ESCUELA POLITÉCTICA DEL EJÉRCITO

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

“IMPLEMENTACIÓN DE LA ETAPA DE RECEPCIÓN DE UN SISTEMA DE COMUNICACIONES UTILIZANDO LA TECNOLOGÍA FPGA”

GABRIEL DARÍO FIERRO PIÑEIROS

DIRECTOR: ING. JULIO LARCO

CODIRECTOR: ING. BYRON NAVAS

SANGOLQUÍ - ECUADOR

2010

RESUMEN

Tanto la modulación BPSK como la modulación QPSK son hoy en día muy utilizadas en diferentes tecnologías de telecomunicaciones debido a su bajo costo de implementación y su simplicidad.

Debido a la fluctuación del nivel de potencia de una señal en un entorno móvil, es difícil mantener los parámetros de calidad por encima de un umbral para todo instante de tiempo por combatir este fenómeno se han desarrollado herramientas como los códigos convolucionles que permiten identificar los errores que se produjeron y corregirlos.

Para la implementación tanto de la modulación como la codificación de canal en hardware los dispositivos Field Programmable Gate Array(FPGA) son una opción altamente viable porque tienen la ventaja que posibilita modificar una determinada acción cambiando el código y también se reconfigura el FPGA cambiando la forma de interconexión de las celdas; todo esto sin hacer cambios en el Hardware.

Sin embargo, los métodos de programación actuales han limitado la accesibilidad a subsistemas de procesamiento de señales basados en FPGAs. Esta situación ha cambiado en los últimos años con la aparición de una nueva clase de programación en FPGA basados en el alto nivel de modelado de MATLAB y SIMULINK.

System Generator es una plataforma software que usa las herramientas de The MathWorks MATLAB/Simulink para representar una visión abstracta de alto nivel del sistema de comunicaciones, y que automáticamente genera el código VHDL de la función de DSP desarrollada usando los más optimizados LogiCOREs de Xilinx, asegurándonos así que se ha producido la implementación más eficiente del diseño.

DEDICATORIA

A Dios, a mis padres y hermanos que han estado presentes en mi vida en todo momento y que me han dando la fuerza y confianza para hacer de cualquier dificultad una oportunidad.

AGRADECIMIENTO

A mi padre por su ejemplo de tenacidad, responsabilidad e integridad y A mí madre por su fuerza, valor y la cualidad de pensar siempre en el bienestar de los demás.

A mis queridas tías por que han estado conmigo y me han apoyado en los momentos más importantes de mi vida.

A mis compañeros y amigos de la Universidad por hacer de cada proyecto un momento para fortalecer nuestra amistad. En especial a Karina por su preocupación y compañía.

A mi Director y Codirector de tesis, Julio Larco y Byron Navas por ofrecerme toda su ayuda y conocimientos para el desarrollo de este documento.

A mis amigos de siempre por ser fuente de inspiración en cada paso que doy y estar siempre a mi lado.

PRÓLOGO

Para una mejor organización y comprensión, este proyecto se ha orientado de forma metódica y siguiendo un orden claro y conciso que consiste en estudio teórico un estudio de las herramientas de desarrollo, elaboración del diseño, simulación e implementación.

Este proyecto pretende desarrollar en una Field Programmable Gate Array (FPGA), la etapa de recepción de un sistema de comunicaciones utilizando una modulación BPSK *y* QPSK con códigos convolucionales. Para ello, inicialmente se realiza un estudio sobre la modulación BPSK, QPSK y los códigos convolucionales para familiarizarse con todas sus características y situar el objetivo final.

Una parte importante del trabajo, es la familiarización con el entorno de programación de FPGAs, que utilizan un lenguaje HDL. Para el desarrollo del sistema se utiliza System Generator de Xilinx, que se ejecuta sobre Matlab/Simulink, y que mediante bloques predefinidos, permite la programación gráfica del lenguaje HDL.

Para la implementación se dispone de la placa Spartan-3E Starter Kit, que entre otros dispositivos está formada por un FPGA Spartan-3E a 50 MHz, convertidores DAC (*Digital Analog Converter*) y ADC (*Analog Digital Converter*), un puerto VGA, dos puertos RS- 232, LEDs, un display, interruptores y botones, etc.

Finalmente se realizan simulaciones para comprobar el funcionamiento de cada etapa del receptor y se compara las diferencias cuando el sistema es implementado en el FPGA.

INDICE DE CONTENIDOS

CAPÍTULO I

INDICE DE TABLAS

INDICE DE FIGURAS

GLOSARIO

ADC	Conversor Análogo Digital.
ASIC	Circuitos Integrados para Aplicaciones Específicas.
ASK	Modulación por Desplazamiento de Amplitud.
ASSP	Producto de Estándar Específico de Aplicación.
BPSK	Modulación por Desplazamiento de Fase Binaria.
CAD:	Diseño Asistido por computadora u Ordenador.
CAE	Ingeniería Asistida por computadora u Ordenador.
CDMA	Acceso Múltiple por División de Código.
CLB	Bloque Lógico Programable.
CMOS	Estructuras Semiconductor-Óxido-Metal Complementarias.
CORE	Núcleo.
BlockSet	Conjunto de bloques.
CPLD	Dispositivo Lógico Programable Complejo.
DAC	Conversor Digital Análogo.
DDS Compiler	Compilador de un Sintetizador Digital Directo.

DRAM	Memoria de Acceso Aleatorio Dinámico.
DSP	Procesamiento Digital de Señales.
EDA	Automatización de Diseño Electrónico.
EEPROM	Memoria de solo Lectura Programable y Borrable Eléctricamente.
EPROM	Memoria de solo Lectura Programable y Borrable.
FEC	Corrección de Errores hacia Adelante.
Flip-Flop	Es un circuito oscilador de onda cuadrada, capaz de permanecer en un estado determinado o en el contrario durante un tiempo indefinido.
FPGA	Matriz de puertas programables.
FSK	Modulación por Desplazamiento de Frecuencia.
HDL	Lenguaje de Descripción de Hardware.
IDE	Ambiente de diseño Integrado.
XSG	Xilinx System Generator.
IEEE	Instituto de Ingenieros Eléctricos y Electrónicos.
ISO	Organización Internacional para la Estandarización.
ISO 14443	Es un estándar internacional relacionado con las tarjetas de identificación electrónicas, en especial las tarjetas inteligentes.

LUT	Tabla de Consulta.
PLD	Dispositivo Lógico Programable.
PROM	Memoria de sólo Lectura Programable.
PSK	Modulación por desplazamiento de Fase.
QPSK	Modulación por Desplazamiento de Fase en Cuadratura.
RAM	Memoria de Acceso Aleatorio.
RFID	Identificación por Radiofrecuencia.
ROM	Memoria de solo Lectura.
RTL	Nivel de la Transferencia de Registro.
SRAM	Memoria Estática de acceso Aleatorio.
VERILOG	Lenguaje de Descripción de Hardware.
VHDL	Lenguaje textual de alto nivel que se utiliza para la descripción del hardware de los sistemas digitales.

CAPITULO I

1. MODULACIÓN DIGITAL

Las señales de datos de terminales digitales y dispositivos afines, generalmente no se transmiten a gran distancia en la forma de señal de banda de base, es decir, tal como se generan, sino que se transmiten en forma de una señal modulada en forma analógica, en efecto, los impulsos o dígitos binarios modulan una portadora sinusoidal cuya frecuencia es compatible con el medio de transmisión utilizado; este tipo de transmisión se denomina "Transmisión Binaria mediante Portadora Modulada o transmisión en banda ancha".[1]

Modular básicamente consiste en convertir una señal digital en una señal analógica que irá variando su amplitud, frecuencia, fase o bien amplitud y fase conjuntamente, según los valores que vaya tomando la señal digital de información. De esta manera, aparecen distintas técnicas de modulación de señales digitales según el tipo de modulación empleado.

1.1 Modulación PSK

PSK es un esquema de modulación digital extenso. La modulación PSK se caracteriza porque la fase de la señal portadora representa cada símbolo de información de la señal moduladora. PSK es ampliamente usado en la industria de las comunicaciones y es la base para otros tipos de modulación como DS-CDMA[1]y OFDM[2].

[1] Acceso múltiple por división de código en secuencia directa
[2] Multiplexación por División de Frecuencias Ortogonales

1.2 Trasmisor BPSK

El transmisor por desplazamiento de fase binaria permite obtener dos fases de salida con una sola portadora. Una fase de salida representa un 1 lógico y la otra un 0 lógico. Si la señal de entrada en el modulador cambia de estado, la fase de la portadora de salida se desplaza entre ángulos que están 180°fuera de fase. La es de la forma.

$$s(t) = A\sum_{k=0}^{\infty} d_k h_e(t - kT_S)\cos(2\pi f_c) \qquad (1.1)$$

donde d_k son los valores de los símbolos binarios asignados a la entrada binaria, $h_e(t)$ es la respuesta impulsional de filtro conformador o filtro transmisor, f_c frecuencia portadora y el tiempo de símbolo, que por ser una modulación binaria coincide con el tiempo de bit T_b.

El esquema del modulador utilizado para generar la señal es el mostrado en la Figura 1.1. Puede observarse que los bits pasan a través de un circuito transformador de niveles denominado *mapping circuit*, que asigna a los bits el valor de amplitud (fase) y a continuación se transforma en una señal de manera que permite su transmisión adecuada en el canal. [2]

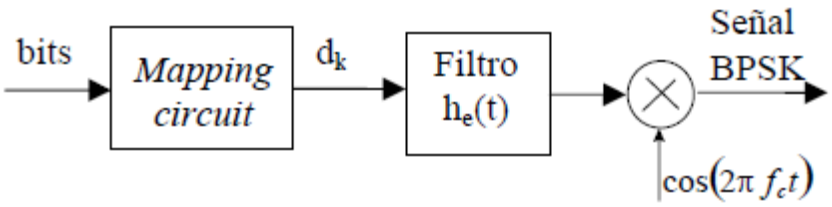


Figura 1.1 Esquema de un modulador BPSK

En la Figura 1.2 se muestra la forma de onda de una señal BPSK con filtro conformador rectangular. Pueden observarse los cambios de fase de la portadora cuando cambia el símbolo transmitido.

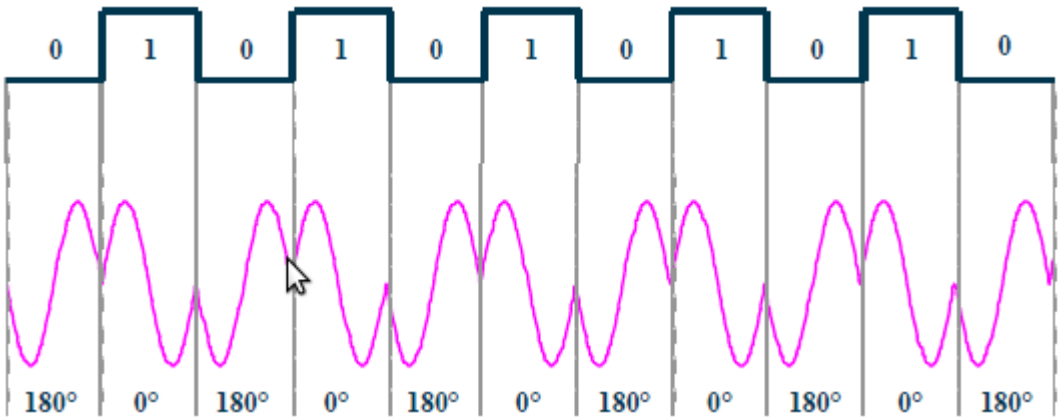


Figura 1.2 Forma de onda de una modulación BPSK

La figura 1.3 muestra el diagrama fasorial y diagrama de la constelación para un modulador BPSK.

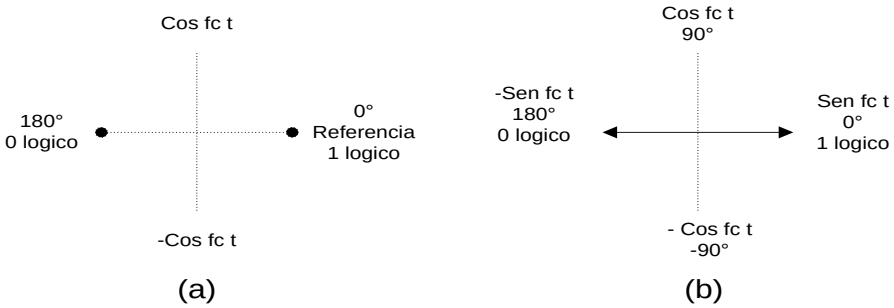


Figura 1.3 (a) Diagrama de Constelación. (b) Diagrama Fasorial.

1.2.1 Ancho de banda para BPSK

En un modulador BPSK la señal de entrada de la portadora se multiplica por los datos binarios, si +1V se asigna a un 1 lógico y -1V se asigna a un 0 lógico, la portadora (Sen f_c t) se multiplica ya sea por **+** o por **-**. En consecuencia la señal de salida es +1 Sen f_c t o –1 Sen f_c t ; la primera representa una señal que

está en fase con el oscilador local, la segunda una señal que esta 180° fuera de fase, cada vez que cambia la condición de entrada cambia la salida. En consecuencia para BPSK la razón de cambio de salida (BAUDIO) es igual a la razón de cambio de entrada (bits/s) y el ancho de banda más grande ocurre cuando los datos binarios a la entrada son una secuencia alternativa de unos y ceros.

La frecuencia fundamental (f_m) de una secuencia alternativa de bits 1/0 es igual a la mitad de la razón de bits ($f_b/2$).

La Figura 1.4 muestra la fase de salida para una forma de onda BPSK el espectro de salida de un modulador BPSK es solo una señal de doble banda lateral con portadora suprimida, en donde las frecuencias laterales superiores e inferiores están separadas de la frecuencia portadora por un valor igual a la mitad de la razón de BIT. En consecuencia el mínimo ancho de banda requerido para permitir el peor caso de la señal de salida del BPSK es igual a la razón de BIT de entrada.

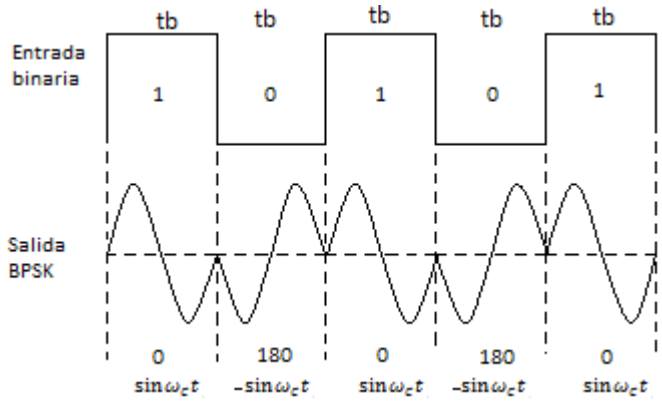


Figura 1.4 Relación de la fase de salida con el tiempo para un modulador BPSK

1.2.2 Receptor BPSK

Existen dos tipos de demodulación, denominados *detección coherente* y *detección no coherente*. Cuando el receptor conoce la fase de la señal portadora recibida y la utiliza en la demodulación, la detección se denomina coherente. Si por el contrario el receptor desconoce la fase de la portadora, el proceso se

denomina no coherente. En comunicaciones digitales los términos *demodulación* y *detección* se usan a menudo indistintamente, aunque demodulación enfatiza la eliminación de la portadora y detección incluye el proceso de la decisión del símbolo transmitido. La ventaja de la demodulación no coherente es que no necesita estimar el valor de la fase de la señal recibida, lo que redunda en una mayor sencillez de implementación. Esto se logra a costa de un incremento en la probabilidad de error. El conocimiento de la fase de llegada de la portadora de la señal recibida, necesario para poder llevar a cabo una demodulación coherente, requiere el uso de un circuito de estimación de dicha fase.

El receptor BPSK está conformado básicamente por tres componentes como se muestra en la Figura 1.5. Un circuito de recuperación de portadora que detecta y genera una señal portadora que es coherente, tanto en frecuencia como en fase con la señal portadora del transmisor, se lo puede suprimir si el receptor conoce estas magnitudes. Un *mapping circuit*, que realiza el producto entre la señal BPSK y la portadora recuperada y un filtro pasa bajos que separa los datos binarios recuperados de la señal demodulada compleja.

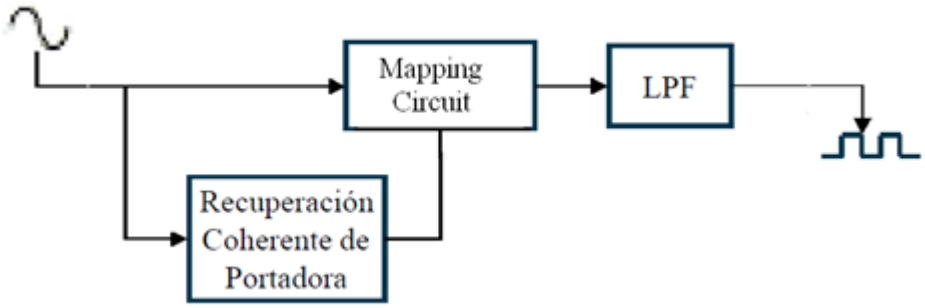


Figura 1.5 Componentes receptor BPSK

1.3 Modulación QPSK

QPSK *(Quadrature Phase Shift Keying)* es un esquema de modulación lineal digital donde la información transportada por la señal transmitida está contenida en la fase. La fase de la portadora toma uno de cuatro valores igualmente espaciados (0, π/2, π y 3π/2), cada uno corresponde a un único símbolo.

Ya que la entrada digital a un modulador QPSK es una señal binaria (base2) para producir cuatro condiciones diferentes de entrada, se necesita más de un solo BIT de entrada, con dos bits hay 4 posibles condiciones: 00,01,10 y 11.En consecuencia con QPSK los datos de entrada binarios se combinan en grupos de 2 bits llamados dibits**.** Cada dibit genera una de las cuatro fases de entrada posibles por tanto para un dibit (2 bits) introducido al modulador ocurre un solo cambio a la salida.

1.3.1 Transmisor QPSK

En la Figura 1.6 dos bits se introducen al derivador de bits, después que ambos bits han sido introducidos en forma serial salen simultáneamente en forma paralela. Un BIT se dirige al canal I y el otro al canal Q. El BIT I modula una portadora que está en fase con el oscilador de referencia (de ahí el nombre I para el canal en fase) y el BIT Q modula una portadora que esta 90° fuera de fase o en cuadratura con la portadora de referencia (de ahí el nombre de Q para el canal de cuadratura). El circuito serie paralelo transforma un tren de datos en dos trenes de bits en paralelo denominados símbolos donde la frecuencia del símbolo es igual a la frecuencia del bit dividido para 2, el espectro de la señal banda base de las derivaciones I e Q tiene la misma forma que el espectro de la señal de entrada, pero el ancho de los lóbulos es la mitad del de entrada. Puede verse que una vez que un dibit ha sido derivado a los canales I y Q la operación es igual que en el modulador BPSK. Se podría decir que en esencia un Modulador QPSK son dos moduladores BPSK en paralelo siendo un uno lógico igual a + 1 V y un cero lógico igual -1 V. En el *mapping circuit* del canal I las fases de salida son, $+\cos w_c t$, $-\cos w_c t$ mientras que en el *mapping circuit* de el canal Q las salidas de fase son $+\operatorname{sen} w_c t$, $-\operatorname{sen} w_c t$

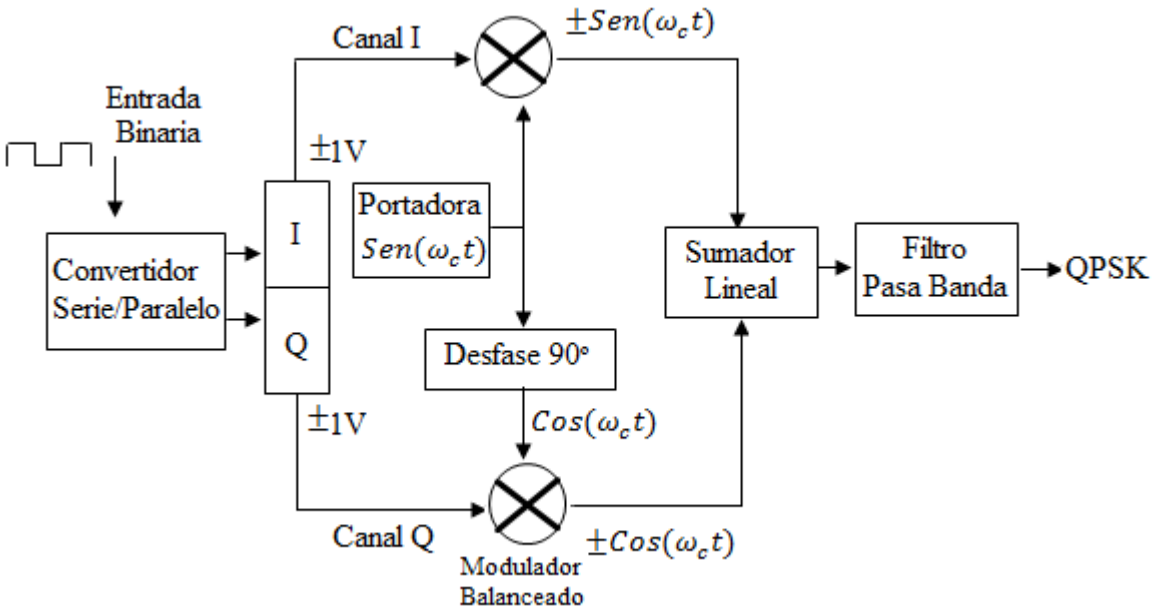


Figura 1.6 Componentes modulador QPSK

Cuando el sumador lineal combina las dos señales de cuadratura (90° fuera de fase) hay cuatro posibles fases resultantes mostradas en la Tabla 1.1.

Tabla 1.1 Valores de fase de la modulación QPSK

Entrada binaria		Fase de salida
Q	I	
0	0	-135°
0	1	+135°
1	0	-45°
1	1	+45°

Cada una de las cuatro posibles salidas tiene exactamente la misma amplitud esta característica de amplitud constante es la más importante del sistema PSK que la distingue de la QAM.

La fase de la portadora toma 1 de 4 valores espaciados equitativamente, es decir pueden ser como en el caso anterior π/4,3π/4,5π/4,7π/4 o también 0, π/2, π, 3π/2,cada uno de los cuales corresponden a un único par del mensaje de bits.El filtro pasa banda que se coloca a la salida del modulador QPSK lo que hace es eliminar los armónicos no significativos de la señal modulada para no interferir con otras señales que pudieran transmitirse por ese mismo canal.

1.3.2 Ancho de Banda para QPSK

Ya que los datos de entrada se dividen en dos canales la tasa de bits también se divide es decir, para el canal I es de fb/2 para el canal Q de fb/2 donde fb es la frecuencia de bit. En esencia el derivador de bits hace que los bits I y Q tengan el doble de tamaño respecto al tamaño de los bits de entrada, en consecuencia la frecuencia fundamental más alta presente a la entrada de datos del modulador será de fb/4. Como resultado la salida de los moduladores balanceados I y Q requiere de un mínimo de ancho de banda de *Nyquist* igual a la mitad de la tasa de bits que están entrando; ya que el modulador genera dos bandas laterales.

$$fn = 2\frac{fb}{4} = \frac{fb}{2} \tag{1.2}$$

La salida de los moduladores balanceados puede expresarse matemáticamente como:

$$(\mathrm{Sen}\ \mathrm{w_a t})(\mathrm{Sen}\ \mathrm{w_c t}) \tag{1.3}$$

Donde $w_a t = 2\pi\frac{\mathrm{fb}}{4}\mathrm{t}$ y $w_c t = 2\pi f c t$

Salida del Modulador

$$= \frac{1}{2}cos2\pi\left(fc - \frac{fb}{4}\right)t - \frac{1}{2}cos2\pi\left(fc + \frac{fb}{4}\right)t \qquad (1.4.)$$

Por lo tanto el mínimo ancho de banda de *Nyquist* será:

$$\left(fc + \frac{fb}{4}\right) - \left(fc - \frac{fb}{4}\right) = \frac{fb}{2} \qquad (1.5.)$$

1.3.3 Receptor QPSK

El diagrama a bloques de un receptor QPSK se muestra en la Figura 1.7. El derivador de potencia dirige la señal QPSK de entrada a los detectores de producto, I y Q, y al circuito de recuperación de la portadora. El circuito de recuperación de la portadora reproduce la señal original del modulador de la portadora de transmisión. La portadora recuperada tiene que ser coherente, en frecuencia y fase, con la portadora de referencia transmisora. La señal QPSK se demodula en los detectores de producto, I y Q, que generan los bits de datos, I y Q, originales. Las salidas de los multiplicadores analógicos se hacen pasar por dos filtros pasa bajos que tienen que tener una frecuencia de corte menor que 2 w_c.

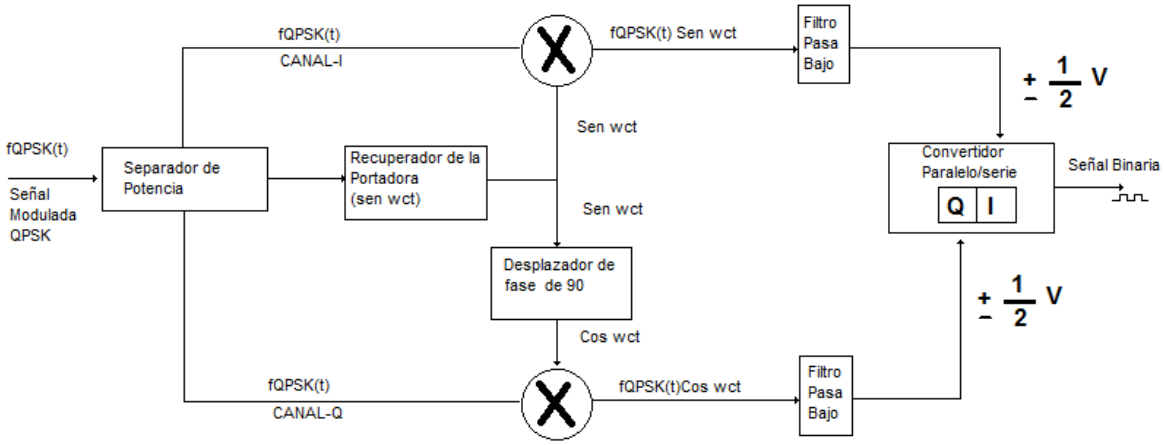


Figura 1.7 Componentes demodulador QPSK

Matemáticamente el proceso de demodulación se describe a seguir:

Suponiendo que el dibit transmitido es QI=10. Entonces las señal analógica $f_{QPSK}(t) = +\cos w_c t - sen\, w_c t$.

Análisis en el canal Q

$$Q = cosw_c t\ f_{QPSK}(t) = cosw_c t(cosw_c t - senw_c t\ )$$

$$= cos^2 w_c t - senw_c t\ cosw_c t\ )$$

$$= \frac{1+cos2w_c t}{2} - \left[\frac{sen(w_c t+w_c t)}{2} + \frac{sen(w_c t-w_c t)}{2}\right] =$$

$$= \frac{1}{2} + \underbrace{\frac{cos2\ w_c t}{2} - \frac{sen2\ w_c t}{2}} - \frac{sen\ 0}{2}$$

Filtro pasa bajo

$$Q = +\frac{1}{2}\ \text{Voltios}\ (1\ \text{l}ógico\ )$$

Análisis en el canal I

$$\text{I} = senw_c t * f_{QPSK}(t) = senw_c t(cosw_c t - senw_c t) =$$

$$= senw_c t\ cosw_c t - sen^2 w_c t =$$

$$= \frac{sen(w_c t+w_c t)}{2} + \frac{sen(w_c t-w_c t)}{2} - \frac{1-cos2w_c t}{2} =$$

$$= \frac{sen0}{2} - \frac{1}{2} + \underbrace{\frac{cos2w_c t}{2} + \frac{sen2w_c t}{2}}$$

Filtro pasa bajo

$$I = -\frac{1}{2}\ \text{Voltios}\ (0\ \text{l}ógico\ )$$

Así queda comprobada la correcta recepción de los datos transmitidos por el demodulador QPSK.

1.4 Aplicaciones de la modulación PSK

Gracias a la simplicidad de la modulación PSK, especialmente cuando se compara con su competidor de modulación de amplitud en cuadratura, es ampliamente utilizada en las tecnologías ya existentes.

El estándar para redes inalámbricas de área local, IEEE 802.11b-1999, utiliza una variedad de PSKs diferentes dependiendo de la velocidad de datos necesaria. Para la transmisión de de 1 Mbit/s, se usa DBPSK (BPSK diferencial). Para la transmisión de 2 Mbit/s se utiliza DQPSK. Para llegar a 5,5 Mbit/s y a una la velocidad máxima de 11 Mbit/s, se utiliza QPSK, pero tiene que estar acoplado con codificación de canal. El estandar IEEE 802.11g-2003 tiene ocho tipos de datos: 6, 9, 12, 18, 24, 36, 48 y 54 Mbit/s. En los modos de 6 y 9 Mbit/s se utiliza OFDM en el que cada sub-portadora es modulada con BPSK y en los modos 12 y 18 Mbit/s también se utiliza OFDM pero las sub-portadoras están moduladas con QPSK.

Debido a su simplicidad BPSK es adecuado para transmisores pasivos de bajo costo, y se ha utilizado en estándares RFID[3] como el ISO / IEC 14443 que ha sido adoptado para los pasaportes biométricos, tarjetas de crédito como American Express ExpressPay, y muchas otras aplicaciones.

Bluetooth 2 utilizará-DQPSK en su tasa más baja (2 Mbit/s) y 8-DPSK en su tasa más alta (3 Mbit/s) cuando el enlace entre los dos dispositivos es suficientemente robusto. Bluetooth 1 modula con GMSK (*Gaussian minimum shift keying*), un esquema binario, por lo que cualquiera de las opciones de modulación en la versión 2 dará lugar a una mayor tasa de datos. Una tecnología similar, IEEE 802.15.4 (el estándar inalámbrico utilizado por *ZigBee*[4]) se basa también en PSK. IEEE 802.15.4 permite el uso de dos bandas de frecuencia: 868-915 MHz usando BPSK y en 2,4 GHz con OQPSK.

[3] Siglas de *Radio Frequency IDentification*, en español identificación por radiofrecuencia.

[4] Es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radiodifusión digital de bajo consumo.

CAPÍTULO II

2. CODIFICACIÓN DE CANAL

Debido a la fluctuación del nivel de potencia de una señal en un entorno móvil, es difícil mantener los parámetros de calidad por encima de un umbral para todo instante de tiempo. Cuando se presentan intervalos más o menos largos por debajo del nivel mínimo de umbral se producen errores en los bits transmitidos que pueden degradar la comunicación. Para evitar este fenómeno es necesario enviar la información codificada, añadiendo una redundancia que permita detectar los errores; disminuye la probabilidad de error en el bit, para una determinada relación señal a ruido.

La codificación de canal es un conjunto de procedimientos que se aplican en la transmisión de una señal para poder manejar y optimizar el medio por donde se propaga la información para evitar bits erróneos. El objetivo de la codificación de canal es reducir la probabilidad de error o bien la relación señal a ruido *(SNR*) necesaria para garantizar una cierta tasa de error. Este efecto se denomina *ganancia del código* y se define como la reducción de la *SNR* requerida para obtener una determinada probabilidad de error en el bit en un canal con ruido gaussiano blanco.

