput){
128         /* Recieved the start order from the UART conection. This interrupt is call when
129         an instruction is received from the BLE module */
130
131         char order;
132
133         if (UART_1_ReadRxStatus() & (uint8)UART_1_RX_STS_FIFO_NOTEMPTY){
134                 order = (char)UART_1_ReadRxData();
135
136                 switch(order){
137                         case 'a':
138                         UART_1_PutString("Starting Adquisition...\r\n");
139
140                         Filter_1_ClearInterruptSource();
141                         CyDmaChEnable(DMA_2_Chan, 1);
142
143                         // Both LEDs turn off at the start
144                         Out1_1_Write(LED_ON);
145                         Out1_2_Write(LED_OFF);
146
147                         count = 0;  // Restart timer counter
148                         last_measure = 0;
149                         acquisition_active = true;
150
151                         break;
152
153                         case 's':
154                         UART_1_PutString("Stopping Adquisition...\r\n");
155
156                         Out1_1_Write(LED_OFF);
157                         Out1_2_Write(LED_OFF);
```

```
158
159                         CyDmaChDisable(DMA_2_Chan);
160
161                         acquisition_active = false;
162
163                         break;
164
165                 } // end switch
166         }
167
168         UART_1_ClearRxBuffer();
169 }
170
171
172 CY_ISR(FilterDone){
173         /* Stores the data from each LED in a different array.
174
175         This interrupt is call each time the filter has send 4000 bytes,
176         so this interrupt is call each 1 seconds (2000 samples/second of 2 bytes/sample). */
177
178         static uint16 i, j, last_index = 0;
179         float R_aux;
180
181         // Turn ON LED IR
182         if(count == 4 && acquisition_active){
183                 Out1_1_Write(LED_OFF); // RED
184                 Out1_2_Write(LED_ON);  // IR
185         }
186
187         // Fill samples arrays
188         if(count % 2 == 0 && count < 8){ // "count" is even
189                 j = 0;
190                 for(i = last_index; i < last_index + LENGTH_ALL_SAMPLES; i++){
191                         if(count <= 3){  // Red
192                                 RED_samples[i] = samples1[j];
193                         }
194                         else{  // IR
195                                 //select_LED = LED_IR;
196                                 IR_samples[i] = samples1[j];
197                         }
198                         j++;
199                 } // end for
200
201                 last_index += LENGTH_ALL_SAMPLES;
202                 count++;
203
204                 if(count > 3){ // Reset the index for the IR LED array
205                         last_index = 0;
206                 }
207         }
208
209         else if(count % 2 != 0 && count < 8){ // "count" is odd
210                 j = 0;
211                 for(i = last_index; i < last_index + LENGTH_ALL_SAMPLES; i++){
212                         if(count <= 3){  // Red
213                                 RED_samples[i] = samples2[j];
214                         }
215                         else{  // IR
216                                 IR_samples[i] = samples2[j];
217                         }
218                         j++;
219                 } // end for
220
221                 // If 8 seconds has passed
222                 if(count == 7){
223
224                         // Activates the sending flag
225                         send_flag = true;
226
227                         // Reset samples acquisition
228                         last_index = 0;
229                         if(acquisition_active){
230                                 Out1_1_Write(LED_ON);
231                                 Out1_2_Write(LED_OFF);
232                         }
233
234                         // Calculate R
235                         R_aux = calculateOxygen();
236                         R = 0.8 * R_aux + 0.2 * last_measure;  // Feeds the value of previous
                                measures
237
238                         last_measure = R;   // Saves the result for the next acquisition
239                         count = 0;          // Restart the loop
240
241                         if((R > 100 || R < 95) && acquisition_active){
242                                 UART_1_PutString("Error in the measure\r\n");
243                         }
```

```
244                 }
245
246                 last_index += LENGTH_ALL_SAMPLES;
247                 count++;
248         }
249
250 }
251
252
253 int main(void){
254         CyGlobalIntEnable;  // Enable global interrupts
255
256         uint8 uart_array[2];
257
258         // Initialize the hardware components
259         TIA_Start();
260         ADC_DelSig_Start();
261         ADC_DelSig_StartConvert();
262         UART_1_Start();
263         Filter_1_Start();
264
265         Filter_1_SetCoherency(Filter_1_CHANNEL_A,Filter_1_KEY_MID);
266
267         DMA_1_Init();
268         DMA_2_Init();
269
270         Out1_1_Write(LED_OFF);  // RED
271         Out1_2_Write(LED_OFF);  // IR
272
273         // Initialize interrupts
274         isr_UART_StartEx(UARTInput);
275         isr_Filter_End_StartEx(FilterDone);
276
277         UART_1_PutString("UART Comunication: ON LINE\r\n");
278
279         // Main loop
280         for(;;){
281                 if(send_flag == true && acquisition_active){
282
283                         UART_1_PutString("R: ");
284
285                         send_flag = false;
286
287                         // Separate bytes from R
288                         uart_array[0] = R >> 8 ;
289                         uart_array[1] = (uint8) R;
290
291                         // Send result
292                         UART_1_PutArray(uart_array, (uint8) 2);
293
294                         UART_1_PutString("\r\n");
295                 }
296         }
297
298 }
```

Listing C.2: Código fuente C para implementación de la adquisición del SpO_2

Apéndice D

Código fuente de la aplicación Android

En este capítulo se puede consultar el código fuente completo respectivo a la aplicación desarrollada en Android y que se trato en el Capítulo 5: Diseño de aplicación compatible en Android. En Actividades se puede consultar el código fuente de las actividades Android usadas por la app, mientras que Librerías propias incluye el código Java de la librería desde donde se pueden llamar a los métodos creados para el tratamiento de las muestras recibidas.La gestión de la comunicación Bluetooth y los UUID se pueden localizar en respectivamente Servicios y Identificadores únicos universales (UUID).

D.1. Actividades

D.1.1. MainActivity.java

```
1  package com.example.fisoapp.presentation;
2
3  import androidx.appcompat.app.AppCompatActivity;
4  import android.bluetooth.BluetoothAdapter;
5  import android.bluetooth.BluetoothManager;
6  import android.content.Context;
7  import android.content.Intent;
8  import android.content.pm.PackageManager;
9  import android.os.Bundle;
10 import android.view.View;
11 import android.widget.Button;
12 import android.widget.Toast;
13 import com.example.fisoapp.R;
14
15
16 public class MainActivity extends AppCompatActivity {
17
18         private BluetoothAdapter mBluetoothAdapter;
19         private static final int REQUEST_ENABLE_BT = 1;
20
21         @Override
22         protected void onCreate(Bundle savedInstanceState) {
23                 super.onCreate(savedInstanceState);
24                 setContentView(R.layout.activity_main);
25
26                 // Determine whether BLE is supported on the device.
27                 // Then you can selectively disable BLE-related features.
28                 if (!getPackageManager().hasSystemFeature(PackageManager.FEATURE_BLUETOOTH_LE))
                       {
29                        Toast.makeText(this, R.string.ble_not_supported, Toast.LENGTH_SHORT).
                             show();
30                        finish();
31                 }
32
33                 // Initializes a Bluetooth adapter. For API level 18 and above, get a reference
                       to
34                 // BluetoothAdapter through BluetoothManager.
35                 final BluetoothManager bluetoothManager =
36                 (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
37                 mBluetoothAdapter = bluetoothManager.getAdapter();
38
```

```
39                  // Checks if Bluetooth is supported on the device.
40                  if (mBluetoothAdapter == null) {
41                          Toast.makeText(this, R.string.ble_not_supported, Toast.LENGTH_SHORT).
                                show();
42                          finish();
43                  }
44
45          }
46
47
48          public void conectar(View view){
49                  // Access to the connect to BLE devices menu
50
51                  Intent intent = new Intent(this, ScanActivity.class);
52                  startActivity(intent);
53          }
54
55          public void salir(View view){
56                  // Closes the app
57
58                  finish();
59          }
60
61          public void configuracion(View view){
62          }
63
64          public void data(View view){
65
66                  // Access to the data display activity after clicking the "measures" button
67
68                  Intent intent = new Intent(getApplicationContext(), ConnectedActivity.class);
69                  startActivity(intent);
70          }
71
72  } // End main activity
```

Listing D.1: MainActivity.java

D.1.2. ConnectedActivity.java

