ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD DE MÁLAGA

PROYECTO FIN DE CARRERA

TOMOGRAFÍA FDOCT BASADA EN DSP

INGENIERÍA DE TELECOMUNICACIÓN

MÁLAGA, 2007 JUAN M. GAGO BENÍTEZ

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

TOMOGRAFÍA FDOCT BASADA EN DSP

INGENIERÍA DE TELECOMUNICACIÓN

MÁLAGA, 2007 JUAN M. GAGO BENÍTEZ

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD DE MÁLAGA

Titulación: Ingeniería de Telecomunicación

Reunido el tribunal examinador en el día de la fecha, constituido por:

D./Dª. _____

D./Dª. _____

D./Dª. _____

para juzgar el Proyecto Fin de Carrera titulado:

TÍTULO DEL PFC

del alumno D./Dª.

dirigido por D./Dª.

ACORDÓ POR _____ OTORGAR LA

CALIFICACIÓN DE _____

Y, para que conste, se extiende firmada por los componentes del tribunal, la presente diligencia

Málaga, a _____ de _____ de _____

El Presidente El Vocal El Secretario

Fdo.: _____ Fdo.: _____ Fdo.: _____

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD DE MÁLAGA

TÍTULO DEL PFC

Tomografía FDOCT basada en DSP

REALIZADO POR:

Juan Manuel Gago Benítez

DIRIGIDO POR:

Fabián Arrebola Pérez

DEPARTAMENTO DE: **Tecnología Electrónica**

TITULACIÓN: **Ingeniería de Telecomunicación**

PALABRAS CLAVE: *FDOCT, FFT en tiempo real, procesadores de señal HammerheadSHARC, cámaras line-scan*

RESUMEN:

La tomografía FDOCT es una aplicación en tiempo real para la reconstrucción de imágenes biomédicas. La cámara line-scan empleada funciona a una alta velocidad de adquisición resultando un flujo de datos muy alto hacia el PC.

En este proyecto se diseña un sistema de detección donde cada línea captada por la cámara es directamente cargada en una tarjeta PCI equipada con varios procesadores DSP que procesan los datos en paralelo. Se programan dos algoritmos para comprobar tiempos de procesado y se analiza su idoneidad para la tomografía FDOCT.

Málaga, Octubre 2007

En memoria de Ana Jiménez Gago

Tomografía FDOCT basada en DSP

Índice

Prefacio xiii

PARTE I: Introducción 1

1. Presentación

1.1 Introducción a la tomografía FDOCT 2
1.2 Objetivos y Desarrollo 3
1.3 Motivación y Justificación 4
1.3.1 Modo standalone 5
1.4 Resumen 5

2. La tomografía FDOCT

2.1 Introducción 6
2.1 Tipos de OCT 7
2.2 Ventajas de la FDOCT 8
2.3 Usos clínicos 8
2.2 El Sistema FDOCT 9
2.3 El nuevo reto 11

3. Descripción del Hardware Disponible

3.1 Introducción 12
3.2 Plataforma de Desarrollo BittWare® HH-PCI 12
3.3 Procesador Hammerhead SHARC 15
3.4 Protocolo CameraLink® 16
3.4.1 Channel Link 17
3.4.2 Asignación de Puertos 17
3.5 Cámara Line-Scan Atmel-Aviiva M2CL 18
3.5.1 Características 18
3.5.2 Modo Free Run 19
3.5.3 Modo Trigger 19
3.6 Scanner de Adquisición de Imágenes 21
3.6.1 Señales de Control del Scanner 22
3.7 Resumen 23

4. Estado del Arte en el Procesado FFT

4.1 Introducción 24
4.2 La CFFT en Procesadores Intel® 24
4.2.1 Procesadores Pentium® y Xeon® 24
4.3 La CFFT en Procesadores Analog Devices® 25
4.3.1 Procesadores SHARC® 25
4.3.2 Procesadores TigerSHARC® 26
4.4 Sistemas Multiprocesadores Analog Devices® 26
4.4.1 Sistemas Multiprocesador HammerheadSHARC® 26
4.4.2 Sistemas Multiprocesador TigerSHARC® 27
4.5 TigerSHARC® vs. HammerheadSHARC® 28
4.5.1 Ventajas del Procesador HammerSHARC® 28
4.5.2 Ventajas del Procesador TigerSHARC® 28
4.5.3 ¿Qué sistema multiprocesador sería recomendable? 29
4.6 Resumen 29

5. Especificaciones y Análisis de Alternativas

5.1 Introducción 31
5.1.1 Parámetros de tiempo real 31
5.1.2 Consideración sobre el formato de los datos 31
5.2 Análisis de las alternativas 32
5.2.1 Procesado de Frames 32
5.2.2 Consideraciones sobre el procesado de frames 33
5.2.3 Procesado de Líneas 34
5.3 Lista de materiales y tareas 34
5.4 Resumen de las especificaciones software 35

6. Diseño Hardware

6.1 Introducción 36
6.2 Implementación del Line Grabber 37
6.2.1 Empaquetado de Píxeles en CameraLink 38
6.2.2 Componentes del Line-Grabber 39
6.3 Aproximación del Line Grabber 41
6.3.1 Generación de patrón fijo 41
6.3.2 Simulación de patrones variables 42
6.4 Sincronización con el scanner 44
6.5 Resumen de las especificaciones hardware 46

PARTE II: Aplicación *dspFDOCT* 47

7. Arquitecturas de Bus Disponibles

7.1 Introducción 48
7.1.1 Memoria Global y Cluster-Bus 48
7.1.2 Memoria Local y Enlaces Link-Port (DPLM) 49
7.1.3 Comparación entre ambas arquitecturas 49
7.2 Tomografía FDOCT 50
7.2.1 Arquitectura Cluster-Bus 52
7.2.2 Arquitectura DPLM 54
7.2.3 Conclusión 54
7.3 Resumen 55

8. Diseño Software (*dspFDOCT*)

8.1 Introducción 56
8.2 Simplificación 58
8.3 Versiones Preliminares 58
8.3.1 FFT compleja 59
8.3.2 FFT real 60
8.4 Versión Final 60
8.4.1 Código DSP 61
8.4.2 Conversión de Formatos de Datos 62
8.4.3 Tiempo de Procesado de Línea (SISD) 64
8.4.4 Tiempo de Procesado de Línea (SIMD) 64
8.5 Resumen 65

9. Resultados y Posibles Mejoras (*dspFDOCT*)

9.1 Introducción 66
9.2 Medidas del tiempo de procesado 66
9.1.1 Diferencia entre Predicción y Resultados 68
9.2 Nuevo Procesador 69
9.3 Nuevas Configuraciones 69
9.2.1 Expansión de la Arquitectura de Bus 70
9.2.2 Temporización del enlace Link Port HH1-HH3 (comunicación maestro-esclavo) 72
9.2.3 Pruebas de la librería Link-Port 73
9.4 Resumen 73

PARTE III: Aplicación *parallelFFT* 74

10. Implementación Paralela de FFTs
10.1 Introducción 75
10.2 Algoritmo 76
10.2.1 Standard FFT Radix-2 76
10.2.2 Matemáticas del Algoritmo 78
10.2.3 Implementación del Algoritmo 79
10.3 Programación de la Implementación en Sistema Multiprocesado 81
10.3.1 Código Matlab (FFT de 256 números complejos) 82
10.4 Resumen 84

11. Diseño Software (*parallelFFT*)
11.1 Introducción 85
11.2 Mapeado del algoritmo 86
11.2.1 Células de procesado 87
11.2.2 Comprobación en Matlab 88
11.3 Simplificación del Mapeado 89
11.3.1 Código DSP 90
11.4 Detalles del Mapeado 91
11.4.1 Pipelines 91
11.4.2 Sincronización de los Esclavos 93
11.4.3 Sincronización Maestro-Esclavo 94
11.5 Resumen 95

12. Resultados y Posibles Mejoras (*parallelFFT*)
12.1 Introducción 96
12.2 Estimación del tiempo fijo de procesado 96
12.3 Medida del tiempo variable de procesado 98
12.4 Análisis de los resultados 99
12.5 Nuevo procesador DSP 101
12.6 Resumen 101

PARTE IV: Análisis Final 102

13. Procedimientos de Prueba
13.1 Introducción 103
13.2 Depuración software para DSPs 104
13.3 Arquitectura software del PC 105
13.4 Proceso de depuración software 106
13.5 Resumen 107

14. Conclusiones
14.1 Resultados 108
14.2 Capacidades adquiridas 109
14.3 Líneas Futuras 110

Referencias 111
Tablas 113
Algoritmos 113
Figuras 114
Acrónimos 116
Glosario 117

APÉNDICES 118

A. Resumen del ADSP21160M

A.1 Procesador ADSP21160M A-1
A.1.1 Frecuencia de Reloj y Temporizador A-1
A.1.2 Controlador DMA A-3
A.1.3 Multiprocesado usando External Port A-4
A.1.4 Multiprocesado usando Link Ports A-5
A.1.5 Flags de E/S e Interrupciones A-6
A.1.6 Puertos Series A-7

B. Entorno de desarrollo VisualDSP++

B.1 Introducción B-1
B.2 Entorno Integrado de Desarrollo (IDE) B-2
B.3 Depurador B-4

C. Programa de diagnóstico interactivo Diag21k

C.1 Introducción C-1
C.2 Secuencia de comandos (script) C-1
C.2.1 Uso de constantes C-2
C.2.1 Uso de variable C-2
C.2.3 Las sentencias If y While C-3
C.2.4 Ejemplo de script: ***run_sys.cmd*** *C-3*
C.3 Depurador embebido C-5
C.3.1 Arranque del depurador C-5
C.3.2 Mostrando la posición del contador de programa C-6
C.3.3 Comandos Reset y Restart C-6
C.3.4 Comandos de Depuración C-6
C.4 Descripción de comandos C-8

D. Configurador de cámaras Intellicam

D.1 Introducción D-1
D.1.1 Matrox Intellicam D-1
D.2 Interfaz de Usuario D-2
D.3 Parámetros más importantes D-3
D.3.1 Configuración de Camera Link D-3
D.3.2 Configuración de la Cámara D-5
D.3.4 Tamaño del Frame D-6
D.3.5 Modo de Grabación D-7
D.3.6 Señal de Exposición D-8

E. Manual de usuario del CD-ROM

E.1 Introducción E-1
E.2 Guía de *dspFDOCT* E-4
E.2.1 Implementación de dspFDOCT en el PC (MS Visual C) E-4
E.2.2 Implementación de dspFDOCT en el PC (LabVIEW) E-6
E.2.3 Implementación de dspFDOCT en los DSPs (VisualDSP) E-9
E.2.4 Llamada a la librería FFTw E-10
E.3 Guía de *parallelFFT* E-11
E.3.1 Implementación de parallelFFT en el PC (MS Visual C) E-12
E.3.2 Implementación de parallelFFT en el PC (LabVIEW) E-13
E.3.3 Implementación de parallelFFT en los DSPs (VisualDSP) E-15
E.3.4 Código Matlab E-17
E.3.5 Programa CFFT_F 64 (MS Visual C) E-18

Prefacio

En el presente trabajo se diseñan dos aplicaciones basadas en el cómputo del algoritmo de la FFT en tiempo real. Se ha estructurado en cuatro partes correspondiendo las dos centrales a la implementación de cada aplicación sobre una tarjeta DSP disponible.

Parte I. Introducción
Parte II. Aplicación *dspFDOCT*
Parte III. Aplicación *parallelFFT*
Parte IV. Análisis Final

La primera parte sirve de introducción a los conceptos necesarios para el diseño de ambas aplicaciones. A continuación se describen brevemente los contenidos de los capítulos.

- **Capítulo 1. Presentación**
  En este capítulo se presenta los objetivos y el desarrollo del proyecto, así como su motivación y justificación.

- **Capítulo 2. La tomografía FDOCT**
  El capítulo 2 introduce los conceptos básicos del funcionamiento de esta técnica de imagen biomédica. Se plantea el nuevo reto que impone el uso de las nuevas cámaras CCD cada vez más rápidas.

- **Capítulo 3. Descripción del Hardware Disponible**
  En este capítulo se detalla el hardware disponible en el laboratorio: cámara line-scan y tarjeta DSP. Se hace un especial hincapié en las características más significativas del interfaz *CameraLink* que emplea la cámara line-scan.

- **Capítulo 4**. **Estado del Arte en el Procesado FFT**

  El capítulo 4 explica el estado del arte en lo referente al cómputo de algoritmo FFT complejo en procesadores DSP actuales. Se hace de forma general, sin aplicarlo a ninguna aplicación real en concreto. Un nuevo procesador se propone en este capítulo para mejorar la velocidad del procesado FFT en tiempo real.

- **Capítulo 5. Especificaciones y Análisis de Alternativas**

  En este capítulo se definen los valores mínimos de los parámetros de tiempo real más significativos de la tomografía FDOCT y se estudian las posibles alternativas disponibles para su desarrollo. Al final de este capítulo la solución más adecuada es elegida, en concreto se justifica el uso de la tarjeta DSP de Bittware como soporte hardware.

- **Capítulo 6. Diseño Hardware**

  En el capítulo 6 se diseñan todas las especificaciones hardware de la tarjeta DSP, tanto el conexionado externo como recursos internos a utilizar. El proceso de toma de decisión es complejo con factores referentes tanto al algoritmo como al hardware. Las decisiones que aquí se toma afectan al resultado final de ambas aplicaciones.

En la **segunda** parte del proyecto se implementa la aplicación en tiempo real llamada *dspFDOCT*. Su nombre lo debe a que va a ser usada junto con el sistema óptico empleado en la tomografía FDOCT.

- **Capítulo 7. Arquitecturas de Bus Disponibles**

  Para comenzar este capítulo describe cómo la arquitectura elegida del bus de la tarjeta DSP determina el rendimiento de la aplicación multiprocesador. Se muestran dos opciones y después de cuidadosas consideraciones la solución más conveniente para la tomografía FDOCT es tomada.

- **Capítulo 8. Diseño Software (*dspFDOCT*)**
  Este capítulo describe el diseño del software de la aplicación *dspFDOCT* en la arquitectura de bus elegida según los datos proporcionados por la cámara line-scan.

- **Capítulo 9. Resultados y Posibles Mejoras (*dspFDOCT*)**
  Este capítulo muestra los resultados obtenidos en el diseño del capítulo 8. Las limitaciones de la tarjeta Bittware son estudiadas y también se analiza las extensiones posibles de la arquitectura de bus elegida así como otras mejoras.

Hasta aquí se extiende la segunda parte del proyecto. En la **tercera** se implementa otro software en tiempo real para la misma tarjeta DSP al que se le denomina *parallelFFT*. Podrá ser usado en cualquier aplicación que requiera calcular la transformada de Fourier de manera continua, incluyendo la tomografía FDOCT con cierta conversión en los datos de entrada y salida. A continuación se describen los contenidos de los capítulos del tercer bloque.

- **Capítulo 10. Implementación Paralela de FFTs**
  El capítulo 10 da comienzo a la tercera parte explicando cómo se puede paralelizar el algoritmo FFT complejo en un sistema multiprocesador. Este método fue referido por primera vez en la conferencia IEE Irish Signals and Systems Conference (Dublin City University, Sept – 2005) y corresponde a la base del programa *parallelFFT* referido anteriormente.

- **Capítulo 11. Diseño Software (*parallelFFT*)**
  Para intentar mejorar la velocidad de procesado alcanzada en la aplicación *dspFDOCT*, el algoritmo del capítulo anterior es mapeado sobre la tarjeta DSP que disponemos.

- **Capítulo 12. Resultados y Posibles Mejoras (*parallelFFT*)**
  Este capítulo muestra los resultados obtenidos con esta nueva forma de computación del algoritmo FFT para el caso de 1024 muestras complejas.

A parte de estas tres secciones en las que fue dividido el proyecto, se incluyó un **cuarto** bloque de análisis final de ambas aplicaciones, compuesto por los siguientes capítulos:

- **Capítulo 13. Procedimientos de Prueba**
  Este capítulo especifica el procedimiento de pruebas llevado a cabo explicando también los tipos de depuración software empleados.

- **Capítulo 14. Conclusiones**
  Este capítulo habla en general de las conclusiones obtenidas con la tarjeta DSP disponible. También enumera las posibles líneas futuras que el presente proyecto deja abiertas.

Al final se incluye un conjunto de apéndices que amplían algunos conceptos y profundizar en algunas de las herramientas utilizadas en el procedimiento de pruebas.

- **Apéndice A**. **Resumen del ADSP21160M**
  Resume los recursos y funcionamiento del procesador DSP usado.

- **Apéndice B. Entorno de desarrollo VisualDSP++**
  Este apéndice da una breve descripción del entorno de desarrollo utilizado para implementar programas sobre los distintos procesadores DSP de la tarjeta.

- **Apéndice C**. **Programa de diagnóstico interactivo Diag21k**
  Se da una breve descripción del programa de diagnóstico de la tarjeta DSP de Bittware. Se detalla también un sencillo programa script diseñado para testear la aplicación *dspFDOCT*.

- **Apéndice D**. **Configurador de cámaras Intellicam**

  Este apéndice describe una herramienta para PC de configuración de cámaras con interfaz CameraLink. Obviamente dicha herramienta está ligada a un frame-grabber[1] al que la cámara será conectada.

- **Apéndice E**. **Manual de usuario del CD-ROM**

  Por último, en el apéndice se da una descripción de la estructura de archivos contenidos del CD-ROM que se adjunta. Los archivos que correspondan a programas mencionados a lo largo de este proyecto dispondrán del correspondiente pseudocódigo explicativo en el apéndice E.

[1] Un frame-grabber suele ser una tarjeta PCI o PMC con interfaz CameraLink que sirve de enlace entre la cámara CCD y el PC anfitrión del frame-grabber u otros dispositivos de control.

PARTE I:
Introducción

Capítulo 1
Presentación

1.1 Introducción a la tomografía FDOCT

La tomografía FDOCT es una aplicación en tiempo real para la obtención de imágenes biomédicas. Se basa en un tratamiento digital para la reconstrucción del perfil en profundidad de tejidos humanos.

La Figura 1-1 muestra el esquema del montaje básico para obtener este tipo de tomografía.

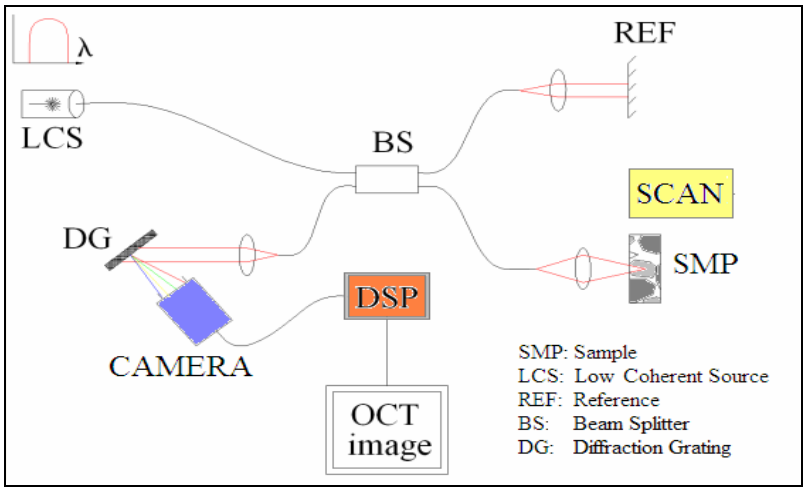


Figura 1-1 Configuración óptica FDOCT junto con detector y sistema de procesado

El ámbito del presente proyecto, **Tomografía FDOCT basada en DSP**, se localiza sólo en el tratamiento digital sin entrar en el análisis del sistema óptico, por ello únicamente se estudian los siguientes elementos que aparecen en la Figura 1-1:

- **Cámara line-scan**: funciona a una alta velocidad de adquisición resultando un flujo de datos muy alto hace el módulo DSP.

- **Scanner:** gracias a él se puede obtener una secuencia de imágenes 2D sin necesidad de mover la cámara de su emplazamiento (genera imágenes 1D). Necesita los mismos disparadores que la cámara (*triggers*) por sincronización.

- **Módulo DSP:** se trata de una tarjeta PCI equipada con varios procesadores DSP que realizan todo el procesado en tiempo real, constituido básicamente por el algoritmo de la FFT. En este proyecto se pretende diseñar un sistema de detección donde cada línea captada por la cámara sea directamente cargada en la tarjeta DSP.

1.2 Objetivos y Desarrollo

La temática principal del presente proyecto es el cómputo del algoritmo FFT en tiempo real y de manera continua. La forma en que se trata este estudio es a través de dos aplicaciones, una para uso práctico en la tomografía FDOCT y otra con un propósito más general:

- ***dspFDOCT*** procesado FFT adaptado a los datos de la tomografía FDOCT: líneas de 1024 píxeles de 10 bits a una frecuencia de 10,000 líneas por segundo.

- ***parallelFFT*** procesado FFT continuo y en tiempo real sin especificar ninguna aplicación práctica. Los datos de entrada serán considerados como líneas de 1024 números complejos en coma flotante.

A cada una de ambas aplicaciones le corresponden varios programas que se ejecutan en los procesadores de las siguientes plataformas:

- **Hammerhead-PCI de Bittware:** tarjeta PCI con cuatro procesadores SHARC de Analog Devices®.

- **DELL Workstation:** PC anfitrión de la tarjeta DSP con interfaz PCI y el procesador Xeon de Intel®.

1.3 Motivación y Justificación

La finalidad primordial por la cual se ha elaborado el presente trabajo es la presentación del mismo como Proyecto Fin de Carrera para la obtención del título de Ingeniero de Telecomunicación de la Universidad de Málaga aunque fue desarrollado en la Escuela Politécnica Federal de Lausana (Suiza).

La idea nació de la necesidad de aplicar la tarjeta DSP en la tomografía FDOCT lo antes posible. La premura se debía a que la tarjeta fue adquirida por el Laboratorio de Óptica Biomédica (http://lob.epfl.ch) en el año 2001 ocurriendo que en breve quedará obsoleta con la consiguiente pérdida de inversión económica así como también del servicio de mantenimiento por parte del distribuidor (www.etools.de).

La principal ventaja de la solución que plantea el presente proyecto es que gracias al procesado in-situ de una tarjeta DSP se estaría cargando el resultado del procesado en la memoria RAM del PC anfitrión donde ningún post-procesado sería ya necesario. Por tanto el microprocesador Xeon del PC quedaría libre para ejecutar otras aplicaciones.

Todo esto es en el caso de que la tarjeta DSP esté conectada a un ordenador ya que también puede trabajar en modo independiente con alimentación propia (éste modo de funcionamiento es referido como *standalone*).

1.3.1 Modo standalone

El modo de trabajo *standalone* es muy interesante a la hora de homologar el equipo para ser empleado en hospitales ya que hasta ahora para el tratamiento digital de la FDOCT siempre había sido necesario homologar todo un equipo informático (pantalla, teclado, ratón, etc.) junto con los generadores de los disparadores de la cámara CCD y scanner.

Todo esto supone un mayor gasto económico que si dispusiéramos de tan solo una única carcasa en la que se integrará la tarjeta DSP, un frame grabber y una tarjeta gráfica. A dicha carcasa se le conectaría por un lado una cámara CCD (frame-gabber) y por el otro una pantalla de interacción táctil (tarjeta gráfica), simplificándose mucho el proceso de homologación.

El objeto final de este trabajo es púes, convertir esta solución en una opción viable para su uso durante operaciones.

1.4 Resumen

El presente proyecto queda, por tanto, enmarcado dentro del cómputo en tiempo real de la FFT. Es motivado por la necesidad de la reconstrucción en tiempo real del perfil de tejidos en FDOCT.

En la primera parte del proyecto se empleará una tarjeta HH-PCI de BittWare con varios procesadores DSP para minimizar el tiempo de procesado y poder así utilizar las nuevas cámaras CCD que proporcionan líneas a un ritmo aún mayor.

En la segunda parte del proyecto se dejará a un lado la tomografía FDOCT para centrarnos en la FFT compleja de 1024 puntos y así ganar algo más de generalidad en los resultados. En concreto, se implementará una innovadora optimización del procesado FFT en la tarjeta multiprocesador de Bittware.

Capítulo 2
La tomografía FDOCT

2.1 Introducción

En este capítulo se describe el principio de funcionamiento de la tomografía FDOCT y sus posibilidades de tiempo real.

La tomografía OCT (Optical Coherence Tomography) es una modalidad no invasiva y sin contacto de imagen biomédica que permite la visualización de la sección transversal de tejidos tridimensionales. El potencial de la OCT reside en su alta sensibilidad y velocidad de adquisición.

La Figura 2-1 muestra el ámbito de aplicación de esta técnica en relación con otras modalidades de imagen biomédica que existen en la actualidad.

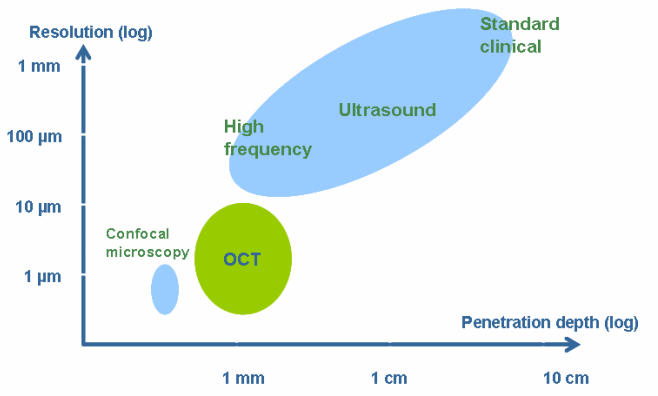


Figura 2-1 Ámbito de aplicación de la OCT

Como se observa con la OCT podemos obtener una resolución de hasta 10 µm a una profundidad de varios milímetros en distintos tipos de tejidos.

2.1.1 Tipos de OCT

El principio de la OCT es interferometría basada en una luz blanca de baja coherencia. El sistema óptico consiste normalmente en un interferómetro (Figura 2-2, típicamente de tipo Michelson) con coherencia baja.

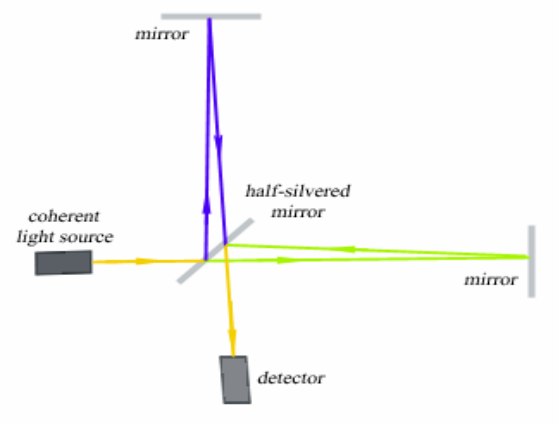


Figura 2-2 Interferómetro de Michelson
La luz es dividida y recombinada usando los espejos de referencia y muestra

En este interferómetro un espejo es la muestra y el otro la referencia. Dependiendo de que dicha referencia sea fija o móvil tendremos dos clases de OCT:

- **referencia móvil:** dominio del tiempo (TDOCT)
- **referencia fija:** dominio de Fourier (FDOCT)

La Figura 2-3 muestra ambas configuraciones.

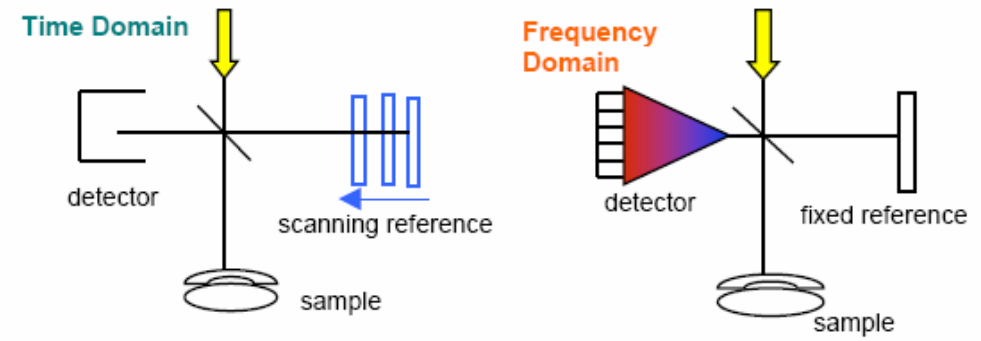


Figura 2-3 Configuraciones ópticas TDOCT y FDOCT.

2.1.2 Ventajas de la FDOCT

Las ventajas que presenta la tomografía FDOCT frente a la TDOCT son las siguientes:

- Eliminación del espejo de referencia para que el diseño del sistema total pueda ser: compacto, portátil, relativamente barato, y que se pueda utilizar fácilmente con cualquier tipo de scanners de adquisición de imágenes.

- La relación SNR es N veces mejor que en TDOCT donde N es el número de píxeles de la cámara line-scan, según es explicado por el profesor Fei Wang del Instituto de Tecnología de California (Caltech), curso EE131-2006.

- La dispersión se puede controlar fácilmente mientras que en otras modalidades de procesado de imagen esto es mucho más difícil.

2.1.3 Usos Clínicos

Este tipo de imagen biomédica se emplea en los siguientes campos:

- Oftalmología
  - Diagnosis de enfermedades de la retina y otras anomalías del ojo

- Dermatología
  - Detección temprana de cáncer de piel
  - Diagnosis de enfermedades de piel y otros problemas

- Cardiovascular
  - Imagen de los vasos sanguíneos y de otras áreas cardiovasculares para la detección temprana de enfermedades

2.2 El Sistema FDOCT

La Figura 2-4 muestra el sistema óptico FDOCT junto con el sistema de procesado.

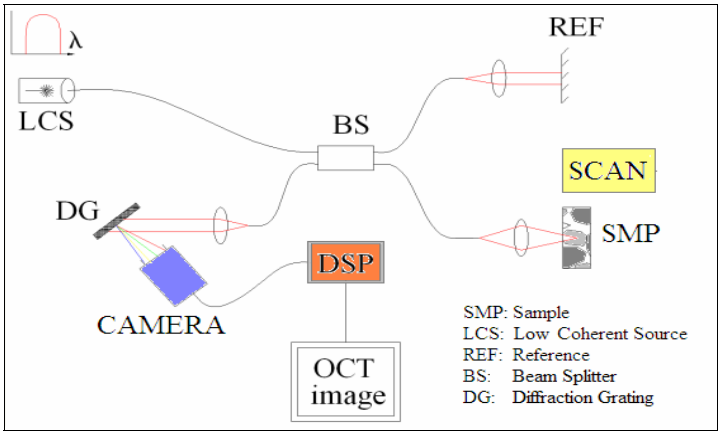


Figura 2-4 Configuración óptica FDOCT junto con detector y sistema de procesado

En FDOCT un espectrómetro proporciona un *interferograma espectral* que es grabado con una cámara CCD de tipo line-scan. El *interferograma espectral* (línea capturada por la cámara) no es más que la transformada de Fourier de la intensidad de la luz que se refleja sobre las distintas capas del tejido considerado. Por tanto la transformada inversa de este espectro permitiría reconstruir el perfil en la vertical del tejido. La Figura 2-5 muestra la imagen 2D que resulta del procesado.

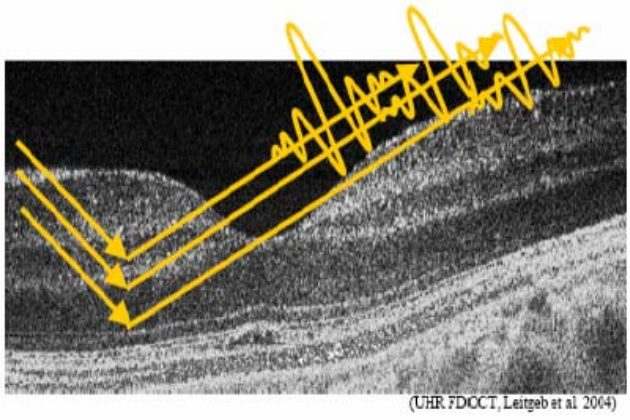


Figura 2-5 Tomografía (imagen 2D)

A pesar de realizar transformadas inversas de Fourier, podemos considerar en lo sucesivo la transformación directa debido a que son computacionalmente equivalentes.

Para obtener una imagen bidimensional del tejido como la Figura 2-5 sólo habría que colocar uno al lado del otro los sucesivos perfiles que va proporcionando el scanner de adquisición de imágenes.

La Figura 2-6 muestra el aspecto real del espectrómetro de la Figura 2-4.

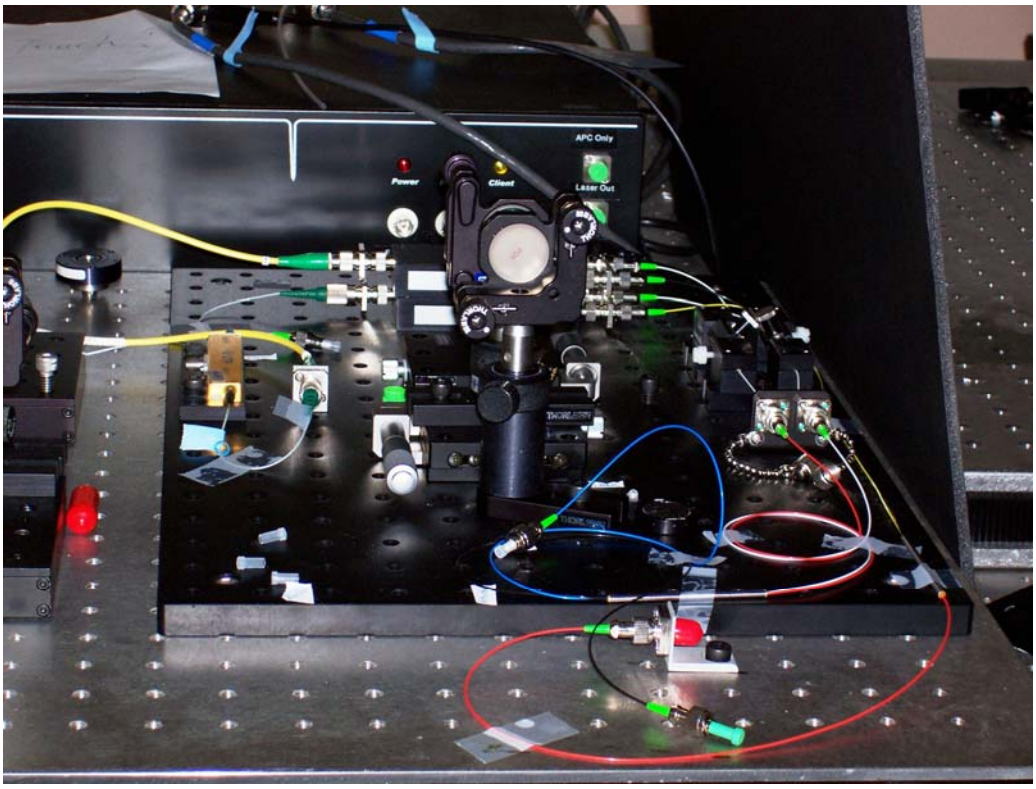


Figura 2-6 Configuración óptica de la FDOCT (fotografía)

2.3 El nuevo reto

Las nuevas cámaras line-scan, cada vez más rápidas, requieren velocidades de procesado de línea acorde a su velocidad de adquisición. La cámara empleada funciona a una velocidad de 20.000 espectros por segundo resultando un flujo de datos muy alto hacia el PC anfitrión.

El perfil de la profundidad del tejido es reconstruido a través de la transformada inversa de Fourier del interferograma espectral grabado. Éste último paso limita la eficiencia en tiempo real para la valoración de volúmenes grandes de tejido tridimensional.

Como fue observado por Touretzky en el congreso del IEEE-World sobre Inteligencia Computacional de 1994.

> *"...el problema real con las cámaras de vídeo es que el procesado de imagen es computacionalmente costoso. Incluso algo tan simple como calcular el flujo óptico en tiempo real requiere más potencia de procesado que la que se requiere para una robot móvil"*.

Sin embargo, la potencia de cálculo está continuamente siendo mejorada. Los procesadores DSP han sido diseñados, entre otras cosas, para realizar eficientemente análisis de Fourier, por tanto sería muy interesante cargar cada línea de la cámara directamente en una tarjeta PCI equipada con varios procesadores DSP que puedan procesar los datos en paralelo.

Capítulo 3
Descripción del Hardware Disponible

3.1 Introducción

Este capítulo describe todo el equipamiento electrónico disponible para realizar el presente proyecto. Se ha estructurado en tres partes:

- La primera da una visión global de la placa DSP utilizada, viéndose por encima la arquitectura del procesador ADSP21160 de Analog Devices®.

- La cámara AViiVA M2CL de Atmel® es caracterizada en la segunda parte del capítulo. Para ello se analizan los conceptos básicos del interfaz CameraLink.

- Por último se describe el funcionamiento del scanner de adquisición de imágenes.

3.2 Plataforma de Desarrollo BittWare® HH-PCI

La placa Hammerhead-PCI de BittWare tiene cuatro procesadores DSP, una SDRAM externa y varios tipos de comunicaciones, como se muestra en la Figura 3-1.

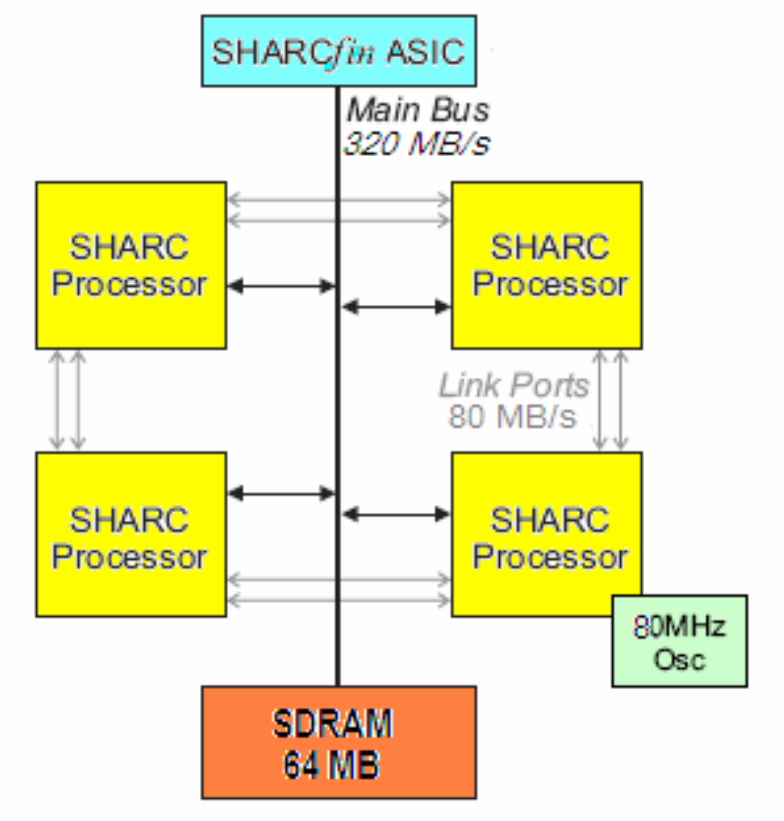


Figura 3-1 Estructura de la tarjeta HH-PCI.

Los SHARCs y la SDRAM están interconectados a través de un bus central, que es controlado por el ASIC SHARCfin. El SHARCfin controla también los accesos externos, por ejemplo el bus del PCI del PC anfitrión. Los procesadores SHARC están conectados en anillo a través de sus puertos de alta velocidad link-port, permitiendo la comunicación directa procesador-procesador

- La principal forma de comunicación externa es a través del bus PCI como interfaz hacia al PC anfitrión. También dispone de dos ranuras PMC (PCI Mezanine Card).

- Los procesadores SHARC están conectados en anillo bidireccional a través de sus puertos link ports (cuatro puertos por cada SHARC), que proporcionan una comunicación de alta velocidad inter-procesador. Los link-port son capaces de proporcionar hasta los 80 Mbytes/s.

- Un bus central que funciona a 40MHz es controlado por el ASIC denominado SHARC*fin*. La mayoría de la transferencia de datos ocurrirá a través de él.

- El SHARC*fin* mapea 64MB de SDRAM y los 2 x 2Mbit de la memoria de cada procesador DSP en el espacio de memoria del PCI. Esto permite acceder desde el ordenador a todas las posiciones de memoria de la tarjeta usando las funciones proporcionadas por el software de Bittware (capítulo 13).