Los bits redundantes que se utilizan para detectar y corregir errores tienen que transformar las secuencias binarias en secuencias mejores que incluyan redundancia estructurada. Existen dos estrategias diferentes para usar la redundancia:

- Detección de errores y retransmisión. Es lo que se conoce como técnicas ARQ (*Automatic Repeat Request*). Esta estrategia utiliza los bits de redundancia (CRC[5]) para detectar si ha habido error. El receptor no intenta corregir el error, sino que pide al transmisor que reenvíe el dato. Esta estrategia implica la existencia de un canal de retorno que permita comunicar al transmisor la recepción correcta o incorrecta de los bits detectados. Además, únicamente puede utilizarse en sistemas de comunicaciones en los que el retardo que implican las retransmisiones no afecte a la calidad del servicio.

- Corrección de errores. Estas técnicas se denominan FEC (*Forward Error Correction*). En este caso, los bits de redundancia se utilizan para corregir errores. Basta con un enlace unidireccional. Los códigos correctores de errores se clasifican según su capacidad de corrección, que en todo caso es limitada.

El control de errores de una comunicación de Datos puede efectuarse por medio de la corrección de errores directa. La Figura 2.1. muestra el modelo de un sistema de comunicación digital que utiliza un procedimiento de este tipo. La fuente genera información en la forma de símbolos binarios. El codificador de canal en el transmisor agrega redundancia al mensaje mediante un procedimiento conocido, produciendo de este modo datos codificados a una tasa de bits más alta. El decodificador del canal en el receptor conoce el procedimiento realizado en el transmisor, siendo capaz de decidir cuáles bits del mensaje se transmiten realmente. La meta combinada del codificador y el decodificador de canal reside en minimizar el efecto del ruido del canal.

[5] Comprobación de redundancia cíclica

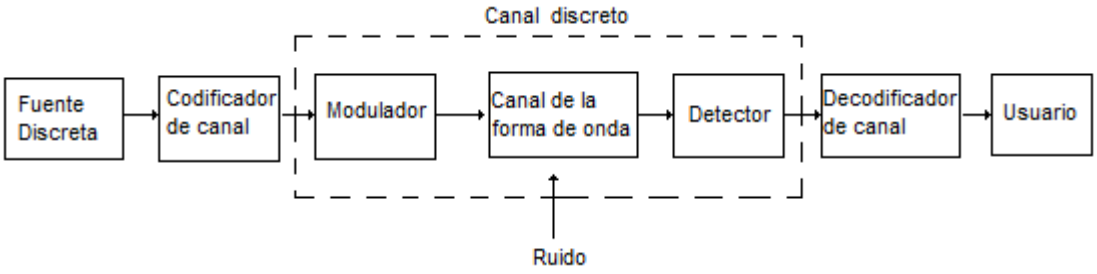


Figura 2.1 Sistema de comunicaciones con corrección de errores directa

En un sistema de modulación fijo, el aumento de redundancia al mensaje implica la necesidad de aumentar el ancho de banda de la transmisión. Además el uso de la codificación de control de errores añade complejidad al sistema, sobretodo la decodificación en el receptor. Por esta razón el diseño de codificación de control de errores tiene que considerar el ancho de banda y la complejidad del sistema.

Los códigos de canal se dividen en dos grandes grupos: los códigos bloque y los códigos Trellis que a su vez se dividen en subfamilias como se muestra en la Figura 2.2.

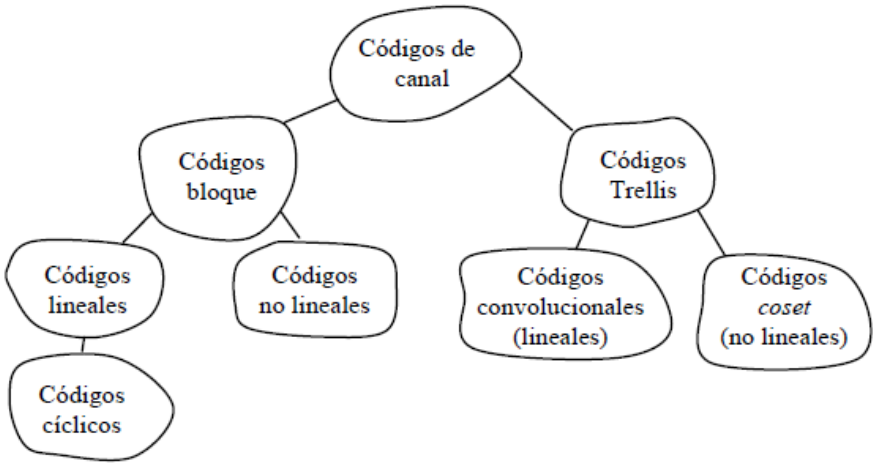


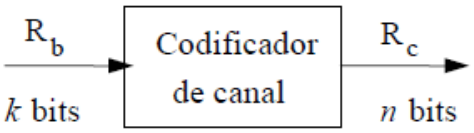
Figura 2.2 Clasificación de códigos de canal [2]

2.1 Códigos de Bloque

El mensaje en los códigos de *bloque* se divide en longitudes *de k* bits que son los bits de información. Estas divisiones son los bloques que representan uno entre los 2^K mensajes distintos posibles. El codificador se encarga de modificar cada división o bloque de *k* bits en un bloque de *n* bits, con n > *k*, generando así la palabra código. La resta de los bits que componen la palabra de código y el mensaje son los bits de redundancia o de paridad, que son los bits utilizados para la corrección de errores. Para los 2^k posibles mensajes, hay 2^k palabras de código. Un código de bloque es eficiente si las palabras código escogidas son las adecuadas. La tasa de código, *r*, es el cociente entre los bits de información, *k,* y la longitud de la palabra código, *n*.

$$r = \frac{k}{n} < 1 \tag{2.1}$$

Como se menciono anteriormente es necesario considerar el ancho de banda disponible por esto se debe incrementar en razón de 1/r porque si se desea que la velocidad de transmisión de los bits de información se mantenga, se debe aumentar la velocidad de salida en un factor *n*/*k*, en el tiempo en que entran *k* bits al codificador, éste debe enviar *n* bits, tal y como se refleja en la Figura 2.3.



$$Rc = R_b \times \frac{n}{k} \tag{2.2}$$

Figura 2.3 Codificador de bloque

2.2 Códigos Convolucionales

A diferencia de los códigos de bloque que utilizan bloques, los códigos convolucionales utilizan el método de codificación de secuencias continuas. Un codificador convolucional genera bits redundantes utilizando convoluciones módulo 2 esto quiere decir que por cada bit que entra en el codificador se obtienen n bits codificados.

Casi siempre los codificadores convolucionales tienen *k* entradas y *n* salidas con *m* registros para cada entrada. Esto quiere decir que cada bit influye en *n* * *m* bits de salida, *m* es longitud de influencia del código y la relación *k/n* es la tasa del código. En la codificación es necesario que todos los bits sean modificados por esto cuando un bit pasa por un registro se le añade *m-1* bits, que son bits de cola que generalmente son ceros.[3]

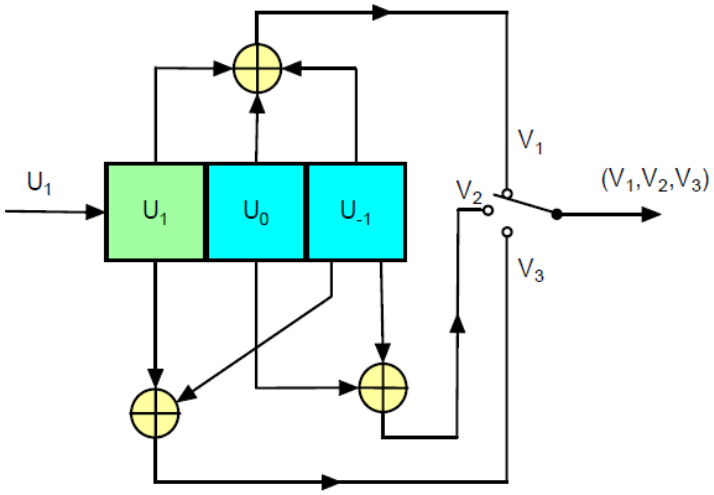


Figura 2.4 Estructura de un codificador convolucional simple.

El codificador de la Figura 2.4, está constituido por un registro de desplazamiento con tres elementos de memoria y tres sumadores en módulo 2 o

generadores de función, en los que se combinan los bits contenidos en las memorias (*flip-flops*). El código de salida se obtiene conmutando secuencialmente las salidas de los generadores de función durante el período de cada bit de entrada, es decir, por cada bit de entrada se producen, en este caso 3 bits de salida.

El primer *flip-flop* contiene al bit de entrada u_1 y los dos siguientes, almacenan los bits anteriores u_0 y u_{-1}. La selección de cuáles bits se suman para producir cada uno de los bits de salida, se designa como *polinomio generador* del código. Los polinomios generadores dan al código la calidad de la protección y ocurre que dos códigos de igual designación, por ejemplo, pueden tener propiedades completamente diferentes dependiendo de los polinomios generadores elegidos para cada uno. Así, para un código de orden *m* hay múltiples polinomios posibles y no todos producen secuencias “buenas” desde el punto de vista de protección contra errores.

2.2.1 Árbol de código

Para conocer la memoria del código se utilizan herramientas que representan en forma gráfica las propiedades estructurales de un codificador convolucional para poder asociar una salida a una determinada entrada.

En el Árbol de código cada rama representa un símbolo de entrada, si el bit que ingresa es un cero se especifica la rama superior de una bifurcación, si es un uno especifica la rama inferior. La trayectoria en el árbol se sigue de izquierda a derecha. En la Figura 2.5 se explica el funcionamiento del árbol de código para la secuencia de mensaje 10011.

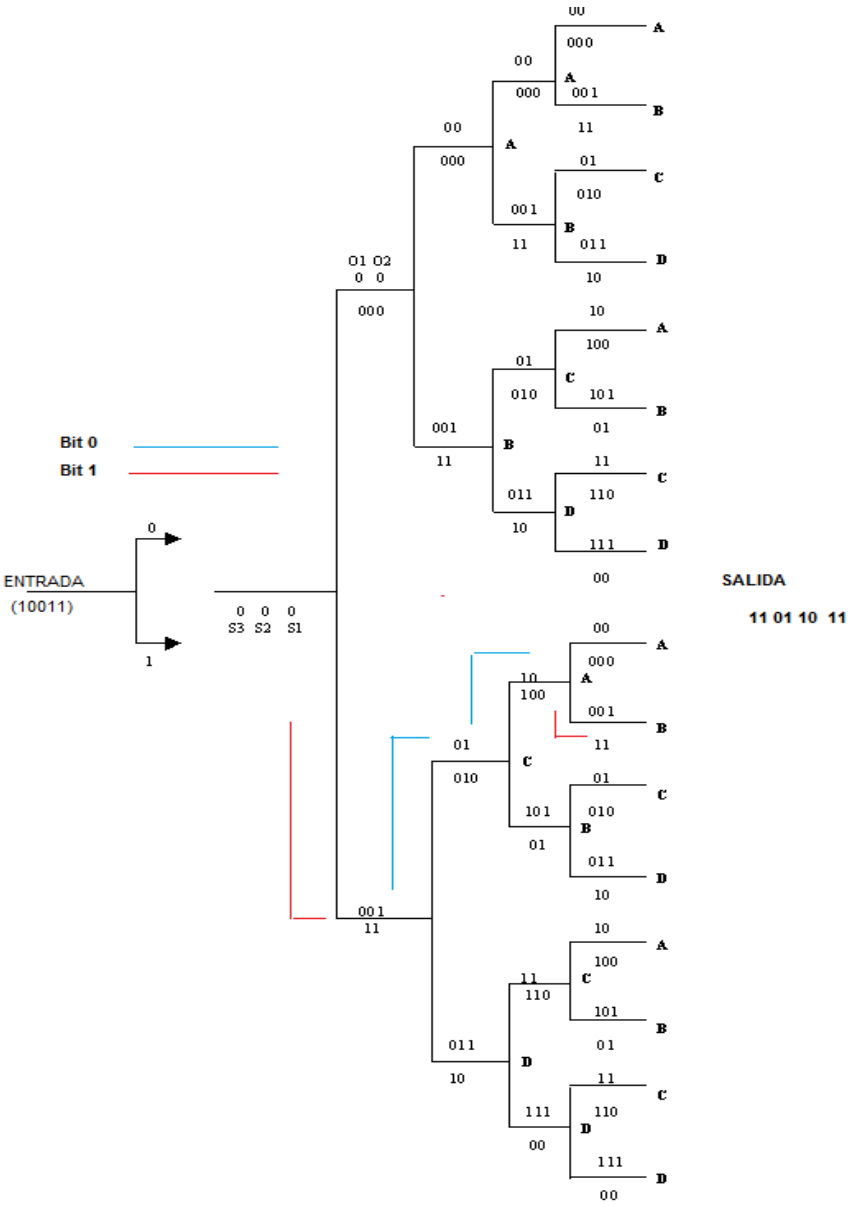


Figura 2.5 Árbol de código.

2.2.2 Diagrama de Trellis

El diagrama de Trellis es un diagrama en forma de red. Cada línea horizontal corresponde con uno de los estados del codificador. Cada línea vertical corresponde con uno de los niveles del árbol del código. Se Parte del estado inicial del codificador en el primer nivel del árbol. A partir de aquí se trazan dos líneas desde este estado. Una para el caso de que la siguiente entrada fuera un 0 y otra para el caso de que fuera un 1. Estas líneas irán hasta el siguiente nivel del

árbol al estado en el que queda el codificador después de haber codificado las correspondientes entradas. Encima de cada una de estas líneas se escribe la salida del codificador para esa codificación. Para cada nivel del árbol se hace lo mismo desde todos los estados en los que el codificador se puede encontrar. En la Figura 2.6. se muestra el diagrama de trellis, donde A,B,C,D son los niveles del árbol de código.

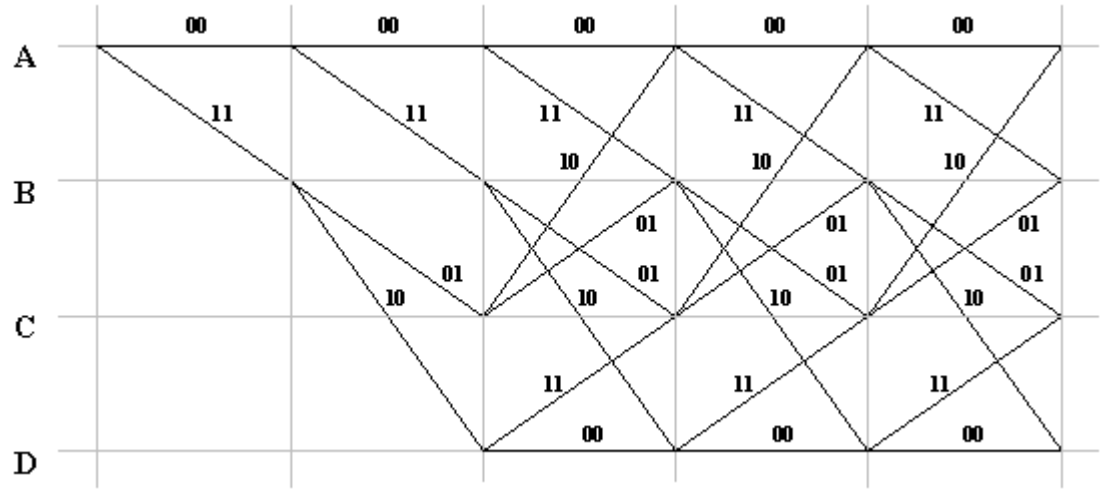


Figura 2.6 Diagrama de Trellis.

2.2.3 Decodificación de Códigos Convolucionales

El proceso de decodificación consiste en buscar una trayectoria en el diagrama de Trellis (o en el árbol del código) donde la secuencia codificada difiera de la secuencia recibida en unos cuantos números de lugares.

La salida de un codificador convolucional sigue unas secuencias determinadas que son forzadas por la estructura del codificador esta característica es la que permite la capacidad correctora en la recepción.

Un método de fuerza bruta de decodificar una secuencia utilizando el principio de máxima verosimilitud es calcular la distancia entre la secuencia recibida y todas las posibles secuencias transmitidas, seleccionado entonces la de mínima distancia. Es decir, si se reciben palabras codificadas de *N* bits de longitud, entonces habrá que hacer 2^N cálculos de distancia. Esto implica que a medida que *N* aumenta se hace impráctica la carga computacional de tal método.

Para realizar la decodificación se utiliza el algoritmo denominado Algoritmo de *Viterbi* que se basa en el principio de máxima verosimilitud; sin embargo reduce la carga de código, El fundamento de este algoritmo está en que no se almacenan todas las secuencias a las que da lugar el codificador. Se basa en el principio de optimalidad: el mejor camino (menor distancia de *Hamming*) a través del diagrama de Trellis que pasa por un determinado nodo, necesariamente incluye el mejor camino desde el principio del diagrama de Trellis hasta este nodo. El principio anterior implica que para cada uno de los nodos del diagrama de Trellis sólo es necesario guardar el mejor camino (secuencia) hasta ese nodo. De esta forma, como mucho se tendrán tantos caminos como estados diferentes.

Se pueden tomar dos tipos de decisiones al momento de la decodificación *duras* (*Hard-Decision*) o*blandas* (*Soft-Decision*).

En un sistema de tipo *Hard-Decision*, primero se decide sobre el dato recibido de forma individual para cada bit, y posteriormente se evalúa la métrica. En cambio en los sistemas *soft-decision* no se realiza la detección independiente de cada símbolo, sino que se realiza la estimación sobre toda la secuencia completa de datos recibidos. La Figura 2.7 muestra la probabilidad de error en los dos tipos de sistemas de decodificación.

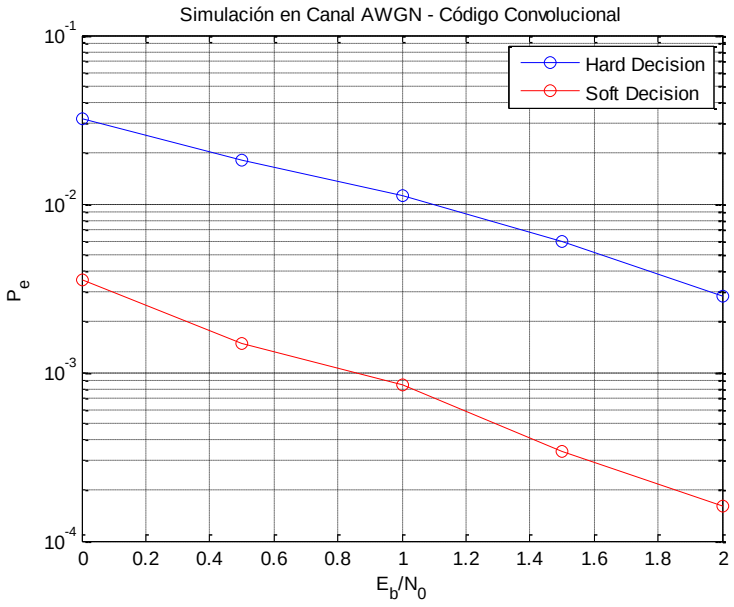


Figura 2.7 Probabilidad de error Hard y Sotf Decision

2.3 Distancia libre de un Código Convolucional

El desempeño de un código convolucional depende no sólo del algoritmo de decodificación utilizado, sino también de las propiedades de distancia del código. Por esta razón, la medida simple más importante de la capacidad de un código convolucional para combatir el ruido del canal es la distancia libre. La distancia libre de un código convolucional se define como la distancia de Hamimng mínima entre cualesquiera dos palabras de código en el código.

2.4 Aplicaciones Códigos Convolucionales

La codificación es muy importante para servicios inalámbricos. Debido a que la propagación de las ondas de radio entre el receptor y la estación base en el medio aire es muy vulnerable al ruido e interferencia. Si las comunicaciones celulares no tendrían la redundancia en los bits transmitidos para aumentar la fiabilidad, no tendrían el éxito que tienen hoy en día. A modo de ejemplo, el primer sistema celular Digital Advance Mobile Phone Service (D-AMPS) utilizaba códigos convolucionales de tasa 1/2 (es decir, el doble de la velocidad de los bits de información) con una longitud de restricción de 6. Las comunicaciones celulares basadas en CDMA a pesar de tener el espectro ensanchado para luchar contra la poca fiabilidad de medio aire, utilizan codificación convolucional de tasa 1/2 en el enlace descendente y 1/3 en el enlace ascendente con una longitud de restricción de 9.

En la televisión digital los códigos convolucionales son altamente utilizados debidos a su capacidad de corregir y detectar errores las tasas que se utilizan son muy variadas dependiendo si aplicación necesita ser robusta o tener una alta capacidad. En el estándar ISDB-T[6] se utiliza códigos convolucionales de tasa 1/2 2/3 3/4 5/6 7/8.

[6] Estándar brasileño de Televisión Digital

CAPÍTULO III

3. TECNOLOGIAS PARA IMPLEMENTACIÓN DE SISTEMAS EN FPGAS

3.1 Introducción a la Tecnología FPGA

Los FPGAs *(Field Programmable Gate Arrays)*, son dispositivos semiconductores que contienen componentes lógicos programables e interconexiones programables entre ellos. Los componentes lógicos programables pueden ser programados para duplicar la funcionalidad de puertas lógicas básicas tales como AND, OR, XOR, NOT o funciones combinacionales más complejas tales como decodificadores o simples funciones matemáticas. En muchos FPGA, estos componentes lógicos programables también incluyen elementos de memoria, los cuales pueden ser simples *flip-flops* o bloques de memoria más complejos. La evolución de los FPGA se ha basado en tres ejes fundamentales. El tecnológico donde el objetivo principal es utilizar geometrías cada vez más pequeñas usando transistores más pequeños y rápidos, acompañadas de costes cada vez menores (por área). El estructural orientado al diseño de sistemas: generadores de acarreos, memorias y multiplicadores embebidos e interconexiones jerárquicas además del control de impedancias E/S. La metodología de programación donde se trabaja en disponibilidad de nuevos módulos sintetizables cada vez más complejos orientándose hacia el diseño modular y en equipo. [4]

Una lógica de interconexiones programables permite que los bloques lógicos de un FPGA, puedan ser interconectados según la necesidad del diseñador de un sistema. Estos bloques lógicos e interconexiones pueden ser programados después del proceso de manufactura por el usuario o diseñador, dependiendo de los recursos del FPGA y así desempeñar cualquier función lógica necesaria.

En la última década la tecnología FPGA ha tenido una gran penetración no solo para fines académicos, sino también para el desarrollo de tecnologías a un menor precio y en un tiempo relativamente corto. El mercado de la tecnología FPGA en el 2008 fue de aproximadamente USD$447 millones y se estima que para el 2012 sea de USD$717 millones. Por esta razón es necesario que en países en vías de desarrollo realicen estudios que involucren a esta tecnología para poder reducir un poco la brecha con las grandes potencias y generar propia tecnología.

3.2 Historia de la Tecnología FPGA

La necesidad de desarrollar y diseñar nueva tecnología dio lugar a la creación de circuitos integrados estándar, los cuales permiten el desarrollo productivo de sistemas relativamente complejos de una forma sencilla, flexibilizando su reproducibilidad y su mantenimiento. Sin embargo, cuando se trata de abordar con esta filosofía el diseño de sistemas más complejos, pronto comienzan a aparecer problemas asociados a la velocidad, el consumo o incluso a la confidencialidad, ya que si un diseño electrónico se encuentra basado en dispositivos estándar es presa fácil de todos aquéllos que quieran copiarlo, siendo esto un enorme inconveniente en un mercado tan competitivo como es el electrónico. Por estos inconvenientes se dio la necesidad de nuevas formas de diseño que origino los sistemas microprogramables. En 1969, Busicon (una joven empresa japonesa) encarga a una pequeña compañía de 10 empleados llamada *Integrated Electronics* (conocida hoy como Intel) un chipset para una computadora. Tratan de utilizar la potencia de la integración con la finalidad de hacer un micro-computador en un chip y luego “personalizarlo” mediante un

programa almacenado en memoria. Como resultado surge en Febrero de 1971 el Intel 4004, ofreciendo a los diseñadores un circuito que permite construir sistemas lo suficientemente complejos mediante una simple programación, pero que es a la vez lo bastante estándar como para poder asegurar la flexibilidad y modularidad comentada anteriormente. Su impacto fue tan grande que pronto los fabricantes desarrollaron versiones económicas, denominadas microcontroladores, orientados al desarrollo e implantación de controladores industriales o incluso versiones más potentes en lo que a habilidades de cálculo se refiere, originando los denominados DSP (*Digital Signal Processor*). Sin embargo, una vez más los requerimientos de diseños cada vez más y más complejos pronto volvieron a colocar una barrera en lo que a velocidad y prestaciones de estos elementos se refiere a la hora de desarrollar sistemas que demandasen una elevada velocidad de trabajo, y la alta confidenciabilidad ofrecida inicialmente pronto se vio que presentaba sus lagunas.

Por un momento y debido a la autoridad intelectual y económica de los diseños se pensó en la opción de los dispositivos ASIC (*Application Specific Integrated Circuit*) como la única solución viable ante tal problemática. Sin embargo, el elevado costo de desarrollo asociado a estos sistemas, tanto en esfuerzo de diseño como de gastos asociados a ingeniería no recurrente, los convertía en una solución poco atractiva para muchos diseñadores. Fue entonces cuando partiendo de la idea que tuvo Texas Instruments con las PAL (*Programmable Array Logic*) en los 70, al difundir puertas lógicas que se puedan unir con la red de interconexiones programables, Ross Freman, co-fundador de Xilinx, inventó el arreglo matricial de compuertas. La raíz histórica de los FPGA son los dispositivos de lógica programable compleja (CPLD) de mediados de los 1980.

Los Dispositivos CPLD tienen un densidad que va desde miles a decenas de miles de compuertas lógicas, en su arquitectura tienen una estructura un poco restringida esto permite una mejor predicción de los tiempos de retraso pero no tienen la alternativa de flexibilidad que permite cambiar de un diseño a otro. Los FPGA van típicamente desde decenas de miles hasta muchos millones de compuertas lógicas. La arquitectura de los FPGAs, son dominadas por las

interconexiones. Esto los hace más flexibles lo que permite ir más lejos en cuanto del diseño inicial. Otra notable diferencia entre CPLDs y FPGAs es la presencia de funciones de más alto nivel (tales como sumadores y multiplicadores) dentro de los FPGAs, además de memorias. Una diferencia importante es que muchos FPGAs modernos, soportan una total o parcial re-configuración del sistema, permitiendo que una parte del diseño sea re-programada mientras las otras partes siguen funcionando. En la figura 3.1 se puede ver el esquema básico de un FPGA. En la estructura interna de un FPGA, cada bloque combinacional (LUT) está seguido por un *flip-flop*, es conveniente realizar la implementación de funciones lógicas de muchas variables por medio del particionamiento de la lógica combinacional en distintas etapas separadas entre sí por registros (estructura de *pipeline*), lo cual permite mantener acotados los tiempos de propagación de las señales. Se sabe que en una estructura de pipeline la frecuencia máxima de operación está determinada por el tiempo de propagación de la etapa más lenta, por lo tanto, el dividir en etapas el cálculo de una función lógica incrementa la frecuencia de operación del circuito. [5]

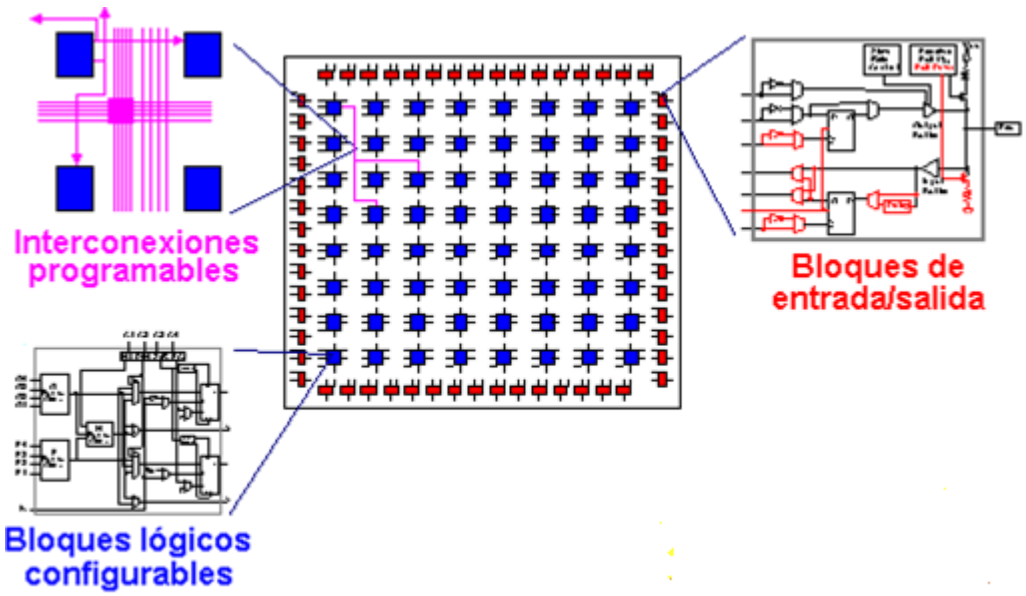


Figura 3.1 Esquema básico de un FPGA

3.3 Arquitectura de un FPGA

La arquitectura de un FPGA consiste en arreglos de compuertas lógicas que son programadas y que están interconectadas entre sí mediante canales de conexiones verticales y horizontales tal como se muestra en la Figura 3.2.

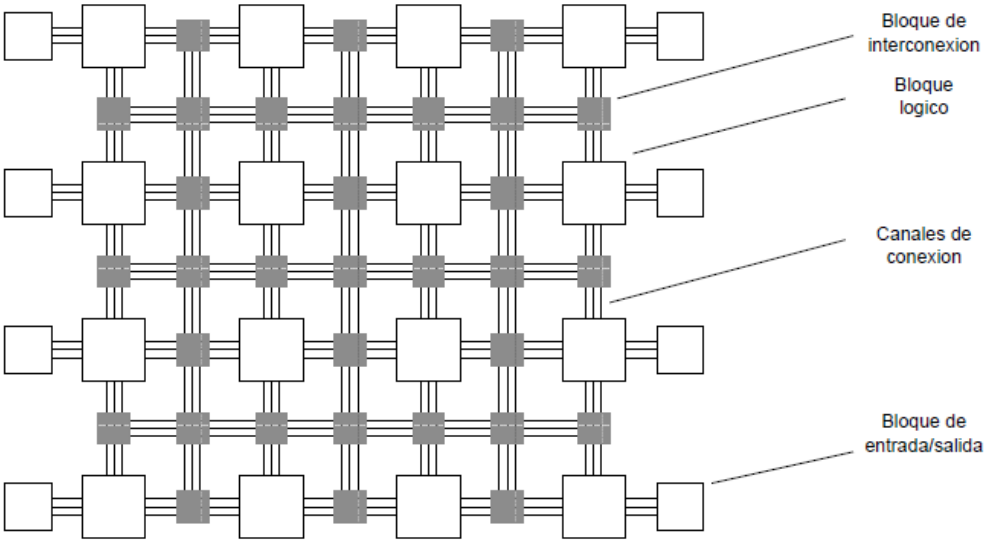


Figura 3.2 Arquitectura general de un FPGA

3.3.1 Bloques lógicos (CLBs)

Los bloques lógicos se conforman de una parte combinacional que implementa operaciones o funciones lógicas booleanas y una parte secuencial formada por flip-flops, que sirven para sincronizar la salida de un bloque con una señal de reloj externa, lo que permite realizar circuitos secuenciales y implementar registros.

