ESCUELA SUPERIOR POLITECNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"GENERACION DE UN CANAL PILOTO EN UN ENLACE DELANTERO DE UN SISTEMA CDMA UTILIZANDO DSP"

TESIS DE GRADO

Previa a la obtención del Título de:

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Presentada por:

Christian Servando Fuentes Allieri

Carlos Fernando Domínguez Miño

GUAYAQUIL – ECUADOR

2007

A G R A D E C I M I E N T O

Agradecemos:

Primeramente a Dios por el

término de este trabajo.

A nuestros padres

por todo su esfuerzo

y sacrificio dedicado

con el fin de que

culminemos

con éxito nuestra carrera.

D E D I C A T O R I A

NUESTROS PADRES

HERMANOS

ESPOSA E HIJA

TIOS

AMIGOS.

TRIBUNAL DE GRADUACIÓN

Ing. Holger Cevallos
SUBDECANO DE LA FIEC

Ing. Germàn Vargas
DIRECTOR DE TESIS

Ing. Rebeca Estrada
VOCAL

Ing. Cesar Yépez
VOCAL

DECLARACIÓN EXPRESA.

"La responsabilidad del contenido de esta Tesis de Grado, nos corresponde exclusivamente; y el patrimonio intelectual de la misma a la ESCUELA SUPERIOR POLITECNICA DEL LITORAL" (Reglamento de Graduación de la ESPOL)

Christian Servando Fuentes Allieri

Carlos Fernando Domínguez Miño

RESUMEN.

Nuestro proyecto esta orientado a la implementación del Canal Piloto en un Enlace Delantero (enlace entre Estación Base y Estación Móvil) de un Sistema CDMA utilizando Procesadores Digitales de Señales. Este canal nos permite identificar a la estación base cuando un móvil desea realizar un enlace para establecer una comunicación, está presente en todas y cada una de las estaciones base que conforman un sistema que provee servicios CDMA.

Nosotros en este proyecto analizaremos todas las partes constitutivas del Canal Piloto, entre las que podemos mencionar:

- La generación del Código Walsh, en nuestro caso el Walsh Wo.
- La generación de la secuencia PN_I y PN_Q.
- Diseño de Filtros Digitales.

En general nuestro estudio está orientado a la implementación de los bloques de este canal, para lo cual usaremos herramientas que la tecnología de hoy en día nos brinda, como es la utilización de Procesadores Digitales de Señales (DSP), además de utilizar para la programación algunas de las

herramientas de desarrollo más populares a nivel de Ingeniería como es MATLAB y Code Composer Studio, las cuales nos ofrecen muchas opciones al momento de realizar la programación para la inmediata implementación. Una de las tendencias del mundo contemporáneo es la de crear sistemas versátiles, flexibles ante nuevas aplicaciones, esto se logra con los procesadores digitales de señal, con solo modificar el software.

INDICE GENERAL.

RESUMEN.....VI

ÍNDICE GENERAL.....VIII

ABREVIATURAS.....XIII

ÍNDICE DE FIGURAS.....XVIII

ÍNDICE DE TABLAS.....XXII

ÍNTRODUCCIÓN.....1

CAPITULO I

1. INTRODUCCION AL SISTEMA CDMA.

1.1 Principios de las redes celulares CDMA……1

1.2 Arquitectura de una red inalámbrica celular……...3

1.3 Pérdidas y propagación en ambientes inalámbricos……9

1.4 El rol del canal Piloto en un sistema CDMA……10

1.4.1 Sincronización de la Unidad móvil……11

1.4.2 Handoff……...11

1.4.3 Localización de Posición……12

1.5 El fundamento del sistema CDMA……13

1.6 Componentes de los sistemas de comunicación……14

1.6.1 Codificación de Fuente……15

1.6.2 Codificación de Canal……..16

1.6.3 Acceso Múltiple. ……..……16

1.6.3.1 Códigos Walsh. ……17

1.6.3.2 Códigos PN. ……..19

1.6.4 Modulación. ……..……21

1.7 CDMA y comunicaciones en Espectro Ensanchado..……23

1.8 La estructura de enlace CDMA. ……26

1.8.1 El enlace Delantero ..……..……26

1.8.1.1 El canal Piloto. ……..27

1.8.1.2 El canal de Sincronismo……...29

1.8.1.3 El canal de Paginación ……30

1.8.1.4 El canal de Tráfico. ……..32

1.8.2 El enlace Inverso... ……..38

1.8.2.1 El canal de Acceso. ……39

1.8.2.2 El canal de Tráfico. ……..42

1.8.2.3 Formatos del canal de Tráfico……44

CAPITULO II

2. INTRODUCCIÓN A PROCESAMIENTO DIGITAL DE SEÑALES.

2.1 Elementos Básicos de un Sistema de Procesamiento Digital……...47

2.1.1 Acondicionamiento de la señal de entrada……...50

2.1.2 Conversión Analógica-Digital A/D……..51

2.1.3 Conversión Digital-Analógica. ……52

2.2 Procesador Digital de Señales. ……..53

2.2.1 Introducción al DSP. ……55

2.2.2 Procesador Digital de Señal TMS320C6416. ……58

2.2.3 Propiedades C64x. ……..62

2.2.4 Arquitectura TMS320C6x. ……...63

2.2.5 Buses Internos. ……66

2.2.6 Periféricos. ……67

CAPITULO III

3. SOFTWARE DE DESARROLLO.

3.1 Matlab……72

3.1.1 Simulink……..76

3.1.1.1 Librerías de Simulink……77

3.1.1.2 Configuración de parámetros iniciales……...78

3.1.1.3 Real-Time Workshop……78

3.2 Code Composer Studio. ……81

3.2.1 Compilación del código fuente en la tarjeta. ……...84

3.2.2 Carga y ejecución del programa objeto. ……..91

CAPITULO IV

4. ANÁLISIS DE LA ESTRUCTURA DEL CANAL PILOTO.

4.1 Sistema por Bloques. ……..94

4.1.1 Bloque Walsh……..……..95

4.1.1.1 Funcionamiento. ……..…….95

4.1.1.2 Función Matemática. ……....95

4.1.1.3 Diagrama Digital. ……..96

4.1.1.4 Diagrama de flujo del Bloque. ……97

4.1.2 Bloque Secuencia PN. ……98

4.1.2.1 Funcionamiento. ……...98

4.1.2.2 Función Matemática. ……99

4.1.2.3 Diagrama Digital. ……100

4.1.2.4 Diagrama de flujo del Bloque. ……..101

4.1.3 Bloque Filtrado. ……..102

4.1.3.1 Funcionamiento. ……102

4.1.3.2 Función Matemática. ……105

4.1.3.3 Diagrama Digital. ……105

4.1.3.4 Diagrama de flujo del Bloque. ……..106

4.2 Implementación del código en el DSP. ……...107

CAPITULO V

5. PRUEBAS.

5.1. Pruebas simuladas en Matlab de los bloques constituyentes……...108

5.2. Prueba de carga de programa fuente generado por Matlab al DSP ……127

5.3. Verificación de Programa en Tarjeta……141

CONCLUSIONES Y RECOMENDACIONES.

ANEXOS.

BIBLIOGRAFIA.

ABREVIATURAS.

3G	Tercera Generación.
AC	Corriente Alterna.
ADC	Convertidor Analógico Digital.
ALU	Unidad Aritmética y Lógica.
ANSI	Instituto de Normas Nacionales Estadounidense.
API	Aplicación de Interface de Programas.
bps	Bits por Segundo.
BS	Estación Base
CCS	Code Composer Studio.
CDMA	Acceso Múltiple por División de Código.
CE 0	Chip Enable Space 0.
CPLD	Dispositivo Lógico Programable Complejo.
CPU	Unidad de Procesamiento Central.
CRC	Chequeo de Redundancia Cíclica.
DAC	Convertidor Digital Analógico.

dB Decibeles.

DIP Conmutación de la línea Dual en Paquete.

DMA Acceso Directo a Memoria.

DRAM Memoria de Acceso Dinámico Aleatorio.

DSK Paquete Digital para principiante.

DSP Procesador Digital de Señales.

DSSS Secuencia Directa por Espectro Expandido.

Eb/No Relación Señal a Ruido.

EDMA Acceso Directo a Memoria Mejorado.

EEPROM Memoria de Solo Lectura Borrable Eléctricamente.

EMAC Controlador de Acceso a Medios Ethernet.

EMIF Interface de Memoria Externa.

EMIF A Interface de Memoria Externa bloque A.

ESN Número de Serie Electrónico.

fb Tasa de Bit.

fc Tasa de Chip.

FCC Comisión Federal de Comunicaciones.

FDMA Acceso Múltiple por División de Frecuencia.

FHSS Saltos de Frecuencia por Espectro Expandido.

FIR Respuesta Finita al Impulso.

GPIO Entrada/Salida de Propósito General.

GSM Sistema Global para Móviles.

GUI Interface Grafica de Usuario.

HLR Registro de Ubicación Base.

HPI Interface de puerto host.

IDE Ambiente de Desarrollo Integrado.

ISDN Red Digital de Servicios Integrados.

Kbps Kilo Bits por Segundo.

LED Diodo Emisor de Luz.

MIPS Millones de Instrucciones por Segundo.

MMACS Millón de Multiplicaciones y Acumulaciones por Segundo.

MODEM Modulador Demodulador.

MS Estación Móvil.

MSC Centro de Conmutación Móvil.

NRZ Sin Retorno a Cero.

OMC Centro de Operación y Mantenimiento.

PCB Bits de Control de Potencia.

PCG Grupo de Control de Potencia.

PCM Modulación por Código de Pulsos.

PCS Sistema de Comunicación Personal.

PLL Circuito de Fase Cerrada.

PN Ruido Pseudo Aleatorio.

PSTN Red Telefónica Conmutada Pública.

QPSK Modulación por Desplazamiento de Fase en Cuadratura.

R Ancho de Banda del Mensaje.

RF Radio Frecuencia.

ROM Memoria Solo de Lectura.

RTT Tecnología de Radio Transmisión.

SDRAM Acceso sincrónico dinámico a memoria aleatorio.

SMS Sistema de Mensajería Instantánea.

SYNC Canal de Sincronía.

TCP Coprocesador Turbo.

TDMA Acceso Múltiple por División de Tiempo.

TIA Asociación de Industrias de Telecomunicaciones.

USB Bus Serial Universal.

UTOPIA Interface Universal de pruebas para ATM.

VCP Coprocesador Viterbi.

VLIW Instrucción de ancho de palabra muy larga.

VLR Registro de Ubicación de Visitante.

Vocoders Codificador de Voz.

VoIP Voz sobre Protocolo de Internet.

W Ancho de Banda de la Señal Ensanchada.

INDICE DE FIGURAS.

Figura 1.1 Red Inalámbrica CDMA. ……..….3

Figura 1.2 Esquema de una red celular…….4

Figura 1.3 Uso de espectro FDMA vs. TDMA vs. CDMA. ……..7

Figura 1.4 Sistema de Comunicación típico……13

Figura 1.5 Modulador OQPSK. …….22

Figura 1.6 Representación de constelación QPSK. ……..23

Figura 1.7 Estructura de un Canal Piloto. ……...28

Figura 1.8 Estructura de canal de Sincronismo …….29

Figura 1.9 Estructura de canal de Paginación…….……...31

Figura 1.10 Estructura de canal de Tráfico. …….33

Figura 1.11 Estructura de modulador del Enlace Delantero…...……37

Figura 1.12 Estructura de canal de Acceso ……..39

Figura 1.13 Estructura de canal de Tráfico. …… ……42

Figura 1.14 Estructura de canal de Tráfico para tasa de datos 2 ...…….44

Figura 2.1 Diagrama de Bloques de un sistema típico……..47

Figura 2.2 Tiempo de Muestreo/Espera/Procesamiento……..…49

Figura 2.3 Ilustración de la conversión ADC……...51

Figura 2.4 Ilustración de la conversión DAC……...52

Figura 2.5 TMS320C64X diagrama de bloques del núcleo……..60

Figura 2.6 Diagrama de Bloques DSP…….…64

Figura 2.7 Ruta de Datos CPU TMS320C64X……65

Figura 2.8 Buses Internos C64x……66

Figura 3.1 Proceso de Construcción en Simulink. ……80

Figura 3.2 Proceso de Desarrollo del código en CCS. ……82

Figura 3.3 Entorno de CCS……83

Figura 3.4 Ventana de Configuración de Parámetros. ……84

Figura 3.5 Ventana de Configuración Hardware Implementación ….……86

Figura 3.6 Ventana de Configuración de Real Time Workshop……...…87

Figura 3.7 Cuadro de selección de Plataformas ……. ……..88

Figura 3.8 Ventana de Configuración de TI C6000…….. ……89

Figura 3.9 Ventana Configuración TI C6000 Generación Código …..…...90

Figura 3.10 Cuadro de dialogo Cargar Programa .. ……92

Figura 3.11 Indicador de Carga de Programa Objeto. ……93

Figura 4.1 Diagrama de bloques del Canal Piloto ……94

Figura 4.2 Diagrama digital Generador Walsh…….……..96

Figura 4.3 Diagrama de flujo bloque Walsh……97

Figura 4.4 Diagrama de bloque Secuencia PN I. ……100

Figura 4.5 Diagrama de bloque Secuencia PN Q. ……..100

Figura 4.6 Diagrama de flujo Bloque Secuencia PN……101

Figura 4.7 Respuesta de Frecuencia Filtro Banda Base. ……102

Figura 4.8 Diagrama Digital Filtro FIR……105

Figura 4.9 Diagrama de Flujo diseño de filtros……....106

Figura 5.1 Modelo Generador de Walsh……108

Figura 5.2 Grafico de Walsh 0……112

Figura 5.3 Grafico de Walsh 32……...113

Figura 5.4 Grafico de Walsh 63……...113

Figura 5.5 Modelo Generador Secuencia PN I……114

Figura 5.6 Modelo Generador Secuencia PN Q……...115

Figura 5.7 Gráfico de Secuencia PN Ramal I……...121

Figura 5.8 Gráfico de Secuencia PN Ramal Q……121

Figura 5.9 Ventana de Configuración FDA tool……123

Figura 5.10 Selección de tipo de Estructura de Filtro……124

Figura 5.11 Diseño de Filtro con coeficientes……126

Figura 5.12 Modelo de Generador de Código Walsh……127

Figura 5.13 Modelo de Generador de Código PN……..132

Figura 5.14 Modelo del Filtro Digital……137

Figura 5.15 Diagrama de Pruebas de Filtro……144

Figura 5.16 Respuesta en Magnitud filtro Pasa Bajo……146

Figura 5.17 Respuesta en Magnitud Real……146

Figura A.1 Diagrama de Bloques del DSK C6416.

Figura A.2 Mapeo de Memoria del C6416 DSK.

Figura C.1 Diagnostico General.

Figura C.2 Diagnostico Avanzado.

Figura C.3 Diagnostico Avanzado CODEC.

Figura C.4 Diagnostico Avanzado Memoria.

Figura C.5 Diagnostico Avanzado DSP.

Figura C.6 Diagnostico Avanzado Led/Swt.

Figura D.1 Estructura Filtro FIR.

Figura D.2 Coeficientes Filtro FIR.

Figura D.3 Respuesta en Frecuencia Filtro FIR.

Figura D.4 Respuesta al Impulso Unitario FIR.

Figura D.5 Respuesta en Magnitud y Fase FIR.

Figura D.6 Diagrama Digital FIR Forma Directa.

Figura D.7 Estructura Digital Fase Lineal.

Figura D.8 Estructura Digital en Cascada.

Figura D.9 Estructura Filtro IIR para N=M=2.

INDICE DE TABLAS.

Tabla I	Parámetros Delantero vs. Inverso	20
Tabla II	Tasa de repetición según velocidad	34
Tabla III	Memoria Interna y Externa	68
Tabla IV	Aplicación de Conjunto de bloques	77
Tabla V	Parámetros de Configuración Resolvedor	85
Tabla VI	Parámetros de Configuración Implementación de Hardware	85
Tabla VII	Parámetros de Configuración RTW	86
Tabla VIII	Configuración TI C6000 SELECCION PLATAFORMA	88
Tabla IX	Configuración TI C6000 GENERACION CODIGO	89
Tabla X	Coeficientes de h(k)	103
Tabla XI	Parámetros de Configuración Filtro	104
Tabla XII	Parámetros de Bloque Walsh	110
Tabla XIII	Parámetros de Bloque Workspace (Walsh)	110
Tabla XIV	Parámetros de Bloque Scope General (Walsh)	111
Tabla XV	Parámetros de Bloque Scope Data History (Walsh)	111
Tabla XVI	Parámetros de Bloque XOR	117
Tabla XVII	Parámetros de bloques Unit Delay	117
Tabla XVIII	Parámetros de bloque Workspace (PN)	118
Tabla XIX	Parámetros bloque Scope General (PN)	119
Tabla XX	Parámetros bloque Scope Data History (PN)	120

Tabla XXI Salida de Códigos Walsh……..128

Tabla XXII Representación de Símbolos bipolar……..129

Tabla XXIII Parámetros bloque Switch (Walsh)……129

Tabla XXIV Parámetros globales Walsh_Final_Hardware……130

Tabla XXV Salidas de Ramales I o Q……..133

Tabla XXVI Representación de Símbolos Unipolar……134

Tabla XXVII Parámetros bloque Switch (PN)……...135

Tabla XXVIII Parámetros globales RAMAL_I_Q_FINAL_HARDWARE.....135

Tabla XXIX Parámetros bloque ADC……137

Tabla XXX Parámetros bloque Diseño de Filtro…..……138

Tabla XXXI Parámetros bloque DAC (Filtro)...……139

Tabla XXXII Parámetros globales FILTRO_FINAL_HARDWARE……140

Tabla XXXIII Verificación de Códigos Walsh……141

Tabla XXXIV Verificación de salida en LEDs (Walsh). ……..142

Tabla XXXV Verificación de Códigos PN. ……..143

Tabla XXXVI Verificación de salida en LEDs (PN). ……143

Tabla XXXVII Valores obtenidos a la salida del Filtro……..145

Tabla C.1 Tipos de función Ventana.

Tabla E.1 Variable Codigo_Walsh.

Tabla E.2 Códigos Walsh bipolar.

INTRODUCCIÓN.

La frecuencia es un recurso limitado, por lo cual las tecnologías presentes en el mercado requieren se administre este recurso con la mayor efectividad posible. Es por eso que, uno de los puntos de interés es el acceso al medio. Este estudio se enfoca en el Acceso Múltiple por División de Código (CDMA), el mismo que, en una de sus partes constitutivas consta de un canal Piloto para establecer la comunicación entre móvil y base. Para el estudio del canal Piloto presente en la base, es necesario comprender las herramientas tecnológicas que el mundo contemporáneo pone a nuestra disposición:
Software (MATLAB y Code Composer Studio) y Hardware (Procesadores Digitales de Señal DSP) que hacen posible cualquier aplicación tecnológica.

CAPITULO 1

INTRODUCCIÓN AL SISTEMA CDMA.

1.1 PRINCIPIOS DE LAS REDES CELULARES CDMA.

En sus orígenes las comunicaciones móviles fueron pensadas sólo para proveer servicios de voz, pero al transcurrir de los años y con los avances de la tecnología estos servicios fueron desarrollándose a tal punto de que hoy en día se ofrecen un sinnúmero de aplicaciones tanto de entretenimiento como de aplicaciones empresariales, dentro de las aplicaciones disponibles hoy en día tenemos desde el servicio de mensajería SMS, hasta el servicio de televisión a través del móvil.

El principio básico sobre el cual se basa la arquitectura celular (sea esta CDMA o GSM) es la estructura de celdas gobernadas por una estación base, como indica la figura 1.1, la cual a su vez es monitoreada por un Centro de Conmutación Móvil (MSC o Mobile Switch Center).

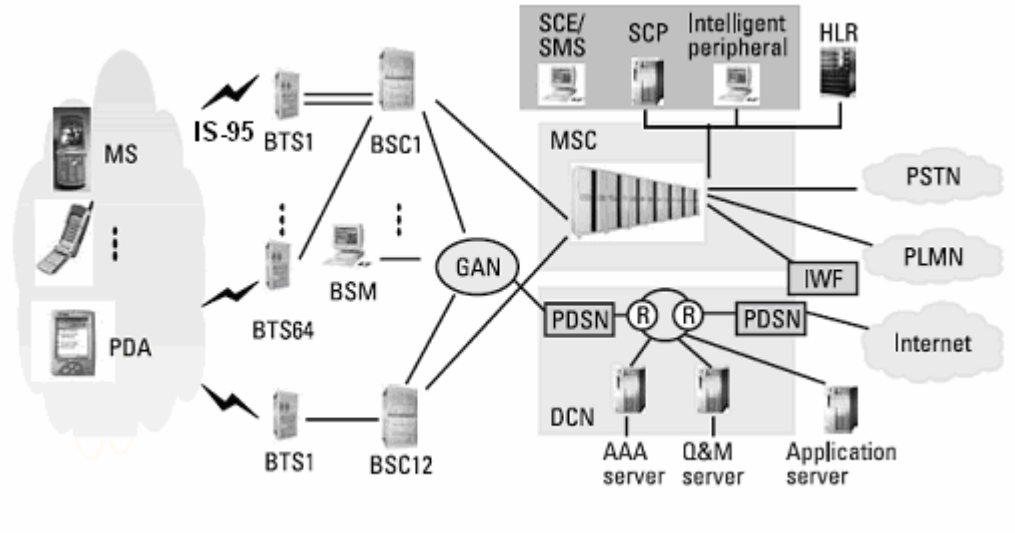


Figura 1.1 Red Inalámbrica CDMA.

1.2 ARQUITECTURA DE UNA RED INALÁMBRICA CELULAR.

La figura 1.2 ilustra un esquema simplificado de lo que es una red celular. Esta red consta de muchas Estaciones Base (BS) las cuales a su vez sirven a las Estaciones Móviles (Mobile Unit MS) en una determinada región geográfica. El sistema CDMA es full duplex, lo cual quiere decir que en cualquier instante de tiempo la comunicación puede ocurrir en ambos sentidos, en el enlace Delantero (desde BS a MS) o en el enlace Inverso (desde MS a BS). El enlace entre la antena de la estación base y la antena de la estación móvil, es comúnmente llamado canal de radio, y puede ser analizado con métodos similares a los usados en análisis de otros sistemas de comunicación inalámbrica.

Los usuarios del sistema celular pueden acceder a la red de cualquiera de las siguientes formas: Acceso Múltiple por División de Frecuencia (FDMA), Acceso Múltiple por División de Tiempo (TDMA) o Acceso Múltiple por División de Código (CDMA). En FDMA, a cada usuario se le asigna un único par de frecuencias, uno para transmisión y uno para recepción. En TDMA a cada usuario no solo se le asigna un par de frecuencias, sino también una ranura de tiempo, en el cual transmisión y recepción pueden ocurrir.

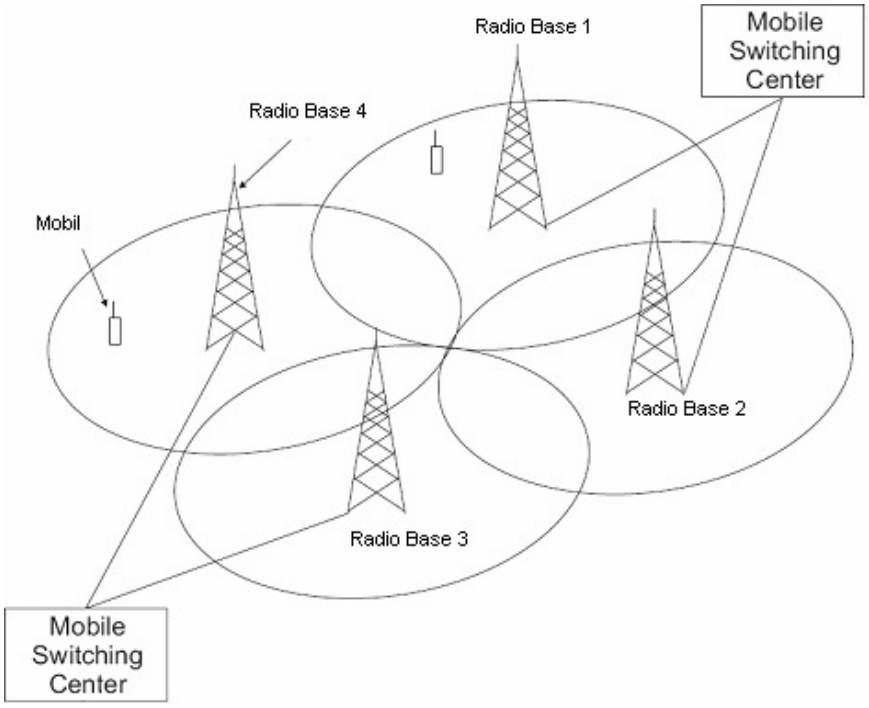


Figura 1.2 Esquema de una red celular.

Por ultimo en CDMA a cada usuario se le asigna un código único, el cual es transportado junto con la señal de usuario. Este código único, permite a la BS y a la MS distinguir entre cada una de las otras señales que circundan el ambiente. Adicionalmente en el sistema CDMA, las frecuencias a las cuales operan la transmisión y recepción son diferentes. Aunque el esquema de múltiple acceso empleado en sistemas celulares determina como la red es implementada, el factor clave, común en cualquier esquema de acceso es lidiar con la cobertura versus la interferencia.

En FDMA y TDMA, se asigna la misma frecuencia a dos usuarios diferentes solo si estos usuarios están suficientemente separados en distancia. Ajustar la potencia de transmisión de cada BS y la orientación de la antena, también ayuda a controlar la interferencia mutua experimentada por dos usuarios de la misma frecuencia. Idealmente se desea que cada BS sirva un área muy grande, lo que se puede lograr aumentado la potencia de transmisión, pero al incrementar la potencia de transmisión también se logra aumentar la interferencia, por lo tanto comúnmente lo que se suele hacer en estos casos es utilizar una potencia de transmisión baja y un gran número de BS, además se reutilizan las frecuencias en diversas regiones, esta práctica se conoce como reutilización de frecuencias.

En el sistema CDMA, aunque todos los usuarios transmiten en la misma frecuencia, no hay necesidad de implementar el reuso de frecuencias. Cada BS esta diseñada para acomodar el máximo número de usuarios posibles y las BS están situadas basadas en la demanda de los usuarios y la cobertura de radio. La figura 1.3 nos muestra gráficamente como se implementan cada uno de los diversos tipos de acceso al medio. En FDMA vemos claramente que cada usuario utiliza una frecuencia para la comunicación, en TDMA además de usar la frecuencia se asigna una ranura de tiempo para acomodar a *n* usuarios en determinada frecuencia. Por último, en CDMA aparentemente todos los usuarios están mezclados en una sola frecuencia, lo cual no es tan cierto, ya que, cada uno de estos usuarios es identificado por un código único que lo mantiene aislado de los otros usuarios.

En resumen, las redes inalámbricas celular proveen comunicación sobre el aire entre BS y MS. Estos enlaces son muy confiables en distancias relativamente pequeñas, típicamente decenas de metros a unos cuantos kilómetros.

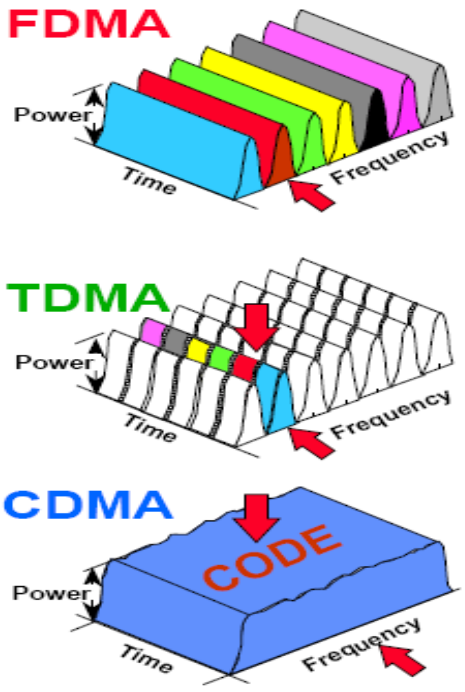


Figura 1.3 Uso de espectro FDMA vs. TDMA vs. CDMA.

Las BS se comunican a través de los MSC, los cuales a la vez se interconectan con redes externas como las redes telefónicas públicas conmutadas (Public Telephone Switching Network o PSTN), las redes digitales de servicios integrados (Integrated Services Digital Network o ISDN), y la Internet.

El móvil es, en cualquier caso libre de moverse a través de la red celular. Estos móviles en la actualidad incluyen internamente microcontroladores, procesadores digitales de señales (Digital

Processing Signal o DSP), memorias, cámaras; los cuales proveen un acondicionamiento de la señal antes de que esta sea transmitida, y también demodulan la señal recibida.

Las BS transforman las señales de radio en paquetes de datos y en mensajes de señalización que son entendibles por la red cableada, la cual a su vez envía la informaron al MSC. Luego de esto el MSC basado en la información de señalización enruta los paquetes de datos a su destino.

La red además incluye puntos de control que interactúan con computadores que proveen acceso a las bases de datos. Por ejemplo, el MSC usa un punto de control para acceder al Registro de ubicación base (Home Location Register o HLR), al Registro de ubicación de visitante (Visitor Location Register o VLR) y al centro de Operación y Mantenimiento (Operation and Maintenance Center o OMC). Estas bases de datos tienen una lista de los suscriptores dentro de su área de cobertura, rastrean suscriptores en roaming y mantienen un archivo de autenticación.

1.3 PÉRDIDAS Y PROPAGACIÓN EN AMBIENTES INALÁMBRICOS MÓVILES.

La interfaz de radio es única en comunicaciones inalámbricas, y es la responsable de gran parte de la complejidad asociada con redes inalámbricas y teléfonos móviles. La interfaz de radio entre el MS y la BS es conocida como el canal de comunicación y es afectada por factores de gran y mediana escala. Lo efectos de gran escala son debidos a la simple atenuación de la señal transmitida a través de la atmósfera, y los efectos de pequeña escala tienen un comportamiento impredecible, ya que son de rápido cambio y además son muy considerables en pequeñas distancias. Una señal transmitida es atenuada conforme se propaga a través de la atmósfera. Este efecto de larga escala conocido como Pérdida de Propagación y es modelado por la siguiente ecuación:

$$r(d) \propto d^{-n} \tag{1}$$

Donde $r(d)$ es la potencia recibida a una distancia d de separación entre el MS y la BS, y *n* es el exponente de pérdida de propagación el cual tiene valor típico de 2.7 a 3.5 para un área urbana. Este modelo es muy simple y es apropiado solo para modelos con línea de vista.

En la práctica, la señal es normalmente obstruida, bloqueada y reflejada; además se le introduce variabilidad estadística al modelo de

pérdida de propagación simple. Este efecto es conocido como *ensombrecimiento* y es modelado como una variable aleatoria Log-normal. Esto lleva a una nueva expresión para la potencia recibida:

$$r(d) \propto 10^{\chi/10}\, d^{-n} \tag{2}$$

Donde χ es la variable aleatoria Log-normal usada para modelar en efecto de *ensombrecimiento*.

1.4 EL ROL DE CANAL PILOTO EN UN SISTEMA CDMA.

Los sistemas CDMA celular (800 – 900 MHz) y PCS (1800 – 1900 MHz) cumplen con los estándares IS-95 y J-STD-008 respectivamente. Estos estándares indican que una señal Piloto, la cual es una portadora modulada por una alta tasa de chip (1.2288 MHz), sea transmitida en el enlace Delantero. A mas de esto, todos las unidades móviles tienen un conocimiento anticipado de esta secuencia PN[1]. Como la secuencia PN es periódica, con un periodo de 2^{15} chips, entonces la señal Piloto puede tratarse en términos de su fase. A continuación se ilustran los múltiples propósitos de la utilización de una señal Piloto.

[1] Secuencia Pseudoaleatoria. Una señal digital con propiedades de ruido.

1.4.1 SINCRONIZACIÓN DE LA UNIDAD MÓVIL.

La operación de demodulación de una señal CDMA requiere que el receptor CDMA genere una versión sincrónica de la señal original. Por lo tanto, uno de los principales propósitos de la señal Piloto es de ayudar a los móviles a alcanzar tal sincronismo con la señal recibida. Por tal razón, la potencia de la señal Piloto es generalmente mayor que la potencia de cualquiera de los otros canales del sistema. El móvil se sincroniza con la señal Piloto emitida por la BS por medio de un proceso de correlación entre la señal que recibe y la señal Piloto generada de manera local por el móvil. Estos procesos de correlación pueden ser implementados utilizando técnicas de procesamiento digital de señales.

1.4.2 HANDOFF.

El handoff ocurre cuando una llamada es transferida entre dos BS o cuando dos o más BS pueden soportar la misma llamada. Esto último es conocido como Soft-Handoff, cuando dos BS pueden soportar una misma llamada, o soft-soft-handoff, cuando tres BS pueden soportar la misma llamada. Estos tipos de soft handoff son posibles debido al uso de un receptor RAKE[2], donde más de tres

[2] Un receptor Rake usa varios correlacionadores de banda base para procesar individualmente varias componentes de la señal multitrayecto. La salida del correlacionador se combinan para obtener una mejor señal.

correlacionadores están disponibles en el móvil para recibir y combinar las señales CDMA. Los móviles miden la potencia de las señales Piloto recibidas de parte de las diversas BS próximas a él, y de esta manera determina cual estación (estaciones) base podrían tomar la llamada.

1.4.3 LOCALIZACIÓN DE POSICIÓN.

Para citar un ejemplo en USA, las regulaciones de la Comisión Federal de Comunicaciones (Federal Communication Commission o FCC) obligan a los proveedores de servicios celular y PCS a la localización de las MS en sus respectivas áreas de servicio. En consecuencia la industria tecnológica inalámbrica y la comunidad académica están uniendo esfuerzos para desarrollar y evaluar varios algoritmos de localización de posición. Una posible solución que aún esta en fase de investigación para sistemas CDMA, requiere que la unidad móvil mida las fases de las señales Piloto que llegan de las diferentes BS a su alrededor, para de esta manera determinar su posición. En la actualidad este sistema esta implementado y su funcionalidad va desde resolución de casos criminales hasta el uso en deportes donde se requiere ubicarse con gran exactitud.

1.5 EL FUNDAMENTO DEL SISTEMA CDMA.

La figura 1.4 muestra un diagrama de bloques funcional de un sistema de comunicaciones típico. La fuente de la información, que en este caso es el habla humana, es primeramente convertida en una forma digital por un bloque codificador de fuente. Entonces la función de codificación de canal codifica la información digital con el propósito de contrarrestar los efectos que provocan la degradación del canal. Luego la información es combinada por la función de acceso múltiple, de esta forma más de un usuario puede compartir el espectro dado. La función moduladora convierte la información en bandabase a una forma de onda pasabanda (RF) que puede ser transmitida.

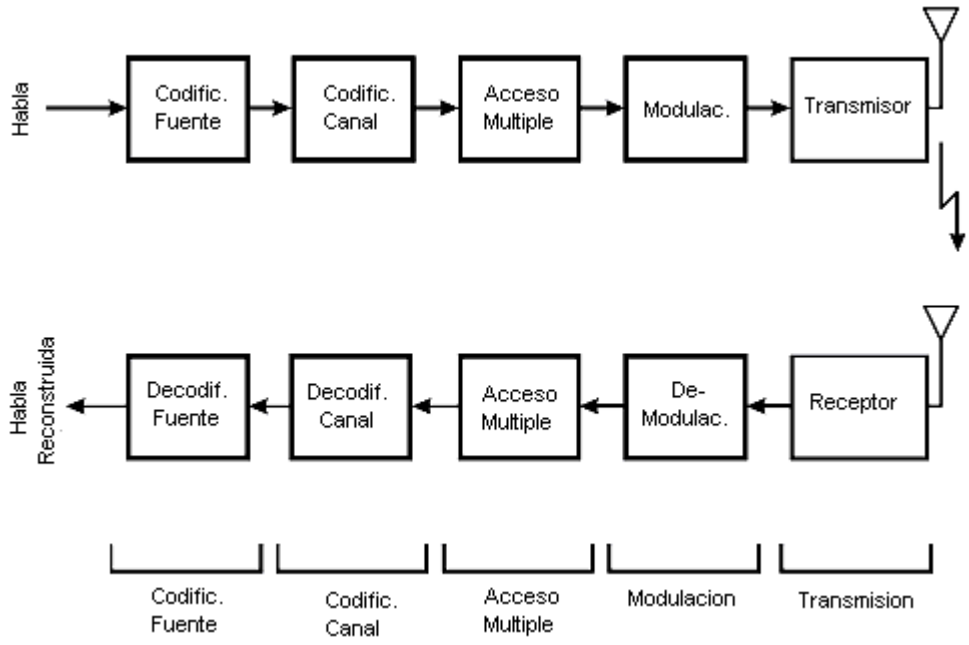


Figura 1.4 Sistema de comunicación típico.

En el lado del receptor, la forma de onda es interceptada. La señal es primero demodulada de RF a bandabase, luego la función de múltiple acceso separa a los diferentes usuarios que están compartiendo un espectro. Después la función codificadora de canal corrige los errores que se han introducido por el canal. Luego la función codificadora de fuente convierte la información bandabase en habla analógica.

1.6 COMPONENTES DE LOS SISTEMAS DE COMUNICACIÓN.

Los sistemas de comunicación básicamente constan de los siguientes componentes; una fuente de información, una etapa de codificación de fuente, que realiza los procedimientos adecuados para convertir la información de analógica a digital o viceversa, luego una codificación de canal se encarga de agregar a la información ciertos bits de cabecera con información variada para evitar y corregir errores, después una etapa de acceso múltiple (en el caso de CDMA) que sirve para permitir que varios usuarios compartan el espectro asignado, por último, la modulación que adecua a la señal para que pueda ser transmitida por el medio. A continuación de presentan con mas detalle las etapas mencionadas.

1.6.1 CODIFICACIÓN DE FUENTE.

La información fuente debe ser de alguna forma codificada en forma digital para ser luego procesada por el sistema de comunicación. Una de las técnicas más usadas en redes fijas es Modulación por código de pulsos (Pulse Code Modulation o PCM), pero PCM no es aplicable a sistemas inalámbricos por el ancho de banda limitado de dichos sistemas. Una alternativa es el uso de Vocoders, dichos dispositivos potencian las características del habla humana y usan pocos bits para representar y replicar los sonidos humanos. Los Vocoders pueden transformar el habla en una cadena de bits de 9.6 Kbps (para Tasa de datos 1) que ciertamente consume un menor ancho de banda que PCM de 64 Kbps. A mas de esto, los Vocoder toman ventaja del hecho de que en una conversación siempre habrá periodos cortos de silencio, por lo tanto los Vocoder pueden ajustar su tasa de datos dinámicamente para hacer un uso óptimo del espectro, pasando de una tasa de 9600, 4800, 2400 a 1200 bps, y dado el hecho de que existe una cantidad fija de ancho de banda disponible, una reducción del ancho de banda de los canales de voz individuales implica un aumento del número total de canales de voz disponibles.

1.6.2 CODIFICACIÓN DE CANAL.

Una vez que la información fuente ha sido codificada en forma digital, se necesita agregar cierta redundancia a esta señal digital bandabase. Esto se hace para mejorar el rendimiento del sistema de comunicación, haciendo a la señal más robusta frente a los daños intrínsecos del canal, como son el ruido y desvanecimientos. La meta principal de la codificación del canal, es dar una probabilidad de error baja, reducir el parámetro E_b / N_o. El precio a pagar para alcanzar esta meta es más ancho de banda, o más bits redundantes que el sistema tendrá que transmitir.

Al hablar de codificación de canal lo que estamos realmente haciendo es agregando mas bits a los bits de información, para que, en caso de producirse errores durante la transmisión sea mucho mas fácil en el lado del receptor encontrarlos y corregirlos haciendo uso de estos bits extra que fueron añadidos a la información original.

1.6.3 ACCESO MÚLTIPLE.

Una vez que la señal bandabase ha pasado por el proceso de codificación de canal, para control de errores, la señal es transformada para permitir lo que se conoce como acceso múltiple. Acceso múltiple implica compartir un mismo recurso para así permitir

comunicación simultánea entre múltiples usuarios, este recurso común a compartir es el espectro RF.

