UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA DE
TELECOMUNICACIONES
DEPARTAMENTO DE SEÑALES Y SISTEMAS

DESARROLLO DE ALGORITMOS DE CODIFICADORES CONVOLUCIONALES USADOS EN ESTÁNDARES DE COMUNICACIONES INALÁMBRICAS BASADO EN LA TARJETA FPGA SPARTAN 6

MARTÍNEZ C. DANIEL A.
SÁNCHEZ L. JONÁS

Bárbula, 26 de Julio del 2015

UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA DE TELECOMUNICACIONES
DEPARTAMENTO DE SEÑALES Y SISTEMAS

DESARROLLO DE ALGORITMOS DE CODIFICADORES CONVOLUCIONALES USADOS EN ESTÁNDARES DE COMUNICACIONES INALÁMBRICAS BASADO EN LA TARJETA FPGA SPARTAN 6

TRABAJO ESPECIAL DE GRADO PRESENTADO ANTE LA ILUSTRE UNIVERSIDAD DE CARABOBO PARA OPTAR AL TÍTULO DE INGENIERO DE TELECOMUNICACIONES

MARTÍNEZ C. DANIEL A.
SÁNCHEZ L. JONÁS

Bárbula, 26 de Julio del 2015

UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA DE TELECOMUNICACIONES
DEPARTAMENTO DE SEÑALES Y SISTEMAS

CERTIFICADO DE APROBACIÓN

Los abajo firmantes miembros del jurado asignado para evaluar el trabajo especial de grado titulado «DESARROLLO DE ALGORITMOS DE CODIFICADORES CONVOLUCIONALES USADOS EN ESTÁNDARES DE COMUNICACIONES INALÁMBRICAS BASADO EN LA TARJETA FPGA SPARTAN 6», realizado por los bachilleres MARTÍNEZ C. DANIEL A., cédula de identidad 19.604.303, SÁNCHEZ L. JONÁS, cédula de identidad 19.573.867, hacemos constar que hemos revisado y aprobado dicho trabajo.

Firma
Prof. ELIMAR HERNANDEZ
TUTOR

Firma
Prof. DEMETRIO REY LAGO
JURADO

Firma
Prof. ANTONIO S. FEDÓN
JURADO

Bárbula, 26 de Julio del 2015

Dedicatoria

A Dios

A mis padres Edwin Martínez y Sabina Castañon

A mis hermanos Carla, Edwin y Romina

A todas aquellas personas que de alguna u otra forma estuvieron presentes y me ayudaron en la formación como ingeniero a lo largo de mi carrera

MARTÍNEZ C. DANIEL A.

A Dios

A mis padres Gilberto Sánchez y Susana López

A mis hermanos Josué, Jonathan y David

A todas esas personas que me ayudaron a consumar este proyecto

SÁNCHEZ L. JONÁS

Agradecimientos

Agradezco a Dios por darme la fuerza de voluntad, la inteligencia y la sabiduría para poder seguir adelante y afrontar cada obstáculo en la realización de este proyecto, así como también por colocarme a las personas indicadas para mi formación académica como ingeniero.

Agradezco a mi padre Edwin Martínez, por ser mi amigo, mi ejemplo a seguir como padre, como profesional, como persona y como hombre de familia, por sus consejos y lo más importante, darme la oportunidad de estudiar, haciendo lo imposible para solo dedicarme a mi formación como ingeniero. Gracias gordito!

Agradezco a mi madre Sabina Castañón, por atenderme, ser mi amiga, mi confidente, la mujer que me enseño que con fe todo se logra, además de ser una de las personas que me inculco los valores de respeto, compromiso y dedicación como persona y como profesional, también doy gracias por apoyarme, aconsejarme, y tranquilizarme con sus palabras y consejos en aquellos momentos de dificultad en la carrera y en la culminación del trabajo especial de grado. De igual manera doy gracias a cada uno de mis hermanos, Carla, Edwin y Romina, por ser mis amigos y confidentes, por ser los primeros en apoyarme, entenderme, estar pendiente de mí, cuidarme como si aún fuera un niño, orgulloso de cada uno de ustedes.

Agradezco a Jonas ese compañero de tesis, más que un compañero o colega mi amigo, a pesar de las dificultades que se presentaron, juntos supimos llevar adelante la culminación de este proyecto, además agradecido por extenderme su mano y abrirme las puertas de su casa y de su familia. De igual modo le doy las gracias a la Sra. Susana López, por su atención y apoyo en todo momento, al igual que los hermanos Sánchez, sin duda alguna agradecido con la familia Sánchez López, por su solidaridad y amistad que perdurara a través del tiempo y la distancia.

Agradecido con Marelyn Saez, persona que me enseño a estudiar, que me enseño a mirar el mundo con otra perspectiva y lo más importante, su apoyo y dedicación prácticamente en toda la carrera.

Es imposible no mencionar y agradecer a todos mis compañeros, amigos y futuros colegas, Jennifer, Yutzani, Rossana, Anyeliz, Manuel y Ronald, juntos vivimos desvelos, disputas, suspenso y lo más importante alegrías al pasar cada una de las materias de la escuela, personas muy influyentes en mi formación como ingeniero, gracias por su apoyo.

A su vez quiero agradecer a Elaimy, Maria, Mariam, Mariel, Patricia y Yessica, amigas que estuvieron pendientes en todo momento en la elaboración de la tesis. Gracias por su apoyo y sus palabras de aliento.

Por ultimo quiero agradecerles a todos los profesores que colaboraron y tomaron de su tiempo para asesorarnos en la realización del proyecto de grado. Gracias por su apoyo, Profesores: Demetrio Rey Lago, Antonio Fedon, Elimar Hernandez y Wilmer Sanz.

Daniel A Martínez C.

Agradezco a Dios por darme la inteligencia, la fuerza, la voluntad y la sabiduría para poder cumplir esta meta de vida. Por lo bueno y por lo malo que viví a lo largo de mi carrera, ya que de alguna u otra manera sirvió como instrumento para superar los obstáculos que hicieron presencia en este camino llamado 'Universidad'.

Te agradezco a ti Sr. juez, mi viejo, mi padre y mejor amigo. Gracias por tu consejo, por tu apoyo, por tu confianza y por velar que mi camino fuera siempre el correcto, hacia la humildad, el respeto y trabajo duro, pero lo más importante, hacia el bien para con el prójimo.

Doy las gracias a la persona que guía y cuida de cada uno de los pasos que doy, a ti madre, mi amiga, mi ángel guardián aquí en la tierra. Gracias por ese amor y apoyo incondicional que desde siempre me haz otorgado, gracias por hacerme la persona que soy hoy, por esa dedicación inagotable con cada uno de tus hijos, por demostrarme que el éxito no es lo que comúnmente establece la sociedad (riqueza material y fama) sino que es la consecución de un objetivo, que por más grande o pequeño que sea, nos llena de felicidad y satisfacción personal, y con esto mamá espero haber cumplido uno de tus objetivos de vida, porque de ser así; puedo dar como alcanzado uno de los míos. Gracias, sin ti nada de esto hubiese sido posible.

De igual manera agradezco a mis tres hermanos, compañeros y protectores de vida. Gracias por asumir de excelente manera, ese rol tan importante para un hermano menor; ese rol que como hijos, hermanos, hombres y profesionales veo constantemente en ustedes. Gracias por esas muestras de optimismo, apoyo, preocupación y regocijo en cada proyecto que emprendo, porque de lo contrario no sería nada fácil llevarlos a cabo.

A esa mujer que me ha acompañado a atravesar todas y cada una de las dificultades que han surgido a lo largo de mi carrera. Gracias por esa paciencia, por esa comprensión, por ese apoyo y por esa compañía que hicieron de esto, un trayecto más sencillo y ligero. Gracias por estar siempre ahí, disponible cuando necesité de un hombro en cual desahogarme, cuando necesité de palabras de aliento para continuar. Por eso y mucho más, gracias María Eldina Laurentín.

A mi colega y compañero de tesis Daniel Martínez. Gracias por haber decidido emprender conmigo la realización de este trabajo, que como en todos los casos, estuvo lleno de disputas pero que supimos sobrellevar de la mejor forma. Gracias por esa amistad, porque puedo estar seguro de que no acaba aquí.

A esos compañeros de lucha y futuros colegas, en especial a esos que vivieron junto a mí noches infinitas de angustia y preocupación, así como también; momentos llenos de alegría y felicidad. Gracias por regalarme su amistad y compañerismo.

Por último, pero no menos importante, gracias a todos esos profesores que dedicaron un poco de su tiempo para prestarnos su ayuda y conocimientos que fueron necesarios para la culminación de este proyecto de grado.

Jonás Sánchez López.

Índice general

Índice de figuras

Indice de tablas

Acrónimos

3GPP	**3**rd **G**eneration **P** artnership **P**roject
3GPP2	**3**rd **G**eneration **P** artnership **P**roject **2**
ARQ	**A**utomatic **R**epeat **r**eQuest
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
AWGN	**A**dditive **W**hite **G**aussian **N**oise
CLB	**C**ofigurable **L**ogic **B**locks
CPLD	**C**omplex **P**rogrammable **L**ogic **D**evice
DVB-S	**D**igital **V**ideo **B**roadcasting by **S**atellite
EVDO	**EV**olution **D**ata **O**ptimazed
FEC	**F**orward **E**rror **C**orrection
FPGA	**F**ield **P**rogrammable **G**ate **A**rrays
GF	**G**alois **F**ield
GSM	**G**lobal **S**ystem for **M**obile communications
HDL	**H**ardware **D**escription **L**anguage
HDMI	**H**igh **D**efinition **M**ultimedia **I**nterface
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
IOB	**I**nput **O**uput **B**locks
ISE	**I**ntegrated **S**ynthesis **E**nvironment
LAN	**L**ocal **A**rea **N**etwork
LED	**L**igth **E**mitting **D**iode
LTE	**L**ong **T**erm **E**volution
LUT	**L**ook **U**p **T**able
MAN	**M**etropolitan **A**rea **N**etwork
OFDMA	**O**rthogonal **F**requency **D**ivision **M**ultiple **A**ccess

PLD	**P**rogrammable **L**ogic **D**evice
QoS	**Q**uality **o**f **S**ervice
RTL	**R**egister **T**ransfer **L**evel
SIPO	**S**erie **I**nput **P**arallel **O**ut
SPLD	**S**implex **P**rogrammable **L**ogic **D**evice
TBCC	**T**ail **B**iting **C**onvolutional **C**odes
UMB	**U**ltra **M**obile **B**roadband
UCF	**U**ser **C**ontrainst **F**ile
UMTS	**U**niversal **M**obile **T**elecomunications **S**ystem
USB	**U**niversal **S**erial **B**us
VHDL	**V**ery **H**ardware **D**escription **L**anguage
VHSIC	**V**ery **H**igh **S**peed **I**ntregated **C**ircuit
WiMAX	**W**orldwide **i**nteroperability for **M**icrowave **A**ccess

DESARROLLO DE ALGORITMOS DE CODIFICADORES CONVOLUCIONALES USADOS EN ESTÁNDARES DE COMUNICACIONES INALÁMBRICAS BASADO EN LA TARJETA FPGA SPARTAN 6

por

MARTÍNEZ C. DANIEL A. y SÁNCHEZ L. JONÁS

Presentado en el Departamento de Señales y Sistemas
de la Escuela de Ingeniería en Telecomunicaciones
el 26 de Julio del 2015 para optar al Título de
Ingeniero de Telecomunicaciones

RESUMEN

La presente investigación plantea el desarrollo de algoritmos de codificadores convolucionales en el lenguaje de descripción en hardware VHDL empleados en los estándares de comunicaciones inalámbricas LTE, UMB, WiFi y WiMax, teniendo como objetivo principal la portabilidad de los diseños, permitiendo de esta manera su implementación en cualquier dispositivo programable (ASCI's, PLD's, FPGA's, entre otros) de cualquier fabricante . El desarrollo del proyecto inicia con la creación en software libre de los codificadores convolucionales de acuerdo a los parámetros y estructuras de cada tecnología usando como lenguaje abierto Python. Posteriormente se realiza el desarrollo de los códigos para los codificadores en VHDL. Una

vez diseñados los algoritmos en cada lenguaje de programación, comienza el proceso de validación, en el cual; fue necesario realizar pruebas con tramas de bits aleatorios con el fin de comprobar su correcto funcionamiento comparando los resultados obtenidos en Python con los conseguidos en VHDL, para que finalmente se consiguieran implementar estos últimos en la tarjeta FGPA spartan-6 . Por último se analiza el rendimiento del dispositivo (como la velocidad de respuesta, el consumo de recursos en hardware y el consumo de potencia) para cada tecnología inalámbrica...

Palabras Claves: Codificadores convolucionales, FPGA, VHDL, Software libre

Tutor: ELIMAR HERNANDEZ
Profesor del Departamento de Señales y Sistemas
Escuela de Telecomunicaciones. Facultad de Ingeniería

Capítulo I

Introducción

1.1. Motivación

Hoy día la población mundial exige comunicarse de forma rápida y efectiva, debido a que es necesario enviar gran cantidad de información en el menor tiempo posible, además; demandan que los sistemas de comunicación sean más seguros y confiables al momento de transmitir la información, es por ello que se ha optado por emplear dispositivos que puedan soportar un número de operaciones por muestra elevado y que su consumo de energía sea mínimo, haciéndolos ideales para reducir los costos asociados a la transmisión de datos [1]. Aun cuando en los sistemas de comunicaciones modernos se hayan vuelto rigurosos con respecto a la transmisión libre de errores, es posible garantizar una buena calidad de servicio (QoS) mediante la detección y corrección de errores, por consiguiente se implementan mecanismos codificadores que sean eficientes y modernos, de manera que tengan lo necesario para cumplir con tal propósito.

Al mismo tiempo, los sistemas de comunicación actuales permiten el desarrollo y optimización de las nuevas tecnologías, dichos sistemas deben poseer características adaptativas en los esquemas de modulación y codificación para que puedan ser ajustados dinámicamente a las condiciones del canal para mejorar el rendimiento

del mismo [2], sin duda; en todas estas implementaciones se perfila al uso de dispositivos reconfigurables, a causa de que la tendencia en general es reducir los costos, tiempo y espacio en la transmisión de datos.

Ahora bien, las tarjetas reprogramables *FPGA's* cumplen con las exigencias antes mencionadas, debido a que ellas poseen características que admiten modificaciones en tiempo de ejecución que las destacan para el diseño de módulos configurables de los sistemas de comunicación, todo esto mediante métodos que permitan describir el funcionamiento y la arquitectura de un sistema digital y puesto que ellas se programan mediante lenguaje de descripción de hardware, se utilizó *VHDL* como herramienta de programación [3].

Por otro lado, el área de aplicación para diseños de sistemas digitales en hardware es posible mediante la implementación de codificadores convolucionales, así como lo muestra **Fernando Serralde Medina** en su trabajo de pregado "*Simular en VHDL la implementación de algoritmos FEC en dispositivos programables reconfigurables*". Allí, se expone un código VHDL para la implementación de un codificador convolucional con tasa de codificación de *1/3*, incluyendo el bloque generador de etapas *k*, mediante un registro electrónico de entrada serie y salida paralelo (*SIPO*), con el fin de visualizar la generación y aplicación directa de dicho bloque al bloque de lógica convolucional. Así mismo, **Felipe Iván Troncoso Hernández** en su investigación de pregado "*Evaluación y aplicación de estrategias para control de errores en canales satelitales mediante codificación algebraica*", implementa un codificador y un decodificador convolucional, con la diferencia de que cuenta con un bloque de activación (`enable`) y reinicio (`reset`), que permite que los codificadores se mantengan en un modo de reposo cuando se inserta una trama de bits, de manera tal; que la trama y los registros de desplazamientos queden en el estado actual, obteniendo un estudio más eficiente del mismo.

En este sentido, usando la codificación convolucional se puede aumentar la capacidad de los bits de salida sin incrementar el ancho de banda, obteniendo como principal ventaja la optimización del espectro radio eléctrico para las comunicaciones inalámbricas. En consecuencia, al no disponer con un modelo de códigos

en VHDL de codificadores convolucionales en código abierto que permita desarrollar, actualizar, optimizar y tener hibridaciones en los sistemas de comunicaciones inalámbricos, en este trabajo se muestra el diseño y generación de los códigos en VHDL para los codificadores convolucionales usados en las tecnologías modernas (UMB, LTE, WiMax y WiFi) porque los existentes son bajo la modalidad de bloques del *System de Matlab* y no ofrecen portabilidad de diseño a diversas plataformas.

Finalmente, en Venezuela se han elaborado estudios con codificadores de canal, pero no se ha encontrado bibliografía disponible que comprenda algún tipo de investigación acerca de los codificadores convolucionales, específicamente en la Universidad de Carabobo, y particularmente los usados en los estándares de comunicaciones inalámbricas modernos mencionados anteriormente, así pues, este estudio ofrece nuevas soluciones en hardware a nivel nacional, además de servir como referencia para el desarrollo u optimización de futuros trabajos, incrementando la competencia de los profesionales de la FACULTAD DE INGENIERIA y en especial a los alumnos de la escuela de INGENIERÍA ELECTRICA y de TELECOMUNICACIONES de la UNIVERSIDAD DE CARABOBO en esta área del conocimiento.