La estructura de un bloque lógico pude variar según el fabricante, pero la parte combinacional se basa principalmente en una *Look-Up Table* (LUT). Una LUT es un componente de memoria que almacena una tabla de verdad. Las direcciones

de la memoria son las entradas de la función lógica que se debe implementar y en cada celda de la memoria se almacena el resultado de la combinación correspondiente de las entradas.

Las LUTs de N entradas son básicamente una memoria que, cuando es programada apropiadamente, puede realizar cualquier función de hasta N bits de entrada. Un FPGA típico tiene CLBs con una o más LUTs de 4 entradas, opcionalmente puede contar con *flip-flops* tipo "D" y con circuitos que implementen el cálculo de acarreos de forma rápida. Las LUTs permiten que cualquier función booleana que no exceda el número de entradas pueda ser implementada, por lo que provee circuitos lógicos genéricos. Los *flip-flops* pueden ser usados para realizar un *pipeline* (técnica para paralelizar un proceso), registros o cualquier otra función en la que se requiera sincronización. Los circuitos para el cálculo de acarreos de forma eficiente son recursos especiales que se encuentran en la celda con la finalidad de acelerar los cálculos que requieran del acarreo. Debido a que las opciones de interconexión son muy pocas, y debido al poco retraso en los cálculos, el uso de estos recursos mejora significativamente la velocidad en la propagación del acarreo. Frecuentemente en los Bloques Lógicos de los dispositivos más recientes, se incluyen componentes lógicos adicionales (como por ejemplo compuertas XOR, multiplexores), los cuales permiten que sean implementadas de manera eficiente una gran variedad de funciones.[6] La Figura 3.3 muestra la estructura de un bloque lógico.

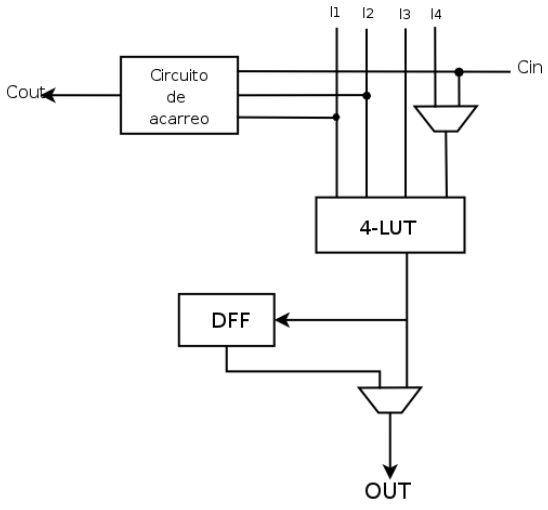


Figura 3.3 Estructura de un bloque lógico

3.3.2 Tecnologías de Programación

En un FPGA es necesario conocer como se almacena la información correspondiente a las funciones lógicas y a las conexiones que deben hacerse entre bloques lógicos que se quiere implementar. Las tecnologías de programación más comunes son las siguientes:

SRAM (RAM estática):

Este tipo de almacenamiento es muy común en la tecnología FPGA y se configura en el momento de encendido del circuito que contiene al FPGA. Debido a que al ser SRAM el contenido de la memoria se pierde cuando se deja de suministrar energía; la información binaria de las celdas SRAN se almacena en memorias seriales EEPROM conocidas como memorias de configuración. En el momento de encendido del circuito, toda la información binaria es transferida a los bloques e interconexión del FPGA mediante el proceso de configuración. Un circuito dedicado del FPGA inicializa todos los bits de la SRAM al encender y configurar los bits con una configuración proporcionada por el usuario. A diferencia otras tecnologías de programación, el uso de celdas SRAM no requiere el procesamiento de circuitos integrados especiales más allá del estándar CMOS. Como resultado, los FPGAs basados en SRAM pueden utilizar la última tecnología CMOS disponible y, así, se benefician de la mayor integración, las velocidades más altas y el menor consumo de potencia dinámica. [7]

ANTIFUSE (antifusible):

Este tipo de programación se utiliza para dejar un diseño permanentemente en un FPGA debido a que solo se le puede programar una vez (*One Time Programed*, OTP). A esta tecnología se la denomina así porque a diferencia de una conexión con un fusible que está establecida, la conexión debe ser creada, es decir que no existe originalmente y una vez que es programada no se puede recuperar el estado original de la conexión. Un antifusible consiste en dos líneas perpendiculares conductoras que comúnmente son de aluminio, separadas por una capa de dieléctrico que puede ser un oxido. La programación se realiza

estableciendo una tensión elevada entre las dos líneas que se cruzan. Esta tensión es superior a la rigidez dieléctrica del oxido y el campo eléctrico lo rompe estableciendo un pequeño arco entre las dos pistas, que las funde parcialmente, creando así una conexión permanente.

La principal ventaja de esta tecnología es considerable disminución en su tamaño. Además tienen menos resistencias y capacitancias parásitas que otras tecnologías. Estas ventajas permiten incluir más interruptores por dispositivo. Estos dispositivos son no volátiles. Su costo disminuye debido a que no es necesaria una memoria para almacenar la información de programación y esto hace que este tipo de FPGAs puedan ser usados en situaciones que requieren una operación inmediata después de encender el equipo.

FLASH:

En los últimos años han evolucionado las celdas de memoria flash y ha sido posible su implementación en los dispositivos programables.Los FPGAs basados en celdas flash tienen las principales ventajas de las dos técnicas de programación mencionadas anteriormente. Su tamaño es intermedio entre una celda SRAM una antifusible; son reprogramables, aunque la velocidad de programación es bastante más lenta que en el caso de una SRAM; y son no volátiles es decir que no necesitan de otro dispositivo para guardar la configuración interna. Otra diferencia con la tecnología SRAM son los buffers de alta y baja tensión que son necesarios para programar la celda y que contribuyen para evitar la sobrecarga del área, sin embargo estos generan un costo relativamente modesto ya que se amortiza a través de numerosos elementos programables.

Una desventaja de los dispositivos con tecnología de programación flash es que no pueden ser reprogramados infinitamente. En muchos casos los FPGAs son programados para una función específica, en ese caso un dispositivo con tecnología flash es más que suficiente. Otra desventaja importante de dispositivos flash es la necesidad de un proceso CMOS no estándar. Igualmente que en la tecnología basada en memoria estática, esta tecnología de programación tiene

una resistencia relativamente alta debido a la utilización de conmutadores basados en transistores.

3.3.3 Bloques de Entrada y Salida

La periferia del FPGA está constituida por bloques de entrada y salida configurables por el usuario cuya función es permitir el paso de una señal hacia dentro o hacia el exterior del dispositivo.

Cada bloque puede ser configurado independientemente para funcionar como entrada, salida o bidireccional, admitiendo también la posibilidad de control triestado. Las entradas o salidas pueden configurarse para trabajar con diferentes niveles lógicos (TTL, CMOS). Además, cada entrada o salida incluye *flip-flops* que pueden utilizarse para registrar tanto las entradas como las salidas.

Los FPGA utilizan cientos hasta miles de terminales de conexión además funcionan a altas velocidades de operación por este motivo los bloques de entrada o salida deben ser capaces de proveer una adecuada terminación en cuanto a términos de impedancia se refiere, de forma de evitar reflexiones de las señales.

3.3.4 Líneas de Interconexión

Constituyen un conjunto de caminos que permiten conectar las entradas y salidas de los diferentes bloques. Están constituidas por líneas metálicas de dos capas que recorren horizontal y verticalmente las filas y columnas existentes entre los componentes lógicos.

En el proceso de conexión participan elementos como los puntos de interconexión Programable que son los que permiten que los bloques lógicos y las entras/salidas del FPGA puedan comunicarse. Otro elemento son las matrices de

interconexión *SW ("Switch Matriz o Magic Box"*) que son dispositivos de conmutación distribuidos de forma uniforme por el FPGA. En la Figura 3.4. Se muestra la conexión de una *SW* [8]

En general, las conexiones internas que pueden realizarse entre los elementos de un FPGA son las siguientes:

Líneas directas. Tienen lugar entre bloques adyacentes.

Conexiones de propósito general: Se realizan a través de la matriz de interconexión cuando se desean conectar dos bloques no adyacentes. Su misión es conectar canales verticales y horizontales que permiten llegar al punto final de la conexión. Cuanto más larga es la línea de ruteo, mayores serán los retrasos introducidos tanto por la propia longitud de la línea como por los elementos de interconexión utilizados a lo largo de la misma.

Líneas largas. Estas conexiones se utilizan cuando una señal debe recorrer una gran longitud, por ejemplo en la implementación de un bus interno, de forma tal de evitar que la misma atraviese elementos de interconexión evitando de esta forma retrasos introducidos por los mismos.

Líneas rápidas. Cuando los bloques lógicos contienen más de una LUT, las líneas rápidas son las encargadas de realizar las conexiones entre las distintas LUTs.

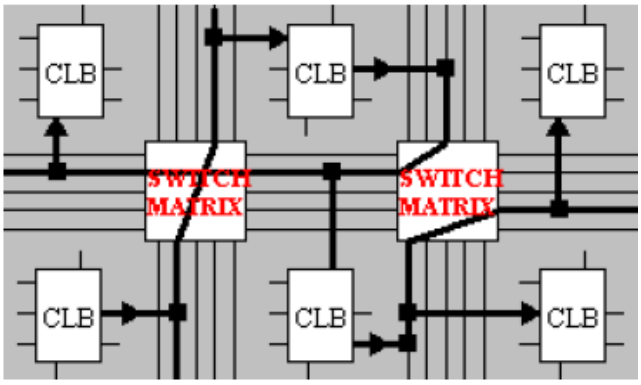


Figura 3.4 Matrices de Interconexión

3.3.5 Recursos internos de los FPGA actuales.

Los FPGA cuentan en su arquitectura con recursos que cumplen funciones de propósito específico a continuación se nombra a los más importantes.

RAM distribuida

Las LUTS que se basan en celdas SDRAN permiten utilizar estas celdas como memoria RAM o como registros de desplazamiento. Una LUT de 4 entradas puede ser utilizada para formar una memoria RAM de 16 x 1 bits. A su vez, las 16 celdas de memoria pueden ser interconectadas en forma de cadena formando un registro de desplazamiento.

Cadenas de acarreo rápidas

Los FPGAs actuales es que incluyen lógica y elementos de interconexión útiles para implementar cadenas de acarreo rápidas (*fast carry chains*) cuya función es la de propagar de forma rápida señales de acarreo entre dos elementos lógicos.

Las cadenas de acarreo permiten realizar de forma eficiente sumadores, multiplicadores, contadores y comparadores de gran velocidad.

RAM embebida

La gran mayoría de dispositivos FPGA incluyen bloques de memoria RAM además de la RAM distribuida en las LUTs. Estos bloques de memoria pueden ser configurados de diversas maneras, formando memorias de un solo puerto (*single-port RAMs*), memorias de doble puerto (*dual-port RAMs*), memorias FIFO (*first-in first-out*), etc.

Multiplicadores

Algunas funciones lógicas tales como suma y multiplicación son inherentemente lentas si se realizan interconectando un gran número de bloques lógicos.

Para aplicaciones de procesamiento digital de señales, donde es necesario el uso de varias sumas y multiplicaciones a mayor velocidad (denominadas operación MAC, *Multiply and Accumulate*), el hecho de que los FPGAs cuenten con una cantidad de multiplicadores embebidos los hace ideales para ser utilizados.

3.4 Flujo de diseño utilizando FPGAs

El proceso de diseño para los FPGA de marca Xilinx se divide en cuatro fases: descripción del modelo, síntesis, implementación y programación. En la Figura 3.5. se muestra el flujo de diseño con más detalle.

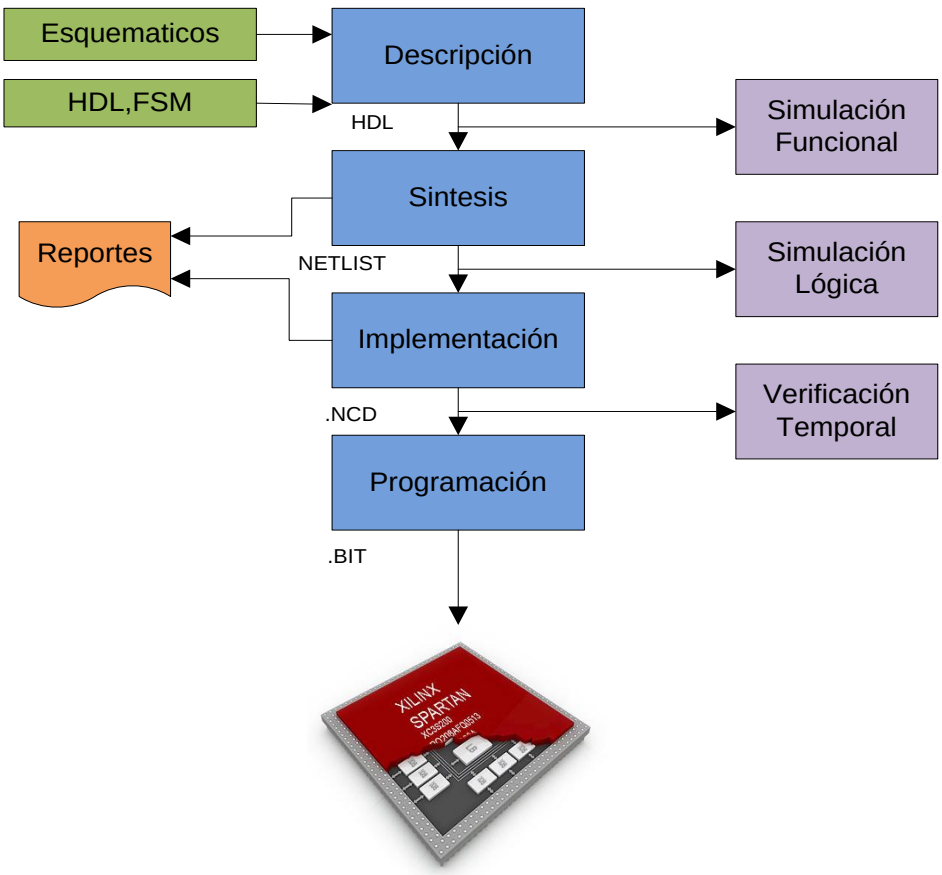


Figura 3.5 Flujo de Diseño para FPGA

3.4.1 Descripción del Modelo

La Descripción del modelo consiste en crear un modelo RTL (*Register Transfer Level*) que representa un circuito digital como un conjunto de primitivas (sumadores, contadores, multiplexores, registros, etc.)

Hay dos herramientas básicas para la creación de un diseño RTL: Lenguaje de descripción de Hardware HDL *("Hardware Description Language")* y lenguaje Esquemático a continuación se describe cada una de estas.

El lenguaje esquemático se basa en una representación elemental en un nivel de compuertas lógicas.

Es muy útil para circuitos básicos o proyectos de mediana escala, su implementación consiste en arrastrar los componentes necesarios para un diseño específico. El lenguaje esquemático muestra algunas deficiencias en cuanto a grandes proyectos, ya que son muy sensibles a errores y necesitan mucho tiempo de desarrollo. En la Figura 3.6. se muestra un diagrama básico del lenguaje esquemático.

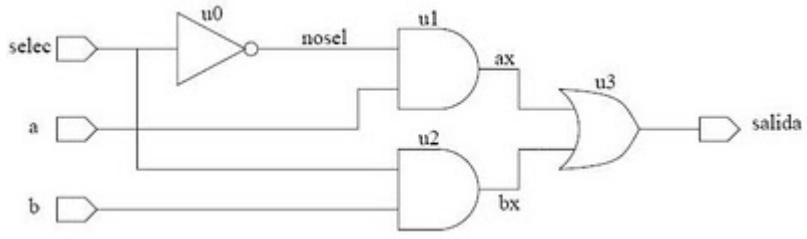


Figura 3.6 Diagrama Esquemático

Lenguaje de Descripción de Hardware (HDL)

El diseño en lenguaje de descripción de Hardware se basa en la creación y uso de descripción de texturas basadas en circuitos o sistemas lógicos digitales. Mediante el uso de cualquier HDL (los dos estándar IEEE más comúnmente

usados en la industria y educación son *Verilog -HDL* y *VHDL*), la descripción del circuito puede ser creada en los diferentes niveles de abstracción, descripción de puertas lógicas básicas de acuerdo con la sintaxis del lenguaje (el orden gramatical de las palabras y símbolos utilizados en el lenguaje) y semántica (el significado de las palabras y símbolos utilizados en el lenguaje).

Circuitos hardware o diseños de sistemas creados usando HDL se genera en diferentes niveles de abstracción. Comenzando en el nivel más alto (es decir, más alejado de los detalles del circuito), la idea o el concepto del sistema es la descripción inicial de alto nivel del diseño que proporciona la especificación del diseño. El nivel de algoritmo describe el comportamiento del diseño en términos matemáticos. Ni la idea de sistema, ni el algoritmo describe cómo será el comportamiento del diseño implementado. La estructura del algoritmo en hardware es descrito por la arquitectura, que identifica los bloques funcionales de alto nivel que se usan y como las funciones son conectadas. Los niveles de arquitectura y algoritmo describen el comportamiento del diseño para ser verificado en simulación.

El siguiente nivel por debajo de la arquitectura es el nivel de registro de trasferencia (RTL), que describe el almacenamiento (en los registros) y el flujo de datos en torno a un diseño, junto con las operaciones lógicas realizadas con los datos. Este nivel es utilizado generalmente por las herramientas de síntesis que describen la estructura del diseño (la lista de red del diseño en términos de las compuertas lógicas y el cableado de interconexión entre las mismas).

En el diseño con HDL, el diseñador elige el lenguaje a utilizar y en qué nivel de abstracción de diseño trabajar. Al elegir el lenguaje, los siguientes aspectos deben ser considerados:

- La disponibilidad de instrumentos adecuados para la automatización de diseño electrónico (EDA) para apoyar el uso del lenguaje (incluida la capacidad de gestión del diseño y la disponibilidad de uso de herramientas dentro de un proyecto).
- Conocimiento previo.
- Preferencias personales.
- Disponibilidad de modelos de simulación.

- Capacidad de síntesis.
- Cuestiones comerciales.
- La reutilización de diseños.
- Requisitos para aprender un nuevo lenguaje y las capacidades del lenguaje.
- Acceso a herramientas de soporte para el lenguaje, tales como la existencia de herramientas de comprobación de código y herramientas que generen documentación.

VHDL

Very high-speed integrated circuit hardware description language—VHSIC HDL o VHDL surgió en 1980 bajo el departamento de defensa de los Estados Unidos de los requisitos para el diseño de circuitos digitales con una metodología de diseño común, proporcionando la capacidad para la auto-documentación y reutilización con nuevas tecnologías. El desarrollo de VHDL comenzó en 1983 y se convirtió en un estándar IEEE en 1987. El lenguaje fue revisado en 1993,2000 y 2002. VHDL también tiene una serie de estándares relativamente asociados a la modelización y a la síntesis.

El código HDL está contenido en un archivo de texto ASCII y por lo tanto es transportable entre herramientas EDA de un mismo sistema operativo, entre computadoras, entre diferentes versiones de herramientas EDA y entre diferentes ingenieros con un equipo de diseño en particular.

El lenguaje VHDL está diseñado para satisfacer determinadas necesidades en el proceso de diseñó:

1. Permite la descripción de la estructura del diseñó, lo que representa la manera en que está dividida en sub-diseños, y cómo esos sub-diseños son interconectados.
2. Permite que las funciones sean especificadas mediante el uso de estructuras de lenguajes de programación que son familiares a los desarrolladores.

3. Como resultado, es posible realizar la simulación del diseñó antes de ser fabricado, por lo que los diseñadores son capaces de comparar las alternativas y probar que sean correctos sin el retraso y el costo del desarrollo de prototipos de hardware.

VHDL divide las entidades (componentes, circuitos o sistemas) en una parte visible o externa (nombre de la entidad y conexiones) y una parte oculta o interna (algoritmo de la entidad e implementación). Después de definir la interfaz externa, otras entidades pueden usarla cuando haya sido completamente desarrollada. Este concepto de vista externa e interna es central para una perspectiva de diseñó de sistemas en VHDL. Una entidad es definida, en relación a otras, por sus conexiones y comportamiento. Se pueden exportar implementaciones externas (arquitecturas) de una de ellas sin cambiar el resto del diseñó. Después de definir una entidad para un diseñó, se puede reutilizar en otros diseños tanto como sea necesario. La Figura 3.7 muestra un modelo de hardware en VHDL.

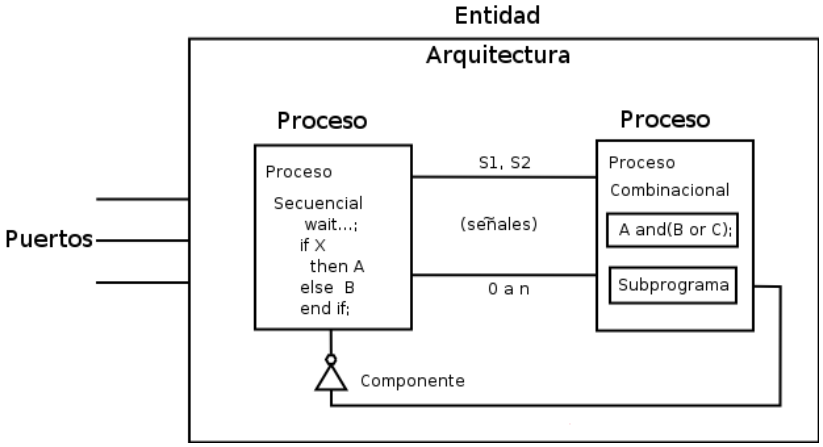


Figura 3.7 Modelo VHDL de Hardware

Una entidad (diseñó) VHDL tiene uno o más puertos de entrada, salida o entrada/salida, los cuales son conectados a sistemas vecinos. Está formada de procesos y componentes interconectados, todos operando concurrentemente. Cada entidad es definida por una arquitectura particular, que se compone de construcciones VHDL tal como operaciones aritméticas, asignaciones de señal, o declaraciones de instanciación de componentes.

En VHDL, los circuitos de modelo secuencial en procesos independientes (síncronos), usan *flip-flops* y *latches,* y los circuitos combinacionales (asíncronos) utilizan típicamente compuertas lógicas. Los procesos pueden definir y llamar (instanciar) subprogramas (diseños secundarios). Los procesos se comunican con cada uno de los otros procesos mediante señales. Una señal tiene una fuente (manejador), uno o más destinos (receptores), y un tipo definido por el usuario.

Las construcciones del lenguaje VHDL están divididas en tres categorías de acuerdo a su nivel de abstracción:

- Una descripción *estructural*, que describe la estructura del circuito mediante compuertas lógicas y el cableado de interconexión entre las mismas que forman la lista de conexiones del circuito.

- Una descripción de *flujo de datos,* que describe la transferencia de datos desde la entrada a la salida y entre las señales. Una operación es definida en términos de una colección de transformación de datos, expresado como una sentencia concurrente.

- Una descripción de *comportamiento,* que describe el comportamiento del diseño en términos del comportamiento de circuitos y sistemas utilizando algoritmos. Esta descripción de alto nivel utiliza construcciones del lenguaje que se asemejan a una programación de alto nivel de cualquier otro lenguaje.

Tanto como la descripción de flujo de datos como la de comportamiento utiliza construcciones de lenguaje similares, pero en VHDL son diferentes: una descripción de comportamiento utiliza la sentencia *process* para inicializar un algoritmo, mientras que una descripción de flujo de datos no lo hace.

VERILOG –HDL

Verilog –HDL fue lanzado en 1983 por Gateway Design System Corporation, junto con un simulador de Verilog-HDL. En 1985, el lenguaje y el simulador fueron mejorados con la introducción del simulador Verilog-XL. En 1989, *Cadance Design Systems,Inc*. compró *Gateway Design System Corporation,* y a principios de los 90, Verilog-HDL y Verilog - XL fueron separados en dos productos. Verilog-HDL, hasta entonces un lenguaje propio de la empresa, fue publicado para el dominio público para facilitar la difusión de conocimientos relacionados con Verilog-HDL y permitir Verilog-HDL competir con VHDL, que ya existía como un lenguaje abierto. En 1990, *Open Verilog International* (OVI) fue formado como un consorcio industrial formado por ingeniería asistida por ordenador (CAE), vendedores y usuarios Verilog HDL-para controlar la especificación del lenguaje. En 1995, Verilog-HDL fue revisado y adoptado como el estándar IEEE 1364. En 2001, el estándar fue revisado, la última versión del estándar es IEEE Std 1364-2001. [9]

Verilog-HDL es un lenguaje que nació ante la necesidad de mejorar el flujo de diseño y por tanto es un lenguaje que cubre todas las necesidades de diseño y simulación. El resultado es un lenguaje simple y flexible que es fácil de aprender aunque tiene ciertas limitaciones para algunos diseños. Las construcciones del lenguaje Verilog-HDL son similares a las de VHDL ya que las categorías de acuerdo al nivel de abstracción son en general para HDL.

Como se muestra en la Figura 3.5 una vez terminada la descripción del circuito, se procede mediante el uso de un simulador, a realizar la simulación funcional del circuito depurando posibles errores en la descripción del mismo. En esta fase de diseño se verifica el comportamiento del circuito. Generalmente, las herramientas de simulación permiten realizar la verificación del diseño en base a comprobación automática de las respuestas del circuito en base a estímulos de entrada conocidos.

3.4.2 Síntesis del Modelo

En la síntesis se convierte la descripción de alto nivel que tiene el circuito a un nivel bajo de abstracción usando componentes contenidos en una biblioteca. Para comprobar los resultados de la síntesis es preciso realizar una simulación funcional (retardos unitarios) a nivel de puertas. El procedimiento a seguir es similar a la simulación funcional RTL.

3.4.3 Implementación del Modelo

En la implementación del modelo se realizan algunos procesos primero se crea un único fichero de salida. El formato de este fichero de salida se denomina: NGD (*Native Generic Design*) y contiene componentes lógicos: puertas lógicas, biestables, RAMs, etc. A continuación se mapean los componentes lógicos en los componentes físicos de que dispone la FPGA: tablas LUT, biestables, buffers triestado, etc. Además se encapsulan estos componentes físicos en bloques configurables (CLBs e IOBs), creando un fichero de salida en formato NCD (*Native Circuit Design*). Para finalizar se pasa a realizar el posicionamiento e interconexión de los bloques configurables en el FPGA y se genera un fichero de lista de conexiones en formato NCD.

3.4.4 Análisis de Tiempos

Una vez implementado físicamente el diseño, se procede a analizar los tiempos y velocidades de operación del mismo. Si los tiempos y velocidades no corresponden con los especificados en la etapa de descripción del diseño, deberá modificarse la descripción del circuito a fin de ajustar los mismos. Después de realizar estas fases se obtiene la información para realizar simulaciones temporales del diseño completo y se puede crear un archivo HDL estructural con base en los componentes físicos de la FPGA y un fichero SDF (*Standart Delay Format*) con los retardos internos del FPGA. Utilizando estos ficheros se puede

realizar la verificación del diseño final mediante una simulación temporal HDL post-síntesis.

3.4.5 Puesta en Producción

Por último, una vez satisfechos todos los objetivos del diseño, se procede a generar el fichero BIT de configuración de la FPGA.

3.5 Herramientas de Implementación

Existen varias herramientas EDA (*Automatización de Diseño Electrónico*) que facilitan y aceleran el proceso de implementación de diseños. Estas herramientas son provistas por los fabricantes de FPGA e incluyen funciones lógicas optimizadas para sus dispositivos, además de simuladores.

XILINX es una empresa que se ha dedicado a la creación de herramientas EDA para facilitar el diseño, implementación y programación de los FPGA con la limitante que estas herramientas están orientadas hacia dispositivos FPGA de esta marca por lo que no son útiles para otras plataformas, esto hace que el flujo de diseño sea poco flexible. Durante el proceso se crean y se utilizan archivos con formatos propietarios de Xilinx, lo cual impide se puedan utilizar en otro tipo de herramientas o aplicaciones. En la Tabla 3.1 se muestra los diferentes paquetes y herramientas de diseño que ofrece Xilinx.

Tabla 3.1 Paquetes de Herramientas Xilinx

Tabla de Comparación de Productos ISE Design Suite	ISE WebPACK Tool	Logic Edition	Embedded Edition	Dsp Edition	System Edition
Herramientas ISE Foundation con ISE Simulator	X	X	X	X	X
PlanAhead Design Analysis Tool	X	X	X	X	X
ChipScope Pro Logic Analyzer		X	X	X	X
Embedded Development Kit (EDK)		X	X	X	X
Software Development Kit(SDK)			X		X
System Generator for Dsp				X	X

Para poder descargar el producto ISE (*Integrated Synthesis Enviroment*) Design Suite 11 en la página oficial de Xilinx es necesario registrarse y crear una cuenta de usuario de Xilinx. Una vez descargado se procede a la obtención y administración de una licencia que difieren entre usuarios y acciones.

3.5.1 Tipos de Usuarios y Acciones

Hay tres tipos de usuarios para la descarga del producto y el sitio de licencias, administrador de la cuenta del cliente, usuario final y usuario de evaluación.

Administrador de la cuenta del cliente

Un ejemplo de un administrador de la cuenta del cliente típico es un administrador de herramientas CAD *(Computer Aided Design)*. Cada cuenta de licencia de productos debe tener al menos un administrador de la cuenta del cliente. Un administrador de cuenta de cliente puede gestionar más de una cuenta de licencia de productos.

Las responsabilidades como administrador de la cuenta del cliente incluyen:

- Generación de nodo bloqueado o licencias flotantes para el software de Xilinx y productos de propiedad intelectual.
- Agregar y quitar usuarios de la cuenta de concesión de licencias de productos.
- Asignación de privilegios administrativos a otros usuarios.
- Solicitud de DVD del producto.

Usuario final

Adición de usuarios finales a una cuenta de licencia de productos permite a un ingeniero o un miembro del equipo de diseño la flexibilidad necesaria para gestionar y generar las claves de licencia por su propia cuenta. El usuario final puede generar las claves de licencia de productos de nodo bloqueado, derechos dentro de la cuenta. El usuario final también puede generar claves de licencia para software de evaluación y la evaluación de los productos de propiedad intelectual sin recargo. El usuario Final tiene algunas restricciones.

- Un usuario final no es capaz de generar claves de licencia flotantes. Sólo los administradores de cuentas de los clientes puede generar claves de licencia basada en un servidor (flotante).
- El usuario final no es capaz de ver las claves de licencia generados por otros usuarios.
- El usuario final no es capaz de agregar o quitar a otros usuarios o de la cuenta de concesión de licencias de productos.

Usuario de Evaluación

El usuario de Evaluación se aquel que pone a prueba por un tiempo límite el software la unidad de disco ISE Design Suite 11. Los usuarios de evaluación pueden:

- Generar una clave de 30 días de licencia gratuita de evaluación de ISE Design Suite 11: System Edition
- Generar las claves de licencia para la evaluación y no hay recarga en productos de propiedad intelectual.
- Solicitar un DVD ISE Design Suite 11. Los usuarios de evaluación deben pagar por el envío de los DVD.

