Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

FPGA LIBRE ALHAMBRA II COMO HERRAMIENTA DE APRENDIZAJE PARA LA ELECTRÓNICA DIGITAL

Jorge Mario López Monterroso
Asesorado por la Inga. Ingrid Salomé Rodríguez de Loukota

Guatemala, febrero de 2020

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

FPGA LIBRE ALHAMBRA II COMO HERRAMIENTA DE APRENDIZAJE PARA LA ELECTRÓNICA DIGITAL

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

JORGE MARIO LÓPEZ MONTERROSO
ASESORADO POR LA INGA. INGRID SALOMÉ RODRÍGUEZ DE LOUKOTA

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN ELECTRÓNICA

GUATEMALA, FEBRERO DE 2020

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANA	Inga. Aurelia Anabela Cordova Estrada
VOCAL I	Ing. José Francisco Gómez Rivera
VOCAL II	Ing. Mario Renato Escobedo Martínez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Christian Moisés de la Cruz Leal
VOCAL V	Br. Kevin Armando Cruz Lorente
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADOR	Ing. Byron Odilio Arrivillaga Méndez
EXAMINADOR	Ing. José Aníbal Silva de los Ángeles
EXAMINADOR	Ing. Guillermo Antonio Puente Romero
SECRETARIA	Inga. Lesbia Magalí Herrera López

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:

FPGA LIBRE ALHAMBRA II COMO HERRAMIENTA DE APRENDIZAJE PARA LA ELECTRÓNICA DIGITAL

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 7 de noviembre de 2018.

Jorge Mario López Monterroso

Guatemala 01 de octubre de 2019

Ingeniero
Julio Cesar 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 **"FPGA Libre Alhambra II como herramienta de aprendizaje para la electrónica digital",** del señor Jorge Mario López Monterroso, por considerar que cumple con los requisitos establecidos.

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

Sin otro particular, me es grato saludarle.

Atentamente,

Inga. Ingrid Rodríguez de Loukota
Colegiada 5,356
Asesora

Ingrid Rodríguez de Loukota
Ingeniera en Electrónica
colegiada 5356

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

Guatemala, 8 de octubre de 2019

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 **FPGA LIBRE ALHAMBRA II COMO HERRAMIENTA DE APRENDIZAJE PARA LA ELECTRÓNICA DIGITAL** desarrollado por el estudiante **Jorge Mario López Monterroso**, 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

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÍA

REF. EIME 76. 2019.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto bueno del Coordinador de Área, al trabajo de Graduación del estudiante: JORGE MARIO LÓPEZ MONTERROSO titulado: FPGA LIBRE ALHAMBRA II COMO HERRAMIENTA DE APRENDIZAJE PARA LA ELECTRÓNICA DIGITAL, procede a la autorización del mismo.

Ing. Armando Alonso Rivera Carrillo

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

GUATEMALA, 8 DE NOVIEMBRE 2019.

Universidad de San Carlos
De Guatemala

Facultad de Ingeniería
Decanato

Ref. DTG.060-2020

La Decana 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: **FPGA LIBRE ALHAMBRA II COMO HERRAMIENTA DE APRENDIZAJE PARA LA ELECTRÓNICA DIGITAL,** presentado por el estudiante universitario: **Jorge Mario López Monterroso,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, se autoriza la impresión del mismo.

IMPRÍMASE.

Inga. Aurelia Anabela Cordova Estrada
Decana

Guatemala, febrero de 2020

AACE/asga

ACTO QUE DEDICO A:

Dios	Por haberme permitido alcanzar esta meta y brindarme entendimiento, inteligencia y sabiduría.
Mis padres	Dania Monterroso y Marcos López. Por sus palabras de aliento y apoyo para seguir adelante y culminar con éxito esta etapa de mi vida. Por enseñarme con su ejemplo lo que es la responsabilidad.
Mi hermana	Jaqueline Monterroso. Por enseñarme con su ejemplo que con dedicación y esfuerzo todo lo que se propone se puede cumplir y brindarme palabras de aliento cuando las he necesitado.
Mi tía	Gicela Monterroso, quien me brindó su apoyo y me motivó a seguir adelante.
Mis amigos	Por acompañarme y apoyarme a lo largo de mi carrera.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por brindarme la oportunidad de formarme como profesional.
Inga. Ingrid Rodríguez de Loukota	Por asesorar este trabajo de graduación, sus consejos, paciencia y opiniones sirvieron para que me sienta satisfecho con la realización de este trabajo.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
A	Amperios
GB	GigaByte
Gb/s	Gigabit por segundo
Kb	Kilobit
Mb/s	Megabit por segundo
MB	MegaByte
Mhz	Megahertz
µA	Microamperios
mA	Miliamperios
mV	Milivoltios
V	Voltaje

GLOSARIO

ASCII Código Estándar Estadounidense para el intercambio de información. Es un sistema de codificación de caracteres alfanuméricos que asigna un número de 0 al 127 a cada letra, número o carácter especial.

Assembler Lenguaje de programación de bajo nivel con un conjunto de instrucciones básicas para microprocesadores, microcontroladores y otros circuitos integrados programables.

BCD Decimal codificado en binario. Representa números decimales en el sistema binario asignando a cada digito decimal una secuencia de 4 bits.

Bit Es la menor unidad de información que equivale a la selección entre dos alternativas que tienen el mismo grado de probabilidad. Es un dígito del sistema de numeración binario.

Bitstream Flujo de bits.

CFM Memoria flash configurable.

CLB Bloque de lógica configurable.

Codificar	Método que permite convertir un carácter de un lenguaje de origen en un símbolo de otro sistema de representación.
FTDI	Future Technology Devices International. Empresa escocesa que desarrolla, fabrica y da soporte a dispositivos y sus drivers para la conversión transmisiones serie o TTL a señales USB.
HDL	Lenguaje de descripción de hardware. Lenguaje utilizado para definir la estructura, diseño y operación de circuitos electrónicos digitales.
IOB	Bloque de entrada/salida.
Microcontrolador	Circuito integrado programable, capaz de ejecutar las ordenes grabadas en memoria. Está formado por una unidad central de procesamiento, unidades de memoria, puertos de entrada/salida y periféricos.
Oscilador	Circuito electrónico que produce una señal electrónica repetitiva.
PLB	Bloque de lógica programable.
Toolchain	Conjunto de herramientas de desarrollo de software distintas que están unidas por etapas específicas.
USB	Bus serial universal.

Verilog	Lenguaje de descripción de hardware utilizado para modelar sistemas electrónicos.
Wire	Un cable o red enrutada físicamente en una FPGA.
Xilinx	Compañía de tecnología estadounidense la cual es reconocida por inventar la FPGA y por ser un distribuidor de dispositivos lógicos programables.

RESUMEN

En este trabajo de graduación se realiza un análisis sobre la viabilidad del uso de la FPGA libre para el aprendizaje de la electrónica digital. Para este fin, se expone la ventaja económica de utilizar una FPGA en el aprendizaje de la electrónica digital haciendo un estudio de gastos en componentes en las prácticas de aprendizaje del Laboratorio de Electrónica.

Al evidenciar la viabilidad del uso de la FPGA, se realiza una comparativa entre la FPGA Libre y una de las FPGA comerciales utilizadas en el Laboratorio de Electrónica de la USAC analizando la estructura y capacidad de procesamiento.

Se documenta software *open source* el cual se puede utilizar en conjunto con la FPGA Libre para el diseño y desarrollo de hardware con la finalidad de dar a conocer herramientas de software sin costo que benefician al estudiante.

Finalmente, se exponen las bases teóricas de la electrónica digital en conjunto con su aplicación utilizando el software libre IceStudio y la FPGA Libre Alhambra II, explicando la forma de utilizar estas herramientas en el aprendizaje de la electrónica digital.

OBJETIVOS

General

Facilitar el aprendizaje de la FPGA y de los conocimientos de electrónica digital por medio de la FPGA libre Alhambra II y el software libre.

Específicos

1. Proporcionar conocimientos en electrónica digital fáciles de entender utilizando la FPGA libre Alhambra II.

2. Utilizar la plataforma IceStudio como medio de aprendizaje del funcionamiento de la FPGA.

3. Introducir la FPGA libre y el software libre como una opción para el desarrollo de hardware.

INTRODUCCIÓN

Los avances tecnológicos que se dan día a día han desencadenado un deseo en el ser humano por aprender sobre la electrónica digital y el funcionamiento de todo el hardware que lo rodea. Uno de estos avances es la FPGA, un dispositivo programable el cual internamente posee una matriz de celdas de lógica cuya funcionalidad e interconexión puede ser configurada mediante un lenguaje de descripción de hardware. En la actualidad, se ha incrementado su uso y enseñanza debido a las ventajas que presenta frente a otros dispositivos.

El problema se encuentra en la complejidad del aprendizaje de la FPGA en el lenguaje de descripción de hardware, debido a que presenta similitudes con los lenguajes de programación secuenciales que se utilizan para desarrollar aplicaciones o los utilizados para configurar microcontroladores y otros dispositivos utilizados en electrónica. Cuando se configura una FPGA utilizando el lenguaje de descripción de hardware, no se puede ver paso a paso qué elementos de hardware se utilizan y tampoco la forma como se interconectan para llevar a cabo las tareas que se necesitan, con lo cual, el aprendizaje en lógica combinacional, secuencial y demás conceptos básicos que se utilizan en el desarrollo de hardware son difíciles de captar.

La realización de este trabajo de graduación desea introducir la FPGA libre como dispositivo de desarrollo de hardware fácil de configurar y de aprender. Ayuda al aprendizaje de los conocimientos de electrónica digital, reduce gastos en la compra de diversos dispositivos de laboratorio (protobord, integrados de compuertas lógicas, contadores, etc.) porque la FPGA es reconfigurable y da la

oportunidad de desarrollar muchos diseños digitales sin tener que hacer gastos extra aparte de comprar la misma FPGA.

Se introduce la plataforma de IceStudio, software libre que proporciona una forma gráfica de configurar e interconectar las celdas de lógica dentro de la FPGA, con lo cual, se ayuda a las personas a pensar de forma lógica en la forma de interconectar todo el hardware para lograr el funcionamiento que se desea en la FPGA.

1. ELECTRÓNICA DIGITAL

En la actualidad, la mayoría de las personas cuentan, como mínimo, con un dispositivo electrónico el cual es utilizado todos los días y, por tanto, se puede afirmar que el ser humano vive rodeado por todo tipo de tecnología que facilita las tareas y vida cotidiana.

La naturaleza del ser humano lo induce a entender cómo funciona y trabaja lo que le rodea. En muchas personas surge el deseo de aprender sobre electrónica digital, ya sea de forma superficial o más a profundidad para diseñar prototipos los cuales puedan cambiar aspectos de la sociedad.

La electrónica digital es una parte de la electrónica en la cual, los valores de voltaje son representados por medio de códigos binarios para poder almacenarlos y utilizarlos, por lo cual, el método de enseñanza-aprendizaje utilizado en los estudiantes de electrónica consiste en impartir la teoría y afianzar los conocimientos con prácticas de laboratorio en las cuales el estudiante se familiariza con el empleo de diversos integrados los cuales tienen funciones específicas.

Cada integrado según su función, tiene un costo diferente y en cada práctica de laboratorio se utilizan diversos dispositivos con el fin de que el estudiante aprenda a utilizar cada uno de ellos. Al finalizar el curso, los integrados y demás componentes utilizados durante cada práctica difícilmente son reutilizados más adelante, sin embargo, debido a que en cursos superiores ya no se necesita el uso de estos materiales, ya sea porque los mismos no pueden llevar acabo la

función que se requiere o la velocidad de trabajo es menor a la requerida, con lo cual, el estudiante termina haciendo una inversión sin retorno a largo plazo.

1.1. Código binario

El código binario utiliza números binarios para representar cifras o valores de voltaje los cuales son utilizados por los componentes electrónicos para ser guardados o realizar funciones aritméticas por medio de funciones lógicas con cada uno de los bits que los conforman.

Un bit puede poseer solo uno de dos valores posibles (“0” o “1”) los cuales en electrónica indican que se tiene voltaje o no. El código binario puede poseer N bits, ya que no hay una cantidad máxima de bits que lo puedan conformar.

El empleo del código binario se debe a la necesidad de representar de diferente manera los datos que se quieren manipular, debido a que los sistemas digitales manejan números binarios para codificar la información. Por ejemplo, el disco duro de una computadora posee números binarios, que son la codificación de toda la información almacenada por el usuario y utiliza el código binario para representar números, imágenes y demás datos de una manera que la computadora puede reconocer y manipular.

1.2. Código BCD

El código BCD o también llamado “decimal codificado en binario” es una forma de representar los números decimales en un código que consta de unos y ceros para que los sistemas digitales puedan manipular la información. La necesidad de usar el código BCD se debe a que los seres humanos en la actualidad utilizan los números decimales para hacer todos los cálculos, por el

contrario, los sistemas digitales funcionan detectando estados en alto y bajo (unos y ceros), por lo cual, el código BCD asigna un código binario de cuatro bits a cada valor decimal y utiliza estos códigos para representar las cifras y datos que se necesitan procesar en los sistemas digitales.

Un código binario distinto corresponde a cada número de cero a nueve, por lo tanto, si se quiere representar un número que conste de más de un símbolo, el código BCD separa en 4 bits el código binario correspondiente y asigna los códigos a cada dígito que forma el número en decimal.

Para comprenderlo de manera más sencilla, en la Tabla I se muestran los códigos binarios que corresponden a cada dígito en decimal.

Por ejemplo, si se quiere representar el número 153 utilizando el código BCD, lo que se debe hacer es tomar cada dígito que forma al número decimal y representarlo con el código binario que le corresponde según la Tabla I, por lo que el número resultante seria: 0001 0101 0011.

Tabla I. **Código BCD**

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Fuente: elaboración propia.

1.3. Álgebra booleana

Es un sistema algebraico utilizado en la electrónica digital para simplificar funciones booleanas, las cuales son expresiones algebraicas que constan de tres componentes: variables binarias, operadores lógicos (+ y ·) y valores constantes (0 y 1).

La Tabla II muestra los once teoremas fundamentales del álgebra booleana los cuales al aplicarse correctamente facilita la creación de circuitos digitales ayudando a reducir en número de componentes a utilizar.

Tabla II. **Teoremas del álgebra booleana**

(A') '= A Si A=0, entonces A' = 1, la salida es (A') '=0 Si A=1, entonces A' = 0, la salida es (A') '=1
A · 0 = 0 Si A=0, la salida es 0 Si A=1, la salida es 0
A + 0 = A Si A=0, la salida es 0 Si A=1, la salida es 1
A · 1 = A Si A=0, la salida es 0 Si A=1, la salida es 1
A + 1 = 1 Si A=0, la salida es 1 Si A=1, la salida es 1
A + A = A Si A=0, la salida es 0 Si A=1, la salida es 1
A · A = A Si A=0, la salida es 0 Si A=1, la salida es 1
A + A' = 1 Si A=0, entonces A' = 1, la salida es 0+1 = 1 Si A=1, entonces A' = 0, la salida es 1+0 = 1
A · A' = 0 Si A=0, entonces A' = 1, la salida es 0 · 1 = 0 Si A=1, entonces A' = 0, la salida es 1 · 0 = 0
A · B+ A · C = A (B + C)
A + A' · B = A + B

Fuente: elaboración propia.

1.4. BCD con compuertas lógicas

El diseño de un BCD con compuertas lógicas es una de las prácticas de laboratorio que regularmente se utilizan para que el estudiante aplique los conocimientos adquiridos durante la clase teórica.

En la práctica 1 se utiliza la tabla III para colocar los códigos binarios correspondientes a cada número decimal y en las salidas se colocan estados en alto y bajo para encender los leds de cada uno de los segmentos que componen el display para mostrar el número decimal que corresponde.

- Práctica de aplicación de conocimientos básicos de electrónica digital.

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERIA
ESCUELA DE MECÁNICA ELÉCTRICA
LABORATORIO DE ELECTRÓNICA
ELECTRONICA 3

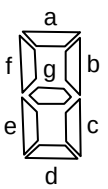
PRACTICA 1: BCD CON COMPUERTAS LÓGICAS

Diseñar un BCD y que los números puedan visualizarse en un display de 7 segmentos. Utilizar la Tabla III para deducir las funciones booleanas para cada segmento de display. Reducir las funciones booleanas utilizando los teoremas del álgebra booleana e implementar cada función booleana en protoboard.

Adjuntar a la hoja de calificación, una hoja con las funciones booleanas obtenidas, las funciones booleanas reducidas y especificar que teorema de álgebra booleana utilizó para obtener cada una de las funciones reducidas.

Tabla III. **Tabla de verdad y display**

Decimal	Código binario			Segmentos del display							
	Z	X	Y	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	
1	0	0	1	0	1	1	0	0	0	0	
2	0	1	0	1	1	0	1	1	0	1	
3	0	1	1	1	1	1	1	0	0	1	
4	1	0	0	0	1	1	0	0	1	1	
5	1	0	1	1	0	1	1	0	1	1	
6	1	1	0	0	0	1	1	1	1	1	
7	1	1	1	1	1	1	0	0	0	0	



Fuente: elaboración propia.

Se tienen 7 variables que corresponden a cada uno de los leds de cada segmento que forma el display, por lo tanto, existen 7 funciones booleanas las cuales resultan de analizar cada una de las columnas de las variables del display en la tabla III.

La forma de obtener las funciones booleanas de cada variable es observar en que fila se encuentra un “1” y luego observar el código binario que corresponde a esa fila y sacar la porción de la función booleana que corresponde, luego se sigue buscando el siguiente “1” en la columna y si se encuentra se agrega un signo + y se suma la siguiente porción de la función booleana. Los códigos binarios se denotan con variables en mayúscula, por lo cual, la función booleana resultante consta de variables en mayúscula.

Si se sigue el proceso correctamente, el estudiante obtendrá las 7 funciones booleanas siguientes:

- a = A'B'C' + A'BC' + ABC' + AB'C + ABC.
- b = A'B'C' + AB'C' + A'BC' + ABC' + A'B'C + ABC.
- c = A'B'C' +AB'C' + ABC' + A'B'C + AB'C + A'BC + ABC.
- d = A'B'C' + A'BC' + A'B'C + AB'C + A'BC.
- e = A'B'C' + A'BC' + A'BC.
- f = A'B'C' + A'B'C + AB'C + A'BC.
- g = A'BC' + ABC' + A'B'C + AB'C + A'BC.