En CDMA la señal de cada usuario en ensanchada a un ancho de banda mayor al necesario para transmitir la información. Este ensanchamiento se realiza mediante el uso de un código ensanchador. Cada uno de estos códigos ensanchadores son ortogonales unos con otros, por lo tanto usando este conjunto de códigos ensanchadores se puede canalizar a muchos usuarios y hacerlos que compartan sin problemas el espectro.

1.6.3.1 CÓDIGOS WALSH.

En los sistemas CDMA, todos los usuarios transmiten en la misma banda RF. Para combatir la interferencia mutua en el enlace Delantero, los códigos Walsh son usados para separar a los usuarios individuales, mientras todos simultáneamente ocupan la misma banda RF. Los códigos Walsh usados son un conjunto binario de secuencias ortogonales. Estas secuencias son ortogonales entre ellas, y son generadas utilizando la matriz Hadamard.

$$H_{2N} = \begin{bmatrix} H_N & H_N \\ H_N & \overline{H_N} \end{bmatrix} \tag{3}$$

La matriz semilla es:

$$H_2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

Por lo tanto, para obtener un conjunto de cuatro secuencias Walsh ortogonales W_0, W_1, W_2, W_3, necesitamos generar la matriz Hadamard de orden 4,

$$H_4 = \begin{bmatrix} H_2 & H_2 \\ H_2 & \overline{H_2} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Las cuatro secuencias ortogonales se toman de las filas de la matriz H_4, así:

$$W_o = [0 \quad 0 \quad 0 \quad 0]$$

$$W_1 = [0 \quad 1 \quad 0 \quad 1]$$

$$W_2 = [0 \quad 0 \quad 1 \quad 1]$$

$$W_3 = [0 \quad 1 \quad 1 \quad 0]$$

El enlace Delantero usa un conjunto de 64 secuencias Walsh ortogonales, así la limitación física en el número de canales en el enlace Delantero es 63 porque W_o no se usa para transmitir ninguna información bandabase.

1.6.3.2 CÓDIGOS PN.

Aunque el enlace Delantero tiene un canal Piloto y Sincronismo para dar sincronización, el enlace Inverso no los tiene. Los móviles transmiten a su voluntad y no tienen ninguna clase de sincronización, así, los códigos Walsh no pueden ser usados en el enlace Inverso. La naturaleza incoherente del enlace Inverso requiere el uso de otra clase de códigos, aquí entran en juego los códigos PN. Por lo tanto en el enlace Inverso se usan los códigos PN para canalización.

En el enlace Inverso de sistemas CDMA se usa un código PN largo para la canalización. Se denomina código largo porque su tamaño es literalmente grande, su longitud es de $2^{42}-1$ chips y se genera empleando un registro de 42 estados. En la sección 1.6.3.1 vimos que el enlace Delantero usa los códigos Walsh para canalización de los usuarios de una BS en particular. Además el enlace Delantero también usa los códigos PN, aquí en cambio se

tiene que a cada BS se le asigna un código PN único, esto se hace para aislar cada base de las otras, este aislamiento es necesario porque cada BS usa el mismo conjunto de 64 códigos Walsh. El código PN usado en el enlace Delantero es llamado "corto". Este código PN corto se genera usando un registro de 15 estados y tiene una longitud de $2^{15} - 1$ chips.

La tabla que se muestra a continuación nos da una comparación entre los parámetros del enlace Delantero e Inverso.

COMPARACIÓN DELANTERO VS. INVERSO		
FUNCIÓN	**DELANTERO**	**INVERSO**
Modulación	QPSK	OQPSK
Código Walsh	Canalización	Secuencia Ensanchadora
Short Code (PN offset)	Identificar BS	
Long code y long-code mask	Privacidad y seguridad	Canalización (Ident. Del móvil)

Tabla I Parámetros de Delantero vs. Inverso.

1.6.4 MODULACIÓN.

La información digital debe ser modulada en una portadora RF para que pueda ser transmitida. Una vez modulada, la señal es entonces transmitida a través del espacio en forma de una onda electromagnética. Pero, por que se debe modular la información digital? Las razones son: Las regulaciones de los gobiernos especifican las frecuencias a las cuales un particular servicio puede transmitir, así, no todos pueden transmitir en bandabase. Y además de esto, si por ejemplo, quisiéramos transmitir en bandabase, la altura de la antena seria demasiado grande, lo cual en el ámbito real no es viable.

En el sistema CDMA se usa el sistema de modulación QPSK (Quadrature Phase-Shift Keying). El enlace Inverso usa una variante de QPSK llamada OQPSK offset quadrature phase-shift keying, OQPSK difiere de QPSK convencional en que antes de la multiplicación por la portadora, se añade un retraso de medio bit (respecto al ramal I) en el ramal Q. Este retraso se añade para prevenir una transición de fase de 180 grados que ocurre en la modulación QPSK convencional. Por ejemplo, cuando el símbolo 0 pasa a símbolo 3, la señal a través del origen haciendo una transición de fase de 180 grados.

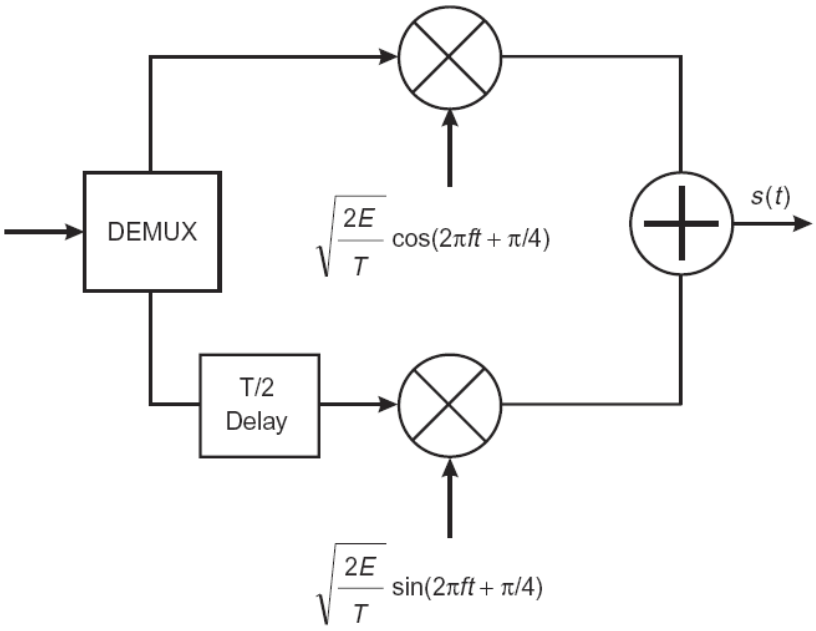


Figura 1.5 Modulador OQPSK.

En el dominio del tiempo, la envolvente de la señal colapsa y llega a cero momentáneamente. Este cruce por cero demanda un rango dinámico del amplificador de potencia. Así, OQPSK es usado en el enlace Inverso donde el amplificador del móvil es limitado en tamaño y rendimiento. El retraso añadido de medio bit en el ramal Q del Modulador asegura que no habrá transición entre los símbolos 0 y 2 ni entre los símbolos 1 y 3, y por lo tanto no habrá cruce por cero.

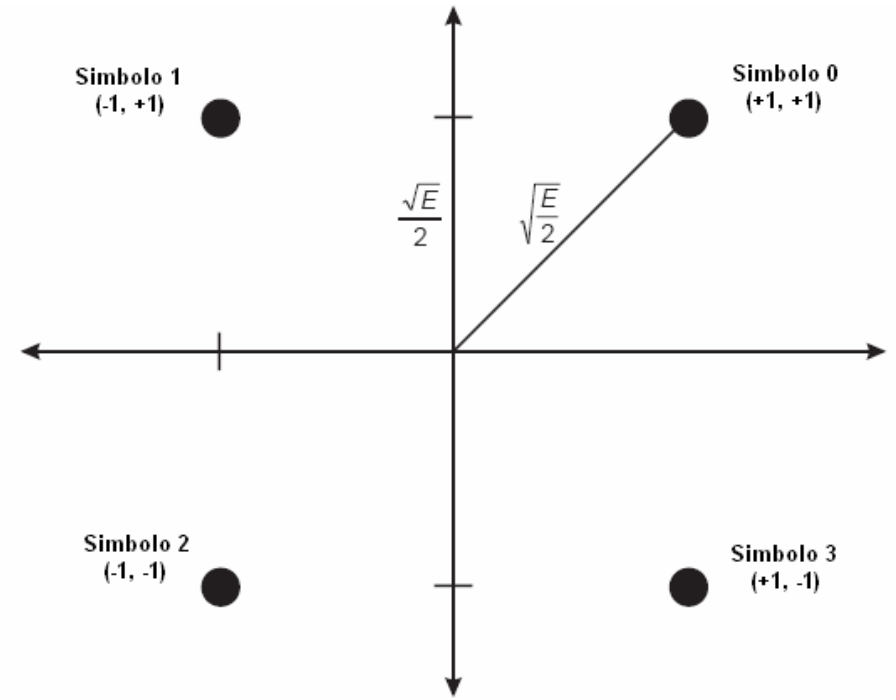


Figura 1.6 Representación de constelación QPSK.

1.7 CDMA Y COMUNICACIÓN EN ESPECTRO ENSANCHADO.

El sistema CDMA es un esquema de acceso múltiple basado en las técnicas de comunicación en espectro ensanchado. En este sistema lo que se hace es “ensanchar” la señal mensaje a un ancho de banda relativamente amplio mediante el uso de códigos únicos los cuales reducen la interferencia, mejorando el procesamiento del sistema y diferenciando a los usuarios. El sistema CDMA no requiere del esquema de acceso por división de tiempo ni de frecuencia, así se mejora de manera considerable la capacidad del sistema. En este

capitulo se presenta una introducción a las técnicas de espectro ensanchado y conceptos básicos de CDMA.

En un sistema de comunicación típico, basado en espectro ensanchado, la señal de mensaje es primeramente modulada con las técnicas tradicionales de modulación en amplitud, frecuencia o fase. Luego una señal de ruido pseudo aleatorio (PN) es aplicada para ensanchar la señal modulada a un ancho de banda mayor al de la señal original. La señal PN puede modular en amplitud la señal mensaje (para generar DS-SS Direct Sequence Spread Spectrum) o puede desplazar la frecuencia de portadora de la señal mensaje (para producir FH-SS Frecuency Hopped Spread Spectrum). Para generar una señal DS-SS se debe multiplicar la señal mensaje *d(t)* por una señal de ruido pseudo aleatorio *pn(t):*

$$g(t) = pn(t)\, d(t) \tag{4}$$

En la mayoría de los casos, la señal PN es de una tasa de datos muy elevada, y además es una secuencia NRZ (sin retorno a cero) que corta la señal mensaje modulada en “chips”. A partir de esto, la tasa de la señal modulada secundaria es llamada *tasa de chip (fc),* mientras la tasa de la señal mensaje se denomina *tasa de bit (fb).* Los procesos de

modulación producen dos anchos de banda conocidos como *R* para la señal mensaje modulada y *W* para la señal ensanchada. Hay que destacar que esta modulación secundaria no incrementa la potencia total de la señal mensaje, únicamente amplia el ancho de banda a uno mayor que el original.

Los códigos ensanchadores son un componente crítico de los sistemas de comunicación de espectro ensanchado. Estos generan la señal pseudo aleatoria empleada en el proceso de ensanchar la señal mensaje. De manera general, la señal ensanchadora es una forma de onda binaria. Esta señal binaria permite la fácil implementación sin sacrificar el rendimiento y permite la sincronización del transmisor con la señal recibida. Las principales características de las secuencias pseudo aleatorias son:

- Tienen una longitud de corrida de *r* chips, y se repiten aproximadamente 2^{-r} veces.
- Desplazando un número de chips se genera una nueva secuencia que tiene un igual número de acuerdos y desacuerdos con la secuencia original.
- Tienen una ocurrencia de chips +1 y -1 casi igual.

1.8 LA ESTRUCTURA DEL ENLACE CDMA.

El sistema CDMA es único en el sentido de que su enlace Delantero e Inverso tienen diferentes estructuras. Esta diferencia en sus estructuras es necesaria para acomodar los requerimientos del sistema de comunicación móvil. El enlace Delantero consiste de cuatro tipos de canales lógicos, los cuales son: Piloto, Sincronismo, Tráfico y Paginación. Hay un canal Piloto, un canal de Sincronismo, unos 7 canales de Paginación y muchos canales de Tráfico. Cada uno de estos canales es primero ensanchado ortogonalmente por una función Walsh, luego es ensanchado por una secuencia PN corta. Luego todos los canales son sumados para formar la señal compuesta a ser transmitida en el enlace Delantero. Por otra parte, el enlace Inverso consiste de dos tipos de canales lógicos: Acceso y Tráfico. Cada uno de estos canales son ensanchados ortogonalmente por una secuencia PN larga, así cada canal es identificado usando un código PN largo distinto. La razón de por que no se usa Piloto en el canal Inverso, es que es impractico que cada móvil transmita su propia secuencia del Piloto (excepto en WCDA donde el móvil genera su propio Piloto).

1.8.1 ENLACE DELANTERO.

En la sección 1.6.3.1, se definió la estructura de la matriz Hadamard, y se describió como son generados los códigos Walsh

con tal matriz. En CDMA se usa una matriz Hadamard de 64 x 64 para generar 64 funciones Walsh que son ortogonales unas a otras, con esto cada uno de los canales lógicos en el enlace Delantero son identificados por su función Walsh asignada.

1.8.1.1 EL CANAL PILOTO.

El canal Piloto tiene básicamente tres funciones principales: estimación del canal para una demodulación coherente, ayudar en el proceso de handoff, y detección de multitrayectoria. El canal Piloto es un canal común transmitido a múltiples usuarios. El canal Piloto se identifica por la función Walsh número 0 (*Wo*). Este canal en si no transporta información bandabase. La secuencia bandabase es una cadena de ceros que son ensanchados por la función Walsh 0, la cual es también una secuencia de ceros. La secuencia resultante (todavía todos ceros) es entonces ensanchada, o multiplicada por un par de secuencias PN. Por lo tanto el canal Piloto es efectivamente una secuencia PN en si misma, ver la figura 1.7. La secuencia PN con un offset específico identifica únicamente el sector particular que esta transmitiendo la señal Piloto. Hay que notar que ambas, la función Walsh 0 y la secuencia PN están transmitiendo a una tasa de datos de 1.2288

Mcps. Después del ensanchamiento PN, se usan los filtros bandabase para formar los pulsos digitales.

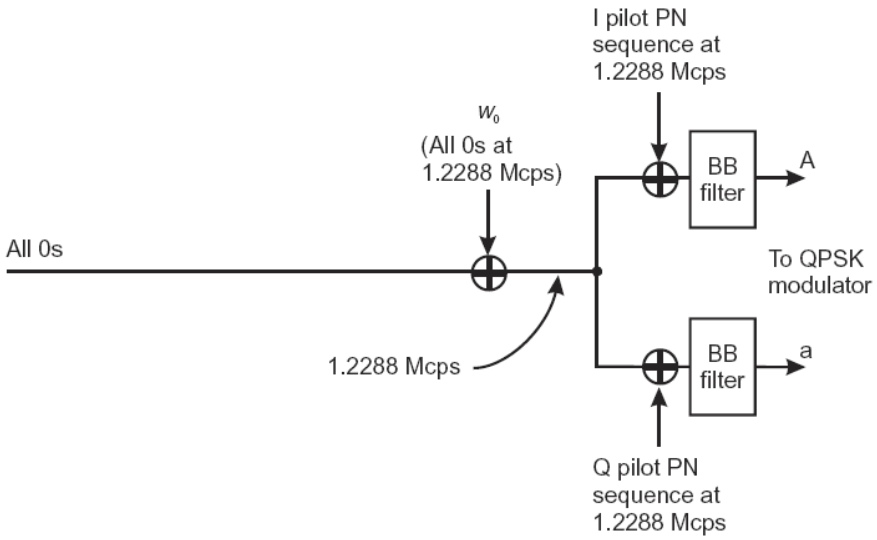


Figura 1.7 Estructura de un canal Piloto.

Estos filtros controlan de manera efectiva los pulsos y el espectro bandabase de la señal, así, el ancho de banda de la señal puede tener una roll-off[3] más definida cerca del límite de la banda.

El canal Piloto es transmitido continuamente por la BS, esto provee al móvil con temporizacion y referencia de fase. La medición que el móvil realiza de la razón “señal a ruido” del canal

[3] El término ROLL-OFF se refiere a la caída en el límite de la banda, e indica la atenuación de un filtro pasa bajos o pasa altos.

Piloto, también da una indicación de cual es el sector que lo puede servir de una mejor manera.

1.8.1.2 CANAL DE SINCRONISMO.

El canal Sincronismo si lleva información bandabase, esta información notifica al móvil sobre la sincronización del sistema y otros parámetros. La figura 1.8 muestra que la información bandabase es protegida contra errores y pasada por el bloque que realiza una aleatorización de bits. Luego es ensanchada por la función Walsh número 32 y ensanchada por una secuencia PN que es identificada con el sector de servicio.

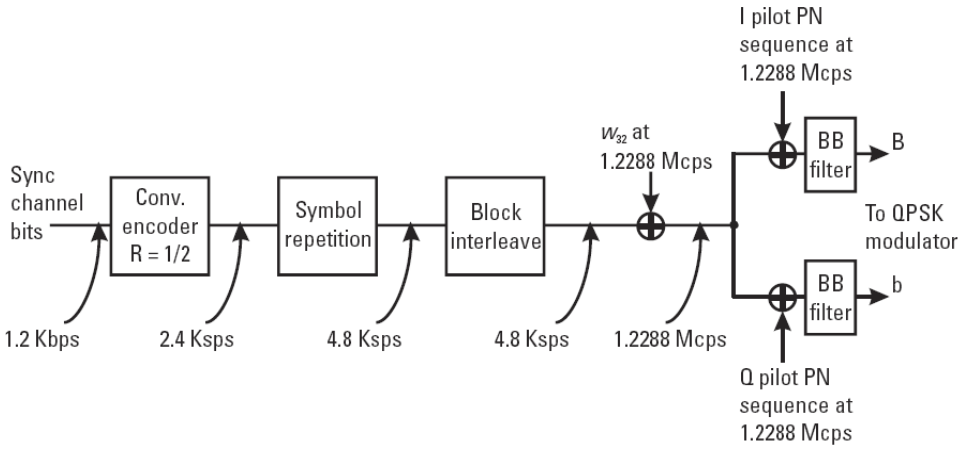


Figura 1.8 Estructura de canal de sincronismo.

La información bandabase esta a una tasa de 1.2 Kbps. Cada trama del canal de Sincronismo esta alineada con la secuencia PN

corta asociada con el sector de la celda que transmite. Hay que recordar que la secuencia PN se repite cada 26.67 ms, y que cada período de la secuencia PN corta esta sincronizado con cada trama del canal de Sincronismo. Por lo tanto, una vez que el móvil se sincroniza con el canal Piloto, la alineación con el canal de Sincronismo es inmediatamente conocida, esto se debe a que el canal de Sincronismo es ensanchado con la misma secuencia PN del Piloto, y además porque la temporización de la trama del Sincronismo esta alineada con la de la secuencia PN del Piloto. Finalmente, una vez que el móvil se logra alinear con el canal Sincronismo, el móvil puede empezar a leer los mensajes de canal Sincronismo.

1.8.1.3 CANAL DE PAGINACION.

De manera similar al canal de Sincronismo, el canal de Paginación también transporta información bandabase. Pero a diferencia del canal de Sincronismo, el canal de Paginación transmite a tasas de datos mas altas, las cuales pueden ser 4.8 o 9.6 Kbps. Una vez que el móvil adquiere temporización y sincronización a través del canal de Sincronismo, el móvil comienza a monitorear el canal de Paginación. Aunque puede haber hasta 7 canales de Paginación por sector, cada móvil solo monitorea un canal de Paginación.

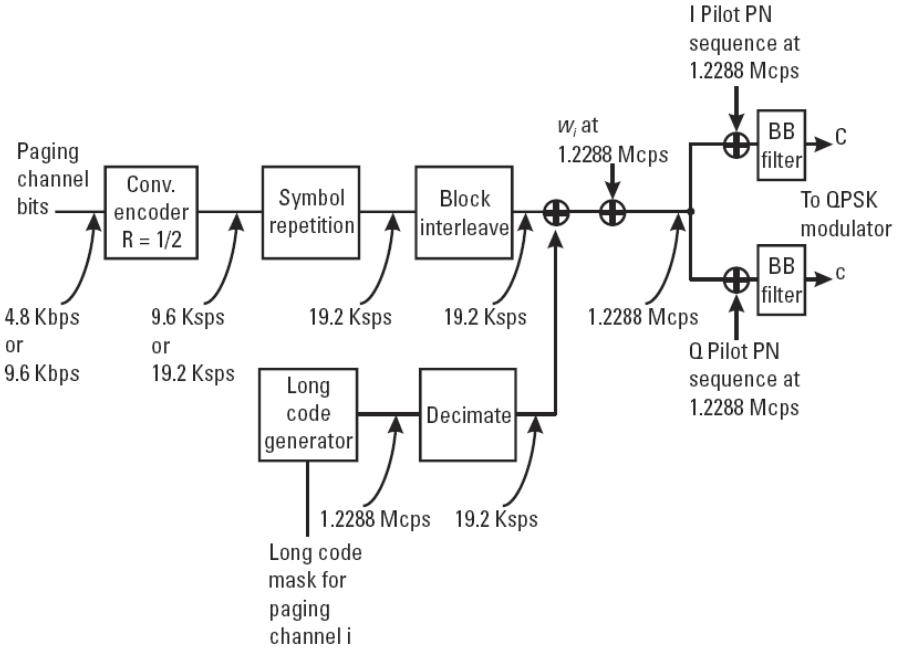


Figura 1.9 Estructura de canal de Paginación.

Como podemos observar en la figura 1.9, la información bandabase es primeramente protegida contra errores, y si la tasa de datos es de 4.8 Kbps, los bits son repetidos una vez, caso contrario no se repiten. Luego del revuelta, la información es pasada por un aleatorizador, luego es ensanchada por una función Walsh asignada, y luego es ensanchada por la secuencia PN corta asignada al sector de servicio. Nótese de la figura 1.9 que la decimación tiene una razón de 64:1, ya que va de 1.2288 Mcps a 19.2 Ksps. El generador de código largo se enmascara con una

mascara única especifica a cada uno de los canales de Paginación (de 1 a 7). Por lo tanto, la máscara del código usada para el canal de Paginación 1 (ensanchado por la función Walsh 1) es diferente de aquella usada por el canal de Paginación 3 (ensanchado por la función Walsh 3).

1.8.1.4 CANAL DE TRÁFICO.

El canal de Tráfico es usado para transmitir voz y datos de usuario; mensajes de señalización también son enviados sobre el canal de Tráfico. La estructura del canal de Tráfico es similar a la estructura del canal de Paginación. La única diferencia es que el canal de Tráfico contiene Bits de Control de Potencia (Power Control Bits o PCB) multiplexados. La figura 1.10 muestra el canal de Tráfico para tasa de datos 1, para esta tasa el Vocoder puede variar su tasa de datos de salida en función de la actividad de voz. Se pueden tener 4 tasas de datos: 9.6, 4.8, 2.4 y 1.2 Kbps. Por ejemplo, durante períodos de silencio, el Vocoder puede elegir 1.2 Kbps que es la tasa de datos mas baja.

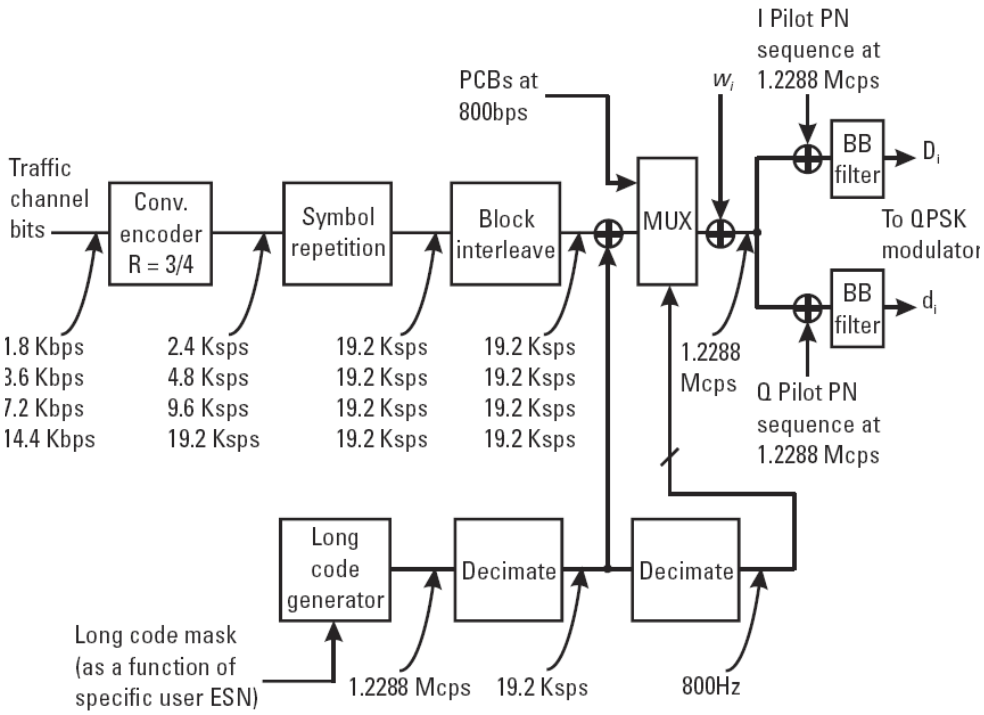


Figura 1.10 Estructura de canal de Tráfico.

La información en bandabase del Vocoder es codificada para protegerla de errores, para el tasa de datos 1, se usa un codificador convolucional de tasa ½. Luego de la codificación convolucional, se realiza una repetición de símbolos, la cual repite los símbolos cuando se producen bajas tasas de datos en el Vocoder. A continuación se presenta una tabla explicativa:

TASA DE DATOS (Kbps)	REPETICIÓN
9.6	Sin repetición
4.8	Cada símbolo se repite 1 vez
2.4	Cada símbolo se repite 3 veces
1.2	Cada símbolo se repite 7 veces

Tabla II Tasa de repetición según velocidad.

La razón para que se realice una repetición de símbolos es reducir interferencia en un tiempo dado cuando tasas de datos bajas son transmitidas, otras palabras, disminuir la potencia por símbolo repetido cuando el Vocoder esta a bajas tasas de datos. El esquema toma ventaja del factor de actividad de voz. En un sistema CDMA real, cuando el Vocoder transmite a 4.8 Kbps, la energía por símbolo transmitida es la mitad de cuando el Vocoder transmite a 9.6 Kbps. Cuando el Vocoder transmite a 2.4 Kbps, la energía por símbolo es un cuarto de cuando el Vocoder transmite a 9.6 Kbps, y cuando corre a 1.2 Kbps, la energía por símbolo es un octavo de la energía por símbolo cuando transmite a 9.6 Kbps. Continuando con el esquema, tenemos que seguido de la

repetición de símbolos, los datos son revueltos con el fin de combatir el desvanecimiento, luego estos datos pasan a un aleatorizador por una secuencia PN larga. La secuencia PN larga es generada por un generador de código PN el cual arroja a la salida una secuencia PN larga a una tasa de 1.2288 Mcps. Ya que la tasa de datos a la salida del aleatorizador es 19.2 Kcps, la secuencia PN es decimada con una razón de 64:1 para alcanzar 19.2 Kcps; la secuencia PN decimada a 19.2 Kcps es entonces multiplicada con la información a 19.2 Ksps. Cabe mencionar que el generador de código largo produce una secuencia PN larga usando una máscara que es especifica para el móvil; en la práctica esta máscara es función del número de serie electrónico (Electronic Serial Number o ESN) del móvil.

Los PCB (Power Control Bits o Bits de Control de Potencia) a 800 bps son multiplexados con la cadena que salió del scrambler a 19.2 Ksps. Un PCB puede ser pinchado[4] en alguno de los primeros 16 bits del Grupo de Control de Potencia (Power Control Bit o PCG) (el cual contiene 24 bits). La posición exacta del PCB en el PCG es determinada pseudoaleatoriamente.

[4] El termino “pinchado” se refiere al hecho de poner un bit de control de potencia en lugar de un bit de información, en alguno de los 16 primeros bits de cada grupo de control de potencia.

Mas específicamente, dado que, la entrada del decimador es la secuencia PN larga, la posición del PCB es determinada por el valor decimal de los cuatro bits más significativos de la salida del decimador. Es importante reconocer que la posición exacta del PCB dentro del PCG no es una posición fija. En este punto, la cadena de datos multiplexados (aún a 19.2 Ksps) es ensanchada ortogonalmente por la función Walsh asignada. La función Walsh está a una tasa de 1.2288 Mcps; cada símbolo es ensanchado por un factor de 64, y el resultado es una cadena de datos ensanchada a una tasa de 1.2288 Mcps. La cadena de datos es además ensanchada por la secuencia PN corta asignada del sector de transmisión. La secuencia PN corta provee una segunda capa de aislamiento, que discrimina entre los diversos sectores de transmisión. De esta manera, todas las 64 funciones Walsh disponibles pueden ser reutilizadas en cada sector, cabe recalcar que cada secuencia PN corta única es caracterizada por su PN offset.

La estructura del canal de Tráfico para tasa de datos 2 es similar. Los Vocoder de tasa de datos 2 son más veloces y además entregan una mejor calidad de voz que los Vocoder de tasa de

datos 1. El Vocoder para tasa de datos 2 maneja 4 tasas: 14.4, 7.2, 3.6 y 1.8 Ksps.

La salida de los canales lógicos ingresan a un modulador. La figura 1.11 muestra la estructura del modulador del enlace Delantero.

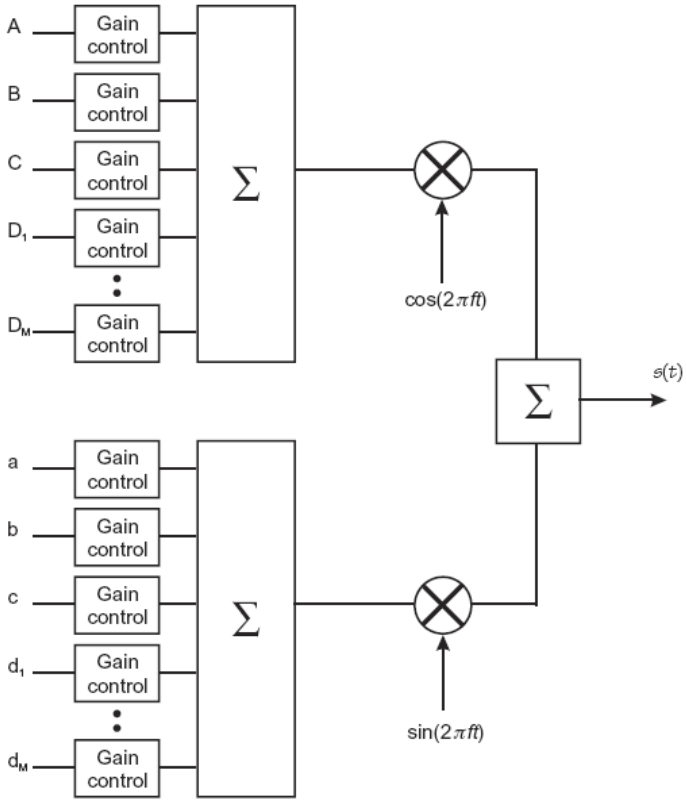


Figura 1.11 Estructura de modulador del Enlace Delantero.

La ganancia de cada canal lógico, incluyendo al Piloto, sincronismo, paginación y Tráfico, es primero ajustada por la

función de control de ganancia. La ganancia de cada canal indica cuanta potencia va a ser transmitida por dicho canal. Cabe mencionar que la potencia de los canales de Tráfico individuales cambian de manera dinámica, ya que ellos son controlados por el proceso de control de potencia.

Una vez ajustada la ganancia del canal, las señales son sumadas coherentemente para formar la señal compuesta con espectro ensanchado. Después de la suma, los ramales I y Q son modulados por sus respectivas portadoras, luego estas señales son sumadas para formar la señal QPSK pasabanda.

1.8.2 EL ENLACE INVERSO.

El enlace Inverso soporta dos tipos de canales lógicos los cuales son: canal de Acceso y canal de Tráfico. Debido a la naturaleza no coherente del enlace Inverso, no se usan funciones para la canalización, en lugar de eso, secuencias PN cortas son usadas para distinguir entre un usuario y otro.

A continuación se explica en detalle las partes del canal de comunicación denominado Enlace Inverso.

1.8.2.1 EL CANAL DE ACCESO.

El canal de Acceso es usado por el móvil para comunicarse con la BS cuando el móvil no tiene un canal de Tráfico asignado. El móvil usa este canal para originación de llamadas y responder mensajes y ordenes. La tasa de datos en bandabase es fija en 4.8 Kbps. Como se muestra en la figura 1.12, la información es primero protegida contra errores por un codificador convolucional de R=1/3. La tasa de codificación mas baja hace que la protección sea mas robusta en el enlace Inverso, ya que normalmente es

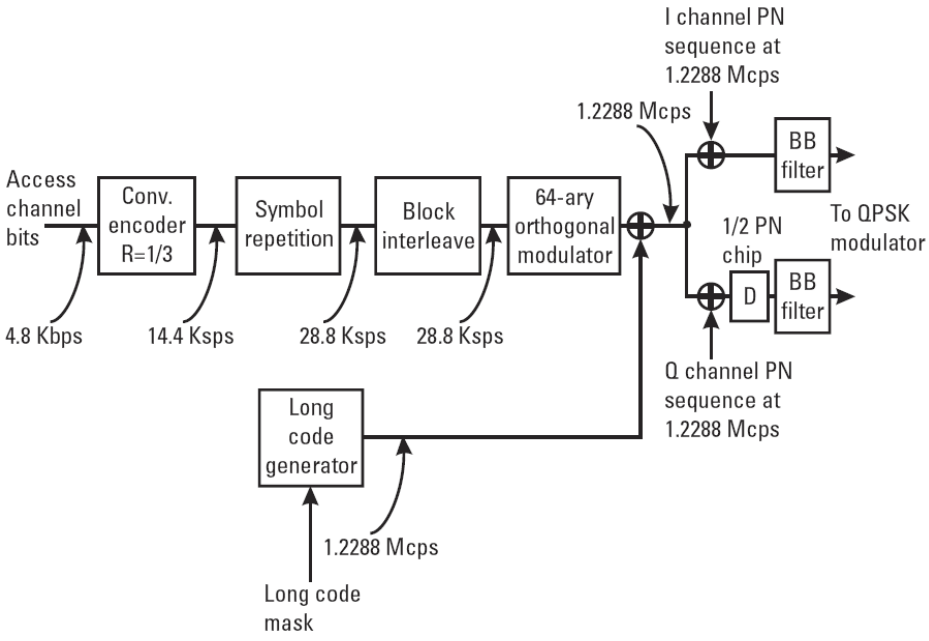


Figura 1.12 Estructura de canal de Acceso.

siempre el enlace más débil entre los dos enlaces. La función repetición de símbolos repite los símbolos una vez, y los datos entran a un interleaver para combatir el desvanecimiento. Luego del interleaver, la información es codificada por un modulador ortogonal 64-ario.

El conjunto de 64 funciones Walsh también es usado, pero aquí las funciones Walsh son usadas para modular, o para representar grupos de 6 símbolos. La razón para la modulación ortogonal de los símbolos es debido a la naturaleza no coherente del enlace Inverso. Cuando la transmisión de un usuario es no coherente, el receptor aun tiene que detectar cada símbolo correctamente.

Por otro lado, si un grupo de 6 símbolos es representado por una función Walsh única, entonces la BS puede fácilmente detectar 6 símbolos a la vez decidiendo cual función Walsh es enviada durante ese periodo. El receptor puede fácilmente decidir cual función Walsh es enviada correlacionando la secuencia recibida con el conjunto de 64 funciones Walsh conocidas. Nótese que en el enlace Delantero, las funciones Walsh son usadas para distinguir entre diferentes canales, mientras que en el enlace Inverso, las funciones Walsh son usadas para distinguir entre los

diferentes símbolos (o entre grupos de seis símbolos). En la realidad, un grupo de seis símbolos binarios corresponde a un valor decimal entre 0 y 63. El patrón del grupo de 6 símbolos (y su correspondiente valor decimal) indica cual función Walsh (0 a 63) es usada para representar ese grupo de 6 símbolos. Por ejemplo, un grupo de seis símbolos (-1,+1,-1,+1,+1,-1) corresponde al valor binario 010110, o al valor decimal 22, así la función Walsh número 22 es la salida del modulador ortogonal. Los datos modulados ortogonalmente a 4.8 Ksps (modulación de símbolos) o a 307.2 Ksps (símbolos codificados) son luego ensanchados por la secuencia PN larga. La secuencia PN larga tiene una tasa de chip de 1.2288 Mcps, y el ancho de banda luego del ensanchamiento es 1.2288 Mcps. Cabe recordar que la secuencia PN larga es usada para distinguir el canal de Acceso de todos los otros canales que ocupan el enlace reversa. Los datos son además revueltos en los ramales I y Q por la secuencia PN corta (que también corre a 1.2288 Mcps).

Debido a que el enlace Inverso usa modulación OQPSK, los datos en el ramal Q son retrasados medio chip. El propósito de este retraso es asegurar que la envolvente de la señal QPSK no colapse a cero. Esta propiedad es importante porque el

amplificador de potencia de la MS es normalmente pequeño y limitado en rendimiento. Si podemos asegurar que la envolvente de la señal no llegue a cero, y además que se mantenga sobre cierto nivel entonces el amplificador se mantendrá en zona lineal en un rango dinámico pequeño.

1.8.2.2 EL CANAL DE TRÁFICO.

El canal de Tráfico del canal Inverso es usado para transmitir datos de usuario y voz; mensajes de señalización también son enviados

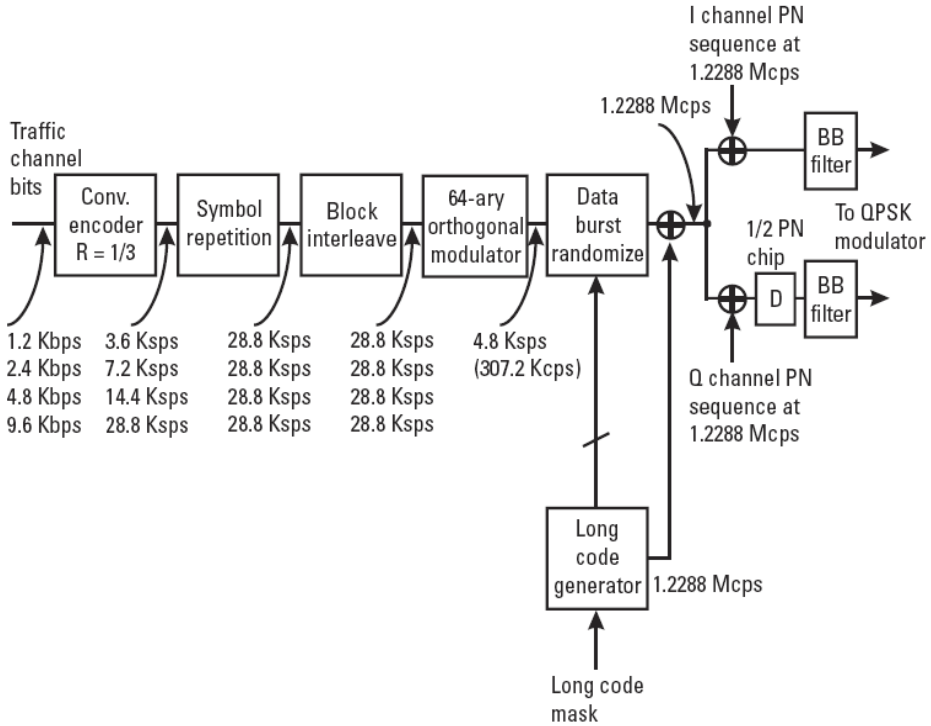


Figura 1.13 Estructura de canal de Tráfico.

sobre este canal. La estructura del canal de Tráfico Inverso es similar a la estructura del canal de Acceso. Como vemos en la figura 1.13 la principal diferencia radica en que el canal de Tráfico Inverso contiene un aleatorizador de datos.

