IMPLEMENTACIÓN DE ALGORITMO DE FORMACIÓN DE IMÁGENES SAR EN PARALELO USANDO PROCESADORES DIGITALES DE SEÑALES

VLADIMIR LINARES DÍAZ

JORGE ANDRÉS TEJEDOR BOTELLO

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FÍSICO MECÁNICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA ELECTRÓNICA Y TELECOMUNICACIONES

BUCARAMANGA

2009

IMPLEMENTACIÓN DE ALGORITMO DE FORMACIÓN DE IMÁGENES SAR EN PARALELO USANDO PROCESADORES DIGITALES DE SEÑALES

VLADIMIR LINARES DÍAZ

JORGE ANDRÉS TEJEDOR BOTELLO

Este trabajo se presenta como requisito para optar al título de ingeniero electrónico.

Directora
M.Sc. ANA BEATRÍZ RAMÍREZ SILVA

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FÍSICO MECÁNICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA ELECTRÓNICA Y TELECOMUNICACIONES

BUCARAMANGA

2009

AGRADECIMIENTOS

Agradecimientos en primer lugar a Dios, quien siempre nos dio fuerza y constancia para proseguir en nuestra investigación.

A nuestros padres, familiares y amigos por el apoyo brindado durante todo este tiempo de vida universitaria.

A la profesora Ana Beatriz por su gran apoyo, colaboración y paciencia como directora y consejera en el desarrollo de este proyecto de grado ya que sin ella esto no hubiera sido posible.

A nuestro compañero Pablo L. Sordo desde España por su generosa ayuda y a todas aquellas personas que de algún modo contribuyeron a la realización de este proyecto y compartieron sus conocimientos para que este trabajo tuviera un buen término.

TABLA DE CONTENIDO

pág.

LISTA DE FIGURAS

pág.

LISTA DE TABLAS

pág.

LISTA DE ANEXOS

pág.

GLOSARIO

1-D: One Dimension

2-D: Two Dimension

AIC23: Audio Codec Device Driver

ALU: Arithmetic Logic Unit

CCS: Code Composer Studio

CPLD: Complex Programmable Logic Device

CPU: Central Processing Unit

DFT: Discrete Fourier Transform

DSK: Digital Starter Kit

DSP: Digital Signal Processor

EDMA: Enhanced Direct Memory Access

EMIF: External Memory Interface

FFT: Fast Fourier Transform

FSG: Frame-Sync Signal Generator

IDFT: Inverse Discrete Fourier Transform

IFFT: Inverse Fast Fourier Transform

McBSP: Multichannel Buffer Serial Port

MFLOPS: Millions of Floating Point Operations Per Second

MIPS: Millions of Instructions Per Second

PC: Personal Computer

RAM: Random Access Memory

ROM: Read Only Memory

SAR: Synthetic Aperture Radar

SDRAM: Synchronous Dynamic Random Access Memory

TI: Texas Instruments

USB: Universal Serial Bus

VLIW: Very Long Instruction Word

RESUMEN

TITULO: IMPLEMENTACIÓN DE ALGORITMO DE FORMACIÓN DE IMÁGENES SAR EN PARALELO USANDO PROCESADORES DIGITALES DE SEÑALES. *

AUTORES: VLADIMIR LINARES DÍAZ y JORGE ANDRÉS TEJEDOR B. **

Palabras claves: radares SAR, transformada rápida de Fourier bidimensional, convolución circular bidimensional, DSP, McBSP, algoritmo en paralelo.

DESCRIPCIÓN

Los radares de apertura sintética (SAR) son dispositivos de buen desempeño que permiten recopilar grandes cantidades de información acerca de la superficie terrestre, aún en difíciles condiciones climáticas. El gran volumen de datos obtenidos por este tipo de radares hace necesaria la utilización de hardware y software adecuados para el procesamiento de estos datos y la obtención de la imagen deseada dependiendo de la aplicación.

En este trabajo se presenta la implementación de un algoritmo en paralelo que es usado para la formación de imágenes de radares SAR junto con los resultados obtenidos de esta implementación. De igual manera, se presentan los fundamentos teóricos en los cuales se basó esta implementación haciéndose una breve explicación de técnicas y conceptos útiles en el diseño del algoritmo como son la transformada rápida de Fourier bidimensional, la transforma rápida inversa de Fourier bidimensional, la convolución circular bidimensional, así como también una descripción del hardware y el software utilizado en el desarrollo de este proyecto.

La implementación del algoritmo se hace sobre dos procesadores digitales de señales (DSPs) de Texas Instruments de la familia C6000 de punto flotante, conectados en paralelo a través del puerto de alta velocidad Multichannel Buffered Serial Port (McBSP) usado para el intercambio de información entre los dispositivos. Esto permite la reducción del tiempo de cómputo total y del tiempo de procesamiento de los datos entregados por el radar en comparación con implementaciones de tipo secuencial, obteniéndose de esta manera una mejor aproximación al procesamiento en tiempo real que se desea alcanzar en radares tipo SAR.

---

* Proyecto de grado.

** Facultad de Ingenierías Físico-Mecánicas, Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. M.Sc. Ana Beatriz Ramírez Silva.

ABSTRACT

TITLE: IMPLEMENTATION OF ALGORITM OF SAR IMAGES FORMATION IN PARALLEL USING DIGITAL SIGNAL PROCESSORS. *

AUTHORS: VLADIMIR LINARES DÍAZ and JORGE ANDRÉS TEJEDOR B. **

Keywords: radar SAR, two-dimensional fast Fourier transform, two-dimensional circular convolution, DSP, McBSP, algoritm in parallel.

DESCRIPTION

Synthetic Aperture Radars (SAR) are good perfomance devices that allow compiling great quantities of data about terrestrial surface, even in hard weather conditions. The volume of the data generated by this type of radars force the adecuade utilization of hardware and software tools for the data processing and the obtaining of the desire image, depending of the application.

This work shows the implementation of one algoritm in parallel which is used for SAR images formation and the results obtained with this implementation. Also, this work shows the theorical fundamentals that are the base of this implementation, explaining brevity useful techniques and concepts like the two-dimensional fast Fourier transform, two-dimensional inverse fast Fourier transform and the two-dimensional circular convolution, as well as a description of hardware and software used in the development of this project.

The implementation of the algoritm is done over two digital signal processors (DSPs) belonging to the family C6000 of floating-point processors of Texas Instruments connected in parallel through the high speed port Multichannel Buffered Serial Port (McBSP), using for the data exchange between the devices. This allows the reduction of the total computing time and the processing time in comparison with implementations of sequential type, achieving a better aproximation for real-time processing desired in radars of SAR type.

***** Project of Grade

****** Faculty of Physical-Mechanical Engineerings, School of Electrical, Electronic and Telecomunications Engineerings. M.Sc. Ana Beatriz Ramírez Silva.

INTRODUCCIÓN

Los radares SAR, son dispositivos que permiten recopilar información de la superficie terrestre enviando a ésta, señales de microondas y recibiendo los ecos resultantes. Debido a las características que poseen las microondas hacen posible que el dispositivo opere exitosamente bajo casi cualquier condición climática y se recojan los datos obtenidos para formar una clara imagen de alta resolución del terreno. Esta versatilidad de este tipo de radares hace que tengan incluso mejor desempeño que sistemas similares y que sean excelentes candidatos a tener en cuenta en numerosas aplicaciones tales como la teledetección y la cartografía. Sin embargo, una de las principales desventajas de este tipo de sistemas es la gran cantidad de datos necesarios para generar una imagen y la respectiva consecuencia consiste, en que para el procesamiento que se debe realizar debe usarse un hardware y software adecuado, que permita producir una imagen correcta que brinde la información relevante para quien la interprete en un tiempo considerado aceptable según la aplicación.

En este trabajo se utilizan dos DSPs de Texas Instruments pertenecientes a la familia de procesadores de punto flotante C6000 como herramienta para el procesamiento de imágenes tipo SAR. Estos procesadores son: un procesador TMS320C6711 y un procesador TMS320C6713.

En el capítulo primero se presentan los fundamentos teóricos sobre radares SAR, transformada de Fourier en 1D y 2D directa e inversa, convolución cíclica en 2D y una descripción del software y el hardware usado en este trabajo para el procesamiento de las imágenes tipo SAR.

En el capítulo segundo se encuentra la descripción de los algoritmos utilizados y una explicación del funcionamiento de los puertos McBSP usados para la comunicación entre los procesadores.

En el tercer capítulo se muestran los resultados obtenidos de la implementación de los algoritmos en paralelo junto con una comparación entre estos resultados y resultados de implementaciones secuenciales.

Finalmente en los capítulos cuarto y quinto se presentan respectivamente las conclusiones derivadas del trabajo realizado y las recomendaciones que puedan servir para futuros proyectos relacionados.

1. CONCEPTOS BÁSICOS

1.1 FUNDAMENTOS TEÓRICOS

1.1.1 Radar SAR. Los radares SAR (*Synthetic Aperture Radar*) o radares de apertura sintética, son sistemas de radares muy versátiles que sirven para generar imágenes de alta resolución de la superficie terrestre aún en difíciles condiciones atmosféricas. Una apertura sintética consiste en un arreglo de sucesivos pulsos de señales dirigidos hacia un blanco que son transmitidos y recibidos por una pequeña antena que se mueve a lo largo de una determinada trayectoria de vuelo. El blanco es observado por el haz de la antena desde distintos puntos a lo largo de toda la trayectoria, y es este hecho el que hace que el dispositivo SAR se convierta en una antena virtual de mayor tamaño lo que sería algo equivalente a prolongar la longitud real de la antena. Este concepto de larga antena virtual es la base de los radares de apertura sintética.

El procesamiento de los datos obtenidos por los radares SAR es un trabajo arduo debido al gran volumen de información correspondiente a cada imagen que se quiera generar. Al final de este complejo procesado se obtienen imágenes con una resolución de algunas decenas de metros por píxel dependiendo del tipo de radar, las cuales proporcionan información valiosa en diversas aplicaciones como agricultura, geología, hidrología, entre otras.

1.1.2 Transformada Rápida de Fourier 1-D y 2-D Radix-2. La Transformada de Fourier Discreta (DFT) convierte una secuencia en el dominio del tiempo en una secuencia equivalente en el dominio de la frecuencia. La Transformada Rápida de Fourier (FFT) es una técnica muy eficiente que realiza el cómputo de la DFT en forma aproximada usando muchas menos operaciones, lo cual reduce considerablemente los requerimientos del hardware para computación.

Se tiene que la DFT de una señal en tiempo discreto $x[nT]$ es:

$$X(k) = \sum_{n=0}^{N-1} x(n) W^{nk} \qquad k = 0,1..., N-1 \qquad (1.1)$$

En donde el periodo de muestreo T está implícito en $x[n]$ y N es el número de puntos de la señal $x[n]$. La constante W se refiere a la constante o factor de giro, la cual representa la fase:

$$W = e^{-\frac{j2\pi}{N}} \qquad (1.2)$$

Y son funciones de la longitud N. La ecuación (1.1) puede ser escrita para $k = 0,1..., N-1$, como:

$$X(k) = x(0) + x(1)W^{k} + x(2)W^{2k} + ... + x(N-1)W^{(N-1)k} \qquad (1.3)$$

De esta forma se puede encontrar una matriz de $N \times N$; $X(k)$ necesita ser calculada para N valores de k dado que la ecuación (1.3) está en términos de exponenciales complejas, para cada valor de k hay $(N-1)$ sumas complejas y N multiplicaciones complejas. Esto resulta en un total de $(N^2 - N)$ sumas complejas y N^2 multiplicaciones complejas. Por lo tanto, los requerimientos de la DFT pueden ser muy extensos, especialmente para valores grandes de N. La FFT reduce los cálculos complejos desde N^2 operaciones de multiplicación hasta

$N\log_2(N)$ [1]. El algoritmo FFT toma ventaja de la periodicidad y simetría de la constante de giro para reducir los requerimientos computacionales de la DFT. Desde la periodicidad de W teniéndose que $W^{k+N} = W^k$ y desde la simetría de W teniéndose que $W^{k+N/2} = -W^k$. La Figura 1 ilustra las propiedades de la constante de giro W para $N = 8$. Si dejamos que $k = 2$, vemos que por ejemplo $W^{10} = W^2$ y que $W^6 = -W^2$.

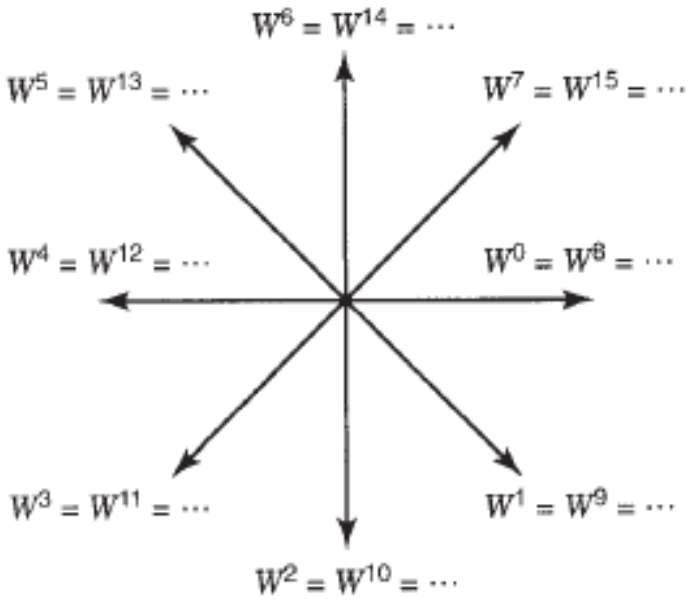


Figura 1. Periodicidad y simetría de la constante de giro W.

Fuente: referencia [1]

Las dos ecuaciones definitivas para la FFT 1D en base 2 (radix-2) son:

$$F(2k) = \sum_{n=0}^{\frac{N}{2}-1}\left[x[n] + x\left[n+\frac{N}{2}\right]\right]W_{\frac{N}{2}}^{kn} \qquad k = 0,1,2,\ldots,N/2-1 \qquad (1.4)$$

$$F(2k+1) = \sum_{n=0}^{\frac{N}{2}-1}\left[x[n] - x\left[n+\frac{N}{2}\right]W_N^n\right]W_{\frac{N}{2}}^{nk} \qquad k = 0,1,2,\ldots,N/2-1 \qquad (1.5)$$

Para la FFT radix-2, la transformada descompone una DFT de N puntos en dos DFTs de $(N/2)$ puntos. Cada DFT de $(N/2)$ puntos es cada vez descompuesta en dos DFTs de $(N/4)$ puntos y así sucesivamente. La más pequeña transformada es determinada por la dimensión de la FFT. Para una FFT radix-2, N debe ser una potencia o base de 2, y la más pequeña transformada o la última descomposición es la DFT de dos puntos. Hay que anotar que al aplicar las ecuaciones (1.4) y (1.5), el resultado de la FFT no se encuentra organizado. Este hecho se muestra en la Figura 2 para la FFT de una señal de 8 puntos:

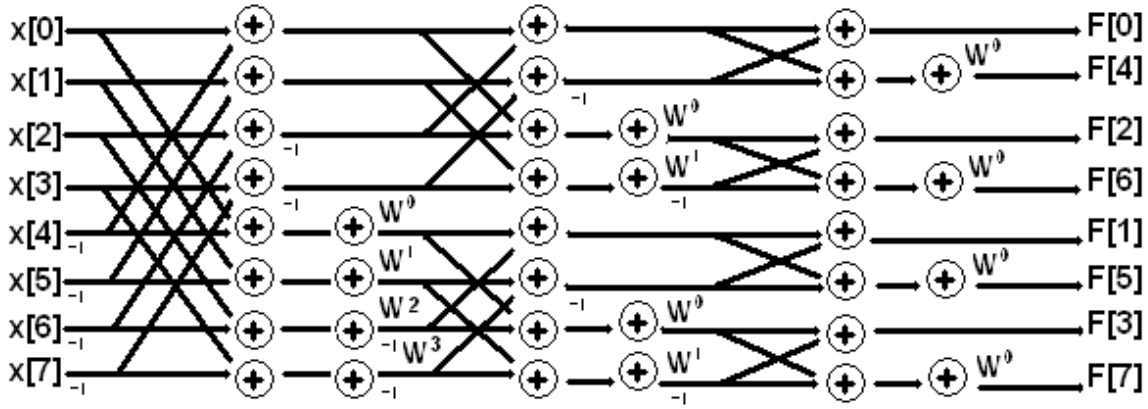


Figura 2. Representación operacional de una FFT de ocho puntos.

Fuente: autores del proyecto

Debido a esto es necesario implementar un proceso de reacomodación de los datos de forma correcta, este proceso se conoce como direccionamiento con inversión de bit el cual consiste en invertir la dirección de memoria de la FFT. Para el ejemplo de la FFT de una señal de 8 puntos, el orden de los resultados y su reacomodación correcta usando la inversión de bits (que hace referencia al índice) se muestran a continuación en la Tabla 1:

Tabla 1. Inversión de bits de memoria.

FFT	Bits de memoria	Inversión de bits	FFT ordenada
F(0)	000	000	F(0)
F(4)	001	100	F(1)
F(2)	010	010	F(2)
F(6)	011	110	F(3)
F(1)	100	001	F(4)
F(5)	101	101	F(5)
F(3)	110	011	F(6)
F(7)	111	111	F(7)

El proceso para realizar la transformada rápida de Fourier en dos dimensiones se hace en primer lugar obteniendo la FFT 1-D de las filas de la matriz y después a esta matriz resultante se le aplica nuevamente la FFT 1-D esta vez sobre las columnas obteniéndose la FFT 2-D.

1.1.3 Transformada Inversa Rápida de Fourier 1-D y 2-D Radix-2. La Transformada Discreta Inversa de Fourier (IDFT) convierte una secuencia en el dominio de la frecuencia $X(k)$ a una secuencia equivalente $x(n)$ en el dominio del tiempo. Esta se define para dos dimensiones como:

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)W^{-nk} \qquad n = 0,1,...,N-1 \quad (1.6)$$

La operación para la transformada rápida inversa de Fourier (IFFT) es muy similar a la de la FFT. El algoritmo descrito anteriormente para la FFT puede ser usado de manera similar para encontrar la IFFT con dos cambios básicos:

- Se adiciona el factor de escala $1/N$

- Se reemplaza el factor W^{nk} por su conjugado complejo W^{-nk}

De la misma manera como se aplica la FFT 1-D inicialmente a las filas de la matriz y luego a las columnas de la matriz resultante, para obtener la FFT directa en 2-D; para obtener la IFFT 2-D de una matriz, se aplica la IFFT 1-D a las filas de la misma y a la matriz resultante se le aplica la IFFT 1-D para las columnas.