3.5.2 Herramientas Xilinx

Integrated Synthesis Enviroment (ISE) es una herramienta EDA de la compañía Xilinx para la síntesis de las aplicaciones de sistemas embebidos en un FPGA. Xilinx proporciona herramientas EDA con licencia (*ISE FOUNDATION*) y sin licencia (*ISE WebPACK*), que tienen la misma funcionalidad pero obviamente la herramienta con licencia tiene mayores ventajas y facilidades.

El entorno de diseño Xilinx ISE incluye todas las etapas necesarias como son:

- La entrada de diseño, bien a través de captura esquemática, lenguajes de descripción hardware como VHDL o Verilog-HDL o representación gráfica de diagramas de estado.
- Herramientas de Verificación para la obtención de una simulación del sistema, tanto a nivel funcional como estimación de retardos.
- Herramientas de implementación.
- Herramientas de Programación.

El entorno de ISE permite combinar las diferentes técnicas de diseño para facilitar la labor de descripción del diseño. Además, se permite la inclusión de restricciones para optimizar el proceso de implementación y adaptarlo a las necesidades del diseño.

3.5.3 Descripción del entorno de desarrollo ISE

1. Ventana de ficheros fuente. En esta ventana se muestran los ficheros fuente utilizados en el diseño y las dependencias entre ellos. También es aquí donde se elige el tipo de dispositivo donde se desea implementar el diseño. Esta ventana posee diversas solapas para visualizar diferentes tipos de información relativa a las fuentes de diseño.
2. Ventana de Procesos. Esta ventana muestra todos los procesos necesarios para la ejecución de cada etapa de diseño.
3. Ventanas de edición. Al hacer doble clic sobre un fichero fuente de la ventana de ficheros fuente se abre una ventana de edición para modificar el fichero (en caso de lenguaje HDL), o bien se ejecuta el programa que permite editar el diseño (en caso de diseños esquemáticos ó máquinas de estado).
4. Ventana de información, situada en la parte inferior. Muestra mensajes de error, aviso o información emitidos por la ejecución de los programas de síntesis, implementación, etc. En la Figura 3.8. Se muestran las ventanas del entorno de ISE

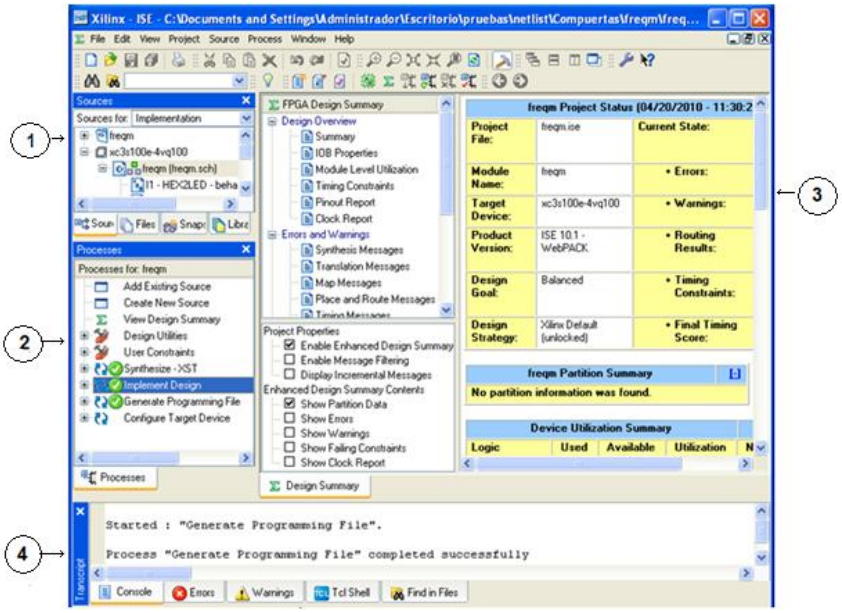


Figura 3.8 Ventana Proyect Navigator

3.5.4 IMPACT

Para la transferencia de la configuración al hardware la herramienta IMPACT permite compilar los archivos .bit en una única imagen para una única cadena en los diferentes modos de configuración del FPGA. También sirve para compilar los distintos conjuntos de datos en una imagen de la Memoria Flash (fichero *.MPM), donde se guardan también los datos de configuración de cada conjunto de datos. Utilizando este software también se programa la memoria Flash. La interfaz IMPACT se muestra en la Figura 3.9

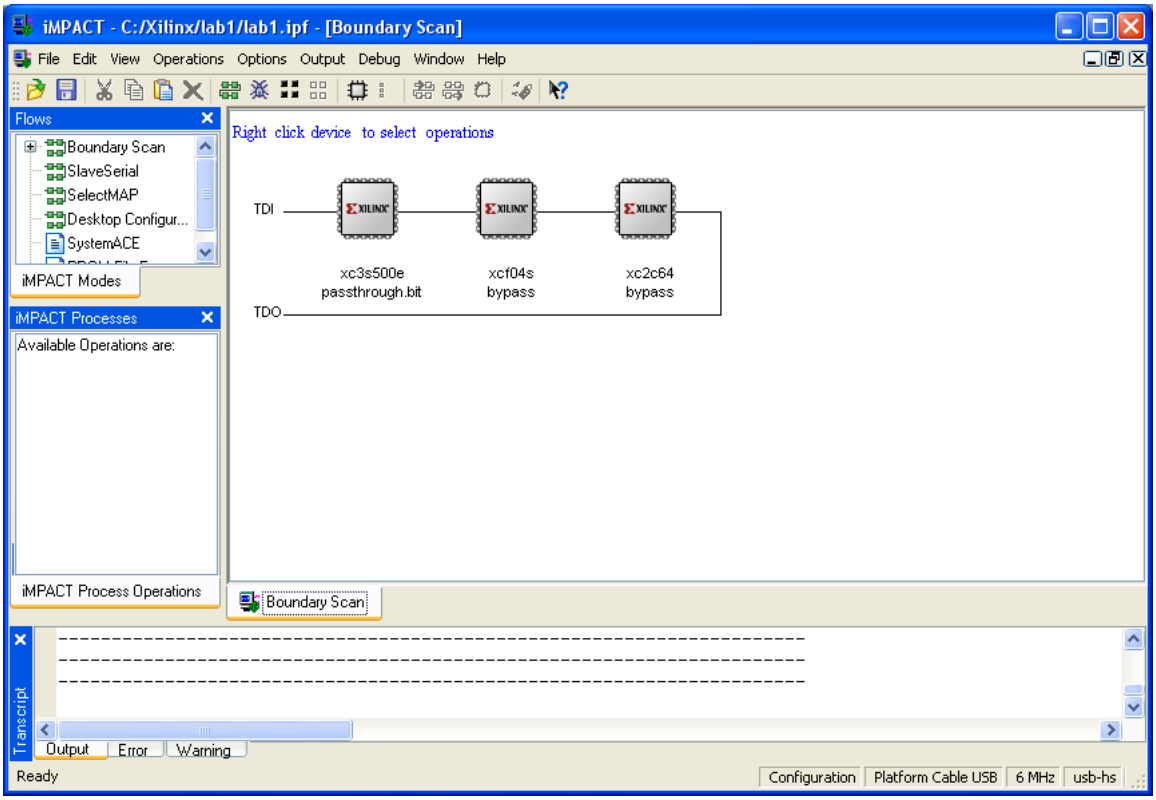


Figura 3.9 Herramienta IMPACT

3.5.5 PlanAhead

PlanAhead agiliza el paso entre la síntesis y el *place* and *route* para dar a los diseñadores un mayor control y una idea de cómo los diseños se aplican para lograr un menor número de iteraciones de diseño. La herramienta permite a los diseñadores utilizar una metodología de diseño jerárquico para minimizar la

congestión de enrutamiento, simplificar la sincronización y la complejidad de la interconexión, y explorar las opciones de aplicación. Con el software PlanAhead, se puede ver los resultados de la aplicación y el momento de analizar fácilmente la lógica crítica, y tomar decisiones dirigidas a mejorar el rendimiento de diseño con *floorplanning*, la modificación de la restricción, y múltiples opciones de la herramienta de implementación.

3.5.6 Xilinx ChipScope Pro

ChipScope Pro está desarrollado por Xilinx, y permite verificaciones en tiempo real para las FPGAs insertando unos núcleos de *debug* software de bajo perfil en el diseño. Una vez esos núcleos (o cores) han sido colocados y ruteados permiten al usuario capturar señales de datos en el chip, en tiempo real. Los datos capturados son enviados fuera del chip para un análisis por algunos de los cables permitidos. La herramienta ChipScopePro consta de tres componentes: el Insertador de Núcleos (Core Inserter), el Generador de Núcleos (Core Generator), y el Analizador ChipScope Pro (ChipScope Pro Analyzer) que se describen a continuación:

- ChipScope Pro Core Generator: Es una interfaz gráfica que permite insertar los cores directamente en el diseño. Se especifica qué señales se van a monitorizar, y luego se procede a hacer el *place & route* del diseño normalmente.
- ChipScope Pro Core Inserter: Inserta automáticamente los cores en el diseño sintetizado del usuario. Esto es bueno para diseños que ya han sido completados o para aquellos usuarios que quieran mantener un mayor control sobre el flujo del diseño.
- ChipScope Pro Analyzer: Proporciona configuración para el dispositivo, un setup del disparo, y una muestra de la traza para los cores. Los cores proporcionan el trigger, control y capacidad decaptura de la traza.[10]

En la Figura 3.10 se muestra un diagrama de bloques del sistema de conexión de la herramienta ChipScope.

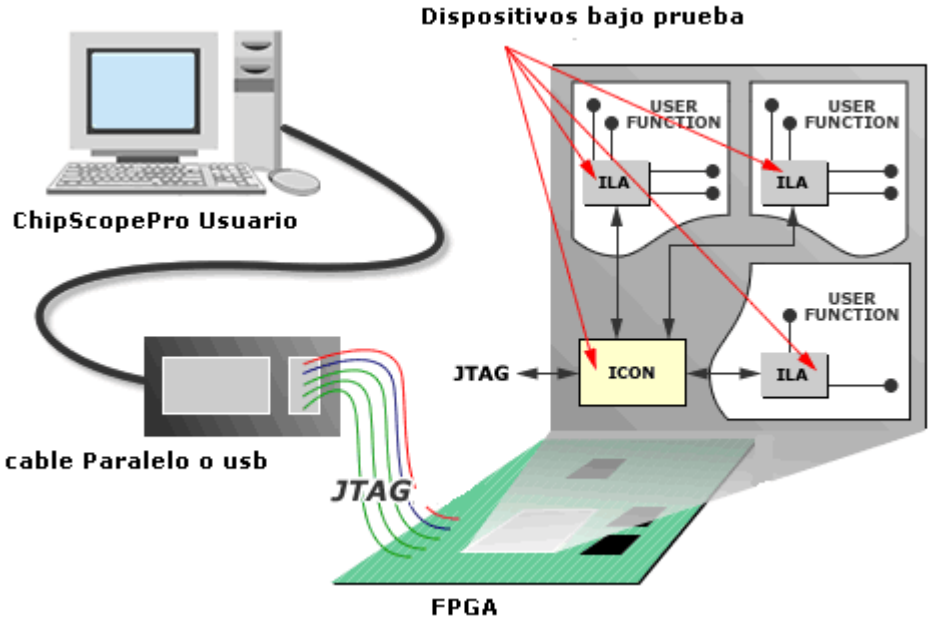


Figura 3.10 Diagrama de Conexión con ChipScopePro

3.5.7 EDK (Embedded Development Kit)

EDK está compuesta por una serie de aplicaciones para la configuración hardware, la codificación y el desarrollo de periféricos específicos. Sin embargo, EDK por sí solo, no es suficiente para construir y cargar un diseño en un FPGA. Todas las tareas de emplazamiento y ruteado, así la descarga al FPGA deben ser realizadas dentro del ISE, la herramienta propia de Xilinx. Así, ambos paquetes son necesarios para el diseño de un sistema *MicroBlaze*. La configuración de un sistema *MicroBlaze* con la herramienta EDK requiere el manejo de varios archivos diferentes.

La interfaz gráfica de usuario XPS (*Xilinx Platform Studio*) dispone de las herramientas que permiten cubrir el flujo de diseño de sistemas empotrados. Este flujo consta de las etapas de descripción de la arquitectura hardware, la síntesis e implementación del circuito controlador, el desarrollo de las aplicaciones software, la compilación de dichas aplicaciones y la programación del FPGA y de la memoria RAM. En una plataforma se integran componentes hardware y software

como periféricos (módulos de propiedad intelectual), según los requerimientos de la aplicación deseada. Existen controladores de memoria, interfaces de comunicaciones (incluso de alta velocidad como *Fast Ethernet*), generadores de señal de reloj, módulos de verificación y depuración, etc. Estos componentes se interconectan a través de sus señales de entrada-salida y de buses fundamentalmente, son altamente configurables presentando una cantidad considerable de parámetros para flexibilizar su operación. Existen *drivers* (manejadores) para estos componentes que incluyen funciones mediante las cuales el desarrollador puede configurar e interactuar con los mismos.

3.5.8 SDK (Software Development Kit)

Xilinx Software Development Kit (SDK) está basado en el estándar de código abierto Eclipse. SDK se incluye en el *ISE Suite Design: Embedded Edition*, pero también está disponible como producto independiente. SDK es el entorno de desarrollo recomendado para proyectos de aplicaciones de software dirigidas a Xilinx integrados *PowerPC* y los procesadores *MicroBlaze*. SDK proporciona un entorno para la creación de plataformas de software y aplicaciones específicas para los procesadores embebidos Xilinx. SDK trabaja con diseños de hardware creado con XPS incorporado herramientas de desarrollo. Un nuevo proyecto de asistente guía al usuario a través de la creación de sus proyectos de software, incluyendo la importación del diseño de hardware y la generación de la *Board Support Package* (BSP).

3.6 System Generator

System Generator para DSP es una plataforma software que usa las herramientas de *The MathWorkMATLAB/Simulink* para representar una visión abstracta de alto nivel del sistema de DSP, y que automáticamente genera el código HDL de la función de DSP desarrollada usando los más optimizados *LogiCOREs de Xilinx*.

De esta forma, System Generator permite modelar directamente mediante un entorno de alto nivel muy flexible, robusto y fácil de utilizar sistemas de DSP y de alto rendimiento para una plataforma hardware específica. Un diseño desarrollado con esta herramienta puede componerse de una gran variedad de "elementos": bloques específicos de System Generator, código de un lenguaje de descripción de hardware tradicional (VHDL, Verilog) y funciones derivadas del lenguaje programación MATLAB. Así, todos estos elementos pueden ser usados simultáneamente, simulados en conjunto y sintetizados para obtener una función de DSP sobre FPGA. El aspecto más interesante de trabajar en MATLAB/Simulink es poder emplear la poderosa herramienta de simulación de sistemas Simulink para realizar la verificación del diseño.

Una de las características más importantes de Xilinx System Generator es que posee abstracción aritmética, es decir, trabaja con representaciones en punto fijo con una precisión arbitraria, incluyendo la cuantización y el sobreflujo. También puede realizar simulaciones tanto en doble precisión como en punto fijo.

Más de 90 bloques de construcción de DSP se encuentran en el Blockset Xilinx DSP para Simulink. Estos bloques son los bloques DSP comunes de construcción tales como sumadores, multiplicadores y registros. También se incluyen un conjunto de bloques de construcción compleja DSP, tales como bloques de corrección de errores, filtros FFT y memorias. Además hay bloques de Xilinx IP que también pueden ser implementados con la debida licencia. En la Figura 3.11 se muestra algunos bloques de System Generator.

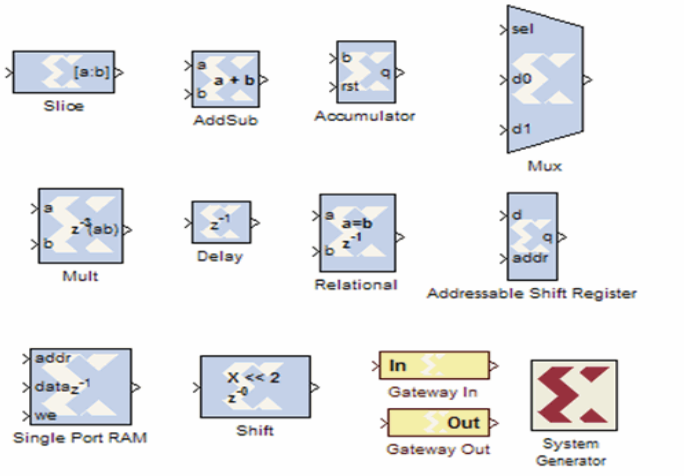


Figura 3.11 Bloques Xilinx Dps para simulink

3.6.1 Requisitos y Recomendaciones para la Instalación

Las recomendaciones de Hardware para System Generator en diferentes sistemas operativos se muestran en las siguientes tablas.

Tabla 3.2 Basado en Recomendaciones para Windows

Recomendaciones	Notas
2.00 GB de RAM	
600 MB de espacio en disco duro	Requerimiento mínimo
Plataforma Xilinx Co-Simulation	Necesarios para el flujo de hardware Co-Simulación

Tabla 3.3 Basado en Recomendaciones para Linux

Recomendaciones	Notas
4 GB de RAM	
600 MB de espacio en disco duro	Requerimiento mínimo
Plataforma Xilinx Co-Simulation	Necesarios para el flujo de hardware Co-Simulación

Los requisitos del sistema operativo y software para la versión 11.1 de System Generator se muestran en la Tabla 3.4.

Tabla 3.4 Requisitos del sistema Operativo Windows y del software

Requerimientos	Notas
Windows XP de 32 bits del sistema operativo Service Pack 2	
Xilinx ISE ® Design Suite versión 11.1	
MathWorks MATLAB,Simulink con Toolbox de Punto-Fijo Versiones 2008a o 2008b	MATLAB se debe instalar en un directorio sin espacios (por ejemplo, C: \ MATLAB \ R2008b) Las señales Punto-Fijo del Toolbox requieren 53 bits

3.6.2 Blockset de Xilinx en Simulink

Después de la Instalación de System Generator en el toobox de simulink se generan nuevos Block set que se describen a continuación.

Xilinx Blockset

Xilinx *Blockset* es una familia de bibliotecas que contienen los bloques básicos del System Generator. Algunos bloques son de bajo nivel, facilitando el acceso al hardware específico del dispositivo. Otros son alto nivel, de ejecución (por ejemplo), procesamiento de señales y algoritmos avanzados de comunicación. Para mayor comodidad, los bloques con una amplia aplicabilidad (por ejemplo, los bloques *Gateway*) son miembros de varias bibliotecas. Cada bloque se encuentra en la biblioteca índice. Las bibliotecas se describen en la Tabla 3.5.

Tabla 3.5 Bibliotecas de Xilinx Blockset

Librería	Descripción
Index	Cada Bloque en el Blockset de Xilinx.
Basic Elements	Bloques de elementos estándar para la construcción de lógica digital.
Communication	Bloques de corrección de errores y moduladores, de uso común en los sistemas de comunicaciones digitales.
Control Logic	Bloques de circuitos de control y máquinas de estado
Data Types	Los bloques que convertir tipos de datos (incluye gateways)
DSP	Bloques de Procesamiento Digital de señales (DSP)
Math	Bloques que implementan funciones matemáticas
Memory	Bloques para la implementación y acceso a memoria.
Shared Memory	Bloques para la implementación y acceso a memoria compartida de Xilinx
Tools	bloques de Herramientas, por ejemplo, la generación de código (bloque System Generator), estimación de recursos, HDL co-simulación, etc

Xilinx Reference Blockset

Xilinx Reference Blockset contiene compuestos de bloques de System Generator que implementan una amplia gama de funciones. Bloques en este Blockset se organizan por su función en diferentes bibliotecas. Las bibliotecas se describen en la Tabla 3.6. Cada bloque en este Blockset es un compuesto es decir, se implementa como un subsistema de enmascarados, con los parámetros que configuran el bloque.

Tabla 3.6 Bibliotecas de Xilinx Reference Blockset

Librería	Descripción
Communication	Bloques de corrección de errores y moduladores, de uso común en los sistemas de comunicaciones digitales.
Control Logic	Bloques de circuitos de control y máquinas de estado
Imaging	Bloques de Procesamineto de Imágenes.
DSP	Bloques de Procesamiento Digital de señales (DSP)
Math	Bloques que implementan funciones matemáticas

Se puede utilizar los bloques de las bibliotecas de referencia Blockset como estan, o como puntos de partida en la construcción de diseños que tienen características similares. Cada bloque de referencia tiene una descripción de su aplicación y los requisitos de recursos de hardware. En la Figura 3.12 se muestra la ubicación de los Blockset de Xilinx en el Toobox de simulink.

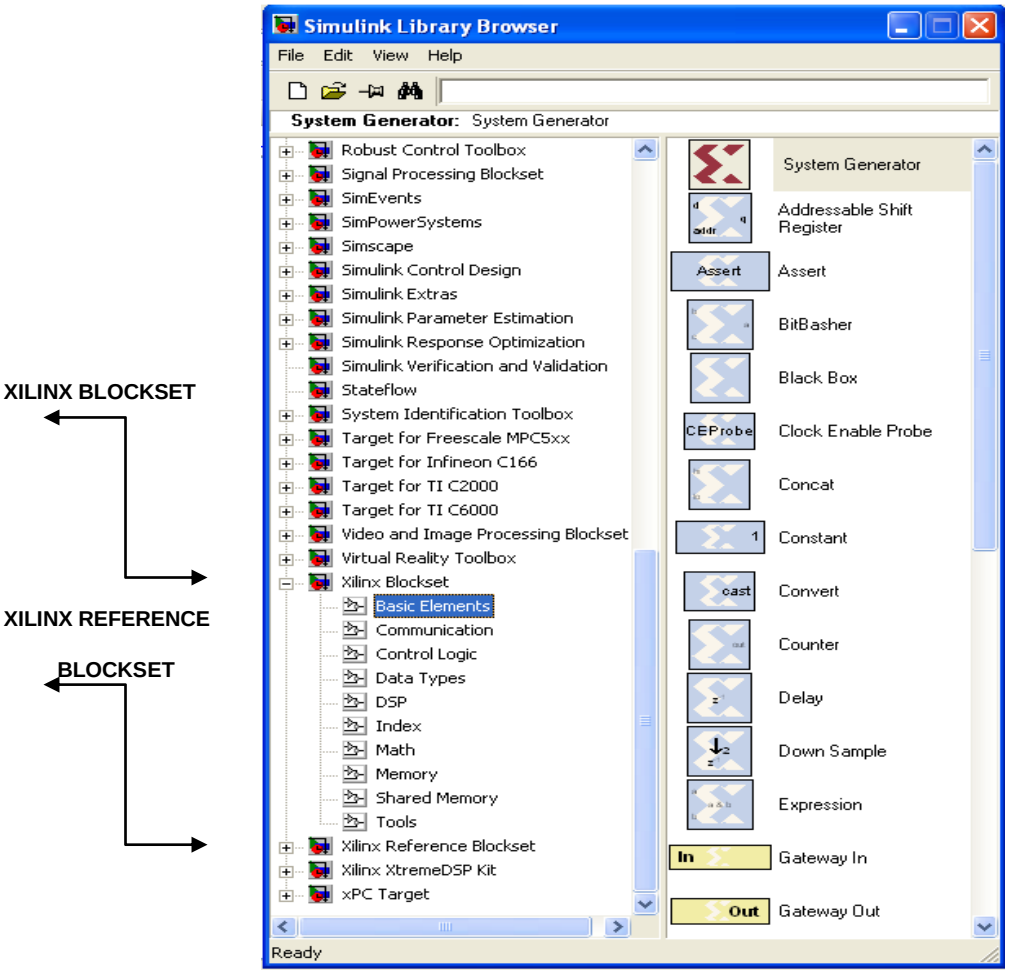


Figura 3.12 Blockset de Xilinx en Toobox de Simulink

3.6.3 Tipos de Señales en System Generator

Los bloques en Xilinx System Generator operan con valores booleanos o valores arbitrarios en punto fijo. Esto es para dar una mejor aproximación a la simulación hardware en simulink. En contraste Simulink trabaja con números de punto flotante de doble precisión. La conexión entre los bloques de Xilinx System Generator y los bloques de Simulink son los bloques “Gateway”.

El Bloque “Gateway In” convierte una señal de doble precision en una señal de Xilinx, y el bloque “Gateway Out” convierte una señal de Xilinx en una de doble precisión. Las señales continuas de Simulink deben ser muestreadas por el bloque “Gateway In”.

La mayoría de los bloques de Xilinx son polimórficos, es decir, son capaces de deducir los tipos adecuados de salida basándose en sus tipos de entrada. Cuando se especifica *full precision* en los parámetros en el cuadro de dialogo del bloque, System Generator elige el tipo de salida para garantizar que no se pierda precisión. La extensión de signo y el relleno de ceros ocurre automáticamente, si es necesario. Esto permite establecer el tipo de salida de un bloque y para especificar como la cuantización y saturación deben ser manejados. Posibilidades de cuantificación objetiva incluyen el redondeo hacia más o menos infinito, dependiendo del signo, o el truncamiento. Las opciones disponibles de saturación son *saturation*, *truncation* y *reporting overflow as an error.*

En la parte de System Generator de un modelo simulink, cada señal debe ser muestreada. Tiempos de muestra deben ser heredados usando reglas de propagación de Simulink, o estableciendo explícitamente la configuración en el cuadro de dialogo de un bloque.

Cuando hay bucles de retroalimentación, System Generator a veces no puede deducir los períodos de la muestra y / o tipos de señales, en cuyo caso se muestra un mensaje de error. *Assert blocks* se deben insertar en los lazos para hacer frente a este problema. No es necesario añadir *Assert blocks* en cada punto en un bucle, por lo general basta con añadir un *Assert block* en el momento de "romper" el lazo.

3.6.4 Bloque System Generator

Cualquier diseño que incluya un bloque de Xilinx debe incluir este bloque. Es el encargado de proporcionar el control del sistema y los parámetros de las simulaciones, e invocar el generador de código. En la Figura 3.13 se muestra el cuadro donde especificar los parámetros para este bloque. Estos son:

- *Compilation:* especifica el tipo de compilación que se producirá cuando se invoque al generador de código.

- *Part:* define la FPGA usada.

- *Target Directory:* nombre del directorio en el que guardar los resultados de la compilación.

- *Synthesis Tool:* indica la herramienta usada para sintetizar el diseño. Las posibles opciones son Synplicity's Synplify Pro, Synplify, y Xilinx's XST.

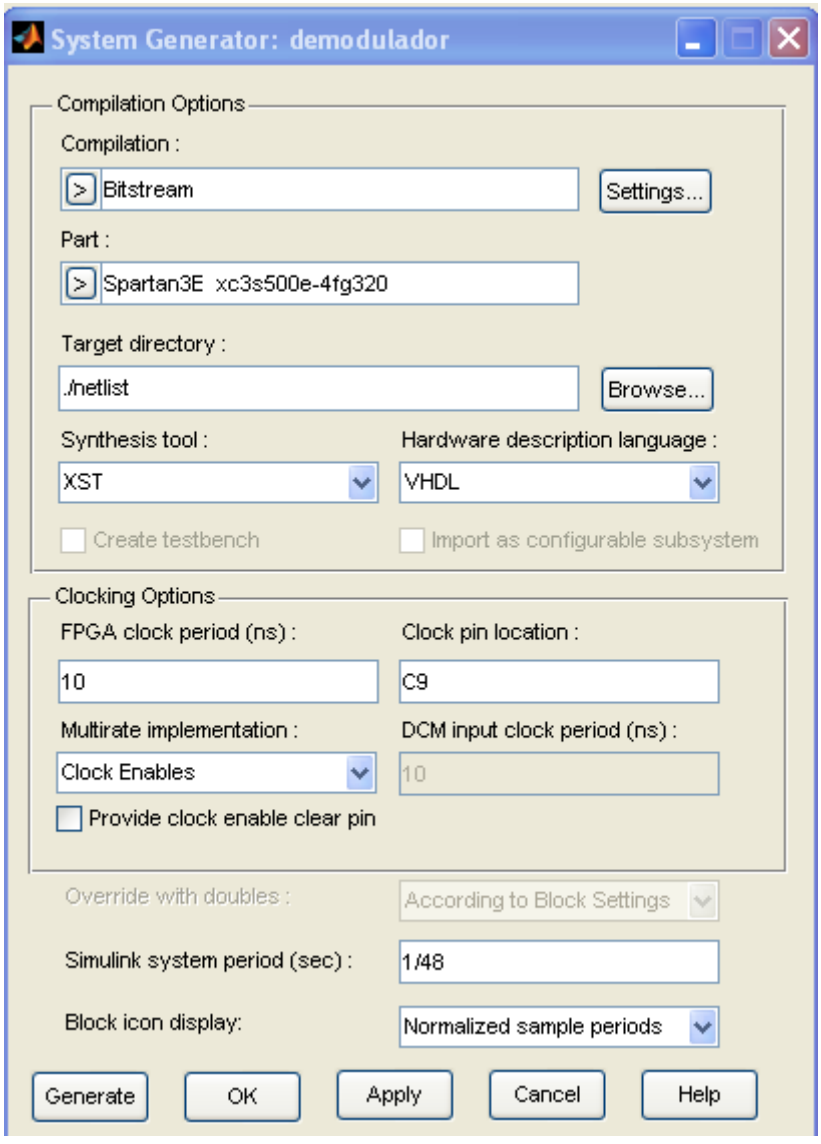


Figura 3.13 Parámetros del bloque System Generator

- *Hardware Description Language:* especifica el lenguaje HDL que se usará para la compilación. Puede ser VHDL o Verilog.

- *FPGA Clock Period:* define el período, en nanosegundos, del reloj hardware. Este valor debe ser un entero.

- *Clock Pin Location:* indica cuál es el pin del reloj. Esta información se le pasa a las herramientas de implementación de Xilinx a través de un archivo (*xcf* o *ngc*) en el que también se incluyen otro tipo de restricciones.

- *Create Testbench:* si está activada esta opción le indica a System Generator que cree un *testbench*.

- *Import as Configurable Subsystem:* le indica a System Generator que haga dos cosas:
  - Que construya un bloque como resultado de la compilación.
  - Que construya un subsistema configurable.

- *Provide clock enable clear pin:* le indica a System Generator que proporcione unbpuerto *ce_clr* en el *top level* que controla la lógica del reloj (*clock wrapper*). La señal *ce_clr* se usa para resetear la lógica de generación de habilitación del reloj. *clock wrapper* es un mecanismo de encapsulación necesario para generar la lógica del reloj y que forma el *top-level* del diseño.

- *Override with doubles:* indica que todos los cálculos deberían realizarse usando precisión doble.

- *Simulink System Period:* define el período de Simulink, en segundos.

- *Block Icon Display:* especifica el tipo de información que será mostrada en los iconos de los bloques. Las opciones disponibles son:

  - *Default:* muestra la información por defecto del bloque.

- *Sample rates:* indica la tasa normalizada para cada puerto de entrada y salida. Esta información suele ser útil para comprobar que no hayerrores.

-*Pipeline stages:* indica la latencia de los puertos de entradas. Esta información no está disponible para algunos bloques.

- *HDL port names:* muestra el nombre de los puertos de entrada y salida.