Lo siguiente es la aplicación de los conocimientos del álgebra booleana. Se le pide al estudiante reducir las funciones obtenidas utilizando los teoremas del álgebra booleana. Las funciones resultantes se muestran a continuación:

- a = AC + C'(A'+B)

- b = C' + A'B' + AB
- c = B' + A + C
- d = C'(A'+B) + A'B + AB'C
- e = A' (B+C')
- f = B' (A'+C) + A'C
- g = B'C + B(A'+C')

El último paso, es la práctica de laboratorio en la cual se le pide al estudiante implementar con compuertas lógicas las funciones obtenidas de la tabla III y las obtenidas aplicando los teoremas de álgebra booleana. El objetivo es que el estudiante pueda observar que a pesar de que las funciones booleanas cambian el resultado se mantiene, ya que se deben mostrar los mismos resultados en ambos displays al momento de la implementación.

1.5. Costo y presupuesto

El valor de los componentes necesarios para implementar la práctica de laboratorio es uno de los puntos de interés de este trabajo de investigación. La práctica propuesta anteriormente es sólo un ejemplo de una de varias prácticas de laboratorio que se realizan en los cursos de electrónica para principiantes, por lo cual, calculando el monto aproximado a gastar, se puede dar una idea del gasto inicial en un curso de electrónica y de los gastos que vienen más adelante con la compra de otros componentes para las siguientes prácticas.

En la tabla IV se muestra los componentes necesarios para la práctica 1 junto con el valor (aproximado debido a que el precio varía según donde se realice la compra) de mercado.

Tabla IV. **Componentes de la práctica 1**

Cantidad	Componente	Descripción	Precio Unidad	Total
22	SN74LS08	Circuito integrado que posee internamente 4 compuertas AND de 2 entradas cada una.	Q9.00	Q198.00
1	SN74LS04	Circuito integrado que posee internamente 6 compuertas NOT.	Q12.00	Q12.00
11	SN74LS32	Circuito integrado que posee internamente 4 compuertas OR de 2 entradas cada una.	Q9.00	Q99.00
			Total	Q309.00

Fuente: elaboración propia.

La tabla IV sugiere un costo elevado, sólo si el estudiante no reutiliza las salidas obtenidas de las operaciones realizadas anteriormente con las compuertas. En el caso contrario, el estudiante puede observar que hay términos comunes entre las funciones booleanas como se muestra en la tabla V.

Tabla V. **Términos comunes**

Término	Número de veces que se repite
A'B'C'	6
A'BC'	3
ABC'	4
AB'C	5
ABC	3
AB'C'	3
A'B'C	5
A'BC	5

Fuente: elaboración propia

Tomando en cuenta lo mostrado en la tabla V, se puede hacer una reducción del gasto de componentes reutilizando las operaciones ya realizadas lo que reduce el número de compuertas AND utilizadas en la práctica de laboratorio y con esto el costo de esta. La tabla VI hace evidente que el valor de la práctica se reduce de forma significativa.

Tabla VI. **Reducción de componentes y precio**

Cantidad	Componente	Descripción	Precio Unidad	Total
8	SN74LS08	Circuito integrado que posee internamente 4 compuertas AND de 2 entradas cada una.	Q9.00	Q72.00
1	SN74LS04	Circuito integrado que posee internamente 6 compuertas NOT.	Q12.00	Q12.00
11	SN74LS32	Circuito integrado que posee internamente 4 compuertas OR de 2 entradas cada una.	Q9.00	Q99.00
			Total	Q183.00

Fuente: elaboración propia.

La reducción del precio es de Q126.00 por lo que se consideran Q183.00 un precio más bajo en consideración al anterior, pero sigue siendo un precio alto para una sola práctica, la cual sólo consta de compuertas lógicas básicas las cuales se utilizarán 3 o 4 veces más en las siguientes prácticas para después ser dejadas a un lado por los nuevos componentes que se utilizarán. Siguiendo esta línea de análisis los nuevos componentes adquiridos también se utilizarán unas cuantas veces más antes de ser dejados de lado repitiendo el ciclo durante el aprendizaje del estudiante.

De los componentes utilizados en las prácticas un pequeño número de ellos podrán ser revendidos por el estudiante para recuperar un poco de la inversión y poder comprar los nuevos componentes que requiere para su estudio. Por otro lado, los otros componentes que no pueda revender quedarán guardados a un lado debido a que se prefiere comprar componentes nuevos ya que se tiene una mayor seguridad de su correcto funcionamiento.

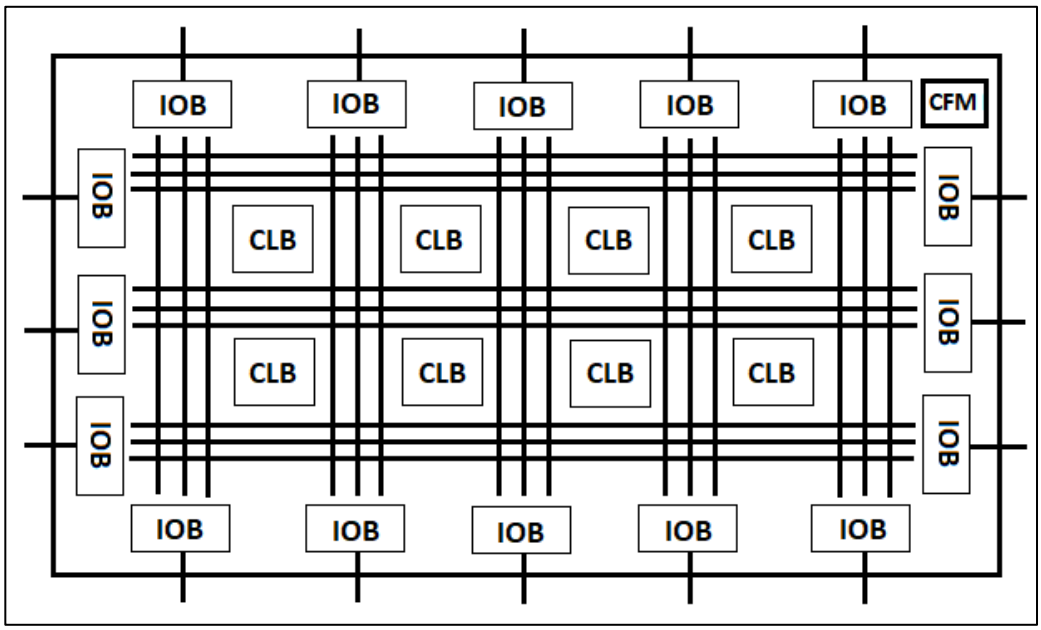
2. FPGA

“Field Programmable Gate Array” o simplemente “FPGA” es el nombre por el cual se le conoce al dispositivo electrónico de interés. Para comprender que es una FPGA se analizará la información que proporciona su nombre: “Field Programable”, indica que es un dispositivo que permite ser configurado y modificar su configuración en cualquier momento, ya sea antes o después de ser instalado para ejercer su función especificada. Por otro lado “Gate Array” denota los arreglos de compuertas establecidos en una matriz para poder ser utilizados por el usuario para configurar el hardware necesario para el trabajo que se necesita realizar.

Colocando todo en un conjunto, la FPGA es un dispositivo configurable el cual ofrece un arreglo de compuertas lógicas las cuales se pueden interconectar de la manera deseada para realizar uno o más trabajos, teniendo la posibilidad de modificar la configuración cuando sea necesario sin remover el dispositivo para hacer efectiva la nueva actualización de hardware.

Para tener un mejor entendimiento, la figura 1 muestra la estructura simplificada de una FPGA, en la cual se puede identificar 3 partes principales que son: los bloques configurables de lógica (CLB), bloques de entrada y salida (IOB) y los wires. Sumado a estos 3 existe un componente adicional o externo que es el bloque de la memoria flash configurable (CFM).

Figura 1. **Interior de la FPGA**



Fuente: elaboración propia, empleando Paint.

- Bloque de entrada/salida (IOB): las siglas provienen de su nombre en inglés “In/Out Block”, son las terminales de la FPGA las cuales se encargan de enviar o recibir información de o hacia el exterior. El número de entradas/salidas que se pueden tener un una FPGA depende del modelo y con esto también de su costo. Son terminales configurables al igual que todo el hardware de la FPGA por lo cual se puede alternar un mismo terminal entre una entrada o una salida dependiendo de las necesidades.

- Bloques configurables de Lógica (CLB): las siglas provienen de su nombre en inglés “configurable logic blocks” conforman la parte de hardware que mediante la plataforma de software se puede configurar para realizar una

tarea en específico. En los CLB se pueden encontrar dos grupos de componentes:

- Compuerta (Gate): unidades básicas en electrónica digital para el desarrollo de circuitos digitales ya que a pesar de que su función no es complicada, al interconectar varios tipos se pueden desarrollar circuitos digitales complejos los cuales son necesarios para diversas aplicaciones. Las operaciones booleanas básicas que desempeñan estas compuertas son: AND, OR y NOT.

- Registro (Register): unidades de memoria las cuales pueden guardar información y son un elemento importante en el desarrollo de circuitos digitales. Estas unidades de memoria al igual que pueden almacenar información también pueden olvidarla y esto sucede cuando se guarda otra información en un registro que ya está lleno, por lo cual, debe olvidar lo que ya sabe para poder memorizar la nueva información entrante.

- Cable (Wire): Son los “caminos” por los cuales transita la información en un sistema digital. Se utilizan para conectar los componentes de la FPGA (compuertas y registros) ya que es necesario la interacción entre ellos para que la FPGA pueda realizar las tareas para las cuales ha sido configurada por el usuario.

- Memoria Flash Configurable (CFM): las siglas provienen de su nombre en inglés “Configurable Flash Memory”. Se le denomina un bloque externo ya que ayuda a la FPGA a recordar que es lo que debe hacer luego que la misma se apaga. Dicho de otra forma, una FPGA por sí sola no es capaz de recordar la configuración y las tareas que se le han sido asignadas

debido a que los CLB no poseen memoria para este propósito. Al encender una FPGA, el mismo inicia sin saber (o recordar) qué hacer y es en este momento donde la memoria flash entra en acción. La memoria Flash almacena la información de la configuración de los CLB, así que, al momento de encender una FPGA la CFM empieza a repartir la configuración a cada bloque sobre lo que debe de hacer, con que otros CLB o IOB está conectado y por qué cable (wires) éstos se comunican.

Debido a que hay diversos modelos de FPGA, también hay diversos fabricantes, cada una de las FPGA difieren en cuanto a su estructura interna dependiendo del fabricante, la figura 1 es sólo una referencia a la estructura interna genérica y componentes básicos los cuales deben estar presentes en cualquier FPGA.

Actualmente hay dos formas de adquirir una FPGA, la primera es comprar el chip y por aparte comprar los componentes externos para su funcionamiento, como lo son la memoria Flash, los osciladores, el cable de datos del PC hacia la FPGA, etc.

La segunda forma es comprar el kit de tarjeta de desarrollo que trae la FPGA y además traen conectados todos los componentes externos necesarios para su correcto funcionamiento, como lo son las tarjetas de Xilinx o en el caso de este tema de investigación la FPGA libre Alhambra II. Esta es la forma más práctica si se está iniciando en el mundo de la FPGA.

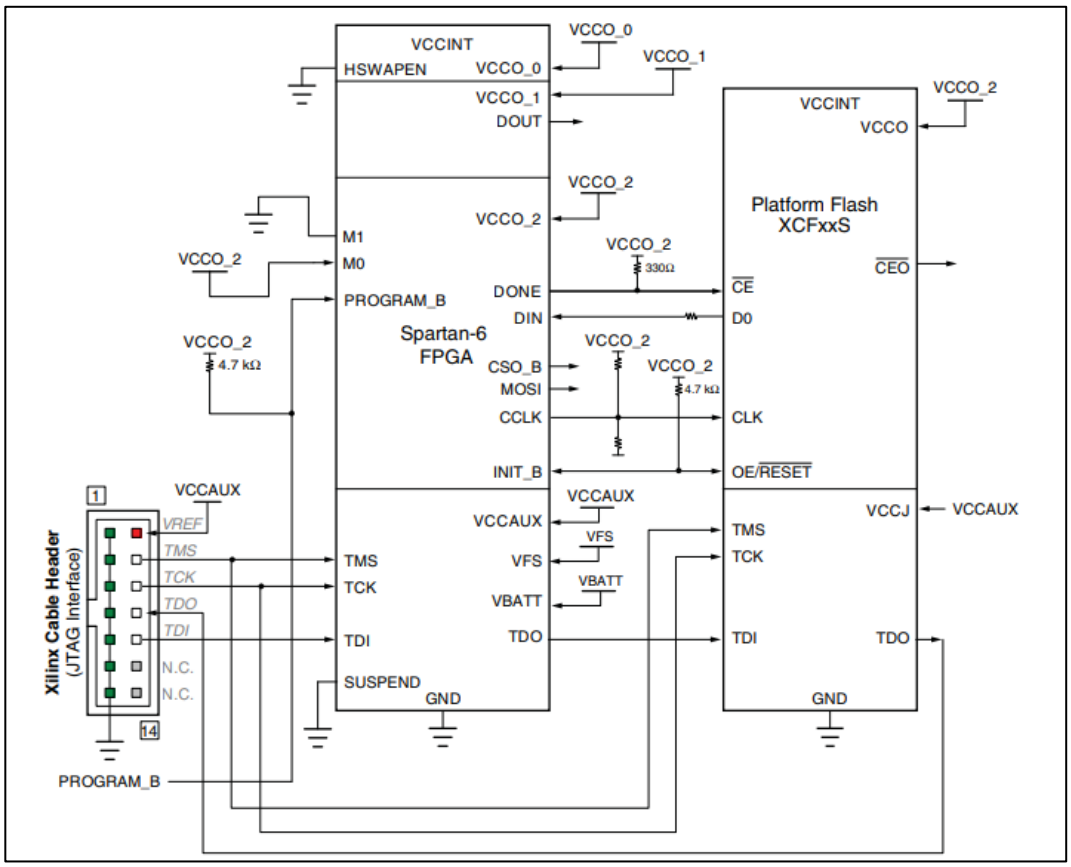
En la figura 2, se muestra el chip de la FPGA (en este caso la Spartan-6 de Xilinx) junto con los componentes externos que son necesarios para su funcionamiento. En la parte derecha de la imagen se puede observar el bloque de la CFM que utiliza Xilinx para su paquete de FPGA en el cual se guarda la

configuración de la FPGA Spartan-6. En la parte inferior izquierda, puede apreciarse el JTAG Interface (que es donde se conecta el cable que va desde la PC hacia la tarjeta del Xilinx) donde se envía toda la información de la configuración de hardware hecha por el usuario hacia la CFM para ser utilizada cada vez que se enciende la tarjeta.

En todo este diagrama se puede observar las diversas conexiones de la fuente de alimentación que se debe realizar para cada componente, así como también las conexiones entre componentes y las conexiones de los osciladores para sincronizar la transmisión de datos del PC-CFM y de la CFM-Spartan-6 FPGA.

Toda esta configuración se debe realizar para que cualquier FPGA sea funcional por lo cual muchos de los usuarios de FPGA deciden comprar los paquetes que ya traen todos los componentes externos ya adaptados a la FPGA para ahorrar inconvenientes.

Figura 2. **FPGA y componentes externos**



Fuente: XILINX. *Spartan-6 FPGA Configuration*. p 26.

2.1. Ventajas y desventajas de una FPGA

Como cualquier otro dispositivo electrónico la FPGA cuenta con aspectos a su favor y con otros en su contra, por lo cual, su implementación queda a criterio del profesional, al evaluar el tipo de trabajo y objetivo que se desea alcanzar.

Como referencia, se analizarán algunas ventajas y desventajas de las FPGA en general.

- Ventajas:

    - Paralelismo: las líneas de comando se ejecutan todas al mismo tiempo, es decir, la ejecución de los comandos no se hace de manera cíclica como en un microcontrolador donde por cada ciclo de reloj se ejecuta una línea de comando. Este proceso aumenta la productividad de la FPGA ya que puede realizar varias rutinas y subrutinas al mismo tiempo por lo cual el tiempo de ejecución es menor.

    - Velocidad: debido a las capacidades de la FPGA y las diversas aplicaciones para las cuales se utiliza, ésta posee una velocidad de procesamiento mucho mayor a los microcontroladores y otros dispositivos disponibles en el mercado. Por ejemplo, en el análisis de señales se requiere una alta velocidad para procesar toda la información proveniente del muestreo externo a la FPGA como sucede de forma similar en el procesamiento de imágenes.

    - Field Programable: posee la flexibilidad de reconfigurar y ajustar su hardware a las exigencias que sean requeridas. Por ejemplo, si se requiere que se añadan más canales para el procesamiento de señales, solo debe reconfigurarse el hardware sin la necesidad de invertir más dinero en nuevos dispositivos para añadir dicha función.

    - Múltiples y reconfigurables pines I/O: al poseer una gran cantidad de terminales para poder ser utilizada como entrada o salida de datos, la FPGA se vuelve una excelente opción si se requiere un gran uso de éstas. Muchos de los dispositivos electrónicos disponibles en el mercado tienen limitadas terminales para entrada

y salida y muchas de ellas no son reconfigurables. Un claro ejemplo son los microcontroladores en los cuales se requiere del método de multiplexación para poder controlar una matriz gran dimensión, luego reconfigurarlos para poder utilizarlos en otra tarea o utilizar otro microcontrolador para ayudar a realizar varias tareas además de controlar la matriz. Con la FPGA se dispone de una gran cantidad de terminales con la ventaja de que cuenta con una gran tasa de velocidad para la transmisión de datos.

- Desventajas

  - Costo elevado: el monto que debe invertirse para solo el chip es elevado debido a las cualidades y hardware interno que posee. Si a eso se le suma el costo de los otros dispositivos externos que se necesitan para su funcionamiento como lo son la memoria flash y los osciladores el costo es mayor. También se tiene la opción de comprar una tarjeta entrenadora que posee la FPGA y todos los demás dispositivos electrónicos necesarios para su funcionamiento como lo son las FPGA de Xilinx o también la FPGA libre pero su costo sigue siendo alto.

