ESCUELA POLITÉCNICA DEL EJÉRCITO

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

PROYECTO DE GRADO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERÍA

IMPLEMENTACIÓN DE LA ETAPA DE TRANSMISIÓN DE UN SISTEMA DE COMUNICACIÓN DIGITAL UTILIZANDO LA TECNOLOGÍA FPGA

JOHANNA LIZETH FERNÁNDEZ YÉPEZ

SANGOLQUÍ - ECUADOR

2010

Resumen

En el presente proyecto de grado se implementa la etapa de transmisión de un sistema de comunicación digital utilizando tecnología FPGA y se analiza el desempeño de los bloques de codificación de canal y modulación digital en hardware. La modulación digital empleada es BPSK y QPSK.

Inicialmente se indaga en la arquitectura, las formas de implementación y las herramientas para la programación de FPGAs. Así como también la configuración y características de cada etapa de un sistema de transmisión digital.

Mediante el software *System Generator*, se implementa el diseño tanto de un transmisor con modulación BPSK como de un transmisor con modulación QPSK. En estos diseños se utiliza código convolucional para la codificación del canal. Para la creación de la señal modulada, se generan portadoras sinusoidales dentro de un conjunto base de señales ortonormales utilizando los osciladores internos del FPGA, definiendo el período de muestreo. En este software el desempeño de los dos sistemas es evaluado mediante la simulación. Para poder observar el desempeño en hardware, utilizando VHDL se configura el DAC el cual puede comunicarse con el diseño en *System Generator* con la ayuda de un bloque muy importante llamado *black box*.

Finalmente, los resultados obtenidos son visualizados en el osciloscopio y posteriormente analizados, además se analiza el consumo de recursos del FPGA.

DEDICATORIA

A mi madre por estar conmigo en cada paso que doy, por ser mi fiel amiga, por confiar en mí y por demostrarme que no hay barreras para alcanzar cualquier sueño.
A mi padre por todo su apoyo y por haberme enseñado que la honestidad siempre debe estar presente en cada decisión de la vida.
A mis abuelos Carlitos y Zoilita, por expresar su amor en cada gesto, por haber entregado y seguir entregando más de lo que pueden y de lo que deben. Para ustedes y por ustedes.

Johanna Fernández Y.

AGRADECIMIENTO

Agradezco a Dios por haberme dado la vida y por haber puesto en mi camino personas maravillosas que me llenan de felicidad. Gracias Diosito por estar a mi lado todo el tiempo.

Gracias a ti madre, por todo lo que has hecho por mí desde el día en que nací. Gracias por llorar y reír conmigo, por dejar de dormir para darme un consejo y sobre todo por ser la persona con la que puedo contar siempre para lo que sea. Te amo.

Gracias papi por ayudarme en todo lo que has podido, gracias por tu apoyo y tu confianza en estos 23 años.

Gracias Jorge y Cristian, por haber hecho de mi niñez algo hermoso, por brindarme su compañía, por ser mis hermanos y amigos, por aguantar mi mal genio y por seguir llenando mi vida de momentos felices.

Gracias abuelitos, son las personas que más admiro, son ejemplos dignos de seguir. Gracias por su amor, por su entrega incondicional, por haberme dado la oportunidad de tener recuerdos hermosos y sobre todo por demostrarme que soy importante para ustedes. Espero poder retribuir todo lo que han hecho por mí.

Gracias a ti Jorge Luis, por estar conmigo en las buenas y en las malas, gracias por haber hecho de uno de los momentos más difíciles de mi vida, un momento llevadero. Gracias por todo el apoyo que me das día a día, gracias por haberme enseñado la importancia de la humildad y gracias por haberme dado la oportunidad de compartir contigo y con tu hermosa familia.

Gracias a mis tíos, tías, primos y primas, por su amor, sus palabras de aliento y por todo su apoyo.

Gracias a mis amigos, gracias por estar conmigo cuando más los he necesitado, gracias por todos esos momentos llenos de alegría.

Gracias a los ingenieros Gonzalo Olmedo y Byron Navas, por haberme guiado a lo largo de todo el proyecto, gracias por su apoyo y por haber confiado en mí.

Gracias a todos los buenos maestros que tuve a lo largo de mi carrera, por haber compartido todo su conocimiento con nosotros y sobre todo por su amistad.

Esto es simplemente el inicio de una vida llena de oportunidades, espero no defraudarlos y seguir adelante cumpliendo nuevas metas con la bendición de Dios.

PRÓLOGO

A principio del año 2000, el desempeño de FPGA da un gran paso ya que presentan muchas ventajas como millares de compuertas lógicas, pines de alta velocidad, etc. Esto hace que hoy en día se utilice en básicamente todos los sistemas de comunicaciones, incluyendo procesamiento digital de señales. La principal ventaja es que son reprogramables, de esta manera se ofrece una gran flexibilidad de diseño y una disminución significativa de los costos de implementación.

El software *System Generator* ofrece grandes ventajas. Un ejemplo muy claro de esto se puede encontrar en la empresa Nallatech, proveedor de sistemas FPGA, quien usó Simulink y Xilinx *System Generator* para diseñar un sistema reconfigurable de codificación de video en tan solo dos semanas. Este sistema permite que todos sus clientes puedan verificar todo el sistema cuando ha habido cambios en sus componentes o interfaces, sin conocer nada acerca de VHDL. La principal ventaja de haber utilizado este software es que el tiempo de trabajo fue considerablemente optimizado.

El estudio realizado en el presente proyecto demuestra que mediante herramientas como Xilinx *System Generator*, se puede trabajar fácilmente con sistemas de comunicación en un entorno gráfico y amigable optimizando el tiempo de desarrollo. Los sistemas de comunicación implementados aprovechan las bondades del software. La más importante es la interactividad entre Simulink e ISE, ya que en el diseño realizado en *System Generator* con bloques de la librería de Xilinx en Simulink, existen también bloques generados mediante programación en VHDL y esta programación fue realizada en ISE.

En el Ecuador existe una falta de desarrollo tecnológico para sistemas de Telecomunicaciones, debido al alto costo de inversión para el desarrollo de circuitos integrados para estos fines, por tal razón todas las soluciones tecnológicas son cerradas y a pesar de todos los conocimientos en comunicación digital, resulta difícil llegar a su implementación en hardware. Es por esto que este estudio busca proponer nuevas soluciones en hardware e incrementar la competencia de los profesionales de la Escuela Politécnica del Ejército en esta área de conocimiento.

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

GLOSARIO

ANSI: (*American National Standards Institute).* Instituto Nacional Estadounidense de Estándares.

ASIC: (*Application-specific Integrated Circuit).* Circuitos Integrados para Aplicaciones Específicas.

ASK: *(Amplitude Shift Keying).* Modulación por Desplazamiento de Amplitud.

ASSP: *(Application Specific Standard Product).* Producto de Estándar Específico de Aplicación.

BPSK: *(Binary Phase Shift Keying***).** Modulación por Desplazamiento de Fase Binaria.

CAD: *(Computer Aided Design).* Diseño Asistido por computadora u Ordenador.

CAE: (*Computer Aided Engineering*). Ingeniería Asistida por computadora u Ordenador.

CDMA: *(Code Division Multiple Access).* Acceso Múltiple por División de Código.

CLB: *(Configurable Logic Blocks).* Bloque Lógico Programable.

CMOS: *(Complementary Metal Oxide Semiconductor).* Estructuras Semiconductor-Óxido-Metal Complementarias.

BlockSet: Conjunto de bloques.

CPLD: *Complex Programmable Logic Device).* Dispositivo Lógico Programable Complejo.

DAC: *(Digital Analog Converter).* Conversor Digital Análogo.

DDS Compiler: Compilador de un Sintetizador Digital Directo.

DRAM: *(Dynamic Random Access Memory).* Memoria de Acceso Aleatorio Dinámico.

DSP: *(Digital Signal Processor)* Procesamiento Digital de Señales.

DVB-S: *(Digital Video Broadcasting by Satellite).* Radiodifusión de Vídeo Digital – Satélite.

EDA: *(Electronic Design Automation).* Automatización de Diseño Electrónico.

EEPROM: *(Electrically-Erasable Programmable Read-Only Memory).* Memoria de solo Lectura Programable y Borrable Eléctricamente.

EPROM: *(Erasable Programmable Read-Only Memory).* Memoria de solo Lectura Programable y Borrable.

FEC: *(Forward Error Correction).* 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: (*Field Programmable Gate Array*). Matriz de puertas programables.

FSK: *(Frecuancy Shift Keying***)**. Modulación por Desplazamiento de Frecuencia.

HDL: *(Hardware Description Languaje).* Lenguaje de Descripción de Hardware.

IDE: *(Integrated device Electronics).* Ambiente de diseño Integrado.

XSG: Xilinx *System Generator*.

IEEE: *(Institute of Electrical and Electronics Engineers*). Instituto de Ingenieros Eléctricos y Electrónicos.

ISE: (*Integrated Software Environment).* Ambiente de Software Integrado.

ISO: *(International Standart Organization).* Organización Internacional para la Estandarización.

ISO 14443: Es un estándar internacional relacionado con las tarjetas de identificación electrónicas, en especial las tarjetas inteligentes.

LSFR: *(Linear Feedback Shift Register).* Registro de Desplazamiento con Retroalimentación Lineal.

LUT: *(Look-Up Table).* Tabla de Consulta.

PLD: *(Programmable Logic Device).* Dispositivo Lógico Programable.

PROM: *(Programmable Read Only Memory).* Memoria de sólo Lectura Programable.

PSK: *(Phase Shift Keying***).** Modulación por desplazamiento de Fase.

QPSK: *(Quadrature Phase Shift Keying***).** Modulación por Desplazamiento de Fase en Cuadratura.

RAM*: (Random Acces Memory).* Memoria de Acceso Aleatorio.

RFID: *(Radio Frequency IDentification).* Identificación por Radiofrecuencia.

ROM: *(Read Only Memory).* Memoria de solo Lectura.

RTL: *(Register Transfer Level).* Nivel de la Transferencia de Registro.

SASIC: *(Specific Application Structured Integrated Circuit).* Circuitos Integrados Estructurados para Aplicaciones Específicas.

SRAM: *(Static Random Acces Memory).* Memoria Estática de acceso Aleatorio.

VHDL: *(Very High Speed Integrated Circuit Hardware Description Languaje).* Lenguaje textual de alto nivel que se utiliza para la descripción del hardware de los sistemas digitales. Es igual a VHSIC + HDL.

VHSIC: *(Very High Structured Integrated Circuit).* Circuitos Integrados de Muy Alta Velocidad.

CAPÍTULO 1

TECNOLOGÍAS PARA LA IMPLEMENTACIÓN DE SISTEMAS EN FPGA'S

1.1 Introducción a las FPGAs

Los dispositivos FPGAs (*Field Programmable Gate Array*)**,** son un conjunto de circuitos integrados digitales que contienen bloques lógicos reconfigurables con interconexiones entre ellos. Su nombre referencial puede ser "campo programable" porque su programación se da "en campo", a diferencia de los dispositivos en los cuales las funciones internas están fuertemente "amarradas" por el fabricante [1]. Esto quiere decir que los FPGAs pueden ser configurados tanto en el laboratorio como también posteriormente en sus ubicaciones definitivas de operación.

Un FPGA es un dispositivo multinivel programable de propósito general. Integra una gran cantidad de dispositivos lógicos programables en un chip. El tamaño y velocidad de los FPGAs es equiparable a los Circuitos Integrados para Aplicaciones Específicas (ASICs), pero los FPGAs son más flexibles, su ciclo de diseño es más corto y los recursos necesarios generan menores gastos. La adopción de chips FPGA en las industrias ha sido impulsada por el hecho de que los FPGAs combinan lo mejor de los ASICs y de los sistemas basados en procesadores. A diferencia de los procesadores, los FPGAs llevan a cabo diferentes operaciones de manera paralela, por lo que éstas no necesitan competir por los mismos recursos.

En el nivel más alto, los FPGAs son chips de silicón reprogramables, los FPGAs son completamente reconfigurables y al instante toman una nueva función cuando se compila una diferente configuración de circuitos. Anteriormente, sólo los ingenieros con un profundo entendimiento de diseño de hardware digital podían trabajar con la tecnología

FPGA. Debido al aumento de herramientas de diseño de alto nivel se están cambiando las reglas de programación de los FPGAs, con nuevas tecnologías que convierten los diagramas a bloques gráficos, o hasta el código ANSI C, a circuitos de hardware digital.

Los FPGAs han creado un nuevo mercado, conocido como computación reconfigurable. Se espera que el mercado de FPGAs en todo el mundo aumente de $1.900 millones en el 2005, a $2.750 millones en el 2010. La adopción de la tecnología FPGA continúa creciendo, mientras que las herramientas de alto nivel evolucionan para brindar a los ingenieros y científicos con todos los niveles de experiencia los beneficios del silicio reprogramable [2].

1.2 Conceptos fundamentales y origen de las tecnologías

A pesar de que existen muchos tipos de circuitos integrados, es interesante conocer el por qué del éxito de los FPGAs. A continuación se exponen las tecnologías de dispositivos lógicos programables (PLDs): ASICs (*Application Specific Integrated Circuit)*, ASSPs (*Aplication Specific Standard Part*) y FPGAs.

PLD, es un dispositivo cuyas características pueden ser almacenadas y modificadas mediante la programación y así poder desempeñar muchas funciones. La mayoría de PLDs están formados por una matriz de conexiones, una matriz de compuertas AND y una matriz de compuertas OR [3]. La mayor dificultad que se presenta al trabajar con estos dispositivos es el número relativamente bajo de compuertas lógicas disponibles comparado con las de los FPGAs, por esta razón las funciones que se pueden implementar son simples y pequeñas.

CPLD *(Complex Programmable Logic Device)*, es un PLD con un mayor nivel de integración ya que permite implementar sistemas más eficientes porque utilizan menos espacio, mejoran la confiabilidad en el circuito y reducen costos. Un CPLD está formado de múltiples bloques lógicos, cada uno similar a un PLD. Los bloques lógicos se comunican entre sí utilizando una matriz programable de interconexiones lo cual hace más eficiente el uso del Silicio y conduce a un mejor desempeño. La arquitectura de estos dispositivos produce circuitos con características lógicas predecibles, por lo que son utilizados en componentes críticos en los que el conocimiento de las características temporales es fundamental.

En el otro extremo se encuentran los ASSPs y los ASICs, estos pueden contener millares de compuertas lógicas para ser utilizadas en funciones muy complejas. Tanto los ASICs como los ASSPs son utilizados en aplicaciones específicas; la principal diferencia es que un ASIC es proyectado y construido para una única empresa, mientras que los ASSPs son destinados a muchos clientes. A pesar de que los ASICs ofrecen una una mejor solución en tamaño, capacidad de procesamiento yo de desempeño, poseen un alto costo de desarrollo, se demoran considerablemente en llegar al mercado *(time-to-market)*, de esta manera no tienen ninguna flexibilidad. Así, dentro de las soluciones de implementación en hardware, los FPGAs representan la mejor opción entre PLDs y ASICs, porque puede ser configurada en campo como los PLDs y al mismo tiempo contener millares de compuertas lógicas. El costo de desarrollo y el *time-to-market* son menores comparados con los ASICs.

Las memorias son componentes fundamentales en la constitución de un dispositivo electrónico y por esa razón es necesario conocerlas, al menos en un nivel básico. Las memorias ROM[6] (*read-only memory)*, que significa "memoria de sólo lectura", es una memoria de semiconductor destinada a ser leída y no destructible, es decir, que no se puede escribir sobre ella y que conserva intacta la información almacenada, incluso en el caso de que se interrumpa la corriente, son llamadas memorias no volátiles.

La generación siguiente de las memorias ROM, fueron las memorias PROM (Programmable Read Only Memory). Las memorias PROM fueron creadas basadas en una tecnología de fusión de conexión. Es importante destacar que estos dispositivos ser usados como memorias capaces de almacenar programas de computadora y valores constantes dados, pero con el transcurso del tiempo se notó su gran utilidad para implementar funciones lógicas simples como tablas de *lookup*.

En 1971, Intel crea la memoria EPROM [8] *(Erasable Programmable Read-Only Memory)* ya que existía la necesidad de poder reprogramar una memoria, ya que hasta ese entonces las memorias sólo podían ser programadas una única vez. El principal problema con el uso de este tipo de memoria es el costo de la ventana de cuarzo y del largo tiempo, aproximadamente 20 minutos para borrar los datos desde el dispositivo.

La siguiente en la evolución de las memorias fue EEPROM [9] *(Electrically-Erasable Programmable Read-Only Memory)*. La diferencia en estructura con la EPROM

es que tiene dos transistores en vez de uno, también hay un cambio en el dopaje de silicio, lo que hace posible la propiedad de eliminar los datos eléctricamente. Continuando con la evolución se presentó la memoria flash [10] la cual es una forma avanzada de EEPROM la cual también borra datos eléctricamente pero no parcialmente.

Posteriormente llegan las memorias RAM [11], estas memorias son volátiles ya que pierden toda la información una vez que se interrumpa la corriente. Son de dos tipos, una tecnología DRAM (Dynamic Random Access Memory), y otra tecnología SRAM (Static Random Access Memory).Las memorias DRAM son llamadas dinámicas porque el capacitor pierde su carga en cada ciclo de tiempo, entonces cada célula debe cargarse periódicamente si necesita mantener los datos. El costo de una memoria DRAM es mucho menor que el de una SRAM. Las memorias SRAM almacenan los datos mientras son alimentadas por energía. Estas operan a una frecuencia diez veces superior a las memorias DRAM.

Los primeros CIs programables disponibles en el mercado fueron los PLDs, que surgieron en los años 70 y se decía que eran simples memorias PROMs. Al final de los años 70 surgieron versiones más complejas de PLDs. Así las versiones más rudimentarias de PLD fueron denominadas SPLDs y las posteriores y más desarrolladas se denominaron PLDs complejos o CPLDs.

En los años 80 se introducen los ASICs entre los cuales existen cuatro tipos que, en orden de complejidad, son: arreglo de compuertas lógicas (Gate Array), ASIC estructurados, celdas estándares y ASIC completamente personalizados.

Los ASIC completamente personalizados, fueron desarrollados para una aplicación específica, en este tipo de ASIC, los ingenieros de desarrollo poseen un total control sobre cada capa de máscara utilizada para fabricar un chip de silicio. El uso de máscaras predefinidas para la fabricación no deja otra opción para la modificación del circuito durante la fabricación, excepto quizás por algún ajuste menor o calibración. Esto significa que un ASIC completamente personalizado no puede ser modificado para adaptarse a diferentes aplicaciones, y se produce generalmente como un producto único, específico para una aplicación en particular. El desenvolvimiento de este tipo de ASIC es muy complejo y requieren un periodo de tiempo bastante largo.

La concepción de arreglo de puertas lógicas *(Gate Arrays)* surgieron en componías como IBM en los años 60, pero solo para uso interno. Su surgimiento al mercado fue muy posterior. El concepto de arreglo de puertas lógicas es basado en la idea de una célula básica constituida a partir de una colección de transistores y resistores desconectados. Cada fabricante de ASIC determina cual es la combinación ideal de los componentes para formar una celda básica. En general estas celdas básicas son representadas por arreglos de columnas, simples o dobles, y un área libre entre ellas llamada canal. La Figura 1.1 representa la arquitectura de arreglo de puertas lógicas con canales y arreglos de celdas básicas en columnas dobles. El fabricante ASIC define un grupo de funciones lógicas (multiplexores, registradores, etc.) que pueden ser usados por el desarrollador. Los arreglos de puertas lógicas ofrecen ventajas en cuanto al costo, una vez que el transistor y otros componentes son pre-fabricados. Las desventajas son que muchos recursos internos pueden no ser utilizados. La posición de las puertas de entrada y de salida son fijas y el enrutamiento interno no se optimiza.

En los años 80 se crean los circuitos integrados de celdas estándares, que tienen muchas semejanzas con el arreglo de puertas lógicas. En este tipo de tecnología también el fabricante define la biblioteca de la celda que puede ser usada por el desarrollador. El fabricante facilita las bibliotecas de elementos como procesadores, funciones de comunicación y funciones de memoria RAM y ROM. Por otra parte, el desarrollador puede optar por reutilizar funciones previamente desarrolladas o comprar bloques de funciones de propiedad intelectual. A diferencia del arreglo de compuertas lógicas estos dispositivos, no utilizan el concepto de celda básica y de elementos prefabricados en un chip.

En los 90 aparecen los circuitos integrados estructurados llamados SASIC *(Structured ASICs)*, para reducir los altos costos y el tiempo de desarrollo de los circuitos integrados de aplicación especifica. La estructura general de un SASIC, mostrada en la Figura 1.2, consiste en una matriz bidimensional o mar de módulos a los que se le unen una serie de bloques prefabricados como pueden ser bloques de entrada/salida, microprocesadores, memorias, etc. En muchos aspectos, este dispositivo es muy similar a un ASIC implementado con celdas estándares ya que los módulos pueden ser este tipo de celdas. La principal diferencia es que todas las celdas y muchas conexiones (como las de tierra y polarización) ya están integradas, por lo que el proceso de integración se limita a especificar unas cuantas capas de metal (aquellas correspondientes a conexiones

específicas). Estos nuevos dispositivos, cubren el espacio de aplicaciones existentes entre los ASICs y los dispositivos programables comunes (FPGAs y CPLDs).

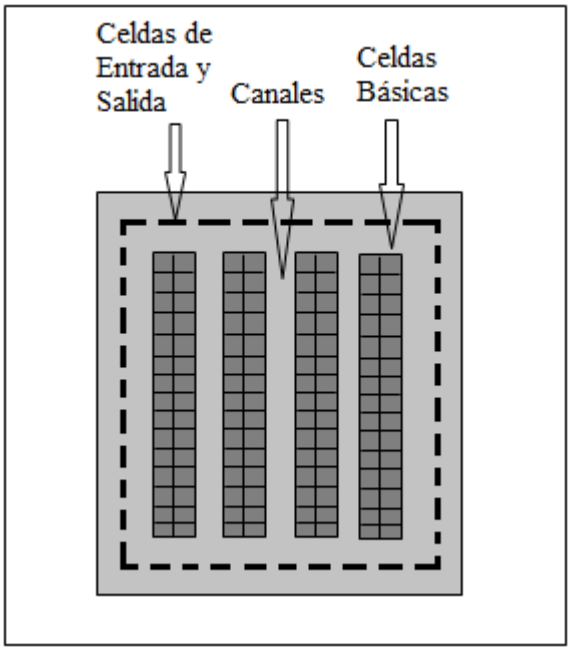

Figura. 1.1 Arquitectura de celdas básicas con canales

A mediados de los años 80, cuando los primeros FPGAs aparecieron en el mercado, eran utilizadas simplemente para implementar “glue logic” (lógicas simples para “pegar” grandes bloques o dispositivos lógicos), máquinas de complejidad media y algunas tareas de procesamiento simples. A comienzo de los años 90, con el aumento de la capacidad de los FPGAs, el mercado de las Telecomunicaciones empezó a utilizar esta tecnología, ya que necesitaba de grandes bloques de procesamientos. Posteriormente se aplicó FPGAs también en algunas aplicaciones industriales como en la industria automovilística. Frecuentemente, los FPGAs son usados para implementar prototipos de ASICs y para comprobar la implementación de algunos algoritmos en hardware. Cada vez más los FPGAs, son usados para productos finales, compitiendo así, directamente con los ASICs.

Al inicio de los años 2000, el desempeño de los FPGAs dio un gran salto, proporcionando a miles de compuertas lógicas, microprocesadores incorporados, y pines de alta velocidad para ser utilizados como insumos y productos. El resultado es que los FPGAs actualmente están presentes en campos tan diversos como la automoción, la electrónica de consumo, investigación espacial, en dispositivos de comunicaciones, funciones DSP, entre otros. Así están por encima de otras tecnologías. La tecnología FPGA

tiene una aplicación horizontal en todas las industrias que requieren computación a alta velocidad.

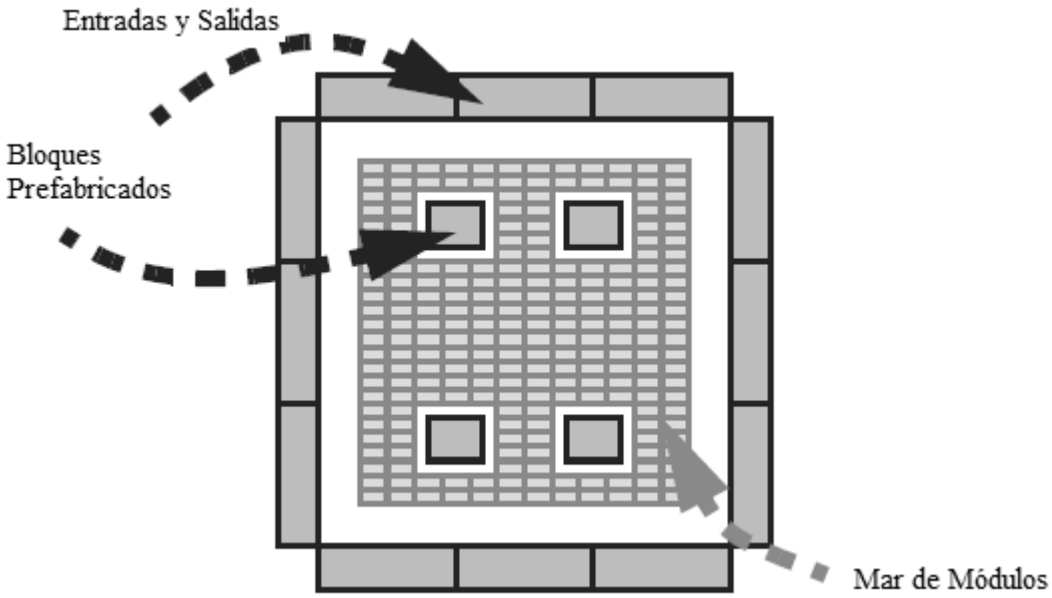

Figura. 1.2 Estructura de un ASIC estructurado.

Las FPGA son generalmente más lentas que sus contrapartes, los circuitos integrados de aplicaciones específicas ASIC, no pueden soportar diseños muy complejos, y consumen más energía. Sin embargo, tienen muchas ventajas tales como la reducción del tiempo para la salida al mercado de productos, la habilidad para ser reprogramadas después de haber salido al mercado a fin de corregir posibles errores, y reduce los costos de ingeniería tales como investigación, diseño y prueba de un nuevo producto.

Se sabe que la regla básica para distinguir un FPGA de un ASIC es el término reconfigurable, sin embargo para que un dispositivo de hardware sea reconfigurable, deben haber algunos mecanismos básicos en el mismo. El primer paso dado por las tecnologías de hardware configurables fue la tecnología de fusión de conexión (*fusible link)*, una de las primeras técnicas que permiten al usuario configurar su propio dispositivo de hardware. En esta técnica, el componente se fabrica con diferentes conexiones en el que hay fusibles que se pueden combinar por el usuario. En su estado inicial, todas las conexiones están intactas y para obtener la lógica deseada, el usuario tiene permiso para volar romper las conexiones de manera irreversible.

Posteriormente, surgió la tecnología de anti fusión de conexión *(antifuse link)* [11]. Esta tecnología en su programación inicial, cada ruta o pista tiene una resistencia tal que el

circuito se puede considerar un circuito abierto. Así, estos dispositivos pueden ser programados mediante la aplicación de pulsos de corriente con una tensión relativamente alta en sus entradas. Entonces, en términos físicos, la conexión puede ser formada a través de la saturación de los materiales semiconductores utilizados en su fabricación.