Los datos modulados ortogonalmente son ingresados en este aleatorizador de datos, el cual toma ventaja del factor de actividad de voz en el enlace Inverso. Nótese que el enlace Delantero usa un esquema diferente para tomar ventaja del factor de actividad de voz cuando el Vocoder opera a bajas tasas de datos, el enlace Delantero transmite símbolos repetidos a una baja energía por símbolo, reduciendo así la potencia del enlace Delantero durante un periodo dado.

El esquema descrito para el enlace Delantero es inadecuado para el enlace Inverso, ya que el requerimiento de velocidad para el control de potencia es mucho mas exigente en el enlace Inverso. La BS mide el Eb/No en el enlace Inverso, entonces, la BS inmediatamente toma la decisión de control de potencia y envía un Bit de Control de Potencia de regreso al móvil. La BS necesita rápidamente detectar cada símbolo, aunque el Vocoder esta

operando a bajas tasas de datos. La figura 1.14 muestra el canal de Tráfico Inverso para tasa de datos 2.

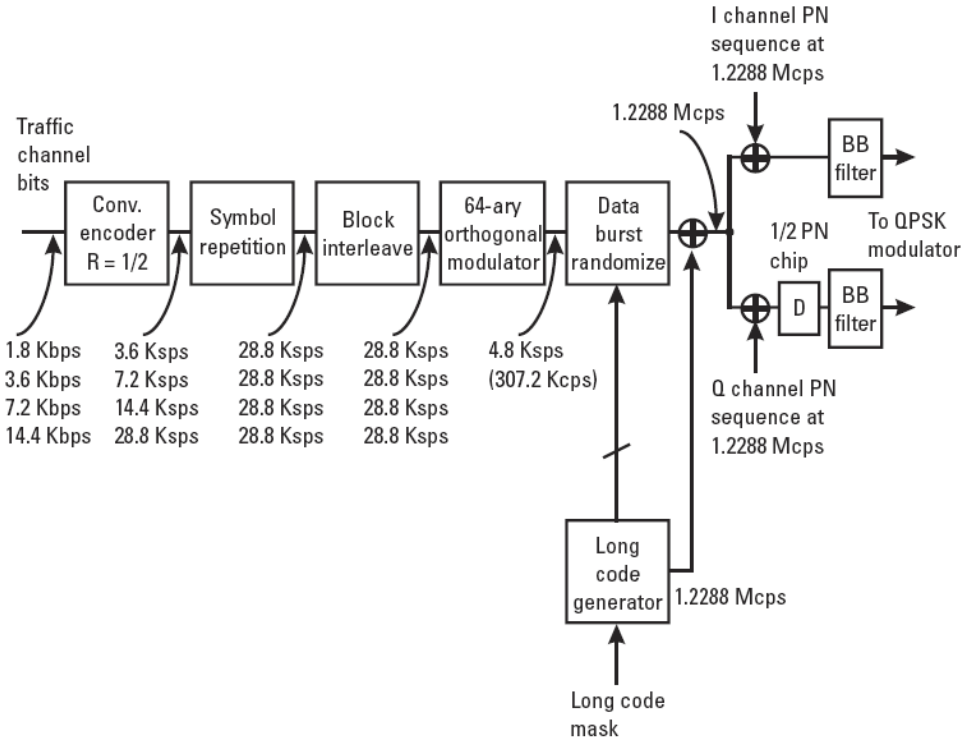


Figura 1.14 Estructura de canal de Tráfico Tasa de datos 2.

1.8.2.3 FORMATOS DEL CANAL DE TRÁFICO.

Para los enlaces Delantero e Inverso, las tramas del canal de Tráfico tienen una duración de 20 ms. La trama a máxima tasa de datos contiene 192 bits, corriendo a 9.6 Kbps; la trama a media tasa de datos contiene 96 bits, corriendo a 4.8 Kbps; la trama a un cuarto de la tasa de datos contiene 48 bits, corriendo a 2.4 Kbps; y la trama a un octavo de la tasa de datos contiene 24 bits,

corriendo a 1.2 Kbps. La trama a tasa máxima y tasa media contienen bits de indicador de calidad de trama (CRC), y todas las tramas contienen bits de cola (8 bits de cola por trama).

Antes se menciono que voz, datos e información de mensajería puede ser transmitida sobre el canal de Tráfico. De hecho, el sistema tiene la habilidad de multiplexar datos primarios y de señalización (o secundarios) en el mismo canal de Tráfico.

Cuando la MS se esta comunicando con la BS en el canal de Tráfico, la BS puede elegir enviar mensajes al móvil mientras el canal de Tráfico esta todavía activo. Durante la operación del canal de Tráfico, la BS envía mensajes de señalización al móvil usando el canal de Tráfico Delantero, la BS puede usar uno o mas canales de Tráfico Delantero para enviar un mensaje en particular.

CAPITULO 2.

INTRODUCCIÓN A PROCESAMIENTO DIGITAL DE SEÑALES.

El mundo contemporáneo esta representado por el cambio constante de los diversos procesos, es por eso que cada día se inventan maquinas sofisticadas, que tratan en lo posible de optimizar tiempo, espacio, costo, consumo de energía. Cada día vemos en el mercado muchos procesadores de distintas marcas que tratan de realizar un proceso en el menor tiempo posible, un ejemplo a esto son los procesadores Intel, AMD, Motorola, Texas Instruments, arquitectura que varía de acuerdo a las aplicaciones y necesidades.

Las técnicas de procesamiento digital, ahora no son tan complicadas como ocurría años atrás, su diseño no les permitía involucrarse en aplicaciones que solamente eran destinadas para sistemas analógicos,

hoy en día podemos realizar esto en el dominio digital, puesto que realizan funciones similares o mejores que los analógicos. Al fabricar maquinas en serie, hacen que el precio de los productos sea inferior y al alcance de todo público. Un ejemplo tangible a esta realidad son los teléfonos celulares de varias aplicaciones a precios módicos.

Con esta pequeña introducción podemos dar fe a la convergencia del mundo tecnológico, las mismas que son la microelectrónica en el dominio digital.

2.1 ELEMENTOS BÁSICOS DE UN SISTEMA DE PROCESAMIENTO DIGITAL.

Todo sistema electrónico por más simple o complicado que este sea, responde a un proceso básico como se muestra en la figura 2.1:

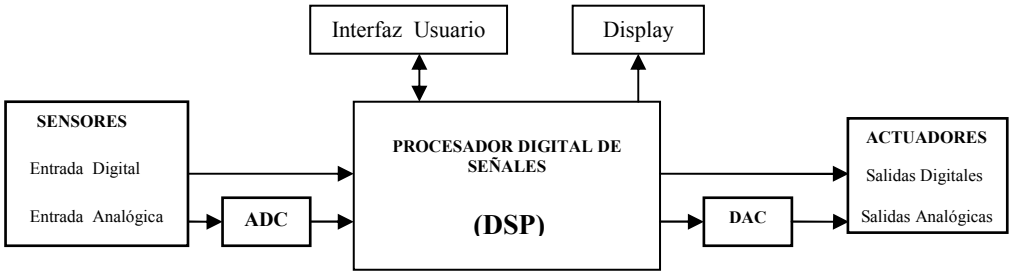


Figura 2.1 Diagrama de Bloques de un sistema típico.

Al realizar un proceso en el dominio digital, ya no tenemos los problemas encontrados en los sistemas analógicos, puesto que estos ya no muestran tolerancias asociadas a temperatura, vibraciones mecánicas, variaciones de voltaje que afectan dramáticamente el desenvolvimiento de dichos dispositivos.

Con la utilización del DSP se superan todos estos obstáculos y además se brinda:

- Aplicaciones Variables.
- Aplicaciones Eficientes.

Adicional a esto el DSP reduce:

- Sensibilidad al ruido.
- El numero de Chips en la Aplicación.
- Tiempo de ejecución.
- Costo.
- Poder de consumo.

Procesamiento Digital de Señal (DSP) es una operación o transformación de una señal en un hardware digital según reglas bien definidas las cuales son introducidas al hardware a través de un software específico que puede o no manejar lenguajes tanto de alto

como de bajo nivel. En la práctica se refiere al procesamiento electrónico de señales tales como sonido, radio y microondas usando técnicas matemáticas para realizar transformaciones o extraer información. En la práctica las características que hacen a los DSP tan buenos en el manejo de señales los hacen adecuados para muchos otros propósitos, tales como procesamiento de gráficos de alta calidad y simulaciones en ingeniería. Un DSP es un microprocesador, increíblemente rápido y poderoso que procesa señales en tiempo real. La definición de Real Time dependerá de la aplicación que se este implementando. Un ejemplo de esto es un Filtro FIR al que necesitamos analizar 100 TAP, el diseño es considerado en tiempo real si el DSP puede completar dicha operación entre 2 muestreos:

$$y(n) = \sum_{k=0}^{99} a(k)x(n-k) \tag{5}$$

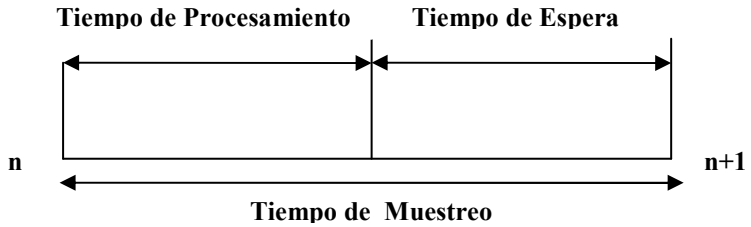


Figura 2.2 Tiempo de Muestreo/Espera/Procesamiento.

(5) Representación de una convoluciòn en tiempo discreto.

Decimos que una aplicación esta en tiempo real si Tiempo de Espera ≥ 0. Esta capacidad de procesamiento en tiempo real hace a los DSP ideales para aplicaciones que no toleran ningún retardo. Un ejemplo es la dificultad que tenemos cuando tratamos hablar por celular y existe retardo en la línea. Esto lleva a que la señal se corte o a confusión, ya que ambos usuarios hablan a la vez. Con los teléfonos celulares actuales, los cuales usan DSP, es posible hablar normalmente. El DSP dentro del teléfono procesa sonido (convirtiendo la señal analógica a digital, filtrando, comprimiendo y realizando otras tareas en forma digital) tan rápidamente que uno puede hablar y escuchar sin problemas de retardo; por tanto se procesa en tiempo real.

2.1.1 ACONDICIONAMIENTO DE LA SEÑAL DE ENTRADA.

Antes de realizar el procesamiento de cualquier señal necesitamos adquirir dicha señal, esto lo realizamos con los distintos dispositivos que existen en el mercado como son los sensores, sean estos de presión, audio, lumínica, temperatura, vibración, etc. En general los sensores generan señales eléctricas analógicas en respuesta a los distintos fenómenos físicos que ocurren en la naturaleza, posterior a esto necesitamos amplificar dicha señal para poderla interpretar, para finalmente filtrarla debido a que pueden presentarse frecuencias no deseadas inmersas en nuestro sistema.

2.1.2 CONVERSIÓN ANALÓGICA-DIGITAL A/D.

Las aplicaciones comunes de los DSP son en tiempo real, tales como sonido y ondas de radio que se originan en forma análoga. Una señal análoga tiene la característica de ser continua en el tiempo. Los computadores digitales, manejan señales de forma discontinua, una serie de números binarios, esta misma característica poseen los DSP. Para lo cual es necesaria la transformación de las señales análogas en digitales.

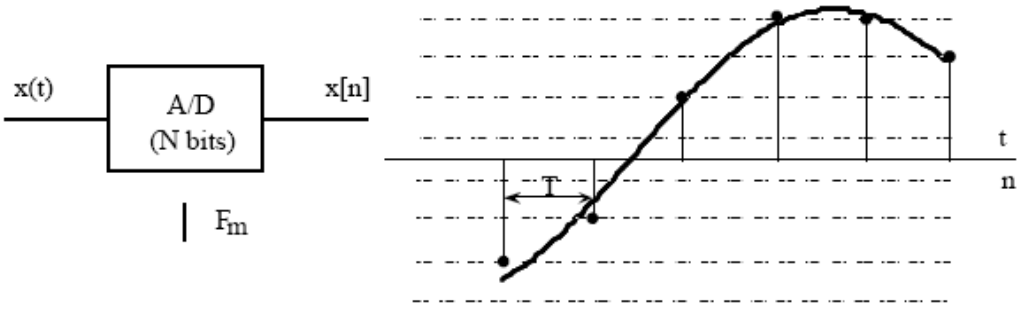


Figura 2.3 Ilustración de la conversión ADC.

Esta transformación la realizan los Convertidores Analógicos Digitales (ADC). Una vez terminada la etapa de conversión analógica-digital, los datos son entregados al DSP, el mismo que dará tratamiento a la señal. Nótese que la limitante para que un

procesamiento digital ocurra de forma rápida, es la velocidad de conversión del ADC.

2.1.3 CONVERSIÓN DIGITAL-ANALÓGICA.

El convertidor Digital-Analógico es un dispositivo electrónico que, a partir de una secuencia digital genera una señal analógica. Conceptualmente la conversión DAC se muestra en la figura 2.4.

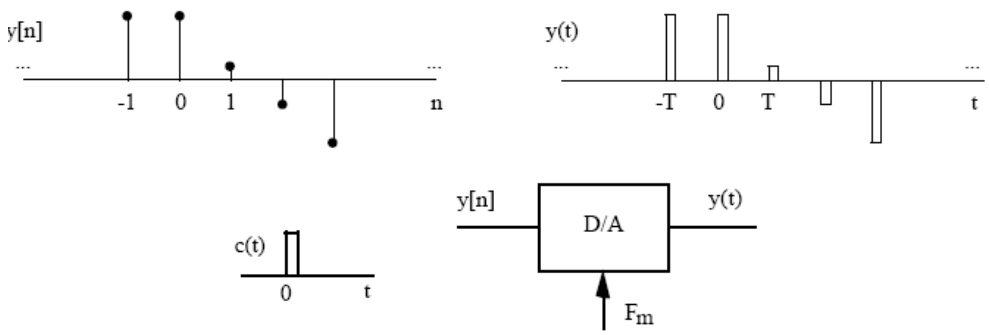


Figura 2.4 Ilustración de la conversión DAC.

Al utilizar una señal analógica c(t) de conversión, por cada muestra de la secuencia y[n] se produce a la salida, a intervalos de T segundos, una réplica de la señal c (t) con amplitud proporcional al valor de la muestra. Es decir, se genera una señal y (t) que responde al pulso que ingreso.

2.2 PROCESADOR DIGITAL DE SEÑALES.

En su núcleo un DSP es altamente numérico y repetitivo. A la vez que cada dato que llega debe ser multiplicado, sumado y además de eso transformado de acuerdo a fórmulas complejas. Lo que permite realizar todo ello es la velocidad del dispositivo. Los sistemas basados en DSP deben trabajar en tiempo real, capturando y procesando información a la vez que ocurre. Los conversores análogo-digital deben adquirir la información suficiente, para captar todas las fluctuaciones relevantes de las señales. Si el ADC es muy lento se perderá información. El DSP también debe trabajar rápido para no perder información que le llega desde el ADC, además de cumplir con el adecuado procesamiento de las señales.

Por ejemplo, un sistema estéreo maneja sonidos de hasta 20 KHz., por lo tanto el DSP deberá ser capaz de procesar alrededor del centenar de millones de operaciones por segundo. Otras señales, tales como transmisiones por satélite son del orden de los GHz por lo que requieren un procesamiento de mayor velocidad.

Dentro de las aplicaciones de interés con los DSP tenemos:

- Eliminar eco en líneas de comunicación.

- Hacer imágenes mas claras en órganos internos en los equipos de diagnóstico médico.
- Cifrar conversaciones en teléfonos celulares para mantener privacidad.
- Analizar datos sísmicos para encontrar nuevas reservas de petróleo.

El DSP debe ser capaz de manejar los números generados tanto en la transformación analógica-digital como durante los cálculos (multiplicaciones, sumas, divisiones) de dicha señal. Si no es capaz de manejar todo el rango de números ocurrirá una sobrecarga, lo cual producirá errores en los cálculos. La capacidad del procesador es una función de su ancho de datos (el numero de bits manipulados) y el tipo de aritmética que posee (punto fijo o flotante). Un procesador de 32 bits tiene un ancho de datos mayor que uno de 24 bits, el cual a su vez tiene un rango mayor que uno de 16 bits.

DSP de punto flotante tienen rangos mayores que uno de punto fijo. Cada tipo de procesador es ideal para un rango particular de aplicaciones. DSP de 16 bits son ideales para sistemas de voz tales como teléfonos ya que ellos trabajan con un estrecho rango de frecuencias de audio. Estéreos de alta fidelidad requieren ADC de 16 bits y un procesador de 24 bits de punto fijo. Los 16 bits del conversor

permiten capturar todo el rango de la señal de audio y los 24 bits del procesador permiten operar cómodamente los grandes valores resultantes de la operación con los datos. Procesamiento de imágenes, gráficos 3-D y simulaciones científicas necesitan un rango dinámico mucho mayor y por lo tanto requieren procesadores de punto flotante de 32 bits y ADC de 24 bits.

2.2.1 INTRODUCCIÓN AL DSP.

Un DSP es un Procesador de Señales Digitales, cuyo objetivo en cualquier sistema es de dar el tratamiento digital adecuado de las señales que ingresan a este dispositivo, dentro de sus características podemos citar:

- El procesamiento de las señales las realiza a muy altas frecuencias, con esto logramos que el DSP este orientado a reflejar un producto real time, gracias a sus muy bajos tiempos de retardos.
- Realiza multiprocesos es decir dar tratamiento simultaneo de las señales que ingresan a dicho dispositivo, con esto logramos satisfacer uno de los requerimientos en telecomunicaciones como es la multi-canalización.
- Pueden ser reprogramados con solo actualizar el programa que reside en la memoria del DSP, una aplicación que se le puede

dar a esta característica es la de actualizar tecnología, esto lo observamos en las BS que constantemente deben actualizar su tecnología

- Su diseño es óptimo, con repeticiones o lazos comunes en algoritmos de procesamiento de señal. El set de instrucciones de los DSP son pequeños y óptimos para operaciones de procesamiento digital con un simple ciclo realizamos multiplicaciones y acumulación.
- DSP tiene modos de direccionamiento indirecto y direccionamiento circular. Estos son mecanismos de direccionamiento eficientes para los algoritmos de procesamiento de señal.
- DSP posee periféricos apropiados que brindan interfaces eficientes de entrada y salida con otros dispositivos.
- En los DSP es posible acceder a memoria con una simple instrucción, en otras palabras estos dispositivos tienen un relativo ancho de banda entre su Unidad Central de Procesamiento (CPU) y memoria.

Los DSP se clasifican de acuerdo a la necesidad de cada usuario, así pues si utilizamos un DSP para una aplicación sencilla, estamos subutilizado el poder que nos proporciona dicho dispositivo,

incrementando así el coste del equipo y a su vez la difícil introducción del mismo para competir en el mercado. Nuestro estudio estará enfocado al DSP de Texas Instruments, por lo que se lo ha dividido de acuerdo a la aplicación:

C2000 (C20x/24x/28x). Este dispositivo es de muy bajo costo y puede ser utilizado en sistemas de Control como por ejemplo:

- Control de Motores.
- Almacenamiento.
- Control de Sistemas Digitales.

C5000 (C54x/55x). La característica de esta familia es estar provista de un procesador eficiente, pues proporciona una mejor MIPS (Mega Instrucción por Segundo) por Watt/Dólar/Medida, dentro de las aplicaciones podemos citar:

- Teléfonos Celulares.
- Reproductores de Audio.
- Cámaras Digitales.
- MODEM.
- Telefonía.
- VoIP.

C6000 (C62x/64x/67x). Este dispositivo fue diseñado para un máximo rendimiento, con una fácil programación (entorno de programación C, MATLAB, CODE COMPOSER), dentro de las aplicaciones podemos citar:

- Multi-canalización y Multi-función.
- Estaciones base, para servicios inalámbricos.
- Procesamiento de Imágenes.
- Servidores Multi-media.
- Video.

Nuestro estudio compete el área de las telecomunicaciones, razón por la que enfocaremos el análisis al procesador de la serie C6000.

2.2.2 PROCESADOR DIGITAL DE SEÑAL TMS320C6416.

Un nuevo miembro de la familia C6000, es el procesador C64x que utiliza aritmética punto fijo, tiene una velocidad de reloj que puede variar desde 600 a 1100 MHz la que puede incrementar a futuro un 83% el poder de procesamiento de señales.

El C64x es mostrado en la figura 2.5. El núcleo del C6416 consiste de 8 unidades funcionales, 2 archivos de registros, y dos rutas de

datos. Software Radio[5] generalmente usa datos de 12-16 bits, para diseño de equipos de tercera generación.

La extensión de 16-bits en la unidad funcional de multiplicación está también presente en las otras seis unidades funcionales. Esta incluye operaciones dobles de 16-bits de adición/sustracción, comparador, desplazamiento, máx. /min y de valor absoluto. Tipos de dato en paquetes de 8-bits y 16-bits son usados por las herramientas de generación de código para tomar ventaja de esta extensión.

Por duplicar los registros en el archivo de registros y la duplicación del ancho de la ruta del dato, como lo utilizado en la instrucción de empaquetamiento, el compilador del C6000 es mucho más eficiente por tener menos restricciones.

[5] Software Radio (o Software Defined Radio) es un concepto cuyo objetivo es poner dentro de un Software la mayor parte de las funcionalidades de un Transciver (Transmisor-Receptor).

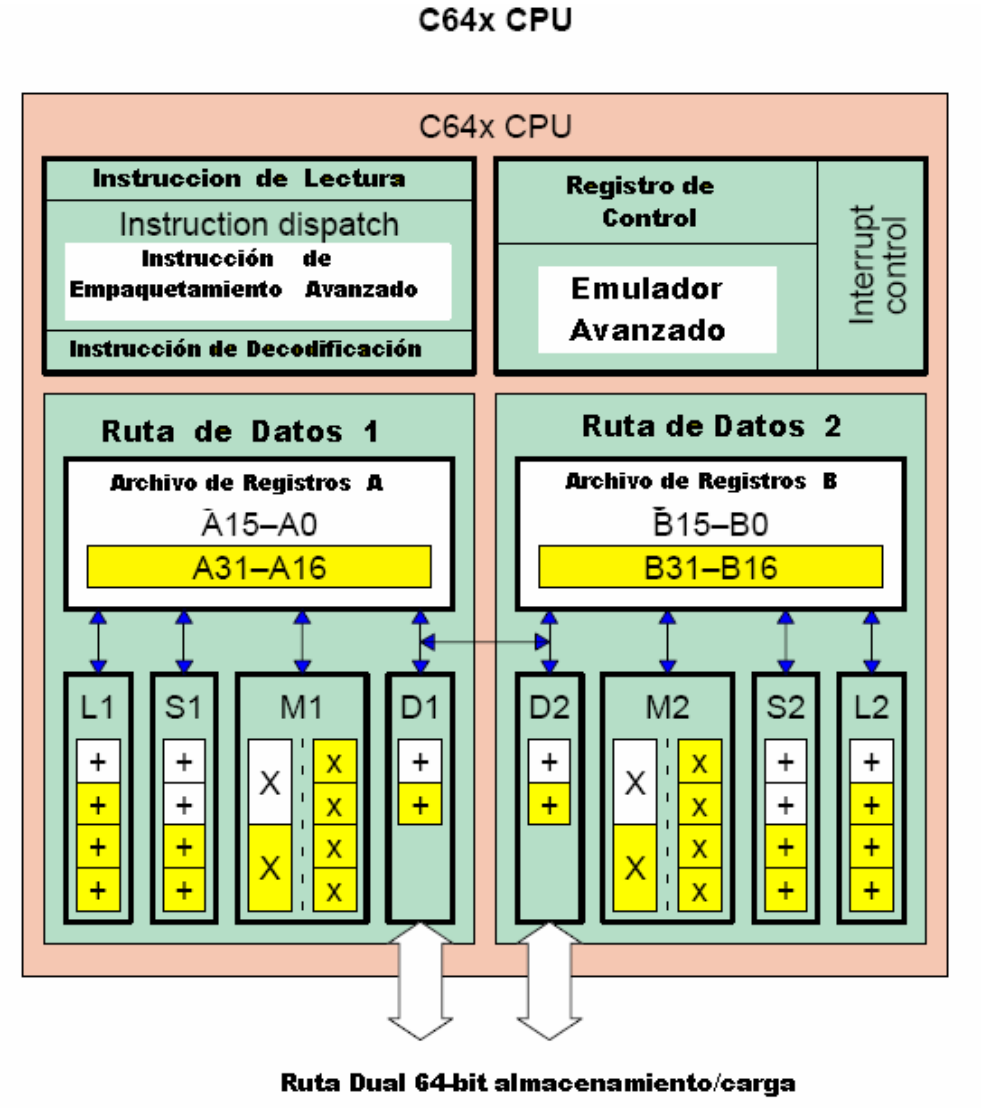


Figura 2.5 TMS320C64X diagrama de bloques del núcleo.

La serie C64X tiene tres dispositivos:

- **TMS320C6414** (propósito general).

  64 canales DMA (Acceso Directo a Memoria), 2 EMI (Interface de Memoria Externa), 3 puertos seriales, puertos host de 32-bit.

- **TMS320C C6415** (networking).

  PCI, interfaces ATM.

- **TMS320C C6416** (infraestructura Inalámbrica).

  Viterbi coprocesador, Turbo coprocesador, PCI, ATM.

Cada dispositivo usa el mismo núcleo e interfaces de memoria para cache nivel 1 y 2. Diferentes dispositivos pueden ser conectados al controlador de DMA. El C6414 es el nivel básico de los dispositivos, posee dos Interfaces de Memoria sofisticadas (EMIF), tres puertos seriales McBSP, un puerto de interface host de 32-bits (HPI), y una entrada/salida de propósito general de16-bit (GPIO). El C6415 está por sobre el C6414 por tener interfaces adicionales en el controlador DMA. Este incluye un Componente de Interconexión Periférica (PCI) o HPI plus GPIO y una UTOPIA 2 o una McBSP.

2.2.3 PROPIEDADES C64X.

Una propiedad del C6416, que lo hace ser utilizado en dispositivos de Tercera Generación de Comunicaciones, es poseer un Co-procesador Viterbi (VCP) y un Co-procesador Turbo (TCP). Estos al

ser programados cumplen los estándares expuestos para tecnología de 3G. El VCP puede soportar hasta quinientos canales de voz a 8 Kbps y el TCP tiene la capacidad de soportar hasta treinta y cinco canales de datos a 384 Kbps. Codificadores convolucionales Viterbi son usados en tráfico de voz para corregir en la transmisión errores y puede ser encontrado en sistemas de segunda y tercera generación.

La mente humana es susceptible a no percibir errores, que ocurren a grandes velocidades; lo que no ocurre con los datos, que demandan una calidad mucho más alta. Incluye más unidades de funciones que manejan datos de bits más anchos, el C6416 incluye cuatro unidades multiplicadoras de 16 x16 bit (cada unidad .M puede ejecutar dos multiplicaciones por ciclo), 64 registros de propósito general de 32 bits. Poseen mas de 1 MB de memoria interna la misma consiste de L2 RAM/cache, 16kB de cada unidad L1P programa cache y L1D datos cache, la mismo se detalla en los siguientes puntos.

2.2.4 ARQUITECTURA TMS320C6X.

El TMS320C64X tiene una Arquitectura VLIW (256 bits de ancho), para alimentar hasta ocho instrucciones de 32 bits para las ocho

unidades funcionales durante cada ciclo de reloj. Incluye dos niveles de memoria interna L1 y L2, el nivel 1 es un L1P Cache de 16 KB y L1D Cache de 16 KB, cada cache nivel 1 (L1P y L1D) está conectado a L2 Cache de 1 MB, unificado para programa y datos. La figura 2.6 muestra el diagrama de bloques del DSP.

Posee dos rutas de datos punto fijo, la ruta A y la ruta B. Cada ruta de datos contiene cuatro unidades de ejecución: ALU, un registro de desplazamiento, un multiplicador, un sumador/restador usado para generar direcciones. Cada ruta de datos también contiene un archivo de registros con 32 registros de propósito general de 32 bit, el doble de lo que tiene TMS320C62X.

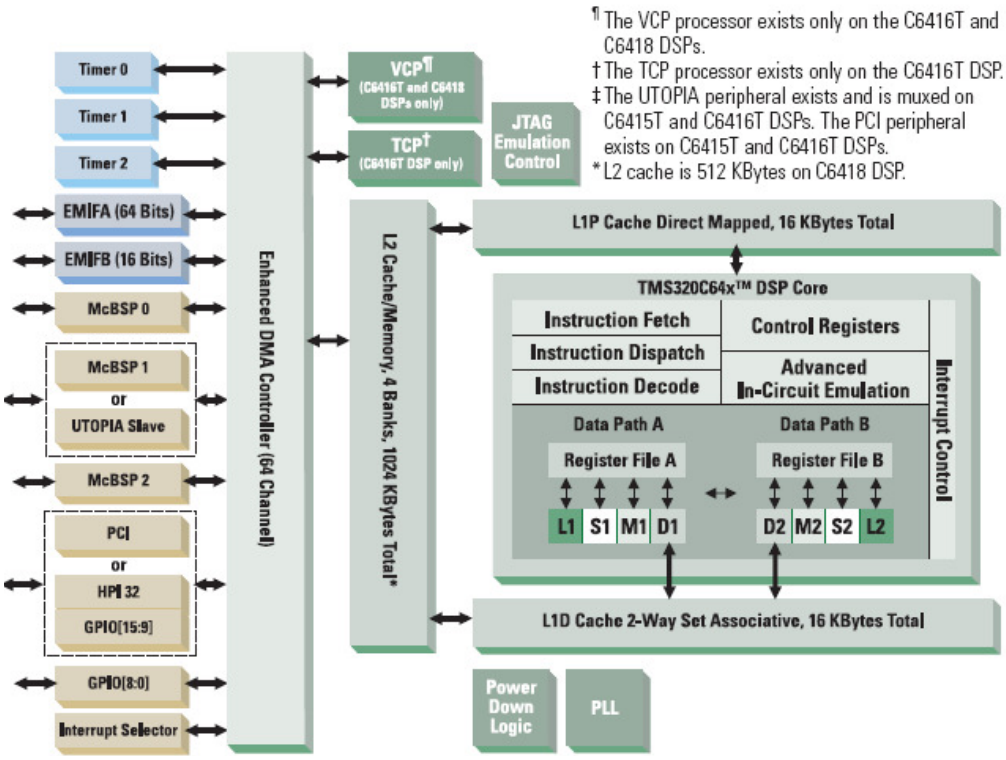


Figura 2.6 Diagrama de Bloques DSP.

Las ocho unidades de ejecución son capaces de ejecutar más de 8 instrucciones de 32 bits en paralelo utilizando los dos archivos de registro. El TMS320C64X puede operar con datos de 8, 16, 32 y 40 bits de largo y también puede operar con 64 bits (doble palabra), cuando se cargan o almacenan datos desde o hacia la memoria.

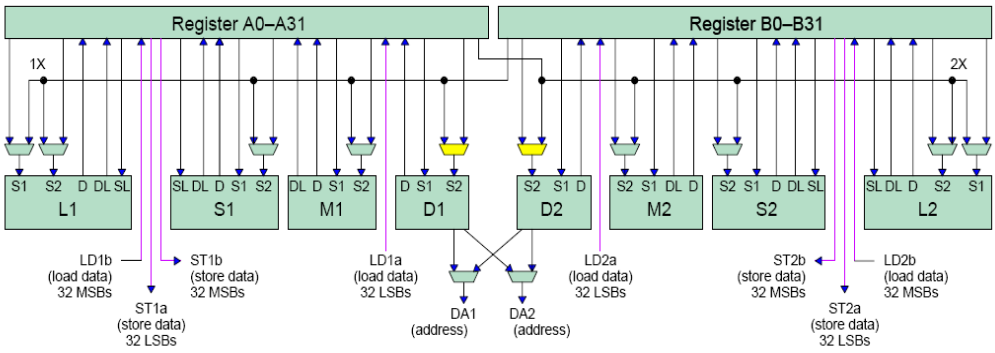


Figura 2.7 Ruta de Datos CPU TMS320C64X.

El TMS320C64X maneja datos de 64 bits usando un par de registros. Típicamente las unidades ALUs, registros de desplazamiento y la generación de direcciones opera con operando de 32 bits, pero el ALU y los registros de desplazamiento pueden operar con de 40 bits. Los multiplicadores realizan multiplicaciones de 16x16 bit, 8x8 bits. Comparando al TMS320C62X, el TMS320C64X agrega 4 de 8-bit y 2 de 16 bits instrucciones aritméticas SIMD y lógica para algunas de las unidades de ejecución, e introduce nueva instrucción producto punto, con vectores, 4 de 8 bits y 2 de 16 bits, para las dos unidades de multiplicación. Con este desarrollo el TMS320C64X puede realizar multiplicaciones paralelas cuatro de 16 bits u ocho de 8 bits.

2.2.5 BUSES INTERNOS.

Como se presenta en la figura 2.8 el bus interno consiste de un bus de direccionamiento de programa de 32-bit, un bus de datos de programa de 256-bit organizados en instrucciones de ocho de 32 bit, dos buses de direccionamiento de datos de 32-bit (DA1 y DA2), dos buses de carga de datos de 32 bit (64-bit para el C64x), y dos buses de almacenaje de datos de 32-bit (ST1 y ST2). Un bus de 32-bit DMA de datos y un bus 32-bit DMA de direcciones. El acceso de memoria es través de un bus de direcciones de 32-bit y un bus de datos de 32-bit.

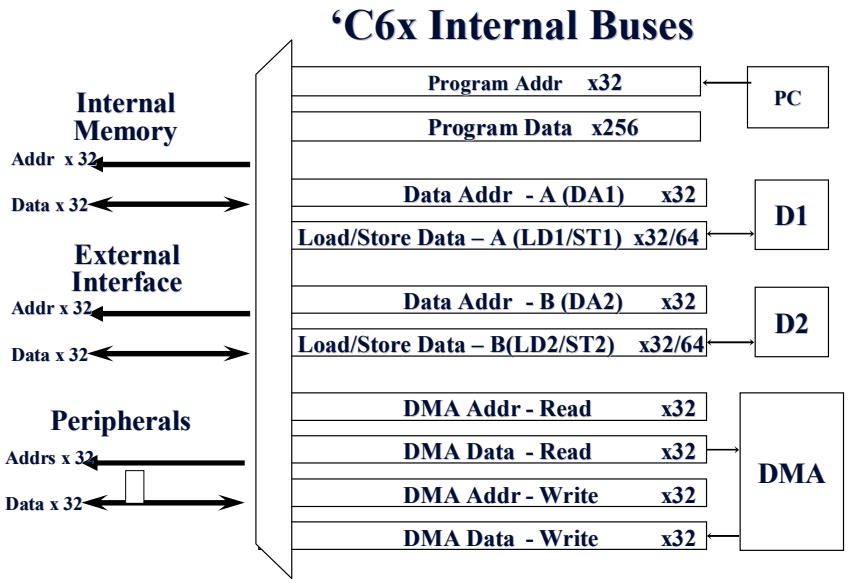


Figura 2.8 Buses Internos C64x.

2.2.6 PERIFÉRICOS.

Los periféricos en un procesador típico C6X incluye:

- Interface de Memoria Externa (EMIF).
- Acceso Directo a Memoria (DMA).
- Puertos Bus Seriales Multi-Canal (McBSP).
- Inicializador de Carga.
- Temporizador.
- Unidad Lógica de Apagado.
- PLL.

Adicional a esto los dispositivos de la serie C6416 incluyen:

- Interfaz de Memoria Externa (EMIF), este posee EMIFA y EMIFB.
- Decodificador Viterbi (VCP).
- Decodificador Turbo (TCP).
- Interface PHY de Operación y Análisis Universal (UTOPIA).
- Entradas y Salidas de Propósito General (GPIO).
- Interface para puerto Host (HPI).
- PCI.

EMIF (INTERFACE DE MEMORIA EXTERNA).

- Acceso a memoria asíncrona (SDRAM, SBSRAM, ZBT SRAM y FIFO)/síncrona (SDRAM y EPROM).

- Trabaja con PC100 SDRAM (barata, rápida y fácil)
- Acceso de datos.
- Ancho del bus de 16, 32, o 64-bit.

DISPOSITIVO	INTERNA	EMIF A	EMIF B
C6414 C6415 C6416	L1P = 16 KB	256 MB 64 Bits	256 MB 16 Bits
	L1D = 16 KB		
	L2 = 1 MB		

Tabla III Memoria Interna y Externa.

DMA (ACCESO DIRECTO A MEMORIA).

- Permite el movimiento de datos desde un lugar en la memoria a otro lugar sin que interfiera con la operación del CPU.

MCBSP (PUERTOS BUS SERIALES MULTI-CANAL).

- 3 Puertos seriales sincrónicos full dúplex (McBSP0, McBSP2 para C6000 y McBSP1 solo en C6416).
- Cada Canal hasta 256.
- Desempeño hasta 100 Mb/sec.
- Interfaces directas (T1, E1, MVIP, SCSA).

- Interface Periférica Serial (SPI) compatible a Motorola.

INICIALIZADOR DE CARGA.

- Inicia la carga de código desde memoria o HPI a la memoria interna.

TEMPORIZADOR/CONTADOR.

- Dos o Tres TEMPORIZADORES/contador de 32-bits.
- Pueden generar Interrupciones.
- Pines Entrada y Salida juntas.
- Entrada CLKIN
- Salida CLKOUT1 (Velocidad de salida del PLL, Velocidad de Instrucción (MIP)), CLKOUT2 (½ de la velocidad de CLKOUT1).

UNIDAD LÓGICA DE APAGADO.

- Usada para ahorrar energía cuando el CPU está inactivo.

PLL (CIRCUITO DE FASE CERRADA).

- Multiplicador (x1, x6, x12) de Clock externo
- Pin seleccionable.

SOLO EN EL PROCESADOR C6416:

TCP (COPROCESADOR TURBO).

- Soporta 35 canales de datos de 384 Kbps.
- Turbo codificador 3GPP/IS2000.
- Modo de parámetros Programables de Turbo Código y Decodificación.

VCP (COPROCESADOR VITERBI).

- Soporta más de 500 canales de voz cerca de 8 Kbps.
- Parámetros de código Programables.

UTOPIA (INTERFACE UNIVERSAL DE ANÁLISIS Y OPERACIÓN PARA ATM).

- Conexión ATM (Modo de Transferencia Asíncrono).
- Conectividad WAN (Red de Área Extensa) de 50MHz por dirección de 8 bits para transmisión y recepción.
- Usuario define el formato de la celda hasta 64 Bytes.

GPIO (ENTRADA/SALIDA DE PROPÓSITO GENERAL).

- El C64x provee 8 o 16 bits de propósito general bitwise I/O.
- Usado para observar o controlar la señal de un simple pin.

HPI (INTERFACE DE PUERTO HOST).

- Dedicado, solamente como esclavo, bus asincrono de 16/32 bits permite al Microprocesador del Host acceder a la memoria interna del C6000.

PCI (INTERCONEXIÓN DEL COMPONENTE PERIFERICO).

- Interface PCI de 33 MHz y estandar de 32-bits.
- Tres Registros de Direcciones de Bus PCI.
- Cuatro Interfaces Seriales EEPROM.
- Control del Programa DSP bajo respuesta a interrupción PCI.
- Interrupción DSP vía Ciclo I/O PCI.

CAPITULO 3.

SOFTWARE DE DESARROLLO.

Para la ejecución de nuestro proyecto se utilizaron varias herramientas de software de desarrollo, como también software de aplicación, las mismas que dada su potencia y fácil manejo nos facilitaron en gran medida la etapa de implementación. Dentro del software de desarrollo se hizo uso de paquetes muy potentes como lo son MATLAB 7 versión 15, Code Componer Studio versión 2.21. A continuación damos una pequeña introducción a las herramientas de desarrollo empleadas.

