IMPLEMENTACION DEL ALGORITMO DE LA TRANSFORMADA DISCRETA DE FOURIER EMPLEANDO EL DSP TMS320C6201

Ing. Rómulo Miguel Ato
d270020@unmsm.edu.pe

Facultad de Ingeniería Electrónica de la Universidad Nacional Mayor de San Marcos de Lima - Perú

Resumen : En el presente estudio se implementa la Transformada Discreta de Fourier (DFT) a través de la tarjeta DSP TMS320C6201, una PC y su correspondiente software de aplicación. El análisis se realiza calculando la magnitud y los componentes armónicos de la señal eléctrica. Los resultados se describen mediante prueba y simulación del algoritmo desarrollado.

Abstract : In the present work is implemented the Transformed Discreet of Fourier (DFT) through the DSP TMS320C6201 card, a PC and its corresponding application software. The analysis is carried out calculating the magnitude and the harmonic components of the electric sign. The results are described by means of test and simulation of the developed algorithm.

Palabras Claves : Procesamiento Digital de Señales (DSP), Transformada Discreta de Fourier (DFT), factor twiddle, operación split

I. INTRODUCCION

En la última década el empleo de los procesadores para ejecutar algoritmos numéricos en tiempo real a crecido, y sus campos de aplicación son: las telecomunicaciones, procesamiento de imágenes y gráficos, control numérico de alta velocidad, procesamiento de voz, instrumentación y cálculo numérico. La tecnología de DSP hace más fácil el desarrollo de poderosos sistemas computacionales, basados en adquisición y análisis de datos.

El requerimiento para que los algoritmos en DSP se realicen en tiempo real, implica que el factor velocidad de procesamiento sea critico. Otro factor limitante es el enorme volumen de cálculo numérico que requieren estas aplicaciones.

II. OBJETIVOS

El objetivo principal es analizar la característica de una señal eléctrica que consiste en la medición de la amplitud y armónicos empleando un computador personal, una tarjeta DSP y un software de aplicación. Para lo cual se ha previsto cubrir las siguientes etapas:

- Instalación y prueba del kit de desarrollo TMS320C6201
- Estudio de la arquitectura y juego de instrucciones del TMS320C6201
- Estudio e investigación de algoritmos de procesamiento de señales (DFT)
- Implementación del algoritmo DFT
- Prueba del algoritmo DFT.

III. DESCRIPCION

3.1 Algoritmo de la Transformada de Fourier de secuencias de valores reales con el DSP TMS320C6201

La transformada rápida de Fourier (fft) es un cálculo eficiente de la transformada discreta de Fourier (dft) y una de las más importantes herramientas usadas en aplicaciones de procesamiento digital de señales.

El desarrollo del algoritmo de la DFT asume una secuencia de entradas que consiste de números complejos. Esto es debido a que factores complejos (factores *twiddle)*, resultan ser variables complejas; de esta manera los algoritmos DFT están diseñados para realizar multiplicaciones y adiciones complejas. Sin embargo la secuencia de entrada generalmente consiste de números reales en la mayoría de aplicaciones reales.

Este proyecto describe la teoría e implementación de un algoritmo para calcular eficientemente la DFT de valores reales muestreados. Esta implementación se realiza en el TMS320C6X.

El algoritmo realiza la DFT de dos secuencias de N puntos de valores reales que usan una DFT compleja de N puntos y realiza algunos cálculos adicionales.

3.1.1 Análisis de la DFT y FFT

Los métodos de realizar la DFT de secuencias reales que implican DFTs de valores complejos. Esta sección revisa la base de la DFT y FFT.

La DFT se visualiza como una representación en el dominio de la frecuencia de la secuencia en el tiempo discreto x(n). La DFT de N puntos de una secuencia de duración finita x(n) está definida como:

$$X(k) = \sum_{n=0}^{N-1} x(n)\ W_N^{kn} \qquad k = 0,1,\ldots N\text{-}1 \qquad (1)$$

y la inversa de la DFT (IDFT) está definida como

$$x(n) = \sum_{K=0}^{N-1} X(n)\ W_N^{-kn} \qquad n = 0,1,\ldots,N\text{-}1 \qquad (2)$$

donde:

$$W_N^{kn} = e^{-2\pi nk/N}$$

el factor W es también referido como el factor *twiddle*.

La observación de las ecuaciones arriba descritas, muestran que los requerimientos de cálculo de la DFT se incrementa rápidamente cuando el número de muestras en la secuencia N se incrementa. Debido al gran número de requerimientos computacionales, la implementación directa de la DFT de secuencias grandes no ha sido práctica para aplicaciones en tiempo real, sin embargo el desarrollo de algoritmos rápidos conocidos como FFTs ha hecho una implementación práctica de la DFT en aplicaciones en tiempo real.