```
1   package com.example.fisoapp.presentation;
2
3   import android.annotation.SuppressLint;
4   import android.bluetooth.BluetoothGattCharacteristic;
5   import android.bluetooth.BluetoothGattService;
6   import android.content.BroadcastReceiver;
7   import android.content.ComponentName;
8   import android.content.Context;
9   import android.content.Intent;
10  import android.content.IntentFilter;
11  import android.content.ServiceConnection;
12  import android.content.SharedPreferences;
13  import android.graphics.Color;
14  import android.os.Bundle;
15  import android.os.IBinder;
16  import android.util.Log;
17  import android.view.Menu;
18  import android.view.MenuItem;
19  import android.view.View;
20  import android.widget.Button;
21  import android.widget.ImageView;
22  import android.widget.TextView;
23
24  import androidx.appcompat.app.AppCompatActivity;
25  import androidx.preference.PreferenceManager;
26
27  import com.example.fisoapp.R;
28  import com.example.fisoapp.data.Acquisition;
29  import com.example.fisoapp.domain.GattAttributes;
30  import com.example.fisoapp.services.BluetoothLeService;
31  import com.github.mikephil.charting.charts.LineChart;
32  import com.github.mikephil.charting.components.XAxis;
33  import com.github.mikephil.charting.components.YAxis;
34  import com.github.mikephil.charting.data.Entry;
35  import com.github.mikephil.charting.data.LineData;
36  import com.github.mikephil.charting.data.LineDataSet;
37
38  import java.io.IOException;
39  import java.util.ArrayList;
40  import java.util.Arrays;
41  import java.util.List;
42  import java.util.UUID;
```

```
43
44
45 public class ConnectedActivity extends AppCompatActivity {
46         private final static String TAG = ConnectedActivity.class.getSimpleName();
47
48         public static final String EXTRAS_DEVICE_NAME = "DEVICE_NAME";
49         public static final String EXTRAS_DEVICE_ADDRESS = "DEVICE_ADDRESS";
50
51         private TextView mConnectionState;
52         private String mDeviceName;
53         private String mDeviceAddress;
54         private BluetoothLeService mBluetoothLeService;
55         private ArrayList<ArrayList<BluetoothGattCharacteristic>> mGattCharacteristics =
56         new ArrayList<ArrayList<BluetoothGattCharacteristic>>();
57         private boolean mConnected = false;
58         private BluetoothGattCharacteristic mNotifyCharacteristic;
59         private BluetoothGattCharacteristic mCommandCharacteristic;
60
61         private com.github.mikephil.charting.charts.LineChart mLineChart;
62         private View mBottomConnect;
63         private View mBottomQuit;
64
65         private PreferenceManager mPrefManager;
66         private SharedPreferences mPreferences;
67
68         private final static ArrayList<String> prefs =
69         new ArrayList<>(Arrays.asList("front","back","coolant","oil","brakeL","brakeP",
70         "wasser"));
71
72         private Acquisition mAcquisition;
73
74         TextView rms_value_display; // RMS value display
75
76
77         @SuppressLint("RestrictedApi")
78         @Override
79         public void onCreate(Bundle savedInstanceState) {
80                 super.onCreate(savedInstanceState);
81                 setContentView(R.layout.activity_connected);
82
83                 // Creates LineChart use for plotting the data
84                 mLineChart = new LineChart(this);
85                 mLineChart = (LineChart) findViewById(R.id.linechart_1);
86                 mLineChart.setTouchEnabled(true);
87                 mLineChart.setPinchZoom(false);
88                 mLineChart.setScaleEnabled(true);
89
90                 mBottomConnect = findViewById(R.id.connectButton);
91                 mBottomQuit = findViewById(R.id.back_button);
92
93                 final Intent intent = getIntent();
94                 mDeviceName = intent.getStringExtra(EXTRAS_DEVICE_NAME);
95                 mDeviceAddress = intent.getStringExtra(EXTRAS_DEVICE_ADDRESS);
96
97                 // Sets up UI references.
98                 mConnectionState = (TextView) findViewById(R.id.connection_state);
99
100                mPrefManager= new PreferenceManager(this);
101                mPreferences = mPrefManager.getSharedPreferences();
102
103                //Set acquisition
104                mAcquisition = new Acquisition();
105
106                Intent gattServiceIntent = new Intent(this, BluetoothLeService.class);
107                bindService(gattServiceIntent, mServiceConnection, BIND_AUTO_CREATE);
108
109                // Defines the used methods
110                renderData();
111                quit();
112                startAcq();
113        }
114
115
116        public void connect(View view){
117                if( !mConnected) {
118                        mBluetoothLeService.connect(mDeviceAddress);
119                }
120        }
121
122
123        public void quit (){
124                // Stops Bluetooth connection and goes back to main menu
125
126                Button back_button = (Button) findViewById(R.id.back_button);
127                back_button.setOnClickListener(new View.OnClickListener() {
128                        @Override
129                        public void onClick(View view) {
```

```
130                             stopAcq(view);
131                             mBluetoothLeService.disconnect();
132                             finish();
133                     }
134             });
135     }
136
137
138     public void startAcq(){
139             // Starts the data acquisition and the BLE communication
140
141             Button acquisition_button = findViewById(R.id.StartButton);
142             acquisition_button.setOnClickListener(new View.OnClickListener() {
143                     @Override
144                     public void onClick(View view) {
145                             if(mCommandCharacteristic != null) {
146                                     mCommandCharacteristic.setValue("a");
147                                     mBluetoothLeService.writeCharacteristic(
                                            mCommandCharacteristic);
148
149                                     // Acquiring message to user
150                                     rms_value_display = findViewById(R.id.rms_value_display
                                            );
151                                     rms_value_display.setTextColor(Color.BLACK);
152                                     rms_value_display.setText(R.string.acquiring_message);
153                             }
154                     }
155             });
156     }
157
158
159     public void stopAcq(View view){
160             // Stops the acquisition of data and the BLE communication
161
162             if (mCommandCharacteristic != null){
163                     mCommandCharacteristic.setValue("q");
164                     mBluetoothLeService.writeCharacteristic(mCommandCharacteristic);
165             }
166     }
167
168
169     private void saveData() {
170             // Save the plotted data in a CSV file
171
172             short[] short_data = mAcquisition.getConvertedData();
173             String file_name = "DataPointsStorage";
174
175             try {
176                     mAcquisition.saveAcquisition(short_data, file_name);
177             } catch (IOException e) {
178                     throw new RuntimeException(e);
179             }
180     }
181
182
183     @Override
184     public boolean onCreateOptionsMenu(Menu menu) {
185             getMenuInflater().inflate(R.menu.connected_menu, menu);
186             if (!mConnected) {
187                     menu.findItem(R.id.menu_disconnect).setVisible(false);
188                     menu.findItem(R.id.menu_connect).setVisible(true);
189
190             } else {
191                     menu.findItem(R.id.menu_disconnect).setVisible(true);
192                     menu.findItem(R.id.menu_connect).setVisible(false);
193             }
194             return true;
195     }
196
197
198     @Override
199     public boolean onOptionsItemSelected(MenuItem item) {
200             switch (item.getItemId()) {
201                     case R.id.menu_connect:
202                     mBluetoothLeService.connect(mDeviceAddress);
203                     break;
204                     case R.id.menu_disconnect:
205                     mBluetoothLeService.disconnect();
206                     break;
207             }
208             return true;
209     }
210
211
212     @Override
213     protected void onResume() {
214             super.onResume();
```

```
215                 registerReceiver(mGattUpdateReceiver, makeGattUpdateIntentFilter());
216                 if (mBluetoothLeService != null) {
217                         final boolean result = mBluetoothLeService.connect(mDeviceAddress);
218                         Log.d(TAG, "Connect request result=" + result);
219                 }
220         }
221
222
223         @Override
224         protected void onPause() {
225                 super.onPause();
226                 unregisterReceiver(mGattUpdateReceiver);
227         }
228
229
230         @Override
231         protected void onDestroy() {
232                 super.onDestroy();
233                 unbindService(mServiceConnection);
234                 mBluetoothLeService = null;
235         }
236
237
238         private void updateConnectionState(final int resourceId) {
239                 runOnUiThread(new Runnable() {
240                         @Override
241                         public void run() {
242                                 mConnectionState.setText(resourceId);
243                         }
244                 });
245         }
246
247
248         private void discoverGattServices(List<BluetoothGattService> gattServices) {
249                 // Inicializar las notificaciones y extraer characteristica de commandos
250
251                 if (gattServices == null) return;
252                 //Inicio notificacion status
253                 BluetoothGattCharacteristic TxChar = gattServices.get(2).getCharacteristic(
254                 UUID.fromString(GattAttributes.TX_CHAR)
255                 );
256
257                 int charaProp = TxChar.getProperties();
258                 if ((charaProp | BluetoothGattCharacteristic.PROPERTY_NOTIFY) > 0) {
259                         mBluetoothLeService.setCharacteristicNotification(TxChar, true);
260                 }
261
262                 mCommandCharacteristic = gattServices.get(2).getCharacteristic(
263                 UUID.fromString(GattAttributes.RX_CHAR)
264                 );
265         }
266
267
268         // Code to manage Service lifecycle.
269         private final ServiceConnection mServiceConnection = new ServiceConnection() {
270
271                 @Override
272                 public void onServiceConnected(ComponentName componentName, IBinder service) {
273                         mBluetoothLeService = ((BluetoothLeService.LocalBinder) service).
                                getService();
274                         if (!mBluetoothLeService.initialize()) {
275                                 Log.e(TAG, "Unable to initialize Bluetooth");
276                                 finish();
277                         }
278                         // Automatically connects to the device upon successful start-up
                                initialization.
279                         mBluetoothLeService.connect(mDeviceAddress);
280                 }
281
282
283                 @Override
284                 public void onServiceDisconnected(ComponentName componentName) {
285                         mBluetoothLeService = null;
286                 }
287         };
288
289
290         // Handles various events fired by the Service.
291         // ACTION_GATT_CONNECTED: connected to a GATT server.
292         // ACTION_GATT_DISCONNECTED: disconnected from a GATT server.
293         // ACTION_GATT_SERVICES_DISCOVERED: discovered GATT services.
294         // ACTION_DATA_AVAILABLE: received data from the device. This can be a result of read
                or
295         // notification operations.
296         private final BroadcastReceiver mGattUpdateReceiver = new BroadcastReceiver() {
297                 @Override
298                 public void onReceive(Context context, Intent intent) {
```

