Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela Mecánica Eléctrica

APLICACIÓN PARA COMPUTADORA DE MONITOREO Y ANÁLISIS DE SEÑALES UTILIZANDO MATLAB Y DSP

José Daniel Hernández Chang

Asesorado por: Ing. Guillermo Puente

Guatemala, febrero de 2011

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

APLICACIÓN PARA COMPUTADORA DE MONITOREO Y ANÁLISIS DE SEÑALES UTILIZANDO MATLAB Y DSP

TRABAJO DE GRADUACIÓN

PRESENTADO A JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR:

JOSÉ DANIEL HERNÁNDEZ CHANG

ASESORADO POR EL ING. GUILLERMO PUENTE

AL CONFERÍRSELE EL TÍTULO DE
INGENIERO ELECTRÓNICO

GUATEMALA, FEBRERO DE 2011

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paiz Recinos
VOCAL I	Ing. Alfredo Enrique Beber Aceituno
VOCAL II	Ing. Pedro Antonio Aguilar Polanco
VOCAL III	Ing. Miguel Ángel Dávila
VOCAL IV	Br. Luis Pedro Ortíz de León
VOCAL V	P.A. José Alfredo Ortíz Herincx
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADOR	Ing. Luis Durán
EXAMINADOR	Ing. Carlos Guzmán
EXAMINADOR	Ing. Marvin Hernández
SECRETARIA	Ing. Marcia Ivónne Velíz Vargas

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

APLICACIÓN PARA COMPUTADORA DE MONITOREO Y ANÁLISIS DE SEÑALES UTILIZANDO MATLAB Y DSP

Tema que me fuera asignado por la Dirección de la Escuela de Mecánica Eléctrica con fecha agosto de 2009.

José Daniel Hernández Chang

Guatemala, 10 de enero de 2011.

Ing. Carlos Eduardo Guzmán Salazar
Coordinador del Área de Electrónica
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC.

Ingeniero Guzmán:

Por este medio me permito dar aprobación al Trabajo de Graduación titulado: **"APLICACIÓN PARA COMPUTADORA DE MONITOREO Y ANALISIS DE SEÑALES UTILIZANDO MATLAB Y DSP"**, desarrollado por el estudiante **José Daniel Hernández Chang con camé 2003 - 12563**, ya que considero que cumple con los requisitos establecidos, por lo que el autor y mi persona somos responsables del contenido y conclusiones del mismo.

Sin otro particular, aprovecho la oportunidad para saludarlo.

Atentamente,

Ing. Guillermo Antonio Puente Romero
ASESOR
Colegiado 5898

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÍA

Ref. EIME 03. 2010
Guatemala, 14 de enero 2011.

Señor Director
Ing. Guillermo Antonio Puente Romero
Escuela de Ingenierìa Mecànica Elèctrica
Facultad de Ingenierìa, USAC.

Señor Director:

Me permito dar aprobación al trabajo de Graduación titulado: "APLICACIÓN PARA COMPUTADORA DE MONITOREO Y ANÁLISIS DE SEÑALES UTILIZANDO MATLAB Y DSP", del estudiante, JOSÉ DANIEL HERNÁNDEZ CHANG, que cumple con los requisitos establecidos para tal fin.

Sin otro particular, aprovecho la oportunidad para saludarle.

Atentamente,
ID Y ENSEÑAD A TODOS

Ing. Carlos Eduardo Guzmán Salazar
Coordinador de Electrónica

CEGS/sro

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÍA

REF. EIME 08. 2011.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto Bueno del Coordinador de Área, al trabajo de Graduación del estudiante; JOSÉ DANIEL HERNÁNDEZ CHANG titulado: "APLICACIÓN PARA COMPUTADORA DE MONITOREO Y ANÁLISIS DE SEÑALES UTILIZANDO MATLAB Y DSP", procede a la autorización del mismo.

Ing. Guillermo Antonio Puente Romero

GUATEMALA, 20 DE ENERO 2,011.

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

DTG. 035.2011

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica, al trabajo de graduación titulado: **APLICACIÓN PARA COMPUTADORA DE MONITOREO Y ANÁLISIS DE SEÑALES UTILIZANDO MATLAB Y DSP,** presentado por el estudiante universitario **José Daniel Hernández Chang,** autoriza la impresión del mismo.

IMPRÍMASE:

Ing. Murphy Olympo Paiz Recinos
Decano

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Guatemala, 4 de febrero de 2011.

/gdech

ACTO QUE DEDICO A:

Dedico este trabajo de graduación a:

La Verdad que se origina en la naturaleza.

AGRADECIMIENTOS

Agradecimientos a quienes colaboraron a que mi existencia llegara a este punto, a los seres supremos por la vida y la verdad; a mis ancestros, a mis progenitores por llevarme por el camino que conduce hacia el conocimiento de la verdad infinita; y a aquellas personas con las cuales he compartido trayectos de este camino.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

GLOSARIO

Armónicos Son frecuencias múltiples mayores a la frecuencia fundamental de trabajo del sistema y cuya amplitud va decreciendo conforme aumenta el múltiplo.

Bit Es una señal electrónica que puede estar encendida (1) o apagada (0). Es la unidad más pequeña de información que utiliza una computadora. Son necesarios 8 bits para crear un byte.

Decibel Es un parámetro, que presenta en pequeños números manejables grandes cantidades de potencia, permitiendo calcular las pérdidas o ganancia absolutas en un módulo o en una red completa, utilizando sumas y restas tanto como multiplicación y división. El decibel es un décimo de un BEL.

Densidad de probabilidad Se utiliza en estadística con el propósito de conocer cómo se distribuyen las probabilidades de un suceso o evento, en relación al resultado del suceso. El área total encerrada bajo la curva es igual a 1.

Espectro de frecuencia Es una medida de la distribución de amplitudes de cada frecuencia. Puede aplicarse a cualquier concepto asociado con frecuencia o movimientos, ondulatorios, sonoro y electromagnético.

Filtro Es un dispositivo que discrimina una determinada frecuencia o grupo de frecuencias de una señal eléctrica que pasa a través de él, con la cual puede cambiar la magnitud y la fase.

HDL* y *VHDL Es un lenguaje descriptivo para hardware (*Hardware Description Language*) usado en los diseños electrónicos para describir sistemas digitales o de señales mixtas en circuitos integrados.

Impulso Es la variación, generalmente breve (unos pocos microsegundos) en intensidad o tensión de una corriente pulsante.

LUT	Es una estructura de datos, generalmente compuestos en vectores, se utilizan para buscar datos previamente grabados, esto se hace por medio de un índice. Es buscar en tabla, donde se almacenan los datos.
RAM	Son las siglas de *random access memory*, un tipo de memoria de procesador a la que se puede acceder aleatoriamente; es decir, se puede acceder a cualquier byte de memoria sin acceder a los bytes precedentes.
Multiplexación	Es la combinación de dos o más canales de información en un solo medio de transmisión usando un dispositivo llamado multiplexor.
Respuesta al impulso	Es la respuesta de un sistema cuando la entrada es un impulso y como salida se tiene una descripción equivalente a la dada por una función de transferencia. Un sistema puede ser completamente descrito por su respuesta al impulso, ya que todas las señales pueden ser representadas por una superposición de impulsos.

RESUMEN

El procesamiento digital de señales es una de las técnicas que más se utiliza en la tecnología moderna, debido al creciente uso de circuitos digitales. En el presente trabajo se realiza el procedimiento para configurar este tipo de chips digitales, en el cual se analizan señales que también son procesadas.

En el presente trabajo se muestra como Matlab y Altera se han unido para crear una forma eficaz de configurar un proceso *DSP* en un chip. Con la ayuda de sus programas simuladores Simulink, Quartus II y Model Sim, se han creado dos modelos con los cuales se ilustra los pasos a seguir para la programación de un dispositivo *DSP* (*digital signal prossesor*). Y culminará llevando el programa listo para ser descargado al chip. Los códigos son generados por los *Blockset* en Simulink que la empresa Altera ha construido y que tiene el nombre de "Altera *DSP Builder*".

Los modelos que se presentarán son la generación de un algoritmo de división entre dos números, usando únicamente bloques primitivos como lo son suma, resta, multiplicación, etc. El siguiente modelo es el análisis de una señal y el cálculo de la transformada rápida de Fourier abreviada *FFT*, que es muy útil en el diseño de sistemas y en el análisis de señales.

OBJETIVOS

Generales:

Elaborar un sistema que incluya tanto el procedimiento como el dispositivo de una aplicación de procesamiento digital de señales, creando un modelo para analizar una señal y calcular su transformada rápida de Fourier. Para desarrollar un modelo que permita procesar dos señales y calcular su división. Llevando el desarrollo hasta el punto que el programa esté listo para ser descargado al procesador.

Específicos:

1. Diseñar el programa para el chip *DSP* por medio de Simulink y el *Blockset DSP Builder* de Altera.
2. Generar códigos *VHDL* correspondientes a cada modelo por medio del programa Quartus II.
3. Visualizar los resultados de la programación en Model Sim, y en Quartus II obtener la distribución de los pines del chip *DSP*.
4. Obtener los reportes de la simulación y del desempeño del *DSP*.
5. Analizar y procesar una señal obtenida por medio de un micrófono.

INTRODUCCIÓN

Cuando se estudia la carrera de ingeniería electrónica uno de los mayores intereses es el de diseñar y desarrollar proyectos utilizando herramientas de alto nivel tecnológico. Es el caso del procesamiento digital de señales, el cual se encuentra en casi todas las aplicaciones de la electrónica, desde filtros hasta centrales de telefonía móvil.

Por esa razón este trabajo está dedicado a buscar la implementación del procesamiento de señales en conjunto con la programación del dispositivo que es un procesador.

El trabajar con *DSPs* es una tarea compleja, ya que hay que conocer tanto la teoría como la parte práctica.

Ésta es la primera intención de este trabajo, unificar en la medida de lo posible estos dos conceptos, poner en práctica la teoría que se enseña en las aulas y en los libros. Sin embargo, debido a la dificultad de obtener un set completo de *DSP* se contempló la simulación del chip de una manera completa.

Es así como se desarrolla el siguiente trabajo, sobre las plataformas de Simulink de Matlab y Quartus II como *software* de simulación. Estas simulaciones incluyen el sistema completo que se diseña. Por ejemplo, un modelo es el de calcular la trasformada rápida de Fourier, pero no solo se desarrolla el cálculo sino también se incluyen elementos de adquisición de datos, control y visualización de datos de salida.

Pero, además de ser un diseño simulado, también se obtendrá el código para chip *DSP*. El código es un archivo *VHDL* el cual es la manera en que se programan los circuitos integrados. Con este código creado el programa estará listo para ser descargado al chip *DSP* y ponerlo a funcionar; y aún más, en el diseño se mostrará como configurar el chip, por ejemplo, los valores de la carga, el voltaje de alimentación, los valores de las señales de entrada y la distribución de los pines del chip.

Las técnicas usadas en el trabajo están basadas en la teoría, como es la transformada rápida de Fourier por el método de discreción en la frecuencia; así también el desarrollo de la serie de Taylor para mostrar cómo se puede calcular la división de dos señales utilizando únicamente operaciones de suma, resta y multiplicación. Esto es debido a que la mayoría de *DSP* en su set de instrucciones básico no incluye la operación de división.

Se explicará detalladamente en cada paso el proceso, el orden que se utilizó en el diseño del sistema por medio de texto que muestra la teoría, la manera de utilizar el software y los resultados al implementar el diseño. También se presentarán reportes de utilización de recursos en el *DSP*.

1. REPRESENTACIÓN MATEMÁTICA DE LAS SEÑALES ELÉCTRICAS

En este capítulo se presentarán las formas elementales de las señales eléctricas y cómo son representadas matemáticamente. Entre las señales que se discutirán están la función de escalón unitario, la función de rampa y la función delta. Con base en estas señales se indicará las propiedades de muestreo y de traslación que se puede hacer con ellas. Así mismo cómo se pueden representar otras señales eléctricas en base a estas tres funciones.

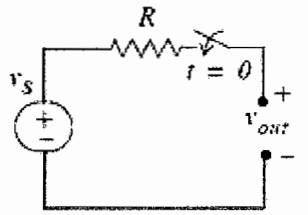
1.1 Funciones Elementales

Estas son un conjunto de funciones que tienen una representación matemática y son utilizadas para describir el comportamiento de algunas señales eléctricas, como la interrupción de voltaje, el crecimiento lineal de voltaje etc.

1.1.1 Función escalón unitario:

Considerando el siguiente circuito mostrado en la figura 1, en el cual hay dos valores de voltaje de salida, dependiendo de la posición del interruptor.

Figura 1. 1 Circuito con interruptor



Fuente: Steven T. Karris
Signals and Sistems with MATLAB Aplications
Pág. 13

Si al momento de cerrar el interruptor se establece como el tiempo cero; se puede escribir esta transición de la siguiente manera:

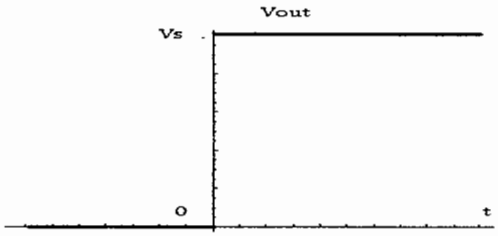
Para el intervalo $-\infty < t < 0$ el interruptor permanecerá abierto, y el voltaje de salida será igual a cero. Para el intervalo de $0 < t < \infty$ el interruptor se cierra, y dejara que fluya corriente por la resistencia, haciendo que exista una diferencia de potencial en los bordes conocida como *Vout*.

De esa manera se puede representar el análisis anterior escribiendo la siguiente expresión:

$$Vout = \begin{cases} 0, & -\infty < t < 0 \\ Vs, & 0 < t < \infty \end{cases} \qquad \text{Ecu. 1.1}$$

Graficando esta función se obtiene:

Figura 1. 2 Voltaje de salida *Vout*



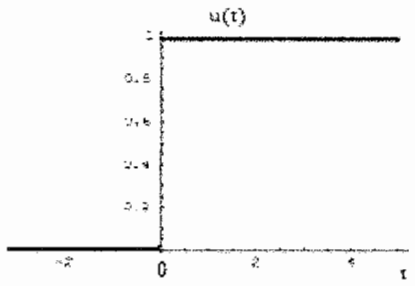
La forma de onda de la figura 1.2 indica que es una función discontinua. Se dice que es discontinua cuando los límites laterales de la función en un punto existen pero no coinciden.

Por lo anterior se puede comparar el comportamiento de esta señal con la función escalón unitario, que se define de la siguiente manera:

$$u(t) = \begin{cases} 0, -\infty < t < 0 \\ 1, \quad 0 \leq t < \infty \end{cases} \qquad \text{Ecu. 1.2}$$

Y su gráfica es:

Figura 1. 3 Escalón unitario

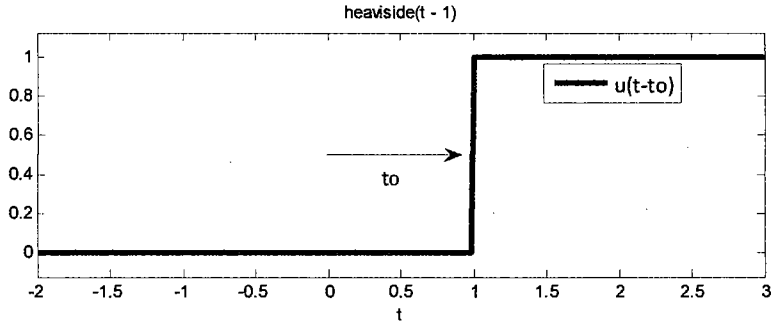


Una propiedad importante de esta función es la de traslación. Estas suceden de la siguiente manera:

Si u (t - t_o) la función se traslada hacia la derecha, como muestra la figura:

$$u(t - t_o) = \begin{cases} 0, & t < t_o \\ 1, & t > t_o \end{cases} \qquad \text{Ecu. 1.3}$$

Figura 1. 4 Señal para u(t-t0)

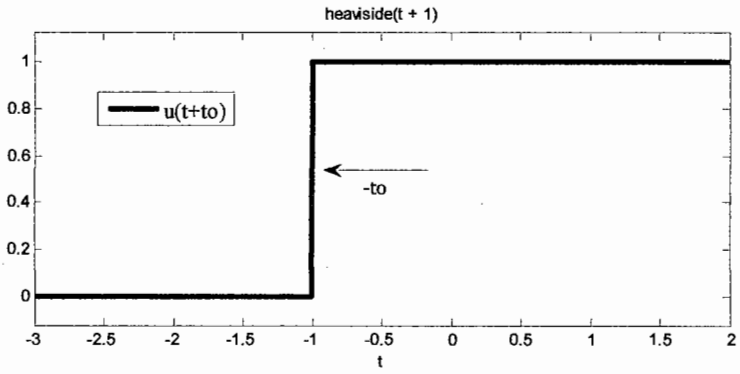


De otra forma si u (t + t_o) la función se traslada hacia la izquierda, como muestra la figura:

$$u(t + t_o) = \begin{cases} 0, & t < -t_o \\ 1, & t > -t_o \end{cases} \qquad \text{Ecu. 1.4}$$

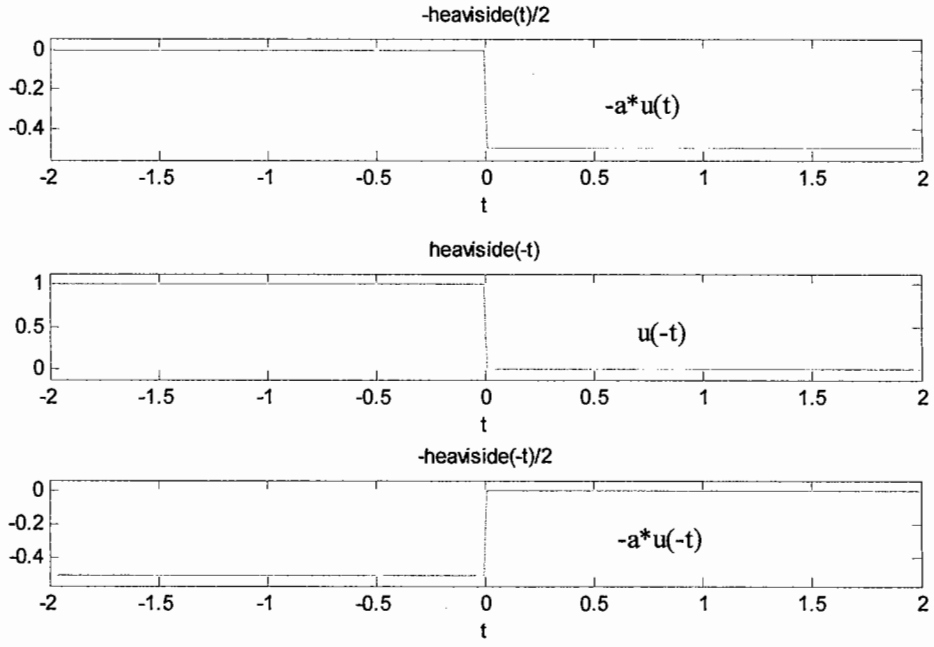
La representación gráfica de la ecuación 1.4 se muestra en la figura 1.5.

Figura 1. 5 Señal para u(t+t0)



Otras formas de representar la función escalón unitario se presentan en la siguiente figura:

Figura 1. 6 Variaciones del escalón unitario



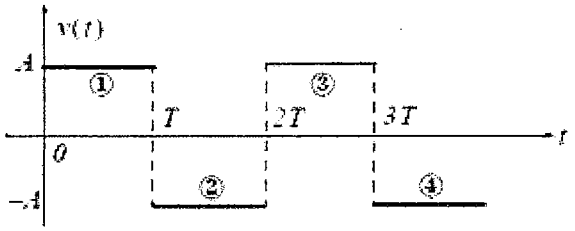
La función escalón unitario ofrece un modelo conveniente para representar la aplicación repentina de una fuente de voltaje o corriente. Por ejemplo una fuente de voltaje de 24 v aplicada en t = 0, puede ser denotada como $24u(t)$ V. De igual manera

una fuente de voltaje senoidal $v(t) = Vm\sin(\omega t)$ V. que es aplicada a un circuito en $t = t_o$, puede ser descrito por $v(t) = Vm\sin(\omega t)u(t - t_o)$ V. Además si la excitación del circuito es un pulso rectangular, triangular, diente de sierra o de otro tipo puede ser representado como una suma o sustracción de funciones de escalón unitario.

A continuación se presenta un ejemplo que ilustra lo dicho en el párrafo anterior, en el cual se muestra la importancia de la función escalón unitario para el estudio de señales eléctricas.

Se desea expresar una señal digital de voltaje que tiene una forma cuadrada y tiene periodo T a partir de t = 0; tal como lo muestra la figura:

Figura 1. 7 Señal cuadrada para ejemplo



Fuente: Steven T. Karris
Signals and Sistems with MATLAB Aplications
Pág. 17

Se divide la señal en cuatro partes y se representa como una suma de funciones escalones. El segmento de línea 1 tiene un valor de A, empieza en t = 0 y termina en t = T, así que este segmento se expresa:

$$v1(t) = A(u(t) - u(t - T)) \qquad \text{Ecu. 1.5}$$

Para el segundo segmento de línea que empieza en t = T y termia en t = 2T el voltaje es:

$$v2(t) = -A(u(t-T) - u(t-2T)) \qquad \text{Ecu. 1.6}$$

De la misma manera el valor para el segmento 3, que empieza en t = 2T y termina en t = 3T

$$v3(t) = A(u(t-2T) - u(t-3T)) \qquad \text{Ecu. 1.7}$$

Y para el último que tiene su inicio en t = 3T y fin en t = 4T:

$$v4(t) = -A(u(t-3T) - u(t-4T)) \qquad \text{Ecu. 1.8}$$

De esta manera está representada por partes la señal de la figura 1.7, ahora para tener una representación completa se sumará cada una de las ecuaciones:

$$v(t) = v1(t) + v2(t) + v3(t) + v4(t) \qquad \text{Ecu. 1.9}$$

$$v(t) = A(u(t) - u(t-T)) - A(u(t-T) - u(t-2T)) + A(u(t-2T) - u(t-3T)) - A(u(t-3T) - u(t-4T))$$

Ecu. 1.10

Sumando la expresión anterior:

$$v(t) = A(u(t) - 2u(t-T) + 2u(t-2T) - 2u(u-3T) + \cdots)$$

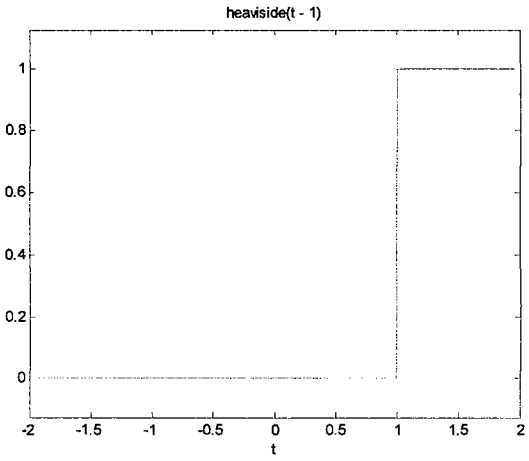
Ecu. 1.11

En Matlab la representación de esta función se hace con la función *Heaviside* (t+to). Así para definir una función escalón unitario y graficarla se hace de la siguiente manera:

```
u=heaviside(t-1);
% crea la función escalón unitario corrido en 1
H=ezplot(u,[-2 2]);
%  obtiene la gráfica de la función anterior
```

A continuación se muestra la gráfica de la función.

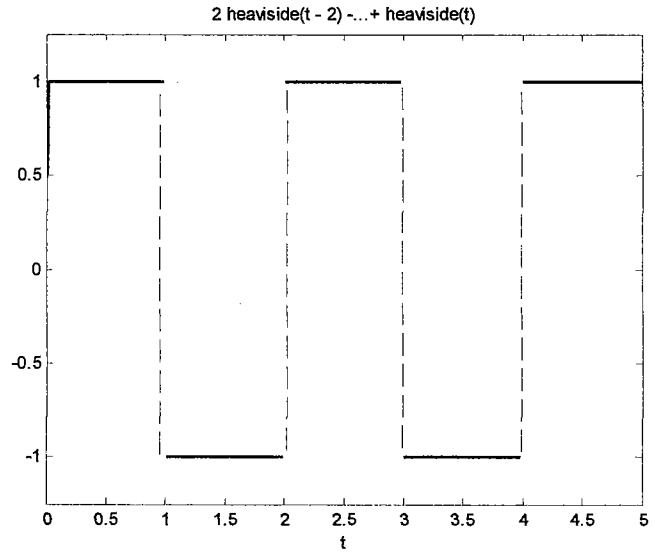
Figura 1. 8 Escalón trasladada una unidad



Para hacer una demostración de cómo Matlab grafica la función del ejemplo anterior, se escribe lo siguiente:

```
>> u=heaviside(t)-2*heaviside(t-1)+2*heaviside(t-2)-2*heaviside(t-3)+2*heaviside(t-4);

>> H=ezplot(u,[0 5]);
```

Figura 1. 9 Suma de escalones unitarios

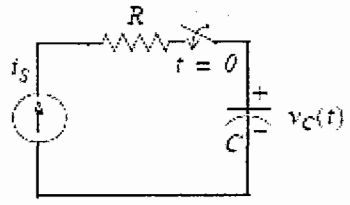


Esta es una función importante para poder representar secuencias de pulsos eléctricos de una misma amplitud y con cierto intervalo de tiempo.

1.1.2 Función rampa unitaria:

Para describir esta función se analizará el voltaje de salida en un circuito que posee un capacitor y una resistencia, como sigue:

Figura 1. 10 Circuito RC con interruptor



Fuente: Steven T. Karris
Signals and Sistems with MATLAB Aplications
Pág. 21

En este caso se tiene una fuente de corriente continua, una resistencia R, el capacitor C y un interruptor que cerrará en un tiempo t=0. Ahora se expresa el voltaje vc(t) como una función de escalones unitarios. La corriente a través del capacitor es Ic(t)=is(t) = constante, de otra manera el voltaje vc(t) es:

$$vc(t) = \frac{1}{C}\int_{-\infty}^{t} ic(t)d\mathcal{T} \qquad \text{Ecu. 1.12}$$

Como el interruptor se cierra en t = 0, se puede escribir como:

$$ic = is\,u(t) \qquad \text{Ecu. 1.13}$$

Si Vc = 0 para t < 0, se puede escribir la ecuación 1.12 como:

$$vc(t) = \frac{1}{C}\int_{-\infty}^{t} is\,u(t)dt = \frac{is}{C}\int_{-\infty}^{0} u(t)t + \frac{is}{C}\int_{0}^{t} u(t)t = 0 + \frac{is}{C}\int_{0}^{t} u(t)dt$$

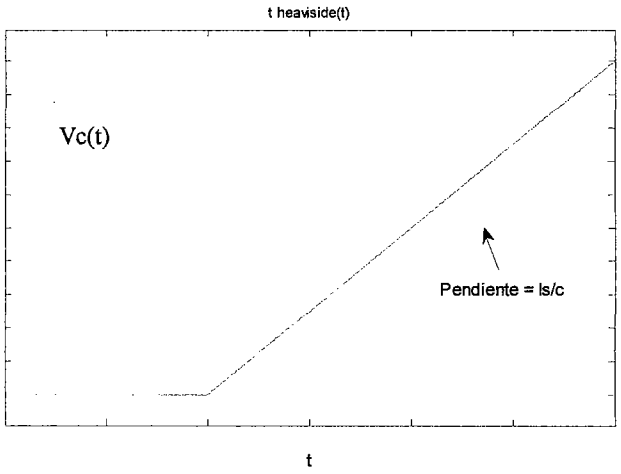
Ecu. 1.14

Que da:

$$Vc(t) = \frac{is}{C}u(t) \qquad \text{Ecu. 1.15}$$

Por lo tanto, se ve que cuando un capacitor es cargado con una corriente constante, el voltaje que se forma es un voltaje que es una función lineal y forma una rampa. Cuya pendiente es is/c, como se muestra en la figura 1.11.

Figura 1.11 Voltaje del capacitor



Utilizando los resultados en el ejemplo anterior se define la función de rampa unitaria como sigue:

$$r(t) = \int_{-\infty}^{t} u(t)dt \qquad \text{Ecu. 1.16}$$

Evaluando la integral se puede dar cuenta que el área bajo la función escalón es igual a t, de esta manera se expresa la función rampa de la siguiente manera:

$$r(t) = \begin{cases} 0, & t < 0 \\ t, & t > 0 \end{cases} \qquad \text{Ecu. 1.17}$$

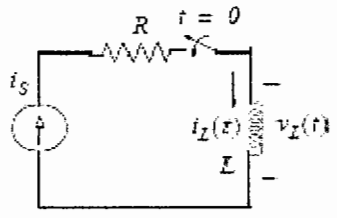
En realidad esta función se puede dar como un caso particular de la integral de esta forma:

$$Un(t) = \begin{cases} 0, & t < 0 \\ t^n, & t > 0 \end{cases} \quad \text{O} \quad Un(t) = \int_{-\infty}^{t} U_{n-1}(t)dt \qquad \text{Ecu. 1.18}$$

1.1.3 Función Delta δ (t)

Para demostrar esta función delta se utilizará el circuito de la figura 1.12. Y se encontrará la corriente del inductor en términos de la función escalón unitario.

Figura 1. 12 Circuito RL con interruptor



Fuente: Steven T. Karris
Signals and Sistems with MATLAB Aplications
Pág. 23

En este circuito el interruptor se cierra en t = 0, y la corriente $i_L = 0$ para t < 0,

El voltaje a través de inductor es:

$$V_L(t) = L\frac{di_L}{dt} \qquad \text{Ecu. 1.19}$$

Y como el interruptor cierra en t = 0 la corriente es:

$$i_L(t) = i_s u(t) \qquad \text{Ecu. 1.20}$$

Sustituyendo la ecuación anterior en la ecuación 1.19 se tiene que:

$$V_L(t) = Li_s\frac{du(t)}{dt} \qquad \text{Ecu.1.21}$$

De esta última ecuación se puede ver algo interesante y es que u (t) es una constante que toma valores de 0 y 1 para todo el tiempo excepto para cuando t = 0 donde se produce la discontinuidad. Como la derivada de una constante es cero, la derivada de la función escalón tiene un valor distinto de cero solamente en t= 0.

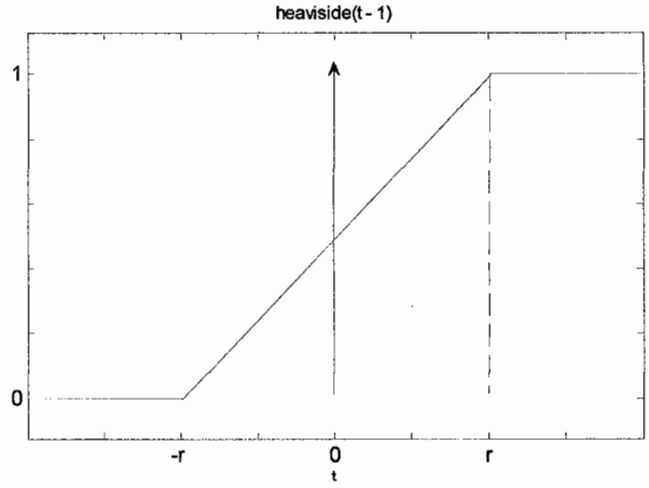
La explicación anterior lleva a la definición de la función delta, que es la derivada de la función escalón, se describe con el símbolo δ (t) y se define:

$$\int_{-\infty}^{t} \delta(t)dt = u(t) \qquad \text{Ecu. 1.22}$$

$$\delta(t) = 0\ para\ \ t \neq 0 \qquad \text{Ecu. 1.23}$$

Para entender mejor la función delta se debe considerar la función escalón:

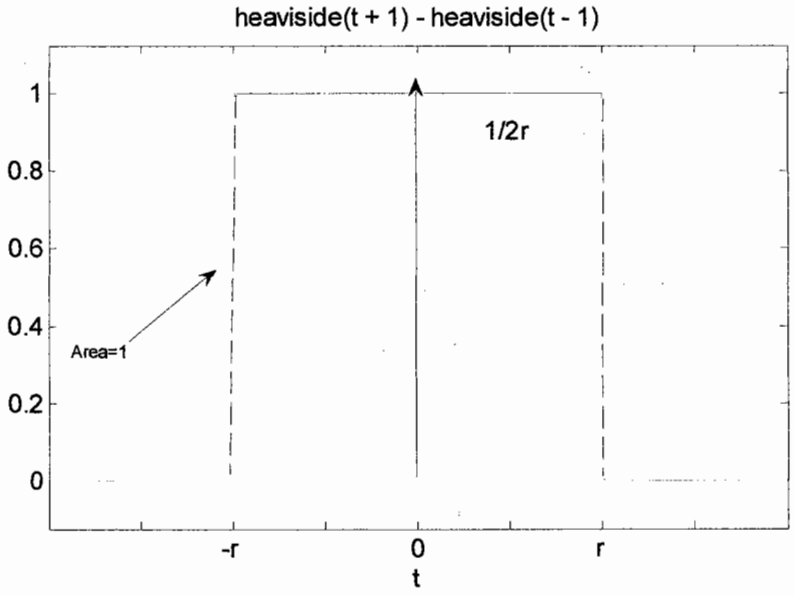
Figura 1. 13 Limite de un escalón unitario



La gráfica anterior se convierte en la función escalón unitario cuando $r \to 0$

La siguiente gráfica es la derivada de la figura 1.13

Figura 1. 14 Representación del área



Con esta gráfica se puede ver que cuando $r \rightarrow 0$, 1/2r crece sin límites, pero el área del rectángulo se mantiene en 1. En el límite se puede decir que la función delta $\delta(t)$ se comporta como un impulso en el origen, con amplitud infinita, cero de ancho y área igual a 1. A continuación se describe dos propiedades de la función delta que serán de gran ayuda en el tratamiento de señales digitales.

1.1.4 Propiedad de muestreo de la Función Delta δ (t)

La propiedad de muestreo de la función delta establece que:

$$f(t)\delta(t-a) = f(a)\delta(t) \qquad \text{Ecu.1.24}$$

Y en el caso que a = 0 queda:

$$f(t)\delta(t) = f(0)\delta(t) \qquad \text{Ecu. 1.25}$$

Lo que significa es que la multiplicación de una función f (t) por la delta $\delta(t)$ resulta en el muestreo de la función en el instante de tiempo donde la función delta no es cero. El estudio de sistemas de tiempo discreto está basado en esta propiedad.

1.1.5 Propiedad de traslado de la función delta δ (t):

Esta propiedad establece lo siguiente:

$$\int_{-\infty}^{\infty} f(t)\delta(t-a) = f(a) \qquad \text{Ecu. 1.26}$$

Esto es, si se multiplica f(t) por la función delta $\delta(t-a)$ y luego se integra de menos infinito a infinito, se obtiene el valor de f(t) evaluada en t = a.

El siguiente es un ejemplo en Matlab:

```
>> syms t a
>> a=4;
>> int(dirac(t-a)*sin(t),-inf,inf)

ans =

sin(4)
```

1.2 Respuesta al Impulso y Convolución

En esta sección se empieza describiendo la respuesta al impulso, que es, la respuesta de un circuito que está sujeto a la excitación de una función impulso. Posteriormente se define la Convolución y cómo se aplica en el análisis de circuitos.

1.2.1 La respuesta al impulso en el Domino del Tiempo

En esta sección se discutirá la respuesta al impulso de un sistema, esto es, la salida, ya sea voltaje o corriente, de un circuito cuando su entrada es la función delta $\delta(t)$. Se puede determinar a partir de las ecuaciones de estado del sistema, con la señal de entrada la función delta y el valor inicial igual a 0. Se resuelven las ecuaciones por cualquier

método, y al final se utiliza la propiedad de traslado de la función delta para obtener la respuesta del sistema.

Un sistema lineal se describe en términos de su respuesta al impulso, un sistema es lineal cuando cumple con el principio de superposición; es decir, la respuesta de un sistema lineal a varias excitaciones aplicadas en forma simultánea es igual a la suma de las respuestas del sistema cuando cada excitación se aplica individualmente.

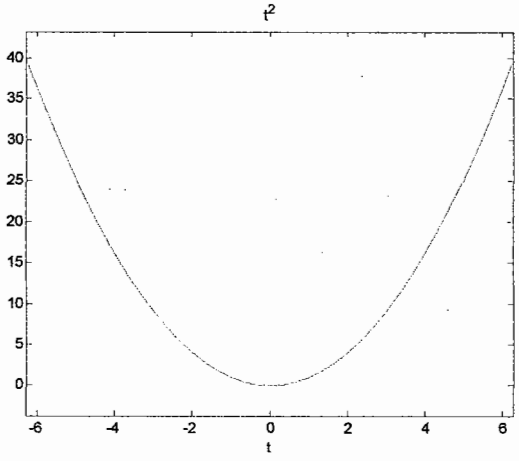
1.2.2 Funciones impares y pares del tiempo:

Una función f(t) es par si la siguiente relación se cumple:

$$f(t) = f(-t) \qquad \text{Ecu.1.27}$$

Esto significa que si en una función par se puede cambiar t por –t y se obtendrá la misma función f (t), a continuación se da una gráfica de $f(t) = t^2$.

Figura 1. 15 Función cuadrática

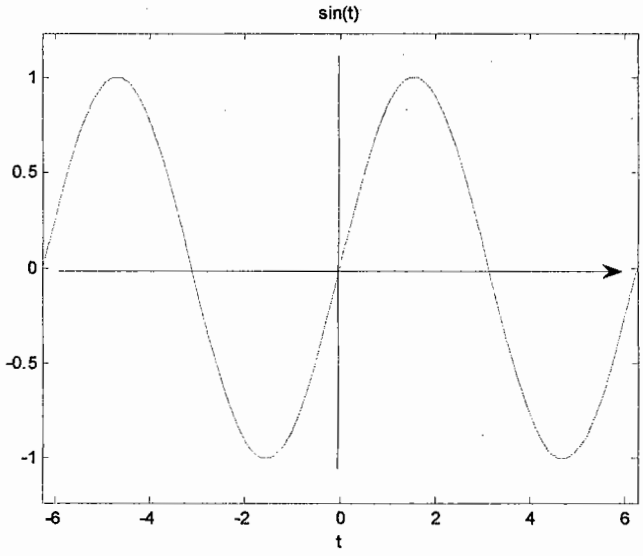


Por otro lado una función impar en el tiempo tiene la siguiente relación:

$$-f(-t) = f(t)$$ Ecu.1.28

Lo que esta relación indica es que si se remplaza t por –t se obtiene la función negativa de f (t). Por ejemplo el seno (t) es una función impar:

Figura 1. 16 Función seno



Un dato importante que notar sobre las funciones impares es que: $f(0) = 0$; pero esto no significa que se cumpla el caso contrario. Es decir que porque una función evaluada en cero sea igual a cero no se debe pensar que la función es impar. Esto resulta útil para saber el valor de f(0) si se sabe que f(t) es una función impar.