El cuello de botella más probable dentro del sistema Hammerhead de BittWare es debido al SHARC*fin*. Este ASIC controla todos los accesos externos, PCI o PMC, a través del bus principal, el cual también es utilizado por los procesadores para tener acceso a la SDRAM (solamente un procesador puede leer o escribir cada vez).

En conclusión, aunque esta plataforma sería conveniente para la implementación final, está diseñada realmente como una plataforma del desarrollo lo que la hace doblemente atractiva.

3.3 Procesadores Hammerhead-SHARC

Un diagrama simplificado del procesador del SHARC (Super Harvard Architecture Computer) es mostrado en la Figura 3-2.

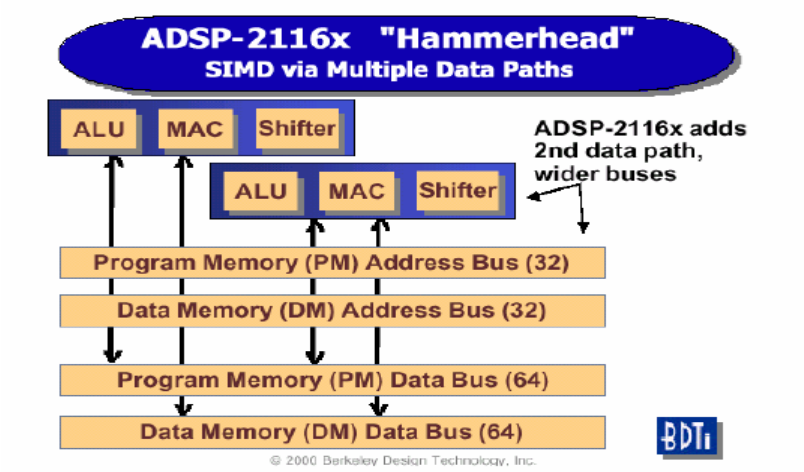


Figura 3-2 Arquitectura del procesador ADSP21160

- Los buses de datos PM y DM (memoria de programa y de datos) tienen acceso a una SRAM interna de doble puerto, la cual está dividida en dos bloques de 2 Mbit.

- El código de cada procesador DSP debe ser cargado en la memoria de programa aunque también puede ser compartida para almacenar datos junto con la memoria de datos.

- Las unidades de computación están compuestas básicamente por un desplazador, una unidad aritmético-lógica y un multiplicador. La unidad de computación secundaria se usa cuando se llevan a cabo operaciones SIMD.

Para más información sobre procesador ADSP21160M véase el apéndice A donde se describen más funcionalidades en el contexto de la tarjeta DSP de Bittware.

3.4 Protocolo Camera Link

El protocolo Camera Link es un interfaz de comunicaciones para aplicaciones de visión basado en la tecnología *Channel Link* La Figura 3-3 proporciona un diagrama de sus tres configuraciones.

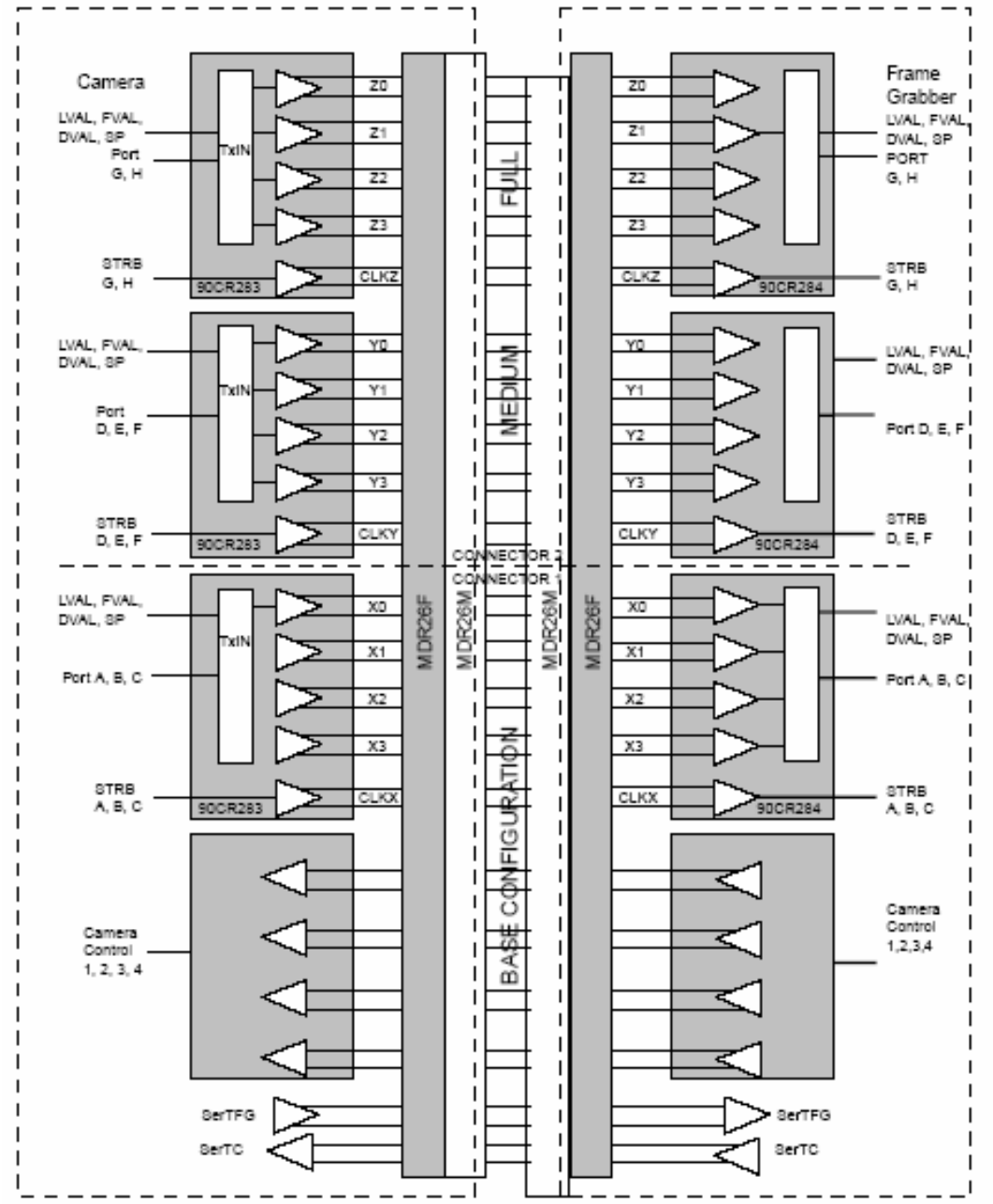


Figure 3-3 Diagrama de bloques de la configuración Base, Medium, y Full

3.4.1 Channel Link

El protocolo Channel Link se define para un transceiver (transmisor – receptor). La Figura 3-4 ilustra su funcionamiento.

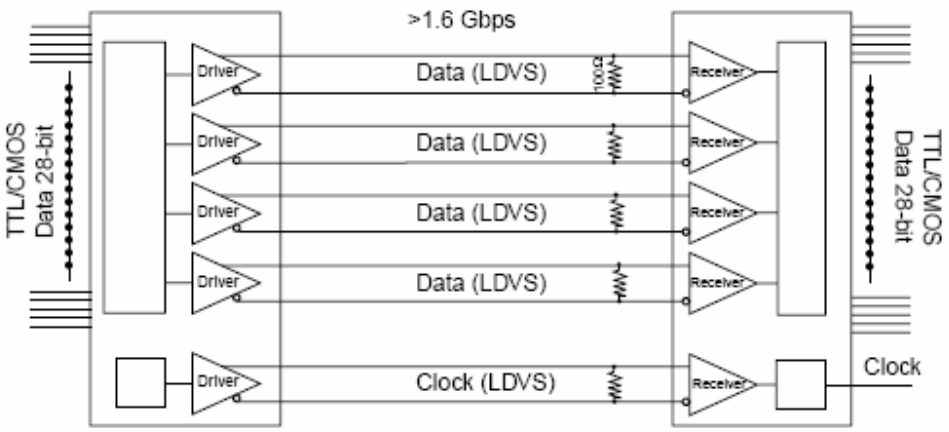


Figura 3-4 Funcionamiento del protocolo Channel Link.

Características de la comunicación Channel Link:

- El transmisor genera 28 señales de datos y una de reloj.
- Los datos son comprimidos por un factor 7:1.
- Las cuatro secuencias de datos se transmiten a través de cuatro pares LVDS.
- El receptor acepta esas cuatro señales LVDS y las conduce hacia la tarjeta destino (normalmente un frame-grabber) de nuevo en formato descomprimido (28 bits)

3.4.2 Asignación de Puertos

Debido a que cada chip Channel Link está limitado a 28 bits de datos, algunas configuraciones del interfaz CameraLink pueden requerir varios chips para transferir los datos.

El interfaz CameraLink tiene tres configuraciones posibles con uno o dos cables:

- Base Un chip Channel Link, un conector para cable.

- Medium Dos chips Channel Link, dos conectores para cable.
- Full Tres chips Channel Link, dos conectores para cable.

En el interfaz CameraLink las transferencias se basan en palabras de 8-bits llamadas puertos. La Tabla 3-1 muestra la asignación de puertos para las configuraciones base, media, y completa.

Configuration	Ports Supported	Number of Chips	Number of Connectors
Base	A, B, C	1	1
Medium	A, B, C, D, E, F	2	2
Full	A, B, C, D, E, F, G, H	3	2

Tabla 3-1 Asignación de Puertos de acuerdo a la Configuración.
Un puerto se define como una palabra de 8-bits.

3.5 Cámara Line-Scan Atmel-Aviiva M2CL

Ésta cámara CCD utiliza la configuración base del estándar de CameraLink. Es por ello que sólo se requiere un cable y únicamente son soportados 3 puertos (A, B, C).

3.5.1 Características

Un resumen de características de la cámara CCD se detalla a continuación:

- Formato de datos CameraLink: Configuración Base
- Los datos se pueden obtener en dos canales o en un solo canal multiplexado.
- Resolución de 1024 píxeles de 10 µm
- El formato de los píxeles se puede configurar en 8, 10 o 12 bits.
- Píxel rate de 10 MHz
- Máximo line-rate de 19.53 Klíneas por segundo

- La cámara se puede utilizar con disparador externo o no (modos de sincronización *free run* y *triggered*).
- La configuración y los ajustes de la cámara se realizan vía línea serie.
- Alta sensibilidad y alta SNR (eficiencia lineal del CCD)

Para la aplicación final se elije píxeles de 10 bits en dos canales (sin multiplexar) y el modo de sincronización con disparador externo (modo trigger).

Para más información sobre la configuración de esta cámara, véase el apéndice D en donde se explica el configurador de cámaras Intellicam de Matrox.

3.5.2 Modo Free Run

Es el modo de adquisición más rápido y simple de programar ya que no requiere disparadores externos. Los tiempos de integración y de lectura comienzan automática e inmediatamente después del período de línea anterior (Figura 3-5).

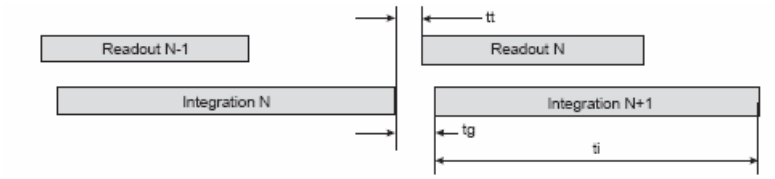


Figura 3-5 Modo Free Run

3.5.3 Modo Trigger

El tiempo de integración comienza inmediatamente después del flanco de subida de la señal de entrada TRIG1 y su duración es programada por la línea serie. A este período de integración le sigue inmediatamente el período de la lectura o también llamado *readout* (Figura 3-6).

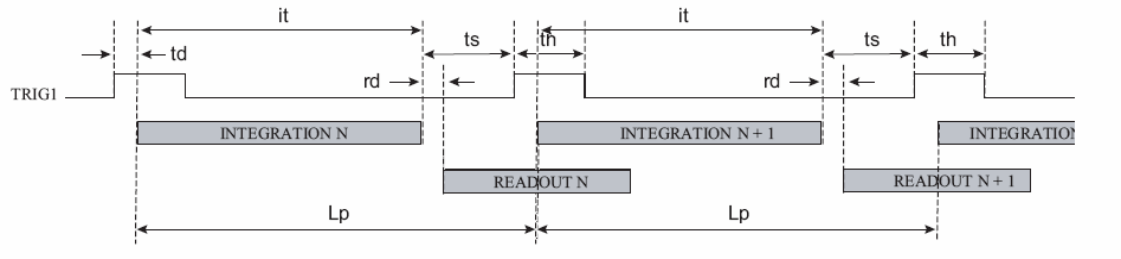


Figura 3-6 Modo Trigger

Especificaciones del Disparador TRIG1 (Line Trigger)

Antes del nuevo tiempo de lectura (*readout)*, habrá suficiente tiempo para procesar en tiempo real los 1024 píxeles de entrada sólo en el caso de que se utilice la cámara a una velocidad lenta de adquisición de líneas (*line rate*).

Todo depende del momento en que el nuevo flanco de subida del disparador TRIG1 esté llegando y de cuánto tiempo dura la integración. Las especificaciones mínimas del disparador *Line Trigger* para garantizar tomografías FDOCT en tiempo real son:

- Line Period = 100 us (10 KHz)
- Integración = 47 us

La duración de la lectura depende de número del píxeles en la cámara y del reloj de píxel (*píxel rate* = 10 MHz).

Camera Data Rate = 2 píxeles @ 10 MHz = 20 Mpixel/s

Readout Time = 0.1 us × 512 = 51.2 us (19.53 KHz) para cualquier *line rate*

El tiempo *readout* es independiente del *line rate* determinado por el disparador *Line Trigger*. Es decir, aun usando la cámara con un *line rate* lento, el flujo de datos producido por la cámara será el mismo que con el máximo *line rate* de la cámara (entorno a 20 Klíneas/s):

3.6 Scanner de Adquisición de Imágenes

El uso de un scanner está justificado para proporcionar una secuencia de imágenes 2D sin necesidad de mover la cámara line-scan de su emplazamiento. La Figura 3-7 es una fotografía de éste dispositivo.

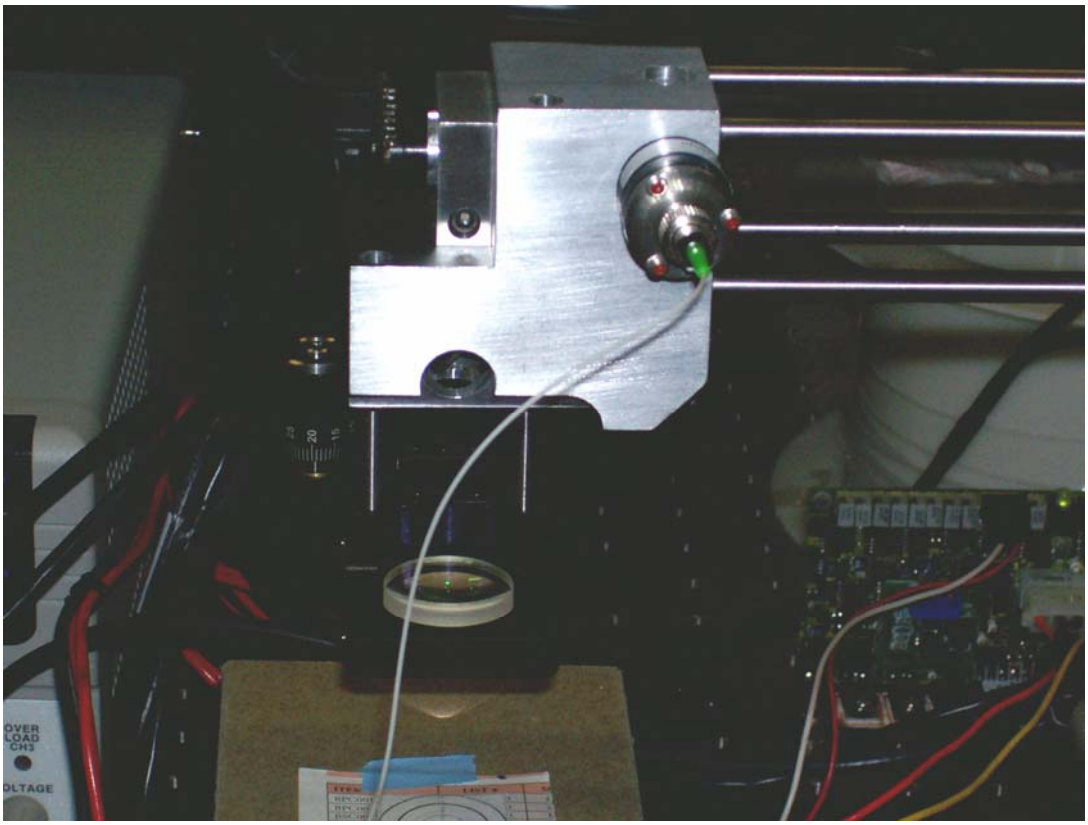


Figura 3-7 Scanner de adquisición de imágenes (fotografía)

El scanner de adquisición de imágenes no es más que un motor de precisión que controla el movimiento de dos espejos perpendiculares: uno para pasar a una nueva imagen (frame) y otro para pasar a una nueva línea a grabar dentro del mismo frame. Este último motor funciona a la misma frecuencia *Line Trigger* de la cámara ya que ambos han de estar sincronizados.

3.6.1 Señales de Control del Scanner

Las señales para el control de los motores internos del scanner de adquisición de imágens son señales analógicas como muestra la Figura 3-8:

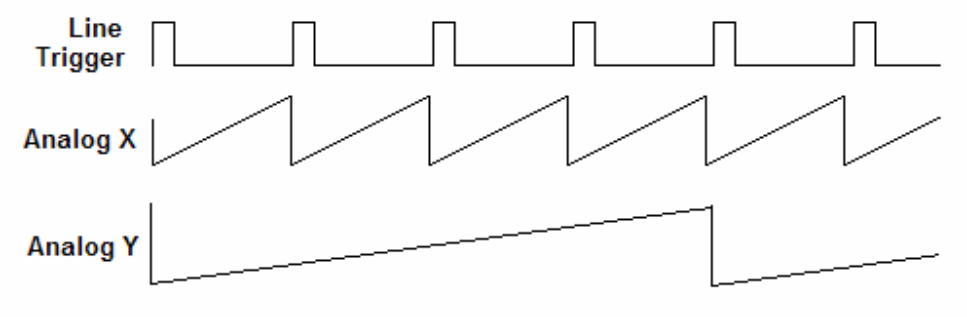


Figura 3-8 Control de motores del scanner (ejemplo para frames de 4 líneas)

Analog X (Line Trigger)

Como se ha comentado, se trata de una señal analógica con la misma frecuencia que el *Line Trigger* de la cámara line-scan.

- Line Period = 100 us (10 KHz)

Analog Y (Buffer Trigger)

Determina el tamaño de la imagen 2D que se genera. A este parámetro también se le conoce como *Frame Trigger* ya que está directamente relacionado con el número de líneas en un frame. En realidad es un reducto del pasado ya que el concepto de frame es solo aplicable cuando hay de por medio un frame-grabber pero este mismo disparador será utilizado por el scanner de adquisición de imágenes.

No es crítico para la aplicación final por ello su valor no es demasiado importante a lo largo del presente proyecto; para frames de 32 líneas su frecuencia valdrá:

- Buffer Trigger = 10 KHz ÷ 32 Líneas = 312.5 fps

3.7 Resumen

En el presente capítulo se ha dado una descripción del hardware disponible:

- Tarjeta DSP (Bittware HammerHead-PCI con cuatro procesadores ADSP21160)
- Cámara line-scan (Atmel AViiVA M2 CameraLink, 1024 píxeles, 16 Bit)
- Scanner de adquisición de imágenes

Únicamente se han descrito las características principales que se van a utilizar en el desarrollo del proyecto, es decir, en lo referente a la tomografía FDOCT.

Capítulo 4
Estado del Arte en el Procesado FFT

4.1 Introducción

En este capítulo se va a comparar el tiempo de procesado del algoritmo FFT complejo en algunas de las plataformas actuales (año 2007):

- Procesador Xeon® (modo 32 bits)
- Procesador Pentium®
- Procesador TigerSHARC®
- Procesador HammerheadSHARC®

Se considerará (como en la mayoría de benchmarks de rendimiento) el caso de 1024 puntos complejos cifrados en punto flotante (precisión simple, de 32 bits). Después de esto, se podrá considerar las ventajas y desventajas de cada elección.

Un nuevo procesador DSP será propuesto en el caso que estemos interesados en una cámara más potente. Además, para terminar se introducirá una visión actualizada sobre sistemas multiprocesadores en el mercado actual.

4.2 CFFT en Procesadores Intel® x86

4.2.1 Procesadores *Pentium*® y *Xeon*®

La librería FFTw está optimizada para la arquitectura de cada procesador de propósito general, por tanto el tiempo de cálculo no sólo depende de la frecuencia de reloj. El programa que se encuentra en el Apéndice E calcula el algoritmo FFT de N = 1024

puntos de precisión simple (32 bits) mostrando el tiempo que la librería FFTw toma para dos tipos diferentes de plataformas:

	Procesador Xeon® @ 3.6 GHz (modo 32 bits)	*Procesador Pentium® @ 1.73 GHz*
Tiempo de Procesado	**19 µs**	**33 µs**

Tabla 4-1 Benchmark de la FFT de 1024 muestras complejas en diferentes procesadores Intel® x86
** Perfiles calculados de la media de 1000 ejecuciones*

4.3. CFFT en Procesadores Analog Devices®

4.3.1 Procesadores *SHARC®*

La rutina FFT raíz-4 compleja, incluida en las librerías del SHARC ha sido optimizada para el modo de funcionamiento SIMD: cuando se procesa dos canales de datos, el programa esencialmente ejecuta dos copias del algoritmo simultáneamente.

Tiempo Procesado = 18,228 ciclos @ 100 MHz = 182 µs (dos canales)

Longitud FFT	**ciclos**	**tiempo (us)**	**tiempo per canal (us)**
64 (mínimo)	832	8	4
1024	18,228	182	91

Tabla 4-2 Benchmark de diferentes longitudes FFT en procesadores Hammerhead®

Procesadores SHARC mas recientes mejoran el tiempo de procesado:

	ADSP-2116xN SIMD	*ADSP-21261 SIMD*	*ADSP-2126x SIMD*	*ADSP-2137x SIMD*	*ADSP-21364 ADSP-21365 SIMD*	*ADSP-21368 ADSP-21369 SIMD*
Clock	100MHz	150MHz	200MHz	266MHz	333MHz	400MHz
FFT_{1024}	**91 µs**	**61.3 µs**	**46 µs**	**34.5 µs**	**28 µs**	**23 µs**

Tabla 4-3 Benchmark de la FFT de 1024 muestras complejas en diferentes procesadores SHARC

4.3.2 Procesadores *TigerSHARC®*

Los procesadores TigerSHARC soportan operaciones en coma flotante de 32 bits (3.6 GFLOPs) y operaciones de punto fijo de 16 bits (GOPs 14.4). Usando su característica SIMD, el TigerSHARC TS201S (revisión 2.0) emplea 15.64 µs para computar el algoritmo FFT de 1024 muestras flotantes (raíz-2, 32 bits):

Tiempo de Proceso = 9384 ciclos @ 600 MHz = 15.64 µs

Longitud FFT	ciclos	tiempo (us)
32 (mínimo)	412	0.69
1024	9384	15.64

Tabla 4-4 Benchmark de diferentes longitudes de la FFT en el TigerSHARC®

4.4 Sistemas Multiprocesadores Analog Devices

Teniendo en cuenta las especificaciones, se considerará como parámetro principal para la elección de la tarjeta los puertos externos de alta velocidad ya que serán utilizados para conectar la cámara CameraLink. Además, se considerará el interfaz hacia el PC anfitrión: para aplicaciones en tiempo real con un alto flujo de datos hacia el PC anfitrión, este parámetro será muy importante.

4.4.1 Sistemas Multiprocesador *HammerheadSHARC®*

BittWare HH-PCI

- Cuatro procesadores @ 80 MHz con 4Mbit
- Interfaz PCI hacia el PC anfitrión
- SDRAM on board de 64 MB
- 4 Link Ports @ 80 MB/sec cada uno

Analog Device ADSP21160 EZ-KIT Lite

- Dos procesadores @ 100 MHz con 4Mbit
- Interfaz de depuración USB con el PC anfitrión
- SBSRAM on board de 512 KB
- 2 Link Ports @ 80 MB/sec cada uno

4.4.2 Sistemas Multiprocesador *TigerSHARC®*

Analog Device TS101S EZ-KIT

- Dos procesadores ADSP-TS101S @ 250 MHz
- Interfaz de depuración USB con el PC anfitrión
- 32 MB SDRAM on board
- 2 Link Ports TigerSHARC @ 250 MB/sec cada uno

Analog Device TS201S-EZ Lite

- Dos procesadores TS201S @ 500 MHz
- Interfaz de depuración USB con el PC anfitrión
- SDRAM on board de 32 MB
- 2 Link Ports TigerSHARC @ 250 MB/sec cada uno

BittWare Tiger-PCI

- Cuatro procesadores TS101S @ 250 MHz
- Interfaz PCI hacia el PC anfitrión
- SDRAM on board de 64 MB
- 4 Link Ports TigerSHARC @ 250 MB/sec cada uno

BittWare T2-PCI

- Cuatro procesadores TS201S @ 600 MHz
- Interfaz PCI hacia el PC anfitrión
- SDRAM on board de 128 MB
- Throughput de E/S externa a 4 GB/sec gracias a la FPGA Virtex-II Pro de Xilinx:
    - 8 Link Ports TigerSHARC @ 250 MB/sec cada uno
    - 128 canales de alta velocidad DIO (single-ended y/o LVDS)

4.5 TigerSHARC® vs HammerheadSHARC®

En primer lugar se va a eliminar la opción del procesador de Xeon® debido a su precio: el coste de un procesador DSP actual es casi 3 veces menor que una estación moderna con dicho modelo de procesador Intel® aunque solamente le llevaría 19 us computar el algoritmo FFT complejo usando la biblioteca de FFTw.

	Procesador Xeon® @ 3.6 GHz	*Procesador TigerSHAR® @ 600MHz*	*Procesador SHARC® @ 100 MHz*
Tiempo de Procesado	**19 µs**	**15.64 µs**	**91 µs**

Tabla 4-5 Benchmark en diferentes procesadores

4.5.1 Ventajas del Procesador *HammerheadSHARC®*

A pesar de que el cálculo del algoritmo FFT requiera más tiempo, como se observa en la Tabla 4-5, únicamente disponemos del procesador ADSP21160 integrado en la tarjeta multiprocesador Bittware HH-PCI. Entre otras, éste proporciona las siguientes ventajas:

1. Más barato que otras opciones; por el mismo precio, la tarjeta Bittware HH-PCI proporciona mejores comunicaciones hacia el PC anfitrión: el bus PCI en vez del USB.

2. Mejor relación Watts/MFLOPs: una frecuencia de reloj más baja implica menor número de transistores lo que repercute en un menor consumo.

3. Más aplicaciones, ejemplos, pruebas… en resumen, mejor conocido gracias a que ha estado más tiempo en el mercado.

4.5.2 Ventajas del Procesador *TigerSHARC®*

El núcleo extremadamente rápido de este procesador y el conjunto extendido de características lo convierte en la actualización ideal para sistemas SHARC ya existentes que buscan una mejora de rendimiento.

4.5.3 ¿Qué sistema multiprocesador sería recomendable?

Las aplicaciones de imagen de uso médico actuales, tales como la FDOCT, MRI, los scanners CT, y los sistemas del ultrasonido poseen una gran variedad de requisitos desde control de motores hasta cálculos a alta velocidad. Las tarjetas híbridas de procesado de señal de BittWare (DSP + FPGA) proporcionan una solución ideal para los diseñadores de sistemas médicos de tratamiento de imagen. Estas tarjetas proporcionan la flexibilidad de la programación de un FPGA y la potencia de cálculo de un DSP permitiendo a los diseñadores de sistemas que las emplean aprovechar estas ventajas para poner rápidamente sus productos en el mercado a un coste rentable.

4.6 Resumen

Como se comentó en las motivaciones del presente proyecto, la tarjeta HH-PCI de BittWare se adquirió en el año 2001, es por esto que sólo cabría comparar la potencia de su procesador ADSP21160 con microprocesadores de esa época[1].

Es claro que cualquier procesador actual (finales del año 2007) podría mejorar la frecuencia máxima de procesado alcanzada por el procesador ADSP21160.

En este capítulo hemos visto el estado del arte en lo referente al cálculo del algoritmo FFT complejo en procesadores actuales en el caso de que en el futuro estemos interesados en una nueva cámara más potente (Figura 4-1).

[1] La Figura 7-3 del capítulo 7 (Arquitecturas de Bus) muestra la velocidad a la que algunos de esos procesadores realizan el procesado FFT.

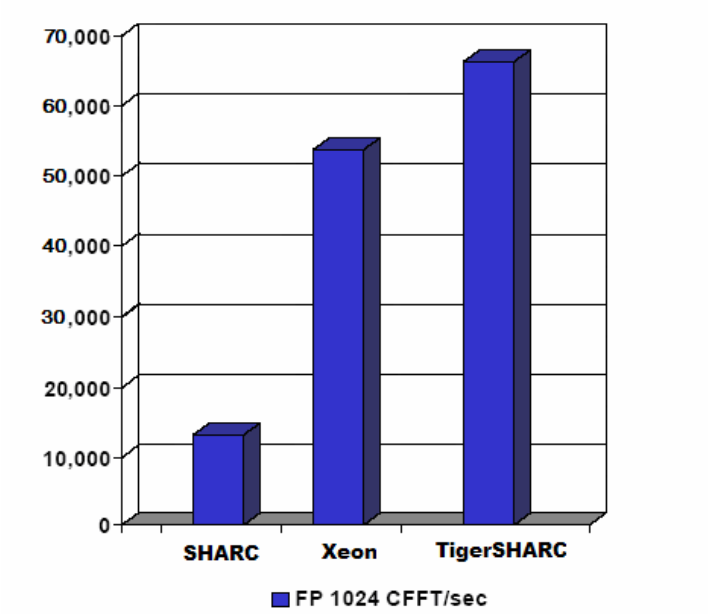


Figura 4-1 Rendimiento de la FFT en diferentes procesadores (año 2007)

El procesador *TigerSHARC®* ha sido propuesto debido a que es la versión actualizada del *HammerheadSHARC®*. En concreto, se ha recomendado emplearlo en un sistema multiprocesador como la tarjeta BittWare T2-PCI.

Capítulo 5
Especificaciones y Análisis de Alternativas

5.1 Introducción

Este capítulo es uno de los más importantes en todo el proyecto debido a que todos los parámetros de tiempo real de la aplicación FDOCT se definen aquí. A partir de esta definición se estudiarán las posibles alternativas disponibles para desarrollar la FDOCT en tiempo real.

5.1.1 Parámetros de tiempo real

En primer lugar, será especificado el throughput mínimo de salida a 10,000 FFTs/sec que en términos de la aplicación final correspondería al tiempo máximo de procesado de línea para el cual aun podemos estar hablando de FDOCT en tiempo real.

La latencia entendida cómo el tiempo que habrá que esperar para obtener la primera línea a la salida no es un parámetro crítico en la FDOCT por tanto se dejará un margen de tolerancia amplio.

5.1.2 Consideración sobre el formato de los datos

A parte de la latencia y el throughput, otro parámetro con bastante relevancia en la aplicación final es la resolución de los datos de salida en la FFT. Debido a que no se desea perder el valor de ningún píxel por saturación, se considerará como necesaria la

conversión previa de los datos provenientes de la cámara a un formato que permita un rango mayor.
De este modo aunque los datos de entrada sean enteros sin signo de 16 bits (en realidad los píxeles son de 10 bits perdiéndose 6 bits por empaquetado en el procesador) sin embargo, internamente se realizará el computo del algoritmo FFT en punto flotante de 32 bits para garantizar que la precisión de las operaciones intermedias sea elevada y que no haya overflow en los datos de salida.

Por tanto habremos de convertir:

- **a la entrada:** de entero real a punto flotante (U16 → double)
- **a la salida:** de punto flotante a entero real (double → U32)

Por último habrá que tomar la magnitud del resultado debido a que el resultado es complejo. Sin embargo en la segunda parte del proyecto (aplicación *parallelFFT*) no se tomará la magnitud ni se tendrá en cuenta el formato de los datos de entrada.

Por tanto, la principal diferencia entre ambas aplicaciones es que en vez de interesarnos por el algoritmo FFT real de 1024 píxeles de 16 bits (aplicación *dspFDOCT* que nos ocupa), en la aplicación *parallelFFT* nos contentaremos con el algoritmo FFT complejo de 1024 muestras de 32 bits para propósitos comparativos.

5.2 Análisis de alternativas

5.2.1 Procesado de Frames

Hay dos formas diferentes de calcular el algoritmo FFT en la tomografía FDOCT: cada frame (2D) o cada línea (1D). También se puede elegir en qué hardware queremos ejecutar el algoritmo: en el PC, en la FPGA interna del frame-grabber o en la tarjeta DSP. En el laboratorio están disponibles las siguientes plataformas:

- Tarjeta DSP (Bittware HammerHead-PCI con cuatro DSPs)
- PC (DELL WorkStation con el Procesador Xeon®)
- Frame Grabber (Matrox SOLIOS xCL CameraLink)

Hasta ahora el cálculo de la FFT siempre se había hecho por frames (2D) en el PC anfitrión o en el frame-grabber usando una de las siguientes posibilidades:

a) una función estándar en C, Matlab o LabVIEW

b) una librería de funciones en C llamada "FFT in the West" (FFTW) optimizada para la arquitectura específica del PC anfitrión

c) una función en C especifica de la tarjeta Matrox (MimTransform).

5.2.2 Consideraciones sobre el Procesado de Frames

Todas las opciones que utilizan el PC anfitrión para el cálculo de la FFT presentan el mismo problema: el frame-grabber debe primero subir el frame a la RAM para que el PC pueda realizar la FFT. Esta transferencia penalizará la eficiencia en tiempo real pero no obstante analicemos las distintas opciones antes mencionadas:

a) C, Matlab o LabVIEW: demasiada lenta para obtener calidad de tiempo real.

b) FFTW: el programa detallado en el apéndice E realiza la FFT de 1024 puntos flotantes (precisión simple de 32 bits) en un tiempo mínimo de 19 µs luego mejoraríamos considerablemente el resultado de la opción (a).

c) MimTransform: sin duda es la mejor opción porque el algoritmo FFT es ejecutado en la FPGA interna del frame-grabber pudiéndose conseguir altas velocidades de procesado. El problema es que la licencia para dicha función Matrox es muy cara.

5.2.3 Procesado de Líneas

En este proyecto se propone emplear los procesadores de una tarjeta DSP para procesar *línea a línea* (1D) y poder así generar los disparadores necesarios para controlar la cámara y el scanner de adquisición de imágenes.

Los procesadores DSP han sido diseñados, entre otras cosas, para realizar eficientemente análisis de Fourier, por tanto sería muy interesante diseñar conceptualmente un sistema de detección donde cada línea de la cámara sea directamente cargada en la tarjeta DSP.

Al igual que la opción (c), el PC anfitrión no procesará la imagen con lo que no será necesario transferir la imagen en RAM con la consiguiente mejora de velocidad.

5.3 Lista de materiales y tareas

Una vez justificado el uso de una tarjeta DSP, se indican a continuación el material del que disponemos en el laboratorio (http://lob.epfl.ch):

- Tarjeta DSP (Bittware HammerHead-PCI con cuatro DSPs) junto con el emulador JTAG (Summit-ICE)
- Cámara line-scan (Atmel AViiVA M2 CameraLink, 1024 píxeles, 16 Bit)
- PC (DELL WorkStation con el Procesador Xeon®)
- Scanner de adquisición de imágenes sincronizado con la camera
- Varios entorno de desarrollo de software (Visual Studio, LabView, diag21k, VisualDSP++, Matlab…)

Por tanto, las tareas necesarias para llevar a cabo la tomografía FDOCT en tiempo real serán:

1. programar la **Tarjeta DSP** existente para que

   a. se comunique con la cámara line-scan y lea los datos
   b. calcule la FFT de dicho flujo de datos
   c. se ocupe de todas las señales de sincronización hacia la cámara line-scan y el scanner

2. escribir un programa para el **PC Anfitrión** para que
   a. se comunique con el procesador DSP maestro
   b. lea los datos de la FFT procesada por la tarjeta DSP
   c. visualices los datos procesados (1/10 frames procesados por ejemplo)

5.4 Resumen de las especificaciones software

Para llevar a cabo el cómputo en tiempo real del algoritmo FFT empleado en la tomografía FDOCT se utiliza una tarjeta PCI con varios procesadores DSP. En este capítulo se justifica esta elección y se considera como adecuada la tarjeta DSP de Bittware modelo HH-PCI.

Además se fijan los requisitos de funcionamiento de la aplicación *dspFDOCT*:

- Cada línea del frame estará formada por 1024 enteros unsigned de 16 bits.

- Internamente los DSPs efectúan la FFT en punto flotante por tanto la conversión de datos *U16* → *double* será requerida por cada píxel de la entrada. A la salida se obtendrá mayor resolución, en concreto, la conversión de datos *double* → *U32* será requerida.

- Los distintos DSPs de la tarjeta HH-PCI trabajarán en conjunto para producir al menos 10,000 FFTs/sec

Capítulo 6
Diseño Hardware (HH-PCI)

6.1 Introducción

En este capítulo se toman decisiones hardware sobre el uso de la tarjeta Hammerhead-PCI, tanto en lo referente al conexionado externo como recursos internos a utilizar.

La Figura 6-1 muestra dos módulos externos a través de los cuales la tarjeta DSP puede controlar:

- la cámara line-scan (line grabber)
- el scanner (circuito conversor digital-analógico)

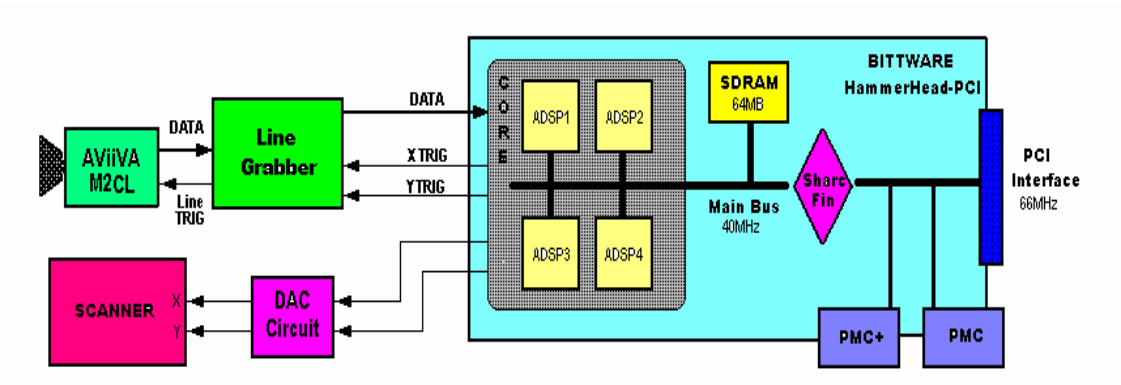


Figura 6-1 Conexión de la tarjeta DSP
a la cámara line-scan (line grabber) y al scanner (conversor DA)

Este capítulo está estructurado en base a estos dos módulos:

- En primera parte del capítulo se especifica el conexionado de la tarjeta DSP con la cámara definiéndose un interfaz *line grabber* diseñado para tal fin.

  Debido a su complejidad, el interfaz *line grabber* no va ser implementado, sin embargo sus especificaciones funcionales serán perfectamente definidas. Para poder continuar analizando las posibilidades de procesado de la tarjeta Hammerhead-PCI, se formula una aproximación a la que nos restringiremos a lo largo de este proyecto.

- Para terminar se especifica el interfaz de la tarjeta DSP con el scanner de adquisición de imágenes.

6.2 Implementación del Line Grabber

Como se ha comentado, la mayor complejidad del nuevo enfoque que plantea este proyecto radica en la necesidad de adaptar la tarjeta DSP al protocolo CameraLink en su configuración básica (2 taps, 10 bit píxeles).

Para ello se define el módulo *line grabber* que adaptará los datos provenientes de la cámara para poder ser utilizados por los procesadores de la tarjeta DSP.

