Universidad de Granada

Departamento de Arquitectura y Tecnología de Computadores

TESIS DOCTORAL

SISTEMA PARA SEPARACIÓN DE SEÑALES EN TIEMPO REAL BASADO EN DSP

Memoria presentada por:

Juan Carlos Moreno Comba

Directores de Tesis:

Carlos García Puntonet

Antonio Francisco Díaz García

Granada, Septiembre 2012

Editor: Editorial de la Universidad de Granada
Autor: Juan Carlos Moreno Comba
D.L.: GR 579-2013
ISBN: 978-84-9028-394-3

D. Carlos García Puntonet, Catedrático de Universidad, y D. Antonio F. Díaz García, Profesor Titular de Universidad del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada,

CERTIFICAN:

Que la memoria titulada "SISTEMA PARA SEPARACIÓN DE SEÑALES EN TIEMPO REAL BASADO EN DSP", ha sido realizada por D. Juan Carlos Moreno Comba bajo nuestra dirección en el Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada para optar al grado de Doctor.

Granada, Septiembre de 2012

Fdo. Carlos García Puntonet

Director de Tesis

Fdo. Antonio F. Díaz García

Director de Tesis

A todos los que me han abierto la puerta a la Luz del Conocimiento

AGRADECIMIENTOS

A mis directores de tesis, D. Carlos G. Puntonet y D. Antonio F. Díaz García, por su apoyo e inestimable ayuda durante el tiempo empleado. Quiero hacer hincapié en que la realización de este trabajo no hubiera sido posible sin el especial apoyo, guía y comprensión por parte de ellos.

También deseo expresar mi más profunda gratitud a todos los que han estado próximos a mí durante la realización de esta memoria, como mi familia, mis amigos y, a todos los que me han ayudado a ver la Luz en este trabajo.

Gracias a todos.

NOTACIÓN

Las referencias a secciones (Sec.c.n), figuras (Figura c.n), tablas (Tabla c.n), ecuaciones y demás expresiones matemáticas ((c.n)), se hacen incluyendo en primer lugar el número, *c*, de capítulo (1,2,..., 6) o apéndice (A) del que se trata, seguido de un número, *n*, de orden correlativo dentro del capítulo o apéndice.

Las referencias bibliográficas se codifican entre corchetes, con las tres primeras letras del primer apellido del primer autor, seguidas de las dos últimas cifras del año de publicación. Caso de que se efectúen varias citas en el mismo año el primer autor tenga varias referencias, se incluye una letra minúscula (a, b, c,...) adicional, para identificar cada una de ellas, por ejemplo: [CAR96a], [CAR96b] corresponden a dos citas de 1996 del mismo autor. Las referencias bibliográficas completas se incluyen al final de la memoria ordenadas según el código indicado anteriormente: orden alfabético/año de publicación.

Tabla de Contenidos

Tabla de Contenidos ..... *xi*

Índice de Figuras ..... *xv*

Índice de Tablas ..... *xix*

Resumen ..... *xxi*

Abstract ..... *xxiii*

1 INTRODUCCIÓN, OBJETIVOS Y ESTRUCTURA DE LA TESIS *25*

1.1 Introducción **25**

1.2 Objetivos principales y nuevas aportaciones **32**

1.2.1 Objetivos principales 32

1.2.2 Motivaciones del trabajo 33

1.2.3 Nuevas aportaciones 35

1.3 Estructura de la tesis **36**

2 PROCESAMIENTO DIGITAL DE SEÑALES *39*

2.1 Estadística, probabilidad y ruido **39**

2.1.1 Terminología de señales y gráficos 39

2.1.2 Desviación media y estándar 41

2.1.3 Señal y procesos subyacentes 44

2.1.4 Histograma, PMF y PDF 46

2.1.5 Distribución normal 52

2.1.6 Generación de ruido digital 55

2.1.7 Precisión y exactitud 59

2.2 Conversión AD y conversión DA **60**

2.2.1 Conversión digital-analógica 61

2.2.2 Filtros analógicos para conversión de datos 65

2.2.3 Conversión de datos multirango 69

2.2.4 Conversión de datos de bit único 70

3 ARQUITECTURAS BASADAS EN DSPs 79

3.1 Características de los DSP 80

3.2 Arquitectura interna de los DSP 83

3.2.1 Diferencias de un DSP con una CPU convencional 83

3.2.2 Evolución de los DSP 84

3.2.3 DSP TMs320C2x 87

3.2.4 Plataforma EZLITE 90

3.3 DSP: TMS320C6000 VLIW de Texas Instruments 94

3.3.1 Restricciones en el acceso a registros del C6000 96

3.4 Métodos actuales de optimización código del DSP TMS320C6416 97

3.4.1 Optimización explícita del código 102

3.4.2 Aplicando palabras clave restringidas 103

4 SEPARACIÓN CIEGA DE FUENTES 109

4.1 Separación ciega de fuentes mediante métodos geométricos 109

4.2 Algoritmos ICA 114

4.2.1 Definición de ICA 118

4.2.2 Ambigüedades de ICA 120

4.2.3 Ilustración de ICA 120

4.2.4 Definición y propiedades fundamentales 122

4.2.5 Las variables no correlacionadas son sólo parcialmente independientes .. 123

4.2.6 Principios de la estimación de ICA 125

4.2.7 Minimización de la Información Mutua 133

4.2.8 ICA y la proyección perseguida 137

4.2.9 Preprocesado para ICA 138

4.2.10 Preprocesado adicional 141

4.3 FastICA 142

4.3.1 FastICA para una unidad. 144

4.3.2 FastICA para varias unidades 146
4.3.3 FastICA y la máxima verosimilitud 147
4.3.4 Propiedades del algoritmo FastICA 148
4.4 Estudios actuales de métodos basados en ICA 149

5 NUEVOS ALGORITMOS DE SEPARACIÓN EN MEDIOS NO ESTACIONARIOS 151

5.1 Algoritmo EASI 152
5.2 Algoritmos MeR-MaID-SIG y SIPEX-G 154
5.3 Algoritmo ICAdec 158
5.4 Algoritmo ASWICA (Adaptive Sliding Window ICA) 162
5.4.1 Estimación de tamaño óptimo de ventana 163
5.5 Algoritmo ADSWICA (Adaptive Dual Sliding Window ICA) 172
5.6 Conclusiones 174

6 IMPLEMENTACIÓN DE LOS NUEVOS ALGORITMOS EN DSP 177

6.1 Implementación de los algoritmos en el DSP 178
6.2 Optimizaciones para el DSP 179
6.2.1 Optimización de la organización de la memoria 179
6.2.2 Optimización de código 184
6.2.3 Optimizaciones globales 189
6.3 Optimización mediante algoritmos genéticos 194
6.4 Conclusiones 197

7 CONCLUSIONES 199

7.1 Principales aportaciones y conclusiones 199
7.2 Trabajos futuros 201
7.3 Trabajos publicados relacionados 202

BIBLIOGRAFÍA *203*

ACRÓNIMOS *221*

Apéndice A *223*

Apéndice B *229*

1. Iteración tamaño constante 229

2. Iteración tamaño variable 234

3. Iteración cálculo covarianza tamaño fijo 238

4. Iteración cálculo covarianza tamaño variable 249

Índice de Figuras

Figura 1.1 Modelo general de mezcla y separación de señales. 26
Figura 1.2 Modelo de mezcla convolutiva. 28
Figura 1.3 Separación ciega de fuentes. 29
Figura 2.1 Ejemplos de dos señales digitalizadas con diferentes media y desviación estándar. 40
Figura 2.2 Relación entre la amplitud rms y la desviación estándar para varias formas de onda comunes. 42
Figura 2.3 Ejemplos de señales generadas desde procesos no estacionarios. 45
Figura 2.4 Ejemplos de histogramas. (a) 128 muestras de una señal muy grande, con cada muestra siendo un entero entre 0 y 255; (b) y (c) histogramas usando 128 y 256.000 muestras de la señal, respectivamente. 47
Figura 2.5 Tres formas de onda comunes y sus funciones de densidad de probabilidad. ... 49
Figura 2.6 Ejemplo de histograma con binning..... 51
Figura 2.7 Ejemplos de curvas gaussianas: (a) forma de la curva incipiente sin normalización o la adición de parámetros ajustables; (b) y (c), curva gaussiana completa es mostrada para varias medias y desviaciones estándar. 53
Figura 2.8 Φ(x), la función de distribución acumulativa de la distribución normal (media=0, desviación estándar=1). 54
Figura 2.9 Conversión de una distribución uniforme en una distribución gaussiana. (a) señal donde cada muestra es generada por un generador de números aleatorios; (b) cada muestra está formada por la suma de dos valores procedentes del generador de números aleatorios; (c), cada muestra es creada sumando doce valores provenientes del generador de números aleatorios. 56
Figura 2.10 Definiciones de exactitud y precisión. 60
Figura 2.11 Teorema de muestreo en el dominio del tiempo y la frecuencia..... 62
Figura 2.12 Análisis de la conversión digital a analógica. 65
Figura 2.13 Filtros analógicos electrónicos utilizados para cumplir con el teorema de muestreo. 65
Figura 2.14 Opciones para la digitalización de una señal codificada en el dominio de tiempo. 68

Figura 2.15 Diagrama de bloques típico de un modulador delta. 71
Figura 2.16 Ejemplo de las señales producidas por el modulador delta en la Figura 2.14. Figura (a) muestra la señal de entrada analógica, y la correspondiente tensión en el condensador. La figura (b) muestra la salida delta modulada, un flujo digital de unos y ceros. 73
Figura 2.17 Diagrama de bloques modulación CVSD. Una lógica del circuito se añade a la base del modulador delta para mejorar pendiente. 75
Figura 2.18 Diagrama en bloque de un conversor delta-sigma analógico-digital. En el caso más simple, los impulsos de un modulador delta se cuentan para un número predeterminado de ciclos de reloj. La salida del contador es capturada para completar la conversión. En un circuito más sofisticado, los pulsos se pasan a través de un filtro digital pasa bajo y, a continuación, remuestreado (diezmado) a una menor tasa de muestreo..... 76
Figura 3.1 Etapas principales en la evolución de los DSP. 85
Figura 3.2 Diagrama de bloques de un video-teléfono basado en el TMS320DM6446. 86
Figura 3.3Familias mono-núcleo y multi-núcleo de DSP de FreeScale. 86
Figura 3.4 Familia de DSPs de Texas Instruments. 87
Figura 3.5 Elementos principales de procesamiento de los DSP TMS320C2x..... 88
Figura 3.6 Implementación de un filtro FIR 89
Figura 3.7 Diagrama general del sistema. 91
Figura 3.8 Prototipo desarrollado. 92
Figura 3.9 Diagrama de bloques de la familia C6x de Texas Instrument. 95
Figura 3.10 Cronograma de la segmentación de cauce. 98
Figura 3.11 Cronograma de la segmentación de cauce de alto nivel para el DSP C6000... 99
Figura 3.12 Cronograma de la segmentación de cauce de alto nivel para el DSP C6000. 100
Figura 4.1 Hiperpolígono del espacio de observación. 111
Figura 4.2 Traslación del hiperparalelepípedo. 112
Figura 4.3Traslación en un caso p=3. 112
Figura 4.4 Señales originales. 115
Figura 4.5 Las mezclas observadas de las señales de la figura 4.4. 116
Figura 4.6 Estimaciones de las señales originales teniendo en cuenta sólo las señales observadas. 116
Figura 4.7 Señales obtenidas originalmente en un ECG fetal. 117
Figura 4.8 Diferentes señales obtenidas aplicando PCA e ICA. 118

Figura 4.9 La distribución conjunta de las componentes independientes s1 y s2 con distribuciones uniformes. En el eje horizontal s1 y en el vertical s2..... 121
Figura 4.10 La distribución multicambiante de dos variables gaussianas..... 124
Figura 4.11 La función densidad de la distribución de Laplace, la cual es una distribución típica supergaussiana. Por comparación, la densidad de la gaussiana es dada por una línea muy pendiente. Ambas densidades están normalizadas a varianza unidad..... 127
Figura 4.12 Ilustración de la proyección perseguida y las proyecciones no-gaussianas... 138
Figura 4.13 La distribución conjunta de las mezclas blanqueadas..... 141
Figura 5.1 Distribución de dos señales con mezcla estática (a) y dinámica(b)..... 151
Figura 5.2 Simulación muestra el efecto del tamaño del paso en SER(symbol error rate)154
Figura 5.3 Vista superior de dos fuentes de sonido y un observador..... 156
Figura 5.4 Diagrama de bloques para BSS de N=2 fuentes y observadores..... 156
Figura 5.5 Relación entre una de las entradas originales y la componente extraída en ICAdec e ICAstat. 162
Figura 5.6 Diagrama de bloques del algoritmo ASWICA..... 163
Figura 5.7 Separación de señales con FastICA muestreadas a 10 KHz. 165
Figura 5.8 Separación de señales con FastICA muestreadas a 5 KHz. 166
Figura 5.9 Separación de señales con FastICA muestreadas a 5 KHz. 166
Figura 5.10 Dispersiones obtenidas para 5 KHz, 10 KHz y 20 KHz.. 167
Figura 5.11 Señales originales y separadas con 100 muestras. 168
Figura 5.12 Las 100 primeras muestras de las señales originales y separadas utilizando 1000 y 4096 muestras..... 169
Figura 5.13 Valores de la matriz de mezcla y amplitud de la señal recibida. 170
Figura 5.14 Relación entre el tamaño de la ventana y la estimación..... 171
Figura 5.15 Salida de la Fase III para un movimiento senoidal y un movimiento triangular. 171
Figura 5.16 Estimación aplicada en la Fase IV 172
Figura 5.17 Diagrama de bloques del algoritmo ADSWICA..... 172
Figura 5.18 Estimaciones de la ventana de 4000 y 7000 puntos en la Fase II. 173
Figura 5.19 Señal real y estimada aplicando el algoritmo ADSWICA. 174
Figura 6.1 Anchos de banda entre las memorias cache internas de la familia C64x de DSPs. 180
Figura 6.2 Procesamiento con buffer circular y ventanas. 190

Figura 6.3 Ganancias de velocidad obtenida en el DSP en diversas configuraciones..... 197

Índice de Tablas

Tabla 3.1Algunas aplicaciones del PDS. 81
Tabla 3.2 Areas de la ciencia y la tecnología en las cuales se emplea PDS. 82
Tabla 5.1 Coeficientes de correlación obtenidos en función de la frecuencia de muestreo. 167
Tabla 5.2 Coeficientes de correlación obtenidos en función del número de muestras..... 169
Tabla 6.1 Número de ciclos del DSP para diversas implementaciones de código de la Ecuación 6.1. 185
Tabla 6.2 Número de ciclos del DSP para diversas implementaciones del cálculo de la matriz de covarianza. 187
Tabla 6.3 Número de ciclos consumidos por el DSP para funciones sqrt()(y pow(). 189
Tabla 6.4 Tiempos de ejecución obtenidos con el DSP y un Intel Core i5 con Matlab. ... 193
Tabla 6.5 Número medio de iteraciones para obtener convergencia en Matlab y el DSP. 194

Resumen

En la bibliografía científica se conoce como “Separación Ciega de Señales” o “Separación Ciega de Fuentes” al procedimiento que consiste en intentar recuperar *p* señales originales o fuentes a partir de, únicamente, las señales proporciona por *q* sensores en un medio lineal, no lineal o convolutivo. Es, por tanto, una separación ciega porque no se conoce el medio en el que se realiza la mezcla (que puede modelarse por una matriz de coeficientes) ni se tiene acceso a las señales originales o fuentes anteriores al proceso de mezcla (ver Figura 1.1).

El presente trabajo pretende contribuir en el ámbito de la Separación Ciega de Señales aportando un sistema autónomo para su procesamiento, empleando para ello Procesadores Digitales de Señales (DSP), cubriéndose los siguientes objetivos:

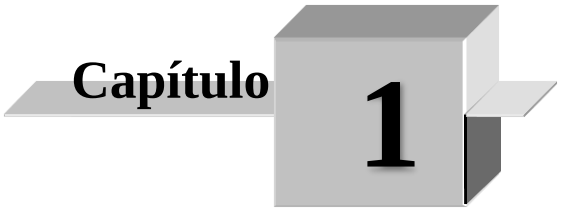
- Realizar una introducción general al Procesado Digital de Señales (PDS), así como cuestiones relacionadas con la conversión A/D, D/A y las soluciones empleadas.
- Describir los Procesadores Digitales de Señales (DSP), sus características y arquitectura interna, mostrando el DSP utilizado, la plataforma de desarrollo y las técnicas actuales de optimización de código para dicha arquitectura.
- Mostrar nociones de la Separación Ciega de Fuentes, y su resolución mediante Análisis en Componentes Independientes, así como una descripción del algoritmo FastICA que es utilizado posteriormente en este trabajo.
- Presentar algunos algoritmos empleados en la separación ciega de fuentes cuando la matriz de mezcla cambia en el tiempo, planteando inicialmente el estado actual, y mostrando algoritmos que se encuentran en la bibliografía para, posteriormente, presentar dos nuevos algoritmos propuestos en esta tesis (ASWICA y ADSWICA), basados en ventanas deslizantes con FastICA.
- Describir cuestiones relacionadas con la optimización del código para DSP de altas prestaciones, así como los resultados del análisis, implementación y optimización de los algoritmos basados en ICA, ASWICA y ADSWICA.
- Desarrollar un sistema automático de optimización de código en DSPs para los algoritmos implementados anteriormente.

Abstract

In the bibliography, Blind Source Separation is a procedure that tries to recover the *p* original sources from the only previous knowledge of the *q* sensors in a lineal, non-lineal or convolutive media. That is, a blind separation without knowledge of the media where the mixture is made (it could be modeled by a coefficients matrix) it hasn´t accessed to the original sources too, previous sources to the mixture process (see Figure 1.1).

This work tries to resolve this issue cover different algorithms and computer experiments, and it aims to contribute to the development of Blind Signal Separation providing an autonomous system for management using for this purpose Digital Signal Processors. The main objectives are:

- Make a general introduction to the Digital Signal Processing (DSP), as to questions related to A/D, D/A conversions and the solutions used.
- Describe the Digital Signal Processors (DSP), their characteristics and internal architecture, showing the DSP used, the develop platform and the actual techniques for the optimization of code for this architecture.
- Show notions of Blind Source Separation, its resolution with Independent Component Analysis, and a description of the FastICA algorithm.
- Present some algorithms for the Blind Source Separation with a Time-Varying Mixing Matrix, showing some algorithms that are in the bibliography and later two algorithms are proposed in this thesis (ASWICA and ADSWICA), based in sliding windows with FastICA.
- Describe questions related to the code optimization for high performance DSP and the results of the analysis, implementation and optimization of the algorithms based in ICA, ASWICA and ADSWICA.
- Develop an automatic system of code optimization for DSP for the algorithms previously implemented.



Resumen:

En este capítulo se presenta una breve introducción de la separación ciega de señales y se presentan los principales objetivos del presente trabajo y las motivaciones del desarrollo, así como las principales aportaciones que se han realizado. También se presenta la estructura de esta memoria, resumiendo brevemente los contenidos de cada capítulo.

1 INTRODUCCIÓN, OBJETIVOS Y ESTRUCTURA DE LA TESIS

1.1 Introducción

El problema de Separación Ciega de Fuentes fue enunciado por primera vez en 1985 por J. Hérault, C. Jutten y B. Ans, y consiste en la obtención de las señales generadas por *p* fuentes a partir de las mezclas detectadas por *q* sensores, conociendo tan sólo estas últimas. La mezcla de señales tiene lugar en el medio en que se propagan y en los sensores. El concepto de la Separación Ciega de Señales se ha empezado a desarrollar en una gran diversidad de campos (radiocomunicaciones, procesamiento de voz y de imágenes, biomedicina, sensores inteligentes, etc.).

Se han desarrollado diversos algoritmos para la resolución de la Separación Ciega de Fuentes, que en la actualidad se considera un caso particular del Análisis en Componentes Independientes, comúnmente denominado ICA (del ingles "Independent Component Analysis"). Uno de los algoritmos ICA más empleados, dado su relativo bajo nivel de exigencia computacional y alto grado de precisión es FastICA [HYV99a].

El problema de separación ciega de fuentes [PRI99c] consiste en la obtención de las señales generadas por p fuentes, s_j, $j=1,\ldots,p$, a partir de las mezclas, e_i, $i=1,\ldots, q$,

detectadas por q sensores. La mezcla de señales tiene lugar en el medio en el que se propagan (Figura 1.1), siendo:

$$\boldsymbol{e_i(t) = F_i\left(s_1(t),..,s_j(t),..,s_p(t)\right)} \qquad \boldsymbol{i = 1,...q} \tag{1.1}$$

Donde *Fi*: $p \rightarrow \Re$ es una función de p variables del espacio s al espacio e. En notación vectorial:

$$\boldsymbol{e(t) = F\left(s(t)\right)} \tag{1.2}$$

Siendo:

$$\boldsymbol{e(t) = \left(e_i(t)\right)^T \;\; ; \;\; s(t) = \left(s_i(t)\right)^T \;\; ; \;\; F = (F_i)^T} \tag{1.3}$$

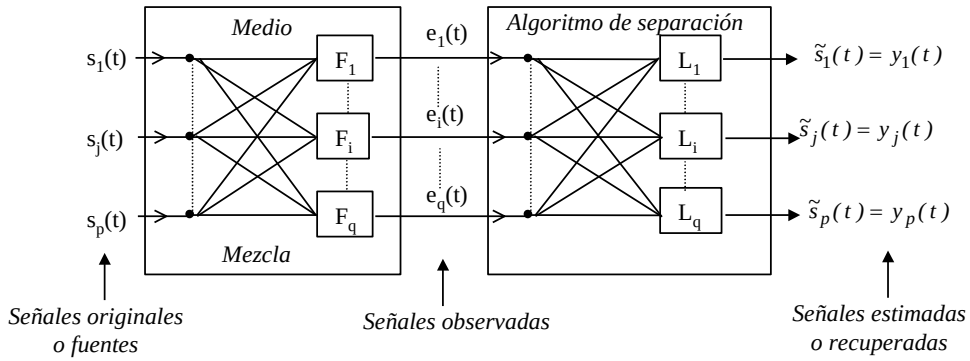


Figura 1.1 Modelo general de mezcla y separación de señales.

La finalidad de la separación de fuentes es obtener p funciones L_j, tales que:

$$\boldsymbol{s_j(t) = L_j\left(e_1(t),..,e_i(t),..,e_q(t)\right)} \quad \boldsymbol{j = 1,..,p} \tag{1.4}$$

Donde L_j: $\Re\, q \rightarrow \Re$ es una función de q variables del espacio e al espacio s. En notación vectorial:

$$\boldsymbol{s(t) = L\big(e(t)\big)} \tag{1.5}$$

Siendo $L = \left(L_j\right)^T$, donde L, representa el conjunto de transformaciones, L_j, y es la inversa del conjunto F de transformaciones F_i.

El modelo no lineal representado ha sido estudiado por diversos autores, pero es demasiado general y con frecuencia se considera un modelo post-nolineal (PNL) en el que la señal se propaga a través de un canal de transmisión lineal tras el cual los sensores introducen no linealidades. Por tanto, (1.1) se puede expresar como:

$$\boldsymbol{e_i(t) = F_i\left(\sum_{j=1}^{p} a_j s_j(t)\right)} \tag{1.6}$$

Un modelo más concreto considera que el medio de propagación actúa como un filtro con p entradas (las fuentes) y q salidas (las señales detectadas por los sensores). Usualmente se admite que este filtrado, introducido por el medio y los sensores, es lineal y estacionario. De esta manera, las señales captadas pueden expresarse de la forma:

$$\boldsymbol{e_i(t) = \sum_{j=1}^{p} \int a_{ij}(t-\tau)\cdot s_j(\tau)d\tau + n_i(t) \quad \forall i = 1,..,q} \tag{1.7}$$

En esta expresión $a_{ij}(t)$ representa la respuesta a un impulso de un filtro que modela la propagación entre la fuente i y el sensor j. El término $n_i(t)$ corresponde a un ruido aditivo.

En el dominio de la frecuencia, ν, la expresión (1.6) puede escribirse como (ver Figura 1.2 [PRI99c]):

$$\boldsymbol{e_i(\nu) = \sum_{j=1}^{p} A_{ij}(\nu)\cdot s_j(\nu) + n_i(\nu) \quad \forall i = 1,..,q} \tag{1.8}$$

En forma vectorial:

$$\boldsymbol{e(\nu) = A(\nu)\cdot s(\nu) + n(\nu)} \tag{1.9}$$

Donde la matriz *A(ν)*, de dimensión *pxq*, contiene todas las funciones de transferencia entre las p fuentes y los q sensores.

Se puede realizar una simplificación adicional consistente en que la propagación no introduce ningún retardo temporal en cuyo caso las señales detectadas pueden expresarse como (Figura 1.3 [PRI99c]):

$$\boldsymbol{e_i(t) = \sum_{j=1}^{p} a_{ij} \cdot s_j(t) + n_i(t) \quad \forall i = 1,..,q} \tag{1.10}$$

Con notación vectorial:

$$\boldsymbol{e(t) = A \cdot s(t) + n(t)} \tag{1.11}$$

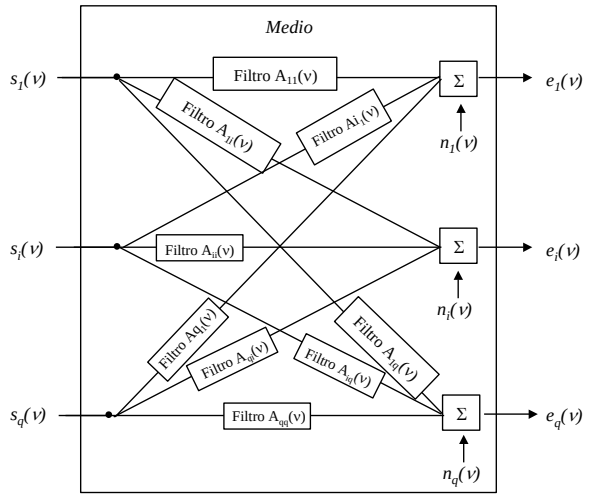


Figura 1.2 Modelo de mezcla convolutiva.

La mezcla obtenida según las expresiones (1.9) y (1.10) se denomina mezcla lineal.

En resumen, en el problema de separación de fuentes se suelen considerar los siguientes modelos:

- **Modelo general (no lineal):** expresado analíticamente según(1.1) y (1.2).
- **Modelo post-no lineal:** donde el medio es lineal y la no-linealidad es introducida en los sensores (1.3) y (1.4).
- **Modelo convolutivo:** el medio y los sensores se comportan como filtros lineales y estacionarios. El modelo convolutivo básico utiliza una representación en el

dominio del tiempo (1.6), y el modelo de mezcla espectral corresponde a la mezcla convolutiva en el dominio de la frecuencia. En este último caso, los valores de las señales y de los elementos de la matriz de mezcla son complejos y vienen descritos por la expresión (1.8).

- **Mezcla lineal:** se considera que tanto el medio como los sensores se comportan de forma lineal (1.9) y (1.10). Este es el modelo que será tratado en el presente documento.

Con frecuencia se consideran las mezclas sin ruido de forma que, por ejemplo, para el caso lineal se tiene que:

$$\boldsymbol{e(t) = A \cdot s(t)} \tag{1.12}$$

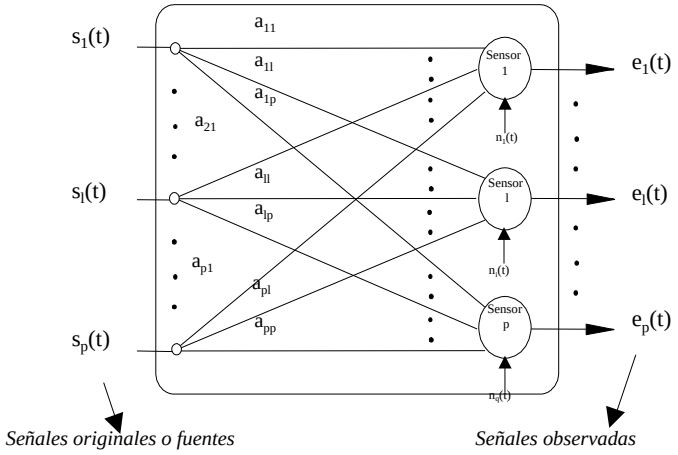


Figura 1.3 Separación ciega de fuentes.

La resolución del problema de separación ciega de fuentes puede enunciarse así:

- Dadas M observaciones independientes del vector de mezcla *e(t)*, encontrar una estimación:

$$\boldsymbol{W \overset{\text{def}}{=} A} \tag{1.13}$$

- De la matriz de mezcla *A*. Con la inversa de esta matriz:

$$\boldsymbol{B} \stackrel{\text{def}}{=} \boldsymbol{W^{-1}} \tag{1.14}$$

- Se podrán recuperar (estimar) las señales originales:

$$\boldsymbol{y} \stackrel{\text{def}}{=} \boldsymbol{s} = \boldsymbol{W^{-1}} \cdot \boldsymbol{e} = \boldsymbol{B} \cdot \boldsymbol{e} \tag{1.15}$$

Los algoritmos de separación de fuentes tratan de explotar principalmente la diversidad espacial producida por la separación de los distintos sensores, lo que hace que cada uno de ellos capte, en cada instante de tiempo, distintas mezclas de las fuentes. Por otra parte, los modelos convolutivos se proponen con objeto de aprovechar la diversidad espectral, aunque el enfoque fundamental de la separación de fuentes es esencialmente espacial y busca la estructura a través de los sensores, no a lo largo del tiempo.

Desde sus inicios, la separación ciega de fuentes se ha abordado fundamentalmente bajo dos enfoques:

- ***Enfoque estadístico:*** parte de la hipótesis de que las señales originales son estadísticamente independientes. Basándose en propiedades de las distribuciones de probabilidad de las fuentes y de las observaciones, caracterizadas por las distribuciones respectivas o por sus cumulantes. Se pretende que las señales recuperadas sean tales que cumplan un determinado criterio estadístico o basado en la Teoría Estadística de la Información.
- ***Enfoque geométrico:*** se basa en las propiedades geométricas y algebraicas de las señales originales y de las observaciones, considerando el tipo de transformación realizada por la matriz de mezcla. Este enfoque ha sido propuesto originalmente por investigadores del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada.

También cabe mencionar que los distintos métodos de resolución del problema de la Separación Ciega de Señales se basan básicamente en alguno o algunos de los siguientes

criterios, e incluso estando, en algunos casos, muy relacionados entre sí los distintos métodos:

- Mínima información mutua entre las señales de salida. Conduce a las funciones de contraste Φ_{MI} y Φ_{ICA}.
- Minimización de la divergencia entre las distribuciones de salida, *f(y),* y las del modelo de fuentes, *f(s)* Se obtiene la función de contraste de máxima verosimilitud, Φ_{ML}.
- La mezcla reduce la entropía de las fuentes, supuestas éstas estadísticamente independientes. El criterio consiste en maximizar la entropía de las salidas (Φ_{EM}).
- Las funciones de distribución se pueden caracterizar adecuadamente con sus cumulantes. Para que las señales de salida estimen a las fuentes, se minimiza la distancia entre los cumulantes de salida y los cumulantes del modelo de fuentes.
- Minimizar el error cuadrático medio entre las señales observadas, e_k, y una reproducción de dichas señales. El método se denomina análisis de componentes principales no lineal y la función de contraste se denominará Φ_{PCAnl}.

Tras esta visión general, se puede mencionar que algunos de los **principales métodos que parten del enfoque estadístico**:

- Arquitectura Neuromimética de Hérault-Jutten.
- Basado en la minimización de la dependencia estadística. ICA.
- Estimación de máxima verosimilitud (ML-Maximum Likelihood).
- Basadas en el principio de máxima entropía.
- Basadas en medidas estadísticas de alto orden.
- Análisis de componentes principales no lineal (PCA no lineal).

Por otro lado, están los **métodos basados en el enfoque geométrico**, cuyo fundamento se va a tratar de exponer brevemente a continuación.

El problema de separación de fuentes consiste en encontrar una matriz $\boldsymbol{W}$ similar a la matriz $\boldsymbol{A}$, partiendo sólo de vectores de observación $\boldsymbol{e(t)}$. Los algoritmos geométricos se basan en tres propiedades algebraicas y geométricas. Siendo el objetivo de la resolución del problema de separación de fuentes obtener una matriz W similar a A. Pero, los algoritmos de separación usualmente no obtienen una matriz W_0 exactamente similar a A,

sino una serie de matrices *W(t)* que, dentro de un proceso iterativo, se van sucesivamente aproximando cada vez más a W_0. En ambos casos interesa medir el grado de aproximación de *W(t)* a W_0. Para ello se van a utilizar a lo largo de la memoria unos índices que medirán de forma objetiva la aproximación de *W(t)* a W_0 (o "calidad" de la separación), o su variación en el tiempo dentro de un proceso iterativo que indicará el grado de convergencia del método utilizado para la separación. Donde los índices que se van a usar son:

- Índice de prestaciones de Amari.
- Error cuadrático medio entre los elementos de *A* y *W*.
- Error cuadrático medio entre *y(t)* y *s(t)*.
- Diafonía ("Crosstalk").

Esta forma de abordar el problema de separación de señales fue propuesta originalmente por investigadores del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada [PUN95a]. La idea fundamental radica en que con cualquier conjunto de p vectores de las p aristas (vectores arista) que incidan en un vértice cualquiera del hiperparalelepípedo contenido en el espacio de observaciones, se puede formar una matriz *W* que estima a *A*. La matriz *W* tiene como elementos los vectores de las aristas ordenados en columnas.

1.2 Objetivos principales y nuevas aportaciones

1.2.1 Objetivos principales

El presente trabajo pretende contribuir al estudio y aplicación de la Separación Ciega de Señales mediante el empleo de sistemas basados en Procesadores Digitales de Señales (DSP) y herramientas como Matlab y lenguajes de programación como C++. Se trata de abordar los siguientes objetivos:

- Realizar una introducción general al Procesado Digital de Señales (PDS), así como cuestiones relacionadas con la conversión A/D, D/A y las soluciones empleadas.
- Conocer los Procesadores Digitales de Señales, su amplio espectro de aplicación en este campo así como su arquitectura.

- Presentar de forma completa y compacta los fundamentos teóricos de los métodos de separación ciega de fuentes basados en el análisis en componentes independientes, y en especial FastICA.
- Conocer el estado del arte en la separación de señales con mezcla no estacionaria.
- Proponer nuevos algoritmos que ofrezcan una buena aproximación en su resolución.
- Implementación de los algoritmos de separación de señales para DSP.
- Estudio de las nuevas técnicas de optimización de código para DSP que puedan aplicarse en la implementación de los algoritmos.
- Evaluar técnicas automáticas de optimización del código que permitan mejorar las implementaciones de los algoritmos para la familia de DSPs TMS320C641x.

1.2.2 Motivaciones del trabajo

En los últimos años, desde los primeros estudios realizados sobre separación de señales, avance y aplicación ha sido incesante, especialmente en los campos en los que sus aplicaciones son más visibles (biomedicina, comunicaciones, defensa). Dada la importancia e interés de algunas de sus aplicaciones se hace más motivador su estudio.

Siempre que se tenga un sistema que emita señales que se propaguen por un medio, se reciban en unos sensores e interese recuperar dichas fuentes con independencia de cualquier ruido o mezcla que haya podido sufrir en su recorrido, se tiene un posible campo de aplicación de la separación ciega de fuentes. Algunas de las principales aplicaciones que se pueden mencionar son:

- Aplicaciones derivadas de la supresión del ruido. Se han estudiado y detectado numerosos tipos de ruido en diversos dispositivos; por ejemplo, el producido por efecto Shot (transporte), el generado por efecto Flicker (modulación), el ruido térmico (fluctuaciones), el ruido Johnson, el ruido impulsivo (atmosférico y solar) y el ruido cuántico (optoelectrónica, superconductores, dispositivos de electrones calientes), eliminación de ruido en imágenes. Hay que tener en cuenta la

importancia que tiene la supresión del ruido en sistemas reales como la voz para las telecomunicaciones.

- Aplicaciones biomédicas. Para mejorar los métodos tradicionales de identificación y medida de respuestas, que se basaban en las amplitudes y latencias de los picos de la forma de onda y en modelos de dipolos múltiples que proporcionaban resultados ambiguos cuando la geometría de las fuentes es desconocida o compleja. Existen líneas de investigación para aplicaciones de tratamiento de señal en biomedicina y de otros fenómenos biológicos, tales como ECG, EEG, fRMI, PET, etc.
- Aplicaciones de audio. Uno de los orígenes de la separación ciega de fuentes se basó en encontrar una explicación al efecto "cocktail party", consistente en la habilidad del ser humano de poder centrar su atención y escuchar a una sola persona, aun habiendo más personas hablando a la vez o existiendo otras fuentes sonoras. En este campo se ha investigado notablemente y podemos disfrutar de la aplicación de parte de dichos avances, como por ejemplo en el campo de la música.
- Aplicaciones en radiocomunicaciones. Donde algunos de sus avances los empleamos a diario, como son: tratamiento de antenas, radar, comunicaciones móviles, separación de mensajes en sistemas de comunicación de múltiple acceso con división de código (CDMA), enlaces ionosféricos de alta frecuencia (HF), señales de identificación por radiofrecuencia de blancos múltiples utilizando separación ciega de fuentes con redes neuronales artificiales, separación de fuentes captadas y/o generadas por satélites de telecomunicaciones.
- Control de máquinas eléctricas rotativas. En una sala con gran cantidad de máquinas eléctricas se puede detectar el sonido individual producido por cada una de ellas, y localizar si se produce alguna avería en alguna máquina.
- Sensores inteligentes. En sensores eléctricos y magnéticos la señal a detectar suele estar enmascarada con señales de otro origen, usualmente térmico. Se trata de separar automáticamente la influencia de las distintas fuentes.
- Otras aplicaciones: Análisis de datos sísmicos, separación de fuentes ópticas, eliminación del ruido en imágenes y procesamiento, en general, de imágenes, clasificación no supervisada, análisis de datos financieros y predicción de series temporales, etc..

Por todo ello, el presente trabajo resulta suficientemente motivado como para tratar de contribuir al desarrollo del estudio y aplicación de la Separación Ciega de Fuentes

mediante el empleo de sistemas como las plataformas de desarrollo de Procesadores Digitales de Señales (DSP) y herramientas tan importantes como Matlab o lenguajes de programación, como C++.

1.2.3 Nuevas aportaciones

En este trabajo se estudia la separación ciega de señales, así como su aplicación e implementación basada en DSP. Las principales aportaciones de esta tesis se centran en tres elementos principales:

- **Desarrollo de dos nuevos algoritmos adaptativos para separación de fuentes con mezclas no estacionarias: ASWICA y ADSWICA**. Los algoritmos propuestos ofrecen mejores resultados que los descritos en la bibliografía, ya que la estimación no se basa sólo en la obtención de los coeficientes de la matriz de mezcla, sino que dichos coeficientes cambian según una función cúbica adaptativa, de igual forma que en un entorno real los coeficientes varían de forma progresiva y continua. A pesar de un mayor coste computacional, estos algoritmos obtienen mejores resultados cuando hay un número suficiente de muestras.
- **Análisis, implementación y optimización de los nuevos algoritmos ICA, ASWICA y ADSWICA en DSP de altas prestaciones para aplicaciones en tiempo real**. Aplicaciones como las que se abordan en este trabajo requieren un elevado coste computacional y es necesaria una optimización manual para poder aprovechar las prestaciones que ofrecen los DSP. Para saber cómo optimizar el código, se han realizado un conjunto de medidas que han determinado la mejor forma de llevar a cabo la optimización y los elementos fundamentales del procedimiento.
- **Desarrollo de un sistema automático de optimización de código en DSPs para los algoritmos implementados anteriormente**. Para complementar la optimización manual, se ha desarrollado una optimización automática. Así pues, combinando ambas técnicas, se ha obtenido un código muy eficiente.

1.3 Estructura de la tesis

Esta memoria describe el trabajo realizado, y se ha estructurado en los siguientes capítulos y apéndices:

Capítulo 1. Introducción, objetivos y estructura de la tesis. En este capítulo se presenta una breve introducción de la separación ciega de señales y se presentan los principales objetivos del presente trabajo y las motivaciones del desarrollo, así como las principales aportaciones que se han realizado. También se presenta la estructura de esta memoria, resumiendo brevemente los contenidos de cada capítulo.

Capítulo 2. Procesamiento digital de señales. En este capítulo se realiza una introducción general al Procesado Digital de Señales (PDS), así como cuestiones relacionadas con la conversión A/D, D/A y las soluciones empleadas.

Capítulo 3. Arquitecturas basadas en DSPs. En este capítulo se describen los Procesadores Digitales de Señales (DSP), sus características y arquitectura interna. A continuación se muestra el DSP empleado, la plataforma de desarrollo y las técnicas actuales de optimización de código para dicha arquitectura.

Capítulo 4. **Separación ciega de fuentes.** En este capítulo se muestran nociones de la separación ciega de fuentes, y su resolución mediante ICA. Finalmente se describe el algoritmo FastICA que se utilizará posteriormente en este trabajo.

Capítulo 5. Nuevos algoritmos de separación en medios no estacionarios. En este capítulo se presentan algunos algoritmos utilizados para la separación ciega de fuentes cuando la matriz de mezcla cambia en el tiempo. Inicialmente se plantea el estado actual mostrando algoritmos que se encuentran en la bibliografía y posteriormente se presentan dos nuevos algoritmos propuestos en esta tesis: el ASWICA y ADSWICA basados en ventanas deslizantes con FastICA.

Capítulo 6. Implementación de los nuevos algoritmos en DSP. En este capítulo se presentan los resultados del análisis, implementación y optimización de los algoritmos ASWICA y ADSWICA en DSP de altas prestaciones. También se presenta un sistema automático de optimización de código en DSPs para los algoritmos implementados anteriormente.

Capítulo 7. Conclusiones. En este capítulo se recogen las conclusiones y principales aportaciones de este trabajo. Finalmente se proponen ciertas mejoras así como las líneas de investigación de interés que abre, para desarrollar en un futuro.

Apéndice A. Incluye documentación de la plataforma de desarrollo de Texas Instruments y el DSP TMS320C6416.

Apéndice B. Muestra información de la optimización de algunas partes de código que ha sido evaluados.

Resumen:

En este capítulo se realiza una introducción general al Procesado Digital de Señales (PDS), así como cuestiones relacionadas con la conversión A/D, D/A y las soluciones empleadas.

2 PROCESAMIENTO DIGITAL DE SEÑALES

2.1 Estadística, probabilidad y ruido

La estadística y la probabilidad son utilizadas en el PDS para caracterizar las señales y los procesos que las generan. Por ejemplo, reducir las interferencias, ruido y otros elementos indeseables en los datos adquiridos. Éstos pueden ser partes inherentes a la señal que está siendo medida, surgir por las tolerancias de los sistemas de adquisición o ser introducidas como subproductos inevitables de algunas operaciones del PDS. La estadística y la probabilidad permiten la medición y clasificación de estas características que distorsionan o modifican las señales.

El primer paso, para el correcto procesado digital de señales, ha de producirse en el sentido de eliminar estas componentes no deseadas. Es por ello que en este capítulo se introducen los conceptos más importantes en estadística y probabilidad, haciendo énfasis en la forma de ser aplicados.

2.1.1 Terminología de señales y gráficos

Una señal es una descripción de cómo un parámetro esta relacionado con otro parámetro (por ejemplo, tensión que varía en función del tiempo). Normalmente serán

señales en las que ambos parámetros puedan tomar un rango variable de valores, o sea, señales continuas. Por otra parte, al pasar esta señal a través de conversor analógico-digital, ambos parámetros son cuantificados. Las señales formadas de parámetros que son cuantificados de esta forma se denominan señales discretas o señales digitales. También se puede dar el caso en que un parámetro sea continuo y el otro sea discreto. La Figura 2.1 [SMI97] muestra dos señales discretas, tales como las que pueden ser adquiridas con un sistema de adquisición de datos digital. En el eje de ordenadas se muestra la amplitud de la señal, *eje y*, variable dependiente, rango u ordenada.

El eje de abscisas se representa el otro parámetro de la señal, y se le conoce como: *eje x*, variable independiente, dominio o abscisa. El tiempo es el parámetro más común que aparece en el eje horizontal de las señales adquiridas, aunque podría ser otro.

En PDS, el dominio será normalmente en el tiempo, en la frecuencia, o en el espacial. Cuando se trata de muestras, a estas señales se las consideran en el dominio en el tiempo. Esto es porque el muestreo a intervalos iguales de tiempo es la forma más común de obtener señales de este tipo.

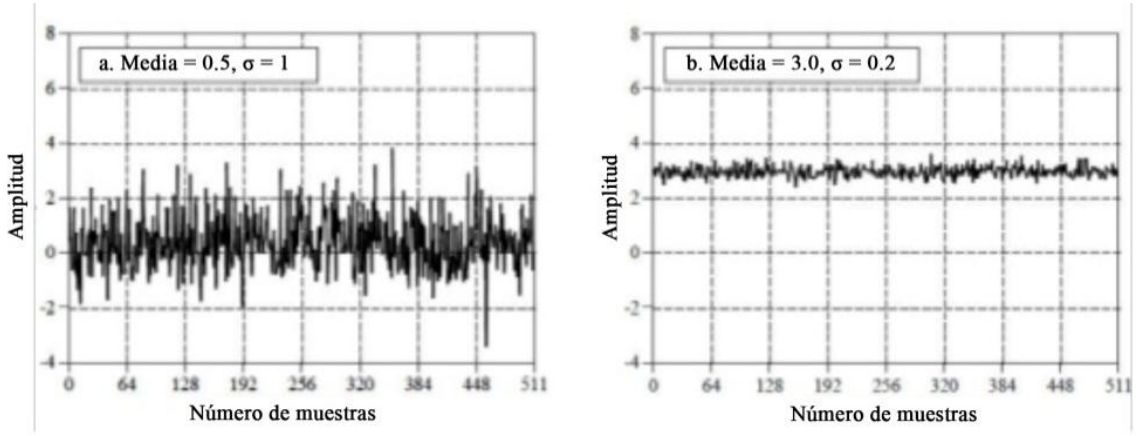


Figura 2.1 Ejemplos de dos señales digitalizadas con diferentes media y desviación estándar.

Aunque las señales de la Figura 2.1 [SMI97] son discretas se representan con líneas continuas (que normalmente se haría con línea discontinua) debido al elevado número de muestras. Por lo tanto, cuando se observen señales, se debe prestar atención a la leyenda del eje horizontal para saber si se está trabajando con una señal continua o discreta.

La variable *N*, es la que normalmente se emplea en el PDS para representar el número total de muestras de una señal. Por ejemplo, *N=512* en la señal de la Figura 2.1. Cada

muestra es nombrada mediante su número de muestra o un índice. Normalmente se suelen utilizar dos tipos de notaciones para asignar números a las muestras. Los matemáticos emplean más la que va de *1* a *N*, mientras, en el PDS es más común hacerlo de *0* a *N-1*, siendo esta última la que se empleará en el presente documento.

2.1.2 Desviación media y estándar

La **media, μ**, es el valor promedio de una señal. Se obtiene como podría esperarse: sumando todas las muestras y dividiendo entre el número total de ellas (*N*):

$$\boldsymbol{\mu = \frac{1}{N}\sum_{i=0}^{N-1} x_i} \qquad (2.1)$$

La ecuación (2.1) muestra el cálculo de la media de una señal. Las muestras de la señal están entre x_0 y x_{n-1}, donde i representa el valor índice de la muestra en curso de la señal, *x*, siendo μ la media. O sea, es la suma de valores en la señal x_i, haciendo al índice, *i*, desplazarse del *0* al *N-1*, terminando el cálculo con la división de la suma entre *N*.

En electrónica, la media es comúnmente denominada valor DC (corriente continua). Del mismo modo, AC (corriente alterna) se refiere a como la señal fluctúa sobre su valor medio. Si la señal es periódica, como lo es la señal seno. Pudiendo ser descrita su trayectoria por su amplitud pico-pico. Aunque por lo general no son periódicas. Un método más generalizado tiene que ser empleado en estos casos, éste es la **desviación estándar, σ**.

La expresión, $|xi-\mu|$, indica en que medida se ha desviado la i-ésima muestra de la media. La **desviación promedio** de una señal es obtenida sumando las desviaciones de todas las muestras individuales y, dividiendo entre el número de muestras, N. Hay que tener en cuenta que se toma el valor absoluto de cada desviación antes de la suma; por otro lado, los términos positivos y negativos se promedian a cero. La desviación promedio proporciona un único valor que representa la distancia típica a la que las muestras se separan de la media. La desviación promedio a pesar de ser conveniente y sencilla, casi nunca se emplea en estadística. Esto se debe a que no se corresponde con la física del comportamiento de las señales. En la mayor parte de los casos, el parámetro importante no es la desviación desde la media, sino la potencia representada por esta desviación de la media. Por ejemplo, cuando señales de ruido aleatorias se combinan en un circuito

electrónico, el ruido resultante es igual a la combinación de las potencias de las señales individuales, no de sus amplitudes combinadas.

La **desviación estándar,** $\boldsymbol{\sigma}$, es similar a la desviación promedio, excepto en que el promedio es dado con la potencia frente a la amplitud. Esto se consigue haciendo el cuadrado de cada una de las desviaciones antes de tomar el promedio. Aunque es más utilizado, en estadística, el término **varianza,** $\boldsymbol{\sigma^2}$.

$$\boldsymbol{\sigma^2 = \frac{1}{N-1}\sum_{i=0}^{N-1}(x_i - \mu)^2} \qquad (2.2)$$

La desviación estándar es una medida de la amplitud de la fluctuación de la señal respecto a la media, mientras que la varianza representa la potencia de esta fluctuación.

Otro término muy empleado en electrónica es el **valor rms** (raíz-media-cuadrado). Por definición, la desviación estándar sólo mide la parte AC de la señal, mientras que el valor rms mide ambas componentes, AC y DC. En caso de una señal sin componente DC, su valor rms será igual al de su desviación estándar. La Figura 2.2 [SMI97] muestra la relación entre la desviación estándar y el valor rms para varias formas de onda comunes.

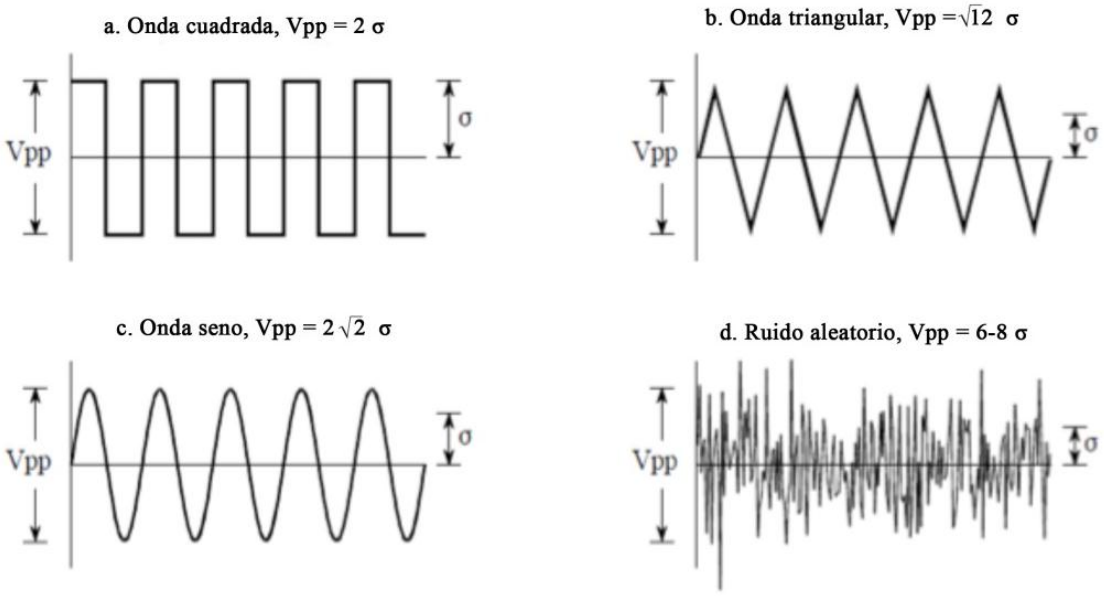


Figura 2.2 Relación entre la amplitud rms y la desviación estándar para varias formas de onda comunes.

Como se puede observar en la Figura 2.2, para la señal cuadrada, esta razón es de *2*; para la onda triangular es de *√12 = 2√3 = 3,46*; para la onda seno es de *2√2 = 2,83*. Mientras el ruido aleatorio no tiene un valor rms exacto, siendo éste aproximadamente entre 6 y 8 veces la desviación estándar.

Este método de cálculo de la media y de la desviación estándar es adecuado en la mayoría de los casos; sin embargo, tiene dos limitaciones. La primera es cuando la media es mucho mayor que la desviación estándar, ya que la ecuación (2.2) extraería dos números muy cercanos en valor, pudiendo provocar un excesivo error de redondeo en los cálculos. La segunda es ideal para recalcular la media y la desviación estándar en la adquisición de nuevas muestras y añadirlas a la señal, pudiendo denominarse estadística en ejecución. Mientras el método de las ecuaciones (2.1) y (2.2) puede ser empleado para la estadística en ejecución, éste requiere que todas las muestras sean incluidas en cada nuevo cálculo, debido a su potencia de cálculo y la memoria necesaria para ello.

A partir de las ecuaciones (2.1) y (2.2) se puede obtener la ecuación siguiente para el cálculo de la desviación estándar:

$$\boldsymbol{\sigma^2 = \frac{1}{N-1}\left[\sum_{i=0}^{N-1} x_i^2 - \frac{1}{N}\left(\sum_{i=0}^{N-1} x_i\right)^2\right]} \tag{2.3}$$

Usando notación simple o pseudocódigo:

$$\boldsymbol{\sigma^2 = \frac{1}{N-1}\left[\mathrm{sum}x_i^2 - \frac{(sum\ x_i)^2}{\mathrm{N}}\right]} \tag{2.4}$$

La ecuación (2.3) muestra el cálculo de la desviación estándar usando la estadística en ejecución. Esta ecuación proporciona el mismo resultado que la ecuación (2.2), pero con menos error de redondeo y una mayor eficiencia computacional. La señal es expresada en función de tres parámetros acumulados: N, número total de muestras; sumatoria de las muestras; y sumatoria de los cuadrados de las muestras. La media y la desviación estándar se calculan con estos tres parámetros acumulados.

Mientras las muestras van procesándose, se van almacenando los tres parámetros:

1. El número de muestras ya procesadas.
2. La sumatoria de dicha muestras.

3. La sumatoria del cuadrado de las muestras (hacer el cuadrado de cada muestra y sumarlo a un valor acumulativo).

Tras un cierto número de muestras procesadas, la media y la desviación estándar puede ser calculada eficientemente usando sólo el valor en curso de los tres parámetros.

En algunos casos, la media está siendo medida mientras que la desviación estándar representa ruido y otras interferencias. En estos casos, la desviación estándar no es importante en sí misma, sino sólo en comparación con la media. Esto proporciona el crecimiento del término: **relación señal-ruido (SNR)**, la cual es igual a la media dividida entre la desviación estándar.

Otro término interesante es el **coeficiente de variación (CV),** el cual es definido como la desviación estándar dividida entre la media, multiplicada por 100 para obtener el porcentaje.

2.1.3 Señal y procesos subyacentes

La estadística es la ciencia que interpreta los datos numéricos. Y a fin de cuentas, las señales adquiridas lo son. Por lo tanto, el PDS utiliza la probabilidad, como herramienta esencial, en el proceso de generación y tratamiento de señales.

Un ejemplo típico de generación de una señal es lanzando una moneda 1000 veces. Cuando la moneda cae de cara el valor que le corresponde a la muestra es 1, en caso contrario es 0. Esta generación tiene una media exacta, 0.5; determinada por la probabilidad relativa de que ocurra cada posible valor: 50% cara, 50% cruz. No obstante, es muy poco probable que la media exactamente en 0,5, ya que seguramente el número de unos y ceros sea ligeramente diferente. Es más, si se genera esta señal varias veces, por probabilidad estadística, el número de unos y ceros, probablemente cambiará. Esta irregularidad aleatoria se denomina variación estadística, fluctuación estadística o ruido estadístico.

Como previamente se ha mencionado, la ecuación (2.2) divide entre *N-1* al cálculo de la media de los cuadrados de las desviaciones, y no simplemente entre *N*. Para comprenderlo se considera que se desea encontrar la media y la desviación estándar de algunos procesos que generan señales. Para lograr esto, se toma una señal de *N* muestras del proceso y se calcula la media usando la ecuación (2.1). Se puede usar como una

estimación de la media del proceso subyacente; sin embargo, se va a tener un error debido al ruido estadístico. Para señales aleatorias, el error típico entre la media de los *N* puntos y la media del proceso es:

$$\boldsymbol{Error\ típico} = \frac{\sigma}{N^{\frac{1}{2}}} \tag{2.5}$$

La ecuación (2.5) muestra el **error típico** en el cálculo de la media de un proceso subyacente empleando un número finito de muestras, *N*.

Si *N* es pequeño, el ruido estadístico en la media calculada será grande. Aumentando el valor de *N*, se hace menor la expectativa de error.

Antes de calcular la desviación estándar usando la ecuación (2.2), es necesario conocer previamente la media, μ. Sin embargo, no se conoce la media del proceso generador, sino sólo la media de la señal de *N* puntos, la cual contiene un error debido al ruido estadístico. Este error tiende a reducir el valor calculado de la desviación estándar. Para compensar este hecho, *N* es sustituido por *N-1*. Si *N* es grande, la diferencia no es significativa, pero si es pequeño, proporciona una estimación más exacta del proceso. La ecuación (2.2) es una estimación de la desviación estándar del proceso generador y si se divide entre *N* en la ecuación, se obtiene la desviación estándar de la señal adquirida.

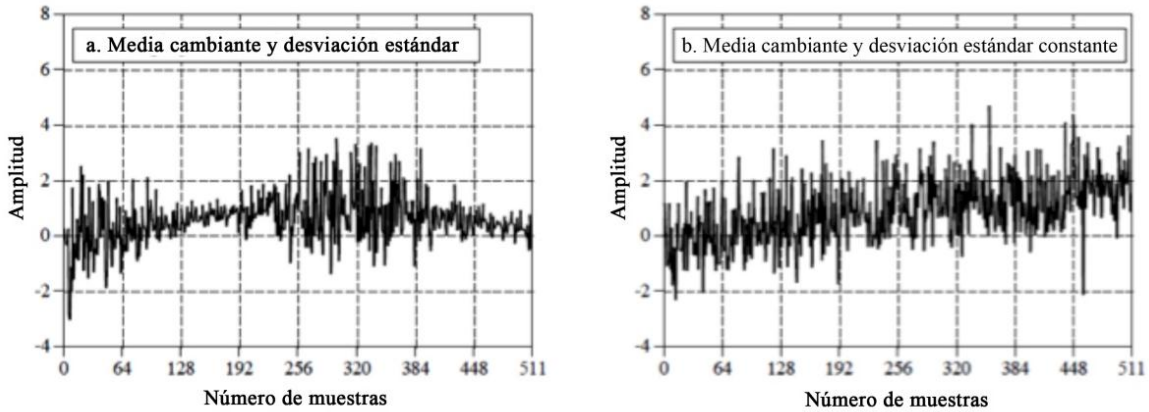


Figura 2.3 Ejemplos de señales generadas desde procesos no estacionarios.

La Figura 2.3 [SMI97] muestra en (a) cuando la media y la desviación estándar cambian, y en (b) cuando la desviación estándar mantiene un valor constante de uno, mientras la media cambia desde el valor de cero a dos. Esta es una técnica de análisis

común dividiendo estas señales en pequeños segmentos y, calculando la estadística de cada segmento individualmente.

En la Figura 2.3 se observa que estos cambios son demasiado grandes como para tratarse de un cambio aleatorio y, debería tratarse del proceso de generación. Los procesos que cambian sus características de esta forma se denominan no estacionarios. En comparación, las señales previamente presentadas en la Figura 2.1, que estaban generadas por procesos estacionarios y las variaciones son totalmente resultado del ruido estadístico. La Figura 2.3 (b) muestra un problema común para las señales no estacionarias: el lento cambio de la media interfiere con el cálculo de la desviación estándar. En este ejemplo, la desviación estándar de la señal, para un pequeño intervalo, es uno, mientras que la desviación estándar de la señal completa es 1,16. Este error puede casi eliminarse mediante la ruptura de la señal en pequeñas secciones y, calculando la estadística para cada sección individualmente. Si es necesario, las desviaciones estándar de cada una de las secciones pueden ser promediadas para producir un valor único.

2.1.4 Histograma, PMF y PDF

Si se considera que se une un convertidor analógico-digital de 8 bits a una computadora y, se adquieren 256.000 muestras de una señal. La Figura 2.4 [SMI97] muestra, en (a), 128 muestras que pueden ser parte de este conjunto de datos. El valor de cada muestra será uno de los 256 posibles, de 0 a 255. El histograma representa el número de muestras que hay en la señal que tiene cada uno de esos posibles valores. En (b) muestra el histograma para las 128 muestras en (a). Por ejemplo, hay 2 muestras que tienen un valor de 110, 7 con un valor de 131, 0 con un valor de 170, etc. Se representa el **histograma** por $\boldsymbol{H_i}$, donde *i* es un índice que va de 0 a *M-1*, y *M* es el número de posibles valores que cada muestra puede tomar. Por lo tanto, H_{50} es el número de muestras que toman el valor de 50. En (c) muestra el histograma la señal empleando el conjunto completo de datos, 256.000 puntos. Como se observa, el gran número de muestras resulta tener una apariencia más suave. Al igual que la media, el ruido estadístico del histograma es inversamente proporcional a la raíz cuadrada del número de muestras empleadas.

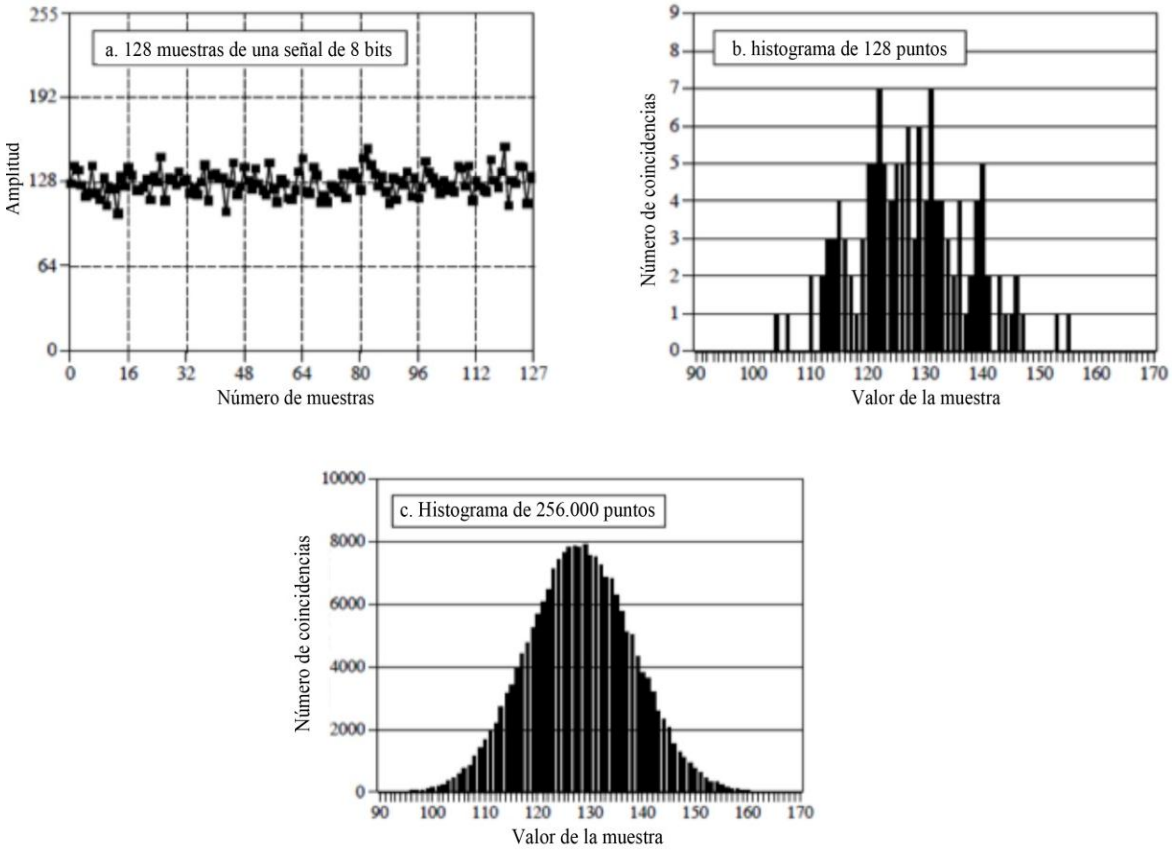


Figura 2.4 Ejemplos de histogramas. (a) 128 muestras de una señal muy grande, con cada muestra siendo un entero entre 0 y 255; (b) y (c) histogramas usando 128 y 256.000 muestras de la señal, respectivamente.

Como se muestra en la Figura 2.4 [SMI97], el histograma es más suave conforme más muestras son usadas. La suma de todos los valores en el histograma debería ser igual al número de puntos en la señal:

$$\boldsymbol{N = \sum_{i=0}^{M-1} H_i} \qquad (2.6)$$

La ecuación (2.6) muestra la sumatoria de todos los valores en el histograma es igual al número de puntos en la señal. Siendo H_i el histograma, N es el número de puntos en la señal y, M el número de puntos en el histograma.