La definición de la FFT es la misma que la DFT pero el método de cálculo difiere. Los algoritmos básicos de la FFT implican una aproximación de división y arreglos, en el cual una DFT de N puntos es dividida sucesivamente dentro de DFTs muy pequeñas. Muchos algoritmos han sido desarrollados tales como radix-2, radix-4, y mixed radix; in-place y no-in-place; decimación en tiempo y decimación en frecuencia.

En la mayoría de algoritmos FFT, las restricciones pueden aplicarse, por ejemplo una FFT radix-2 restringe el número de muestras en la secuencia a potencia de dos. Además, algunos algoritmos FFT requieren que la entrada o salida sean reordenados; por ejemplo el algoritmo radix-2 decimación en frecuencia requiere que la salida sea en bit inverso.

La Tabla 1 compara el número de cálculos matemáticos que implican en el cálculo directo de la DFT versus el algoritmo radix-2 FFT. Como se puede ver, la mejora de la velocidad de la FFT se incrementa cuando N se incrementa.

Las siguientes secciones describen métodos de cálculo eficiente de la DFT de secuencias de valores reales usando la DFT de valores complejos.

Tabla 1. Comparación entre el cálculo directo de la DFT y el *algoritmo FFT Radix-2.*

Número de Puntos	***Cálculo Directo de la DFT***		***FFT Radix 2***	
	Multiplicaciones complejas	*Adiciones complejas*	*Multiplicaciones complejas*	*Adiciones complejas*
N	N^2	N^2-N	$(N/2)\log_2 N$	$(N/2)\log_2 N$
4	*16*	*12*	*4*	*8*
16	*256*	*240*	*32*	*64*
64	*4096*	*4032*	*192*	*384*
256	*65536*	*65280*	*1024*	*2048*
1024	*1048576*	*1047552*	*5120*	*10240*

3.2 DFT de secuencias reales

En muchas aplicaciones, las secuencias de datos que van a ser procesadas son valores reales. Aún cuando el dato es real, los algoritmos DFT de valores complejos se pueden usar. Una simple aproximación crea una secuencia compleja de datos reales. La DFT de valor complejo puede ser aplicada directamente, sin embargo este método no es eficiente.

3.2.1. Cálculo eficiente de la DFT de 2 secuencias reales

Suponemos que $x_1(n)$ y $x_2(n)$ son secuencias de valores reales de longitud N, y x(n) es una secuencia de valor

complejo definida como:

$$x(n) = x_1(n) + + jx_2(n) \qquad 0 \le n \le N-1 \tag{3}$$

La DFT de las dos secuencias de longitud N, $x_1(n)$ y $x_2(n)$ puede ser hallado realizando una sola DFT de longitud N sobre la secuencia de valor complejo y algunos cálculos adicionales, los cuales son referidos como la operación *split* y se muestran a continuación:

$$X_1(k) = \frac{1}{2}[X(k) + X^*(N-k)] \qquad \text{para } k = 0,1,\ldots N\text{-}1 \tag{4}$$

$$X_2(k) = \frac{1}{2j}[X(k) - X^*(N-k)]$$

Como se puede ver de las ecuaciones arriba mencionadas, la transformada de $x_1(n)$ y $x_2(n)$, $X_1(k)$ y $X_2(k)$ respectivamente, son resueltas calculando una DFT de valor complejo, X(k), y algunos cálculos adicionales.

Ahora asumimos que deseamos volver a tener $x_1(n)$ y $x_2(n)$ a partir de $X_1(k)$ y $X_2(k)$, respectivamente. Así como con la DFT directa, la IDFT de $X_1(k)$ y $X_2(k)$ se halla usando una DFT de valor complejo, debido a que la operación DFT es lineal, la DFT de la ecuación (3) puede ser expresada como:

$$X(k) = X_1(k) + + jX_2(k) \tag{5}$$

Esto muestra que X(k) puede ser expresado en términos de $X_1(k)$ y $X_2(k)$; así tomando la inversa de DFT de X(k), obtenemos x(n), el cual nos da $x_1(n)$ y $x_2(n)$.

Las ecuaciones de arriba requieren aritmética compleja no directamente soportadas por DSPs; así para implementar estas ecuaciones de valores complejos, es útil expresar los términos reales e imaginarios en aritmética real.