1.1.4 Convolución Cíclica 2-D. Cuando se tienen señales discretas en el tiempo con períodos idénticos, la operación de convolución entre ellas se denomina convolución cíclica o circular representada matemáticamente como:

$$y[n] = x[n] \otimes h[n] \quad (1.7)$$

En donde $y[n]$ es la salida del sistema $x[n]$ es la señal de entrada y $h[n]$ la respuesta al impulso del sistema. Si las señales son de longitud N la operación de convolución cíclica también será de longitud N .

La convolución cíclica de dos matrices por el método de las transformadas rápidas de Fourier directas e inversas, se obtiene como resultado de computar la IFFT 2-D de la matriz resultante de la multiplicación de elemento a elemento de las FFT 2-D de las matrices que son entrada a la operación de convolución .En este caso, las matrices de interés que serán usadas en el proceso de convolución bidimensional corresponden a:

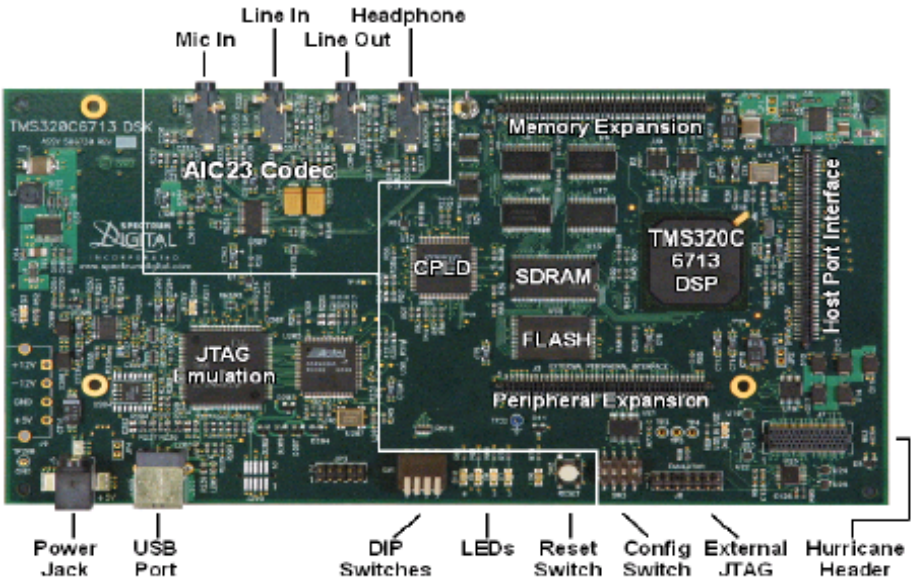
- La matriz de los datos crudos obtenidos por el radar
- La matriz correspondiente a la respuesta al impulso del radar.

Esta operación de convolución en dos dimensiones nos permite obtener una imagen del radar que más adelante requiere un pos-procesamiento para mejorar la calidad de la misma y ser utilizada en diferentes aplicaciones.

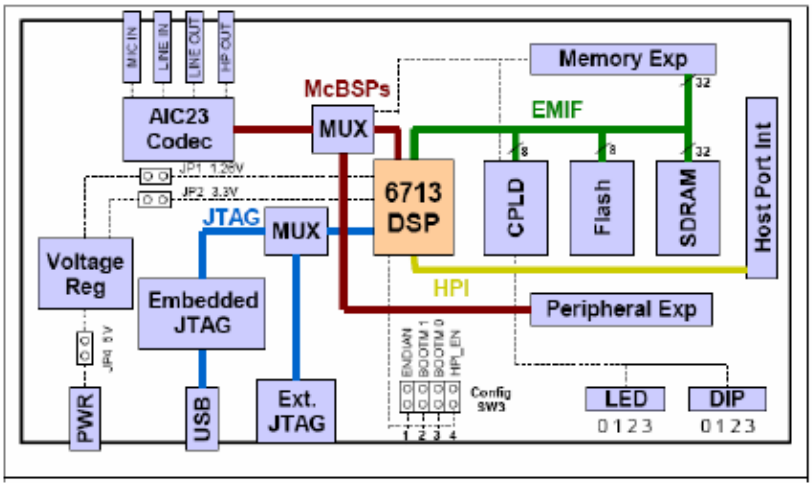
1.2 DESCRIPCIÓN DEL HARDWARE

Para la realización de este trabajo se requieren dos procesadores digitales de señales, los cuales serán interconectados en paralelo para realizar las operaciones de FFT 2-D y convolución cíclica 2-D. Los procesadores escogidos corresponden a la familia de procesadores digitales de señales de Texas Instruments TMS320C6000.

1.2.1 TMS320C6713 DSK. La tarjeta de desarrollo TMS320C6713 DSK es una tarjeta de bajo costo y de alto rendimiento que incluye un procesador de punto flotante Texas Instruments TMS320C6713 DSP, por lo que es muy usada para el procesamiento de señales en tiempo real.



a)



b)

Figura 3. TMS320C6713 DSK: a) tarjeta. b) diagrama de bloques.

Fuente: referencia [2]

Las características más relevantes de la tarjeta son:

- Un procesador TMS320C6713 DSP con frecuencia de operación de 225Mhz.
- Un moderno códec de audio stereo AIC23.
- Memoria SDRAM de 16 MBytes.
- Memoria Flash de 512 KBytes con 256 KBytes reservados para la configuración del sistema.
- 4 LEDS y 4 interruptores DIP accesibles por el usuario.
- Conector USB.
- Fuente de voltaje sencilla de +5V.
- CPLD con 4 registros para control interno y encargado de manejar la lógica que controla los periféricos del DSK.
- Opciones de arranque configurables**.**

La TMS320C6713 (C6713) es capaz de realizar más de 1350 millones de operaciones de punto flotante por segundo (MFLOPS) y más de 1800 millones de

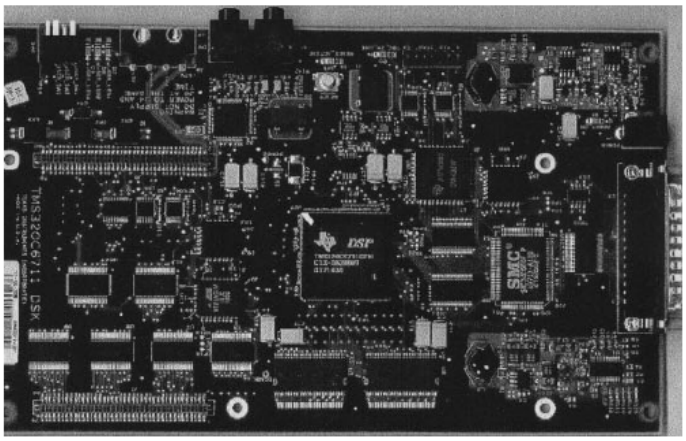
instrucciones por segundo (MIPS). Esto la hace muy atractiva para implementar operaciones tales como la FFT con grandes matrices de datos como es nuestro caso.

Además de esto, la C6713 posee una gran variedad de dispositivos periféricos tales como 2 Multichannel Audio Serial Ports (McASPs), 2 Multichannel Buffered Serial Ports (McBSPs), dos buses I2C, un módulo dedicado GPIO, una interfase HPI y una interfase EMIF.

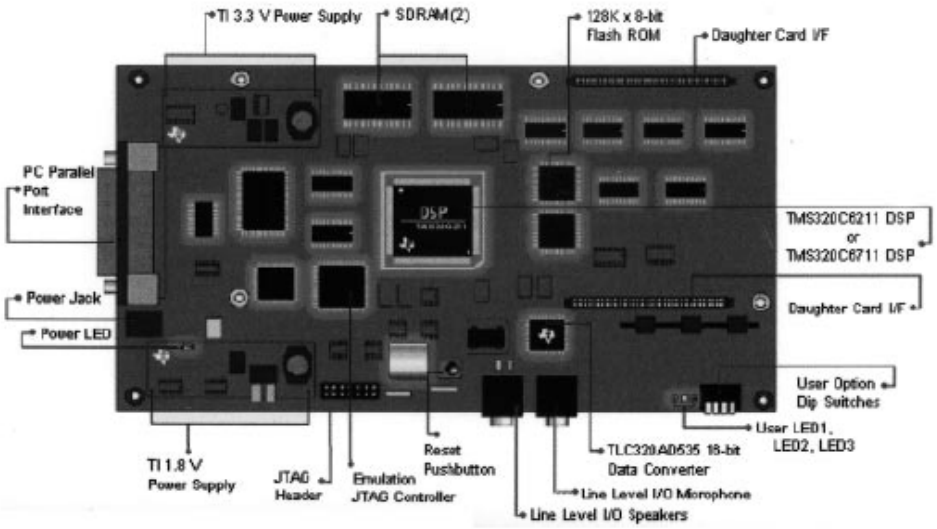
1.2.2 TMS320C6711 DSK. El paquete TMS320C6711 DSK es una herramienta muy útil, relativamente económica, con el soporte necesario tanto en software como en hardware para llevar a cabo el procesamiento de señales en tiempo real. Es un sistema completo DSP. La tarjeta DSK, con una dimensión aproximada de 5*8 pulgadas, incluye un procesador TMS320C6711 de señales digitales de punto flotante y un códec de audio de 16 bits AD535 para la entrada y salida y se conecta al PC a través de un cable paralelo. El códec de audio AD535, usa una tecnología sigma-delta que provee conversión análogo a digital (ADC) y una conversión digital a análogo (DAC). Un reloj de 4 MHz en la DSK, conecta este códec para proveer un rango de muestreo fijo de 8 KHz. Dos conexiones externas para daughter cards también son proveídas en la tarjeta DSK con el fin de ampliar la memoria RAM y conectar otros dispositivos a la tarjeta.

La tarjeta DSK incluye 16 MB de RAM dinámica síncrona (SDRAM) y 128kB de ROM flash. Dos conectores dentro de la tarjeta proporcionan la entrada y salida que están enmarcadas como IN (J7) y OUT (J6), respectivamente. Tres de los cuatro interruptores DIP en la DSK pueden ser leídos desde un programa. La DSK TMS320C6711 consta de un reloj interno de 150 MHz. También consta de reguladores de tensión que proveen 1,8 V para el núcleo de la TMS320C6711 (C6711) y 3,3 V para sus memorias y periféricos.

En la Figura 4 se observa la C6711 junto con un diagrama de bloques funcional de la misma:



a)



b)

Figura 4. TMS320C6711 DSK: *a*) tarjeta *b*) diagrama.

Fuente: referencia [10]

La C6711 está basada en la arquitectura de very-long-instruction-word (VLIW), la cual es muy conveniente para algoritmos intensivos numéricamente. La memoria del programa interno está estructurada de manera que un total de ocho

instrucciones pueden ser llevadas cada ciclo. Por ejemplo, con una velocidad de reloj de 150 MHz, la C6711 es capaz de llevar ocho instrucciones de 32 bits cada 1/(150 MHz) o 6,66 ns. Las características de la C6711 incluyen 72 kB de memoria interna, ocho unidades funcionales o de ejecución compuestas de seis ALUs y dos unidades multiplicadoras, un bus de dirección de 32 bits para dirigir 4 GB (gigabytes), y dos grupos de registros de 32 bits con propósitos generales. La C6711 pertenece a la familia de los procesadores C6000 de punto flotante que consta tanto de procesamiento de punto fijo como de punto flotante.

1.3 DESCRIPCIÓN DEL SOFTWARE

1.3.1 Code Composer Studio IDE (CCS). El software Code Composer Studio (CCS) es un ambiente de desarrollo integrado utilizado para crear aplicaciones en C o en lenguaje ensamblador y sirve como plataforma de programación de los DSPs de Texas Instruments Inc. Algunas de las principales características de este ambiente son la posibilidad de revisar variables y registros desde el DSK y su utilidad para intercambiar datos entre la tarjeta y otros lenguajes de programación como MATLAB o Microsoft Excel. La versión del software utilizado en este trabajo es la v3.1.

Los algoritmos que quieran ser implementados en el DSK pueden ser creados, compilados, depurados y cargados al DSK por medio de esta herramienta software, para luego ser ejecutados directamente por la tarjeta.

El entorno de trabajo del Code Composer Studio se encuentra en la Figura 5**:**

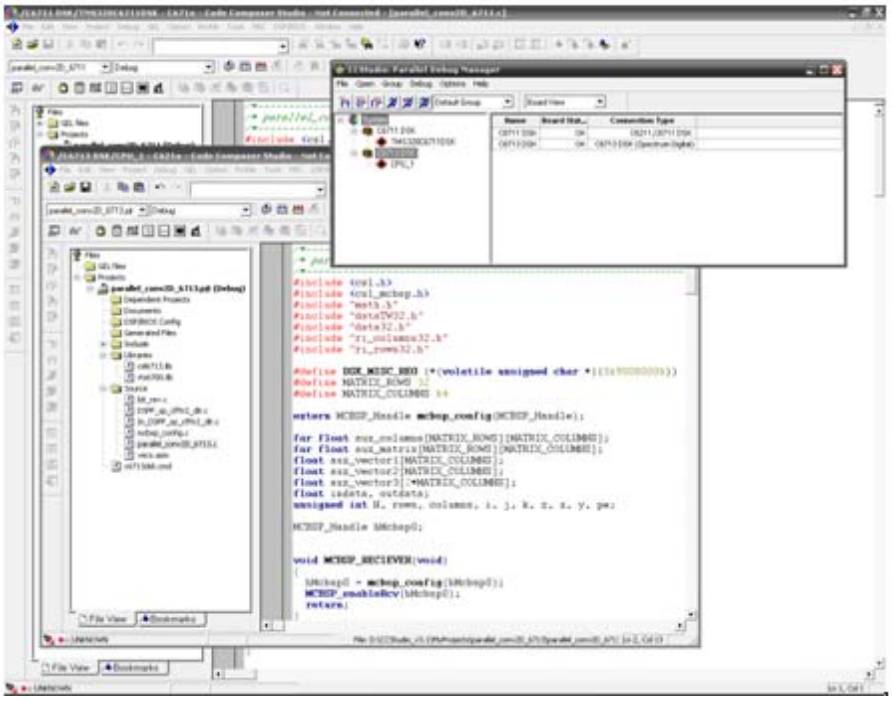


Figura 5. Entorno de trabajo Code Composer Studio V3.1.

Fuente: autores del proyecto

En la Figura 5 también se puede observar que cuando son conectadas dos o más tarjetas al PC, el CCS permite manejarlas conjuntamente usando el Parallel Debug Manager (PDM) el cual, basado en la creación de los denominados grupos, permite tener funciones como compilar, cargar, ejecutar y detener programas sobre múltiples procesadores al mismo tiempo además de muchas otras funciones útiles para trabajar con procesamiento en paralelo.

Los algoritmos implementados en este trabajo y que se encuentran en los anexos deben estar contenidos en proyectos de extensión *.pjt* que se crean con el CCS los cuales a su vez deben contener los archivos que contengan los respectivos algoritmos de extensión *.c*, además de las librerías de soporte necesarias *.lib*, los

archivos en lenguaje ensamblador *.asm* y el archivo con la descripción de la distribución de memoria en el DSP *.cmd.*

En la Figura 6 se observa la totalidad del hardware que fue usado en este trabajo incluyendo un PC que cuenta con el software CCS y los DSPs TMS320C6713 y TMS320C6711 utilizados junto con la conexión entre tarjetas a través del puerto McBSP que se explicará en el siguiente capítulo.

Figura 6. Hardware total implementado. 1) Conexión USB al PC y Fuente de alimentación C6713. 2) Conexión a través del McBSP para la transmisión y recepción. 3) Conexión puerto paralelo al PC y Fuente de alimentación C6711. 4) PC con CCS.

Fuente: autores del proyecto

2. IMPLEMENTACIÓN DE LOS ALGORITMOS: HERRAMIENTAS UTILIZADAS Y DIAGRAMAS DE FUNCIONAMIENTO

2.1 MULTICHANNEL BUFFERED SERIAL PORT (McBSP)

Antes de cualquier discusión sobre los algoritmos implementados en este trabajo, es de gran importancia hablar acerca del Multichannel Buffered Serial Port (McBSP) el cual es el pilar fundamental para la computación de los algoritmos aquí diseñados, ya que permite la transferencia de datos entre los dos procesadores de señales, haciendo posible el paralelismo necesario. Para utilizar este puerto debe incluirse las librerías *csl.h* y *csl_mcbsp.h* en el CCS.

2.1.1 Generalidades del McBSP. El McBSP consiste de una línea de datos y una línea de control que se conectan a dispositivos externos. Pines separados para la transmisión y la recepción permiten comunicar los datos a estos dispositivos externos. Cuatro pines llevan información de control (señales de reloj y *frame* de sincronización). El dispositivo se comunica con el McBSP usando registros de datos y control de 32 bits accesibles a través del bus periférico interno. Este puerto serial puede tener velocidades de transmisión de hasta 35.71 Mbps para dispositivos como la C6711 y de hasta 125 Mbps para procesadores de la serie C64x [7].

El diagrama de bloques de un puerto McBSP a nivel de registros se muestra a continuación en la Figura 7. Las líneas de control y de datos a dispositivos externos se conectan a través de los siete pines mostrados en la Tabla 2.