  - Consumo de energía: su consumo inicia desde el momento en que se enciende debido al gran número de componentes internos que posee. Desde el momento en que se descarga la configuración de los CLB, wires y los IOB de la memoria flash externa, se necesita energía para polarizar cada una de las compuertas lógicas internas del diseño de hardware que se ha creado.

- Complejidad: el aprender o utilizar una FPGA no es fácil, debido a que se debe de cambiar a una forma de pensar lógica para poder sintetizar hardware. En un microcontrolador se utilizan líneas de código en un lenguaje específico (por ejemplo, C, C++, Asembler, etc.) para decirle al dispositivo qué es lo que debe de hacer, dónde obtener la información y por dónde o cómo debe de enviar información. En una FPGA se sintetiza hardware, para lo cual, se utiliza un lenguaje cuya función es describir hardware e indicarle a la FPGA que componentes de los CLB deben utilizarse y como deben conectarse con otros para hacer funciones específicas y qué terminales se utilizan para recibir o enviar datos y de qué manera se necesita hacerlo.

- Incompatibilidad: cada fabricante posee una arquitectura propia para sus chips lo que ocasiona que sea difícil comparar de forma objetiva cada FPGA que hay en el mercado.

- Lenguaje complicado: HDL es el lenguaje de descripción de hardware utilizado para desarrollar hardware en una FPGA. Para los estudiantes o profesionales de la electrónica digital es complicada la transición de desarrollar hardware utilizando componentes digitales visibles como flip-flops, compuertas, contadores, etc. hacia el desarrollo de hardware utilizando un lenguaje que es similar al utilizado para desarrollar software. Para evitar este problema se debe tener un buen nivel de comprensión de la forma en que trabaja la FPGA y como la herramienta de software propia de la FPGA traduce el código del HDL a un circuito complicado con sus respectivos componentes de lógica digital lo

cual resulta complicado para quienes se adentran en el mundo del desarrollo de hardware con una FPGA.

2.2. Tipos de FPGA

Por un lado, están las FPGA las cuales se les llamará “comerciales” y por el otro las “FPGA libres” las cuales han despegado en el mercado dando una nueva opción a los desarrolladores y entusiastas de hardware.

2.2.1. FPGA comerciales

Se encuentran en el mercado desde que la FPGA se volvió un dispositivo a disposición de todos y que comparten ciertas características las cuales las agrupan en esta categoría. Generalmente, las características que agrupan a estas FPGA en “comerciales” son algunas desventajas: la incompatibilidad y el lenguaje complicado.

A diferencia de las FPGA libres tienen la ventaja de ser más potentes, a esto se le suma su tiempo en el mercado lo cual las posiciona como una opción más segura al momento de decidir adquirir una FPGA.

Para el análisis y comparativa de los tipos de FPGA, se tomará como muestra de análisis para las PFGA comerciales la FPGA Spartan-6 de Xilinx en el kit de tarjeta de desarrollo. Este kit de FPGA de Xilinx es utilizado en el Laboratorio de Electrónica de la Escuela de Mecánica Eléctrica de la USAC para los cursos de Electrónica 6, Electrónica Aplicada, entre otros, debido a la capacidad que posee y su utilización para el procesamiento de imágenes y señales en dichos cursos.

2.2.1.1. Spartan-6 Xilinx

Por el lado de las FPGA comerciales la FPGA Spartan-6 de Xilinx tiene varias características que la hacen una buena opción de compra si lo que se requiere es el procesamiento de imágenes o alto rendimiento. Las características internas del chip son:

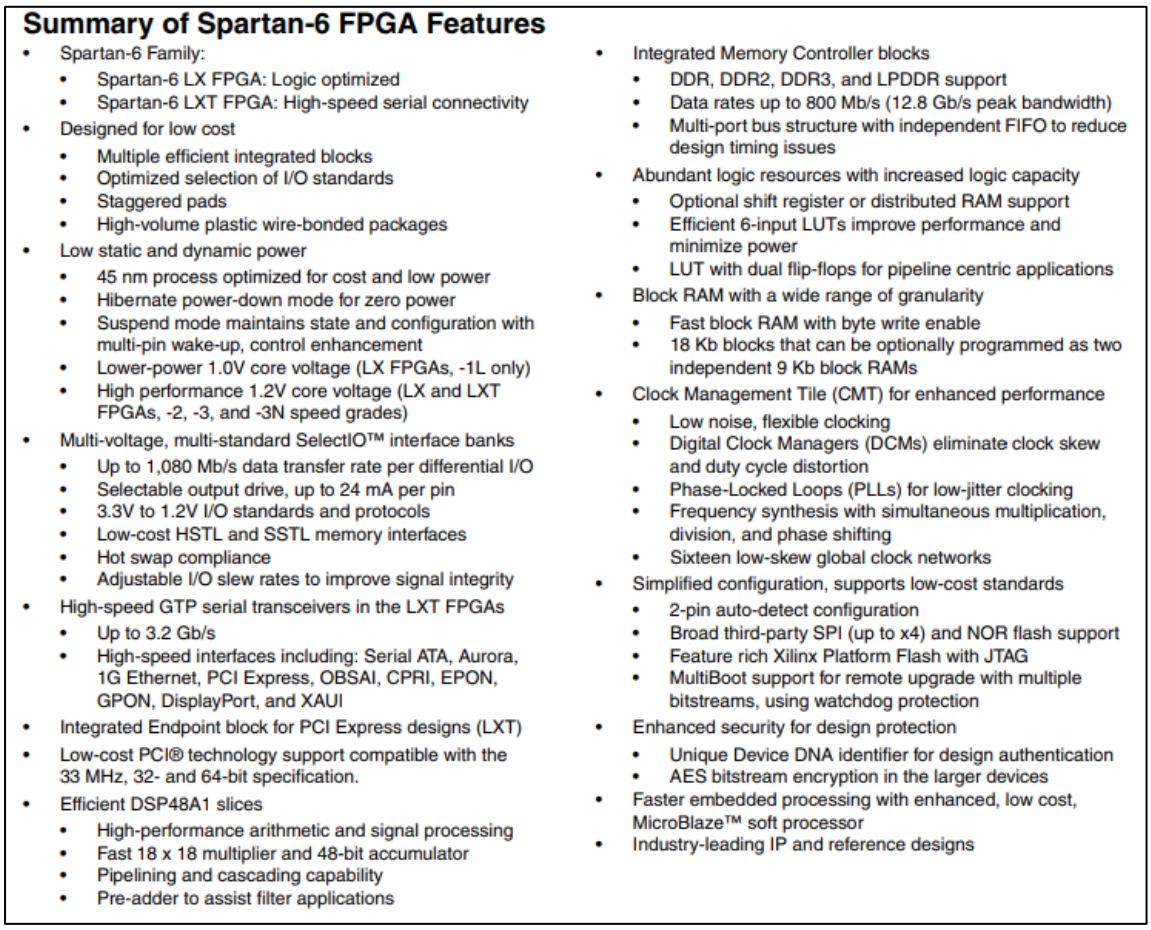
- Logic cells: o celdas lógicas, indican la capacidad de componentes lógicos que tiene disponible la FPGA para el desarrollo de hardware. En este caso, la FPGA Spartan-6 tiene un número mínimo de 3,840 y un máximo de 147,443 celdas lógicas y su disposición depende del modelo de Spartan-6 que se adquiera.

- CLBs: contienen un mínimo de 4800 flip-flops llegando hasta un máximo de 184,304. Estos flip-flops están distribuidos en 600 slices para el mínimo y 23,038 para el máximo. En la arquitectura de esta FPGA cada slice contiene 8 flip-flops y 4 LUTs (o tablas de verdad que indican que salida es para determinadas entradas).

- I/O: tiene como mínimo de 132 y como máximo de 540 I/O que se encuentran distribuidas 4 bancos de I/O para el mínimo y de 6 bancos de I/O para el máximo. La cantidad disponible que puede utilizar el usuario depende del modelo de Spartan-6 que se adquiera.

La figura 3 muestra un resumen de las características de la FPGA Spartan-6 las cuales deben tomarse en cuenta al momento de escoger esta FPGA para asegurarse de que cumple con los requerimientos para alcanzar los objetivos que se esperan para este dispositivo en la tarea que se desea llevar a cabo.

Figura 3. **Características de la FPGA Spartan-6 de Xilinx**



Summary of Spartan-6 FPGA Features

- Spartan-6 Family:
  - Spartan-6 LX FPGA: Logic optimized
  - Spartan-6 LXT FPGA: High-speed serial connectivity
- Designed for low cost
  - Multiple efficient integrated blocks
  - Optimized selection of I/O standards
  - Staggered pads
  - High-volume plastic wire-bonded packages
- Low static and dynamic power
  - 45 nm process optimized for cost and low power
  - Hibernate power-down mode for zero power
  - Suspend mode maintains state and configuration with multi-pin wake-up, control enhancement
  - Lower-power 1.0V core voltage (LX FPGAs, -1L only)
  - High performance 1.2V core voltage (LX and LXT FPGAs, -2, -3, and -3N speed grades)
- Multi-voltage, multi-standard SelectIO™ interface banks
  - Up to 1,080 Mb/s data transfer rate per differential I/O
  - Selectable output drive, up to 24 mA per pin
  - 3.3V to 1.2V I/O standards and protocols
  - Low-cost HSTL and SSTL memory interfaces
  - Hot swap compliance
  - Adjustable I/O slew rates to improve signal integrity
- High-speed GTP serial transceivers in the LXT FPGAs
  - Up to 3.2 Gb/s
  - High-speed interfaces including: Serial ATA, Aurora, 1G Ethernet, PCI Express, OBSAI, CPRI, EPON, GPON, DisplayPort, and XAUI
- Integrated Endpoint block for PCI Express designs (LXT)
- Low-cost PCI® technology support compatible with the 33 MHz, 32- and 64-bit specification.
- Efficient DSP48A1 slices
  - High-performance arithmetic and signal processing
  - Fast 18 x 18 multiplier and 48-bit accumulator
  - Pipelining and cascading capability
  - Pre-adder to assist filter applications
- Integrated Memory Controller blocks
  - DDR, DDR2, DDR3, and LPDDR support
  - Data rates up to 800 Mb/s (12.8 Gb/s peak bandwidth)
  - Multi-port bus structure with independent FIFO to reduce design timing issues
- Abundant logic resources with increased logic capacity
  - Optional shift register or distributed RAM support
  - Efficient 6-input LUTs improve performance and minimize power
  - LUT with dual flip-flops for pipeline centric applications
- Block RAM with a wide range of granularity
  - Fast block RAM with byte write enable
  - 18 Kb blocks that can be optionally programmed as two independent 9 Kb block RAMs
- Clock Management Tile (CMT) for enhanced performance
  - Low noise, flexible clocking
  - Digital Clock Managers (DCMs) eliminate clock skew and duty cycle distortion
  - Phase-Locked Loops (PLLs) for low-jitter clocking
  - Frequency synthesis with simultaneous multiplication, division, and phase shifting
  - Sixteen low-skew global clock networks
- Simplified configuration, supports low-cost standards
  - 2-pin auto-detect configuration
  - Broad third-party SPI (up to x4) and NOR flash support
  - Feature rich Xilinx Platform Flash with JTAG
  - MultiBoot support for remote upgrade with multiple bitstreams, using watchdog protection
- Enhanced security for design protection
  - Unique Device DNA identifier for design authentication
  - AES bitstream encryption in the larger devices
- Faster embedded processing with enhanced, low cost, MicroBlaze™ soft processor
- Industry-leading IP and reference designs

Fuente: Xilinx, *Xilinx DS160 Spartan-6 Family Overview*. p 1.

La familia de FPGA Spartan-6 posee una alta capacidad y varias otras características que la hacen muy competente en el mercado. Al momento de desear adquirir uno de estos dispositivos, se debe tomar en cuenta cuál de todos los que conforman esta familia se adapta al diseño para reducir los gastos y sacar todo el provecho a la FPGA.

2.2.2. FPGA libre

Nace a partir de la necesidad de innovar en el campo de las FPGA y de buscar alternativas en la forma en que se sintetiza el hardware utilizando herramientas de software más amigables. En términos generales una FPGA libre es una FPGA privativa la cual puede configurarse y programarse utilizando solo herramientas de software libre.

Aunque las FPGA tienen varios años de haberse creado, no es una tecnología abierta para la comunidad de hardware libre, los detalles internos de la FPGA no están publicados por lo cual no se pueden crear otras alternativas de software para el desarrollo de hardware, dando como resultado que sólo pueda utilizarse el software privativo de la empresa que creó la FPGA.

Todo este panorama tiene un cambio con el aporte de Clifford Wolf quien utiliza un proceso llamado ingeniería inversa sobre una FPGA Lattice iCE40. Este personaje se asignó la tarea de probar en cada uno de los pines de la FPGA diversos estímulos y ver qué resultado se obtenía como salida en la FPGA, con esto, él logra descifrar que sucede dentro de la FPGA, cambiando la configuración y los parámetros le es posible conocer como modificar el bitstream final.

El bitstream es el archivo que contiene la información de programación de la FPGA, contiene los bits que le indican que uniones se deben realizar para interconectar todos los componentes internos y obtener el hardware que se ha descrito por medio del lenguaje utilizado (como puede ser HDL).

Con todo este trabajo de ingeniería inversa Clifford Wolf desarrolla el proyecto ICESTORM con lo cual libera un toolchain que hace posible programar

la FPGA Lattice iCE40 utilizando solo herramientas libres las cuales permiten pasar del lenguaje de descripción de hardware al bitstream.

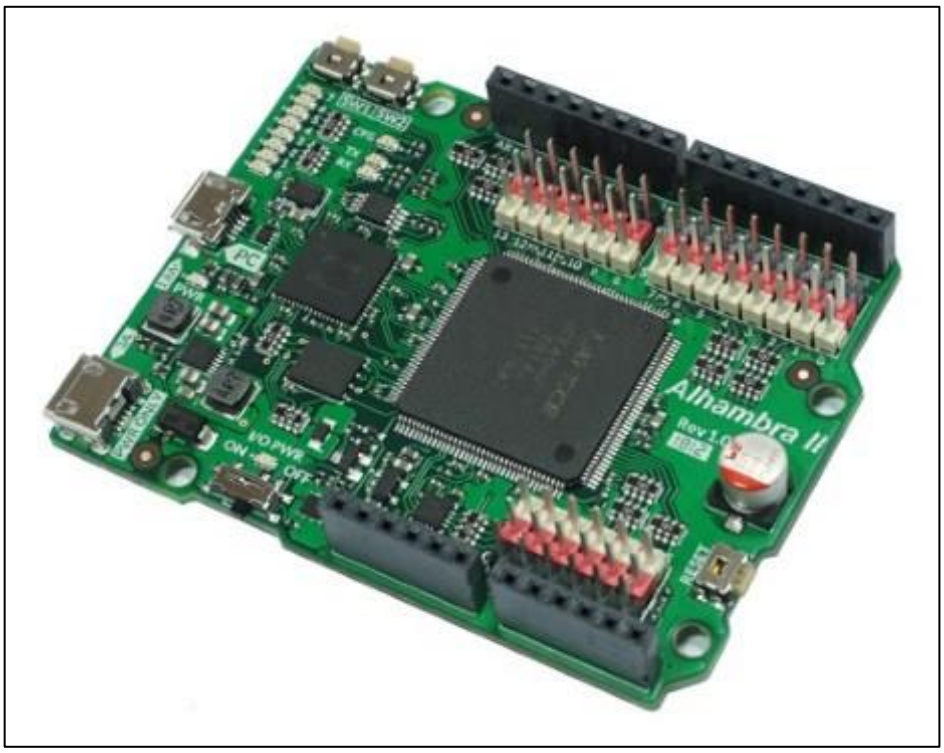
En la actualidad solo se cuenta con herramientas de software libre para programar la FPGA Lattice iCE40, pero se espera que, en un futuro cercano, alguien más pueda descifrar cómo funcionan más FPGA y poder aumentar el número de opciones para las FPGA libres.

2.2.2.1. FPGA libre Alhambra II

Esta tarjeta de desarrollo fue diseñada por Eladio Delgado Mingorance, con el propósito que fuese fácil de utilizar para mackers, estudiantes y profesionales de la electrónica. La tarjeta cuenta con una placa similar a la de Ardruino, en la cual es fácil conectar elementos digitales externos (servomotores, switches, entre otros.).

En la figura 4 se muestra la tarjeta de desarrollo Alhambra II en la cual se nota una semejanza con las tarjetas de Adruino. Para realizar una conexión más fácil de los elementos externos, la tarjeta cuenta con 2 lineas de polarización a un lado de cada puerto I/O. La FPGA que viene instalada en la tarjeta es una Lattice iCE40HX4K que es para la cual se encuentran disponibles las herramientas de software libre para su configuración.

Figura 4. **FPGA libre Alhambra II**



Fuente: FPGAWars, https://raw.githubusercontent.com/FPGAwars/Alhambra-II-FPGA/master/wiki/V1.0/Alhambra-II-01.jpg. Consulta: marzo de 2019

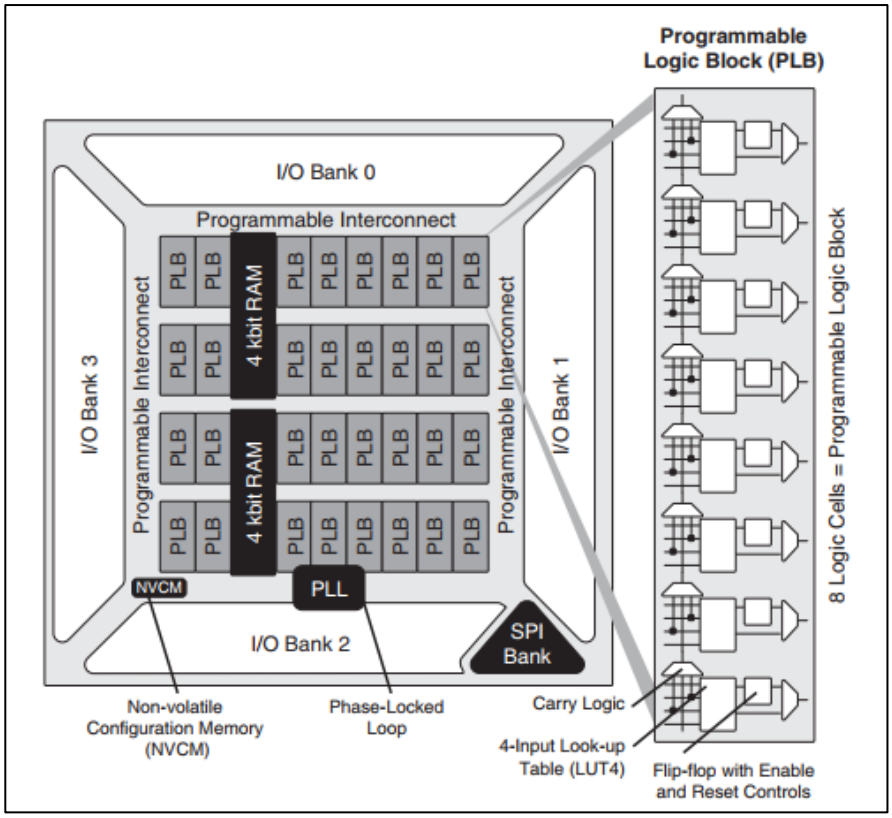
Entre las características de esta tarjeta de desarrollo se encuentran:

- FPGA iCE40HX4K- TQ144 de Lattice.
- Tiene pines de I/O similares al Ardruino One.
- Oscilador de memoria de 12Mhz.
- Switch de encendido/apagado.
- Pines análogos (a través del i2c ADC).
- 20 pines I/O de 3.3v (hasta 5v).
- 8 leds de propósito general.

