Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Electrónica

EVALUACIÓN DE CAPACIDADES TÉCNICAS Y LIMITES DEL CONJUNTO DE COMPUERTAS LÓGICAS DEL DISPOSITIVO XC3S50A UTILIZADO POR LA SPARTAN 3A

Andrés Noel Ortiz Betancourth
Asesorado por Ing. Miguel Ventura Pérez

Guatemala, octubre de 2024

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

EVALUACIÓN DE CAPACIDADES TÉCNICAS Y LIMITES DEL CONJUNTO DE COMPUERTAS LÓGICAS DEL DISPOSITIVO XC3S50A UTILIZADO POR LA SPARTAN 3A

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

ANDRÉS NOEL ORTIZ BETANCOURTH
ASESORADO POR ING. MIGUEL VENTURA PÉREZ

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN ELECTRÓNICA

GUATEMALA, OCTUBRE DE 2024

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. José Francisco Gómez Rivera (a. i.)
VOCAL II	Ing. Mario Renato Escobedo Martinez
VOCAL III	Ing. José Milton De León Bran
VOCAL IV	Ing. Kevin Vladimir Cruz Lorent
VOCAL V	Ing. Fernando José Paz González
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANA	Inga. Aurelia Anabela Cordova Estrada
EXAMINADOR	Ing. Brian Enrique Chicol Morales
EXAMINADORA	Ing. Ana María Navarro Orozco
EXAMINADOR	Ing. José Aníbal Silva de los Angeles
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

EVALUACIÓN DE CAPACIDADES TÉCNICAS Y LIMITES DEL CONJUNTO DE COMPUERTAS LÓGICAS DEL DISPOSITIVO XC3S50A UTILIZADO POR LA SPARTAN 3A

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 27 de octubre de 2021.

Andrés Noel Ortiz Betancourth

Guatemala 30 de octubre 2023

Ingeniero
Julio César Solares Peñate
Coordinador del Área de Electrónica
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC.

Apreciable Ingeniero Solares,

Me permito dar aprobación al trabajo de graduación titulado ¨**Evaluación de capacidades técnicas y límites del conjunto de compuertas lógicas del dispositivo XC3S50A utilizado por la Spartan 3A**¨, del señor **Andrés Noel Ortiz Betancourth**, por considerar que cumple con los requisitos establecidos.

Por tanto, el autor de este trabajo de graduación y, yo, como su asesor, nos hacemos responsables por el contenido y conclusiones de este.

Sin otro particular, me es grato saludarle.

Atentamente,

F:_____

Miguel Ventura Perez
Ingeniero en Electrónica
Colegiado 10,524
Asesor

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

Guatemala, 28 de noviembre de 2023

Señor director
Armando Alonso Rivera Carrillo
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC

Estimado Señor director:

Por este medio me permito dar aprobación al Trabajo de Graduación titulado **EVALUACIÓN DE CAPACIDADES TÉCNICAS Y LIMITES DEL CONJUNTO DE COMPUERTAS LÓGICAS DEL DISPOSITIVO XC3S50A UTILIZADO POR LA SPARTAN 3A**, desarrollado por el estudiante **Andrés Noel Ortiz Betancourth**, ya que considero que cumple con los requisitos establecidos.

Sin otro particular, aprovecho la oportunidad para saludarlo.

Atentamente,

ID Y ENSEÑAD A TODOS

Ing. Julio César Solares Peñate
Coordinador de Electrónica

ESCUELA DE INGENIERÍA MECÁNICA ELÉCTRICA
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

SIST.LNG.DIRECTOR.19.EIME.2024

El Director de la Escuela de Ingeniería Mecánica Eléctrica de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del Asesor, con el Visto Bueno del Coordinador de Área, al trabajo de Graduación del estudiante Andrés Noel Ortiz Betancourth: Evaluación de capacidades técnicas y limites del conjunto de compuertas lógicas del dispositivo XC3S50A utilizado por la Spartan 3A, procede a la autorización del mismo.

Ingeniero Armando Alonso Rivera Carrillo
Director
Escuela de Ingeniería Mecánica Eléctrica

Guatemala, agosto de 2024

Decanato
Facultad e Ingeniería

24189101- 24189102

LNG.DECANATO.OIE.617.2024

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica, al Trabajo de Graduación titulado: **EVALUACIÓN DE CAPACIDADES TÉCNICAS Y LIMITES DEL CONJUNTO DE COMPUERTAS LÓGICAS DEL DISPOSITIVO XC3S50A UTILIZADO POR LA SPARTAN 3A.**, presentado por: **Andrés Noel Ortiz Betancourth** después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Firmado electrónicamente por: José Francisco Gómez Rivera
Motivo: Informe final TESIS
Fecha: 23/10/2024 12:10:21
Lugar: Facultad de Ingeniería, USAC.

Ing. José Francisco Gómez Rivera
Decano a.i.

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO a.i.
Facultad de Ingeniería

Guatemala, octubre de 2024

ACTO QUE DEDICO A:

Dios	Por acompañarme durante todo el camino de mi vida y guiarme a través de todos los procesos.
Mis padres	Jenny Betancourth y Noel Ortiz, por todo su amor, apoyo y motivación incondicional a lo largo de los años.
Mis hermanos	Por estar siempre a mi lado, por ser un apoyo incondicional y unos compañeros de vida excepcionales.
Mi familia en general	Gracias por todo su cariño demostrado y por estar presentes en cada uno de mis avances
Mis amigos y compañeros de carrera	Por acompañarme en cada etapa de la carrera, ser un apoyo en cada circunstancia de la vida y por su sincera amistad, muchas gracias por formar parte de esta etapa.

AGRADECIMIENTOS A:

Mi familia	Por todo el apoyo y cariño mostrado durante el proceso.
Mis amigos	Por estar presentes en cada etapa de la carrera y la vida y ser parte fundamental de ambas.
Universidad de San Carlos de Guatemala	Por darme un espacio para el aprendizaje de mi carrera.
Facultad de Ingeniería	Por enseñarme los conocimientos necesarios para desempeñarme como profesional.
Ing. Miguel Ventura	Por apoyarme y asesorarme durante todo el proceso para culminar con éxito esta etapa.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
A	Amperios
BPS	Baudios por segundo
<=	Carga de archivo (VHDL)
Hz	Hercio
=	Igual que
KB	Kilo Byte
MB/s	Mega Bytes por segundo
mm	Milímetros
Ω	Ohms
V	Voltios

GLOSARIO

AC	Señal de corriente alterna.
ADC	Conversor analógico digital.
ARM	Arquitectura de microchips que poseen una configuración tipo RISC.
DC	Señal de corriente directa.
Encoder	Dispositivo utilizado para convertir una posición angular en un código digital.
FPGA	Matriz de compuertas programables.
GND	Tierra física en un circuito.
GPIO	Puertos de entrada y salida.
HSync	Sincronización horizontal en una señal de video.
IA	Inteligencia artificial.
IoT	Es el acrónimo al que se le conoce como internet de las cosas.

JTAG	Es un conector estándar que posee de dos a cinco pines.
LED	Diodo emisor de luz.
PWM	Modulación por ancho de pulso.
RAM	Memoria que almacena la información de forma temporal.
RGB	Composición de color formada por el rojo, verde y azul.
ROM	Memoria que almacena la información de forma permanente.
Temporizador 555	Es un tipo de circuito que puede generar ondas periódicas.
TTL	Es un tipo de tecnología de construcción que utiliza transistores bipolares para las funciones lógicas.
UART	Protocolo de comunicación para transmisión y recepción de datos.
VCC	Es la alimentación de voltaje de un circuito.

VGA	Es un estándar de conexión y resolución de video, conocido por ser popular a partir del año 1980 hasta la creación del HDMI.
VHDL	Es un lenguaje de descripción de hardware que se utiliza para el diseño digital de circuitos electrónicos.
VSync	Sincronización vertical en una señal de video.

RESUMEN

En el presente trabajo se puede encontrar un análisis de las capacidades de la FPGA Elbert V2. En el primer capítulo se explica a grandes rasgos que es una FPGA. Se muestra a grandes rasgos los módulos de la Elbert V2 y se explora de forma muy general el *software* utilizado para su programación.

En el segundo capítulo se realiza una explicación detallada del lenguaje utilizado por las FPGAs llamado VHDL. Se muestran los conceptos y elementos básicos del lenguaje con ejemplos prácticos para facilitar su comprensión.

El tercer capítulo se exponen las aplicaciones y configuraciones del código VHDL en la tarjeta Elbert V2. Se muestra como debe ser la instalación del *software* que se utiliza y se muestran todos los módulos útiles para la Elbert V2.

En el cuarto capítulo muestra las capacidades reales de la Elbert V2. Se realizan múltiples ejemplos de implementación de externos de comunicación y transmisión a la Elbert V2 a través del código VHDL, explicando paso a paso la creación y configuración de estos.

En el quinto capítulo se expone la posibilidad de migrar a otra tarjeta más moderna, evaluando las fortalezas y debilidades de la Elbert V2 dejando abierta la puerta a una posible renovación del hardware utilizado para la enseñanza del lenguaje VHDL y las FPGAs.

OBJETIVOS

General

Realizar un estudio para conocer las capacidades de *hardwar*e y *software* de la FPGA Elbert V2 para obtener el máximo rendimiento y potencial de la tarjeta de desarrollo.

Específicos

1. Obtener los datos numéricos de la capacidad de memoria y procesamiento utilizada por la FPGA Elbert V2 para ejecutar instrucciones.

2. Obtener los valores máximos y mínimos prácticos de cada uno de los puertos de entrada y salida de la FPGA Elbert V2.

3. Desarrollar una guía básica con programas y algoritmos que obtengan el máximo potencial de la FPGA Elbert V2.

4. Evaluar si la FPGA Elbert V2 sigue siendo la mejor opción de entrada para el aprendizaje de las FPGAs y su lenguaje VHDL, comparándola con la CMOD A7-35T.

.

INTRODUCCIÓN

Las FPGAs dentro del mercado han recibido una enorme inversión en los últimos años. Los datos que reflejan el crecimiento exponencial que está teniendo este tipo de tecnología y crea la necesidad de adentrarse en este mundo lo antes posible, ya que a medida que crece el mercado de las automatizaciones también crecerá la necesidad de profesionales que dominen la programación de las FPGAs al ser la mejor opción en los mercados.

En los laboratorios de la Universidad San Carlos de Guatemala se hace uso de la tarjeta desarrollada por Xilinx, la Elbert V2. Esta FPGA es funcional para tener un acercamiento inicial al mundo de las FPGAs, sin embargo, su poco soporte y poca documentación hace complicado el aprendizaje de la misma. En este documento se busca otorgar al lector las habilidades necesarias para obtener el máximo rendimiento de dicha tarjeta.

Aprender el lenguaje VHDL dedicado a las FPGAs es una tarea compleja si no se tiene la guía correcta, se pretende que, con los análisis realizados, los proyectos presentados y el código mostrado, se pueda otorgar una mejor guía y pueda facilitar el aprendizaje del lenguaje y sus aplicaciones.

Por último, se ha comparado la tarjeta Elbert V2 contra equipos modernos para evaluar si sigue siendo una buena opción para el aprendizaje. Se pudo determinar que a pesar de los años que tiene dicha tarjeta en el mercado, sigue siendo una excelente opción para entusiastas e ingenieros novatos que requieran el aprendizaje del uso de las FPGAs.

1. CONCEPTOS GENERALES DE FPGA Y LA ESTRUCTURA DEL LENGUAJE DE SU PROGRAMACIÓN

1.1. ¿Qué es una FPGA?

Una Matriz De Puertas Programables (FPGA por su acrónimo en inglés) no es más que un conjunto de compuertas lógicas organizadas en arreglos que pueden ejecutar múltiples tareas. Las compuertas lógicas son circuitos combinacionales que se conforman de muchos transistores y darán salidas determinadas cuando se les aplique cambios a sus entradas. Con esto una FPGA puede ser capaz de manejar instrucciones simples como la activación de una única compuerta lógica o acciones más complejas como tener un sistema combinacional completo programado en la misma. Se puede llegar a creer que el uso de una FPGA es similar al uso de un microcontrolador, pero existen algunas diferencias que resultan ventajosas al utilizar FPGA, entre estas están:

- Una FPGA es capaz de llevar a cabo múltiples procesos, lo que permite una programación paralela en una misma tarjeta.
- Una FPGA tiene mayor capacidad de desarrollo de procesos que un microcontrolador, con la pequeña diferencia que es más complicado de aprender y entender.
- Un proyecto en FPGA puede ser fácilmente modificable sin afectar el funcionamiento de todos los procesos existentes, en caso de los microcontroladores se debe tomar en cuenta los procesos existentes para que las actualizaciones no afecten el sistema.

1.1.1. Desarrolladoras de FPGA

La importancia de la competencia en el campo de las FPGA radica en el mismo crecimiento en el mercado de estas, gracias a esto se tiene varias empresas dedicadas al desarrollo y creación de las tarjetas FPGA, entre las empresas más populares se pueden encontrar:

- AMD Xilinx Inc: es una empresa con localidad en Australia, originalmente se dedicó al desarrollo de microchips y es conocida también por ser la empresa creadora de las FPGA. Posee distintos modelos de FPGA entre las cuales se encuentra el modelo Spartan que es el explorado en este trabajo. La empresa AMD anunció en 2020 la compra de Xilinx, *Newsroom* de AMD (AMD, 2020) adquiriendo la compañía por una suma de 30 billones de dólares.

Figura 1.

Logo de la compañía AMD Xilinx

Nota. El gráfico muestra el logo de la compañía AMD. Obtenido de NotebookCheck (2020). *AMD may acquire FPGA maker Xilinx for $30 billion*. (https://www.notebookcheck.net/AMD-may-acquire-FPGA-maker-Xilinx-for-30-billion.497524.0.html), consultado el 25 de octubre de 2022. De dominio público.

- Altera Corporation: es una empresa de origen estadounidense que es pionera y líder en el mercado en cuanto a creación de microchips y arreglos de chips programables, actualmente desarrolla productos dedicados a la simulación que puedan ser programados utilizando lenguaje de descripción de hardware. En el año 2015 fue adquirida por Intel lo que le ha permitido a la compañía ser parte de una red de experimentación más avanzada en los campos de las FPGA y de los productos basados en ARM según indica la propia compañía (Intel, 2015).

Figura 2.

Logo de la compañía Altera Intel

Nota. El gráfico muestra el logo de la compañía Altera Intel. Obtenido de PR Newswire (2015). *Altera Stockholders Approve Merger with Intel*. (https://www.prnewswire.com/news-releases/altera-stockholders-approve-merger-with-intel-300155330.html), consultado el 25 de octubre de 2022. De dominio público.

- Achronix Semiconductor Corporation: es una empresa relativamente más joven que las otras dos ya mencionadas, está dedicada directamente con la creación e innovación de las FPGA aplicando tecnologías de alta velocidad y habilitando la posibilidad de poder

trabajar con redes y tecnologías de comunicación. Es una empresa con localidad en Estados Unidos y pese a ser fundada en 2004 tiene gran presencia en el mercado gracias a sus FPGA de avanzada tecnología que las distingue de su competencia, entre las tecnologías más destacables se puede mencionar la capacidad de aprendizaje por IA, la implementación de respuestas de alta velocidad y la implementación de IoT en su desarrollo.

Figura 3.

Logo de la compañía Achronix Semiconductor Corporation

Nota. El gráfico muestra el logo de la compañía Achronix Semiconductor Corporation. Obtenido de Crunchbase (2013). *Achronix Semiconductor*. (https://www.crunchbase.com/organization/achronix-semiconductor), consultado el 26 de octubre de 2022. De dominio público.

1.1.2. Aplicaciones y usos de FPGA

El uso de las FPGA en la industria se ha incrementado conforme pasan los años por sus características únicas de potencia de procesamiento, reducido tamaño y bajo consumo de voltaje. En la industria actual las

aplicaciones de una FPGA abarcan distintos campos, entre ellos se pueden mencionar:

- Inteligencia artificial
- Internet de las cosas (IoT)
- Telecomunicaciones
- Industria militar
- Domótica

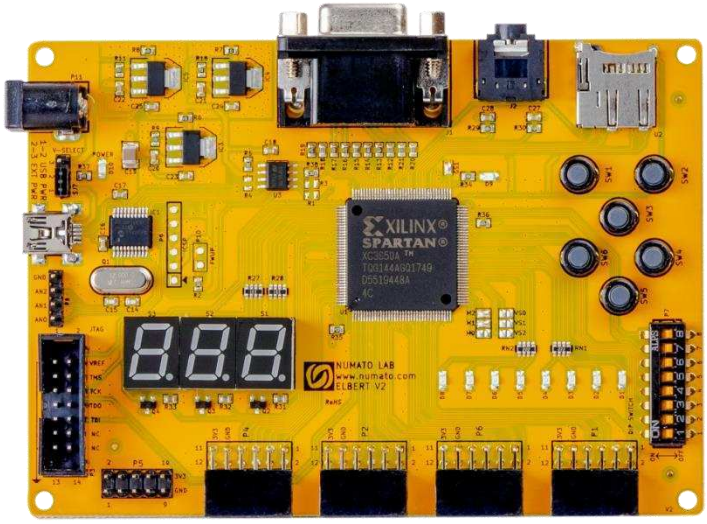
Al existir tantos campos aplicables para una FPGA es lógico que existirá un incremento en la demanda de estas. Según un reporte de Global Market Insights el crecimiento del mercado de las FPGA ha ido en aumento en los últimos años en una tasa del 12 % anualmente, esto se debe según dicho estudio al incremente de las investigaciones de inteligencias artificiales. Se indica que la pandemia de COVID-19 ayudo al crecimiento de las FPGA en la industria de la salud. Todos estos eventos han ayudado a crear una predicción que indica que entre el 2022 al 2028 el mercado de las FPGA puede tener un crecimiento económico del 15 % anual, Global Market Insights (2022). Con el mercado en potencia de las FPGA se hace evidente la necesidad de entender cómo funcionan para poder explotar esas zonas del mercado.

1.2. Tarjeta de desarrollo Elbert V2 Spartan 3A

La FPGA Elbert V2 Spartan 3A es una tarjeta ensamblada y patentada por Numanto Lab que posee múltiples características útiles que ayudan al desarrollo de proyectos. Entre los usos principales y recomendados se encuentra domótica, procesamiento de señales o desarrollo de prototipos de varios productos, no es recomendable su uso industrial por la capacidad del microchip que posee.

Figura 4.

FPGA Elbert V2 Spartan 3 A con Microchip XC3S50A



Nota. El gráfico muestra la tarjeta Elbert V2 con el microchip XC3S50A. Obtenido de Numanto. *Elbert V2 – Spartan 3A FPGA Development Board*. (https://numato.com/product/elbert-v2-spartan-3a-fpga-development-board/), consultado el 1 de noviembre de 2022. De dominio público.

El microchip utilizado es fabricado por Xilinx, dicho microchip pertenece a la familia Spartan® y es el XC3S50A. Según indica la hoja técnica del chip XC3S50A, otorgado por Xilinx (2018) se tiene un arreglo de 50,000 compuertas lógicas que permiten unas 1,584 funciones lógicas derivadas de los mismos arreglos, cabe mencionar que este tipo de chip está diseñado únicamente para un uso doméstico o de pequeña escala, a diferencia de sus contrapartes más robustas como lo pueden ser el chip XC3S200A con 200,000 compuertas que además soporta 4,032 funciones lógicas, más del doble que el chip utilizado.

1.2.1. Características del hardware de la tarjeta Elbert V2

La tarjeta Elbert V2 además de tener el chip fabricado por Xilinx posee varios componentes de entrada y salida que sirven para poder realizar prototipos funcionales sin necesitar componentes externos, existen algunas excepciones en el caso de video y audio, pero el resto de los componentes pueden funcionar directamente en la tarjeta. Entre los módulos más utilizados se encuentran los siguientes:

- Interfaz USB: es un módulo con conector mini USB que permite la comunicación entre la computadora donde se programan las instrucciones y la tarjeta. Gracias a este módulo es posible realizar la transferencia del programa a la FPGA para que esta pueda ejecutar las instrucciones. Este módulo puede entregar 5V de forma similar a una fuente de voltaje, es recomendable su uso solamente para comprobar funcionamiento de los programas que se carguen a la FPGA.

- Fuente de alimentación DC: es un conector hembra DC de dimensiones 5.5x2.5mm y permite dar una mayor potencia a la FPGA en caso sea necesaria. La interfaz USB puede funcionar como fuente tal y como lo indico el punto anterior, sin embargo, hay aplicaciones en los proyectos que requieren un mayor consumo de corriente que no puede darlo el módulo USB. El módulo de fuente de DC puede dar una mayor potencia de voltaje a la tarjeta, está diseñada para soportar un voltaje máximo de 9 V, cualquier voltaje que sobrepase dicho valor puede llegar a dañar el chip XC3S50A tal y como lo indica su hoja de datos.

- Pin selector de tensión: es un conjunto de tres pines machos que se encuentran en medio del conector hembra de la alimentación DC y el

módulo de mini USB, este pin habilita la alimentación dependiendo que se necesite en la FPGA, de fábrica viene unido el pin 2 y 3 que son los que habilitan el voltaje por medio del módulo mini USB, si se desea utilizar la alimentación DC simplemente se debe colocar el conector en la posición 1 y 2.

Figura 5.

Pin selector de tensión

Nota. El Pin selector de la tarjeta Elbert V2 configurado para funcionar por medio de alimentación USB. Elaboración propia.

- Módulo VGA**:** es un módulo VGA conectado al chip XC3S50A permite una salida de video de 8 bits que es capaz de mostrar 256 colores utilizando combinaciones del RGB del módulo. Para lograr las combinaciones cada uno de los colores conecta de forma directa con tres salidas del chip, estas salidas le dan las combinaciones necesarias para asignar un valor al color que será traducido junto con los demás colores a una pantalla que pueda mostrar la imagen.

- LED, Botones Push y Dip switches: estos módulos se unen en uno mismo por permitir la interacción entre el usuario y la FPGA directamente en la tarjeta sin utilizar componentes externos. El LED

consiste en un arreglo de 8 leds conectados directamente al chip por medio de resistencias que ayudan a la reducción de potencia. Los botones push y los dip switches también conectan directamente con el chip con la diferencia que estos completan el camino a tierra de dichas entradas y con esto le comunican al chip que botón fue seleccionado o pulsado.

- Reloj Interno: el reloj que utiliza la tarjeta Elbert V2 es un cristal con una frecuencia de 12 MHz, se debe considerar este valor para poder crear divisores de frecuencia y de esta manera, en un futuro, realizar retardos y demás funciones.

- GPIOs: este término representa la Entrada/Salida de Propósito General *GPIO* por sus siglas en inglés y son los pines programables que pueden configurarse como una salida o entrada según las aplicaciones que se requieran. A diferencia del resto de módulos como los Led o los botones, estos necesitan componentes externos para funcionar. La FPGA Elbert V2 posee un total de 39 GPIOs que se distribuyen en tres Headers que están denominados como P1, P6, P2 y P4. Cada header posee 12 pines en los que se dividen los GPIOs, un pin con una referencia GND y un pin con voltaje de 3.3V.

Existen otros módulos secundarios que complementan la FPGA que son utilizados por aplicaciones muy específicas, estos módulos son:

- Módulo de salida de audio conector de 3.5 mm
- Módulo de tarjeta SD
- Módulo display de 8 segmentos. (Estos son tres display unidos a un mismo módulo)

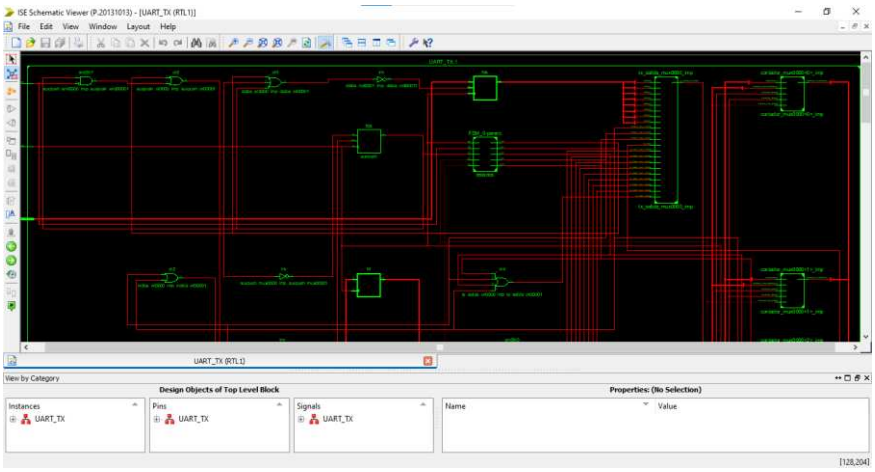
- Módulo de conector JTAG

1.2.2. *Software* utilizado para la programación de la tarjeta Elbert V2

Xilinx al ser la encargada de la creación del chip que lleva tarjeta Elbert V2 proporciona un programa licenciado que facilita el programar y cargar los programas para la FPGA. El *software* diseñado por Xilinx tiene el nombre de ISE Design Suite, dicho *software* es capaz de compilar los programas antes de cargarlos a la tarjeta, puede simular dichos programas mostrando cómo funcionan las salidas y entradas de la tarjeta y es capaz de devolver un diagrama esquematizado de todos los módulos programados para un fácil entendimiento.

Figura 6.

Esquema de un programa de módulo UART en ISE Desing Suite



Nota. En la figura se puede observar un circuito creado a partir de código utilizando ISE Design Suite 14. Elaboración propia, realizado con ISE Design Suite 14.

El *software* ISE Design Suite tiene tres posibles formas de programarse, dos de estas formas es utilizando un lenguaje descriptivo de *software* los cuales son:

- VHDL
- Verilog

Además de los lenguajes mencionados con anterioridad se puede programar la FPGA utilizando bloques que representen compuertas preprogramadas con el programa incluido en el ISE Design Suite llamado ISE iMPACT, este método facilita mucho la tarea de poder crear un programa funcional en una FPGA pero tendrá la limitante de solo poder utilizar las compuertas preprogramadas en el sistema, por eso este método es recomendable solo para familiarizarse con el comportamiento de la FPGA.

Dentro del programa de diseño creado por Xilinx existe la capacidad de crear un archivo con todas las especificaciones del comportamiento del chip enlazadas con el comportamiento de los periféricos de la tarjeta. Este programa es de tipo .bin y es capaz de almacenar un conjunto de instrucciones específicas en forma binaria, es uno de los formatos más utilizados especialmente por los procesadores ya que todas las instrucciones al estar en el tren binario del archivo se pueden desencriptar y convertir a las señales eléctricas que representen las combinaciones de 1 y 0. Es de suma importancia este tipo de herramienta ya que una vez finalizada la programación y asignación de entradas y salidas solo se debe generar el archivo para cargarlo a la FPGA y poder tener el comportamiento deseado en la tarjeta.

2. CONCEPTOS BÁSICOS DE VHDL

Para poder realizar programas y circuitos en las FPGA es de vital importancia conocer las posibles instrucciones, comandos y estructuras que integran el lenguaje que estas utilizan, por este motivo se estudian las instrucciones y conceptos más importantes de VHDL.

2.1. Concepto

Un lenguaje de descripción de hardware es aquel que puede reemplazar en forma de código los diferentes comportamientos típicos de los circuitos lógicos y combinacionales. Estos lenguajes se caracterizan por ser muy similares a los lenguajes de programación típicos con las diferencias de tener que utilizar arreglos con variables que representaran señales de la tarjeta.

2.2. Elementos básicos de VHDL

Al ser un lenguaje de descripción de hardware posee ciertas propiedades que lo hacen único comparados con otras estructuras de lenguajes de programación. Dichas estructuras pueden ser capaces de describir los comportamientos de entradas y salidas de los puertos, pueden describir las instrucciones que ejecutaran o bien pueden indicar los distintos tipos de variables a utilizar. Los elementos que conforman el lenguaje VHDL son los siguientes:

- Entidad

- Arquitectura
- Objetos

2.2.1. ¿Qué es una entidad?

Una entidad es la forma en que se pueden representar los circuitos en VHDL mencionando solamente las entradas y salidas de dicho circuito ignorando su funcionamiento. Sánchez-Élez (2024) indica que es “Un sistema digital está descrito por sus entradas y sus salidas y la relación que existe entre ellas” (p.9).

La estructura de una entidad la conforman tres partes, todas estas juegan un papel importante para su funcionamiento y se pueden describir de la siguiente manera:

- Nombre: esta parte indicara como se le llamara a la entidad, es importante escoger un nombre identificable para el programador ya que será utilizada en la aplicación de grandes programas que involucren varias entidades.

- Valores genéricos: los valores genéricos son valores que permanecen fijos en el circuito, son utilizados para establecer longitudes de vectores o para establecer valores que se utilizaran como referencias dentro de la arquitectura del programa. Esta parte de la entidad puede ser omitida sin afectar el funcionamiento del sistema.

- Puertos: estos son todos los puertos de entrada o salida del circuito que se está representando. Tienen la peculiaridad de no poder ser utilizados dentro de la arquitectura para instrucciones lógicas de forma directa ya

que se deben almacenar sus valores en objetos auxiliares y no se puede altera su propiedad de entrada o salida dentro de la arquitectura.

Para poder tener un mejor entendimiento de la entidad se puede describir el circuito mostrado en la Figura 7, dicho circuito consta de una entrada de un único bit y una salida simple de un bit también. Lo que se desea realizar es un circuito que encienda un led cuando se pulse un botón. Para la entidad no es de relevancia que ocurre dentro del circuito, lo único que interesa es saber quiénes son las entradas, salidas y si existen constantes en el circuito, en este caso su entrada es denominada como btn y su salida es led.

Figura 7.

Presentación de la caja negra del circuito de ejemplo

Nota. En la figura se puede observar el circuito mencionado en el ejemplo como una caja negra donde no es de interés su contenido, solo importan sus salidas. Elaboración propia, realizado con Word.

Figura 8.

Entidad en VHDL del circuito de ejemplo

```
6   entity EJEM1 is
7       Port ( btn : in  STD_LOGIC;
8              led : out  STD_LOGIC);
9   end EJEM1;
```

Nota. En la figura se puede observar la entidad programada utilizando VHDL. Elaboración propia, realizado con ISE Design Suite 14.

Como se puede observar en la Figura 8 al momento de trasladar el circuito del ejemplo al lenguaje VHDL se puede observar la estructura a seguir. Se puede detallar paso a paso las instrucciones de la siguiente manera:

- Primer paso: se debe detallar el nombre de la Entidad, en este caso es EJEM1. Se sigue la sintaxis presentada colocando el nombre tanto al inicio como al final del módulo y es de suma importancia que al finalizar dicho módulo se indique con un; ya que de no seguir la sintaxis VHDL interpreta que aún se está asignando puertos a la entidad.

- Segundo paso: en este paso es donde deben definirse las entradas y salidas del circuito, según el circuito de la Figura 7 se tiene una entrada y una salida, al momento de definirlas se debe redactar dentro de los puertos haciendo uso de la instrucción Port();. Dentro de Port es donde se definen el tipo de dato que se desea utilizar y se debe indicar si es entrada o salida, en el caso del ejemplo se desea utilizar un botón como entrada se ha colocado el nombre de la entrada como btn y se hace referencia que es entrada utilizando el comando in, en el caso de la salida se hará con un led y por este motivo el nombre de la salida es ese mismo nombre led y se debe colocar como una salida utilizando el

comando out. Con esto declarado la entidad queda lista para poder ser utilizada por la arquitectura.

La estructura de la entidad es simple de entender ya que no interesa realmente como se utilizarán las constantes o como llegara la información de la entrada y salida, solamente interesa establecer las propiedades básicas del circuito que se desee crear para que la FPGA pueda conectar los puertos a los pines que posea la tarjeta donde ha sido instalada.

2.2.2. ¿Qué es una arquitectura?

La arquitectura de un programa de VHDL es donde se realiza toda la programación de las instrucciones del circuito presentado en la entidad. En este apartado se puede indicar que acciones se realizaran con las señales de entrada que se obtienen, como será tratada la señal o si es necesario compararla con una constante y cuando se finalizan todas las instrucciones se debe indicar a que salidas de la entidad debe ir cada señal.

En las arquitecturas se debe considerar que las señales que se obtienen de la entidad no se pueden utilizar de forma directa en las instrucciones, siempre se deben utilizar variables auxiliares que sean del mismo tipo de la variable de entrada o salida. Cuando se quiere registrar una entrada de la entidad dentro de la arquitectura se debe realizar un proceso de carga donde se colocar el símbolo <= para ejemplificar la carga de una entrada a una variable auxiliar y del mismo modo la variable auxiliar se debe cargar a la salida haciendo uso del mismo símbolo.

Para lograr un mejor entendimiento de las funciones de una arquitectura se puede retomar el ejemplo dado en la entidad donde se desea realizar un

circuito que lea una entrada por medio de un botón y refleje su salida a través de un led. Como se observa en la Figura 9 se tiene la estructura de la arquitectura que realiza la acción deseada, se puede enumerar los paso a seguir para armar una correcta arquitectura de la siguiente manera:

- Primer paso: se debe especificar el inicio de una arquitectura haciendo uso del nombre de la Entidad de donde se tomarán las entradas y salidas, la forma de escritura se puede notar en la línea 11 de la Figura 9 donde se puede ver cómo se ha hecho mención de EJEM1 que es el nombre de la entidad del ejemplo anterior. Se debe delimitar el inicio y el final de la arquitectura antes de agregar instrucciones para mantener el orden, la forma de hacerlo es utilizando el comando begin y después un end que debe hacer mención al nombre del comportamiento del circuito tal y como lo muestra la línea 19.

- Segundo paso: una vez se ha establecido el inicio y el final de la arquitectura se deben crear las señales auxiliares que darán la opción de trasladar la información de los pines de entrada a los de salida. En el ejemplo se puede notar en la línea 13 una señal llamada oled que es de tipo std_logic que es el mismo utilizado por las entradas y salidas, esto se crea de esta forma ya que al ser un programa que simplemente debe trasladar la información de un botón a un led la única operación que se debe realizar es la carga y descarga de la información. Una vez se crean las señales auxiliares se puede proceder al siguiente paso.

- Tercer paso: en este paso es donde se inician todas las instrucciones del comportamiento de la arquitectura, este paso es el que más puede variar en el diseño de los circuitos por lo simples o complejos que pueden llegar a ser. Para el caso del ejemplo lo único que se hace es

tomar la entrada btn y se le asigna a oled utilizando el símbolo de carga <= y finalizando la línea con un; tal y como muestra la línea 16. Cuando ya se tiene de forma interna la información del botón lo único que resta es asignarlo a la salida, para esto se toma la señal oled y se carga con el símbolo <= a la señal de salida led y con esto se consigue trasladar la información del botón al led de salida.

Figura 9.

Arquitectura en VHDL del circuito de ejemplo

```
11  architecture Behavioral of EJEM1 is
12
13  signal oled: std_logic;
14
15  begin
16     oled <= btn;
17     led <= oled;
18
19  end Behavioral;
```

Nota. En la figura se puede observar el comportamiento de la entidad programada utilizando VHDL. Elaboración propia, realizado con ISE Design Suite 14.

Como se puede observar el apartado de la arquitectura es un poco más complejo por la descripción que se debe realizar, pero una vez se logran dominar todos los conceptos que sirvan para ejecutar instrucciones en VHDL es posible crear circuitos multifuncionales con solo escribir las instrucciones precisas.

2.2.3. ¿Qué son los objetos?

Los objetos dentro de un programa VHDL son utilizados para poder representar señales, constantes o valores numéricos que serán útiles para

realizar la representación de los circuitos dentro de la FPGA. Existen tres tipos de objetos que se pueden utilizar, estos objetos tienen diferente tipo de uso dentro de los programas, dichas categorías se pueden clasificar en:

- Constant
- Variable
- Signal

Dichas categorías se pueden evaluar de forma individual para lograr una mejor comprensión de estas.

2.2.3.1. Objeto Constant

Este tipo de objeto son constantes o valores constantes que serán utilizados dentro del programa. Se le asigna un valor al inicio del programa y posee la característica de que dicho valor no cambia a lo largo del programa. Para este tipo de objetos no es posible su modificación dentro del programa, tendrán un valor fijo siempre. Son útiles para marcar ciclos de reloj, como valores de referencia para comparaciones o simplemente como un valor numérico constante que se desee utilizar en una operación matemática.

Figura 10.

Uso de constantes en VHDL del circuito de ejemplo

```
30  architecture Behavioral of ejemplo1fisico is
31     constant novariable: std_logic := '1';
32  begin
33
34  LED_ONOFF: process (push_button)
35  begin
36
37     if push_button = novariable then
38        led <= '1';
39     else
40        led <= '0';
41     end if;
42
43  end process LED_ONOFF;
44  end Behavioral;
```

Nota. En la figura se puede observar la implementación de una constante al programa utilizando VHDL. Elaboración propia, realizado con ISE Design Suite 14.

El circuito que se ha mostrado en la Figura 10 se muestra cómo se puede realizar la implementación de una constante, esto se realiza en la línea 31, para el ejemplo es una constate llamada novariable. Se debe destacar la ubicación de la declaración de la constante, siempre debe ser antes del inicio de la arquitectura y justo después de la declaración de esta. El tipo de dato que esta constante almacena sirve para realizar una comparación para evaluar si se debe enviar un 1 o un 0 a la salida del circuito.

2.2.3.2. Objeto Variable

Los objetos son variables son valores que pude ser modificables durante la ejecución del programa, estos valores solo sirven para una lógica interna como puede ser el conteo de los datos, variables que delimiten los ciclos, procesos o una comparación con algún valor cambiante en el programa. Las variables tienen la característica de no poder ser asignados directamente

a los pines IO de la FPGA, pero pueden funcionar como enlaces entre las entradas y salidas de esta.

Figura 11.

Uso de variables en VHDL del circuito de ejemplo

```
30  architecture Behavioral of ejemplo1fisico is
31  begin
32  LED_ONOFF: process (push_button)
33  variable var: std_logic := '1';
34  begin
35
36     if push_button = '1' then
37        var := push_button;
38        led <= var;
39     else
40        var := push_button;
41        led <= var;
42     end if;
43
44  end process LED_ONOFF;
45  end Behavioral;
```

Nota. En la figura se puede observar la implementación de una variable al programa utilizando VHDL. Elaboración propia, realizado con ISE Design Suite 14.

El ejemplo mostrado en la Figura 11 hace uso de una variable llamada var. Para poder declarar correctamente una variable se debe conocer en qué proceso será utilizada. Las variables en VHDL son de uso exclusivo de sus procesos, por esta razón deben declararse después de abrir el proceso y antes de comenzar las instrucciones de este, tal como se realiza en el ejemplo al ser colocado en la línea 33.

La variable var se declara inicialmente con un valor de 1. Una vez declarada se pueden cargar datos tal como se muestra en las líneas 37 y 40 utilizando el := para que VHDL pueda entender que se desea cargar el valor de la derecha a la variable, en este caso es el valor de push_button. Por último,

la variable vuelve a estar presente enlazando directamente el valor recibido de la entrada del circuito con la salida de este.

Como se pudo validar en el ejemplo las variables son únicamente de uso interno y se tiene la libertad de sobrescribir los datos y enviarlos a las salidas según sean las necesidades del circuito.

2.2.3.3. Objeto Signal o Señales

Este tipo de objeto tal y como su nombre lo indica son las señales que están en el circuito. Con señales se refiere que son las que pueden ser utilizadas para una escritura y lectura de entradas, salidas, elementos de memoria o cualquier componente de circuito que se desee. Lo que diferencia a una señal de una variable es el comportamiento que poseen en simulaciones, cuando se simula una señal VHDL lo toma como una señal física real y el comportamiento de dicha señal será la que corresponda a su asignación, la variable solo sirve para almacenar valores y no para simulación. Al momento que VHDL sintetice el circuito tomara las señales y las tratara como puertos físicos dentro de la lógica de VHDL. Se opta más por el uso de señales en los circuitos por la facilidad que estas ofrecen en el traslado de información.

Figura 12.

Uso de señales (signal) en VHDL del circuito de ejemplo

```
30  architecture Behavioral of ejemplo1fisico is
31  signal senal: std_logic;
32  begin
33  LED_ONOFF: process (push_button)
34  begin
35
36     if push_button = '1' then
37        senal <= push_button;
38        led <= senal;
39     else
40        senal <= push_button;
41        led <= senal;
42     end if;
43
44  end process LED_ONOFF;
45  end Behavioral;
```

Nota. En la figura se puede observar la implementación de una señal al programa utilizando VHDL. Elaboración propia, realizado con ISE Design Suite 14.

En el ejemplo mostrado en la Figura 12 se puede observar el proceso de la creación de una señal dentro de VHDL en la línea 13. Salta a la vista una de las diferencias comparado con una variable ya que la señal se pude crear sin ningún valor inicial especifico y pueden ser utilizadas en toda la arquitectura del programa. El proceso de asignación de información para las señales se puede observar en la línea 37 y 40. Las señales en VHDL ofrecen una gran facilidad de uso por ser del mismo tipo que las señales iniciales y generalmente siempre serán puentes de enlace entre entradas y salidas.

2.2.4. Tipos de datos en objetos

Los tipos datos en objetos son los que van a determinar el comportamiento de los datos que serán almacenados dentro de los objetos. Existen múltiples datos posibles de almacenar dentro de un objeto, pueden ser

desde datos numéricos a datos lógicos. La relación entre objetos en VHDL solo es posible de lograr si estos objetos poseen el mismo tipo de dato, esto debe ser una consideración importante al momento de realizar cualquier asignación o proceso lógico dentro de la programación y es por eso que toma importancia el conocer y entender el funcionamiento de los tipos de datos que existen dentro de VHDL.

2.2.4.1. Bit

Este es un tipo de dato solo admitirá valores como el 0 o 1. Los objetos que tengan asignado dato Bit no podrán asignarse a salidas ni tampoco se podrán cargar datos de entrada. La forma de asignar valores a este tipo de objetos será utilizando siempre comillas simples, ya sea 1 o 0.

2.2.4.2. Bit vector

Este es un tipo de dato vectorial, lo que quiere decir que puede almacenar un arreglo de datos que contengan valores como el 0 o 1. Los objetos que tengan asignado dato Bit no podrán asignarse a salidas ni tampoco se podrán cargar datos de entrada. Su tamaño lo delimita el rango y estos deben ser delimitados por el usuario, la forma más común es utilizar la delimitación downto para marcar el inicio y el final del vector. El tamaño recomendado de uso para este tipo de vector y para el tipo de tarjeta que es utilizada es de 2^16 que dará un total de 65536 bits de información. Lo recomendable es utilizar solo hasta 2^8, puede soportar más, pero en algunos casos por la cantidad de procesamiento de la tarjeta se podría ocasionar un error en el compilador. El modo de cargar valores a este tipo de dato es utilizar siempre las comillas para indicar que valor es el que se ingresara y debe de ingresarse cadenas que contengan valores 1 y 0.

2.2.4.3. String(rango)

A diferencia de otros lenguajes donde string sería una cadena de texto útil para almacenar cierta información en VHDL no tiene mucho uso este tipo de objetos, ya que VHDL al ser un lenguaje que describe hardware el uso de string es realmente poco. Un objeto de tipo string no es más que una matriz de caracteres y la forma de ingresar valores a estos objetos es haciendo uso de las comillas.

Su uso es enfocado a la simulación a modo de banderas o marcas que ayuden a identificar en que parte del proceso se encuentra el programa a la hora de realizar la simulación.

2.2.4.4. Integer

Se utiliza este tipo de objeto al momento en que se necesita trabajar con números enteros dentro de la descripción del programa, es útil para poder utilizar librerías matemáticas para realizar cálculos dentro de VHDL. El valor máximo y mínimo va delimitado por el número 2147483647, este número es el máximo valor que un objeto integer puede almacenar.

2.2.4.5. Std_logic

Este es el objeto más común para utilizar en programas VHDL porque posee la lógica de 9 posibles valores en un mismo tipo de dato. Este tipo de objeto también será el predefinido para enlazar el programa con el hardware descrito, se utiliza tanto para entradas como para salidas.

2.2.4.6. Std_logic_vector(rango)

Este objeto no es más que la forma vectorial del std_logic. Posee las mismas características con la diferencia que este almacenara varios valores lógicos. Al momento de utilizarlo se debe tener en cuenta la cantidad de entradas y salidas que se le asignen ya que en las tarjetas de Xilinx no puede quedar ningún espacio del vector sin ser asignado. El modo para declarar el rango que abarca este objeto es similar al que se utiliza en el de bit_vector.

Todos los tipos de objetos vistos con anterioridad son útiles para lograr una buena descripción de *software* haciendo uso de las herramientas de Xilinx. En la Figura 13 se puede ver una arquitectura dedicada solamente a explorar la sintaxis para la creación de los objetos mencionados con anterioridad y se muestra también como debe ser el ingreso de datos a estos objetos.

Figura 13.

Ejemplo de declaración e ingreso de datos a los objetos

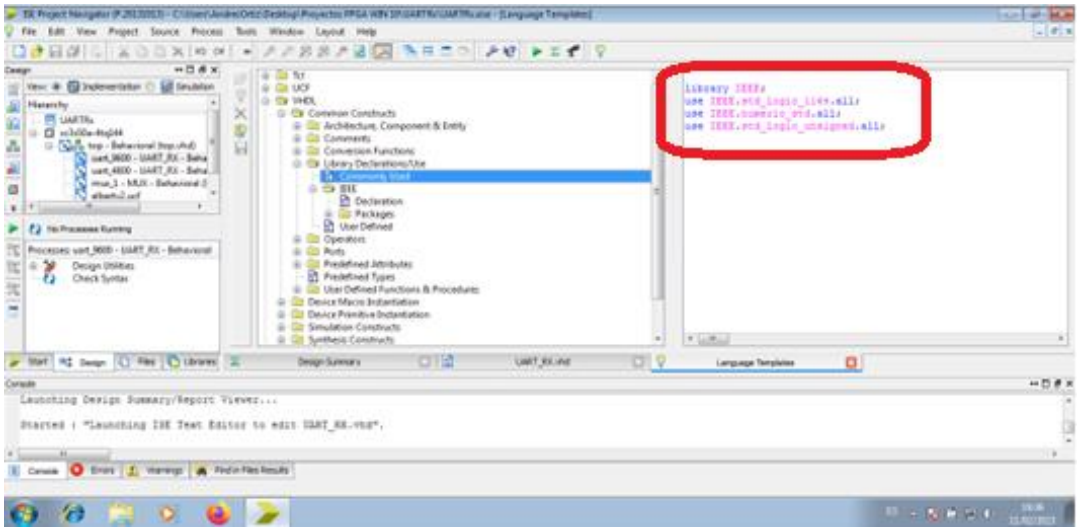
```
28  architecture Behavioral of ejemplo1fisico is
29
30  signal bt : bit ; --Crea una objeto de tipo señal que almacene bit.
31  signal btv: bit_vector (7 downto 0); --Crea un objeto de tipo señal de N=8 bits
32  signal str: string(1 to 10);
33  signal int: integer; --Crea un objeto integer normal
34  signal intrange: integer range 0 to 8 ; --Crea un objeto integer limitado de 0  a 8, no puede almacenar mas
35
36  begin
37
38  bt <= '1'; --La asignacion se realiza utilizando solo comilla simple.
39  btv <= "10001010"; -- En este caso se utiliza comilla  doble y siempre se debe llenar la N cantidad con la que se creo.
40  str <= "academicos"; --Se ha colocado una cadena de 10 letras y siempre se debe cumplir con la cantidad.
41  int <= -2147483647; --Se puede ingresar numeros desde -2147483647 hasta 2147483647
42  intrange <= 8; --Debido al rango  solicitado solo se pueden ingresar valores del 0 al 8
43
44  end Behavioral;
45
46
```

Nota. En la figura se puede observar el método de ingreso de los diferentes tipos de objetos utilizando VHDL. Elaboración propia, realizado con ISE Design Suite 14.