3.1 MATLAB.

Matlab es un lenguaje de muy alto rendimiento para cómputo técnico. Este software integra en un solo paquete varias herramientas entre las cuales podemos citar: cálculos, visualización y programación en un

ambiente fácil de usar, donde los problemas y soluciones son expresadas en una notación matemática familiar.

Matlab se puede usar típicamente en:

- Matemática y computación.
- Desarrollo de algoritmos.
- Adquisición de datos.
- Modelación, simulación y desarrollo de prototipos
- Análisis de datos, exploración y visualización.
- Gráficos científicos y de ingeniería.
- Desarrollo de aplicaciones, incluyendo una interfaz grafica de usuario.

Matlab es un sistema interactivo en el cual el elemento básico de dato es una matriz que no requiere ser dimensionado previo. Esto permite resolver varios problemas técnicos computacionales, especialmente aquellos que involucran el uso de matrices y vectores.

El nombre Matlab proviene de Matrix Laboratory. Matlab ha evolucionado a lo largo de los años con la ayuda de varios usuarios. En ambientes universitarios, el curso de Matlab se dicta como una introducción para cursos más avanzados de matemáticas, ingeniería y

ciencia. En la industria, Matlab es la herramienta escogida para la investigación de la alta productividad, desarrollo y análisis.

Matlab provee de una familia de aplicaciones específicas llamadas toolboxes, dichos toolboxes permiten aprender y aplicar tecnología especializada. Los toolboxes son una extensiva colección de funciones de Matlab (M-files) los cuales extienden el ambiente de Matlab para resolver problemas de clase particular. Algunas de las área en las cuales se emplean los toolboxes son: procesamiento de señales, sistemas de control, redes neurales, simulación, etc. El entorno de Matlab consiste de cinco partes principales.

Ambiente de desarrollo: Este es un set de herramientas que ayudan a utilizar las funciones y archivos de Matlab. Muchas de estas herramientas son utilizan GUI (Interface de Usuario Grafica).

Librería de funciones matemáticas: Esta es una colección de algoritmos computacionales, los cuales van desde funciones como suma, seno, coseno y aritmética complejas, hasta funciones mas sofisticadas como matriz inversa, funciones de Bessel, y transformadas de Fourier.

El lenguaje Matlab: Este es un lenguaje de alto nivel con instrucciones de control de flujo, estructuras de datos, y programación orientada a objetos.

Gráficos: Matlab tiene una extensiva facilidad para la visualización de vectores y matrices como gráficos, axial como facilidad de impresión de los mismos.

Aplicación de Interface de Programas de Matlab (API): Esta es una librería que permite escribir programas en lenguaje C y Fortran los cuales pueden interactuar con Matlab.

El uso de Matlab en este proyecto nos da la facilidad de interactuar de una manera rápida con el hardware, que para nuestro caso es el DSK C6416, ya que a media que uno se va familiarizando con el software descubre que el tiempo de desarrollo de cualquier tipo de aplicación decrece de manera considerable. Esto se debe en gran medida a que mientras se esta desarrollando el código de determinada aplicación, este a su vez se puede cargar a la tarjeta para ver su funcionamiento en tiempo real, permitiendo de esta manera aprender con un método de ensayo-error, además al ver cual es el comportamiento de nuestro

programa en tiempo real se puede realizar un ajuste preciso de los diversos parámetros que configuran la aplicación.

3.1.1 SIMULINK.

Simulink es un software que es utilizado para modelar, simular y analizar sistemas dinámicos. Simulink soporta sistemas que son lineales y no lineales, modelados en tiempo continuo, muestreo o un hibrido de ambos. Dichos sistemas pueden ser multitasa por ejemplo diferentes etapas pueden ser muestreadas a diferentes tasas.

Para la simulación, Simulink provee una interface gráfica para la construcción de los modelos como diagramas de bloques, para lo cual se usa operaciones de clic y arreste del mouse. Con esta interface, se pueden construir modelos como si fuera con un lápiz y papel. Simulink ofrece una extensiva librería de componentes lineales y no lineales, conectores, también se pueden personalizar y crear bloques propios de acuerdo a la función que se requiera por parte del programador.

Luego de definir el modelo, se puede proceder a simularlo, y los resultados de la simulación pueden ser colocados en workspace de matlab para su posterior visualización y procesamiento. Luego de

este proceso, Simulink se enlaza con el Real Time Workshop, que es otra herramienta que ayuda a que MATLAB se comunique con CCS y posteriormente con el DSP.

3.1.1.1 LIBRERIAS DE SIMULINK.

La principal característica de Simulink es la de condensar una gran cantidad de librerías en las llamadas toolboxes, las cuales a su vez despliegan los diferentes bloques que se utilizan, a continuación se enumeran algunas de las librerías más utilizadas:

SET DE BLOQUES	APLICACIÓN
Para CDMA	Diseñar y simular plataformas CDMA
Para Comunicaciones	Diseñar y simular sistemas de Comunicación
Para Sistemas de Control	Diseñar y simular Sistemas de Control
Para Diseño de Filtros	Analizar y diseñar varios tipos de filtros
Para Procesamiento de señales	Diseño de sistemas de procesamiento de señal

Tabla IV Aplicación de Conjunto de bloques.

3.1.1.2 CONFIGURACION DE PARAMETROS INICIALES.

El diseño del modelo de cualquier sistema requiere de un ajuste de valores de las variables que intervienen, en nuestro caso en particular se requiere configurar parámetros tales como:

- Plataforma de hardware.
- Puertos de comunicación PC – DSK.
- Niveles de optimización de código fuente.
- Documentación interna de codificación.
- Tiempo de simulación del modelo.
- Ordenamiento de bits.

Estos parámetros se utilizaran como base para la simulación, compilación, generación de código e implementación el en DSP.

3.1.1.3 REAL TIME WORKSHOP.

Real time Workshop, es una extensión de las capacidades de Simulink y Matlab, el cual permite la rápida realización y prototipo de aplicaciones de tiempo real de una variedad de sistemas. Real time Workshop junto con las otros componentes de MathWorks, proveen:

- Generación automática de código el cual puede ser tolerado por una gran variedad de plataformas.

- Una rápida y directa ruta desde el diseño del sistema hacia la implementación en el hardware.
- Una GUI sencilla y amigable para el diseñador.
- Una arquitectura abierta.

COMPONENTES Y CARACTERÍSTICAS:

Los principales componentes y características de Real-Time Workshop son:

- **Simulink generador de código**: Genera automáticamente el código en C desde el modelo de bloques que se ha diseñado en Simulink
- **Simulink modo externo**: permite la comunicación entre Simulink y el modelo ejecutándose en un ambiente en tiempo real.
- **Apoyo de Hardware**: permite simular modelos de una variedad de ambientes, incluyendo Tornado y DOS.

La figura 3.1 adjunta ilustra el proceso completo. El recuadro sombreado muestra la porción del proceso ejecutado por Real-Time Workshop.

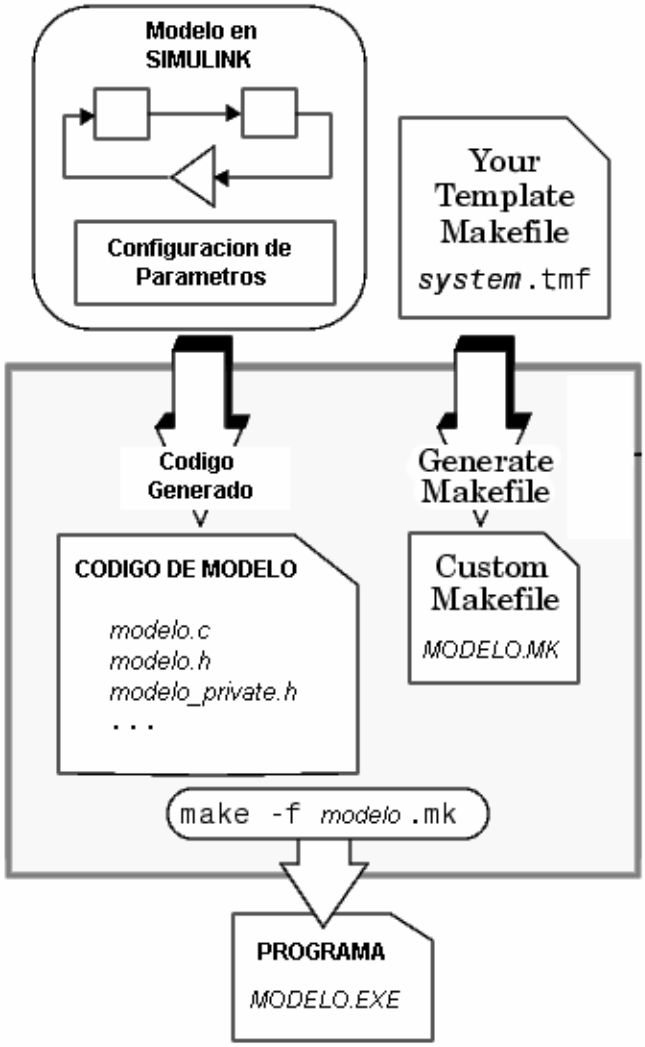


Figura 3.1 Proceso de Construcción en Simulink.

En nuestro caso Real-Time Workshop es de gran utilidad ya que nos ayuda a simplificar y acelerar algunas fases en lo que respecta a la etapa de desarrollo de líneas de código, para este

propósito de siguió una serie de procedimientos los cuales de detallan a continuación:

- Configurar parámetros de acuerdo a los requerimientos del proyecto.
- Ajustar parámetros de configuración.
- Simular y anticipar resultados.
- Realizar la generación de código.
- Obtener un programa ejecutable.
- Verificar los resultados obtenidos.
- Bajar programa a DSP.
- Ejecución del programa.

3.2 CODE COMPOSER STUDIO.

El Code Composer Studio versión 2.21 es una herramienta de Texas Instruments la misma que viene incluida dentro del KIT del DSK TMS320C6416.

El CCS crea un ambiente de trabajo que permite escribir, compilar, simular y realizar la depuración de los códigos que se crean. La figura 3.2 muestra las fases asociadas con el desarrollo del software CCS.

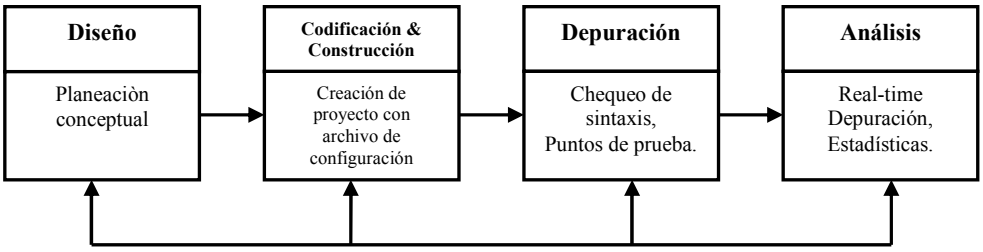


Figura 3.2 Proceso de Desarrollo del código en CCS.

El CCS es una interfaz estándar tipo Windows que posee menús, barras de herramientas que ayudan a construir, revisar y probar aplicaciones en tiempo real.

La figura 3.3 muestra una vista general del CCS 2.21. En ella se observa a la izquierda una ventana en la que se ha declarado el proyecto en el que actualmente se está trabajando y los diferentes módulos que éste posee.

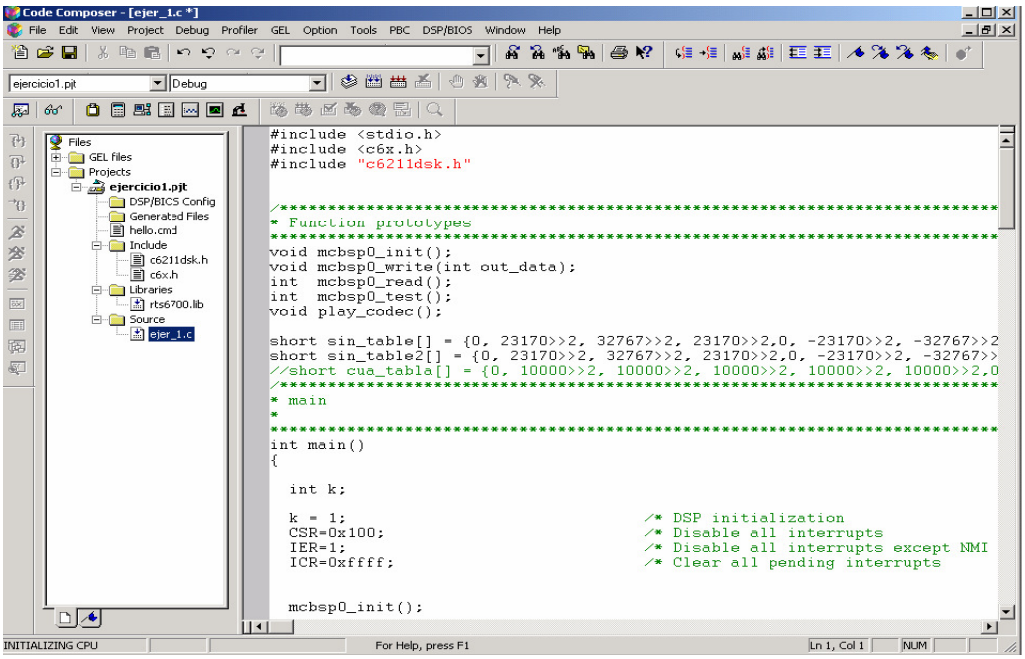


Figura 3.3 Entorno de CCS.

Los módulos están formados por el o los archivos en C, C++ o Assembler, librerías de funciones, archivos include y otros más específicos a la aplicación que se esté realizando.

La ventana principal puede mostrar el editor de texto que se utiliza para escribir el código, además puede mostrar gráficos de variables utilizadas en el DSP, un mapa de la memoria de programa y/o datos, etc.

3.2.1 COMPILACION DEL CODIGO FUENTE EN LA TARJETA.

Una vez que hemos diseñado el modelo en Simulink, hay que proceder a definir los parámetros de configuración específicos para el modelo y tipo de plataforma que se usa, nuestro caso es un TMS320C6416 DKS. El procedimiento a seguir es el siguiente:

CONFIGURACIÓN DE PARÁMETROS.

En la ventana del modelo en Simulink, dentro del menú hacer clic en *Simulation -> Configuration Parameters*, luego de lo cual aparecerá el cuadro de configuración que se muestra en la figura 3.4:

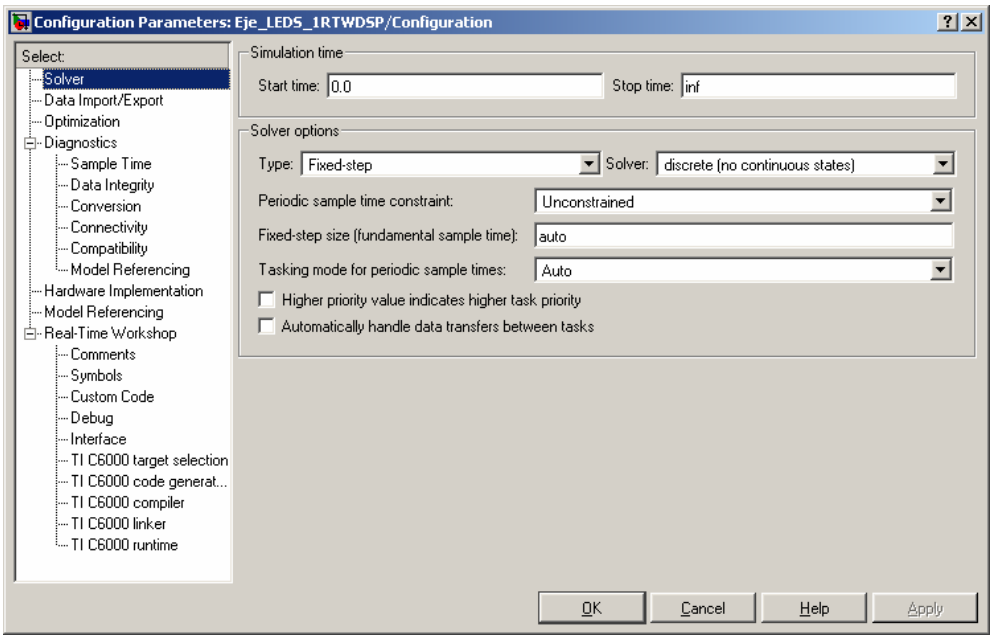


Figura 3.4 Ventana de Configuración de Parámetros.

Los parámetros para RESOLVEDOR quedaron de la sgte. manera:

PARAMETROS DE CONFIGURACION RESOLVEDOR.	
Tiempo Simular	Start time: **0.0**
	Stop time: **inf**
Opción de Resolvedor	Type: **Fixed Step**
	Solver: Discrete

Tabla V Parámetros de Configuración Resolvedor.

Las opciones de configuración siguientes: *Data Import/Export, Optimization, Diagnostics* se los deja con los valores por omisión. Luego de esto situarse en la opción IMPLEMENTACION DE HARDWARE y configurar como se muestra en la figura 3.5:

PARAMETROS DE CONFIGURACION IMPLEMENTACION DE HARDWARE.	
Embedded hardware	Device Type: **TI C6000**
	Byte Ordering: **Little Endian**

Tabla VI Parámetros de Configuración Implementación de Hardware.

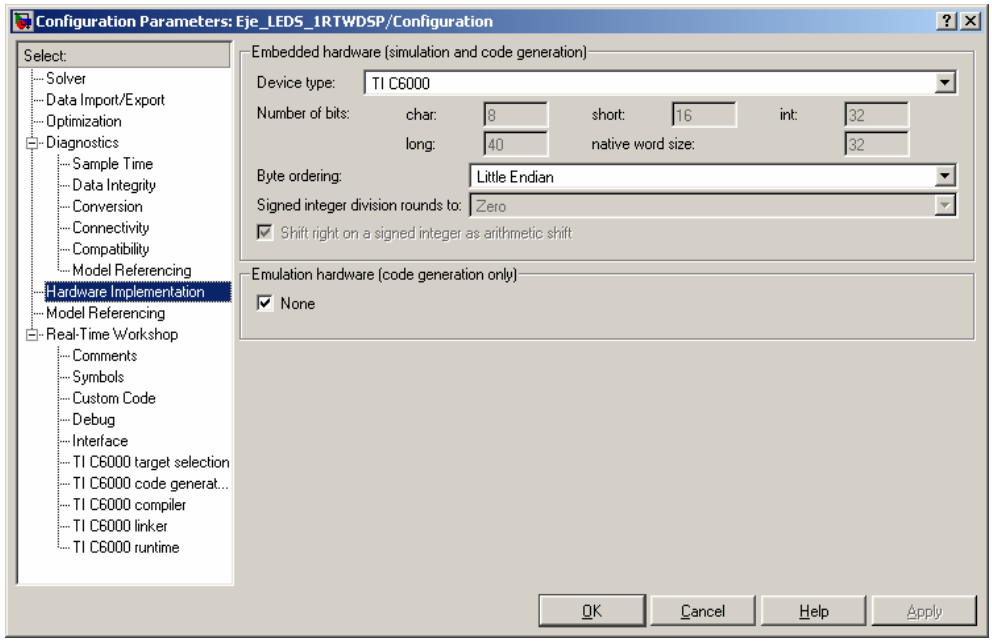


Figura 3.5 Ventana de Configuración Hardware Implementación.

Luego situarse en la opción de configuración REAL TIME WORKSHOP y configurar tal como se muestra en la figura 3.6:

PARAMETROS DE CONFIGURACION REAL TIME WORKSHP.	
Selección Plataforma	RTW system targetFile: **ti_c6000.tlc**

Tabla VII Parámetros de Configuración RTW.

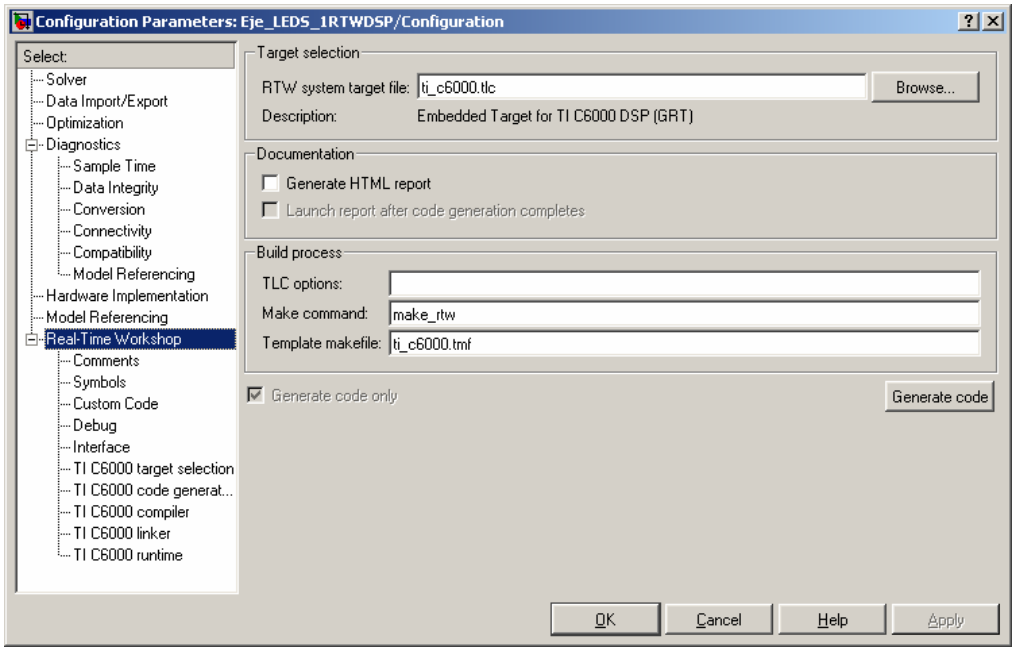


Figura 3.6 Ventana de Configuración Real Time Workshop.

Aquí dentro de la selección de plataformas tenemos varias alternativas que se seleccionaran de acuerdo al tipo de hardware que se este empleando en la implementación, la figura 3.7 muestra las diversas plataformas de hardware que están disponibles, para nuestro caso empleamos el hardware *ti_c6000.tlc*, ya que es la que se acopla con nuestro DSP TMS320C6416.

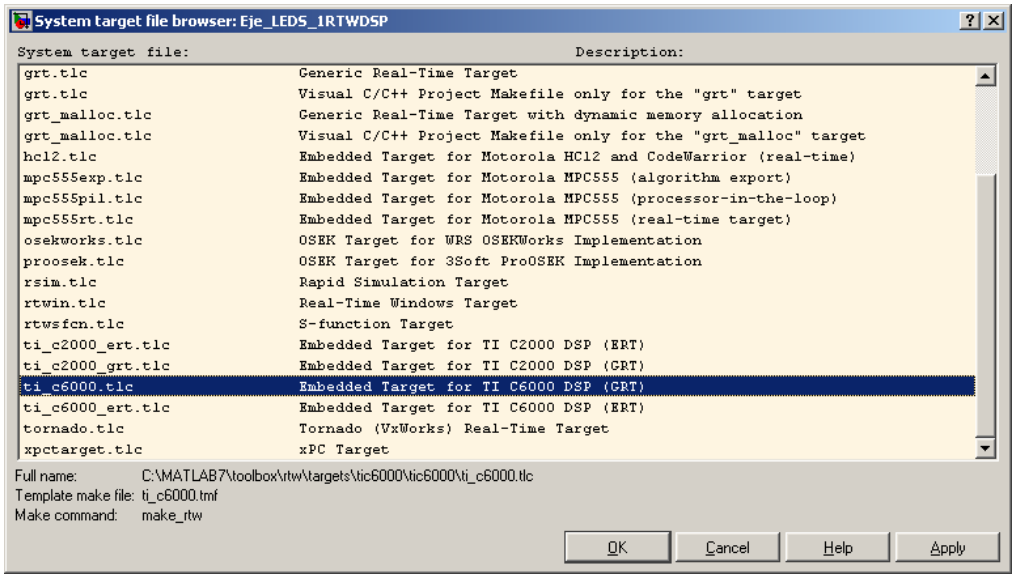


Figura 3.7 Cuadro de selección de Plataformas.

Luego nos ubicamos en la opción de configuración TI C6000 SELECCION PLATAFORMA, configuramos como la figura 3.8:

PARAMETROS DE CONFIGURACION TI C6000 SELECCIÓN PLATAFORMA.	
Tipo de Código Generado	**C6416 DSK**

Tabla VIII Configuración

TI C6000 SELECCION PLATAFORMA.

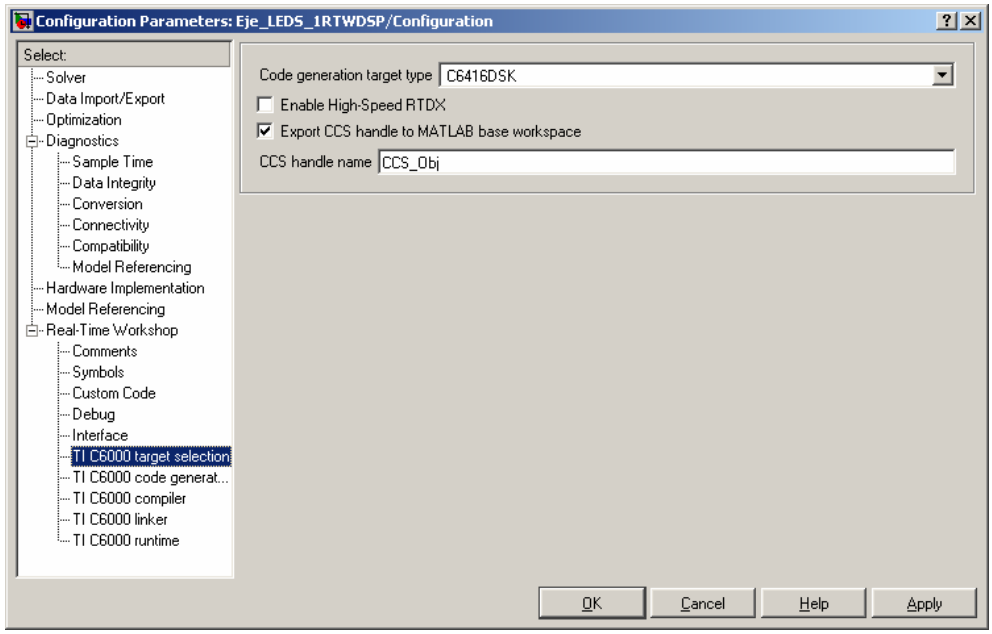


Figura 3.8 Ventana de Configuración TI C6000.

Por último nos ubicamos en la opción de configuración TIC6000 Generación de Código y seleccionamos de acuerdo a la figura 3.9:

PARAMETROS DE CONFIGURACION TI C6000 GENERACION CODIGO.	
Función de Procesamiento Digital En-línea.	**Activado**

Tabla IX Configuración

TI C6000 GENERACION CODIGO.

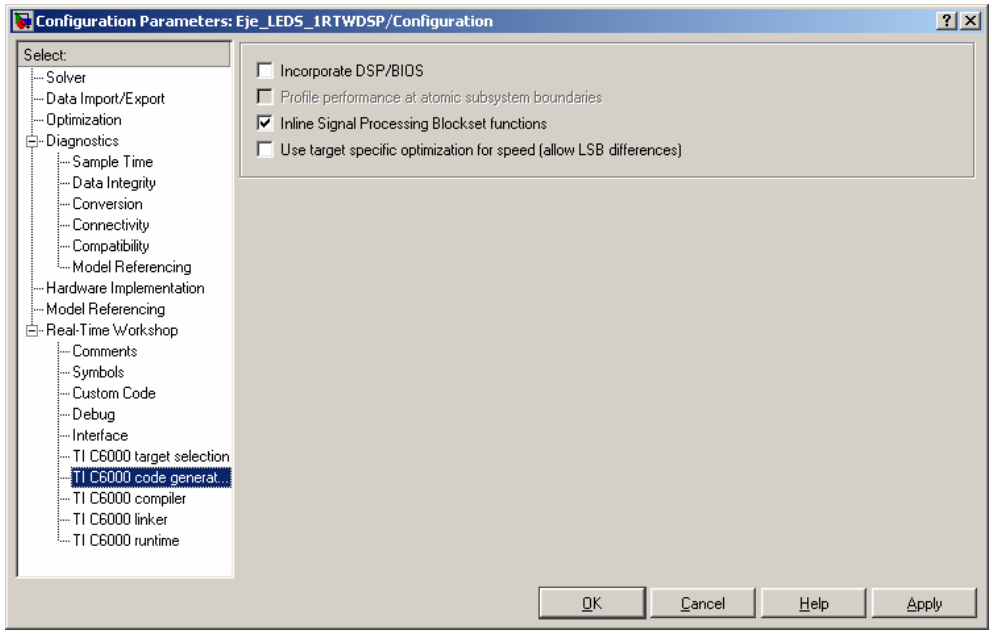


Figura 3.9 Ventana Configuración TI C6000 Generación de Código.

Una vez seleccionados los valores específicos para el modelo a implementar se deben aplicar los valores presionando *Apply* luego *Ok*. Luego en la ventana que nos muestra el modelo hacer clic en *Tools -> Real-Time Workshop -> Build Model*, con esto se inicia el proceso de compilación del modelo y en la ventana de MATLAB aparecen los archivos que genera dicha compilación. Al finalizar el proceso de compilación debe aparecer en el ambiente de programación de CCS nuestro modelo como activo, lo cual se comprueba porque el nombre de nuestro proyecto aparece en

negrillas y debe contener varios archivos generados los cuales serán:

nombre_del_archivo.c

*nombre_del_archivo*_data.c

*nombre_del_archivo*_main.c

MW_c6xxx_csl.c

rt_nonfinite.c

rt_sim.c

nombre_del_archivo.cmd

donde *nombre_del_archivo* es el nombre con que hemos guardado nuestro modelo.

3.2.2 CARGA Y EJECUCION DEL PROGRAMA OBJETO.

Una vez que se han concluido los procesos de desarrollo, simulación, ajuste y verificación del modelo en el ambiente de desarrollo de Simulink y además se ha ejecutado el procedimiento que realiza la compilación del modelo (este proceso se detalló en el punto 3.2.1), entonces es el momento de proceder con el punto final del proceso de desarrollo del modelo, el cual consiste en tomar el archivo que se generó luego de compilar el modelo y que debe estar

ahora activo en el entorno de desarrollo de CCS, es de aquí de donde se ejecuta el proceso de carga del archivo objeto en la plataforma que en nuestro caso es el DSK C6416.

A continuación se detalla como cargar y la ejecutar modelo:

a) Dar clic en el menú *Project -> Build*. Automáticamente se inicia el proceso de compilación del proyecto activo.
b) Luego (si no hay errores) seleccionar *File -> Load Program*. Con lo que se abre una ventana de donde se debe seleccionar el archivo generado *nombre_del_archivo*.out y clic en Abrir.

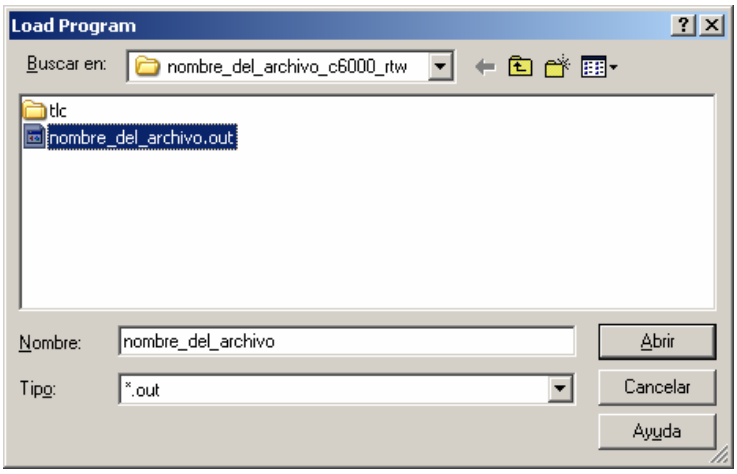


Figura 3.10 Cuadro de dialogo Cargar Programa.

c) Al presionar Abrir se inicia de forma automática el proceso mediante el cual el archivo compilado se baja al DSP.

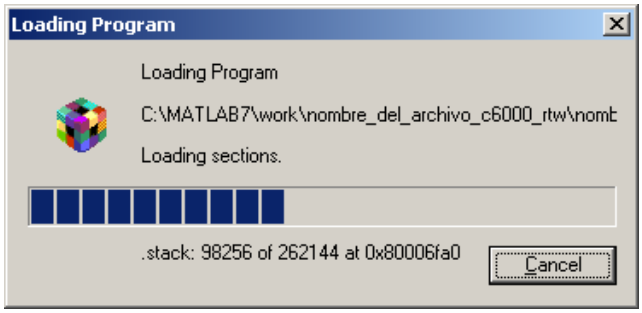


Figura 3.11 Indicador de Carga de Programa Objeto.

d) Finalmente el archivo objeto que esta físicamente en el DSP puede ser ejecutado dando clic en *Debug -> Run*.

CAPITULO 4.

ANALISIS DE LA ESTRUCTURA DEL CANAL PILOTO.

4.1 SISTEMA POR BLOQUES.

En este capitulo abordaremos la estructura del canal Piloto en bandabase. A continuación detallaremos las partes constitutivas del este canal que son las bases para el análisis de espectro ensanchado.

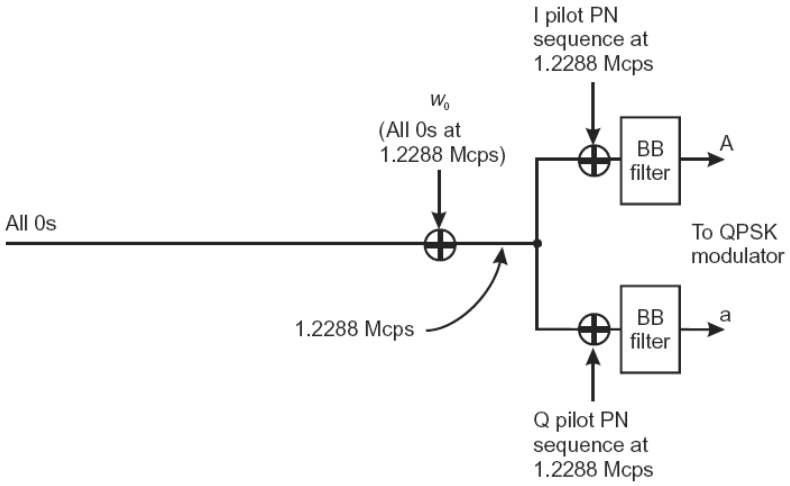


Figura 4.1 Diagrama de bloques del Canal Piloto.

4.1.1 BLOQUE WALSH.

4.1.1.1 FUNCIONAMIENTO.

El sistema CDMA utiliza los códigos Walsh de dos distintas formas dependiendo del tipo de enlace en el que se encuentre, en el enlace Delantero se usan estos códigos para realizar canalización y en el enlace Inverso se emplea para ensanchar. El bloque generador del código Walsh presenta en la salida el código correspondiente al orden del código Walsh requerido. Para nuestro proyecto es el código Walsh de orden 0 (Wo).

4.1.1.2 FUNCIÓN MATEMÁTICA.

Los códigos WALSH se generan a partir de una matriz llamada HADAMARD, en la cual cada una de sus filas forman un conjunto ortogonal de códigos. Las matrices Hadamard son matrices cuadradas en la que sus valores son +1 y -1. Entonces, si *N* es una potencia de 2 no negativa, la matriz Hadamard de *NxN*, detonada como H_N, es definida recursivamente como se muestra a continuación:

$$H_1 = 1$$

$$H_{2N} = \begin{bmatrix} H_N & H_N \\ H_N & -H_N \end{bmatrix}$$

Por ejemplo:

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

4.1.1.3 DIAGRAMA DIGITAL.

El diagrama digital del generador de código Walsh indica que se requiere los parámetros como son la longitud del código Walsh a generar, el índice del código que se desea representar.

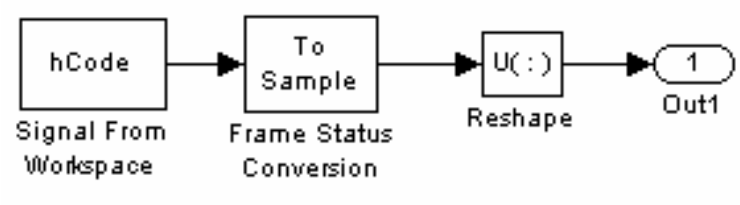


Figura 4.2 Diagrama digital Generador Walsh.

4.1.1.4 DIAGRAMA DE FLUJO DEL BLOQUE.

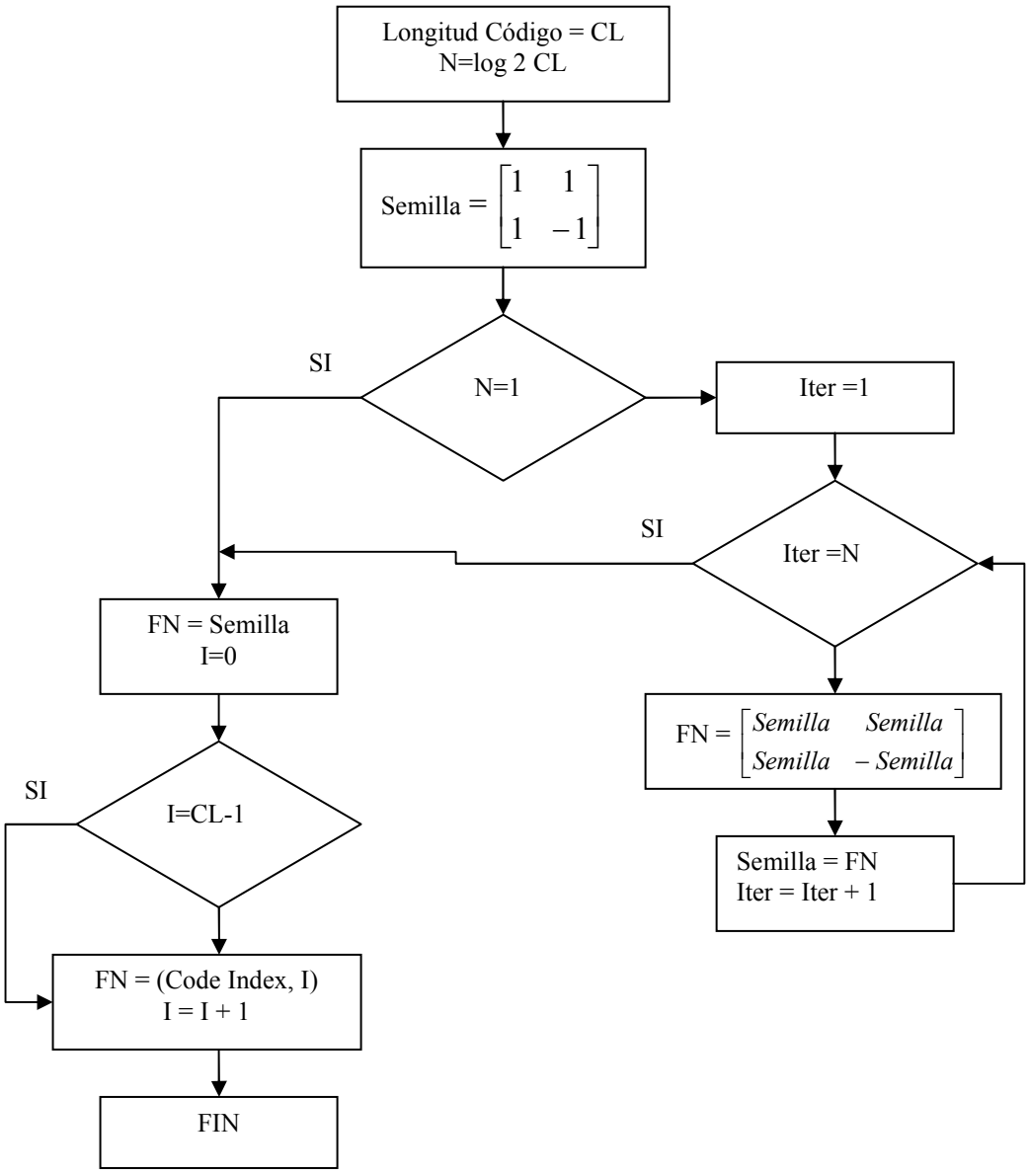


Figura 4.3 Diagrama de flujo bloque Walsh.