```
299                         final String action = intent.getAction();
300                         if (BluetoothLeService.ACTION_GATT_CONNECTED.equals(action)) {
301                                 mConnected = true;
302                                 updateConnectionState(R.string.connected);
303                                 ImageView view = findViewById(R.id.conn_image);
304                                 view.setVisibility(View.VISIBLE);
305                                 invalidateOptionsMenu();
306                         } else if (BluetoothLeService.ACTION_GATT_DISCONNECTED.equals(action))
                                {
307                                 mConnected = false;
308                                 updateConnectionState(R.string.disconnected);
309                                 ImageView view = findViewById(R.id.conn_image);
310                                 view.setVisibility(View.INVISIBLE);
311                                 invalidateOptionsMenu();
312                         } else if (BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED.equals(
                                action)) {
313                                 // Show all the supported services and characteristics on the
                                        user interface.
314                                 discoverGattServices(mBluetoothLeService.
                                        getSupportedGattServices());
315
316                         } else if (BluetoothLeService.ACTION_DATA_AVAILABLE.equals(action)) {
317                                 // Go here if there is info ready to be stored
318
319                                 byte[] data = intent.getByteArrayExtra(BluetoothLeService.
                                        EXTRA_DATA);
320                                 mAcquisition.addPacket(data);
321
322                                 // Plot and save the data if the buffer is full
323                                 if (mAcquisition.isReadyToPlot()){
324                                         mAcquisition.updateReadyToPlot(false);
325                                         mAcquisition.adaptSamples(mAcquisition.getData());
326                                         updateGraphics();
327                                         saveData();
328                                 }
329                         }
330                 }
331         };
332
333
334         private static IntentFilter makeGattUpdateIntentFilter() {
335                 final IntentFilter intentFilter = new IntentFilter();
336                 intentFilter.addAction(BluetoothLeService.ACTION_GATT_CONNECTED);
337                 intentFilter.addAction(BluetoothLeService.ACTION_GATT_DISCONNECTED);
338                 intentFilter.addAction(BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED);
339                 intentFilter.addAction(BluetoothLeService.ACTION_DATA_AVAILABLE);
340                 return intentFilter;
341         }
342
343
344         private void updateGraphics(){
345                 // Displays the RMS value in mV and plot the signal
346
347                 double RMS = mAcquisition.calculateRMS(mAcquisition.getConvertedData());
348
349                 // Adapt RMS to mV
350                 double ADC_resolution = 16;     // ADC resolution bits
351                 double ADC_top_limit = 3.3;     // Max ADC voltage level
352                 double ADC_bottom_limit = 0;    // Minimum ADC voltage level
353
354                 double mRMS = (((RMS * ADC_top_limit) / (Math.pow(2, ADC_resolution) - 1) +
355                 ADC_bottom_limit) * 1000);
356
357                 // Displays mRMS value in screen
358                 String mRMS_display = "RMS: " + mRMS + " mV";
359                 rms_value_display = findViewById(R.id.rms_value_display);
360                 rms_value_display.setText(mRMS_display);
361
362                 if (mRMS>2){
363                         rms_value_display.setTextColor(Color.RED);
364                 }
365                 else {
366                         rms_value_display.setTextColor(Color.GREEN);
367                 }
368
369                 // Update mlineChart with the data from mAcquisition
370                 renderData();
371         }
372
373
374         public void renderData(){
375                 // Configures the axis for the LineChart before plotting and then calls the
                        function to plot
376
377                 // Horizontal axis configuration
378                 XAxis xAxis = mLineChart.getXAxis();
379                 xAxis.enableGridDashedLine(10f, 10f, 0f);
```

```
380                 xAxis.setAxisMaximum(60f); // Represent 60 seconds of the signal
381                 xAxis.setAxisMinimum(0f);
382
383                 // Vertical axis configuration
384                 YAxis leftAxis = mLineChart.getAxisLeft();
385                 leftAxis.removeAllLimitLines();
386                 leftAxis.setDrawZeroLine(false);
387                 leftAxis.setAxisMinimum(0f);
388
389                 mLineChart.getAxisRight().setEnabled(false);
390                 setData();
391         }
392
393
394         private void setData() {
395                 // Plot the received 2 bytes data
396
397                 short[] short_data = mAcquisition.getConvertedData();
398                 int data_array_length = short_data.length;
399
400                 // Conversion: Byte[] -> int[]
401                 int[] int_data = new int[data_array_length];
402                 for (int i = 0; i < data_array_length; i++){
403                         int_data[i] = short_data[i] & 0xFFFF;
404                 }
405
406                 // Conversion: int[] -> float[]
407                 float[] float_data = new float[data_array_length];
408                 for (int i = 0; i < data_array_length; i++){
409                         float_data[i] = (float) int_data[i];
410                 }
411
412                 // Conversion from samples to milivolts
413                 double ADC_resolution = 16;     // ADC resolution bits
414                 double ADC_top_limit = 3.3;     // Max ADC voltage level
415                 double ADC_bottom_limit = 0;    // Minimum ADC voltage level
416
417                 float[] mV_data = new float[data_array_length];
418                 for (int i = 0; i < data_array_length; i++){
419                         mV_data[i] = (float) (((float_data[i] * ADC_top_limit) /
420                         (Math.pow(2, ADC_resolution) - 1) + ADC_bottom_limit) * 1000);
421                 }
422
423                 // Add values to entry list
424                 ArrayList<Entry> data_points = new ArrayList<>();
425                 float j = 0;
426                 for (int i = 0; i < data_array_length; i++){
427                         j += 0.001;  // Seconds per sample
428                         data_points.add(new Entry(j, mV_data[i]));
429                 }
430
431                 LineDataSet plot1;
432                 plot1 = new LineDataSet(data_points, "ECG");
433                 LineData data = new LineData(plot1);
434                 plot1.setDrawCircles(false);
435                 plot1.setLineWidth(0f);
436                 plot1.setColor(Color.argb(255,25,190,190));
437
438                 mLineChart.setData(data);
439         }
440
441 } //End class
```

Listing D.2: ConnectedActivity.java

D.1.3. ScanActivity.java

```
1  package com.example.fisoapp.presentation;
2
3  import android.Manifest;
4  import android.annotation.SuppressLint;
5  import android.app.Activity;
6  import android.bluetooth.BluetoothAdapter;
7  import android.bluetooth.BluetoothDevice;
8  import android.bluetooth.BluetoothManager;
9  import android.bluetooth.le.BluetoothLeScanner;
10 import android.bluetooth.le.ScanCallback;
11 import android.bluetooth.le.ScanFilter;
12 import android.bluetooth.le.ScanResult;
13 import android.bluetooth.le.ScanSettings;
14 import android.content.Context;
15 import android.content.DialogInterface;
```