6.2.1 Empaquetado de píxeles en el interfaz CameraLink

La configuración del protocolo CameraLink utilizada en este proyecto proporciona dos flujos simultáneos de píxeles de 10 bits (dual tap). Esto quiere decir que a cada ciclo de reloj están disponibles dos píxeles.

La Tabla 6-1 muestra la manera en que ambos píxeles se empaquetan en los tres puertos disponibles en la configuración básica de CameraLink.

Port A		Port B		Port C	
Port A0	A0	Port B0	A8	Port C0	B0
Port A1	A1	Port B1	A9	Port C1	B1
Port A2	A2	Port B2	nc	Port C2	B2
Port A3	A3	Port B3	nc	Port C3	B3
Port A4	A4	Port B4	B8	Port C4	B4
Port A5	A5	Port B5	B9	Port C5	B5
Port A6	A6	Port B6	nc	Port C6	B6
Port A7	A7	Port B7	nc	Port C7	B7

Tabla 6-1 Asignación de bits de los píxeles (configuración base)

La correspondencia espacio-temporal de los dos píxeles simultáneos a lo largo del array de 1024 CCDs y los 512 pulsos de reloj de una línea es explicada en la Figura 6-2.

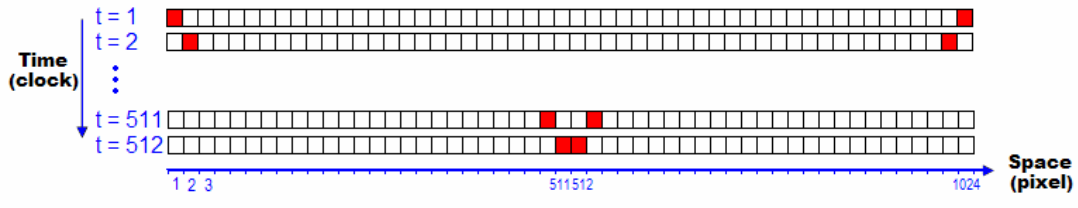


Figura 6-2 Configuración de taps.
Correspondencia espacio-temporal de píxeles a lo largo del array CCD y durante el periodo de una línea.

En esta figura se observa que el flujo de datos procedente de la cámara consiste en 2 píxeles no contiguos cada flanco activo del reloj interno a la cámara (10 MHz). Esto quiere decir que la línea que el *line grabber* carga en la tarjeta DSP no está ordenada sino que se alternan píxeles de la mitad izquierda con los de la mitad derecha.

6.2.2 Componentes del Line Grabber

En la Figura 6-3 se muestra con más detalle el conexionado de la tarjeta DSP con la cámara a través del módulo externo al que denominamos *line grabber*.

En esta figura se puede distinguir los siguientes elementos:

- el chip 90CR284 corresponde a un receptor *Channel Link*
- el transmisor LVDS que genera la señalización hacia la cámara
- el transceiver LVDS y la UART para la comunicación serie.

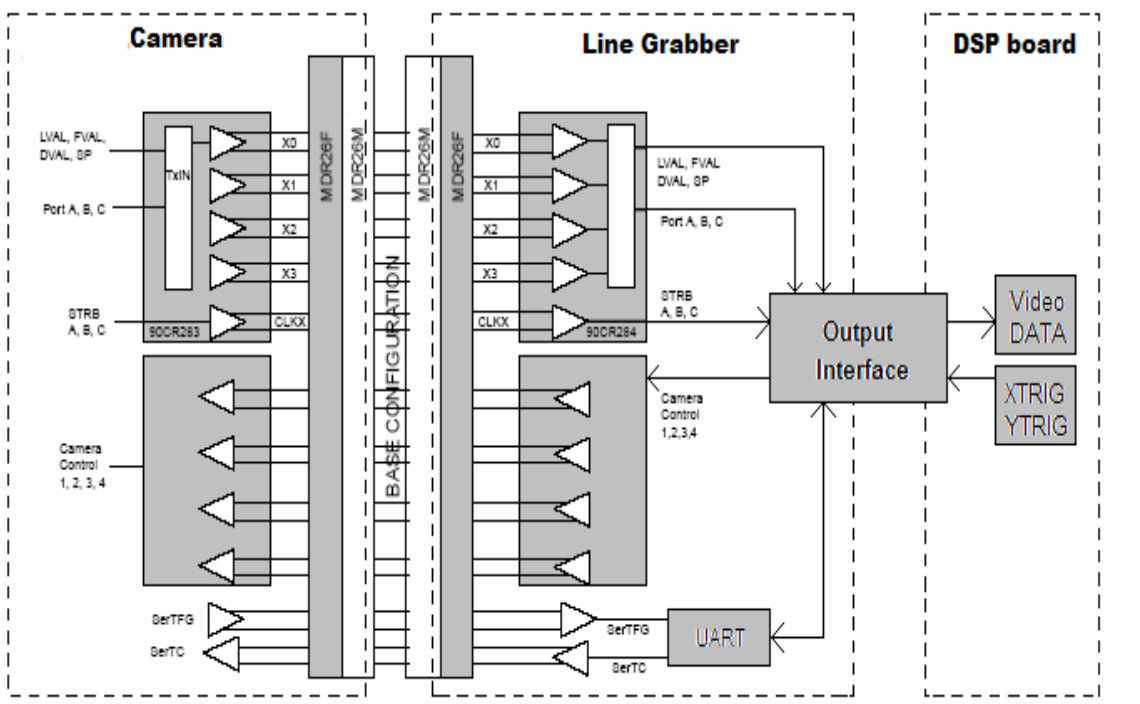


Figura 6-3 Componentes del Line Grabber

El módulo denominado *Output Interface* podría ser implementado para adaptar el protocolo CameraLink a los siguientes recursos del ADSP21160:

- La forma más sencilla de capturar el flujo de datos es a través de los enlaces link port del procesador DSP maestro[1] de la tarjeta ya que éstos pueden soportar hasta 16 bits @ 40 MHz. Como tenemos un flujo de entrada de 16 bits @ 10 MHz por taps, usaremos dos link ports (uno por tap) trabajando a frecuencia máxima del reloj principal del procesador (8 bits @ 80MHz).

- Los triggers XTRIG e YTRIG serán implementados usando las entradas/salidas FLAG2 y FLAG3 del procesador DSP maestro (*Line Y Frame Trigger* respectivamente).

Hay que tener en cuenta que la adaptación de datos realizada por el módulo *Output Interface* para que puedan ser transmitidos a través de los enlaces link ports consistirá principalmente en las siguientes tareas:

- ordenado de la posición de los píxeles en un buffer a medida que se vayan recibiendo (Figura 6-3)

- empaquetado de dos píxeles de 10/16 bits en 32 bits (se ha elegido este formato por simplicidad pero se podría optimizar transferencia en el futuro empaquetando hasta 3 píxeles de 10 bits en 32 bits)

- transmisión usando el protocolo link port propio del procesador ADSP21160

Se considera que una FPGA o un microcontrolador podrían ser utilizados para hacer estas tareas (*Output Interface).*

[1] Denominamos procesador DSP maestro aquel que se comunica directamente con el PC anfitrión de la tarjeta DSP y que a su vez recibe los datos de entrada y proporciona la señalización de sincronización hacia el scanner y la cámara CCD.

6.3 Aproximación del Line Grabber

Debido al trabajo que supondría diseñar e implementar físicamente el interfaz *line grabber*, se simulará su salida. De otro modo habría que hacer pruebas con la cámara y los enlaces link ports del procesador, lo que se nos antoja complicado a primera vista.

6.3.1 Generación de patrón fijo

Una posible aproximación sería simular que los CCD de la cámara capturasen una línea patrón constante como la que se muestra en la Figura 6-4.

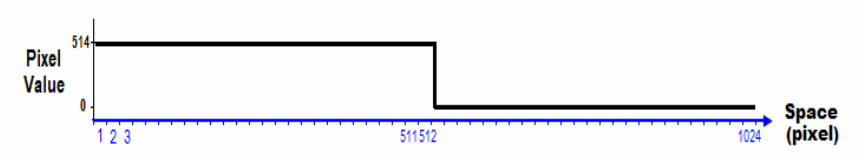


Figura 6-4 Línea Patrón (valores simulados)

Veamos cómo se puede generar una imagen como la mostrada en dicha figura.

Puesto que tenemos 2 píxeles por cada pulso del reloj de la cámara (10 MHz) se va a usar dos link ports del procesador DSP maestro para la entrada de datos:

- uno para los píxeles de la zona izquierda del array de CCDs (píxeles 1 a 512)
- otro para los píxeles de la derecha del array de CCDs (píxeles 513 a 1024).

Cada link port está formado por 8 bits de datos. Podríamos, por ejemplo, fijar a uno el bit número #1 (D1) del link port asociado a los píxeles de la izquierda y el resto de bits a cero.

Aunque el cable link-port esté formado por 8 hilos, el procesador sólo interpreta los datos en formato de 32 bits debido a que su núcleo está basado en ese tamaño. Esto significa

que el valor interpretado por el DSP no cabe en un número de 10 bits que es el tamaño real de un píxel:

Bit D1 de los píxeles zona izquierda = 1 → (32 bits) = 0x0202.0202 > 2^{10} = 1024

Para ser lo más fiel posible al comportamiento de la cámara, empaquetaremos dos píxeles (de 16 bits) en una palabra de 32 bits y se solucionaría el problema:

Bit D1 píxeles zona izqda = 1 → dos píxeles con el mismo valor 0x0202 = 514 < 1024

De este modo se optimiza la transferencia de píxeles y se respeta el formato de salida de los datos de la cámara. Todas las transmisiones se harán a la máxima frecuencia de octeto, esto es:

8 bits @ 80 MHz = 16 bits @ 40 MHz = 32 bits @ 20 MHz.

Podemos decir que cada 50 ns (20 MHz) obtendremos 2 píxeles izquierdos y 2 derechos (4 píxeles en total). El segundo link port asociado a los píxeles de la izquierda, estará fijado siempre a cero (cortocircuitado a un cero lógico).

6.3.2 Simulación de patrones variables

A pesar de haberse adquirido dos cables link port para llevar a cabo la prueba anterior, se estimó que tomaría mucho tiempo su implementación. Además dicha prueba no es exhaustiva ya que sólo prueba el algoritmo FFT para un escalón de entrada dejando a un lado muchos otros patrones.

La configuración de la Figura 6-5 sirve para poder comprobar el procesado de líneas más complejas.

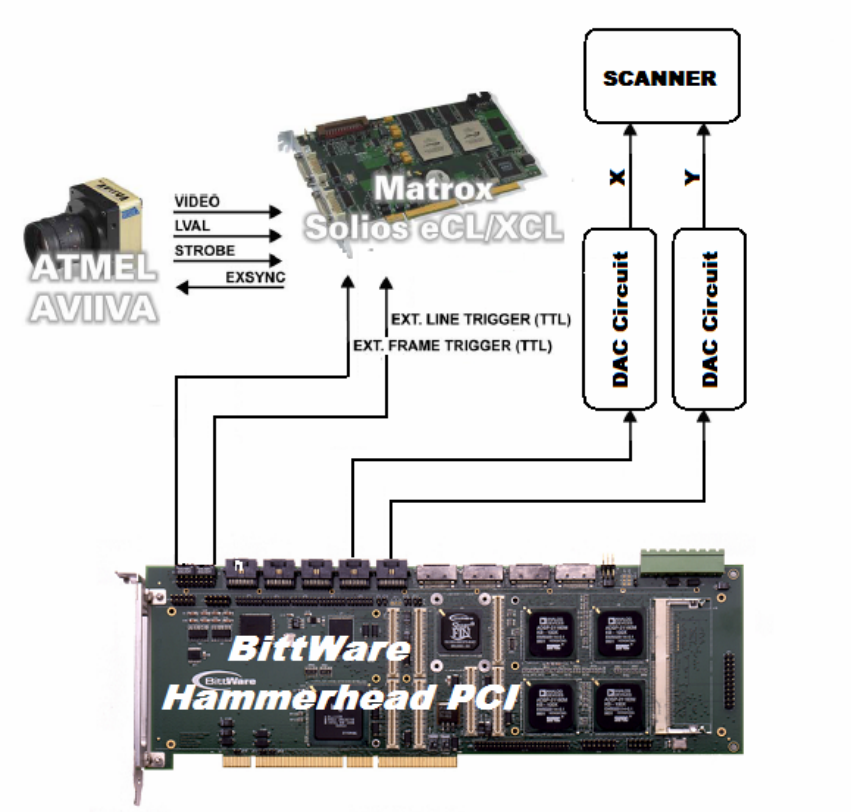


Figura 6-5 Conexión de la tarjeta DSP, cámara line scan y scanner (frame grabber)

Supondremos que el *frame-grabber* carga imágenes directamente en la SDRAM de la tarjeta DSP[2] a un ritmo indicado por ella misma a través de la señal *frame-trigger*. Por otra parte, para simular la señal *line-trigger*, procesaremos línea a línea cada frame cargado, a un ritmo indicado por un temporizador (simulación software).

En la práctica, será el propio PC anfitrión quien cargue en la memoria de la tarjeta los frames previamente preparados simplificando así la parte hardware para poder centrarnos en las posibilidades del procesado en tiempo real. Ningún frame grabber será empleado como indica la Figura 6-5.

[2] Existen *frame-grabbers* que pueden ser conectados localmente a la tarjeta DSP por ejemplo a través del bus PMC

6.4 Sincronización con el scanner

Hasta ahora sólo se había considerado los disparadores generados por la tarjeta HH-PCI para controlar la cámara a través del *line grabber* (salidas digitales XTRIG e YTRIG en la Figura 6-1).

Sin embargo, para poder obtener una secuencia de imágenes bidimensionales es necesario sincronizar la cámara con el scanner. La Figura 6-6 muestra las señalas adicionales que el scanner necesita.

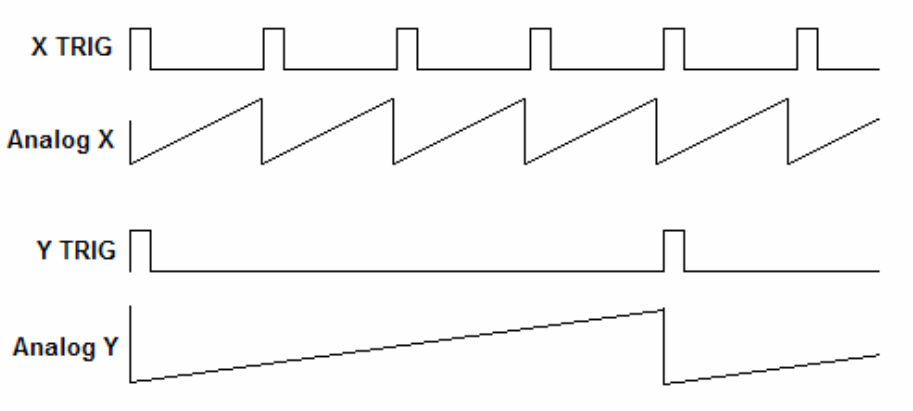


Figura 6-6 Sincronización Scanner – Cámara.

X TRIG / Y TRIG = *Line / Frame Trigger de la cámara (digital)*
Analog X / Analog Y = *Line / Frame Trigger del scanner (analógico)*

El scanner requiere por tanto entradas analógicas para poder trabajar (Analog X e Y). Debido a que la tarjeta HH-PCI no dispone de ninguna salida analógica se requerirá un circuito de conversión D/A externo.

Los puertos series del ADSP21160 (SPORT0 y/o SPORT1) serán empleados para atacar conversores serie D/A que proporcionarán estas señales analógicas de control del scanner.

En la Figura 6-7 se muestra el conexionado externo del puerto SPORT0 siendo análogo para el SPORT1.

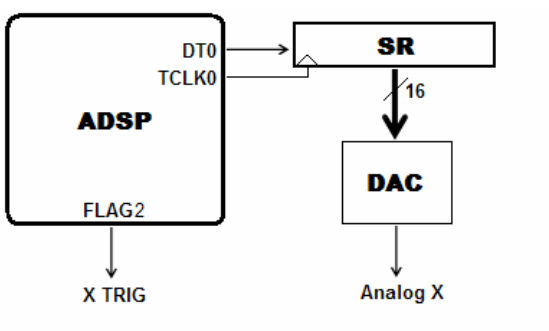


Figure 6-7 Circuito de Conversión D/A

Hay que matizar que la sincronización entre las señales *X-TRIG* y *Analog X* puede ser complicada de obtener. La precisión temporal es muy importante en este sentido ya que cualquier retraso por pequeño que sea puede ser fatal en la adquisición de la imagen final.

Es por eso que en el presente proyecto se limita a la programación de los puertos series y no se entra en la construcción hardware del circuito conversor ni en posteriores pruebas con el scanner.

6.5 Resumen de las especificaciones hardware

En este capítulo se han fijado los siguientes requisitos hardware de las aplicaciones *dspFDOCT* y *parallelFFT*:

- El PC anfitrión carga las imágenes directamente en la SDRAM de la tarjeta DSP a un ritmo indicado a través de la señal *frame-trigger*.

- Para simular la señal *line-trigger*, se procesa cada frame cargado línea a línea, a un ritmo impuesto por el temporizador del procesador maestro.

- Los disparadores de la cámara serán implementados usando las entradas/salidas FLAG2 y FLAG3 del DSP maestro.

- Se programará los puertos series SPORT0 y SPORT1 para generar la señalización de control del scanner sin embargo no se testeará con el scanner.

PARTE II:
Aplicación *dspFDOCT*

Capítulo 7
Arquitecturas de Bus (HH-PCI)

7.1 Introducción

Para implementaciones multiprocesador, la posibilidad de escalar la velocidad del sistema añadiendo procesadores es un factor crítico y a veces sobrestimado que puede limitar la eficiencia global del sistema.

Actualmente las tarjetas con procesadores de Analog Devices ofrecen dos tipos de arquitecturas de bus disponibles: con solo una memoria global a todos los procesadores o bien con memorias locales a cada procesador.

7.1.1 Memoria Global y Cluster-Bus

En esta arquitectura, todos los procesadores, módulos de E/S y memoria global están conectados al mismo bus común (Figura 7-1).

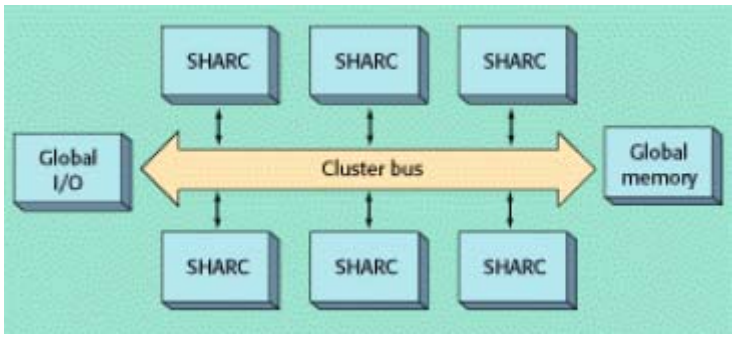


Figura 7-1 Memoria Global y Cluster-Bus

El principal inconveniente de este diseño es que todos los procesadores deben competir por el acceso al bus común, resultando en una degradación del rendimiento sobre todo durante operaciones intensivas de E/S o memoria. En el caso peor donde la aplicación está limitada en memoria, un único procesador trabajará igual de bien o incluso mejor solo que con dos o más procesadores.

7.1.2 Memoria Local y Enlaces Link-Port (DPLM)

En un diseño con memoria privada, también llamado Dual Port Local Memory, cada procesador tiene su propia memoria de la cual puede leer o escribir sin competir por los recursos que están siendo utilizados por otros procesadores. Cada procesador tiene un link-port para conectarse con el procesador maestro. Éste transfiere los datos de la memoria global hacia la privada de cada procesador esclavo (Figura 7-2).

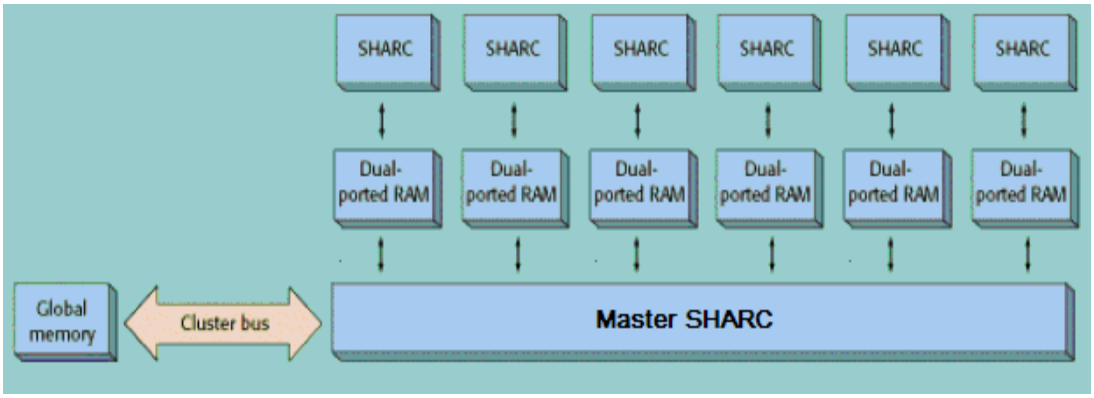


Figura 7-2 Memoria Local y Enlaces Link-Port (DPLM)

El procesador maestro usa el cluster-bus para la entrada de datos y los link-ports que tenga disponibles para distribuir los datos leídos hacia los procesadores esclavos. En este caso el throughput será una variable de entrada.

$$N_{max} = 6 \text{ procesadores esclavos}$$

7.1.3 Comparación entre ambas arquitecturas

A diferencia del diseño de solo una memoria global, la eficiencia de la arquitectura de memorias locales no es degradada cuantos más procesadores sean añadidos porque el ancho de banda disponible por procesador para accesos a memoria permanecerá siempre constante. Para aplicaciones limitadas en memoria, la eficiencia es mejorada con cada nuevo procesador porque el ancho de banda a memoria es escalable linealmente con el número de procesadores.

Debido a dichas disputas de E/S por la memoria entre los procesadores en la arquitectura cluster-bus, se limita el ancho de banda disponible. En el diseño con memorias locales, el bus global debe tener un ancho de banda igual al flujo de datos de

entrada (para acomodar la entrada de datos y el posterior desvío hacia las memorias privadas). Una vez que este requerimiento sea cumplido, la adición de nuevos procesadores no cambia la carga del bus global. Normalmente un procesador maestro o un motor DMA es el encargado de controlar la transferencia de datos desde la memoria global hacia las memorias locales de los procesadores.

Por todo esto, los diseñadores debemos tener en cuenta el ancho de banda de memoria o de E/S disponible al elegir la arquitectura óptima. Algunas arquitecturas fallan porque la memoria común no llega a entregar los datos lo suficientemente rápido a los procesadores, limitando críticamente el rendimiento total de la aplicación. Las memorias locales, por otra parte, pueden obviar este problema.

Una aplicación del mundo real como la tomografía FDOCT ilustra la mejora que se pude obtener con la arquitectura de memorias locales.

7.2 Tomografía FDOCT

La cámara típicamente usada en la tomografía FDOCT proporciona líneas de 1024 píxeles. En una primera aproximación podemos encapsular los píxeles en datos complejos de 64 bits (cada uno formado por dos palabras de 32 bits) resultando un flujo de entrada de 160 MB/s (2048 palabras de 32 bits a una velocidad máxima de 20Klineas por segundo).

El tiempo de procesado requerido deberá ser menor que el line-period de la cámara seleccionado por el técnico lo cual se traduce en un número equivalente de FFTs por segundo (de 1024 números complejos).

La Figura 7-3 muestra la velocidad a la que distintos procesadores del año 2001 realizan el procesado FFT.

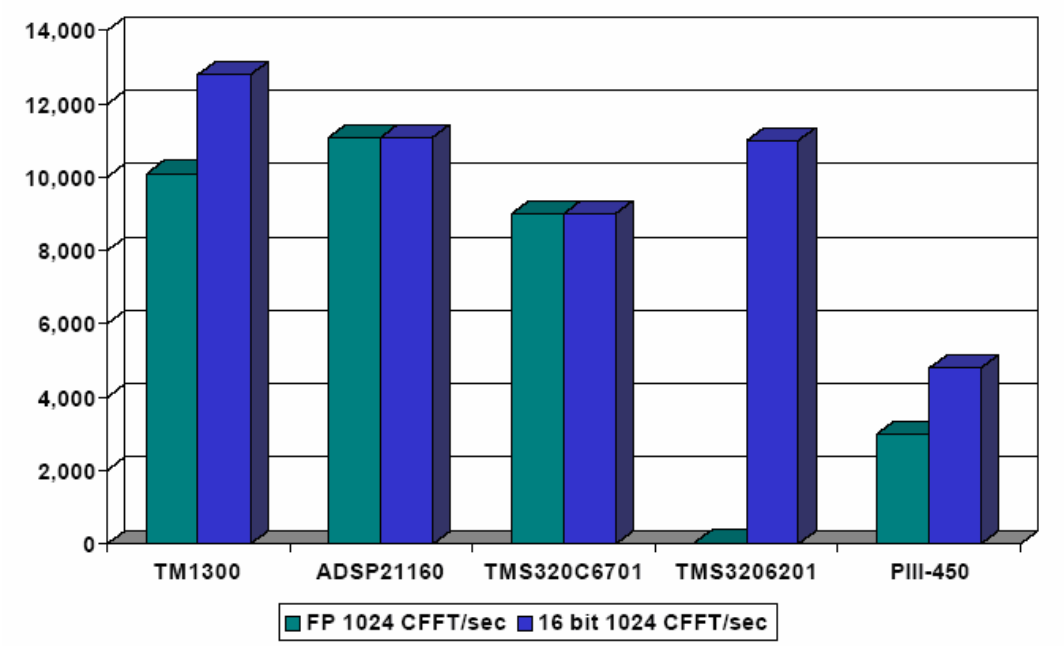


Figura 7-3 Rendimiento de la FFT en diferentes procesadores (año 2001)

Esto resulta en la siguiente tabla que muestra el número de procesadores requeridos para el throughput deseado. Podemos asumir un escalado casi lineal en la mayoría de los diseños excepto para el caso del PIII-450.

Number of Processors Required					
16 bit 1024 CFFT/sec	TM1300	ADSP21160	TMS320C6701	TMS320C6201	PIII-450
5000	1	1	1	1	1
7000	1	1	1	1	2
10000	1	1	2	1	2
20000	2	2	3	2	4
50000	4	5	6	5	NP
70000	6	6	8	7	NP
100000	8	9	11	9	NP

Tabla 7-1 Número de procesadores según el throughput deseado

Para el siguiente análisis tomaremos el procesador ADSP21160 en su configuración estándar, es decir, con una frecuencia de reloj de 100 MHz y en el modo de funcionamiento SIMD (Single Instruction Multiple Data). Para tener una referencia lo compararemos con su versión precedente, el ADSP21060 (denotaremos a ambos como SHARC I y SHARC II respectivamente).

Dos consideraciones serán importantes en el siguiente análisis: el flujo al que los datos son pasados a la memoria global y la arquitectura de bus elegida.

7.2.1 Arquitectura Cluster-Bus

Cada lectura de datos y escritura de resultados requiere una transferencia de 16KB[1] por cada FFT. Para estos dos tipos de procesadores dicha transferencia significará:

SHARC I Bus = 160 MB/s → 16 KB x 160 MB/s = 102 us
SHARC II Bus = 320 MB/s → 16 KB x 320 MB/s = 52 us

Consideremos el caso con mayor flujo de datos que puede llegar a dar la cámara, esto es 160 MB/s como dijimos con anterioridad. Para la entrada de datos en la memoria global y la lectura por cada procesador se requerirán 320 MB/s, lo que excede a los 160 MB/s de ancho de banda que ofrece el cluster-bus del procesador SHARC I. Por tanto, dos buses (es decir, dos tarjetas DSP) serán necesarios para proporciona el ancho de banda suficiente. Para este tipo de placas, un único procesador SHARC I requiere 102 microsegundos para leer los datos y escribir el resultado de la FFT. La FFT de 1K muestras complejas es completada en 410 microsegundos, lo cual significa que no más de cuatro procesadores pueden trabajar al mismo tiempo en el cluster-bus sin interferencia, alcanzándose un throughput total de:

throughput = 9,756 FFTs/sec.

Hay que hacer notar que más procesadores no mejorarían la velocidad final del sistema ya que el procesado estaría limitado por el ancho de banda del bus.

El procesador SHARC II (ADSP 21160) tiene un ancho de banda del bus de datos de 320 MB/s; por tanto, un único bus global sería suficiente. No obstante, este procesador emplea 52 microsegundos para leer los datos y escribir el resultado y 91 microsegundos para realizar la FFT de 1K muestras complejas.

Tiempo de procesado = 18,228 ciclos @ 100 MHz = 182 μs (dos canales)

Análogamente, dos procesadores podrían competir por el ancho de banda del bus, y se podría mejorar la velocidad total del sistema por un factor de 1.6. Si un tercer

[1] Línea = 1024 x 4bytes = 4KB Entrada/Salida Rutina FFT = 4KB x 2 = 8 KB

procesador fuera añadido, la eficiencia total del sistema no mejoraría en nada. Con dos procesadores, el throughput sería:

throughput = 1/ (2 x 52 us) = 19,531 FFTs/sec.

Aunque el SHARC II trabaja a cinco veces la velocidad pico del SHARC original, sólo puede mejorar por un factor de 2 la eficacia del sistema usando la arquitectura cluster bus.

CPUs	SHARC I Cluster	SHARC II Cluster
1	2439	10,989
2	4878	**19,230**
4	**9756**	**19,230**
8	**9756**	**19,230**

Tabla 7-2 FFTs/sec versus número de procesadores (Cluster Bus)

7.2.2 Arquitectura DPLM

En una arquitectura con memorias privadas, la lectura y escritura se harán en memorias locales evitando disputas por el control del bus global. La mejora en la velocidad se puede escalar linealmente con el número de procesadores.

Como se puede ver en la Tabla 7-3, un solo ADSP 21060 proporciona 2439 FFTs/sec, mientras que ocho 19,512 FFTs/sec. Similarmente un ADSP 21160 proporciona 10,989 FFTs/sec, dos 21,390 FFTs/sec y así sucesivamente.

CPUs	SHARC I DPLM	SHARC II DPLM
1	2439	10,989
2	4878	21,930
4	9756	43,780
8	19,512	89,560

Tabla 7-3 FFTs/sec versus número de procesadores (DPLM)

Sin embargo, el número máximo de procesadores en esta configuración coincide con el número de Link Ports disponibles en el procesador maestros. Para el caso del ADSP 21160 es seis con lo que el throughput máximo alcanzable con la arquitectura DPLM es:

throughput = 65,943 FFTs/sec.

7.2.3 Conclusión

Como indica la Figura 7-4, una vez que el cluster-bus ha saturado, nuevos procesadores no mejoran el rendimiento total de la aplicación. En la arquitectura de memorias locales, sin embargo, el ancho de banda es escalable añadiendo buses locales a través de nuevos procesadores.

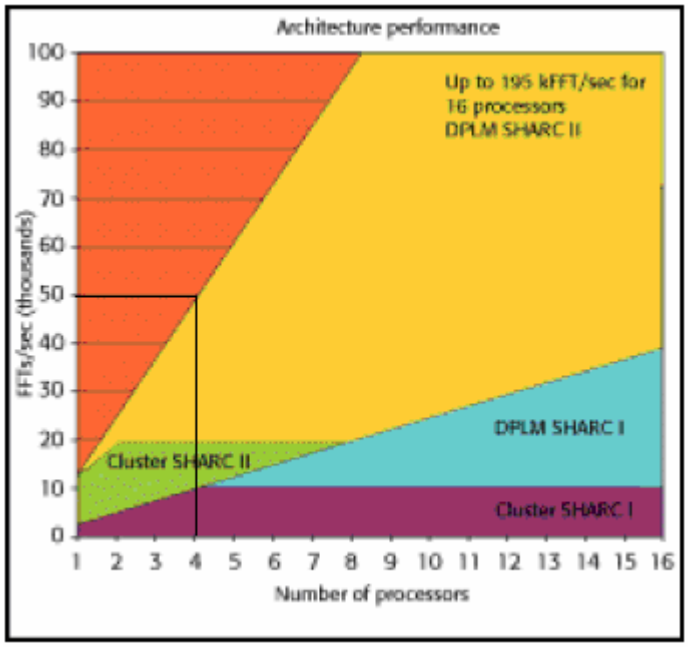


Figura 7-4 Número de FFTs/sec obtenible para las arquitecturas Cluster Bus y DPLM y para los procesadores SHARC I y II

Si volvemos a la aplicación que nos ocupa, vemos que con la arquitectura cluster-bus no es suficiente para procesar todo el flujo de entrada que llega (20K FFTs/sec). Sin embargo con el uso de memorias locales podemos superar sin problemas ese flujo de entrada de datos.

7.3 Resumen

Este capítulo describe cómo la arquitectura elegida del bus de la tarjeta de DSP determina el rendimiento de la aplicación multiprocesador. Se mostraron dos opciones: cluster-bus y DPLM.

Después de cuidadosas consideraciones se consideró la arquitectura DPLM como la solución más conveniente para la tomografía FDOCT debido a que la velocidad de procesado es escalable al número de procesadores utilizados en dicha arquitectura.

Capítulo 8
Diseño Software *(dspFDOCT)*

8.1 Introducción

En este capítulo se propone el diseño software de la aplicación *dspFDOCT* tomando la arquitectura del bus elegida en el capítulo precedente y acorde a los datos proporcionados por la cámara.

El algoritmo de la FFT real en tiempo continuo se ejecuta en los cuatro procesadores ADSP21160 de la tarjeta BittWare HH-PCI que funcionan a 80 MHz. El código se escribe en C y se desarrolla usando el entorno de programación VisualDSP++ de Analog Devices.

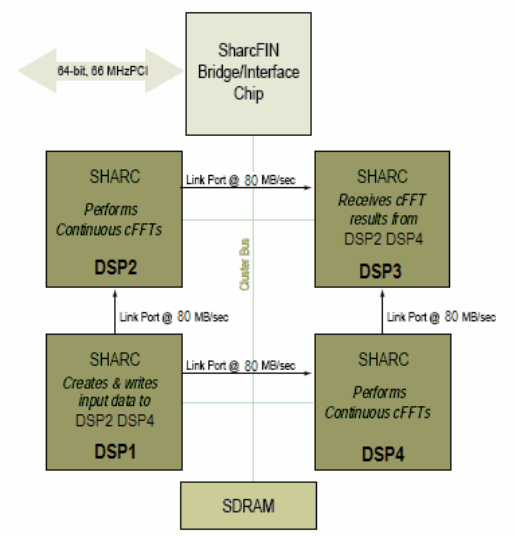


Figura 8-1 Los link ports del SHARC proporcionan una manera fácil de mover datos entre procesadores.

Se usan dos SHARCs (DSP2 y DSP4) para procesar las FFTs de manera continua. Dos SHARCs adicionales son empleados para generar la entrada de datos (DSP1) y recibir los resultados (DSP3) vía link ports durante la prueba.

Los canales link port del SHARC suponen el método más sencillo de mover datos entre procesadores. Como la Figura 8-1 indica, se utilizan dos HammerheadSHARCs para realizar continuamente el procesado FFTs reales. Otros dos HammerheadSHARCs adicionales se utilizan como instrumentos de prueba para generar los datos de entrada y para recibir los resultados (también vía link port). Esos link ports se podrían también utilizar para mover datos desde/a otros dispositivos de entrada-salida tales como FPGAs – el uso de los otros HammerheadSHARCs de la tarjeta HH-PCI se tomó simplemente por simplicidad.

Puesto que el procesador SHARC es capaz de soportar accesos DMA de entrada-salida a memoria interna[1], se utilizan búferes de entrada y salida para implementar un esquema "ping-pong" (Figura 8-2). Usando este esquema de ejecución, la memoria interna del procesador debe almacenar el código de la rutina FFT así como un dos búferes de entrada y salida. Se puede verificar que cuando estos búferes son usados, se mejora dramáticamente el rendimiento final de la aplicación.

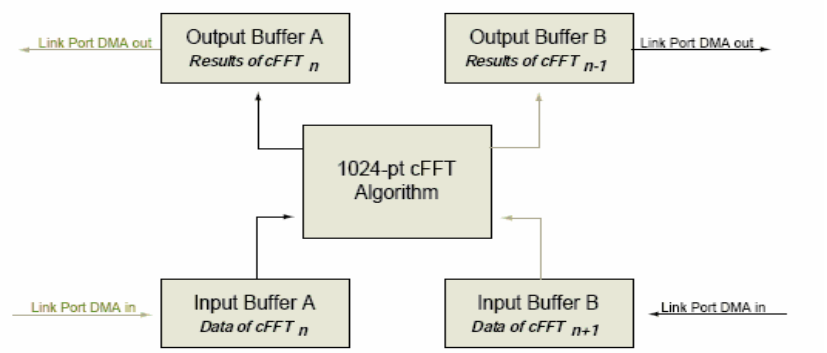


Figura 8-2 Se usaron dos búferes de entrada-salida para implementar el esquema ping-pong.

> *Mientras que la rutina FFT está procesando los datos del buffer de entrada A y escribiendo los resultados al buffer de salida A, los motores DMA están haciendo dos cosas: moviendo los datos para la siguiente rutina FFT en el buffer de entrada B desde un link port al tiempo que los resultados de la rutina FFT anterior están siendo escritos en el otro link port desde el buffer B de salida. Después de completarse la FFT y ambas operaciones DMAs, los búferes ping-pong se intercambian; la FFT procesa ahora los búferes B mientras que los motores DMAs trabajan sobre los búferes A*

[1] Estos accesos se ejecutan en segundo plano

8.2 Simplificación

Por simplicidad se va a eliminar el procesador HH3 de la Figura 8-1 que recibe los resultados de los dos esclavos.

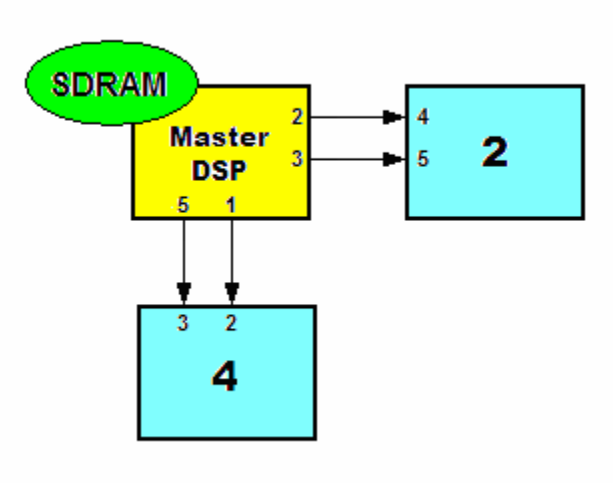


Figura 8-3 Configuración de la tarjeta HH-PCI

En la topología jerárquica mostrada en la Figura 8-3, el procesador maestro lee dos líneas[2] de la SDRAM y las transmite a cada procesador esclavo a un ritmo preestablecido. Este período viene determinado por el tiempo de procesado ya que el ritmo de paso de datos de entrada no debe superar nunca al de consumo de los mismos. Para la comunicación maestro-esclavos se emplean dos enlaces link-ports.

8.3 Versiones Preliminares

Antes de diseñar todos los detalles de implementación de la versión final de la aplicación *dspFDOCT*, se considera los casos siguientes basados en las rutinas FFT que proporciona Analog Devices (sin conversión de datos de entrad y/o salida):

- algoritmo FFT complejo
- algoritmo FFT real

[2] El procesado FFT en el ADSP21160 será óptimo cuando se disponga de dos líneas a la entrada.

8.3.1 FFT Compleja

En primer lugar, se estima el tiempo de procesado en un contexto más simple que la aplicación *dspFDOCT*. Para ello en esta versión se toman los píxeles como "números complejos"

Los procesadores DSP de la tarjeta empleada trabajan a una frecuencia de 80 MHz en vez de 100 MHz como se tomó en todos los cálculos comparativos del capítulo anterior. El procesado FFT de 1024 muestras complejas durará entonces 114 us (228 us para dos FFTs simultáneas) en vez de 91 us (181 us para dos FFTs simultáneas).