La DFT directa de las ecuaciones mostradas en (4) pueden ser escritas como sigue:

$$X_1r(k) = \frac{1}{2}[Xr(k) + Xr(N-k)] \quad \text{y} \quad X_1i(k) = \frac{1}{2}[Xi(k) - Xi(N-k)]$$

para $k = 0,1,\ldots N\text{-}1$ (6)

$$X_2r(k) = \frac{1}{2}[Xi(k) + Xi(N-k)] \quad \text{y} \quad X_2i(k) = \frac{-1}{2}[Xr(k) - Xr(N-k)]$$

Además debido a que la DFT de secuencias de valores reales tiene las propiedades de simetría y periodicidad de conjugada compleja, los números de cálculos en (6) pueden ser reducidos.

Usando las propiedades de las ecuaciones en (6) pueden ser reescritas como sigue:

$$X_1r(0) = Xr(0) \qquad X_1i(0) = 0$$
$$X_2r(0) = Xi(0) \qquad X_2i(0) = 0$$

$$X_1r(N/2) = Xr(N/2) \qquad X_1i(N/2) = 0$$
$$X_2r(N/2) = Xi(N/2) \qquad X_2i(N/2) = 0 \tag{7}$$

$$X_1r(k) = \frac{1}{2}[Xr(k) + Xr(N-k)] \qquad X_1i(k) = \frac{1}{2}[Xi(k) - Xi(N-k)]$$

$$X_2r(k) = \frac{1}{2}[Xi(k) + Xi(N-k)] \qquad X_2i(k) = \frac{-1}{2}[Xr(k) - Xr(N-k)]$$

$$X_1r(N-k) = X_1r(k) \qquad X_1i(N-k) = -X_1i(k)$$

$$X_2r(N-k) = X_2r(k) \qquad X_2i(N-k) = -X_2i(k)$$

para k = 0,1,...N/2-1

Similarmente, los cálculos adicionales que involucran en el cálculo de la IDFT pueden ser escritos de la siguiente forma:

$$Xr(k) = X_1r(k) - X_2i(k) \qquad \text{para k = 0,1,...N/2-1} \tag{8}$$
$$Xi(k) = X_1i(k) - X_2r(k)$$

ver Apéndice B para una detallada derivación de estas ecuaciones.

Ahora que tenemos las ecuaciones para la operación *split* usadas en el cálculo de la DFT de dos secuencias de valores reales seguimos los siguientes pasos que señalan como usar las ecuaciones. La DFT directa es indicada:

- Los valores complejos de N puntos x(n) de las dos secuencias de longitud N por $x_1(n)$ y $x_2(n)$.

- **Para n = 0,1,...N-1**

$$x_r(n) = x_1(n)$$
$$x_i(n) = x_2(n)$$

Calcula la DFT compleja de longitud N de x(n).

X(k) = DFT(x(n))

La DFT puede ser cualquier algoritmo eficiente (tal como uno de los varios algoritmos FFT) pero la salida debe estar en orden normal.

- Calcula las ecuaciones de operación *split*.

$$X_1r(0) = Xr(0) \qquad X_1i(0) = 0$$
$$X_2r(0) = Xi(0) \qquad X_2i(0) = 0$$
$$X_1r(N/2) = Xr(N/2) \qquad X_1i(N/2) = 0$$
$$X_2r(N/2) = Xi(N/2) \qquad X_2i(N/2) = 0$$

Para k=0,1,....N/2-1;

$$X_1r(k) = 0.5[Xr(k) + Xr(N-k)] \qquad X_1i(k) = 0.5[Xi(k) - Xi(N-k)]$$

$$X_1r(k) = 0.5[Xi(k) + Xi(N-k)] \qquad X_1i(k) = -0.5[Xr(k) - Xr(N-k)]$$

$$X_1r(N-k) = X_1r(k) \qquad X_1i(N-k) = X_1i(k)$$

$$X_2r(N-k) = X_2r(k) \qquad X_2i(N-k) = -X_2i(k)$$

Para dos secuencias en el dominio de la frecuencia $X_1(k)$ y $X_2(k)$, derivado desde las secuencias de valores reales, realizar los siguientes pasos para tomar la IDFT de $X_1(k)$ y $X_2(k)$:

- Forma la secuencia de valor complejo X(k) desde $X_1(k)$ y $X_2(k)$ usando las ecuaciones *split* IDFT.

Para k = 0, ..., N-1

$Xr(k) = X_1r(k) - X_2i(k)$
$Xi(k) = X_1i(k) + X_2r(k)$

- Calcula la IDFT de longitud N de X(k).

- x(n) = IDFT[X(k)]

- Como con la DFT directa, la IDFT puede ser cualquier algoritmo eficiente. La IDFT puede ser calculada usando la DFT directa y algunas operaciones conjugadas.

$$x(n) = [DFT\{X^*(k)\}]^*$$

donde, * es el operador conjugado complejo.

