UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

FACULTAD DE INGENIERÍA ELECTRÍCA

Implementación de un algoritmo CORDIC en un FPGA

TESIS

Que para obtener el título de

INGENIERA EN ELECTRÓNICA

Presenta

Elizabeth Rivera Trigueros

Asesor

M.C. Alberto Carlos Salas Mier

Enero, 2018

Morelia, Michoacán.

RESUMEN

En esta tesis se presenta la documentación, simulación e implementación de un algoritmo CORDIC, utilizando un PLD *(Logic Programmable Device)*. Se muestran los fundamentos de los temas para el desarrollo de la implementación del algoritmo pertenecientes al Álgebra Lineal y Sistemas Digitales, entre los cuales están: sistemas de ecuaciones, matrices, matriz de rotación, espacio vectorial, operaciones lógicas básicas, tablas de verdad y sistemas secuenciales.

Se explica el funcionamiento del algoritmo mediante la rotación de vectores, con un sistema de ecuaciones que permite realizar iteraciones y así alcanzar la aproximación de un ángulo, dada la posición del vector. Se describe el algoritmo en el lenguaje Verilog, y se realizan simulaciones, se implementa en un Cyclone III EP3C16F484C6N para mostrar su funcionamiento. Para ello se diseña un sistema de prueba utilizando la terjeta de experimentación DE0.

PALABRAS CLAVE

CORDIC, Field Programmable Gate Array, Verilog HDL, Quartus II, COordinate Rotation DIgitaL Computer, Sistemas Digitales, Matriz de rotación, Dispositivo Lógico Programable.

ABSTRACT

In this thesis, the documentation, simulation and implementation of a CORDIC algorithm are presented, using a PDL *(Programmable Logic Device)*. The fundamentals topics of Linear Algebra and Digital Systems needed for the algorithms implementation are described. Among these concepts are equation systems, matrices, the rotation matrix, vector spaces, logic-based operations, truth tables and sequential systems.

The algorithm's operation is explained through rotation vectors, with an equations system that allows iterations and, thus, achieve angle approximations given the position vector. Verilog lenguage is used to describe the algorithm and simulations are performed. The algorithm was implemented usign a Cyclone III EP3C16F484C6N to show its performance. In order to test the algorithm, a testbed is designed usign the DE0 experimentation board.

AGRADECIMIENTOS

Antes que nada quiero agradecer a Dios por permitirme llegar hasta esta etapa de mi vida, y lograr una meta más, por darme la fuerza para seguir adelante. Agradezco infinitamente a mis padres José Manuel Rivera y Maricela Trigueros, por los sacrificios hechos, su confianza y apoyo incondicional, gracias a mis hermanas, en especial a Andrea Rivera por soportarme en momentos de estrés y mostrarme siempre su cariño y apoyo. Gracias a toda mi familia ya que sin su ayuda nada de esto hubiera sido posible.

Agradezco a mi prima Emelina Martínez por siempre estar cuando la necesito, ser esa hermana mayor y enseñarme a no darme por vencida, a pesar de las dificultades.

Gracias a todos mis profesores por brindarme una buena formación académica, pero en especial gracias a mi asesor de tesis el M.C. Alberto Carlos Salas Mier, por haberme permitido trabajar con él, por el tema de tesis propuesto y brindarme la información necesaria para llevarla acabo, sin su ayuda y tolerancia no hubiera sido posible llegar hasta aquí.

Gracias a la Facultad de Igeniería Eléctrica por formar ingenieros capaces de crear, desarrollar e implementar sistemas electrónicos y darme la oportunidad de crecer.

Finalmente gracias a la Universidad Michoacana de San Nicolás de Hidalgo por haberme permitido ingresar y alcanzar una meta más, por los apoyos brindados de condonaciones de reinscripción, entre otros.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

LISTA DE SÍMBOLOS Y ABREVIATURAS

CORDIC (COordinate Rotation Digital Computer)

CPLD*C* (Complex Programmable Logic Device)

FPGA (Field-Programmable Gate Array)

HDL (Hardware Description Language)

ISP (In-System Programing)

PAL (Programmable Array Logic)

PLA (Programable Logic Array)

PLD (Programmable Logic Devices)

RTL (Register Transfer Level)

$\oplus$ Suma de vectores

Θ Multiplicacion de vectores

ϵ Pertenencia de un elemento de un conjunto

$\vee$ Unión (“o” lógico)

$\prod$ Producto

DEDICATORIA

Para las personas más importantes de mi vida…

En especial dos de ellas que gracias a su ejemplo de fortaleza me enseñaron a no rendirme y dar lo mejor de mi siempre:

A esa mujer que ha sido, es y será la mujer que más amo, mi mamá que con su entrega y dedicación ha formado una maravillosa familia, y una hija que haría todo por ella.

A mi héroe y primer amor, mi papi que sin importar las adversidades simpre hace todo lo posible por sacar adelante a su familia, apoyando de manera incondicional a lograr nuestros sueños y metas.

Los amo familia.

CAPÍTULO I. INTRODUCCIÓN

1.1 Introducción

En este capítulo se presentan los antecedentes históricos acerca del algoritmo CORDIC, cómo es que surge y por qué, se definen los objetivos, se describen las motivaciones, se presenta la justificación y finalmente se describe cada capítulo realizado en esta tesis.

1.2 Reseña Histórica

Existen una gran cantidad de algoritmos eficientes que pueden emplearse para el cálculo de diversas funciones matemáticas, sin embargo sólo algunos de estos algoritmos pueden implementarse adecuadamente en hardware. Entre estos algoritmos se destaca una clase de los mismos basada únicamente en sumas y desplazamientos, colectivamente denominados algoritmos CORDIC. Esta clase de algortimos pueden utilizarse para calcular funciones trigonométricas, circulares, hiperbólicas y funciones lineales.

El cálculo de las funciones trigonométricas está basado en rotaciones de vectores. La denominación CORDIC, es un acrónimo de COordinate Rotation DIgital Computer *(Computadora Digital para Rotación de Coordenadas).*

El algorítmo CORDIC fue desarrollado originalmente como una solución digital para los problemas de navegación en tiempo real. El trabajo original es acreditado a Jack Volder [1] quien investigó el algoritmo CORDIC para el caso de rotaciones circulares. Ciertas extenciones a la teoría de CORDIC estan basadas en trabajos de Jhon Walther [2] entre otros, y proveen soluciones para una clase más amplia de funciones.

Como lo propuso Jack Volder [1], el algoritmo CORDIC realiza únicamente operaciones de suma y desplazamiento, lo cual lo hace idóneo para ser implementado en hardware. No obstante al implementar dicho algoritmo se puede

optar por diversas arquitecturas de diseño y se debe balancear la complejidad del circuito respecto del desempeño. Las arquitecturas utilizadas para implementar el algoritmo CORDIC son, bit-paralela desplegada, bit-paralela iterativa y bit-serie iterativa.

1.3 Objetivos

Los objetivos son: investigar el algoritmo de cómputo numérico CORDIC, implementarlo en el FPGA - EP3C16F484C6N por medio del lenguaje de descripción de hardware Verilog. Realizar simulaciones, para comprobar su funcionamiento.

1.4 Motivaciones

Es un algoritmo muy útil para el control de posición, además de que es un técnica rápida y fácil para el cálculo de funciones trigonométricas, en comparación con otras, por ejemplo la serie de Taylor.

Lograr la comprensión de las metodologías, técnicas y herramientas de software utilizadas, además de reforzar los conociminetos en el área de sistemas digitales.

1.5 Justificación

La principal razón por la cual se realizó este proyecto es la comprensión del funcionamiento del algoritmo CORDIC para el cálculo de funciones trigonómetricas. Entre sus ventajas se encuentran que es un algoritmo que no necesita de muchas iteraciones para aproximarse al valor, pero si se desea obtener una aproximación lo más exacta posible es necesario tener un número de

iteraciones mayor. Se caracteriza por ser rápido independientemente del número de iteraciones.

1.6 Descripción de capítulos

En el Capítulo 1 se presenta una breve reseña histórica del algoritmo CORDIC, se dan a conocer los objetivos, motivación y justificación para realizar esta tesis.

En el Capítulo 2 se describe el fundamento teórico de rotación de vectores y sistemas digitales, correspodiente a cada tema utilizado en esta tesis.

En el Capítulo 3 se describe la tecnología bajo la que se implementa y el lenguage de descripción.

En el Capítulo 4 se muestra el desarrollo e implementación del algoritmo.

Y finalmente en el Capítulo 5 se presentan las conclusiones y trabajos futuros.

CAPÍTULO II. MÁRCO TEÓRICO

2.1 Álgebra Lineal

Existen diferentes ramas de las matemáticas que ayudan a realizar algún trabajo en específico, en sí, éstas son indispensables en la vida cotidiana. En este capítulo se explica sobre los temas del Álgebra Lineal que son esenciales para éste tema de tesis, como son: sistemas de ecuaciones, rotación de matrices y espacios vectoriales.

Al buscar la palabra “lineal” en el diccionario se encuentra, entre otras definiciones, la siguiente: lineal: (del lat. *linealis*). 1. adj. Perteneciente o relativo a la línea. Sin embargo, en matemáticas la palabra “lineal” tiene un significado mucho más amplio [3]. El Álgebra Lineal es la rama de las matemáticas que se orienta a la generalización de las operaciones aritméticas a través de signos, letras y números, estudia conceptos tales como vectores, matrices, sistemas de ecuaciones lineales y de manera más formal espacios vectoriales y sus transformaciones lineales.

La importancia del Álgebra Lineal se eleva súbitamente con el uso y presencia actual de las computadoras y de manera generalizada todo el campo de la informática y la computación. La mayoría de los algoritmos computacionales usados en áreas como la optimización, necesitan de soluciones directas o indirectas de algún problema de Álgebra Lineal, ya que se requiere de un número grande de operaciones. El manejo de imágenes, sonido y digitalización de toda clase de información requiere de vectores o arreglos.

2.1.1 Sistemas de Ecuaciones

Los sistemas de ecuaciones están compuestos por dos o más ecuaciones que comparten dos o más incógnitas, estos pueden ser lineales o no lineales, lo que las diferencia es que, una ecuación lineal no está elevada a ninguna potencia

mientras que las ecuaciones no lineales se encuentran elevadas a alguna potencia; por tanto los sistemas de ecuaciones lineales son mucho más fáciles de resolver. Las soluciones de un sistema de ecuaciones son todos los valores que satisfacen todas las ecuaciones, o los puntos donde las gráficas de las ecuaciones se intersectan.

La forma de representar algebraicamente una recta en el plano xy es mediante la ecuación:

$$a_1 \mathrm{x} + a_2 \mathrm{y} = b \tag{2.1}$$

Una ecuación de este tipo se conoce como ecuación lineal en las variables *x* y *y*. En forma general, se define una ecuación lineal en las n variables $x_1, x_2, \dots, x_n$ como aquella que se puede expresar en la forma:

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = b \tag{2.2}$$

en donde $a_1, a_2, \dots, a_n$ son constantes reales.

Un sistema de ecuaciones lineales, se representa de la siguiente forma:

$$a_{11} \mathrm{x} + a_{12} \mathrm{y} = b_1 \tag{2.3}$$

$$a_{21} \mathrm{x} + a_{22} \mathrm{y} = b_2 \tag{2.4}$$

donde a_{11}, a_{12}, a_{21}, a_{22}, b_1 y b_2 son números dados. Cada una de estas ecuaciones corresponde a una línea recta. Una solución al sistema es un par de números, denotados por (x,y) que satisfacen al sistema. Las preguntas que surgen en forma natural son: ¿Tiene este sistema varias soluciones y, de ser así, cuántas? Existen diferentes formas de solucionar un sistema de ecuaciones lineal o no lineal, ya sea por el método gráfico, sustitución, igualación, reducción, combinación lineal etc.

En estos sistemas se presentan los casos donde se tienen múltiples soluciones, una única solución o no tiene solución. Una forma rápida de saber cuál podría ser el caso es mediante el método gráfico, cabe mencionar que éste

método sólo es viable para un sistema de máximo tres ecuaciones con tres incógnitas, ya que al aumentar el número de incógnitas, aumenta el número de dimensiones en el plano, es decir, a *n* incógnitas se tendrán *n* dimensiones, en la Figura 2. 1 se ilustran los casos posibles a obtener.

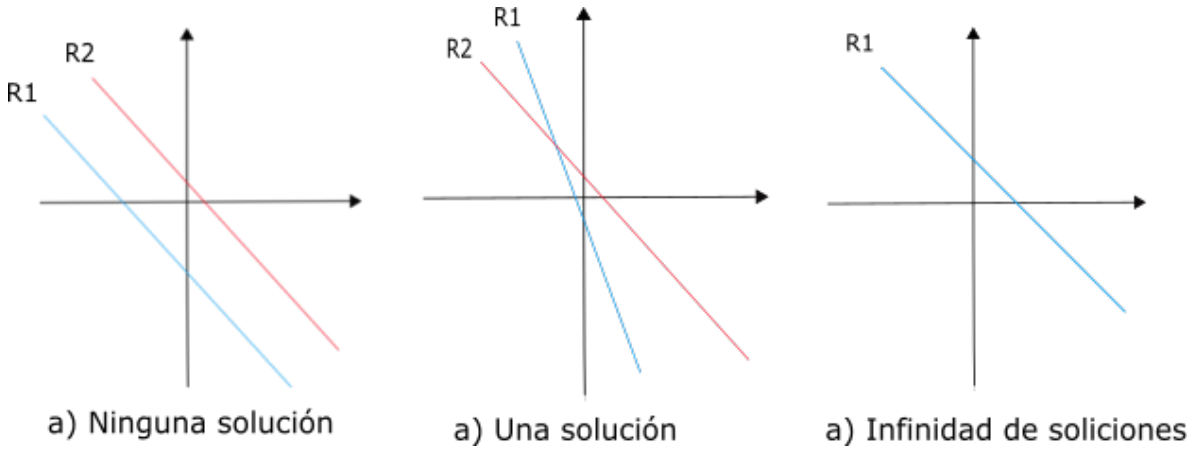

Figura 2. 1- Método gráfico

Cuando el número de ecuaciones es mayor a tres, es más complicado resolverlo por lo que se utilizan matrices.

2.1.2 Matrices

Otra forma fácil de resolver un sistema de ecuaciones es atrevés de una matriz. Ésta se define como un arreglo de números ordenados en filas y columnas, donde una fila es cada una de las líneas horizontales de la matriz y una columna es cada una de las líneas verticales, se representan dentro de paréntesis cuadrados o redondos, son denotadas por una letra mayúscula (A). El matemático inglés James Joseph Sylvester (1814 - 1897) fue el primero que utilizó el término “matriz” en 1850, para distinguir las matrices de los determinantes. La idea era que el término “matriz” tuviera el significado de madre de los determinantes [4].

Las dimensiones de una matriz siempre se dan con el número de filas por el número de columnas, y se representa como matriz de m x n. En general, este nuevo sistema se obtiene siguiendo una serie de pasos aplicando los siguientes

tipos de operaciones elementales sobre renglones a fin de eliminar sistemáticamente las incógnitas.

- Multiplicar uno de los renglones por una constante diferente de cero.
- Intercambiar dos de los renglones.
- Sumar un múltiplo de uno de los renglones a otro renglón.

2.1.3 Matriz de rotación

Las matrices de rotación definen algebraicamente una rotación en un espacio en tres dimensiones considerando un ángulo en el que está girando. Las matrices de rotación tienen las siguientes propiedades:

- Sus ejes de coordenadas son vectores ortogonales, es decir, que forman un ángulo de 90° entre ellos.
- Su determinante es 1.
- Si se saca la normal de cualquier vector perteneciente a la matriz, el resultado es uno, por lo que es una matriz unitaria.
- Al ser una matriz ortogonal su transpuesta es igual a su inversa.

La matriz de rotación se denota de la siguiente manera:

$$R = \begin{bmatrix} Cos\ \theta & -Sin\ \theta \\ Sin\ \theta & Cos\ \theta \end{bmatrix} \qquad (2.5)$$

En la Figura 2. 2 se ejemplifica una matriz de rotación. Una vez proporcionado un ángulo hace que el vector rote la cantidad de grados dada una dirección.