```
16 import android.content.Intent;
17 import android.content.pm.PackageManager;
18 import android.location.LocationManager;
19 import android.os.Build;
20 import android.os.Bundle;
21 import android.os.Handler;
22 import android.os.ParcelUuid;
23 import android.provider.Settings;
24 import android.view.LayoutInflater;
25 import android.view.Menu;
26 import android.view.MenuItem;
27 import android.view.View;
28 import android.view.ViewGroup;
29 import android.widget.AdapterView;
30 import android.widget.BaseAdapter;
31 import android.widget.ListView;
32 import android.widget.TextView;
33 import android.widget.Toast;
34
35 import androidx.appcompat.app.AlertDialog;
36 import androidx.appcompat.app.AppCompatActivity;
37 import androidx.core.app.ActivityCompat;
38
39
40 import com.example.fisoapp.R;
41 import com.example.fisoapp.domain.GattAttributes;
42
43 import java.util.ArrayList;
44
45 public class ScanActivity extends AppCompatActivity {
46
47
48         private LocationManager mLocationManager;
49         private BluetoothAdapter mBluetoothAdapter;
50         private BluetoothLeScanner mBluetoothLeScanner;
51         private static final int REQUEST_ENABLE_BT = 1;
52         private static final int REQUEST_SCANN_BT = 2;
53         private LeDeviceListAdapter mLeDeviceListAdapter;
54         private ListView mScannerList;
55         private boolean mScanning;
56         private Handler mHandler;
57
58         //   private PreferenceManager mPrefManager;
59         //   private SharedPreferences mPreferences;
60
61
62         private ScanFilter mScanFilter;
63         private ArrayList<ScanFilter> filters = new ArrayList<ScanFilter>();
64         private ScanSettings mScanSettings;
65         private ParcelUuid CheckControlUUID;
66
67         private static final int PERMISSION_REQUEST_LOCATION = 1;
68         private static final int PERMISSION_REQUEST_SCANN = 2;
69         private static final int PERMISSION_REQUEST_CONNECT = 3;
70         private static final int PERMISSION_REQUEST_BLUETOOTH = 4;
71
72         // Stops scanning after 10 seconds.
73         private static final long SCAN_PERIOD = 10000;
74
75         @SuppressLint("RestrictedApi")
76         @Override
77         public void onCreate(Bundle savedInstanceState) {
78                 super.onCreate(savedInstanceState);
79                 setContentView(R.layout.activity_scan);
80                 mHandler = new Handler();
81
82                 final BluetoothManager bluetoothManager =
83                 (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
84                 mBluetoothAdapter = bluetoothManager.getAdapter();
85                 mBluetoothLeScanner = mBluetoothAdapter.getBluetoothLeScanner();
86                 mLocationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE)
                        ;
87                 mScannerList = findViewById(R.id.scanned_list);
88
89
90                 CheckControlUUID = ParcelUuid.fromString(GattAttributes.CHECKCONTROL_SERVICE);
91                 mScanFilter = new ScanFilter.Builder()
92                 .setServiceUuid(CheckControlUUID)
93                 //.setDeviceName("Kadett GSi 8v")
94                 //setDeviceAddress("00:A0:50:D1:36:47").
95                 .build();
96                 filters.add(mScanFilter);
97
98                 mScanSettings = new ScanSettings.Builder()
99                 .setScanMode(ScanSettings.SCAN_MODE_BALANCED)
100                .build();
101
```

```
102         }
103
104         @Override
105         public boolean onCreateOptionsMenu(Menu menu) {
106                 getMenuInflater().inflate(R.menu.scan_menu, menu);
107                 if (!mScanning) {
108                         menu.findItem(R.id.menu_stop).setVisible(false);
109                         menu.findItem(R.id.menu_scan).setVisible(true);
110
111                 } else {
112                         menu.findItem(R.id.menu_stop).setVisible(true);
113                         menu.findItem(R.id.menu_scan).setVisible(false);
114                 }
115                 return true;
116         }
117
118         @Override
119         public boolean onOptionsItemSelected(MenuItem item) {
120                 switch (item.getItemId()) {
121                         case R.id.menu_scan:
122                         mLeDeviceListAdapter.clear();
123                         scanLeDevice(true);
124                         break;
125                         case R.id.menu_stop:
126                         scanLeDevice(false);
127                         break;
128                 }
129                 return true;
130         }
131
132
133         @Override
134         protected void onResume() {
135                 super.onResume();
136
137                 // Ensures Bluetooth is enabled on the device. If Bluetooth is not currently
                        enabled,
138                 // fire an intent to display a dialog asking the user to grant permission to
                        enable it.
139                 if (!mBluetoothAdapter.isEnabled()) {
140                         Intent enableBtIntent = new Intent(BluetoothAdapter.
                                ACTION_REQUEST_ENABLE);
141                         startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
142
143                 }
144
145                 boolean gps_enabled = false;
146                 boolean network_enabled = false;
147
148                 try {
149                         gps_enabled = mLocationManager.isProviderEnabled(LocationManager.
                                GPS_PROVIDER);
150                 } catch (Exception ex) {
151                 }
152
153                 try {
154                         network_enabled = mLocationManager.isProviderEnabled(LocationManager.
                                NETWORK_PROVIDER);
155                 } catch (Exception ex) {
156                 }
157
158                 if (!gps_enabled && !network_enabled) {
159                         // notify user
160                         new AlertDialog.Builder(this)
161                         .setMessage(R.string.gps_network_not_enabled)
162                         .setPositiveButton(R.string.open_location_settings, new DialogInterface
                                .OnClickListener() {
163                                     @Override
164                                     public void onClick(DialogInterface paramDialogInterface, int
                                            paramInt) {
165                                         startActivity(new Intent(Settings.
                                                ACTION_LOCATION_SOURCE_SETTINGS));
166                                     }
167                         })
168                         .setNegativeButton(R.string.cancel, null)
169                         .show();
170                 }
171
172                 // Initializes list view adapter.
173                 mLeDeviceListAdapter = new LeDeviceListAdapter();
174                 mScannerList.setAdapter(mLeDeviceListAdapter);
175                 mScannerList.setOnItemClickListener(ListClickListner);
176
177                 //Check permissions needed along the activity
178
179                 if( Build.VERSION.SDK_INT >=Build.VERSION_CODES.S ){//Android 12 or higher //
                        ToDo: Check correct working of this section
```

```
180
181
182                         if (ActivityCompat.checkSelfPermission(this, Manifest.permission.
                                BLUETOOTH_SCAN)
183                         != PackageManager.PERMISSION_GRANTED) {
184
185                                 requestPermission(Manifest.permission.BLUETOOTH_SCAN, "Necesito
                                        Bluetooth", PERMISSION_REQUEST_SCANN, this);
186
187                         }
188                         else if (ActivityCompat.checkSelfPermission(this, Manifest.permission.
                                BLUETOOTH_CONNECT)
189                         != PackageManager.PERMISSION_GRANTED) {
190
191                                 requestPermission(Manifest.permission.BLUETOOTH_CONNECT, "
                                        necesito connect", PERMISSION_REQUEST_CONNECT,
192                                 this);
193                 }}
194
195                 else if(ActivityCompat.checkSelfPermission(this, Manifest.permission.BLUETOOTH)
                            // Android 11 or lower
196                 != PackageManager.PERMISSION_GRANTED) {
197
198                         requestPermission(Manifest.permission.BLUETOOTH, "necesito connect",
                                PERMISSION_REQUEST_BLUETOOTH,
199                         this);
200                 }
201
202                 if (ActivityCompat.checkSelfPermission(this, Manifest.permission.
                        ACCESS_FINE_LOCATION)
203                 != PackageManager.PERMISSION_GRANTED) {
204
205                         requestPermission(Manifest.permission.ACCESS_FINE_LOCATION,
206                         "Sin el permiso localizacion no es posible escanear dispositivos" + "
                                Bluetooth.", PERMISSION_REQUEST_LOCATION, this);
207
208                 }
209                 else {
210                         scanLeDevice(true); // ToDo si el Bluetooth no estaba activado al
                                entrar a la actividad, el programa crashea
211                 }
212
213         }
214
215         @Override
216         protected void onActivityResult(int requestCode, int resultCode, Intent data) {
217                 // User chose not to enable Bluetooth.
218                 if (requestCode == REQUEST_ENABLE_BT && resultCode == Activity.RESULT_CANCELED)
                            {
219                         finish();
220                         return;
221                 }
222                 super.onActivityResult(requestCode, resultCode, data);
223         }
224
225
226         private final ListView.OnItemClickListener ListClickListner =
227         new AdapterView.OnItemClickListener() {
228                 @SuppressLint("MissingPermission")
229                 @Override
230                 public void onItemClick(AdapterView<?> parent, View view, int position, long id
                            ) {
231                         final BluetoothDevice device = mLeDeviceListAdapter.getDevice(position)
                                ;
232                         if (device == null) return;
233                         final Intent intent = new Intent(ScanActivity.this, ConnectedActivity.
                                class);
234                         intent.putExtra(ConnectedActivity.EXTRAS_DEVICE_NAME, device.getName())
                                ;
235                         intent.putExtra(ConnectedActivity.EXTRAS_DEVICE_ADDRESS, device.
                                getAddress());
236                         if (mScanning) {
237                                 mBluetoothLeScanner.stopScan(mLeScanCallback);
238                                 mScanning = false;
239                         }
240                         startActivity(intent);
241
242
243                 }
244         };
245
246
247         @SuppressLint("MissingPermission")
248         protected void onListItemClick(ListView l, View v, int position, long id) {
249                 final BluetoothDevice device = mLeDeviceListAdapter.getDevice(position);
250                 if (device == null) return;
251                 //        final Intent intent = new Intent(this, ConnectedActivity.class);
```