Para lograr la correcta funcionalidad de cada uno de los objetos es necesario hacer uso de las librerías IEEE. Dichas librerías pueden ser incluidas de forma manual o utilizando la herramienta de ISE Design Suite, la forma de agregarla es buscando el icono de foco de luz dentro de ISE, buscar la capeta VHDL, luego Common Constructs y seleccionar la carpeta Library Declarations/Use para poder desplegar el texto, en la Figura 14 se puede ver dicha selección.

Figura 14.

Librerías IEEE implementadas en VHDL



Nota. En la figura se puede observar la forma de agregar librerías en ISE Design Suite. Elaboración propia, realizado con ISE Design Suite 14.

2.2.5. Tipos de sentencias condicionales en VHDL

Las sentencias condicionales son utilizadas en los programas para tomar decisiones, dependiendo de los valores de las entradas pueden optar por varias rutas de solución. Las sentencias en VHDL funcionan de forma similar con la diferencia que estas pueden interpretar los objetos y dar salida a estos mismos objetos que pueden interactuar directamente con los pines de salida y entrada de la tarjeta FPGA. Las sentencias condicionales más comunes existentes en VHDL son IF y CASE.

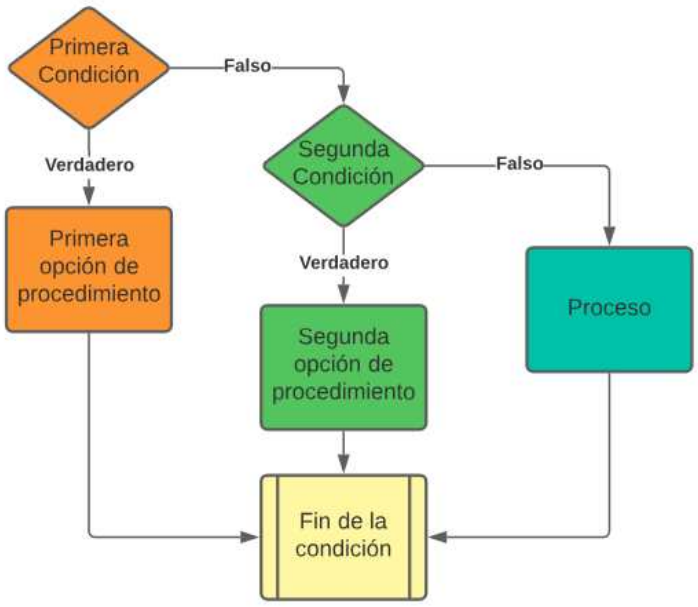
2.2.5.1. IF

La sentencia IF es similar a las conocidas en otros lenguajes, al momento de iniciar dicha sentencia realiza una consulta lógica, si la consulta es verdadera puede ejecutar una serie de instrucciones y finalizar. Si la consulta es falsa se pueden tomar dos rutas, la primera es hacer una segunda comparación lógica y de ser verdadera se ejecutan las instrucciones correspondientes y la segunda opción es no hacer ninguna otra comparación y simplemente ejecutar las instrucciones que prosiguen en el programa.

Para todo IF se puede usar la analogía en la que se supone que el programa realiza la pregunta, ¿Esto es verdadero? ¿Sí o no? y con la respuesta se pueden derivar varios escenarios. En la Figura 15 se puede observar un pequeño diagrama lógico del funcionamiento.

Figura 15.

Lógica de la sentencia IF



Nota. En la figura se puede observar el diagrama de flujo de una sentencia IF. Elaboración propia, realizado con Lucidchart.

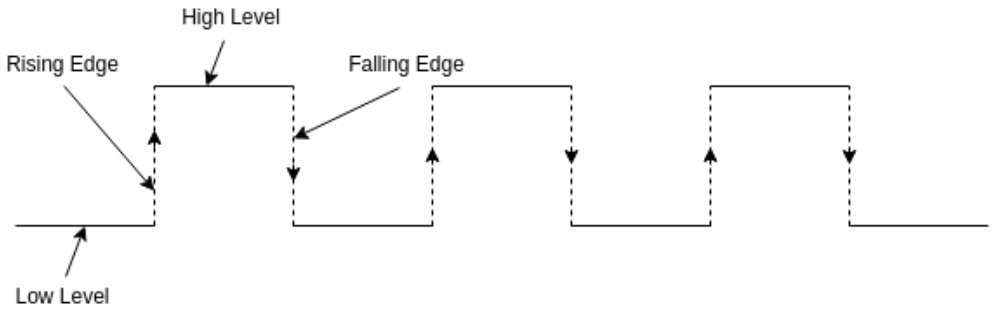
La sentencia IF es comúnmente utilizada, además de las comparaciones, para medir los ciclos del reloj interno de las tarjetas FPGA. La forma de utilizarla es realizando una evaluación de la serie completa de pulsos y se busca si ocurrieron los siguientes tres estados:

- Estado de flanco ascendente (rising edge): es una transición que se mide dentro de los ciclos del reloj, se basa en detectar los momentos que pasan de un estado bajo a un estado alto.

- Estado de flanco descendente (falling edge). es una transición que se mide dentro de los ciclos del reloj, se basa en detectar los momentos que pasan de un estado alto a un estado bajo.

Figura 16.

Explicación gráfica de un Rising Edge y un Falling Edge



Nota. En la figura se puede observar el diagrama de reloj que muestra los flancos ascendentes y descendentes. Obtenido de TeachIcs . *Edge triggering and pulse triggering*. (https://teachics.org/computer-organization-and-architecture-tutorial/edge-triggering-and-pulse-triggering/), consultado el 12 de febrero de 2023. De dominio público.

Los conceptos de los flancos es un tema que será retomado al momento de realizar la escritura de los programas de ejemplo que brinden una mayor explicación del uso del reloj interno de las FPGA.

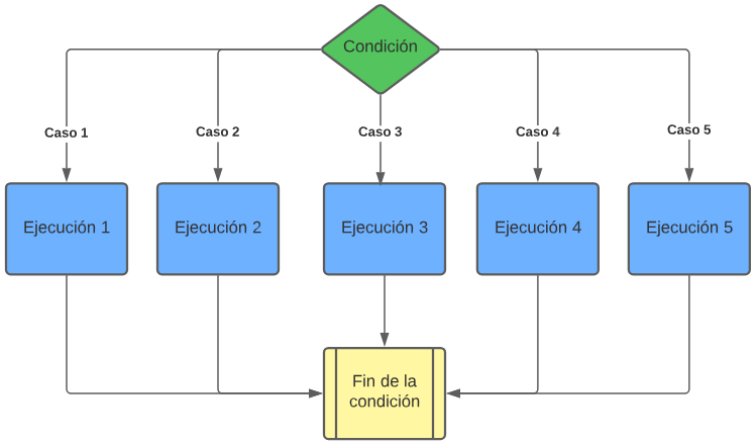
2.2.5.2. CASE

La sentencia CASE es semejante a tener varios IF anidados, esto quiere decir que realiza un recorrido valor por valor preguntando cual de todas las sentencias es la verdadera. Se puede diferenciar de una sentencia IF por el tipo de facilidad que ofrece para realizar múltiples comparaciones, toma una variable principal para evaluar y realiza una inspección de varias sentencias

para encontrar cual coincide con el valor de la variable principal para después ejecutar la serie de tareas que están asignados a la sentencia que es verdadera verdadero.

Figura 17.

Lógica de la sentencia CASE



Nota. En la figura se puede observar el diagrama de flujo de una sentencia CASE. Elaboración propia, realizado con Lucidchart.

Al analizar la lógica de la sentencia CASE se observa que opera en forma paralela las posibles condiciones que pueda cumplir la variable que se está analizando, gracias a esta forma de operar es más rápida y efectiva de utilizar, comparado con una sentencia IF que para realizar la misma operación se tendría que evaluar paso por paso las condiciones hasta llegar a la sentencia que cumpla y sea ejecutado el procedimiento. El uso de CASE es común en el análisis de vectores ya que brinda la facilidad de evaluar todos los valores del vector realizando la conexión solamente de la entrada deseada y con esta realizar todas las comparaciones en un único nodo, esto hace el uso de CASE más eficiente comparado con el IF.

2.2.6. Tipos de ciclos programables en VHDL

Los ciclos en programación son funciones que permiten realizar en bucle las acciones que se programen dentro de estos mismos. Los ciclos son útiles ya que al volver un conjunto determinado de acciones en un ciclo repetitivo se puede lograr la creación de un proceso que funcionara de forma automática, muchos programas al no tener ciclos incorporados realizan la acción repetitiva solo cuando se inicia el programa y para repetir la acción debe ser apagado y encendido nuevamente, los ciclos eliminan esta necesidad al poder repetir todo el conjunto de instrucciones una vez se llegue al final de estas.

Los bucles en VHDL funcionan de una forma especial, por la arquitectura que poseen las tarjetas y el modo de funcionar de una FPGA todo lo que se programe está en bucle sin necesidad de utilizar los ciclos, por ejemplo, si se escribe un simple programa que asigne la entrada directamente a la salida y no se coloca dentro de un ciclo puede aun así funcionar en bucle. Esto se debe a que el lenguaje VHDL describe circuitos está diseñado de tal forma que dichos circuitos se actualizan siempre a la misma frecuencia del reloj interno de la FPGA.

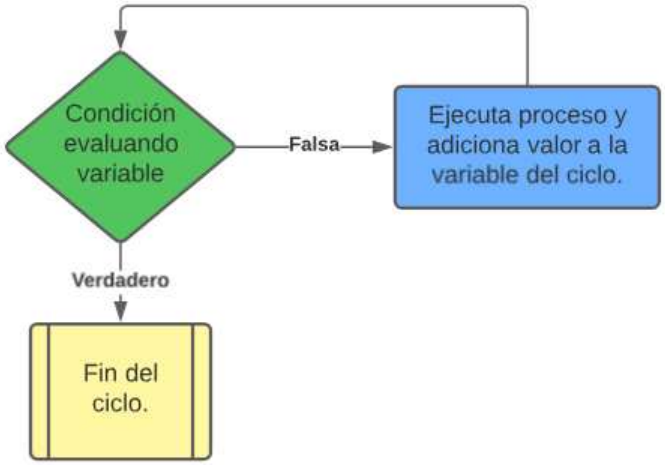
En VHDL existen únicamente dos ciclos, estos son el For-Loop y While-Loop, la lógica y funcionamiento es diferente para cada uno de ellos y por esta razón se hace uso de estos dependiendo de las aplicaciones que se requieran realizar, dichas aplicaciones y usos se abordaran más adelante en este mismo trabajo.

2.2.6.1. For-Loop

El ciclo For posee un funcionamiento en el cual es necesario utilizar una lista de valores que son recorridos por la sentencia. El ciclo consiste en tomar una lista de valores, delimitar la lista con un inicio y final y recorrer los valores de la lista uno a uno hasta que coincida con la sentencia o llegue al valor final de la lista.

Figura 18.

Lógica del ciclo For



Nota. En la figura se puede observar el diagrama de flujo de un ciclo For. Elaboración propia, realizado con Lucidchart.

Se puede observar en la Figura 18 la lógica de funcionamiento de los ciclos For. Este ciclo debe utilizar una variable como condición inicial, usualmente es una variable vectorial y puede contener distintos valores y tener rangos modificables, a fines de mejorar la explicación se puede suponer que tiene valores de 0 a un valor máximo N. Una vez inicie el ciclo se realiza la

condición de evaluar si el contador del ciclo ha llegado ya al valor de N, de no ser el caso el ciclo ejecuta el proceso interno y al final adiciona un valor a la variable que se condiciona en el inicio el ciclo, en la segunda ejecución vuelve a cuestionarse el valor de la variable y de no ser verdadera nuevamente se ejecuta el proceso interno del ciclo y finalmente se adiciona de nuevo el valor. El ciclo repetirá el proceso de ejecución y adición hasta que la variable haya llegado a igualar el valor de N propuesto en la condición.

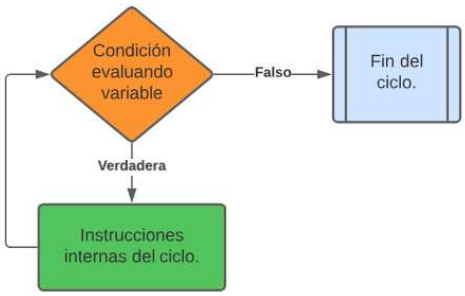
Este tipo de ciclos recorren todos los valores de un rango y por esta razón son útiles para llenar las casillas de variables vectoriales y puede ser útil para automatizar el recorrido de todos los valores de estos vectores. Si lo que se busca es que el ciclo finalice una vez se cumpla una condición especifica lo ideal será utilizar otro tipo de sentencia ya que utilizar el ciclo For para esta tarea no es lo más practico ni lo más sencillo.

2.2.6.2. While-Loop

El ciclo While utiliza una condición inicial para su funcionamiento, mientras esta condición sea verdadera el ciclo se repetirá, si por alguna razón dentro de las instrucciones de dicho ciclo algo llegara a modificar la variable que utiliza la condición el ciclo terminara hasta su última instrucción y al iniciar de nuevo la comparación tomara dicho valor y terminara el ciclo.

Figura 19.

Lógica del ciclo While



Nota. En la figura se puede observar el diagrama de flujo de un ciclo While. Elaboración propia, realizado con Lucidchart.

Tomando como referencia el diagrama mostrado en la Figura 19 se puede dar una mejor explicación del comportamiento del ciclo. Usualmente se coloca una variable dentro de la condición inicial del ciclo, dicha variable es cambiada dentro de las instrucciones internas cuando ocurre un cambio que sea de interés para el diseñador del programa, por lo general se utiliza cuando se llega al valor deseado dentro del ciclo. Un detalle importante que debe ser resaltado en el ciclo While es que la variable de la condición puede cambiar en cualquier lugar dentro del conjunto de instrucciones internas, pero no significa que el ciclo terminara inmediatamente ya que solo se repetirá una vez finalicen todas las instrucciones que deban ejecutarse internamente en el ciclo. Una vez termina dicha ejecución la variable al entrar en la condicional y modificar su valor arrojara una respuesta falsa, al momento de hacer esta acción el comparador envía directamente al final del ciclo sin ejecutar ninguna otra instrucción.

El uso común de este tipo de ciclos es el despliegue de algunos valores o la creación de retardos que son útiles para realizar pausas. Una ventaja que

ofrece una FPGA comparado con un microcontrolador es que puede ejecutar estos ciclos de forma paralela a otras funciones, lo que es especialmente útil si se desean realizar múltiples tareas en un mismo proyecto.

2.2.7. Sentencias especiales de VHDL

Las sentencias especiales en VHDL hacen referencia a instrucciones capaces de reducir circuitos en los programas que se escriban. Existen dos tipos de funciones que permiten realizar subprogramas en los cuales pueden interactuar las entradas y salidas, son especialmente útiles para lograr que la tarjeta FPGA trabaje varios circuitos en forma paralela. Lo más parecido a este tipo de instrucciones son las funciones en otros lenguajes, se puede programar todo un procedimiento y dicha función lo único que esperará será una entrada para devolver una salid. En VHDL las dos funciones especiales a las que se hace referencia son Function y Process.

2.2.7.1. Sentencia Function

La sentencia function realiza un procedimiento en el cual con unas entradas puede generar una salida. Entre sus características esta que realizan una acción sin que corra el tiempo, esto da como limitación que no puedan generarse pausas dentro de estas mismas, son de una única ejecución. Solo pueden utilizar los parámetros que se declaren en un inicio y no pueden interactuar con variables externas a la función.

El uso general de este tipo de sentencias se da cuando el interés de la creación del circuito es obtener una salida especifica de algún tipo o categoría de objeto, solo puede entregar como salida la que se especifica al inicio del proceso, no puede operar salidas que no estén declaradas lo que hace su uso

limitado. Si se utilizara esta sentencia siempre es recomendable conocer de antemano que salida se busca y que entradas van a interactuar con esta salida, solo de esta forma se lograra obtener un funcionamiento eficiente para la función. En la Figura 20 se puede observar la estructura de código que se debe seguir para la creación de una función.

Figura 20.

Estructura de la sentencia especial Function

```
function identificador(…) return tipo
-- señales, variables
begin
-- cuerpo del programa
-- se puede utilizar cualquier sentencia
-- propia de VHDL
return valor;
end function identificador
```

Nota. En la figura se puede observar la sintaxis de la sentencia Function. Obtenido de Facultad de Informática Universidad Complutense de Madrid. (2014). *Introducción A La Programación en VHDL*. p. 61.

2.2.7.2. Sentencia Process

La sentencia Process son descripciones de circuitos dentro de VHDL que no fuerzan una salida dentro de los parámetros iniciales, utilizan una variable sensitiva como entrada (o varias variables) y se activa el proceso que se programe dentro de estos cada vez que la variable sensitiva de entrada realice algún cambio o alteración. Permite realizar pausas internas, utilizar variables fuera del proceso y también permite utilizar señales de entrada o salida que no sean declaradas dentro de los parámetros iniciales. En la Figura

21 puede observarse la estructura con la que debe ser escrita para que VHDL la tome como válida.

Figura 21.

Estructura de la sentencia especial Process

```
process (lista de sensibilidad)
begin
-- código de descripción
end process;
```

Nota. En la figura se puede observar la sintaxis de la sentencia Process. Obtenido de Facultad de Informática Universidad Complutense de Madrid. (2014). Introducción A La Programación en VHDL. p. 23.

Se puede observar que a diferencia de Function la sentencia Process es más fácil de utilizar y de comprender. En general la variable sensitiva que va dentro de los parámetros del Process es siempre una asignada a un botón o la que se le asigna al reloj interno. Se hace de esta forma para que VHDL solo ejecute las instrucciones internas a menos que exista una acción inicial que altere la variable sensitiva. Es recomendable colocar en esta lista sensitiva todas las entradas que se vayan a utilizar dentro del proceso y se debe recalcar que, si se altera una salida estando dentro de Process, esta salida no podrá ser modificada en ningún otro sitio que no sea dentro de ese mismo Process ya que VHDL lo ejecutara en forma paralela. La función Process es la más utilizada en VHDL por la facilidad que da para trabajar de forma paralela, permite tener múltiples circuitos que operen de diferentes formas, con diferentes salidas y con diferentes frecuencias de reloj en un mismo programa. Más adelante se explorarán más usos para esta función especial.

3. EJECUCIÓN DE INSTRUCCIONES VHDL EN LA FPGA ELBERT V2

En los dos capítulos anteriores se han abordado todos los temas relacionados con VHDL y de las FPGA de forma completamente teórica con muy pocos ejemplos prácticos. En este capítulo se explorará un poco más el cómo se puede trasladar un código en VHDL a la tarjeta Elbert V2, se evaluará el *software* necesario para lograr el correcto funcionamiento de ISE Design Suite y la sintaxis de las instrucciones previamente descritas.

3.1. Uso y recomendaciones de ISE Designe Suite

Como se describió en el primer capítulo de este documento, ISE Design Suite es un *software* de desarrollo que puede compilar código VHDL y puede implementar diseños especiales para las FPGA. Este *software* pertenece a AMD Xilinx, tiene algunas consideraciones importantes que deben ser tomadas en cuenta al momento de instalar dicha herramienta.

Este *software* de desarrollo fue diseñado especialmente para ser utilizado en Windows 7 y conforme pasaron los años dicho *software* no sufrió de grandes modificaciones, por esta razón es particularmente difícil de instalar para las versiones más recientes de Windows, ya que hace uso de aplicaciones que fueron diseñadas originalmente para Windows 7. Por este motivo se dedicará una sección de este capítulo para explicar con algunos detalles el correcto proceso de instalación de este *software*.

3.1.1. Virtualización de un entorno de trabajo

Para lograr que ISE Design Suite no cause inconvenientes al momento de desarrollar las aplicaciones se recomienda crear un espacio virtualizado utilizando cualquiera de las siguientes maneras:

- Utilizar la guía oficial que presenta Xilinx para la instalación de ISE Design Suite 14.7 para Windows 10. Esta guía puede ser encontrada en la página oficial de AMD Xilinx en Soporte/Descargas.

- Utilizar el programa ISE Design Suite 14.7 para Windows 7 (el sistema original) haciendo uso de un computador con Windows 7 como su sistema principal o, la opción más recomendable, utilizar en un computador moderno una máquina virtual con Windows 7 instalado.

Para este trabajo se utilizará el segundo método utilizando una máquina virtual en la que se tendrá un sistema Windows 7 Professional instalado por ser el sistema operativo para el que fue diseñado el programa.

3.1.2. Instalación del sistema operativo utilizando Oracle VM VirtualBox

Oracle VM VirtualBox es un *software* que puede virtualizar entornos de distintos sistemas operativos, esto quiere decir que permite a una computadora tener otro sistema operativo sin necesidad de tener varias particiones en el disco duro o sin desinstalar el sistema operativo que utiliza dicha computadora.

Los pasos para realizar la instalación de este *software* se detallan a continuación:

- Se debe verificar que en la BIOS de la computadora a utilizar se active la opción de virtualización, este proceso puede variar dependiendo de las marcas entonces lo recomendable es realizar una breve investigación de este proceso considerando la marca del computador que se posea.

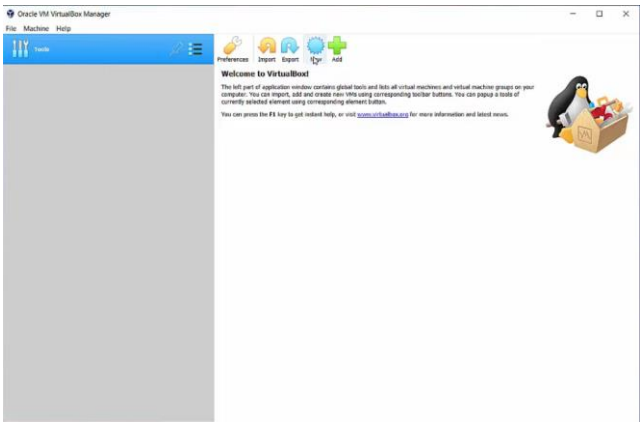
- Descargar el *software* de la página oficial de Oracle VM VirtualBox, se deberá seleccionar el sistema que se esté utilizando en la computadora asignada para este trabajo.

- Una vez se tenga el instalador este deberá ser ejecutado, dentro del instalador no existe ninguna instrucción especial a seguir, por lo tanto, se puede utilizar la guía que este trae por defecto.

- Cuando la instalación finalice es obligatorio el reinicio, una vez este se realice quedara listo el *software* para iniciar la instalación del sistema operativo.

Figura 22.

Pantalla inicial de Oracle Virtual Box



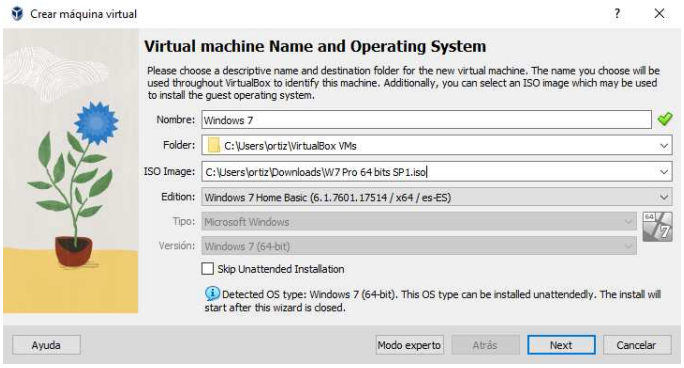
Nota. En la figura se puede observar una captura de la pantalla inicial de Oracle Virtual Box. Elaboración propia, realizado con Oracle Virtual Box.

Una vez se realiza la instalación de la máquina virtual se debe realizar la instalación del sistema operativo Windows 7. Para realizar este proceso se debe contar con una imagen de Windows 7, para realizar este proceso bastara con dirigirse a la página de Microsoft para obtener una imagen oficial del sistema operativo y se debe proceder con la instalación dentro de Oracle VirtuaBox.

Para realizar la instalación bastará con crear una nueva máquina virtual, para hacerlo se puede seleccionar en la barra de opciones Máquina y seguido de esto en Nueva, esta acción deberá desplegar una ventana como muestra en la Figura 23.

Figura 23.

Creación máquina virtual

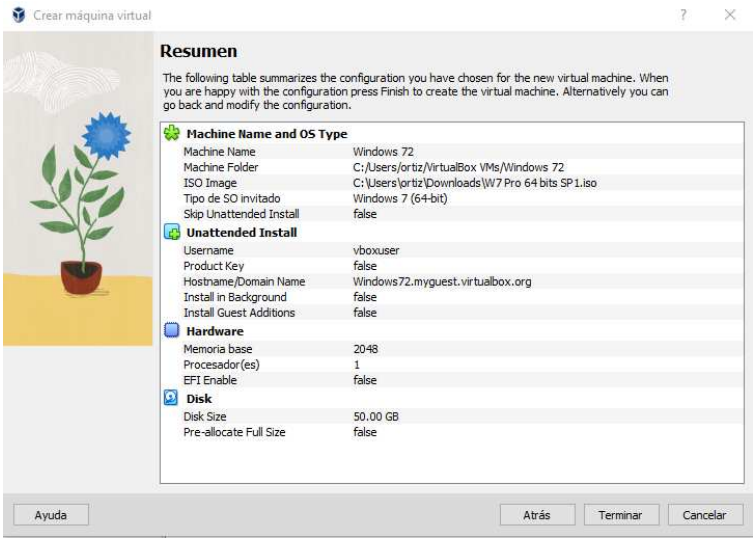


Nota. En la figura se puede observar una captura de la creación y configuraciones de una máquina virtual en Oracle Virtual Box. Elaboración propia, realizado con Oracle Virtual Box.

En la Figura 23 se muestra la ventana desplegada con las casillas ya llenas, estas se llenan de forma automática una vez se carga la imagen del sistema operativo descargado en la casilla ISO Image. Al momento de continuar con la instalación se puede seguir la guía de instalación que brinda Oracle VirtualBox. Al momento que se llegue a la asignación del espacio en disco se debe de contar con 50 GB de espacio libre, esto se debe al peso final que tiene ISE Design Suite al momento de instalarse, además que esta asignación de espacio dará libertad de instalación para añadir aplicaciones complementarias al sistema virtualizado. Al finalizar el proceso de creación desplegara un resumen similar al que se muestra en la Figura 24, con esto se ha creado el entorno virtual que permitirá una mayor facilidad en la instalación del software ISE.

Figura 24.

Finalización de creación de máquina virtual

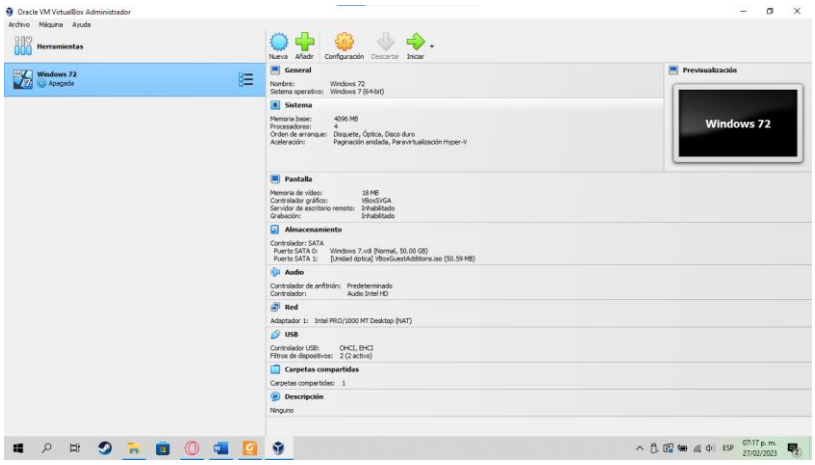


Nota. En la figura se puede observar una captura con el resumen de configuraciones de una máquina virtual en Oracle Virtual Box. Elaboración propia, realizado con Oracle Virtual Box.

Al momento que se haga click en Terminar aparecerá dentro de la casilla derecha de Oracle VirtualBox la máquina virtual, al momento que se seleccione iniciara el instalador de Windows 7, a partir de este punto la guía que da Microsoft para la instalación será suficiente para realizar la instalación. Una vez termine la instalación del nuevo sistema operativo se podrá proceder a la instalación de ISE Design Suite 14.7.

Figura 25.

Oracle VMVirtualBox con la máquina virtual Windows 7



Nota. En la figura se puede observar una captura con la máquina virtual creada en Oracle Virtual Box. Elaboración propia, realizado con Oracle Virtual Box.

3.1.3. Instalación de ISE Design Suite 14

La instalación de este *software* es igual a cualquier programa, las diferencias están en que se debe contar con una licencia para realizar la instalación. El programa es gratuito, pero AMD Xilinx por temas de seguridad solicita a cualquier usuario interesado en realiza la instalación una licencia en donde especifique que uso le dará al programa. La gestión de la licencia puede gestionarse por medio del tutorial que se encuentra dentro de la página de AMD Xilinx o se puede utilizar la licencia que fue gestionada para el presente trabajo de graduación.

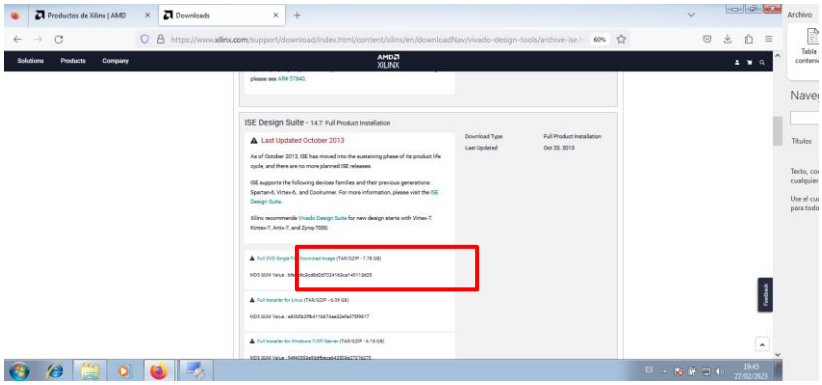
El *software* se puede descargar desde la página oficial de AMD Xilinx, se debe descargar la versión 14.7 y debe ser seleccionada la versión completa que posee un peso de 7.78 GB para que no exista problemas relacionados

con que una de las partes falle en el proceso de descarga, esto se puede ver más detallado en la Figura 26.

Una vez se descarga la versión completa se procede con la instalación siguiendo la guía de instalación paso a paso. En el proceso de instalación sigue normal las primeras tres pantallas del instalador, al momento de llegar al apartado donde el título indica Select Products to Install se debe instalar la opción que dice ISE Design Suite System Edition la cual tiene un peso equivalente a 20.4 GB. Para las siguientes pantallas la instalación sigue un curso normal como cualquier otro, el usuario debe considerar que la instalación en algunas ocasiones puede llegar a tardar hasta una hora en completarse, en todo este periodo se debe instalarán los controladores de algunas tarjetas y algunos programas de apoyo que servirán para el correcto funcionamiento de ISE Desing Suite.

Figura 26.

Programa por descargar

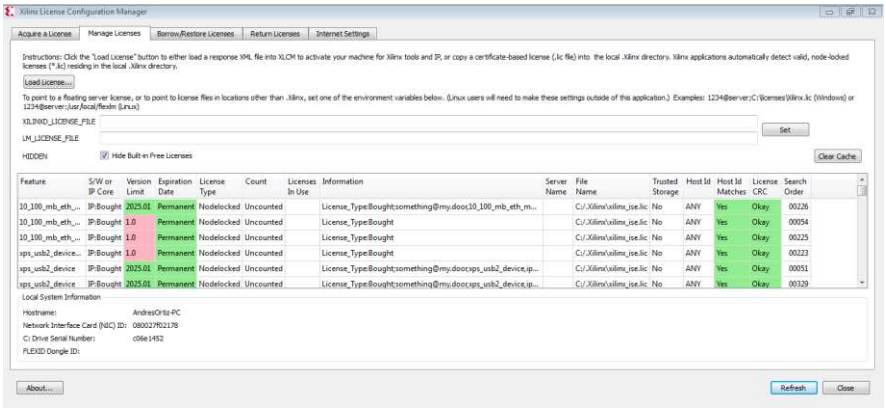


Nota. En la figura se puede observar una captura de la página AMD Xilinx con el instalador que debe ser descargado para instalarlo en la máquina virtual creada en Oracle Virtual Box. Elaboración propia, realizado con Firefox.

Una vez se finalice la instalación se debe cargar la licencia del programa, dicha licencia permitirá acceder a las simulaciones, el compilador, el generador de archivos. bit y demás funciones. Sin esta licencia ISE Desing Suite no funcionara en toda su capacidad. La pestaña en donde debe ser añadida la licencia se muestra en la Figura 27, usualmente esta pestaña debe de aparecer de forma automática una vez finaliza la instalación, en caso esto no haya ocurrido también se puede acceder a ella iniciando el programa ISE Desing Suite 14, ir a la barra de herramientas en Help y después en Manage License. En la pestaña de Xilinx License Configuration Manager solo se debe cargar la licencia utilizando el botón Load License y cargar el archivo. lic que se encuentra en el repositorio de este trabajo mencionado una paginas atrás. Se debe considerar que una vez se cargue la licencia esta no debe moverse a otra carpeta, ya que ISE Desing Suite no carga una copia de esta, sino que utiliza la ruta que se le dio la vez que se cargó. Con este proceso ya se tiene instalado y listo para usarse el *software* ISE Desing Suite.

Figura 27.

Carga de licencia en ISE Design Suite 14



Nota. En la figura se puede observar una captura de la carga de la licencia dentro del *software* ISE Design Suite 14. Elaboración propia, realizado con ISE Design Suite 14.

3.1.4. *Software* inicial FPGA Spartan 3A

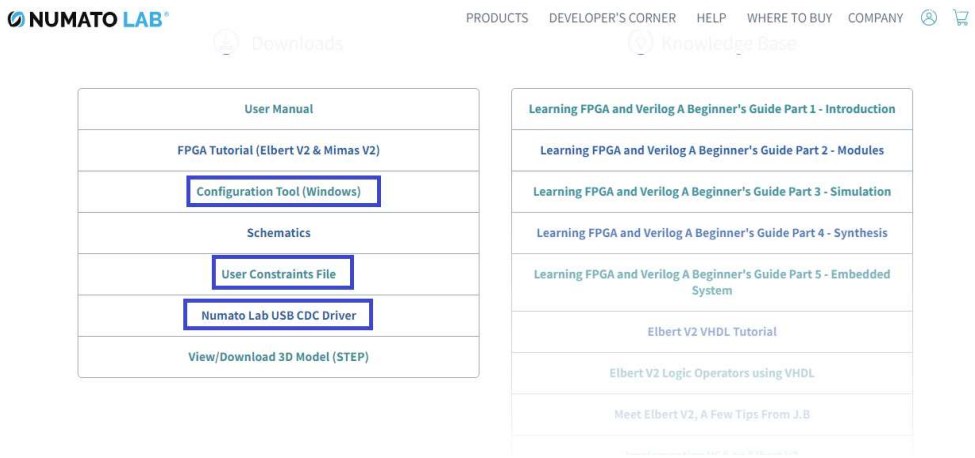
La Spartan 3A requiere ciertos programas, controladores y archivos para que sea funcional al momento de conectar la tarjeta a un computador.

Para que una computadora pueda identificar correctamente la FPGA debe poseer el controlador de dicha tarjeta. Este controlador puede ser descargado desde la página oficial de Numato. El controlador debe ser instalado desde el administrador de dispositivos al momento que se conecte la FPGA a la computadora.

En la página oficial se puede encontrar dos archivos necesarios para el funcionamiento de la FPGA, uno de ellos es el archivo UCF el cual contiene todos los puertos específicos de la Elbert V2 (dicho documento se puede encontrar también en el enlace mencionado en el capítulo anterior) y también puede ser descargado de dicha página la herramienta de configuración para Windows, esta herramienta será útil al momento de realizar la carga de los archivos .bit a la FPGA para que esta pueda ejecutar las instrucciones programadas.

Figura 28.

Software necesario de Numato Lab



Nota. En la figura se puede observar una captura de la página oficial de Numato Lab. Elaboración propia, realizado con Firefox.

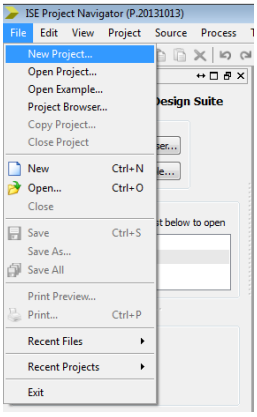
3.1.5. Primer programa en VHDL para la FPGA Spartan 3A

Una vez se ha logrado realizar la instalación de la máquina virtual y del programa ISE Desing Suite 14 se puede ejecutar la programación del primer programa en VHDL. El procedimiento que será mostrado lo siguen todos los proyectos sin excepción, por este motivo para este primer ejemplo se abordara a detalle cada uno de los pasos a ejecutar. Los pasos en la creación del programa son los siguientes:

- Inicializar el programa ISE Design Suite 14. Una vez cargue su pantalla de inicio se debe localizar en la barra de herramientas la pestaña File y después New Project. Esto se puede observar en la Figura 29.

Figura 29.

Creación de proyecto

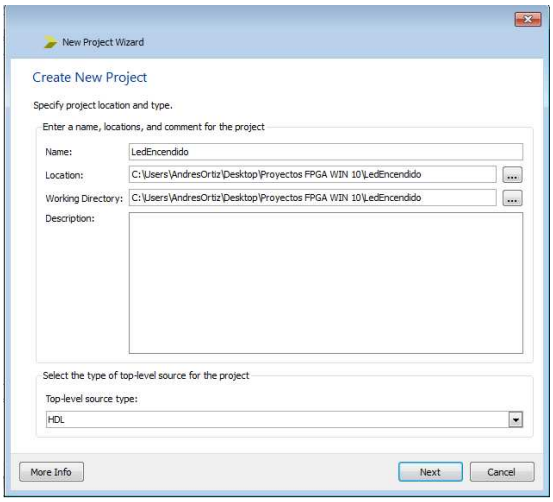


Nota. En la figura se puede observar una captura de la forma en que se debe crear un proyecto nuevo. Elaboración propia, realizado con ISE Design Suite14.

- Una vez se cumple el paso 1 debe aparecer una nueva ventana en donde se debe ingresar el nombre del nuevo proyecto y la carpeta en donde se desea guardar, para fines de este ejemplo se creará el proyecto bajo el nombre LedEncendido. Algo muy importante en este paso es que en la casilla inferior debe de seleccionarse el lenguaje que se desea utilizar, para todos los programas mostrados en este trabajo se utilizara VHDL, por lo tanto, debe seleccionarse HDL.

Figura 30.

Nombre y lenguaje del proyecto

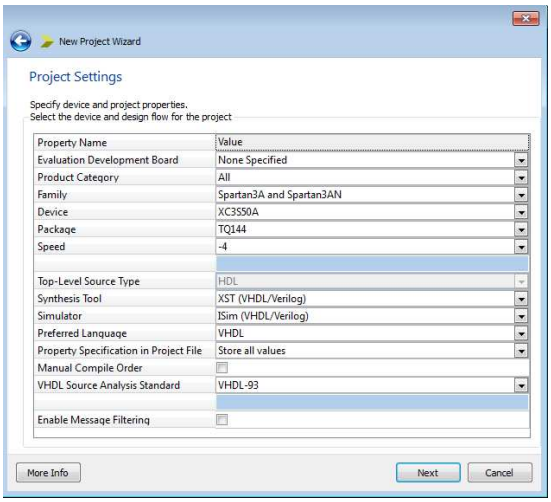


Nota. En la figura se puede observar una captura de la ventana del nombre y ubicación del proyecto nuevo. Elaboración propia, realizado con ISE Design Suite14.

- El tercer paso consiste en la configuración del entorno en el que el proyecto se utilizara, para este caso se debe seleccionar la tarjeta Spartan3A and Spartan3AN además de seleccionar el chip XC3S50A, para un ejemplo más práctico de todos los campos que se deben configurar para lograr el correcto funcionamiento en la FPGA se puede consultar la Figura 31. Seguido a esta pestaña se tiene un resumen de todo lo realizado del paso 1 al 3, si se realizó correctamente bastara con seleccionar Finish para terminar con el proceso.

Figura 31.

Configuración del proyecto

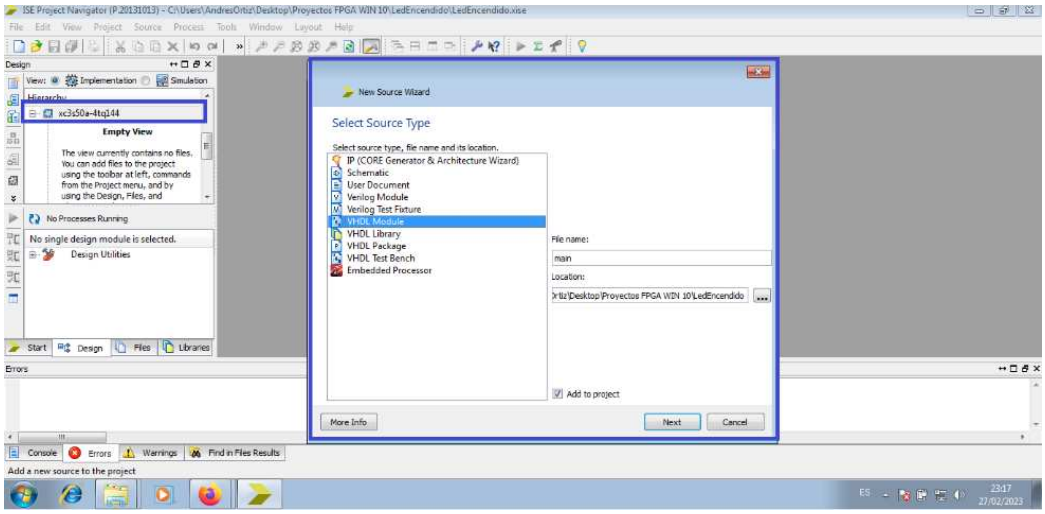


Nota. En la figura se puede observar una captura de la ventana de la configuración que debe tener el proyecto para funcionar en la tarjeta Elbert V2. Elaboración propia, realizado con ISE Design Suite14.

- Con los pasos anteriores se ha logrado crear un ambiente de trabajo para la FPGA, este paso consistirá en crear un módulo en VHDL que simplemente conecte una entrada con una salida, la entrada será asignada al SW1 de la FPGA y la salida será asignada al led D1. Para lograr la creación de dicho módulo se debe seleccionar con click derecho el icono con forma de chip que posee la descripción xc3s50a-4tq144, acto seguido se selecciona New Source lo cual desplegara una nueva ventana. En la ventana se seleccionará VHDL Module y se le pondrá el nombre de main.

Figura 32.

Creación del módulo principal

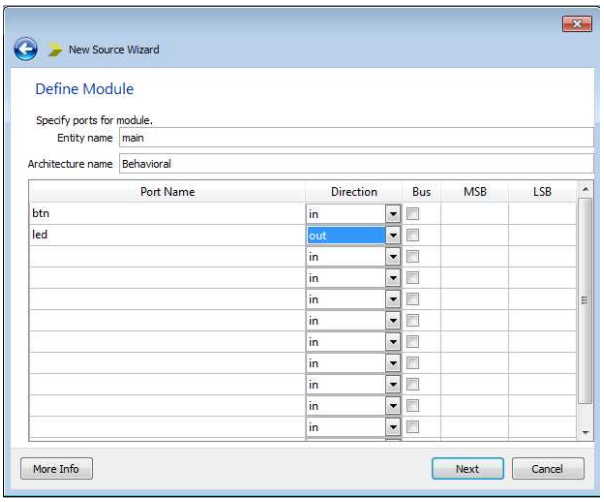


Nota. En la figura se puede observar una captura de la ventana de creación de un módulo principal en VHDL. Elaboración propia, realizado con ISE Design Suite14.

- Una vez se crea el módulo VHDL se debe indicar las entradas y salidas, este paso se puede obviar ya que siempre se pueden modificar las entradas y salidas después si no se conocen con exactitud, pero para el caso del ejemplo se sabe que solo se necesita una entrada que será llamada btn y una salida llamada led. Después de esto se podrá ver un resumen de lo realizado y se puede seleccionar Finish para terminar con este paso.

Figura 33.

Creación del módulo principal

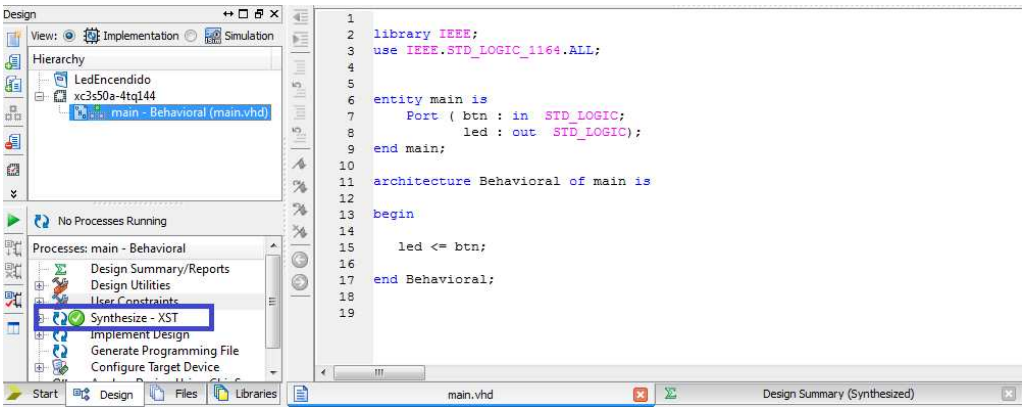


Nota. En la figura se puede observar una captura de la ventana de asignación de entradas y salidas del módulo. Elaboración propia, realizado con ISE Design Suite14.

- Una vez finalice el paso 5 automáticamente desplegara el entorno de trabajo. Dentro de este entorno presenta automáticamente la entidad y la arquitectura de dicha entidad. Dentro de la entidad ya están establecidos los puertos de entrada y salidas que fueron colocadas en el paso anterior. Lo que debe de realizarse ahora es la redacción del código, dicho código es solo asignar la entrada a la salida, en la Figura 34 se puede observar el código con mayor detalle. Una vez finalice la redacción debe de sintetizarse, para hacerlo se debe hacer doble click izquierdo sobre el símbolo de flechas azules que tiene la descripción de Synthesize – XST para comprobar la sintaxis del programa, cuando sea correcta la escritura mostrara un símbolo de palomita verde.

Figura 34.

Código y sintetización del módulo principal

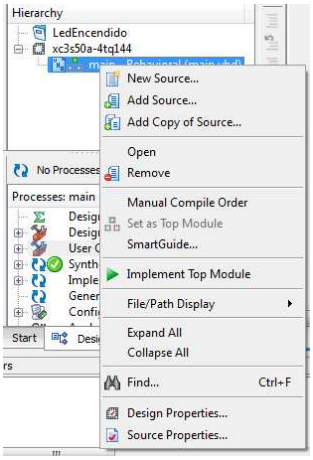


Nota. En la figura se puede observar una captura del módulo con un código de ejemplo implementado y una señalización del botón para sintetizar el proyecto. Elaboración propia, realizado con ISE Design Suite14.

- Una vez se ha escrito el programa y ha sintetizado se debe cargar el archivo UCF. Para cargar el archivo se debe seleccionar el archivo main y seleccionar Add Copy of Source y se deberá seleccionar el archivo UCF de la tarjeta Spartan 3A. Cuando se seleccione desplegara una ventana, no debe de ser modificado nada, solo hacer click en Ok. Esto añadirá un archivo al main que puede ser visualizado si se hace click en el símbolo + que se encuentra a su lado izquierdo. En casos de instanciación se debe añadir el archivo haciendo click en el símbolo del chip, pero esto se documentará de forma más detallada en un capítulo más adelante dentro de este trabajo.

Figura 35.

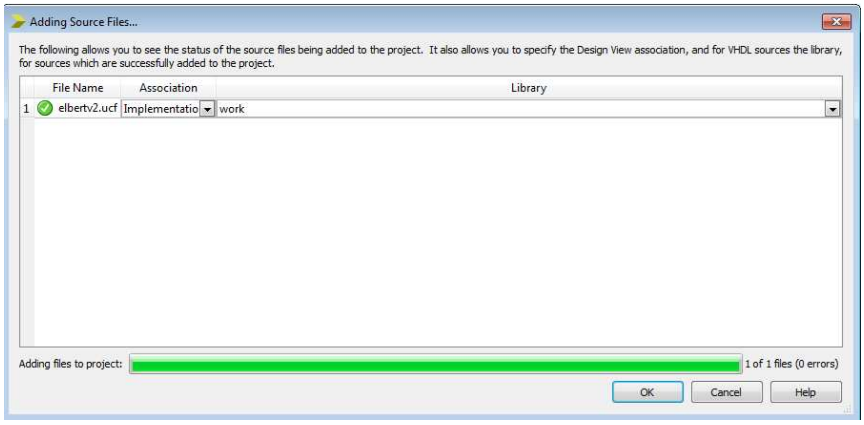
Añadir copia de un archivo al módulo principal



Nota. En la figura se puede observar una captura de la forma en que se debe agregar una copia del archivo. Elaboración propia, realizado con ISE Design Suite14.

Figura 36.

Ventana de carga del UCF

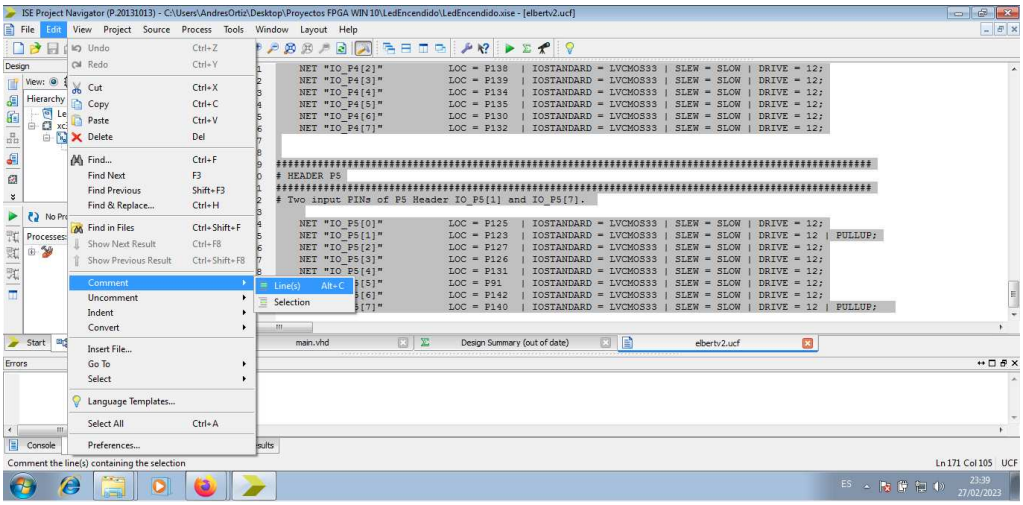


Nota. En la figura se puede observar una captura de la ventana con el archivo UCF cargado a ISE Design Suite 14. Elaboración propia, realizado con ISE Design Suite14.

- Se debe hacer click en el nuevo documento dentro de main para abrir el UCF, este documento deberá ser comentado por completo antes de ser utilizado ya que ISE Desing Suite tomará como activos todos los puertos de no hacerlo, para hacer esta tarea bastará con seleccionar todo el texto del documento UCF, dirigir el cursor a la barra de herramientas y seleccionar Edit/Comment/Selection.

Figura 37.

Comentar el archivo UCF



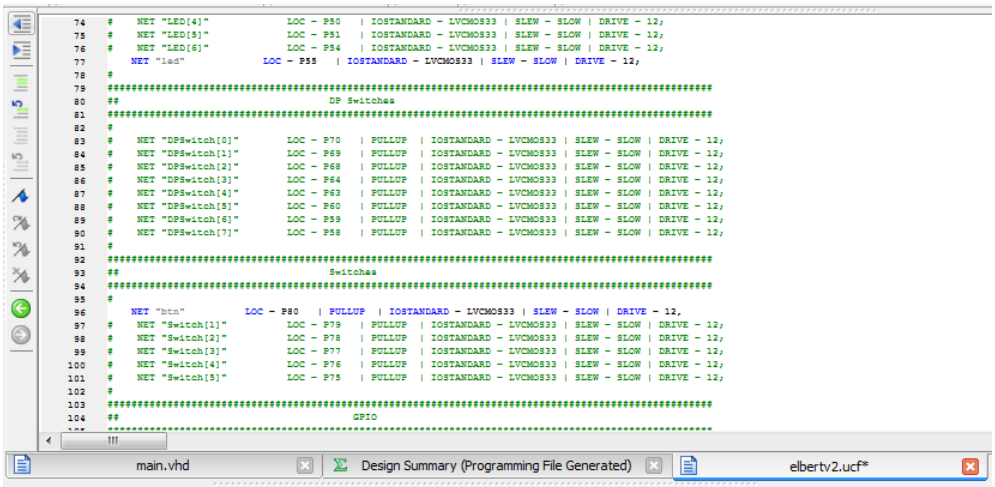
Nota. En la figura se puede observar una captura de la forma en que se puede comentar cualquier archivo dentro de ISE Design Suite14. Elaboración propia, realizado con ISE Design Suite14.

- Con el UCF comentado por completo se procede a asignar los puertos de la entidad de main, en este caso según se indicó en el enunciado del inicio se desea la asignación con btn a SW1 y led a D1.

- Para hacer esta asignación se debe encontrar los puertos en el UCF, D1 es el LED[7] y el SW1 es el Switch[0], se debe quitar el comentario eliminando el numeral y cambiar los nombres que se encuentran dentro de las comillas dobles con los nombres de los puertos de la entidad.

Figura 38.

Asignar los puertos al archivo UCF

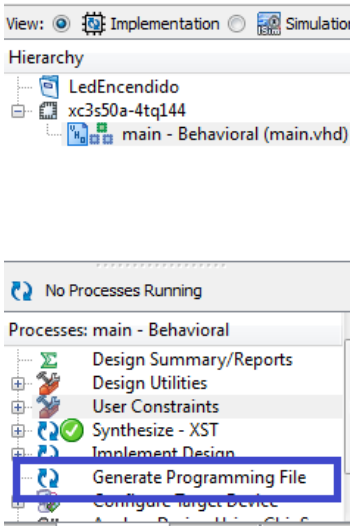


Nota. En la figura se puede observar una captura de la asignación de entradas y salidas al archivo UCF. Elaboración propia, realizado con ISE Design Suite14.

- Este paso es el que realizara el programa. bit que necesita la FPGA para funcionar, lo que se debe hacer el volver al archivo main de la pestaña Hierarchy para que ISE vuelva a desplegar el menú de sinterización, se guarda todo el proyecto y se hace doble click en el símbolo de las flechas azules que dice Generate Programming File. Esto ejecuta el compilador y realiza la preparación del archivo .bit con los puertos que fueron asignados en el UCF. Una vez finaliza este proceso se tendrá el programa listo para ser cargado.

Figura 39.

Generación del archivo .bit

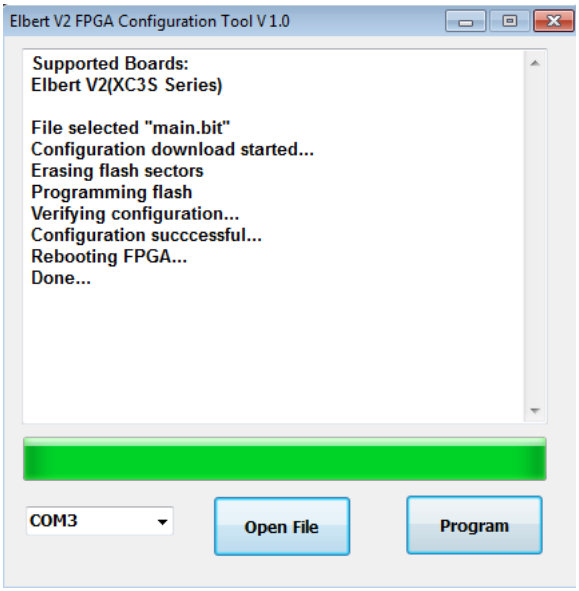


Nota. En la figura se puede observar la función que puede realizar la creación de un archivo bit. Elaboración propia, realizado con ISE Design Suite14.

- El paso final consiste en utilizar el programa de configuración Elbert V2 FPA Configuration Tool V 1.0 para cargar el archivo. bit creado en el paso anterior. Para encontrar el archivo bastará con dirigirse a la carpeta del proyecto y localizar el archivo main.bit, una vez cargado al programa se debe conectar la FPGA, seleccionar el puerto COM en la que se encuentra y hacer click en Program, cuando este paso finalice se tendrá el programa ejecutándose dentro de la FPGA. (Dentro del repositorio se podrá encontrar un video mostrando el comportamiento del programa).

Figura 40.

Carga del archivo .bit a la FPGA



Nota. En la figura se puede observar el programa que sirve para cargar al archivo bit a la FPGA. Elaboración propia, realizado con ISE Design Suite14.

3.2. Comando Process

En el capítulo 2 fue tratado este comando de forma parcial describiendo el proceso de escritura y el funcionamiento de los Process, en este capítulo se busca complementar los conceptos tratados utilizando ejemplos del funcionamiento de la instrucción.

Tal como fue descrito con anterioridad, el comando process se utiliza para crear circuitos más pequeños que funcionen siempre que exista cambios en la variable sensitiva que es descrita en los parámetros iniciales. Se debe tomar en cuenta que toda salida o entrada utilizada en los process de forma interna no funcionara afuera de ellos una vez han sido declarados, por

ejemplo, al momento de cargar un dato a una de las salidas de la entidad adentro del process a dicha salida no se le puede cargar otro valor afuera del process ya que VHDL marcara un error al usar la misma salida en circuitos ejecutados en forma paralela.

Las variables sensitivas más comunes para utilizar son en su mayoría a las que se les asigna el reloj de la FPGA o los botones push. Pueden llegar a existir ocasiones en las que los procesos no requieran uso de variables sensitivas, aunque para esos casos se debe considerar el no utilizar entradas ya que ISE solicitara que estas sean añadidas como parte de las listas sensitivas. En el ejemplo que será descrito a continuación solo servirá para entender el funcionamiento ya que este comando siempre se utilizara como complemento de otros comandos que serán descritos más adelante.

3.2.1. Ejemplo No.1: entender la función Process

Para este ejemplo se requiere un circuito que tenga una entrada y dos salidas, la entrada será asignada a uno de los Dip Switches de la tarjeta y las salidas serán asignadas a dos de los leds. Las variables serán nombradas así:

- Btn: Entrada (Hay que recordar que son PullUp)
- Led: Salida 1
- Led2: Salida 2

Para poder observar el uso de la función process con una variable sensitiva se utilizará la variable Btn en el ejemplo. El objetivo es modificar las salidas Led y Led2, la forma de hacerlo será distintas para ambas. Las instrucciones son las siguientes:

- Para Led la asignación de valor cambiará por medio de una variable auxiliar que almacenará el valor negado de Btn y dicha asignación no estará dentro de Process.

- Para Led2 la asignación será el valor de Btn de forma directa y esta será asignada dentro de Process.

Figura 41.

Programa de Ejemplo No.1: entender la función Process

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  use IEEE.std_logic_unsigned.all;
5
6  entity main is
7      Port ( Btn : in  STD_LOGIC;
8             Led : out  STD_LOGIC;
9             Led2 : out  STD_LOGIC);
10 end main;
11 architecture Behavioral of main is
12 signal aux : std_logic:= '1';
13 begin
14    Led <= aux;
15    process(Btn)
16    begin
17       Led2 <= Btn;
18       aux <= not(Btn);
19    end process;
20 end Behavioral;
```

Nota. En la figura se puede observar el código del ejemplo que ilustra la función process. Elaboración propia, realizado con ISE Design Suite14.

El funcionamiento del código mostrado en la Figura 41 es exactamente cómo fue descrito en cada una de las instrucciones para Led y Led2, se puede observar que las salidas cambian únicamente cuando se acciona el botón. Se

debe tomar en cuenta que VHDL ejecuta el process justo al momento de conectar la tarjeta, ya que el estado de Btn cambia en su estado inicial y por este motivo las salidas se encuentran inicializadas.

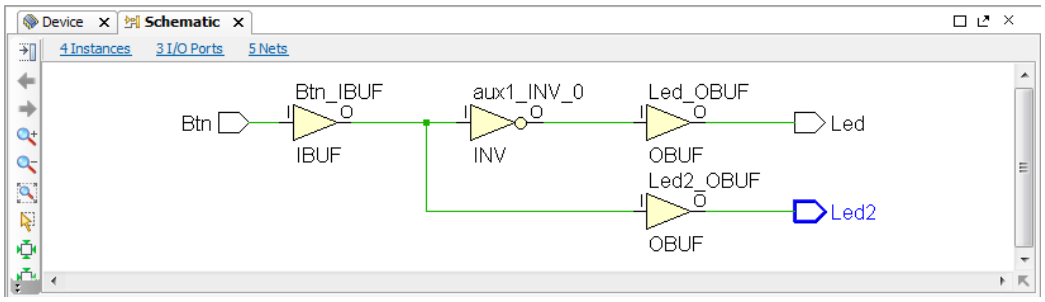
Con el ejemplo se puede entender el comando al ver que este puede afectar salidas externas a él siempre que se utilice señales auxiliares para enviar estos valores, en este caso se utilizó la señal aux como señal auxiliar para llevar los datos fuera del proceso, es por esto que el valor de Led puede cambiar.

3.2.1.1. Análisis esquemático

ISE Design Suite puede crear esquemáticos a partir de los programas en VHDL creados. Para realizarlo se utiliza la herramienta PlanAhead, esta herramienta puede encontrarse en la opción de Tools/PlanAhead/User Constrains/Floorplan Area/IO/Logic (PlanAhead).

Figura 42.

Esquemático de Ejemplo No.1: entender la función Process



Nota. En la figura se puede observar el esquemático generado a partir del código del ejemplo que ilustra la función process. Elaboración propia, realizado con ISE Design Suite14.

En la Figura 42 se puede ver cómo es la construcción del circuito, junto con las entradas, las instancias y los enlaces que posee el circuito. Para el análisis del esquemático se dividirá en tres partes, las cuales se detallan a continuación:

- El comando process para VHDL en este ejemplo no es más que un circuito seguidor IBUF, este es la representación del process y de su variable sensitiva Btn. Una vez VHDL pasa este proceso donde registra la actividad de Btn se ejecutan las tareas de forma paralela tal como lo indica el esquemático.

- La segunda instancia por evaluar es la que da el valor de salida de Led. En el programa la asignación del valor auxiliar a la salida se encuentra fuera de process, pero en los esquemáticos está al mismo nivel que la salida de Led2, esto se debe a que al utilizar una señal auxiliar lo único que se hace es crear un camino alternativo de salida, ese camino es la compuerta negada INV que conecta con la compuerta de salida seguidora de process OBUF hasta entregar el valor a la salida Led. Con este ejemplo se puede observar cómo VHDL gestiona las instrucciones, a pesar de poseer el valor ya negado y podría entregarlo directo a la salida Led, este valor pasa por una compuerta de salida, esta compuerta de salida es en realidad el final del process. Resumiendo, VHDL opera todas las instrucciones de un proceso dentro del mismo, para que las salidas solo tengan el valor para mostrar.

- Al evaluar la instancia que envía el valor a asignar a la salida Led2 se puede observar que es similar a la salida de Led, con la diferencia que conecta la instancia sensitiva de Btn directamente con la salida del

process OBUF, dicha salida la asigna a Led2 para mostrar el valor del cambio del botón.

El esquemático mostrado puede explicar el modo en que VHDL puede interpretar el código que se ha escrito, al analizarlo ha sido evidente que no sigue el orden secuencial con el que se ha escrito, sino que realiza una ejecución diferente aislando la entrada y las dos salidas teniendo únicamente la ejecución del process cuyo inicio lo marca seguidor IBUF y el final lo marca el seguidor OBUF. En resumen, la instrucción process es capaz de tomar los puertos y manejar sus datos con cualquier instrucción ya que solo asignara los valores a los puertos una vez tenga lista la información que entregara.

3.3. Comandos condicionales

Los comandos condicionales If y Case son de mucha importancia para el complemento del comando process. Estos comandos condicionales fueron descritos a nivel conceptual en el capítulo dos. En este capítulo será explicada su sintaxis y la forma de utilizar la instrucción process para que funcione correctamente. En los ejemplos que se evalúan a continuación se puede ver el uso de las sentencias condicionales, estas van desde programas comparadores hasta el uso del reloj interno para la ejecución de tareas.

3.3.1. Ejemplo No.1. Funcionamiento y sintaxis de una sentencia If

Este ejemplo busca explicar la sentencia if en su forma más básica, para realizarlo se propone encender una salida en función de la entrada que tiene el circuito, el programa es similar a conectar la entrada directamente con

la salida, pero se recomienda este ejemplo con fin explicativo. Los puertos que serán utilizados para el programa se detallan a continuación:

- bnt: este es el botón push que será la entrada.
- led: este será el led que será la salida.

El objetivo del circuito es analizar que el valor de la entrada btn, si la entrada btn es igual al valor 1 (que para el caso del push seria cuando no está siendo pulsado) la salida led tendrá un valor 0 y no encenderá. Al momento que btn es pulsado el valor cambia a 0, el led tendrá un valor 1 y encenderá. Este programa es descrito en la Figura 43. btn es pulsado el valor cambia a 0, el led tendrá un valor 1 y encenderá. Este programa es descrito en la Figura 43.

Figura 43.

Programa de Ejemplo No.1: Sintaxis de una sentencia If

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  use IEEE.std_logic_unsigned.all;
5  entity main is
6      Port ( btn : in  STD_LOGIC;
7             led : out  STD_LOGIC);
8  end main;
9  architecture Behavioral of main is
10 begin
11    ConIf: process (btn)
12    begin
13       if (btn = '1') then
14          led <= '0';
15       else
16          led <= '1';
17       end if;
18    end process ConIf;
19 end Behavioral;
```

Nota. En la figura se puede observar el código del ejemplo que ilustra la función if. Elaboración propia, realizado con ISE Design Suite14.

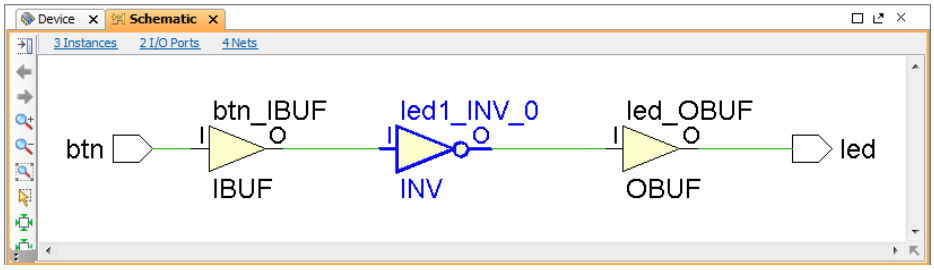
En la sintaxis del circuito descrito se puede notar que este funciona dentro de un process, esto se debe a que al ser btn una señal que cambia de valor y al estar siendo comparada con una sentencia condicional es necesario que funcione dentro del process para esa actualización constante de valores. El process posee un nombre a diferencia de los ejemplos anteriores, este nombre puede escogerse con total libertad siempre y cuando no invoque una palabra reservada de VHDL.

3.3.1.1. Análisis esquemático

Al utilizar la herramienta PlanAhead para el código planteado en la Figura 44 se puede observar que sigue el mismo patrón visto en el ejemplo de uso de process teniendo los seguidores IBUF y OBUF, además que no existe como tal un circuito definido para el if, únicamente es un circuito negado que toma la entrada y envía el resultado a la salida. En los próximos ejemplos se podrá observar cómo aumenta la complejidad de estos diagramas, pero siempre que se utilice un process existirán los mismos circuitos seguidores de inicio y final.

Figura 44.

Esquemático de Ejemplo No.1: funcionamiento de una sentencia If



Nota. En la figura se puede observar un diagrama esquemático del ejemplo que ilustra la función if. Elaboración propia, realizado con ISE Design Suite14.

3.3.2. Ejemplo No.2. Funcionamiento del reloj y uso de retardos en VHDL

Una vez se ha visto el uso básico de la instrucción process y la instrucción if se puede realizar procesos un poco más complejos. Para este ejemplo se realizará un ejemplo de retardos para la Elbert V2. Se debe considerar que estos retardos no pausaran el reloj a la espera de datos, la forma de trabajarlos es un poco diferente a un microcontrolador y se parece más a los lenguajes de programación primitivos donde se realizaban ciclos para hacer retardos. Para realizarlo se debe retomar lo que fue mencionado en el capítulo dos con el funcionamiento del reloj.

El reloj trabaja a una frecuencia de 12 MHz, esto significa que en un segundo ocurren 12,000,000 ciclos, este número es muy importante ya que para lograr realizar retardos se debe conocer la cantidad de cambios o ciclos de reloj que ocurren en un segundo. Teniendo el conocimiento de cuantos ciclos realiza el reloj por segundo se puede utilizar una estructura que realice una cuenta cada vez que el reloj cambie de estado. Se hace evidente con la descripción anterior que para medir los cambios del reloj se necesita un process con la señal del reloj como variable sensitiva para que se ejecute cada vez que cambie de estado. Además de la instrucción process se debe realizar la cuenta de los ciclos que han transcurrido, para realizarlo se debe medir los flancos ascendentes o descendientes de la señal de reloj, para realizar esta acción VHDL posee ciertas palabras reservadas las cuales son:

- rising_edge(Variable)
- falling_edge(Variable)

Estas palabras reservadas se deben de colocar como condición lógica en un if para que este pueda realizar los conteos de los ciclos de reloj y así poder realizar retardos. El código se presenta en la Figura 45.

Figura 45.

Ejemplo No.2: Funcionamiento del reloj y uso de retardos en VHDL

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  use IEEE.std_logic_unsigned.all;
5  entity main is
6     Generic(ciclos : Natural:= 12000000); -- Esta es la constante de los ciclos de reloj.
7     Port (  clk : in  STD_LOGIC;
8             led : out  STD_LOGIC);
9  end main;
10 architecture Behavioral of main is
11 signal cont : Natural:=0; --Esto es para el contador.
12 signal aux : std_logic:='1'; --Esta señal es una auxiliar para enviar datos a la salida.
13 begin
14    process(clk)
15       begin
16          if (rising_edge(clk)) then --Condiciòn para el conteo de ciclos.
17             if (cont = ciclos) then -- Evaluaciòn del contador
18                cont <= 0;
19                aux <= not(aux);
20             else
21                cont <= cont + 1;
22             end if;
23          end if;
24    end process;
25    led <= aux; -- Asignaciòn del valor de salida.
26 end Behavioral;
```

Nota. En la figura se puede observar el código de ejemplo que ilustra una función de un retardo de 1 segundo. Elaboración propia, realizado con ISE Design Suite14.

En el código resaltan elementos nuevos que no se mencionaron antes. Se crea un valor constante con la cantidad de ciclos del reloj, esto se hace para poder modificar fácilmente los segundos de retardo del programa, para variarlos simplemente se debe modificar la condición de la evaluación del contador multiplicando la constante ciclos con un valor N que sean los segundos que se desean (Recordar que los números naturales poseen limite dentro de VHDL). El retardo funciona gracias a las evaluaciones del segundo if, esta condición realiza el conteo que permite a VHDL realizar varias

sumatorias hasta que tarde 1 segundo exacto en cambiar el estado de la salida.

Figura 46.

Señal de reloj dentro del UCF

```
##++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#
## This file is a .ucf for ElbertV2 Development Board                                                           #
## To use it in your project :                                                                                  #
## * Remove or comment the lines corresponding to unused pins in the project                                    #
## * Rename the used signals according to the your project                                                      #
##++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#
#
##**************************************************************************************************************
##                                          UCF for ElbertV2 Development Board
##**************************************************************************************************************
#CONFIG VCCAUX = "3.3" ;
#
# # Clock 12 MHz
  NET "clk"                 LOC = P129  | IOSTANDARD = LVCMOS33 | PERIOD = 12MHz;
#
##############################################################################################################
##                                          VGA
##############################################################################################################
#
#    NET "HSync"             LOC = P93   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "VSync"             LOC = P92   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "Blue[2]"           LOC = P98   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "Blue[1]"           LOC = P96   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "Green[2]"          LOC = P102  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "Green[1]"          LOC = P101  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "Green[0]"          LOC = P99   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "Red[2]"            LOC = P105  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "Red[1]"            LOC = P104  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "Red[0]"            LOC = P103  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#
```

Nota. En la figura se puede observar la forma en asignar la entrada del reloj al archivo UCF. Elaboración propia, realizado con ISE Design Suite14.

La asignación de la entrada de reloj debe de realizarse tal como lo muestra la Figura 46, ya que dentro del UCF este puerto es el que tiene asignado la Elbert V2 para tomar lectura del estado de la señal del reloj.

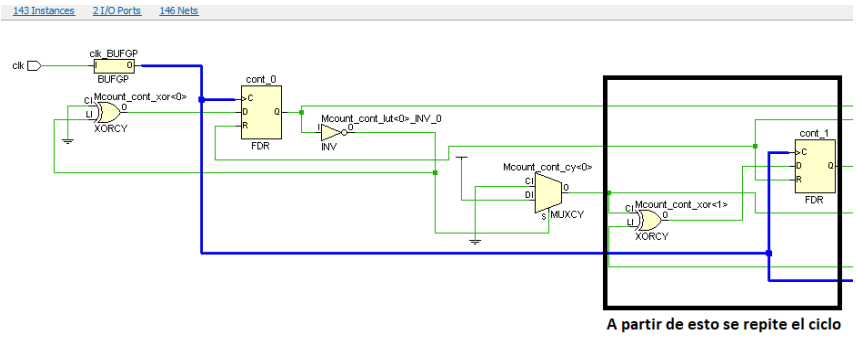
El uso del reloj dentro de VHDL es sencillo, la diferencia principal comparado con un microcontrolador radica en que no se ejecutan pausas, lo que se ejecuta son condiciones que simulan ciclos y hasta que no se cumpla cierta condición no realizara ni ejecutara ninguna instrucción programada dentro de la condición.

3.3.2.1. Análisis esquemático

El esquemático de este ejemplo es extremadamente complejo de analizar, basta con mencionar que posee una cantidad de 143 instancias en cascada las cuales se repiten con la misma información. Para poder explicar este esquemático de una forma sencilla se utilizará solamente un pequeño segmento. El esquemático toma la señal de reloj e inicia el proceso con el circuito BUFGP. Conecta el reloj a una cascada de Flip Flop que tienen como fin llegar hasta un multiplexor que permite tomar todas las salidas de los FIip Flops en cascada. La función que realizan los Flip Flops es la de un sumador binario y por la enorme cantidad que se debe sumar es que existen hasta 30 contadores que se unen a distintos multiplexores. Los Flip Flops poseen como entrada una compuerta XOR, esta compuerta es la condición que mide los flancos ascendentes del reloj y solo da la señal si existió un flanco ascendente. Esta descripción VHDL la repite varias veces, de nuevo se hace énfasis en que es por la cantidad de ciclos que debe de contar. El extracto del esquemático puede observarse en la Figura 47.

Figura 47.

Ejemplo No.2: funcionamiento del reloj y uso de retardos en VHDL



Nota. En la figura se puede observar un diagrama esquemático del ejemplo del retardo, se muestra solo un segmento ya que son 143 instancias las que posee dicho módulo. Elaboración propia, realizado con ISE Design Suite 14.

3.3.3. Ejemplo No.3. Funcionamiento y sintaxis de una sentencia Case

El uso de la sentencia Case en VHDL se realiza en la mayoría de los casos para realizar operaciones donde una misma variable o señal pueda tener múltiples estados. El ejemplo más sencillo consiste en tomar un vector de una cierta cantidad de bits y revisar las entradas que este vector tenga para ejecutar distintas tareas.

Para el ejemplo que será presentado se realizará un circuito que sea capaz de transformar una entrada que represente el sistema decimal y convierta el numero entrante en binario. La entrada decimal será asignada a los switches de la Elbert V2 y la salida binaria serán los leds. La forma de hacer la conversión será validando el vector de entrada asignado a los switches, será un vector de N=3 para representar los números del 0 al 3 en binario, al ser el

valor decimal más alto el numero 3 esto indica que el vector de salida debe ser de N=2 para poder representar todos los valores. Las entradas y salidas a utilizar se detallan así:

- Clk: entrada de reloj para ejecución constante de lecturas.
- Num[2]: entrada de los switches que representaran la entrada numérica decimal.
- Sal[1]: salida de los leds que representaran la salida numérica binaria.

Figura 48.

Código de Ejemplo No.3: funcionamiento y sintaxis de una sentencia Case

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  use IEEE.std_logic_unsigned.all;
5  entity main is
6      Port ( Clk : in  STD_LOGIC;
7             Num : in  STD_LOGIC_VECTOR (2 downto 0);
8             Sal : out  STD_LOGIC_VECTOR (1 downto 0));
9  end main;
10 architecture Behavioral of main is
11 signal aux: std_logic_vector (1 downto 0);
12 begin
13    reloj: process (Clk, Num)
14    begin
15       if (rising_edge(Clk)) then
16          case (Num) is
17             when "100" =>
18                aux <= "01";
19             when "010" =>
20                aux <= "10";
21             when "001" =>
22                aux <= "11";
23             when others =>
24                aux <= "00";
25          end case;
26          Sal <= aux;
27       end if;
28    end process reloj;
29 end Behavioral;
```

Nota. En la figura se puede observar el código del ejemplo que muestra el uso de la sentencia case. Elaboración propia, realizado con ISE Design Suite 14.

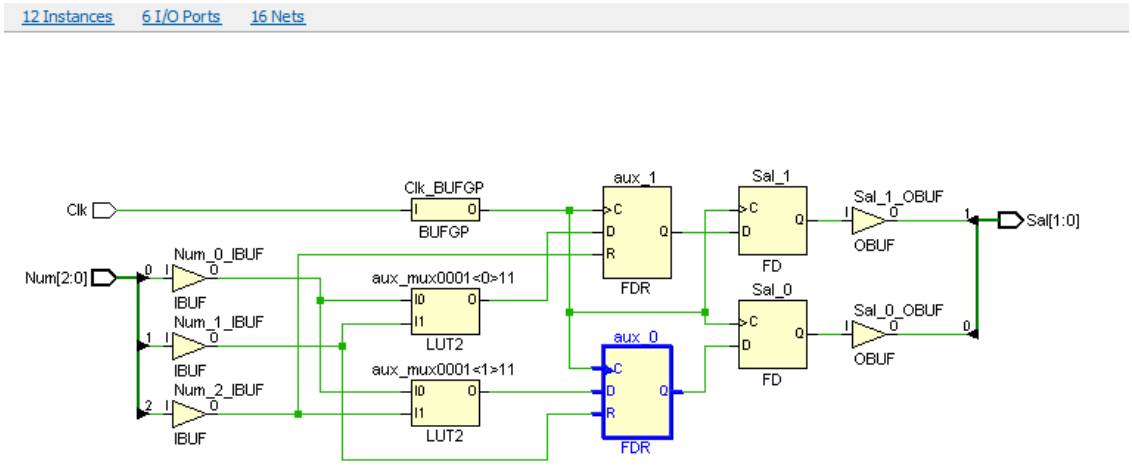
Al momento de cargar el código anterior a la FPGA se tiene el conversor de decimal a binario. La forma de funcionar del código es únicamente guiándose con los switches activos individuales, en la tarjeta FPGA marcan estos switches como 1,2 y 3, solo debe de levantarse uno de los tres para que se tenga una salida, de lo contrario no ocurrirá nada en el programa.

3.3.3.1. Análisis esquemático

El esquemático que se crea a partir de las instrucciones case es muy ordenado, consiste en una entrada de reloj y una evaluación de las tres entradas. El reloj sirve para sincronizar los Flip-Flops que sirven para retener los valores que van ingresando. La señal que se conecta a las entradas de los Flip-Flops provienen de multiplexores que realizan las comparaciones de la entrada, son dos ya que la salida contiene también dos bits. Las salidas que han mantenido los Flip-Flops son llevadas a otro Flip-Flop más sencillo para entregar las salidas del vector de bits que representan los números binarios, esto se hace de esta manera para estabilidad del circuito.

Figura 49.

Ejemplo No.3: funcionamiento y sintaxis de una sentencia Case



Nota. En la figura se puede observar el circuito esquemático el código del ejemplo que muestra el uso de la sentencia case. Elaboración propia, realizado con ISE Design Suite 14.

La sentencia CASE es sencilla de explicar con los circuitos, todas las decisiones siempre serán tomadas utilizando multiplexores y Flip-Flops que retengan sus salidas. Para el caso presentado al tener 4 decisiones se ha utilizado 12 instancias de las cuales 2 son multiplexores, 4 Flip-Flops y el resto son seguidores.

3.3.4. Ejemplo No.4. Comparación de If y Case

El interés de este ejemplo es demostrar las principales diferencias a nivel esquemático que tiene un If y un Case realizando la misma instrucción. Para este ejemplo no se hará uso del reloj para que solo interactúen las señales de interés. Las señales serán una entrada y tres salidas. La entrada consiste en un vector de 2 bits los cuales serán asignados a los botones push

de la FPGA y la salida será un vector también de 2 bits que conecte a los leds que se encuentran en la FPGA. El objetivo es sumar señales, si solo uno de los botones (sin importar cuál sea) es pulsado solo debe encenderse un led, si se pulsan dos botones deben encender los dos leds. La forma en que se asignaran son las siguientes:

- Btn[1] : entrada de los botones push, vector de 2 bits.
- Led[1]: salida de los les, vector de 2 bits.

Figura 50.

Código de Ejemplo No.4: comparación de If y Case, programa If

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  use IEEE.std_logic_unsigned.all;
5  entity main is
6      Port ( Btn : in  STD_LOGIC_VECTOR (1 downto 0);
7             Led : out  STD_LOGIC_VECTOR (1 downto 0));
8  end main;
9  architecture Behavioral of main is
10 begin
11    sif: process (Btn)
12    begin
13       if ((Btn = "01") or (Btn = "10")) then
14          Led <= "01";
15       elsif (Btn = "00") then
16          Led <= "11";
17       else
18          Led <= "00";
19       end if;
20    end process sif;
21 end Behavioral;
```

Nota. En la figura se puede observar el circuito esquemático el código del ejemplo que muestra el uso de la sentencia case. Elaboración propia, realizado con ISE Design Suite 14.

En la Figura 50 se puede observar el código en VHDL que ejecuta la acción haciendo uso de la sentencia if. Consiste simplemente en dos comparaciones lógicas que evalúa el valor que está presente en la entrada, la primera condición consiste en evaluar si al menos uno de los dos botones es pulsado utilizando or y la segunda condición evalúa si ambos botones han sido pulsados. No se implementó una señal auxiliar para evaluar el esquemático sin ningún dato extra.

En la Figura 51 se puede observar el mismo programa utilizando la sentencia case. Una de las principales diferencias es que en el Case no se puede agrupar o unir las condicionales con conectores lógicos como el or sino que se deben evaluar todos los casos posibles en donde cumpla la condición. Esto hace que sea más largo que el uso del if, pero será más efectivo si se llega a requerir distintas evaluaciones de varios condicionales.

Figura 51.

Código de Ejemplo No.4: comparación de If y Case, programa Case

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  use IEEE.std_logic_unsigned.all;
5  entity main is
6      Port ( Btn : in  STD_LOGIC_VECTOR (1 downto 0);
7             Led : out  STD_LOGIC_VECTOR (1 downto 0));
8  end main;
9  architecture Behavioral of main is
10 begin
11    scase: process (Btn)
12    begin
13       case (Btn) is
14          when "00" =>
15             Led <= "11";
16          when "01" =>
17             Led <= "01";
18          when "10" =>
19             Led <= "01";
20          when "11" =>
21             Led <= "00";
22          when others =>
23             Led <= "00";
24       end case;
25    end process scase;
26 end Behavioral;
```

Nota. En la figura se puede observar el circuito esquemático el código del ejemplo que muestra el uso de la sentencia case. Elaboración propia, realizado con ISE Design Suite 14.

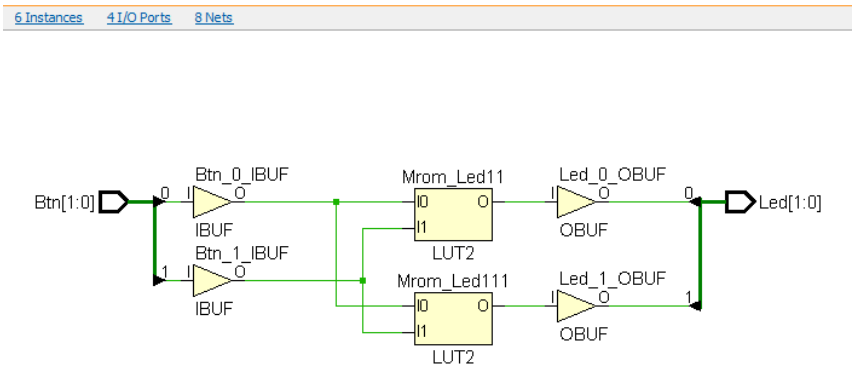
3.3.4.1. Análisis esquemático

En el análisis esquemático de este tipo de circuito son idénticos ambos circuitos. PlanAhead al momento de realizar la representación de las instrucciones muestra el mismo resultado, ambos circuitos inician con un seguidor de los valores de entrada, inicio del process, y evalúan dicha entrada en un circuito Mask Rom (Mrom) que es un circuito que contiene dentro la

programación de las condicionales que se han escrito. La salida de los Mrom conecta por medio de otro seguidor, el que cierra el process), para conectar con la salida del circuito. Esto hace evidente que el uso de case o if para VHDL es semejante, al ejecutar una instrucción a nivel de VHDL es igual.

Figura 52.

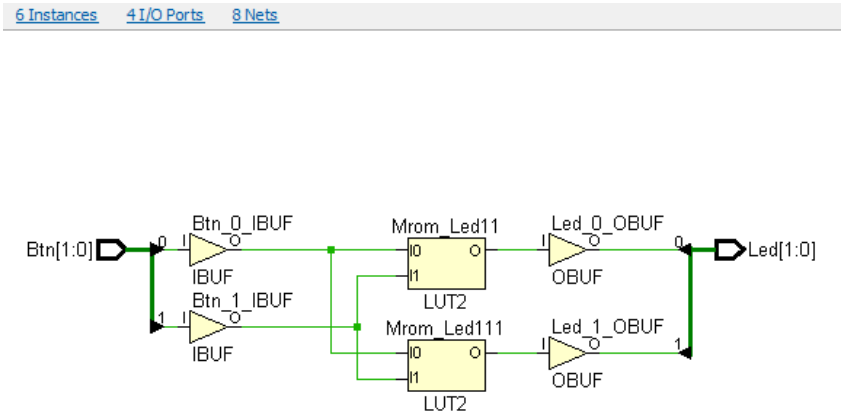
Esquemático programa If Ejemplo No.4



Nota. En la figura se puede observar el circuito esquemático el código del ejemplo que muestra el programa de la sentencia if. Elaboración propia, realizado con ISE Design Suite 14.

Figura 53.

Esquemático programa Case Ejemplo No.4



Nota. En la figura se puede observar el circuito esquemático el código del ejemplo que muestra el programa de la sentencia case. Elaboración propia, realizado con ISE Design Suite 14.

3.3.4.2. Conclusión If vs Case

Al evaluar los esquemáticos se puede notar que VHDL logra traducir el código a un mismo circuito funcional a pesar de utilizar diferentes instrucciones, incluso al evaluar el peso de los archivos. bit (comprobable en el enlace compartido) es igual, no existen cambios. La respuesta al cuestionamiento de que utilizar dependerá solamente de las necesidades del programador. Al proponer un ejemplo ficticio donde un programador necesite una aplicación que requiere solamente una comparación sencilla se puede utilizar if por la simpleza y lo compacta de la instrucción, en cambio sí se propone múltiples estados es más sencillo el uso de un case por el orden que lleva, al final VHDL siempre realizara la traducción y el funcionamiento será el mismo. En la tabla que se muestra a continuación se puede ver una lista de las aplicaciones y usos sugeridos para cada tipo de instrucciones:

Tabla 1.

Comparativa If vs Case

If	Case
Retardos.	Máquinas de Estado.
Decisiones simples.	Decisiones con vectores (múltiples valores).
Decisiones en modo cascada.	Decisiones similares a multiplexor.

Nota. Se exponen los posibles usos que tiene cada una de las sentencias. Elaboración propia, realizado con Excel.

3.3.5. Máquinas de estado en VHDL

Una máquina de estados es aquel circuito secuencial que permite la representación de un sistema por medio de sus entradas, salidas y estados. Dentro de las máquinas de estados existe un concepto llamado Estado Actual que se puede interpretar como el estado inmediato del circuito, este puede ser el estado de una conexión, de una entrada o una salida.

Existen dos tipos de modelos de máquinas de estado, los cuales son:

- Modelo Mealy: La salida se encuentra en función del estado de entrada como del estado actual.

- Modelo Moore: La salida solo se encuentra en función del estado actual.

Muchos circuitos secuenciales son máquinas de estado, pero dentro de VHDL existe una forma de crear una máquina de estado por medio de palabras reservadas, para ilustrar este ejemplo es necesario primero entender la

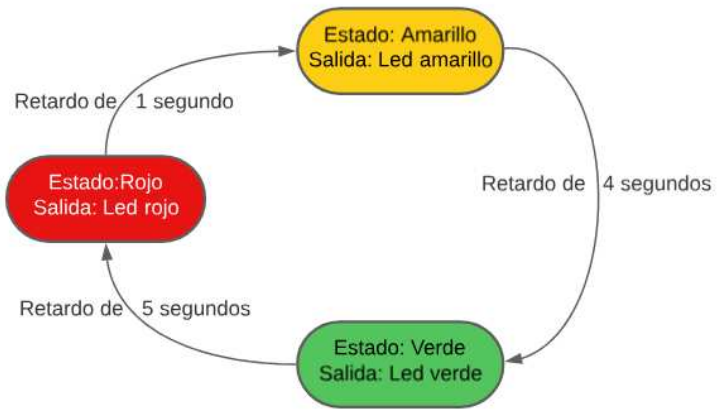
máquina de estados e ilustrar su grafico de transiciones para poder explicarlo de mejor forma en VHDL.

3.3.5.1. Máquina de estado Semáforo automático

Un semáforo es una máquina de estados de las más sencillas y fáciles de entender. Para este caso no se tendrán entradas ya que será un semáforo automático. Las salidas del semáforo son tres luces de colores rojo, amarillo y verde, estas salidas serán representadas con leds. Los estados son tres, estos serán relacionados con las salidas antes mencionadas. El diagrama de estados del semáforo que será utilizado para el ejemplo se puede observar en la Figura 54.

Figura 54.

Diagrama de estados de la máquina semáforo



Nota. En la figura se puede observar el diagrama de estados de la máquina semáforo. Elaboración propia, realizado con Lucidchart.

Se puede observar a simple vista que es una máquina de estados modelo Moore, cuyo estado inicial es el rojo, este activa la salida del led rojo y crea un retardo de 5 segundos antes de cambiar a otro estado, al momento del cambio al estado verde la salida cambia y crea otro retardo de 4 segundos para mantener esta salida. Finalmente, al terminar el retardo del led verde realiza el cambio al estado amarillo, cambiando la salida y creando un retardo de solo 1 segundo para retornar al estado inicial.

Esta misma máquina de estados ha sido redactada en VHDL. Para la creación de esta máquina de estados se sigue el mismo proceso que con el resto de los programas, cambia un detalle al momento de añadir las señales auxiliares ya que es en este punto donde se realiza la creación de la máquina de estados que contendrá todos los estados del semáforo. En la Figura 55 se puede observar un fragmento del código, la máquina de estados se crea en la línea 14 utilizando la nomenclatura type seguida del nombre de la máquina e indicando todos los estados posibles. Además de la creación de la máquina se debe crear una señal que sirva como cursor para navegar entre los estados, dicho cursor se crea en la línea 15 y se debe crear utilizando un nombre cualquiera y declarar el tipo con el mismo nombre que se le ha colocado a la máquina de estados.

Figura 55.

Declaración de puertos y señales

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  use IEEE.std_logic_unsigned.all;
5  entity main is
6      Generic ( ciclos: Natural:= 12000000);
7      Port ( Clk : in  STD_LOGIC;
8             Rojo : out  STD_LOGIC;
9             Amarillo : out  STD_LOGIC;
10            Verde : out  STD_LOGIC);
11 end main;
12
13 architecture Behavioral of main is
14    type semaforo is (red,yellow, green);
15    signal estados : semaforo;
16    signal meter : Natural := 0;
17 begin
```

Nota. En la figura se puede observar el segmento de código en donde se realiza la declaración de puertos y el inicio de la arquitectura de la máquina de estados de un semáforo. Elaboración propia, realizado con ISE Design Suite 14.

Para la estructura que manejará la arquitectura se realiza utilizando un process para utilizar el reloj y un if que se ejecute siempre que se tengan flancos ascendentes, se debe realizar esto ya que dentro de los estados se deberá ejecutar un retardo. La máquina de estados se crea utilizando un case en donde se nombren todos los estados que se crearon con type. Dentro de cada estado se ejecutan las instrucciones, para este caso se puede observar en la Figura 56. Posee un primer estado llamado red, este estado activa la salida roja y desactiva las salidas del amarillo y verde, además de realizar un retardo en su primera condicional. Una vez el contador alcanza el valor de la sentencia condicional procede a resetear el contador y ejecutar el cambio de estado, dicho cambio se ejecuta simplemente cargando a la señal estado uno de los estados que fueron inicializados en type.

Figura 56.

Máquina de estados utilizando case

```
21          case estados is
22             when red =>
23                Amarillo <= '0';
24                Verde <= '0';
25                Rojo <= '1';
26                if (meter = (5*ciclos))then
27                   meter <= 0;
28                   estados <= green;
29                else
30                   meter <= meter + 1;
31                end if;
32             when yellow =>
33                Amarillo <= '1';
34                Verde <= '0';
35                Rojo <= '0';
36                if (meter = (ciclos))then
37                   meter <= 0;
38                   estados <= red;
39                else
40                   meter <= meter + 1;
41                end if;
42             when green =>
43                Amarillo <= '0';
44                Verde <= '1';
45                Rojo <= '0';
46                if (meter = (4*ciclos))then
47                   meter <= 0;
48                   estados <= yellow;
49                else
50                   meter <= meter + 1;
51                end if;
52             when others =>
53                estados <= red;
54          end case;
```

Nota. En la figura se puede observar el segmento de código en donde se realiza la máquina de estados utilizando case. Elaboración propia, realizado con ISE Design Suite 14.

En esta máquina de estados se ejecutan automáticamente las transiciones sin ninguna intervención externa solamente utilizando la condicional del tiempo que transcurre entre estados. Siempre que se utiliza esta sintaxis de máquinas de estados en VHDL se debe tener una serie de consideraciones para evitar fallos al momento de sintetizar el código.

- Todas las salidas deben de cambiar dentro de la máquina de estados, VHDL no admite un estado permanente en la salida, si se llega a realizar esto VHDL desactiva la salida y marca una advertencia.

- Si se utilizan vectores dentro de la máquina de estados se debe recorrer todo el vector y utilizar todos los valores, ya que de no hacerlo VHDL desactiva estas entradas y marca una advertencia.

- Los estados de transición siempre deben tener una posibilidad de ejecutarse, ya que de otro modo la sinterización fallara.

El uso de las máquinas de estado simplifica los diseños en VHDL, solo debe de realizarse el diagrama de estados para poder asegurar una correcta redacción del código y tomar en cuenta las consideraciones realizadas anteriormente para evitar errores en la sintetización.

3.3.5.1.1. Esquemático de Máquina de estado Semáforo automático

Para realizar el esquemático en PlanAhead se han realizado algunas modificaciones al código para eliminar los retardos y solamente analizar los cambios de estado, esto se ha realizado por lo visto en el ejemplo del retardo y las múltiples instancias que este genera a partir de agregar un contador que retarde los procesos. El código utilizado puede observarse en la Figura 57.

Figura 57.

Código nuevo de máquina de estados

```
17  begin
18     process(Clk)
19     begin
20        if (rising_edge(Clk)) then
21           case estados is
22              when red =>
23                 Amarillo <= '0';
24                 Verde <= '0';
25                 Rojo <= '1';
26                 estados <= green;
27              when yellow =>
28                 Amarillo <= '1';
29                 Verde <= '0';
30                 Rojo <= '0';
31                 estados <= red;
32              when green =>
33                 Amarillo <= '0';
34                 Verde <= '1';
35                 Rojo <= '0';
36                 estados <= yellow;
37              when others =>
38                 estados <= red;
39           end case;
40        end if;
41     end process;
42
43  end Behavioral;
```

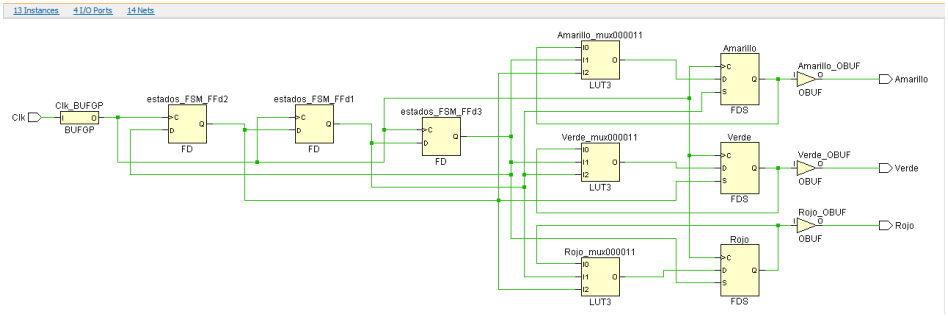
Nota. En la figura se puede observar el segmento de código modificado para realizar la evaluación de su circuito esquemático. Elaboración propia, realizado con ISE Design Suite 14.

Este nuevo código da como resultado un esquemático con 13 instancias. Inicialmente existe una entrada de reloj que permite sincronizar todos los flip flops. El primer estado en ejecutarse es el estado FFd2, que es el que va a iniciar la salida del led rojo. Este estado está directamente conectado a las salidas y al siguiente flip flop que al momento de recibir la señal realiza el cambio en la salida, en este caso es el estado FFd1 y es quien activara el led verde. Finalmente, el tercer y último estado FFd3 recibe la señal y envía su respectiva salida. La forma en que tiene este circuito para controlar las salidas es utilizando multiplexores nombrados como Rojo, Verde y Amarillo. Los nombres son por las salidas que se tienen en los circuitos, cada salida de los multiplexores está asociada a un flip flop de tipo D que mantiene

el valor hasta que sufre un cambio a 0 en sus entradas, esto ocurre siempre en cada cambio de estado y es de esta forma en que logra encender uno de los leds y apagar los otros dos. El mecanismo de la máquina de estados es sencillo ya que el funcionamiento se basa en flip flops conectados en cascada y que pueden hacer variar sus salidas a medida que los bits de reloj activan progresivamente los flip flops iniciales. El esquemático de este circuito se puede observar a mayor detalle en la Figura 58.

Figura 58.

Esquemático de máquina de estados de un semáforo



Nota. En la figura se puede observar la captura del diagrama del circuito de la máquina de estados del semáforo. Elaboración propia, realizado con ISE Design Suite 14.

3.3.5.2. Máquina de estado Semáforo automático con botón de reducción de tiempos

La máquina de estados de un semáforo es secuencial por la forma en que se debe construir, pero esto puede cambiar si se le agrega una variable adicional. Se propone en el ejemplo realizar la implementación de un botón para peatones que reduzca el tiempo de espera para cruzar la calle, este tipo de botones funcionan reduciendo los tiempos del estado en verde haciendo el

cambio a rojo de forma más rápida, por lo tanto, se toma el ejemplo base de la máquina de estados y se agrega una entrada más que será el botón.

Figura 59.

Cambio de la máquina de estados del semáforo

```
43              when green =>
44                 Amarillo <= '0';
45                 Verde <= '1';
46                 Rojo <= '0';
47                 if (meter = (6*ciclos))then
48                    meter <= 0;
49                    estados <= yellow;
50                 else
51                    meter <= meter + 1;
52                    if ((Btn = '0') and (meter < (2*ciclos))) then
53                       meter <= 0;
54                       estados <= pulso;
55                    end if;
56                 end if;
57              when pulso =>
58                 Amarillo <= '0';
59                 Verde <= '1';
60                 Rojo <= '0';
61                 if (meter = (ciclos))then
62                    meter <= 0;
63                    estados <= yellow;
64                 else
65                    meter <= meter + 1;
66                 end if;
67              when others =>
68                 estados <= red;
69           end case;
```

Nota. En la figura se puede observar la captura del cambio realizado en la máquina de estados del semáforo agregando un estado de retardo. Elaboración propia, realizado con ISE Design Suite 14.

El cambio que se realiza al programa es implementar dentro del contador del estado green una segunda condición que evalúa si el botón ha sido pulsado y el tiempo que ha transcurrido en verde el semáforo, esto se hace para que si el semáforo está a punto de cambiar no se pulse el botón y alargue el tiempo en verde del mismo. Una vez se pulsa el botón y se cumple con el criterio de tiempo se produce un cambio de estado. En el nuevo estado llamado pulso se realiza un nuevo retardo que tarda solamente 1 segundo en cambiar de estado al yellow y que continúe con el ciclo normal del semáforo.

Este ligero cambio ha hecho que la máquina de estados esté condicionada al estado actual y a una salida, lo que la convierte en una maquina modelo Mealy.

3.3.5.3. Conclusiones de las máquinas de estados en VHDL

Las máquinas de estados son útiles en VHDL para mantener el orden de los circuitos secuenciales, podrían no ser utilizadas con las palabras reservadas que se han mostrado con anterioridad, pero al momento en que VHDL las sintetizaran se obtendría el mismo esquemático y por este motivo es recomendable su uso, puede simplificar ideas complejas y optimiza los recursos empleados al momento de crearlas.

Las máquinas de estados serán utilizadas en ejemplos futuros como lo son la comunicación UART, es aquí donde radica la importancia de comprenderlas y saber implementarlas dentro de VHDL para que simplifiquen la implementación de cualquier circuito.

3.3.6. Instanciación de proyectos

Realizar una instanciación en un proyecto de VHDL es tener varios módulos y unirlos por medio de conectores internos en VHDL. Esto es muy útil a la hora de realizar proyectos grandes, al trabajar de forma modular cada uno de los circuitos solo se debe pensar en entradas y salidas, no en la arquitectura interna de los módulos y como relacionarlos.

En VHDL la forma de realizar esto es:

- Crear y programar los diferentes módulos en VHDL.

- Crear un módulo top que tendrá la jerarquía más alta entre los módulos, dentro de este módulo se realizan las conexiones.

- Enlazar el archivo UCF para configurar los puertos de la Elbert V2.

Para representar de mejor forma la manera en que se puede lograr la instanciación de un proyecto se propone el siguiente ejemplo:

Se realizará un proyecto en VHDL en el cual se desplegarán los valores en uno de los display de 8 segmentos de la tarjeta Elbert V2. El objetivo es mostrar el número del dip switch que se active. Esto puede realizarse sin realizar instanciación, pero el proyecto sería muy grande, por lo tanto, se dividirá en dos módulos, la forma de realizarlos es:

- El primer módulo consiste en un lector, utilizara una entrada de 8 bits y evaluara los 8 posibles estados, la salida del módulo será un vector de 4 bits que indicara en binario que dip switch fue accionado.

- El segundo módulo tomara la salida de 4 bits del primer módulo y realizara la evaluación de que numero es y activara las salidas correspondientes para desplegar el número en el display.

Los módulos serán sincronizados por medio el reloj interno, por lo tanto, existe la entrada de reloj para ambos. La forma de realizar la programación de ambos módulos es igual a crear un archivo main dentro de VHDL, es un programa normal tal como se puede observar en las Figuras 60 y 61.

Figura 60.

Programación del primer módulo

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3   use IEEE.numeric_std.all;
4   use IEEE.std_logic_unsigned.all;
5   entity lector is
6       Port ( clk    : in STD_LOGIC;
7              numero : in  STD_LOGIC_VECTOR (7 downto 0);
8              salida : out  STD_LOGIC_VECTOR (3 downto 0));
9   end lector;
10  architecture Behavioral of lector is
11  begin
12  process (clk, numero)
13     begin
14        if (rising_edge(clk)) then
15        case (numero) is
16           when "00000001" =>      salida <= "0001";
17           when "00000010" =>      salida <= "0010";
18           when "00000100" =>      salida <= "0011";
19           when "00001000" =>      salida <= "0100";
20           when "00010000" =>      salida <= "0101";
21           when "00100000" =>      salida <= "0110";
22           when "01000000" =>      salida <= "0111";
23           when "10000000" =>      salida <= "1000";
24           when others =>          salida <= "0000";
25        end case;
26        end if;
27     end process;
28  end Behavioral;
```

Nota. En la figura se puede observar la captura del código del registro de las 8 entradas de la FPGA. Elaboración propia, realizado con ISE Design Suite 14.

Figura 61.

Programación del segundo módulo

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
entity numeros is
    Port ( clk : in  STD_LOGIC;
           btn : in  STD_LOGIC_VECTOR (3 downto 0);
           Enable0 : out  STD_LOGIC;
           Enable1 : out  STD_LOGIC;
           Enable2 : out  STD_LOGIC;
           salida : out  STD_LOGIC_VECTOR (7 downto 0));
end numeros;
architecture Behavioral of numeros is
begin
process (clk, btn)
   begin
      if (rising_edge(clk)) then
      case (btn) is
         when "0001" =>
            salida <= "11111001";
            Enable0 <= '1';
            Enable1 <= '1';
            Enable2 <= '0';
         when "0010" =>
            salida <= "10100100";
            Enable0 <= '1';
            Enable1 <= '1';
            Enable2 <= '0';
         when "0011" =>
            salida <= "10110000";
            Enable0 <= '1';
            Enable1 <= '1';
            Enable2 <= '0';
         when "0100" =>
            salida <= "10011001";
            Enable0 <= '1';
            Enable1 <= '1';
            Enable2 <= '0';
         when "0101" =>
            salida <= "10010010";
            Enable0 <= '1';
            Enable1 <= '1';
            Enable2 <= '0';
         when "0110" =>
            salida <= "10000010";
            Enable0 <= '1';
            Enable1 <= '1';
            Enable2 <= '0';
         when "0111" =>
            salida <= "11111000";
            Enable0 <= '1';
            Enable1 <= '1';
            Enable2 <= '0';
         when "1000" =>
            salida <= "10000000";
            Enable0 <= '1';
            Enable1 <= '1';
            Enable2 <= '0';
         when others =>
            salida <= "01111111";
      end case;
      end if;
   end process;

end Behavioral;
```

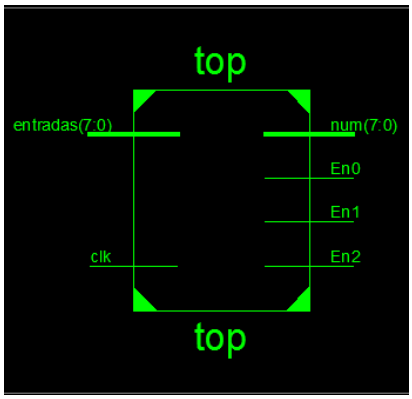
Nota. En la figura se puede observar la captura del código de las salidas con los números que se deben desplegar en el Display. Elaboración propia, realizado con ISE Design Suite 14.

Estos módulos pueden funcionar como circuitos independientes enlazando sus entradas y salidas a un archivo UCF y mostrarlos en la FPGA, por lo tanto, para lograr la instanciación se debe crear un módulo extra que sirva como conector de estos módulos.

Para realizar el circuito se debe pensar su construcción como un módulo completo incluyendo todos sus submódulos. Inicialmente se debe considerar las salidas y entradas que interactúan directamente con la FPGA, estas son las 8 entradas de los dip switches, 8 salidas o segmentos del display, 1 entrada de reloj y 3 salidas que conectan con el encendido o apagado de los 3 displays.

Figura 62.

Entradas y salidas del ejemplo de instanciación



Nota. En la figura se puede observar el módulo principal del ejemplo de instanciación. Elaboración propia, realizado con ISE Design Suite 14.

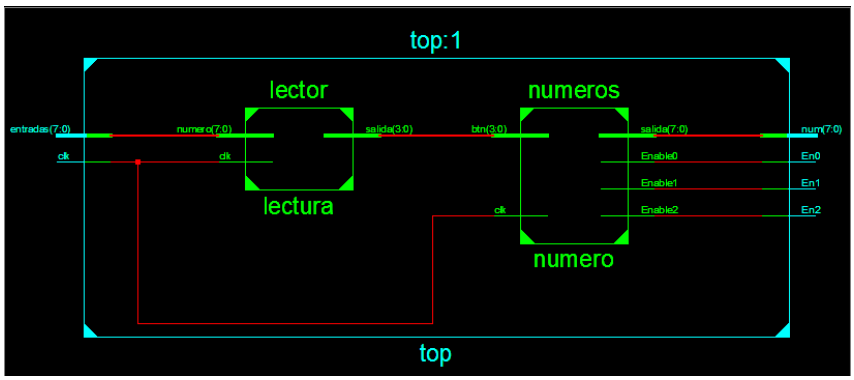
La arquitectura del módulo principal consiste en enlazar las entradas y salidas de sus submódulos. El primer módulo lector debe enlazar su entrada de 8 bits y la entrada de reloj con las entradas del módulo top. El segundo

módulo números debe enlazar su entrada de reloj con la entrada del módulo top y la salida de 8 bits junto con las 3 salidas que habilitan los display con las salidas del módulo top.

Una vez se han realizado los enlaces de entradas y salidas de forma directa se debe considerar los enlaces internos, para este caso los enlaces internos es la salida de 4 bits del primer módulo y la entrada de 4 bits del segundo módulo. En la Figura 63 se ilustra de una manera más sencilla todo lo mencionado con anterioridad.

Figura 63.

Entradas y salidas internas del ejemplo de instanciación



Nota. En la figura se puede observar los submódulos que componen el módulo principal del ejemplo de instanciación. Elaboración propia, realizado con ISE Design Suite 14.

Ahora que se tiene claro el modo en que serán ordenadas las entradas y salidas del circuito se mostrara el cómo se realiza la instanciación dentro de VHDL. La sintaxis con la que se realizan las instanciaciones se puede observar en la Figura 64. Inicialmente se debe nombrar la instancia con cualquier nombre, es recomendable hacerlo con uno que identifique lo que hace el módulo, seguido a esto se escriben las palabras reservadas entity work. y

después del punto debe colocarse el nombre del submódulo para que VHDL reconozca el enlace. Una vez se han realizado los nombramientos de los módulos se debe enlazar los puertos, esto se hace por medio de mapeado y la palabra reservada es port map, en el mapeado se debe colocar todos los puertos del submódulo con el nombre exacto en el que están programados y serán asignados a los puertos del módulo top. En los casos en donde se debe realizar un enlace interno de los puertos se debe hacer uso de señales auxiliares para llevar la información de forma interna de un módulo a otro.

Figura 64.

Sintaxis de instanciación

```
cualquiernombre:
   entity work.nombredelsubmodulo
      port map(
         Puerto_del_submodulo => puerto_del_modulo_principal
      );
```

Nota. En la figura se puede observar la sintaxis de la instanciación. Elaboración propia, realizado con ISE Design Suite 14.

Para el caso del ejemplo que se ha propuesto se puede ver el módulo top en la Figura 65. El primer módulo lector fue asignado a un submódulo llamado lectura y se puede observar que las entradas originales del módulo han sido asignadas a sus respectivos puertos con excepción a la salida que fue asignada a una señal auxiliar con nombre data_1. El segundo submódulo se ha asignado a número y se ha realizado el mapeado. Para este módulo se ha utilizado la señal auxiliar del módulo anterior con nombre data_1 para asignar la salida del primer submódulo con la entrada del segundo submódulo, dicha carga de datos puede observarse en la línea 25.

Figura 65.

Módulo top del ejemplo de instanciación

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  entity top is
4     port(
5        clk    : in STD_LOGIC;
6        entradas : in  STD_LOGIC_VECTOR (7 downto 0);
7        En0 : out  STD_LOGIC;
8        En1 : out  STD_LOGIC;
9        En2 : out  STD_LOGIC;
10       num : out  STD_LOGIC_VECTOR (7 downto 0));
11 end top;
12 architecture Behavioral of top is
13 signal data_1: std_logic_vector(3 downto 0);
14 begin
15    lectura:
16       entity work.lector
17          port map(
18             clk => clk,
19             numero => entradas,
20             salida => data_1);
21    numero:
22       entity work.numeros
23          port map(
24             clk => clk,
25             btn => data_1,
26             Enable0 => En0,
27             Enable1 => En1,
28             Enable2 => En2,
29             salida => num);
30 end Behavioral;
```

Nota. En la figura se puede observar el código del módulo principal donde se realiza la instanciación del proyecto. Elaboración propia, realizado con ISE Design Suite 14.

Al momento que ISE realiza la sinterización entiende que la señal auxiliar data_1 sirve como conector y une directamente la salida de un módulo con la entrada de otro, esto es el punto más importante en las instanciaciones ya que entre módulos no es importante la información interna o los procesos que cada uno ejecuta porque solo importa la salida que de uno y la entrada del otro. Esto da una gran ventaja de diseño en paralelo al poder distribuir distintos módulos esperando solo una determinada salida o entrada de los diferentes sistemas.

Al momento de llevar este circuito a la Elbert V2 bastara simplemente con cargar el UCF al icono del chip en ISE y asignar solo las entradas y salidas del módulo Top ya que todas las demás serán de uso interno.

4. CIRCUITOS CON APLICACIONES EN VHDL

El lenguaje VHDL tiene la capacidad de traducir las instrucciones por código a configuraciones con compuertas y flip flops que realizan dichas instrucciones dentro de una tarjeta FPGA. Esta capacidad puede ser explotada para realizar aplicaciones de uso común y de esta forma poder aprovechar toda la potencia que posee la Elbert V2.

4.1. Aplicación de antirrebote de botón en VHDL

El efecto antirrebote de un botón es de gran utilidad para múltiples aplicaciones que necesiten realizar interacciones con los botones mecánicos, estos al ser componentes físicos que sufren desgaste pueden ocasionar lecturas erróneas en algunas ocasiones, el antirrebote ayuda a que esto se minimice y los programas tengan una mayor calidad de ejecución, a lo largo del capítulo se explorara la programación de dicho botón en VHDL.

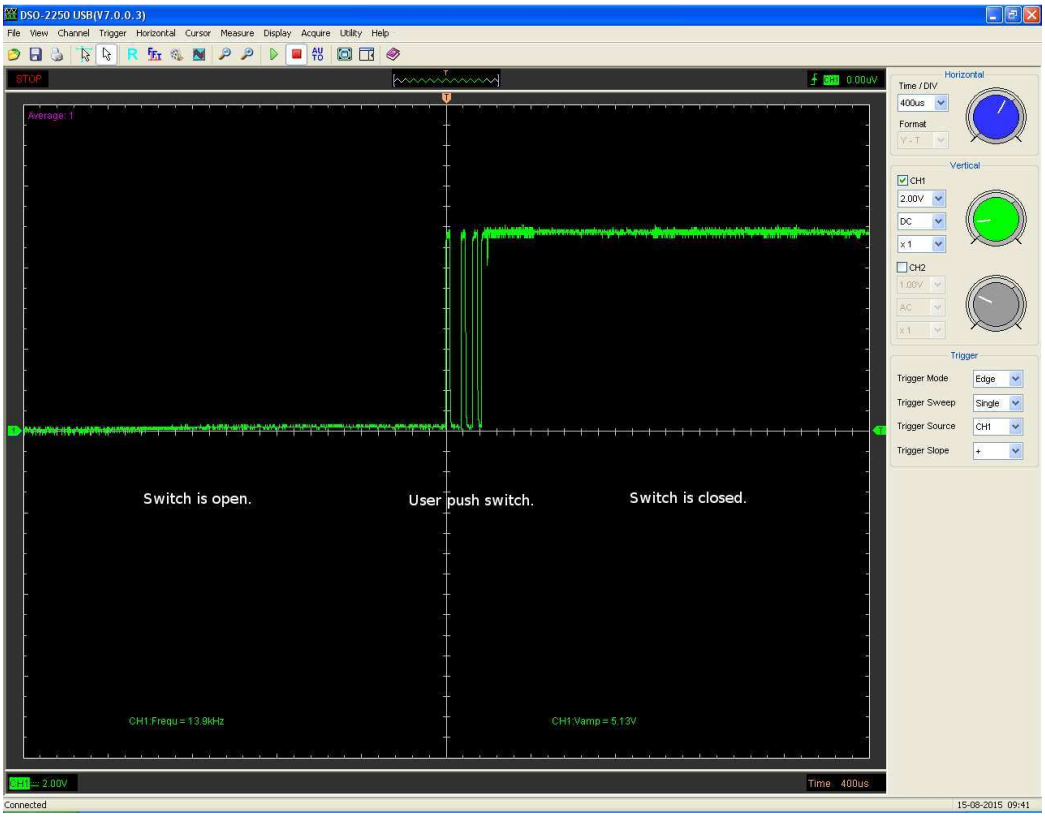
4.1.1. Teoría función de antirrebote en un botón

La función antirrebote hace referencia a un tipo de código interno que interactúa con el hardware de un microcontrolador o microprocesador. El efecto rebote en los botones es un inconveniente mecánico de su funcionamiento, este consiste en que al momento de pulsar un botón los contactos metálicos del botón producen conexiones inestables hasta que transcurre un cierto tiempo y se estabilizan, en este periodo debido a la velocidad del reloj se puede interpretar de forma interna que se ha pulsado más de 1 vez el botón. Debido a este efecto pueden ocurrir fallos en el

funcionamiento de los programas que sean susceptibles a inicios con el uso de botones.

Figura 66.

Gráfico del efecto rebote de un botón push



Nota. Gráfico que muestra los estados por los que pasa una señal al momento que se pulsa un botón. Obtenido de ALL ABOUT CIRCUITS. *Switch Bounce and How to Deal with It*. (https://www.allaboutcircuits.com/technical-articles/switch-bounce-how-to-deal-with-it/), consultado el 10 de abril de 2023. De dominio público.

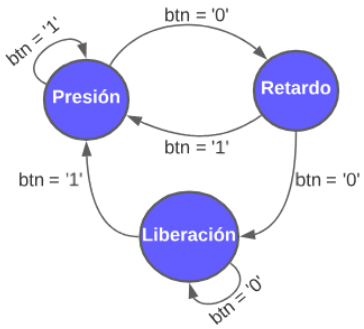
4.1.2. Programa en VHDL de la función antirrebote.

La función antirrebote debe realizar la evaluación del estado inicial del botón, almacenar el dato en un variable y corroborar que el botón ha cambiado de estado. En termino más simples, comprueba si el botón es pulsado y liberado por medio de variables auxiliares.

El programa de la función antirrebote es sencillo de implementar en muchos lenguajes de programación, pero en VHDL se debe pensar esta función de una forma un poco diferente. En VHDL para lograr realizar el antirrebote es necesario plantear el programa como una máquina de estados por la forma en que tiene VHDL de sintetizar el código. Como solución se propone una máquina de tres estados que van a realizar la función de antirrebote (aunque para aplicaciones cambiaria a cuatro estados). Los estados de la maquina deben verificar si el botón es presionado, producir un retardo y por último realizar una evaluación si el botón fue liberado o sigue siendo pulsado.

Figura 67.

Máquina de estados de la función antirrebote



Nota. En la figura se puede observar la máquina de estados que será programa en VHDL. Elaboración propia, realizado con Lucidchart.

Tal como se puede observar en la Figura 67 se tienen tres estados que controlaran la función del antirrebote. El estado presión será el estado inicial y evaluara si el botón es presionado, si dicho botón se presiona realiza el cambio de estado y si no se presiona simplemente el estado permanece. El segundo estado retardo es un retardo de 30ms que brinda un espacio de tiempo que permite evaluar si el botón fue pulsado o es un error de hardware por un pulso inicial, la forma en que funciona es aplicando el retardo y una vez termina evalúa el estado del botón. Si el botón se ha pulsado el estado registra la pulsación y realiza el cambio al estado de liberación, de lo contrario interpreta el estado pulsado como un error y regresa al estado inicial presión. El ultimo estado llamado liberación consiste en evaluar que se deje de presionar el botón, si el botón se mantiene pulsado la maquina permanece en este estado y si el botón se suelta regresa al estado inicial esperando una nueva pulsación. Si se desea utilizar esta máquina de estados para alguna aplicación bastara con agregar el estado de la acción después del estado de liberación ya que es aquí donde se confirma la pulsación y liberación del botón.

El código en VHDL de esta función se puede observar en la Figura 68. La máquina de estados es implementada y la salida de este ejemplo se implementa en el estado liberación en la línea 41 y 42. La salida consiste en un estado auxiliar que oscila entre 1 y 0 con cada pulsación de botón. Este código puede servir para múltiples aplicaciones que requieran conteo de pulsaciones, solo se debe modificar la etapa de la salida para que tenga aplicaciones universales.

Figura 68.

Máquina de estados de la función antirrebote implementada en VHD

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
entity main is
    Port ( clk : in  STD_LOGIC;
           btn : in  STD_LOGIC;
           sal : out  STD_LOGIC);
end main;
architecture Behavioral of main is
   type maquina is (presion, retardo, liberacion);
   signal estados: maquina;
   signal led : STD_LOGIC := '0';
   signal conteo : Natural := 0;
begin
   process (clk, btn)
   begin
      if(rising_edge(clk)) then
         case estados is
            when presion =>
               if (btn = '0') then
                  estados <= retardo;
               else
                  estados <= presion;
               end if;
            when retardo =>
               if (conteo = 360000) then -- El 360000 es para los 30ms
                  conteo <= 0;
                  if (btn = '0') then
                     estados <= liberacion;
                  else
                     estados <= presion;
                  end if;
               else
                  conteo <= conteo + 1;
               end if;
            when liberacion =>
               if (btn = '0') then
                  estados <= liberacion;
               else
                  led <= not(led);
                  sal <= led;
                  estados <= presion;
               end if;
            when others =>
               estados <= presion;
         end case;
      end if;
   end  process;

end Behavioral;
```

Nota. En la figura se puede observar la máquina de estados programada en VHDL. Elaboración propia, realizado con ISE Design Suite 14.

4.2. Módulo ADC en la FPGA Elbert V2

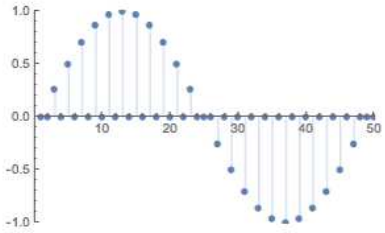
Los módulos ADC son útiles para realizar la medición de algunas señales que pueden transmitir los sensores que funcionan de forma análoga, gracias a que estos al entregan la señal digital de dichos sensores y estas pueden ser utilizadas por la FPGA para ejecutar distintas tareas. A continuación, se explora más a detalle el uso y creación de este tipo de módulos.

4.2.1. Definición de módulo ADC

Un módulo ADC hace referencia a un tipo de circuito que es capaz de transformar una señal analógica a una señal digital. La forma en que funcionan estos módulos es por medio de dos procesos, el primero llamado muestreo el cual consiste en representar una señal por medio de varios pulsos y el segundo se le conoce como cuantificación que consiste en asignar valores de voltaje a los pulsos obtenidos de la señal muestreada. Además de los procesos mencionados con anterioridad se debe realizar un último paso que consiste en convertir las señales cuantificadas en valores binarios para que puedan ser leídos utilizando un microcontrolador o una FPGA.

Figura 69.

Ejemplo de una función muestreada



Nota. En la figura se puede observar una función senoidal con varias muestras que conforman su señal. Elaboración propia, realizado con Wolfram Online.

4.2.2. Diseño de ADC

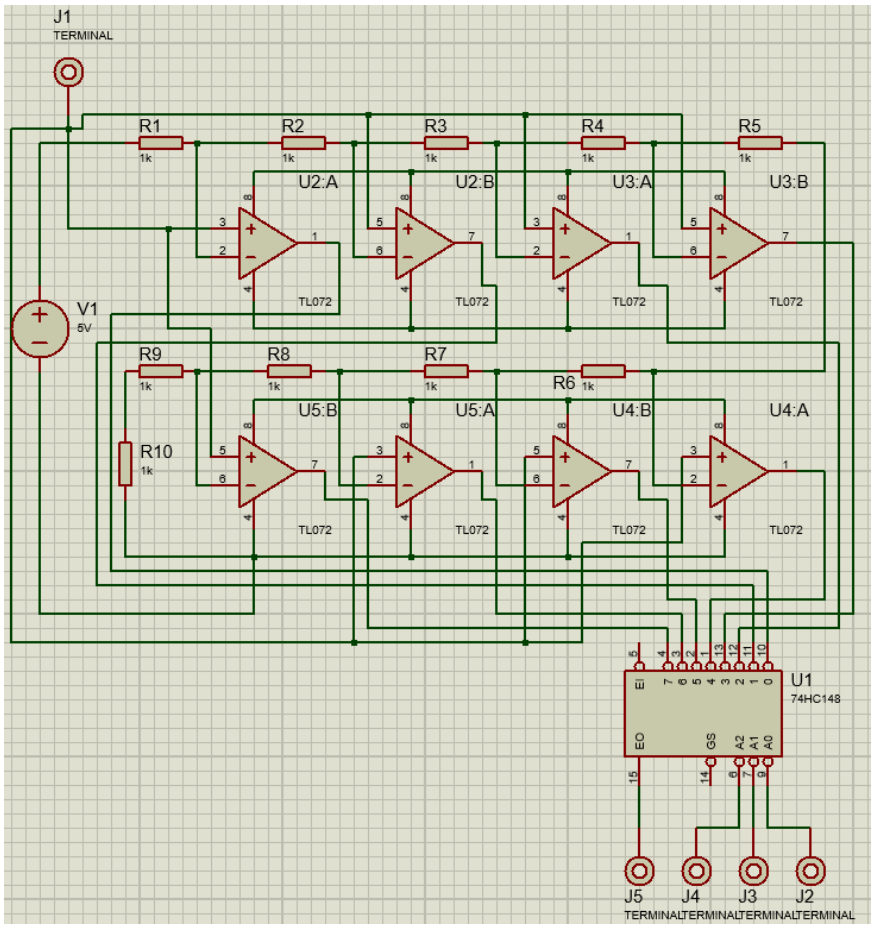
La FPGA Elbert V2 a diferencia de otras tarjetas no cuenta con puertos que posean conversores análogo-digital y por este motivo se debe de diseñar un módulo alternativo que sea un ADC y pueda entregar a la Elbert V2 señales en forma binaria para que puedan ser interpretadas por medio de código en VHDL.

La solución que se propone es utilizar un módulo externo que entregue los niveles de voltaje de forma binaria para que estas salidas puedan ser conectadas a la FPGA. Se optará por realizar un circuito ADC desde cero para no involucrar módulos prefabricados que estén sujetos a disponibilidad. Para realizar el módulo se necesita de un componente que pueda tomar la señal en una de sus entradas y dependiendo los valores de voltaje que este tenga accione salidas. La forma de realizar dicho módulo es utilizando amplificadores operacionales junto a un enconder que entregará una señal binaria y esta será enviada a la tarjeta. Los componentes que serán utilizados son los siguientes:

- 4 amplificadores operacionales TL072
- 10 resistencias de 1 kΩ
- 1 *Encoder* 8 a 3 modelo SN74HC148N

Figura 70.

Diagrama esquemático del ADC



Nota. Esquemático del circuito ADC. Elaboración propia, realizado con Proteus 8 Professional.

La forma en que funciona el módulo es utilizando un divisor de voltaje y configurando el amplificador operacional en modo de comparador para tomar

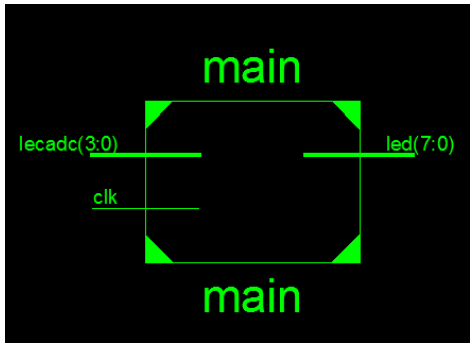
cada uno de los voltajes y activar las salidas conforme se iguale los voltajes comparados. La entrada de la terminal J1 es donde va la señal analógica que se desea convertir y las salidas J2, J3, J4 y J5 son las que irán conectadas a la FPGA Elbert V2. Estas salidas funcionan como un contador binario, siendo J2 la menos significativa y J4 la más significativa. La salida J5 cambia de estado únicamente si todas las entradas del *decoder* son altas, por lo tanto, se utiliza para tener una mayor resolución del ADC.

4.2.3. Código lector ADC en VHDL

El lector del circuito ADC por medio de VHDL simplemente se basa en realizar una lectura de los 4 bits de salida que tiene el lector, se utiliza la numeración binaria para conocer los valores que arroja el módulo. Es necesario hacer uso de dos entradas, la primera es la entrada de reloj y la segunda es la entrada de los bits del ADC. La salida para hacer el módulo de muestra serán los 8 leds que tiene disponible la tarjeta.

Figura 71.

Diagrama lector ADC en VHDL



Nota. Esquemático del lector ADC en VHDL. Elaboración propia, realizado con ISE Design Suite 14.

Al definir las entradas y salidas se puede evaluar la lógica que debe ser programada para leer correctamente el módulo creado. En el circuito creado se debe considerar cual será el valor menor y el mayor a leer. El valor menos significativo seria 0000 que es el equivalente a no tener entrada de voltaje para el ADC. El valor más significativo será 1111 que representa una entrada de voltaje igual al voltaje de referencia con el que es alimentado el módulo ADC. Entre los valores que se leerán se debe considerar que existe un salto entre el orden de los números binarios del 0111 al 1111, ya que en realidad solo se tienen 9 opciones de salida debido a la configuración del *encoder* SN74HC148N y solo se puede obtener el 1 en el valor más significativo si todos los demás se encuentran activos o en estado 1.

En VHDL se puede realizar un recorrido del vector lecadc por medio de un case, esto funcionara siempre que exista un cambio de estado en el reloj y en la salida del ADC. El programa puede ser modificado partiendo de este concepto, siempre que la entrada cumpla con la condición del case se puede ejecutar cualquier instrucción dentro del mismo. La programación del lector puede observarse en la Figura 72.

Figura 72.

Código lector ADC en VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
entity main is
    Port ( clk : in  STD_LOGIC;
           lecadc : in  STD_LOGIC_VECTOR (3 downto 0);
           led : out  STD_LOGIC_VECTOR (7 downto 0));
end main;
architecture Behavioral of main is
begin
   process (clk, lecadc)
   begin
      if (rising_edge(clk)) then
         case (lecadc) is
            when "0000" => led <= "00000000";
            when "0001" => led <= "00000001";
            when "0010" => led <= "00000010";
            when "0011" => led <= "00000100";
            when "0100" => led <= "00001000";
            when "0101" => led <= "00010000";
            when "0110" => led <= "00100000";
            when "0111" => led <= "01000000";
            when "1111" => led <= "10000000";
            when others => led <= "00000000";
         end case;
      end if;
   end process;
end Behavioral;
```

Nota. Código del lector ADC en VHDL. Elaboración propia, realizado con ISE Design Suite 14.

4.3. Movimiento de un servomotor usando VHDL

Una FPGA puede realizar el movimiento de un servomotor con el uso del reloj y la memoria ROM una señal PWM. Existen algunas de estas que ya poseen un módulo implementado para realizar dichas operaciones, pero en el caso de la Elbert V2 esto no es así, para la tarjeta Elbert V2 se deben crear varios módulos que permitan crear estas señales, pero antes de indicar el código y su funcionamiento es vital conocer la naturaleza y funcionamiento de

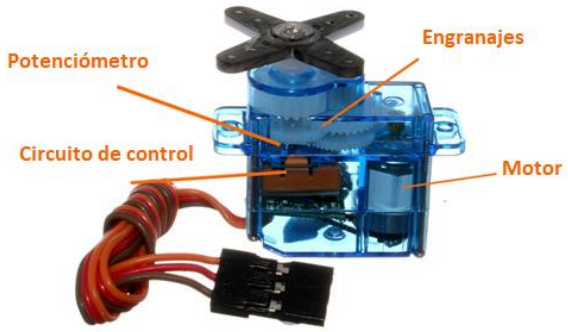
los servomotores para poder aprovechar al máximo el código utilizado y lograr realizar el movimiento de dicho motor.

4.3.1. Funcionamiento y teoría de un servomotor

Un servomotor es un tipo especial de motor que utiliza señales a diferentes frecuencias para lograr realizar movimientos de precisión, estos motores también reciben el nombre de motores síncronos. Un servomotor está compuesto principalmente por tres componentes los cuales son un motor DC/AC (dependiendo de la aplicación), un circuito controlador y un sensor que indique posiciones. Las construcciones pueden variar dependiendo de la aplicación a la que se dirija, en ocasiones puede contener engranajes, sensores de efecto hall o *encoders*, pero siempre son las mismas piezas con ligeras variaciones.

Figura 73.

Gráfico partes generales de un servomotor



Nota. Gráfico que muestra las partes internas de un servomotor SG90. Obtenido de Solo Motor Controllers. *SERVO MOTOR | BASICS, WORKING PRINCIPLE, THEORY, AND MORE*. (https://www.solomotorcontrollers.com/blog/servo-motor/), consultado el 17 de junio de 2023. De dominio público.

La forma de conexión de un servomotor puede variar dependiendo del fabricante del que se adquiera el motor, pero casi todos siguen el siguiente estándar:

- Cable Rojo: señal de 5 o 12 volteos dependiendo del tipo de servomotor.
- Cable Café: señal de tierra general para el motor.
- Cable Anaranjado: señal PWM que viene del generador de señal externa.

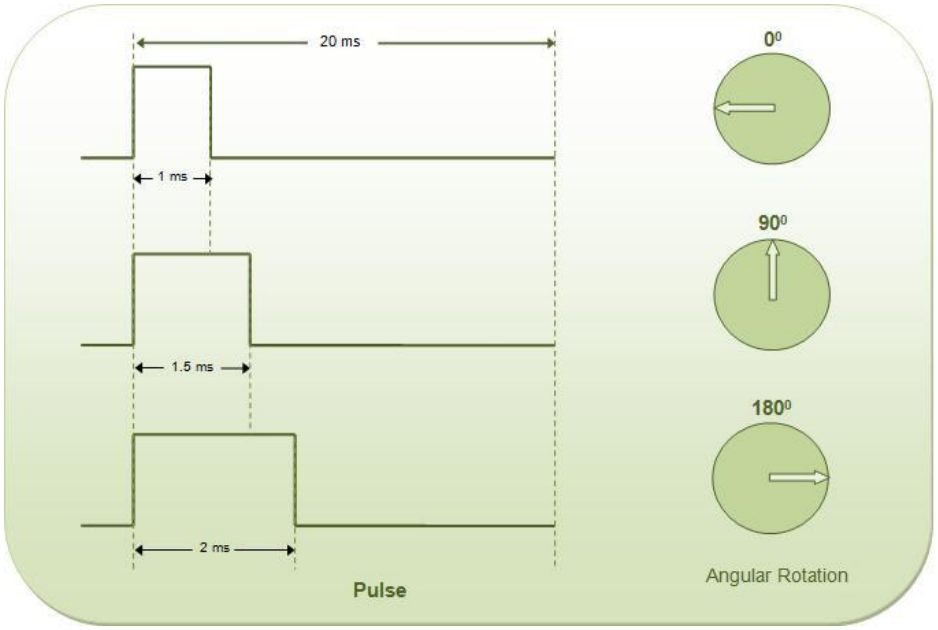
Dependiendo de los fabricantes los colores que varían son el café por un negro o el anaranjado por un amarillo, pero la gran mayoría sigue este patrón. Se debe identificar correctamente cada uno de los cables y que señales son las que utiliza su diseño, ya que los servomotores son de manejo delicado y es sencillo estropearlos por una conexión incorrecta. Es recomendable aislar de la mejor manera posible los cables de los servomotores del ruido y de las interferencias electromagnéticas, esto se realiza para evitar que los cambios de los ángulos sean inestables o inesperados y debe de realizarse como una consideración en la etapa de diseño de cualquier proyecto.

Tal como fue mencionado con anterioridad la forma en que tiene el servomotor de mover su eje al ángulo deseado es utilizando una señal PWM con una frecuencia fija. Se conoce gracias a múltiples pruebas y por algunas hojas de datos de los fabricantes que se requiere un pulso de 1 milisegundo para lograr colocar un servomotor a 0 ° y requiere otro pulso de 2 milisegundos para lograr colocar dicho servomotor a 180 °. Los pulsos deben durar un total de 20 milisegundos y deben ser enviados con una frecuencia de 50 Hz para mantener la orientación del ángulo en su posición, si es necesario hacer un cambio en el ángulo se puede variar entre los pulsos de 1 y 2 milisegundos

para lograr los cambios deseados, esto se ilustra de una mejor manera en la Figura 74.

Figura 74.

Pulsos y movimiento del eje de un servomotor



Nota. Gráfico que muestra los movimientos del eje de un servomotor dependiendo los anchos de los pulsos PWM. Obtenido de Engineers Garage. *SERVO Servo Motor : Basics and Working. (*https://www.engineersgarage.com/servo-motor-basics-and-working/*),* consultado el 18 de junio de 2023. De dominio público.

La señal que marque los cambios en el ángulo del eje debe ser enviadas constantemente, por lo tanto, es posible obtener la señal utilizando una señal de un microcontrolador o microprocesador que envíe una señal PWM que tenga las características ya mencionas o hacer uso de un circuito con un temporizador como el 555 para lograr las señales PWM, la señal podría ser generada por un generador de ondas, aunque esta aplicación es poco usual.

Una vez se ha comprendido el funcionamiento básico de un servomotor se puede iniciar con el diseño de un módulo que pueda crear la señal PWM ya mencionada. Se debe recalcar que dicha señal debe ser tratada con un modelo matemático para poder funcionar en los pines de la tarjeta Elbert V2 por las limitaciones de la propia tarjeta.

4.3.2. Módulos utilizados para mover un servomotor

El código en VHDL buscara realizar un recorrido desde 0° hasta los 180° que puede rotar el servomotor SG90, para recrear las señales del PWM se necesita de un módulo que sea capaz de brindar pulsos con duraciones especificas tal como fue indicado en los párrafos anteriores, para hacerlo se requiere que el módulo cree dichas señales y establezca una periodicidad haciendo las conversiones necesarias para hacer los divisores de frecuencia para marcar la periodicidad de cada señal.

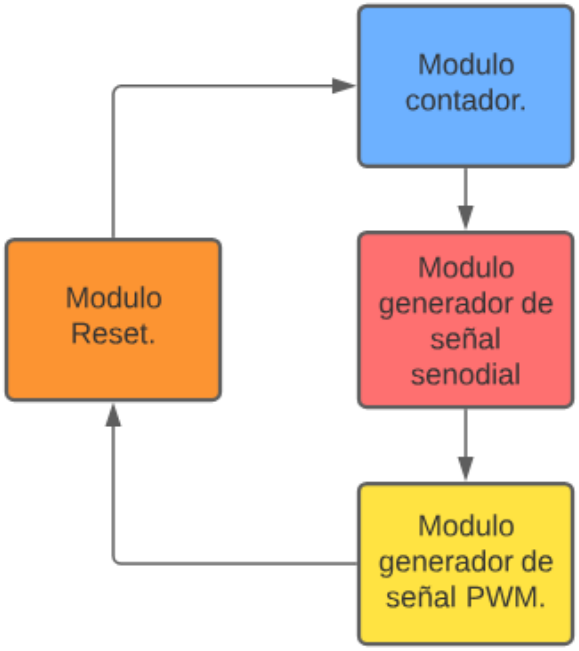
En el módulo del PWM se debe dejar una entrada que indique el cambio de posición, para poder realizar los cambios correctamente se debe realizar un desplazamiento utilizando una función sinusoidal dentro de los controles de posición del PWM, para hacer esto se debe crear un módulo que haga un recorrido de varios resultados de la señal senoidal y envíe los resultados al módulo de PWM para que sea posible el cambio de posición dentro del mismo, sin esta función no es posible hacer el cambio de altos y bajos típicos del PWM debido al reloj de la FPGA.

Por último, se debe tener un contador independiente que brinde el recorrido del módulo de la señal senoidal y otro módulo que pueda servir para reiniciar los valores de las variables a sus valores iniciales. El resultado final

dará una interacción de cuatro módulos con los que se podrá realizar los desplazamientos de los ángulos que serán integrados de forma modular en un módulo final en el cual serán asignadas las constantes de todos los submódulos.

Figura 75.

Diagrama de flujo del módulo para rotar eje de servomotor



Nota. En la figura se puede observar el flujo de los módulos que serán realizados en VHDL. Elaboración propia, realizado con Visme.

4.3.3. Módulo PWM del Servomotor

Tal como se mencionó con anterioridad en este módulo se busca realizar las señales que se han indicado en la Figura 74, para lograr esto se debe trabajar utilizando varios divisores de frecuencias y algunos cálculos para

obtener los conteos aproximados que marquen los tiempos de los cambios. Como primer paso se debe utilizar una librería especial para poder incorporar constantes matemáticas dentro de los cálculos, dicha librería es IEEE.math_real.round y puede observarse en la imagen a continuación.

Figura 76.

Librerías que se usaran en el módulo del PWM del servomotor

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  use IEEE.math_real.round;
```

Nota. En la figura se puede observar las librerías que se utilizan para el programa. Elaboración propia, realizado con ISE Design Suite 14.

Una vez se han declarado las librerías es necesario declarar las señales que serán utilizadas, en la Figura 77 se puede observar las entradas y salidas del módulo, la función de cada señal se describe a continuación:

- clk_hz: reloj de la FPGA, será un número real.
- Pulso: esta es la frecuencia de 50 Hz que surge de la longitud de los pulsos indicada en la teoría del movimiento del eje de un servomotor en el capítulo 4.3.1, será un número real.
- Pulsomin: ya que se realizará un recorrido de 0 a 180 ° se utilizará esta señal como la frecuencia de pulso para la posición mínima, será un número real.
- Pulsomax: ya que se realizará un recorrido de 0 a 180 ° se utilizará esta señal como la frecuencia de pulso para la posición máxima, será un número real.

- Conteo: esta es la señal que recibe los datos del módulo contador general, esto se realiza para sincronizar el módulo PWM con el módulo de la señal senoidal, será un numero positivo entero, esto se realiza de esta manera para utilizar esta constante como un limitador de rangos y por esto debe establecerse siempre como un valor positivo.
- clk: es la señal del reloj general, este será uno de los puertos de entrada del módulo.
- rst: es la señal que recibirá en caso se pulse un botón para regresar a los valores predefinidos iniciales, este será uno de los puertos de entrada del módulo y estará conectado al módulo de reset.
- Posición: es la señal que recibe la cadena de valores de la señal senoidal para lograr el movimiento del eje, este será uno de los puertos de entrada del módulo y estará conectado al módulo del generador de la señal senoidal.
- spwm: es la salida que debe de ser conectada al pin del servomotor que solicita la señal PWM, este será la única señal de salida del módulo.

Figura 77.

Declaración de las señales del módulo del PWM del servomotor

```
 6  entity ServoPWM is
 7     Generic(
 8           clk_hz : real;
 9           Pulso : real;
10           Pulsomin : real;
11           Pulsomax : real;
12           conteo : positive);
13      Port ( clk : in  STD_LOGIC;
14             rst : in  STD_LOGIC;
15             posicion : in  integer range 0 to conteo - 1;
16             spwm : out  STD_LOGIC);
17  end ServoPWM;
```

Nota. En la figura se puede observar las señales que se utilizan para el programa. Elaboración propia, realizado con ISE Design Suite 14.

El siguiente paso es realizar una función para ahorrar instrucciones dentro de la arquitectura, se realiza dicha función antes del inicio de las instrucciones de la arquitectura y su función es realizar una conversión de los valores de los periodos a unos valores que indiquen cuantos pulsos representan, para realizarlo se utilizaran las constantes antes mencionadas y se debe multiplicar con la frecuencia del reloj, dicho resultado debe ser dividido por el valor 1 x10^6 para que los valores de megas sean eliminados y se tengan los conteos generales, esta función es presentada en la siguiente imagen.

Figura 78.

Función de conversión de los valores de los periodos a pulsos

```
20    function ciclos_us (conteo_us: real) return integer is
21    begin
22       return integer(round(clk_hz / 1.0e6 * conteo_us));
23    end function;
```

Nota. En la figura se puede observar la función que realiza la conversión de periodos a pulsos. Elaboración propia, realizado con ISE Design Suite 14.

Al momento de tener la función de las conversiones el siguiente paso es crear unas constantes con las señales iniciales. En la Figura 79 se puede observar en la línea 26 y 27 que estas constantes realizan la conversión de los pulsos mínimos y máximos, estas señales son importantes porque establecen el rango de trabajo y el conteo total de pasos que debe realizar la FPGA, esto puede observarse de la línea 28 a la 30.

Una vez se tienen los ciclos por paso que debe realizar el servomotor se debe establecer una constante de conteo máxima, esta constante parte de la división de los ciclos de reloj entre la frecuencia del pulso (50 Hz) que indica

la teoría del servomotor. Se realiza el divisor de frecuencia para obtener el rango máximo al que deben llegar las señales para completar los ciclos de 50 Hz de forma exitosa, esta señal es declarada en la línea 32. Como paso final en la declaración de constantes y señales se puede observar en la última línea mostrada en la imagen la declaración de la señal que marca el ciclo de trabajo del servomotor, esta señal se crea utilizando un rango de enteros que tiene como un posible valor máximo a la constante de la cantidad de pulsos necesarios para trasladar el eje del servomotor al valor indicado por el ángulo mayor.

Figura 79.

Declaración de constantes del código del PWM del servomotor

```
26    constant conteo_min    : integer := ciclos_us(Pulsomin);
27    constant conteo_max    : integer := ciclos_us(Pulsomax);
28    constant rango_max_min : real := Pulsomax - Pulsomin;
29    constant conteo_us : real := rango_max_min / real(conteo - 1);
30    constant ciclos_por_paso : positive := ciclos_us(conteo_us);
31    constant cont_max : integer := integer(round(clk_hz / Pulso)) - 1;
32    signal cont : integer range 0 to cont_max;
33    signal duty_cycle : integer range 0 to conteo_max;
```

Nota. En la figura se puede observar el segmento de código que brinda la forma en que las constantes son creadas. Elaboración propia, realizado con ISE Design Suite 14.

Una vez se han creado todas las constantes y señales se puede iniciar con las instrucciones principales del módulo PWM, las acciones serán realizadas en tres módulos, el primero será el encargado de realizar el conteo de los pasos utilizando la constante del conteo máximo, el segundo será el encargado de dar la señal en alto y bajo del PWM con la frecuencia necesaria dependiendo del ciclo de trabajo y el tercero modificará el ciclo de trabajo para lograr el movimiento del servomotor. Todos estos procesos serán ejecutados en procesos diferentes, los cuales son dependientes unos de otros.

El primer proceso llamado Contador se encarga de realizar un conteo hasta la constante de conteo máximo que tiene el valor del divisor de frecuencia del Pulso, esto se realiza para tener el número de ciclos necesarios para completar la frecuencia del PWM (recordar los 50 Hz de la teoría) y el contador es utilizado para los siguientes procesos, una vez llega al número de pasos de la frecuencia de 50 Hz realiza un reinicio que a su vez reinicia el resto de los procesos.

Figura 80.

Proceso del contador del código del PWM del servomotor

```
45    Contador : process (clk)
46    begin
47       if (rising_edge(clk)) then
48          if (rst = '1') then
49             cont <= 0;
50          else
51             if cont < cont_max then
52                cont <= cont + 1;
53             else
54                cont <= 0;
55             end if;
56          end if;
57       end if;
58    end process;
```

Nota. En la figura se puede observar el segmento de código que brinda el conteo utilizado para la generación de la señal PWM para el Servomotor. Elaboración propia, realizado con ISE Design Suite 14.

El segundo proceso llamado PWMProceso es el encargado de dar las salidas en alto y bajo del PWM. El código puede ser observado en la Figura 81 y este funciona gracias al valor del contador del primer proceso, la señal cont debe ser menor al ciclo de trabajo para que la salida sea en alto, esto se realiza de esta manera para que pueda coincidir con los periodos en alto que debe tener la señal para mover el eje del motor a ciertos ángulos, es la señal

duty_cycle la que brinda la posibilidad de hacer las variaciones de los ángulos. La señal duty_cycle es modificada dentro del tercer proceso y este será explicado a detalle en el siguiente párrafo.

Figura 81.

Proceso del generador de PWM del código del PWM del servomotor

```
62    PWMProceso : process (clk)
63    begin
64       if (rising_edge(clk)) then
65          if (rst = '1') then
66             spwm <= '0';
67          else
68             spwm <= '0';
69             if cont < duty_cycle then
70                spwm <= '1';
71             end if;
72          end if;
73       end if;
74    end process;
```

Nota. En la figura se puede observar el segmento de código que genera la señal en alto y bajo del PWM para el Servomotor. Elaboración propia, realizado con ISE Design Suite 14.

El último proceso a utilizar es el que marca el ciclo de trabajo y el nombre de dicho proceso es Ciclotrabajo. El proceso realiza la asignación de valores a la señal duty_cycle, esta señal es en donde se almacenan los valores de los ciclos necesarios para alcanzar los ángulos deseados en el eje del servomotor. En el funcionamiento normal del duty_cycle se almacena dentro de la señal la multiplicación del valor de posición, que son los valores del seno obtenido del módulo de la señal senoidal, con la constante ciclos_por_paso. La multiplicación es realizada para obtener la cantidad de ciclos que debe contar la FPGA para cubrir el rango de movimiento del eje y debe ser mencionado que es variable gracias a la constante posición que tiene almacenados distintos resultados de los valores del seno, gracias a esta

acción es posible realizar un recorrido constante de 0 ° a 180 °. Adicional a la multiplicación se debe sumar a la señal la constante de los ciclos que deben ejecutarse para alcanzar la mínima posición, esto debe realizarse de esta forma para que la FPGA tenga un punto de referencia inicial que represente al ángulo mínimo al momento en que la constante de posición cambie de valor a 0.

Figura 82.

Proceso del ciclo de trabajo del código del PWM del servomotor

```
79    CicloTrabajo : process (clk)
80    begin
81       if (rising_edge(clk)) then
82          if (rst = '1') then
83             duty_cycle <= conteo_min;
84          else
85             duty_cycle <= posicion * ciclos_por_paso + conteo_min;
86          end if;
87       end if;
88    end process;
```

Nota. En la figura se puede observar el segmento de código que modifica los ciclos de trabajo del PWM para el Servomotor. Elaboración propia, realizado con ISE Design Suite 14.

Con los procesos mencionados es posible realizar la señal PWM que utiliza el servomotor, como se ha podido observar el módulo tiene un funcionamiento interno independiente del resto de módulos con la excepción de una señal externa que afecta el movimiento del ángulo. La señal a la que se hace referencia fue la presentada en el tercer proceso, la señal posición. La señal será explicada a mayor detalle en el siguiente capítulo y se ilustrará la importancia de la variación de la señal senoidal dentro de la misma, con esto se espera dar una mejor explicación del módulo y lograr expresar de mejor forma el motivo de cada señal presentada en este capítulo.

4.3.4. Módulo senoidal del Servomotor

En el módulo anterior se indicó que una de las entradas tenía que tener una conexión directa con un módulo que genere una señal senoidal, esto se realiza de esta forma para poder darle al módulo de servomotor valores que permitan el recorrido de la señal y con esto se lograría el movimiento característico del eje del servomotor.

Para la construcción del módulo de la señal senoidal solo se depende de dos entradas y una salida, además de dos señales genéricas. Las señales necesarias para la construcción del módulo se detallan a continuación:

- angbit: esta es una constante a la cual se le asigna un valor de bits para dar una resolución de valores para las señales, en el caso del ejemplo presentado el valor a utilizar es de 8 bits.
- datobit: esta constante es idéntica a angbit la única diferencia es que esta constante tendrá un espacio más disponible, esto se hace de esta forma para poder almacenar los datos en la rom que se creara en módulo.
- clk: es la señal del reloj general, este será uno de los puertos de entrada del módulo.
- ang: es la señal en donde se almacena el contador utilizado, el contador proviene de un módulo secundario, la señal es concatenada y posterior a esto es asignada a la señal ang para que adentro del módulo se pueda utilizar para generar los distintos valores de seno.
- Seno: es la señal que entregara los valores de la rom que tiene el cálculo de cada uno de los valores senoidales que se calculan.

Figura 83.

Declaración de las señales del módulo generador de la señal senoidal

```
 6  entity sseno is
 7      Generic (
 8           angbit  : integer range 1 to 30;
 9           datobit : integer range 1 to 31);
10      Port ( clk : in  STD_LOGIC;
11             ang : in unsigned(angbit - 1 downto 0);
12             seno : out  unsigned(datobit - 1 downto 0));
13  end sseno;
```

Nota. En la figura se puede observar el segmento de código en donde se declaran las señales a utilizar en el módulo generador de la señal senoidal. Elaboración propia, realizado con ISE Design Suite 14.

Una vez se han creado las señales de los puertos se puede crear la memoria rom utilizando tipos y subtipos para la misma, esto se debe realizar dentro de la arquitectura del programa y antes del inicio de estas. Se emplean arreglos de señales que asignen un rango útil de uso que serán dependientes de los puertos iniciales angbit y datobit, es necesario que sea de esta forma para tener la resolución de ocho bits que será asignada a esta en el módulo principal. La declaración de las señales se puede observar en la siguiente figura.

Figura 84.

Declaración de las señales para la rom de la señal senoidal

```
16      subtype angrango is integer range 0 to 2**angbit - 1;
17      type rom_type is array (angrango) of unsigned(datobit - 1 downto 0);
```

Nota. En la figura se puede observar el segmento de código en donde se declaran las señales a utilizar en el módulo generador de la señal senoidal. Elaboración propia, realizado con ISE Design Suite 14.

El siguiente paso es crear una función con la que se puedan llenar los valores de la memoria ROM con los valores del seno. Se utilizarán tres señales para guardar los valores que se obtengan de la función, dichas señales serán variables las cuales son:

- rom_v: esta será una señal idéntica a la señal rom_type que fue declarada antes de la función, tendrá la tarea de almacenar todos los valores del seno en un formato unsigned para que pueda ser utilizado por la posición del módulo del PWM.
- Ángulo: esta será la señal en la cual se almacene el ángulo que está siendo representado en el recorrido, se almacena en una variable por separado para que el cálculo del seno sea más sencillo al utilizar esta señal como el ángulo.
- senoescala: es la señal en la cual se almacena el valor del seno una vez se calcula con la resolución y el ángulo antes calculado, este valor será almacenado en rom_v.

Una vez se crean las variables inician los procesos internos de la función, estos consisten en llenar los vectores que han sido mencionados por medio de un ciclo for que tendrá como delimitador al subtipo angrango ya que es el valor dentro del módulo que marca la resolución deseada del generador de la señal senoidal, esto crea múltiples valores senoidales que cumplirán con la resolución de ocho bits. La función completa se puede encontrar en la siguiente imagen.

Figura 85.

Función que crea la cadena de señales senoidales del módulo

```
22    function init_rom return rom_type is
23       variable rom_v : rom_type;
24       variable angulo : real;
25       variable senoescala: real;
26    begin
27       for i in angrango loop
28          angulo := real(i) *((2.0*MATH_PI)/ 2.0**angbit);
29          senoescala := (1.0 + sin(angulo))*(2.0**datobit - 1.0)/2.0;
30          rom_v(i) := to_unsigned(integer(round(senoescala)), datobit);
31       end loop;
32       return rom_v;
33    end init_rom;
```

Nota. En la figura se puede observar el segmento de código en donde se realiza la creación de las señales senoidales utilizadas en el generador. Elaboración propia, realizado con ISE Design Suite 14.

Una vez se tiene la función que crea la cadena de senos se puede iniciar con la estructura principal del programa. La estructura será un simple proceso con un if que evaluará los cambios de la señal del reloj. Deberá entregar la señal de la función creada con anterioridad a la salida seno del módulo, esto será realizado por medio de una constante similar a rom_type que almacenará la función init_rom (ver línea 35 del código) y será esta constante la que será utilizada para invocar la función de la cadena. La constante rom recibirá la señal de entrada ang que tendrá enlazada la salida del módulo contador y esto hará variar progresivamente los valores de la función senoidal, haciendo que la memoria rom pueda almacenar distintos valores senoidales que cumplan con una secuencia previamente establecida. La constante rom será almacenada directamente a la señal de salida seno que conectará con el módulo visto en el anterior capitulo por medio de un enlace en el módulo principal. De esta forma finaliza el código generador de señales senoidales, en la siguiente figura se puede observar con mejor detalle el ultimo fragmento del código descrito.

Figura 86.

Función principal del módulo generador de senoidales

```
22    function init_rom return rom_type is
23       variable rom_v : rom_type;
24       variable angulo : real;
25       variable senoescala: real;
26    begin
27       for i in angrango loop
28          angulo := real(i) *((2.0*MATH_PI)/ 2.0**angbit);
29          senoescala := (1.0 + sin(angulo))*(2.0**datobit - 1.0)/2.0;
30          rom_v(i) := to_unsigned(integer(round(senoescala)), datobit);
31       end loop;
32       return rom_v;
33    end init_rom;
```

Nota. En la figura se puede observar el segmento de código en donde se realiza la creación de las señales senoidales utilizadas en el generador. Elaboración propia, realizado con ISE Design Suite 14.

4.3.5. Módulo contador y reset del Servomotor

Los módulos del contador y del reset para la aplicación de mover el eje del servomotor son sencillos comparados con los módulos del generador de la señal senoidal y el generador de la señal PWM, sin embargo, estos son igual de importantes ya que darán un mejor control y estabilidad al programa general.

El módulo del contador consiste en un simple programa que utiliza tres entradas y una salida, las señales que utiliza son:

- Conteo_bits: esta es una constante que sirve como delimitador para la salida del módulo.
- clk: esta es la señal del reloj de la FPGA.
- rst: esta es la señal de reset del módulo principal.

- Habilitar: esta es la señal que indica si se puede utilizar o no el módulo, para la aplicación que se creará se tendrá en un valor constante de ‘1’.
- Cont: esta es la señal de salida del contador, esta señal es la que se enlaza con el módulo del generador de señal senoidal.

Figura 87.

Librerías y señales del módulo contador del programa del servomotor

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  use IEEE.math_real.all;
5  entity contador is
6      Generic
7           ( conteo_bits : integer);
8      Port ( clk : in  STD_LOGIC;
9             rst : in  STD_LOGIC;
10            habilitar :  STD_LOGIC;
11            cont : out  unsigned(conteo_bits - 1 downto 0));
12 end contador;
```

Nota. En la figura se puede observar las librerías utilizadas y las señales creadas para el módulo contador del programa del servomotor. Elaboración propia, realizado con ISE Design Suite 14.

Una vez se definen las señales a utilizar lo único que se hace es crear una señal de conteo auxiliar en la que se almacenaran todos los cambios y que estará enlazada con la salida del contador. Dentro del módulo se debe crear un proceso que mida cada pulso de reloj y que realice el conteo siempre y cuando no se pulse el botón de reset y se tenga la señal habilitar en alto. Como se puede observar es una sintaxis simple en la cual el valor más importante es la salida del módulo por su conexión con el generador de la señal senoidal. La arquitectura del módulo se puede observar de forma más detallada en la figura mostrada a continuación.

Figura 88.

Arquitectura del módulo contador del programa del servomotor

```
13  architecture Behavioral of contador is
14     signal cont_i : unsigned(cont'range);
15  begin
16     cont <= cont_i;
17     Contador: process(clk)
18     begin
19        if (rising_edge(clk)) then
20           if (rst = '1') then
21              cont_i <= (others => '0');
22           else
23              if (habilitar = '1') then
24                 cont_i <= cont_i + 1;
25              end if;
26           end if;
27        end if;
28     end process;
29  end Behavioral;
```

Nota. En la figura se puede observar la arquitectura del módulo contador del programa del servomotor. Elaboración propia, realizado con ISE Design Suite 14.

El módulo reset consiste en enviar una señal a todos los módulos que les indique el cambio de los valores a los valores iniciales. Para lograr dicha tarea se hace uso de tres tipos de señales, las cuales son:

- clk: esta es la señal del reloj de la FPGA.
- Btn_rst: esta es la señal que está conectada al botón de la FPGA que dará inicio al módulo de reset.
- Rst: esta es la señal de salida del módulo, esta es la que se debe conectar en el módulo Top con el resto de los módulos.

Figura 89.

Librerías y señales del módulo reset del programa del servomotor

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity reset is
6     port (
7       clk : in STD_LOGIC;
8       btn_rst : in STD_LOGIC;
9       rst : out STD_LOGIC);
10  end reset;
```

Nota. En la figura se puede observar las librerías utilizadas y las señales creadas para el módulo reset del programa del servomotor. Elaboración propia, realizado con ISE Design Suite 14.

Con las señales anteriores se puede realizar un programa que no utilice retardos o contadores extras para evitar rebotes de botón, para realizar este proceso es necesario programar dos procesos, el primero dependiente del reloj y el segundo dependiente de la variable interna del proceso de reloj. Para el proceso del reloj bastara con realizar un if que mida cada vez que existe un cambio de reloj y concatenara el valor del botón externo en una señal de cuatro bits, esto se hace de esta forma para poder registrar las pulsaciones del botón y evitar de esta forma los errores por una pulsación errónea y evitar así un efecto rebote de los botones. En el segundo proceso se realizara una comparación de la señal que almaceno la concatenación y se evaluara si en esta señal existe una cada de valores igual a 1, si se cumple este caso significa que el botón no fue pulsado y no es necesario realizar un reset, si el botón llega a ser pulsado los valores no serán iguales y esto provocara que se envié una señal de reset a todos los módulos externos logrando de esta forma regresar al estado inicial a todas las variables de los distintos programas y

módulos Estas instrucciones se realizan dentro de la arquitectura del programa, es un código simple y puede observarse en la imagen presentada a continuación.

Figura 90.

Arquitectura del módulo reset del programa del servomotor

```
12  architecture Behavioral of reset is
13   signal concat : std_logic_vector(3 downto 0);
14     Concatenar : process(clk)
15     begin
16        if (rising_edge(clk)) then
17           concat <= concat(concat'high - 1 downto 0) & btn_rst;
18        end if;
19     end process;
20     resetear : process(concat)
21        constant aux : std_logic_vector(concat'range) := (others => '1');
22     begin
23        if (concat = aux) then
24           rst <= '0';
25        else
26           rst <= '1';
27        end if;
28     end process;
29
30  end Behavioral;
```

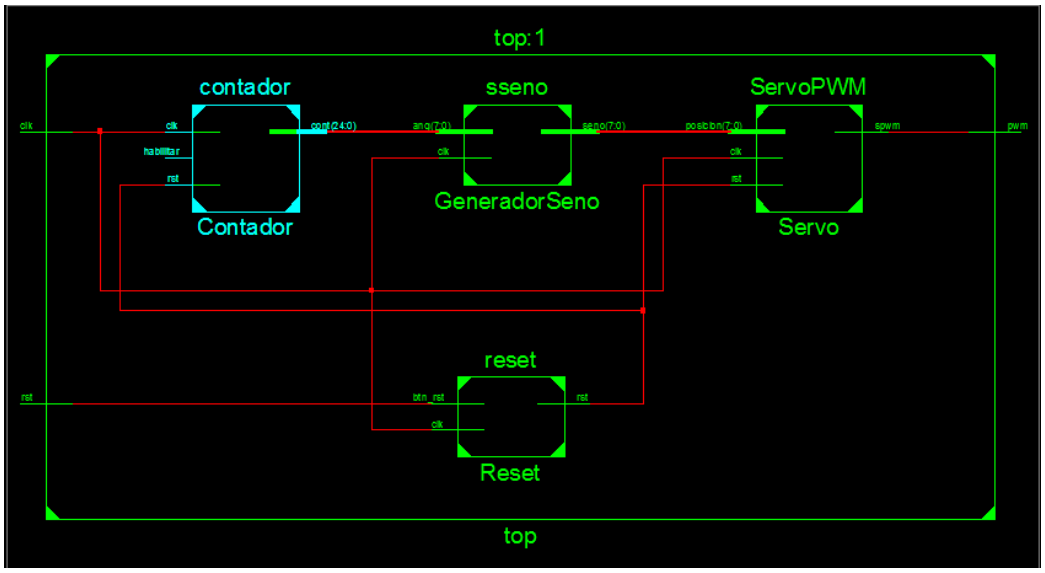
Nota. En la figura se puede observar la arquitectura del módulo reset del programa del servomotor. Elaboración propia, realizado con ISE Design Suite 14.

4.3.6. Módulo top y asignación de salidas de la FPGA al Servomotor

Una vez se han creado los módulos que integran las principales funciones del servomotor se debe realizar una instanciación de todos estos módulos en un módulo principal. Debido a la arquitectura utilizada se deben declarar dentro del módulo Top todas las constantes mencionadas al momento de la creación del módulo. En la siguiente ilustración se puede ver una muestra de cómo debe ser la interconexión entre los módulos creados.

Figura 91.

Interconexiones de los módulos del programa del servomotor



Nota. En la figura se puede observar las interconexiones de los módulos del programa del servomotor. Elaboración propia, realizado con ISE Design Suite 14.

Tal como se ha podido observar en el diagrama de interconexiones el módulo principal solo tiene tres señales, dos de estas señales son entradas y la otra es una salida, estas señales corresponden al reloj, al botón del reset y la salida de PWM que se conectara con el servomotor. Adicional a esto deben ser declaradas ciertas constantes que utilizan los módulos internos, esto debe ser realizado antes del inicio de la arquitectura. La muestra del segmento de código que realiza las declaraciones anteriores se muestra a continuación.

Figura 92.

Señales y constantes del módulo principal del programa del servomotor

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.numeric_std.all;
4  entity top is
5      Port ( clk : in  STD_LOGIC;
6             rst : in  STD_LOGIC;
7             pwm : out  STD_LOGIC);
8  end top;
9  architecture Behavioral of top is
10    constant clk_hz :real := 12.0e6; --Frecuencia de la Elbert V2
11    constant Pulso : real:= 50.0; --Este pulso es el utilizado en el PWM para fraccionar la frecuencia del reloj
12    constant Pulsomin : real:= 500.00; -- Frecuencia de pulso para la posicion minima.
13    constant Pulsomax : real:= 1000.00; -- Frecuencia de pulso para la posicion maxima.
14    constant Bitpaso : positive := 8; --Esto no podria ser eliminado
15    constant conteo : positive := 2**Bitpaso; --Esto no podria ser eliminado
16
17 --Constantes del conteo
18    constant contbit : integer := 25;
19    signal cont : unsigned(contbit - 1 downto 0);
20
21    signal rstb : std_logic; --Usado para obtener la salida del reset.
22    signal posicion : integer range 0 to conteo - 1;
23    signal rom_ang : unsigned(Bitpaso - 1 downto 0);
24    signal rom_sseno : unsigned(Bitpaso - 1 downto 0);
```

Nota. En la figura se puede observar las señales y constantes del módulo principal del programa del servomotor. Elaboración propia, realizado con ISE Design Suite 14.

Las instanciaciones de los submódulos se realizan de la misma forma que un programa normal, pero dentro del módulo top existen dos señales que deben ser consideradas por la manera en que varían y alteran los valores de la posición de los submódulos, estas señales son posicion y rom_ang.

La señal de posición es la que realiza la conversión de la salida del submódulo de seno y sirve para que pueda ser enviada la señal al siguiente módulo que sería el generador de la señal PWM. Cabe resaltar que se debe realizar dicha conversión afuera de la instanciación para que VHDL no considere un error secuencial al momento de realizar la sintetización. Los segmentos de código que involucran esta acción son las líneas 26 y 70.

En el caso de la señal rom_ang consiste en una señal que indicara el ángulo a utilizar en las operaciones de los valores senoidales que controlan la posición del eje del servomotor. Lo que se realiza dentro de la señal es una

concatenación de los valores del contador, estos deben ser llenados utilizando los valores del contador y la resolución de 8 bits que se marca en la línea 17, esto se realiza de esta forma debido a la facilidad de desplazamiento numérico que brinda el contador y al utilizar como el delimitador la resolución se logra una señal que puede cambiar de estados dentro de un rango establecido y esto ayuda al módulo del número senoidal a entregar varios valores que dan siempre el mismo recorrido. En las siguientes figuras se pueden observar el resto del código que realiza la instanciación de los módulos.

Figura 93.

Instanciación de los módulos Reset y generador del PWM

```
25  begin
26     posicion <= to_integer(rom_sseno);
27     rom_ang <= cont(cont'left downto cont'left - Bitpaso + 1);
28     Reset :
29        entity work.reset
30        port map(
31           clk => clk,
32           btn_rst => rst,
33           rst => rstb
34        );
35     Servo :
36        entity work.ServoPWM
37        generic map(
38           clk_hz => clk_hz,
39           Pulso => Pulso,
40           Pulsomin => Pulsomin,
41           Pulsomax => Pulsomax,
42           conteo => conteo
43           )
44        port map(
45           clk => clk,
46           rst => rstb,
47           posicion => posicion,
48           spwm => pwm
49           );
```

Nota. En la figura se puede observar la asignación de valores a las señales especiales y la instanciación de los módulos Reset y Generador del PWM en el orden correcto. Elaboración propia, realizado con ISE Design Suite 14.

Figura 94.

Instanciación del módulo contador y del módulo generador de la señal senoidal

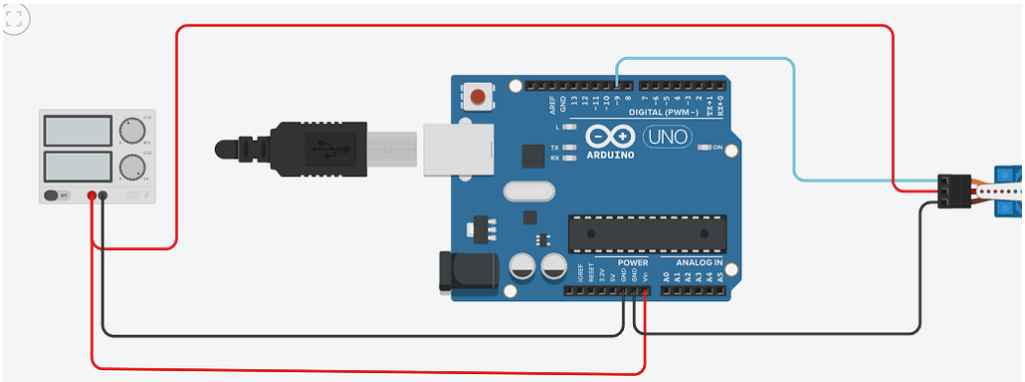
```
50     Contador :
51        entity work.contador
52        generic map(
53           conteo_bits => contbit
54           )
55        port map(
56           clk => clk,
57           rst => rstb,
58           habilitar => '1',
59           cont => cont
60           );
61     GeneradorSeno :
62        entity work.sseno
63        generic map(
64           angbit => Bitpaso,
65           datobit => Bitpaso
66           )
67        port map(
68           clk => clk,
69           ang => rom_ang,
70           seno => rom_sseno
71           );
72
73  end Behavioral;
```

Nota. En la figura se puede observar la instanciación de los módulos de contador y del generador de la señal senoidal. Elaboración propia, realizado con ISE Design Suite 14.

Una vez se han realizado las instanciaciones y asignaciones de los valores constantes el paso final es asignar las tres señales a la tarjeta Elbert V2. Es recomendable asignar la señal del PWM al pin 1 del cabezal P6, esto se debe a que dicho pin según la hoja técnica pertenece al grupo de pines de tipo LHCLK que significa Left-Half Clock lo que significa según el diseñador que es un pin con salida dedicada de reloj y esto hace el pin más adecuado para aplicaciones de pulsos temporizados que pueden abarcar los usos del PWM que se necesitan. Para el resto de las conexiones pueden utilizarse el GND y VCC de la Elbert V2, aunque por el tipo de aplicación es más recomendable hacer uso de una fuente externa para tener una mayor entrega de corriente siempre que se aplique la unión común de tierras para lograrlo, en la Figura 95 se puede observar un diagrama simple que ilustra la acción de unir una tierra común de dos fuentes.

Figura 95.

Ejemplo de unión de tierras utilizando múltiples fuentes con un Arduino Uno



Nota. En la figura se puede observar un ejemplo de la forma en que se debe conectar una fuente externa a un microcontrolador para entregarle una mayor corriente al servomotor. Obtenido de Arduino Forum. *Grounding an Arduino Project: External Power Source*. (https://forum.arduino.cc/t/grounding-an-arduino-project-external-power-source/658642/), consultado el 14 de julio de 2023. De dominio público.

Como se pudo observar el crear un módulo que pueda girar el eje de un servomotor es un proceso un poco largo pero sencillo, se puede utilizar para otro tipo de aplicaciones con el simple hecho de alterar los rangos propuestos en las líneas 12 y 13 del módulo Top, el resto del programa puede funcionar para servomotores comerciales y si se desea realizar una aplicación un poco más orientada a un módulo industrial bastara con realizar las modificaciones en el resto de las constantes.

4.4. Movimiento de motor Stepper usando VHDL

Un motor Stepper es un tipo de motor especial que funciona accionando sus bobinas internas en un orden definido para lograr la rotación del eje de dicho motor. El motor Stepper también posee el nombre de motor a pasos, el

código para lograr el movimiento de este en VHDL es mucho más simple y sencillo de implementar que el servomotor debido al funcionamiento del mismo, en los siguientes párrafos se explicara más a detalle el comportamiento del motor y sus aplicaciones en VHDL.

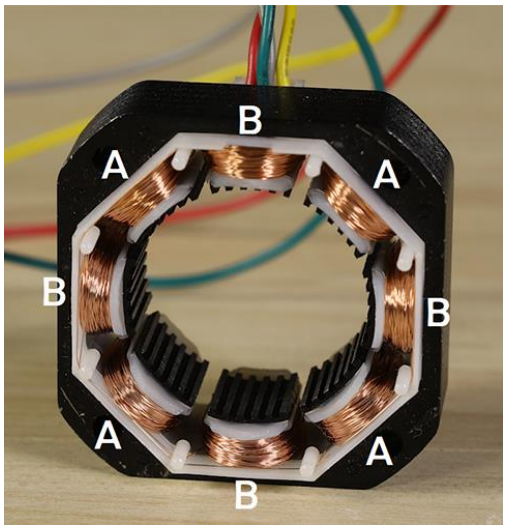
4.4.1. Funcionamiento de un motor Stepper

Tal como fue indicado al inicio del capítulo el motor stepper tiene una construcción de múltiples bobinas internas que al ser accionadas en una determinada secuencia logra el movimiento del eje del motor. Dicha construcción puede observarse en la siguiente Figura 96.

Al observar la construcción interna en la figura se puede notar las distintas bobinas que conforman el motor. La construcción de este tipo de motor hace que el funcionamiento del mismo se base en energizar las bobinas una por una para que el rotor se alinee a la bobina energizada provocando así un movimiento de un único paso. La forma en que se logra realizar un movimiento continuo del rotor es apagando la bobina que se energiza primero y acto seguido se energiza la segunda bobina, esto vuelve a provocar una alineación nuevamente, pero esta vez dando un segundo paso gracias al cambio de bobina, al realizar esta acción en repetidas ocasiones y en el orden correcto lo que se consigue es que el rotor pueda alinearse con todas las bobinas internas del estator y el eje rote. La acción de la rotación puede ser controlada y solo funcionara si se cumplen ciertos criterios dependiendo siempre de lo que indique el fabricante del motor a utilizar.

Figura 96.

Imagen de la estructura interna de un motor stepper



Nota. En la figura se puede observar una imagen que muestra las partes internas de un motor stepper. Obtenido de DigiKey. *The Basics of Stepper Motors*. (https://www.digikey.com.mx/es/blog/the-basics-of-stepper-motors), consultado el 21 de julio de 2023. De dominio público.

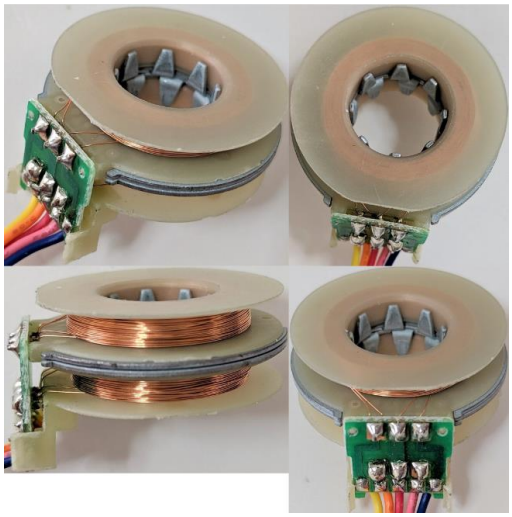
El motor que se utilizara dentro del proyecto es el motor stepper 28BYJ-48 de 5V, la construcción de este motor es un poco más sencilla y barata, además que existe una gran disponibilidad de estos al ser un componente de uso común y es útil para entender el funcionamiento de los motores stepper.

La construcción del motor 28BYJ-48 se basa en dos bobinas que tienen la misma cantidad de vueltas, a esta configuración se le conoce como motor bipolar y entre las bobinas se encuentran separadores metálicos que permiten la polarización del rotor, la estructura se puede observar de mejor forma en la siguiente figura. Se debe hacer mención que cada una de las bobinas de forma

interna tiene una separación que crea algo similar a dos bobinas por cada separador.

Figura 97.

Imagen de las bobinas internas del motor 28BYJ-48

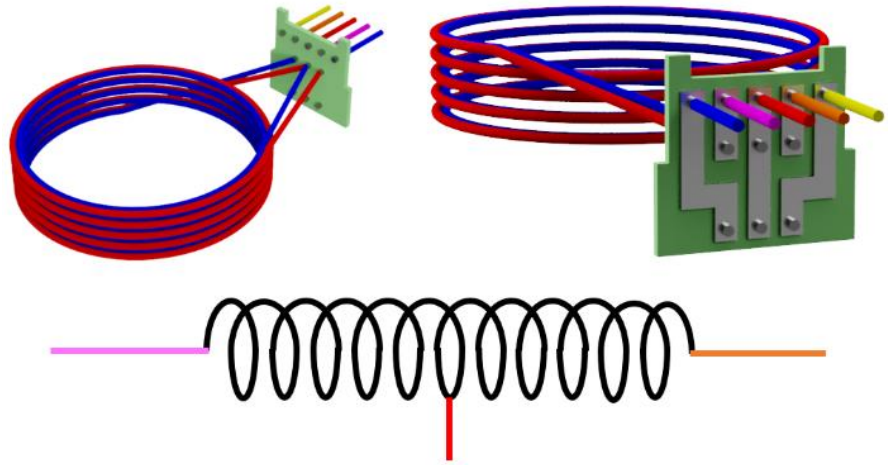


Nota. En la figura se puede observar la estructura interna de las bobinas y los separadores de un motor stepper 28BYJ-48. Obtenido de Cookierobotics. *28BYJ-48 Structure*. (https://cookierobotics.com/042/), consultado el 21 de julio de 2023. De dominio público.

En la Figura 97 se puede observar dentro de las bobinas una serie de ocho dientes por cada capa, estos dientes son los que se polarizan por la ley de ampere al rodear la bobina y es lo que permite (de forma muy simplificada) la rotación del rotor. Como se mencionó con anterioridad, dentro de cada una de las bobinas existe una separación, dicha separación crea el efecto de dos bobinas unidas en un nodo en común, por lo tanto, el motor posee en realidad dos bobinas en el eje superior y dos bobinas en el eje inferior creando de esta forma un motor de cuatro fases. En la figura 98 se puede observar con mayor detalle la construcción interna del embobinado mencionado.

Figura 98.

Imagen representativa de la parte interna de las bobinas del motor 28BYJ-48



Nota. En la figura se puede observar la división que crea las dos bobinas dentro de la bobina principal del motor stepper 28BYJ-48. Obtenido de Cookierobotics. *28BYJ-48 Structure*. (https://cookierobotics.com/042/), consultado el 21 de julio de 2023. De dominio público.

Para que el motor funcione de forma adecuada debe obedecer una cierta secuencia de acción de bobinas con cierta periodicidad para lograr el paso a paso. Según la hoja de datos del motor 28BYJ-48 se necesita de una demora entre cambios mínima de 10 ms para que el motor tenga tiempo suficiente de alinearse con la siguiente fase, de utilizar una demora menor el motor no podrá realizar los movimientos. La secuencia dependerá de la aplicación del motor stepper, en general existen tres tipos de configuraciones las cuales serán explicadas a continuación.

En la Tabla 2 se puede observar el orden en el que deben ser accionadas las bobinas para lograr el funcionamiento básico del motor stepper, en esta configuración simplemente basta con encender una a una las bobinas y apagar el resto respetando la periodicidad de los cambios mencionada en el

párrafo anterior. Esta configuración tiene el nombre de paso completo simple y tal como su nombre lo indica es la más simple de todas, permite el movimiento del motor y se puede replicar de una forma muy sencilla en VHDL, sin embargo, es la configuración que brinda la fuerza de torque menor de todas las configuraciones y es la que menor consumo posee.

Tabla 2.

Secuencia paso a paso de la configuración de paso completo simple

Paso	Bobina 1	Bobina 2	Bobina 3	Bobina 4
Paso 1	1	0	0	0
Paso 2	0	1	0	0
Paso 3	0	0	1	0
Paso 4	0	0	0	1

Nota. La secuencia de paso completo simple consiste solamente en encender una por una las bobinas para lograr el movimiento del eje del motor stepper. Elaboración propia, realizado con Excel.

La secuencia de paso completo simple es útil para aplicaciones básicas y de poco consumo, si se requiere una configuración que brinde un mayor torque se puede utilizar la secuencia llamada paso completo de doble bobina. Este método consiste en encender por parejas las bobinas, con esto se logra que se mantengan energizadas y brinden una mayor fuerza de torque en el eje, la representación de esta secuencia se puede observar en la Tabla 3. Tiene los mismos cuatro pasos que el método de paso simple, la única diferencia es que se mantienen siempre dos bobinas energizadas en cada uno de los pasos, haciendo transiciones de pares de bobina al momento de dar los pasos. Esta aplicación brinda el máximo torque en los motores stepper y a su vez consume una mayor corriente que el método básico.

Tabla 3.

Secuencia paso a paso de la configuración de paso completo de doble bobina

Paso	Bobina 1	Bobina 2	Bobina 3	Bobina 4
Paso 1	1	1	0	0
Paso 2	0	1	1	0
Paso 3	0	0	1	1
Paso 4	1	0	0	1

Nota. La secuencia de paso completo de doble bobina consiste en encender las bobinas formando pares para lograr un mayor torque en el eje del motor stepper. Elaboración propia, realizado con Excel.

La configuración anterior es útil si se desea un mayor torque, pero tiene una pequeña desventaja al no dar pasos tan precisos si se requiere para ciertas aplicaciones, por este motivo existe un tercer método que combina precisión y torque con la pequeña desventaja que es el que más corriente consume, este método se llama configuración de medio paso. Esta secuencia consiste en ocho pasos, es similar a la secuencia de paso completo de doble bobina con la principal diferencia que entre los cambios de las bobinas se energizan dos bobinas a la vez entre las transiciones, esto provoca que en la transición no se apague directamente la bobina que se energiza inicialmente y esto da una mayor estabilidad al eje al no soltar el rotor hasta que la segunda bobina es energizada también. Para que la explicación tenga un mayor sentido se puede observar la secuencia en la Tabla 4, en los pasos dos, cuatro, seis y ocho se están energizadas dos bobinas y estas son siempre una combinación de la bobina anterior y bobina siguiente, por este motivo es que en esta configuración se consigue un mayor torque.

Tabla 4.

Secuencia paso a paso de la configuración de medio paso

Paso	Bobina 1	Bobina 2	Bobina 3	Bobina 4
Paso 1	1	0	0	0
Paso 2	1	1	0	0
Paso 3	0	1	0	0
Paso 4	0	1	1	0
Paso 5	0	0	1	0
Paso 6	0	0	1	1
Paso 7	0	0	0	1
Paso 8	1	0	0	1

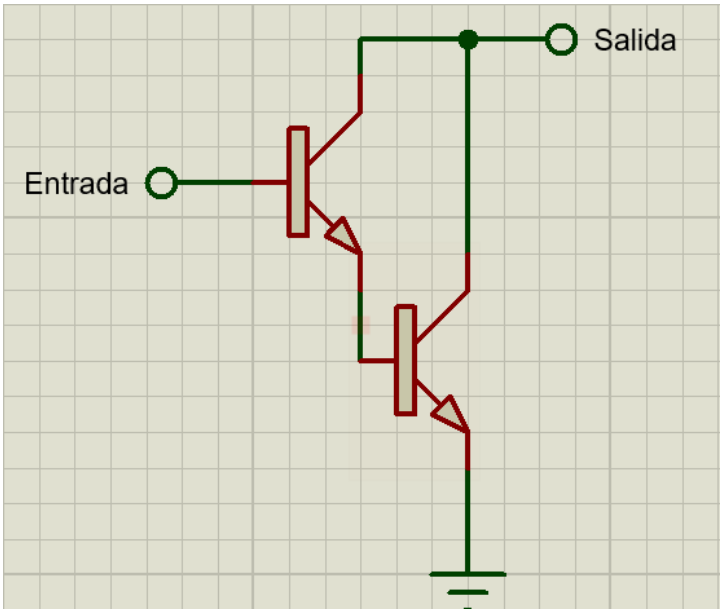
Nota. La secuencia de medio paso consiste en energizar dos bobinas entre los cambios de pasos, esto logra una mayor estabilidad y torque del motor stepper utilizando una mayor potencia para su uso. Elaboración propia, realizado con Excel.

Los motores 28BYJ-48 tienen un consumo de corriente de aproximadamente 55 mA, esto es importante ya que la entrega de corriente de un 1 lógico en una salida TTL estándar es de 0.4 mA y a veces por diseño puede entregar alrededor de 1 mA, es evidente la diferencia de corriente y por esto mismo un pin de la FPGA no puede realizar la polarización de los inductores del motor por su propia cuenta. Para solucionar el problema de entrega de corriente al motor se creó un módulo a partir de un circuito integrado que es capaz de tomar las salidas de baja corriente TTL y transformarlas en salidas con una mayor potencia que cumple con las exigencias de corriente del motor, el módulo se llama ULN2003.

El módulo ULN2003 posee siete compuertas que tienen dos transistores en una configuración Darlington, dicha configuración permite incrementar la potencia de salida de la pareja de transistores al incrementar de forma exponencial la ganancia de los mismos y esto vuelve a esta configuración ideal para su aplicación en la activación de motores.

Figura 99.

Circuito de transistores en arreglo Darlington



Nota. En la figura se puede observar el circuito esquemático de una configuración de dos transistores en modo Darlington, esta configuración garantiza una ganancia exponencial y por lo tanto una corriente de salida mayor a la de entrada. Elaboración propia, realizado con Proteus 8 Professional.

El módulo posee un arreglo como el de la Figura 99 en cada una de sus compuertas, tendrá una corriente máxima de aproximadamente 500 mA lo que multiplica por una enorme cantidad la corriente de entrada que brindan los puertos de la FPGA. Adicional al uso del módulo ULN2003 el módulo consta

de cuatro pines de entrada para que sean conectados los puertos de salida de la FPGA o microcontrolador a utilizar, tiene cuatro leds adicionales que muestran el estado de dichas entradas para dar una referencia visual y por último cuenta con dos pines configurados para soportar salidas de 5 o 12 V dependiendo de la exigencia del motor que se conecte. En la Figura 100 se puede observar con mayor detalle el módulo que se utilizara para las pruebas del proyecto.

Figura 100.

Módulo para controlar motores stepper utilizando el integrado ULN2003

Nota. En la figura se puede observar la forma que tiene el módulo controlador con el integrado ULN2003 que es muy popular en la mayoría de las tiendas de electrónica por su facilidad de uso y bajo costo. Obtenido de La Electrónica. *MÓDULO CONTROLADOR DE STEPPER ULN2003*. (https://laelectronica.com.gt/modulo-uln2003-controlador-de-motor-stepper), consultado el 24 de julio de 2023. De dominio público.

El motor stepper abre las puertas a múltiples aplicaciones que requieren precisión y un gran torque para ejecutarlas, posee un funcionamiento mucho más simple que el de un servomotor, aunque consuma una mayor cantidad de

memoria, además que necesita de un módulo externo para que pueda funcionar un microcontrolador o una FPGA para controlarlo. En los párrafos que son presentados a continuación se explicaran algunas aplicaciones en VHDL que pueden servir como base para utilizar los programas en algún proyecto que requiera el uso de estos.

4.4.2. Módulo de movimiento básico del motor stepper.

El movimiento de un motor stepper a nivel de programación consta únicamente de establecer la secuencia correcta en los pines que alimentaran el módulo que conectan al motor stepper, debido a esto para lograr el movimiento básico del motor se propone utilizar el método de paso completo simple que consiste en encender en el orden correcto pin por pin para lograr el movimiento del motor. Inicialmente se propone el uso del reloj y de las salidas, para lograr el movimiento básico del motor no se necesita nada más, se utilizan entonces las entradas y salidas de la siguiente forma:

Figura 101.

Señales de entrada y salida de la entidad del módulo principal

```
6   entity main is
7       Port ( clk : in  STD_LOGIC;
8              motor : out  STD_LOGIC_VECTOR (3 downto 0));
9   end main;
```

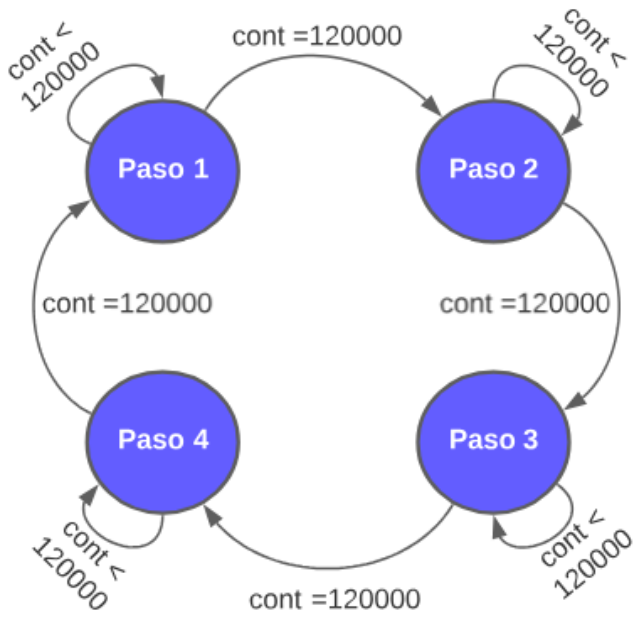
Nota. En la figura se puede observar la creación de las señales que serán utilizadas para el proyecto que realizara el movimiento del eje del motor. Elaboración propia, realizado con ISE Design Suite 14.

Una vez se tienen las señales que serán asignadas a los puertos de entrada y salida se debe realizar el análisis lógico que debe tener el programa.

El movimiento del motor requiere que se enciendan las casillas del vector motor una por una, además que debe cumplir con una espera mínima de 10 ms. La solución que se propone es realizar una máquina de estados similar a la que se trabajó en el ejemplo del semáforo introduciendo la variación de la espera cada 10 ms. La máquina de estados se puede observar con un mayor detalle en la Figura 102.

Figura 102.

Programa que realiza el movimiento básico del eje de un motor stepper



Nota. En la figura se puede observar la máquina de estados que será programa en VHDL. Elaboración propia, realizado con Lucidchart.

La máquina de estados tendrá como único comparador para realizar los cambios una señal llamada cont y la forma en que realizara los cambios es igualando el número equivalente a los 10 ms en la FPGA Elbert V2. Para

obtener dicho valor se debe hacer una regla de tres en donde 1 segundo equivale a los 12 MHz de la tarjeta, entonces se busca cuantos pasos o veces se debe contar para que pasen los 10 ms, el cálculo arroja el valor de 120000 y por este motivo es que en la máquina de estados se utiliza este valor como el comparador. La declaración de las señales necesarias para realizar dicha maquina se encuentran en la siguiente figura.

Figura 103.

Señales de entrada y salida de la entidad del módulo principal

```
11  architecture Behavioral of main is
12     type pasos_motor is (paso1, paso2, paso3, paso4);
13     signal estados : pasos_motor;
14     signal cont : natural:=0;
15  begin
```

Nota. En la figura se puede observar la creación de las señales que serán utilizadas para el proyecto que realizara el movimiento del eje del motor. Elaboración propia, realizado con ISE Design Suite 14.

Al momento en que llega el valor cont a 120000 realiza los cambios de los estados. Inicia con el estado Paso 1 almacenando en la señal de salida motor el primer valor de la secuencia que es 1000, una vez el valor del contador alcanza los 120000 realiza el cambio de estado al Paso 2 y este almacena en la señal motor el segundo valor de la secuencia, el valor 0100, esto ocurre un total de cuatro veces antes que se repita de nuevo el ciclo. En la Figura 104 se puede observar de forma más detallada el proceso que se ha seguido para la creación de la máquina de estados mencionada con anterioridad.

Figura 104.

Máquina de estados que realiza el paso a paso del eje del motor stepper

```
17      Pasos: process(clk)
18      begin
19         if (rising_edge(clk)) then
20            case estados is
21               when paso1 =>
22                  motor <= "1000";
23                  if (cont = 120000) then
24                     cont <= 0;
25                     estados <= paso2;
26                  else
27                     cont <= cont + 1;
28                  end if;
29               when paso2 =>
30                  motor <= "0100";
31                  if (cont = 120000) then
32                     cont <= 0;
33                     estados <= paso3;
34                  else
35                     cont <= cont + 1;
36                  end if;
37               when paso3 =>
38                  motor <= "0010";
39                  if (cont = 120000) then
40                     cont <= 0;
41                     estados <= paso4;
42                  else
43                     cont <= cont + 1;
44                  end if;
45               when paso4 =>
46                  motor <= "0001";
47                  if (cont = 120000) then
48                     cont <= 0;
49                     estados <= paso1;
50                  else
51                     cont <= cont + 1;
52                  end if;
53               when others =>
54                  estados <= paso1;
55            end case;
56         end if;
57        --aqui va un delay
58      end process;
59
60
61   end Behavioral;
```

Nota. En la figura se puede observar la máquina de estados que lleva el control del movimiento del eje del motor. Elaboración propia, realizado con ISE Design Suite 14.

El resultado del código al momento de realizar correctamente la conexión con el módulo ULN2003 debería ser un movimiento del eje de forma constante y sin un final establecido. El código anterior puede ser utilizado como un módulo genérico en aplicaciones más grandes, solo necesitaría ser adaptado para recibir una activación de forma externa que inicie el movimiento del motor stepper. Si en algún caso se necesite un mayor torque bastara con realizar las modificaciones a los pasos de la máquina de estados utilizando una secuencia que tenga una mayor cantidad de pasos.

4.4.3. Código de VHDL de movimiento con distintos ángulos con motor stepper

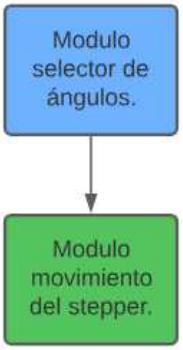
Una vez se tiene un funcionamiento básico del motor stepper se pueden utilizar una serie de instrucciones más complejas para lograr el movimiento del eje del motor una cierta cantidad de ángulos. Para demostrar el funcionamiento de dicho programa se deben realizar algunas modificaciones al módulo anterior e implementar un nuevo módulo que marque el funcionamiento de la conversión de pasos dependiendo el ángulo seleccionado.

En la Figura 105 se puede observar el diagrama de bloques que será utilizado para la programación del módulo principal del proyecto planteado. El diagrama consiste en unir dos módulos para lograr el movimiento del eje de un motor stepper a un ángulo deseado, el módulo que debe seleccionar ángulos realizara una conversión de una señal externa transformándola a un valor integer que represente los pasos necesarios para alcanzar el ángulo y enviara ese valor por medio de una señal STD_LOGIC_VECTOR al módulo que realiza el movimiento. El módulo de movimiento utilizará la señal recibida como un delimitador de pasos, una vez llegue a este valor el contador interno

la salida que conecta al motor stepper se detendrá. La explicación detallada de cada uno de los módulos se presentará en los párrafos siguientes.

Figura 105.

Módulo de operación de un motor stepper guiado por un selector de ángulos



Nota. En la figura se puede observar el diagrama de flujo que será programa en VHDL. Elaboración propia, realizado con Lucidchart.

4.4.3.1. Módulo selector de ángulos

El módulo selector de ángulos consiste en asignar a una salida STD_LOGIC_VECTOR el valor de un número entero que se obtiene al realizar la conversión de ángulo a pasos. El selector consiste en una entrada utilizando los Dip-switches de la FPGA los cuales asignaran el valor numérico a la salida del vector STD_LOGIC. Para lograr que los movimientos se deben mencionar que se requiere de un total de 512 pasos completos para lograr una vuelta de 360°. Con estos datos se puede realizar entonces una ecuación que funcione como conversión directa entre ángulos y pasos, al ser el valor 512 el máximo bastara con fraccionar el ángulo de 360 ° entre los ángulos deseados, en otras palabras, se debe tomar el ángulo deseado y dividirlo entre 360 ° para que su

resultado sea multiplicado por el valor de 512 para obtener el número de pasos necesarios para alcanzar el ángulo. En la tabla que se muestra a continuación se puede observar los ángulos y su valor equivalente en pasos que serán utilizados dentro del código de VHDL para el módulo selector.

Tabla 5.

Valores de ángulos convertidos a su equivalente en pasos

Angulo	Equivalente en pasos
30	43
45	64
60	85
90	128
120	170
135	192
150	213
180	255
225	340
270	383
315	447
360	511

Nota. Los valores obtenidos en los pasos equivalentes se obtienen considerando a los ángulos como x y usados en la ecuación: x/360 * 511. Elaboración propia, realizado con Excel.

Con la tabla anterior se puede explicar de forma más sencilla la estructura general del módulo. Se necesitan de dos entradas y una salida, las entradas serán los valores de los dip-switches necesarios para lograr la asignación de todos los ángulos y el valor del reloj interno, mientras que la salida será un único valor que soporte como valor máximo el número 511. La declaración de librerías, entradas y salidas se muestra a continuación.

Figura 106.

Señales de entrada y salida de la entidad del módulo selector de ángulos

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  use IEEE.std_logic_unsigned.all;
5
6  entity selectorangulo is
7      Port ( clk : in  STD_LOGIC;
8             selector : in  STD_LOGIC_VECTOR (4 downto 0);
9             angulo : out  STD_LOGIC_VECTOR (8 downto 0));
10 end selectorangulo;
```

Nota. En la figura se puede observar la creación de las señales que serán utilizadas para el módulo selector de ángulos. Elaboración propia, realizado con ISE Design Suite 14.

La arquitectura principal consiste simplemente en la implementación de un selector case que realizara el recorrido de las opciones de la señal de entrada selector, dentro de cada selección case se asignara a una señal integer llamada ang el valor del ángulo usando el equivalente de los pasos mostrado en la Tabla 5. La señal de ang será asignada a la señal de salida ángulo realizando una conversión a un vector unsigned y posteriormente se convierte a un valor STD_LOGIC_VECTOR. La arquitectura de dicha instrucción puede ser observada en la figura 107 que se muestra a continuación.

Figura 107.

Arquitectura del módulo selector de ángulos

```
architecture Behavioral of selectorangulo is
   signal ang: integer range 0 to 511;
begin
   Seleccion: process(clk)
   begin
      if (rising_edge(clk)) then
         case (selector) is
            when "00000" =>
               ang <= 0; --0°
               angulo <= STD_LOGIC_VECTOR(to_unsigned(ang, 9));
            when "00001" =>
               ang <= 43; --30°
               angulo <= STD_LOGIC_VECTOR(to_unsigned(ang, 9));
            when "00010" =>
               ang <= 64; --45°
               angulo <= STD_LOGIC_VECTOR(to_unsigned(ang, 9));
            when "00011" =>
               ang <= 85; --60°
               angulo <= STD_LOGIC_VECTOR(to_unsigned(ang, 9));
            when "00100" =>
               ang <= 128; --90°
               angulo <= STD_LOGIC_VECTOR(to_unsigned(ang, 9));
            when "00101" =>
               ang <= 170; --120°
               angulo <= STD_LOGIC_VECTOR(to_unsigned(ang, 9));
            when "00110" =>
               ang <= 192; --135°
               angulo <= STD_LOGIC_VECTOR(to_unsigned(ang, 9));
            when "00111" =>
               ang <= 213; --150°
               angulo <= STD_LOGIC_VECTOR(to_unsigned(ang, 9));
            when "01000" =>
               ang <= 255; --180°
               angulo <= STD_LOGIC_VECTOR(to_unsigned(ang, 9));
            when "01001" =>
               ang <= 298; --210°
               angulo <= STD_LOGIC_VECTOR(to_unsigned(ang, 9));
            when "01010" =>
               ang <= 319; --225°
               angulo <= STD_LOGIC_VECTOR(to_unsigned(ang, 9));
            when "01011" =>
               ang <= 340; --240°
               angulo <= STD_LOGIC_VECTOR(to_unsigned(ang, 9));
            when "01100" =>
               ang <= 383; --270°
               angulo <= STD_LOGIC_VECTOR(to_unsigned(ang, 9));
            when "01101" =>
               ang <= 426; --300°
               angulo <= STD_LOGIC_VECTOR(to_unsigned(ang, 9));
            when "01110" =>
               ang <= 447; --315°
               angulo <= STD_LOGIC_VECTOR(to_unsigned(ang, 9));
            when "01111" =>
               ang <= 468; --330°
               angulo <= STD_LOGIC_VECTOR(to_unsigned(ang, 9));
            when "10000" =>
               ang <= 511; --360°
               angulo <= STD_LOGIC_VECTOR(to_unsigned(ang, 9));
            when others =>
               ang <= 0;
               angulo <= STD_LOGIC_VECTOR(to_unsigned(ang, 9));
         end case;
      end if;
   end process;

end Behavioral;
```

Nota. En la figura se puede observar la arquitectura con las instrucciones utilizadas para realizar un selector de ángulos utilizando la entrada del selector y asignando los valores a la salida. Elaboración propia, realizado con ISE Design Suite 14.

En la Figura 107 al observar detenidamente cada uno de los casos que se han colocado se repite un mismo patrón, este módulo solo realiza la conversión del valor de los pasos dependiendo de la entrada que ingresa, la señal que envía a la salida será útil para el módulo del movimiento del stepper

al funcionar como un delimitador, por este motivo este módulo debe ser sencillo y su función es ser un multiplexor para entregar únicamente la señal que delimitara el conteo de los pasos.

4.4.3.2. Módulo movimiento delimitado del stepper

Este módulo es una variación del que fue presentado en el capítulo 4.4.2, la variación consiste en agregar a la máquina de estados dos funciones extras. Estas funciones realizan una comparación de los pasos y el valor recibido del módulo selector que decide si se debe continuar con el movimiento o no.

En la Figura 108 se puede observar el diagrama de la máquina de estados, los estados que se encuentran del Paso 1 al 4 son idénticos a los del módulo del capítulo 4.4.2 pero justo después del estado del Paso 4 realiza un cambio de estados a una serie completamente diferente, esta serie de estados son Pasofin y Reseteo.

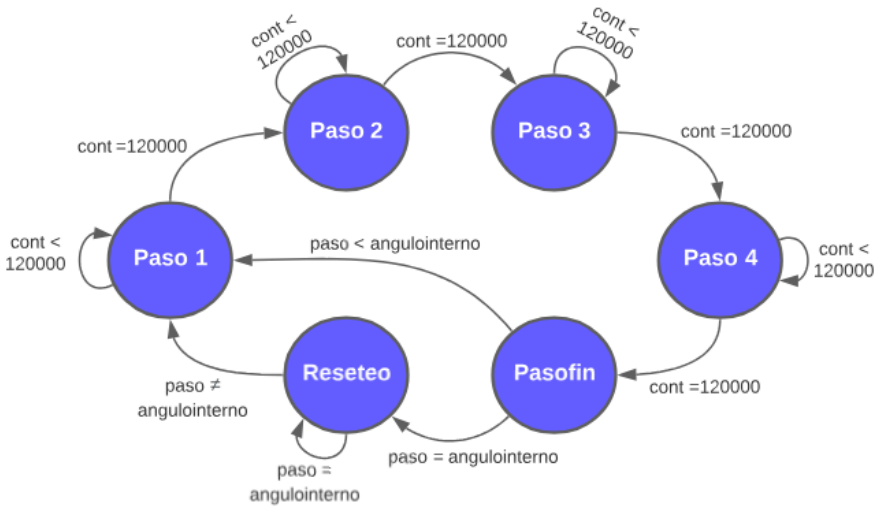
El estado Pasofin realiza una comprobación de un contador interno llamado paso con el valor que el módulo selector de ángulos envía, el valor se almacena en la señal angulointerno y es el utilizado en la comparación con paso. Si la señal paso es menor a la de angulointerno el estado Pasofin regresa a la secuencia del Paso 1 al Paso 4 para que el motor stepper de otro paso, si el valor del contador paso es idéntico al de angulointerno realiza un cambio al estado Reseteo y esto provocara que no siga la secuencia de pasos y el motor se detendrá. La idea de colocar un estado de transición entre la secuencia de movimiento y la secuencia que detiene dicho movimiento ayuda a reiniciar el contador de varias señales y da paso a posibles modificaciones

del módulo al poder iniciar un proceso diferente después del estado de transición.

El estado Reseteo funciona a modo de retenedor de señal. Adentro del módulo no existe ningún contador y lo único que lo puede hacer cambiar es que se modifique el valor del ángulo solicitado, mientras el ángulo no se modifique nunca se cambiara de estado y el cambio solo es posible si los dip-switches cambian de estado. El estado Reseteo al realizar la comparación y notar el cambio reinicia las variables del contador pasos para después hacer el cambio a la secuencia de movimiento del motor stepper.

Figura 108.

Máquina de estados del módulo de movimiento del eje del motor stepper



Nota. En la figura se puede observar la máquina de estados que será programa en VHDL. Elaboración propia, realizado con Lucidchart.

Para lograr realizar la máquina de estados se debe considerar las entradas y salidas del módulo necesarias para la operación, también las

señales de uso interno que se necesitan para lograr realizar el cambio de los estados. La entrada más importante es de la que se obtiene el valor del ángulo, esta debe ser idéntica a la señal de salida del módulo selector de ángulo y será la única variante dentro del módulo, el resto de las entradas y salidas son idénticas a las del módulo base.

Figura 109.

Señales de entrada y salida del módulo de movimiento del motor stepper

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  use IEEE.std_logic_unsigned.all;
5
6  entity motorstep is
7      Port ( clk : in  STD_LOGIC;
8             angulo : in STD_LOGIC_VECTOR(8 downto 0);
9             motor : out  STD_LOGIC_VECTOR (3 downto 0));
10 end motorstep;
```

Nota. En la figura se puede observar la creación de las señales que serán utilizadas para el módulo de movimiento del motor stepper. Elaboración propia, realizado con ISE Design Suite 14.

Con las señales de entrada y salida creadas se puede realizar la creación de la máquina de estados dentro de la arquitectura junto con el resto de las señales auxiliares a utilizar, entre estas señales se encuentran las de pasos y angulointerno que fueron mencionadas en el diagrama de la máquina de estados de la Figura 108. Ambas señales son de tipo integer y es necesario inicialar en cero la variable pasos para que el sintetizador tenga el rango de referencia y no asigne un valor desconocido a la señal, para el caso de angulointerno no será necesario porque se le asignara el valor de la señal de entrada angulo antes de que inicien las comparaciones.

Figura 110.

Máquina de estados del módulo de movimiento del motor stepper

```
12  architecture Behavioral of motorstep  is
13     type pasos_motor is (paso1, paso2, paso3, paso4, pasofin, reseteo);
14     signal estados : pasos_motor;
15     signal cont : natural:=0;
16     signal paso : integer:=0;
17     signal angulointerno : integer;
```

Nota. En la figura se puede observar el inicio de la arquitectura del módulo de movimiento del motor stepper, la creación de la máquina de estados y de las señales auxiliares que utilizara la máquina de estados para los procedimientos internos. Elaboración propia, realizado con ISE Design Suite 14.

El inicio de la arquitectura es por medio de un proceso llamado Pasos, este proceso es el mismo que fue trabajado en el programa del movimiento básico del motor stepper. Los primeros estados consisten en el mismo contador que se utilizó con anterioridad que al llegar a los 120000 pasos realiza el cambio al siguiente estado, la diferencia para este caso tal como fue mostrado en el diagrama de la máquina de estados es que al momento de llegar al estado paso 4 y terminar el conteo de los 120000 cambios de reloj, utilizando otras medidas los 10ms, el cambio de estado no regresa al estado paso 1 sino que realiza el cambio de estado al de Paso fin. El cambio a dicho estado es para realizar la comprobación de cuantas veces se completa la secuencia de activación de las bobinas, si se llega a igual a la señal que ha enviado el selector de ángulo no se seguirá con la secuencia del movimiento del eje. La secuencia encargada del movimiento del eje del stepper se puede observar en la imagen mostrada a continuación.

Figura 111.

Estados de los pasos que realizan el movimiento del motor stepper

```
21    Pasos: process(clk)
22    begin
23       if (rising_edge(clk)) then
24          case estados is
25             when paso1 =>
26                motor <= "1000";
27                if (cont = 120000) then
28                   cont <= 0;
29                   estados <= paso2;
30                else
31                   cont <= cont + 1;
32                end if;
33             when paso2 =>
34                motor <= "0100";
35                if (cont = 120000) then
36                   cont <= 0;
37                   estados <= paso3;
38                else
39                   cont <= cont + 1;
40                end if;
41             when paso3 =>
42                motor <= "0010";
43                if (cont = 120000) then
44                   cont <= 0;
45                   estados <= paso4;
46                else
47                   cont <= cont + 1;
48                end if;
49             when paso4 =>
50                motor <= "0001";
51                if (cont = 120000) then
52                   cont <= 0;
53                   estados <= pasofin;
54                else
55                   cont <= cont + 1;
56                end if;
```

Nota. En la figura se puede observar el inicio de la arquitectura del módulo de movimiento del motor stepper, la creación de la máquina de estados y de las señales auxiliares que utilizara la máquina de estados para los procedimientos internos. Elaboración propia, realizado con ISE Design Suite 14.

El primer paso del estado paso fin es cargar a la señal ángulo interno la conversión del valor que tiene el vector ángulo que contiene la información del ángulo utilizado, al obtener el dato realiza la comparación con su contador de

pasos interno llamado paso para evaluar si tiene el mismo valor. Si el eje no llega al valor de ángulo deseado el estado paso fin seguirá realizando la secuencia de movimiento iniciándola nuevamente haciendo el cambio de estado al de paso 1 pero al momento en el que el contador iguale el valor del ángulo realizara el cambio al estado reseteo que retendrá la señal a menos que exista algún cambio. En el módulo reseteo se realiza siempre la lectura del valor del ángulo y esta es comparada nuevamente con el valor de paso, esto realiza porque al hacer el cambio de estado paso fin a reseteo se ha conservado el valor instantáneo del ángulo en la señal paso y esta no puede ser modificada a menos que el valor del ángulo externo sea modificado. Con esta acción se consigue que el estado reseteo se mantenga en un ciclo de lectura constante y cuando detecte el cambio del ángulo reinicia el valor de paso y vuelve al estado inicial de paso1 para repetir el ciclo de lectura.

Figura 112.

Estados de la arquitectura del módulo de movimiento del motor stepper

```
57              when pasofin =>
58                 angulointerno <= to_integer(unsigned(angulo));
59                 if (paso = angulointerno) then
60                    estados <= reseteo;
61                 else
62                    paso <= paso + 1;
63                    estados <= paso1;
64                 end if;
65              when reseteo =>
66                 angulointerno <= to_integer(unsigned(angulo));
67                 if (paso = angulointerno) then
68                    estados <= reseteo;
69                 else
70                    paso <= 0;
71                    estados <= paso1;
72                 end if;
73              when others =>
74                 estados <= paso1;
75              end case;
76           end if;
77     end process;
78  end Behavioral;
```

Nota. En la figura se puede observar los últimos dos estados del módulo de movimiento del motor stepper. Elaboración propia, realizado con ISE Design Suite 14.

Para el módulo principal bastara con asignar la señal de entrada del reloj, de los dip-switches y la salida vectorial de los pines conectados al motor stepper. Los módulos del selector de ángulos y del movimiento del motor stepper se deben combinar en el módulo principal utilizando un único conector de tipo STD_LOGIC_VECTOR para que puedan intercambiar la información del ángulo. La creación de dicho módulo es sencilla y puede observarse en la Figura 113.

Figura 113.

Módulo principal del selector de angulo y movimiento del motor stepper

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  use IEEE.std_logic_unsigned.all;
5  entity top is
6      Port ( clk : in  STD_LOGIC;
7             selector : in  STD_LOGIC_VECTOR (4 downto 0);
8             motor : out  STD_LOGIC_VECTOR (3 downto 0));
9  end top;
10 architecture Behavioral of top is
11    signal anguloint: STD_LOGIC_VECTOR (8 downto 0);
12 begin
13    Angulo:
14       entity work.selectorangulo
15       port map(
16          clk => clk,
17          selector => selector,
18          angulo => anguloint
19          );
20    Stepper:
21       entity work.motorstep
22       port map(
23          clk => clk,
24          angulo => anguloint,
25          motor => motor
26       );
27 end Behavioral;
```

Nota. En la figura se puede observar el módulo principal que realiza la combinación del módulo selector de ángulo y del movimiento del motor stepper. Elaboración propia, realizado con ISE Design Suite 14.

La conclusión del proyecto deja como resultado un selector de movimientos del eje de un total de 16 posibles ángulos, se pueden realizar ciertas modificaciones en las secuencias para realizar estos mismos movimientos en sentido contrario para tener un sistema más preciso de movimiento o modificar la secuencia de paso completo por si se necesita un mayor torque en el motor. La parte más importante será la conversión de los ángulos a cuantos pasos da el motor, que cumplan la secuencia de encendido de bobinas y las conversiones de los tipos de datos integer a STD_LOGIC.

4.5. Uso de protocolo de comunicación UART en VHDL

Entre las múltiples funciones y aplicaciones que tienen las FPGA existe una en particular que puede permitir él envió de datos a una computadora, una FPGA o un microcontrolador, esta función es la creación de un protocolo de comunicación para él envió de datos. Existen múltiples tipos de protocolos estandarizados, entre los más populares se encuentran el I2C, UART y SPI. En este capítulo se presentará la implementación del protocolo UART en la Elbert V2 para realizar un intercambio de información entre una computadora y la tarjeta.

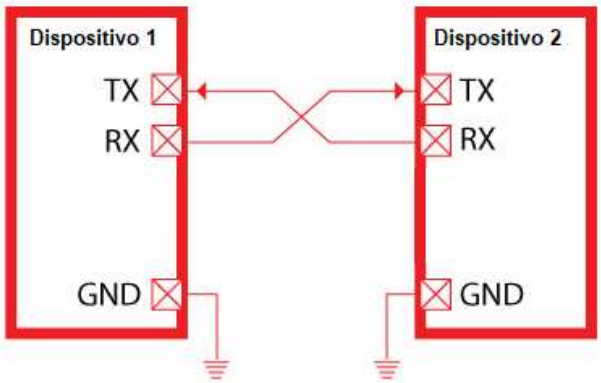
4.5.1. Funcionamiento y teoría del protocolo de comunicación UART

El protocolo receptor/transmisor asíncrono universal o denominado UART por sus siglas en ingles consiste en una serie de parámetros estandarizados que permiten la comunicación entre dos dispositivos haciendo uso de un par de cables o puertos asincrónicos. Este protocolo además de la señal de tierra posee dos conexiones para la comunicación, las cuales son:

- Tx: es la que se encarga de la escritura de datos.
- Rx: es la que se encarga de la lectura de datos.

Figura 114.

Terminales en el protocolo de comunicación UART entre dos dispositivos



Nota. En la figura se puede observar la forma de conexión habitual entre dos dispositivos que utilizan el protocolo de comunicación UART. Elaboración propia, realizado con Proteus 8 Professional.

La información es enviada por medio del puerto Tx y en el otro dispositivo el puerto Rx lee esta información. Para enviar la información el protocolo empaqueta la información en una serie de bits llamada trama. Una trama para el protocolo UART consiste generalmente en una serie de 10 bits, en los cuales la información ocupa 8 de estos bits y los otros 2 bits son ocupados por algo llamado bit de inicio y parada. El protocolo requiere armar la trama de tal forma que el primer bit que se envía es el de inicio o de aviso, esto es para que el puerto Rx pueda activarse y preparar la lectura de los 8 bits que llegaran, para que el puerto Rx sepa que la información ha sido

enviada por completo se utiliza el bit de parada o fin el cual indica que él envió ha finalizado.

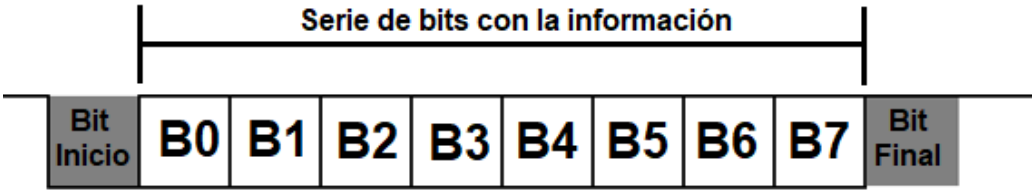
Al ser un protocolo asíncrono y no tener una señal de reloj incorporada en la comunicación se debe establecer un convenio en la velocidad de lectura y escritura para que el emisor y el receptor puedan entender el mensaje, esto se logra por medio de una sincronización que estandariza la velocidad en baudios de la transmisión. Dicha velocidad representa el número de veces que pueden cambiar la señal por cada segundo que transcurre, los tipos de velocidad más comunes que utilizan los protocolos son:

- 4800 baudios
- 9600 baudios
- 19200 baudios

En resumen, para garantizar el funcionamiento del protocolo UART se debe establecer la velocidad en baudios idéntica en el emisor y receptor y encápsular la información dentro de los bits de aviso de inicio y fin de trama.

Figura 115.

Estructura de una trama de bits del protocolo de comunicación UART



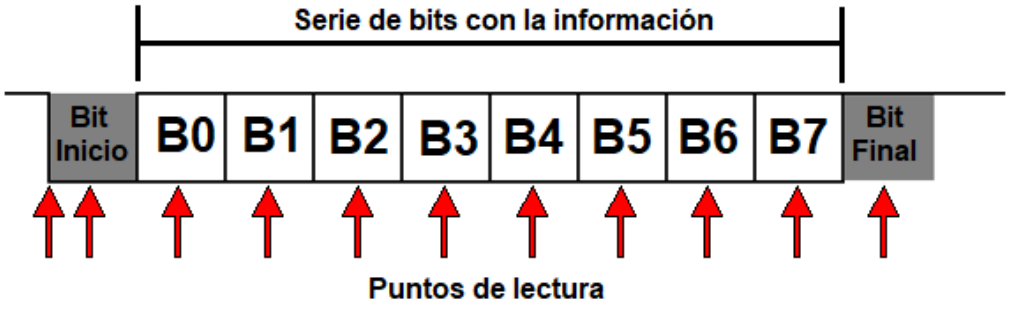
Nota. En la figura se puede observar la distribución de los bits de información, inicio y final dentro de una trama de bits utilizada en la comunicación con el protocolo UART. Elaboración propia, realizado con Word.

4.5.2. Módulo Rx del UART en VHDL

El puerto o señal Rx es el encargado de la lectura de los datos que se reciban en el protocolo de comunicación UART. El puerto Rx siempre recibe una señal en alto cuando no se está enviando información, lo que marca el inicio del envió de datos es el cambio de estado 1 a 0, dicho cambio de estado se encuentra en el llamado Bit de Inicio. Una vez el Rx realiza la lectura del bit de inicio debe realizar la lectura de la cadena de bits que está ingresando, esto se repetirá un total de 8 veces para tomar la lectura de los 8 bits. El final de la serie lo marcará el Bit de Fin que devolverá a un estado 1 la señal. Se debe señalar que todas las lecturas de los bits es recomendable realizarlas justo en medio del periodo de transición de bits.

Figura 116.

Lectura de la trama bits utilizada en el protocolo de comunicación UART



Nota. En la figura se puede observar por medio de flechas los puntos en donde debe realizarse la lectura en el puerto Rx del módulo UART. Elaboración propia, realizado con Word.

La estructura que permite realizar las lecturas vistas en la anterior figura puede ser programada creando una máquina de estados en VHDL que contenga cuatro estados para lograr la lectura correcta de los datos. El módulo que será creado debe tener cuatro estados, las funciones y los nombres son mencionados a continuación:

- Idle: sera el estado principal, representara el periodo de inactividad del módulo Rx. Este estado realizará la revisión constante de la entrada de la señal Rx, dicha entrada será programada con el nombre rx_in. Este estado no realizara cambio siempre que tenga una entrada en alto y cambiara si y solo si se lee el bit de inicio de la trama que será cuando la entrada rx_in sea igual a 0.

- Start: sera el estado que confirme el inicio de la trama, en este estado se realiza una segunda comprobación del bit de inicio realizando la lectura en medio del periodo de actividad de dicho bit, en caso el estado

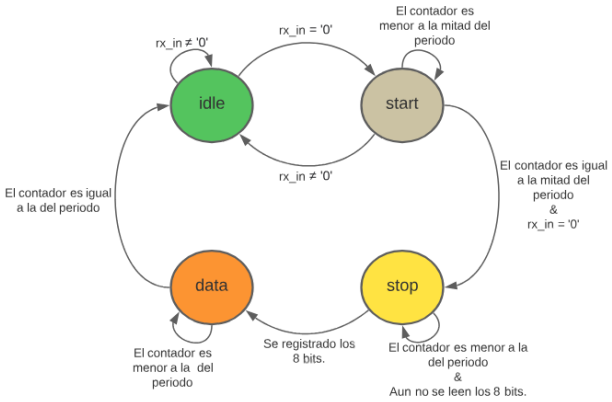
Idle haya realizado el cambio por error el estado Start lo detectará y regresará a Idle, de lo contrario dará paso al siguiente estado.

- Data: en este estado es donde se realizan las lecturas de los 8 bits de la trama, almacena en una señal los datos recibidos y realiza las lecturas en medio de los periodos de actividad de cada bit de información. Una vez realiza la lectura de los 8 bits cambiara al siguiente estado.

- Stop: en este estado se realiza una segunda comprobación utilizando el bit final y esperando un tiempo determinado para finalizar con la lectura, una vez realiza esto regresa al estado Idle y la maquina repite los procesos.

Figura 117.

Máquina de estados del módulo Rx del protocolo UART



Nota. En la figura se puede observar la máquina de estados que será programa en VHDL que realizará la lectura de datos utilizando el protocolo UART. Elaboración propia, realizado con Lucidchart.

Para adaptar la máquina de estados mostrada en la Figura 117 se debe establecer las entradas y salidas que tendrá el programa en VHDL, se propone tener una serie de constantes que establecerán los temporizadores de conteo de la máquina de estados adicional a las señales necesarias para el funcionamiento. Los contadores deben tener como limite el valor de la tasa de baudios que se utilizará en la lectura, para hacer un equivalente se debe realizar la división de la frecuencia del reloj interno con la taza de baudios que será utilizada. En la lista que se muestra a continuación se pueden observar las entradas y constantes que serán declaradas en VHDL, además de dar una breve explicación de cada una de dichas señales:

- n_bits: esta será la constante que tendrá el valor de la cantidad de bits a recibir.

- contador_bits: esta constante almacenara el valor límite del contador, este valor surge dividiendo la frecuencia del reloj (12 MHz) con la tasa de baudios a utilizar, para el ejemplo se utilizarán 9600 bps.

- contador_half_bits: esta constante tendrá la mitad del valor de la señal contador_bits y se realiza de esta forma por la forma en que se tomaran las lecturas en la máquina de estados.

- clk: esta es la señal de reloj de la FPGA.

- rx_in: esta será la señal que almacenará la información del puerto que recibe la trama de bits externa.
- rx_data: para corroborar que él envió de datos es correcto se desplegara en los leds que tiene la FPGA, en la señal rx_data será donde serán almacenados esos datos.

En la Figura 118 se muestra la creación de la entidad del módulo, también las librerías necesarias para garantizar el funcionamiento y la creación de cada una de las señales mencionadas anteriormente. Se puede observar que se utiliza la señal genérica n_bits para establecer la longitud del vector a utilizar para la entrega de datos recibidos, esto se realiza de esta forma para que pueda ser modificado el dato de 8 bits y no afecte el funcionamiento del módulo, en instrucciones dentro de la máquina de estados también será utilizado este valor. Adicional a lo anterior también se muestra el inicio de la arquitectura y la creación de la máquina de estados, se crean las señales auxiliares del contador interno y el índice que servirá para recorrer el vector que entregara la información recibida.

Figura 118.

Entidad UART_RX con las señales genéricas y puertos a utilizar

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  use IEEE.std_logic_unsigned.all;
5
6  entity UART_RX is
7     generic(
8        n_bits : integer := 8;
9        contador_bits : integer := 1250;
10       contador_half_bits : integer := 625
11    );
12    port(
13       clk : in STD_LOGIC;
14       rx_in : in STD_LOGIC;
15       rx_data : out STD_LOGIC_VECTOR(n_bits - 1 downto 0)
16    );
17 end UART_RX;
18
19 architecture Behavioral of UART_RX is
20    type uart_rx is (idle, start, data, stop);--FSM uart_rx
21    signal estados : uart_rx;
22    signal contador : integer range 0 to contador_bits - 1 := 0;
23    signal indice: integer range 0 to n_bits - 1 := 0;
```

Nota. En la figura se puede observar la entidad creada UART_RX junto con las librerías y puertos necesarios para su funcionamiento. Elaboración propia, realizado con ISE Design Suite 14.

Una vez se ha creado la máquina de estados y las diferentes señales a utilizar se puede iniciar a programar los estados, el primero de ellos es el estado idle. Para dicho estado se inicializan las señales del contador y del índice a cero y comienzan con la comparación de la señal rx_in para verificar si ha pasado algún 0 por dicha señal. Esto se realiza de esta forma para verificar si el estado permanente en 1 fue alterado, si rx_in registra el cambio de 1 a 0 ejecuta un cambio de estado y si este cambio no ocurre regresa al inicio del estado idle y vuelve a reiniciar variables. Este proceso se repetirá hasta que se detecte el cambio de estados, en la figura que se muestra a continuación se puede observar dicha comparación en la línea 32 del código.

Figura 119.

Inicio de la arquitectura y la programación del primer estado llamado Idle

```
24  begin
25     process (clk)
26     begin
27        if (rising_edge(clk)) then
28           case estados is
29              when idle =>--Cuando esta en espera
30                 contador <= 0;
31                 indice <= 0;
32                 if (rx_in = '0') then
33                    estados <= start;
34                 else
35                    estados <= idle;
36                 end if;
```

Nota. En la figura se puede observar el proceso que contiene la máquina de estados y el primer estado programado siendo este el estado idle. Elaboración propia, realizado con ISE Design Suite 14.

Si el estado idle ha cambiado y dado inicio con la lectura de la trama de bits el siguiente paso es el estado start. En el estado start se realiza un conteo de la mitad del valor total del periodo que dura la tasa de baudios, esto se hace para realizar una lectura justo a la mitad del periodo de cada bit y puede ser observado en la siguiente figura en la línea 38. Una vez completa el conteo realiza la lectura nuevamente de la entrada rx_in para hacer una segunda comprobación del valor 0. Si el estado idle obtuvo una lectura errónea de la señal rx_in el estado start lo detectara y cambiara al estado anterior, pero si logra identificar que el valor de rx_in ha permanecido en 0 realiza el cambio al estado que ejecutara la lectura completa de la trama. Se puede interpretar al estado start como un estado que realiza una detección de error al realizar una segunda comprobación del bit de inicio de la trama para confirmar que idle no haya fallado en la lectura. El código de dicho estado puede ser observado en la imagen que se muestra a continuación.

Figura 120.

Programación del segundo estado llamado start

```
37              when start =>
38                 if (contador = contador_half_bits) then
39                    case rx_in is
40                       when '0' =>
41                          contador <= 0;
42                          estados <= data;
43                       when others=>
44                          estados <= idle;
45                    end case;
46                 else
47                    contador <= contador + 1;
48                    estados <= start;
49                 end if;
```

Nota. En la figura se puede observar el segundo programado siendo este el estado start. Elaboración propia, realizado con ISE Design Suite 14.

Una vez se confirma por medio de los estados idle y start que se está recibiendo una trama de bits se hace uso del estado data. En este estado se realiza la lectura de los 8 bits entrantes y se almacenan en una salida. Para realizar las lecturas en un tiempo correcto y capturar los 8 bits se utiliza un contado que tiene como delimitador la tasa de baudios completa declarada en la señal del inicio, este conteo se puede observar en la condicional de la línea 51. Una vez se cumple con el conteo se carga a la señal de salida rx_data el valor instantáneo de la entrada del Rx, para lograr almacenar todos los bits entrantes en un orden determinado se utiliza una señal auxiliar llamada indice que realiza el recorrido del vector. Una vez la señal índice finaliza con el recorrido puede realizar el cambio al estado final del módulo y los datos que son recibidos pueden ser observados. La condición que debe cumplir el índice se encuentra en la línea 54 del código y este cambiara cuando se iguale el valor de índice con el valor del número de bits establecido en la declaración de señales de la entidad.

Figura 121.

Programación del tercer estado llamado data

```
50              when data =>
51                 if (contador = contador_bits - 1) then
52                    contador <= 0;
53                    rx_data(indice) <= rx_in;
54                    if (indice = n_bits - 1) then
55                       indice <= 0;
56                       estados <= stop;
57                    else
58                       indice <= indice + 1;
59                       estados <= data;
60                    end if;
61                 else
62                    contador <= contador + 1;
63                    estados <= data;
64                 end if;
```

Nota. En la figura se puede observar el tercer estado programado siendo este el estado data. Elaboración propia, realizado con ISE Design Suite 14.

Al finalizar la captura de la trama de bits del estado data llega el último paso que consiste en realizar un conteo extra de un ciclo completo para realizar un retorno al estado inicial, dicho conteo lo realiza el estado stop y solo realiza una comparación en la línea 66 que al cumplir con la condición realiza el cambio al estado inicial idle terminando con el proceso de lectura del protocolo UART.

Figura 122.

Programación del cuarto estado llamado stop y finalización del módulo Rx

```
65              when stop =>
66                 if (contador = contador_bits - 1) then
67                    contador <= 0;
68                    estados <= idle;
69                 else
70                    contador <= contador + 1;
71                    estados <= stop;
72                 end if;
73              when others =>
74                 estados <= idle;
75           end case;
76        end if;
77     end process;
78  end Behavioral;
```

Nota. En la figura se puede observar el cuarto y último estado llamado stop y el resto de código que finaliza el módulo Rx del protocolo UART. Elaboración propia, realizado con ISE Design Suite 14.

4.5.3. Módulo Tx del UART en VHDL

El puerto Tx del módulo de comunicación UART es el encargado realizar el empaquetado de los datos de información dentro de la trama de bits que es enviada a otros dispositivos. Como se muestra en la Figura 115 la trama de bits se envía después de que la señal se encuentra en un estado alto, esto indica que se debe programar un estado permanente en 1 hasta que se envié la trama de datos, una vez envía la dicha trama el estado debe volver a 1.

Para hacer posible él envió de datos utilizando la FPGA se diseñará una máquina de estados que permita colocar los valores que se desean enviar haciendo uso de los dip-switches de la tarjeta y al pulsar un botón estos datos sean enviados a través de un puerto que actuara como un Tx del protocolo de comunicación UART. Los estados que se necesitan para realizar dicho

proceso se requieren cuatro estados, dichos estados y sus funciones se detallan a continuación:

- Idle: este será el estado inicial en el que el programa del envió de datos espera la pulsación de un botón para dar paso al contador que enviará dichos datos. Dentro de este estado existe una función antirrebote que evita enviar en múltiples ocasiones la misma instrucción. Las señales auxiliares como contadores y los índices son reseteadas también dentro de este estado.

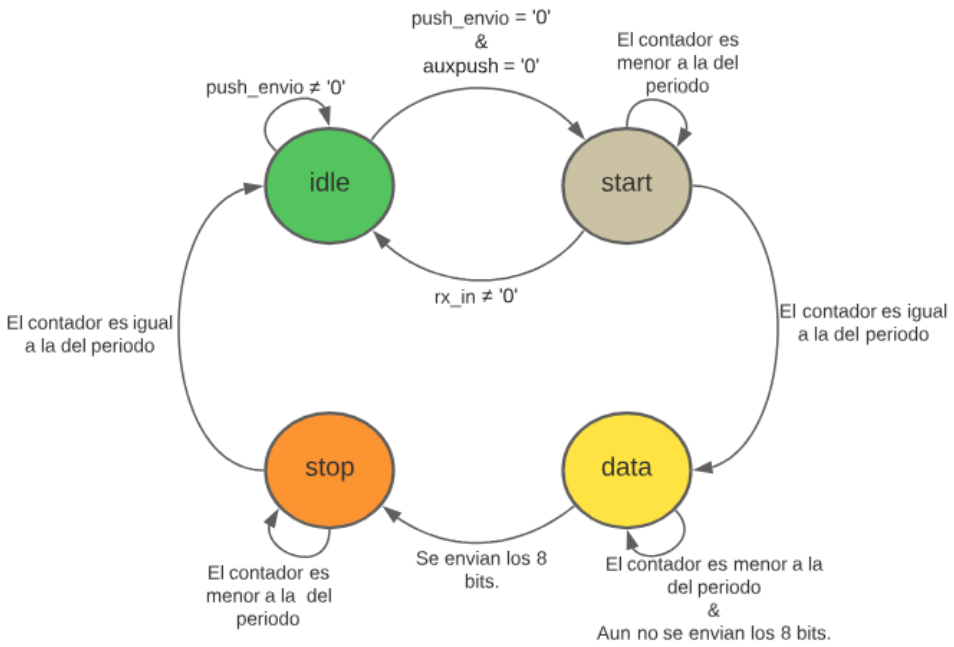
- Start: en este estado cambia la salida de 1 a 0 para dar aviso al Rx de que vendrá una trama de bits y solo cambiara de estado una vez se complete el conteo del equivalente de la tasa de baudios, esto se hace para que transcurra un solo periodo para iniciar la transmisión de los bits de información.

- Data: este estado realiza él envió de los 8 bits de información, para hacerlo hace un recorrido del vector de entrada con la información de los dip-switches para enviar los datos a la salida Tx , la señal que controla el recorrido actúa como un índice y utiliza el contador interno para enviar los datos siempre y cuando el contador cumpla con la periodicidad requerida. Al finalizar el envío de los 8 bits de información hace un cambio de estado nuevamente.

- Stop: este es el último estado, al hacer el cambio coloca nuevamente la señal de salida en un 1 permanente y realiza un conteo de un solo ciclo, una vez cumple con el conteo regresa al estado Idle.

Figura 123.

Máquina de estados del móduloTx del protocolo UART



Nota. En la figura se puede observar la máquina de estados que será programa en VHDL que realizará el envió de datos utilizando el protocolo UART. Elaboración propia, realizado con Lucidchart.

Las señales de entrada y salida necesarias para la creación de la máquina de estados se detallan a continuación:

- contador_bits: esta será una señal genérica que tendrá el valor de conversión de los 9600 bps en su equivalente de pasos para igualar el periodo activo de cada una de las señales, para este caso será el valor de 1250.

- n_bits: esta señal genérica contendrá el valor de la cantidad de bits de información que lleva la señal. Para la aplicación que será programa se necesitan un total de 8 bits.

- clk: esta será la señal de reloj de la FPGA.

- push_envio: al utilizar un botón para él envió de la información se ha creado esta señal para realizar la captura de datos de dicho botón, una vez se detecte algún cambio en esta señal dará inicio la transmisión de la trama de bits.

- tx_datos: es la señal en donde serán asignados los dip-switches para poder modificar los datos que se desean enviar en la trama de bits.

- tx_salida: esta señal será a la que se le asigne el puerto en donde la trama de bits será transmitida.

En la Figura 119 se muestra el código necesario para la creación de la entidad del módulo con las señales genéricas y los puertos a utilizar. Dentro de la figura que se ha descrito se encuentran también las librerías para garantizar el funcionamiento, las señales auxiliares que se utilizaran dentro del código de la arquitectura, además de mostrar la creación de la máquina de estados siguiendo el diagrama mencionado con anterioridad.

Figura 124.

Creación de la entidad UART_TX con las señales genéricas y puertos a utilizar

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  use IEEE.std_logic_unsigned.all;
5
6  entity UART_TX is
7        generic(
8           contador_bits : integer := 1250;
9           n_bits : integer := 8
10       );
11       port(
12          clk :          in STD_LOGIC;
13          push_envio:    in STD_LOGIC;
14          tx_datos:      in STD_LOGIC_VECTOR(n_bits - 1 downto 0);
15          tx_salida:     out STD_LOGIC
16       );
17 end UART_TX;
18 architecture Behavioral of UART_TX is
19    Type uart_tx is (idle, start, data, stop);
20    signal estados : uart_tx;
21    signal auxpush : STD_LOGIC:= '0';
22    signal contador : integer range 0 to contador_bits-1 := 0;
23    signal indice : integer range 0 to 7 := 0;
24    signal datos   : STD_LOGIC_VECTOR(7 downto 0);
```

Nota. En la figura se puede observar la entidad creada UART_TX junto con las librerías y puertos necesarios para su funcionamiento. Elaboración propia, realizado con ISE Design Suite 14.

Los estados utilizados comparten nombre con los que se mostraron en el módulo Rx, sin embargo, en algunos de estos estados el funcionamiento es completamente distinto debido a las funciones del Tx. Para el caso del primer estado de la máquina, el estado idle, realiza la comprobación de la pulsación del botón para verificar si se puede iniciar con la transmisión del Tx. Dicho estado también se encarga de realizar un reset de las señales estableciendo un valor inicial cero al contador y al índice para que no exista problemas en los demás módulos, la condición que debe cumplir para poder realizar el cambio de estado se encuentra en la línea 35 y tiene una segunda verificación antirrebote simple que puede observarse en la línea 36, solo si es pulsado el

botón de envío realiza el cambio al siguiente estado. En la figura que se muestra a continuación puede ser observado el código del primer estado con mejor detalle.

Figura 125.

Creación del estado idle del módulo UART_TX

```
27    process (clk)
28    begin
29       if (rising_edge(clk)) then
30          case estados is
31             when idle =>
32                   tx_salida <= '1';
33                   contador <= 0;
34                   indice <= 0;
35                   if (push_envio = '0') then
36                      if (auxpush = '0') then
37                            auxpush <= '1';
38                            datos <= tx_datos;
39                            estados <= start;
40                      end if;
41                   else
42                      auxpush <= '0';
43                      estados <= idle;
44                   end if;
```

Nota. En la figura se puede observar el primer estado llamado idle del módulo UART_TX. Elaboración propia, realizado con ISE Design Suite 14.

Una vez se pulsa el botón y se confirma que no ha sido una acción accidental por parte del ruido tipo de dichos botones la máquina de estados cambia al estado start. El estado start consiste en realizar el cambio de la señal que se encontraba en un constate 1 a un 0 para dar aviso al Rx que está conectado al módulo que se aproxima una trama de bits, el cambio del estado de la variable de salida se realiza en la línea 47 para después realizar un conteo de un periodo, una vez se completa el conteo del periodo asignado se puede realizar el cambio de estado para poder comenzar él envió de la información.

Figura 126.

Creación del estado start del módulo UART_TX

```
46              when start =>
47                    tx_salida <= '0';
48                    if (contador = contador_bits) then
49                       contador <= 0;
50                       estados   <= data;
51                    else
52                       contador <= contador + 1;
53                       estados   <= start;
54                    end if;
```

Nota. En la figura se puede observar el segundo estado llamado start del módulo UART_TX. Elaboración propia, realizado con ISE Design Suite 14.

El estado que controla el envío de información es el estado data, en este estado se realiza una lectura constante de la señal de entrada datos haciendo un recorrido del vector con el uso de la señal auxiliar índice que suma por uno cada vez que se completa el ciclo de espera para el envío del bit. El ciclo de espera funciona igual que el contador del estado start, realiza la espera de un periodo para ejecutar un reinicio de la señal del contador y de realizar la suma a la señal del índice, con esta acción se logra recorrer todo el vector de entrada datos y una vez termina de leer los ocho valores realiza el cambio de estado al siguiente llamado stop.

Para el estado stop lo único que se realiza es un cambio de nuevo en la salida del puerto conectado a Tx para que tenga una salida constante de 1, con esto se indica al Rx que este leyendo la trama que la transmisión ha sido finalizada. El estado stop al cambiar la salida realiza un conteo nuevamente de un solo periodo, una vez se completa dicho conteo realiza el cambio de estado a idle haciendo que se repita el ciclo una vez más si se llega a pulsar el botón. Tanto el código del estado data como el estado stop puede ser observado en la Figura 127 que se muestra a continuación.

Figura 127.

Creación del estado data y stop del módulo UART_TX

```
56              when data =>
57                    tx_salida <= datos(indice);
58                    if (contador = contador_bits) then
59                       contador <= 0;
60                       if (indice < 7) then
61                          indice <= indice + 1;
62                          estados   <= data;
63                       else
64                          indice <= 0;
65                          estados   <= stop;
66                       end if;
67                    else
68                       contador <= contador + 1;
69                       estados   <= data;
70                    end if;
71
72              when stop =>
73                     tx_salida <= '1';
74                    if (contador = contador_bits) then
75                       contador <= 0;
76                       estados   <= idle;
77                    else
78                       contador <= contador + 1;
79                       estados   <= stop;
80                    end if;
81              when others =>
82                 estados <= idle;
83           end case;
84        end if;
85     end process;
86  end Behavioral;
```

Nota. En la figura se puede observar los últimos dos estados de la máquina de estados y la finalización de la arquitectura del módulo UART_TX. Elaboración propia, realizado con ISE Design Suite 14.

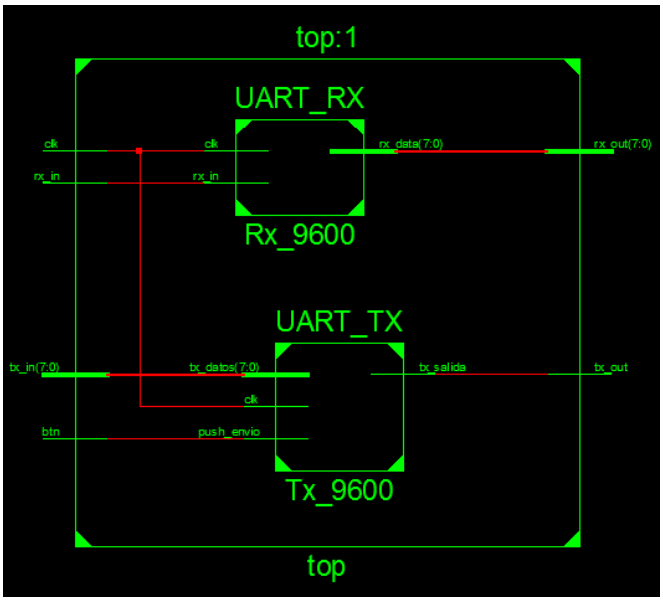
4.5.4. Módulo Top del UART en VHDL y su uso con el módulo CP2102

Para poder tener en la Elbert V2 un módulo de comunicación UART completo capaz de interactuar con módulos externos a través del protocolo se deben unificar los módulos Tx y Rx en un módulo general que conecte las

señales y puertos necesarios para que pueda funcionar correctamente. Para que el programa pueda ser utilizado para comunicaciones externas se deben programar los módulos de forma independiente en sus salidas, no deben conectar entre sí, en la figura 128 se puede observar la forma en que deben ser las conexiones para lograr este objetivo.

Figura 128.

Unificación de los módulos Rx y Tx.



Nota. En la figura se puede observar la forma en que deben ser conectados los módulos Rx y Tx para que pueda ser utilizado el protocolo UART con módulos de comunicación externos. Elaboración propia, realizado con ISE Design Suite 14.

Como se puede observar en la figura anterior, las únicas conexiones que son compartidas son las de reloj, el resto son completamente independientes. Realizando la unificación de los módulos de esta manera se pueden utilizar microcontroladores externos para comunicarse con la FPGA

siempre que se utilice el valor de los 9600 bps y un protocolo de comunicación UART. La programación del módulo top se muestra en la Figura 129.

Figura 129.

Creación del módulo Top para unificar el Rx y Tx del protocolo UART

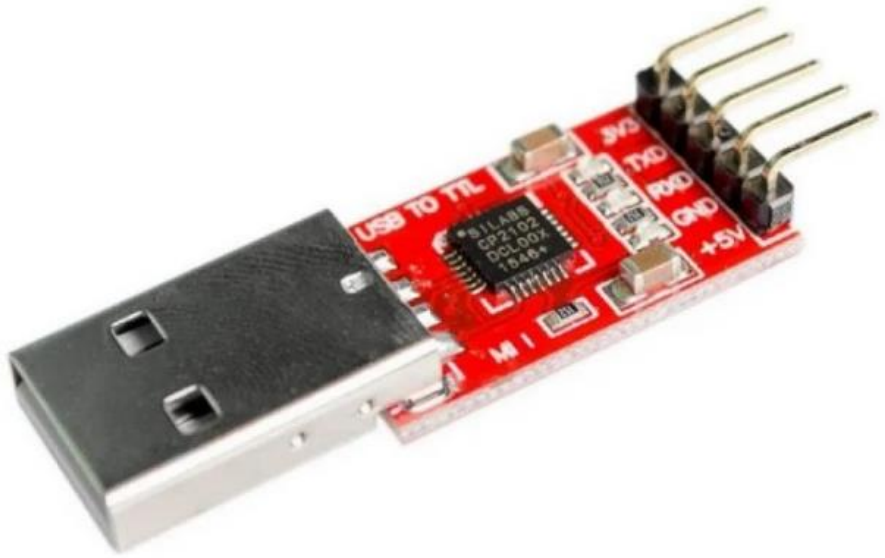
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
entity top is
      port(
         clk :     in STD_LOGIC;
         btn:      in STD_LOGIC;
         rx_in :   in STD_LOGIC;
         tx_in:    in STD_LOGIC_VECTOR(7 downto 0);
         rx_out :  out STD_LOGIC_VECTOR(7 downto 0);
         tx_out:   out STD_LOGIC
      );
end top;
architecture Behavioral of top is
begin
   Tx_9600:
      entity work.UART_TX
         generic map(
            contador_bits => 1250,
            n_bits => 8
         )
         port map(
            clk => clk,
            push_envio => btn,
            tx_datos => tx_in,
            tx_salida => tx_out
         );
   Rx_9600:
      entity work.UART_RX
         generic map(
            n_bits => 8,
            contador_bits=> 1250,
            contador_half_bits=> 625
         )
         port map(
            clk => clk,
            rx_in => rx_in,
            rx_data =>rx_out
         );
end Behavioral;
```

Nota. En la figura se puede observar la programación del módulo Top del protocolo de comunicación UART. Elaboración propia, realizado con ISE Design Suite 14.

Los puertos rx_in y tx_out son los que tienen las funciones principales del módulo que son la lectura y escritura de información, el resto de las señales pueden ser asignados a los puertos que más se acomoden al uso, aunque siempre será recomendable asignar el vector tx_in a los dip-switches y el vector rx_out a los leds de la FPGA.

Figura 130.

Imagen del módulo USB a TTL CP2102



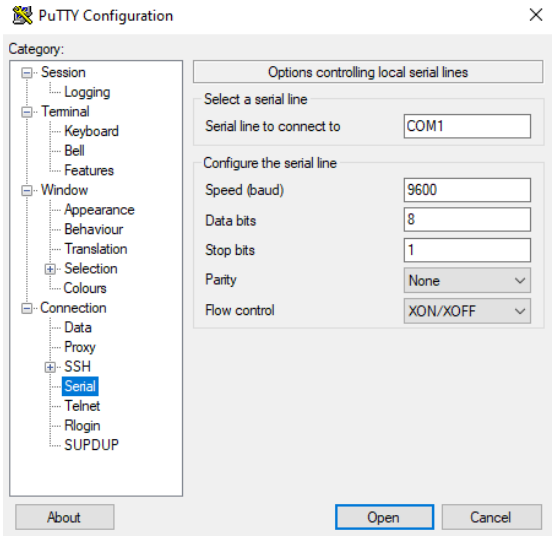
Nota. En la figura se puede observar la forma que tiene el módulo CP2102 que es utilizado en la mayoría de los casos como puente de comunicación entre tarjetas y computadoras. Obtenido de La Electrónica. *MÓDULO CONVERTIDOR USB A TTL CP2102 - ROJO*. (https://laelectronica.com.gt/modulo-convertidor-usb-a-ttl--cp2102), consultado el 09 de agosto de 2023. De dominio público.

Para realizar las pruebas del programa se puede utilizar el módulo CP2102 que se observa en la figura anterior a este párrafo. Dicho módulo puede realizar conversiones de señales TTL a USB, utilizando un controlador que puede ser descargado de la página oficial de Silicon Labs se puede configurar para que sea capaz de recibir señales que utilizan el protocolo UART, para hacerlo bastara con conectar el Rx del módulo al Tx de la FPGA y el Tx de la FPGA al Rx del módulo para realizar las pruebas de lectura y escritura. Para realizar el envío y lectura de los datos del módulo UART es necesario utilizar un programa que controle el flujo de datos de los terminales de la computadora, puede ser utilizado el programa PuTTY como referencia

para la prueba, la configuración del lector de los puertos seriales debe ser idéntica a la mostrada en la Figura 131.

Figura 131.

Configuración del puerto serial para usarlo con el módulo UART de la FPGA



Nota. En la figura se puede observar la configuración que debe ser colocada en el *software* PuTTY o similares para lograr una comunicación exitosa con el módulo UART e la FPGA usando el módulo CP2102. Elaboración propia, realizado con PuTTY.

En caso se necesite un programa que pueda leer y enviar datos a la FPGA por medio del módulo CP2102 utilizando el protocolo UART solo bastara con realizar la misma configuración mostrada en PuTTY para que las dos partes se encuentren sincronizadas y no existan errores en la comunicación.

4.6. Uso del módulo VGA en VHDL

La tarjeta Elbert V2 posee dentro de los módulos un puerto VGA que puede ser controlado por medio de código utilizando VHDL. Para lograr un

correcto control del puerto se explicará el funcionamiento general del VGA instalado y después se demostrará el código a utilizar.

4.6.1. Funcionamiento y descripción del VGA

El VGA o puerto VGA es una matriz de gráficos de video que puede convertir la información de bits en una forma gráfica. VGA proviene del acrónimo de Video Graphics Array que significa matriz de gráficos de video, este tipo de puerto fue desarrollado en el año 1987 por IBM en una versión estándar que puede soportar una resolución de 640 x 480 pixeles. La versión que posee la Elbert V2 es la más básica de todas, soportando únicamente la resolución 640 x 480 pixeles.

La forma en que los puertos VGA interpretan la información que reciben es un tanto especial, poseen dos hileras de control para lograr desplegar la imagen y solamente se puede imprimir información en ciertas posiciones de las hileras. Las hileras son HSync, quien controla el eje horizontal, y VSync, quien controla el eje vertical. Para una configuración de resolución 640 x 480 se necesita de un reloj de 25 MHz para que pueda funcionar correctamente, se menciona el reloj porque de esto también depende la distribución de pixeles dentro de un display VGA. En la figura 132 se puede observar la distribución de pixeles, el área que abarca el display es solo una pequeña porción de la imagen completa ya que existen áreas donde se realiza la sincronización de video y los cambios de estados de los puertos HSync y VSync, por lo tanto, para poder desplegar una figura dentro del área de resolución 640 x 480 hace falta rellenar ciertos pixeles que no han sido mencionados aún.

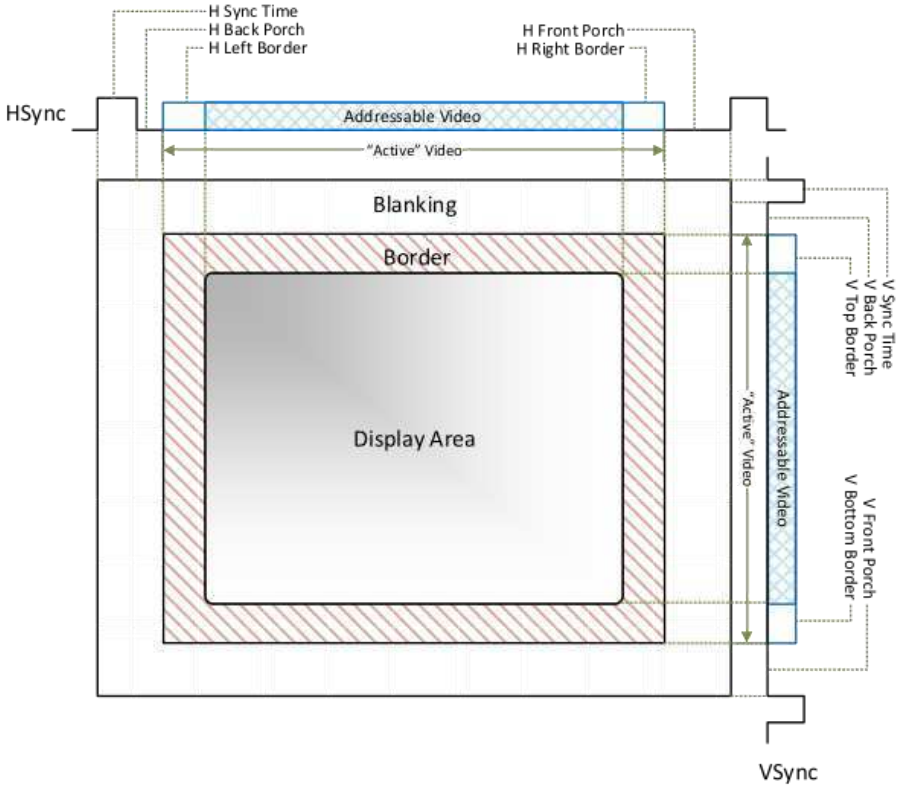
Para los dos puertos de sincronización del módulo VGA de la Elbert V2 se necesita que inicien en un nivel siempre alto o en su defecto en 1. Para el

puerto HSync inicialmente el valor de la señal cambia a 0 y se requiere que transcurra un total de 96 pixeles en el tramo llamado HSync Time que es el que inicia la sincronización, posterior a este tiempo se coloca de nuevo en un estado 1 la señal y debe transcurrir otros 48 pixeles en el tramo llamado Back Porch que es el tramo que marca los bordes del área del display y al transcurrir estos últimos dos tramos se pueden desplegar los 640 pixeles correspondientes a la imagen a desplegar. Una vez se ha desplegado la información en la imagen se debe realizar un último conteo de 16 pixeles el cual es llamado Front Porch que cierra el borde utilizado por el HSync, en total se deben ejecutar 800 pixeles de información que solo cambian a 0 en el primer tramo de pixeles.

Para el caso del VSync los pixeles son en realidad líneas que marcan la parte vertical y cambio de fila del display. Al iniciar se debe realizar el cambio a 0 solo para recorrer 2 líneas para el tramo VSync Time para iniciar la sincronización, al completar el recorrido se coloca la señal de nuevo a 1 y después se recorren otras 33 líneas para recorrer el tramo del borde Back Porch. Se realiza el despliegue de las 480 líneas y finalmente se cierra el borde ejecutando un recorrido de 10 líneas para completar con el despliegue de los datos, se realiza el recorrido de 525 líneas en total.

Figura 132.

Distribución del display para una resolución de 640 x 480



Nota. En la figura se puede observar la distribución de los pixeles para entregar una imagen con resolución 640 x 480. Obtenido de Digilent. *VGA Display Controller* (https://digilent.com/reference/learn/programmable-logic/tutorials/vga-display-congroller/start), consultado el 14 de agosto de 2023. De dominio público.

Es importante tomar en cuenta los desplazamientos iniciales de los pixeles y las líneas para poder desplegar la información en el área correcta, las configuraciones y desplazamientos explicados solo funcionan para el VGA con resolución 640 x 480 extraído de la página Digilet, si se requiere otro tipo de despliegue los desplazamientos deben ser variados.

Tabla 6.

Pixeles y líneas para desplegar una imagen en resolución 640 x 480

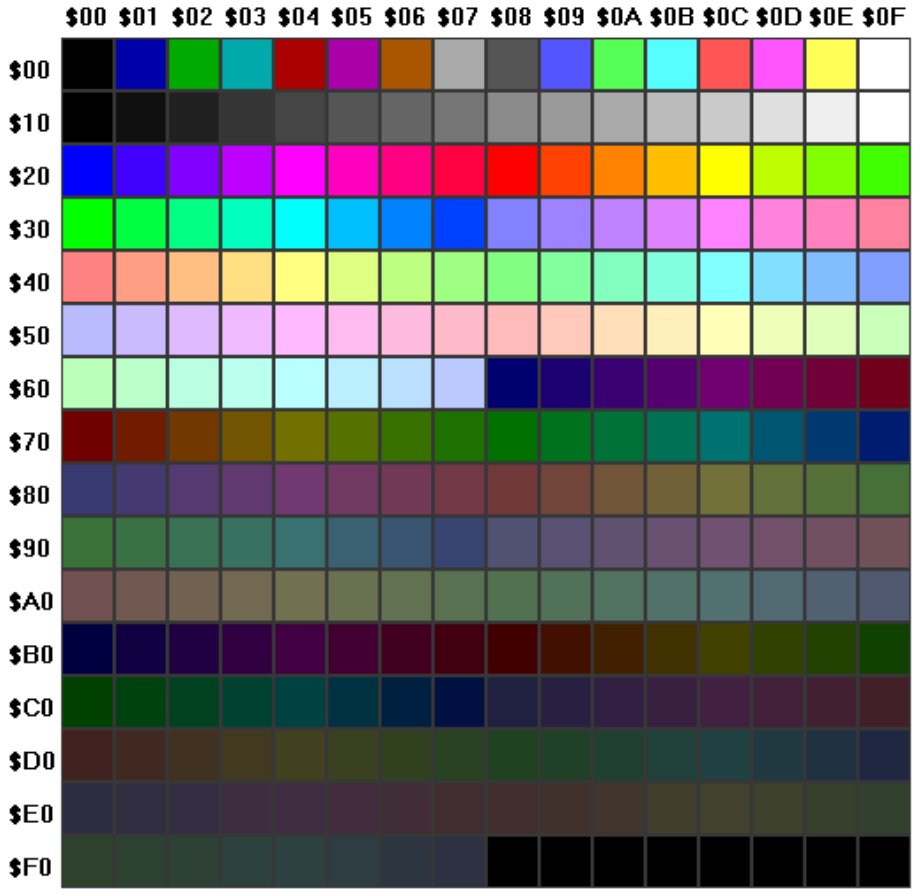
Paso	Sincronización	Back Porch	Display	Front Porch
HSync	96	48	640	16
VSync	2	33	480	10

Nota. El número de pausas y conteos que se debe esperar están relacionados con la frecuencia del reloj de 25 MHz que se utiliza para el módulo, pero para simplificar el concepto se han representado en forma de pixeles y líneas en lugar de periodos de tiempo. Elaboración propia, realizado con Excel.

La gama de color que puede manejar el VGA de la Elbert V2 es un RGB de 8 bits. El RGB hace referencia a que utiliza una mezcla de tres colores; rojo, verde y azul. La Elbert V2 tiene asignado en su UCF un vector de 3 bits para el color rojo y verde, mientras que para el color azul solo se encuentra disponible un vector de 2 bits. Combinando los valores de los tres colores y alterando el valor de los bits es posible desplegar 256 colores siguiendo correctamente los valores para cada uno de los vectores de los colores.

Figura 133.

Paleta de colores RGB en modo de 8 bits



Nota. En la figura se puede observar la paleta de colores que pueden tener las combinaciones de RGB de 8 bits. Obtenido de Fountainware. *VGA Color Palettes. (*https://www.fountainware.com/EXPL/vga_color_palettes.htm), consultado el 23 de agosto de 2023. De dominio público.

4.6.2. Programación del código del control del VGA en VHDL

Para lograr el correcto funcionamiento del módulo VGA en la Elbert V2 se necesita utilizar un reloj de 25 MHz en conjunto con el grupo de

instrucciones, para facilitar la explicación se dividirá el capítulo en tres partes, siendo la primera el código que controla el VGA, la segunda la forma de agregar un módulo que permita el uso de un reloj de 25 MHz y la tercera será la combinación del reloj con el módulo.

4.6.2.1. Código del módulo VGA en VHDL

Para lograr realizar el recorrido de la escritura de información en un display VGA se utilizará como referencia la explicación del capítulo 4.6.1. Tal como fue explicado en la Tabla 6, para lograr desplegar una resolución de 640 x 480 se debe hacer uso de un recorrido mayor que permita realizar la sincronización, por este motivo se utilizaran dos señales genéricas para asignar estos valores. Adicionalmente se deben programar los puertos que serán utilizados a modo de entradas y salidas para garantizar el funcionamiento del VGA. Los puertos y señales que serán declaradas son:

- Count_h: es la señal genérica que contendrá el valor entero 800 que permitirá funcionar como delimitador para el recorrido de los pixeles.

- Count_v: s la señal genérica que contendrá el valor entero 525 que permitirá funcionar como delimitador para el recorrido de las líneas.

- Clk: es el puerto en el cual se conectará el reloj de 25 MHz.

- HSync: es el puerto que se conectara a la salida que controle la sincronización horizontal del módulo VGA.

- VSync: es el puerto que se conectara a la salida que controle la sincronización vertical del módulo VGA.

- Red: es el puerto que conectara al color rojo del módulo VGA.

- Green: es el puerto que conectara al color rojo del módulo VGA.

- Blue: es el puerto que conectara al color rojo del módulo VGA.

Adicional a las señales anteriores, se deben crear dos señales auxiliares que realizaran el recorrido de los pixeles y las líneas dentro de la pantalla, la declaración de todas las señales se puede observar en la siguiente imagen.

Figura 134.

Entidad VGA_main con las señales genéricas y puertos a utilizar

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  entity VGA_main is
4     generic (
5        count_h : integer := 800;
6        count_v : integer := 525
7     );
8     port(
9        clk :   in STD_LOGIC;
10       HSync : out STD_LOGIC;
11       VSync : out STD_LOGIC;
12       Red   : out STD_LOGIC_VECTOR(2 downto 0);
13       Green : out STD_LOGIC_VECTOR(2 downto 0);
14       Blue  : out STD_LOGIC_VECTOR(2 downto 1)
15    );
16 end VGA_main;
17 architecture Behavioral of VGA_main is
18    signal pixel  : integer range 0 to count_h-1;
19    signal line   : integer range 0 to count_v-1;
```

Nota. En la figura se puede observar la entidad creada VGA_main junto con las librerías y puertos necesarios para su funcionamiento. Elaboración propia, realizado con ISE Design Suite 14.

Una vez se han declarado las señales el próximo paso es realizar la programación de los pasos mencionados en el capítulo 4.6.1. Lo primero que debe ser programado es la delimitación de los contadores de pixeles y las líneas, para realizarlo se debe realizar primero una condición que si el contador pixel iguala al valor de count_h para saber si se han recorrido las 800 columnas, si es el caso realizara la suma del contador line para hacer el cambio de línea. Dentro del cambio de línea también se encuentra una segunda condición que evalúa el contador de line y lo compara con count_v para determinar si se han desplazado las 525 lineas del VGA para realizar un reset de line. Si en algún caso el contador pixel no ha llegado aún al valor de 800 la primera condición falla y solo realiza la suma del valor pixel.

Una vez se establece el delimitador de los rangos máximos se realiza el conteo de la sincronización de los pixeles y las líneas. Dicha sincronización se realiza en la línea 33 y 38, se mantendrán en un valor bajo siempre que se encuentren dentro del rango de sincronización que son 96 para los pixeles y 2 para las líneas.

Figura 135.

Delimitación de los rangos de los pixeles y las líneas

```
20  begin
21     process (clk)
22     begin
23        if (rising_edge(clk)) then
24           if (pixel = (count_h-1)) then
25              pixel <= 0;
26              line <= line + 1;
27              if (line = (count_v-1)) then
28                 line <= 0;
29              end if;
30           else
31              pixel <= pixel + 1;
32           end if;
33           if (pixel >= 96) then
34              HSync <= '1';
35           else
36              HSync <= '0';
37           end if;
38           if (line >= 2) then
39              VSync <= '1';
40           else
41              VSync <= '0';
42           end if;
```

Nota. En la figura se puede observar las condiciones que deben cumplirse para delimitar los rangos y la sincronización para los puertos HSync y VSync del módulo VGA. Elaboración propia, realizado con ISE Design Suite 14.

El último paso es realizar el dibujo de la imagen que se desea desplegar, esto se realiza solo dentro del límite permitido. Para las líneas (movimiento vertical) el rango del contador en donde se puede dibujar es a partir de 35 y antes del valor 515, con esto se abarcan las 480 lineas del VGA. Para el caso de los pixeles (movimiento horizontal) el rango del contador donde se puede dibujar es a partir de 114 y antes de 784, con esto se logran abarcar los 640 pixeles para el VGA.

Para el ejemplo se empleará un rectángulo que abarque desde los pixeles 200 a la 400 y que utilizará las líneas desde la 150 a la 300. Para el ejemplo se utiliza un color azul y por este motivo solo a este vector se le asigna un valor. El ejemplo se puede observar en la siguiente imagen.

Figura 136.

Delimitación la figura a desplegar activando los vectores de los colores

```
43          if ((pixel >= 200) and (pixel <= 400) and (line >= 150) and (line <= 300)) then
44             Red <= "000";
45             Green <= "000";
46             Blue <= "11";
47          else
48             Red <= "000";
49             Green <= "000";
50             Blue <= "00";
51          end if;
52       end if;
53    end process;
54
55 end Behavioral;
```

Nota. En la figura se puede observar las condiciones que deben cumplirse para delimitar los rangos y la sincronización para los puertos HSync y VSync del módulo VGA. Elaboración propia, realizado con ISE Design Suite 14.

Como se ha podido observar, la última parte del código es la que puede ser modificada para poder crear diferentes figuras, incluso puede ser reemplazada por una memoria que contenga imágenes más elaboradas y con diferentes colores.

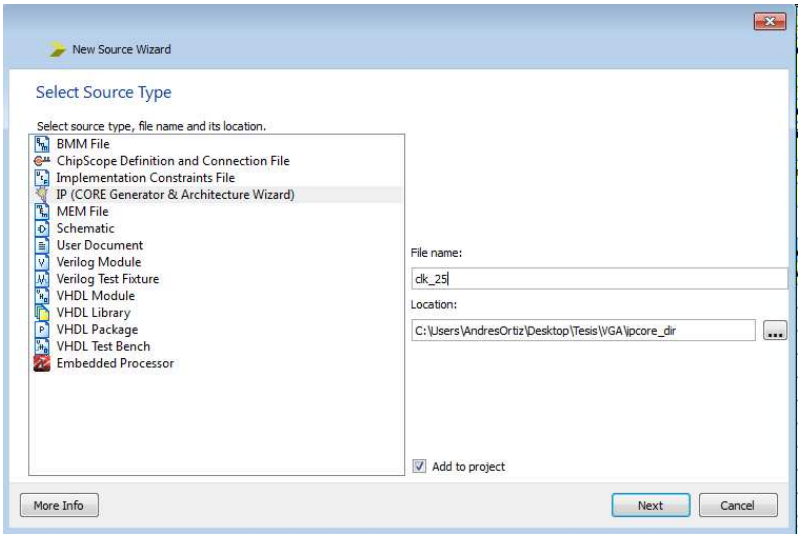
4.6.2.2. Módulo del reloj de 25MHz en VHDL

En las aplicaciones que han sido programadas con anterioridad se utiliza el reloj de 12 MHz interno de la Elbert V2. Para el módulo VGA requiere el uso de un reloj de mayor frecuencia para que funcionen las tazas de refresco y la sincronización de la imagen, dicha frecuencia debe ser de 25 MHz. Existe una forma de utilizar un duplicador de frecuencia en la tarjeta para simular un cambio de frecuencia de reloj utilizando los medios ciclos del reloj interno de 12 MHz. Para realizar dicha acción se debe crear un archivo nuevo utilizando la opción New Source al seleccionar el módulo principal. Se debe seleccionar la opción IP (CORE Generator & Architecture Wizard) y se colocara el nombre

de clk_25. En la siguiente imagen se puede observar la forma correcta de realizar la creación.

Figura 137.

Creación del nuevo archivo llamado clk_25

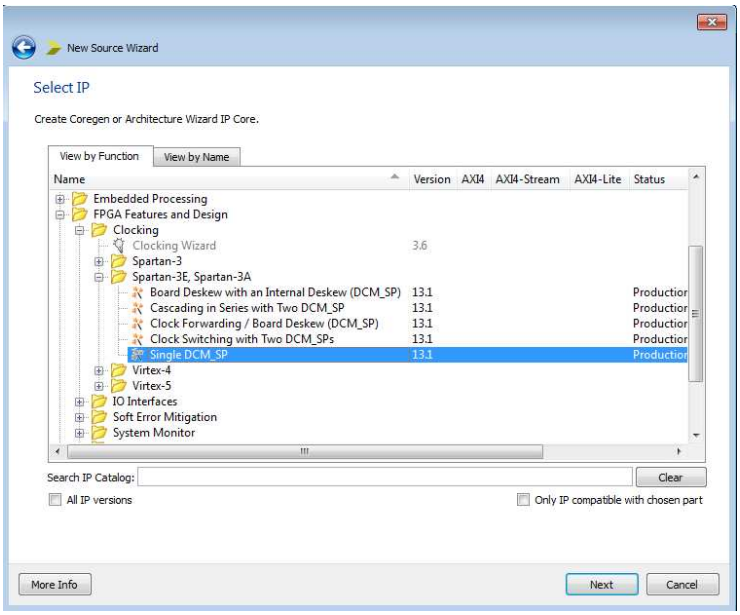


Nota. En la figura se muestra la opción que debe ser seleccionada para implementar correctamente el reloj de 25 MHz. Elaboración propia, realizado con ISE Design Suite 14.

Una vez se pulsa el botón Next el programa despliega una nueva pestaña que pide seleccionar un módulo. La creación de la arquitectura del nuevo reloj debe ser seleccionada dentro del grupo de carpetas que despliega ISE Design Suite. El módulo se encuentra dentro de la carpeta FPGA Features and Design/Spartan-3E, Spartan-3A y tiene el nombre de Single DCM_SP.

Figura 138.

Selección del módulo Single DCM_SP

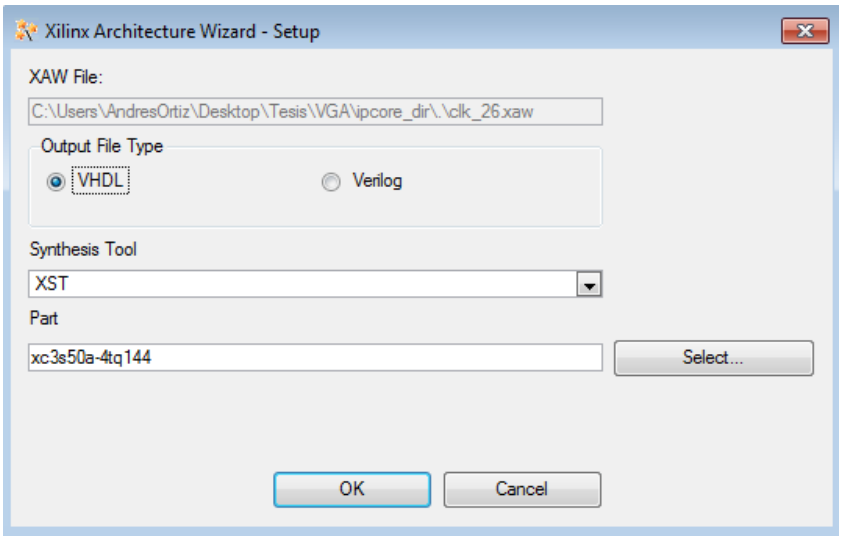


Nota. En la figura se muestra la opción que debe ser seleccionada para implementar correctamente el reloj de 25 MHz. Elaboración propia, realizado con ISE Design Suite 14.

Una vez se selecciona el módulo el programa ISE despliega un resumen de lo que será creado, bastará con seleccionar el botón Finish para que despliegue el siguiente paso. La creación del módulo puede tardar un poco dependiendo la capacidad del computador que se está utilizando, una vez carga despliega una pestaña en la cual se debe seleccionar el lenguaje a utilizar, el sintetizador y el chip. Esto se puede observar en la siguiente imagen.

Figura 139.

Opciones iniciales con la selección del lenguaje, sintetizador y chip a utilizar

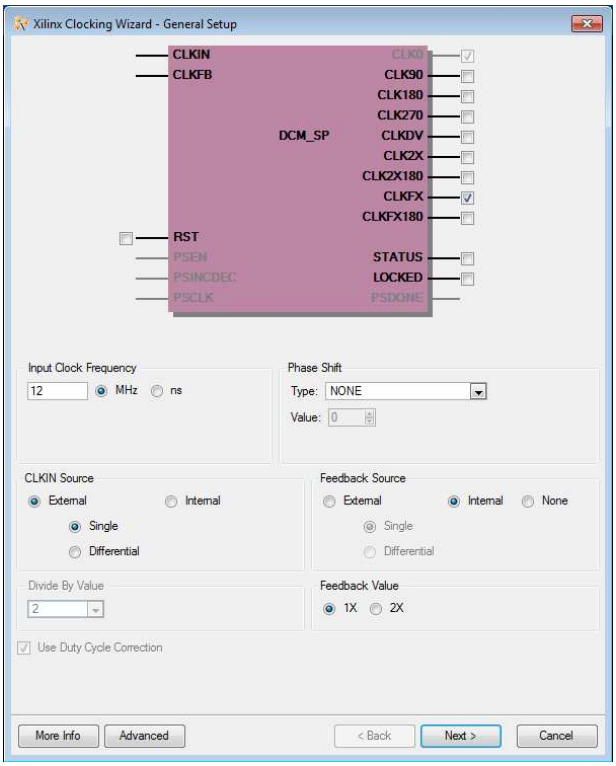


Nota. En la figura se muestra la pestaña que se despliega con las configuraciones iniciales del nuevo módulo de reloj, se puede dejar las opciones por defecto ya que ISE toma las mismas configuraciones iniciales que se colocan al iniciar un nuevo proyecto. Elaboración propia, realizado con ISE Design Suite 14.

Al seleccionar la opción Ok el programa despliega una configuración general del módulo de reloj, se deben colocar ciertos valores para poder adaptar el módulo de 25 MHz correctamente. El primer paso debe ser establecer la frecuencia del reloj de entrada, se deben colocar los 12 MHz en la pestaña Input Clock Frequency. El segundo paso es desactivar los puertos LOCKED y RST, además de activar el puerto CLKFX que será el puerto que contiene la señal de 25 MHz de salida. El resto de las configuraciones no debe de ser modificada ya que alteraría el propósito del módulo, se puede observar un ejemplo de las configuraciones en la imagen mostrada a continuación.

Figura 140.

Frecuencias de entrada y puertos del módulo de reloj de 25 MHz

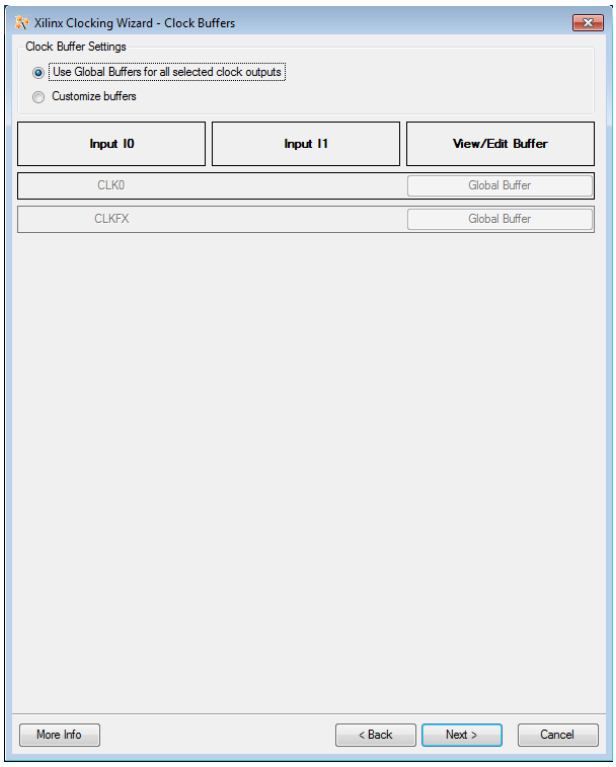


Nota. En la figura se muestra el módulo del reloj en el cual se deben colocar los puertos a utilizar y la frecuencia del reloj de entrada, se puede utilizar la imagen de referencia para la creación de este tipo de módulos. Elaboración propia, realizado con ISE Design Suite 14.

La siguiente pestaña es para realizar modificaciones al buffer del reloj, para la aplicación que será utilizada no es necesario una configuración especial y puede observarse en la siguiente imagen la configuración por defecto que será utilizada.

Figura 141.

Configuración por defecto del buffer del reloj de 25 MHz

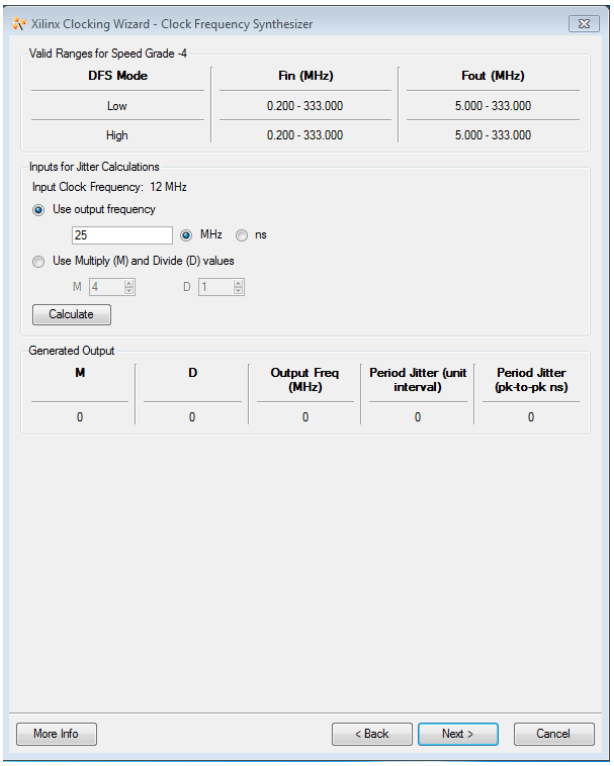


Nota. En la figura se muestra la configuración por defecto del buffer del módulo de reloj. Elaboración propia, realizado con ISE Design Suite 14.

El siguiente paso muestra una pestaña en donde se debe colocar la frecuencia de salida, es en este punto en donde se colocará la frecuencia de 25 MHz a utilizar por el módulo VGA. El valor debe ser colocado en Use output frequency y será lo único que se modificara. ISE realiza internamente los cálculos para obtener esta salida de reloj, simplemente fracciona los ciclos de 12 MHz a un equivalente que pueda entregar la misma señal, pero con ciclos de 25 MHz. La imagen en donde se pueden observar las configuraciones realizadas en la pestaña.

Figura 142.

Configuración de la salida del reloj de 25 MHz



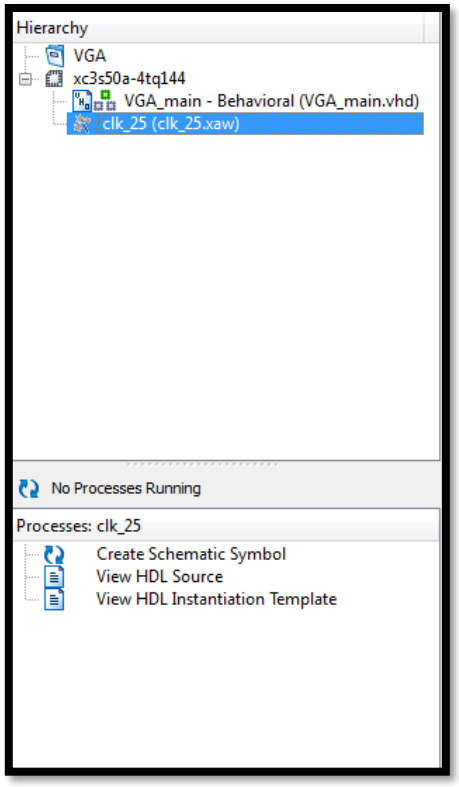
Nota. En la figura se muestra la configuración de la salida de 25 MHz del módulo de reloj. Elaboración propia, realizado con ISE Design Suite 14.

Una vez se han completado los pasos de configuración ISE despliega una pestaña de resumen en donde se puede observar las configuraciones realizadas, bastara con seleccionar el botón Finish para que ISE cree el módulo.

Al completar los pasos mencionados con anterioridad aparecerá dentro de los módulos del proyecto el nuevo reloj, este reloj será utilizado dentro de un módulo Top para que entregue la señal de reloj al módulo VGA.

Figura 143.

Conjunto de módulos VGA y el reloj de 25 MHz



Nota. En la figura se muestra la forma en que deberían de quedar los módulos una vez se finaliza con la configuración del módulo del reloj de 25 MHz. Elaboración propia, realizado con ISE Design Suite 14.

4.6.2.3. Módulo Top del VGA

Para este caso el módulo Top se crea únicamente para enlazar el módulo de reloj con el módulo VGA, para realizar dicha tarea se utilizan las mismas entradas y salidas vistas en el módulo principal del VGA y se crea una señal auxiliar llamada clk_25 que tendrá la señal del reloj de 25 MHz. La

declaración de puertos y señales para el módulo top queda de la siguiente manera.

Figura 144.

Entidad y arquitectura de top con las señales y puertos a utilizar

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  entity top is
4     port(
5        clk_12 :   in STD_LOGIC;
6        HSync : out STD_LOGIC;
7        VSync : out STD_LOGIC;
8        Red   : out STD_LOGIC_VECTOR(2 downto 0);
9        Green : out STD_LOGIC_VECTOR(2 downto 0);
10       Blue  : out STD_LOGIC_VECTOR(2 downto 1)
11    );
12 end top;
13
14 architecture Behavioral of top is
15    signal clk_25 : STD_LOGIC;
16 begin
```

Nota. En la figura se muestran las señales y puertos que serán utilizadas dentro de la arquitectura del módulo top. Elaboración propia, realizado con ISE Design Suite 14.

La única parte que debe ser mapeada para la entidad del reloj de 25 MHz son los puertos CLKIN_IN que debe ser asignado al reloj de 12 MHz que será asignado al reloj interno de la FPGA y el segundo puerto es el CLKFX_OUT que será asignado a la señal auxiliar clk_25. La señal auxiliar se asigna al mapeo de la entidad VGA y para el resto de puertos simplemente se realiza el mapeo siguiendo el nombre de la señal. El mapeo de las entidades se puede observar a continuación.

Figura 145.

Contenido de la arquitectura del módulo Top para el programa de VGA

```
17
18      VGA_mod:
19          entity work.VGA_main
20              port map(
21                  clk => clk_25,
22                  HSync => HSync,
23                  VSync => VSync,
24                  Red   => Red,
25                  Green => Green,
26                  Blue  => Blue
27              );
28      Inst_VGA_CLK:
29          entity work.clk_25
30              port map(
31              CLKIN_IN => clk_12,
32              CLKFX_OUT => clk_25
33              );
34
35  end Behavioral;
```

Nota. En la figura se muestran la arquitectura con los mapeos de las señales para las dos entidades que servirán para crear las imágenes VGA. Elaboración propia, realizado con ISE Design Suite 14.

Una vez se ha realizado el mapeo de las dos entidades se puede asignar los valores de salida al UCF de la tarjeta Elbert V2. Para la asignación de los valores simplemente se debe buscar la parte donde indique los puertos VGA y quitar los numerales para activar las casillas y así ya no aparecen como comentarios. En la siguiente figura se muestra la asignación de los puertos realizada al UCF de la tarjeta.

Figura 146.

Declaración de los puertos en el UCF de la Elbert V2

```
14    NET "clk_12"                LOC = P129  | IOSTANDARD = LVCMOS33 | PERIOD = 12MHz;
15  #
16  ##############################################################################################
17  ##                                        VGA
18  ##############################################################################################
19  #
20      NET "HSync"             LOC = P93   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE =
21      NET "VSync"             LOC = P92   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE =
22      NET "Blue[2]"           LOC = P98   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE =
23      NET "Blue[1]"           LOC = P96   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE =
24      NET "Green[2]"          LOC = P102  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE =
25      NET "Green[1]"          LOC = P101  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE =
26      NET "Green[0]"          LOC = P99   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE =
27      NET "Red[2]"            LOC = P105  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE =
28      NET "Red[1]"            LOC = P104  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE =
29      NET "Red[0]"            LOC = P103  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE =
```

Nota. En la figura se muestran el UCF y los puertos modificados, con esto ya se puede generar él. bit y cargar a la tarjeta Elbert V2. Elaboración propia, realizado con ISE Design Suite 14.

5. CONCLUSIONES SOBRE LA ELBERT V2 SPARTAN 3A Y LA TRANSICIÓN A LA CMOD A7-35T

A lo largo de los capítulos anteriores se ha podido observar las capacidades de la FPGA Elbert V2 Spartan 3A. Se ha demostrado que al realizar aplicaciones de pocos requerimientos se puede hacer de forma sencilla haciendo uso de la tarjeta usando lenguaje VHDL, sin embargo, se debe evaluar si vale la pena una evolución a la siguiente generación debido a la antigüedad del chip XC3S50A que contiene la tarjeta Elbert V2, por este motivo, en este capítulo explorará la posibilidad de migrar a un sistema más reciente evaluando las fortalezas y debilidades de la Elbert V2 realizando la comparación con la FPGA CMOD A7-35T.

5.1. Comparación técnica de los chips

En el primer capítulo se expuso un resumen general de los módulos que posee la Elbert V2 pero no se mencionaron características del chip XC3S50A que integra a la tarjeta como la corriente que soportan o entregan los GPIOs, la capacidad de memoria y demás características que influyen directamente en el rendimiento técnico de la tarjeta.

Para el caso de la tarjeta CMOD A7-35T utiliza un chip de la familia Artix-7 llamado XC7A35T. Se evaluarán las mismas características para los dos chips y con esto se pretende tener una mejor visibilidad de la capacidad de cada una de las tarjetas.

5.1.1. Características del chip XC3S50A utilizado por la Elbert V2

Al mencionar las características relacionadas con la Elbert V2 se deben mencionar las características del chip que integra a la tarjeta. Tal como se menciona al inicio de este trabajo, el chip utilizado es el XC3S50A perteneciente a la familia Spartan 3A del fabricante Xilinx.

5.1.1.1. Celdas lógicas y procesamiento

Las celdas lógicas en un chip FPGA hace referencia a una unidad estándar de capacidad lógica y es un equivalente a decir cuántas compuertas lógicas simples pueden realizarse utilizando un determinado chip. En el caso del chip XC3S50A posee una capacidad de celdas lógicas equivalente a 1584 celdas, pero se debe mencionar que los modelos 3A pueden llegar a tener hasta 25344 celdas en los empaques más robustos y grandes.

Otra característica importante dentro de las capacidades de procesamiento de una FPGA son las llamadas DSP o Digital Signal Processing por sus siglas en ingles. Las DSP sirven para procesar y analizar señales digitales como las señales de video, audio y demás señales dedicadas a la telecomunicación, aceleran el rendimiento de aplicaciones que estén dedicadas a los campos mencionados. La familia de chips 3A solo tiene disponible este tipo de mejora en los chips Spartan 3A DSP. El chip XC3S50A no cuenta con este tipo de mejora y esto limita un poco a la tarjeta a la hora de analizar señales digitales complejas. Si se desea utilizar la Elbert V2 para alguna aplicación de este tipo se recomienda utilizar módulos auxiliares que ayuden a la tarjeta con este tipo de procesamiento.

5.1.1.2. GPIOs y sus características

El chip Spartan 3A en su presentación XC3S50AN, cuenta con una capacidad de 144 pines entrada/salida. De los 144 pines que se encuentran en el chip XC3S50AN la tarjeta Elbert V2 solamente utiliza 39 dedicados exclusivamente a los puertos de salida y entrada, el resto de los pines son utilizados para conectar con el resto de los módulos de la Elbert V2 como lo son los leds, botones push, puerto VGA, entre otros. Adicionalmente se debe mencionar que los pines tienen una propiedad llamada Hot Swap, esto permite conectar y desconectar componentes a los puertos de la tarjeta sin necesidad de apagarla y sin que sufra algún tipo de daño en el proceso.

Al momento de configurar los pines como una entrada pueden utilizar dos configuraciones de resistencia de entrada, estas son la resistencia pull-up y pull-down. Los rangos de voltaje y su resistencia se muestran en la siguiente tabla.

Tabla 7.

Rango de voltaje y resistencia de los GPIOs de entrada la tarjeta Elbert V2

Rango de Voltaje (V)	Rango de Resistencia Pull-Up (k Ω)
3.0 a 4.6	1.27 a 4.11
2.3 a 2.7	1.15 a 3.25
1.7 a 1.9	2.45 a 9.10

Nota. Los valores que se muestran en la tabla fueron extraídos de la hoja técnica de la familia de chips Spartan 3A. Elaboración propia, realizado con Excel.

Las corrientes que soportan los pines configurados como entradas son 2, 4, 6, 8, 12, 16 y 24 mA, dichos rangos son los estándares de señales de

entrada/salida utilizados por la industria. En la documentación oficial de Xilinx son conocidos como los IOSTANDARD.

5.1.1.3. Características de la RAM

Las FPGAs utilizan una memoria RAM es similar a la que utilizan las computadoras comunes, la diferencia se encuentra en el acceso a dichas memorias. En una FPGA se puede utilizar la memoria RAM para interacciones directas que pueden ser configuradas por el programador, ya que su implementación se encuentra en forma de bloques de memoria dentro del chip de la FPGA.

Tal como lo especifica la documentación oficial del chip Spartan 3A, la RAM de la tarjeta se encuentra distribuida en los bloques de memoria SelectRAM. Dichos bloques pueden ser programados utilizando instancias especiales o las instancias que se pueden crear en las funciones de Xilinx CORE Generator.

El chip XC3S50 posee únicamente 4 bloques de memoria SelectRAM, estos bloques pueden utilizarse para almacenar datos temporales de algún tipo de información que se reciba de los GPIOs o de algún otro medio para que sean utilizados en algún programa o función con VHDL. Cada uno de los bloques posee la capacidad de almacenar 16 KB de datos, lo que brinda las siguientes combinaciones de organizaciones de memoria:

- 16 KB x 1
- 8 KB x 2
- 4 KB x 4
- 2 KB x 8

- 1 KB x 16

Se mencionan las combinaciones porque al formar los bloques de memoria dentro de los módulos de VHDL se debe considerar el tamaño por los bloques SelectRAM de la Elbert V2. La memoria RAM total de la tarjeta es de 72 KB, aunque 8 KB de esta memoria es utilizada para los procesos de arranque y almacenamiento inicial de la tarjeta. Adicionalmente, la versión de la RAM es una DDR2 con la capacidad de lectura y escritura de 400 MB/s.

5.1.2. Características del chip XC7A35T utilizado por la CMOD A7-35T

La tarjeta CMOD A7-35T está equipada con el chip Artix-7 XC7A35T fabricado por la compañía Xilinx. Este tipo de familia de chips salió al mercado en el año 2010 según la documentación oficial de Xilinx y en el año 2016 vio la luz la versión utilizada por el CMOD. Desde ese entonces ha recibido varias actualizaciones que dan más detalles del componente, siendo en octubre del 2020 la última actualización recibida.

5.1.2.1. Celdas lógicas y procesamiento

Para el chip XC7A35T la compañía Xilinx incluyó un total de 33280 celdas lógicas. El XC7A35T es el segundo chip más potente de la familia Artix-7 en la categoría de los que miden 10x10 mm, lo que lo convierte en un chip ideal para su uso en tarjetas de reducido tamaño como lo es la CMOD.

A pesar del reducido tamaño el chip, posee la tecnología de procesamiento digital DSP. Dentro del XC7A35T se encuentran 90 segmentos DSP, cada segmento contiene un multiplicador 25 x 18, un sumador y un

acumulador. Estas funciones permiten un procesamiento de múltiples señales digitales que funcionan de forma paralela y al tener un segmento dedicado a dicho procesamiento hacen más eficientes algunas tareas. Además, la presencia de DSP abre la puerta a poder crear módulos internos que traten el uso de señales digitales pesadas, como lo puede ser el procesamiento de audio, video, modulaciones y filtrado de señales digitales dedicas a las telecomunicaciones.

5.1.2.2. GPIOs y sus características

El chip XC7A35T tiene un modo un tanto diferente de funcionar comparado con el chip XC3S50. Mientras que las entradas del chip XC3S50 funcionan con resistencias a modo de pull-ups y pull-downs para el ingreso y entrega de los voltajes, el XC7A35T funciona con una tecnología llamada DCI.

La tecnología DCI trata de una impedancia controlada de forma digital que ajusta el valor de referencia de los pines de entrada y salida tomando en consideración las variaciones de temperatura y el voltaje de suministro. La forma en que funciona el DCI es utilizando un pin llamado VRP, este pin es de referencia y está conectado a tierra utilizando una resistencia de 240 Ω. La calibración que realiza el circuito activa y desactiva de forma selectiva las resistencias de entrada y salida, permitiendo de esta forma tomar las lecturas.

Gracias al funcionamiento de los DCI, los rangos de voltaje se simplifican únicamente a un solo rango, el cual abarca desde los -0.20 V hasta los 3.665 V. Los rangos de corriente siguen el mismo estándar IOSTANDARD de Xilinx, siendo las corrientes de 4, 8, 12, 16 y 24 mA.

5.1.2.3. Características de la RAM

La memoria RAM del chip XC7A35T funciona de la misma forma que la del XC3S50. La diferencia es que la XC7A35T al ser más moderna tiene la capacidad de brindar un único bloque de 36 KB, o utilizar el mismo bloque dividido en dos bloques independientes de memoria RAM, cada uno de 18 KB. En el caso de la XC7A35T, si se utiliza el bloque de 36 KB se tiene la opción de tener 50 bloques de memoria y si se utiliza la opción de los dos bloques de 18 KB se puede tener 100 bloques en total. Independientemente de la configuración utiliza, la capacidad máxima de memoria es de 1.8 MB.

Esta memoria también puede ser configurada en distintas combinaciones de organizadores de memoria, siendo las combinaciones recomendadas las siguientes:

- 32 KB x 1
- 16 KB x 2
- 8 KB x 4
- 4 KB x 9
- 2 KB x 18
- 1 KB x 36
- 512 B x 72

Se hace evidente que el chip XC7A35T tiene una mayor capacidad de distribución y organización de memoria que su contraparte. Además de esto, posee una memoria RAM más rápida teniendo la tecnología DDR3 equipada, capaz de realizar procedimientos de lectura y escritura a una velocidad de 1866 MB/s.

5.1.3. Comparación final

Los chips XC3S50A y el XC7A35T tienen una diferencia de creación de 7 años, siendo el XC3S50A del año 2007 y el XC7A35T del 2014. A pesar de que existe esta diferencia de años, para fines de laboratorio y pruebas, el chip XC3S50A posee características más robustas que lo hacen ideal para estos entornos.

Dentro de la comparación se excluye el tema de los rangos de corriente, ya que ambos chips utilizan el estándar IOSTANDARD de Xilinx. Entre las diferencias se encuentran el modo que tiene cada uno de los chips para realizar la lectura de las entradas. Mientras que el XC3S50A utiliza un sistema de resistencias pull-up y pull-down fijas, las cuales robustecen el modelo ante posibles cortos, mientas que el XC7A35T utiliza la tecnología de impedancia dinámica DCI, eso explica su menor tamaño, pero sacrifica la durabilidad que ofrecen los pull-up y pull-down. Se debe recalcar que la XC7A35T cuenta con una memoria RAM más grande y rápida que la del XC3S50A, esto ofrece una ventaja en el ámbito industrial al poder crear una mayor cantidad de módulos paralelos, lo cual marca una ventaja comparada contra el XC3S50A. El detalle y resumen de los datos se describen en la siguiente tabla.

Tabla 8.

Comparación final de las diferencias entre el chip XC3S50A y XC7A35T

Característica	XC3S50A	XC7A35T
Cantidad de Celdas Lógicas	1584	33280
Voltaje Máximo Admitido	4.6 V	3.7 V
Bloques RAM	4 bloques de 16 KB	50 bloques de 36 KB
Memoria RAM Total	72 KB	1800 KB
Generación de RAM	DDR2	DDR3

Nota. Los valores que se muestran en la tabla fueron extraídos de la hoja técnica de la familia de chips Spartan 3A y Artix 7, los detalles de las características fueron discutidos en los párrafos anteriores. Elaboración propia, realizado con Excel.

Al comparar los valores de forma directa es evidente la diferencia de potencia entre los dos chips, sin embargo, es importante resaltar que el chip XC3S50A es más económico que el XC7A35T y para aplicaciones de aprendizaje, es una excelente opción. La diferencia principal se marca en los bloques de celdas lógicas y RAM, el XC7A35T puede procesas mayor información, pero al carecer de una mayor protección en los puertos no es ideal para un entorno educativo en el cual lo común es cometer errores, lo que la hace no ser la mejor opción al menos para el sector educativo.

5.2. Comparación de la tarjeta Elbert V2 y la CMOD A7-35T

Una vez se han comparado los chips de las tarjetas se debe realizar la comparación de sus montajes. La tarjeta Elbert V2 y CMOD A7-35T tienen ciertas diferencias muy remarcables dentro de los módulos que integran. En este capítulo se explorará más a detalle los módulos del CMOD A7-35T y se

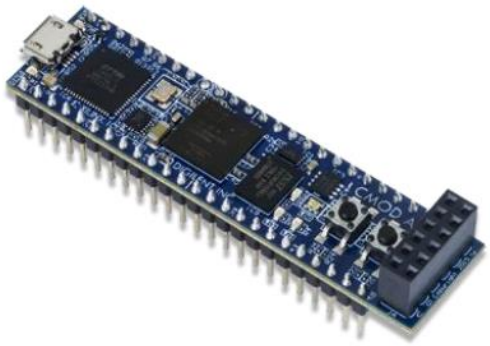
realizará un breve resumen del primer capítulo para hablar de los módulos de la Elbert V2.

5.2.1. Características tarjeta CMOD A7-35T

La FPGA CMOD A7-35T es una tarjeta ensamblada por la compañía Digilent Inc, fundada en el año 2000. Uno de los principales atractivos de la CMOD es su bajo costo, calidad de componente y capacidad de potencia que hace a la tarjeta una opción atractiva de entrada al mundo de las FPGA. La tarjeta fue desarrollada en el año 2016, cuenta con un chip de la familia de Xilinx Artix-7 teniendo el encapsulado XC7A35T.

Figura 147.

Apariencia de la FPGA CMOD A7-35T

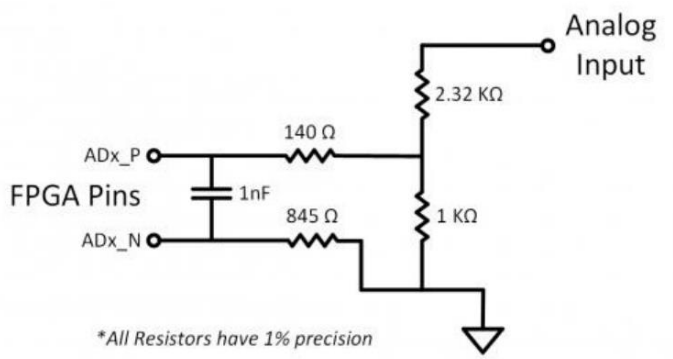


Nota. En la figura se puede observar la forma del ensamblaje para una FPGA CMOD A7-35T. Obtenido de Farnell. *Cmod A7 Reference Manual (*https://www.farnell.com/datasheets/2876438.pdf*),* consultado el 08 de octubre de 2023. De dominio público.

En el montaje de la tarjeta se tienen 48 pines de conector tipo DIP, de estos 48 se encuentran 44 como GPIOs y los otros 2 son entradas analógicas. Las entradas analógicas funcionan gracias a un módulo ADC Integrado de 12 bits el cual soporta rangos de voltaje de 0 hasta 3.48 V, mientras que los pines de salidas y entradas digitales pueden soportar máximo hasta 3.7V.

Figura 148.

Diagrama del módulo ADC que se encuentra en la FPGA CMOD A7-35T



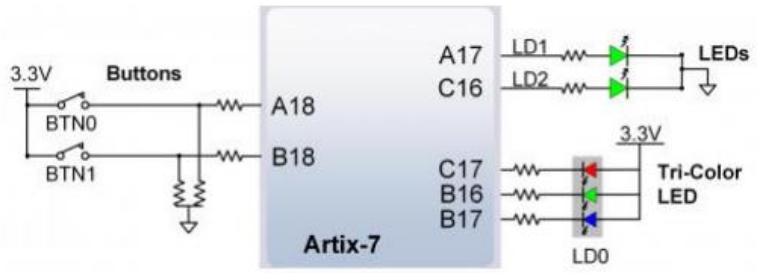
Nota. En la figura se puede observar un diagrama que presenta la forma en que se encuentra configurado el módulo ADC de la FPGA CMOD A7-35T para la lectura de los pines análogos. Obtenido de Farnell. *Cmod A7 Reference Manual (*https://www.farnell.com/datasheets/2876438.pdf*),* consultado el 08 de octubre de 2023. De dominio público.

De forma adicional, también cuenta con un módulo conector tipo Pmod que ofrece 8 pines más que funcionan como pines de entrada/salida. Se debe mencionar la existencia de los pines 24 y 25 de la tarjeta que funcionan como un VCC y un GND respectivamente. El pin 24 puede soportar un rango de voltajes de 3.32 como mínimo hasta 5.5 V máximo, este pin puede ser alimentado con una fuente externa de voltaje para alimentar a la tarjeta.

De forma adicional, la FPGA CMOD cuenta con 2 leds normales, 1 led RGB y dos botones push configurables para ser pull-up o pull-down dependiendo de la aplicación que se les dé.

Figura 149.

Diagrama de conexión de los leds y botones de la FPGA CMOD A7-35T

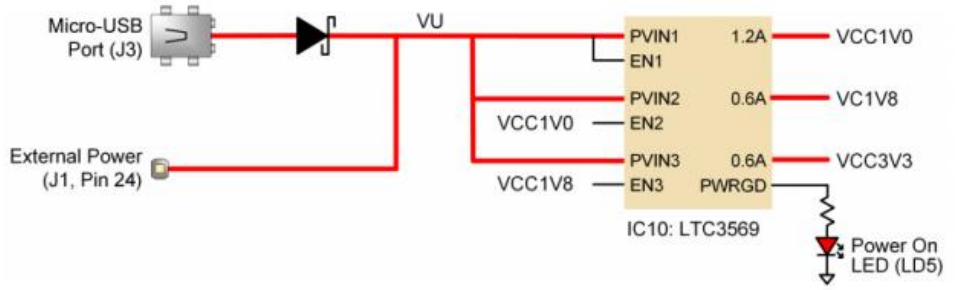


Nota. En la figura se puede observar un diagrama que representa las conexiones de los botones y los leds a la FPGA CMOD A7-35T. Obtenido de Farnell. *Cmod A7 Reference Manual (*https://www.farnell.com/datasheets/2876438.pdf*),* consultado el 08 de octubre de 2023. De dominio público.

Como último punto se puede mencionar el módulo Micro USB, tiene una capacidad de soportar rangos de voltaje externos de 3.3 hasta 5.5V similares al pin 24 de VCC. Para estos pines recomienda el fabricante de no utilizarlos a la vez, ya que pueden dañar severamente a la tarjeta y a la fuente de voltaje que se esté utilizando.

Figura 150.

Diagrama del módulo de alimentación de la FPGA CMOD A7-35T

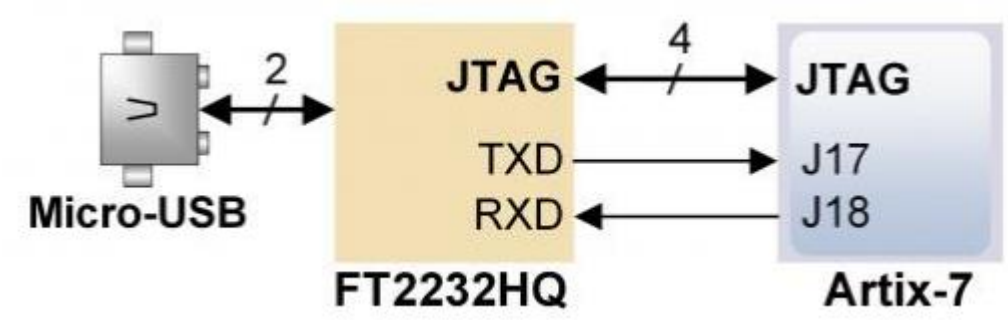


Nota. En la figura se puede observar un diagrama que presenta la forma en que se encuentra configurado el módulo de alimentación de la FPGA CMOD A7-35, se puede apreciar cómo se encuentran unidos los pines que salen del puerto Micro-USB y el pin 24 de alimentación externa. Obtenido de Farnell. *Cmod A7 Reference Manual (*https://www.farnell.com/datasheets/2876438.pdf*),* consultado el 08 de octubre de 2023. De dominio público.

Existe una diferencia importante en el módulo Micro USB que hace destacar a la CMOD entre su competencia, el módulo integra un controlador llamado FT2232HQ que permite utilizar el puerto USB con un protocolo de comunicación UART. El controlador hace el intercambio de información entre los pines J17 y J18 del chip XC7A35T y el puerto del Micro-USB, lo que permite intercambiar información usando UART sin necesidad de montar un módulo externo para intercambiar la información.

Figura 151.

Diagrama del módulo UART de la FPGA CMOD A7-35T



Nota. En la figura se puede observar un diagrama que presenta la forma en que se encuentra conectado el módulo UART con el módulo micro-USB y el chip XC7A35T. Obtenido de Farnell. *Cmod A7 Reference Manual (*https://www.farnell.com/datasheets/2876438.pdf*),* consultado el 08 de octubre de 2023. De dominio público.

5.2.2. Características tarjeta Elbert V2 Spartan 3A

Las características de la Elbert V2 se pueden observar de forma detallada en el primer capítulo de este trabajo. En este capítulo se explorará únicamente los detalles más relevantes para hacer la comparación con la CMOD A7-35T.

La Elbert V2 utiliza un puerto de alimentación DC capaz de soportar voltajes de hasta 9 V. Adicionalmente cuenta con un puerto Mini-USB que es capaz de soportar hasta los 6 V, sin embargo, este puerto sirve únicamente para transferir el programa .bit a la FPGA y como alimentación de la misma. No posee un módulo integrado UART que permita al usuario utilizarlo para un protocolo de comunicación.

Otra diferencia relevante es la cantidad de GPIOs que posee la Elbert V2, posee un total de 39 GPIOs que soportan un rango de voltaje de 0 hasta los 5.2 V. Adicional a estos puertos, se encuentra sobre la tarjeta 8 leds, 6 botones tipo push y 8 botones tipo DIP Switch, lo que le otorga a la tarjeta una gran autonomía a la hora de realizar aplicaciones de aprendizaje sin tener la necesidad de conectar módulos externos.

Como último punto, se deben mencionar los módulos especiales que contiene la tarjeta. Posee un puerto VGA capaz de entregar video de 480 x 640 a 8 bits, también tiene un conector tipo Jack 3.5 mm hembra, una ranura micro-SD con la que se puede interactuar directamente desde el chip XC3S50A y un kit de 3 displays de 7 segmentos capaces de desplegar valores numéricos encendiendo y apagando sus puertos.

Al analizar el resumen de las capacidades y módulos de la Elbert V2, se hace evidente que su diseño a pesar de ser antiguo para la época en la que se realiza este trabajo, es funcional y otorga la variedad necesaria para llevar a cabo el aprendizaje del lenguaje de las FPGA a un entorno aplicado.

5.2.3. Comparación final

Al analizar los módulos que componen a cada una de las tarjetas se observa que la Elbert V2 posee una mayor variedad de módulos de aplicación ya que incorporan módulos de video, audio, displays y demás. Sin embargo, la CMOD A5-35T posee una mayor variedad de módulos de comunicación, como lo es el módulo UART o los pines análogos dentro de la misma. Esto indica que ambas tarjetas están pensadas para un diferente ámbito de trabajo. La Elbert V2 es ideal para un entorno educativo, la variedad de módulos que ofrece puede permitir a un usuario novato aprender la implementación de

módulos en VHDL sin necesidad de depender de módulos externos, mientas que la CMOD A5-35T puede permitirle a un usuario desarrollar un proyecto que se adapte mejor al ámbito industrial, utilizando los módulos de comunicación internos para interactuar con más dispositivos, sin la opción de tener interacciones visuales dentro de la propia tarjeta. A continuación, se muestra una tabla resumen de las propiedades de cada una.

Tabla 9.

Comparación de las diferencias entre las tarjetas Elbert V2 y CMOD A7-35T

Característica	Elbert V2	CMOD A7-35T
Cantidad de GPIOs	39	56
Cantidad de ADCs	0	2
Tipo de conector	Mini USB	Micro USB
Cantidad de leds	8	3
Cantidad de botones Push	6	2
Cantidad de Dip-Switch	8	0
¿Posee módulo Display 7 Segmentos?	Si	No
¿Posee módulo UART?	No	Si
¿Posee módulo VGA?	Si	No
¿Posee módulo de audio?	Si	No
¿Posee módulo de tarjeta SD?	Si	No
Precio	Q.1000.00	Q.950.00

Nota. Los valores que se muestran en la tabla fueron extraídos de las hojas de datos de los fabricantes de las tarjetas Elbert V2 y CMOD A7-35T. Elaboración propia, realizado con Excel.

Ambos modelos son útiles y se encuentran en el mismo rango de precio, siendo la Elbert V2 la opción más costosa por una diferencia de Q.50.00, aunque se debe mencionar que la CMOD se debe exportar lo que puede resultar la variación de precios.

En conclusión, dependiendo de las aplicaciones que se le dé al aprendizaje, podría ser más conveniente para el usuario principiante adquirir la Elbert V2 por la cantidad de módulos básicos y visuales que ofrece, dejando la CMOD A7-35T para usuario que requieran un uso industrial básico.

5.3. Comparación del *software* utilizado

Para poder realizar la programación de las tarjetas Elbert V2 y CMOD A7-35T se requiere un tipo de *software* sintetizador que sea capaz de transformar el código en VHDL o Verilog en un archivo .bit que pueda ser cargado a la FPGA.

En el caso de la Elbert V2 se tiene el *software* ISE Design Suite. Este sistema es un poco complicado de utilizar en las computadoras modernas, se debe realizar la instalación en una máquina virtual o utilizar la máquina virtual que viene incluida en el instalador. Al tener que virtualizar el entorno se presentan algunas dificultades como lo pueden ser la compatibilidad de los drivers, la configuración que se modifica en el BIOS de los ordenadores y demás factores que afectan al momento de realizar las instalaciones, además que dicho *software* ya no cuenta con un soporte o actualizaciones activas.

Para la CMOD A7-35T, AMD creo el *software* Vivado Design Suite. Este es un *software* reemplazó el creado por Xilinx y permitió una mayor accesibilidad al estar en constante actualización. Permite la instalación de los controladores de las distintas FPGA de Xilinx de forma muy sencilla y amigable. Permite cargar los programas de forma directa a la tarjeta desde la interfaz principal, posee una mejor distribución de proyectos, permite la migración de archivos del programa ISE al nuevo *software*, además de poseer con una biblioteca de recursos mucho mayor que su predecesor.

Algo muy importante que se debe mencionar en este punto son las comunidades que poseen cada uno de los *softwares* mencionados. En el caso de Vivado, posee una comunidad altamente activa y mucho material de soporte. El *software* ISE al ser un poco antiguo no posee una comunidad tan activa como la de Vivado y mucho del material que se puede encontrar se encuentra desactualizado.

En conclusión, ambos programas son útiles para sintetizar el código para sus respectivas tarjetas, sin embargo, debido a las actualizaciones, practicidad en instalación y facilidad de uso, es recomendable migrar la creación de los proyectos del entorno ISE al nuevo sistema Vivado.

5.4. Análisis final

A lo largo de estos capítulos se han observado las principales ventajas y diferencias entre las tarjetas Elbert V2 y CMOD A7-35T. Se sabe que la CMOD posee una mayor capacidad de procesamiento, por el lado de la Elbert V2 se conoce que posee una mayor cantidad de módulos incorporados y una mayor facilidad de adquisición. En el caso de los programas que utilizan, el *software* Vivado utilizado para la programación de la CMOD es mucho mejor que el ISE Design Suite utilizado por la Elbert V2, también se conoce que el soporte es mucho mayor para la CMOD.

Al analizar los puntos mencionados, se puede concluir que la elección de una tarjeta sobre otra dependerá únicamente del uso que se le dé. La Elbert V2 es útil para conocer los fundamentos básicos del mundo de las FPGA, a pesar de utilizar *software* antiguo, es una excelente opción para un entorno educativo. Debido a lo anterior, se puede seguir recomendando el uso de la

Elbert V2 para los laboratorios de electrónica, teniendo en cuenta la CMOD A7-35T para futuros proyectos que requieran un procesamiento mayor.

CONCLUSIONES

1. Se identificó que la FPGA Elbert V2 está equipada con un microchip que puede soportar hasta 4.6 V de voltaje de entrada. Además, cuenta con 4 bloques de memoria RAM de 16 KB y una capacidad lógica de 1584 celdas.

2. La FPGA Elbert V2 es capaz de integrar módulos externos como UART, ADC, controladores de servomotores y controladores de motores paso a paso. Esta capacidad es útil para los proyectos de los laboratorios de Electrónica 3 y 6.

3. Aunque la FPGA CMOD A7-35T es una opción tecnológica viable, la FPGA Elbert V2 sigue siendo la mejor opción para el aprendizaje y la enseñanza en los laboratorios de Electrónica 3 y 6.

4. Se comprobó que la FPGA Elbert V2 posee la capacidad necesaria para ejecutar los algoritmos y programas requeridos en los laboratorios de Electrónica 3 y 6. Se ha proporcionado una guía para la ejecución de estos algoritmos en el repositorio compartido de este trabajo.

RECOMENDACIONES

1. Ejecutar únicamente con *software* oficial de la compañía desarrolladora el desarrollo de los programas, proyectos o códigos que se ejecuten en una FPGA. El utilizar *software* oficial, garantiza que la configuración de parámetros se encuentre adaptada a la FPGA utilizada.

2. Desglosar el funcionamiento de los módulos que se utilizaran con la FPGA Elbert V2 a modo que se pueda reinterpretar las funciones en una máquina de estados para que esta sea adaptada al lenguaje VHDL de manera más sencilla.

3. Implementar un repositorio general con ejemplos de uso de la FPGA Elbert V2 para los Laboratorios de Electrónica 3 y 6 para que los estudiantes y profesores tengan un punto de intercambio de conocimiento para mejorar los proyectos e incentivar la colaboración a nivel universitario.

4. Verificar los avances en la tecnología FPGA para identificar nuevas oportunidades de mejora en la enseñanza y la investigación. Si bien la FPGA Elbert V2 es adecuada para el aprendizaje, el realizar revisiones periódicas de las capacidades de hardware emergente podría revelar herramientas que ofrezcan ventajas significativas en términos de rendimiento y funcionalidad.

REFERENCIAS

AMD. (2020). *AMD Completes Acquisition of Xilinx*. https://www.amd.com/en/press-releases/2022-02-14-amd-completes-acquisition-xilinx

Global Market Insights (2022) *Field Programmable Gate Array (FPGA) Market Size*. https://www.gminsights.com/industry-analysis/field-programmable-gate-array-fpga-market-size

Intel (2015). *Intel Completes Acquisition of Altera*. https://newsroom.intel.com/news-releases/intel-completes-acquisition-of-altera/#gs.gapva2

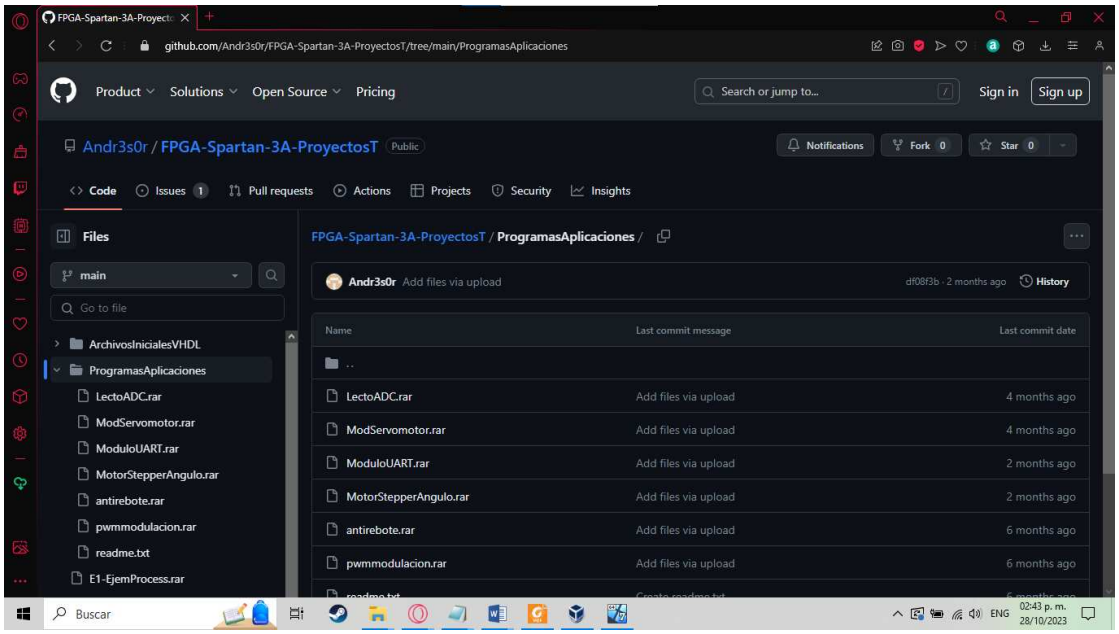
Sánchez Élez, M (2015). *Introducción A La Programación en VHDL*. [Tesis de pregrado, Facultad de Informática Universidad Complutense de Madrid]. Archivo digital.. https://docta.ucm.es/rest/api/core/bitstreams/4ded6d60-6b62-4f59-a7cd-2511b9a73861/content

Xilinx (2018*). Spartan-3A FPGA Family:Data Sheet.* https://docs.xilinx.com/v/u/en-US/ds529

APÉNDICES

Apéndice 1.

Acceso a la carpeta en github

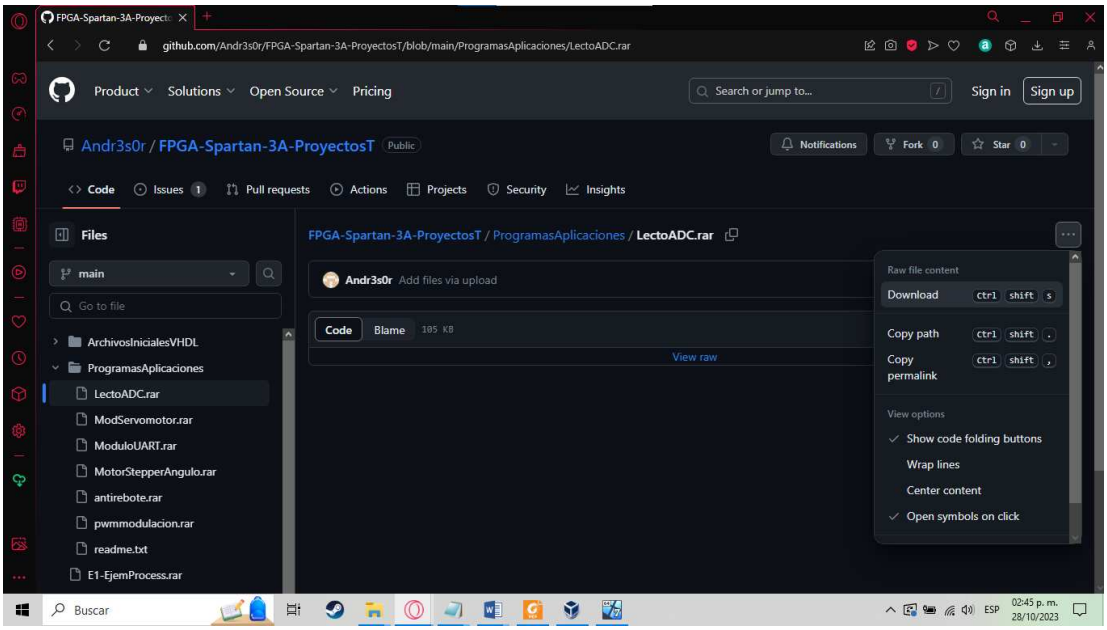


Nota. Se muestra la carpeta de GitHub en la que se encuentran todos los programas realizados. Elaboración propia, realizado con Opera GX.

Se debe descargar el archivo .rar del proyecto que se quiera utilizar, para el ejemplo se utilizara el proyecto Lector ADC.

Apéndice 2.

Descarga del archivo rar completo

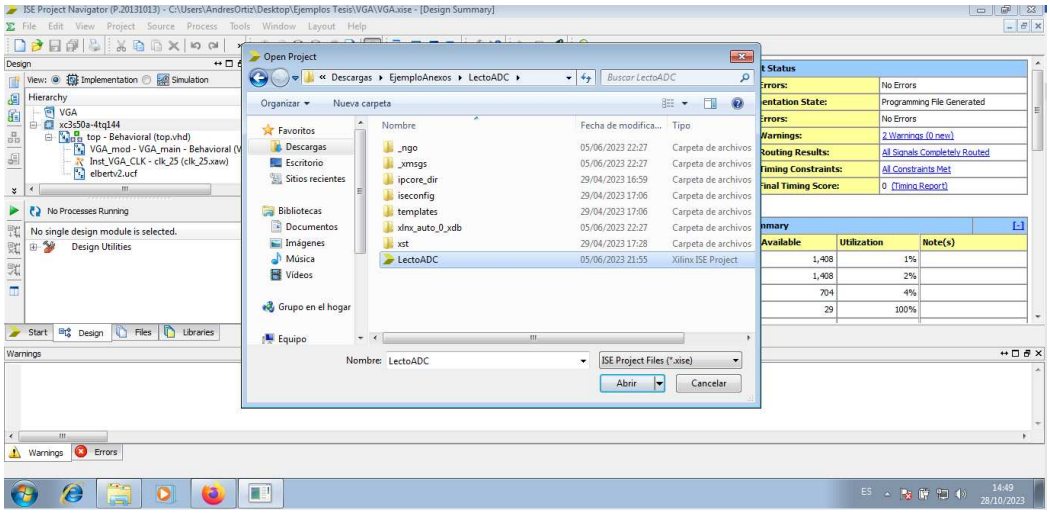


Nota. Se muestra la forma en que se debe hacer la descarga del archivo rar, se debe descargar completo porque de lo contrario al momento de hacer la carga del archivo puede generar error. Elaboración propia, realizado con Opera GX.

Una vez se ha descargado el archivo, se debe descomprimir. Dentro del programa ISE Design Suite, una vez se ha descomprimido el archivo, se debe seleccionar el archivo haciendo uso de la función Open Project ubicado en la pestaña de File.

Apéndice 3.

Carga del proyecto a ISE Design Suite

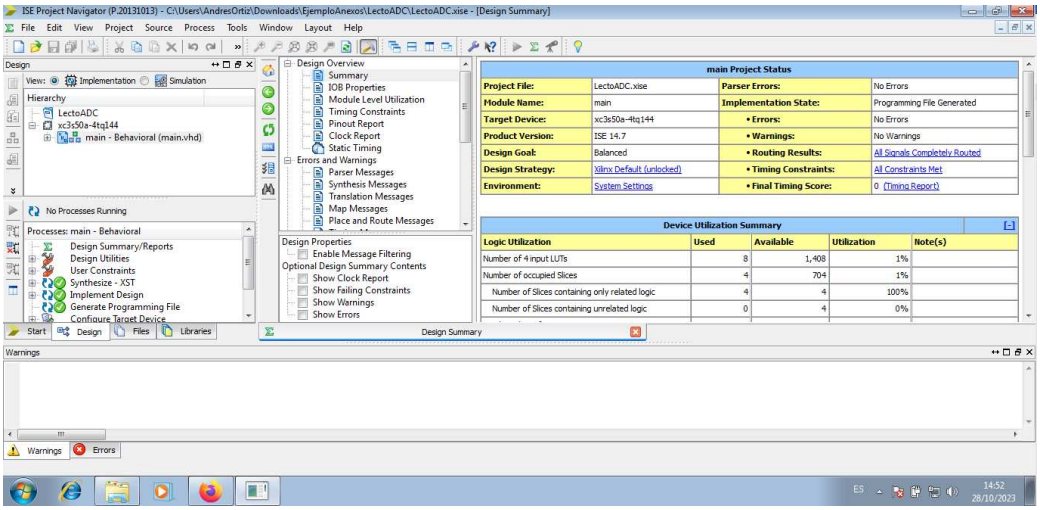


Nota. Se muestra la forma en que se debe cargar el proyecto al *software* ISE Design Suite. Elaboración propia, realizado con ISE Design Suite 14.

Se debe seleccionar el archivo con el símbolo > y que tenga el tipo de archivo Xilinx ISE Project. Una vez se ha cargado el archivo, el *software* ISE desplegara el proyecto, teniendo el ultimo estado de ejecución previo al guardado. Para el caso del ejemplo se puede observar que el programa fue sintetizado y la generación del archivo. bit fue exitosa.

Apéndice 4.

Despliegue de los datos una vez se ha cargado el proyecto a ISE Design Suite



Nota. Se muestra la forma en que se debe cargar el proyecto al *software* ISE Design Suite. Elaboración propia, realizado con ISE Design Suite 14.