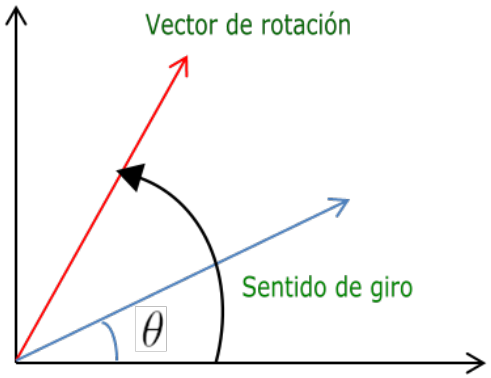

Figura 2. 2- Representación gráfica de una matriz de rotación

Si se desea rotar un vector $\begin{bmatrix} x \\ y \end{bmatrix}$ en un plano x, y se multiplica por la matriz de rotación de dos dimensiones, como se muestra a continuación:

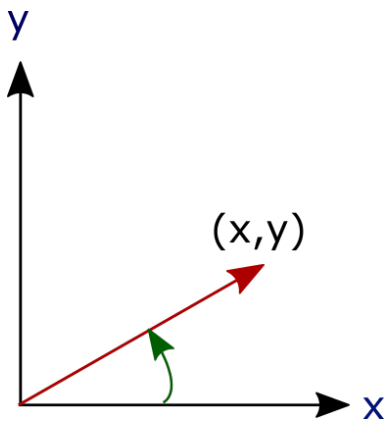

Figura 2. 3- Matriz de rotación de dos dimensiones

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} Cos\ \theta & -Sin\ \theta \\ Sin\ \theta & Cos\ \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \qquad (2.6)$$

Así las coordenadas (x',y') del punto (x,y) después de la rotación son:

$$\begin{matrix} x' \\ y' \end{matrix} = \begin{matrix} xcos\ \theta - ysin\ \theta \\ xsin\ \theta + ycos\theta \end{matrix} \qquad (2.7)$$

La dirección del vector rotado es antihoraria, si θ es positivo (por ejemplo 90°), y tiene sentido horario si θ es negativo (por ejemplo -90°) por lo tanto la matriz de rotación horaria es:

$$R(-\theta) = \begin{bmatrix} Cos\,\theta & Sin\,\theta \\ -Sin\,\theta & Cos\,\theta \end{bmatrix} \qquad (2.8)$$

Se observa que para el caso de dos dimensiones es el único caso no trivial donde el grupo de matrices de rotación es conmutativo, ésto quiere decir que no importa el orden en que se realicen varias rotaciones.

2.1.4 Espacio vectorial

El estudio de vectores y matrices es la médula del Álgebra Lineal. Éste comenzó esencialmente con el trabajo del matemático irlandés Sir William Hamilton (1805 - 1865). Su deseo de encontrar una forma de representar un cierto tipo de objetos en el plano y en el espacio lo llevó a descubrir lo que él llamó los cuaterniones (nuevos números que obedecían la propiedad conmutativa de la aritmética común). Esta noción condujo al desarrollo de lo que ahora se conoce como vectores. A lo largo de toda su vida y del resto del siglo XIX hubo un debate considerable sobre la utilidad de los cuaterniones y de los vectores. Al final del siglo el físico inglés Lord Kelvin escribió que los cuaterniones "aun cuando son bellamente ingeniosos, han sido un mal peculiar para todos aquellos que los han manejado de alguna manera y los vectores… nunca han sido de menor utilidad para ninguna criatura" [3].

Pero Kelvin estaba equivocado. En la actualidad casi todas las ramas de la física clásica y moderna se representan mediante el lenguaje de vectores, también se usan cada vez más en la ciencias biológicas y sociales [3].

La idea de vector R^n entró en las matemáticas de forma callada. Más aún, se puede decir que la idea de vector abstracto fue introducida por Euler (1707-1783), al resolver la ecuación diferencial que hoy se conoce como lineal de orden *n* homogénea [4].

Un espacio vectorial real V es un conjunto de vectores, junto con dos operaciones binarias denominadas suma y multiplicación por escalar. La suma de vectores es una regla o función que asocia a dos vectores, se suponen ***u*** y ***v*** un tercer vector, a éste se le representará como $\boldsymbol{u} \oplus \boldsymbol{v}$. La multiplicación es una regla

que asocia a un escalar y a un vector, ***c*** y ***u*** un segundo vector representado por ***c*** Θ ***u***, satisfacen diez axiomas que se enumeran a continuación:

i. Si ***x*** Є V y ***y*** Є V, entonces ***x*** + ***y*** Є V (cerradura bajo la suma).

ii. Para todo ***x***, ***y*** y ***z*** en V, (***x*** + ***y***) + ***z*** = x + (***y*** + ***z***) (ley asociativa de la suma de vectores).

iii. Existe un vector ***0*** Є V tal que para todo ***x*** Є V, ***x*** + ***0*** = ***0*** + ***x*** = ***x*** (vector cero o idéntico aditivo).

iv. Si x Є V, existe un vector –x en Є tal que x + (-x) = 0 (-x se llama inverso aditivo de x).

v. Si ***x*** y ***y*** están en V, entonces ***x*** + ***y*** = ***y*** + ***x*** (ley conmutativa de la suma de vectores).

vi. Si ***x*** Є V y α es un escalar, entonces *α**x*** Є V (cerradura bajo la multiplicación por un escalar).

vii. Si ***x*** y ***y*** están en V y α es un escalar, entonces α(***x*** + ***y***) = α***x*** + α***y*** (primera ley distributiva).

viii. Si ***x*** Є V y α y β son escalares, entonces (α + β) ***x*** = α***x*** + β***x*** (segunda ley distributiva).

ix. Si ***x*** Є V y α y β son escalares, entonces α (β***x***) = (αβ) ***x*** (ley asociativa de la multiplicación por escalares).

x. Para cada vector x Є V, Ix = x.

2.2 Sistemas Digitales

Los sistemas digitales resultan del conjunto de dispositivos destinados a la generación, transmisión, procesamiento o almacenamiento de señales digitales. Un sistema digital, es un sistema dependiente del tiempo cuyas entradas y salidas se dan en momentos determinados y sólo pueden tomar valores pertenecientes a un conjunto finito y discreto de valores. El término digital se deriva de la forma en

que las computadoras realizan las operaciones contando dígitos. Durante muchos años, las aplicaciones de electrónica digital se limitaron a sistemas informáticos. Hoy en día, la tecnología digital tiene aplicación en una amplia variedad de áreas además de la informática. Aplicaciones como la televisión, los sistemas de comunicaciones, de radar, sistemas de navegación y guiado, sistemas militares, instrumentación médica, control de procesos industriales y electrónica de consumo, usan todos ellos técnicas digitales. La tecnología digital ha progresado desde los circuitos de válvulas de vacío hasta los circuitos integrados y los microprocesadores [5].

Por ejemplo, la temperatura varía dentro de un rango continuo de valores. A lo largo de un día, ésta no varía por ejemplo entre 20°C Y 25°C de forma instantánea, sino que alcanza todos los infinitos valores que hay en ese intervalo. Ahora supongamos que simplemente se mide la temperatura cada hora. Lo que se tiene ahora son muestras que representan la temperatura en instantes discretos de tiempo (cada hora) a lo largo de un periodo de 24 horas. De esta forma se ha convertido de forma efectiva una magnitud analógica a un formato que ahora puede digitalizarse, representando cada valor mediante un código digital [5].

Los circuitos electrónicos se pueden dividir en dos amplias categorías: digitales y analógicos. La electrónica digital utiliza magnitudes con valores discretos, mientras que la electrónica analógica emplea magnitudes con valores continuos.

La mayoría de las cosas que se pueden medir cuantitativamente aparecen en la naturaleza en forma analógica. Para comprender mejor qué son los sistemas digitales, se realizan las siguientes definiciones.

1) Sistemas Continuos: operan con señales analógicas y su principal característica es presentar continuidad tanto en magnitud como en tiempo, registran y manipulan la información mediante señales analógicas como son: voltaje, corriente, presión, temperatura, posición o alguna variable física.

2) Sistemas Discretos: su principal característica es operar con señales discontinuas, éstas presentan su discontinuidad tanto en magnitud como en tiempo, en una señal discreta se tiene un número finito de combinaciones.

   Cabe mencionar que una señal es cualquier cantidad física que varía con el tiempo, espacio o cualquier otra variable.

2.2.1 Importancia de los sistemas digitales

Los sistemas digitales son importantes debido a los dispositivos que son destinados, dispositivos de generación, de transmisión, procesamiento o almacenamiento de señales digitales. Sus combinaciones de dispositivos son diseñados para manipular cantidades físicas o información que estén representadas en forma digital. La mayoría de las veces estos dispositivos son electrónicos pero también pueden ser mecánicos, magnéticos o neumáticos. Tienen una alta importancia en la tecnología moderna y sistemas de control automáticos.

Una de las principales ventajas de la representación digital con respecto a la representación analógica, es que los datos digitales pueden ser procesados y transmitidos de forma más fiable y eficiente que los datos analógicos. También los datos digitales disfrutan de una ventaja importante cuando es necesario su almacenamiento. El ruido (fluctuaciones de tensión no deseadas) no afecta a los datos digitales tanto como a las *señales analógicas.*

2.2.2 Operaciones lógicas básicas

En su forma más simple, la lógica es la parte del razonamiento que dice que una determinada proposición (oración con valor referencial o informativo, de la cual se puede predicar su veracidad o falsedad, es decir, que puede ser falsa o verdadera pero no ambas) es cierta si cumple ciertas condiciones. Las proposiciones se pueden clasificar como verdaderas o falsas. Muchas situaciones

y procesos que encontramos en nuestra vida cotidiana pueden expresarse como funciones proposicionales o lógicas. Dado que tales funciones son sentencias verdaderas/falsas o afirmativas/negativas, pueden aplicarse a los circuitos digitales, ya que estos se caracterizan por sus dos estados [5].

Una operación lógica asigna un valor (cierto o falso) a la combinación de condiciones (cierto o falso) de uno o más factores, y el resultado de una operación lógica puede ser, tan sólo, cierto o falso. Cuando se combinan varias proposiciones, se forman funciones lógicas o proposicionales. Por ejemplo, la proposición “la luz está encendida” sería cierta si “la bombilla no está fundida”. Por tanto, esta proposición lógica puede formularse de la manera siguiente: la luz está encendida sólo si la bombilla no está fundida. En este ejemplo, la primer sentencia es verdadera, si la segunda también lo es.

Hacia 1850, el matemático irlandés George Boole desarrolló un sistema matemático para formular proposiciones lógicas con símbolos, de manera que los problemas puedan formularse y resolverse de forma similar a como se hace en el álgebra ordinaria. El álgebra de Boole, como se le conoce hoy en día, encuentra aplicaciones en el diseño y el análisis de los sistemas digitales [5].

El término *lógico* se aplica a los circuitos digitales que se utilizan para implementar funciones lógicas. Existen varios tipos de circuitos lógicos que son elementos básicos que constituyen los bloques sobre los que se construyen los sistemas digitales más complejos, como por ejemplo una computadora [5].

Existen tres operaciones básicas, las cuales son:

- **NOT**

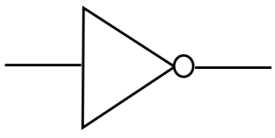

Figura 2. 4- Compuerta NOT

La operación NOT, cuyo símbolo se muestra en la Figura 2. 4, cambia de un nivel lógico al nivel lógico opuesto. Cuando la entrada está a nivel alto (1), la salida se pone a nivel a bajo (0) como se muestra en la Figura 2. 5. En cualquier

caso, la salida no es la misma que la entrada. La operación NOT se emplea mediante un circuito lógico conocido como inversor.

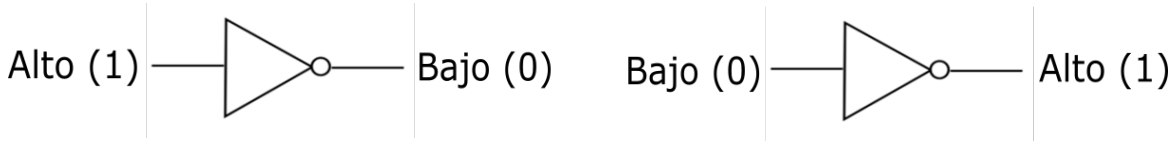

Figura 2. 5- Estados de la compuerta NOT

- **AND**

Figura 2. 6- Compuerta AND

La operación AND, cuyo símbolo se ilustra en la Figura 2. 6, genera un nivel alto sólo cuando todas las entradas están a nivel alto. Cuando cualquiera de las entradas o todas ellas están a nivel bajo, la salida se pone a nivel bajo como se observa en la Figura 2. 7. La operación AND se implementa mediante un circuito lógico conocido cómo puerta AND.

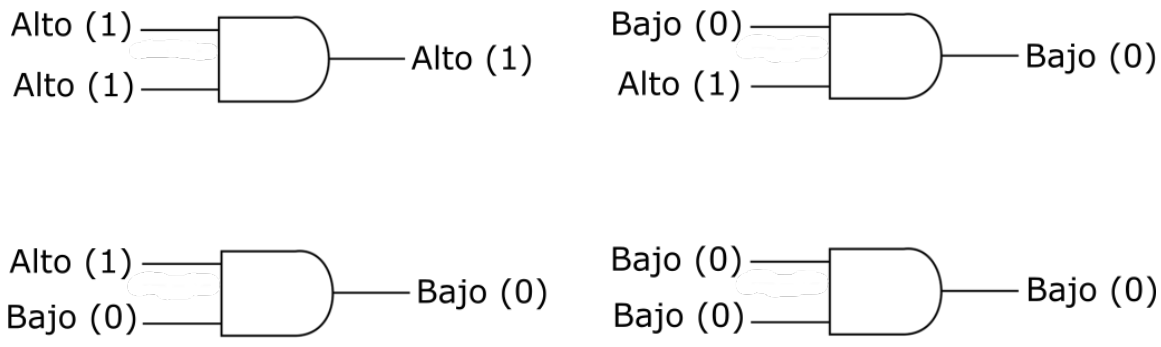

Figura 2. 7- Estados de la compuerta AND

- **OR**

Figura 2. 8- Compuerta OR

La operación OR, cuyo símbolo se muestra en la Figura 2. 8, genera un nivel alto cuando una o más entradas están a nivel alto. Cuando una de las entradas está a nivel alto o ambas entradas están a nivel alto, la salida es un nivel alto. Cuando ambas entradas están a nivel bajo, la salida será un nivel bajo en la Figura 2. 9 se muestra el comortamiento de ésta compuerta. La operación OR se implementa mediante un circuito lógico denominado puerta OR.

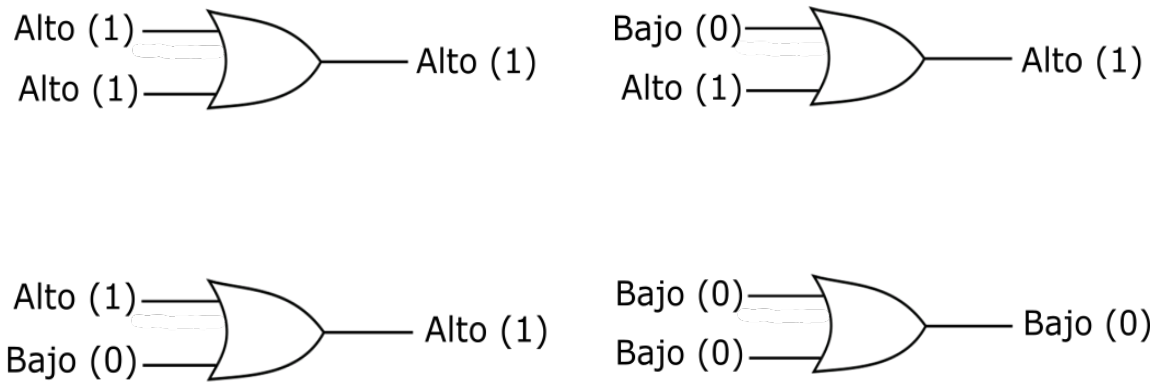

Figura 2. 9- Estados de la compuerta OR

2.2.3 Tablas de verdad

Las tablas de verdad son una buena forma de representar la función de una compuerta lógica. Muestran los estados de salida para cada posible combinación de los estados en sus entradas. Los simbolos 0 (falso) y 1 (verdadero) suelen usarse en las tablas de verdad.

La Tabla 2. 1 muestra la tabla de verdad de la compuerta NOT, la Tabla 2. 2 y Tabla 2. 3 muestran las tablas de verdad de las compuertas AND y OR respectivamente.