Todo esto se traduce para N procesadores en una arquitectura de memorias locales en el siguiente throughput total:

$$\text{Tiempo de Procesado} = 18{,}228 \text{ ciclos @ } 80 \text{ MHz} = 228 \text{ µs (dos canales)}$$

$$\text{Throughput} = N \times \frac{1}{228\mu s / 2} = N \times 8777 \text{ FFTs/sec.}$$

En la Figura 8-1 se aprecia que el número máximo de procesadores esclavos con conexión directa al procesador maestro es dos en vez de seis como en teoría permitiría la configuración de memorias locales, por tanto el throughput máximo alcanzable es:

$$\text{Throughput} = 2 \times 8777 \text{ FFTs/sec} = 17{,}554 \text{ FFTs/sec}$$

Se puede concluir que el flujo de entrada de datos máximo que podemos llegar a procesar con la tarjeta HH-PCI es superior al que vamos utilizar en la FDOCT (10 Klíneas/sec.) pero no supera al que en teoría podría llegar a entregar la cámara: 20 Klíneas/sec.

8.3.2 FFT Real

A continuación nos ajustaremos un poco más a la aplicación final y se tomarán píxeles reales de 32 bits con lo que podemos efectuar el procesado FFT real (simetría a la salida). No obstante no se efectuará ningún tipo de compresión de datos a la entrada ni se calculará el módulo de los de salida.

Un solo procesador DSP emplea **22,965** ciclos en efectuar la FFT de 1024 números reales en punto flotante, es decir, 287 us (3.5 KHz) proporcionando un throughput de hasta 7000 FFTs/sec por procesador (procesado de dos líneas al mismo tiempo).

Two Lines Period = 287 us (3.5 KHz)

Throughput = 2 × Two Lines Frequency = 7000 FFTs/sec.

En principio con dos procesadores esclavos (Figura 8-3) se puede alcanzar un throughput de procesado suficiente para el flujo de datos requerido en la tomografía FDOCT (10 Klíneas/sec.):

Throughput = 2 × Two Lines Frequency × N

Throughput Total = 14,000 FFTs/sec.

8.4 Versión Final

El código DSP correspondiente a la versión precedente sigue siendo adecuado para la versión final de *dspFDOCT*; la única diferencia es que habrá una conversión de datos antes y después el procesado debido al formato de datos de la cámara.

8.4.1 Código DSP

Comenzamos con el pseudocódigo del procesador maestro.

Master DSP Code (VisualDSP++)

```
FFT_SIZE   = 1024            LINE_SIZE = 1024 pixels
FRAME_SIZE = 32 lines        LINE_FREQ = 5 KHz
sdram1 = 0x800000            sdram1'= sdram1 + Total_Size / 2
line2[2][LINE_SIZE]          line4[2][LINE_SIZE]

DO

        WHILE (!input)
        input = 0

        Read_line(line2[sbuf], sdram1  + offset, BusMaster)        // first 2 lines
        Read_line(line4[sbuf], sdram1' + offset, BusMaster)        // 2 lines in middle
        Wait for timer complete

        DO
                Read_line(line2[sbuf], sdram1  + offset, BusMaster)
                Read_line(line4[sbuf], sdram1' + offset, BusMaster)

                Transmit_line(line2[lbuf], HH2_link_ports)
                Transmit_line(line4[lbuf], HH4_link_ports)

                offset += LINE_SIZE
                Wait for timer complete

        WHILE (1 < offset < FRAME_SIZE/2)

        Transmit_line(line2[lbuf], HH2_link_ports)
        Transmit_line(line4[lbuf], HH4_link_ports)                 // last transmission

        frames_count ++
        output = 1

WHILE (!done);
```

Algoritmo 8-1 Código principal del DSP maestro

Pseudocódigo de cada uno de los procesadores esclavo.

Slave DSP Code (VisualDSP++)

```
FFT_SIZE   = 1024            LINE_SIZE = 1024 pixels
FRAME_SIZE = 32 lines        rcvd[FRAME_SIZE/2][LINE_SIZE]

FOR (;;)

   Receive_line(rcvd[rxbuf], local_link_ports)
   Wait for reception complete

   DO

       Receive_line(rcvd[rxbuf], local_link_ports)
       Process_line(rcvd[lbuf])
       Wait for reception complete

   WHILE (rxpkts < Frame_Size/2)

   Process_line(rcvd[lbuf])
```

Algoritmo 8-2 Código principal del DSP esclavo

8.4.2 Conversión de Formatos de Datos

La cámara line-scan proporcionará 1024 píxeles de 16 bits, por lo que se podría empaquetar sin compresión una línea en un array de 512 palabras de 32 bits (tanto la tarjeta DSP como el PC están diseñados para trabajar en este formato de datos).

1 linea = 1024 píxeles of 16 bit = 512 palabras de 32 bits
2 lines = 1024 palabras de 32 bits (4KB)

Debido al hecho de que la función de la FFT real está optimizada para trabajar con dos líneas al mismo tiempo, se leerá un array de 1024 palabras de 32 bits, es decir, dos líneas en cada iteración del bucle principal. El algoritmo 831 muestra las instrucciones de la función de *Process_line* () ejecutada por los procesadores esclavos para realizar las conversiones oportunas.

```
Process_line( lbuf )
{
        array1024 → twolines512
        line512_line1024 (short int)
        int2float → data_one_real
        line512_line1024 (short int)
        int2float → data_two_real

       rfftf_2( data_one_real[], rttf1_im[],
                data_two_real[], rttf2_im[],
                FFT_SIZE )

       fftf_magnitude( data_one_real, rttf1_im,
                       rspectrum_1, N_FFT, 2 )

       fftf_magnitude( data_two_real, rttf2_im,
                        rspectrum_2, N_FFT, 2 )

        rspectrum_1 → float2int
        rspectrum_2 → float2int
        twolines512_array1024
}
```

Algoritmo 8-3 Función del DSP maestro Process_Line()

La Figura 8-4 muestra gráficamente la pre- y post-conversión ejecutada en la función *Process_line*().

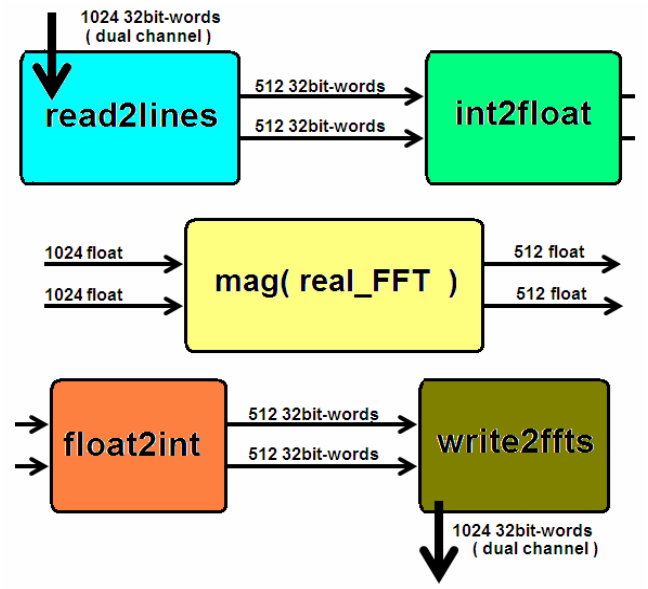


Figura 8-4 Pre & post-procesado de datos

La latencia y el flujo de datos hacia los procesadores auxiliares definidos a continuación serán empleados como parámetros de referencia para la comparación del algoritmo de esta aplicación con aquel de la aplicación *parallelFFT* que será estudiado en el capítulo 11.

Data transfer per Two Lines Period = N × 4 KB

Data Flow = N × 4 KB ÷ Two Lines Period
Latency = Two Lines Period (N × 2 lines)

8.4.3 Tiempo de Procesado de Línea (SISD)

Sorprendentemente un solo procesador DSP emplea **94,796** ciclos en computar la FFT simultánea de 2 líneas de 1024 números reales incluyendo el tratamiento de datos descrito anteriormente. Podría parecer demasiado en un primer momento pero, no sorprende tanto si se considera todas las acciones suplementarias que se realizan para traducir los datos del protocolo CameraLink a los procesadores de 32 bits:

- desempaquetado de 16 a 32 bits: desplazamientos y funciones AND
- conversión de entero a flotante y viceversa
- copia en el orden correcto de las dos líneas como variables de entradas a la función que realiza la FFT

No obstante se puede disminuir el tiempo anterior y posterior al procesamiento aprovechando la ventaja SIMD del procesador ADSP21160. Consistiría en tratar dos píxeles en paralelo durante el bucle *FOR* (ver código del Apéndice B) gracias a la disponibilidad de dos unidades de ejecución independiente (doble núcleo).

8.4.4 Tiempo de Procesado de Línea (SIMD)

Gracias a la capacidad SIMD se consigue reducir el cómputo de la FFT simultánea de 2 líneas de 1024 números reales a **63,943** ciclos (caso peor).

Sin embargo, este resultado demuestra que estamos aun lejos de nuestras aspiraciones ya que sólo el pre-procesado requiere 26,632 ciclos superando incluso al propio tiempo de procesado FFT (**22,965** ciclos como vimos en el apartado anterior). El post-procesado también es importante llegando a los 10,000 ciclos.

De cualquier modo esos 63,943 ciclos ya no se pueden reducir más, incluso si programamos a bajo nivel. La conversión temporal corresponde a 800 us en el caso peor y siempre que la capacidad SIMD esté habilitada.

Two Lines Period = 800 us (1250 Hz)

Throughput = 2 × Two Lines Frequency = 2500 FFTs/sec**.**

Si tenemos N esclavos podemos alcanzar un throughput de procesado de:

Throughput = N × 2500 FFTs/sec.

8.5 Resumen

Comparando el resultado de la aplicación final *dspFDOCT* con el que se podría estimar a partir de las versiones preliminares, se observa que desgraciadamente se van a necesitar más procesadores esclavos de los que la tarjeta HH-PCI dispone.

Para poderla usar tal y como está (sin procesadores adicionales) se tendrá que ceder un poco en las especificaciones finales en cuanto a *real-time* se refiere y nos satisfacernos con una frecuencia de 5000 FFTs/sec. Esto se alcanza cuando se dispone de dos procesadores esclavos:

N = 2 esclavos (una tarjeta HH-PCI)

throughput = 5000 FFTs/sec.	Latency = 800 us (2 × 2 lines)
→ Lines Period = 800 us	Data Flow = 2 × 4 KB ÷ 800 us = 10 MB/s

Efectivamente éste no es un buen resultado ya que no alcanzamos las 10,000 FFTs/sec mínimas para poder proporcionar *real-time* a la tomografía FDOCT. Sin embargo, nos conformaremos con él debido a la simplicidad de la idea y a que es fácilmente escalable a más procesadores en una arquitectura de memorias locales.

Capítulo 9
Resultados y Posibles Mejoras (*dspFDOCT*)

9.1 Introducción

En este capítulo se ven los resultados reales de la aplicación *dspFDOCT* diseñada y analizada en el capítulo anterior. También se analizan algunas mejoras que podrían aplicarse.

9.2 Medida del tiempo de procesado

La temporización del procesado de línea se hace off-line, es decir, se registra para cada línea. Una vez finalizada la ejecución del programa en la tarjeta DSP, el PC calcula la media de estos datos (la aplicación final lo muestra con el indicador llamado 2*Lines-Process Average).* El código de la aplicación *dspFDOCT* ha sido escrito bajo la plataforma gráfica de programación de LabView, la cual permite elaborar una aplicación "relativamente" rápida.

No obstante hemos de comentar que fue necesaria la utilización de Microsoft Visual C++ para generar una librería donde se llamara de forma adecuada a las funciones de control de la tarjeta DSP proporcionadas por Bittware (capítulo 13).

En la Figura 9-1 se puede observar la duración del procesado de línea cuando tenemos una imagen saturada (DC).

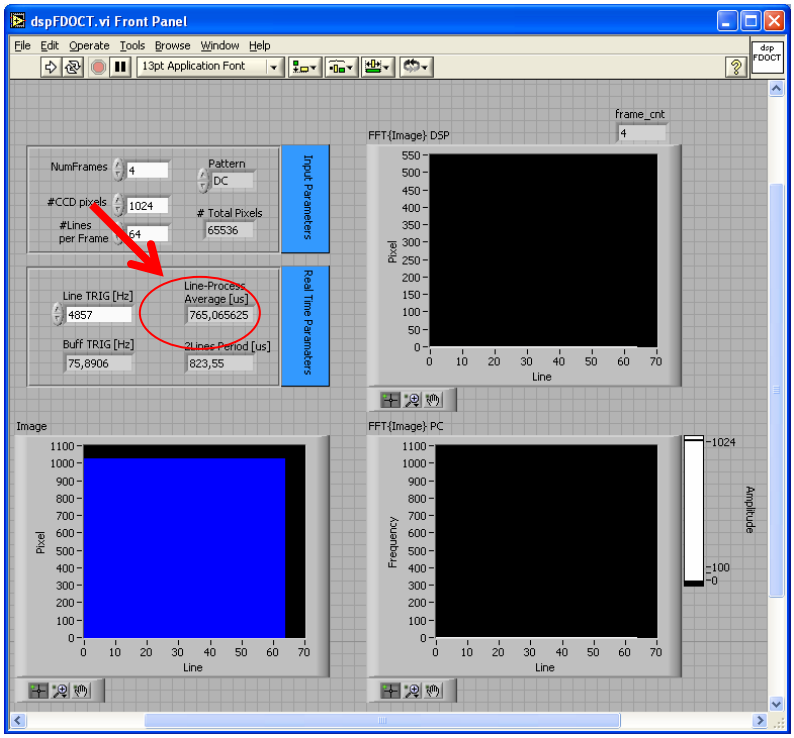


Figura 9-1 Interfaz de usuario de la aplicación final (componente DC)

En la Figura 9-2 se observa cómo responde el programa cuando tenemos una imagen en la que barremos algunas frecuencias significativas hasta llegar a la de Nyqyuist.

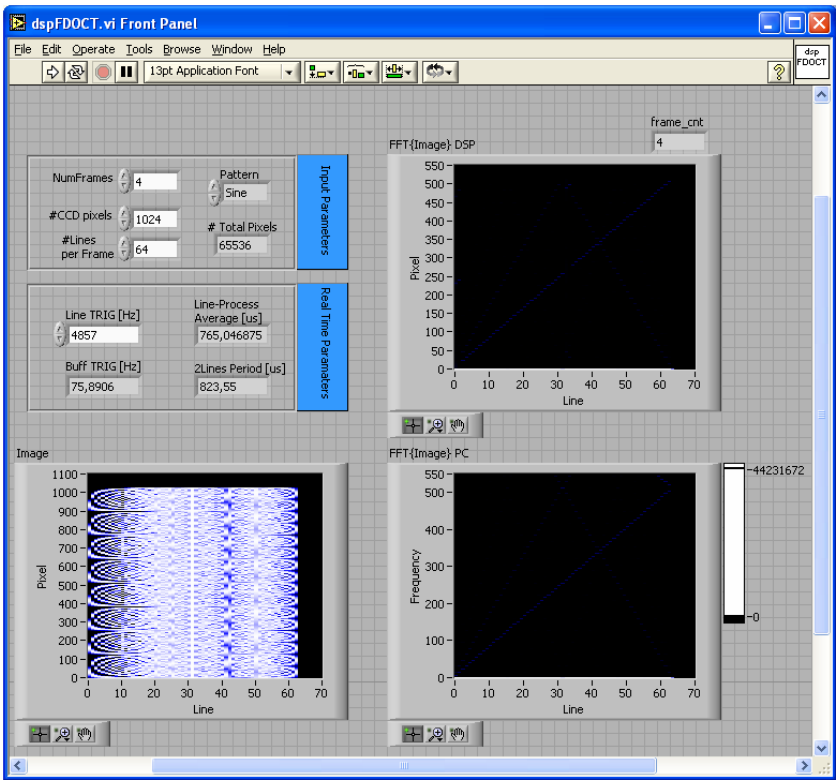


Figura 9-2 Interfaz de usuario de la aplicación final (patrón sinusoidal)

En dicha representación se ha tomado la mitad de los 1024 píxeles disponibles por línea debido a que los de la otra mitad tomarán el mismo valor (simetría de la transformada de vectores reales).

Para el valor máximo de 5000 Hz del Line Trigger obtuvimos errores severos en la transformada de la imagen. Por tanto, se redujo un poco dicha frecuencia de línea hasta alcanzar los 4857 Hz, frecuencia a la que se obtuvo un resultado aceptable.

9.2.1 Diferencia entre la Predicción y los Resultados

Un examen más detallado de los resultados predichos en el capítulo anterior y las implementaciones revelaron que la predicción descuidó cualquier sobrecarga en el acceso directo de memoria (DMA). La configuración de los DMAs asociado a cada link port, la manipulación de interrupciones y la comprobación del estado de los DMAs para ver si han terminado agrega una sobrecarga del 10% aproximadamente que explica esta diferencia de tiempos.

Por conveniencia el código de gestión del DMA fue escrito directamente en C, y puesto que la mejora de resultados de la prueba total sería pequeña (el 10% en el mejor de los casos), no se hizo ninguna tentativa de reducir al mínimo estas sobrecargas.

9.3 Nuevo Procesador DSP

Se analiza aquí cómo la actualización del procesador TigerSHARC podría mejorar el cálculo de la FFT de 1024 píxeles en punto flotante:

- La principal ventaja del TigerSHARC es que procesa la FFT en menos tiempo que los procesadores HammerheadSHARC (15.64 µs son equivalentes a 64.000 FFTs/sec. lo cuál es suficiente para la tomografía FDOCT).

- En el caso que se esté utilizando los datos proporcionados por la cámara line-scan, se necesitará convertir el formato de los datos en coma flotante. El TigerSHARC es más eficiente en este sentido que el ADSP21160; podríamos

ahorrar muchos ciclos de reloj con los nuevos tipos de datos **int2x16** y las funciones asociadas para el tratamiento de píxeles en palabras de 32 bits:

```
int     a, b;          // 16 bit pixels
float   x, y;          // 32 bit intermediate values
                int2x16        c = compact_to_i2x16(a, b);    // packed pixels
                x = (float)    expand_low_of_i2x16 (c);       // expansion
                y = (float)    expand_high_of_i2x16 (c);      // expansion
```

9.4 Nuevas Configuraciones de la HH-PCI

Si nos abstraemos por un momento del hardware que disponemos y suponiendo que podamos diseñar una tarjeta con un número flexibles de procesadores ADSP21160, deberíamos tomar aquella con N = 4 esclavos para poder alcanzar el throughput requerido en la FDOCT:

$$\text{throughput} = N \times 2500 \text{ FFTs/sec.} = 10{,}000 \text{ FFTs/sec.}$$

Sin embargo el conexionado interno de la tarjeta HH-PCI nos permite tener hasta N = 2 esclavos directos (Figura 9-3) obteniendo tan sólo 5000 FFTs/sec.

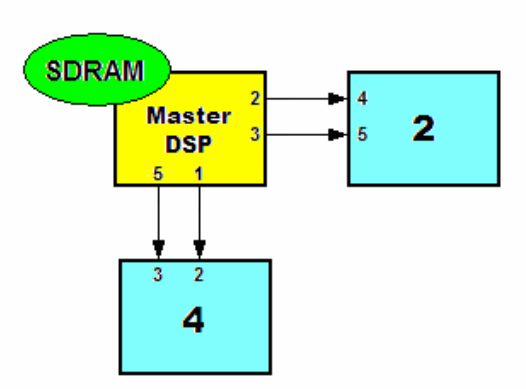


Figura 9-3 Configuración de memorias locales en la tarjeta HH-PCI (N=2)

En esta sección se analiza las posibilidades y limitaciones de la arquitectura de bus que hemos elegido. Ninguna de las configuraciones es implementada porque no se estimó

oportuno aunque se podría haber hecho sin mucho esfuerzo gracias a la escalabilidad del programa.

9.4.1 Expansión de la Arquitectura de Bus

La solución que se propone será conectar al procesador maestro nuevos procesadores a través de link ports externos de este modo se podría utilizar los cables que hemos adquirido para implementar el *Line Grabber* (recordamos que dada la dimensión de este proyecto bastará con simular la adquisición de líneas mediante la lectura en la SDRAM de manera periódica).

Como sabemos el procesador maestro dispone de hasta seis puertos libres. No obstante el Link-Port 0 está dedicado para el bus PMC+ (apéndice A), con lo que no se podrá usar para conectar un nuevo procesador esclavo.

Parecerá por tanto que el número máximo de procesadores esclavos es N = 3 lo que se consigue con la configuración de la Figura 9-4.

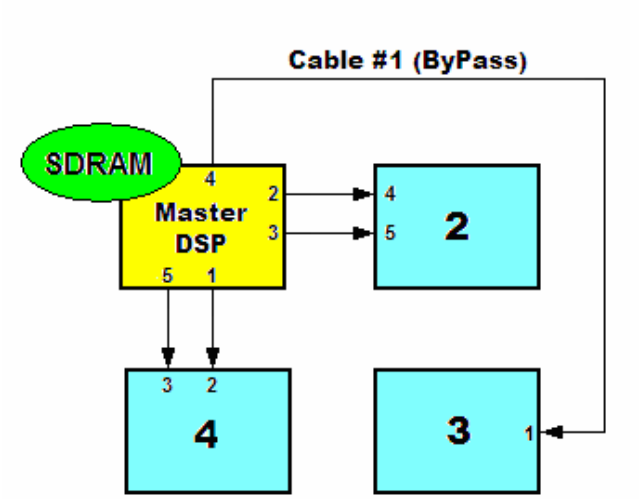


Figura 9-4 Configuración de memorias locales en la tarjeta HH-PCI (N=3)

N = 3 esclavos (una tarjeta HH-PCI)

throughput = 7,500 FFTs/sec.	Latency = 133 us (3 × 2 lines)
→ Lines Period = 800 us	Data Flow = 3 × 4 KB ÷ 800 us = 15 MB/s

Sin embargo, si dedicamos un solo link-port a los procesadores HH2 y HH4 en vez de dos, dispondremos de dos enlaces más en el procesador maestro con lo que podemos elevar el número de procesadores esclavos al máximo (N = 5).

N = 5 esclavos (dos tarjetas HH-PCI)

throughput = 12,500 FFTs/sec.
→ Lines Period = 800 us

Latency = 80 us (5 × 2 lines)
Data Flow = 5 × 4 KB ÷ 800 us = 25 MB/s

Recapitulando, los procesadores esclavos disponibles serían:

- dos procesadores ADSP21160 en la misma tarjeta HH-PCI con conexionado interno; este es el caso del HH2 y HH4

- un procesador ADSP21160 en la misma tarjeta HH-PCI con conexionado externo; este es el caso del HH3

- dos procesadores ADSP21160 en otra tarjeta HH-PCI o similares con puertos Link-Port accesible desde el exterior

Esta última solución nos permitirá otorgar *real-time* a la tomografía FDOCT aunque para ello tengamos que adquirir una nueva tarjeta DSP. En el mercado encontramos dos versiones que podríamos utilizar igualmente:

- la tarjeta EZ-KIT Lite de Analog Devices con dos procesadores ADSP21160

- la versión de la HH–PCI con sólo dos procesadores ADSP21160 (Figura 9-5)

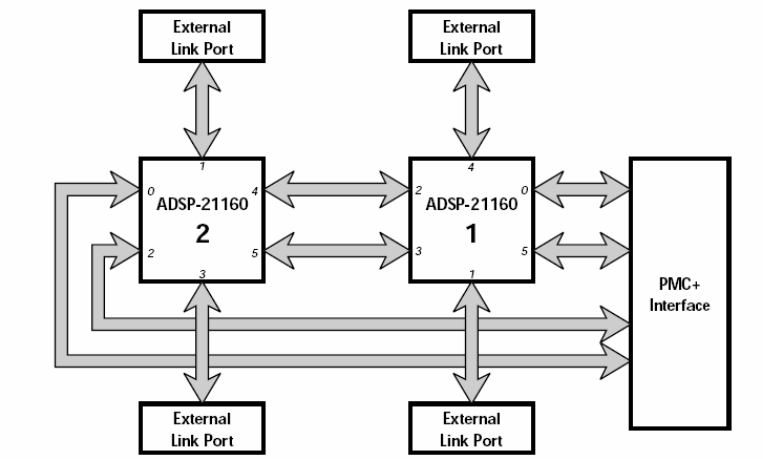


Figura 9-5 Diagrama de Bloques de las Conexiones Link Port: Tarjeta con dos Procesadores

Es cierto que podríamos alcanzar hasta 12,500 FFTs/sec, pero se considera que no compensará el gasto económico que supone la compra de una tarjeta que es obsoleta con respecto al nivel tecnológico en el que se encuentran las tarjetas DSP actuales, si bien el consumo de energía sería del orden de 10 veces menor.

9.4.2 Temporización del Link Port HH1-HH3 (maestro-esclavo)

En la solución anterior no se podrá utilizar los cables Link Port a su máxima velocidad por el hecho de ser externos a la placa ya que de otro modo no se puede garantizar la fiabilidad en las transferencias. El fabricante nos recomienda trabajar a la mitad de la frecuencia máxima de funcionamiento para la que han sido diseñados (40 MHz).

Hay que remarcar que hasta ahora para conectar los procesadores esclavos al procesador maestro no sólo usábamos dos enlaces Link-Port sino que también a su máxima frecuencia de trabajo diseñads (80 MHz).

Por tanto, ¿es posible que con el uso de un solo Link Port la recepción de datos sea crítica en la eficiencia de la aplicación final? Podría ocurrir una ralentización final debida al hecho de que la comunicación con los procesadores esclavos estaría basada ahora en un solo enlace link-port y con una frecuencia de transmisión de 40 MHz.

Si echamos un vistazo al pipeline en el código de los procesadores esclavos (Apéndice E), vemos que la recepción y el procesado son dos actividades paralelas con lo que la

ralentización de las comunicaciones podría penalizar en la eficiencia final. No obstante, se puede comprobar que el tiempo necesario para la recepción de datos para el procesador maestro es siempre inferior al procesado de éstos (800 us).

Link Port Data Reception = 8160 ciclos × 12.5 ns = 102 us (caso peor)

Podemos garantizar así que el uso de un único Link-Port para conectar procesadores esclavos externos no planteará ningún problema inesperado.

9.4.3 Pruebas de la librería Link-Port

En esta sección se temporizaron para el procesador maestro las funciones de gestión de los canales Link-Ports incluidas en la librería *linkdma.h* de Bittware. Existe 2 funciones:

- función de transmisión al procesador esclavo por un canal DMA
- función de recepción por otro canal DMA desde el mismo procesador esclavo

Se comprobó que la transferencia de una línea de 4 KB entre dos procesadores usando la función llamada link_dma_noint_xmt requirió 102 us (8160 ciclos).

9.5 Resumen

En este capítulo se mostraron los resultados reales obtenidos mediante el programa *dspFDOCT* diseñado teóricamente en el capítulo anterior para una sola tarjeta Bittware. Desgraciadamente no se alcanzó la velocidad de procesado que la FDOCT de tiempo real requiere (10,000 FFTs/sec) no obstante se propuso dos soluciones:

- Añadir más procesadores DSP aunque para ello habría que adquirir una nueva tarjeta DSP.

- Cambiar la tarjeta DSP por otra más moderna, por ejemplo alguna con la versión TigerSHARC para el procesador.

PARTE III:
Aplicación *parallelFFT*

Capítulo 10
Implementación Paralela de FFTs

10.1 Introducción

El rápido avance en paralelizabilidad de los nuevos procesadores, tales como los de la familia SHARC® de Analog Devices, requiere encontrar formas eficientes de implementar de manera paralela algunos de los algoritmos mas usados. Este capítulo explica cómo podría funcionar una implementación paralela del algoritmo FFT en sistemas multiprocesador SHARC; la FFT de 256 puntos se utiliza como ejemplo específico pero la idea y las matemáticas asociadas pueden aplicarse igualmente a otros tamaños (no más pequeño que 16 puntos).

Se verá pues, un algoritmo de reestructurado que divide la FFT en células más pequeñas que puedan entonces ser paralelizadas. En el caso de 256 puntos, la FFT puede ser dividida en 16 FFTs de 16 puntos cada una. Si hiciéramos la FFT de 512 puntos, tendríamos que hacer 16 FFTs de 32 puntos cada una (y después, 32 FFTs de 16 puntos cada una). Estas diferencias implican que sería difícil escribir un código para paralelizar la FFT de tamaño genérico; aunque el algoritmo que se propone es genérico y se aplica igualmente de bien a todos los tamaños de la FFT, el código no lo es, y debe ser configurado a mano para cada número de puntos para poder aprovechar una completa optimización.

10.2 Algoritmo

10.2.1 Standard FFT Raíz-2

La Figura 10-1 muestra la implementación estándar de la FFT radix-2 de 16 puntos, después de que la entrada haya sido reordenada (*bit reversing*). Tradicionalmente en este algoritmo, las etapas en 1 y 2 se realizan con el requerido *bit reversing* en el mismo bucle optimizado (puesto que estas dos etapas no requieren ninguna multiplicación, sólo sumas y restas). Cada uno de las etapas restantes se hace generalmente combinando en los mismos grupos las mariposas que comparten los mismos factores de multiplicación (así necesitamos leer estos factores *twiddle* una sólo vez por cada grupo).

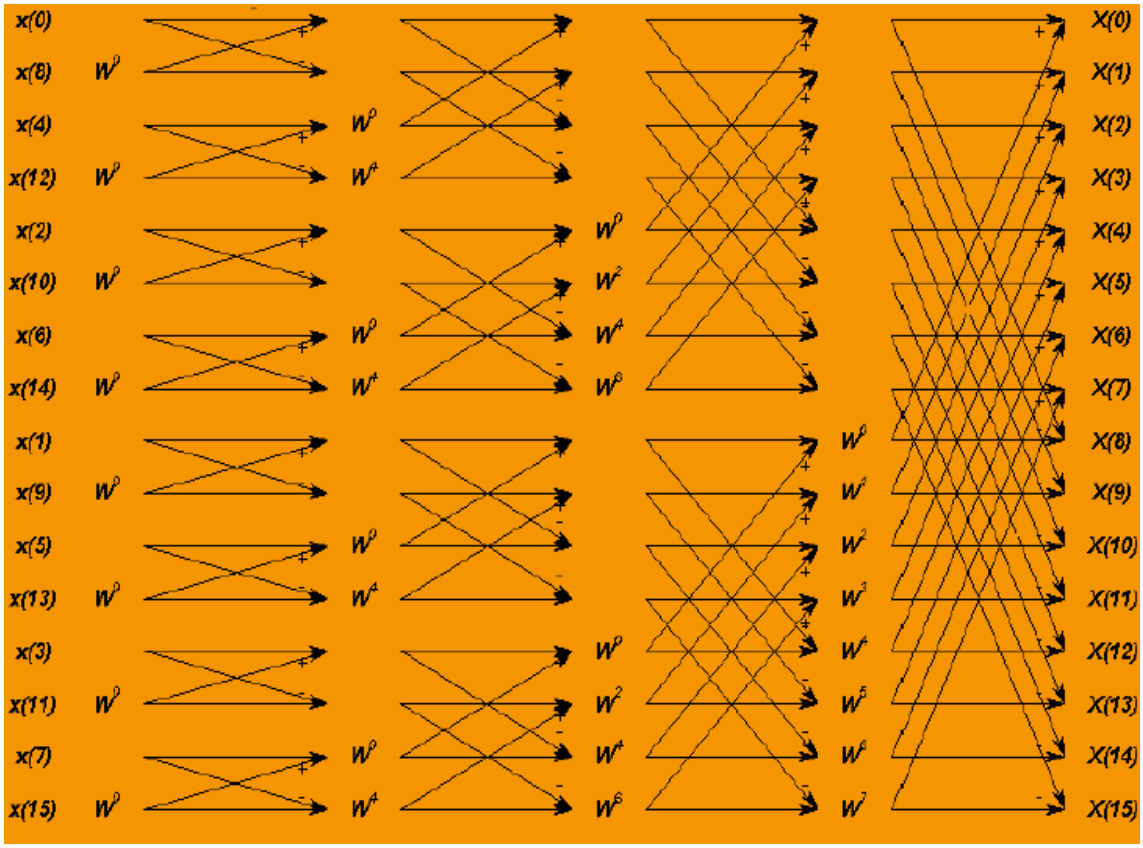


Figura 10-1 Standard FFT de 16-point radix-2

Viendo el diagrama de la Figura 10-1, nos gustaría romper la FFT en al menos cuatro procesos paralelos. Podemos hacer esto simplemente combinando los datos en bloques, de cuatro puntos a la vez, es decir:

1st block = {x(0), x(8), x(4), x(12)}
2nd block = {x(2), x(10), x(6), x(14)}
3rd block = {x(1), x(9), x(5), x(13)}
4th block = {x(3), x(11), x(7), x(15)}

Estos grupos no tienen ninguna interdependencia entre ellos y serán paralelizables muy fácilmente para las primeras dos etapas del FFT. Después estaremos en apuros ya que el paralelismo no es posible. En este momento, sin embargo, podríamos reordenar los datos en diferentes bloques para asegurar que en el resto del proceso los nuevos bloques no se interfirieran entre ellos y, así, poder ser paralelizados. Un examen cuidadoso demuestra que el reordenamiento requerido es una operación de interleaving (o de-interleaving), siendo los nuevos bloques:

1st block = {x(0), x(2), x(1), x(3)}
2nd block = {x(8), x(10), x(9), x(11)}
3rd block = {x(4), x(6), x(5), x(7)}
4th block = {x(12), x(14), x(13), x(15)}

Otra manera de ver estos nuevos bloques es como la matriz 4x4 transpuesta (donde cada bloque define las filas de la matriz). Por supuesto, hay un significativo efecto colateral - después del reordenamiento de los datos para que las últimas etapas sean paralelas, los datos de salida no estarán en el orden correcto. Podemos compensar esto comenzando con un orden diferente al bitreverse con el que habíamos comenzado antes, pero dejaremos este detalle para más adelante con un análisis matemático más riguroso.

En este momento, el análisis de la FFT de 16 puntos FFT parece sugerir que, dado en general una FFT de N puntos, podemos verlo en dos dimensiones como una matriz $\sqrt{N} \times \sqrt{N}$ de datos y procesar en paralelo por filas o columnas, entonces transponer la matriz y procesar de nuevo en paralelo por filas o columnas. Otro requisito que se deduce de este análisis es que N debe ser un cuadrado perfecto. Por ahora podemos suponer esta hipótesis, aunque no obstante será estudiada más adelante. En este momento nos limitaremos por tanto a la FFT de 256 puntos que por suerte es cuadrado de 16, $256 = 16^2$.

10.2.2 Matemáticas del Algoritmo

La siguiente notación será utilizada:

N = número de puntos en la FFT original (256 en nuestro ejemplo),

$M = \sqrt{N}$,

$\hat{x}$ = Transformada Discreta de Fourier (abreviado como DFT) de x.

Ahora, dada una señal x,

$$\hat{x}(n) = \sum_{k=0}^{N-1} x(k) e^{\frac{-2\pi i n k}{N}} = \sum_{m=0}^{M-1} \sum_{l=0}^{M-1} x(Ml+m) e^{\frac{-2\pi i n (Ml+m)}{N}} =$$

$$\sum_{m=0}^{M-1} e^{\frac{-2\pi i n m}{N}} \sum_{l=0}^{M-1} x(Ml+m) e^{\frac{-2\pi i n l}{M}} = \sum_{m=0}^{M-1} e^{\frac{-2\pi i n m}{N}} \hat{x}_m(n)$$

donde:

$$x_m := x(Ml + m) \qquad (1)$$

y $\hat{x}_m$ es su función DFT de M puntos. Ahora, veamos como modificar el índice de salida n para obtener una matriz $M \times M$ (es decir, $n = Ms + t,\ 0 \le s,t < M - 1$). De este modo,

$$\hat{x}(Ms+t) = \sum_{m=0}^{M-1} e^{\frac{-2\pi i (Ms+t) m}{N}} \hat{x}_m(Ms+t) =$$

$$\sum_{m=0}^{M-1} e^{\frac{-2\pi i s m}{M}} e^{\frac{-2\pi i t m}{N}} \hat{x}_m(t)$$

debido a que $\hat{x}_m$ es una función DFT de M puntos, es periódica de periodo = M. Así,

$$\hat{x}(Ms+t) = \sum_{m=0}^{M-1} e^{\frac{-2\pi i s m}{M}} x_t^*(m) = \hat{x}_t^*(s) \qquad (2)$$

donde:

$$x_t^*(m) := e^{\frac{-2\pi i t m}{N}} \hat{x}_m(t) \qquad (3)$$

y $\hat{x}_t^*$ es su función DFT de M puntos.

10.2.3 Implementación del Algoritmo

Las ecuaciones (1), (2), y (3) muestra cómo calcular la DFT de *x* usando los pasos siguientes (volvemos a las especificaciones de nuestro ejemplo de *N*=256, *M*=16):

1. Ordenar los 256 puntos de la entrada de datos *x*(n) no linealmente, sino considerando una matriz 16x16:

$$\begin{bmatrix} x(0) & x(1) & x(2) & \cdots & x(15) \\ x(16) & x(17) & x(18) & \cdots & x(31) \\ x(32) & x(33) & x(34) & \cdots & x(47) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x(240) & x(241) & x(242) & \cdots & x(255) \end{bmatrix}$$

2. Usando la ecuación (1), rescrita como:

$$\begin{bmatrix} x_0(0) & x_1(0) & x_2(0) & \cdots & x_{15}(0) \\ x_0(1) & x_1(1) & x_2(1) & \cdots & x_{15}(1) \\ x_0(2) & x_1(2) & x_2(2) & \cdots & x_{15}(2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_0(15) & x_1(15) & x_2(15) & \cdots & x_{15}(15) \end{bmatrix}$$

3. Ahora calculamos la FFTs de cada columna en paralelo obteniendo:

$$\begin{bmatrix} \hat{x}_0(0) & \hat{x}_1(0) & \hat{x}_2(0) & \cdots & \hat{x}_{15}(0) \\ \hat{x}_0(1) & \hat{x}_1(1) & \hat{x}_2(1) & \cdots & \hat{x}_{15}(1) \\ \hat{x}_0(2) & \hat{x}_1(2) & \hat{x}_2(2) & \cdots & \hat{x}_{15}(2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \hat{x}_0(15) & \hat{x}_1(15) & \hat{x}_2(15) & \cdots & \hat{x}_{15}(15) \end{bmatrix}$$

4. Multiplicamos por la matriz *exp (-2π tm / 256) 0≤ t,m ≤ 15)* obteniendo

$$\begin{bmatrix} \hat{x}_0(0)e^{\frac{-2\pi i 0}{256}} & \hat{x}_1(0)e^{\frac{-2\pi i 0}{256}} & \hat{x}_2(0)e^{\frac{-2\pi i 0}{256}} & \cdots & \hat{x}_{15}(0)e^{\frac{-2\pi i 0}{256}} \\ \hat{x}_0(1)e^{\frac{-2\pi i 0}{256}} & \hat{x}_1(1)e^{\frac{-2\pi i 1}{256}} & \hat{x}_2(1)e^{\frac{-2\pi i 2}{256}} & \cdots & \hat{x}_{15}(1)e^{\frac{-2\pi i 15}{256}} \\ \hat{x}_0(2)e^{\frac{-2\pi i 0}{256}} & \hat{x}_1(2)e^{\frac{-2\pi i 2}{256}} & \hat{x}_2(2)e^{\frac{-2\pi i 4}{256}} & \cdots & \hat{x}_{15}(2)e^{\frac{-2\pi i 30}{256}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \hat{x}_0(15)e^{\frac{-2\pi i 0}{256}} & \hat{x}_1(15)e^{\frac{-2\pi i 15}{256}} & \hat{x}_2(15)e^{\frac{-2\pi i 30}{256}} & \cdots & \hat{x}_{15}(15)e^{\frac{-2\pi i 225}{256}} \end{bmatrix}$$

la cual, de acuerdo a la ecuación (3) es precisamente

$$\begin{bmatrix} x_0^*(0) & x_0^*(1) & x_0^*(2) & \cdots & x_0^*(15) \\ x_1^*(0) & x_1^*(1) & x_1^*(2) & \cdots & x_1^*(15) \\ x_2^*(0) & x_2^*(1) & x_2^*(2) & \cdots & x_2^*(15) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{15}^*(0) & x_{15}^*(1) & x_{15}^*(2) & \cdots & x_{15}^*(15) \end{bmatrix}$$

5. Ahora nos gustaría calcular las FFTs de 16 puntos de $x_t^*(m)$, pero éstas están ordenadas para ser paralelizadas en filas en vez de columnas. Por tanto, debemos transponer para obtener

$$\begin{bmatrix} x_0^*(0) & x_1^*(0) & x_2^*(0) & \cdots & x_{15}^*(0) \\ x_0^*(1) & x_1^*(1) & x_2^*(1) & \cdots & x_{15}^*(1) \\ x_0^*(2) & x_1^*(2) & x_2^*(2) & \cdots & x_{15}^*(2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_0^*(15) & x_1^*(15) & x_2^*(15) & \cdots & x_{15}^*(15) \end{bmatrix}$$

6. Calculamos las FFTs paralelas por columnas y usamos la ecuación (2) para obtener

$$\begin{bmatrix} \hat{x}(0) & \hat{x}(1) & \hat{x}(2) & \cdots & \hat{x}(15) \\ \hat{x}(16) & \hat{x}(17) & \hat{x}(18) & \cdots & \hat{x}(31) \\ \hat{x}(32) & \hat{x}(33) & \hat{x}(34) & \cdots & \hat{x}(47) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \hat{x}(240) & \hat{x}(241) & \hat{x}(242) & \cdots & \hat{x}(255) \end{bmatrix}$$

Éste es el resultado de la FFT que deseamos, y está en la orden correcto! Las matemáticas funcionan, ya estamos listos para considerar la implementación del programa *parallelFFT*. En la discusión siguiente, referiremos a los pasos referidos arriba como pasos 1 a 6.