4.1.2 BLOQUE SECUENCIA PN.

4.1.2.1 FUNCIONAMIENTO.

En el sistema CDMA actualmente se usan dos versiones del generador de secuencia PN, uno de estos generadores es el que me da como salida un código llamado corto y el otro me da como salida un código llamado largo. La diferencia básica entre el código corto y el código largo es la función que cada uno desempeña en los canales Delantero e Inverso y además como su nombre lo indica el código PN corto tiene un periodo relativamente pequeño en comparación con el periodo del código largo:

Código corto: 2^{15} chips ≈ 0,02667 seg.

Código largo: $2^{41}-1$ chips ≈ 1789569,706 seg. ≈ 0,71 días

El bloque generador de código largo me da a la salida una versión decimada o completa de la secuencia, dicha secuencia se utiliza para ensanchamiento en el canal Inverso, mientras que la versión decimada de la secuencia larga es usada para hacer el scrambling en el enlace Delantero. El código largo es periódico, con periodo de $2^{41}-1$ chips (2199023255551 chips). Por otro lado el bloque generador de código corto genera las secuencias PN cortas I y Q que se utilizaran en el ensanchamiento de las señales CDMA.

4.1.2.2 FUNCIÓN MATEMÁTICA.

Luego del ensanchamiento ortogonal, cada canal es ensanchado en cuadratura (ver figura 4.1), la secuencia ensanchadora será periódica con una longitud de 2^{15} (32768 chips de longitud). Esta secuencia se denomina *"secuencia PN Piloto"* y se origina mediante los siguientes polinomios característicos:

$$PI(x) = x^{15} + x^{13} + x^{9} + x^{8} + x^{7} + x^{5} + 1 \quad (6)$$

para la secuencia IN-PHASE, y

$$PQ(x) = x^{15} + x^{12} + x^{11} + x^{10} + x^{6} + x^{5} + x^{4} + x^{3} + 1 \quad (7)$$

para la secuencia en CUADRATURA.

La tasa de chips del código generado será 1.2288Mcps. El período de la secuencia PN Piloto es 32768/1228800 = 26.666 ms, por lo que cada 2 segundos se repiten 75 veces cada secuencia PN Piloto.

---

(6) y (7) Estos polinomios característicos fueron tomados del estándar CDMA del 3GPP2, Physical Layer Standard for cdma2000 Spread Spectrum Systems, capitulo 3 pagina 109.

4.1.2.3 DIAGRAMA DIGITAL.

SECUENCIA PN I.

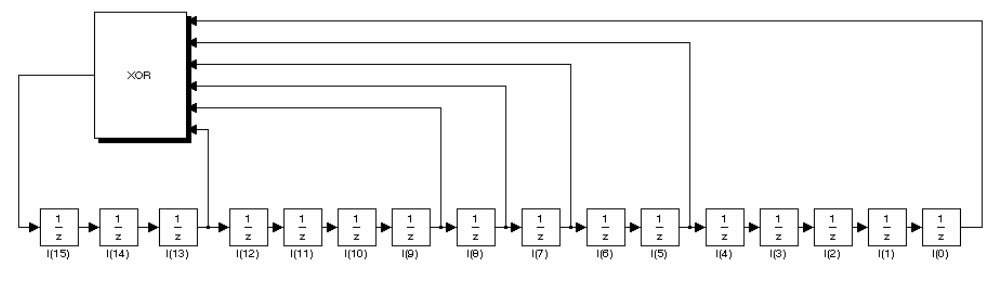


Figura 4.4 Diagrama de bloque Secuencia PN I.

SECUENCIA PN Q.

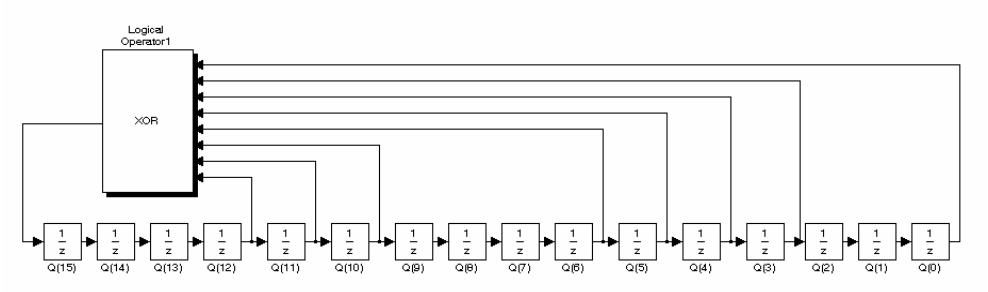


Figura 4.5 Diagrama de bloque Secuencia PN Q.

4.1.2.4 DIAGRAMA DE FLUJO DEL BLOQUE.

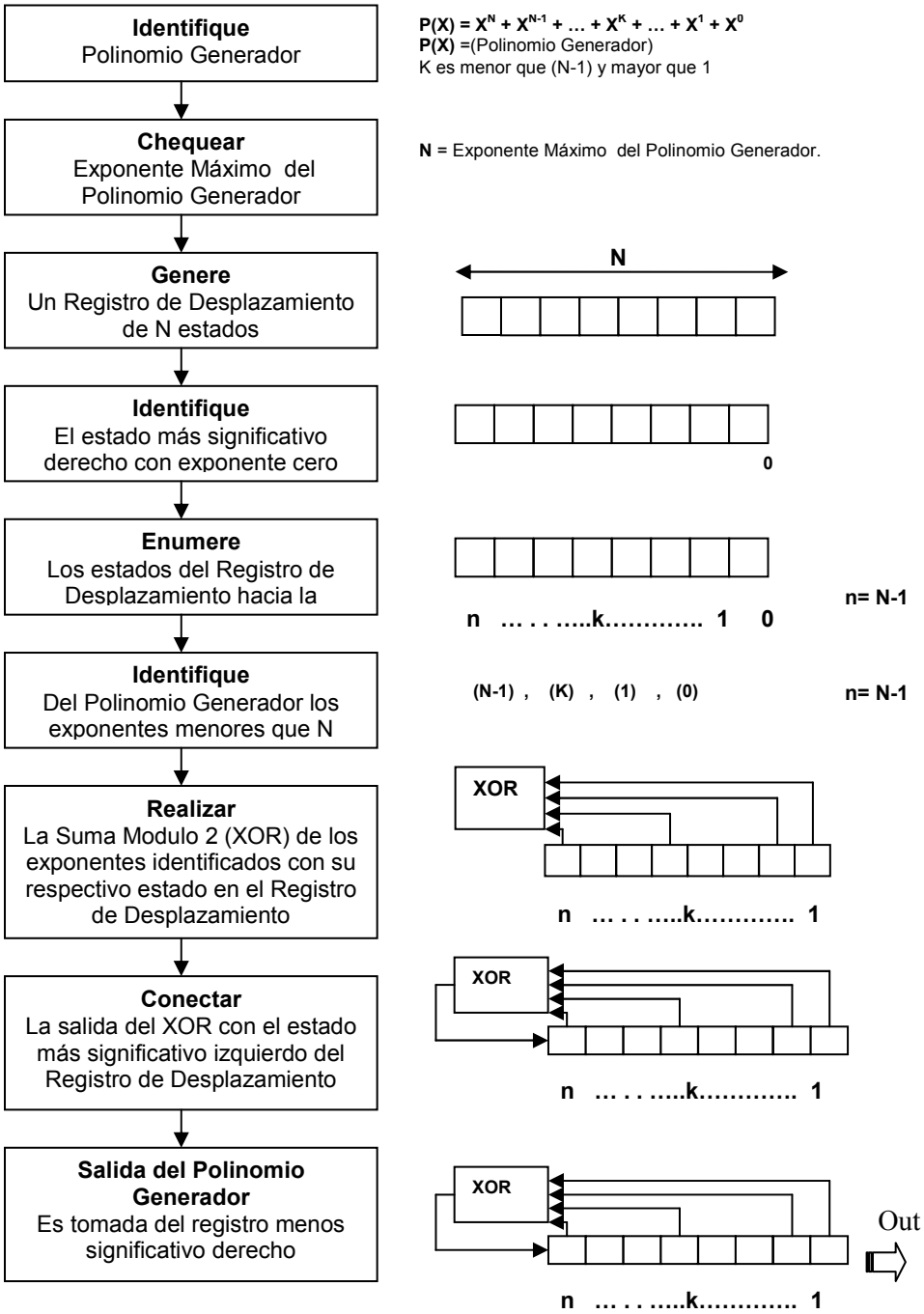


Figura 4.6 Diagrama de flujo Bloque Secuencia PN.

4.1.3 BLOQUE FILTRADO.

4.1.3.1 FUNCIONAMIENTO.

Después de la operación Ensanchamiento, los impulsos I y Q son aplicadas al ingreso del Filtro Banda Base. Los Filtros Banda Base tienen una respuesta de S (f) que satisface los límites dados en la figura 4.7. Específicamente, la respuesta de frecuencia normalizada del filtro estará contenida en +/- δ_1 en el pasa banda 0 ≤ f ≥ f_p y serán menor que o igual a δ_2 en la banda de parada f ≥ f_s. El valor numérico para los parámetros son δ_1 = 1.5 dB, δ_2 = 40 dB, f_p =590 KHz. y f_s = 740 KHz.

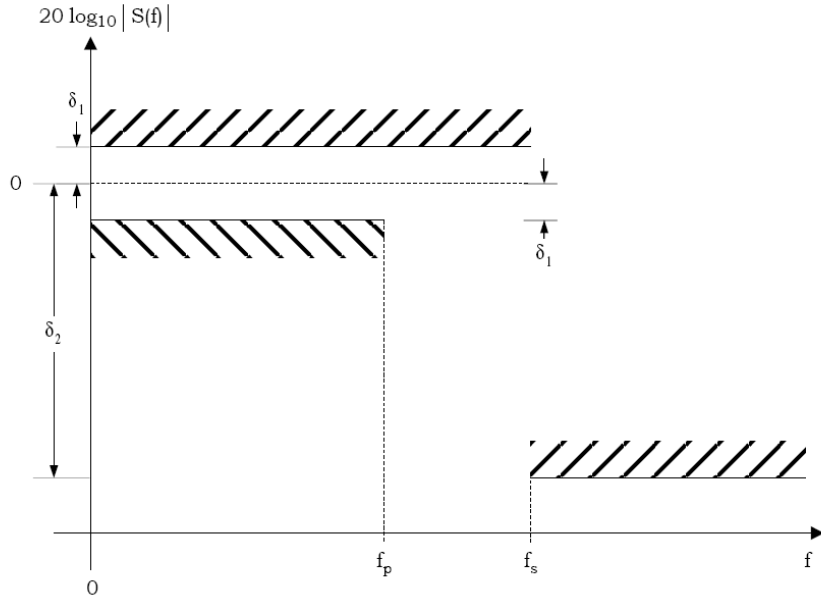


Figura 4.7 Respuesta de Frecuencia Filtro Banda Base.

Si s(f) es la respuesta el impulso del filtro Banda Base, entonces s(f) satisface la siguiente ecuación:

$$Mean\ Square\ Error = \sum_{k=0}^{\infty}\left[\alpha s\left(kTs - \tau\right) - h\left(k\right)\right]^2 \leq 0.03 \qquad (8)$$

Donde la constante α y τ son usadas para minimizar el error mediano cuadrado. La constante T_s es igual a 203.51… ns. T_s es igual a un cuarto de un PN chip. Los valores de los coeficientes h(k), para k < 48 están dados en la tabla X; h(k) = 0 para k ≥ 48. Note que h(k) es igual a h(47 – k).

k	h(k)	k	h(k)
0, 47	-0.025288315	12, 35	0.007874526
1, 46	-0.034167931	13, 34	0.084368728
2, 45	-0.035752323	14, 33	0.126869306
3, 44	-0.016733702	15, 32	0.094528345
4, 43	0.021602514	16, 31	-0.012839661
5, 42	0.064938487	17, 30	-0.143477028
6, 41	0.091002137	18, 29	-0.211829088
7, 40	0.081894974	19, 28	-0.140513128
8, 39	0.037071157	20, 27	0.094601918

9, 38	-0.021998074	21, 26	0.441387140
10, 37	-0.060716277	22, 25	0.785875640
11, 36	-0.051178658	23, 24	1.0

Tabla X Coeficientes de h(k).

PARAMETROS DE CONFIGURACION FILTRO.	
Tipo de Filtro	Bandabase
Rizado pasa banda	3 dB
Frecuencia superior pasa banda	590 KHz.
Mínima atenuación banda de parada	40 dB
Frecuencia baja banda de parada	740 KHz.

Tabla XI Parámetros de Configuración Filtro.

4.1.3.2 FUNCION MATEMATICA.

Esta función matemática representa la ecuación de un filtro FIR, el mismo que se loa aborda en detalle en el anexo D de FILTROS.

$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k] \tag{9}$$

4.1.3.3 DIAGRAMA DIGITAL.

El diagrama digital mostrado es una forma general de representación de cualquier filtro FIR Forma-Directa.

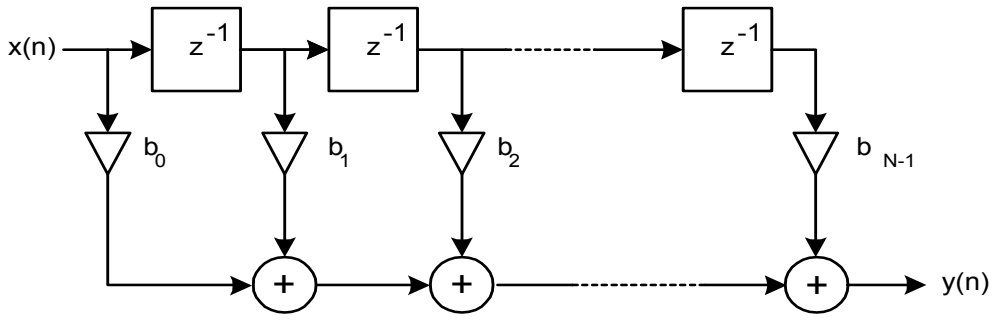


Figura 4.8 Diagrama Digital Filtro FIR.

4.1.3.4 DIAGRAMA DE FLUJO DEL BLOQUE.

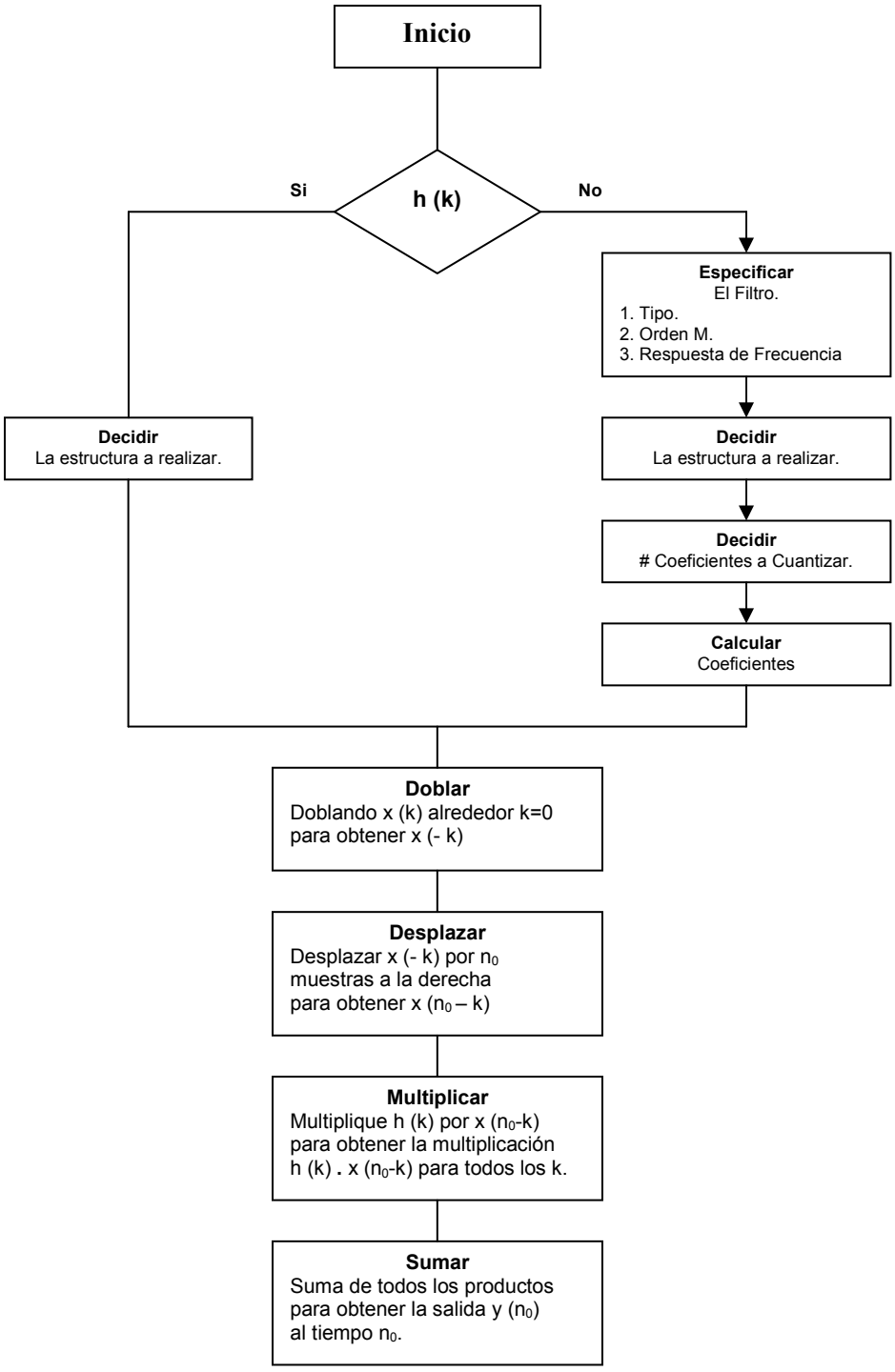


Figura 4.9 Diagrama de Flujo diseño de filtros.

4.2 IMPLEMENTACION DEL CODIGO EN EL DSP.

Para la implementación del código de cada bloque se debe primero establecer un procedimiento de diseño, simulación, verificación y codificación de los modelos a desarrollar. En base al funcionamiento y teniendo en claro la función que desempeña cada etapa se deben tomar los parámetros a modelar, estos parámetros pueden ser entre otros: tipo de plataforma, tiempos de muestreo, nivel de optimización, etc.; a partir de esto se crea el entorno de desarrollo y simulación del sistema.

CAPITULO 5.

PRUEBAS.

5.1 PRUEBAS SIMULADAS EN MATLAB DE LOS BLOQUES CONSTITUYENTES.

GENERADOR DE CODIGO WALSH:

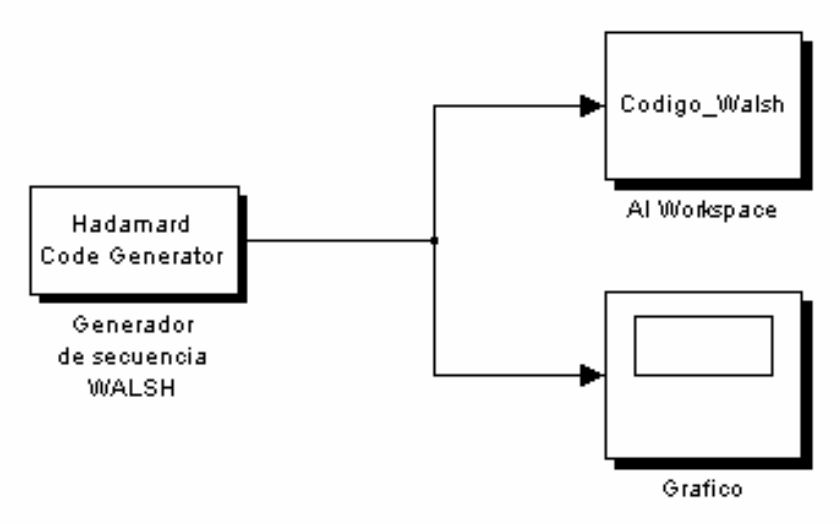


Figura 5.1 Modelo Generador de Walsh

Para esta prueba en el entorno de diseño de Simulink importamos las siguientes herramientas:

Hadamard Code Generator: el mismo que esta ubicado en *Simulink Library Browser -> Communications Blockset -> Comm Sources -> Sequence Generators -> Hadamard Code Generator,* este bloque tiene la función de generar en base a la matriz semilla Hadamard (ver sección 4.1.1.2) cualquiera de las secuencias Walsh en formato bipolar.

To Workspace: el mismo que esta ubicado en *Simulink Library Browser -> Simulink -> Sinks -> To Workspace*, este bloque tiene la función de almacenar en una variable de memoria el resultado de la generación del Código Walsh.

Scope: el mismo que esta ubicado en *Simulink Library Browser -> Simulink -> Sinks -> Scope*, este bloque tiene la función de visualizar en el dominio del tiempo lo que esta ocurriendo en el bloque Generador de Código Walsh.

Una vez realizado el modelo mostrado en la figura 5.1 procedemos a configurar los parámetros iniciales de cada bloque. Para nuestro caso los parámetros iniciales quedaron de la siguiente manera:

PARÁMETROS DE BLOQUE GENERADOR DE SECUENCIA WALSH.			
	W0	W32	W63
Longitud Código	**64**		
Índice Código	**0**	**32**	**63**
Tiempo Muestreo	**0.2**		
Salida Basada en Trama	**Desactivado**		

Tabla XII Parámetros de Bloque Walsh.

PARÁMETROS DE BLOQUE TO WORKSPACE			
	W0	W32	W63
Nombre Variable	**Codigo_Walsh**		
Limit data points to last	**inf**		
Decimation	**1**		
Tiempo Muestreo	**-1**		
Formato De Grabado	**Structure**		
Log fixed-point data as a fi object	**Desactivado**		

Tabla XIII Parámetros de Bloque Workspace (Walsh).

PARÁMETROS DE BLOQUE SCOPE			
GENERAL			
	W0	W32	W63
Ejes	**1**		
Rango Tiempo	**14**		
Tick Labels	**Bottom axis only**		
Muestreo	**Decimation**		
	1		
Visor Flotante	**Desactivado**		

Tabla XIV Parámetros de Bloque Scope General (Walsh).

PARÁMETROS DE BLOQUE SCOPE			
DATA HISTORY			
	W0	W32	W63
Limit data points to last	**5000**		
Grabar en Workspace	**Variable Name: Del_grafico**		
	Format: Structure		

Tabla XV Parámetros de Bloque Scope Data History (Walsh).

Luego de diseñar y configurar los parámetros del modelo, debemos darle un nombre, para luego guardar el diseño dando clic en *File -> Save: Walsh_final.*

Una vez guardado procedemos a ejecutar la simulación dando un tiempo de parada de 12.6 segundos.

A continuación se muestran los resultados obtenidos en el visor (Scope):

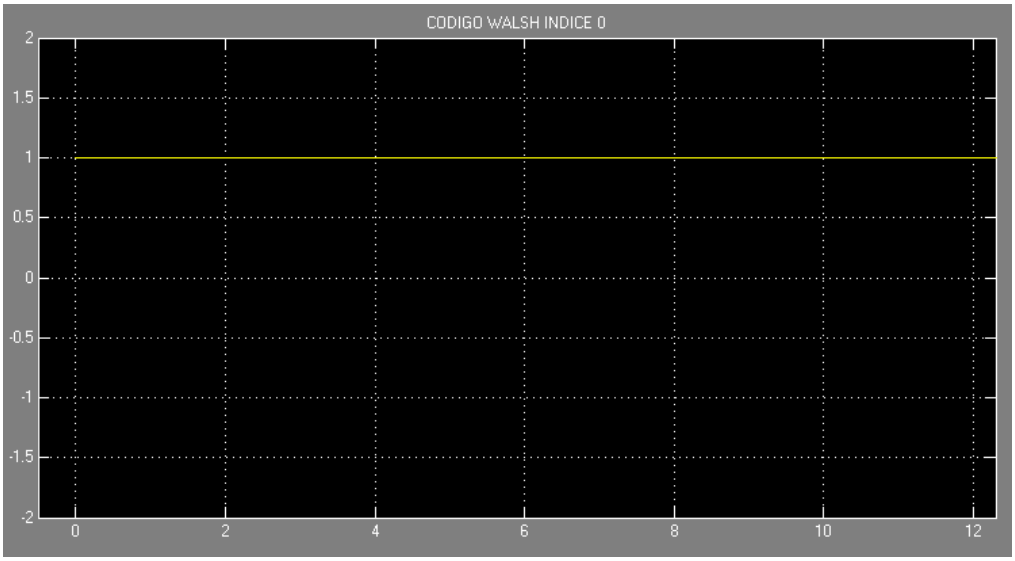


Figura 5.2 Gráfico de Walsh 0.

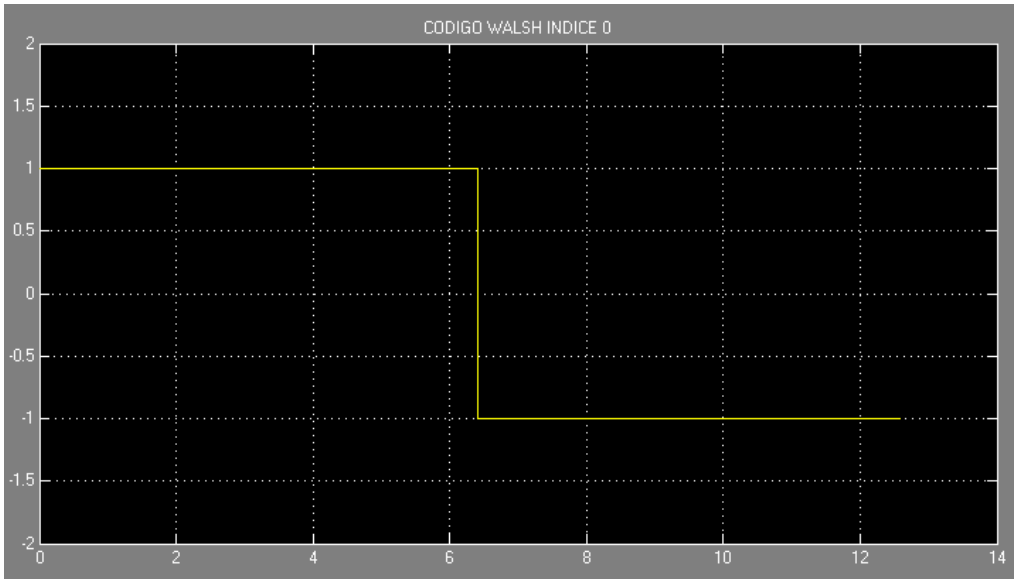


Figura 5.3 Gráfico de Walsh 32.

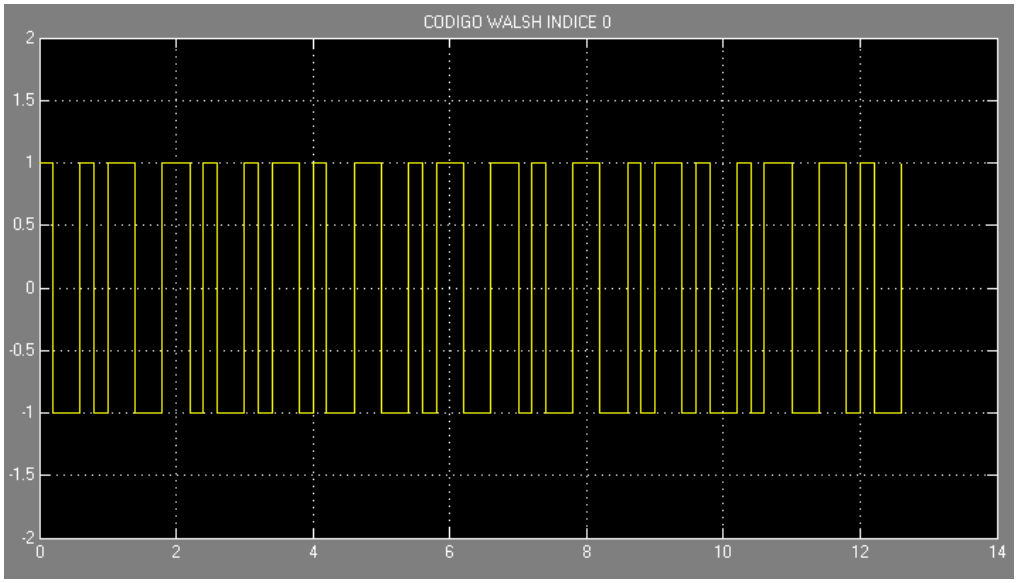


Figura 5.4 Gráfico de Walsh 63.

También estos valores simultáneamente fueron almacenados en la variable que dirigimos al Workspace, en nuestro caso se llama: *Codigo_Walsh.*

Nota: Los resultados de la variable *Codigo_Walsh* se muestran en el Anexo E.

GENERADOR DE SECUENCIA PN:

RAMAL I

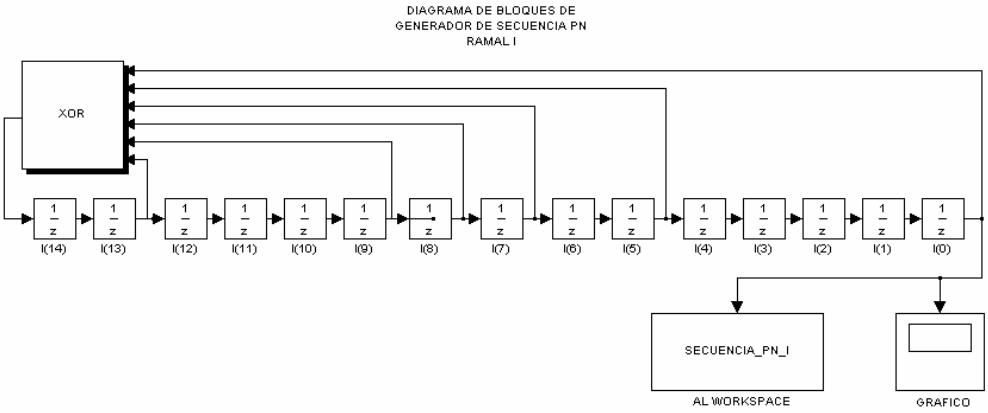


Figura 5.5 Modelo Generador Secuencia PN I

RAMAL Q

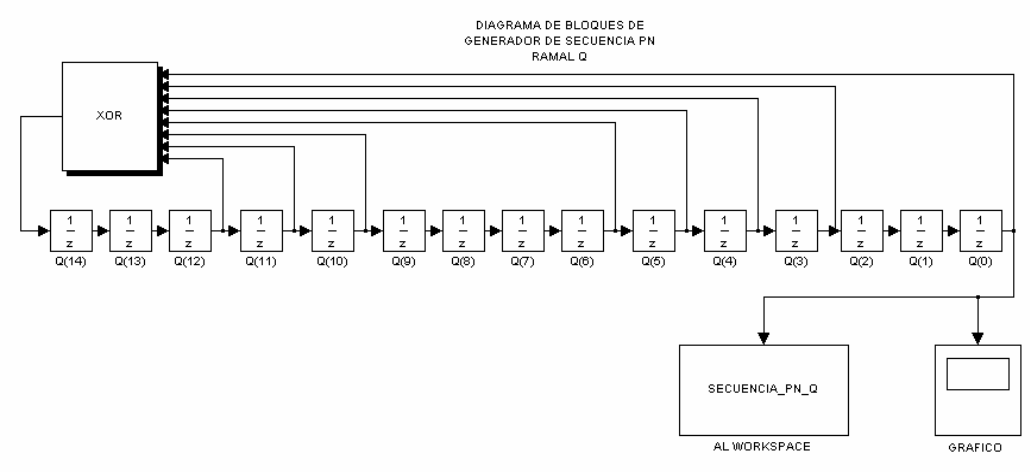


Figura 5.6 Modelo Generador Secuencia PN Q

Para esta prueba en el entorno de diseño de Simulink importamos las siguientes herramientas:

XOR: el mismo que esta ubicado en *Simulink Library Browser -> Simulink -> Logic and bit Operations -> Logical Operador*, este bloque tiene la función de realizar la función lógica requerida.

Unit Delay: el mismo que esta ubicado en *Simulink Library Browser -> Simulink -> Commonly Used Blocks -> Unit Delay,* este bloque tiene la

función de muestrear y almacenar un estado, para nuestro caso este trabajara como un registro de desplazamiento con un valor inicial fijo.

To Workspace: el mismo que esta ubicado en *Simulink Library Browser -> Simulink -> Sinks -> To Workspace*, este bloque tiene la función de almacenar en una variable de memoria el resultado de la generación del Código PN.

Scope: el mismo que esta ubicado en *Simulink Library Browser -> Simulink -> Sinks -> Scope*, este bloque tiene la función de visualizar en el dominio del tiempo lo que esta ocurriendo en el bloque Generador de Código PN.

Una vez realizado los modelos mostrados en las figuras 5.5 y 5.6 procedemos a configurar los parámetros iniciales de cada bloque. Para nuestro caso los parámetros iniciales quedaron de la siguiente manera:

PARÁMETRO DE BLOQUE XOR		
	Ramal I	Ramal Q
Operador	**XOR**	
Puertos de Entrada	**6**	**8**

Forma de Icono	**Rectangular**
Tiempo Muestreo	**-1**

Tabla XVI Parámetros de bloque XOR.

PARAMETROS DE BLOQUES UNIT DELAY		
	RAMAL I	**RAMAL Q**
Condición Inicial	I(14)=0	Q(14)=0
	I(13)=0	Q(13)=0
	I(12)=0	Q(12)=0
	I(11)=0	Q(11)=0
	I(10)=0	Q(10)=0
	I(9) = 0	Q(9) = 0
	I(8) = 0	Q(8) = 0
	I(7) = 0	Q(7) = 0
	I(6) = 0	Q(6) = 0
	I(5) = 0	Q(5) = 0
	I(4) = 0	Q(4) = 0

	I(3) = 0	Q(3) = 0
	I(2) = 0	Q(2) = 0
	I(1) = 0	Q(1) = 0
	I(0) = 1	Q(0) = 1
Tiempo Muestreo	**-1**	

Tabla XVII Parámetros de bloques Unit Delay.

Nota: Las condiciones iniciales de los bloques Unit Delay son el código semilla del Generador de Código PN. Además debe tenerse en consideración que al menos uno de estos debe ser 1, en nuestro caso se configuro a I (0)=1 y Q (0)=1.

PARÁMETROS DE BLOQUE TO WORKSPACE		
	RAMAL I	RAMAL Q
Nombre Variable	**SECUENCIA_PN_I**	**SECUENCIA_PN_Q**
Limit data points to last	**inf**	
Decimation	**1**	
Tiempo Muestreo	**-1**	

Formato Grabado	**Structure**
Log fixed-point data as a fi object	**Desactivado**

Tabla XVIII Parámetros de bloque Workspace (PN).

PARÁMETROS DE BLOQUE SCOPE	
GENERAL	
Ejes	**1**
Rango Tiempo	**500**
Tick Labels	**Bottom axis only**
Muestreo	**Decimation**
	1
Visor Flotante	**Desactivado**

Tabla XIX Parámetros bloque Scope General (PN).

PARÁMETROS DE BLOQUE SCOPE		
DATA HISTORY		
	RAMAL I	RAMAL Q
Limit data points to last	**5000**	
Grabar al WS	Variable Name: **VAL_RAMAL_I**	Variable Name: **VAL_RAMAL_Q**
	Format: **Structure**	

Tabla XX Parámetros bloque Scope Data History (PN).

Luego de diseñar y configurar los parámetros del modelo, debemos darle un nombre, para luego guardar el diseño dando clic en *File -> Save: RAMAL I* para el primer modelo *y File -> Save*: *RAMAL Q* para el segundo modelo.

Una vez guardado procedemos a ejecutar la simulación dando un tiempo de parada de 32768 segundos.

A continuación se muestran los resultados obtenidos en el visor (Scope):

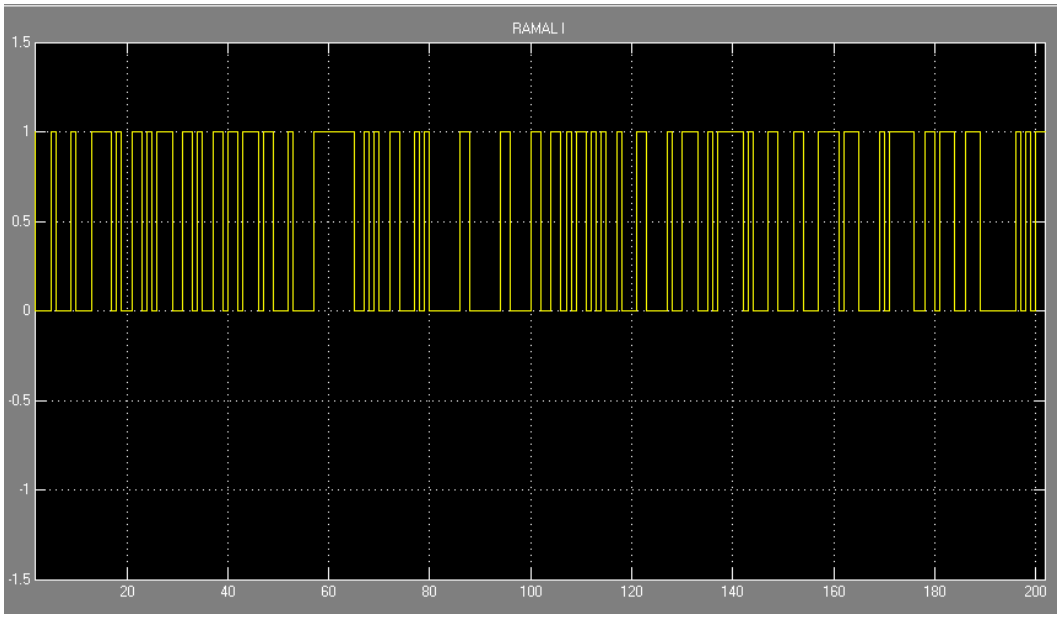


Figura 5.7 Grafico de Secuencia PN Ramal I.

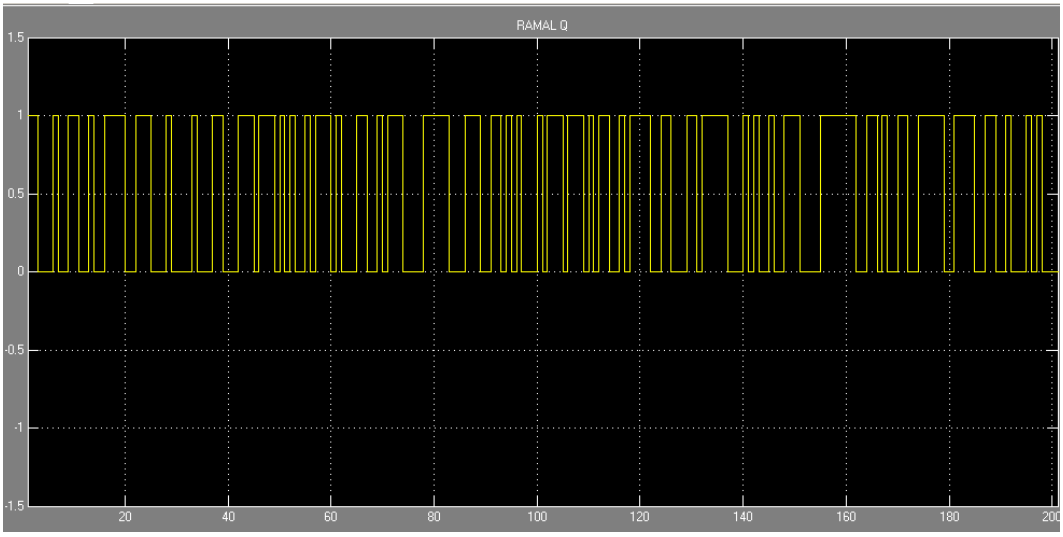


Figura 5.8 Grafico de Secuencia PN Ramal Q.