- *Input data types:* muestras los tipos de datos de las señales de entrada a los bloques. Esta información suele ser útil para resolver errores.

- *Output data types:* muestras los tipos de datos de las señales de salida de los bloques. Esta información suele ser útil para resolver errores.

Tipos de Compilación.

- **HDL**

Ésta es la compilación por defecto. En ella se generan los archivos HDL, NGC y EDIF necesarios para implementar el modelo diseñado. Adicionalmente se crean otros archivos que simplifican este proceso. Todos los archivos creados se guardan en el directorio que se indique en el bloque *System Generator.*

- **NGC**

Si se selecciona esta opción permite a System Generator crear un archivo NGC. Este archivo contiene tanto la información lógica como los requisitos temporales y localización de los pines de los puertos. Esto significa que toda la información relativa al diseño está contenida en un solo archivo.

- **Bitstream**

Esta compilación crea un archivo *bit* que se puede cargar directamente en la FPGA. Este archivo se llama *nombre_cw.bit* y se guarda en el directorio que se indique en el bloque *System Generator*.

- **Hardware Co-simulation**

Este tipo de compilación es la que se utiliza para crear el modelo que se usa en la co-simulación hardware.

- **Timing**

En ocasiones el hardware creado por System Generator puede no cumplir con los requisitos de tiempo. System Generator proporciona una herramienta que permite realizar un análisis temporal para resolver este tipo de conflictos. Mediante este análisis se muestra el camino más lento de la parte hardware diseñada, así como aquellos que no cumplen con los requisitos temporales. Para poder realizar este tipo de análisis System Generator se basa en la herramienta Trace, que forma parte del paquete ISE de Xilinx.[11]

Opciones de Clocking

Como se muestra en la Figura 3.13 cuando se utiliza el bloque de System Generator para copilar el diseño en Hardware, hay tres opciones de clocking para implementación multirate: (1) Clock Enables (por default), (2) Hybrid DCM-CE, and (3) Expose Clock Ports.

- **Clock Enable**

Cuando System Generator compila un modelo en el hardware con la opción *Clock Enable*, System Generator conserva la información de frecuencia de muestreo del diseño de tal manera que las partes correspondientes en hardware funcionan a velocidades adecuadas. System Generator genera frecuencias relacionadas utilizando un reloj en conjunto con el clock enable,

permitiendo una sola frecuencia. El período de cada clock enable es un múltiplo entero del periodo del reloj del sistema.

Dentro de Simulink, ni relojes, ni clock enable se requieren como señales explícitas en un diseño del System Generator. Cuando el System Generator compila un diseño en el hardware, utiliza las frecuencias de muestreo en el diseño para deducir que clock enables son necesarios. Para ello, emplea dos valores especificados por el usuario del Bloque de System Generator: el período del sistema en simulink y el período del reloj del FPGA. Estos números definen el factor de escala entre el tiempo en una simulación de Simulink, y el tiempo en la aplicación de hardware real. El periodo del sistema en simulink debe ser el máximo común divisor (mcd) de los períodos de muestreo que aparece en el modelo, y el período de reloj FPGA, en nanosegundos, del reloj del sistema. Si p representa el período del sistema en Simulink, y c representa el período del reloj del sistema FPGA, entonces si un componente que tiene unidades de tiempo kp en Simulink, k momentos del reloj del sistema (de ahí nanosegundos kc) en el hardware.

Para ilustrar este punto, se puede considerar un modelo que tiene tres periodos de muestra de diferentes componentes en Simulink 2, 3 y 4. El mcd de estos períodos de muestra de los componentes es 1, este valor debe especificarse en el campo de periodo de Simulink en el bloque de System Generator. Suponiendo que el reloj del FPGA es 10ns. Con esta información, el periodo del clock enable correspondiente puede ser determinado en el hardware.

En cuanto a hardware se refiere el clock enable correspondiente para periodos de muestra de simulink 2,3 y 4 como CE2, CE3 y CE4, respectivamente. La relación de cada período de clock enable con el período del reloj del sistema puede ser determinado dividiendo el correspondiente período de los componentes de Simulink por el valor del período del sistema en Simulink. Por lo tanto, los peridos de CE2,CE3 y CE4 son iguales a 2,3,4 respectivamente ya que el valor del periodo del sistema en simulink es1. El la Figura 3.14 se muestra un diagrama de este análisis.

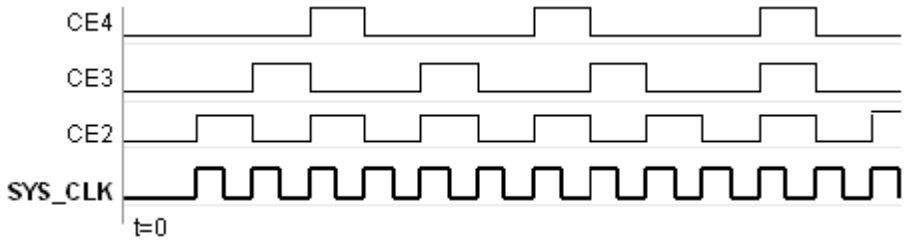


Figura 3.14 Clock Enables para diferentes periodos de muestra.

- **Opción Hybrid DCM-CE**

Si el objetivo es implementar en un FPGA con un *Digital Clock Manager* (DCM), se puede escoger manejar un árbol de reloj con DCM. La opción DCM es deseable cuando *fanout* de alta en las redes de Reloj hacen que sea difícil lograr el cierre de tiempo.

System Generator instancia el DCM en un contenedor de reloj top-level HDL y configura el DCM para proporcionar hasta tres puertos de reloj a diferentes frecuencias para Virtex 4 y Virtex 5 y hasta dos puertos de reloj para la Spartan-3A DSP. Si el diseño tiene más puertos de reloj que el DCM puede soportar, los relojes restantes son compatibles con la configuración de CE (*clock Enable*).

La opción *Hybrid DCM-CE* tiene limitaciones conocidas en algunos bloques de System Generator que se muestran a continuación.

• Clock Enable Probe

• Clock Probe

• DAFIR

• Downsample

• FIR Compiler

• Parallel to Serial

• Time Division De-Multiplexer

• Time Division Multiplexer

• Upsample

- **Opción Expose Clock Ports**

Al seleccionar esta opción, el System Generator crea un contenedor top_level que expone un puerto de reloj para cada frecuencia. Entonces se puede instanciar manualmente un generador de reloj fuera de la unidad de diseño para los puertos del reloj.

3.6.5 Importación de Código

System Generator posee una librería con un gran número de bloques que cumplen una amplia variedad de funciones. Sin embargo puede surgir la necesidad de crear nuevos bloques ya sea para optimizar los recursos hardware o porque no haya ninguno que se adapte a los requerimientos del diseñador. Para estos casos existe la posibilidad de importar código HDL o Matlab.

.

Black Box

Este bloque proporciona una forma de incluir código VHDL o Verilog en el sistema que se esté creando.

Requerimientos del código HDL

Al crear el código HDL que describa al circuito se deben tener en cuenta las siguientes restricciones:

- El nombre de la entidad no debe coincidir con el nombre de ninguna entidad que esté reservada para System Generator (generalmente suelen

llamarse como el bloque anteponiendo el prefijo *xl*, por ejemplo, xlfir, xlregister).

- El nombre de la entidad no debe coincidir con el nombre del modelo Simulink que se está creando. Se recomienda evitar coincidencias entres los nombres de los archivos y las entidades.
- Los puertos bidireccionales no están permitidos en el *top-level* del *Black Box*.
- Para *Black Box* basadas en Verilog los módulos y nombres de los puertos deben seguir el mismo estándar que para el caso de VHDL.
- Cualquier puerto que sea para el reloj o la habilitación del reloj debe ser de tipo *std_logic* (para el caso de Verilog tales puertos deben ser de tipo *nonvector*).

El reloj se trata como sigue:

- Siempre debe aparecer un puerto para el reloj y otro para la habilitación del reloj (para cada reloj hay una habilitación de reloj y viceversa). Aunque para un *Black Box* pueda haber más de un puerto para el reloj, sólo se usará uno para dirigir al resto, que sólo se diferencian en la velocidad.
- Los nombres de los puertos del reloj y habilitación del reloj deben contener las cadenas *clk* y *ce* respectivamente. El nombre de la habilitación del reloj debe ser el mismo que el del reloj pero cambiando la cadena clk por ce. Por ejemplo: *src_clk_1* y *src_ce_1*.
- Ninguno de los puertos (reloj y habilitación) aparecen en el icono del *Black Box*.

CAPÍTULO IV

4. SIMULACIÓN E IMPLEMENTACIÓN DEL RECEPTOR

4.1 Introducción

La implementación de la etapa de recepción en el FPGA se va a realizar con la ayuda del software System Generator en la tarjeta de desarrollo *Spartan 3-E* que se describe más adelante. El receptor va a estar conformado por tres etapas que se muestran en la Figura 4.1.

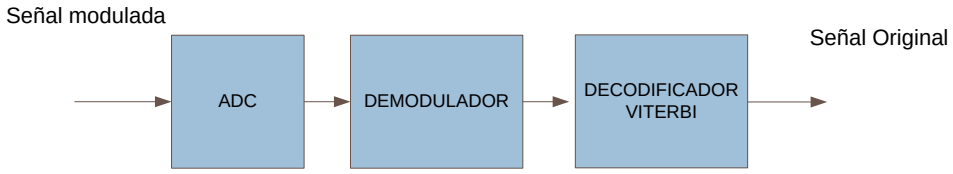


Figura 4.1 Componentes del Receptor

El conversor análogo digital es el encargo de la adquisición de datos hacia el FPGA. Esta etapa del receptor es desarrollada en lenguaje de descripción de Hardware verilog. El demodulador y el decodificador viterbi son desarrollados con elementos de las distintas librerías del System Generator. A continuación se describe la tarjeta de desarrollo *Spartan 3-E* y cada una de las etapas del receptor.

4.2 Tarjeta de Desarrollo Spartan 3-E

Para el desarrollo del receptor la tarjeta seleccionada es la tarjeta *Spartan 3E Starter Board*, manufacturada y distribuida por *Digilent Inc*.

El FPGA que contiene la tarjeta es de la empresa Xilinx (XC3S500E-FG320), este contiene 500K compuertas que son equivalentes a 10476 celdas lógicas. Su arquitectura incluye 20 bloques de 18 Kb de RAM, 20 multiplicadores de hardware de 18 x 18 bits, 4 *Digital Clock manangers* y hasta 232 señales de E/S.

Los periféricos disponibles en la tarjeta son: Memoria Flash 16 MByte (128 Mbit) para aplicaciones, DDR (*double data rate*) SDRAM de 64 MByte (512 Mbit), CLPD XC2C64A, familia *CoolRunner*, Memoria Flash de 4 Mbit para configuración, Memoria Flash 16 Mbits acceso serial, vía SPI(*serial peripheral interface*), una interface de capa física Lan Ethernet 10/100 y un oscilador de 50 Mhz.

Los puertos externos que tiene la tarjeta son dos puertos seriales RS-232 de nueve terminales, un puerto VGA, un puerto PS/2 para teclado o mouse un puerto Ethernet 10/100 Mb/seg, dos puertos para la programación. El principal puerto de programación es un controlador empotrado USB. La tarjeta de desarrollo Spartan-3E tiene algunos accesorios, como se muestra en la Figura 4.2.

Cuatro pulsadores, cuatro switchs, un botón rotatorio, ocho leds y una pantalla LCD de 16 caracteres por 2-líneas. Además tiene dos leds que verifican la alimentación y la configuración de la tarjeta. Tiene una conexión para expansión de 100 pines y tres conectores de 6 pines que se utilizan para ampliar la capacidad de la tarjeta adicionando periféricos externos por estos conectores.
La tarjeta integra un convertidor Digital a Analógico SPI de cuatro salidas (DAC), con resolución de 12 bits y un convertidor Analógico a Digital SPI de dos entradas

(ADC) con resolución de 14 bits y pre - amplificador con ganancia programable todos manufacturados por *Linear Technology*.[12]

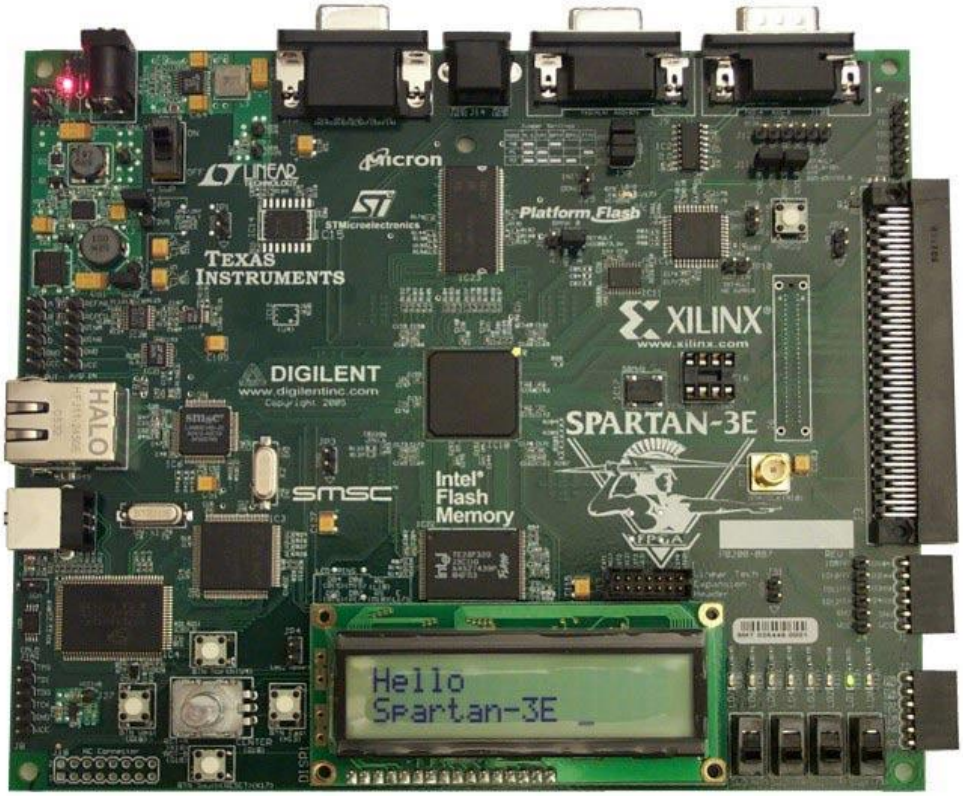


Figura 4.2 Tarjeta de Desarrollo Spartan 3E

4.3 Conversor Análogo Digital

El conversor Análogo Digital en este proyecto se desarrolla con herramientas diferentes a System Generator debido a la no existencia de un bloque en Simulink que permita manipular el ADC integrado en la Tarjeta de desarrollo Spartan 3-E.

El control de este dispositivo periférico está diseñado en el lenguaje de descripción de hardware verilog y para la integración con el resto de sistema diseñado en System Generator se utilizan herramientas del propio System Generator. El desarrollo de cada una de estas etapas se explica a continuación.

La tarjeta de desarrollo Spartan- 3E proporciona una interfaz periférica serial (SPI), dos canales de circuitos de captura analógica que consisten en un preamplificador con ganancia analógica programable y un ADC de 14 bits. El bus

SPI es fullduplex con comunicación serial sincrónica. EL bus SPI establece un esquema Maestro-Esclavos manejado por 3 señales: MOSI (*Master Output,Slave Input*), MISO (*Master Input, Slave Output*) y SCK (Reloj del Sistema). En este caso el maestro es el FPGA y los esclavos son: el ADC y el pre-amplificador .Puesto que los esclavos comparten el bus SPI, además de estas tres señales cuentan con señales propias para su habilitación, de manera que no existan conflictos para un dispositivo mientras otro se está utilizando. En la Figura 4.3 se muestra un esquema del amplificador y el ADC.

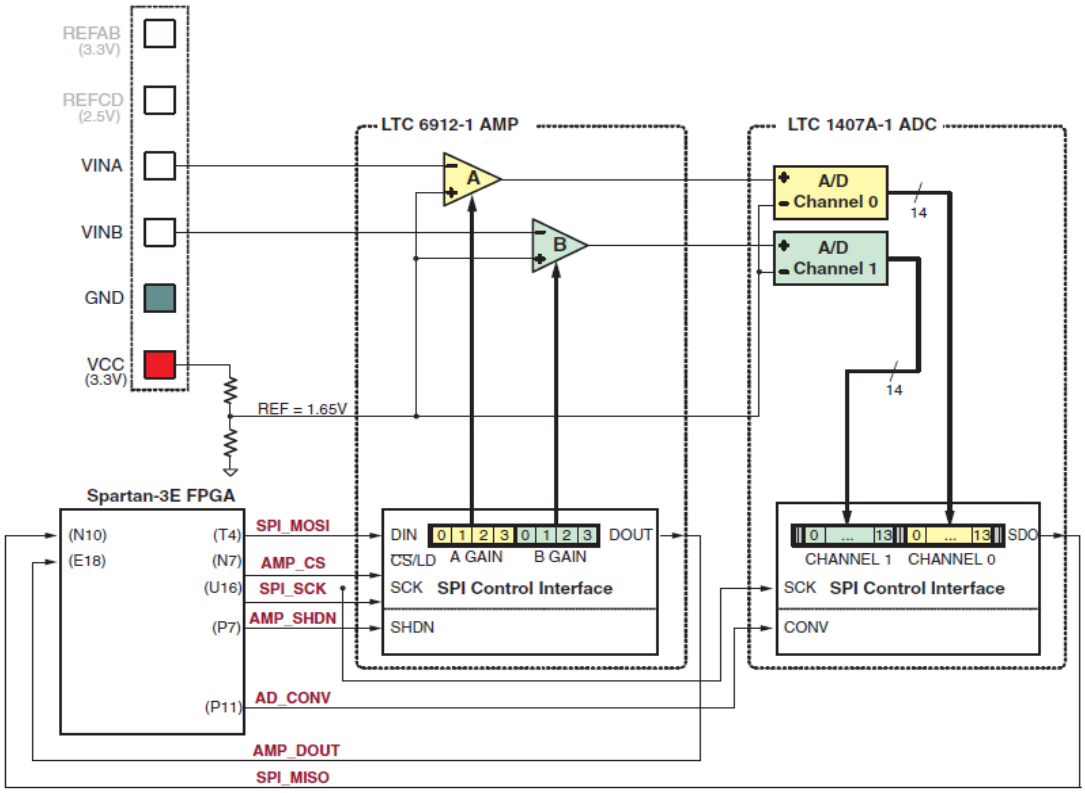


Figura 4.3 Esquema del amplificador y ADC [13]

El circuito de captura analógica de los canales A o B convierte la señal analógica en una representación digital de 14 bits, D [13:0] mediante la ecuación 4.1

$$\mathrm{D}[13:0] = GANANCIA \times \frac{(\mathrm{V_{IN}} - 1.65\mathrm{V})}{1.25\mathrm{V}} \times 8192 \quad (4.1)$$

La GANANCIA es el valor que está establecido en la programación del preamplificador .El voltaje de referencia para el amplificador y el ADC es 1.65V, generado a través del divisor de tensión como se muestra en la Figura 4.3. El alcance máximo del ADC es de ± 1,25 V, en torno a la tensión de referencia, 1.65V.El valor de 8192 se obtiene de la representación de los 14 bits en complemento a dos.

Las señales del bus SPI son compartidas por otros dispositivos en el tablero. Es necesario que otros dispositivos se desactiven cuando se comunica el FPGA con el amplificador o ADC para evitar conflicto en el bus. Tabla 4.1 proporciona las señales y los valores lógicos necesarios para desactivar los otros dispositivos.

Tabla 4.1 Desactivar otros dispositivos en el bus SPI

Señales	Dispositivo Deshabilitado	Valor para Deshabilitar
SPI_SS_B	SPI Serial Flash	1
AMP_CS	Preamplificador programable	1
DAC_CS	DAC	1
SF_CE0	StrataFlash Parallel Flash PROM	1
FPGA_INIT_B	Platform Flash PROM	1

Después de que los dispositivos en el bus SPI son desactivados, para que el preamplificador se active a la señal AMP_CS se le asigna el valor lógico 0. El flaco de subida de la señal de SCK es usado para transmitir una palabra de 8 bits, los 4 bits más significativos se refieren al canal B y los menos significativos se

refieren al canal A. En la tabla 4.2. Se muestra el valor de ganancia y el rango de voltaje de entrada adecuado según los bits enviados.

Tabla 4.2 Configuración de ganancia para el preamplificador programable.

Ganancia	A3	A2	A1	A0	Rango de Voltaje de Entrada	
	B3	B2	B1	B0	mínimo	máximo
0	0	0	0	0		
-1	0	0	0	1	0.4	2.9
-2	0	0	1	0	1.025	2.275
-5	0	0	1	1	1.4	1.9
-10	0	1	0	0	1.525	1.775
-20	0	1	0	1	1.5875	1.7125
-50	0	1	1	0	1.625	1.675
-100	0	1	1	1	1.6375	1.6625

El modulo para la programación del preamplificador se desarrollo en verilog mediante una maquina de estados con un registro de estado de 5 bits que proporciona señales para controlar y configurar el dispositivo.

En el ADC el protocolo de comunicación SPI comienza cuando la señal AD_CONV tiene un valor lógico de 1. El flanco de bajada de la señal SCK es usada para leer 34 bits que consisten en 2 bits sin importancia en la secuencia, 14 bits de datos de canal A del ADC, 2 bits sin importancia en la secuencia, 14 bits de datos del canal B del ADC, y 2 bits más sin importancia en la secuencia. El bit más significativo de los 14 bits del ADC es enviado primero. En la Figura 4.4. Se muestra este comportamiento.

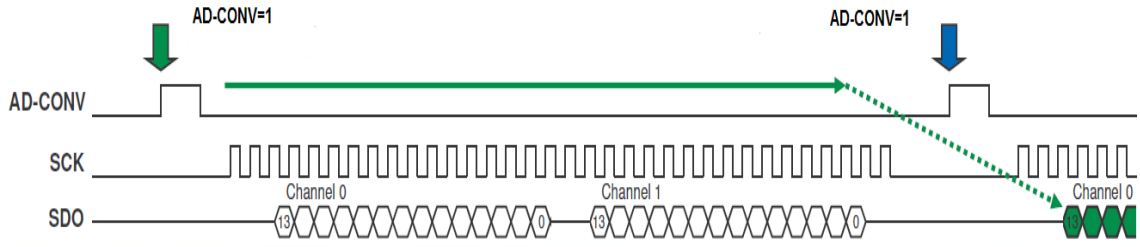


Figura 4.4 Secuencia de datos del ADC [13].

El intervalo de entrada analógica es de ± 1,25 V en relación con 1.65v, cada salida digital de 14 bits en complemento a dos corresponde a una entrada analógica. En la Figura 4.5. Se muestra la relación de entrada analógica con su respectiva la salida digital.

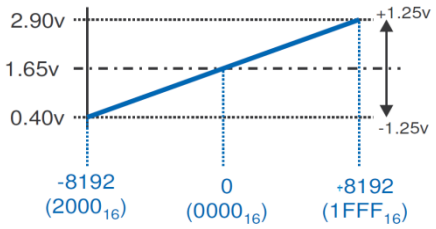


Figura 4.5 Salida Digital en complemento a dos con ganancia 1.

El modulo para la programación del ADC se desarrollo mediante una maquina de estados en verilog con un registro de estado de 7 bits que proporciona señales para controlar y configurar el dispositivo.

El ADC funciona a una frecuencia máxima de reloj en el bus SPI de 50 MHZ y el oscilador de la tarjeta Spartan 3-E es también 50 MHZ. Sin embargo, solo una frecuencia de 12.5 MHZ en el bus SPI produce datos fiables del ADC en la tarjeta. La frecuencia de reloj en el bus SPI en el modulo en verilog del ADC es 12.5 MHZ porque las transiciones en la máquina de estados se producen con una frecuencia de 25 MHZ. Para esto se utiliza un registro que divide la señal del oscilador de 50 MHZ en dos y genera el reloj para la máquina de estados.

El modelo *Top* del diseño es el encargado de enviar los 4 bits para la configuración de la ganancia del preamplificador, estos bits se obtienen a través de tres switch de la tarjeta que equivalen a los bits menos significativos y el bit más significativo es un cero lógico. Un cuarto switch es utilizado para seleccionar el canal del ADC a utilizar. En la Figura 4.6 se observa la organización del diseño. Todos los componentes se desarrollaron en verilog.[14]

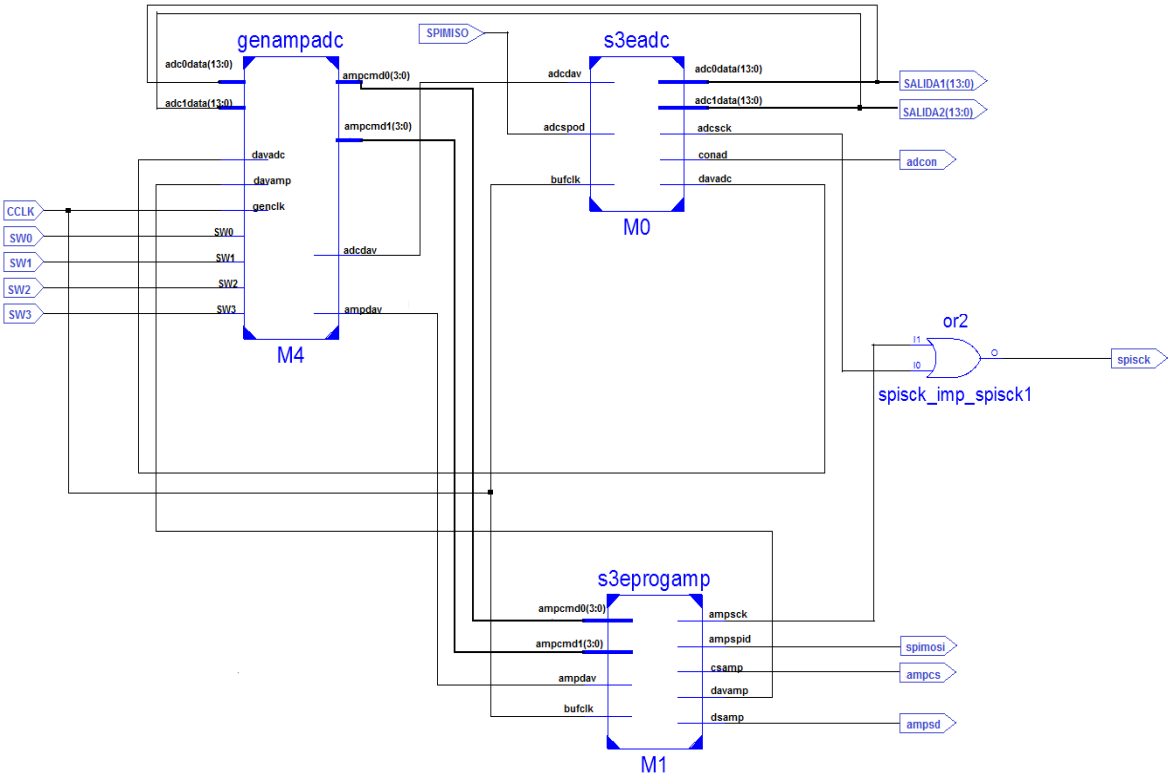


Figura 4.6 Organización del Diseño

4.3.1 Integración del componente ADC a System Generator

Como se menciono en el capítulo 3 para importar un diseño en HDL al esquema realizado en System Generator se utiliza el bloque BACK BOX del blockset de Xilinx en simulink.

Cuando este bloque es añadido a un archivo de simulink se despliega la ventana en la Figura 4.7 que permite seleccionar el archivo HDL, siempre se debe seleccionar el archivo TOP de un diseño en este caso se selecciona el modulo top del ADC y automáticamente se genera un archivo M-FILE que establece la asociación del modulo top del ADC con el bloque BACK BOX y se crear un bloque como se muestra en la Figura 4.8.

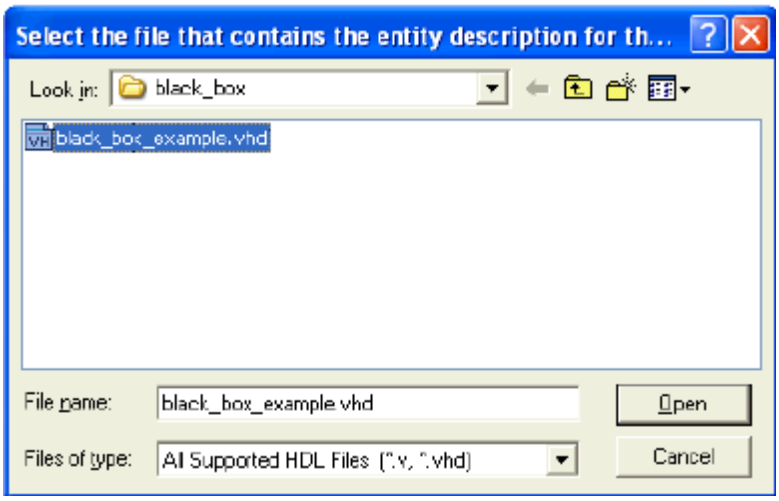


Figura 4.7 Ventana para le selección del archivo HDL

Es necesario que el archivo HDL esté en la misma carpeta donde se crea el diseño en simulink para evitar conflictos en la generación del archivo M-FILE del bloque.

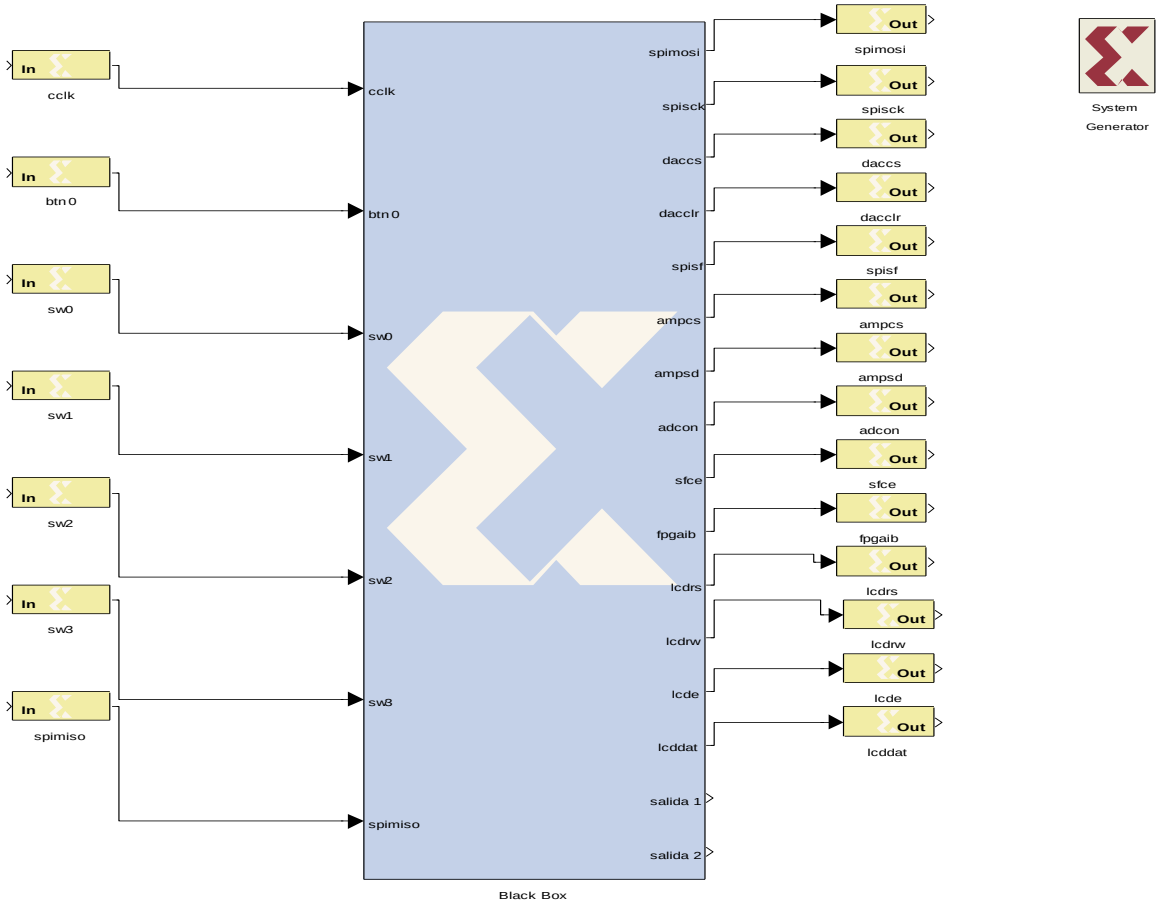


Figura 4.8 BLACK BOX del modulo top del ADC

Para añadir el resto de componentes verilog se edita el final de archivo M-FILE. Se usa la sentencia *this_block.addFile* donde los argumentos son la ubicación y el nombre de los archivos. En la Figura 4.9 se muestra un archivo M-FILE con los componentes verilog añadidos.