- Desde x(n), forma $x_1(n)$ y x_2(n).
  Para n = 0, 1, ... N-1

$x_1(n) = xr(n)$
$x_2(n) = xi(n)$

3.3 Implementación de la DFT de valores reales

El apéndice A contiene el listado del código fuente de la implementación en lenguaje C para realizar la DFT de secuencias de valores reales. El propósito principal de esta implementación particular es verificar la funcionalidad de las operaciones *split* y proporcionar un modelo conocido para comparar con versiones optimizadas. Otro beneficio es que esta implementación está en código genérico en lenguaje C y así puede portar a otros DSPs o CPUs.

El programa se compila de la siguiente manera:

Cl6x –g vectors.asm realdft2.c split2.c data2.c dft.c –z –o test2.out –l rts6201.lib lnk.cmd

El ejemplo usa el archivo ejecutable que puede ser cargado en el C62xx y correr:

Test2.out

La opción -g es usado para decir al compilador que fabrique el código con información depurada. Esto significa que el compilador no usa el optimizador pero permite al código fácilmente visualizar.

IV. CONCLUSIONES

Se ha realizado una introducción al algoritmo de la Transformada de Fourier, partiendo con el algoritmo DFT el cual nos permite calcular los espectros de señales cuyas muestras se guardan en una posición de memoria, así estos resultados nos permiten conocer la naturaleza de una señal cualquiera.

Se demuestra que el empleo de técnicas de procesamiento de señales es importante ya que mediante ellas se puede suplir el uso de técnicas convencionales como el análisis espectral analógico, además las técnicas de procesamiento digital nos permiten una mayor exactitud en los resultados.

A partir de estos resultados obtenidos en la implementación de la transformada discreta de Fourier se puede implementar el algoritmo de la FFT que viene a ser la Transformada Discreta de Fourier con menor numero de operaciones. Posteriormente se podrán realizar otros tipos de algoritmos tales como filtros digitales.

La idea fundamental de este proyecto es introducirse en el campo del procesamiento de señales y avanzar progresivamente con el entendimiento de los diversos algoritmos DSP y en la implementación de estos. Un siguiente paso a este avance será investigar e implementar filtros digitales.

V. REFERENCIAS BIBLIOGRÁFICAS

Texas Instruments, *"TMS320C62X Optimizing C Compiler User's Guide"*, 1999
Texas Instruments, *"TMS320C62X C source DebuggerUser's Guide"*, 1999
Texas Instruments, *"TMS320C62X Programer's Guide"*, 1999
Texas Instruments, *"TMS320C62X/CPU and Instruction Set Reference Guide"*, 1999
Texas Instruments, *"TMS320C62X Peripheral Support Library"*, 1999
Texas Instruments, *"TMS320C62X/6701 Peripherals Reference Guide"*, 1999
La FFT: Aplicación en el DSP56000/1: J. Gonzales de Mendívil, J. Garitagoitia.
Mundo Electrónico Abril de 1992.
The Fast Fourier Transform. E. Oran Brigham
Mixed Signal Processing Design Seminar: Analogo Devices

Apéndice A

```
implementación del Algoritmo de la Transformada Discreta de Fourier de dos secuencias de valores reales

/********************************** realdft2.c ***************************************
Codigo fuente C de  la implementacion de la DFT de dos secuencias reales de N puntos, usando una DFT
compleja de N puntos.
*************************************************************************************
 Este programa es un ejemplo de la implementacion de la DFT de dos secuencias de valores reales. Asumimos
que tenemos dos secuencias de valores reales de longitud N - x1[n] y x2[n].
La DFT de x1[n] y x2[n] pueden ser calculados con una DFT de valor complejo de longitud N, como
mostramos anteriormente:

1. Forma la secuencia de valor complejo x[n] desde x1[n] y x2[n]
xr[n] = x1[n] y xi[n] = x2[n],                                   0,1, ..., N-1

2. Calcule X[k] = DFT{x[n]}

3. Calcule las  siguientes ecuaciones para obtener la DFTs de x1[n] y x2[n].
X1r[0] = Xr[0]
X1i[0] = 0
X2r[0] = Xi[0]
X2i[0] = 0
X1r[N/2] = Xr[N/2]
X1i[N/2] = 0
X2r[N/2] = Xi[N/2]
X2i[N/2] = 0
Para k = 1,2,3, ...., N/2-1
X1r[k] = (Xr[k] + Xr[N-k])/2
X1i[k] = (Xi[k] – Xi[N-k])/2
X1r[N-k] = X1r[k]
X1i[N-k] = X1i[k]
X2r[k] = (Xi[k] + Xi[N-k])/2
X2i[k] = (Xr[N-k] - Xr[k])/2
X2r[N-k] = X2r[k]
X2i[N-k] = X2i[k]
4. Formar X[k] desde X1[k] y X2[k]
Para k = 0,1, ..., N-1
Xr[k] = X1r[k] - X2i[k]
Xi[k] = X1i[k] + X2r[k]

*************************************************************************************/
#include <math.h>                   /* incluye la librería math C RTS */
#include "params2.h"                /* incluye archivo con parametros */
#include "params.h"                 /* include archivo con parametros */

extern short x1[ ];
extern short x2[ ];
 void dft(int, COMPLEX *);
extern void split2(int, COMPLEX *, COMPLEX *, COMPLEX *);
```