El producto de dos funciones pares o impares da como resultado una función par. El conocer el tipo de función puede ayudar a resolver integrales u operaciones que involucran las siguientes formas. Para una función par se puede resolver la integral de esta manera:

$$\int_{-T}^{T} f(t)dt = 2\int_{0}^{T} f(t)dt$$ Ecu. 1.29

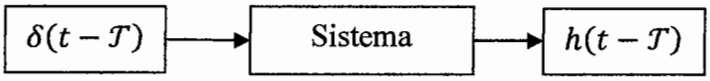
Para el caso que la función sea impar se tiene:

$\int_{-T}^{T} f(t)dt = 0$ Ecu. 1.30

1.2.3 Convolución

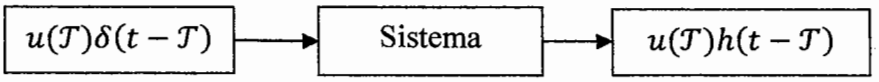
Considerando una red donde la entrada es δ(t) y tiene como salida la respuesta al impulso h (t). Se puede representar en diagrama de bloques de la siguiente manera:

Figura 1. 17 Diagrama respuesta al impulso



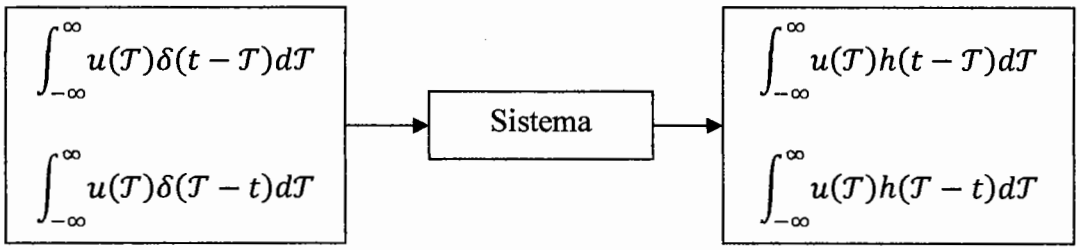
Si se coloca a la entrada una función u (t) cuyo valor en $t = \mathcal{T}$ es u ($\mathcal{T}$) entonces:

Figura 1. 18 Respuesta al escalón unitario



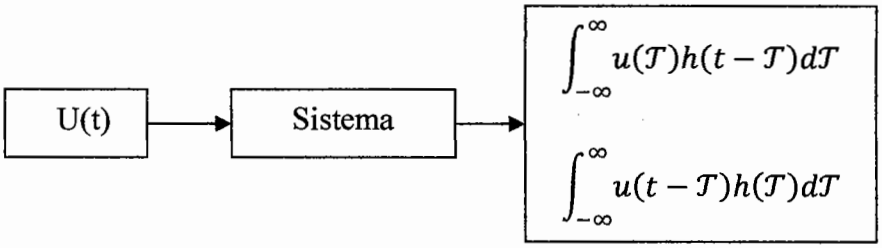
Multiplicando ambos lados por una constante d$\mathcal{T}$, integrando de -∞ a ∞, y utilizando la igualdad: $\delta(t-\mathcal{T}) = \delta(\mathcal{T}-t)$se obtiene la siguiente figura:

Figura 1. 19 Propiedades de Convolución



Utilizando las propiedades de la función delta se obtiene para el lado izquierdo que las integrales son igual a u(a) y queda:

Figura 1. 20 Convolución



A las integrales

$\int_{-\infty}^{\infty} u(\mathcal{T})h(t-\mathcal{T})d\mathcal{T}$ Y $\int_{-\infty}^{\infty} u(t-\mathcal{T})h(\mathcal{T})d\mathcal{T}$ Ecu. 1.31

Se les conoce como las integrales de Convolución, lo que estas integrales dicen es que si se conoce la respuesta al impulso h (t) de una red, se puede conocer la respuesta a cualquier entrada u (t) utilizando las ecuaciones 1.31. La integral de Convolución se

denota de la siguiente manera: $u(t) * h(t)$ o $h(t) * u(t)$, donde el signo * indica la Convolución de las dos funciones.

Si se tiene un sistema que tiene una respuesta al impulso tal que $h(t) = e^{At}B$. Entonces se puede conocer la salida de este sistema $y(t)$ a cualquier entrada $u(t)$con la siguiente relación:

$$y(t) = \int_{-\infty}^{\infty} e^{A(t-\mathcal{T})} bu(\mathcal{T})d\mathcal{T} = e^{At}b \int_{-\infty}^{\infty} e^{-\mathcal{T}} u(\mathcal{T})d\mathcal{T} \qquad \text{Ecu. 1.32}$$

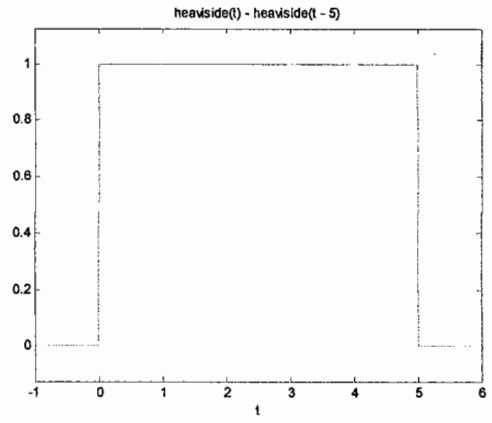
Otra propiedad importante de la Convolución es que cumple con las leyes asociativas y conmutativas.

A continuación se presenta un ejemplo de la Convolución utilizando Matlab.

Se comienza definiendo un pulso unitario de t = 0 hasta t = 5,

Figura 1. 21 Pulso unitario

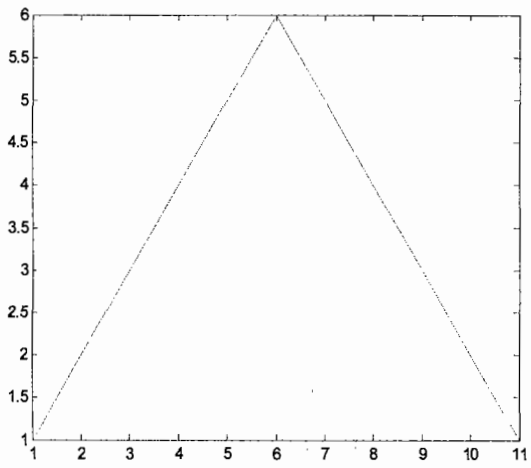
```
>> y=[1;ones(5,1)]
y =
     1
     1
     1
     1
     1
     1
>> plot(y)
```



Se calculará la Convolución de dos pulsos de la misma forma, para luego volver a convolucionar la respuesta que se obtuvo con otro pulso idéntico.

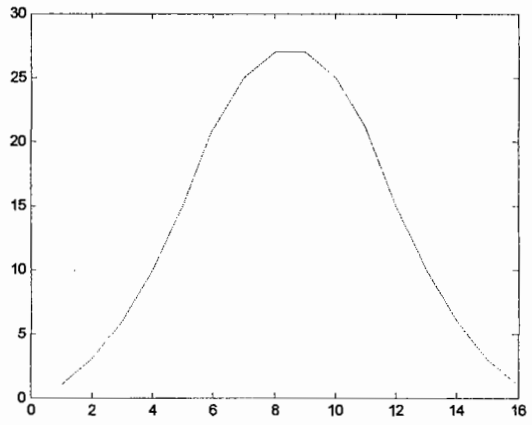
```
>> w=conv(y,y)
w =
     1
     2
     3
     4
     5
     6
     5
     4
     3
     2
```

Figura 1. 22 Resultado de Convolución



Esta es la gráfica de la primera convolución, se puede apreciar que la convolucion de dos pulsos tiene la respuesta de un triangulo.

Figura 1. 23 Doble Convolución



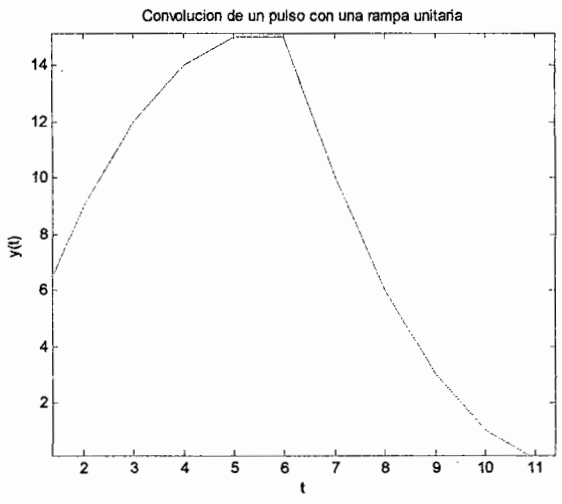
Esta otra gráfica es el resultado de hacer la Convolución de un pulso con la respuesta triangular obtenida anteriormente.

El próximo ejemplo es una Convolución de una función rampa con pendiente negativa siendo convolucionado por un pulso unitario:

Figura 1. 24 Ejemplo Convolución

```
t=(0:5);
ramp=5-t;
puls=[1;ones(5,1)];
y=conv(ramp,puls);
plot(y)
```

Con esta gráfica se muestra el resultado.



1.3 Transformaciones del Dominio

Una señal se caracteriza por tener dos atributos esenciales, su representación en el tiempo y en la frecuencia. En esta sección se mostrará la manera en que una señal puede ser representada en estas dos formas. Se explicará primero las representaciones en el tiempo, esto con las series de Fourier, para luego entrar a la transformada de Fourier y las transformada Z, con lo cual concluye este capítulo dedicado a la representación matemática de señales.

1.3.1 Series de Fourier

El matemático francés Joseph Fourier, encontró que cualquier forma de onda periódica puede ser expresada por la suma de series de harmónicas con relación senoidales, o sea senoidales cuyas frecuencias son múltiplos de la frecuencia fundamental de la señal en cuestión (o la primera armónica).

Por ejemplo una forma de onda periódica puede expresarse de la siguiente manera:

$$f(t) = \frac{1}{2}a_0 + a_1 \cos wt + a_2 \cos 2wt + a_3 \cos 3wt + a_4 \cos 4wt + \cdots + b_1 \sin wt + b_2 \sin 2wt + b_3 \sin 3wt + b_4 \cos 4wt + \cdots \qquad \text{Ecu. 1.33}$$

O abreviado:

$$f(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty}(a_n Cos\, nwt + b_n Sin\, nwt) \qquad \text{Ecu. 1.34}$$

Donde el primer término $\frac{1}{2}a_0$ es una constante, este representa el valor promedio de f (t), y en una señal eléctrica es el valor DC. Los términos con los coeficiente a_1 y b_1 juntos representan los componentes w de la frecuencia fundamental. De la misma manera los términos con coeficientes a_2 y b_2 representan los componentes debido a la segunda armónica 2w, y de esta forma se sigue.

Como cualquier señal periódica puede representarse como una serie de Fourier, se puede decir que las sumas de las componentes *DC*, primera armónica, segunda armónica, etc. Deben de producir la forma de onda f(t).

1.3.1.1 Evaluación de los Coeficientes:

Los coeficientes de las series se calculan con las siguientes integrales:

$$\frac{1}{2}a_0 = \frac{1}{2\pi}\int_0^{2\pi} f(t)dt \qquad \text{Ecu. 1.35}$$

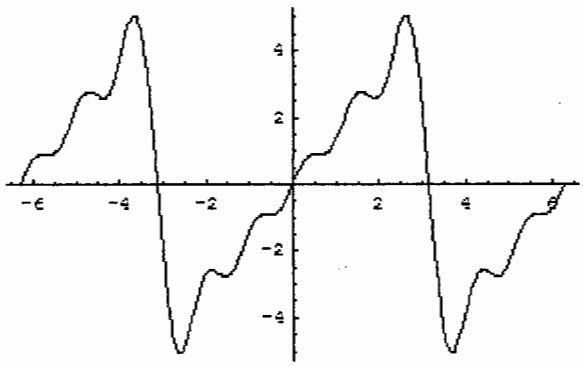
$$a_n = \frac{1}{2\pi}\int_0^{2\pi} f(t)\cos nt\, dt \qquad \text{Ecu. 1.36}$$

$$b_n = \frac{1}{2\pi}\int_0^{2\pi} f(t)\sin nt\, dt \qquad \text{Ecu. 1.37}$$

Las ecuaciones anteriores combinadas con la ecuación 1.34 generan la forma de onda f (t), al agregar más coeficientes se tendrá una representación más exacta, como se muestra en las figuras 1.25 y 1.26:

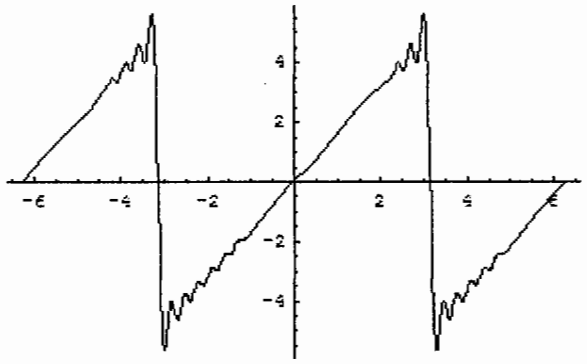
Para una forma de onda diente de sierra impar, utilizando los 5 primeros armónicos se obtiene:

Figura 1. 25 Serie de Fourier diente de sierra n=5



Para n= 20 se obtiene:

Figura 1. 26 Serie de Fourier diente de sierra n=20



Se puede observar la mejora de la forma de onda. La rampa de la función se ve más fina y con menos oscilaciones para n=20 que para n=5. Otra cosa importante que observar en estas graficas son los puntos donde la función es discontinua, en esta parte existen más oscilaciones y con una amplitud máxima cerca de la discontinuidad, a este fenómeno se le conoce como fenómeno de Gibbs.

Las series de Fourier también suelen ser expresadas en su forma exponencial. La ventaja de usar la forma exponencial es que solo requiere una integración en vez de calcular dos, para a_n y b_n. La forma exponencial se escribiría:

$$f(t) = \cdots + C_{-2}e^{-2jwt} + C_{-1}e^{-1jwt} + C_0 + C_1e^{jwt} + C_2e^{2jwt} + \cdots \quad \text{Ecu. 1.38}$$

Donde el coeficiente Cn se obtiene de la siguiente integral:

$$C_n = \frac{1}{T}\int_0^T f(t)e^{jnwt}dwt \quad \text{Ecu. 1.39}$$

También se puede obtener por relación de los coeficientes de la serie trigonométrica. a_n y b_n.

$$a_n = C_{-n} + C_n \quad \text{Ecu. 1.40}$$

$$b_n = j(C_n - C_{-n}) \quad \text{Ecu. 1.41}$$

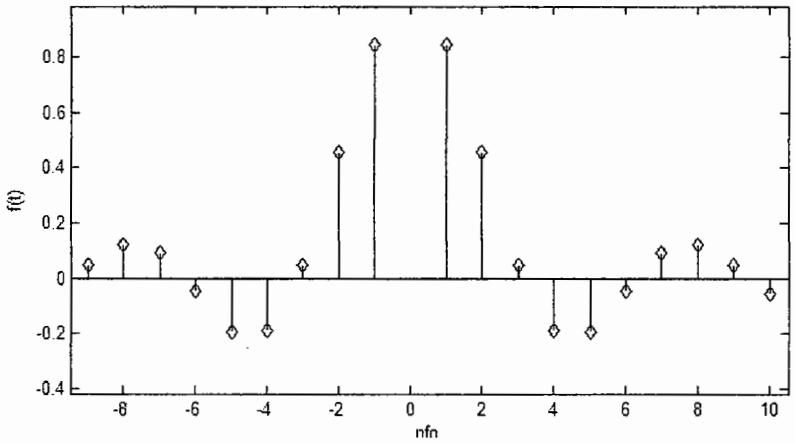
Una propiedad importante es que:

$$C_{-n} = C_n * \quad \text{Ecu. 1.42}$$

Una vez que ya se conozca la serie de Fourier, es de mucha utilidad graficar las magnitudes de las armónicas de la forma de onda, en la escala de la frecuencia. A este

tipo de gráfico se le conoce como líneas de espectro, y esta es la base de un analizador de espectro.

Figura 1. 27 Representación espectral



En la figura 1.27 se muestra una representación de estas líneas de espectro. Se puede apreciar la ayuda que presenta para reconocer las magnitudes debidas a cada armónica. En el caso de la figura 1.27 se indica que las magnitudes de las componentes de cada armónica van decreciendo. Otro punto importante con las series de Fourier, es que presenta una forma sencilla de calcular el valor *RMS (root mean square)* de una señal. El valor *RMS* de una señal consta de senoidales de diferente frecuencia, es igual a la raíz cuadrada de la suma de los valores *RMS* de cada sinodal al cuadrado. Por ejemplo si:

$$i(t) = I_0 + I_1 \cos(w_1 t \pm \emptyset_1) + I_2 \cos(w_2 t \pm \emptyset_2) + \cdots + I_n \cos(w_n t \pm \emptyset_n) \quad \text{Ecu. 1.43}$$

Donde I representa la magnitud de la corriente de las senoidales, de esta manera el *RMS* se obtiene de la siguiente manera:

$$I_{RMS} = \sqrt{{I_0}^2 + \frac{1}{2}{I^2}_{1m} + \frac{1}{2}{I^2}_{2m} + \cdots + \frac{1}{2}{I^2}_{nm}} \qquad \text{Ecu. 1.44}$$

La ecuación 1.44 también se le conoce como teorema de Parseval, y servirá mucho en el transcurso de la elaboración de este proyecto para poder encontrar el valor *RMS* de las señales que se medirán. También se puede calcular la potencia promedio de una serie de Fourier usando la siguiente relación:

$$P_{pro} = P_{dc} + P_{1pro} + P_{2pro} + \cdots \qquad \text{Ecu. 1.45}$$

$$P_{pro} = V_{DC}I_{DC} + V_{1RMS}I_{1RMS}\cos(\theta 1) + V_{2RMS}I_{2RMS}\cos(\theta 2) + \cdots \qquad \text{Ecu. 1.46}$$

1.3.2 Transformada de Fourier

Cuando se trabaja con series de Fourier se debe notar que las señales deben ser periódicas en función al tiempo, estas formas de onda producían unas líneas de espectro discretas, localizadas en las frecuencias o harmónicas de la forma de onda. Ahora se tratará con señales no periódicas, como por ejemplo la función escalón unitario, función de rampa, o la delta entre otras. Se verá que estas funciones no periódicas van a producir una gráfica de espectro continuo.

Se supone una señal no periódica, se puede asumir que tiene un periodo extendiéndose desde -∞ a ∞, entonces para esta señal se formula esta integral:

$$F(w) = \int_{-\infty}^{\infty} f(t)e^{-jwt}dt \qquad \text{Ecu. 1.47}$$

La F(w) se llama la transformada de Fourier o la integral de Fourier.

La función inversa de Fourier es la siguiente:

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(w)e^{jwt}dw \qquad \text{Ecu.1.48}$$

Algunas propiedades importantes de la transformada de Fourier son:

- **Linealidad:** La propiedad de linealidad, establece que la transformada de Fourier de una suma de varias funciones es igual a la suma de las transformadas individuales.

$$f_1(t) + f_2(t) + ... + f_n(t) \leftrightarrow F_1(t) + F_2(t) + ... + F_n(t) \qquad \text{Ecu. 1.49}$$

- **Simetría:** Si F(w) es la transformada de Fourier, entonces:

$$F(t) \leftrightarrow 2\pi f(-w) \qquad \text{Ecu. 1.50}$$

O sea, si en F(w), se remplaza w por t, se obtendrá la relación de la ecuación 1.50.

- **Escalamiento de tiempo:** Si a es una constante real, y si F(w) es la transformada de f(t):

$$f(at) \leftrightarrow \frac{1}{|a|}F\left(\frac{w}{a}\right) \qquad \text{Ecu. 1.51}$$

Remplazando t por at en el dominio del tiempo, se tendrá que remplazar w por w/a en el dominio de la frecuencia y dividir F (w/a) por el valor absoluto de 1/a.

- **Desplazamiento en el tiempo y frecuencia:** Si F(w) es la transformada de f(t):

$$f(t - t_o) \leftrightarrow F(w)e^{-jwto} \qquad \text{Ecu. 1.52}$$

Si se desplaza la función en el tiempo f (t) por una constante to, la magnitud de la transformada de Fourier no cambia, pero se le agrega el termino wto a su ángulo de fase.

Ahora la versión del desplazamiento en la frecuencia:

$$e^{jwot} f(t) \leftrightarrow F(w - w_0) \qquad \text{Ecu. 1.53}$$

Esto es la multiplicación de la función del tiempo f (t) por la exponencial donde wo es una constante, resulta en un desplazamiento en la transformada de Fourier por una cantidad igual a wo.

- **Diferenciación en el tiempo y en la frecuencia:** Siendo F(w) la transformada de f(t):

$$\frac{d^n}{dt^n} f(t) \leftrightarrow (jw)^n F(w) \qquad \text{Ecu. 1.54}$$

Esta es la diferenciación en la frecuencia

$$(-jt)^n f(t) \leftrightarrow \frac{d^n}{dw^n} F(w) \qquad \text{Ecu. 1.55}$$

- **Convolución en el tiempo y frecuencia:** Si F1 (w) es la transformada de Fourier de f1 (t), y F2(w) es la respectiva de f2(t), entonces:

$$f1(t) * f2(t) \leftrightarrow F1(w)F2(w) \qquad \text{Ecu.1.56}$$

La ecuación 1.56 corresponde a que la Convolución de dos funciones en el tiempo es la multiplicación de sus transformadas en el dominio de la frecuencia.

Ahora este teorema cambia un poco para el dominio de la frecuencia, y queda como sigue:

$$f1(t)f2(t) \leftrightarrow \frac{1}{2\pi} F1(w) * F2(w) \qquad \text{Ecu. 1.57}$$

La multiplicación en el dominio del tiempo de dos funciones, corresponde a la Convolución de sus transformadas en el dominio de la frecuencia, divido por una constante 1/2pi.

- **Teorema de Parseval:** Si F(w) es la transformada de Fourier de f(t), entonces:

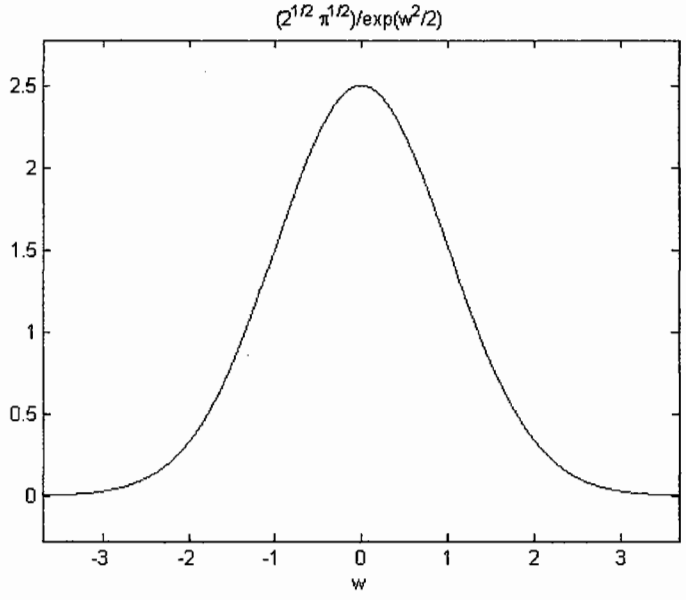
$$\int_{-\infty}^{\infty} |f(t)|^2 dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} |F(w)|^2 dw \qquad \text{Ecu.1.58}$$

Esto es, si la función f(t) representa el voltaje o corriente que pasa a través de una resistencia de 1 ohm, la potencia instantánea absorbida por el resistor es, v^2 *o* i^2. Entonces la integral de la magnitud al cuadrado, representa la energía (en *watt* por segundo o *Joule*) disipada por el resistor. Por esta razón, la integral se le conoce como la energía de la señal. Según la relación dada en la ecuación

1.58, si por algún caso no se conoce la energía de la función del tiempo f(t), pero si se sabe su transformada de Fourier, se puede calcular la energía sin la necesidad de evaluar la transformada inversa de Fourier. Y se mostrará con el siguiente ejemplo utilizando Matlab.

```
%se calculará la siguiente transformada de Fourier
syms t v w ;
ft=exp(-t^2/2);
Fw=fourier(ft)
ezplot(Fw)

Fw =

(2^(1/2)*pi^(1/2))/exp(w^2/2)
```

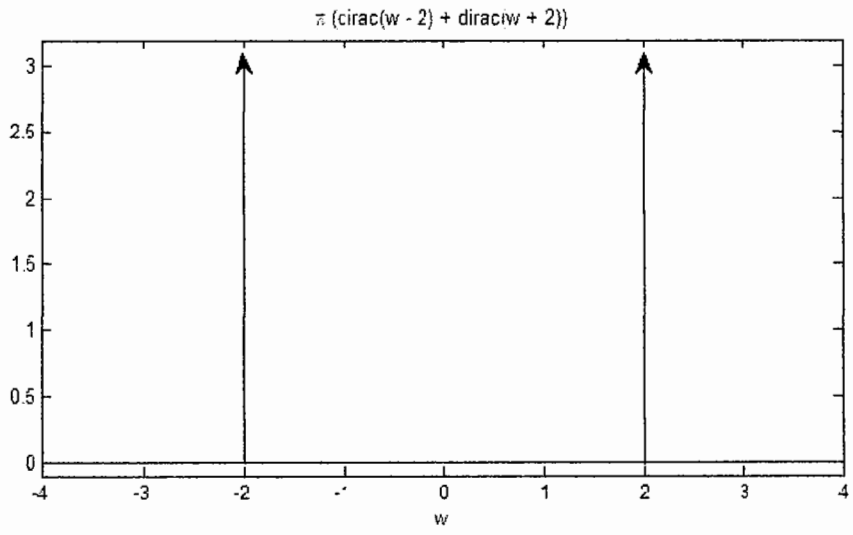
Figura 1. 28 Transformada de Fourier




```
%Esta es una representación de cómo se distribuye la energía de la
%señal
%Ahora a comprobar este resultado y obtener la señal f(t) se
%calcula la transformada inversa usando "ifourier"
```

```
ft=ifourier(Fw)

ft =

1/exp(x^2/2)

%Otro ejemplo:
syms t1 v1 w1 x;

fourier(sym('sin(2*t)'))

ans =

-pi*i*(dirac(w - 2) - dirac(w + 2))
```

Figura 1. 29 Ejemplo Transformada de Fourier del Seno



A continuación se presenta en la tabla 1 los pares de las transformadas de Fourier.

Tabla I Pares de Transformadas de Fourier

$f(t)$	$F(\omega)$
$\delta(t)$	1
$\delta(t-t_0)$	$e^{-j\omega t_0}$
1	$2\pi\delta(\omega)$
$e^{-j\omega t_0}$	$2\pi\delta(\omega-\omega_0)$
$sgn(t)$	$2/(j\omega)$
$u_0(t)$	$\frac{1}{j\omega}+\pi\delta(\omega)$
$\cos\omega_0 t$	$\pi[\delta(\omega-\omega_0)+\delta(\omega+\omega_0)]$
$\sin\omega_0 t$	$j\pi[\delta(\omega-\omega_0)-\delta(\omega+\omega_0)]$
$e^{-at}u_0(t)$ $a>0$	$\frac{1}{j\omega+a}$ $a>0$
$te^{-at}u_0(t)$ $a>0$	$\frac{1}{(j\omega+a)^2}$ $a>0$
$e^{-at}\cos\omega_0 t u_0(t)$ $a>0$	$\frac{j\omega+a}{(j\omega+a)^2+\omega^2}$ $a>0$
$e^{-at}\sin\omega_0 t u_0(t)$ $a>0$	$\frac{\omega}{(j\omega+a)^2+\omega^2}$ $a>0$
$A[u_0(t+T)-u_0(t-T)]$	$2AT\frac{\sin\omega T}{\omega T}$

Fuente: Steven T. Karris
Signals and Sistems with MATLAB Aplications
Pág. 286

1.3.3 Sistemas de tiempo discreto y transformada Z:

La transformada Z realiza la transformación del dominio de señales en tiempo discreto, a otro domino llamado domino Z. se utiliza con señales de tiempo discreto, de la misma manera que las transformadas de Laplace y de Fourier se usan con señales de tiempo continuo. La transformada Z conlleva una descripción de la frecuencia para las señales en tiempo discreto, y forma la base para el diseño de sistemas digitales, como los filtros digitales.

Una razón para introducir la transformada Z es que la transformada de Fourier no converge para todas las secuencias y es muy útil tener una generalización de la transformada de Fourier que comprenda un mayor tipo de señales. Otra ventaja es que la notación es más conveniente en problemas analíticos.

La transformada de Fourier de una secuencia x[n] se define como

$$X(e^{jw}) = \sum_{n=-\infty}^{\infty} x[n]e^{-jwn}$$

La transformada Z de la misma secuencia es:

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}$$

La ecuación es, en general, una suma infinita o una serie infinita de potencias, con z siendo una variable compleja. El operador de la transformada se define como:

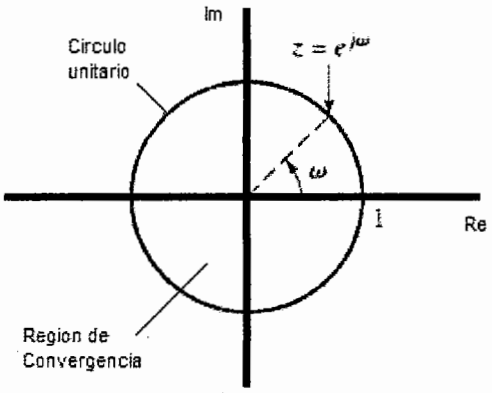
$$Z\{x[n]\} = \sum_{n=-\infty}^{\infty} x[n]z^{-n} = X(z)$$

Con esta interpretación, se puede obtener el cambio del dominio de la secuencia x[n]. Otro punto que hay que resaltar es la relación que existe entre la transformada de Fourier y la Z. Si se reemplaza la variable compleja z con la variable compleja e^{jw}, entonces la transformada Z sería la transformada de Fourier. Si se expresa la variable z en forma polar como $z = re^{jw}$, para el caso en que $|z| = 1$, la transformada z corresponde a la transformada de Fourier.

Para obtener una representación gráfica que ayude a describir y a interpretar la variable compleja, se usa el plano complejo z. En este plano, el contorno corresponde a $|z| = 1$ que es el círculo unitario. La transformada Z evaluada en el contorno representa la transformada de Fourier. Este círculo unitario comprende la región de convergencia, esto es, si la secuencia es absolutamente sumable, la transformada z converge. Debido a la multiplicación de la secuencia por un exponente real r^{-n}, es posible que la transformada Z converja aun si la transformada de Fourier no lo hace.

La siguiente gráfica muestra el plano complejo de Z:

Figura 1. 30 Circulo unitario en el plano complejo Z



La transformada Z es más útil cuando la suma infinita es expresada en una forma abreviada. Entre las más importantes y útiles transformadas Z son aquellas para las que X(z) es una función racional dentro de la región de convergencia.

$$X(z) = P(z)/Q(z)$$

Donde P(z) y Q(z) con polinomios de z. Los valores para los cuales X(z) =0 son llamados ceros y para los valores que hacen X(z) sea infinito se llaman polos.

A continuación se presenta una tabla de los pares de transformadas Z.

Tabla II Pares de Transformadas de Z

$f[n]$	$F(z)$
$\delta[n]$	1
$\delta[n-m]$	z^{-m}
$a^n u_0[n]$	$\frac{z}{z-a} \quad \lvert z\rvert > a$
$u_0[n]$	$\frac{z}{z-1} \quad \lvert z\rvert > 1$
$(e^{-naT})u_0[n]$	$\frac{z}{z-e^{-aT}} \quad \lvert e^{-aT}z^{-1}\rvert < 1$
$(\cos naT)u_0[n]$	$\frac{z^2 - z\cos aT}{z^2 - 2z\cos aT + 1} \quad \lvert z\rvert > 1$
$(\sin naT)u_0[n]$	$\frac{z\sin aT}{z^2 - 2z\cos aT + 1} \quad \lvert z\rvert > 1$
$(a^n \cos naT)u_0[n]$	$\frac{z^2 - az\cos aT}{z^2 - 2az\cos aT + a^2} \quad \lvert z\rvert > a$
$(a^n \sin naT)u_0[n]$	$\frac{az\sin aT}{z^2 - 2az\cos aT + a^2} \quad \lvert z\rvert > a$
$u_0[n] - u_0[n-m]$	$\frac{z^m - 1}{z^{m-1}(z-1)}$
$nu_0[n]$	$z/(z-1)^2$
$n^2 u_0[n]$	$z(z+1)/(z-1)^3$
$[n+1]u_0[n]$	$z^2/(z-1)^2$
$a^n n u_0[n]$	$(az)/(z-a)^2$
$a^n n^2 u_0[n]$	$az(z+a)/(z-a)^3$
$a^n n[n-1]u_0[n]$	$2az^2/(z-a)^3$

Fuente: Steven T. Karris
Signals and Sistems with MATLAB Aplications, *Pág. 347*

Un ejemplo de la transformada Z utilizando Matlab:

Hallar la transformada z de: $f[n] = 2(0.5)^n - 9(0.75)^n + 8$

Calculo de la transformada Z. Definir la función para luego calcular su transformada

```
syms n z
fn=2*(0.5)^n-9*(0.75)^n+8;
Fzz=ztrans(fn,n,z);
Fz=simple(Fzz)

Fz =

(8*z^3)/(8*z^3 - 18*z^2 + 13*z - 3)
```

Ahora se calcula su inversa para verificar

```
iztrans(Fz)

ans =

2*(1/2)^n - 9*(3/4)^n + 8
```

Ahora se calcula la respuesta al impulso de $F(z) = \frac{z^3}{(z-1)(z^2+2z+2)}$

Este ejemplo muestra el uso de las funciones de Matlab para calcular transformadas Z Primero se expande el denominador de F(z)

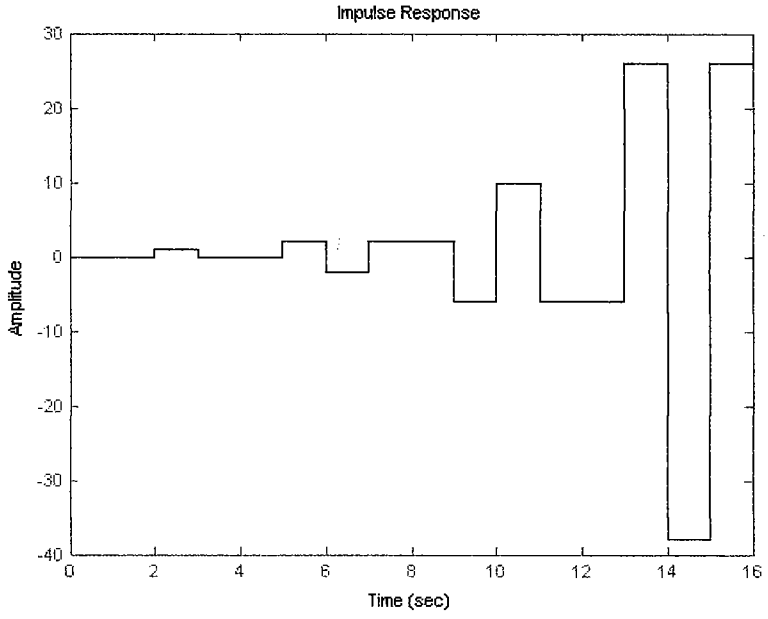
```
syms n z
collect((z-1)*(z^2+2*z+2))

ans =

z^3 + z^2 - 2
```

Se calcula ahora la respuesta al impulso

```
num=[0 0 1 1];
den=[1 1 0 -2];
fn=dimpulse(num,den,11)
dimpulse(num,den,16)
```

Figura 1. 31 Respuesta al impulso

```
fn =

     0
     0
     1
     0
     0
     2
    -2
     2
     2
    -6
    10
```

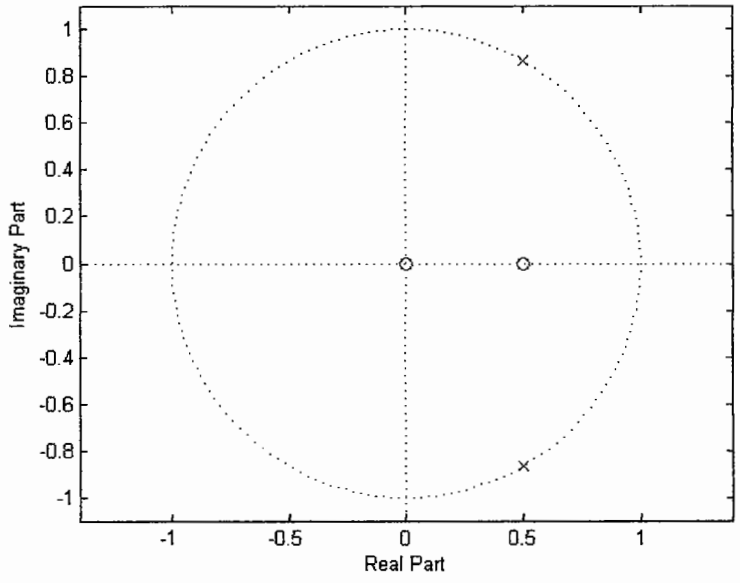


La representación gráfica de las transformadas Z se hace en el plano Z. En este plano se localizan los ceros y polos de las funciones de transferencia $H(z) = \frac{A(z)}{B(z)}$, donde A(z) es un polinomio y de la misma manera B(z). La gráfica lo hace alrededor de una circunferencia unitaria, sus ejes son parte real contra parte imaginario y de esta manera se localizan los puntos.

A continuación se presenta un ejemplo de cómo Matlab grafica una función H(z):

```
syms n
h=cos(pi/3*n);
H=ztrans(h,z);
[n,d]=numden(H);
N=collect(n);
D=collect(d);
Num=sym2poly(N);
Den=sym2poly(D);
zplane(Num,Den)
```

Figura 1. 32 Representación plano Z



2. PROCESAMIENTO DIGITAL DE SEÑALES

En el capítulo anterior se mostró como representar señales eléctricas en una manera matemática de modo que se pueda conocer un modelo de cómo se comporta la señal. Esto será útil para pasar de la parte teórica a la práctica. En este capítulo se mostrará la manera en que las señales reales son analizadas, obtenidas y transformadas en el modelo del cual se ha hablado, para que pueda ser procesada y utilizada.

El procesamiento digital de señales empieza con la adquisición de la señal, que normalmente está en forma analógica o es una señal continua, esta forma de onda puede convertirse en una señal digital, que es lo que en este trabajo se está buscando, una vez se tenga la señal en una forma digital estará lista para procesarla, analizarla, transmitirla, etc.

En este capítulo se desarrollan dos temas importantes, estos son el proceso del trabajo con señales y el procesador de señales como dispositivo. Al finalizar este capítulo se hará un ejemplo de simulación de un *DSP* usando Simulink de Matlab.

2.1 Procesamiento de señales

Para comenzar con lo que es el procesamiento de señales, primero se debe de tener una señal que se va a analizar; cambiar algunas de sus características, como la amplitud y la fase; esto se logra a través de modular, filtrar, mezclar o darle diferentes efectos a la señal, entre otras cosas que se pueden hacer con una señal eléctrica.

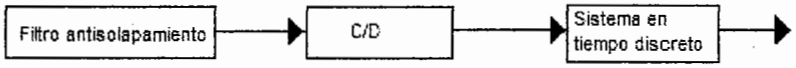
La manera en que se obtiene esta señal, para procesarla, es por medio de un dispositivo transductor, como un micrófono, que captura la señal para luego muestrearla, y posteriormente convertirla de una forma de onda analógica a una digital.

2.1.1 Filtrado Previo

La mayor parte de señales que se van a procesar no tendrán una banda limitada, haciendo que la frecuencia de muestreo pueda ser muy grande o variante. Para mejorar la forma de escoger una frecuencia de muestreo es recomendable filtrar para limitar la señal. Aun si las señales tienen banda limitada estas pueden estar contaminadas con ruido, el filtrado previo ayudará también a remover las componentes del ruido de frecuencias altas.

Para evitar el solapamiento, hay que forzar a la señal que sea de banda limitada a las frecuencias por debajo de la frecuencia de *Nyquist*. Esto se logra usando un filtro pasa bajo, a este filtro se le conoce como filtro anti solapamiento.

Figura 2. 1 Uso de filtro



Este filtrado se hace previo a la conversión digital. La respuesta en frecuencia del filtro debería ser:

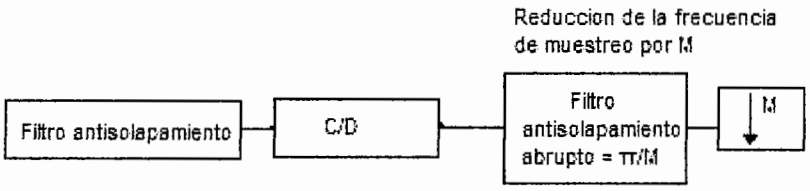
$$H(j\Omega) = \begin{cases} 1, & |\Omega| < \Omega c < \pi/T \\ 0, & |\Omega| > \Omega c \end{cases} \qquad \text{Ecu. 2.1}$$

Este filtro necesita tener una respuesta muy selectiva. Esto funciona bien para la ecuación pero en la realidad estos tipos de filtros son muy difíciles de realizar además de que tiene un costo elevado, por otra parte si el sistema va a utilizar una frecuencia de muestreo variable, se necesitaría filtros ajustables. Asimismo, los filtros con caídas bruscas en la frecuencia de corte tienen una respuesta en frecuencia altamente no lineal, en especial en la banda de paso. Por estas razones se prefiere remover los filtros analógicos, que no son tan flexibles como los digitales, o simplificar sus requerimientos de diseño.