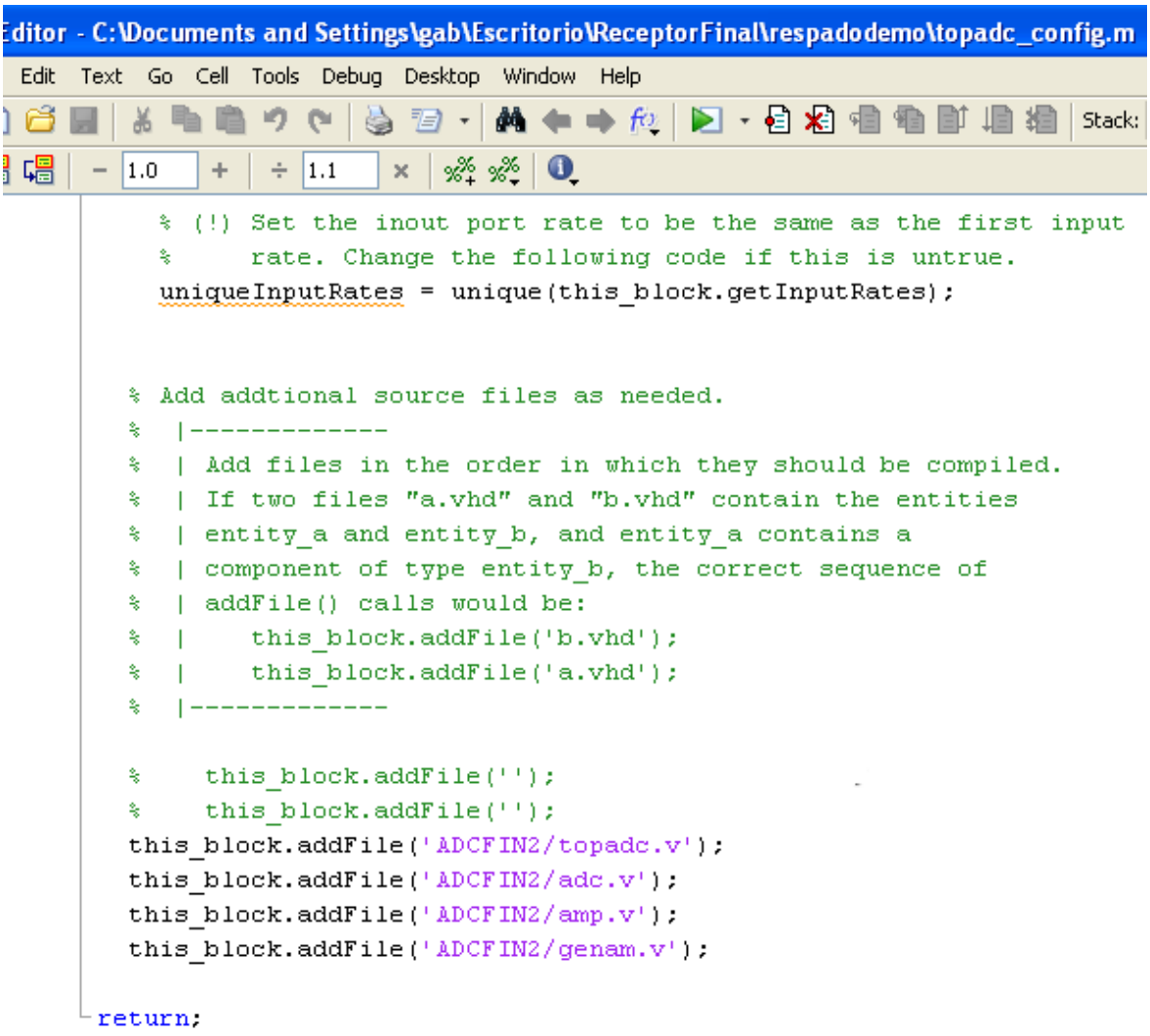


Figura 4.9 Archivo M-FILE del BLACK BOX del ADC

Las entradas del BLACK BOX mediante los bloques *Gateway In* están configuradas como booleanos y para su implementación en la tarjeta los pines se configuran directamente el archivo .ucf (*User Constraints File*) una vez generado el proyecto de ISE después de la compilación en System Generator. La señal digitalizada por el ADC se enlaza con los demás componentes de System

Generator. El resto de salidas del BLACK BOX son puertos de salida del FPGA que se configuran mediante el bloque *Gateway Out*.

En la Figura 4.10 se muestra la configuración para la compilación del modelo . El proyecto se genera en lenguaje VHDL porque ofrece facilidades a la hora de modificar la estructura de los componentes de System Generator. En *Part* se selecciona el FPGA Xc3s500e que es el FPGA de la tarjeta de desarrollo Spartan 3-E. Para generar el ejecutable se debe activar el botón *generate*, el resto es automático.

La entrada de Reloj que tiene el BLACK BOX se modifica una vez generado el Proyecto en ISE, en el *PORT MAP* que corresponde al BLACK BOX, relacionando esta entrada de reloj con la señal de reloj de los demás componentes del diseño.

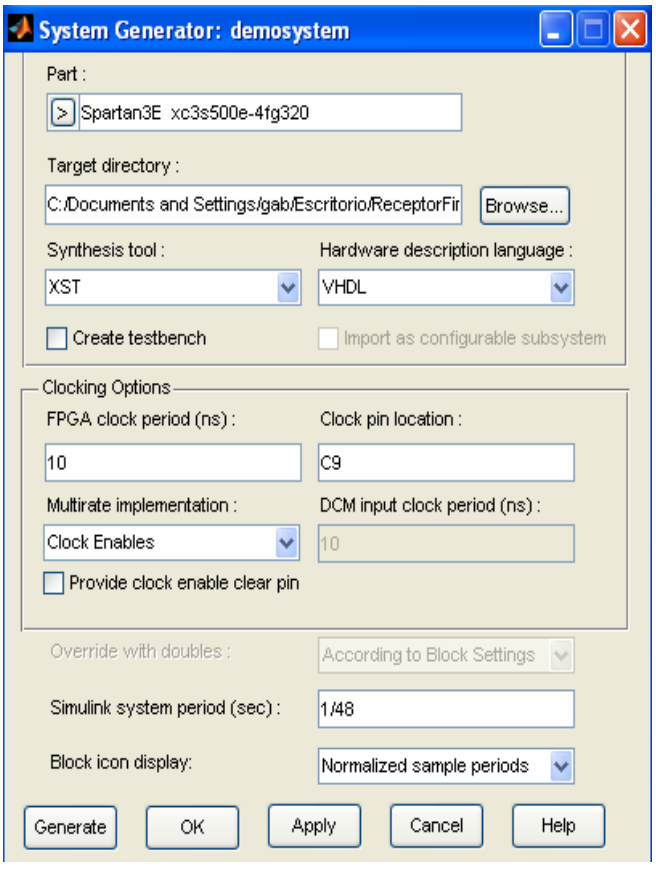


Figura 4.10 Ventana de Configuración de System Generator.

4.3.2 Comprobación del funcionamiento del ADC en tiempo real mediante Chipscope Pro Analyzer.

El bloque de Chipscope Pro se encuentra en el blockset de Xilinx en simulink. En la Figura 4.11 se muestra la conexión del bloque chipscope con el BLACK BOX del adc. El bloque chipscope se encarga de insertar núcleos en el diseño para analizar la respuesta en tiempo real de diferentes señales. En este caso se analiza la señal digital generada por el ADC para comprobar si se está digitalizando correctamente y a la vez conocer si la señal entra a System Generator adecuadamente.

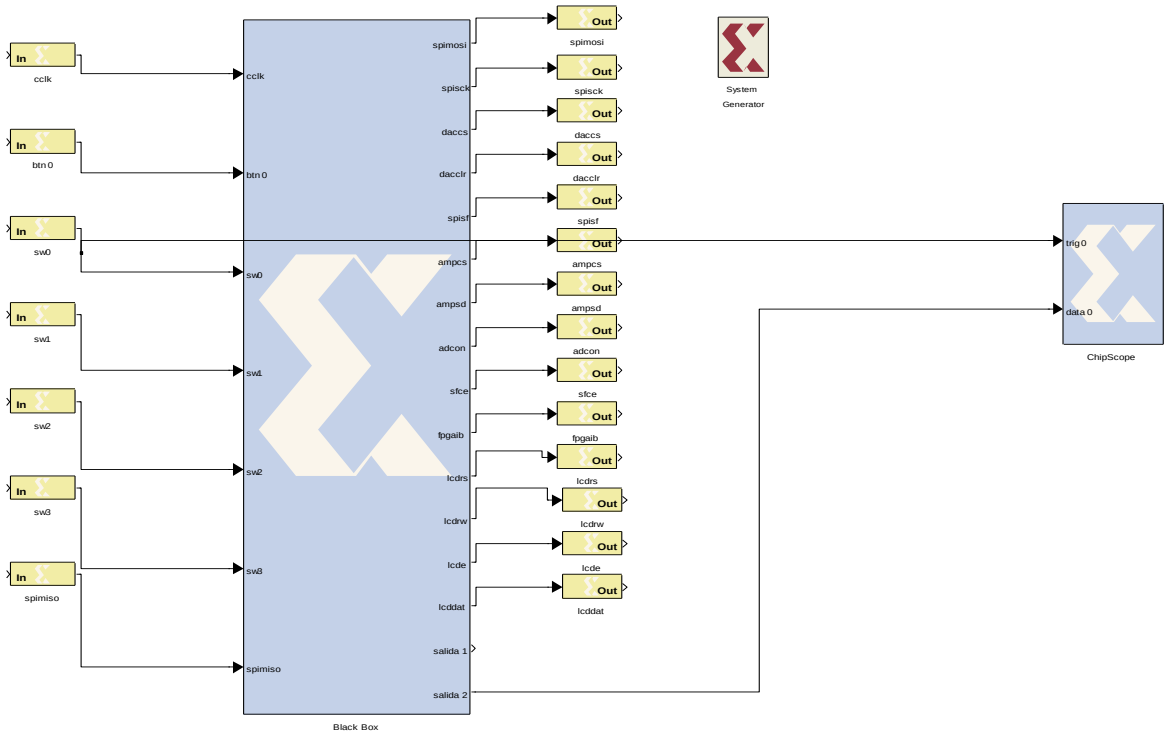


Figura 4.11 Conexión bloque Chipscope

Este Bloque se puede configurar para analizar varias señales de datos con varias señales de disparo (trigger) además permite configurar la cantidad de muestras capturadas en el búfer. Para comprobar el funcionamiento del ADC se utiliza solo una señal de datos y una señal de disparo que está relacionada a un switch de la

tarjeta Spartan 3-E. En la Figura 4.12. Se muestra la configuración del bloque Chipscope para el análisis del ADC.

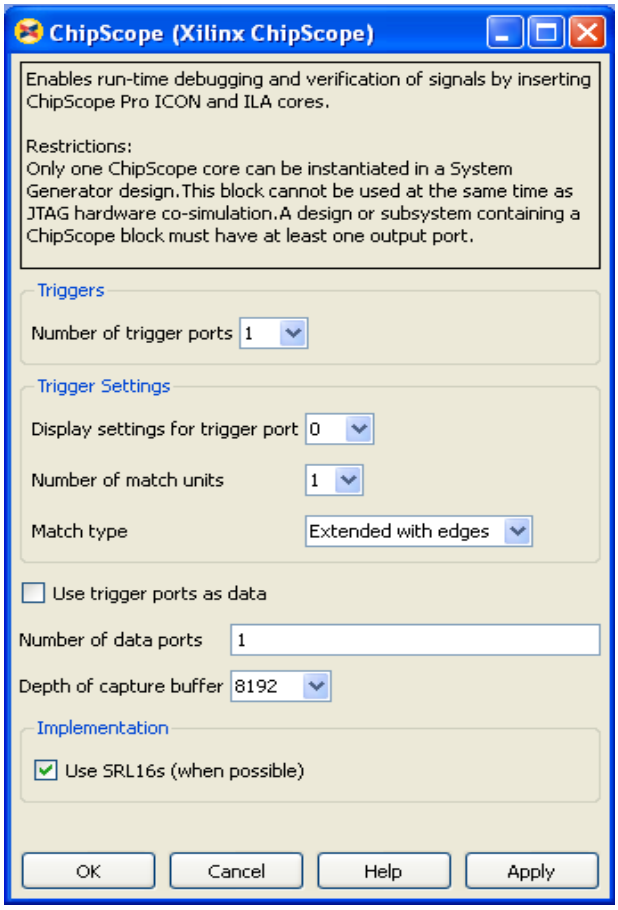


Figura 4.12 Ventana de configuración del bloque ChipScope

Una vez que está grabado el diseño en la tarjeta y el ordenador está conectado mediante el cable de programación USB a la misma, se debe arrancar Chipscope Pro Analyzer. Cuando se abre el puerto una ventana muestra los componentes que tiene la tarjeta, lo que demuestra que la comunicación entre el ordenador y la tarjeta es la adecuada. Como se muestra en la Figura 4.13.

ChipScope Pro Analyzer después de comprobar si hay núcleos insertados en el diseño, abre algunas ventanas que sirven para el análisis del diseño. La ventana *trigger setup* sirve para configurar las condiciones para la señal del disparo y la cantidad de muestras que representan una señal. En la Figura 4.14 se muestra la configuración de la ventana trigger setup para el análisis del ADC.

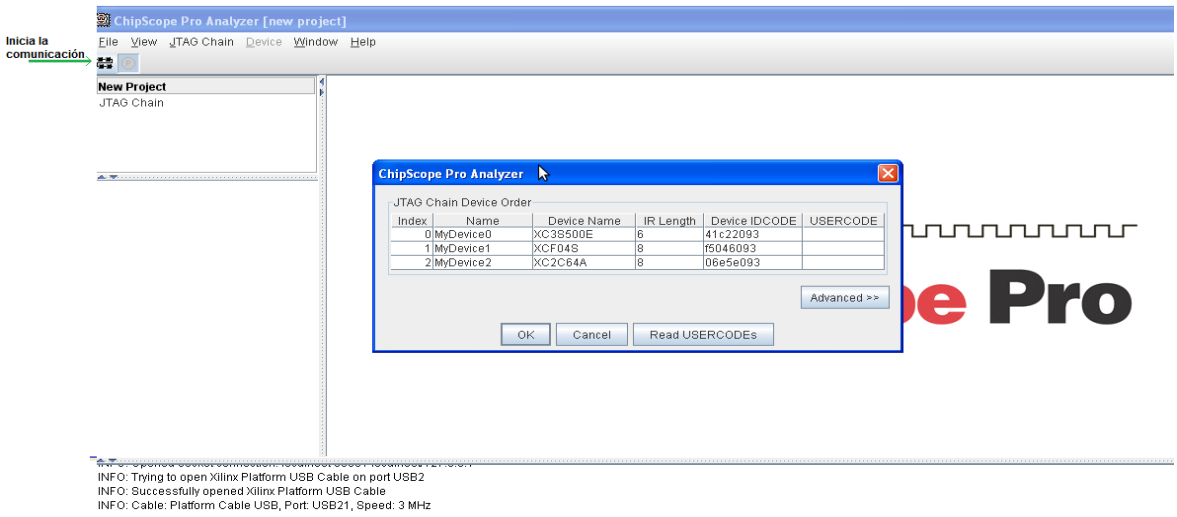


Figura 4.13 Ventana ChipScope Analyzer

La ventana *Waveform* muestra la variación de cada bit en el puerto de datos con la cantidad de muestras establecidas en la ventana *trigger setup*. En la Figura 4.15 se muestra las variaciones de los 14 bits de la señal del ADC.

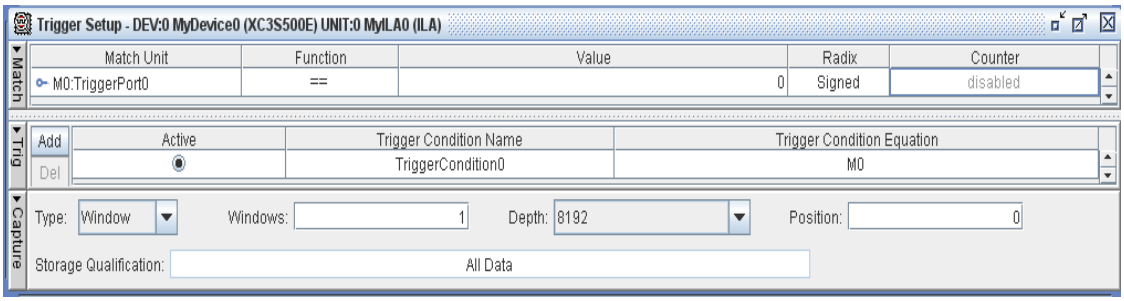


Figura 4.14 Ventana Trigger Setup

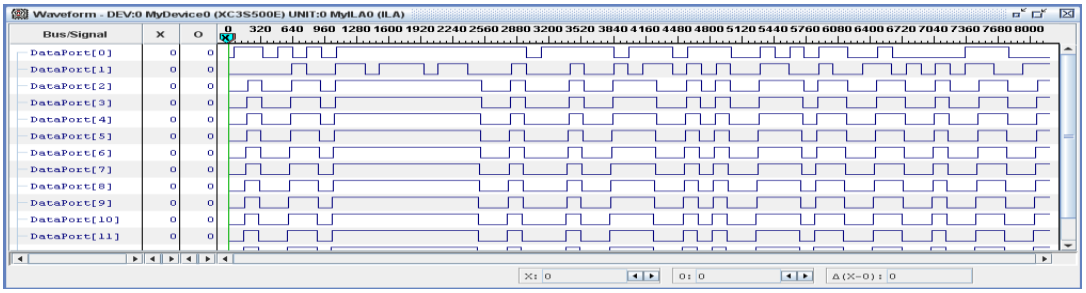


Figura 4.15 Ventana Waveform

La ventana *Bus Plot* muestra una gráfica de la variación de un bus de datos con el tiempo o con otro bus de datos. Para crear un bus se seleccionan puertos de datos que se quieren representar y con *click* derecho en el mouse se despliega una lista que permite crear un nuevo bus como se muestra en la Figura 4.16.

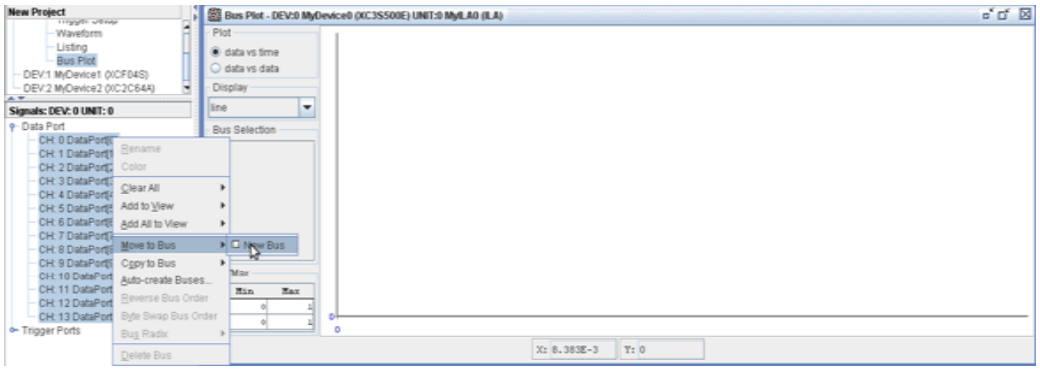


Figura 4.16 Ventana Bus Plot

La gráfica de la Figura 4.17 representa la señal digitalizada obtenida del ADC a una frecuencia de 10 Khz con ganancia en el preamplificador de 1. La señal es una representación con signo (en complemento a dos) que tiene un valor de voltaje de entrada entre 0.4 y 2.9.

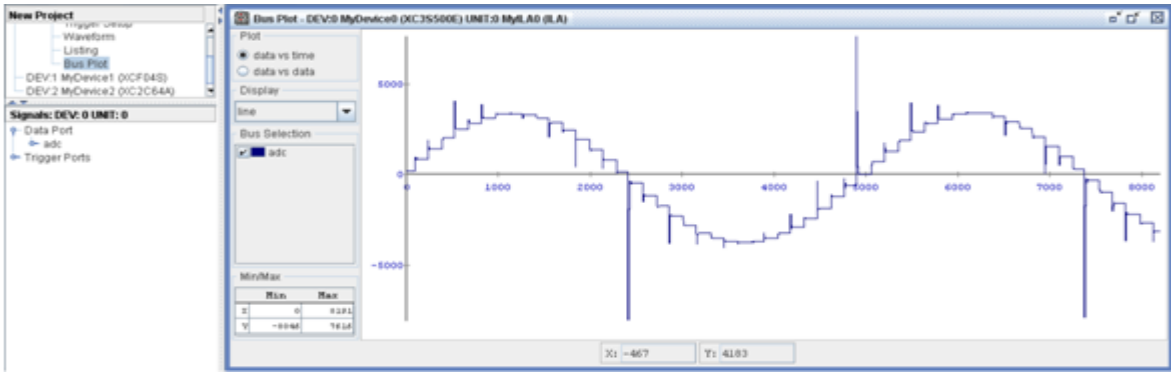


Figura 4.17 Señal digitalizada con ganancia 1.

La gráfica de la Figura 4.18 representa la misma señal de la Figura 4.17 el mismo voltaje de entrada y la misma frecuencia pero con una ganancia de 2.

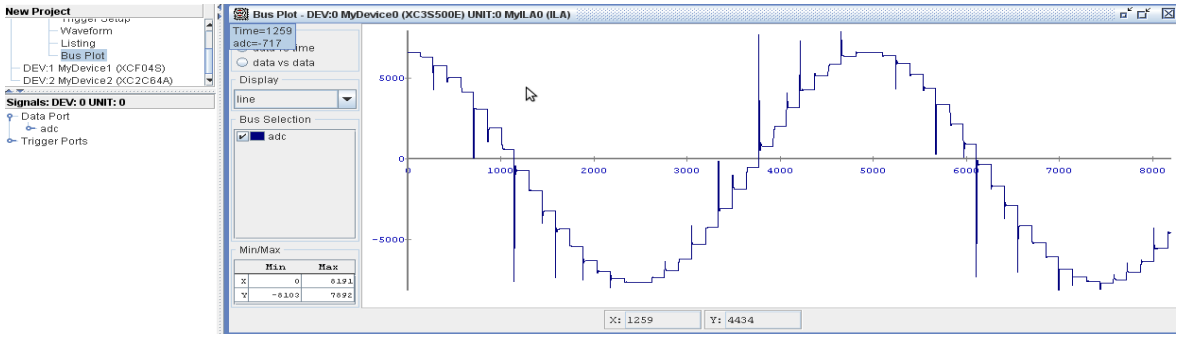


Figura 4.18 Señal digitalizada con ganancia 2.

La gráfica de la figura 4.19 representa la misma señal de la figura 4.17 y 4.18 el mismo voltaje de entrada y la misma frecuencia pero con una ganancia de 5.
La señal presenta una saturación debido a que el voltaje no está en el rango que se muestra en la tabla 4.2.

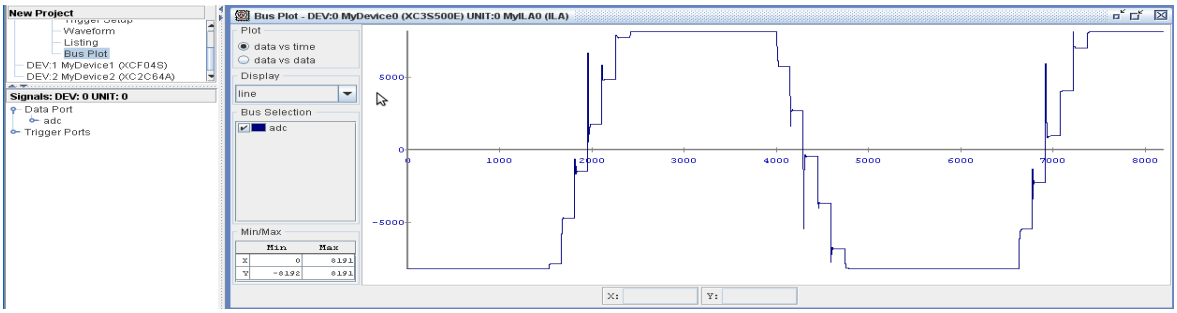


Figura 4.19 Señal digitalizada con ganancia 5.

4.4 Demodulador

En este proyecto se desarrollan dos tipos de demoduladores en System Generator, el BPSK y el QPSK con las características teóricas expuestas en Capítulo I.

4.4.1 Demodulador BPSK

El demodulador BPSK diseñado es un demodulador coherente sin un circuito de estimación de fase porque se conoce la fase de la señal portadora que entra al receptor. La Figura 4.20 muestra los componentes en System Generator del demodulador BPSK.

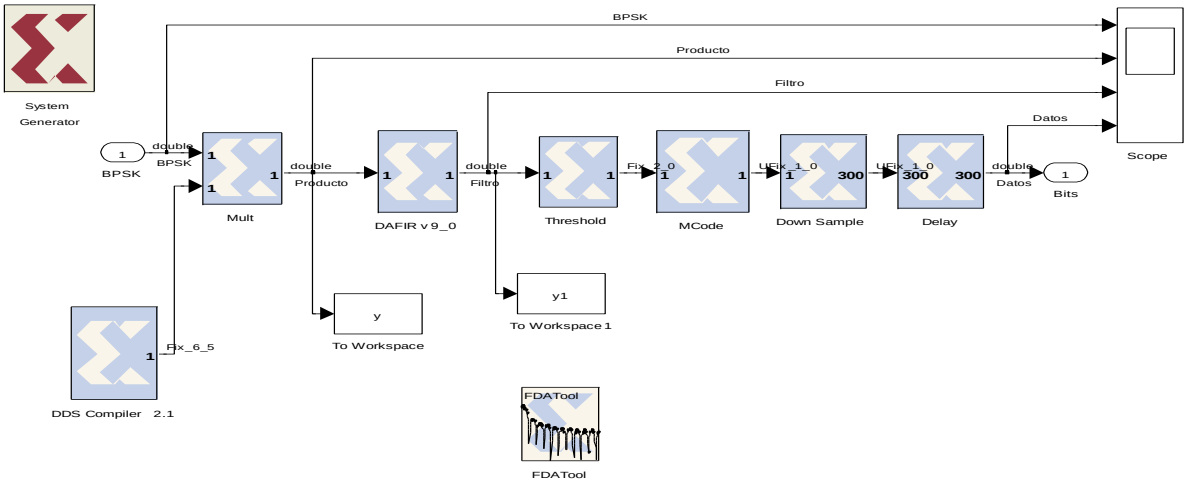


Figura 4.20 Componentes del demodulador BPSK en System Generator.

El bloque *Mult* es el encargado de multiplicar la señal BPSK por la portadora, para tener una mejor precisión en la multiplicación, se configura el bloque para que reconozca todos los bits que entran al mismo.

La portadora es generada por el Bloque *DDS Compiler*. Este es un bloque que tiene la opción de generar una señal seno, coseno o seno y coseno y pueden ser positivas o negativas. Para este demodulador se utiliza la señal seno negativa a una frecuencia de 500 Khz para que la portadora esté en fase con la señal BPSK que ingresa al receptor. En la Figura 4.21 se muestra la ventana de configuración del bloque DDS Compiler.

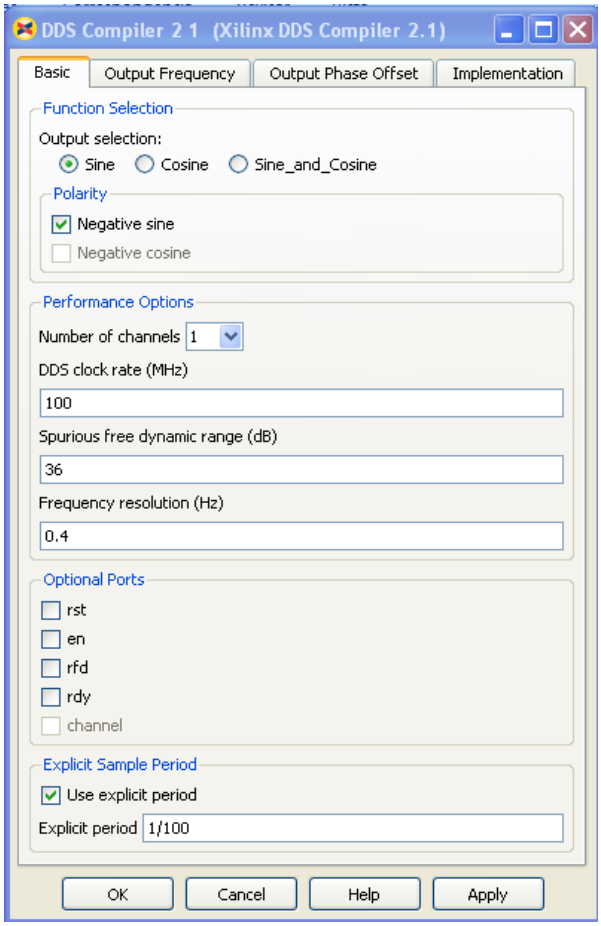


Figura 4.21 Ventana de configuración del bloque DDS Compiler.

El siguiente componente del demodulador es un filtro pasa-bajo que se encarga eliminar el rizado de alta frecuencia de la señal. Para implementar el filtro digital se utiliza el bloque *DAFIR* que es un bloque que recibe los coeficientes del filtro digital diseñado, además permite configurar el número de bits que se utilizan en el proceso.