Figura 1. realdft2.c

```
main()
{
int n, k;
COMPLEX X1[NUMDATA];   /*arreglo de valores reales de la secuencia de salida DFT,  X1(k)*/
COMPLEX X2[NUMDATA];       /*arreglo de valores reales de la secuencia de salida DFT, X2(k) */
COMPLEX x[NUMPOINTS+1];   /*arreglo of complex DFT data, X(k)*/

/********************************** DFT ************************************/
/* Desde las dos secuencias reales de N-puntos, x1(n) y x2(n), forma la secuencia compleja de
  N puntos, x(n) = x1(n) + jx2(n) */
for (n=0; n<NUMDATA; n++)
{
x[n].real   =  x1[n];
x[n].imag  =  x2[n];
}
/*Calcule la DFT de x(n), X(k) = DFT{x(n)}. Note, la DFT puede ser cualquier implementacion DFT,
tal como FFTs. */
dft(NUMPOINTS, x);
/* Debido de la propiedad de periodicidad de la DFT, conocemos que X(N+k)=X(k). */
x[NUMPOINTS].real = x[0].real;
x[NUMPOINTS].imag = x[0].imag;
/* The function split realiza los calculos adicionales requeridos para dar X1(k) y X2(k) desde X(k). */
split2(NUMPOINTS, x, X1, X2);

return(0);
}
```

Figura 2. realdft2.c (continuación)

```
/************************************ split2.c ********************************
Este es el codigo C  para la implementacion de la rutina split  el cual es un calculo adicional en el desarrollo
de la  DFT de dos secuencias de valores reales de N puntos usando una DFT compleja de N puntos.
*******************************************************************************
El calculo de la DFT de dos secuencias de valor real de N puntos puede ser eficientemente calculado
usando uno DFT compleja de N puntos y algunos cálculos adicionales. Esta función implementa esos
calculos adicionales, los cuales son mostrados abajo.
X1r[0] = Xr[0]
X1i[0] = 0
X2r[0] = Xi[0]
X2i[0] = 0
X1r[N/2] = Xr[N/2]
X1i[N/2] = 0
X2r[N/2] = Xi[N/2]
X2i[N/2] = 0
for k = 1,2,3, ...., N/2-1
X1r[k] = (Xr[k] + Xr[N-k])/2
X1i[k] = (Xi[k] - Xi[N-k])/2
X1r[N-k] = X1r[k]
X1i[N-k] = X1i[k]
X2r[k] = (Xi[k] + Xi[N-k])/2
X2i[k] = (Xr[N-k] - Xr[k])/2
X2r[N-k] = X2r[k]
X2i[N-k] = X2i[k]
*******************************************************************************/
#include "params.h"
void split2(int N, COMPLEX *X, COMPLEX *X1, COMPLEX *X2)
{
int k;
X1[0].real = X[0].real;
X1[0].imag = 0;
X2[0].real = X[0].imag;
X2[0].imag = 0;
X1[N/2].real = X[N/2].real;
X1[N/2].imag = 0;
X2[N/2].real = X[N/2].imag;
X2[N/2].imag = 0;
for (k=1; k<N/2; k++)
{
X1[k].real = (X[k].real + X[N-k].real)/2;
X1[k].imag = (X[k].imag - X[N-k].imag)/2;
X2[k].real = (X[k].imag + X[N-k].imag)/2;
X2[k].imag = (X[N-k].real - X[k].real)/2;
X1[N-k].real = X1[k].real;
X1[N-k].imag = -X1[k].imag;
X2[N-k].real = X2[k].real;
X2[N-k].imag = -X2[k].imag;
}
}
```