Debido a las desventajas antes dichas, un método alternativo puede ser de ayuda. Este consiste en aplicar primero un filtro anti solapamiento muy simple con una banda

de transición gradual y con una atenuación importante en MΩn. Seguido de esto se convierte la señal continua a una discreta, en la figura se muestra en el bloque como C/D, a una frecuencia mucho mayor que 2Ωn. Después se realiza una reducción de la frecuencia de muestreo por un factor de M que incluye un filtrado anti solapamiento con caída abrupta en la frecuencia de corte. Lo importante de hacer esto es que el filtrado se realiza en el domino de tiempo discreto. Este procedimiento en tiempo discreto posterior se puede realizar a frecuencia de muestreo baja, con lo que se reduce el coste computacional, el cual es uno de los objetivos de este filtrado. La figura 2.2 muestra como se realiza este procedimiento.

Figura 2. 2 Filtro en el sistema



2.1.2 Teorema de Muestreo

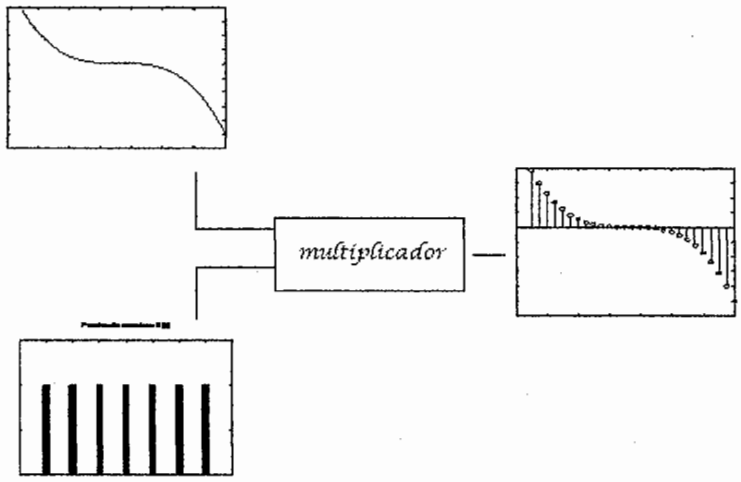
Para comenzar se debe conocer el principio fundamental de las comunicaciones digitales:

Siendo m(t) es una señal que es limitada, de manera que su componente de mayor frecuencia en el espectro es fm, al determinarse los valores de m(t) a intervalos regulares separados por espacios de tiempos Ts≤1/2fm, o sea, la señal es muestrea periódicamente cada Ts segundos. Entonces se tendrá las muestras m(nTs); donde n es un número entero; determinando de forma única la señal, y esta puede ser reconstruida a

partir de estas muestras sin mostrar distorsión. El tiempo Ts se conoce como *tiempo de muestreo*. Es de notar lo siguiente:

"El teorema requiere que la razón de muestreo sea lo suficiente rápida hasta que al menos dos muestras sean tomadas durante el curso del periodo correspondiente al componente espectral de mayor frecuencia."

Figura 2. 3 Teorema de muestreo



En la figura 2.3 se ve la señal m(t) que va a ser muestreada por un tren de pulsos de amplitud unitaria S(t), estas son aplicadas a un multiplicador lo que da la señal resultante en la salida S(t)m(t).

Cuando un pulso ocurre, la salida del multiplicador tiene el mismo valor que tendría m(t), para cuando los pulsos son ceros también lo es la salida.

La expansión de Fourier de la señal S(t) es, con un periodo de Ts:

$$S(t) = \frac{dt}{Ts} + \frac{2dt}{Ts}(\cos(2\pi\, t/Ts) + \cos(4\pi\, t/Ts) + \cdots) \qquad \text{Ecu. 2.2}$$

Para el caso en que Ts=1/2fm el producto de S(t)m(t) es:

$$S(t)m(t) = \frac{dt}{Ts}m(t) + \frac{dt}{Ts}(2m(t)\cos(2\pi\,(2fm)t) + 2\text{m(t)}\cos(4\pi\,(2fm)t) + \ldots \text{ Ecu. 2.3}$$

De la ecuación 2.2 se puede observar que el primer término de la serie es la señal m(t) únicamente multiplicada por un factor constante. También el segundo término es el producto de m(t) y una senoidal de frecuencia 2fm . Este producto produce una señal de doble banda y portadora suprimida con una portadora de 2fm, los siguientes términos son con portadoras de 4fm, 6fm, etc.

El espectro del primer término en la ecuación 2.2 se extiende desde 0 hasta fm, el espectro del segundo término es simétrico con respecto a la frecuencia 2fm y se extiende desde fm hasta 3fm.

En el caso en que fs = 1/Ts es mayor que 2fm existe una separación entre el límite superior fm del espectro de señal y el límite inferior del siguiente término. De aquí se puede filtrar la salida para obtener la señal de entrada.

Por otro lado si fs < 2fm ocurre un traspaso entre el espectro de m(t) y el espectro de la señal muestreada. De esta manera ningún filtro posible podrá recuperar la señal.

La conclusión de esta discusión es entonces: La señal muestreada puede ser recuperada exactamente cuando Ts ≤ 1/2fm. Así la mínima razón de muestreo debe ser 2fm. A esta razón se conoce como la razón de *Nyquist*.

Otro fenómeno que ocurre cuando se muestrea a una frecuencia que no sea la de *Nyquist* se conoce como "*aliasing*", que produce que las señales cambien de frecuencia cuando se muestrea.

2.1.3 Cuantificación

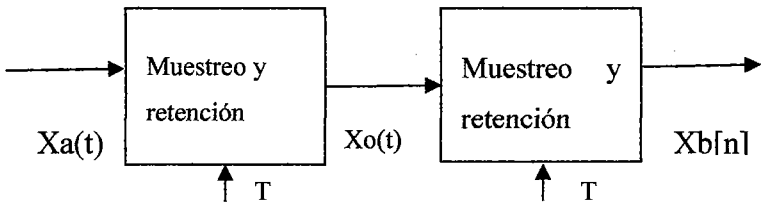
Cuando se cuantifica una señal m(t), se crea una nueva señal mq(t) que es una aproximación de m(t). Esta señal cuantificada m(t) tiene el mérito de que es separable del ruido aditivo. La señal cuantificada mq(t) es generada de la siguiente forma:

Al momento que m(t) esté en el rango Δ_0, la señal mq(t) se mantiene en el nivel constante m_0; cuanto m(t) está en el rango de Δ_1, mq(t) se encuentra en el nivel m_1; y de esa manera continúa. De esta forma la señal siempre se encontrará en un nivel.

2.1.4 Conversión analógico-digital (A/D)

Esta sección explicará cómo se convierte una señal en tiempo continuo (analógica) en una señal digital, o sea una secuencia de muestras de precisiones finitas o cuantificadas.

Figura 2. 4 Conversión analógica digital



El convertidor A/D es un dispositivo físico que convierte una amplitud de tensión o corriente a su entrada en un código binario que representa la amplitud cuantificada más cercana a la amplitud de entrada. La conversión está controlada por un reloj externo, el convertidor puede iniciar y completar una conversión A/D cada T segundos. A pesar de esto, la conversión no es instantánea, y por este motivo los sistemas A/D incluyen generalmente un sistema de muestreo y retención como la que se muestra en la figura 2.4. El sistema de muestreo y retención ideal tiene como salida:

$$x_o(t) = \sum_{n=-\infty}^{\infty} x[n]h_0(t - nT) \qquad \text{Ecu.2.4}$$

Donde x[n]= xa(nT) son las muestras ideales de xa(t) y ho(t) es la respuesta al impulso del sistema de retención de orden cero, o sea:

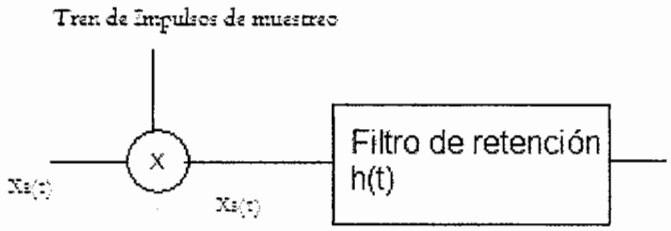
$$h_0(t) = \begin{cases} 1, & 0 < t < T \\ 0, en\ otro\ caso \end{cases} \qquad \text{Ecu.2.5}$$

La ecuación 2.4 también se puede escribir como una Convolución:

$$x_o(t) = \sum_{n=-\infty}^{\infty} x(nT)\delta(t - nT) \qquad \text{Ecu.2.6}$$

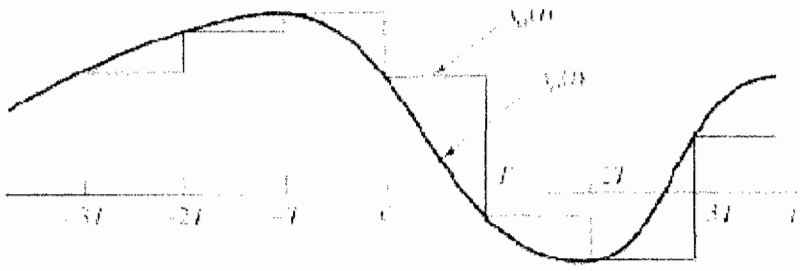
Esto es semejante a una modulación de un tren de impulsos seguido de un filtrado lineal con el sistema de retención de orden cero, como se muestra en la figura 2.25.

Figura 2. 5 Muestreo



La salida del sistema de retención de orden cero tiene forma de escalera. Como se muestra en la figura 2.6.

Figura 2. 6 Señal muestreada

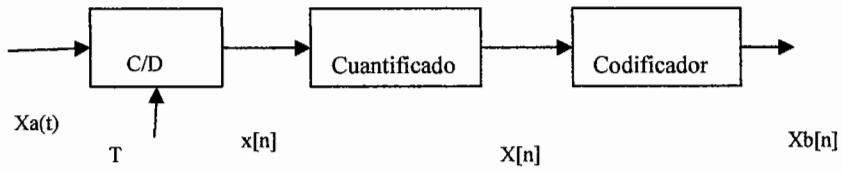


Fuente: Oppenheim shafer & buck
Discrete-time digital signal processing
Pág. 189

Los valores de las muestras se mantienen constantes durante el periodo de muestreo de T segundos. El diseño se hace de manera que se muestree la señal xa(t) lo más instantáneamente posible y mantiene el valor de la muestra tan cercano a una constante como sea posible hasta que la siguiente muestra sea tomada.

Con esto aún no termina el A/D, falta una parte en la que se cuantifica la señal y se codifica como se muestra en la figura 2.7.

Figura 2. 7 Convertidor A/D con codificador

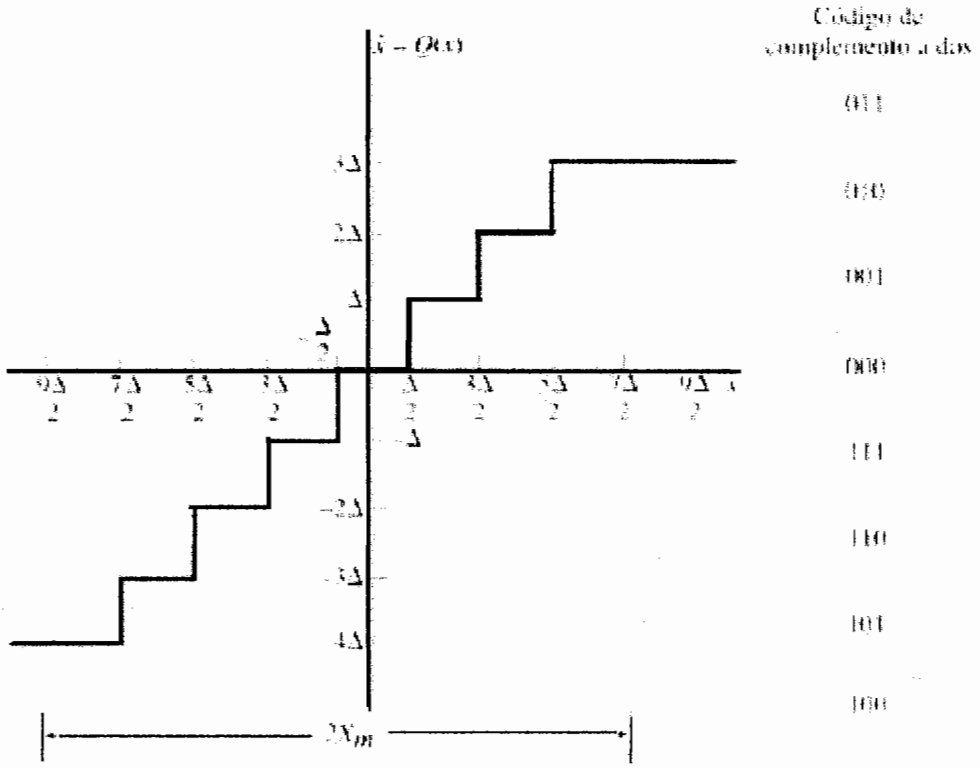


El cuantificador es un sistema no lineal cuyo propósito es transformar la muestra de entrada x[n] en un valor discreto de conjunto finito de valores preestablecidos. Esto se representa como:

$$\hat{X}[n] = Q(x[n]) \qquad \text{Ecu.2.7}$$

$\hat{X}[n]$ Será la muestra cuantificada. Los niveles de cuantificación se pueden definir para que sean uniformes o no uniformes. La siguiente figura muestra la característica de transferencia de un cuantificador uniforme típico, en el que los valores de las muestras se redondean hasta el nivel de cuantificación más próximo.

Figura 2. 8 Cuantificación y codificación



Fuente: Oppenheim shafer & buck
Discrete-time digital signal processing, Pág. 190

Por lo general el número de niveles de cuantificación será una potencia de 2, en la figura 2.8 también se muestran la manera en que será codificado cada nivel de cuantificación. En la figura, se indica cual será el código binario de salida, este consta de 3 bits y un total de 8 códigos diferentes.

La relación entre las palabras del código y los niveles de la señal cuantificada dependen del parámetro Xm. En general este parámetro se denomina margen dinámico del convertidor. Esto indica el tamaño de paso:

$$\Delta = \frac{2Xm}{2^{B+1}} = \frac{Xm}{2^{B}} \qquad \text{Ecu.2.8}$$

Los niveles de cuantificación más pequeños (±Δ), corresponden al bit menos significativo de la palabra del código binario. Además, la relación numérica entre las palabras del código y las muestras cuantificadas es:

$$\hat{x}[n] = Xm\hat{x}_B[n] \qquad \text{Ecu.2.9}$$

Sobre esta forma, las muestras codificadas en binario son directamente proporcionales a las muestras cuantificadas, y por lo tanto, se pueden usar como una representación numérica de las amplitudes de las muestras. Existen varias formas de asignar códigos a los niveles de cuantificación, dependiendo del propósito que se le va a dar.

2.1.5 Error de Cuantificación

Es de notar que a la hora de cuantificar una señal, esta difiere de la señal original en una manera aleatoria. A esta diferencia se le conoce como error de cuantificación.

La representación estadística de los errores de cuantificación se basa en lo siguiente:

1. La secuencia de error e[n] pertenece a un proceso aleatorio estacionario
2. La secuencia de error no tiene correlación con la secuencia x[n]
3. Las variables aleatorias del proceso de error es un proceso de ruido blanco.
4. La distribución de probabilidad del proceso de error es uniforme en el intervalo del error de cuantificación.

Un análisis del error cuadrático medio de cuantificación $\overline{e^2}$, donde e es la diferencia entre la señal original y la cuantificada, se obtiene dividiendo el rango total pico a pico de la señal de mensaje m(t) en M intervalos equivalentes de voltaje , cada uno de magnitud S voltios. Al centro de cada intervalo de voltaje se localiza un nivel de cuantificación m1, m2.... El error será $e = m(t) - mk$.

Si f(m)dm es la probabilidad que m(t) se encuentre in el rango de voltaje $m - dm/2$ a $m + dm/2$, entonces el error cuadrático medio de cuantificación es:

$$\overline{e^2} = \int_{m1-S/2}^{m1+S/2} f(m)(m-m1)^2 dm + \int_{m2-S/2}^{m2+S/2} f(m)(m-m2)^2 dm + \cdots$$

Ecu.2.10

Normalmente la densidad de probabilidad f(m) de la señal de mensaje m(t) no será una constante. Sin embargo, suponiendo que el número M de cuantificación es grande, de manera que el tamaño de S es pequeño en comparación con el rango pico a pico de la señal de mensaje, en este caso se puede hacer la aproximación que f(m) es constante dentro de cada rango de cuantificación. Entonces el primer término de la ecuación 2.10 se escribe $f(m) = f^1$, una constante. En el segundo término se procede de igual manera $f(m) = f^2$, etc. Ahora se quitan estos valores constantes de la integral y con la sustitución $x \equiv m - mk$, se obtiene:

$$\overline{e^2} = (f^1 + f^2 + \cdots) \int_{-\frac{S}{2}}^{\frac{S}{2}} x^2 dx = (f^1 + f^2 + \cdots) \frac{S^3}{12} \qquad \text{Ecu.2.11.a}$$

$$= (f^1 S + f^2 S + \cdots) \frac{S^2}{12} \qquad \text{Ecu.2.11.b}$$

Ahora $f^1 S$ es la probabilidad que la señal de voltaje m(t) esté en el primer rango de cuantificación y $f^2 S$ es la probabilidad que se encuentre en el segundo rango y así sucesivamente. La suma de los términos en los paréntesis en la ecuación 2.11.b tiene un valor igual a la unidad, por lo tanto, el error cuadrático medio de cuantificación es:

$$\overline{e^2} = \frac{S^2}{12} \qquad \text{Ecu.2.12}$$

En el caso general para un cuantificador de (B+1) bits con margen dinámico Xm, la varianza o potencia del ruido es:

$$\overline{e^2} = \frac{2^{-2B} X_m^2}{12} \qquad \text{Ecu.2.13}$$

Este resultado será de ayuda para encontrar la degradación de la señal cuando se contamina con ruido aditivo, esto se conoce como relación señal-ruido y se define como el cociente entre la varianza (potencia) de la señal y la varianza del ruido. Si se expresa en decibelios (dB), la relación señal-ruido para un cuantificador de (B+1) bits es:

$$SNR = 10\log\left(\frac{\sigma_x^2}{\overline{e^2}}\right) = 10\log\left(\frac{12\,2^{2B}\sigma_x^2}{X_m^2}\right)$$

$$= 6.02B + 10.8 - 20\log\left(\frac{X_m}{\sigma_x}\right) \qquad \text{Ecu2.14}$$

Estos resultados dejan ver que la relación señal-ruido se incrementa aproximadamente 6 dB por cada bit que se añade a la longitud de palabra de las muestras cuantificadas, es decir, al doblar el número de niveles de cuantificación.

Ahora analizando el término:

$$-20\log\left(\frac{X_m}{\sigma_x}\right) \qquad \text{Ecu.2.15}$$

Con Xm siendo un parámetro del cuantificador, este tendrá un valor fijo en un sistema real concreto. La cantidad $\sigma_x$es el valor rms de la amplitud de la señal, necesariamente inferior a la amplitud de pico de la señal. Si σ_x es demasiado grande, la amplitud de pico de la señal será mayor que el margen dinámico Xm del convertidor A/D y en tal caso se produciría una distorsión severa. Ahora lo interesante sucede cuando σ_x es demasiado pequeño, el término correspondiente a la ecuación 2.15 se hace grande y negativo, con esto la relación señal-ruido de la ecuación 2.14 decrece.

2.2 Filtros Digitales

Un filtro digital es un sistema discreto utilizado para extraer características desde el dominio de la frecuencia sobre señales muestreadas. La operación de filtrado se realiza por medio de cálculos directos con las señales muestreadas. Las ventajas que presentan los filtros digitales frente a los analógicos son las siguientes:

- Respuesta dinámica: El ancho de banda del filtro digital está limitado por la frecuencia de muestreo, mientras que en los filtros analógicos con componentes activos suelen estar restringidos por los amplificadores operacionales.
- Intervalo dinámico: En filtros analógicos aparecen parámetros que limitan por abajo el rango y se saturan con la alimentación. En cambio en los filtros digitales es fijado por el número de bits que representa la secuencia, y el límite inferior por el ruido de cuantificación y los errores de redondeo
- Conmutabilidad: Si los parámetros de un filtro se conservan en registros, los contenidos de dichos registros pueden ser modificados a voluntad. Además, estos filtros se pueden conmutar, pudiéndose multiplexar en el tiempo para procesar varias entradas a la vez.
- Adaptabilidad: Un filtro digital puede ser implementado en soporte físico *(hardware)* o mediante un programa de ordenador *(software)*.
- Ausencia de problemas de componentes: Los parámetros de los filtros se representan por medio de números binarios y no derivan con el tiempo. Al no haber componentes, no hay problemas de tolerancia o deriva de componentes, y ningún otro problema asociado con un comportamiento no ideal de resistencias, condensadores, bobinas o amplificadores. Tampoco existen problemas de impedancia de entrada ni salida, ni efectos de adaptación de impedancias entre etapas.

Una distinción fundamental en los sistemas discretos dinámicos lineales e invariantes, y en particular en los filtros digitales, es la duración de la respuesta ante el impulso. Existen sistemas de respuesta de pulso finito o no recursivo *(FIR, finite impulse response),* y de sistemas de respuesta infinita o recursivo *(IIR, infinite impulse response).*

Partiendo de la ecuación en diferencias que modela el comportamiento dinámico de estos sistemas se tiene:

$$Y_k = a_1 y_{k-1} + a_2 y_{k-2} + \cdots + a_n y_{k-n} + b_0 x_{k-0} + b_1 x_{k-1} + \cdots + b_m x_{k-m}$$

Ecu.2.16

En el caso de tener todos los coeficientes a_i iguales a cero se tendrá un filtro *FIR*, con lo que quedará la ecuación reducida a:

$$Y_k = b_0 x_{k-0} + b_1 x_{k-1} + \cdots + b_m x_{k-m} \qquad \text{Ecu.2.17}$$

Donde m es el orden del filtro. Esta ecuación tiene su transformada z igual a:

$$G(z) = b_0 + b_1 z^{-1} + \cdots + b_m z^{-m} \qquad \text{Ecu.2.18}$$

En estos filtros el valor de la secuencia de salida sólo depende de un número finito de valores de la secuencia de entrada. Además también se desprende la carencia de polos en la función de transferencia. En cambio, las expresiones de los filtros recursivos corresponden a:

$$Y_k = a_1 y_{k-1} + a_2 y_{k-2} + \cdots + a_n y_{k-n} + b_0 x_{k-0} + b_1 x_{k-1} + \cdots + b_m x_{k-m}$$

Ecu.2.19

La transformada Z de la ecuación 2.19 es:

$$G(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_m z^{-m}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n}}$$ Ecu.2.20

En estos casos, la secuencia de salida depende tanto de la entrada como de la salida. De estas ecuaciones se deducen las siguientes propiedades; Primero, la secuencia de ponderación es infinita para los filtros *IIR*, aun teniendo un número finito de coeficientes, mientras la respuesta al impulso de un filtro no recursivo es siempre finita e igual al orden del filtro. En segundo lugar, los filtros *FIR* son siempre estables, esto es, la secuencia de salida tiene todos sus valores acotados. No es el caso de filtros recursivos, su estabilidad depende de la función de transferencia, por lo que habrá de utilizar alguno de los procedimientos algebraicos, para analizar su estabilidad. Tercera, cualquier filtro recursivo puede ser reemplazado por otro no recursivo con infinitos coeficientes, sus valores vendrán dados por la secuencia de ponderación del *IIR*. De manera inversa no se cumple.

Algunas de las consideraciones que se deben tomar a la hora de procesar señales es que la relación de fase es importante, y a menudo se debe evitar perturbarlas con el filtrado. Por esta razón existen filtros con desfase nulo, a estos filtros se les llaman de fase lineal o no dispersivos.

2.2.1 Construcción de Filtros no Recursivos (*FIR*)

Los filtros no recursivos tienen ventajas muy interesantes que les hacen ser ampliamente utilizados en múltiples aplicaciones. La característica más destacable es su facilidad de diseño para conseguir una respuesta en frecuencias de fase lineal, esto es, la señal que pase a través de él no será distorsionada. Los *FIR* son por su propia constitución estables, no habiendo problemas en su diseño o fase de implementación.

Su mayor desventaja está en que, para cumplir con las especificaciones dadas, El filtro *FIR* necesita un orden mayor al que se obtuviera con un *IIR*, implicando programas más largos o circuitos mayores.

Los filtros digitales suelen ser caracterizados en términos de rangos de frecuencia, tanto de la banda pasante como de la supresora. Sus respuestas en la frecuencia son periódicas con la frecuencia de Nyquist, w_N, por lo que sólo se considerará el intervalo $[-w_N, w_N]$.

El modelo matemático que caracteriza la respuesta en frecuencia de un filtro típico pasa bajo es:

$$G(w) = \begin{cases} e^{-j\lambda w} & |w| \leq w_c \\ 0 \; En\ otro\ caso \end{cases} \qquad \text{Ecu.2.21}$$

La notación w_c indica la frecuencia de corte, donde λ es el desfase, y ws la frecuencia de muestreo. Si se aplica la transformada inversa a la ecuación 2.21 se obtiene:

$$g_n = \frac{1}{2\pi}\int_{-w_c}^{w_c} G(w)e^{jnwt}dw = \frac{2w_c}{w_s}\frac{\sin\left[(n-\lambda)w_cT\right]}{(n-\lambda)w_cT} \quad \text{n=...-2,-1,0,1,2...}$$

Ecu.2.22

Una respuesta en frecuencia del filtro se representa de la siguiente manera:

$$G(w) = e^{-jNwT}\{g_N + 2\sum_{i=0}^{N-1} g_i \cos\left[(N-i)wT\right]\} \qquad \text{Ecu.2.23}$$

El desfase introducido por el filtro es –NwT.

Este mismo análisis se puede utilizar para diseñar otro tipo de filtros, únicamente es necesario utilizar una transformación como se muestra en la siguiente tabla.

Tabla III Conversión de filtros digitales

Conversión	Transformación	Parámetros
A paso alto	$g_{n(PA)} = (-1)^n g_{n(PB)}$	$(w_c)_{PA} = (w_N) - (w_c)_{PB}$
A pasa banda	$g_{n(PBANDA)} = (2\cos(nw_0T))g_{n(PB)}$	$w_0 = frecuencia\ central$ $w_1 = w_0 - (w_c)_{PB}$ $w_2 = (w_c)_{PB} - w_0$

Algo que hay que mencionar es que, aunque el orden del filtro sea elevado, los rizados tanto en la banda pasante como en la supresora se mantienen, haciéndose mayores las oscilaciones en las zonas de transición entre las bandas. Además, la atenuación en la banda no pasante no es cero y la transición entre las bandas no es abrupta. A este fenómeno se llama efecto de *Gibbs*.

Sin embargo, la aplicación de ciertas funciones ventanas permiten aliviar este efecto no deseado. Si se quiere disminuir las oscilaciones, la respuesta del pulso infinito original debe ser multiplicado por una función ventana que no sea un pulso rectangular puro, de manera que la nueva secuencia de ponderación *gn'* quedará como:

$$g_n' = g_n w_n \qquad \text{Ecu.2.24}$$

Donde $g_n$es la función de la ecuación 2.22 y w_n una función ventana definida por:

$$w_n = \begin{cases} 1 & |n| \leq N \\ 0 & |n| > N \end{cases} \qquad \text{Ecu.2.25}$$

En el cual N es el número de la secuencia de transición entre la banda pasante y la supresora, aunque en la práctica esta ventana causaría también el efecto *Gibbs*. Para evitarlo hay diferentes tipos de ventanas, como las que siguen:

$$w_n = \begin{cases} \alpha + (1-\alpha)\cos\left(\frac{\pi n}{N}\right) & |n| \leq N \\ 0 & |n| > N \end{cases} \qquad \text{Ecu.2.26}$$

En el cual si α =0.5, se le conoce como ventana de *von Hann,* y cuando $\alpha = 0.54$, se denomina ventana de *Hamming.*

La ventana de *Blackman* se define como:

$$w_n = \begin{cases} 0.42 + 0.5\cos\left(\frac{\pi n}{N}\right) + 0.08\cos\left(\frac{2\pi n}{N}\right) & |n| \leq N \\ 0 & |n| > N \end{cases} \qquad \text{Ecu.2.27}$$

Las características de esta ventana es que reduce el rizado en comparación con las dos anteriores, pero la transición entre bandas es muy suave. Si lo que se busca es una relación entre el rizado y la ruptura abrupta, es preferible usar una ventana de *Kaiser* definida como:

$$w_n = \begin{cases} I_0(\beta)/I_0(\alpha) & |n| \leq N \\ 0 & |n| > N \end{cases}$$ Ecu.2.28

Siendo α un parámetro y

$$\beta = \alpha\sqrt{(1-\left(\frac{n}{N}\right)^2)}$$ Ecu.2.29

Donde I(x) es una función de *Bessel* de orden cero definida por la serie:

$$I_0(x) = 1 + \sum_{k=1}^{\infty}(\frac{1}{k!}(\frac{x}{2})^k)^2$$ Ecu.2.30

A medida de tener mayor β se disminuye el rizado pero también disminuye la pendiente de transición entre la banda pasante y la supresora.

2.2.2 Construcción de filtros recursivos (*IIR*)

La ventaja de los filtros *IIR* respecto a los *FIR* es la de tener un menor orden del filtro para iguales especificaciones de diseño. Aunque la desventaja, es la falta de desfase lineal introducido por el filtro, así como la necesidad de realizar estudios de estabilidad, pues ésta no está garantizada en el diseño.

Los filtros no recursivos basados en modelos conocidos, como por ejemplo: *Butterworth*, *Chebyschev*, etc., pueden ser diseñados por varios métodos, basado, siendo

el más común el basado en las transformaciones bilineales. Este procedimiento requiere del conocimiento de la función de transferencia en el dominio *s* del filtro a diseñar. Los coeficientes del filtro en el dominio *s* son transformados a uno equivalente en el dominio *z*, y los coeficientes discretos formarán el filtro *IIR* en tiempo discreto.

El método de diseño de filtros *IIR* basados en transformaciones bilineales tiene el siguiente procedimiento:

1. Definir las características del filtro digital $w_{d1}, w_{d2}, \dots, w_{dk}$
2. Realizar la operación de la siguiente función $w_{ai} = \frac{2}{T}\tan\left(\frac{w_{d1}T}{2}\right)\ 1 \leq i \leq k$, Ecu.2.31, de esa manera se encuentran las frecuencias analógicas.
3. Diseñar el filtro analógico con las frecuencias definidas en el apartado anterior.
4. Cambiar por s en el filtro analógico los resultados de $s = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} = \frac{2}{T}\frac{z-1}{z+1}$, Ecu. 2.32.

2.3 Transformada Discreta de Fourier *DFT*

El *DFT* es una secuencia en sí, en vez de una función con una variable constante y es correspondiente a las muestras, espaciadas equitativamente en la frecuencia. La importancia de *DFT* es en la implementación de una variedad de algoritmos de procesamiento digital de señales. Un punto importante que se debe notar, es que hay una relación entre secuencias periódicas y secuencias de duración finita. Esta relación se obtiene construyendo una secuencia periódica, para la cual cada periodo es idéntico al de secuencia de duración finita. Es decir que la representación de las series de Fourier de una secuencia periódica corresponde a la *DFT* de una secuencia de duración finita.

Para conseguir lo anterior, se debe conocer primero la representación de una secuencia periódica como serie de Fourier, que también se conoce como series discretas de Fourier.

2.3.1 Serie Discreta de Fourier (*DFS*)

Considerando una secuencia $\tilde{x}[n]$que es periódica con periodo N, de la manera que $\tilde{x}[n] = \tilde{x}[n + rN]$ para cualquier valor de n y r que sea entero, esta secuencia puede ser representada como una suma de armónicas relacionadas a secuencias exponenciales complejas. La representación de la serie de Fourier sería de la siguiente forma:

$$\tilde{x}[n] = \frac{1}{N}\sum_{k=0}^{N-1} \tilde{X}[k]e^{j\left(\frac{2\pi}{N}\right)kn} \qquad \text{Ecu.2.33}$$

Donde los coeficientes se encuentran con la siguiente función:

$$\tilde{X}[k] = \sum_{n=0}^{N-1} \tilde{x}[n]e^{j\left(\frac{2\pi}{N}\right)kn} \qquad \text{Ecu.2.34}$$

Los coeficientes de la serie de Fourier pueden ser interpretados como la secuencia de duración finita así como una secuencia periódica.

Sin embargo una ventaja de interpretar los coeficientes $\tilde{X}[k]$ de la serie como secuencias periódicas, es que existe una dualidad entre el dominio del tiempo y de la frecuencia. El análisis anterior de la *DFS* es el camino hacia las transformadas de Fourier.

2.3.2 Transformada discreta de Fourier de Señales periódicas

Si $\tilde{x}[n]$ tiene un periodo de N y su correspondiente serie de Fourier es $\tilde{X}[n]$, entonces la transformada de Fourier de $\tilde{x}[n]$ es definida por el tren de impulsos:

$$\tilde{X}(e^{jw}) = \sum_{k=-\infty}^{\infty} \left(\frac{2\pi}{N}\right) \tilde{X}[k]\delta(w - \frac{2\pi k}{N} \qquad \text{Ecu.2.35}$$

Es de notar que $\tilde{X}(e^{jw})$tiene necesariamente un periodo 2л a raíz de que $\tilde{X}[k]$ tiene un periodo de N y los impulsos están separados por enteros múltiplos 2л/N, con N un entero.

2.3.3 Muestreo de la transformada de Fourier

En esta sección se tratará más generalmente la relación entre una secuencia aperiódica con transformada de Fourier $\tilde{X}(e^{jw})$ y la secuencia periódica para la cual los coeficientes de *DFS* corresponden a las muestras de $\tilde{X}(e^{jw})$ espaciadas equitativamente en la frecuencia.

Una secuencia aperiódica x[n] con trasformada de Fourier $X(e^{jw})$, y la secuencia $\tilde{X}[k]$se obtiene muestreando $\tilde{X}(e^{jw})$ a frecuencias wk=2πk/N. Estas muestras $\tilde{X}[k]$, que tiene periodo N, pueden ser la secuencia de los coeficientes de la serie discreta de Fourier $\tilde{x}[n]$.

Generalmente se obtiene para $\tilde{x}[n]$:

$$\tilde{x}[n] = x[n] * \sum_{r=-\infty}^{\infty} \delta[n - rN] = \sum_{r=-\infty}^{\infty} x[n - rN] \qquad \text{Ecu.2.36}$$

Esto indica que $\tilde{x}[n]$ es una secuencia periódica que resulta de la Convolución aperiódica de x[n] con un tren de impulsos unitarios periódicos. Un punto que señalar es que, se está muestreando en el dominio de la frecuencia en vez del dominio del tiempo.

Al desarrollar y aplicar *DFT*, es importante recordar que la representación a través de muestras de la transformada de Fourier es en efecto la representación de una secuencia de duración finita por una secuencia periódica.

2.3.4 Representación de Fourier de una Secuencia de Duración-Finita: La Transformada Discreta de Fourier

Finalmente habrá que reunir todos los apartados anteriores para mostrar el modelo de la DFT. Si se tiene una secuencia de duración-finita x[n] con longitud de N muestras tal que x[n]=0 fuera del rango 0 ≤n≤N-1, recodando que los coeficientes *DFS* de $\tilde{x}[n]$ son muestras (espaciadas en frecuencia por 2л/N) de la transformada de Fourier de x[n], se utiliza esta información para describir las ecuaciones para la *DFT* como:

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{j\left(\frac{2\pi}{N}\right)kn} \qquad \text{Ecu.2.38}$$

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k] e^{j\left(\frac{2\pi}{N}\right)kn} \qquad \text{Ecu.2.39}$$

Esto es con el hecho de que, X[k]=0 para k fuera del intervalo 0≤k≤N-1, y que x[n]=0 para n fuera del intervalo 0 ≤ n ≤N-1, donde X[k] es la transformada discreta de Fourier de x[n]. Para poder explicar con un ejemplo de visualización se presenta una visualización de *DFT*.

La función *fftgui* se utiliza para mostrar cual es el resultado de *DFT* una señal, y se utiliza la función *FFT* (*fast Fourier tranform*) para calcularla. Primero se define la señal como un vector x, con esto la función *fftgui* grafica la parte real e imaginaria del vector x y su *DFT* (fft(x)), como ejemplo se crea un vector x que representa cos(4лt).

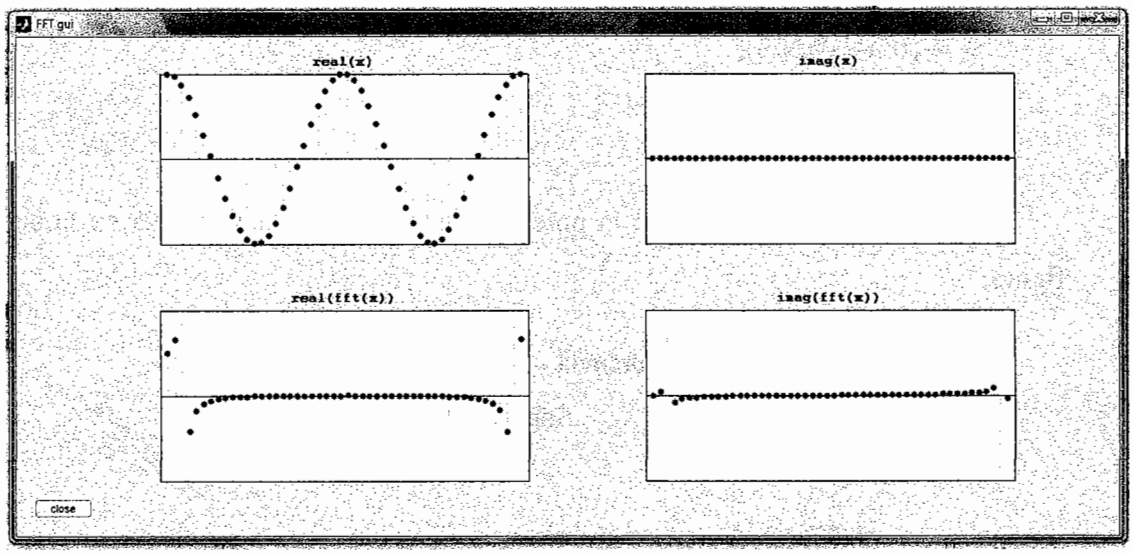
```
>> t = linspace(0,1,50);
```

>> x=cos(4*pi*t);

>> fftgui(x)

El resultado queda así:

Figura 2. 9 Representación de una *DFT*



La función de Matlab fft calcula el *DFT* de una señal. Un ejemplo de cómo se usa esta función es:

Para calcular la potencia de la señal $x = 2.1\cos(2\pi 30t) + 1.5\sin(2\pi 120t) +$ *ruido gausiano*:

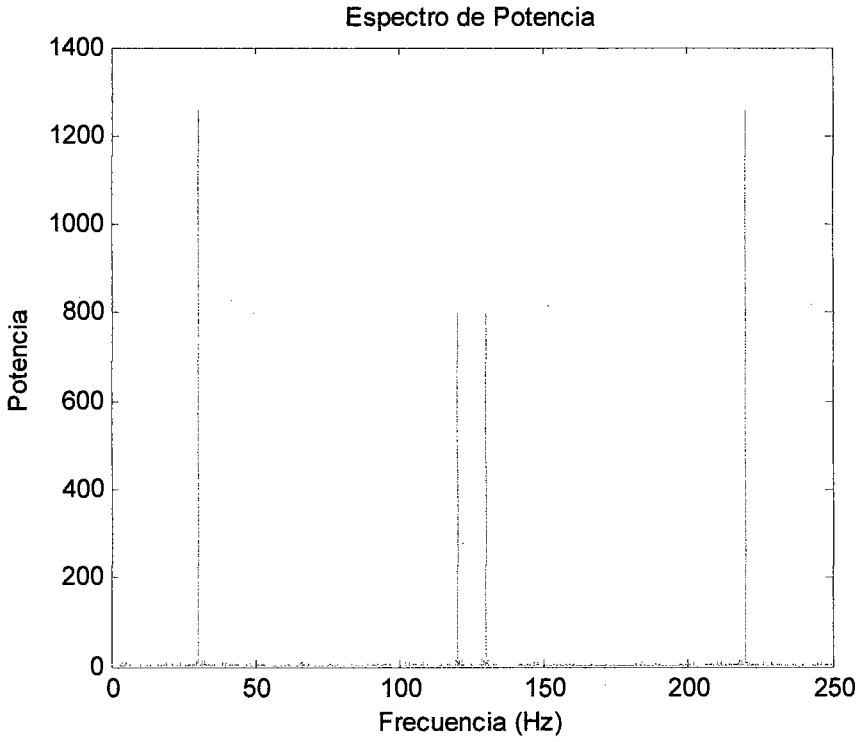
>> N=250; t=0:1/N:10-1/N;

>> x= 2*rand(n)*cos(2*pi*30*t)+1.5*sin(2*pi*120*t)+(1.8)*randn(size(t));

```
>> x= 2.1*cos(2*pi*30*t)+1.5*sin(2*pi*120*t)+(1.8)*randn(size(t));

>> m=length(x);

>> n=pow2(nextpow2(m));

>> y=fft(x,n);

>> f=(0:n-1)*(N/n);

>> Potencia=y.*conj(y)/n;
```

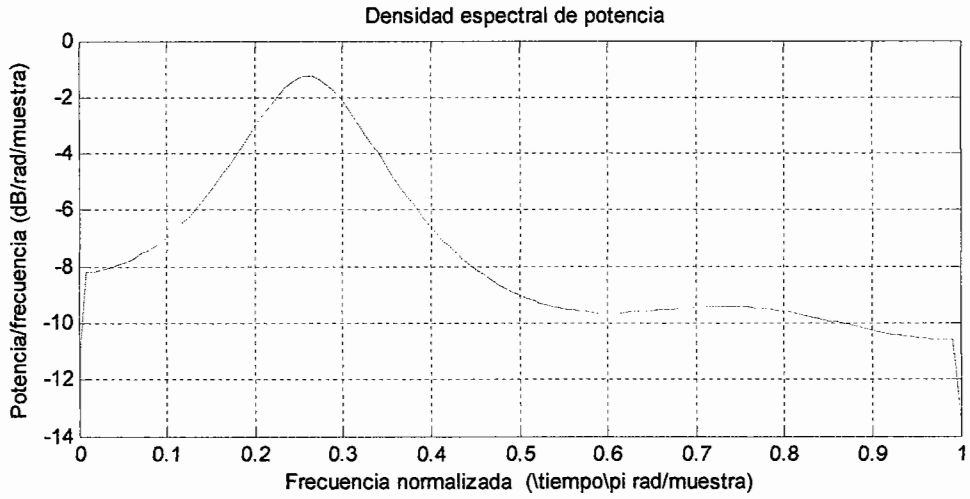
Hasta este punto se calculó la potencia de la señal, y se visualiza en la siguiente figura:

Figura 2. 10 Espectro de Potencia



Se puede observar los componentes a la frecuencia 30 y 120 de las señales coseno y seno más las componentes incluidas por el ruido gausiano. Para una visualización de la densidad espectral de potencia se presenta la siguiente gráfica que representa su amplitud en decibeles.