- 2 push buttons de propósito general.
- Una memoria flash de 4MB.

La arquitectura de la familia de FPGAs Lattice iCE40 se muestra en la figura 5. En esta arquitectura los CLB reciben el nombre de PLB (Programmable Logic Block). Cada uno de los PLB está formado por los elementos de lógica digital básicos para una FPGA (compuertas y flip-flops) además de un juego de LUT4 (Imput Look-up Table de 4 entradas). Se puede observar 4 bancos de puertos I/O de 5 pines cada uno para sumar un total de 20.

Figura 5. **Arquitectura de la familia de FPGA iCE40 de Lattice**

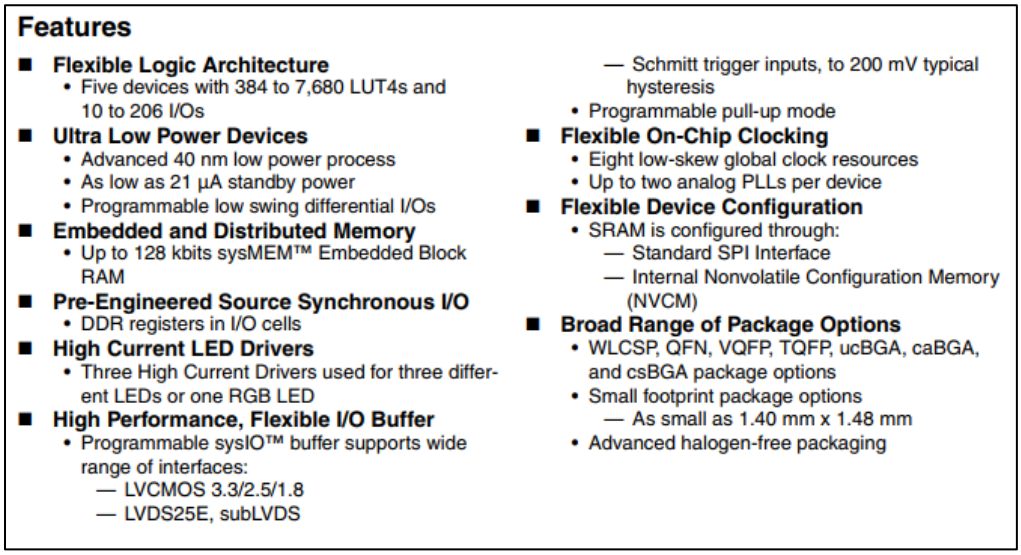


Fuente: Lattice Semiconductor. *iCE40LPHXFamilyDataSheet*, p 2-1.

- Logic Cells: Dependiendo del modelo que se elija de esta familia se pueden tener entre 384 hasta 7680 celdas lógicas (LUT + Flip-Flop).
- Pines I/O: según el modelo tienen un mínimo de 63 y un máximo de 206 pines programables de entrada/salida.

La figura 6 muestra un resumen de las características de familia de FPGA iCE40 de Lattice las cuales deben tomarse en cuenta al momento de escoger esta FPGA para asegurarse de que cumple con los requerimientos para alcanzar los objetivos que se esperan para este dispositivo en la tarea que se desea llevar a cabo.

Figura 6. **Características de la familia de FPGA iCE40 de Lattice**



Features

- **Flexible Logic Architecture**
  - Five devices with 384 to 7,680 LUT4s and 10 to 206 I/Os
- **Ultra Low Power Devices**
  - Advanced 40 nm low power process
  - As low as 21 µA standby power
  - Programmable low swing differential I/Os
- **Embedded and Distributed Memory**
  - Up to 128 kbits sysMEM™ Embedded Block RAM
- **Pre-Engineered Source Synchronous I/O**
  - DDR registers in I/O cells
- **High Current LED Drivers**
  - Three High Current Drivers used for three different LEDs or one RGB LED
- **High Performance, Flexible I/O Buffer**
  - Programmable sysIO™ buffer supports wide range of interfaces:
    - — LVCMOS 3.3/2.5/1.8
    - — LVDS25E, subLVDS
    - — Schmitt trigger inputs, to 200 mV typical hysteresis
  - Programmable pull-up mode
- **Flexible On-Chip Clocking**
  - Eight low-skew global clock resources
  - Up to two analog PLLs per device
- **Flexible Device Configuration**
  - SRAM is configured through:
    - — Standard SPI Interface
    - — Internal Nonvolatile Configuration Memory (NVCM)
- **Broad Range of Package Options**
  - WLCSP, QFN, VQFP, TQFP, ucBGA, caBGA, and csBGA package options
  - Small footprint package options
    - — As small as 1.40 mm x 1.48 mm
  - Advanced halogen-free packaging

Fuente: Lattice Semiconductor. *iCE40LPHXFamilyDataSheet*, p 1-1.

Al observar y comparar las características por parte de la FPGA comercial (FPGA Spartan-6 de Xilinx) y de la FPGA libre (iCE40 de Lattice) se puede observar que existe una gran brecha entre ambas.

Si lo que se busca es una FPGA con mucha capacidad procesamiento y se requiere de muchos CLBs en el diseño, lo mejor es optar por un kit de FPGA comercial como la Spartan-6 de XilinxSi los requerimientos no son muchos y se necesita una FPGA con herramientas de desarrollo más amigables y que sean Open Source, la mejor opción sería una FPGA iCE40 de Lattice en el kit de FPGA de Alhambra II u otro que resulte conveniente.

3. SOFTWARE LIBRE

Una FPGA libre es aquella que puede configurarse y programarse utilizando únicamente software libre. En la actualidad sólo la FPGA iCE40 de Lattice puede denominarse como FPGA libre ya que es la única, que gracias al proyecto ICESTORM, cumple con esta característica.

Para comprender mejor todo el ambiente de la FPGA libre y sus herramientas de software libre, se debe tener claro el concepto de lo que es en sí el software libre.

El software libre es todo aquel software en el cual los usuarios tienen la libertad de ejecutar, copiar, modificar, distribuir, estudiar y mejorar dicho software. El “libre” en el nombre no significa que el software sea gratuito, se trata de dar libertad al usuario sobre la manipulación de dicho software para que sea capaz de adaptarlo a la necesidad que se tenga.

Para considerar un programa como software libre, éste debe cumplir con 4 estatutos denominados “libertades esenciales”, las cuales son:

- La libertad de ejecutar el programa como se desee, con cualquier propósito. Esta libertad esencial señala que cualquier persona u organización es libre de utilizar el software como desee y para la finalidad especifica que desea, sin necesidad de notificar a la persona que creo el software (el programador) sobre que uso se le dará al mismo.

- La libertad de estudiar cómo funciona el programa, y cambiarlo para que haga lo que el usuario quiera. El acceso al código fuente es una condición necesaria para ello. En esta libertad se señala que el código fuente debe ser accesible para que otras personas puedan analizarlo y hacer mejoras en éste de así desearlo. No cuenta como código fuente todo aquel que haya sido modificado para que sea más difícil entenderlo o analizarlos por los usuarios.

- La libertad de redistribuir copias para ayudar a otros. La tercera libertad señala sobre la libertad que se tiene de modificar el programa y distribuir las versiones modificadas u originales de este, cobrando por ello o hacerlo de forma gratuita, sin la necesidad de notificarlo. También se tiene la libertad de no hacer publica cualquier modificación realizada al programa si el usuario no lo desea.

- La libertad de distribuir copias de sus versiones modificadas a terceros. Esto le permite ofrecer a la comunidad de usuarios la oportunidad de beneficiarse de las modificaciones realizadas por algún programador. El acceso al código fuente es una condición necesaria para ello.

Al tener un mejor entendimiento sobre que es en realidad el software libre, se hace evidente el porqué de su uso en la FPGA iCE40 de Lattice. Con la utilización del software libre en esta FPGA se les brinda a los desarrolladores de hardware la posibilidad de mejorar el software basado en sus experticias al utilizarlo (lo cual es una de las libertades del software libre) y con esto también los demás usuarios se benefician de las mejoras en el software realizadas y compartidas por otros usuarios.

Como resultado del proyecto ICESTORM y de la iniciativa de muchos desarrolladores de hardware, nacen varias herramientas de software libre que junto a la FPGA ICE40 de Lattice mejoran la experiencia de desarrollo de hardware creando un ambiente más amigable y fácil para utilizar la FPGA no solo para las personas que ya son diestras en cuanto a la electrónica digital, también para los estudiantes o personas aficionadas a la electrónica que recién inician su camino a través de esta rama de la tecnología.

Entre los programas de software libre disponibles se encuentran: IceStrudio, Apio IDE, Icarus Verilog, GTKWave, icestrom. Todos estos programas forman el paquete de software libre disponible actualmente para la FPGA ICE40 de Lattice, dicho de otra forma, para la FPGA libre.

3.1. GNU/LINUX

Desde que se tiene la opción de utilizar programas de software libre junto a una FPGA, un sistema operativo de software libre es una excelente opción para iniciar este ciclo, ya que, a diferencia de Windows y Mac, no se necesita hacer un gasto monetario en una licencia para utilizar un sistema operativo potente, amigable y sobre todo que posee todas las actualizaciones que se ofrecen además de contar con toda una gama de programas de software libre que son libres de pago.

El movimiento de software libre tiene sus inicios con el proyecto GNU, el cual, tenía como objetivo crear un sistema libre que fuese parecido a Unix, un sistema operativo que entre sus características se encuentra: ser multitarea, portable y multiusuario.

Del proyecto GNU nace un conjunto de herramientas las cuales se complementan con el núcleo LINUX, que al unirse se obtiene el sistema GNU/LINUX.

Basado en GNU/LINUX se puede encontrar diversos sistemas operativos diferentes pero que al mismo tiempo comparte una misma base, un núcleo LINUX y las herramientas de software libre de GNU.

3.1.1. Debian GNU/LINUX

Debian es un sistema operativo que tiene un núcleo Linux y posee todas las herramientas de software libre características de GNU. Este sistema operativo nace del “proyecto Debian” el cual está formado por un gran número de voluntarios cuyo objetivo es ponerlo a disposición de todos en internet para que pueda ser modificado y distribuido libremente siempre que se respete su licencia, en otras palabras, buscan crear un sistema operativo de software libre.

Al igual que UNIX, Debian es multitarea, multiusuario y portable. Puede ser instalado mediante un CD, DVD, USB o directamente desde la red. Cuenta con una gran cantidad de programas de software libre preinstalados y varios más disponibles para su descarga los cuales van desde software de oficina, multimedia, ocio y mucho más.

La versión más reciente es Debian 9.8 la cual está disponible desde febrero de 2019. Entre las arquitecturas que soportan Debian se encuentran:

- PC de 64 bits (amd64).
- PC de 32 bits (i386).
- EABI ARM (armel).

- ABI ARM (armhf).
- MIPS (Little endian).
- MIPS (big endian).
- IBM System z.
- ARM de 64 bits (AArch64).
- Procesadores POWER.
- MIPS de 64 bits (Little endian).

Los requerimientos mínimos que necesita el sistema operativo varían dependiendo del tipo de instalación a realizar. Se tiene la opción de instalarlo “con escritorio” (modo gráfico) o “sin escritorio” que es utilizar el sistema operativo por medio de la línea de comando.

Tabla VII. **Requerimientos de Debian**

Tipo de instalación	RAM mínima	RAM máxima	Disco Duro
Sin escritorio	64 Mb	256 Mb	1Gb
Con escritorio	128 Mb	512 Mb	5 Gb

Fuente: Debian.org, https://www.debian.org/releases/stretch/armel/ch03s04.html.es. Consulta: abril de 2019.

Como se muestra en la tabla VII los requerimientos para utilizar Debian sin escritorio (línea de comando) son menores, lo cual, es una ventaja para los usuarios experimentados que tienen la habilidad de utilizar el sistema operativo sin el ambiente gráfico, ahorrando gastos en recursos de hardware. Los requerimientos para utilizar Debian con su ambiente gráfico (con escritorio) son el doble, pero al compararlo con otros sistemas operativos se hace evidente que

Debian no necesita grandes recursos de hardware para poder funcionar de manera eficiente en cualquiera de sus dos tipos de instalación.

3.1.2. Ubuntu

Es un sistema operativo de código abierto basado en la arquitectura de Debian, por lo cual, cuenta con un núcleo Linux. Actualmente se encuentra operativo en las arquitecturas Intel, ARM y AMD.

Ubuntu se destaca por enfocarse en facilitar el uso del sistema operativo a los usuarios promedio, garantizando que las personas con poco conocimiento sobre el uso de software puedan gozar de un sistema operativo de software libre. Por otra parte, para garantizar la experiencia de usuario, Ubuntu cuenta con actualizaciones constantes en las cuales se ocupan de arreglar todo tipo de problemas o “bugs” que han sido encontrados por el equipo de desarrollo y también por los usuarios finales.

Ubuntu cuenta con soporte por parte de Canonical (su patrocinador), el cual se mantiene con la venta de soporte técnico y servicios vinculados no solo a Ubuntu sino también a sus derivados. Los derivados de Ubuntu ofrecen diferentes entornos de trabajo según el gusto del usuario, entre ellos se tiene:

- Kubuntu.
- Xubuntu.
- Ubuntu MATE.
- Edubuntu.
- Ubuntu Studio.
- Ubuntu Gnome.
- Lubuntu.

No solo varía el entorno gráfico del sistema, también los requerimientos de este para poder funcionar, en algunas de estas variantes se llega a requerir mucho menos recursos que en Ubuntu original, por lo tanto, el usuario puede elegir la versión que mejor se adapte a las capacidades de su hardware.

Al igual que Debian, Ubuntu cuenta con un gran catálogo de aplicaciones de software libre las cuales permiten llevar a cabo todo tipo de tareas ya sea de oficina, multimedia, ocio y mucho más.

Al momento de realizar este trabajo de investigación la versión más reciente de Ubuntu es la 18.04 LTS (Long Term Support) que posee 5 años de soporte y la versión 19.04 con 9 meses de soporte y actualizaciones.

Los requerimientos recomendados para el sistema son:

- Procesador dual core de 2GHz o mejor.
- Memoria de 2Gb.
- 25 Gb de espacio libre en el disco duro.
- Acceso a internet para actualizar el sistema al momento de instarlo.
- Un puerto USB o lector de discos para llevar a cabo la instalación.

3.2. ICESTORM

El proyecto ICESTORM tiene como objetivo la ingeniería inversa y la documentación del bitstream de la FPGA Lattice iCE40 dando como resultado una serie de herramientas de software libre con las que se puede analizar y crear archivos de bitstream para este modelo y variantes de esta FPGA.

Las herramientas de ICESTORM son un grupo de programas pequeños utilizados para trabajar con los archivos de bitstream para la ICE40 de Lattice. El paquete completo de herramientas consta de:

- Arachne-PNR: herramienta de ubicación y ruteo.
- Yosys: síntesis de Verilog.
- Herramientas ICESTROM.
    - IcePack/IceUnpack: el programa de IceUnpack se encarga de convertir un archivo .bin de iCE40 a un formato ASCII ICESTORM que contiene bloques de 0 y 1 para para los bits de configuración de la FPGA. Por otra parte, el programa de IcePack hace lo contrario, convierte del formato ASCII a un archivo .bin de iCE40.
    - IceTime: es una herramienta que se encarga del análisis de temporización iCE40.
    - IceBox: contiene una librería de Python y herramientas para trabajar con los archivos ASCII de ICESTORM para poder acceder a la base de datos del dispositivo.
    - IceProg: es un programa que contiene los controladores para programar basado en FTDI.
    - IceMulti: se encarga de empaquetar en un solo archivo de imagen de inicio multiple iCE40 multiples archivos de bitstream.

- IcePLL: programa encargado de calcular los parámetros de configuración para el PLL de iCE40.

- IceBRAM: es un programa utilizado para el intercambio de contenidos BRAM en archivos ASCII de ICESTORM.

- ChipDB: el IceStorm MakeFile contruye dos archivos los cuales contienen la información necesaria para que el programa arachne-pnr pueda colocar y enrutar el diseño. Luego procede a crear un archivo ASCII de IceStorm para el diseño colocado y enrutado.

3.2.1. Instalación de las herramientas IceStorm en Ubuntu

La forma de instalar estos paquetes de programas es muy sencilla, solo se debe abrir el terminal y seguir los siguientes pasos:

- Se deben instalar algunos programas previos a la instalación de las herramientas (requisitos previos):

  - $ sudo apt-get install build-essential clang bison flex libreadline-dev \ gawk tcl-dev libffi-dev git mercurial graphviz \ xdot pkg-config python python3 libftdi-dev \ qt5-default python3-dev libboost-all-dev cmake.

- Para la instalación de las herramientas IceStrom:

  - $ git clone https://github.com/cliffordwolf/icestorm.git icestorm.
  - $ cd icestorm.
  - $ make -j $ (nproc).

- $ sudo make install.

- Instalación de Arachne-PNR:

  - $ git clone https://github.com/cseed/arachne-pnr.git arachne-pnr.
  - $ cd Arachne-pnr.
  - $ make -j $ (nproc).
  - $ sudo make install.