También estos valores simultáneamente fueron almacenados en la variables que designamos al Workspace, en nuestro caso: *SECUENCIA_PN_I* para el primer modelo, y *SECUENCIA_PN_Q.* para el segundo modelo. Al simular nuestro modelo y para confirmar el correcto funcionamiento utilizamos el bloque PN Sequence Generator del Communications Blockset y configurando los parámetros correspondientes con su estado inicial igual al de nuestro modelo diseñado, concordaron todos los valores generados.

FILTRO:

Para el diseño de este filtro y en general de cualquier filtro utilizamos la herramienta de Matlab FDA Tool, la cual nos permite configurar un filtro de acuerdo a las necesidades. Como se observamos anteriormente en el diagrama de flujo correspondiente a filtros nosotros debemos conocer si tenemos o no los coeficientes que definen la estructura del filtro.

SIN COEFICIENTES (DISEÑADO):

Para diseñar el filtro y obtener coeficientes, en la ventana de comandos de Matlab tipeamos

>> fdatool, inmediatamente aparecerá la siguiente pantalla, en la cual se configuran los parámetros del filtro.

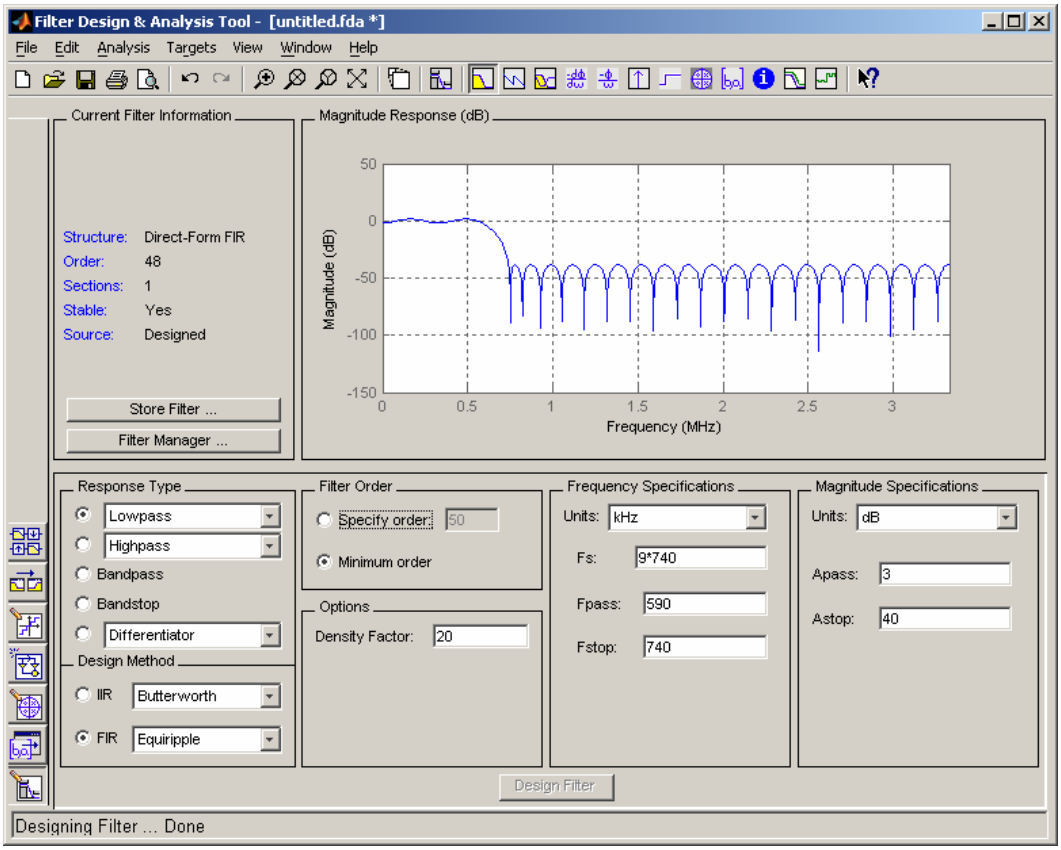


Figura 5.9 Ventana de Configuración FDA tool.

- Especificar el tipo de respuesta (Lowpass, Highpass, Bandpass, Bandstop), para nuestro caso es el filtro lowpass.
- Especificar Frecuencia:

  Fpass = Frecuencia pasabanda = 590 KHz.

  Fstop = Frecuencia de parada = 740 KHz.

  Fs = Frecuencia de muestreo = 9*Fstop KHz.

  Units = Unidades = KHz.

Note que al incrementar la frecuencia de muestreo aumenta el orden del filtro.

- Especificar Magnitud:

  Apass = 3 dB

  Astop = 40 dB.

- Decidir el tipo de estructura: *Edit -> Convert Structure -> Direct-Form FIR.*

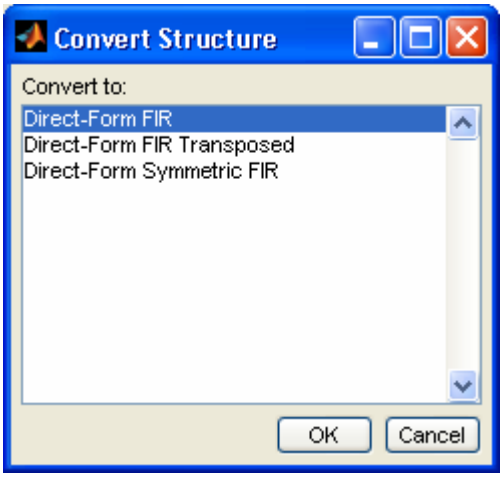


Figura 5.10 Selección de tipo de Estructura de Filtro.

- Calcular los coeficientes: realizados los puntos anteriores presionamos Design Filter. Inmediatamente se procesara y calculara los coeficientes, los mismos que pueden ser almacenados dando clic en: *Targets -> Generate C Header*, se

debe hacer click en Generate y especificar una ruta y un nombre para el archivo de coeficientes.

- Realizados los puntos anteriores y siguiendo las especificaciones de diseño, procedemos a crear el modelo final del filtro dando click en el icono Realize Model.

CON COEFICIENTES (IMPORTADOS):

- El primer paso para crear este filtro, es crear una variable en el Workspace que contenga los coeficientes, realizando los siguientes pasos. Dar click en el botón New variable, asignar nombre a la variable, para nuestro caso *coeficientes*.
- Ingresar los valores de los coeficientes.
- En el entorno Matlab tipeamos >> fdatool, inmediatamente aparecerá la pantalla mostrada en la figura 5.11
- En la parte inferior izquierda seleccionamos el icono Import Filter.
- En el campo estructura de filtro escogemos Forma-Directa FIR.
- En el campo Numerator invocamos la variable creada en el primer paso coeficientes
- En el campo Units, escogemos la frecuencia deseada.
- Luego damos click en el botón Import Filter.

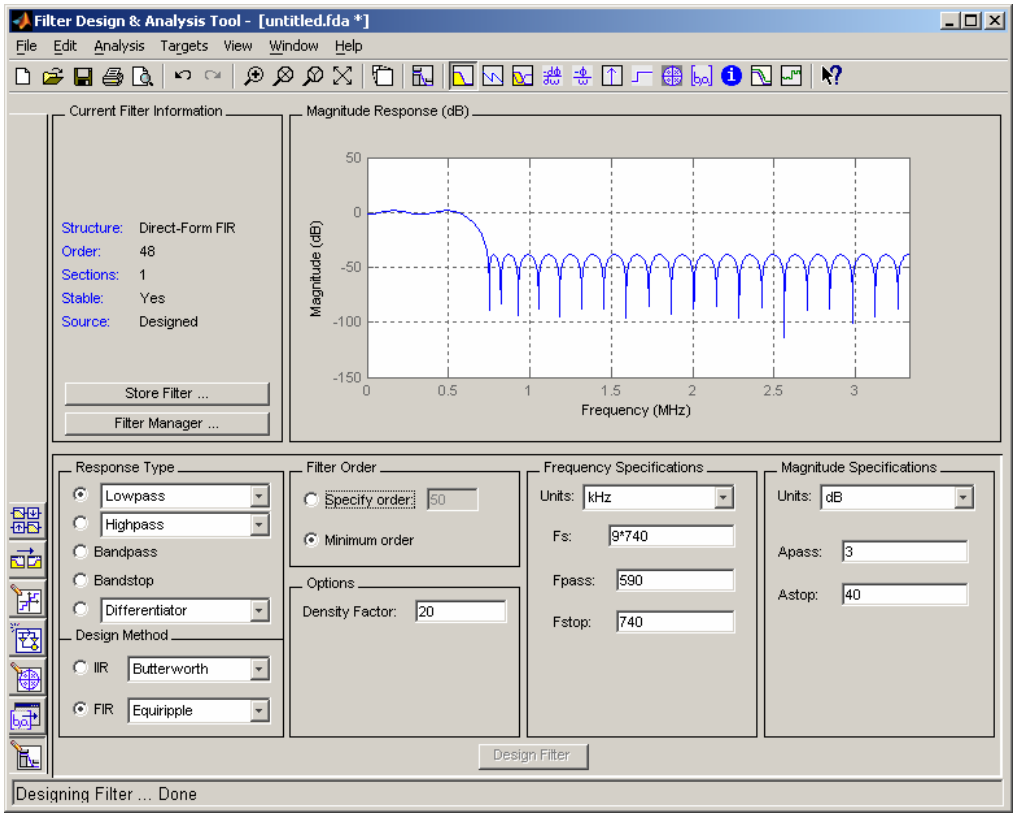


Figura 5.11 Diseño de Filtro con coeficientes.

- Realizados los puntos anteriores y siguiendo las especificaciones de diseño, procedemos a crear el modelo final del filtro dando click en el icono Realize Model.

5.2 PRUEBA DE CARGA DE PROGRAMA FUENTE GENERADO POR MATLAB AL DSP.

GENERADOR DE CODIGO WALSH:

El objetivo de este punto es descargar el modelo Simulink en nuestro DSP, para lo cual generamos un archivo objeto a partir del modelo Walsh_final_hardware mostrado en la figura 5.9.

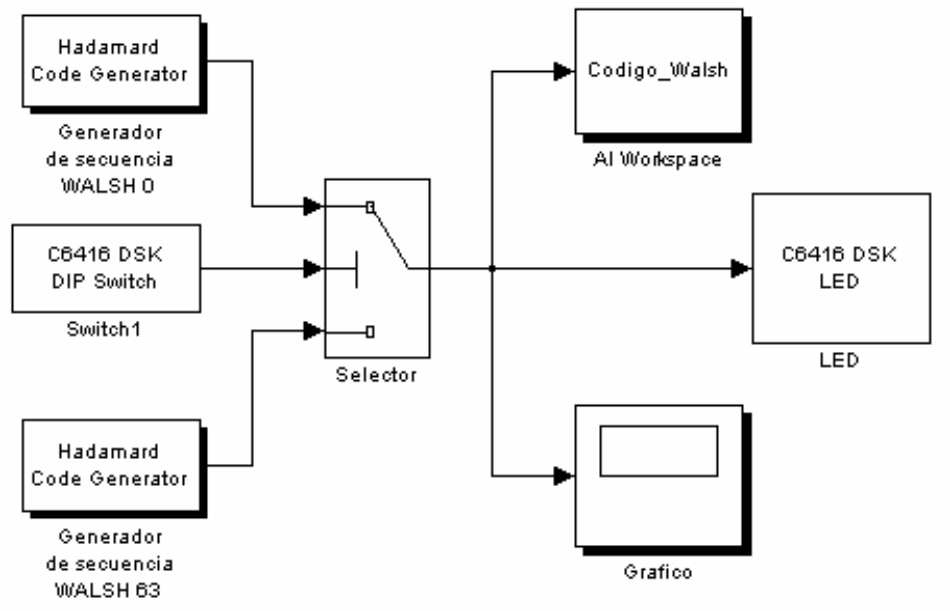


Figura 5.12 Modelo de Generador de Código Walsh.

Al modelo mostrado en el punto 5.1.a le hemos agregado los siguientes bloques:

DIP SWITCH: este bloque esta ubicado en *Simulink Library Browser -> Embedded Target for TIC6000 DSP -> C6416 DSK Board Support -> Switch*, y su función es la de generar una trama de 4 bits que al ser cargados en el DSK C6416 pueden controlar externamente la múltiplexación del Walsh 63 o del Walsh 0. En la siguiente tabla se muestran las combinaciones posibles de los DIPs:

DIP				SALIDA A LED
3	2	1	0	
0	0	0	0	WALSH 63
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	WALSH 0
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Tabla XXI Salida de Códigos Walsh.

LED: este bloque esta ubicado en *Simulink Library Browser -> Embedded Target for TIC6000 DSP -> C6416 DSK Board Support -> LED*, y su función es la de mostrar los resultados del Código Walsh

seleccionado por los DIP. A continuación se muestra la interpretación de los LEDs:

LED				NÚMERO BIPOLAR
3	2	1	0	
OFF	OFF	OFF	ON	1
ON	ON	ON	ON	-1

Tabla XXII Representación de Símbolos bipolar.

SWITCH: este bloque esta ubicado en *Simulink Library Browser -> Simulink -> Signal Routing -> Switch*, y su función es multiplexar los datos que ingresan, estos a su vez son controlados por el valor límite configurado previamente como se muestra en la tabla:

PARÁMETROS DE BLOQUE SWITCH	
MAIN	
Criterio de salida	**U2 > threshold**
Limite	**4**
Habilitar Cero	**Activado**
Tiempo de Muestreo	**-1**

Tabla XXIII Parámetros bloque Switch (Walsh).

Una vez realizado la configuración de los bloques que componen el modelo procedemos a configurar los parámetros generales de Simulink para que la implementación en el hardware suceda (esto se detallo en la sección 3.2.1).

A continuación se muestra una tabla con los parámetros que se configuraron para nuestro DSK:

PARAMETROS DE CONFIGURACION: WALSH_FINAL_HARDWARE/CONFIGURATION		
Resolvedor	Start time: **0.0**	Stop time : **12.6**
	Type: **Fixed-step**	Solver: **Discrete**
Implementacion Hardware	Device Type: **TI C6000**	
Real-Time Workshop	RTW system target file: **ti_c6000.tlc**	

Tabla XXIV Parámetros globales Walsh_Final_Hardware.

Una vez configurados los parámetros generales de Simulink en Matlab procedemos a:

1. Conectar el DSK C6416 a la línea de alimentación.
2. Automáticamente se ejecuta el POST (Power On Self Test).
3. Conectamos el cable USB del DSK al computador.

4. Realizamos el Diagnostics Utility (Ver Anexo DIAGNOSTICO DSK).
5. Hacemos clic en el icono de C6416DSK CCS.
6. En el entorno de Diseño Simulink hacemos clic en Tools -> Real-Time Workshop -> Build Model o CTRL + B.

Como se detallo en la sección 3.2.1, al realizar el punto 6 se generan los archivos de nuestro modelo (*Walsh_final_hardware*) en el entorno de Matlab, y a su vez se genera un proyecto activo en el ambiente CCS. En el entorno de CCS con el proyecto activo procedemos a compilar, cargar y ejecutar el modelo en nuestro DSK (ver sección 3.2.2)

GENERADOR DE SECUENCIA PN:

El objetivo de este punto es descargar el modelo Simulink en nuestro DSP, para lo cual generamos un archivo objeto a partir del modelo *RAMAL_I_Q_FINAL_HARDWARE* mostrado en la figura 5.10.

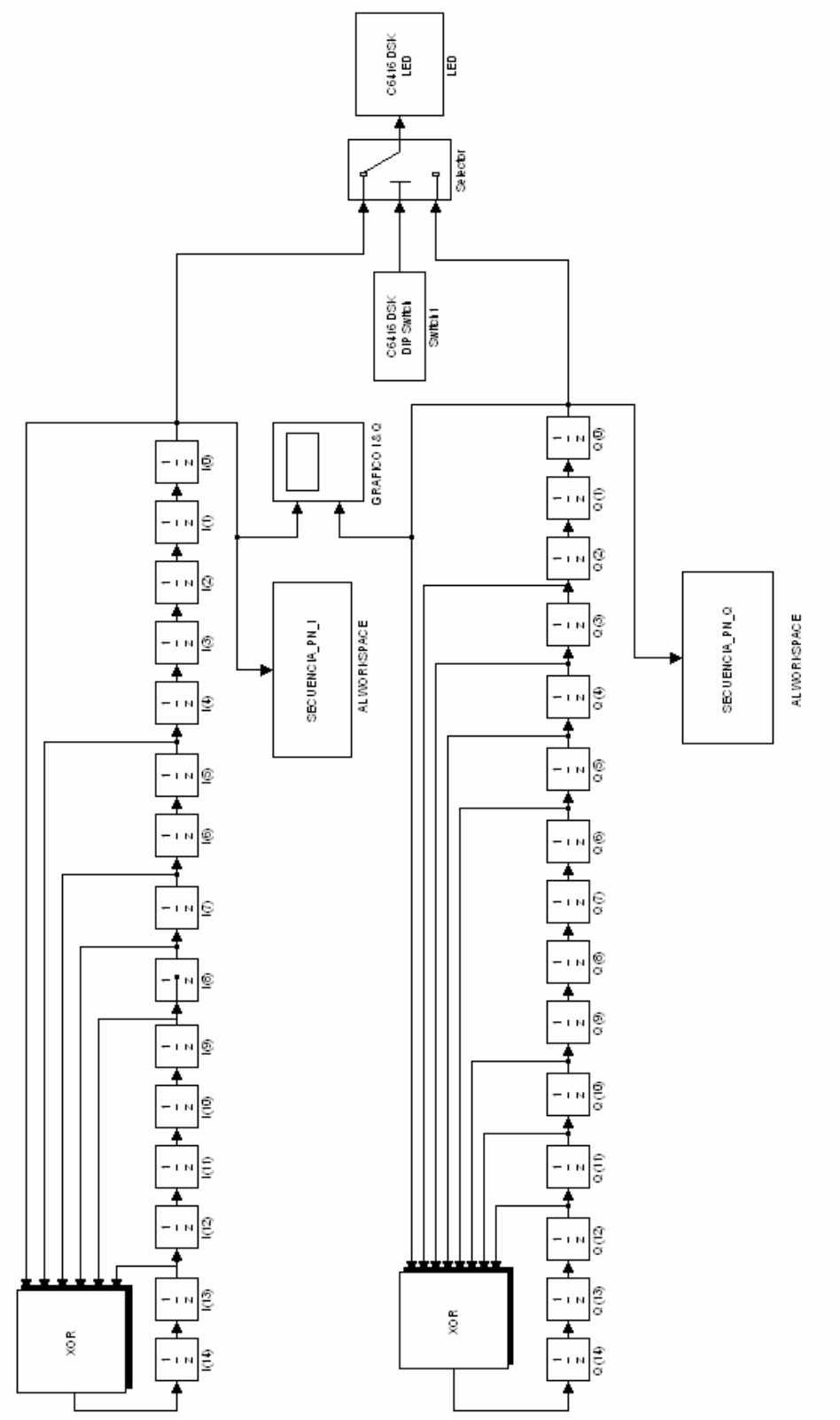


Figura 5.13 Modelo de Generador de Código PN.

Los modelos de las figuras 5.5 y 5.6 se juntaron para generar las secuencias I y Q en paralelo y obtener sus respectivas salidas en los LED dependiendo de que el usuario presione en los DIP. Si se presiona un valor mayor a 4 entonces los LED mostraran la secuencia del Ramal I, pero si no presionamos los DIP entonces los LED mostraran la secuencia del Ramal Q.

DIP SWITCH: este boque esta ubicado en *Simulink Library Browser -> Embedded Target for TIC6000 DSP -> C6416 DSK Board Support -> Switch*, y su función es la de generar una trama de 4 bits que al ser cargados en el DSK C6416 pueden controlar externamente la múltiplexación del Ramal I o del Ramal Q. En la siguiente tabla se muestran las combinaciones posibles de los DIPs:

DIP				SALIDA A LED
3	**2**	**1**	**0**	
0	0	0	0	RAMAL Q
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	RAMAL I
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	

1	1	0	1	
1	1	1	0	
1	1	1	1	

Tabla XXV Salidas de Ramales I o Q.

LED: este bloque esta ubicado en *Simulink Library Browser -> Embedded Target for TIC6000 DSP -> C6416 DSK Board Support -> LED*, y su función es la de mostrar los resultados del Código PN seleccionado por los DIP. A continuación se muestra la interpretación de los LEDs:

LED				NUMERO
3	2	1	0	
OFF	OFF	OFF	ON	1
OFF	OFF	OFF	OFF	0

Tabla XXVI Representación de Símbolos Unipolar.

SWITCH: este bloque esta ubicado en *Simulink Library Browser -> Simulink -> Signal Routing -> Switch*, y su función es multiplexar los datos que ingresan, estos a su vez son controlados por el valor límite configurado previamente como se muestra en la tabla:

PARÁMETROS DE BLOQUE SWITCH	
MAIN	
Criterio de Salida	**U2 > threshold**
Salida	**4**
Habilitar cero	**Activado**
Tiempo de Muestreo	**-1**

Tabla XXVII Parámetros bloque Switch (PN).

Una vez realizado la configuración de los bloques que componen el modelo procedemos a configurar los parámetros generales de Simulink para que la implementación en el hardware suceda (esto se detallo en la sección 3.2.1).

A continuación se muestra una tabla con los parámetros que se configuraron para nuestro DSK:

PARAMETROS DE CONFIGURACION: RAMAL_I_Q_FINAL_HARDWARE /CONFIGURATION		
Resolvedor	Start time: **0.0**	Stop time : **32768**
	Type: **Fixed-step**	Solver: **Discrete**

Implementacion Hardware	Device Type: **TI C6000**
Real-Time Workshop	RTW system target file: **ti_c6000.tlc**

Tabla XXVIII Parámetros globales RAMAL_I_Q_FINAL_HARDWARE.

Una vez configurados los parámetros generales de Simulink en Matlab procedemos a, compilar el modelo CTRL + B.

Como se detallo en la sección 3.2.1, al compilar se generan los archivos de nuestro modelo (*RAMAL_I_Q_FINAL_HARDWARE*) en el entorno de Matlab, y a su vez se genera un proyecto activo en el ambiente CCS. En el entorno de CCS con el proyecto activo procedemos a compilar, cargar y ejecutar el modelo en nuestro DSK (ver sección 3.2.2)

FILTROS

El objetivo de este punto es descargar el modelo Simulink en nuestro DSP, para lo cual generamos un archivo objeto a partir del modelo *FILTRO_FINAL_HARDWARE* mostrado en la figura 5.11.

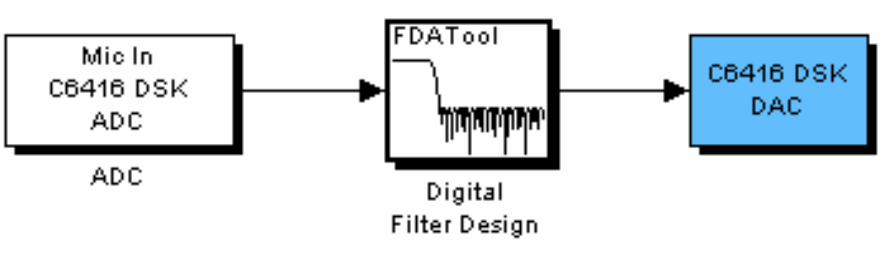


FIGURA 5.14 Modelo del Filtro Digital.

Este modelo consta de tres bloques los cuales se detallan a continuación:

MIC IN: este boque esta ubicado en *Simulink Library Browser -> Embedded Target for TIC6000 DSP -> C6416 DSK Board Support -> ADC*, y su función es la de proveer un puerto de ingreso de la señal analógica. Los parámetros de este bloque se muestran en la siguiente tabla:

PARÁMETROS DE BLOQUE ADC	
Fuente ADC	**Mic In**
+20 dB Mic gain boost	**Desactivado**
Stereo	

Tiempo Muestreo	**32 KHz.**
Ancho de palabra	**32 bits**
Tipo dato de salida	**Double**
Escalamiento	**Normalize**
Muestras por trama	**64**

Tabla XXIX Parámetros bloque ADC.

DIGITAL FILTER DESIGN: este bloque esta ubicado en *Simulink Library Browser -> Signal Processing Blockset -> Filtering -> Filter Designs -> Digital Filter Design*, y su función es la de crear un filtro digital basado en las especificaciones de diseño. Para nuestro caso los parámetros quedaron de la siguiente manera:

PARÁMETROS DE BLOQUE DISEÑO DE FILTRO.	
Estructura Filtro	**Direct-form FIR**
Numerador	**Tabla de coeficientes**

Frecuencia Muestreo	**Hz**
	30000

Tabla XXX Parámetros bloque Diseño de Filtro.

DAC: este bloque esta ubicado en *Simulink Library Browser -> Embedded Target for TIC6000 DSP -> C6416 DSK Board Support -> DAC*, su función es proveer en el puerto de salida una señal analógica. Los parámetros quedaron de la siguiente manera:

PARÁMETROS DE BLOQUE DAC	
Ancho Palabra	**32 bits**
Escalamiento	**Normalize**
Modo de Saturacion	**Saturate**

Tabla XXXI Parámetros bloque DAC (Filtro).

Una vez realizado la configuración de los bloques que componen el modelo procedemos a configurar los parámetros generales de Simulink,

para que la implementación en el hardware ocurra (esto se detallo en la sección 3.2.1).

A continuación se muestra una tabla con los parámetros que se configuraron para nuestro DSK:

PARAMETROS DE CONFIGURACION: RAMAL_I_Q_FINAL_HARDWARE /CONFIGURATION		
Resolvedor	Start time: **0.0**	Stop time : **inf**
	Type: **Fixed-step**	Solver: **Discrete**
Implementacion Hardware	Device Type: **TI C6000**	
Real-Time Workshop	RTW system target file: **ti_c6000.tlc**	

Tabla XXXII Parámetros globales FILTRO_FINAL_HARDWARE.

Una vez configurados los parámetros generales de Simulink en Matlab procedemos a, compilar el modelo CTRL + B.

Como se detallo en la sección 3.2.1, al compilar se generan los archivos de nuestro modelo (*FILTRO_FINAL_HARDWARE*) en el entorno de Matlab, y a su vez se genera un proyecto activo en el ambiente CCS. En el entorno de CCS con el proyecto activo procedemos a compilar, cargar y ejecutar el modelo en nuestro DSK.

5.3 VERIFICACION DE PROGRAMA EN TARJETA.

GENERADOR DE CODIGO WALSH.

Una vez realizados los procedimientos detallados en las secciones 5.1 y 5.2 correspondientes al Generador de Código Walsh, procedemos a verificar el correcto funcionamiento del programa objeto que reside en nuestro DSK. Al manipular los DIPs y comprobar el funcionamiento del código en nuestro DSK se cumple la tabla que se muestra a continuación:

DIP				SALIDA A LED
3	2	1	0	
0	0	0	0	WALSH 63
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	WALSH 0
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Tabla XXXIII Verificación de Códigos Walsh.

Nótese que el Walsh 63 es una cadena variante en el tiempo de símbolos (1, -1, -1, 1……..1, -1, -1, 1), mientras que el Walsh 0 es una cadena invariante de símbolos (1, 1, 1, 1……1, 1, 1, 1). Estos símbolos se representan de acuerdo a la siguiente tabla:

NUMERO BIPOLAR	LED			
	3	2	1	0
1	OFF	OFF	OFF	ON
-1	ON	ON	ON	ON

Tabla XXXIV Verificación de salida en LEDs (Walsh)

.

Finalmente al desconectar el cable USB del DSK se comprueba que efectivamente el programa objeto reside en nuestra tarjeta.

GENERADOR DE SECUENCIA PN:

Una vez realizados los procedimientos detallados en las secciones 5.1 y 5.2 correspondientes al Generador de Código PN, procedemos a verificar el correcto funcionamiento del programa objeto que reside en nuestro DSK. Al manipular los DIPs y comprobar el funcionamiento del código en nuestro DSK se cumple la tabla que se muestra a continuación:

DIP				SALIDA A LED
3	2	1	0	
0	0	0	0	RAMAL Q
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	RAMAL I
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Tabla XXXV Verificación de Códigos PN.

Nótese que las secuencias PN I y Q son una cadena que varia en el tiempo encendiendo o apagando los LED de acuerdo a la siguiente representación:

NUMERO	LED			
	3	2	1	0
1	OFF	OFF	OFF	ON
0	OFF	OFF	OFF	OFF

Tabla XXXVI Verificación de salida en LEDs (PN).

Finalmente al desconectar el cable USB del DSK se comprueba que efectivamente el programa objeto reside en nuestra tarjeta.

FILTRO:

Una vez realizados los procedimientos detallados en las secciones 5.1 y 5.2 correspondientes al Filtro, procedemos a verificar el correcto funcionamiento del programa objeto que reside en nuestro DSK.

La prueba del filtro consiste en ingresar varias señales a distintas frecuencias al DSK C6416 por medio de su línea de entrada MIC IN

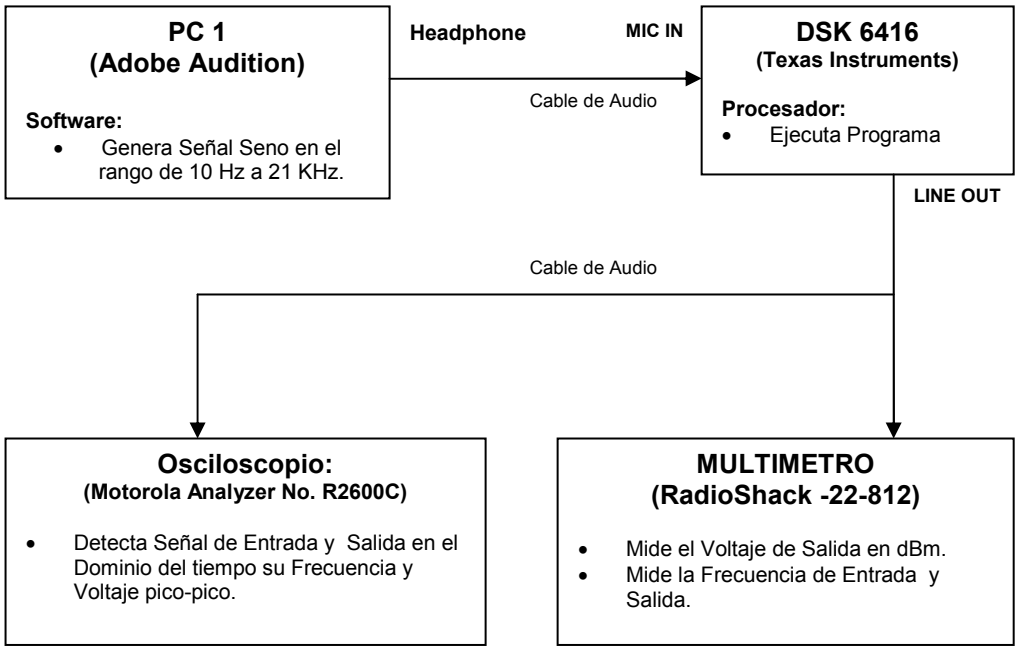


FIGURA 5.15 Diagrama de Pruebas de Filtro.

utilizando un generador externo. La señal es convertida internamente del dominio analógico al digital. Luego es procesada por el filtro que se esta ejecutando en el DSP. Por ultimo la señal es transformada del dominio digital al analógico, mostrándola en la LINE OUT del DSK C6416. En la tabla siguiente se muestran las mediciones realizadas:

FRECUENCIA (HZ)	dBm (MULTÌMETRO)
1000	5,3
4100	3,2
4200	3,0
4300	2,5
4400	0,1
4450	-1,1
4500	-3,4
5000	-24,0
9000	-20,0
12000	-13,0
14000	-16,0
16000	-50,0
18000	-24,0
20000	-40,0
21000	-36,0

Tabla XXXVII Valores obtenidos a la salida del Filtro.

A continuación se presenta una comparación entre el filtro simulado con la herramienta FDA tool de matlab versus un gráfico construido a partir de los valores reales de la tabla XXXVII.

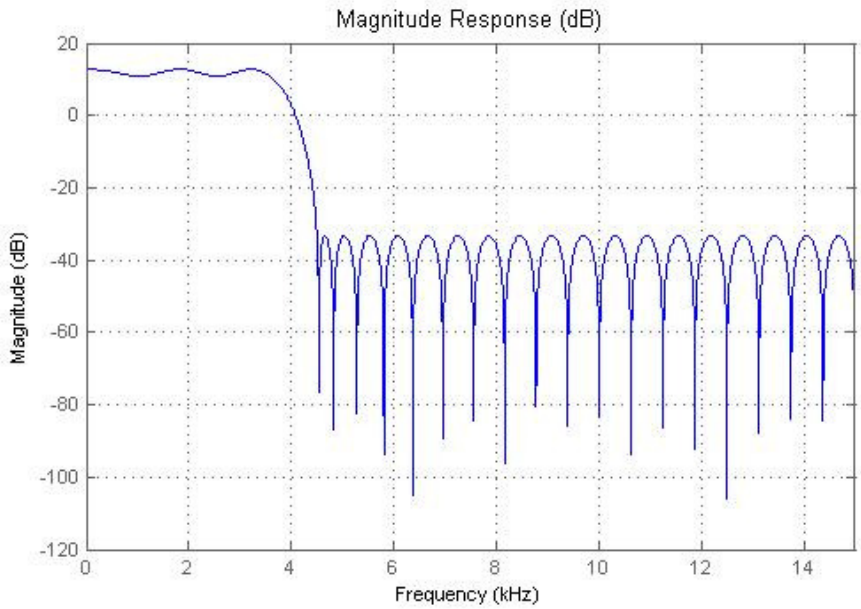


Figura 5.16 Respuesta en Magnitud filtro Pasa Bajo.

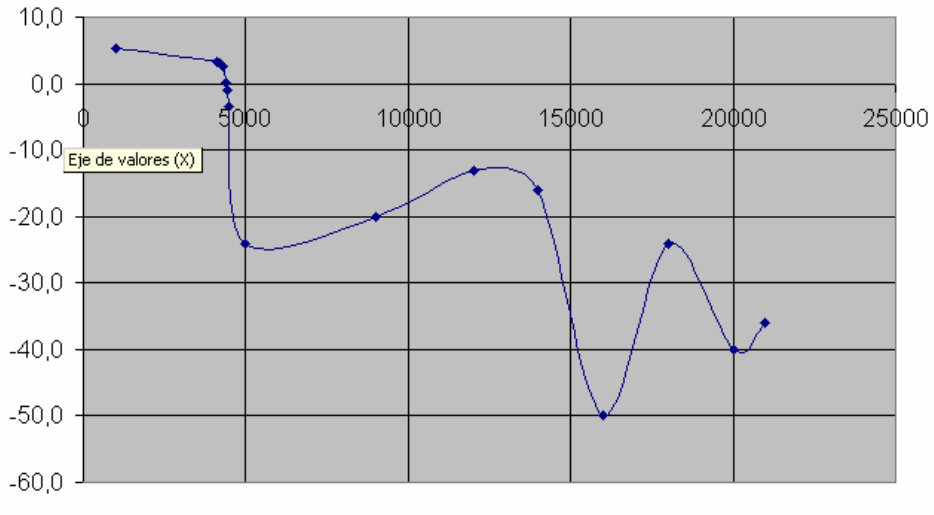


Figura 5.17 Respuesta en Magnitud Real.

CONCLUSIONES Y RECOMENDACIONES.

El uso óptimo de un recurso limitado como es el espectro radioeléctrico ha hecho que las investigaciones apunten al mejoramiento continuo de las técnicas de accesar al medio, para este propósito se invierten muchos recursos en el desarrollo de métodos de codificación robustos y eficientes, haciendo de estos sistemas, invulnerables a cualquier agente no deseado, con tal propósito el estándar CDMA hace uso de códigos Walsh o Pseudoaleatorios (PN). Hoy en día CDMA es uno de los estándares mas aceptados a nivel mundial debido al ancho de banda que maneja y a la gran capacidad de alojar usuarios, además por la seguridad que provee. Nuestro proyecto mostró las bases en las que se fundamenta CDMA, pues se generó el código Walsh 0 (o cualquier Walsh N) y las secuencias PN I y PN Q, con esto sabemos, que si a futuro deseamos actualizar el software, bastaría modificar los parámetros que propondrá el nuevo estándar.

Este proyecto da un primer paso en lo que respecta a la introducción de nuevas tecnologías en el procesamiento de señales digitales con DSP de la familia 6000, que son necesarias en el proceso de implementación de aplicaciones en el área de Telecomunicaciones, que requieren operaciones matemáticas precisas y complejas utilizando pocos ciclos de reloj para su ejecución, permitiendo así obtener aplicaciones en tiempo real.

En los últimos años se han dado cambios significativos en los métodos de enseñanza de Ingeniería y nuevas tecnologías, hoy en día disponemos de excelentes herramientas de análisis y modelación de sistemas como Matlab, que simplifican cálculos complejos y nos permiten realizar una exploración de manera interactiva del comportamiento de la variables involucradas en los problemas, mejorando de esta forma nuestra experiencia de aprendizaje que incentiva a vernos inmersos en el mundo de la investigación, pues nuestros conocimientos teóricos y necesidades se verán reflejadas en el proceso de desarrollo en cualquier aplicación.

Para futuras aplicaciones con el DSP C6416, se recomienda explorar aplicaciones de codificadores convolucionales, pues este dispositivo tiene en su arquitectura interna módulos que realizan coprocesamientos Turbo y Viterbi con una sola palabra de dato de configuración.

Para las herramientas de software Code Composer Studio y Matlab, recomendamos conseguir una versión de CCS 3.2 o superior, pues la que disponemos es la versión 2.21, la misma que no permite habilitar el Linker for CCS de Matlab, la misma que nos obliga a realizar compilaciones sucesivas del modelo para la verificación de los resultados en la tarjeta. Al disponer de la versión 3.2, podríamos realizar cambios en el modelo de Simulink, reflejándolos en tiempo real en el DSP (RTDX).

Para aplicaciones que ameritan el análisis de una señal externa, nuestro DSK dispone de un Codec AIC23 (24 bits y 96 KSPS) que realiza conversión Digital-Analógica D/A y Analógica-Digital A/D, el cual limita nuestras aplicaciones a bajas frecuencias. Lo que se recomienda es adquirir dispositivos de altas tasas de muestreo MSPS y alta resolución de bits compatibles con el DSK:

DAUGHTER CARDS (EVM)			
TIPO	**SERIE EVM**	**BITS**	**MSPS**
D/A	TLV5619/5639	12	1
A/D	THS1206	12	6

Nota: Visite el sitio Web: www.dspvillage.com/dskaccessories.

Para el correcto uso del DSK y de cualquier otro dispositivo electrónico se recomienda: Una correcta conexión a tierra, pues al no disponer de esto, la estática presente en nuestro cuerpo podría averiar permanentemente el dispositivo.

ANEXOS.

ANEXO A.

TMS320C6416 DSK.

El TMS320C6416 DSK, nos ayuda con la evaluación y desarrollo de aplicaciones, como esquemas lógicos, ecuaciones lógicas. Dispone de un hardware que reduce el tiempo de desarrollo en las aplicaciones.