Figura 2. 11 Densidad espectral de potencia



3. USANDO DSP EN MATLAB

En los capítulos anteriores se ha mostrado la base matemática del procesamiento digital; con estos conocimientos, se verá como se puede usar una herramienta muy útil en el procesamiento de señales, la cual es un *software* llamado Matlab de la empresa *Mathworks*, y su principal programa de simulación, Simulink. Así que en este capítulo se explicará cómo hacer *DSP* con Matlab, las diversas herramientas que presenta y la forma de realizar un proyecto con Matlab-Simulink.

3.1 Qué es Matlab

MATLAB es el nombre abreviado de "*MATrix LABoratory*". MATLAB es un programa para realizar cálculos numéricos con vectores y matrices. Como caso particular puede también trabajar con números escalares tanto reales como complejos, con cadenas de caracteres y con otras estructuras de información más complejas. MATLAB tiene también un lenguaje de programación propio.

MATLAB es un programa de cálculo técnico y científico. Para ciertas operaciones es muy rápido, esto sucede al ejecutar sus funciones en código propio con los tamaños adecuados para aprovechar sus capacidades de trabajo con vectores. También dispone de

un código básico y de varias librerías especializadas (*toolboxes*). Sobre estas *toolboxes* se realizará la mayoría del trabajo.

El entorno de trabajo de MATLAB es muy gráfico e intuitivo, similar al de otras aplicaciones profesionales de *Windows*. El espacio de trabajo (*Workspace*) es un conjunto de variables y de funciones de usuario que en un determinado momento están definidas en la memoria del programa o de la función que se está ejecutando. La ventana *Workspace* constituye un entorno gráfico para ver las variables definidas en el espacio de trabajo.

3.1.1 El editor / *debugger* de Matlab

En MATLAB tienen particular importancia los *archivos-M* (o *M-files*). Los cuales son ficheros de texto *ASCII*, con la extensión ****.m***, que contienen conjuntos de comandos o definición de funciones. La importancia de estos ficheros-M es que al teclear su nombre en la línea de comandos, se ejecutan todos los comandos, contenidos en dicho fichero, uno tras otro. El poder guardar instrucciones y grandes matrices en un fichero permite ahorrar mucho trabajo de tecleado.

Aunque los ficheros ****.m*** se pueden crear con cualquier editor de ficheros *ASCII* tal como *Notepad*, MATLAB dispone de un *editor* que permite tanto crear y modificar estos ficheros, como ejecutarlos paso a paso para ver si contienen errores (a este proceso se le denomina *Debug* o depuración).

3.1.2 Simulink

Simulink es un entorno para diseño y simulación multidominio, basado en modelos para sistemas dinámicos y embebidos. Proporciona un entorno gráfico interactivo y un conjunto de bibliotecas de bloques personalizables que permiten diseñar, simular, implementar y probar una variedad de sistemas de tiempo variable, incluidas las comunicaciones, los controles, procesamiento de señales, procesamiento de video y procesamiento de imágenes.

Simulink está integrado con MATLAB, proporciona acceso inmediato a una amplia gama de herramientas que permiten desarrollar algoritmos, analizar y visualizar simulaciones, crear secuencias de comandos de procesamiento por lotes, personalizar el entorno de modelado, y definir señales, parámetros y datos de prueba.

3.1.2.1 Características principales

- Posee una biblioteca extensa de bloques predefinidos, que pueden ser ampliados.
- Tiene un editor gráfico interactivo para la recopilación y gestión de los diagramas de bloque.
- Capacidad para gestionar diseños complejos mediante la segmentación de los modelos en jerarquías de los componentes de diseño.

- Modelo *Explorer* para navegar, crear, configurar y buscar todas las señales, parámetros, propiedades y código generado asociado al modelo.
- Diferentes modos de simulación.
- Acceso a Matlab para analizar y visualizar resultados, personalizar el entorno de modelado, y la definición de señales, parámetros y datos de prueba.
- Identificador de errores de modelado.

3.1.2.2 Crear y trabajar con modelos

Con Simulink, se puede rápidamente crear, modelar y mantener un diagrama de bloques detallado de su sistema utilizando un conjunto completo de bloques predefinidos. Simulink proporciona herramientas para el modelado jerárquico, gestión de datos y la personalización del subsistema, lo que facilita la creación de representaciones concisas, precisas, independientemente de la complejidad de su sistema.

3.1.2.3 Selección y personalización de los bloques

El *software* de Simulink incluye una amplia biblioteca de funciones de uso común en el modelado de un sistema. Estos incluyen:

- Bloques dinámicos para análisis continuos y discretos.
- Bloques de algoritmos.
- Bloques de estructuras.

La ventaja que da Simulink, es que permite personalizar los bloques construidos o crear nuevos directamente en Simulink y utilizarlos como librerías personales.

3.1.2.4 Construcción y Edición del modelo

Simulink permite construir modelos por medio de arrastrar y soltar bloques desde el buscador de librerías, ponerlas en el editor gráfico y conectar los diversos bloques con líneas que establecen las relaciones matemáticas entre los bloques. Se puede mejorar los modelos utilizando las funciones para editar, tales como copiar, pegar, regresar, alinear, distribuir y cambiar tamaño. La ventaja con Simulink es que la interfaz de usuario da un control total sobre lo que se desea ver o analizar en la pantalla.

3.1.2.5 Organizar un modelo

Simulink permite organizar el modelo en niveles claros y manejables de jerarquía mediante el uso de subsistemas. Los subsistemas son un grupo de *toolboxes* y señales encapsuladas en un solo bloque. Simulink permite agregar una interfaz de usuario personalizada para ocultar el contenido de cada subsistema y hacer que éstos aparezcan como un bloque con su propio ícono y caja de diálogo de parámetros.

También puede segmentar el modelo en componentes de diseño para modelar, simular y verificar cada componente de forma independiente. Los componentes pueden ser guardados como modelos separados mediante el uso de referencia de modelo, o

como subsistemas en una biblioteca. Se puede volver a utilizar los componentes de diseño en múltiples proyectos. Los bloques lógicos permiten modelar comandos simples de control que habiliten o seleccionen subsistemas.

3.1.2.6 Trabajando con señales y parámetros

Simulink permite definir y controlar los atributos de las señales y parámetros asociados con el modelo. Las señales son cantidades variables en el tiempo, representadas por las líneas que conectan los bloques. Los parámetros son coeficientes que ayudan a definir la dinámica y el comportamiento del sistema.

Los atributos de las señales y parámetros se pueden especificar directamente en el diagrama o en un diccionario de datos por separado. Utilizando el explorador de modelo, se puede manejar su diccionario de datos y rápidamente cambiar la finalidad de un modelo mediante la incorporación de diferentes conjuntos de datos.

Se pueden definir los siguientes atributos a las señales y parámetros:

- Tipo de dato: *single*, *double*, *signed* o *unsigned* de 8, 16 o 32 bits enteros; *boolean*; y punto fijo.
- Dimensiones: escalar, vector, matriz, o un cadena de N-D
- Complejidad: valores reales o complejos.
- Rango mínimo y máximo, valores iniciales, y unidades de ingeniería.

Los datos de tipo punto fijo proporcionan soporte para escalonamiento y longitud arbitraria de palabra hasta 128 bits. También se pueden especificar el modo de la señal de muestreo, para permitir una ejecución más rápida del procesamiento de señales.

3.1.2.7 Llevar a cabo una simulación

Después de construir el modelo, se puede simular su comportamiento dinámico y ver los resultados en vivo. También proporciona varias características y herramientas para asegurar la velocidad y precisión de la simulación, incluyendo paso fijo y solucionadores de paso variable, un depurador gráfico, y un perfilador de modelo.

Los solucionadores son los algoritmos de integración numérica que calculan la dinámica del sistema en el tiempo, usando la información contenida en el modelo. Simulink ofrece solucionadores de apoyo a la simulación de una amplia gama de sistemas, incluidos los de tiempo continuo (analógico), en tiempo discreto (digital), sistemas híbridos (de señal mixta), y multitarea de cualquier tamaño.

Estos solucionadores pueden simular sistemas rígidos y sistemas con las condiciones de estado, tales como discontinuidades, incluyendo cambios instantáneos en un sistema dinámico. Se puede especificar las opciones de simulación, incluyendo el tipo y las propiedades del programa de solución, empezar la simulación y los tiempos de parada, y si se debe cargar o guardar los datos de simulación. También se puede

establecer la optimización y la información de diagnóstico para su simulación. Las diferentes combinaciones de opciones se pueden guardar con el modelo.

3.1.2.8 Depurar una simulación

Esta es una herramienta interactiva para examinar los resultados de la simulación, la localización y el diagnóstico de un comportamiento inesperado en un modelo Simulink. Permite rápidamente identificar los problemas en el modelo, reforzando a través de una simulación de un método a la vez, y examinar los resultados de la ejecución de ese método. Los métodos son funciones que Simulink utiliza para resolver un modelo en cada paso de tiempo durante la simulación. Los bloques se componen de varios métodos.

El depurador permite establecer puntos de interrupción, control de la ejecución de la simulación, y la información del modelo de visualización. Este programa puede ejecutarse desde una interfaz gráfica de usuario (*GUI*) o desde la línea de comandos de MATLAB. La interfaz gráfica de usuario proporciona una visión clara, un código de colores del estado de ejecución del modelo. A medida que el modelo se simula, puede mostrar información sobre los estados de bloque, bloque de entradas y salidas, y demás información, así como la ejecución animada de métodos de bloques directamente sobre el modelo.

3.1.3 *Blocksets* de Simulink y *Toolboxes* de Matlab

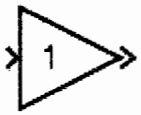
En esta sección se detallará algunos de los bloques más importantes en Simulink así como los *toolboxes* de Matlab que son especiales para el procesamiento de señales. Debido a la extensa librería que Simulink tiene, solamente aquellos comandos o bloques que estén relacionados con el *DSP* serán expuestos.

Demux

El bloque *Demux* extrae los componentes de una serie de datos de entrada y tiene por salidas, componentes como serie de datos separadas. Las señales de salida están ordenadas de arriba hacia abajo en el puerto de salida.

El parámetro número de salidas permite especificar el número y, opcionalmente, la dimensión de cada puerto de salida.

Gain



El bloque de ganancia multiplica la entrada por un valor constante que representa una ganancia. La entrada y la ganancia puede ser cada uno un escalar, vector o matriz. Se especifica el valor de la ganancia en el parámetro de ganancia. El parámetro de multiplicación permite especificar la multiplicación de elemento racional o de la matriz.

Mux

El bloque *Mux* combina sus entradas en un solo vector de salida. Una entrada puede ser una señal de escalar o vectorial. Todas las entradas deben ser del mismo tipo de datos así como tipo numérico. Los elementos de la señal de salida vectorial se ordenan de arriba a abajo o de izquierda a derecha de las señales de puerto de entrada.

Scope

El bloque *Scope* muestra su entrada con respecto al tiempo de simulación. El bloque permite tener varios ejes (uno por cada puerto) y los ejes tienen un rango de tiempo común con eje "Y". El bloque *Scope* permite ajustar la cantidad de tiempo y el rango de valores de entrada mostradas. Puede mover y cambiar el tamaño de la ventana y puede modificar los valores del alcance de los parámetros durante la simulación. Es un bloque de visualización, en tiempo de simulación.

Suma de elementos

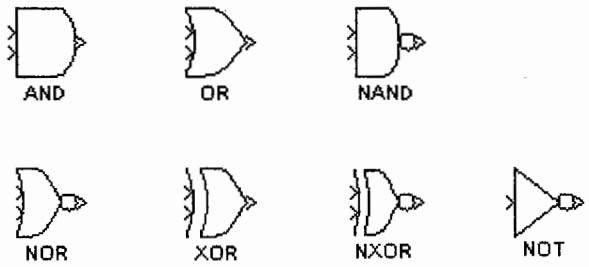
El bloque de Suma realiza adición o sustracción de las entradas. Este bloque puede sumar o restar escalar, vector o matriz como entradas. También puede contraer los elementos de una señal. El bloque de Suma acepta señales reales o complejas de los tipos de datos siguientes:

- Punto flotante.
- Número entero
- Punto fijo
- Booleano

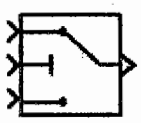
Operador lógico

Realiza la operación lógica especificada en las entradas. Un valor de entrada es *TRUE* (1) si es distinto de cero y *FALSE* (0) si es cero.

A continuación se presentan los iconos que representan las funciones lógicas que las que Simulink trabaja.

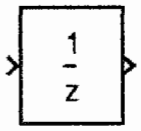


Switch



El bloque *Switch* tiene como salida la señal en la entrada uno o tres, dependiendo del valor de la entrada dos. La primera y tercera entradas se denominan entradas de datos. La segunda entrada se llama la entrada de control. Selecciona la condición bajo la cual el bloque pasa a la primera entrada con los criterios para el paso de parámetros de la primera entrada.

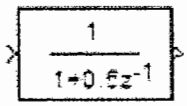
Retraso unitario



Este bloque representa un retraso de la señal de entrada por un período de muestreo especificado. Este bloque es equivalente al operador z^{-1} en tiempo discreto de la transformada Z. El bloque acepta una entrada y genera una salida, que puede ser tanto escalar o vector de ambos. Si la entrada es un vector, todos los elementos del vector se retrasan por el mismo período de muestreo.

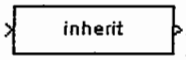
Primero se especifica el bloque de salida para el período de muestreo con el parámetro de las condiciones iniciales. El tiempo entre las muestras se especifica con el parámetro de tiempo de muestra. Un valor de -1 significa que el tiempo de la muestra se hereda.

Filtros digitales



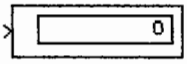
Este bloque filtra cada canal de la señal de entrada de forma independiente con el tiempo, tratando a cada elemento de la entrada como un canal individual. Las dimensiones de salida siempre serán equivalentes a las dimensiones de la señal de entrada filtrada. El bloque implementa filtros estáticos con coeficientes fijos. Se puede ajustar los coeficientes de un filtro estático.

Especificación de señal



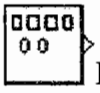
Este bloque permite especificar los atributos de la señal conectada a sus puertos de entrada y de salida. Si hay conflicto con los atributos especificados por los bloques conectados a sus puertos, Simulink muestra un error al compilar el modelo. Por ejemplo, en el inicio de una simulación. Si no existe ningún conflicto, Simulink elimina el bloque de señal de especificaciones del modelo elaborado. En otras palabras, el bloque de especificación de señal es un bloque virtual. Sólo existe para especificar los atributos de una señal y no juega ningún papel en la simulación del modelo.

Display



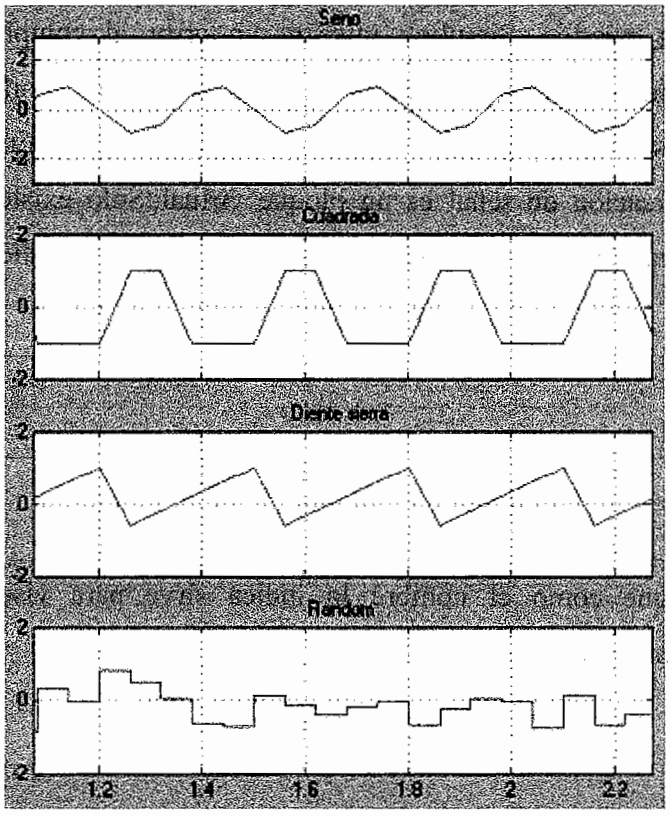
Este bloque como el nombre lo indica sirve para visualizar resultados numéricos, dependiendo del formato del dato.

Generador de señal


El bloque generador de señal puede producir una de las cuatro diferentes formas de onda: onda sinusoidal, onda cuadrada, onda diente de sierra, y la onda al azar. Los parámetros de la señal puede ser expresada en *Hertz* (por defecto) o radianes por segundo.

En la figura 3.1 se muestra cada señal que aparece en un ámbito de aplicación utilizando los valores predeterminados de parámetros.

Figura 3. 1 Generador de señales

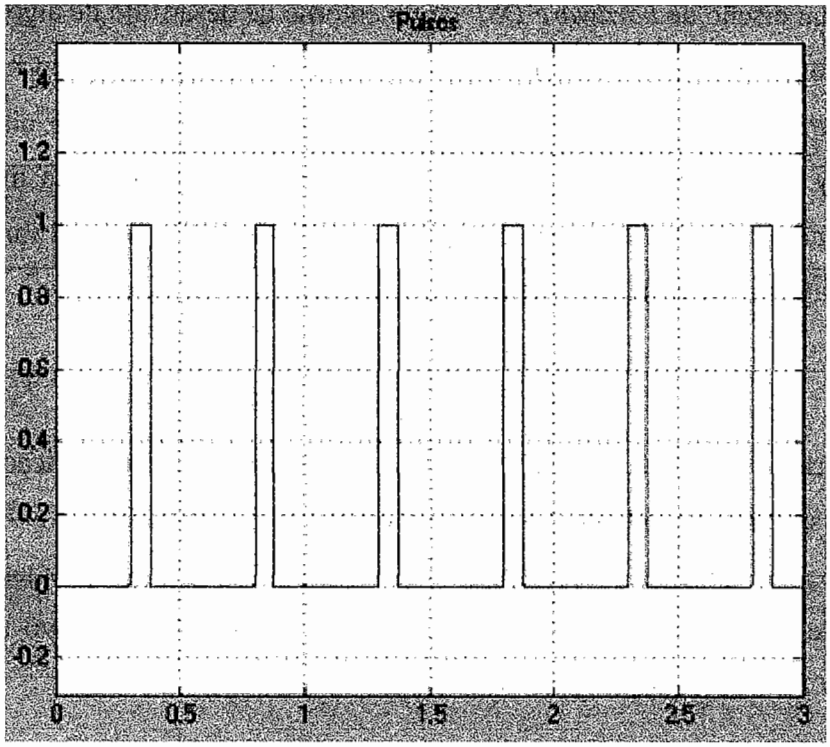


Generador de pulso

 El bloque genera pulsos de onda cuadrada a intervalos regulares. Los parámetros de forma de onda del bloque son amplitud, ancho de pulso, periodo, y la fase de retardo, estos determinan la forma de la onda de salida.

El siguiente diagrama muestra cómo cada parámetro afecta a la forma de onda.

Figura 3. 2 Generador de pulsos



3.1.3.1 *DSP builder* Simulink *Blockset*

El diseño del sistema de procesamiento digital de señal (*DSP*) en dispositivos lógicos programables de Altera (*PLDs*) requiere tanto algoritmo de alto nivel y el idioma de descripción de *hardware* (*HDL*) como herramientas de desarrollo. Altera *DSP Builder* integra estas herramientas mediante la combinación del desarrollo de algoritmos, simulación y verificación de las capacidades de MATLAB y Simulink de The *MathWorks*, herramientas de diseño a nivel de sistema con *VHDL* y *Verilog HDL* flujos de diseño, incluyendo el software de Altera Quartus II.

DSP Builder acorta los ciclos de diseño de *DSP*, ayudando a crear la representación de un diseño de *hardware DSP* en un entorno de desarrollo de algoritmos a usar. Se puede combinar las actuales funciones de MATLAB y Simulink con bloques de Altera *DSP Builder* y Quartus II, vinculando funciones *MegaCore* para diseño a nivel de sistema y aplicación, con el desarrollo de algoritmos *DSP*. De esta manera, permite que el sistema *DSP Builder*, cree los algoritmos, y los diseños de hardware para compartir una plataforma de desarrollo común.

También permite utilizar los bloques de *DSP Builder* para crear una aplicación de *hardware* de un sistema modelado en Simulink en el tiempo de muestreo. *DSP Builder* contiene bloques que cubren las operaciones básicas como la aritmética o almacenamiento de funciones y se aprovecha de las funciones del dispositivo a usar, tales como *built-in PLLs*, bloques *DSP*, o memoria incrustada.

3.2 *DSP* de alta velocidad usando lógica programable

Los dispositivos de lógica programable son un arreglo de elementos, cada uno de los cuales se puede configurar como una rutina de procesador complejo. Una ventaja que presentan los *PLD* es que se pueden vincular rutinas juntas en serie, de la misma forma que un procesador de señal digital lo haría, o bien conectarse en paralelo. Cuando se conecta en paralelo, favorece a mejorar su rendimiento más que los procesadores de señal digital estándar, mediante la ejecución de cientos de instrucciones al mismo tiempo.

3.2.1 Procedimiento de diseño

El diagrama de diseño incluye los siguientes pasos:

Se comienza por crear un modelo de diseño en Simulink. Después de haber creado el modelo, se compila directamente en el software Quartus II, la producción de archivos de *VHDL* para la síntesis y compilación Quartus II, o generar archivos para la simulación *VHDL*.

- Crear un modelo con una combinación de bloques de Simulink y el DSP *Builder*.
- Incluir un bloque de reloj de la biblioteca de *DSP Builder AltLab* para especificar el reloj de base al diseño, que debe tener un período mayor que 1ps (pico segundo), pero menos de 2,1 ms.
- Establecer un solucionador discreto (no continuo) en Simulink. Elegir un tipo de programa de solución de paso fijo si se utiliza un único reloj, o un tipo paso variable si se está utilizando múltiples dominios de reloj.
- Simular el modelo en Simulink utilizando un bloque *Scope* para visualizar y supervisar los resultados.
- Ejecutar la señal del compilador para configurar la simulación y la síntesis *RTL*.
- Realizar simulación *RTL*. Con la simulación hecha, el software *DSP Builder* tiene la opción de comunicarse con el *software* ModelSim. para una mejor visualización de la simulación.
- Utilizar la salida de los archivos generados por el bloque *DSP* Generador de Señal del compilador para realizar la síntesis *RTL*.
- Compilar el diseño en el software Quartus II.
- Descargar a una placa de desarrollo de *hardware* y de prueba.

Con estos pasos se realiza un sistema de *DSP*, así conociendo la respuesta de la simulación se obtiene un mejor diseño.

3.3 Adquisición de datos

A continuación se presentará la manera en que se obtendrán los datos.

3.3.1 Bloque de Entradas analógicas

Este bloque adquiere los datos de múltiples canales analógicos de dispositivos de adquisición de datos. Lo que hace este bloque es abrir, iniciar, configurar y controlar un dispositivo de adquisición de datos analógicos. La apertura, inicialización y configuración del dispositivo se producen una vez al comienzo de la ejecución del modelo. Durante el tiempo de ejecución del modelo, el bloque adquiere datos de forma sincrónica (entrega al bloque actual los datos que el dispositivo proporciona) o asincrónica (búfer de datos de entrada).

El bloque no tiene puertos de entrada. Más sin embargo, puede tener uno o más puertos de salida, dependiendo de la configuración que se elija, en el cuadro fuente del

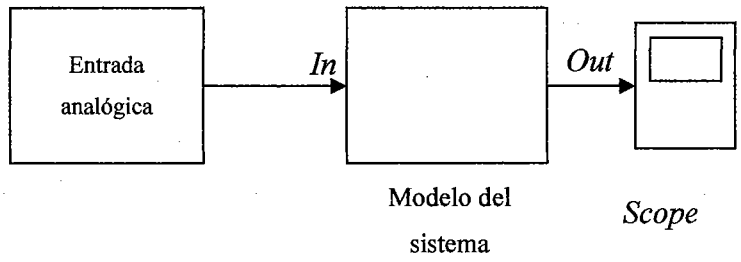
bloque de diálogo Parámetros. Se utiliza el bloque de la entrada análoga para incorporar los datos medidos en vivo en Simulink para:

- Caracterización del sistema.
- Algoritmo de verificación
- Modelado del sistema y algoritmo
- Validación del diseño y modelo
- Diseño de control

Permite utilizar el bloque de entrada analógica de manera sincrónica o asincrónica. Seleccionando el modo de adquisición en el cuadro fuente del bloque de diálogo Parámetros. El siguiente diagrama muestra el escenario del sistema básico de entrada analógica, donde se tiene que:

- Adquirir datos en cada paso de tiempo o una vez por ejecución del modelo
- Analizar los datos, o utilizarlos como entrada a un sistema en el modelo
- Por último, mostrar los resultados

Figura 3. 3 Ejemplo de adquisición de datos



3.3.2 Bloque salida análoga

Este bloque tiene como salida de datos, los cuales van hacia varios canales analógicos de dispositivos de adquisición de datos. Este bloque tiene las operaciones de abrir, iniciar, configurar y controlar un dispositivo de adquisición de datos analógicos. La apertura, inicialización y configuración del dispositivo se producen una vez al comienzo de la ejecución del modelo. Durante el tiempo de ejecución del modelo, el bloque de salida entrega datos al *hardware* de manera sincrónica o asincrónica (búfer de datos de salida).

Este bloque hereda el tiempo de la muestra desde el bloque de conducción conectado al puerto de entrada. Los tipos de datos válidos de la señal en el puerto de entrada son los datos de tipo doble o los admitidos por el *hardware*.

3.3.3 Entrada Digital

El bloque de entrada digital sincroniza las salidas de la última exploración de los datos disponibles en las líneas digitales seleccionadas en cada paso de tiempo de simulación. Adquiere sin búfer de datos digitales, y los datos suministrados son un vector binario. Obtiene el conjunto de valores de múltiples líneas digitales de dispositivos de adquisición de datos.

3.3.4 Salida Digital

La salida digital hereda el tiempo de la muestra desde el bloque de conducción conectado al puerto de entrada. El tipo de datos de la señal en el puerto de entrada debe ser un tipo de datos lógico.

El bloque de salida digital sincroniza las salidas de la última serie de datos al hardware cada paso de tiempo de simulación. Hace salir sin búfer los datos digitales. Los datos de salida son siempre un vector binario (*binvec*).

3.4 Salida y visualización en dominio de la frecuencia

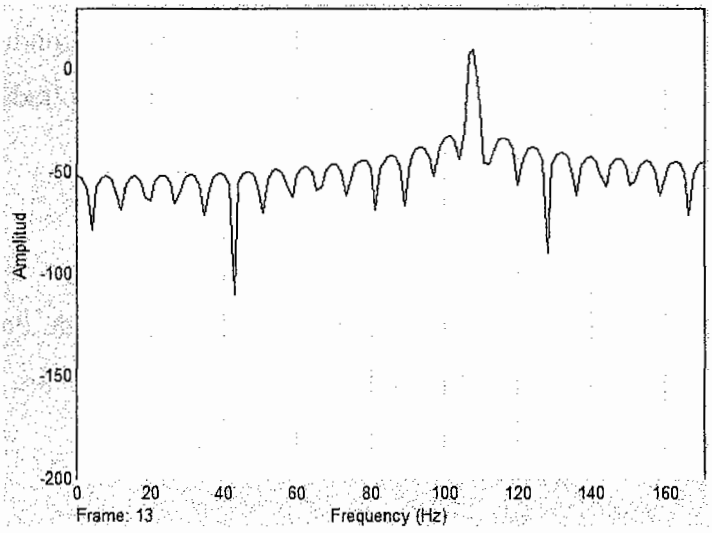
En esta sección se dará detalle de la forma en que se puede visualizar la representación de una señal en el dominio de la frecuencia.

3.4.1 Analizador espectral

El bloque de analizador espectral calcula y muestra el periodograma de la entrada. La entrada puede ser un vector basado en muestras, basado en imágenes o de una matriz basada en marcos.

La siguiente gráfica es un ejemplo de la salida de este bloque.

Figura 3. 4 Analizador de espectros



4. SIMULACIÓN CON EL CONSTRUCTOR DSP EN SIMULINK, QUARTUS II Y MODELSIM

Después de haber mostrado las herramientas con las que se trabajará, es tiempo de ponerlas en conjunto para alcanzar los objetivos planteados en esta tesis.

Para comenzar, se utilizará Simulink y en especial el *Blockset* de Altera, *DSP Builder* 1.9.0, tanto la versión normal como *DSP Builder Advanced Blockset*. Una vez creados los modelos, simulados y revisados los resultados, se procederá a mostrar cómo este modelo puede situarse dentro de un dispositivo *FPGA* que funcione como *DSP*, mostrando los reportes de compilación, análisis de tiempo, la ubicación de los pines en el dispositivo, así como la forma esquemática del modelo, y muchos otros reportes que se puedan generar con el software Quartus II.

Por último se ilustrará el resultado por medio de una simulación en *ModelSim* de cómo se ven las señales en un osciloscopio. De esta forma se mostrará las señales que atraviesan el modelo, tanto las de entrada como de salida y señales auxiliares también.

Ésta es la forma de preparar una programación de un *DSP*, y el desarrollo de un proyecto, llevándolo hasta el punto que esté listo el modelo para ser descargado al mismo *Chip*.

Al final se tendrá un modelo que simula el análisis de señales, tal como lo haría un chip *DSP*.

Un punto clave que se debe aclarar es que, todas las herramientas que se usarán para el procesamiento digital de señales serán las que proporciona el *software* de Altera, con su constructor de *DSP*; se pueden tener los mismo resultados usando los bloques propios de Simulink, pero solo serviría de simulación y visualización del modelo en general, por lo que no estaría listo para ser descargado a un dispositivo para ser programado.

Los únicos bloques propios de Simulink que se usarán son los que permiten adquirir datos, como las fuentes de señales, los dispositivos de salida, tanto visores numéricos como el visualizador de señales.

El tipo de señales que se utilizarán será variado, en general se trabajará con variables de punto flotante, en otros casos variables entera con signos de 16 o 32 *bits*. Utilizando las herramientas de conversión de datos es como se mantendrá la compatibilidad entre ambas señales. Habrá que agregar señales de tipo booleana que servirán como indicadores, iniciadores, terminadores y de control de algunas partes del modelo.

El orden en que se presentarán los modelos son los siguientes:

- Un modelo de división de dos números en punto flotante.
- Un modelo que calcula la transformada rápida de Fourier, utilizando algoritmo que incluye multiplicadores en mariposa, sumadores, y elementos primitivos.

- Está basado en el algoritmo *Radix*-2. En la salida se podrá visualizar los componentes reales e imaginarios así como la magnitud.

Las señales que se tratarán son:

- Generadas matemáticamente, por bloques de fuente de señales propias de Simulink.
- De un archivo de audio.
- De adquisición de datos, en este caso será a través de un micrófono hacia la computadora.

Todos los resultados se podrán ver tanto con los bloques de Simulink, como en ModelSim y Quartus II.

4.1 Algoritmo para la División

Un detalle primordial en el diseño de microprocesadores modernos es la capacidad del *Hardware* para soportar aritmética de punto flotante. Aunque las operaciones de división y raíz cuadrada son relativamente poco comunes, de utilizarlas en aplicaciones de propósito general, éstas son indispensables e importantes en varias aplicaciones modernas.

En esta sección se presenta un algoritmo para la división basado en la utilización de una serie de Taylor.

4.1.1 Algoritmo

Un punto importante en el diseño de un algoritmo para la división es tener en cuenta el tiempo de retraso que esta operación usará para ejecutarse. De aquí viene la complicación del diseño. A comparación de las otras operaciones aritméticas que tienen una forma de calculó más rápida. Los principios de este diseño se hacen en base a series de Taylor, pero el que se utilizará en esta sección será una serie de *Maclaurin* alrededor de 1, esta serie da el inverso de un número comprendido de 0 a 1, con esta base se pueden conseguir, por medio de multiplicar por múltiplos de 10, la representación de otros números.

A continuación se presenta la serie que se utiliza:

```
    %%Generar
Serie
    syms x
    f= 1/x;
    sdiv=taylor(f,
6,1);
    pretty(sdiv)
```

Y el resultado es:

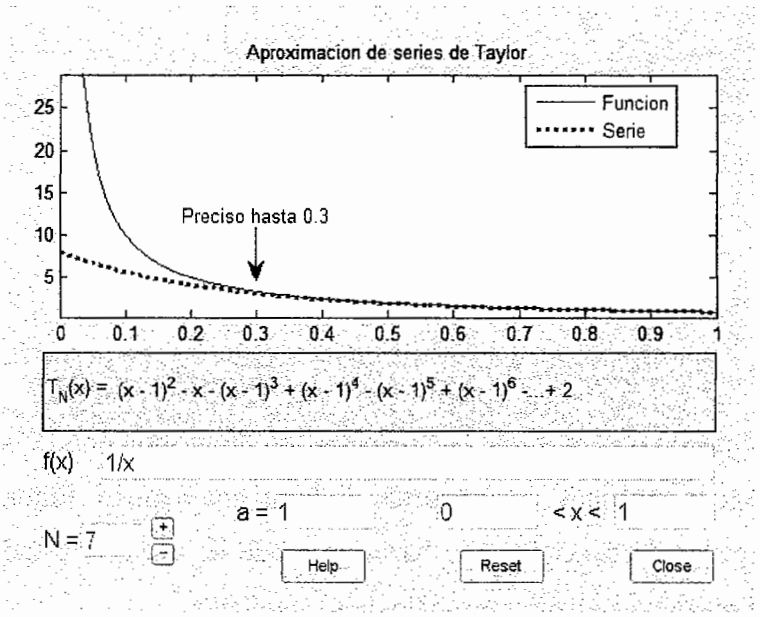
$$1-(x-1)+(x-1)^2-(x-1)^3+(x-1)^4-(x-1)^5$$

Ecu.4.1

Este es el resultado para una serie de grado 6. Ahora la pregunta es, que tan certero es este polinomio, o ¿hasta qué grado se debe de incluir?, para responder esta pregunta se visualizará con un *Toolbox* de Matlab la representación gráfica. El *Toolbox* que se usará se llama *Taylortool*.

Representación gráfica de la serie por medio de Matlab:

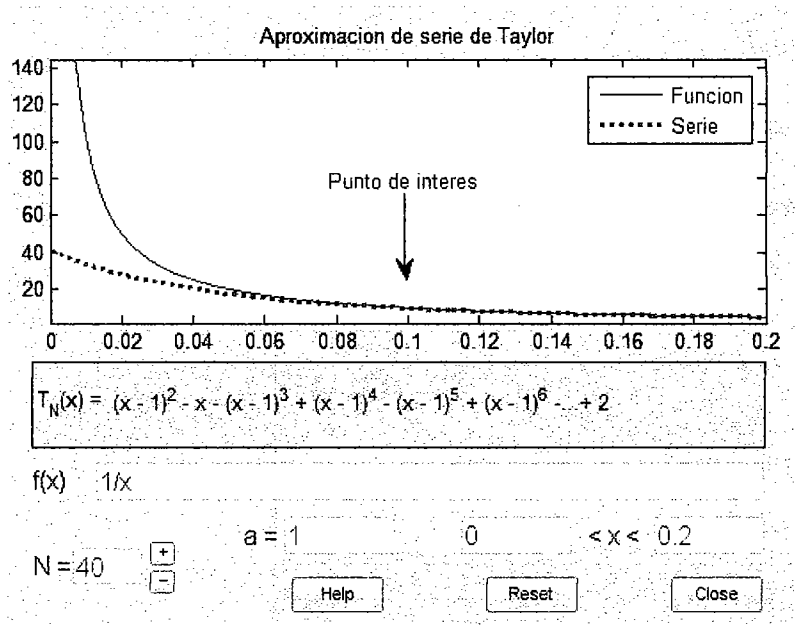
Figura 4. 1 Aproximación de series de Taylor



De esta gráfica se puede observar que para un grado de 7 se tiene suficiente precisión de 0.3 hasta 1, ahora lo que se busca es que por lo menos sea preciso a partir de 0.1, para lograr esto se incrementa el valor de N, a aproximadamente un valor de 40.

A continuación se grafica la serie de Taylor con una cantidad de 40 coeficientes.