Figura 3. split2.c

```
/************************ data2.c ************************************************
muestra de datos usados en  realdft2.c
*********************************************************************************/
arreglo de secuencias de entrada de valores reales, x1(n)  */
short x1[ ] = {255, 255, 255, 255, 255, 255, 255, 255, 0, 0, 0, 0, 0, 0, 0, 0};

/*arreglo de secuencias de entrada de valores reales, x2(n) */
short x2[ ] = {-35, -35, -35, -35, -35, -35, -35, -35,0, 0, 0, 0, 0, 0, 0, 0};
```

Figura 4. data2.c

```
/************************************ params2.h ******************************
este es el archivo de cabecera C  para el ejemplo dela implementacion de la FFT real.
*****************************************************************************/
#define NUMDATA 16                      /* numero de muestras de datos reales */
#define NUMPOINTS NUMDATA                       /* numero de puntos en la DFT */
```

Figura 5. params2.h

```
/************************************ params.h ******************************
Esto es el archivo de cabecera C  para implementacion de la FFT real
*****************************************************************************/
#define TRUE 1
#define FALSE 0
#define BE TRUE
#define LE FALSE
#define ENDIAN LE                       /* selecciona apropiado endianess. Si se fabrica codigo en
   Big Endian, use BE, sino use LE */
#define PI 3.141592653589793      /* definicion de pi */
/* BIG Endian */
#if ENDIAN == TRUE
typedef struct {
short imag;
short real;
} COMPLEX;
#else
/* LITTLE Endian */
typedef struct {
short real;
short imag;
} COMPLEX;
#endif
```

Figura 6. Params.h

```
/************************************** dft.c ********************************
Esta funcion calcula  la DFT de una secuencia de valores complejos de longitud N.
 X(k) = SUM x(n) * exp(-j2pikn/N) k = 0,1,2, ..., N-1
N=0
Esto es siempre util para expresar la ecuacion de arriba en sus terminos  real e imaginaria.
Exp(-j2*pi*n*k/N) = cos(2*pi*n*k/N) - jsin(2*pi*n*k/N) -> diversas identidades usadas aqui
E(jb) = cos(b) + j sin(b)
e(-jb) = cos(-b) + j sin(-b)
cos(-b) = cos(b) and sin(-b) = -sin(b)
e(-jb) = cos(b) - j sin(b)
N-1
X(k) = SUM {[xr(n) + j xi(n)][cos(2*pi*n*k/N) - jsin(2*pi*n*k/N)]}
N=0
K=0,1,2, ... ,N-1
OR
N-1
Xr(k) = SUM {[xr(n) * cos(2*pi*n*k/N)] + [xi(n) * sin(2*pi*n*k/N)]}
N=0
K=0,1,2, ... ,N-1
N-1
Xi(k) = SUM {[xi(n) * cos(2*pi*n*k/N)] - [xr(n) * sin(2*pi*n*k/N)]}
N=0
*****************************************************************************/
#include <math.h>
#include "params.h"
void dft(int N, COMPLEX *X)
{
int n, k;
double arg;
int Xr[1024];
int Xi[1024];
short Wr, Wi;
for(k=0; k<N; k++)
{
Xr[k] = 0;
Xi[k] = 0;
For(n=0; n<N; n++)
{
arg =(2*PI*k*n)/N;
Wr = (short)((double)32767.0 * cos(arg));
Wi = (short)((double)32767.0 * sin(arg));
Xr[k] = Xr[k] + X[n].real * Wr + X[n].imag * Wi;
Xi[k] = Xi[k] + X[n].imag * Wr - X[n].real * Wi;
}
}
For (k=0;k<N;k++)
{
X[k].real = (short)(Xr[k]>>15);
X[k].imag = (short)(Xi[k]>>15);
}
}
```

Figura 7. dft.c

```
/************************************ vectors.asm ************************************/
/* vectors.asm - reset vector assembly */
/*************************************************************************************/
.def RESET
.ref _c_int00
.sect ".vectors"
RESET:
Mvk .s2 _c_int00, B2
mvkh .s2 _c_int00, B2
b .s2 B2
nop
nop
nop
nop
nop
```

Figura 8. Vector.asm

```
/************************************ lnk.cmd ************************************/
/* lnk.cmd – ejemplo de archivo de command linker */
/*********************************************************************************/
-c
-heap 0x2000
-stack 0x8000
MEMORY
{
VECS: o = 00000000h l=00200h        /* reset & vectores de interrupcion*/
IPRAM: o = 00000200h l=0FE00h       /* memoria de programa interna*/
IDRAM: o = 80000000h l=10000h       /* memoria de dato interna */
}
SECTIONS
{
vectors > VECS
.text > IPRAM
.tables > IDRAM
.data > IDRAM
.stack > IDRAM
.bss > IDRAM
.sysmem > IDRAM
.cinit > IDRAM
.const > IDRAM
.cio > IDRAM
.far > IDRAM
}
```

Figura 9. Lnk.cmd

Apéndice B

Este apéndice provee una detallada derivación de las ecuaciones usadas para calcular la DFT/IDFT de dos secuencias reales usando una DFT/IDFT compleja.