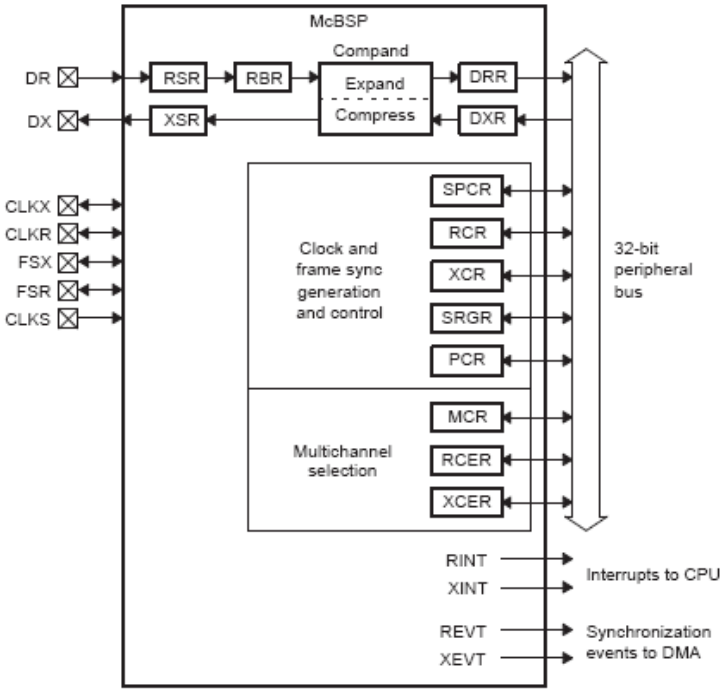


Figura 7. Diagrama de bloques del McBSP.

Fuente: referencia [5]

Tabla 2. Pines del McBSP.

Pin	I/O/Z	Descripción
CLKR	I/O/Z	Reloj Receptor
CLKX	I/O/Z	Reloj Transmisor
CLKS	I	Reloj Externo
DR	I	Información Serial Recibida
DX	O/Z	Información Serial Transmitida
FSR	I/O/Z	Frame de Sincronización de Recepción
FSX	I/O/Z	Frame de Sincronización de Transmisión

Nota: I es un pin de entrada, **O** es un pin de salida y **Z** es un pin de alta impedancia.

Los datos se intercambian entre los dispositivos externos a través del pin de datos transmitidos DX para el transmisor y a través del pin de datos recibidos DR para el receptor. La información de control como la señal de reloj y la señal de *frame* de sincronización se comunican a través de los pines CLKS, CLKX, CLKR, FSX y FSR.

Tanto la CPU como un controlador DMA/EDMA permiten leer los datos desde el registro de recepción de datos (DRR) y permiten escribir los datos a transmitir en el registro de transmisión de datos (DXR). Los datos que son escritos en DXR son desplazados hacia DX a través del registro de desplazamiento de transmisión (XSR). De igual manera, los datos recibidos en el pin DR son copiados a DRR, los cuales pueden ser leídos por la CPU o el controlador DMA/EDMA. Esto permite tener simultáneamente movimientos de datos internamente y comunicación de datos hacia el exterior.

En el conector periférico J3 de ambas tarjetas se encuentran los pines del McBSP necesarios para intercomunicar los dos DSPs. La conexión de los pines del McBSP para los algoritmos de la FFT 2-D y la convolución cíclica 2-D se muestra en la Figura 8:

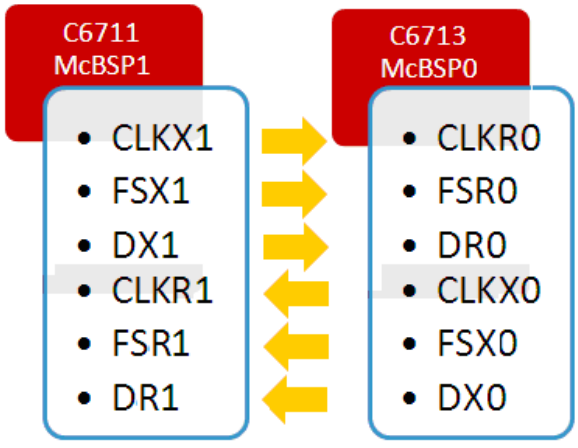


Figura 8. Conexión de los pines del McBSP para el transmisor y el receptor.

Fuente: Autores del proyecto

Es de gran importancia conocer y anotar en este punto que para la C6713 los dos puertos, McBSP0 y McBSP1, vienen por defecto vía software, sirviendo al códec de audio AIC23 así que es necesario fijar en 1 el valor del bit preciso en el registro MISC del CPLD para poder usarlo en el conector J3. De igual forma la C6711 tiene el puerto McBSP0 sirviendo por defecto al códec de audio AD535 pero el puerto McBSP1 está habilitado para usarse en el conector J3 sin inconvenientes.

La conexión entre tarjetas se muestra en la Figura 9:

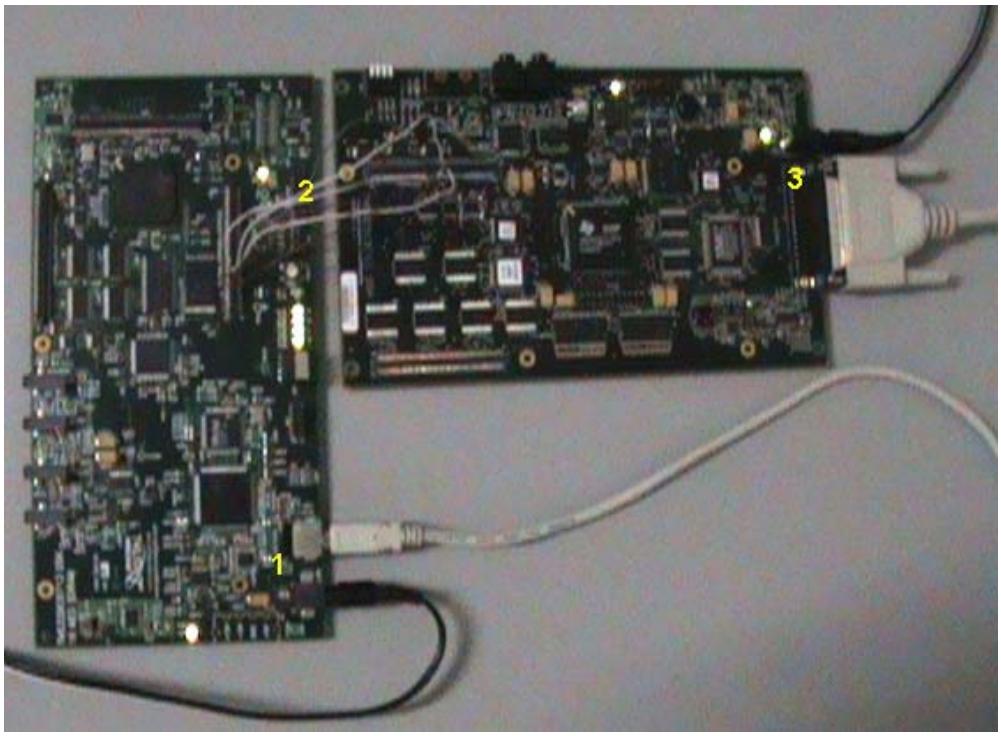


Figura 9. Conexión entre tarjetas a través del McBSP. 1) Conexión USB y Fuente de alimentación C6713. 2) Conexión a través del McBSP para transmisión y recepción. 3) Conexión puerto paralelo y Fuente de alimentación C6711.

Fuente: autores del proyecto

2.1.2 Funcionamiento y configuración básica del McBSP. El problema más importante a resolver es la siguiente pregunta ¿cómo inicializar el McBSP para una correcta sincronización e intercambio de información? Para esto hay que conocer cuáles son los componentes de control principales del McBSP, ya no a nivel de registros, sino usando las instrucciones de programación del CCS que en nuestro caso son C++. Estos bloques del McBSP se observan en la Figura 10:

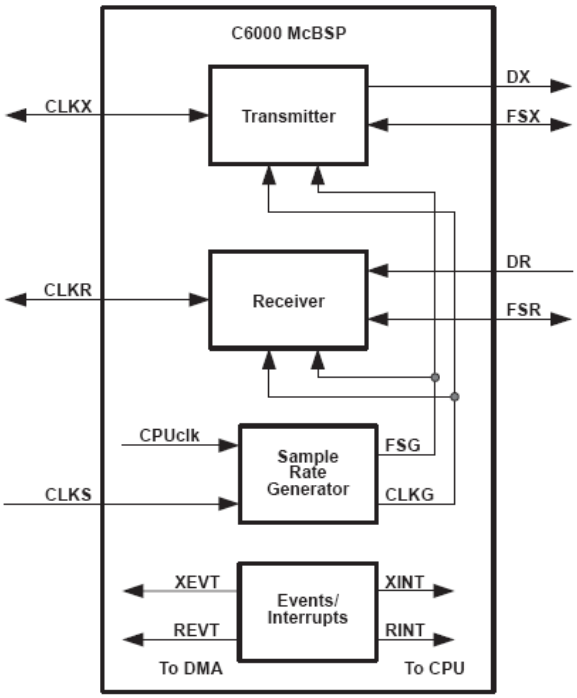


Figura 10. Diagrama de bloques funcional McBSP.

Fuente: referencia [11]

Aquí se observan dos componentes importantes aparte de los bloques de transmisión y de recepción: el Sample Rate Generator (SRG) y la generación de las interrupciones a la CPU.

El SRG como implica su nombre, es un módulo que genera las dos señales internas de control tales como los ciclos de reloj y *frame* de sincronización (CLKG

y FSG), necesarias para la transferencia desde y hacia el McBSP. El circuito de generación de reloj permite al usuario escoger entre el reloj de la CPU o una fuente externa vía el pin CLKS para generar a CLKG que es la que finalmente va a los pines CLK(R/X). El SRG no es usado cuando CLKX, FSX, CLKR y FSR son manejados por una fuente externa. Sin embargo, el bit GRST en SPCR no requiere ser habilitado (GRST=1) para esta configuración.

Las tres etapas del SRG son:

- Clock-divide-down (CLKGDV): número de entradas de reloj por bit de información de reloj.
- Periodo del *frame* (FPER): periodo del *frame* en bits de información de reloj.
- Ancho del *frame* (FWID): el ancho de un pulso de *frame* activo en bits de información de reloj.

Además, una detección del pulso de *frame* y el módulo de sincronización de reloj, permite la sincronización del clock-divide-down con un pulso de *frame* de entrada.

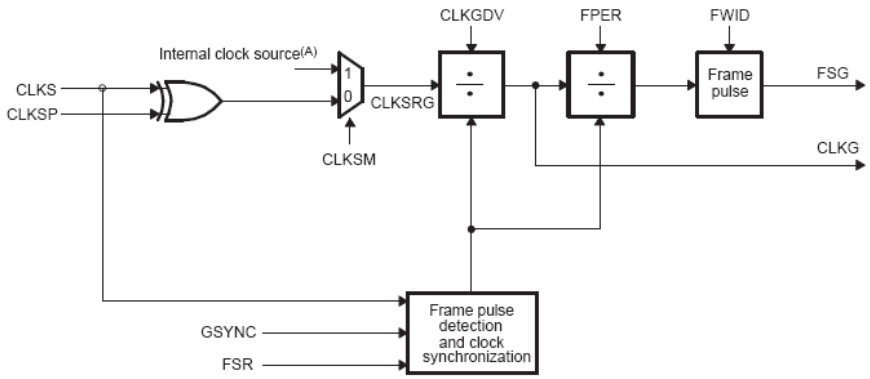


Figura 11. Diagrama de bloques del SRG.

Fuente: referencia [5]

La señal de sincronización es programable de manera independiente para el receptor y transmisor. Cuando se pone en 1 el bit FRST en el registro SPCR, se

activa el generador lógico de *frame* para generar señales de sincronización resultando que FSGM=1 en SPCR. El transmisor puede disparar su propia señal de *frame* de sincronización que es generada por una copia DXR a XSR. Esto produce un *frame* de sincronía en cada copia de DXR a XSR. La información retardada puede ser programada como se requiera.

Además de esto, las propiedades de la señal de *frame* de sincronización, tales como el periodo del *frame* y el ancho del *frame*, son también programables. FS(R/X), CLK(R/X) son pines bidireccionales, y por lo tanto pueden ser entradas o salidas independientemente unas de otras.

Para generar las interrupciones al procesador existen 3 formas básicas: generación de eventos usados por el controlador DMA/EDMA, interrupción vía CPU o usando *polling*. Este último es finalmente el que se utilizó en los algoritmos diseñados en este trabajo, dado que, como se explica a continuación, solo se requiere revisar el estado de los registros de escritura y lectura para que la CPU inicie el proceso de intercambio de datos. Los bits RRDY y XRDY en el registro SPCR de la configuración del McBSP indican el estado de *ready* del receptor y transmisor respectivamente. Cuando se lee DRR y se escribe DXR se modifican directamente RRDY y XRDY, respectivamente.

RRDY=1 indica que el contenido de RBR ha sido copiado a DRR y que la información puede ahora ser leída por la CPU. Una vez que la información ha sido leída por esta, RRDY es borrado a cero (0). También, en el *reset* del dispositivo o el *reset* del receptor del puerto serial (RRST =0), el bit RRDY es borrado a cero (0) para indicar que ninguna información ha sido recibida y cargada dentro de DRR.

XRDY=1 indica que el contenido de DXR ha sido copiado a XSR y que DXR está listo para ser cargado con una nueva palabra de información. Cuando el transmisor pasa del estado de *reset* a *no-reset* (XRST cambia de 0 a1), XRDY

también cambia de cero (0) a uno (1) indicando que DXR está lista para nueva información. Una vez la nueva información es cargada por la CPU el bit XRDY es borrado a cero (0). Sin embargo, una vez esta información es copiada de DXR a XSR, el bit XRDY cambia nuevamente de cero (0) a uno (1). El procesador, puede escribir a DXR aunque XSR no haya sido movido dentro de DX.

Cuando el método de *polling* es usado para servir al transmisor, la CPU escribe o lee directamente a los registros DXR o DRR. También la CPU espera por un bit de reloj del McBSP (CLKX) antes de volver a llevar a cabo el *polling* para escribir el nuevo elemento en DXR. Esto es porque las transiciones de XRDY ocurren a partir de un bit de reloj generado y no del reloj de la CPU. El reloj de la CPU es mucho más rápido y puede causar un falso estado de XRDY, llevando a errores de información debido a sobre-escritura de información.

La Figura 12 muestra la operación típica de las señales de reloj y *frame* del McBSP. Los relojes seriales CLKR y CLKX manejan la recepción y transmisión de los datos. Similarmente las señales de *frame* de sincronización FSR y FSX para el receptor y transmisor respectivamente, definen el comienzo de un elemento y/o el traspaso del *frame*. El McBSP permite la configuración de los siguientes parámetros para la sincronización:

- Polaridades de FSR, FSX, CLKX y CLKR.
- Elección de *frame* con simple o doble fase.
- Para cada fase, el número de elementos por *frame.*
- Para cada fase, el número de bits por elemento.
- Si la sincronización del *frame* se reinicia, la información serial es ignorada.
- El retardo de la información desde el *frame* de sincronización al primer bit de información, el cual puede ser 0, 1 o 2 bits de retardo.
- Justificación derecha o izquierda así como la extensión de signo o rellenado de ceros para la información recibida.

- La configuración es independiente para el receptor y el transmisor.

.

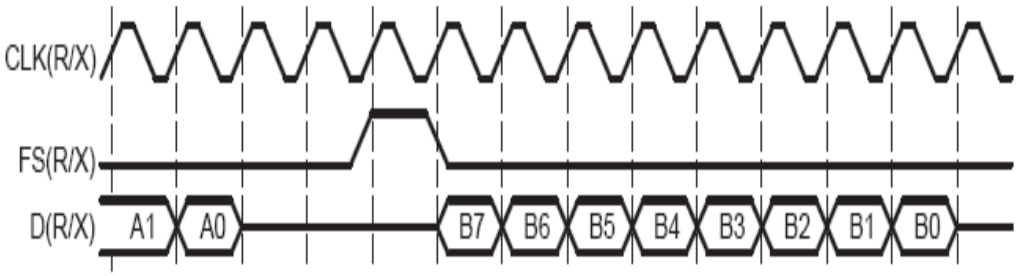


Figura 12. Operación del reloj y la señal de sincronización.

Fuente: referencia [5]

Los pulsos de operación de *frame* para el receptor y transmisor (FSR/X) y relojes (CLKR/X), pueden o ser generados internamente por el SRG, o ser manejados por una fuente externa. La fuente de *frame* de sincronización y reloj es seleccionada programando los bits, FS(R/X)M y CLK(R/X)M respectivamente, en el registro PCR. FSR es también afectada por el bit de GSYNC en el registro SRGR.

Cuando FSR y FSX son entradas (FSXM = FSRM = 0) estas señales son derivadas de una fuente externa que puede provenir de los pines FS(R/X) en el conector J3 de las tarjetas. Cuando FSR o FSX son salidas, éstas son manejadas internamente por el SRG. De igual manera, FSRP, FSXP, CLKRP, y CLKXP sirven para configurar las polaridades de FSR, FSX, CLKR, y CLKX.

En el lado del transmisor, el bit de polaridad de reloj de transmisión, CLKXP, coloca el límite usado para cambiar y registrar la información transmitida. En la configuración diseñada para este trabajo, el receptor usa el flanco opuesto que el del transmisor para la polaridad de reloj de recepción CLKRP (transmisor en flanco de subida y receptor en flanco de bajada en el registro PCR) para asegurar una configuración válida y una correcta operación.

La Tabla 3 muestra cómo se podrían seleccionar varias fuentes para proporcionar la señal de *frame* de sincronización del receptor. Se puede notar que en el modo de realimentación digital (DLB=1), la señal de *frame* de sincronización del transmisor es usada como la señal de sincronización del receptor y que DR está internamente conectado a DX vía software.

Tabla 3. Selección del *frame* de Sincronización del Receptor.

Bit DLB en SPCR	**Bit FSRM en PCR**	**Bit GSYNC en SRGR**	**Fuente del *frame* de Sincronización del Receptor**	**Función del Pin FSR**
0	0	X	Una Señal de Sincronía de *frame* externa maneja el pin de entrada FSR, la cual es luego invertida a medida que es determinada por FSRP antes de ser usada como FSR interna.	Entrada.
0	1	0	La Señal de *frame* de Sincronización (FSG) del SRG maneja el FSR interno, FRST=1.	Salida. FSG es invertida a medida que es determinada por FSRP antes de ser movido dentro del pin FSR.
0	1	1	La Señal de *frame* de Sincronización (FSG) del SRG maneja el FSR interno, FRST=1.	Entrada. La entrada externa de frame de Sincronización dentro de FSR es usada para

				sincronizar CLKG y generar FSG.
1	0	0	FSX interna maneja a FSR interna.	Alta Impedancia
1	x	1	FSX interna maneja a FSR interna.	Entrada. FSR externo no es usado para el *frame* de sincronización pero es usado para sincronizar CLKG y generar FSG dado que GSYNC=1.
1	1	0	FSX interna maneja a FSR interna	Salida. El *frame* de sincronización del receptor (el mismo que el transmisor) es invertido a medida que es determinado por FSRP antes de ser llevado fuera.