Figura 4. 2 Serie de Taylor n=40

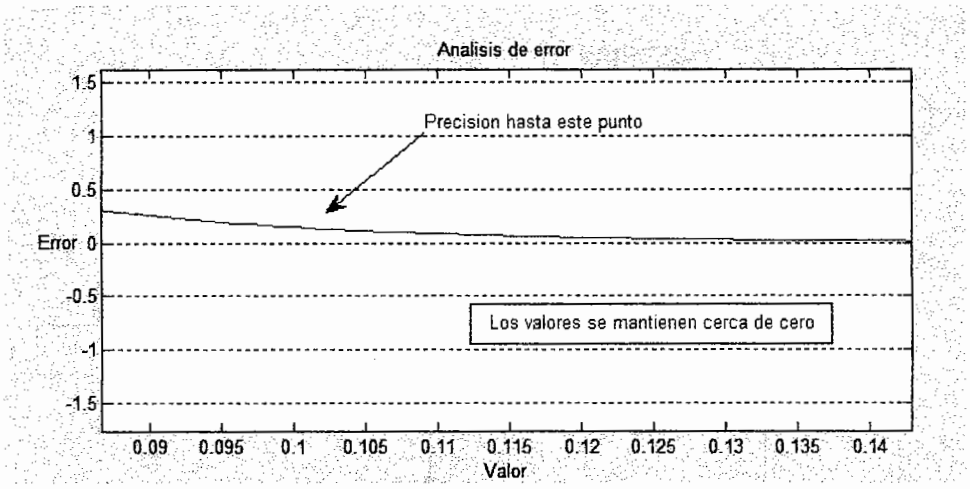


Con esta gráfica se puede observar que la aproximación de la serie con la función real a partir del punto 0.1, las gráficas se comportan de la misma manera. Entonces el algoritmo tiene que ser capaz de hacer la serie hasta un grado de 40. Un estudio del error de aproximación se puede visualizar por medio de lo siguiente:

```
     %%Análisis de error
     syms x  % generar la
serie
     f=1/x;
     tayy=taylor(f,40,1);
     pol=sym2poly(tayy);
     y=polyval(pol,t);

     t=0.001:0.001:1;
%generar la función
     gg=1./t;
     error=gg-y;
     plot(t, error)
```

Y la gráfica es:

Figura 4. 3 Análisis de la serie

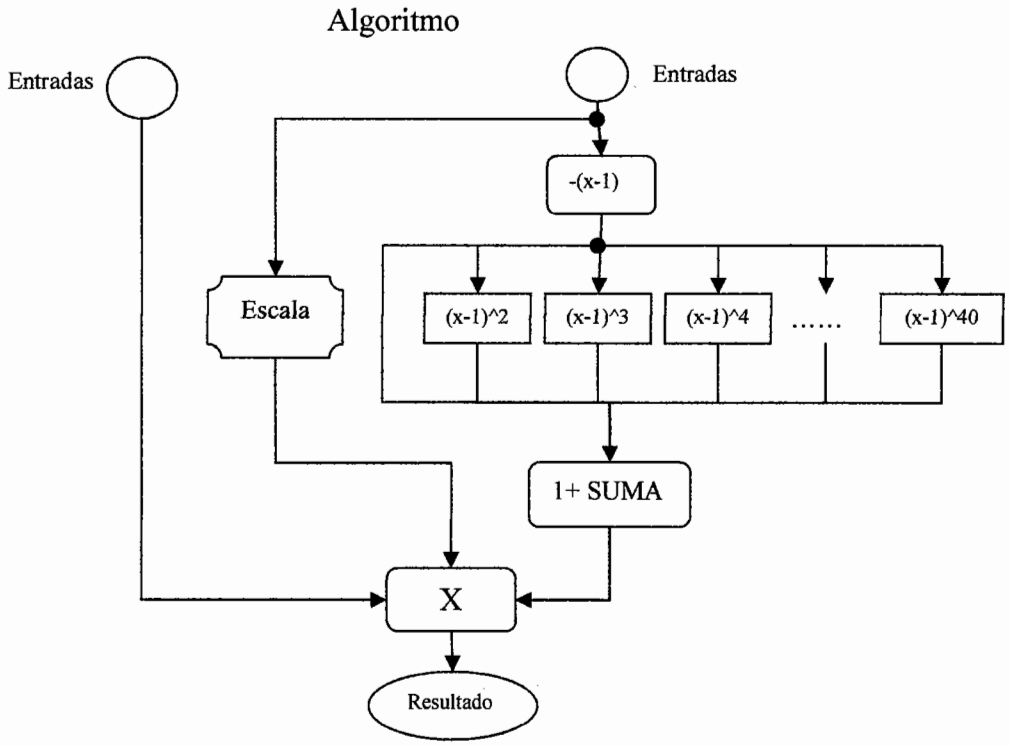


Se pude ver que al acercase a 0.1 el valor de aproximación es mayor, pero aún así suficiente para obtener el resultado que se busca. El error para 0.1 es aproximadamente de 0.1478 unidades.

El diagrama de bloques del algoritmo se presenta de la siguiente manera:

Continuación de la gráfica

Figura 4. 4 Diagrama del Algoritmo



Tiene como entradas los valores de x, y. Se calcula por medio de la serie de Taylor el valor de 1/x, esto se logra por medio de multiplicadores de hasta grado 40.

El bloque que tiene como nombre "Escala" es para comparar en que rango se encuentra el número, y lo multiplica por un factor que le permita ser calculado, para luego ser multiplicado por ese mismo factor para obtener la respuesta deseada. Al final de la serie se tiene la suma de cada uno de los multiplicadores, a estos se les suma el valor de 1, y se obtiene la aproximación a 1/x.

A la salida de esta suma, se tiene el paso final que es multiplicar por el valor de entrada y, para obtener y/x. Todo este diseño se realizará con los bloques disponibles como herramientas de programación propias del *DSP*, éstas son los bloques primitivos como se conocen, que incluyen, suma, resta, multiplicación, comparación, valor absoluto, entre otros.

4.1.2 Diseño en Simulink

Este es el siguiente paso, poner el algoritmo en una forma que Simulink pueda interpretar y tener los resultados que se desean obtener. Para esta parte se utilizará el *blockset* de Altera *DSP Builder Advanced* en conjunto con los bloques que proporciona Simulink.

Los parámetros iniciales con los que se debe colocar la simulación son:

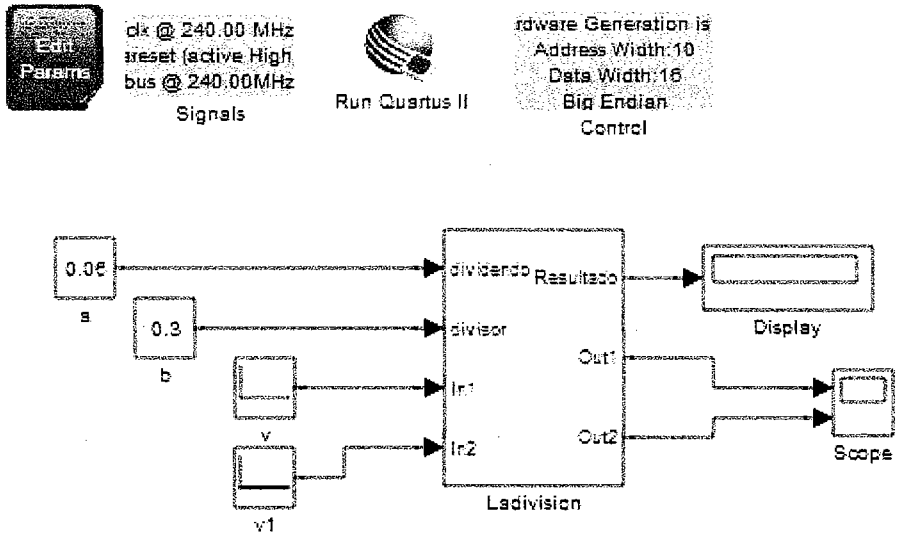
- El tipo de solver debe ser *Fixed-step.*
- Configurado a valores discretos.
- Un tiempo de 250-300 segundos.

Y los parámetros de las señales son:

- Señal de reloj de 240.00 Mhz.
- Bus a 240.00 Mhz.
- El tipo del sistema del bus es *Big Endian*
- El tipo de señales que se trabajarán es un valor de paso fijo (*fixed-step*) de una longitud de palabra de 16 y de fracción de 15.

El diseño de mascara se presenta de la siguiente manera:

Figura 4. 5 Modelo Simulink para la división



En esta parte se ve en la Fig.4.5 varios bloques que configuran las características del diseño; tal como el bloque "*Signals*" y "*Control*". En el bloque de control está la opción de generar un archivo que representa el *Hardware*, que es un archivo de extensión *.rtl, este archivo se utilizará luego para generar los reportes del *DSP*.

Los bloques "v" y "v1" se usan para señalizar y marcar tiempos internos en el *DSP*. El bloque de "v" se ajusta como usa señal discreta que se repite y tiene un valor de 1. De igual manera el bloque "v1" se asigna un valor de 0. Los bloques de entrada "a" y "b", son el divisor y el dividendo. Tienen la característica de representar el número de entrada como uno *fixed-step*. Con las longitudes de palabra antes descritas. El bloque "a" será el que se trabajará para generar su inverso por medio de la aproximación a la serie de Taylor 1/a, para posteriormente ser multiplicada por "b".

El siguiente bloque llamado "Ladivision", el que contiene todo los procedimientos y representa en si el dispositivo *DSP*, está representado por un subsistema, de 4 entradas y 3 salidas. Así que el DSP tendría 2 entradas de datos "a", "b"; y una salida con el nombre de "Resultado". Las otras entradas y salidas son para propósitos de control y uso interno del dispositivo.

4.1.2.1 Subsistema "Ladivision":

Uno de los motivos principales de usar un subsistema es para organizar el modelo que se está diseñando. Otra de las razones por las que se utiliza subsistema en el blockset de *DSP* es que permite visualizarlo como el dispositivo real, dentro de este subsistema van incluidas todas las señales, procesos y bloques propios del *DSP*; las entradas y salidas son por lo general bloques de Simulink que son útiles para generar señales, visualizar, y hacer una representación de lo que se desee hacer con el *DSP*.

Se especifica la familia y el modelo de *DSP* al cual representa este bloque, para luego hacer reportes de cómo se comportaría el dispositivo real. Este reporte se creará a partir del programa Quartus II, así también con ModelSim.

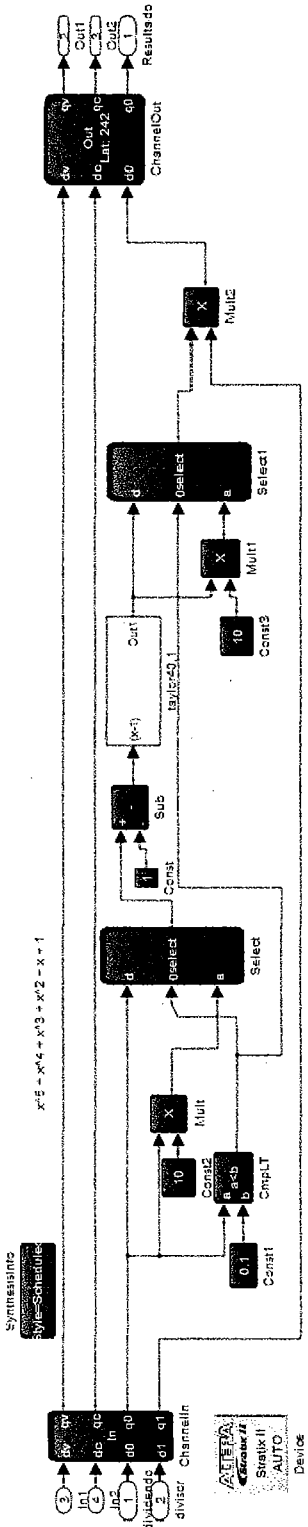


Figura 4. 6 Subsistema del modelo

El diagrama anterior contiene los siguientes bloques:

Bloque "*Device*"

Se utiliza para marcar un subsistema específico de Simulink como el grado más alto de un dispositivo *DSP* y establece las opciones de este.

Los parámetros que se le asignaron son:

- Familia del Dispositivo: *Stratix* II
- Miembro de Familia: Automático
- Grado de velocidad: -3

Bloque "*Channelin*"

Establece el límite de valores de entrada de un subsistema primitivo. Este bloque entrega sus valores atreves de él hacia la salida sin presentar cambios. El propósito principal es para indicar a la herramienta que estas señales llegarán sincronizadas desde su fuente, de esta manera la herramienta de sintetizar puede interpretarlos correctamente. Para este caso tiene dos entradas de señales de datos, que vienen del exterior. Y son identificadas como "d0" y "d1".

Bloque *"Synthesisinfo"*

Controla el flujo de síntesis de este modelo. El valor se ha colocando en: "*Scheduled*" o programado. Lo que hace es que las señales estén trabajando de una manera sincronizada.

Bloque *"ChannelOut"*

Delimita los valores de salida de un subsistema primitivo. Este bloque pasa sus entradas atreves de él hacia la salida sin presentar cambio. El propósito principal es de indicar a la herramienta que las señales deben ser sincronizadas con la salida de este bloque. Este bloque garantiza esta sincronización. Es la interfaz de salida del dispositivo, la encargada de entregar las señales tal como han sido procesadas.

Otro tipo de información que se obtiene por medio de este bloque, es el valor de retraso presentado en el dispositivo debido al procesamiento, este se conoce como Latencia. De esta manera se puede diseñar y simular con datos que afectaran el dispositivo real.

Es necesario entonces que el tiempo de ejecución del modelo sea igual o mayor a este valor de latencia. En este modelo la latencia fue calculada a un valor de 242, por eso el tiempo de ejecución se ajustó a 250, para que dé tiempo a pasar la información al exterior.

Tiene una entrada y una salida, ésta es la que lleva el resultado de la operación hacia los bloques que mostrarán el valor.

4.1.3 Funcionamiento del modelo.

Se trabajará con la señal "q0" que es la salida del bloque "channelin" para calcular su valor inverso. Para luego multiplicarlo por "q1", que es la otra salida de este bloque. La señal "q0" encuentra dos caminos, uno es hacia el bloque "*select1*" y el otro hacia un multiplicador y comparador. Se coloca el comparador primeramente para verificar si el valor de la señal es menor que 0.1, a la salida del comparador hay una señal booleana (1 o 0 lógico) que activa el bloque de selección de señales, si la salida del comparador es 0 este bloque dejará pasar la señal establecida en la entrada "d", en este caso el numéro es mayor de 0.1 y será procesado por el algoritmo de la aproximación de su inversa.

Por otro lado si el valor de "q0" es menor que 0.1 el comparador tendrá una salida 1, lo que hará que el selector escoja la señal en la entrada "a". Y de la misma manera será calculado su inverso. La señal en "a" del selector es el resultado de escalar el valor de "q0" a un rango al cual el algoritmo pueda aproximar su valor inverso. Este escalonamiento se hace multiplicando la señal "q0" por un factor de 10.

Cabe mencionar que el rango para este escalonamiento es de 0.01 a 0.09999, si se desea tener rangos menores a este, es necesario incluir otros bloques que hagan la escala a cada rango deseado. Puede ser multiplicar por factores de 10^n, donde n es un número entero positivo o negativo.

En este trabajo solo se incluyen dos rangos, con el propósito de demostrar el funcionamiento del algoritmo. Posteriormente, la señal que sale del selector será procesada con una sección que calcula $(x-1)$, antes de ser introducida a al subsistema "Taylor40,1", en éste se lleva a cabo las operaciones necesarias para realizar la serie de Taylor. Como su nombre lo indica es una serie de grado 40 alrededor de 1.

A la salida del bloque generador de la serie, se encuentra otra vez un bloque selector, que arregla la escala, en caso que se haya usado en el primer selector. Se vuelve a multiplicar por un factor de 10 la señal seleccionada para que esté en la escala correcta. A este punto ya se tiene la señal $1/a$. A continuación, simplemente se multiplica por el valor de la señal "q1" para obtener el resultado de la división. Este valor pasa por "*Channelout*".

4.1.3.1 Implementación de la serie de Taylor

En esta sección se describirá los distintos procesos que se realizaron para obtener la aproximación a la serie de Taylor, con el fin de calcular el inverso de un número.

Subsistema Taylor 40,1

En este bloque se tiene por señal de entrada un valor igual a $(x-1)$ y en la salida el resultado del cálculo de la serie. La gráfica Fig.4.7 muestra el esquema en que se trabajó este subsistema.

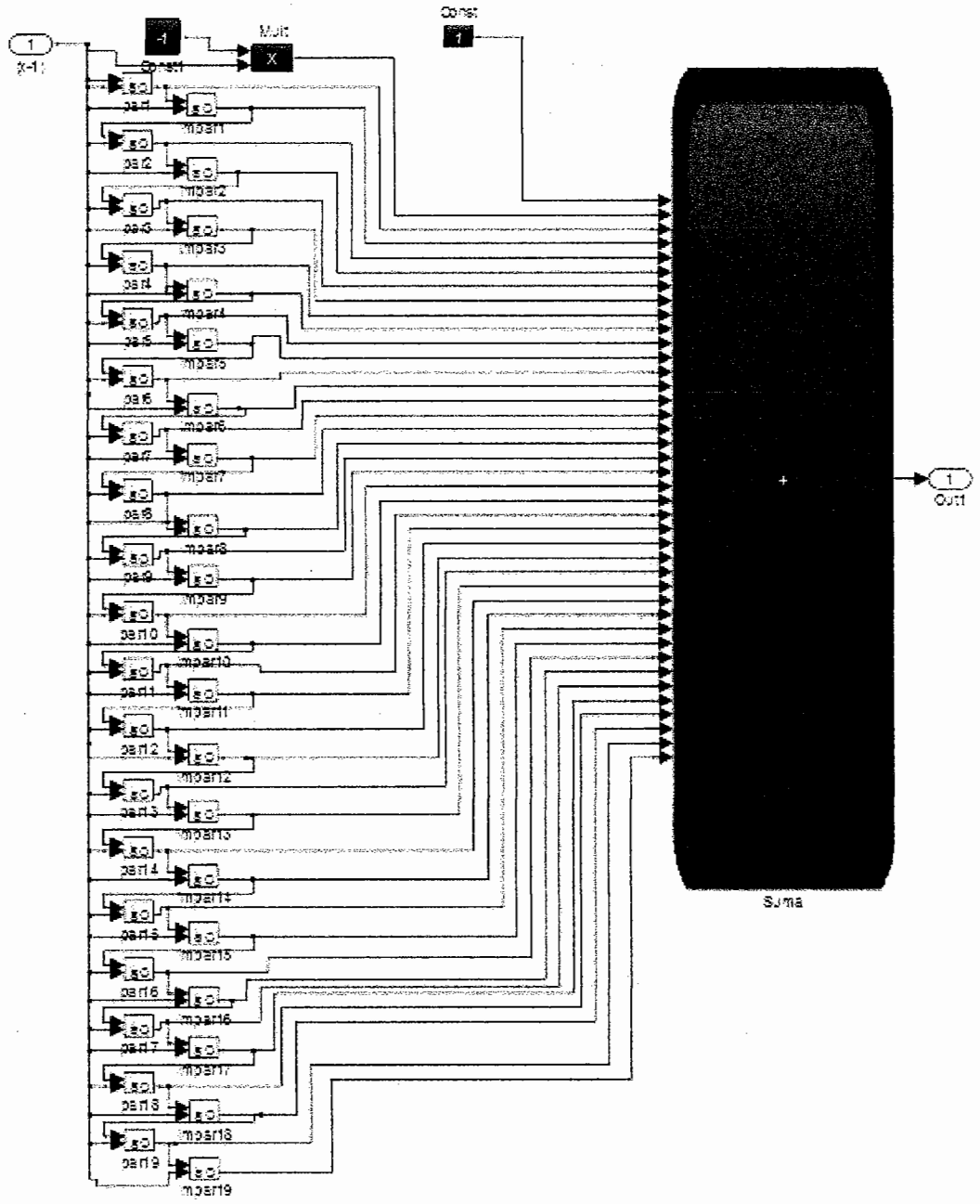
Se visualiza el uso de varios bloques pequeños, estos calculan el valor de x^w donde w es el grado de la serie, en este caso es 40. El procedimiento para lograr esto, es multiplicar el resultado de la multiplicación anterior por el número de entrada así: $x^4 = x^3 * x = (x^2 * x) * x = x * x * x * x$

La salida de estas multiplicaciones son todas sumadas por el bloque "Suma" a este se le agrega el valor de 1 en la suma. Al calcular la suma el valor es resultado de la aproximación que se busca.

Un punto importante por resaltar es que en la serie, hay componentes de exponente par e impar, y resulta que los de exponente impar deben ser restados de los componentes pares, que son sumados. La serie se muestra así: $(x-1)^4 - (x-1)^3 + (x-1)^2 - x + 2$, entonces tomando en cuenta que los valores de exponentes impares se pueden representar de la siguiente manera: $-x^7 = |x^7|$, esta igualdad se cumple si x es un número negativo. Se crearon dos tipos de subsistemas que representarán los componentes pares e impares.

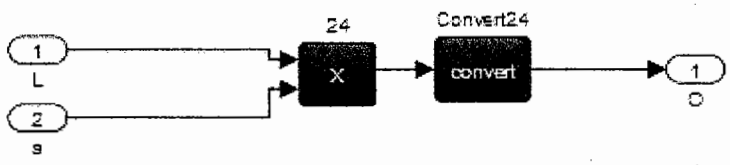
Esto resulta bien debido a que los valores de x por el momento están restringidos a ser menores que la unidad, y como la operación $(x-1)$ asegura que los valores vayan a ser negativos, la anterior igualdad se cumplirá siempre.

Figura 4. 7 Implementación de la serie



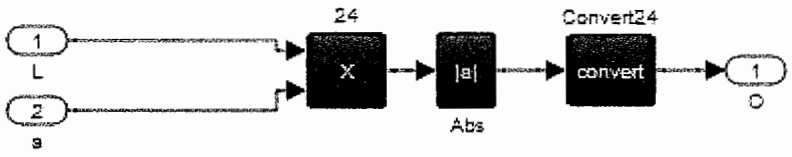
Queda por describir el diseño de los subsistemas para multiplicaciones pares e impares, las siguientes gráficas ilustran el modelo:

Figura 4. 8 Componentes pares



La figura 4.8 es la implementación de la multiplicación para componentes pares. Consiste de un bloque que realiza la multiplicación más un bloque convertidor, este bloque es de mucha importancia ya que asegura que los tipos de variables no cambien. Es decir, que no aumente la longitud de la palabra, y que se mantenga en 16 con una parte fraccional de 15. También realiza un tipo de aproximación por redondeo para preservar el valor de manera que no cambie mucho.

Figura 4. 9 Componentes impares



La figura 4.9 es la representación de los componentes de exponente impar, en este modelo se puede ver como se trabaja con la igualdad antes descrita para mantener el

valor absoluto, y de esa manera no tener que realizar la resta entre estos tipos de componentes. De igual manera está compuesta de un bloque multiplicador, valor absoluto y un convertidor, por las mismas razones descritas en el párrafo anterior.

4.1.4 Pruebas y simulación del modelo.

Ahora que se conoce el modelo y qué hace cada parte, se procederá a simular el diseño.

Para poder simular se necesitan tener dos valores de entrada.

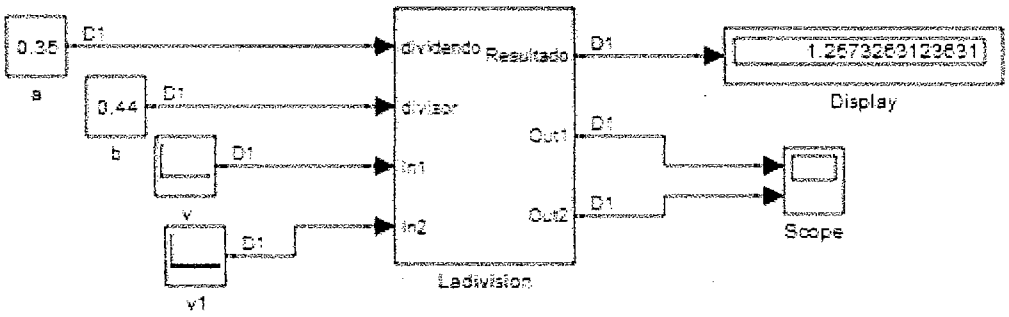
Los valores a probar serán:

Tabla IV. Resultados del modelo

Valor de a	Valor de b	Operación	Resultado	Valor obtenido	Gráfica
0.35	0.44	0.44/0.35	I.257I	I.257326	Fig.4.I0
0.075	0.5	0.5/0.075	6.6667	6.66549682	Fig.4.II
0.2565	0.333	0.333/0.2565	I.2982	I.29825	Fig.4.I2

La gráfica de la primera simulación es la siguiente:

Figura 4. 10 Resultado de la división 1



Continuación de los resultados:

Figura 4. 11 Resultado de la división 2

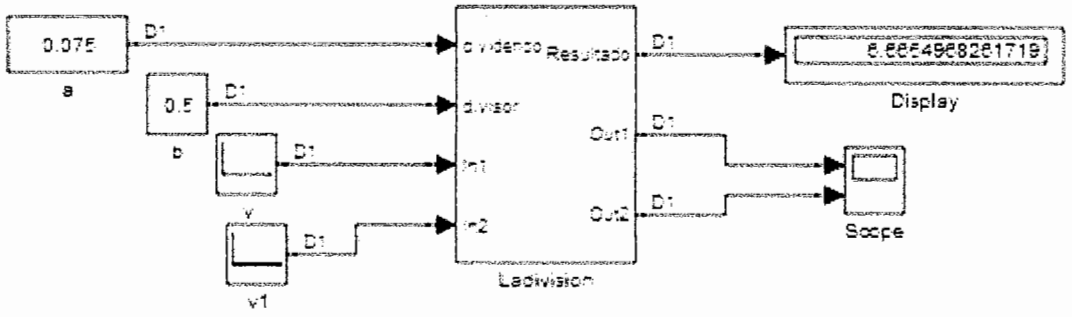
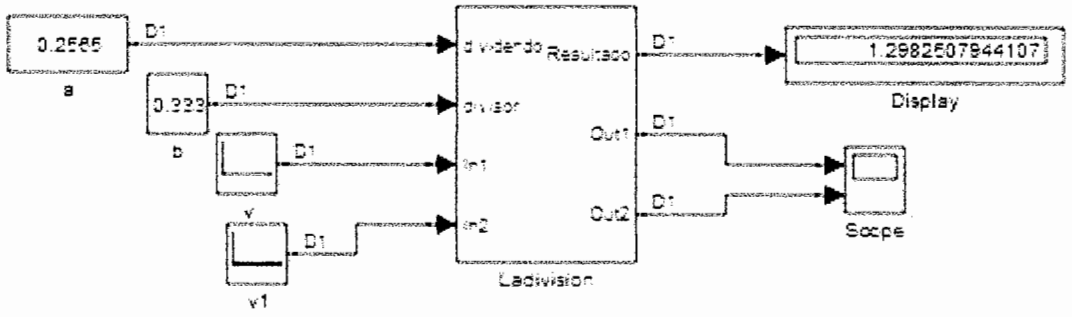


Figura 4. 12 Resultado de la división 3



Para comprobar gráficamente si se ajusta a las gráficas de la función 1/x, se pondrá un bloque que represente una señal creciendo una cantidad dada desde 0.01 hasta1.

Continuación de los resultados:

Figura 4. 13 Representación lineal del variable tiempo

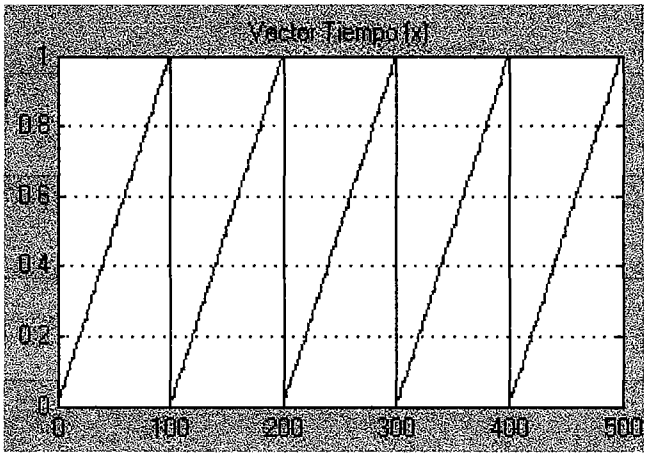
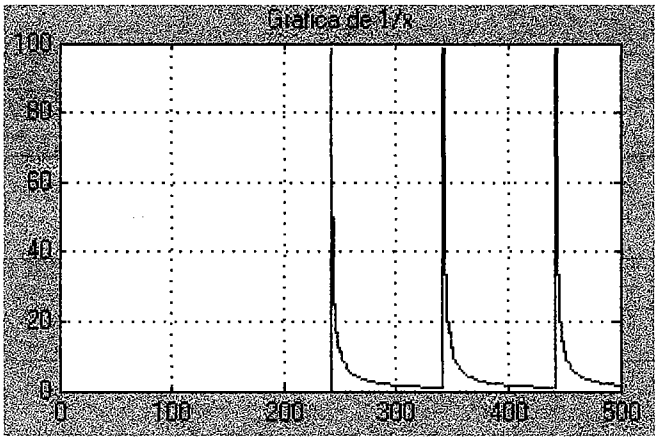


Figura 4. 14 Respuesta del inverso de un número



A continuación se visualiza la ejecución del modelo:

Figura 4. 15 Modelo en ejecución

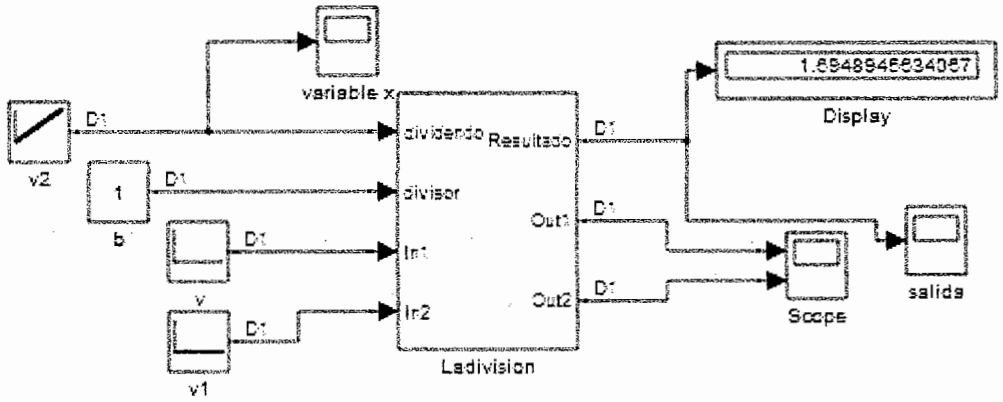
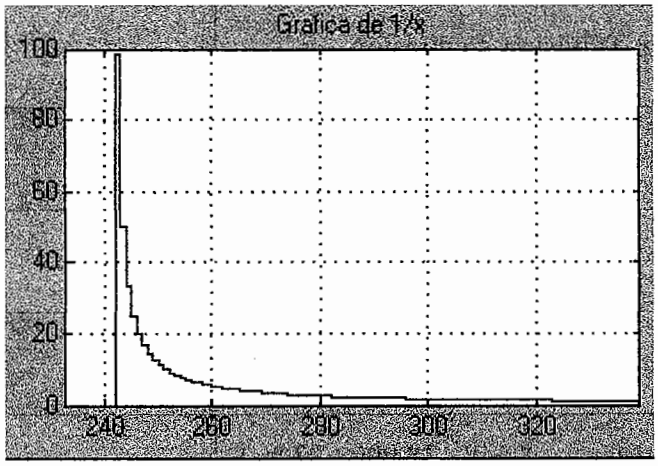


Figura 4. 16 Gráfica de 1/X



Ahora que se ha mostrado el funcionamiento y los resultados de la simulación, se procede a crear reportes en el programa Quartus II. Este será una herramienta de mucha ayuda. Una vez terminada la simulación, se ejecuta el programa Quartus II, en el bloque que se incluyó en Simulink.

Una vez cargado el modelo como un nuevo proyecto en Quartus II, se procede a compilarlo.

A continuación se presentan los resultados y resúmenes de la compilación, así como datos de interés.

- **Tabla V. Resumen de compilación: se muestran datos de recursos utilizados.**

Flow Status	*Successful* - Mon Jun 07 12:14:45
Quartus II Version	9.1 *Build* 350 03/24/2010
Revision Name:	Ladivision
Top-level Entity Name:	Tdivision_Ladivision
Family:	Stratix III
Logic utilization	32 %
Combinational ALUTs	5,177 / 38,000 (14 %)
Memory ALUTs	2,632 / 19,000 (14 %)
Dedicated logic registers	8,648 / 38,000 (23 %)
Total registers	8648
Total pins	103 / 296 (35 %)
Total block memory bits	68,736 / 1,880,064
DSP block 18-bit elements	156 / 216 (72 %)
Device	EP3SL50F484C2

- Resumen del análisis y síntesis:

En este resumen se muestra cuantas unidades de *DSP* se utilizaron, que son 156, que trabajan a 18 bits.

Tabla VI. Resumen del análisis y síntesis

Analysis & Synthesis Status *Successful* - Mon Jun 07 11:59:15 2010
Quartus II Version 9.1 *Build* 350 03/24/2010 SP 2 SJ *Web Edition*
Revision Name Ladivision
Top-level Entity Name *Tdivision_Ladivision*
Family *Stratix* III
Combinational ALUTs 5,177
Memory ALUTs 2,632
Dedicated logic registers 7,323
Total registers 7323
Total pins 103
Total block memory bits 68,736
DSP block 18-bit elements 156

Continuación de la tabla VI.

Recursos	Usados
Total fan-out	70164
Total block memory bits	68736
Total MLAB memory bits	33558
Maximum fan-out	10599
Estimated ALUT/register pairs used	8768
Estimated ALUTs Used 7809	
-- *Combinational ALUTs*	5177
-- *Memory ALUTs*	2632
-- *LUT_REGs*	0
Total registers	7323
-- *Dedicated logic registers*	7323
-- *LUT_REGs*	0
-- *I/O registers*	0
Dedicated logic registers	7323
Total combinational functions	5177
DSP block 18-bit elements	156
I/O pins 103	

Tabla VII. Resumen de uso de memoria *RAM*

USO DE MEMORIA *RAM* POR BLOQUE	Port A		Port B		
Nombre	epth	idth	epth	idth	ize
Id_ChannelIn_In2_to_ChannelOut_Out2_replace_mem_dmem|	38		38		904
Id_ChannelIn_divisor_to_Mult2_replace_multhi_b_replace_mem_dmem|	34	6	34	6	744
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_Impar8_24	2	7	2	7	394
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_Impar9_24	4	7	4	7	598
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_Impar10_24	06	7	06	7	802
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_Impar11_24	18	7	18	7	006
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_Impar12_24	30	7	30	7	210
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_Impar13_24	42	7	42	7	414
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_Impar14_24	54	7	54	7	618
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_Impar15_24	66	7	66	7	822
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_Impar16_24	78	7	78	7	026
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_Impar17_24	90	7	90	7	230
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_Impar18_24	02	7	02	7	434
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_Impar19_24	14	7	14	7	638
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_par8_24	6	7	6	7	292
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_par9_24	8	7	8	7	496
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_par10_24	00	7	00	7	700
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_par11_24	12	7	12	7	904
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_par12_24	24	7	24	7	108
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_par13_24	36	7	36	7	312
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_par14_24	48	7	48	7	516
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_par15_24	60	7	60	7	720
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_par16_24	72	7	72	7	924
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_par17_24	84	7	84	7	128
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_par18_24	96	7	96	7	332
Id_taylor40_1_Impar1_24_replace_split_low_b_to_taylor40_1_par19_24	08	7	08	7	536

Tabla VIII Bloques de DSP usados

Resumen de Bloques de *DSP* usados	
Estadística	
Valores	
Simple Multipliers (36-bit)	1
Simple Multipliers (18-bit)	76
Signed Multipliers	77

Tabla IX Registros

Resumen de Registros
Total registers
7323
Number of registers using Synchronous Clear
0
Number of registers using Synchronous Load
80
Number of registers using Asynchronous Clear 4788

Tabla X. Resumen de ajustes

Fitter Status	*Successful - Mon Jun 07 12:12:54 2010*
Quartus II Version 9.1 Build 350 03/24/2010 SP 2 SJ Web Edition	
Revision Name	*Ladivision*
Top-level Entity Name	*Tdivision_Ladivision*
Family	*Stratix III*
Device	EP3SL50F484C2
Timing Models	*Final*
Logic utilization	*32 %*
Combinational ALUTs	5,177 / 38,000 (14 %)
Memory ALUTs	2,632 / 19,000 (14 %)
Dedicated logic registers	8,648 / 38,000 (23 %)
Total registers	8648
Total pins	103 / 296 (35 %)
Total virtual pins	*0*
Total block memory bits	68,736 / 1,880,064 (4 %)

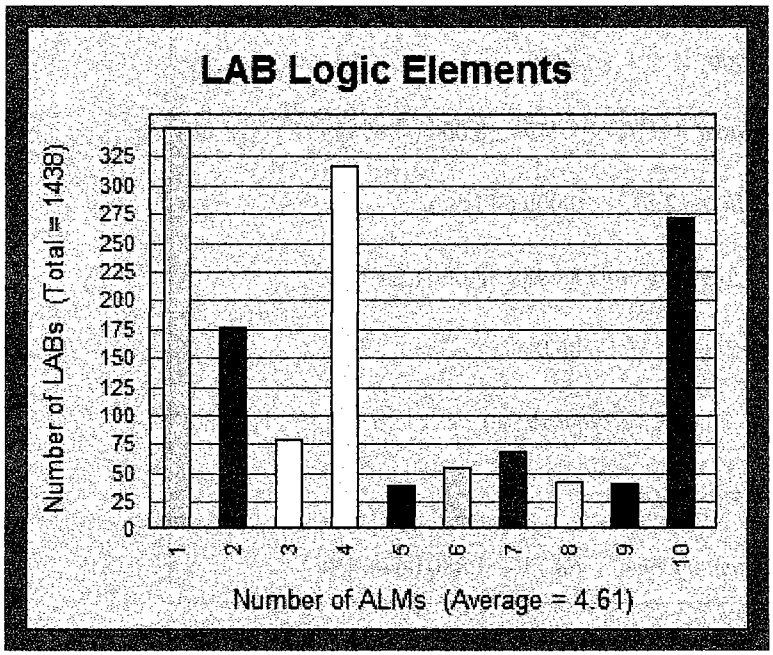
Términos importantes que se deben definir:

LAB: Es una matriz de bloques lógicos. Estos consisten de una matriz de celdas lógicas.

ALM: (*Adaptive Logic Module*): modulo de lógica adaptable, es el bloque básico para ciertos dispositivos y es diseñado para maximizar su rendimiento y el uso de recursos. Cada *ALM*, está compuesto de dos Buscar en tabla (ALUT), y puede soportar hasta 8 entradas y cuatro salidas. Muy eficaz para usar en operaciones aritméticas.