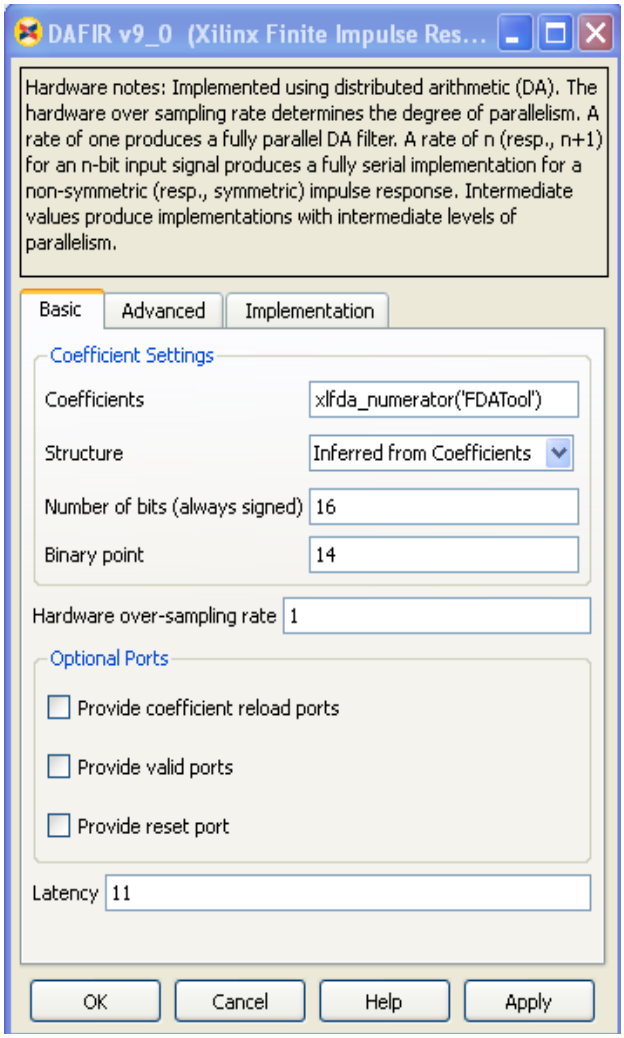


Figura 4.22 Ventana de configuración del bloque DAFIR.

Como se muestra en la Figura 4.22 en el campo *coefficients* se utiliza el comando *xlfda_numerator('FDAtool'),* este comando permite exportar los coeficientes de la herramienta FDAtool de MATLAB a este bloque de System Generator.

FDAtool es un herramienta que facilita el diseño de filtros digitales, para el demodulador se utilizo un filtro donde la frecuencia de corte sea dos veces la frecuencia de la señal portadora. En la Figura 4.23 se muestra la configuración del Filtro mediante la herramienta FDAtool.

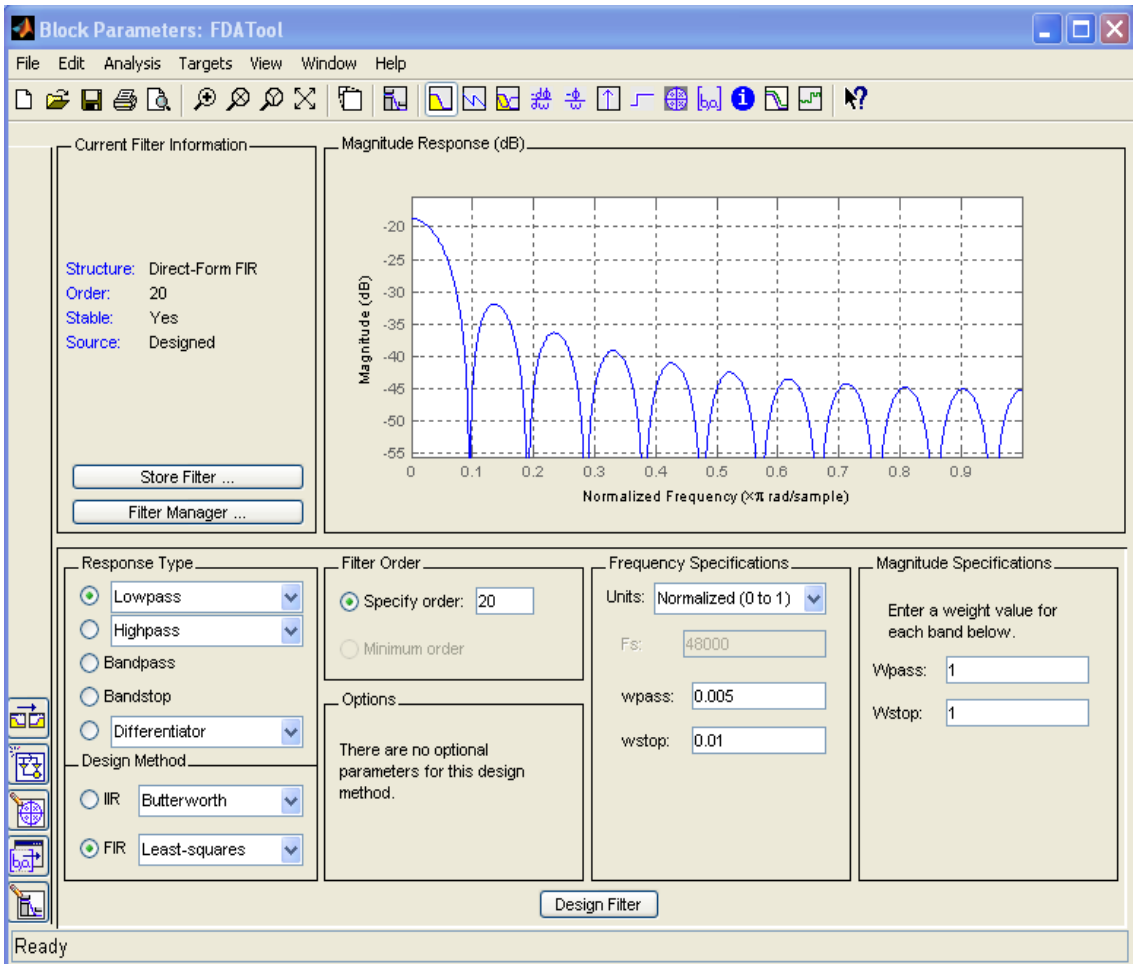


Figura 4.23 Ventana de la Herramienta FDATool.

Una vez que la señal pasa el filtro pasa–bajos el siguiente elemento del demodulador es el bloque *Threshold* que es un detector de umbral que establece 1 si la señal es mayor que 0 y -1 si la señal es menor que 0. Este bloque permite que la señal tenga mejor nivel referencia después de pasar por el filtro lo que facilita la elección de los datos en el decisor.

Para la implementación del decisor se utiliza el bloque *Mcode* que permite ingresar código de MATLAB al diseño en System Generator. En este caso en el bloque *MCode* se implementa una función que establece que los valores de entrada al bloque mayores a 0 sean igual a 1 caso contrario sean igual a cero.

Como se muestra en la Figura 4.24 se puede editar el archivo M-File o se puede buscar un archivo M-File diferente.

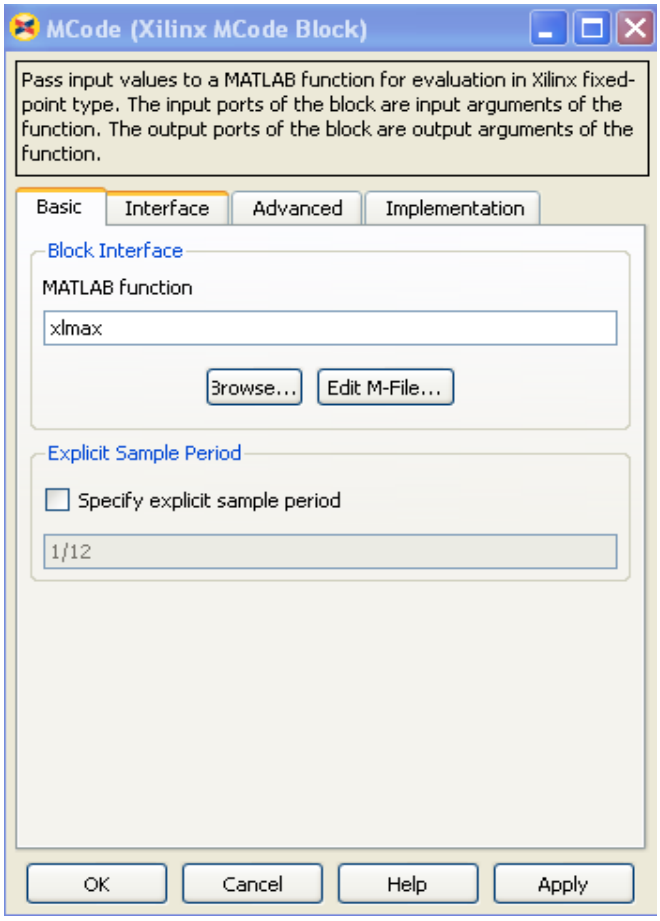


Figura 4.24 Ventana de Configuración del bloque MCODE

Después de pasar por el bloque *MCode* se obtiene la señal demodulada y se utiliza el bloque *Down Sample* para reducirla cantidad de muestras obtenidas después del procesamiento.

Para la simulación que se muestra en la Figura 4.25 se utilizó la herramienta *scope* de *Simulink* con la conexión que se muestra en la Figura 4.20.

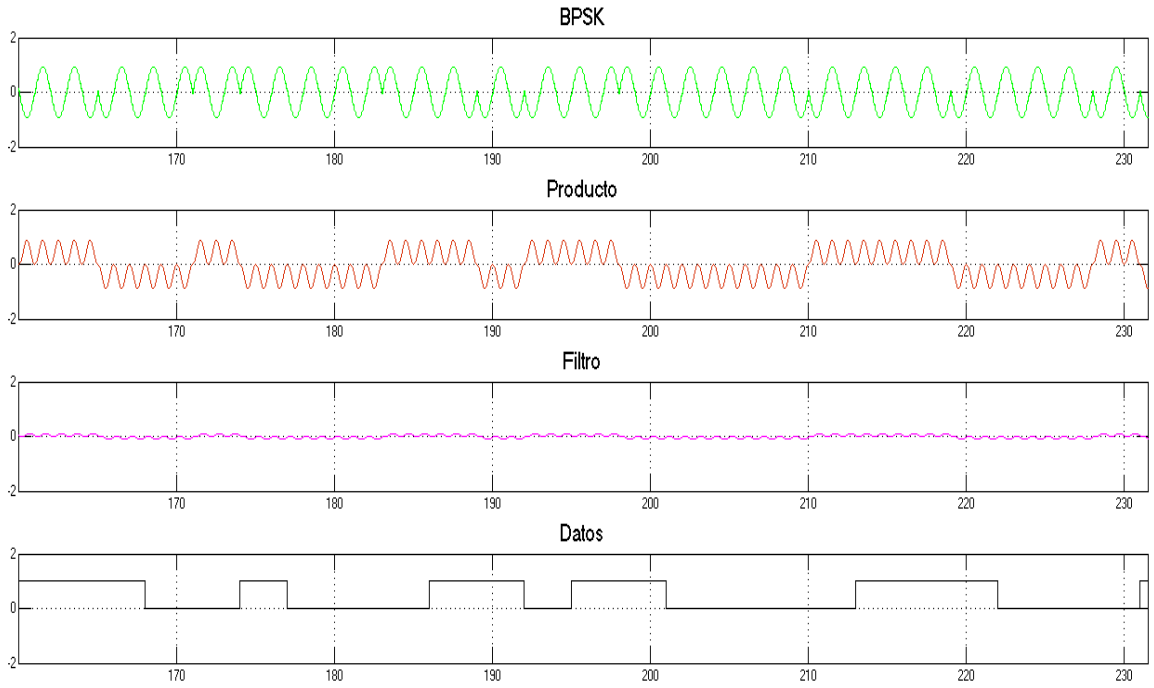


Figura 4.25 Simulación de las Etapas del demodulador BPSK en Simulink.

Implementación del demodulador BPSK

Para la compilación del demodulador y la generación del código HDL la configuración del bloque de System Generator se muestra en la Figura 4.10. En esta etapa del receptor no se configura ningún tipo de entrada o salida del FPGA en el archivo .ucf. Para la verificación del funcionamiento en tipo real de las diferentes etapas del demodulador se utiliza el bloque ChipScope con la configuración de la Figura 4.12 y se genera una señal BPSK como se muestra en la Figura 4.26.

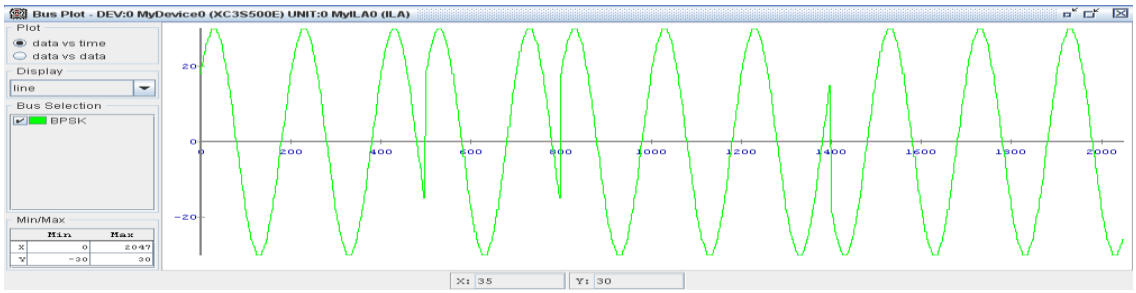


Figura 4.26 Señal modulada BPSK en tiempo real.

En la Figura 4.27 se muestra el resultado del producto de la señal BPSK y la portadora. Esta señal es un bus de datos de 10 bits en complemento a dos.

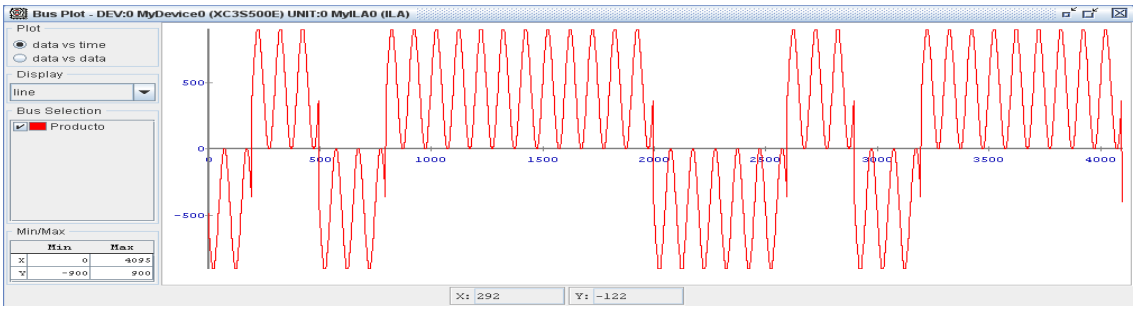


Figura 4.27 Producto de la señal BPSK por la señal portadora en tiempo real.

La Figura 4.28 muestra la señal procesada después del filtro- pasa bajo. Esta señal es un bus de datos de 20 bits en complemento a dos.

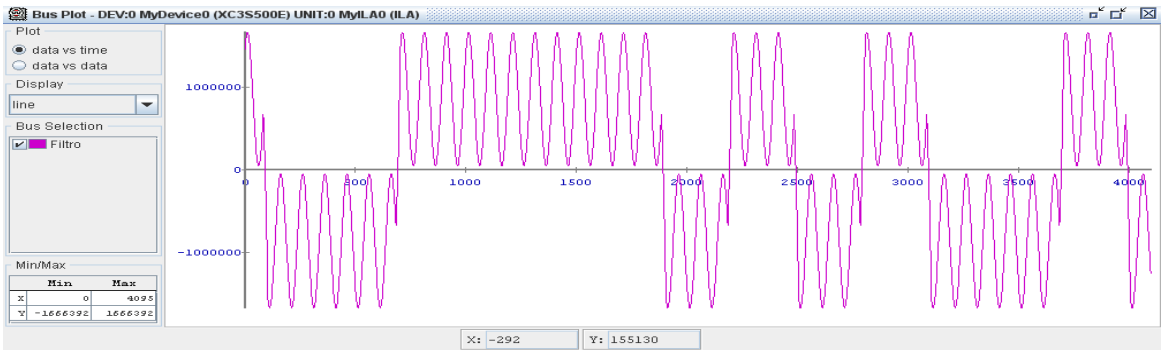


Figura 4.28 Respuesta del Filtro pasa-bajos en tiempo real

El resultado de la demodulación en tiempo real se muestra en la Figura 4.29.

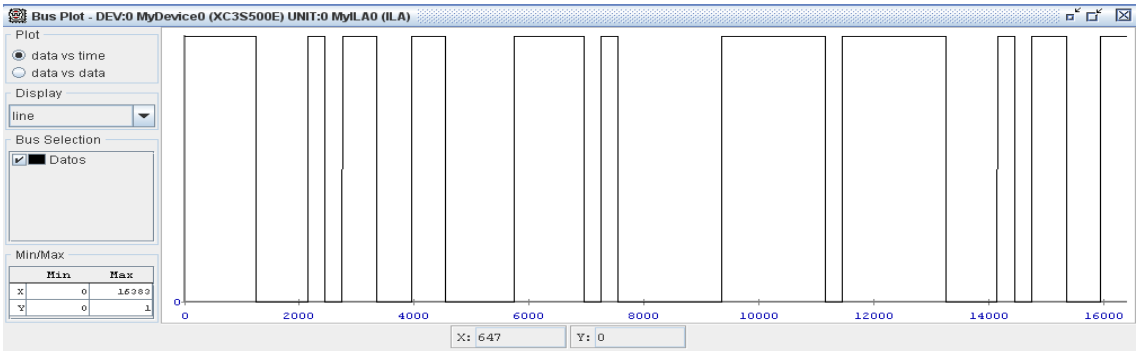


Figura 4.29 Datos demodulados BPSK

4.4.2 Demodulador QPSK

Se conoce que el proceso de modulación QPSK es muy similar al BPSK con algunas diferencias. Para la implantación del demodulador QPSK se sabe que el modulador QPSK está conformado por dos canales de datos. Un canal modula los datos en fase (I) y el otro canal modula los datos en cuadratura (Q). La Figura 4.30 muestra los componentes en System Generator del demodulador QPSK.

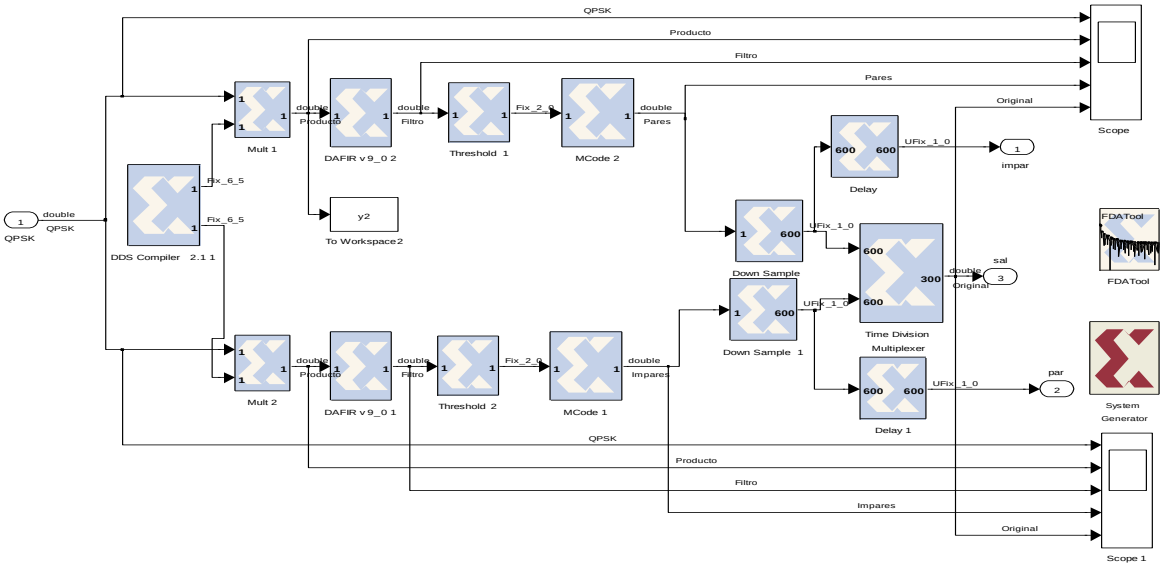


Figura 4.30 Componentes del demodulador QPSK en System Generator

La señal QPSK entra al receptor y se multiplica con una señal que está en fase (seno) y una en cuadratura(coseno) mediante el bloque *Mult,* estas señales son generadas por el bloque *DDS Compiler* que tiene la misma configuración de la Figura 4.21 pero con la opción *Sine_and_Cosine* y están a una frecuencia de 500 Khz.

Al igual que la modulación BPSK se utilizo un bloque *DAFIR* para la implementación de los filtros pasa-bajos y se los diseño con la herramienta FDAtool. En la Figura 4.31 se muestra el diseño del filtro en la herramienta FDAtool.

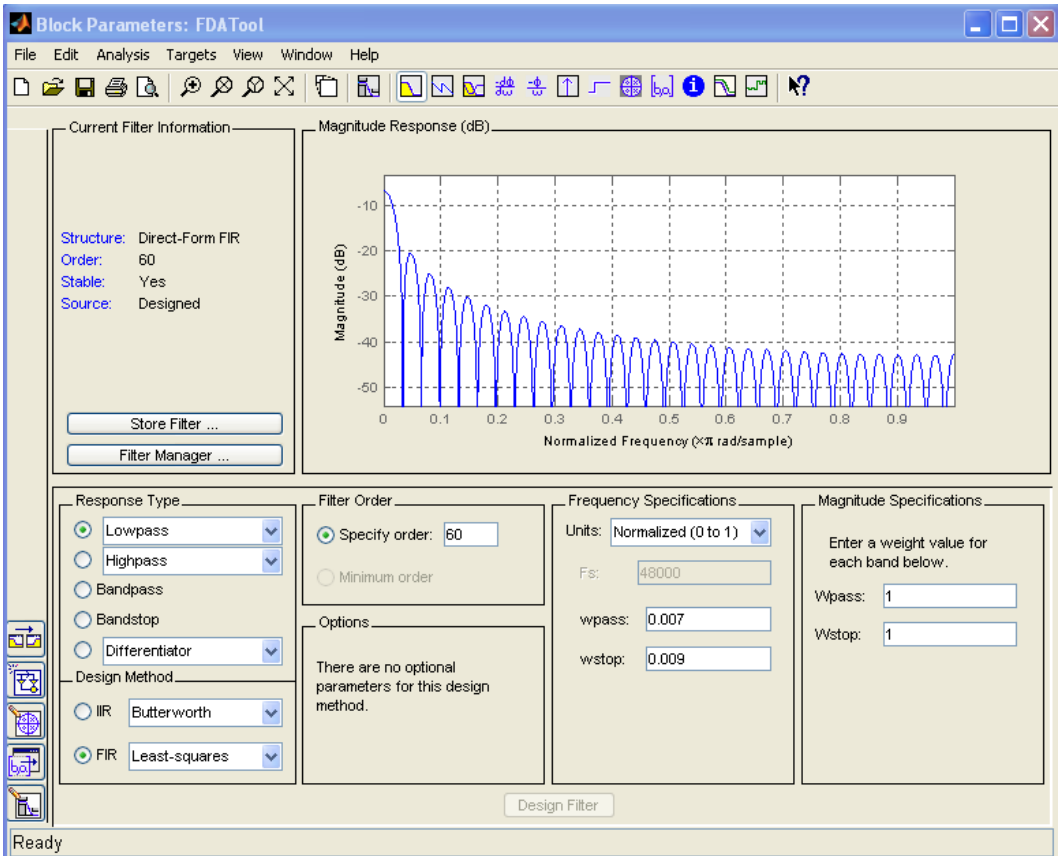


Figura 4.31 Configuración del Filtro pasa-bajos para QPSK

Los siguientes bloques *Threshold* , *Mcode* y *Down Sample* tienen la misma configuración que se utilizo en la demodulación BPSK.

Después de obtener los datos demodulados en los dos canales, para recuperar la señal que entro al modulador se utilizo el bloque *Time Division Multiplexer* que es un bloque que permite multiplexar varias señales que están en paralelo y presentarlas en una sola señal de datos seriales. La simulación del canal I se muestra en la Figura 4.32 y en la Figura 4.33 se muestra la simulación del canal Q.

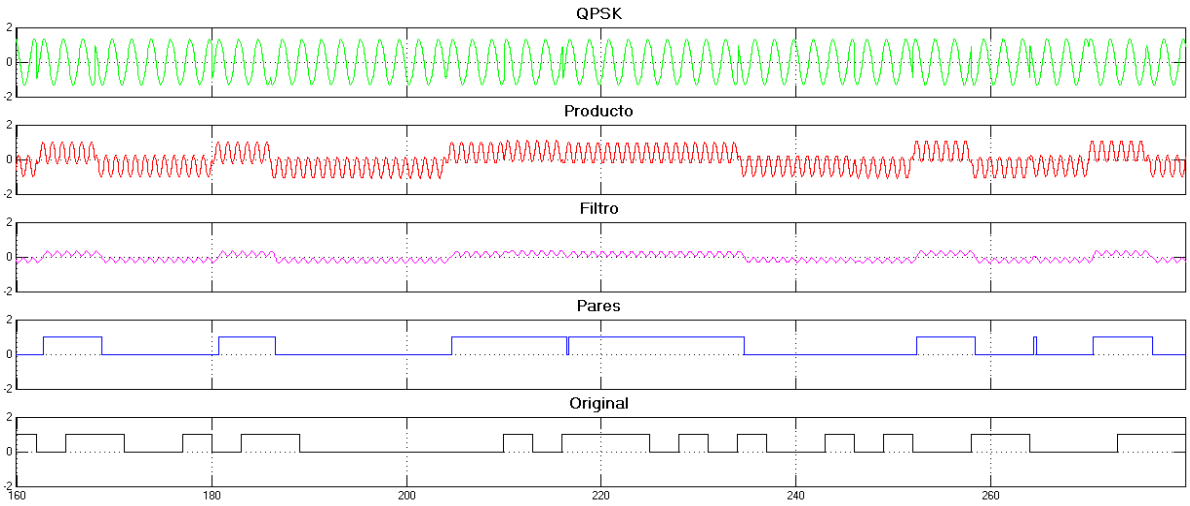


Figura 4.32 Simulación del canal I.

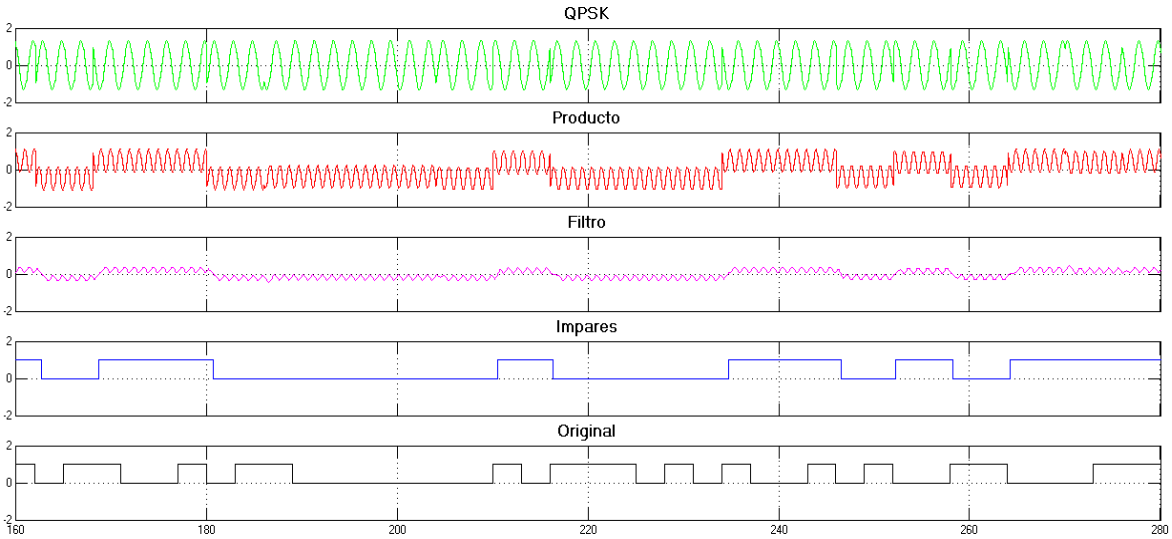


Figura 4.33 Simulación del canal Q.

Implementación del demodulador QPSK

Para la compilación del modelo se configura del bloque de System Generator como se muestra en la Figura 4.10. Para verificar el funcionamiento en tiempo real en el FPGA se utiliza el bloque *ChipScope* con la configuración de la Figura 4.12 y se genera una señal QPSK como se muestra en la Figura 4.34.

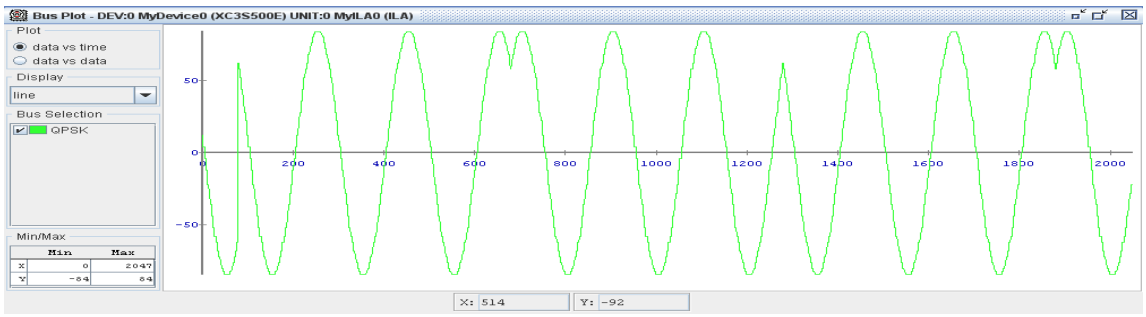


Figura 4.34 Señal modulada QPSK en Tiempo Real

El producto de la señal QPSK con la portadora del canal I se muestra en la Figura 4.35. Esta señal es un bus de datos de 10 bits en complemento a dos.

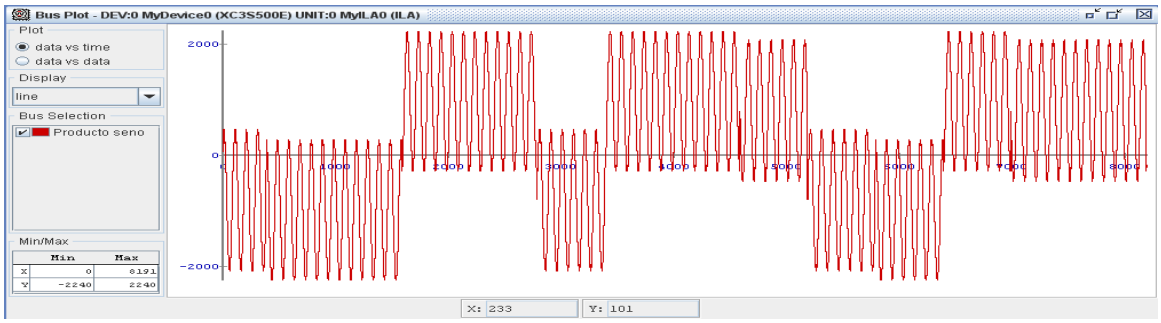


Figura 4.35 Producto de la señal QPSK del canal I en tiempo real.

La Figura 4.36 muestra la señal procesada después del filtro- pasa bajo. Esta señal es un bus de datos de 32 bits en complemento a dos.