- Instalación de Yosys:

  - $ git clone https://github.com/cliffordwolf/yosys.git yosys.
  - $ cd yosys.
  - $ make -j $.
  - $ sudo make install.

Se debe crear un archivo /etc/udev/rules.d/53-lattice-ftdi.rules y agregar la siguiente línea, con el objetivo de permitir la carga del bitstream a un iCEstick de Lattice y/o una placa de ruptura iCE40-HX8K como usuario sin privilegios.

- ATTRS {idVendor}==”0403”, ATTRS {idProduct}==“6010”, MODE=“0660”, GROUP=”plugdev”, TAG+”uaccess”.

3.3. Icarus Verilog

Es un software desarrollado por Stephen Williams, quien es un ingeniero en software especializado en controladores de dispositivos y sistemas integrados.

Esta herramienta se utiliza para el análisis y síntesis de Verilog. Funciona como un compilador, el cual, compila el código fuente escrito en Verilog en el formato de destino elegido por el usuario. Es utilizado principalmente en Linux, aunque funciona en muchos otros sistemas operativos.

La versión más reciente se encuentra disponible a través de la fuente y de formularios binarios en el directorio FTP: fpt://icarus.com/pub/eda/verilog/v10/. Las actualizaciones se realizan periódicamente hasta que la versión sea reemplazada por una nueva que sea estable.

3.3.1. Instalación de Icarus Verilog en Ubuntu

Para instalar Icarus Verilog en Ubuntu es necesario hacerlo por medio de la terminal como sigue:

- Agregar el nuevo ppa.
    - $ sudo add-apt-repository ppa:team-electronics/ppa.
- Es necesario hacer una actualización.
    - $ sudo apt-get update.
- Solicitar la instalación del paquete Icarus Verilog.
    - $ sudo apt-get install verilog.
- Como paso opcional queda instalar el paquete GTKWave.

- $ sudo apt-get install gtkwave.

3.4. Apio

Apio es una herramienta multiplataforma de software libre escrita en Python funcional en Linux, Mac Os y Windows. Apio posee como características principales una interfaz de comando amigable para poder verificar, simular, sintetizar y cargar los diseños de verilog en una FPGA libre, a esto se le suma una serie de paquetes predesarrollados y herramientas para configuración de proyectos.

El proceso de instalación de apio es muy sencillo y debe realizarse por medio de Python package manager (pip) lo que define a Python como requerimiento indispensable para el proceso.

3.4.1. Instalación de Apio en Ubuntu

Debido a que el objetivo de este trabajo de investigación es la utilización de herramientas de software libre, solo se exponen las instrucciones de instalación para el sistema operativo Ubuntu. El proceso de instalación se realiza como sigue:

- Por medio del Python package manager (pip) se instala apio.
  - Easy_install pip.
  - $ pip install -U apio.
  - $ sudo pip install -U apio (si se tienen problemas de permisos con el comando anterior).

- Para instalar los drivers FTDI.
    - Apio drivers –ftdi-enable.
- Instalar los drivers serial para FPGAs con interfaz serial.
    - Apio drivers –serial-enable.
- Instalar los paquetes de Apio.
    - Apio install –all.
- Instalar GTKWave.
    - Apio install gtkwave.

3.4.2. Comandos y sus características en Apio

La estructura de los comandos en apio es muy simple: apio [OPTIONS] COMMAND [ARGS]. De la misma forma, si como usuario se tienen dudas respecto a la lista de comandos disponibles en apio, basta con ejecutar el comando “apio” para desplegar una lista con todos los comandos disponibles en la plataforma.

En apio los comandos se agrupan en 4 categorías las cuales son: Options, project commands, setup commands y utility commands.

- Options:

- --version: muestra la versión de Apio.
- --help: muestra el listado de comandos disponibles en Apio.

- Project commands:

  - Apio build: sintetiza el bitstream, genera un archivo bin desde un verilog y un archivo pcf. Los argumentos que se pueden utilizar con este comando son: -b, --board, --fpga, --size, --type, --pack, -p, --project-dir, -v, --verbose, --verbose-yosys, --verbose-arachne.

  - Apio clean: limpia los archivos generados previamente (blif,asc,bin,rpt y out). Los argumentos que se pueden utilizar con este comando son: -p, --project-dir. Los argumentos que se pueden utilizar con este comando son: -a, --all, -t, --top, --nostyle, --nowarn, --warn, -p, --project-dir.

  - Apio lint: filtra el código verilog. No utiliza el archivo pcf. Los argumentos que se pueden utilizar con este comando son: -p, --project-dir.

  - Apio sim: inicia la simulación de verilog utilizando GTKWave. Los argumentos que se pueden utilizar con este comando son: -p, --project-dir.

  - Apio time: análisis del tiempo del bitstream. Se genera un archivo rpt con un reporte topológico de análisis de tiempo de un archivo verilog y pcf. Los argumentos que se pueden utilizar con este comando son: -b, --board, --fpga, --size, --type, --pack, -p, --project-dir, -v, --verbose, --verbose-yosys, --verbose-arachne.

- Apio upload: carga el bitstream a la FPGA. Realiza un descubrimiento y validación automático del chip FTDI dependiendo de la placa seleccionada. Los argumentos que se pueden utilizar con este comando son: -b, --board, --serial-port, --ftdi-id, -s, --sram, -p, --project-dir, -v, --verbose, --verbose-yosys, --verbose- arachne.

- Apio verify: verifica todo el código verilog. Los argumentos que se pueden utilizar con este comando son: -p, --project-dir.

- Setup commands.

  - Apio drivers: habilita o deshabilita los drivers FTDI. En Linux se agrega un archivo que pueda requerir el reinicio o desconectar y conectar de nuevo la placa. Los argumentos que se pueden utilizar con este comando son: --ftdi-enable, --ftdi-disable, --serial-enable, --serial-disable.

  - Apio init: se encarga del manejo de los proyectos creados en Apio. Además del código, un proyecto de Apio puede incluir un archivo de configuración apio.ini y un scrip de Scons Sconstruct. Los argumentos que se pueden utilizar con este comando son: -s, --scons, -b, --board, -p, --project-dir, -y, --sayyes.

  - Apio install: instala paquetes. Automáticamente instala la última versión del paquete, además, las versiones anteriores pueden ser instaladas utilizando la notación pacakge@version. Los paquetes disponibles son: drivers, ejemplos, gtkwave, icestorm, iverilog, scons, system y verilator. Los argumentos que se pueden utilizar con este comando son: -a, --all, -l, --list, -f, --force, -p, --platform.

- Apio uninstall: desinstala paquetes por lo que antes de desinstalar un paquete se solicita al usuario una confirmación. Los paquetes disponibles son: drivers, ejemplos, gtkwave, icestorm, iverilog, scons, system y verilator. Los argumentos que se pueden utilizar con este comando son: -a, --all, -l, --list, -f, --force, -p, --platform.

- Utility commands.

  - Apio boards: muestra la información de las placas de FPGA. Los argumentos que se pueden utilizar con este comando son: -l, --list, -f, --fpga.

  - Apio config: comandos de configuración de apio. Los argumentos que se pueden utilizar con este comando son: -l, --list, -v,--verbose [0|1], -e, --exe, [default|native].

  - Apio examples: administra los ejemplos en verilog por lo que este comando necesita el paquete de ejemplos. Los argumentos que se pueden utilizar con este comando son: -l, --list, -d, --dir, -f, --files, -p, --project-dir, -n, --sayno.
  - Apio raw: ejecuta comandos utilizando los paquetes de Apio. Los argumentos que se pueden utilizar con este comando son: cmd.

  - Apio system: este comando necesita los paquetes “system” disponibles en: https://github.com/FPGAwars/tools-system. Los argumentos que se pueden utilizar con este comando son: --lsftdi, --lsusb, --lsserial, -i, --info,

  - Apio upgrade: verifica la última versión de Apio.

3.5. IceStudio

Es un editor visual para FPGA libre que utiliza como base el conjunto de herramientas desarrolladas por proyecto ICESTORM. La versión más reciente disponible al momento de realizar este trabajo de investigación es la 0.3.3 y se encuentra disponible para Windows, GNU/Linux y Mac OS. IceStudio trabaja junto con Apio para el desarrollo del hardware, por lo que, al momento de la instalación, Python y Apio son instalados como requerimientos para su funcionamiento.

El desarrollo de hardware en la FPGA Alhambra II realizado en este proyecto de investigación se lleva a cabo utilizando esta herramienta de software por lo cual, se profundiza en las características de esta.

3.5.1. Instalación de IceStudio en Windows

La instalación para el sistema operativo Windows es muy simple, solo se debe descargar el archivo .exe desde el github de FPGAWars. La versión 2.7 de Python será instalada automáticamente si no se encuentra instalada previo a la instalación. Debido a que también se realiza la instalación de Apio, en algunos casos los antivirus impiden el proceso por lo cual se recomienda desactivarlos temporalmente en caso de que ocurra un error en la instalación.

- Descargar el archivo icestudio-0.3.3-win64.exe/icestudio-0.3.3-win32.exe (según el sistema operativo que se utilice) desde: https://github.com/FPGAwars/icestudio/releases.

- Desactivar el antivirus de forma momentánea en caso de presentarse un error con Apio.

- Seguir las instrucciones del instalador.

3.5.2. Instalación de IceStudio en Ubuntu

Con el sistema operativo Ubuntu se debe de realizar la instalación de Python y xclip por aparte por medio de la terminal.

- Instalación de IceStudio.
  - $ sudo apt-get update.
  - $ sudo apt-get install Python 2.7.
  - $ sudo apt-get install xclip.
  - Descargar la appimage y volverlo un ejecutable. La appimage se descarga desde: https://github.com/FPGAwars/icestudio/releases. Se descarga ya sea icestudio-0.3.3-linux32.Appimage o icestudio-0.3.3-linux64.Appimage según el sistema operativo que se utilice.
  - $ chmod a+x icestudio-0.3.3-linux**.Appimage.
  - Otra opción de instalación es descargar el archivo .zip en vez del archivo .Appimage. Al descomprimir el archivo se encuentra un script llamado Linux_installer.sh cuya función es registrar los archivos .ice como proyectos de Icestudio.

3.5.3. Instalación de complementos y prueba en la plataforma

Luego de instalarse IceStudio, el siguiente paso a realizar es instalar los paquetes de Apio y los drivers de las placas de FPGA libres disponibles. Todos estos pasos se realizan dentro de la aplicación.

- Instalar la toolchain: Tools>Toolchain>Install. No es necesario tener acceso a internet para este paso.

- Lo siguiente es conectar la placa y habilitar los drivers correspondientes, para lo cual, se necesitan permisos de administrador: Tools>Drivers>Enable.

- Ahora que se tienen instalados los drivers, sólo queda seleccionar la placa que se estará utilizando: Select>Board>Alhambra II.

- Para comprobar que todo funciona, se puede cargar un proyecto desde el apartado de ejemplos: File>Examples>1.Basic>01. One Led.

- Después de abrir el ejemplo solo se debe cargar a la placa (en este caso la Alhambra II): Tools>Verify | Build | Upload.

3.5.4. Herramientas y características

IceStudio se encarga de hacer una serie de conversiones datos para lograr obtener el bitstream que se envía a la FPGA utilizando las herramientas desarrolladas por el proyecto ICESTORM. Al crear el diseño gráfico del hardware, la plataforma obtiene el código en Verilog de la secuencia que se debe seguir

para obtener el comportamiento deseado en la FPGA. El archivo en Verilog se pasa a un archivo pcf del cual se construye el archivo de bitstream que se carga a la placa para configurar la FPGA.

La placa Alhambra II no es la única FPGA libre disponible en el mercado, por lo cual, IceStudio también puede utilizarse con un total de 11 modelos de FPGA libre las cuales son:

- IceZUM Alhambra.
- Nandland Go board.
- iCEstrick Evaluation Kit.
- Alhambra II.
- BlackIce.
- BlackIce II.
- IcoBOARD 1.0.
- Kéfir I iCE40-HX4K.
- iCE40-HX8K Breakout Board.
- TinyFPGA B2.
- TinyFPGA BX.

Figura 7. **Barra de tareas y barra de diseño**

Fuente: elaboración propia utilizando IceStudio.

Como se puede observar en la figura 7, IceStudio presenta una barra de tareas muy sencilla e intuitiva la cual consta de 6 menús desplegables: File, Edit, View, Select, Tools, help. Estos 6 menús permiten la configuración de la plataforma para adecuar el área de trabajo a gusto del usuario según sus necesidades.

En el menú de “File” se encuentran las opciones básicas del manejo del proyecto: crear, abrir y guardar. Se tiene la opción de exportar otros tipos de archivo los cuales son: Verilog, PCF, Testbench, GTKWave, BLIF, ASC, BitStream. Como ayuda al usuario, se tiene un submenú en el cual se encuentra

una serie de ejemplos de desarrollo de hardware utilizando la placa IceZUM Alhambra, los cuales, al abrirlos, la plataforma genera la opción de adaptarlos a la placa con la que se encuentra trabajando para que el usuario pueda realizar los cambios necesarios en cuanto a pines de entrada/salida y cargar el ejemplo con éxito a la placa utilizada.

“Edit” es el menú encargado del control de contenido como lo es seleccionar, copiar, cortar, pegar, rehacer y deshacer. Al final del menú se encuentra “preferencias”, un submenú en el que se puede cambiar el lenguaje de la plataforma.

El menú “Ver” consta de un conjunto de opciones que proporcionan información sobre el proyecto, la placa utilizada o los archivos relacionados a ambos. La opción “Datasheet” abre una página en el navegador la cual contiene todos los datos relacionados a la placa que se está trabajando en la plataforma. “PinOut” es una opción que muestra el mapa de todos los pines de la placa utilizada, los puntos de alimentación, etc. “Recursos de la FPGA” muestra en una barra inferior los recursos que se están utilizando en la FPGA a medida que se trabaja en un proyecto, lo cual resulta muy útil, ya que se puede saber la cantidad de recursos disponibles mientras se avanza en el proyecto.

“Select” como lo indica su nombre, es un menú encargado de la configuración de la placa utilizada para trabajar en la plataforma y consta de 2 submenús. El submenú board permite seleccionar la placa con la cual se está trabajando en la plataforma de un listado de FPGA libres las cuales soporta IceStudio. El segundo submenú es collections el cual permite cargar colecciones de ejemplos o bloques ya desarrollados por otros usuarios para poder utilizarlo en la plataforma.

El menú “Tools” contiene 3 opciones para el proyecto las cuales son: verify, el cual se encarga de verificar el código verilog generado por el diseño, build el cual sintetiza el bitstream en base al diseño realizado y por último upload que sintetiza y carga el bitstream a la FPGA. En el apartado de “toolchain” se encuentran las opciones para actualizar, remover o resetear las herramientas de IceStudio basadas en el proyecto IceStrorm conocidas como toolchain. Al final se muestra la versión de Apio que está utilizando IceStudio. El apartado de “drivers” solo se utiliza para habilitar o no los drivers del FTDI.

En el menú de “help” se muestra la versión de IceStudio, la documentación, el código fuente de la plataforma y un hipervínculo hacia un foro de la comunidad de los usuarios de IceStudio y de FPGA libre.

La plataforma cuenta con una barra de diseño de hardware que dispone con 4 menús desplegables: Basic, Bit, Logic, Setup. Cada uno de este menú posee componentes diferentes los cuales al combinarse forman el diseño digital deseado por el usuario.

En el menú “Basic” se pueden seleccionar las entradas/salidas que se desean para el diseño, colocándole a cada uno el nombre o clave deseado para identificarlo según sea conveniente. “Constant” se encuentra como tercera opción, y puede tener cualquier valor que permita la plataforma. La cuarta opción disponible es “memory”, consiste en 2 columnas donde la primera columna es la dirección de memoria y la segunda columna se utiliza para que el usuario coloque el valor que desea guardar. “Code” permite al usuario configurar un módulo por medio del código en verilog, por lo tanto, antes de utilizarlo la plataforma solicita ingresar el nombre de las entradas, salidas y parámetros, tras lo cual, genera el editor de texto en el que se puede empezar la descripción de hardware con

verilog. “Information” genera un bloque en el que puede escribir o hacer anotaciones referentes al diseño, algo parecido a una etiqueta.

El menú “Bit” posee dos opciones: 0 o 1. El valor de 0 proporciona un estado bajo (0v) y el valor de 1 proporciona un estado alto (3v). Estos estados son permanentes ya que son una constante generada por la plataforma, conectando directamente VCC o GND.

“Logic” es un menú que presenta 3 submenús desplegables: combinational, gate y secuential. En el submenú “combinational” presenta por defecto solo una opción la cual es un multiplexor de 2 entradas, una salida y un selector. El submenú “gate” proporciona por defecto 7 compuertas lógicas: AND, OR, NOT, NAND, NOR, XOR y XNOR. Estas compuertas son fundamentales para diseñar otros bloques que son útiles en la electrónica (esto se detalla más adelante). Por último, el submenú de “secuencial” provee las herramientas de antirrebote, flip-flops tipo D y T.

El último menú disponible es el de “Setup” el cual permite colocar las resistencias de pull-up para los pines que se desean como entradas y un bloque de tri-estado.

4. CONCEPTOS DE ELECTRÓNICA DIGITAL

Para programar la FPGA libre Alhambra II utilizando la herramienta de software IceStudio, es necesario dominar la teoría de los fundamentos de la electrónica digital, ya sea que se utilice el ambiente gráfico para construir el diseño digital o también utilizando código en Verilog, se debe conocer cómo funcionan los componentes digitales que conforman la FPGA para saber cuáles son los límites que se tiene al momento de realizar el diseño.

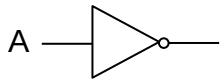
4.1. Compuertas lógicas

Las compuertas lógicas son los elementos básicos en electrónica digital. Se utiliza el álgebra booleana para expresar la salida de las compuertas en términos de las entradas. Cada compuerta lógica combina niveles lógicos (0 o 1) en forma específica, por lo tanto, su comportamiento se puede observar en la tabla de verdad correspondiente a cada compuerta lógica e indica la respuesta de la compuerta a los diversos cambios en la entrada. Los componentes lógicos más complejos utilizados en la electrónica digital no son más que arreglos de compuertas lógicas básicas y cuyo comportamiento se puede predecir utilizando el álgebra booleana y construyendo la tabla de verdad correspondiente. Se consideran como compuertas básicas: AND, NAND, OR, NOR, NOT (inversor).