1.2. Objetivos

1.2.1. Objetivo General

Desarrollar algortimos de codificadores convolucionales usados en estándares de comunicaciones inalámbricas, empleando el lenguaje de descripción de hardware VHDL, haciendo uso de la tarjeta FPGA Spartan-6 de Xilinx.

1.2.2. Objetivos Específicos

- Diseñar los algoritmos para la codificación convolucional apoyados en el lenguaje de programación interpretado *Python*.
- Diseñar los algoritmos para la codificación de códigos convolucionales haciendo uso del lenguaje de descripción especializado VHDL.

- Evaluar la implementación de algortimos de codificación convolucional en la tarjeta de desarrollo FPGA Spartan-6.
- Validar los códigos convolucionales generados en los diferentes lenguajes de programación.

1.3. Alcance

La investigación comprende una fase teórica, donde se analizan los desarrollos científicos en el área de codificadores convolucionales para estándares de comunicaciones inalámbricas WiFi, WiMax, LTE y UMB.

También abarca una fase de investigación aplicada, la cual consta de la descripción en VHDL de los codificadores convolucionales usados en los estándares de los sistemas de comunicación inalámbricos, con los parámetros l y k usados generalmente de acuerdo a la tecnología demandada, validado cada uno de ellos por el análisis de eficiencia del diseño.

Así, se mantiene el interés de ofrecer a la comunidad tecnológica un modelo de codificador convolucional que pueda ser usado en sistemas de comunicaciones donde puedan considerarse diversos aspectos, como la velocidad de respuesta, el consumo de recursos en hardware y el consumo de potencia.

De esta manera se presentan como productos finales: los códigos en VHDL de los componentes diseñados, la simulación del comportamiento de estos, el análisis de utilización de recursos, especificar las ecuaciones para la descripción en VHDL de los componentes de interés, la validación de los códigos generados en VHDL mediante Matlab y Python y la implementación de dichos códigos en la tarjeta FPGA Spartan-6.

Capítulo II

Marco conceptual

2.1. Introducción a los sistemas de comunicaciones digitales

En la actualidad existe una fuerte tendencia hacia las comunicaciones digitales, esto debido a sus ventajas frente a las analógicas, claramente cada una posee sus respectivas atribuciones y el uso de herramientas programables reconfigurables (por ejemplo FPGA's, CPLD'S, entre otros), resultan de excelente ayuda en el diseño de un sistema de comunicación digital.

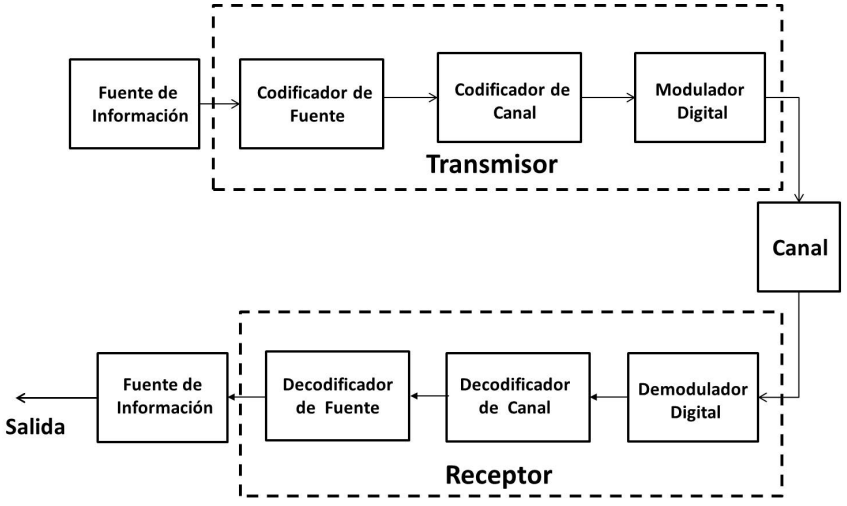

Figura 2.1: Sistema de Comunicaciones. Fuente: [4]

En la figura 2.1 se muestra la configuración básica de un sistema digital de comunicaciones [4], con los respectivos elementos que lo conforman. La fuente de información puede ser una señal analógica o una señal digital. Una vez que llega la información al codificador de fuente, este se encarga de transformarla en una secuencia digital, y para el caso de una fuente continua, este proceso involucra una conversión de analógico a digital.

Posteriormente, la secuencia digital llega al codificador de canal, donde de manera controlada se introducen algunos bits de redundancia a la secuencia de información, con la finalidad de que en el receptor se disminuyan los efectos del ruido y distorsiones generadas por el canal y así poder darle fiabilidad al mensaje. El modulador convierte los símbolos discretos en formas de ondas que luego serán transmitidas a través del canal.

En el receptor, el demodulador convierte la señal recibida a secuencias digitales que representan un estimado de los símbolos transmitidos, esta secuencia pasa a través del decodificador de canal, el cual, trata de reconstruir la secuencia de información a la secuencia original transmitida, con la ayuda del conocimiento del código y los bits de redundancia introducidos por el codificador de canal. Por último, para obtener el mensaje de salida, es necesario que la secuencia de bits provenientes del decodificador de canal, pase a través del decodificador de fuente, que intenta reconstruir la señal original.

2.2. Codificación de canal

En cualquier proceso de comunicación, es de vital importancia, poder obtener en el receptor la información del emisor lo más fielmente posible, sin embargo; durante el proceso de transmisión pueden producirse alteraciones en el mensaje debido a la presencia de ruido en el canal, dichas alteraciones son llamadas errores. Buscando proteger la información de estos errores, se realiza el proceso de codificación de canal, el cual, primordialmente protege la información ante las degradaciones del medio, así como detecta y pretende corregir los errores producidos en el

mismo, mediante el agregado de redundancia y/o alguna secuencia estructurada, principalmente.

La codificación de canal puede tener dos vertientes; ya sea mediante un esquema de repetición, denominado ARQ (*Automatic Repeat reQuest*) o mediante le extensión del mensaje agregando símbolos de redundancia en función de algún algoritmo implementado en el codificador, denominado FEC (*Forward Error Correction*) [5].

El diseño de códigos especializados en detección o corrección depende de las características del sistema en donde se considera aplicar y se pueden clasificar de la manera siguiente:

- Códigos lineales de bloques.
- Códigos convolucionales.
- Turbo códigos.

2.2.1. Codificación convolucional

Los codificadores convolucionales son una de las principales técnicas de corrección de errores usado en aplicaciones como: comunicaciones inalámbricas, sistemas digitales espaciales, sistemas de broadcasting, entre otros. Un codificador convolucional, es un código corrector de errores que procesa la información de manera continua en bloques de pequeña longitud. El codificador convolucional tiene memoria, donde los símbolos de salida dependen no solamente de los símbolos de entrada, sino que también de los símbolos previos a la salida, en otras palabras el codificador es un circuito secuencial que contiene máquinas de estado. El estado del codificador es definido por los símbolos insertados en la memoria [6].

En la figura 2.2 se muestra la estructura de un codificador convolucional, en general; la codificación convolucional contiene un número K de tramas almacenadas en los registros de desplazamientos, uno por cada bit de información de entrada,

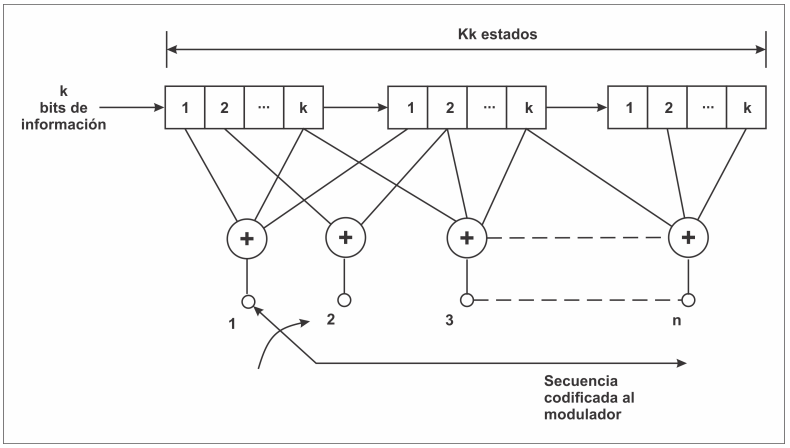

Figura 2.2: Codificador convolucional. Fuente: [4]

con n bits de salida codificada, el cual es una combinación lineal del contenido de los registros y la información de los bits de entrada [6].

Estos códigos convolucionales operan en el campo de galois *GF(2)*, es decir las operaciones de suma y multiplicación binaria son en módulo 2, las cuales se realizan solo con compuertas exclusivas *OR* [7] ,[8]. Un código convolucional se denota como *C (n,k,m)*, donde:

- n: es el número de bits de la palabra codificada.
- k: es el número de bits de la palabra de datos.
- m: es la memoria del codificador, es decir, son todos los registros de desplazamiento que existen en el codificador.

Conviene resaltar que mientras más grande sea el valor de la memoria más capacidad de corregir errores tendrá el codificador, sin embargo más complejo será su decodificación [6].

Otro parámetro importante a mencionar es la tasa del código, definida como $R = k/n$, que representa el número de bits de información enviados en una transmisión a través del canal de comunicaciones [4].

2.2.1.1. Representación de los codificadores convolucionales

Existen cuatro métodos alternativos que se utilizan para describir el proceso que realiza un codificador convolucional, y están representados de la siguiente manera:

- Representación del generador.
- Representación por diagrama de árbol.
- Representación por diagrama de trellis.
- Representación por diagrama de estado.

Representación del generador

Este método se caracteriza por representar al codificador mediante el uso de polinomios generadores [9], haciéndolo de manera algebraica.

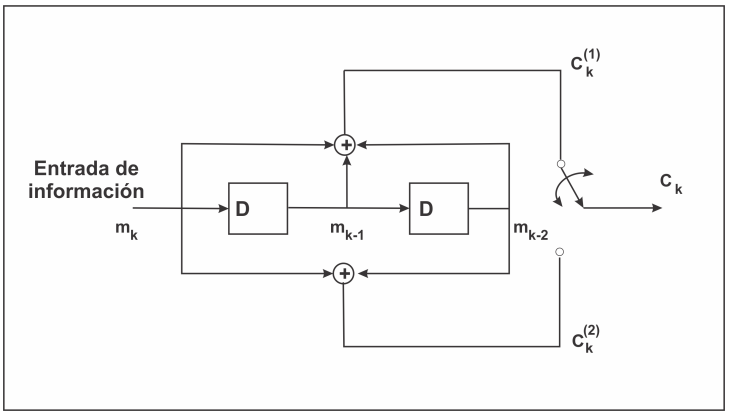

Figura 2.3: Codificador convolucional estándar de la NASA. Fuente: [10]

Por ejemplo, la representación del generador para el codificador convolucional estándar de la NASA [10] mostrado en la figura 2.3, se puede expresar como:

$$g_0 = 1 + D + D^2$$
$$g_1 = 1 + D^2$$

Por su parte, los polinomios del codificador generalmente se representan con notación octal, por lo que:

$$g_0 = 111 \rightarrow 7_8$$
$$g_1 = 101 \rightarrow 5_8$$

2.3. Estándares de comunicaciones inalámbricas

Las telecomunicaciones en los últimos años se han enfocado en la implementación de sistemas de comunicaciones a través de estándares, dichos estándares son un conjunto de normas y recomendaciones técnicas encargadas de la regulación del sistema. Para este trabajo se tomaron en consideración cuatro ellos.

2.3.1. 802.11 WiFi

Este estándar es usado para redes de árrea locales (*LAN*), con la finalidad de ofrecer conectividad inalámbrica para estaciones móviles portátiles fijas dentro de un área. Dicha norma ofrece también organismos reguladores para el acceso a las bandas de frecuencias para la conexión de la comunicación de área local [11].

2.3.2. 802.16 WiMax

Es un estándar de transmisión inalámbrica de datos, diseñado para ser usado en redes de área metropolitana (*MAN*), el mismo permite un acceso inalámbrico de banda ancha de largo alcance, el cual facilita la entrega de grandes cantidades de información de manera económica. Se utiliza principalmente para ofrecer coberturas a zonas de difícil acceso y para solucionar los problemas de conectividad de *última milla*, además se encarga de brindar a los proveedores y consumidores una conexión ininterrumpida para aplicaciones en tiempo real [12].

2.3.3. Ultra Mobile Broadband (UMB)

El estándar 3GPP2 UMB, es diseñado para el acceso de banda ancha móvil, es optimizado para una alta eficiencia espectral y latencias cortas, gracias a avanzados esquemas de modulación (principalmente OFDMA), adaptaciones en los enlaces y técnicas de transmisión para multiantenas. También permite un rápido *Handoff* y un mayor control de potencia. En el diseño del sistema se incorpora el manejo de interferencia inter-sector, con el objetivo de facilitar la comunicación en entornos móviles. UMB posiblemente se convierta en el sucesor de *1xEVDO* como el siguiente sistema inalámbrico de alta velocidad en 3GPP2 [13].

2.3.4. Long Term Evolution (LTE)

Es un estándar de comunicaciones inalámbricas móviles desarrollado por 3GPP, siendo la generación avanzada de GSM y UMTS. En comparación con las otras generaciones, LTE permite altas velocidades de transmisión/recepción de bits con una baja latencia, es un sistema fácil de desplegar por los operadores móviles [14].

2.4. VHDL (Very Hardware Language Descriptor)

VHDL [3] es un lenguaje descriptor de hardware para circuitos integrados de muy alta velocidad (*VHSIC: Very High Speed Integrated Circuit*). Surgió del programa de Gobierno de los Estados Unidos que se inició en 1980 y fue en 1987 cuando es reconocido como un HDL estándar por el Instituto de Ingenieros en Electricidad y Electrónica (*IEEE*), aspecto muy importante que respalda el uso del lenguaje. Actualmente su uso está ampliamente aceptado como un estándar para diseñar sistemas digitales y electrónicos.

VHDL puede cubrir una serie de necesidades en el proceso de diseño [15], ya que permite la descripción de la estructura del mismo, también da como resultado un diseño que puede ser simulado antes de ser fabricado, de modo que se puedan

comparar alternativas rápidamente y pruebas de corrección sin la demora y los gastos de creación de prototipos de hardware.

Finalmente, VHDL es un lenguaje portable y reusable ya que es independiente de la tecnología o fabricante (Xilinx, Altera, Actel, QuickLogic, entre otros). Sus sentencias a diferencia de un programa de software se ejecutan inherentemente en forma *concurrente* [16] salvo aquellas que se incluyan dentro de un procedimiento (*Procedure*), Proceso (*Process*) o Función (*Function*) donde se ejecutaran en forma secuencial.

2.4.1. Estructura de un programa VHDL

La estructura general de un programa en VHDL, esta definida por módulos o unidades de diseño, cada uno de ellos formado por un conjunto de declaraciones e instrucciones que se encargan de describir y estructurar el comportamiento del circuito digital a diseñar. Las unidades fundamentales que forman un código VHDL son: Librerías (*Lybrary*), entidad (*Entity*) y arquitectura (*Architecture*).

Librerías:

Las librerías son unidades de diseño predeterminadas que permiten declarar, almacenar estructuras lógicas y definir datos, con la finalidad de facilitar y agilizar el diseño.

Entidad:

Es el bloque elemental de un diseño en VHDL, mediante el cual se especifican las entradas y salidas del circuito. Una entidad esta compuesta por tres elementos:

- **Puertos de entradas y salida** que contienen un nombre, un modo y un tipo de dato.
- **Modos**, para la cual existen cuatro tipos:

1. Modo `in`: Puerto para señales de entradas.
2. Modo `out`: Puerto para señales de salidas.
3. Modo `inout`: Puerto bidireccional con retroalimentación dentro o fuera de la entidad.
4. Modo `buffer`: Puerto que permite la retroalimentación interna dentro de la entidad, sin embargo; el puerto declarado actúa como una terminal de salida.

- **Tipos de datos**, los cuales definen los valores establecidos para los puertos de la entidad, pueden ser: `std_logic` (bit), `boolean` (valores de verdadero o falso), `std_logic_vector` (vectores de bits) e `integer` (números enteros).

Arquitectura:

Es la estructura que especifica el funcionamiento de la entidad, es decir; describe el comportamiento y procedimientos que se llevaran a cabo con la finalidad de que la entidad cumpla con las condiciones de diseño deseadas.

2.4.2. Elementos sintácticos de VHDL

El lenguaje VHDL, contiene elementos sintácticos, estructuras y tipos de datos, a continuación se presentan los elementos mas usados en el diseño de los codificadores. Para mayor detalle consultar [16].