En 1984 Xilinx, introduce la primera era moderna FPGA, la cual estaba basada en CMOS [5] y utilizaba celdas SRAM para la configuración. A pesar de que el primer modelo representa una pequeña capacidad de implementación y pocos pines de entrada y salida, muchos de los aspectos de su arquitectura son utilizados actualmente. Estos dispositivos fueron basados en el concepto de bloques lógicos programables los cuales contenían tres tablas de *lookup* (LUTs), un registrador que podría funcionar como *flip-flop* o *latch*, un multiplexor y otros elementos menores. Así cada FPGA poseía 64 bloques lógicos programables, células SRAM apropiadas, las cuales permitían que cada uno de esos bloques puedan ser configurados para desempeñar una función diferente, y 58 entradas y salidas.

Los FPGA modernos han crecido enormemente, actualmente la arquitectura general de un FPGA se divide en aproximadamente 330.000 bloques lógicos programables que son individualmente más pequeños que un PLD. Se encuentran distribuidos a través de todo el chip en un mar de interconexiones programables y todo el arreglo se encuentra rodeado de 1100 entradas y salidas programables (IOBs), además de un gran número de bloques especializados. Un bloque lógico programable (CLB o slice) de FPGA es menos eficiente que un PLD típico, pero un chip FPGA contiene muchos más bloques lógicos que los PLD que contiene un CPLD del mismo tamaño. En la Figura 1.3 se muestra un esquema de la arquitectura general de un FPGA. Los elementos programables más importantes son los generadores reprogramables de función lógica, realizadas por las denominadas LUT, que son celdas de memoria SRAM y multiplexores para seleccionar la salida.

Las industrias han adoptado chips FPGA por el hecho de que los FPGAs combinan lo mejor de los ASICs y de los sistemas basados en procesadores. Ofrecen velocidades temporizadas por hardware y fiabilidad, pero sin requerir altos volúmenes de recursos para compensar el gran gasto que genera un diseño personalizado de ASIC. El silicio reprogramable tiene la misma capacidad de ajustarse que un software que se ejecuta en un sistema basado en procesadores, la diferencia es que no está limitado por el número de

núcleos de proceso disponibles, los FPGAs llevan a cabo diferentes operaciones de manera paralela, por lo que éstas no necesitan competir por los mismos recursos. Cada tarea de procesos independientes se asigna a una sección dedicada del chip, y puede ejecutarse de manera autónoma sin ser afectada por otros bloques de lógica. Entonces el rendimiento de una parte de la aplicación no se ve afectado cuando se introducen otros procesos.

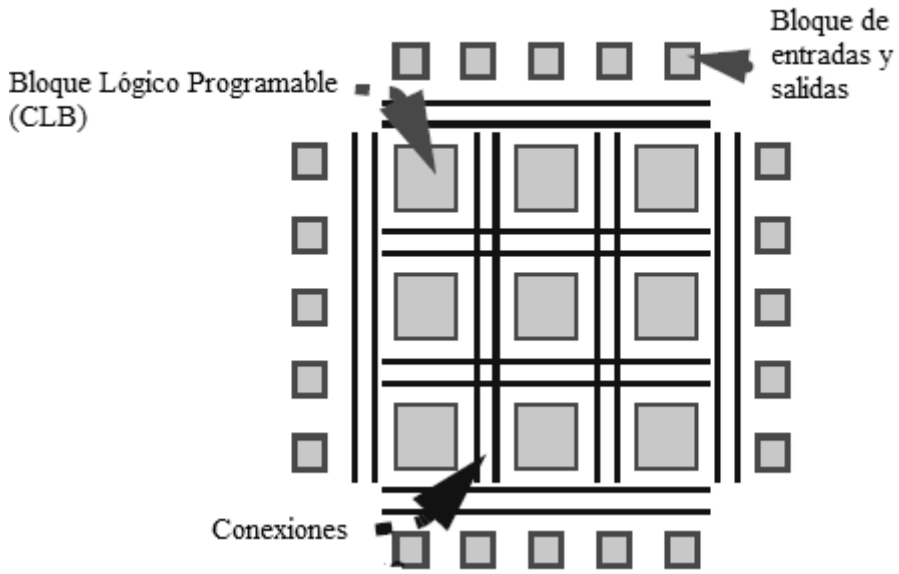

Figura. 1.3 Arquitectura General de un FPGA

1.3 Tecnologías de programación en los FPGAs

Existe una serie de tecnologías de programación en los FPGAs y sus diferencias tienen un efecto significativo en su arquitectura. Las tecnologías que han sido utilizadas históricamente incluyen EPROM, EEPROM, flash, SRAM y anti-fusibles. De estas en las FPGA modernas solamente se utilizan las SRAM, las memorias flash/EEPROM y los enfoques anti-fusible [11].

1.3.1 Tecnología de Celdas de Memoria Estática.

La mayoría de FPGAs actuales están basadas en el uso de configuraciones de celdas de memoria estática, etsa tecnología permite que los FPGAs sean configurados un sin número de veces. Esta tecnología se ha convertido en el enfoque dominante para los FPGAs, debido a sus dos ventajas principales: nueva programación y el uso de la tecnología de proceso CMOS estándar. Otra ventaja de esta técnica es que las nuevas ideas de diseño pueden ser rápidamente implementadas y probadas. Existen también algunas desventajas, una de ellas es que una memoria SRAM, por ser volátil, debe ser recargada

cada vez que se reinicie el sistema. El contenido se almacena mediante un proceso de configuración en el momento de encendido del circuito que contiene al FPGA. El contenido de estos bloques se pierde cuando se deja de suministrar la energía. Frente a esto, es necesario el uso especial de memoria externa, no volátil. Otro inconveniente es el tamaño, ya que una celda SRAM necesita 5 o 6 transistores y el elemento programable para interconectar las señales requiere al menos un transistor. Además hay la posibilidad de que la información de la configuración pueda ser interceptada y robada por la competencia, debido a que esta información debe ser cargada en el dispositivo al momento de encenderlo. Para eliminar este riesgo existen técnicas de encriptación en algunas familias de FPGAs.

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.

1.3.2 Tecnología Flash/EEPROM

Para contrarrestar las deficiencias de la tecnología SRAM, se puede usar las tecnologías de programación de puerta flotante que inyectan carga en una puerta que "flota" por encima del transistor. Este enfoque es utilizado en celdas de memorias flash o EEPROM. Estas celdas no son volátiles, es decir, no pierden la configuración cuando el dispositivo se apaga. Históricamente, las celdas de memoria EEPROM no se usaban directamente para cambiar las señales FPGA, estas células eran de uso general para implementar funciones AND cableadas en dispositivos PLD. Estos planteamientos ya no son usados, excepto para dispositivos de baja capacidad. Con los modernos procesos de fabricación de CI, es posible utilizar celdas de puerta flotante directamente como interruptores. Las celdas de memoria Flash, son las más empleadas debido eficiencia en área.

La tecnología de programación basada en flash ofrece varias únicas ventajas, lo más importante y ya mencionada es su “no volatilidad”. Esta característica elimina la necesidad de recursos externos necesarios para almacenar y configurar datos a diferencia de la

tecnología de programación basada en SRAM. Además, un dispositivo basado en flash puede funcionar inmediatamente después del encendido sin tener que esperar a la carga de datos. El enfoque flash también es más eficiente en cuanto al tamaño ya que no necesita el mismo número de transistores que la tecnología basada en SRAM. 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. En comparación con FPGAs basados en la tecnología de anti-fusibles, una alternativa no volátil, la cual se abordara posteriormente, los FPGAs basados en memoria flash son reconfigurables y se pueden programar sin ser removidos de la placa del circuito impreso.

El uso de una puerta flotante para controlar el transistor de conmutación añade complejidad al diseño porque se debe tener cuidado para garantizar que la tensión de la fuga de fuente sea lo suficientemente baja para evitar la inyección de carga en la puerta flotante. Este problema puede ser no tan preocupante en el futuro ya que los procesos irán requiriendo menores niveles de tensión. Una desventaja de estos dispositivos es que no pueden ser reprogramados infinitamente. En muchos casos los FPGAs son programados para una función específica, en este 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.

Recientemente el uso de almacenamiento flash en combinación con la tecnología SRAM ha surgido en algunos dispositivos de las marcas más importantes de FPGAs como Xilinx, Altera y Lattice. En estos dispositivos el chip memoria flash se utiliza para proporcionar almacenamiento no volátil mientras que las celdas de SRAM todavía se utilizan para controlar los elementos programables en el diseño. Esta nueva fusión compensa los problemas relacionados con la inestabilidad de los FPGAs basados en SRAM, tales como el costo de los dispositivos de almacenamiento adicionales o la posibilidad de la configuración de la interceptación de datos, manteniendo al mismo tiempo el infinito numero reconfiguraciones de los dispositivos basados en SRAM.

1.3.3 Tecnología Anti-fusible

La tecnología Anti-fusible está basada en estructuras que tienen una resistencia muy alta bajo circunstancias normales pero a través de la programación se puede crear un enlace de baja resistencia. El elemento programable, un anti-fusible, es usado directamente para transmitir las señales del FPGA.

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.

Existen también algunas desventajas significativas. Los FPGAs basados en la tecnología anti-fusible requieren un proceso CMOS no estándar, por lo tanto están por detrás del proceso de fabricación de los FPGAs basados en SRAM. Además, el mecanismo fundamental de la programación, que implica cambios significativos en las propiedades de los materiales en el fusible, conducen a algunos retos cuando los procesos de fabricación de un nuevo CI son considerados. De hecho, existe evidencia de que los anti-fusibles no son escalables como la mayoría de los dispositivos más avanzados que utilizan la tecnología de 0,15μm [12], esta tecnología está muchas generaciones detrás de la tecnología utilizada para los dispositivos con el nuevo estándar CMOS.

Al igual que la tecnología PROM un FPGA que utiliza este tipo de tecnología sólo se puede programar una vez. Esto hace que este tipo de FPGAs sean inapropiados para aplicaciones que requieren cambios. Esta “programabilidad única” hace imposible que se puedan hacer pruebas para detectar posibles fallas. Algunas fallas sólo serán descubiertas después de la programación, por lo tanto el rendimiento después de la programación será menor que el rendimiento de los dispositivos SRAM o los de puerta flotante. Algunos de los actuales dispositivos anti-fusibles tienen un 90% de confiabilidad.

A continuación en la Tabla 1.1 se presenta una comparación entre las tres tecnologías utilizadas actualmente en los FPGAs.

Tabla 1.1 Comparación de las tecnologías usadas en los FPGAs

	SRAM	Flash	Anti-fusible
Volatilidad	Sí	No	No
Reprogramable	Sí	Sí	No
Tamaño del elemento de almacenamiento	Grande (6 transistores)	Mediano (1 transistor)	Pequeño (0 Transistores)
Proceso de Fabricación	Estándar CMOS	Proceso Flash	Anti-fusible
Rendimiento de la programación	100%	100%	>90%

1.4 Formas de desarrollo e implementación

Los FPGAs se pueden programar de algunas maneras. Éstas pueden ser utilizando diagramas esquemáticos, lenguaje de descripción de hardware (HDL), compiladores de lenguajes de alto nivel y herramientas basadas en algoritmos. A continuación se detallaran los lenguajes más utilizados para la programación de los FPGAs.

1.4.1 Lenguaje Esquemático

El lenguaje esquemático se basa en una representación elemental en un nivel de compuertas lógicas. El primer paso para viabilizar la implementación y la simulación de una forma más automática fue la creación de una representación textual, llamada *netlist* a nivel de compuertas lógicas.

Basándose en esta representación textual, surgieron las herramientas de simulación y verificación de los circuitos que se deseaban implementar físicamente. Esas herramientas se las clasificó como CAE (*Computer Aided Engineering*). Posteriormente surgieron las herramientas de implementación que permitían el ruteamiento de conexiones para los dispositivos, que eran clasificados como CAD *(Computer Aided Design)[13]*. Finalmente en los años 80 surgió la tecnología EDA *(Electronic Design Automation)* y también herramientas que permitían una captura gráfica de lo esquemático para la creación interactiva de circuitos, uso de varios recursos como la biblioteca de símbolos esquemáticos, generación automática de *netlist* entre otras. Así se podía pasar un archivo *netlist* a un simulador para el análisis del funcionamiento del circuito, además de hacer una distribución y ruteamiento de los dispositivos.

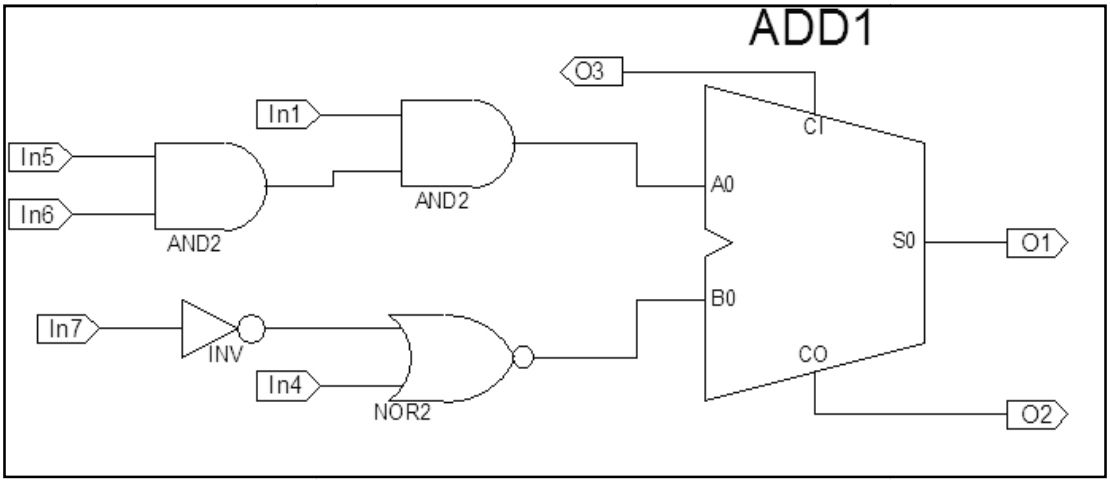

Figura. 1.4 Diagrama Esquemático

La forma más común de describir un circuito es mediante la utilización de esquemas que son una representación gráfica de lo que se pretende realizar. Con la aparición de herramientas EDA cada vez más complejas, que integran en el mismo marco de trabajo las herramientas de descripción, síntesis, simulación y realización; apareció la necesidad de crear un método de descripción de circuitos que permitiera el intercambio de información entre las diferentes herramientas que componen el ciclo de diseño. Al principio se utilizó un lenguaje de descripción que permitía, mediante sentencias simples, describir completamente un circuito. A estos lenguajes se les llamó *Netlist* puesto que eran un conjunto de instrucciones que indicaban las interconexiones entre los componentes de un determinado diseño.

Actualmente el lenguaje esquemático es una manera gráfica, como se observa en la figura 1.4, de representar por medio de diagramas electrónicos, un proyecto, resumiéndose en realizar las interconexiones entre los bloques. Los FPGAs desde su entrada al mercado se desarrollaban en plataformas esquemáticas. La programación a través de un lenguaje esquemático todavía se utiliza principalmente entre ingenieros antiguos y también por las personas que necesitan hacer pequeños cambios en sus proyectos. El lenguaje esquemático muestra algunas deficiencias en cuanto a grandes proyectos, ya que son muy sensibles a errores y se necesita de mucho tiempo de desarrollo. Este lenguaje no puede ser utilizado en proyectos modernos, los cuales tienen una mayor complejidad y mayor tamaño.

1.4.2 Lenguaje de Descripción de Hardware (HDL)

Los lenguajes de descripción de hardware (HDLs) son utilizados para describir la arquitectura y comportamiento de un sistema electrónico los cuales fueron desarrollados para trabajar con diseños complejos. El objetivo de un HDL es describir un circuito mediante un conjunto de instrucciones de alto nivel de abstracción para que el programa de síntesis genere un circuito que pueda ser implementado físicamente.

A partir de estos lenguajes simples, como el lenguaje esquemático, se descubrió el interés que podría tener el describir circuitos utilizando un lenguaje en vez de usar esquemas. Aún así resultaba más fácil programar de manera gráfica, a pesar de que con un lenguaje de programación se la edición sería más rápida y sencilla. Conforme las herramientas de diseño se volvieron más sofisticadas, y la posibilidad de desarrollar circuitos digitales mediante dispositivos programables era más viable, apareció la necesidad de poder describir los circuitos mediante un lenguaje de alto nivel de abstracción. Con la aparición de técnicas para la síntesis de circuitos a partir de lenguajes de alto nivel de abstracción, se comenzaron a utilizar los lenguajes de simulación para sintetizar circuitos.

HDL posee varias ventajas principales sobre la metodología tradicional de diseño a nivel de compuertas. Una de ellas es que se puede verificar el funcionamiento del sistema y probar su arquitectura antes de que sea implementado mediante compuertas, para verificar posibles errores y realizar cambios. Además este método permite optimizar el tiempo y reduce errores los cuales eran producidos al armar el circuito. Finalmente la programación HDL es más fácil de leer y comprender que los *netlist* o circuitos esquemáticos [13]. Existen dos tipos de HDL: Verilog creado en el año de 1985 y VHDL que es el que actualmente más se usa en el campo de los FPGAs.

Verilog es un lenguaje de descripción hardware utilizado para describir sistemas digitales, tales como procesadores, memorias o un simple *flip-flop*. Es uno de los estándares HDL disponibles hoy en día en la industria para el diseño hardware. Este lenguaje permite la descripción del diseño a diferentes niveles, denominados niveles de abstracción, entre los que se destacan tres: Nivel de Puerta, Nivel de transferencia de registro o nivel RTL y Nivel de comportamiento [14].

El Nivel de Puerta corresponde a una descripción a bajo nivel del diseño, también conocido como modelo estructural. El diseño se realiza mediante el uso de primitivas lógicas (AND, OR, NOT, etc...), conexiones lógicas y añadiendo las propiedades de tiempo de las diferentes primitivas.

En el Nivel de transferencia de Registro o RTL, los diseños especifican las características de un circuito mediante operaciones y la transferencia de datos entre registros. Mediante el uso de especificaciones de tiempo las operaciones se realizan en instantes determinados. Un diseño RTL tiene la propiedad de ser sintetizable, por esta razón es el nivel más utilizado en HDL.

El Nivel de comportamiento tiene una característica principal que es la total independencia de la estructura del diseño. El diseñador define el comportamiento del diseño. En este nivel, el diseño se define mediante algoritmos en paralelo.

VHDL es un lenguaje de descripción de hardware utilizado para describir circuitos en un alto nivel de abstracción. VHDL, viene de VHSIC (Very High Speed Integrated Circuit) Hardware Description Language desarrollado por el Departamento de Defensa de los Estados Unidos a finales los 70's. Se creó para diseñar, modelar, y documentar circuitos complejos, de tal forma, que tanto el humano como la máquina lo puedan leer y entender. Así, un diseño desarrollado por una empresa pueda entenderse por otra y también para que pudiera ser procesado por software con propósitos de simulación.

VHDL es un lenguaje con una sintaxis amplia y flexible que permite el modelado estructural, en flujo de datos y de comportamiento hardware. VHDL permite el modelado preciso, en distintos estilos, del comportamiento de un sistema digital conocido y el desarrollo de modelos de simulación. El objetivo del modelado es la simulación. Otro de los usos de este lenguaje es la síntesis automática de circuitos. En el proceso de síntesis, se parte de una especificación de entrada con un determinado nivel de abstracción, y se llega a una implementación más detallada es decir menos abstracta. La síntesis a partir de VHDL constituye hoy en día una de las principales aplicaciones del lenguaje con una gran demanda de uso.

VHDL es reconocido como un estándar de los lenguajes HDL por el Instituto de Ingenieros en Electricidad y Electrónica IEEE. Los estándares más utilizados en síntesis de

circuitos por la mayoría de las herramientas de diseño son el IEEE-1164 y el IEEE-1076.3. En la actualidad VHDL es un estándar de la industria para la descripción, modelado y síntesis de circuitos digitales. Al estar basado en un estándar los ingenieros de toda la industria de diseño pueden usar este lenguaje para minimizar errores de comunicación y problemas de compatibilidad [15].

1.5 Herramientas para la programación de FPGA's

Hoy en día cualquier proceso de ingeniería dispone de un soporte software que asiste al ingeniero en el desarrollo de sistemas complejos. Los sistemas electrónicos reconfigurables del tipo FPGA son un buen ejemplo de la complejidad que se puede alcanzar, pero esta no sería abarcable sin la ayuda de un entorno con herramientas que asistan en el proceso de diseño, simulación, síntesis del resultado y configuración del hardware. Existen varias herramientas de distintos fabricantes, se nombrará las más conocidas.

La herramienta más utilizada, es el software de la empresa Xilinx denominado ISE (*Integrated Software Environment*). Este software constituye un verdadero entorno EDA (*Electronic Desing Automation*). Otra herramienta importante es LabVIEW [19], que utiliza la tecnología de LabVIEW *Embedded* para ampliar el desarrollo gráfico de LabVIEW y descargar a los FPGAs, en hardware de E/S reconfigurable (RIO). LabVIEW es claramente adecuado para la programación del FPGA ya que representa paralelismo y flujo de datos.

La programación en VHDL para FPGAs, hoy en día también se la puede hacer en un entorno gráfico. Esto, gracias a la herramienta *System Generator*, que proporciona bloques de hardware básicos (retardos, memorias, operadores aritméticos, etc...) con los que es posible componer sistemas complejos. Este incluye un generador de código que, a partir del modelo, crea automáticamente una lista de conexiones (netlist) en VHDL sintetizable.

En el presente proyecto se utilizará *System Generator* e ISE de Xilinx, para el diseño de los transmisores BPSK y QPSK, y para la programación del FPGA. Estas herramientas se abordarán en el capítulo 3.

CAPÍTULO 2

SISTEMA DE TRANSMISIÓN DIGITAL

2.1 Introducción

El propósito de un sistema de comunicaciones es transmitir información desde un emisor hasta un receptor a través de un canal. En un sistema de comunicación digital, la transmisión de información se da por medio de señales digitales. Este tipo de sistemas pueden adoptar diferentes configuraciones, principalmente en cuanto a la fuente de información y al canal de transmisión. Existen sistemas digitales en el que la fuente genera información digital y esta información es transmitida a través de un canal digital. En la Figura 2.1 se muestra el esquema de un sistema de transmisión digital a través de un canal analógico.

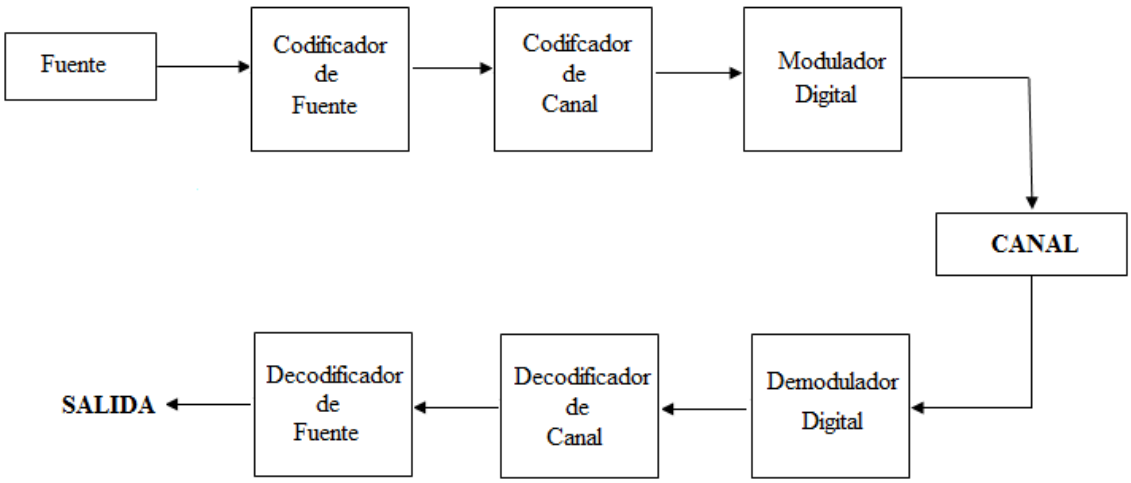

Figura. 2.1 Diagrama de bloques de un Sistema de Comunicación Digital

2.2 Codificación de Fuente

Codificación de fuente, es el proceso de codificar información comprimiendo los datos, para transmitirla con una mayor eficiencia. En un sistema de transmisión digital los símbolos son generados por una fuente y a la conversión se le llama codificación de fuente. El dispositivo que lleva a cabo este proceso es el codificador de fuente. Existen dos requerimientos importantes en un codificador de fuente eficiente. El primero es que las palabras de código producidas por el codificador estén en forma binaria. El segundo es que el código de fuente es decodificable en forma única, así la secuencia de fuente original puede reconstruirse perfectamente a partir de la secuencia binaria codificada. En la Figura 2.2 se muestra el diagrama de bloques de la codificación de fuente.

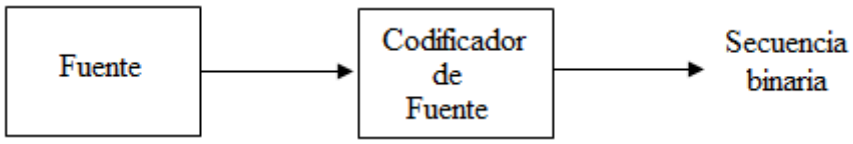

Figura. 2.2 Codificación de Fuente

Para cumplir el primer requerimiento, los símbolos generados por la fuente se transforman en símbolos de un código binario más adecuado para ser transmitido a través del canal de comunicaciones. Los símbolos codificados pueden ser comprimidos para tener una transmisión más rápida. Por ejemplo, más bits a los símbolos menos probables y menos bits a los más probables. El codificador de canal introducirá bits redundantes de una manera controlada, con el fin de fortalecer la información frente al ruido o a la interferencia que pueda conseguir en el canal, con el fin de corregir errores. Este código de fuente se conoce como código de longitud variable. Para el segundo requerimiento codificador debe cumplir que cada palabra de código debe decodificarse de forma única.

2.3 Codificación de Canal

En un canal de comunicaciones la información enviada se ve afectada por efectos de ruido o desvanecimiento, evitando así alcanzar el nivel de señal adecuado para obtener una relación señal a ruido (s/n) suficiente para recuperar la información sin errores. El objetivo de la codificación de canal es reducir la probabilidad de error, aumentando la resistencia del sistema digital al ruido del canal. Y así garantizar una s/n necesaria para garantizar una cierta tasa de error. La codificación del canal consiste en hacer corresponder la secuencia

de datos de entrada con una secuencia de entrada del canal y el proceso inverso a la salida. De esta manera el ruido del canal se hace mínimo.

La finalidad de la transmisión reside en que el receptor reciba exactamente lo que se le envía desde un emisor dado. La primera operación de correspondencia se la realiza en el transmisor por medio de un codificador de canal y la operación de correspondencia inversa se la hace en el receptor por medio de un decodificador de canal, como se muestra en la Figura 2.3. Por motivos de ejemplificación se han omitido la codificación y decodificación de fuente.