4.1.1. Compuerta NOT o inversora

La característica principal de la compuerta NOT es la de poseer sólo una entrada y una salida, esto se debe a su función inversora.

Tabla VIII. **Símbolo y tabla de verdad de la compuerta NOT**

Símbolo	Entrada	Salida
	A	Y
	0	1
	1	0

Fuente: elaboración propia.

La compuerta invierte la entrada a la salida dando como resultado el comportamiento expuesto en la tabla VIII. Si en la entrada A se tiene un valor lógico de 0, su salida correspondiente es un 1 lógico y en el caso contrario siendo A un valor lógico de “1” la salida es un valor lógico de 0.

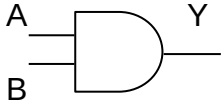
El símbolo correspondiente a la compuerta lógica NOT es de un triángulo acostado con un círculo en la salida. El círculo se denomina “círculo de inversión” el cual indica que se está invirtiendo un valor lógico. El círculo de inversión puede estar en la entrada o en la salida lo cual no causa ningún cambio en la tabla de verdad, pero por conveniencia, en el símbolo de la compuerta se coloca en la salida.

En el álgebra booleana la expresión para la salida de la compuerta NOT es: Y=A' o Y=Ā.

4.1.2. Compuerta AND

Se obtiene en la salida un valor lógico de 1 cuando sus entradas (A y B) son 1. Una forma sencilla de recordar su tabla de verdad es asociando la operación aritmética de multiplicación donde todo valor multiplicado por 0 es 0.

Tabla IX. **Símbolo y tabla de verdad de la compuerta AND**

Símbolo	Entradas		Salida
	A	B	Y
	0	0	0
	0	1	0
	1	0	0
	1	1	1

Fuente: elaboración propia.

El comportamiento de la compuerta AND se muestra en la tabla VIII. La compuerta AND puede tener N entradas pero la salida siempre será la misma (1lógico siempre que todas las entradas estén en 1).

El número de combinaciones lógicas que se pueden presentar se rigen por la siguiente ecuación:

$$Combinaciones = 2^N \; Ecuación\,(1)$$

Donde:

N: número de entradas o variables.

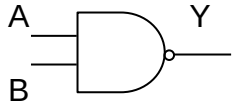
Aplicando la ecuación 1 utilizando N=2 debido a las 2 entradas, se obtiene un total de 4 combinaciones que son exactamente las que pueden observarse en la tabla IX.

En el álgebra booleana la expresión para la salida de la compuerta AND es: Y=AB o Y=A·B. El símbolo de la compuerta AND es parecido a la letra “D” en mayúscula con una salida y un número N de entradas.

4.1.3. Compuerta NAND

Esta compuerta es la unión de la compuerta AND y la compuerta NOT. Su característica es que sólo se obtiene un 0 lógico en la salida cuando todas sus entradas tienen un 1 lógico.

Tabla X. **Símbolo y tabla de verdad de la compuerta NAND**

Símbolo	Entradas		Salida
	A	B	Y
	0	0	1
	0	1	1
	1	0	1
	1	1	0

Fuente: elaboración propia.

Como puede observarse en la tabla X, la salida (Y) de la compuerta NAND es la salida de una compuerta AND a la cual se le aplica una negación o inversión, lo cual, se hace evidente al observar el símbolo de la compuerta NAND que lo conforma una compuerta AND con un círculo de inversión en la salida.

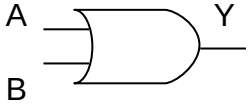
La compuerta puede tener un número N de entradas por lo que el número de combinaciones posibles en la tabla de verdad se rige bajo la ecuación (1), lo único que no varía es la condición para obtener un 0 lógico en la salida, el cual es que todas las entradas tengan un 0 lógico.

En el álgebra booleana la expresión para la salida de la compuerta NAND es: $Y= \overline{A \cdot B}$.

4.1.4. Compuerta OR

La compuerta OR presenta un 1 lógico en la salida cuando cualquiera de sus entradas tenga un 1 lógico.

Tabla XI. **Símbolo y tabla de verdad de la compuerta OR**

Símbolo	Entradas		Salida
	A	B	Y
	0	0	0
	0	1	1
	1	0	1
	1	1	1

Fuente: elaboración propia.

La tabla XI muestra la tabla de verdad con las combinaciones que se pueden presentar en las entradas y la salida correspondiente a cada una observándose como característica que se presenta un 0 lógico solamente cuando todas las entradas de la compuerta tienen un 0 lógico.

El símbolo que caracteriza a la compuerta OR es parecido al símbolo de la compuerta AND con la diferencia en la curva presente del lado de las entradas.

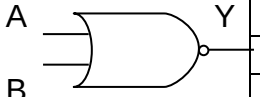
La compuerta OR puede tener N número de entradas y el número de combinaciones que pueden presentarse en la tabla de verdad se rige por la ecuación (1).

En el álgebra booleana la expresión para la salida de la compuerta OR es: Y=A+B. Para recordar de forma sencilla la tabla de verdad de la compuerta OR se puede asociar a la operación aritmética de la suma.

4.1.5. Compuerta NOR

De la misma forma que la compuerta NAND, la compuerta NOR es una combinación de compuertas que en este caso son la compuerta OR y la compuerta NOT.

Tabla XII. **Símbolo y tabla de verdad de la compuerta NOR**

Símbolo	Entradas		Salida
	A	B	Y
	0	0	1
	0	1	0
	1	0	0
	1	1	0

Fuente: elaboración propia.

Como puede observarse en la tabla XII, la salida (Y) de la compuerta NOR es la salida de una compuerta OR a la cual se le aplica una negación o inversión, lo cual, se hace evidente al observar el símbolo de la compuerta NOR que lo conforma el símbolo de la compuerta OR con un círculo de inversión en la salida.

La compuerta puede tener un número N de entradas por lo que el número de combinaciones posibles en la tabla de verdad se rige bajo la ecuación (1), lo único que no varía es la condición para obtener un 1 lógico en la salida, el cual es que todas las entradas tengan un 0 lógico.

En el álgebra booleana la expresión para la salida de la compuerta NOR es: $Y= \overline{A+B}$.

4.2. Lógica combinacional

Se le llama lógica combinacional o combinatoria al proceso de combinar compuertas lógicas con la finalidad de obtener a la salida valores lógicos de unos o ceros según la combinación de valores lógicos de entrada. Un circuito combinacional consta de tres partes: variables de entrada, compuertas lógicas y variables de salida. Tanto las variables de entrada como de salida son especificadas por el diseñador del circuito mientras que las compuertas lógicas a utilizar se determinan según la función booleana obtenida para cada salida.

En los circuitos combinacionales la tabla de verdad juega un valor importante, ya que es el mapa utilizado para ver el comportamiento de las salidas del circuito respecto a cada combinación posible que puede presentarse en las entradas, para lo cual la ecuación (1) sigue siendo útil para determinar el número de combinaciones y poder especificar el valor de salida que corresponde a cada una.

Un circuito se considera combinacional cuando el diagrama del circuito tiene compuertas lógicas sin ninguna retroalimentación y sin unidades de memoria, la presencia de cualquiera de estos elementos clasifica al circuito como secuencial. Para obtener la función booleana que describe la salida con respecto a las

entradas del circuito se pueden utilizar 2 métodos: álgebra booleana o mapas de Karnaugh.

4.2.1. Mapas de Karnaugh

El mapa de Karnaugh es un método utilizado para obtener la función booleana reducida de la salida de un circuito combinacional en base a sus entradas. En este método se utiliza una matriz cuyas dimensiones están determinadas por las variables de entrada del circuito combinacional. Cada celda de la matriz se llena con un valor de 1 o 0 lógico pertenecientes a la salida del circuito combinacional a la que se busca obtener su función booleana reducida.

El proceso de resolución se basa en agrupar los 1 adyacentes siguiendo las siguientes reglas:

- El tamaño de cada grupo debe de ser de 1, 2, 4, 8, 16 … 2^n.
- Los 1 en las columnas laterales de la matriz pueden agruparse siguiendo la regla anterior.
- Los 1 en la filas superior e inferior de la matriz pueden agruparse siguiendo la primera regla.

Para describir el método de resolución por mapas de Karnaugh se utiliza la tabla III del capítulo 1, tomando la salida del segmento “a” del display y encontrando su función booleana.

El primer paso es definir el número de entradas y salidas del circuito según lo requiera el problema a resolver, en este caso se tienen 3 variables de entrada y 7 variables de salida correspondientes a cada segmento del display.

El segundo paso es construir la matriz donde se colocan los unos y ceros que forman cada salida del circuito. Para este ejemplo cada matriz consta de 4 columnas y 2 filas.

Tabla XIII. **Matriz de mapa de Karnaugh**

C/BA	00 / B'A'	01/ B'A	11/ BA	10/BA'
0 / C'	1	0	1	1
1 / C	0	1	1	0

Fuente: realización propia.

La tabla XIII muestra la forma en la que quedan colocados los unos y ceros, en la parte superior de cada columna esta rotulada la posición/variables que corresponde, del mismo modo cada fila esta rotulada con su posición/variable. La posición 3 y 2 de las columnas se intercambia, del mismo modo se efectúa el mismo cambio si la tabla tuviese 4 filas.

Como tercer paso se combinan los unos en grupos de 1, 2, 4, entre otros, según corresponda. Una característica importante es que la tabla se toma como continua así que la primera y última columnas están conectadas, por lo que se pueden agrupar los 1 de las orillas.

Tabla XIV. **Matriz resultante**

C/BA	00 / B'A'	01/ B'A	11/ BA	10/BA'
0 / C'	1	0	1	1
1 / C	0	1	1	0

Fuente: elaboración propia.

La tabla XIV es la matriz resultante, por lo que el cuarto paso consiste en tomar cada grupo y sacar su función booleana correspondiente para finalmente sumar cada expresión y formar la función booleana final. Para obtener la función booleana de cada grupo, se observa que variables de entrada son comunes en las columnas y final que forman al grupo. Por ejemplo, si toma el grupo formado por los unos de las columnas laterales, se observa que las variables comunes son A' y C', la multiplicación de los términos comunes forma la función booleana que en este caso es: A'C'.

Repitiendo el proceso con los 2 grupos restantes y sumando todas las funciones booleanas en una sola, se obtiene la función booleana del segmento “a” del display, el resultado es: segmento a= BC'+AC+A'C'.

Para encontrar la función booleana del resto de segmentos del display que se encuentran en la tabla III, se repite todo el proceso, construyendo una nueva matriz para cada segmento, agrupar los unos de cada matriz y formar la función boolena con la suma de cada producto de los grupos de unos.

4.3. Multiplexor y demultiplexor

El multiplexor y demultiplexor son circuitos combinacionales con funciones opuestas, por lo que su aplicación en el diseño digital el variada.

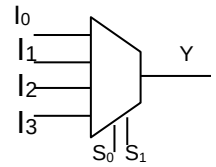
4.3.1. Multiplexor

El multiplexor es un circuito combinacional que se encarga de transportar información de forma binaria (unos y ceros) de uno de varios canales de entrada hacia un único canal de salida. El canal de entrada del cual se transporta la

información se selecciona por medio de un N número de variables o líneas de selección.

Existe una relación entre el número de líneas/canales de entrada y las líneas/variables de selección dado por la ecuación (1) de donde el número de entradas disponibles es 2^N.

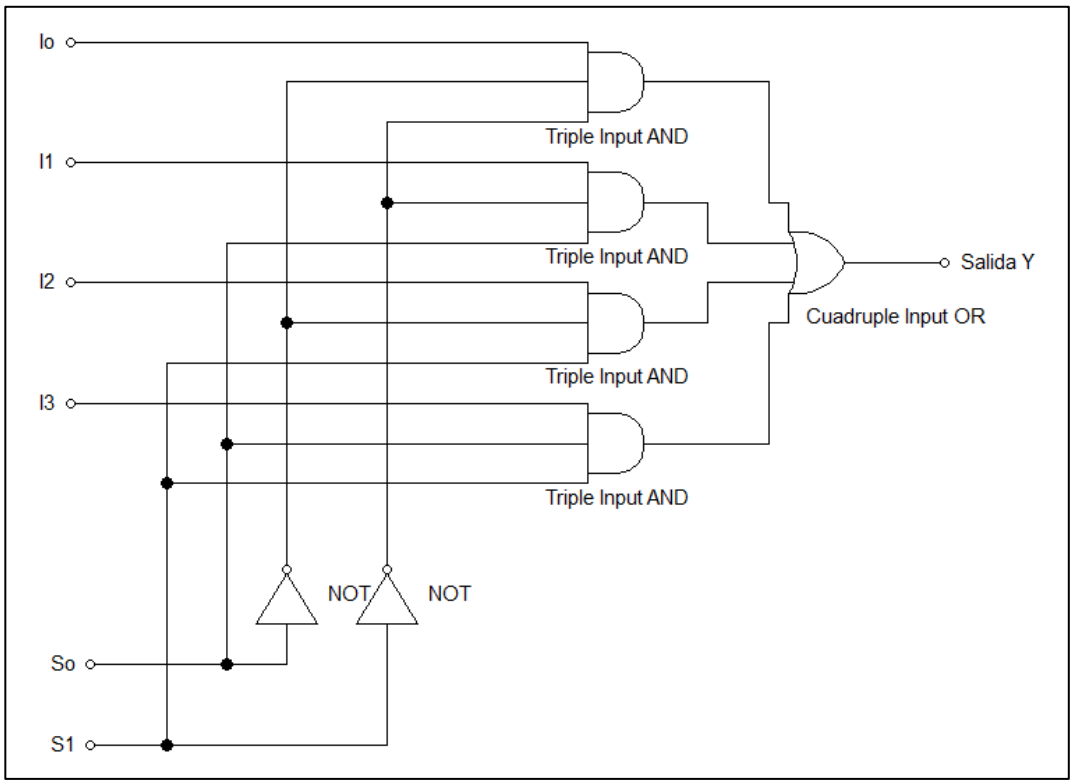
Tabla XV. **Símbolo y tabla de verdad multiplexor 4:1**

Símbolo	Selectores		Salida
	S_o	S_1	Y
	0	0	I_0
	0	1	I_1
	1	0	I_2
	1	1	I_3

Fuente: elaboración propia.

La tabla XV muestra el símbolo para un multiplexor de 4:1 (4 entradas y 1 salida) seguido de la tabla de verdad correspondiente. Las salidas I_0, I_1, I_2 e I_3 son también las entradas, esto se debe a que se desea en la salida el mismo dato que se tiene en la entrada seleccionada. El circuito combinacional que se encuentra internamente de un multiplexor consiste en un arreglo de compuertas básicas que se encargan de permitir el paso de datos binarios de una entrada u otra según el código binario presente en las entradas de selección.

Figura 8. **Diagrama con compuertas lógicas del Mux 4:1**



Fuente: elaboración propia.

La figura 8 muestra el diagrama interno de un Mux 4:1 compuesto por compuertas lógicas básicas. La primera sección del diagrama muestra 4 arreglos de compuertas AND que tienen conectada una entrada de información asignada y las dos entradas de selección. Si se desea en la salida Y la información de la entrada I_0, las dos entradas de selección deben estar en 0 según la tabla XV, por lo tanto, las dos entradas de selección deben negarse utilizando compuertas NOT para que en la salida se tenga un 1 lógico, estas salidas se multiplican con la entrada I_0, teniendo así una salida de 1 cada vez que I_0 tenga un valor lógico de 1. La salida de este arreglo se suma con la salida de los otros 3 arreglos de compuerta AND para finalmente sumarse en un arreglo de compuertas OR. La

salida Y mantiene el valor de I_0 siempre que los valores de las entradas de selección no cambien, en caso contrario, la salida Y tendrá el valor de la entrada seleccionada según la tabla XV.

Para el diseño de un multiplexor con más entradas se debe tomar en cuenta los cálculos necesarios utilizando la ecuación (1) para obtener el número de entradas de selección acorde al número de entradas de información que se desea. La estructura del diagrama interno solo se expande agregando más arreglos de compuertas AND para cada entrada de información y de selección necesaria, así como también las compuertas OR extra necesarias en el arreglo final del diseño.

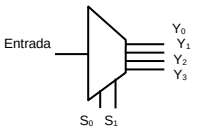
4.3.2. Demultiplexor

El demultiplexor tiene la función contraria al multiplexor, su función es enviar la información de una sola entrada hacia una de las 2^N salidas disponibles en el multiplexor por medio de las N líneas de selección.

El demultiplexor es un circuito combinacional que tiene la función contraria al multiplexor, se encarga de transportar información de forma binaria (unos y ceros) desde el canal/línea de entrada hacia uno de varios canales/líneas de salida. El canal/línea de salida se selecciona por medio de un N número de variables o líneas de selección.

Existe una relación entre el número de líneas/canales de salida y las líneas/variables de selección dado por la ecuación (1) de donde el número de líneas/canales de salida disponibles es 2^N.