Tabla 2. 1- Compuerta NOT

Input A	Output Q
0	1
1	0

Tabla 2. 2- Compuerta AND

Input A	Input B	Output Q
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 2. 3- Compuerta OR

Input A	Input B	Output Q
0	0	0
0	1	1
1	0	1
1	1	1

2.3 Sistemas Secuenciales

Los circuitos secuenciales constan de un circuito combinacional al que se conectan elementos de memoria para formar una trayectoria de retroalimentación. Los elementos de almacenamiento son dispositivos capaces de guardar información binaria. La información binaria guardada en ellos en cualquier instante, define el estado del circuito secuencial en ese momento. El circuito secuencial recibe informacion binaria proveniente de entradas internas que, junto con el estado presente de los elementos de almacenamiento, determina el valor binario de las salidas. Estas entradas externas también determinan la condición para cambiar el estado en los elementos de almacenamiento. En la Figura 2. 10 se muestra un diagrama de bloques de un circuito secuencial [6].

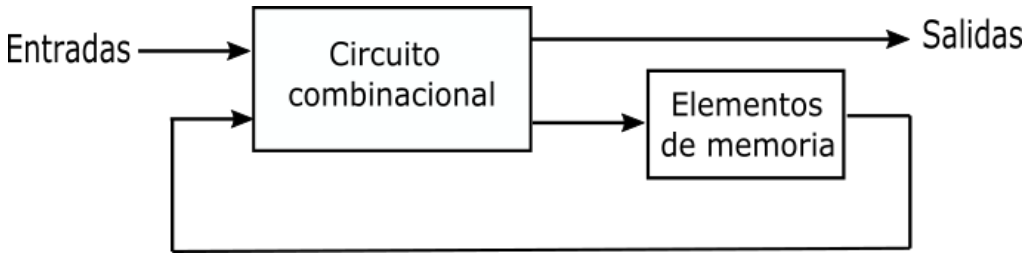

Figura 2. 10- Diagrama de bloques de un circuito secuencial

Hay dos tipos principales de circuitos secuenciales, y su clasificación es una función de la temporización de sus señales.

El comportamiento de un circuito secuencial *asíncrono* depende de las señales de entrada en cualquier instante y del orden en que cambian las entradas. Los elementos de almacenamiento comúnmente utilizados en circuitos secuenciales asíncronos son dispositivos retardadores.

Un circuito secuencial *síncrono* es un sistema cuyas señales en instantes de tiempo discretos definen su comportamiento, emplea señales que afectan a los elementos de almacenamiento en instantes discretos. La sincronización se logra con un dispositivo de temporización llamado *generador de reloj* que produce una

señal de reloj que tiene la forma de un tren periódico de pulsos. La señal de reloj se denota con los identificadores *clock* y *clk*.

Se llaman circuitos síncronos porque su actividad interna y la actualización resultante de los valores guardados se sincronizan con los pulsos de reloj. En la Figura 2. 11 se muestra el diagrama de bloques de un circuito secuencial síncrono y la Figura 2. 12 representa el pulso de reloj para dicho circuito.

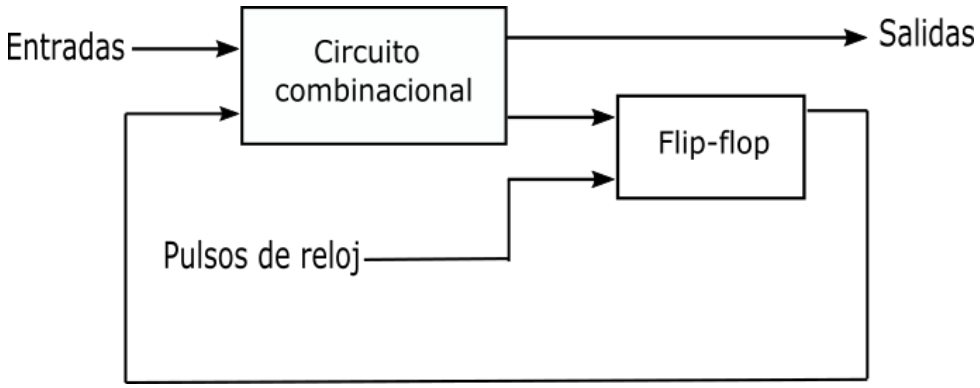

Figura 2. 11- Circuito secuencial síncrono con reloj

Figura 2. 12- Diagrama de temporización de pulsos de reloj

Los elementos de almacenamiento (memoria) utilizados en circuitos secuenciales de reloj se llaman FLIP-FLOP. Un FLIP-FLOP es un dispositivo de almacenamiento binario capaz de almacenar un bit de información. En un estado estable, la salida de un FLIP-FLOP es 0 o 1. Un circuito secuencial puede usar muchos FLIP-FLOPs para guardar tantos bits como sea necesario. El valor que se guarda en ellos cuando ocurre el pulso de reloj también esta determinado por las entradas al circuito y los valores actualmente guardados en el FLIP-FLOP (o ambos) [6].

CAPÍTULO III. HERRAMIENTAS DE HARDWARE Y SOFTWARE

3.1 FPGA

3.1.1 Dispositivos lógicos programables

Es posible fabricar chips que contengan relativamente grandes cantidades de circuitos lógicos con una estructura que no sea fija. Tales chips se introdujeron por primera vez en el decenio de 1970 y se llaman *dispositivos lógicos programables* (PLD, *Programmable Logic Device*) [7].

Un PLD es un chip de uso general para implementar circuitos lógicos. Incluye un conjunto de elementos de circuito lógico que pueden adaptarse de diferentes formas. Un PLD puede considerarse una “caja negra” que contiene compuertas lógicas e interruptores programables. Estos últimos permiten que las compuertas lógicas en el interior del PLD se conecten juntas para implementar el circuito lógico que se necesite. Un dispositivo lógico programable (PLD) es cualquier dispositivo cuya función está especificada por el usuario, después de fabricado el dispositivo. Se usan para reemplazar lógica SSI (*Integración a pequeña escala)* y MSI (*Integración a mediana escala*), ahorrando así en costo y tiempo.

Hay muchos tipos de PLD comerciales. El primero en desarrollarse fue el *arreglo lógico programable* (PLA, *Programmable Logic Array*) [7], es un circuito que puede programarse para ejecutar una función compleja. Normalmente se utiliza para implementar lógica combinacional, pero algunos PLA pueden utilizarse para implementar diseño lógicos secuenciales. El PLA es una solución con un solo circutio integrado a muchos problemas lógicos, que puede tener muchas entradas y muchas salidas. Con base en la idea de que las funciones lógicas se pueden realizar en forma de suma de productos, un PLA comprende un juego de compuertas AND que eliminan un conjunto de compuertas OR.

Un dispositivo de lógica de arreglo programable (PAL, *Programmable Array Logic*), son más simples de fabricar y por tanto menos costosos que los PLA, aparte de ofrecer mejor rendimiento [7]. Fueron desarrollados para superar ciertas desventajas que tiene el PLA, consiste en una matriz AND programable y una OR fija.

3.1.2 Historia de los FPGA

Ross Freman, co-fundador de Xilinx, inventó el arreglo matricial de compuertas. La raíz histórica de las FPGA son los dispositivos de lógica programable compleja (CPLD, *Complex Programmable Logic Device*) a mediados de 1980. CPLD y FPGA incluyen un relativo número de elementos lógicos programables. El rango de densidad de los CPLD va desde miles a decenas de miles de compuertas lógicas, mientras que el de las FPGA va desde decenas de miles hasta muchos millones. La diferencia primaria entre CPLDs y FPGAs son sus arquitecturas. Un CPLD tiene una estructura un poco restringida, consistiendo la unión de uno o más arreglos lógicos que alimentan a un número pequeño de registros con entrada de reloj (clock). El resultado de esto es menos flexibilidad, con la ventaja de una mejor predicción de los tiempos de retraso.

La arquitectura de los FPGAs, por otro lado, son dominadas por las interconexiones. Esto las hace más flexibles, en términos de rango de diseños prácticos para los cuales pueden ser usadas. Tienen un conjunto muy grande de componentes digitales elementales, combinacionales y secuenciales, compuertas AND, OR, NOT, FLIP-FLOPs entre otros, lo interesante está en que se pueden programar las conexiones entre las compuertas y FLIP-FLOPs de modo que se puede crear cualquier diapositivo digital, los únicos limitantes son la frecuencia y la cantidad de compuertas, y estos son los factores que determinan el precio del FPGA.

Los CPLD y FPGA se usan actualmente en aplicaciones diversas, como productos de consumo, reproductores de DVD y aparatos de televisión de alta

definición, por ejemplo, circuitos controladores para fábricas automotrices y equipos de prueba, enrutadores de internet e interruptores de redes de alta velocidad, así como equipos de cómputo, como grandes sistemas de almacenamiento en disco y cinta [7].

En una situación de diseño puede elegirse un CPLD siempre que el circuito necesario no sea muy grande o cuando el dispositivo deba realizar su función en seguida de la aplicación de potencia al circuito. Los FPGA no son buenos para este último caso porque, como ya se señaló, están configurados mediante elementos de almacenamiento volátiles que pierden el contenido que guardan cuando la enegía se desconecta. Esta propiedad resulta en un retraso antes de que el chip FPGA pueda cumplir su función cuando se enciende.

Los FPGA son adecuados para la implementación de circuitos de una gran variedad de tamaños, desde cerca de 1000 hasta más de un millón de compuertas lógicas equivalentes. Además del tamaño, el diseñador ha de considerar otros criterios, como la velocidad de operación necesaria del circuito, la restricciones de disipación de potencia y el costo de los chips [7].

3.1.3 Arreglos de compuertas de campos programables

Un arreglo de compuertas de campos programables (FPGA) es un dispositivo *lógico* programable que soporta la implementación de circuitos lógicos hasta cierto punto grandes [7]. Los FPGA ofrecen bloques lógicos para la implementación de las funciones requeridas. Como se observa en la Figura 3. 1 la estructura general de un FPGA contiene tres tipos principales de recursos: bloques lógicos, bloques I/O (input/output) para conectar a los pines del paquete, y cables de interconexión e interruptor.

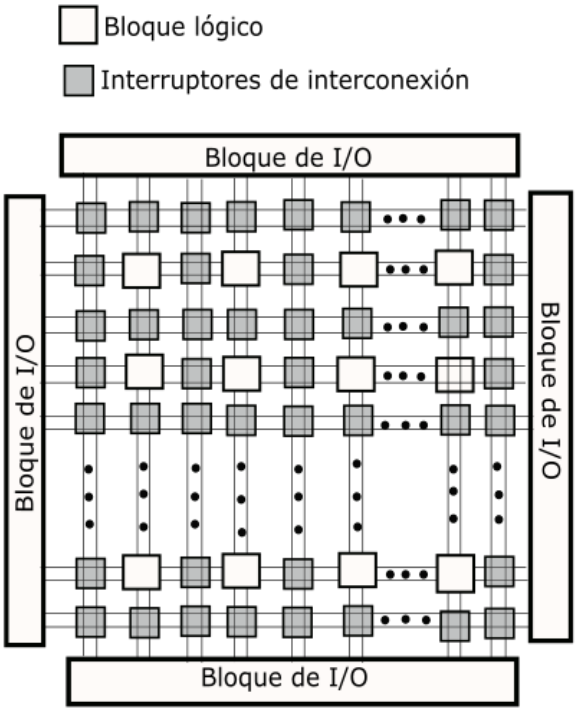

Figura 3. 1- Estructura general de un FPGA

Los bloques lógicos estan dispuestos en un arreglo bidimensional, en tanto que los cables de interconexión estan organizados como *canales de enrutamiento* horizontales y verticales entre filas y columnas de bloques lógicos. Los *canales de enrutamiento* contienen cables e interruptores programables que permiten que los bloques lógicos se conecten de muchas formas.

Los FPGA sirven para implementar circuitos lógicos con un tamaño de más de un millón de compuertas equivalentes [7].

Cada bloque lógico en un FPGA tiene un pequeño número de entradas y salidas. En el mercado se pueden encontrar varios productos FPGA, que presentan diferentes tipos de bloques lógicos. El más usado de éstos es una tabla de consulta (LUT, *Lookup Table*), que contiene celdas de almacenamiento que sirven para implementar una pequeña función lógica. Cada celda puede contener un solo valor lógico, 0 ó 1. El valor almacenado se produce como la salida de la celda de almacenamiento. Pueden crearse LUT de varios tamaños, en los que el tamaño se define mediante el número de entradas. Una celda corresponde al valor de salida de cada fila de la tabla de verdad [7].

Para realizar un circuito lógico en un FPGA, cada función lógica del circuito ha de ser lo suficientemente pequeña para encajar un solo bloque lógico. Cuando se implementa un circuito en un FPGA, los bloques lógicos se programan para cumplir las funciones necesarias y los canales de enrutamiento para realizar las interconexiones requeridas entre los bloques lógicos. Los FPGA se configuran con el método ISP (*In-System Programing*). Las celdas de almacenamiento de las LUT en un FPGA son volátiles, lo que significa que pierden el contenido que almacenan siempre que la fuente de poder para el chip se apague.

3.2 Verilog

Otra de las herramientas utilizadas para el desarrollo de esta tesis, es el lenguaje de descripción de hardware Verilog HDL (*Hardware Description Language*) utilizado para modelar sistemas electrónicos. Éste leguanje soporta el diseño, prueba e implementación de circuitos analógicos y digitales como procesadores, memorias o un simple FLIP-FLOP. Ésto significa que realmente un lenguaje de descripción puede utilizarse para describir cualquier hardware digital a cualquier nivel.

A finales de 1980, los diseñadores comenzaron a dejar de utilizar lenguajes de propietarios, Hilo y Verilog avanzaron hacia el Standard HDL [8].

Verilog fue inventado por Phil Moorby en 1985, mientras trabajaba en Automated Integrated Design Systems, más tarde renombrada Gateway Design Automation. El objetivo de Verilog era ser un lenguaje de modelado de hardware. En el año de 1994 Verilog se convirtió en uno de los lenguajes de descripción de mayor relevancia. Verilog es uno de los estándares de HDL disponibles hoy en día en la industria para el diseño de hardware. Este lenguaje permite la descripción del diseño a diferentes niveles. A continuación se mencionan los niveles de abstracción en Verilog.

- Nivel de puerta: Corresponde a una descripción a bajo nivel de diseño, también denominada modelo estructural. Se describe el diseño mediante

el uso de compuertas lógicas (AND, OR, NOT, etc.), conexiones lógicas y añadiendo las propiedades de tiempo de las diferentes compuertas. Todas las señales son discretas, pudiendo tomar únicamente los valores 0, 1, X o Z (siendo X estado indefinido y Z estado de alta impedancia). Este nivel no resulta muy adecuado de utilizar, ya que al tener *n entradas,* será necesario utilizar un gran número de compuertas lo que se vuelve complicado de representar [8].

- Nivel de transferencia de registro o nivel RTL *(Register Transfer Level)*: Los diseños descritos a nivel RTL especifican las características de un circuito mediante operaciones y la transferencia de datos entre registros. Mediante el uso de las operaciones de tiempo, las operaciones se realizan en instantes determinados. La especificación de un diseño a nivel RTL le confiere la propiedad de diseño sintetizable, por lo que hoy en día el diseño a nivel RTL se define cómo código sintetizable o Código RTL [8].
- Nivel de comportamiento: La principal característica de este nivel, es su total independencia de diseño. El diseñador, más que definir la estructura, define el comportamiento del diseño. En este nivel, el diseño se define mediante algoritmos en paralelo. Cada uno de estos algoritmos consiste en un conjunto de instrucciones que se ejecutan de forma secuencial. La descripción a este nivel puede hacer uso de sentencias no sintetizables, y su uso se justifica en la realización de los denominados testbenches [8].

3.2.1 Diseño al nivel de transferencia de registro

Un sistema digital es un sistema lógico secuencial construido con FLIP-FLOP y compuertas. Los módulos de un sistema digital se definen mejor con un conjunto de registros y las operaciones que realizan con la información binaria guardada en ellos. Algunos ejemplos de operaciones de registros son *shift*

(desplazamiento), *count* (contar), *clear* (borrar) y *load* (cargar) [8]. Los registros son los componentes básicos del sistema digital. Un sistema digital se representa al nivel de transferencia de registro (RTL) cuando se especifica mediante los siguientes tres componentes:

1- El conjunto de registros en el sistema.

2- Las operaciones que se efectúan con los datos guardados en los registros.

3- El control que supervisa la secuencia de operaciones en el sistema.

El tipo de operaciones más frecuente en sistemas digitales se clasifica en cuatro categorías:

1. Operaciones de transferencia, transfieren (es decir, copian) datos de un registro a otro.
2. Operaciones aritméticas, realizan aritmética (p.ej., multiplicación) con los datos contenidos en los registros.
3. Operaciones lógicas, manipulan los bits (p.ej., OR lógica) de datos no numéricos guardados en los registros.
4. Operaciones de desplazamiento, desplazan datos entre registros.

La operación de transferencia no cambia el contenido de información de los datos que se van a transferir del registro origen al destino a menos que el origen y el destino sean los mismos.

Las otras tres operaciones cambian el contenido de la información durante la transferencia. La notación de transferencia de registro y los símbolos utilizados para representar las varias operaciones de transferencia no se estandarizan.

3.2.2 Operadores HDL

En la Tabla 3. 1 se muestran los tipos de operadores HDL.

Tabla 3. 1- Tipos de operadores HDL

Tipo de operador	**Símbolo**	**Operación realizada**
Aritmético	+ - * / % **	Suma Resta Multiplicación División Módulo Exponenciación
Bit por bit o Reducción	~ & | ^	Negación (complemento) AND OR OR exclusiva (XOR)
Lógico	! && ||	Negación AND OR
Desplazamiento	>> << >>> << [,]	Desplazamiento lógico a la derecha Desplazamiento lógico a la izquierda Desplazamiento aritmético derecha Desplazamiento aritmético izquierda Concatenación
Relacional	> < == != === !== >= <=	Mayor que Menor que Igualdad Desigualdad Igualdad case Desigualdad case Mayor que o igual Menor que o igual

La Tabla 3. 2 se observa la precedencia de operadores HDL.

Tabla 3. 2- Tabla de precedencia de operadores

+ - ! ~ & ~& | ~| ^ ~^ ^~ (unario)	**Precedencia máxima**
**	
* / %	
+ - (binario)	
<< >> <<< >>>	
< <= > >=	
== != === !==	
& (binario)	
^ ^~ ~^ (binario)	
| (binario)	
&&	
||	
?: (operador condicional)	
{ } { { } }	Precedencia mínima

Los operadores lógicos y relacionales especifican condiciones de control y tienen expresiones booleanas como sus complementos.

Los operandos de los operadores aritméticos son números. Los operadores +,- , * y / forman la suma, la diferencia y el cociente, respectivamente de un par de operandos. El operador de exponenciación (**) forma un valor de punto flotante de doble precisión con una base exponente que tiene un valor entero, real o con

signo. Los números negativos se representan en forma de complemento de 2. El operador módulo produce el residuo de la división de dos números.

3.2.3 Descripción del diseño

Un diseño en Verilog comienza con la sentencia:

module <nombre_módulo> <(definición de las señales de interfaz)>

En segundo lugar se declaran las entradas/salidas.

input/output <ancho > señal;

Después se describe el módulo/diseño, y se termina la descripción con:

endmodule (notar que a diferencia de las demás sentencias, no se introduce ";")

Antes de proseguir con la definición de diseño, es importante hacer notar las siguientes consideraciones:

- Comentarios: De una sola línea, pueden introducirse precedidos de //. De varias líneas, pueden introducirse con el formato /* comentario */.
- Uso de mayúsculas: Verilog es sensible al uso de mayúsculas, se recomienda el uso únicamente de minúsculas.
- Identificadores: Los identificadores deben comenzar con un carácter, pueden contener cualquier letra de la **a** a la **z** caracteres numéricos, además de los símbolos "_" y "$". El tamaño máximo es de 1024 caracteres.

3.2.4 Instrucciones cíclicas

El Verilog HDL tiene cuatro tipos de ciclos que ejecutan instrucciones de procedimiento repetidamente *repeat*, *forever*, *while* y *for*. Todas las instrucciones cíclicas deben aparecer adentro de un bloque **initial** u **always**.

- Initial: Este tipo de proceso se ejecuta una sola vez comenzando su ejecución en tiempo cero. Este proceso no es sintetizable, es decir, no se puede utilizar en una descripcion RTL. Su uso esta íntimamente ligado a la relación del testbench.
- Always: Este tipo de proceso se ejecuta continuamente a modo de bucle. Tal y como su nombre lo indica, se ejecuta siempre. La ejecución de este proceso está controlada por una temporización, es decir, se ejecuta cada determinado tiempo o por eventos. La sintaxis de este proceso es:

always <temporización> o <@(lista sensible)>

El ciclo *repeat* ejecuta las instrucciones asociadas un número específico de veces. En el siguiente código se conmuta el reloj 16 veces y produce ocho ciclos de reloj en un tiempo de ciclo de 10 unidades.

```
initial
  begin
    clock = 1'b0;
  repeat (16)
     #5 clock =~ clock;
 end
```

El ciclo *forever* produce una ejecución repetitiva incondicional de una instrucción de procedimiento o un bloque de instrucciones de procedimiento. Por ejemplo, el siguiente ciclo produce un reloj continuo con un tiempo de ciclo de 20 unidades.

```
initial
  begin
    clock = 1'b0;
  forever
     #10 clock =~ clock;
 end
```

El ciclo *while* ejecuta una instrucción o un bloque de instrucciones repetidamente mientras una expresión es verdadera. Si para empezar la expresión

es falsa, la instrucción nunca se ejecuta. El siguiente ejemplo ilustra el uso de este ciclo.

```
integer count;
 initial
   begin
    count = 0;
    while (count < 64)
    #5 count =~ count + 1;
  end
```

El ciclo *for* es una forma compacta de expresar las operaciones implicadas por una lista de instrucciones cuyas variables se indexan. Este ciclo contiene tres partes, separadas por dos puntos y coma:

- Una condición inicial.
- Una expresion para verificar la condición de terminación.
- Una asignación para cambiar la variable de control.

En el siguiente ejemplo el ciclo for repite las instrucciones de procediemiento ocho veces. La variable de control es j, la condición inicial es j=0, y el ciclo se repite en tanto j es menor que 8. Depues de cada ejecución de la instrucción cíclica, el valor de j se incrementa en 1.

```
for (j=0; j<8; j=j+1)
 begin
   // Las instrucciones de procedimiento van aquí
 end
```

3.2.5 Números en Verilog

Las constantes numéricas en Verilog pueden especificarse en decimal, hexadecimal, octal o binario. Los números negativos se representan en complemento a 2 y carácter "_" puede utilizarse para una representación más clara del número, si bien no se interpreta. La sintaxis para representar una constante numérica es:

- <Tamaño> <base> <valor>

Los números negativos se especifican poniendo el signo “-” delante del tamaño de la constante. La representación interna de los números negativos en Verilog se realiza en complemento a 2. Tal como se especifica en el siguiente ejemplo:

$$-8\ d2 \rightarrow 11111110$$

3.2.6 Tipo de datos

En Verilog, existen principalmente dos tipos de datos.

- **Nets:** Representan conexiones estructurales entre componentes. No tienen capacidad de almacenamiento de información. La más destacada es wire.
- **Registers:** Representan variables con capacidad. De los diferentes tipos de registers, los más usados son el tipo reg y el tipo integer.

3.3 QUARTUS

En esta sección se darán a conocer las herramientas que se utilizaron para la realización de este proyecto de tesis.

La plataforma QUARTUS II es un software producido por ALTERA ® (fabricante líder de dispositivos lógicos programables) para el análisis y síntesis de diseños realizados en HDL, integra herramientas de desarrollo necesarias para procesar diseños en forma amigable e incluso manejar proyectos jerárquicos. También cuenta con métodos poderosos de síntesis lógica, compilación, partición, simulación funcional, simulación en tiempo y simulación enlazada con varios dispositivos.

Para el manejo de esta plataforma se requiere introducir un diseño, sintetizarlo y finalmente configurarlo y grabarlo en el dispositivo seleccionado, sin embargo, es recomendable simularlo y analizarlo en el tiempo.

La Figura 3. 2 muestra la ventana de ejecución de Quartus II 13.1.

Figura 3. 2- Ventana de ejecución de Quartus II 13.1

3.3.1 Proyecto

Un proyecto contiene todos los archivos de la jerarquía de un diseño. Los módulos que contenga un proyecto son llamados fuentes. La plataforma QUARTUS realiza la compilación, análisis en el tiempo y programación de un dispositivo en un solo proyecto. Para compilar un proyecto con archivos independientes se debe especificar, primero, qué archivo se desea compilar.

3.3.2 Introducción del diseño

Introducir un diseño significa el proceso de describir la arquitectura del diseño, utilizando algún método que sea soportado por QUARTUS. La selección del dispositivo a utilizar se hace en el momento de introducir el diseño a esta plataforma.

3.3.3 Síntesis del diseño

Sintetizar un diseño significa traducirlo a código creador de hardware. Para sintetizar un proyecto se requiere compilarlo. Puede darse el caso de que un diseño sea compilado exitosamente y sin embargo, no sea sintetizable debido a que la plataforma QUARTUS no tenga los constructores necesarios para alguna instrucción de dicho diseño.

Después de sintetizar cualquier diseño, éste queda listo para programarse en un CPLD o para configurarse en una FPGA. Una correcta y detallada definición del proyecto es imprescindible para una correcta planificación.

3.3.4 Creación de un proyecto

Lo primero que se requiere, es crear una carpeta en donde se tendrán todos los archivos generados durante el desarrollo del proyecto. Posteriormente, se requiere ejecutar el programa. Esta acción inicia la ejecución del entorno de desarrollo de la plataforma Quartus II como se muestra en la Figura 3. 3, que permite acceder a los recursos y procesos.

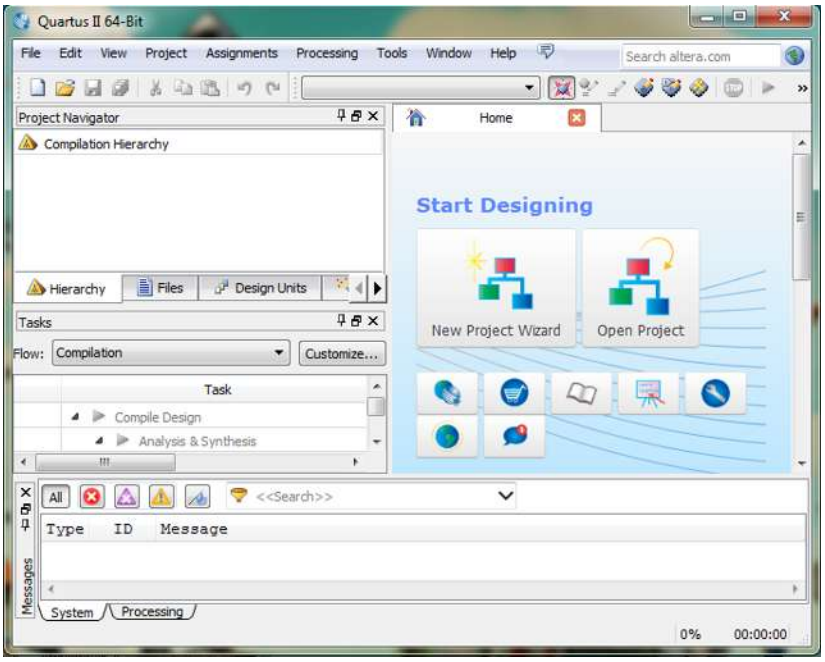

Figura 3. 3- Ventana principal del entorno de desarrollo Quartus II 13.1

A continuación se enlistan los pasos a seguir para crear un nuevo proyecto.

- En la ventata principal se selecciona la opción “New Project Wizard”, como se ilustra en la Figura 3. 4.

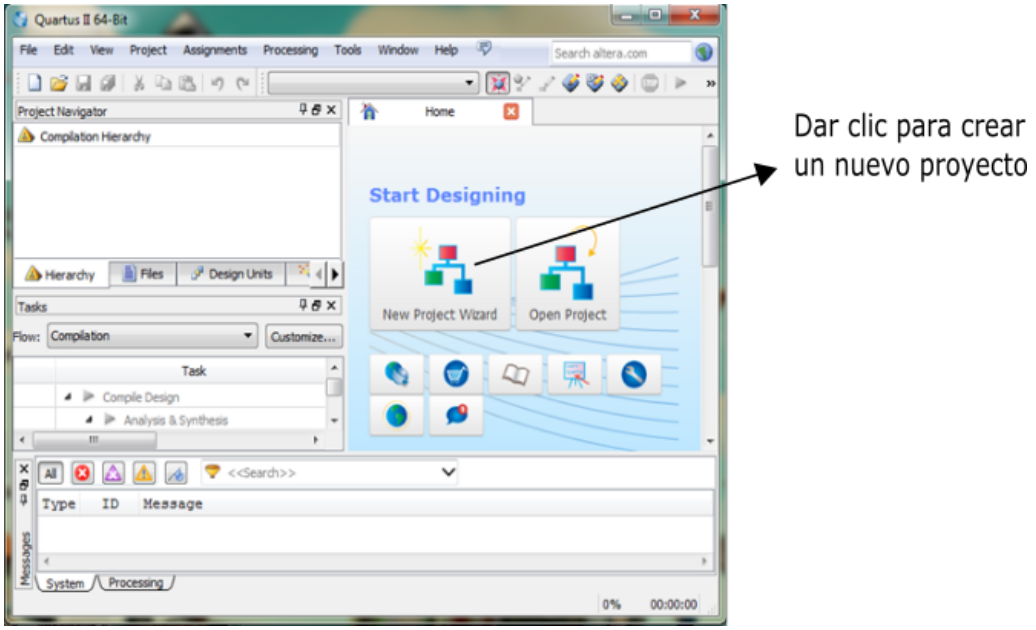

Figura 3. 4- Ventana de inicio en Quartus II

- En la Figura 3. 5 se da clic en Next. Al hacer esto se muestra otra ventana, Figura 3. 6, en la que se introduce el nombre y la localización del proyecto, después se selecciona Next.

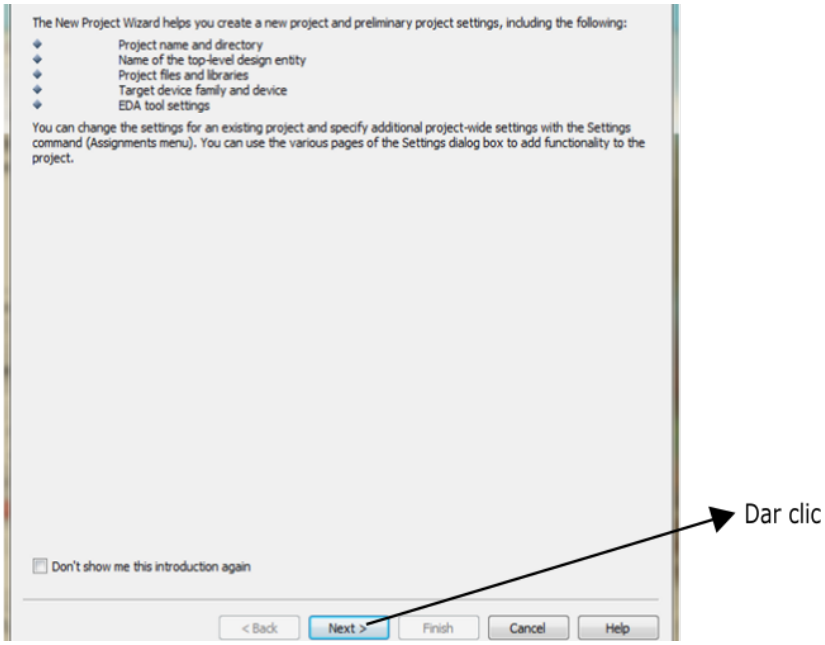

Figura 3. 5- Creación de un nuevo proyecto

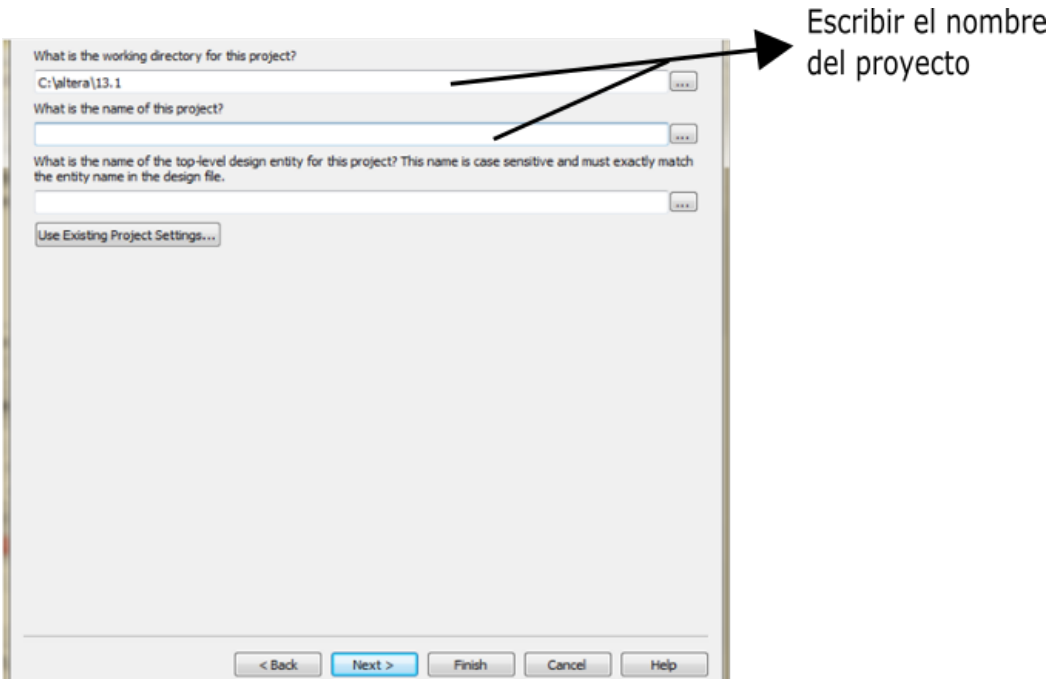

Figura 3. 6 - Carpeta y nombre con que se va a guardar el proyecto

- Después, en la Figura 3. 7, se pide seleccionar archivos de diseños que se quieran agregar al proyecto, en este caso, no se tiene ningún archivo, por lo que sólo se da clic en Next.

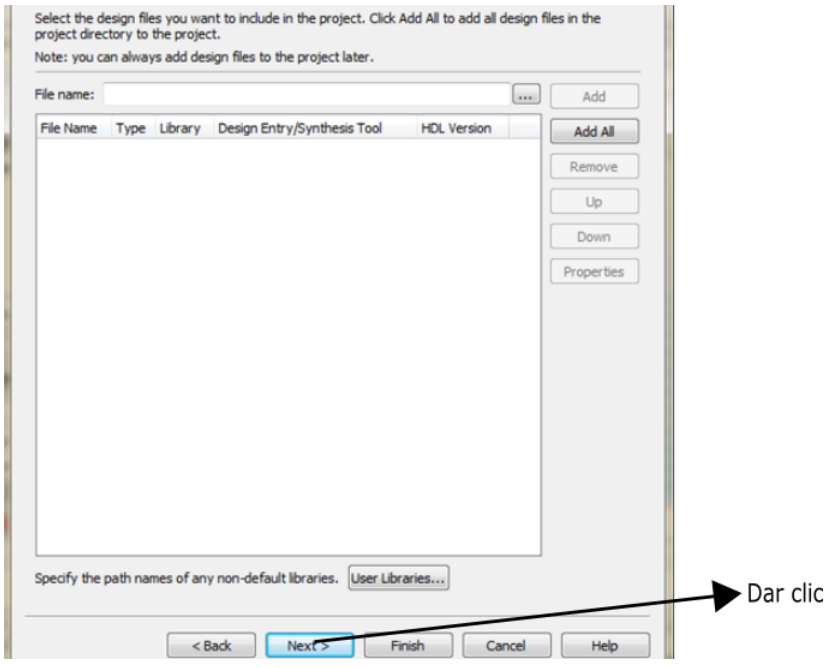

Figura 3. 7- Seleccionar archivo de diseño

- En la Figura 3. 8,se selecciona la matrícula de la FPGA en la que se trabajó, para este caso es perteneciente a la familia de Cyclone III - EP3C16F484C6N.

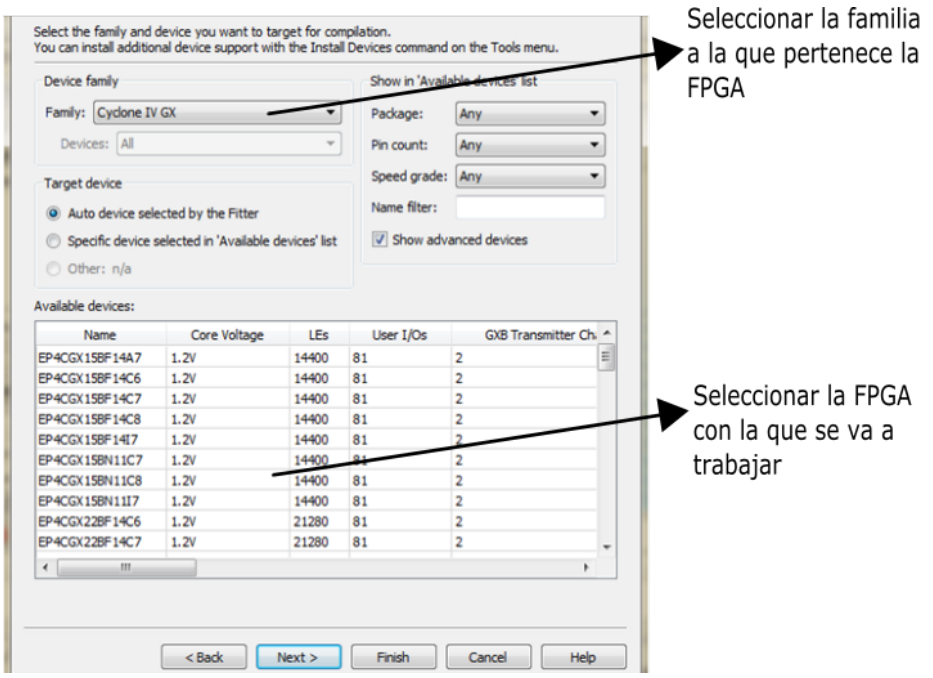

Figura 3. 8- Seleccionar el dispositivo con el que se va a trabajar

- En la Figura 3. 9, se elige el lenguaje para la herramienta de simulación que se desee, ya sea VHDL o Verilog. Para este caso Verilog.

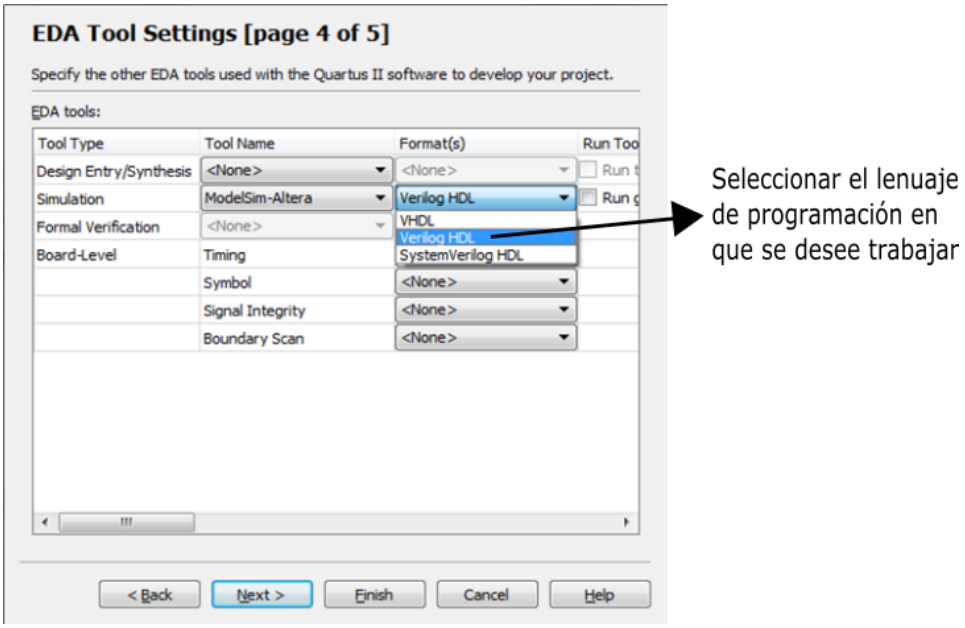

Figura 3. 9- Seleccionar el formato de simulación

- La tabla de propiedades del proyecto aparece como se muestra en la Figura 3. 10.

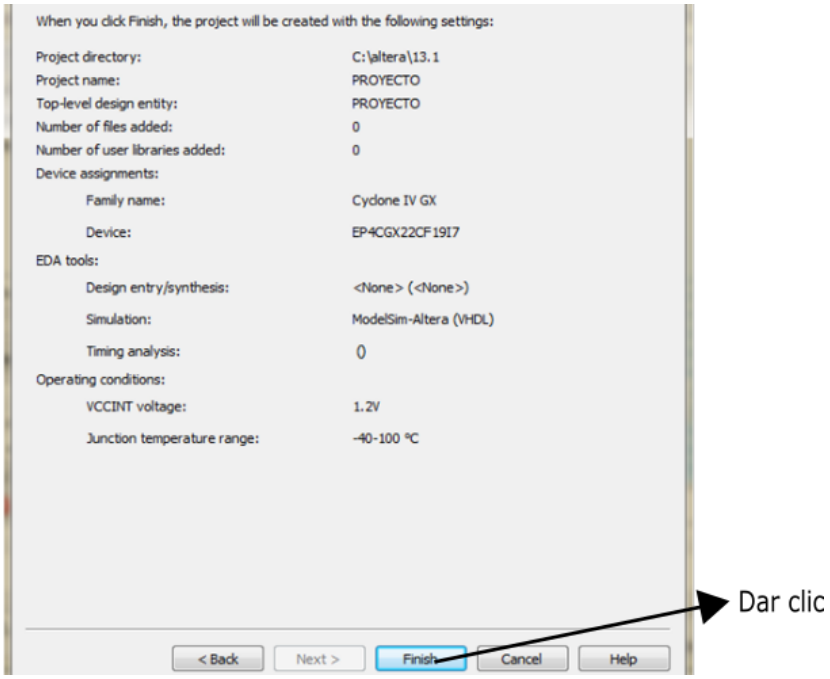

Figura 3. 10- Propiedades del proyecto

- Al dar clic en finalizar se abre otra ventana, Figura 3. 11, en ésta se selecciona la opción “New” y, en la ventana que se presenta, se selecciona “Verilog VHDL file”.

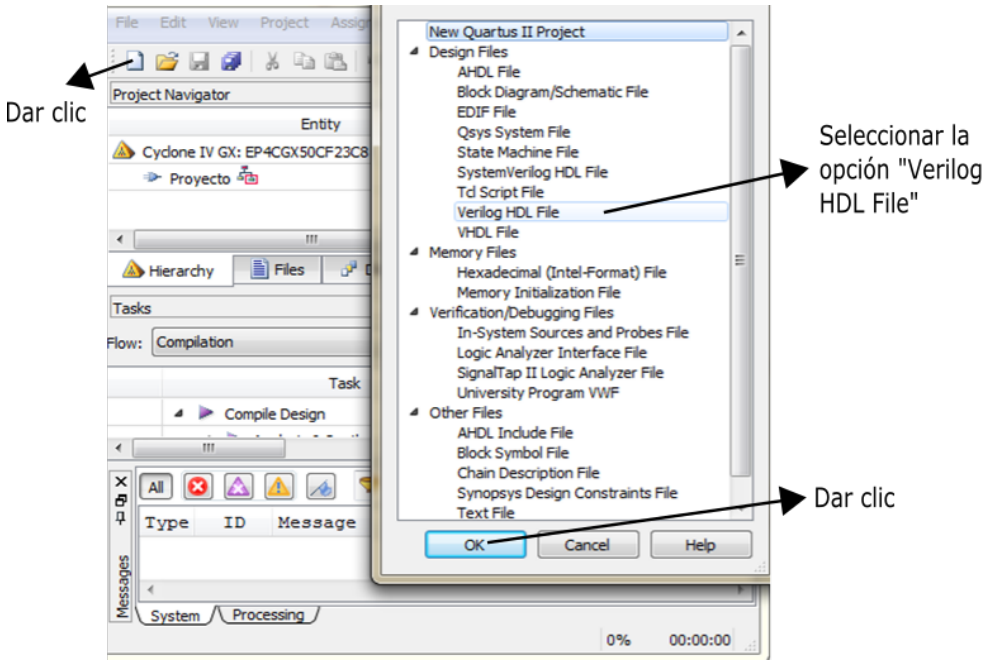

Figura 3. 11- Elegir el lenguaje de diseño Verilog HDL

- De lo anterior, ahora ya es posible escribir el código correspondiente al proyecto, como se ilustra en la Figura 3. 12.

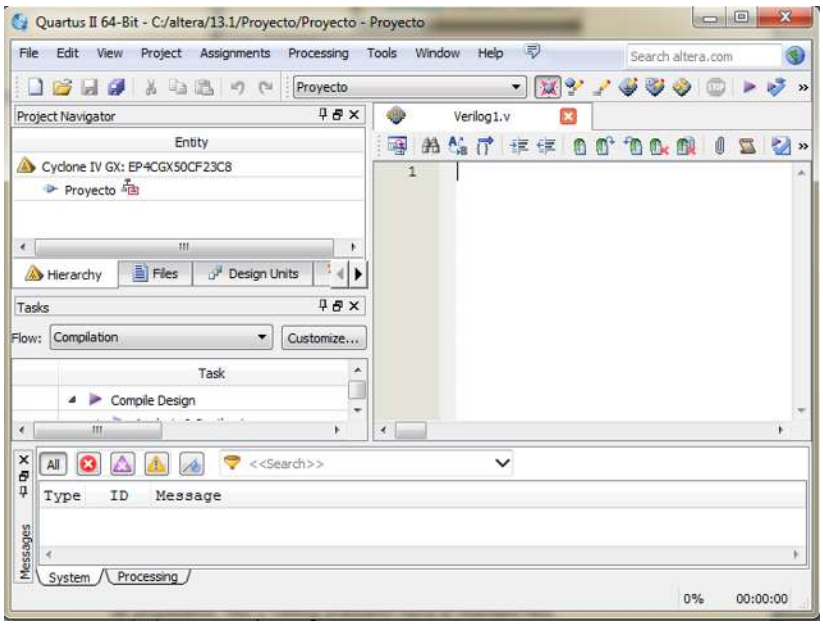

Figura 3. 12- Ventana para escribir el código

- Al escribir el código, se compila y se corrigen errores, si es que existen. Para compilar se da clic en la opción “Start Analysis & Synthesis”, como se muestra en la Figura 3. 13.

Figura 3. 13- Comenzar análisis y síntesis del código

- Una vez compilado correctamente el código, se realiza la simulación del mismo. Para esto, se da clic en “New”, se selecciona la opción “University Program VWF”, finalmente se da clic en “OK” como se muestra en la Figura 3. 14.

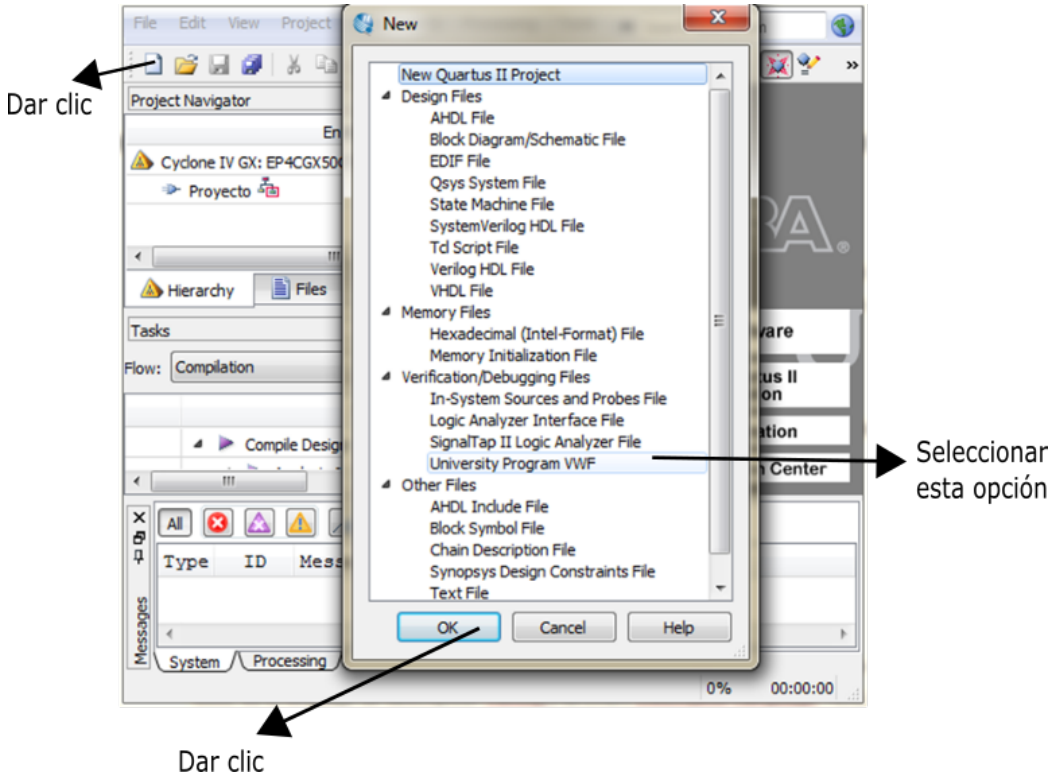

Figura 3. 14- Iniciar simulación

- Para realizar la simulación, es necesario agregar las variables de entrada y salida del código (in, out). Se da clic derecho sobe la parte izquierda de la ventana, seguido de esto, se selecciona la opción “Insert Node or Bus”, como se muestra en la Figura 3. 15, después en la Figura 3. 16 se da clic en “Node Finder”.

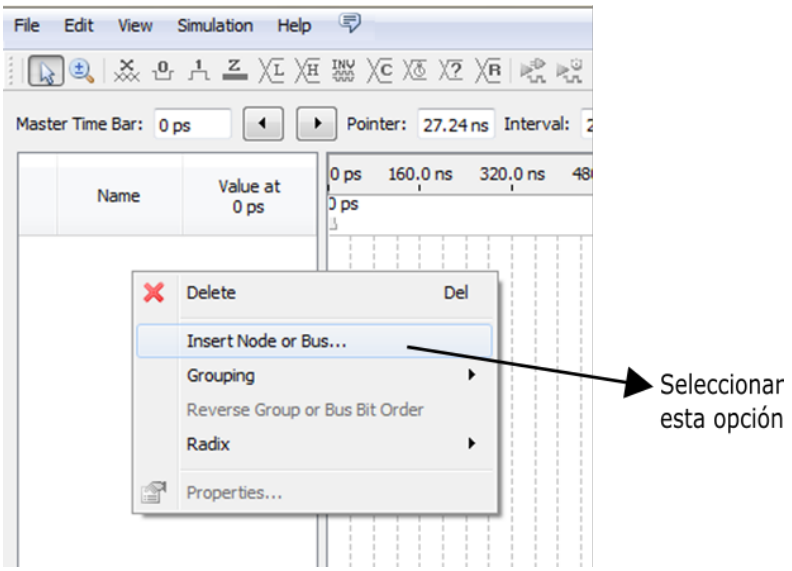

Figura 3. 15- Agregar entradas y salidas

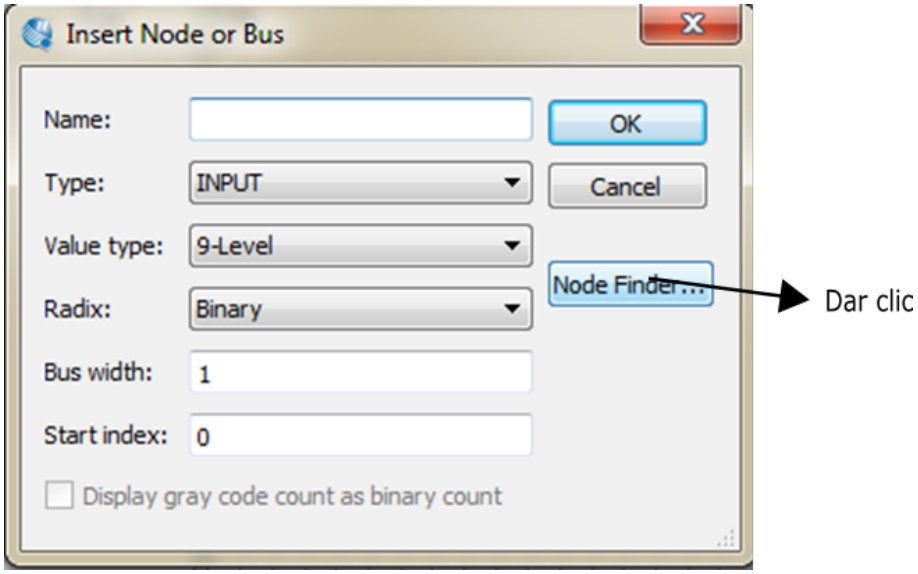

Figura 3. 16- Insertar nodo o bus

- Seguido de esto, se muestra una nueva ventana en la cual se seleccionan las variables que se desean observar en la simulación. En dicha ventana, se da clic en “List”, se muestran las variables existentes en el código, y se eligen las que sean necesarias en la simulación. Como se muestra en la Figura 3. 17 y Figura 3. 18.

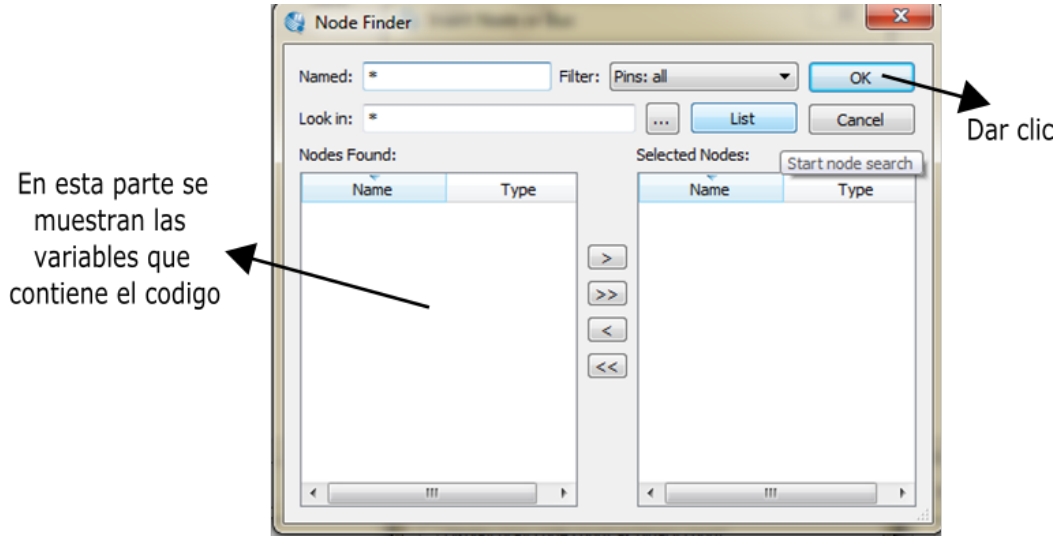

Figura 3. 17- Selección de las variables

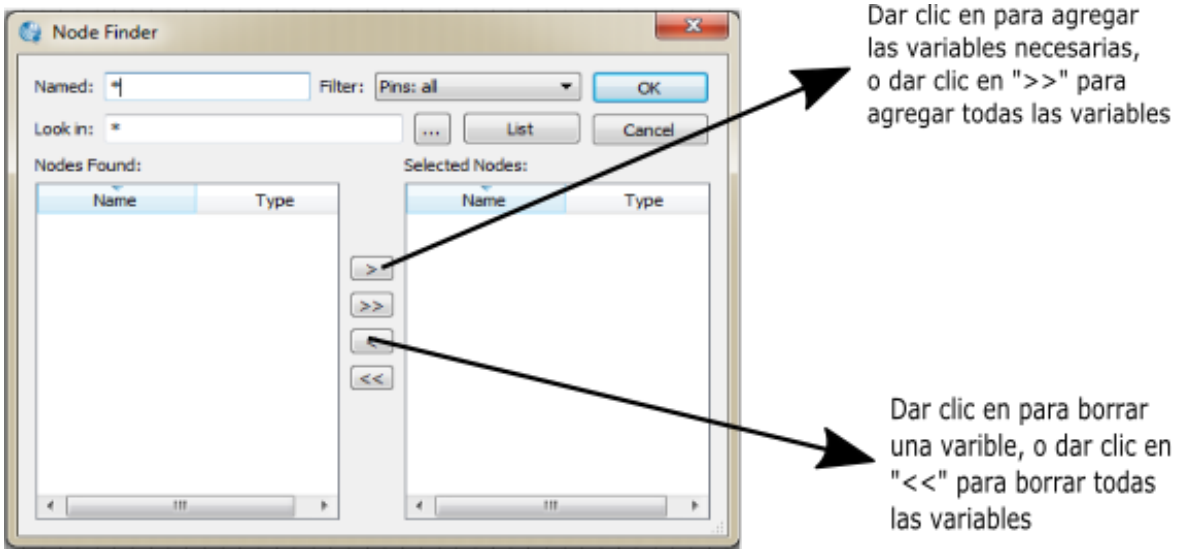

Figura 3. 18- Agregar o quitar variables

- Para determinar las propiedades de las variables, es decir, si es base binario, hexadecimal, etc.. Se realiza como se muestra en la Figura 3. 19 y la Figura 3. 20.

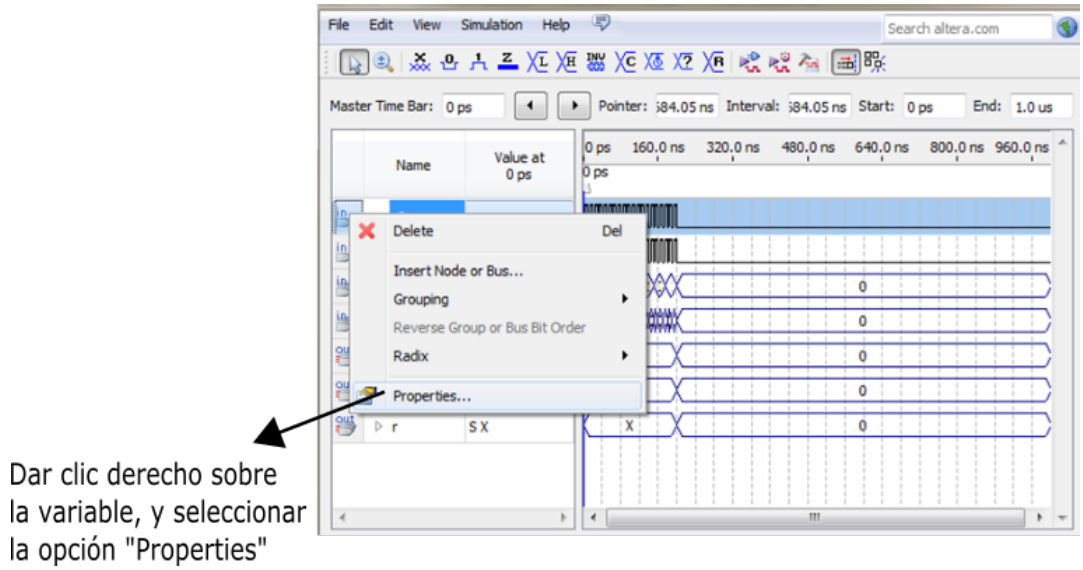

Figura 3. 19- Propiedades de las variables

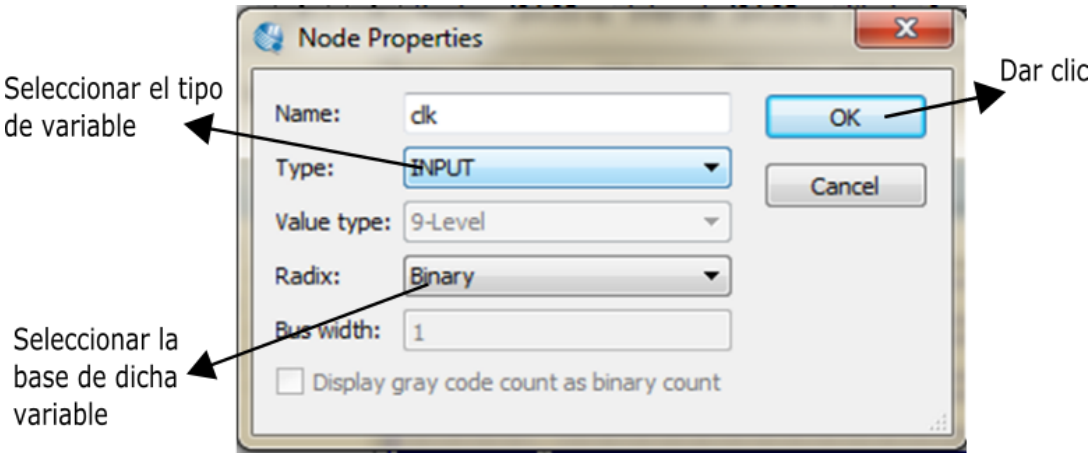

Figura 3. 20- Seleccionar el tipo de variable

- Para correr la simulación se da clic “Run Functional Simulation”. Como se muestra en la Figura 3. 21.

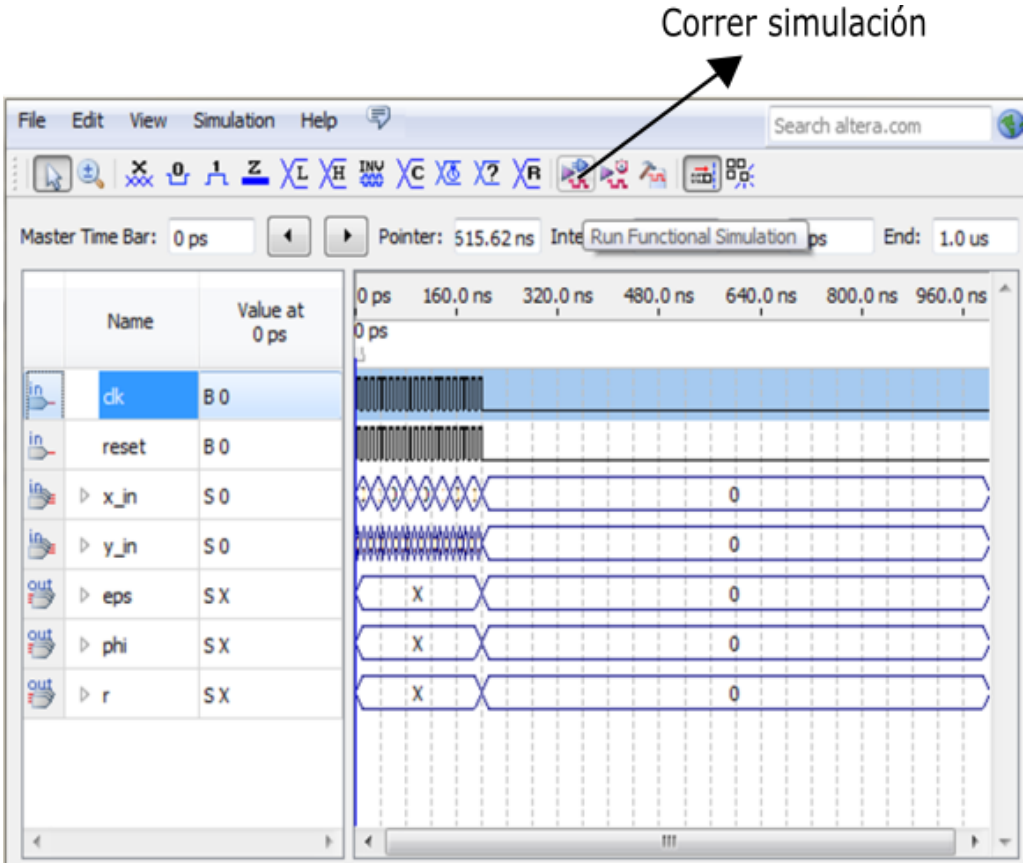

Figura 3. 21- Correr simulación

- Se configuran los pines, de acuerdo con el manual Altera DE0 Board, para el Cyclone III - EP3C16F484C6. Se selecciona la opción “Pin planner”, como se ilustra en la figura Figura 3. 22.

Figura 3. 22- Seleccionar la opción “Pin Planner”

- En la Figura 3. 23 se muestran las entradas y salidas del código, y se configuran a consideración del usuario.

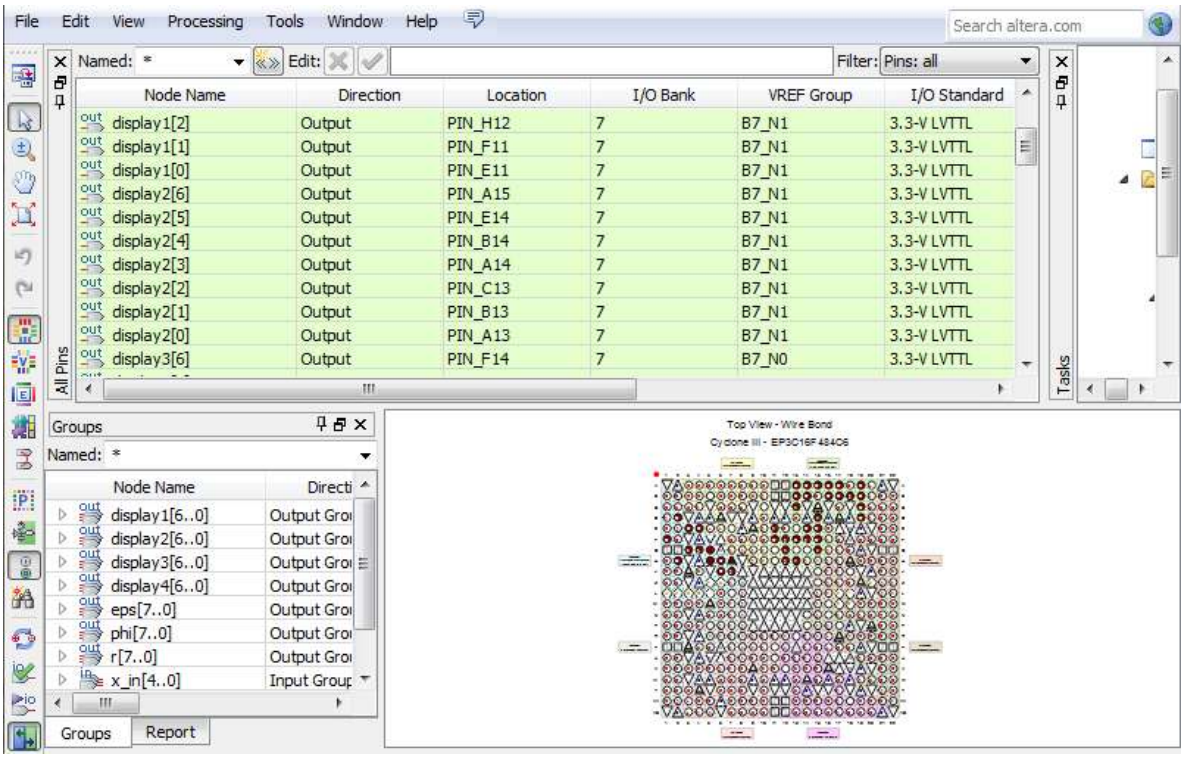

Figura 3. 23- Configuración de pines

- Al terminar la configuración de pines, se cierra la ventana y se selecciona la opción “Programmer” como se ilustra en la Figura 3. 24, para así se observar su funcionamiento en el FPGA.

Figura 3. 24- Seleccionar la opción “Programmer”

CAPÍTULO IV. IMPLEMENTACIÓN DEL ALGORITMO CORDIC DESCRITO EN VERILOG HDL.

4.1 Algoritmo CORDIC

CORDIC, es una técnica iterativa que se basa en la rotación de vectores para evaluar funciones trigonométricas, hiperbólicas, multiplicaciones y divisiones. Dicha técnica involucra solo operaciones como suma, resta y corrimiento de bits [9]. El algoritmo es muy atractivo para la implementación de hardware porque utiliza un cambio único elemental y agrega pasos para realizar la rotación vectorial en el plano 2D.

Para la estructura iterativa CORDIC, el rendimiento de velocidad de operación esta limitada por un número de iteraciones, N. A nivel algorítmico, una solución trivial para superar tal problema es reducir directamente el número de iteraciones, sin embargo la señal se verá seriamente distorsionada por el ruido de aproximación y cuantificación en implementaciones prácticas.

A través de este algoritmo se puede obtener el valor aproximado del seno o coseno de un ángulo dado usando sólo las operaciones ya mencionadas. Algunos ejemplos de uso para este algoritmo se encuentra en las calculadoras para calcular la función trigonométrica de algún ángulo, en los microcontroladores, procesos evaluativos en FPGA, etcétera. Para el caso de las calculadoras, se puede pensar que usan la serie de Taylor, pero cuando el número de iteraciones es muy grande se vuelve lento, por esa razón es viable usar el algoritmo CORDIC. En los microcontroladores, por ejemplo, en un panel solar, se quiere saber el ángulo en el que se encuentra inclinado el panel, es ahí donde el algoritmo CORDIC se emplea.

4.1.1 Modo rotacional para el CORDIC circular

El algoritmo se implementa en el sistema coordenado circular, modo rotacional, en este modo se desea que el vector original (x_0, y_0) rote un águlo determinado, este valor de angulo (z_0) también es un parámetro de entrada; y para completar ese giro se hacen pequeñas rotaciones angulares hasta que se complete el ángulo de entrada (z_0).

Si el valor del águlo al que se pretende aproximar es menor a 90°, la primera rotación será en sentido anti-horario y comienza en 0°, por otro lado si el ángulo a aproximarse es mayor a 90°, la primera rotación será en sentido horario y comienza en 180°, ya que si no se respeta esta condición y el ángulo es mayor que 90°, por más iteraciones que se hagan no será posible la aproximación a dicho valor, ya que éste algoritmo solo converge a 99.88°. En la Figura 4. 1 se ilustra el funcionamiento del algoritmo CORDIC, con el vector K en el origen este comienza a rotar, hasta aproximarse al vector R. El vector (x', y') es el vector resultante después de N rotaciones.

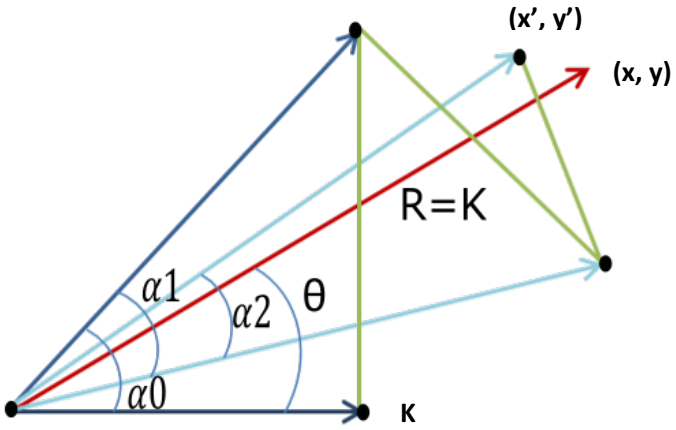

Figura 4. 1- Representación gráfica del Algoritmo CORDIC

El algoritmo se define de la siguiente manera:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \frac{1}{cos\theta_k} \begin{bmatrix} 1 & -d_k\, tan\theta_k \\ d_k\, tan\theta_k & 1 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \end{bmatrix} \tag{2.9}$$

$$= \sqrt{1 + 2^{-2k}} \begin{bmatrix} 1 & -d_k\, 2^{-k} \\ d_k\, 2^{-k} & 1 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \end{bmatrix} \tag{2.10}$$

Donde, *dk* es la dirección de giro, es decir *dk = 1* para sentido anti horario y es *-1* para sentido horario. Se sustituye el valor de $\tan\theta_k = 2^{-k}$, mientras que *1/cos* θ_k se sustituye por su identidad trigonométrica $\sqrt{1 - tan^2\,\theta_k}$.

Se desarrollan las ecuaciones, y las iteraciones quedan de la siguiente forma:

$$x_{k+1} = x_k - \, d_k 2^{-k} y_k \tag{2.11}$$

$$y_{k+1} = y_k + \, d_k 2^{-k} x_k \tag{2.12}$$

$$z_{k+1} = z_k - \, d_k tan^{-1}\,(2^{-k}) \tag{2.13}$$

el resultado después de *k* iteraciones es:

$$x_k = G\,(x_0 cos\, z_0 \, - \, y_0 sin\, z_0) \tag{2.14}$$

$$x_k = G\,(x_0 sin\, z_0 + \, y_0 cos\, z_0) \tag{2.15}$$

$$z_k = 0 \tag{2.16}$$

donde el factor de escala, *G*, es la ganacia de rotación y es constante para un número de iteraciones dado.

$$G \; = \prod_{K=0}^{K-1} \sqrt{1 + 2^{-2K}} \; \approx 1.46676 \tag{2.17}$$

Se debe multiplicar x_k y y_k por un factor A_n el cual es la inversa de de la constante *G*:

$$A_n = \frac{1}{G} \tag{2.18}$$

4.1.2 Modo vectorial

La aplicación más usual del algoritmo de Volder es la conversión de coordenadas cartesianas a coordenadas polares; esto se consigue modificando ligeramente el procediemiento explicado anteriormente.

En este caso, dado un vector *(x, y)* se trata de hacer cero su componente mediante una serie de rotaciones igual que se hacía en el modo de rotación, almacenando el ángulo resultante en la variable *z.* El sentido de giro para éste modo cumple con la condición *y>0* dk = -1, y para *y<0* dk = 1.

En el ejemplo de la Figura 4. 2 se aproxima al valor del ángulo, utilizando el Algoritmo CORDIC, con un vector R, $x_0 = 6,\ y_0 = 10$ y un ángulo.

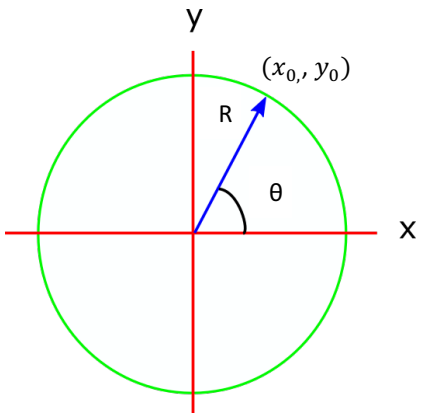

Figura 4. 2- Ejemplo en el que se utiliza el Algoritmo CORDIC

Primera iteración:

k = 0; $d_0 = -1$

$x_{k+1} = x_0 - dk2^{-0}\, y_0$
$x_1 = 6 - (-1)(1)(10)$
$x_1 = 16$

$y_1 = y_0 + dk2^{-0}\, x_0$
$y_1 = 10 + (-1)(1)(6)$
$y_1 = 4$

$z_1 = z_0 - dk\,(tan^{-1}(2^{-0}))$

Segunda iteración:

k =1; $d_1 = -1$

$x_2 = x_1 - dk2^{-1}\, y_1$
$x_2 = 16 - (-1)(0.5)(4)$
$x_2 = 16 + 2 = 18$

$y_2 = y_1 + dk2^{-1}\, x_1$
$y_2 = 4 + (-1)(0.5)(16)$
$y_2 = 4 - 8 = -4$

$z_2 = z_1 - dk\,(tan^{-1}(2^{-1}))$

$z_1 = 0 - tan^{-1}(-1)(1)$
$z_1 = 0.78$ rad

$z_2 = 0.78 - tan^{-1}(-1)(0.5)$
$z_2 = 1.25$ rad

Tercera iteración:

k = 2; $d_1 = 1$

$x_3 = x_2 - d_k 2^{-2}\, y_2$
$x_3 = 18 - (+1)(0.25)(-4)$
$x_3 = 19$

$y_3 = y_2 + dk2^{-2}\, x_2$
$y_3 = -4 + (+1)(0.25)(18)$
$y_3 = 0.5$

$z_3 = z_2 - dk\ (tan^{-1}\ (2^{-2})\ )$
$z_3 = 1.25 - tan^{-1}(+1)(0.25)$
$z_3 = 1.005 = 57.58°$

Para obtener los valores de x_k y y_k , se realiza los siguiente:

$$x_k = G\ (x_0 cos\ z_0\ -\ y_0 sin\ z_0)$$

$$x_k = G\ (x_0 sin\ z_0 + y_0 cos\ z_0)$$

$$z_k = 0$$

$x_3 = G\ (6\ (\cos 0) - 10(sin\ 0))$

$y_3 = G(6\ (sin\ 0) + 10\ (cos\ 0))$

$z_3 = 57.52$

$x_3 = 1.64676\ (6)$ =9.88

$y_3 = 1.64676(10\ )$ = 16.46

$z_3 = 57.52$

Se multiplican x_3 y y_3 por An.

$x_3 = (9.88)(0.60725303) = 5.99$

$y_3 = (16.46)(0.60725303) = 9.99$

En el diagrama que se muestra en la Figura 4. 3, se define el proceso de iteraciones, vistas anteriormente en forma de ecuaciones. Como se puede observar en el esquema, se inicializan los registros de las variables $x_{0,}\ y_0$ y z_0, el incremeto de cada una se guarda en x, y y z. Para cada ciclo de reloj se realiza una iteración, de acuerdo con el valor de dk (es decir el sentido de giro), se toma el signo para cada variable. También se tiene una memoria ROM en donde se almacena el valor de cada ángulo evaluado en $tan^{-1}\,2^{-k}$. >> K representa el incremento con cada ciclo de reloj.

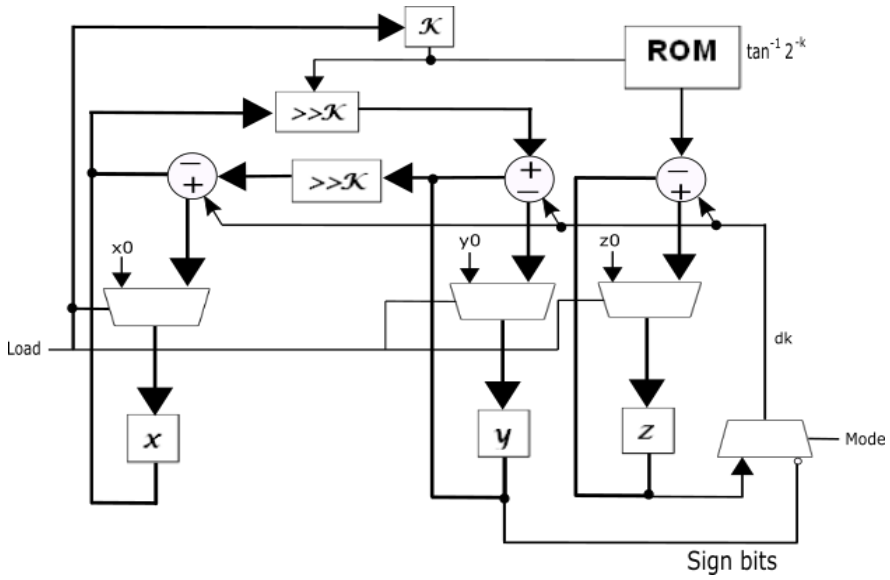

Figura 4. 3- Diagrama de funcionamiento del algoritmo CORDIC

4.2 Implementación en QUARTUS II.

En la herramienta QUARTUS II se implementa el siguiente código en Verilog.

```
//Algoritmo CORDIC

//Se declaran las variables

module Cordic #(parameter W=7)          //Bit de ancho -1
(input clk,                             //Sistema de reloj
 input reset,                           //Reset asíncrono
 input boton,                           //Botón muestra el valor de
                                        //phi
```

```
 input signed [4:0] x_in,              //Sistema real o entrada x
 input signed [4:0] y_in,              //Sistema imaginario o
                                       //entrada y
 output reg signed [W:0] r,            //Resultado del radio
 output reg signed [W:0] phi,
 output reg signed [W:0] eps,          //Resultado del error
 output [6:0] display1, display2, display3, display4);


reg [4:0] a1, a2, a3, a4, a5, a6;
reg [4:0] div, mod, div2, mod2, mod3, div3;
wire [6:0] w1, w2;

//Se defienen los tamaños para cada vector x,y,z
reg signed [W:0] x [0:3];
reg signed [W:0] y [0:3];
reg signed [W:0] z [0:3];


//En este módulo always se realizan las iteraciones necesarias
//para la aproximación del  ángulo

always @(negedge reset or posedge clk)
begin :P1
      integer k;
      if (~reset) begin
      //Reset asincrono
            for(k=0; k<=3; k= k+1) begin
                  x[k] <= 0; y[k] <= 0; z[k] <= 0;
            end
            r <= 0; eps <= 0; phi <= 0;
      end else begin
            if (x_in >= 0)
      //Variable rotante
                  begin
                        x[0] <= x_in;
      //Entrada en el registro 0;
                        y[0] <= y_in;
                        z[0] <= 0;
            end
      else  if (y_in >= 0)
            begin
                  x[0] <= y_in;
                  y[0] <= -x_in;
                  z[0] <= 90;
            end

      else
      begin
                  x[0] <= -y_in;
                  y[0] <= x_in;
                  z[0] <= -90;
```

```
end
if (y[0] >= 0)
     begin
          x[1] <= x[0] + y[0];
          y[1] <= y[0] - x[0];
          z[1] <= z[0] +45;
     end
else
     begin
          x[1] <= x[0]- y[0];
          y[1] <= y[0] + x[0];
          z[1] <= z[0] -45;
     end

if (y[1] >= 0)
//Rotacion 26 grados
     begin
          x[2] <= x[1] + (y[1] >>> 1);
          y[2] <= y[1] - (x[1] >>> 1);
          z[2] <= z[1] + 26;
     end
else
     begin
          x[2] <= x[1] - (y[1] >>> 1);
          y[2] <= y[1] + (x[1] >>> 1);
          z[2] <= z[1] - 26;
     end

if (y[2] >= 0)
     begin
          x[3] <= x[2] + (y[2] >>> 2);
          y[3] <= y[2] - (x[2] >>> 2);
          z[3] <= z[2] + 14;
     end
else
     begin
     x[3] <= x[2] - (y[2] >>> 2);
     y[3] <= y[2] + (x[2] >>> 2);
     z[3] <= z[2] - 14;
     end

     r <= x[3];
     eps <= y[3];
     phi <= z[3];

end
end

//En este módulo always se realizan las operaciones
//necesarias para mostrar en cada display los valores de x_in
//y y_in
```

```
      always@(*)
      begin

            if(x_in <= 5'b01001)
                  a1 = x_in;
            if(x_in > 5'b01001)
                  div = (x_in/5'b01010);
                  mod = (x_in % 5'b01010);
                  a1 = mod;
                  a2 = div;
            if(y_in <= 5'b01001)
                  a3 = y_in;
            if(y_in > 5'b01001)
                  div2 = (y_in/5'b01010);
                  mod2 = (y_in % 5'b01010);
                  a3 = mod2;
                a4 = div2;
            if(phi <= 5'b01001)
                  a5 = phi;
            if(phi > 5'b01001)
                  div3 = (phi/5'b01010);
                  mod3 = (phi % 5'b01010);
                  a5 = mod3;
                  a6 = div3;

end

//La instrucción assign hace funcionar al botón, y muestra el
//valor de phi

assign w1 = boton?a3:mod3;
assign w2 = boton?a4:div3;

//Se realiza la instanciación con el convertidor

      SEG7_LUT U0(

             .iDIG(a1),
             .oSEG(display1));

            SEG7_LUT U1(

             .iDIG(a2),
             .oSEG(display2));

            SEG7_LUT U2(

             .iDIG(w1),
             .oSEG(display3));

            SEG7_LUT U3(
```

```
            .iDIG(w2),
            .oSEG(display4));

endmodule



//Código del convertidor

module SEG7_LUT (     oSEG,iDIG  );
input [3:0] iDIG;
output      [6:0] oSEG;
reg         [6:0] oSEG;

always @(iDIG)
begin
           case(iDIG)
           4'h1: oSEG = 7'b1111001;   // ---a----
           4'h2: oSEG = 7'b0100100;   // |    |
           4'h3: oSEG = 7'b0110000;   // f    b
           4'h4: oSEG = 7'b0011001;   // |    |
           4'h5: oSEG = 7'b0010010;   // ---g----
           4'h6: oSEG = 7'b0000010;   // |    |
           4'h7: oSEG = 7'b1111000;   // e    c
           4'h8: oSEG = 7'b0000000;   // |    |
           4'h9: oSEG = 7'b0011000;   // ---d----
           4'ha: oSEG = 7'b0001000;
           4'hb: oSEG = 7'b0000011;
           4'hc: oSEG = 7'b1000110;
           4'hd: oSEG = 7'b0100001;
           4'he: oSEG = 7'b0000110;
           4'hf: oSEG = 7'b0001110;
           4'h0: oSEG = 7'b1000000;
           endcase
end

endmodule
```

4.2.1 Simulación y pruebas en el dispositivo de trabajo

En la Figura 4. 4 se muestran los resultados de simulación. Se tienen como entradas *clk* (reloj) a 10 ns y un ciclo de trabajo del 50%, *reset* que muestra en 1 y resetea en 0, un *botón* que al presionar muestra el valor de *phi* en los display's 3 y 4, *x_in* y *y_in* corresponden a los valores de coordenadas del vector, valores que introduce el usuario.

Para representar algún número en un display de siete segmentos, en este caso, los ceros (0) indican que el segmento está en alto (encendido) y los unos indican que está en bajo (apagado). Los segmentos se representan de izquierda a derecha, es decir, el primer número a la izquierda es el segmento *a*, el último número representa el segmento *g*. Por ejemplo si se tiene un 1111000 *(g f e d c b a)*, el número que se muestra en el segmento es un siete (7).

Las salidas display1 y display2 muestran el valor de x_in, en este caso es 6 (0000010), por lo que solo se muestra en display1. Display3 y display4 muestran el valor de y_in que es 10, 1 (1111001) y 0 (1000000) mostrados en display4 y display3, respectivamente. Se observa que cuando el botón está en cero se muestra el valor de phi en display3 y display4, para este ejemplo phi = 57, 0010010 representa 5 y 1111000 representa 7.

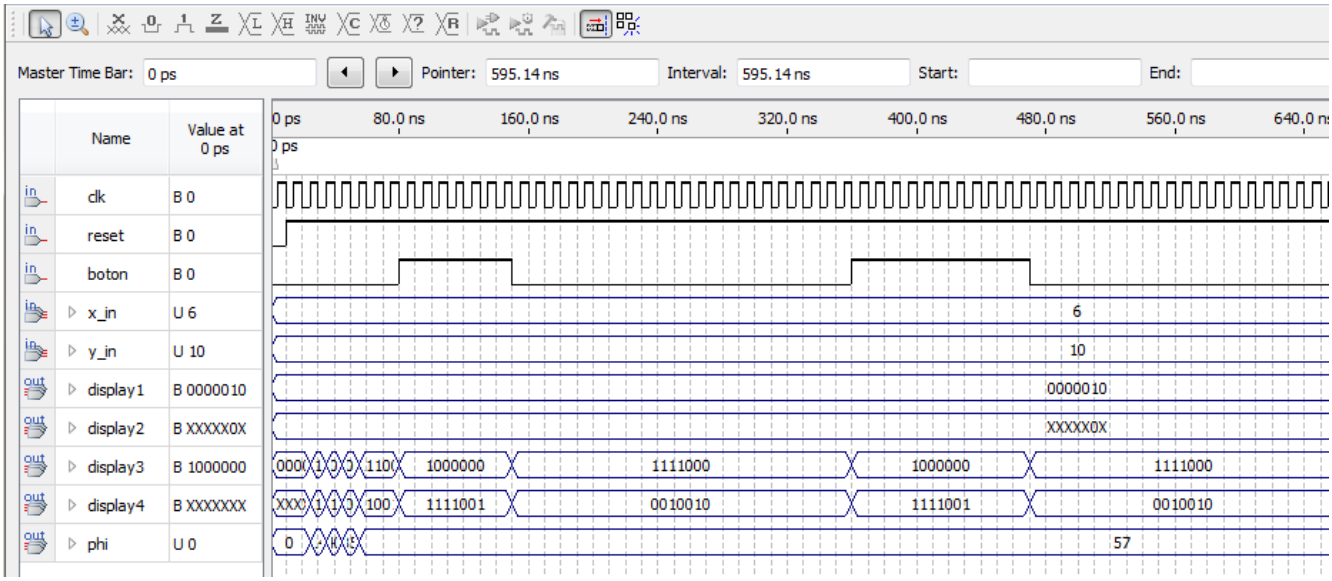

Figura 4. 4- Simulación del algoritmo CORDIC

En la Figura 4. 5 se muestran los valores de *x_in* y *y_in*, introducidos por el usuario. La Figura 4. 6 presenta el valor del ángulo phi, al presionar un botón.

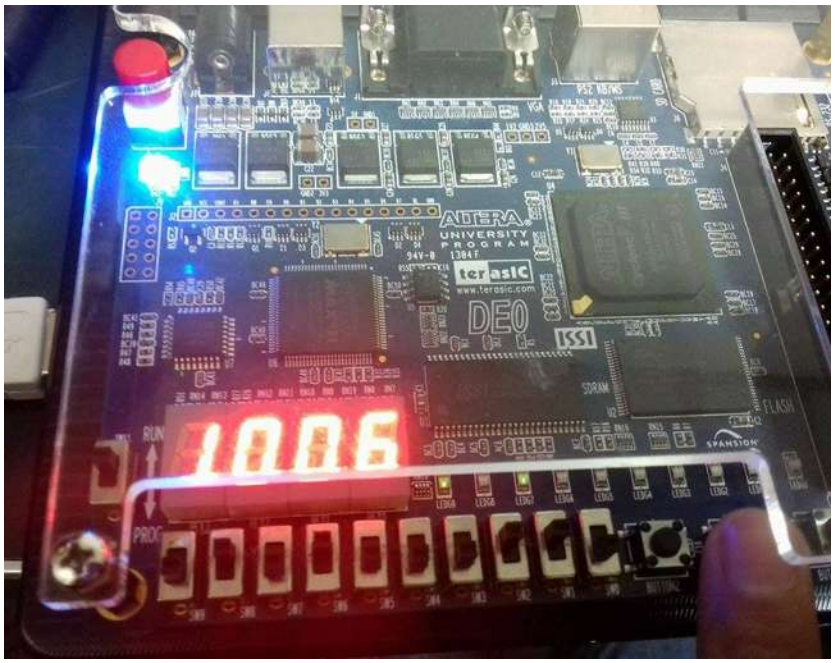

Figura 4. 5- Valores x_in, y_in

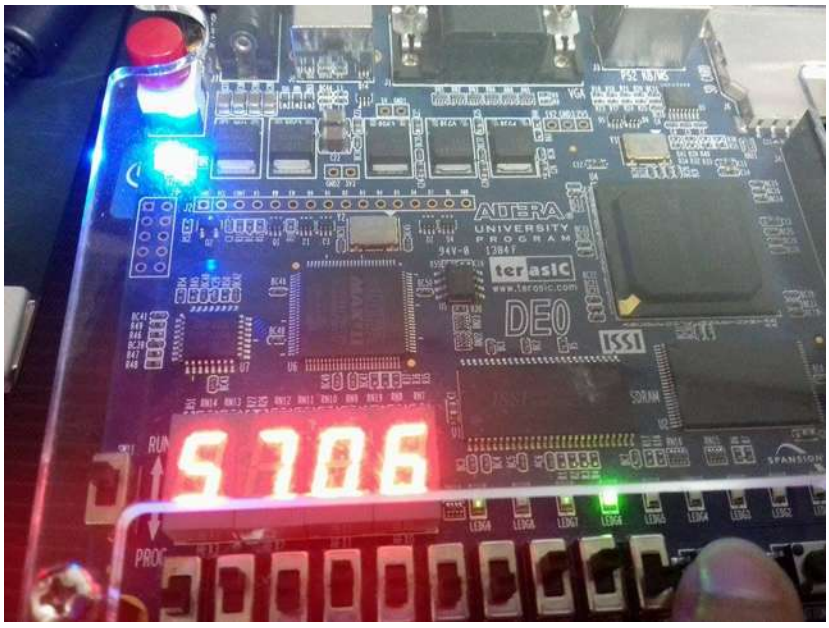

Figura 4. 6- Valor del ángulo (phi)

En la Figura 4. 7 se muestra que el reset del valor de *phi* en los display's 3 y 4.

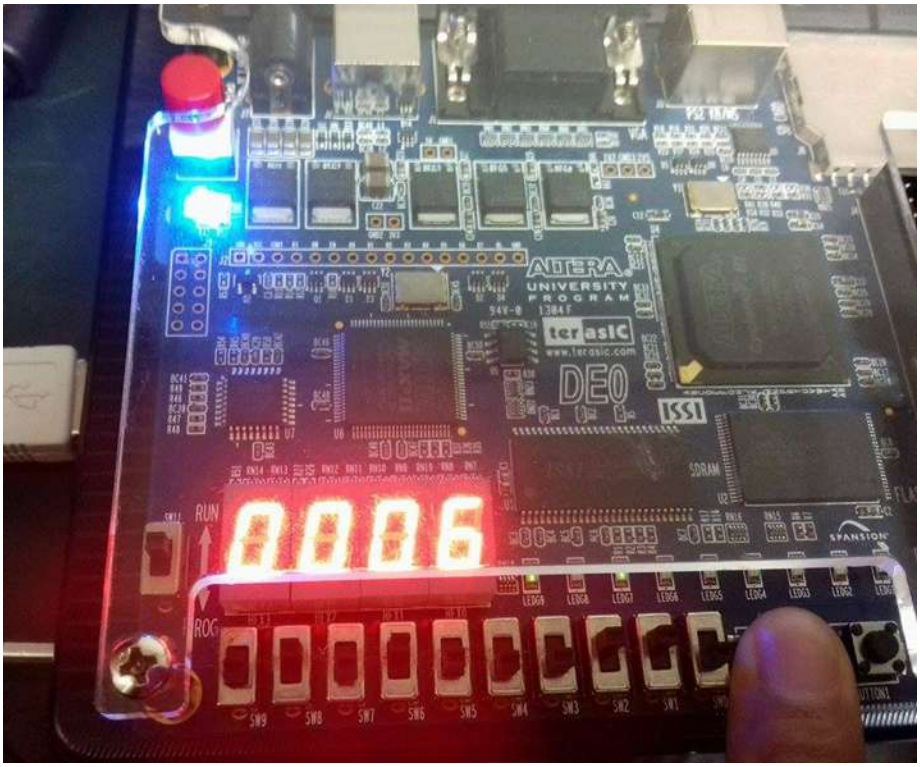

Figura 4. 7- Funcionamiento del reset

CAPÍTULO V. CONCLUSIONES Y TRABAJOS FUTUROS

5.1 Conclusiones

De acuerdo con lo estudiado acerca del algoritmo CORDIC, dispositivos lógicos programables y hardware de diseño se puede concluir:

- Se investigó y estudió un algoritmo CORDIC y se observó que es una técnica iterativa sencilla para la aproximación de ángulos, además de que el número de iteraciones es suficiente para aproximarse al ángulo.

- Se describió el algoritmo CORDIC con la ayuda del lenguaje de descripción de hardware Verilog, se simuló en la herramienta de diseño QUARTUS II y se observó su comportamiento en un diagrama de tiempo.

- De acuerdo a la simulación satisfactoria, se implementó en un FPGA Cyclone III EP3C16F484C6N y se demostró su funcionamiento con sólo tres iteraciones.

5.2 Trabajos futuros

Se desea implementar el algoritmo en la pantalla LT24, para observar en ella los valores de los ángulos. Se pretende acondicionar el algoritmo como un modulador universal, es decir, modulador AM, PM y FM.

Existe una amplia cantidad de aplicaciones en control, por lo que se desea realizar el control de posición de una antena, o un panel solar.

REFERENCIAS

[1] Jack E. Volder, *The CORDIC Trigonometric Computing Technique*, 8th ed.: Electronic Computing, 1959.

[2] JS. Walther, *A unifield algorithm for elementary functions*.: Spring Joint Computer Conf, 1971.

[3] Stanley I. Grossman, *Álgebra Lineal*, Sexta edición ed.: McGraw- Hill, 2008.

[4] M. Kline, *Mathematical Thought From Ancient to Modern Times*.: Oxford University Press, 1972.

[5] Thomas L. Floyd, *Fundamentos de Sistema Digitales*, Novena Edición ed.: PEARSON Prentice Hall, 2006.

[6] Michael D. Ciletti M. Morris Mano, *Diseño Digital*, Quinta edición ed.: Pearson , 2013.

[7] Zvonko Vranesic Stephen Brown, *Fundamentos de Lógica Digital con Diseño VHDL*, Segunda edición ed.: McGraw-Hill, 2006.

[8] José Manuel Marín de la Rosa. Field Programmable Gate Array (FPGA).

[9] Donald G. Bailey, *Design for Embedded Image Processing on FPGAs*, 2011th ed.: Jhon Wile & Sons (Asia) Pte Ltd.

[10] Stephen Brown, *Digital Logic*, Segunda edición ed.: McGraw-Hill, 2008.