De igual manera, la Tabla 4 muestra cómo se puede seleccionar la fuente de la señal de *frame* de sincronización para el transmisor. Consta de tres opciones:

- Entrada externa de *frame* de sincronización
- La señal de *frame* de sincronización del SRG, FSG
- Una señal que indica cuando una copia de DXR a XSR ha sido hecha

Tabla 4. Selección de *frame* de Sincronización del Transmisor.

Bit FSXM en PCR	Bit FSGM en SRGR	Fuente del *frame* de Sincronización del Transmisor	Función del Pin FSX
0	X	Entrada externa del *frame* de sincronización dentro del pin FSX. Esta es invertida por FSXP antes de ser usada como FSX interno.	Entrada.
1	1	La señal de *frame* de sincronización (FSG) del SRG maneja el FSR interno, FRST=1.	Salida. FSG es invertida por FSXP antes de ser movido dentro de FSX.
1	0	Una copia de DXR-XSR activa la señal de *frame* de sincronización del transmisor.	Salida. La señal de reloj de 1-bit de ancho es invertida a medida que es determinada por FSXP antes de ser movida dentro de FSX.

El *frame* de sincronización puede llegar a tener una o dos fases. El número de fases puede ser seleccionado por el bit de fase, (R/X)PHASE, en los registros RCR y XCR. El número de elementos por *frame* y bits por elemento puede ser independientemente seleccionado para cada fase vía (R/X)FRLEN1/2 y (R/X)WDLEN1/2, respectivamente. El máximo número de elementos por *frame* es de 128 para una sola fase, y de 256 elementos en un *frame* de fase dual. El número de bits por elemento puede ser 8, 12, 16, 20, 24 o 32.

En la Figura 13 un elemento de información de 32 bits es transferido hacia y desde el McBSP por la CPU o el controlador DMA/EDMA. Así, una lectura de DRR y una escritura de DXR son necesarias para cada *frame*. Como resultado, el número de transferencias es menor (un cuarto menor) que si se transmitiera la equivalencia en bits de cuatro elementos de 8 bits, también con fase simple. Esta manipulación reduce el porcentaje de tiempo del bus requerido para movimiento de información en el puerto serial.

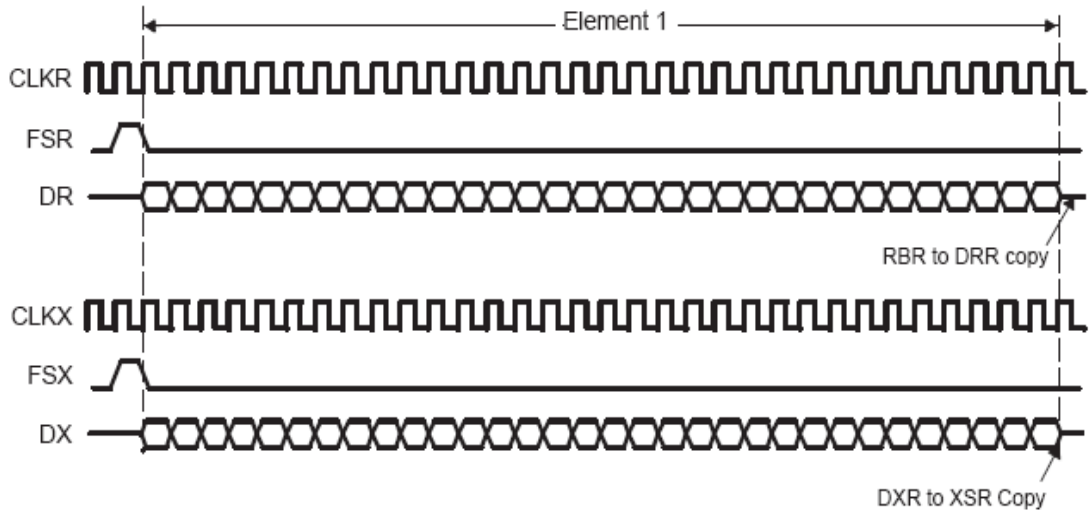


Figura 13. *Frame* de Fase Simple de un elemento de 32 bits.

Fuente: referencia [5]

Estas señales se pueden observar físicamente con un analizador lógico digital, como el mostrado en la Figura 14, el cual fue utilizado en este proyecto como una herramienta recomendada y muy útil para comprobar la operación y el correcto funcionamiento del McBSP.

Figura 14. Analizador Lógico Digital.

Fuente: autores del proyecto.

La configuración diseñada en este proyecto para los puertos McBSP tanto para la transmisión como para la recepción es casi idéntica salvo la aclaración hecha acerca de la utilización del puerto McBSP0 para la C6713 y el puerto McBSP1 para la C6711. La configuración de los puertos se encuentra en el Anexo C. Esta configuración para los registros del McBSP se explica a continuación:

MCBSP_Config mcbspCfg = {
MCBSP_SPCR_RMK(
MCBSP_SPCR_FREE_NO, // Durante la detención de la emulación el bit SOFT determina la operación del McBSP.
MCBSP_SPCR_SOFT_NO, // En conjunción con FREE = NO, el puerto serie detiene su reloj durante la detención de la emulación abortando cualquier transmisión.

MCBSP_SPCR_FRST_YES, // lógica de *frame* de sincronización está en *reset.*
La señal de *frame* de sincronización FSG no es generada por SRG.
MCBSP_SPCR_GRST_YES, // SRG está en *rese*t.
MCBSP_SPCR_XINTM_XRDY, //XINT es manejada por XRDY.
MCBSP_SPCR_XSYNCERR_NO, // No hay error de sincronización detectado (transmisor).
MCBSP_SPCR_XRST_NO, // Puerto serie está habilitado.
MCBSP_SPCR_DLB_OFF, // Modo DLB está deshabilitado.
MCBSP_SPCR_RJUST_RSE, // Justificación de signo-extendido
MCBSP_SPCR_CLKSTP_DISABLE, // Reloj normal para modo no-SPI.
MCBSP_SPCR_DXENA_OFF, // DX habilitador está en *off.*
MCBSP_SPCR_RINTM_RRDY, RINT es manejada por RRDY.
MCBSP_SPCR_RSYNCERR_NO, No hay error de sincronización detectado (receptor).
MCBSP_SPCR_RRST_YES // El puerto serie del receptor está desactivado y en estado de *reset.*
),
MCBSP_RCR_RMK(
MCBSP_RCR_RPHASE_SINGLE, // Fase de *frame* sencilla.
MCBSP_RCR_RFRLEN2_DEFAULT, //Valor por defecto.
MCBSP_RCR_RWDLEN2_DEFAULT, //Valor por defecto.
MCBSP_RCR_RCOMPAND_MSB, // Sin *companding.*
MCBSP_RCR_RFIG_NO, // Pulsos de frame de sincronización del receptor después del primer pulso reinicia la transferencia.
MCBSP_RCR_RDATDLY_1BIT, // Retardo de 1 bit.
MCBSP_RCR_RFRLEN1_OF(0), // 1 elemento por *frame.*
MCBSP_RCR_RWDLEN1_32BIT, // Elementos recibidos de 32 bits.
MCBSP_RCR_RWDREVRS_DISABLE //Reversión de 32-bits del receptor está desactivada.
),

MCBSP_XCR_RMK(

MCBSP_XCR_XPHASE_SINGLE, // Fase de *frame* sencilla.

MCBSP_XCR_XFRLEN2_DEFAULT, //Valor por defecto.

MCBSP_XCR_XWDLEN2_DEFAULT, // Valor por defecto.

MCBSP_XCR_XCOMPAND_MSB, // Sin *companding*.

MCBSP_XCR_XFIG_NO, // Pulsos de *frame* de sincronización del transmisor después del primer pulso reinicia la transferencia.

MCBSP_XCR_XDATDLY_1BIT, // Retardo de 1 bit.

MCBSP_XCR_XFRLEN1_OF(0), //1 elemento por frame.

MCBSP_XCR_XWDLEN1_32BIT, //Elementos transmitidos de 32 bits.

MCBSP_XCR_XWDREVRS_DISABLE // Reversión de 32-bits del transmisor está desactivada.

),

MCBSP_SRGR_RMK(

MCBSP_SRGR_GSYNC_FREE, // El reloj del SRG (CLKG) está en modo *free-running.*

MCBSP_SRGR_CLKSP_RISING, // Flanco de subida de CLKS genera CLKG y FSG.

MCBSP_SRGR_CLKSM_INTERNAL, // El reloj del SRG es derivado del reloj de la CPU.

MCBSP_SRGR_FSGM_DXR2XSR, // Señal de *frame* de sincronización de transmisión se genera por la copia de los registros DXR-hacia-XSR. Registros FPER y FWID son ignorados.

MCBSP_SRGR_FPER_DEFAULT, //Valor por defecto.

MCBSP_SRGR_FWID_DEFAULT, //Valor por defecto.

MCBSP_SRGR_CLKGDV_OF(30) // Valor de división para generar la frecuencia del reloj del SRG.

),

MCBSP_MCR_DEFAULT, // Valor por defecto.

MCBSP_RCER_DEFAULT, // Valor por defecto.

MCBSP_XCER_DEFAULT, // Valor por defecto.

MCBSP_PCR_RMK(

MCBSP_PCR_XIOEN_SP, // DX, FSX, CLKX pines son configurados como pines de puerto serie y no funcionan como pines GPIO.

MCBSP_PCR_RIOEN_SP, // DR, FSR, CLKR pines son configurados como pines de puerto serie y no funcionan como pines GPIO.

MCBSP_PCR_FSXM_INTERNAL, // Señal de *frame* de sincronización es determinada por el bit FSGM en el registro SRGR.

MCBSP_PCR_FSRM_EXTERNAL, // Señal de *frame* de sincronización es derivada de una fuente externa. El pin FSR es un pin de entrada.

MCBSP_PCR_CLKXM_OUTPUT, // El pin CLKX es un pin de salida manejado por el SRG.

MCBSP_PCR_CLKRM_INPUT, // El pin CLKR es un pin de entrada manejado por una fuente externa.

MCBSP_PCR_CLKSSTAT_DEFAULT, // Valor por defecto.

MCBSP_PCR_DXSTAT_DEFAULT, // Valor por defecto.

MCBSP_PCR_FSXP_ACTIVEHIGH, //Pulso de *frame* de sincronización del transmisor se activa al flanco de subida.

MCBSP_PCR_FSRP_ACTIVEHIGH, // //Pulso de *frame* de sincronización del receptor se activa al flanco de subida.

MCBSP_PCR_CLKXP_RISING, // Datos se transmiten en el flanco de subida del CLKX.

MCBSP_PCR_CLKRP_FALLING Datos se reciben en el flanco de bajada del CLKR.

)

}

Si se quiere saber más y conocer mas a fondo todos los modos de operación del puerto McBSP se recomienda remitirse a las referencias bibliográficas [5] y [8].

2.2 DESCRIPCIÓN DE LOS ALGORITMOS

2.2.1 Algoritmo de la Transformada Rápida de Fourier 2-D en Paralelo. El algoritmo que realiza la operación de FFT 2-D de una matriz cuadrada contiene dos partes: Luego de que los datos se carguen a las tarjetas desde el PC, el algoritmo inicia con la primera parte que se encarga de realizar paralelamente la FFT 2-D de las filas de la matriz, seguidamente se intercambian los datos a través del McBSP y la segunda parte se encarga de realizar paralelamente la FFT 2-D de las columnas de la matriz obteniéndose así la matriz resultante FFT 2-D quedando almacenada en la memoria interna de la tarjeta. Un diagrama que muestra este procedimiento se presenta a continuación en la Figura 15:

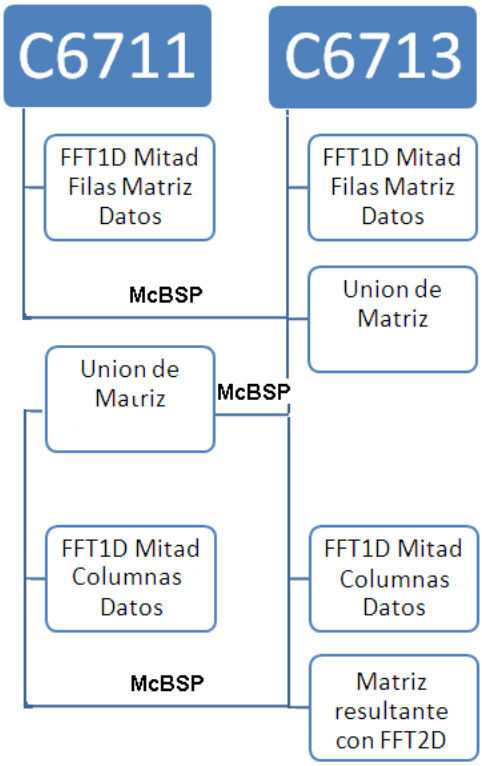


Figura 15. Diagrama para la FFT 2-D en paralelo.

Fuente: autores del proyecto.

El algoritmo en paralelo aquí utilizado para la FFT 2-D no fue usado de manera exacta sobre el algoritmo de convolución cíclica 2-D en paralelo el cual también debe realizar la operación de FFT 2-D. Sin embargo, este algoritmo sirve como base para la estructura general del algoritmo de convolución circular 2-D.

2.2.2 Algoritmo de la Convolución Cíclica 2-D en Paralelo. En la Figura 16 se muestra el proceso completo de la convolución circular en dos dimensiones de la matriz de datos crudos obtenidos del radar y la respuesta al impulso del mismo.

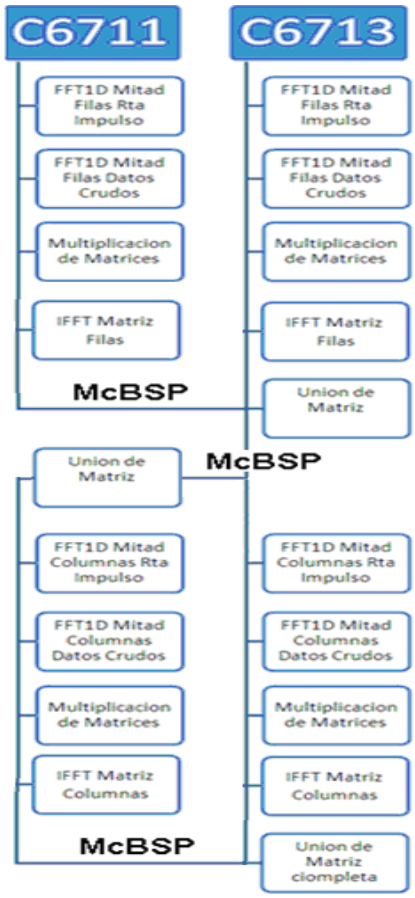


Figura 16. Diagrama de la Convolución 2-D en paralelo.

Fuente: autores del proyecto.

Como se observa en la Figura 16, la estructura general del algoritmo de convolución cíclica 2-D en paralelo es la misma que la del algoritmo para la FFT 2-D en paralelo. Luego de que se carguen los datos a las tarjetas (matriz de datos crudos y matrices de la respuesta al impulso) se aprecia de nuevo que simultáneamente los dos procesadores realizan la operación de convolución cíclica en la primera dimensión (filas), seguidamente intercambian la información necesaria a través de los puertos McBSP y luego realizan la operación de convolución cíclica en la segunda dimensión (columnas) para luego transferir sus resultados obteniéndose de esta manera la matriz de convolución cíclica en 2-D quedando almacenada en la memoria de la tarjeta. En este punto se obtiene la imagen deseada del radar.

Es necesario aclarar que para realizar este proceso, el radar SAR entrega una matriz cuya transformada de Fourier es usada para realizar la multiplicación por cada una de las filas de la transformada de Fourier de la matriz de datos; y otra matriz cuyas transformadas son usadas para multiplicarse con las columnas respectivas de la transformada de Fourier de la matriz obtenida luego de la transferencia de datos entre tarjetas. Este hecho se debe al funcionamiento del radar SAR y su justificación y explicación no hacen parte de este trabajo.

3. RESULTADOS DE LA IMPLEMENTACIÓN

A continuación se presentan los resultados obtenidos de la implementación de los algoritmos tanto secuenciales como en paralelo, que sirvieron para el procesamiento de imágenes de radares SAR. Aquí se muestran los tiempos de ejecución de cada algoritmo, tanto para la FFT 2-D como para la convolución cíclica 2-D; y se comparan las implementaciones secuenciales con la implementación en paralelo.

3.1 RESULTADOS DE LAS IMPLEMENTACIÓNES DE LA TRANSFORMADA RÁPIDA DE FOURIER 2-D SECUENCIAL Y EN PARALELO

Los tiempos de cómputo de los algoritmos mostrados en las Tablas 5, 6 y 7 fueron tomados para matrices de distintos tamaños hallándose la FFT 2-D secuencialmente para ambas tarjetas, y en paralelo. Estos tiempos se tomaron midiendo los ciclos de reloj mostrados en la ventana del CCS y dividiendo este valor entre la frecuencia de reloj del procesador respectivo.

Tabla 5 .Resultados implementación secuencial de la FFT 2-D en la C6711.

Tamaño de la matriz	Ciclos de reloj C6711	Tiempo de cómputo secuencial (s)
32x32	22.292.272	0.148
64x64	99.635.069	0.665
128x128	453.313.426	3.022
256x256	2.011.337.997	13.408
512x512	8.587.665.227	57.251

Tabla 6. Resultados implementación secuencial de la FFT 2-D en la C6713.