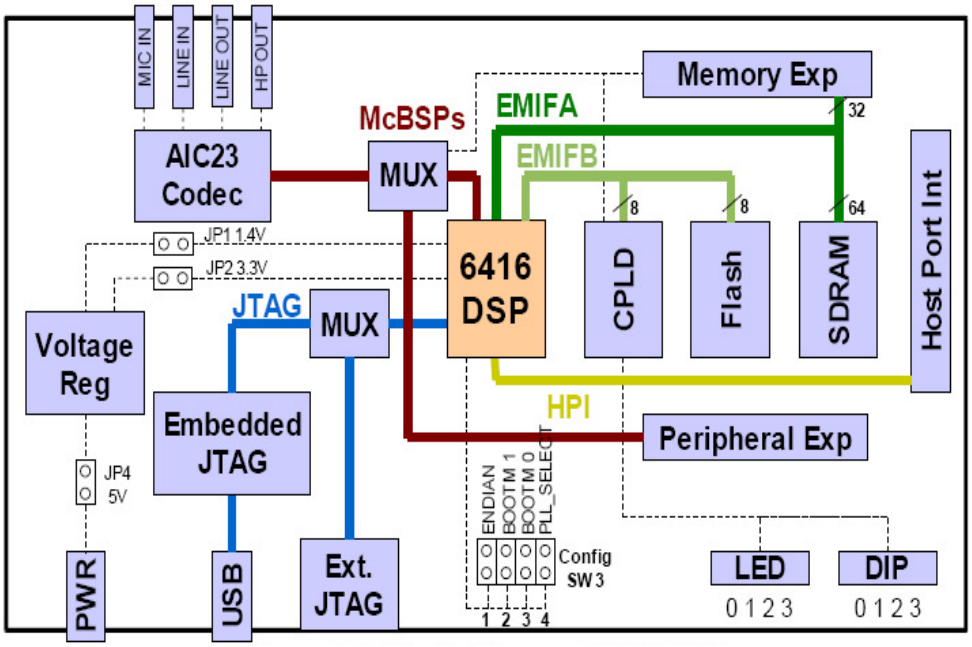


Figura A.1 Diagrama de Bloques del DSK C6416.

Dentro de las características podemos citar las siguientes:

- El Procesador Texas Instruments TMS320C6416 DSP, opera de 600 a 720 MHz.
- Posee en tarjeta un *AIC23* Codificador Stereo.
- Una DRAM sincrónica de 16MB.
- 512 KB de memoria FLASH no volátil.
- LEDS y DIP conmutadores, para el usuario.
- Software de configuración a través de los registros implementados en CPLD.
- Opciones de arranque configurable y selección de entrada del CLOCK.
- Conectores de expansión para el uso de Daughter Card.
- Fuente de voltaje de +5V.

FUNCIONAMIENTO DEL TMS320C6416 DSK.

El DSP en el 6416 DSK, posee comunicación a través uno de los dos buses, el de 64 bit de ancho EMIFA y el de 8 bit de ancho EMIFB. La SDRAM (EMIFA), Flash (EMIFB) y el CPLD (EMIFB) están conectados a estos buses. EMIFA esta también conectada a la tarjeta de expansión. El Codec AIC23 permite que el DSP transmita y reciba señales Analógicas. McBSP1, es usada para el codec control interfaces. McBSP2, es usada para datos.

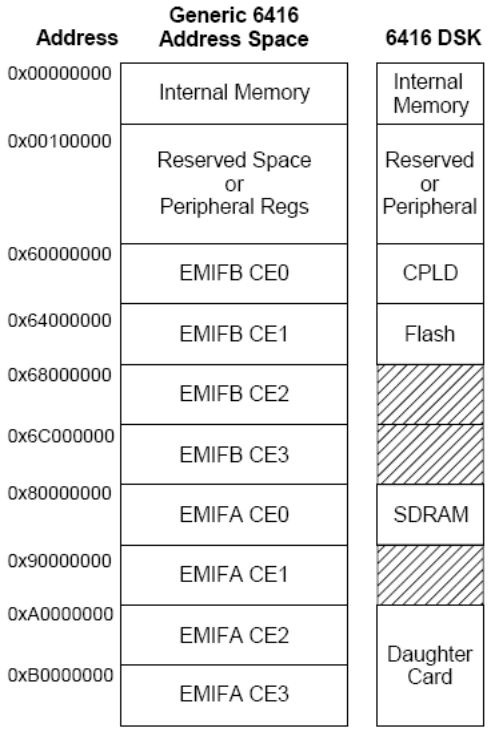


Figura A.2 Mapeo de Memoria del C6416 DSK.

Las entradas y salidas analógicas están localizadas a través de cuatro conectores de audio de 3.5 mm que corresponden a:

- MIC IN: Micrófono señal de entrada.
- LINE IN: Línea de Entrada.
- LINE OUT: Línea de Salida, ganancia fija.
- HP OUT: Audífonos de Salida, ganancia ajustable

El CODEC puede seleccionar el micrófono o la línea de ingreso como ingresos activos. McBSP1 y McBSP2 pueden ser re-enrutados a los conectores de expansión a través de software. El CPLD (Dispositivo Lógico Programable), es usado para implementar *glue logic* que están en la tarjeta. El CPLD también tiene una interface de usuario basada en un registro, que permiten al usuario configurar la tarjeta, para que lea o grabe los registros CPLD.

El DSK incluye cuatro LEDS y cuatro DIP switch de simple vía que permite interactuar al usuario. Ambos son accesos realizados a la lectura y escritura de los registros CPLD. Incluye una fuente de poder externa de +5V, que se usa para energizar la tarjeta.

El voltaje regulado conmutado es de 1.4V para el núcleo del DSP y se suministra 3.3V a las entradas y salidas. Un regulador de voltaje separado energiza con 3.3V a la interface de expansión.

CCS se comunica con el DSK a través de un emulador JTAG embebido con una interface con el USB Host. El DSK también puede ser usado con un emulador externo a través del conector JTAG externo.

OPERACIÓN BÁSICA.

El DSK esta diseñado para trabajar con el CCS de TI con la versión especifica para trabajar con la tarjeta. El CCS se comunica con la tarjeta a través del emulador JTAG presente en la tarjeta. Para iniciar, sigua las instrucciones en la guía de instalación (ver Anexo GUIA DE INSTALACION) para instalar el CCS. Este brindara la instalación de las herramientas de desarrollo, documentación y controladores. Después de la instalación completa, debe seguir los siguientes pasos para correr CCS.

1.- Conecte la fuente de Poder del DSK.

2.- Conecte el DSK a su computadora por medio del cable USB.

3.- Ejecutar CCS dando clic en el icono en su escritorio.

ANEXO B.

GUIA DE INSTALACION.

El kit TMS320C6416 DSK contiene:

- Una tarjeta DSK C6416.
- Fuente de poder universal de +5V.
- Cable de poder AC.
- CD-ROM con Code Composer Studio para el DSK C6416.
- Manual de Referencia Técnica TMS320C6416 DSK.
- Cable USB.

REQUERIMIENTOS DE HARDWARE Y SOFTWARE.

Estos requerimientos son necesarios para instalar el Code Composer Studio IDE y soportar el puerto USB. Los requerimientos para la plataforma son:

CONFIGURACIÓN MÍNIMA DEL HARDWARE.

- Pentium de 233MHz o superior.
- 600 MB de disco duro disponible.
- Windows 98SE, 2000 o XP.
- 64MB de RAM.

- CD-ROM Drive

CONFIGURACIÓN DE HARDWARE RECOMENDADA.

- 128MB de RAM.
- Monitor SVGA (1024x768) color.
- Pentium 500MHz o superior.

El DSK de 720MHz es una versión actualizada del original del DSK de 600MHz, el cual incluye modificaciones de hardware y software.

INSTALACIÓN DEL DSK CODE COMPOSER STUDIO.

Antes de instalar el Software DSK asegúrese que su PC tiene USB y sistema operativo que soporte este puerto.

Para Windows 2000 y XP debemos instalar el CCS en la sesión de administrador. Para ejecutar el CCS en estos sistemas se requiere permiso de escritura en el registro. Si instalas el hardware seguir las instrucciones que vienen con el hardware. También antes de instalar el DSK CCS asegúrese de que el antivirus este deshabilitado. Este puede ser habilitado nuevamente cuando ejecutes el CCS.

1. Insertar en CD con CCS en el CD-ROM drive. Una ventana de instalación debe aparecer, si no ir a Explorador de Windows y ejecutar Setup.exe desde el CDROM.

2. Escoger una de las siguientes opciones de instalación:
    - Code Composer Studio. Instalación completa de CCS para el C6416DSK.
    - C6416DSK incluido en CCS v2.21. parche para una versión existente de CCS 2.21. Requiere CCS v2.21. Si se selecciona esta opción entonces ejecutar CCS 2 (6000) e importar la configuración "TMS320c6416dsk – 0x540".
3. Responder a los cuadros de dialogo mientras las instalación se ejecuta.
4. Deje el CD de CCS en el CD-ROM, este se usara para instalar el USB hardware. Reinicie el PC.

El DSK CCS v2.21 automáticamente configurara tu sistema con una configuración pre ajustada para el dispositivo C6416 DSK USB.

El proceso de instalación creara dos iconos en el escritorio:

C6416 DSK startup – C6416 DSK CCS

C6416 DSK Diagnostic Utility (ver anexo Diagnostico DSK)

CONECTANDO EL C6416 DSK AL PC:

1. Conecte el cable USB al PC o Laptop. Si conectas el cable USB a un USB HUB, debes estar seguro que el HUB este conectado a tu PC o Laptop.
2. Si usted planea instalar un micrófono, parlante o tarjeta de expansión, estas deben estar conectadas apropiadamente antes de activar el DSK.
3. Conecte el cable de poder AC a la fuente de poder.
4. Precaución: el cable de poder debe estar conectado a la fuente AC antes de conectar los 5V DC al DSK.
5. Conecte el cable de poder a la tarjeta.
6. Cuando el poder es aplicado a la tarjeta, el Power-ON self test (POST) se ejecutara. Los LEDS de cero al tres parpadearan. Cuando el POST esta completo todos los LEDS parpadearan y luego se quedaran encendidos.
7. En este punto su DSK esta funcionando y usted puede ahora terminar la instalación del USB.
8. Asegúrate de que el CCS CD-ROM DRIVE este instalado en su PC. Ahora conecte el cable USB en el DSK, luego de pocos minutos Windows detectara el nuevo hardware.

Siga las instrucciones en la pantalla y deje que Windows encuentre los controladores del USB sdusbemu.inf y sdusdemu.sys están en su CCS

CD-ROM. En Windows XP deberán encontrar los controladores automáticamente.

INICIANDO EL CODE COMPOSER STUDIO.

Para iniciar Code Composer Studio, haga doble clic en el icono de C6416 DSK CCS en escritorio del PC.

CORRIENDO EL TUTORIAL DEL CODE COMPOSER.

La ayuda en línea incluida en el C6416 DSK contiene información fundamental acerca del hardware y el software que contiene el Kit. Esta también contiene el tutorial que ayudara a iniciar su DSK y puede aprender acerca de sus características. Para acceder a la ayuda en línea y correr el tutorial, siga los siguientes pasos:

1. Inicie CCS (ignore este si CCS esta corriendo) con un doble clic en el icono de tu escritorio.
2. Abra el CCS seleccionando Help – Contents en el menú del CCS.
3. Abra la ayuda especifica del DSK C6416 abriendo el tópico etiquetado TMS320C6416 DSK. Este aparece en la parte última de dicho tópico.
4. Mire en la sección titulada Welcome to Your C6416 DSK, aquí encontrara el tutorial y otros materiales de introducción.

ANEXO C.

DIAGNOSTICO DSK.

La aplicación C6416 Diagnostics, realiza un chequeo General y Avanzado del DSK para verificar la posible existencia de problemas de hardware.

El chequeo *General* de los componentes de la tarjeta verifica:

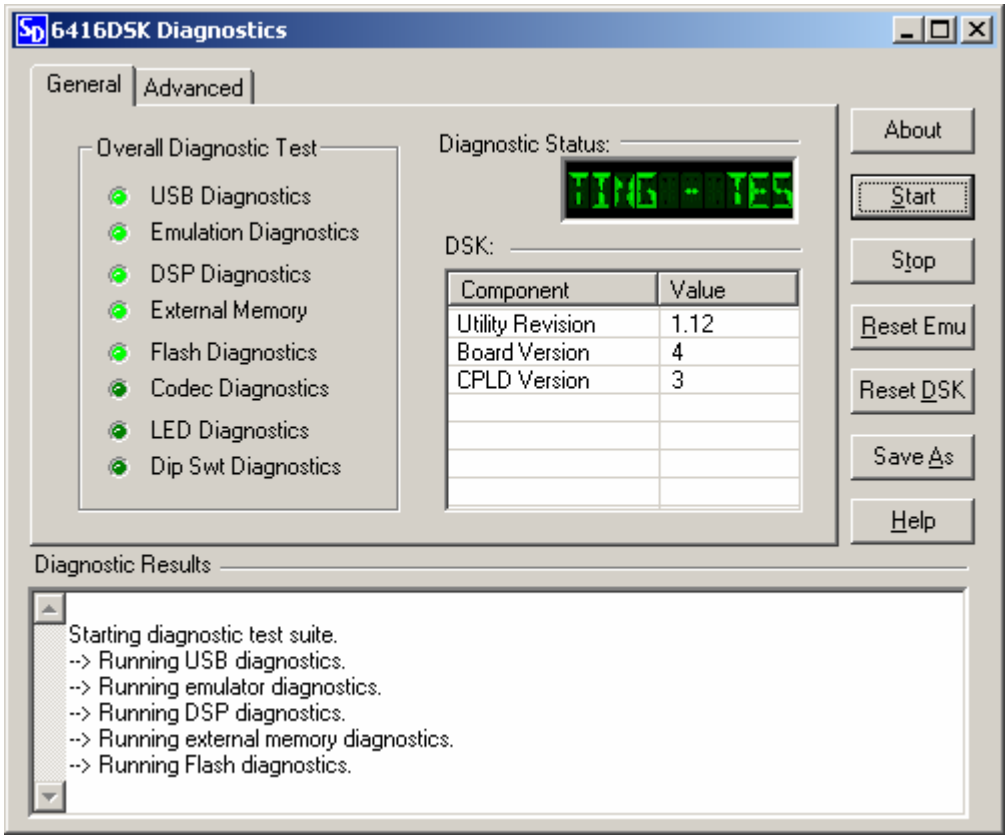


Figura C.1 Diagnostico General.

DIAGNOSTICO USB.- Detecta e inicializa e USB emulador de la tarjeta.

DIAGNOSTICO EMULACION.- Chequea que el emulador se comunica con el JTAG del DSP.

DIAGNOSTICO DSP.- Corre internamente un análisis en el núcleo del DSP y sus periféricos.

MEMORIA EXTERNA.- Ejecuta y direcciona análisis en la SDRAM externa.

DIAGNOSTICO FLASH.- Programa y verifica un patrón en la external Flash.

DIAGNOSTICO CODEC.- Genera un tono al parlante y línea de salida, muestra la línea de ingreso.

DIAGNOSTICO LED.- Corre un destello de los LED de izquierda a derecha que un usuario puede visualizar y verificar.

DIAGNOSTICO DIP SWITCH.- Muestra el valor de los Switch con luz en su correspondiente LED.

El chequeo *Avanzado*, contiene análisis que permite al usuario chequear la operación de un dispositivo con particular detalle.

Seleccionamos un determinado análisis al hacer click en un botón en el panel. Los botones de control se encuentran en el lado derecho de la aplicación, tiene la misma función que la función General**,** pero cada análisis lo realiza de manera individual.

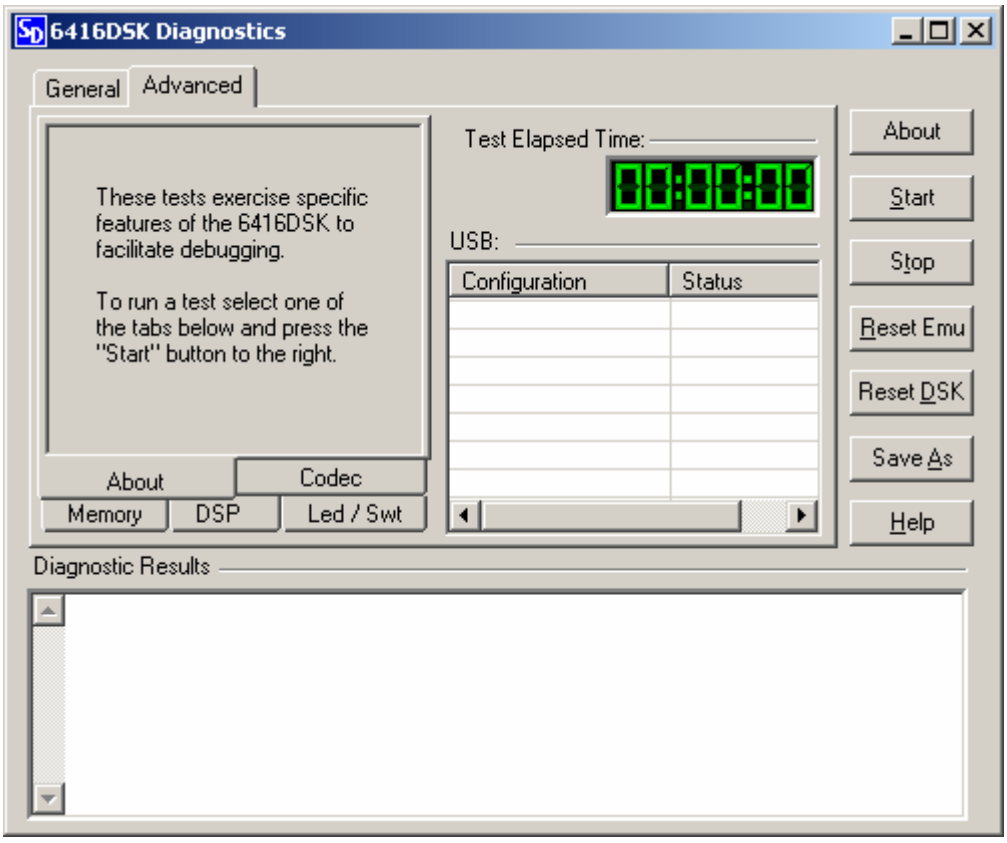


Figura C.2 Diagnostico Avanzado.

Dentro de los análisis que realiza tenemos:

CODEC.- Realiza el análisis del Codec inyectando una onda seno de 1 KHz. en el canal derecho (amarillo) y una onda seno de 2 k.o. en el canal izquierdo (azul).

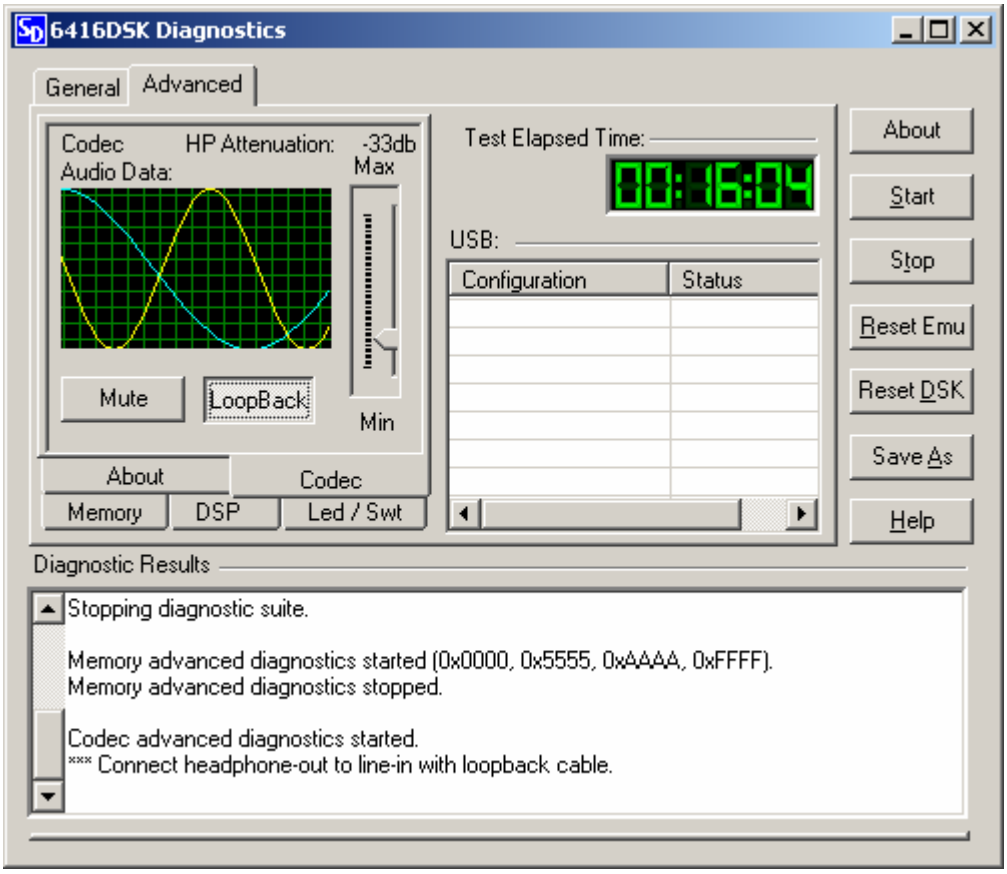


Figura C.3 Diagnostico Avanzado CODEC.

MEMORIA. - Analiza la memoria Interna y externa.

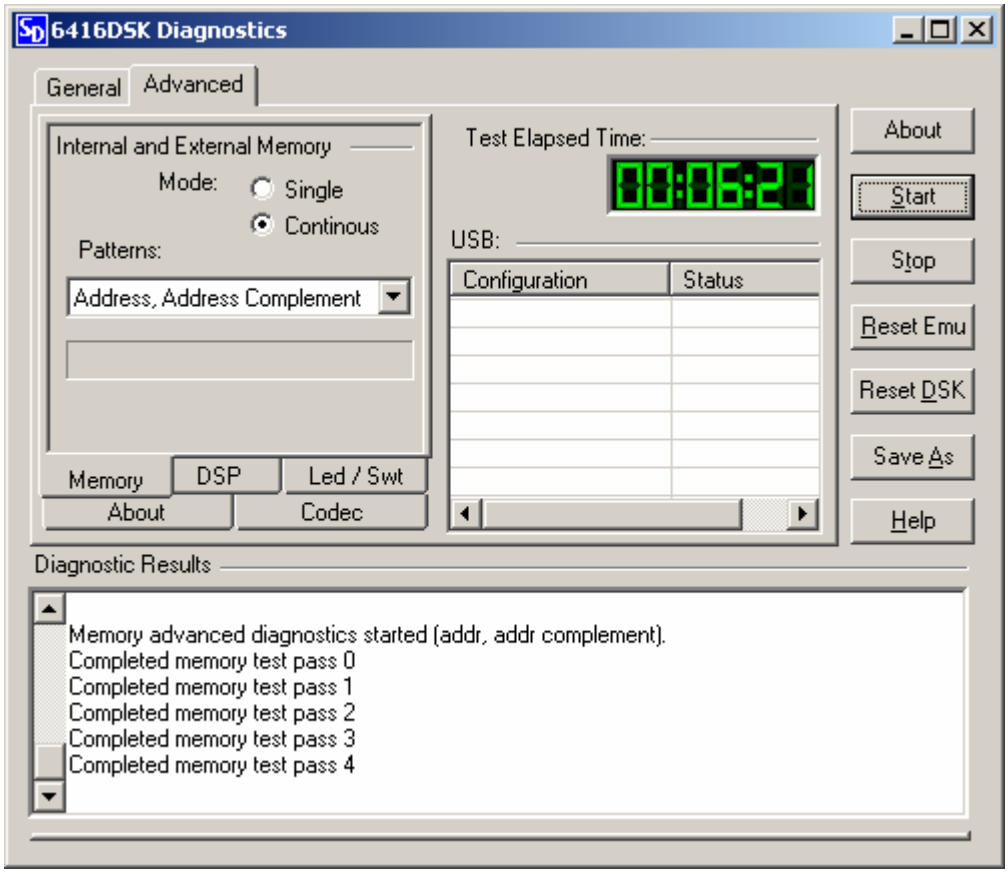


Figura C.4 Diagnostico Avanzado Memoria.

DSP.- Este diagnostico ejecuta cuatro análisis en el periférico del chip. El análisis temporizado muestra un LED parpadeante moviéndose de izquierda a derecha y regreso. El movimiento del LED permite al usuario verificar que el timer esta trabajando. La DMA usa el DMA del chip para copiar datos de un buffer a otro. El diagnostico McBSP pasa un conjunto de datos a través del puerto Serial 0 y 1 (McBSP es una puerto serial Texas Instruments) usando el mecanismo de internal loop-back y generador de frecuencia. Características de análisis es indicar si un problema con el DSP ocurre.

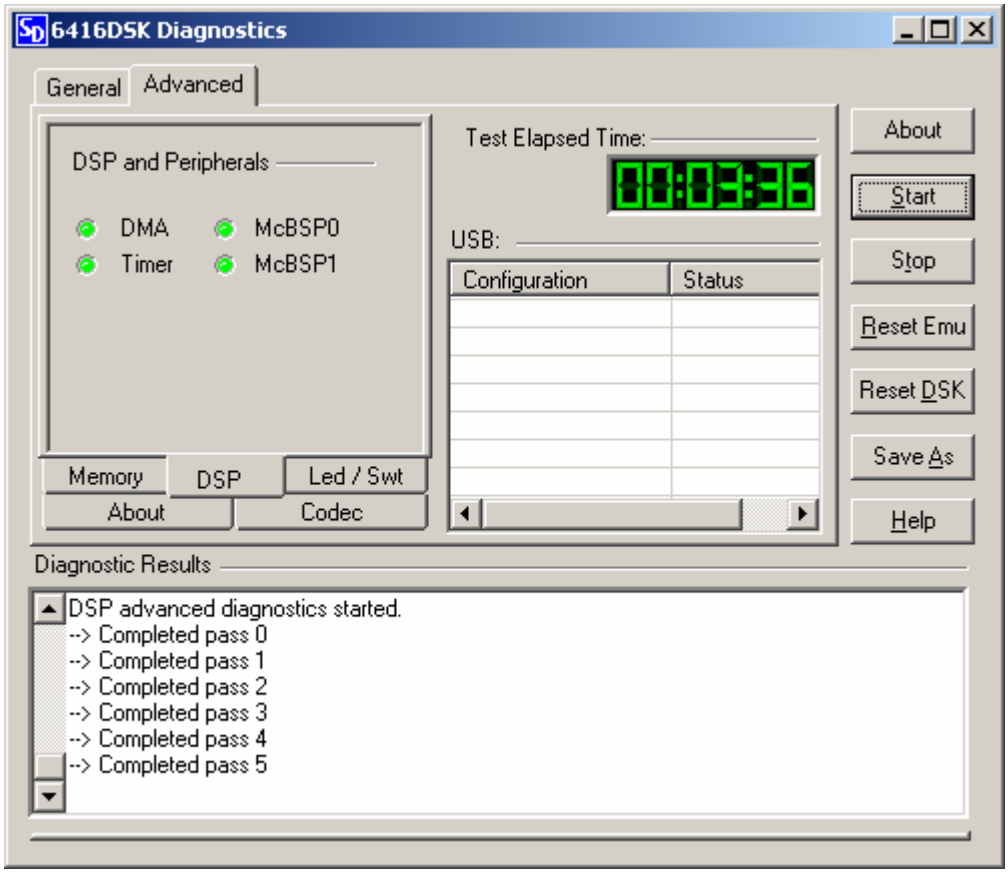


Figura C.5 Diagnostico Avanzado DSP.

LED/SWITCH.- Realiza el diagnostico del estado del DIP switch y constantemente actualiza el chequeo de los LED de la tarjeta. Los LED tienen la misma polaridad de los switches.

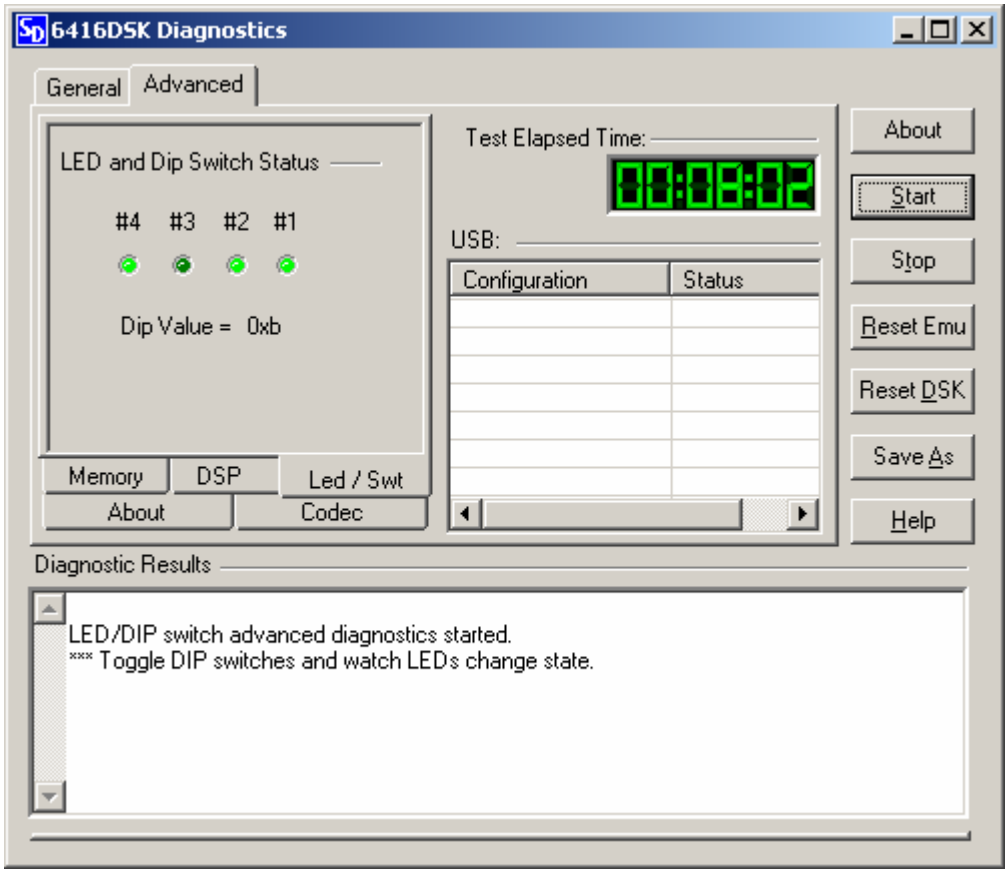


Figura C.6 Diagnostico Avanzado Led/Swt.

.

ANEXO D.

FILTROS.

INTRODUCCIÓN A FILTROS FIR.

Son innumerables los avances que ofrecen los filtros digitales, el filtro FIR puede garantizar características de “fase lineal”, ningún filtro analógico o IIR (Respuesta al Impulso Infinito) puede tener este éxito.

PROPIEDADES.

Coeficientes del Filtro:

$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k] \qquad (1)$$

Donde:

x[n] representa el ingreso del filtro.

b_k representa los coeficientes del filtro.

y[n] representa la salida del filtro.

N es el número de coeficientes del filtro (orden del filtro).

ESTRUCTURA DEL FILTRO.

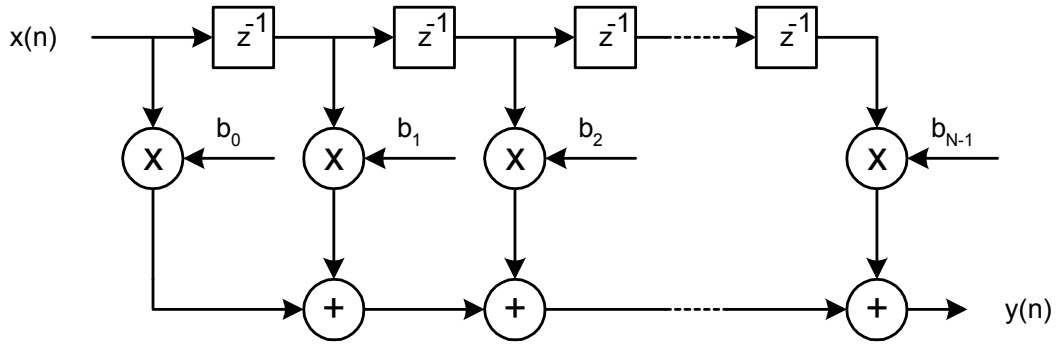


Figura D.1 Estructura Filtro FIR.

Si la señal x[n] es reemplazado en la ecuación de coeficientes por un impulso $\delta[n]$ entonces:

$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k]$$

Tenemos:

$$y[n] = \sum_{k=0}^{N-1} b_k \delta[n-k] \tag{2}$$

$$y[0] = b_0\delta[0] + b_1\delta[-1] + \cdots + b_k\delta[-N]$$

Si la señal x[n] es reemplazada por un impulso $\delta[n]$ entonces:

$$y[n] = b_0\delta[n] + b_1\delta[n-1] + \cdots + b_k\delta[n-N] \tag{3}$$

$$\delta[n-k] = \begin{cases} 1 \text{ for } n = k \\ 0 \text{ for } n \neq k \end{cases} \tag{4}$$

Tenemos finalmente que:

$$b_0 = h[0]$$
$$b_1 = h[1]$$
$$\vdots$$
$$b_k = h[k]$$

Podemos concluir que:

$$b_k = h[k] \tag{5}$$

Los coeficientes de un filtro son los mismos que cuando se realiza un muestreo esperando una respuesta al impulso unitario $\delta[n]$.

RESPUESTA EN FRECUENCIA DE UN FILTRO FIR.

PROCEDIMIENTO DE DISEÑO.

Por tomar la transformada z de h[n], H(z):

$$H(z) = \sum_{n=0}^{N-1} h[n] z^{-n} \tag{6}$$

Reemplazando z por $e^{-j\omega}$ para encontrar la respuesta de frecuencia orienta a:

$$H(z)\big|_{z=e^{j\omega}} = H(\omega) = \sum_{n=0}^{N-1} h[n] e^{-jn\omega}$$

Desde $e^{-j2\pi k} = 1$ entonces:

$$H(\omega + 2\pi) = \sum_{n=0}^{N-1} h[n] e^{-jn(\omega+2\pi)} = \sum_{n=0}^{N-1} h[n] e^{-jn\omega}$$

Por lo tanto:

$$H(\omega + 2k\pi) = H(\omega) \qquad (7)$$

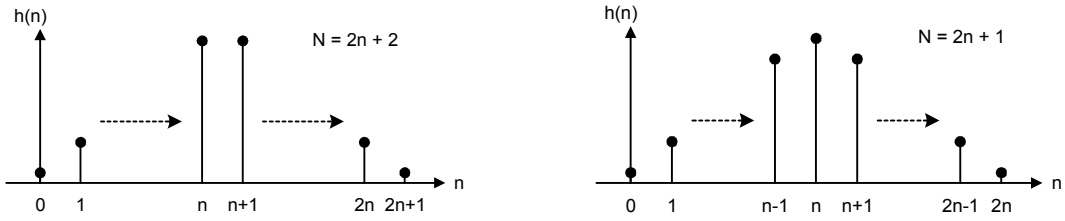
El Filtro FIR tiene una respuesta de frecuencia periódica y el periodo es 2π.

RESPUESTA DE FRECUENCIA:

$$H(\omega + 2\pi) = H(\omega)$$

FASE LINEAL DE UN FILTRO FIR.

Un filtro causal FIR cuya respuesta al impulso es simétrica esta garantizando para tener una repuesta de fase lineal.



Simetría Par **Simetría Impar**

Figura D.2 Coeficientes Filtro FIR.

Un filtro causal FIR cuya respuesta al impulso es simétrica (h[n] = h[N-1-n] para n = 0, 1, …, N-1) esta garantizado para tener una respuesta de fase lineal.

PROCEDIMIENTO DE DISEÑO.

Para un completo diseño e implementación de un filtro estos cinco pasos son necesarios:

1. Especificaciones del Filtro.
2. Cálculo de Coeficientes.
3. Selección de Estructura.
4. Simulación (Opcional).
5. Implementación.

1. ESPECIFICACIONES DEL FILTRO.

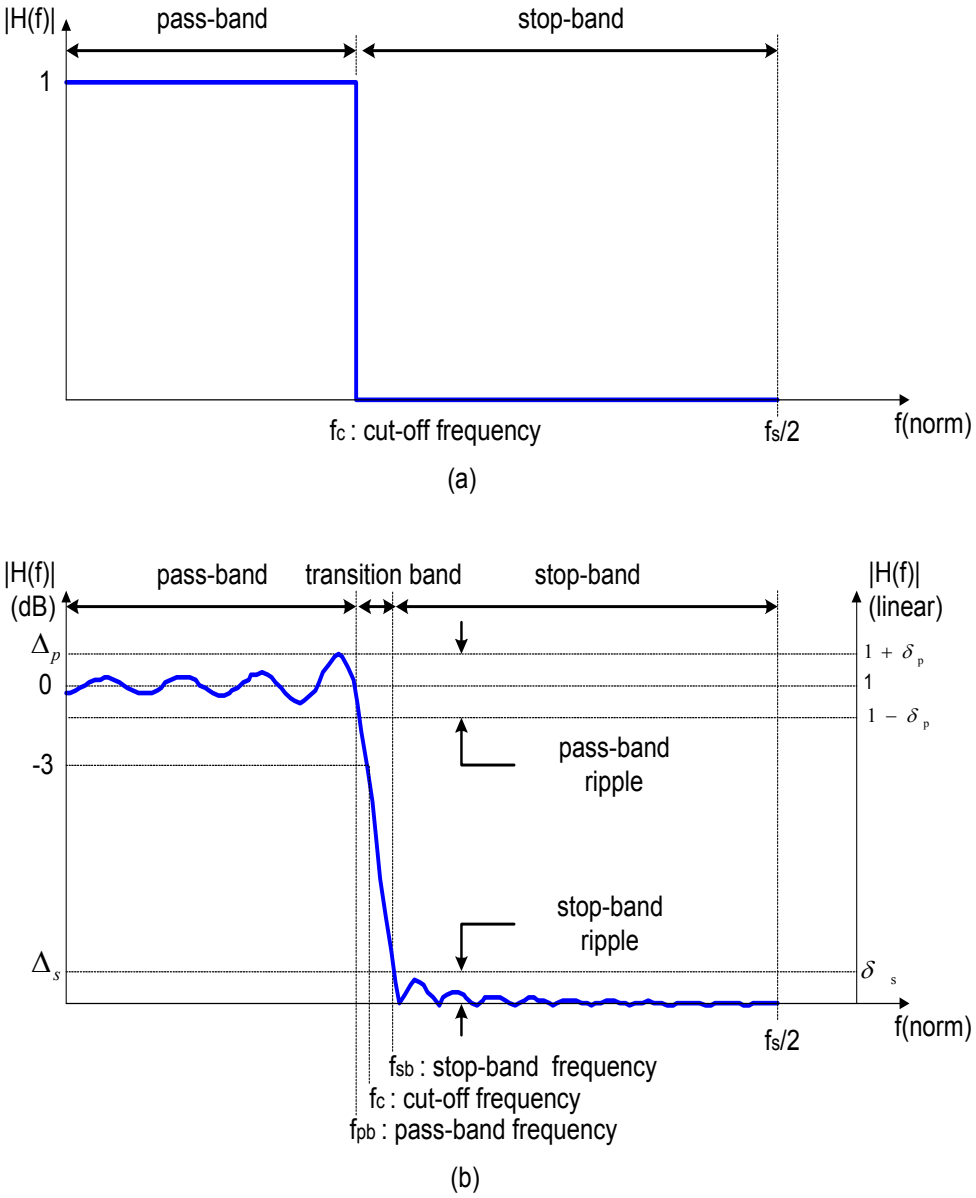


Figura D.3 Respuesta en Frecuencia Filtro FIR.

2. CÁLCULO DE COEFICIENTES.

Estos son los diferentes métodos disponibles para el cálculo de coeficientes:

- Método Ventana.
- Muestreo de Frecuencia.
- Parks-McClellan.

MÉTODO VENTANA

El primer estado de este método es el cálculo de coeficientes del filtro ideal.

El cálculo es como sigue:

$$h_d(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H(\omega)e^{j\omega n}d\omega \tag{8}$$

$$= \frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} 1\cdot e^{j\omega n}d\omega$$

$$= \begin{cases} \dfrac{2f_c \sin(n\omega_c)}{n\omega_c} & \text{for n} \neq 0 \\ 2f_c & \text{for n} = 0 \end{cases}$$

Segundo estado de este método es seleccionar una función ventana basada en especificaciones pasa banda y atenuación, entonces se determina la longitud del filtro basados en el ancho requerido por la banda de transición.

Window Type	**Normalised Transition Width (Δf(Hz))**	**Passband Ripple(dB)**	**Stopband Attenuation (dB)**
Rectangular	$\frac{0.9}{N}$	0.7416	21
Hanning	$\frac{3.1}{N}$	0.0546	44
Hamming	$\frac{3.3}{N}$	0.0194	53
Blackman	$\frac{5.5}{N}$	0.0017	74
Kaiser	$\frac{2.93}{N} \rightarrow \beta = 4.54$ $\frac{5.71}{N} \rightarrow \beta = 8.96$	0.0274 0.000275	50 90

Tabla C.1 Tipos de función Ventana.