A continuación se presentan gráficas del uso de *LABs* (*logical array block*)

Figura 4. 17 Gráfica de los elementos *LAB*



Continuación de las graficas que representan uso de datos y recursos

Figura 4. 18 Ancho de las señales *LAB*

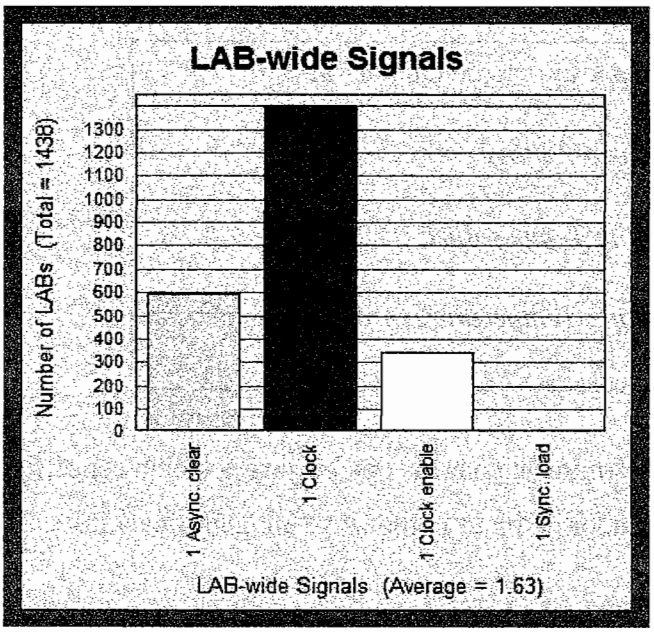


Figura 4. 19 Salidas de las señales *LAB*

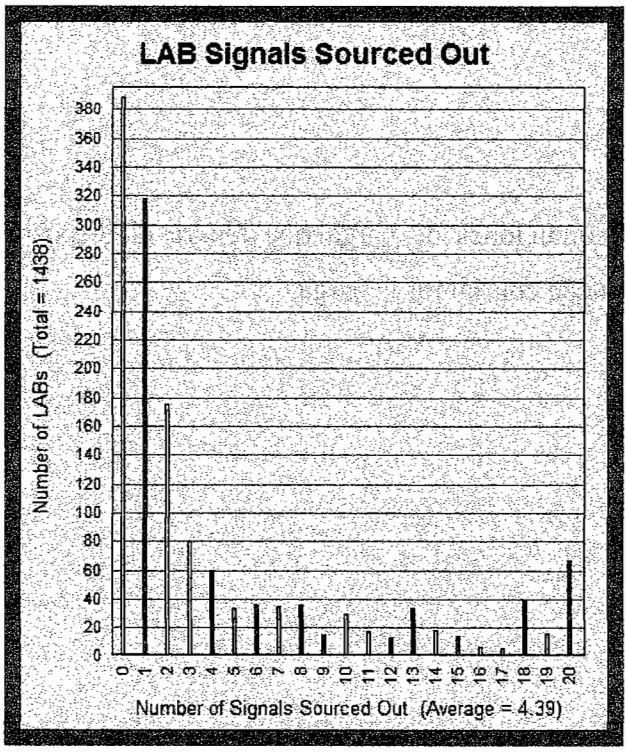


Tabla XI Condiciones y ajustes

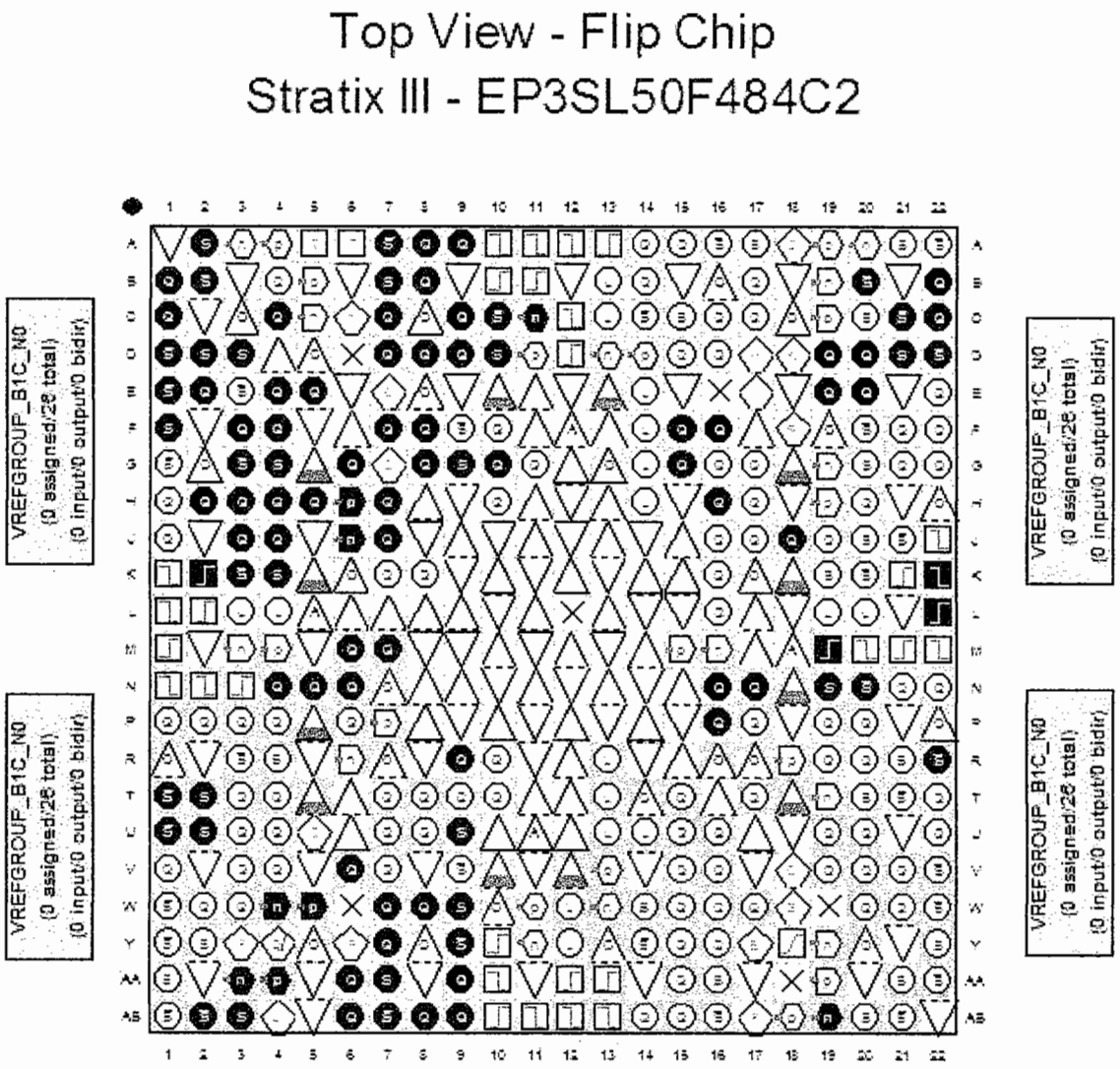
Condiciones y ajustes de Operación	
Nominal Core Voltage	1.10 V
Low Junction Temperature	0 °C
High Junction Temperature	85 °C

El programa de Quartus II permite asignar los pines a una posición específica en el dispositivo, por medio del planificador de pines como se ve en la gráfica 4.20:

Los pines que están resaltados están asignados. Los diferentes símbolos indican el tipo de dato que contienen, tanto si son de entrada o salida, o señales de reloj, señal de tierra o voltaje de referencia.

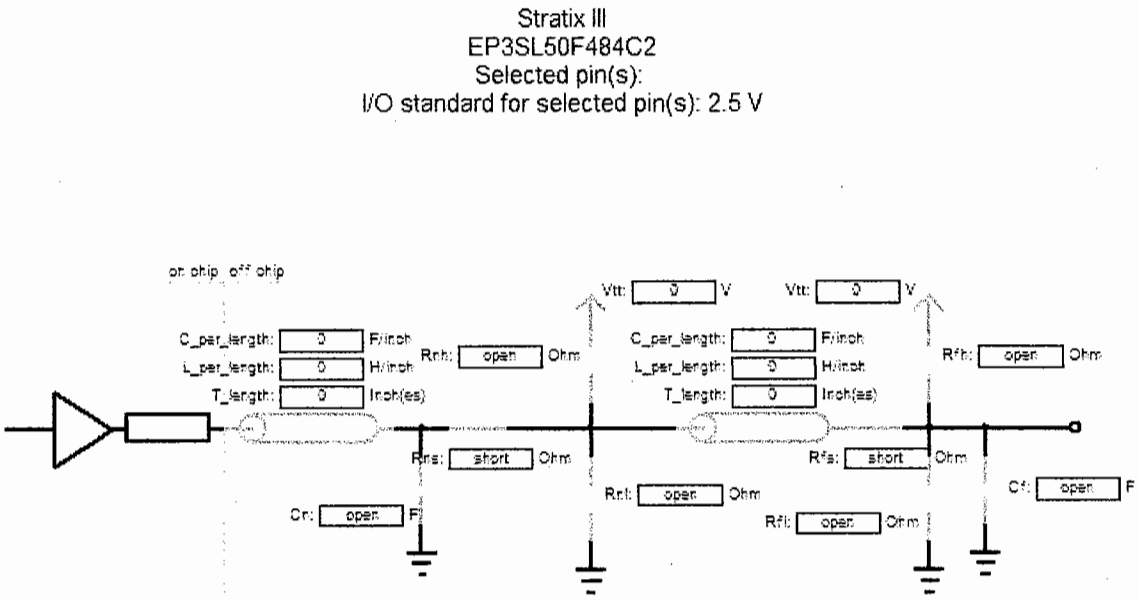
También permite editar las condiciones de carga que tendrá el dispositivo, la siguiente gráfica muestra como se puede configurar la carga.

Figura 4. 20 Distribución de pines en el *DSP*



Continuación de gráficas del diseño:

Figura 4. 21 Ajustes de los valores de carga



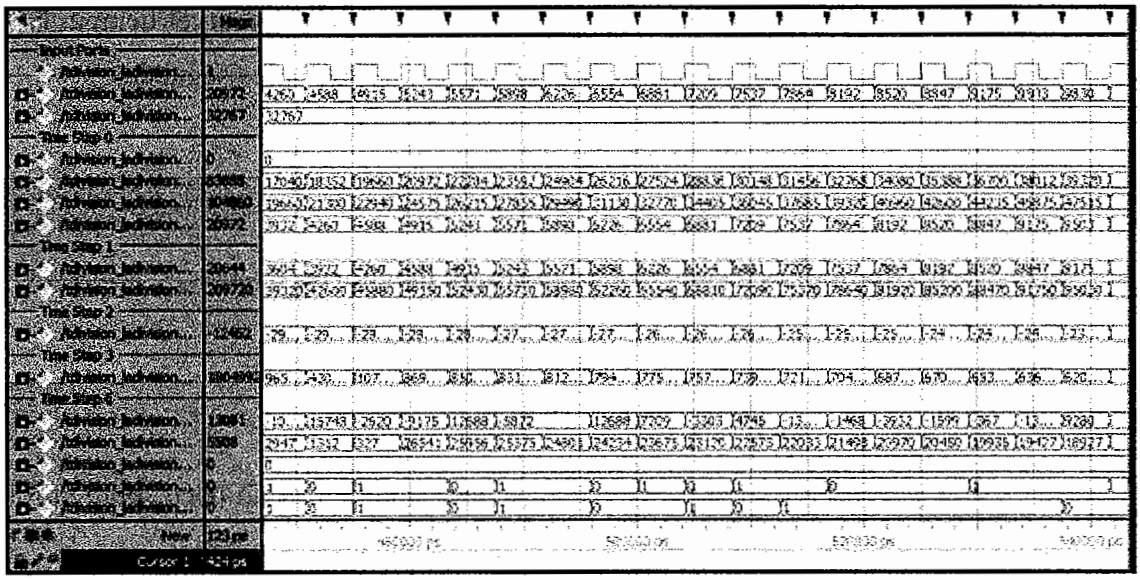
Para visualizar las señales se utiliza el programa Model Sim

Las siguientes gráficas muestran el movimiento de las señales dentro del modelo, en forma digital. Primero se muestra las señales de reloj y de entrada. El formato en que se visualizan es por evento.

Los números que aparecen encerados, son el valor decimal de la suma de cada bit, estos por supuesto solo son una representación decimal, ya que en la simulación son variables de punto flotante.

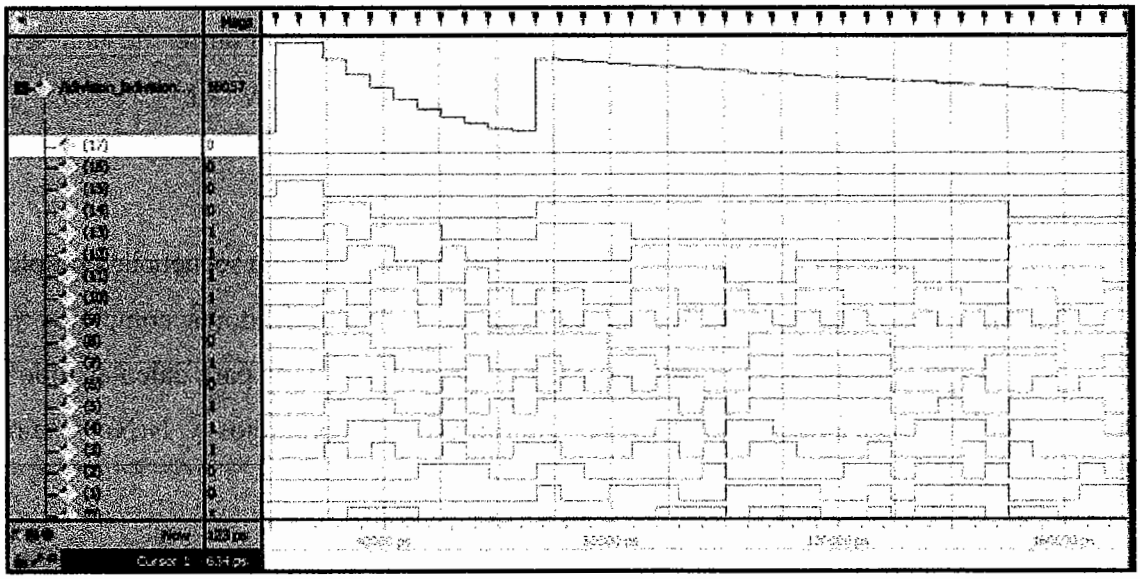
Gráficas de resultados de operaciones.

Figura 4. 22 Gráfica los procesos en el *DSP*



Para tener una vista de cómo se comportaría las señales en una forma continua, y cuál es el valor de cada bit se muestra la siguiente figura.

Figura 4. 23 Detalle de los valores por Bit



Aquí la gráfica superior es el valor de la suma de los 17 bits. Y en las gráficas inferiores se muestran los valores de los bits y cómo cambian en el tiempo. Nótese la escala del tiempo, desde 20300 pico segundos hasta 170000 pico segundos.

4.2 Algoritmo para Transformada Rápida de Fourier

Para poder desarrollar este modelo se necesita explicar un poco sobre la forma en que este algoritmo funciona. El modelo simulará un sistema que realice un *FFT* de 512 puntos de muestra.

4.2.1 Introducción

La manera de representar la transformada discreta de Fourier de una secuencia finita de tamaño N es: $X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn},\ k = 0, 1, \ldots\ldots, N-1$

Y se tiene que $W_N = e^{-j2\pi/N}$ donde x[n] puede ser una secuencia compleja, por lo tanto si se quiere evaluarla se requerirá de N multiplicadores complejos y (N-1) sumadores complejos, para calcular cada valor de la *DFT*. Y para calcular todos los N valores es necesario un total de N^2 multiplicadores complejos y N(N-1) sumadores complejos.

Ahora para realizar un multiplicador complejo se requiere cuatro multiplicadores reales y dos sumadores. Y cada sumador complejo está compuesto por dos sumadores reales.

Debido a esto, es evidente que el número de operaciones aritméticas requeridas para calcular la *DFT* por medio de este método directo se convierte muy extenso para valores grandes de N. Por esta razón se plantea otra forma de calcular, se busca que sea más eficiente y que requiera menor cantidad de multiplicadores y sumadores.

4.2.2 Algoritmo de partición en el tiempo

Estos algoritmos están basados en el principio de descomponer el cálculo de *DFT* de tamaño N en una serie de pequeñas transformadas discretas de Fourier. Esto se logra aprovechando la simetría y la forma periódica de $W_N = e^{-j2\pi/N}$. En el caso especial que N sea un entero de potencia de 2, se calcula X[k] en dos secuencias de (N/2) puntos, que consisten en una serie de números pares y una impar. Se puede escribir como:

$$X[k] = \sum_{r=0}^{\left(\frac{N}{2}\right)-1} x[2r]W_N^{2rk} + \sum_{r=0}^{\left(\frac{N}{2}\right)-1} x[2r+1]W_N^{(2r+1)k}$$

$$= \sum_{r=0}^{\left(\frac{N}{2}\right)-1} x[2r]W_N^{2rk} + W_N^k \sum_{r=0}^{\left(\frac{N}{2}\right)-1} x[2r+1]W_N^{(2r)k}$$

Ecu. 4.2

Pero $W_N^2 = W_{N/2}$

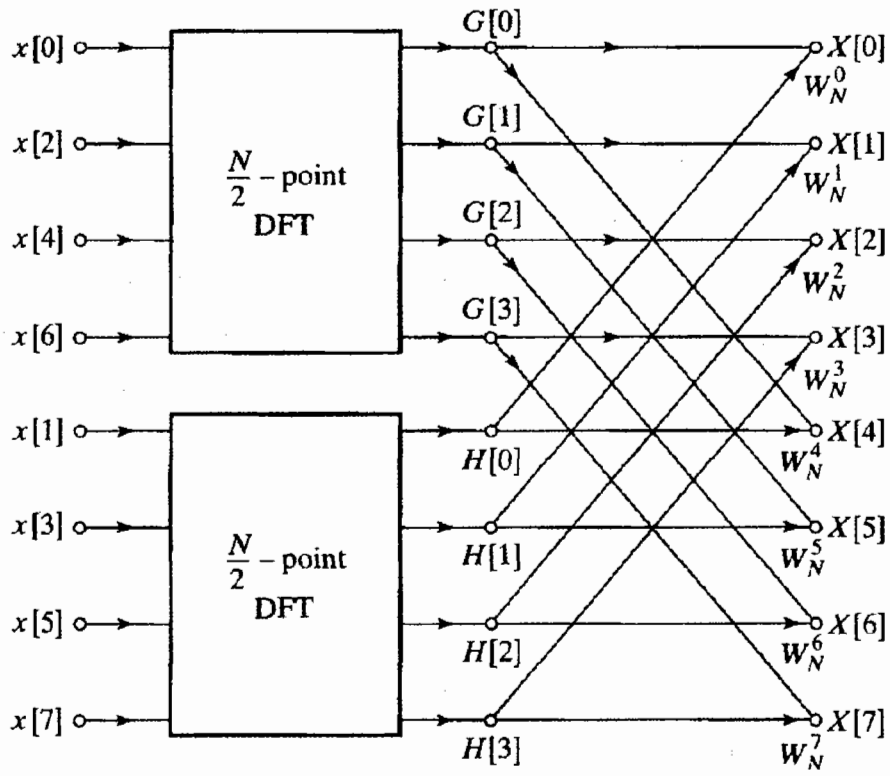
Se puede reescribir la ecuación como:

$$= \sum_{r=0}^{\left(\frac{N}{2}\right)-1} x[2r]W_{N/2}^{rk} + W_N^k \sum_{r=0}^{\left(\frac{N}{2}\right)-1} x[2r+1]W_{N/2}^{(r)k}$$

$$= G[k] + W_N^k H[k], k = 0, 1, \ldots, N-1$$

Cada una de las sumas es un *DFT* N/2 puntos, la primera suma corresponde a la *DFT* de los puntos pares de la secuencia original, y la segunda a los números impares.

Figura 4. 24 Algoritmo para calcular FFT



Fuente: Oppenheim shafper & buck
Discrete-time digital signal processing
Pag, 661

En la figura anterior se calcula un *DFT* con N=8. Se calculan 2 *DFT* de 4 puntos, con G[k] siendo el *DFT* de los números pares y H[k] de los impares. Para encontrar X [0] se multiplica H[0] por Wn y se suma con G[0], y así sucesivamente para el resto de los valores.

Ahora la razón de hacer esto en un principio, era para disminuir el número de componentes necesarios para realizar la *DFT*.

Con este algoritmo se necesitan $N+N^2/2$ multiplicadores y sumadores complejos, este valor resulta ser menor para N>2 que la cantidad N^2 que se requerían si se hace de la manera directa.

4.2.2.1 *FFT Butterfly*

Este término se refiere a una *DFT* de tamaño 2 que toma dos entradas (x[0], x[1]) y da dos salidas (X[0], X[1]) por la fórmula:

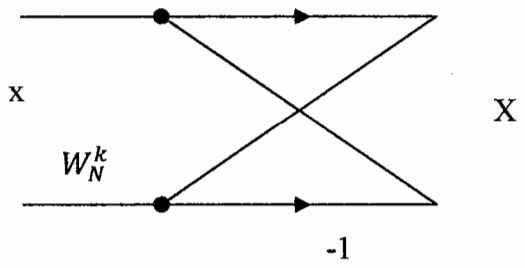
$$X[0] = x[0] + x[1]w^k$$

$$X[1] = x[0] - x[1]w^k$$

Ecu. 4.3

Y el diagrama se muestra en la figura 4,25:

Figura 4. 25 *Butterfly* para FFT



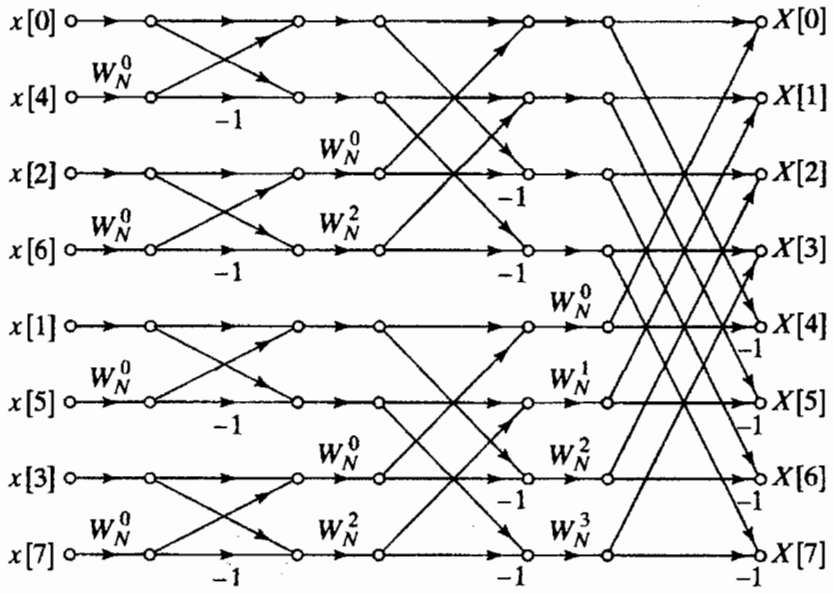
4.2.2.2 Inversión de bits

Esta es la forma en que se ordena y acceden los datos de cada etapa de la *FFT* al algoritmo.

La inversión de bits es la permutación donde el dato en el índice n, en binario b4b3b2b1b0 (para N=32), es cambiado a la siguiente forma b0b1b2b3b4.

Para entender porque es necesario esto, se ilustra la siguiente figura.

Figura 4. 26 Diagrama completo para calcular una FFT

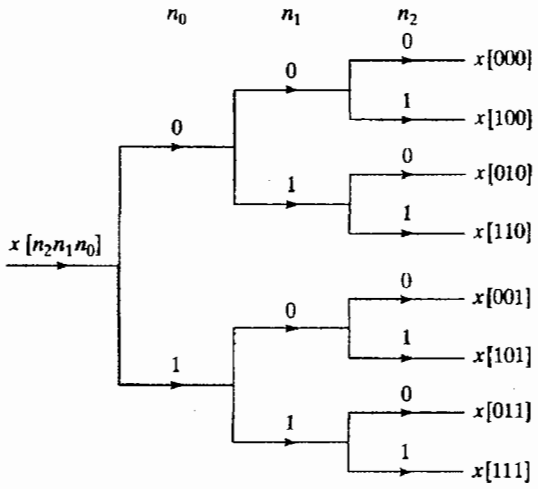


Fuente: Oppenheim shafper & buck
Discrete-time digital signal processing
Pag, 664

Se observa que la secuencia x[n] fue dividida primero en las muestras de numeración par, y seguidamente se colocaron en la parte superior del diagrama, y los de numeración impar en parte inferior. Esta separación de datos se puede alcanzar examinando el bit menos significativo b0. Si el bit menos significativo es 0 entonces se trata de un número par, luego si es 1 el número es impar. Si el segundo bit menos significativo es 0, el valor de la secuencia es un número par en la sub secuencia. Y si este es 1 es un número impar en la sub secuencia. Este proceso se repite hasta que N sub secuencias de longitud 1 se obtengan.

Esto se puede ver en diagrama de la fig.4.27

Figura 4. 27 Diagrama del orden de los bits



Fuente: Oppenheim shafper & buck
Discrete-time digital signal processing
Pag, 667

4.2.3 Algoritmo transformada rápida de Fourier partición en la frecuencia

El algoritmo de *FFT* por partición en el tiempo, calculaba la *DFT* por medio de formar sub secuencias cada vez más pequeñas de la secuencia de entrada. También se puede realizar este proceso a partir de las secuencias de salida. Si se separan las secuencias de muestras de número par e impar:

$$X[k] = \sum_{n=0}^{N-1} x[n]W_N^{nk}, k = 0, 1, \dots N-1$$

Ecu. 4.5

Y las muestras pares:

$$X[2r] = \sum_{n=0}^{N-1} x[n]W_N^{n2r}, k = 0, 1, \ldots (\frac{N}{2}) - 1$$

Ecu.4.6

Después de hacer un cierto procedimiento, se obtiene la ecuación general para el DFT de (N/2) puntos:

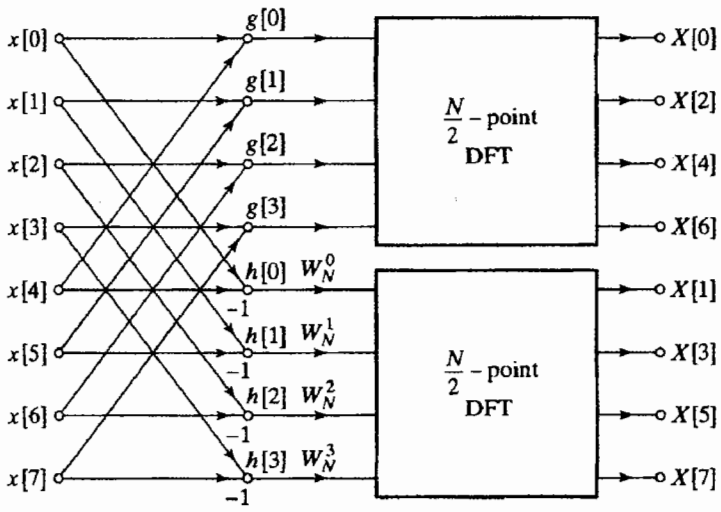
$$X[2r] = \sum_{n=0}^{N/2-1} (x[n] + x[n + (\frac{N}{2})]W_{N/2}^{rn}, r = 0, 1, \ldots, \left(\frac{N}{2}\right) - 1$$

Y para los números pares:

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

La siguiente gráfica indica la implementación de un *DFT* de 8 puntos.

Figura 4. 28 *DFT* de 8 puntos



Fuente: Oppenheim shafper & buck
Discrete-time digital signal processing Pag, 73

4.2.4 Parámetros iniciales

Para configurar el modelo se creó un archivo *setup_FFT8K*.m donde se crean las variables y valores que dan inicio al modelo.

Tabla XII Parámetros iniciales

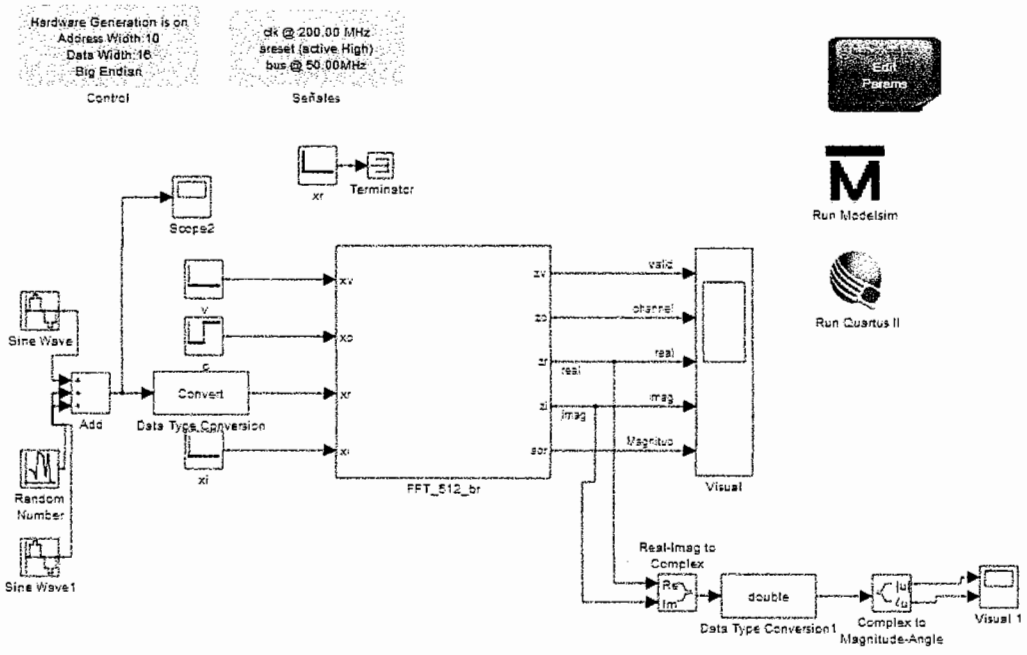
```
%% Parámetros de configuración
RataMuestra  = 1;
RataReloj    = RataMuestra*200;
MarjenReloj  = 0.0;
Periodo                =   RataReloj   /
RataMuestra;
TiempoMuestra= 1;
%% Ancho de datos
Palabra      = 16;
Fraccion     = 19;
MaxOut       = inf;
%% FFT datos
data_N_FFT           = 512;
data_L               = 4;
```

4.2.5 Modelo en Simulink del FFT de 512 puntos

La siguiente gráfica muestra el modelo para calcular la transformada rápida de Fourier.

La figura 4.29 se muestra el diagrama del modelo base, es su primera etapa.

Figura 4. 29 Modelo en Simulink para FFT



Y los datos a mostrar en Matlab son:

```
    %% Imprimir parámetros en matlab
    disp(['Parametros: ' ...
        'RataMuestra  =  '   num2str(RataMuestra)    ';  '
...
        'RataReloj = '   num2str(RataReloj)    '; ' ...
        'MarjenReloj = ' num2str(MarjenReloj) '; ' ...
        'TiempoMuestra   =   '     num2str(TiempoMuestra)
. ...
```

Los bloques que se usaron se detallarán a continuación:

Bloque Señales:

Los parámetros que se configuraron son:

- Señal de reloj como: clk
- Frecuencia de reloj: RataReloj

Bloque Control:

- Ancho del sistema de direcciones: 10
- Ancho en el sistema de datos: 16
- Tipo de bus: *Big Endian*

En esta primera etapa se distinguen tres tipos de bloques:

- Bloques de señales de entrada y control.
- Bloque que representa el dispositivo *DSP*
- Bloques de salida y visualización
- Bloques de verificación

Para poner a prueba el modelo se creó un conjunto de señales a las cuales se les calculará su trasformada de Fourier. Esta señal está compuesta de dos ondas de función seno a diferentes frecuencias y de un generador de números aleatorios, para producir un fenómeno de ruido. Todas estas señales son sumadas. A la salida del sumador ahí un visualizador de la señal resultante. También hay un convertidor de tipo de señal, normalmente las señales de interés serán de tipo doble; y el *DSP* trabaja con tipo punto fijo.

Bloque Conversión de tipo de datos:

Los parámetros son:

- Tipo de salida de dato: *sfix*(16), y valor de fracción de 19.
- Tiempo de Muestra: se configuró a que sea igual a TiempoMuestra
- Modo de estimación: hacia el más bajo (*Floor*)

Esto es para una señal real, por lo que va a la entrada del bloque *FFT_512* llamada xr. Para señales imaginarias estas tienen una entrada xi, pero en la mayoría de casos se trabaja con señales reales.

4.2.5.1 Sistema de verificación

Este consiste en un conjunto de bloques que realizarán una comparación de la señal que se debería obtener a la salida del *DSP*, estos bloques no se incluyen en la generación de *hardware*.

Consisten en un bloque que combina señales reales e imaginarias para concatenarlas de manera que forma un número complejo, seguido de un convertidor de tipo de dato, el tipo al que se cambia es de precisión "Doble", luego un bloque que analiza la magnitud de la señal y es graficada por "Visual 1".

4.2.5.3 Bloque *FFT_512*

Seguidamente en la parte de en medio se encuentra el bloque principal, éste es la representación del dispositivo de *DSP*, dentro de él se encuentran todos los bloques necesarios para realizar el algoritmo y así calcular la transformada rápida de Fourier.

Posee puertos de 4 entradas y 5 salidas. Los puertos de entrada están titulados como:

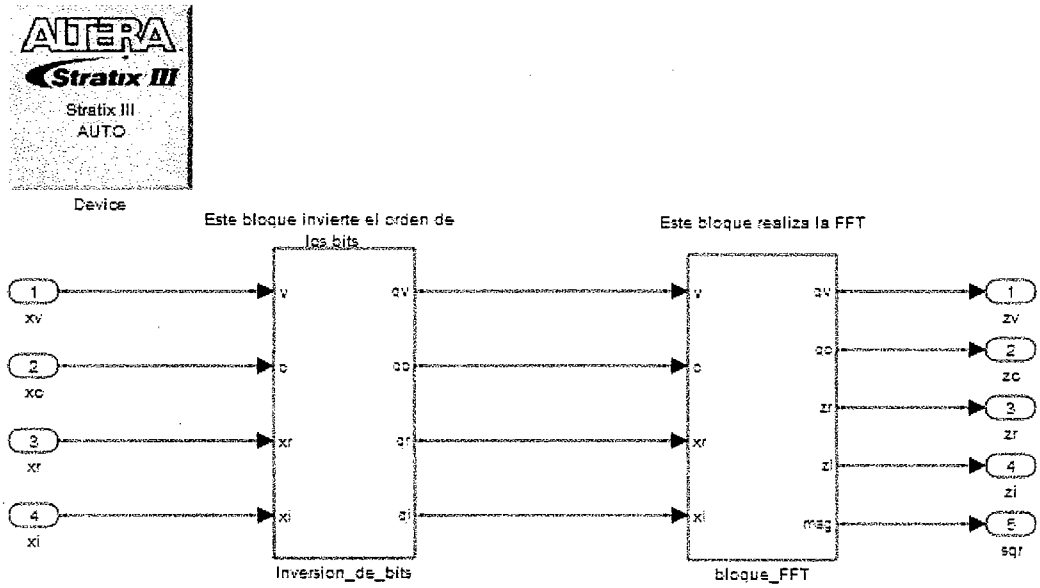
- Xv; señal para llevar control de bloque.
- Xc; señal de reloj.
- Xr; señal real
- Xi; señal imaginaria

Los puertos de salida son:

- Zv; indica cuando empieza a trabajar el *DSP*
- Zc; señal de canal
- Zr; salida de señal real.
- Zi; salida de señal imaginaria
- Sqr; salida de señal de densidad espectral de potencia

La siguiente es una figura del segundo plano del modelo:

Figura 4. 30 Diagrama de bloques principales



En esta figura se muestran tres bloques:

Bloque *Device*

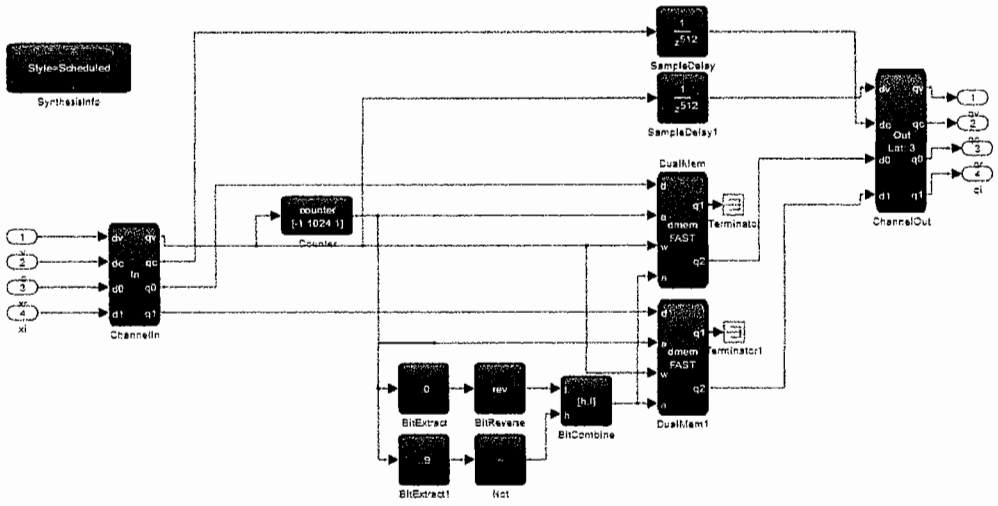
Este bloque es el que representa el dispositivo que se estará simulando. En este caso se trabaja con la familia *STRATIX* III, posteriormente se le asignará un producto específico cuando se cargue en el programa de Quartus II.

Bloque Inversión de bits

Como se mostró, parte del algoritmo para calcular *FFT* consiste en ordenar los bits de entrada. Para eso se incluye este bloque. El *Block set* de Altera ya incluye un bloque que realiza la inversión de bits. Pero no se utilizó debido a que tiene un error. Por lo

tanto aquí se corrigió el error y se muestra en su forma primitiva. En la figura 4.31 se muestra su diagrama.

Figura 4. 31 Diagrama de inversión de bits



El bloque consiste en crear un retraso y almacenar los bits invertidos en los módulos de memoria.

Los bloques de retraso tienen un valor de 512.

Bloque *FFT*

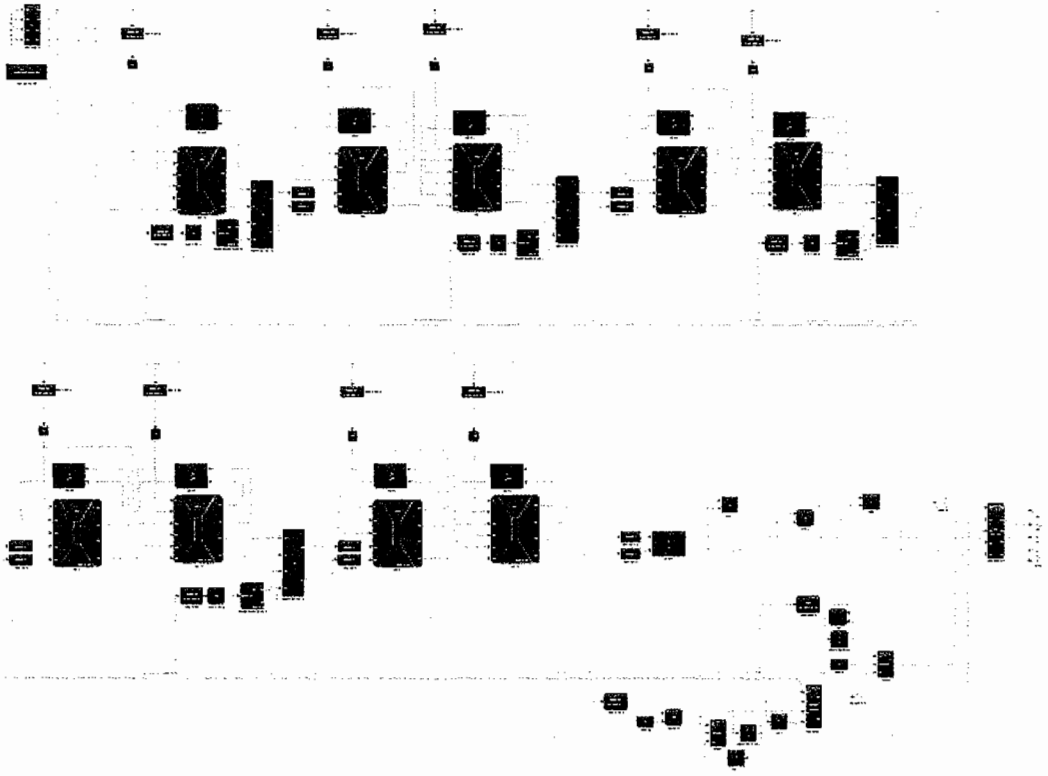
Finalmente el bloque donde se encuentra implementado el algoritmo para calcular *FFT*, sigue teniendo las entradas de señales reales e imaginarias.

Estos son los parámetros con los que se trabaja en este bloque:

- Bits de entrada: 16
- Exponente de escala de señal de entrada: 19
- Máximo ancho de bit de salida: infinito

La siguiente es la gráfica de su diagrama.

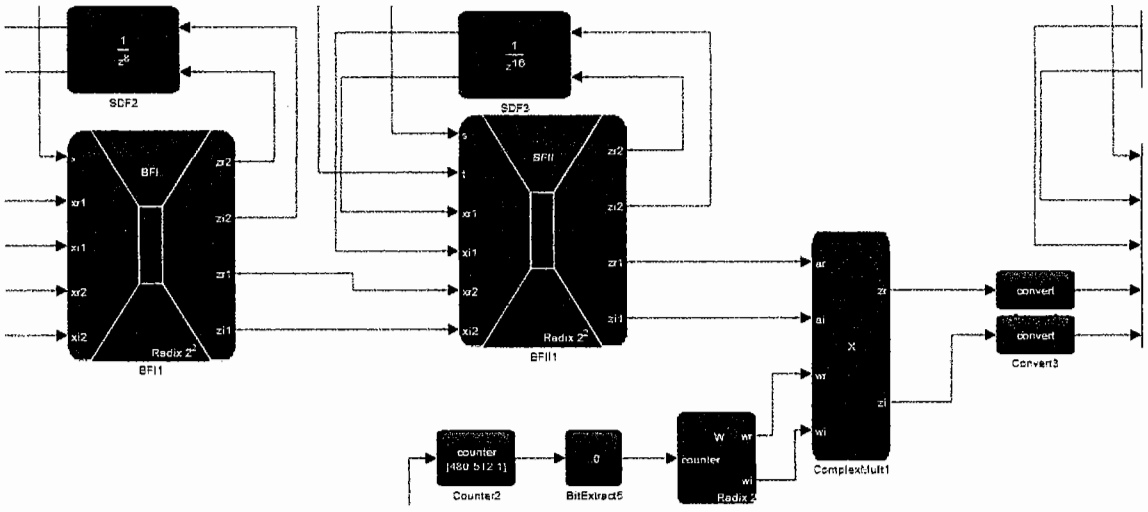
Figura 4. 32 Diagrama para calcular FFT



Este diagrama consiste de 9 bloques que implementan la función *Butterfly* asociada con la transformada de raíz 2.