Tamaño de la matriz	Ciclos de reloj C6713	Tiempo de cómputo secuencial (s)
32x32	29.683.793	0.132
64x64	133.612.110	0.594
128x128	591.607.710	2.629
256x256	2.659.521.023	11.82
512x512	11.245.052.811	49.978

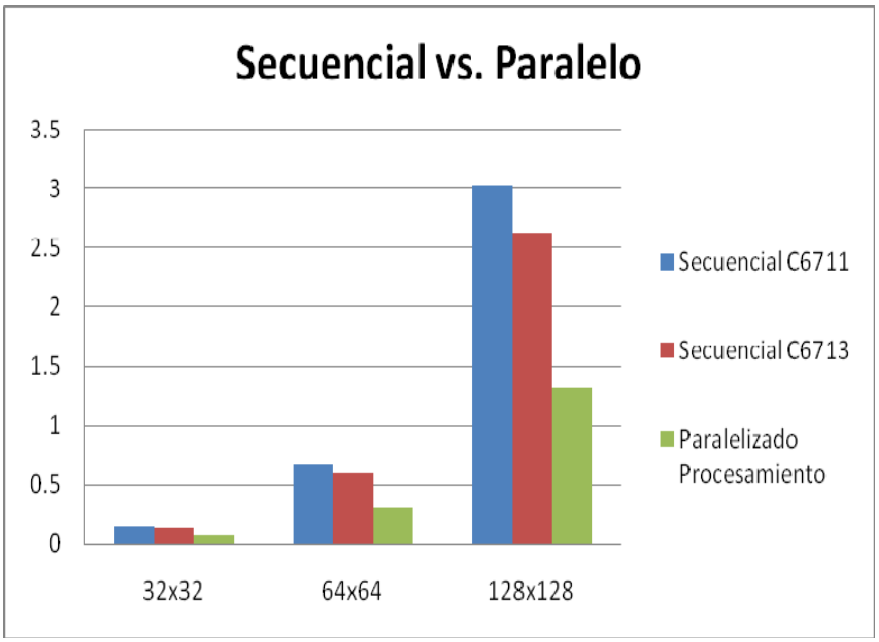
Tabla 7 . Resultados implementación paralelizada de la FFT 2-D.

Tamaño de la matriz	Ciclos de reloj C6713 Procesamiento	Tiempo de Procesa-miento (s)	Ciclos de reloj C6713 Transferencia	Tiempo de Transfe-rencia (s)	Ciclos de reloj C6713 Total	Tiempo de computo total (s)
32x32	14.981.354	0.066	278.279.972	1.23	293.261.326	1.303
64x64	67.290.152	0.299	295.793.466	1.315	363.083.618	1.614
128x128	297.485.327	1.322	410.303.018	1.824	707.788.345	3.145
256x256	1.306.465.385	5.806	794.300.732	3.53	2.100.766.117	9.337
512x512	5.537.948.747	24.613	2.841.075.001	12.625	8.379.342.270	37.24

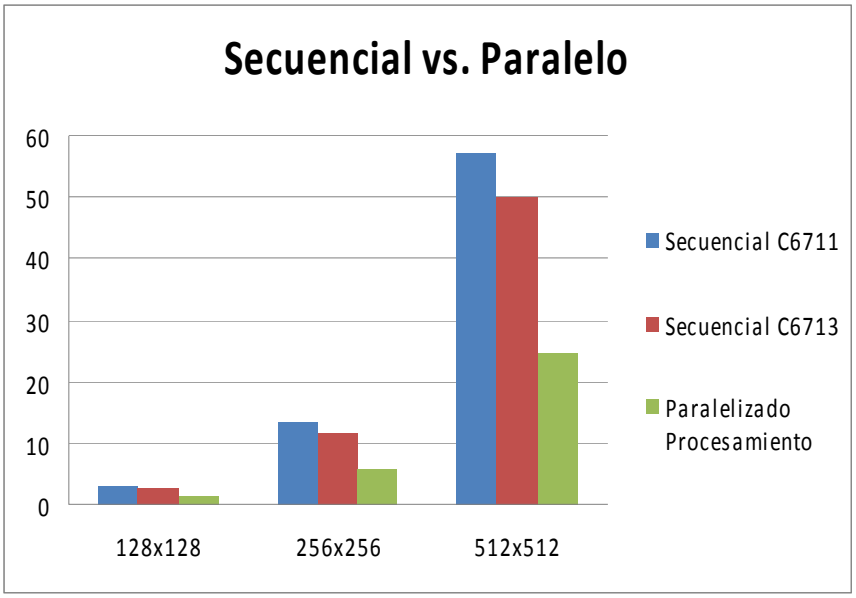
En las Tablas 5 y 6 para la implementación secuencial, la primera columna tiene el tamaño de las matrices utilizadas, la segunda columna muestra la cantidad de ciclos de reloj requeridos para el procesamiento de los datos y la tercera columna muestra el tiempo de ejecución en segundos del algoritmo de la FFT 2-D secuencial, para la C6711 y C6713, respectivamente. En la Tabla 7 para la implementación paralelizada, la primera columna contiene el tamaño de las matrices, la segunda columna muestra los ciclos de reloj solo para el

procesamiento, la tercera columna muestra el tiempo de procesamiento, la cuarta columna muestra los ciclos de reloj para la transferencia, la quinta columna muestra el tiempo de transferencia, la sexta columna muestra el numero de ciclos total y finalmente la séptima columna muestra el tiempo de computo total de la implementación completa.

En la Figura 17 a), se presentan estos resultados gráficamente comparando las matrices de tamaño 32x32, 64x64 y 128x128, y en la parte b), para las matrices de tamaño 128x128, 256x256 y 512x512; teniendo en cuenta las columnas 3 de las Tablas 5 y 6, con respecto a la columna 3 de la Tabla 7:



a)



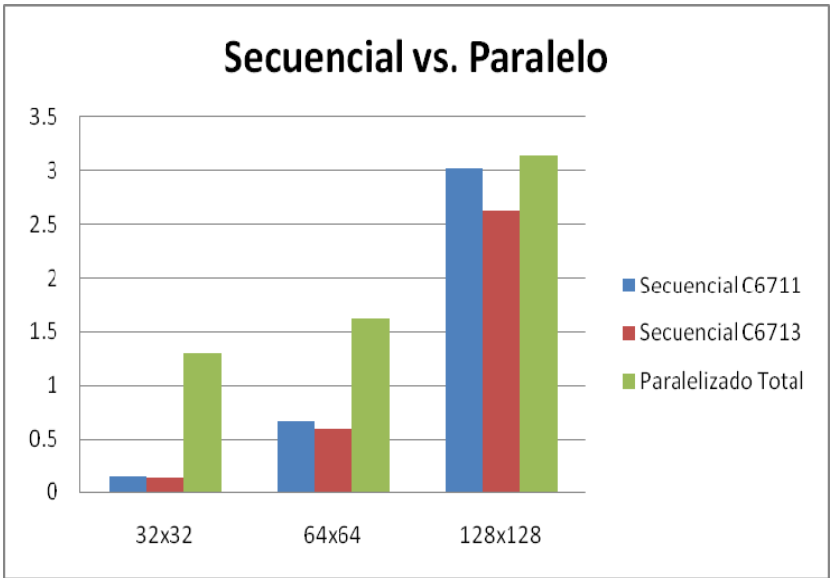
b)

Figura 17. Comparación de tiempos de procesamiento para la FFT 2-D.

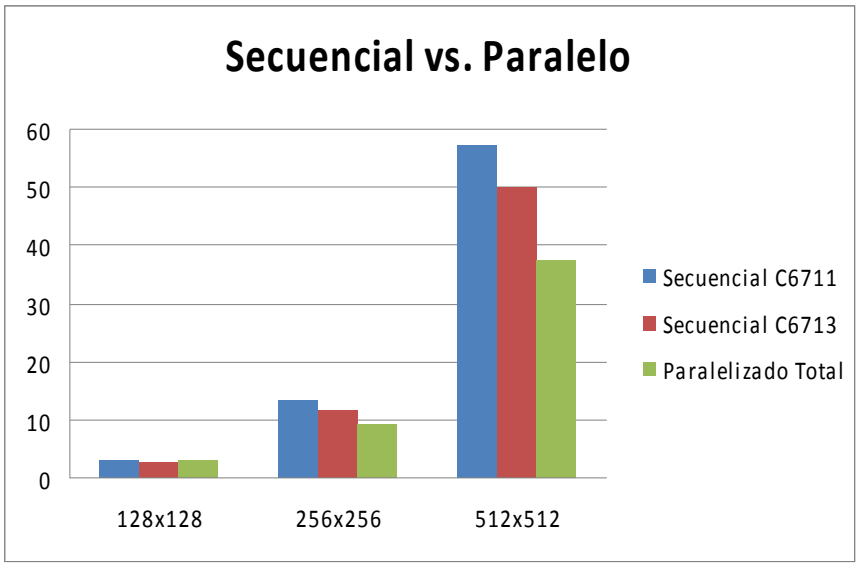
Fuente: autores del proyecto.

De estas gráficas y de la tablas anteriores se puede observar la diferencia entre los tiempos de cómputo del algoritmo FFT 2-D secuencial para la C6711 (azul) y C6713 (rojo), debido a que esta ultima posee una frecuencia de reloj mayor (225 MHz). Además, se aprecia la diferencia entre las dos implementaciones secuenciales con respecto a la paralelizada (verde), solo teniendo en cuenta el tiempo de procesamiento, es decir, omitiendo el tiempo que tardan las tarjetas en comunicarse. Nótese que el tiempo de procesamiento en paralelo es casi exactamente el 50% del tiempo de procesamiento secuencia tomando como referencia la C6713. El tiempo total de cómputo en paralelo equivale a un 880% del tiempo de procesamiento secuencial tomando como referencia la C6711 (ya que en el procesamiento paralelo influye el hecho de tener una tarjeta mas lenta que otra) para matrices de 32x32 y mejora notablemente disminuyendo hasta un 65% para matrices de tamaño 512x512.

En la Figura 18 se observa la comparación de los tiempos totales secuenciales y el tiempo de cómputo completo en la implementación paralelizada (verde), es decir, incluyendo tiempo de procesamiento y el tiempo de transferencia.



a)



b)

Figura 18. Comparación de tiempos totales para la FFT 2-D.

Fuente: autores del proyecto.

3.2 RESULTADOS DE LAS IMPLEMENTACIÓNES DE LA CONVOLUCIÓN CÍCLICA 2-D SECUENCIAL Y EN PARALELO

En las Tablas 8, 9 y 10 los tiempos de cómputo del algoritmo para realizar la convolución circular 2-D, tanto de manera secuencial implementados sobre la C6711 y C6713 respectivamente, como paralelizada implementada sobre los dos procesadores, para distintos tamaños de matrices desde 32x32 hasta matrices de 512x512:

Tabla 8. Resultados implementación de la convolución cíclica 2-D secuencial en la C6711.

Tamaño de la matriz	Ciclos de reloj C6711	Tiempo de cómputo secuencial (s)
32x32	66.190.954	0.441
64x64	279.812.537	1.865
128x128	1.180.864.275	7.873
256x256	5.268.702.750	35.125
512x512	21.301.686.751	142.011

Tabla 9. Resultados implementación de la convolución cíclica 2-D secuencial en la C6713.

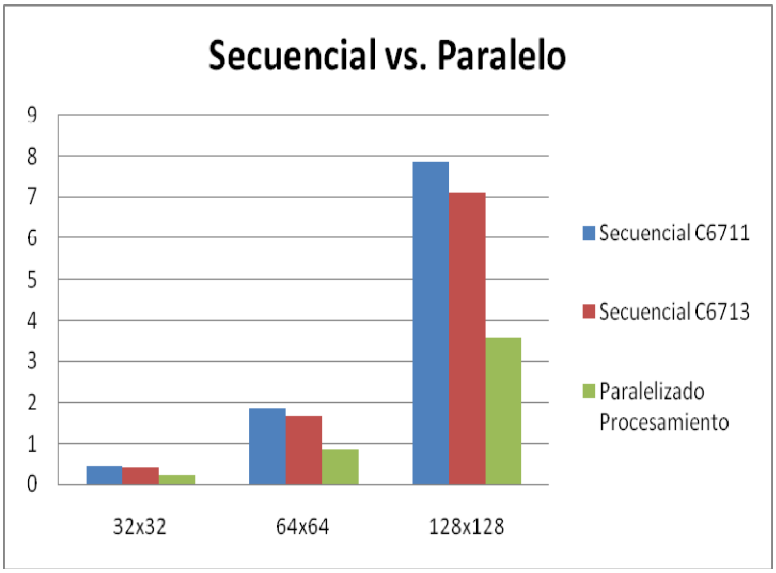
Tamaño de la matriz	Ciclos de reloj C6713	Tiempo de cómputo secuencial (s)
32x32	88.995.161	0.395
64x64	375.227.005	1.667
128x128	1.595.556.284	7.091
256x256	6.999.470.280	31.108
512x512	28.451.345.510	126.45

Tabla 10. Resultados de la implementación de la convolución cíclica 2-D paralelizada.

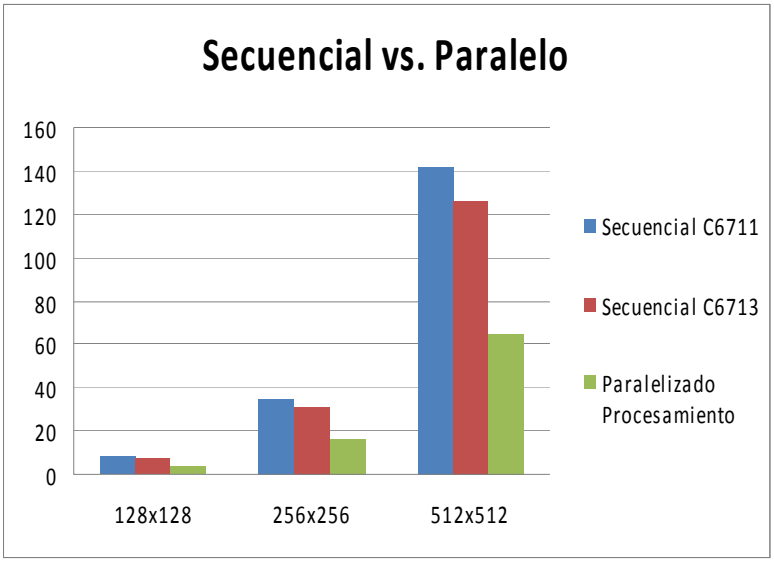
Tamaño de la matriz	Ciclos de reloj C6713 Procesamiento	Tiempo de Proce-samiento (s)	Ciclos de reloj C6713 Transferencia	Tiempo de Transfe-rencia (s)	Ciclos de reloj C6713 Total	Tiempo de cómputo total (s)
32x32	47.021.897	0.209	302.148.776	1.343	349.170.673	1.552
64x64	192.861.571	0.857	431.899.608	1.919	624.761.179	2.77
128x128	808.412.631	3.592	980.248.706	4.357	1.788.661.337	7.949
256x256	3.791.921.041	16.852	3.093.356.724	13.75	6.885.277.765	30.601
512x512	14.650.879.851	65.115	11.799.273.482	52.44	26.422.248.473	117.432

De nuevo, en las Tablas 8 y 9 para la implementación secuencial, la primera columna tiene el tamaño de las matrices utilizadas, la segunda columna muestra la cantidad de ciclos de reloj requeridos para el procesamiento de los datos y la tercera columna muestra el tiempo de ejecución en segundos del algoritmo de la Convolución cíclica 2-D secuencial, para la C6711 y C6713, respectivamente. En la Tabla 10 para la implementación paralelizada, la primera columna contiene el tamaño de las matrices, la segunda columna muestra los ciclos de reloj solo para el procesamiento, la tercera columna muestra el tiempo de procesamiento, la cuarta columna muestra los ciclos de reloj para la transferencia, la quinta columna muestra el tiempo de transferencia, la sexta columna muestra el numero de ciclos total y finalmente la séptima columna muestra el tiempo de computo total de la implementación completa para la Convolución cíclica 2-D.

En la Figura 19 a), se presentan estos resultados gráficamente comparando las matrices de tamaño 32x32, 64x64 y 128x128, y en la parte b), para las matrices de tamaño 128x128, 256x256 y 512x512; teniendo en cuenta las columnas 3 de las Tablas 8 y 9, con respecto a la columna 3 de la Tabla 10:



a)



b)

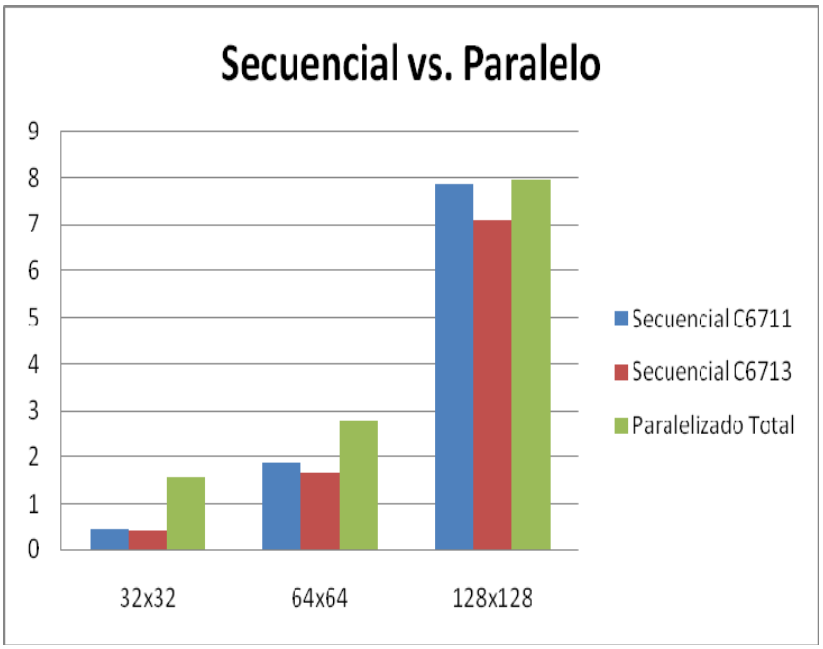
Figura 19. Comparación de tiempos procesamiento para la convolución 2-D.

Fuente: autores del proyecto.