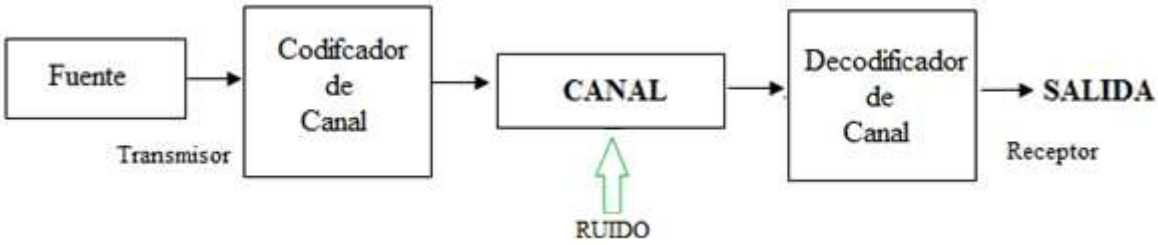

Figura 2.3 Diagrama de bloques reducido de un sistema de comunicación digital.

El principal problema se encuentra en que en el canal se puede presentar un ruido aleatorio. En consecuencia, se necesitará de un proceso mediante el cual podamos decidir qué mensaje, de todos los posibles, ha sido enviado. El enfoque que sigue es introducir redundancia en el codificador, de manera que la secuencia de fuente original sea reconstruida lo mejor posible. La introducción de redundancia en la codificación del canal tiene como finalidad mejorar la fiabilidad de la transmisión. La codificación de canal se puede considerar como el complemento de la codificación de fuente, ya que la primera introduce redundancia para mejorar la confiabilidad y la segunda para mejorar la eficiencia.

Dentro de la gama de códigos de canal podemos distinguir 2 grandes grupos: códigos detectores de error y códigos correctores de error. Los primeros permiten detectar que un mensaje ha sido recibido erróneamente, mientras que los segundos también permiten detectar un mensaje recibido erróneamente, pero al mismo tiempo corregir un número limitado de errores en éste. Los códigos detectores de errores se utilizan principalmente en aplicaciones que son insensibles a retardos, como lo son la transferencia de datos o imágenes. Por el contrario, los códigos correctores de error son implementados en sistemas

de comunicación donde los retardos son un factor determinante en el servicio que proveen, como lo son transmisión de audio o video en tiempo real.

Los códigos correctores de errores pueden ser clasificados en: Códigos de Bloque, Códigos Convolucionales y Turbocódigos. Para este proyecto se utilizará códigos convolucionales en el codificador de canal.

2.3.1 Códigos Convolucionales

Los Códigos Convolucionales son un tipo de códigos de control de error de tipo FEC *(Forward Error Correction)*, al igual que los códigos de bloque. Su operación consiste mapear una secuencia de bits a otra nueva secuencia de bits que corresponde a la salida del codificador. La codificación convolucional puede implementarse mediante un esquema como el que se muestra en la Figura 2.4.

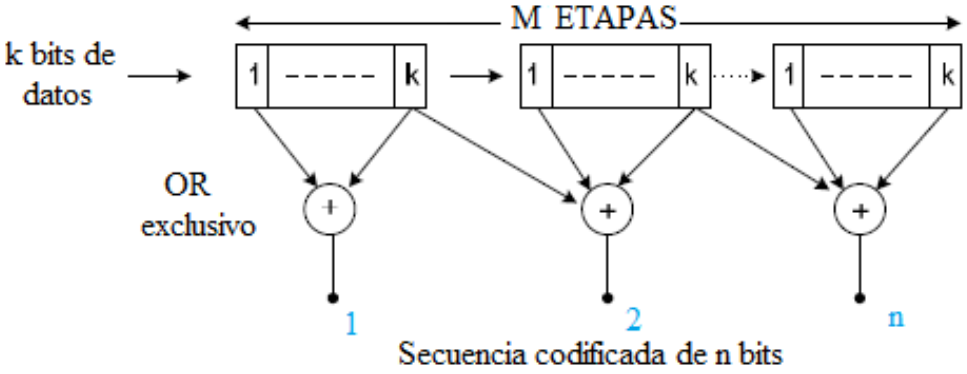

Figura. 2.4 Esquema de un Codificador Convolucional

El código convolucional se logra al pasar la secuencia de bits por un registro de desplazamiento de estados finitos. En el esquema, la secuencia de entrada es desplazada k bits a la vez, a lo largo de los registros de desplazamiento. También se puede apreciar que se tendrán *n* bits como secuencia de salida para cada *k* bits de entrada. El parámetro M es el número de memorias o etapas del codificador. Existen otros parámetros importantes en un codificador convolucional, como el parámetro *r* que representa la tasa de código, está y se pude representar como:

$$r = \frac{número\ de\ entradas}{número\ de\ salidas} = \frac{BITS\ de\ mensaje}{BITS\ enviados} = \frac{k}{n}\ bits/símbolo$$

Otro parámetro es *K*, que es la longitud de restricción o tamaño del código, está expresada en términos de bits de mensaje. Se define como el número de corrimientos sobre el cual un bit de un solo mensaje puede influir en la salida del codificador, es decir indica el número de bits de entrada de la cual depende la actual salida. Esta constante determina que tan poderoso y complejo es el código [16], y se puede expresar como $K = M + 1$. Finalmente la distancia libre representada por *dfree*, es la distancia mínima de código, es decir la distancia mínima entre cualquiera de las palabras de código y la palabra de código de puros ceros.

Cada trayectoria que conecta la salida con la entrada del codificador, se caracteriza con su respuesta al impulso, definida como la respuesta de esa trayectoria a un símbolo 1 aplicada en su entrada, con cada *flip-flop* en el conjunto codificador que inicialmente está en cero. También se puede caracterizar cada trayectoria con un polinomio generador, definido como la transformada de retorno unitario de la respuesta al impulso. El polinomio generador está definido por: $g^i(D) = g_0^{(i)} + g_1^{(i)}D + g_2^{(i)}D^2 + \cdots + g_M^{(i)}D^M$.

Donde D denota la variable de retardo unitario. El codificador convolucional completo se describe mediante un conjunto de polinomios generadores $\{g^{(1)}(D), g^{(2)}(D), \ldots, g^{(n)}(D)\}$. Otra forma de representar la trayectoria es mediante una matriz generadora:

$$g(i) = \begin{bmatrix} g_0^i \\ g_1^i \\ g_2^i \\ \vdots \\ g_M^i \end{bmatrix}$$

2.3.2 Representaciones gráficas de un codificador convolucional

Las propiedades estructurales de un codificador convolucional, se las pueden representar de manera gráfica mediante tres diagramas equivalentes: árbol de código, árbol de Trellis y diagrama de estados. En este capítulo se explicarán los 2 diagramas más utilizados, el diagrama de estados y el árbol de Trellis.

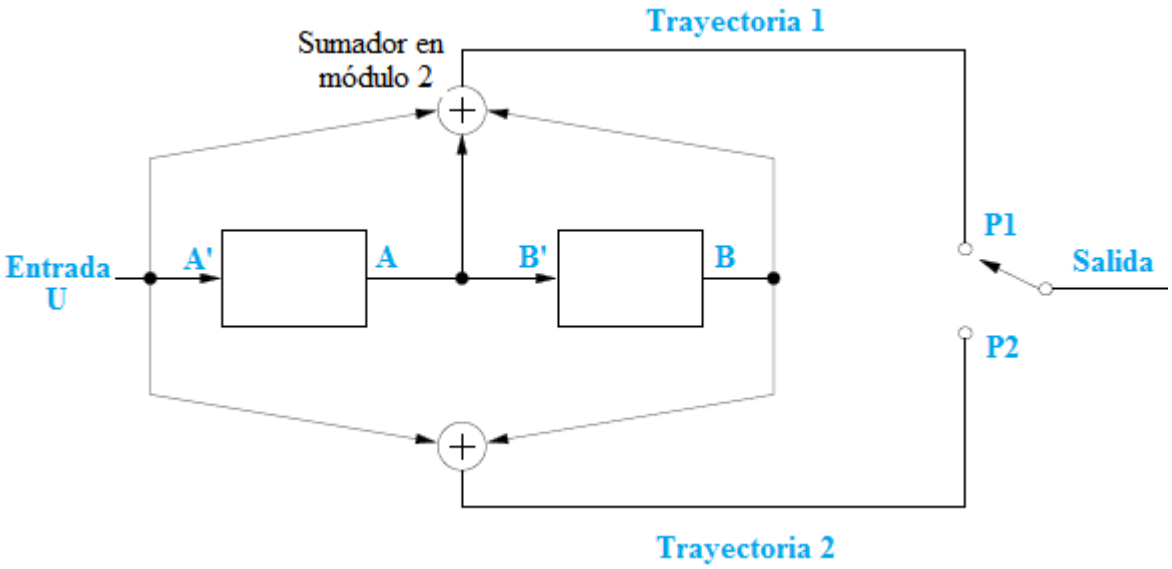

Figura. 2.5 Codificador convolucional de tasa ½ y longitud de restricción 3

Para facilitar esta explicación se usará un codificador convolucional de tasa ½ con una restricción *K*=3, mostrado en la Figura 2.5. El diagrama de estados se basa en el número de elementos de la memoria del circuito. Los bits utilizados para la codificación del mensaje se representan como cambios de estado en el tiempo. Para facilitar la construcción del diagrama de estados mostrado en la Figura 2.6, se puede realizar la Tabla 2.1, esta es una tabla de verdad en la cual se muestra la entrada, el estado presente, el próximo estado y la salida.

Tabla 2.1 Tabla de Verdad

ESTADO PRESENTE		ENTRADA	PRÓXIMO ESTADO		SALIDA	
A	B	U	A’	B’	P1	P2
0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	0	0	0	1	1
0	1	1	1	0	0	0
1	0	0	0	1	0	1
1	0	1	1	1	1	0
1	1	0	0	1	1	0
1	1	1	1	1	0	1

Con esta tabla es más simple apreciar como varían los estados del codificador y así conocer como, dependiendo de la entrada, será la salida. En el diagrama de estados, los bits que están en los cuadrados, representan los estados. Los números que están sobre las

líneas, representan la entrada y la salida del codificador. Por ejemplo si se tiene 0/11, quiere decir que ‘0’es la entrada y ‘11’es la salida.

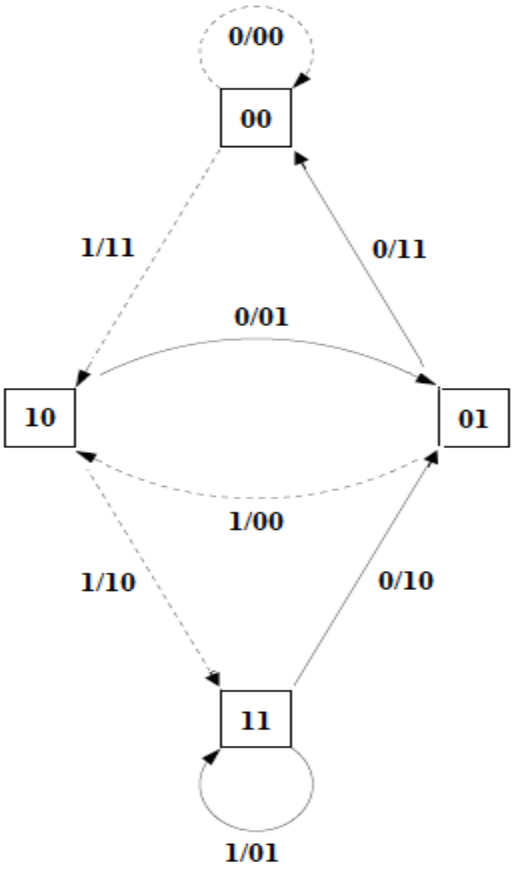

Figura. 2.6 Diagrama de Estados

El siguiente diagrama es el árbol de Trellis o enramado, y se muestra en la Figura 2.7. Este es el más utilizado ya que permite realizar la codificación de la manera más sencilla. Este es un diagrama en forma de red. Se parte del estado inicial del codificador. A partir de esto 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 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 distinguir los símbolos de entrada ‘0‘ y ‘1’, en este diagrama, una entrada ‘0’ se dibuja en línea continua, mientras que una entrada ‘1’ se dibuja en línea punteada.

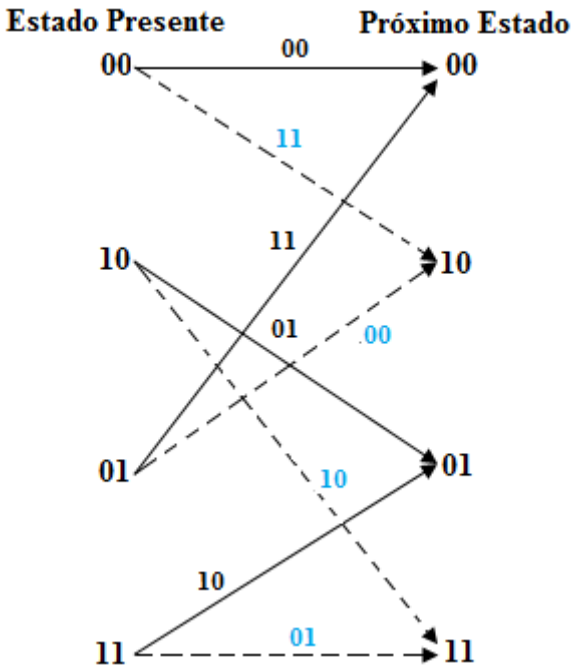

Figura. 2.7 Árbol de Trellis

2.4 Modulación Digital

Los esquemas de modulación digital, transforman señales digitales en formas de onda que son compatibles con el canal de comunicaciones. La modulación digital proporciona más capacidad de información, compatibilidad con servicios de datos digitales, mayor seguridad de datos, mejor calidad en las comunicaciones, y más rápida disponibilidad del sistema. Existen varios problemas al momento de desarrollar los sistemas de comunicaciones, estos pueden ser el ancho de banda disponible, la potencia admitida y el nivel de ruido inherente del sistema. El espectro de radiofrecuencia es compartido, aún así, cada día hay más usuarios en los servicios de comunicaciones, entonces la demanda del espectro aumenta.

Los esquemas de modulación digital tienen mayor capacidad para transmitir grandes cantidades de información que los esquemas de modulación analógica. Existen dos categorías en modulación digital. Una categoría utiliza una portadora de amplitud constante y lleva la información en variación de fase o frecuencia, estas modulaciones son FSK *(Frecuancy Shift Keying***)** y PSK *(Phase Shift Keying***)**. La otra categoría transmite la información en las variaciones de amplitud de la portadora y se conoce como ASK *(Amplitude Shift Keying***)** [17].

2.4.1 Modulación en Fase

En el presente proyecto, se utilizaran modulaciones que son variaciones de la modulación PSK. Estas variaciones son: Modulación BPSK *(Binary Phase Shift Keying***)** y QPSK *(Quadrature Phase Shift Keying***)**.

La modulación digital BPSK es una de las más simples, puesto que emplea dos símbolos con un bit de información cada uno. En la Figura 2.8 se puede observar el esquema del modulador. Esta modulación también es la que tiene mayor inmunidad al ruido ya que la diferencia en fase entre símbolos es máxima. La fase de la señal portadora de amplitud constante se mueve entre 0º y 180º. La velocidad de transmisión es la más baja de las modulaciones en fase.

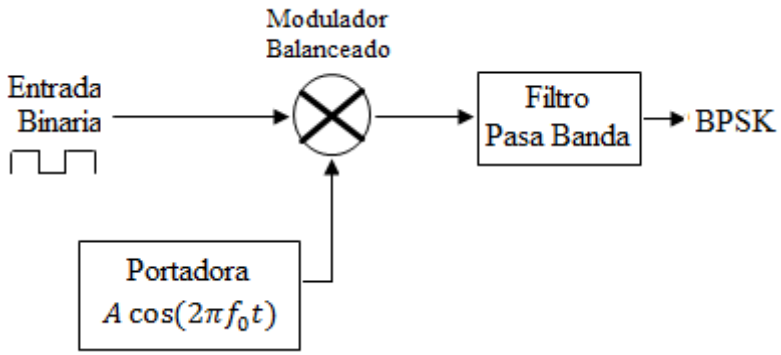

Figura. 2.8 Esquema del modulador BPSK

En este tipo de modulación, un símbolo binario ‘1’ y ‘0’, modulan la fase de la portadora. La portadora se puede representar de la siguiente manera:

$$S(t) = A\cos(2\pi f_0 t)$$

En donde *A* representa el valor pico de la portadora senoidal, si se habla de una carga de registro estándar de 1Ω, la potencia disipada sería $P = \frac{1}{2}A^2$.

El funcionamiento del modulador radica en que cuando el símbolo binario cambia, la fase de la portadora varía en 180 grados. Las señales de entrada, tanto la binaria como la portadora pasan a través de un modulador balanceado, este modulador actúa como un conmutador para invertir la fase.

Entonces cuando el símbolo es un ‘1’ lógico, la fase a la salida del modulador se representa así:

$$S1(t) = A\cos(2\pi f_0 t)$$

Cuando el símbolo es un ‘0’ lógico, la fase a la salida del modulador será:

$$S2(t) = -A\cos(2\pi f_0 t).$$

Entonces la ecuación general que define a la modulación BPSK es:

$$S(t) = b(t)A\cos(2\pi f_0 t)$$

En donde $b(t)$, tiene el valor de 1 cuando un ‘1’ es transmitido y -1 cuando es un ‘0’ [18]. La modulación BPSK es utilizada en transmisores de bajo costo y que no requieran velocidades altas. También es utilizada en estándares RFID como el ISO 14443, que se ha adoptado en pasaportes biométricos o tarjetas de crédito. Finalmente BPSK es usada en la banda de 868-915MHz. En la Figura 2.9 se representa gráficamente la modulación BPSK, para apreciar el cambio de fase de mejor manera.

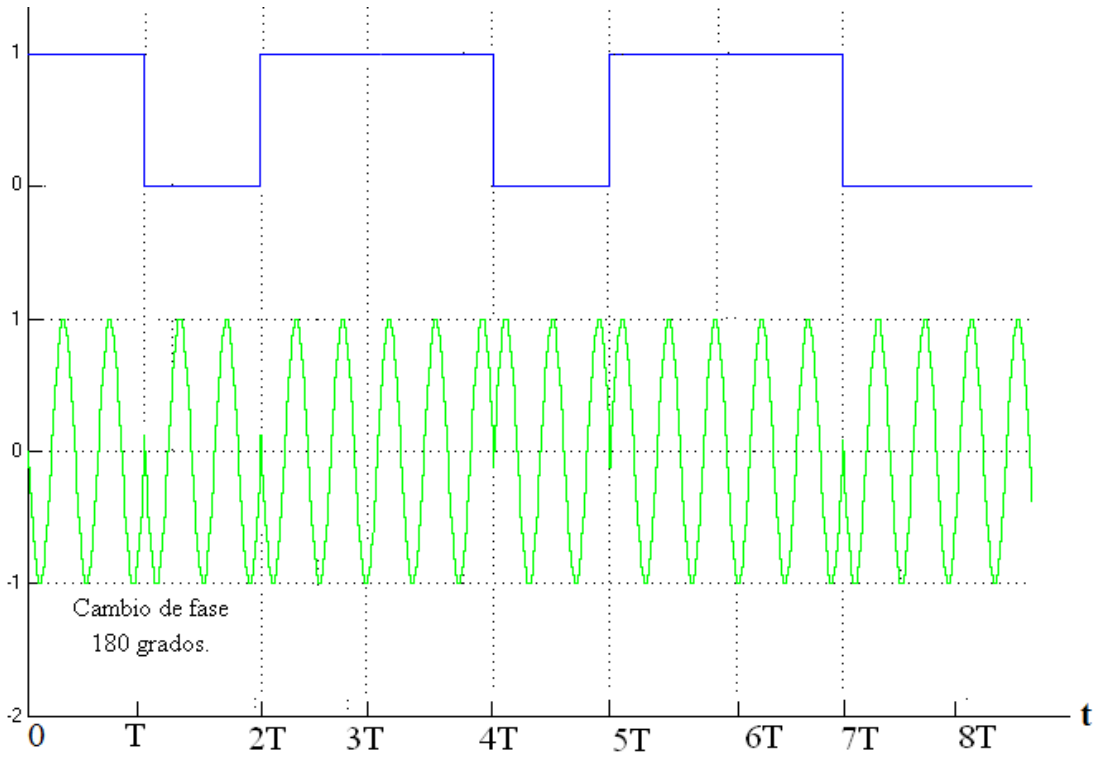

Figura 2.9 Modulación BPSK.

La modulación QPSK es un tipo más común de modulación de fase, es una forma de modulación angular o constante. Es una técnica M-aria de modulación con M=4. Lo que significa que en esta modulación la señal portadora puede tener 4 fases de salida diferentes. Debido a esto tiene que haber cuatro diferentes condiciones de entrada. Se necesita 2 bits en la entrada del modulador para producir las 4 posibles condiciones a la salida. Los datos de entrada binarios están compuestos por grupos de dos bits, y son llamados *dibits.* Cada *dibit* de 2 bits introducidos al modulador ocasiona un solo cambio de salida; así, que la razón de cambio de salida (razón de baudio) es la mitad de la razón *dibits* de entrada.

El funcionamiento de un modulador QPSK, como se muestra en la Figura 2.10, consiste en que dos bits se sincronizan en el convertidor, después de haber entrado ambos bits en serie, salen simultáneamente en paralelo. Si se tiene un '1' lógico será +1V y si se tiene un '0' lógico será -1V. Un bit se dirige por el canal I y modula a la portadora en fase, mientras que el bit que se dirige por el canal Q, modula la portadora cuando esta ha sido desfasada 90º. Una vez que el *dibit* se ha dividido en los canales I y Q, la operación es igual que en un modulador BPSK. Un modulador QPSK, trabaja como si se tuviera dos moduladores BPSK en paralelo. La modulación QPSK posee la propiedad de la ortogonalidad ya que se envía una portadora seno y simultáneamente se envía una portadora coseno. Esta propiedad hace posible que QPSK se pueda utilizar para enviar información al doble de velocidad que BPSK, con el mismo ancho de banda.

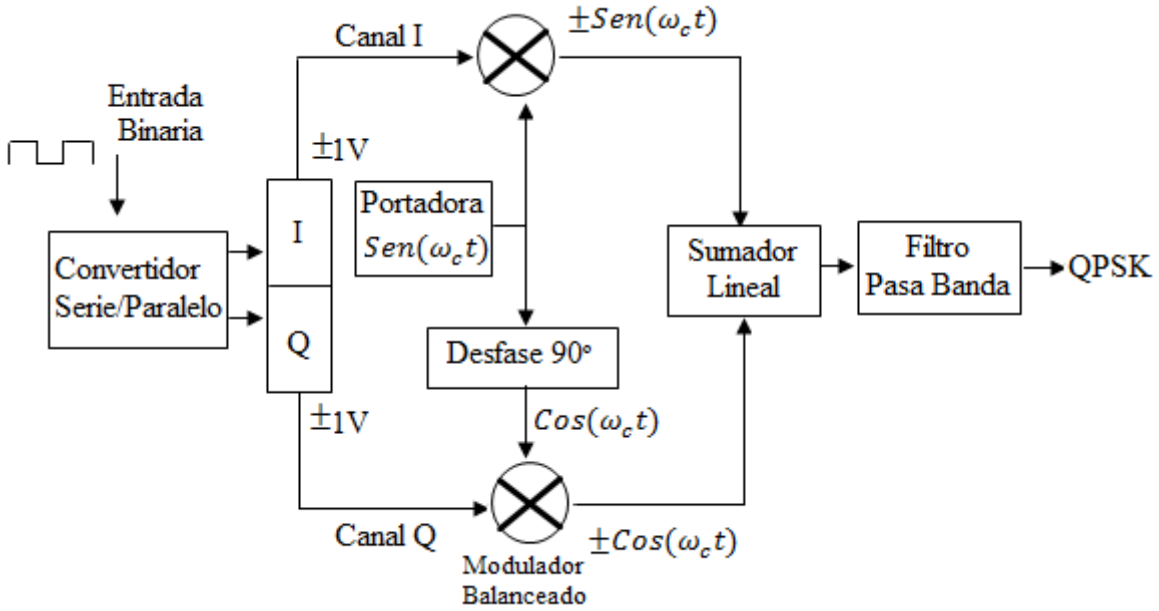

Figura. 2.10 Esquema del modulador QPSK

A la salida del modulador balanceado I, dependiendo del bit de entrada, puede haber dos posibles fases:

$$+Sen(\omega_c t) \quad o \quad -Sen(\omega_c t)$$

De igual manera, a la salida del modulador balanceado Q:

$$+Cos(\omega_c t) \quad o \quad -Cos(\omega_c t)$$

La ecuación general que define a la modulación QPSK, se representa así:

$$S_{QPSK}(t) = \sqrt{\frac{2E}{T}} cos\left[2\pi f_c t + (i-1)\frac{\pi}{2}\right] \qquad 0 \leq t \leq T$$

Donde E es la energía símbolo, T es el tiempo de símbolo y es igual a dos veces el periodo de bit y la variable i, toma los valores de 1, 2 ,3 y 4.

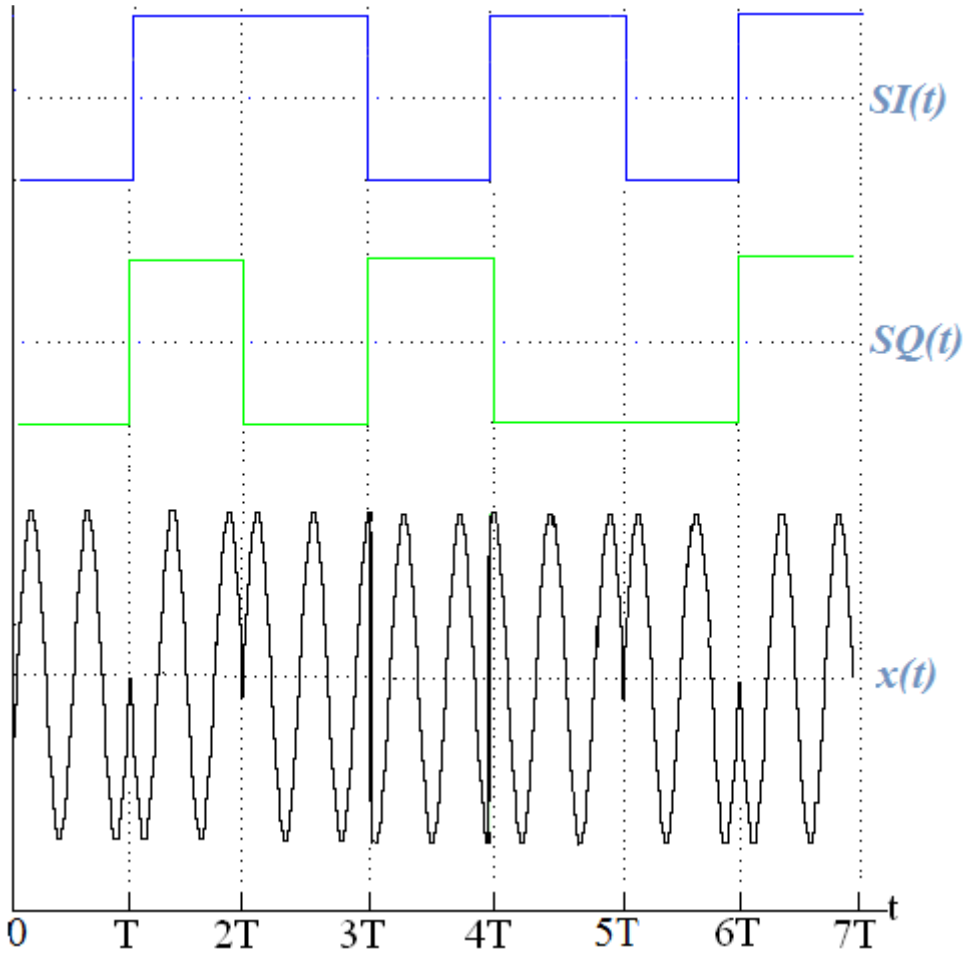

Figura. 2.11 Modulación QPSK

Siendo así, las 4 fases a la salida del modulador tienen la misma amplitud y se las puede representar de la siguiente manera:

- $sen(\omega_c t) + cos(\omega_c t)$
- $sen(\omega_c t) - cos(\omega_c t)$
- $-sen(\omega_c t) + cos(\omega_c t)$
- $-sen(\omega_c t) - cos(\omega_c t)$

El ancho de banda mínimo para una señal QPSK se calcula a través de la siguiente ecuación:

$$f_{N.Q-PSK} = B = \frac{f_b}{2}$$

En la Figura 2.11 se representa gráficamente la modulación QPSK, para apreciar el cambio de fase de mejor manera. En donde $SI(t)$ es la componente en fase y $SQ(t)$ es la componente en cuadratura. Esta modulación se utiliza ampliamente en aplicaciones CDMA *(Code Division Multiple Access)* de servicio celular, servicios inalámbricos locales, transmisión de datos por satélire y DVB-S (Radiodifusión de Vídeo Digital - Satélite).