Usando Ventana Hamming:

$$N = \frac{3.3}{\Delta f} = \frac{3.3}{(1.2 - 1.4)kHz} \cdot 8kHz = 132$$

MÉTODO VENTANA.

El tercer estado es para calcular el set de truncamientos o coeficientes de respuesta al impulso windowed, h[n]:

$$h(n) = h_d(n) \cdot W(n) \tag{9}$$

para

$$-\frac{N-1}{2} \le n \le \frac{N-1}{2} \quad \text{para N} = \text{impar}$$
$$-\frac{N}{2} \le n \le \frac{N}{2} \quad \text{para N} = \text{par}$$

Donde:

$$W(n) = 0.54 + 0.46\cos\left(\frac{2\pi n}{N}\right)$$
$$= 0.54 + 0.46\cos\left(\frac{2\pi n}{133}\right) \quad \text{para} \quad -66 \le n \le 66$$

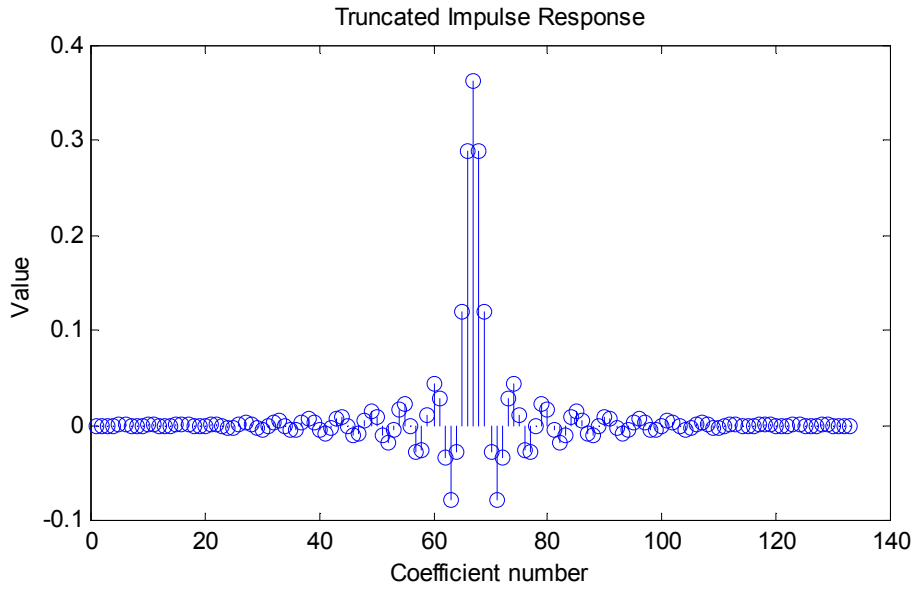


Figura D.4 Respuesta al Impulso Unitario FIR.

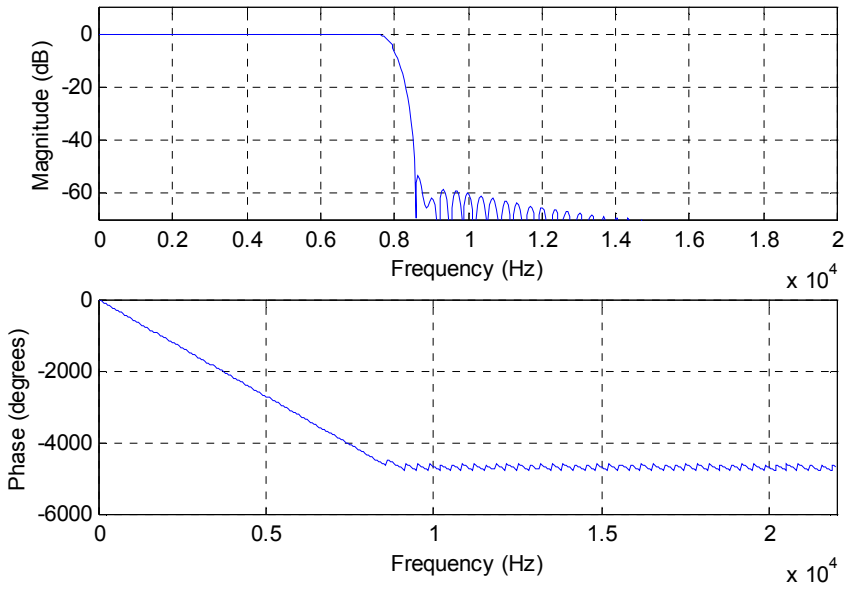


Figura D.5 Respuesta en Magnitud y Fase FIR.

3. SELECCIÓN DE ESTRUCTURA.

- **ESTRUCTURA FIR FORMA-DIRECTA**:

$$H(z) = \sum_{k=0}^{N-1} b_k z^{-k} \tag{10}$$

$$Y(z) = H(z) \cdot X(z) \tag{11}$$

$$y(n) = b_0 x(n) + b_1 x(n-1) + \ldots + b_{N-1} x(n-N+1) \tag{12}$$

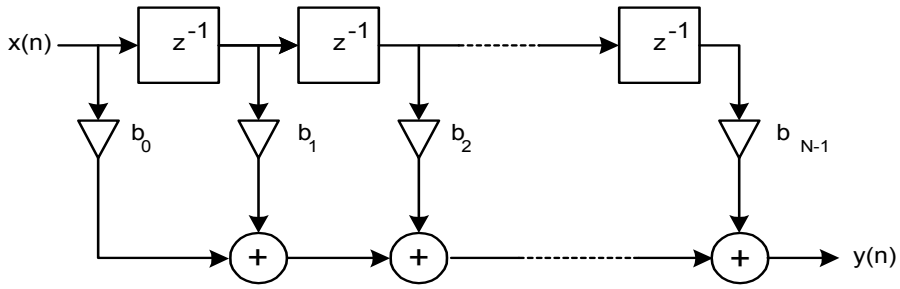


Figura D.6 Diagrama Digital FIR Forma-Directa.

Estructura para un Filtro FIR Forma-Directa:

$$H(z)=\sum_{k=0}^{N-1} b_k z^{-k}$$

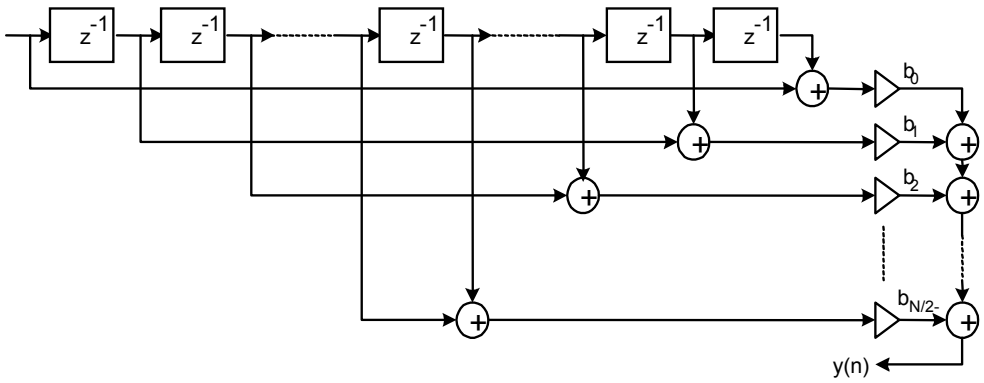
ESTRUCTURA FASE LINEAL :

- N par, Figura (a):

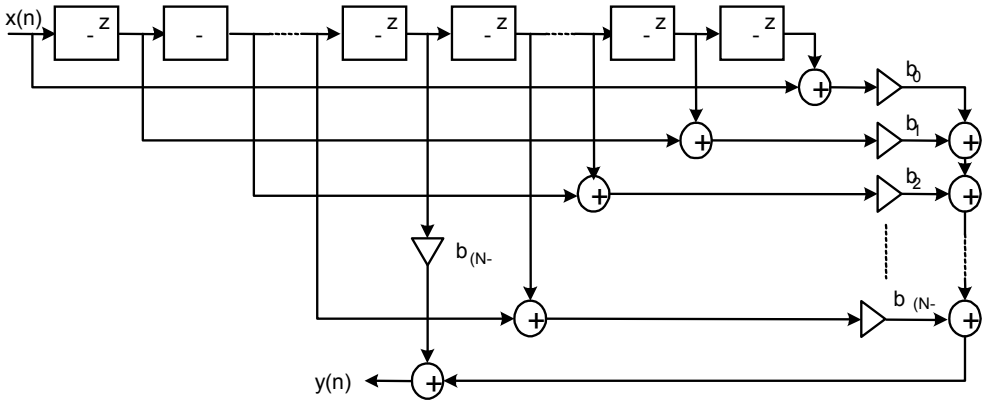
$$H(z)=\sum_{k=0}^{\frac{N}{2}-1} b_k \left(z^{-k}+z^{N-k-1}\right) \tag{13}$$

- N impar, Figura (b):

$$H(z)=\sum_{k=0}^{\frac{N-1}{2}} b_k \left(z^{-k}+z^{N-k-1}\right)+b_{\frac{N-1}{2}} z^{-\frac{N-1}{2}} \tag{14}$$



(a)



(b)

Figura D.7 Estructura Digital Fase Lineal.

ESTRUCTURA CASCADA:

$$
\begin{aligned}
H(z) &= \sum_{k=0}^{N-1} b_k z^{-k} = b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_{N-1} z^{-(N-1)} \\
&= b_0 \left[1 + \frac{b_1}{b_0} z^{-1} + \frac{b_2}{b_0} z^{-2} + \ldots + \frac{b_{N-1}}{b_0} z^{-(N-1)} \right] \\
&= b_0 \prod_{k=1}^{M} \left(1 + b_{k,1} z^{-1} + b_{k,2} z^{-2} \right)
\end{aligned}
\tag{15}
$$

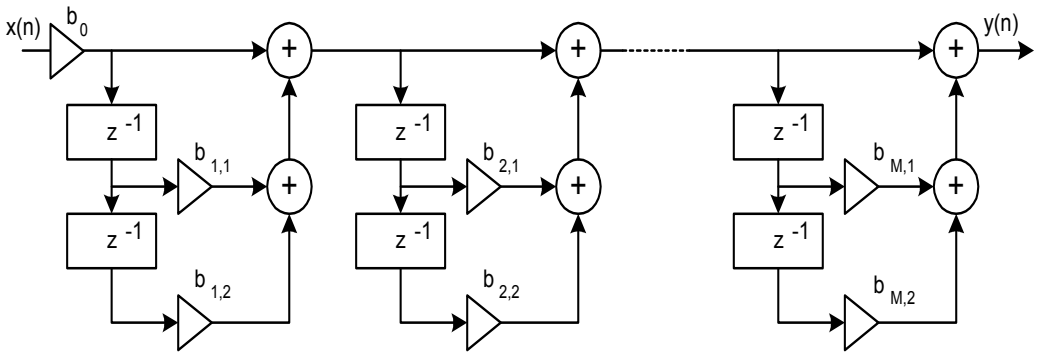


Figura D.8 Estructura Digital en Cascada.

INTRODUCCIÓN A FILTROS IIR.

Los Filtros Respuesta al Impulso Infinito (IIR) son los utilizados cuando:

- Velocidad es de suma importancia.
- Fase no-lineal es aceptable.

Filtros IIR son calculados más eficientemente que los filtros FIR, debido a que requieren pocos coeficientes, debido a que estos usan polos y realimentación. Sin embargo la realimentación puede resultar apropiada en los filtros inestables, si los coeficientes se apartan de sus valores reales.

PROPIEDADES.

La ecuación general de un filtro IIR puede ser expresada como:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + \ldots + a_M z^{-M}} = \frac{\sum_{k=0}^{N} b_k z^{-k}}{1 + \sum_{k=1}^{M} a_k z^{-k}} \tag{16}$$

Donde a_k y b_k son los coeficientes del filtro.

La función de transferencia puede ser factorizada y obtenemos:

$$H(z) = k \frac{(z - z_1)(z - z_2)\cdots(z - z_N)}{(z - p_1)(z - p_2)\cdots(z - p_N)} = \frac{Y(z)}{X(z)} \tag{17}$$

Donde: z_1, z_2,…, z_N son los ceros, p_1, p_2, …, p_N son los polos. Para realizar la implementación de la anterior ecuación necesitamos la ecuación diferencial:

$$y[n] = \sum_{k=0}^{\infty} h[k]x[n-k] = \sum_{k=0}^{N} b[k]x[n-k] + \sum_{k=1}^{M} a[k]y[n-k] \tag{18}$$

ESTRUCTURA DEL FILTRO IIR.

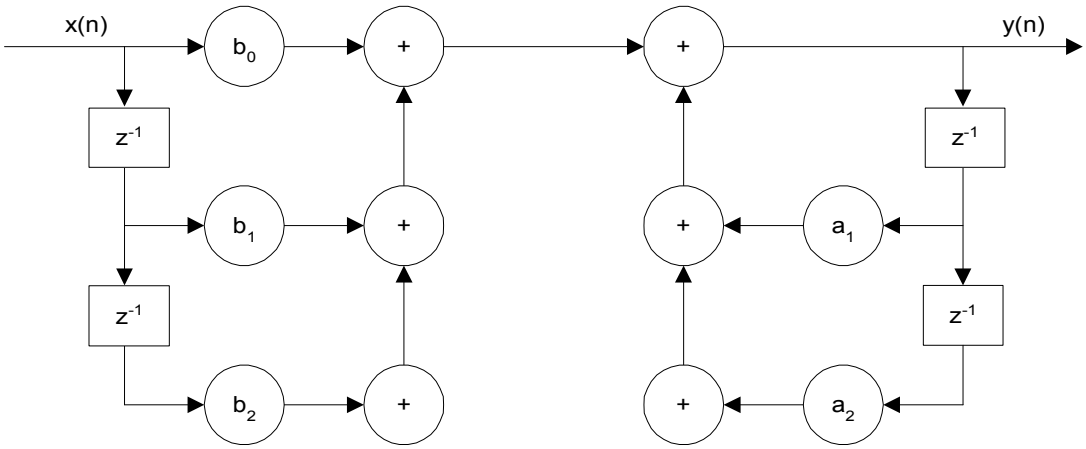


Figura D.9 Estructura Filtro IIR para N=M=2.

PROCEDIMIENTO DE DISEÑO.

Para un completo diseño e implementación de un filtro estos cinco pasos son necesarios:

1. Especificaciones del Filtro.
2. Cálculo de Coeficientes.
3. Selección de Estructura.
4. Simulación (Opcional).
5. Implementación.

ANEXO E.

PRUEBAS.

Resultados de la generación de códigos Walsh tomados de la variable Codigo_Walsh del Workspace de Matlab.

WALSH 0	WALSH 32	WALSH 63
1	1	1
1	1	-1
1	1	-1
1	1	1
1	1	-1
1	1	1
1	1	1
1	1	-1
1	1	-1
1	1	1
1	1	1
1	1	-1
1	1	1
1	1	-1
1	1	-1
1	1	1
1	1	-1
1	1	1
1	1	1
1	1	-1
1	1	1
1	1	-1
1	1	-1
1	1	1
1	1	1
1	1	-1
1	1	-1
1	1	1

1	1	-1
1	1	1
1	1	1
1	1	-1
1	-1	-1
1	-1	1
1	-1	1
1	-1	-1
1	-1	1
1	-1	-1
1	-1	-1
1	-1	1
1	-1	1
1	-1	-1
1	-1	-1
1	-1	1
1	-1	-1
1	-1	1
1	-1	1
1	-1	-1
1	-1	1
1	-1	-1
1	-1	-1
1	-1	1
1	-1	-1
1	-1	1
1	-1	1
1	-1	-1
1	-1	-1
1	-1	1
1	-1	1
1	-1	-1
1	-1	1
1	-1	-1
1	-1	-1
1	-1	1

Tabla E.1 Variable Codigo_Walsh.

TABLA DE CÓDIGOS WALSH FORMATO BIPOLAR.

W 0	W 1	W 2	W 3	W 4	W 5	W 6	W 7	W 8	W 9	W 10	W 11	W 12	W 13	W 14	W 15	W 16	W 17	W 18	W 19	W 20	W 21	W 22	W 23	W 24	W 25	W 26	W 27	W 28	W 29	W 30	W 31	W 32	W 33
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1
1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1
1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1
1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1
1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1
1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1
1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1
1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1
1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1
1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1
1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1
1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1
1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1
1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	1	1
1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	1
1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	1	-1
1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	1	1
1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	1	-1
1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	1	1
1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	1	-1
1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	1	1
1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	1	-1
1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	1	1
1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	1	-1
1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	1	1
1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	1	-1
1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	1	1
1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	1	-1
1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	1	1
1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	1	-1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1	-1
1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	1
1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	-1	-1
1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	-1	1
1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	-1	-1
1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	-1	1
1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	-1	-1
1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	-1	1
1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	-1	1
1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	-1	-1
1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	-1	1
1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1	-1
1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	-1	1
1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	-1	-1
1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	-1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1
1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1
1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1
1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1
1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1
1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1
1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1
1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	-1	-1
1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	-1	1
1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	-1	-1
1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1
1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1
1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1
1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1
1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1

W 34	W 35	W 36	W 37	W 38	W 39	W 40	W 41	W 42	W 43	W 44	W 45	W 46	W 47	W 48	W 49	W 50	W 51	W 52	W 53	W 54	W 55	W 56	W 57	W 58	W 59	W 60	W 61	W 62	W 63
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1
-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1
-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1
1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1
1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1
-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1
-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1
1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1
-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1
-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1
1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1
1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1
-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1
-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1
-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1
-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1
1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1
1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1
-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1
-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1
1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1
1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1
-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1
-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1
1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	-1	-1	-1	-1
1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1
-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1
-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1
1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1
1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1
-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1
-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1
1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1
1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1
-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1
-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1
1	1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1
1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1
-1	-1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	-1	-1	-1	-1
-1	1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1
1	1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1
1	-1	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1
1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1
1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1
-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1
-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1
1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1
1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1
-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	1	-1	1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1
1	1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1
1	-1	-1	1	1	-1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1
-1	-1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1
-1	1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1
1	1	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1
1	-1	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1

Tabla E.2 Códigos Walsh bipolar.

ANEXO F.

CODIGOS FUENTES.

RAMAL_I_Q_FINAL_HARDWARE.c

```
#include "RAMAL_I_Q_FINAL_HARDWARE.h"
#include "RAMAL_I_Q_FINAL_HARDWARE_private.h"
#pragma DATA_ALIGN(RAMAL_I_Q_FINAL_HARDWARE_B,8)
BlockIO_RAMAL_I_Q_FINAL_HARDWARE
RAMAL_I_Q_FINAL_HARDWARE_B;
#pragma DATA_ALIGN(RAMAL_I_Q_FINAL_HARDWARE_DWork,8)
D_Work_RAMAL_I_Q_FINAL_HARDWARE
RAMAL_I_Q_FINAL_HARDWARE_DWork;
rtModel_RAMAL_I_Q_FINAL_HARDWARE
RAMAL_I_Q_FINAL_HARDWARE_M_;
rtModel_RAMAL_I_Q_FINAL_HARDWARE
*RAMAL_I_Q_FINAL_HARDWARE_M =
  &RAMAL_I_Q_FINAL_HARDWARE_M_;
static void RAMAL_I_Q_FINAL_HARDWARE_output(int_T tid)
{  int32_T rtb_Switch1;
  boolean_T rtb_Q0;
  boolean_T rtb_Selector;
  rtb_Switch1 = 15 - (( *(volatile uint8_T *)(0x60000000) ) >> 4);
  rtb_Q0 = RAMAL_I_Q_FINAL_HARDWARE_DWork.Q0_DSTATE;
  if(rtb_Switch1 > RAMAL_I_Q_FINAL_HARDWARE_P.Selector_Threshold) {
    rtb_Selector = RAMAL_I_Q_FINAL_HARDWARE_DWork.I0_DSTATE;
  } else {
    rtb_Selector = rtb_Q0;
  }
  *(volatile uint8_T *)(0x60000000) = (unsigned char)(rtb_Selector);
  RAMAL_I_Q_FINAL_HARDWARE_B.I1 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.I1_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.I10 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.I10_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.I11 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.I11_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.I12 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.I12_DSTATE;
```

```
  RAMAL_I_Q_FINAL_HARDWARE_B.I13 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.I13_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.I14 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.I14_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.I2 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.I2_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.I3 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.I3_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.I4 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.I4_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.I5 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.I5_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.I6 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.I6_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.I7 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.I7_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.I8 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.I8_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.I9 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.I9_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.LogicalOperator =

(boolean_T)((boolean_T)((boolean_T)((boolean_T)((boolean_T)(RAMAL_I_Q
_FINAL_HARDWARE_DWork.I0_DSTATE
    ^ RAMAL_I_Q_FINAL_HARDWARE_B.I5) ^
RAMAL_I_Q_FINAL_HARDWARE_B.I7) ^
    RAMAL_I_Q_FINAL_HARDWARE_B.I8) ^
RAMAL_I_Q_FINAL_HARDWARE_B.I9) ^
    RAMAL_I_Q_FINAL_HARDWARE_B.I13);
  RAMAL_I_Q_FINAL_HARDWARE_B.Q3 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.Q3_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.Q4 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.Q4_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.Q5 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.Q5_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.Q6 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.Q6_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.Q10 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.Q10_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.Q11 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.Q11_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.Q12 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.Q12_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.LogicalOperator1 =
```

```
(boolean_T)((boolean_T)((boolean_T)((boolean_T)((boolean_T)((boolean_T)(
(boolean_T)(rtb_Q0
    ^ RAMAL_I_Q_FINAL_HARDWARE_B.Q3) ^
RAMAL_I_Q_FINAL_HARDWARE_B.Q4) ^
    RAMAL_I_Q_FINAL_HARDWARE_B.Q5) ^
RAMAL_I_Q_FINAL_HARDWARE_B.Q6) ^
    RAMAL_I_Q_FINAL_HARDWARE_B.Q10) ^
RAMAL_I_Q_FINAL_HARDWARE_B.Q11) ^
    RAMAL_I_Q_FINAL_HARDWARE_B.Q12);
  RAMAL_I_Q_FINAL_HARDWARE_B.Q1 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.Q1_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.Q13 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.Q13_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.Q14 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.Q14_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.Q2 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.Q2_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.Q7 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.Q7_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.Q8 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.Q8_DSTATE;
  RAMAL_I_Q_FINAL_HARDWARE_B.Q9 =
RAMAL_I_Q_FINAL_HARDWARE_DWork.Q9_DSTATE;
}
static void RAMAL_I_Q_FINAL_HARDWARE_update(int_T tid)
{
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I0_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.I1;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q0_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.Q1;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I1_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.I2;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I10_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.I11;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I11_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.I12;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I12_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.I13;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I13_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.I14;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I14_DSTATE =
  RAMAL_I_Q_FINAL_HARDWARE_B.LogicalOperator;
```

```
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I2_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.I3;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I3_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.I4;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I4_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.I5;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I5_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.I6;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I6_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.I7;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I7_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.I8;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I8_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.I9;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I9_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.I10;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q3_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.Q4;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q4_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.Q5;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q5_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.Q6;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q6_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.Q7;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q10_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.Q11;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q11_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.Q12;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q12_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.Q13;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q1_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.Q2;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q13_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.Q14;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q14_DSTATE =
  RAMAL_I_Q_FINAL_HARDWARE_B.LogicalOperator1;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q2_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.Q3;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q7_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.Q8;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q8_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.Q9;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q9_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_B.Q10;
```

```
  if(!(++RAMAL_I_Q_FINAL_HARDWARE_M->Timing.clockTick0))
  ++RAMAL_I_Q_FINAL_HARDWARE_M->Timing.clockTickH0;
  RAMAL_I_Q_FINAL_HARDWARE_M->Timing.t[0] =
    RAMAL_I_Q_FINAL_HARDWARE_M->Timing.clockTick0 *
    RAMAL_I_Q_FINAL_HARDWARE_M->Timing.stepSize0 +
    RAMAL_I_Q_FINAL_HARDWARE_M->Timing.clockTickH0 *
    RAMAL_I_Q_FINAL_HARDWARE_M->Timing.stepSize0 * 4294967296.0;
}
void RAMAL_I_Q_FINAL_HARDWARE_initialize(boolean_T firstTime)
{
  if (firstTime) {
    (void)memset((char_T *)RAMAL_I_Q_FINAL_HARDWARE_M, 0,
     sizeof(rtModel_RAMAL_I_Q_FINAL_HARDWARE));
    {
      int_T *mdlTsMap = RAMAL_I_Q_FINAL_HARDWARE_M-
>Timing.sampleTimeTaskIDArray;
      int_T i;
      for(i = 0; i < 1; i++) {
        mdlTsMap[i] = i;
      }
      RAMAL_I_Q_FINAL_HARDWARE_M->Timing.sampleTimeTaskIDPtr =
(&mdlTsMap[0]);
      RAMAL_I_Q_FINAL_HARDWARE_M->Timing.sampleTimes =
        (&RAMAL_I_Q_FINAL_HARDWARE_M->Timing.sampleTimesArray[0]);
      RAMAL_I_Q_FINAL_HARDWARE_M->Timing.offsetTimes =
        (&RAMAL_I_Q_FINAL_HARDWARE_M->Timing.offsetTimesArray[0]);
      RAMAL_I_Q_FINAL_HARDWARE_M->Timing.sampleTimes[0] = (0.2);
      RAMAL_I_Q_FINAL_HARDWARE_M->Timing.offsetTimes[0] = (0.0);
    }
    rtmSetTPtr(RAMAL_I_Q_FINAL_HARDWARE_M,
     &RAMAL_I_Q_FINAL_HARDWARE_M->Timing.tArray[0]);
    {
      int_T *mdlSampleHits = RAMAL_I_Q_FINAL_HARDWARE_M-
>Timing.sampleHitArray;
      int_T i;
      for(i = 0; i < 1; i++) {
        mdlSampleHits[i] = 1;
      }
      RAMAL_I_Q_FINAL_HARDWARE_M->Timing.sampleHits =
(&mdlSampleHits[0]);
    }
    RAMAL_I_Q_FINAL_HARDWARE_M->Timing.stepSize0 = 0.2;
    RAMAL_I_Q_FINAL_HARDWARE_M->solverInfoPtr =
      (&RAMAL_I_Q_FINAL_HARDWARE_M->solverInfo);
```

```
    RAMAL_I_Q_FINAL_HARDWARE_M->Timing.stepSize = (0.2);
    rtsiSetFixedStepSize(&RAMAL_I_Q_FINAL_HARDWARE_M->solverInfo,
0.2);
    rtsiSetSolverMode(&RAMAL_I_Q_FINAL_HARDWARE_M->solverInfo,
     SOLVER_MODE_SINGLETASKING);
    {
      void *b = (void *) &RAMAL_I_Q_FINAL_HARDWARE_B;
      RAMAL_I_Q_FINAL_HARDWARE_M->ModelData.blockIO = (b);
      (void)memset(b, 0, sizeof(BlockIO_RAMAL_I_Q_FINAL_HARDWARE));
    }
    RAMAL_I_Q_FINAL_HARDWARE_M->ModelData.defaultParam =
((real_T *)
      &RAMAL_I_Q_FINAL_HARDWARE_P);
    RAMAL_I_Q_FINAL_HARDWARE_M->Work.dwork = ((void *)
      &RAMAL_I_Q_FINAL_HARDWARE_DWork);
    (void)memset((char_T *) &RAMAL_I_Q_FINAL_HARDWARE_DWork, 0,
     sizeof(D_Work_RAMAL_I_Q_FINAL_HARDWARE));
  }
}
void RAMAL_I_Q_FINAL_HARDWARE_terminate(void)
{
}
void MdlOutputs(int_T tid) {
  RAMAL_I_Q_FINAL_HARDWARE_output(tid);
}
void MdlUpdate(int_T tid) {
  RAMAL_I_Q_FINAL_HARDWARE_update(tid);
}
void MdlInitializeSizes(void) {
  RAMAL_I_Q_FINAL_HARDWARE_M->Sizes.numContStates = (0); /*
Number of continuous states */
  RAMAL_I_Q_FINAL_HARDWARE_M->Sizes.numY = (0); /* Number of
model outputs */
  RAMAL_I_Q_FINAL_HARDWARE_M->Sizes.numU = (0); /* Number of
model inputs */
  RAMAL_I_Q_FINAL_HARDWARE_M->Sizes.sysDirFeedThru = (0); /* The
model is not direct feedthrough */
  RAMAL_I_Q_FINAL_HARDWARE_M->Sizes.numSampTimes = (1); /*
Number of sample times */
  RAMAL_I_Q_FINAL_HARDWARE_M->Sizes.numBlocks = (35); /* Number
of blocks */
  RAMAL_I_Q_FINAL_HARDWARE_M->Sizes.numBlockIO = (30); /* Number
of block outputs */
```

```
  RAMAL_I_Q_FINAL_HARDWARE_M->Sizes.numBlockPrms = (31); /* Sum
of parameter "widths" */
}
void MdlInitializeSampleTimes(void) {
}
void MdlStart(void) {
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I0_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.I0_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q0_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.Q0_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I1_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.I1_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I10_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.I10_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I11_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.I11_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I12_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.I12_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I13_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.I13_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I14_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.I14_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I2_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.I2_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I3_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.I3_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I4_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.I4_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I5_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.I5_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I6_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.I6_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I7_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.I7_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I8_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.I8_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.I9_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.I9_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q3_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.Q3_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q4_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.Q4_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q5_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.Q5_X0;
```

```
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q6_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.Q6_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q10_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.Q10_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q11_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.Q11_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q12_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.Q12_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q1_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.Q1_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q13_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.Q13_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q14_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.Q14_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q2_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.Q2_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q7_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.Q7_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q8_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.Q8_X0;
  RAMAL_I_Q_FINAL_HARDWARE_DWork.Q9_DSTATE =
RAMAL_I_Q_FINAL_HARDWARE_P.Q9_X0;
}
rtModel_RAMAL_I_Q_FINAL_HARDWARE
*RAMAL_I_Q_FINAL_HARDWARE(void) {
  RAMAL_I_Q_FINAL_HARDWARE_initialize(1);
  return RAMAL_I_Q_FINAL_HARDWARE_M;
}
void MdlTerminate(void) {
  RAMAL_I_Q_FINAL_HARDWARE_terminate();
}
```

WALSH_FINAL_HARDWARE.C

```
#include "Walsh_final_hardware.h"
#include "Walsh_final_hardware_private.h"
#pragma DATA_ALIGN(Walsh_final_hardware_DWork,8)
D_Work_Walsh_final_hardware Walsh_final_hardware_DWork;
rtModel_Walsh_final_hardware Walsh_final_hardware_M_;
rtModel_Walsh_final_hardware *Walsh_final_hardware_M =
&Walsh_final_hardware_M_;
static void Walsh_final_hardware_output(int_T tid)
{
```

```
  real_T rtb_FromWorkspace;
  real_T rtb_temp1;
  int32_T rtb_Switch1;
  {
    int_T currIndex =
Walsh_final_hardware_DWork.FromWorkspace_IWORK.PrevIndex+1;
    real_T *pDataValues = (real_T *)
      Walsh_final_hardware_DWork.FromWorkspace_PWORK.DataPtr;
    if (currIndex >= 64) {
      currIndex = 0;
    }
    if (currIndex < 64) {
      pDataValues += currIndex;
      rtb_temp1 = *pDataValues;
    } else {
      pDataValues += (63);
      rtb_temp1 = *pDataValues;
    }
    Walsh_final_hardware_DWork.FromWorkspace_IWORK.PrevIndex =
currIndex;
  }
  rtb_Switch1 = 15 - (( *(volatile uint8_T *)(0x60000000) ) >> 4);
  {
    int_T currIndex =
      Walsh_final_hardware_DWork.FromWorkspace_IWORK_k.PrevIndex+1;
    real_T *pDataValues = (real_T *)
      Walsh_final_hardware_DWork.FromWorkspace_PWORK_c.DataPtr;
    if (currIndex >= 64) {
      currIndex = 0;
    }
    if (currIndex < 64) {
      pDataValues += currIndex;
      rtb_FromWorkspace = *pDataValues;
    } else {
      pDataValues += (63);
      rtb_FromWorkspace = *pDataValues;
    }
    Walsh_final_hardware_DWork.FromWorkspace_IWORK_k.PrevIndex =
currIndex;
  }
  if(rtb_Switch1 > Walsh_final_hardware_P.Selector_Threshold) {
    rtb_temp1 = rtb_temp1;
  } else {
    rtb_temp1 = rtb_FromWorkspace;
```

```
  }
  *(volatile uint8_T *)(0x60000000) = (unsigned char)(rtb_temp1);
}
static void Walsh_final_hardware_update(int_T tid)
{
  if(!(++Walsh_final_hardware_M->Timing.clockTick0))
  ++Walsh_final_hardware_M->Timing.clockTickH0;
  Walsh_final_hardware_M->Timing.t[0] =
    Walsh_final_hardware_M->Timing.clockTick0 *
    Walsh_final_hardware_M->Timing.stepSize0 +
    Walsh_final_hardware_M->Timing.clockTickH0 *
    Walsh_final_hardware_M->Timing.stepSize0 * 4294967296.0;
}
void Walsh_final_hardware_initialize(boolean_T firstTime)
{
  if (firstTime) {
    (void)memset((char_T *)Walsh_final_hardware_M, 0,
     sizeof(rtModel_Walsh_final_hardware));
    {
      int_T *mdlTsMap = Walsh_final_hardware_M-
>Timing.sampleTimeTaskIDArray;
      int_T i;
      for(i = 0; i < 1; i++) {
        mdlTsMap[i] = i;
      }
      Walsh_final_hardware_M->Timing.sampleTimeTaskIDPtr =
(&mdlTsMap[0]);
      Walsh_final_hardware_M->Timing.sampleTimes =
        (&Walsh_final_hardware_M->Timing.sampleTimesArray[0]);
      Walsh_final_hardware_M->Timing.offsetTimes =
        (&Walsh_final_hardware_M->Timing.offsetTimesArray[0]);
      Walsh_final_hardware_M->Timing.sampleTimes[0] = (0.2);
      Walsh_final_hardware_M->Timing.offsetTimes[0] = (0.0);
    }
    rtmSetTPtr(Walsh_final_hardware_M,
     &Walsh_final_hardware_M->Timing.tArray[0]);
    {
      int_T *mdlSampleHits = Walsh_final_hardware_M-
>Timing.sampleHitArray;
      int_T i;
      for(i = 0; i < 1; i++) {
        mdlSampleHits[i] = 1;
      }
      Walsh_final_hardware_M->Timing.sampleHits = (&mdlSampleHits[0]);
```

```
    }
    Walsh_final_hardware_M->Timing.stepSize0 = 0.2;
    Walsh_final_hardware_M->solverInfoPtr =
      (&Walsh_final_hardware_M->solverInfo);
    Walsh_final_hardware_M->Timing.stepSize = (0.2);
    rtsiSetFixedStepSize(&Walsh_final_hardware_M->solverInfo, 0.2);
    rtsiSetSolverMode(&Walsh_final_hardware_M->solverInfo,
     SOLVER_MODE_SINGLETASKING);
    Walsh_final_hardware_M->ModelData.defaultParam = ((real_T *)
      &Walsh_final_hardware_P);
    Walsh_final_hardware_M->Work.dwork = ((void *)
&Walsh_final_hardware_DWork);
    (void)memset((char_T *) &Walsh_final_hardware_DWork, 0,
     sizeof(D_Work_Walsh_final_hardware));
  }
}
void Walsh_final_hardware_terminate(void)
{
}
void MdlOutputs(int_T tid) {
  Walsh_final_hardware_output(tid);
}
void MdlUpdate(int_T tid) {
  Walsh_final_hardware_update(tid);
}
void MdlInitializeSizes(void) {
  Walsh_final_hardware_M->Sizes.numContStates = (0); /* Number of
continuous states */
  Walsh_final_hardware_M->Sizes.numY = (0); /* Number of model outputs */
  Walsh_final_hardware_M->Sizes.numU = (0); /* Number of model inputs */
  Walsh_final_hardware_M->Sizes.sysDirFeedThru = (0); /* The model is not
direct feedthrough */
  Walsh_final_hardware_M->Sizes.numSampTimes = (1); /* Number of
sample times */
  Walsh_final_hardware_M->Sizes.numBlocks = (5); /* Number of blocks */
  Walsh_final_hardware_M->Sizes.numBlockIO = (0); /* Number of block
outputs */
  Walsh_final_hardware_M->Sizes.numBlockPrms = (1); /* Sum of parameter
"widths" */
}
void MdlInitializeSampleTimes(void) {
}
void MdlStart(void) {
  {
```

```
    static real_T pDataValues[] = { 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
      1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
      1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
      1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
      1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0 };
    Walsh_final_hardware_DWork.FromWorkspace_PWORK.TimePtr = (void
*) 0;
    Walsh_final_hardware_DWork.FromWorkspace_PWORK.DataPtr = (void
*)
      pDataValues;
    Walsh_final_hardware_DWork.FromWorkspace_IWORK.PrevIndex = -1;
  }
  {
    static real_T pDataValues[] = { 1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, -1.0,
      -1.0, 1.0, 1.0, -1.0, 1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, -1.0, 1.0,
      -1.0, -1.0, 1.0, 1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, -1.0, -1.0, 1.0,
      1.0, -1.0, 1.0, -1.0, -1.0, 1.0, 1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0,
      -1.0, 1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, -1.0, -1.0, 1.0, 1.0, -1.0,
      1.0, -1.0, -1.0, 1.0 };
    Walsh_final_hardware_DWork.FromWorkspace_PWORK_c.TimePtr =
(void *) 0;
    Walsh_final_hardware_DWork.FromWorkspace_PWORK_c.DataPtr =
(void *)
      pDataValues;
    Walsh_final_hardware_DWork.FromWorkspace_IWORK_k.PrevIndex = -1;
  }
}
rtModel_Walsh_final_hardware *Walsh_final_hardware(void) {
  Walsh_final_hardware_initialize(1);
  return Walsh_final_hardware_M;
}
void MdlTerminate(void) {
  Walsh_final_hardware_terminate();
}
```

▪▪

BIBLIOGRAFIA.

1. GROE JOHN B., LARSON LAWRENCE E., CDMA Mobile Radio Design, Artech House, 2000.
2. T. KEITH BLANKENSHIP, Design and Implementation of a Pilot Signal Scanning Receiver for CDMA Personal Communication Services Systems, Tesis de Maestría, May 1998.
3. YANG SAMUEL C., CDMA RF System Engineering, Artech House, 1998
4. BURNS PAUL, Software Defined Radio for 3G, Artech House, 2002.
5. GARG VIJAY K., IS-95 CDMA and cdma2000 Cellular/PCS Systems Implementation, Prentice Hall PTR, 2000.
6. KEHTARNAVAZ NASSER, Real-Time Digital Signal Processing Based on the TMS320C6000, Elsevier, 2005.
7. 3RD GENERATION PARTNERSHIP PROJECT 2, Physical Layer Standard for cdma2000 Spread Spectrum Systems, 3GPP2, 2002.
8. DAHNOUN NAIM, DSP Implementation using the TMS320C6000 DSP Platform, Prentice Hall, Primera edición, 2000.
9. THE MATHWORKS, Simulink Model-Based and System-Based Design: Using Simulink Version 5, The MathWorks, 2002.
10. THE MATHWORKS, Real-Time Workshop For Use with Simulink: Getting Started Version 5, The MathWorks, 2002.

11. SPECTRUM DIGITAL, TMS320C6416 DSK Technical Reference, Spectrum Digital, 2003.
12. Texas Instruments, TMS320C64X/C64X+ DSP CPU and Instruction Set Reference Guide, 2006.
13. Texas Instruments, TMS320C6000 DSP Peripherals Overview Reference Guide, 2007.
14. Physical Layer Standard for cdma2000 Spread Spectrum Systems Release B, http://www.3gpp2.org/Public_html/specs/C.S0002-D_v2.0_051006.pdf, 2002.