10.3 Programación de la Implementación en un Sistema Multiprocesador

Una vez que el algoritmo ha sido definido, podemos pasar a su implementación en un sistema multiprocesador real como la tarjeta DSP de la que se dispone.

Un primer requisito será establecer una jerarquía maestro - esclavo (Figura 10-2).

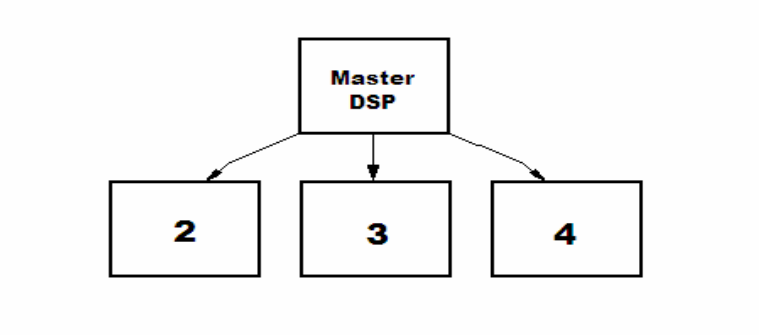


Figura 10-2 Link Ports en un sistema multiprocesador SHARC.
En este caso habrá 1 maestro y 3 esclavos.

Sigamos los pasos de la sección anterior, agrupemos uno o dos cada vez.

- Paso 1 y 2. Los datos de entrada son ordenados en el orden apropiado y distribuidos hacia los procesadores esclavos por el DSP maestro.

- Paso 3. Los procesadores esclavos computan en el paralelo las 16 FFTs de 16 puntos complejos.

- Paso 4 y 5. El procesador maestro recibe los datos y los ordena en el orden apropiado. Mientras se están ordenando los datos, el DSP maestro multiplica y transpone al mismo tiempo.

- Paso 6. Los procesadores esclavos computan en el paralelo las 16 FFTs de 16 puntos complejos.

La Figura 10-3 es el diagrama de flujo de datos. En ella se muestra una nueva etapa de lectura del vector de entrada ya que requiere un tiempo adicional.

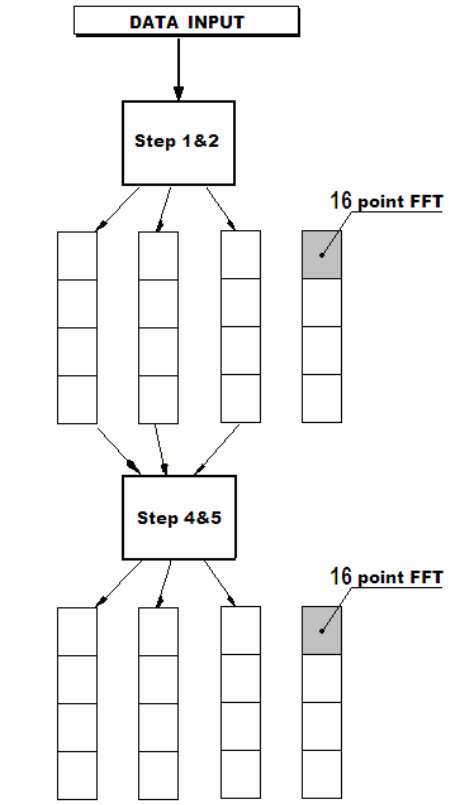


Figura 10-3 Flujo de datos en el mapeado del algoritmo sobre cuatro DSPs

El procesador maestro también procesa parte de los datos después de haber realizado las tareas de control y ordenado. Obsérvese que los pasos 3 y 6 son análogos, se han divido en 4 flujos paralelos e independientes de datos ya que disponemos de 4 procesadores DSP.

Este algoritmo fue probado usando el código Matlab mostrado en la sección siguiente; éste genera simplemente una señal coseno de frecuencia variable y compara su transformada calculada de esta nueva forma con respecto a la transformada que proporcionaría la función Matlab.

10.3.1 Código Matlab (FFT de 256 números complejos)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ParallelFFT.m
% Parallel Implementation of Float-Point FFTs
% DSP based FDOCT (Juan GAGO)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
close all;
N = 256;
M = 16;
R = 16;

for f0 = 1:(N/2)

n = 0 : 1/N : 1-1/N;
x = cos(2*pi*f0*n);

X = fft(x);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Step 1&2: Arrange the input array in a R*M matrix
for row = 1 : R
    A(row, 1:M) = x([1:M] + (row-1)*M);
end;

B = fft(A);         % Step 3: 1st FFT stage: 32 FFTs of 64 complex
points

% Step 4: Change the phase of the elements
for m = 1 : R
    for k = 1 : M
        C(m,k) = B(m,k) * exp (- 2 * pi * i * (m-1) * (k-1) / N);
    end;
end;

D = fft (C');       % Step 5&6: 2nd FFT stage: 64 FFTs of 32 complex
points
                    % Theses FFTs are optimizable thanks to real-
signal properties

% % Arrange the output M*R matrix in an N-array
for row = 1 : M
    Y([1:R] + (row-1)* R) = D(row, R:-1:1);
end;
Z = circshift(Y,[0 -R]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(1);clf;
title('Parallel FFT');
xlabel('freq samples');
ylabel('magnitude');
line(1:N, abs(X),'color','b','LineStyle','none', 'Marker','+');
line(1:N, abs(Z),'color','r','LineStyle','none', 'Marker','.');
drawnow;
end;
```

10.4 Resumen

El ejemplo examinado aquí es el de la FFT para 256 puntos. En el momento de escribir este documento, los códigos para calcular la FFT de 64, 1024, 2048 y 4096 puntos también fueron comprobados en Matlab usando este algoritmo aquí descrito. En esos casos, fueron consideradas matrices 8x8, 32x32, 32x64, y 64x64, respectivamente.

Para el caso de 2048 puntos, éstos fueron ordenados en una matriz de 32 columnas y de 64 filas. 32 FFTs de 64 puntos cada una se hacen en paralelo por columnas. Aplicándose una multiplicación punto se transporta da una matriz de 64 columnas y de 32 filas. Haciendo 64 FFTs de 32 puntos cada una en paralelo por columnas se completa el algoritmo. El único efecto secundario es que la porción de código que paraleliza la FFT no puede ser reutilizada (recuerde que el algoritmo lo necesita dos veces) porque el número de filas y columnas ya no es el mismo. Esto da lugar a un código fuente más largo, pero la eficacia en el número de ciclos es igual de buena.

Capítulo 11
Diseño Software (*parallelFFT)*

11.1 Introducción

En este capítulo se describe cómo la implementación paralela de la FFT en tiempo real de 1024 muestras complejas detallada en el capítulo 10 puede ser mapeada en la tarjeta Hammerhead-PCI. El código se escribe en C y se desarrolla usando el entorno de programación VisualDSP++ de Analog Devices.

Del mismo modo que en el diseño de la aplicación *dspFDOCT* llevada a cabo en el capítulo 8, aquí se tomarán las siguientes decisiones.

- Los canales link port del SHARC suponen el método más sencillo de mover datos entre procesadores.

- Puesto que el procesador SHARC es capaz de soportar accesos DMA de entrada-salida a memoria interna[1], se utilizan búferes de entrada y salida para implementar un esquema “ping-pong”. Se puede verificar que cuando estos búferes son usados, se mejora dramáticamente el rendimiento final de la aplicación.

[1] Estos accesos se ejecutan en segundo plano

11.2 Mapeado del algoritmo

El algoritmo para esta implementación de la FFT puede ser visto de forma simplificada, como la secuencia de bloques mostrada en la Figura 11-1.

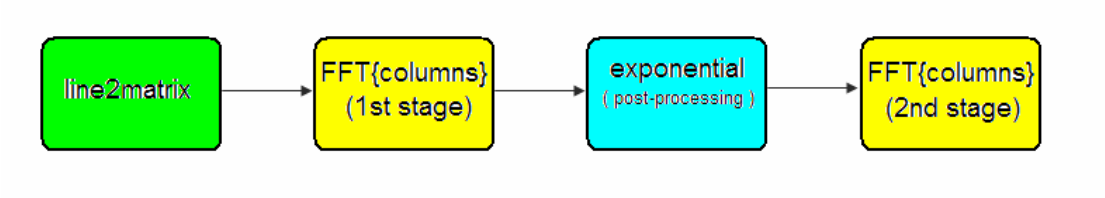


Figura 11-1 Esquema del algoritmo

En esta figura se puede observar dos etapas de procesado FFT idénticas separadas en el tiempo por un post-procesado sin importancia a primera vista.

El mapeado sobre nuestra tarjeta DSP podría ser el siguiente (en este mapeado se hace referencia a los pasos de del algoritmo que aparece en la sección 10.2.3):

- **Pasos 1 y 2:** el procesador DSP maestro distribuye la entrada de datos en el orden adecuado hacia los tres esclavos. Es decir la línea de 1024 puntos se divide en cuatro porciones de 256 puntos que cada procesador de la tarjeta procesará en paralelo.

- **Paso 3 - FFT por columnas (1ª etapa):** los cuatro DSPs calculan en paralelo 32 FFTs de 32 puntos complejos. Cada procesador debe calcular por tanto 8 FFTs.

- **Pasos 4 y 5:** el procesador DSP maestro recibe los datos y los ordena en el orden adecuado. Mientras los datos están siendo ordenados, el maestro DSP multiplica por las exponenciales.

- **Paso 6 - FFT por columnas (2ª etapa):** los cuatro DSPs calculan en paralelo 32 FFTs de 32 puntos complejos. Cada procesador debe calcular por tanto 8 FFTs.

- **Paso 7:** el procesador maestro recibe las tres porciones de línea y añade la porción por él procesada. El resultado es guardado en la memoria SDRAM.

11.2.1 Células de procesado

A cada una de estas FFTs de 32 puntos las llamaremos *células de procesado* y van a ser implementadas aprovechando el código optimizado por Analog Devices para 64 puntos ya que es el valor mínimo de entrada para la eficiente rutina **cfftf**[2].

Para que sea equivalente a la FFT de 32 puntos, se intercalarán ceros entre cada muestra de entrada y se despreciará las 32 últimas muestras de salida. La demostración matemática de este hecho se puede deducir fácilmente y ha sido requerida en el ejercicio 7.9 del libro *Señales y Sistemas* de Alan V. Oppenheim cuyo enunciado es el siguiente.

> 7.9 Considering a length-N signal
>
> $$x = [\ x[0] \quad x[1] \quad \dots \quad x[N-1]\ ]^T$$
>
> and the corresponding vector of DFT coefficients
>
> $$X = [\ X[0] \quad X[1] \quad \dots \quad X[N-1]\ ]^T$$
>
> Consider the length-2N signal obtained by interleaving the values of x with zeros:
>
> $$x_2 = [0 \quad x[0] \quad 0 \quad x[1] \quad 0 \quad x[2] \quad \dots \quad 0 \quad x[N-1]\ ]^T$$
>
> It is easy to demonstrate that the corresponding vector of DFT coefficients will be:
>
> $$X_2 = [\ X[0] \quad X[1] \quad \dots \quad X[N-1] \quad X[0] \quad X[1] \quad \dots \quad X[N-1]\ ]^T$$

Por tanto, las *células de procesado* pueden ser implementadas aprovechando el código optimizado por Analog Devices.

[2] A pesar de computar la FFT de 64 puntos es más rápida que la rutina standard **cfft** de 32 puntos.

11.2.2 Comprobación en Matlab

Aunque el algoritmo de paralelizado fue comprobado en Matlab en el capítulo anterior, veamos si funciona también utilizando *células de procesado de 64 puntos*.

El siguiente código compara esta nueva forma de calcular la transformada de un vector de 1024 puntos con respecto a la calculada teóricamente por la función Matlab (véase el código completo en el apéndice E):

```
N = 1024;
M = sqrt(N);
f0 = 384;
n = 0: 1/N : 1-1/N;
x = cos(2*pi*f0*n);
X = fft(x);


Y =     % Step 1&2:    Arrange the input array in a M*M matrix
        % Step 3:      1st FFT stage: 32 FFTs of 64 complex points
        % Step 4:      Change the phase of the elements
        % Step 5&6:    2nd FFT stage: 32 FFTs of 64 complex points
        %              Arrange the output M*M matrix in an N-array called Y


% Plot X with marker "+"
% Plot Y with marker "."
```

El resultado fue satisfactorio como se observa en la Figura 11-2. En ella vemos cómo las posiciones de las cruces coinciden con las de los puntos.

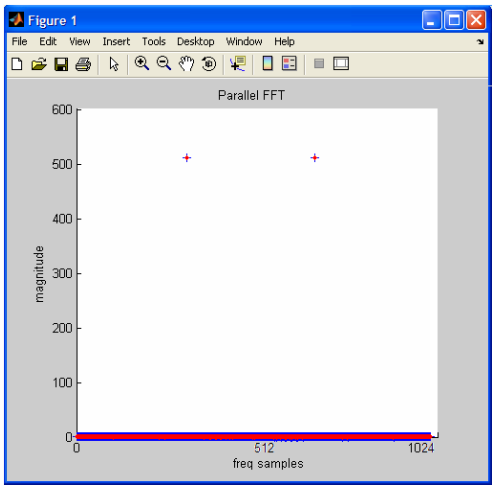


Figura 11-2 Resultado de ParallelFFT.m
Magnitud de la FFT de un tono de 384 Hz.

11.3 Simplificación del mapeado

La Figura 11-3 muestra cómo se puede mapear el algoritmo cuando se está tratando el conjunto de líneas que forman una imagen o frame. Para simplificar se realiza el ***procesado de cada frame en dos pasos*** coincidiendo cada uno de ellos con una de las etapas de cómputo de la FFT en el ***procesado de cada línea*** (pasos 3 y 6 del apartado anterior).

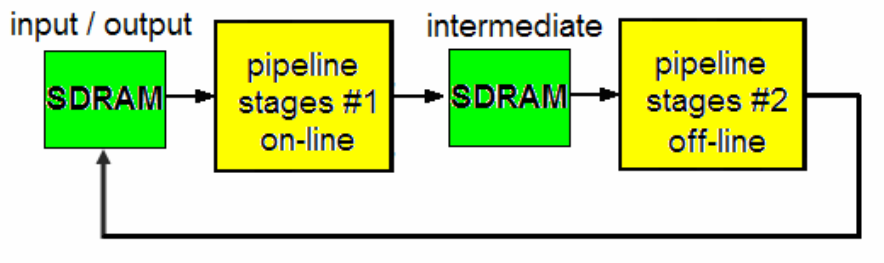


Figura 11-3 Procesado de cada Frame

Las diferencias de esta simplificación con respecto a la aplicación original *dspFDOCT* son las siguientes:

- el procesador maestro no se limita a orquestar el trabajo sino que él mismo también computa algunas de las FFTs.

- se dobla el *Frame Period* ya que la segunda etapa se hace off-line, es decir, se supone que se dispone del suficiente tiempo como para volver a procesar la imagen resultante de la primera etapa antes de recibir una nueva imagen. Ésta hipótesis no tiene por qué ser siempre cierta[3].

- tras el procesado de cada línea de la primera etapa habrá un post-procesado que no será necesario en la segunda etapa off-line. Esto es reflejado a través de los valores que toma el flag *post_proc_flag* en la función Process_Frame (mem1, mem2, post_proc_flag) del código del procesador maestro del Algoritmo 11-1.

[3] No obstante esta hipótesis servirá para inducir que el algoritmo de paralelizado no es de utilidad para nuestros propósitos: si el resultado de esta solución simplificada no es satisfactorio, el de la solución sin simplificar tampoco lo será.

11.3.1 Código DSP

Código del programa principal del procesador maestro:

```
Master DSP Main Program (VisualDSP++)

              FFT_SIZE   = 64              LINE_SIZE = 1024 pixels
              FRAME_SIZE = 32 lines        LINE_FREQ = 3300 Hz

              sdram1 = 0x800000            sdram2 = 0x1000000

              DO
                     WHILE (!input)
                     input = 0
                     Transpose_Frame(sdram2,sdram1)
                     Process_Frame  (sdram1,sdram2, 1)

                     Process_Frame  (sdram2,sdram1, 0)
                     Transpose_Frame(sdram1,sdram2)
                     output = 1

              WHILE (!done);
```

Algoritmo 11-1 Código principal del DSP maestro

Las instrucciones de la función *Process_Frame*() se muestras a continuación:

```
Master DSP Function (VisualDSP++)

       Process_Frame(int *orig, int *dest, int post_proc_flag)

       read_line(sbuf, orig+offset1, DMA10)        // first line (offset1 = 0)

       -----

       read_line(sbuf, orig+offset1, DMA10)        // second line (offset1 = 1)
       process_line_piece(lbuf)

       -----

       DO
              read_line(sbuf, orig+offset1, DMA10)
              process_line_piece(lbuf)
              write_line(lbuf, dest+offset2, DMA11, post_p)

       WHILE (1 < offset1 < FRAME_SIZE)

       -----

       process_line_piece(lbuf)                          // last processing
       write_line(lbuf, dest+offset2, DMA11, post_p)

       -----

       write_line(lbuf, dest+offset2, DMA11, post_p)     // last transmission
```

Algoritmo 11-2 Función del DSP maestro Process_Frame()

A continuación se muestra el pseudo-código ejecutado en cualquiera de los procesadores esclavos:

```
Slave DSP Main Program (VisualDSP++)

              FFT_SIZE   = 64              LINE_SIZE = 1024 pixels
              FRAME_SIZE = 32 lines        NSTAGES   = 2

              DO

                  read_line(sbuf, link_port_rx)

                  DO
                       process_line_piece(lbuf)
                       read_line(sbuf, link_port_rx)
                       write_line(lbuf, link_port_tx)

                  WHILE (line < NSTAGES*FRAME_SIZE)

                  process_line_piece(lbuf)
                  write_line(lbuf, link_port_tx)