Transformada directa

Asumamos que $x_1(n)$ y $x_2(n)$ son secuencias de valores reales de longitud N, y hagamos que x(n) sea una secuencia de valor complejo definido como:

$$x(n) = x_1(n) + jx_2(n) \qquad 0 \le n \le N-1$$

La operación DFT es lineal, así la DFT de x(n) puede ser expresada como:

$$X(k) = X_1(k) + jX_2(k) \qquad 0 \le n \le N-1$$

Podemos expresar las secuencias $x_1(n)$ y $x_2(n)$ en términos de x(n) como sigue:

$$x_1(n) = \frac{x(n) + x^*(n)}{2}$$

donde * es el operador conjugado complejo.

$$x_2(n) = \frac{x(n) + x^*(n)}{2j}$$

Lo que sigue muestra que esas igualdades son verdaderas:

$$\frac{x(n) + x^*(n)}{2} = \frac{x_1(n) + jx_2(n) + x_1(n) + jx_2(n)}{2} = x_1(n)$$

$$\frac{x(n) + x^*(n)}{2j} = \frac{x_1(n) + jx_2(n) + x_1(n) + jx_2(n)}{2j} = x_2(n)$$

Por tanto, expresamos la DFT de $x_1(n)$ y $x_2(n)$ en términos de x(n) como se muestra:

$$X_1(k) = DFT[x_1(n)] = \frac{1}{2}\{DFT[x(n)] + DFT[x^*(n)]\}$$

$$X_2(k) = DFT[x_2(n)] = \frac{1}{2j}\{DFT[x(n)] - DFT[x^*(n)]\}$$

De igual manera a $X_1(k)$ y $X_2(k)$ como sigue:

$$X_1(k) = \frac{1}{2}[X(k) + X^*(N-k)]$$

$$X_2(k) = \frac{1}{2j}[X(k) - X^*(N-k)]$$

De las ecuaciones anteriores observamos que realizando una sola DFT de secuencias de valores complejos x(n), se ha obtenido la DFT de dos secuencias de valores reales con sólo un pequeño incremento de cálculos adicionales para hallar $X_1(k)$ y $X_2(k)$ a partir de X(k).

Además, debido a que $x_1(n)$ y $x_2(n)$ son secuencias de valores reales $X_1(k)$ y $X_2(k)$ tienen conjugadas complejas simétricas $X_1(N-k) = X_1^*(k)$ y $X_2(N-k) = X_2^*(k)$, así sólo necesitamos calcular $X_1(k)$ y $X_2(k)$ para k = 0,1,2,....N/2.

$$X_1(k) = \frac{1}{2}[X(k) + X^*(N-k)]$$

$$X_1(N-k) = X_1^*(k) \quad \text{para k = 0,1,...N-1}$$

$$X_2(k) = \frac{1}{2j}[X(k) - X^*(N-k)]$$

$$X_2(N-k) = X_2^*(k)$$

Para calcular estas ecuaciones es útil expresarlas en términos real e imaginario:

$$X_1(k) = \frac{1}{2}[Xr(k) + jXi(k) + Xr(N-k) - Xi(N-k)]$$

$$= \frac{1}{2}[(Xr(k) + Xr(N-k)) + j(Xi(k) - Xi(N-k))]$$

para k = 0,1,...N/2

o $$X_1r(k) = \frac{1}{2}[Xr(k) + Xr(N-k)]$$

$$X_1i(k) = \frac{1}{2}[Xi(k) - Xi(N-k)]$$

Similarmente, esto muestra que:

$$X_2r(k) = \frac{1}{2}[Xi(k) + Xi(N-k)]$$

para k = 0,1,...N/2

$$X_2i(k) = \frac{-1}{2}[Xr(k) - Xr(N-k)]$$

Existen dos casos especiales con las ecuaciones indicadas en el párrafo anterior , k=0 y k=N/2.