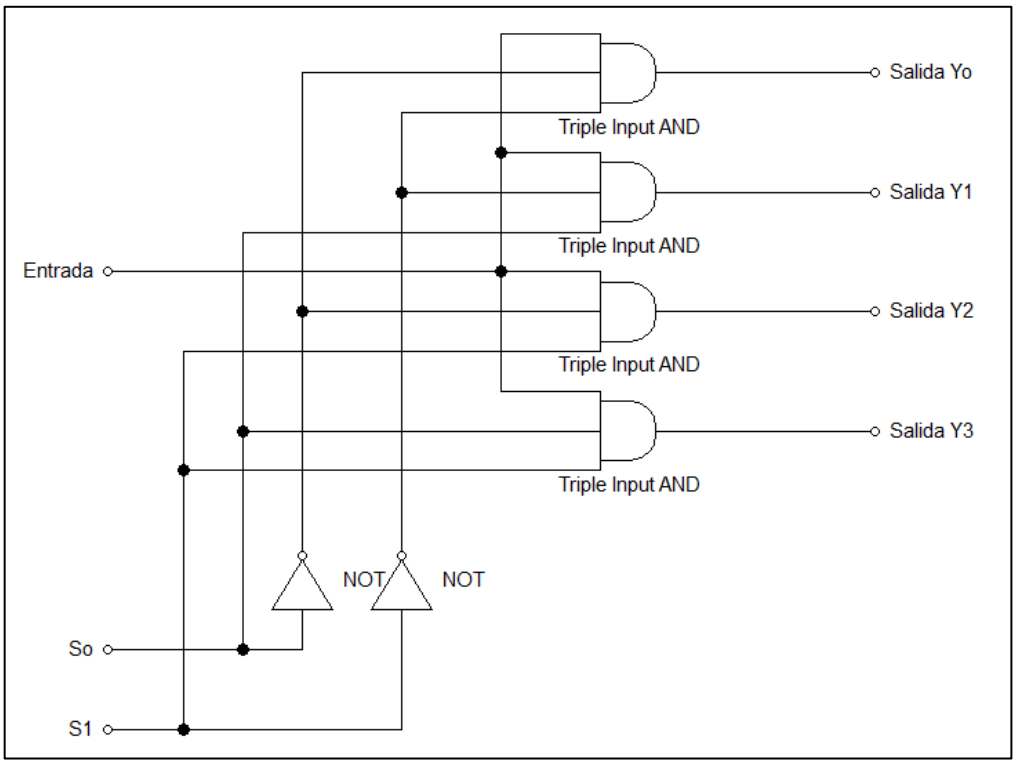
Tabla XVI. **Símbolo y tabla de verdad del demultiplexor**

Símbolo	Selectores		Salidas			
	S_0	S_1	Y_0	Y_1	Y_2	Y_3
	0	0	Entrada	0	0	0
	0	1	0	Entrada	0	0
	1	0	0	0	Entrada	0
	1	1	0	0	0	Entrada

Fuente: elaboración propia.

La tabla XVI muestra el símbolo para un demultiplexor de 1:4 (1 entrada y 4 salidas) seguido de la tabla de verdad correspondiente. El circuito combinacional que se encuentra internamente de un demultiplexor consiste en un arreglo de compuertas básicas que se encargan de permitir el paso de datos binarios de la entrada hacia una de las salidas según el valor presente en los selectores.

Figura 9. **Circuito combinacional del demultiplexor**



Fuente: elaboración propia.

La figura 9 muestra el diagrama interno del demultiplexor 1:4 el cual consiste en un circuito combinacional compuesto por compuertas lógicas básicas. El circuito se basa en 4 arreglos de compuertas AND que en común la entrada de datos. Si se desea en la salida Y_0 la información de la entrada, las dos entradas de selección deben estar en 0 según la tabla XVI, por lo tanto, las dos entradas de selección deben negarse utilizando compuertas NOT para que en la salida se tenga un 1 lógico, estas salidas se multiplican con la entrada, teniendo así una salida de 1 cada vez que en la entrada se presente un valor lógico de 1. La salida Y_0 mantiene el valor de la entrada siempre que los valores de las entradas de selección no cambien, en caso contrario, la información de la entrada estará

presente en cualquier otra salida según el valor presente en las entradas de selección correspondientes a la tabla XVI.

Para el diseño de un demultiplexor con más salidas se debe tomar en cuenta los cálculos necesarios utilizando la ecuación (1) para obtener el número de entradas de selección acorde al número de salidas de información que se desea. La estructura del diagrama interno solo se expande agregando más arreglos de compuertas AND para cada salida de información y las entradas de selección necesarias.

4.4. Comparador de magnitudes

Este circuito combinacional compara dos números A y B de N bits obteniendo a la salida tres respuestas posibles: A=B, A>B y A<B.

Para diseñar un comparador de magnitudes para números de N bits se debe seguir un algoritmo para diseñar un circuito combinacional para cada salida del comparador (A=B, A>B y A<B). Para ejemplificar el algoritmo y la forma de aplicarlo, se toma como ejemplo el diseño de un comparador de magnitudes de 4 bits.

El primer paso es definir los coeficientes de los números del más al menos significativo, la variable para identificar cada número es independiente del proceso por lo que en este ejemplo se utilizan las variables A y B para cada número. Siguiendo lo anterior se define el número A = $A_3A_2A_1A_0$ y el número B= $B_3B_2B_1B_0$.

El algoritmo para el circuito combinacional de la función A=B se obtiene realizando una comparación de cada uno de los pares de bits (A_3=B_3, A_2=B_2, etc)

y si todos resultan siendo iguales significa que tanto A como B son iguales. Esta comparación se lleva a cabo por medio del siguiente algoritmo:

$$X_i = A_i B_i + A'_i B'_i \; para \; i = 1,2,3 \ldots N \;\; Ecuación \; (2)$$

Donde:

X_i = es el resultado de la operación con cada bit de ambos números en la posición i.

Para obtener el resultado final, se hace una operación AND entre todos los resultados de aplicar la ecuación (2) a cada par de bits en cada posición i. A será igual a B cuando el resultado de la operación AND sea 1. El algoritmo para la función A=B es el siguiente:

$$(A = B) = X_0 X_1 X_2 X_3 \ldots . X_i \; Ecuación \; (3)$$

Al aplicar la ecuación (3) al ejemplo planteado, se obtiene: (A=B) = $X_0X_1X_2X_3$, ya que son solo 4 bits los que componen cada número comparado.

En el caso de que A≠B, se deben explorar otras opciones las cuales son que A>B o A<B. Para ambos casos se tiene un algoritmo que se plantea según el número de bits máximo que componen los números comparados. Los algoritmos correspondientes a cada función son:

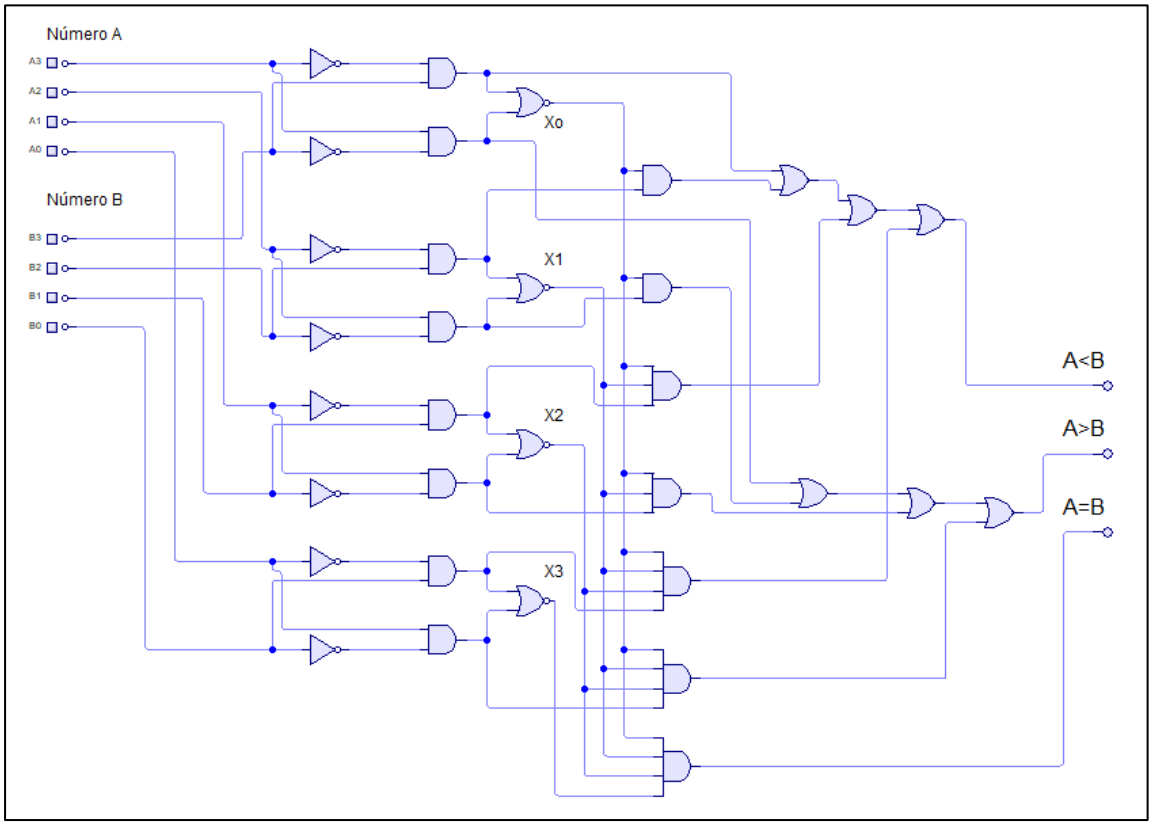
$$(A > B) = A_3 B'_3 + X_3 A_2 B'_2 + X_2 A_1 B'_1 + X_1 A_0 B'_0 \;\; Ecuación \; (4)$$

$$(A < B) = A'_3 B_3 + X_3 A'_2 B_2 + X_2 A'_1 B_1 + X_1 A'_0 B_0 \;\; Ecuación \; (5)$$

Se puede observar que el resultado de la ecuación (2) se utiliza tanto en la ecuación (4) como en la ecuación (5) lo cual facilita el diseño del comparador ya

que las salidas de las operaciones utilizadas para la función (A=B) pueden utilizarse para las funciones (A>B) y (A<B).

Figura 10. **Circuito combinacional del comparador de magnitudes**



Fuente: elaboración propia.

La figura 10 muestra el circuito combinacional completo resultado de implementar las ecuaciones 2,3,4 y 5. El circuito para la comparación de números con más o menos bits afecta directamente al número de compuertas lógicas a utilizar debido a que el algoritmo se extiende o minimiza según el caso.

4.5. Flip-flop

Los flip-flops son pequeñas unidades de memoria que mantienen un valor binario a su salida de forma indefinida siempre que el circuito se encuentre alimentado por voltaje. El valor de salida cambia solo cuando una señal se lo indica al flip-flop, por lo que el nombre que recibe esta señal es “disparo”.

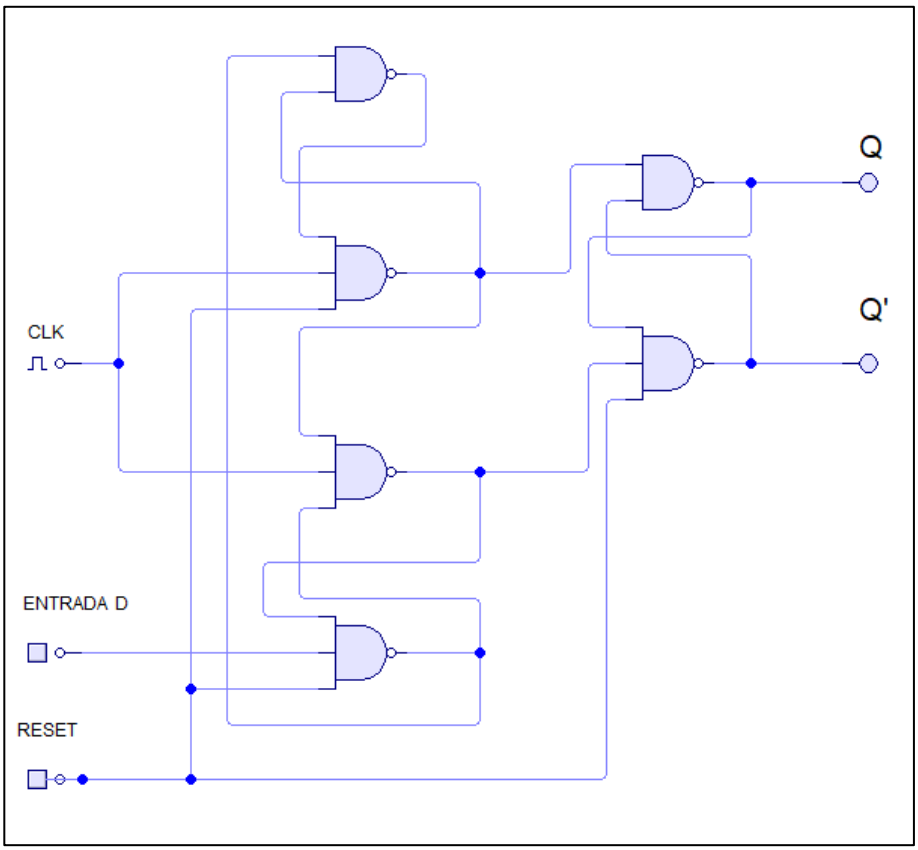
Se utilizan 2 tipos de “disparo”: disparo por flanco positivo y disparo por flanco negativo. El disparo por flanco positivo consiste en un cambio de estado de 0 a 1, en cuanto el disparo por flanco negativo consiste en un cambio de 1 a 0.

Existen varios tipos de flip-flop que se utilizan para diferentes propósitos debido a que cada uno presenta características tanto en el número de entradas que posee, así como en la forma en que se comporta la salida según los valores que se presentan en la entrada, la única característica que no varía es la presencia de 2 pines de salida de los cuales uno es el inverso del valor que presenta el otro y se denotan como Q y Q'.

4.5.1. Flip-flop tipo D

El flip-flop tipo D está formado por un circuito combinacional con arreglos de compuertas NAND de entradas cruzadas.

Figura 11. **Circuito combinacional del flip-flop tipo D**



Fuente: elaboración propia.

La figura 11 muestra el diagrama del circuito combinacional de un flip-flop tipo D con 3 entradas y las 2 salidas características de los flip-flops. Los pines de entrada son: el reloj (CLK) que se utiliza para dar el pulso de disparo ya sea por flanco positivo o flanco negativo, el pin de RESET que devuelve las salidas a un estado base y el pin de entrada D que, como su nombre indica, es donde ingresan los bits de entrada al flip-flop. Los pines de salida Q y Q' son los típicos encontrados en todos los flip-flop cuya característica es que uno es el complemento del otro.

Tabla XVII. **Tabla de verdad del flip-flop tipo D**

Entradas			Salidas	
RESET	CLK	D	Q	Q'
0	X	X	0	1
1	↑	0	0	1
1	↑	1	1	0

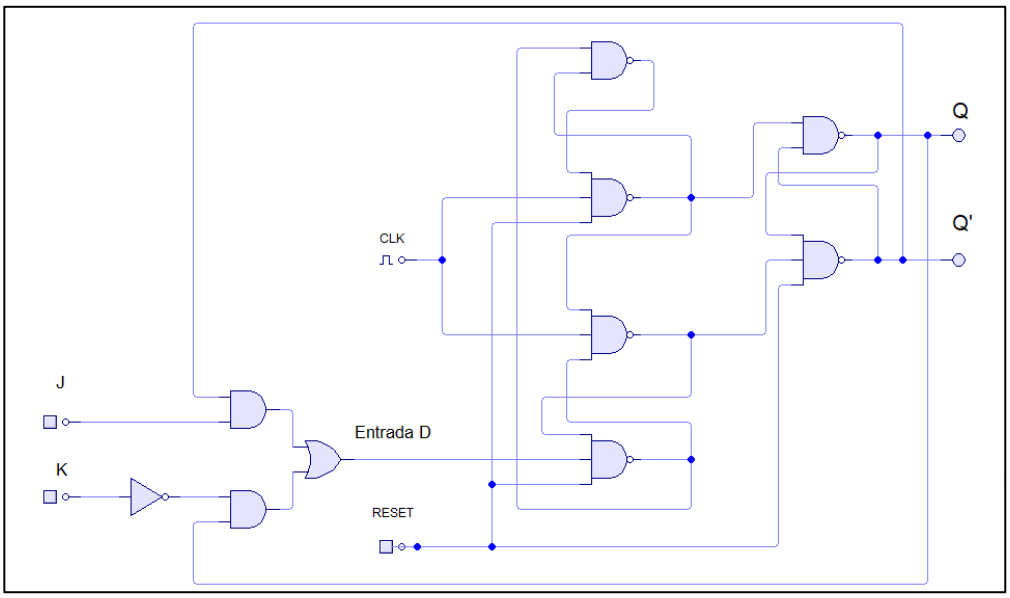
Fuente: elaboración propia.

La tabla XVII muestra el comportamiento de las salidas correspondiente a los cambios en las entradas. Como se puede observar la entrada de RESET se activa cuando se presenta un bit de entrada 0, por lo que devuelve las salidas a su estado base, que en este caso es de Q=0 y Q'=1. Cuando la entrada de RESET se encuentra con un bit de 1, se encuentra inactiva, por lo que al presentarse un cambio de 0 a 1 (disparo) en la entrada CLK se obtiene un cambio en la salida que depende del valor presente en la entrada D. Si D=0 entonces Q=0 y Q'=1, si D=1 entonces Q=1 y Q'=0.

4.5.2. Flip-flop JK

El flip-flop JK es otro de los más utilizados en los diseños digitales debido a las características que presenta la tabla de verdad bajo la influencia de sus 2 entradas J y K.

Figura 12. **Circuito combinacional del flip-flop JK**



Fuente: elaboración propia.

La figura 12 muestra el diagrama del circuito combinacional del flip-flop JK, en el que se puede observar la presencia de un flip-flop tipo D al cual se le acopla un circuito combinacional a la entrada D que utiliza las salidas Q y Q' en conjunto a las entradas J y K para obtener el comportamiento característico de este flip-flop.

El flip-flop JK a diferencia del flip-flop tipo D presenta 4 entradas y 2 salidas. Las entradas que se mantienen igual que en flip-flop tipo D son CLK y RESET. Por otro lado, las líneas J y K son las entradas de datos que pueden tomar valores de 1 o 0 lógicos. Las salidas se comportan de la misma manera que en un flip-flop tipo D, una salida es la inversa de la otra, por esta razón presentan la misma notación Q y Q'.

Tabla XVIII. **Tabla de verdad del flip-flop JK**

Entradas		Salidas
J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q'(t+1)

Fuente: elaboración propia.

La tabla XVIII muestra el comportamiento de la salida del flip-flop JK en base a los cambios que se presentan en las entradas J y K. Cuando ambas entradas son 0, la salida se mantiene sin cambios (Q(t)). Por otra parte, cuando una entrada tiene un valor de 1 y la otra un valor de 0, la salida puede ser 1 o 0 dependiendo el caso. Por último, cuando ambas entradas tienen un valor de 1, la salida es el estado negado de la salida presente antes de presentarse el pulso de disparo en CLK.

4.6. Circuitos secuenciales

Los circuitos secuenciales son circuitos combinacionales a los que se les añade unidades de memoria con el objetivo de guardar información la cual afecta el estado del circuito en un momento específico. Las entradas externas del circuito afectan los datos almacenados en la memoria lo que causa un cambio en la salida.