              WHILE (!done);
```

Algoritmo 11-3 Código principal del DSP esclavo

11.4 Detalles del Mapeado

11.4.1 Pipelines

Existen dos niveles de pipeline en el procesador DSP maestro:

- Gracias al esquema “ping-pong” implementado en el procesador DSP maestro se podrá procesar una línea mientras se está descargando una nueva de la SDRAM al mismo tiempo que se transmite el resultado del último procesamiento a una nueva posición de memoria (pipeline de primer nivel).

- Dentro del procesado de línea implementado en cada uno de los cuatro procesadores, habrá otro pipeline de segundo nivel entre la porción de línea tratada por el procesador maestro y las otras tres porciones procesadas por los esclavos.

Se establecería, por tanto, un pipeline en el procesador maestro de latencia de dos ciclos y en los DSPs esclavos de latencia de un ciclo.

La Figura 11-4 muestra estos dos pipelines diferentes dentro del procesado del frame en una de sus dos posibles etapas.

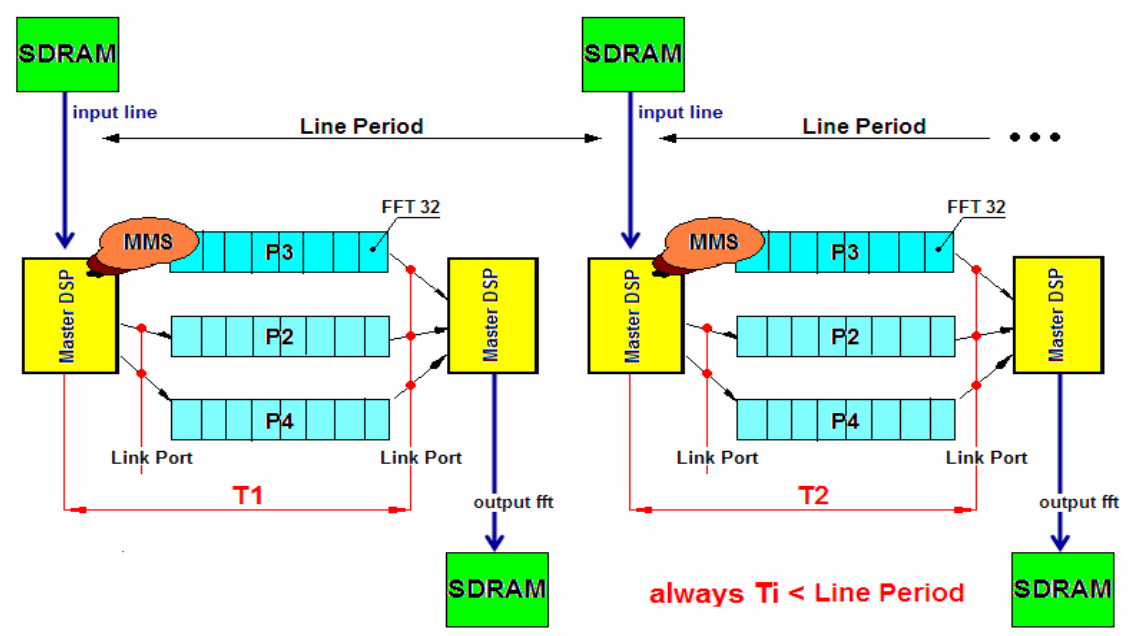


Figura 11-4 Pipelines dentro del procesado del frame en una de sus dos posibles etapas. Detalle de la interconexión de DSPs a través de link ports y MMS

En la Figura 11-4 se puede intuir cómo el cómputo de la porción de la nueva línea en el procesador maestro se solapa en el tiempo con la recepción de datos de los procesadores esclavos. Gracias al pipeline de segundo nivel, el tiempo de procesado corresponderá como mínimo al procesado fijo de la porción de la nueva línea llevado a cabo por el procesador maestro.

Sin embargo la recepción de datos de los procesadores esclavos puede ocurrir con posterioridad a dicho instante. Por tanto, el máximo de estos dos tiempos será el verdadero determinante de la frecuencia máxima del *Line Trigger* (si no consideramos post-procesado):

Tiempo Procesado = max{Cómputo de la porción de la nueva línea en el procesador maestro, Recepción de datos de los procesadores esclavos}

11.4.2 Sincronización de Esclavos

Como se ha referido antes, utilizaremos un pipeline de primer nivel para la descarga y subida de datos. El procesado de cada línea será a sí mismo dividido en unidades de ejecución menores e independientes entre sí para su ejecución en paralelo (pipeline de segundo nivel).

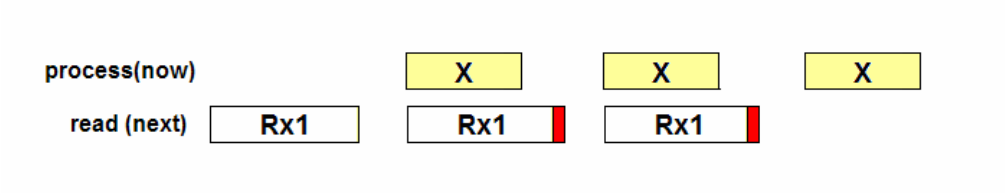


Figura 11-5 Pipeline (esclavo)

Dichas unidades en el procesador esclavo serán (Figura 11-5):

- X: ocho células de procesado de la porción de línea previamente cargada.

- Rxi: recepción de una nueva porción de ¼ de línea y posterior transmisión de la porción previamente procesada (coloreado en rojo).

Hay que hacer notar que el procesador esclavo no comienza a procesar hasta que no haya recibido completamente una porción de línea. Análogamente, no transmitirá la porción recién procesada hasta que no haya recibido otra nueva porción. De este modo quedará garantizada la sincronización.

11.4.3 Sincronización Maestro-Esclavo

La Figura 11-6 representa el procesado de una línea entre el DSP maestro y un esclavo:

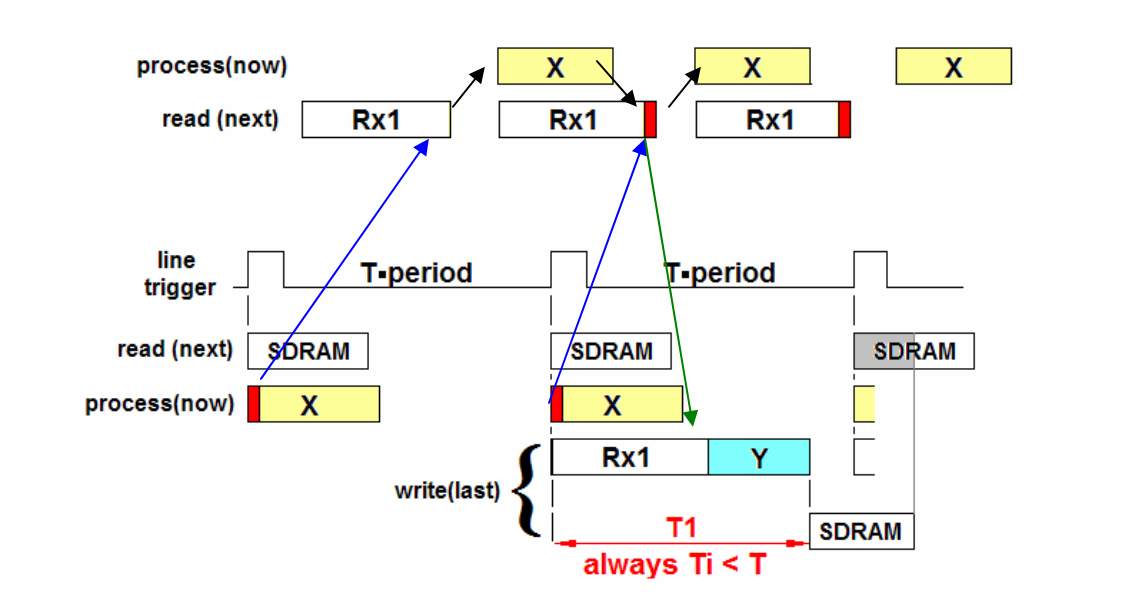


Figura 11-6 Pipeline (arriba esclavo, abajo maestro). Detalle del camino crítico

En ella podemos analizar de manera detallada el camino crítico que limita la frecuencia máxima del *Line Trigger*. Para ello se definen antes las unidades de ejecución independientes entre sí que se ejecutan en paralelo:

- X: transmisión de porciones de ¼ de línea, en paralelo con ocho células de procesado. La transmisión de la porción de línea es asíncrona y concurrente al procesado.

- Rxi: recepción de las porciones procesadas anteriormente por los procesadores esclavos; su valor mínimo será igual a la duración de la transferencia de la porción de línea actual (caso en el que no haya procesado ni en el procesador maestro ni en el esclavo).

- Y: reajustes de la línea antes de guardarla en la SDRAM, post-procesado si estamos en la primera etapa del algoritmo.

Los tiempos correspondientes a los bloques sin colorear *Rxi* dependen del pipeline y de la sobrecarga actual de los link-ports y/o bus del sistema; por tanto serán distintos en cada línea. La temporización de estos bloques se hará off-line, es decir, se registrará para cada línea; una vez finalizada la ejecución del programa en la tarjeta DSP, el PC anfitrión calculará la media de estos datos (la aplicación final lo mostrará en un indicador llamado *Line Process Average*).

11.5 Resumen

En este capítulo se ha mapeado el algoritmo de paralelizado de la FFT descrito en la conferencia IEE Irish Signals and Systems Conference (Dublin City University, Sept – 2005) en la tarjeta Hammerhead-PCI.

La medida del tiempo de procesado que se realiza en el próximo capítulo nos dirá si es posible la aplicación de este algoritmo en la tomografía FDOCT (10,000 FFTs/sec).

Sin embargo, se han tomado hipótesis de simplificación las cuales servirán para inducir que dicho algoritmo de paralelizado no es de utilidad para nuestros propósitos: si el resultado de esta solución simplificada no es satisfactorio (menos de 10,000 FFTs/sec), el de la solución sin simplificar tampoco lo será.

Capítulo 12
Resultados y Posibles Mejoras (*parallelFFT*)

12.1 Introducción

En este capítulo se estudian los resultados de la aplicación *parallelFFT* diseñada en el capítulo anterior. En concreto se medirá el tiempo de procesado que nos dirá si es posible la aplicación del paralelizado de la FFT en la tomografía FDOCT (10,000 FFTs/sec). También se analizan las mejoras que podrían derivarse del uso del nuevo procesador TigerSHARC.

12.2 Estimación del tiempo fijo de procesado

Como se dijo en el capítulo anterior, el tiempo de procesado tiene dos componentes: un tiempo fijo necesario para hacer los cálculos y otro tiempo variable debido a la transmisión de datos hacia los procesadores esclavos.

Tiempo Procesado = max{Cómputo de la porción de la nueva línea en el procesador maestro, Recepción de datos de los procesadores esclavos}

En primer lugar nos centraremos en el tiempo fijo del procesado FFT; en primera aproximación se puede estimar al siguiente valor:

- post-procesado = **13,896** ciclos
- ocho células de procesado = 8 × 808 ciclos[1]
- conversión entero–flotante–entero de los búferes de entrada y salida ≈ 0 ciclos
- Pasos 1&2 + Paso 7 ≈ 0 ciclos

[1] FFTs de 64 puntos complejos = 808 ciclos sin SIMD

Tiempo fijo de procesado = 8 × (808) + 13,896 = 6464 + 13,896 = 20,360 ciclos

Este resultado no es bueno ya que ni siquiera reduciría el tiempo que un solo procesador DSP emplearía en procesar la FFT de 1024 números complejos[2]: **18,228** ciclos (114 us).

Si se analiza con más detenimiento el número de ciclos requeridos para el post-procesado se observa que es excesivo debido a que se trata de multiplicaciones complejas que son computacionalmente muy costosas:

$$a \times e^{x} = a \times \sum_{n=0}^{\infty} \frac{x^{n}}{n!} = a \times \left[1 + x + \frac{x^{2}}{2} + \frac{x^{3}}{6} + \ldots \right]$$

Los **13,896** ciclos pueden ser fácilmente reducidos utilizando la aproximación de Taylor de un orden menor, por ejemplo de primer orden:

$$a \times e^{x} \approx a \times (1 + x)$$

No obstante en este capítulo se va a hacer sólo una aproximación a los tiempos de procesado con lo que será suficiente con la multiplicación original que, aunque nos dé una precisión que no se necesita, es bastante más sencilla de utilizar. La multiplicación en cuestión se realiza a través de la siguiente función específica del procesador:

complex_float **cmltf** (*complex_float* **a**, *complex_float* **x**);

[2] Hay que recordar que si se hubiera aprovechado al máximo los 64 puntos de la célula de procesado (en vez de implementar con ellas FFTs de 32 puntos) podríamos haber obtenido la FFT de 4096 puntos en unos tiempos del orden de:

Tiempo fijo de procesado = 16 × (808 ciclos) + 13,896 ciclos

= 12,928 + 13,896 ciclos

Este resultado representaría una reducción aproximada de un factor de 3 con respecto al que un solo procesador DSP emplearía en computar la FFT de 4096: **86,615** ciclos. Esto demuestra que no es una inutilidad plantearnos la implementación en paralelo de la FFT entre los cuatros procesadores de la tarjeta HH-PCI.

12.3 Medida del tiempo variable de procesado

De la misma manera que se hizo en *dspFDOCT*, la temporización del procesado de línea en los procesadores esclavos se hará off-line, es decir, se registrará para cada línea. Una vez finalizada la ejecución del programa en la tarjeta DSP, el PC anfitrión calculará la media de estos datos (la aplicación final lo mostrará a través del indicador llamado *Lines-Process Average).* En la Figura 12-1 se puede observar su valor cuando tenemos una imagen en la que barremos algunas frecuencias significativas hasta llegar a la de Nyqyuist.

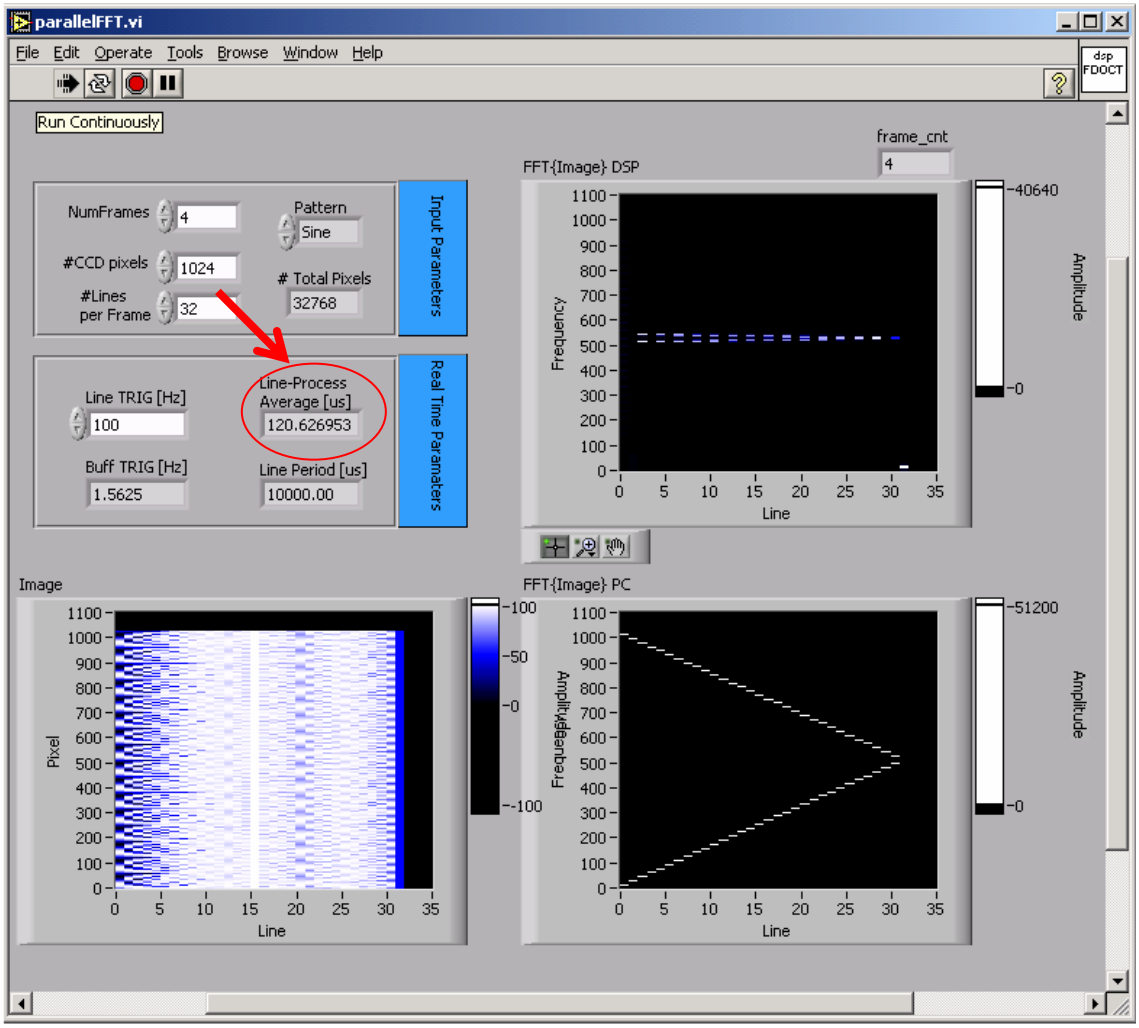


Figura 12-1 Interfaz de usuario de la aplicación final. Posición del indicador.

12.4 Análisis de los Resultados

Como puede verse en la Figura 12-1, los bloques *Rxi* (definidos en la sección 11.4.3) durarán **120 us** de media sobrepasando el *Line Period* requerido para la tomografía FDOCT.

Rxi = 120 us

Como mostraba la Figura 11-6, el camino crítico tomará lugar sólo durante la primera etapa del algoritmo (con post-procesado) en el procesador maestro. Su valor será el siguiente:

T_{FFT} (one stage) = 8 × 808 cycles = 6464 cycles = 83.2 us

Line Processing Time= max $\{T_{FFT}$, Rxi$\}$ + Tpost_processing
= 120 us + 173.7 us ≈ 300 us

Por tanto se puede concluir que la implementación de este algoritmo en la tarjeta HH-PCI no servirá para que la tomografía FDOCT alcance velocidades de 10,000 FFTs/sec. Es por esta razón que no se considera necesario optimizar el programa debido a que no funcionará aun tomando varias simplificaciones. Algunas de ellas se enumeran a continuación:

- se supuso que se disponía de suficiente tiempo como para volver a procesar la imagen resultante de la primera etapa antes de recibir una nueva imagen

- cálculo del la magnitud de los resultados y conversión de datos ≈ 0 us

- se supuso que no se podía optimizar las multiplicaciones por exponencial aunque se podría haber empleado la aproximación de Taylor de orden uno. Pero, ¿para qué se va a optimizar el post-procesado si al hacerlo no se van a reducir esos 120 us de comunicación maestro-esclavo?

En la aplicación *parallelFFT* nos limitaremos a los siguientes valores:

Line Trigger

Line Period = 300 us

Line Trigger Frequency = 3.3 KHz [3]

Throughput = 2 × 3 × 1 KB ÷ T = 20 MB/s

Buffer Trigger

A diferencia de la recepción síncrona *Rxi*, la finalización del procesado de la última línea es indicada al exterior a través de un cambio de nivel en la línea *Buffer Trigger* (flag software llamado *output* en el código del Apéndice E). Debido al hecho de calcular la FFT de la imagen en dos etapas, el *Buffer Period* valdrá como mínimo:

$$Buffer\_Period = 2 \times Num\_Lines \times Line\_Period$$

En la aplicación *parallelFFT* se ha fijado a este valor.

Line Period = 300 us (3.3 KHz)

Num_Lines = 100 → Buffer Period = 0.06 s

Buffer Trigger Frequency = 16.7 fps

Fuente de errores

Como se aprecia en la Figura 12-1, existe una clara diferencia entre lo que obtenemos y lo que deberíamos tener. No obstante existe simetría (como toda transformada de señal real) lo cual es importante aunque en el barrido de frecuencias existan offset y ausencias periódicas de determinadas frecuencias.

[3] En la versión preliminar de la aplicación *dspFDOCT* mostrada en el apartado 5.2.1 del presente proyecto, se calculaba el algoritmo FFT complejo. El throughput de procesado se estimó en N × 8777 FFTs/sec donde N es el número de procesadores esclavos. Lo que quiere decir que con solo dos procesadores esclavos se obtendrían 17,554 FFTs/sec. suficiente para la tomografía FDOCT.

12.5 Nuevo Procesador DSP

Se analiza aquí cómo la actualización del procesador TigerSHARC podría mejorar el cálculo de la paralelización de la FFT de 1024 números complejos en puntos flotante.

- La principal ventaja del TigerSHARC es que procesa la FFT en menos tiempo debido a que normalmente trabaja a 500 MHz mientras que el ADSP21160 sólo a 80 MHz.

- La FFT de 32 puntos está ya disponible para este procesador (en vez de las FFTs básicas de 64 puntos al usar el ADSP21160) por tanto se podría ejecutar el algoritmo paralelo de una manera más eficiente.

12.6 Resumen

Este capítulo muestra los resultados obtenidos del programa *parallelFFT* diseñado en el capítulo anterior.

Se concluye que el mapeado del algoritmo sobre la tarjeta HH-PCI no servirá para que la tomografía FDOCT alcance velocidades de 10,000 FFTs/sec. Es por esta razón que no se considera necesario optimizarlo debido a que no funcionará aun tomando varias simplificaciones.

No obstante dicho algoritmo podría ser de bastante utilidad en alguna tarjeta multiprocesador TigerSHARC.

PARTE IV:
Análisis Final

Capítulo 13
Procedimientos de Prueba

13.1 Introducción

Este capítulo muestra las herramientas empleadas en la depuración de los programas tanto para el PC anfitrión como para cada uno de los procesadores que forman parte de la tarjeta DSP.

El software del lado del PC anfitrión requiere para poder funcionar la tarjeta Hammerhead-PCI y los correspondientes drivers de BittWare. El *kit* DSP21K-SF de Bittware proporciona dichos drivers junto con algunas herramientas de diagnóstico de la tarjeta.

La Figura 13-1 indica que tanto la aplicación final de usuario como las herramientas de diagnóstico de la tarjeta están basadas en la librería *HIL (Host Interface Library).*

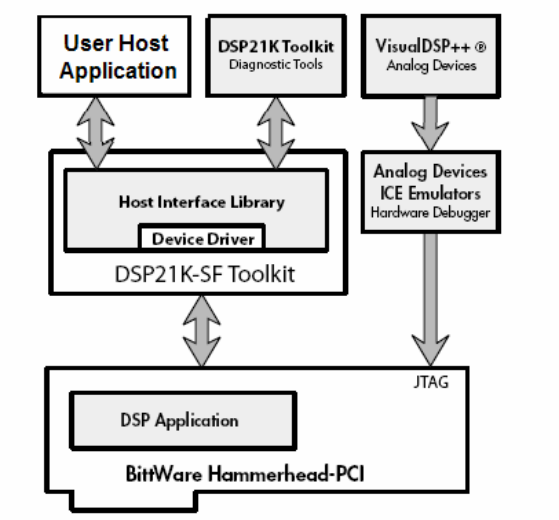


Figura 13-1 Arquitectura Software de BittWare

En la misma Figura 13-1 se advierte que para la depuración software del lado de los procesadores DSP se requiere de un emulador JTAG además del entorno *VisualDSP++*.

13.2 Depuración software para DSPs

En un primer momento se empleó el depurador embebido en el entorno de diagnóstico *diag21k* de Bittware. Este pequeño depurador es adecuado para eliminar los posibles errores del código de un solo procesador DSP (ver apéndice C).

No obstante para comprobar una aplicación multiprocesador tan compleja como la nuestra, se optó por el emulador *Summit-ICE* que se conecta a la tarjeta DSP a través de un conector JTAG como muestra la Figura 13-2.

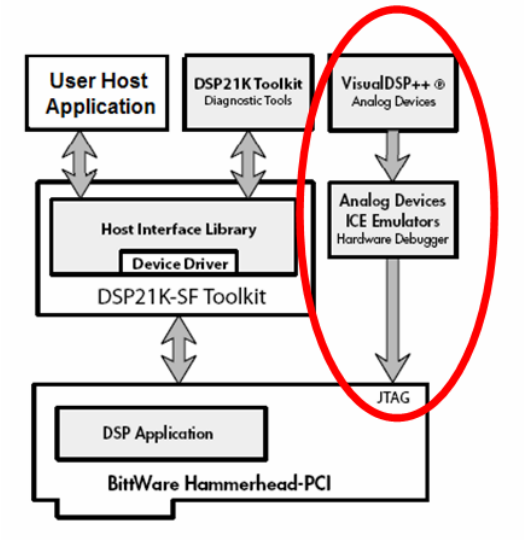


Figura 13-2 Emulador JTAG

Gracias a que su integración junto con el entorno *VisualDSP++* la depuración resulta más amigable al programador incluso para el caso monoprocesador (apéndice B).

La instalación al PC de este emulador se hizo a través del bus PCI. Fue necesario cargar sus correspondientes drivers, como no podría ser de otra manera.

13.3 Arquitectura software del PC

La Figura 13-3 detalla las aplicaciones que se pueden ejecutar sobre el PC anfitrión. Se observa que todos los niveles de programación se apoyan en los distintos componentes del *kit* de herramienta DSP21K-SF de Bittware los cuales están todos basados en la librería *HIL* que podemos considerar como el nivel básico.

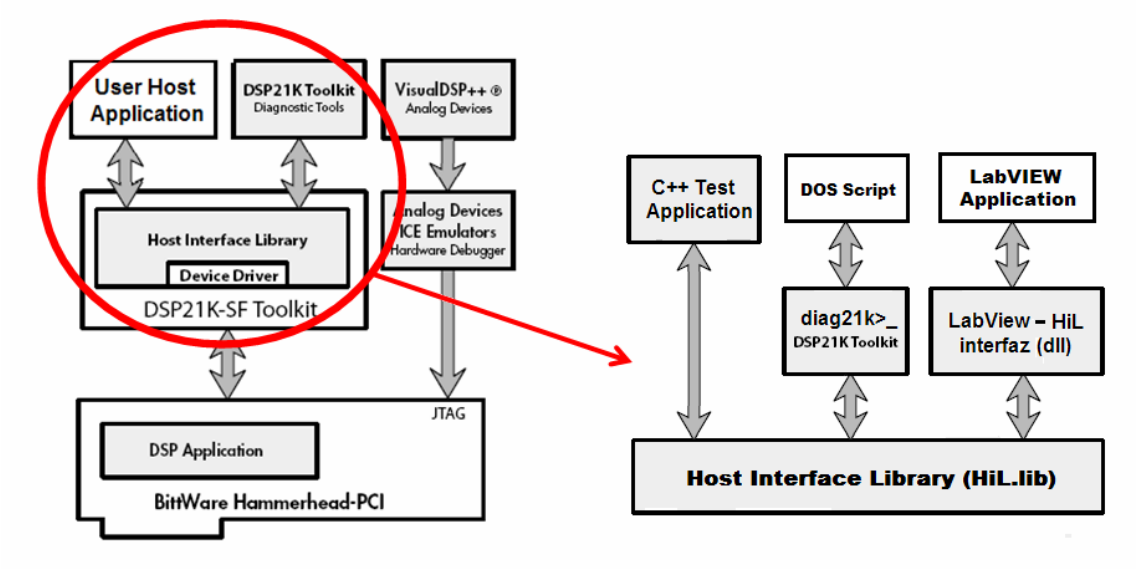


Figura 13-3 Arquitectura Software basada en la librería HIL.lib de BittWare

La Figura 13-3 muestra los distintos niveles de programación del PC anfitrión:

- A nivel de representación podemos elegir entre los entornos C++ o LabView. Para usar éste último debemos incluir un nivel intermedio que sirva de interfaz entre la librería *HIL.lib* y el diagrama de bloques característico de LabView donde se hagan las llamadas a dicha librería a través de "bloques" como el de la Figura 13-4.

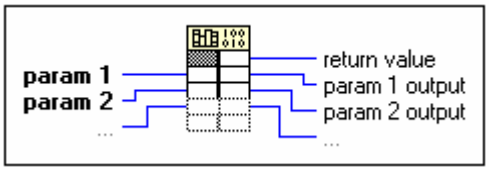


Figura 13-4 Llamada a función de librería dll

- El intérprete de comandos *diag21k* proporcionado por *BittWare* también se encuentra a un nivel intermedio de programación. Es una herramienta basada en

la librería *HIL* muy útil para la depuración de errores en la primera fase del desarrollo. Para obtener programas independientes crearemos *scripts* de comandos del mismo modo que los ficheros de ejecución por bloques de MS-DOS (ver apéndice C).

Bittware ofrece el *kit* DSP21K-SF para ser usado bajo Windows aunque también sería posible conseguir una versión para Linux. Se optó por Windows XP debido a que la licencia para *VisualDSP++* requería este sistema operativo.

No obstante, Linux ofrecería las siguientes ventajas de programación.

- mayor estabilidad para aplicaciones en tiempo real
- mayor flexibilidad durante la fase de programación
- libre

13.4 Proceso de depuración software

El desarrollo software de la aplicación *dspFDOCT* se llevó a cabo a través de una serie de etapas:

- En primer lugar se utilizó el depurador de *VisualDSP++* costumizado para funcionar junto con el emulador *Summit-ICE*. También se utilizó el depurador embebido dentro de la consola de comandos *diag21k* de Bittware para comprobar la transferencia entre procesadores y tiempos de procesado.

- Una vez que las comunicaciones habían sido comprobadas se obtuvo la primera versión de la aplicación *dspFDOCT* utilizando el entorno *MS Visual C++* y la librería *HIL.lib* de BittWare. En esta etapa se priorizaron conceptos como la inicialización de la tarjeta DSP, gestión de errores, etc. más que la presentación de resultados (se empleó una imagen patrón DC que fue testeada con un entorno gráfico tipo consola).

- Para testear patrones de imágenes más complejos se optó por el entorno gráfico de programación *LabView* ya que desde él se pueden llamar fácilmente las mismas funciones de configuración de la tarjeta DSP que fueron usadas en el programa en C. Además este entorno es el que normalmente se utiliza en biomedicina gracias a la potente presentación gráfica de resultados.

El desarrollo del software de la aplicación *parallelFFT* siguió un proceso similar. En la primera etapa se utilizó Matlab para probar un número de ideas. Una vez que se probaron los conceptos iniciales, la implementación en DSP fue desarrollada.

13.5 Resumen

Tras diseñar e implementar cada una de las aplicaciones que componen este proyecto, se han realizando las pruebas oportunas que verifican su buen funcionamiento. Esta etapa del proyecto es una de las más importantes, ya que, suponiendo un funcionamiento correcto del hardware, es la manera de depurar la fase software.

Capítulo 14
Conclusiones

14.1 Resultados

Para finalizar el proyecto se presenta un resumen de las principales conclusiones que se han obtenido en la elaboración del mismo.

En primer lugar, hemos de destacar el hecho de que no se han alcanzado la de velocidad de procesado inicialmente planteada para la aplicación *dspFDOCT* en 10,000 FFTs/sec. A medida que el proyecto ha sido desarrollado su valor se han visto reducido a 5000 FFTs/sec, no obstante, el valor fijado inicialmente sería fácilmente alcanzable con tan sólo añadir más procesadores DSP que computen la FFT paralelamente[1].

Como estábamos limitados por la plataforma hardware que disponíamos (de sólo 4 procesadores DSP) se pensó en una forma alternativa de paralelizado que parecía en principio adecuada para aumentar la velocidad del procesado FFT. Se la denominó *parallelFFT* y al ser una idea tan nueva se consideraron en un primer momento solo muestras complejas a la entrada a fin de facilitar el proceso de depuración.

Tomando ésta y otras hipótesis de simplificación, la versión implementada en el presente proyecto fue capaz de producir solo 3,300 FFTs/sec. A pesar de este resultado insuficiente, lo significativo fue que en el proceso de desarrollo de la aplicación se descubrió que era imposible alcanzar las velocidades de procesado requeridas en la tomografía FDOCT con el método propuesto. Es por esta razón que la versión *parallelFFT* que el presente proyecto propone no es la más óptima pudiéndose depurar su programación.

[1] Cinco procesadores esclavos + un procesador maestro producirían 12,500 FFTs/sec

14.2 Capacidades adquiridas

Para la consecución de estos resultados fue necesaria la realización de las siguientes tareas:

- Instalación y uso del entorno de programación Matlab
- Instalación y uso del entorno de programación LabView
- Instalación y uso del entorno de programación Microsoft Visual C++
- Instalación y uso del entorno de programación VisualDSP++
- Instalación de la tarjeta Hammerhead-PCI junto con sus drivers (HIL)
- Instalación del emulador Summit-ICE e integrarlo en VisualDSP++

Aunque no está directamente relacionado, también se llevó a cabo las siguientes tareas:

1. Instalación del frame-grabber Solios XCL junto con sus drivers (Matrox Intellicam)

2. Conexionado de la cámara AViiVA M2CL de Atmel al frame-grabber en dos niveles (apéndice D):
    - hardware: diseño e implementación del cableado de disparo.
    - software: configuración con la herramienta Intellicam de un fichero DCF que permita el uso de la cámara en cualquier lenguaje de programación.

Gracias a estas últimas, se comprendió el modo de funcionamiento del protocolo CamaraLink que es necesario para el diseño del *Line Grabber* llevado a cabo en el capítulo 6.

14.3 Líneas Futuras

Por supuesto todo proyecto es susceptible de ser mejorado y éste no va a ser una excepción. Es por ello que mencionamos algunos puntos de posibles mejoras:

- En el capítulo 6 quedaría por realizar la construcción PCB del *Line Grabber*. Esto conllevaría hacer pruebas con la cámara que usa el protocolo CameraLink y los enlaces link ports del procesador ADSP21160. Así mismo la construcción PCB del circuito DAC tendría que ser llevada a cabo para que la tarjeta DSP pueda controlar el scanner de adquisición de imágenes

- En el capítulo 9 se exponen como posible mejora de la aplicación *dspFDOCT* la compra de la tarjeta EZ-KIT Lite de Analog Devices para aumentar el número de procesadores ADSP21160 que trabajarían en paralelo.

- Hacemos mención a las líneas futuras comentadas en el capítulo 3 donde se propone el procesador *TigerSHARC* como actualización del procesador ADSP21160 que se ha utilizado en el presente proyecto.

- En el capítulo 13 de pruebas se propuso también la posibilidad de utilizar Linux en vez de Windows como sistema operativo soporte de la aplicación de usuario final.

Referencias

[1] Bittware HammerHead-PCI with four ADSP21160. User Manual.
http://www.bittware.com/products/name/hammerhead.cfm

[2] Matrox SOLIOS xCL. User Manual.
http://www.matrox.com/imaging/products/solios_xcl/home.cfm

[3] ATMEL AViiVA M2CL 1024 User Manual.
http://www.atmel.com/dyn/products/product_card.asp?part_id=2263

[4] ADSP-21160M and TS201 Datasheet
http://www.analog.com/UploadedFiles/Data_Sheets/

[5] Summit-ICE Emulator. Hardware and Software Installation Guide.
Analog Device Inc, December 2002.

[6] Fourier Transform in the West, MIT.
http://www.fftw.org/

[7] CameraLink Protocol
http://www.siliconimaging.com/ARTICLES/CameraLink.htm

[8] Optical Coherence Tomography, Rainer Leitgeb , 2004
http://lob.epfl.ch/

[9] IEEE 754 Floating Point Optimised DSP Library for the TigerSHARC TS201 DSP. Cycle Performance Manual. BittWare Inc. version 2.1.0

[10] Real-Time Signal Processing - Comparing TigerSHARC and PowerPC Via Continuous cFFTs. BittWare Inc. December 2003.

[11] Parallel Implementation of Fixed-Point FFTs on TigerSHARC® Processors.
Analog Devices Inc. http://www.analog.com/

[12] System Architecture determines Performance of Multiprocessor Applications
Alacron Inc. http://www.alacron.com/

Referencias

Citas

[1] Conferencia IEE Irish Signals and Systems Conference (Dublin City University) Septiembre, 2005.

[2] Curso EE131. Instituto de Tecnología de California (Caltech), Fei Wang, 2006.

[3] Congreso *del IEEE-World sobre Inteligencia Computacional*, Touretzky 1994.

Bibliografía Complementaria

[1] Inside of the FFT Black Box. Eleanor Chu, Alan George. CRC Press, 2000.

[2] Señales y Sistemas, V. Oppenheim, Alan S. Willsky. Prentice Hall, 1994

[3] The Fast Fourier Transform. E. Oran Brigham. Prentice Hall: New Jersey, 1974.

Tablas

3-1 Asignación de Puertos de acuerdo a la Configuración.

4-1 Número de procesadores según el throughput deseado
4-2 FFTs/sec versus número de procesadores (Cluster Bus)
4-3 FFTs/sec versus número de procesadores (DPLM)
4-4 Benchmark de diferentes longitudes de la FFT en el TigerSHARC®
4-5 Benchmark en diferentes procesadores

6-1 Asignación de bits de los píxeles (configuración base)

7-1 Número de procesadores según el throughput deseado
7-2 FFTs/sec versus número de procesadores (Cluster Bus)
7-3 FFTs/sec versus número de procesadores (DPLM)

Algoritmos

8-1 Código principal del DSP maestro
8-2 Código principal del DSP esclavo
8-3 Función del DSP maestro Process_Line()

11-1 Código principal del DSP maestro
11-2 Función del DSP maestro Process_Frame()
11-3 Código principal del DSP esclavo

Figuras

1-1 Configuración óptica FDOCT junto con detector y sistema de procesado

2-1 Ámbito de aplicación de la OCT
2-2 Interferómetro de Michelson
2-3 Configuraciones ópticas TDOCT y FDOCT
2-4 Configuración óptica FDOCT junto con detector y sistema de procesado
2-5 Tomografía (imagen 2D)
2-6 Configuración óptica de la FDOCT (fotografía)

3-1 Estructura de la tarjeta HH-PCI.
3-2 Arquitectura del procesador ADSP21160
3-3 Diagrama de bloques de la configuración Base, Medium, y Full
3-4 Funcionamiento del protocolo Channel Link.
3-5 Modo Free Run
3-6 Modo Trigger
3-7 Scanner de adquisición de imágenes (fotografía)
3-8 Control de motores del scanner (ejemplo para frames de 4 líneas)

4-1 Rendimiento de la FFT en diferentes procesadores (año 2007)

6-1 Conexión de la tarjeta DSP a la cámara line-scan y al scanner (line grabber)
6-2 Configuración de taps.
6-3 Componentes del Line Grabber
6-4 Línea Patrón (valores simulados)
6-5 Conexión de la tarjeta DSP, cámara line scan y scanner (frame grabber)
6-6 Sincronización scanner – cámara.
6-7 Circuito de Conversión D/A

7-1 Memoria Global y Cluster-Bus
7-2 Memoria Local y Enlaces Link-Port (DPLM)
7-3 Rendimiento de la FFT en diferentes procesadores (año 2001)
7-4 Número de FFTs/sec obtenible para las arquitecturas Cluster Bus y DPLM y para los procesadores SHARC I y II

Figuras

8-1 Los link ports del SHARC proporcionan una manera fácil de mover datos entre procesadores.
8-2 Se usaron dos búferes de entrada-salida para implementar el esquema ping-pong.
8-3 Configuración de la tarjeta HH-PCI
8-4 Pre & post-procesado de datos

9-1 Interfaz de usuario de la aplicación final (componente DC)
9-2 Interfaz de usuario de la aplicación final (patrón sinusoidal)
9-3 Configuración de memorias locales en la tarjeta HH-PCI (N=2)
9-4 Configuración de memorias locales en la tarjeta HH-PCI (N=3)
9-5 Diagrama de Bloques de las Conexiones Link Port: Tarjeta con dos Procesadores

10-1 Standard FFT de 16-point radix-2
10-2 Link Ports en un sistema multiprocesador SHARC.
10-3 Flujo de datos en el mapeado del algoritmo sobre cuatro DSPs

11-1 Esquema del algoritmo
11-2 Resultado de ParallelFFT.m
11-3 Procesado de cada Frame
11-4 Pipeline dentro del procesado del frame en una de sus dos posibles etapas.
11-5 Pipeline (esclavo)
11-6 Pipeline (arriba esclavo, abajo maestro).

12-1 Interfaz de usuario de la aplicación final. Posición del indicador.

13-1 Arquitectura Software de BittWare
13-2 Emulador JTAG
13-3 Arquitectura Software basada en la librería HIL.lib de BittWare
13-4 Llamada a función de librería dll

Acrónimos

FDOCT	Fourier Domain Optical Coherence Tomography
MRI	Magnetic Resonance Imaging
CT	Computerized Tomography
FFT	Fourier Fast Transform
DSP	Digital Signal Processor
FPGA	Field Programmable Gate Array
SHARC	Super Harvard ARchitecture Computer
SIMD	Single-Instruction, Multiple-Data
SISD	Single-Instruction, Single-Data
DMA	Direct Memory Access
PCI	Peripheral Component Interconnect
PMC	PCI MezanineCard
DPLM	Dual Port Local Memory
GFLOPs	Giga or billions of FLoating-point Operations Per Second
MFLOPs	Mega or millions of FLoating-point Operations Per Second
GOPs	Giga or billions of Operations Per Second
CCD	Charge-Coupled Device
PCB	Printed Circuit Borrad.
LVDS	Low-Voltage Differential Signaling

Glosario

line scan Tipo de cámara digital consistente en un vector unidimensional de píxeles (línea). Se diferencia de las cámaras frame-scan porque aquellas generan una imagen 2D (frame).
A veces también se la conoce como array CCD pero estrictamente hablando "CCD" refiere únicamente a la forma en la que la imagen es leída desde el chip (no es ningún sensor óptico).

Glosario

frame-grabber Tarjeta PCI o PMC con interfaz CameraLink que sirve de enlace entre la cámara CCD y el PC anfitrión del frame-grabber u otros dispositivos de control.

PC anfitrión Equipo informático dispuesto de conectores que cobija cierta placa de aplicación específica: tarjeta DSP, frame-grabber, gráfica, etc.

Link Port Enlace de entrada/salida de datos a alta velocidad del procesador ADSP21160.

CamaraLink Interfaz de la cámara line-scan empleada.

ChannelLink Protocolo de comunicación en el que está basado el interfaz CamaraLink.

Cluster Bus Bus principal de la tarjeta DSP para interconectar procesadores, memorias y comunicaciones con el bus PCI del PC anfitrión.

Unidades de Medidas

throughput	cantidad de información digital por unidad de tiempo que es entregada obre la memoria destino. Su unidad es **FFTs/sec** o líneas procesadas por segundo.
latency	latencia [líneas]
data flow/rate	tasa del flujo de datos [Byte/s]
line period/rate	período/frecuencia de línea [Klíneas/s]
frame period/rate	período/frecuencia de frama [fps]
exposure/integration time	tiempo de exposición/integración [us]

APÉNDICES

Apéndice A
Resumen del ADSP-21160M

A.1 Procesador ADSP-21160M

En este apéndice se da una visión con algo más de detalle aquellas funcionalidades del procesador ADSP-21160M que serán usadas en el presente proyecto.

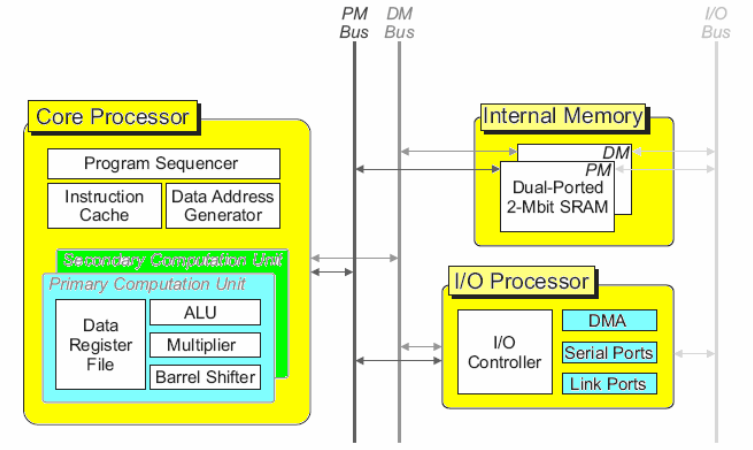


Figura A-1 Diagrama de bloques del procesador ADSP-21160

A.1.1 Frecuencia del Reloj y Temporizador

Los procesadores ADSP-21160M SHARC a bordo de la plataforma Hammerhead-PCI de Bittware operan a 80 MHz, aunque la velocidad se ha incrementado a 100MHz en las tarjetas más modernas.

Selecting Core to CLKIN Ratio

CLK_CFG3-0	Core/CLKIN ratio
0010	2:1
0011	3:1
0100	4:1
All others	Reserved

	CLKIN Input (MHz) ⇓			
	25	33.3	40	50
Clock Ratios	Core CLK (MHz) ⇓			
2:1	50	66.6	80	80 (21160M) 95 (21160N)
3:1	75	100	N/A	N/A
4:1	100	N/A	N/A	N/A

Tabla A-1 Entrada de reloj para el procesador ADSP-21160

Para ayudar a la optimización y/o depuración del programa es importante obtener información exacta sobre las características de funcionamiento del algoritmo que está siendo probado. El procesador de SHARC contiene un *timer* en el secuenciador de programa, que cuenta los ciclos de reloj existentes entre la inicialización y cada lectura de su valor (Figura A-2). Este *timer* cuenta descendentemente un número dado, y causa una interrupción cuando alcanza cero; el tiempo más largo que puede temporizar es aproximadamente 240 segundos.

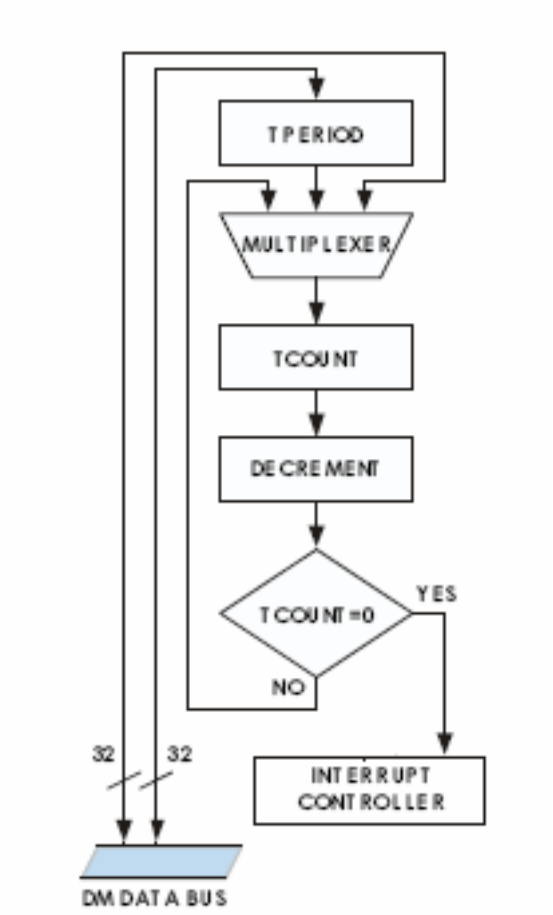


Figura A-2 Temporizador del procesador ADSP-21160

Por tanto, este contador de tiempo será extensivamente utilizado para el desarrollo del código y todas las medidas del funcionamiento de SHARC han sido contabilizadas en ciclos de reloj durante este trabajo. La tarjeta de desarrollo Hammerhead de BittWare no contiene ningún sistema operativo, aunque hay que hacer notar que están disponibles funciones de tiempo real que permiten la temporización exacta de segmentos de código

A.1.2 Controlador DMA

El controlador de DMA a bordo del chip del procesador ADSP-21160, permite transferencias de datos sin sobrecarga por cabeceras (*zero-overhead*) y sin necesidad de la intervención del procesador DSP. El controlador DMA funciona independiente e invisiblemente al núcleo, es decir, permite que las operaciones DMA ocurran simultáneamente mientras el núcleo está ejecutando las instrucciones de programa.

Las transferencias DMA pueden ocurrir entre la memoria interna del ADSP-21160 y la memoria externa, periféricos externos, o el PC anfitrión. Todas estas transferencias ocurren a través del bus principal (*cluster-bus*) así que son controlados a través del SHARC*fin* (Figura A-3).

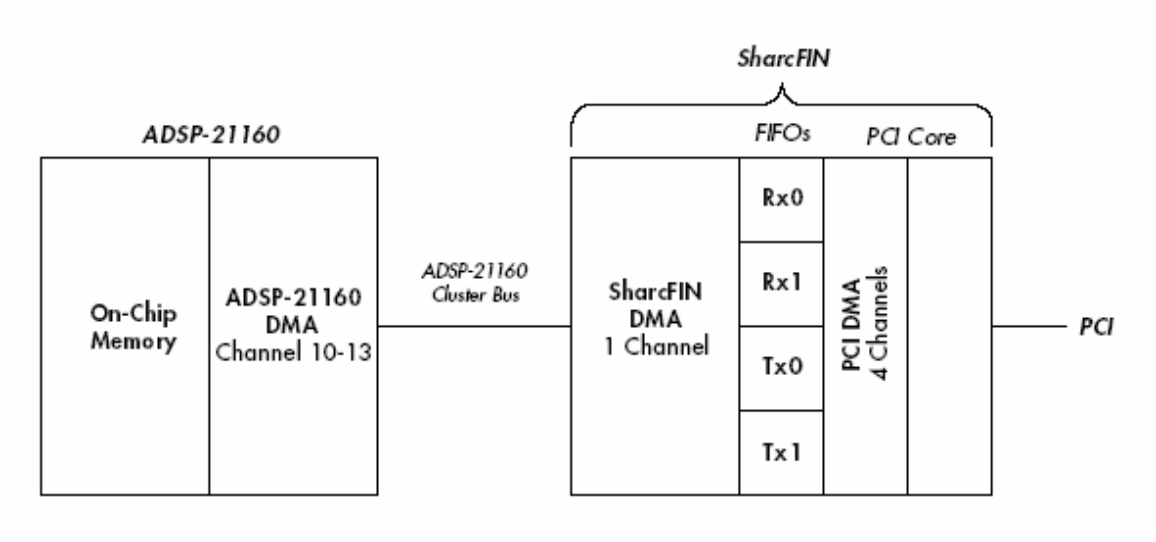


Figura A-3 Canales DMA dentro de la tarjeta HH-PCI.

Las transferencias DMA pueden también ocurrir entre la memoria interna de ADSP-21160 y sus puertos series o los puertos *link-ports*.

A.1.3 Multiprocesado usando External-Ports

El puerto externo (***external port***) soporta un espacio de direcciones unificado que permite a todos los procesadores accesos directos a la memoria interna de cada ADSP-21160M. El máximo throughput de transferencia de datos entre procesadores es de 320Mbytes/s sobre dicho puerto externo. La escritura en difusión o tipo *broadcast* permite la transmisión simultánea de datos a todos los ADSP-21160Ms y se puede utilizar para implementar semáforos reflexivos.

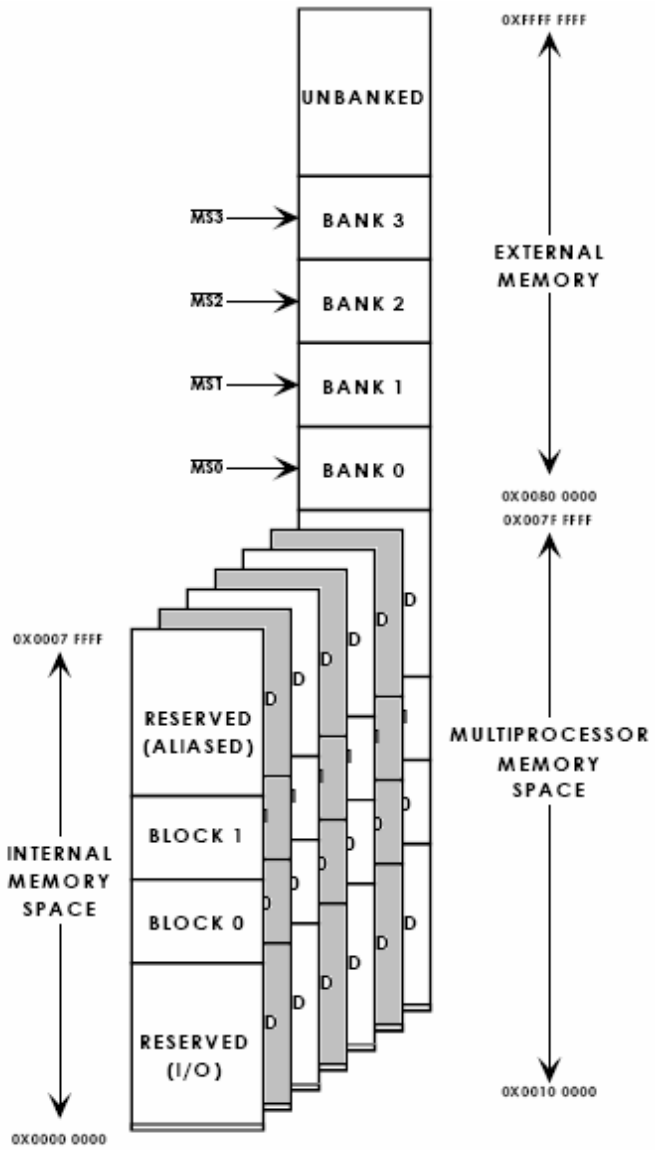
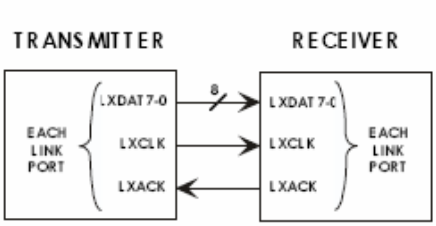


Figura A-4 Sistema de memoria en los procesadores ADSP21160

A veces, esta disposición se llama “cluster multiprocessing” porque el *cluster-bus* de la tarjeta DSP se utiliza para conectar los puertos externos de los DSPs.

A.1.4 Multiprocesado usando Link-Ports

Seis link ports proporcionan un segundo método de comunicaciones de multiprocesado a parte del puerto externo. Los link-port posibilitan comunicaciones entre DSPs y otros dispositivos de entrada y/o salida como cámaras CCD, sensores y otros dispositivos que se adapten a sus especificaciones (Figura A-5).



Link Port Pin(s)	Link Port Function
LxDAT7-0	Link Port x Data
LxCLK	Link Port x Clock
LxACK	Link Port x Acknowledge
"x" denotes the link port number, 0-5	

Figura A-5 Link Ports en el Procesador ADSP21160

Usando los link-ports se puede construir un sistema del multiprocesador siguiendo una arquitectura 2D o 3D. Los sistemas pueden utilizar los link-ports y el multiprocesado de cluster independientemente o de forma concurrente.

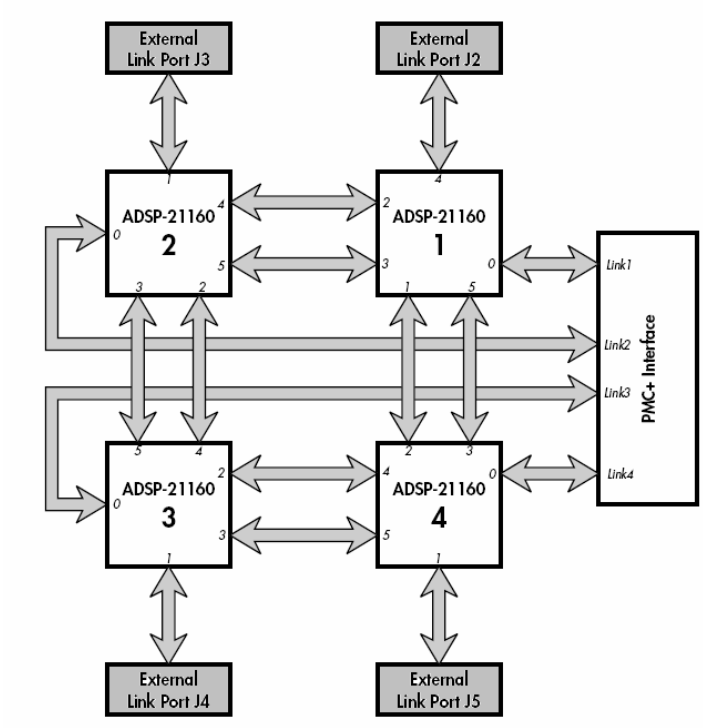
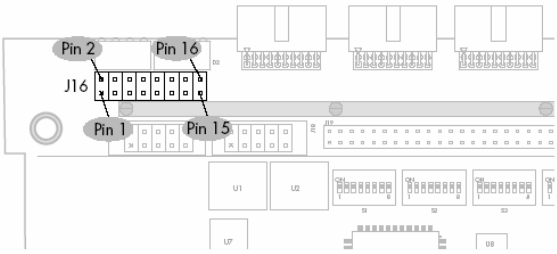


Figura A-6 Diagrama de bloques de las conexiones Link Port de la tarjeta HH-PCI.

A.1.5 Flags de E/S e Interrupciones

Cada ADSP-21160 tiene cuatro señales flags. Dos de esos flags están conectados con el SHARC*fin*. Los dos flags restantes de cada DSP se conectan, uno al LED de la tarjeta HH-PCI y el otro hacia otro DSP para comunicación entre procesadores (Figura A-8).



Pin	Signal	Description	Pin	Signal
1	HA1_F2	21160-1 FLAG2	2	GND
3	HA1_F3	21160-1 FLAG3	4	GND

Figura A-7 Posición en la tarjeta HH-PCI de los Flags de entrada/salida del ADSP-21160. Ejemplo del DSP1-FLAG2 y DSP1-FLAG3 en los LEDs de la tarjeta

Cada ADSP-21160 tiene tres interrupciones. Una interrupción está dedicada a la comunicación entre procesadores. Otra interrupción va al SHARC*fin*, donde se puede configurar su encaminamiento en el espacio de configuración del SHARC*fin*.

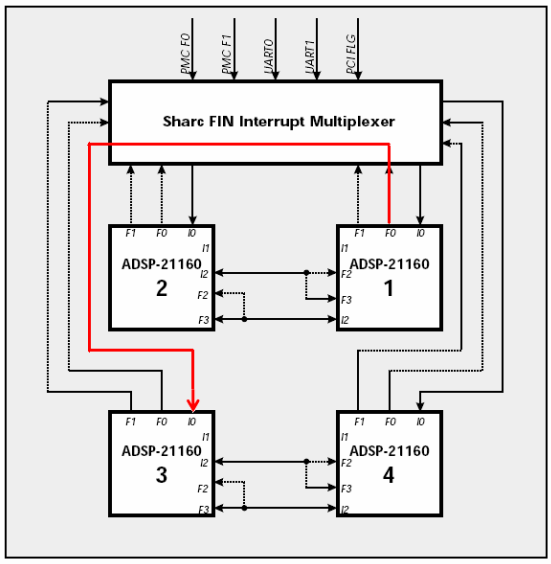


Figura A-8 Diagrama de bloques del conexionado de interrupciones y Flags. Ejemplo de encaminado de la interrupción DSP1- Int0 hacia el SHARCfin.

La Figura A-8 muestra el encaminamiento empleado en la aplicación *parallelFFT* del presente proyecto. En ella se ve como una interrupción del procesador HH1 está cortocircuitada con un flag de entrada del procesador HH3.

1.3.6 Puertos Series

El ADSP-21160 ofrece dos puertos series síncronos que proporcionen un interfaz barato a una gran variedad amplia de dispositivos periféricos digitales y/o analógicos. Los puertos series pueden funcionar hasta la mitad de la frecuencia de reloj del núcleo, proveyendo cada uno de una tasa de datos máxima de 50 Mbit/s.

Las funciones independientes de transmisión y recepción proporcionan mayor flexibilidad para las comunicaciones series. Los datos del puerto series se pueden transferir automáticamente a y desde la memoria interna vía un canal DMA dedicado.

SPORT0 Pins	SPORT1 Pins	Description
DT0	DT1	Transmit Data
TCLK0	TCLK1	Transmit Clock
TFS0	TFS1	Transmit Frame Sync
DR0	DR1	Receive Data
RCLK0	RCLK1	Receive Clock
RFS0	RFS1	Receive Frame Sync

Tabla A-2 Descripción de pines en los puertos series

En la tarjeta Hammerhead-PCI, un puerto serie de cada procesador está conectado con el bus serial TDM. Este bus es mostrado en la Figura A-9 e interconecta los siguientes módulos:

- los cuatro DSPs
- el interfaz de PMC+
- el interfaz RS-232

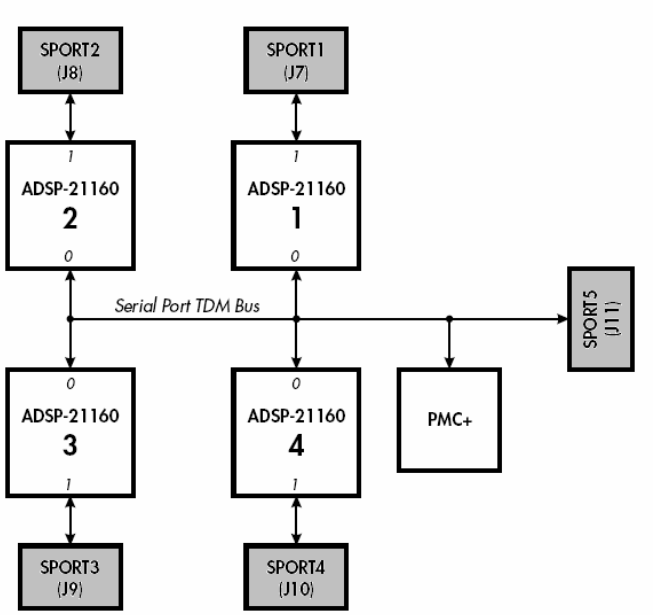


Figura A-9 Diagrama de bloques de conectores del puerto serie de la tarjeta HH-PCI

El puerto serie restante en cada procesador está conectado con uno de los cuatro puertos series externos de la tarjeta DSP.

Apéndice B
Entorno de desarrollo VisualDSP++

B.1 Introducción

En este apéndice se explica el manejo del entorno de desarrollo *VisualDSP++ v4.5* utilizado para desarrollar los programas *dspFDOCT* y *parallelFFT* definidos y diseñados en el presente proyecto.

VisualDSP++ es la herramienta software para el desarrollo de aplicaciones sobre los diferentes procesadores de la familia SHARC. Es compatible con *Windows*® y está compuesta básicamente por dos aplicaciones:

- ***Entorno Integrado de Desarrollo (IDE)***. Permite el acceso a todas las actividades necesarias para crear proyectos a través de su compilador, enlazador (*linker*) y librerías de C y ensamblador.

- ***Depurador (Debugger)***. Incluye el simulador y el emulador.

Debido a estas dos aplicaciones, el entorno *VisualDSP++* es considerado como una herramienta IDDE (*Integrated Development And Debugging Environment*). El desarrollo de cualquier programa en *VisualDSP++* debe incluir los siguientes pasos:

1. Crear un nuevo archivo para el proyecto.
2. Escoger el procesador DSP.
3. Agregar y editar los archivos fuente del proyecto.
4. Definir las opciones del archivo a crear.
5. Construir un archivo ejecutable .dxe (tipo de archivo para el depurador).
6. Depurar el proyecto (Debugger).

7. Construir la versión final del proyecto.

B.2 Entorno Integrado de Desarrollo (IDE)

Una vez instalado el *software* en el PC, se arrancará la aplicación y se observará un entorno como el mostrado en la Figura B-1.

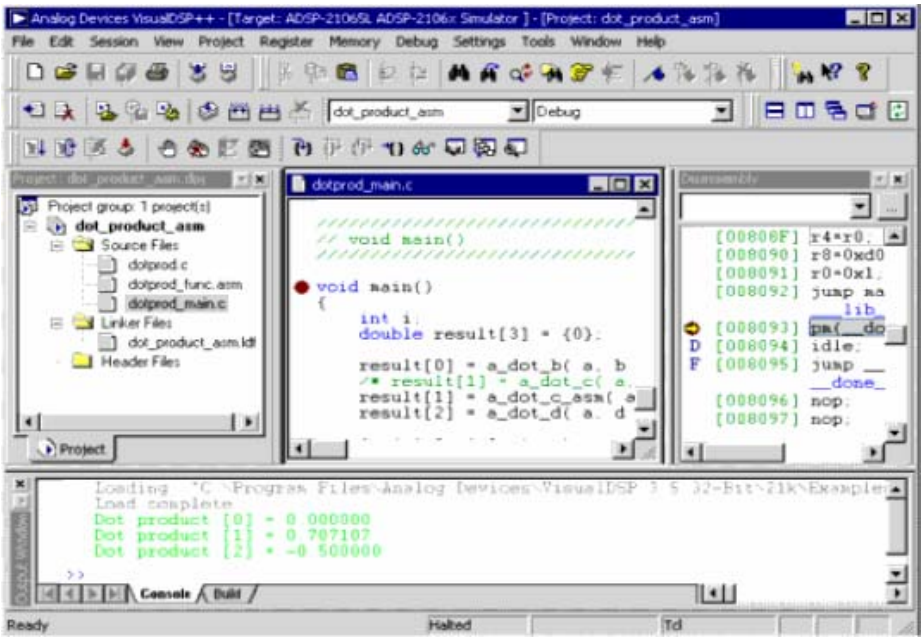


Figura B-1. Ventana principal del entorno de desarrollo VisualDSP++.

Se observan cuatro ventanas principales:

- A la izquierda arriba está el visor de proyecto, que muestra un árbol con los archivos que componen el proyecto.

- La ventana inmediatamente inferior, es la ventana de mensajes, en la que el entorno plasma toda la información acerca del compilador y enlazador.

- La ventana central es la de desarrollo, en ella se escriben los programas. Se pueden abrir tantas ventanas como se desee de este tipo, y tener los archivos abiertos para su modificación.

- La ventaza de la derecha es la ventada de desensamblado. Permite examinar el código ensamblador generado por el compilador de C/C++.

- En la parte superior se puede ver una barra de herramientas y un menú.

Cabe destacar que el enlazador de *VisualDSP++* permite combinar distintos tipos de código para obtener un único ejecutable. Por ejemplo, la Figura B-1 muestra varios ficheros C junto con otro .asm dentro de un mismo proyecto.

Esto debe ser incluido en un archivo de tipo .ldf (Linker Description File) que junto con la arquitectura de memoria de los procesadores definen el mapeado de memoria en el sistema destino (Figura B-2).

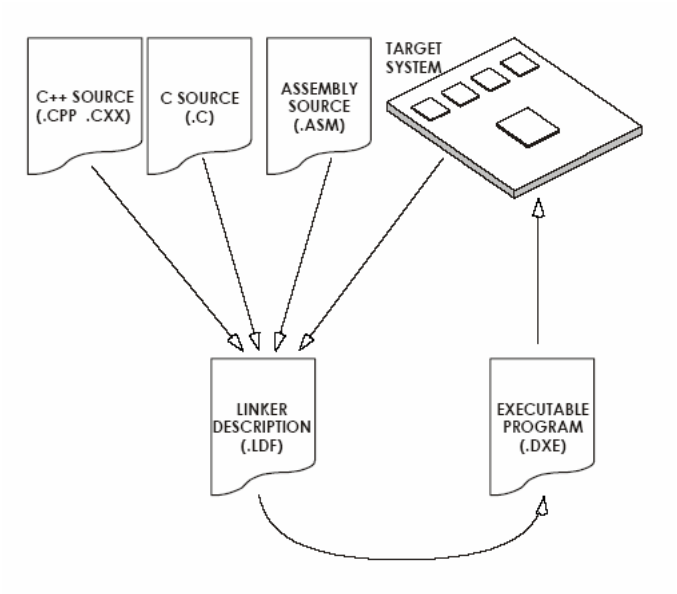


Figura B-2 Enlazador VisualDSP++ de Analog Device

B.3 Depurador

VisualDSP++ suministra un depurador en el mismo entorno de desarrollo. Con esta herramienta es posible hacer dos cosas principalmente; un programa y depurarlo.

VisualDSP++ puede conectar con un número diferente de sesiones de depuración, donde cada sesión tiene su propia aplicación y ventajas. Los tipos de sesión disponibles con *VisualDSP++* son:

- **Simulador**. Basado en un modelo del software del procesador. Los simuladores ofrecen ventajas únicas, la primera es que no se requiere ningún hardware externo. Además, los simuladores dan una visión única de los modos de funcionamientos internos del procesador (pipeline, caches, y más), que no está posible con sesiones basadas en hardware. La desventaja es que un simulador es varias órdenes de magnitud más lento que el hardware real. El modelo software simula solamente el procesador, siendo difícil simular exactamente un sistema complejo que implique más de un procesador

- **Emulador**. Basado en el conector JTAG, es el dispositivo ideal para conectar con el hardware, dando el mejor rendimiento y la máxima flexibilidad. Este módulo separado del PC proporciona una conexión de gran ancho de banda entre el PC y el dispositivo que está siendo depurado. Actualmente, Analog Devices oferta emuladores basados en interfaces USB o PCI.

Se presenta en la Figura B-3, la ventana principal del depurador.

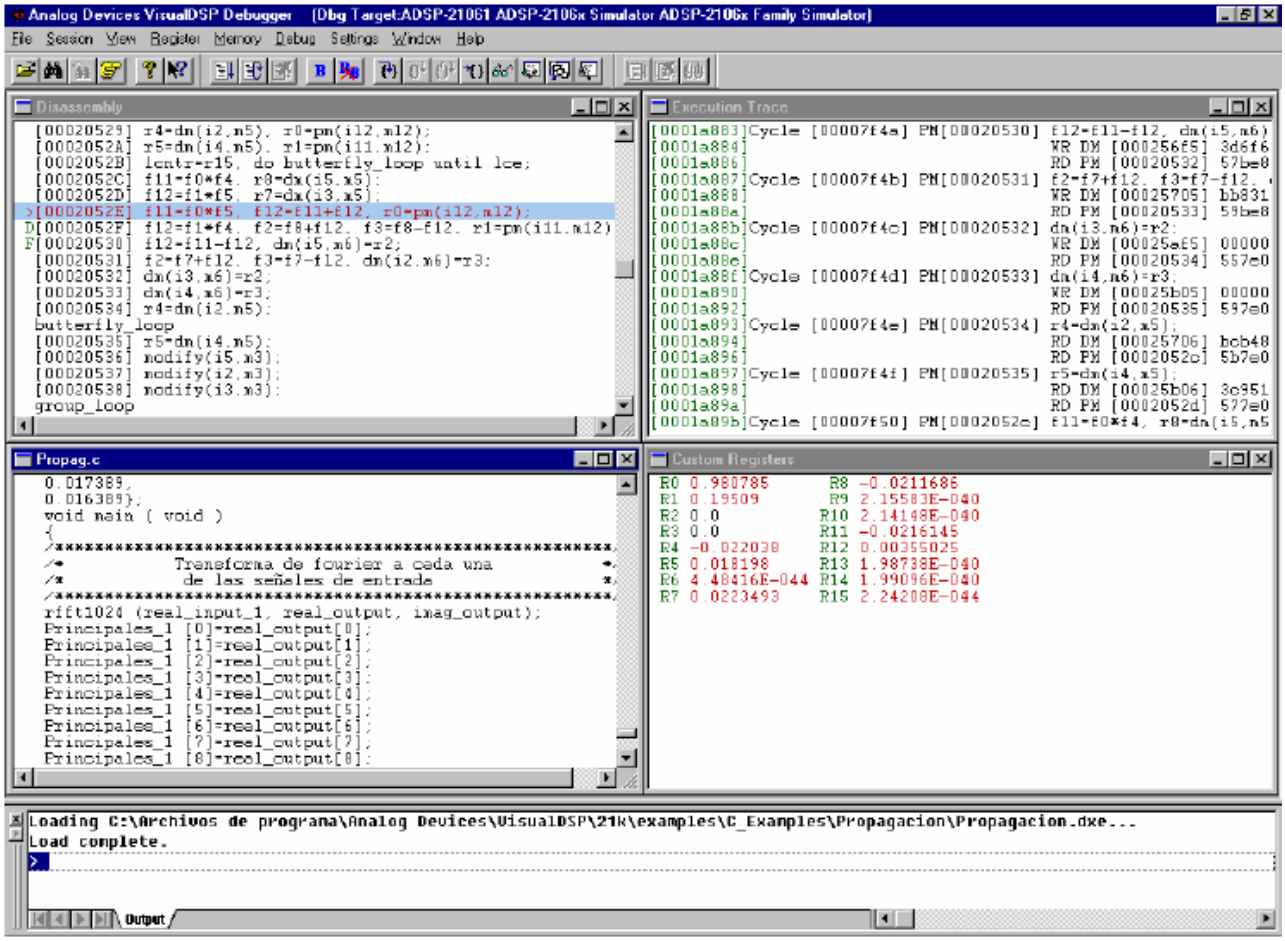


Figura B-3. Pantalla principal del depurador.

El entorno esta compuesto de ventanas, donde se pueden visualizar el contenido de la memoria o los registros, así como qué línea de código se está ejecutando.

En la Figura B-4 se muestra la barra de herramientas utilizada a la hora de depurar.

Figura B-4. Barra de herramientas del programa de depuración.

De izquierda a derecha se encuentran las siguientes funciones.

- *Run*, para iniciar la ejecución. Se puede optar por ejecutar el programa en bucle infinito.
- *Break*, para detener la ejecución del programa.

- *Reset* del programa, para cargarlo en memoria y reiniciar el contador de programa.
- Los cuatro botones siguientes se utilizan para poner puntos de ruptura en el programa.
- Los cuatro botones siguientes son estilos de ejecución paso a paso.
- Los cuatro botones siguientes sirven para visualizar el contenido de la memoria o los registros.

En el entorno multiprocesador, se encuentra las funciones básicas con el mismo significado pero referidas a cada uno de los procesadores activos en la sesión de depuración.

Apéndice C
Programa de diagnóstico interactivo Diag21k

C.1 Introducción

Este apéndice describe cómo utilizar la herramienta de diagnóstico Diag21k de Bittware.

El entorno de Diag21k está basado en caracteres, es decir, su interfaz de usuario consiste en un intérprete de comandos interactivo. Diag21k permite descargar programas al DSP, comenzar y parar su ejecución y acceder a la memoria del DSP. También proporciona capacidades de depuración a nivel de ensamblador, incluyendo breakpoints, realizar pruebas de memoria y ver las posiciones de memoria específicas en una gran variedad de formatos. Este apéndice describe

- las capacidades de programación secuencial de Diag21k (script)
- el depurador incorporado en Diag21k
- los comandos Diag21k disponibles

C.2 Secuencia de comandos (script)

Las secuencias de comandos (script) en Diag21k es la manera de automatizar tareas repetitivas como por ejemplo la comprobación de las comunicaciones con la tarjeta DSP antes de implementar un programa en el PC anfitrión. Los scripts de Diag21k son sólo archivos de texto plano que contienen comandos de Diag21k, uno por línea. Por ejemplo, el siguiente script simplemente resetea la tarjeta y configura el procesador,