Para k=0:

$$X_1r(0) = \frac{1}{2}[Xr(0) + Xr(N)]$$

$$X_1i(0) = \frac{1}{2}[Xi(0) - Xi(N)]$$

$$X_2r(0) = \frac{1}{2}[Xi(0) + Xi(N)]$$

$$X_2i(0) = \frac{-1}{2}[Xr(0) - Xr(N)]$$

Debido a la propiedad de periodicidad de la DFT, conocemos X(k+N)=X(k). Por lo tanto, $X_r(0)= X_r(N)$ y $X_i(0)$ y $X_i(N)$. Usando esta propiedad, las ecuaciones indicadas en el párrafo anterior, pueden ser expresadas como sigue:

$$X_1r(0) = Xr(0)$$
$$X_1i(0) = 0$$
$$X_2r(0) = Xi(0)$$
$$X_2i(0) = 0$$

Para K = N/2

$$X_1r(N/2) = \frac{1}{2}[Xr(N/2) + Xr(N/2)]$$

$$X_1i(N/2) = \frac{1}{2}[Xi(N/2) - Xi(N/2)]$$

$$X_2i(N/2) = \frac{1}{2}[Xi(N/2) + Xi(N/2)]$$

$$X_2i(N/2) = \frac{-1}{2}[Xr(N/2) - Xr(N/2)]$$

$$X_1r(N/2) = Xr(N/2)$$
$$X_1i(N/2) = 0$$
$$X_2r(N/2) = Xr(N/2)$$
$$X_2i(N/2) = 0$$

Apéndice C

DSP TMS320C6201

El TMS320C62001 es un DSP de punto fijo, con un funcionamiento de hasta 1,600 millones de instrucciones por segundo (MIPS). Este DSP opera a 200MHz, es decir con un ciclo de instrucción de 5nseg, ejecutando hasta 8 instrucciones de 32 bits en cada ciclo.
Los dispositivos de esta familia están basados sobre una CPU avanzada VLIW (Very Long Instruction Word) con 8 unidades funcionales que incluyen dos multiplicadores y 6 unidades lógicas aritméticas (ALUs).
La CPU puede ejecutar hasta 8 instrucciones por ciclo. Para complementar la arquitectura se tiene un compilador C muy eficiente que incrementa el funcionamiento y tiempo de desarrollo del código.

Características del TMS320C6201

La Unidad de Control y Proceso (CPU) de este dispositivo consiste de las siguientes etapas:
- 32 registros de propósito general de una longitud de 32 bits y 8 unidades funcionales: dos multiplicadores y seis ALUs.
- Tiene un completo conjunto de herramientas de desarrollo, incluido un eficiente compilador C, un optimizador en assembler para programación en lenguaje assembler simplificado y una interface debbugger bajo windows para ver las características de ejecución del código fuente.
- RAM incorporada para ejecución de algoritmos rápidos.
- Soporte de interface de memoria externa de 32 bits SDRAM, SBSRAM, SRAM, y otras memorias asíncronas para un amplio requerimiento de memoria externa.
- Puerto Host de 16 bits para accesar a las memorias y a los periféricos.
- Controlador DMA multicanal.
- Puertos seriales multicanal.
- Timers de 32 bits.

El procesador C62xx consiste de tres partes principales:
- CPU
- Periféricos
- Memoria

La CPU del C62xx es el bloque de fabricación central de todos los TMS320C62xx y se caracteriza por tener dos rutas de datos cuando ocurre el procesamiento. Cada ruta de datos contiene 4 unidades funcionales (.L,.S,.M,.D) junto con un archivo de registro que contiene 16 registros de propósito general de 32 bits.

El C62xx tiene una arquitectura de almacenamiento/carga en el cual todas las unidades funcionales obtienen operandos desde un archivo de registros, más que directamente desde la memoria.

La unidad .D de datos almacena/carga desde y hasta la memoria desde el registro, con una dirección de búsqueda de 32 bits. La arquitectura C62xx es también direccionable por byte, la unidad .D puede almacenar o cargar datos en 8 bits, 16 bits o 32 bits; además la unidad puede realizar adiciones y restas de 32 bits y cálculos de direcciones.

La unidad .M realiza multiplicaciones y se caracteriza por un multiplicador de 16 bit por 16 bits que produce un resultado de 32 bits. Además de su multiplicador incluye la habilidad de seleccionar ya sea 16 MSBs o 16 LSBs de un operando registro y opcionalmente desplaza a la izquierda la salida del multiplicador por saturación.

La unidad .S realiza primariamente divisiones y desplazamientos pero también realiza operaciones a nivel de bits tales como extraer, fijar y borrar campos de bits, así como operaciones lógicas de 32 bits y adiciones y sustracciones de 32 bits.

Aunque la unidad .S y .D realizan funciones de ALUs, la unidad .L es la principal ALU para la CPU realizando aritmética de 32 y 40 bits.