```
252                 //        intent.putExtra(ConnectedActivity.EXTRAS_DEVICE_NAME, device.getName
                        ());
253                 //        intent.putExtra(ConnectedActivity.EXTRAS_DEVICE_ADDRESS, device.
                        getAddress());
254                 if (mScanning) {
255                         mBluetoothLeScanner.stopScan(mLeScanCallback);
256                         mScanning = false;
257                 }
258                 //        startActivity(intent);
259         }
260
261
262         @Override
263         protected void onPause() {
264                 super.onPause();
265                 if(mScanning == true){
266                         scanLeDevice(false);
267                 }
268                 mLeDeviceListAdapter.clear();
269         }
270
271         @SuppressLint("MissingPermission")
272         private void scanLeDevice(final boolean enable) {
273                 if (enable) {
274                         // Stops scanning after a pre-defined scan period.
275                         mHandler.postDelayed(new Runnable() {
276                                 @SuppressLint("MissingPermission")
277                                 @Override
278                                 public void run() {
279                                         mScanning = false;
280
281                                         mBluetoothLeScanner.stopScan(mLeScanCallback);
282                                         invalidateOptionsMenu();
283                                 }
284                         }, SCAN_PERIOD);
285
286                         mScanning = true;
287                         mBluetoothLeScanner.startScan(null, mScanSettings, mLeScanCallback);
288                 } else {
289                         mScanning = false;
290                         mBluetoothLeScanner.stopScan(mLeScanCallback);
291                 }
292                 invalidateOptionsMenu();
293         }
294
295         private static void requestPermission(final String permiso, String
296         justificacion, final int requestCode, final Activity actividad) {
297                 if (ActivityCompat.shouldShowRequestPermissionRationale(actividad,
298                 permiso)) {
299                         new AlertDialog.Builder(actividad)
300                         .setTitle("Solicitud de permiso")
301                         .setMessage(justificacion)
302                         .setPositiveButton("Ok", new DialogInterface.OnClickListener() {
303                                 public void onClick(DialogInterface dialog, int whichButton) {
304                                         ActivityCompat.requestPermissions(actividad,
305                                         new String[]{permiso}, requestCode);
306                                 }
307                         }).show();
308                 } else {
309                         ActivityCompat.requestPermissions(actividad,
310                         new String[]{permiso}, requestCode);
311                 }
312         }
313
314         @Override
315         public void onRequestPermissionsResult(int requestCode,
316         String[] permissions, int[] grantResults) {
317                 super.onRequestPermissionsResult(requestCode, permissions, grantResults);
318
319                 switch (requestCode){
320                         case PERMISSION_REQUEST_SCANN:
321                         if (grantResults.length == 1 && grantResults[0] == PackageManager.
                                PERMISSION_DENIED) {
322                                 Toast.makeText(this,"No se puede escanear sin el permiso" +
323                                 " de escaneo",Toast.LENGTH_SHORT).show();
324                                 finish();
325                         }
326                         break;
327                         case PERMISSION_REQUEST_LOCATION:
328                         if (grantResults.length == 1 && grantResults[0] == PackageManager.
                                PERMISSION_DENIED) {
329                                 Toast.makeText(this,"No se puede escanear sin el permiso" + "de
                                        localizacion",Toast.LENGTH_SHORT).show();
330
331                                 finish();
332                         }
333                         break;
```

```
334                         case PERMISSION_REQUEST_CONNECT:
335                         if (grantResults.length == 1 && grantResults[0] == PackageManager.
                                PERMISSION_DENIED) {
336                             Toast.makeText(this,"No se puede escanear sin el permiso " +
337                             " de conexion",Toast.LENGTH_SHORT).show();
338                             finish();
339                         }
340                         break;
341
342                         case PERMISSION_REQUEST_BLUETOOTH:
343                         if (grantResults.length == 1 && grantResults[0] == PackageManager.
                                PERMISSION_DENIED) {
344                             Toast.makeText(this,"No se puede escanear sin acceso a" +
345                             " Bluetooth",Toast.LENGTH_SHORT).show();
346                             finish();
347                         }
348                         break;
349                         default:
350                         break;
351
352
353                 }
354         }
355
356
357         // Adapter for holding devices found through scanning.
358         private class LeDeviceListAdapter extends BaseAdapter {
359                 private ArrayList<BluetoothDevice> mLeDevices;
360                 private LayoutInflater mInflator;
361
362                 public LeDeviceListAdapter() {
363                         super();
364                         mLeDevices = new ArrayList<BluetoothDevice>();
365                         mInflator = ScanActivity.this.getLayoutInflater();
366                 }
367
368                 public void addDevice(BluetoothDevice device) {
369                         if (!mLeDevices.contains(device)) {
370                                 mLeDevices.add(device);
371                         }
372                 }
373
374                 public BluetoothDevice getDevice(int position) {
375                         return mLeDevices.get(position);
376                 }
377
378                 public void clear() {
379                         mLeDevices.clear();
380                 }
381
382                 @Override
383                 public int getCount() {
384                         return mLeDevices.size();
385                 }
386
387                 @Override
388                 public Object getItem(int i) {
389                         return mLeDevices.get(i);
390                 }
391
392                 @Override
393                 public long getItemId(int i) {
394                         return i;
395                 }
396
397                 @Override
398                 public View getView(int i, View view, ViewGroup viewGroup) {
399                         ViewHolder viewHolder;
400                         // General ListView optimization code.
401                         if (view == null) {
402                                 view = mInflator.inflate(R.layout.list_item, null);
403                                 viewHolder = new ViewHolder();
404                                 viewHolder.deviceAddress = (TextView) view.findViewById(R.id.
                                        device_address);
405                                 viewHolder.deviceName = (TextView) view.findViewById(R.id.
                                        device_name);
406                                 view.setTag(viewHolder);
407                         } else {
408                                 viewHolder = (ViewHolder) view.getTag();
409                         }
410
411                         BluetoothDevice device = mLeDevices.get(i);
412                         @SuppressLint("MissingPermission")
413                         final String deviceName = device.getName();
414                         if (deviceName != null && deviceName.length() > 0)
415                         viewHolder.deviceName.setText(deviceName);
416                         else
```

```
417                         viewHolder.deviceName.setText(R.string.unknown_device);
418                         viewHolder.deviceAddress.setText(device.getAddress());
419
420                         return view;
421                 }
422
423         }
424         // Device scan callback.
425         private ScanCallback mLeScanCallback =
426         new ScanCallback() {
427
428                 @Override
429                 public void onScanResult(int callbackType, ScanResult result) {
430                         super.onScanResult(callbackType,result);
431                         mLeDeviceListAdapter.addDevice(result.getDevice());
432                         mLeDeviceListAdapter.notifyDataSetChanged();
433                 }
434         };
435
436
437         static class ViewHolder {
438                 TextView deviceName;
439                 TextView deviceAddress;
440         }
441 }
```

Listing D.3: ScanActivity.java

D.2. Librerías propias

```
1  package com.example.fisoapp.data;
2
3  import android.os.Environment;
4  import com.opencsv.CSVWriter;
5  import java.io.File;
6  import java.io.FileWriter;
7  import java.io.IOException;
8  import java.time.format.DateTimeFormatter;
9  import java.time.LocalDateTime;
10
11 public class Acquisition {
12
13         private byte[] mData;
14         private short[] convertedData;
15         private int lastIndex;
16         private boolean ready_to_plot;
17         private final int length_mData = 120000; // Length of the 1 byte buffer send from BLE
18
19         public Acquisition(){
20                 // Initial conditions
21
22                 mData = new byte[length_mData];             // Data buffer 1 byte
23                 convertedData = new short[mData.length/2];  // Data buffer reconverted to 2
                       bytes
24                 lastIndex = 0;
25                 ready_to_plot = false;
26         }
27
28
29         public byte[] getData(){
30                 return mData;
31         }
32
33
34         public void setData(byte[] Data){
35                 mData = Data;
36         }
37
38
39         public int getLastIndex(){
40                 return lastIndex;
41         }
42
43
44         public void setLastIndex(int LastIndex){
45                 lastIndex = LastIndex;
46         }
47
48
49         public short[] getConvertedData(){
50                 return convertedData;
```