Esta figura es un acercamiento a un módulo, compuesta de un bloque *butterfly* I, seguido por otro *butterfly* II;

Figura 4. 33 Uso de bloques *butterfly* tipo I y II



A la salida de este se encuentra un multiplicador de números complejos, que multiplica los coeficientes exponenciales, generados por el bloque "*TwiddleGenerator*". Este bloque tiene parámetros de entrada del tipo de dato de entrada para asegurar que la precisión de los datos se mantenga.

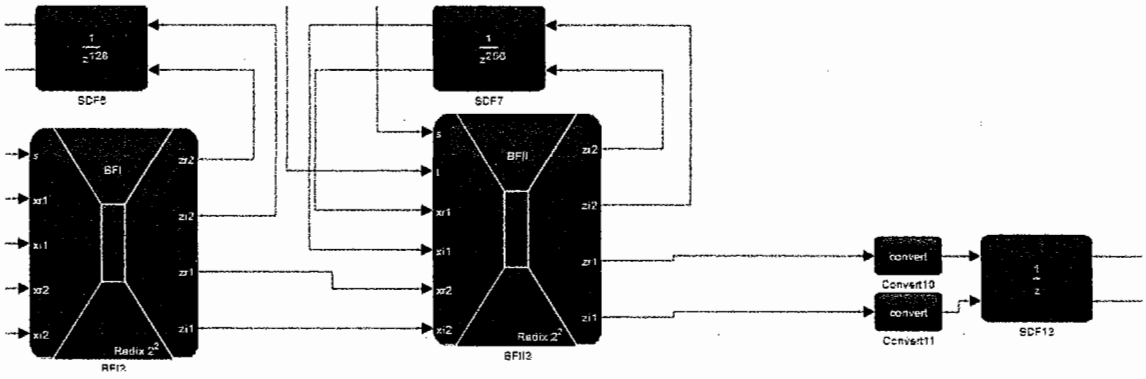
El puerto "s" es conectado a un control lógico. Este control es la extracción del bit correspondiente de un contador de módulo N. El valor "s" también determina la dirección de la señal de cada muestra y de la combinación matemática con otras muestras. El bloque "*Butterfly* 11" tiene los mismos parámetros de entrada que el anterior, con excepción de que incluye un puerto extra, el puerto "t", este se conecta también a un control lógico, pero el bit extraído difiere del usado en el puerto "s".

El valor de "t" determina cuando una multiplicación por –j ocurre dentro de la bloque del *Butterfly*. Existe un bloque de retraso por cada módulo de estos, empezando con un retraso de 1 hasta llegar a 256.

Los bloques "*Convert*" se incluyen para mantener el tamaño de la palabra y el orden de escalonamiento de la parte fraccional. Los generadores en los bloques "*TwiddleGenerator*" generan los coeficientes apropiados seno/coseno que son multiplicados por la secuencia de datos en una arquitectura de *FFT*.

Debe de tener en la entrada un contador de modulo N (un entero potencia de 2). Y la correspondiente secuencia compleja es generada en la salida.

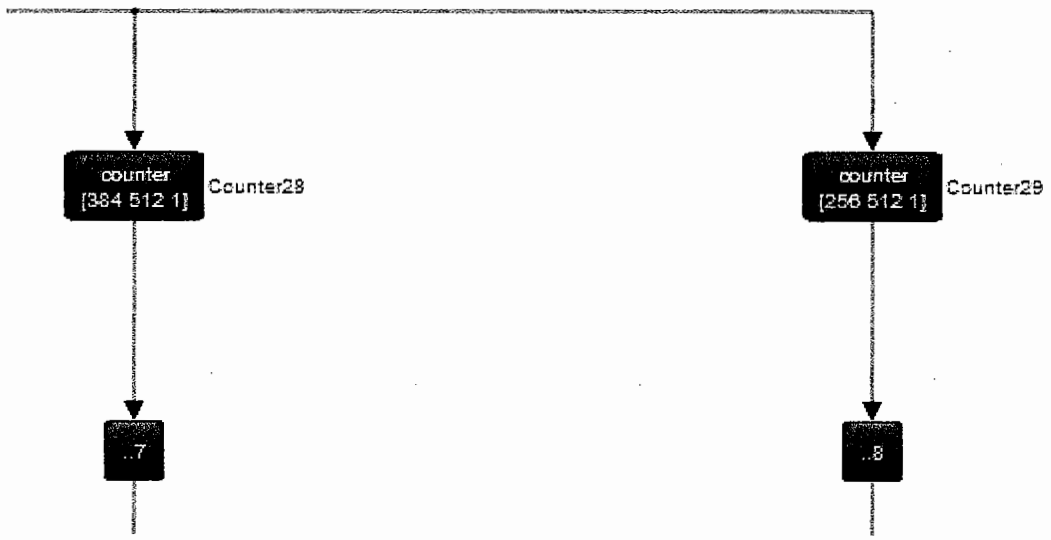
Figura 4. 34 Bloques en última etapa



A la salida del último retraso se generan las componentes reales e imaginarias de la transformada de Fourier. El último bloque de retraso lo hace con un valor de 256, previo a llegar a los 512.

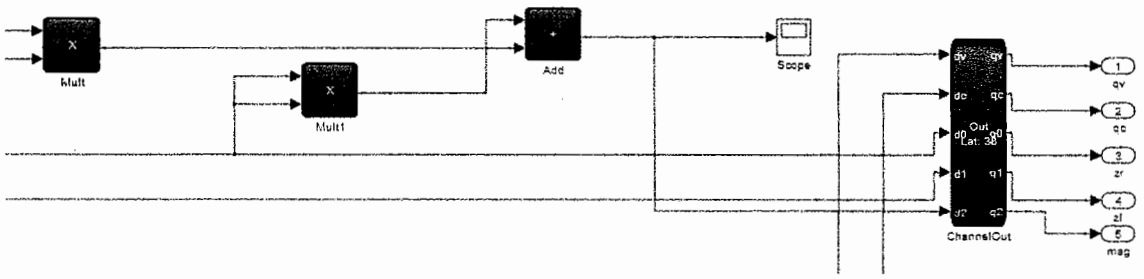
Esto se hace para indicar el tiempo en que debe procesar, el tiempo lo controlan contadores, que mantienen el registro para cada bloque de *Butterfly*.

Figura 4. 35 Contadores y sustractores de bits



Los bloques que se indican como "...7" y "...8" se llaman extractores de bits. En el caso de "...7" extraen bits hasta el séptimo. De igual manera para los otros bloques similares. Por último esta parte calcula la magnitud de las señales, elevando al cuadrado cada uno de los resultados y sumándolos.

Figura 4. 36 Bloques para calcular la magnitud



4.2.6 Simulación y resultados del modelo.

Los parámetros de solucionador son:

Tiempo de inicio: 0.0

Tiempo de fin: 1600

Tipo: Paso fijo (*Fixed step*), Discreto (sin estados continuos)

La señal de entrada será la siguiente:

Es la sumatoria de dos ondas senoidales de:

1. Frecuencia 50 rad/sec y amplitud de 0.0045
2. Frecuencia de 200 rad/sec y amplitud de 0.0045

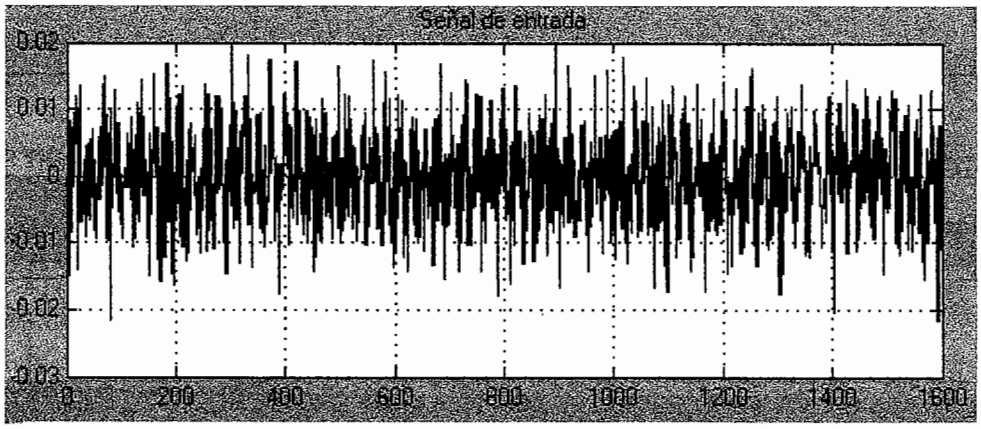
Más la suma de un generador de números aleatorios:

- Valor medio 0.00023; varianza: 0.0000234

Esta señal está basada en una señal aleatoria de distribución normal (Gaussiana).

La salida en total de la señal de prueba es:

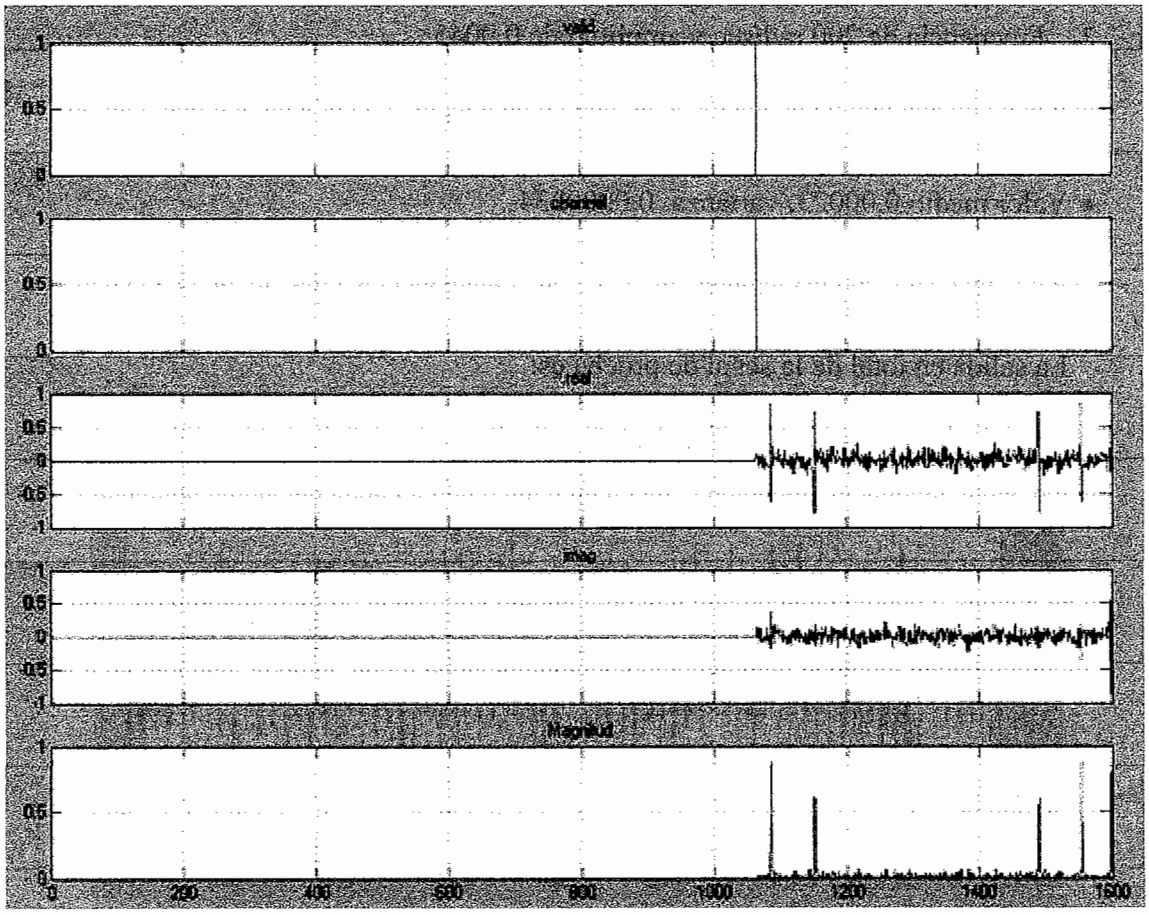
Figura 4. 37 Señal con ruido agregado



Se puede apreciar la presencia de ruido debido a la generación de números aleatorios. A continuación la figura 4.58 muestra los resultados obtenidos a la *FFT* de la señal anterior:

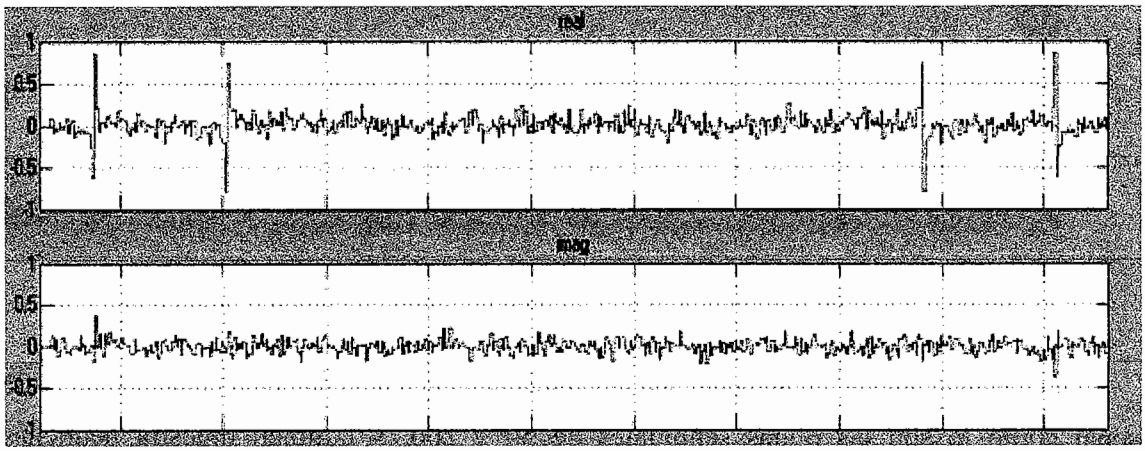
La primera señal con nombre "*valid*" indica desde qué punto se empieza a tener resultados después de los retrasos debidos al procesamiento del algoritmo. El valor en el cual empieza es 1065. Esto también significa que en 1065 muestras se obtienen resultados, debido a que el tiempo de muestreo se configuró a 1. Con el propósito de que la simulación pueda ser mejor percibida. La siguiente señal "*Channel*", indica el valor del canal, por ejemplo actualmente el valor indica 1.

Figura 4. 38 Respuesta FFT de 512 puntos



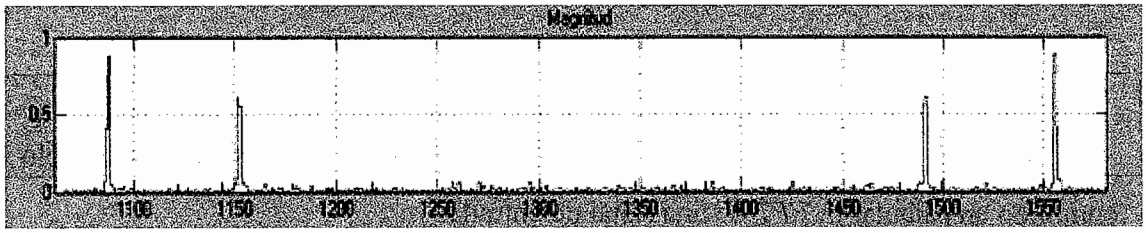
Las dos señales posteriores corresponden a la respuesta de la *FFT* en su forma real e imaginaria, un acercamiento de la señal se verá en las siguientes gráficas:

Figura 4. 39 Acercamiento de señales real e imaginaria



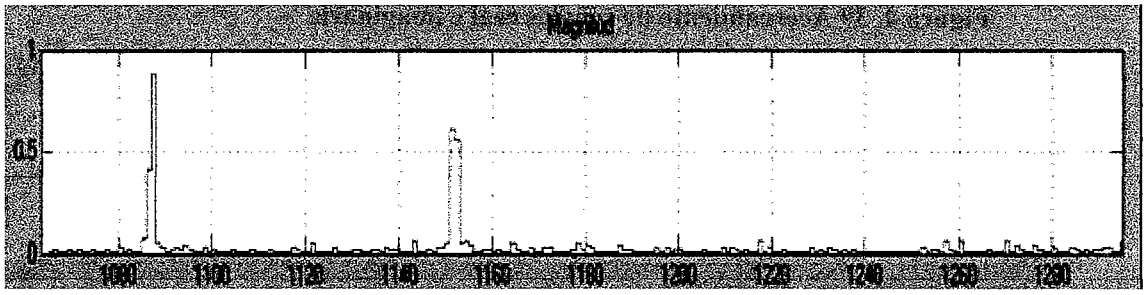
Este tipo de señales son muy difíciles de interpretar en esta manera, así que es mejor observar su magnitud. La siguiente figura muestra la respuesta magnitud *FFT* a la señal de entrada.

Figura 4. 40 Magnitud espectro completo



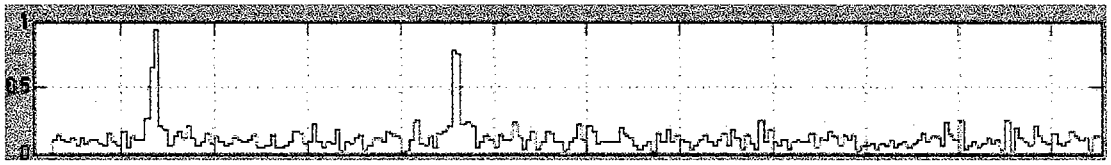
Y para visualizar solo de 0 hasta fs/2:

Figura 4. 41 Magnitud espectro desde 0 a Fs/2



La gráfica muestra bien las dos componentes que aportan mayor potencia en la señal como lo son las ondas senoidales de 50 y 200, y también muestra pequeñas componentes que son debidas a los números aleatorios. Ahora para confirmar la respuesta se comparará con la gráfica obtenida por bloque propio de Simulink:

Figura 4. 42 Comparación de la magnitud



Se puede observar la similitud entre ambas respuestas. Por lo tanto el modelo ha cumplido en calcular la *FFT* de una señal de entrada.

4.2.6.1 Generación de reportes y creación de Hardware.

Tabla XIII Reporte del modelo *FFT*

Flow Status	*Successful* - Mon Jun 21 12:53:46 2010
Quartus II Version 9.1 Build 350 03/24/2010 SP 2 SJ *Web Edition*	
Revision Name	FFT_512
Top-level Entity Name	FFT8K_FFT_512
Family	Stratix III
Met timing requirements	N/A
Logic utilization	8 %
Combinational ALUTs	1,843 / 38,000 (5 %)
Memory ALUTs	324 / 19,000 (2 %)
Dedicated logic registers	2,979 / 38,000 (8 %)
Total registers	2979
Total pins	190 / 296 (64 %)
Total block memory bits	77,346 / 1,880,064 (4 %)
DSP block 18-bit elements	36 / 216 (17 %)
Total PLLs	0 / 4 (0 %)

Tabla XIV. Cantidades de bloques *DSP* usados

Estadística	Numero
DSP Block 18-bit Elements	36
Signed Multipliers	18
Two-Multipliers Adders (18-bit)	14
Mixed Sign Multipliers	12
Simple Multipliers (36-bit)	2

Tabla XV. Estadísticas de registros generales

Estadística	Valor
Total registers	2979
Number of registers using Synchronous Clear	3
Number of registers using Synchronous Load	945
Number of registers using Asynchronous Clear	2525
Number of registers using Asynchronous Load	0
Number of registers using Clock Enable	88
Number of registers using Preset	0

Tabla XVI. Configuraciones y condiciones de operación

Configuración	Valor
Nominal Core Voltage	1.10 V
Low Junction Temperature	0 °C
High Junction Temperature	85 °C

Tabla XVII. Retraso estimado agregado durante el proceso:

Fuente/Destino	Retraso agregado en ns
Clk/clk	0.03229693

Tabla XVIII Relojes

Nombre de reloj	Tipo	Periodo	Frecuencia	Subida	Caida
bus_clk	Base	20	50.0 MHz	0	10
Clk	Base	5	200.0 MHz	0	2.5

Tabla XIX Tiempos para proporcionar la salida

Puerto de datos	Reloj	Subida	Caida	Borde del reloj	Reloj de referencia
sqr[*]	clk	9.118	8.841	*Rise*	clk
zc[*]	clk	9.16	8.871	*Rise*	clk
zi[*]	clk	9.375	9.137	*Rise*	clk
zr[*]	clk	9.23	8.962	*Rise*	clk
zv[*]	clk	9.209	8.895	*Rise*	clk

Tabla XX Uso de *RAM*

	Puerto A		Puerto B		
Nombre	Profundidad	Ancho	Profundidad	Ancho	Tamaño
DualMem1_dmem|altsyncram_v424	1024	16	1024	16	16384
DualMem_dmem|altsyncram_e324	1024	16	1024	16	16384
SampleDelay_replace_mem_dmem|altsyncram_cgq3	513	8	513	8	4104

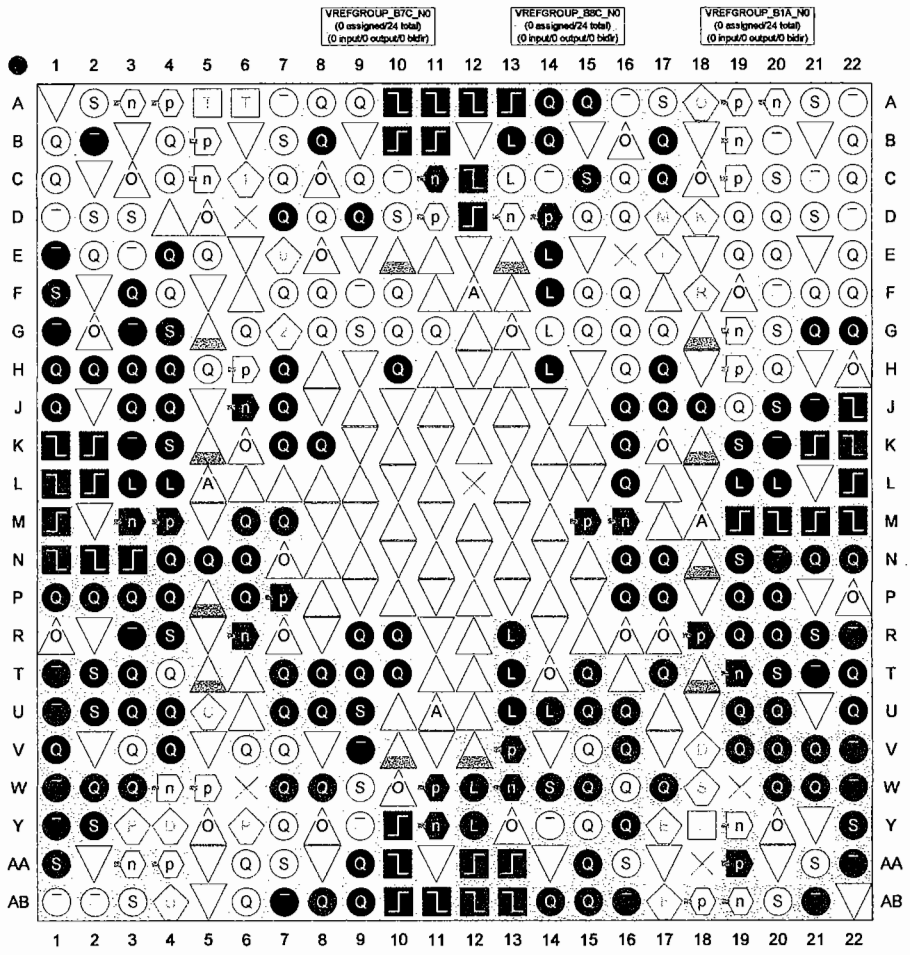
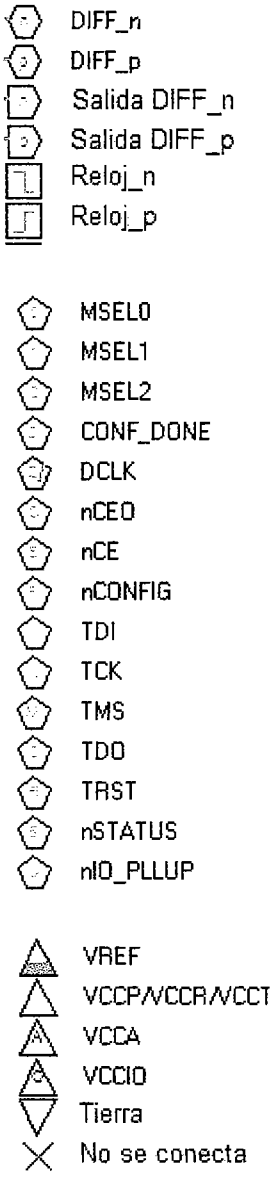


Figura 4. 43 Distribución de pines en el DSP

Asignación de pines:

Las unidades resaltadas han sido asignadas automáticamente durante el proceso de compilación, son los puertos de entrada o salida del modelo. Existen diferentes tipos de puertos estos son según su gráfico:

Tabla XXI Símbolos de los pines en el *DSP*



Se puede visualizar también que existen diferentes grupos que tienen diferentes colores de fondo, esta es la forma de organizar por bloques o tipo de señales.

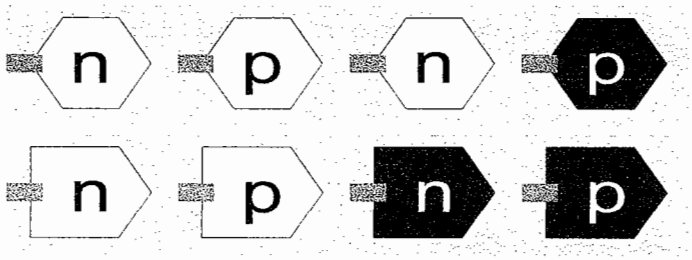
A continuación se muestra donde están ubicados los pines que se utilizaran en el dispositivo *DSP*.

La vista es desde abajo del paquete.

Figura 4. 44 Distribución de pines en el chip

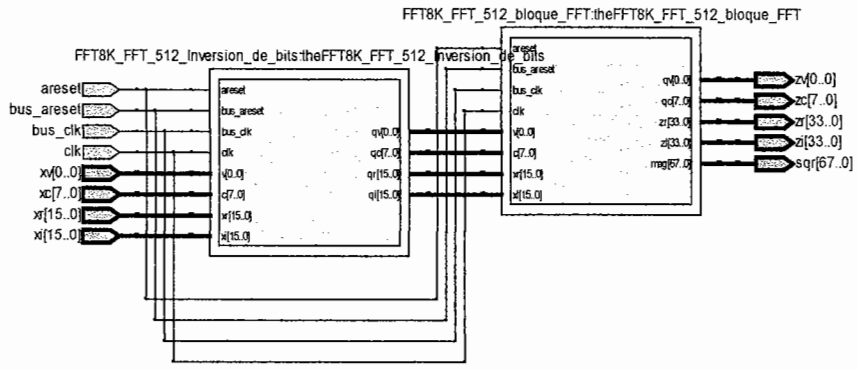
Un acercamiento de las patas del chip y se verán las asignaciones a cada uno.

Figura 4. 45 Detalle de pines del chip



Esta figura muestra el diagrama de los registros utilizados por el *Software* de Quartus II, después de la compilación, así es como interpreta el modelo en Simulink. Cada uno de los cuadros en su interior está compuesto de diferentes registros y compuertas lógicas.

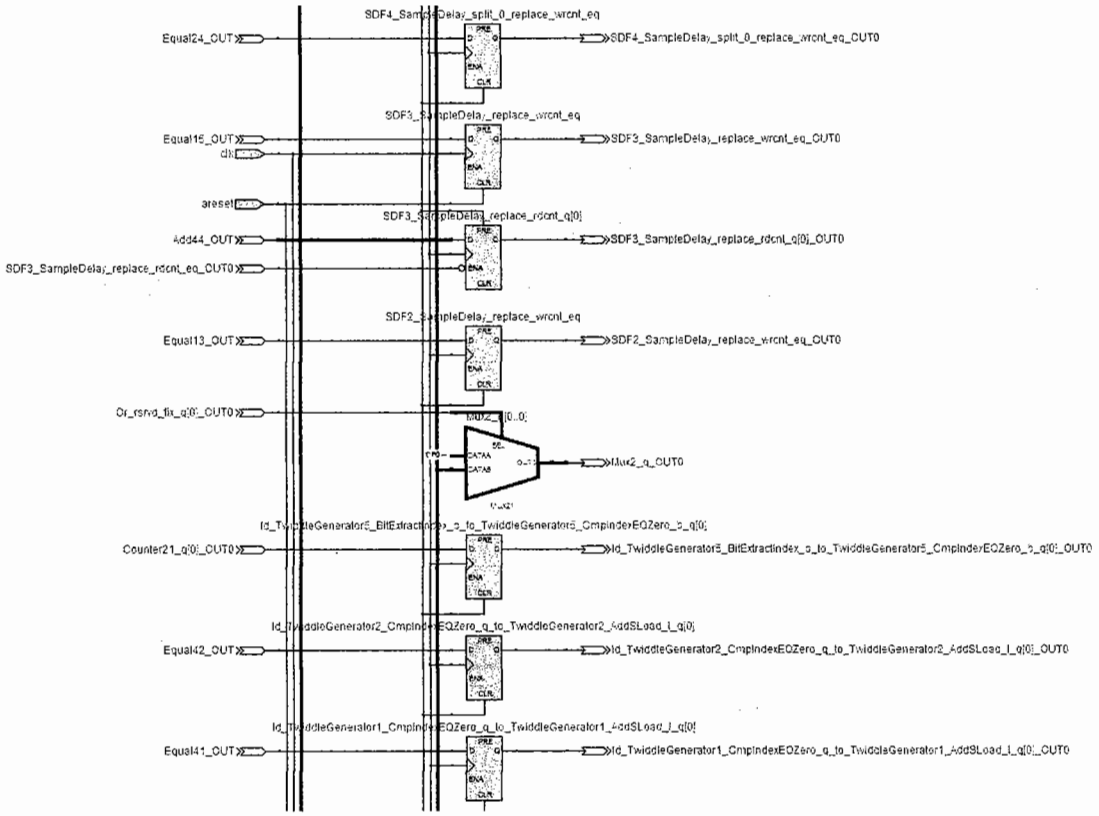
Figura 4. 46 Diagramas de registros del DSP



En los puertos de entrada se indica el tamaño de palabra de las señales. Así mismo en los puertos de salida. El siguiente es un ejemplo del uso de los registros dentro de los bloques:

En la gráfica se ven algunos bloques de retraso de muestras, también los generadores de los factores multiplicativos llamados "*Twiddle*", así también un multiplicador, varias señales de entrada provenientes de otros bloques similares, y las señales de salida. Cada una de los bloques indica la señal procedente y su destino. De esta manera el programa puede organizarse, realizar las conexiones y asignar los puertos que se utilizarán.

Figura 4. 47 Registros del *DSP*



Algunos códigos que fueron creados durante el proceso de compilación son:

El siguiente es el código generado para un solo multiplicador complejo:

```
SUBDESIGN mult_add_j39d
(
  aclr0    :        input;
  clock0   :        input;
  dataa[11..0]      :        input;
  datab[31..0]      :        input;
  result[22..0]     :        output;
)
VARIABLE
  mac_mult1 : stratixiii_mac_mult
          WITH (
                  dataa_clear = "0",
                  dataa_clock = "0",
                  dataa_width = 7,
                  datab_clear = "0",
                  datab_clock = "0",
                  datab_width = 16
          );
  mac_mult2 : stratixiii_mac_mult
          WITH (
                  dataa_clear = "0",
                  dataa_clock = "0",
                  dataa_width = 7,
                  datab_clear = "0",
                  datab_clock = "0",
                  datab_width = 16
          );
  mac_out3 : stratixiii_mac_out
```

```
        WITH (
                dataa_width = 23,
                datab_width = 23,
                dataout_width = 72,
                first_adder0_clear = "0",
                first_adder0_clock = "0",
                first_adder0_mode = "add",
                operation_mode = "one_level_adder",
                output_clear = "0",
                output_clock = "0"
        );
  aclr1    : NODE;
  aclr2    : NODE;
  aclr3    : NODE;
  clock1   : NODE;
  clock2   : NODE;
  clock3   : NODE;
  dataa_bus[13..0] : WIRE;
  datab_bus[31..0] : WIRE;
  ena0     : NODE;
  ena1     : NODE;
  ena2     : NODE;
  ena3     : NODE;

BEGIN
  mac_mult1.aclr[] = ( aclr3, aclr2, aclr1, aclr0);
  mac_mult1.clk[] = ( clock3, clock2, clock1, clock0);
  mac_mult1.dataa[] = ( dataa_bus[6..0]);
```

```
mac_mult1.datab[] = ( datab_bus[15..0]);
mac_mult1.ena[] = ( ena3, ena2, ena1, ena0);
mac_mult1.signa = B"1";
mac_mult1.signb = B"1";
mac_mult2.aclr[] = ( aclr3, aclr2, aclr1, aclr0);
mac_mult2.clk[] = ( clock3, clock2, clock1, clock0);
mac_mult2.dataa[] = ( dataa_bus[13..7]);
mac_mult2.datab[] = ( datab_bus[31..16]);
mac_mult2.ena[] = ( ena3, ena2, ena1, ena0);
mac_mult2.signa = B"1";
mac_mult2.signb = B"1";
mac_out3.aclr[] = ( aclr3, aclr2, aclr1, aclr0);
mac_out3.clk[] = ( clock3, clock2, clock1, clock0);
mac_out3.dataa[] = ( mac_mult1.dataout[22..0]);
mac_out3.datab[] = ( mac_mult2.dataout[22..0]);
mac_out3.ena[] = ( ena3, ena2, ena1, ena0);
mac_out3.signa = B"1";
mac_out3.signb = B"1";
aclr1 = GND;
aclr2 = GND;
aclr3 = GND;
clock1 = VCC;
clock2 = VCC;
clock3 = VCC;
dataa_bus[] = ( dataa[11..11], dataa[11..5], dataa[5..0]);
datab_bus[] = ( datab[31..0]);
ena0 = VCC;
ena1 = VCC;
```

```
  ena2 = VCC;
  ena3 = VCC;
  result[22..0] = mac_out3.dataout[22..0];
END;
--VALID FILE
```

4.2.6.1.1 Código *VHDL*

En el programa de modelsim se puede leer el código para este modelo, el cual fue creado después de compilar los diagramas que se hicieron en Matlab:

```
-- VHDL created from FFT8K_FFT_512
-- VHDL created on Mon Jun 28 17:12:14 2010
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_STD.all;
use IEEE.MATH_REAL.all;
use std.TextIO.all;
USE work.FFT8K_safe_path.all;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;
LIBRARY lpm;
USE lpm.lpm_components.all;
```

```
-- Text written from .\hw_model.cpp:938
entity FFT8K_FFT_512 is
  port (
    sqr : out std_logic_vector(67 downto 0);
    xc : in std_logic_vector(7 downto 0);
    xi : in std_logic_vector(15 downto 0);
    xr : in std_logic_vector(15 downto 0);
    xv : in std_logic_vector(0 downto 0);
    zc : out std_logic_vector(7 downto 0);
    zi : out std_logic_vector(33 downto 0);
    zr : out std_logic_vector(33 downto 0);
    zv : out std_logic_vector(0 downto 0);
    clk : in std_logic;
    areset : in std_logic;
    bus_clk : in std_logic;
    bus_areset : in std_logic
    );
end;
architecture normal of FFT8K_FFT_512 is
  attribute altera_attribute : string;
```

```
attribute altera_attribute of normal : architecture is "-name NOT_GATE_PUSH_BACK
OFF; -name AUTO_SHIFT_REGISTER_RECOGNITION OFF; -name MESSAGE_DISABLE 10036; -
name MESSAGE_DISABLE 10037; -name MESSAGE_DISABLE 14130; -name MESSAGE_DISABLE
14320";

real=0

--  FFT8K_FFT_512_Inversion_de_bits_qc  =>  FFT8K_FFT_512_Inversion_de_bits_qc,
width=8, real=0

--  FFT8K_FFT_512_Inversion_de_bits_qr  =>  FFT8K_FFT_512_Inversion_de_bits_qr,
width=16, real=0

--  FFT8K_FFT_512_Inversion_de_bits_qi  =>  FFT8K_FFT_512_Inversion_de_bits_qi,
width=16, real=0

  signal FFT8K_FFT_512_Inversion_de_bits_qv : std_logic_vector (0 downto 0);

  signal FFT8K_FFT_512_Inversion_de_bits_qc : std_logic_vector (7 downto 0);

  signal FFT8K_FFT_512_Inversion_de_bits_qr : std_logic_vector (15 downto 0);

  signal FFT8K_FFT_512_Inversion_de_bits_qi : std_logic_vector (15 downto 0);

  component FFT8K_FFT_512_Inversion_de_bits is

    port (

    v : in std_logic_vector(0 downto 0);

    c : in std_logic_vector(7 downto 0);

    xr : in std_logic_vector(15 downto 0);

    xi : in std_logic_vector(15 downto 0);

    qv : out std_logic_vector(0 downto 0);
```

```
    qc : out std_logic_vector(7 downto 0);
    qr : out std_logic_vector(15 downto 0);
    qi : out std_logic_vector(15 downto 0);
    clk : in std_logic;
    areset : in std_logic;
    bus_clk : in std_logic;
    bus_areset : in std_logic
    );
end component;
signal FFT8K_FFT_512_bloque_FFT_qv : std_logic_vector (0 downto 0);
signal FFT8K_FFT_512_bloque_FFT_qc : std_logic_vector (7 downto 0);
signal FFT8K_FFT_512_bloque_FFT_zr : std_logic_vector (33 downto 0);
signal FFT8K_FFT_512_bloque_FFT_zl : std_logic_vector (33 downto 0);
signal FFT8K_FFT_512_bloque_FFT_mag : std_logic_vector (67 downto 0);
component FFT8K_FFT_512_bloque_FFT is
    port (
    v : in std_logic_vector(0 downto 0);
    c : in std_logic_vector(7 downto 0);
    xr : in std_logic_vector(15 downto 0);
    xl : in std_logic_vector(15 downto 0);
```

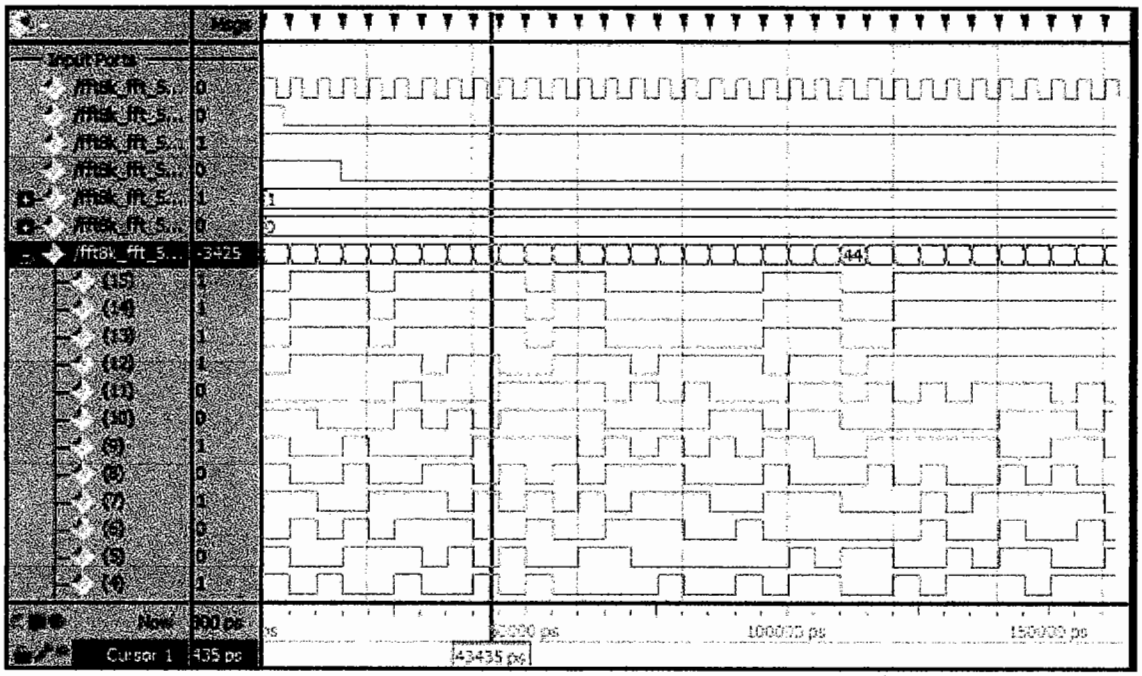
```
    qv : out std_logic_vector(0 downto 0);
    qc : out std_logic_vector(7 downto 0);
    zr : out std_logic_vector(33 downto 0);
    zl : out std_logic_vector(33 downto 0);
    mag : out std_logic_vector(67 downto 0);
    clk : in std_logic;
    areset : in std_logic;
    bus_clk : in std_logic;
    bus_areset : in std_logic
    );
  end component;
begin
  --GND(CONSTANT,1)
  --VCC(CONSTANT,2)
  --FFT8K_FFT_512_Inversion_de_bits(BLACKBOX,3)
  theFFT8K_FFT_512_Inversion_de_bits : FFT8K_FFT_512_Inversion_de_bits port map (
    v => xv,
    c => xc,
    xr => xr,
    xi => xi,
```

```
    qv => FFT8K_FFT_512_Inversion_de_bits_qv,
    qc => FFT8K_FFT_512_Inversion_de_bits_qc,
    qr => FFT8K_FFT_512_Inversion_de_bits_qr,
    qi => FFT8K_FFT_512_Inversion_de_bits_qi,
    clk => clk,
    areset => areset,
    bus_clk => bus_clk,
    bus_areset => bus_areset       );
--FFT8K_FFT_512_bloque_FFT(BLACKBOX,4)
 theFFT8K_FFT_512_bloque_FFT : FFT8K_FFT_512_bloque_FFT port map (
    v => FFT8K_FFT_512_Inversion_de_bits_qv,
    c => FFT8K_FFT_512_Inversion_de_bits_qc,
    xr => FFT8K_FFT_512_Inversion_de_bits_qr,
    xl => FFT8K_FFT_512_Inversion_de_bits_qi,
    qv => FFT8K_FFT_512_bloque_FFT_qv,
    qc => FFT8K_FFT_512_bloque_FFT_qc,
    zr => FFT8K_FFT_512_bloque_FFT_zr,
    zl => FFT8K_FFT_512_bloque_FFT_zl,
    mag => FFT8K_FFT_512_bloque_FFT_mag,
    clk => clk,
```

```
    areset => areset,
    bus_clk => bus_clk,
    bus_areset => bus_areset      );
--sqr_auto(GPOUT,5)
 sqr <= FFT8K_FFT_512_bloque_FFT_mag;
--xc_auto(GPIN,6)
--xi_auto(GPIN,7)
--xr_auto(GPIN,8)
--xv_auto(GPIN,9)
--zc_auto(GPOUT,10)
 zc <= FFT8K_FFT_512_bloque_FFT_qc;
--zi_auto(GPOUT,11)
 zi <= FFT8K_FFT_512_bloque_FFT_zl;
--zr_auto(GPOUT,12)
 zr <= FFT8K_FFT_512_bloque_FFT_zr;
--zv_auto(GPOUT,13)
 zv <= FFT8K_FFT_512_bloque_FFT_qv;
end normal;
```

La forma de onda producida por la señal de entrada en función de cada bit de la palabra se puede apreciar en la siguiente figura:

Figura 4. 48 Procesos dentro del *DSP*

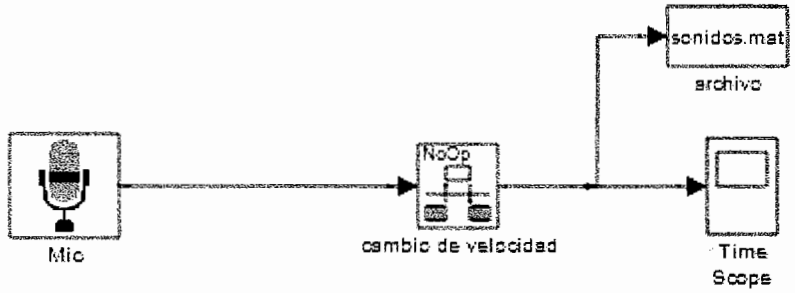


4.3 Adquisición de datos:

Como parte de la simulación para agregar otro tipo de datos, se diseñó otro modelo que obtendrá datos con el uso de un micrófono. En este modelo de Simulink los datos obtenidos serán guardados como una matriz en el directorio de Matlab, posteriormente se cargan a el espacio de trabajo con un archivo .m, este acomoda la señal para poder ser introducida a el modelo del cálculo de *FFT*.