2.4.2 Diagrama de Constelación

La representación de señales PSK suele hacerse con los llamados "diagramas constelación". Un Diagrama de Constelación es una representación de un esquema de modulación digital en el plano complejo. Los ejes real e imaginario suelen ser llamados I (*In-phase*) y Q (*Cuadrature*). Los puntos en la constelación representan símbolos de modulación los que componen el alfabeto, es decir todas las "palabras" que podrán usarse en un intercambio de información.

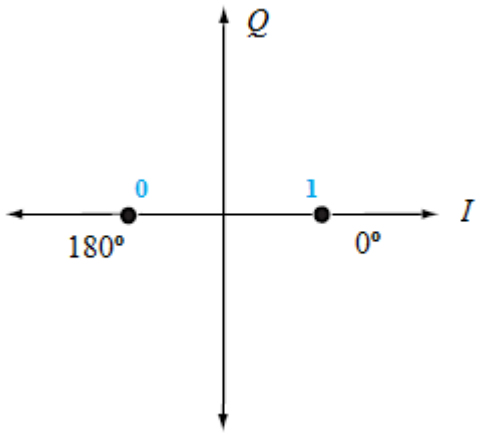

Figura. 2.12 Diagrama de constelación para la modulación BPSK.

Para la modulación BPSK, en un diagrama I/Q, el estado I tiene dos valores diferentes, como se observa en la Figura 2.12. Hay dos posibles ubicaciones en el diagrama de constelación. En esta modulación existen dos símbolos, esto quiere decir que un '1' ó un '0' binario se pueden enviar. La velocidad de símbolo es un bit por símbolo. Cuando se tiene una entrada binaria de un '0' lógico, la salida en fase es 180º. Cuando la entrada binaria es un '1' lógico, la salida en fase es 0º.

Para la modulación QPSK, la señal varía en incrementos de 90 grados desde 45 a 135, -45, o -135 grados. Estos puntos se pueden representar fácilmente usando un diagrama *I/Q*. Se necesitan dos valores de I y dos valores de Q como se observa en la Figura 2.13. Esto da una tasa de dos bits por símbolo.

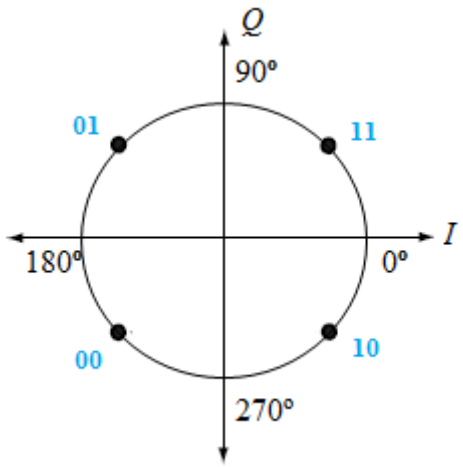

Figura. 2.13 Diagrama de constelación para la Modulación QPSK.

Tabla 2.2 Posibles fases de salida QPSK

Entrada Binaria		**Fase de Salida**
I	**Q**	
0	0	-135º
0	1	-45º
1	0	135º
1	1	45º

El bit I modula una portadora que está en fase con el oscilador de referencia y el bit Q modula una portadora que está 90° fuera de fase o en cuadratura con la portadora de referencia. En esta modulación se asocian parejas de bits a los estados de la señal. Existen

4 estados, por lo tanto, es un tipo de modulación el doble de eficiente, en cuanto a potencia, que la modulación BPSK. Cada una de las cuatro posibles fases de salida tiene, exactamente, la misma amplitud. En la Tabla 2.2 se observa la fase de salida de acuerdo a la entrada binaria.

CAPÍTULO 3

DESCRIPCIÓN DE SOFTWARE Y HARDWARE

3.1 MATLAB

MATLAB es una herramienta matemática que integra funciones de visualización con un lenguaje flexible. Ha tenido gran aceptación como herramienta para ciencia e ingeniería. Su arquitectura abierta permite la integración con otras herramientas. Incluye herramientas para: Adquisición de datos; análisis de datos; visualización y procesamiento de imágenes; desarrollo de algoritmos; modelado, simulación y desarrollo de aplicaciones. La versión que se utiliza en este proyecto es MATLAB R2007a.

Los modelos algorítmicos de MATLAB, pueden ser incorporados en *System Generator* a través de *AccelDSP*, que es una herramienta de síntesis, en lenguaje MATLAB de alto nivel para diseñar bloques DSP para FPGA's. *AccelDSP* incluye una gran síntesis algorítmica que toma el punto flotante de MATLAB como entrada y genera un modelo totalmente programado en punto fijo para usarlo con *System Generator*.

3.1.1 Simulink

Simulink es una herramienta integrada en MATLAB para modelar, simular y analizar sistemas dinámicos. Esta herramienta permite realizar diferentes sistemas mediante un método gráfico que se puede combinar con el método del lenguaje de programación utilizado en MATLAB. Muy utilizado para diseño de sistemas de control, DSP, comunicaciones, entre otras aplicaciones. Permite construir diagramas a bloques, simular el comportamiento del sistema, evaluar su eficiencia y mejorar el diseño. El ambiente gráfico de Simulink es el mismo que se utiliza en *System Generator*. De esta manera es posible unir bloques de Simulink y de *System Generator* para realizar simulaciones.

3.1.2 Xilinx *System Generator*

La versión que se utiliza en este proyecto es *Xilinx System Generator* (XSG) 10.1. *System Generator* es una herramienta de diseño de alto nivel de Xilinx que permite el uso de *MathWorks* basado en el modelo del entorno de Simulink. En la Figura 3.1 se observa el entorno de *Simulink* utilizando bloques de la librería de Xilinx. La experiencia previa con Xilinx FPGAs o metodologías de diseño RTL no se requieren al usar *System Generator*. Es decir no requiere una programación en VHDL o Verilog. A partir de un esquema modelado es posible generar el código del proyecto para su implementación en hardware. Los diseños son capturados en un ambiente DSP utilizando un *Blockset* Xilinx específico. Todos los pasos intermedios para el desarrollo de aplicaciones con FPGA incluyendo la síntesis, el lugar y la ruta se realizan automáticamente para generar un archivo de programación para el FPGA.

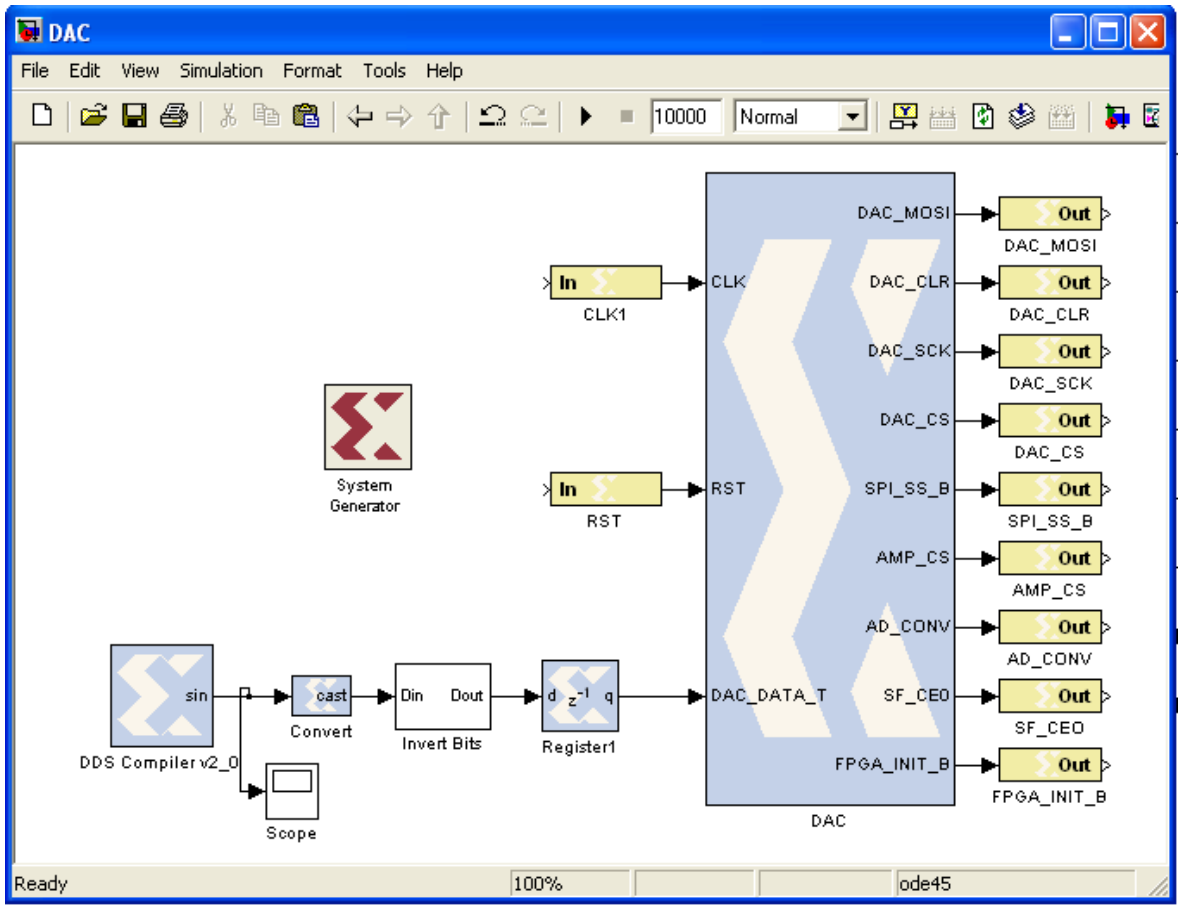

Figura. 3.1 Entorno Simulink con bloques de Xilinx

Xilinx System Generator, es un ambiente de diseño integrado (IDE) a nivel del sistema para FPGAs, que utiliza Simulink, como entorno de desarrollo y se hace presente en forma de blockset. Tiene un flujo de diseño integrado, para pasar directo al archivo de configuración (*.bit) necesario para la programación del FPGA. En la Figura 3.2 se muestra el flujo de diseño de *System Generator*. XSG genera automáticamente el código VHDL. Una de las características más importantes 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.

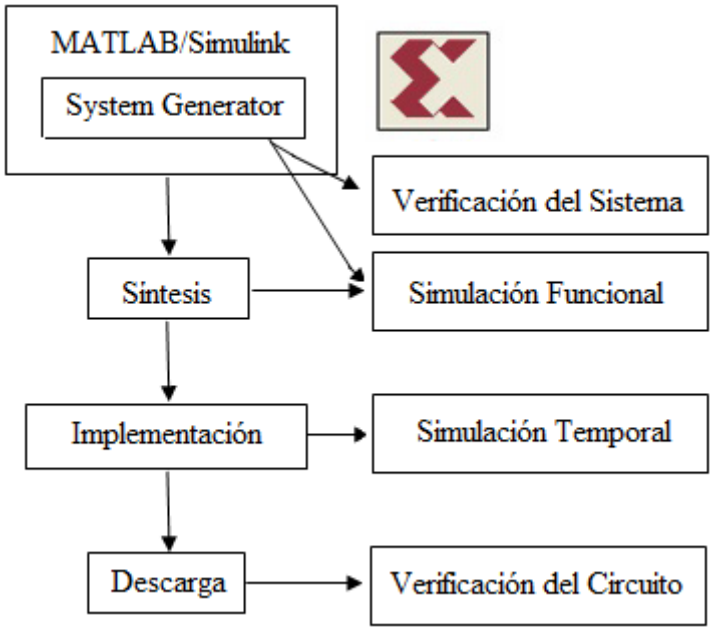

Figura. 3.2 Flujo de Diseño en *System Generator*

XSG fue creado principalmente para trabajar con aplicaciones DSP complejas, pero también se puede usar en otro tipo de aplicaciones. Los bloques en XSG operan con valores booleanos o valores arbitrarios en punto fijos. De esta manera se puede dar una mejor aproximación a la implementación en hardware.

La implementación en XSG, es muy eficiente ya que usa los núcleos de propiedad intelectual (*IP cores*), desde los más simples como operaciones aritméticas, hasta los más complicados como algoritmos complejos de DSP. Estos están en los *toolboxes* de Simulink como se puede ver en la Figura 3.3. Más de 90 módulos DSP se presentan en el DSP *Blockset* Xilinx para Simulink. Estos bloques aprovechan los generadores principales de

Xilinx IP *(Intelectual Property)* para entregar resultados óptimos para el dispositivo seleccionado.

La clave para la utilización de XSG, es el uso de dos bloques muy importantes, estos son *Gateway In* y *Gateway Out*. El bloque *Gateway In* representa la entrada del sistema. En procesos hechos en el ambiente Simulink, este bloque representa la interface entre Simulink y *System Generator*. Cada bloque de *Gateway In* que se haya utilizado en el diseño, representa una entrada física del FPGA, al momento de la implementación. Estos bloques deben ser programados dependiendo del pin que se requiere utilizar en la tarjeta. Es necesario también establecer el tipo de la entrada (booleana, punto fijo con señal o punto fijo sin señal), y el tamaño en bits. El bloque *Gateway Out*, representa el paso de punto fijo a punto flotante, cuando se realiza la simulación en Simulink. También representa los pines de salida del FPGA, los llamados pines de chip, en el momento de la implementación.

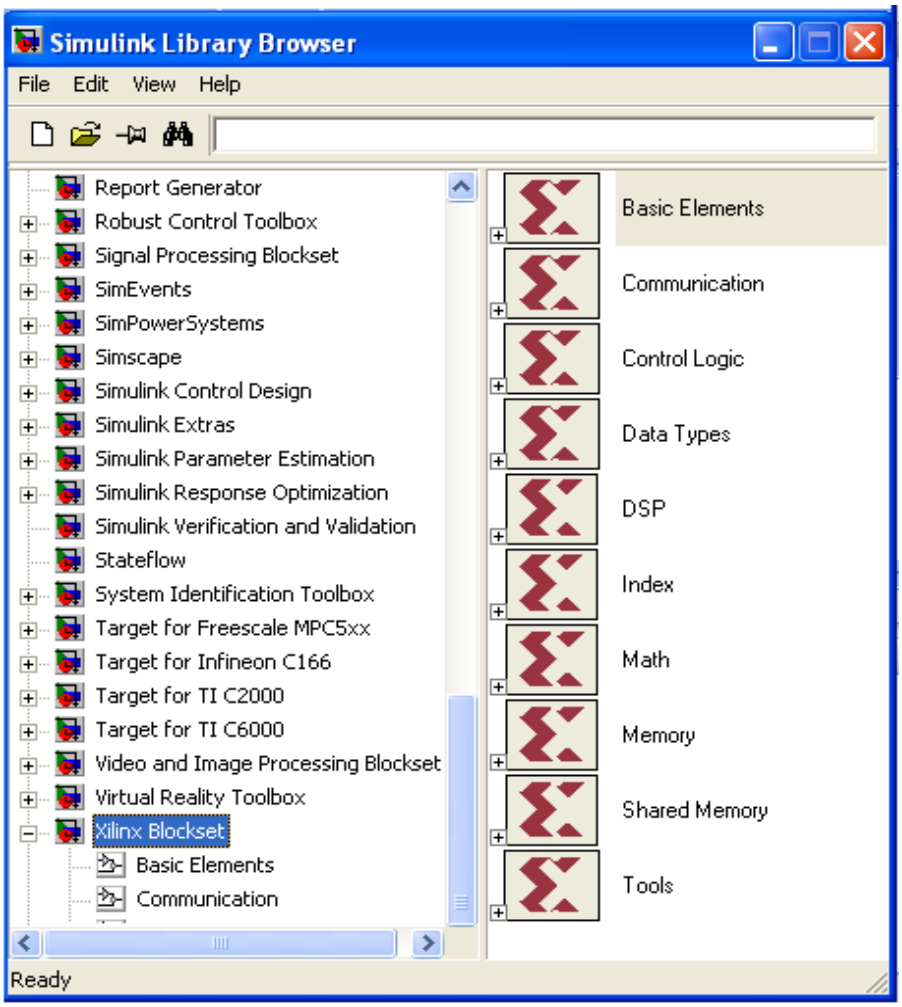

Figura. 3.3 Bloques Xilinx para Simulink

Todo sistema desarrollado en Simulink, para FPGA, debe incluir en su diseño un bloque denominado *System Generator*. Este bloque está disponible en los *toolboxes* Xilinx de Simulink, como se observa en la parte superior derecha de la Figura 3.4.

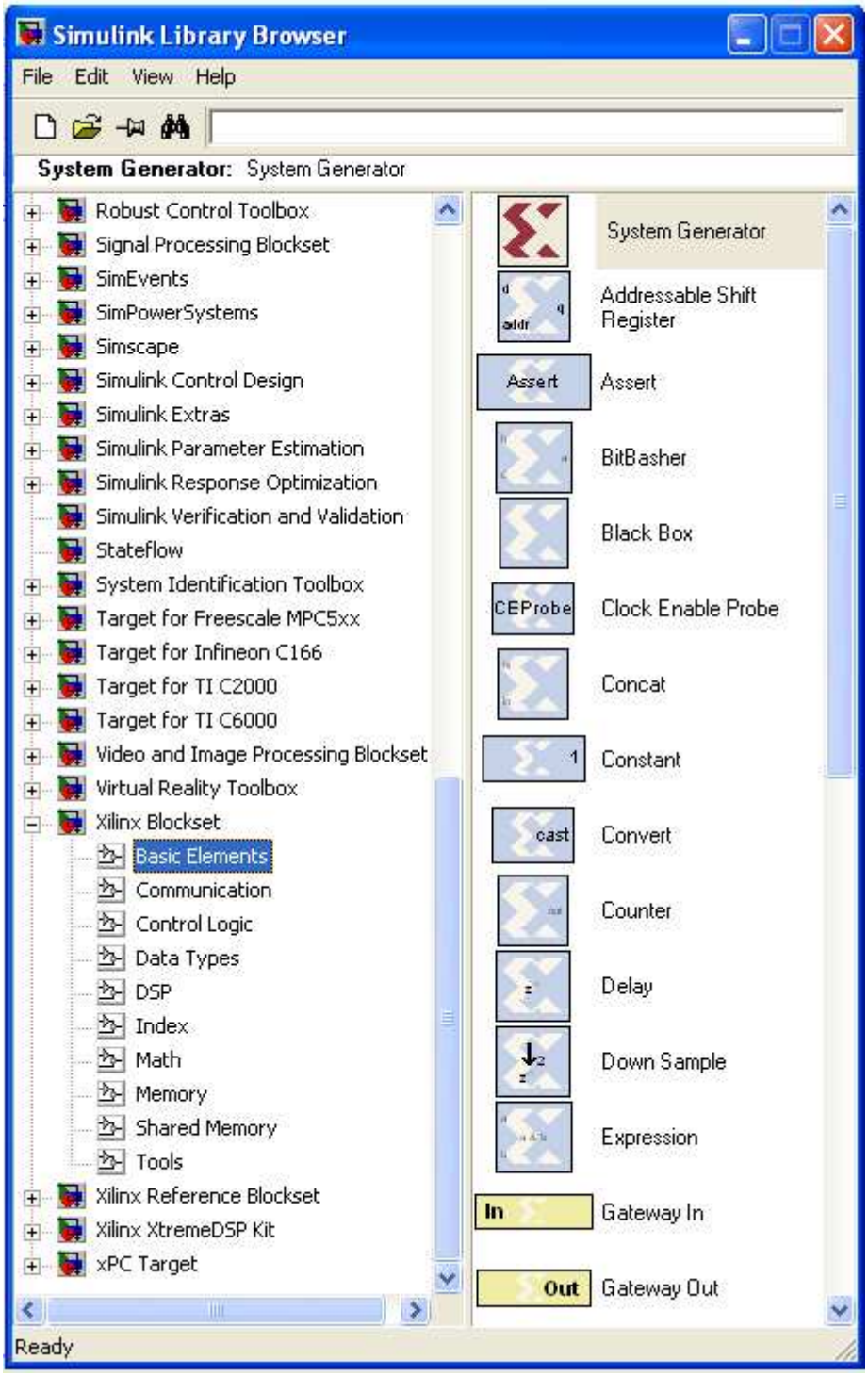

Figura. 3.4. Bloques Xilinx Básicos para Simulink

En este bloque se especifica las características del proyecto, como la familia del dispositivo en el que se lo va a implementar, la frecuencia de operación del circuito, la herramienta utilizada para la síntesis, el lenguaje de descripción de hardware utilizado, que puede ser VHDL o Verilog, y el tipo de compilación a utilizar.

Existen varios tipos de compilaciones. Una de ellas puede ser, HDL *netlist*. En esta compilación se genera un archivo *netlist* basado en archivos VHDL o Verilog. Estos archivos contienen la lógica y la información de restricción del desarrollo, es decir el archivo **.ucf*, en donde se asignan los pines que se utilizan en el diseño. El archivo generado puede ser considerado como un NPI, y se lo puede utiliza en otro proyecto si lo amerita. Otra alternativa y la que se utilizará en este proyecto en la compilación del modo *bitstream*. Con este modo, a partir de *System Generator*, se genera directamente el archivo .bit que será grabado en el FPGA. Para esta compilación los bloques *Gateway In* y *Gateway Out*, deben ser correctamente nombrados, para que coincidan con los pines de la tarjeta los cuales serán mapeados, y así se podrá generar el archivo **.ucf*. Una vez generado el archivo **.bit*, se lo cargará en el FPGA con la ayuda de la herramienta IMPACT de ISE, que se abordará más adelante.

Existe otro recurso muy importante en *System Generator*, es un bloque llamado "*Black-Box*". Esto permite que una función en VHDL pueda ser incorporada a otro proyecto en *System Generator*. Gracias a este bloque es posible utilizar nuevamente funciones ya hechas en VHDL o puede ser una alternativa si no existe un bloque que se necesite. Este bloque se explica detalladamente en el capítulo 4.

Finalmente las grandes ventajas que brinda *System Generator*, como la existencia de interfaces de co-simulación, que posibilitan la incorporación de un FPGA directamente en el lazo de Simulink. Además de, un flujo de diseño esquemático de alto nivel, potentes herramientas de visualización de MATLAB y la metodología basada en IP COREs, hacen de este software, una herramienta muy efectiva a la hora de reducir tiempo y costo en los diseños.

3.2 ISE

Como ya se mencionó en el capítulo 1, esta es una de las herramientas más utilizadas para la programación de FPGAs. En este proyecto, esta herramienta se la utilizará para la programación en lenguaje VHDL del DAC (*Digital Analog Converter)* y para la etapa de

implementación física del proyecto. Esta etapa consiste en la generación del archivo binario y la programación del FPGA. Se describirá brevemente el funcionamiento de esta herramienta. En la Figura 3.5 se muestran los pasos para el flujo de diseño con FPGA.

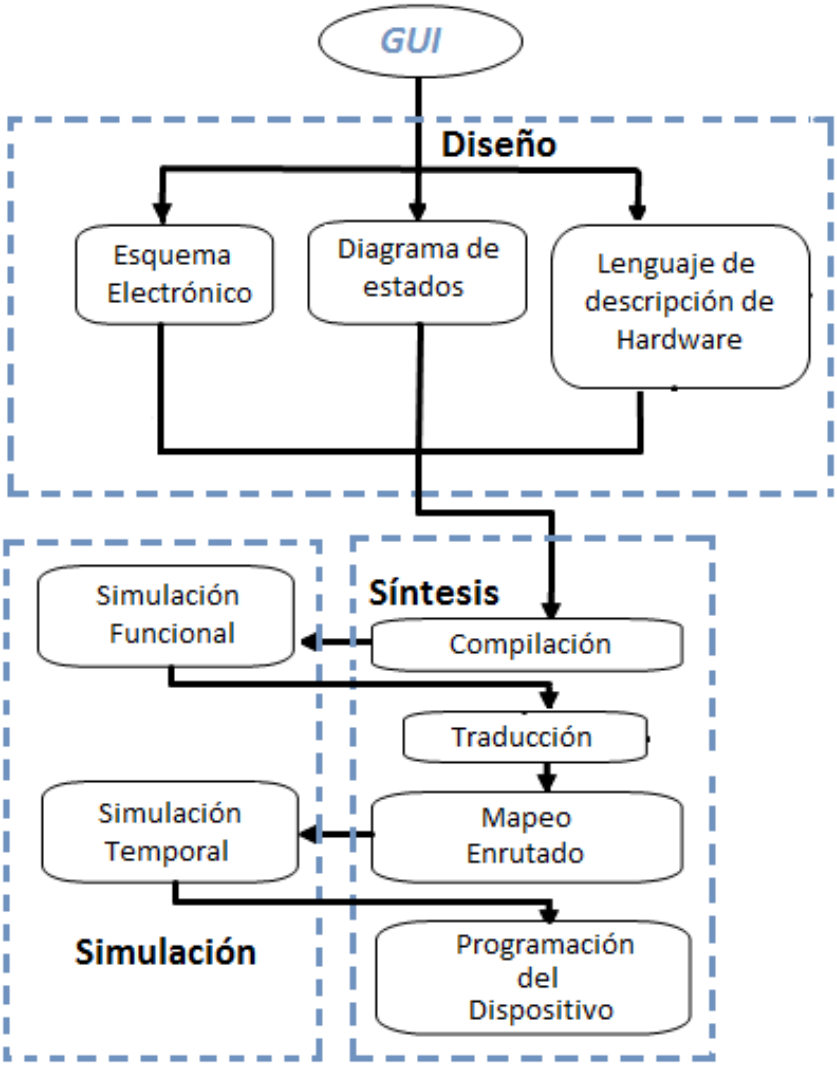

Figura. 3.5 Pasos en el desarrollo de aplicaciones con FPGAs

ISE posee una interfaz gráfica de usuario GUI (*Graphic User Interface*) que se denomina *Project Navigator*. Para el inicio de un nuevo proyecto, se tiene la etapa de diseño. En esta etapa lo que se hace es seleccionar el tipo de descripción de diseño. Los diseños se pueden introducir en diferentes formatos. Los más utilizados son: los esquemáticos, los diagramas de estados y las descripciones hardware en VHDL o *Verilog*. Posteriormente en la síntesis se compila el diseño y se crea la *netlist*, una vez compilado, se puede simular a nivel funcional. Después de esta simulación, mediante las herramientas de implementación como la traducción, el mapeo y el ruteo, se permite la especificación de restricciones o indicaciones, a través del archivo **.cfg*, para realizar una implementación

óptima en el FPGA. Así se puede realizar una simulación temporal, de esta forma se puede estimar cómo se comportará el sistema en el FPGA. La simulación a nivel funcional no tiene en cuenta los retardos provocados por el hardware y a nivel temporal simula el diseño teniendo en cuenta la configuración en hardware. Finalmente se crea el fichero de programación **.bit* y se descarga el diseño sobre el dispositivo.

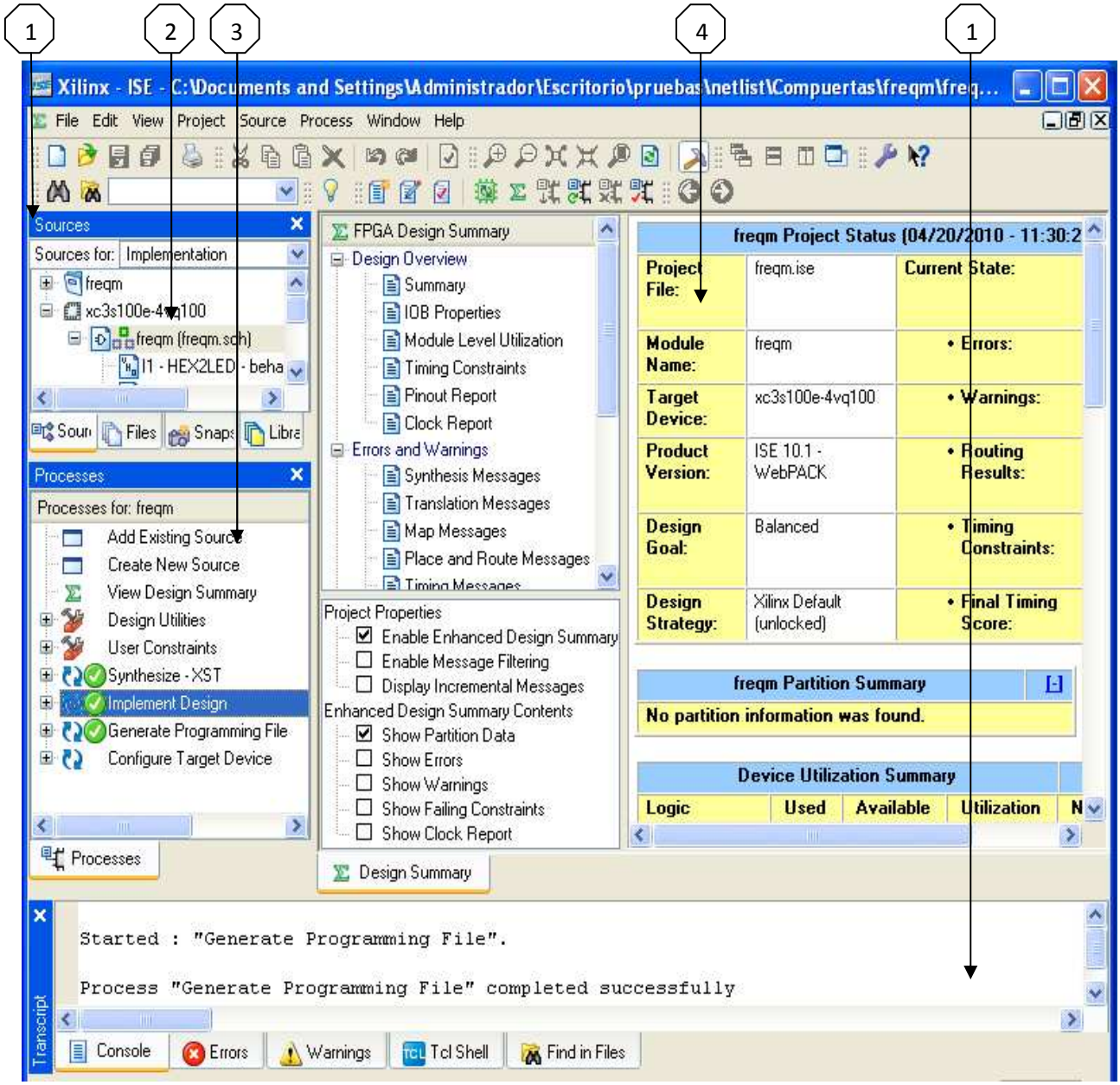

Figura. 3.6 Interfaz de Project Navigator

En la Figura 3.6 se muestra la interfaz de *Project Navigator*. En el punto número uno se puede observar la barra de herramientas. En el punto número dos se puede apreciar la ventana de fuentes, en la que se ve los archivos fuente que constituyen el proyecto. El

punto 3, se indica la ventana de procesos. Estos procesos pueden ser, síntesis, simulación, implementación entre otros. En el punto 4, se representa el área de trabajo, esta área puede ser de tipo esquemático y también VHDL, dependiendo del tipo de descripción del proyecto seleccionado. Finalmente el punto 5, es la ventana de transcripción, en donde se puede ver el estado de las operaciones ya procesadas y las que se están procesando.

El FPGA se configura mediante la programación de la memoria flash. Para esto se debe utilizar la herramienta IMPACT, que se encuentra en *Project Navigator*. El archivo que se grabe en la tarjeta como se mencionó anteriormente será el binario generado. La interfaz de IMPACT, mostrada en la Figura 3.7, es muy amigable y es la que se utiliza en este proyecto.

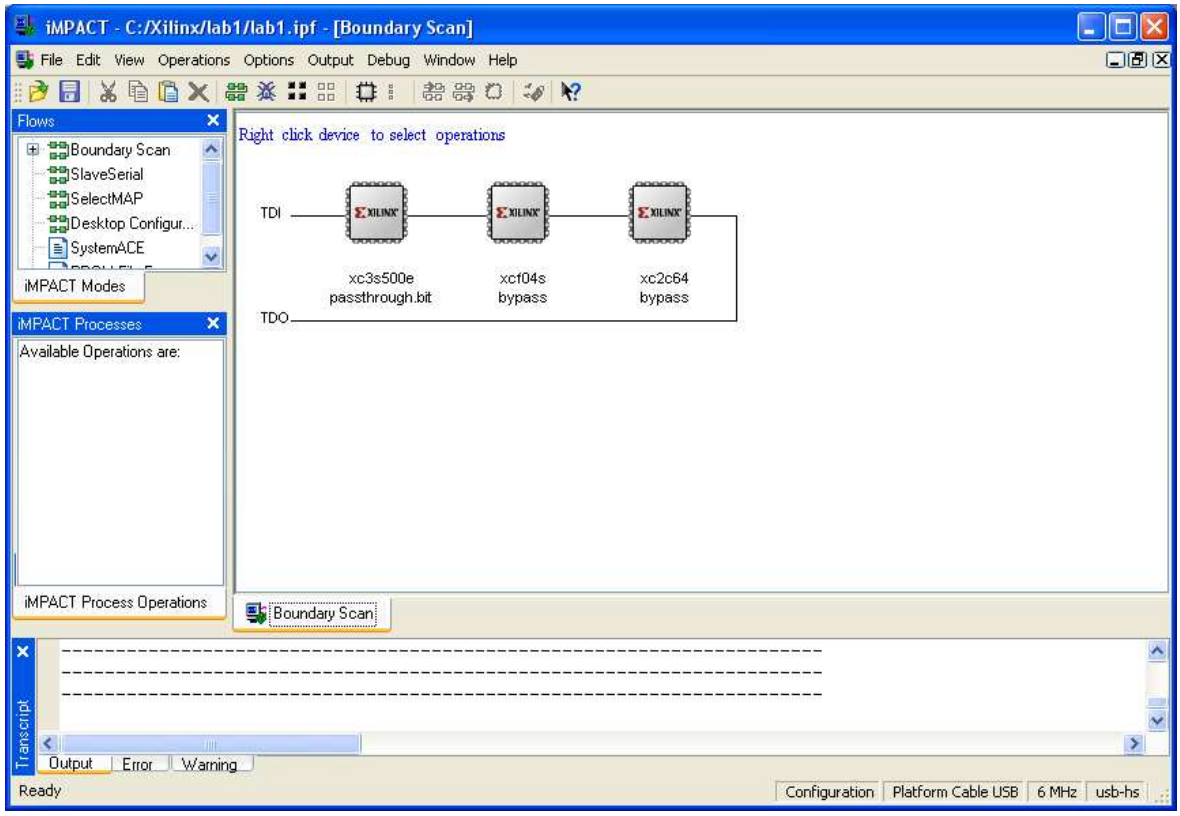

Figura. 3.7 Herramienta IMPACT de Xilinx

3.3 Descripción de la tarjeta de desarrollo Spartan 3E.

El dispositivo FPGA incluido en esta tarjeta de desarrollo es el XC3S500E-FG320 de Xilinx. Otros dispositivos de Xilinx incluídos son: CoolRunner™-II CPLD (XC2C64A-5VQ44C) y Plataforma Flash (XCF04S-VO20C). A continuación se mencionan las características más importantes de esta tarjeta.

Contiene 500 K 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 (DCM) y hasta 232 señales de E/S. En cuanto a memoria se encuentran las siguientes: Memoria Parallel Flash de 128 Mbit para aplicaciones, DDR SDRAM de 64 MByte, Memoria Flash de 4 Mbit para configuración y una Memoria Flash 16 Mbits con interfaz SPI (*Serial Peripheral Interface*).

Figura 3.8 Tarjeta de Desarrollo Spartan 3E

Posee dispositivos de interfaz análogos, un ADC de 4 canales con resolución de 14 bits, un DAC de 4 canales, con resolución de 12 bits y un amplificador de señal. Entre conectores e interfaces, posee un puerto Ethernet 10/100 PHY, un puerto JTAG USB para programación, dos puertos seriales RS-232 de 9 pines, un puerto PS/2 para teclado o mouese, un puerto VGA, 8 salidas individuales LED, cuatro slider switches, cuatro botones "push-button", un puerto de expansión de 100 pines, tres conectores de expansión de 6 pines y una pantalla LCD de 16 caracteres por 2-líneas. Finalmente esta tarjeta posee un reloj de 50 MHz.

Las interfaces que se utilizan para la implementación del transmisor son: El conversor digital-análogo y el puerto SPI. Los mismos que serán configurados en ISE Project Navigator mediante programación en VHDL e introducidos al diseño en *System Generator* con la ayuda del bloque black box que posee a librería de Xilinx para Simulink.

En la Figura 3.8, se puede observar la tarjeta de desarrollo FPGA Spartan 3E y la ubicación del DAC y del puerto SPI.

3.3.1 Descripción del DAC LT2624

El DAC que está incluido en la tarjeta es el LT2624 de *Linear Technology* [21]. El FPGA utiliza la interface SPI para comunicar los valores digitales a cada uno de los cuatro canales del DAC, como se observa en la Figura 3.9.

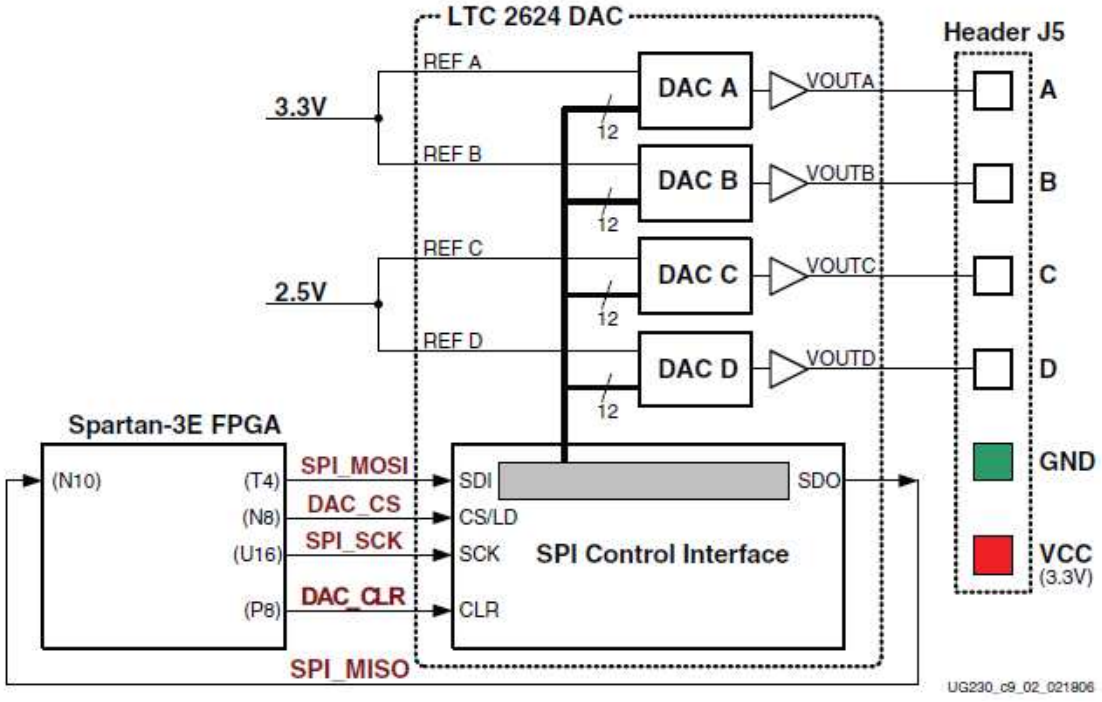

Figura. 3.9 Esquema de conexión para el DAC

Esta comunicación se la hace mediante las siguientes señales: la señal de reloj (SPI_SCK), la transmisión serial de datos hacia el DAC a través de SPI_MOSI, al mismo tiempo estos datos se transmiten serialmente al FPGA a través de SPI_MISO, Chip-Select activado en bajo a través de DAC_CS y una entrada asincrónica de reset (DAC_CLR). Estas señales se explican detalladamente en la Tabla 3.1.

Tabla. 3.1 Señales para manejo del DAC

FPGA Signal	Pin	Dirección	Descripción
SPI_MOSI	T4	FPGA ==> DAC	Datos Seriales. Master Output(FPGA), Slave Input(DAC)
DAC_CS	N8	FPGA ==> DAC	Chip-Select activado en bajo. La conversión inicia cuando está en alto.
SPI_SCK	U16	FPGA ==> DAC	Reloj
DAC_CLR	P8	FPGA ==> DAC	Entrada de reset asíncrona activada en bajo.
SPI_MISO	N10	FPGA <== DAC	Datos Seriales. Master Input(FPGA), Slave Output(DAC)

Las señales del bus SPI, están compartidas con varios dispositivos de la tarjeta, es por esto que para configurar el DAC hay algunos dispositivos que deben ser deshabilitados, estos dispositivos se detallan en la Tabla 3.2.

Tabla. 3.2 Señales deshabilitadas en el bus SPI

Señal	Dispositivo Deshabilitado	Valor de Deshabilitación
SPI_SS_B SPI	Serial Flash	1
AMP_CS	Pre-amplificador programable	1
AD_CONV	ADC	0
SF_CE0	StrataFlash Parallel Flash PROM	1
FPGA_INIT_B	Platform Flash PROM	1

El DAC LT2624, acepta dos protocolos para su configuración: 24 y 32 bits. En la Figura 3.10, se muestra el protocolo de 24 bits que es el que se usa en este proyecto. Dentro del conversor digital análogo, el bus SPI está formado por un registro de desplazamiento de 24 bits. Cada palabra de 24 bits, consiste de un comando, una dirección seguida por los datos. Los datos son enviados desde el bit más significativo. Es decir primero se envían los 4 bits de “COMMAND”, seguido de los 4 bits de ‘ADDRESS”, estos bits especifican el canal del DAC por el que se va a obtener la señal.

Las posibles asignaciones de (C3-C0) y (A3-A0), se muestran en la Tabla 3.3.

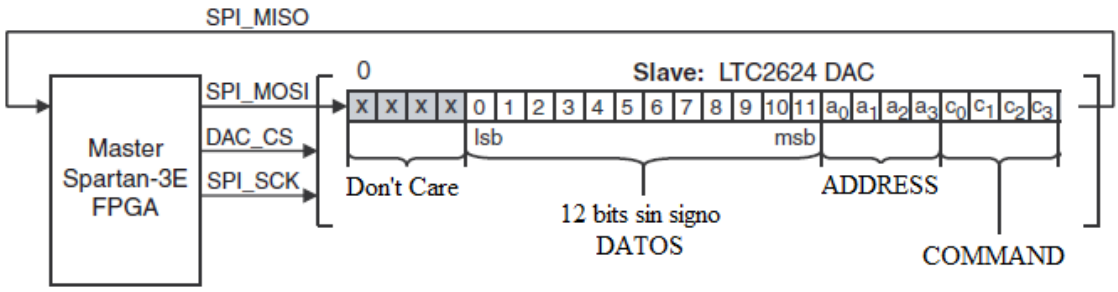

Figura. 3.10 Protocolo de 24 bits para el DAC.

El comando más usado es COMMAND[3,0] = ”0011”, que actualiza inmediatamente la salida del DAC con el valor de los datos. Los códigos no mostrados en la tabla son reservados y no deben ser usados. Finalmente se envían los 12 bits de datos sin signo[22].

Tabla. 3.3 Asignación de valores para COMMAND y ADDRESS

ADDRESS						**COMMAND**				
a_3	**a_2**	**a_1**	**a_0**	**OUT**		**c_3**	**c_2**	**c_1**	**c_0**	**OPERACIÓN**
0	**0**	**0**	**0**	DAC A		**0**	**0**	**0**	**0**	Escribir en un registro de entrada n
0	**0**	**0**	**1**	DAC B		**0**	**0**	**0**	**1**	Actualizar el registro n
0	**0**	**1**	**0**	DAC C		**0**	**0**	**1**	**0**	Escribir en un registro de entrada n, Actualizar todo
0	**0**	**1**	**1**	DAC D		**0**	**0**	**1**	**1**	Escribir y actualizar n
1	**1**	**1**	**1**	TODAS		**0**	**1**	**0**	**0**	Apagar n
						1	**1**	**1**	**1**	No hay operación

CAPÍTULO 4

SIMULACIÓN E IMPLEMENTACIÓN DEL TRANSMISOR

Para la simulación e implementación se debe instalar lo siguiente: ISE, *System Generator y* MATLAB. La tarjeta de desarrollo que se utiliza en este proyecto es la Spartan 3E.

4.1 Diseño del transmisor

Para el diseño del transmisor con modulación BPSK y del transmisor con modulación QPSK, se parte de los esquemas representados en las figuras 2.8 y 2.10 respectivamente. Sin embargo hay algunos cambios por facilidad de diseño y por el hecho de que en este proyecto se utiliza codificación convolucional en el canal de transmisión.

4.1.1 Estructura del transmisor

En la Figura 4.1 se muestra el diagrama de bloques de la estructura del transmisor con modulación BPSK.

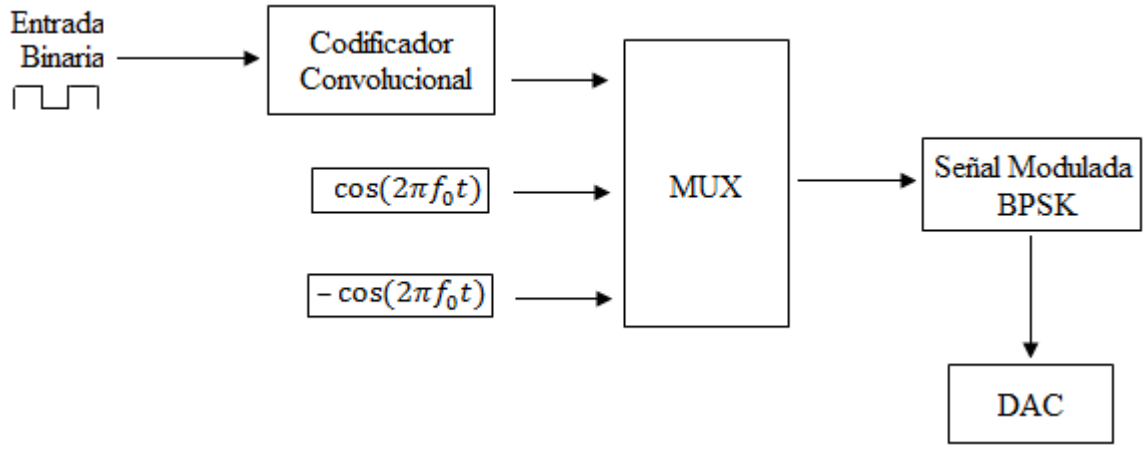

Figura. 4.1 Estructura del transmisor con modulación BPSK

En la Figura 4.2 se muestra el diagrama de bloques de la estructura del transmisor con modulación QPSK.

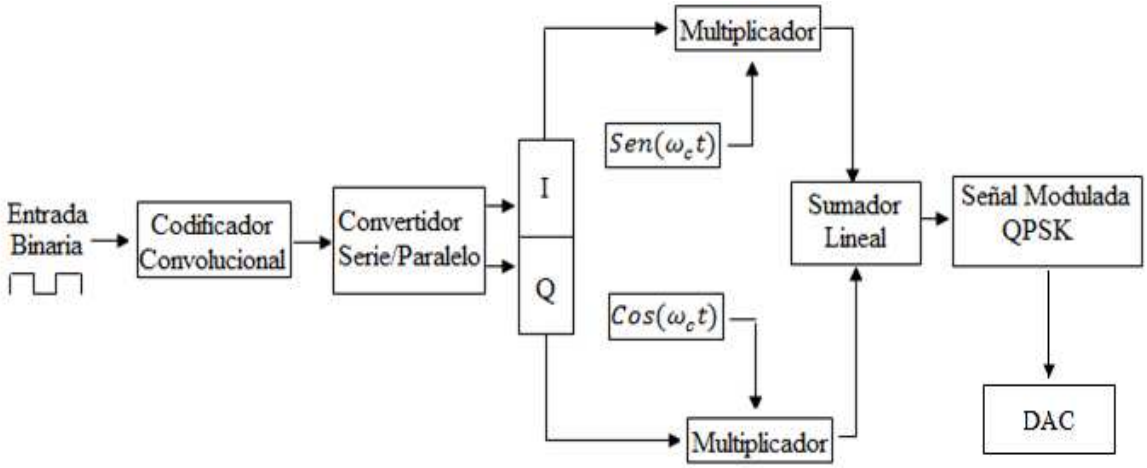

Figura. 4.2 Estructura del transmisor con modulación QPSK

4.3 Implementación en *System Generator*

Para la implementación se ingresa a *System Generator* y automáticamente el entorno de MATLAB-Simulink se abre. Como ya se explicó en el capítulo anterior, en la librería de Simulink, se encuentra el Xilinx *blockset*. En este bloque están todas las herramientas necesarias para la implementación de los transmisores.

En cada uno de los bloques de *System Generator*, se deben ajustar ciertos parámetros como frecuencia, número de bits, nombre de pines de la tarjeta, entre otros. Para esto, es muy importante que se conozca muy bien la estructura así como las especificaciones de la tarjeta que se está utilizando, de esta manera se evitará cometer errores. Una vez que se han configurado los bloques es posible realizar una simulación exitosa.

En el diseño se han utilizado varios bloques, a continuación se nombran los utilizados específicamente para lograr la generación de señales y la codificación. En el trasmisor BPSK y en el transmisor QPSK, las entradas binarias y las portadoras, se generan por medio de bloques de Xilinx *blockset* de Simulink, de esta manera no hace falta ingresar señales externas al FPGA. Para generar las portadoras el bloque utilizado es el DDS Compiler, para la entrada binaria se utiliza el bloque LSFR en el caso del transmisor con modulación BPSK y una memoria ROM para el transmisor con modulación QPSK, estos

bloques generan una señal cuadrada la cual es codificada por medio de un bloque llamado *Convolutional Encoder*. Estos bloques se observan en la Figura 4.3.

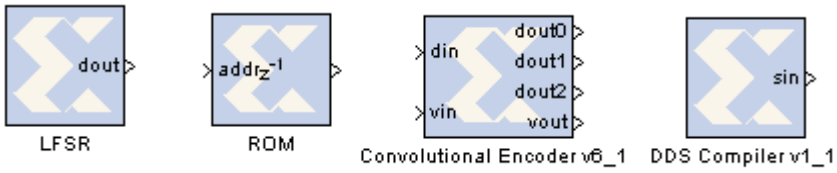

Figura. 4.3 Bloques utilizados para la generación y codificación de señales.

En el bloque LSFR no se cambian los parámetros, se trabaja con los que están ya establecidos. Este bloque genera la misma entrada binaria siempre ya que es un registro de desplazamiento determinístico entonces la secuencia de valores producidos está completamente determinada por el estado actual o el estado anterior. La secuencia tiene un periodo de repetición, es decir que la secuencia vuelve a generarse y se repite indefinidamente. Lo más adecuado es generar una entrada binaria aleatoria, por esta razón también se utiliza el bloque ROM. Este bloque puede ser utilizado en los dos transmisores, pero para poder conocer el funcionamiento de la mayoría de los bloques de la librería de xilinx se utiliza un bloque diferente en cada transmisor. El bloque ROM tiene una entrada la cual es la dirección en que se almacenan los datos. Esta entrada debe estar manejada por un número entero y sin signo. En el diseño se maneja por medio de un contador. En la salida de este bloque también se pueden ajustar varios parámetros, como el tipo de palabra y número de bits. En este caso la salida es de un bit y sin signo, ya que esta salida se conecta a la entrada del codificador convolucional, y este el tipo de entrada que este bloque acepta.

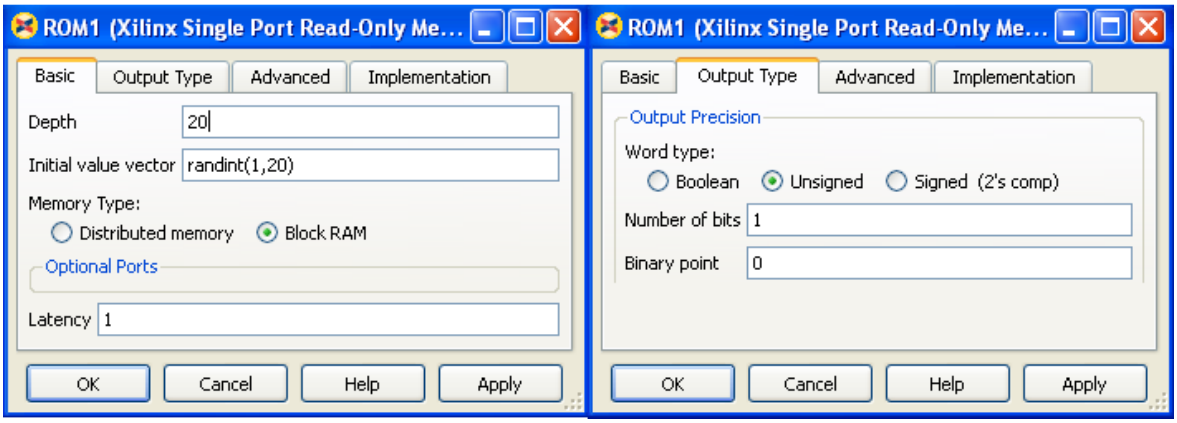

Figura. 4.4 Configuración para la generación de una señal binaria aleatoria.

Otro parámetro muy importante es el valor inicial del vector, aquí se puede generar una señal utilizando las funciones de Matlab. Para la entrada binaria la función es *randint()*, la cual genera una señal de unos y ceros aleatoriamente. En la Figura 4.4 se muestra la configuración de este bloque.

Los bloques DDS Compiler se debe establecer una frecuencia muy baja debido a que el DAC LT2624 que se encuentra en la tarjeta FPGA, no soporta frecuencias en el orden de los megas. Además este DAC tiene 4 canales de 12 bits, por esta razón la señal generada será de 12 bits. En la Figura 4.5, se muestra como configurar dichos parámetros en la portadora. Para este diseño se escoge la salida de tipo seno y en el caso de la portadora desfasada la salida será una onda sinusoidal negativa

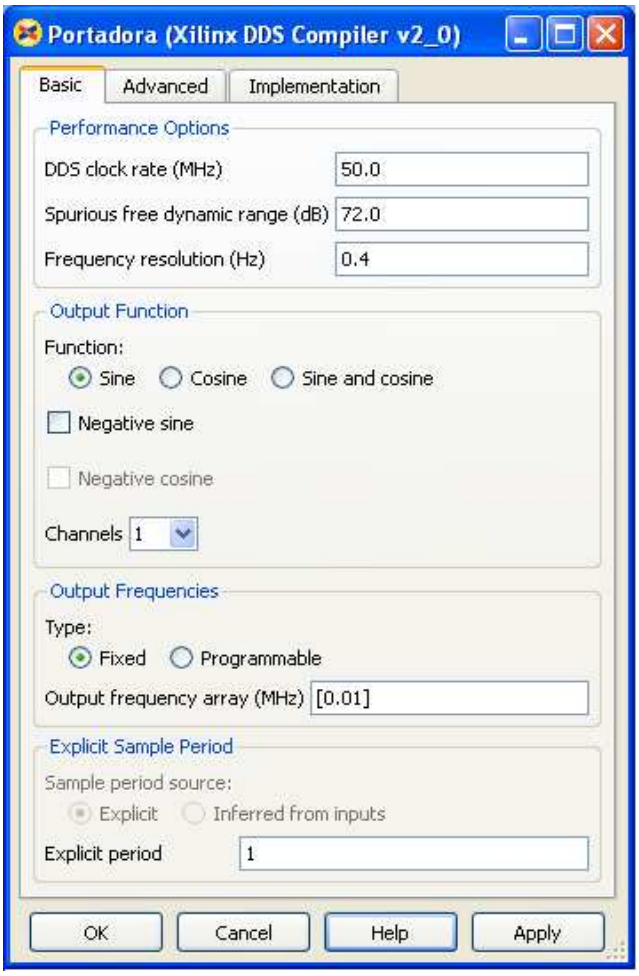

Figura. 4.5 Parámetros del bloque DDS Compiler.

La frecuencia de salida es de 10 kHz. Los mismos valores se utilizan en la portadora desfasada. Para el transmisor con modulación QPSK, el único cambio será en la señal desfasada la cual es una señal coseno.

Para la codificación del mensaje, como ya se había mencionado en capítulos anteriores, se utiliza codificación convolucional. El bloque en *System Generator* para la codificación es *Convolutional Encoder V6_1*. Los parámetros que se necesitan configurar en este bloque para el transmisor son la longitud restricción de codificación y la matriz del código convolucional, la cual especifica la tasa de código. En este proyecto la longitud de restricción *K* es igual a 3 y la tasa de código es de ½. En el puerto din, se ingresan los valores que serán codificados, es decir el mensaje de entrada. Para que este bloque funcione correctamente, se necesita habilitar el puerto *vin*, el cual indica que los valores presentados en el puerto *din* son válidos. Solo los valores válidos son codificados. Los puertos *dout* son las salidas del mensaje codificado. Para unir las bits de estas salidas serialmente, se emplea el bloque *Time Division Multiplexer*, con dos entradas. La configuración del codificador queda como se ve en la Figura 4.6. La misma configuración es utilizada en el transmisor con modulación QPSK.

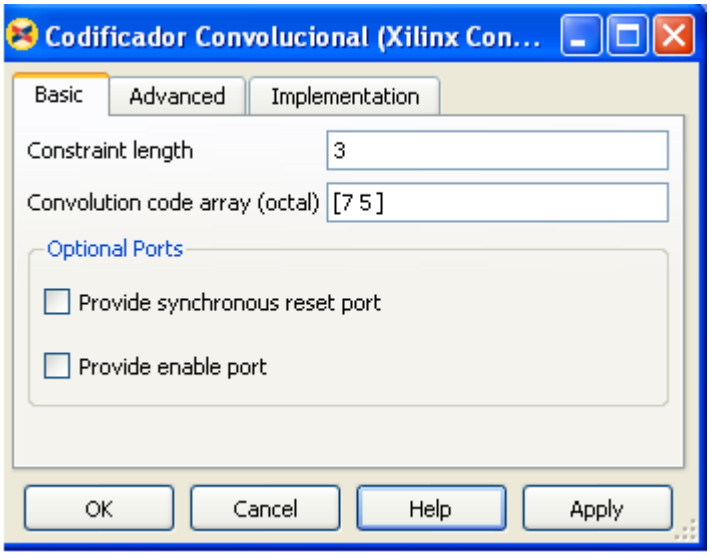

Figura. 4.6 Parámetros del codificador convolucional.

Para el proceso de modulación existen varias diferencias entre los dos transmisores. En el transmisor BPSK Para lograr que la fase cambie dependiendo del estado del mensaje, se emplea un multiplexor, mostrado en la Figura 4.7. Es decir si se tiene un ‘0’, deja pasar

la portadora sin desfase y cuando se tiene un '1', deja pasar la portadora desfasada 180º. De esta manera se logra el cambio de fase, en cada cambio de estado del mensaje.

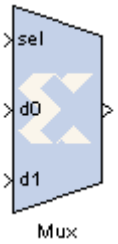

Figura. 4.7 Multiplexor de Xilinx *Blockset*

Para el transmisor QPSK, se utilizan muchos más bloques, debido a que es más complejo que el transmisor BPSK. Un modulador QPSK tiene la características particular de necesitar una interfaz serie-paralelo para la separación de los canales I y Q (fase y cuadratura).

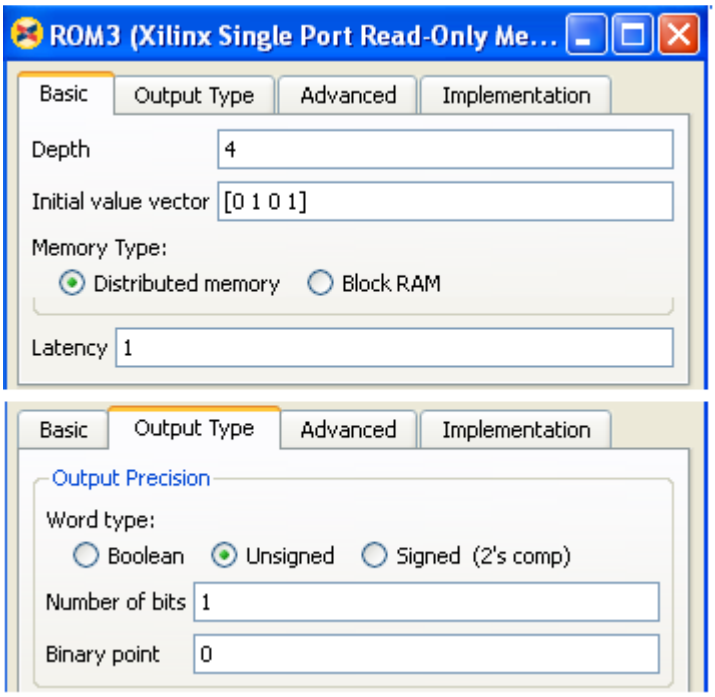

Figura. 4.8 Bloque ROM

Para lograr esto el mensaje codificado, tiene que pasar por un convertidor, y los bits que se tienen a la salida son los llamados pares e impares. Para esto se usó una señal de reloj que se genera por medio del bloque ROM. La configuración para lograr esta señal se muestra en la Figura 4.8. En esta señal el ancho de pulso es igual al tiempo de bit de la señal a modular.

Para obtener el canal I lo que se hace es multiplicar esta señal por la señal binaria La multiplicación se la hace mediante una compuerta AND. De esta señal se toman dos bits consecutivos y pasan a través de una compuerta XOR. Para tomar dos bits consecutivos a lo largo de toda la señal es necesario emplear bloque de retardo para recorrer de bit en bit. Para el canal Q simplemente se invierte la amplitud del la señal de reloj y se realiza el mismo proceso. Este proceso se muestra en la Figura 4.9.

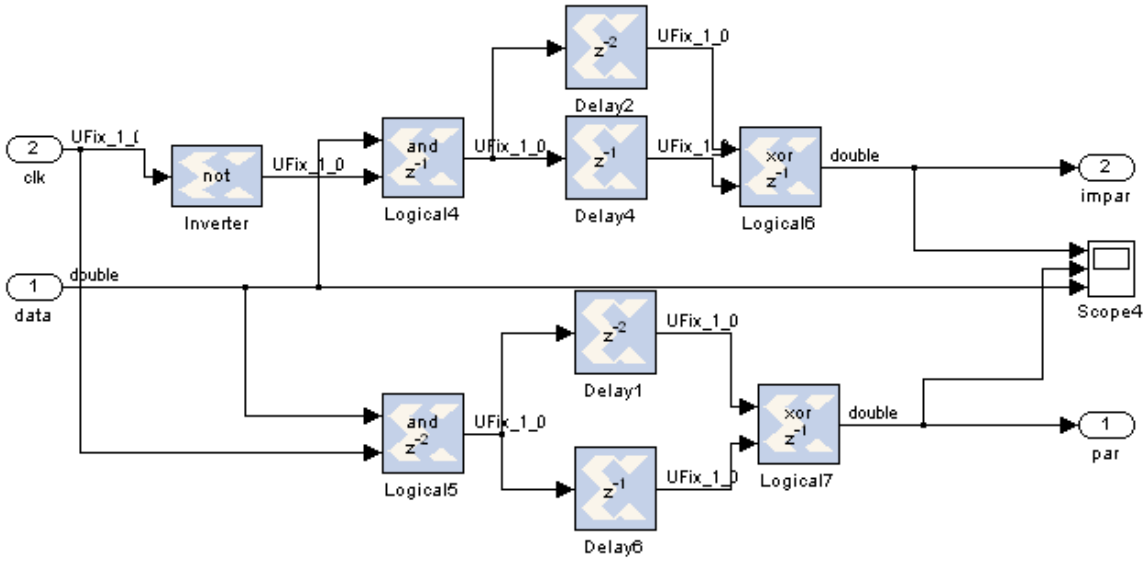

Figura. 4.9 Proceso para recuperar el tiempo de bit.

Para entender de mejor manera el proceso anterior se tiene el siguiente ejemplo. Se considera la entrada binaria 11101011100111, la señal de reloj es 10101010101010 y la misma señal de reloj invirtiendo la amplitud sería 01010101010101. Entonces al realizar la operación lógica AND entre las dos primeras señales teniendo en cuenta la Tabla 4.1, se tiene como resultado 10101010100010. El resultado de la entrada binaria y la señal de reloj invertida es 01000001000101.

Tabla. 4.1 Tabla de verdad de la compuerta AND

AND		
A	**B**	**OUT**
0	0	0
0	1	0
1	0	0
1	1	0

Una vez que se tiene esos resultados se toma los bits de dos en dos y se realiza la operación lógica XOR como se indica en la Tabla 4.2.

Tabla. 4.2 Tabla de verdad de la compuerta XOR

XOR		
A	**B**	**OUT**
0	0	0
0	1	1
1	0	1
1	1	0

Esta operación se observa en la Tabla 4.3.

Tabla. 4.3 Comprobación del proceso de separación de bits

Entrada binara por la señal de reloj														
	1	0	1	0	1	0	1	0	1	0	0	0	1	0
Bits Pares	XOR 1		XOR 1		XOR 1		XOR 1		XOR 1		XOR 0		XOR 1	
Entrada binara por la señal de reloj invertida														
	0	1	0	0	0	0	0	1	0	0	0	1	0	1
Bits Impares	XOR 1		XOR 0		XOR 0		XOR 1		XOR 0		XOR 1		XOR 1	

Entonces si se combina los bits pares impares se obtiene la entrada binaria 11101011100111. De esta manera se verifica que el proceso sea el adecuado.

El siguiente paso es la conversión a un código de línea ideal para la transmisión, el código polar sin retorno a cero. Esto se logra fácilmente como se muestra en el esquema de la Figura 4.10.

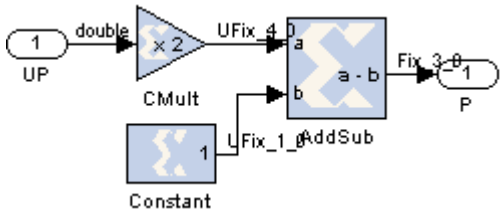

Figura. 4.10 Código polar sin retorno a cero

El siguiente paso para lograr la modulación es multiplicar los canales I y Q por las portadoras y finalmente sumar los dos canales en cuadratura para la transmisión de la señal. Esto con la ayuda de los bloques Mult y AddSub, mostrados en la Figura 4.11.

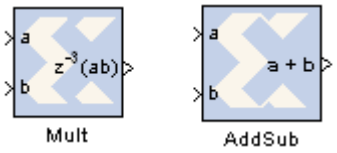

Figura. 4.11 Bloques *Mult* y *AddSub* de Xilinx *Blockset.*

Para poder extraer las señales por medio del FPGA es necesario utilizar el DAC. Su configuración también se la hará con la ayuda de *System Generator*, aunque su programación se la hace en ISE, en lenguaje VHDL como ya se menciono anteriormente.

El bloque del DAC es un *black box*, este bloque proporciona el mayor grado de flexibilidad a costa de la complejidad del diseño. Al utilizar este bloque se puede interactuar cualquier procesador HDL con un diseño de *System Generator*. Todas las entradas y salidas que estén citadas en el procesador, son expuestas en el diagrama de *System Generator*, de esta manera se pueden conectar con otros bloques y así lograra la interactividad [20]. Este bloque se encuentra en los elementos básicos de la librería de Xilinx en Simulink.

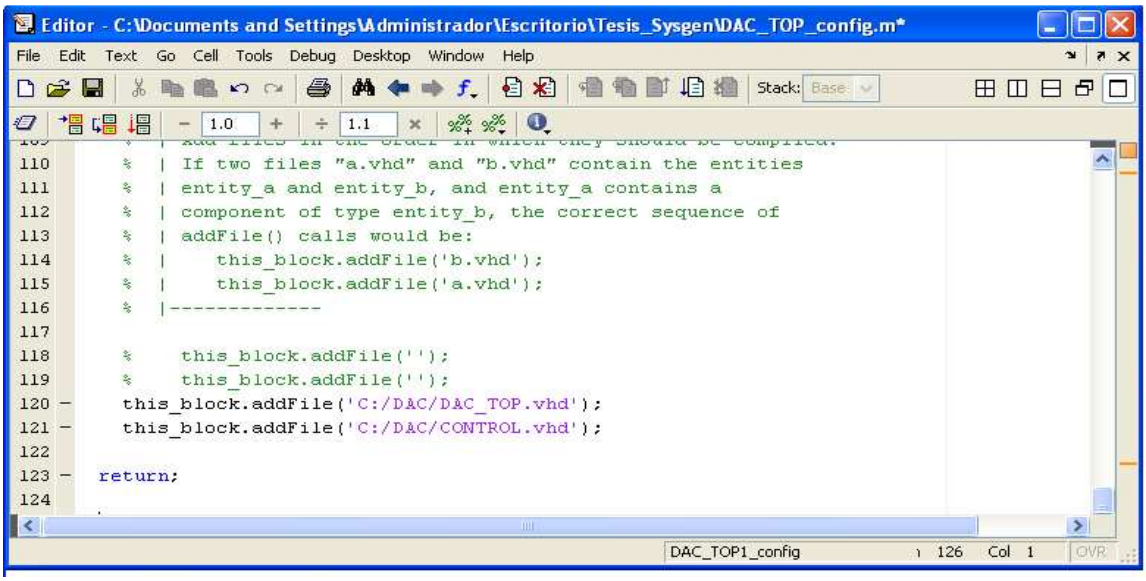

Figura. 4.12 Archivo *m-file* creado en el *black box*

Una vez que se introduzca este bloque en el diseño de *System Generator*, se debe añadir en este caso el fichero TOP.vhd para el control del DAC. Cuando se añade el archivo, se crea automáticamente un archivo del tipo mfile. Al final de este archivo se debe añadir el fichero CONTROL.vhd como se observa en la Figura 4.12, ya que para el control del DAC la programación en vhdl se divide en dos entidades TOP.vhd y CONTROL.vhd (Ver Anexo A1 y A2). En el diseño tanto del transmisor con modulación BPSK como del transmisor con modulación QPSK, los datos de entrada al bloque *black box*, que en este caso sería utilizado como el control del DAC de la tarjeta, son enviados mediante bloques de *System Generator*. Las demás entradas y salidas del mismo, están conectadas a los bloques *Gateway In* y *Gateway Out* respectivamente. En estos bloques se debe configurar los pines a los que irán conectados físicamente en la tarjeta. En la Figura 4.13, se muestra como queda el *black box* una vez que se hayan añadido los archivos necesarios, y colocado las entradas y salidas.

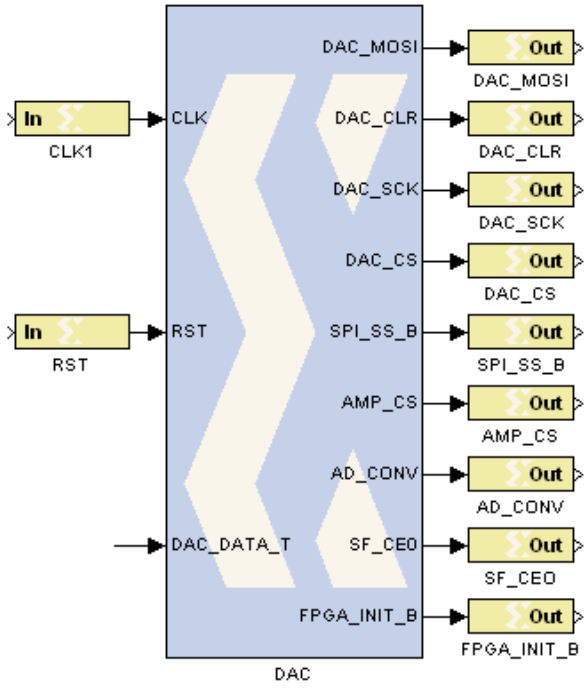

Figura. 4.13 DAC LT2624 configurado en un black box.

El DAC LT2624 sólo acepta datos sin signo, por esta razón, es necesaria la etapa de inversión de bits mencionada anteriormente ya que las señales que se genera con el DDS *Compiler* son con signo. Esta etapa se muestra en la Figura 4.14. En esta etapa, se utiliza el bloque *Convert*, este bloque sirve para cambiar un número determinado de bits, en un tipo

aritmético deseado. En este diseño se lo utiliza simplemente para asegurarse de que todos los bits que van a ser invertidos sean de un mismo tipo, en este caso con signo.

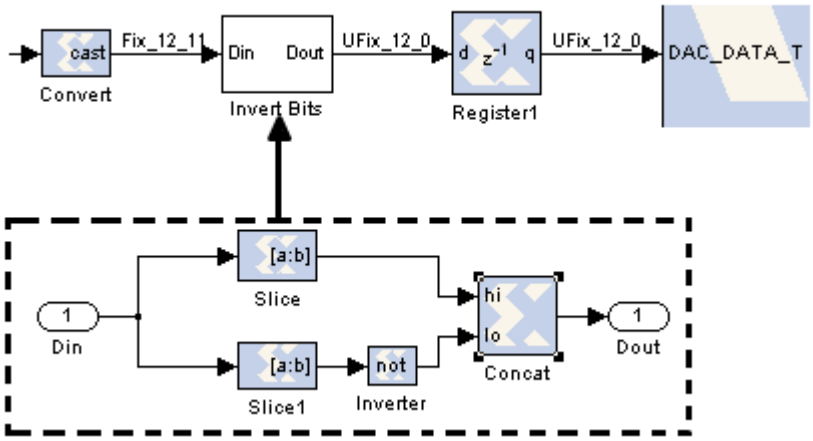

Figura. 4.14 Etapa de Inversión de Bits

En este bloque se ajusta el parámetro de número de bits en 12. Para la inversión de bits lo que se debe hacer es dejar pasar el primer bit y negar todos los siguientes, para lograr esto se utilizan dos bloques *Slice*, un negador, y el bloque *Concat*.

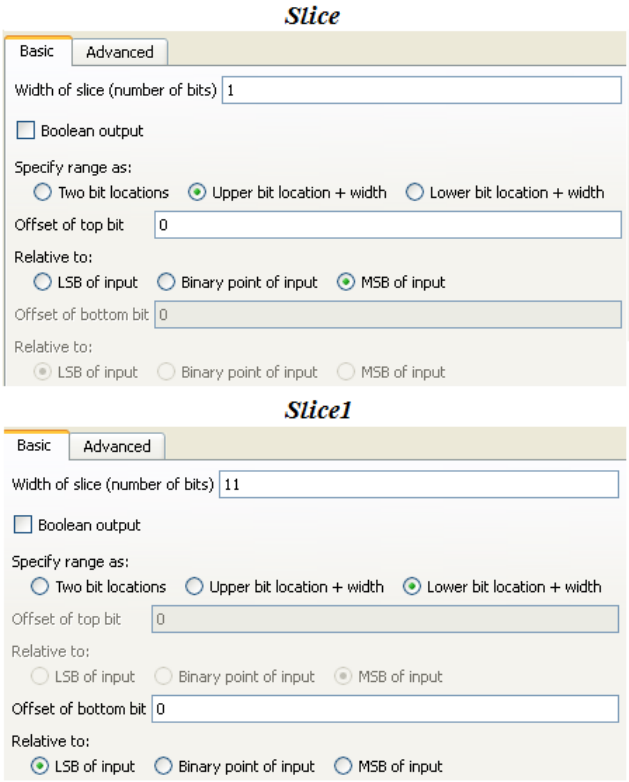

Figura. 4.15 Configuración de parámetros para el bloque *Slice*

El bloque *Slice*, es utilizado para separar los bits y dejar pasar sólo los que se desee, entonces la salida de este bloque será completamente diferente a la entrada original. Uno

de estos bloques deja pasar sólo el primer bit más significativo y el otro deja pasar los 11 bits restantes. Después del bloque *Slice1*, se coloca un bloque *not*, para negar los 11 bits y así lograr la inversión. La configuración de los parámetros para estos bloques es mostrada en la Figura 4.15. A la salida de estos bloques, es necesario el bloque *Concat*, este bloque concatena todos los bits, es decir vuelve a unir todos los bits, indicando el bit más significativo. En la salida de este bloque se tiene la palabra de 12 bits, sin signo .Al final de esta etapa se coloca el bloque *Register*, este bloque actúa como un *flip-flop* tipo D. Lo que hace es almacenar los datos y dejarlos salir con un retardo entre cada uno.

Una vez que todo el diseño esta implementado en el software, se pueden colocar bloques *Gateway Out* para poder visualizar el mensaje original y el mensaje codificado directamente desde la tarjeta hacia el osciloscopio, también. La asignación de pines de cada uno de los bloques *Gateway In* y *Gateway Out*, utilizados en los dos diseños, se encuentra en la Tabla 4.4. Solo para el caso del transmisor con modulación QPSK, se asigna los pines para la visualización de bits pares e impares.

Tabla. 4.4 Asignación de pines

ASIGNACIÓN DE PINES		
TIPO	**NOMBRE**	**PIN**
IN	CLK1	C9
IN	RST	K17
OUT	Mensaje	D5
OUT	MensajeCod	C5
OUT	Bits_Impares	F8
OUT	Bits_Pares	E8
OUT	DAC_MOSI	T4
OUT	DAC_CLR	P8
OUT	DAC_SCK	U16
OUT	DAC_CS	N8
OUT	SPI_SS_B	U3
OUT	AMP_CS	N7
OUT	AD_CONV	P11
OUT	SF_CEO	D16
OUT	FPGA_INIT_B	T3

En la Figura 4.16 y 4.17 se muestra el esquema final del transmisor BPSK y del transmisor QPSK repsectivamente, implementados para la simulación con el software *System Generator*.

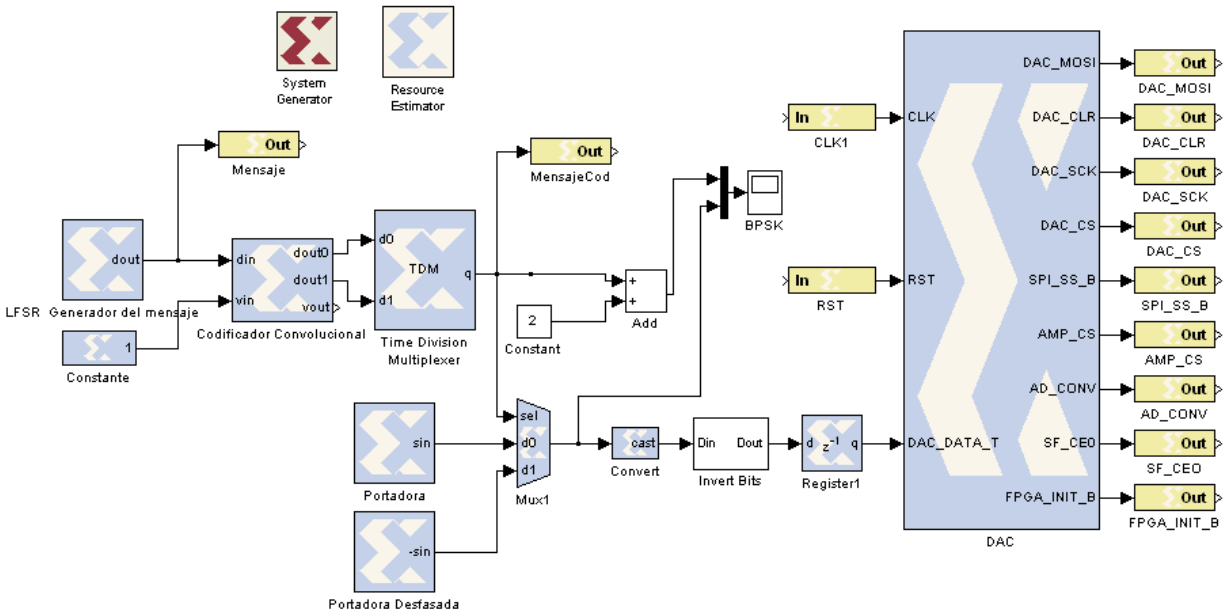

Figura. 4.16 Esquema del transmisor BPSK en System Generator

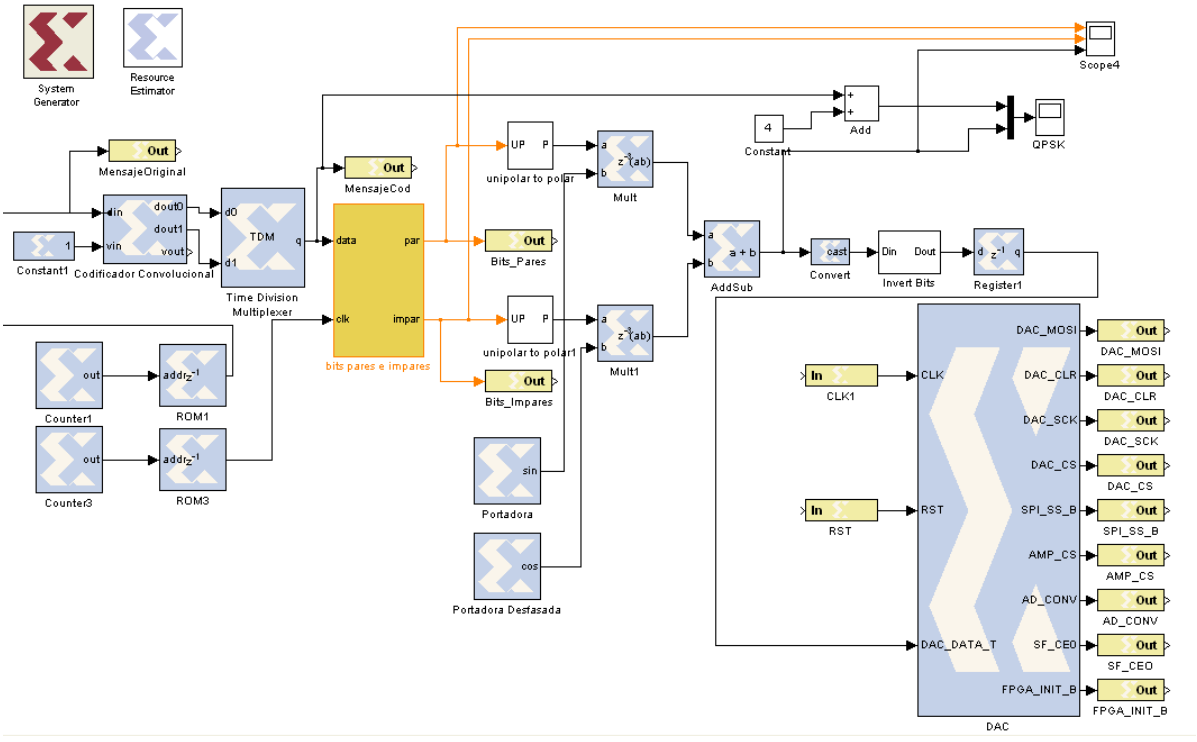

Figura. 4.17 Esquema del transmisor QPSK en *System Generator*

Finalmente se debe configurar el bloque de *System Generator*. Este bloque se configura específicamente para hardware. Los parámetros que se configuran son: El tipo de compilación, la tarjeta que se va a utilizar, el directorio en donde se guarda el archivo, la herramienta de síntesis, el lenguaje de descripción de hardware, el periodo del reloj del FPGA y el pin de reloj del FPGA. Estos parámetros se observan en la Figura 4.18 y son iguales en el transmisor con modulación QPSK.

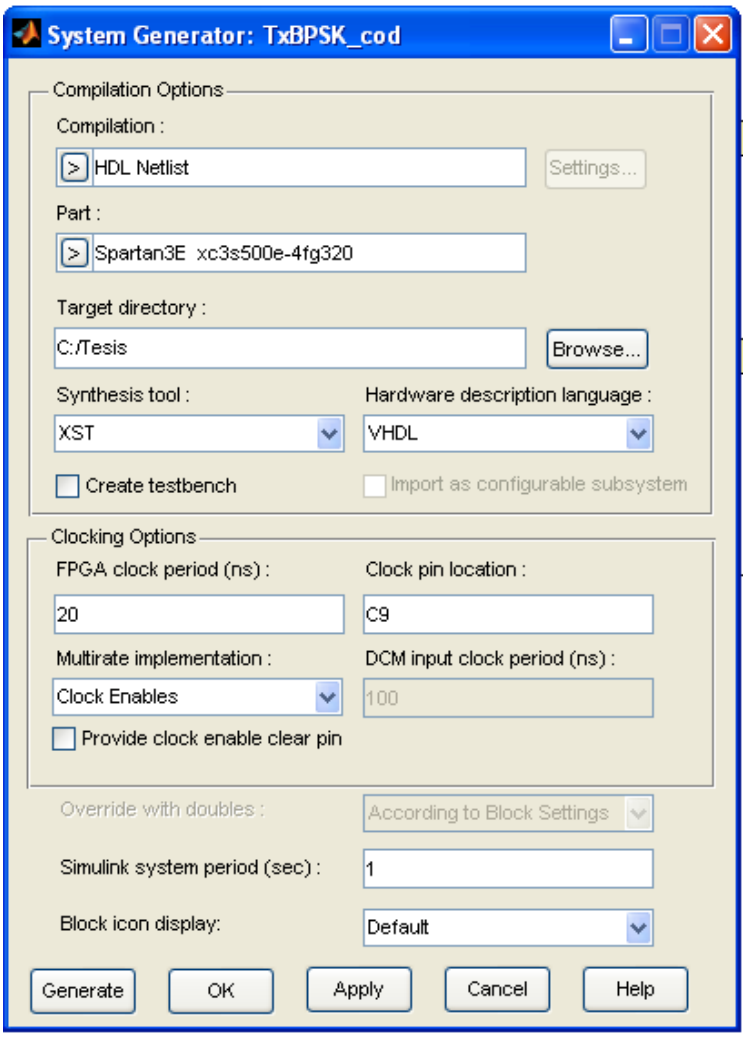

Figura. 4.18 Parámetros del bloque *System Generator*

En el tipo de compilación se escoge HDL netlist, a pesar de que se podría escoger *bitstream* para generar directamente el archivo .bit para la programación del FPGA. La razón por la cual se escoge este tipo de compilación se debe a que se deben hacer cambios en *Project Navigator* para unir el reloj del *black box*, con el reloj de todo el sistema de *System Generator*, ya que no es posible asignar el mismo pin dos veces en diferentes

entradas. La tarjeta utilizada en este proyecto es la Spartan 3E xc3s500e-4fg320. Entonces en el parámetro "*part*", se escoge esta tarjeta.

La herramienta de síntesis será XST, y el lenguaje de compilación de hardware puede ser *Verilog* o VHDL, en este caso se escoge VHDL por ser el lenguaje más conocido, así los cambios que se deban hacer en el código generado se realizaran en este lenguaje. Para las opciones de reloj, se asigna un periodo de 20 ns y el pin de reloj de la tarjeta utilizada es 'C9'. El último paso es generar el archivo .vhd, dando click en el botón *Generate.*

4.3 Simulación en *System Generator*

4.3.1 Transmisor con Modulación BPSK

Una vez que todos los parámetros están correctos, con la ayuda del osciloscopio de Simulink se puede realizar la simulación, para observar el funcionamiento del transmisor. En la Figura 4.19, se observa el mensaje original en amarillo y el mensaje con codificación convolucional en magenta.

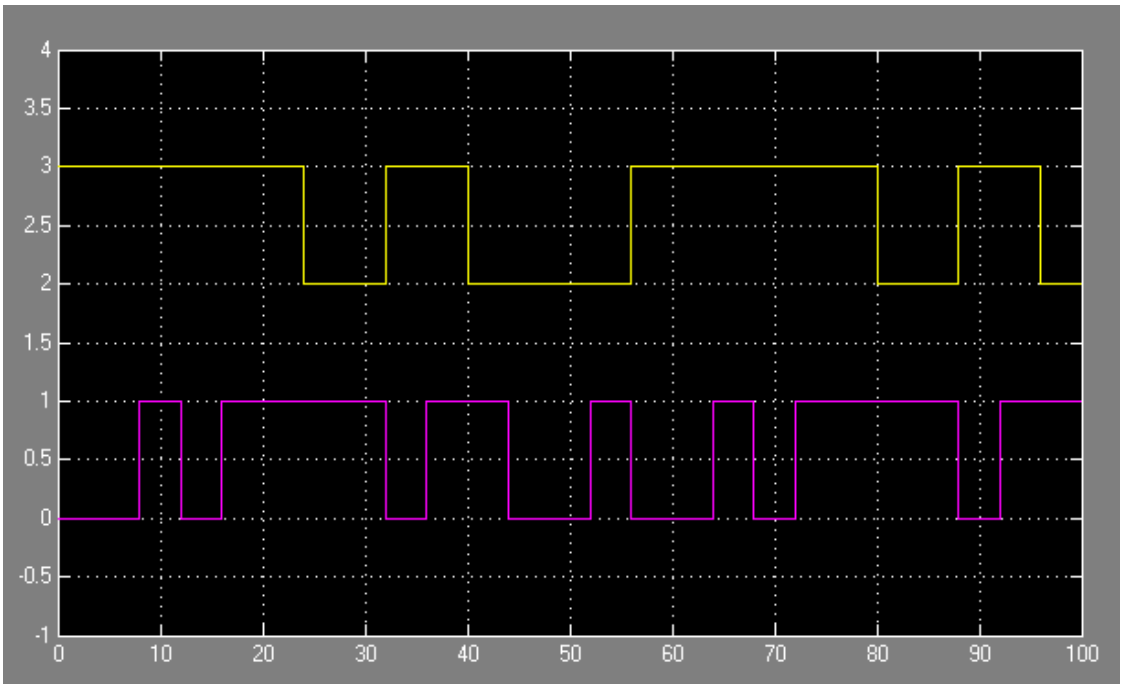

Figura. 4.19 Mensaje Original y Mensaje Codificado. BPSK

En la Figura 4.20, se puede apreciar el mensaje codificado en color amarillo y la señal con modulación BPSK en color magenta. Los resultados son los esperados, la señal modulada varía su fase entre 0º y 180º cada vez que el mensaje cambia de estado.

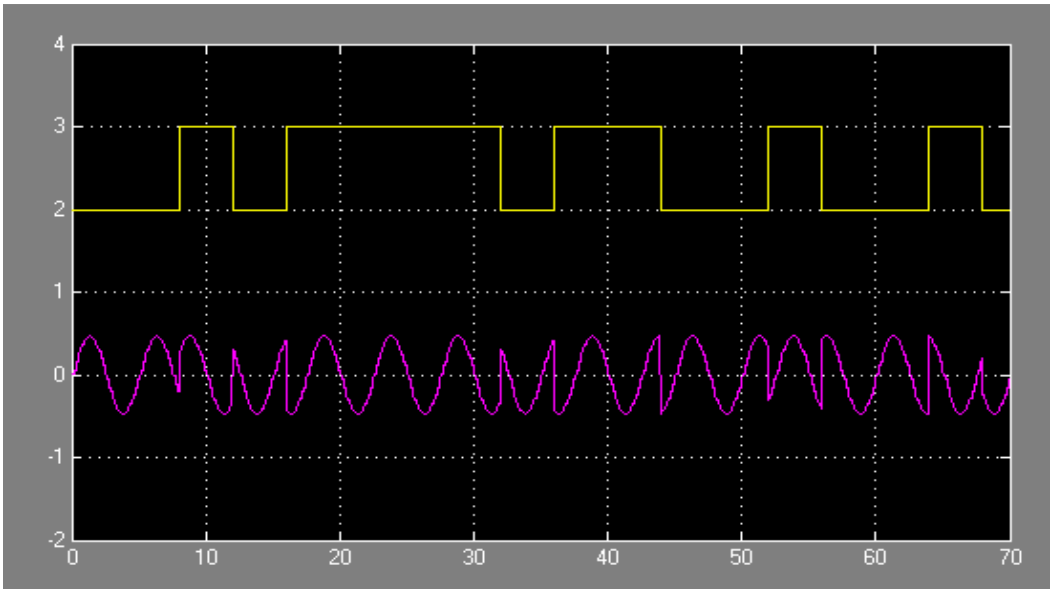

Figura. 4.20 Simulación del transmisor BPSK en *System Generator*

4.3.1 Transmisor con Modulación QPSK

En la Figura 4.21 se observa el mensaje original en amarillo y el mensaje con codificación convolucional en magenta.

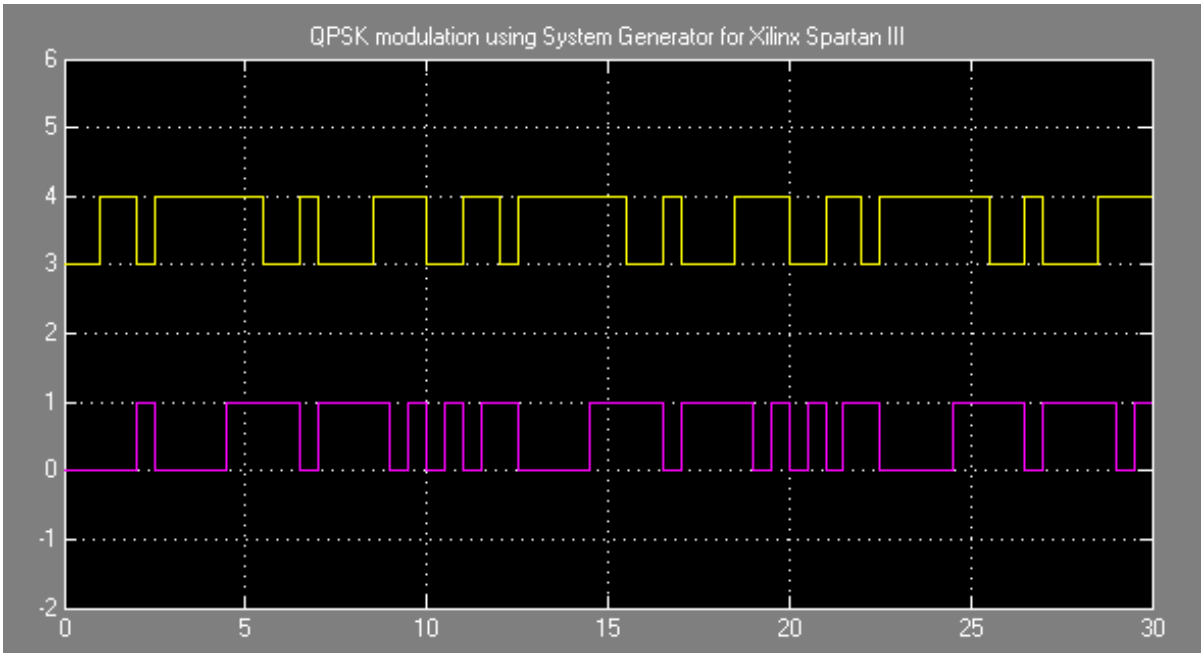

Figura 4.21 Mensaje Original y Mensaje Codificado. QPSK

En la Figura 4.22, se observan los canales I y Q y la señal modulada. Mediante esta simulación se demuestra que la señal modulada cambia su fase, dependiendo del par de bits que se tengan.

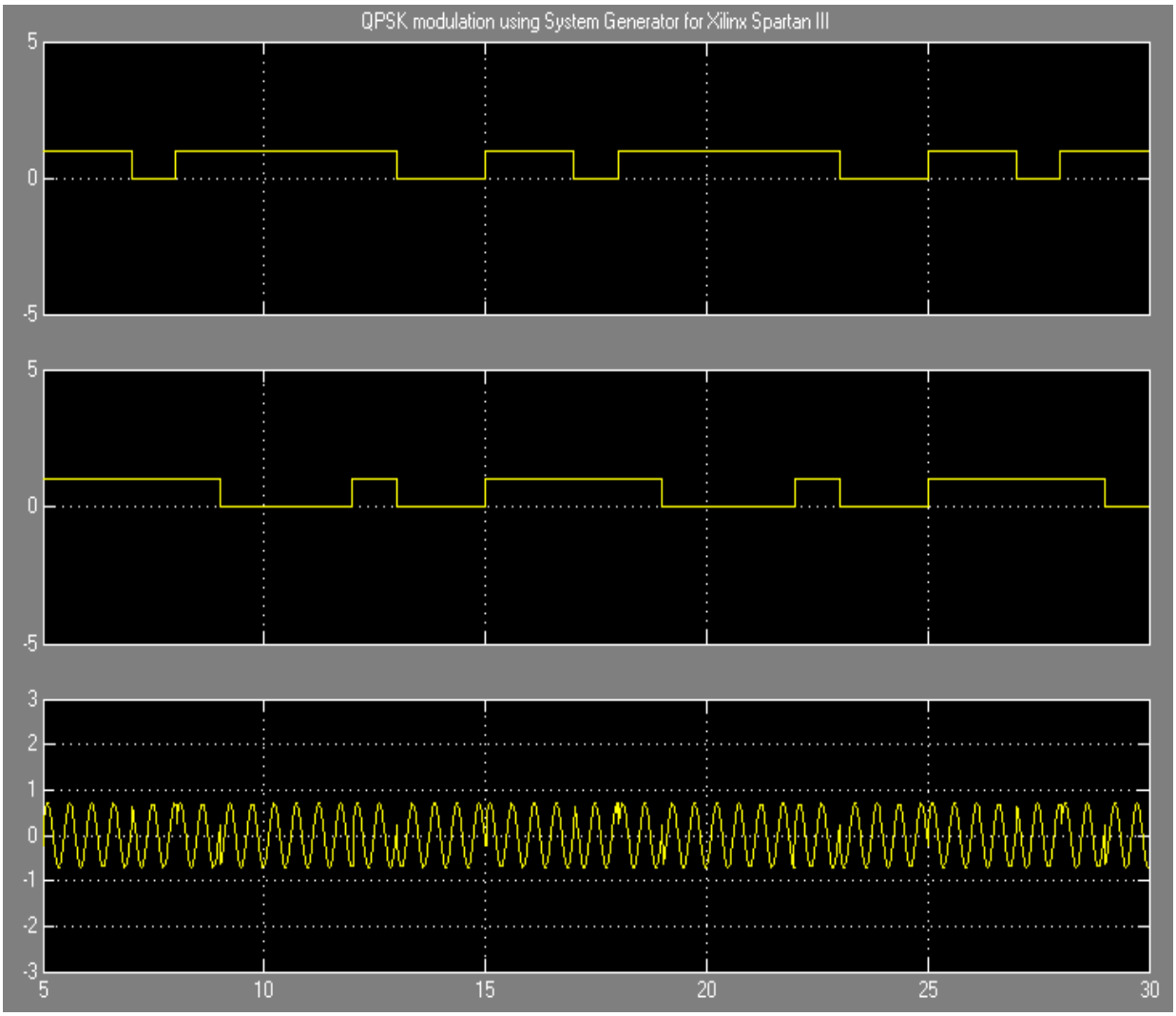

Figura. 4.22 Canales I, Q y señal modulada.

Finalmente en la Figura 4.23, se muestra el mensaje codificado en el canal 1 y la señal modulada en el canal 2.

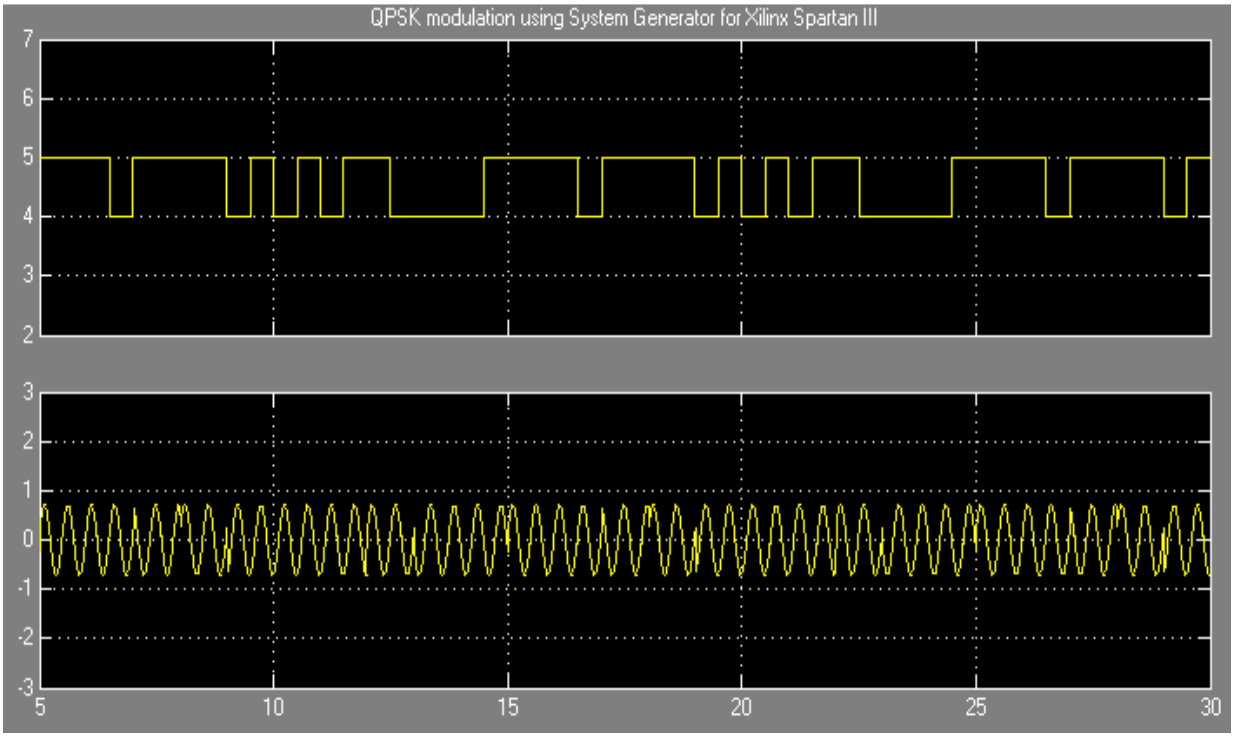

Figura. 4.23 Simulación del transmisor QPSK en *System Generator*.

4.4 Implementación en el FPGA

Como se explicó anteriormente para la implementación en el FPGA, se deben hacer algunos cambios en el código VHDL generado. Estos cambios consisten en conectar internamente los relojes del sistema. Es decir el de la entrada CLK1, la cual está conectada al bloque *black box* utilizado para el control del DAC, con el mismo reloj de todo el sistema, el cual está asignado al pin C9. Estos cambios se realizan tanto en el transmisor con modulación BPSK como en el transmisor con modulación QPSK. Esto se logra haciendo una simple modificación, siguiendo los siguientes pasos:

1. Ingresar a *Project Navigator*.
2. Abrir el proyecto creado en *System Generator*.
3. Dar doble click en el fichero TOP. Es decir en el fichero principal que contiene los demás ficheros.
4. Una vez que este fichero se abre, se observa el código generado en VHDL. En este código se debe comentar o eliminar la línea:
   clk1_net <= clk1;

Y se debe cambiar la siguiente línea:

clk1 => clk1_net; por clk1 => clkNet;

Al guardar estos cambios, se puede generar el archivo .bit para la programación del FPGA.

5. Para generar el archivo, basta con hacer doble click en *Configure Target Device* en la ventana de procesos indicada anteriormente en la Figura 3.5. Para realizar este paso es necesario que la tarjeta este conectada y reconocida.
6. Finalmente se abre la herramienta IMPACT, indicada en la Figura 3.6, en donde se añade el archivo .bit generado. Una vez hecho esto se presiona el botón derecho del mouse sobre el chip y se elige la opción *Program*. Cuando aparezca el mensaje de que la programación ha sido exitosa, es posible visualizar las señales en el osciloscopio.

CAPÍTULO 5

ANÁLISIS DE RESULTADOS

5.1 Resultados obtenidos

Es importante tener claro, como está distribuido el proyecto en la tarjeta, para conocer las funciones que se están realizando en el FPGA y otras que se realizan fuera del FPGA. La generación de señales, tanto como el mensaje y las portadoras, la modulación y la codificación de cada uno de los transmisores están implementadas en el chip FPGA, utilizando bloques de *System Generator*. La salida de las señales moduladas hacia el entorno real se la hizo mediante el DAC incluido en la tarjeta FPGA y estos resultados se los visualizan en un osciloscopio. Este esquema se lo visualiza en la Figura 5.1.

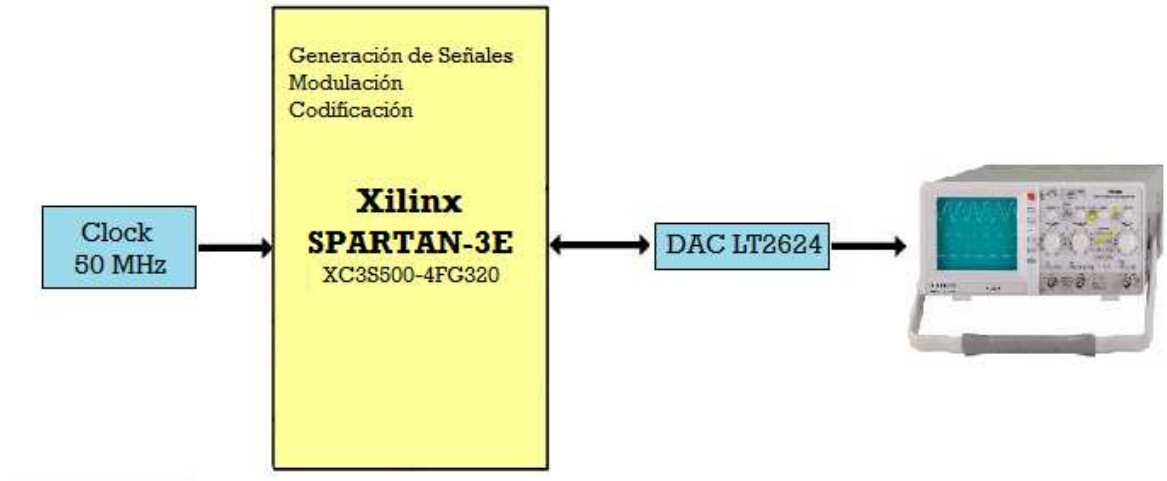

Figura. 5.1 Esquema final del proyecto

5.1.1 Resultados del transmisor con modulación BPSK

Los resultados obtenidos después de la implementación en el FPGA para el transmisor con modulación BPSK, son los que se muestran en la Figura 5.2 y 5.3.

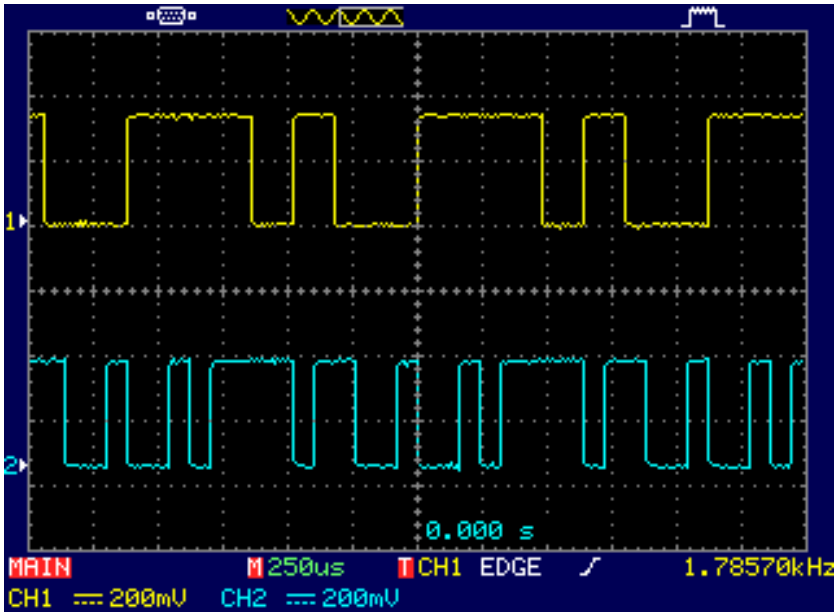

Figura. 5.2 BPSK. 1) Mensaje Original y 2) Mensaje Codificado.