- **Comentarios:** Linea que empieza con dos guiones "−− "
- **Identificadores:** Son usados para identificar módulos, señales, variables, nombres de rutina, entre otros. Puede ser cualquier nombre compuesto por mayúsculas, minúsculas o el símbolo de subrayado "_ ".
- **Caracteres:** Cualquier letra o carácter entre comillas simples: '1', '3','z'.
- **Cadenas:** Es cualquier letra o carácter entre comillas dobles "`Cadena` ".

- **Cadena de bits:** `bit` y `bit_vector` son tipos de carácter y matriz de carácter respectivamente. Una manera elegante de definirlo en VHDL, es indicando su base mediante su prefijo.

2.4.3. Operadores varios, aritméticos y lógicos.

Los operadores lógicos y aritméticos en VHDL, son muy similares a los usados en otros lenguajes de programación, permitiendo mayor simplicidad a la hora del aprendizaje. Algunos de estos operadores y expresiones delimitaron el diseño del código. Para mayor información consultar [16].

1. **Operadores varios:**
   - & (concatenación): Concatena matrices de manera que la matriz resultante es la suma de las dimensiones de las matrices sobre las que opera.
2. **Operadores aritméticos:**
   - `**` exponencial.
   - `ABS()` Valor absoluto.
   - `*,/` Multiplicación y división.
   - `+,-` Suma y resta si va entre dos operandos, indica el signo si va al principio del operando.
3. **Operadores Lógicos:**
   - `NOT`, `AND`, `NAND`, `OR`, `NOR`, `XOR`. El funcionamiento de estos operadores es el habitual, si son usados con vectores, la operación es bit a bit.

2.4.4. Declaración de un proceso

La declaración de un proceso define una etapa secuencial independiente, que representa el comportamiento de una parte del diseño. Se compone de estados secuenciales, cuya orden de ejecución depende del diseño, es decir; en el interior de un proceso siempre va una sentencia secuencial.

Un proceso contiene una lista de sensibilidad, la cual es una serie de señales que, al cambiar su valor, hacen que se ejecute el proceso. Para mayor detalle de las sentencias secuenciales, procesos, y descripciones estructurales se recomienda consultar [16], [3].

2.5. Dispositivos de lógica programable

Una vez conocida parte de la función y estructura del lenguaje de descripción de hardware VHDL, es importante destacar como se mencionó anteriormente que, en una realización física de un diseño se utiliza la combinación entre estos lenguajes acompañados de algún dispositivo de lógica programable, que junto al procesamiento digital de señales, permite llevar a cabo toda operación que el usuario requiera aplicar a cualquier señal digital.

Los dispositivos lógicos programables o *PLD's*, como el nombre lo indica, son dispositivos configurables por el usuario que estan compuestos por un conjunto de elementos lógicos (*AND, OR, NOT, FLIP-FLOP*) para que realicen una determinada función o grupo de funciones lógicas [10].

Ahora bien, los PLD's están disponibles para que cada usuario seleccione aquel que se adapte a su aplicación y su hardware final responda adecuadamente a la situación para la cual fue diseñado, y de acuerdo a su complejidad, el dispositivo pueda ser reprogramado (si lo admite) para adaptarse a nuevos cambios o requerimientos del usuario. Estos dispositivos son una buena alternativa al momento de realizar diseños electrónicos digitales, ya que permiten disminuir los costos y el tiempo de realización del prototipo.

Así pues, cabe señalar que existen varios dispositivos que tienen funciones y capacidades diferentes, por lo que algunos son más adecuados para determinadas aplicaciones [10], entre ellos tenemos:

- SPLD (*Simplex Programmable Logic Device*)
- CPLD (*Complex Programmable Logic Device*)

- FPGA (*Field Programmable Gate Array*)

2.5.1. FPGA (Field Programmable Gate Array)

Este trabajo ha sido orientado hacia una solución de hardware de los codificadores convolucionales, por lo cual su descripción debe estar dada para un dispositivo de hardware configurable y de tecnología actualizada, en este aspecto se selecciona la tecnología FPGA.

Una FPGA es un dispositivo semiconductor programable que está basado en una matriz de bloques lógicos configurables conectados a través de interconexiones programables. De manera contraria a lo que sucede con los ASIC (*Application Specific Integrated Circuit*) donde el dispositivo es hecho a la medida del diseño particular [10], una FPGA como se mencionó en el aparatado anterior, puede ser programada para una aplicación deseada o para cumplir ciertos requerimientos de funcionalidad colocando a estos dispositivos como una de las herramientas más versátiles en estos momentos.

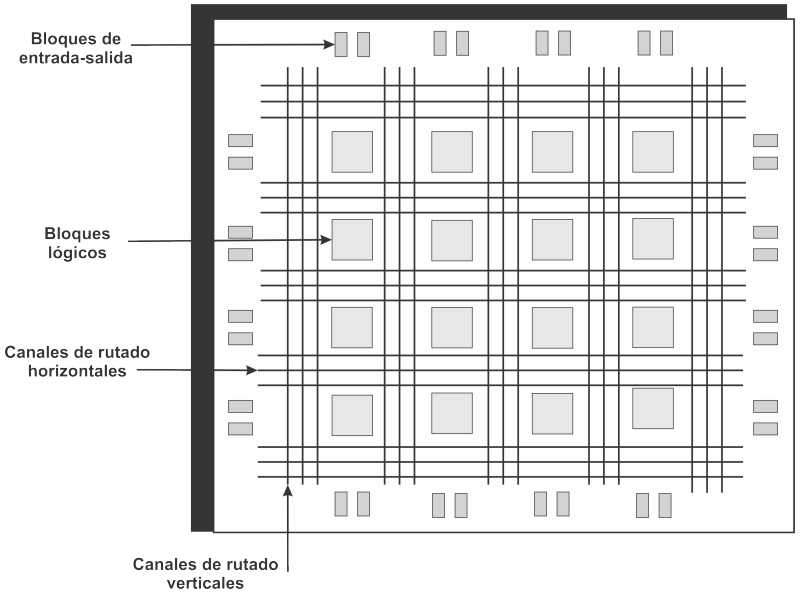

Figura 2.4: Arquitectura de una FPGA. Fuente: [3]

Los dispositivos FPGA se basan en lo que se conoce como arreglos de compuertas, los cuales consisten en la parte de la arquitectura que contiene varios elementos configurables como los mostrados en la figura 2.4, estos son: los bloques lógicos configurables (*CLB: Configurable Logic Blocks*), los bloques de entrada y salida (*IOB: Input Output Blocks*), los recursos de interconexión y la memoria *RAM* [3]. La densidad de las FPGA's se establece en cantidades equivalentes a cierto número de compuertas.

Es por ello, que la combinación del lenguaje HDL y los dispositivos FPGA's permiten a los diseñadores simular, probar e implementar físicamente circuitos digitales sofisticados, reduciendo el tiempo que existe desde el diseño hasta la producción final.

Tarjeta de desarrollo FPGA ATLYS Spartan-6

La tarjeta *ATLYS* de *Digilent* combina la alta capacidad de la Spartan-6 XC6SLX45 de Xilinx con los circuitos y dispositivos necesarios para crear los sistemas digitales más exigentes de hoy en día. Ella goza de una memoria de alta velocidad *DDR2*, además de tener múltiples puertos *HDMI*, Gigabit Ethernet, y cuenta con los circuitos de sincronización y de suministro de energía más avanzados.

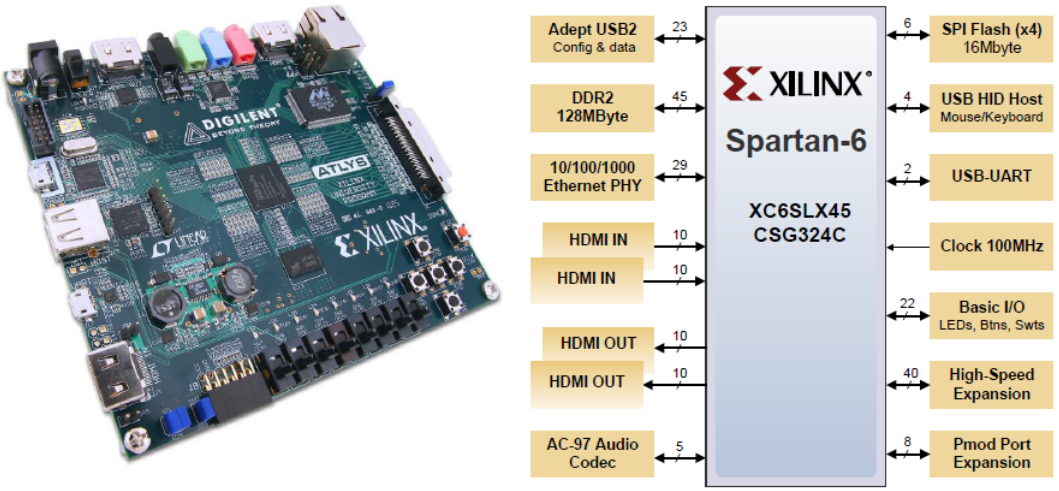

Figura 2.5: Tarjeta de Desarrollo FPGA ATLYS Spartan-6/XC6SLX45 y su esquema. Fuente: [17]

En la figura 2.5 se puede ver la tarjeta programable junto con su esquema, mostrando algunas de sus principales características [17], tales como:

- FPGA Spartan-6 XC6SLX45 de Xilinx, con paquetes BGA de 324 pines
- Dos puertos de entrada y dos puertos de salida de video HDMI
- Memoria DDR2 de 128MByte con un ancho de 16 bits de datos
- Interfaz trimodo Ethernet PHY(10/100/1000)
- Puertos USB2 para la la programación y transferencia de datos
- Puerto USB-UART y puerto USB-HID (por ratón / teclado)
- AC-97 Codec con entrada de línea, salida de línea, micrófono y auriculares
- Monitores de energía en tiempo real sobre todos los carriles de alimentación
- 16MByte x 4 SPI Flash para almacenamiento y configuración de datos
- Oscilador CMOS de 100MHz
- 48 E/S para enrutado de conectores de expansión
- Pin GPIO que incluye ocho LEDs, seis botones y ocho interruptores deslizantes

2.6. Digilent ADEPT

Adept es una aplicación creada por *Digilent Inc.* la cual permite configurar los dispositivos de la tarjeta programable, ampliar la capacidad de *E/S*, transferir, cargar y descargar archivos de datos de la PC o laptop a la tarjeta mediante un cable USB o JTAG [17] y además, ejecuta pruebas para confirmar el funcionamiento correcto del aparato, todo esto por medio de una interfaz destinada a dispositivos lógicos programables Xilinx (*FPGA's, CPLD's y PROM's*). Teniendo como ventaja de que se puede programar en la mayoría de tarjetas de la serie SPARTAN y VIRTEX con archivos *.bit* [18], [19].

2.7. Xilinx ISE Design Suites

El Xilinx ISE (*Integrated Synthesis Environment*) es una herramienta de software producido por Xilinx para la síntesis y análisis de los diseños de HDL, lo que permite al desarrollador sintetizar (*compilar*) sus diseños, realizar análisis de tiempo, simular la reacción de un diseño a diferentes estímulos, y configurar el dispositivo de destino con el programador. Así mismo, se utiliza principalmente para la síntesis de circuitos y diseño, mientras que el simulador lógico *ModelSim* se utiliza para las pruebas de nivel del sistema [20].

El Xilinx ISE es un entorno de diseño para los productos FPGA de Xilinx, por lo que no puede ser utilizado con productos FPGA de otros proveedores.

2.8. Python

En General Python es un lenguaje de programación de código abierto de alto nivel. Fue creado para la optimización de la calidad del software, la portabilidad del programa, la integración de componentes, es decir, posee una sintaxis muy limpia, permitiendo un código legible, además de que es un lenguaje interpretado, de tipado dinámico, multiplataforma y orientado a objetos. Python es uno de los lenguajes de programación más usados hoy en día a nivel mundial, abarcando distintas áreas como la programación de sistemas, interfaces de usuarios, personalización de productos, programación numérica y más [21] [22].

Capítulo III

Procedimientos de la investigación

En este trabajo de grado se planteó el diseño e implementación de un codificador convolucional en la tarjeta de desarrollo *FPGA ATLYS Spartan-6,* mediante el uso del lenguaje descriptor de hardware *VHDL* y a su vez; se validaron los códigos generados por medio del lenguaje de programación de código abierto *Python*. Desarrollando los objetivos específicos en fases de la siguiente manera:

3.1. Fase 1. Diseño en Python

Esta primera fase consta de cuatro etapas que facilitaron el desarrollo de los algoritmos requeridos para la ejecución y validación con el lenguaje descriptor de hardware de los códigos convolucionales para los diversos estándares de comunicaciones inalámbricas seleccionados. Cumpliendo de esta manera el primero de los objetivos específicos del presente trabajo.

3.1.1. Estudio del lenguaje de programación interpretado

Los lenguajes de programación están formados por un conjunto de símbolos y reglas sintácticas que definen su estructura y el significado de sus elementos y

expresiones, en efecto; se realizó el estudio y entendimiento del lenguaje de programación *Python*, así como también la estructura del mismo (tipos básicos, colecciones, control de flujo, funciones, excepciones, entre otros), examinando la bibliográfica respectiva.

Ahora bien, siendo Python un lenguaje interpretado [21], tiene la desventaja que su tiempo de ejecución es más lento, ya que traduce el programa instrucción por instrucción, situación contraria a los lenguajes compilados, que lo hacen desde su descripción al código de máquina del sistema, donde la ejecución del programa es más rápida, pero a pesar de esto, Python es más flexible y más portable, porque no depende de la plataforma en la que corre, o sea; es un lenguaje multiplataforma.

Por otra parte, Python posee la característica de tipado dinámico [22], lo que significa que una misma variable puede tomar valores de distinto tipo en distintos momentos, otra ventaja que posee con respecto a otros lenguajes donde el tipo de la variable cambiaría para adaptarse al comportamiento esperado, por consiguiente; son más propenso a errores [23].

Y a pesar de que existen diferentes tipos de lenguaje de programación de alto nivel, como **Java**, **C++** y **Matlab**, Python se eligió como el lenguaje de programación para diseñar los algoritmos de codificación convolucional, porque es más interactivo y compacto a la hora aprender y aplicar, a diferencia de C++ y Matlab por ejemplo, debido a que Matlab requiere una licencia para su uso [24] (conocido en ingles como "*proprietary software*"), o sea; no es un lenguaje de código abierto como Python.

3.1.2. Estudio de la teoría de códigos

Para realizar un sistema de comunicaciones analógico o digital que evite que la información transmitida del emisor al receptor sufra algún tipo de degradación causado por el canal, es necesario implementar un mecanismo que pueda detectar y/o corregir los errores la señal, es por ello se han desarrollado toda una teoría de códigos, que en general, se aplican primordialmente con tres grandes objetivos: comprimir mensajes, detectar y corregir errores y garantizar la calidad del mensaje.

Por esta razón, la investigación se estableció en la revisión de las características de los códigos correctores, específicamente en los codificadores convolucionales para las tecnologías de comunicaciones inalámbricas, de acuerdo a la bibliográfica referida [5], [4].

3.1.3. Selección de los estándares a evaluar

La tendencia actual de los sistemas de comunicaciones digitales es hacia la comunicación a todo momento, en todo lugar, de cualquier forma y con el mínimo de coste posible, es por ello que los organismos encargados para esta tarea necesitaron encontrar la manera de cumplir con las exigencias del mercado, así pues; las comunicaciones inalámbricas han venido evolucionando precipitadamente y de forma continua, implementando sistemas encaminados a la creación de estándares que admiten establecer un lenguaje en común para los distintos fabricantes de las múltiples tecnologías desarrolladas independientemente.

Ahora bien, los estándares por los cuales están regidos las nuevas tecnologías describen el tipo de modulación, frecuencia con la que trabajan, descripción de los canales que usan, el tipo de codificación, entre otros, sin embargo; este trabajo se basó en las técnicas implementadas de corrección para los problemas de distorsión del mensaje causados por el canal de comunicaciones, así pues; se estudió y revisó que los estándares UMB, LTE, 802.11 (WiFi) y el 802.16 (WiMax) utilizan codificadores convolucionales como método para la corrección y/o detección de errores [11], [12], [25], [26]. En las figuras 3.1, 3.2, 3.3 y 3.4 se exponen los codificadores usados para cada estándar.

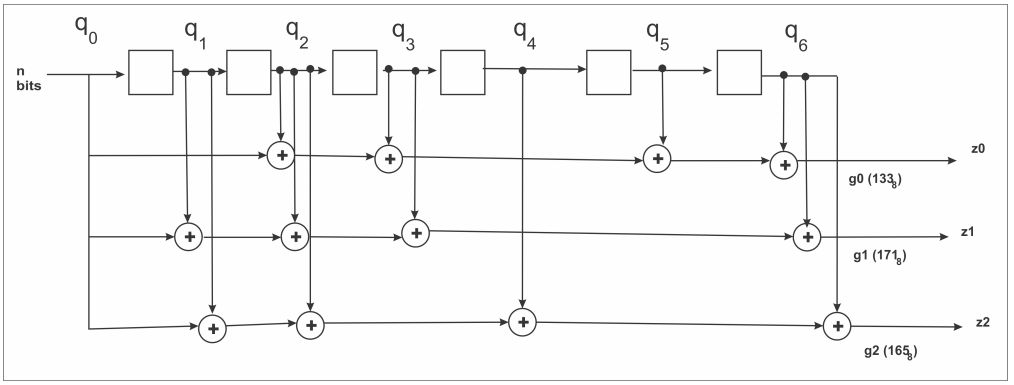

Figura 3.1: Codificador convolucional usado en LTE. Fuente: [25]

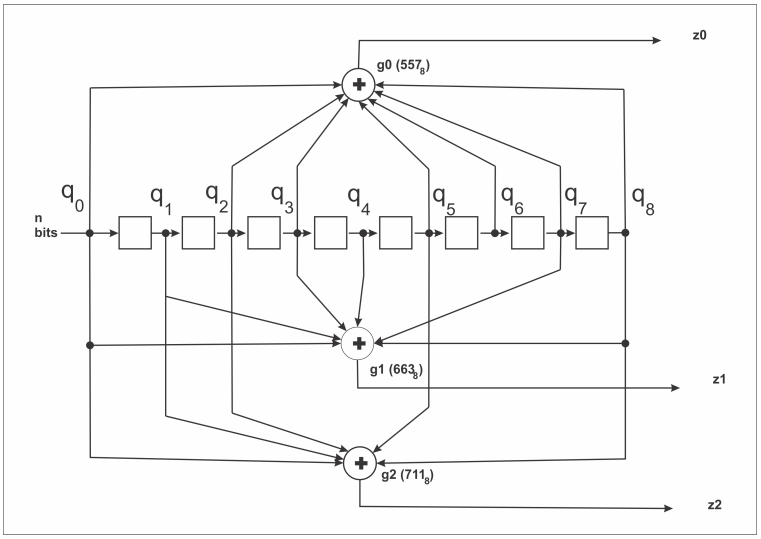

Figura 3.2: Codificador convolucional usado en UMB. Fuente: [26]

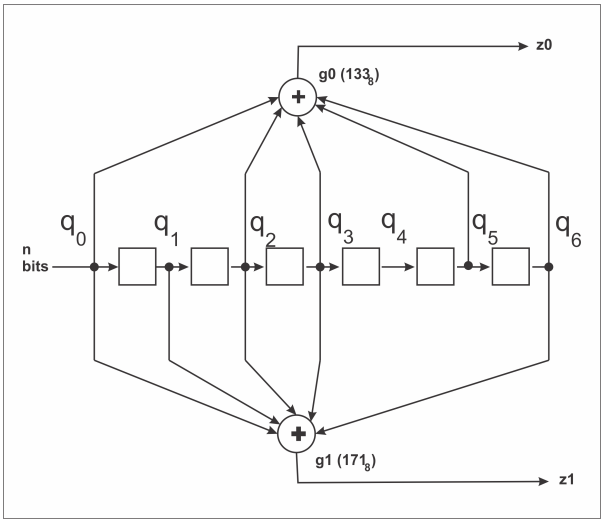

Figura 3.3: Codificador convolucional usado en WiFi. Fuente: [11]

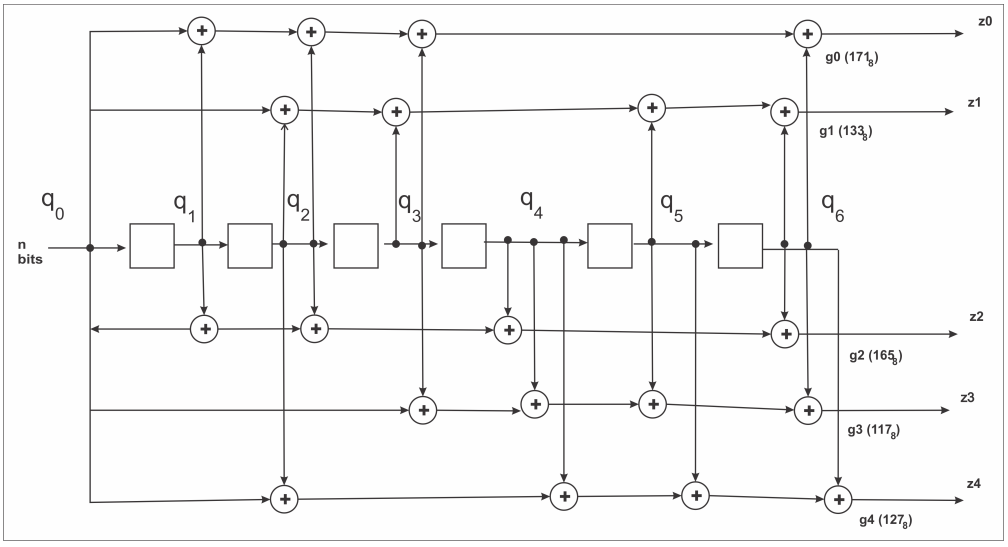

Figura 3.4: Codificador convolucional WiMax. Fuente: [12]

En las tablas 3.1 y 3.2 se ilustran los parámetros de codificación y los polinomios generadores de cada tecnología usada, de acuerdo a la estructura de codificación de cada estándar.

	UMB	LTE	WiMax	WiFi
K	9	7	7	7
R	1/3	1/3	1/5	1/2

Tabla 3.1: Parámetros de codificación de las tecnologías seleccionadas. Fuente: Propia

	UMB	LTE	WiMax	WiFi
g_0	557_8	133_8	171_8	133_8
g_1	663_8	171_8	133_8	171_8
g_2	711_8	165_8	165_8	–
g_3	–	–	117_8	–
g_4	–	–	127_8	–

Tabla 3.2: Polinomios generadores de las tecnologías seleccionadas. Fuente: Propia

Adicionalmente, se estudió que cuando se implementa un código convolucional de una longitud de restricción de K >3, se produce una ganancia de codificación que posee el mismo ancho de banda y la misma velocidad de información que cuando se implementa una señal sin codificación, por consiguiente; se optimiza el sistema de comunicación [5].

3.1.4. Diseño de códigos en Python

Como se mencionó anteriormente en 3.1.1, Python se seleccionó como lenguaje de programación, conocido esto; se procedió a diseñar los códigos para los codificadores convolucionales de las tecnologías UMB, LTE, WiMax y Wifi, de acuerdo a su estructura de codificación.

Ahora bien, es cierto que estas tecnologías difieren una de la otra tanto en sus parámetros como en su estructura de codificación, pero para Python se trata solamente de un conjunto de instrucciones, símbolos y reglas que a fin de cuentas se

diferencian en declarar una, dos, tres o más variables y por consiguiente, adicionar lineas de código al programa, es por esto que se expone sólo el código usado para la tecnología LTE. Los demás algoritmos de codificación se muestran en el **Anexo A**.

Evidentemente se determinó que Python posee un conjunto de ventajas que lo hacen atractivo frente otros lenguajes de programación de alto nivel, pero una de las ventajas que más resalta de este programa es la existencia y disponibilidad de múltiples librerías que facilitan notablemente una gran cantidad de cómputos matemáticos necesarios para la ciencia y la ingeniería. Para este trabajo se hizo uso de la librería *NumPy* [27], que no es más que una extensión de Python que le agrega mayor soporte a los vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con los mismos.

En el **Anexo A** se muestra el algoritmo completo realizado en Python del codificador convolucional para LTE, donde la entrada de datos se puede hacer mediante un archivo de texto o por consola (ya sea insertando una entrada que le indique el usuario o por una entrada aleatoria, que de igual manera el usuario también elige la longitud de la trama). A continuación se explica el proceso del código para una entrada por archivo de texto.

Definir la entrada:

El código convolucional se generó creando un archivo de texto con un bit o trama de bits (`n`) para que el programa pudiera leerlo, de lo contrario se produciría un error al compilar el código, así como se muestra en la tabla 3.3:

Tabla 3.3: Definición de la entrada para LTE en Python. Fuente:Propia

```
f = open('arch1.txt','rb')
o = open('salida1.txt','wb')
```

Como `f` es el archivo de texto de entrada, fue necesario colocar el modo de lectura de archivo de bits (`'rb'`), igualmente se realizó para `o`, sólo que en este caso

se colocó (`'wb'`) como modo de escritura para generar la entrada y los resultados de la codificación en el archivo de texto de salida.

Definir la estructura del codificador:

La estructura del codificador como se vio anteriormente, viene dada por la longitud de restricción que es el número de bits de entrada almacenados en el registro de desplazamiento y los polinomios generadores, así como se muestra en la tabla 3.4:

Tabla 3.4: Definición de la estructura de codificación para LTE en Python. Fuente:Propia

```
q=[0,0,0,0,0,0,0]
z=[]
for i in range(len(n)):
    q=numpy.insert(q,0,n[i])
    q=numpy.delete(q,len(q)-1)
    g0=[0,2,3,5,6]
    g1=[0,1,2,3,6]
    g2=[0,1,2,4,6]
```

Donde inicialmente los bits almacenados en el registro de desplazamiento **q** están inicializados en cero porque no hay dato de entrada, luego de que se leyera la trama de bits, cada uno de ellos se desplazó de izquierda a derecha, con ayuda de `numpy.insert` y `numpy.delete` de acuerdo a la longitud de `n`. Por último, se identificaron los polinomios generadores `g0`, `g1` y `g2`.

Definir la salida:

Como ya se especificaron la entrada y la estructura del codificador, solo faltó fijar la salida, donde fue necesario crear un conjunto de variables auxiliares (`z0`, `z1`, `z2`, `zr0`, `zr1`, `zr2` y `zf`) que permitieron conectar los registros de desplazamiento y los sumadores modulo dos (gracias a la función `def`) de acuerdo a los datos de entrada, obteniendo así `z`, como se muestra en la tabla 3.5:

Tabla 3.5: Definición de la salida para LTE en Python. Fuente:Propia

```
z0=numpy.take(q,g0)
z1=numpy.take(q,g1)
z2=numpy.take(q,g2)
def concatenar(a,b):
        return a^b
zr0=reduce(concatenar,z0)
zr1=reduce(concatenar,z1)
zr2=reduce(concatenar,z2)
zf=numpy.hstack((zr0,zr1,zr2))
z=numpy.append(z,zf)
z = [numpy.int32(k) for k in z]
```

Finalmente, se generó el archivo de texto de salida con los datos de entrada `n` y la salida codificada `z` haciendo uso de `o.write`.

Para complementar, resulta oportuno mencionar que se realizó un programa en consola más interactivo en este mismo lenguaje, ya que se incorporan las cuatro tecnologías en un mismo algoritmo, con la finalidad de que se pueda seleccionar el tipo de entrada (por consola o de manera aleatoria) y también permitir elegir el estándar en la cual se desea codificar el mensaje (UMB, LTE, WiMax y WiFi). Este programa interactivo se muestra en el **Anexo A**.

3.2. Fase 2. Diseño en VHDL.

En esta fase se procedió a realizar el estudio del lenguaje de descripción en hardware VHDL, así como también; del software donde se ejecutó el diseño, con el objetivo de facilitar el desarrollo de los algoritmos de codificación convolucional en VHDL.

3.2.1. Estudio del lenguaje descriptor en hardware.

El lenguaje descriptor en hardware (HDL), es un lenguaje diseñado para la descripción de sistemas electrónicos digitales, pudiendo ser implementado en un dispositivo lógico programable. Los lenguajes HDL más populares son VHDL y Verilog, sin embargo; en la Facultad de Ingeniería de la Universidad de Carabobo, el lenguaje HDL que mas se maneja es VHDL, teniendo como principal ventaja una mayor asesoría para el aprendizaje del lenguaje, por lo tanto; se seleccionó VHDL como lenguaje descriptor. Asimismo, se estudió la declaración de entidades, descripción de arquitectura, uso de operadores lógicos, aritméticos, uso de procesos, entre otros), revisando la bibliográfica respectiva [16], [3].

3.2.2. Estudio del software Xilinx Ise Design Suite.

Existen diferentes herramientas en software que sintetizan y analizan códigos en HDL, por lo general el software depende del fabricante de la tarjeta FPGA, como se dispone de una Spartan-6 de fabricante Xilinx, se estudió el manejo de la herramienta *Xilinx ISE Design Suite*, consultando diferentes manuales y tutoriales [28], [29], y de esta manera, se generaron los códigos convolucionales en VHDL, se obtuvieron los circuitos esquemáticos, se pudo realizar la evaluación de los recursos en hardware y del consumo de potencia de los codificadores.

A continuación, se describen de manera general los pasos seguidos a la hora de elaborar el proyecto en ISE Design Suite.

1. Se generó un nuevo proyecto, en el cual se especificó el tipo de FPGA donde se va a sintetizar el diseño (*Spartan-6 XC6SLX45*), ademas se seleccionó el encapsulado (*CSG324*), la velocidad de la misma(**-3**),y el lenguaje HDL a utilizar(*VHDL*).

2. Se incorporaron fuentes o módulos, seleccionando el modulo VHDL, donde se describió la programación para el diseño de los codificadores convolucionales.

3. Ya obtenidos los codificadores convolucionales en VHDL, se realizó un banco de pruebas, agregando el modulo *test bench*. Mediante el test bench y la herramienta *ModelSim*, donde se configuró la frecuencia del reloj y se comprobó el funcionamiento del codificador.

4. Se agregó el archivo UCF (*User Contrainst File*), el mismo contiene toda la información respecto a la asignación de pines de entrada y salida del codificador convolucional.

5. Se generó el archivo binario (*bitstream*) que contiene la programación en VHDL de los códigos. Ya generado el archivo bitstream, se creó el reporte de los recursos usados en la tarjeta con el diseño de los codificadores convolucionales.

6. Finalmente, se utilizaron las herramientas *constraint editor* y *xpower analyzer* para obtener la potencia consumida [30], [31].

3.2.3. Diseño de códigos en VHDL.

Para poder diseñar los códigos en VHDL se hizo uso del software Xilinx Ise Design Suite, así como se explico en 3.2.2, de este modo; se programó el código en el lenguaje descriptor de hardware los algoritmos de codificación para los estándares UMB, LTE, WiMax y WiFi, para ello se inició con la definición de la entidad, posteriormente, en la arquitectura del programa se recurrió al uso de los elementos sintácticos (operadores, procesos, entre otros), como lo expuesto en 3.2.1.

A continuación, se presentan de manera detallada la programación en hardware para la obtención del codificador convolucional empleado en la tecnología LTE. La programación para los otros estándares se exponen en el **Anexo B**.

3.2.3.1. Entidad del codificador convolucional LTE

En la figura 3.5, se muestra los puertos de entrada y de salida del codificador, definidos de la siguiente manera:

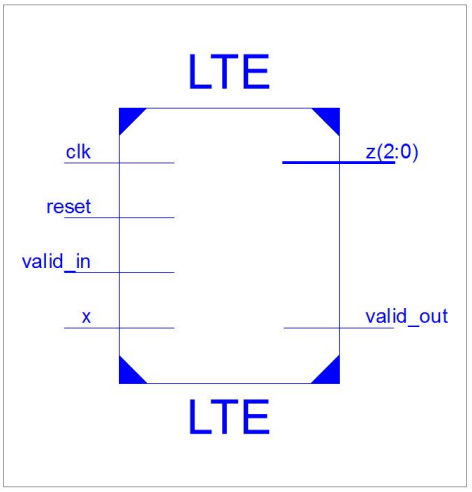

Figura 3.5: Entidad del codificador convolucional LTE. Fuente: Propia

1. **Puertos de entrada:**
   - `clk`: Reloj del sistema.
   - `reset`: Reset asíncrono que se activa en nivel alto (1). Reinicia todos los puertos de salida, las señales internas y los registros internos llevándolos a cero .
   - `valid_in`: Se activa a nivel alto (1), su activación indica al codificador que debe leer el bit de entrada. Si `valid_in` vale '0', el codificador entra en un estado de pausa, de manera que no codifica, manteniendo los valores en los registros internos.
   - `x`: Bit de entrada al codificador.
2. **Puertos de Salida:**
   - `valid_out`: Si la señal esta activa (1), entonces se muestra una salida valida, de lo contrario, la salida no tiene valores validos.
   - `z`: Bits de salida del codificador.

En la tabla 3.6, se aprecia como se definió la entidad en VHDL del codificador LTE.

Tabla 3.6: Descripción de la entidad del codificador convolucional LTE en VHDL. Fuente: Propia

```
entity LTE is
    Port ( x : in  STD_LOGIC;
           valid_in : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           valid_out : out  STD_LOGIC;
           z : out  STD_LOGIC_VECTOR (2 downto 0));
end LTE;
```

3.2.3.2. Arquitectura del codificador convolucional LTE

Tabla 3.7: Declaración de señales del codificador convolucional LTE en VHDL. Fuente: Propia

```
architecture Behavioral of LTE is
signal q:  STD_LOGIC_VECTOR(6 downto 0);
signal v: STD_LOGIC_VECTOR(2 downto 0);
```

En la tabla 3.7, se muestra la declaración de señales auxiliares en VHDL donde:

- `signal q`: Es un vector el cual indica la cantidad de registros de desplazamientos usados en el codificador.
- `signal v`: Es una señal auxiliar que especifica el vector de salida.

Ya declaradas las señales auxiliares, comienza la etapa donde se configuró el reinicio de todas las salidas y registros internos, posteriormente se estableció el proceso donde el codificador comienza a generar el patrón de salida a través de las entradas, es decir; cada símbolo de salida depende del símbolo de entrada actual y de seis símbolos de entradas anteriores. En la tabla 3.8, se puede apreciar el código en VHDL del proceso del codificador, que es controlado por el reloj del sistema, así como también por la entrada `reset`.

Tabla 3.8: Proceso del codificador convolucional LTE en VHDL. Fuente: Propia

```
begin
conv: process(clk,reset)
    begin
      if reset = '1' then
        q(0) <= '0';
        q(1) <= '0';
        q(2) <= '0';
        q(3) <= '0';
        q(4) <= '0';
        q(5) <= '0';
        q(6) <= '0';
        valid_out <= '0';
      elsif (clk='1' and clk'event) then
        valid_out <= valid_in;
        if valid_in = '1' then
          q(0) <= x    ;
          q(1) <= q(0) ;
          q(2) <= q(1) ;
          q(3) <= q(2) ;
          q(4) <= q(3) ;
          q(5) <= q(4) ;
          q(6) <= q(5) ;
        end if;
      end if;
end process conv;
```

Fuera del proceso, viene la fase donde se especificaron los polinomios generadores del codificador, es decir; se indicaron las conexiones entre los registros de desplazamiento y los sumadores modulo dos, posteriormente se asignó la salida del codificador.

Tabla 3.9: Descripción en VHDL de los polinomios generadores de LTE. Fuente: Propia

```
v(2) <= q(0) XOR q(2) XOR q(3) XOR q(5) XOR q(6) after 1.35 ns; --g0(133)
v(1) <= q(0) XOR q(1) XOR q(2) XOR q(3) XOR q(6) after 1.35 ns;--g1(171)
v(0) <= q(0) XOR q(1) XOR q(2) XOR q(4) XOR q(6) after 1.35 ns;-- g2(165)
z <= v;
end Behavioral;
```

En la tabla 3.9 se presenta el código VHDL de los polinomios generadores empleados en LTE.

Después del proceso de síntesis del codificador, se muestra en la figura 3.6 el esquemático *RTL* del codificador LTE, dicho esquema, es una representación del

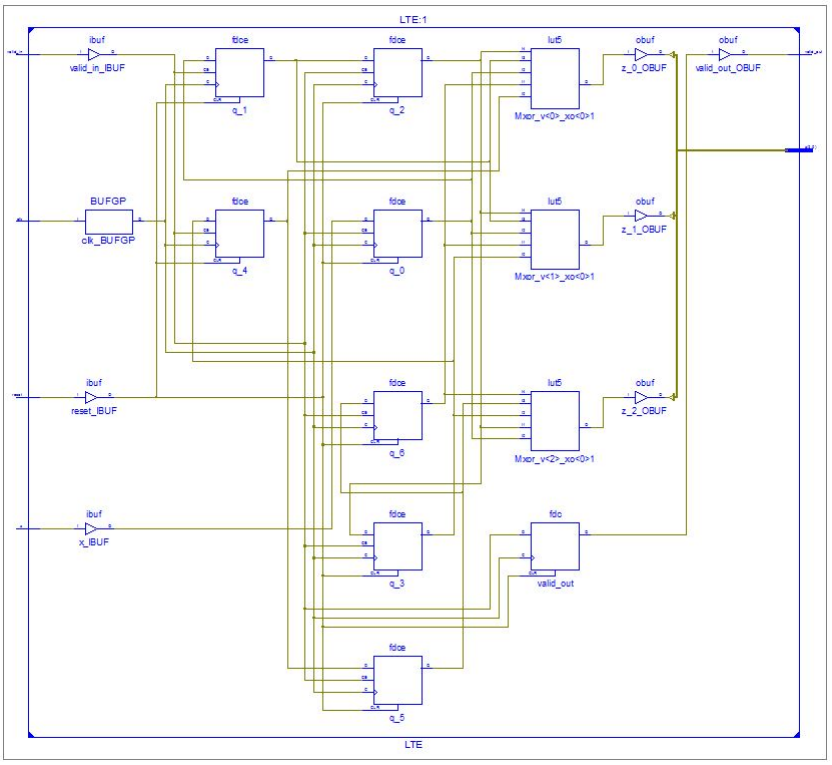

Figura 3.6: Esquemático RTL del codificador Convolucional LTE. Fuente: Propia

diseño en términos de elementos lógicos, como por ejemplo, flip-flops, inversores, compuertas XOR, entre otros.

3.3. Fase 3. Implementación

En esta fase, se consideraron todos los aspectos relevantes que permitieron la implementación del código diseñado en VHDL en la tarjeta FPGA Spartan-6.

3.3.1. Implementando los códigos en la FPGA

Para lograr la implementación de los códigos en la tarjeta, primero se tomó en cuenta las características de la FPGA Spartan-6, posteriormente se establecieron los puertos de entrada y salida, luego, en el diseño en VHDL se realizó la asignación de

pines a través del archivo UCF, finalmente se cargó el archivo a la tarjeta mediante el software Adept.

3.3.1.1. Asignación de pines

Después de haber consultado las características de la tarjeta [17], se procedió a identificar los pines de entrada y salida correspondiente a la entidad del codificador.

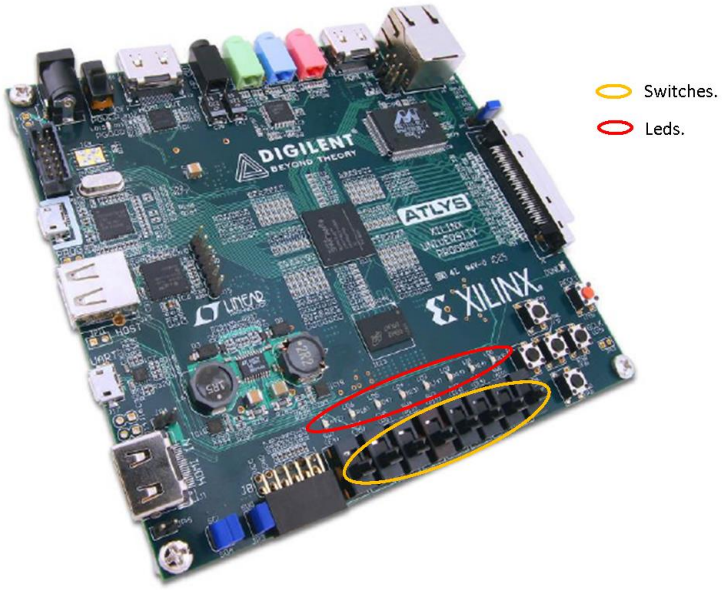

Figura 3.7: Switches y Leds de la tarjeta Atlys Fuente: [17]

De la figura 3.7, se observa que las entradas asociadas a la entidad del codificador, se ejecutan con los switches, mientras que las salidas se muestran por los leds.

Para poder apreciar las salidas a través de los leds se tuvo que dividir la frecuencia a 50Mhz generando un modulo en VHDL, obteniendo un nuevo programa principal que lo incluye. El código en VHDL que permite la división de frecuencia, así como también; los nuevos códigos diseñados usando el modulo divisor para cada codificador convolucional, se muestran en el **Anexo B**.

Finalmente se asignaron los pines a través de la creación del archivo UCF, donde se asoció en la tarjeta, las entradas del codificador a los switches y las salidas a los leds. En la tabla 3.10 se observa la programación para generar el archivo UCF.

Tabla 3.10: Asignación de pines mediante el archivo UCF.

```
NET "x" LOC = "A10"; # Entrada
NET "valid_in" LOC = "D14"; # Entrada
NET "reset" LOC = "C14"; # Entrada
NET "clk"   LOC = "L15"; # Entrada
NET "ena" LOC = "P15"; # Entrada
NET "z<0>" LOC = "D4";  # Salida
NET "z<1>" LOC = "P16"; # Salida
NET "z<2>" LOC = "N12"; # Salida
```

En la figura 3.8 se indican cuales switches y leds fueron asociados a la entidad del codificador, como se puede observar se tiene una nueva entrada `ena`, la misma; se activa en un nivel alto (1) y se desactiva en un nivel bajo (0), por ejemplo; en un nivel alto la codificación continúa, mientras que en un nivel bajo se detiene, almacenando su estado. El switche `ena` puede ser activado y desactivado en cualquier momento. El resto de las entradas de la entidad siguen cumpliendo la misma función como lo expuesto en 3.2.3.1.

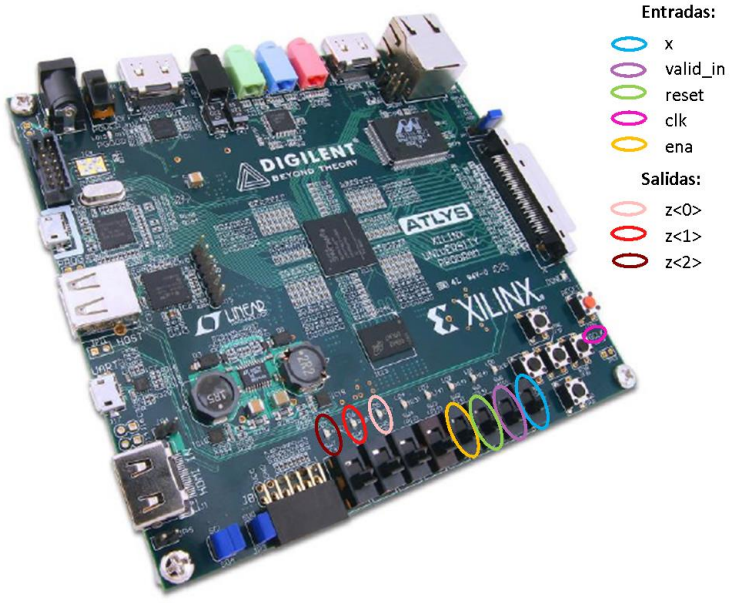

Figura 3.8: Switches y Leds de la tarjeta Atlys Fuente: [17]

3.3.1.2. Programando la FPGA

Luego de que se generó el archivo bitstream del diseño en VHDL, se cargó en la FPGA, mediante el software Adept. Para ello se siguieron los siguientes pasos:

1. Se conectó la tarjeta a una fuente de alimentación.
2. Se conectó el cable USB en el puerto Adept USB, el cual permite la transferencias de los archivos.
3. Se ejecutó el software Adept.
4. Se encendió la tarjeta mediante el switche de encendido.
5. Una vez que reconoció la tarjeta FPGA Spartan-6, en la interfaz del software se buscó la ubicación del archivo .bit y se programó en el dispositivo.

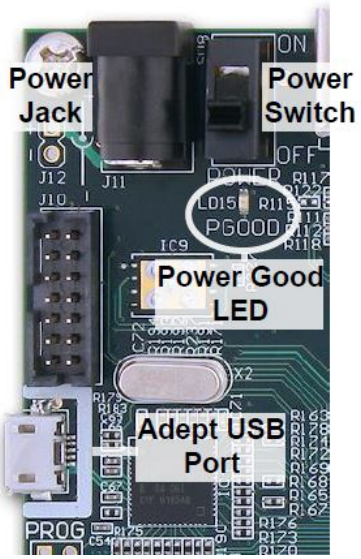

Figura 3.9: Puertos y switches para la configuración de la tarjeta atlys. Fuente: [17]

En la figura 3.9 se muestran los puertos y switches mencionados anteriormente [17].

3.4. Fase 4. Validación

Ya diseñados los códigos, e implementado cada uno de los algoritmos de codificación para las diferentes tecnologías inalámbricas en la tarjeta, solo resta validar que la programación creada cumpla con lo requerido para la codificación convolucional. En esta fase se comprobó el correcto funcionamiento de cada código convolucional para UMB, LTE, WiFi y WiMax diseñado en los lenguajes de programación Python y VHDL, se verificó en la tarjeta FPGA Spartan-6 y se generó el resumen de los recursos empleados en el dispositivo programable, además del reporte de potencia y la velocidad de respuesta.

3.4.1. Validación de códigos Python

La manera en que se validaron los resultados de la implementación en hardware fue mediante la programación en Python, es por ello que resultó de gran importancia asegurar el buen desempeño de los algoritmos de codificación, lo cual se realizó de dos maneras:

- Con ecuaciones que describen el sistema de codificación
- Con la herramienta Matlab como lenguaje soporte

Validación de códigos mediante ecuaciones que describen el sistema

En esta sección se obtuvieron las ecuaciones que describen a cada sistema de acuerdo a la estructura de codificación de cada tecnología seleccionada, para esto se tomaron tramas de ejemplo de 3, 5 y 7 bits respectivamente.

Validación en Matlab

La herramienta de software matemático Matlab ofrece entre sus prestaciones: la manipulación de matrices, de algoritmos y mucho más, por lo pronto; para este

proyecto sólo se trabajó en él como un lenguaje de soporte para el desarrollo de los algoritmos de codificación convolucional. De igual manera que en el apartado anterior, se propusieron ejemplos con las mismas tramas de bits de entrada (3, 5 y 7 bits).

3.4.2. Validación de códigos VHDL

Después de diseñados y sintetizados los codificadores convolucionales en VHDL, inició el proceso de validación a través de simulaciones usando la herramienta computacional en software Modelsim (simulador de Xilinx), que permitió observar el comportamiento de los diseños en HDL. Para ello, se realizó el siguiente procedimiento:

- Se generó primero un banco de pruebas en el lenguaje descriptor de cada codificador
- Se insertaron dos tramas aleatorias de 8 bits
- Con Modelsim se apreciaron las entradas, las salidas y los registros de desplazamiento de cada codificador

Finalizada la simulación, se insertaron en Python las mismas tramas de bits, y se verificaron las salidas. Por último, en la tarjeta FPGA Spartan-6 se introdujeron las tramas y se compararon las salidas con las obtenidas anteriormente. Si todas las salidas coinciden, queda demostrado los codificadores funcionan correctamente.

3.4.3. Análisis de los recursos usados en la tarjeta FPGA

Con Xilin Ise Design Suite, se logró la síntesis, el mapeo y el enrutamiento de cada codificador convolucional en VHDL, para posteriormente usarlos en la FPGA Spartan-6 XC6SLX45 de Xilinx. En general el software que realiza la síntesis asigna los recursos lógicos de manera automática, sin necesidad de la intervención del diseñador.

Por otra parte, los reportes generados para los codificadores convolucionales se dividieron en dos:

- Los codigos convolucionales que no disponen del modulo divisor de frecuencias. Mencionándolos como **codigos convolucionales.**
- Los codigos convolucionales que disponen del divisor de frecuencias, los cuales son los archivos .bit que se implementaron en la FPGA. Citándolos como **codigos convolucionales implementados en la FPGA.**

Ahora bien, para analizar los recursos usados en la tarjeta, el consumo de potencia y la velocidad en respuesta, se generó cada reporte como se detalla a continuación.

3.4.3.1. Reporte de recursos

El diseño pasó por tres fases de implementación antes de crear el reporte de lo recursos, estas fases fueron:

- **Translate**: Combina los archivos de diseño en una sola lista de conexiones (netlist).
- **Map**: Mapea símbolos lógicos de cada netlist, en componentes físicos (Slices y IOBs).
- **Place & Route**: Coloca y conecta los componentes en el chip, y genera un reporte de la sincronización de los datos.

Finalizada la síntesis del diseño, se creó el reporte de recursos utilizados, el cual se indicó, la cantidad de registros slice, los números de *LUT's* utilizados y los *IOB's* usados [32].

3.4.3.2. Consumo de potencia

El consumo de potencia se creó con la herramienta interactiva gráfica *Xpower Analyzer*, la cual fue empleada para el consumo de potencia estática y dinámica del diseño. Donde la potencia estática es la corriente de fuga del transistor cuando el dispositivo esta apagado y la dinámica se asocia con la actividad que ocurre en la tarjeta de los puertos entrada y salida a través de los switches.

3.4.3.3. Velocidad de respuesta

La velocidad de respuesta, se define como la frecuencia máxima en la que el codificador convolucional crea la palabra codificada de manera correcta.

La obtención de este parámetro se hizo de manera empírica, primero se revisó el datasheet [33], específicamente el tiempo de retraso que tiene una señal en el registro de desplazamiento usado en la salida, el cual fue de $t_{reg} = 1{,}35\text{ns}$. Este retraso se fijó en las salidas del código en VHDL, luego, en el banco de pruebas se fue aumentando la frecuencia del reloj y se realizó el proceso de validación expuesto en 3.4.2, hasta llegar al punto donde deje de codificar, de esta manera se obtuvo la frecuencia máxima en la que el codificador puede operar.

Capítulo IV

Análisis, interpretación y presentación de los resultados

Ya finalizadas cada una de las fases planteadas para la construcción de este trabajo, se presentan y analizan los resultados obtenidos, de acuerdo a la programación elaborada para el desarrollo de códigos convolucionales implementados en las tecnologías descritas anteriormente.

4.1. Validación de algoritmos de los codificadores convolucionales

4.1.1. Validación de códigos en Python

Como se especificó en la sección 3.4.1 del capitulo III, se validaron los códigos realizados en Python de las dos maneras allí mencionadas, señalando que; cada descripción y cada algoritmo para los codificadores convolucionales de las distintas tecnologías, se ejemplificaron con tramas de 3, 5 y 7 bits respectivamente. A modo ilustrativo, en este capitulo sólo se explicará para la trama de 3 bits. En el **Anexo A** se explican para las tramas ejemplo de 5 y 7 bits.

4.1.1.1. Validación de códigos mediante ecuaciones que describen el sistema

Descripción para LTE:

Sea el codificador convolucional usado en LTE con parámetros $R = 1/3$ y $K = 7$, el cual; tiene como estructura de codificación la expuesta en la figura 3.1 de la sección 3.1.3.

Donde las salidas están dadas por:

$$z_0 = q_0 + q_2 + q_3 + q_5 + q_6 \rightarrow 133_8$$
$$z_1 = q_0 + q_1 + q_2 + q_3 + q_6 \rightarrow 171_8$$
$$z_2 = q_0 + q_1 + q_2 + q_4 + q_6 \rightarrow 165_8$$

Por tanto, el estado del codificador viene dado como:

$$q = (q_0, q_1, q_2, q_3, q_4, q_5, q_6)$$

Ya que las salidas del codificador dependen de la secuencia de entrada, además de que sus registros de desplazamientos están iniciados en cero, se propuso una secuencia de entrada `110` para obtener su correspondiente salida, explicando en detalle la entrada de cada bit:

$$n_1 \rightarrow 1 \qquad n_2 \rightarrow 1$$
$$q = 1000000 \qquad q = 1100000$$
$$z_0 = 1+0+0+0+0+0 = 1 \qquad z_0 = 1+0+0+0+0+0 = 1$$
$$z_1 = 1+0+0+0+0+0 = 1 \qquad z_1 = 1+1+0+0+0+0 = 0$$
$$z_2 = 1+0+0+0+0+0 = 1 \qquad z_2 = 1+1+0+0+0+0 = 0$$
$$Salida = 111 \qquad Salida = 100$$

$$n_3 \to 0$$
$$q = 0110000$$
$$z_0 = 0 + 1 + 0 + 0 + 0 = 1$$
$$z_1 = 0 + 1 + 1 + 0 + 0 = 0$$
$$z_2 = 0 + 1 + 1 + 0 + 0 = 0$$
$$Salida = 100$$

De este modo, el resultado de la descripción arrojó como salida:

$$z = [111100100]$$

Descripción para UMB:

Sea el codificador convolucional usado en UMB con indice $R = 1/3$ y $K = 9$, el cual; tiene como estructura de codificación la expuesta en la figura 3.2 de la sección 3.1.3.

Donde las salidas están dadas por:

$$z_0 = q_0 + q_2 + q_3 + q_5 + q_6 + q_7 + q_8 \to 557_8$$
$$z_1 = q_0 + q_1 + q_3 + q_4 + q_7 + q_8 \to 663_8$$
$$z_2 = q_0 + q_1 + q_2 + q_5 + q_8 \to 711_8$$

Por tanto, el estado del codificador viene dado como:

$$q = (q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8)$$

El codificador con los registros en cero, se propuso una secuencia de entrada `110` para obtener su correspondiente salida, explicando en detalle la entrada de cada bit:

$$
\begin{aligned}
n_1 &\to 1 \\
q &= 100000000 \\
z_0 &= 1+0+0+0+0+0+0+0 = 1 \\
z_1 &= 1+0+0+0+0+0+0+0 = 1 \\
z_2 &= 1+0+0+0+0+0+0+0 = 1 \\
Salida &= 111
\end{aligned}
\qquad
\begin{aligned}
n_2 &\to 1 \\
q &= 110000000 \\
z_0 &= 1+0+0+0+0+0+0+0 = 1 \\
z_1 &= 1+1+0+0+0+0+0+0 = 0 \\
z_2 &= 1+1+0+0+0+0+0+0 = 0 \\
Salida &= 100
\end{aligned}
$$

$$
\begin{aligned}
n_3 &\to 0 \\
q &= 011000000 \\
z_0 &= 0+1+0+0+0+0+0+0 = 1 \\
z_1 &= 0+1+0+0+0+0+0+0 = 1 \\
z_2 &= 0+1+1+0+0+0+0+0 = 0 \\
Salida &= 110
\end{aligned}
$$

De este modo, el resultado de la descripción arrojó como salida:

$$z = [111100110]$$

Descripción para WiFi:

Sea el codificador convolucional usado en WiFi con indice $R = 1/2$ y $K = 7$, el cual; tiene como estructura de codificación la expuesta en la figura 3.3 de la sección 3.1.3.

Donde las salidas están dadas por:

$$
\begin{aligned}
z_0 &= q_0 + q_2 + q_3 + q_5 + q_6 \to 133_8 \\
z_1 &= q_0 + q_1 + q_2 + q_3 + q_6 \to 171_8
\end{aligned}
$$

Por tanto, el estado del codificador viene dado como:

$$q = (q_0, q_1, q_2, q_3, q_4, q_5, q_6)$$

El codificador con los registro en cero, así que se propuso una secuencia de entrada `110` para obtener su correspondiente salida, explicando en detalle la entrada de cada bit:

$$\begin{aligned} n_1 &\to 1 \\ q &= 1000000 \\ z_0 &= 1+0+0+0+0+0=1 \\ z_1 &= 1+0+0+0+0+0=1 \\ Salida &= 11 \end{aligned} \qquad \begin{aligned} n_2 &\to 1 \\ q &= 1100000 \\ z_0 &= 1+0+0+0+0+0=1 \\ z_1 &= 1+1+0+0+0+0=0 \\ Salida &= 10 \end{aligned}$$

$$\begin{aligned} n_3 &\to 0 \\ q &= 0110000 \\ z_0 &= 0+1+0+0+0+0=1 \\ z_1 &= 0+1+1+0+0+0=0 \\ Salida &= 10 \end{aligned}$$

De este modo, el resultado de la descripción arrojó como salida:

$$z = [111010]$$

Descripción para WiMax:

Sea el codificador convolucional usado en WiMax con indice $R = 1/5$ y $K = 7$, el cual; tiene como estructura de codificación la expuesta en la figura 3.4 de la sección 3.1.3.

Donde las salidas se encuentran dadas por:

$$\begin{aligned} z_0 &= q_0+q_1+q_2+q_3+q_6 \to 171_8 \\ z_1 &= q_0+q_2+q_3+q_5+q_6 \to 133_8 \\ z_2 &= q_0+q_1+q_2+q_4+q_6 \to 165_8 \\ z_3 &= q_0+q_3+q_4+q_5+q_6 \to 117_8 \\ z_4 &= q_0+q_1+q_4+q_5+q_6 \to 127_8 \end{aligned}$$

Por tanto, el estado del codificador viene dado como:

$$q = (q_0, q_1, q_2, q_3, q_4, q_5, q_6)$$

El codificador con los registros en cero, así que se propuso una secuencia de entrada `110`, para obtener su correspondiente salida, explicando en detalle la entrada de cada bit:

$$\begin{aligned} n_1 &\to 1 \\ q &= 1000000 \\ z_0 &= 1+0+0+0+0+0 = 1 \\ z_1 &= 1+0+0+0+0+0 = 1 \\ z_2 &= 1+0+0+0+0+0 = 1 \\ z_3 &= 1+0+0+0+0+0 = 1 \\ z_4 &= 1+0+0+0+0+0 = 1 \\ Salida &= 11111 \end{aligned} \qquad \begin{aligned} n_2 &\to 1 \\ q &= 1100000 \\ z_0 &= 1+1+0+0+0+0 = 0 \\ z_1 &= 1+0+0+0+0+0 = 1 \\ z_2 &= 1+1+0+0+0+0 = 0 \\ z_3 &= 1+0+0+0+0+0 = 1 \\ z_4 &= 1+0+0+0+0+0 = 1 \\ Salida &= 01011 \end{aligned}$$

$$\begin{aligned} n_3 &\to 0 \\ q &= 0110000 \\ z_0 &= 0+1+1+0+0+0 = 0 \\ z_1 &= 0+1+0+0+0+0 = 1 \\ z_2 &= 0+1+1+0+0+0 = 0 \\ z_3 &= 0+0+0+0+0+0 = 0 \\ z_4 &= 0+1+0+0+0+0 = 1 \\ Salida &= 01001 \end{aligned}$$

De igual manera que en las tecnologías anteriores, el resultado de la descripción arrojó como salida:

$$z = [111110101101001]$$

Este análisis expone el funcionamiento general de un codificador convolucional. Resultó claro que, en el codificador primeramente los registros se encontraron

inicializados en cero, entonces; la secuencia de datos de entrada se desplazó hacia la derecha en el momento que el primer bit ingresó, en ese instante y dependiendo del código convolucional en particular que fue usado, se efectuó la suma en módulo 2 de acuerdo a los polinomios generadores, presentando así la data codificada, posteriormente cuando el otro dato ingrese se determine nuevamente el estado del codificador y así sucesivamente hasta el fin del bloque de bits de datos, para luego ser recibida por el modulador digital (así como lo muestra la figura 2.1). Es recomendable que una vez finalizado el bloque de datos se ingresen k ceros para reinicializar los registros de desplazamiento del codificador a su estado original, de esta manera es conocido el estado en que inicia y finaliza el mismo, teniendo como ventaja, la facilidad en el proceso de decodificación, sin embargo; la tasa de codificación disminuye debido a la transmisión de bits empleados para el reinicio de la memoria del codificador.

4.1.1.2. Validación en Matlab

Código para las tecnologías inalámbricas LTE, UMB, WiFi y WiMax:

En la tabla 4.1 se expone los algoritmos codificadores realizados en Matlab, dichos resultados se muestran en conjunto con los obtenidos utilizando las ecuaciones descriptivas del sistema y con los originados mediante los códigos en Python, ilustrados en las figuras 4.1, 4.2, 4.3 y 4.4.

Tabla 4.1: Código para las tecnologías inalámbricas LTE, UMB, WiFi y WiMax respectivamente, para una trama de 3 bits. Fuente: Propia

Algoritmo del codificador para LTE:

```
Entrada = [1 1 0];
l = 7;
g = [133 171 165];
cod = poly2trellis(l,g);
Salida = convenc(Entrada,cod);
Entrada
Salida
```

Algoritmo del codificador para UMB:

```
Entrada = [1 1 0];
l = 9;
g = [557 663 711];
cod = poly2trellis(l,g);
Salida = convenc(Entrada,cod);
Entrada
Salida
```

Algoritmo del codificador para WiFi:

```
Entrada = [1 1 0];
l = 7;
g = [133 171];
cod = poly2trellis(l,g);
Salida = convenc(Entrada,cod);
Entrada
Salida
```

Algoritmo del codificador para WiMax:

```
Entrada = [1 1 0];
l = 7;
g = [ 171 133 165 117 127];
cod = poly2trellis(l,g);
Salida = convenc(Entrada,cod);
Entrada
Salida
```

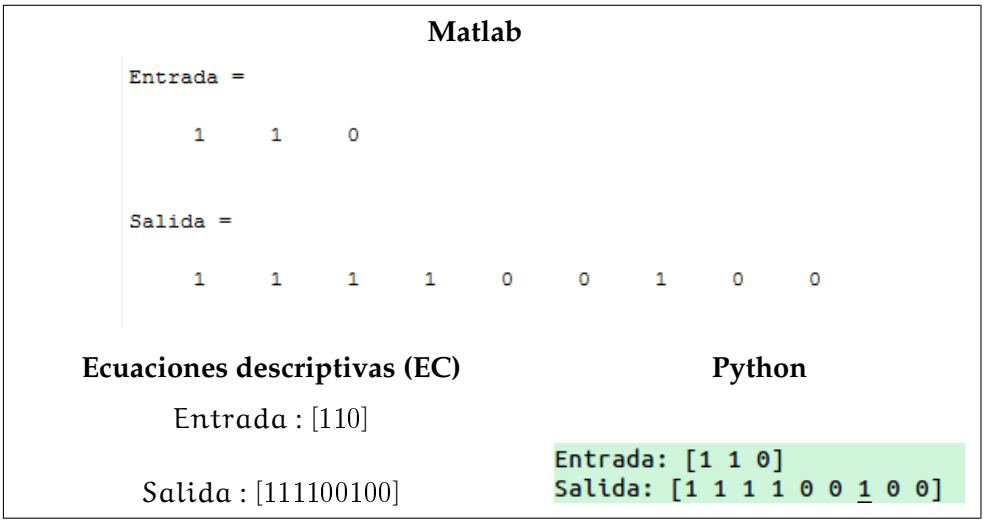

Figura 4.1: Salida codificada para `n = 3bits` utilizando EC, Matlab y Python para LTE. Fuente: Propia

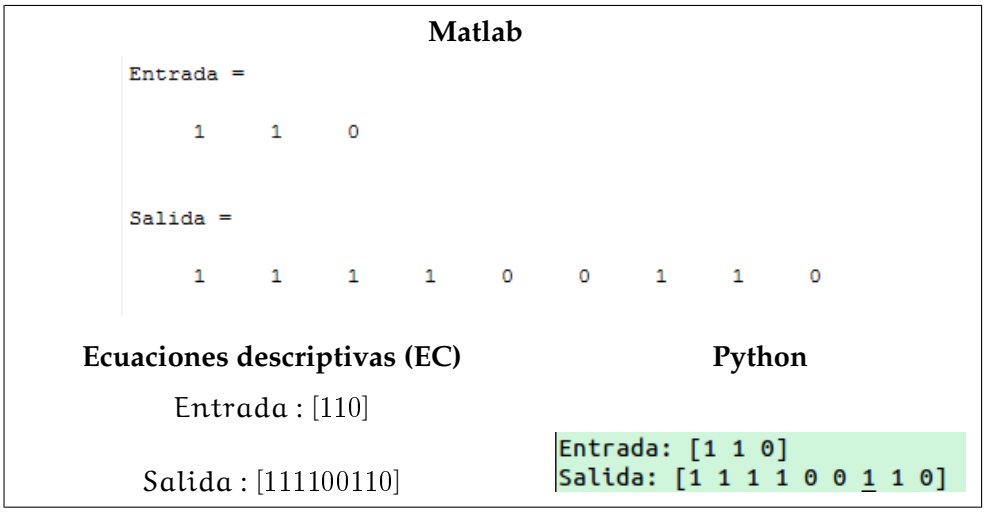

Figura 4.2: Salida codificada para `n = 3bits` utilizando EC, Matlab y Python para UMB. Fuente: Propia

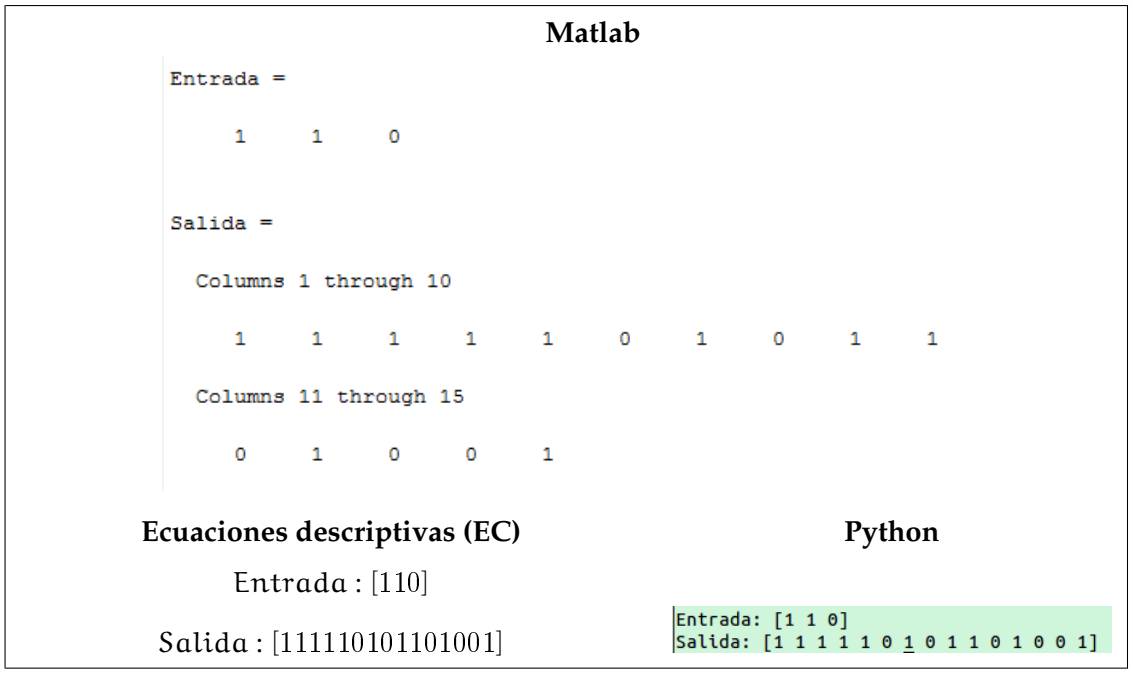

Figura 4.3: Salida codificada para `n = 3bits` utilizando EC, Matlab y Python para WiMax. Fuente: Propia

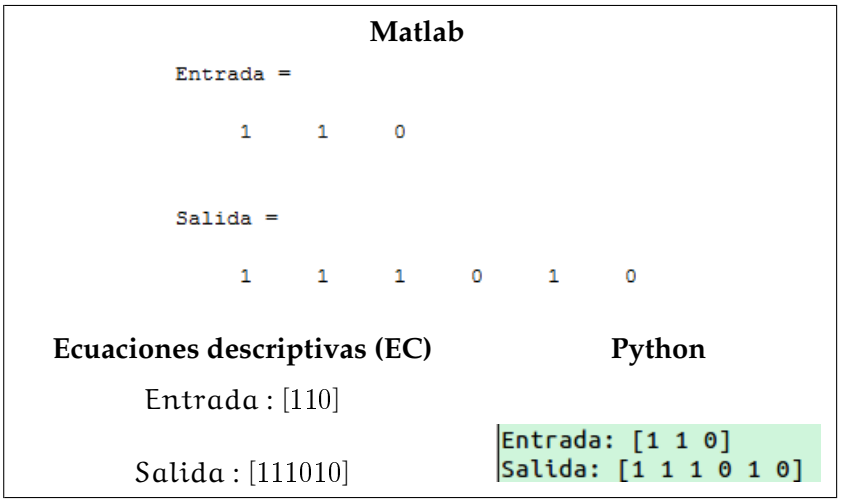

Figura 4.4: Salida codificada para `n = 3bits` utilizando EC, Matlab y Python para WiFi. Fuente: Propia

Al comparar los resultados, se verificó que las salidas de las distintas tecnologías para los sistemas de comunicaciones inalámbricos presentados en este trabajo de investigación, coinciden, por consiguiente; se puede decir entonces que los códigos realizados en el lenguaje interpretado Python cumplieron con las exigencias de validación y están aptos para ser usados como soporte para la comprobación de los algoritmos programados en el lenguaje descriptivo VHDL.

4.2. Validación de los códigos en VHDL

A continuación se muestra el proceso de validación y la descripción del funcionamiento para el codificador que se utiliza en el estándar LTE. La validación de los otros codificadores se exponen en el **Anexo B**.

4.2.1. Simulación en Modelsim

En el banco de pruebas se configura la frecuencia de reloj a $\texttt{freq} = 50\text{Mhz}$, se generan dos tramas de 8 bits a la misma frecuencia del reloj, la primera $T_1 = 10011011$ y la segunda $T_2 = 10011001$.

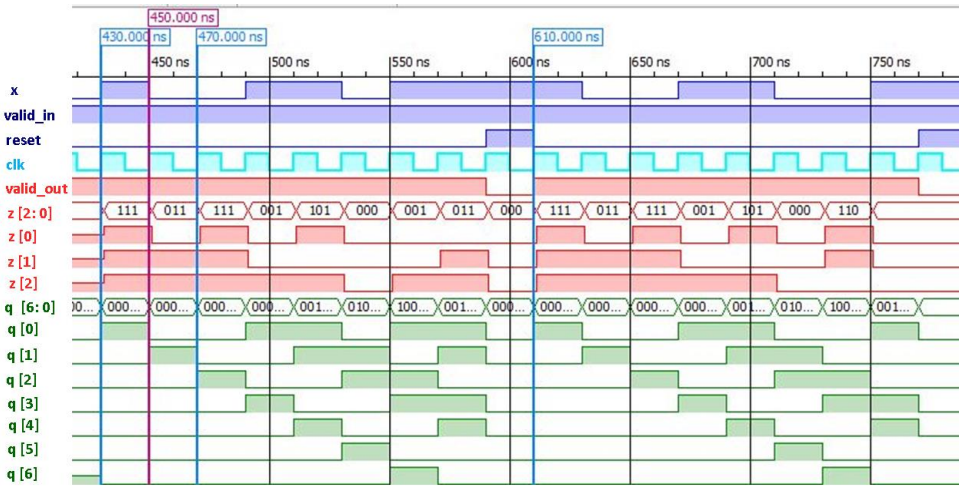

Figura 4.5: Simulación en Modelsim del codificador convolucional LTE. Fuente: Propia

En la figura 4.5, se identifican las señales en orden descendente de la siguiente manera:

1. **Señales de Entradas** (——) :
   - `x`
   - `valid_in`
   - `reset`
2. **Reloj** (——) :
   - `clk`
3. **Señales de salida** (——):
   - `valid_out`
   - `z[2:0]` (Vector de bits de salida)
   - `z[2]`
   - `z[1]`
   - `z[0]`
4. **Registros de Desplazamientos** (——) :

- `q[6:0]` (Vector de bits de registros de desplazamiento)
- `q[0]`
- `q[1]`
- `q[2]`
- `q[3]`
- `q[4]`
- `q[5]`
- `q[6]`

Ya identificadas las señales, se puede constatar de la figura 4.5 lo siguiente:

1. La inserción de la primera trama comienza a los $t = 430ns$, terminando aproximadamente a los $t = 590ns$, la segunda trama inicia a los $t = 610ns$, finalizando a los $t = 770ns$.
2. El reloj tiene un periodo de $T_clk = 20ns$, por lo tanto, el codificador opera a una frecuencia de $50Mhz$.
3. El bit de entrada `x` posee una duración de $20ns$, el mismo que el del reloj, por lo tanto la señal de entrada y el reloj están sincronizadas.
4. Entre la entrada `x` y la salida `z`, se observa un ligero retraso.
5. La señal cuadrada de **x**, concuerda con los $1's$ y $0'$ generados en cada trama.
6. La señal de salida `z[2:0]` es la composición de `z[2]`, `z[1]`, `z[0]` .
7. Por cada bit de entrada se obtienen tres de salida, coincidiendo con la tasa de codificación de LTE ($R = 1/3$).
8. Los registros de desplazamiento (`q[6:0]`), tienen la misma frecuencia que el reloj.
9. `q[0]` y `x` disponen de la misma señal de entrada, mientras que en los otros registros se observa como la cadena de bits de entrada se van desplazando, en cada periodo del reloj.

10. A los $590ns$, `reset`, toma un valor alto, reiniciando todos los registro de desplazamiento y salidas. En el siguiente periodo comienza la segunda trama.

11. El funcionamiento del codificador parece ser el correcto, debido a que esta desplazando las señales de entrada y generando datos codificados a la misma tasa de código que en LTE, sin embargo; para finalizar la validación se tiene que verificar que por cada bit de entrada (`x`), se obtiene la salida(`z[2:0]`) apropiada, con la ayuda de los códigos creados en Python.

4.2.2. Simulación en Python

Utilizando como software principal Python, se procede a introducir las dos tramas de LTE, especificada en el apartado anterior.

```
Trama de entrada: [1 0 0 1 1 0 1 1]
Trama de salida: [1 1 1 0 1 1 1 1 1 0 0 1 1 0 1 0 0 0 0 0 1 0 1 1]
```

Figura 4.6: Simulación de la trama T_1 del codificador convolucional LTE en Python. Fuente: Propia

En la figura 4.6, se comprueba que la primera trama contiene los mismos bits que T_1, posteriormente se comparan con los obtenidos en la figura 4.5, demostrando que la codificación tuvo éxito.

```
Trama de entrada: [1 0 0 1 1 0 0 1]
Trama de salida: [1 1 1 0 1 1 1 1 1 0 0 1 1 0 1 0 0 0 1 1 0 0 0 0]
```

Figura 4.7: Simulación de la trama T_2 del codificador convolucional LTE en Python. Fuente: Propia

Ahora se procede a comparar que en la segunda trama generada en Python posea los mismos bits que T_2, seguidamente se verifica la salida, por lo tanto, se constata que la figura 4.7 y 4.5 coinciden en la entrada, como en los datos codificados, de este modo, se confirma que la codificación fue exitosa.

Finalmente para culminar el proceso que valida el codificador convolucional en LTE, se configura en la tarjeta Atlys, a traves de Digilent Adept, el archivo LTE.bit, el cual contiene toda la información necesaria para que se ejecute el diseño correctamente en el dispositivo, hecho esto; se introduce la cadena de bits T_1 a través de

los switches, inmediatamente se procede a verificar las salidas en los leds, comparándolas con las obtenidas en la figura 4.5 o 4.6, de igual manera se hace para la trama T_2.

En la figura 4.8, se muestra el proceso donde se inserta la trama de bits T_1, y se verificó nuevamente que la codificación fue exitosa, comprobando que el algoritmo en VHDL para el codificador convolucional LTE, funciona para tramas de bits de longitud variable.

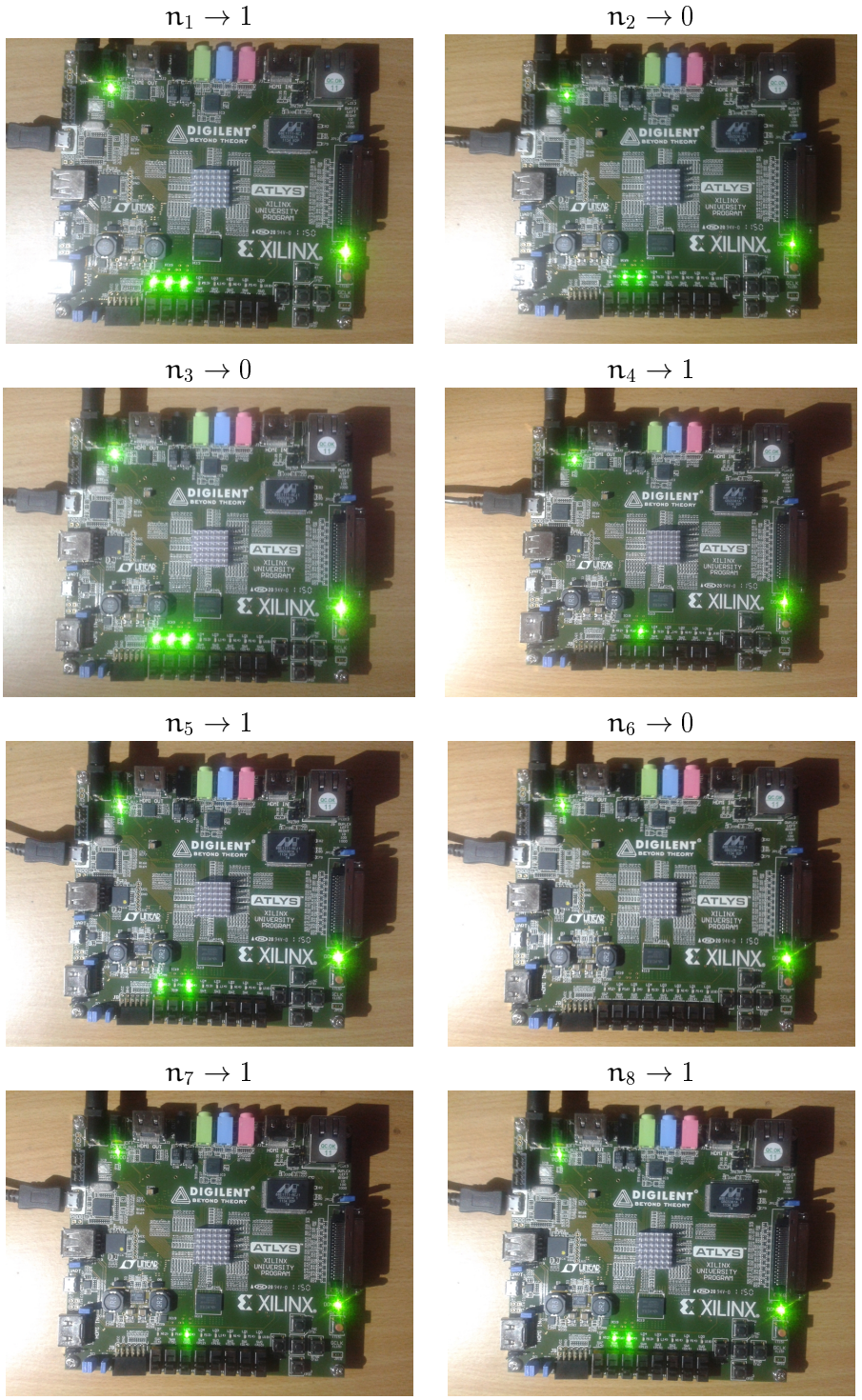

Figura 4.8: Salida codificada en la FPGA para LTE. Fuente: Propia

4.3. Análisis de los recursos en hardware

Como se expuso en el apartado 3.4.3 se procede a mostrar los resultados obtenidos.

4.3.1. Reporte de recursos

En las tablas creadas para el informe de los recursos utilizados en hardware, se detalla lo siguiente:

- **Disponible**: Indica la cantidad de celdas que contiene la tarjeta
- **U= Utilizadas**: Indica la cantidad de compuertas usadas en el proceso de síntesis de la FPGA Spartan-6 XC6SLX45 de Xilinx.
- **%**: Es el porcentaje de compuertas utilizadas en la FPGA, con respecto a las disponibles.
- **Number of Slice Registers**: Indica la cantidad de celdas usadas como registros.
- **Number of Slice LUTs**: Indica las celdas usadas como LUTs
- **Number of occupied Slices**: Indica la cantidad de celdas que son ocupadas en la FPGA.
- **Number of bonded Iobs**: Son los puertos de entrada y salida que se utilizan en la tarjeta.

4.3.1.1. Reportes de los recursos para los codificadores convolucionales

Se generó el reporte de los recursos para cada tecnología como se muestra en la tabla 4.2.

Tabla 4.2: Reporte de síntesis de los codificadores convolucionales de las diferentes tecnologías. Fuente: Propia

		LTE		UMB		WiFi		WiMax	
	Disponible	U	%	U	%	U	%	U	%
Number of Slice Registers	54576	7	1	9	1	7	1	7	1
Number of Slice LUTs	27288	6	1	8	1	4	1	8	1
Number of occupied Slices	6822	3	1	3	1	2	1	4	1
Number of bonded IOBs	218	8	3	8	3	7	3	10	4

De la tabla 4.2, se observó como varió la asignación de compuertas lógicas para cada codificador convolucional, ya que los mismos, poseen diferentes salidas y registros de desplazamientos. Cabe destacar que cada codificador utilizó el 1 % de la capacidad máxima de la tarjeta.

4.3.1.2. Reporte de los recursos para los codificadores convolucionales implementados en la FPGA

Los codificadores convolucionales implementados en la FPGA son los que disponen del modulo que disminuye la frecuencia, permitiendo observar la salida en los LEDs, a continuación se muestra el reporte de los recursos utilizados de los codificadores convolucionales de las diferentes tecnologías.

Tabla 4.3: Reporte de síntesis de los codificadores convolucionales implementados en la FPGA de las diferentes tecnologias. Fuente: Propia

		LTE		UMB		WiFi		WiMax	
	Disponible	U	%	U	%	U	%	U	%
Number of Slice Registers	54576	35	1	37	1	35	1	35	1
Number of Slice LUTs	27288	71	1	73	1	70	1	72	1
Number of occupied Slices	6822	20	1	21	1	20	1	22	1
Number of bonded IOBs	218	9	4	9	4	8	3	11	5

De la tabla 4.3, se apreció como aumentaron los recursos empleados en hardware, en comparación con los obtenidos en la tabla 4.2. Estos recursos son mayores debido al modulo en VHDL que divide la frecuencia, sin embargo; a pesar del incremento de la cantidad de compuertas utilizadas en la FPGA, aun la capacidad de la tarjeta sigue siendo de 1 %.

En vista de que la capacidad de la tarjeta es de 1 %, se diseñó un codigo en VHDL que permite el funcionamiento de los codificadores en paralelo, y con un multiplexor se puede seleccionar la salida del estándar que se desea observar mediante los leds, posteriormente se generó el reporte de síntesis y se implementó en la FPGA. En la figura 4.9, se aprecia el esquemático de la implementación.

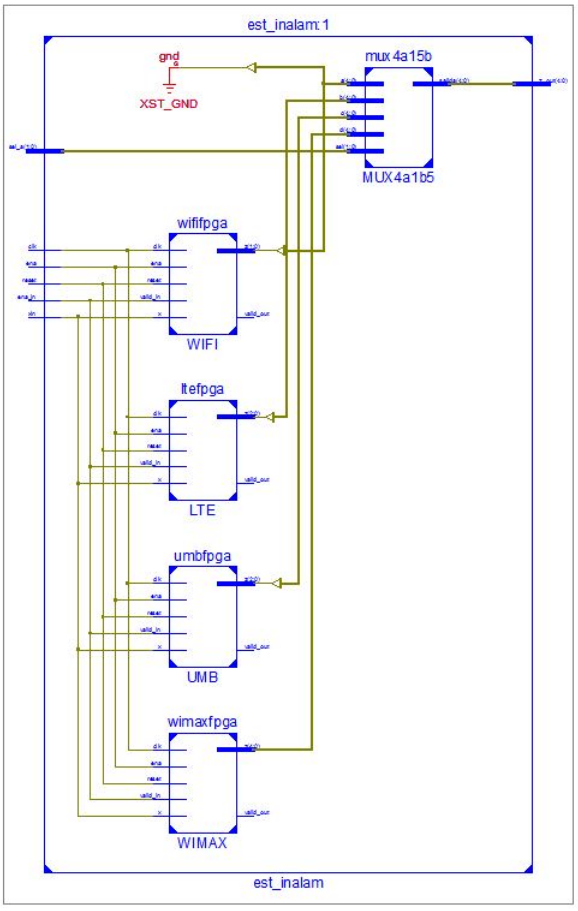

Figura 4.9: Esquemático de los 4 codificadores convolucionales implementados en la FPGA . Fuente: Propia

.

De la tabla 4.4, se hace notorio el incremento de las diferentes celdas de la FPGA, sin embargo la capacidad sigue siendo de 1 %. Por otro lado, Xilinx recomienda que la utilización de una tarjeta no debe superar el 80 % del consumo de recursos en hardware [34], por último queda demostrado que la tarjeta tiene la capacidad para la implementación de módulos más complejos.

Tabla 4.4: Reporte de síntesis de los codificadores convolucionales trabajando en paralelo implementados en la FPGA. Fuente: Propia

	Estandares Inlambricos		
	Disponible	Utilizadas	%
Number of slice Registers	54576	142	1
Number of Slice LUTs	27288	289	1
Number of occupied Slices	6822	84	1
Number of bunded IOBs	218	12	5

4.3.2. Reporte de consumo de potencia

La potencia total consumida en los códigos convolucionales, es la suma de la potencia dinámica con la estática, en las tablas 4.5, 4.6 y 4.7 se muestra la potencia absorbida de los diseños en VHDL.

Tabla 4.5: Potencia consumida de los codificadores convolucionales. Fuente: Propia.

	Total (mW)	Dinámica(mW)	Estática(mW)
LTE	40	4	36
UMB	41	5	36
WiFi	39	3	36
WiMax	41	5	36

Tabla 4.6: Potencia consumida de los codificadores convolucionales implementados en la FPGA

	Total (mW)	Dinámica(mW)	Estática(mW)
LTE	41	5	36
UMB	42	6	36
WiFi	40	4	36
WiMax	42	6	36

Tabla 4.7: Potencia de los codificadores convolucionales trabajando en paralelo implementados en la FPGA. Fuente: Propia

	Total(mW)	Dinámica(mW)	Estática(mW)
Estándares Inalámbricos	46	9	36

Comparando las tablas 4.5, 4.6 y 4.7 se constató que la potencia estática es la misma para todos los codificadores diseñados, así como también se demostró que

mientras más son los recursos en hardware usados en el diseño, mayor será la potencia consumida.

4.3.3. Velocidad de Respuesta

La velocidad de respuesta, se consigue aumentando la frecuencia e insertando la trama de bits, hasta llegar al punto donde el codificador no pueda operar. En la figura 4.10, se observa como se varió la frecuencia del codificador convolucional LTE. Para periodos mayores a $t = 1{,}35ns$ se apreció como se codifica la trama de manera correcta, mientras que; para un periodo menor a $t = 1{,}35ns$ las salidas del codificador son erróneas, por lo tanto; queda demostrado que el codificador convolucional para esta tecnología opera a una frecuencia máxima de $740{,}740Mhz$. De igual manera, este proceso se realizó para los codificadores convolucionales empleados en UMB, WiFi y WiMax, en el cual; la frecuencia máxima de operación es de $740{,}704Mhz$ para cada uno de ellos, por consiguiente; el método empleado para comprobar la velocidad de respuesta de los codificadores depende del modelo de la FPGA en que se implementen los codificadores convolucionales.

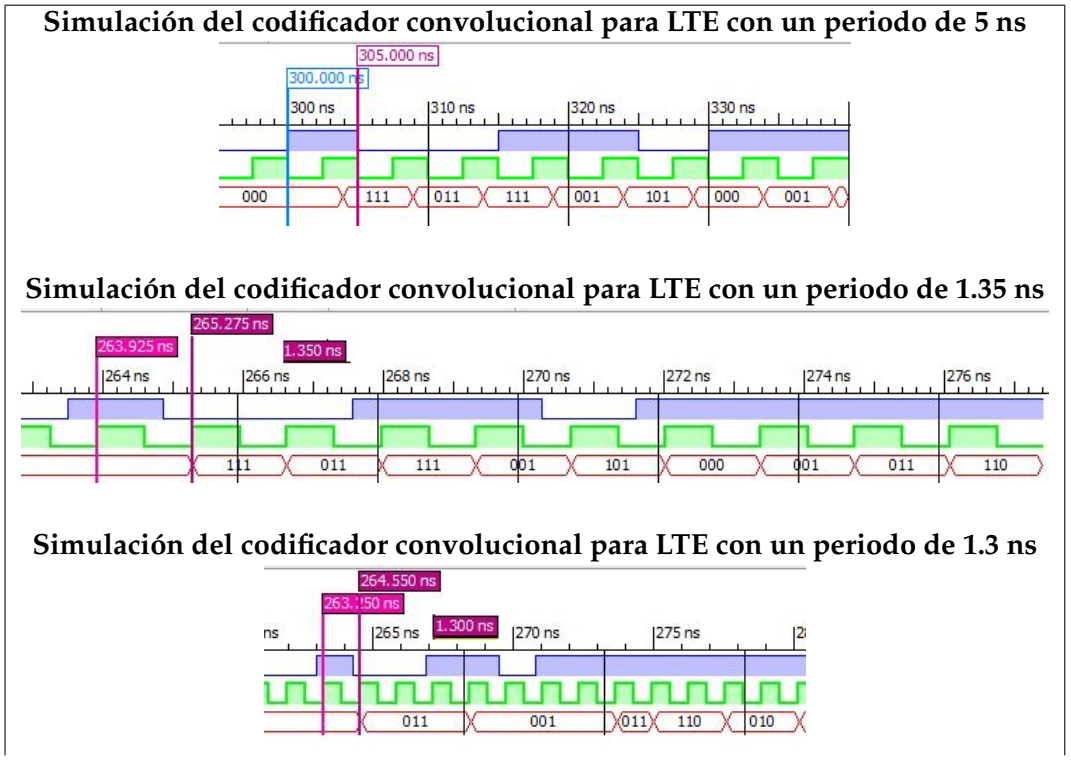

Figura 4.10: Simulación del codificador convolucional para LTE a diferentes periodos del reloj. Fuente: Propia

Capítulo V

Conclusiones y recomendaciones

5.1. Conclusiones

El aporte más representativo de la investigación fue el desarrollo de algoritmos de codificadores convolucionales para las tecnologías de comunicación inalámbrica bajo lenguaje VHDL implementados en la tarjeta FPGA Spartan-6, ya que al ser configurados en un HDL estándar, poseen características portables y reusables, permitiendo que se implementen con cualquier herramienta de síntesis y en cualquier soporte de hardware que lo acepte, de modo que si por una razón u otra es necesario cambiar de fabricante de FPGA, sea una transición eficiente.

Por lo tanto los códigos acá presentados abren la puerta a un campo de investigación adaptativa, permitiendo el desarrollo, actualización, optimización e hibridación tanto en hardware como en software, de los sistemas de comunicaciones.

Con base en la experiencia obtenida en el desarrollo de cada una de las etapas de la investigación se concluye que:

- Python a pesar de que ejecuta los programas instrucción por instrucción comparándolo con otros lenguajes como Java, C++ y Matlab, funciona como lenguaje de programación de alto nivel, demostrando que puede ser usado para

la creación de algoritmos correctores para sistemas de comunicaciones inalámbricos.

- Python cuenta con múltiples librerías que facilitan los cómputos matemáticos necesarios para la creación de códigos correctores, en este caso; se contó con la librería *numpy*, la cual permitió el correcto funcionamiento de los registros de desplazamiento de cada tecnología implementada, de acuerdo a sus estructuras de codificación.

- Como los codificadores convolucionales son codificadores correctores con memoria, fue necesario validar los códigos diseñados en Python mediante ecuaciones descriptoras, debido a que ellas rigen el comportamiento del dispositivo, definiendo las salidas del sistema de acuerdo a la suma en módulo 2 de los registros de desplazamiento.

- Los algoritmos diseñados para los codificadores convolucionales son un punto de partida en la línea de investigación y desarrollo de los cuatro sistemas de comunicaciones inalámbricos implementados en el dispositivo reconfigurable.

- Xilinx Desing Suite, es un software completo para cualquier familia de FPGA's de Xilinx, que permitió la codificación, consideraciones de prueba y verificación del diseño HDL. Ademas, posee la ventaja de que los diseños pueden ser simulados antes de ser fabricados, evitando perdida de tiempo y dinero en la creación de prototipos de hardware.

- Los circuitos diseñados en VHDL tendrán un consumo de potencia proporcional a la complejidad computacional o lógica del diseño, por tanto, se determinó que si la utilización de los recursos de la tarjeta FPGA aumenta, mayor será la potencia consumida.

- EL tiempo del reloj donde los datos de salidas aún son estables, dependerá del fabricante, del modelo y del tipo de FPGA donde se implemente el diseño.

- Se demostró que se cuenta con una FPGA Spartan-6 de alto rendimiento, lo que permite la inclusión de circuitos de comunicaciones más complejos.

- Se demostró que la codificación fue exitosa, ya que las salidas en el lenguaje interpretado, en el banco de prueba y en la tarjeta Atlys, son iguales, sin embargo; de suceder lo contrario, se recomienda verificar que los datos de entradas sean iguales para cada proceso (Python, test bench y tarjeta) debido a que, si las entradas coinciden, entonces se deduce que el codificador en VHDL no esta trabajando correctamente.

- El desarrollo de los códigos convolucionales, sirven de herramienta para un mejor entendimiento en las distintas asignaturas referidas al área de la codificación de canal, de acuerdo a la cátedra de la Escuela de Ingeniería en Telecomunicaciones e Ingeniería Eléctrica de la Universidad de Carabobo.

5.2. Recomendaciones

Es importante destacar que los modelos en descripción en hardware diseñados para los codificadores convolucionales de las diferentes tecnologías inalámbricas se mantienen abiertos a futuras optimizaciones de eficiencia y actualizaciones, además; de acuerdo al análisis realizado y a los objetivos alcanzados, se exponen las recomendaciones para mejoras de la presente investigación.

Recomendaciones con respecto al lenguaje de programación abierta Python:

- Desarrollar un programa que permita, generar la codificación de un codificador convolucional de acuerdo a sus parámetros de diseño, como la constante de restricción, los polinomios generadores y la tasa de código.

- Adición de los bloques siguientes que forman parte de los sistemas de comunicaciones (modulador, canal de ruido, demodulador, decodificador), con la finalidad de generar un sistema de comunicación completo.

- Diseño de una interfaz gráfica e interactiva como herramienta de aprendizaje de los códigos convolucionales con una amigable interacción con el usuario.

Recomendaciones con respecto al lenguaje de descripción en hardware HDL:

- Diseño de un módulo en VHDL que permita obtener las tramas de entradas de alguno de los puertos (USB, HDMI y Ethernet), que dispone la tarjeta FPGA Spartan-6.
- Diseños de módulos en VHDL que permita la visualización de la palabra codificada a través de los puertos HDMI que dispone la tarjeta FPGA Spartan-6.
- Desarrollo en HDL de los diferentes bloques de comunicaciones (modulador, canal de ruido, demodulador, decodificador), con el fin de realizar pruebas más reales de los sistemas de comunicaciones inalámbricos.
- Aprendizaje de las herramientas PlanAhead, User Constraint Editor, visor de esquemático de Xilinx Ise Design Suite, para obtener un diseño mas optimo.

Referencias Bibliográficas

[1] Eduardo Boemo Scalvinoni. Estado del arte de la tecnología fpga. Technical report, Instituto Nacional de Tecnología Industrial, Octubre 2005.

[2] N. Ismail N. K. Noordin, B. M. Ali and S.S. Jamuar. Adaptive techniques in orthogonal frequency division multiplexing in mobile radio environment. *International Journal of Engineering and Technology*, 1:115–123, 2004.

[3] Ing. Adrés Simone P. Ing. Aída Pérez R. Incorporación del lenguaje vhdl para la síntesis, descripción y simulación de circuitos lógicos digitales bajo el estándar ieee 1076-2002. Master's thesis, Sistemas Digitales de la Escuela de Ingeniería Eléctrica, de la Universidad de Carabobo, Mayo 2006.

[4] John G. Proakis and Masoud Salehi. *Digital Communications*. McGraw-Hil, 5ta edition, 2008.

[5] II Leon W. Couch. *Sistemas de Comunicación Digitales y Analógicos*. Pearson Educación, 7ma edition, 2008.

[6] Robert H. Morelos Zaragoza. *The Art of Error Correcting Coding*. Wiley, 1ra edition, 2002.

[7] Séguin G. On a class of convolutional codes. *Information Theory, IEEE Transactions*, 29(2):215–221, March 1983.

[8] Christian B. Schelegel and Lance C. Pérez. *Trellis and Turbo Coding*. IEEE Press Series, 1ra edition, 2004.

[9] Fu hua Huang. Evaluation of soft output decoding for turbo codes. Master's thesis, Virginia Polytechnic Institute and State University, May 1997.

[10] Hugo Díaz Jaramillo. and Osvaldo Hurtado Chávez. Implementación en software de un decodificador de viterbi para aplicaciones dvb-s. Master's thesis, Instituto Politécnico Nacional, Escuela Superior de Ingeniería Mecanica y Eléctrica, Junio 2013.

[11] IEEE 802.11 Working Group. Part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. Technical report, The Institute of Electrical and Electronics Engineers (IEEE))., February 2012. IEEE Std 802.11.

[12] IEEE 802.16 Working Group on Broadband Wireless Access. Ieee standard for wirelessman-advanced air interface for broadband wireless access systems. Technical report, The Institute of Electrical and Electronics Engineers (IEEE))., June 2012. IEEE Std 802.16.1.

[13] Mustafa Ergen. *Mobile Broadband Including WiMAX and LTE*. Springer, 1ra edition, 2009.

[14] Stefania Sesia, Issam Toufilk, and Matthew Baker. *The UMTS Long Term Evolution FROM THEORY TO PRACTICE*. Wiley, 2da, inluding release 10 for lte-advanced edition, 2011.

[15] Peter J. Ashenden. The vhdl cookbook. Master's thesis, Dept. Computer Science University of Adelaide South Australia, July 1990.

[16] Fernando Pardo Carpio. Vhdl lenguaje para descripción y modelado de circuitos. Master's thesis, Universidad de Valencia, Ingeniería Informatica, Octubre 1997.

[17] *Atlys Board Reference Manual*. Digilent, August 2013.

[18] *Adept™ Application User's Manual*. Digilent, May 2010.

[19] *Digilent Adept Suite User's Manual*. Digilent, November 2006.

[20] *Xilinx Design Tools: Release Notes Guide*. Xilinx, v2012.2 edition, July 2012.

[21] Mark Lutz. *Programming Python*. Oreilly, 4ta edition, 2010.

[22] Rául González Duque. *Python para Todos*. 2010.

[23] Fundación de Software Python. Acerca de python. http://www.python.org/about., 2014.

[24] Inc. The MathWorks. Pricing and licensing. http://es.mathworks.com/products/matlab/, 2015.

[25] 3GPP Technical Specification 36.212. Evolved universal terrestrial radio access (e-utra); multiplexing and channel coding). Technical report, ETSI 3rd Generation Partnership Project (3GPP)., Abril 2013. version 11.2.0 Release 11.

[26] 3GPP Technical Specification 36.212. Physical layer for ultra mobile broadband (umb) air interface specification). Technical report, ETSI 3rd Generation Partnership Project 2 (3GPP2)., Augustl 2007. version 2.0.

[27] Numpy developers. Numpy. http://www.numpy.org/, 2013.

[28] *ISE Simulator (ISim) In-Depth Tutorial*. Xilinx, v1.0 edition, April 2009.

[29] *Xililinx Ise WebPACK VHDL Tutorial*. Digilent, February 2010.

[30] *Xilinx Power Tools Tutorial Spartan-6 and Virtex-6 FPGAs*. Xilinx, .

[31] *Timing Closure User Guide*. Xilinx, .

[32] *Spartan-6 FPGA Configurable Logic Block*. Xilinx, v1.1 edition, February 2010.

[33] *Spartan-6 FPGA Data Sheet: DC and Switching Characteristics*. Xilinx, v3.1.1 edition, January 2015.

[34] *Synthesis and Simulation Desing Guide*. Xilinx, 10.1 edition, .

Anexo A

Python

Anexo B

VHDL