```
-- reset board and configure processor
br
pc
```

C.2.1 Uso de constantes

Diag21k almacena una lista de las constantes que se pueden utilizar en la línea de comandos. Cuando se analiza el comando introducido, las constantes son substituidas por sus valores. Una lista de todas las constantes y de sus valores en Diag21k se puede ver usando el comando "sc" (*show constants*). En el ejemplo siguiente, la constante SHARED_MEM_BUF contiene la dirección del último buffer de memoria reservado:

```
-- allocate host physical memory buffer
lpm 0x10000

-- write address to dsp variable
mw s _host_phys_addr SHARED_MEM_BUF
```

C.2.2 Uso de variables

Diag21k tiene la capacidad de almacenar la información de usuario bajo la forma de variables. Como sucede con las constantes, cuando el comando tecleado se analiza, las variables son substituidos por sus valores. Diag21k mantiene una lista interna de las variables de usuario que se puede ver usando el comando "set". Para indicar que se trata de variables se añade el símbolo de "$". Los ejemplos siguientes muestran la creación de variables de distintos tipos:

```
-- create or set a variable from an immediate value
set $i 10
set $address 0x50020
set $str bli21160.dxe
```

Las variables se pueden utilizar como parámetros en casi todos los comandos de Diag21k. Por ejemplo,

```
-- read $count locations starting at address $addr
mr s $addr $count
```

El lugar más ventajoso para utilizar variables es dentro de las declaraciones de sentencias *if* y *while*.

C.2.3 Las sentencias *If* y *While*

Las sentencias *If* y *While* proporcionan control de flujo dentro de los scripts de Diag21k. Ambas sentencias comprueban un valor inmediato o el resultado de una expresión para elegir qué comandos ejecutar dentro de la declaración de *If* o si el bucle *While* continúa ejecutándose:

```
-- if DSP's error is negative, exit Diag21k
if (mr s _error) < 0 {echo Error!; x}

-- wait until count is at least 100 before continuing
while (mr s _count) < 100 {sp 1}
```

C.2.4 Ejemplo de script: run_sys.cmd

En el CD-ROM adjunto se incluye un script de Diag21k que sirve para probar la aplicación multiprocesador *dspFDOCT* en una tarjeta Hammerhead-PCI. Suponiendo que el directorio local MYDIR contiene los ejecutables HH1_Generator.dxe, HH2_Consumer.dxe y HH4_Consumer.dxe, ejecutamos en la consola de comandos de MS-DOS la siguiente orden:

```
C:\MYDIR\>diag21k -xrun_sys
```

Esta prueba implica 3 procesadores:

- PROC1 – Es el generador de los datos, el ritmo de transmisión es establecido por un timer, se envían paquetes de datos vía link-ports hacia el PROC2 y PROC4. Este procesador también inicializa los datos de la SDRAM escribiendo un patrón de incremento antes de comenzar las transferencias.

- PROC2 – Es el receptor de una porción de los datos enviado por PROC1 vía link ports.

- PROC4 – Es el receptor de una porción de los datos enviado por PROC1 vía link ports.

A continuación se detalla el contenido de dicho script:

```
-- reset board, select proc, configure processor
br
bs 3
br
pc

-- clear an area of internal memory --
echo
echo Clearing internal mem buffer on PROC3
mw ih 0x58000 0 0x1000
os pause

-------------------------------------------
-- Load and start PROC2, consumer
-------------------------------------------
bs 2
pc
fl debug\hh2_consumer.dxe
mw li _Frame_Size 32
ps

-------------------------------------------
-- Load and start PROC4, consumer
-------------------------------------------
bs 4
pc
fl debug\hh4_consumer.dxe
mw li _Frame_Size 32
ps

-------------------------------------------
-- Load and start PROC1, the generator
-------------------------------------------
bs 1
pc
fl debug\hh1_generator.dxe
mw li _Line_Freq 5000
mw li _Frame_Size 32
ps

-------------------------------------------
-- Monitor PROC1 for completion
-------------------------------------------
echo Image ready?
os pause
mw li _input 1

echo Image ready?
os pause
mw li _input 1

echo Hit a key to stop:
os pause
mw li _done 1

-------------------------------------------
-- Dump PROC1 status variables
-------------------------------------------
bs 1
echo PROC1 variables offset and done:
mr lh _offset
mr li _frame_cnt
os pause

-------------------------------------------
-- View PROC2 data
-------------------------------------------
bs 4
echo Link data on PROC2. beginning of rcvd_data[0]:
mr lh _rcvd_data 20
echo For end of rcvd_data[1],
os pause
mr lh _rcvd_data 0x2af
os pause
bs 2
echo Processing Times:
mr lh _stat 16
os pause
```

```
-----------------------------------------
-- Check DMA metrics
-----------------------------------------
bs 1
echo STAT (TIMER INTR/DMAC10/DMAC11/LINK2/LINK3):
mr lh _stat 30

-- exit diag21k without a reset --
x
```

C.3 Depurador embebido

Diag21k contiene un depurador a nivel ensamblador para eliminar errores en el código de un único procesador SHARC. Por esta razón no es adecuado para una aplicación multiprocesador como la nuestra en la que se empleará el depurador de *VisualDSP++*. No obstante para pequeñas comprobaciones este intuitivo depurador resulta muy útil.

El depurador de Diag21k puede ejecutar y parar el programa, avanzar una instrucción, fijar y quitar breakpoints, y leer y escribir los registros internos. Puesto que el depurador está embebido directamente en el entorno Diag21k, la mayor parte de las capacidades de Diag21k están aun disponibles durante la depuración, incluyendo scripting, acceso a los registros de la tarjeta, memoria, registros IOP, etc.

C.3.1 Arranque del depurador

Para lanzar el depurador, se utiliza el comando **`debug`**. Un programa puede estar ya cargado, o se puede cargar después de usar el comando **`debug`**. El depurador responderá mostrando el contador de programa y las posiciones de memoria circundantes. La depuración puede entonces ser parada en cualquier momento usando de nuevo el comando **`debug`**, reseteando la tarjeta (**`br`**), o saliendo de Diag21k (**`x`**). La ayuda para los comandos del depurador puede ser exhibida usando el comando **`dh`**.

```
diag21k[1]>fl prm21160.dxe
"c:\dsp21ksf\etc\prm21160.dxe" loaded

diag21k[1]>debug
Debugger initialized.

diag21k[1]>
        40002=0000:0000:0000          nop;
        40003=0000:0000:0000          nop;
        [___lib_RSTI]
        40004=0000:0000:0000          nop;
->      40005=063e:0004:009f          <- jump ___lib_start;
        40006=0000:0000:0000          nop;
        40007=0000:0000:0000          nop;

diag21k[1]>
```

C.3.2 Mostrando la posición del contador de programa

En cualquier momento durante una sesión de depuración, se puede utilizar el comando **`dd`** para mostrar el contador de programa actual y las posiciones de memoria circundantes. El comando **`dd 20`** por ejemplo, exhibirá 20 líneas con el contador de programa colocado en la mitad de ellas. La posición del contador de programa actual es indicada por las flechas que señalan a la dirección (**`->`**) y a la instrucción en ensamblador (**`<-`**).

C.3.3 Comandos Reset y Restart

El comando **`reset`** es utilizado para realizar un reset del procesador. El contador de programa será situado al comienzo del vector de reset. Para realizar un reset de la tarjeta, se utiliza el comando **`br`** como de costumbre. Para recomenzar el programa utilice el comando **`restart`**. Este comando equivale al comando del **`reset`** seguido por el comando **`run.`**

C.3.4 Comandos de Depuración

- **Single-stepping**

La posibilidad de single-stepping a través del código se consigue usando el comando **`step`** o presionando la tecla **`F11`**. El contador de programa avanzará a la posición siguiente o aquella que apunte la instrucción de salto si se da el caso.

- **Stepping over**

El depurador de Diag21k no puede realizar la función step over. En su lugar, cuando se presiona F10, se ejecuta la siguiente instrucción válida. Esto puede ser útil para saltar sobre instrucciones de salto, no obstante esta instrucción puede causar consecuencias indeseadas: el depurador esperará hasta que el contador de programa alcance la instrucción después de la instrucción de vuelta, pero el DSP puede que nunca consiga llegar hasta allí.

- **Running**

El comando **`run`** controla la ejecución del DSP y lo pone en modo free-run hasta que se introduzca un comando **`halt`** o bien que se alcance un breakpoint. Por tanto equivaldría al comando **`ps`** (*processor start*). Si al comando **`run`** se le da una dirección como parámetro, un breakpoint temporal será fijado en esa posición y el DSP ejecutará instrucciones hasta dicho breakpoint.

```
diag21k[1]>run main
Debugger running

diag21k[1]>
401fd=7c04:d731:0cd2 if le , r14<->s0;
401fe=887c:3dc2:09b5 if not bm r11=rot r5 by 0x709,
r11=dm(i4,m1);
401ff=1279:f9db:7364 stkyy=pm(0xf9db7364);
[_main]
[seg_pmco]
->40200=1607:ffff:fff0<- modify (i7,0xfffffff0);
40201=ad0f:ffff:ffef dm(0xffffffef,i6)=r15;
40202=0f02:0000:0000 r2=0;

diag21k[1]>
```

- **Breakpoints**

Un breakpoint puede ser fijado o quitado usando el comando **`break`**.

- Para fijar un breakpoint se usa el comando **`break set`** con la dirección como parámetro.
- Para eliminarlo se usa el comando **`break`** junto con la dirección del breakpoint específico.

El comando **`break remall`** quitará todos los breakpoints. Los breakpoints existentes se denotan con un asterisco (*) cuando se muestra el código ensamblador usando el comando **`dd`** o el comando **`mr`** (*memory read*).

```
diag21k[1]>break set main
Breakpoint added at 0x040200

diag21k[1]>mr p 0x401ff 5
[000401ff] = 1279:f9db:7364 stkyy=pm(0xf9db7364);
[_main]
[seg_pmco]
*[00040200] = 1607:ffff:fff0 modify (i7,0xfffffff0);
[00040201] = ad0f:ffff:ffef dm(0xffffffef,i6)=r15;
[00040202] = 0f02:0000:0000 r2=0;
[00040203] = 1102:0005:0001 dm(_num_reps)=r2;
diag21k[1]>
```

- **Lectura y escritura de registros**

El comando **uh** muestra todos los registros internos que pueden ser accedidos mediante el depurador de Diag21k. En la mayoría de los casos, todos los registros internos que se pueden acceder desde *VisualDSP++*, se pueden también alcanzar en Diag21k. No obstante, el acceso a las pilas internas no está disponible. Para leer un registro, se utiliza el comando **ur** y el nombre del registro. Para escribir un registro, se utiliza el comando **uw**, el nombre del registro y el valor hexadecimal a escribir.

C.4 Descripción de comandos

Command	Description
–	Comment (ignore remainder of line)
?	Display command list (help)
bch	Broadcast Command Help
bciw	Broadcast IOP Register Write
bcmw	Broadcast Memory Write
bcpc	Broadcast Processor Configure
bcpr	Broadcast Processor Reset
bcps	Broadcast Processor Start
bi	Board Info: display information about active board
br	Board Reset: hard-reset entire DSP board
break	Modify Breakpoints
cd	Change Directory
ds	DSP Select: specify active DSP
echo	Print a message to the screen
dd	Debug Display
debug	Toggle Debugging Mode
dh	Debug Help
dmard	Read from processor memory using DMA
dmawr	Write to processor memory using DMA
dn	DSP New: open a new DSP

Command	Description
dx	DSP close
fl	File Load: download DSP executable
fpm	Free Physical Memory buffer
fx	File Execute: run a command file
goto	Script Goto label statement
halt	Halt Debugger
ic	Interrupt Count
if	Script if statement
ih	IOP Help: display IOP register descriptions
ii	Install (/uninstall) interrupts
ir	IOP Read: display IOP register value
iw	IOP Write: set IOP register value
log	Start, stop, continue, or view a log file
lpm	Lock (allocate) Physical Memory buffer
mbd	Memory Buffer Dump
mbl	Memory Buffer Load
mbh	Memory Buffer Help
mbr	Memory Buffer Read
mbv	Memory Buffer View
mbw	Memory Buffer Write
mc	Memory Compare: read and compare address buffers
md	Memory Dump: read DSP memory and write output to file
mh	Memory Help: display memory command help screen
ml	Memory Load: read data from file and write to DSP memory
mm	Memory Map
mr	Memory Read
mt	Memory Test

Command	Description
mw	Memory Write
os	Operating System: open a command shell
pc	Processor Configure
pr	Processor Reset
ps	Processor Start
q	Quit Diag21k; reset all active boards
reset	Reset Debugger
restart	Restart Debugger
rr	Read Board Register
run	Run Debugger
rw	Write Board Register
rem	Comment (ignore remainder of line)
sc	Show Constants
set	View or modify board settings and Diag21k variables
sh	Script Help
sl	Symbol Load: load symbol table for a DSP executable
sp	Script Pause: pause script execution for a time
ss	Show Symbols: show information for loaded symbol(s)
step	Step Debugger
sw	Script Wait: pause script execution until variable equals a value
uh	Universal Register Help
ur	Universal Register Read
uw	Universal Register Write
while	Script While statement
x	Exit: quit Diag21k, leave processor(s) running

Apéndice D
Configurador de cámaras Matrox Intellicam

D.1 Introducción

Este apéndice explica el modo de funcionamiento de la cámara *AViiVA M2* de Atmel® detallando los parámetros más importantes del protocolo CameraLink. Para ello se recurre al frame-grabber *Solios XCL* de Matrox® que es la plataforma hardware para PC que se ha venido empleando hasta ahora para poder utilizar cámaras CameraLink en la tomografía FDOCT.

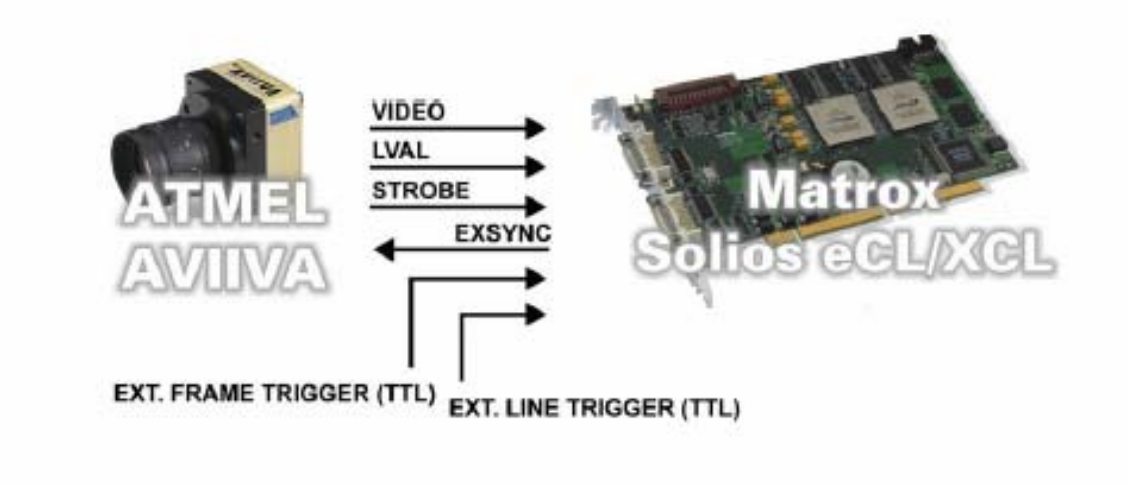


Figura D-1 Interfaz entre la cámara y el frame-grabber.

En concreto, veremos cómo conectar dos entradas TTL para que sean el *Line Trigger* y el *Buffer Trigger* en este tipo de frame-grabbers (Figura D-1) y la configuración software de éste usando la herramienta Matrox Intellicam.

D.1.1 Matrox Intellicam

Matrox Intellicam es un programa de alto nivel para Windows que proporciona conexión rápida de cámaras line-scan y acceso interactivo a todas las características y

funcionalidades de cualquier frame-grabber de Matrox. Para probar el conexionado, también permite grabar en diversos modos y visualizar los resultados inmediatamente.

Habitualmente el fabricante de cámaras CCD, Atmel® en nuestro caso, proporciona un archivo que configura el digitalizador del frame-grabber con los parámetros necesarios para una adquisición estándar.

También se puede utilizar Matrox Intellicam para crear y/o modificar estos archivos de formato de configuración del digitalizador (*Digitizer Configuration Format* o DCF) para aplicaciones de la cámara que no requieran una adquisición estándar, por ejemplo en el caso de la tomografía FDOCT.

Este fue nuestro caso: aunque se partió del DCF facilitado por Atmel® que configura la cámara en modo free-run, se llevaron a cabo los cambios oportunos para poder usarla en modo trigger.

D.2 Interfaz de Usuario

El interfaz de Matrox Intellicam es de tipo Múltiple Documento (MDI) y sigue las convenciones generales de un Interfaz Gráfico de Usuario de Windows (GUI).

La información detallada referente al interfaz de usuario se encuentra en la ayuda de Matrox Intellicam. La Figura D-2 muestra una ventana típica de Matrox Intellicam.

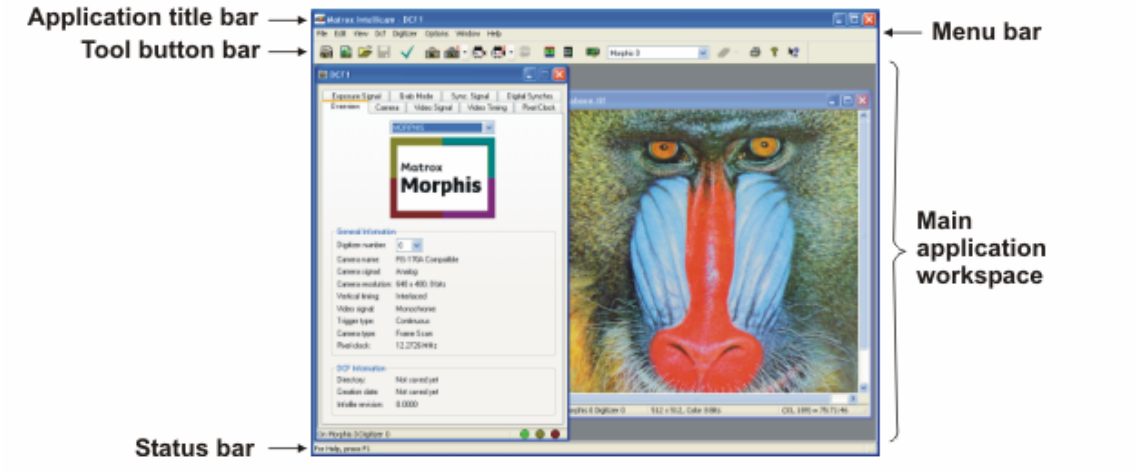
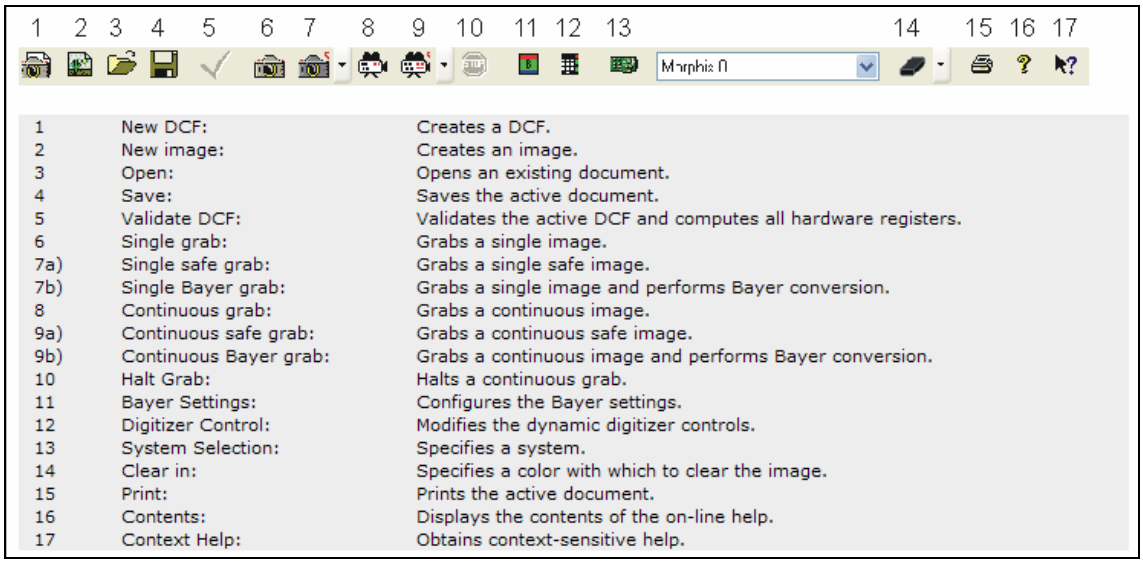


Figura D-2 Interfaz de usuario

Una vez cargado el archivo DCF correspondiente a la cámara conectada, se puede comenzar la adquisición pulsando *Single grab* (imagen estática) o *Continuous grab* (video). En la Figura D-3 se indica el significado de todos los comandos de la barra de herramientas.



1	New DCF:	Creates a DCF.
2	New image:	Creates an image.
3	Open:	Opens an existing document.
4	Save:	Saves the active document.
5	Validate DCF:	Validates the active DCF and computes all hardware registers.
6	Single grab:	Grabs a single image.
7a)	Single safe grab:	Grabs a single safe image.
7b)	Single Bayer grab:	Grabs a single image and performs Bayer conversion.
8	Continuous grab:	Grabs a continuous image.
9a)	Continuous safe grab:	Grabs a continuous safe image.
9b)	Continuous Bayer grab:	Grabs a continuous image and performs Bayer conversion.
10	Halt Grab:	Halts a continuous grab.
11	Bayer Settings:	Configures the Bayer settings.
12	Digitizer Control:	Modifies the dynamic digitizer controls.
13	System Selection:	Specifies a system.
14	Clear in:	Specifies a color with which to clear the image.
15	Print:	Prints the active document.
16	Contents:	Displays the contents of the on-line help.
17	Context Help:	Obtains context-sensitive help.

Figura D-3 Barra de herramientas (toolbar) de Intellicam

D.3 Parámetros más importantes

Los archivos DCF (*Digitizer Configuration Format*) son usados para conectar cualquier cámara al frame-grabber.

Para conectar al frame-grabber *Solios XCL* dos entradas TTL para que sean el Line Trigger y el Buffer Trigger de la cámara *AViiVA M2* se crea un archivo DCF con los parámetros que se muestran a continuación.

D.3.1 Configuración de Camera Link

La configuración base del protocolo CameraLink (Figura D-4) se utiliza porque solamente un solo cable CL es requerido por esta cámara (hoy en día el coste de 5 metros de cable es cercano a 300€).

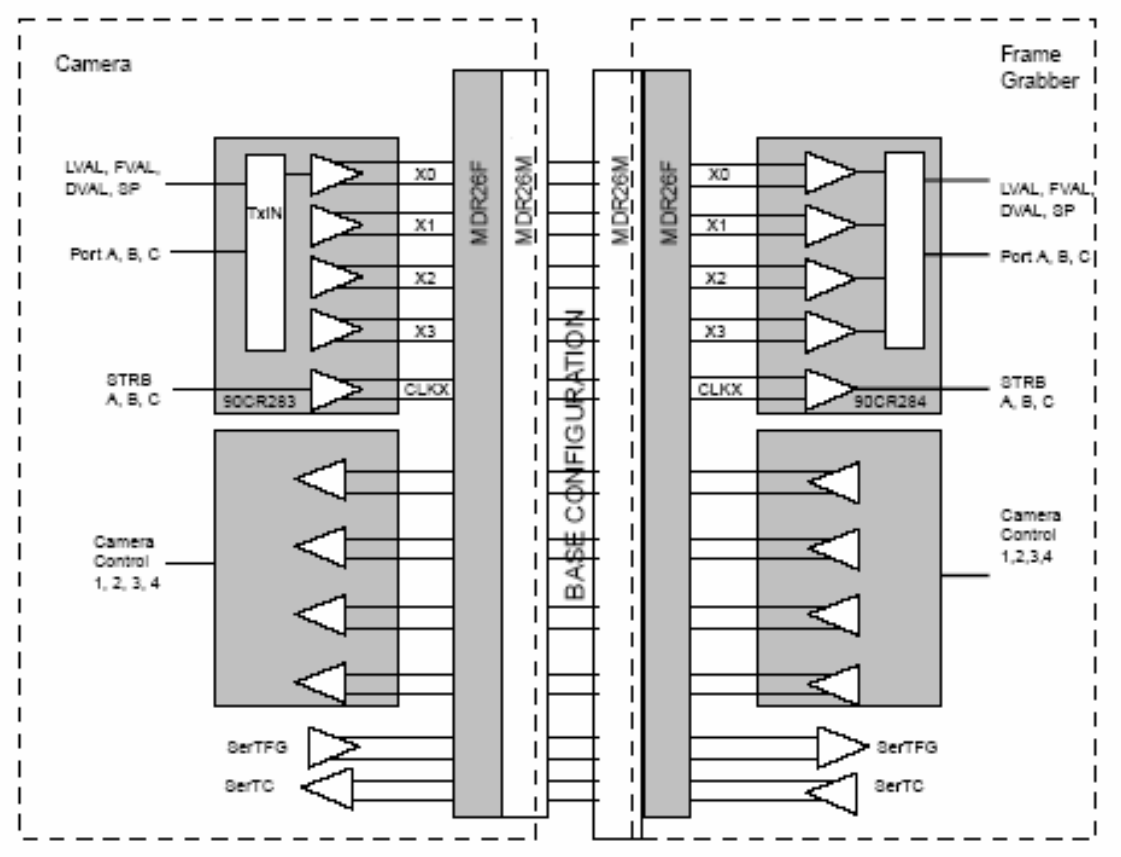


Figura D-4 Cable CL (configuración base)

En esta configuración los puertos A, B y C están disponibles (Figura D-4). La Tabla D-1 muestra cómo empaquetar dos píxeles de 10 bits en esos tres puertos de 8 bits.

Port A		Port B		Port C	
Port A0	A0	Port B0	A8	Port C0	B0
Port A1	A1	Port B1	A9	Port C1	B1
Port A2	A2	Port B2	nc	Port C2	B2
Port A3	A3	Port B3	nc	Port C3	B3
Port A4	A4	Port B4	B8	Port C4	B4
Port A5	A5	Port B5	B9	Port C5	B5
Port A6	A6	Port B6	nc	Port C6	B6
Port A7	A7	Port B7	nc	Port C7	B7

Tabla D-1 Asignación de bits dentro de los píxeles (configuración base)

Por otra parte todas las transferencias de datos se hacen en 32 bits (octetos X0, X1, X2 y X3 en la Figura D-5) así que podríamos elegir cómo empaquetar esos 24 bits en el paquete estándar de transferencia (ver zona coloreada en la Figura D-5a).

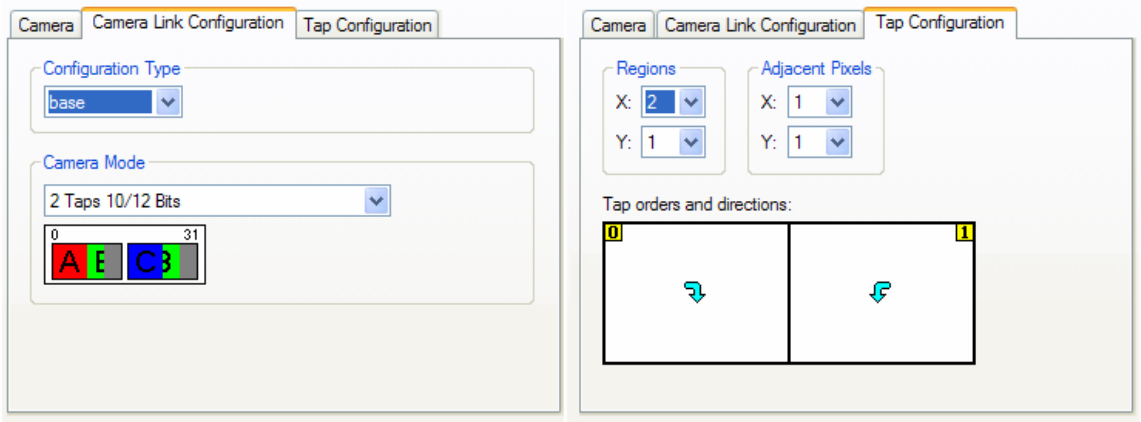


Figura D-5 Configuración CameraLink (a). Configuración de Tap (b) Intellicam

La Figura D-5b muestra la correspondencia espacio-temporal de cada par de píxeles. Las flechas azules indican la posición siguiente del par de píxeles a lo largo de los 512 ciclos de reloj y a lo largo de las 1024 celdas de la línea.

D.3.2 Configuración de la Cámara

Primeramente el tipo de cámara debe ser configurada: Line o Frame-Scan Camera.

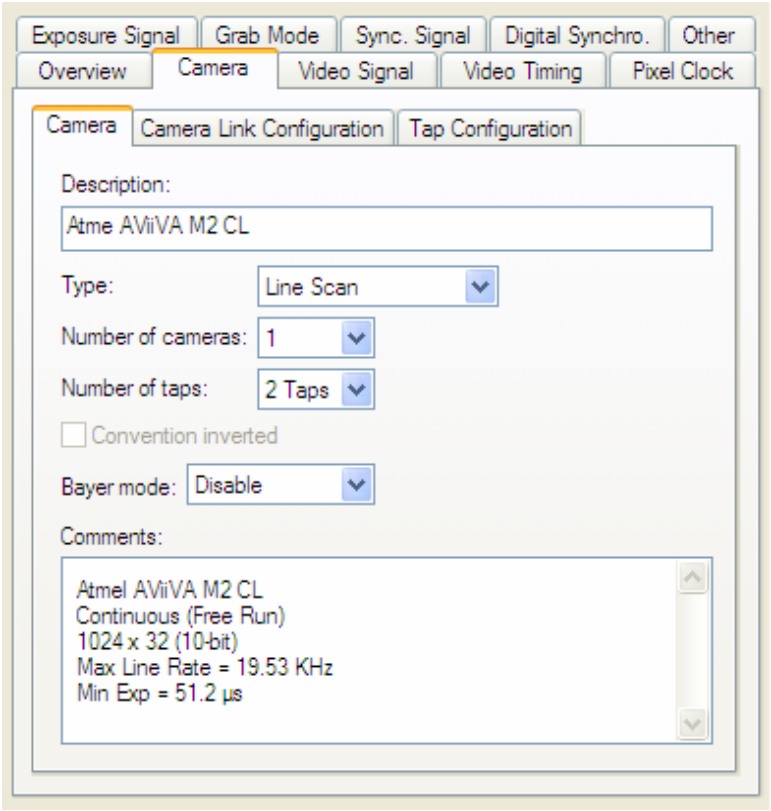


Figura D-6 Configuración de la Cámara (Intellicam)

D.3.3 Tamaño del Frame

El tamaño del frame se configura través de la ventana *Video Timing*. El campo llamado *image size X* (tamaño X de la imagen) es el número de píxeles por tap. Debido a que la cámara tiene 1024 píxeles y dos taps, este campo debe ser igual a 512.

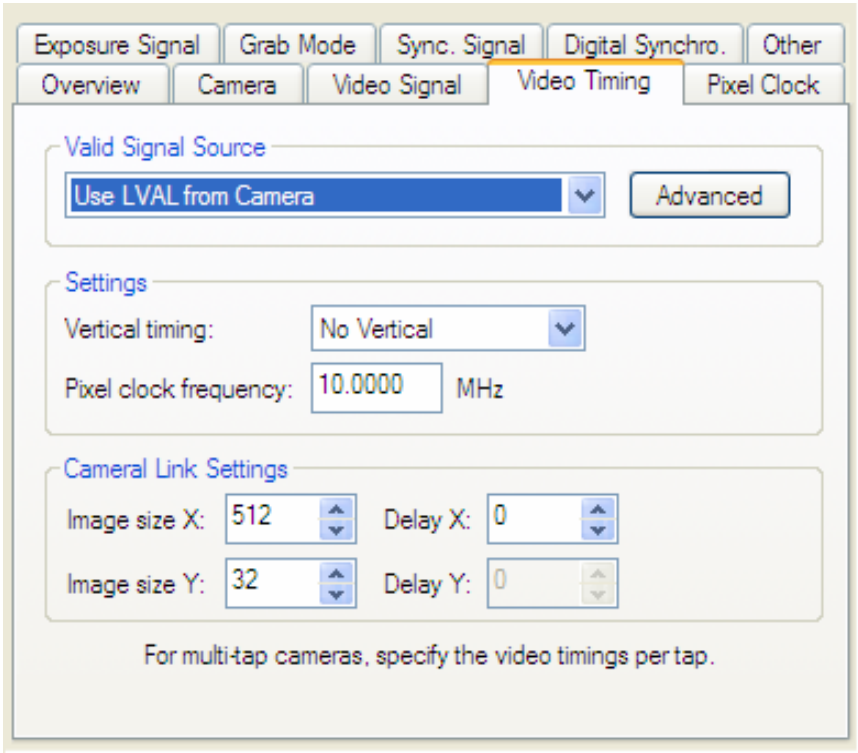


Figura D-7 Temporización de Video (Intellicam)

El número de líneas por frame es variable. El frame va a ser almacenado en un área reservada de la RAM del PC. Su tamaño limita el tamaño máximo del frame.

$$\text{Frame Size} = 1024 \times 32 = 2 \text{ Kbytes} \times 32 = 64 \text{ Kbytes} << 512 \text{ Mbytes}$$

D.3.4 Modo de Grabación

La configuración del *Buffer Trigger* se hace a través de la ventana *Grab Mode*. En nuestro caso el frame-grabber está esperando un disparador externo configurado en la entrada *INPUT 1* del conector DB9.

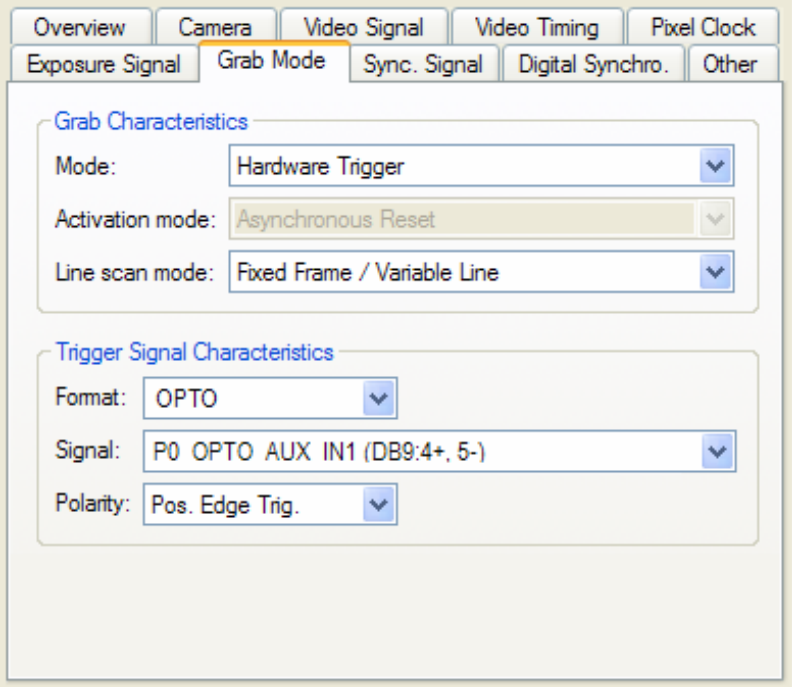


Figura D-8 Modo de Grabado (Intellicam)

El campo *Line Scan Mode* debe estar siempre configurado como *Fixed Frame*:

- Fixed Frame / Fixed Line: *line-rate* fijo con Frame Trigger (señal de exposición periódica)
- Fixed Frame / Variable Line: *line-rate* variable con Frame Trigger (señal de exposición controlada por trigger)

D.3.5 Señal de Exposición

La configuración del *Line Trigger* se hace a través de la ventana *Exposure Signal*. En nuestro caso el frame-grabber está esperando un disparador externo configurado en la entrada *INPUT 0* del conector DB9

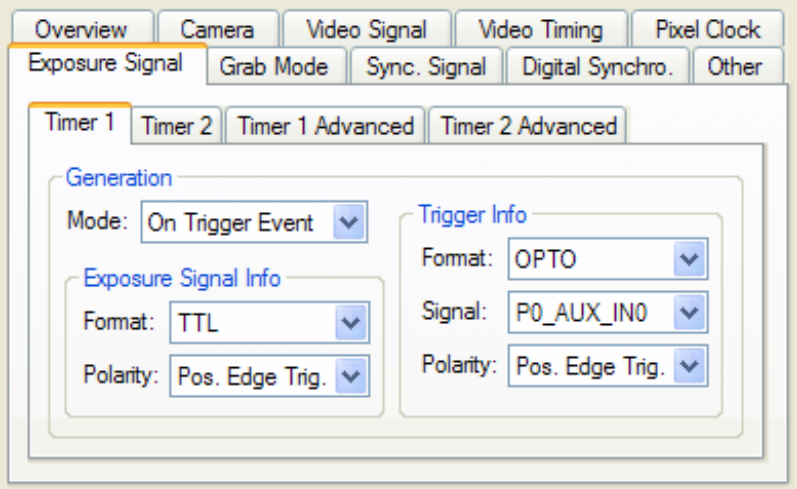


Figura D-9 Señal de Exposición (Intellicam)

El campo *Generation Mode* debe estar siempre configurado como *On Trigger Event*:

- Periodic: line-scan rate fijo
- On Trigger Event: line-scan rate variable (*Line Trigger*)

Apéndice E
Manual de usuario del CD-ROM

E.1 Introducción

En este apéndice se describe la estructura de archivos contenidos en el CD-ROM que se adjunta. Así mismo este apéndice sirve de manual de usuario para los programas *dspFDOCT* y *parallelFFT* que se han definido y testeado en capítulos precedentes.

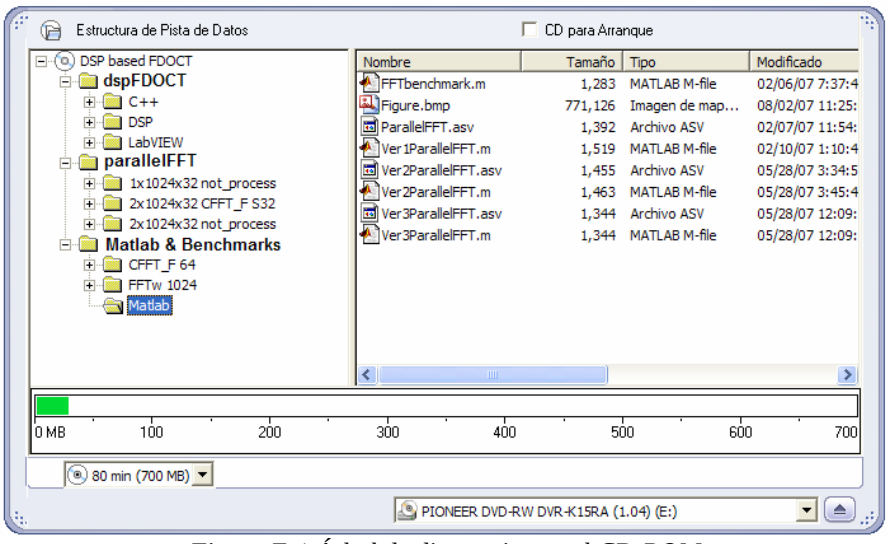


Figura E-1 Árbol de directorios en el CD-ROM

Para cada una de dichas aplicaciones existirán dos implementaciones para ambas plataformas en la que la misma aplicación es ejecutada: el PC anfitrión y en los distintos procesadores de la tarjeta DSP.

La implementación del PC anfitrión se desarrolló en C++ y LabView y ha sido discutida en los capítulos anteriores. La implementación DSP ha sido desarrollada bajo el entorno VisualDSP++ v4.5 específico de Analog Devices®.

El CD-ROM posee el árbol de directorios mostrado en la Figura E-1. Si exploramos su contenido podemos observar tres carpetas:

- **Matlab&Benchmarks:**

\Matlab	Programa que simula la implementación paralela de la CFFT de 1024 puntos usando *células de procesado* (FFT de 64 puntos).
\CFFT_F 64	Medida del número de ciclos que emplea un DSP en completar una célula de procesado. *VisualDSP++ y diag21k de BittWare.*
\FFTw 1024	Medida del tiempo que emplea una estación Xeon @ 3.6 GHz en realizar la CFFT de 1024 puntos. *MS Visual C++, librería FFTw.*

- **dspFDOCT**

\DSP	Código y ejecutables para cada uno de los procesadores DSPs. *VisualDSP++*
\DSP \runsys.cmd	Script para comprobar la transferencia de datos entre procesadores, tiempos de procesado y resultados de una imagen patrón DC. *diag21k de BittWare*
\C++	Adaptación tipo consola MS-DOS del código anterior. Inicialización de la tarjeta DSP, gestión de errores, etc. *MS Visual C++, librería HIL.lib de BittWare*
\LabVIEW\MyDLL	Librería de funciones en C que son llamadas en LabView. *MS Visual C++, librería HIL.lib de BittWare*
\LabVIEW	Programa ***dspFDOCT.vi*** (véase cápitulo 9) para testear patrones más complejos. *LabView, librería MyDLL.dll*

- **parallelFFT**

Todos los siguientes subdirectorios están organizados de igual manera que la carpeta *dspFDOCT* (estructura \DSP, \C++, \LabVIEW, \LabVIEW\MyDLL). La única diferencia es que la aplicación cambia; ahora estamos tratando con *parallelFFT*.

\1x 1024x32 not_process	1ª etapa del algoritmo sin células de procesado
\2x 1024x32 not_process	algoritmo sin células de procesado
\2x 1024x32 CFFT_F S32	algoritmo completo, datos de salida en formato S32

E.2 Guía de dspFDOCT

En primer lugar se van a explicar todos los parámetros de los dos programas de alto nivel que se incluyen en la carpeta \dspFDOCT. Como se dijo con anterioridad están escritos en C++ y LabView respectivamente, pero ambos están basados en la misma versión del ejecutable para cada uno de los procesadores DSP.

Al final de la sección se incluye la llamada a la librería FFTw referida en el capítulo 3 (Estado del Arte en el Procesado FFT).

E.2.1 Implementación de *dspFDOCT* en el PC (MS Visual C++)

En el interfaz de usuario de la Figura E-2 se introducirá el tamaño de cada frame en *parejas de líneas* ya que la implementación de la FFT en el ADSP21160 está optimizada para que se procesen dos líneas al mismo tiempo.