El histograma puede ser usado eficientemente para calcular la media y la desviación estándar de conjuntos de un gran número de datos. Esto es especialmente importante para imágenes, las cuales pueden contener millones de muestras. El histograma agrupa muestras de igual valor, esto permite que las estadísticas sean calculadas con más facilidad mediante

el trabajo con pequeños grupos que usando un número grande de muestras individuales. Con esta aproximación, la media y la desviación estándar son calculadas.

$$\boldsymbol{\mu = \frac{1}{N}\sum_{i=0}^{M-1} i \cdot H_i} \tag{2.7}$$

La ecuación (2.7) muestra el cálculo de la media desde el histograma. Puede ser vista como una combinación de todas las muestras teniendo el mismo valor dentro de los grupos y, entonces usando la ecuación (2.1) para cada grupo:

$$\boldsymbol{\sigma^2 = \frac{1}{N-1}\sum_{i=0}^{M-1}(i-\mu)^2 H_i} \tag{2.8}$$

La ecuación (2.8) muestra el cálculo de la desviación estándar desde el histograma. Este es el mismo concepto que en la ecuación (2.2), excepto que todas las muestras que tienen el mismo valor son operadas a la vez.

La señal adquirida es una versión ruidosa del proceso subyacente, por lo que a algunos de los conceptos le son dados nombres diferentes. El histograma se forma a partir de la señal adquirida. La curva correspondiente para el proceso subyacente es llamada **función de la probabilidad de masa (PMF**). Un histograma es siempre calculado usando un número finito de muestras, mientras el PMF podría ser obtenido con un infinito número de muestras. El PMF puede ser estimado desde el histograma o, ser deducido mediante una técnica matemática (caso del lanzamiento de la moneda).

La Figura 2.5 [SMI97] muestra un ejemplo de PMF y, uno de los posibles histogramas que pueden estar asociados a él. Hay que tener muy en cuenta el eje vertical, o sea, del número de veces que un valor en particular se da en una señal. El eje vertical del PMF contiene información similar, excepto que es expresado en un número fraccional de muestras para aproximar el PMF, o sea, cada valor del histograma se divide entre el número total muestras para aproximar el PMF. Esto significa que cada valor en el PMF debería encontrarse entre cero y uno y, que la suma de todos los valores en el PMF será igual a uno.

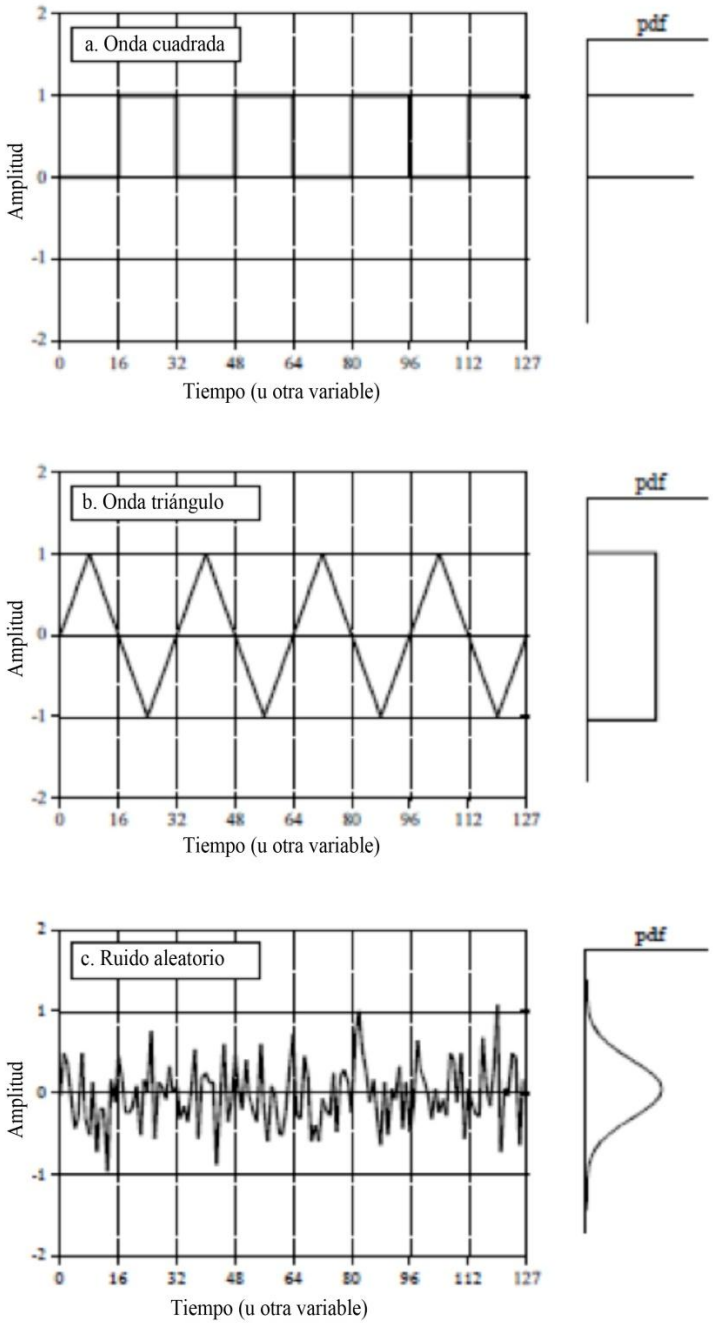


Figura 2.5 Tres formas de onda comunes y sus funciones de densidad de probabilidad.

El PMF describe la probabilidad de que un cierto valor será generado. El histograma y el PMF sólo pueden ser usados con datos discretos, tales como los de las señales digitalizadas residentes en una computadora. Un concepto similar aplicado a señales continuas, tal como la tensión en electrónica analógica. La **función de densidad de**

probabilidad (PDF), también denominada función de distribución de la probabilidad, es para señales continuas como la PMF para las señales discretas.

El histograma, PMF y, PDF son conceptos muy similares. La Figura 2.5 muestra tres formas de onda continuas y sus PDF. Si éstas fueran señales discretas, habría entonces que cambiar la etiqueta del eje x a "número de muestras", observándose en ese caso los PMF.

Aparecen problemas en el cálculo del histograma cuando el número de valores que cada muestra puede tomar es mayor que el número de muestras de la señal. Esto es siempre así para señales representadas en punto flotante, en el que cada muestra es almacenada como un valor fraccional. Por ejemplo, la representación de enteros puede requerir que el valor de la muestra esté entre 3 y 4, mientras que en punto flotante permite millones de posibles niveles fraccionales entre 3 y 4. Esto no es posible con datos en punto flotante porque billones de posibles niveles deberían ser tenidos en cuenta. Aún peor, casi todos estos niveles no tendrían muestras que se le correspondieran. Por ejemplo, imaginar una señal de 10.000 muestras, en la que cada muestra tiene un billón de posibles valores. El histograma convencional podría consistir en un billón de puntos de datos, alrededor de 10.000 tendrán un valor de cero.

La solución para estos problemas es la **técnica de binning**, la cual selecciona arbitrariamente la longitud del histograma para que tenga ciertos valores convenientes, como 1.000 puntos, denominados normalmente **bins**. El valor de cada punto representa el número total de muestras en la señal que tiene un valor dentro de cierto rango. Por ejemplo, si se tiene una señal en punto flotante que contiene valores entre 0,0 y 10,0 y, un histograma con 1.000 bins. El bin 0 en el histograma es el número de muestras en la señal con un valor entre 0 y 0,01, el bin 1 es el número de muestras entre 0,01 y 0,02, y así hasta el bin 999 que contiene un número de muestras con un valor entre 9,99 y 10,0.

El gráfico del PDF se gira normalmente un cuarto y situado en la cara de la que describe la señal. El PDF de una señal cuadrada, mostrada en (a), consiste en dos picos infinitesimalmente estrechos, correspondiendo a la señal que sólo tiene dos posibles valores. El PDF de la señal triángulo, (b), tiene un valor constante sobre el rango y, se denomina normalmente distribución uniforme. El PDF del ruido aleatorio, como en (c), es el más interesante, teniendo forma de curva en forma de campana de gauss o gaussiana.

Como se muestra en la Figura 2.5(a), la señal usada en este ejemplo tiene una longitud de 300 muestras, con cada muestra en una señal en punto flotante uniformemente

distribuida entre 1 y 3. Las Figura 2.5(b) y (c) muestran histogramas de binning de esta señal, usando 601 y 9 bins, respectivamente. Como se muestra, un gran número de bins resulta una resolución deficiente a lo largo del eje vertical, mientras un pequeño número de bins proporciona una pobre resolución a lo largo del eje horizontal. Usando más muestras se mejora la resolución en ambas direcciones.

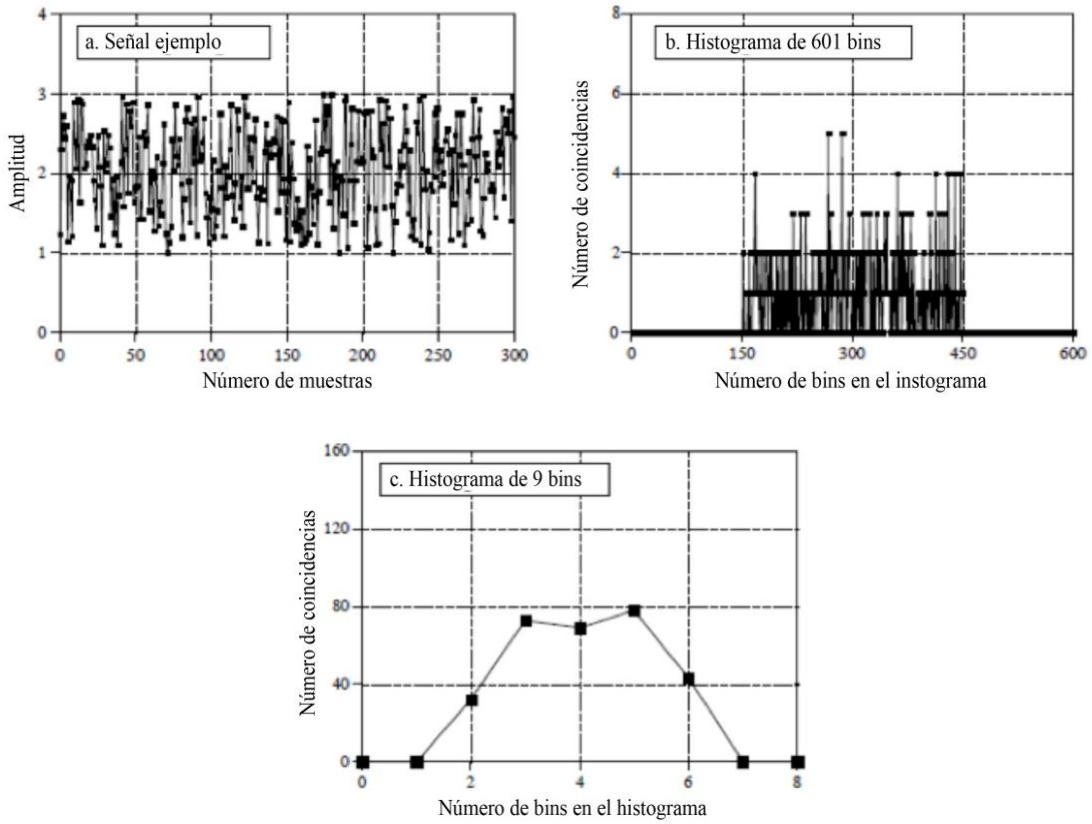


Figura 2.6 Ejemplo de histograma con binning

La forma de usar los bins conlleva un compromiso entre dos problemas (Figura 2.6 [SMI97]), muchos bins tienen dificultad para ser estimada su amplitud del PMF subyacente. Esto es porque unas pocas muestras caen dentro de cada cubo, haciendo que el ruido estadístico sea muy alto. En el otro extremo, muy pocos bins hacen difícil la estimación del PMF subyacente o en la dirección horizontal. En otras palabras, el número de bins controla la relación entre las resoluciones a lo largo del eje x y del eje y.

2.1.5 Distribución normal

Las señales obtenidas de un proceso aleatorio normalmente tienen un PDF con forma de campana, denominada distribución normal, distribución de Gauss o Gaussiana, en honor al gran matemático alemán, Kart Friedrich Gauss (1777-1855). La expresión matemática de la forma básica de esta curva es:

$$\boldsymbol{y(x) = e^{-x^2}} \tag{2.9}$$

Esta curva inicial puede ser convertida en una completa gaussiana añadiendo una media ajustable, μ y una desviación estándar, σ. en. La ecuación debe de ser normalizada para que el área total bajo la curva sea 1, un requerimiento de todas las funciones de distribución de probabilidad. Resultando, en la forma general de las distribuciones normales, una de las relaciones más importantes en estadística y probabilidad:

$$\boldsymbol{P(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}} \tag{2.10}$$

La ecuación (2.10) muestra la ecuación para la distribución normal, distribución de Gauss o, simplemente gaussiana. En esta relación, *P(x)* es la función de distribución de probabilidad, μ, es la media y σ la desviación estándar.

La Figura 2.7 [SMI97] muestra varios ejemplos de curvas gaussianas con varias medias y desviaciones estándar. La media centra la curva sobre un valor particular, mientras que la desviación estándar controla el ancho de la campana.

Una característica interesante de la gaussiana es que la curva desciende a cero muy rápidamente, mucho más rápido que otras funciones comunes tales como caídas exponenciales o *1/x*. Por ejemplo para valores de dos, cuatro o seis de desviación estándar desde la media, el valor de la curva gaussiana ha caído hasta *1/19, 1/7563* y *1/166.666.666* respectivamente. En la práctica, la forma de caída brusca de la PDF de la gaussiana indica que estos extremos casi nunca ocurren, proporcionándole una forma de onda de apariencia redondeada con una amplitud pico-pico de unos *6-8*σ.

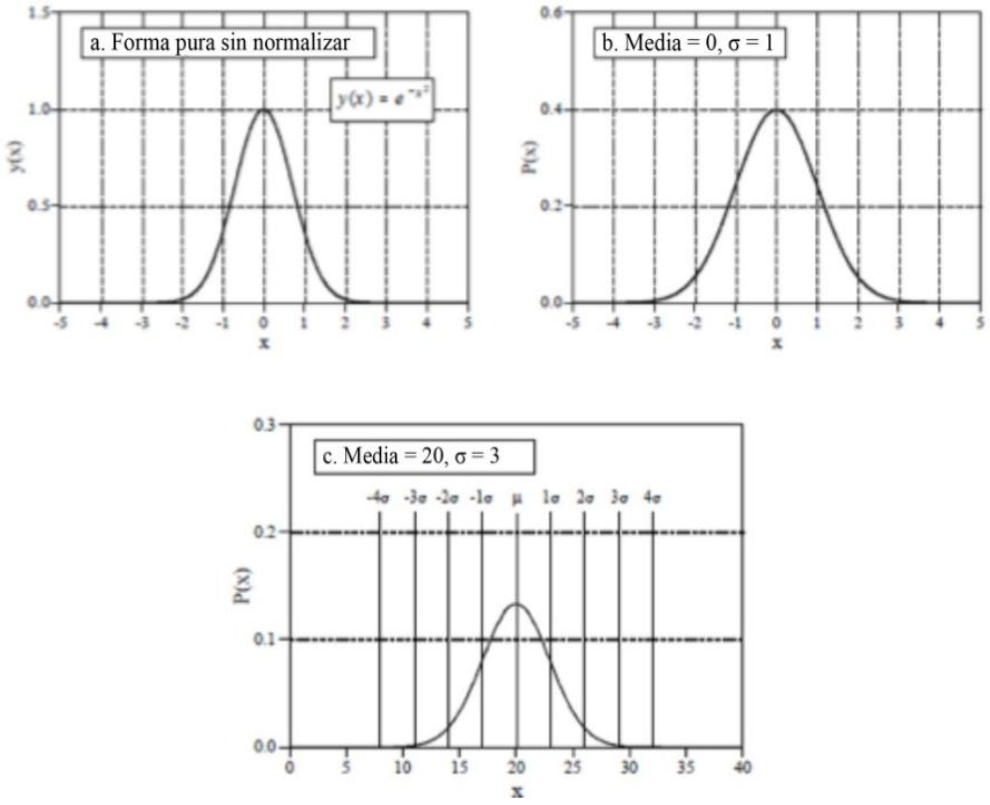
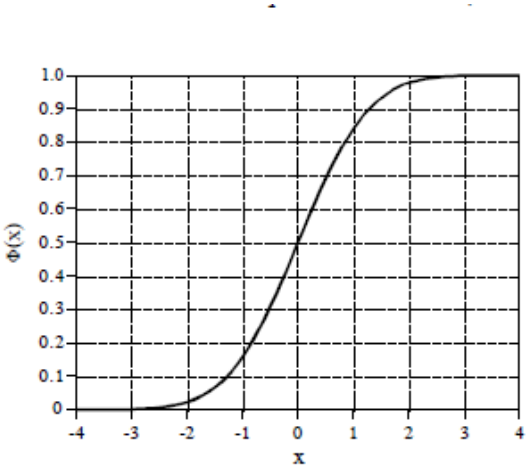


Figura 2.7 Ejemplos de curvas gaussianas: (a) forma de la curva incipiente sin normalización o la adición de parámetros ajustables; (b) y (c), curva gaussiana completa es mostrada para varias medias y desviaciones estándar.

La integral del PDF es empleada para encontrar la probabilidad de que la señal se encontrará dentro de un cierto rango de valores. Esto hace la integral del PDF tan importante como para recibir su propio nombre, **función de distribución acumulativa (CDF)**. Un problema especialmente indeseable con la gaussiana es que no puede ser integrada usando métodos elementales. Para lograrlo, la integral de la gaussiana puede ser calculada usando la integración numérica. Esto significa muestrear continuamente la gaussiana, es decir, un millón de puntos entre -10σ y $+10\sigma$. Las muestras en esta señal discreta son sumadas para simular integración. La curva discreta resultante de la integración simulada es entonces almacenada en una tabla para emplearla en el cálculo de probabilidades.



x	Φ(x)	x	Φ(x)
-3.4	.0003	0.0	.5000
-3.3	.0005	0.1	.5398
-3.2	.0007	0.2	.5793
-3.1	.0010	0.3	.6179
-3.0	.0013	0.4	.6554
-2.9	.0019	0.5	.6915
-2.8	.0026	0.6	.7257
-2.7	.0035	0.7	.7580
-2.6	.0047	0.8	.7881
-2.5	.0062	0.9	.8159
-2.4	.0082	1.0	.8413
-2.3	.0107	1.1	.8643
-2.2	.0139	1.2	.8849
-2.1	.0179	1.3	.9032
-2.0	.0228	1.4	.9192
-1.9	.0287	1.5	.9332
-1.8	.0359	1.6	.9452
-1.7	.0446	1.7	.9554
-1.6	.0548	1.8	.9641
-1.5	.0668	1.9	.9713
-1.4	.0808	2.0	.9772
-1.3	.0968	2.1	.9821
-1.2	.1151	2.2	.9861
-1.1	.1357	2.3	.9893
-1.0	.1587	2.4	.9918
-0.9	.1841	2.5	.9938
-0.8	.2119	2.6	.9953
-0.7	.2420	2.7	.9965
-0.6	.2743	2.8	.9974
-0.5	.3085	2.9	.9981
-0.4	.3446	3.0	.9987
-0.3	.3821	3.1	.9990
-0.2	.4207	3.2	.9993
-0.1	.4602	3.3	.9995
0.0	.5000	3.4	.9997

Figura 2.8 Φ(x), la función de distribución acumulativa de la distribución normal (media=0, desviación estándar=1).

El CDF de la distribución normal es mostrado en la Figura 2.8 [SMI97], con su valor numérico listado en la tabla que muestra. Debido a que esta curva es usada frecuentemente en probabilidad, se le ha dado su su propia simbología: *Φ(x).* Por ejemplo, *Φ (-2)* toma un valor de 0,0228. Esto indica que hay un 2,28% de posibilidades de que el valor de la señal esté entre menos infinito y dos desviaciones estándar por debajo de la media, en cualquier instante de tiempo. Del mismo modo, el valor *Φ(1)=0,8413*, significa que tiene un 84,13% de posibilidades de que el valor de la señal, en cualquier instante de tiempo, esté entre menos infinito y una desviación estándar por encima de la media.

Para calcular la probabilidad de que la señal esté entre dos valores, es necesario substraer los números apropiados encontrados en la tabla *Φ(x).* Por ejemplo, la probabilidad de que el valor de la señal, en cualquier instante de tiempo, esté entre dos desviaciones estándar por debajo de la media y una desviación estándar por encima de la media está dada por *Φ(1)- Φ(-2)= 0,8185* o sea 81,85%.

Usando este método, las muestras tomadas de una señal normalmente distribuida estarán entre $\pm 1\sigma$ de la media en aproximadamente un 68% del tiempo. Podría estar dentro de $\pm 2\sigma$ en un 95% del tiempo y, dentro de $\pm 3\sigma$ en un 99,75% del tiempo. La probabilidad de que la señal sea más de 10 desviaciones estándar de la media es ínfima, se puede esperar que ocurra solo en unos pocos microsegundos desde el inicio del universo, alrededor de 10 billones de años.

La ecuación (2.10) puede ser empleada para expresar la función de probabilidad de masa de señales discretas normalmente distribuidas. En este caso, x está limitada a ser uno de los niveles cuantizados que la señal puede tomar, tal como uno de los 4096 valores binarios existentes en un convertidor analógico-digital de 12 bits. Se ignora el término $1/\sqrt{2\pi\sigma}$, sólo usado para hacer el área total bajo la curva PDF igual a uno en su lugar, se puede incluir el término que sea necesario para hacer que la suma de todos los valores en el PMF sea igual a uno. En la mayoría de los casos, esto es dado por la generación de la curva sin preocuparse por la normalización, sumando todos los valores no normalizados y, entonces dividiendo todos los valores entre la suma.

Los valores de la Figura 2.8 [SMI97] (a) son calculados por integración numérica de la distribución normal mostrada en la (b). Siendo $\Phi(x)$ la probabilidad de que el valor de la señal normalmente distribuida, en cualquier instante de tiempo, sea menor que x. En esta tabla, el valor de x es expresado en unidades de desviación estándar referenciadas a la media.

2.1.6 Generación de ruido digital

El ruido aleatorio es un problema importante tanto en electrónica como en PDS. Por ejemplo, limita el tamaño mínimo de una señal a medir por un instrumento de medición, la distancia a la que un sistema de radio puede llegar o cuanta radiación se requiere para producir una imagen de rayos X. Una necesidad común en PDS es generar señales que simulen ruidos aleatorios. Esto es necesario para evaluar el rendimiento de algoritmos que deben trabajar en presencia de ruido.

La generación de una señal digital es realizada por un generador de números aleatorios. Cada número aleatorio tiene un valor entre cero y uno, con una probabilidad igual de estar en cualquier lugar entre estos dos extremos. La Figura 2.9(a) muestra 128 muestras

componentes de una señal obtenida a partir de un generador de números aleatorios de este tipo. La media del proceso subyacente que genera esta señal es 0.5, la desviación estándar es *1/√12=0,29* y, la distribución es uniforme entre cero y uno.

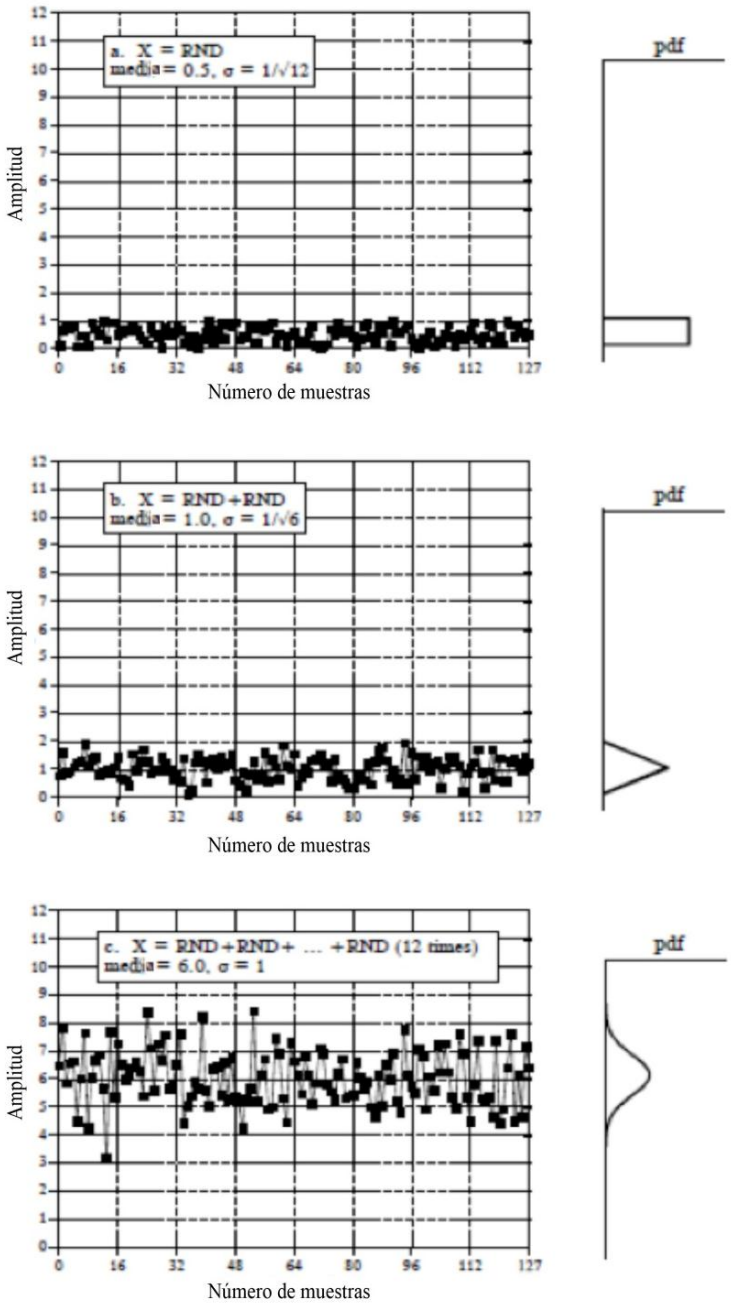


Figura 2.9 Conversión de una distribución uniforme en una distribución gaussiana. (a) señal donde cada muestra es generada por un generador de números aleatorios; (b) cada muestra está formada por la suma de dos valores procedentes del generador de números aleatorios; (c), cada muestra es creada sumando doce valores provenientes del generador de números aleatorios.

Es necesario crear ruido digital con un PDF gaussiano. Existen dos métodos para generar tales señales usando un generador de números aleatorios. La Figura 2.9 [SMI97] (a) ilustra el primer método. La Figura 2.9(b) muestra una señal obtenida mediante la adición de dos números aleatorios para formar cada muestra, esto es, $X=RND+RND$. Debido a que cada uno de los números aleatorios puede ir desde cero a uno, la suma puede ir de cero a dos. La media es ahora uno y, la desviación estándar es $1/\sqrt{6}$ (recordar cuando las señales aleatorias independientes son añadidas, también las varianzas son sumadas). Como se muestra, la PDF ha cambiado desde una distribución uniforme a una distribución triangular. Esto es, la señal consume la mayor parte del tiempo alrededor del valor de uno, con menos tiempo consumido cerca del cero o dos.

La Figura 2.9(c) muestra esta idea dando un paso más mediante la suma de doce números aleatorios para producir cada muestra. La media es ahora seis y, la desviación estándar es uno. Lo que es más importante, el PDF se ha convertido virtualmente en una gaussiana. Este procedimiento puede ser usado para producir una señal de ruido uniformemente distribuido con unas media y desviación estándar arbitrarias. Para cada muestra en la señal: (1) sumar doce números aleatorios, (2) extraer seis para hacer la media igual a cero, (3) multiplicar por la desviación estándar deseada y, (4) sumar la media deseada.

Las bases matemáticas para este algoritmo se encuentran en el Teorema del Límite Central, uno de los conceptos más importantes en probabilidad. En su forma más simple, el Teorema del Límite Central plantea que una suma de números aleatorios se convierte en normalmente distribuida conforme más números aleatorios son añadidos.

El Teorema del Límite Central no requiere que los números aleatorios individuales sean de una distribución particular o incluso que sean de la misma distribución. Proporciona las razones por las que las señales normalmente distribuidas se dan ampliamente en la naturaleza. Donde quiera que haya fuerzas aleatorias diferentes interactuando, la PDF resultante es una gaussiana.

En el segundo método para la generación de números aleatorios normalmente distribuidos, el generador de números aleatorios es llamado dos veces, para obtener R_1 y R_2. Un número aleatorio normalmente distribuido, X:

$$\boldsymbol{X = (-2\log R_1)^{\frac{1}{2}} cos(2\pi R_2)} \quad (2.11)$$

La ecuación (2.11) muestra la generación de números aleatorios normalmente distribuidos. R_1 y R_2 son números aleatorios con una distribución uniforme entre cero y uno. Este resulta en X siendo normalmente distribuida con una media de cero y, una desviación estándar de uno. El logaritmo es en base e y el coseno está en radianes.

Al igual que anteriormente, esta aproximación puede generar señales aleatorias normalmente distribuidas con una media arbitraria y una desviación estándar. Se logra tomando cada número generado mediante esta ecuación, multiplicándolo por la desviación estándar deseada y sumando la media deseada.

El generador de números aleatorios opera empezando con una semilla, un número entre cero y uno. Cuando el generador de números aleatorios es invocado, la semilla es pasada a través de un algoritmo fijo, resultando un número entre cero y uno. Este nuevo número es aportado como número aleatorio y, es almacenado internamente para ser usado como semilla la próxima vez que el generador de números aleatorios sea llamado. Su algoritmo podría ser:

$$\boldsymbol{R = (aS + b)\ modulo\ c} \quad (2.12)$$

La ecuación (2.12) muestra el algoritmo común para la generación de números aleatorios uniformemente distribuidos entre cero y uno. En este método, S es la semilla, *R* es el nuevo número aleatorio *y*, *a*, *b* y *c* son las constantes elegidas apropiadamente. En decir, la cantidad *aS+b* dividida entre c y el resto es tomado como *R*.

De esta forma, una secuencia continua de números aleatorios puede ser generada, empezando todos por la misma semilla. Esto permite correr un programa múltiples veces usando exactamente las mismas secuencias de números aleatorios. Una técnica común es usar el tiempo como semilla, proporcionando de esta manera una nueva secuencia cada vez que se ejecuta el programa.

Desde un punto de vista puramente matemático los números generados de esta forma no pueden ser absolutamente aleatorios ya que cada número es obtenido a partir de un

número previo. El término **pseudo-aleatorio** es muy usado para describir esta situación. Sin embargo, esto no es problema por el que preocuparse ya que las secuencias generadas son estadísticamente aleatorias en un alto grado. Es poco probable encontrarse con una situación en la que estos no sean adecuados.

2.1.7 Precisión y exactitud

Precisión y exactitud son términos empleados para describir sistemas y métodos que miden o estiman (Figura 2.10 [SMI97]). En todos estos casos, hay algún parámetro del cual se desea saber su valor, denominado valor verdadero o, verdad. El método empleado proporciona un valor medido, que se desea que sea tan cercano al valor verdadero tanto como sea posible. Precisión y exactitud son caminos para describir el error existente entre estos dos valores.

En una medición que tenga una buena exactitud, pero pobre precisión; el histograma es centrado sobre el valor verdadero, pero es muy ancho. Aunque las medidas son correctas como un grupo, cada lectura individual es una medición pobre del valor verdadero. A esta situación se le da el nombre de **poca repetitividad**; las mediciones tomadas no concuerdan bien. Se tienen resultados de poca precisión debido a errores aleatorios. Este es el nombre dado a los errores que cambian cada vez que la medición se repite. Promediando muchas mediciones siempre mejorará la precisión. En resumen, la **precisión** es una medida del ruido aleatorio.

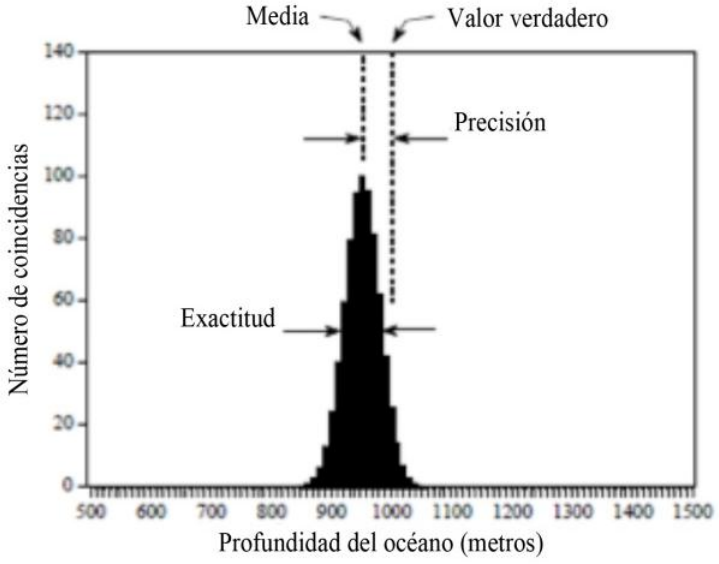


Figura 2.10 Definiciones de exactitud y precisión.

La **exactitud** es la diferencia entre el valor verdadero y la media del proceso subyacente que genera los datos. La Precisión es la dispersión de los valores, especificado por la desviación estándar, la relación señal-ruido, o el CV.

Si se toma una medida muy precisa, pero con poca exactitud esto hará que el histograma sea muy delgado, pero no centrado sobre el valor verdadero. Las lecturas sucesivas están cercanas en valor; sin embargo, todas ellas tienen un gran error y una pobre exactitud como resultado de errores sistemáticos. Estos son los errores que se repiten de la misma forma cada vez que se efectúa una medición. La exactitud normalmente depende de cómo esté calibrado el sistema. Por ejemplo, en la medida de la profundidad del océano, el parámetro directamente medido es el tiempo transcurrido. Empleándose un procedimiento de calibración que relaciona milisegundos con metros. Esto puede hacerse con una simple multiplicación por la velocidad fijada o puede ser tan complicado como correcciones del orden de decenas de segundos. Promediando mediciones individuales no se mejora la exactitud. La exactitud es una medida de la calibración.

2.2 Conversión AD y conversión DA

La mayoría de las señales encontradas directamente en la ciencia y la ingeniería son continuas. La conversión analógica-digital (CAD) y la conversión digital-analógica (CDA) son los procesos que permiten a las computadoras interactuar con estas señales de la vida cotidiana. La información digital difiere de la continua en dos aspectos importantes: se trata de una muestra, y está cuantificada. Ambos limitan la cantidad de información que una señal digital puede contener. Se debe establecer la comprensión de la información que se necesita conservar, y qué información permitirse perder. A su vez, esto exige la selección de la frecuencia de muestreo, número de bits, y el tipo de filtrado analógico necesario para la conversión de entre los ámbitos analógico y digital.

2.2.1 Conversión digital-analógica

En teoría, el método más simple de conversión digital a analógica es sacar las muestras de la memoria y convertirlas en un tren de impulsos.

La Figura 2.11 [SMI97], en (a) y (b) muestra una señal analógica integrada por componentes de frecuencia de entre cero y 0,33 de la frecuencia de muestreo, f_s. En (c), la señal analógica es muestreada convirtiéndola en un tren de impulsos. En el dominio de la frecuencia, (d), se observa en el espectro infinito de bandas laterales superiores e inferiores. Debido a que las frecuencias originales en (b) no están falseadas en (d), se realizado un muestreo correcto. En contraposición, la señal analógica en (e) es muestreada a 0,66 de la frecuencia de muestreo, un valor superior a la tasa de Nyquist. Esto da lugar a aliasing, indicado por las bandas laterales (f) superpuestas.

De (a), con el correspondiente espectro de frecuencias se obtiene (b). La señal analógica original, puede ser perfectamente reconstruida pasando un tren de impulsos a través de un filtro de paso bajo, con la frecuencia de corte igual a la mitad de la tasa de muestreo. En otras palabras, la señal original y el tren de impulsos tienen los mismos espectros de frecuencia por debajo de la frecuencia Nyquist. A frecuencias más altas, el tren de impulsos contiene una duplicación de la información, mientras que la señal analógica original, no contiene nada (suponiendo que no se produjo aliasing).

Si bien este método es matemática pura, es difícil generar los impulsos estrechos en la electrónica. Para lograr esto, casi todos los CDAs funcionan manteniendo el último valor hasta que se recibe otra muestra (retención de orden cero - zeroth-order hold), el equivalente CDA del sample-and-hold usado en CAD. La retención de orden cero produce la apariencia de escalera (c).

En el dominio de la frecuencia, retención de orden cero resulta en el espectro de un tren de impulsos multiplicado por la curva oscura mostrada en (d), dada por la ecuación:

$$\boldsymbol{H(f)} = \left| \frac{\boldsymbol{sin}\left(\frac{\boldsymbol{\pi f}}{\boldsymbol{f_s}}\right)}{\frac{\boldsymbol{\pi f}}{\boldsymbol{f_s}}} \right| \qquad (2.13)$$

La ecuación (2.13) muestra la reducción de la amplitud de alta frecuencia, debido al seno y la retención de orden cero. Esta expresión es de la forma general: *sin (Bx) / (Bx),*

llamada la función seno. La función *sin (x)* es muy común en PDS. La retención de orden cero puede ser entendida como la convolución entre un tren de impulsos y pulsos rectangulares, con un ancho igual al período de muestreo, resultando estar, la multiplicación por la transformada de Fourier del pulso rectangular, en el dominio de la frecuencia. En la Figura (d), la línea clara muestra el espectro de frecuencias del tren de impulsos, mientras que la línea oscura muestra el seno. El espectro de frecuencias de la señal de la retención de orden cero es igual al producto de estas dos curvas.

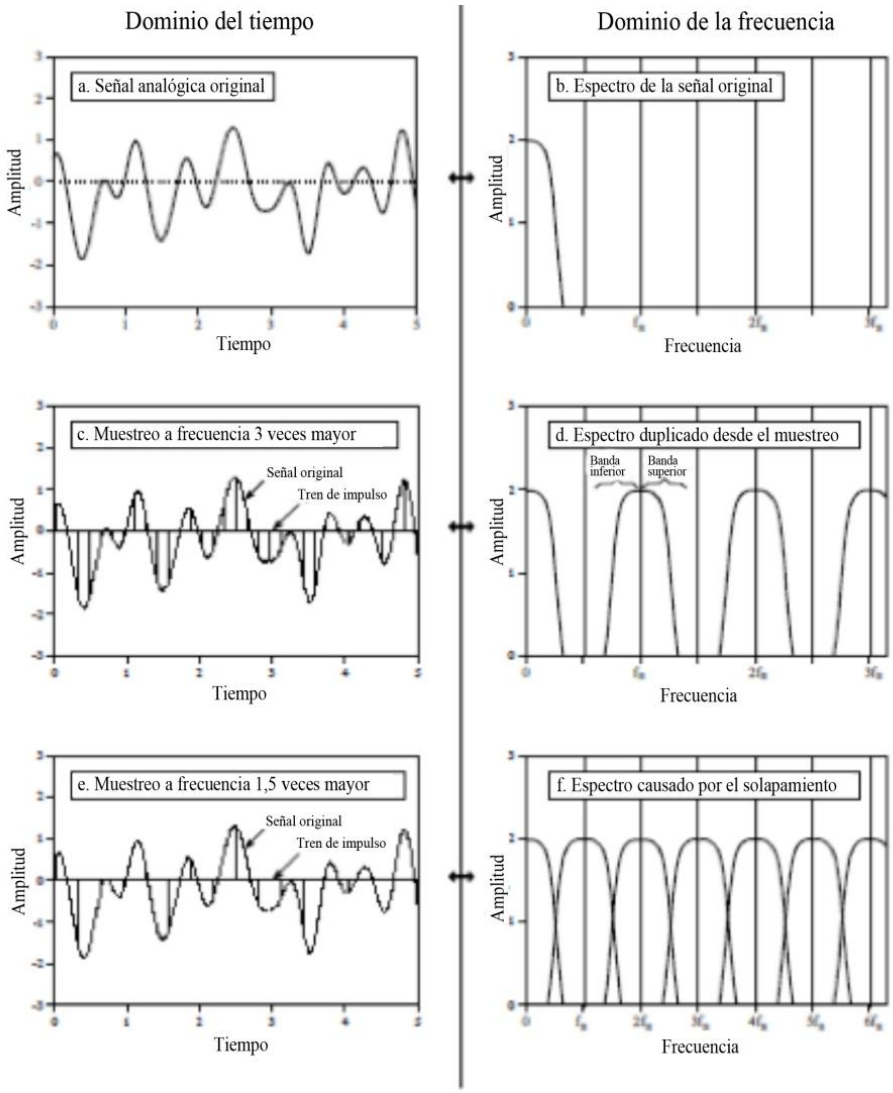


Figura 2.11 Teorema de muestreo en el dominio del tiempo y la frecuencia.

El filtro analógico utilizado para convertir la señal de la retención de orden cero, (c), en la señal reconstruida, (f), necesita hacer dos cosas: (1) eliminar todas las frecuencias por encima de la mitad de la tasa de muestreo, y (2) aumentar las frecuencias por el recíproco del efecto de la retención de orden cero, es decir, 1/sin (x). Esto equivale a una ampliación de cerca de 36% a la mitad de la frecuencia de muestreo. La figura (e) muestra la respuesta de frecuencia ideal de este filtro analógico.

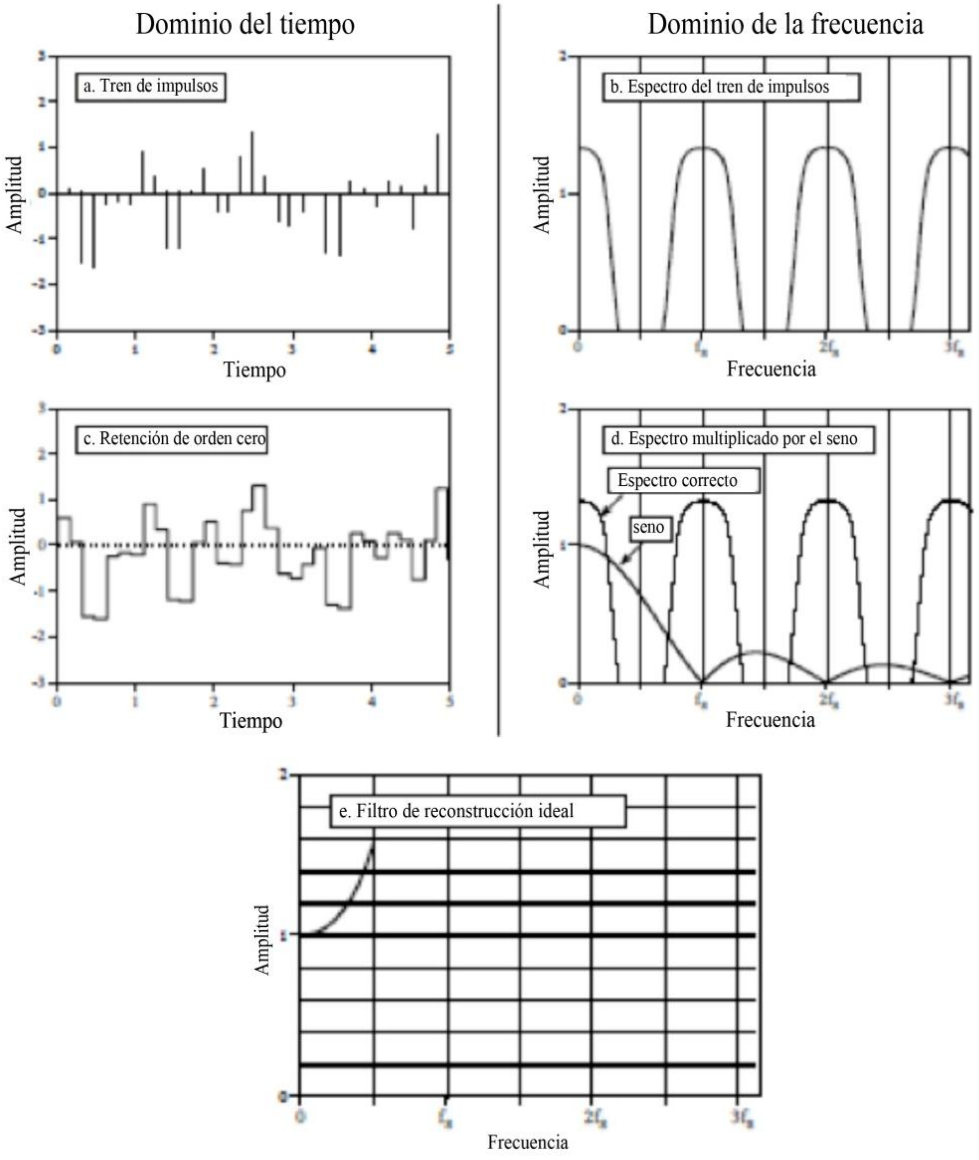
Este aumento de la frecuencia *1/sin (x)* se pueden manejar de cuatro maneras: (1) hacer caso omiso de él y aceptar las consecuencias, (2) diseño de un filtro analógico para incluir la respuesta *1/sin (x),* (3) usar una técnica multitasa o (4) hacer la corrección en el software antes de que el CDA.

En la Figura 2.12 [SMI97], en (a) se muestran los datos digitales se convierten en un tren de impulsos, con el espectro (b). Esto se transformó en la señal reconstruida, (f), mediante el uso de una electrónica de filtro de paso bajo para eliminar frecuencias por encima de la mitad de la tasa de muestreo. Sin embargo, la mayoría de los CDAs electrónicos crean una forma de onda entorno al cero (c), en lugar de un tren de impulsos. El espectro del entorno del orden cero es igual al tren de impulsos multiplicado por la función seno mostrada en (d). Para pasar de ese entorno de orden cero a la señal reconstruida, el filtro analógico debería eliminar todas las frecuencias por en cima del rango de Nyquist y, corregida por la seno, como se muestra en (c).

La cantidad de información transportada en una señal digital está limitada de dos maneras: en primer lugar, el número de bits por muestra limita la resolución de la variable dependiente, es decir, pequeños cambios en la amplitud de la señal puede perderse en el ruido de cuantización. En segundo lugar, la tasa de muestreo limita la resolución de la variable independiente, es decir, acontecimientos poco espaciados en la señal analógica se pueden perder entre las muestras, o lo que es lo mismo, las frecuencias por encima de la mitad de la tasa de muestreo se han perdido.

Dado que las señales analógicas usan parámetros continuos tienen infinitamente mejor resolución en ambas variables, las dependientes y las independientes. Las señales analógicas se ven limitadas por los mismos dos problemas que las señales digitales: el ruido y el ancho de banda (la frecuencia más alta permitida en la señal). El ruido en una señal analógica limita de la medición de la amplitud de la onda, al igual que hace el ruido de cuantización en una señal digital. Del mismo modo, la capacidad para separar los

eventos espaciados en una señal analógica depende de la frecuencia más alta permitida en la forma de onda.



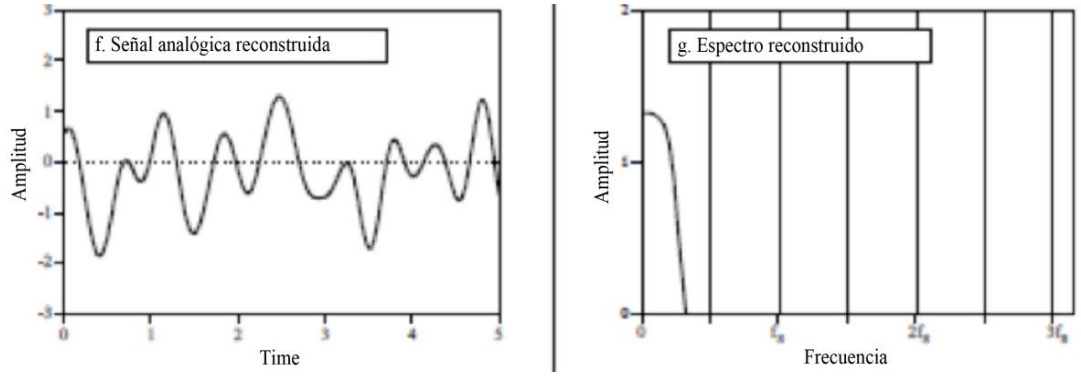


Figura 2.12 Análisis de la conversión digital a analógica.

2.2.2 Filtros analógicos para conversión de datos

La Figura 2.13 [SMI97] muestra un diagrama de bloques de un sistema de PDS, como indica el teorema de muestreo debe de estar antes del conversor analógico-digital.

En la Figura 2.13, se observa que el filtro electrónico colocado antes de un ADC es un **filtro antialias**. Se utiliza para eliminar los componentes de frecuencia por encima de la mitad de la tasa de muestreo. El filtro electrónico colocado después de un CDA se llama **filtro de reconstrucción** y también elimina las frecuencias por encima de la tasa de Nyquist, pudiendo incluir una corrección para la retención de orden cero.

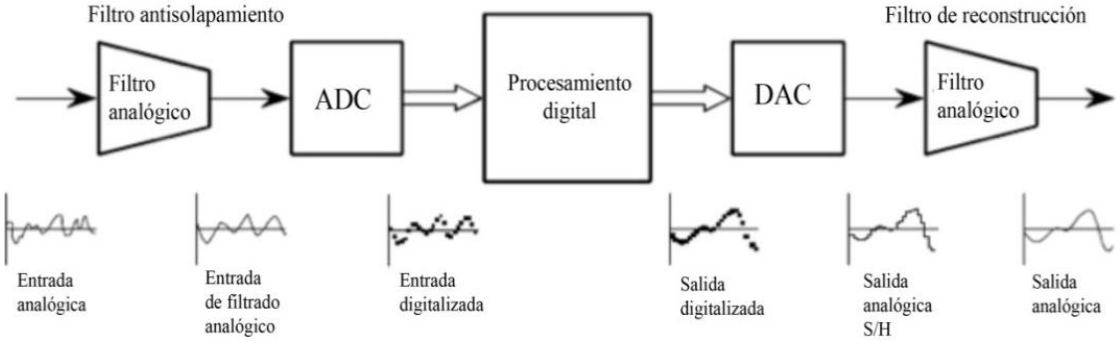


Figura 2.13 Filtros analógicos electrónicos utilizados para cumplir con el teorema de muestreo.

La señal de entrada se procesa con un filtro electrónico pasa bajo para suprimir todas las frecuencias por encima de la frecuencia Nyquist (mitad de la tasa de muestreo. En el otro extremo, la señal digitalizada se pasa a través de un conversor analógico digital y otro

filtro paso bajo ajustado a la frecuencia Nyquist. Este filtro de salida se denomina filtro de reconstrucción, y puede incluir el anteriormente descrito-cero para aumentar la frecuencia de retención. Las características de cada señal digitalizada dependerá de qué tipo de filtro antialias se utilizó cuando fue adquirido.

En la codificación en el dominio de la frecuencia, la información está contenida en ondas sinusoidales que se combinan para formar la señal. Las señales de audio son un excelente ejemplo de ello. Cuando una persona habla o escucha la música, la percepción del sonido depende de la frecuencia actual, y no en la forma particular de la forma de onda. Esto puede ser demostrado mediante la aprobación de una señal de audio a través de un circuito que cambia la fase de las diversas sinusoides, pero conserva su frecuencia y amplitud. La señal resultante se ve completamente diferente en un osciloscopio, pero los sonidos idénticos. La información pertinente se ha dejado intacta, a pesar de que la forma de onda ha sido alterada. Debido al solapamiento de los de componentes de frecuencia, se destruye directamente la información codificada en el dominio de la frecuencia. En consecuencia, la digitalización de estas señales implica generalmente un antialias con un filtro de corte, como un Chebyshev, elíptica, o Butterworth. No viéndose afectada la información codificada no se ve afectada por este tipo de distorsión.

En contraste, la codificación en el dominio del tiempo utiliza la forma de la onda para almacenar información. Por ejemplo, los médicos pueden controlar la actividad eléctrica del corazón de una persona por su conexión a los electrodos del pecho y los brazos (un electrocardiograma o ECG). La forma de la onda del ECG proporciona la información solicitada, como en el caso de las diversas cavidades se contraen, durante un latido del corazón. Las imágenes son otro ejemplo de este tipo de señal. Más que una forma de onda que varía con el tiempo, la codificación de imágenes varía más con la distancia. Las imágenes se forman a partir de las regiones de brillo y de color, y cómo se relacionan con otras regiones del brillo y color.

El problema se encuentra en que el teorema de muestreo es un análisis de lo que sucede en el dominio de la frecuencia durante la digitalización, lo que lo hace que sea ideal para comprender la conversión analógico-digital de las señales que tengan su información codificada en el dominio de la frecuencia. Sin embargo, el teorema de muestreo es poca ayuda en la comprensión de cómo las señales codificadas en el dominio del tiempo debe ser digitalizada.

La Figura 2.14 [SMI97] (a) es un ejemplo de señal analógica a ser digitalizada. En este caso, la información que queremos capturar es la forma rectangular de los pulsos. Una breve ráfaga de una señal de alta frecuencia sinusoidal se incluye también en este ejemplo la señal. Esto representa ruido de banda ancha, interferencias y similares que siempre aparece en las señales analógicas. Las otras figuras muestran cómo la señal digitalizada aparecería con diferentes opciones de filtros antialias: un filtro de Chebyshev, un filtro de Bessel, y ningún filtro.

Es importante comprender que ninguna de estas opciones permitirá a la señal original a ser reconstruida a partir de los datos muestreados. Esto se debe a que la señal original en sí contiene componentes de frecuencia superior a la mitad de la tasa de muestreo. Dado que estas frecuencias no pueden existir en la señal digitalizada, la señal reconstruida no puede contenerlas. Estas altas frecuencias provienen de dos fuentes: (1) el ruido y las interferencias que se pretenden eliminar, y (2) picos agudos en la forma de onda, que probablemente contengan información que desea mantener.

El filtro de Chebyshev, que se muestra en (b), ataca el problema de manera agresiva por la eliminación de todos los componentes de alta frecuencia. Esto resulta en una señal analógica filtrada que puede ser perfectamente muestreada y posteriormente reconstruida. Sin embargo, la señal analógica reconstruida es idéntica a la señal filtrada, no la señal original. Aunque nada se pierde en el muestreo, la forma de onda se ha visto gravemente distorsionada por el filtro de antialias, como se muestra en (b).

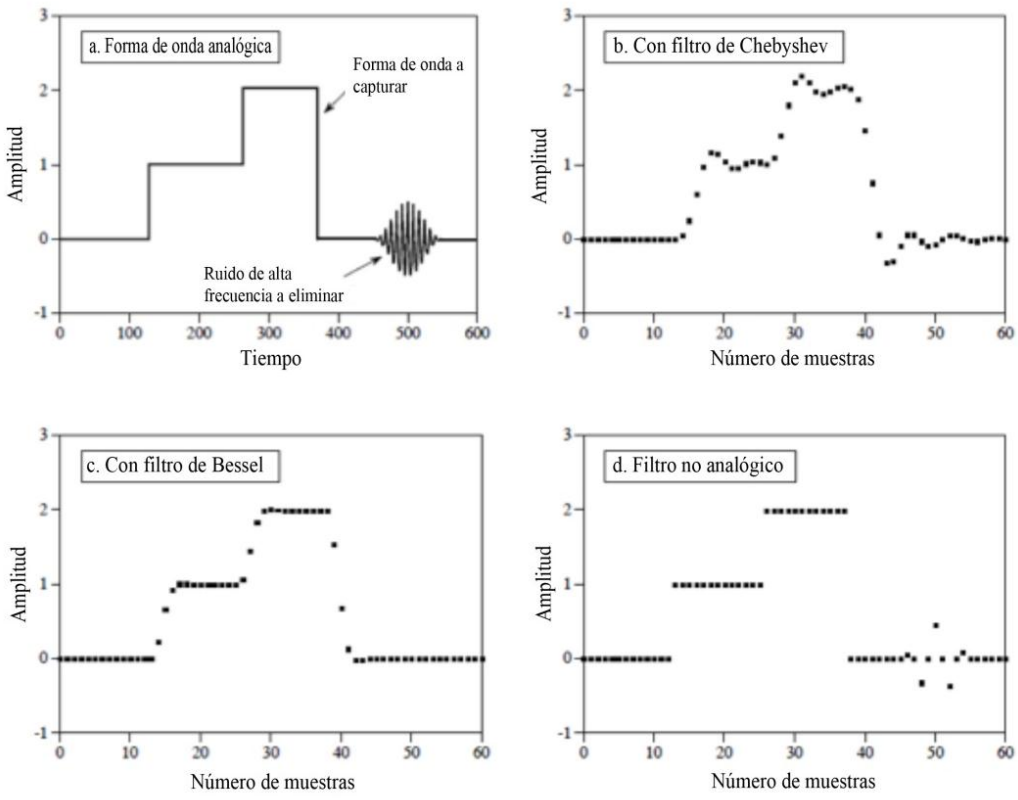


Figura 2.14 Opciones para la digitalización de una señal codificada en el dominio de tiempo.

El filtro de Bessel, (c), está diseñado sólo para este problema, su salida se asemeja a la forma de onda original, con un suave redondeo de los bordes, ajustando la frecuencia de corte del filtro, la suavidad de los bordes puede ser objeto de tratamiento para la eliminación de componentes de alta frecuencia en la señal. El uso de más polos en el filtro permite un mejor compromiso entre estos dos parámetros. Una directriz es establecer la frecuencia de corte en alrededor de una cuarta parte de la frecuencia de muestreo. Esto resulta en cerca de dos muestras a lo largo de la porción creciente de cada borde. Tanto el filtro de Bessel y el filtro de Chebyshev han eliminado las ráfagas de ruido de alta frecuencia presentes en la señal original.

La última opción es no utilizar ningún filtro antialias, como se muestra en (d), teniendo la ventaja de que el valor de cada muestra es idéntico al valor de la señal analógica, o sea, los bordes tienen perfecta nitidez; un cambio en la señal original es inmediatamente reflejado en los datos digitales. La desventaja es que puede distorsionar la señal por aliasing. Esto se hace de dos formas diferentes. En primer lugar, la interferencia de alta frecuencia y el ruido, como ejemplo la ráfaga sinusoidal, se convertirán en muestras sin

sentido, como se muestra en (d). Es decir, cualquier ruido de alta frecuencia presente en la señal analógica aparecerá como ruido de aliasing en la señal digital. En un sentido más general, esto no es un problema de la toma de muestras, sino un problema de la electrónica analógica. Puede suceder que un filtro de Bessel deba colocarse antes del digitalizador para controlar este problema, sin embargo, esto significa que el filtro debe considerarse como parte del procesamiento analógico, y no algo que se está haciendo para ayudar al digitalizador.

La segunda manifestación de solapamiento es más sutil. Cuando ocurra un suceso en la señal analógica (como un borde), la señal digital en (d), se detecta el cambio en la próxima muestra. No hay información en los datos digitales para indicar lo que sucede entre las muestras.

2.2.3 Conversión de datos multirango

Hay una fuerte tendencia a sustituir la electrónica analógica con algoritmos. La conversión de datos es un excelente ejemplo de ello. Observando el diseño de una grabadora de voz digital, un sistema que digitaliza una señal de voz guarda los datos en forma digital y, más tarde, reconstruir la señal para la reproducción. Para almacenar el habla, el sistema debe capturar las frecuencias entre aproximadamente 100 y 3000 hertzios. Sin embargo, la señal analógica producida por el micrófono también contiene muchas frecuencias más altas, digamos hasta 40 kHz. Un enfoque es para pasar la señal analógica a través de ocho polos de paso bajo Chebyshev filtro a 3 kHz y, a continuación, muestrear a 8 kHz. En el otro extremo, el CAD reconstruye la señal analógica a 8 kHz,. Otro filtro Chebyshev a 3 kHz se utiliza para producir la señal de voz final.

Se tienen tres opciones de filtro antialias para señales codificadas en el dominio del tiempo. Siendo su objetivo eliminar las frecuencias altas (solapamiento durante la toma de muestras), mientras que al mismo tiempo conservando la nitidez de borde (que transporta la información). Hay algunas ventajas en este muestreo más rápido que el análisis directo. Por ejemplo, en el rediseño de la grabadora de voz digital utilizando una frecuencia de muestreo 64 kHz. El filtro antisolapamiento tiene ahora una tarea más sencilla: pasar todas las frecuencias-por debajo del 3 kHz, mientras que el rechazo de todas las frecuencias por encima de 32 kHz. La misma simplificación se produce para la reconstrucción filtro. En

resumen, la mayor tasa de muestreo permite que los ocho polos del filtro que se sustituye con una simple red resistencia-condensador (RC). El problema que el sistema digital está saturado con la mayor tasa de muestreo de los datos.

Las técnicas multirango también pueden utilizarse en la salida del sistema de nuestro ejemplo. Los datos de 8KHz son tomados de la memoria y son convertidos a una frecuencia de muestreo 64 kHz, un procedimiento llamado interpolación. Consistente en colocar siete muestras, con un valor de cero, entre cada una de las muestras obtenidas de la memoria. El resultado es una señal de tren de impulso digital, que contiene la banda deseada voz entre 100 y 3000 hercios, más duplicaciones espectrales entre 3 kHz y 32 kHz. Después de la conversión de una señal analógica a través de un CAD, una simple red RC es todo lo que se requiere para producir la señal de voz final.

La conversión de datos multirango es útil porque sustituye componentes analógicos con el software (ventaja económica en productos producidos en serie) y porque puede alcanzar niveles más altos de rendimiento en aplicaciones críticas.

2.2.4 Conversión de datos de bit único

Una técnica popular en las telecomunicaciones y la reproducción de música de alta fidelidad es el CAD y CDA de bit único. Estas son técnicas multirango que exigen una mayor tasa de muestreo por un menor número de bits. En el extremo, sólo un poco que se necesita para cada muestra. Aunque hay muchas configuraciones diferentes de circuitos, la mayoría se basan en el uso de la modulación delta. A continuación se muestran tres circuitos a modo de ejemplo, todos ellos, implementados en circuitos integrados.

En la Figura 2.15 se muestra el diagrama de bloques típico de un modulador delta, donde, si la entrada analógica es una señal de voz con una amplitud de unos pocos voltios, la señal de salida es un flujo de unos y ceros digitales. Una comparación decide qué tiene el mayor voltaje, la entrada de señal analógica, o la tensión almacenada en el condensador. Esta decisión, en forma de un cero o uno digital, se aplica al interruptor de entrada. En cada de pulso de reloj, normalmente en unos pocos cientos de kHz, sea cual sea el conmutador transfiere cualquier estado digital que esté siendo aplicado a la entrada del conmutador. La salida del interruptor es actualizada en sincronización con el reloj, y usada en la realimentación para que el voltaje del condensador siga la tensión de entrada.

La Figura 2.15 [SMI97] muestra el diagrama de bloques típico de un modulador delta. Para implementar un circuito de realimentación se toma la salida digital y se utiliza un interruptor electrónico para controlar el circuito. Si la salida es digital, el conmutador conecta el condensador a una carga positiva. Este es un circuito que aumenta la tensión en el condensador en una cantidad fija, por ejemplo 1 mV por ciclo de reloj. Esto puede realizarse con tan sólo una resistencia conectada a una tensión positiva. Si la salida digital es un cero, el conmutador está conectado a una carga negativa, disminuyendo la tensión en el condensador por la misma cantidad fija.

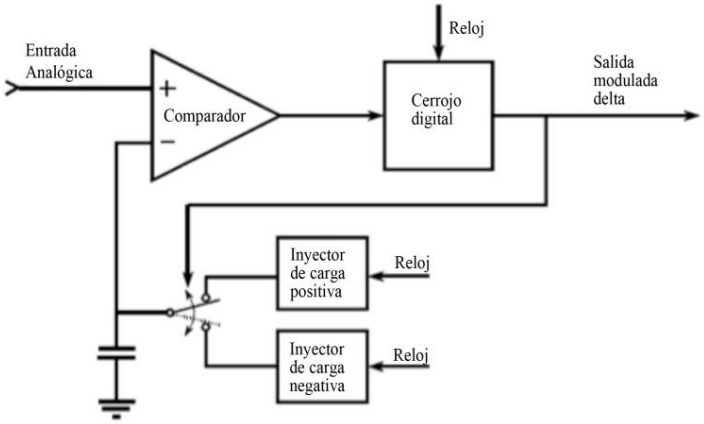


Figura 2.15 Diagrama de bloques típico de un modulador delta.

La Figura 2.16 [SMI97] muestra las señales producidas por este circuito. En el instante igual a cero, la entrada analógica y la tensión en el condensador tienen una tensión igual a cero. Como se muestra en (a), la señal de entrada de repente se eleva a 9,5 voltios en el octavo ciclo de reloj. Dado que la señal de entrada es ahora más positiva que la tensión en el condensador, la salida digital cambia a 1, como se muestra en (b). Esto sucede cuando el conmutador está conectado a la carga positiva del inyector, y la tensión creciente en el condensador por una pequeña cantidad en cada ciclo de reloj. A pesar de un incremento de 1 V por ciclo de reloj se muestra en (a), esto es sólo con fines ilustrativos, y un valor de 1 mV es más típico.

Este aumento de la tensión en escalera del condensador continúa hasta que se supere el voltaje de la señal de entrada. En este caso, el sistema alcanzado un equilibrio con la salida digital oscila entre uno y cero, provocando la tensión en el condensador a oscilar entre los 9 V y 10 V. De esta manera, el circuito de realimentación fuerza al condensador a cargarse

con la tensión de la señal de entrada. Si la señal de entrada cambia muy rápidamente, la tensión en el condensador cambia a un ritmo constante (slew rate) hasta que se obtiene de un valor equivalente.

Ahora, considerando las características de la señal de salida delta modulada, si la entrada analógica está aumentando en valor, la señal de salida se compondrá más unos que ceros. Del mismo modo, si la entrada analógica está disminuyendo de valor, la salida consistirá en más ceros que unos. Si la entrada analógica es constante, la salida digital se alternará entre cero y uno con un número igual de cada uno. Puesto en términos más generales, el número relativo de los ceros es directamente proporcional a la pendiente de la entrada analógica.

Este circuito proporciona un método barato de transformar una señal analógica en digital. Una característica especialmente interesante es que todos los bits tienen el mismo significado, a diferencia del formato convencional de serie: bit de arranque, LSB, MSB, bit de parada, etc. El circuito en el receptor es idéntico al bloque de realimentación del circuito origen información de la transmisión de circuito. Tal como hace la tensión en el condensador del circuito de transmisión siguiendo la entrada analógica, así mismo actúa la tensión en el condensador del circuito de recepción. Es decir, la tensión en el condensador (a) representa también la forma en que la señal reconstruida aparecería.