```
51          }
52
53
54          public void setConvertedData(short[] data16){
55                  convertedData = data16;
56          }
57
58
59          public boolean isReadyToPlot(){
60                  return ready_to_plot;
61          }
62
63
64          public void updateReadyToPlot(boolean ready_state){
65                  ready_to_plot = ready_state;
66          }
67
68
69          public void saveAcquisition(short[] data, String fileName) throws IOException {
70                  // Saves the data received in a .csv file
71
72                  // CSV file name
73                  DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy_MM_dd_HH_mm_ss");
74                  LocalDateTime now = LocalDateTime.now();
75                  String file_path = Environment.getExternalStoragePublicDirectory(Environment.
                        DIRECTORY_DOWNLOADS)
76                  + "/" + fileName + dtf.format(now) + ".csv";
77                  File file = new File(file_path);
78
79                  CSVWriter writer;
80
81                  // If the file already exists -> Appends data
82                  if (file.exists() && !file.isDirectory()){
83                          FileWriter mFileWriter = new FileWriter(file_path, true);
84                          writer = new CSVWriter(mFileWriter, CSVWriter.DEFAULT_SEPARATOR,
85                          CSVWriter.NO_QUOTE_CHARACTER);
86                  }
87                  // If the file does not exists -> Creates it and adds data
88                  else {
89                          writer = new CSVWriter(new FileWriter(file_path), CSVWriter.
                                DEFAULT_SEPARATOR,
90                          CSVWriter.NO_QUOTE_CHARACTER);
91                  }
92
93                  // Conversion: short[] -> String[]
94                  String[] data_string = new String[data.length];
95                  for (int i = 0; i < data.length; i++){
96                          data_string[i] = "" + (data[i] & 0xFFFF);
97                  }
98
99                  // Save data
100                 writer.writeNext(data_string);
101                 writer.close();
102         }
103
104
105         public double calculateRMS(short[] data){
106                 // Calculate the RMS from a data array
107
108                 double rms = 0;
109
110                 for (int i = 0; i < data.length; i++) {
111                         rms += data[i] * data[i];
112                 }
113                 rms = Math.sqrt(rms / data.length);
114                 return rms;
115         }
116
117
118         public void addPacket(byte[] Packet){
119                 // Stores the 1 byte data received in a single array
120
121                 int lastIndex = getLastIndex();
122                 byte[] data = getData();
123
124                 // Copy array "Packet" into array "data"
125                 System.arraycopy(Packet,0, data, lastIndex, Packet.length);
126                 lastIndex += Packet.length;
127                 if (lastIndex >= length_mData){
128                         lastIndex = 0;
129                         updateReadyToPlot(true); // The buffer is full
130                 }
131                 setData(data);
132                 setLastIndex(lastIndex);
133         }
134
135
```

```
136         public void adaptSamples(byte[] Samples){
137                 // Adapts samples from 8 bits to 16 bits
138
139                 short[] data16 = new short[Samples.length/2];
140
141                 for(int i = 0; i<(Samples.length/2); i++){
142                         data16[i] = (short) ((Samples[2*i] << 8) | Samples[2*i+1] & 0xFF);
143                 }
144
145                 setConvertedData(data16);
146         }
147 }
```

Listing D.4: Acquisition.java

D.3. Servicios

D.3.1. BluetoothLeService

```
1  package com.example.fisoapp.services;
2  import android.Manifest;
3  import android.app.Service;
4  import android.bluetooth.BluetoothAdapter;
5  import android.bluetooth.BluetoothDevice;
6  import android.bluetooth.BluetoothGatt;
7  import android.bluetooth.BluetoothGattCallback;
8  import android.bluetooth.BluetoothGattCharacteristic;
9  import android.bluetooth.BluetoothGattDescriptor;
10 import android.bluetooth.BluetoothGattService;
11 import android.bluetooth.BluetoothManager;
12 import android.bluetooth.BluetoothProfile;
13 import android.content.Context;
14 import android.content.Intent;
15 import android.content.pm.PackageManager;
16 import android.os.Binder;
17 import android.os.IBinder;
18 import android.util.Log;
19 import androidx.core.app.ActivityCompat;
20 import java.util.List;
21 import java.util.UUID;
22
23 /* Service for managing connection and data communication with a GATT server hosted on a given
       Bluetooth LE device. */
24 public class BluetoothLeService extends Service {
25         private final static String TAG = BluetoothLeService.class.getSimpleName();
26         protected static final UUID CHARACTERISTIC_UPDATE_NOTIFICATION_DESCRIPTOR_UUID = UUID.
               fromString("00002902-0000-1000-8000-00805f9b34fb");
27
28         private BluetoothManager mBluetoothManager;
29         private BluetoothAdapter mBluetoothAdapter;
30         private String mBluetoothDeviceAddress;
31         private BluetoothGatt mBluetoothGatt;
32         private int mConnectionState = STATE_DISCONNECTED;
33
34         private static final int STATE_DISCONNECTED = 0;
35         private static final int STATE_CONNECTING = 1;
36         private static final int STATE_CONNECTED = 2;
37
38         public final static String ACTION_GATT_CONNECTED =
39         "com.example.bluetooth.le.ACTION_GATT_CONNECTED";
40         public final static String ACTION_GATT_DISCONNECTED =
41         "com.example.bluetooth.le.ACTION_GATT_DISCONNECTED";
42         public final static String ACTION_GATT_SERVICES_DISCOVERED =
43         "com.example.bluetooth.le.ACTION_GATT_SERVICES_DISCOVERED";
44         public final static String ACTION_DATA_AVAILABLE =
45         "com.example.bluetooth.le.ACTION_DATA_AVAILABLE";
46         public final static String EXTRA_DATA =
47         "com.example.bluetooth.le.EXTRA_DATA";
48         public final static String EXTRA_CHARACTERISTIC =
49         "com.example.bluetooth.le.Characteristic";
50
51         // Implements callback methods for GATT events that the app cares about.  For example,
52         // connection change and services discovered.
53         private final BluetoothGattCallback mGattCallback = new BluetoothGattCallback() {
54                 @Override
55                 public void onConnectionStateChange(BluetoothGatt gatt, int status, int
                       newState) {
56                     String intentAction;
57                     if (newState == BluetoothProfile.STATE_CONNECTED) {
58                             intentAction = ACTION_GATT_CONNECTED;
59                             mConnectionState = STATE_CONNECTED;
```

```
60                                  broadcastUpdate(intentAction);
61                                  Log.i(TAG, "Connected to GATT server.");
62                                  // Attempts to discover services after successful connection.
63                                  Log.i(TAG, "Attempting to start service discovery:" +
64                                  mBluetoothGatt.discoverServices());
65
66                          } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
67                                  intentAction = ACTION_GATT_DISCONNECTED;
68                                  mConnectionState = STATE_DISCONNECTED;
69                                  Log.i(TAG, "Disconnected from GATT server.");
70                                  broadcastUpdate(intentAction);
71                          }
72                  }
73
74                  @Override
75                  public void onServicesDiscovered(BluetoothGatt gatt, int status) {
76                          if (status == BluetoothGatt.GATT_SUCCESS) {
77                                  broadcastUpdate(ACTION_GATT_SERVICES_DISCOVERED);
78                          } else {
79                                  Log.w(TAG, "onServicesDiscovered received: " + status);
80                          }
81                  }
82
83                  @Override
84                  public void onCharacteristicRead(BluetoothGatt gatt,
85                  BluetoothGattCharacteristic characteristic,
86                  int status) {
87                          if (status == BluetoothGatt.GATT_SUCCESS) {
88                                  broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic);
89                          }
90                  }
91
92                  @Override
93                  public void onCharacteristicChanged(BluetoothGatt gatt,
94                  BluetoothGattCharacteristic characteristic) {
95                          broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic);
96                  }
97          };
98
99          private void broadcastUpdate(final String action) {
100                 final Intent intent = new Intent(action);
101                 sendBroadcast(intent);
102         }
103
104         private void broadcastUpdate(final String action,
105         final BluetoothGattCharacteristic characteristic) {
106                 final Intent intent = new Intent(action);
107
108                 // This is special handling for the Heart Rate Measurement profile.  Data
                        parsing is
109                 // carried out as per profile specifications:
110                 // http://developer.bluetooth.org/gatt/characteristics/Pages/
                        CharacteristicViewer.aspx?u=org.bluetooth.characteristic.
                        heart_rate_measurement.xml
111
112                 // For all other profiles, writes the data formatted in HEX.
113                 final byte[] data = characteristic.getValue();
114                 if (data != null && data.length > 0) {
115                         final StringBuilder stringBuilder = new StringBuilder(data.length);
116                         for(byte byteChar : data)
117                         stringBuilder.append(String.format("%02X ", byteChar));
118                         intent.putExtra(EXTRA_DATA, data);
119                         intent.putExtra(EXTRA_CHARACTERISTIC, characteristic.getUuid().toString
                                ());
120                 }
121
122                 sendBroadcast(intent);
123         }
124
125         public class LocalBinder extends Binder {
126                 public BluetoothLeService getService() {
127                         return BluetoothLeService.this;
128                 }
129         }
130
131         @Override
132         public IBinder onBind(Intent intent) {
133                 return mBinder;
134         }
135
136         @Override
137         public boolean onUnbind(Intent intent) {
138                 // After using a given device, you should make sure that BluetoothGatt.close()
                        is called
139                 // such that resources are cleaned up properly.  In this particular example,
                        close() is
140                 // invoked when the UI is disconnected from the Service.
```