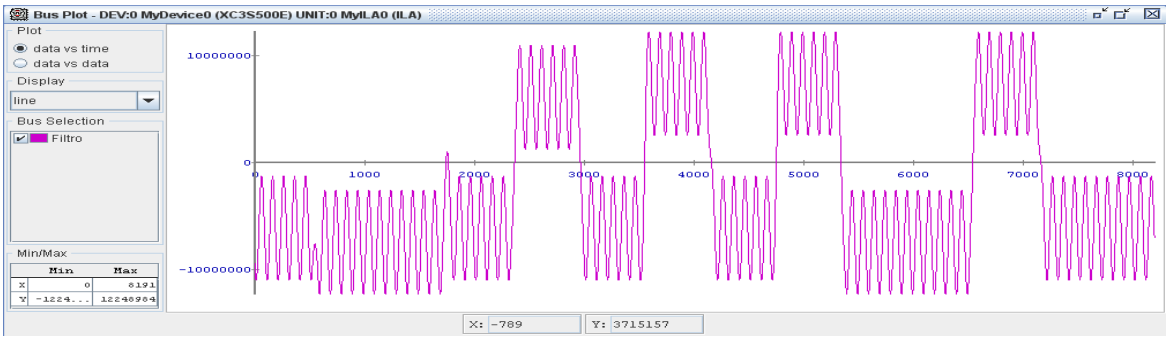


Figura 4.36 Respuesta del Filtro pasa-bajos en el canal I

La señal original y los datos pares e impares se muestran en la Figura 4.37.

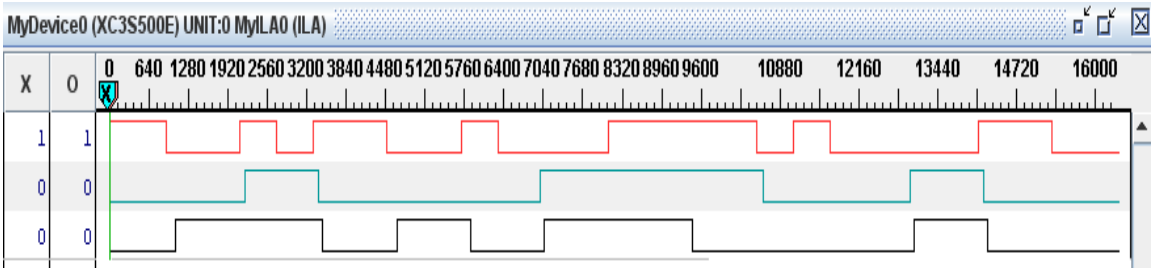


Figura 4.37 Señal Original, datos pares e impares

4.5 Decodificador Viterbi

Ya recuperada la señal digital se conoce que en el transmisor se codificó con códigos convolucionales con una profundidad de código de 7 y con polinomios generadores 171 y 133 en formato octal que generan una tasa de codificación de 1/2. En la Figura 4.38 se muestra los bloques que conforman el decodificador.

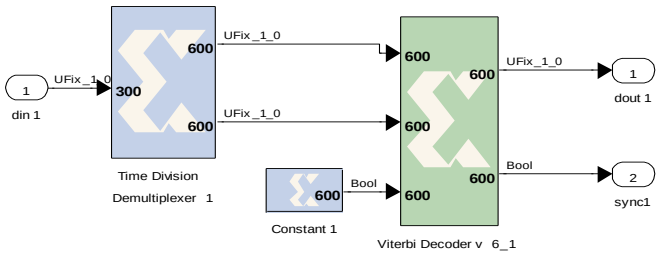


Figura 4.38 Componentes del Decodificador en System Generator

Los parámetros de configuración del bloque *Viterbi Decoder* que se muestran en la Figura 4.39 se establecieron según las características del codificador convolucional antes mencionadas. Esta configuración establece dos entradas en bloque por este motivo es necesario utilizar el bloque *Time División Demultiplexer* que permite dividir la señal de entrada en dos para que puedan ser procesadas por cada canal del *Viterbi decoder.* El bloque *Constant* es un booleano de valor 1 que sirve para habilitar el bloque *Viterbi decoder.*

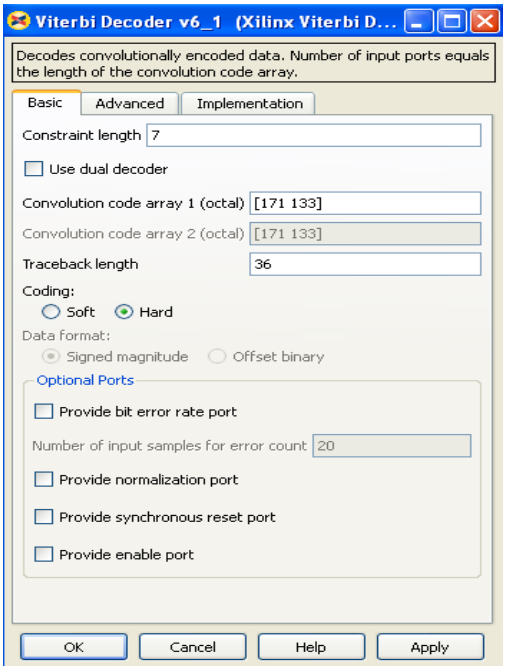


Figura 4.39 Ventana de Configuración del bloque Viterbi decoder

Para la decodificación se utilizó *Hard-Decision* porque después del proceso de demodulación la señal está compuesta por un solo bit que hace más sencillo el proceso de estimación del dato. Además *Hard-Decision* utiliza menos rescursos del FPGA.

El valor de profundidad de rastreo (*Traceback length*) es 36 porque es un valor que permite un menor retraso en la decodificación, sin modificar el correcto funcionamiento del decodificador. La simulación de la Figura 4.40 muestra el desempeño del decodificador Viterbi en el receptor.

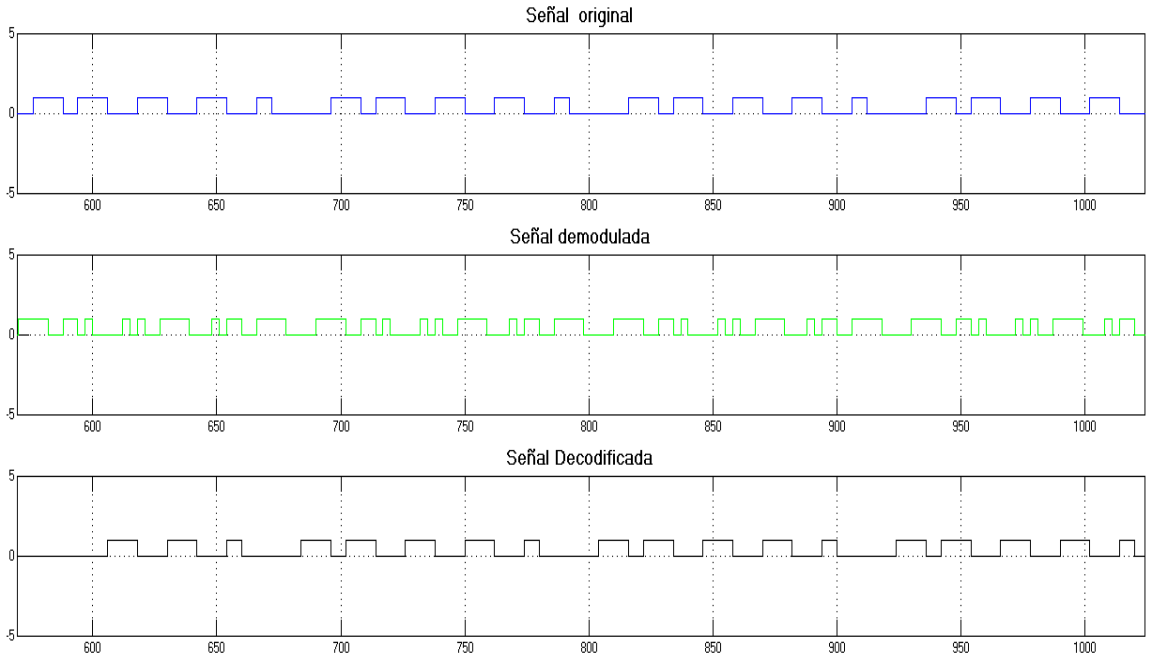


Figura 4.40 Señal original, señal demodulada, señal decodificada del sistema.

Implementación del decodificador Viterbi

Al igual que las otras etapas del receptor para verificar el funcionamiento de este elemento se configura del bloque de System Generator como se muestra en la Figura 4.10 con la diferencia de que en este punto de la implementación ya participan todos los componentes antes mencionados en el desempeño del decodificador implementado en el FPGA.

Para comprobar esta sección del modelo se utiliza el bloque *ChipScope* con la configuración de la Figura 4.12. Al igual que en la simulación en la Figura 4.41 se muestra el desempeño del Decodificador Viterbi en tiempo real con todos los componentes del receptor utilizando modulación BPSK.

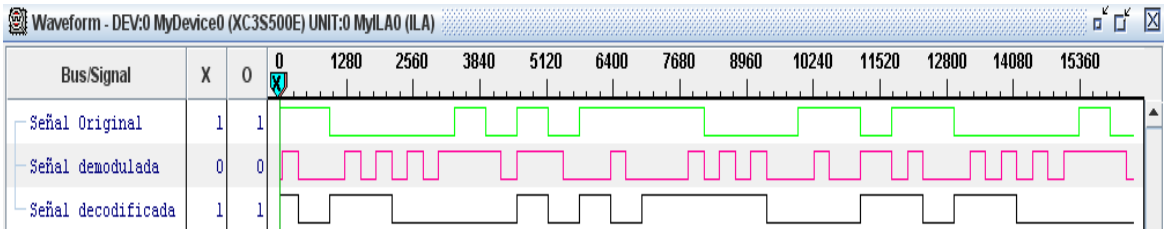


Figura 4.41 Señal original, señal demodulada, señal decodificada del sistema en tiempo real.

CAPÍTULO V

5. ANÁLISIS DE RESULTADOS

En este capítulo se analiza la diferencia entre las señales simuladas y las señales implementadas en el FPGA mediante *Hardware co-simulation* que se describe más adelante. Además se analizara las propiedades de las señales más importantes del recetor.

5.1 Compilación Hardware Co-Simulation

System Generator permite compilar un diseño que puede ser simulado en tiempo real en la interface gráfica de simulink. Para poder disponer de esta herramienta en el bloque de System Generator se configura la compilación como se muestra en la Figura 5.1.

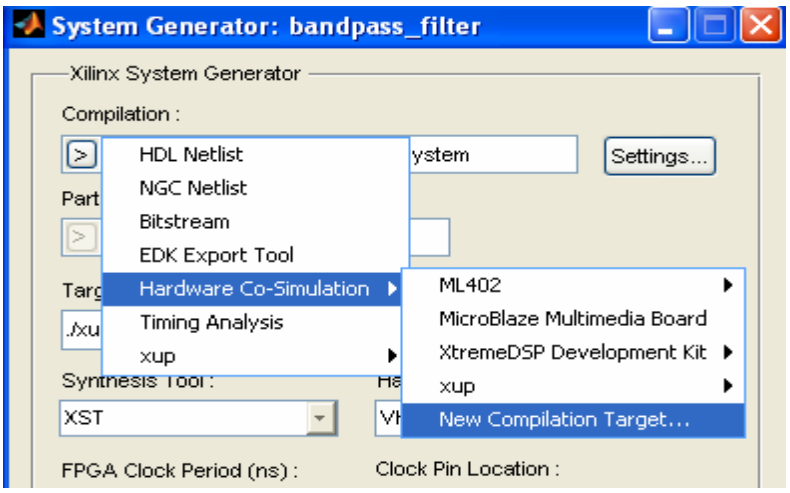


Figura 5.1 Opciones de Configuración para Hardware Co-Simulation

Si la tarjeta no está disponible se selecciona la opción *New compilation target* y se despliega una ventana como se muestra en la Figura 5.2 que permite configurar las características especificas de las diferentes tarjetas de desarrollo.

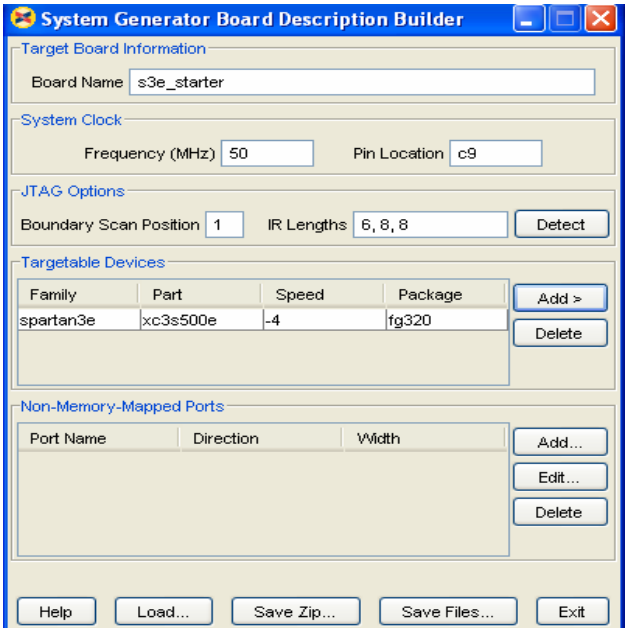


Figura 5.2 Configuración Hardware Co-Simulation de la tarjeta Spartan 3e

Para generar el bloque *Hardware Co-simulation* de un modelo cualquiera en el bloque de System Generator, en la opción de compilación se selecciona el nombre de la tarjeta que se configuro anteriormente y se oprime el botón generate. Al terminara el proceso de compilación se genera un bloque como el que se muestra en la Figura 5.3 que está conformado por todos los elementos del modelo como original.

Figura 5.3 Bloque para Hardware Co-Simulation

Para simular este bloque se debe tener conectada la tarjeta de desarrollo mediante el cable de programación ya sea paralelo o usb, se da *click* en el botón de inicio de la simulación en simulink y automáticamente se graba el diseño en el FPGA y se simula su comportamiento como cualquiera de los otros bloques de simulink, pero en tiempo real.

5.2 Señales simuladas vs Señales reales.

Es importante analizar las diferencias entre la simulación en System Generator y el diseño implementado en el FPGA para establecer parámetros que permitan que el modelo tenga un mejor desempeño. Para esto se compara la simulación normal de System Generator con la simulación mediante Hardware Co-Simulation.

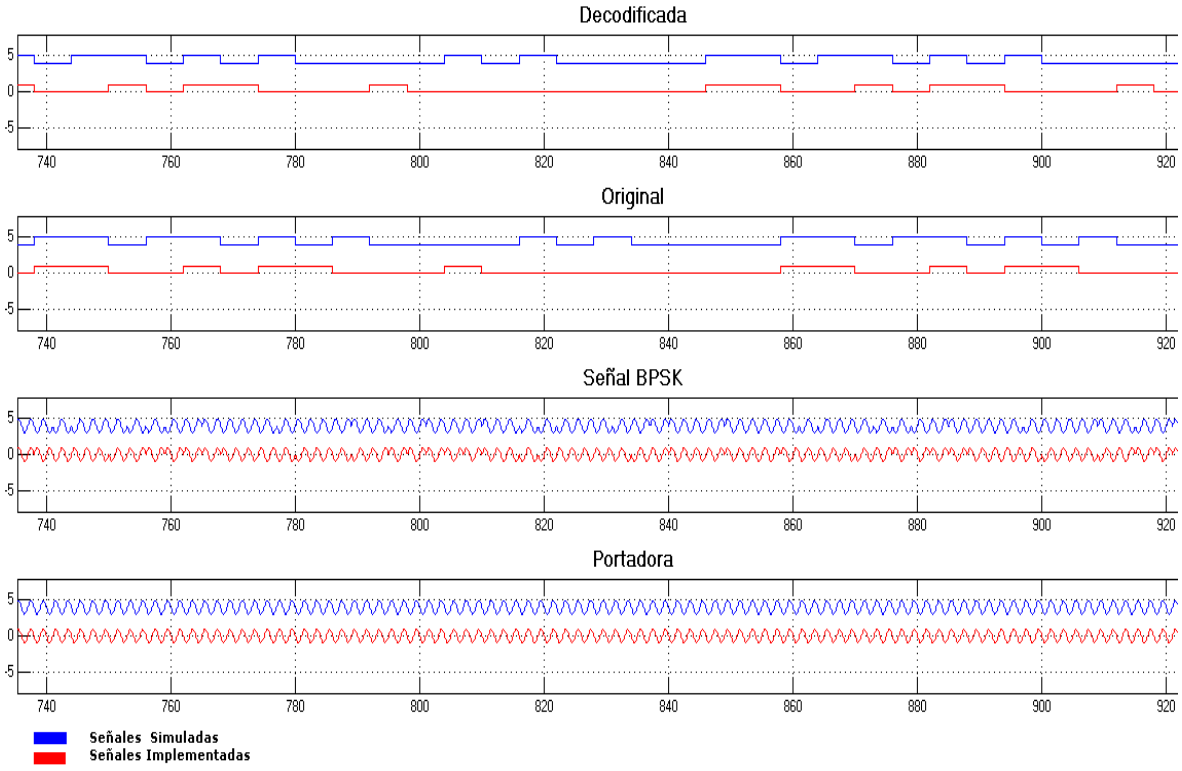


Figura 5.4 Señales simuladas y señales implementadas en el FPGA

En la Figura 5.4 se puede apreciar que la mayoría de señales tanto las obtenidas de la tarjeta como las simuladas son bastante similares lo que nos da como referencia que la simulación tiene un gran acercamiento a la realidad esto es debido a que System Generator considera los retardos de hardware para la simulación.

5.3 Características de las señales implementadas en el FPGA

Para analizar cómo se está comportando el sistema de comunicaciones en el FPGA se utilizo el diagrama de constelación, el diagrama de trayectoria y el diagrama ocular que pertenecen al block set de simulink.

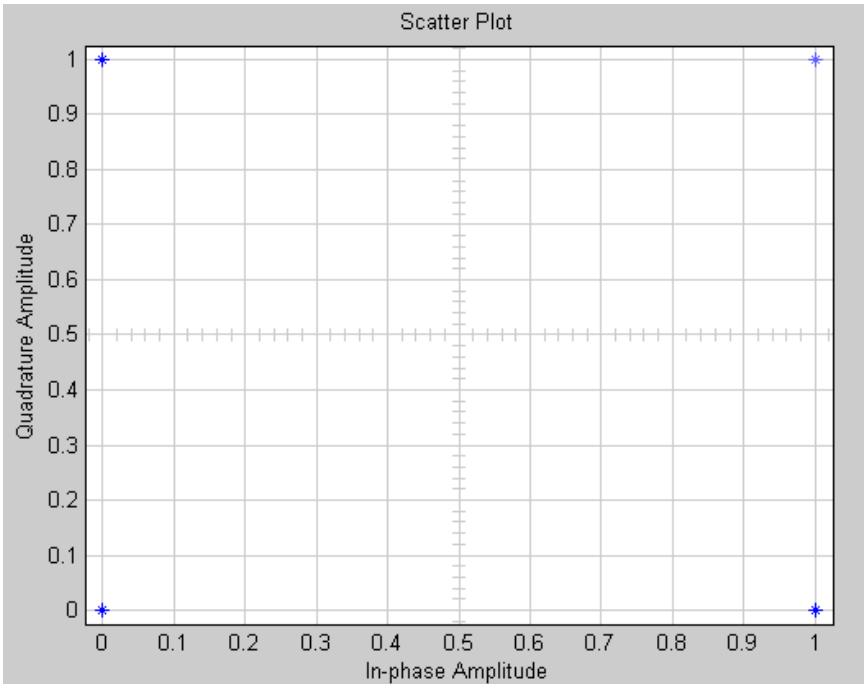


Figura 5.5 Diagrama de constelación.

En la Figura 5.5 se observan las constelaciones de la demodulación QPSK después del filtro receptor. Se puede apreciar que las constelaciones no presentan ninguna variación esto se debe a la no presencia de ruido en el

sistema lo que facilita la decisión de los símbolos, además no se observa un efecto de interferencia inter-simbólica.

La gráfica de la Figura 5.6 es el diagrama de trayectoria de la demodulación QPSK, aquí se observa que la señal cambia de un valor a otro ubicado en su diagonal, la señal tiene que cruzar por 0.5, esto se debe a que no se utilizo un código de cruce por cero en la modulación, es necesario tener en cuenta este factor para no perder sincronismo en el receptor.

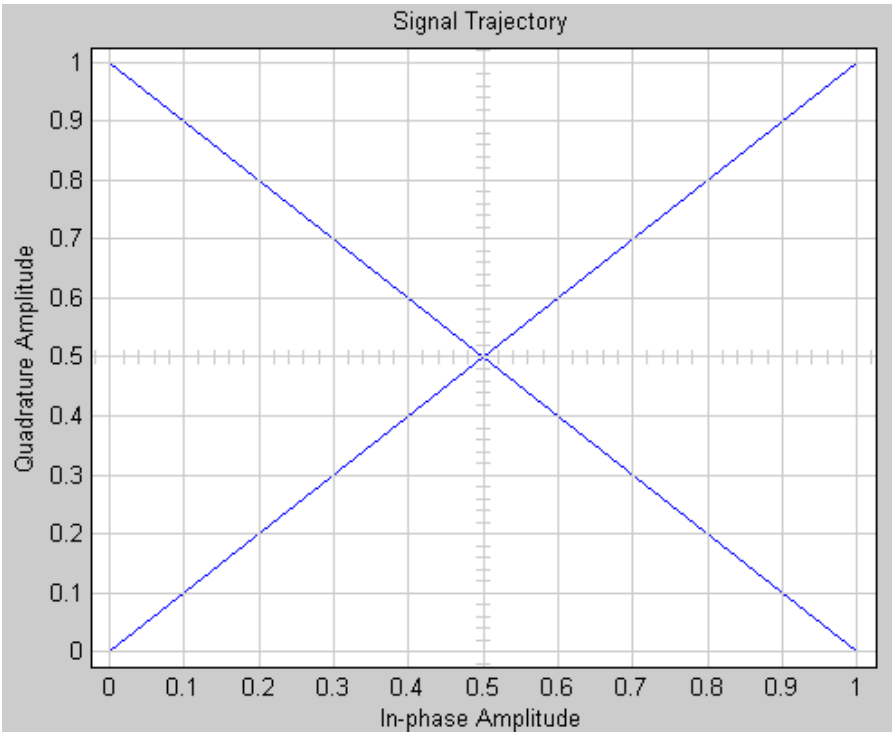


Figura 5.6 Diagrama de Trayectoria.

El diagrama del ojo de la Figura 5.7 muestra algunos parámetros que determinan la calidad de la señal, debido a que no se utiliza un código de cruce por cero no se puede apreciar la forma del ojo, pero si se denota una gran amplitud entre los valores de cero y uno lo que demuestra que la probabilidad de error es casi nula en este sistema. Además las líneas están bien definidas lo que demuestra casi no hay distorsión en la señal.

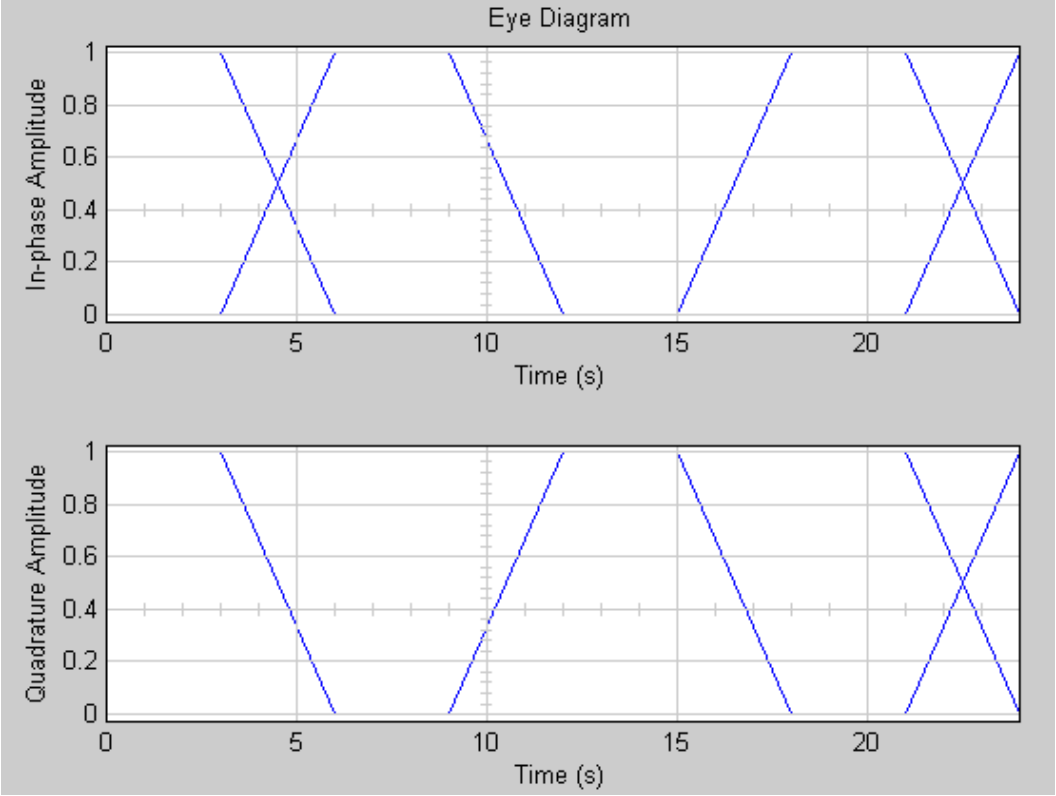


Figura 5.7 Diagrama del OJO.ma del OJO.

CONCLUSIONES Y RECOMENDACIONES

Después de analizar la información de la etapa de recepción de un sistema de comunicaciones, conocer el funcionamiento de la tecnología FPGA y finalizadas las simulaciones y pruebas en la tarjeta Spartan 3-E es posible alcanzar una serie de conclusiones relacionadas con el desarrollo de este proyecto.

La Tecnología FPGA es un conjunto de dispositivos semiconductores que contienen componentes lógicos programables e interconexiones programables entre ellos que permiten implementar sistemas electrónicos digitales en un tiempo reducido sin la necesidad de verificación de sus componentes, tarea que ya es realizada por el fabricante.

La necesidad de desarrollar un diseño en un FPGA, de forma rápida, eficiente, fiable con un bajo coste asociado ha obligado a fabricantes de herramientas EDA a desarrollar aplicaciones como System Generator, que tiene una interfaz gráfica que facilita y agiliza el labor del diseñador además mediante el uso de bloques permite una mejor asimilación del modelo que se está realizando.

Para tener control sobre los periféricos (ADC) o el reloj en hardware de una tarjeta de desarrollo, System Generator permite al diseñador importar código HDL(VHDL o verilog) o crear un bloque mediante código matlab pero con ciertas restricciones.

ChipScope Pro es una herramienta de la compañía Xilinx que permite comprobar en tiempo real el funcionamiento de las señales de un diseño facilitando la depuración y el análisis de los elementos más críticos de un modelo.

Para poder optimizar los recursos del FPGA en la demodulación BPSK la frecuencia de la señal portadora debe ser casi igual a la mitad de la frecuencia de de los símbolos obtenidos después de la codificación convolucional debido a que el filtro digital paso bajo en el receptor utiliza un menor orden y por ende un menor número de bits para diferenciar los niveles positivos o negativos del producto de la señal BPSK por la portadora.

System generator es una herramienta que da muchas facilidades al momento de diseñar debido a bloques de simulink que permiten analizar las diferentes señales, pero por estar empleando un lenguaje de más alto nivel (gráfico) se pierde cierto control sobre la estructura hardware, porque al usar bloques generales es más difícil optimizar en cuanto a área de silicio ocupada, tiempos y rendimientos

La co-simulación hardware, es una de las cualidades más importantes de System Generator por que permite simular en tiempo real un diseño implementado en el FPGA y comparar a la vez con la simulación del diseño sin implementar.

Es necesario conocer algunos aspectos de lenguaje HDL, por que permiten dar una visión del código HDL generado después de la compilación del modelo en System Generator y así poder corregir algunas líneas de código que no se pueden configurar directamente en cada bloque.

Para poder manipular dispositivos periféricos al FPGA como el conversor análogo digital es necesario conocer con detalle su funcionamiento y como esta interactuando con el FPGA.

Para sincronizar mejor las etapas del modelo los periodos de tiempo de los bloque deben ser iguales en lo posible y cuando sea necesario utilizar bloques de retardo.

System Generator es una herramienta ideal para el aprendizaje de las diferentes teorías por que permite la fácil implementación de modelos electrónicos en hardware permitiendo al estudiante la posibilidad de entender cómo se desempeñan los diferentes métodos en la práctica.

Para la optimización del recetor se debería diseñar un circuito de recuperación de portadora en System Generator para hacer una demodulación coherente sin la necesidad de conocer la fase de la portadora en el transmisor.

REFERENCIAS BIBLIOGRÁFICAS

[1] Joaquín Luque Rodríguez, Sebastián Clavijo Suero, *Modulación De Señales Digitales*, UNIVERSIDAD DE SEVILLA, DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA.

[2] Oriol Sallent Roig, José Luis Valenzuela González, Ramón Agustí Comes, *Principios de comunicaciones móviles,* Primera Edición, Edición de la Universidad Politécnica de Catalunya, Septiembre 2003

[3] Simon Haykin, *Sistemas de Comunicación*, Primera edición, Editorial *LIMUSA WILEY*, 2002

[4] Basil M. Al-Hadithi, Juan Suardíaz Muro, "*Nuevas Tendencias En El Diseño Electrónico Digital: Codiseño Hardware/Software*", Universidad Alfonso X El Sabio.

[5] Octavio H. Alpago, Miguel A. Sagreras, "*Lógica Programable: FPGA",* Universidad de Buenos Aires, Facultad de Ingeniería

[6] Emmanuel López Trejo, "*Implementación Eficiente en FPGA del Modo CCM usando AES"*, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional Unidad Zacatenco

[7] Ian Kuon, Russell Tessier, Jonathan Rose, "*FPGA Architecture: Survey and Challenges"*, Vol 2, 2008

[8] Gerardo Castro Jiménez, *"Procedimiento de diseño e implementación de circuitos digitales utilizando herramientas EDA de código abierto"*, Universidad de Costa Rica.

[9] Ian Grout, *"Digital Systems Design with FPGAs and CPLDs*", Elsevier Ltd. 2008

[10]http://bibing.us.es/proyectos/abreproy/11155/fichero/Memoria%252F1_Introducci%F3n.pdf

[11] Manuel Gutiérrez Rizo*,"Evaluación de herramientas de alto nivel para diseño hardware*", Departamento de Ingeniería Electrónica, UNIVERSIDAD DE SEVILLA, 2007

[12] Digilent Inc., Digital Design Engineer's Source < http://www.digilentinc.com/ >

[13] Xilinx, "Spartan-3E Starter Kit Board User Guide", Xilinx Ltd, p.75-81,2006.

[14] Dennis Silage," *Embedded Design Using Programmable Gate Arrays*",Electrical and computer Engineering, Temple University,2008

Sangolquí, 7 de Diciembre de 2010

ELABORADO POR:

_____

Gabriel Darío Fierro Piñeiros

C.I. 171674506-0

_____

Ing. Gonzalo Olmedo Ph.D

DIRECTOR DE LA CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES