Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Telecomunicaciones y Electrónica

TRABAJO DE DIPLOMA

Realización de Codificadores y Decodificadores Convolucionales en Hardware Lógico Programable

Autor: Rafael González Rivera

Tutor: Erisbel Orozco Crespo

Santa Clara

2013

Año 55 de la Revolución

Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Telecomunicaciones y Electrónica

TRABAJO DE DIPLOMA

Realización de Codificadores y Decodificadores Convolucionales en Hardware Lógico Programable

Autor: Rafael González Rivera

Tutor: Erisbel Orozco Crespo

Profesor del Dpto. Electrónica y Telecomunicaciones

Santa Clara

2013

Año 55 de la Revolución

Hago constar que el presente trabajo de diploma fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería en Telecomunicaciones y Electrónica, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

_____

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del Tutor

Firma del Jefe de Departamento donde se defiende el trabajo

_____

Firma del Responsable de Información Científico-Técnica

PENSAMIENTO

"El conocimiento es la mejor inversión que se puede hacer"

Abraham Lincoln

DEDICATORIA

A mi madre, padre y hermana, para ustedes es este sueño.

AGRADECIMIENTOS

A mi madre que fue la principal causa de estos cinco años de estudio

A mi padre que siempre confió en mí en los momentos más difíciles

A mi hermana, sin sus consejos hubiera sido mucho más difícil

A mi tutor por la ayuda brindada

A todos ustedes, gracias.

TAREA TÉCNICA

Con el propósito de cumplir los objetivos trazados y conseguir los resultados esperados en este trabajo, se tuvo en cuenta una serie de tareas técnicas que serán una guía para la elaboración del informe final.

- Caracterización de los códigos convolucionales, dispositivos FPGAs y lenguajes de descripción de hardware.
- Proposición de una metodología de diseño sobre hardware lógico programable y en específico para las FPGAs de Xilinx.
- Investigación referida a las principales formas de generar automáticamente códigos en VHDL.
- Comprobación del funcionamiento de un modelo generado en Simulink, con el uso de Xilinx ISE y mediante la realización en el Kit de Nexys II.

Firma del Autor	Firma del Tutor

RESUMEN

En el presente informe se recoge el desarrollo de una tarea, que consiste en la realización de codificadores y decodificadores convolucionales en una FPGA de Xilinx. La idea surge debido a la necesidad de estudiar temas referidos al hardware lógico programable, ya que existe una tendencia actual al desarrollo de diversas aplicaciones en los dispositivos FPGAs. Se une este tema a las comunicaciones digitales por medio de los códigos convolucionales, que son un método de corrección de errores que se usa en gran medida en los sistemas modernos de comunicación. Para la ejecución de este proyecto se realizó una investigación sobre las principales características de los codificadores convolucionales, la decodificación mediante el algoritmo de Viterbi, los dispositivos FPGAs actuales y los softwares Xilinx ISE y Matlab/Simulink. También se describen algunas de las principales formas de generar automáticamente códigos VHDL. Una de estas formas, basada en el Simulink HDL coder, se implementa en un FPGA de Xilinx.

TABLA DE CONTENIDOS

INTRODUCCIÓN

EL hardware lógico programable es muy utilizado en el mundo de la ingeniería con distintos fines como: el procesamiento digital de señales, aplicaciones de control, comunicaciones, creación de prototipos de hardware, etc. En la Facultad de Eléctrica de la UCLV, en la carrera de Telecomunicaciones y Electrónica, existen asignaturas de la disciplina de electrónica que tratan el tema. Se tiene entonces la necesidad de realizar investigaciones que relacionen el hardware lógico programable con los temas de electrónica y comunicaciones.

Se dispone de un kit con una FPGA Spartan 3E de Xilinx. Con tal de tributar a los intereses de la carrera fue elegido el tema de los códigos convolucionales por su importancia para las comunicaciones y su aparición en los sistemas de comunicaciones actuales; por ejemplo, el sistema UMTS que utiliza de formas diversas (micro programada o con hardware dedicado) dichos mecanismos de codificación y detección de errores. Además resulta de interés la utilización de varias herramientas para generar códigos VHDL. De esta forma la realización de codificadores y decodificadores convolucionales se facilita un tanto.

Una de las técnicas más novedosas para el diseño electrónico digital, implica la generación automática del modelo en lenguaje de descripción de hardware. Para ello existen varias soluciones y todas tienen un conjunto de beneficios en común. Es de esperar que un sistema de corrección de errores, tan bien documentado y utilizado como como el de los códigos convolucionales, tenga al menos una alternativa para hacerlo funcionar sin tener que escribir todo el modelo desde cero.

Dominar este tipo de metodologías de trabajo con FPGAs implicaría un ahorro considerable de tiempo, inversiones y, en el caso de usar herramientas del propio fabricante, lograr diseños bien optimizados.

Por tanto, el objetivo general de este trabajo de diploma es realizar una aplicación con codificadores y decodificadores convolucionales para una FPGA de Xilinx, a partir de un modelo HDL generado automáticamente con el Simulink HDL Coder. Con el fin de darle cumplimiento al objetivo general se han trazado los siguientes objetivos específicos:

- Realizar una búsqueda y revisión bibliográfica al respecto de los códigos convolucionales y su realización en hardware lógico programable.
- Seleccionar la metodología de diseño sobre hardware lógico programable, en específico para las FPGAs de Xilinx.
- Hacer una revisión de las principales formas de generación automática de códigos VHDL.
- Implementar un diseño para FPGA que utilice codificadores y decodificadores convolucionales, a partir de un modelo generado con el Simulink HDL Coder.

A partir de los objetivos específicos aparecen las siguientes interrogantes:

- ¿En qué medida la bibliografía justifica la utilización de sistemas correctores de errores en electrónica digital programable?
- ¿Cuál es la metodología más factible para el diseño sobre FPGA cuando se genera el modelo HDL automáticamente?
- ¿Cuáles son las principales formas de generar códigos VHDL automáticamente?
- ¿Cómo diseñar y comprobar con un solo kit de FPGA un par codificador/decodificador convolucional?

El informe del trabajo de diploma está dividido en introducción, tres capítulos, conclusiones y recomendaciones.

CAPÍTULO 1: Los códigos convolucionales y la electrónica digital programable.

Este capítulo recoge algunos conocimientos teóricos necesarios para introducir el tema de los códigos convolucionales desarrollados en FPGAs, que es el objeto de estudio de este trabajo. En él se expone de manera breve la arquitectura y características funcionales de los FPGAs y los lenguajes de descripción de hardware. También presenta en síntesis varias publicaciones realizadas por autores que, encontraron utilidad en los diseños de códigos convolucionales para arquitecturas FPGAs.

CAPÍTULO 2: Códigos convolucionales en lenguaje de descripción de hardware.

Se dedicó al establecimiento de una metodología para el diseño sobre hardware lógico programable de Xilinx. También se realizó un estudio sobre algunas de las principales herramientas y fuentes que generan códigos VHDL. A modo de material, se comentó sobre el Xilinx ISE como herramienta de desarrollo.

CAPÍTULO 3: Realización de un ejemplo con códigos convolucionales sobre FPGA

En este capítulo se presentó el resultado de la realización de los modelos sobre el kit Nexys2. Claro que para ello hubo que comentar el diseño antes, en el sentido de definir cada modelo. Además se describió sobre la utilización de los recursos del Nexys2, como por ejemplo lo módulos de puertos periféricos. Por último, contiene la verificación del diseño.

CAPÍTULO 1. LOS CÓDIGOS CONVOLUCIONALES Y LA ELECTRÓNICA DIGITAL PROGRAMABLE

En este capítulo se hace una recopilación de los conocimientos teóricos necesarios para introducir el tema de los códigos convolucionales y su realización en hardware lógico programable. Se expone de manera breve la arquitectura y características funcionales de los FPGAs y los lenguajes de descripción de hardware. También se hace una revisión de las principales publicaciones realizadas con respecto a diseños de codificadores y decodificadores convolucionales en FPGA.

1.1 Fundamentos de los códigos convolucionales

Debido a la fluctuación del nivel de potencia de una señal en un entorno móvil, es difícil mantener los parámetros de calidad por encima de un umbral para todo instante de tiempo. Cuando se presentan intervalos más o menos largos por debajo del nivel mínimo de umbral, se producen errores en los bits transmitidos que pueden degradar la comunicación. Para evitar este fenómeno es necesario enviar la información codificada, añadiendo una redundancia que permita detectar los errores; disminuye así la probabilidad de error en el bit, para una determinada relación señal a ruido (PIÑEIROS, 2010). Este escenario es solo un ejemplo para el cual podrían encontrar aplicación los códigos convolucionales.

Los códigos convolucionales son códigos lineales. Se utilizan para proteger la información añadiendo redundancia a la misma, de manera que las palabras de código tengan la distancia mínima necesaria. Las palabras de un código convolucional se generan no sólo a partir de los dígitos de información actuales, sino también con la información anterior en el tiempo, es decir, un codificador convolucional es un sistema con memoria (GEA, 2009).

1.1.1 Codificación

Un código convolucional se genera pasando la secuencia de información a transmitir por un registro de desplazamiento lineal y está especificado principalmente por tres parámetros:

- n es el número de bits de la palabra codificada
- k es el número de bits de la palabra de datos
- K longitud del código

La codificación convolucional es una codificación continua, en la que la secuencia de bits codificada depende de los bits previos. El codificador consta de un registro de desplazamiento de K segmentos de longitud k. La tasa de codificación o razón de código es $R = k/n$ (Sklar, 1988).

El proceso de codificación se realiza utilizando un dispositivo lógico en el codificador. La palabra codificada se obtendría como el resultado de realizar una serie de operaciones lógicas entre determinados bits que están almacenados en los registros intermedios.

El codificador de la Figura 1.1 tiene una tasa de codificación $R = 1/2$, por cada bit de la palabra de datos que se introduzca, a la salida existen dos bits de palabra de código. Cada vez que entra un bit, es colocado en el espacio extremo izquierdo del registro y el resto de los bits son desplazados una posición a la derecha. El *switch* muestrea a la salida de cada sumador modulo-2 alternamente, cada vez que ingresa un nuevo bit a la entrada. La conexión entre los sumadores y los bits del registro, define la otra parte de las características de la secuencia código. Las conexiones no son escogidas o cambiadas arbitrariamente, según (Sklar, 1988), el problema de escoger conexiones para producir buenas propiedades de distancia es complicado y en general no ha sido resuelto; aunque buenos códigos han sido encontrados mediante la búsqueda computarizada con longitudes menores que 20.

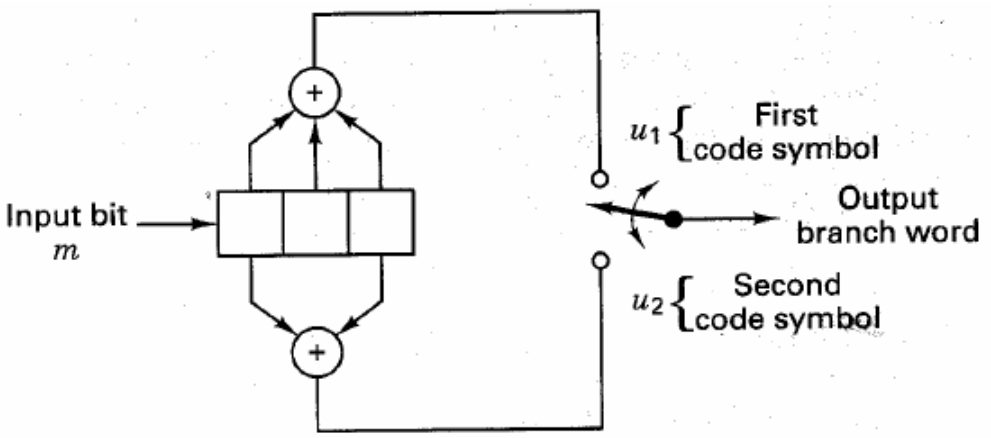


Figura 1.1 Diagrama del codificador convolucional tomado de (Sklar, 1988).

A diferencia de un código de bloque que tiene una longitud de palabra n fija, un código convolucional no tiene un tamaño de bloque particular. Aunque, los códigos convolucionales a veces son forzados a una estructura de bloque mediante el *truncado periódico*. Esto requiere de un relleno con ceros al final de la secuencia de datos de entrada, con el propósito de limpiar o nivelar los bits de datos en el registro de desplazamiento del codificador. Dado que los ceros sumados no transportan información, la relación de código efectiva cae por debajo de k/n. Para mantener la relación de código k/n, el período de truncado debe hacerse tan grande como se pueda.

Varias formas existen para representar un codificador convolucional, las más comunes son: vectores de conexión o polinomiales, el diagrama de estado, el diagrama de árbol y el diagrama *trellis* (Sklar, 1988).

Un codificador convolucional es también de la clase de máquinas de estado finitas, algo que está en perfecta correspondencia con el tipo de máquinas de estado que pueden realizarse con electrónica digital secuencial. El adjetivo finito se refiere a que hay un número finito de estados únicos en los que el sistema se puede encontrar. La Figura 1.2 muestra el diagrama de estados para el codificador convolucional mostrado en la Figura 1.1.

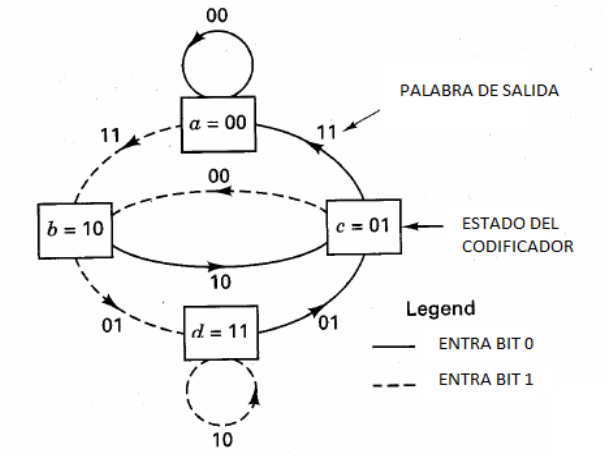


Figura 1.2 Diagrama de estados para el codificador de la Figura 1.1, también tomado de (Sklar, 1988).

1.1.2 Algoritmo de Viterbi

El algoritmo de Viterbi es un método muy usado para decodificar los códigos convolucionales. Esencialmente realiza la decodificación por máxima probabilidad o principio de máxima verosimilitud; sin embargo, reduce la carga computacional aprovechándose de la estructura especial en el código Trellis.

El diagrama de Trellis, como se muestra en la Figura 1.3, posee una estructura repetitiva y proporciona una descripción del codificador muy manejable, por lo cual es quizás la forma de representación más usada.

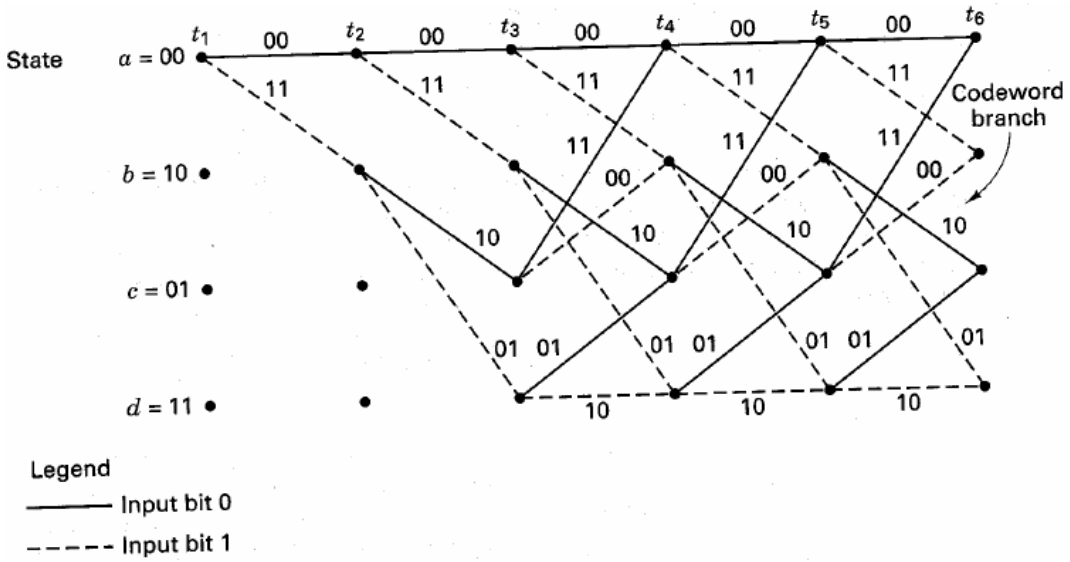


Figura 1.3 Diagrama de Trellis.

El algoritmo de Viterbi elimina de la consideración aquellos caminos de Trellis que posiblemente no podrían ser candidatos para la opción de máxima probabilidad. Cuando dos caminos entran en el mismo estado, el que tiene la mejor métrica es escogido; este camino se llama camino superviviente. La selección de caminos supervivientes es realizada para todos los estados. El rechazo de los caminos improbables reduce la complejidad de la decodificación. En 1969 se demostró que el algoritmo Viterbi es, de hecho, la máxima probabilidad (Sklar, 1988).

Varios algoritmos han sido propuestos por autores, aunque en realidad son prácticamente el mismo. El siguiente fue tomado de (Kaur, 2006):

1. Calcular la métrica de cada una de las distancias.
2. Almacenar recursivamente el camino más corto al tiempo n, en términos del camino más corto al tiempo $n-1$; se debe actualizar el camino superviviente de la señal. Esto se conoce como el mecanismo de recursividad sumar-comparar-seleccionar o ACS[1].
3. Encontrar recursivamente el camino más corto que conduzca a cada estado de Trellis usando las decisiones del paso 2. Este proceso es la decodificación del camino sobreviviente. Si todos los caminos sobrevivientes son seguidos hacia atrás en el tiempo, se unen en un único camino: el más probable para la señal.

1.1.3 Turbo códigos

Entre los desarrollos tecnológicos más importantes de los sistemas de comunicación en los últimos años se encuentran los Turbo-códigos, los cuáles casi permiten alcanzar los límites teóricos de la capacidad del canal planteados por Claude E. Shannon en 1948 para canales con ruido blanco gaussiano aditivo (FRANCO, 2006). Los efectos prácticos de una codificación cercana a la ideal se traducen en realizar transmisión de datos usando la potencia más baja posible, a la mayor velocidad posible y con valores de BER (*Bit Error Rate*) muy bajos. Los turbo-códigos fueron propuestos por los ingenieros franceses Claude Berrou y Alain Glavieux en la conferencia internacional de la IEEE de 1993 en Ginebra, Suiza.

[1] Del inglés *add-compare-select*.

Desde su invención, los Turbo-códigos han sido ampliamente estudiados y adoptados por varios sistemas de comunicación. Tasas de transmisión elevadas permiten la entrega de contenido multimedia y de otras formas densas de información a dispositivos portables. La ganancia de codificación extra ofrecida por los Turbo-códigos también puede utilizarse en dispositivos inalámbricos para reducir la fuerza de sus señales, lo que conlleva a que más dispositivos puedan compartir el mismo espectro de frecuencia al reducirse la interferencia inter-dispositivo.

Las tres áreas claves en que los Turbo-códigos proveen un mejoramiento en el desempeño son:

- Capacidad: los Turbo-códigos logran un desempeño cercano a los límites teóricos de la capacidad.
- Eficiencia en el costo del sistema: un usuario es capaz de enviar la misma cantidad de información empleando únicamente la mitad del ancho de banda.
- El número de usuarios: ejemplo, un proveedor de servicio satelital es capaz de duplicar el número de usuarios sin incrementar la capacidad del satélite (FRANCO, 2006).

Más que un esquema de codificación de errores, los Turbo-códigos son una forma de pensamiento que se basa en la aplicación de dos conceptos básicos y poderosos al control de errores: la retroalimentación y el principio conocido como *divide y vencerás*.

En la actualidad la tecnología ha incorporado los Turbo-códigos para los desarrollos de enlaces satelitales, en los sistemas de comunicación espacial y en los sistemas de telefonía de tercera generación. También se están utilizando para aumentar la velocidad en la transmisión de datos en versiones mejoradas de redes WiFi y en dispositivos portátiles con aplicaciones de datos de multimedia (Chaudhry and Mahboob, 2011).

Los Turbo-códigos están siendo implementados y probados en arquitecturas de hardware reconfigurables, pues los costos de implementación en estas son menores en comparación con los de un ASIC[2]. Además es versátil debido a que la reconfiguración del hardware

[2] *Application Specific Integrated Circuit*

puede adaptarse a las particularidades de las aplicaciones y permite una constante verificación y corrección de los diseños antes de ser implementados finalmente.

1.2 Principales aplicaciones de los códigos convolucionales

Los códigos convolucionales se utilizan generalmente como códigos de canal para corregir los errores de los canales ruidosos, ya sean debidos a limitaciones de potencia, típico en los canales digitales banda base, como a limitaciones en banda, propio de los canales digitales paso banda.

En algunas aplicaciones, los códigos convolucionales se utilizan concatenados con otro código (ejemplo, Reed-Solomon) de manera que éste es el código exterior y el código convolucional es el código interior al canal. Dicha combinación de códigos se prefiere, para obtener mayor capacidad de corrección de errores, frente a la utilización de un código convolucional con mayor distancia libre, en aras a la simplicidad del decodificador.

Se emplean en los sistemas audiovisuales que requieren corrección de errores en tiempo real como por ejemplo: en los estándares de televisión digital por satélite (DVB-S), por cable (DVB-C) y terrestre (DVB-T), así como en algunos estándares de telefonía móvil como GSM.

Asimismo, incluyen códigos convolucionales algunos sistemas de transmisión para el bucle de abonado como, por ejemplo, los antiguos modem telefónicos V.32bis y V.34 o los modernos estándares xDSL como ADSL2+ y SHDSL. También es frecuente incluir codificación convolucional en los sistemas de comunicación para el espacio profundo: sondas espaciales, misiones no tripuladas, etc. (GEA, 2009).

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

codificación convolucional de tasa $1/2$ en el enlace descendente y $1/3$ en el enlace ascendente con una longitud de restricción de 9.

En la televisión digital los códigos convolucionales son altamente utilizados debido a su capacidad de corregir y detectar errores. Las tasas que se utilizan son muy variadas en dependencia de si la aplicación necesita ser robusta o tener una alta capacidad. En el estándar ISDB-T6 se utilizan códigos convolucionales de tasa $1/2$, $2/3$, $3/4$, $5/6$ y $7/8$ (PIÑEIROS, 2010).

1.3 Lógica programable: FPGA

Un dispositivo lógico programable (PLD). Es un circuito integrado, formado por una matriz de puertas lógicas y flip-flops.

La estructura básica de un PLD permite realizar cualquier tipo de circuito combinacional basándose en una matriz formada por compuertas AND, seguida de una matriz de compuertas OR. Tres son los tipos más extendidos de PLDs, la PROM, PAL, y la PLA.

1.3.1 Generalidades de la FPGA

Los FPGAs (*Field Programmable Gate Arrays*), son dispositivos semiconductores que contienen componentes lógicos programables e interconexiones programables entre ellos. Los componentes lógicos programables pueden ser programados para duplicar la funcionalidad de puertas lógicas básicas tales como AND, OR, XOR, NOT o funciones combinacionales más complejas tales como decodificadores, o simples funciones matemáticas. En muchos FPGAs, estos componentes lógicos programables también incluyen elementos de memoria, los cuales pueden ser simples *flip-flops* o bloques de memoria más complejos. La evolución de los FPGA se ha basado en tres ejes fundamentales. El objetivo principal es utilizar geometrías cada vez más pequeñas usando transistores más pequeños y rápidos, acompañadas de costos por área cada vez menores.

Una lógica de interconexiones programables permite que los bloques lógicos de un FPGA, puedan ser interconectados según la necesidad del diseñador de un sistema. Estos bloques lógicos e interconexiones pueden ser programados después del proceso de manufactura por

el usuario o diseñador, dependiendo de los recursos del FPGA y así desempeñar cualquier función lógica necesaria (PIÑEIROS, 2010).

1.3.2 Arquitectura de un FPGA

La arquitectura de un FPGA consiste en arreglos de compuertas lógicas que son programadas y que están interconectadas entre sí mediante canales de conexiones verticales y horizontales tal como se muestra en la Figura 1.4.

Los bloques lógicos o CLBs se conforman de una parte combinacional que implementa operaciones o funciones lógicas booleanas y una parte secuencial formada por *flip-flops*, que sirven para sincronizar la salida de un bloque con una señal de reloj externa, lo que permite realizar circuitos secuenciales e implementar registros (Castillo et al., 2008).

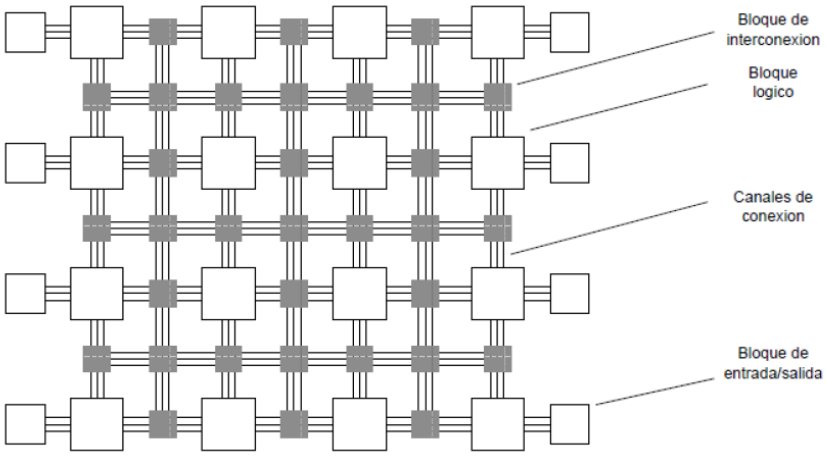


Figura 1.4 Arquitectura general de un FPGA (PIÑEIROS, 2010).

La estructura de un bloque lógico pude variar según el fabricante, pero la parte combinacional se basa principalmente en una Look-Up Table (LUT). Una LUT es un componente de memoria que almacena una tabla de verdad. Las direcciones de la memoria son las entradas de la función lógica que se debe implementar y en cada celda de la memoria se almacena el resultado de la combinación correspondiente de las entradas.

Las LUTs de N entradas son básicamente una memoria que, cuando es programada apropiadamente, puede realizar cualquier función de hasta N bits de entrada. Un FPGA típico tiene CLBs con una o más LUTs de 4 entradas, opcionalmente puede contar con *flip-*

flops tipo *D* y con circuitos que implementen el cálculo de acarreos de forma rápida (Castillo et al., 2008).

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

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

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

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

Las LUTS que se basan en celdas SDRAN permiten utilizar estas celdas como memoria RAM o como registros de desplazamiento. Una LUT de 4 entradas puede ser utilizada para formar una memoria RAM de 16x1 bits. A su vez, las 16 celdas de memoria pueden ser interconectadas en forma de cadena formando un registro de desplazamiento.

Los FPGAs actuales incluyen lógica y elementos de interconexión útiles para implementar cadenas de acarreo rápidas (*fast carry chains*), cuya función es la de propagar de forma rápida señales de acarreo entre dos elementos lógicos. Las cadenas de acarreo permiten realizar, de forma eficiente, sumadores, multiplicadores, contadores y comparadores de gran velocidad (PIÑEIROS, 2010).

La gran mayoría de dispositivos FPGA incluyen bloques de memoria RAM, además de la RAM distribuida en las LUTs. Estos bloques de memoria pueden ser configurados de

diversas maneras, formando memorias de un solo puerto (*single-port RAMs*), memorias de doble puerto (*dual-port RAMs*), memorias FIFO (*first-in first-out*), etc.

Algunas funciones lógicas tales como suma y multiplicación son inherentemente lentas si se realizan interconectando un gran número de bloques lógicos. Para aplicaciones de procesamiento digital de señales, donde es necesario el uso de varias sumas y multiplicaciones a mayor velocidad, denominadas como MAC de *Multiply and Accumulate*, el hecho de que los FPGAs cuenten con una cantidad de multiplicadores embebidos los hace ideales para ser utilizados.

1.4 Revisión de materiales publicados sobre la realización de códigos convolucionales en dispositivos lógico programables

A continuación se muestra una reseña de los principales artículos revisados sobre la realización de códigos convolucionales en los dispositivos lógico programables.

En el artículo de (Chaudhry and Mahboob, 2011) se hace una presentación de un decodificador de Viterbi de 4 bits para FPGA. Los datos de telemetría son codificados con código convolucional según el *Telemetry Synchronization and Channel Coding Standard* del ECSS (*European Cooperation for Space Standardization*). En el proyecto de estos autores se emplea Verilog para programar la placa de evaluación Xilinx Virtex-4 LX. Constituyó una buena referencia para este trabajo, en cuanto a arquitectura del decodificador de Viterbi con un fin específico.

En el artículo (Costello Jr et al., 2006) se hace una comparación entre los códigos convolucionales y los bloques LDCP, basada en varios factores como: complejidad de codificación, complejidad de la decodificación computacional, complejidad de la decodificación de hardware, requerimientos de memoria para la decodificación, retardo de decodificación y los requisitos de implementación VLSI. Sirvió como referencia para la conceptualización del diseño sobre hardware, desde el punto de vista matemático.

En (Xu et al., 2007) se realiza el análisis de un decodificador convolucional de razón $1/3$ y su decodificador. Como conclusión se presenta el diseño de un decodificador de Viterbi para la multibanda OFDM (MB-OFDM) y se implementa en FPGA, logrando un rendimiento de hasta 432 Mbps en el dispositivo Virtex-4 de Xilinx.

Uno de los trabajos más recientes que se puede encontrar es (Yanyan et al., 2012), en este se aborda el tema de la decodificación para lograr mayor eficiencia con menor complejidad. Se logra el planteamiento y los resultados experimentales muestran que con un reloj del sistema de 64 MHz, la razón de decodificación no cae por debajo de 16Mbps.

En el artículo (Alneema and Qassim, 2009) se hace el diseño de un codificador convolucional con longitud de restricción 3, razón de código $1/2$ y de un decodificador de Viterbi. Los autores publicaron sobre la realización en la FPGA Spartan 3E de Xilinx.

En la tesis de (PHU, 2010) se diseña un sistema de longitud de restricción 7 y razón de código $R = 1/2$ que posteriormente se implementa en una FPGA de Altera.

1.5 Generalidades de los lenguajes de descripción de hardware

El HDL es una herramienta formal para describir el comportamiento y la estructura de sistemas usando código. El diseñador puede así describir el funcionamiento del sistema, definiendo que es lo que el sistema debe hacer (comportamiento), cómo debe hacerlo (algoritmo) y con qué hacerlo (flujo de datos y estructuras).

Existen muchos lenguajes para la descripción de circuitos digitales, pero el lenguaje de descripción de circuitos integrados de muy alta velocidad, en inglés Very High Speed Integrated Circuit Hardware Description Language (VHDL) es uno de los de mayor popularidad. La mayoría de los otros lenguajes suelen ser propios de una determinada herramienta o fabricante de circuitos integrados, por ello resultan a veces más óptimos pero no estándar. El lenguaje VHDL tiene también un amplio campo de aplicación, desde el modelado para simulación de circuitos, hasta la síntesis automática de circuitos (Arnone, 2008).

1.5.1 Características del lenguaje VHDL

VHDL es un lenguaje de descripción de hardware para circuitos integrados de alta velocidad. Un lenguaje de estas características permitiría describir los circuitos para su documentación, modelarlos y evaluarlos mediante simulaciones antes de incurrir en los grandes gastos de fabricación. En la actualidad, VHDL se utiliza no solo para modelar circuitos electrónicos sino también para crear, o sintetizar nuevos circuitos. La capacidad de

sintetizar circuitos a partir de los modelos en VHDL (u otro lenguaje de descripción de hardware) surgió después de la creación del lenguaje, con la aparición de herramientas que traducen los modelos VHDL a circuitos reales (Pérez et al., 2002).

Los lenguajes de descripción de hardware, y el VHDL en particular, posibilitan la descripción de sistemas digitales a diferentes niveles de abstracción. Las metodologías de diseño pueden ser clasificadas en dos grupos dependiendo del nivel de abstracción de las descripciones: ascendente, en ingles *bottom-up*, donde el diseñador construye el sistema de componentes elementales, como compuertas lógicas y transistores. Este método necesita de una fase previa donde el sistema ha sido dividido en bloques más pequeños. Descendente, en inglés *top-down*, en este caso, el diseñador construye el sistema partiendo de una descripción funcional, y a través de un proceso de síntesis, llega a la implementación final, entendiendo por síntesis a la conversión de una descripción de alto nivel a otra de nivel inferior.

La solución *top-down* es más adecuada para metodologías de diseño basadas en HDL, y está teniendo gran difusión gracias a las herramientas de diseño asistido por computadora, en inglés Computer Aided Design (CAD), de síntesis y simulación (Arnone, 2008).

Ventajas del VHDL:

- El uso de VHDL permite la descripción de sistemas electrónicos digitales a cualquier nivel de abstracción.
- Al modelar el sistema independientemente de la tecnología, puede ser sintetizado y realizado mediante diferentes soluciones con muy poco esfuerzo. Esto también implica que el diseño sea reutilizable.
- El VHDL proporciona un interfaz común entre las diferentes personas involucradas en el proceso de diseño.
- El uso de herramientas de síntesis automática permite al diseñador concentrarse más en los aspectos de la arquitectura que en la estructura.
- La probabilidad de error se reduce considerablemente así como el tiempo de diseño y el esfuerzo dedicado. Todo ello redunda en una mejora de la calidad y tiempo de puesta en el mercado.

Otro aspecto relevante del lenguaje es que tiene un grupo de trabajo para la estandarización en la IEEE. El lenguaje es un estándar de la IEEE. Este grupo es el P1076 con nombre VHDL Analysis and Standardization Group (VASG). En el año 2008, la IEEE incluye la versión VHDL 4.0 en su estándar IEEE 1076-2008 que fue publicado en enero del 2009. De cualquier forma es común que el software actual brinde la opción de especificar qué versión del lenguaje se emplea en los modelo (Ashenden and Lewis, 2008).

1.5.2 Características del lenguaje Verilog

Verilog: es un lenguaje de descripción de hardware usado para modelar sistemas electrónicos. El lenguaje, algunas veces llamado Verilog HDL, soporta el diseño, prueba e implementación de circuitos analógicos y digitales a diferentes niveles de abstracción. Un diseño en Verilog consiste de una jerarquía de módulos. Los módulos son definidos con conjuntos de puertos de entrada, salida y bidireccionales. Internamente un módulo contiene una lista de cables y registros. Las sentencias concurrentes y secuenciales definen el comportamiento del módulo, describiendo las relaciones entre los puertos, cables y registros (Oliver, 2007).

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

Verilog también está estandarizado como IEEE 1364 y su evolución llega a la actualidad con el estándar IEEE 1800-2009 que representa al System Verilog.

Conclusiones del capítulo

La definición y representación de los sistemas codificadores y decodificadores convolucionales, tiene mucho que ver con la electrónica digital secuencial. Varios autores han abordado el tema con fines prácticos y académicos. Las arquitecturas con hardware digital programable constituyen una variante donde se pueden aplicar estos mecanismos de corrección de errores, como parte de una aplicación mayor. Los lenguajes de descripción de

hardware VHDL y Verilog, existen como parte de cuerpos de estandarización e investigación y desarrollo de la IEEE, lo que garantiza su universalidad.

CAPÍTULO 2. CÓDIGOS CONVOLUCIONALES EN LENGUAJE DE DESCRIPCIÓN DE HARDWARE

Este capítulo está dedicado al establecimiento de una metodología de trabajo para realizar aplicaciones en FPGAs de Xilinx. También se describen algunas de las principales formas de generación automática de códigos VHDL y se hace una breve caracterización de la herramienta Xilinx ISE.

2.1 Metodología de diseño sobre Hardware Lógico Programable

Para realizar cualquier trabajo que tenga que ver con el desarrollo de un sistema digital es necesario seguir ciertos pasos y tener en cuenta distintos factores para que el diseño funcione correctamente y se termine en tiempo. Es muy importante seguir un método ordenado para lograr buenos resultados.

Los diseños implementados sobre FPGA pueden llegar a ser complicados, debido a las grandes capacidades que han alcanzado estos circuitos y el potencial de las herramientas de diseño que se encuentran disponibles. Para desarrollar en esta línea no necesariamente tiene que hacerlo una sola persona, sino que puede ser un equipo que se encuentre separado en varias localidades. Con una buena planificación y un trabajo ordenado es posible enfrentar el diseño de sistemas digitales complejos, por esto es necesario tener una metodología que guíe todo el proceso de diseño.

2.1.1 Flujo de diseño

Cuando se diseña con lógicas programables, cualquiera sea el método usado para diseñar el circuito (HDLs, esquemáticos, etc.), el proceso desde la definición del circuito hasta tenerlo funcionando sobre un FPGA implica varios pasos intermedios y en general utiliza una variedad de herramientas. A este proceso se le denomina ciclo o flujo de diseño. Para cada

uno de estos pasos se utilizan herramientas de software diferentes que pueden o no estar integradas bajo un ambiente de desarrollo. En algunos casos, las herramientas utilizadas en cada paso del diseño son provistas por diferentes empresas.

El proceso de diseño de un módulo hardware sobre una FPGA se divide en cuatro fases: descripción del modelo, síntesis, implementación y programación. Al finalizar cada una de las fases puede comprobarse la validez del diseño mediante simulación, utilizándose distintos tipos de simulación en cada una de ellas (Gil et al., 2005).

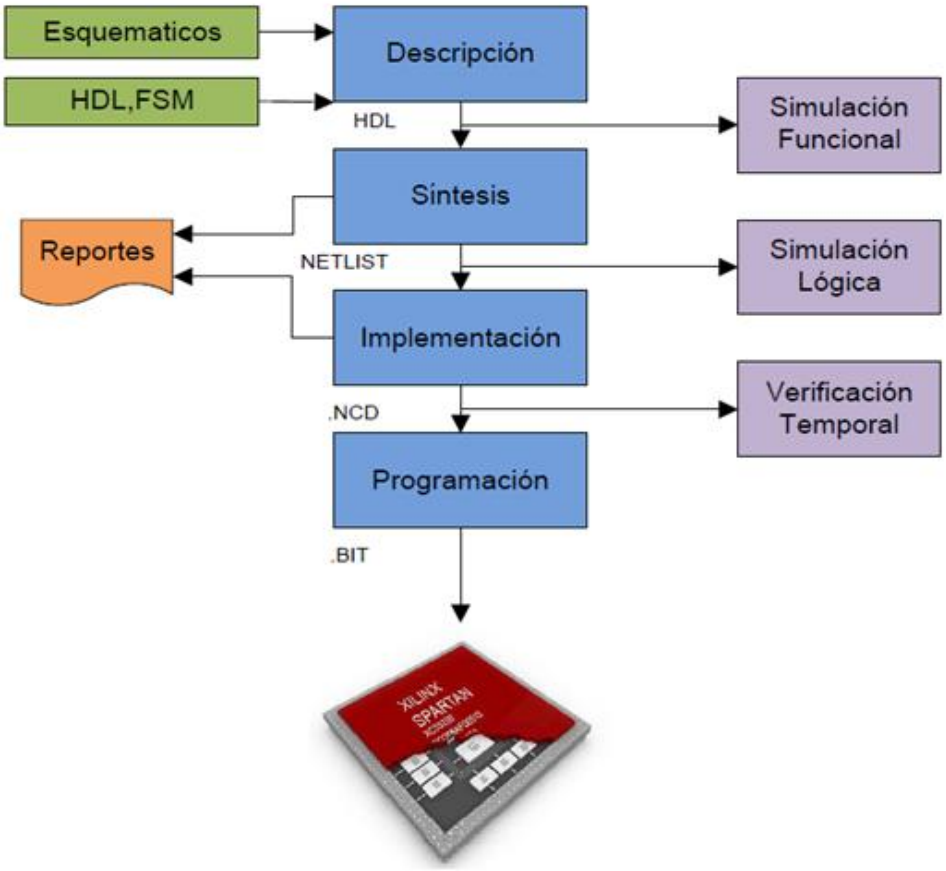


Figura 2.1 Flujo de Diseño para FPGA (PIÑEIROS, 2010).

Descripción del Modelo: este es el paso en el que se describe el diseño, muchas veces usando un lenguaje de descripción de hardware como el VHDL. Muchas herramientas permiten ingresar el diseño no solo como HDLs, sino también como un diagrama esquemático o estructural, una representación gráfica de una máquina de estados o una tabla de entrada-salida. Estas herramientas simplifican en gran medida el diseño y la tarea

del diseñador. El código HDL puede ingresarse utilizando cualquier editor de texto, pero se recomienda uno que tenga coloreado automático de sintaxis ya que ayuda y hace más fácil esta etapa (GÜICHAL, 2005).

Síntesis del Modelo: En este paso se traduce el VHDL original a su implementación con lógica digital, utilizando los componentes específicos del FPGA que va a utilizarse. Esta traducción puede llegar hasta el nivel más básico de elementos lógicos o hasta un nivel superior, en el que el diseño se presenta en módulos básicos estándar provistos en una librería por el proveedor del dispositivo (GÜICHAL, 2005).

Implementación del Modelo: En la implementación del modelo se realizan algunos procesos, primero se crea un único fichero de salida. El formato de este fichero de salida se denomina: NGD (Native Generic Design) y contiene componentes lógicos: puertas lógicas, biestables, RAMs, etc. A continuación se mapean los componentes lógicos en los componentes físicos de que dispone la FPGA: tablas LUT, biestables, buffers triestado, etc. Además se encapsulan estos componentes físicos en bloques configurables (CLBs e IOBs), creando un fichero de salida en formato NCD (Native Circuit Design). Para finalizar se pasa a realizar el posicionamiento e interconexión de los bloques configurables en el FPGA y se genera un fichero de lista de conexiones en formato NCD (PIÑEIROS, 2010).

Análisis de Tiempos: Una vez implementado físicamente el diseño, se procede a analizar los tiempos y velocidades de operación del mismo. Si los tiempos y velocidades no corresponden con los especificados en la etapa de descripción del diseño, deberá modificarse la descripción del circuito a fin de ajustar los mismos. Después de realizar estas fases se obtiene la información para realizar simulaciones temporales del diseño completo y se puede crear un archivo HDL estructural con base en los componentes físicos de la FPGA y un fichero SDF (Standart Delay Format) con los retardos internos del FPGA. Utilizando estos ficheros se puede realizar la verificación del diseño final mediante una simulación temporal HDL post-síntesis (PIÑEIROS, 2010).

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

2.2 Particularidades en la generación del código

Actualmente cualquier proceso de ingeniería dispone de un soporte software que asiste al ingeniero de aplicaciones o sistemas en el desarrollo de sistemas complejos. Los sistemas electrónicos reconfigurables del tipo FPGA son un buen ejemplo de la complejidad que se puede alcanzar. Esta complejidad 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.

Con el objetivo de agilizar y facilitar el proceso de diseño de hardware existen variantes para la generación automática de descripciones VHDL a partir de modelos gráficos. Varias herramientas de software, permiten obtener la descripción VHDL a partir de un diagrama de estados, incluso tipo ASM (*Algorithmic State Machine*) o a partir de un esquemático. Estas herramientas están directamente relacionadas con cada fabricante, por lo que cada una de ellas es diferente de otra.

Para crear un diseño que se fundamente en la realización de una aplicación sobre FPGA no es necesario crear el código VHDL desde cero. Existen varias formas de generar este código automáticamente y de forma sencilla, ahorrando tiempo y dando cierto grado de sencillez al proyecto.

2.1.1 Simulink HDL Coder

La empresa Mathworks oferta la herramienta Matlab/Simulink, esta es la base que sustenta otras herramientas de co-diseño bastante atractivas. Simulink es un paquete de software para modelar, simular y analizar sistemas dinámicos. Soporta sistemas lineales y no lineales: modelados en tiempo continuo, muestreados, o un híbrido de los dos. Los sistemas pueden ser también multifrecuencia, es decir, tienen diferentes partes que se muestrean o actualizan con diferentes velocidades.

Simulink proporciona una interfaz gráfica de usuario (GUI) para construir modelos como diagramas de bloques, con solo pulsar y arrastrar el ratón. Con esta interfaz, los modelos se representan de forma similar a como se dibujan con el lápiz y el papel, o como los representan la mayor parte de los libros de texto. Simulink incluye una amplia biblioteca de sumideros, fuentes, componentes lineales y no lineales y conectores. Además, es posible generar modelos jerárquicos usando un diseño descendente/ascendente. Se puede visualizar

el sistema en un nivel superior desde donde, mediante un doble clic sobre los modelos, se puede ir descendiendo a través de los niveles a fin de ver con más detalle el modelo. Esto proporciona una comprensión de cómo se organiza un modelo y cómo interactúan sus partes (Muro, 2004).

Para generar un código VHDL usando Simulink, MATLAB en su ayuda sugiere varios ejemplos como referencia. No todos los bloques de Simulink generan códigos VHDL así que, para realizar el diseño primeramente hay que conocer si los bloques que van a ser usados pueden ser convertidos en VHDL. MATLAB genera una librería mediante la línea de comandos *hdllib* donde muestra los bloques que generan lenguajes de descripción de hardware.

Después de verificar que los bloques se pueden usar en el proyecto, se puede dar el siguiente paso para la creación del código. Existen dos métodos fundamentales: usando la interfaz de línea de comandos de MATLAB y el método gráfico. Ambas tienen sus ventajas pero el resultado en general es el mismo, la decisión sobre cuál escoger es casi solo cuestión de costumbre.

Primeramente es necesario crear la carpeta de trabajo en un lugar conveniente para el proyecto, para hacerlo se escribe el comando *mkdir C: \Codigo*, en este caso se proporcionó la ruta "C:\Codigo" pero en realidad es a conveniencia de la persona que esté realizando el proyecto. A continuación hay que abrir la carpeta de trabajo mediante el comando *cd C:\Codigo* y el ambiente está listo para generar el código.

El paso siguiente será crear un nuevo modelo desde la ventana de Simulink y añadir los distintos bloques para conformar el sistema general. Desde esta ventana es posible hacer la simulación y comprobar que el sistema alcanza los resultados esperados.

Después de la comprobación del diseño mediante simulación se puede proceder a generar el código, para hacerlo hay que seleccionar la opción *tools* en la barra de herramientas, situarse sobre *HDL Code Generator* y abrir las opciones. Se inicia una nueva ventana donde aparece la opción de generar el código y hacer un Test Bench, esta se muestra en la Figura 2.2.

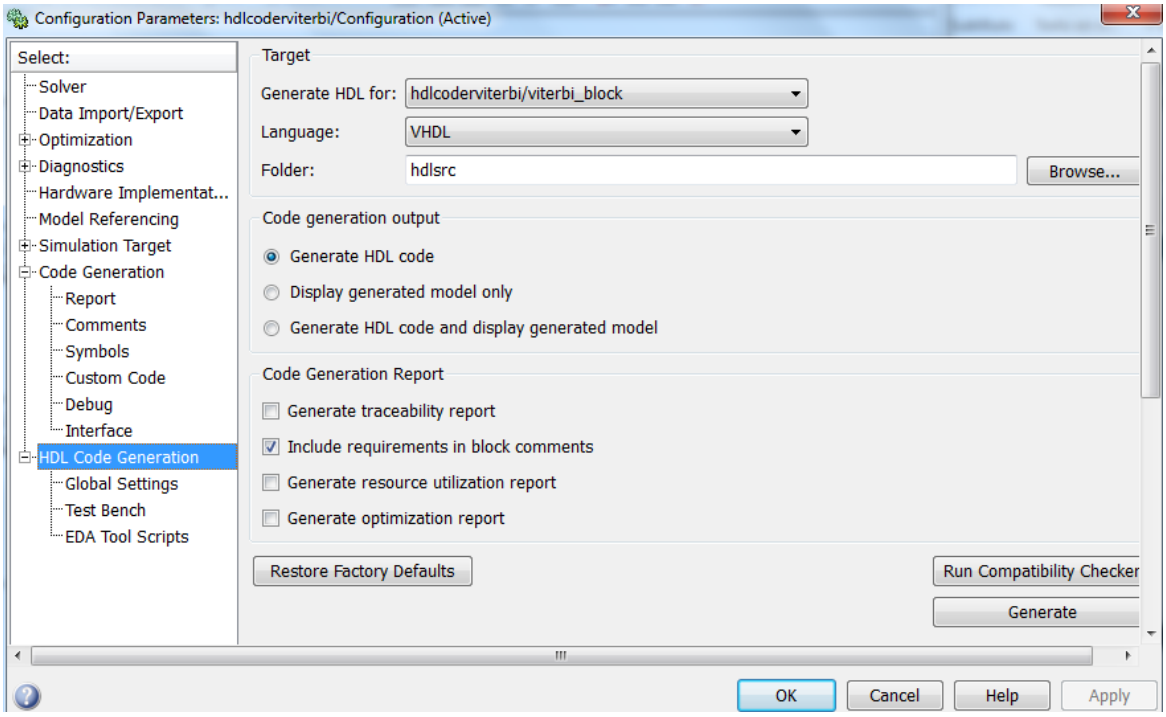


Figura 2.2 Ventana de configuración para generar el código HDL de manera gráfica.

2.2.2 System Generator

System Generator es una herramienta de diseño de alto nivel desarrollada por Xilinx que permite el diseño basado en modelos en el entorno Matlab/Simulink, para el desarrollo de sistemas de procesamiento digital en FPGAs (MATLAB, 2011).

System Generator se integra a Simulink como un toolbox al instalarse en la computadora Xilinx ISE. Esta herramienta permite un alto nivel de abstracción en el diseño que va a ser compilado para la FPGA.

Las librerías de propiedad intelectual que posee System Generator permiten la realización eficiente de funciones. En este sentido, no es necesario tener siempre conocimiento de los detalles de los recursos internos de la FPGA y aun así poder realizar un diseño eficiente incluyendo bloques como FFTs (Fast Fourier Transform).

System Generator genera de forma automática, partiendo de un modelo realizado en Simulink, una implementación basada en lenguajes de descripción de hardware (HDL). Para ello, el sistema convierte un conjunto de bloques adicionales, que extienden el programa Simulink, a una serie de “cores” optimizados por Xilinx.

Las principales características de esta herramienta se muestran a continuación:

- Permite representar a nivel de sistema diseños basados en FPGAs. Para ello utiliza una interfaz gráfica muy sencilla de usar, la del Simulink, y un conjunto de bibliotecas que encapsulan las funciones más usuales.
- Permite una generación automática de código en VHDL.
- Flujo de diseño optimizado, gracias a la conversión de los modelos a *cores* optimizados de Xilinx, llevada a cabo por el sistema CORE Generator.
- Ofrece una surtida biblioteca de modelos, mediante bloques parametrizables asociados a funciones aritméticas, lógicas o incluso DSPs (Procesamiento Digital de Señales).
- A la par de la generación del sistema, el programa genera también de forma automática, una serie de bancos de prueba (Test-Bench) y vectores de datos (Muro, 2004).

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

2.2.3 Xilinx CORE Generator

Xilinx CORE Generator es una herramienta que se instala con el programa ISE Xilinx. Permite la creación automática de códigos HDL a partir de una serie de *cores* que pueden ser seleccionados en la ventana IP Catalog.

Es preciso para el comienzo crear un nuevo proyecto donde se especifica la configuración general: nombre, carpeta de destino, FPGA que va a ser utilizada así como las características del kit de desarrollo, tipo de código (VHDL o Verilog), directorio temporal, etc.

La Figura 2.3 presenta la ventana de trabajo de Xilinx CORE Generator que muestra la función Convolution Encoder, presente en la sección Comunication & Networking, Error Correction. También es posible acceder a la información de esta versión del codificador a través de la página web del producto, cuyo enlace, y otros como el de la hoja de datos, aparecen en la ventana del codificador convolucional.

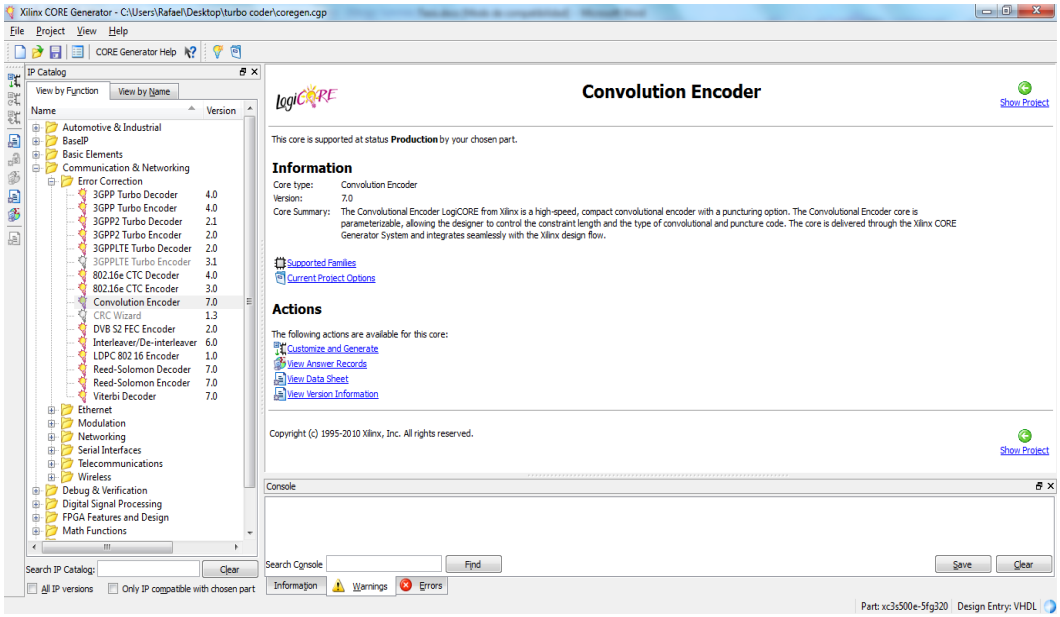


Figura 2.3 Xilinx CORE Generator.

Seleccionando *Customize and Generate* se abre una nueva ventana mostrada en la Figura 2.4 donde se seleccionan las características que deseamos como la razón de entrada, razón de salida y longitud de restricción, todas para el codificador convolucional que será generado en VHDL. Click en *Generate* y el código queda guardado en la carpeta de trabajo.

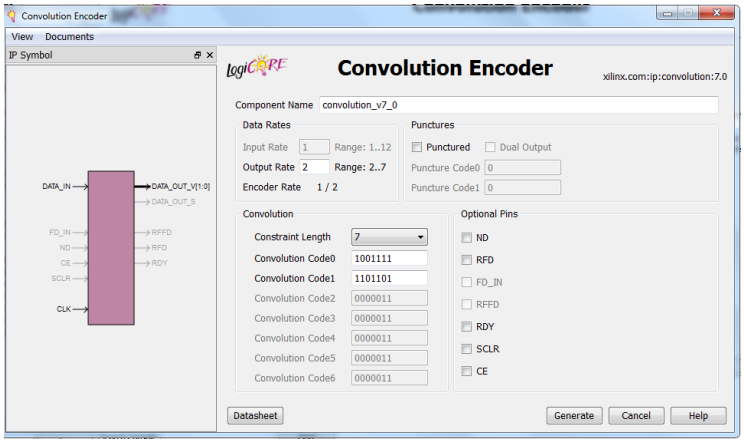


Figura 2.4 Configuración del Codificador Convolucional.

2.2.4 OpenCores

La página web www.opencores.org es una importante fuente de lenguajes de descripción de hardware. Existen varios autores que publican sus trabajos en este sitio y realizan actualizaciones cada cierto tiempo, además se muestra toda la información del proyecto como es el comentario profundo del código presentado, algunas veces, guía de usuario, guía de implementación, propiedades del proyecto, descripción, características, beneficios y en algunos casos también se brindan los nichos de pruebas. Se hace difícil en algunos casos implementar estos diseños porque dado el grado de actualización de los trabajos, las FPGAs requeridas son de fabricación reciente y por lo general difícil de conseguir. El sitio hospeda además a una gran comunidad de desarrolladores y ávidos del mundo del diseño digital.

Para descargar los códigos es necesario ser usuario de esta página web, por ello hay que registrarse, lo cual es gratis y los requisitos que se piden son mínimos. La búsqueda se puede hacer desde la pestaña de herramientas, se escribe el tema de interés y al ejecutar se abre una ventana similar a la del buscador Google donde hay que elegir el proyecto más conveniente en dependencia de lo que se aborda y descargar. En la Figura 2.5 se exhibe una vista del sitio www.opencores.com con el proyecto Viterbi Decoder (AXI4-Stream compliant) donde se puede acceder a todas sus características.

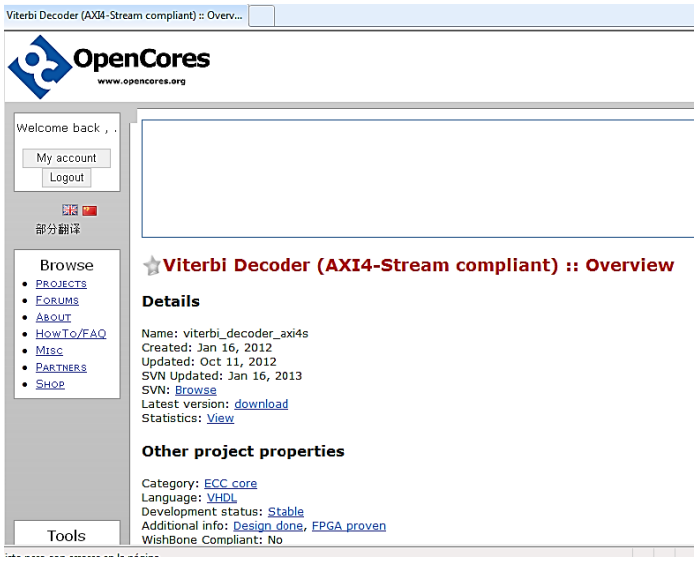


Figura 2.5 Vista del sitio OpenCores desde Internet Explorer.

2.3 Herramienta Xilinx ISE

Xilinx ISE (*Integrated Synthesis Enviroment*) es una herramienta de la compañía Xilinx para la síntesis de las aplicaciones de sistemas embebidos en un FPGA.

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

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

El entorno de ISE permite combinar las diferentes técnicas de diseño para facilitar la labor de descripción del diseño. Además, se admite la inclusión de restricciones para optimizar el proceso de implementación y adaptarlo a las necesidades del diseño (PIÑEIROS, 2010).

El *Proyect Navigator* o navegador de proyecto organiza los ficheros y corre procesos que permiten ir desde la entrada del diseño, pasando por la implementación y finalmente la programación de la tarjeta FPGA. Incluye todas las etapas del flujo de diseño basado en FPGA necesarias: entrada del diseño, síntesis, implementación, verificación y configuración del dispositivo. Posee también varias herramientas, de verificación, implementación y programación.

ISE combina diferentes técnicas de diseño para facilitar la labor de descripción del diseño, mediante un conjunto de herramientas que permiten el diseño de circuitos digitales como esquemas lógicos, máquinas de estado o utilizando lenguajes de descripción de hardware.

2.3.1 Descripción del entorno de desarrollo ISE

El entorno de desarrollo ISE se muestra en la Figura 2.6. Dicha interfaz es dividida en cuatro ventanas. La ventana diseño se usan para mostrar los archivos fuente del proyecto. La de procesos muestra todos los procesos necesarios para la ejecución de cada etapa de diseño. También se encuentra la ventana consola, donde se muestran los errores y advertencias, en sus paneles respectivos, así como la ventana de interfaz multi documentos

o ventana de trabajo, que se activa con los informes de diseño, archivos de texto, entre otros (Xilinx®, 2009).

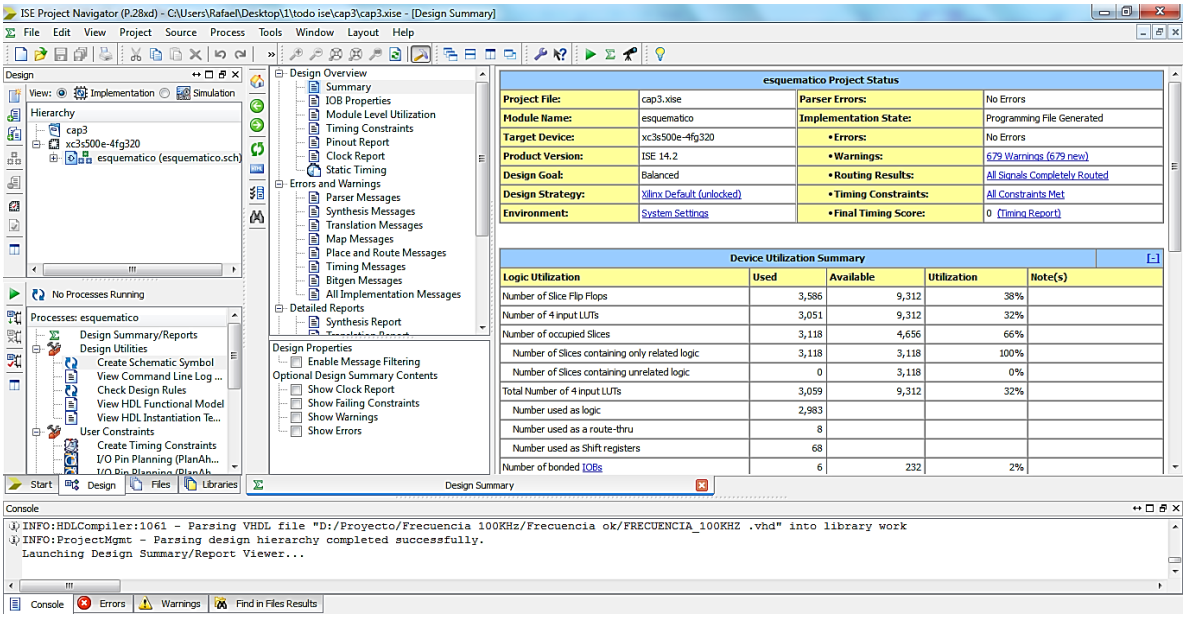


Figura 2.6 Ventana principal del entorno Xilinx ISE

2.3.1 Herramienta IMPACT

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

2.3.2 Herramienta PlanAhead

PlanAhead agiliza el paso entre la síntesis y el *place and route* para dar a los diseñadores un mayor control y una idea de cómo los diseños se aplican para lograr un menor número de iteraciones de diseño. La herramienta permite a los diseñadores utilizar una metodología de diseño jerárquico para minimizar la congestión de enrutamiento, simplificar la sincronización y la complejidad de la interconexión, y explorar las opciones de aplicación.

Conclusiones del capítulo

Existen diversos métodos para generar y obtener código de modelos. Si una herramienta de software de este tipo es del fabricante de la FPGA, es posible que la síntesis del diseño sea mucho más eficiente y optimizada. La mejor documentación al respecto del flujo de diseño con la FPGA de Xilinx, es del propio Xilinx. El entorno de desarrollo del Xilinx ISE es realmente impresionante en cuanto a las características que ofrece. No se pudo usar la variante de generación automática de codificadores/decodificadores convolucionales del Xilinx CORE Generator debido a la licencia; se presentaba como el mejor candidato. En su lugar se eligió el Simulink HDL Coder por su versatilidad y por contarse con la versión y licencia apropiadas.

CAPÍTULO 3. REALIZACIÓN DE UN CIRCUITO CON CÓDIGOS CONVOLUCINALES SOBRE FPGA

En este capítulo se genera un sistema corrector de errores en lenguaje VHDL mediante Simulink. Se presentan las simulaciones realizadas, las cuales confirman el correcto funcionamiento de la aplicación desarrollada. Además se muestra la aplicación corriendo en el Kit Nexys2.

3.1 Preparación de un sistema de corrección de errores en Simulink

El sistema de corrección de errores que se realiza en este trabajo consta de tres bloques fundamentales: generador PN, codificador convolucional y decodificador de Viterbi. El esquema que se utiliza se muestra en la Figura 3.1. De manera pragmática el sistema no tiene caso, pues no existe un canal que fundamente el propósito de utilizar códigos convolucionales, el objetivo real es aplicar el proceso de desarrollo sobre FPGA de manera que sí pueda ser usado dado el caso.

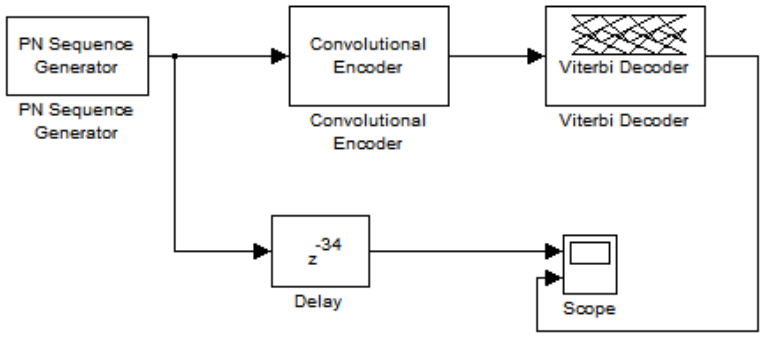


Figura 3.1 Esquema del sistema con corrección de errores en Simulink.

Al sistema se añade una demora (*Delay*) para eliminar el desfasaje entre la salida del generador de bits y el decodificador de Viterbi; así es posible observar la señal con más claridad cuando se monte el circuito real. El valor de la demora se tomó analizando la gráfica que muestra el osciloscopio cuando no se usa el bloque de demora (ver Anexo II).

La Figura 3.2 muestra las señales de salida del generador y decodificador de Viterbi en fase gracias al bloque de demora.

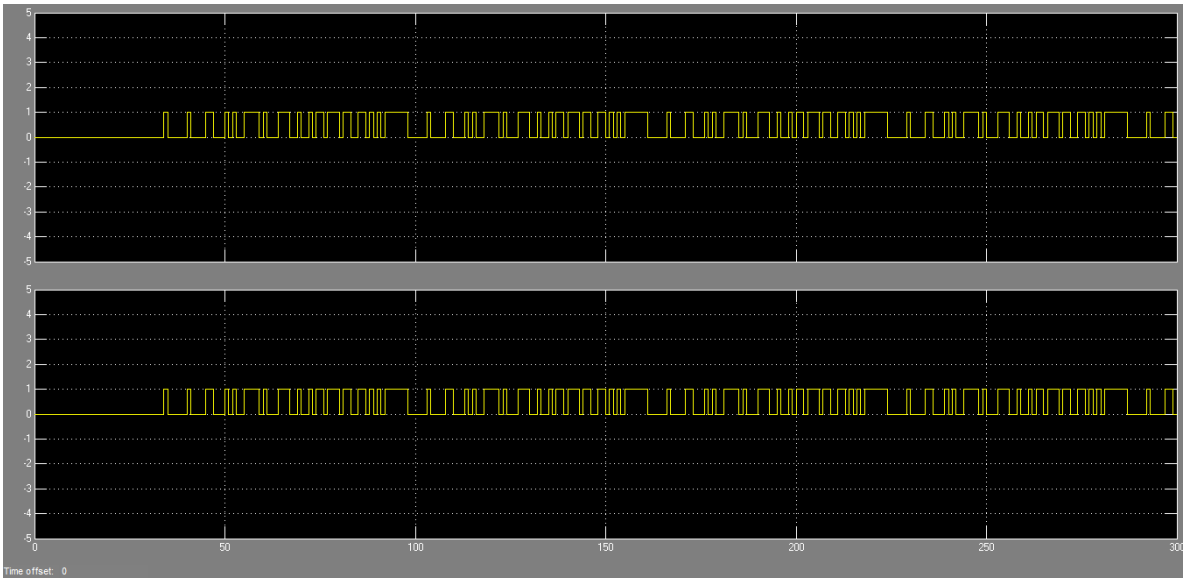


Figura 3.2 Señal en la salida del generador y decodificador de Viterbi.

3.1.1 Bloque del codificador convolucional

El bloque *Convolutional Encoder* codifica una secuencia de vectores binarios de entrada y produce una secuencia de vectores binarios a la salida. En la ventana *Function Block Parameters* se establecen los parámetros del codificador.

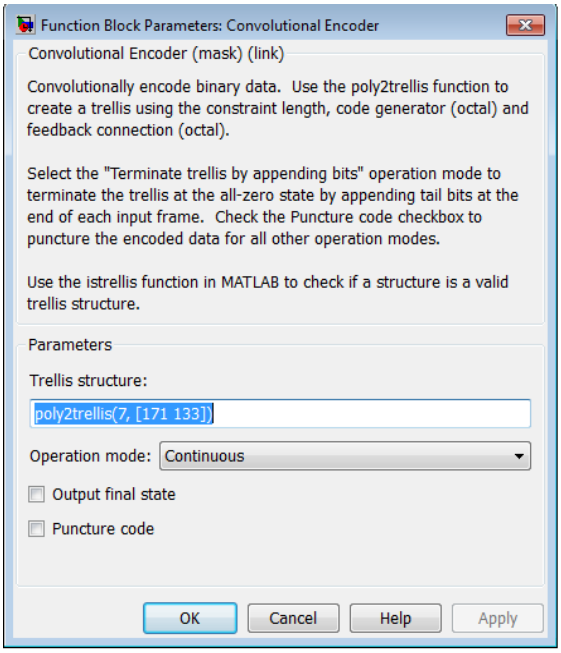


Figura 3.3 Parámetros del codificador convolucional.

En la Figura 3.3 aparece la función *poly2trellis* que puede recibir tres argumentos. El primero de los argumentos es un vector de la misma longitud que el vector de entrada, que establece la demora de las k entradas del flujo de bits. El segundo argumento es una matriz de $k \times n$ de números octales, que especifica las n conexiones de salida por cada bit de entrada. El tercero de los argumentos es para los codificadores convolucionales con realimentación y consiste en un vector de longitud k que especifica la conexión de realimentación para cada uno de los k bits de entrada. La estructura que se usa es poly2trellis(7,[171 133]) que viene configurada por defecto, dado que no hay interés práctico en el uso del código convolucional (MathWorks, 2011).

La opción *puncture code* es una técnica que elimina algunos de los bits de paridad después de la codificación. Esto tiene el mismo efecto que codificar con un código de corrección de errores con una tasa más alta, o menos redundancia (MathWorks, 2011).

3.1.2 Bloque Viterbi Decoder

El bloque Viterbi Decoder decodifica los símbolos de entrada y produce símbolos binarios de salida. Este bloque puede procesar varios símbolos a la vez y así actúa más rápido. En la Figura 3.4 se muestra la ventana de configuración del decodificador de Viterbi.

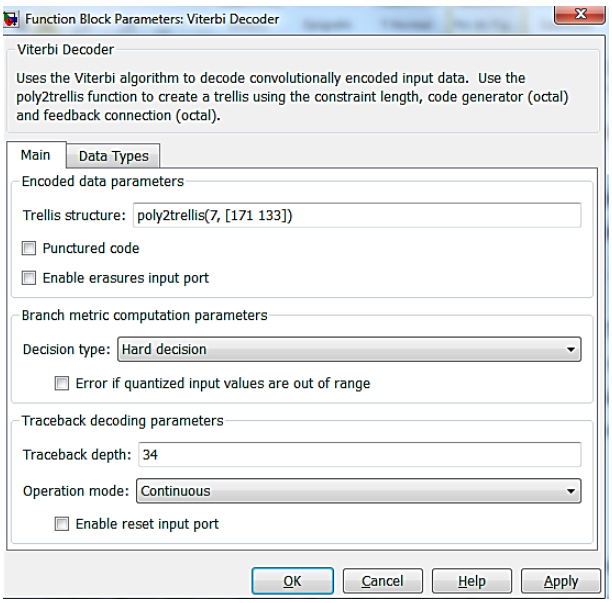


Figura 3.4 Parámetros de decodificador de Viterbi.

La opción *puncture code* solo puede ser usada cuando también se configura en el codificador.

Cuando se marca la casilla *Enable erasures input port*, en el decodificador se abre un puerto de entrada llamado *Era* que se marca en el bloque. A través de este puerto, se especifica un patrón de borrado de 1s y 0s, donde el 1s indica los bits para los cuales el decodificador no actualiza la métrica (MathWorks, 2011).

La sección *Decision type* tiene las opciones *unquantized*, *soft-decision* y *hard-decision*. El primer tipo calcula las métricas de rama mediante el algoritmo de Euclides, es decir, como una distancia euclidiana. Los otros dos tipos de decisión calculan las métricas de rama como una distancia de Hamming. En teoría de la información, un decodificador de decisión blanda es una clase de algoritmo que se utiliza para decodificar los datos que han sido codificados con un código corrector de errores. Mientras que un decodificador de difícil decisión opera sobre los datos que tienen un conjunto fijo de valores posibles (normalmente 0 o 1 en un código binario), las entradas a un decodificador de decisión flexible pueden adoptar toda una serie de valores en el medio. Es por ello que para el caso de estudio definido, basta con especificar el tipo *hard-decision.*

Traceback depht o la profundidad de rastreo, establece el número de ramas de trellis que se usan para construir cada ruta de rastreo (MathWorks, 2011).

Operation mode es un método para la transición entre tramas sucesivas de entrada: continuo, terminado, y truncado.

Enable reset imput port, cuando se marca esta casilla, en el decodificador se abre un puerto de entrada llamado *Rst*. Si se introduce un valor distinto de cero a *Rst* a este puerto hace que el bloque ajuste su memoria interna al estado inicial antes de procesar los datos de entrada.

En *Data Types* se elige el tipo de datos de la señal de salida, puede ser doble, individual, booleano, int8, uint8, int16, uint16, int32, uint32 (MathWorks, 2011).

3.1.3 Generación del código VHDL

Después de verificar que el diseño funciona correctamente se procede a la generación del código. Para esto se elimina el osciloscopio, se introducen dos bloques de salida, uno para

comprobar el dato que se va codificar y el otro para observar el dato decodificado y se hace un subsistema de cada bloque y, como se muestra en la Figura 3.5.

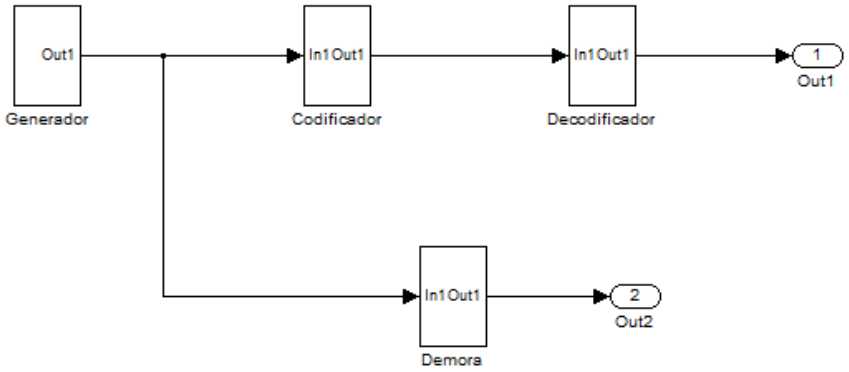


Figura 3.5 Sistema listo para la generación del código VHDL.

Se genera el código y Test Bench de cada bloque por separado, para simularlos en Xilinx ISE. Posteriormente, de todo el conjunto, este es el que va a ser sintetizado en la FPGA.

3.2 Simulación de cada bloque en Xilinx ISE

El primer paso es abrir un nuevo proyecto en Xilinx ISE. Una de las especificaciones necesarias consiste en definir el dispositivo que se va a usar. La Figura 3.6 muestra la ventana de selección del dispositivo.

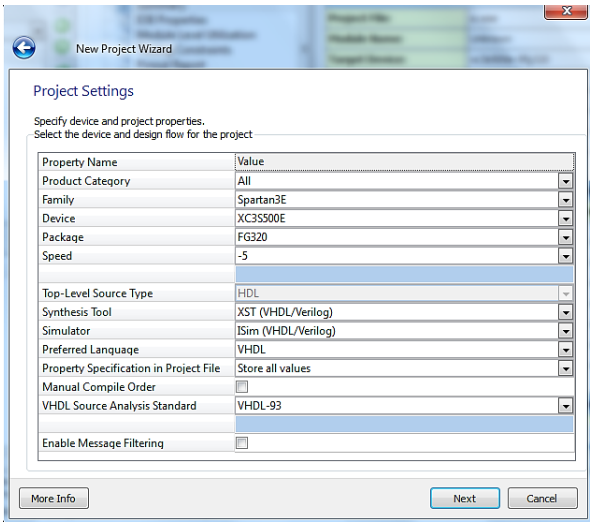


Figura 3.6 Selección del dispositivo y propiedades del proyecto.

Después de crear un nuevo proyecto, se añaden las fuentes del modelo del codificador o del decodificador. A este diseño se le pueden aplicar un conjunto de herramientas, de las cuales una de las más comunes es el chequeo de sintaxis. Tratándose de un código generado automáticamente, el resultado no es menos que el esperado: una sintaxis correcta. Esta es una de las ventajas de usar los generadores de código.

Con la adición del Test Bench del codificador se procede a la simulación. La Figura 3.7 muestra el resultado.

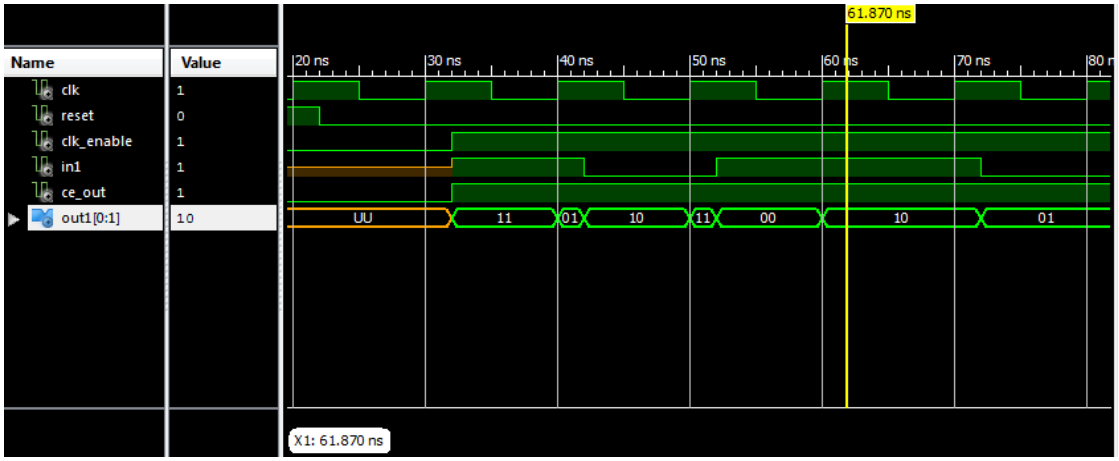


Figura 3.7 Resultado de la simulación del codificador convolucional.

En el momento que *clk-enable* toma el valor de uno y *reset* el valor cero, además de existir un dato a la entrada comienza a aparecer un dato a la salida. Este dato depende de la señal de reloj y del dato de entrada, cada vez que se produzca una subida del borde de la señal de reloj se alcanza un nuevo dato de salida. Si no existe un borde de subida de la señal *clk* pero el dato de entrada cambia, también se genera un nuevo dato a la salida. Existe la posibilidad de que ante una subida de reloj el dato a la salida se mantenga igual, en realidad es la salida de otro estado, pero este puede coincidir con el anterior y da la impresión de que no cambió.

Después de simular el decodificador de Viterbi se obtienen los resultados mostrados en la Figura 3.8. En este caso el proceso de decodificación, al igual que en la codificación, ocurre cuando *clk_enable* está en el nivel alto y *reset* en el nivel bajo. Cada vez que hay una subida de la señal *clk*, teniendo en cuenta el dato que hay a la entrada del decodificador, se obtiene un dato a la salida.

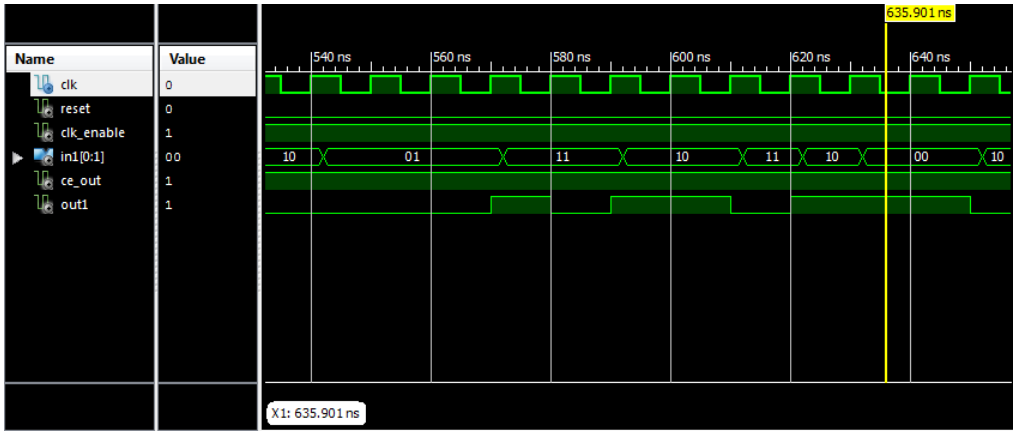


Figura 3.8 Resultado de la simulación del decodificador de Viterbi.

3.3 Implementación del sistema de corrección de errores

Una vez examinado cada bloque por separado y obtenido resultados satisfactorios, se procede cargar una nueva fuente que es un bloque que contiene en su interior el conjunto generador-codificador-decodificador que se generó en Simulink y sintetiza en un solo bloque el generador PN, codificador convolucional, decodificador de Viterbi y demora. Se cuenta con un bloque llamado FRECUENCIA_100KHZ el cual fue programado en VHDL como un contador, con un comportamiento de salida divisor de la señal de entrada. En realidad el código estaba disponible en los materiales de la asignatura de electrónica digital II y se modificó para el uso en este proyecto. Su objetivo es disminuir la frecuencia de trabajo del dispositivo FPGA y obtener una señal más fácil de percibir a la salida, pues hay que recordar que el kit de desarrollo Nexys2 tiene un reloj de 50MHz. El paso siguiente es crear dos símbolos: uno con la descripción VHDL del Generador_Codificador_Decodificador y otro con FRECUENCIA_100KHZ. Estos símbolos que se crean, pasan a formar parte de la librería de componentes del proyecto, podrán utilizarse como un dispositivo más. En el Anexo IV se muestra la fase de creación de un símbolo.

En la Figura 3.9 se muestra el sistema de corrección de errores que se sintetizó en la FPGA.

Para comprobar el funcionamiento del bloque Generador_Codificador_Decodificador se genera un Test Bench y se realiza una simulación. Las dos señales de salida están desfasadas, algo que existía en Simulink propio de la propagación de la señal y que se resolvió con un bloque de demora. En el anexo III se muestra el resultado de esta

simulación que nuevamente está desfasada. El asunto de simultanear las señales solo tiene caso para demostrar en este proyecto que los sistemas convolucionales funcionan correctamente y que la señal a la entrada es obtenida nuevamente a la salida. En un caso real, esto no tiene ninguna utilidad práctica.

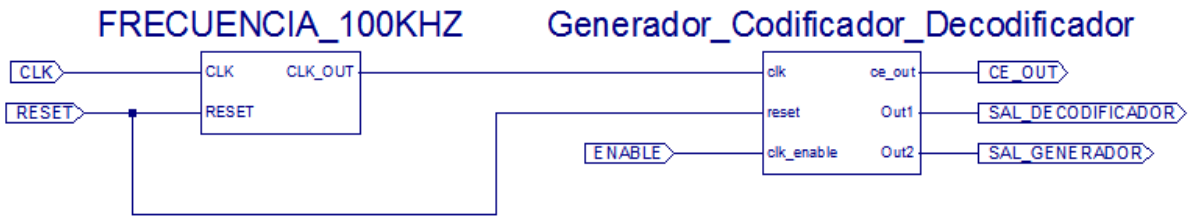


Figura 3.9 Sistema de corrección de errores en Xilinx ISE.

Entonces se hizo un análisis y se llegó a la conclusión de que en Simulink el tiempo que demoran los datos para ser codificados y decodificados es diferente al de Xilinx ISE, lógicamente porque se trata de motores de software diferentes. El problema consiste en determinar la demora a utilizar en Simulink para generar el código y que no exista desfasaje de las señales de salida en Xilinx ISE. Después de hacer dos pruebas se obtiene que una demora de una y dos unidades en Simulink equivale a 530ns y 520ns en Xilinx ISE respectivamente. Se llegó a la conclusión de que hace falta una demora en Simulink de valor igual a 54 y quedó resuelto el problema del desfasaje.

Con el valor de la demora calculado se regresa al Simulink y se genera nuevamente el código VHDL para realizar la simulación en Xilinx ISE del bloque rectificado. En la Figura 3.10 se muestra el resultado de esta simulación donde las señales están en fase.

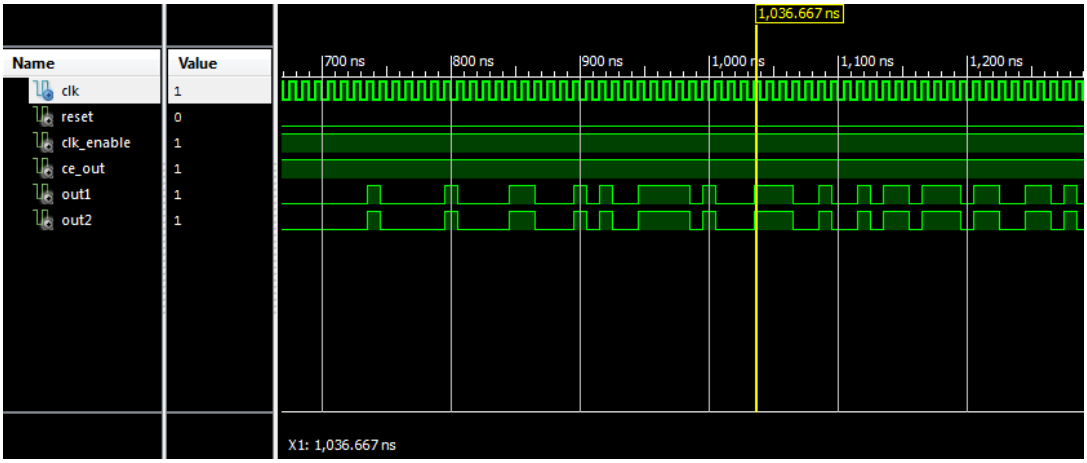


Figura 3.10 Salidas del generador y decodificador en fase. Como era de esperarse, la secuencia de código es idéntica.

Para continuar la metodología de diseño se procede a la asignación de pines mediante la opción Floorplan Area/IO/Logic (PlanAhead). Esta abre la herramienta PlanAhead 14.2 mediante la cual se puede asignar cada pin a las entradas y salidas. Después se realiza la síntesis del diseño usando la opción mostrada en el Anexo VI. Cuando se termina la síntesis llega el paso final donde se elige *Generate Programing File* y queda en la carpeta donde se guardó el proyecto, el archivo *.bit* que es el que se usa para programar la FPGA.

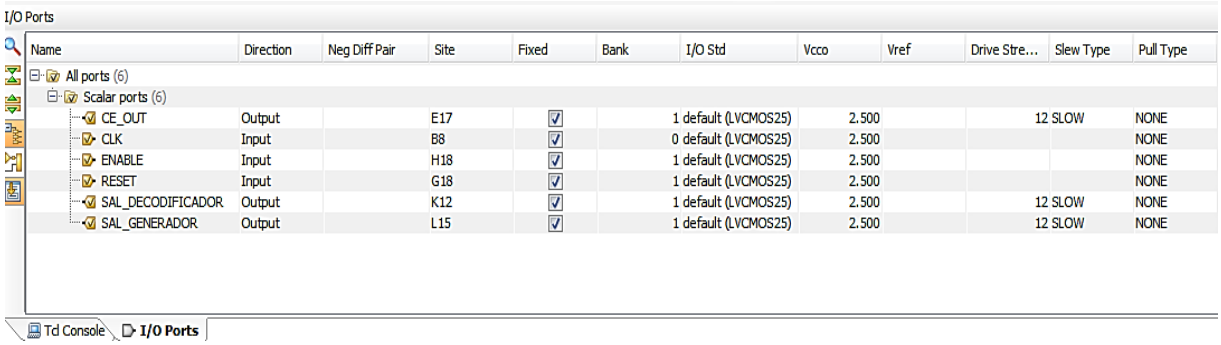


Figura 3.11 Asignación de pines.

3.3.1 Configuración de la tarjeta Nexis 2

Para programar la tarjeta se transfiere un fichero con extensión *.bit* a las celdas de memoria de la FPGA para definir las funciones lógicas y la interconexión de los circuitos.

La Figura 3.12 muestra el circuito de programación de la tarjeta.

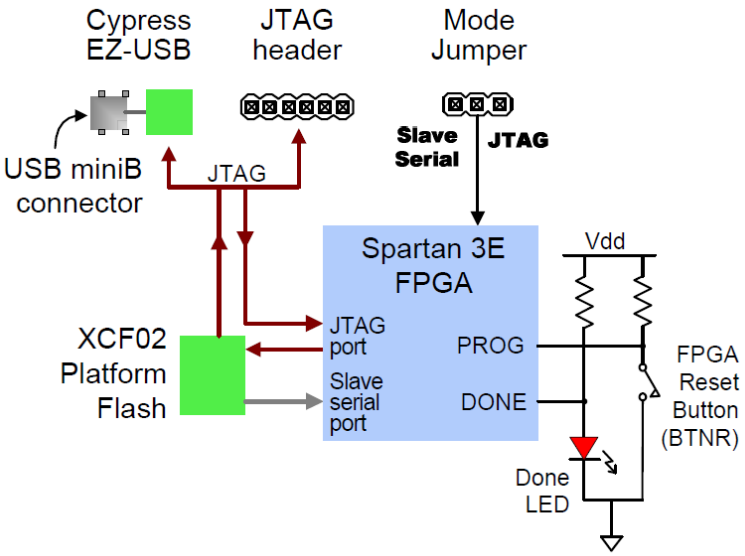


Figura 3.12 Circuito de programación de la tarjeta.

Para configurar la FPGA se usa el puerto USB conectado a una PC y solo se demanda un cable USB de la PC a la tarjeta y un software para cargar el fichero. En este caso se usa el software Digilent Adept, cuyo ambiente se muestra en la Figura 3.13. Es importante aclarar que con las versiones recientes del propio Xilinx ISE es posible programar la tarjeta, debido a que le fueron integrados los controladores USB apropiados para esto. De cualquier forma, se hizo con el Digilent Adept por su fácil utilización y porque está muy bien documentado.

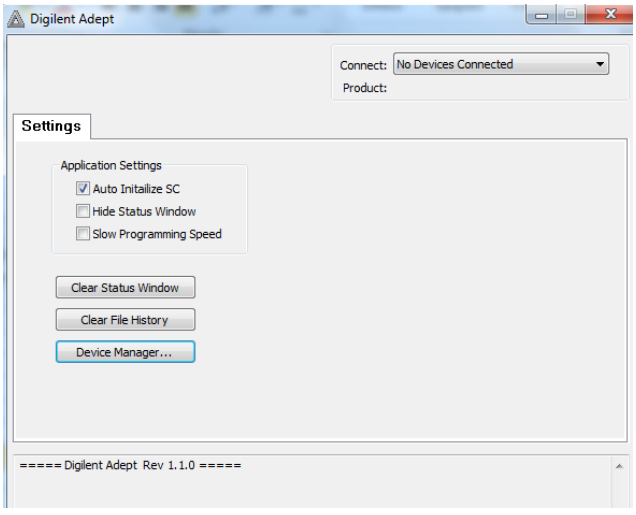


Figura 3.13 Ambiente de trabajo del software Digilent Adept.

3.3.2 Comprobación de funcionamiento del sistema

Después de implementado el sistema en el kit Nexys2 y conectado al osciloscopio por los puertos JA1:L15 y JA2:K12, se pudo comprobar el correcto funcionamiento de la aplicación como se muestra en la Figura 3.14. El módulo de periféricos (Pmod) JA de 6 pines, es el más cercano al conector de alimentación.

De estos 6 pines, uno es tierra (GND) y otro es fuente (Vdd). Los cuatro restantes son señales para datos, que electrónicamente están protegidas por resistencias contra cortocircuitos y por una estructura con diodos Zener contra descargas electrostáticas o ESD. Un selector (*jumper*) permite colocar el valor de la fuente del puerto Vdd a 3.3V o al bus de alimentación de entrada (VU). Si la fuente del puerto es puesta al bus de alimentación de entrada y es el USB quien lleva el bus principal de alimentación, se debe tener especial cuidado de que la aplicación conectada al Pmod no consuma más de 200mA. Otro aspecto que tiene que ver con la seguridad en la utilización del kit Nexys2, es que se tienen que evitar conflictos con fuentes externas, pues con el selector puesto en VU se pueden colocar fuentes regentes a través del propio Pmod. En este trabajo se utilizó la fuente interna de 3.3V.

Figura 3.14 Sistema de corrección de errores implementado sobre el kit Nexys2.

En la Figura 3.15 se muestran las señales a la salida del generador y del decodificador. Como se puede observar hay perfecta correspondencia entre las dos señales. La correspondencia no deja apreciar claramente las señales por separado.

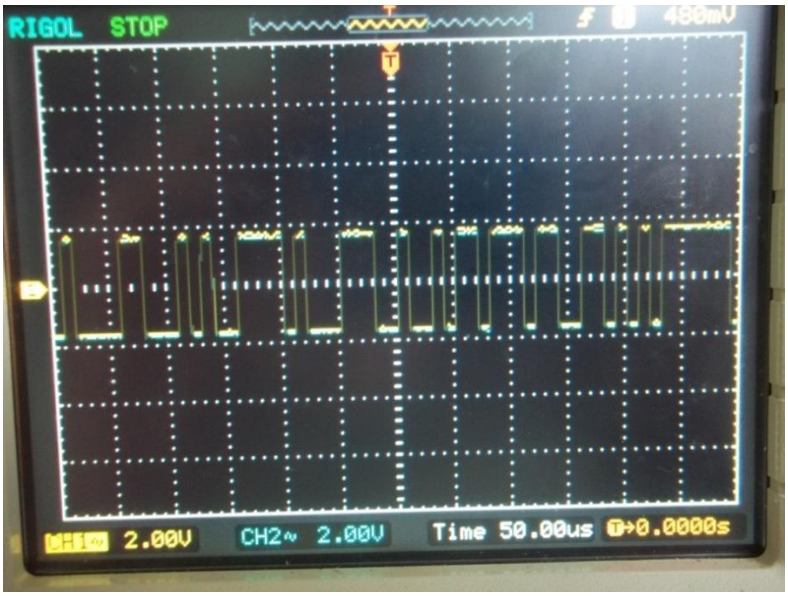


Figura 3.15 Señales reales del sistema de corrección de errores.

En la Fig. 3.16 se desplaza la señal 1, que es la salida del decodificador, para que sea posible la comparación con la señal 2, que representa la salida del generador. Se puede observar que las señales se corresponden.

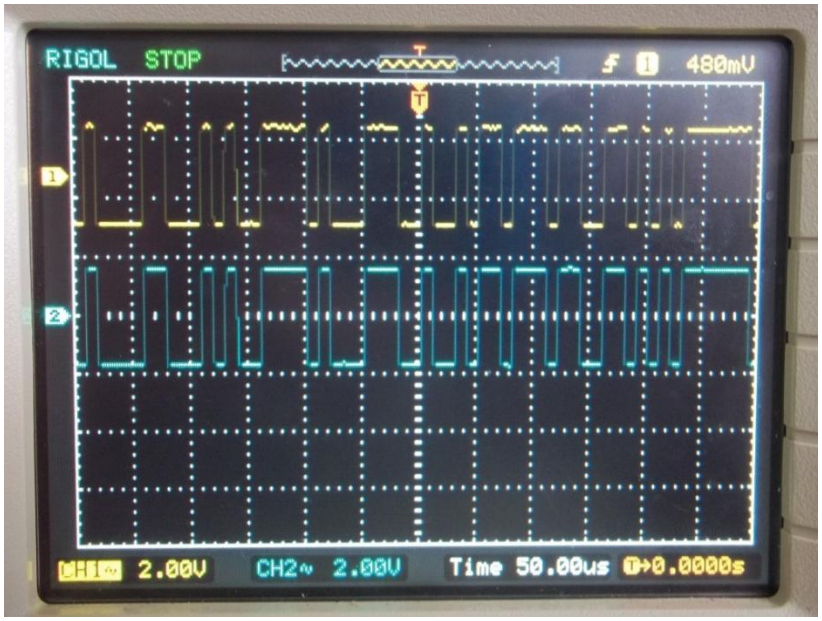


Figura 3.16 Señales reales del sistema de corrección de errores.

El consumo de recursos en la tarjeta FPGA para esta aplicación se muestra en la Fig. 3.17.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	3,586	9,312	38%	
Number of 4 input LUTs	3,051	9,312	32%	
Number of occupied Slices	3,118	4,656	66%	
Number of Slices containing only related logic	3,118	3,118	100%	
Number of Slices containing unrelated logic	0	3,118	0%	
Total Number of 4 input LUTs	3,059	9,312	32%	
Number used as logic	2,983			
Number used as a route-thru	8			
Number used as Shift registers	68			
Number of bonded IOBs	6	232	2%	
Number of BUFGMUXs	2	24	8%	
Average Fanout of Non-Clock Nets	3.73			

Figura 3.17 Sumario de la utilización del dispositivo.

Conclusiones del capítulo

Para hacer funcionar el sistema en la FPGA, y además comprobar su correcto funcionamiento, fue necesario llevar a cabo un diseño con ciertas decisiones imprácticas, pero justificadas dado el objetivo perseguido. Fueron generados los modelos con el Simulink HDL Coder e insertados en el Xilinx ISE con éxito. El empleo de demoras permitió representar las señales de manera tal que permita compararlas.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

Con la realización de este trabajo se desarrolló un sistema de corrección de errores en Simulink y se comprobó el funcionamiento del mismo en una FPGA Spartan 3E de Xilinx. Esto ha permitido arribar a las siguientes conclusiones y recomendaciones:

1. Los códigos convolucionales son un mecanismo de corrección de errores con un alto nivel de utilización en los sistemas de comunicaciones digitales. En la actualidad existe una tendencia a unir este tema a los dispositivos lógico programables por la versatilidad que estos ofrecen.
2. La metodología que se obtuvo es de carácter general y puede emplearse para cualquier diseño que se realice en las FPGAs de Xilinx. Explica de forma detallada los pasos que se deben seguir desde que se concibe un diseño determinado hasta tenerlo funcionando en la tarjeta.
3. La herramienta de generación Xilinx CORE Generator se presentó como un importante candidato para obtener el código automáticamente, pero la licencia no permitía el trabajo a fondo con el modelo generado. En su lugar se utilizó Simulink HDL Coder por su versatilidad.
4. El sistema presentado funcionó correctamente, lo cual significa que los modelos obtenidos pueden ser reutilizados o reobtenidos con otras variaciones, para su uso en aplicaciones que lo demanden.

Recomendaciones

1. Profundizar en el estudio de los códigos convolucionales y su fusión con los dispositivos lógico programables.
2. En cuanto se pueda, obtener los modelos con el Xilinx CORE Generator y comparar los resultados.

REFERENCIAS BIBLIOGRÁFICAS

ALNEEMA, D. D. A. & QASSIM, Y. T. 2009. FPGA Based Implementation of Convolutional Encoder- Viterbi Decoder Using Multiple Booting Technique.

ARNONE, L. J. 2008. *Transmisión segura en comunicaciones inalámbricas de corto alcance.*

ASHENDEN, P. J. & LEWIS, J. 2008. *VHDL-2008 Just the new stuff,* Boston, Morgan Kaufmann Publishers, Elsevier.

CASTILLO, A., VÁZQUEZ, J., ORTEGÓN, J. & RODRÍGUEZ, C. 2008. Prácticas de laboratorio para estudiantes de ingeniería con FPGA. *IEEE Latin Am Trans,* 6**,** 130-136.

COSTELLO JR, D. J., PUSANE, A. E., BATES, S. & ZIGANGIROV, K. S. A comparison between LDPC block and convolutional codes. Proc. Information Theory and Applications Workshop, 2006. 6-10.

CHAUDHRY, A. A. & MAHBOOB, R. FPGA implementation of 4-bit soft input viterbi decoder for satellite telemetry system. International Bhurban Conference on Applied Sciences & Technology, 2011 Islamabad, Pakistan. SUPARCO, 74-77.

FRANCO, M. S. 2006. *Diseño e implementación de los turbo codificadores definidos en los estándares de telecomunicaciones CDMA2000 (TIA/EIA2002.2d) y WCDMA (3GPP ts 25.212 v7.2.0) usando hardware reconfigurable.* M. Sc., UNIVERSIDAD DEL VALLE.

GEA, S. E. 2009. Códigos convolucionales.

GIL, R. M., LORENZO, I. F. & SÁNCHEZ, P. M. 2005. Diseño en VHDL para FPGAs.

GÜICHAL, G. 2005. *Diseño Digital Utilizando Lógicas Programables* Universidad Tecnológica Nacional Facultad Regional Bahía Blanca.

KAUR, P. 2006. *Implementation Of Low Power Viterbi Decoder On FPGA.* M. Sc., DEEMED UNIVERSITY.

MATHWORKS 2011. Product Help.

MATLAB 2011. System Generator for DSP user guide.

MURO, J. S. 2004. Nuevas tendencias en el diseño electrónico digital: codiseño Hardware/Software.

OLIVER, J. P. 2007. *Diseño Digital Utilizando Lógica Programable:*

Aplicaciones a la Enseñanza. Universidad de la República.

PHU, B. 2010. FPGA implementation of a Viterbi decoder for next generation broadband wireless acces systems.

PIÑEIROS, G. D. F. 2010. *Implementación de la etapa de recepción de un sistema de comunicaciones utilizando la tecnología FPGA.* SANGOLQUÍ.

SKLAR, B. 1988. *Digital Comunications Fundamentals and Applications,* Tarzana, California, Prentice Hall.

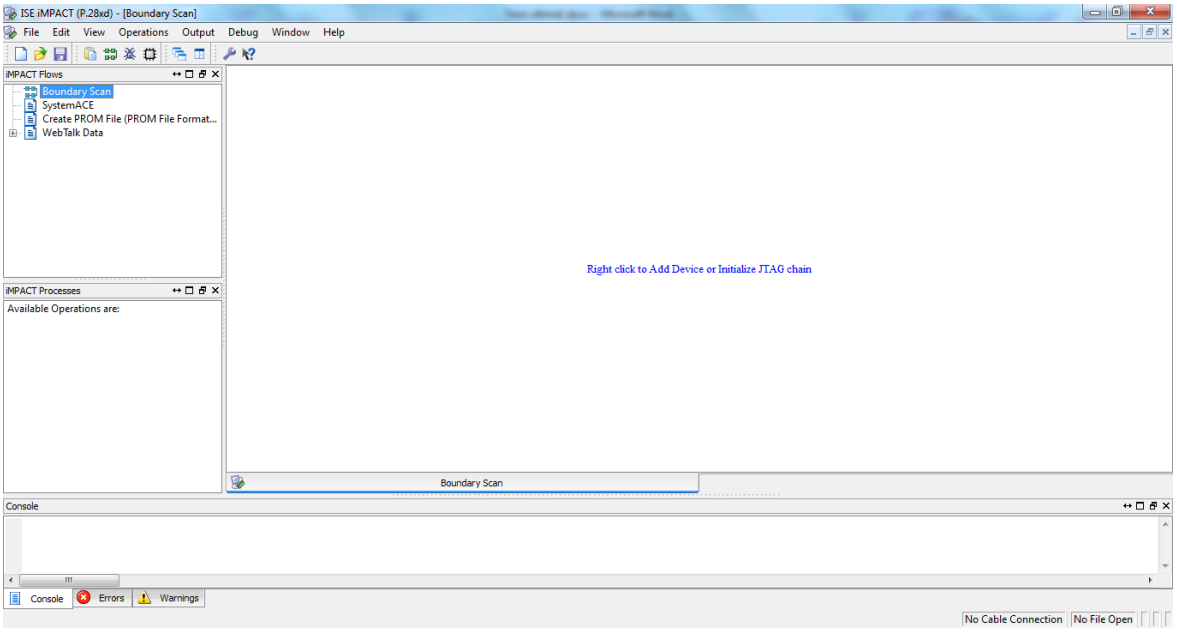
XILINX® 2009. ISE In-DepthTutorial.

XU, Z., REN, J., WANG, X.-J. & YE, F. Implementation of folded sliding block Viterbi decoders for MB-OFDM UWB communication system. Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on, 2007. IEEE, 2574-2577.

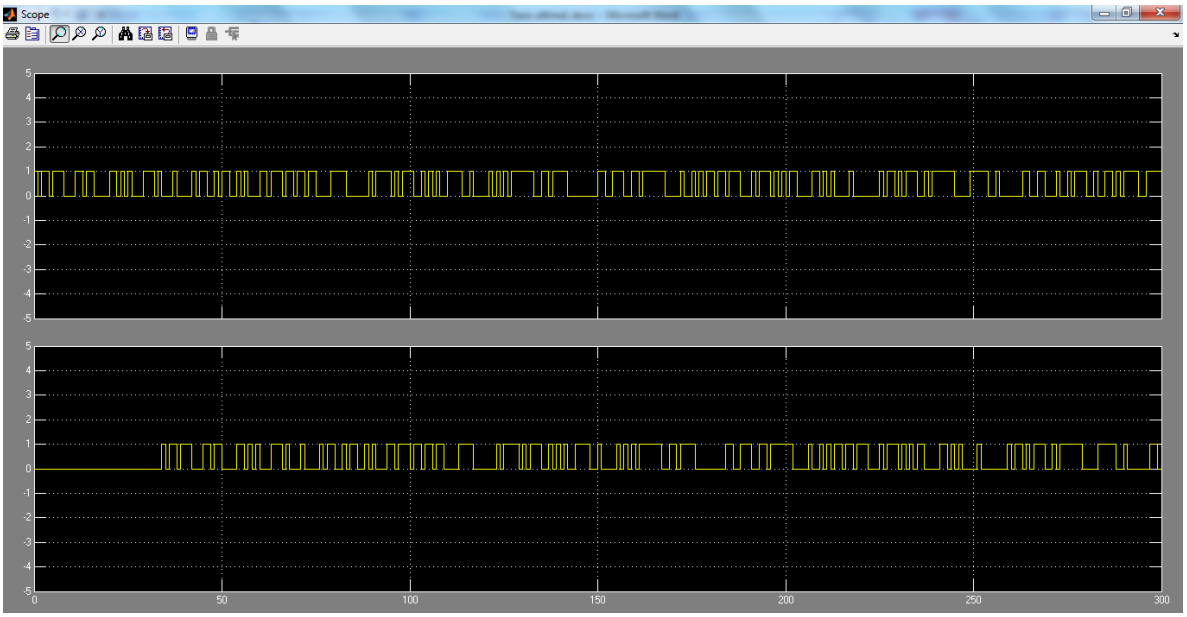
YANYAN, L., XIAOHUI, Y. & DIANREN, C. 2012. The Design of Efficient Viterbi Decoder and Realization by FPGA.

ANEXOS

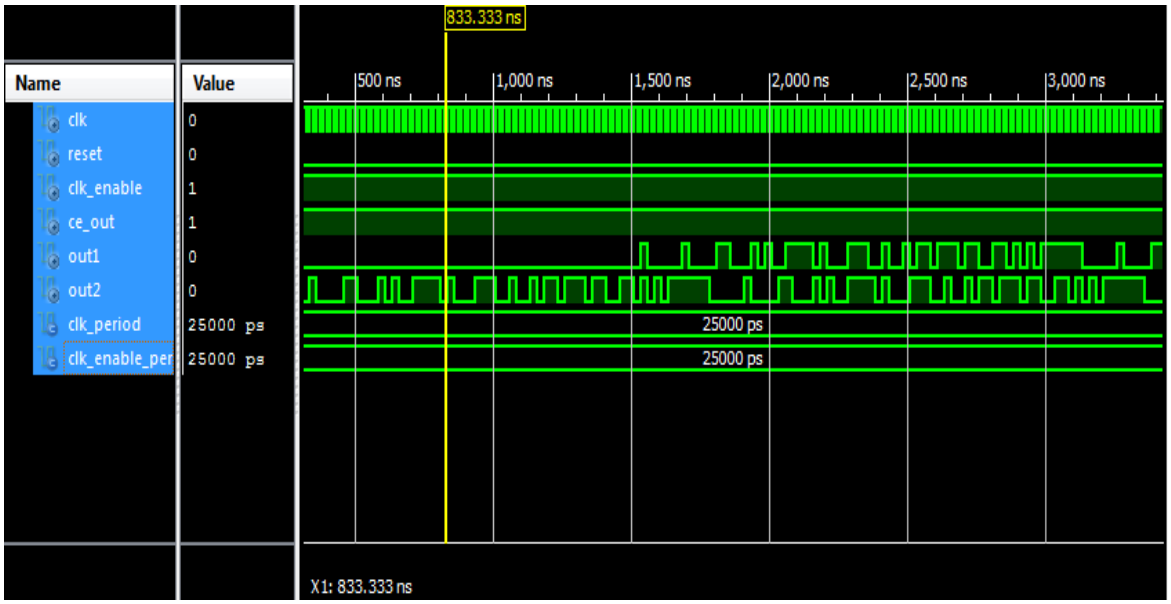
Anexo I Ventana principal de IMPACT



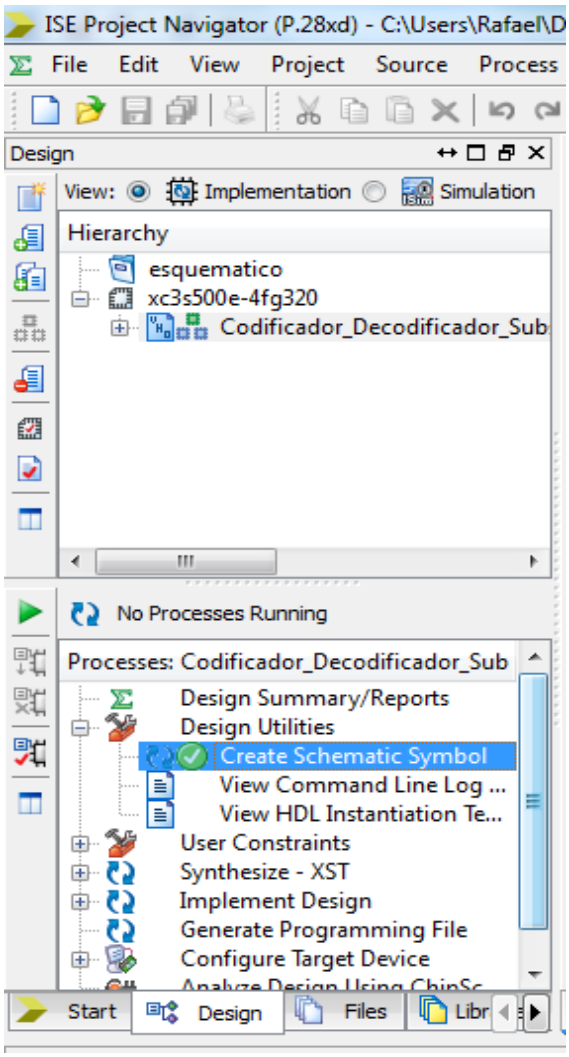
Anexo II Simulación en Simulink del sistema corrector de errores



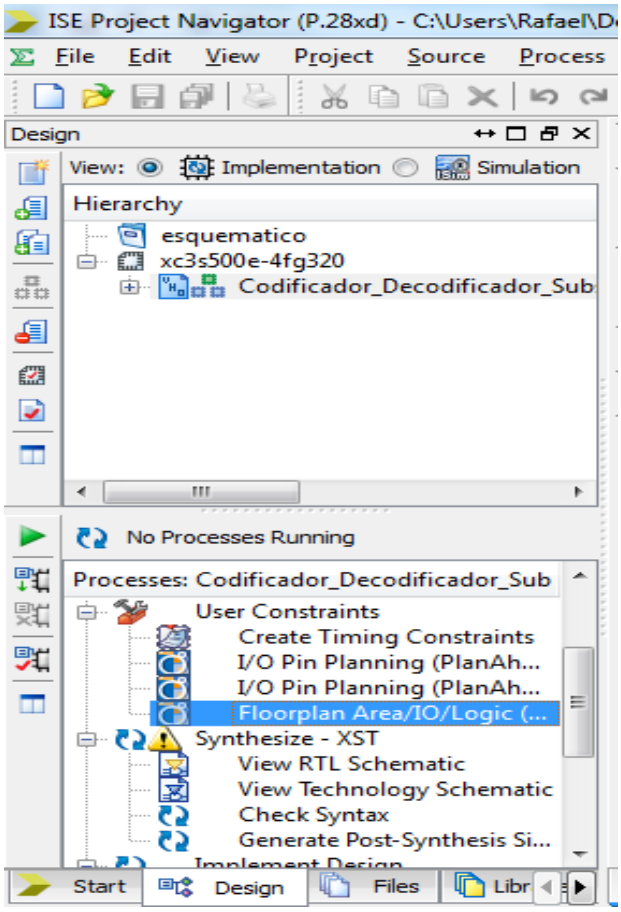
Anexo III Simulación de las señales de salida con desfasaje



Anexo IV Creación de un símbolo



Anexo V Inicio de la herramienta para la asignación de pines



Anexo VI Implementación del diseño