```
141                 close();
142                 return super.onUnbind(intent);
143         }
144 
145         private final IBinder mBinder = new LocalBinder();
146 
147         /**
148         * Initializes a reference to the local Bluetooth adapter.
149         *
150         * @return Return true if the initialization is successful.
151         */
152         public boolean initialize() {
153                 // For API level 18 and above, get a reference to BluetoothAdapter through
154                 // BluetoothManager.
155                 if (mBluetoothManager == null) {
156                         mBluetoothManager = (BluetoothManager) getSystemService(Context.
                                BLUETOOTH_SERVICE);
157                         if (mBluetoothManager == null) {
158                                 Log.e(TAG, "Unable to initialize BluetoothManager.");
159                                 return false;
160                         }
161                 }
162 
163                 mBluetoothAdapter = mBluetoothManager.getAdapter();
164                 if (mBluetoothAdapter == null) {
165                         Log.e(TAG, "Unable to obtain a BluetoothAdapter.");
166                         return false;
167                 }
168 
169                 return true;
170         }
171 
172         /**
173         * Connects to the GATT server hosted on the Bluetooth LE device.
174         *
175         * @param address The device address of the destination device.
176         *
177         * @return Return true if the connection is initiated successfully. The connection
                result
178         *         is reported asynchronously through the
179         *         {@code BluetoothGattCallback#onConnectionStateChange(android.bluetooth.
                BluetoothGatt, int, int)}
180         *         callback.
181         */
182         public boolean connect(final String address) {
183                 if (mBluetoothAdapter == null || address == null) {
184                         Log.w(TAG, "BluetoothAdapter not initialized or unspecified address.");
185                         return false;
186                 }
187 
188                 // Previously connected device.  Try to reconnect.
189                 if (mBluetoothDeviceAddress != null && address.equals(mBluetoothDeviceAddress)
190                 && mBluetoothGatt != null) {
191                         Log.d(TAG, "Trying to use an existing mBluetoothGatt for connection.");
192                         if (mBluetoothGatt.connect()) {
193                                 mConnectionState = STATE_CONNECTING;
194                                 return true;
195                         } else {
196                                 return false;
197                         }
198                 }
199 
200                 final BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);
201                 if (device == null) {
202                         Log.w(TAG, "Device not found.  Unable to connect.");
203                         return false;
204                 }
205                 // We want to directly connect to the device, so we are setting the autoConnect
206                 // parameter to false.
207                 mBluetoothGatt = device.connectGatt(this, false, mGattCallback);
208                 Log.d(TAG, "Trying to create a new connection.");
209                 mBluetoothDeviceAddress = address;
210                 mConnectionState = STATE_CONNECTING;
211                 return true;
212         }
213 
214         /**
215         * Disconnects an existing connection or cancel a pending connection. The disconnection
                result
216         * is reported asynchronously through the
217         * {@code BluetoothGattCallback#onConnectionStateChange(android.bluetooth.BluetoothGatt,
                int, int)}
218         * callback.
219         */
220         public void disconnect() {
221                 if (mBluetoothAdapter == null || mBluetoothGatt == null) {
222                         Log.w(TAG, "BluetoothAdapter not initialized");
```

```
223                         return;
224                 }
225                 mBluetoothGatt.disconnect();
226         }
227
228         /**
229         * After using a given BLE device, the app must call this method to ensure resources are
230         * released properly.
231         */
232         public void close() {
233                 if (mBluetoothGatt == null) {
234                         return;
235                 }
236                 mBluetoothGatt.close();
237                 mBluetoothGatt = null;
238         }
239
240         /**
241         * Request a read on a given {@code BluetoothGattCharacteristic}. The read result is
                reported
242         * asynchronously through the {@code BluetoothGattCallback#onCharacteristicRead(android.
                bluetooth.BluetoothGatt, android.bluetooth.BluetoothGattCharacteristic, int)}
243         * callback.
244         *
245         * @param characteristic The characteristic to read from.
246         */
247         public void readCharacteristic(BluetoothGattCharacteristic characteristic) {
248                 if (mBluetoothAdapter == null || mBluetoothGatt == null) {
249                         Log.w(TAG, "BluetoothAdapter not initialized");
250                         return;
251                 }
252                 mBluetoothGatt.readCharacteristic(characteristic);
253         }
254
255         public void writeCharacteristic(BluetoothGattCharacteristic characteristic){
256                 if (mBluetoothAdapter == null || mBluetoothGatt == null) {
257                         Log.w(TAG, "BluetoothAdapter not initialized");
258                         return;
259                 }
260
261                 mBluetoothGatt.writeCharacteristic(characteristic);
262
263         }
264
265
266         /**
267         * Enables or disables notification on a give characteristic.
268         *
269         * @param characteristic Characteristic to act on.
270         * @param enabled If true, enable notification.  False otherwise.
271         */
272         public void setCharacteristicNotification(BluetoothGattCharacteristic characteristic,
273         boolean enabled) {
274                 if (mBluetoothAdapter == null || mBluetoothGatt == null) {
275                         Log.w(TAG, "BluetoothAdapter not initialized");
276                         return;
277                 }
278                 mBluetoothGatt.setCharacteristicNotification(characteristic, enabled);
279                 BluetoothGattDescriptor descriptor = characteristic.getDescriptor(
                        CHARACTERISTIC_UPDATE_NOTIFICATION_DESCRIPTOR_UUID);
280                 descriptor.setValue(true ? BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE :
                        new byte[]{0x00, 0x00});
281
282                 mBluetoothGatt.writeDescriptor(descriptor); //descriptor write operation
                        successfully started?
283         }
284
285
286
287         /**
288         * Retrieves a list of supported GATT services on the connected device. This should be
289         * invoked only after {@code BluetoothGatt#discoverServices()} completes successfully.
290         *
291         * @return A {@code List} of supported services.
292         */
293         public List<BluetoothGattService> getSupportedGattServices() {
294                 if (mBluetoothGatt == null) return null;
295
296                 return mBluetoothGatt.getServices();
297         }
298 }
```

Listing D.5: BluetoothLeService.java

D.4. Identificadores únicos universales (UUID)

D.4.1. GattAttributes.java

```java
package com.example.fisoapp.domain;

import java.util.HashMap;

public class GattAttributes {
        private static HashMap<String, String> attributes = new HashMap();
        public static String HEART_RATE_MEASUREMENT = "00002a37-0000-1000-8000-00805f9b34fb";
        public static String CLIENT_CHARACTERISTIC_CONFIG = "00002902-0000-1000-8000-00805
            f9b34fb";
        public static String TX_CHAR = "0003CDD1-0000-1000-8000-00805f9b0131";
        public static String RX_CHAR = "0003CDD2-0000-1000-8000-00805f9b0131";
        public static String OIL_LEVEL_CHARACTERISTIC = "5909b27f-3ccc-4c5f-9fbd-89efd55ea4ec";
        public static String OIL_TEMP_CHARACTERISTIC = "869c4dca-6b7d-46e8-9b01-12e8314c57c6";
        public static String CHECKCONTROL_SERVICE = "536eb3d3-2bca-4909-8e56-d89474fa235e";


        static {
                // Sample Services.
                attributes.put(CHECKCONTROL_SERVICE, "Check Control Service");

                // Sample Characteristics.
                attributes.put(TX_CHAR, "Tx");
                attributes.put(RX_CHAR, "Rx");
        }

        public static String lookup(String uuid, String defaultName) {
                String name = attributes.get(uuid);
                return name == null ? defaultName : name;
        }
}
```

Listing D.6: GattAttributes.java

Apéndice E

Script de Matlab para ECG

En este anexo se puede consultar el *script* de Matlab completo utilizado para filtrar y representar las muestras de una señal cardíaca extraída a partir del ECG realizado con el dispositivo de procesamiento central. Adicionalmente, este código también permite el cálculo del ritmo cardíaco y del RMS de la señal.