```
Number of pairs of lines: 32

Two Lines Frequency: 1250 Hz (MAX_FREQ)
Number of Frames   : 4

4 frames of 32 pair of lines (64 lines per frame)
lines of 1024 pixels to be processed @ 5000 Hz

Processing frame... 1, 2, 3, 4

4 frames processed by DSP-board
pixel values of the last one:

2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
...

Line 1 : 804 cycles
Line 32: 723 cycles

Processing Time per Two Lines : 765 us (average)
Maximum Two Lines Frequency   : 1307 Hz
```

Figura E-2 Consola MS-DOS de dspFDOCT

Por este motivo, la frecuencia del *Line Trigger* ha de ser dividida entre dos antes de ser introducida por teclado.

Esto sería en el caso de que dispusiéramos de un solo procesador esclavo pero la tarjeta HH-PCI permite disponer de otro más, por lo que se ha de dividir entre 4 la frecuencia del *Line Trigger* que deseamos antes de introducirla como parámetro de entrada.

Si tomamos los parámetros de tiempo real especificados en el capitulo 9 de resultados de la aplicación *dspFDOCT*, vemos que la frecuencia máxima del *Line Trigger* es 5000 Hz y por tanto el valor de *Two Lines Frequency* es 1250 Hz.

N = 2 esclavos (una tarjeta HH-PCI)

throughput = 5000 FFTs/sec.	Latency = 800 us (2 × 2 lines)
→ Lines Period = 800 us	Data Flow = 2 × 4 KB ÷ 800 us = 10 MB/s

Este sería el pseudocódigo:

Host PC Code (MS Visual C++)

```
NUM_FRAMES = 4 frames          LINE_SIZE = 1024 pixels
FRAME_SIZE = 32 lines          LINE_FREQ = 5 KHz
sdram1 = 0x800000              sdram1'= sdram1 + Total_Size / 2

FOR ( frame_cnt = 0; frame_cnt < NUM_FRAMES; frame_cnt++ )

        download_image(processor1, sdram1)
        input = 1

        WHILE (!output)
        output = 0

        upload_image(processor2, rcvd[FRAME_SIZE/2])
        upload_image(processor4, rcvd[FRAME_SIZE/2])

done = 1
```

E.2.2 Implementación de *dspFDOCT* en el PC (LabVIEW)

En la Figura E-3 se observa el interfaz de usuario y en la E-4 el esquema del programa.

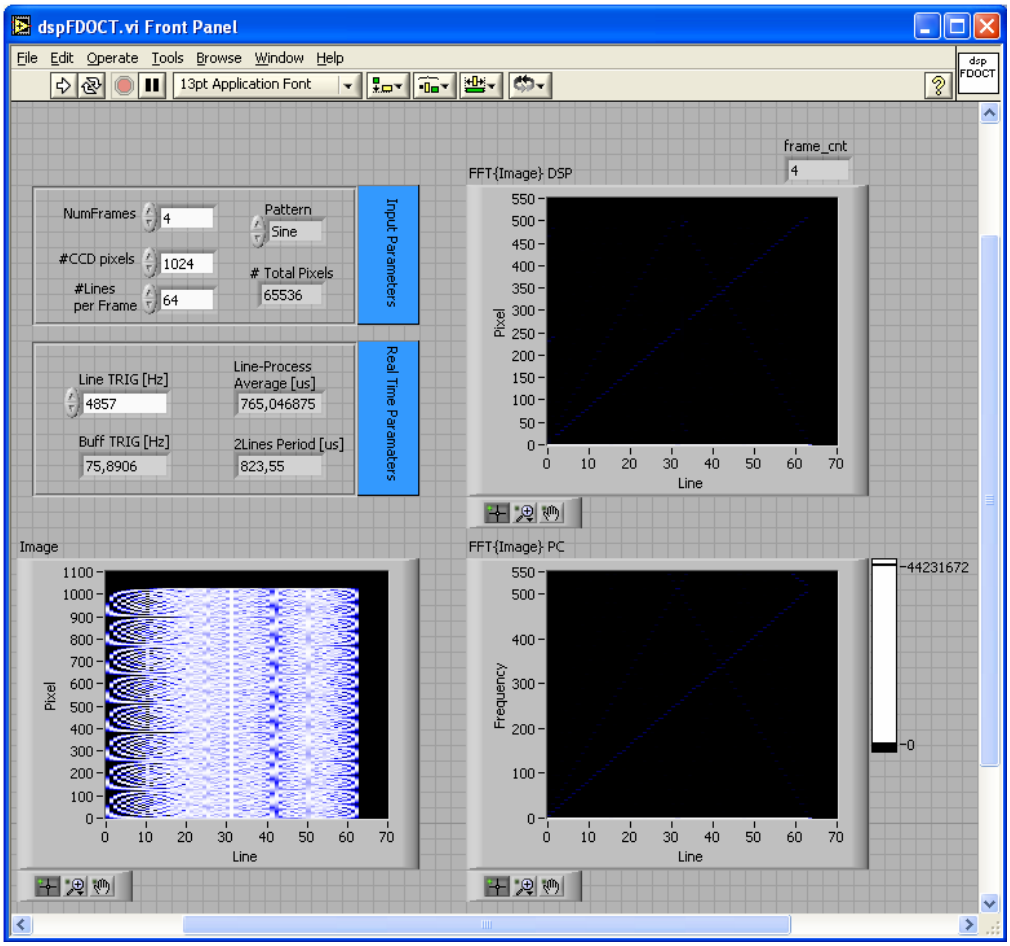


Figura E-3 Panel Frontal de dspFDOCT

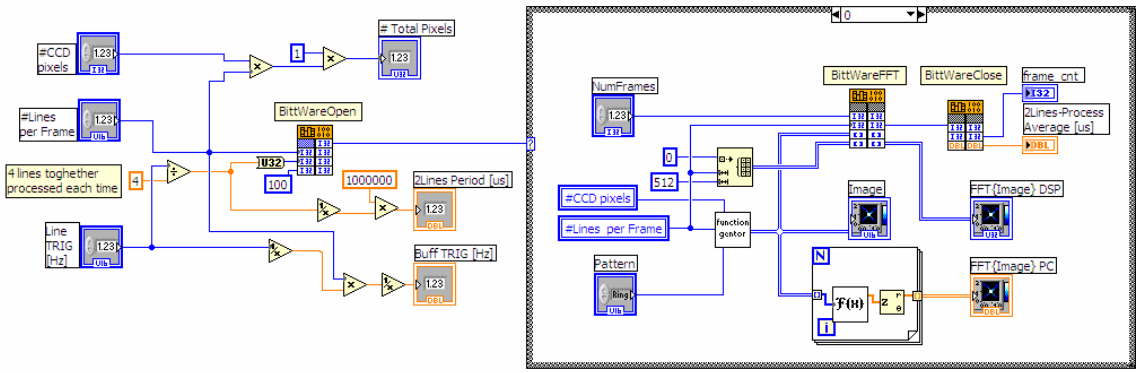


Figura E-4 Diagrama de Bloques de dspFDOCT

Controles

#CCD pixels número de píxeles en el array CCD de la cámara *line scan* (1024 píxeles)

#Lines per Frame número de líneas en cada frame (64 líneas)

Pattern contenido de cada línea de la imagen a tratar (valor constante, rampa, distintos tonos)

Line TRIG frecuencia del *Line Trigger* (4857 ≈ 5000 Hz)

NumFrames número de imágenes que van a ser cargadas por el Host PC para ser procesadas por la tarjeta DSP al pulsar *RUN*.

NOTA: Este control al final se reduce al número de veces que se aplica el algoritmo sobre la misma imagen ya que el PC anfitrión siempre cargará la misma imagen en la memoria de la tarjeta DSP.

Indicadores

#Total Pixel Número total de píxeles en un frame (1024 × 64)

2Lines Period Doble del período del *Line Trigger* ya que cada DSP calculará la FFT simultánea de 2 líneas.

2Line-Process Average Temporización del procesado simultaneo de dos líneas. Siempre habrá de cumplirse:

$$2Line\ Process\ Average \leq 2Lines\ Period$$

Buf TRIG Su período viene determinado por *#Lines per Frame* veces el periodo del *Line Trigger*

frame_cnt número de imágenes que han sido procesadas por la tarjeta DSP cuando la aplicación ha finalizado.

Diagrama de Bloques

En el diagrama de la Figura E-4 se ve cómo se realiza las llamadas a las funciones de la librería \LabView\MyDLL creada para inicializar la tarjeta DSP (BittWareOpen), cargarle varios frames y leer el último frame tratado (BittWareFFT) y cerrar la tarjeta (BittWareClose).

E.2.3 Implementación de *dspFDOCT* en los DSPs (VisualDSP++)

Comenzamos con el pseudocódigo del procesador maestro.

Master DSP Code (VisualDSP++)

```
FFT_SIZE   = 1024            LINE_SIZE = 1024 pixels
FRAME_SIZE = 32 lines        LINE_FREQ = 5 KHz
sdram1 = 0x800000            sdram1'= sdram1 + Total_Size / 2
line2[2][LINE_SIZE]          line4[2][LINE_SIZE]

DO

        WHILE (!input)
        input = 0

        Read_line(line2[sbuf], sdram1  + offset, BusMaster)        // first 2 lines
        Read_line(line4[sbuf], sdram1' + offset, BusMaster)        // 2 lines in middle
        Wait for timer complete

        DO
                Read_line(line2[sbuf], sdram1  + offset, BusMaster)
                Read_line(line4[sbuf], sdram1' + offset, BusMaster)

                Transmit_line(line2[lbuf], HH2_link_ports)
                Transmit_line(line4[lbuf], HH4_link_ports)

                offset += LINE_SIZE
                Wait for timer complete

        WHILE (1 < offset < FRAME_SIZE/2)

        Transmit_line(line2[lbuf], HH2_link_ports)
        Transmit_line(line4[lbuf], HH4_link_ports)                 // last transmission

        frames_count ++
        output = 1

WHILE (!done);
```

Pseudocódigo de cada uno de los procesadores esclavo.

Slave DSP Code (VisualDSP++)

```
                FFT_SIZE   = 1024            LINE_SIZE = 1024 pixels
                FRAME_SIZE = 32 lines        rcvd[FRAME_SIZE/2][LINE_SIZE]

                FOR (;;)

                   Receive_line(rcvd[rxbuf], local_link_ports)
                   Wait for reception complete

                   DO

                       Receive_line(rcvd[rxbuf], local_link_ports)
                       Process_line(rcvd[lbuf])
                       Wait for reception complete

                   WHILE (rxpkts < Frame_Size/2)

                   Process_line(rcvd[lbuf])
```

E.2.4 Llamada a la librería FFTW

```
#include "fftw3.h"
#include <time.h>

#define N 1024
#define ITERATIONS 1000

LARGE_INTEGER start_time;
LARGE_INTEGER stop_time;

int main(void)
{
       fftwf_complex *in, *out;
       fftwf_plan   p;

// Array allocation
in  = (fftwf_complex*) malloc(sizeof(fftwf_complex) * N);
out = (fftwf_complex*) malloc(sizeof(fftwf_complex) * N);

// Plan = data that FFTW needs to compute the FFT
p = fftwf_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE);

// FFTW execution
time_snap(&start_time);
for (i = 0; i < ITERATIONS; i++)
       fftwf_execute(p);
time_snap(&stop_time);
printf("Computation Time: ");
display_time(ITERATIONS, &start_time, &stop_time);

// De-allocation
fftwf_destroy_plan(p);
fftwf_free(in);
fftwf_free(out);

return 0;
}
```

E.3 Guía de parallelFFT

En primer lugar se van a explicar todos los parámetros de los dos programas de alto nivel que se incluyen en la carpeta \parallelFFT. Como se dijo al principio están escritos en C++ y LabView respectivamente, pero ambos están basados en la misma versión del ejecutable para cada uno de los procesadores DSP.

Al final de la sección se ha añadido el diseño Matlab del algoritmo simplificado usado en *parallelFFT*. Este fue probado para reducir al mínimo los problemas de depuración antes de su implementación en el procesador DSP.

También se incluye el programa usado para temporizar el número de ciclos que emplea un procesador DSP en completar una célula de procesado. Dicha medida fue analizada en el capítulo 12 de resultados de la implementación de la aplicación *parallelFFT*.

E.3.1 Implementación de *parallelFFT* en el PC (MS Visual C++)

En el interfaz de usuario de la Figura E-5 se introducirá el tamaño de cada frame por líneas en vez de parejas de líneas como en la versión anterior. De este modo la frecuencia que introduzcamos en el interfaz de usuario corresponderá a la frecuencia del *Line Trigger*.

```
Number of pairs of lines: 32

Lines Frequency:  3300 Hz (MAX_FREQ)
Number of Frames: 4

4 frames of 32 pair of lines
lines of 1024 pixels to be processed @ 3300 Hz

Processing frame... 1, 2, 3, 4

4 frames processed by DSP-board
pixel values of the last one:

2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
...

Line 1 : 804 cycles
Line 32: 723 cycles

Processing Time per Lines: 300 us (average)
Maximum Lines Frequency  : 3300 Hz
```

Figura E-5 Consola MS-DOS de parallelFFT

Este sería el pseudocódigo:

```
Host PC Code (Visual C++)

        NUM_FRAMES = 4 frames          LINE_SIZE = 1024 pixels
        FRAME_SIZE = 32 lines          LINE_FREQ = 3300 Hz

        sdram1 = 0x800000              sdram2 = 0x1000000

        FOR ( frame_cnt = 0; frame_cnt < NUM_FRAMES; frame_cnt++ )

                download_image(sdram2)
                input = 1

                WHILE(!output)
                output = 0
                upload_image(sdram2)

        done = 1
```

E.3.2 Implementación de *parallelFFT* en el PC (LabVIEW)

En la Figura E-6 se observa el interfaz de usuario y en la E-7 el esquema del programa.

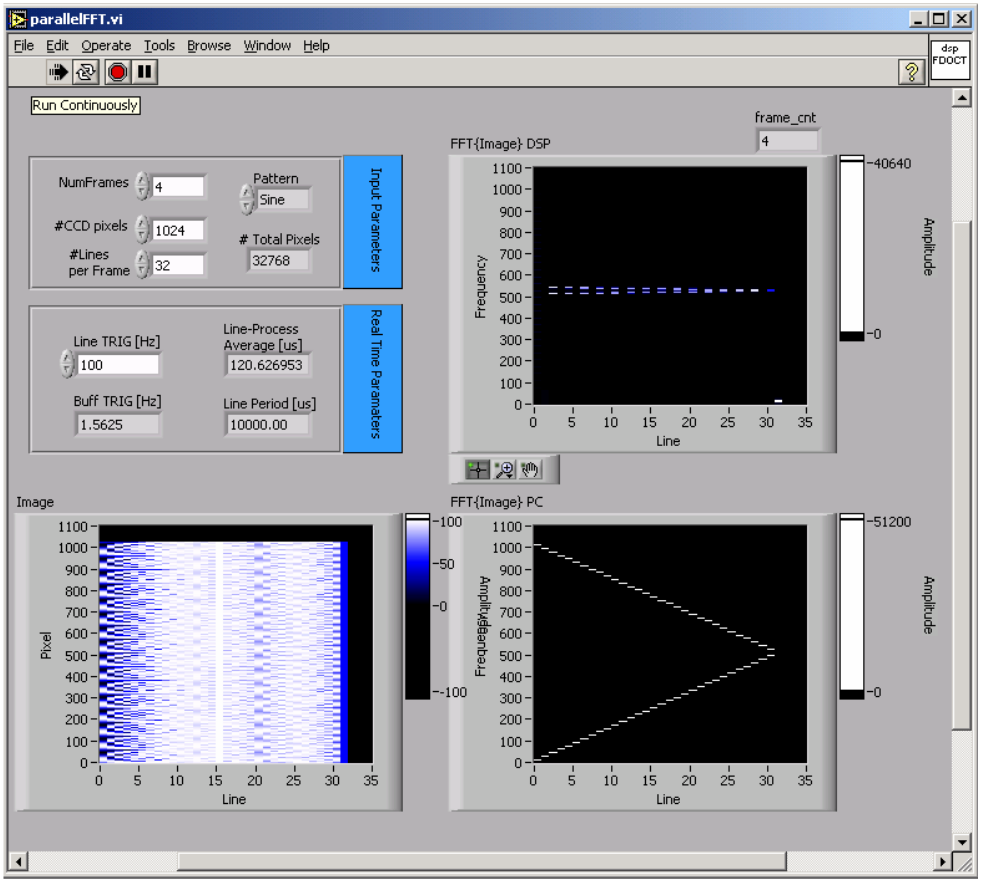


Figura E-6 Panel Frontal de parallelFFT

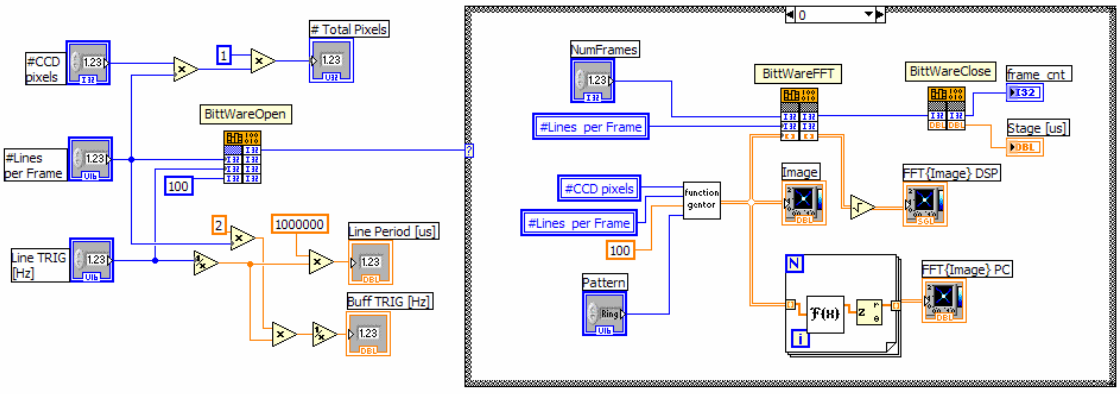


Figura E-7 Diagrama de Bloques de parallelFFT

Controles

#CCD pixels número de píxeles en el array CCD de la cámara *line scan* (1024 píxeles)

#Lines per Frame número de líneas en cada frame (32 líneas)

Pattern contenido de cada línea de la imagen a tratar (valor constante, rampa, distintos tonos)

Line TRIG frecuencia del *Line Trigger* (3300 Hz)

NumFrames número de imágenes que van a ser cargadas por el Host PC para ser procesadas por la tarjeta DSP al pulsar *RUN*.

NOTA: Este control al final se reduce al número de veces que se aplica el algoritmo sobre la misma imagen ya que el PC anfitrión siempre cargará la misma imagen en la memoria de la tarjeta DSP.

Indicadores

#Total Pixel Número total de píxeles en un frame (1024 × 32)

Lines Period Período del *Line Trigger*

Line-Process Average Temporización del procesado de líneas. Siempre habrá de cumplirse:

$$Line\ Process\ Average \leq Lines\ Period$$

Buf TRIG Su período viene determinado por dos por *#Lines per Frame* veces el periodo del *Line Trigger*

frame_cnt número de imágenes que han sido procesadas por la tarjeta DSP cuando la aplicación ha finalizado.

Diagrama de Bloques

En el diagrama de la Figura E-7 se ve cómo se realiza las llamadas a las funciones de la librería \LabView\MyDLL creada para inicializar la tarjeta DSP (BittWareOpen), cargarle varios frames y leer el último frame tratado (BittWareFFT) y cerrar la tarjeta (BittWareClose).

E.3.3 Implementación de *parallelFFT* en los DSPs (VisualDSP++)

Comenzamos con el pseudocódigo de los procesadores esclavo.

Slave DSP Main Program (VisualDSP++)

```
FFT_SIZE   = 64              LINE_SIZE = 1024 pixels
FRAME_SIZE = 32 lines        NSTAGES   = 2

DO

   read_line(sbuf, link_port_rx)

   DO
       process_line_piece(lbuf)
       read_line(sbuf, link_port_rx)
       write_line(lbuf, link_port_tx)

   WHILE (line < NSTAGES*FRAME_SIZE)

   process_line_piece(lbuf)
   write_line(lbuf, link_port_tx)

WHILE (!done);
```

Pseudocódigo del procesador maestro.

Master DSP Main Program (VisualDSP++)

```
FFT_SIZE   = 64                LINE_SIZE = 1024 pixels
FRAME_SIZE = 32 lines          LINE_FREQ = 3300 Hz

sdram1 = 0x800000              sdram2 = 0x1000000

DO
        WHILE (!input)
        input = 0
        Transpose_Frame(sdram2,sdram1)
        Process_Frame  (sdram1,sdram2, 1)

        Process_Frame  (sdram2,sdram1, 0)
        Transpose_Frame(sdram1,sdram2)
        output = 1

WHILE (!done);
```

Master DSP Functions (VisualDSP++)

```
Transpose_Frame(int *orig, int *dest)

DO
        For (i < FFT_SIZE) For (j < FFT_SIZE)
                aux[j][i] = orig[i][j]

        For (i < FFT_SIZE) For (j < FFT_SIZE)
                dest[i][j] = aux[i][j]

        offset += LINE_SIZE

WHILE (offset < FRAME_SIZE);

Process_Frame(int *orig, int *dest, int post_p)

read_line(sbuf, orig+offset1, DMA10)        // first line (offset1 = 0)

-----

read_line(sbuf, orig+offset1, DMA10)        // second line (offset1 = 1)
process_line_piece(lbuf)

-----

DO
        read_line(sbuf, orig+offset1, DMA10)
        process_line_piece(lbuf)
        write_line(lbuf, dest+offset2, DMA11, post_p)

WHILE (1 < offset1 < FRAME_SIZE)

-----

process_line_piece(lbuf)                          // last processing
write_line(lbuf, dest+offset2, DMA11, post_p)

-----

write_line(lbuf, dest+offset2, DMA11, post_p)     // last transmission
```

E.3.4 Código Matlab (FFT de 1024 puntos complejos)

La implementación en el procesador DSP fue emprendida solamente una vez que el algoritmo específico hubo sido diseñado y probado en Matlab.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ParallelFFT.m
% Parallel Implementation of Float-Point FFTs
% DSP based FDOCT (Juan GAGO)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
close all;
N = 1024;
M = sqrt(N);

for f0 = 1:(N/2)

n = 0: 1/N : 1-1/N;
x = cos(2*pi*f0*n);

X = fft(x);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Step 1&2: Arrange the input array in a M*M matrix
for row = 1 : M
    A(row, 1:M) = x([1:M] + (row-1)*M);
end;

% Step 3 - 1st FFT stage: 32 FFTs of 64 complex points
for row = 2: 2: 64
    D(row-1, 1:M)    = zeros(1,M);
    D(row, 1:M)      = A(row/2, 1:M);
end ;
B = fft(D,64);

% Step 4: Change the phase of the elements
for m = 1 : M
    for k = 1 : M
C(m,k) = B(m,k) * exp (- 2 * pi * i * (m-1) * (k-1) / N);
    end;
end;

% Step 5&6 - 2nd FFT stage: 32 FFTs of 64 complex points
A = C';
for row = 2: 2: 6
    D(row-1, 1:M)    = zeros(1,M);
    D(row, 1:M)      = A(row/2, 1:M);
end ;
B = fft (D,64);

% Arrange the output M*M matrix in an N-array
for row = 1 : M
    Y([1:M] + (row-1)*M) = B(row, M:-1:1);
end;
Y = circshift(Y,[0 -M]);

end;
```

E.3.5 Programa CFFT_F 64 (MS Visual C++)

En esta sección trataremos la medida del número de ciclos que emplea un procesador DSP en completar una célula de procesado. Además, en la primera etapa del algoritmo será necesaria la multiplicación del resultado por un exponencial vector.

```
float input r[FFT SIZE], output r[FFT SIZE];    // FFT SIZE = 64
float input_i[FFT_SIZE], output_i[FFT_SIZE];

complex_float temp, exponent;

      exponent.re = 0;
      exponent.im = 2 * pi * 45;

      time_temp = count_start();          // Benchmark routine start

      cfftf(input_r, input_i, output_r, output_i, FFT_SIZE);

      for (i=0, j=0; i<FFT_SIZE; i++, j +=2)
      {
            temp.re = output_r[i];
            temp.im = output_i[i];
            exponent.im = i * exponent.im;

            temp = cmltf( temp, cexp(exponent));

            output_r[i] = temp.re;
            output_i[i] = temp.im;
      }
      cycle_count = count_end(time_temp); // Benchmark routine end
```

Para realizar dicha medida se empleará las funciones en ensamblador.

```
  count start:     /* call this to start cycle count */
r1=mode1;
bit clr mode1 IRPTEN;
r0=emuclk;
mode1=r1;
exit;

_count_end:       /* call this to end cycle count */
r2=mode1;
bit clr mode1 IRPTEN;
r0=emuclk;
r0=r0-r4;
r1=14;            /* fudge factor to compensate for overhead */
r0=r0-r1;
mode1=r2;
exit;
```