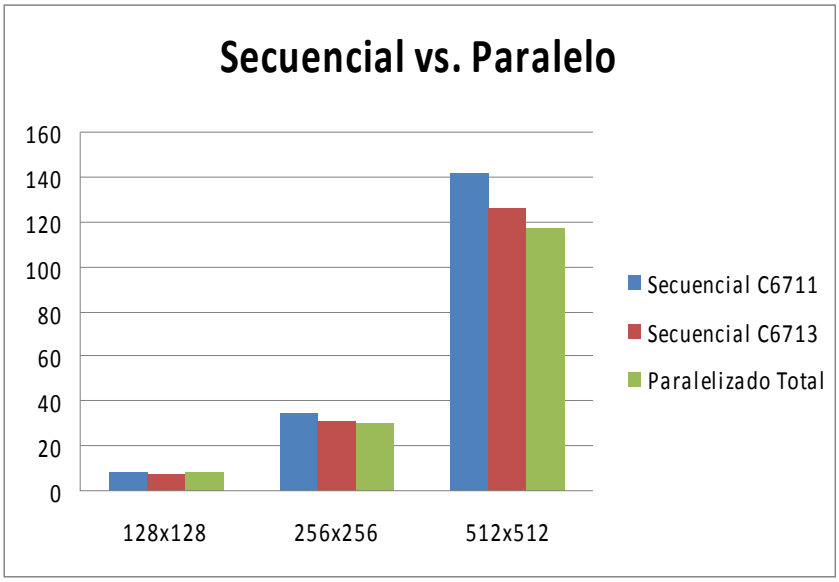
De esta gráfica y de las tablas anteriores se puede observar la diferencia entre los tiempos de ejecución del algoritmo de Convolución 2D secuencial para la C6711 (azul) y C6713 (rojo), debido a que esta ultima posee una frecuencia de reloj mayor (225 MHz). Además, se aprecia la diferencia entre las dos

implementaciones secuenciales con respecto a la paralelizada (verde), solo teniendo en cuenta el tiempo de procesamiento, es decir, omitiendo el tiempo que tardan las tarjetas en comunicarse. De nuevo, el tiempo de procesamiento en paralelo es prácticamente el 50% del tiempo de procesamiento secuencial tomando como referencia la C6713.

En la Figura 20 se observa la comparación de los tiempos totales secuenciales con el tiempo de cómputo completo en la implementación paralelizada de la convolución 2-D (verde), es decir, incluyendo tiempo de procesamiento y el tiempo de transferencia.



a)



b)

Figura 20. Comparación de tiempos totales para la convolución 2-D.

Fuente: autores del proyecto

Nótese que para matrices de 32x32 el tiempo de cómputo en paralelo equivale a un 352% del tiempo de cómputo secuencial tomando como referencia la C6711 (ya que ésta por ser mas lenta influye en el tiempo de cómputo en paralelo) y mejora su eficiencia notablemente a medida que el tamaño de la matriz aumenta pasando a ser un 83% del tiempo de cómputo secuencial para matrices de tamaño 512x512, esta mejora progresiva en la eficiencia se mantendrá para matrices de mayor tamaño.

Todos estos resultados obtenidos en este trabajo fueron validados usando MATLAB el cual permitió verificar paso a paso el correcto funcionamiento de los algoritmos.

4. CONCLUSIONES

- Se implementó exitosamente el algoritmo de la FFT 2-D en paralelo, reduciendo el tiempo de procesamiento a la mitad para un conjunto de matrices de dimensiones 32x32, 64x64, 128x128, 256x256 y 512x512. De igual manera a partir de matrices de tamaño 256x256 se observa el menor tiempo requerido total para el procesamiento en paralelo, incluyendo procesamiento y transferencia de los datos en comparación con las implementaciones secuenciales.

- Se implementó exitosamente el algoritmo de la convolución cíclica 2-D en paralelo y de igual manera se redujo el tiempo de procesamiento en la mitad en comparación con implementaciones secuenciales. Igualmente, a partir de matrices de tamaño 256x256 el tiempo de cómputo total en paralelo es inferior al tiempo de cómputo de las implementaciones secuenciales. De esta manera se logra una mejor aproximación al procesamiento en tiempo real que se desea alcanzar en radares de tipo SAR.

- Se validó la implementación de los algoritmos con datos reales provenientes de un radar SAR por medio del software MATLAB. Esta herramienta fue útil para comparar los resultados que se obtienen en este software de desarrollo con respecto a los obtenidos en el procesamiento en paralelo con los DSPs en CCS.

- Se aprovecharon las características especiales del ambiente CCS, como son la posibilidad de revisar variables y registros desde el DSK y su utilidad para intercambiar información entre las tarjetas y los algoritmos los cuales se implementaron sobre el lenguaje de alto nivel C.

- Se demostró que el puerto McBSP posee muy buenas características a la hora de transmitir los datos entre las tarjetas. No obstante, este puerto requiere la manipulación de los registros de datos y control para dicha transmisión, para lograr una eficiente comunicación entre las tarjetas a través de él.

- Se evidenció la capacidad que posee la familia de procesadores de TI TMS320C6000 cuando se refiere a interconectar más de una tarjeta con el puerto de comunicación de alta velocidad McBSP, contando con una señal de reloj y un *frame* de sincronización para la comunicación más eficiente.

- En esta implementación en paralelo solo fueron requeridas las herramientas de CCS y los DSPs para llevar a cabo la transferencia y procesamiento de datos y sin necesidad de usar ningún tipo de comunicación con el PC para el procesamiento ni hardware alguno para acelerar la comunicación de datos (Blackhawk Emulator, etc.) los cuales son dispositivos de alto costo. El PC fue utilizado únicamente para diseñar, compilar y cargar los algoritmos sobre las tarjetas, mas no para el procesamiento de los datos.

- El análisis a nivel lógico de las salidas y entradas de los puertos del procesador, con el propósito de observar físicamente las señales que se transmiten o reciben requieren el uso de una herramienta como el analizador lógico digital dado que software Code Composer Studio solo permite observar los valores de los registros internos de las tarjetas. Esta herramienta fue muy útil para observar la correcta operación de los puertos McBSP.

5. RECOMENDACIONES

- Analizar la posibilidad de incrementar la capacidad de almacenamiento de las tarjetas, ya sea por medio del conector de expansión (external memory interface) de memoria o a través de algún tipo de disco duro conectado a la tarjeta.

- Analizar la posibilidad de implementar un controlador DMA/EDMA con el cual, teniéndose una adecuada configuración del mismo, puede operar continuamente sin ningún de intervención directa de la CPU, permitiendo que la CPU aplique toda su potencialidad solo en el procesamiento de la información mientras que el controlador se utilice únicamente para el manejo de la transferencia de datos.

- Si se requiere usar más de dos procesadores (DSPs), analizar la posibilidad del uso de TDM (*Time Division Multiplexing*) o multiplexaje por división de tiempo para la transferencia de los datos a través de los puertos McBSP. Este es uno de los tipos de multiplexación más utilizado en la actualidad, especialmente en los sistemas de transmisión digitales.

- Existen muchas bases de información (notas de aplicación, notas técnicas, guías de usuario, guías personalizadas y *papers*) que pueden ser de gran ayuda a la hora de trabajar con DSPs de la familia TMS320C6000, como son los sitios de internet www.dsprelated.com, www.ti.com y www.spectrumdigital.com, donde además se cuenta con actualizaciones para el Code Composer Studio y *drivers* para las tarjetas.

REFERENCIAS BIBLIOGRÁFICAS

[1] Chassaing Rulph. Digital Signal Processing and Applications with the C6713 and C6416 DSK. Wiley- Interscience. 2005.

[2] Ardila Ender, Puerto Luis F. Implementación de Algoritmos para el Procesamiento de Imágenes de radar SAR usando DSPs. Trabajo de grado. Universidad Industrial de Santander, escuela de ingenierías eléctrica, electrónica y de telecomunicaciones. Febrero de 2008.

[3] Texas Instruments, TMS320C6713 Floating-Point Digital Signal Processor, noviembre de 2005

[4] Texas Instruments, TMS320C6713 DSK, Technical Reference, Spectrum Digital Inc, noviembre de 2003.

[5] Texas Instruments**,** TMS320C6000 DSP Multichannel Buffered Serial Port (McBSP), Reference Guide, diciembre de 2006.

[6] Texas Instruments, TMS320C6000 Peripherals, Reference Guide, febrero de 2001.

[7] Texas Instruments, Using the TMS320C6000 McBSP as a High Speed Communication Port, Application Report, agosto de 2001.

[8] Texas Instruments, TMS320C6000 Chip Support Library, API Reference Guide, agosto de 2004.

[9] Platonov Alexei. Aplicación de Imágnes de Satélite SAR, en estudios de contaminación marina y de dinámica de las aguas en el mediterraneo occidental, Seccion 2.3 Satelites_con_SAR.pdf. http://www.tdcat.cesca.es/TESIS_UPC/AVIABLE/TDX-0905102-135541//2_3_Satelites_con_SAR.pdf. Julio de 2002

[10] Chassaing Rulph. DSP Applications Using C and the TMS320C6x DSK. John Wiley & Sons, INC. 2002.

[11] Texas Instruments, TMS320C6000 McBSP Initialization, Mayo de 2004.

[12] Ramírez S. Ana B. On Implementing Time-Frecuency Representations On Hardware/Software Computational Structures For SAR Applications. University Of Puerto Rico Mayagüez Campus. Junio de 2006.

[13] Franceschetti,G. and Schirinzi,G. A SAR Processor Based on Two Dimensional FFT Codes. IEEE Transactions on Aerospace and Electronics Systems. 1990.

ANEXOS

Anexo A. ALGORITMO SECUENCIAL PARA LA TRANSFORMADA DE FOURIER EN DOS DIMENSIONES

```
/*-------------------------------------------------------------------------------------------------------*/
/*  secuencial_FFT2D.c                                                                          * /
This program makes the two-dimensional fast fourier transform of any                          */
complex square matrix.                                                                        */
/*-------------------------------------------------------------------------------------------------------*/
/* Include files  */
#include "math.h"
#include "dataW512.h"     /* w  data file  */
#include "data512.h"    /* data file  */
-------------------------------------------------------------------------------------------------------
/*  Define constants */
#define MATRIX_ROWS 512
#define MATRIX_COLUMNS 1024
-------------------------------------------------------------------------------------------------------
/*  Define global variables */
far float aux_matrix[MATRIX_ROWS][MATRIX_COLUMNS];
far float aux_vector1[MATRIX_COLUMNS];
unsigned int N, rows, columns, i,j;
```

```
-------------------------------------------------------------------------------------------------------

/*  Define functions */

void FFT_TI(void)    /* Fast Fourier Transform  */

{

N = MATRIX_COLUMNS/2;

DSPF_sp_cfftr2_dit((float*)aux_vector1, (float*)w,N);

bit_rev((float*)aux_vector1,(2*N)>>1);

}

-------------------------------------------------------------------------------------------------------

/*  Main  */

void main(void)

{

rows = MATRIX_ROWS;   /*  Define local variables for rows and columns */

columns = MATRIX_COLUMNS;

for(i=0;i<rows;i++)    {

  for(j=0;j<columns;j++)    {

    aux_vector1[j]=data[i][j];     /* extracted row by row in the data matrix */

}

  FFT_TI();      /* Call to Fast Fourier Transform function  */

  for (j=0;j<columns;j++)    {
```

```
    data[i][j]=aux_vector1[j];    }   /* Place the processed data back into the  matrix  */
}
for(i=0;i<columns;i=i+2)    {
  for(j=0;j<rows;j++)     {
    aux_vector1[2*j]=data[j][i];    /* extracted column by column in the data matrix */
    aux_vector1[2*j+1]=data[j][i+1];    }
  FFT_TI();     /* Call to Fast Fourier Transform function  */
  for(j=0;j<rows;j++)    {
    aux_matrix[j][i]=aux_vector1[2*j]; /* Place the processed data into the  auxiliary matrix  */
    aux_matrix[j][i+1]=aux_vector1[2*j+1];  }
}
}
/*------------------End of secuencial_FFT2D.c--------------------*/
```

Anexo B. ALGORITMO SECUENCIAL PARA LA CONVOLUCIÓN CÍCLICA EN DOS DIMENSIONES

```
/*-------------------------------------------------------------------------------------------------------*/
/* secuencial_conv2D.c                                                                                   */
This l program makes the two-dimensional  circular  convolution.                                          */
/*-------------------------------------------------------------------------------------------------------*/
/* Include files  */
#include "math.h"
#include "dataW64.h"    /* w elements  data file  */
#include "data64.h"   /* data matrix file  */
#include "ri_columns64.h"  /* impulse response matrix files  */
#include "ri_rows64.h"
---------------------------------------------------------------------------------------------------------
/*  Define constants  */
#define MATRIX_ROWS 64
#define MATRIX_COLUMNS 128
---------------------------------------------------------------------------------------------------------
/*  Define global variables  */
far float aux_columns[MATRIX_ROWS][MATRIX_COLUMNS];
far float aux_matrix2[MATRIX_ROWS][MATRIX_COLUMNS];
```

```
float aux_vector1[MATRIX_COLUMNS];

float aux_vector2[MATRIX_COLUMNS];

float aux_vector3[2*MATRIX_COLUMNS];

unsigned int N, rows, columns, i, j, k, z, x, y, pe;

-----------------------------------------------------------------------------------------------------------

/*  Define functions  */

void FFT_TI(void)  /*  Fast Fourier Transform function  */

{

  N= MATRIX_COLUMNS/2;

  DSPF_sp_cfftr2_dit((float*)aux_vector1, (float*)w,N);

  bit_rev((float*)aux_vector1,(2*N)>>1);

}

void IFFT_TI (void)  /* Inverse Fast Fourier Transform function  */

{

  N = MATRIX_COLUMNS/2;

  In_DSPF_sp_cfftr2_dit((float*) aux_vector3,(float*) w, N);

  bit_rev((float *)aux_vector3, (2 * N)>>1);

  for(i=0;i<4*N;i++) {

    aux_vector3[i]= aux_vector3[i]/N;                    }

}
```

```
void PRODUCT (void)  /*  Matrix product  function  */

{

  columns = MATRIX_COLUMNS;

  for(j=0;j<(columns-1);j++) {

  aux_vector3[2*j]=  aux_vector1[2*j]  *  aux_vector2[2*j]-  aux_vector1[2*j+1]  *
aux_vector2[2*j+1];

  aux_vector3[2*j+1]= aux_vector1[2*j] * aux_vector2[2*j+1]+ aux_vector1[2*j+1] *
aux_vector2[2*j];                          }

}

-------------------------------------------------------------------------------------------------------
/*  Main  */

void main(void)

{

  rows = MATRIX_ROWS;  /*  Define local variables  */

  columns = MATRIX_COLUMNS;

  for(i=0;i<columns;i++)  {

        aux_vector1[i]=ri_rows[i];   }

        FFT_TI();   /* Make the FFT of the impulse response matrix by rows*/

         for(i=0;i<columns;i++)   {

         aux_vector2[i]=aux_vector1[i];     }   /*   Placed   in   the   appropiate   vector   to
multiply*/

       for(i=0;i<rows;i++) {
```

```
        for(j=0;j<columns;j++) {
          aux_vector1[j]=data[i][j];                    }
        FFT_TI();       /* Make the FFT of the data matrix by rows*/
       PRODUCT();      /* Multiply the two matrices  */
      for (j=0;j<columns;j++) {
    data[i][j]=aux_vector3[j];                    }
}
    for(i=0;i<2*columns;i++)  {   /* Deletes  the auxilary vector  */
            aux_vector3[i]=0;   }
    for(z=0;z<rows;z++)  {
    for(k=0;k<columns;k++)      {
         aux_vector3[k]=data[z][k];              }
    IFFT_TI();   /* Make the IFFT of the matrix by rows*/
    for (k=0;k<columns;k++)    {
    data[z][k]=aux_vector3[k];      }
}
   for(i=0;i<9;i++)  {
            for(j=0;j<columns/2;j++)   {  /* Conjugate the impulse response matrix  */
         aux_columns[i][2*j]=ri_columns[i][2*j];
        aux_columns[i][2*j+1]=ri_columns[i][2*j+1]*(-1);  }
```

```
}
    for(i=0;i<9;i++)  {
     for(j=0;j<columns;j++)  {
         aux_vector1[j]=aux_columns[i][j];  }
                    FFT_TI();   /* Make the FFT of the matrix by columns*/
           for(j=0;j<columns;j++)  {
          ri_columns[i][j]=aux_vector1[j];             }
      }
            for(i=0;i<columns;i++)  {
           aux_vector2[i]=0;   }
   pe=(int)(rows/9);
     for(i=0;i<columns;i=i+2) {
        for(j=0;j<rows;j++)   {
        aux_vector1[2*j]=data[j][i];
        aux_vector1[2*j+1]=data[j][i+1];     }
      FFT_TI();  /* Make the FFT of  matrix by columns*/
      if(i<2*pe) {   /* Select the appropiate row of the impulse response matrix to
multiply */
           x=0; }
      else if(i>2*pe-1 && i<4*pe) {
           x=1; }
```

```
    else if(i>4*pe-1 && i<6*pe) {
    x=2; }
    else if(i>6*pe-1 && i<8*pe) {
    x=3; }
    else if(i>8*pe-1 && i<10*pe) {
    x=4; }
     else if(i>10*pe-1 && i<12*pe) {
    x=5; }
    else if(i>12*pe-1 && i<14*pe) {
    x=6; }
    else if(i>14*pe-1 && i<16*pe) {
    x=7; }
    else if(i>16*pe-1) {
    x=8; }
     for(y=0;y<columns/2;y++)   {
   aux_vector2[2*y]=ri_columns[x][2*y];
  aux_vector2[2*y+1]=ri_columns[x][2*y+1]*(-1);  }
    PRODUCT();    /*Multiply the two matrix  */
for(j=0;j<rows;j++)   {
  aux_matrix2[j][i]=aux_vector3[2*j];
```

```
        aux_matrix2[j][i+1]=aux_vector3[2*j+1];     }
}
     for(i=0;i<2*columns;i++)  {
            aux_vector3[i]=0;   }
    for(k=0;k<columns;k=k+2)   {
        for(z=0;z<rows;z++)   {
       aux_vector3[2*z]=aux_matrix2[z][k];
      aux_vector3[2*z+1]=aux_matrix2[z][k+1];               }
           IFFT_TI();  /* Make the IFFT of the matrix by columns*/
          for(z=0;z<rows;z++)   {
        data[z][k]=aux_vector3[2*z];
        data[z][k+1]=aux_vector3[2*z+1];     }
}
}  /*------------------End of secuencial_conv2D.c--------------------*/
```