```
%% Script that plots and filters the ECG data from a CSV file

clc; clear vars; close all; format long;

%% Read data from CSV file
filename = 'Name of the CSV file';
filedir = strcat('AdquisicionesECG/',filename);
filedir = strcat(filedir,'.csv');
data = csvread(filedir);

fs = 1000;  % Sampling frequency

%% Choose filter
n = 3;
switch n
        case 1
                disp('No filter')
        case 2  % 50 Hz filter (2 order)
                disp('Order 2 filter')
                f = 50;    % Frequency to filter
                BW = 1;     % Bandwidth
                [num,den] = iirnotch(f/(fs/2),BW/(fs/2));
                fvtool(num,den,'Fs',fs);
                data = filter(num,den,data);
        case 3  % 50 Hz filter (6 order)
                disp('Order 6 filter')
                % Filter coeficients
                num =   [0.987512174869590, -5.63519056682043, ...
                13.6815175263959, -18.0667531080674, ...
                13.6815175263959, -5.63519056682043, ...
                0.987512174869590];
                den = [1, -5.68254876635053, 13.7387485793903, ...
                -18.0664564757185, 13.6241305276252, ...
                -5.58812899963934, 0.975180295515611];
                data = filter(num,den,data);
        otherwise
                disp('No filter')
end
```

```
39
40 %% Spectrum of the filtered signal
41 y = fft(data');
42 n = length(data);        % Number of samples
43 f = (0:n-1)*(fs/n);      % Frequency range
44 power = abs(y).^2/n;     % Power of the DFT
45 power(1,1) = 0;
46
47 figure
48 plot(f,power')
49 grid minor
50 xlabel('Frequency (Hz)', 'FontSize', 20); ylabel('Power', 'FontSize
       ', 20)
51 xlim([0 100])
52
53 %% RMS and mean
54 % ADC
55 ADC_resolution = 16;  % Bits of the ADC
56 ADC_top_limit = 3.3;
57 ADC_bottom_limit = 0;
58
59 mV_data = zeros(1, length(data));
60
61 for i = 1:1:length(data)
62 mV_data(i) = ((data(i) * ADC_top_limit) / (2^ADC_resolution - 1) +
       ...
63 ADC_bottom_limit) * 1000/51;
64 end
65
66 fprintf('Mean: %f mV\n', mean(mV_data));
67 fprintf('RMS: %f mV\n', rms(mV_data));
68
69 mV_data = mV_data - mean(mV_data); % Remove offset
70
71 %% Calculate heart rate (BPM)
72 min_amp = 0.4;     % Minimun amplitud (0.4 mV)
73 min_dis = 30e-3;   % Minimun distante between peaks (30 ms)
74
75 [pks,locs] = findpeaks(mV_data, fs, 'MinPeakHeight', min_amp,...
76 'MinPeakDistance', min_dis);
77 BPM = length(locs) / (locs(end)- locs(1)) * 60;  % Beats per minute
78 fprintf('BPM: %f \n', BPM);
79
80 %% Plot filtered signal
81 i = 0.001:0.001:60; % Represent from 0 to 60 seconds
82
83 figure
84 hold on
85 plot(i, mV_data, 'Color', [0.6350 0.0780 0.1840])
86 % stem(i, mV_data, 'x', 'LineStyle', 'none')
87 grid minor
88 xlabel('Time (s)', 'FontSize', 20); ylabel('mV', 'FontSize', 20)
89
90 %% Represent the detected R peaks
91 figure
92 hold on
93 plot(i, mV_data)
94 stem(locs,pks, 'red')
```

```
95 grid minor
96 xlabel('Time (s)'); ylabel('mV')
97
98 %%
99
100 fprintf('\n')
```

Listing E.1: Scritp de Matlab para representación, filtrado rechazo banda de la señal cardíaca y cálculo del BPM.

Apéndice F

Imágenes del sistema físico

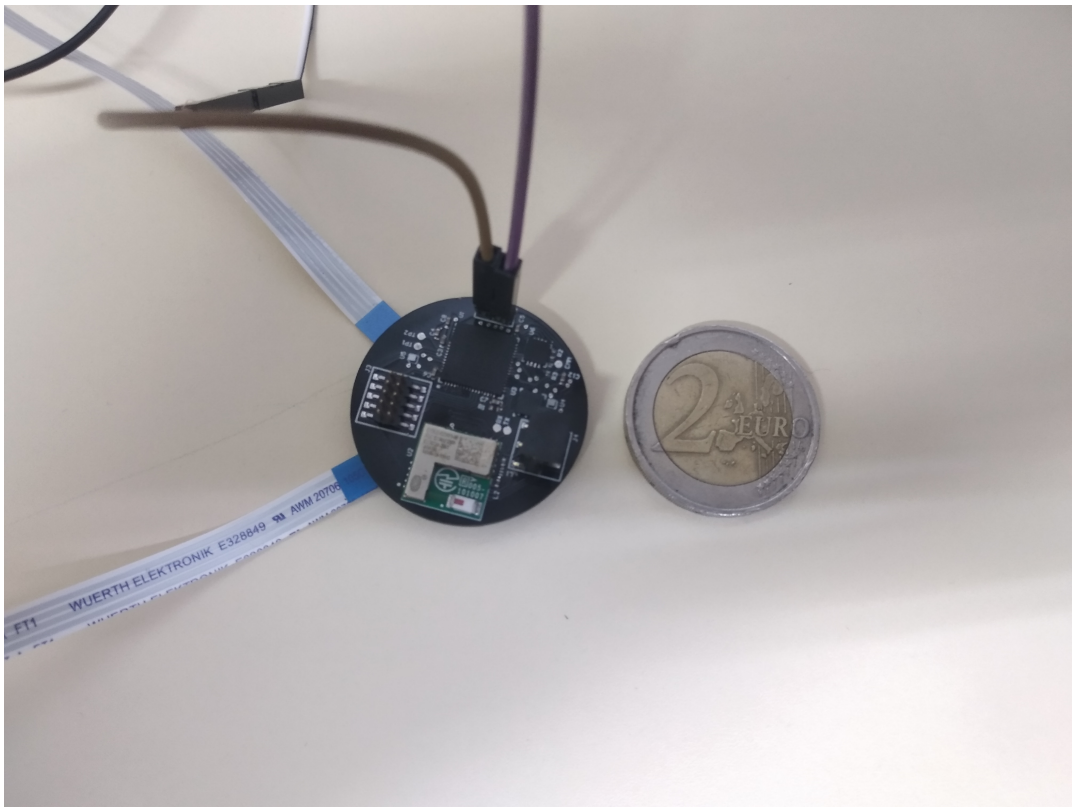


Figura F.1: Comparativa del tamaño del dispositivo de procesamiento central desarrollado respecto de una moneda de 2 euros.

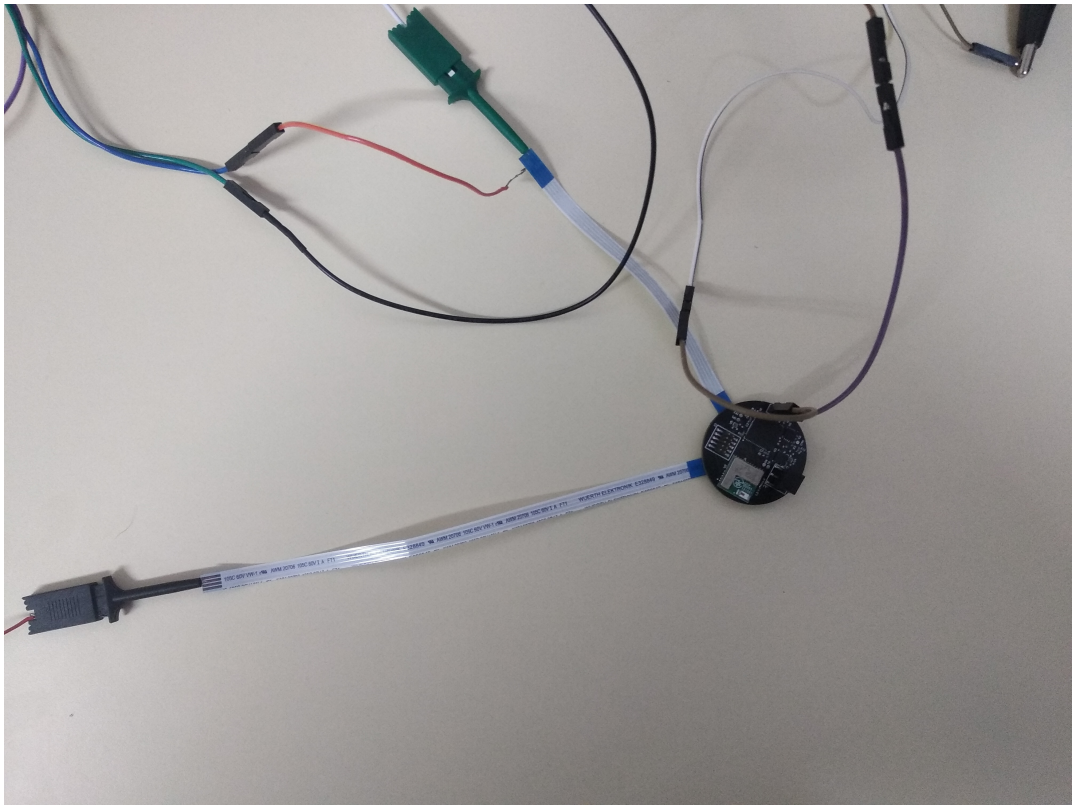


Figura F.2: Dispositivo de procesamiento central con dos cables flexibles conectados a sus puertos de entrada.

Figura F.3: Ejemplo de adquisición de la señal del ECG.