La siguiente figura es del modelo de adquisición.

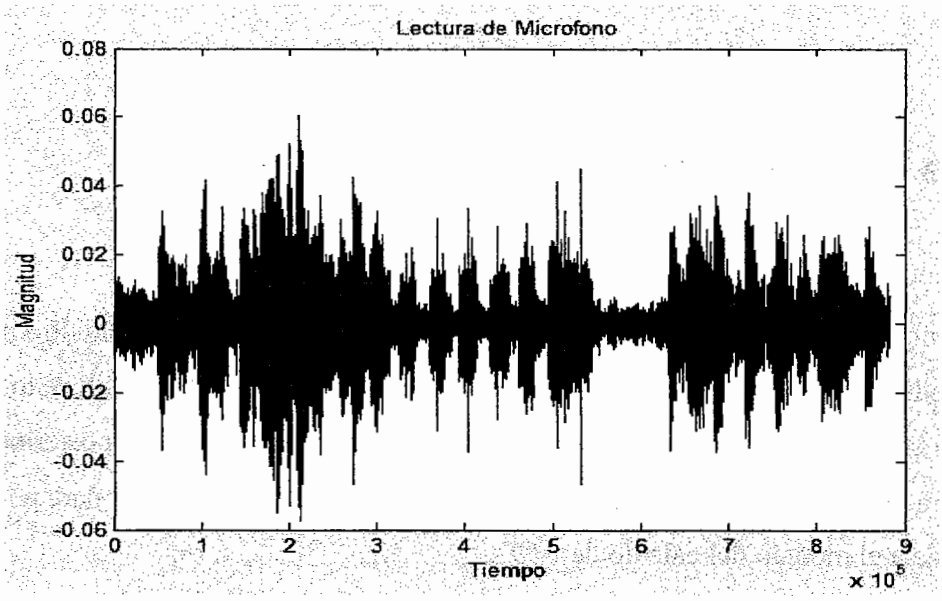
Figura 4. 49 Diagrama adquisición de datos



Y la función que carga y arregla la matriz es:

```
%% cargar
load sonidos
%% vector
datos=sunds(2,:);
tiempo=sunds(1,:);
var=[tiempo', datos'];
```

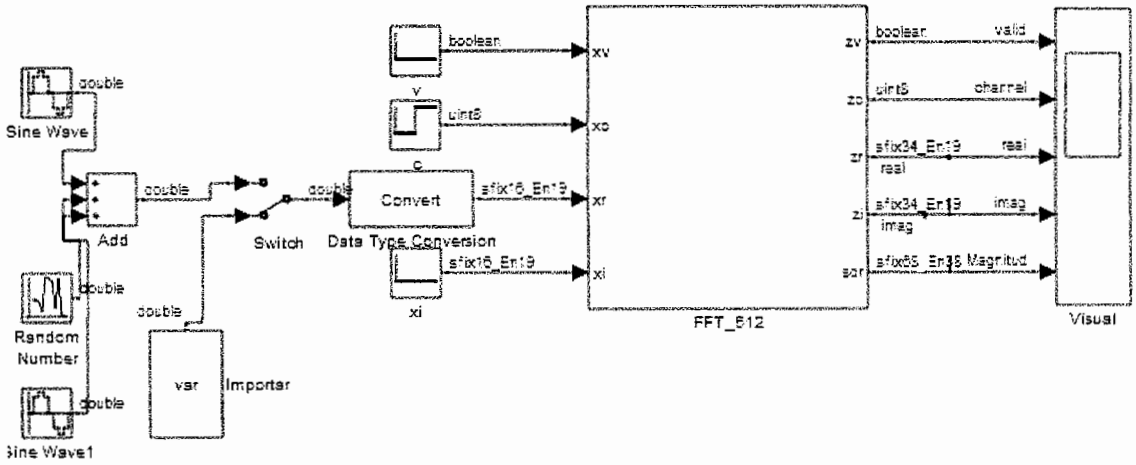
Figura 4. 50 Señal del micrófono



Se grabó sonido del micrófono que se muestra en la figura anterior.

Ahora el modelo se ha modificado como se muestra en la siguiente figura:

Figura 4. 51 Modelo completo incluyendo la adquisición de datos

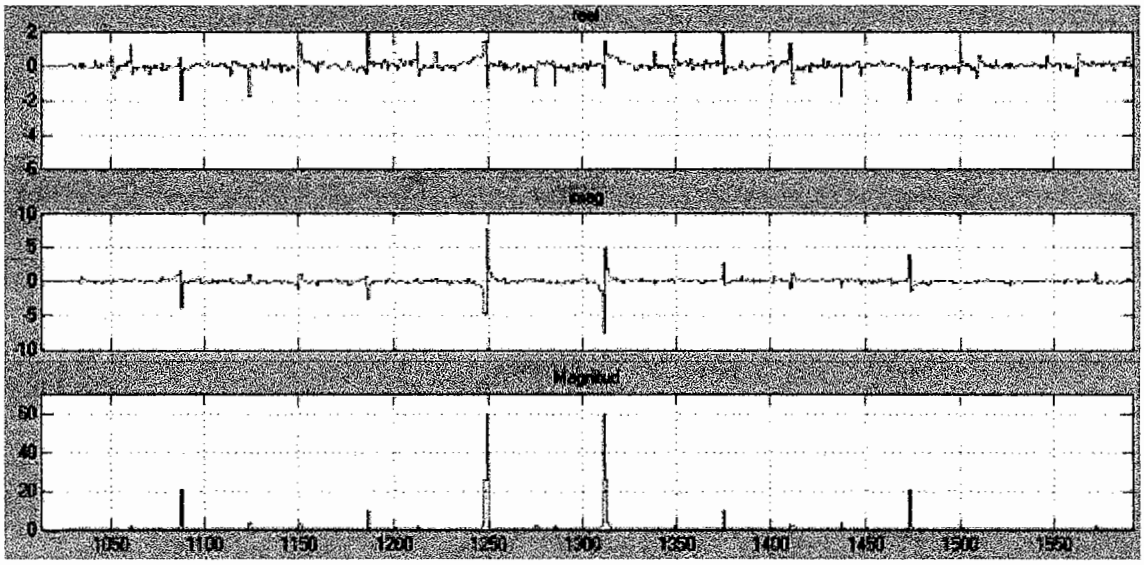


Se agregó el bloque "importar", cuya función es el de traer al modelo los datos de una variable que se encuentra en el espacio de trabajo de Matlab, esta variable se llama "Var".

Al correr el modelo se obtienen las siguientes gráficas:

Gráficas obtenidas

Figura 4. 52 Resultados del FFT aplicados a la entrada del micrófono



Se puede observar las diferentes componentes de frecuencia de la secuencia de sonido grabada.

5. APLICACIONES DE LOS MODELOS

A modo de ilustrar la utilidad de los modelos presentados en el capítulo anterior, se dedicará un espacio para indicar algunos ejemplos en los que se pueden aplicar tanto el modelo de la división, como el de la transformada rápida de Fourier. Así mismo también se incluirán comparaciones entre los modelos creados contra los modelos hechos por otras fuentes.

5.1 Aplicación del modelo de división

En el procesamiento digital de señales se requiere tomar una muestra o dato y llevarla a un proceso en el cual relacionará la variable de entrada con el resultado en la salida.

Durante este proceso varios cálculos son necesarios, en este caso se tratará con la operación de la división. Una aplicación del modulo de división que se desarrolló, es el de obtener el valor de la velocidad angular de un objeto.

Ésta es una medida de la velocidad de rotación. Se la define como el ángulo girado por unidad de tiempo y se designa mediante la letra griega ω. La importancia de la

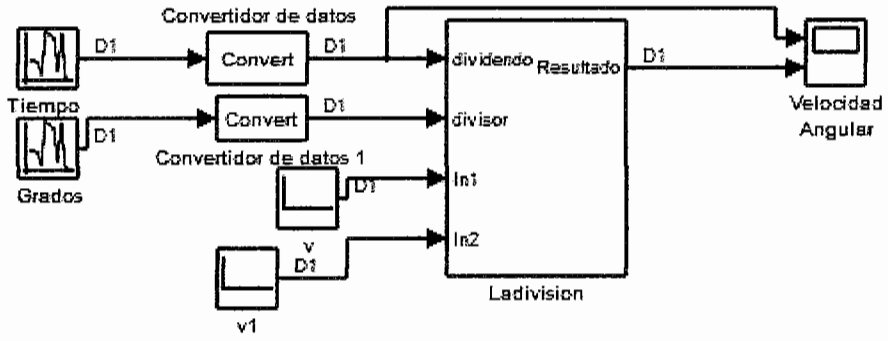
velocidad angular es que caracteriza al movimiento de rotación del sólido rígido en torno a un eje fijo.

La manera de calcular su valor instantáneo es:

$$\omega = \frac{d\theta}{dt}$$

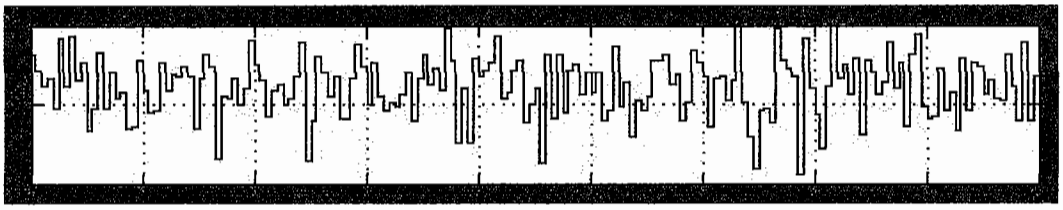
Siendo los valores de entrada el cambio de θ y el tiempo.

Figura 5. 1 Cálculo de velocidad angular



Los valores son los siguientes:

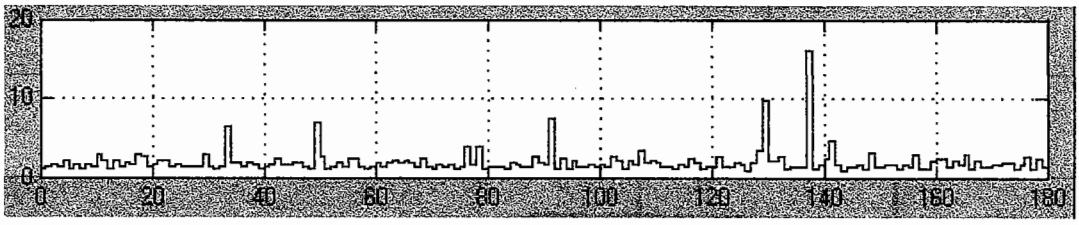
Figura 5. 2 Valores de grados por tiempo



La gráfica muestra el tiempo que tomó mover un grado del objeto.

La velocidad angular sería la siguiente:

Figura 5. 3 Resultado de la velocidad angular



5.1.1 Comparaciones

El algoritmo que se utilizó en este modelo presenta ventajas en comparación a otros algoritmos. Un algoritmo que se trabaja comúnmente es el siguiente:

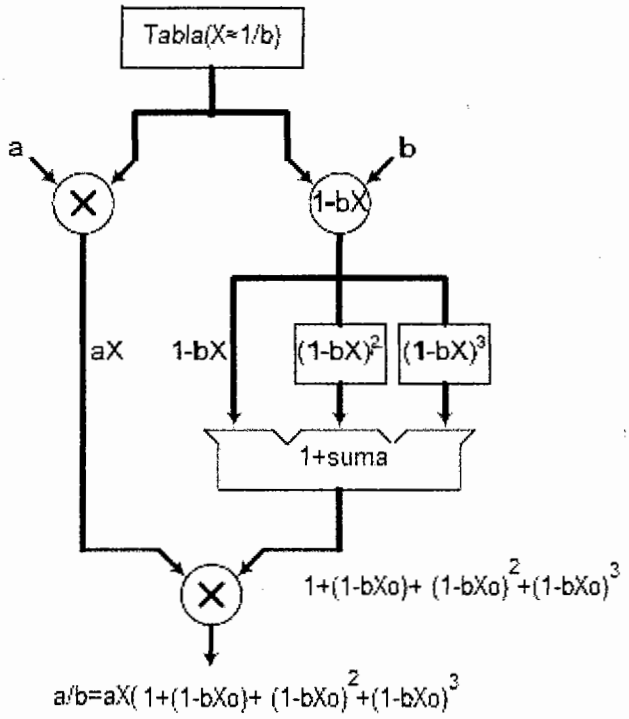
Se logra una aproximación tal que

$$q = \frac{a}{b} \approx a\,Xo\,\{1 + (1 - bXo) + (1 - bXo)^2 + (1 - bXo)^3\}$$

Donde $Xo \approx 1/b$

Con esto se logra lo siguiente:

Figura 5. 4 Otro algoritmo para calcular la división



La base de este algoritmo es una tabla que se debe crear antes de ser procesada por el algoritmo. El valor que la tabla tendrá de salida es un valor aproximado al inverso del número "b", posteriormente será multiplicado por (1-bXo), y seguidamente se calcula el valor del cuadrado y cubo, esto pude ser hasta un número determinado. Al concluir con este cálculo se suma todos los valores, para luego ser multiplicados por el factor "a Xo".

La característica principal en este algoritmo es la tabla que incluye los valores de 1/b. En un principio éste sería fácil de crear por medio de un generador cíclico o introducir los valores en la memoria. Al utilizar esta tabla es necesario tener una memoria a la que el *DSP* tenga acceso y busque los valores. Lo que hay que agregar al sistema es hardware como software, que indexe y busque los datos, este software tendría que tomar la decisión de cual número se aproxima de mejor manera.

A comparación con el algoritmo presentado en el modelo de Simulink no se necesita de ningún tipo de dispositivo de almacenamiento, o sea memorias internas o externas. Haciendo el cálculo aun más rápido debido a que es un proceso que aprovecha los resultados anteriores para calcular el siguiente.

Un problema que presenta al utilizar las tablas es que si el valor de Xo es igual a 1/b, los factores (1-bXo) serán cero.

$$para\ Xo = \frac{1}{b}$$

$$(1 - b\ Xo) = \left(1 - b * \frac{1}{b}\right) = 0$$

Esto puede resultar aún si el valor es muy cercano a 1/b y el *DSP* lo termine por aproximar al valor real. Las similitudes entre los dos algoritmos, es que se basan en el uso de los polinomios de Taylor; y que primero se calcula el inverso del número "b" y posteriormente se multiplica con el valor de "a".

El algoritmo del modelo en Simulink es muy versátil debido a que tiene una sección de escalonamiento, es decir, si el modelo se espera que solamente trabaje en un rango específico de números, este se puede configurar, y no desperdiciar recursos en valores que estén fuera de este rango; como sucedería en caso de usar una tabla de datos.

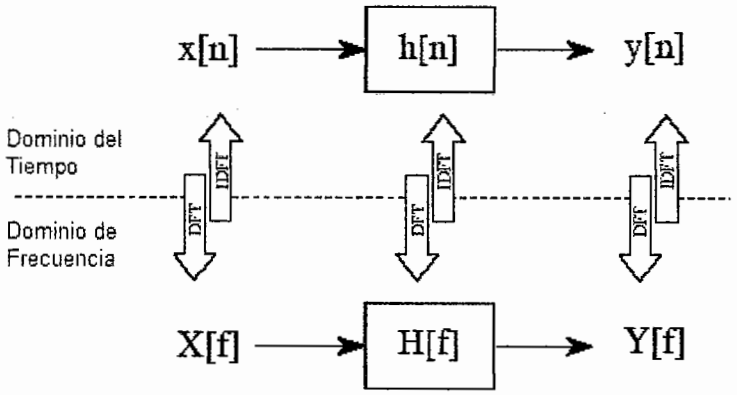
5.2 Aplicación del modelo de *FFT*

La transformada rápida de Fourier es una de las más importantes herramientas en el procesamiento digital de señales. Entre sus usos está el cálculo del espectro de frecuencia de una señal. Éste es una forma directa para examinar la información codificada en la frecuencia, fase y amplitud de la señal.

Por otra parte la *DFT* puede encontrar la respuesta a la frecuencia de un sistema a partir de la respuesta al impulso del sistema. Esto permite que los sistemas sean analizados en el dominio de la frecuencia. Usando la transformada de Fourier, cada señal de entrada puede ser representada como un grupo de ondas tipo coseno, cada una con una amplitud específica y cambio de fase. De la misma manera, la *DFT* puede usarse para representar cada señal de salida en una forma similar. Esto significa que cualquier sistema lineal puede ser descrito completamente por cómo cambia la amplitud. A esta información se le conoce como la respuesta en frecuencia del sistema.

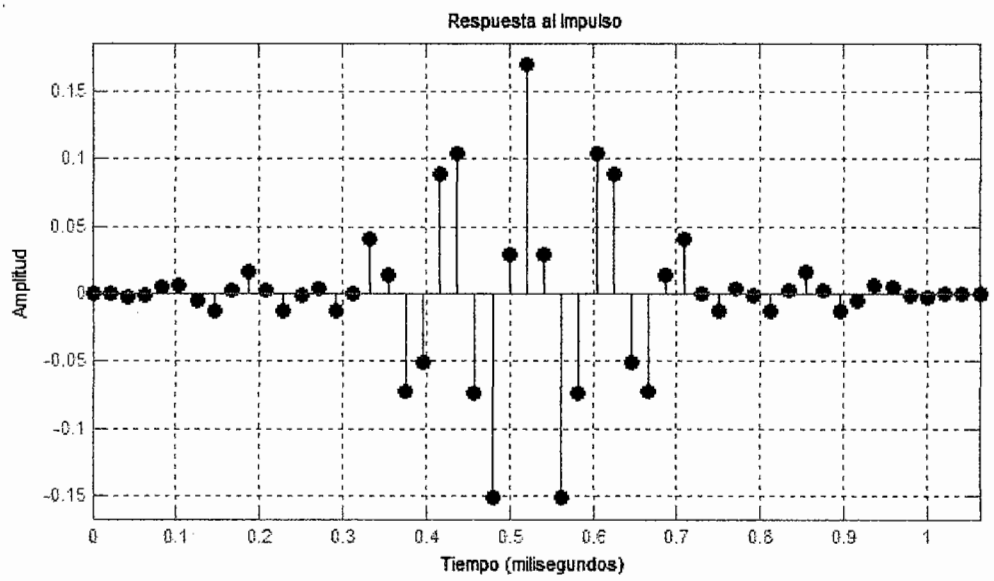
La siguiente figura muestra los resultados de calcular la respuesta en frecuencia:

Figura 5. 5 Respuestas de sistemas en diferentes dominios



Por ejemplo, si se tiene una respuesta al impulso de la siguiente manera:

Figura 5. 6 Señal de entrada respuesta al impulso

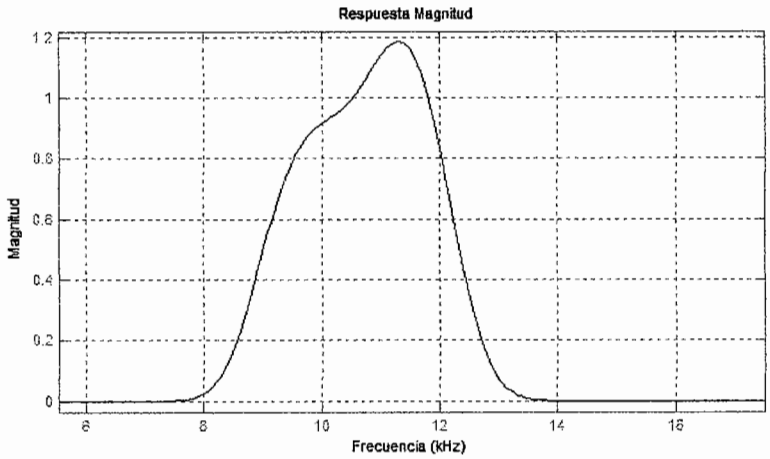


Es muy difícil identificar cual es el efecto que el sistema ejerce sobre el impulso. Pero si se analiza en la frecuencia se puede lograr a entender:

En la figura 5.7 se percibe ahora, que se trata de un filtro pasa banda, esta información ahora se conoce debido al comportamiento que presenta la figura. La amplitud a frecuencias cercanas a 11k está ligeramente aumentada, mientras que al disminuir la frecuencia más allá de 10k también disminuye la amplitud de la señal. Lo mismo sucede se aumenta los valores de frecuencia a más de 12k.

La siguiente gráfica demuestra la respuesta en la frecuencia del sistema.

Figura 5. 7 Respuesta del sistema



Ahora a modo de mostrar como el modelo ayuda a realizar esta aplicación se diseñará un sistema que tenga características de pasa banda. Primero se creará el sistema usando el *SPTOOL* de Matlab.

Tiene las siguientes frecuencias normalizadas a 1:

Frecuencia de bloqueo 1: 0.1

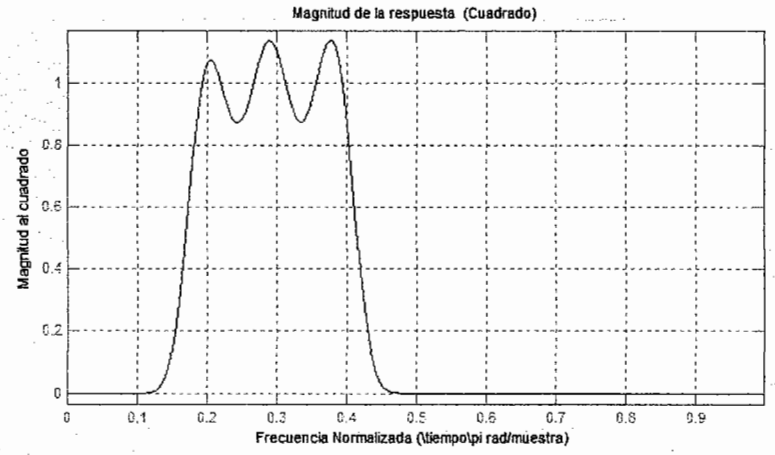
Frecuencia de paso 1: 0.2

Frecuencia de paso 2: 0.4

Frecuencia de bloqueo 2: 0.5

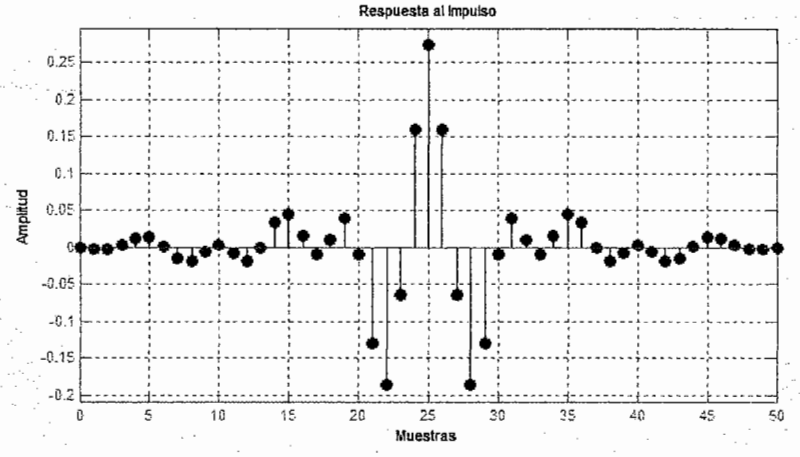
La representación gráfica de la magnitud, obtenida del código anterior se muestra a continuación:

Figura 5. 8 Magnitud de la respuesta al sistema



A continuación se calcula la respuesta al impulso.

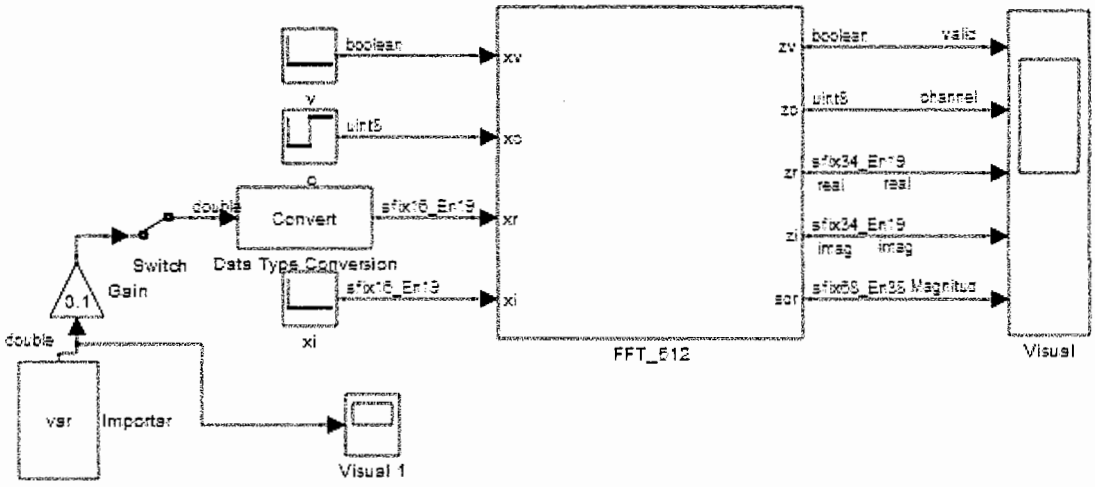
Figura 5. 9 Respuesta al impulso del sistema



Teniendo los datos de la respuesta al impulso se introducen al modelo del *FFT* para conocer y comprobar la respuesta del sistema.

Poniendo en prueba el sistema se obtiene lo siguiente:

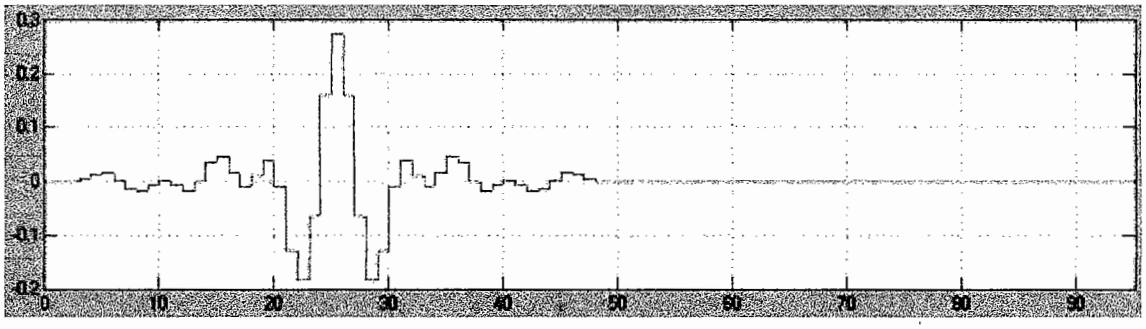
Figura 5. 10 Modelo para calcular la respuesta al sistema



Al ejecutar el modelo se obtienen los siguientes resultados:

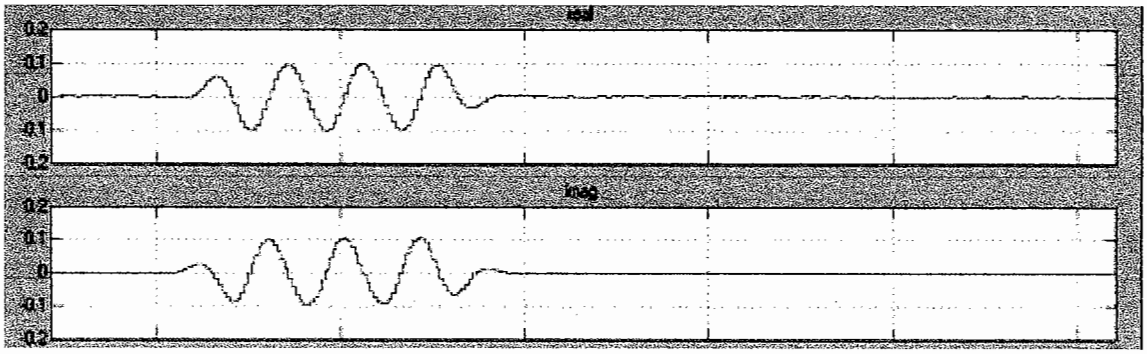
En el Visual 1 se tiene la entrada, la respuesta al impulso:

Figura 5. 11 Señal de entrada



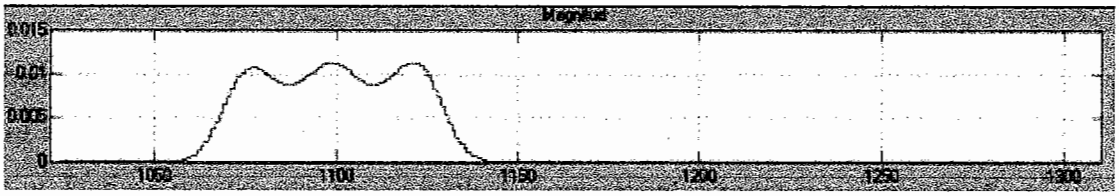
Y la salida en la parte real e imaginaria:

Figura 5. 12 Salida real e imaginaria



Y la magnitud se muestra en la salida:

Figura 5. 13 Salida de la respuesta al sistema (magnitud)



De esta manera se puede visualizar que se trata de un sistema tipo pasa banda.

5.2.1 Comparaciones

El método usado, calculando la transformada rápida de Fourier es la forma más eficiente de obtener la *DFT*. Considerando el algoritmo de evaluación directa de la expresión *DFT*, es necesario calcular cada valor de N.

Por otro lado estas son las propiedades de realizar *FFT*:

- Se descompone x(n) en sub secuencias sucesivamente más pequeñas
- Se aprovecha la simetría y periodicidad de los coeficientes W_N^{nk}
- Se utiliza longitudes de tamaño $N = 2^v$ donde v es un entero.
- Se separa en n pares e impares
- Se reduce el número de operaciones y operadores.

5.3 Análisis del tamaño de palabra

Al finalizar con el trabajo de simulación y modelaje del sistema, aun queda algo por resolver, esto es ¿Cuál es el efecto del tamaño de palabra a usar? Si se toma un tamaño aleatorio ¿se tendrán los mismos resultados?, en esta sección se hará un estudio de cuáles son las implicaciones a la hora de escoger el tamaño de palabra que se utilizará para trabajar.

Ambos modelos se han trabajado con una representación numérica de punto fijo. En este tipo de representación los bits a la izquierda del punto decimal se denominan bits de magnitud y representan valores enteros, en cambio los bits a la derecha del punto decimal representan valores fraccionales (potencias inversas de 2). Es decir que el primer bit fraccional es ½, el segundo es ¼, el tercero es 1/8, etc.

Se puede calcular la representación en punto fijo de un número dado de la siguiente manera:

$$\mp 2^{m-1} - 1/2^{f}$$

Donde m son los bits de magnitud y f son los bits fraccionales.

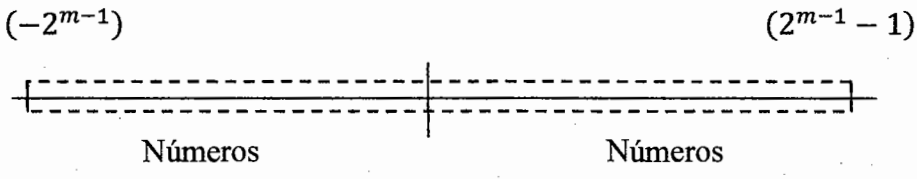
5.3.1 Rango y Precisión

El rango de un número da los límites de la representación, mientras que la precisión indica la distancia entre dos números sucesivos en la representación.

Rango

El rango se puede visualizar así:

Figura 5. 14 Gráfica del rango



Precisión

Para obtener la precisión se verá cómo Matlab establece los límites. En los modelos se utilizó números con punto binario con signo. Los límites son los siguientes:

Tabla XXII Cálculo de la precisión

Límite inferior	Límite superior	Precision
-2^{m-1-f}	$(2^{m-1}-1)2^{f}$	2^{f}

Tabla XXIII Precisión según tamaño de palabra

Tamaño :				
Palabra	Fraccional	Límite Inferior	Limite Superior	Precisión
16	19	-32768	32767	1.90735E-06
8	2	-128	127	0.25
16	3	-32768	32767	0.125
10	15	-512	511	3.05176E-05
4	10	-8	7	0.000976563

En la tabla se visualiza los diferentes rangos de valores a los que se trabaja para distintos tamaños de palabra y fraccional. El que se usó en los modelos fue un tamaño de palabra de 16 y parte fraccional de 19, como indica en la tabla esto da una precisión de 1.90735E-06. Haciendo posible cubrir adecuadamente los valores que se procesarán por el *DSP*.

CONCLUSIONES

1. Con la ayuda del *software* de Simulink se logró crear un modelo que representa el programa para la ejecución de procesamiento digital de señales. Esta programación se hace por medio de un diagrama de bloques, llamados *Blocksets*; en especial con el *Blockset* distribuido por Altera, llamado *DSP Builder*, que integra las funciones propias del *DSP*. Por medio de este procedimiento se realizaron varias ejecuciones del modelo y varias pruebas, y como se observó en los resultados, éste es un buen método para realizar la programación del *DSP*.

2. Después de realizar el modelo en Simulink se generó por medio del programa Quartus II los códigos *VDHL*, que son las líneas de instrucciones que el chip DSP interpreta y pone en ejecución. La gran ventaja del trabajo conjunto de estos dos *software* es tal, que permitió de una manera rápida y sencilla obtener los códigos *VHDL*.

3. Las señales tanto de salida como internas del DSP fueron visualizadas por medio de los bloques de Simulink y el *software* de *Model Sim*. También se logró por medio del *software* Quartus II, asignar pines a las señales de entrada, salida, control y voltaje.

4. Se crearon reportes que muestran el rendimiento del dispositivo *DSP* a la hora de estar en ejecución, dando a conocer el número de registros internos usados, cantidad de memoria *RAM* utilizada, porcentaje de utilización del *CPU* en el chip, retrasos y tiempos de ejecución, resumen de los relojes usados, configuraciones y condiciones de operación.

5. Se adquirió señales de tipo voz y música por medio de un micrófono para obtener su representación en la frecuencia, por medio del modelo que se diseñó para calcular la *FFT* de una señal.

RECOMENDACIONES

1. Obtener los dispositivos de comunicación entre el programa Quartus II y el chip *DSP*. De esta manera se puede descargar y trabajar físicamente con el *DSP* y crear el sistema, montando el chip en una tarjeta lista para su utilización.

2. Agregar más etapas al modelo de *FFT*, que incluyan un bloque *Butterfly* tipo I y II, para poder expandir el modelo a medidas más exactas, aunque ésto a cambio de una mayor cantidad de memoria y recursos.

3. Asignar los pines de forma manual, de la manera que se ajuste mejor a las necesidades del usuario.

BIBLIOGRAFÍA

1. Altera, ***DSP Builder advanced blockset user guide***, California: s.e, 2009. 124pp.

2. Karris, Steven T., ***Signals and systems with MATLAB applications***. 2da. Edición. Fremont, California: Orchard Publications, 2003. 598pp.

3. National Semiconductor, ***An introduction to the sampling theorem***, Texas: s.e. 1980, 87pp.

4. Oppenheim, Alan V. y Shafer, Ronald W., ***Discrete-time signal processing***. 2da edición, Upper Saddle River, New Jersey: Prentice Hall, 1999. 895pp.

5. Platero, Carlos. **Introducción al procesamiento digital de señales,** Madrid, España: Universidad Politécnica de Madrid. 760pp.

6. Smith, Steve. ***The Scientist and Engineer´s guide to digital signal processing***. 3ra edición. San Diego, California: s.e. 2002. 566pp.

7. Taek-Jun, Kwon y otros, ***Floating-point division and square root using a Taylor-series expansion algorithm***, California: University of Southern California. 4pp.

8. Taub, Herbert y Schilling Donnald T., ***Principles of communication systems***. 2da edición, New York, USA: McGraw-Hill Book Company. 587pp.