Anexo C. ALGORITMOS EN PARALELO DE LA TRANSFORMADA RÁPIDA DE FOURIER EN DOS DIMENSIONES PARA LA C6711 Y LA C6713

```
/*-------------------------------------------------------------------------------------------------------*/
/* parallel_FFT2D_6711.c                                                                                 */
/* This program makes the two-dimenstional  Fast Fourier  Transform in a parallel
way for the C6711 using the McBSP1 port to communicate with C6713.                                        */
/*-------------------------------------------------------------------------------------------------------*/
/*  Include files  */
#include <csl.h>
#include <csl_mcbsp.h>
#include "math.h"
#include "dataW32.h"   /* w elements data file  */
#include "data32.h"      /* data file  */
-------------------------------------------------------------------------------------------------------
/*Define constants */
#define MATRIX_ROWS 32
#define MATRIX_COLUMNS 64
-------------------------------------------------------------------------------------------------------
/* Define global variables  */
far float aux_matrix[MATRIX_ROWS][MATRIX_COLUMNS];
float aux_vector1[MATRIX_COLUMNS];
float indata,outdata;
unsigned int N, rows, columns, i,j;
-------------------------------------------------------------------------------------------------------
/*Declare CSL objects */
MCBSP_Handle hMcbsp1;   /* Handles for McBSP1 port  */
-------------------------------------------------------------------------------------------------------
/* Declare external functions and functions prototypes  */
```

```
extern MCBSP_Handle mcbsp_config(MCBSP_Handle);
void FFT_TI(void)     /*Fast Fourier Transform function */
{
  N = MATRIX_COLUMNS/2;
  DSPF_sp_cfftr2_dit((float*)aux_vector1, (float*)w,N);
  bit_rev((float*)aux_vector1,(2*N)>>1);
}
void MCBSP_TRANSMITTER(void)  /* McBSP transmitter function  */
{
  hMcbsp1 = mcbsp_config(hMcbsp1); /* Configures the McBSP1 port  */
  MCBSP_enableSrgr(hMcbsp1);       /* Enable the sample rate generator */
  for (i=0; i<0x10; i++);
  MCBSP_enableXmt(hMcbsp1);  /* Enable  McBSP1 port as the transmitter */
  MCBSP_enableFsync(hMcbsp1); /* Enable frame syncronization */
  return;
}
void MCBSP_RECIEVER(void)  /* McBSP reciever function  */
{
  hMcbsp1 = mcbsp_config(hMcbsp1); /* Configures the McBSP port */
  MCBSP_enableRcv(hMcbsp1); /* Enable  McBSP1 port as the reciever */
  return;
}
-------------------------------------------------------------------------------------------------------
/*  Main  */
void main(void)
{
  rows = MATRIX_ROWS;   /* Define local variables  */
  columns = MATRIX_COLUMNS;
    CSL_init();   /* Initialize the CSL libraries  */
      for(i=0;i<rows/2;i++)  {
```

```
        for(j=0;j<columns;j++)  {
          aux_vector1[j]=data[i][j];
                              }
        FFT_TI();   /* Makes the FFT of the of the first  rows  in the data matrix */
      for (j=0;j<columns;j++)  {
    data[i][j]=aux_vector1[j];                          }
                      }
for (i=0;i<0x100000;i++);
MCBSP_TRANSMITTER();   /* Transmit the data */
  for (i=0;i<rows/2;i++)  {
    for (j=columns/2;j<columns;j++) {
      outdata=data[i][j];
      while(!MCBSP_xrdy(hMcbsp1));
      MCBSP_write(hMcbsp1, outdata);                      }
                    }
  for (i=0; i<0x10000; i++);
  MCBSP_close(hMcbsp1);   /* close the port  */
  MCBSP_RECIEVER();  /* Recieve the data  */
for (i=rows/2;i<rows;i++) {
for (j=0;j<columns/2;j++) {
  while(!MCBSP_rrdy(hMcbsp1));
  indata = MCBSP_read(hMcbsp1);
  data[i][j]=indata;   }
}
  MCBSP_close(hMcbsp1);   /* close the port  */
     for(i=0;i<columns/2;i=i+2) {
        for(j=0;j<rows;j++)   {
        aux_vector1[2*j]=data[j][i];
        aux_vector1[2*j+1]=data[j][i+1];     }
       FFT_TI();  /* Makes the FFT of the first columns of the matrix   */
```

```
        for(j=0;j<rows;j++)   {
         aux_matrix[j][i]=aux_vector1[2*j];
         aux_matrix[j][i+1]=aux_vector1[2*j+1];     }
}
for (i=0;i<0x100000;i++);
MCBSP_TRANSMITTER();  /* Transmit data  */
  for (i=0;i<columns/2;i++)  {
    for (j=0;j<rows;j++) {
      outdata=aux_matrix[j][i];
      while(!MCBSP_xrdy(hMcbsp1));
      MCBSP_write(hMcbsp1, outdata);                    }
                  }
  for (i=0; i<0x10000; i++);
  MCBSP_close(hMcbsp1);   /* close the port  */
}   /*------------------End of parallel_FFT2D_6711.c--------------------*/


/*-------------------------------------------------------------------------------------------------------*/
/* parallel_FFT2D_6713.c                                                                                 */
/* This program makes the two-dimenstional  Fast Fourier  Transform in a parallel
way for the C6713 using the McBSP0 port to communicate with C6711.              */
/*-------------------------------------------------------------------------------------------------------*/
/*  include files  */
#include <csl.h>
#include <csl_mcbsp.h>
#include "math.h"
#include "dataW32.h "  /* w elements data file  */
#include "data32.h"   /* data file  */
-------------------------------------------------------------------------------------------------------
```

```
/* Define constants  */
#define DSK_MISC_REG (*(volatile unsigned char *)(0x90080006)) /* MISC register  */
#define MATRIX_ROWS 32
#define MATRIX_COLUMNS 64
-----------------------------------------------------------------------------------------------------

/* Define global variables  */
far float aux_matrix[MATRIX_ROWS][MATRIX_COLUMNS];
float aux_vector1[MATRIX_COLUMNS];
float indata, outdata;
unsigned int  N, rows, columns, i,j;
MCBSP_Handle hMcbsp0;   /* Handles for McBSP  */
-----------------------------------------------------------------------------------------------------
/*  Declare external functions and functions prototypes  */
extern MCBSP_Handle mcbsp_config(MCBSP_Handle);
void FFT_TI(void)  /* Fast Fourier Transform function  */
{
  N = MATRIX_COLUMNS/2;
  DSPF_sp_cfftr2_dit((float*)aux_vector1, (float*)w,N);
  bit_rev((float*)aux_vector1,(2*N)>>1);
}
void MCBSP_RECIEVER(void)  /* McBSP reciever function  */
{
  hMcbsp0 = mcbsp_config(hMcbsp0);  /* Configures the McBSP0 port */
  MCBSP_enableRcv(hMcbsp0);  /* Enable the McBSP0 port as the reciever */
  return;
}
void MCBSP_TRANSMITTER(void)  /* McBSP transmitter function  */
{
```

```
  hMcbsp0 = mcbsp_config(hMcbsp0); /* Configures the McBSP0 port */
  MCBSP_enableSrgr(hMcbsp0);       /* Enable sample rate generator */
  for (i=0; i<0x10; i++);
  MCBSP_enableXmt(hMcbsp0); /*Enable the McBSP0 port as the transmitter */
  MCBSP_enableFsync(hMcbsp0); /*Enable frame syncronization */
  return;
}
-------------------------------------------------------------------------------------------------------
/*  Main  */
void main(void)
{
  rows = MATRIX_ROWS;   /*Define local variables  */
  columns = MATRIX_COLUMNS;
    CSL_init();  /* Initialize CSL libraries  */
     DSK_MISC_REG |= 1;  /* Sets the appropiate bit in MISC register to use the
McBSP0 port for communication */
      for(i=rows/2;i<rows;i++)  {
        for(j=0;j<columns;j++)  {
          aux_vector1[j]=data[i][j];                          }
        FFT_TI();  /*  Makes the FFT of the last rows in the data matrix  */
      for (j=0;j<columns;j++)  {
    data[i][j]=aux_vector1[j];                        }
                    }
  MCBSP_RECIEVER();  /* Recieve data  */
for (i=0;i<rows/2;i++) {
for (j=columns/2;j<columns;j++) {
  while(!MCBSP_rrdy(hMcbsp0));
indata = MCBSP_read(hMcbsp0);
  data[i][j]=indata;   }
}
```

```
  MCBSP_close(hMcbsp0);   /* close the port  */
for (i=0;i<0x100000;i++);
 MCBSP_TRANSMITTER();  /* Transmit data  */
  for (i=rows/2;i<rows;i++)  {
    for (j=0;j<columns/2;j++) {
      outdata=data[i][j];
      while(!MCBSP_xrdy(hMcbsp0));
      MCBSP_write(hMcbsp0, outdata);                    }
                    }
  for (i=0; i<0x10000; i++);
  MCBSP_close(hMcbsp0);   /* close the port  */
      for(i=columns/2;i<columns;i=i+2) {
        for(j=0;j<rows;j++)   {
        aux_vector1[2*j]=data[j][i];
        aux_vector1[2*j+1]=data[j][i+1];      }
       FFT_TI();  /* Makes the FFT of the last columns of the matrix  */
        for(j=0;j<rows;j++)   {
        aux_matrix[j][i]=aux_vector1[2*j];
        aux_matrix[j][i+1]=aux_vector1[2*j+1];      }
}
  MCBSP_RECIEVER();  /* Recieve data  */
for (i=0;i<columns/2;i++) {
for (j=0;j<rows;j++){
  while(!MCBSP_rrdy(hMcbsp0));
  indata = MCBSP_read(hMcbsp0);
  aux_matrix[j][i]=indata;   }
}
  MCBSP_close(hMcbsp0);   /*close port  */
}     /*------------------End of parallel_FFT2D_6713.c----------------------*/
```