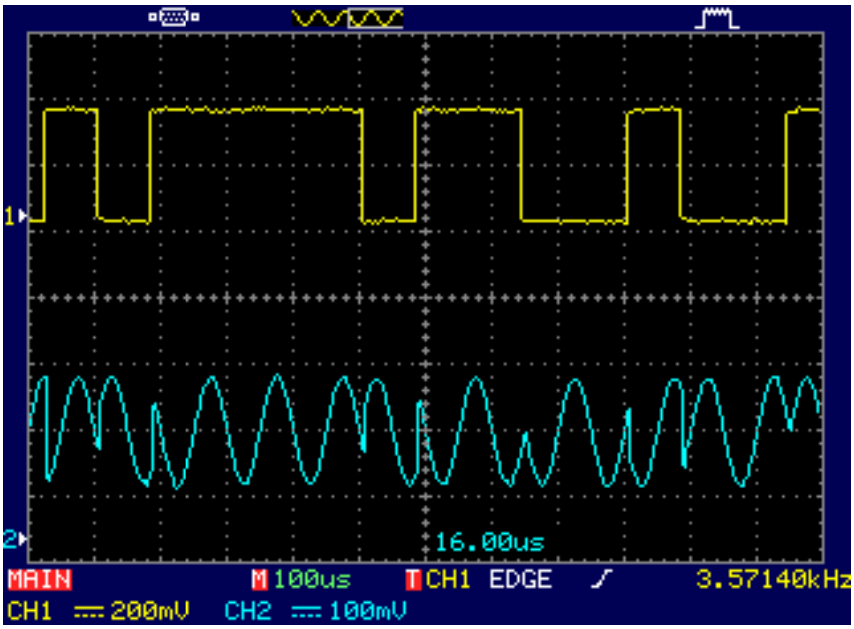

Figura. 5.3 Modulación BPSK en un osciloscopio real

Los resultados coinciden con la simulación y son los esperados. La señal modulada cambia de fase en cada cambio de estado del mensaje, como se muestra en la Figura 5.3.

5.1.2 Resultados del transmisor con modulación QPSK

En la Figura 5.4, se observa el mensaje original en amarillo y el mensaje codificado en celeste.

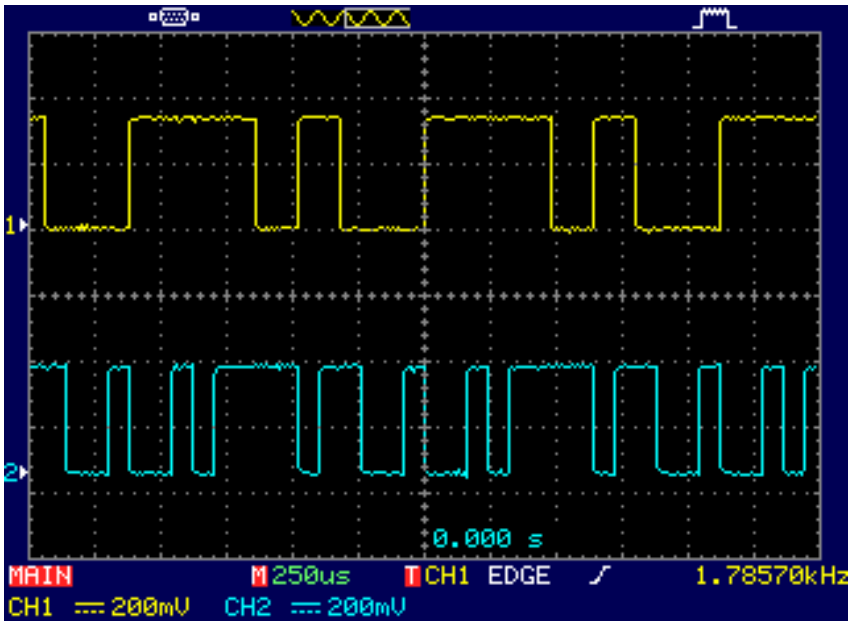

Figura. 5.4 QPSK. 1) Mensaje Original y 2) Mensaje Codificado

Los canales I y Q, se observan en la Figura 5.5. El canal I en amarillo y el canal Q en celeste.

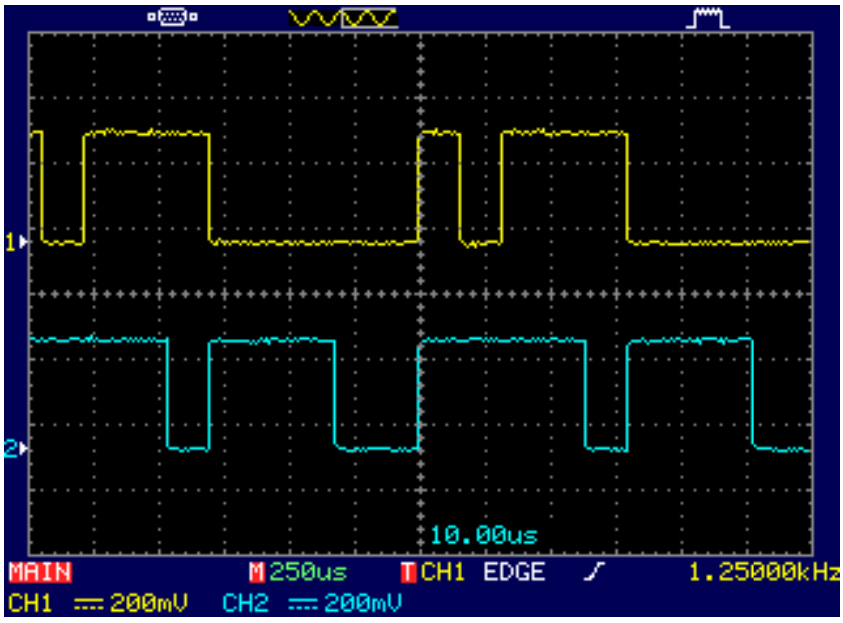

Figura. 5.5 1) Canal en fase y 2) Canal en cuadratura

Para observar el comportamiento de la señal modulada respecto de los canales en fase y cuadratura, se toma las imágenes en dos partes puesto que el osciloscopio que se utiliza solo tiene dos canales. A diferencia de la simulación en la cual se observaron las tres señales simultáneamente.

En la Figura 5.6 se observa el canal I y la señal modulada. Se puede apreciar como la señal modulada cambia la fase en cada uno de los cambios de estado del canal I, así como en algunos momentos que la señal I no cambia. Esto se debe a que en esos momentos la señal que está variando es el canal Q, entonces el par de bits es diferente y la fase varía.

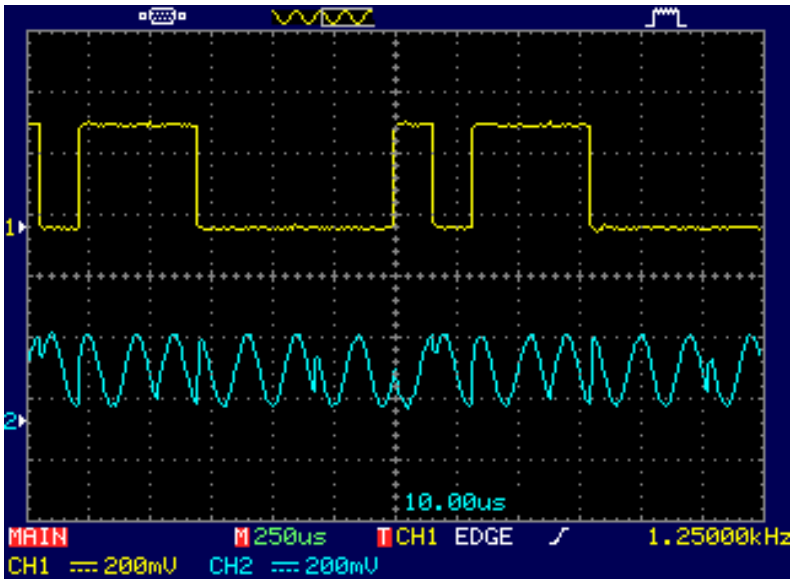

Figura. 5.6 1) Canal en Fase y 2) Señal Modulada

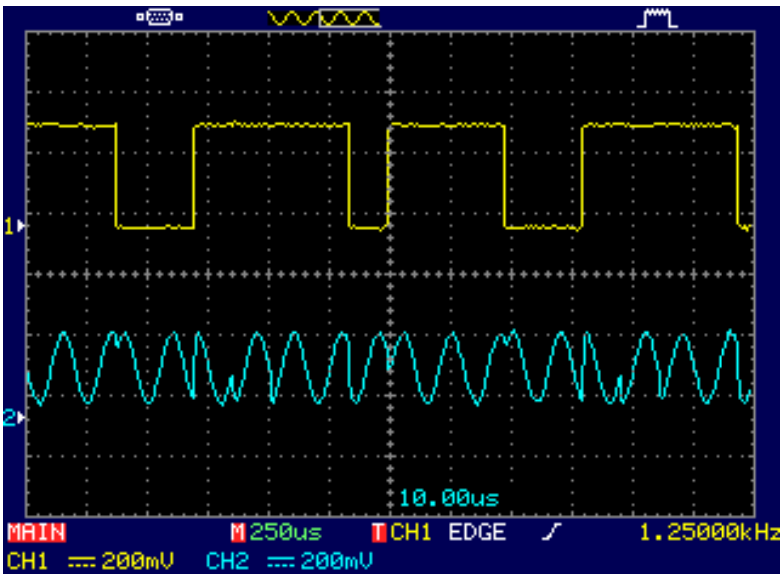

Figura. 5.7 1) Canal en Cuadratura y 2) Señal Modulada

En la Figura 5.7, se observa el canal Q y la señal modulada. El comportamiento de la señal modulada es el mismo que el de la Figura 4.23.

Finalmente en la Figura 5.8, se observa el resultado de la modulación QPSK. La señal modulada en amarillo y el mensaje codificado en celeste.

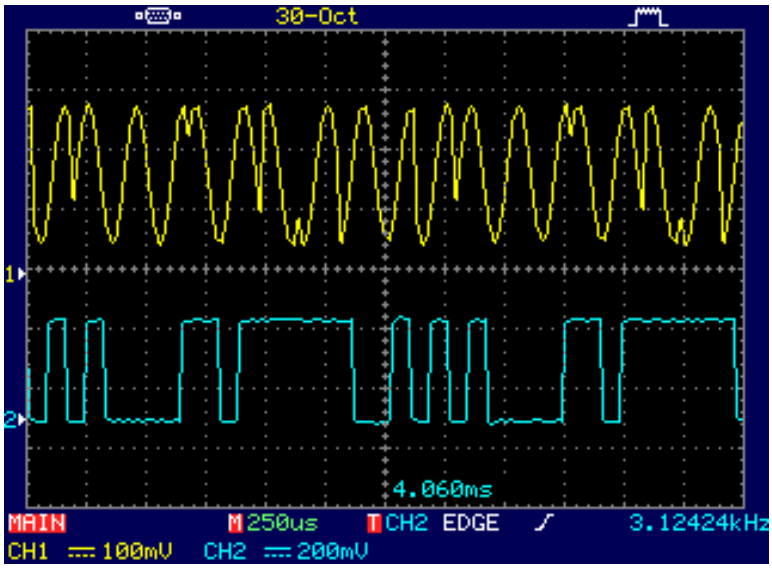

Figura. 5.8 1) Señal Modulada y 2) Mensaje Codificado

5.2 Consumo del FPGA

Los recursos empleados para implementar los transmisores tanto con modulación BPSK como con modulación QPSK se detallan en las tablas 5.1 y 5.2 respectivamente. Como se puede apreciar no se usa más del 15% del total de determinados recursos. Esto implica que hay recursos suficientes para añadir módulos si se desea. En el anexo A3, se detallan algunos de estos recursos.

Tabla. 5.1 Consumo del FPGA para el transmisor BPSK

Recursos utilizados en la implementación		
Recursos	**Usados**	**Porcentaje**
Flip Flops	233	2%
4 input LUTs	114	1%
Slices	152	3%
IOBs	13	6%
Bloques de memoria RAM	2	10%

Evidentemente hay más recursos utilizados en el transmisor con modulación QPSK. Como se explicó en la implementación, este diseño es más complejo y necesita de más módulos. El consumo del FPGA, puede ser fácilmente visualizado en ISE, al momento de abrir el proyecto creado en VHDL.

Tabla. 5.2 Consumo del FPGA para el transmisor QPSK

Recursos utilizados en la implementación		
Recursos	**Usados**	**Porcentaje**
Flip Flops	362	3%
4 input LUTs	191	2%
Slices	222	4%
IOBs	15	6%
Bloques de memoria RAM	3	15%

CONCLUSIONES Y RECOMENDACIONES

1. *System Generator* ofrece grandes ventajas en el campo de las comunicaciones, brindando una gran variedad de bloques para la implementación de cualquier sistema de comunicación. Esto hace que el tiempo de trabajo disminuya notablemente. Su entorno gráfico es muy amigable y permite al usuario realizar varias pruebas simplemente con la variación de ciertos parámetros.

2. El hecho de que sea posible interactuar un diseño realizado en el entorno gráfico de *System Generator*, con programas en VHDL o Verilog realizados en ISE, permite que haya más flexibilidad al momento de realizar un diseño. Esta característica es muy importante ya que se puede generar bloques que no existan en la librería de Xilinx en Simulink, o a su vez generar bloques ya existentes pero con mejores prestaciones.

3. La integración del DAC en XSG, se logró mediante un bloque llamado *Black Box*, 2 bloques de *Gateway In*, es decir 2 entradas, y 9 bloques de *Gateway Out* ó 9 salidas. Estas entradas y salidas fueron declaradas en el código VHDL y automáticamente generadas en Simulink al momento de introducir el código en el *black box*, y son de gran importancia para el correcto desempeño del sistema. En el bloque *black box*, es posible insertar los ficheros VHDL que se desee, esto facilitó de gran manera la integración del código VHDL para el DAC en el diseño de Simulink.

4. El desempeño del sistema, puede ser observado de manera más rápida en hardware que en software, es por esto que es importante contar con herramientas como *System Generator* las cuales permiten realizar implementaciones en hardware sobre FPGAs.

5. Los resultados obtenidos en el FPGA, fueron los esperados. El desempeño de los transmisores con modulación BPSK y QPSK, coincide con la teoría. Esto indica que la implementación de todo el diseño fue la adecuada y que el software de desarrollo y la metodología, son muy efectivos y confiables.

6. Se comprobó que la frecuencia que soporta el DAC LT2624 incluido en la tarjeta SPARTAN 3E, no va más allá del orden de los KHz, ya que las pruebas realizadas con frecuencias en las portadoras de más de 100 KHz no tuvieron un buen desempeño.

7. Se pudo observar que el consumo del FPGA no excedió al 15%. En el transmisor con modulación BPSK, el mayor consumo fue del 10 % y en el transmisor con modulación QPSK, el mayor consumo fue del 15%. Esta diferencia del 5%, se da debido a que en la modulación QPSK se utilizaron más recursos en Simulink, es decir más bloques de Xilinx. El resultado en porcentaje del consumo indica que a pesar de la gran cantidad de bloques utilizados, todavía es posible añadir nuevos módulos al diseño, si se desea modificar el sistema o integrarlo con otro proceso.

8. El máximo retardo desde una entrada hacia una salida fue de 4,373 ns. Esto demuestra la rápida respuesta que tiene un FPGA al trabajar con estos sistemas ya que el retardo obtenido es mínimo.

9. Se pudo comprobar que la implementación de cualquier sistema en un FPGA tiene muchas ventajas, ya que ofrece facilidad en cuanto a modificación, velocidad en tiempo real, optimización en tiempo y disminución de costos debido a su gran capacidad de reutilización.

10. Es aconsejable que se conozca el funcionamiento de cada bloque de *System Generator*, de esta manera será mucho más fácil configurar los parámetros de una manera adecuada y evitar errores. Esto ayudará a disminuir el tiempo en cuanto a la implementación del diseño.

11. Se recomienda tener conocimientos en cuanto a lenguaje de alto nivel, ya que a pesar de que se realice una programación mediante bloques, no siempre van a existir los que se necesite. En este caso la única solución es utilizar el bloque *black box*, añadiendo archivos programados en lenguaje de alto nivel.

12. Es recomendable seguir investigando la programación de FPGAs mediante el software *System Generator*, debido a que se deben aprovechar las ventajas que

ofrece y en el Ecuador hay pocos proyectos que se han realizado utilizando esta herramienta.

ANEXOS

A1. TOP.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity DAC_TOP1 is
    Port ( CLK:       in  STD_LOGIC;
           RST:       in  STD_LOGIC;
           DAC_MOSI: out  STD_LOGIC;
           DAC_CLR:  out  STD_LOGIC;
           DAC_SCK:  out  STD_LOGIC;
           DAC_CS:   out  STD_LOGIC;
           SPI_SS_B: out  STD_LOGIC;  -- Serial Flash
           AMP_CS:   out  STD_LOGIC;  -- Amplificador para el ADC
            AD_CONV:  out  STD_LOGIC; -- Inicio de la conversión
                                         del ADC
           SF_CE0:    out  STD_LOGIC; -- StrataFlash
           FPGA_INIT_B: out  STD_LOGIC;-- Platform Flash
           DAC_DATA_T : in STD_LOGIC_VECTOR(11 downto 0));

end DAC_TOP1;


architecture DAC of DAC_TOP1 is

signal rdy,daccs,dacsck,dacmosi : std_logic;
signal command : std_logic_vector(3 downto 0);
signal address : std_logic_vector(3 downto 0);
signal dacdata : std_logic_vector(23 downto 0);

component DAC_Control1
      Port ( CLK:          in  STD_LOGIC;
           RST:           in  STD_LOGIC;
           DAC_DATA:      in STD_LOGIC_VECTOR(23 downto 0);
           DAC_MOSI:      out  STD_LOGIC;
           DAC_SCK:       out  STD_LOGIC;
           DAC_CS:        out  STD_LOGIC;
           RDY:           out  STD_LOGIC);
     end component;


begin

     U1 : DAC_Control1
     Port map ( CLK => CLK,
                       RST => RST,
                       DAC_MOSI => dacmosi,
                       DAC_SCK => dacsck,
                       DAC_CS => daccs,
```

```
                        RDY => RDY,
                        DAC_DATA => dacdata);

process(RST,CLK,daccs,dacsck,dacmosi)
      begin
            if (RST='1') then
                  DAC_MOSI <= '0';
                  DAC_CLR <= '0';
                  DAC_SCK <= '0';
                  DAC_CS <= '1';

            elsif rising_edge(CLK) then
                  if rdy = '1' then  -- Comprobación envoi de bits
                  command <= "0011"; -- Setea el command register
                  address <= "0000"; -- Salida (DAC A)
                  dacdata(23 downto 20) <= command;
                  dacdata(19 downto 16) <= address;
                  dacdata(15 downto 4) <= DAC_DATA_T;
                  dacdata(3 downto 0) <=(others => '0');--Don't care
            end if;

            DAC_CLR <= '1';
            end if;
            DAC_CS <= daccs;
            DAC_SCK <= dacsck;
            DAC_MOSI <= dacmosi;
end process;

      ----- Desabilitando dispositivos no necesarios-----

      SPI_SS_B <= '1';
      AMP_CS <= '1';
      AD_CONV <= '0';
      SF_CE0 <= '1';
      FPGA_INIT_B <= '1';

end DAC;
```

A2. CONTROL.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity DAC_Control1 is
       Port (
            CLK:        in  STD_LOGIC;
            RST:        in  STD_LOGIC;
            DAC_DATA:   in STD_LOGIC_VECTOR(23 downto 0);
            DAC_MOSI:   out  STD_LOGIC;
            DAC_SCK:    out  STD_LOGIC;
            DAC_CS:     out  STD_LOGIC;
            RDY:        out  STD_LOGIC);
end DAC_Control1;
```

```
architecture DAC_Control1 of DAC_Control1 is

type state_type is (idle,ready,send,dummy,check);
signal state :    state_type;
signal DAC_SEND : std_logic_vector(23 downto 0);

begin
     process(DAC_DATA)
     begin
          for i in 23 downto 0 loop
          DAC_SEND(i) <= DAC_DATA(23 - i); --Empieza por MSB
          end loop;
     end process;

     process(CLK,RST)

     variable index : integer range 0 to 24 := 0;

     begin
          if (RST = '1') then
               index := 0;
          elsif rising_edge(CLK) then
               case state is
                    when idle =>
                         DAC_SCK <= '0';
                         DAC_CS <= '1';
                         index := 0;
                         DAC_MOSI <= '0';
                         RDY <= '1';
                         state <= ready;
                    when ready =>
                         RDY <= '0';
                         DAC_CS <= '0';
                         DAC_SCK <= '0';
                         DAC_MOSI <= DAC_SEND(index);
                         state <= dummy;
                    when dummy =>
                         state <= send;
                    when send =>
                         DAC_SCK <= '1';
                         state <= check;
                         index := index + 1;
                    when check =>
                         DAC_SCK <= '1';
                         if (index = 24) then
                              state <= idle;
                         else
                              state <= ready;
                         end if;
               end case;
          end if;
     end process;
end DAC_Control1;
```

A3. Arquitectura del FPGA 3E de Xilinx

Los FPGA Spartan 3E de Xilinx están conformadas por un conjunto de Bloques Lógicos Configurables rodeados por un perímetro de Bloques Programables de entrada/salida IOBs.. Estos elementos funcionales están interconectados por una jerarquía de canales de conexión *(Routing Channels)*, la que incluye una red de baja capacitancia para la distribución de señales de reloj de alta frecuencia. Adicionalmente el dispositivo cuenta con 24 bloques de memoria RAM de 2Kbytes de doble puerto, cuyos anchos de buses son configurables, y con 12 bloques de multiplicadores dedicados de 18 X 18 bits. Esta arquitectura se muestra en la Figura A4.1

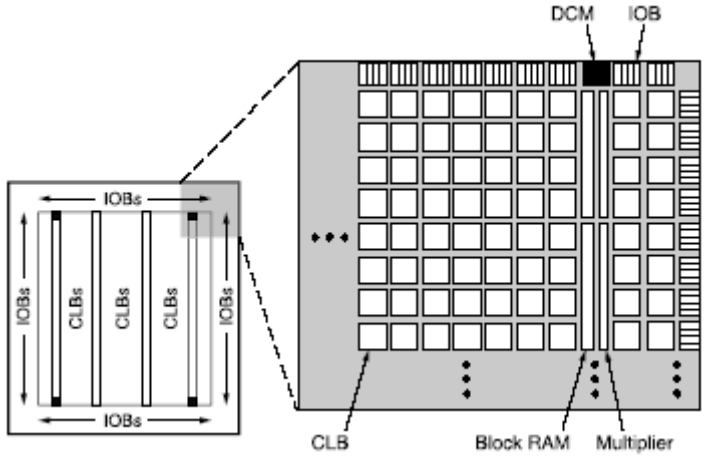

Figura A4.1 Arquitectura del FPGA 3E

A continuación se detalla algunos de los elementos funcionales del FPGA.

Los bloques de entrada/salida (IOB) suministran una interfaz bidireccional programable entre un pin de entrada/salida y la lógica interna del FPGA. El bloque básico de la red que compone la FPGA es el *slice*. Estos *slices* se organizan en bloques lógicos elementales o CLBs. Los Bloques de Lógica Configurable (CLBs) constituyen el recurso lógico principal para implementar circuitos síncronos o combinacionales. Cada CLB está compuesta de cuatro *slices* interconectados entre sí. La memoria de bloque RAM [23] consiste en varios bloques internos del FPGA de 18 Kbits. Cada uno se comporta como un chip de memoria de doble puerto. Cada puerto tiene sus propias señales de control para las operaciones de lectura y escritura.

Referencias Bibliográficas.

[1] Lumertz, Fabio," Implementação de um Codificador LDPC para um Sistema de TV Digital usando Ferramentas de Prototipagem Rápida", Universidad Estatal de Campinas, Campinas 2006.

[2] "Una Introducción a la Tecnología FPGA: Los Cinco Beneficios Principales", http://zone.ni.com/devzone/cda/tut/p/id/8259, Julio 2010.

[3] Torres, Francisco, "Dispositivos Lógicos programables", Universidad Distrital Francisco José de Caldas, Colombia.

[4] Jiménez, Raúl, SASICs, Escuela Politécnica Superior "La Rábida"

[5] "Dispositivos Semiconductores", Universidad de Buenos Aires, 2009.

[6] Olsen, Kenneth, "Memoria ROM", Universidad de Playa Ancha.

[7] "Memorias de Datos", Universidad Nacional de Río Cuarto, Argentina

[8] D. Frohman-Dentchkowsky, "A fully-decoded 2048-bit electrically programable MOS ROM," *Conferencia Internacional de Circuitos en estado Sólido Recopilación de Articulos Técnicos*, pp. 80–81, Febrero 1971.

[9] R. Cuppens, C. D. Hartgring, J. F. Verwey, H. L. Peek, F. A. H. Vollebraft, E. G. M. Devens, y A. Sens, "An EEPROM for microprocessors and custom logic," *Revista IEEE Solid-State Circuits,* vol. 20, no. 2, pp. 603–608, Abril 1985.

[10] D. C. Guterman, L. H. Rimawi, T.-L. Chiu, R. D. Halvorson, y D. J. McElroy, "An electrically alterable nonvolatile memory cell using a floating-gate structure," *Transacciones IEEE en Dispositivos Electrónicos*, vol. 26, no. 4, pp. 576–586, Abril 1979.

[11] FPGA Architecture: Survey and Challenges

[12] "Axcelerator family FPGAs", Corporación Actel , May 2005.

[13] Torres, Francisco, "Lenguajes de Descripción de Hardware", Universidad Distrital Francisco José de Caldas, Colombia.

[14] Tutorial Verilog.

[15]Pardo, Fernando, “VHDL Lenguaje para descripción y modelado de circuitos”, Universidad de Valencia, Octubre 1997.

[16]T. S. Rappaport, “Wireless Communications, Principles and Practice” 2da Edición, Cap. 7.12- 7.16, 11.3.

[17] “Digital Modulation in Communications Systems”, Agilent.

[18] Chitode, J.S, “Digital Communications”, *Technical Publications*, 2009.

[19] NI LABVIEW, www.ni.com/labview

[20] “System Generator User Guide”, Xilinx.

[21] “Spartan-3E FPGA Starter Kit Board User Guide”, Xilinx.

[22] “Datasheet LTC2604/LTC2614/LTC2624”, Linear Technology.

[23] Fazakas, A, Neag, M, Festila, L, “Block RAM versus distributed RAM implementation of SVM Classifier on FPGA”, *Applied Electronics*, 12 Noviembre 2007.