Una limitación fundamental de este circuito es el compromiso inevitable entre la pendiente máxima, la cuantización de tamaño, y la tasa de datos. En particular, si la pendiente máxima y el tamaño de cuantización se ajustan a valores aceptables para la comunicación de voz, la velocidad de transmisión de datos termina en los MHz.

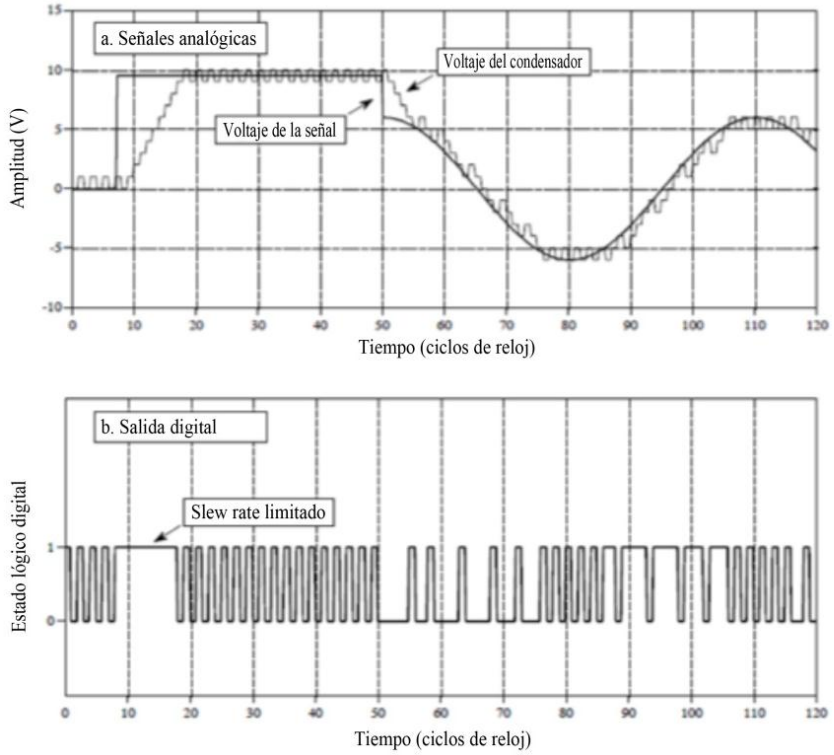


Figura 2.16 Ejemplo de las señales producidas por el modulador delta en la Figura 2.14. Figura (a) muestra la señal de entrada analógica, y la correspondiente tensión en el condensador. La figura (b) muestra la salida delta modulada, un flujo digital de unos y ceros.

Una solución a este problema es el modulador CVSD, una técnica que se aplicó en la familia Motorola MC3518. En este enfoque, el reloj y el tamaño de cuantización se establecen a un valor razonable. Resultando en una pendiente anormal que se corrige con circuitería adicional. Durante el funcionamiento, una resistencia variable examina continuamente los últimos cuatro bits que el sistema ha producido. Si el circuito se encuentra en una condición de pendiente limitada, los últimos cuatro bits serán todos aquellos (pendiente positiva) o todos los ceros (pendiente negativa). Una lógica de circuito detecta esta situación y produce una señal analógica que aumenta el nivel de carga producida por los componentes de carga. Esto aumenta la pendiente por la tasa de aumento del tamaño de los pasos de tensión que se aplican a los condensadores.

Un filtro analógico normalmente se coloca entre la lógica y la circuitería de carga. Esto permite que el tamaño de paso dependa de cuánto tiempo ha estado el circuito en una condición de pendiente limitada. Denominándose filtro silábico, ya que sus características dependen de la duración media de las sílabas que componen el habla.

Según las especificaciones, datos de 16 a 32 kHz producen una calidad aceptable de expresión. Los continuos cambios en el tamaño de los pasos hace más difícil entender los datos digitales, pero afortunadamente, no es necesario. En el receptor, la señal es reconstruida incorporando un filtro silábico que es idéntico al del circuito de transmisión. Si los dos filtros son igualados, resulta poca la distorsión de la modulación CVSD. CVSD es probablemente la forma más fácil de transmitir digitalmente una señal de voz.

Mientras que la modulación CVSD es ideal para la codificación de señales de voz, no puede ser para CAD de propósito general.

El convertidor delta-sigma, que se muestra en la Figura 2.18, elimina estos problemas combinando inteligentemente la electrónica analógica con algoritmos PDS. Se observa que la tensión en el condensador se está comparando con 0V. El bucle también se ha modificado de modo que la tensión en el condensador disminuye cuando el circuito de salida es un uno digital, y aumenta cuando se trata de un cero digital. Como la señal de entrada aumenta y disminuye su tensión, trata de subir y bajar la tensión en el condensador.

Si la tensión de entrada es positiva, la salida digital se compone de más unos que ceros. El exceso de unos que se necesita para generar la carga negativa que se cancela con la positiva señal de entrada. Asimismo, si la tensión de entrada es negativa, la salida digital se compone de más ceros que unos con una carga positiva neta. Si la señal de entrada es igual a 0 V, un número igual de unos y ceros se generará en la salida, siendo cero el aporte de carga.

El número de unos y ceros en la salida está relacionado con el nivel de la tensión de entrada, no como la pendiente en el anterior circuito. Esto es mucho más simple. Por ejemplo, se puede formar un CAD de 12 bits alimentando con la salida digital un contador, y contando el número de unos durante 4.096 ciclos de reloj. Un número digital de 4095, que correspondería a la máxima tensión de entrada positiva. Asimismo, el 0 digital corresponde a la máxima tensión de entrada negativa, y 2048 corresponde a una tensión de entrada de 0 V. Esto también muestra el origen de su nombre, el delta-sigma: modulación delta seguida del sumador (sigma).

Los unos y ceros producidos por este tipo de modulador delta son muy fáciles de transformar de nuevo en una señal analógica. Todo lo que se requiere es un análogo filtro analógico pasa bajo, que podría ser tan simple como una única red RC.

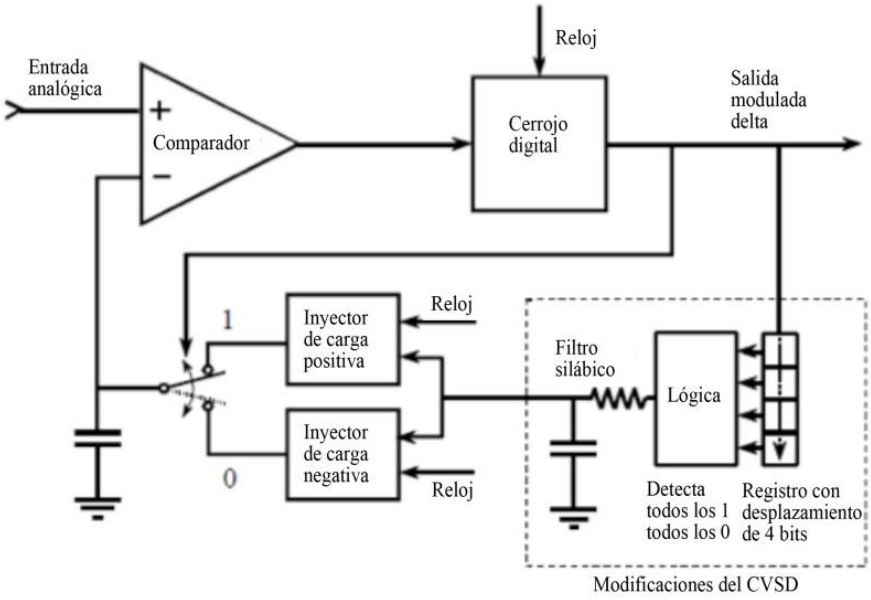


Figura 2.17 Diagrama de bloques modulación CVSD. Una lógica del circuito se añade a la base del modulador delta para mejorar pendiente.

Los valores altos y bajos de tensión digitales correspondientes a los unos y ceros se promedian para formar voltaje analógico correcto. Por ejemplo, supongamos que los unos y ceros están representados por 5 voltios y 0 voltios, respectivamente. Si el 80% de los bits en el flujo de datos son, unos y el 20% son ceros, la salida del filtro pasa bajo será de 4 V.

Este método de transformar el único bit de datos de nuevo en la forma de onda es importante por varias razones. En primer lugar, se describe una forma hábil de sustituir el contador en el circuito la CDA delta-sigma. En lugar de limitarse a contar los impulsos de la delta del modulador, la señal binaria se pasa a través de filtro digital pasa bajo y, a continuación, diezmado para reducir la tasa de muestreo. Por ejemplo, este procedimiento podría empezar por cambiar cada uno de los unos y ceros en el flujo digital de 12 bits en una muestra, los unos se convierten en un valor de 4095, mientras que los ceros un valor de 0. Utilizando un filtro pasa bajo sobre esta señal produce una versión digitalizada de forma de onda original, como un filtro analógico pasabajo. La decimación entonces reduce la velocidad de muestreo descartando la mayoría de las muestras, dando como resultado una señal digital que es equivalente a muestrear directamente la forma de onda original.

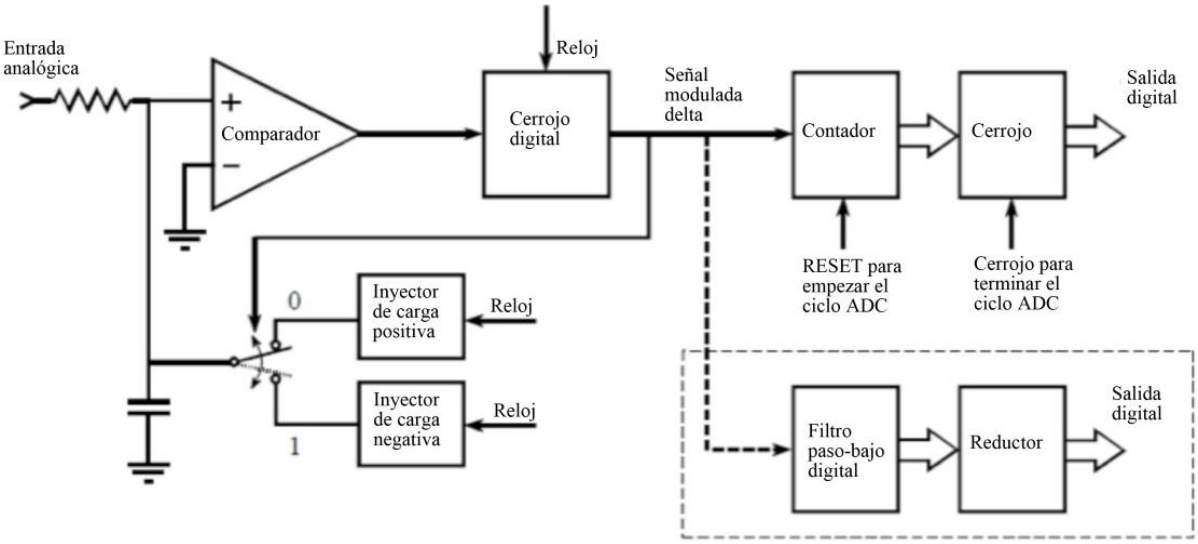


Figura 2.18 Diagrama en bloque de un conversor delta-sigma analógico-digital. En el caso más simple, los impulsos de un modulador delta se cuentan para un número predeterminado de ciclos de reloj. La salida del contador es capturada para completar la conversión. En un circuito más sofisticado, los pulsos se pasan a través de un filtro digital pasa bajo y, a continuación, remuestreado (diezmado) a una menor tasa de muestreo.

Este enfoque se utiliza en muchos CAD comerciales para digitalizar la voz y otras señales de audio. Un ejemplo es el ADC16071, que dispone de conversión de 16 bits analógico-digital con una frecuencia de muestreo de hasta 192 kHz. A una tasa de muestreo de 100 kHz, el delta del modulador opera con una frecuencia de reloj de 6,4 MHz. El filtro pasa bajo digital FIR de 246 etapas elimina todas las frecuencias de los datos digitales de más de 50 kHz, la mitad aproximadamente de la frecuencia de muestreo. Conceptualmente, esto puede ser visto como la formación de una señal digital a 6,4 MHz, con cada muestra representada por 16 bits. La señal es entonces diezmada de 6,4 MHz a 100 kHz, logrado mediante la supresión de todos los 63 de las 64 muestras.

Los conversores delta-sigma también se pueden utilizar para CDA de la voz y las señales de audio. La señal digital se recupera de la memoria, y se convertirá en un flujo delta modulado de unos y ceros. Como se mencionó anteriormente, esta señal de bit único puede ser fácilmente modificada en la señal analógica reconstruida con un simple filtro paso bajo analógico. Con el filtro antialias, generalmente una sola red RC es requerida. Esto se debe a que la mayoría del filtraje se realiza mediante filtros digitales de alto rendimiento.

Los CAD delta-sigma tienen varias peculiaridades que limitan su uso a aplicaciones específicas. Por ejemplo, es difícil multiplexar sus entradas. Cuando se cambia la entrada de una señal a otra, no hay un funcionamiento correcto, no se establece hasta que del filtro digital se puedan eliminar datos de la señal anterior. Los convertidores Delta Sigma están limitados en otros aspectos también: no se sabe exactamente cuando fue tomada cada una de las muestras. Cada muestra adquirida es una composición de la información tomada de un bit de la señal de entrada. Esto no es un problema para las señales codificadas en el dominio de la frecuencia, tales como el audio, pero es una limitación significativa las señales codificadas en el dominio del tiempo. Para entender la forma de onda de una señal a menudo se necesita saber el instante en que fue tomada la muestra. Por último, la mayoría de estos dispositivos están diseñados específicamente para aplicaciones de audio, y sus especificaciones de rendimiento se evalúan en ese sentido. Por ejemplo, un CAD de 16 bits utilizado para señales de voz no significa necesariamente que cada muestra tiene 16 bits de precisión. Es probable, que el fabricante diga que las señales de voz pueden ser digitalizadas a 16 bits de rango dinámico con el mismo. No se espera tener 16 bits de información útil con este dispositivo de propósito general para la adquisición de datos.

Resumen:

En este capítulo se describen. los Procesadores Digitales de Señales (DSP), sus características y arquitectura interna. A continuación se muestra el DSP empleado, la plataforma de desarrollo y las técnicas actuales de optimización de código para dicha arquitectura.

3 ARQUITECTURAS BASADAS EN DSPs

El uso de DSP (Digital Signal Processor - Procesador Digital de Señales) para procesamiento de señales en tiempo real va en progresivo aumento, ampliándose en aplicaciones no sólo en el ámbito de la defensa, del audio o las telecomunicaciones, sino incluso en elementos de seguridad activa.

Aunque el DSP tiene un núcleo basado en un procesador, realmente presenta una gran cantidad de elementos que lo diferencian y complican su comprensión. Por ello, se comienza aquí mencionando que el procesamiento digital de señales (PDS) es una de las tecnologías más importantes que conforman y conformarán la ciencia y la ingeniería del presente siglo. Cambios revolucionarios han tenido lugar en un amplio rango de campos: comunicaciones, imágenes médicas, radar, sonar y reproducción de música de alta fidelidad, etc. Cada una de estas áreas ha profundizado en la tecnología PDS, con sus propios algoritmos, matemáticas y, técnicas especializadas. El gran abanico de aplicaciones y el desarrollo alcanzado por esta tecnología dificulta que alguien pueda dominar todo su amplio espectro. La formación en el PDS contempla dos tareas principalmente: el aprendizaje de conceptos generales que aplican al campo como un todo y el aprendizaje de técnicas especializadas en una determinada área.

Se distingue de otras áreas de las ciencias de la computación por el único tipo de dato empleado: las señales. En la mayoría de los casos estas señales provienen del mundo real: vibraciones sísmicas, imágenes visuales y ondas sonoras, entre otras. El PDS es una materia interdisciplinar, aunando matemáticas, algoritmos y técnicas de tratamiento de

señales, una vez que éstas han sido convertidas en digitales. Presentando una gran variedad de objetivos, tales como: mejoramiento de imágenes visuales, reconocimiento y generación de voz, compresión de datos para almacenamiento y trasmisión, etc.

3.1 Características de los DSP

Los comienzos del PDS se remontan a los años 60s y 70s cuando empezaron a salir las primeras computadoras digitales. En sus inicios estaba sólo dirigido a algunas aplicaciones críticas de seguridad nacional, como radar y sonar; exploración de petróleo; exploración del espacio; y captura de imágenes médicas. La expansión de los ordenadores personales de los años 80s y 90s provocó una revolución en el PDS con nuevas aplicaciones, introduciéndose muy rápidamente en el mercado comercial, más que por las necesidades militares y gubernamentales en esta segunda etapa. Posteriormente llegó al resto de la población en productos tales como teléfonos móviles, reproductores de discos compactos y mensajería de voz. La Tabla 3.1 muestra algunas de estas aplicaciones.

DSP	Espacio	Realce de fotografías espaciales Compresión de datos Análisis de sensores inteligentes mediante pruebas remotas
	Médicas	Diagnóstico por la imagen (RMN, ultrasonidos) Análisis de electrocardiogramas Recuperación/almacenaje de imágenes médicas
	Audio	Filtros Ecualizadores Sintetizadores
	Vídeo	Tratamiento de imágenes Efectos visuales Compresión y descompresión de imágenes

		MPEG HDTV Animación.
	Comercial	Compresión de imagen y sonido para presentaciones multimedia Efectos especiales en películas Video conferencias
	Telefonía	Compresión de datos y voz Cancelación del eco Multiplexamiento de señales Filtrado Telefonía móvil
	Telecomunicaciones	De/codificadores de señales y voz Modems ADSL Difusión de radio digital
	Militar	Radar Identificación de blancos Sonar Comunicaciones seguras
	Industrial	Control Industrial Prospecciones petrolíferas y minerales Procesos de monitorización y control CAD y diseño de herramientas
	Automócion	ABS Control estabilidad Inyección
	Científico	Análisis y grabación de terremotos Adquisición de datos Análisis espectral Simulación y modelado

Tabla 3.1Algunas aplicaciones del PDS.

Por otro lado, a principios de los años ochenta, el PDS se enseñaba como parte de cursos de enseñanza postgraduada para ingenieros eléctricos y electrónicos, una década más tarde ya formaba parte del programa de postgrado de esta especialidad. Hoy el PDS constituye una formación imprescindible para científicos e ingenieros de muchos campos.

Los avances en este campo en los últimos años han tenido un tremendo impacto en el desarrollo del PDS. Al analizar cada una de sus aplicaciones, se constata su carácter interdisciplinar. La Tabla 3.2 sugiere la relación entre el PDS y otras disciplinas técnicas, aunque no siempre está muy definida. Aunque está claro que para quien se quiera especializar en PDS resulta interesante conocer las principales áreas de la ciencia y la tecnología en las cuales se emplea PDS.

Procesado Digital de Señales	Teoría de la comunicación Análisis numérico Estadística y probabilidad Procesado analógico de señales Teoría de decisión Electrónica digital Electrónica analógica

Tabla 3.2 Areas de la ciencia y la tecnología en las cuales se emplea PDS.

En la actualidad unos de los DSP más relevantes son los producidos por Texas Instruments (TI), cuyo primer DSP producido, el TMS32010, operaba a una velocidad de 10 MHz, ejecutando en paralelo 5 MIPS en un solo ciclo de 200 ns, distando bastante de los actuales, observándose su gran avance tecnológico. Una de las principales mejoras de los actuales es que son escalables, permitiéndoles así trabajar en paralelo con otros dispositivos, y por lo tanto mejorando tiempos y eficacia.

Como se ha comentado anteriormente, las áreas de aplicación de los DSP se han visto ampliadas, no limitándose al procesado de señales, sino que también, por ejemplo, se aplican al control de motores, convertidores de corriente, sensores, etc. Esta amplitud de aplicaciones ha favorecido la gran expansión en la aplicación de estos dispositivos. Así mismo, otro aspecto importante y al que normalmente no se le presta la debida atención, es el consumo de energía necesario, el cual, se ha visto muy reducido y con ello se ha

mejorado la fiabilidad y la vida útil del dispositivo, ya que al tener un menor consumo también sufre un menor calentamiento, que es uno de los principales enemigos de los dispositivos electrónicos.

Como ejemplos de aplicaciones destacables en la actualidad, además de las implementaciones en sistemas de equipos de alta fidelidad, sistemas de separación ciega de señales, y en general, procesamiento de señales, se pueden observar otras como:

- Convertidor de frecuencia para controlar un motor de inducción (IM). Implementado con una tarjeta DS1102 del fabricante dSPACE, la cual, combinada con el software de diseño, análisis y optimización de algoritmos de control, reduciendo los tiempos de desarrollo. Este DSP es programable desde el diagrama de bloques de Simulink, permitiendo una integración con MATLAB.
- Control de velocidad de un sistema de generación eléctrica mediante energía eólica. En este caso emplea un generador IPMSG (Interior Permanent-Magnet Synchronous Generator), y utiliza un sistema de prueba con un servomotor de AC, en lugar de la turbina de viento. El sistema es controlado por un DSP TMS320C32.

3.2 Arquitectura interna de los DSP

Los DSP son procesadores diseñados para optimizar dos factores: coste y consumo manteniendo unas prestaciones elevadas. Combinan aspectos de los microcontroladores como los elementos integrados: memoria, elementos de E/S, control de interrupciones, … y en bastantes ocasiones son una solución alternativa para diseños basados en electrónica analógica.

3.2.1 Diferencias de un DSP con una CPU convencional

Desde el punto de vista del software los DSP suelen ejecutar aplicaciones cíclicas, de duración acotada, donde se requiere altísima eficiencia de ejecución. Para ello se suele optar por el uso de ensamblador y dialectos especiales del lenguaje C para optimizar el código. Algunos algoritmos usuales son: filtrado, convolución, correlación.

Por sus recursos de hardware se pueden encontrar:

- Modos de direccionamiento especializados: (Ej: bit-reversal, colas circulares)
- Varias unidades de procesamiento operando en forma concurrente (MAC, Barrel Shift,...)
- Operaciones aritméticas especiales: algunos DSPs con unidades FP.
- Esquema de temporización e interrupciones mucho más orientado a acciones en tiempo real
- Pocos o nulos recursos que generan latencias, como memoria virtual, caches, etc.
- Arquitecturas tipo HARVARD con mapas de datos e instrucciones separados. Dos o más mapas de memoria de datos que permiten leer concurrentemente operandos y coeficientes
- Manejo especializado de punteros de direcciones a través de unidades de cálculo dedicadas
- Opciones para la digitalización y captura de señales con intervalos regulares (DMA)
- Recursos internos o dispositivos periféricos especializados para la conversión A/D y D/A de señales, así como para el filtrado anti-alias y la reconstrucción

Los DSP están diseñados para ofrecer una elevada capacidad de procesamiento aritmético de datos en tiempo real, con elevada precisión, para evitar problemas de redondeo y truncamiento. Para ello incorporan etapas Multiplicadora/Acumuladora (MAC) que permiten resolver ecuaciones del tipo A = A + (B x C) en un único ciclo. Además, para operaciones de punto fijo se incluyen circuitos Barrel Shifter (BS) para desplazar un dato varios bits a derecha/izquierda en un único ciclo de instrucción. Algunos DSP incluyen una ALU que opera de forma independiente al MAC y al BS. Para utilizar estas unidades de forma eficiente se utilizan códigos de operación para controlar MAC, ALU y BS en una única instrucción (varias operaciones concurrentes).

3.2.2 Evolución de los DSP

En la Figura 3.1 se muestra la evolución histórica de los DSP de distintos fabricantes. Partiendo de la estructura básica de los primeros DSP que ya contenían las unidades MAC,

los fabricantes evaluaron el gran potencial que tenían por lo que desarrollaron familias específicas con sus propias características. Rápidamente incorporaron la arquitectura Harvard, la segmentación de cauce e incluso unidades de punto flotante en algunos DSP.

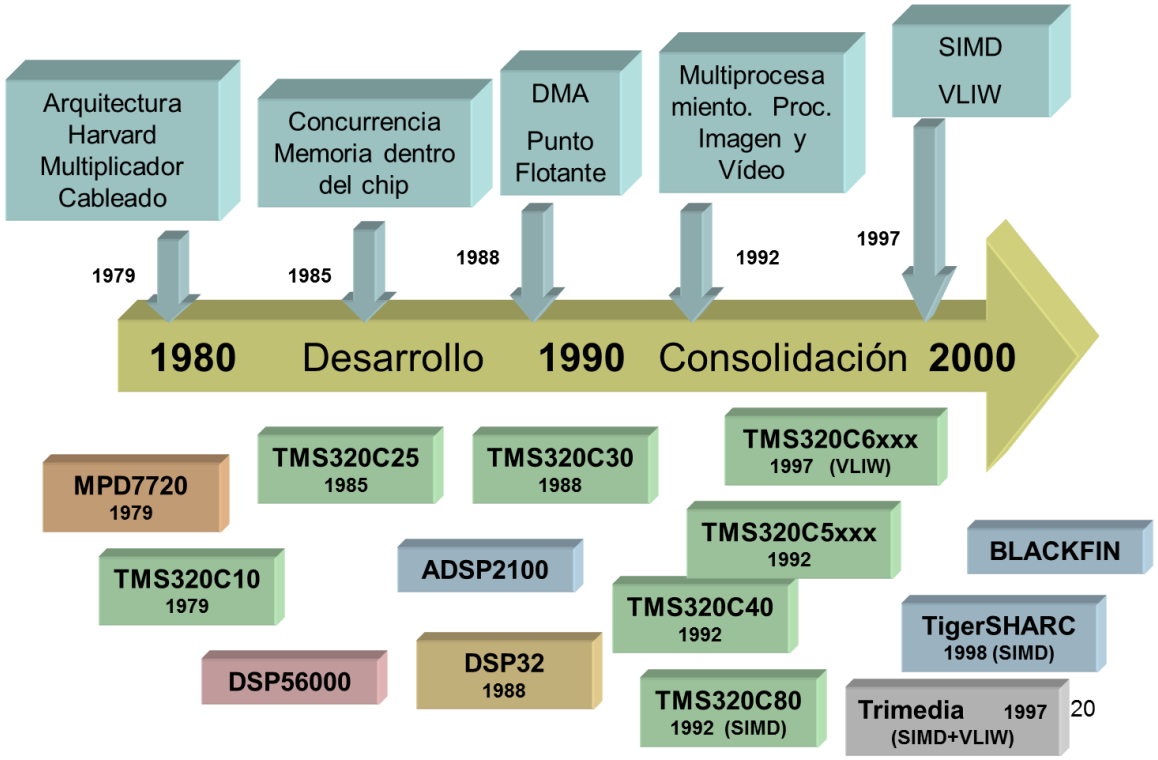


Figura 3.1 Etapas principales en la evolución de los DSP.

Incluso la familia de DSP que se utiliza en este trabajo que corresponde con el TMS320C6xxx se empezó a introducir en 1997, pero durante los últimos años estos procesadores han evolucionado continuamente con el doble flujo de instrucciones en paralelo, mejoras de memoria cache, mayor frecuencia de funcionamiento o inclusión de aceleradores hardware.

El fabricante, Texas Instruments ha ampliado su línea de productos con los denominados procesadores media. Son Sistemas en un chip (SoC) que incluyen un procesador de propósito general (generalmente procesadores ARM) para incluir un sistema operativo empotrado (Linux, Windows, …) y un DSP orientado exclusivamente al procesamiento de señales. En la Figura 3.2 se muestra el diagrama de bloques de un sistema basado en el TMS320CDM6446 para un video-teléfono IP. Este circuito incluye los controladores de memoria y algunos elementos de interconexión lo que reduce el número de componentes finales.

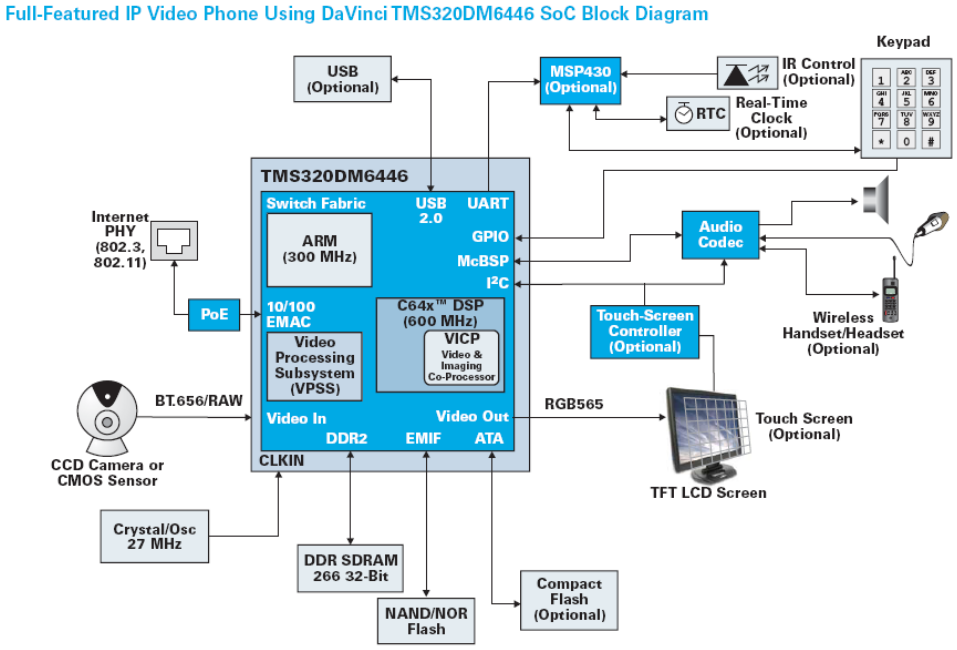


Figura 3.2 Diagrama de bloques de un video-teléfono basado en el TMS320DM6446.

Aprovechando la densidad de integración que ofrece la tecnología, los procesadores actuales pueden incorporar millones de transistores. Igual que los procesadores de propósito general han introducido varios núcleos para aumentar su potencia de cálculo algunos DSP de gama alta también los incorporan. Es el caso de la familia MSC8XXX de FreeScale que se muestra en la .Figura 3.3.

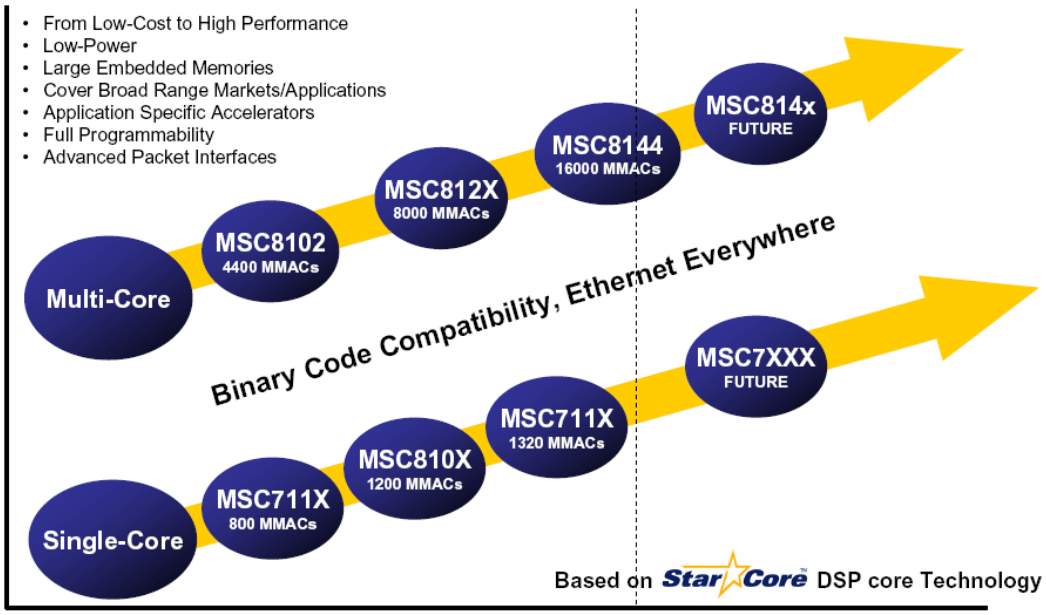


Figura 3.3Familias mono-núcleo y multi-núcleo de DSP de FreeScale.

Finalmente, la Figura 3.4 muestra las diferentes familias de DSP del fabricante Texas Instruments. Como se muestra en la columna de la derecha el nombre completo de los procesadores es TMS320CXXXX, pero tanto el fabricante como en la bibliografía se utilizan las últimas cifras para hacer referencia a algún procesador of familia de ellos. Para mostrar el funcionamiento de un DSP se mostrará en el siguiente apartado un DSP de la familia C2x. Como puede observarse en dicha figura, el DSP utilizado en este trabajo corresponde con la familia C6000 de altas prestaciones.

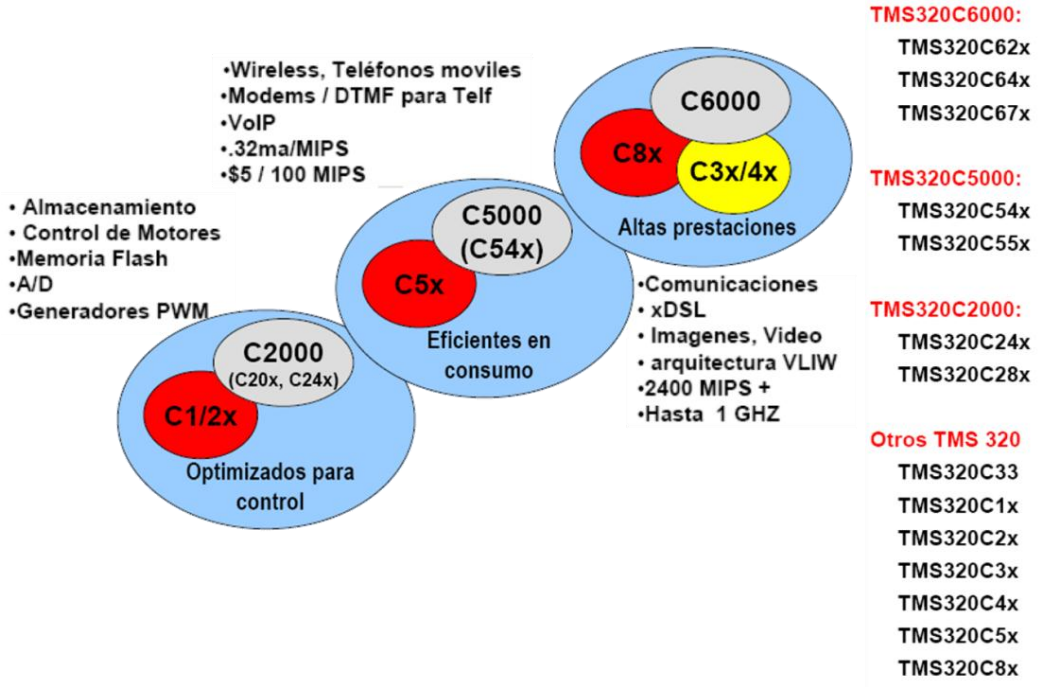


Figura 3.4 Familia de DSPs de Texas Instruments.

3.2.3 DSP TMs320C2x

Aunque esta familia de DSP de punto fijo se desarrolló en los años 80, se siguen utilizando en la actualidad para aplicaciones sencillas de procesamiento digital de señales. En este apartado se describe a título ilustrativo ya que permite familiarizarse con los elementos internos de un DSP y la optimización que puede desarrollarse con ellos.

Estos procesadores incluyen diversas unidades que pueden funcionar simultáneamente. En la Figura 3.5 se muestran los elementos principales de dicho DSP relacionados el cálculo. Pueden funcionar simultáneamente:

- Acumulador de 32 bits
- Multiplicador de 16x16 bits

- Unidad aritmética de registros auxiliares (ARAU)
- Unidades de desplazamiento (barrel-shift) que pueden desplazar : -6, 0, 1 ó 4 bits en entrada y 0 a 7 bits a la salida del acumulador.
- Acceso simultáneo a memoria de programa y datos por su arquitectura Harvard.

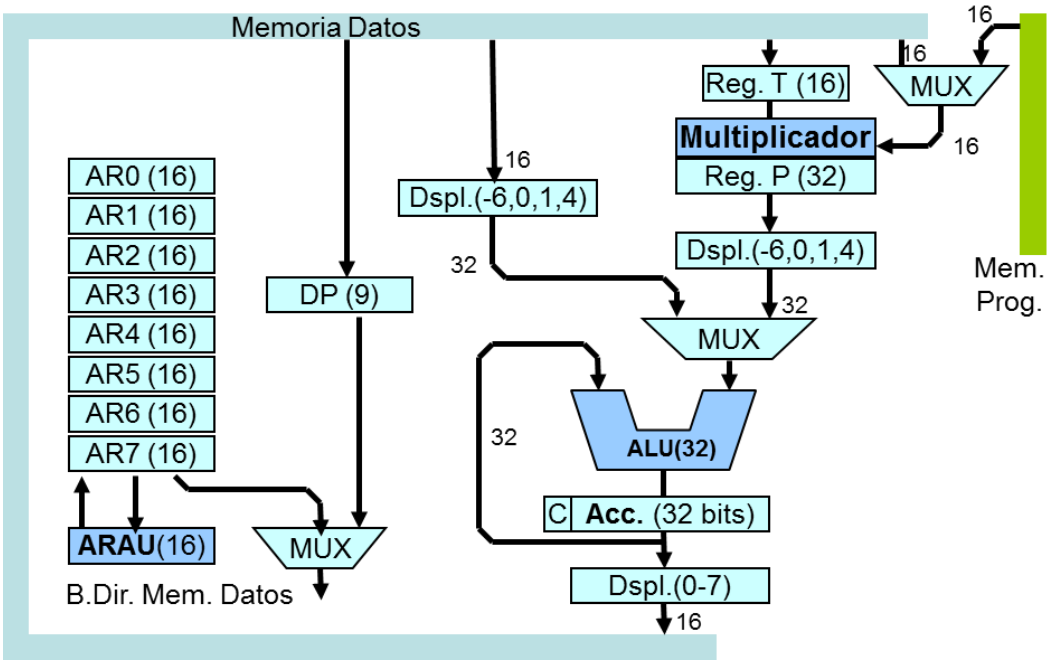


Figura 3.5 Elementos principales de procesamiento de los DSP TMS320C2x.

Para ver como funcionan varias de estas unidades de forma simultánea se muestra como ejemplo la implementación de un filtro FIR (Finite Impulse Response).

Un filtro FIR es un tipo de filtro digital cuya respuesta a una señal impulso tiene a su salida un número finito de términos no nulos. Se implementa mediante la siguiente ecuación:

$$y_n = \sum_{i=0}^{n-1} b_i \; x(n-i) \qquad (3.1)$$

Donde N es el orden del filtro. La Figura 3.6 muestra los pasos que deben realizarse en cada etapa del filtro FIR. En cada etapa (TAP) del filtro FIR se debe realizar una multiplicación, una suma, el desplazamiento de los datos, avance de los punteros y decremento del contador. Como el DSP no puede ejecutar el producto y la suma de los

mismos elementos de forma simultánea, aunque sí puede realizar una suma y un producto simultáneamente, primero se calcula un producto, a partir del siguiente ciclo se calcula el nuevo producto y la acumulación del producto anterior, lo que permite que ambas unidades puedan funcionar al mismo tiempo. Finalmente hay que añadir el último producto para que la sumatoria incluya todos los términos.

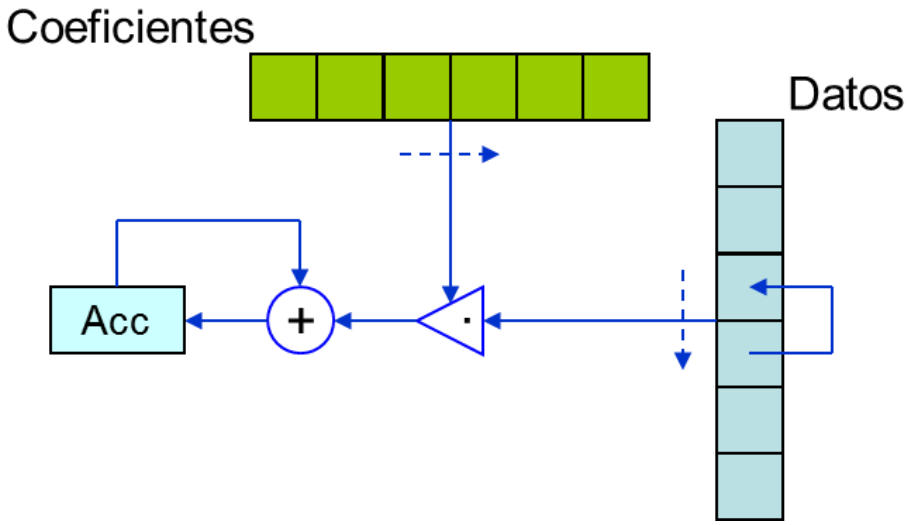


Figura 3.6 Implementación de un filtro FIR

A continuación se muestra el código para implementar digo filtro:

```
OEF       .set  02000;      Tabla de coeficientes en memoria de
          programa
ULMUES    .set  047Fh;      Posición última muestra
   LAR    AR3, #ULMUES;     Inicia el puntero de datos
   ZAC;                     Pone a 0 el acumulador
   RPTK   #127;             Repetir 128 veces la siguiente instrucción
   MACD   COEF, *-;         Multiplica, suma producto previo y desplaza
          datos
   APAC;                    Suma el último producto que quedó en el
          multiplicador
   SACH   Y,1;              Almacena el resultado (incluye
          desplazamiento)
```

La instrucción MACD del procesador es muy potente ya que: realiza la multiplicación, la suma, el desplazamiento de los datos, avance de los punteros y decremento del contador, requiriendo sólo 4 ciclos de reloj en cada iteración. Así, un DSP trabajando a 40 MHz

puede ejecutar 10 millones de etapas (TAP) de un filtro FIR por segundo. La instrucción APAC es la que incluye el último producto en la sumatoria final.

3.2.4 Plataforma EZLITE

La plataforma EZLITE fue empleada en el primer estadio del presente trabajo, y por ello resulta interesante mencionarla y conocer algunos de sus principales características. Este sistema permite implementar equipos autónomos con procesamiento avanzado de señales en tiempo real. Las principales características de éste son:

- Sistema basado en DSP.
- Completamente autónomo.
- Interfaz gráfica de usuario, pantalla LCD.
- Mandos giratorios y botones.
- Sistema de entrada/salida (Analógicas y Digitales).
- Posibilidad de aumentar el número de entradas y salidas.

El hardware del sistema parte de un sistema de evaluación de Analog Devices, el EZ-LITE. Este kit integra algunos elementos como son:

- DSP ADSP-21061.
- Codec AD1847.
- Interfaz básico de Entrada / Salida.

El elemento principal es el DSP de Analog Devices ADSP-21061, el cual ofrece las siguientes características:

- Super Harvard Architecture Computer (SHARC).
- Procesador de 32 bits orientado a aplicaciones DSP de altas prestaciones.
- Procesador de punto flotante con aritmética IEEE de 32 bits, extendida de 40 bits y formatos de datos flexibles.
- 120 MFLOPs de pico, y 80 MFLOPS sostenidos.
- Memoria interna: 128 Kbytes configurable.
- Direccionamiento externo: 4 Gwords.
- 6 canales de DMA.
- Arquitectura escalable multiprocesador.
- Velocidad de transferencia externa: 300 Mbytes/s.

El DSP es el núcleo del sistema pero requiere de una serie de elementos que lo complementan para poder disponer de un equipo realmente operativo. Para la captura de datos en tiempo real se utiliza el CODEC AD1847 que incluye como entrada un conversor sigma-delta de 16 bits de dos canales que permite muestrear hasta 48 Kmuestras/segundo, y simultáneamente como salida otros dos canales a la misma frecuencia de muestreo.

Una de las novedades que incluye la plataforma desarrollada es incluir una pantalla LCD controlada directamente por el DSP lo que permite analizar los datos de forma directa.

La pantalla LCD de Hitachi es la LM6733 con 320 x 240 pixel. Esta pantalla es muy interesante para poder observar las señales en tiempo real y observar los parámetros obtenidos.

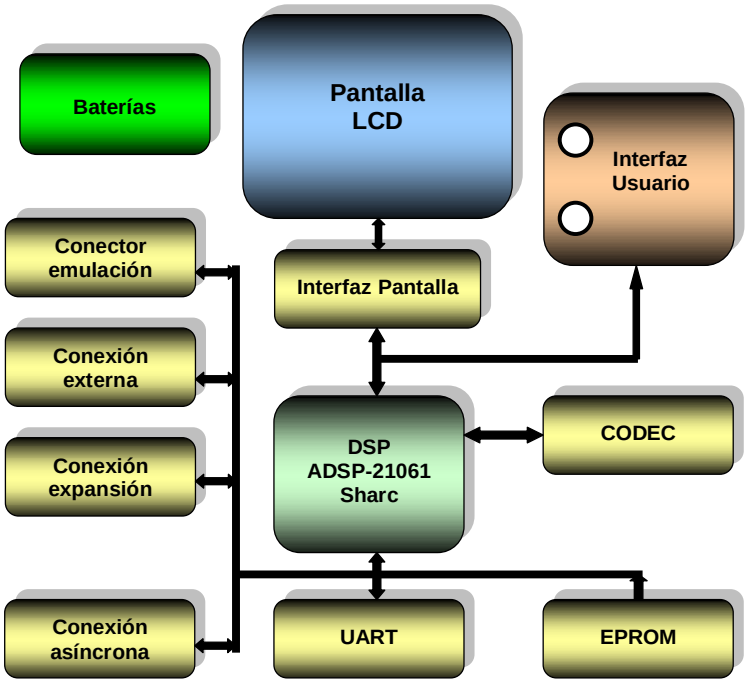


Figura 3.7 Diagrama general del sistema.

La Figura 3.7 muestra un diagrama general del sistema. En la Figura 3.8 puede observarse el aspecto del prototipo que se desarrolló.

La interfaz de usuario está compuesta de 2 mandos sin fin, codificados angularmente, para realizar funciones de movimientos ilimitados como desplazamientos. 4 botones permiten el acceso a menús y funciones rápidas.

El sistema puede operar con batería e implementar un sistema completamente autónomo lo que se puede aproximar a modelos como puede ser el de la telefonía móvil digital.

Esta plataforma desarrollada no sólo incluye elementos hardware sino también elementos software para facilitar su uso. Se incluyen funciones que facilitan el acceso al nuevo hardware como son las relacionadas con el acceso a la pantalla gráfica, trazado de líneas, escritura de texto, captura de bitmaps, y acceso a la entrada de botones y señales. Estas funciones se han escrito en ensamblador para optimizar al máximo el funcionamiento, minimizar el consumo adicional de CPU.

Figura 3.8 Prototipo desarrollado.

El kit que se utiliza como base de esta plataforma incluye para su la programación un conjunto práctico de herramientas, ensamblador, compilador de C y el entorno de depuración, Visual DSP. Éste está compuesto de dos herramientas básicas, un simulador y un emulador. Como primer paso para la depuración, el simulador permite verificar algunas partes de código sin necesidad de disponer de la plataforma.

El emulador permite la carga de programas en la plataforma para poder ejecutar en tiempo real.

Adicionalmente, se incluye un conjunto de funciones que lo hace incluso más práctico pues éstas ya están optimizadas en ensamblador y simplifica inicialmente la labor de aprendizaje. También permiten desarrollar un sistema complejo que pueda funcionar en poco tiempo. Además de las funciones matemáticas “básicas” que podemos encontrar en

bastantes implementaciones de C, (funciones trigonométricas, exponenciales y logaritmos), también podemos encontrar un conjunto de funciones implementadas en ensamblador muy prácticas como son filtros FIR, IIR, procesamiento FFT y estadística como medias, varianzas y valores rms.

También se ha trata de homogeneizar el funcionamiento interno de las aplicaciones de forma que el código puede adaptarse fácilmente a la plataforma. Para poder desarrollar código internamente se divide en dos partes:

- Tiempo real.
- Pseudo tiempo real.

El código de tiempo real generalmente está asociado con las interrupciones y tiene que ser un código reducido, o bien, requiere del procesamiento de cada muestra en el dominio del tiempo.

El código para pseudo tiempo real permite poder combinar la ejecución de algoritmos y el tiempo restante con la visualización de señales en tiempo real, FFTs, etc. En esta zona también se ejecutan las rutinas asociadas a procesamiento por ventanas, principalmente en el dominio de la frecuencia, en las que se requiere un conjunto de muestras para ser tratadas.

Con este sistema se pueden visualizar señales con el DSP, mostrando un sistema con el que puede interactuar. Muchos sistemas de evaluación están basados en un procesador, un codec y una conexión vía serie a un ordenador personal pero esto también ofrece bastantes limitaciones:

- Transmisión de los datos al ordenador por el puerto serie (suele ser lenta).
- Conflictos al compartir el puerto serie.
- La interfaz de usuario permite modificar en tiempo real parámetros que afectan al procesado de la señal.

La programación de DSP se realiza generalmente en C o ensamblador. En el caso de los procesadores convencionales, suelen ser muy limitados los casos en los que se requiere la programación en ensamblador. En el caso de los DSP, se suele necesitar dicha programación en ensamblador para conseguir optimizar el código a ejecutar en un DSP concreto.

3.3 DSP: TMS320C6000 VLIW de Texas Instruments

Se estaba buscando un sistema lo suficiente eficiente y con la capacidad necesaria de procesamiento de audio e imágenes en tiempo real, encontrando finalmente el DSP TMS320C6x Very-Long Instruction Word (VLIW). El cual cumple con todos los requisitos y exigencias que el problema planteado demanda. A continuación se detallan a grandes rasgos las características fundamentales de esta familia. En capítulos posteriores, se entrará más a fondo en la exposición del sistema y su aplicación al trabajo. Estas son algunas de sus características:

- La familia C6x se compone de la familia C62x, procesadores de 16 bits en punto flotante, y de la familia 67x de 32 bits en punto flotante. El C6x tiene dos sumadores de 32 bits y dos multiplicadores de 16x16, los cuales operan en paralelo y completan sus ejecuciones en un ciclo de reloj. En el C67x, la multiplicación en punto flotante requiere cuatro ciclos de reloj (esencialmente cuatro 16x16 multiplicaciones y numerosas sumas).
- La familia C6x no es una familia de DSP tradicional:
    - El C6x no tiene una instrucción multiplicación-acumulación. Primero hace la multiplicación y hace la acumulación tras su finalización.
    - No tiene una unidad hardware dedicada al direccionamiento módulo o acarreo invertido; sin embargo, dos de las unidades sumadoras permiten este modo de direccionamiento.
- Los procesadores de la familia C6x tienen super-segmentación. En el C62x captura, decodifica, ejecuta y termina en cuatro ciclos de reloj, 2 ciclos, y 1-5 ciclos respectivamente, en lugar de uno cada ciclo. El problema es que cuando el procesador se encuentra un bucle condicional se encuentra en la segmentación de cauce, éste no se puede detener, aunque se puede evitar usar bucles condicionales porque el C6x soporta evaluación condicional de todas las instrucciones.
- El TMS320C62x tiene seis direcciones de 32 bits y dos multiplicadores 16 x 16, los cuales operan en paralelo y terminan su ejecución en un ciclo de reloj. La acumulación múltiple en punto flotante se realiza en dos instrucciones: multiplica en un ciclo y acumula en el siguiente. Debido a la

segmentación de cauce, se pueden calcular eficazmente dos acumulaciones y multiplicación por ciclo.

La familia TMS320C67x es un DSP de 32 bits en punto flotante que tiene sus pines compatibles con el TMS320C62x. Realizando la multiplicación de 32 bit en cuatro ciclos.

La plataforma DSP empleada es la TMS320C6000 VLIW de Texas Instruments. A continuación se aportan los datos principales sobre el sistema de desarrollo basado en DSP, denominado TMS320C6000 VLIM de Texas Instruments.

Por otro lado, el DSP, TMS320C6000 VLIW tiene un alto rendimiento frente al coste y al volumen; sobresaliendo en el procesado de señales multidimensionales; con un máximo de 8 instrucciones RISC por ciclo.

En la siguiente Figura 3.9 [TEX10] se pueden observar las características principales de este sistema.

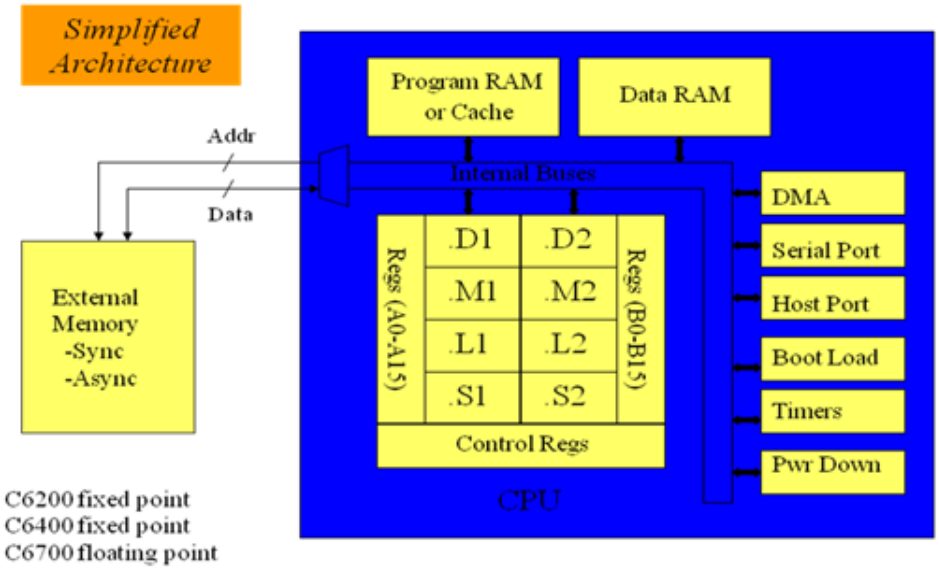


Figura 3.9 Diagrama de bloques de la familia C6x de Texas Instrument.

De entre las cuales cabe destacar las siguientes:

- Direccionamiento de 8/16/32 bit de datos + 64 bit de datos en el C67x
  - Almacenamiento-carga en arquitectura RISC con 2 caminos de datos
  - Registros de 16/32 bit para el camino de datos (A0-A15 y B0-B15)
- 48 instrucciones en el C6200 y 79 en el C6700
  - 2 caminos de datos paralelos con unidades RISC de 32 bit

  - Direccionamiento de cálculo de 32 bit de la unidad de datos (módulo, lineal)
  - Unidad multiplicadora de 16 bit x 16 bit con 32-bit para el resultado
  - Unidad lógica de 40 bit (con saturación), aritmética y de comparaciones
  - Unidad de desplazamiento de 32 bit enteros ALU y 40 bit desplazados
  - Ejecución condicionales basada en registros A1-2 y B0-2
- Puede trabajar con 2 paquetes de medias palabras de 16 bit dentro de 32 bits
  - M Unidad multiplicadora
- 16 bit x 16 bit con signo/sin signo empaquetado/desempaquetado
  - .L Unidad aritmético-lógica
  - Operaciones de comparación y lógicas (and, or, and xor)
- Saturación aritmética y cálculo del valor
  - .S Unidad de desplazamiento
  - Manipulación de Bit (set, get, shift, rotate) y ramificación
- Suma y suma empaquetada
  - .D Unidad de datos
  - Memoria de Carga/Almacenamiento
- Suma y Aritmética de puntero

3.3.1 Restricciones en el acceso a registros del C6000

- Acceso de las unidades funcionales a los archivos de registro
  - 1 (2) Unidades de camino de lectura/escritura en los registros A (B)
  - (1) 2 caminos de datos pueden leer un registro A (B) por ciclo de instrucción con una latencia de un ciclo
  - 2 accesos a memoria simultáneos no pueden usar los mismos archivos de registro del mismo archivo de registro como punteros direccionados
  - Límite de 4 lecturas de 32 bit por registro por ciclo de instrucción
- Almacenamiento de 40-bit largos in registros par/impar adyacentes

- Acumulación de precisión extendida de números de 32 bit
- Sólo un resultado de 40 bit puede ser escrito por ciclo
- Una lectura de 40 bit no puede producirse en el mismo ciclo que una escritura de 40 bit
- Rendimiento 4:1 en modo 40 bit

- Otras desventajas del C6000:

- No dispone de aceleración de ALU para la manipulación de la segmentación en bit
  - Computación, en C, al 50% en el decodificador MPEG-2 consumiendo decodificación de longitud variable en el C6200
  - Acceso directo a memoria en el C6400 controlando segmentado en bits (para video conferencia y estaciones base inalámbricas).
- Ramificado de las instrucciones deseables en la segmentación de cauce: Evitando el ramificado mediante el empleo de una ejecución condicional.
- No tiene protección hardware contra la segmentación casual: el programa y las herramientas deberán encargarse de ello.
- Debería emular muchas características de los DSP convencionales:
  - No tiene bucle hardware: usar segmentado de registros/condicionales.
  - No tiene acarreo inverso: usar un algoritmo rápido mediante Elster .
  - No tiene registro de estados: sólo el bit de saturación proporcionado por la unidad aritmético-lógica.

3.4 Métodos actuales de optimización código del DSP TMS320C6416

La CPU del DSP ejecuta una instrucción dada en un ciclo, pudiendo comenzar con la siguiente instrucción tras completar la previa. Cada unidad funcional de la CPU del DSP puede capturar y ejecutar una instrucción dada en un ciclo de instrucción, mientras que la

unidad funcional empieza el procesamiento de la siguiente instrucción tras la finalización de la previa.

El procesado de una instrucción no es una operación de un solo paso. Los tres pasos que emplea son:

- Obtener la instrucción de la memoria.
- Analizar la instrucción para saber que hay que hacer.
- Realizar la operación.

Cada etapa requiere su propia lógica para completar su funcionalidad. Para maximizar la realización, se debe emplear todo el soporte hardware disponible. Es para ello, que resulta útil el empleo de la segmentación de cauce para varias instrucciones.

En el DSP de la familia C6000, además de la gran arquitectura paralela, la segmentación de cauce está diseñada para funcionar de forma eficiente.

Como se puede observar en la Figura 3.10 [TEX11], con la segmentación de cauce se consigue reducir el tiempo efectivo de ejecución, llevándolo en este caso al punto de consumir 1 ciclo por instrucción gracias a su aplicación, sin añadir más unidades funcionales al sistema. Llegando en el caso mostrado a triplicar la eficiencia de la CPU.

	Clock Cycles								
CPU Type	1	2	3	4	5	6	7	8	9
Non-Pipelined	F1	D1	E1	F2	D2	E2	F3	D3	E3
Pipelined	F1	D1	E1						
		F2	D2	E2					
			F3	D3	E3				

Figura 3.10 Cronograma de la segmentación de cauce.

Para alcanzar una óptima ejecución de la segmentación de cauce, las distintas etapas son subdivididas para la arquitectura del C6000; tomando cada una de estas subetapas un ciclo para ser completada. Esta optimización es más evidente conforme mayor es el nivel de segmentación empleado, como se puede observar en la Figura 3.11 [TEX11].

Existen casos en los que la segmentación de cauce no es útil para mantener el óptimo modo de operación, como por ejemplo:

- La instrucción en curso depende del resultado de la previa, por lo que toma más de un ciclo para tener el resultado listo.

- La ejecución de un ramal del proceso, por lo que no se sabe cual es la siguiente instrucción a ejecutar hasta que no se ejecuta el salto a dicho ramal.

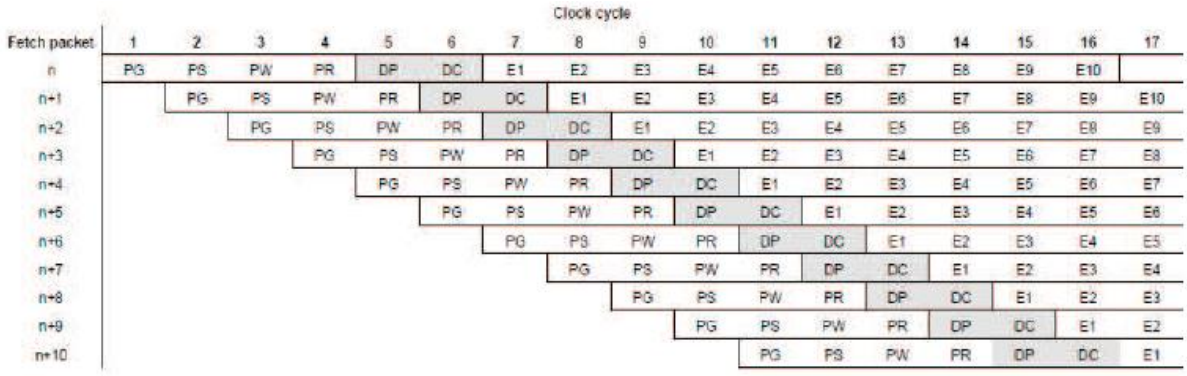


Figura 3.11 Cronograma de la segmentación de cauce de alto nivel para el DSP C6000.

En estos dos casos se produce un retardo debido a la espera a realizar. Los ciclos que la CPU ha de esperar son los retardos de las instrucciones. Por lo tanto, una carga tendrá cuatro retardos, una multiplicación en punto flotante de simple precisión tendrá tres y un ramal cinco. Estos retardos pueden provocar una degradación de la ejecución.

Para ocupar esos retardos y así optimizar el uso completo de la segmentación de cauce, se puede emplear una técnica software. La técnica de esquema software para rellenar u ocupar los retardos es el llamado software pipeline, mientras que la realización hardware se denomina buffer sploop (buffer del software pipeline).

El software pipeline es una técnica para organizar las instrucciones de un bucle con múltiples iteraciones que pueden ejecutarse en paralelo y la segmentación de cauce se puede emplear tanto como sea posible. Su efectividad se ve demostrada en la comparativa entre un esquema tradicional y otro mediante software pipeline de la Figura 3.12 [TEX11].

Solución 1: Esquema temporal tradicional: repetir el código siguiente 15 veces

```
Load *pointer_value++, data_value
add data_value, data_sum, data_sum
```

Solución 2: Software de pipeline

Nota 1: Para simplificar el ejemplo se omite las instrucciones de ramal.

Nota 2: Como la instrucción carga (load) tiene 4 retardos, la instrucción suma debería ser ejecutada en 5 ciclos.

Nota 3: Tras su correspondiente carga (si la carga tiene 10 retardos, una suma sería ejecutada 1 ciclo después de la carga).

La mejora se obtiene al cargar desde una iteración del bucle y sumar desde otra en paralelo, utilizándose la segmentación de cauce y consiguiendo una gran reducción del tiempo de CPU empleado.

Solution 1				Solution 2			
Clock	.D1	.L1		Clock	.D1	.L1	
1	Load1			1	Load1		
2				2	Load2		
3				3	Load3		Prolog
4				4	Load4		
5				5	Load5		
6		Add1		6	Load6	Add1	
7	Load2			7	Load7	Add2	
8				8	Load8	Add3	
9				9	Load9	Add4	
10				10	Load10	Add5	Kernel
11				11	Load11	Add6	
12		Add2		12	Load12	Add7	
13	Load3			13	Load13	Add8	
14				14	Load14	Add9	
15				15	Load15	Add10	
16				16		Add11	
17				17		Add12	
18		Add3		18		Add13	Epilog
19	Load4			19		Add14	
20				20		Add15	

Figura 3.12 Cronograma de la segmentación de cauce de alto nivel para el DSP C6000.

Respecto al buffer sploop cabe decir que está implementado en la familia C64x, entre otras. Este buffer almacena una instrucción simple del bucle en un buffer especializado, con las correctas dependencias del tiempo. Sobreescribe las instrucciones simples del bucle para así obtener una optimización del proceso. Este buffer no puede emplearse en bucles de más de 14 paquetes de ejecución (un paquete de ejecución consiste en hasta ocho instrucciones que pueden ser ejecutadas en paralelo). Para bucles complejos, su eficacia puede verse comprometida, por lo tanto se recomienda evitar estas situaciones.

Otra herramienta para la optimización es el óptimo uso del compilador, siguiendo básicamente tres métodos de optimización:

- **Opciones del compilador que optimizan el control.** Controlando como procesa el compilador todas las entradas.
- **Palabras clave.** Permiten calificar e indicar al compilador como tratar ciertas funciones y objetos.
- **Directivas *#pragma*.** Le indican al compilador como tratar las funciones específicas, objetos o sección de código.

Un elemento más a tener en cuenta en la optimización es el saber interpretar correctamente la información que nos proporciona el compilador a través de los datos que ofrece, como se muestra en el apéndice B de este trabajo. Esta parte es la más importante durante la optimización del código. Siendo el punto más crítico de la información que nos proporciona, el análisis del bucle.

El compilador trabaja mediante un proceso de tres etapas cuando optimiza un bucle:

- **Etapa 1:** Evalúa el bucle mediante el software pipeline. La mayor parte de los códigos no indican a priori el número de iteraciones, sino que se va determinando a lo largo del proceso. Para optimizar este proceso, el compilador necesita información del mínimo, el máximo y el máximo factor de iteración (mayor entero que puede dividir el número de iteraciones).
- **Etapa 2:** Reúne las fuentes del bucle y la información dependiente. Para tener una optimización mayor se ha de minimizar el intervalo de iteración (número de ciclos de CPU para completar una iteración). Este valor depende del bucle de dependencia forzada y la partición de recursos forzada.
    - **Bucle de dependencia forzada.** Este el que conlleva un mayor recorrido del bucle, lo cual ocurre cuando una iteración de un bucle produce un valor que será empleado por otro bucle.
    - **Partición de recursos forzada.** Es el máximo número de ciclos por unidad funcional empleado en una iteración simple, tras ser asignadas, todas las instrucciones a las unidades funcionales individuales. Para un bucle concreto, el número de veces que una unidad es empleada durante cada iteración también afecta. Si la información indica que una unidad funcional es mucho más utilizada

que otras, el programador puede considerar el uso de una expresión alternativa que emplee diferentes unidades funcionales.

- **Etapa 3:** Búsqueda de un esquema temporal del software pipeline del bucle. Se basa en el conocimiento de lo mejorado en las dos etapas anteriores. El compilador trata de optimizar el bucle encontrando el mínimo valor que se puede emplear en el esquema temporal del software pipeline del bucle. El compilador empieza con un valor que es igual de grande que los dos aspectos forzados indicados en la etapa anterior. Si, con ello, se puede crear el esquema temporal, la tarea está completada, si no, el compilador incrementa en 1 este valor y así hasta su obtención.

3.4.1 Optimización explícita del código

La optimización del compilador puede ser muy efectiva y ahorrar mucho tiempo, pero no es suficiente. Cuando el código depende de llamadas en un bucle, su implementación resulta compleja, o cuando la implementación de ciertas operaciones es complicada, se debe emplear tres recursos software del C6000:

- **Operaciones intrínsecas.** Ahorrando el uso de declaraciones, especificadas mediante marcadores, realizar operaciones incómodas o no expresables en C.
- **Bibliotecas optimizadas del DSP C6000.** Se pueden emplear para numerosas operaciones matemáticas, reduciendo el tiempo de desarrollo y optimización como ejemplo de como usar las operaciones intrínsecas para crear funciones muy optimizadas.
- **Funciones C en línea.** Cuando una función C en línea es llamada, el compilador la inserta automáticamente en el lugar de la llamada (expansión de la función en línea). Es muy beneficiosa, especialmente en funciones cortas por:
    - Ahorrar el gasto principal de la función llamada.
    - Optimizar la función de acuerdo al resto de código.
    - Optimizar el bucle.

Aunque el tamaño del código puede verse incrementado debido a la inserción directa de la función llamada, especialmente si la llamada a la función en línea se realiza en varios lugares, por lo que se ha de tratar de evitar esto.

3.4.2 Aplicando palabras clave restringidas

Las palabras clave restringidas son manuales y locales, aunque mejor alternativa a la opción global –mt del compilador.

Pueden ser usadas para informar al compilador que los punteros de una función dada no serán nunca referencias de un mismo buffer de memoria, evitando la sobrecarga innecesaria que el compilador puede tener. Pueden ser aplicados a punteros, teniendo en cuenta si es un parámetro, una variable local o un elemento de referencia de un objeto. Sólo es útil dentro de la función, pudiendo usarse cuando se necesite (al contrario de la opción –mt del compilador que tiene efecto en toda la aplicación). Como ejemplo de la aplicación de restricción de una palabra llave se puede observar el siguiente código:

```
void myFunction(int * restrict input1, struct *s) {
  int * restrict output;
  int * restrict input2 = s->pointer;
}
```

Como ejemplo de la aplicación de restricción de una palabra llave en el código de una suma simple se puede observar el siguiente código:

```
void simple_sum(short * restrict sum, short * restrict in1, unsigned
int N){
int i;
  for (i = 0; i < N; i++) {
    sum[i] = in1[i] + 1;
  }
}
```

Por lo general, si el compilador indica que el bucle implica una dependencia forzada más elevada de lo esperado, se ha de considerar la aplicación de restricción de una palabra.

3.4.2.1 Directiva *#pragma Must_iterate*

La directiva #pragma MUST_ITERATE le especifica al compilador ciertas propiedades del contador de bucle, teniendo éste la siguiente sintaxis, y debiendo ser insertado inmediatamente encima del código del bucle.

#pragma MUST_ITERATE(lower_bound, upper_bound, factor)

El *lower_bound* define el mínimo posible del total de iteraciones del bucle, el *upper_bound* define su máximo, y el factor, le indica al compilador que la iteración total es un entero múltiplo de este número. El *lower_bound* y el *upper_bound* debe ser divisible y al menos tan grande como el factor. Se puede omitir alguno de estos tres parámetros si no se conoce, pero si alguno de sus valores es conocido, es recomendable proporcionar esta información al compilador.

#pragma MUST_ITERATE(10, , 2)

El *lower_bound* y el *upper_bound* puede resultar extremadamente útil para el compilador durante la optimización.

3.4.2.2 Deshacer el bucle y deshacer pragma

En muchos casos, la partición de recursos o la utilización de unidades funcionales se convierten en no equilibradas, como se puede observar en el siguiente código de bucle con partición de recursos no equilibrada:

```
   void Loop(int * restrict output, int * restrict input1, int * restrict
input2, int n){
   int i;
     for (i=0; i<n; i++) {
       output[i] = input1[i] + input2[i];
     }
   }
   //An excerpt from its compiler feedback
```

```
;* Partitioned Resource Bound(*) : 2
;* Resource Partition:
;* A-side B-side
;* .L units 0 0
;* .S units 0 1
;* .D units 2* 1
;* .M units 0 0
```

3.4.2.3 Deshacer manualmente un bucle

El deshacer un bucle se puede realizar manualmente. El número de iteraciones es reducido a la mitad, pudiendo llegar, en teoría, a una reducción del 25% en la cuenta de ciclos global.

Este proceso puede ser tedioso para la mayoría de los bucles. Generalmente se recomienda que se utilicen el compìlador y directivas pragmas del C6000 al deshacer los bucles.

3.4.2.4 Deshacer el compilador

La directiva *#pragma must_iterate* le proporciona al compilador la flexibilidad para deshacer automáticamente cuando sea posible, así mismo, el compilador puede, normalmente, ejecutarse en este modo de trabajo.

Cuando el código es más complejo esta operación incrementa su difiultad. La directiva *#pragma UNROLL* puede emplearse para indicar al compilador cómo quiere el programador que se deshaga el bucle, por lo que ha de usarse justo antes de que el bucle trate de deshacerse:

#pragma UNROLL(factor)

Antes de usar *#pragma UNROLL* hay que asegurarse que la cuenta del bucle es divisible entre el factor, como se puede observar en el siguiente listado usando la directiva *#pragma UNROLL*:

```
void Loop(int * restrict output, int * restrict input1, int * restrict
input2, int n){
int i;
#pragma MUST_ITERATE(lower_bound, ,2)
```

```
#pragma UNROLL(2)
  for (i=0; i<n; i++){
// n & lower_bound >= the minimum safe trip count, and divisible by 2
    output[i] = input1[i] + input2[i];
  }
}
```

En general, si la porción de la partición de los recursos del análisis del compilador muestra un desequilibrio significativo al emplear la directiva *#pragma UNROLL*, se recomienda que se pruebe con un valor diferente y recoger el factor que proporcione el mejor resultado. Hay que tener presente que el deshacer los bucles conlleva un incremento en el tamaño del bucle, si el tamaño del bucle es demasiado grande, bloquea la posibilidad del empleo del bufer SPLOOP y por lo tanto, si aparece cualquiera de los siguientes cuatro mensajes en la información del compilador hay que considerar la posibilidad de dividirlo en trozos menores o reducir la complejidad del bucle para que se pueda emplear la segmentación de cauce:

- Bucle descalificado: demasiadas instrucciones.
- No se pueden localizar los registros de la máquina.
- Contador de ciclos demasiado grande. No realizable.
- No se encuentra el esquema temporal.

3.4.2.5 Deshaciendo bucle y SIMD

Otra ventaja que presenta el deshacer el bucle es que proporciona al compilador más oportunidades para realizar instrucciones SIMD (Simple Instruction Multiple Data). Para instrucciones simples con datos múltiples, las cuales describen el estado cuando el núcleo del DSP puede operar con datos múltiples durante cada operación soportada. Esto puede ser muy beneficioso especialmente durante la carga y almacenamiento de datos.

3.4.2.6 Habilitando instrucciones SIMD

Al deshacer los bucles se da la posibilidad de que instrucciones SIMD puedan ser usadas. Sin embargo, si el tamaño de los datos es demasiado grande, su efectividad se ve limitada.

Dos de las más usadas son las instrucciones LDDW y STDW, que realizan la carga y el almacenamiento de 64 bits. Estas dos instrucciones requieren que los datos sean alineados en grupos de 8 bytes, en caso contrario se deberían usar las instrucciones LDNDW y STNDW.

3.4.2.7 Usando operaciones intrínsecas y bibliotecas optimizadas

Las operaciones intrínsecas pueden ayudar notablemente en la optimización del código así como al programador para invocar directamente las operaciones en ensamblador del C6000, las cuales presentan dificultad para ser expresadas en C. El DSP C6000 soporta numerosas operaciones intrínsecas, las cuales se pueden agrupar como se muestran aquí:

- Soportan acceso rápido a memoria: _amemd8_const.
- Soportan operaciones con datos rápidas: _mpy4.
- Soportan operaciones no estándar incómodas de escribir en C: _norm; _shfl.
- Nassert: una operación intrínseca especial que no genera código. Aportando al compilador información acerca de que optimizaciones pueden ser válidas: void _nassert(int).

Las bibliotecas TI del DSP no son siempre fáciles de implementar en una función o algoritmo usando operaciones intrínsecas. Para muchas funciones matemáticas estándar y de procesado digital de señales, tales como funciones de procesado de audio y video el fabricante proporciona unas bibliotecas de instrucciones muy optimizadas para la familia de DSP C6000. Estas bibliotecas pueden emplearse para reducir y optimizar el tiempo o como ejemplo de marco para usar operaciones intrínsecas para crear funciones muy optimizadas. Para muchas de las bibliotecas se permite su modificación y ajuste al caso concreto en que se va a aplicar.

Capítulo 4

Resumen:

En este capítulo se muestran nociones de la separación ciega de fuentes y su resolución mediante ICA. Finalmente se describe el algoritmo FastICA que se utilizará posteriormente en este trabajo.

4 SEPARACIÓN CIEGA DE FUENTES

Antes de comenzar a hablar de los algoritmos ICA, se muestra el método geométrico de separación ciega de fuentes para pasar luego al análisis y revisión del algoritmo ICA y algunas de sus versiones que han sido estudiadas actualmente. Para, finalmente, terminar con el más representativo y eficiente, el FastICA.

4.1 Separación ciega de fuentes mediante métodos geométricos

Para abordar el estudio de los métodos geométricos hay que comenzar recordando que el modelo de mezcla que se va a utilizar es el lineal instantáneo y que además de las hipótesis usuales para la separación ciega de fuentes (número de fuentes igual al número de sensores, matriz de mezcla regular), en los métodos geométricos se tienen las siguientes:

- No se parte de la hipótesis de independencia estadística de las fuentes.
- Se considera que las fuentes están acotadas (las señales físicas normalmente lo están, fundamentalmente en amplitud y tienen energía finita).

El modelo de mezcla lineal es de la forma: $e(t)=As(t)$, donde $e,s \in \Re^p$, $A \in \Re^{pxp}$, por lo que se debe obtener $s(t)=A^{-1}e(t)$, verificándose que $y(t)=W^{-1}As(t)$, debiendo tener en cuenta las siguientes propiedades analíticas y geométricas:

- Al estar las fuentes acotadas los valores generados por las mismas estarán circunscritos en un hiperparalelepípedo rectangular (en el caso de 2 fuentes será un rectángulo).
- Las imágenes de las fuentes forman un hiperparalelepípedo**,** y los vértices imágenes no han de estar necesariamente en el mismo orden absoluto o relativo que los vértices fuente.

Este algoritmo [PRI99c], a partir de los vectores de observación, estima uno de los vértices del hiperparalelepípedo, que se toma como origen, y un conjunto de p vectores, cada uno de ellos perteneciente a una arista distinta de las que inciden en dicho vértice. Esta estimación se efectúa considerando que el conjunto de p vectores buscados, satisfacen la condición de maximizar su separación angular. El algoritmo se implementa con dos redes neuronales artificiales con pesos comunes, realizando un proceso reiterativo para recuperar las señales originales, y con aprendizaje competitivo para obtener los valores de los pesos.

Este algoritmo se basa en la utilización de dos redes neuronales artificiales, cuyos pesos son comunes y estiman los coeficientes de la mezcla. La red de obtención de las fuentes es lineal y recursiva, mientras que la red de aprendizaje es competitiva, con *p* neuronas también lineales y otra de salida no lineal (función escalón) modificándose durante el aprendizaje tanto los pesos como el umbral de la función escalón. El aprendizaje y la separación se llevan a cabo muestra a muestra.

El algoritmo converge más rápidamente conforme más uniformes sean las funciones de distribución de las fuentes originales, debido a su mayor probabilidad de obtener puntos en los ejes del hiperparalelepípedo del espacio de observaciones.

Este método considera que las fuentes están acotadas, y el espacio de observaciones formando un hiperparalelepípedo con un vértice en el origen. Estimando la matriz de mezcla mediante la obtención de los p vectores (vectores arista) situados cada uno de ellos en uno de los ejes del cono que contiene el hiperparalelepípedo de observaciones. Además, se considera que un conjunto de p vectores, con uno de sus extremos en el vértice del cono y con separación angular máxima constituye un conjunto de vectores arista.

Como se conocen las componentes de dos vectores de mezcla, *e(u)* y *e(v),* se puede calcular el coseno del ángulo que forman, o proximidad angular, γ_{uv}, a partir de su producto escalar. O sea, se puede evaluar la siguiente expresión:

$$\gamma_{\boldsymbol{uv}} = \boldsymbol{cos}[\boldsymbol{e}(\boldsymbol{u}), \boldsymbol{e}(\boldsymbol{v})] = \frac{\sum_{j=1}^{\boldsymbol{p}} \boldsymbol{e}(\boldsymbol{u})\cdot\boldsymbol{e}(\boldsymbol{v})}{|\boldsymbol{e}(\boldsymbol{u})||\boldsymbol{e}(\boldsymbol{v})|} = \frac{\boldsymbol{e}(\boldsymbol{u})\cdot\boldsymbol{e}(\boldsymbol{v})}{|\boldsymbol{e}(\boldsymbol{u})||\boldsymbol{e}(\boldsymbol{v})|} \quad (4.1)$$

Por lo que, la proximidad angular de dos vectores es máxima, +1, cuando los dos vectores coinciden (ángulo de 0º) y es mínima, -1, cuando la separación sea máxima (ángulo de 180º).

Por último, hay que indicar que se seguiría el mismo procedimiento si en lugar de dos vectores, es un conjunto de ellos. El proceso iterativo que se propone como método de separación considera los *p* primeros vectores observados y forma con ellos un supuesto conjunto, C_w, de vectores arista, y con ellos calcula su proximidad angular, $\gamma_{\mathbf{w}}$. Cuando llega un nuevo vector, e, dicho vector se añade al conjunto, formando ahora un conjunto de test, C_t, que contiene *p+1* vectores. El subconjunto *C* de C_t, de *p* vectores, que haga mínimo $\gamma_{\mathbf{C}}$, se toma como nuevo hipotético conjunto de vectores arista, C_w. Así, al final del proceso iterativo se tendrá un conjunto de p vectores cuya proximidad será mínima (separación angular máxima) y que corresponderá a los vectores arista del hiperparalelepípedo de observación.

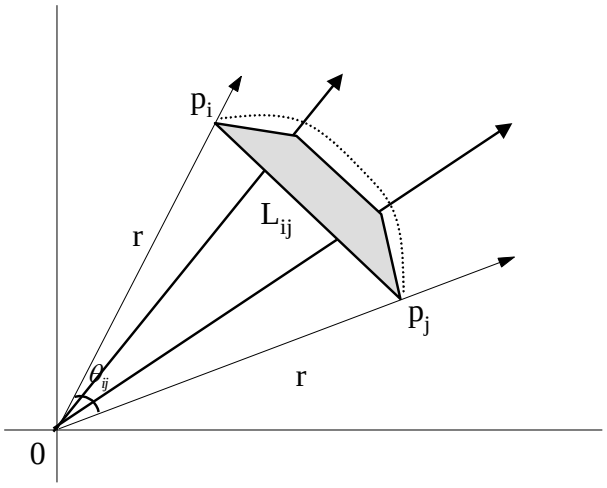


Figura 4.1 Hiperpolígono del espacio de observación.

A continuación, hay que desplazar el hiperparalelepípedo. Desde un punto de vista geométrico y teniendo en cuenta la linealidad de la matriz A, que el caso más general, $s_i(t)\in[s_{im}, s_{iM}]$ con $s_{im}, s_{iM}\in\Re$, $s_{im}<s_{iM}$, equivale a la situación anterior después de efectuar

un cambio de coordenadas con un vector ***v***, que representa un vector formado por las coordenadas de uno de los vértices del hiperparalelepípedo. Este cambio de coordenadas es equivalente a trasladar el hiperparalelepípedo de forma que el vértice v se sitúe en el origen de coordenadas. En el caso de *p=3*, se tendría que hacer el cambio indicado en la Figura 4.3. Para llevar a cabo dicha traslación, el algoritmo tiene que detectar el vector *v*. Para ello tiene en cuenta que dicho vector será el vector del espacio de observaciones cuyas componentes sean las coordenadas de cualquiera de los vértices.

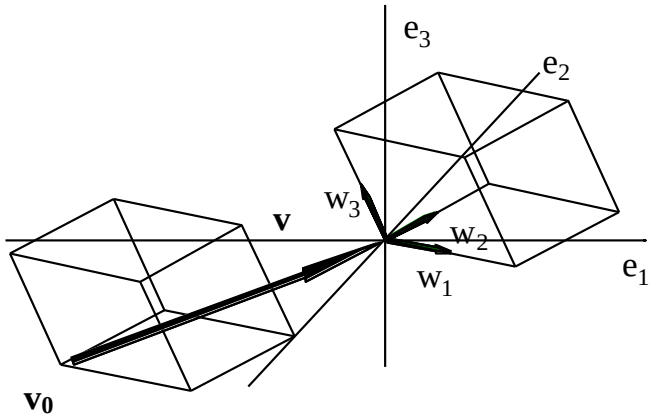


Figura 4.2 Traslación del hiperparalelepípedo.

En el caso de tres fuentes, se tendrían que detectar uno de los 8 vértices del hiperparalelepípedo, pudiéndose utilizar como referencia.

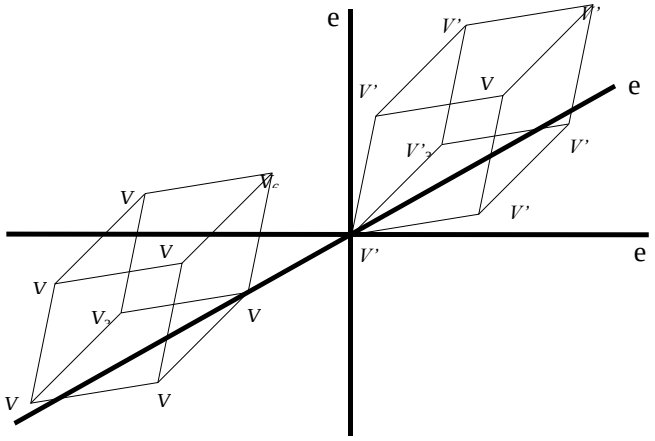


Figura 4.3Traslación en un caso p=3.

Para identificar el medio y llevar a cabo la separación es necesario determinar:

- Cualquier vértice, $\boldsymbol{v}_c$, del hiperparalelepípedo para poder realizar el cambio de coordenadas.
- Cualesquiera *p* vectores, cada uno de ellos de una de las p aristas que convergerán en el vértice de traslación.

Estas condiciones implican que tienen que tienen que generarse ciertas combinaciones de las señales originales, de forma que sin ellas no se podría identificar el medio. Como los vértices del espacio de observaciones se corresponden con los vértices del espacio de las fuentes, para obtener un vértice del hiperparalelepípedo de observación, las fuentes tienen que generar un vector tal que todas sus componentes tengan valores extremos, que se correspondan con sus cotas. El conjunto formado por estos vectores más el vértice se denominará de aquí en adelante conjunto de vectores críticos. Las condiciones para obtener los vectores arista basta con encontrar un sólo vector en cada una de las aristas. Para *p* fuentes, habrá *p+1* vectores críticos. Hay que tener en cuenta que puede haber dependencias estadísticas entre las fuentes, que impidan la generación de los vectores críticos, en cuyo caso el procedimiento propuesto no es válido. Sin embargo, hay situaciones con dependencias estadísticas, en las que se satisfacen todos los requisitos y por tanto no se puede afirmar que el procedimiento no sea válido, en general, cuando hay dependencia estadística entre las fuentes.

Respecto a la implementación del algoritmo, la separación de señales se realiza, básicamente, mediante las fases de preprocesado, aprendizaje y separación. En el preprocesado, además de la obtención adaptativa del vértice y la normalización de los vectores de mezcla, se debe reducir el efecto que produce la incertidumbre derivada del desconocimiento exacto de los vértices y del posible ruido al obtener las muestras. La detección de un vértice distinto al correcto provoca cierto error en las coordenadas de los vértices obtenidos, y por lo tanto, de los elementos de las columnas de la matriz W. Siendo menor dicho error conforme mayor sea el módulo del vector seleccionado como vector peso. Lo mismo ocurre respecto al ruido de la mezcla. Si se supone que dicho ruido es aditivo y está acotado, el vector erróneo se desplazará dentro de un intervalo. Dicho error también afectará al vértice v_0, el cual se verá desplazado.

Si el ruido es acotado y uniforme (es decir, afecta de igual forma a todos los puntos del medio) y se detectan como valores extremos el vértice y el vector arista, el ruido tiende a

ser compensado, pero es necesario más tiempo para localizar los valores extremos en presencia de ruido.

El efecto del ruido disminuye cuando la norma de *w(i)* con respecto a n_i - n_0 aumenta. En la implementación, además del vértice de cambio de coordenadas, v_0 se obtiene también el vértice opuesto, v_1. Para considerar un vector de observación como un candidato a vector arista, debe tener una norma del orden de:

$$\boldsymbol{n_r = \frac{|v_1 v_0|}{2}} \qquad \textbf{(4.2)}$$

Como los vértices se van obteniendo adaptativamente, al principio del proceso los puntos están muy juntos. Para compensar este efecto sólo se consideran aquellos vectores de observación, *e(t),* cuyas normas sean mayores de:

$$\boldsymbol{n_{test} = (n_0 - n_r) \cdot e^{-\frac{t}{\tau}} + n_r} \qquad \textbf{(4.3)}$$

Donde n_0 y τ representan la norma inicial del test y una constante de tiempo, respectivamente.

En la fase de aprendizaje se realizan las siguientes operaciones:

- Normalización de los vectores de mezcla.
- Obtención y actualización de los pesos a partir del cálculo de las proximidades angulares.

4.2 Algoritmos ICA

Para comenzar a estudiar el algoritmo ICA lo mejor es empezar mediante un ejemplo básico de problema a solventar con este algoritmo. Considerando que se está en una habitación en la que hay dos personas hablando simultáneamente. Se tienen dos micrófonos, los cuales colocaremos en diferentes lugares. Los micrófonos proporcionan dos grabaciones de dos señales en el tiempo, $x_1(t)$ y $x_2(t)$, con x_1 y x_2 las amplitudes, y t el parámetro del tiempo. Cada una de estas señales grabadas son una suma de las señales emitidas por las dos personas y, se nombrarán $s_1(t)$ y $s_2(t)$. Esta situación se puede expresar en forma de ecuación lineal donde:

$$\boldsymbol{x_1(t) = a_{11}s_1 + a_{12}s_2} \qquad \boldsymbol{x_2(t) = a_{21}s_1 + a_{22}s_2} \qquad (4.4)$$

Donde a_{11},a_{12},a_{21} y a_{22} son algunos parámetros que dependen de las distancias de los micrófonos a las dos personas. Sería muy útil si se pudiera estimar las dos señales originales $s_1(t)$ y $s_2(t)$, empleando sólo las dos señales grabadas $x_1(t)$ y $x_2(t)$. A esto se le conoce como el problema del *cocktail-party*. Por el momento, se omite cualquier retardo u otros factores extra para nuestro modelo de mezcla simplificado.

Si se consideran, a modo de ejemplo, las formas de onda de las Figuras 4..4 y 4.5.**¡Error! No se encuentra el origen de la referencia.** Las señales originales habladas podrían verse como algo parecido a las mostradas en la Figura 4.4 y, la mezcla de señales como las de la Figura 4.5. El problema consiste en recuperar los datos de la Figura 4.4 usando sólo los de la Figura 4.5.

En realidad, si los parámetros a_{ij} fueran conocidos, se podría solucionar el sistema de ecuaciones anteriores mediante el empleo de métodos clásicos. Sin embargo, si no se conocen $a_{ij,}$ el problema se complica considerablemente.

Para resolver este problema se puede utilizar alguna información sobre las propiedades estadísticas de las señales $s_i(t)$ para estimar la a_{ii}. En realidad, basta considerar que $s_1(t)$ y $s_2(t)$, en cada instante de tiempo t, son estadísticamente independientes (lo cual es realista en muchos casos). El desarrollo de la técnica del Análisis en Componentes Independientes, ICA, puede ser usado para estimar a_{ij} tomando como base la información de su independencia, la cual permite separar las dos señales originales $s_1(t)$ y $s_2(t)$ a partir de sus mezclas $x_1(t)$ y $x_2(t)$. La Figura 4.6 muestra las dos señales estimadas por el método ICA, las cuales resultan muy aproximadas a las originales (en este caso con inversión de signo, pero esto no es significativo).

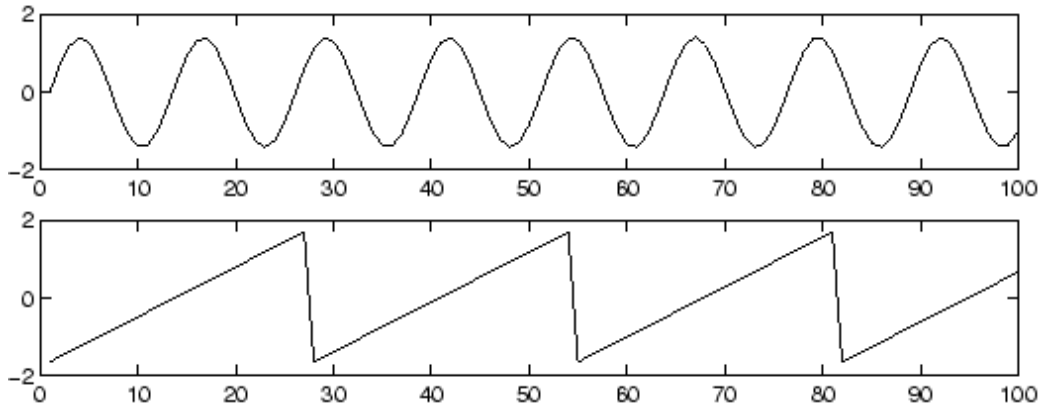


Figura 4.4 Señales originales.

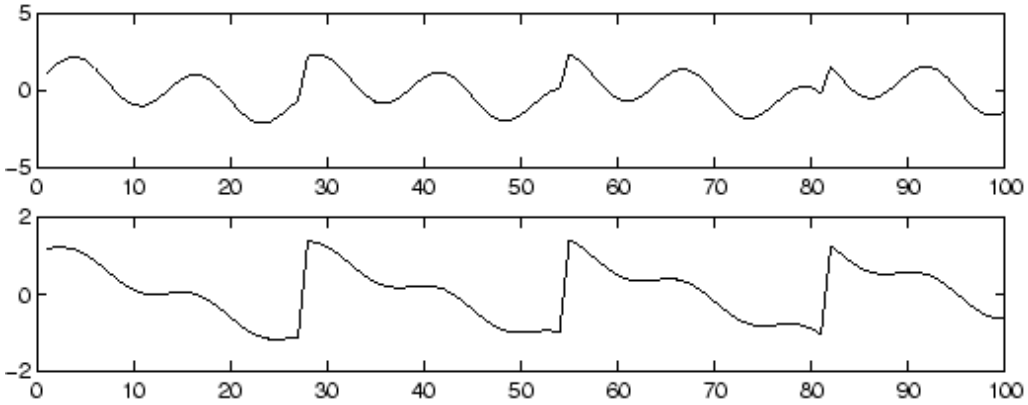


Figura 4.5 Las mezclas observadas de las señales de la figura 4.4.

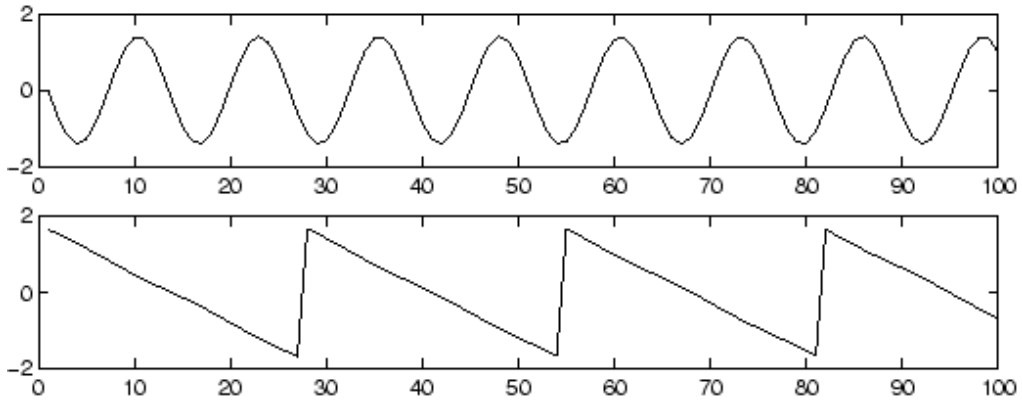


Figura 4.6 Estimaciones de las señales originales teniendo en cuenta sólo las señales observadas.

El análisis en componentes independientes se desarrolló originalmente para tratar los problemas que están estrechamente relacionados con el problema del cocktail-party. Dado el reciente aumento del interés en el ICA, ha quedado claro que este principio tiene también otras muchas aplicaciones interesantes. Por ejemplo, si se consideran los registros eléctricos de la actividad cerebral como los dados por un electroencefalograma (EEG). El EEG consiste en grabaciones de datos de potenciales eléctricos en muchos lugares diferentes en el cuero cabelludo. Estos potenciales son supuestamente generados por la mezcla de algunos componentes subyacentes de la actividad cerebral. Esta situación es bastante similar al problema del cocktail-party: tratando de encontrar las componentes originales de la actividad cerebral, aunque sólo es posible observar las mezclas de las componentes. ICA puede revelar información interesante sobre la actividad cerebral, dando acceso a sus componentes independientes.

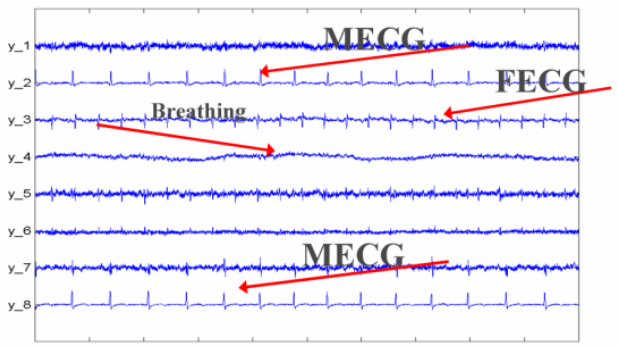
Otra, muy distinta aplicación de la ICA es en la extracción de características. Un problema fundamental en el procesamiento digital de señales es encontrar representaciones correctas de video, audio u otro tipo de datos para tareas como la compresión y la eliminación del ruido. Las representaciones de datos a menudo se basan en transformaciones lineales (discretas). Las transformaciones lineales estándar ampliamente empleadas en el procesamiento de imágenes son las de Fourier, Haar, coseno, etc.

Las técnicas basadas en ICA permiten obtener buenas aproximaciones al problema incluso para señales con varias componentes. A título ilustrativo, una aplicación interesante es el análisis de señales ECG (Electrocardiografía) fetales. Debido a la naturaleza de dichas señales y a la forma de capturarlas presentan un problema claro de señales que se han mezclado y que interesa extraer las señales originales. La Figura 4.7 muestra un conjunto de señales obtenidas en un ECG donde las señales de la madre y el feto están mezcladas.

Figura 4.7 Señales obtenidas originalmente en un ECG fetal.

En la Figura 4.8 puede observarse claramente las diferencias entre en el análisis de componentes principales y el análisis de componentes independientes.

(A) PCA



(B) ICA

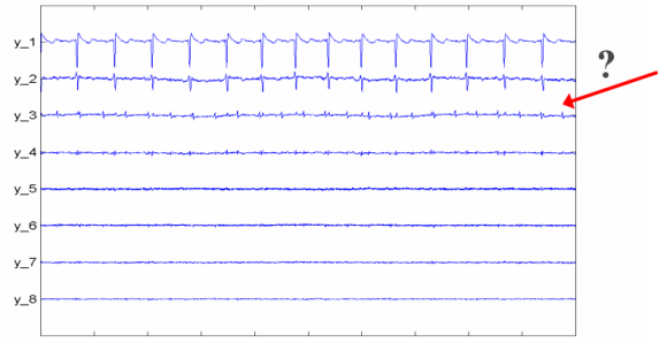


Figura 4.8 Diferentes señales obtenidas aplicando PCA e ICA.

Como puede comprobarse en este caso, el análisis de componentes principales apenas ofrece información interesante ya que la tercera de las señales no se sabe exactamente con qué corresponde, mientras que el análisis de componentes independientes sí muestra información clara de las fuentes.

4.2.1 Definición de ICA

Para definir rigurosamente ICA, se puede emplear el modelo estadístico de variables latentes. Asumiendo que se observan n mezclas lineares $x_1,...,x_n$ de n componentes independientes:

$$\boldsymbol{x_j = a_{j1}s_1 + a_{j2}s_2 + .. + a_{jn}s_n, \forall j} \qquad (4.5)$$

Estando en función del tiempo, *t*; en el modelo de ICA, se supone que cada mezcla, x_j, y cada componente independiente, s_j, es una variable aleatoria, en lugar de una señal en tiempo. Los valores observados $x_j(t)$ serían las señales de micrófono en el problema del cocktail-party. Tanto las variables mezcladas como las componentes independientes tienen de media cero: si esto no es cierto, entonces las variables observables, x_j, siempre pueden ser centradas restándoles la media de la muestra, lo cual hace que el modelo tenga media cero.

Es conveniente utilizar la notación de matriz-vector en lugar de las cantidades como en la anterior ecuación. Si se llama $\boldsymbol{x}$, al vector aleatorio cuyos elementos son las mezclas x_1, ..., x_n; $\boldsymbol{s}$, al vector aleatorio con elementos s_1, ... , s_n y $\boldsymbol{A}$, a la matriz con los elementos a_{ij}.

(En minúscula se designarán los vectores, mientras que en mayúsculas lo harán las matrices). Todos los vectores son considerados como matrices columna; así el transpuesto de **X**, $\mathbf{X}^{\mathbf{T}}$, es una matriz fila. Usando esta notación vector-matriz, la mezcla del modelo anterior se escribe como:

$$\boldsymbol{x} = \boldsymbol{A}\,\boldsymbol{s} \tag{4.6}$$

En ocasiones se precisa la matriz de columnas, **A**; denominando como $\mathbf{a_j}$ al modelo:

$$\boldsymbol{x} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{a_i s_i} \tag{4.7}$$

El modelo estadístico de esta ecuación se denomina análisis en componentes independientes, o modelo ICA. El ICA es un modelo generativo, lo que significa que describe cómo los datos observados son generados por un proceso de mezclado de los componentes s_i. Los componentes independientes son variables latentes, lo que significa que no puede observarse directamente. Asimismo, la matriz de mezcla es considerada como desconocida. Lo que se observa es el vector aleatorio *x* y, debiendo estimar tanto *A* como *s*.

El punto de partida para el ICA es la hipótesis de las componentes s_i son estadísticamente independientes y que las componentes independientes deben tener distribuciones no-gaussianas. Sin embargo, en el modelo básico no se toman como distribuciones conocidas (si son conocidas, el problema se simplifica considerablemente). Para simplificar, también estamos suponiendo que la matriz de mezcla desconocida es cuadrada. Después de estimar la matriz *A*, se puede calcular su inversa, denominándola *W*, y obtener la componente independiente haciendo simplemente:

$$\boldsymbol{s} = \boldsymbol{W}\,\boldsymbol{x} \tag{4.8}$$

4.2.2 Ambigüedades de ICA

En el modelo ICA de la ecuación (4.7), es fácil ver que las siguientes ambigüedades son consideradas:

- No es posible determinar las diferencias de las componentes independientes, ya que *s* y *A* son desconocidos. Cualquier multiplicador escalar en una de las fuentes, $s_{i,}$ siempre puede ser cancelado mediante la división de la correspondiente columna a_i de *A* entre el mismo escalar. Como consecuencia de ello, se pueden fijar las magnitudes de las componentes independientes, ya que son variables aleatorias, la forma más natural de hacerlo es suponer que cada una tiene una varianza unitaria: $E\{s_i^2\} = 1$. Entonces la matriz *A* se adaptará en la solución del método ICA. Hay que tener en cuenta que esto todavía mantiene la ambigüedad del signo: se podría multiplicar el elemento independiente, a_n, por -1 sin afectar el modelo. Esta ambigüedad es, afortunadamente, insignificante en la mayoría de las aplicaciones.
- No es posible determinar el orden de las componentes independientes, por ser ambas, *s* y *A* desconocidas. Se puede cambiar el orden de los términos en la suma de la ecuación (4.7) y, llamar a cualquiera de las componentes independientes de la primera. Formalmente, una permutación de matriz *P* y su inversa pueden ser sustituidas en el modelo para dar $x = AP^{-1}P_s$. Los elementos de P_s son variables originariamente independientes s_j, pero en otro orden. La matriz AP^{-1} no es más que una nueva matriz mezcla desconocida, resoluble mediante los algoritmos ICA.

4.2.3 Ilustración de ICA

Para ilustrar el modelo ICA en términos estadísticos, consideremos dos componentes independientes con las siguientes distribuciones uniformes:

$$\boldsymbol{p(s_i)} = \begin{cases} \frac{\mathbf{1}}{\mathbf{2\sqrt{3}}} & \boldsymbol{si\ |s_i| \le \sqrt{3}} \\ \mathbf{0} & \boldsymbol{otro\ caso} \end{cases} \qquad (4.9)$$

El rango de valores para esta distribución uniforme es elegido para que la media sea cero y la varianza igual a uno. La densidad conjunta de s_1 y s_2 es entonces uniforme en un cuadrado. Esto se deduce de la definición básica de que la densidad conjunta de dos variables independientes de sus densidades marginales; simplemente hay que calcular el producto. La densidad conjunta se muestra en la Figura 4.9, mostrando los puntos de datos aleatorios procedentes de esta distribución.

Ahora se van a mezclar estas dos componentes independientes. Tomemos la siguiente matriz de mezcla:

$$\boldsymbol{A_0} = \begin{pmatrix} \mathbf{2} & \mathbf{3} \\ \mathbf{2} & \mathbf{1} \end{pmatrix} \tag{4.10}$$

Esto da dos variables mezcladas, x_1 y x_2. Siendo fácilmente calculable que el dato mezclado tiene una distribución uniforme formando un paralelogramo, como se muestra en la Figura 4.9. Notar que las variables aleatorias x_1 y x_2 no son independientes. Esto se puede ver cuando es posible predecir el valor de una de ellas, por ejemplo x_2, desde el valor de los otros. Es evidente que si x_1 alcanza uno de sus valores máximos o mínimos, entonces esto determina completamente el valor de x_2. Por lo tanto, no son independientes. (Para las variables s_1 y s_2, la situación es diferente: a partir de la Figura 4.9 [HYV00] puede verse que conociendo el valor de s_1 no tenemos modo alguno de conocer el valor de s_2).

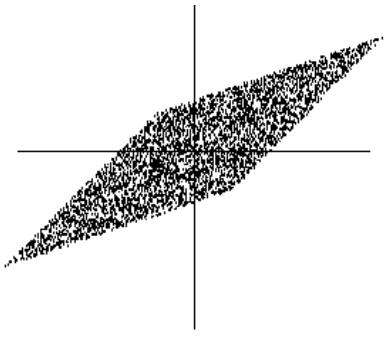


Figura 4.9 La distribución conjunta de las componentes independientes s1 y s2 con distribuciones uniformes. En el eje horizontal s1 y en el vertical s2.

El problema de la estimación del modelo de datos de ICA es encuentra ahora en cómo estimar la matriz de mezcla A_0 utilizando sólo la información contenida en las mezclas x_1 y x_2. En realidad una manera intuitiva de estimación A: los ejes del paralelogramo están en las direcciones de las columnas de A. Esto significa que se podría, en principio, estimar el modelo ICA mediante la primera estimación de la densidad conjunta de x_1 y x_2, y a continuación localizar los ejes. Pareciendo tener solución el problema.

Este método sería muy pobre si sólo funcionara con variables que tienen exactamente distribución uniforme. Además, resultaría ser un método complicado computacionalmente. Cuando lo que realmente se necesita es un método fiable y que sirva para cualquier distribución de componentes independientes.

A continuación se tendrá en cuenta la definición exacta de independencia antes de empezar a desarrollar métodos para la estimación del modelo ICA.

4.2.4 Definición y propiedades fundamentales

Para definir el concepto de independencia, se considera la posibilidad de escalar dos variables aleatorias con valores de y_1 e y_2. Básicamente, las variables y_1 y y_2 se dice que son independientes si la información sobre el valor de y_1 no da ninguna información sobre el valor de y_2, y viceversa. Anteriormente se señaló que este es el caso de las variables s_1, s_2, pero no con la mezcla de variables x_1, x_2.

Técnicamente, la independencia puede ser definida por la densidad de probabilidad. Vamos a denotar por $p(y_1,y_2)$ la función de densidad de probabilidad conjunta (pdf) de y_1 *e* y_2, nombrando como $p_1(y_1)$ al pdf de y_1, cuando se considera por sí solo:

$$\boldsymbol{p_1(y_1) = \int p(y_1, y_2) dy_2} \tag{4.11}$$

E igualmente para y_2. Pudiendo definir que y_1 e y_2 son independientes si y sólo si el pdf conjunto es factorizable de la siguiente forma:

$$\boldsymbol{p(y_1, y_2) = p_1(y_1) p_2(y_2)} \tag{4.12}$$

Esta definición se extiende para cualquier número n de variables aleatorias, en cuyo caso la densidad de probabilidad conjunta debería ser el producto de n términos.

Esta definición puede ser empleada para conducirnos a una propiedad más importante de variables aleatorias independientes, dando dos funciones, h_1 y h_2, siempre se tendrá:

$$\boldsymbol{E\{h_1(y_1)h_2(y_2)\} = E\{h_1(y_1)\}E\{h_2(y_2)\}} \quad (4.13)$$

4.2.5 Las variables no correlacionadas son sólo parcialmente independientes

Una forma más débil de independencia es la no-correlación. Dos variables aleatorias y_1 e y_2 se dicen ser no-correlacionadas si su covarianza es cero:

$$\boldsymbol{E\{y_1y_2\} - E\{y_1\}E\{y_2\} = 0} \quad (4.14)$$

Si las variables son independientes, son no-correlacionadas, como se observa en la ecuación (4.11), tomando $h_1(y_1)=y_1$ y $h_2(y_2)=y_2$.

Por otro lado, la no-correlación no implica independencia. Por ejemplo, se considera que (y_1,y_2) son valores discretos y seguidos tal como la distribución que la pareja es con probabilidad 1/4 igual a cualquiera de los siguientes valores: (0,1),(0,-1),(1,0),(-1,0). Entonces, y_1 e y_2 son no-correlacionados, como puede ser fácilmente calculado. Por otro lado se tiene que:

$$\boldsymbol{E\{y_1^2y_2^2\} = 0 \neq \frac{1}{4} = E\{y_1^2\}E\{y_2^2\}} \quad (4.15)$$

Como la condición en la ecuación (4.15) no se cumple y las variables no pueden ser independientes. La independencia implica no-correlación, muchos métodos ICA limitan el procedimiento de estimación de modo que siempre da estimaciones no-correlacionadas de

las componentes independientes. Esto reduce el número de parámetros libres, y simplifica el problema.

La principal restricción en ICA es que las componentes independientes deberían ser no-gaussianas. Para ver por qué las variables gaussianas hacen imposible ICA, hay que tener en cuenta que la matriz mezcla es ortogonal y los s_i son gaussianas. Entonces x_1 y x_2 son gaussianas, no-correlacionadas y, de varianza unidad. Su densidad de probabilidad conjunta es dada por:

$$\boldsymbol{p(x_1, x_2) = \frac{1}{2\pi} exp\left(-\frac{x_1^2+x_2^2}{2}\right)} \tag{4.16}$$

Esta distribución se ilustra en la Figura 4.10 [HYV00]. Esta figura muestra que la densidad es completamente simétrica. Por lo tanto, no contiene ninguna información en las direcciones de las columnas de la matriz *A* y, por lo tanto, A no puede ser estimada.

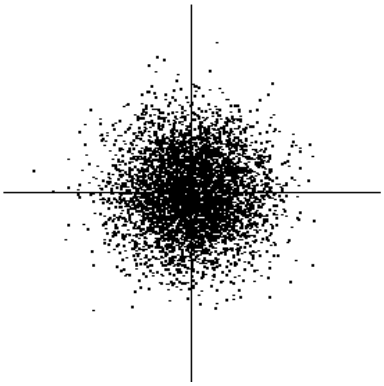


Figura 4.10 La distribución multicambiante de dos variables gaussianas.

Se puede probar que la distribución de cualquier transformación ortogonal de la gaussiana (x_1,x_2) tiene exactamente la misma distribución que (x_1,x_2), y que x_1 y x_2 son independientes. Así, en el caso de las variables gaussianas, sólo es posible estimar el modelo ICA hasta una transformación ortogonal. En otras palabras, la matriz A no es identificable para componentes gaussianas independientes. Si sólo una de los componentes independientes es gaussiana, puede ser estimado el modelo ICA.

4.2.6 Principios de la estimación de ICA

4.2.6.1 "No-gaussianas son independientes"

La clave para la estimación del modelo ICA es la no-gaussianidad. Realmente, sin no-gaussianidad la estimación no es totalmente posible.

El **Teorema del Límite Central**, un resultado clásico en la teoría de probabilidad, dice que la distribución de una suma de variables aleatorias independientes tiende hacia una distribución gaussiana, bajo ciertas condiciones. Así que, una suma de dos variables aleatorias independientes generalmente tiene una distribución que es más cercana a la gaussiana que cualquiera de las dos variables aleatorias originales.

Se considera que el vector de datos *x* está distribuido de acuerdo al modelo de datos ICA, esto es, una mezcla de componentes independientes. Por simplificar, se asume en esta sección que todas las componentes independientes tienen idénticas distribuciones. Para estimar una de las componentes independientes consideramos una combinación lineal de x_i, pudiéndose escribir como:

$$\boldsymbol{y = W^T x = \sum_i w_i x_i} \tag{4.17}$$

Donde w es un vector que va a ser determinado. Si *w* era una de la filas de la inversa de *A*, esta combinación lineal sería realmente igual a una de las componentes independientes. La cuestión es cómo usar el Teorema del Límite Central para determinar *w*, siendo éste igual a una de las filas de la inversa de *A*. En la práctica, no es posible determinar w exactamente, porque la matriz *A* no es conocida, pero si se puede encontrar una estimación que proporcione una buena aproximación.

Para ver como esto se aproxima al principio básico de la estimación de ICA, se puede realizar un cambio de variables, definiendo $z=A^T w$:

$$\boldsymbol{y = w^T x = w^T A s = z^T s} \tag{4.18}$$

Siendo *y* una combinación lineal de s_i, con el peso dado por z_i. Una suma de dos variables aleatorias independientes es más gaussiana que las variables originales. $z^T s$ es más gaussiana que cualquiera de los s_i y se convierten en menos gaussianas cuando sean

iguales a una de las s_i. En este caso, obviamente solo uno de los elementos z_i de z es no-cero.

Por lo tanto, es posible tomar como w al vector que maximiza la no-gaussianidad de $w^T x$. Tal vector correspondería necesariamente (en el sistema de coordenadas transformado) a *z*, el cual tiene sólo una componente no-cero. Esto significa que $w^T x = z^T s$ es igual a una de las componentes independientes.

Maximizando la no-gaussianidad de $w^T x$ se obtiene una de las componentes independientes. De hecho, la perspectiva de la optimización para la no-gaussianidad en el espacio n-dimensional de vectores w tiene 2n-máximos locales, dos para cada componente independiente, correspondiendo a s_i y $-s_i$ (renombradas las componentes independientes pueden ser estimadas sólo hasta un signo multiplicativo). Para encontrar muchas componentes independientes, hay que encontrar todos los máximos locales, lo cual no es difícil, porque las diferentes componentes independientes son no-correlacionadas: podemos siempre limitarnos a la investigación del espacio que proporciona estimaciones no-correlacionadas con las anteriores. Esto corresponde a la ortogonalización en una adecuada transformación (blanqueado) del espacio.

4.2.6.2 Kurtosis

La clásica medida de la no-gaussianidad es la kurtosis o el cumulante de cuarto orden. La kurtosis de *y* es clásicamente definida como:

$$\boldsymbol{kurt(y) = E\{y^4\} - 3(E\{y^2\})^2} \tag{4.19}$$

Realmente, se asume que *y* es de varianza unidad, el lado derecho se simplifica a $E\{y^4\} - 3$. Mostrando que la kurtosis es simplemente una versión normalizada del cuarto momento $E\{y^4\}$. Para una gaussiana, *y*, el cuarto momento es igual a $3(E\{y^2\})^2$. Así la kurtosis es cero para una variable gaussiana aleatoria. Para la mayoría (pero no casi todos) de las variables gaussianas aleatorias, la kurtosis es no-cero.

La kurtosis puede ser tanto positiva como negativa. Las variables aleatorias de valor negativo de kurtosis se denominan subgaussianas, y las de kurtosis positiva son supergaussianas. En los libros de estadística se usan también las expresiones

correspondientes platykurtic y leptokurtic. Las variables supergaussianas aleatorias tienen típicamente un pdf puntiagudo con pesadas raíces, esto es, el pdf es relativamente grande en cero y los valores grandes de la variable, mientras que es pequeño para los valores intermedios. Un ejemplo típico es la distribución de Laplace, cuyo pdf (normalizado para varianza unidad) viene dado por:

$$\boldsymbol{p(y) = \frac{1}{\sqrt{2}} exp\left(\sqrt{2}|y|\right)} \tag{4.20}$$

Este pdf se observa en un ejemplo típico como el mostrado en la Figura 4.11 [HYV00]. De variables subgaussianas aleatorias, por otro lado, tienen típicamente un pdf plano, el cual es bastante constante cerca de cero y, muy pequeño para valores grandes de la variable.

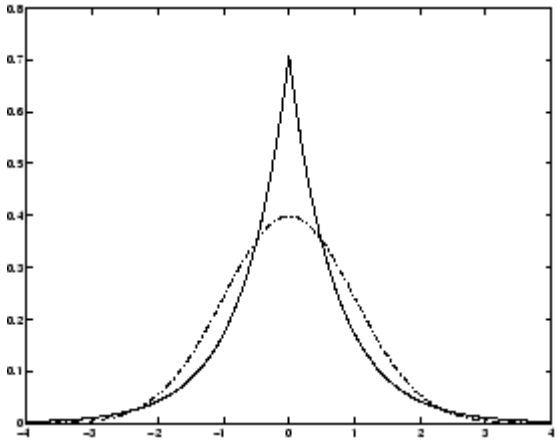


Figura 4.11 La función densidad de la distribución de Laplace, la cual es una distribución típica supergaussiana. Por comparación, la densidad de la gaussiana es dada por una línea muy pendiente. Ambas densidades están normalizadas a varianza unidad.

La no-gaussianidad es medida mediante el valor absoluto de la kurtosis. El cuadrado de la kurtosis también se puede emplear. Estos son cero para una variable gaussiana, y mayores que cero para la mayoría de variables no-gaussianas aleatorias. Hay variables no-gaussianas aleatorias que tienen cero como valor de kurtosis, pero éstos pueden considerarse casos raros.

La kurtosis, o más bien su valor absoluto, se ha utilizado ampliamente como una medida de no-gaussianidad en ICA y otras campos relacionados, debido principalmente a su simplicidad, tanto teórica como computacional. Computacionalmente, la kurtosis se

puede estimar simplemente utilizando el cuarto momento de los datos muestreados. El análisis teórico se simplifica mediante la propiedad de linealidad: si x_1 y x_2 son dos variables aleatorias independientes:

$$\boldsymbol{kurt(x_1 + x_2) = kurt(x_1) + kurt(x_2)} \tag{4.21}$$

$$\boldsymbol{kurt(\alpha x_1) = \alpha^4 kurt(x_1)} \tag{4.22}$$

Donde α es un escalar.

Se toma un modelo bidimensional ($x=As$) para encontrar la kurtosis y las componentes independientes, mediante la minimización o maximización de la kurtosis,

Considerando que las componentes independientes s_1, s_2 tienen unos valores de kurtosis de $kurt(s_1)$, $kurt(s_2)$, respectivamente, ambos diferentes de cero (recordar que se considera que tienen varianza unidad). Para una componente independiente resultaría:

$$\boldsymbol{y = w^T x} \tag{4.23}$$

Haciendo de Nuevo la transformación $z=A^Tw$, se tiene:

$$\boldsymbol{y = w^T x = w^T A s = z^T s = z_1 s_1 + z_2 s_2} \tag{4.24}$$

Basándose en la propiedad aditiva de la kurtosis, se tiene:

$$\boldsymbol{kurt(y) = kurt(z_1 s_1) + kurt(z_2 s_2) = z_1^4 kurt(s_1) + z_2^4 kurt(s_2)} \tag{4.25}$$

Por otro lado, se ha limitado a que la varianza de y sea igual a 1, basado en la misma consideración concerniente a s_1, s_2. Esto implica una limitación en z:

$$\boldsymbol{E\{y^2\} = z_1^2 + z_2^2 = 1} \tag{4.26}$$

Geométricamente significa que el vector *z* está limitado al círculo unidad en el plano bidimensional. El problema de la optimización es ahora obtener el máximo de la función siguiente en el círculo unidad:

$$\boldsymbol{|kurt(y)| = \left|z_1^4 kurt(s_1) + z_2^4 kurt(s_2)\right|} \tag{4.27}$$

Para simplificar, se considera que la kurtosis tiene el mismo signo, en cuyo caso los operadores valor absoluto pueden ser omitidos. El gráfico de esta función es la optimización para el problema.

En la práctica hay que empezar desde el vector peso, *w*, computando la dirección en la cual la kurtosis de $y=w^Tx$ crece más fuertemente (si la kurtosis es positiva) o decrece más rápidamente (si la kurtosis es negativa) basado en las muestras viables *x(1),..,x(T)* del vector mezcla *x*, y usando el método del gradiente o una de las extensiones para encontrar un nuevo vector *w*. El ejemplo puede ser generalizado para dimensiones arbitrarias, mostrando que la kurtosis puede, teóricamente, ser usada como un criterio de optimización para el problema de ICA.

Sin embargo, la kurtosis tiene también inconvenientes en la práctica, cuando su valor ha de ser estimado a partir de una muestra medida. El principal problema es que la kurtosis puede ser muy sensible a los valores extremos. Su valor puede depender de sólo unas pocas observaciones en las colas de la distribución, que pueden ser erróneas u observaciones irrelevantes. En otras palabras, la kurtosis no es una medida robusta de la no-gaussianidad.

Por lo tanto, otras medidas de no-gaussianidad podrían ser mejor que la kurtosis en algunas situaciones. A continuación se va a examinar negentropía, y, por último, introducir las aproximaciones de negentropía que más o menos combinan las buenas propiedades de ambos tipos de medidas.

4.2.6.3 Negentropía

Una segunda medida muy importante de no-gaussianidad viene dada por la negentropía. La negentropía se basa en la cantidad de información-teórica (diferencial) de entropía.

La **entropía** es el concepto básico de la teoría de la información. La entropía de una variable aleatoria puede ser interpretada como el grado de información que la observación de la variable da. Cuanto más aleatoria es la variable, mayor resulta su entropía. La entropía está estrechamente relacionada con la longitud de la codificación de la variable aleatoria, de hecho, en algunas hipótesis de simplificación, la entropía es la longitud de codificación de la variable aleatoria.

La entropía *H* se define para una variable aleatoria discreta *Y* como:

$$\boldsymbol{H(Y) = -\sum_i P\,(Y = a_i) log P(Y = a_i)} \tag{4.28}$$

Donde los a_i son los valores posibles de *Y*. Esta definición bien conocida puede ser generalizada para variables aleatorias y vectores de valores continuos, en cuyo caso se denomina normalmente entropía diferencial. La **entropía diferencial,** $\boldsymbol{H_{of}}$, de un vector aleatorio y con densidad f(y) es definida como:

$$\boldsymbol{H(y) = -\int f(y) \log f(y)\, dy} \tag{4.29}$$

Un resultado fundamental de la teoría de la información es que una variable gaussiana tiene la mayor entropía entre todas las variables aleatorias de igual varianza. La entropía puede ser utilizada como una medida de no-gaussianidad, de hecho, demuestra que la distribución gaussiana es la "más aleatoria" de todas las distribuciones. La entropía es pequeña para las distribuciones que están claramente concentradas en determinados valores, es decir, cuando la variable es agrupada, o tiene un pdf que es muy puntiagudo.

Para obtener una medida de no-gaussianidad que es igual a cero para una variable gaussiana y siempre no-negativa, a menudo se utiliza una versión ligeramente modificada de la definición de entropía diferencial, denominada **negentropía**. Negentropía, ***J***, que se define como sigue:

$$\boldsymbol{J(y) = H(y_{gauss}) - H(y)} \tag{4.30}$$

Donde y_{gauss} es una variable gaussiana aleatoria de la misma matriz de covarianza como y. Debido a las propiedades anteriormente mencionadas, la negentropía es siempre no-negativa, y es cero si y sólo si y tiene distribución gaussiana. La negentropía es una interesante propiedad adicional invariante para transformaciones lineales invertibles.

La ventaja de usar negentropía, como medida de no-gaussianidad es que está bien justificada por la teoría estadística. De hecho, la negentropía es en cierto sentido el mejor estimador de no-gaussianidad, en lo que respecta a propiedades estadísticas se refiere. El problema en el uso de negentropía es, sin embargo, que es computacionalmente compleja. La estimación de la negentropía utilizando la definición requeriría una estimación de la pdf. Por lo tanto, la simplificación de las aproximaciones de negentropía es muy útil.

4.2.6.4 Aproximaciones de negentropía

La estimación de la negentropía es compleja, y por lo tanto, este contraste sigue siendo principalmente una función teórica. En la práctica, se puede utilizar una cierta aproximación. Introduciendo las aproximaciones y utilizarlas en los siguientes pasos para obtener un método eficaz para ICA.

El método clásico de aproximación de la negentropía está utilizando los momentos de orden superior, por ejemplo, de la siguiente manera:

$$\boldsymbol{J(y) \approx \frac{1}{12} E\{y^3\}^2 + \frac{1}{48} kurt(y)^2} \tag{4.31}$$

La variable aleatoria *y* se supone que resulta ser cero y de varianza unidad. Sin embargo, la validez de tales aproximaciones puede ser más bien limitada.

Para evitar los problemas surgidos con la anterior aproximación de negentropía, han sido desarrolladas nuevas aproximaciones, las cuales se basan en el principio de máxima entropía. En general se obtiene la siguiente aproximación:

$$\boldsymbol{J(y) \approx \sum_{i=1}^{p} k_i [E\{G_i(y)\} - E\{G_i(\gamma)\}]^2} \quad (4.32)$$

Donde γ es una variable gaussiana de media cero y varianza unidad, y k_i son constantes positivas.

En el caso de que se use sólo una función no-cuadrática, *G*, la expresión mostrada en (4.32) se convierte en la aproximación:

$$\boldsymbol{J(y) \propto [E\{G(y)\} - E\{G(\gamma)\}]^2} \quad (4.33)$$

Para prácticamente cualquier función cuadrática distinta de *G*. Esto es claramente una aproximación basada en el momento. En efecto, teniendo $G(y)=y^4$, entonces se obtiene una aproximación basada en la kurtosis.

Pero el punto es que al elegir correctamente *G*, se obtiene una aproximación de negentropía. En particular, de la elección de *G* se obtienen unas estimaciones más robustas. Las siguientes elecciones de *G* han resultado ser muy útiles:

$$\boldsymbol{G_1(u) = \frac{1}{a_1} \log \cosh a_1 u} \qquad \boldsymbol{G_2(u) = -exp\left(-\frac{u^2}{2}\right)} \quad (4.34)$$

Donde $1 \leq a_1 \leq 2$ es una constante adecuada.

Por lo tanto, obtener aproximaciones de negentropía que dan un buen compromiso entre las propiedades de las dos medidas de no gaussianidad, dadas por la kurtosis y negentropía. Son conceptualmente sencillos, rápidos para calcular, sin embargo, tienen propiedades estadísticas atractivas, sobre todo la robustez.

4.2.7 Minimización de la Información Mutua

Otro enfoque para la estimación de ICA, inspirado en la teoría de la información, es la reducción al mínimo de la información mutua. Este enfoque conduce al mismo principio encontrando la mayoría de las direcciones de las no-gaussianas descrito anteriormente, en particular, da una justificación rigurosa de los principios heurísticos utilizados.

4.2.7.1 Información Mutua

Usando el concepto de entropía diferencial, se define la **información mutua, *I***, entre m (escalar) y las variables aleatorias y_i, *i=1...m* como sigue:

$$\boldsymbol{I(y_1, y_2, .., y_m) = \sum_{i=1}^{m} H(y_i) - H(y)} \quad (4.35)$$

La **Información mutua** es una medida de la dependencia entre variables aleatorias. De hecho, es equivalente a la conocida divergencia Kullback-Leibler, entre el conjunto de la densidad *f(y)* y el producto de su densidad marginal, una medida muy natural para la independencia. Es no-negativa, y cero si y sólo si las variables son estadísticamente independientes. Por lo tanto, la información mutua tiene en cuenta toda la estructura de la dependencia de las variables, y no sólo la covarianza.

La información mutua puede ser analizada mediante la interpretación de la entropía como código de longitud. Los términos $H(y_i)$ como la longitud de los códigos para la y_i cuando estos se codifican por separado, y *H(y)* le da la longitud de código, cuando *y* se codifica como un vector aleatorio, es decir, todos los componentes están codificados en el mismo código. La información mutua así lo muestra. El código de reducción de la longitud se obtiene por la codificación del conjunto de vectores en lugar de los componentes por separado. En general, se puede obtener una mejora de los códigos mediante la codificación de todo el vector. Sin embargo, si los y_i son independientes, no dan información sobre sí, y se mantendría la necesidad de aumentar la longitud de código.

Una propiedad importante de la información mutua es que se tiene una transformación lineal invertible para *y=Wx*:

$$\boldsymbol{I(y_1, y_2, .., y_n) = \sum_i H(y_i) - H(x) - \log|\det W|} \quad (4.36)$$

Ahora, se puede considerar qué ocurre si limitamos los y_i a ser no-correlacionados y con varianza unidad. Esto significa que:

$$\boldsymbol{E\{yy^T\} = WE\{xx^T\}W^T = I} \tag{4.37}$$

Lo cual implica:

$$\boldsymbol{\det I = 1 = (\det W E\{xx^T\} W^T) = (\det W)(\det E\{xx^T\})(\det W^T)} \tag{4.38}$$

Y esto implica que *det W* debería ser constante, más aún, para y_i de varianza unidad, la entropía y la negentropía difieren sólo en una constante y en el signo. Así se tiene:

$$\boldsymbol{I(y_1, y_2, .., y_n) = C - \sum_i J(y_i)} \tag{4.39}$$

Donde *C* es una constante que no depende de *W*. Mostrando la relación fundamental entre la negentropía y la información mutua.

4.2.7.2 Definiendo ICA mediante la información mutua

Dado que la información mutua es el elemento natural de información teórica medida de la independencia de variables aleatorias, se podría utilizar como criterio para la búsqueda de transformar ICA. Bajo este enfoque, que es una alternativa al enfoque de la estimación del modelo, se define ICA de forma aleatoria como un vector *x* de transformación invertible, donde la matriz *W* se determina de modo que la información mutua de los componentes de transformarse si es minimizada.

Es evidente que la búsqueda de una transformación invertible *W* que minimiza la información mutua es aproximadamente equivalente a la búsqueda de direcciones en las que se maximiza la negentropía. Más concretamente, es aproximadamente equivalente a la búsqueda de *1-D* subespacios tales que las proyecciones en estos subespacios tienen negentropía máxima. Rigurosamente hablando, la estimación de ICA por la minimización

de la información mutua es equivalente a maximizar la suma de no-gaussianidades de las estimaciones, cuando las estimaciones se ven limitadas a ser no-correlacionadas. La limitación de la no-correlacionalidad, aun cuando no es necesaria, simplifica considerablemente los cálculos.

La verosimilitud (Likelihood)

Un enfoque muy popular para la estimación de ICA es el modelo de estimación de máxima verosimilitud, que está estrechamente relacionada con el principio de Infomax. Este enfoque es esencialmente equivalente a la reducción al mínimo de información mutua. Es posible formular directamente la verosimilitud en el modelo ICA libre de ruido. Entonces estima el modelo mediante el método de máxima verosimilitud. Denominando $W=(w_1,..,w_n)^T$ a la matriz A^{-1}, el **logaritmo de la verosimilitud** toma la forma:

$$\boldsymbol{L = \sum_{t=1}^{T} \sum_{i=1}^{n} \log f_i \left(w_i^T x(t) \right) + T \log|\det W|} \qquad (4.40)$$

Donde las f_i son las funciones densidad de las s_i (aquí consideramos que son conocidas), y $x(t)$, $t=1,...,T$ son las realizaciones de x. El término $log\ |det\ W|$ en la verosimilitud viene de la regla clásica para (linealmente) transformar las variables aleatorias y sus densidades: En general, para un vector aleatorio x con densidad p_x y para cualquier matriz W, la densidad de $y=Wx$ es dada por $px(Wx)\ |det\ W|$.

El principio de Infomax

Esta función de contraste deriva del punto de vista de red neuronal. Esta está basada en la maximización de la entropía de salida (o flujo de información) de una red neuronal con salidas no-lineales. Considerando que x es la entrada de la red neuronal cuya salidas son dadas de la forma $g_i(wT_ix)$, donde los gi son algunas funciones escalares no-lineales, y los w_i son los vectores peso de las neuronas. Entonces hay que maximizar la entropía de las salidas:

$$\boldsymbol{L_2 = H\left(g_1(w_1^T x), .., g_n(w_n^T x) \right)} \qquad (4.41)$$

Si los g_i son bien elegidos, este marco también permite la estimación del modelo de ICA. Los resultados del principio de maximización de la entropía de red, o "infomax", son equivalentes a la estimación de la máxima verosimilitud. Esta equivalencia requiere que las no-linealidades g_i usadas en la red neural sean elegidas como funciones de distribución acumulativas correspondiendo a las densidades f_i, esto es, $g_i'(x)=f_i(x)$.

4.2.7.3 Conexión con la información mutua

Para ver la conexión entre la verosimilitud y la información mutua, se considera la esperanza del logaritmo de la verosimilitud:

$$\frac{1}{T}\boldsymbol{E}\{\boldsymbol{L}\} = \sum_{i=1}^{n} \boldsymbol{E}\{\log \boldsymbol{f}_i(\boldsymbol{w}_i^T \boldsymbol{x})\} + \log|\det \boldsymbol{W}| \qquad (4.42)$$

Realmente, si los f_i son iguales a las distribuciones actuales de $w^T{}_i x$, el primer termino será igual a $-\sum_i H(w^T{}_i x)$, y por lo tanto, la verosimilitud será igual, hasta una constante adaptativa, quedando como el negativo de la información mutua.

Realmente, en la práctica la conexión es aún fuerte porque las distribuciones de las componentes independientes no son conocidas. Una aproximación razonable sería estimar la densidad de $w^T{}_i x$ como parte del método de estimación ML (Maximum Likelihood o Máxima Verosimilitud), y usado como una aproximación de la densidad de s_i. En este caso, la verosimilitud y la información mutua son, para todos los propósitos prácticos, equivalentes.

Hay una pequeña diferencia que puede ser muy importante en la práctica. El problema con la estimación de la máxima verosimilitud es que las densidades f_i deberían ser estimadas correctamente. No es necesario estimarlas con una gran precisión, de hecho es suficiente con estimar si son sub- o supergaussianas. En muchos casos, de hecho, basta con el conocimiento previo de las componentes independientes, y no se necesita estimar la naturaleza de los datos. En algunos casos, si la información de la naturaleza de las componentes independientes no es correcta, la estimación ML dará resultados completamente erróneos. Por lo tanto, hay que tener cuidado con la estimación ML. Por el contrario, usando medidas razonables de no-gaussianidad, este problema normalmente no aparece.

4.2.8 ICA y la proyección perseguida

La aproximación a ICA hace explicita la conexión entre ICA y la proyección perseguida. La proyección perseguida es una técnica desarrollada en estadística para encontrar proyecciones interesantes de datos multidimensionales. Tales proyecciones pueden ser usadas para la visualización óptima del dato, y como la estimación de la densidad y la regresión. En la proyección perseguida básica *(1-D),* intentamos encontrar direcciones tales que las proyecciones de los datos en estas direcciones tienen distribuciones interesantes, esto es, describiendo alguna estructura. Esto ha sido argumentado por Huber y por Jones y Sibson que la distribución gaussiana es al menos interesante, y que las direcciones más interesantes son aquellas que muestran al menos una distribución gaussiana. Esto es exactamente lo que se estima con el modelo ICA.

La utilidad de encontrar tales proyecciones puede ser vista en la Figura 4.12 [HYV00], donde la proyección en la dirección de la proyección perseguida (horizontal) muestra la estructura agrupada de los datos. La proyección en la primera componente principal (vertical), por otro lado, los fallos muestran esta estructura.

Se observa que los datos se encuentran divididos en dos agrupaciones, situándose verticalmente la componente principal (la dirección de máxima varianza). Y por lo tanto, permitiendo la fácil separación entre las agrupaciones. Por el contrario, la fuerte proyección no-gaussiana en la dirección perseguida es horizontal, facilitándose una separación optima de ambas agrupaciones.

Así, en la formulación general, ICA puede ser considerado una variante de la proyección perseguida. Todas las no-gaussianidades medidas y los correspondientes algoritmos ICA presentados aquí podrían también ser denominadas índices de las proyecciones perseguidas y algoritmos. En particular, la proyección perseguida permite alcanzar la situación donde son menos las componentes independientes s_i que las variables originales x_i. Asumiendo que estas dimensiones del espacio que no son ocupadas por las componentes independientes son llenadas por el ruido gaussiano, se observa que computando las direcciones de las proyecciones no-gaussianas perseguidas, estimamos efectivamente las componentes independientes. Donde todas las direcciones no-gaussianas

han sido encontradas, todas las componentes independientes han sido estimadas. Tal que el procedimiento puede ser interpretado como una mezcla de la proyección perseguida e ICA.

Figura 4.12 Ilustración de la proyección perseguida y las proyecciones no-gaussianas.

Sin embargo, se podría decir que en la formulación de la proyección perseguida, no se hizo ningún modelo de dato ni ninguna suposición sobre las componentes independientes. Si el modelo ICA aborda, optimizando las medidas de no-gaussianidad producidas por las componentes independientes, entonces se tomarán sus direcciones como las proyecciones perseguidas.

4.2.9 Preprocesado para ICA

Sin embargo, antes de aplicar el algoritmo ICA a un dato, es generalmente muy útil realizar algún preprocesado. En esta sección, se discutirán algunas técnicas de preprocesado que simplifican el problema de la estimación de ICA y hacen un mejor condicionado.

4.2.9.1 Centrado

El preprocesado más básico y necesario es para centrar x, esto es, substraer su vector media m=E{x} tanto como hacer x una variable de media cero. Esto implica que *s* es de media cero.

Este preprocesado está hecho solamente para simplificar los algoritmos ICA, lo cual no significa que la media no debería ser estimada. Tras estimar la matriz de muestras A con el dato centrado, podemos completar la estimación añadiendo el vector media de s detrás de las estimaciones centradas de *s*. El vector media de s está dado por $A^{-1}m$, donde m es la media que era substraída en el preprocesado.

4.2.9.2 Blanqueamiento

Otra estrategia de preprocesado muy útil en ICA es blanquear primero las variables observadas. Esto significa que antes de la aplicación del algoritmo ICA (y después del centrado), transformamos el vector de observaciones *x* linealmente tal que obtenemos un nuevo vector $\tilde{x}$, el cual es blanco, esto es, sus componentes están no-correlacionadas y sus varianzas son unidad. En otras palabras, la matriz de covarianza de $\tilde{x}$ igualada a la matriz identidad:

$$\boldsymbol{E\{\tilde{x}\tilde{x}^T\} = I} \tag{4.43}$$

La transformación del blanqueamiento siempre es posible. Un método popular para el blanqueamiento es usar la descomposición del autovalor (EVD) de la matriz de covarianza:

$$\boldsymbol{E\{xx^T\} = EDE^T} \tag{4.44}$$

Donde E es la matriz ortogonal de autovectores de $E\{xx^T\}$, *D* es la matriz diagonal de sus autovalores, $D=diag(d_1,..d_n)$. Notar que $E\{xx^T\}$ puede ser estimado de una manera estándar desde la muestra accesible *x(1),..,x(T).* El blanqueamiento puede ahora escribirse como:

$$\boldsymbol{\tilde{x} = ED^{-1/2}E^Tx} \tag{4.45}$$

Donde la matriz $D^{-1/2}$ es computada por una simple operación como $D^{-1/2}=diag(d_1^{-1/2},..,d_n^{-1/2})$. Es fácilmente chequeable que ahora $E\{xx^T\} = I$

El blanqueamiento transforma la matriz mezcla en una nueva, $\tilde{A}$. Teniendo:

$$\tilde{\boldsymbol{x}} = \boldsymbol{E}\boldsymbol{D}^{-1/2}\boldsymbol{E}^{\boldsymbol{T}}\boldsymbol{A}\boldsymbol{s} = \tilde{\boldsymbol{A}}\boldsymbol{s} \tag{4.46}$$

La utilidad del blanqueamiento reside en el hecho que la nueva matriz de mezcla, $\tilde{A}$, es ortogonal. Se puede escribir:

$$\boldsymbol{E}\{\tilde{\boldsymbol{x}}\tilde{\boldsymbol{x}}^{\boldsymbol{T}}\} = \tilde{\boldsymbol{A}}\boldsymbol{E}\{\boldsymbol{s}\boldsymbol{s}^{\boldsymbol{T}}\}\tilde{\boldsymbol{A}}^{\boldsymbol{T}} = \tilde{\boldsymbol{A}}\tilde{\boldsymbol{A}}^{\boldsymbol{T}} = \boldsymbol{I} \tag{4.47}$$

Se observa que el blanqueamiento reduce el número de parámetros a estimar. A menos de haber estimado los parámetros n^2 que son los elementos de la matriz original *A*, sólo se necesita la nueva matriz de mezcla ortogonal $\tilde{A}$. Una matriz ortogonal contiene *n(n-1)/2* grados de libertad. Por ejemplo, en dos dimensiones, una transformación ortogonal es determinada mediante el parámetro del ángulo simple. Para grandes dimensiones, la matriz ortogonal contiene solo sobre la mitad del número de parámetros de una matriz arbitraria. Así se puede decir que el blanqueamiento resuelve la mitad del problema de ICA. Porque el blanqueamiento es muy simple y un procedimiento estándar, mucho más simple que los algoritmos ICA, esta es una buena idea para reducir la complejidad del problema de esta forma.

Puede también ser muy útil para reducir la dimensión del dato a la vez que se hace el blanqueamiento. Entonces se observan los autovalores $d_{j,}$ de $E\{\tilde{x}\tilde{x}^T\}$, y se descartan aquellos que son muy pequeños, como suelen ser los efectos del ruido.

Una ilustración gráfica del efecto del blanqueamiento se puede observar en la Figura 4.13 [HYV00], en la cual el dato de la Figura 4.9 ha sido blanqueado. El cuadrado definiendo la distribución es ahora claramente una versión rotada del cuadrado original.

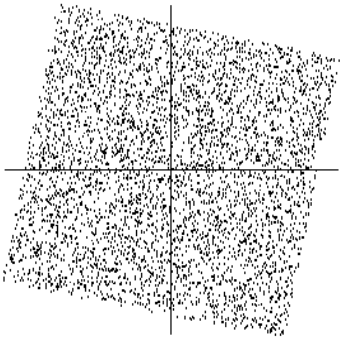


Figura 4.13 La distribución conjunta de las mezclas blanqueadas.

En adelante se supondrá que los datos han sido preprocesados mediante el centrado y el blanqueamiento. Para simplificar la notación, se denotará al dato preprocesado como *x*, y la matriz de mezcla transformada por A, omitiendo las tildes.

4.2.10 Preprocesado adicional

El éxito de ICA para el conjunto de datos dado puede depender crucialmente de la ejecución de algunos pasos del preprocesado de aplicaciones dependientes. Por ejemplo, si el dato consiste en señales en tiempo, podría resultar útil realizar algún filtrado paso-bajo. Notar que si filtramos linealmente las señales observadas $x_i(t)$ para obtener nuevas señales, $x_i^*(t)$, el modelo ICA se mantiene con la misma matriz de mezcla.

Nombrando como *X* a la matriz que contiene las observaciones *X(1),..,X(T)* como sus columnas, el modelo ICA puede ser expresado como:

$$\boldsymbol{X = AS} \tag{4.48}$$

Ahora, el filtrado en el tiempo de *X* corresponde a multiplicar *X* desde el lado derecho de la matriz, podría nombrarse como *M*, resultando:

$$\boldsymbol{X^* = XM = ASM = AS^*} \tag{4.49}$$

Expresión que muestra que el modelo ICA sigue siendo válido.

4.3 FastICA

Anteriormente han sido introducidas diferentes medidas de no-gaussianidad, es decir, funciones objetivas para la estimación de ICA. En la práctica, también se necesita un algoritmo para maximizar la función de contraste. Ahora, se aporta un método muy eficiente de maximización adecuada para esta tarea. Se supone que los datos son preprocesados aquí mediante el centrado y el blanqueo.

El algoritmo más popular de ICA, el cual es aplicado satisfactoriamente a muchos problemas del mundo real es el FastICA.

Para este algoritmo, las observaciones y la diagonal de la matriz de covarianza tienen que tener media cero. La última condición puede ser totalmente completada si las observaciones son blanqueadas. Para este propósito, la descomposición del autovalor de la matriz de observaciones $\underline{C}$ tiene que ser computada:

$$\underline{\boldsymbol{C}} = \underline{\boldsymbol{V}}\,\underline{\boldsymbol{D}\boldsymbol{T}^{\boldsymbol{T}}} \tag{4.50}$$

Donde $\underline{V}$ es la matriz de autovalor y $\underline{D}$ es la matriz diagonal conteniendo el autovalor de $\underline{X}$ en su diagonal. Basado en las matrices $\underline{D}$ y $\underline{V}$, el blanqueamiento de la matriz $\underline{Q}$ se calcula como:

$$\underline{\boldsymbol{Q}} = \boldsymbol{D}^{\boldsymbol{-1/2}}\,\underline{\boldsymbol{V}^{\boldsymbol{T}}} \tag{4.51}$$

Multiplicando esta matriz con la matriz de observaciones $\underline{X}$ encabeza a la deseada no-correlación de las observaciones. Pudiendo considerarse que las observaciones tienen media cero y son blancas.

El logro del algoritmo de FastICA es encontrar una matriz $\underline{W}$ tal que las filas de $\underline{Y}=\underline{W}\,\underline{X}$ son estadísticamente independientes. En FastICA la independencia de los componentes de un vector aleatorio es medida mediante la información mutua, *I*. Esta independencia medida puede ser expresada en términos de negentropía, *J*, tal como se indica a continuación:

$$\boldsymbol{I(y_1, y_2, .., y_N) = J(y) - \sum_i J(y_i)} \quad (4.52)$$

I es una función no negativa la cual desaparece si los elementos de *y* son estadísticamente independientes, así como la negentropía, *J(y)*, es independiente, bajo transformaciones lineares *(J(Wx)=J(x))*, para alguna matriz *W* y algún vector x. Sólo el segundo término en la ecuación anterior necesita ser optimizada en el contexto de ICA.

Así, el problema de FastICA puede estar establecido como:

$$\textbf{Maximizar: SUM N, i = 1 } \mathbf{J(w_i)\ w.r.t.\ } \boldsymbol{w_{i,j} = 1, .., n} \quad (4.53)$$

$$\textbf{Bajo la limitación: } \mathbf{E\{(w_k T_x)(w_j^T x)\} = d_{jk}} \quad (4.54)$$

Como $y_i=w_i^T X$, la normalización $E\{(w_{kT}x)(wj^T x)\}=d_{jk}$ es necesaria así como se consideran las fuentes subyacentes para que sean estadísticamente independientes y, por ello también no-correlacionadas, en ICA.

Para la tarea de optimización se necesita una buena estimación de la negentropía de la variable aleatoria. Se emplea la siguiente fórmula de estimación:

$$\boldsymbol{J(y_i) = \frac{1}{d^2}(E\{G(y_i)\} - E\{G(n)\})^2} \quad (4.55)$$

Donde G es una función no lineal, n es la variable gaussiana estandarizada y d es la constante normalizada.

Esta expresión para $J(y_i)$ puede ser optimizada bajo las construcciones dadas por el significado de las condiciones de Kuhn-Tucker y el método Newton. Tras considerables cálculos, éste encabeza la siguiente actualización de la regla para las filas individuales w de *W*:

$$\boldsymbol{W += E\{xG'(w^T x)\} - E\{G''(w^T x)\}} \quad (4.56)$$

$$\boldsymbol{w}^* = \frac{\boldsymbol{w}+}{||\boldsymbol{w}+||} \tag{4.57}$$

Este procedimiento es usado iterativamente hasta la determinación de todas las filas de $\underline{W}$. Sin embargo, se debería evitarse que el algoritmo converja numerosas veces al mismo máximo de la negentropía. Esto puede ser obtenido, si después de cada iteración del método de Newton la fila determinada nuevamente w_{q+1} es no-correlacionada respecto a las anteriores determinadas previamente, $w_1, w_2, .., w_q$ de $\underline{W}$.

Una vez que la matriz $\underline{W}$ ha sido determinada, las fuentes pueden ser obtenidas mediante la simple multiplicación del inverso de $\underline{W}$ por las observaciones.

4.3.1 FastICA para una unidad.

Para empezar, vamos a mostrar una versión para una unidad del algoritmo FastICA. Por una "unidad" se refiere a una unidad computacional, eventualmente con una neurona artificial, teniendo un vector peso, w, que la neurona es capaz de actualizar mediante una regla de aprendizaje. La regla de aprendizaje FastICA encuentra una dirección, es decir, un vector unitario, w, tal que la proyección, $w^T x$, maximiza la no-gaussianidad.

La no-gaussianidad es aquí medida por la aproximación de negentropía, $J(w^T x)$. Volvamos a mencionar que la varianza, $w^T x$, debería ser aquí limitada a la unidad, para que el blanqueado de los datos sea equivalente para la limitación de la norma de w sea la unidad.

El FastICA se basa en un esquema de iteraciones en punto fijo para encontrar el máximo de la de no-gaussianidad de $w^T x$. Puede ser también derivado como una aproximación a la iteración de Newton. Decir que g es la derivada de la función no-cuadrática, G; por ejemplo se tiene:

$$\mathbf{G_1} = \mathbf{tanh}(\mathbf{a_1 u}) \tag{4.58}$$

Donde $1 \leq a_1 \leq 2$ es una constante ajustable, a menudo tomando el valor de $a_1 = 1$. La expresión básica del algoritmo FastICA es:

1. Elegir un vector peso, w inicial (por ejemplo, aleatorio).
2. Hacer:

$$\boldsymbol{w}+= \boldsymbol{E}\{\boldsymbol{x}\boldsymbol{g}(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x})\} - \boldsymbol{E}\{\boldsymbol{g}'(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x})\}\boldsymbol{w} \tag{4.59}$$

3. Tomando:

$$\boldsymbol{w} = \frac{\boldsymbol{w}+}{||\boldsymbol{w}+||} \tag{4.60}$$

4. Si no converge, volver a 2.

Hay que tener en cuenta que la convergencia significa que los antiguos y los nuevos valores del punto w en la misma dirección, es decir, su producto escalar es (casi) igual a 1. No es necesario que el vector converja a un solo punto, desde *w* y *-w* define la misma dirección. Esto es debido a que las componentes independientes se pueden definir sólo hasta un signo multiplicativo. Teniendo en cuenta también que aquí se supone que los datos son pre-blanqueados.

La derivación de FastICA es la siguiente: en primer lugar, tener en cuenta que la máxima de la aproximación de la negentropía de *w* y *-w* obtenida de la óptima de $E\{G(w^T x)\}$. Según la condiciones de Kuhn-Tucker, la óptima de $E\{G(w^T x)\}$, bajo la restricción de $E\{(w^T x)^2\} = \left|\left|w\right|\right|^2 = 1$, es obtenida en puntos donde:

$$\boldsymbol{E}\{\boldsymbol{x}\boldsymbol{g}(\boldsymbol{w}^{\boldsymbol{T}}) - \boldsymbol{\beta}\boldsymbol{w} = \boldsymbol{0}\} \tag{4.61}$$

Se va a tratar de resolver esta ecuación mediante el método de Newton, obteniendose su matriz Jacobiana, *JF(w)*, como:

$$\boldsymbol{JF}(\boldsymbol{w}) = \boldsymbol{E}\{\boldsymbol{x}\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{g}'(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x})\} - \boldsymbol{\beta}\boldsymbol{I} \tag{4.62}$$

Para simplificar la inversión de esta matriz, se aproxima el primer término, resultando:

$$\boldsymbol{E}\{\boldsymbol{x}\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{g}'(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x})\} \approx \boldsymbol{E}\{\boldsymbol{x}\boldsymbol{x}^{\boldsymbol{T}}\}\boldsymbol{E}\{\boldsymbol{g}'(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x})\} = \boldsymbol{E}\{\boldsymbol{g}'(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x})\}\boldsymbol{I} \tag{4.63}$$

Así la matriz Jacobiana se convierte en diagonal y, puede ser fácilmente invertida. Obteniendo la siguiente iteración aproximativa de Newton:

$$\boldsymbol{w} += \frac{\boldsymbol{w} - [\boldsymbol{E}\{\boldsymbol{x}\boldsymbol{g}(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x})\} - \boldsymbol{\beta}\boldsymbol{w}]}{\boldsymbol{E}\{\boldsymbol{g}'(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x})\} - \boldsymbol{\beta}} \tag{4.64}$$

Este algoritmo puede ser simplificado mediante la multiplicación de ambos lados por β-$E\{g'(w^Tx)\}$. Dando, tras una simplificación algebraica, la iteración de FastICA.

En la práctica, las esperanzas en FastICA deberían ser sustituidas por sus estimaciones. Las estimaciones naturales son las correspondientes medias mezcladas. Idealmente, todos los datos disponibles deben ser utilizados, pero esto no es a menudo una buena idea porque puede ser demasiado exigente a nivel de necesidades computacionales. Entonces, los promedios pueden estimarse utilizando una muestra más pequeña, cuyo tamaño puede tener un efecto considerable sobre la exactitud de las estimaciones finales. Los puntos de muestreo deben elegirse por separado en cada iteración. Si la convergencia no es satisfactoria, entonces se puede aumentar el tamaño de la muestra.

4.3.2 FastICA para varias unidades

El algoritmo para una unidad visto anteriormente estima sólo una de las componentes independientes, o un ejercicio de proyección activa en dirección. Para estimar varias componentes independientes, hay que ejecutar FastICA utilizando un algoritmo para varias unidades (por ejemplo, las neuronas), con vectores peso: $w_1,..,w_n$.

Para evitar diferentes vectores de convergencia para los mismos máximos se deben correlacionar las salidas $w_1^Tx,..,w_n^Tx$ después de cada iteración.

Una forma sencilla de lograr la decorrelación es emplear el esquema de la deflación basado en la correlación de Gram-Schmidt. Con ello se van estimando las componentes independientes una a una. Cuando se han estimado p componentes independientes, o p vectores, $w_1,..,w_p$, se ejecuta el algoritmo de un punto fijo para una unidad, y después de cada iteración se restan las proyecciones $w_{p+1}^Tw_jw_{j,j}=1,..,p$, de los vectores p estimados previamente, a w_{p+1}, y entonces se renormaliza w_{p+1}:

- Hacer:

$$\boldsymbol{w_{p+1} = w_{p+1} - \sum_{j=1}^{p} w_{p+1}^{T} w_j w_j} \quad (4.65)$$

- Hacer:

$$\boldsymbol{w_{p+1} = \frac{w_{p+1}}{\sqrt{w_{p+1}^{T} w_{p+1}}}} \quad (4.66)$$

En ciertas aplicaciones, sin embargo, puede quererse utilizar una decorrelación simétrica, en la que ningún vector tiene privilegios sobre los otros. Esto puede lograrse, por ejemplo, por el método clásico aplicando la matriz de raíces cuadradas.

- Hacer:

$$\mathbf{W = (WW^{T})^{-1/2} W} \quad (4.67)$$

Donde W es la matriz $(w_1,..,w_n)^T$ de los vectores, y la raíz cuadrada inversa $(WW^T)^{-1/2}$ es obtenida de la descomposición del autovalor $WW^T=F\Lambda F^T$ como $(WW^T)^{-1/2}=F\Lambda^{-1/2}F^T$.

Una alternativa más simple es el siguiente algoritmo iterativo:

1. Hacer:

$$\boldsymbol{W = \frac{W}{\sqrt{||WW^{T}||}}} \quad (4.68)$$

2. Hacer:

$$\boldsymbol{W = \frac{2}{3} W - \frac{1}{2} WW^{T} W} \quad (4.69)$$

3. Repetir 2 hasta la convergencia.

4.3.3 FastICA y la máxima verosimilitud

Finalmente, se da una versión de FastICA que muestra explícitamente la conexión con el algoritmo de Infomax o máxima verosimilitud. Se puede expresar FastICA como:

$$\boldsymbol{W += W + \Gamma[diag\,(-\beta_i) + E\{g(y)y^{T}\}]W} \quad (4.70)$$

Donde y=Wx, $\beta_i=E\{y_i g(y_i)\}$, y $\Gamma=diag(1/(\beta_i-E\{g'(y_i)\}))$. La matriz W necesita ser ortogonalizada tras cada paso. En esta versión de matriz es natural ortogonalizar W simétricamente.

La versión interior de FastICA podría ser comparada con el método de gradiente estocástico para máxima verosimilitud:

$$\boldsymbol{W} += \boldsymbol{W} + \boldsymbol{\mu}[\boldsymbol{I} + \boldsymbol{g}(\boldsymbol{y})\boldsymbol{y}^{\boldsymbol{T}}]\boldsymbol{W} \tag{4.71}$$

Donde μ es la tasa de aprendizaje, no necesariamente constante en el tiempo. Se observa que FastICA puede ser considerado como un algoritmo en punto fijo para una estimación de la máxima verosimilitud del modelo de datos de ICA. En FastICA, la velocidad de convergencia es optimizada mediante la elección de las matrices Γ y $diag(-\beta_i)$. Otra ventaja de FastICA es que se pueden estimar tantas componentes independientes sub- como supergaussianas, lo cual es normal en contraste con los algoritmos de ML, que sólo trabajan para una determinada clase de las distribuciones.

4.3.4 Propiedades del algoritmo FastICA

El algoritmo FastICA y las subyacentes funciones de contraste tienen una serie de propiedades deseables en comparación con los métodos existentes de ICA:

1. La convergencia es cúbica (o por lo menos cuadrada), bajo el supuesto del modelo de datos de ICA (para una prueba). Esto contrasta con los algoritmos ICA ordinarios basados en métodos estocásticos de gradiente descendente, donde la convergencia es sólo lineal. Esto significa una convergencia muy rápida, como ha sido confirmado por las simulaciones y los experimentos con datos reales.
2. Frente a los algoritmos basados en gradiente, no hay parámetros de tamaño de paso a elegir, significando esto que el algoritmo es fácil de usar.
3. El algoritmo encuentra directamente las componentes independientes de (prácticamente) cualquier distribución no-gaussiana utilizando cualquier no linealidad g, lo cual contrasta con muchos algoritmos, donde algunas estimaciones

de la función de distribución de probabilidad tienen que estar primero disponibles, y la no-linealidad se debe elegir en consecuencia.

4. El rendimiento del método puede ser optimizado mediante la elección de una adecuada no-linealidad g. En particular, se puede obtener algoritmos que son robustos y/o de mínima varianza. De hecho, las dos no-linealidades tienen algunas propiedades óptimas.
5. Las componentes independientes se pueden estimar una por una, que es aproximadamente equivalente a obtener las proyecciones perseguidas. Resultando útil en el análisis de los datos explorados, y reduce la carga computacional del método en los casos en que sólo algunas de las componentes independientes deben ser estimadas.
6. El FastICA tiene la mayoría de las ventajas de los algoritmos neuronales: es paralelo, distribuido, computacionalmente simple, y requiere poco espacio de memoria. Los métodos estocásticos de gradiente parecen ser preferibles si la adaptabilidad rápida en un entorno cambiante es necesaria.

4.4 Estudios actuales de métodos basados en ICA

En los últimos años se ha investigado en el entorno de la Separación Ciega de Señales y los algoritmos ICA. Dentro de los trabajos realizados por distintos autores, cabe destacar:

Una unidad de procesado digital de señales (DSP) para implementar un ICA Infomax extendido para la Separación Ciega de Señales.

Algoritmo C-FICA, que es una extensión convolutiva del algoritmo FastICA. Es un algoritmo en punto fijo rápido que realiza la separación ciega de señales de mezclas convolutivas. Está basado en un proceso esférico convolutivo que permite el uso de FastICA actualizado para extraer iterativamente el proceso de innovación de las fuentes mediante un procedimiento de deflación.

Estudios sobre un método de separación ciega de mezclas con relación al tiempo-frecuencia para mezclas lineares instantáneas, LI-TIFROM. LI-TIFROM es un método de separación en escasas muestras de base. Se basa en el análisis en el tiempo-frecuencia. Encontrando primero las zonas de señales simples, para posteriormente estimar sobre ellas

una matriz de mezcla; para, por último, para cuando todas las columnas de la matriz de mezcla han sido estimadas, recuperar las señales originales.

Estudio de sistemas digitales de altas prestaciones para aplicaciones en electrónica de potencia.

Estudio del comportamiento de un algoritmo para la separación de señales recibidas con pequeños retardos. Este método considera el efecto de los retardos de las señales en su propagación, así como el efecto del ruido existente en el medio de propagación. Cuando los retardos son pequeños se puede tener en cuenta su efecto sin tener que emplear ICA convolutivo, el cual tiene una gran complejidad matemática. Para su resolución teniendo en cuenta los retardos, estudia las mezclas producidas instantáneamente, las cuales se consideran formadas por las señales de interés y sus primeras derivadas, teniendo, por lo tanto, en cuenta el retardo. Para que este algoritmo sea válido, el retardo ha de estar acotado.

Capítulo 5

Resumen:

En este capítulo se presentan algunos algoritmos utilizados para la separación ciega de fuentes cuando la matriz de mezcla cambia en el tiempo. Inicialmente se plantea el estado actual mostrando algoritmos que se encuentran en la bibliografía y posteriormente se presentan dos algoritmos propuestos en esta tesis: el ASWICA y ADSWICA basados en ventanas deslizantes con FastICA.

5 NUEVOS ALGORITMOS DE SEPARACIÓN EN MEDIOS NO ESTACIONARIOS

Uno de los objetivos de este trabajo es abordar el problema de la separación ciega de fuentes cuando la matriz mezcla de señales cambia dinámicamente. Este problema resulta interesante de abordar con DSPs principalmente por varios motivos:

- El coste computacional resulta más elevado que el procesamiento convencional de ICA donde la matriz cambia en el tiempo.
- Los problemas, donde pueden utilizarse las plataformas de DSPs, pueden tener carácter de mezcla no estacionaria.

En la Figura 5.1 [COM12] se representan en dos dimensiones dos señales mezcladas de forma estática (izquierda) y dinámica (derecha).

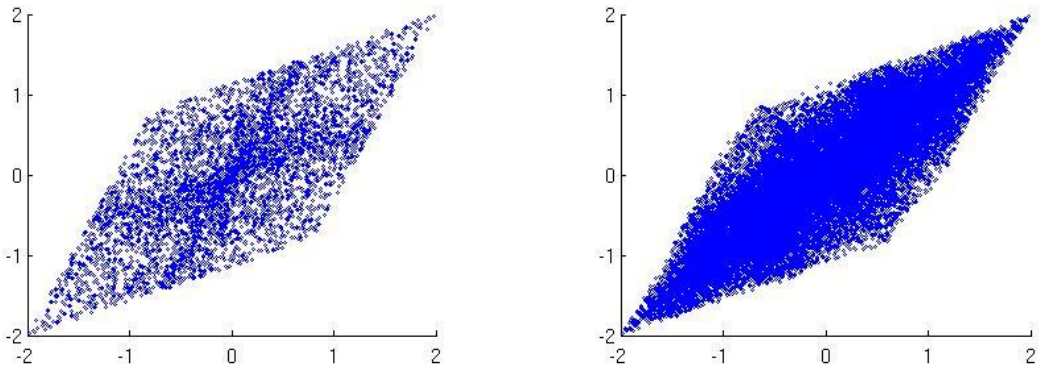


Figura 5.1 Distribución de dos señales con mezcla estática (a) y dinámica(b).

Si se aplica un algoritmo de separación tomando todas las muestras, en el caso de mezcla dinámica, no se obtiene un resultado óptimo ya que no converge y además no se extraerían correctamente las fuentes originales.

A continuación se describen algunos algoritmos que pueden encontrarse en la bibliografía: EASI, MeR-MaID-SIG, SIPEX-G, ICAdec y se proponen dos nuevos algoritmos ASWICA y ADSWICA.

5.1 Algoritmo EASI

La separación ciega de fuentes se puede resolver aplicando técnicas de resolución para mezclas de señales digitales de comunicaciones. Pudiendo predecir el rango de error cuando las señales son recibidas con la misma portadora de frecuencia y separadas ciegamente. En un entorno inalámbrico, donde las fuentes son móviles o el entorno es cambiante, la matriz de mezcla variará con el tiempo. Se identifica la mayor fuente de error en la separación, extendiéndolo a un algoritmo adaptativo del tamaño del paso a un caso con valor complejo para mitigar los errores en un entorno dinámico.

Con este sistema se puede predecir: el rango de error simbólico (SER) de las señales separadas; la mayor fuente de error en la separación para matrices de mezcla que varían con el tiempo; y extender un algoritmo iterativo de tamaño de paso adaptativo, pero sin conocer la matriz de mezcla para el caso de usar valores complejos.

Estudiando el problema de la separación de señales para comunicaciones digitales multicanal en un entorno inalámbrico, e ilustrando la relación entre la habilidad para señales de comunicaciones digitales medidas mediante la interferencia entre señales y el detector de rango del error de símbolo de la máxima verosimilitud. Bajo la suposición de un camino múltiple difuso y canales independientes espacialmente se observa la existencia de un incremento en la dificultad en la separación cuando se presenta un fallo de la matriz del canal. Aportando el algoritmo de tamaño de paso adaptativo EASI-base para datos con valores complejos, el cual proporciona buenos resultados en la simulación comparada para la aproximación del tamaño de paso constante.

Cuando la matriz se convierte en condicionada al fallo, el tamaño del paso se incrementa para amoldarse a los cambios rápidos en la matriz de separación. En otros tipos de algoritmos de gradiente estocástico descendiente se han logrado implementaciones

mejoradas permitiendo que varíe el tamaño del paso. En muchos de estos algoritmos, el tamaño del paso es actualizado de acuerdo con su gradiente estocástico descendente:

$$\boldsymbol{\lambda_{t+1} = \lambda_{t+1} - \varrho_t \frac{\vartheta \zeta(\cdot)}{\vartheta \lambda_t}} \tag{5.1}$$

Donde $\zeta(\cdot)$ es la función de coste y ρ es una constante pequeña. En el caso LMS, la función de coste sería el error cuadrado entre la señal filtrada y la deseada. En separación ciega de fuentes se medirá la independencia de la función.

Se puede encontrar un algoritmo de tamaño de paso adaptativo para el algoritmo EASI de valores complejos partiendo de una aproximación del tamaño de paso adaptativo para el algoritmo EASI. El gradiente relativo respecto al tamaño de paso es:

$$\boldsymbol{\nabla_{\lambda\mu}\phi|_{\lambda=\lambda_t} = \langle \Gamma^H(t+1), \Gamma(t) \rangle} \tag{5.2}$$

$$\boldsymbol{(t) = [y_t y_t^H - I + g(y_t) y_t^H - y_t g(y_t)^H] W_t} \tag{5.3}$$

Este es un gradiente de valores complejos, los cuales proporcionan información sobre la dirección descendente en los planos real e imaginario, teniendo una aproximación de una combinación simple mediante la elección del porcentaje de las componentes reales e imaginarias:

$$\boldsymbol{\lambda_{t+1} = \lambda_t - \frac{1}{2}\left[\Re\{\nabla_{\lambda\mu}\phi|_{\lambda=\lambda_t}\} + \Im\{\nabla_{\lambda}\phi|_{\lambda=\lambda_t}\}\right]} \tag{5.4}$$

El uso de las variables complejas en la aproximación del tamaño de paso beneficia el proceso de separación de las fuentes.

La simulación mostrada en la Figura 5.2 muestra el efecto del paso en el SER (Symbol Error Rate) frente al valor estimado para $n=m=8$ fuentes y sensores. SNR se ha fijado a 15 dB.

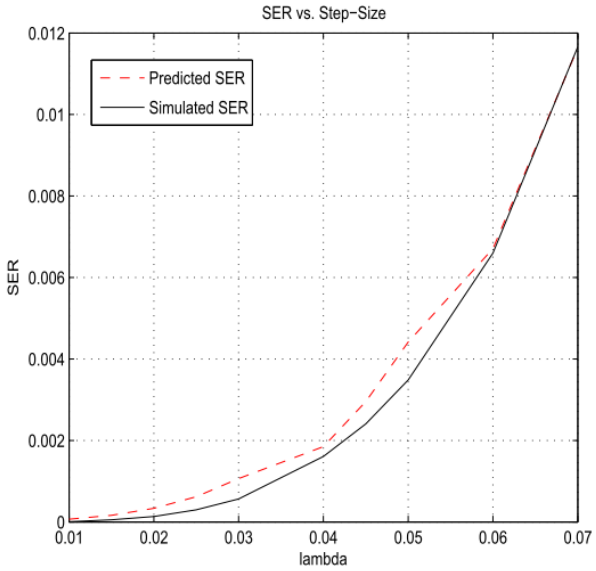


Figura 5.2 Simulación muestra el efecto del tamaño del paso en SER(symbol error rate)

5.2 Algoritmos MeR-MaID-SIG y SIPEX-G

Este es un algoritmo de separación ciega de fuentes en línea para la separación de mezcla instantánea de fuentes independientes desconocidas y variables con el tiempo. Este procedimiento utiliza la información teórica del criterio MeR-MaID-SIG y un algoritmo PCA on-line, referido al SIPEX-G. Sus resultados indican que con la combinación de los dos criterios on-line es posible rastrear cambios rápidos en la matriz de mezcla.

La separación de mezclas instantáneas no es un modelo muy realista para muchas mezclas reales. Hay muchas variaciones en el modelo de mezcla que pueden ser una solución a este problema: la suma del ruido al modelo, la posibilidad de que la mezcla que está poco determinada (por tener pocas observaciones para el número de señales existentes), el uso de arquitecturas para la obtención de las fuentes de las mezclas convolutivas y/o no lineares, y que las mezclas varíen con el tiempo.

Las mezclas que varían con el tiempo son muy importantes para los audífonos. Esto es debido al hecho de que la matriz de mezcla puede variar rápidamente. Conforme la matriz de mezcla cambia más rápidamente, para una frecuencia de muestras fija, menor es el número de datos hábiles para estimar los coeficientes para obtener las fuentes originales de las mezclas. Por esto, no resulta muy adecuado emplear *k*-*N* muestras para estimar la actualización *k* de los coeficientes para la obtención de las fuentes originales de las

mezclas para un tiempo *k*, partiendo de que las muestras estáticas de las observaciones pueden cambiar drásticamente en las últimas *N* muestras. Los algoritmos MeRMaID y MeRMaID-SIG muestran una buena respuesta; siendo buenos para rastrear mezclas con cambios rápidos, ideal para un método on-line. Actualmente, de estos dos métodos sólo se tiene en consideración el algoritmo MeRMaID-SIG.

Si se considera que la presión producida por el sonido varía inversamente con la distancia a la fuente de sonido de cada sensor, se obtiene un modelo de mezcla instantáneo simple adecuado para audífonos, ignorando el retardo respecto a la señal de referencia, así como el efecto de la función de transferencia relacionada con la cabecera.

En este modelo, H_k es la muestra en cada momento (determinada por la frecuencia de muestreo) para la geometría de la fuente de sonido deseada. Las observaciones, x_k, están relacionadas con las fuentes, s_k:

$$\mathbf{x_k} = \mathbf{W_k^T H_k^T s_k} \tag{5.5}$$

Donde x_k y s_k son vectores de N_{x1}, H_k y W_k son matrices mezcla y s_k esféricas de *NxN*, respectivamente, y siendo *k* el tiempo, y *T* la matriz transpuesta.

Para explicar mejor este método, se muestra el ejemplo siguiente [KEN01], en el que se han considerando dos fuentes y dos sensores, y que cada sensor está localizado en cada uno de los oídos del observador, se puede mostrar el ejemplo siguiente, en el que se considera el caso donde la distancia entre los oídos es 1/6 metros y la fuente de sonido está localizada 1 metro directamente en frente y 2 metros a la izquierda del observador. Resultando la matriz H_k:

$$\mathbf{H_k^T} = \begin{bmatrix} \sqrt{\frac{144}{145}} & \frac{12}{23} \\ \sqrt{\frac{144}{145}} & \frac{12}{25} \end{bmatrix} \tag{5.6}$$

El número 12/23 se obtiene tomando el inverso de la distancia entre la segunda fuente y el oído izquierdo del observador, 1/(2-1/12), donde 1/12 es la mitad del ancho de la cabeza.

Aplicando la ecuación anterior, las fuentes estimadas, y_k, se determinan como:

$$\mathbf{y_k = R_k^T x_k y} \tag{5.7}$$

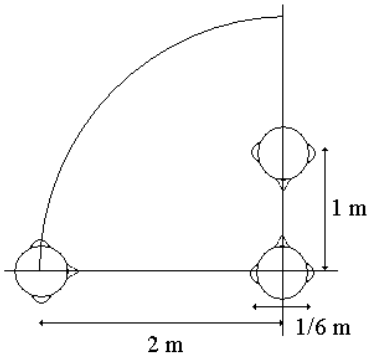


Figura 5.3 Vista superior de dos fuentes de sonido y un observador.

Donde y_k y R_k son las *Nx1* salidas y las *NxN* matrices de obtención de las fuentes, respectivamente. Para MeRMaID-SIG, la matriz R_k está ajustada para ser la rotación pura. Es más, la matriz esférica, *Wk* es idealmente $\Phi\Lambda^{-1/2}$, donde Φ la matriz de autovalores de la autocorrelación de H^T_s, y Λ es la correspondiente matriz de autovalores. Sin embargo, el valor ideal de W_k cambia a la vez que lo hace la matriz de mezcla, por lo tanto, se debe adaptar continuamente al igual que la matriz de obtención de las fuentes. Se observa en la figura este proceso para un diagrama de bloques del proceso de mezclado y obtención de las muestras.

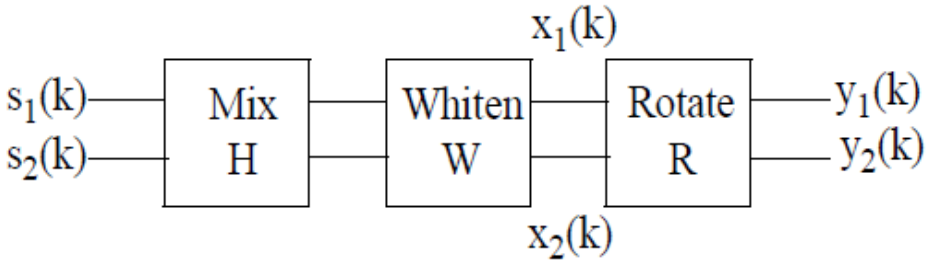


Figura 5.4 Diagrama de bloques para BSS de N=2 fuentes y observadores.

Hay dos algoritmos que constituyen este método, el SIPEX-G, que es un criterio on-line de actualización de la matriz esférica, y el algoritmo MeRMaID-SIG que es un método de actualización de la matriz de obtención de las fuentes.

El algoritmo SIPEX-G considera que la matriz esférica es el producto de una matriz de autovalores, la cual es ortonormal, y la raíz cuadrada inversa de la correspondiente matriz de autovalores, la cual es diagonal. Ignorando la matriz diagonal por el momento, es fácil observar que las observaciones pueden hacerse ortogonales para cada otra usando sólo la matriz ortonormal, la cual puede ser implementada empleando tan sólo la rotación de Givens. Además, desde la determinación de la matriz esférica empleando sólo técnicas de segundo orden, resultando una función de coste simplemente como:

$$\mathbf{J} = \sum_{\mathbf{i=1}}^{\mathbf{N-1}} \boldsymbol{\gamma}_{\mathbf{i}} \mathbf{Var}(\mathbf{y_i}) \tag{5.8}$$

Donde $V_{ar}(y_i)$ es la varianza de la i-ésima salida y γ_i es la ganancia correspondiente para dos.

Este método converge más rápidamente que otros métodos (convergencia simultánea para todas las componentes principales). Otro ajuste debe hacerse para hacer esférico el dato, esto es, escalar cada salida para tener varianza unidad. Los factores de escala, los cuales son la raíz cuadrada inversa de los autovalores de la matriz de covarianza de la observación no esférica, siendo computados por el algoritmo SIPEX-G y por lo tanto, no requiriendo más computación.

Por otro lado, el algoritmo MeRMaID-SIG es un método no paramétrico de la teoría de la información que permite minimizar la información mutua cuadrática de Renyi entre las salidas, siendo el coste de la función:

$$\mathbf{H_{R_2}}(\mathbf{y_i}) = -\mathbf{log} \int_{-\infty}^{\infty} \mathbf{f_{y_i}}(\mathbf{y_i})^{\mathbf{2}} \mathbf{dy} \tag{5.9}$$

Donde $f_y(y_i)$ es el pdf marginal de la i-ésima salida. Cuando la ventana de Parsen es empleada con kernels gaussianos, la expresión resultante para la entropía marginal de Renyi es:

$$\mathbf{H_{R_2}}(\mathbf{y_i}) = -\mathbf{log} \frac{\mathbf{1}}{\mathbf{L^2}} \sum_{\mathbf{j=1}}^{\mathbf{L}} \sum_{\mathbf{k=1}}^{\mathbf{L}} \mathbf{G}(\mathbf{y_i(f)} - \mathbf{y_i(k)}, \mathbf{2\sigma^2}) \tag{5.10}$$

Donde $G(x, \sigma^2)$ es el pdf gaussiano evaluado de x, haciendo cero la mantisa y una varianza de σ^2, y L es el tamaño del bloque. La expresión anterior pertenece al algoritmo MeRMaID original, el único cambio requerido para producir el algoritmo MeRMaID-SIG es reducir uno de los sumandos de la ecuación y sumar consecutivamente los diferentes ejemplos. Obteniendo:

$$\mathbf{H_{R_2}(y_i) = -\log \frac{1}{j=1} G\left(y_i(j)\right) - y_i(j-1), 2\sigma^2)}$$

(5.11)

Indicar el tamaño del bloque, L, en esta ecuación tan sólo representa el número de muestras anteriores al resultado acumulado, aplicado a los pesos.

5.3 Algoritmo ICAdec

Este algoritmo está orientado a la separación ciega de fuentes con mezcla no estacionaria mediante ICA usando la **transformada de wavelets**. Antes de comenzar a hablar de este método es necesario aclarar que la transformada de wavelet es una variante de la transformada de Fourier. Representa una señal en términos de versiones trasladadas y dilatadas de una onda finita.

Todas las transformaciones de wavelets se consideran formas de representación en tiempo-frecuencia, o sea, relacionadas con el análisis armónico. Las transformadas de wavelets son un caso particular de filtro de respuesta finita al impulso. Las wavelets, continuas o discretas responden al principio de incertidumbre de Hilbert (principio de incertidumbre de Heisenberg), tal que el producto de las dispersiones obtenidas en el espacio directo y en el de las frecuencias no puede ser más pequeño que una cierta constante geométrica.

El presente algoritmo es empleado para resolver el problema de mezclado estático basado en la representación de wavelets de las señales. Permitiendo estimar el desmezclado partiendo desde un número pequeño de muestras.

Este método usa la representación de wavelets de los datos para hacer el ICA más efectivo, siendo incorporado dentro de un algoritmo basado en una ventana deslizante para

tratar con el problema de separación ciega de fuentes en el caso en el que el proceso de mezclado no sea estacionario. Esta ventana deslizante es capaz de rastrear el proceso de mezclado dinámico y no estacionario en la separación ciega de fuentes, y se va calculando basándose en una estimación suave del proceso de desmezclado teniendo en cuenta las ventanas previamente obtenidas, es decir, realiza un proceso que conlleva un aprendizaje iterativo. Cuando el proceso de mezclado no es estacionario, la ventana deslizante proporcionando resultados útiles, aportando la aproximación de las wavelets una mejor ejecución del mismo.

Si se aplica la transformada de wavelets al algoritmo ICA parece razonable suponer que si se tiene una representación para las fuentes que se desea obtener, de manera que en algunas capturas o modelos de una estructura en estas fuentes, se puede conseguir una separación ciega de fuentes más efectiva. Es más, incluso con pocas iteraciones se consigue obtener un notable beneficio, siendo catalizador del desmezclado de un número mayor de muestras de posibles sensores. Para una pequeña representación tiene una media de la mayoría de los componentes para una señal cercana a cero, y con un número pequeño de coeficientes significativos se consiguen decodificar la mayoría de la información sobre la señal. Se puede plantear una pequeña representación, dibujando los coeficientes representados partiendo de una extremadamente pesada distribución de cola. Dichas distribuciones están muy alejadas de las gaussianas y, por lo tanto, son más fáciles de separar para ICA. Se puede considerar que en una buena representación en el entorno ICA los coeficientes para las señales buscadas tienen una distribución de cola pesada.

Las wavelets proporcionan pequeñas representaciones para muchas clases de señales naturales. Cuando las fuentes han sido separadas con ICA muestran que las wavelets son una representación válida para las señales involucradas en muchos problemas donde es útil ICA. La representación de wavelets de las fuentes resulta muy ventajosa para ICA.

El algoritmo ICA para el caso de la mezcla estática quedaría como sigue:

- Aplicar la transformada discreta de wavelets al dato para obtener los coeficientes de las wavelets.
- Aplicar un algoritmo ICA para los coeficientes de wavelets y así obtener los coeficientes de desmezclado.
- Aplicar la transformada inversa discreta de las wavelets para los coeficientes desmezclados para obtener estimaciones de las fuentes originales.

Se ha utilizado el algoritmo ICAdec, basado en especificar las distribuciones prioritarias para las fuentes. Éste busca la matriz de desmezclado que maximiza la verosimilitud del dato partiendo de todas las matrices de desmezclado que decorrelacionan el dato. Existiendo muchas posibles elecciones de distribuciones de las fuentes, incluyendo la distribución exponencial generalizada..

Dividiendo el dato en un número de ventanas superponiéndose y un ICA estándar y estacionario, se estima la matriz de desmezclado para cada ventana. Considerando que el proceso de mezclado cambia suficientemente despacio como para poder encontrar una longitud de ventana que lo bastante pequeña como para que la mezcla pueda ser aproximada como estacionaria dentro de ella, aunque lo suficientemente grande como para proporcionar suficientes muestras de datos para ICA como para que sea útil en el proceso de aprendizaje iterativo de desmezclado.

El algoritmo opera como se indica:

1. Se elige una longitud de ventana, L, y un tamaño de medida de paso, $d<L$.
2. Se aplica ICA para una ventana con los primeros L puntos de datos.
3. Se mueve la ventana sobre d puntos de datos y se aplica ICA para la nueva ventana de datos, usando el método inicial.
4. Despermutar las fuentes obtenidas.
5. Repetir los pasos 3 y 4 hasta completar la obtención de todas las fuentes.

El algoritmo ICAdec requiere una matriz o ventana inicial para empezar el proceso de obtención de la matriz de desmezclado óptima. La matriz de inicialización ideal es la que varía sólo lentamente, con lo que la estimación de la matriz de mezcla obtenida desde la ventana previa está cerca de la matriz de mezclas óptima para la nueva ventana. Por lo tanto, para hacer que el algoritmo sea más eficiente y exacto, se inicializa el algoritmo con una matriz relacionada con las salidas de la ventana previa.

No se puede considerar la ventana previa como inicialización para la nueva ventana, debido a que la matriz de inicialización deberá tender varias dependiendo del dato que cambia desde una ventana a la siguiente, no existiendo garantía de la matriz de desmezclado estimada para una ventana, sino, como colector para la siguiente. Pudiéndose proyectar la salida desde la ventana previa como base para la siguiente ventana, usando ésta como la nueva matriz de inicialización.

Se puede observar como una matriz arbitraria, *W*, sobre las múltiples decorrelacionadas. Siendo *X* una matriz tal que la i-ésima fila contiene las observaciones del sensor *i*, y $X=U\sum V^T$ el valor singular de la descomposición de *X*. O sea, *U* y *V* son matrices unitarias y $\sum$ una matriz diagonal. La matriz *M* está decorrelacionada para *X* si y sólo si tiene la forma:

$$\boldsymbol{M = DQ \sum M^{-1} U_t} \tag{5.12}$$

Donde *D* es diagonal y *Q* es ortogonal. Para simplificar los cálculos se considera que *D* no cambia mucho al pasar de una ventana a la siguiente. Tomando el valor de *D* para que sea igual al obtenido para la ventana previa. Manteniendo así una matriz ortogonal *Q* que minimiza:

$$\boldsymbol{\left\| W - DQ \sum^{-1} U^T \right\|^2} \tag{5.13}$$

Esto facilita el cálculo de la descomposición del valor singular *de* $D^{-1}WU\sum^{-1}$. Hacer *Y* y *Z* matrices unidad y *C* diagonal tal que $D^{-1}WU\sum^{-1}=YCZ^T$, siendo el valor de *Q* que minimiza la expresión anterior:

$$\boldsymbol{Q = YZ^T} \tag{5.14}$$

Incluso utilizando la proyección de la última sobre la matriz de desmezclado, el algoritmo no converge a una solución. Para que lo haga, se estima una matriz de desmezclado como base para la inicialización. Hay que hacer que W_t sea la estimación de la matriz de desmezclado para la ventana *t*, para posteriormente usar $1/2\ (W_{T-2} + W_{T-1})$ como la matriz de inicialización para la nueva ventana en el tiempo *t = T*.

En todos los algoritmos de ICA las fuentes recuperadas son las salidas en un orden aleatorio, siendo posible que en la presente ventana se recuperen en un orden diferente de la de la ventana previa.

Si se elige el tamaño del paso mucho menor que la longitud de la ventana, a continuación, dos ventanas consecutivas tendrán una zona de superposición en la que

ambas producen una estimación para la misma fuente. Hay que hacer que a_1 sea una de las fuentes recuperadas desde la primera ventana y a_2 sea una de las fuentes recuperada desde la segunda ventana. Suponiendo que en ambas ventanas ICA recupera las fuentes con una precisión razonable, entonces, si a_1 y a_2 se corresponden con la misma fuente subyacente su correlación debe estar muy cerca de cualquiera más o menos uno. Si se corresponden a diferentes fuentes entonces su correlación estará muy cercana a cero. Por lo tanto, las fuentes pueden ser despermutadas por la elección de la permutación que maximiza la suma de los valores absolutos de las correlaciones entre las fuentes recuperadas en la región de solapamiento. Una vez que las fuentes han sido despermutadas, el signo de la fuente estima que puede ser invertida si fuera necesario para asegurar que las correlaciones son de hecho positivas.

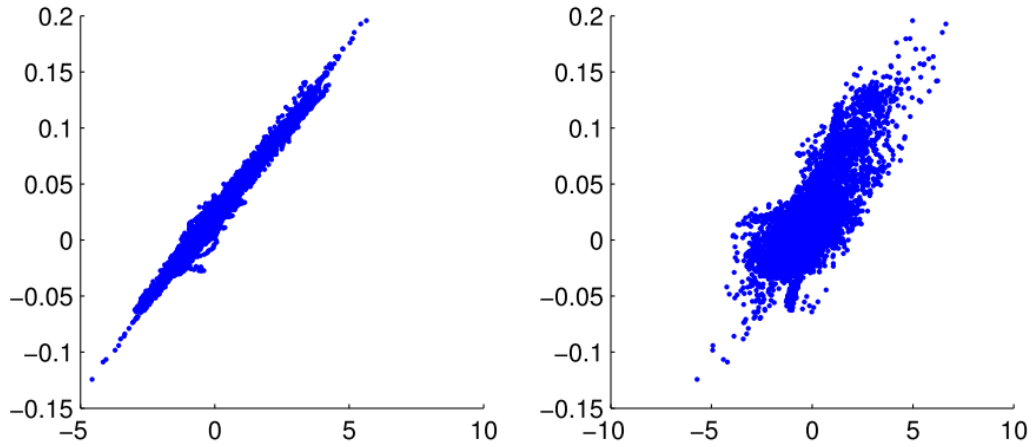


Figura 5.5 Relación entre una de las entradas originales y la componente extraída en ICAdec e ICAstat.

5.4 Algoritmo ASWICA (Adaptive Sliding Window ICA)

En este trabajo se presenta un modelo adaptativo de ventanas deslizantes basado en ICA (Algoritmo ASWICA (Adaptive Sliding Window ICA)) para resolver el problema de la mezcla no estacionaria. Este modelo se ajusta bien en el caso de cambios lentos de dicha mezcla lo que permite extraer valores que permitan extraen una estimación de su evolución. Cuando la mezcla de señales cambia dinámicamente el problema es computacionalmente más complejo para resolverlo en tiempo real. El algoritmo adaptativo permite ajustarse a la capacidad computacional del sistema de forma que puede cambiarse

tanto el paso como el tamaño de la ventana permitiendo que procesadores de señales digitales, DSPs, más potentes puedan obtener mejores aproximaciones.

El algoritmo tiene cuatro fases principales que se muestran en la Figura 5.6. En la primera se procesa con FastICA una parte del vector de la señal de entrada, lo que permite obtener una primera aproximación de la matriz de mezcla. Esto plantea un problema ya que el algoritmo FastICA partiendo de los mismos datos iniciales puede obtener soluciones distintas pues la matriz resultante puede tener sus componentes desordenadas y con distinta amplitud, impidiendo que puedan considerarse sólo la secuencia de los valores obtenidos.

La segunda fase normaliza y ordena, por proximidad con los valores obtenidos previamente, los valores obtenidos de la matriz de mezcla. Esto reduce el número de permutaciones que deberían estimarse de los valores de la matriz de mezcla.

La tercera fase hace un control adaptativo que realiza dos funciones: estimar mediante un spline cúbico los elementos de la matriz de mezcla y estimar, en función de los tiempos de ejecución, el paso de la ventana deslizante de la fase primera.

La fase de extracción utiliza el spline cúbico obtenido en la fase anterior lo que le permite modificar la matriz de mezcla para cada muestra.

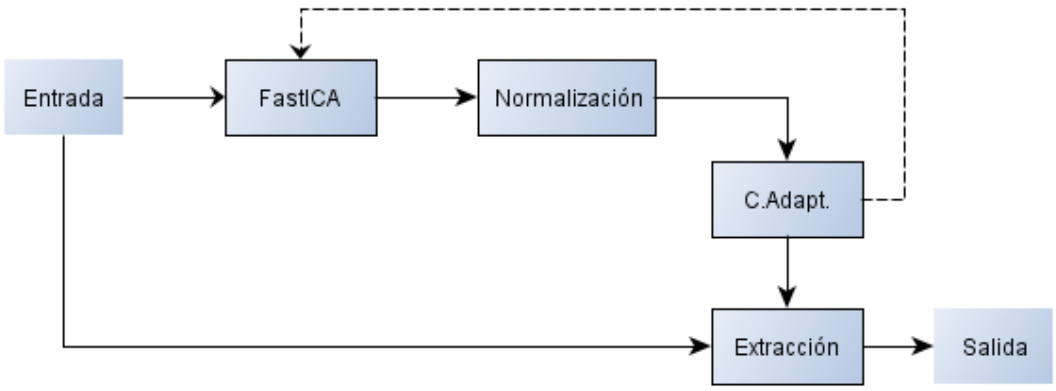
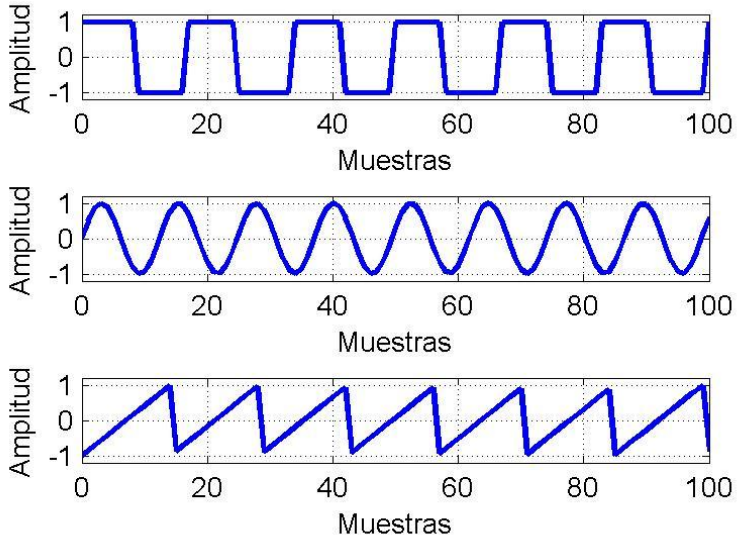


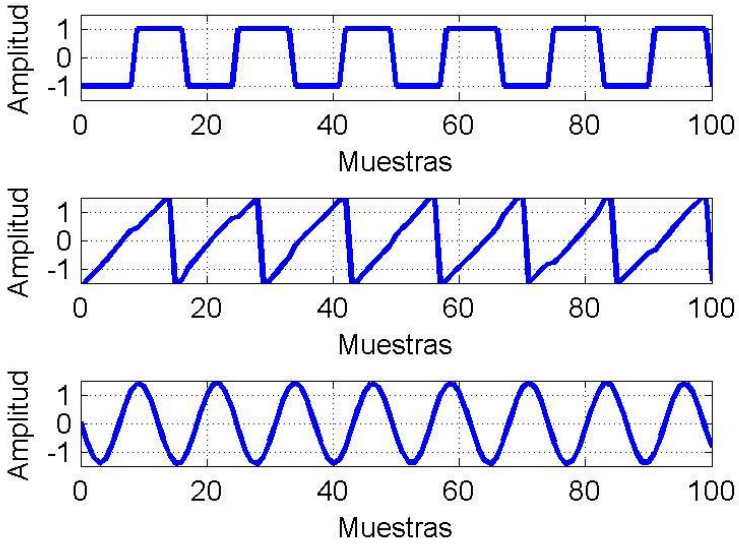
Figura 5.6 Diagrama de bloques del algoritmo ASWICA

5.4.1 Estimación de tamaño óptimo de ventana

Un parámetro importante es el tamaño de la ventana deslizante sobre la que se calcula el FastICA. Aunque este algoritmo que obtiene valores óptimos con pocos ajustes adicionales es importante en sí la información de cada muestra. En el siguiente

experimento podemos observar como en afecta la frecuencia de muestreo en la separación de la señal. Supongamos el caso para tres señales de entrada, una senoidal de 810 Hz, un diente de sierra de 707 Hz y una señal cuadrada de 605 Hz que están muestreadas a 10 KHz. La Figura 5.7 muestra las señales originales (a), las señales separadas (b) y las señales mezcladas (c). Se obtienen los coeficientes de correlación: -1, -0,9999 y 0,9955.





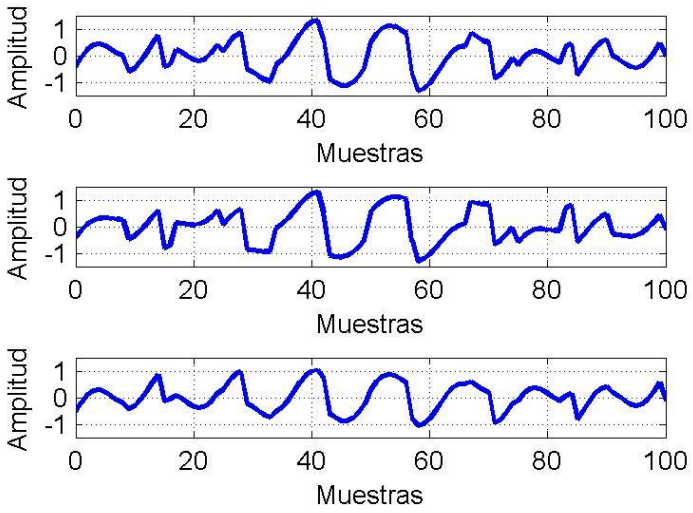
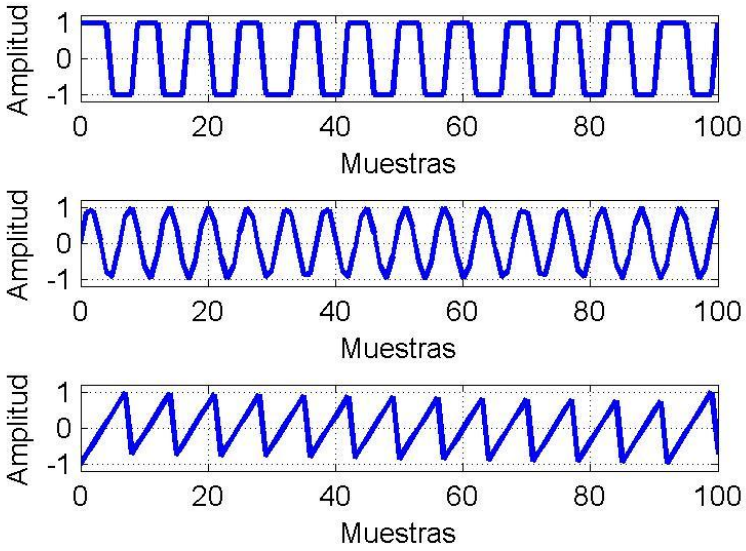


Figura 5.7 Separación de señales con FastICA muestreadas a 10 KHz.

El siguiente experimento se mantiene el número de muestras para comprobar cómo afecta la frecuencia de muestreo en la información que lleva implícita cada muestra. En la Figura 5.8 se muestra sólo la señal original y separada para 5 KHz. Como puede observarse la señal separada está algo más distorsionada obteniendo unos peores coeficientes de correlación: -0,9996 , 0,9999 y 0,9949.



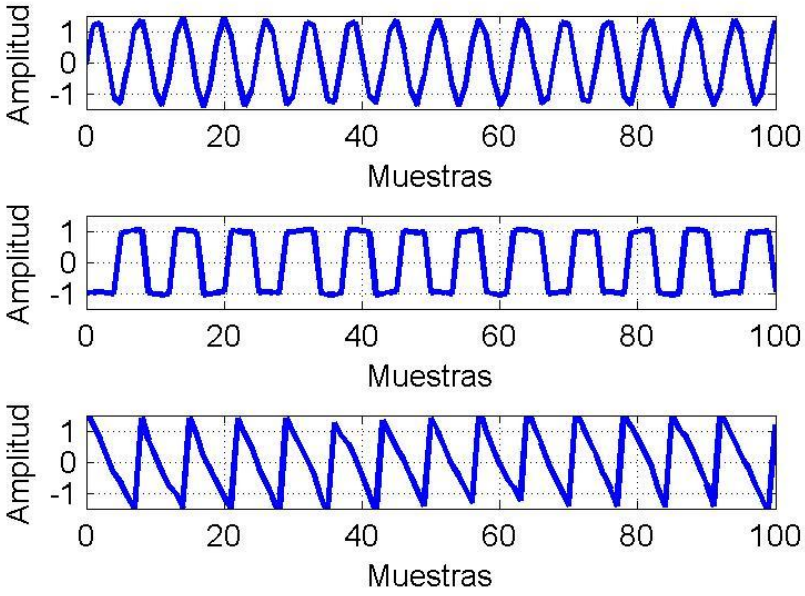


Figura 5.8 Separación de señales con FastICA muestreadas a 5 KHz.

En la Figura 5.9 se muestra la señal original y la separada para 20 KHz. Aunque el muestreo es el doble que a 10 KHz, apenas muestra mejora y los coeficientes de correlación son: -1, -0,9999 y 0,9999.

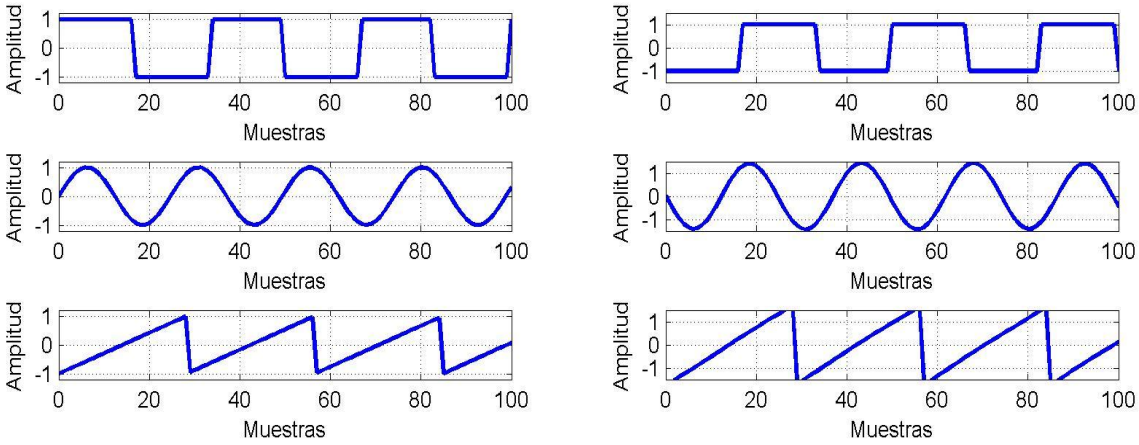


Figura 5.9 Separación de señales con FastICA muestreadas a 5 KHz.

La Tabla 5.1 muestra los coeficientes de correlación obtenidos para las diferentes señales en función de la frecuencia de muestreo y con igualdad del número de muestras. Aunque la que obtiene mejor correlación es la de 20 KHz, la de 5KHz también obtiene unos buenos coeficientes de correlación.

Fmuestreo	Coeficientes de correlación		
	Señal cuadrada	**Señal senoidal**	**Señal diente de sierra**
5 KHz	-0,9996	0,9999	0,9949
10 KHz	-1	-0,9999	0,9955
20 KHz	-1	-0,9999	0,9999

Tabla 5.1 Coeficientes de correlación obtenidos en función de la frecuencia de muestreo.

De esta observación puede determinarse que en igualdad del número de muestras afecta la frecuencia de muestreo de la señal y esto es debido a la información que lleva implícita cada muestra, lo que determina una mejor aproximación del algoritmo. La Figura 5.10 muestra la dispersión de puntos entre la señales senoidal y diente de sierra de la señal para (a) 5 KHz, (b) 10 KHz y (c) 20 KHz. Un aumento en la frecuencia de muestreo mejora la nube de puntos y ofrece mejor aproximación.

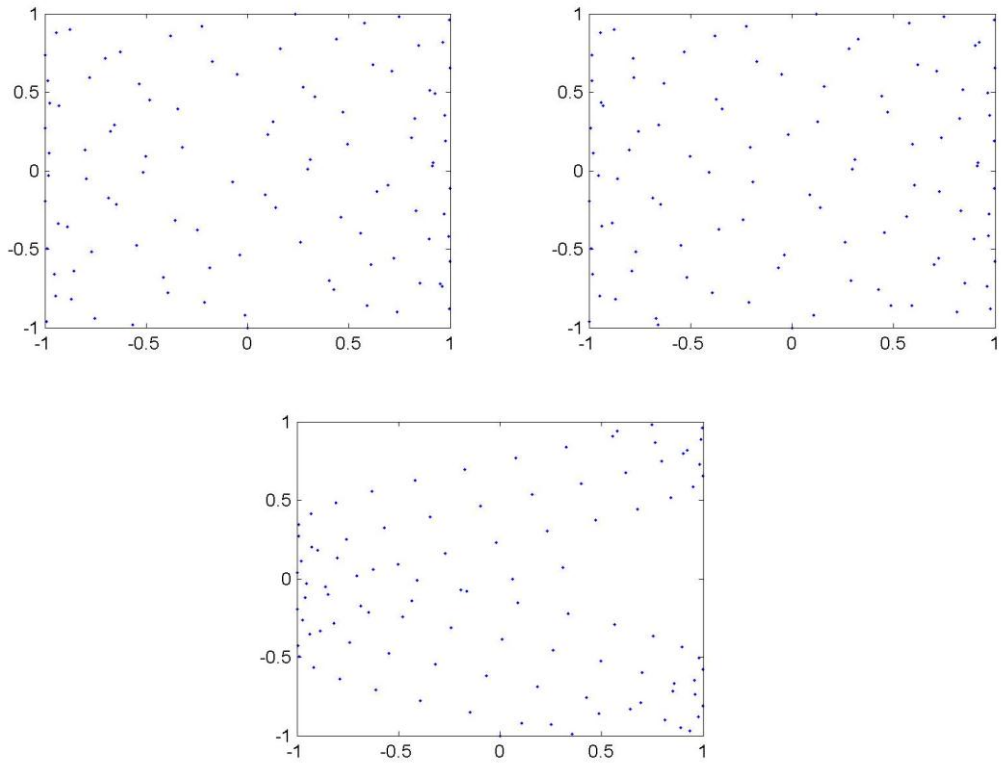


Figura 5.10 Dispersiones obtenidas para 5 KHz, 10 KHz y 20 KHz..

A partir de estos experimentos podría pensarse que el algoritmo FastICA converge rápidamente con un número reducido de muestras, como puede comprobarse en las figuras y la tabla de correlación mostradas anteriormente, pero haciendo experimentos con otras formas de onda algo más complejas se observa que los tamaños de ventana deben ser mayores.

En un nuevo experimento se van a utilizar dos señales más complejas: una compuestas por la suma de dos senoidales de 810 Hz y 550Hz, otra por dos dientes de sierra de 707 Hz y 753 Hz, muestreadas a 20KHz. La Figura 5.11 muestra la señal original y la señal separada cuando el número de muestras es 100. La correlación obtenida es 0,8396 y –0,8916.

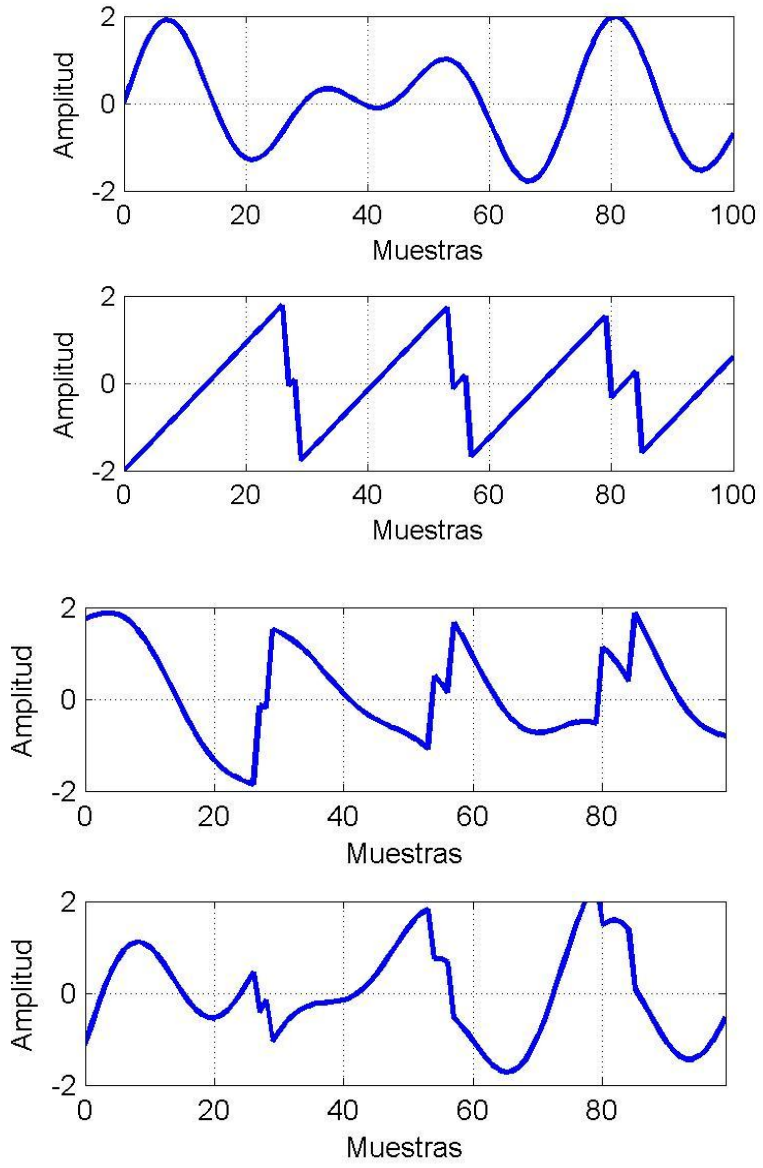


Figura 5.11 Señales originales y separadas con 100 muestras.

En cambio, cuando aumenta el número de muestras mejora significativamente la separación. En la Figura 5.12 (a) se observan las 100 primeras muestras la señal original y separada cuando el número de muestras para la separación es 1000. Los coeficientes de correlación obtenidos son 0,9992 y -0,9997. La Figura 5.12 (b) muestra el caso con 4096 donde los coeficientes de correlación son 0,9999 y 0,9999.

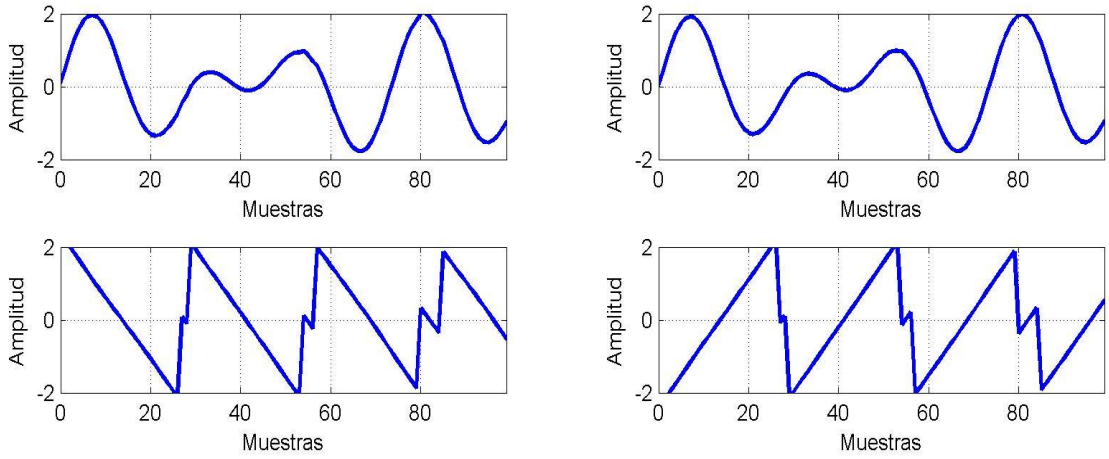


Figura 5.12 Las 100 primeras muestras de las señales originales y separadas utilizando 1000 y 4096 muestras.

Número de muestras	Coeficientes de correlación	
	Señal seno doble	Señal diente de sierra doble
100	0,8396	-0,8916
1000	0,9992	-0,9997
4096	0,9999	0,9999

Tabla 5.2 Coeficientes de correlación obtenidos en función del número de muestras.

La Tabla 5.2 muestra los diferentes coeficientes de correlación obtenidos en el experimento.Teniendo en cuenta estos resultados, es razonable considerar tamaños de ventana más grande para que las estimaciones sean mejores, lo que provocará un mayor coste computacional. Un valor de 4000 muestras ya es un valor bastante bueno y teniendo en cuenta las optimizaciones posteriores para el DSP es aconsejable que el tamaño de dicha ventana sea potencia de 2, por lo que se utilizará 4096 como un tamaño aconsejado de ventana para poder procesar señales de diverso tipo.

En los ejemplos propuestos en los algoritmos MerNaID-SIG y SIPEX-G se considera que la presión producida por el sonido en cada sensor varía inversamente con la distancia a la fuente de sonido, se obtiene un modelo de mezcla instantáneo. Supónganse dos fuentes que se desplazan de forma sinusoidal (con frecuencia 0,5 Hz) delante de un par de sensores que se encuentran delante a 1 m de distancia y se desplazan $\pm 1m$, produciendo cada una dos señales de unos cientos de hertzios. La Figura 5.13 muestra los coeficientes y la amplitud recibida por uno de los sensores.

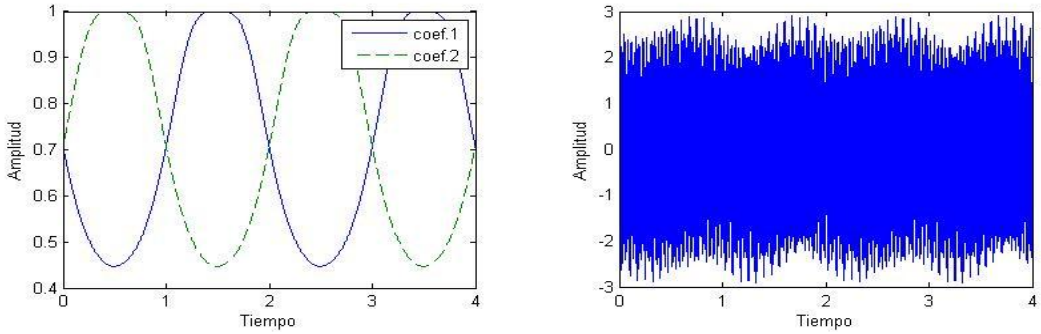


Figura 5.13 Valores de la matriz de mezcla y amplitud de la señal recibida.

Como se ha comentado anteriormente, el tamaño de la ventana afecta significativamente a la estimación obtenida por el FastICA. I, La Figura 5.14 muestra la salida obtenida de la Fase II en función del tamaño de ventana, siendo 5000 muestras en el primer caso y 1000 muestras en el segundo. La línea discontinua representa el valor de uno de los componentes de la matriz de mezcla y la línea escalonada el valor estimado resultante de la fase II. Reducir considerablemente el tamaño de dicha ventana provoca un mayor error en la estimación de dichos coeficientes de la mezcla.

En condiciones reales, la matriz de mezcla va cambiando prácticamente en cada muestra, ya que si proviene de una fuente que está en movimiento implicará un cambio progresivo en los coeficientes de mezcla. La fase III del algoritmo se encarga de estimar el carácter cambiante de dichos coeficientes. Suponiendo que los coeficientes cambian de forma continua, y sin cambios excesivamente rápidos, la estimación basada en spline cúbicos ofrece una buena aproximación.

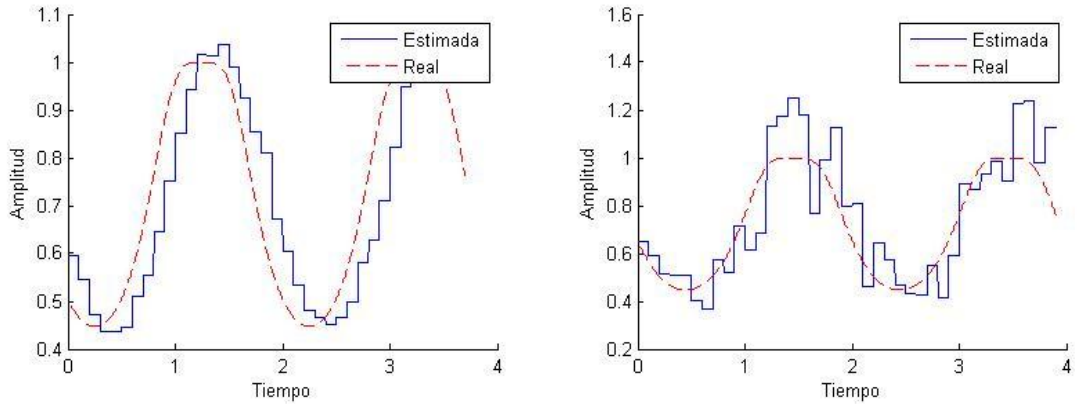


Figura 5.14 Relación entre el tamaño de la ventana y la estimación.

Partiendo del entorno anterior donde hay un desplazamiento de las fuentes a 1m de distancia de los sensores y, considerando la amplitud como la inversa de la distancia, la Figura 5.15 muestra la salida obtenida en la Fase III para un movimiento senoidal y otro triangular. También puede observarse en dicha figura que en el caso del desplazamiento triangular, cuando el emisor está cerca de los sensores no se aprecia una forma triangular en la amplitud real y es debido a que dicha amplitud depende de la distancia, por lo que en la parte superior aparece más redondeada.

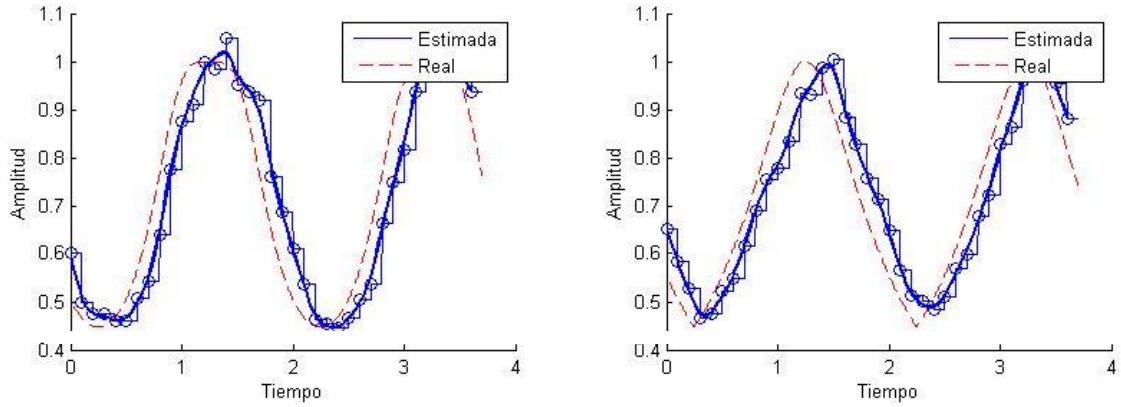


Figura 5.15 Salida de la Fase III para un movimiento senoidal y un movimiento triangular.

Y a que la estimación está retrasada con respecto a las muestras originales, la Fase IV de extracción combina la estimación obtenida en la Fase III con las muestras retrasadas el tamaño de la ventana, lo que permite correponder los datos con los coeficientes más aproximados. La Figura 5.16 muestra la superposición entre la real y estimada cuando se aplica el retraso en la Fase IV para tamaños de ventana de 5000 y 1000 muestras respectivamente. Cabe destacar que, como se indicó anteriormente, si el tamaño de la

ventana es menor provoca mayor indeterminación de los puntos de estimación para el spline cúbico provocando una peor estimación global.

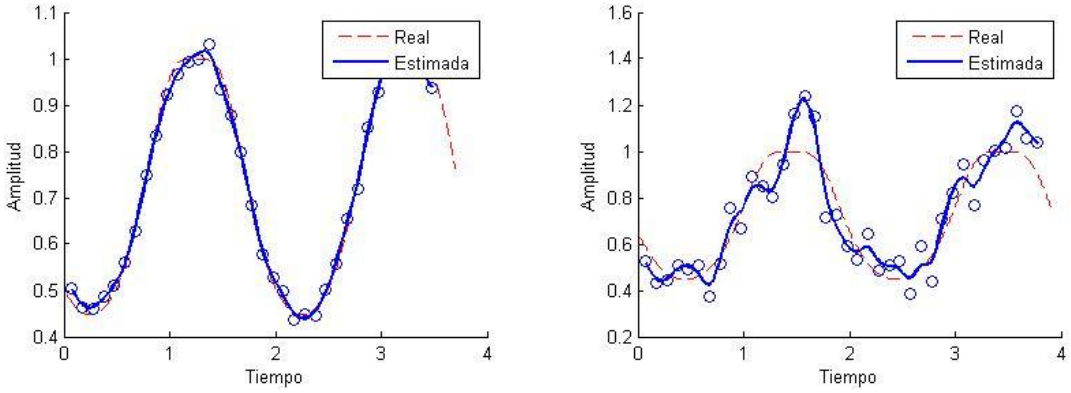


Figura 5.16 Estimación aplicada en la Fase IV

5.5 Algoritmo ADSWICA (Adaptive Dual Sliding Window ICA)

Este modelo combina la estimación de dos ventanas de distinto tamaño, para obtener un mayor número de puntos para estimar mejor los coeficientes que se utilizan realmente para la conversión.

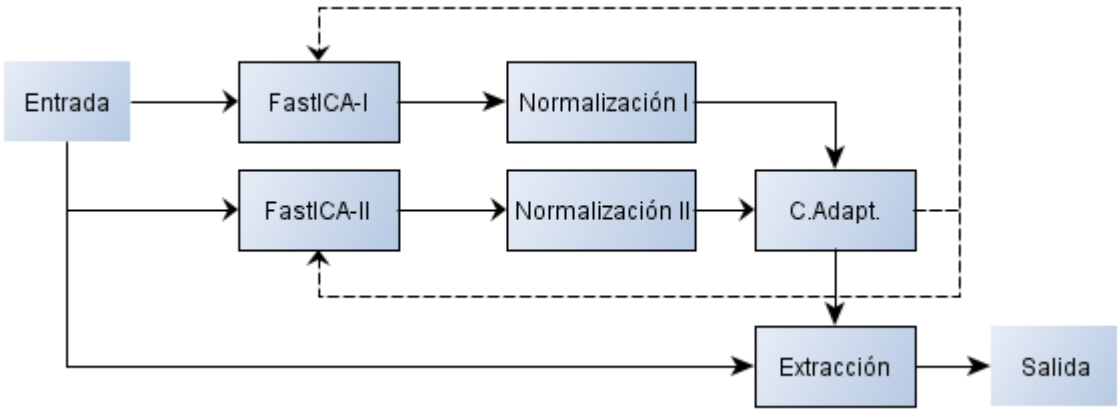


Figura 5.17 Diagrama de bloques del algoritmo ADSWICA

El control adaptativo se basa en los valores obtenidos por los dos FastICA de tamaños distintos. La ventana de mayor tamaño representa la estimación lenta y la ventana de menor tamaño detecta los cambios más rápidos. En este caso se aplican dos ventanas, una de 4000 muestras y otra de 7000 muestras que estima cambios un poco más lentos en la señal, de forma que la combinación de todos los puntos permite una mejor estimación final.

La Figura 5.17 muestra el diagrama de bloques del algoritmo ADSWICA. En este caso son necesarias dos ventanas donde se procesa simultáneamente el FastICA así como las correspondientes etapas de normalización. La Figura 5.18 muestra la salida obtenida para ventanas de 4000 y 7000 muestras respectivamente.

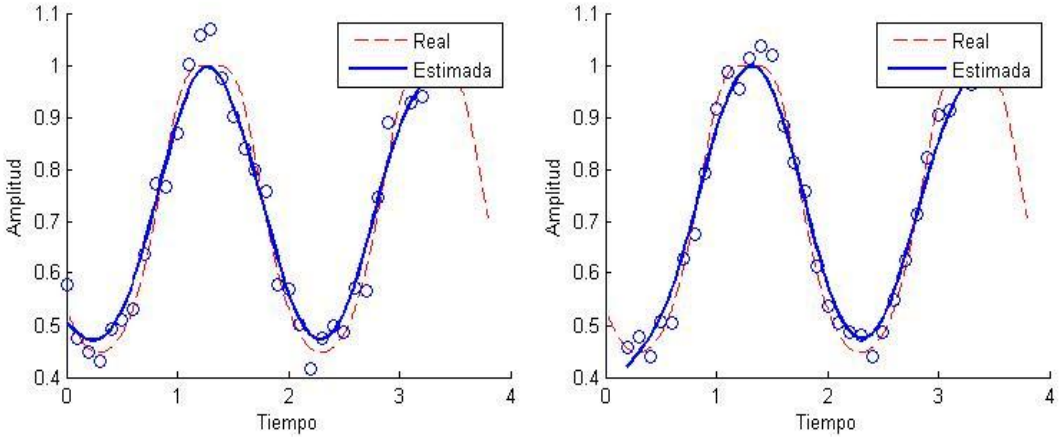


Figura 5.18 Estimaciones de la ventana de 4000 y 7000 puntos en la Fase II.

Combinando las salidas de ambos se obtiene una nube de puntos más densa con mayor cantidad de información ya que incorporará elementos tanto de mayor frecuencia como de menor frecuencia de la señal a estimar lo que permite mejorar la estimación final. La Figura 5.18 muestra la componente real y estimada que va cambiando con el tiempo y cómo esta señal se aproxima mejor que el algoritmo basado en una única ventana.

La Figura 5.19 muestra como la señal estimada se aproxima más a la real ya que la nube de puntos para la estimación cúbica es mayor. Computacionalmente requiere más tiempo ya que la aproximación cúbica requiere más operaciones.

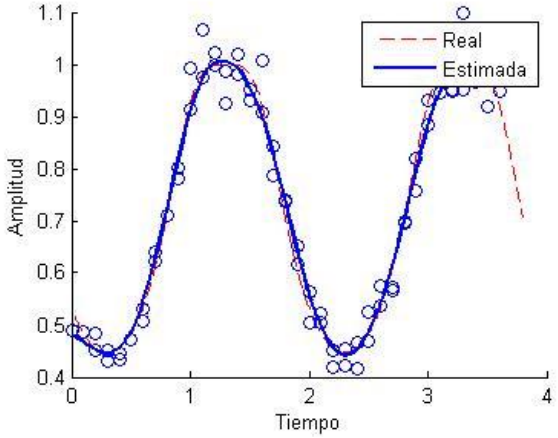


Figura 5.19 Señal real y estimada aplicando el algoritmo ADSWICA.

5.6 Conclusiones

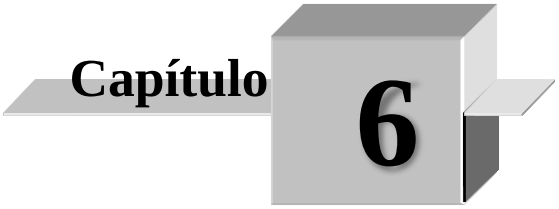
La separación ciega de fuentes con mezcla no estacionaria permite abordar problemas reales que ocurren de forma frecuente en la naturaleza ya que las condiciones de captura de las señales así como el entorno donde se suelen propagar pueden ser dinámicas y de carácter cambiante.

En este capítulo se han mostrado algunos de los algoritmos que pueden encontrarse en la bibliografía a tal fin y, estudiando las capacidades del DSP para el procesamiento de las señales en tiempo real se han propuesto dos algoritmos: ASWICA y ADSWICA, basados en ventanas deslizantes. Para la estimación inicial de los coeficientes se utiliza FastICA por ser bastante rápido en la convergencia así como por no necesitar parámetros adicionales, facilitando una búsqueda óptima de la solución.

La escasa bibliografía que aborda este problema no especifica el método ICA utilizado y asume un carácter pseudo-estático de los coeficientes. Los nuevos algoritmos desarrollados son más precisos, ya que no sólo estiman los coeficientes de la matriz de mezcla, sino que en su lugar, utilizan en una función cúbica que cambia tras cada muestra, de igual forma que en un entorno real los coeficientes cambian de forma progresiva y continua.

Los nuevos algoritmos desarrollados en este trabajo, permiten obtener una buena aproximación de los coeficientes de la matriz de mezcla cuando hay un número suficiente de muestras, si bien presentan un mayor coste computacional que los algoritmos existentes

en la bibliografía. Este incremento en el coste computacional queda claramente compensado con la calidad y precisión en las señales obtenidas tras la separación.



Resumen:

En este capítulo se presentan cuestiones relacionadas con la optimización del código para DSP de altas prestaciones así como los resultados del análisis, implementación y optimización de los algoritmos ICA, ASWICA y ADSWICA en DSP. También se presenta un sistema automático de optimización de código en DSPs para los algoritmos implementados anteriormente.

6 IMPLEMENTACIÓN DE LOS NUEVOS ALGORITMOS EN DSP

Los DSPs son bastante sensibles a que el código esté adecuadamente optimizado a su arquitectura interna. Así, para conseguir que ejecuten eficientemente dicho código es necesario que funcionen de forma paralela el mayor número de recursos internos, sus registros se usen correctamente, utilizar los modos de direccionamiento correctos y cuando tienen memoria cache interna que ésta tenga la mayor tasa de aciertos posible.

Por este motivo, el código que debe ejecutar un DSP no es una simple copia del código que puede ejecutarse en otras plataformas como es un computador de propósito general ya que o bien carecen de elementos específicos para el procesamiento de señales o bien, disponen de arquitecturas que soportan ejecución fuera de orden para optimizar en tiempo de ejecución el código. Por lo tanto, resulta imprescindible que las unidades MAC del DSP o bien sus unidades aritméticas puedan funcionar a su máximo rendimiento.

Las arquitecturas basadas en ejecución fuera de orden resultan muy eficientes para ejecutar código poco optimizado pero no pueden utilizarse en DSPs principalmente por dos motivos:

- Estos sistemas requieren un hardware mucho más complejo lo que encarecería el DSP y añadiría un consumo excesivo. Como ejemplo, la mayoría de procesadores de propósito general de Intel incorporan ejecución fuera de orden,

excepto los Intel Atom orientados a sistemas portátiles de bajo consumo por lo que necesitan reducir su tamaño y consumo.

- La ejecución fuera de orden implica una mayor indeterminación en los tiempos de ejecución, lo que contradice el modelo determinista de los DSP que pueden garantizar los tiempos de respuesta para atender interrupciones o estimaciones más reales y precisas en los tiempos de ejecución del código.

6.1 Implementación de los algoritmos en el DSP

Para esta implementación se ha utilizado un DSP de punto fijo de altas prestaciones. El TMS320C6416 es un dispositivo capaz de trabajar en punto fijo. Es un procesador de 32 bits de segunda generación, con palabras de instrucción muy largas (VLIW), capaz de ejecutar más de 8000 millones de instrucciones por segundo (MIPS) y con una velocidad de 1 GHz. Tiene la flexibilidad operacional de controladores de alta velocidad y con la capacidad de procesadores matriciales. Posee cuatro acumuladores múltiples de 16 bits (MAC) por ciclo para un total de 4000 operaciones de acumulación múltiple, con acumulación por segundo o MAC de 8 bits por ciclo para un total de 8000 MMAC. Por último, tiene dos memorias caché, una de 1056 KB y otra 1024 KB.

Se puede implementar el punto fijo programando en C y aplicando las rutinas en el Code Composer Studio (CCS), observándose la ventaja de su alta precisión aunque con un elevado coste en tiempo de ejecución. Para reducir este tiempo, manteniendo un buen nivel de precisión se puede programar manualmente el código de punto fijo. Esta optimización del tiempo se puede lograr básicamente con los siguientes pasos:

- Remplazar variables de punto flotante por variables en punto fijo, codificadas como enteros.
- Selección del formato apropiado de punto fijo para evitar el desbordamiento y reducir la pérdida de precisión.
- La implementación de operaciones aritméticas (resta, suma, división, raíz cuadrada, desplazamiento, redondeo, cambio de exponente, etc.) en aritmética de punto fijo utilizando las macros del procesador.

Recordando la representación en punto fijo, un número en punto fijo puede ser expresado como un entero multiplicado por 2 elevado a exponente negativo, esto es, $QN = Q \cdot 2^{-N}$, siendo Q la mantisa y N el exponente. También puede notarse como *M.N,* donde *M* es el número de bits enteros y *N* el de bits fraccionarios.

El rango de un número en punto fijo queda definido por su parte entera. Y su precisión es la menor diferencia entre dos números sucesivos.

El utilizar la aritmética de punto fijo añade dos problemas en la implementación:

- Problemas relacionados con el redondeo o truncamiento.
- Problemas con el desbordamiento.

Ambos problemas están directamente relacionados ya que si se utiliza un número elevado de bits para mantener precisión en las operaciones puede ocurrir que la secuencia de operaciones de producto y suma acaben produciendo un desbordamiento indeseado.

Es por ello que el tamaño de las ventanas y número del número de componentes queden fijados previamente. Además de otras optimizaciones que se describirán posteriormente, es recomendable conocer de antemano dichos factores para buscar una buena aproximación para el número de bits.

6.2 Optimizaciones para el DSP

Para que el código que se ejecute sea realmente eficiente hay que conocer la arquitectura del procesador ya que facilita la comprensión de los elementos que afectan realmente a los tiempos de ejecución.

A continuación se describen algunos elementos que afectan de forma decisiva al rendimiento en la ejecución del procesador.

6.2.1 Optimización de la organización de la memoria

Aunque los procesadores han aumentado considerablemente su velocidad y las memorias no han mejorado de la misma forma. Aunque las tecnologías de proceso avanzadas han permitido integrar en un chip tanto la CPU como parte de la memoria, el tiempo de acceso de dicha memoria no ha disminuido proporcionalmente. La memoria externa permite almacenar mayor cantidad de datos pero su tiempo de acceso es mayor que el tiempo de acceso a la memoria interna, por lo que a menudo es un cuello de botella.

Los primeros DSPs no disponían de memoria caché pero de forma progresiva los fabricantes han analizado sus ventajas y la han introducido. Los DSPs de alto rendimiento, como la familia C6000, procesan datos digitales rápidamente y, en el caso del DSP que se utiliza en este trabajo, funciona a una velocidad de reloj de 1 GHz. Para estas velocidades se requieren memorias rápidas que estén conectadas directamente a la Unidad central de procesamiento (CPU). y por eso incorporan memorias cache en su arquitectura interna. Los programadores deben tener en cuenta las características de dichas memorias para mejorar las prestaciones que pueden obtenerse.

Muchos procesadores de gama alta, incluyendo el DSP C64x, emplean una caché de dos niveles para accesos de programa y datos en el chip. En esta jerarquía existe una cache de nivel 1 para programa (L1P) y para otra para datos (L1D). Estas caches eliminan conflictos por los recursos de memoria entre los buses de programa y datos, ya que pueden accederse simultáneamente. Estas cachés L1 funcionan a la misma velocidad que la CPU permitiendo accesos rápidos. Ambas memorias están conectadas a una memoria mayor, L2, que también está en el chip y contiene datos y programa. Esta memoria L2 unificada proporciona asignación de memoria flexible entre el programa y los datos de accesos fuera de L1. La memoria L2 actúa como puente entre la memoria L1 y memoria que se encuentra fuera del chip.

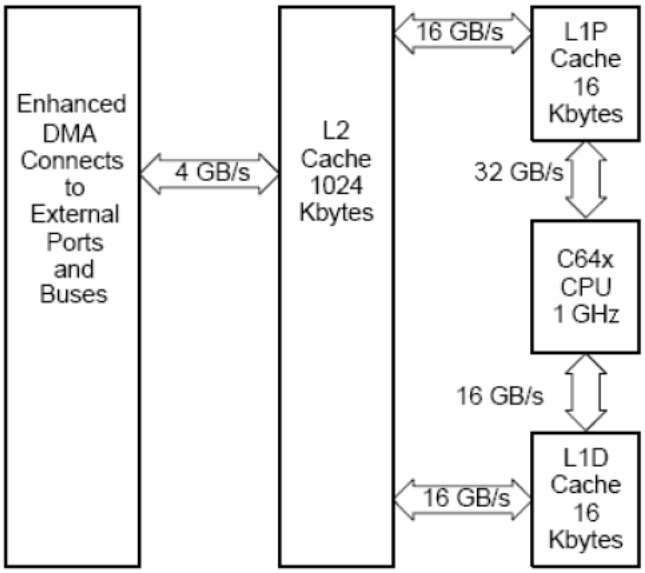


Figura 6.1 Anchos de banda entre las memorias cache internas de la familia C64x de DSPs.

Como se ha indicado anteriormente, la arquitectura VLIW del DSP no ejecuta instrucciones fuera de orden de forma que cuando se produce un fallo de caché (de no estar el dato disponible), el procesador espera hasta que esté disponible el dato. Supóngase el caso de la L1P, si el dato no está en la L1P pero sí en L2, los datos del programa se copian en L1P y, a continuación, se envía al procesador. Si el valor no está en L1P y tampoco está en L2, los datos se copian de memoria externa a L2, y luego se copian en L1P y son enviados al procesador. La arquitectura de C64x proporciona un mecanismo para aumentar la eficiencia de la caché cuando existen múltiples errores de caché L1P/L1D. Esta técnica se conoce como "mispipelining". Si los errores se producen agrupados, la recuperación de las líneas perdidas puede superponerse. Esto aumenta la efectividad de caché ya que reduce el tiempo que se tarda en recuperar las líneas perdidas.

La cache L1P es 16 KB, de mapeo directo y tamaño de línea 32bytes, (8 instrucciones). La cache L1D es de 16 KB, asociativa de 2-conjuntos tamaño de línea de 256 bits.

La cache L2 es unificada ya que puede almacenar datos y programa. En los dispositivos C641x iniciales, el tamaño de la memoria de L2 es 1024K bytes. El gran tamaño de memoria L2 aumenta enormemente la eficacia de la memoria caché porque los fallos son mucho menos probables. Esta memoria tiene un comportamiento especial ya que una parte (256 KB) de dicha memoria puede tener doble funcionalidad: actuar como una cache asociativa de 4 conjuntos, con líneas de 64 bytes, o bien como una memoria SRAM. El resto, 768KB, se utiliza siempre como SRAM.

Una cuestión fundamental en un sistema DSP es mantener la ejecución en tiempo real. En muchos de estos sistemas, se toma una muestra de datos y procesada hasta que llega la siguiente muestra y, de esta forma, ese necesario garantizar el tiempo de ejecución, o de lo contrario podrían perderse muestras o ejecutar erróneamente los algoritmos. Ya que la cache afecta de forma decisiva en los tiempos de ejecución parece inicialmente que los algoritmos tarden el mismo tiempo en ejecutarse.

Así, en un sistema basado en memoria caché, podemos determinar una cota superior para esta pregunta. Podemos hacer la hipótesis de que la caché está vacía y que se deben buscar las instrucciones y datos de la memoria fuera de la caché. Partiendo de esta suposición, entonces se puede comprobar si hay suficiente tiempo para ejecutar el algoritmo. Este problema también existe en un modelo de memoria plana donde el programador administra la jerarquía de memoria en lugar de hacerlo con el controlador de memoria caché.

A menudo es necesario que el programador administre el flujo de datos en un sistema de memoria plana para asegurar una operación en tiempo real, teniendo en cuenta si hay múltiples canales de DMA compitiendo por los recursos, o si llegan un elevado número de interrupciones que consuman excesivo tiempo e impidan el procesamiento entre muestra y muestra. Estas cuestiones reflejan que los problemas relacionados con el determinismo son dependientes del sistema y no son exclusivas de un sistema basado de caché. El programador debe tener en cuenta todos los factores que afectarían su capacidad de mantener en tiempo real para cualquier modelo de memoria.

Como se ha comentado anteriormente, la cache L1P del DSPC64x es de mapeo directo y es de 16 KB, lo que corresponde en capacidad a unas 4000 instrucciones. Algoritmos sencillos como filtros, FTT, etc., suelen consistir en bucles que se procesan con distintos datos. Cada uno de estos algoritmos se denominan núcleos (kernels). La L1P es lo suficientemente grande como para contener varios núcleos. Normalmente, estos núcleos se ejecutan secuencialmente. Por lo tanto, si estos algoritmos se colocan juntos en la memoria pueden reducirse el número de fallos de cache en accesos a la memoria de programa (código).

La memoria L1D al ser la asociación de dos conjuntos de memoria permite que el procesador pueda acceder a datos que caigan en la misma línea de cache ya que dispone de dos conjuntos, aunque interesa que los datos estén distribuidos para evitar que la cache tenga que descartar algunas de las líneas por conflictos con otros datos.

Cuando hay que desalojar una línea de la cache (eviction) el modelo que utiliza es de remplazar los datos usados menos recientemente, basándose en el principio de localidad.

El rendimiento de una memoria caché se mantiene por la localidad en el acceso a los datos. Así, algoritmos que hacen una gran cantidad de procesamiento en bloques trabajan muy bien en un entorno de caché. Ejemplos de estos algoritmos son filtros (FIR, IIR) y la transformada rápida de Fourier (FFT). Estos algoritmos son iterativos por naturaleza, por lo que se logra reutilización de código y datos aprovechando localidad temporal. Estos algoritmos suelen alcanzar hasta un 95% de eficiencia.

Los algoritmos no reutilizan los datos de entrada al producto escalar de vectores o una suma de vectores, requiriendo también, muy poco procesamiento por muestra, por lo que se dedica más tiempo solicitando nuevos datos. Por lo tanto, se puede mejorar el rendimiento de estos algoritmos empleando optimizaciones del sistema. Con la optimización del sistema también se pueden lograr eficiencias del 85% o incluso mayores.

Así, la clave de esta optimización es minimizar errores de caché:

- **Fallos obligatorios:** estos errores se producen durante el primer acceso a una línea de caché. Estos fallos se producen al no estar aun cargados o asignados los datos en la caché. A veces se conocen como "fallos de primera referencia".
- **Fallos de capacidad:** estos errores se producen cuando la caché no tiene suficiente espacio para almacenar todos los datos durante la ejecución de un programa.
- **Fallos de conflicto:** estos errores se producen porque varios bloques de código de programa o datos están compitiendo por la misma línea de caché.

Estas fuentes de errores de caché pueden reducirse mediante una serie de técnicas de optimización de código. Aunque los fallos obligatorios no pueden evitarse, se pueden minimizar aprovechando las ventajas de los fallos de la segmentación. Los fallos de conflicto pueden eliminarse cambiando la ubicación de código de programa o datos en memoria para que coincidan en la misma línea de caché. Los fallos de capacidad pueden reducirse al trabajar con pequeñas cantidades de código o datos en un momento, que puede lograrse mediante la reordenación de los accesos de los datos o por dividir los algoritmos en trozos más pequeños.

Es más importante que los datos de lectura estén consecutivos que las escrituras estén consecutivas ya que los datos de salida no tienen que ser almacenados en L1D. Esto se debe a que cuando hay fallos de escritura en L1D, los datos se escriben directamente en L2. Teniendo en cuenta este criterio, para mejorar las lecturas se puede utiliza una tabla de consulta intermedia que permita reordenar los datos de entrada para que se puedan leer consecutivamente y luego escribir los resultados al azar en un rango mayor de direcciones.

La cache L1 es de mapeo directo y esto puede afectar de forma decisiva en la ejecución del código ya que si el código está situado en memoria de forma inadecuada puede causar conflicto entre líneas y provocar fallos de cache durante la ejecución del programa. La mayoría de estos fallos puede evitarse alterando el orden en que las funciones están situadas en la memoria. En general, el código que se accede dentro de una ventana de tiempo debe asignarse de forma contigua en memoria.

Si dos funciones se superponen en L2, cuando la primera función se llama la primera vez, se asigna en L1P. Una llamada posterior a la segunda función hace que su código se asigne en L1P. Esto también provoca, si coinciden en líneas de caché, descartes de partes de la primera función. Cuando se vuelve a llamar a la primera función de nuevo en la siguiente iteración, estas líneas tienen que ser devueltas en L1P, por lo que estas ahora

excluirán líneas donde estaba la función segunda. El resultado final es un total de cuatro fallos de cache por iteración del bucle. Estos errores pueden evitarse completamente colocando el código de las dos funciones de forma contigua en memoria.

Así, en el código escrito se ha cuidado el diseño y localización de los bucles para reducir en lo posible este tipo de conflictos.

Otra técnica de optimización es el algoritmo de partición. Un algoritmo puede ser dividido en trozos más pequeños cuando el programa es demasiado grande para caber en la caché de L1P o los datos están demasiado grandes para caber en la caché de siga. Una serie de funciones puede encajar en L1P, mientras que al mismo tiempo proporcionar bufferes de datos adecuada para estas funciones en L1D.

Algoritmos como un producto escalar de vectores que no reutilizan datos de entrada. Estos algoritmos también requieren muy poco procesamiento por muestra, por lo que se dedica más tiempo solicitando nuevos datos de entrada. Las funciones individuales de la aplicación (la búsqueda de vectores y producto escalar) no presentan gran rendimiento en la cache visto de forma aislada. Sin embargo, incluso sin aplicar técnicas de optimización de nivel de sistema, la aplicación en su conjunto tiene una eficiencia de caché global de 79%. Además, después de aplicar las técnicas de optimización de la función y la partición de datos, la función de agrupación y optimizaciones en la segmentación de cauce, la eficiencia de la memoria caché puede elevarse hasta 93%.

6.2.2 Optimización de código

Las implementaciones de código de FastICA en C y C++ que pueden encontrarse en Internet están orientadas a computadores de propósito general y no a DSPs. A continuación se muestra como ejemplo una de las optimizaciones realizadas: el cálculo de la matriz de covarianza.

Para comprender mejor las optimizaciones primero se analizan tiempos de ejecución del código relacionado con la ecuación (6.1).

$$\boldsymbol{y} = \sum_{\boldsymbol{i=0}}^{\boldsymbol{n-1}} \boldsymbol{x_i^2} \qquad (6.1)$$

Mediante esta prueba se trataba de comprobar tanto los tiempos de ejecución en punto flotante como en punto fijo y diversas técnicas de optimización. La Tabla 6.1 muestra el número de ciclos de reloj consumidos en el DSP en función de diversas implementaciones para n=4096. A continuación se describen cada uno de los tiempos obtenidos y la información que puede extraerse de estos. Esta tabla muestra varias columnas: el número de ciclo que consume el algoritmo si los datos no están en cache (con fallos de cache), cuando los datos sí están en cache, y a partir de estos ciclos el speedup (mejora en velocidad) que se obtienen en las diversas pruebas considerando la primera de ellas como valor 1. Se considera este valor como referencia ya que el código que generalmente se encuentra en bibliotecas de funciones matemáticas está basado en variables de tipo punto flotante de doble precisión (Double), y si se utilizan estas funciones y se compilan directamente para el DSP serían los ciclos de reloj que se consumirían. Se recuerda que el DSP utilizado trabaja a 1 GHz, por lo que los tiempos de ejecución que se obtendrían sería multiplicando el número de ciclos por 10^{-9} s.

Código	N.Ciclos Datos no cache	N.Ciclos Datos cache	Speedup
Double	625696	626952	1
Float	415752	414136	1,5
Int	9760	8280	75,7
Opt I: Short . Short →Int	1840	**1032**	**607**
+ Opt II: Datos entrelazados	2856	2056	304
+ Opt III: Datos separados	1848	1032	**607**
+ Opt IV: Bucle tam.variable	4904	4112	152

Tabla 6.1 Número de ciclos del DSP para diversas implementaciones de código de la Ecuación 6.1.

Este DSP no incluye unidad de punto flotante, las operaciones con Double y con Float están bastante penalizadas. Se observa que utilizando operaciones con enteros mejora considerablemente la velocidad (*S=75,7*), aunque, también tiene un número elevado de ciclos si no se tienen en cuenta ciertas optimizaciones ya que la mejora final que puede obtenerse tras la optimizaciones es *S=607*, es decir 8 veces más rápido que el código original con enteros. Aunque los compiladores de la familia C6000 generan un código muy

optimizado están limitados por los requerimientos del sistema que son desconocidos por este: tipos de variables, localización, iteraciones de los bucles y ciertas técnicas de paralelización no pueden producirse de forma automática, pero si se tienen en cuenta en el diseño se consiguen grandes mejoras. La primera optimización (Opt. I) se basa en que el DSP incorpora multiplicadores de 16 x 16 bits y el resultado queda en 32 bits, por lo que las variables de entrada son *short* y las de salida son *int*. En la segunda optimización (Opt. II) paraleliza el código para que puedan ejecutarse dos operaciones por iteración correspondientes a dos canales que están entrelazados en el mismo buffer, donde se observa una penalización del 50%. En la optimización III se reorganizan los datos de entrada y se cambia la localización de las variables de forma que están en variables completamente independientes lo que detecta el compilador y vuelve a obtener los mismos ciclos, por lo que resulta importante separar los datos de componentes que no estén en el mismo buffer. Finalmente, en la optimización IV se observa la penalización cuando los bucles son de tamaño variable, observándose que para el compilador es muy importante conocer el número de iteraciones de un bucle para poder hacer *loop unrolling*.

Se observa que generalmente la cache tiene un efecto positivo en los tiempos de ejecución pero en la ejecución con variables de tipo doble precisión (Double) sorprendentemente empeora la ejecución. Este efecto también se observará en las siguientes pruebas y es debido a que las rutinas para dicha aritmética tienen problemas de conflicto de líneas en la cache L1P provocando fallos que requieren más ciclos de reloj.

El compilador de C de esta familia genera un código muy optimizado cuando el código es preciso. Como puede observarse, aunque el bucle tiene que realizar 4096 iteraciones consigue completar el código es sólo 1032 ciclos. Quitando los ciclos iniciales del bucle quedarían 1024 ciclos, lo que quiere decir que está haciendo por ciclo de reloj 4 iteraciones del bucle y cada iteración es una suma y un producto de $x_i.x_i$. En cada ciclo de reloj ejecuta 4 sumas de 32 bits, 4 productos de 16 bits por 16 bits, avance de punteros en memoria, incremento del contador del bucle y comprobación del número de iteraciones. Como se ha observado, el compilador necesita saber el número de iteraciones para obtener un código eficiente y utilizar todas las unidades En el apéndice B.1 se muestra la información que produce el compilador cuando conoce el número de iteraciones y en B.2 cuando desconoce el número de iteraciones. A partir de las pruebas realizadas anteriormente se estudia una función más compleja y las optimizaciones que pueden realizarse en el cálculo de la matriz de covarianza.

La función de cálculo de la matriz de covarianza debe ejecutar tres bucles anidados para completar las combinaciones de las componentes de forma genérica. En esta rutina se han realizado optimizaciones adicionales ya que, teniendo en cuenta que la implementación está orientada inicialmente a separación de 2 señales, se pueden recorrer sólo en una dimensión y calcular simultáneamente las tres componentes distintas (Optimización II, Tabla 6.2). La Tabla 6.2 muestra el número de ciclos totales para el cálculo de la matriz de covarianza de 4096x2 aplicando diversas optimizaciones que se describen a continuación.

Código	N.Ciclos Datos no cache	N.Ciclos Datos cache	Speedup
Double	2470040	2476896	1
Float	1873080	1874032	1,3
Int	79784	79592	31
Opt I: Short . Short →Int	40464	40288	61
+ Opt II: Bucles optimizados	13144	12336	201
+ Opt III: Bucle tam.fijo	6144	5288	**468**
+ Opt IV: Bucle tam.fijo –O3	6152	5376	461

Tabla 6.2 Número de ciclos del DSP para diversas implementaciones del cálculo de la matriz de covarianza.

La primera medida muestra el número de ciclos obtenidos en el código genérico que puede obtenerse en las función *cov* de la biblioteca itpp [ITP12] compilada directamente para el DSP. Esta medida se considera como valor 1 en la columna de mejora en velocidad (speedup). Se vuelve a comprobar que incluso cambiando las variables a tipo entero (int) sólo mejora la velocidad *s=31*, que es bastante, pero optimizando manualmente el código se consigue finalmente una mejora *s=461*, lo que representa incluso una mejora de 15 veces en comparación con la implementación original con enteros. En el apéndice B.3 se muestra la información que produce el compilador cuando conoce el número de iteraciones y en B.4 cuando desconoce el número de iteraciones.

Cuando el código está bien optimizado se reduce el número de ciclos y se eleva el ratio entre estar los datos en la caché y que no lo estén (6144/5288 ciclos). En este caso se

reduce un 14% el número de ciclos consumidos cuando los datos sí están en cache, por lo que es recomendable garantizar que los datos sí estén en ella.

El código se ha generado con el nivel de optimización *O2*. Se comprueba que incluso un nivel más elevado, *O3*, consume más ciclos de reloj. Generalmente los compiladores generan un código más rápido en *O3* pero el código en C está optimizado y no consigue optimizar más. Así, de estas implementaciones se extraen ciertas conclusiones:

- Conviene utilizar en lo posible operaciones de 16 bits (short) x 16 bits (short) acumulación en 32 bits (int) para aprovechar la arquitectura del DSP, descartando operaciones con punto flotante o incluso sólo con enteros de 32 bits (int).
- Utilizar bucles de tamaño fijo mejora considerablemente los tiempos de ejecución ya que el compilador puede aprovechar mejor las unidades de ejecución haciendo *loop unrolling*. Experimentalmente se ha observado que los bucles con tamaño 2^n se optimizan mejor.
- La optimización manual del código en C resulta muy aconsejable ya que se observa en la función de la matriz de covarianza que el código es 14 veces más rápido que el código con enteros o incluso 468 veces más rápido que el código original disponible en la biblioteca it++
- Cuando el código está bien optimizado se reduce el número de ciclos y la relación entre estar los datos en cache y no estar es mayor, por lo que es recomendable organizar los datos de forma que sí estén presentes.
- Conviene utilizar zonas distintas de memoria para componentes distintas el lugar de que los datos estén entremezclados para mejorar el paralelismo que puede observar el compilador.

Hay que tener en cuenta que este DSP no tiene unidad de punto flotante lo que obliga al procesador a realizar esas operaciones mediante software. La Tabla 6.3 muestra los ciclos de reloj consumidos para las funciones raíz cuadrada *(sqrt(x))* y *x* elevado a *y (pow(x,y))* utilizadas en algunas ecuaciones.

Tipo	Sqrt()	Pow()	Sqrt() optimizado
Double	3480	7696	
Float	3464	7120	
Int	2048	7816	**360**

Tabla 6.3 Número de ciclos consumidos por el DSP para funciones sqrt()(y pow().

Se observa que la función raíz cuadrada consume bastantes ciclos incluso para operaciones enteras (2048 ciclos) ya que esa operación no se realiza por hardware, por lo que se ha implementado una función que requiere sólo 360 ciclos, mejorando en 5,6 veces el tiempo de ejecución original. Dicha función es necesaria en los cálculos de la norma de un vector.

6.2.3 Optimizaciones globales

Las optimizaciones evaluadas anteriormente se han tenido en cuenta para el desarrollo del resto de funciones de forma que se obtenga un código eficiente. Además, existen otras optimizaciones que afectan a diversas funciones entre sí y se consideran optimizaciones globales. A continuación se describen las más importantes.

Aunque la reutilización de datos mejora la tasa de aciertos en la cache, resulta más importante evitar en lo posible el cálculo repetido de ciertas funciones. Para combinar ambos elementos se utiliza un doble buffer circular basado en ventanas y subventanas. La Figura 6.2 muestra cómo funciona uno de estos buffers.

Los algoritmos propuestos se basan en ventanas deslizantes que analizan un conjunto de muestras. Si estos algoritmos se calculan en MATLAB cada desplazamiento de la ventana implica partir de unos nuevos vectores y realizar de nuevo los cálculos, ya que las funciones de FastICA parten de un bloque de datos.

Con la implementación propuesta hay una serie de datos que sólo es necesario recalcular en cada conjunto nuevo de elementos. Así, la ventana representa el conjunto de datos que se analizan y la subventana la porción de datos nuevos que se incorporan. Cuando se desplaza la ventana se pueden aprovechar cálculos anteriores y calcular sólo ciertas funciones sobre la nueva subventana, lo que reduce los tiempos de ejecución.

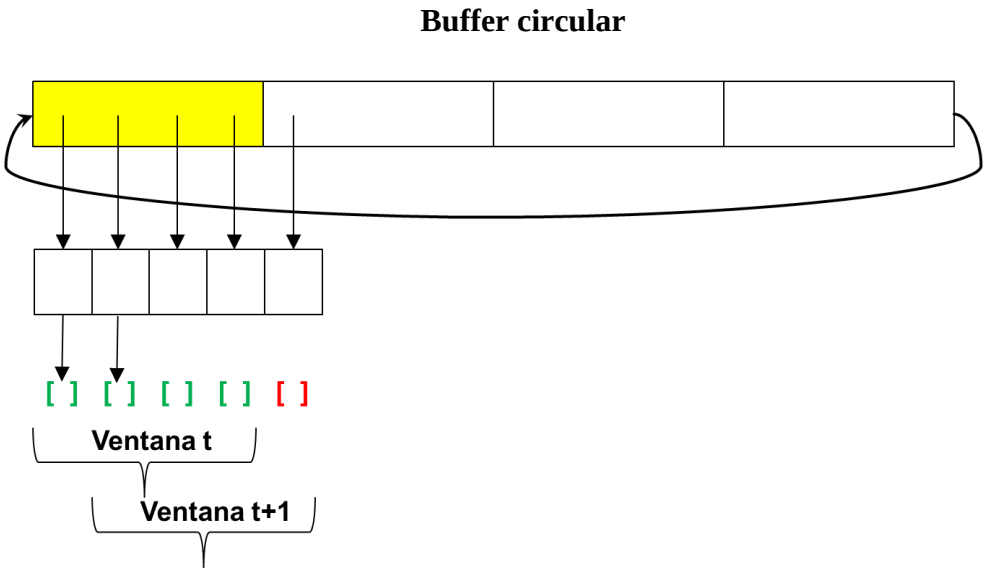


Figura 6.2 Procesamiento con buffer circular y ventanas.

De esta forma también se optimizan los datos en la cache ya que el buffer siempre está en el mismo sitio mejorando la tasa de aciertos, evitando desplazamientos de datos y almacenando cálculos también en posiciones fijas. En el modelo propuesto, las muestras van llegando mediante la rutina de interrupciones, llenándose el buffer y, cuando hay un número de muestras suficiente se provoca un evento que se activa en el bucle principal que activa el procesamiento de la ventana actual mientras siguen llegando muestras que se incorporan en el espacio libre del buffer circular.

Por otra parte se han mejorado los algoritmos de multiplicación de matrices ya que para optimizar la ejecución se parte de un número fijo de componentes, que inicialmente se fija en 2. Esto permite que las multiplicaciones de matrices no sea necesario ejecutar dos bucles anidados en ambas dimensiones ya que al conocer el número fijo de componentes, se reduce la multiplicación a un único bucle. Además, se trata de utilizar ventanas de tamaño fijo para que compilador estime el tamaño de loop-unrolling.

Con los elementos que son variables se ha generado un conjunto de parámetros que pueden evaluarse. Aunque se ha tomado una primera aproximación, en el apartado 6.3 se propone un sistema automático que pueda mejorar la optimización.

En el algoritmo de la ventana deslizante no se puede considerar la ventana previa como inicialización para la nueva ventana, debido a que la matriz de inicialización deberá tener varias dependiendo del dato que cambia desde una ventana a la siguiente, no existiendo

garantía de la matriz de desmezclado estimada para una ventana, sino, como colector para la siguiente. Pudiéndose proyectar la salida desde la ventana previa como colector para la siguiente ventana, usando ésta como la nueva matriz de inicialización.

Se puede observar como una matriz arbitraria, *W*, sobre las múltiples decorrelacionadas. Siendo *X* una matriz tal que la i-ésima fila contiene las observaciones de la componente *i*, y $X=U\sum V^T$ la descomposición en valores singulares de *X*. O sea, *U* y *V* son matrices unitarias y es $\sum$ una matriz diagonal. La matriz *M* está en el colector decorrelacionado para *X* si y sólo si tiene la forma:

$$\boldsymbol{M = DQ\sum^{-1}U^T} \tag{6.2}$$

Donde *D* es diagonal y *Q* es ortogonal. Para simplificar los cálculos se considera que *D* no cambia mucho al pasar de una ventana a la siguiente. Tomando el valor de *D* para que sea igual al obtenido para la ventana previa. Manteniendo así una matriz ortogonal *Q* que minimiza:

$$\boldsymbol{\left\|W - DQ\sum^{-1}U^T\right\|^2} \tag{6.3}$$

Esto se calcula mediante la descomposición de en valores singulares de $D^{-1}WU\sum^{-1}$. Hacer *Y* y *Z* matrices unidad y *C* diagonal tal que $D^{-1}WU\sum^{-1}=YCZ^T$, siendo el valor de Q que minimiza la expresión anterior:

$$\boldsymbol{Q = YZ^T} \tag{6.4}$$

Incluso utilizando la proyección de la última matriz de desmezclado sobre flujo, el algoritmo converge a una solución subóptima. Para corregirlo se utiliza una estimación suavizada de la matriz de desmezclado como base para la inicialización. Hay que hacer que W_t sea la estimación de la matriz de desmezclado para la ventana t, para posteriormente usar $1/2\ (W_{T-2} + W_{T-1})$ como la matriz de inicialización para la nueva ventana en el tiempo $t = T$.

En todos los algoritmos de ICA las fuentes recuperadas son las salidas en un orden aleatorio, siendo posible que en la presente ventana se recuperen en un orden diferente de la de la ventana previa. Si se elige el tamaño del paso mucho menor que la longitud de la ventana, a continuación, dos ventanas consecutivas tendrán una zona de superposición en la que ambas producen una estimación para la misma fuente. Si a_1 es una de las fuentes recuperadas desde la primera ventana y a_2 es una de las fuentes recuperada desde la segunda ventana. Suponiendo que en ambas ventanas ICA recupera las fuentes con una precisión razonable, entonces, si a_1 y a_2 se corresponden con la misma fuente subyacente su correlación debe estar muy cerca de cualquiera más o menos uno. Si se corresponden a diferentes fuentes entonces su correlación estará muy cercana a cero. Por lo tanto, las fuentes pueden ser despermutadas por la elección de la permutación que maximiza la suma de los valores absolutos de las correlaciones entre las fuentes recuperadas en la región de solapamiento. Una vez que las fuentes han sido despermutadas, el signo de la fuente estima que puede ser invertida si fuera necesario para asegurar que las correlaciones son de hecho positivas.

En la implementación del algoritmo FastICA la mayor parte del tiempo se consume en la iteración de la aproximación de w hasta que converge. Si se considera:

$$\boldsymbol{w} = \frac{\left(\boldsymbol{X} * \left((\boldsymbol{X}' * \boldsymbol{w}) .\hat{}\, \mathbf{3}\right)\right)}{\boldsymbol{num\_muestras}} - \mathbf{3} * \boldsymbol{w} \qquad (6.5)$$

Esta ecuación se implementa tanto en Matlab como en C++ mediante tres bucles, mientras que de forma optimizada se puede implementar en un solo bucle en el DSP.

Además, el cálculo de elevar al cubo es la parte que más penaliza, ya que en el caso de una ventana de 4096 hay que calcular precisamente 4096 números al cubo. Como se ha visto anteriormente la función pow() requiere bastantes ciclos de reloj e incluso para enteros, por lo que se ha optado por multiplicar por sí mismo, consumiendo finalmente 26352 ciclos para iteración.

En la Tabla 6.4 se muestran los tiempos de ejecución del DSP y un Intel Core i5 (2.66 GHz) con Matlab de diversos algoritmos implementados con ventanas de 4096 muestras y 2 componentes.

Algoritmo	Tiempo Ejec. DSP	Tiempo Ejec. PC (Matlab)
Iteración FastICA	26352 ciclos 26,3 µs	457 µs
FastICA	384 µs	3,7 ms
ASWICA	54 ms	74 ms
ADSWICA	65 ms	83 ms

Tabla 6.4 Tiempos de ejecución obtenidos con el DSP y un Intel Core i5 con Matlab.

Cabe destacar que para el FastICA el DSP (a 1 GHz) es más rápido para que el Intel Core i5 (2,66 GHz) con Matlab debido a todas las optimizaciones que se han considerado. Por otra parte, para el FastICA los tiempos de ejecución pueden variar para muestras distintas e incluso para las mismas muestras ya que el algoritmo converge tras cada iteración partiendo de un estado inicial aleatorio. Dicha convergencia es bastante rápida, pero si los datos requieren más iteraciones el tiempo aumenta, de ahí que se muestre en la tabla también el tiempo por iteración. Aunque la implementación de FastICA es bastante eficiente, con la implementación de los algoritmos ASWICA y ADSWICA se pierde eficiencia ya que el cálculo del spline cúbico debe realizarse en punto flotante. Incluso en esos casos, el DSP muestra mejores tiempos de ejecución, que aunque en esos algoritmos es algo menor, en el caso del FastICA es 10 veces más rápido.

Ya que FastICA es un algortimo que parte de un estado inicial aleatorio, y la solución se obtiene como una serie de aproximaciones, puede variar el número de iteraciones que necesita el algoritmos para determinar la convergencia y eso afecta por lo tanto en el tiempo de ejecución final. Se ha estudiando cómo afecta la precisión determinada por los redondeos de la aritmética de punto fijo en el DSP y en Matlab al número de iteraciones para que converja FastICA.

Partiendo de una señal muestreada a 20 KHz se han realizado diversos experimentos cambiando el número de muestras y la precisión en el DSP. En el caso de Matlab sólo se ha cambiado el número de muestras 1024 y 4096. Se han realizado 10 iteraciones en cada caso y en la Tabla 6.5 me muestran los valores obtenidos del número medio de iteraciones necesarias para que converja. En el caso del DSP, si la precisión es al menos de 1/8192 el algoritmo ya puede converger y encontrar soluciones. También cabe destacar que con el

redondeo por las operaciones de punto fijo incluso reduce el número medio de iteraciones para el caso de 1024 muestras.

Precisión DSP	Matlab		DSP	
	1024 Muestras	4096 Muestras	1024 Muestras	4096 Muestras
1/8192	9,8	4,7	11,1	5,4
1/ 131072			8,6	4,8

Tabla 6.5 Número medio de iteraciones para obtener convergencia en Matlab y el DSP.

Puede observarse en dicho experimento que afecta más el número de muestras utilizados que los errores de redondeo. El número medio de iteraciones toma un valor óptimo para tamaño de ventana de 4096, que coincide con el valor estimado en otros experimentos realizados anteriormente en el apartado 5.4.1.

6.3 Optimización mediante algoritmos genéticos

La optimización automática del código permite reducir tiempos de ejecución y con ello aumentar la eficiencia del mismo. Se realiza mediante la aplicación de algoritmos genéticos. Estos algoritmos genéticos heredan su filosofía de trabajo de la evolución biológica, basados en el azar y en funciones de probabilidad.

Estos algoritmos crean inicialmente un conjunto inicial (población), los cuales contienen información asociada (genoma). Este conjunto inicial va evolucionando según el comportamiento que va adoptando cada elemento de ese conjunto inicial, evaluándose en cada iteración el progreso de la solución, así como el comportamiento de cada uno de estos individuos. Se modifican y combinan los elementos para evolucionar hasta una mejor solución. Para que el proceso no se prolongue hasta el infinito, se le aporta un criterio de parada.

Se puede emplear este tipo de algoritmos para obtener las óptimas opciones de compilación. Hay que tener en cuenta que la optimización se valora principalmente respecto al tiempo de ejecución necesario del programa compilado. El número de componentes del conjunto permanecerá igual en todo momento, por lo que cuando se

generan más elementos, a través de las modificaciones y combinaciones de los mismos, hay que ir descartando los menos optimizados y útiles para el conjunto.

En el proceso de modificación y combinación, se van dejando sin modificar el 25 % mejor. Por lo que, teniendo en cuenta esto y el hecho de mantener el tamaño del conjunto, los criterios de parada serán:

- Agotar el número de iteraciones fijado. Se devuelve el mejor conjunto de opciones de compilación.
- Agotar el número de iteraciones sin mejora, para ello se van comparando los valores de cada elemento del conjunto con los de la iteración anterior. Devolviendo al final el mejor conjunto de opciones de compilación.

Las opciones de compilación se codifican como una cadena de caracteres cuya información, así como las operaciones aplicables y las constantes se agrupan formando lo que se denomina clase. Existen tres tipos de opciones:

- Opciones de listas fijas de valores posibles.
- Opciones parametrizadas.
- Marcas.

Cada una de estas opciones ha de almacenarse en una lista distinta, añadiéndole un atributo que guarda el tiempo de ejecución de la última iteración, para poder comprobar el grado de optimización existente hasta ese momento. Las probabilidades de modificación y combinación se obtienen aplicando funciones de probabilidad para comprobar si se realizan dichas operaciones o no.

Si se necesita mayor rapidez de convergencia, se puede tomar una parte del conjunto. Los elementos del conjunto se han de ir almacenando en un vector de elementos, mientras que las opciones de compilación se almacenan en una estructura de datos específicamente diseñada, y se leen de un fichero de entrada. Por último, el número de probabilidades de cada opción es guardado en un vector de enteros.

A continuación se muestra un pseudo-código a modo de ejemplo:

```
crear población de forma aleatoria
evaluar fitness del inicial
for generacion = 1 to límite_generación do
  evaluar fitness de todos
  ordenar todos por fitness
```

```
    if mejor_resultado_generation hay mejora then
      contador_no._mejora = 0
    else
      contador_no._mejora = contador_no._mejora  + 1
      if contador_no._mejora == limite_contador_no._mejora =then
        terminar
      end if
    end if
    if tamaño_población_variable then
      reducir tamaño_población
      if tamaño_población == 1 then
        terminar
      end if
    end if
  reemplazar la mitad inferior de la población con la mitad superior
{selección natural}
  aleatoriamente mezclar individuos del 1er cuarto con el 4° y del 2°
con el 3°{crossover}
  aleatoriamente mutar el 50% inferior de la población
  end for
```

Ya que se ha realizado una optimización bastante exhaustiva tratando de fijar la mayor cantidad de elementos variables para que el compilador pueda optimizar de la mejor manera posible el código, realmente quedan menos elementos variables de optimización. Estos son:

- Parámetros de loop-unrolling. Directiva *#pragma UNROLL*
- Directivas *#pragma MUST_ITERATE*
- Optimizaciones del compilador: -o2, -o3
- Opción del compilador: -mt . Este parámetro le indica al compilador que para cualquier función del compilador los parámetros que se pasan como punteros nunca apuntarán a la misma dirección, lo que puede mejorar las prestaciones del código.
- Aplicando selectivamente la palabra clave *restrict* en los parámetros que corresponden con punteros.

Así, dentro del código se introducen unas marcas que pueden reconocerse fácilmente de forma que se sustituyen automáticamente por los parámetros que estime el algoritmo genético, volviendo a compilarse y a calcularse los tiempos de ejecución.

La Figura 6.3 muestra para el algoritmo FastICA la ganancia en velocidad obtenida tanto en la optimización original con optimizaciones *-O2 (S=1)* y *–O3 (S=1,05)* así como aplicando al algoritmo genético para la optimización de *(S=1,07).*

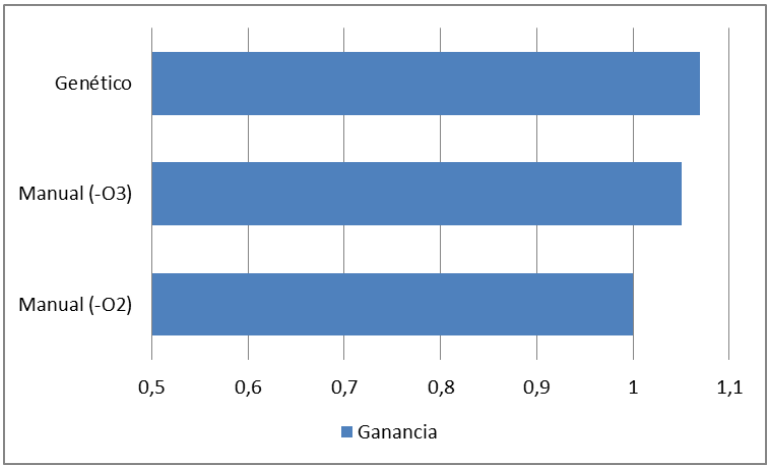


Figura 6.3 Ganancias de velocidad obtenida en el DSP en diversas configuraciones.

Al principio de este trabajo se planteó como un objetivo que la optimización automática podría ser necesaria debido a la cantidad de elementos que deben considerarse para generar un código optimizado. Resulta mucho más eficiente el análisis del código y la optimización manual que la optimización automática, que aunque consigue una ligera mejora, parte del código ya optimizado previamente de forma manual. Cabe destacar que el elevado número de optimizaciones manuales del código reducen considerablemente las combinaciones posibles que se deben evaluar.

6.4 Conclusiones

Código que puede funcionar de forma eficiente en procesadores de propósito general suele no funcionar de forma óptima en los DSP. Esto es debido a que los procesadores de propósito general incorporan elementos que obtienen información en tiempo de ejecución y mejora el rendimiento. Elementos como la ejecución fuera de orden o la predicción de

saltos o grandes tamaños de cache pueden mejorar considerablemente los tiempos de ejecución pero requieren más recursos hardware.

Para reducir tamaño, consumo y coste, los DSP carecen de estos elementos, penalizando la ejecución general de aplicaciones. Por eso los DSP no se utilizan como procesadores para ejecutar un sistema operativo orientado a usuario o ejecutar aplicaciones genéricas, ya que están orientados a aplicaciones de procesamiento de señales. Los DSP son procesadores que pueden obtener un gran rendimiento cuando las aplicaciones están optimizadas para su arquitectura interna. Los compiladores para estas plataformas son muy sofisticados y puede optimizar bien cuando disponen de información suficiente.

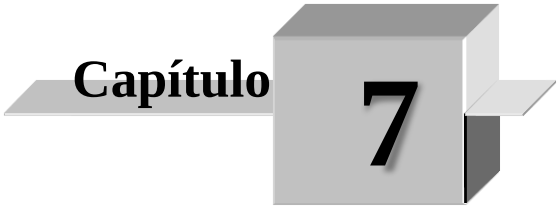
Aplicaciones como las que se abordan en este trabajo requieren una optimización previa para poder aprovechar las prestaciones que ofrecen los DSP, ya que así el compilador puede obtener el máximo provecho de la arquitectura VLIW del procesador.

Así pues, una parte importante de esta tesis es la optimización de dicha implementación para poder obtener un código eficiente, planteando una primera optimización manual y posteriormente una optimización automática de ciertos parámetros.

Para la primera fase de optimización manual se ha estudiado en profundidad la arquitectura del DSP utilizado y se han realizado un conjunto de experimentos para determinar qué elementos pueden optimizarse y cómo hacerlo. Por ejemplo, el modelo de cache que implementa el DSP limita el funcionamiento del procesador, la forma de invocar a las funciones o cómo realizar las iteraciones son algunos de ellos además de considerar la propia optimización del código y variables.

Dicha optimización manual ha permitido obtener un código muy eficiente y, por lo tanto, las posibles mejoras adicionales se ven reducidas por las limitaciones del compilador.

En la segunda fase, se ha desarrollado una optimización automática partiendo de la optimización previa. Los cambios establecidos de forma automáticas han permitido mejorar sólo un poco los tiempos obtenidos y esto ha sido debido a la elevada optimización alcanzada. Combinando tanto las técnicas de optimización manual como automática se ha podido obtener un código muy eficiente, con buenos resultados en los tiempos de ejecución de los diversos algoritmos, consiguiendo el DSP a 1 GHz tiempos menores en todos los casos que en un Intel Core i5 a 2,5 GHz con Matlab, y con un consumo menor.



7 CONCLUSIONES

En el presente trabajo se ha presentado el desarrollo de un sistema que resuelve autónomamente el problema de la Separación Ciega de Fuentes, basado en la fusión de algoritmos ICA (FastICA), la propuesta de dos nuevos algoritmos (ASWICA y ADSWICA), para la separación de señales con mezclas no estacionarias y el empleo de plataformas basadas en Procesadores Digitales de Señales (DSP). Hay que destacar la importancia de los logros obtenidos en este trabajo de cara a nuevas líneas de investigación, así como desde el punto de vista docente, ya que permite motivar y apoyar el estudio e investigación de nuevas aplicaciones en separación de señales utilizando procesadores digitales de señales.

7.1 Principales aportaciones y conclusiones

Las principales aportaciones y conclusiones obtenidas pueden resumirse en los siguientes puntos:

- Se ha estudiado la forma de aprovechar la potencia de cálculo del DSP para mejorar la separación de fuentes en entornos de mezcla no estacionaria.
- Se han analizado los algoritmos que se encuentran en la bibliografía y se han propuesto dos nuevos algoritmos adaptativos para la separación de señales con mezclas no estacionaria, denominados ASWICA y ADSWICA, que se basan en FastICA y utilizan ventanas deslizantes.
- Los algoritmos propuestos modelan eficazmente los cambios en la matriz de mezcla no estacionaria ya que, debido al modelo desarrollado, mantienen un seguimiento continuo de las señales.

- Una vez elaborados los algoritmos a implementar, se ha realizado un estudio de las optimizaciones de código que pueden hacerse para el DSP. De este estudio se desprende que para un uso eficiente del DSP es necesaria una optimización manual del código, además de las optimizaciones propias del compilador. Esta optimización aporta al compilador información importante para poder obtener el máximo rendimiento y eficiencia del procesador con arquitectura VLIW.
- Se han obtenido una serie de consideraciones para aplicar a la optimización del código para el DSP, que están orientadas a la implementación de cualquier algoritmo ICA por su naturaleza (número constante de componentes a procesar, localización de la memoria de almacenamiento de almacenamiento, reaprovechamiento de cálculos intermedios, etc…).
- Se han implementado en el DSP tanto el algoritmo básico FastICA como los algoritmos propuestos ASWICA y ADSWICA, partiendo de un estudio riguroso de las optimizaciones, y analizando experimentalmente todo el código.
- Se han obtenido resultados óptimos en los tiempos de ejecución de los diversos algoritmos, consiguiendo el DSP a 1 GHz tiempos de ejecución menores en todos los casos que en un Intel Core i5 a 2,5 GHz con Matlab.
- Estas mejoras no hubieran sido posibles sin la optimización manual del código, ya que el DSP, por su arquitectura, necesita información adicional complementaria para poder generar un código eficiente.
- Los procesadores de propósito general obtienen buenos resultados en los tiempos de ejecución debido a que incorporan elementos como la ejecución fuera de orden o grandes tamaños de cache, lo que les permite optimizar sus unidades de ejecución en tiempo real y disponer de gran cantidad de datos de forma rápida. Con las optimizaciones realizadas en este trabajo, el DSP resulta mucho más adecuado, mejorando la calidad de las señales, pudiendo utilizarse en numerosas aplicaciones con mayor portabilidad y menor consumo.
- Los DSP, al estar orientados a plataformas móviles o de tamaño más reducido, deben limitar la complejidad de la arquitectura y por ello reducir el tamaño de sus caches, así como limitar elementos complejos. Los sistemas de ejecución fuera de orden requieren más recursos hardware, por lo que no se utilizan en

DSP. Además, provocan indeterminación en los tiempos de respuesta del procesador.

- Se ha desarrollado un sistema automático de optimización de código para el DSP y los algoritmos implementados anteriormente.
- Partiendo de la optimización manual de los algoritmos, se han realizado cambios en el código y en los argumentos del compilador, para que el algoritmo genético mejore los tiempos de ejecución.

7.2 Trabajos futuros

Por otra parte, el trabajo descrito en esta memoria abre nuevas perspectivas de investigación y desarrollo, tanto en el ámbito de la docencia como en el de la investigación.

Algunos temas que se pueden acometer de forma inmediata, dada la posibilidad de procesar señales físicas reales, y que quedaron fuera del alcance inicialmente propuesto para este trabajo, son:

- Aplicaciones en medicina, y en especial a la detección, seguimiento y clasificación de determinadas enfermedades (cáncer, alzhéimer, etc.).
- Aplicaciones en el sector aeroespacial.
- Aplicaciones para la mejora de los sistemas de seguridad de los vehículos (automóviles, motocicletas, etc.).
- Aplicaciones en la astronomía.
- Aplicaciones para la Ayuda para el desarrollo del Tercer Mundo (localización de acuíferos, etc.).

Esta tesis aborda diferentes líneas de aplicación, y cada una de ellas puede ampliarse de forma más profunda. En el caso de la separación de mezclas no estacionarias pueden estudiarse diversos algoritmos que posibiliten el cambio de señales de forma más rápida con respecto al número de muestras. En el caso de la optimización del código para DSP sería interesante disponer de herramientas que realicen algún tipo de preprocesado del código fuente, lo que permitiría una optimización más profunda del código así como la

posibilidad de combinar ésta con los algoritmos que obtendrían la optimización evaluando diversos parámetros, tanto en el código como al realizar la compilación.

Finalmente, hay que indicar que apenas existen en la bibliografía implementaciones en procesadores digitales de señal de procedimientos de análisis en componentes independientes, lo que permite asegurar que el trabajo desarrollado en esta tesis doctoral marcará un antes y un después en este apasionante y reciente campo de la separación de señales.

7.3 Trabajos publicados relacionados

- **J. Carlos M. Comba**, A. F. Díaz, M. Rodríguez-Alvarez, J.David M. Comba, C. G. Puntonet "Separación ciega de señales adaptativa para procesamiento en tiempo real con DSP", Actas III Jornadas de Computación Empotrada (JCE2012), Septiembre 2012
- A. F. Díaz; C. G. Puntonet; B. Del Pino; A. Cañas, "Plataforma flexible de procesamiento de señales para el aprendizaje de sistemas basados en DSP" JENUI 2005: XI Jornadas de Enseñanza Universitaria de la Informática". pp.397-403.Madrid 2005.
- A.F. Díaz, L. Parrilla, M. Rodríguez Alvarez, C. G. Puntonet. "Decodificación de un sonido encriptado mediante la utilización de un procesador digital de señal, Seminario Anual de Automática, Electrónica Industrial e Instrumentación, (SAAEI'98).Universidad Publica de Navarra, pp. 127-128, 16 a 18 de Septiembre de 1998.

Bibliografía B

BIBLIOGRAFÍA

[ABR05] F. Abrard and Y. Deville, A time-frequency blind signal separation method applicable to underdetermined mixtures of dependent sources, Signal Processing, vol. 85, issue 7, pp. 1389-1403, July 2005.

[ADD09] William Addison, Stephen Roberts. Blind source separation with non-stationary mixing using wavelets. 2009.

[AGA06] F. Agakov, E. Bonilla, J. Cavazos, B. Franke, G. Fursin, M.F.P. O'Boyle, J. Thomson, M. Toussaint, and C.K.I. Williams. Using machine learning to focus iterative optimization. In Proceedings of the 4thAnnual International Symposium on Code Generation and Optimization (CGO), March 2006.

[ALM99] L.B.Almeida, G.C.Marques, "Nonlinear Blind Source Separation by Pattern Repulsion". Lecture Notes in Computer Science, vol. 1607, pp.674-682, Springer-Verlag, 1999.

[ALT02] David M. Alter, "Getting Started in C and Assembly Code With the TMS320LF240x DSP" Application Report, No. SPRA755A, Jul 2002. Texas Instruments.

[AMA95] S. Amari, A. Cichocki, H. H. Yang. "Recurrent Neural Networks for Blind Separation". Proceedings of 1995 International Symposium on Non-linear Theory and Applications (NOLTA'95), Las Vegas (USA), Vol. 1, pp. 37-42, December 1995.

[AMA96] S. Amari, A. Cichocki, and H.H. Yang, "A new learning algorithm for blind signal separation". Advances in Neural Information Processing Systems, Vol.8, MIT Press: Cambridge, MA, pp. 757-763, 1996.

[AMA97a] S. Amari, J. F. Cardoso, "Blind source separation – semiparametric statistical approach". IEEE Trans. on Signal Processing, vol. 45, pp. 2692-2700, 1997.

[AMA97b] Shun-ichi Amari, "Neural learning in structured parameter spaces - natural Riemannian gradient," Proc. of 1996 Advances in Neural Information Proc. Systems, NIPS '96, pp. 127-133, MIT Press, 1997.

[AMA98a] S. Amari, "Natural gradient works efficiently in learning". Neural Computation, vol. 10, pp. 251-175. 1998.

[AMA98b] S. Amari, A. Cichocki, "Adaptive blind signal processing – Neural networks approach". Proceedings of the IEEE vol. 86, nº 10, pp. 2026-2048, 1998.

[AMB05] Vanja Ambrozic, Manuele Bertoluzzo, Giuseppe S. Buja, and Roberto Menis, "An Assessment of the Inverter Switching Characteristics in DTC Induction Motor Drives", IEEE Transactions on Power Electronics, vol. 20, No. 2, pp. 457-465, March 2005.

[ANG01] W.-P. Ang and B. Farhang-Boroujeny, "A new class of gradient adaptive step-size lms algorithms," IEEE Transaction on Signal Processing, vol. 49, no. 4, pp. 805–810, April 2001.

[BAS07] Guy Bashkansky, Yaakov Yaari, _Black Box Approach for Selecting Optimization Options Using Budget-Limited Genetic Algorithms_, IBM Haifa Research Lab, 2007.

[BAX06] Paul Baxter, Geoff Spence, and John McWhirter. Blind signal separation on real data: Tracking and implementation. In Justinian Rosca, Deniz Erdogmus, Jose C. Principe, and Simon Haykin, editors, Independent Component Analysis and Blind Signal Separation: 6th International Conference, volume 3889 of Lecture Notes in Computer Science, pages 327–334. Springer-Verlag, 2006.

[BEL95a] A. Bell, T. Sejnowski, "An information-maximisation approach to blind separation and blind deconvolution". Neural Computation, vol. 7, pp. 1129-1159, 1995.

[BEL95b] Bell, A. J., & Sejnowski, T. J. (1995). A Non-linear Information Maximisation Algorithm that Performs Blind Separation. Advances in Neural Information Processing Systems , 7, 467 - 474.

[BEL95c] Anthony J. Bell and Terrence J. Sejnowski, "An informationmaximization approach to blind separation and blind deconvolution," Neural Computation, Vol. 7, No. 6, pp. 1129-1159, Nov. 1995.

[CAL01] Daniel E. Callan, Akiko M. Callana, Christian Kroosb, and Eric Vatikiotis-Batesonb. Multimodal contribution to speech perception revealed by independent component analysis: a single-sweep eeg case study. Cognitive Brain Research, 10(3):349–353, January 2001.

[CAS99] Llamuza Castell, R. Implementación del algoritmo de codificación de voz GSM 06.10 en un sistema de tiempo real basado en DSPs. Proyecto fin de carrera Ingeniería Electrónica, Dpto. Arquitectura y Tecnología de Computadores. Universidad de Granada.1999.

[CRU99] S. Cruces, "Una visión unificada de los algoritmos de separación ciega de fuentes", Tesis Doctoral, Universidad de Vigo (1999).

[CAI06a] Caiafa, C. F., & Proto, A. (2006). Separation of Statistically Dependent Sources Using an L2-distance Non-gaussianity Measure. Signal Processing , 86 (11), 3404 - 3420.

[CAI06b] Caiafa, C. F., Kuruoglu, E., & Proto, A. (2006). A Minimax Entropy Method for Blind Separation of Dependent Components in Astrophysical Images. Maxent 2006 Conference Proceedings. 872, pp. 81 - 88. Paris: AIP.

[CAI06c] Caiafa, C., Salerno, E., Proto, A., & Fiumi, L. (2006). Dependent Component Analysis as a Tool for Blind Spectral Unmixing of Remote Sensed Images. EUSIPCO 2006 (pp. 1- 8). Florence: EURASIP.

[CAI07] Caiafa, C. R., Salerno, E., & Proto, A. (2007). Blind Source Separation Applied to Spectral Unmixing: Comparing Different Measures of Nongaussianity. Lecture Notes of Computer Science (LNCS) , 4694, 1 - 8.

[CAI08] Caiafa, C. F., Salerno, E., Proto, A., & Fiumi, L. (2008). Blind Spectral Unmixing by Local Maximization of Non-gaussianity. Signal Processing , 88 (1), 50 - 68.

[CAR97] J. F. Cardoso, "Statistical principles of source separation". Proceedings of the SYSID'97, 11th IFAC Symposium on System Identification, pp. 1837-1844, 1997.

[CAS07] Marco Antonio Castro Liera, _Algoritmos Genéticos Distribuidos_, La Paz, B.C.S., 2007.

[CAR93] JF. Cardoso, A. Souloumia, "Blind beamforming for non-Gaussian signals". IEE Proceedings (London), Part F, vol. 140, pp.362-370, 1993.

[CAR96b] J.-F. Cardoso and B. Hvam Laheld. Equivariant adaptive source separation. IEEE Trans. on Signal Processing, 44(12):3017-3030, 1996.

[CAR96c] J.F. Cardoso, B. Laheld, "Equivariant adaptive source separation". IEEE Trans. on signal processing, vol. 44, nº 12, pp. 3017-3031, 1996.

[CAR97] J.-F. Cardoso. Infomax and maximum likelihood for source separation. IEEE Letters on Signal Processing, 4:112-114, 1997.

[CAR97] J. F. Cardoso, "Statistical principles of source separation". Proceedings of the SYSID'97, 11th IFAC symposium on system identification, pp. 1837-1844, 1997.

[CAR98] J. F. Cardoso, "Blind signal separation: statistical principles". Proceedings of the IEEE, vol. 86, nº 10, pp.2009-2025, 1998.

[CAR99] J. F. Cardoso, "High order contrasts for independent component analysis". Neural computation, vol. 11, pp. 157-192, 1999.

[CIC97] A. Cichocki, R.E. Bogner, L. Moszczynski, and K. Pope. Modified Herault-Jutten algorithms for blind separation of sources. Digital Signal Processing, 7:80 - 93, 1997.

[CAR96] J.-F. Cardoso and B. Laheld, "Equivariant adaptive source separation," IEEE Transactions on Signal Processing, vol. 44, no. 12, pp. 3017–3030, Dec. 1996.

[CAR97b] J. F. Cardoso, "Infomax and Maximum Likelihood for Blind Source Separation". IEEE Signal Processing Letters, Vol. 4; Issue 4, ISSN: 1070-9908, pp. 112-114, April 1997.

[CHE05] Alian Chen, Lei Hu, Lifeng Chen, Yan Deng, and Xiangning He, "A Multilevel Converter Topology With Fault-Tolerant ability", IEEE Transactions on Power Electronics, vol. 20, No. 2, pp. 405-415, March 2005.

[CIC95] A. Cichocki, W. Kasprzak, S.I. Amari, "Multi-layer neural networks with a local adaptive learning rule for blind separation of source signals". Proceedings of NOLTA'95, Las Vegas (USA), Dec. 10- 14, pp. 61-65.

[CIC96a] A. Cichocki, R. Unbehaven, "Robust neural networks with on-line learning for blind identification and blind separation". IEEE Transactions on circuits and systems I, vol. 43, nº 11, pp.894-906, November 1996.

[CLE99] R.M. Clmente, J.I. Acha, "A new algorithm for the adaptive separation of sources". Proc. of ICA'99, pp. 71-74, January 11-15, Aussois (France), 1999.

[COM10] Comon, J., & Jutten, C. (2010). Handbook of Blind Source Separation: Independent Component Analysis and Applications. Academic Press.

[COM12] Separación ciega de señales adaptativa para procesamiento en tiempo real con DSP, J. Carlos M. Comba , Antonio F. Díaz, M. Rodríguez-Alvarez, J.David M. Comba, Carlos G. Puntonet, Junio 2012.

[COO99] Keith D. Cooper, Philip J. Schielke, and Devika Subramanian. Optimizing for reduced code space using genetic algorithms. In Proceedings of the ACM SIGPLAN 1999 Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES '99), pages 1–9, New York, NY, USA, 1999. ACM Press.

[DAU92] Ingrid Daubechies. Ten Lectures onWavelets. Society for Industrial and AppliedMathematics, Philadelphia, PA, USA, 1992.

[DEL95] N. Delfosse and P. Loubaton. Adaptive blind separation of independent sources: a deflation approach. Signal Processing, 45:59-83, 1995.

[DEV96b] Y. Deville, "Application of the Hérault-Jutten source separation neural network to multi-tag radio-frequency identification systems". Proceedings of Ecole des techniques avancées en signal image parole, pp. 265-272, Grenoble, France, Sept. 2-6 1996.

[DEV99] Y. Deville, J. Damour, N. Charkani, "Improved multi-tag radio-frequency identification systems based on new source separation neural networks". Proc of ICA'99, pp. 449-454, January 11-15, Aussois (France), 1999.

[DEV04] Y. Deville, M. Puigt, and B. Albouy, Time-frequency blind signal separation : extended methods, performance evaluation for speech sources, Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN 2004), pp. 255-260, Budapest, Hungary, July 25-29, 2004.

[DEV10] The C-FICA algorithm. Deville, Yannick - 1er/02/2010.

[DEY11] Marcus R. DeYoung and Brian L. Evans Department of Electrical and Computer Engineering The University of Texas. Blind Source Separation with a Time-Varying Mixing Matrix. 2009.

[DIA98] A. Díaz García, L. Parrilla Roure, M. Rodríguez Alvarez, C. G. Puntonet. "Decodificación de un sonido encriptado mediante la utilización de un procesador digital de señal, Seminario Anual de Automática, Electrónica Industrial e Instrumentación, (SAAEI'98).Universidad Publica de Navarra, pp. 127-128, 16 a 18 de Septiembre de 1998.

[DIA05] A. F. Díaz; C. G. Puntonet; B. Del Pino; A. Cañas, "Plataforma flexible de procesamiento de señales para el aprendizaje de sistemas basados en DSP" JENUI 2005: XI Jornadas de Enseñanza Universitaria de la Informática". pp.397-403.Madrid 2005.

[DON95] D. L. Donoho, I. M. Johnstone, G. Kerkyacharian, and D. Picard. Wavelet shrinkage: asymptopia Journal of the Royal Statistical Society ser. B, 57:301-337, 1995.

[ERD02] Deniz Erdogmus, Yadunandana N. Rao, Jose C. Principe, Jing Zhao, and Kenneth E. Hild II, "Simultaneous extraction of principal components using Givens rotations and output variances," submitted to the IEEE Intl. Conf. on Acoustics, Speech and Signal Proc., ICASSP '02, Orlando, FL, May 2002.

[EVE01] Richard Everson and Stephen Roberts. Independent Component Analysis, Principles and Practice, chapter Particle filters for non-stationary ICA, pages 280–298. Cambridge University Press, 2001.

[FEN99] M. Feng, K-D. Kammeyer, "Application of source separation algorithms for mobile communication environment". Proc of ICA'99, pp. 431-436, January 11-15, Aussois (France), 1999.

[FRA05] B. Franke, M. O'Boyle, J. Thomson, and G. Fursin. Probabilistic source-level optimisation of embedded systems software. In Proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'05), pages 78–86, June 2005.

[FRA07] DSP's Past Can't Hold A Candle to its Future. Gene Frantz, Mar 2007, Texas Instruments.

[FUR02] G.G.Fursin, M.F.P.O'Boyle, and P.M.W. Knijnenburg. Evaluating iterative compilation. In Proceedings of the 15th Workshop on Languages and Compilers for Parallel Computing (LCPC'02), 2002.

[FUR05] Grigori Fursin, Albert Cohen, Michael O'Boyle, and Oliver Temam. A practical method for quickly evaluating program optimizations. In Proceedings of the 1st International Conference on High Performance Embedded Architectures and Compilers (HiPEAC 2005), pages 29–46, November 2005.

[GAE90] M. Gaeta, J. L. Lacoume, "Source separation wthout a priori knowledge: the maximum likelihood solution", Proc. EUSIPCO, pp. 621-624, 1990.

[GEL99] G. Gelle, M. Colas, G. Delaunay, "Separation of convolutive mixtures of harmonic signals with a temporal approach. Application to rotating machine monitoring". Proc of ICA'99, pp. 109-114, January 11-15, Aussois (France), 1999.

[GIA98] X. Giannakopoulos, J. Karhunen, and E. Oja. Experimental comparison of neural ICA algorithms. In Proc. Int. Conf. on Artificial Neural Networks (ICANN'98), pages 651-656, Skövde, Sweden, 1998.

[GRA90] R. Gray, "Entropy and Information Theory". New York: Springer-Verlag, 1990.

[GRA03] A. Grajdeanu and K. A. De Jong. Fixed budget allocation strategy for noisy fitness landscapes. In Late Breaking Papers of the Genetic and Evolutionary Computation Conference (GECCO-2003), 2003.

[HAB00] G. Haber, E. A. Henis, and V. Eisenberg. Reliable post-link optimizations based on partial information. In Proceedings of Feedback Directed and Dynamic Optimizations.3 Workshop, pages 91 – 100, December 2000..

[HAN97] D. Hanselman, B. Littlefield, "The Student Edition of MATLAB"., Prentice-Hall, 1997.

[HAR05] Harte, N., Hurley, N., Fearon, C., Rickard, S. (2005). Towards a hardware realization of time frequency source separation of speech. In Proceedings of the IEEE European Conf. on Circuit Theory and Design, Vol. 1, 71-74.

[HAY99] S. Haykin, "Neural Networks", Prentice Hall, 1999.

[HEN01] Programmable Digital Signal Processors: Architecture, Programming, and Applications. Hu, Yu Hen. New York, NY, USA: Marcel Dekker Incorporated, 2001.

[HIL01a] Kenneth E. Hild II, Deniz Erdogmus, and Jose C. Principe, "Blind source separation using Renyi's Mutual Information," IEEE Signal Proc. Letters, Vol. 8, No. 6, pp. 174-176, June 2001.

[HIL01b] Kenneth E. Hild II, Deniz Erdogmus, and Jose C. Principe, "On-line Minimum Mutual Information Method For Timevarying Blind Source Separation," accepted to Third Intl. Workshop on Independent Component Analysis and Signal Separation, ICA '01, San Diego, CA, Dec. 2001.

[HIL09] Kenneth E. Hild II, Deniz Erdogmus, and Jose C. Principe Computational NeuroEngineering Laboratory (www.cnel.ufl.edu) The University of Florida, Gainesville, Florida, 32611, USA. Blind source separation of time-varying, instantaneous mixtures using an on-line algorithm. 2009.

[HIN05] Marko Hinkkanen, Veli-Matti Leppänen, and Jorma Luomi, "Flux Observer Enhanced with Low-Frecuency Signal Injection Allowing Sensorless Zero-Frecuency Operation of Induction Motors", IEEE. Transactions on Industry applications, vol. 41, No. 1, pp. 52-59, Jan/Feb 2005.

[HUA97] Howard Hua Yang and Shun-ichi Amari, "Adaptive online learning algorithms for blind separation: maximum entropy and minimum mutual information," Neural Computation, Vol. 9, No. 7, pp. 1457-1482, Oct. 1997.

[HYV97] A. Hyvärinen and E. Oja. A fast fixed-point algorithm for independent component analysis. Neural Computation, 9(7):1483-1492, 1997.

[HYV98] A. Hyvärinen and E. Oja. Independent component analysis by general nonlinear Hebbian-like learning rules. Signal Processing, 64(3):301-313, 1998.

[HYV98] A. Hyvärinen. Independent component analysis in the presence of gaussian noise by maximizing joint likelihood. Neurocomputing, 22:49-67, 1998.

[HYV98] A. Hyvärinen. New approximations of differential entropy for independent component analysis and projection pursuit. In Advances in Neural Information Processing Systems, volume 10, pages 273-279. MIT Press, 1998.

[HYV99a] A. Hyvärinen. Fast and robust fixed-point algorithms for independent component analysis. IEEE Trans. on Neural Networks, 10(3):626-634, 1999.

[HYV99b] A. Hyvärinen. The fixed-point algorithm and maximum likelihood estimation for independent component analysis. Neural Processing Letters, 10(1):1-5, 1999.

[HYV99c] A. Hyvärinen. Gaussian moments for noisy independent component analysis. IEEE Signal Processing Letters, 6(6):145-147, 1999.

[HYV99d] A. Hyvärinen. Sparse code shrinkage: Denoising of nongaussian data by maximum likelihood estimation. Neural Computation, 11(7):1739-1768, 1999.

[HYV99e] A. Hyvärinen. Survey on independent component analysis. Neural Computing Surveys, 2:94-128, 1999.

[HYV00] A. Hyvarinen, E. Oja, "Independent Component Analysis: algorithm and applications". Neural Networks, vol 13 , March 28, pp. 411-430, Elsevier Science, 2000.

[HYV03] A. Hyvärinen, J. Karhunen, E. Oja. "Independent Component Analysis" John Wiley & Sons, Inc. 2003.

[ING98] Ingole, P.V., Sapkal, A.K., Sarate, G.G., Hirekhan, S.R. (2011). Implementation of signal generator (DSP) using TMS 320C6713 DSK. International Journal of Computer Science & Emerging Technologies, 1 (2), 95-98.

[IVA07] Ivancescu, G. (2007). Fixed point arithmetic and tricks. Retrieved August 12, 2011, from http://x86asm.net/articles/fixed-point-arithmetic-andtricks/

[JEN03] R. Jenssen and T. Eltoft "ICA FilterBank for Segmentation of Textured Images" .4th International Symposium on Independent Component Analysis and Blind Signal Separation, Proc of ICA'2003, April 2003, Nara (Japan), 2003.

[JUT91] C. Jutten, and J. Hérault, "Blind Separation of sources, part I, II, III. Signal Processing, vol.24, pp.1-29, 1991.

[JUT91] C. Jutten and J. Herault. Blind separation of sources, part I: An adaptive algorithm based on neuromimetic architecture. Signal Processing, 24:1-10, 1991.

[JUT97] C. Jutten, "From source separation to independent component analysis", Proceedings of the European Symposium on Artificial Neural Networks (ESANN'97), Bruges, April 16-18, 1997, pp. 243-248, D facto, Brussels, 1997.

[JUT10] Comon, P.; Jutten C., (2010): "Handbook of Blind Source Separation, Independent Component Analysis and Applications". Academic Press, Oxford UK. ISBN 978-0-12-374726-6- M.P.S. Chawla, H.K. Verma, Vinod Kumar. "A new statistical PCA-ICAalgorithm for location of R-peaks in ECG". International Journal of Cardiology, Volume 129, Issue 1, 16 September 2008, Pages 146-148. - Miljkovic, N. ; Matice, V. ; Van Huffel, S. ; Popovic, M.B. "Independent Component Analysis (ICA) methods for neonatal EEG artifact extraction: Sensitivity to variation of artifact properties". 10th Symposium on Neural Network Applications in Electrical Engineering (NEUREL), 23-25 September, 2010. Belgrado (Serbia).

[KAR96] J. Karhunen, "Neural approaches to independent component analysis and source separation". Proc. 4th European Symposium on Artificial Neural Networks (ESANN'96), pp. 249-266, Bruges, April 1996.

[KAR97] J. Karhunen, E. Oja, L. Wang, R. Vigario, J. Joutsensolo, "A class of neural networks for independent component analysis". IEEE Transactions on neural networks, vol. 8, no. 3, pp. 486-504, May 1997.

[KEN01] Blind source separation of time-varying instantaneous mixtures using an on-line algorithm. Kenneth E. Hild II, Deniz Erdogmus, and José C. Príncipe, 2001.

[KRI] Krishnaveni, V., Jayaraman, S., Arvind, S., Hariharasudhan, V., Ramdoss, K. (2006). Automatic identification and removal of ocular artifacts from EEG using wavelet transform. Measurement Science Review, 6, 45-57.

[KUL06] A. P. Kulkarni, D. B. Whalley, G. S. Tyson, and J. W. Davidson. Exhaustive optimization phase order space exploration. In Proceedings of the International Symposium on Code Generation and Optimization, 2006.

[KUO04] Digital Signal Processors Architectures, Implementations, and applications. Sen M. Kuo and Woon –Seng Gan. Ed. Prentice Hall. 2004.

[KUO08] Kuo-Kai Shyu, Ming-Huan Lee and Po-Lei Lee, "Implementation of Pipelined FastICA on FPGA for Real-Time Blind Source Separation". IEEE transactions on Neural Networks, pp 958-970, Vol. 19, No. 6, June 2008.

[KWA05] Sangshin Kwak, Hamid A. Toliyat, "A Hybrid Solution for Load-Commutated-Inverter-Fed Induction Motor Drives", IEEE Transactions on Industry Applications, vol. 41, No. 1, pp. 83-90, Jan/Feb 2005.

[LAC88] J. L. Lacoume, P. Ruiz, "Sources identification: a solution based on the cumulants". In Proc. 4th ASSP Workshop spectral estimation modelling, pp. 199-203, Minneapolis, 1988.

[LAC99] J. L. Lacoume, "A survey of source separation". Proc of ICA'99, pp. 1-6, January 11-15, Aussois (France), 1999.

[LAH94] B. Lahed, "Séparation auto-adaptative de sources. Implantation et performances". Tesis, Escuela nacional superior de telecomunicaciones de París, 1994.

[LAP97] DSP Processor Fundamentals Architectures and Features. Phil Lapsley and Jeff Bier, IEEE.1997.

[LAU06] Lauha, J. (2006). The neglected art of Fixed point arithmetic. Retrieved August 12, 2011, from http://jet.ro/files/The_neglected_art_of_Fixed_point_arithmetic_20060913.pdf.

[LEE99] T.-W. Lee, M. Girolami, and T. J. Sejnowski. Independent component analysis using an extended infomax algorithm for mixed sub-gaussian and super-gaussian sources. Neural Computation, 11(2):417-441, 1999.

[LEO05] Lux Roxana De León Lomeli,"Detección de Interarmonicos Usando Wavelets", Tesis de Maestría, Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET), Departamento de Ingeniería Electrónica, Cuernavaca, Morelos, México. 7 de enero 2005.

[LEÑ05] Implementación de un algoritmo para la separación ciega de fuentes con pequeños retardos. Juan Antonio Leñero Badallo, Sergio A. Cruces Álvarez, Septiembre 2005.

[MAL89] S. G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. IEEE Trans. on PAMI, 11:674-693, 1989.

[MAL99] Stephane Mallat. A Wavelet Tour of Signal Processing. Academic Press, second edition, 1999. ISBN 0-12-466606-X.

[MAN95] A. Mansour, C. Jutten. "Fourth-Order criteria for blind separation". IEEE Transactions on signal processing, Vol. 43 nº8, August 1995, pp. 2022-2025.

[MAR99] R. Martín Clemente, J. I. Acha. "A new algorithm for the adaptive separation of sources". ". Proc of ICA'99, pp. 257-260, January 11-15, Aussois (France), 1999.

[MEN95] W. Mendenhall, T. Sincich. "Probabilidad y estadística para ingeniería y ciencias". Prentice Hall, 1995.

[MIN03] C. Min Kim, H. Min Park, T. Kim, Y. Kyung Choi, S. Young"FPGA Implementation of ICA Algorithm for Blind Signal Separation and Adaptive Noise Canceling". IEEE transactions on Neural Networks, pp. 1038-1046, Vol. 14, No. 5, September 2003.

[MIT93] S. K. Mitra, J. F. Kaiser (Eds.). "Handbook for digital signal processing". John Wiley & Sons, 1993.

[MOR05] Shigeo Morimoto, Hideaki Nakayama, Masayuki Sanada, and Yoji Takeda, "Sensorless Output Maximization Control for Variable-Speed Wind Generation Systems Using IPMSG", IEEE. Transactions on Industry Applications, vol. 41, No. 1, pp. 60-67, Jan/Feb 2005.

[NAK97] S. Nakamura, "Análisis numérico y visualización gráfica con MATLAB". Prentice-Hall,1997.

[NAS05] M. Nasir Uddin, M.A. Abido, and M.A. Rahman, "Real-Time Performance Evaluation of a Genetic-Algorithm-Based Fuzzy Logic Controller for IMP Motor Drives", IEEE Transactions on Industry. Applications, vol. 41, No. 1, pp. 246-252, Jan/Feb 2005.

[NGO99] J.T. Ngo, N. A. Bhadkamkar, "Adaptive blind separation of audio sources by a physically compact device using second-order statistics". Proc of ICA'99, pp. 257-260, January 11-15, Aussois (France), 1999.

[NGU95] H. L. Nguyen, C. Jutten, "Blind source separation for convolutive mixtures". Siganl Processing, vol. 45, pp. 209-229, 1995.

[NUZ99] D. Nuzillard, J-M. Nuzillard, "Blind source separation applied to non-orthogonal signals". Proc of ICA'99, pp. 25-30, January 11-15, Aussois (France), 1999.

[OJA85] E. Oja, J. Karhunen, “An stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix”. J. Math. Anal. Appl. vol. 106, pp. 69-84, 1985.

[OJA97] E. Oja, “The non-linear PCA learning rule in independent component analysis”. Neurocomputing vol. 17, pp. 25-45, 1997.

[OJA95] E. Oja, J. Karhunen, L. Wang, and R. Vigario, "Principal and independent components in neural-networks -recent developments", In Proc. Italian Workshop on Neural Networks, WIRN'95, Vietri, Italy, 1995.

[OLI06] L. N. Oliva, J.A. Moreno, L. M. Flores and F. Gomez, “DSP Implementation of Extended Infomax ICA Algorithm for Blind Source Separation”, 3rd International Conference on Electrical and Electronics Engineering (ICEEE 2006), Veracruz, Mexico, September 6-8, 2006.

[OLI06b] L. N. Oliva, A. Oliverio, J.A. Moreno, L. M. Flores y F. Gómez, “Implementación de un Algoritmo ICA con Circuitos CMOS Analógicos para el Problema de Separación Ciega de Fuentes”, XII Workshop Iberchip , San José, Costa Rica, 22 a 24 de Marzo de 2006.

[OLI08] L. N. Oliva, M. A. Aleman, J. García, J. A. Moreno, “Implementation of Blind Source Separation for Multi-input”, Proceedings of the 5th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE 2008), pp. 470-474. ISBN: CFP08827-CDR. Mexico City, Mexico, Nov. 12-14, 2008.

[OVA07] Roberto II Ovando Domínguez, "Emulador de Turbina Eólica para el Banco de Pruebas de Generación Eoloeléctrica", Tesis de Maestría, Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET), Departamento de Ingeniería Electrónica, Cuernavaca, Morelos, México. 13 de Julio 2007.

[PEA97] B. A. Pearlmutter and L. C. Parra. Maximum likelihood blind source separation: A context-sensitive generalization of ica. In Advances in Neural Information Processing Systems, volume 9, pages 613-619, 1997.

[PHA92] D.-T. Pham, P. Garrat, and C. Jutten. Separation of a mixture of independent sources through a maximum likelihood approach. In Proc. EUSIPCO, pages 771-774, 1992.

[PIN04] R. P. J. Pinkers, P. M. W. Knijnenburg, M. Haneda, and H. A. G. Wijshoff. Statistical selection of compiler options. In Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS '04), pages 494–501, Washington, DC, USA, 2004. IEEE Computer Society.

[PRI97] A. Prieto, C. G. Puntonet, B. Prieto, and M. Rodríguez-Alvarez, "A Competitive Neural Network for Blind Separation of Sources Based on Geometric Properties". Lecture Notes in Computer Science, vol. 1240, pp.1095-1106, Springer-Verlag, 1997.

[PRI98] A. Prieto, C. G. Puntonet, B. Prieto; "A Neural Learning Algorithm for blind separation of sources based on geometric properties". Signal Processing, Vol.64, No. 3, pp. 315-331, 1998.

[PRI99a] A.Prieto, B.Prieto, C,G,Puntonet, A.Cañas, P.Martín-Smith, "Geometric separation of linear mixtures of sources: Application to speech signals",International workshop on Independent Component Analysis and Blind Signal Separation (ICA'99) , pp. 295-300, Aussois, France, January 11-15, 1999.

[PRI99b] B. Prieto, A. Prieto, C. G. Puntonet, M. Rodríguez, "A software tool to simulate linear mixture separation algorithms", Demostración en el "1st. Inter. Workshop on Independent Component Analysis and Signal separation (ICA'99)", January 11-15, 1999 Aussois, France.

[PRI99c] B. Prieto, "", Tesis Doctoral, Nuevos algoritmos para Separación Ciega de Fuentes utilizando métodos geométricos", Tesis Doctoral, Departamento Arquitectura de Computadores, Universidad de Granada, 1999.

[PRO98] Tratamiento Digital de Señales. Principios, algoritmos y aplicaciones. John G. Proakis, Dimitris G.Manolakis. Ed. Prentice Hall. 1998.

[PUG99] A. T. Puga, A. P. Alves, "An experiment on comparing PCA and ICA in classical transform image coding", proceedings of ICA'99, pp. 105-108, Aussois, France, Jan 11-15, 1999.

[PUI05] M. Puigt and Y. Deville, Time-frequency ratio-based blind separation methods for attenuated and time-delayed sources, Section 3 : "Summary of the TIFROM method for linear instantaneous mixtures", Mechanical

Systems and Signal Processing, vol. 19, issue 6, pp. 1348-1379, November 2005.

[PUI08] Linear Instantaneous TIme-Frequency Ratio Of Mixtures. Puigt, Matthieu - 27/10/2008.

[PUN95a] C. G. Puntonet, A. Prieto, C. Jutten, M. Rodríguez-Alvarez, and J. Ortega, "Separation of sources: a geometry-based procedure for reconstruction of n-valued signals". Signal Processing, vol. 46, no. 3, pp. 267-284, 1995.

[PUN95b] C.G.Puntonet, M. Rodriguez-Alvarez,; A. Prieto, "A Geometrical Based Procedure for Source Separation Mapped to a neural Network", Lecture Notes in Computer Science, Vol.930, pp.736-743, Springer Verlag, 1995.

[PUN95c] C.G.Puntonet, A. Prieto, "An adaptive geometrical procedure for blind separation of sources", Neural Processing Letters, Vol.2, No.5, pp.23-27, Sept. 1995.

[PUN95d] C. G. Puntonet, A. Mansour, C. Jutten, "Geometrical Algorithm for Blind Separation of Sources", Proc. of 15me. Colloque sur le Traitement du Signal et des Images (GRETSI'95), Vol.1, pp. 273-276. Juan les Pins, 18-22 Sept. 1995.

[PUN96a] C.G.Puntonet, M. Rodriguez, A.Prieto. "Adaptive blind separation of signals through a new geometrical approach". IASTED-96 (Artificial Intelligence, Expert Systems and Neural Networks, Honolulu, Hawaii (USA), 19-22 Agosto, 1996.

[PUN96b] C.G.Puntonet, A.Prieto, J. Ortega. "New geometrical approach for blind separation of sources mapped to a neural network". NICROSP-96, International Workshop on Neural Networks for Identification, Control, Robotics, and Signal-Image Processing, pp.174-182, 21-23 Agosto, 1996, Venecia, Italia.

[PUN96c] C.G.Puntonet, M. Rodriguez, A.Prieto."Blind separation of signals through a geometrical approach".IIZUKA-96, 4th International Conference on Soft Computing, pp.667-670, 30 Septiembre al 5 Octubre, 1996, Fukuoka, Japón.

[PUN97a] C. G. Puntonet, A. Prieto, "Geometric approach for blind separation of signals", Electronic Letters, vol.33, no. 10, pp. 835-836, 1997.

[PUN97b] Puntonet, C.G., Rodriguez-Alvarez, M, Prieto, A: "A geometric method for blind separation of sources". Artificial Intelligence and Soft Computing,

IASTED-97, Banff, Canada, 27-31 July, pp. 372-375, IASTED Acta Press, 1997.

[PUN98a] C.G.Puntonet, A. Prieto, "Neural net approach for blind separation of sources based on geometric properties", Neurocomputing Vol. 18, No.1-3, pp. 141-164, Jan. 1998.

[PUN98b] C.G. Puntonet, M. R. Alvarez, A. Prieto ,B. Prieto, "Separation of Sources in a Class of Post-Nonlinear Mixtures", European Symposium on Artificial Neural Networks (ESANN'98), pp. 321-326, IEEE, Bruges- April 22-23-24, 1998.

[PUN99] C.G. Puntonet; M. R. Alvarez; A. Prieto; B. Prieto. "Separation of Speech Signals for Nonlinear Mixtures". Lecture Notes in Computer Science, vol. 1607, pp.665-673, Springer-Verlag, 1999.

[SHI05] Yen-Shin and Fu-San Shyu, "Optimal Common-Mode Voltaje Reduction PWM Technique for Invertir Control With Consideration of the Dead-Time Effects---Part I: Basic Development", IEEE. Transactions on Industry Applications, vol. 40, No. 6, pp. 1605-1612, Jan/Feb 2005.

[SHY06] Shyu, K., Li, M. (2006). FPGA Implementation of FastICA based on Floating point arithmetic design for real-time blind source separation. In Proceedings of the IEEE Int. Joint Conf. on Neural Networks, 2785-2792.

[RIS99] T. Ristaniemi and J. Joutsensalo. On the performance of blind source separation in CDMA downlink. in Proc. Int. Workshop on Independent Component Analysis and Signal Separation (ICA'99), pages 437-441, Aussois, France, 1999.

[RIS99] T. Ristaniemi, J. Joutsensalo, "On the performance of blind source separation in CDMA downlink". Proc of ICA'99, pp. 437-442, January 11-15, Aussois (France), 1999.

[ROD96] V. Rodellar, V. García, P. Gómez, E. Martínez, V. Peinado. "Optimal arithmetic dimension of adaptive filters for speech characterization". Proc. of the MELECON'96, Bari, Italy, 1996.

[SCH99] I. Schief, M. Stetter, J. E. M. Mayhew, S. Askew, N. McLoughlin, J. B. Levitt, J. S. Lund, K. Obermayer, "Blind separation of spatial signal patterns from optical imaging records". Proc of ICA'99, pp. 179-184, January 11-15, Aussois (France), 1999.

[SIV09] S.N. Sivanadam, _Introduction to Genetic Algorithms_, Springer, 2009.

[SIM05] MSC711x Overview. Donald Simond. Aplication Note. No. AN3056 Rev. 0, Dic. 2005. Freescale Semiconductor.

[SMI97] Smith, S.W. The Scientist and Engineer's Guide to Digital Signal Processing. California Technical Publishing, 1997. ISBN 0-9660176-3-3.

[STE03] M. Stephenson, U. O'Reilly, M. Martin, and S. Amarasinghe. Genetic programming applied to compiler heuristic optimisation. In Proceedings of The 6th European Conference on Genetic Programming, 2003.

[SUV00] Suvak, Dave, IrDA and Bluetooth: A complementary comparison, Extended systems Inc. 2000.

[TAL98] A. Taleb, C. Jutten, S. Olympief, "Source separation in post non-linear mixtures: an entropy-based algorithm", ICASSP'98, Seattle (USA).

[TEX04] Texas Instruments, TMS320C6711 /11B /11C /11D Floating-Point Digital Signal Processors (Rev. L) Data Sheet.. Mayo 2004.

[TEX05] Texas Instruments Inc. (2001, revised 2005). Floating-Point Digital Signal Processors (manual, sprs1861). Houston, Texas.

[TEX10] Texas Instrument, TMS320C6416DSK Technical Reference, 2010.

[TEX11] Texas Instrument, Introduction to TMS320C6000 DSP Optimization, Paul Yin, October 2011.

[THA05] L. He, T. Thaiupathump, and S. Kassam, "Blind separation of complex I/Q independent sources with phase recovery," IEEE Signal Processing Letters, vol. 12, no. 5, pp. 419–422, May 2005.

[THI95] H. N. Thi, C. Jutten, "Blind source separation for convolutive mixtures". Signal Processing, vol. 45, nº 2, pp. 209-299, 1995.

[THO09] Blind Source Separation Softwares. Thomas, Johan - 29/04/2009.

[TOR99] K. Torkkola, "Blind separation for audio signals – are we there yet?". Proc of ICA'99, pp. 239-244, January 11-15, Aussois (France), 1999.

[TRI03] S. Triadaphillou, A.J.Morris and E.B. Martin "Application of Independet Analysis to Chemical Reactions". 4th International Symposium on Independent Component Analysis and Blind Signal Separation, Proc of ICA'2003, April 2003, Nara (Japan), 2003.

[VEG01] Fundamentals of Digital Signal Processing. Joyce Van de Vegte. Ed. Ed. Prentice Hall. 2001.

[WON00] Te-Won Lee "Independent Component Analysis Theory and applications" Kluwer Academic Publishers. 2000.

[YAT09] Yates, R. (2009). Fixed point arithmetic: An Introduction. Retrieved August 12, 2011, from http://www.digitalsignallabs.com.

[YUN05] Ji-Yun Seol and In-Joong Ha, "Feedback-Linearizing Control of IPM Motors Considering Magnetic Saturation Effect, IEEE Transactions on Power Electronics, vol. 20, No. 2, pp. 416-424, March 2005.

[WES99] A. Westner, V. M. Bove, "Blind separation of real world audio signals using overdetermined mixtures". Proc of ICA'99, pp. 251-256, January 11-15, Aussois (France), 1999.

[WHI94] D. Whitley. A genetic algorithm tutorial. Statistics and Computing, 4(2):65–85, 6, 1994.

[WHI97] D. Whitley, S. Rana, and R. B. Heckendorn, _ Island Model Genetic Algorithms and Linearly Separable Problems_, In Evolutionary Computing, AISB Workshop pp. 109-125, Springer-Verlag, 1997.

[YAN97] H. H. Yang, and S. Amari, "Adaptive on-line learning algorithms for blind separation maximum entropy and minimum mutual information", Neural Computation, vol.9, pp.1457-1482, 1997.

[ZHO02] Keliang Zhou, Danwei Wang: "Relationship Between Space Vector Modulation and Three Phase Carrier Based PWM. A Comprehensive Analysis", IEEE Transactions on Industrial Electronics, vol. 49, No. 1, Febrero 2002.

[ZIB01] M. Zibulevsky, B. A. Pearlmutter, P. Bofill, and P. Kisilev. Independent Component Analysis, Principles and Practice, chapter Blind source separation by sparse decomposition in a signal dictionary, pages 181–208. Cambridge University Press, 2001.

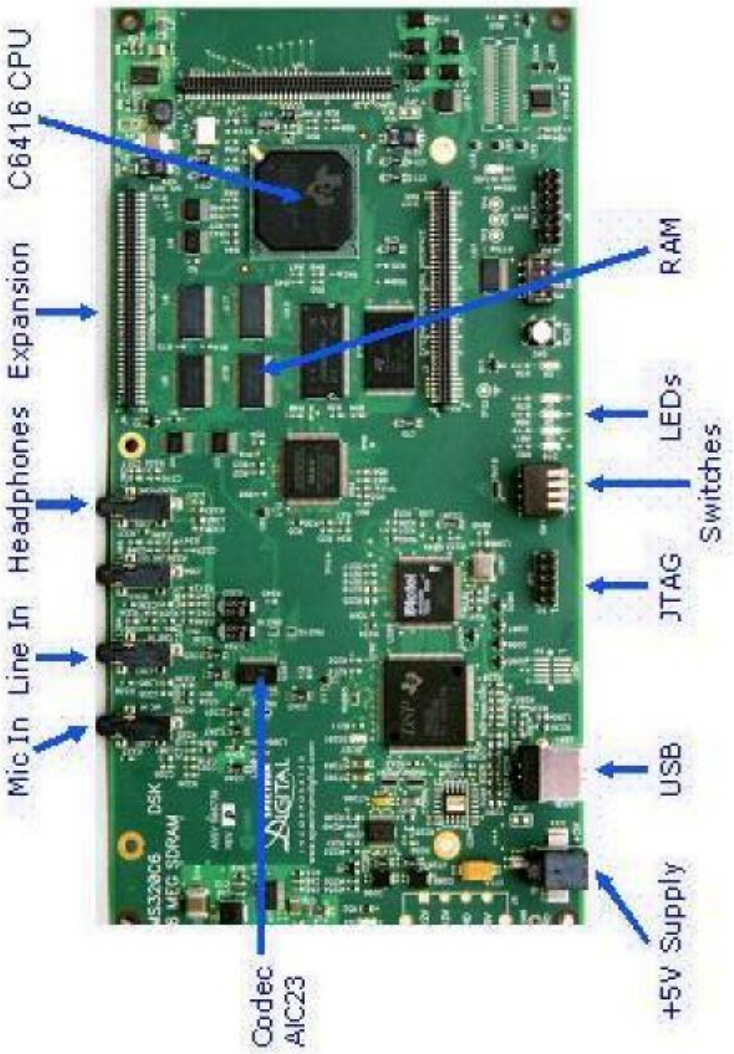
ACRÓNIMOS

AC	Corriente alterna (Alternating Current).
A/D	Conversión Analógica-Digital
CAD	Conversión Analógica-Digital
CDA	Conversión Digital-Analógica
CDMA	Comunicación de múltiple acceso con división de código.
CV	Coeficiente de variación.
CVSD	Modulado de la variable continua pendiente delta.
D/A	Conversión Digital-Analógica
DC	Corriente continua (Direct Current).
DSP	Procesador Digital de Señales (Digital Signal Processor).
ECG	Electrocardigrama
EGV	Autovalor (Eligenvalue).
FFT	Transformada Rápida Fourier (Fast Fourier Transform)
FIR	Respuesta impulso finita (Finite Impulse Response)
HF	Alta frecuencia (High Frecuency).
I	Información mutua.
ICA	Análisis de Componentes Independientes ("Independent Component Analysis").
IIR	Respuesta impulso infinita (Infinite Impulse Response)
LSB	Bit menos significativo (Less Significative Bit).
MI	Información Mutua ("Mutual Information").
ML	Máxima verosimilitud ("Maximum Likelihood").
MSB	Bit más significativo (More Significative Bit).
N	Número total de muestras.
PCA	Análisis de Componentes Principales ("Principal Components Analysis").

PDF	Función de densidad de probabilidad.
PDS	Procesamiento Digital de Señales.
PMF	Función de probabilidad de masa.
PNL	Modelo Post-Nonlineal.
SIMD	Simple Instruction Multiple Data
SFR	Special Function Registers.
SNR	Relación Señal/Ruido (“Signal to Noise Rate”).
S/H	Muestreo y retención (Sample and Hold).
TI	Texas Instruments.
VLIW	Very-Long Instruction Word
σ	Desviación estándar.
σ2	Varianza.

Apéndice A

Placa TMS320C6416 DSK


C6416 CPU
Expansion
Headphones
Line In
Mic In
RAM
LEDs
Switches
JTAG
USB
+5V Supply
Codec
AIC23

SPECTRUM DIGITAL INCORPORATED

TMS320C6416 DSK

Technical Reference

1.1 Key Features

The C6416 DSK is a low-cost standalone development platform that enables users to evaluate and develop applications for the TI C64xx DSP family. The DSK also serves as a hardware reference design for the TMS320C6416 DSP. Schematics, logic equations and application notes are available to ease hardware development and reduce time to market.

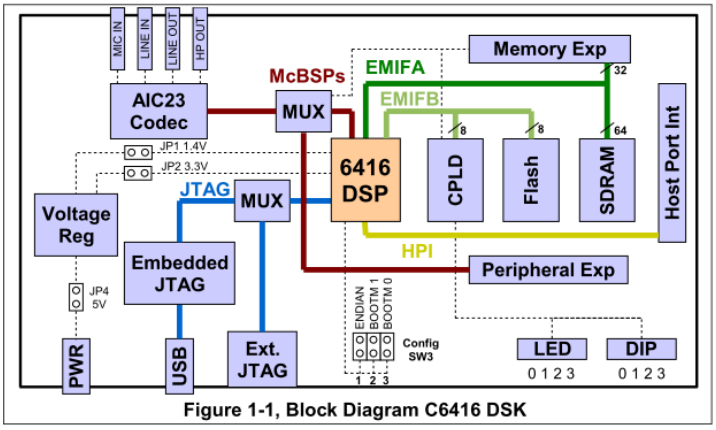


Figure 1-1, Block Diagram C6416 DSK

The DSK comes with a full compliment of on-board devices that suit a wide variety of application environments. Key features include:

- A Texas Instruments TMS320C6416 DSP operating at 600 MHz.
- An AIC23 stereo codec
- 16 Mbytes of synchronous DRAM
- 512 Kbytes of non-volatile Flash memory
- 4 user accessible LEDs and DIP switches
- Software board configuration through registers implemented in CPLD
- Configurable boot options
- Standard expansion connectors for daughter card use
- JTAG emulation through on-board JTAG emulator with USB host interface or external emulator
- Single voltage power supply (+5V)

TMS320C6414, TMS320C6415, TMS320C6416
FIXED-POINT DIGITAL SIGNAL PROCESSORS

SPRS146N – FEBRUARY 2001 – REVISED MAY 2005

- Highest-Performance Fixed-Point Digital Signal Processors (DSPs)
  - 2-, 1.67-, 1.39-ns Instruction Cycle Time
  - 500-, 600-, 720-MHz Clock Rate
  - Eight 32-Bit Instructions/Cycle
  - Twenty-Eight Operations/Cycle
  - 4000, 4800, 5760 MIPS
  - Fully Software-Compatible With C62x™
  - C6414/15/16 Devices Pin-Compatible
- VelociTI.2™ Extensions to VelociTI™ Advanced Very-Long-Instruction-Word (VLIW) TMS320C64x™ DSP Core
  - Eight Highly Independent Functional Units With VelociTI.2™ Extensions:
    - Six ALUs (32-/40-Bit), Each Supports Single 32-Bit, Dual 16-Bit, or Quad 8-Bit Arithmetic per Clock Cycle
    - Two Multipliers Support Four 16 x 16-Bit Multiplies (32-Bit Results) per Clock Cycle or Eight 8 x 8-Bit Multiplies (16-Bit Results) per Clock Cycle
  - Non-Aligned Load-Store Architecture
  - 64 32-Bit General-Purpose Registers
  - Instruction Packing Reduces Code Size
  - All Instructions Conditional
- Instruction Set Features
  - Byte-Addressable (8-/16-/32-/64-Bit Data)
  - 8-Bit Overflow Protection
  - Bit-Field Extract, Set, Clear
  - Normalization, Saturation, Bit-Counting
  - VelociTI.2™ Increased Orthogonality
- Viterbi Decoder Coprocessor (VCP) [C6416]
  - Supports Over 600 7.95-Kbps AMR
  - Programmable Code Parameters
- Turbo Decoder Coprocessor (TCP) [C6416]
  - Supports up to 7 2-Mbps or 43 384-Kbps 3GPP (6 Iterations)
  - Programmable Turbo Code and Decoding Parameters
- L1/L2 Memory Architecture
  - 128K-Bit (16K-Byte) L1P Program Cache (Direct Mapped)
  - 128K-Bit (16K-Byte) L1D Data Cache (2-Way Set-Associative)
  - 8M-Bit (1024K-Byte) L2 Unified Mapped RAM/Cache (Flexible Allocation)
- Two External Memory Interfaces (EMIFs)
  - One 64-Bit (EMIFA), One 16-Bit (EMIFB)
  - Glueless Interface to Asynchronous Memories (SRAM and EPROM) and Synchronous Memories (SDRAM, SBSRAM, ZBT SRAM, and FIFO)
  - 1280M-Byte Total Addressable External Memory Space
- Enhanced Direct-Memory-Access (EDMA) Controller (64 Independent Channels)
- Host-Port Interface (HPI)
  - User-Configurable Bus Width (32-/16-Bit)
- 32-Bit/33-MHz, 3.3-V PCI Master/Slave Interface Conforms to PCI Specification 2.2 [C6415/C6416]
  - Three PCI Bus Address Registers: Prefetchable Memory, Non-Prefetchable Memory I/O
  - Four-Wire Serial EEPROM Interface
  - PCI Interrupt Request Under DSP Program Control
  - DSP Interrupt Via PCI I/O Cycle
- Three Multichannel Buffered Serial Ports
  - Direct I/F to T1/E1, MVIP, SCSA Framers
  - Up to 256 Channels Each
  - ST-Bus-Switching-, AC97-Compatible
  - Serial Peripheral Interface (SPI) Compatible (Motorola™)
- Three 32-Bit General-Purpose Timers
- Universal Test and Operations PHY Interface for ATM (UTOPIA) [C6415/C6416]
  - UTOPIA Level 2 Slave ATM Controller
  - 8-Bit Transmit and Receive Operations up to 50 MHz per Direction
  - User-Defined Cell Format up to 64 Bytes
- Sixteen General-Purpose I/O (GPIO) Pins
- Flexible PLL Clock Generator
- IEEE-1149.1 (JTAG†) Boundary-Scan-Compatible
- 532-Pin Ball Grid Array (BGA) Package (GLZ, ZLZ and CLZ Suffixes), 0.8-mm Ball Pitch
- 0.13-μm/6-Level *Cu* Metal Process (CMOS)
- 3.3-V I/Os, 1.2-V/1.25-V Internal (500 MHz)
- 3.3-V I/Os, 1.4-V Internal (600 and 720 MHz)

C62x, VelociTI.2, VelociTI, and TMS320C64x are trademarks of Texas Instruments.
Motorola is a trademark of Motorola, Inc.
† IEEE Standard 1149.1-1990 Standard-Test-Access Port and Boundary Scan Architecture.

Apéndice B

1. Iteración tamaño constante

```
;** --------------------------------------------------------------------*
;           EXCLUSIVE CPU CYCLES: 4
           MV      .L1     A4,A14            ; |51|
      .dwpsn      file "../clk.c",line 52,column 11,is_stmt
           MVC     .S2     CSR,B17

           AND     .L2     -2,B17,B4
||         MVKL    .S1     _buf,A9
||         MVK     .S2     0x100,B0          ; |52|

           MVC     .S2     B4,CSR            ; interrupts off
||         SUB     .L2     B0,2,B0
||         MVKH    .S1     _buf,A9
||         MVK     .L1     0x2,A0            ; init prolog collapse
predicate

;*--------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop source line                 : 52
;*      Loop opening brace source line   : 52
;*      Loop closing brace source line   : 57
;*      Loop Unroll Multiple             : 16x
;*      Known Minimum Trip Count         : 256
;*      Known Maximum Trip Count         : 256
;*      Known Max Trip Count Factor      : 256
;*      Loop Carried Dependency Bound(^) : 0
;*      Unpartitioned Resource Bound     : 4
;*      Partitioned Resource Bound(*)    : 4
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     0        0
;*      .S units                     0        1
;*      .D units                     4*       0
;*      .M units                     4*       4*
;*      .X cross paths               0        0
;*      .T address paths             2        2
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          4        4     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             0        1
;*      Bound(.L .S .D .LS .LSD)     3        2
;*
```

```
;*      Searching for software pipeline schedule at ...
;*         ii = 4  Schedule found with 4 iterations in parallel
;*
;*      Register Usage Table:
;*          +-------------------------------------------------------------+
;*
|AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA|BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB|
;*
|00000000001111111111222222222233|00000000001111111111222222222233|
;*
|01234567890123456789012345678901|01234567890123456789012345678901|
;*           |--------------------------------+------------------------|
;*       0: |*  *******                       |**  ******      *
|
;*       1: |*  *******      *                |**  ******
|
;*       2: |*  *******                       |**  ******      *
|
;*       3: |*  *******      *                |**  ******
|
;*          +-------------------------------------------------------------+
;*
;*      Done
;*
;*      Epilog not removed
;*      Collapsed epilog stages       : 0
;*
;*      Prolog not entirely removed
;*      Collapsed prolog stages       : 2
;*
;*      Minimum required memory pad   : 0 bytes
;*
;*      For further improvement on this loop, try option -mh96
;*
;*      Minimum safe trip count       : 3 (after unrolling)
;*
;*
;*      Mem bank conflicts/iter(est.) : { min 0.000, est 0.000, max 0.000
}
;*      Mem bank perf. penalty (est.) : 0.0%
;*
;*
;*      Total cycles (est.)         : 10 + min_trip_cnt * 4 = 1034
;*-----------------------------------------------------------------------*
;*       SETUP CODE
;*
;*                   SUB            B0,1,B0
;*
;*        SINGLE SCHEDULED ITERATION
;*
;*        $C$C26:
;*   0               LDDW    .D1T1   *A9++(32),A5:A4   ; |53|
;*   1               LDDW    .D1T2   *-A9(16),B5:B4    ; |53|
;*   2               LDDW    .D1T1   *-A9(8),A5:A4     ; |53|
;*   3               LDDW    .D1T2   *-A9(24),B5:B4    ; |53|
;*   4               NOP             1
;*   5               DOTP2   .M1     A5,A5,A16         ; |55|
;*   6               DOTP2   .M1     A4,A4,A5          ; |55|
;*     ||            DOTP2   .M2     B5,B5,B16         ; |55|
```

```
;*   7              DOTP2   .M2     B4,B4,B5          ; |55|
;*     ||           DOTP2   .M1     A4,A4,A16         ; |55|
;*   8              DOTP2   .M2     B4,B4,B16         ; |55|
;*     ||           DOTP2   .M1     A5,A5,A4          ; |55|
;*     ||   [ B0]   BDEC    .S2     $C$C26,B0         ; |52|
;*   9              ADD     .S1     A16,A3,A3         ; |55|
;*     ||           DOTP2   .M2     B5,B5,B4          ; |55|
;*  10              ADD     .L1     A5,A6,A6          ; |55|
;*     ||           ADD     .L2     B16,B8,B8         ; |55|
;*  11              ADD     .L2     B5,B9,B9          ; |55|
;*     ||           ADD     .L1     A16,A8,A8         ; |55|
;*  12              ADD     .L2     B16,B7,B7         ; |55|
;*     ||           ADD     .L1     A4,A7,A7          ; |55|
;*  13              ADD     .S2     B4,B6,B6          ; |55|
;*  14              ; BRANCHCC OCCURS {$C$C26}        ; |52|
;*----------------------------------------------------------------------*
$C$L1:    ; PIPED LOOP PROLOG
;          EXCLUSIVE CPU CYCLES: 2

           MVK     .L2     0x1,B1            ; init prolog collapse
predicate
||         ZERO    .D2     B8                ; |52|
||         ZERO    .L1     A6                ; |52|
||         ZERO    .S1     A8                ; |52|
||         LDDW    .D1T1   *A9++(32),A5:A4   ; |53| (P) <0,0>
|| [ B0]   BDEC    .S2     $C$L2,B0          ; |52| (P) <0,8>

           ZERO    .L2     B9                ; |52|
||         ZERO    .S2     B7                ; |52|
||         ZERO    .L1     A3                ; |52|
||         ZERO    .D2     B6                ; |52|
||         ZERO    .S1     A7                ; |52|
||         LDDW    .D1T2   *-A9(16),B5:B4    ; |53| (P) <0,1>

;** ----------------------------------------------------------------------*
$C$L2:    ; PIPED LOOP KERNEL
$C$DW$L$_cov1$4$B:
;          EXCLUSIVE CPU CYCLES: 4

   [!A0]   ADD     .L1     A5,A6,A6          ; |55| <0,10>
|| [!A0]   ADD     .L2     B16,B8,B8         ; |55| <0,10>
||         DOTP2   .M1     A4,A4,A5          ; |55| <1,6>
||         DOTP2   .M2     B5,B5,B16         ; |55| <1,6>
||         LDDW    .D1T1   *-A9(8),A5:A4     ; |53| <2,2>

   [!A0]   ADD     .L2     B5,B9,B9          ; |55| <0,11>
|| [!A0]   ADD     .L1     A16,A8,A8         ; |55| <0,11>
||         DOTP2   .M2     B4,B4,B5          ; |55| <1,7>
||         DOTP2   .M1     A4,A4,A16         ; |55| <1,7>
||         LDDW    .D1T2   *-A9(24),B5:B4    ; |53| <2,3>

   [!A0]   ADD     .L2     B16,B7,B7         ; |55| <0,12>
|| [!A0]   ADD     .L1     A4,A7,A7          ; |55| <0,12>
|| [ B0]   BDEC    .S2     $C$L2,B0          ; |52| <1,8>
||         DOTP2   .M2     B4,B4,B16         ; |55| <1,8>
||         DOTP2   .M1     A5,A5,A4          ; |55| <1,8>
||         LDDW    .D1T1   *A9++(32),A5:A4   ; |53| <3,0>

   [ B1]   SUB     .L2     B1,1,B1           ; <0,13>
```

```
|| [ A0]   SUB     .L1     A0,1,A0           ; <0,13>
|| [!A0]   ADD     .S2     B4,B6,B6          ; |55| <0,13>
|| [!B1]   ADD     .S1     A16,A3,A3         ; |55| <1,9>
||         DOTP2   .M2     B5,B5,B4          ; |55| <1,9>
||         DOTP2   .M1     A5,A5,A16         ; |55| <2,5>
||         LDDW    .D1T2   *-A9(16),B5:B4    ; |53| <3,1>

$C$DW$L$_cov1$4$E:
;** --------------------------------------------------------------------*
$C$L3:    ; PIPED LOOP EPILOG
;          EXCLUSIVE CPU CYCLES: 9

           ADD     .L1     A5,A6,A6          ; |55| (E) <1,10>
||         ADD     .L2     B16,B8,B8         ; |55| (E) <1,10>
||         DOTP2   .M1     A4,A4,A5          ; |55| (E) <2,6>
||         DOTP2   .M2     B5,B5,B16         ; |55| (E) <2,6>
||         LDDW    .D1T1   *-A9(8),A5:A4     ; |53| (E) <3,2>

           ADD     .L2     B5,B9,B9          ; |55| (E) <1,11>
||         ADD     .L1     A16,A8,A8         ; |55| (E) <1,11>
||         DOTP2   .M2     B4,B4,B5          ; |55| (E) <2,7>
||         DOTP2   .M1     A4,A4,A16         ; |55| (E) <2,7>
||         LDDW    .D1T2   *-A9(24),B5:B4    ; |53| (E) <3,3>

           ADD     .L2     B16,B7,B7         ; |55| (E) <1,12>
||         ADD     .L1     A4,A7,A7          ; |55| (E) <1,12>
||         DOTP2   .M2     B4,B4,B16         ; |55| (E) <2,8>
||         DOTP2   .M1     A5,A5,A4          ; |55| (E) <2,8>

           ADD     .L2     B4,B6,B6          ; |55| (E) <1,13>
||         ADD     .L1     A16,A3,A3         ; |55| (E) <2,9>
||         DOTP2   .M2     B5,B5,B4          ; |55| (E) <2,9>
||         DOTP2   .M1     A5,A5,A16         ; |55| (E) <3,5>

           ADD     .L1     A5,A6,A6          ; |55| (E) <2,10>
||         ADD     .L2     B16,B8,B8         ; |55| (E) <2,10>
||         DOTP2   .M1     A4,A4,A5          ; |55| (E) <3,6>
||         DOTP2   .M2     B5,B5,B16         ; |55| (E) <3,6>

           ADD     .L2     B5,B9,B9          ; |55| (E) <2,11>
||         ADD     .L1     A16,A8,A8         ; |55| (E) <2,11>
||         DOTP2   .M2     B4,B4,B5          ; |55| (E) <3,7>
||         DOTP2   .M1     A4,A4,A16         ; |55| (E) <3,7>

           ADD     .L2     B16,B7,B7         ; |55| (E) <2,12>
||         ADD     .L1     A4,A7,A7          ; |55| (E) <2,12>
||         DOTP2   .M2     B4,B4,B16         ; |55| (E) <3,8>
||         DOTP2   .M1     A5,A5,A4          ; |55| (E) <3,8>

           ADD     .L2     B4,B6,B6          ; |55| (E) <2,13>
||         ADD     .L1     A16,A3,A3         ; |55| (E) <3,9>
||         DOTP2   .M2     B5,B5,B4          ; |55| (E) <3,9>

           MV      .L1     A3,A10
||         ADD     .S1     A5,A6,A6          ; |55| (E) <3,10>
||         ADD     .L2     B16,B8,B8         ; |55| (E) <3,10>

;** --------------------------------------------------------------------*
;          EXCLUSIVE CPU CYCLES: 9
```

```
           ADD     .L2     B5,B9,B9          ; |55| (E) <3,11>

           MV      .L1     A6,A11
||         MV      .L2     B8,B11

           ADD     .L2     B4,B6,B10         ; |55|
||         MV      .L1X    B9,A12
||         ADD     .D2     B16,B7,B12        ; |55|
||         ADD     .S1     A4,A7,A13         ; |55|
||         MVC     .S2     B17,CSR           ; interrupts on
||         ADD     .D1     A16,A8,A15        ; |55|
```

2. Iteración tamaño variable

```
;** --------------------------------------------------------------------*
;          EXCLUSIVE CPU CYCLES: 9
           MV      .L1     A4,A12            ; |51|
      .dwpsn      file "../clk.c",line 50,column 2,is_stmt
           ZERO    .L1     A10               ; |50|
      .dwpsn      file "../clk.c",line 52,column 11,is_stmt
           CMPGT   .L1     A11,0,A0          ; |52|
   [!A0]   BNOP    .S1     $C$L6,5           ; |52|
           ; BRANCHCC OCCURS {$C$L6}         ; |52|
;** --------------------------------------------------------------------*
;          EXCLUSIVE CPU CYCLES: 7
      .dwpsn      file "../clk.c",line 53,column 6,is_stmt

           CMPGT   .L2X    A11,6,B1
||         MVKL    .S1     _buf,A5

   [ B1]   BNOP    .S2     $C$L2,5
||         MVKH    .S1     _buf,A5
||         MV      .L2X    A11,B0            ; |53|

           ; BRANCHCC OCCURS {$C$L2} {0}
;** --------------------------------------------------------------------*
;**   BEGIN LOOP $C$L1
;** --------------------------------------------------------------------*
$C$L1:
$C$DW$L$_cov1$4$B:
;          EXCLUSIVE CPU CYCLES: 5
      .dwpsn      file "../clk.c",line 55,column 6,is_stmt
           LDH     .D1T1   *A5++,A3          ; |55|
           NOP             4
$C$DW$L$_cov1$4$E:
;** --------------------------------------------------------------------*
$C$DW$L$_cov1$5$B:
;          EXCLUSIVE CPU CYCLES: 10
           MPY     .M1     A3,A3,A3          ; |55|
           NOP             1
           ADD     .L1     A3,A10,A10        ; |55|
      .dwpsn      file "../clk.c",line 52,column 11,is_stmt
           SUB     .L2     B0,1,B0           ; |52|
   [ B0]   B       .S1     $C$L1             ; |52|
   [!B0]   BNOP    .S1     $C$L6,4
           ; BRANCHCC OCCURS {$C$L1}         ; |52|
$C$DW$L$_cov1$5$E:
;** --------------------------------------------------------------------*
;          EXCLUSIVE CPU CYCLES: 1
           NOP             1
           ; BRANCH OCCURS {$C$L6}
;** --------------------------------------------------------------------*
$C$L2:
;          EXCLUSIVE CPU CYCLES: 3

           MVC     .S2     CSR,B5
||         SUB     .L2     B0,6,B0

           AND     .L2     -2,B5,B4
```

```
           MVC     .S2     B4,CSR            ; interrupts off
;*----------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop source line                 : 52
;*      Loop opening brace source line   : 52
;*      Loop closing brace source line   : 57
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 0
;*      Unpartitioned Resource Bound     : 1
;*      Partitioned Resource Bound(*)    : 1
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     0        0
;*      .S units                     0        1*
;*      .D units                     1*       0
;*      .M units                     1*       0
;*      .X cross paths               0        0
;*      .T address paths             1*       0
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          1        0     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             0        1*
;*      Bound(.L .S .D .LS .LSD)     1*       1*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 1  Schedule found with 8 iterations in parallel
;*
;*      Register Usage Table:
;*          +-----------------------------------------------------------+
;*
|AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA|BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB|
;*
|00000000001111111111222222222233|00000000001111111111222222222233|
;*
|01234567890123456789012345678901|01234567890123456789012345678901|
;*           |--------------------------------+------------------------|
;*       0: |*  ****                          |*
|
;*           +-----------------------------------------------------------+
;*
;*      Done
;*
;*      Redundant loop generated
;*      Epilog not removed
;*      Collapsed epilog stages       : 0
;*
;*      Prolog not entirely removed
;*      Collapsed prolog stages       : 2
;*
;*      Minimum required memory pad   : 0 bytes
;*
;*      For further improvement on this loop, try option -mh14
;*
;*      Minimum safe trip count       : 7
;*
;*
```

```
;*      Mem bank conflicts/iter(est.) : { min 0.000, est 0.000, max 0.000
}
;*      Mem bank perf. penalty (est.) : 0.0%
;*
;*
;*      Total cycles (est.)         : 7 + trip_cnt * 1
;*-------------------------------------------------------------------------*
;*       SETUP CODE
;*
;*                  SUB             B0,1,B0
;*
;*        SINGLE SCHEDULED ITERATION
;*
;*        $C$C25:
;*   0              LDH     .D1T1   *A5++,A4          ; |55|
;*   1              NOP             1
;*   2      [ B0]   BDEC    .S2     $C$C25,B0         ; |52|
;*   3              NOP             2
;*   5              MPY     .M1     A4,A4,A3          ; |55|
;*   6              NOP             1
;*   7              ADD     .S1     A3,A6,A6          ; |55|
;*   8              ; BRANCHCC OCCURS {$C$C25}        ; |52|
;*-------------------------------------------------------------------------*
$C$L3:    ; PIPED LOOP PROLOG
;          EXCLUSIVE CPU CYCLES: 5

           LDH     .D1T1   *A5++,A4          ; |55| (P) <0,0>
|| [ B0]   BDEC    .S2     $C$L4,B0          ; |52| (P) <0,2>

           LDH     .D1T1   *A5++,A4          ; |55| (P) <1,0>
|| [ B0]   BDEC    .S2     $C$L4,B0          ; |52| (P) <1,2>

           LDH     .D1T1   *A5++,A4          ; |55| (P) <2,0>
|| [ B0]   BDEC    .S2     $C$L4,B0          ; |52| (P) <2,2>

           LDH     .D1T1   *A5++,A4          ; |55| (P) <3,0>
|| [ B0]   BDEC    .S2     $C$L4,B0          ; |52| (P) <3,2>

           MV      .L1     A10,A6
||         MVK     .S1     0x2,A0            ; init prolog collapse
predicate
||         LDH     .D1T1   *A5++,A4          ; |55| (P) <4,0>
|| [ B0]   BDEC    .S2     $C$L4,B0          ; |52| (P) <4,2>

;** -------------------------------------------------------------------------*
$C$L4:    ; PIPED LOOP KERNEL
$C$DW$L$_cov1$9$B:
;          EXCLUSIVE CPU CYCLES: 1

   [ A0]   SUB     .L1     A0,1,A0           ; <0,7>
|| [!A0]   ADD     .S1     A3,A6,A6          ; |55| <0,7>
||         MPY     .M1     A4,A4,A3          ; |55| <2,5>
|| [ B0]   BDEC    .S2     $C$L4,B0          ; |52| <5,2>
||         LDH     .D1T1   *A5++,A4          ; |55| <7,0>

$C$DW$L$_cov1$9$E:
;** -------------------------------------------------------------------------*
$C$L5:    ; PIPED LOOP EPILOG
;          EXCLUSIVE CPU CYCLES: 6
```

```
           ADD     .L1     A3,A6,A5          ; |55| (E) <1,7>
||         MPY     .M1     A4,A4,A5          ; |55| (E) <3,5>

           ADD     .L1     A3,A5,A3          ; |55| (E) <2,7>
||         MPY     .M1     A4,A4,A5          ; |55| (E) <4,5>

           ADD     .L1     A5,A3,A3          ; |55| (E) <3,7>
||         MPY     .M1     A4,A4,A5          ; |55| (E) <5,5>

           ADD     .L1     A5,A3,A3          ; |55| (E) <4,7>
||         MPY     .M1     A4,A4,A4          ; |55| (E) <6,5>

           ADD     .L1     A5,A3,A3          ; |55| (E) <5,7>
||         MPY     .M1     A4,A4,A4          ; |55| (E) <7,5>

           MVC     .S2     B5,CSR            ; interrupts on
||         ADD     .L1     A4,A3,A3          ; |55| (E) <6,7>

;** --------------------------------------------------------------------*
;          EXCLUSIVE CPU CYCLES: 1
           ADD     .L1     A4,A3,A10         ; |55|
;** --------------------------------------------------------------------*
```

3. Iteración cálculo covarianza tamaño fijo

```
;** --------------------------------------------------------------------*
;           EXCLUSIVE CPU CYCLES: 6
           MV      .L1     A4,A14            ; |70|
       .dwpsn      file "../clk.c",line 71,column 11,is_stmt
           MVKL    .S2     _buf,B19
           MVC     .S2     CSR,B9
           MVKH    .S2     _buf,B19

           AND     .L2     -2,B9,B4
||         ZERO    .L1     A19               ; |71|
||         ZERO    .S1     A5                ; |71|
||         ZERO    .D1     A0
||         MVK     .S2     0x200,B0          ; |71|

           MVC     .S2     B4,CSR            ; interrupts off
||         MV      .L1X    B19,A7
||         SUB     .L2     B0,2,B0
||         ZERO    .S1     A4                ; |71|
||         ZERO    .D1     A6                ; |71|
||         ZERO    .D2     B7                ; |71|

;*--------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop source line                 : 71
;*      Loop opening brace source line   : 71
;*      Loop closing brace source line   : 74
;*      Loop Unroll Multiple             : 8x
;*      Known Minimum Trip Count         : 512
;*      Known Maximum Trip Count         : 512
;*      Known Max Trip Count Factor      : 512
;*      Loop Carried Dependency Bound(^) : 0
;*      Unpartitioned Resource Bound     : 5
;*      Partitioned Resource Bound(*)    : 5
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     0        0
;*      .S units                     5*       4
;*      .D units                     2        0
;*      .M units                     0        0
;*      .X cross paths               0        3
;*      .T address paths             2        0
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          5        4     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             3        2
;*      Bound(.L .S .D .LS .LSD)     4        3
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 5  Schedule found with 3 iterations in parallel
;*
;*      Register Usage Table:
;*          +----------------------------------------------------------+
```

```
;*
|AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA|BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB|
;*
|00000000001111111111222222222233|00000000001111111111222222222233|
;*
|01234567890123456789012345678901|01234567890123456789012345678901|
;*          |---------------------------------+---------------------------
-----|
;*       0: |*  *******      ****             |*    ****
|
;*       1: |*  ******       ****             |*   *****
|
;*       2: |*  ******        ***             |*   * ***
|
;*       3: |*  *****         ***             |*   * ***
|
;*       4: |*  *******        **             |*     ***
|
;*          +-----------------------------------------------------------+
;*
;*      Done
;*
;*      Epilog not removed
;*      Collapsed epilog stages       : 0
;*
;*      Prolog not entirely removed
;*      Collapsed prolog stages       : 1
;*
;*      Minimum required memory pad   : 0 bytes
;*
;*      For further improvement on this loop, try option -mh32
;*
;*      Minimum safe trip count       : 2 (after unrolling)
;*
;*
;*      Mem bank conflicts/iter(est.) : { min 0.000, est 0.000, max 0.000
}
;*      Mem bank perf. penalty (est.) : 0.0%
;*
;*
;*      Total cycles (est.)         : 6 + min_trip_cnt * 5 = 2566
;*-------------------------------------------------------------------------*
;*       SETUP CODE
;*
;*                  SUB             B0,1,B0
;*
;*        SINGLE SCHEDULED ITERATION
;*
;*        $C$C59:
;*   0              LDDW    .D1T1   *A7++(16),A9:A8   ; |72|
;*   1              LDDW    .D1T1   *-A7(8),A17:A16   ; |72|
;*   2              NOP             3
;*   5              MV      .L2X    A8,B5             ; |72| Define a
twin register
;*     ||           EXT     .S1     A9,16,16,A3       ; |72|
;*     ||   [ B0]   BDEC    .S2     $C$C59,B0         ; |71|
;*   6              ADD     .L1     A3,A19,A19        ; |72|
;*     ||           SHR     .S1     A9,16,A3          ; |72|
;*     ||           SHR     .S2X    A16,16,B4         ; |72|
```

```
;*   7              EXT     .S2     B5,16,16,B4       ; |72|
;*     ||           ADD     .L1     A3,A18,A18        ; |72|
;*     ||           EXT     .S1     A16,16,16,A3      ; |72|
;*     ||           ADD     .L2     B4,B8,B8          ; |72|
;*   8              ADD     .L2     B4,B7,B7          ; |72|
;*     ||           SHR     .S2X    A8,16,B4          ; |72|
;*     ||           ADD     .L1     A3,A6,A6          ; |72|
;*     ||           EXT     .S1     A17,16,16,A3      ; |72|
;*   9              ADD     .L2     B4,B6,B6          ; |72|
;*     ||           ADD     .L1     A3,A5,A5          ; |72|
;*     ||           SHR     .S1     A17,16,A3         ; |72|
;*  10              ADD     .L1     A3,A4,A4          ; |72|
;*  11              ; BRANCHCC OCCURS {$C$C59}        ; |71|
;*----------------------------------------------------------------------*
$C$L1:    ; PIPED LOOP PROLOG
;          EXCLUSIVE CPU CYCLES: 1

           ZERO    .L1     A18               ; |71|
||         ZERO    .L2     B6                ; |71|
||         ZERO    .D2     B8                ; |71|
||         MVKH    .S1     0x10000,A0        ; init prolog collapse
predicate
||         LDDW    .D1T1   *A7++(16),A9:A8   ; |72| (P) <0,0>
|| [ B0]   BDEC    .S2     $C$L2,B0          ; |71| (P) <0,5>

;** ----------------------------------------------------------------------*
$C$L2:    ; PIPED LOOP KERNEL
$C$DW$L$_cov2$4$B:
;          EXCLUSIVE CPU CYCLES: 5

   [!A0]   ADD     .L1     A3,A19,A19        ; |72| <0,6>
||         SHR     .S1     A9,16,A3          ; |72| <0,6>
||         SHR     .S2X    A16,16,B4         ; |72| <0,6>
||         LDDW    .D1T1   *-A7(8),A17:A16   ; |72| <1,1>

   [!A0]   ADD     .L1     A3,A18,A18        ; |72| <0,7>
||         EXT     .S1     A16,16,16,A3      ; |72| <0,7>
|| [!A0]   ADD     .L2     B4,B8,B8          ; |72| <0,7>
||         EXT     .S2     B5,16,16,B4       ; |72| <0,7>

           SHR     .S2X    A8,16,B4          ; |72| <0,8>
|| [!A0]   ADD     .L1     A3,A6,A6          ; |72| <0,8>
||         EXT     .S1     A17,16,16,A3      ; |72| <0,8>
|| [!A0]   ADD     .L2     B4,B7,B7          ; |72| <0,8>

   [ A0]   MPYSU   .M1     2,A0,A0           ; <0,9>
|| [!A0]   ADD     .L2     B4,B6,B6          ; |72| <0,9>
||         SHR     .S1     A17,16,A3         ; |72| <0,9>
|| [!A0]   ADD     .L1     A3,A5,A5          ; |72| <0,9>

   [!A0]   ADD     .L1     A3,A4,A4          ; |72| <0,10>
|| [ B0]   BDEC    .S2     $C$L2,B0          ; |71| <1,5>
||         EXT     .S1     A9,16,16,A3       ; |72| <1,5>
||         MV      .L2X    A8,B5             ; |72| <1,5> Define a twin
register
||         LDDW    .D1T1   *A7++(16),A9:A8   ; |72| <2,0>

$C$DW$L$_cov2$4$E:
;*----------------------------------------------------------------------*
```

```
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop source line                 : 80
;*      Loop opening brace source line   : 80
;*      Loop closing brace source line   : 90
;*      Loop Unroll Multiple             : 16x
;*      Known Minimum Trip Count         : 256
;*      Known Maximum Trip Count         : 256
;*      Known Max Trip Count Factor      : 256
;*      Loop Carried Dependency Bound(^) : 0
;*      Unpartitioned Resource Bound     : 10
;*      Partitioned Resource Bound(*)    : 10
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                    0        0
;*      .S units                    1        0
;*      .D units                   10*      10*
;*      .M units                    4        4
;*      .X cross paths              1        5
;*      .T address paths           10*      10*
;*      Long read paths             0        0
;*      Long write paths            0        0
;*      Logical  ops (.LS)          0        0     (.L or .S unit)
;*      Addition ops (.LSD)         9        9     (.L or .S or .D unit)
;*      Bound(.L .S .LS)            1        0
;*      Bound(.L .S .D .LS .LSD)    7        7
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 10 Schedule found with 3 iterations in parallel
;*
;*      Register Usage Table:
;*          +-------------------------------------------------------------+
;*
|AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA|BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB|
;*
|00000000001111111111222222222233|00000000001111111111222222222233|
;*
|01234567890123456789012345678901|01234567890123456789012345678901|
;*           |--------------------------------+------------------------|
;*       0:  |** *******      ***  *          |*    * ***      **** **
|
;*       1:  |** *******      ***  *          |*    *****      ****  *
|
;*       2:  |** *******      *** **          |*   ******      ***** *
|
;*       3:  |** *******      ******         |*   ******      ****  *
|
;*       4:  |** *******      **** ***        |*   *** **      ****  *
|
;*       5:  |** ** ****      **** ***        |*   ******      ***** *
|
;*       6:  |** *******      ****  **        |*   ******      ****  *
|
;*       7:  |** ***** *      ****  **        |*   **** *      ****  *
|
;*       8:  |** *  ****      ***  ***        |*   **** *      ****  *
|
;*       9:  |** *  ****      ***  ***        |*   *  ***      *******
|
```

```
;*           +---------------------------------------------------------+
;*
;*       Done
;*
;*       Epilog not removed
;*       Collapsed epilog stages       : 0
;*       Collapsed prolog stages       : 2
;*       Minimum required memory pad   : 0 bytes
;*
;*       For further improvement on this loop, try option -mh64
;*
;*       Minimum safe trip count       : 2 (after unrolling)
;*       Min. prof. trip count  (est.) : 3 (after unrolling)
;*
;*       Mem bank conflicts/iter(est.) : { min 0.000, est 0.625, max 4.000
}
;*       Mem bank perf. penalty (est.) : 5.9%
;*
;*       Effective ii                  : { min 10.00, est 10.63, max 14.00 }
;*
;*
;*       Total cycles (est.)           : 11 + min_trip_cnt * 10 = 2571
;*-------------------------------------------------------------------------*
;*        SETUP CODE
;*
;*                    SUB             A0,1,A0
;*                    MV              A3,B22
;*                    MV              A16,B19
;*                    ADD             4,B22,B22
;*
;*         SINGLE SCHEDULED ITERATION
;*
;*         $C$C27:
;*    0               LDDW    .D1T1   *A16++(32),A23:A22 ; |81|
;*    1               LDDW    .D1T2   *-A16(16),B7:B6    ; |81|
;*    2               LDDW    .D1T1   *-A16(8),A5:A4     ; |81|
;*    3               NOP             2
;*    5               LDDW    .D1T2   *-A16(24),B21:B20 ; |81|
;*    6               NOP             2
;*    8               SUB2    .L2X    B7,A9,B7           ; |81|
;*      ||            SUB2    .L1     A4,A9,A21          ; |81|
;*      ||            SUB2    .S1     A5,A9,A8           ; |81|
;*    9               SUB2    .L2X    B6,A9,B8           ; |81|
;*      ||            DOTP2   .M2     B7,B7,B20          ; |84|
;*      ||            DOTP2   .M1     A21,A21,A20        ; |84|
;*      ||            STW     .D1T1   A8,*-A16(4)        ; |83|
;*   10               SUB2    .L1     A22,A9,A5          ; |81|
;*      ||            SUB2    .S1     A23,A9,A4          ; |81|
;*      ||            SUB2    .S2X    B20,A9,B5          ; |81|
;*      ||            DOTP2   .M2     B8,B8,B7           ; |84|
;*      ||            DOTP2   .M1     A8,A8,A5           ; |84|
;*   11               STW     .D2T1   A5,*B19++(32)      ; |83|
;*      ||            DOTP2   .M1     A4,A4,A5           ; |84|
;*      ||            DOTP2   .M2     B5,B5,B6           ; |84|
;*      ||            SUB2    .L2X    B21,A9,B6          ; |81|
;*   12               DOTP2   .M1     A5,A5,A4           ; |84|
;*      ||            MV      .L2X    A4,B4              ; |81| Define a
twin register
;*      ||            STW     .D2T2   B5,*-B19(24)       ; |83|
```

```
;*     ||           DOTP2   .M2     B6,B6,B20         ; |84|
;*  13              STW     .D1T1   A5,*++A3(32)      ; |87|
;*     ||           MV      .L1X    B6,A19            ; |81| Define a
twin register
;*     ||           STW     .D2T2   B8,*-B19(16)      ; |83|
;*     ||           MVD     .M2     B7,B4             ; |81| Split a long
life
;*     ||           ADD     .L2     B20,B18,B18       ; |84|
;*  14              STW     .D2T2   B6,*-B19(20)      ; |83|
;*     ||           ADD     .L2     B7,B17,B17        ; |84|
;*     ||           STW     .D1T1   A21,*-A16(8)      ; |83|
;*     ||           ADD     .L1     A20,A6,A6         ; |84|
;*     ||           ADD     .S1     A5,A7,A7          ; |84|
;*  15              ADD     .L1     A5,A18,A18        ; |84|
;*     ||           STW     .D2T1   A4,*++B22(32)     ; |87|
;*     ||           ADD     .L2     B6,B9,B9          ; |84|
;*     ||   [ A0]   BDEC    .S1     $C$C27,A0         ; |80|
;*  16              ADD     .L1     A4,A17,A17        ; |84|
;*     ||           STW     .D2T2   B4,*-B19(28)      ; |83|
;*     ||           ADD     .L2     B20,B16,B16       ; |84|
;*     ||           STW     .D1T1   A21,*+A3(24)      ; |87|
;*  17              STW     .D2T2   B8,*+B22(12)      ; |87|
;*     ||           STW     .D1T1   A8,*+A3(28)       ; |87|
;*  18              STW     .D2T1   A19,*+B22(8)      ; |87|
;*     ||           STW     .D1T2   B4,*+A3(20)       ; |87|
;*  19              STW     .D2T2   B5,*+B22(4)       ; |87|
;*  20              STW     .D2T2   B4,*-B19(12)      ; |83|
;*  21              ; BRANCHCC OCCURS {$C$C27}        ; |80|
;*-----------------------------------------------------------------------*
$C$L3:    ; PIPED LOOP EPILOG AND PROLOG
;          EXCLUSIVE CPU CYCLES: 26

           ADD     .L1     A3,A19,A3         ; |72| (E) <1,6>
||         SHR     .S1     A9,16,A7          ; |72| (E) <1,6>
||         SHR     .S2X    A16,16,B4         ; |72| (E) <1,6>
||         LDDW    .D1T1   *-A7(8),A17:A16   ; |72| (E) <2,1>

           ADD     .L1     A7,A18,A7         ; |72| (E) <1,7>
||         EXT     .S1     A16,16,16,A9      ; |72| (E) <1,7>
||         ADD     .L2     B4,B8,B4          ; |72| (E) <1,7>
||         EXT     .S2     B5,16,16,B5       ; |72| (E) <1,7>

           SHR     .S2X    A8,16,B7          ; |72| (E) <1,8>
||         ADD     .L1     A9,A6,A6          ; |72| (E) <1,8>
||         EXT     .S1     A17,16,16,A8      ; |72| (E) <1,8>
||         ADD     .L2     B5,B7,B5          ; |72| (E) <1,8>

           ADD     .L2     B7,B6,B6          ; |72| (E) <1,9>
||         SHR     .S1     A17,16,A5         ; |72| (E) <1,9>
||         ADD     .L1     A8,A5,A18         ; |72| (E) <1,9>

           ADD     .L1     A5,A4,A20         ; |72| (E) <1,10>
||         SHR     .S2X    A8,16,B31         ; |72| (E) <2,8>
||         SHR     .S1     A9,16,A31         ; |72| (E) <2,6>

           MVC     .S2     B9,CSR            ; interrupts on
||         ADD     .L2     B31,B6,B6         ; |72| (E) <2,9>
||         MV      .D2X    A8,B7             ; |72| (E) <2,5> Define a
twin register
```

```
||          ADD     .L1     A31,A7,A4         ; |72| (E) <2,7>
||          EXT     .S1     A9,16,16,A5       ; |72| (E) <2,5>

            SHR     .S1     A17,16,A28        ; |72| (E) <2,9>
||          EXT     .S2     B7,16,16,B7       ; |72| (E) <2,7>
||          ADD     .L1     A5,A3,A19         ; |72| (E) <2,6>

            SHR     .S2X    A16,16,B5         ; |72| (E) <2,6>
||          EXT     .S1     A17,16,16,A29     ; |72| (E) <2,8>
||          ADD     .L2     B7,B5,B7          ; |72| (E) <2,8>
||          ADD     .L1     A28,A20,A3        ; |72| (E) <2,10>
||          ADD     .D1     A19,A4,A27

            EXT     .S1     A16,16,16,A30     ; |72| (E) <2,7>
||          ADD     .L2     B5,B4,B8          ; |72| (E) <2,7>
||          ADD     .L1     A29,A18,A5        ; |72| (E) <2,9>

            ADD     .L1     A30,A6,A4         ; |72| (E) <2,8>
||          ADD     .L2X    B6,A27,B4

            ADD     .L2     B7,B4,B4
            ADD     .L2X    A3,B4,B5
            ADD     .L2X    A5,B5,B4
            ADD     .L2     B8,B4,B4
            NOP             1
            ADD     .L1X    A4,B4,A3
            SHR     .S1     A3,11,A4
            SHRU    .S1     A4,20,A4
            ADD     .L1     A3,A4,A3
            SHR     .S1     A3,12,A3
            PACK2   .L1     A3,A3,A9
      .dwpsn      file "../clk.c",line 80,column 11,is_stmt

            ZERO    .L2     B17               ; |80|
||          MVK     .L1     0x1,A1            ; init prolog collapse
predicate
||          MVK     .D2     0x2,B0            ; init prolog collapse
predicate
||          ZERO    .D1     A7                ; |80|
||          MVK     .S1     0x100,A3          ; |80|
||          MVKL    .S2     _buf2-32,B22

            ZERO    .L1     A17               ; |80|
||          ZERO    .L2     B18               ; |80|
||          ZERO    .D2     B16               ; |80|
||          MV      .S1X    B19,A16
||          SUB     .D1     A3,1,A0
||          MVKH    .S2     _buf2-32,B22

            ZERO    .L1     A18               ; |80|
||          ZERO    .L2     B9                ; |80|
||          ZERO    .S1     A6                ; |80|
||          MVC     .S2     CSR,B23

            ADD     .L2     4,B22,B22
||          MV      .L1X    B22,A3
||          AND     .S2     -2,B23,B4

            MVC     .S2     B4,CSR            ; interrupts off
```

```
;** --------------------------------------------------------------------*
$C$L4:    ; PIPED LOOP KERNEL
$C$DW$L$_cov2$6$B:
;          EXCLUSIVE CPU CYCLES: 10

           SUB2    .L2X    B21,A9,B6          ; |81| <0,11>
|| [!B0]   STW     .D2T1   A5,*B19++(32)      ; |83| <0,11>
||         DOTP2   .M2     B5,B5,B6           ; |84| <0,11>
||         DOTP2   .M1     A4,A4,A5           ; |84| <0,11>
|| [!A1]   LDDW    .D1T2   *-A16(16),B7:B6    ; |81| <1,1>

           MV      .L2X    A4,B4              ; |81| <0,12> Define a twin
register
|| [!B0]   STW     .D2T2   B5,*-B19(24)       ; |83| <0,12>
||         DOTP2   .M1     A5,A5,A4           ; |84| <0,12>
||         DOTP2   .M2     B6,B6,B20          ; |84| <0,12>
|| [!A1]   LDDW    .D1T1   *-A16(8),A5:A4     ; |81| <1,2>

           MVD     .M2     B7,B4              ; |81| <0,13> Split a long
life
|| [!B0]   STW     .D1T1   A5,*++A3(32)       ; |87| <0,13>
|| [!B0]   ADD     .L2     B20,B18,B18        ; |84| <0,13>
|| [!B0]   STW     .D2T2   B8,*-B19(16)       ; |83| <0,13>
||         MV      .L1X    B6,A19             ; |81| <0,13> Define a twin
register

   [!B0]   STW     .D1T1   A21,*-A16(40)      ; |83| <0,14>
|| [!B0]   ADD     .L2     B7,B17,B17         ; |84| <0,14>
|| [!B0]   STW     .D2T2   B6,*-B19(20)       ; |83| <0,14>
|| [!B0]   ADD     .L1     A20,A6,A6          ; |84| <0,14>
|| [!B0]   ADD     .S1     A5,A7,A7           ; |84| <0,14>

   [!B0]   ADD     .L1     A5,A18,A18         ; |84| <0,15>
|| [ A0]   BDEC    .S1     $C$L4,A0           ; |80| <0,15>
|| [!B0]   ADD     .L2     B6,B9,B9           ; |84| <0,15>
|| [!B0]   STW     .D2T1   A4,*++B22(32)      ; |87| <0,15>
|| [!A1]   LDDW    .D1T2   *-A16(24),B21:B20  ; |81| <1,5>

   [!B0]   ADD     .L2     B20,B16,B16        ; |84| <0,16>
|| [!B0]   STW     .D2T2   B4,*-B19(28)       ; |83| <0,16>
|| [!B0]   STW     .D1T1   A21,*+A3(24)       ; |87| <0,16>
|| [!B0]   ADD     .L1     A4,A17,A17         ; |84| <0,16>

   [!B0]   STW     .D1T1   A8,*+A3(28)        ; |87| <0,17>
|| [!B0]   STW     .D2T2   B8,*+B22(12)       ; |87| <0,17>

   [!B0]   STW     .D1T2   B4,*+A3(20)        ; |87| <0,18>
|| [!B0]   STW     .D2T1   A19,*+B22(8)       ; |87| <0,18>
||         SUB2    .L2X    B7,A9,B7           ; |81| <1,8>
||         SUB2    .L1     A4,A9,A21          ; |81| <1,8>
||         SUB2    .S1     A5,A9,A8           ; |81| <1,8>

   [ A1]   SUB     .L1     A1,1,A1            ; <0,19>
|| [!B0]   STW     .D2T2   B5,*+B22(4)        ; |87| <0,19>
|| [!A1]   STW     .D1T1   A8,*-A16(4)        ; |83| <1,9>
||         SUB2    .L2X    B6,A9,B8           ; |81| <1,9>
||         DOTP2   .M2     B7,B7,B20          ; |84| <1,9>
||         DOTP2   .M1     A21,A21,A20        ; |84| <1,9>
```

```
   [ B0]   SUB     .L2     B0,1,B0           ; <0,20>
|| [!B0]   STW     .D2T2   B4,*-B19(12)      ; |83| <0,20>
||         SUB2    .L1     A22,A9,A5         ; |81| <1,10>
||         DOTP2   .M2     B8,B8,B7          ; |84| <1,10>
||         SUB2    .S2X    B20,A9,B5         ; |81| <1,10>
||         SUB2    .S1     A23,A9,A4         ; |81| <1,10>
||         DOTP2   .M1     A8,A8,A5          ; |84| <1,10>
||         LDDW    .D1T1   *A16++(32),A23:A22 ; |81| <2,0>

$C$DW$L$_cov2$6$E:
;** --------------------------------------------------------------------*
$C$L5:    ; PIPED LOOP EPILOG
;          EXCLUSIVE CPU CYCLES: 16

           SUB2    .L2X    B21,A9,B6         ; |81| (E) <1,11>
||         STW     .D2T1   A5,*B19++(32)     ; |83| (E) <1,11>
||         DOTP2   .M2     B5,B5,B6          ; |84| (E) <1,11>
||         DOTP2   .M1     A4,A4,A5          ; |84| (E) <1,11>
||         LDDW    .D1T2   *-A16(16),B7:B6   ; |81| (E) <2,1>

           MV      .L2X    A4,B4             ; |81| (E) <1,12> Define a
twin register
||         STW     .D2T2   B5,*-B19(24)      ; |83| (E) <1,12>
||         DOTP2   .M1     A5,A5,A4          ; |84| (E) <1,12>
||         DOTP2   .M2     B6,B6,B20         ; |84| (E) <1,12>
||         LDDW    .D1T1   *-A16(8),A5:A4    ; |81| (E) <2,2>

           MVD     .M2     B7,B4             ; |81| (E) <1,13> Split a
long life
||         STW     .D1T1   A5,*++A3(32)      ; |87| (E) <1,13>
||         ADD     .L2     B20,B18,B18       ; |84| (E) <1,13>
||         STW     .D2T2   B8,*-B19(16)      ; |83| (E) <1,13>
||         MV      .L1X    B6,A19            ; |81| (E) <1,13> Define a
twin register

           STW     .D1T1   A21,*-A16(40)     ; |83| (E) <1,14>
||         ADD     .L2     B7,B17,B17        ; |84| (E) <1,14>
||         STW     .D2T2   B6,*-B19(20)      ; |83| (E) <1,14>
||         ADD     .L1     A20,A6,A6         ; |84| (E) <1,14>
||         ADD     .S1     A5,A7,A7          ; |84| (E) <1,14>

           ADD     .L1     A5,A18,A18        ; |84| (E) <1,15>
||         ADD     .L2     B6,B9,B9          ; |84| (E) <1,15>
||         STW     .D2T1   A4,*++B22(32)     ; |87| (E) <1,15>
||         LDDW    .D1T2   *-A16(24),B21:B20 ; |81| (E) <2,5>

           ADD     .L2     B20,B16,B16       ; |84| (E) <1,16>
||         STW     .D2T2   B4,*-B19(28)      ; |83| (E) <1,16>
||         STW     .D1T1   A21,*+A3(24)      ; |87| (E) <1,16>
||         ADD     .L1     A4,A17,A17        ; |84| (E) <1,16>

           STW     .D1T1   A8,*+A3(28)       ; |87| (E) <1,17>
||         STW     .D2T2   B8,*+B22(12)      ; |87| (E) <1,17>

           STW     .D1T2   B4,*+A3(20)       ; |87| (E) <1,18>
||         STW     .D2T1   A19,*+B22(8)      ; |87| (E) <1,18>
||         SUB2    .L2X    B7,A9,B7          ; |81| (E) <2,8>
||         SUB2    .L1     A4,A9,A21         ; |81| (E) <2,8>
||         SUB2    .S1     A5,A9,A8          ; |81| (E) <2,8>
```

```
           STW     .D2T2   B5,*+B22(4)       ; |87| (E) <1,19>
||         STW     .D1T1   A8,*-A16(4)       ; |83| (E) <2,9>
||         SUB2    .L2X    B6,A9,B8          ; |81| (E) <2,9>
||         DOTP2   .M2     B7,B7,B20         ; |84| (E) <2,9>
||         DOTP2   .M1     A21,A21,A20       ; |84| (E) <2,9>

           STW     .D2T2   B4,*-B19(12)      ; |83| (E) <1,20>
||         SUB2    .L1     A22,A9,A5         ; |81| (E) <2,10>
||         DOTP2   .M2     B8,B8,B7          ; |84| (E) <2,10>
||         SUB2    .L2X    B20,A9,B5         ; |81| (E) <2,10>
||         SUB2    .S1     A23,A9,A4         ; |81| (E) <2,10>
||         DOTP2   .M1     A8,A8,A5          ; |84| (E) <2,10>

           SUB2    .L2X    B21,A9,B6         ; |81| (E) <2,11>
||         STW     .D2T1   A5,*B19++(32)     ; |83| (E) <2,11>
||         DOTP2   .M2     B5,B5,B6          ; |84| (E) <2,11>
||         DOTP2   .M1     A4,A4,A5          ; |84| (E) <2,11>

           MV      .L2X    A4,B4             ; |81| (E) <2,12> Define a
twin register
||         STW     .D2T2   B5,*-B19(24)      ; |83| (E) <2,12>
||         DOTP2   .M1     A5,A5,A4          ; |84| (E) <2,12>
||         DOTP2   .M2     B6,B6,B20         ; |84| (E) <2,12>

           MVD     .M2     B7,B4             ; |81| (E) <2,13> Split a
long life
||         STW     .D1T1   A5,*++A3(32)      ; |87| (E) <2,13>
||         ADD     .L2     B20,B18,B18       ; |84| (E) <2,13>
||         STW     .D2T2   B8,*-B19(16)      ; |83| (E) <2,13>
||         MV      .L1X    B6,A19            ; |81| (E) <2,13> Define a
twin register

           STW     .D1T1   A21,*-A16(8)      ; |83| (E) <2,14>
||         ADD     .L2     B7,B17,B17        ; |84| (E) <2,14>
||         STW     .D2T2   B6,*-B19(20)      ; |83| (E) <2,14>
||         ADD     .L1     A20,A6,A6         ; |84| (E) <2,14>
||         ADD     .S1     A5,A7,A7          ; |84| (E) <2,14>

           MV      .L1     A7,A13
||         MV      .S1     A6,A12
||         ADD     .D1     A5,A18,A18        ; |84| (E) <2,15>
||         ADD     .L2     B6,B9,B9          ; |84| (E) <2,15>
||         STW     .D2T1   A4,*++B22(32)     ; |87| (E) <2,15>

           MV      .L1X    B17,A10
||         MVC     .S2     B23,CSR           ; interrupts on
||         ADD     .L2     B20,B16,B16       ; |84| (E) <2,16>
||         STW     .D2T2   B4,*-B19(28)      ; |83| (E) <2,16>
||         STW     .D1T1   A21,*+A3(24)      ; |87| (E) <2,16>
||         ADD     .S1     A4,A17,A17        ; |84| (E) <2,16>

;** --------------------------------------------------------------------*
;          EXCLUSIVE CPU CYCLES: 10

           STW     .D1T1   A8,*+A3(28)       ; |87| (E) <2,17>
||         STW     .D2T2   B8,*+B22(12)      ; |87| (E) <2,17>

           STW     .D2T1   A19,*+B22(8)      ; |87| (E) <2,18>
```

```
||          STW     .D1T2   B4,*+A3(20)       ; |87| (E) <2,18>

            STW     .D2T2   B5,*+B22(4)       ; |87| (E) <2,19>
||          MV      .L2X    A18,B11
||          MV      .S2     B16,B13

            STW     .D2T2   B4,*-B19(12)      ; |83| (E) <2,20>
||          MV      .L2X    A17,B10
||          MV      .L1X    B18,A11
||          MV      .S2     B9,B12
```

4. Iteración cálculo covarianza tamaño variable

```
;** --------------------------------------------------------------------*
;          EXCLUSIVE CPU CYCLES: 7
      .dwpsn      file "../clk.c",line 72,column 4,is_stmt

           CMPGT   .L2     B10,4,B1
||         MVKL    .S1     _buf,A5

   [ B1]   B       .S2     $C$L2
||         MVKH    .S1     _buf,A5

   [!B1]   LDH     .D1T1   *A5++,A3          ; |72|
           MV      .L2     B10,B0            ; |72|
   [ B1]   SUB     .L2     B0,5,B0
           NOP             2
           ; BRANCHCC OCCURS {$C$L2} {0}
;** --------------------------------------------------------------------*
;**   BEGIN LOOP $C$L1
;** --------------------------------------------------------------------*
$C$L1:
$C$DW$L$_cov2$4$B:
;          EXCLUSIVE CPU CYCLES: 8
           ADD     .L1     A3,A4,A4          ; |72|
      .dwpsn      file "../clk.c",line 71,column 11,is_stmt
           SUB     .L2     B0,1,B0           ; |71|

   [ B0]   B       .S1     $C$L1             ; |71|
|| [ B0]   LDH     .D1T1   *A5++,A3          ; |72|

   [!B0]   BNOP    .S1     $C$L5,4
           ; BRANCHCC OCCURS {$C$L1}         ; |71|
$C$DW$L$_cov2$4$E:
;** --------------------------------------------------------------------*
;          EXCLUSIVE CPU CYCLES: 1
           NOP             1
           ; BRANCH OCCURS {$C$L5}
;*--------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop source line                 : 71
;*      Loop opening brace source line   : 71
;*      Loop closing brace source line   : 74
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 0
;*      Unpartitioned Resource Bound     : 1
;*      Partitioned Resource Bound(*)    : 1
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     0        0
;*      .S units                     0        1*
;*      .D units                     1*       0
;*      .M units                     0        0
;*      .X cross paths               0        0
;*      .T address paths             1*       0
;*      Long read paths              0        0
```

```
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          1        0     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             0        1*
;*      Bound(.L .S .D .LS .LSD)     1*       1*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 1  Schedule found with 6 iterations in parallel
;*
;*      Register Usage Table:
;*          +-------------------------------------------------------+
;*
|AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA|BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB|
;*
|00000000001111111111222222222233|00000000001111111111222222222233|
;*
|01234567890123456789012345678901|01234567890123456789012345678901|
;*           |---------------------------------+-----------------------|
;*       0: |*  ***                           |*
|
;*          +-------------------------------------------------------+
;*
;*      Done
;*
;*      Redundant loop generated
;*      Epilog not removed
;*      Collapsed epilog stages       : 0
;*
;*      Prolog not entirely removed
;*      Collapsed prolog stages       : 1
;*
;*      Minimum required memory pad   : 0 bytes
;*
;*      For further improvement on this loop, try option -mh10
;*
;*      Minimum safe trip count       : 5
;*
;*
;*      Mem bank conflicts/iter(est.) : { min 0.000, est 0.000, max 0.000
}
;*      Mem bank perf. penalty (est.) : 0.0%
;*
;*
;*      Total cycles (est.)         : 5 + trip_cnt * 1
;*-----------------------------------------------------------------------*
;*       SETUP CODE
;*
;*                  SUB             B0,1,B0
;*
;*        SINGLE SCHEDULED ITERATION
;*
;*        $C$C79:
;*   0              LDH     .D1T1   *A5++,A3          ; |72|
;*     ||   [ B0]   BDEC    .S2     $C$C79,B0         ; |71|
;*   1              NOP             4
;*   5              ADD     .S1     A3,A4,A4          ; |72|
;*   6              ; BRANCHCC OCCURS {$C$C79}        ; |71|
;*-----------------------------------------------------------------------*
$C$L2:    ; PIPED LOOP PROLOG
```

```
;          EXCLUSIVE CPU CYCLES: 5
   [ B0]   BDEC    .S2     $C$L3,B0          ; |71| (P) <0,0>

           LDH     .D1T1   *A5++,A3          ; |72| (P) <0,0>
|| [ B0]   BDEC    .S2     $C$L3,B0          ; |71| (P) <1,0>

           LDH     .D1T1   *A5++,A3          ; |72| (P) <1,0>
|| [ B0]   BDEC    .S2     $C$L3,B0          ; |71| (P) <2,0>

           LDH     .D1T1   *A5++,A3          ; |72| (P) <2,0>
|| [ B0]   BDEC    .S2     $C$L3,B0          ; |71| (P) <3,0>

           MVK     .L1     0x1,A0            ; init prolog collapse
predicate
||         LDH     .D1T1   *A5++,A3          ; |72| (P) <3,0>
|| [ B0]   BDEC    .S2     $C$L3,B0          ; |71| (P) <4,0>

;** --------------------------------------------------------------------*
$C$L3:    ; PIPED LOOP KERNEL
$C$DW$L$_cov2$7$B:
;          EXCLUSIVE CPU CYCLES: 1

   [ A0]   SUB     .L1     A0,1,A0           ; <0,5>
|| [!A0]   ADD     .S1     A3,A4,A4          ; |72| <0,5>
|| [ B0]   BDEC    .S2     $C$L3,B0          ; |71| <5,0>
||         LDH     .D1T1   *A5++,A3          ; |72| <5,0>

$C$DW$L$_cov2$7$E:
;** --------------------------------------------------------------------*
$C$L4:    ; PIPED LOOP EPILOG
;          EXCLUSIVE CPU CYCLES: 4
           ADD     .L1     A3,A4,A4          ; |72| (E) <1,5>
           ADD     .L1     A3,A4,A4          ; |72| (E) <2,5>
           ADD     .L1     A3,A4,A4          ; |72| (E) <3,5>
           ADD     .L1     A3,A4,A4          ; |72| (E) <4,5>
;** --------------------------------------------------------------------*
;          EXCLUSIVE CPU CYCLES: 1
           ADD     .L1     A3,A4,A4          ; |72| (E) <5,5>
;** --------------------------------------------------------------------*
$C$L5:
;          EXCLUSIVE CPU CYCLES: 12
      .dwpsn      file "../clk.c",line 77,column 2,is_stmt
           SHR     .S1     A4,11,A3          ; |77|
           SHRU    .S1     A3,20,A3          ; |77|
           ADD     .L1     A4,A3,A3          ; |77|
           EXT     .S1     A3,4,16,A8        ; |77|
      .dwpsn      file "../clk.c",line 79,column 2,is_stmt
           ZERO    .L1     A10               ; |79|
      .dwpsn      file "../clk.c",line 80,column 11,is_stmt
           CMPGT   .L2     B10,0,B0          ; |80|
   [!B0]   BNOP    .S1     $C$L11,5          ; |80|
           ; BRANCHCC OCCURS {$C$L11}        ; |80|
;** --------------------------------------------------------------------*
;          EXCLUSIVE CPU CYCLES: 7
      .dwpsn      file "../clk.c",line 81,column 4,is_stmt

           CMPGT   .L2     B10,3,B1
||         MVKL    .S1     _buf,A6
||         MVKL    .S2     _buf2,B4
```

```
   [ B1]   BNOP    .S2     $C$L7,1
||         MVKH    .S1     _buf,A6

   [!B1]   LDH     .D1T1   *A6,A3           ; |81|
           MVKH    .S2     _buf2,B4
           MV      .L2     B10,B0           ; |81|
           MV      .L1X    B4,A9
           ; BRANCHCC OCCURS {$C$L7} {0}
;** --------------------------------------------------------------------*
;          EXCLUSIVE CPU CYCLES: 1
           NOP             1
;** --------------------------------------------------------------------*
;**   BEGIN LOOP $C$L6
;** --------------------------------------------------------------------*
$C$L6:
$C$DW$L$_cov2$13$B:
;          EXCLUSIVE CPU CYCLES: 14
           SUB     .L1     A3,A8,A3         ; |81|
           EXT     .S1     A3,16,16,A3      ; |81|
      .dwpsn      file "../clk.c",line 83,column 4,is_stmt
           STH     .D1T1   A3,*A6++         ; |83|
      .dwpsn      file "../clk.c",line 84,column 4,is_stmt
           MPY     .M1     A3,A3,A4         ; |84|
           NOP             1
           ADD     .L1     A4,A10,A10       ; |84|
      .dwpsn      file "../clk.c",line 87,column 4,is_stmt
           STH     .D1T1   A3,*A9++         ; |87|
      .dwpsn      file "../clk.c",line 80,column 11,is_stmt
           SUB     .L2     B0,1,B0          ; |80|

   [ B0]   B       .S1     $C$L6            ; |80|
|| [ B0]   LDH     .D1T1   *A6,A3           ; |81|

   [!B0]   BNOP    .S1     $C$L11,4
           ; BRANCHCC OCCURS {$C$L6}        ; |80|
$C$DW$L$_cov2$13$E:
;** --------------------------------------------------------------------*
;          EXCLUSIVE CPU CYCLES: 1
           NOP             1
           ; BRANCH OCCURS {$C$L11}
;** --------------------------------------------------------------------*
$C$L7:
;          EXCLUSIVE CPU CYCLES: 3

           MVC     .S2     CSR,B6
||         SUB     .L2     B0,3,B0

           AND     .L2     -2,B6,B4
           MVC     .S2     B4,CSR           ; interrupts off
;*--------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop source line                 : 80
;*      Loop opening brace source line   : 80
;*      Loop closing brace source line   : 90
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 0
```

```
;*      Unpartitioned Resource Bound     : 2
;*      Partitioned Resource Bound(*)    : 2
;*      Resource Partition:
;*                                 A-side   B-side
;*      .L units                     0        0
;*      .S units                     1        1
;*      .D units                     2*       1
;*      .M units                     1        0
;*      .X cross paths               0        1
;*      .T address paths             2*       1
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          2        1     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             1        1
;*      Bound(.L .S .D .LS .LSD)     2*       1
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 2  Schedule found with 5 iterations in parallel
;*
;*      Register Usage Table:
;*          +----------------------------------------------------------+
;*
|AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA|BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB|
;*
|00000000001111111111222222222233|00000000001111111111222222222233|
;*
|01234567890123456789012345678901|01234567890123456789012345678901|
;*           |--------------------------------+------------------------|
;*       0: |*  * *****                       |**   *
|
;*       1: |*  *******                       |**  **
|
;*           +----------------------------------------------------------+
;*
;*      Done
;*
;*      Redundant loop generated
;*      Epilog not removed
;*      Collapsed epilog stages       : 0
;*
;*      Prolog not entirely removed
;*      Collapsed prolog stages       : 2
;*
;*      Minimum required memory pad   : 0 bytes
;*
;*      For further improvement on this loop, try option -mh8
;*
;*      Minimum safe trip count       : 4
;*
;*
;*      Mem bank conflicts/iter(est.) : { min 0.000, est 0.125, max 1.000
}
;*      Mem bank perf. penalty (est.) : 5.9%
;*
;*      Effective ii                  : { min 2.00, est 2.13, max 3.00 }
;*
;*
;*      Total cycles (est.)           : 8 + trip_cnt * 2
```

```
;*----------------------------------------------------------------------*
;*       SETUP CODE
;*
;*                  SUB             B0,1,B0
;*                  MV              A6,B5
;*
;*        SINGLE SCHEDULED ITERATION
;*
;*        $C$C26:
;*   0              LDH     .D1T1   *A6++,A4          ; |81|
;*   1              NOP             3
;*   4      [ B0]   BDEC    .S2     $C$C26,B0         ; |80|
;*   5              SUB     .S1     A4,A8,A3          ; |81|
;*   6              EXT     .S1     A3,16,16,A5       ; |81|
;*   7              MPY     .M1     A5,A5,A3          ; |84|
;*     ||           STH     .D1T1   A5,*A9++          ; |87|
;*   8              MV      .L2X    A5,B4             ; |81| Define a
twin register
;*   9              STH     .D2T2   B4,*B5++          ; |83|
;*     ||           ADD     .L1     A3,A7,A7          ; |84|
;*  10              ; BRANCHCC OCCURS {$C$C26}        ; |80|
;*----------------------------------------------------------------------*
$C$L8:    ; PIPED LOOP PROLOG
;          EXCLUSIVE CPU CYCLES: 4

           MV      .L2X    A6,B5
||         LDH     .D1T1   *A6++,A4          ; |81| (P) <0,0>
|| [ B0]   BDEC    .S2     $C$L9,B0          ; |80| (P) <0,4>

           NOP             1

           ZERO    .L1     A3
||         LDH     .D1T1   *A6++,A4          ; |81| (P) <1,0>
|| [ B0]   BDEC    .S2     $C$L9,B0          ; |80| (P) <1,4>

           MV      .L1     A10,A7
||         MVK     .L2     0x1,B1            ; init prolog collapse
predicate
||         SET     .S1     A3,0xf,0xf,A0     ; init prolog collapse
predicate

;** --------------------------------------------------------------------*
$C$L9:    ; PIPED LOOP KERNEL
$C$DW$L$_cov2$17$B:
;          EXCLUSIVE CPU CYCLES: 2

   [ A0]   MPYSU   .M1     2,A0,A0           ; <0,8>
||         MV      .L2X    A5,B4             ; |81| <0,8> Define a twin
register
||         EXT     .S1     A3,16,16,A5       ; |81| <1,6>
|| [ B0]   BDEC    .S2     $C$L9,B0          ; |80| <2,4>
||         LDH     .D1T1   *A6++,A4          ; |81| <4,0>

   [ B1]   SUB     .L2     B1,1,B1           ; <0,9>
|| [!A0]   ADD     .L1     A3,A7,A7          ; |84| <0,9>
|| [!A0]   STH     .D2T2   B4,*B5++          ; |83| <0,9>
|| [!B1]   STH     .D1T1   A5,*A9++          ; |87| <1,7>
||         MPY     .M1     A5,A5,A3          ; |84| <1,7>
||         SUB     .S1     A4,A8,A3          ; |81| <2,5>
```

```
$C$DW$L$_cov2$17$E:
;** --------------------------------------------------------------------*
$C$L10:    ; PIPED LOOP EPILOG
;          EXCLUSIVE CPU CYCLES: 3

           MV      .L2X    A5,B4             ; |81| (E) <1,8> Define a
twin register
||         EXT     .S1     A3,16,16,A5       ; |81| (E) <2,6>

           ADD     .L1     A3,A7,A3          ; |84| (E) <1,9>
||         STH     .D2T2   B4,*B5++          ; |83| (E) <1,9>
||         STH     .D1T1   A5,*A9++          ; |87| (E) <2,7>
||         MPY     .M1     A5,A5,A5          ; |84| (E) <2,7>
||         SUB     .S1     A4,A8,A4          ; |81| (E) <3,5>

           MV      .L2X    A5,B4             ; |81| (E) <2,8> Define a
twin register
||         EXT     .S1     A4,16,16,A6       ; |81| (E) <3,6>

;** --------------------------------------------------------------------*;
EXCLUSIVE CPU CYCLES: 5

           MPY     .M1     A6,A6,A5          ; |84| (E) <3,7>
||         SUB     .L1     A4,A8,A4          ; |81| (E) <4,5>
||         ADD     .S1     A5,A3,A3          ; |84| (E) <2,9>

           EXT     .S1     A4,16,16,A4       ; |81| (E) <4,6>

           MPY     .M1     A4,A4,A5          ; |84| (E) <4,7>
||         ADD     .L1     A5,A3,A3          ; |84| (E) <3,9>
||         MV      .L2X    A6,B4             ; |81| (E) <3,8> Define a
twin register
||         STH     .D2T2   B4,*B5++          ; |83| (E) <2,9>
||         STH     .D1T1   A6,*A9++          ; |87| (E) <3,7>

           MV      .L2X    A4,B4             ; |81| (E) <4,8> Define a
twin register
||         STH     .D2T2   B4,*B5++          ; |83| (E) <3,9>
||         STH     .D1T1   A4,*A9++          ; |87| (E) <4,7>

           STH     .D2T2   B4,*B5++          ; |83| (E) <4,9>
||         ADD     .L1     A5,A3,A10         ; |84|
||         MVC     .S2     B6,CSR            ; interrupts on
```