Anexo D. ALGORITMOS EN PARALELO DE LA CONVOLUCIÓN EN DOS DIMENSIONES PARA LA C6711 Y LA C6713

```
/*-------------------------------------------------------------------------------------------------------*/
/* parallel_conv2D_6711.c                                                                                */
/* This program makes the two-dimenstional  Fast Fourier  Transform in a                                 */
parallel way for the C6711 using the McBSP1port to communicate with C6713.   */
/*-------------------------------------------------------------------------------------------------------*/
/*Include files  */
#include <csl.h>
#include <csl_mcbsp.h>
#include "math.h"
#include "dataW32.h"  /* w elements data file */
#include "data32.h"  /*data file */
#include "ri_columns32.h"  /* impulse response matrix by columns */
#include "ri_rows32.h"  /* impulse response matrix by rows */
-------------------------------------------------------------------------------------------------------
/*Define constants  */
#define MATRIX_ROWS 32
#define MATRIX_COLUMNS 64
-------------------------------------------------------------------------------------------------------
/* Define global variables */
far float aux_columns[MATRIX_ROWS][MATRIX_COLUMNS];
far float aux_matrix[MATRIX_ROWS][MATRIX_COLUMNS];
float aux_vector1[MATRIX_COLUMNS];
float aux_vector2[MATRIX_COLUMNS];
float aux_vector3[2*MATRIX_COLUMNS];
float indata, outdata;
unsigned int N, rows, columns, i, j, k, z, x, y, pe;
```

```
-----------------------------------------------------------------------------------------------------
/*Declare external functions and functions prototypes  */
extern MCBSP_Handle mcbsp_config(MCBSP_Handle);
MCBSP_Handle hMcbsp1;   /*Handles for McBSP   */
void MCBSP_RECIEVER(void)  /*Reciever function  */
{
  hMcbsp1 = mcbsp_config(hMcbsp1);  /* Configure the McBSP1 port  */
  MCBSP_enableRcv(hMcbsp1);  /* Enable McBSP1 port as the reciever */
  return;
}
void MCBSP_TRANSMITTER(void)  /*Transmiter function  */
{
  hMcbsp1 = mcbsp_config(hMcbsp1);  /* Configure the McBSP1 port  */
  MCBSP_enableSrgr(hMcbsp1);         /* Enable sample rate generator */
  for (i=0; i<0x10; i++);
  MCBSP_enableXmt(hMcbsp1);  /* Enable McBSP1 port as the transmitter */
  MCBSP_enableFsync(hMcbsp1); /* Enalbe frame syncronization  */
  return;
}
void FFT_TI(void)  /* Fast Fourier Transform function */
{
  N= MATRIX_COLUMNS/2;
  DSPF_sp_cfftr2_dit((float*)aux_vector1, (float*)w,N);
  bit_rev((float*)aux_vector1,(2*N)>>1);
}
void IFFT_TI (void)  /* Inverse Fast Fourier Transform function  */
{
  N = MATRIX_COLUMNS/2;
  In_DSPF_sp_cfftr2_dit((float*) aux_vector3,(float*) w, N);
  bit_rev((float *)aux_vector3, (2 * N)>>1);
```

```
  for(i=0;i<4*N;i++) {
    aux_vector3[i]= aux_vector3[i]/N;
                  }
}
void PRODUCT (void)  /* Product function   */
{
  columns = MATRIX_COLUMNS;
  for(j=0;j<(columns-1);j++) {
  aux_vector3[2*j]=  aux_vector1[2*j]  *  aux_vector2[2*j]-  aux_vector1[2*j+1]  *
aux_vector2[2*j+1];
  aux_vector3[2*j+1]= aux_vector1[2*j] * aux_vector2[2*j+1]+ aux_vector1[2*j+1] *
aux_vector2[2*j];                          }
}
-------------------------------------------------------------------------------------------------------------
/*Main  */
void main(void)
{
  rows = MATRIX_ROWS;    /* Define local variables  */
  columns = MATRIX_COLUMNS;
    CSL_init();   /* Initialize the CSL libraries  */
  for(i=0;i<columns;i++)   {
        aux_vector1[i]=ri_rows[i];   }
      FFT_TI();  /* Makes the FFT of the impulse response matrix by rows */
        for(i=0;i<columns;i++)   {
        aux_vector2[i]=aux_vector1[i];   }
      for(i=0;i<rows/2;i++) {
       for(j=0;j<columns;j++) {
         aux_vector1[j]=data[i][j];                          }
       FFT_TI();   /* Makes the FFT data matrix of the first  rows */
       PRODUCT();  /* Multiply the two matrix  */
```

```
      for (j=0;j<columns;j++) {
    data[i][j]=aux_vector3[j];                              }
}
    for(i=0;i<2*columns;i++)  {
            aux_vector3[i]=0;   }
    for(z=0;z<rows/2;z++)  {
    for(k=0;k<columns;k++)      {
         aux_vector3[k]=data[z][k];              }
    IFFT_TI();  /* Makes the IFFT of the resulting matrix of the first rows  */
    for (k=0;k<columns;k++)    {
    data[z][k]=aux_vector3[k];      }
}
for (i=0;i<0x10000;i++);
MCBSP_TRANSMITTER();  /* Transmit data  */
  for (i=0;i<rows/2;i++)  {
    for (j=columns/2;j<columns;j++) {
      outdata=data[i][j];
      while(!MCBSP_xrdy(hMcbsp1));
      MCBSP_write(hMcbsp1, outdata);                        }
                  }
  for (i=0; i<0x10000; i++);
  MCBSP_close(hMcbsp1);   /* close the port  */
MCBSP_RECIEVER();   /* Recieve data  */
for (i=rows/2;i<rows;i++) {
for (j=0;j<columns/2;j++) {
  while(!MCBSP_rrdy(hMcbsp1));
  indata = MCBSP_read(hMcbsp1);
  data[i][j]=indata;   }
}
  MCBSP_close(hMcbsp1);   /* close the port  */
```

```
  for(i=0;i<9;i++)  {
          for(j=0;j<columns/2;j++)   {  /* Conjugates the matrix  */
        aux_columns[i][2*j]=ri_columns[i][2*j];
       aux_columns[i][2*j+1]=ri_columns[i][2*j+1]*(-1);  }
}
   for(i=0;i<9;i++)  {
    for(j=0;j<columns;j++)  {
        aux_vector1[j]=aux_columns[i][j];  }
                FFT_TI();    /* Makes the FFT of the impulse response matrix by
columns  */
                for(j=0;j<columns;j++)  {
         ri_columns[i][j]=aux_vector1[j];            }
     }
           for(i=0;i<columns;i++)  {
          aux_vector2[i]=0;   }
  pe=(int)(rows/9);
    for(i=0;i<columns;i=i+2) {
       for(j=0;j<rows;j++)   {
       aux_vector1[2*j]=data[j][i];
       aux_vector1[2*j+1]=data[j][i+1];     }
     FFT_TI(); /* Makes the FFT of the first columns in the matrix  */
     if(i<2*pe) {   /* Select the appropiate column of the impulse response matrix to
         multiply  */
          x=0; }
     else if(i>2*pe-1 && i<4*pe) {
          x=1; }
          else if(i>4*pe-1 && i<6*pe) {
          x=2; }
          else if(i>6*pe-1 && i<8*pe) {
          x=3; }
```

```
            else if(i>8*pe-1 && i<10*pe) {
            x=4; }
             else if(i>10*pe-1 && i<12*pe) {
            x=5; }
            else if(i>12*pe-1 && i<14*pe) {
            x=6; }
            else if(i>14*pe-1 && i<16*pe) {
            x=7; }
            else if(i>16*pe-1) {
            x=8; }
             for(y=0;y<columns/2;y++)   {
          aux_vector2[2*y]=ri_columns[x][2*y];
         aux_vector2[2*y+1]=ri_columns[x][2*y+1]*(-1);  }
            PRODUCT();  /* Multiply the two matrix  */
       for(j=0;j<rows;j++)   {
         aux_matrix[j][i]=aux_vector3[2*j];
         aux_matrix[j][i+1]=aux_vector3[2*j+1];     }
}
     for(i=0;i<2*columns;i++)  {  /* Deletes the auxliary vector  for IFFT */
            aux_vector3[i]=0;   }
      for(k=0;k<columns;k=k+2)   {
        for(z=0;z<rows;z++)   {
      aux_vector3[2*z]=aux_matrix[z][k];
      aux_vector3[2*z+1]=aux_matrix[z][k+1];                }
       IFFT_TI();  /* Makes the IFFT of the first columns  in the matrix */
        for(z=0;z<rows;z++)   {
         data[z][k]=aux_vector3[2*z];
         data[z][k+1]=aux_vector3[2*z+1];     }
}
for (i=0;i<0x10000;i++);  /* Transmit data  */
```

```
MCBSP_TRANSMITTER();
  for (i=0;i<columns/2;i++)  {
    for (j=0;j<rows;j++) {
      outdata=data[j][i];
      while(!MCBSP_xrdy(hMcbsp1));
      MCBSP_write(hMcbsp1, outdata);                    }
                  }
  for (i=0; i<0x10000; i++);
  MCBSP_close(hMcbsp1);   /* close the port
}     /*------------------End of parallel_conv2D_6711.c--------------------*/
```

```
/*-------------------------------------------------------------------------------------------------------*/
/* parallel_conv2D_6713.c                                                                                */
/* This program makes the two-dimenstional  Fast Fourier  Transform in a                                 */
parallel way for the C6713 using the McBSP0 port to communicate with C6711    */
/*-------------------------------------------------------------------------------------------------------*/
/*  Include libraries  and files */
#include <csl.h>
#include <csl_mcbsp.h>
#include "math.h"
#include "dataW32.h"  /* w elements data file */
#include "data32.h"   /*data file  */
#include "ri_columns32.h"  /* impulse response matrix by columns  */
#include "ri_rows32.h"  /* impulse response matrix by rows  */
-------------------------------------------------------------------------------------------------------
/*Define constants */
#define DSK_MISC_REG (*(volatile unsigned char *)(0x90080006))
#define MATRIX_ROWS 32
#define MATRIX_COLUMNS 64
```

```
-------------------------------------------------------------------------------------------------
/*Define global variables */
far float aux_columns[MATRIX_ROWS][MATRIX_COLUMNS];
far float aux_matrix[MATRIX_ROWS][MATRIX_COLUMNS];
float aux_vector1[MATRIX_COLUMNS];
float aux_vector2[MATRIX_COLUMNS];
float aux_vector3[2*MATRIX_COLUMNS];
float indata, outdata;
unsigned int N, rows, columns, i, j, k, z, x, y, pe;
-------------------------------------------------------------------------------------------------
/*  Declare CSL objects  */
MCBSP_Handle hMcbsp0;  /* Hanles for McBSP  */
-------------------------------------------------------------------------------------------------
/* Declare external functions and functions prototypes */
extern MCBSP_Handle mcbsp_config(MCBSP_Handle);
void MCBSP_RECIEVER(void)   /* Reciever function  */
{
  hMcbsp0 = mcbsp_config(hMcbsp0);  /* Configure McBSP0 port */
  MCBSP_enableRcv(hMcbsp0);  /* Enable McBSP0 port as the reciever */
  return;
}
void MCBSP_TRANSMITTER(void)  /* Transmitter function  */
{
  hMcbsp0 = mcbsp_config(hMcbsp0); /* Configure McBSP0 port */
  MCBSP_enableSrgr(hMcbsp0);         /* Enable sample rate generator */
  for (i=0; i<0x10; i++);
  MCBSP_enableXmt(hMcbsp0); /*Enable McBSP0 port as the transmitter */
  MCBSP_enableFsync(hMcbsp0); /* Enable frame syncronization */
  return;
}
```

```
void FFT_TI(void)  /* Fast Fourier Transform function  */
{
  N= MATRIX_COLUMNS/2;
  DSPF_sp_cfftr2_dit((float*)aux_vector1, (float*)w,N);
  bit_rev((float*)aux_vector1,(2*N)>>1);
}
void IFFT_TI (void)   /* Inverse Fast Fourier Transform function */
{
  N = MATRIX_COLUMNS/2;
  In_DSPF_sp_cfftr2_dit((float*) aux_vector3,(float*) w, N);
  bit_rev((float *)aux_vector3, (2 * N)>>1);
  for(i=0;i<4*N;i++) {
    aux_vector3[i]= aux_vector3[i]/N;                    }
}
void PRODUCT (void)  /* Product function  */
{
  columns = MATRIX_COLUMNS;
  for(j=0;j<(columns-1);j++) {
  aux_vector3[2*j]=   aux_vector1[2*j]   *   aux_vector2[2*j]-   aux_vector1[2*j+1]   *
          aux_vector2[2*j+1];
  aux_vector3[2*j+1]= aux_vector1[2*j] * aux_vector2[2*j+1]+ aux_vector1[2*j+1] *
          aux_vector2[2*j];                          }
}
-------------------------------------------------------------------------------------------------------
/*Main  */
void main(void)
{
  rows = MATRIX_ROWS;   /* Define local variables */
  columns = MATRIX_COLUMNS;
    CSL_init();   /* Initialize CSL libraries  */
```

```
            DSK_MISC_REG |= 1;   /* Sets the appropiate bit in MISC register for  the
          use of McBSP0  port*/
  for(i=0;i<columns;i++)   {
        aux_vector1[i]=ri_rows[i];   }
       FFT_TI();  /* Makes the FFT of the impulse responser matrix by rows */
        for(i=0;i<columns;i++)   {
        aux_vector2[i]=aux_vector1[i];   }
       for(i=rows/2;i<rows;i++) {
        for(j=0;j<columns;j++) {
          aux_vector1[j]=data[i][j];                    }
        FFT_TI();  /* Makes the FFT of the data matrix of the last rows */
          PRODUCT();  /* Call to product function */
      for (j=0;j<columns;j++) {
    data[i][j]=aux_vector3[j];                          }
}
    for(i=0;i<2*columns;i++)  {
           aux_vector3[i]=0;   }  /*Deletes the auxiliary vector for IFFT */
    for(z=rows/2;z<rows;z++)    {
    for(k=0;k<columns;k++)      {
         aux_vector3[k]=data[z][k];             }
    IFFT_TI();/* Makes the IFFT of the matrix of the last rows */
    for (k=0;k<columns;k++)    {
    data[z][k]=aux_vector3[k];      }
}
MCBSP_RECIEVER();  /* Recieve data */
for (i=0;i<rows/2;i++) {
for (j=columns/2;j<columns;j++)   {
  while(!MCBSP_rrdy(hMcbsp0));
  indata = MCBSP_read(hMcbsp0);
  data[i][j]=indata;   }
```

```
}
  MCBSP_close(hMcbsp0);   /* close the port */
  for (i=0;i<0x100000;i++);  /*Transmit data */
 MCBSP_TRANSMITTER();
  for (i=rows/2;i<rows;i++)  {
    for (j=0;j<columns/2;j++) {
      outdata=data[i][j];
      while(!MCBSP_xrdy(hMcbsp0));
      MCBSP_write(hMcbsp0, outdata);                    }
                  }
  for (i=0; i<0x10000; i++);
  MCBSP_close(hMcbsp0);   /* close the port  */
    for(i=0;i<9;i++)  {   /*Conjugate the impulse response matrix by columns */
            for(j=0;j<columns/2;j++)   {
         aux_columns[i][2*j]=ri_columns[i][2*j];
        aux_columns[i][2*j+1]=ri_columns[i][2*j+1]*(-1);  }
}
    for(i=0;i<9;i++)  {
     for(j=0;j<columns;j++)  {
         aux_vector1[j]=aux_columns[i][j];  }
                  FFT_TI();   /* Makes the FFT of the impulse responser matrix by
          columns*/
                  for(j=0;j<columns;j++)  {
          ri_columns[i][j]=aux_vector1[j];             }
      }
             for(i=0;i<columns;i++)  {  /*Deletes the auxiliary vector  */
           aux_vector2[i]=0;   }
   pe=(int)(rows/9);
     for(i=columns/2;i<columns;i=i+2)   {
        for(j=0;j<rows;j++)   {
```

```
      aux_vector1[2*j]=data[j][i];
      aux_vector1[2*j+1]=data[j][i+1];      }
    FFT_TI();/* Makes the FFT of matrix of the last columns */
    if(i<2*pe) {  /* Select the appropiate row of the impulse response matrix for the
        product  function*/
          x=0; }
    else if(i>2*pe-1 && i<4*pe) {
          x=1; }
          else if(i>4*pe-1 && i<6*pe) {
          x=2; }
          else if(i>6*pe-1 && i<8*pe) {
          x=3; }
          else if(i>8*pe-1 && i<10*pe) {
          x=4; }
           else if(i>10*pe-1 && i<12*pe) {
          x=5; }
          else if(i>12*pe-1 && i<14*pe) {
          x=6; }
          else if(i>14*pe-1 && i<16*pe) {
          x=7; }
          else if(i>16*pe-1) {
          x=8; }
           for(y=0;y<columns/2;y++)     {   /* Conjugate the matrix */
       aux_vector2[2*y]=ri_columns[x][2*y];
      aux_vector2[2*y+1]=ri_columns[x][2*y+1]*(-1);   }
          PRODUCT();  /* Call to product function  */
     for(j=0;j<rows;j++)   {
      aux_matrix[j][i]=aux_vector3[2*j];
      aux_matrix[j][i+1]=aux_vector3[2*j+1];     }
}
```

```
    for(i=0;i<2*columns;i++)  {  /*Deletes the auxiliary vector for IFFT */
            aux_vector3[i]=0;   }
   for(k=columns/2;k<columns;k=k+2)   {
        for(z=0;z<rows;z++)   {
          aux_vector3[2*z]=aux_matrix[z][k];
     aux_vector3[2*z+1]=aux_matrix[z][k+1];                }
          IFFT_TI();  /* Makes the IFFT of the  matrix of the last columns */
       for(z=0;z<rows;z++)   {
        data[z][k]=aux_vector3[2*z];
        data[z][k+1]=aux_vector3[2*z+1];     }
}
MCBSP_RECIEVER();  /*Recieve data  */
for (i=0;i<columns/2;i++) {
for (j=0;j<rows;j++) {
  while(!MCBSP_rrdy(hMcbsp0));
indata = MCBSP_read(hMcbsp0);
  data[j][i]=indata;   }
}
  MCBSP_close(hMcbsp0);   /*close the port  */
 }
/*------------------End of parallel_conv2D_6713.c--------------------*/
```

Anexo E. ALGORITMO DE LA CONFIGURACIÓN DE LOS PUERTOS McBSP PARA LA C6711 Y LA C6713.

```
/*-------------------------------------------------------------------------------------------------------*/
/* mcbsp_config.c                                                                                        */
/* This program configure the McBSP for data exchange.                                                   */
/*-------------------------------------------------------------------------------------------------------*/
/* Include files */
#include <csl.h>
#include <csl_mcbsp.h>
-------------------------------------------------------------------------------------------------------
/* McBSP port registers configuration  */
MCBSP_Handle mcbsp_config(MCBSP_Handle hMcbsp)
{
MCBSP_Config mcbspCfg = {
MCBSP_SPCR_RMK(
MCBSP_SPCR_FREE_NO,
MCBSP_SPCR_SOFT_NO,
MCBSP_SPCR_FRST_YES,
MCBSP_SPCR_GRST_YES,
MCBSP_SPCR_XINTM_XRDY,
MCBSP_SPCR_XSYNCERR_NO,
MCBSP_SPCR_XRST_NO,
MCBSP_SPCR_DLB_OFF,
MCBSP_SPCR_RJUST_RSE,
MCBSP_SPCR_CLKSTP_DISABLE,
MCBSP_SPCR_DXENA_OFF,
MCBSP_SPCR_RINTM_RRDY,
```

```
MCBSP_SPCR_RSYNCERR_NO,
MCBSP_SPCR_RRST_YES
),
MCBSP_RCR_RMK(
MCBSP_RCR_RPHASE_SINGLE,
MCBSP_RCR_RFRLEN2_OF(0),
MCBSP_RCR_RWDLEN2_8BIT,
MCBSP_RCR_RCOMPAND_MSB,
MCBSP_RCR_RFIG_NO,
MCBSP_RCR_RDATDLY_1BIT,
MCBSP_RCR_RFRLEN1_OF(0),
MCBSP_RCR_RWDLEN1_32BIT,
MCBSP_RCR_RWDREVRS_DISABLE
),
MCBSP_XCR_RMK(
MCBSP_XCR_XPHASE_SINGLE,
MCBSP_XCR_XFRLEN2_OF(0),
MCBSP_XCR_XWDLEN2_8BIT,
MCBSP_XCR_XCOMPAND_MSB,
MCBSP_XCR_XFIG_NO,
MCBSP_XCR_XDATDLY_1BIT,
MCBSP_XCR_XFRLEN1_OF(0),
MCBSP_XCR_XWDLEN1_32BIT,
MCBSP_XCR_XWDREVRS_DISABLE
),
MCBSP_SRGR_RMK(
MCBSP_SRGR_GSYNC_FREE,
MCBSP_SRGR_CLKSP_RISING,
MCBSP_SRGR_CLKSM_INTERNAL,
MCBSP_SRGR_FSGM_DXR2XSR,
```

```
    MCBSP_SRGR_FPER_DEFAULT,
    MCBSP_SRGR_FWID_DEFAULT,
    MCBSP_SRGR_CLKGDV_OF(30)
    ),
    MCBSP_MCR_DEFAULT,
    MCBSP_RCER_DEFAULT,
    MCBSP_XCER_DEFAULT,
    MCBSP_PCR_RMK(
    MCBSP_PCR_XIOEN_SP,
    MCBSP_PCR_RIOEN_SP,
    MCBSP_PCR_FSXM_INTERNAL,
    MCBSP_PCR_FSRM_EXTERNAL,
    MCBSP_PCR_CLKXM_OUTPUT,
    MCBSP_PCR_CLKRM_INPUT,
    MCBSP_PCR_CLKSSTAT_DEFAULT,
    MCBSP_PCR_DXSTAT_DEFAULT,
    MCBSP_PCR_FSXP_ACTIVEHIGH,
    MCBSP_PCR_FSRP_ACTIVEHIGH,
    MCBSP_PCR_CLKXP_RISING,
    MCBSP_PCR_CLKRP_FALLING
    )
    };
    hMcbsp = MCBSP_open(MCBSP_DEV1, MCBSP_OPEN_RESET);
    /*  MCBSP_DEV0 for the  C6713 y MCBSP_DEV1 for the  C6711   /*
    MCBSP_config(hMcbsp, &mcbspCfg);
return hMcbsp;
}
/*------------------------------End of mcbsp_config.c-------------------------------------------*/
```

Anexo F. ALGORITMO DE TI PARA LA FUNCIÓN DE LA FFT 1-D.

```
/*----------------------------------------------------------------------------------------------------*/
/*DSPF_sp_cfftr2_dit.c                                                                                */
/*----------------------------------------------------------------------------------------------------*/
void DSPF_sp_cfftr2_dit(float* x, float* w, short n)
{
short n2, ie, ia, i, j, k, m;
float rtemp, itemp, c, s;
n2 = n;
ie = 1;
  for(k=n; k > 1; k >>= 1)  {
  n2 >>= 1;
  ia = 0;
  for(j=0; j < ie; j++)  {
    c = w[2*j];
    s = w[2*j+1];
    for(i=0; i < n2; i++)  {
      m = ia + n2;
      rtemp     = c * x[2*m]   + s * x[2*m+1];
      itemp     = c * x[2*m+1] - s * x[2*m];
      x[2*m]    = x[2*ia]   - rtemp;
      x[2*m+1]  = x[2*ia+1] - itemp;
      x[2*ia]   = x[2*ia]   + rtemp;
      x[2*ia+1] = x[2*ia+1] + itemp;
       ia++;                          }
     ia += n2;                       }
   ie <<= 1;            }   }
/*----------------------------------End of DSPF_sp_cfftr2_dit.c ----------------------------------*/
```

Anexo G. ALGORITMO DE TI PARA LA FUNCIÓN DE LA IFFT 1-D.

```
/*----------------------------------------------------------------------------------------------------*/
/*In_DSPF_sp_cfftr2_dit.c                                                                             */
/*----------------------------------------------------------------------------------------------------*/
void In_DSPF_sp_cfftr2_dit(float* x, float* w, short n)
{
short n2, ie, ia, i, j, k, m;
float rtemp, itemp, c, s ;
n2 = n;
ie = 1;
for(k=n; k > 1; k >>= 1)  {
  n2 >>= 1;
  ia = 0;
  for(j=0; j < ie; j++)  {
    c = w[2*j];
    s = -1*w[2*j+1];
    for(i=0; i < n2; i++)  {
      m = ia + n2;
      rtemp = c* x[2*m] + s * x[2*m+1];
      itemp = c* x[2*m+1] - s * x[2*m];
      x[2*m] = (x[2*ia] - rtemp);
      x[2*m+1] = (x[2*ia+1] - itemp);
      x[2*ia] = (x[2*ia] + rtemp);
      x[2*ia+1] = (x[2*ia+1] + itemp);
       ia++;                  }
     ia += n2;                  }
     ie <<= 1;                }
  }  /*----------------------End of In_DSPF_sp_cfftr2_dit.c    ----------------------------------*/
```

Anexo H. ALGORITMO DE TI PARA LA FUNCIÓN DEL ORDENAMIENTO DE BITS.

```
/*-------------------------------------------------------------------------------------------------------*/
/* bit_rev.c                                                                                             */
/*-------------------------------------------------------------------------------------------------------*/
void bit_rev(float* x, int n)
{
int i, j, k;
double rtemp, itemp;
j = 0;
for(i=1; i < (n-1); i++)  {
  k = n >> 1;
    while(k <= j)  {
    j -= k;
    k >>= 1;
                        }
  j += k;
  if(i < j)  {
    rtemp = x[j*2];
    x[j*2] = x[i*2];
    x[i*2] = rtemp;
    itemp = x[j*2+1];
    x[j*2+1] = x[i*2+1];
    x[i*2+1] = itemp;
              }
                                }
 }
/*-----------------------------------------End of bit_rev.c-----------------------------------------*/
```