Los circuitos secuenciales se dividen en dos tipos: circuitos secuenciales sincrónicos y circuitos secuenciales asincrónicos. Los circuitos secuenciales asincrónicos utilizan dispositivos de retardo de tiempo y su comportamiento depende de las señales de entrada, tanto en el orden en que cambian como en

el instante en que ocurre el cambio. Por otro lado, en los circuitos secuenciales sincrónicos se maneja información de entrada en instantes de tiempo discretos. Los cambios en la salida, así como el ingreso y proceso de los datos se lleva a cabo en armonía con los pulsos de reloj. Los elementos de memoria que utilizan los pulsos de reloj se denominan circuitos secuenciales con reloj y son los ya expuestos anteriormente flip-flops. Debido a que los circuitos secuenciales sincrónicos son los que presentan mejor estabilidad, en este trabajo de investigación sólo se abordan este tipo de circuitos secuenciales.

4.6.1. Ejemplos de circuitos secuenciales sincrónicos

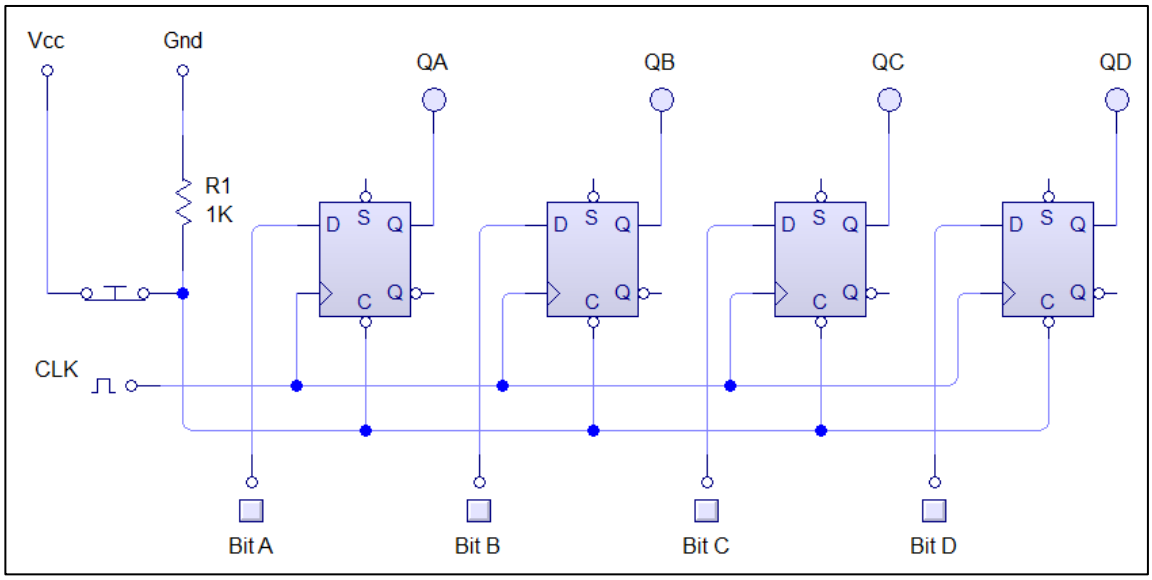
Dos de las aplicaciones más comunes de los flip-flop también son ejemplos de los circuitos secuenciales sincrónicos. Por un lado, se tienen los registros, que son las unidades de memoria y también los contadores, utilizados para llevar conteos configurables por medio del diseño digital.

4.6.1.1. Registros

Los flip-flop pueden mantener información de forma indefinida (siempre que se mantenga conectados a voltaje) por lo que se utilizan para crear memorias temporales para almacenar datos. Se sabe, que los datos almacenados electrónicamente no son más que bits, por lo que cada bit puede asignarse a un flip-flop para quedar almacenado para utilizarse en un proceso posterior, por lo que, si se desea guardar una serie de bits, se utiliza un arreglo de flip-flops para almacenar los bits de forma ordenada y es a este arreglo que se le conoce como registro.

Un registro puede ser tan grande como el número de bits que se desea almacenar en él. Debido a que el registro está formado por flip-flips y estos a su vez por circuitos secuenciales, la adición de circuitos secuenciales externos al flip-flop no cambian la naturaleza de este circuito y solo afecta en la forma que se maneja la información ya sea en el proceso de ingreso o salida del registro.

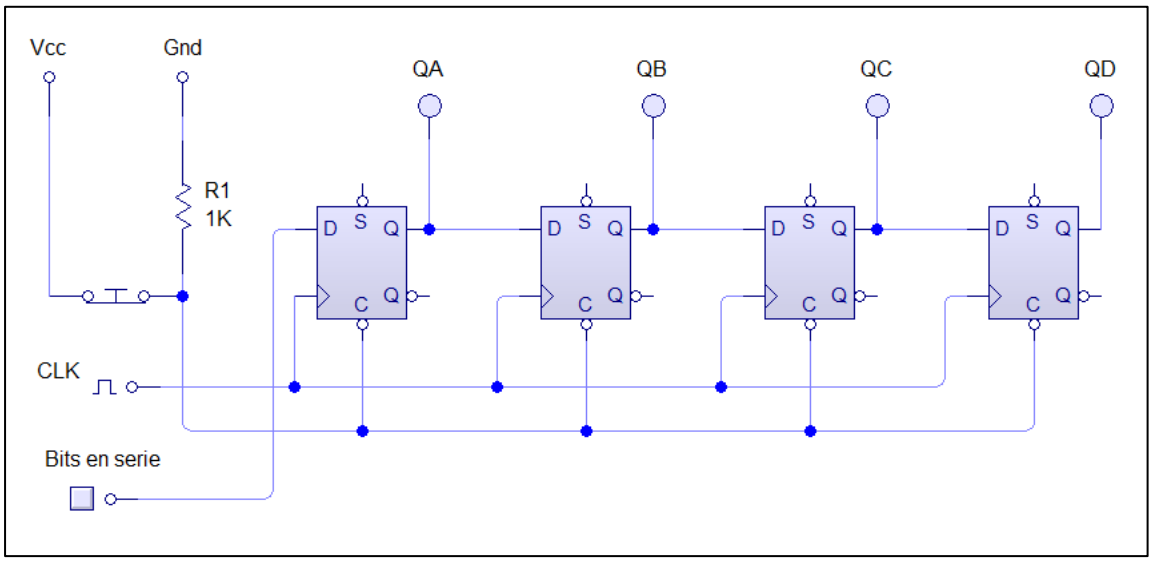
Figura 13. **Registro con ingreso en paralelo**



Fuente: elaboración propia.

La figura 13 muestra un registro de 4 bits en el cual los bits de datos ingresan en el mismo ciclo de reloj (en paralelo). Para conseguir este comportamiento, se conectan a una misma línea de disparo las 4 entradas de CLK. El botón de reset también se conecta a una sola línea que interconecta las 4 entradas de reset de los flip-flop. Cuando ingresa un bit de 0 a la línea de reset, todos los flip-flop regresan a su estado base en la salida.

Figura 14. **Registro de 4 bits con ingreso serial unilateral**



Fuente: elaboración propia.

La figura 14 es un registro de 4 bits que difiere de la figura 13 en el método de ingreso de los bits al registro. En este ejemplo, los bits entran uno seguido del otro, cada uno en un ciclo de disparo diferente por lo que toma 4 ciclos de disparo ingresar los 4 bits a su flip-flop correspondiente. Este circuito es de ingreso serial unilateral porque el corrimiento de los bits se hace a la derecha, por otro lado, si el circuito tuviese la opción de elegir el corrimiento hacia la izquierda o derecha se conoce como circuito de corrimiento bidireccional. La interconexión entre los flip-flop se mantiene igual que en la figura 13 a excepción de la conexión entrada-salida, la cual, asegura un corrimiento de bits de un flip-flop a otro con cada ciclo de disparo, logrando de esta manera un corrimiento ordenado de bits.

4.6.1.2. Contadores

Cuando la conexión de los flip-flops les permite pasar por una sucesión prescrita de estados al aplicar pulsos de disparo en la entrada CLK, se obtiene un contador. Generalmente el flip-flop cuenta con una conexión externa hacia un circuito combinacional que le permite obtener el comportamiento para seguir un orden de conteo establecido. Los pulsos de disparo (como en los registros) se obtienen de una fuente externa, ya sea por un generador de pulsos, por medio de botones o teclas que el operador controla, por la finalización de un proceso en un circuito previo al contador o registro, etc.

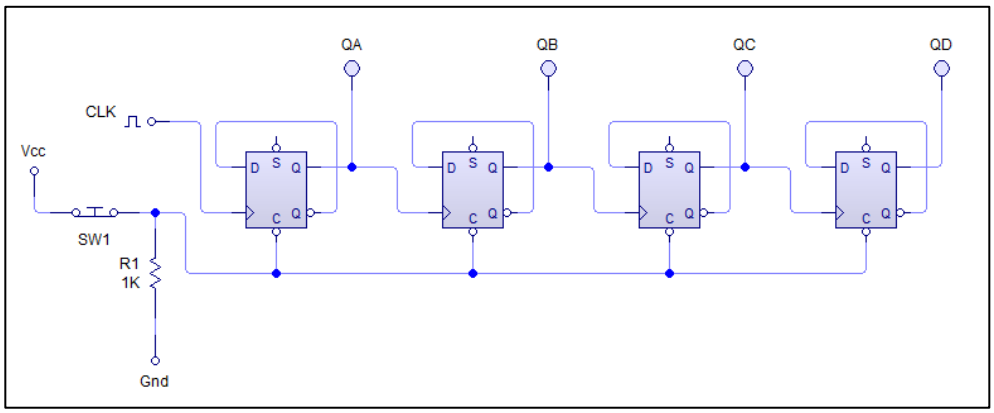
Un contador puede tener N número de bits que es proporcional al número de flip-flops con el que éste cuenta. El alcance del conteo que puede alcanzar dicho contador esta dado por:

$$Númeor\ máximo\ de\ conteo = \ 2^N - 1 \ \ Ecuación\ (6)$$

Donde:

N= número de bits o flip-flops.

Figura 15. **Contador binario de rizo**



Fuente: elaboración propia.

La figura 15 muestra la conexión entre 4 flip-flop tipo D para construir un contador binario de 4 bits. Como se puede observar, las entradas D están conectadas a la salida Q' del mismo flip-flop, mientras que la salida Q se conecta a la entrada del pulso de disparo del flip-flop siguiente. Al igual que en un registro, se agrega un botón de reset cuya salida está conectada a cada entrada de reset de los flip-flop que conforman el contador. El conteo se realiza de forma ordenada iniciando en 0, terminando en 15 (binario) y regresando a 0 nuevamente para seguir el conteo.

Si se desea controlar la forma en que se realiza el conteo, se debe diseñar los estados actuales y futuros en una tabla de verdad, sacar las funciones booleanas necesarias e implementarlas en un circuito combinacional conectado a los flip-flop.

Tabla XIX. **Tabla de verdad del flip-flop JK con estados “no importa”**

SALIDA		ENTRADAS	
PRESENTE	FUTURO	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Fuente: elaboración propia.

La tabla XIX muestra el valor de la salida del flip-flip JK en estados presentes y futuros. Para que un valor en el presente pase a un valor futuro esperado debe presentarse en las entradas J y K un valor fijo, el cual se presenta en la tabla de verdad donde, el valor “X” es un estado “no importa” el cual puede tomar un valor de 0 o 1 ya que no afectará el comportamiento de la salida a futuro que se espera en Q.

Al aplicar los mapas de Karnaugh para sacar las funciones booleanas de J y K, entran en juego los valores de las variables “no importa” ya que se puede tomar su valor como 1 o 0 según convenga al momento de agrupar los 1 para reducir significativamente las funciones obtenidas.

Tabla XX. **Tabla de verdad de contador con flip-flop JK**

PRESENTE			FUTURO			ENTRADAS					
A	B	C	A	B	C	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	1	0	0	1	X	X	1	0	X
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	0	0	0	X	1	X	1	0	X

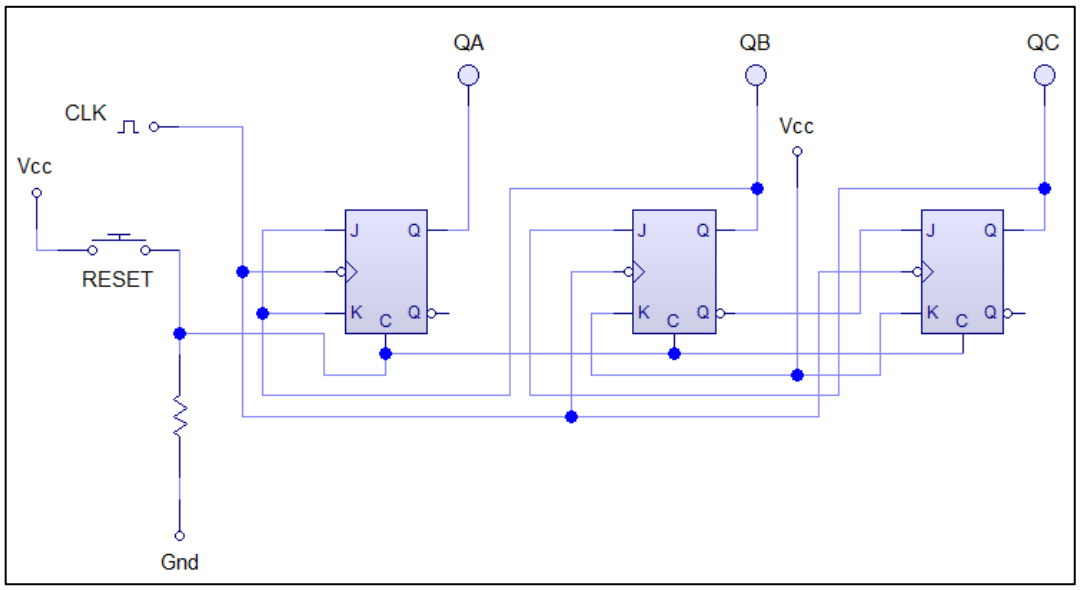
Fuente: elaboración propia.

La tabla XX es la tabla de verdad utilizada para el diseño de un contador binario de 3 bit cuyo conteo es: 000 → 001 → 010 → 100 → 101 → 110 →000, en números decimales el conteo es: 0 → 1 → 2 → 4 → 5 → 6 → 0. Como puede observarse el conteo no es lineal por lo que se debe utilizar un circuito combinacional externo a los contadores para permitir este comportamiento. La tabla XX muestra el uso de los variables “no importa” utilizando la tabla XIX según los valores presentes y futuros deseados para este conteo.

Al aplicar los mapas de Karnaugh para encontrar las funciones de J_A, K_A, J_B, K_B, J_C y K_C se aplica el criterio para darle el valor conveniente a las variables “no importa” obteniendo así las funciones correspondientes.

- $J_A = B$
- $K_A = B$
- $J_B = C$
- $K_B = 1$
- $J_C = B'$
- $K_C = 1$

Figura 16. **Contador de 3 bits con flip-flop JK**



Fuente: elaboración propia.

La figura 16 muestra el diagrama del contador de 3 bits. Se puede observar que en las funciones booleanas no aparecen compuertas lógicas y esto se debe a la aplicación de criterio de la variable “no importa” para lograr la máxima reducción de las funciones booleanas al tomar estas variables como 1 lógico de la manera más conveniente al momento de agrupar bits en los mapas de Karnaugh. Como en los ejemplos anteriores, se presenta el botón de reset, pero en este caso en particular, la entrada de clear de los flip-flop no presenta el círculo de inversión por lo que se necesita un 1 lógico para activar el reset en los flip-flop y por eso el botón se encuentra conectado de manera distinta.

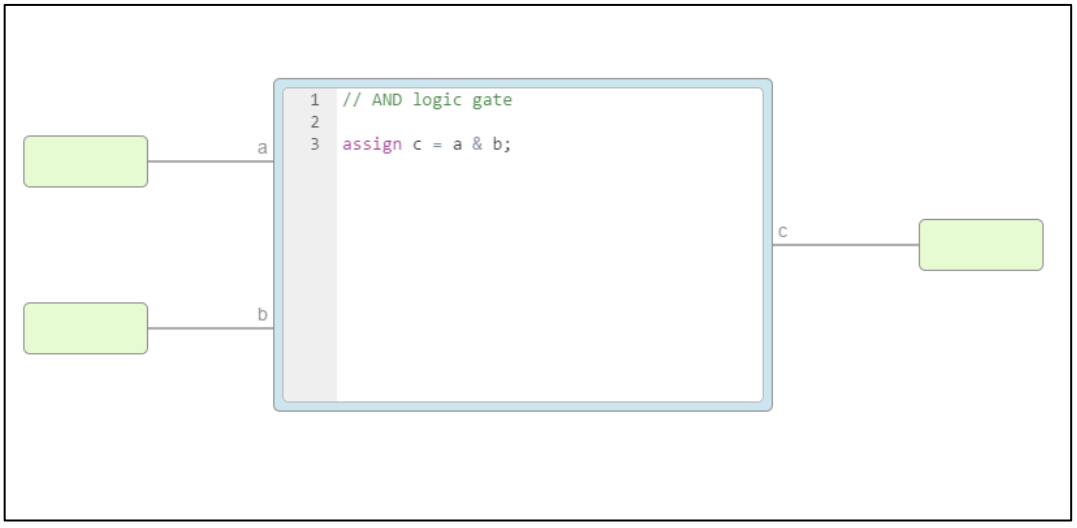
5. ELECTRÓNICA DIGITAL EN FPGA LIBRE ALHAMBRA II

5.1. Bloques en IceStudio

Para fines de este trabajo de investigación se define como bloque a un arreglo de elementos digitales y segmentos de código que están interconectados y se agrupan en una sección del diagrama digital completo.

Dicho de otra forma, un bloque es como una “caja” en la que se coloca un circuito digital complejo para separarlo del resto de un circuito. De esta caja solo se sacan los pines de entrada y salida de todo el circuito interno para poder ser utilizados en cualquier momento.

Figura 17. **Bloque de la compuerta AND**



Fuente: Colección de IceStudio.

La figura 17 es el bloque que forma la compuerta AND que viene por defecto en IceStudio. Como puede observarse, se tiene a la izquierda 2 pines de entrada y a la izquierda un pin de salida. En el centro se encuentra una sección de código que puede ser también un conjunto de compuertas lógicas u otros elementos digitales que realicen determinado proceso.

5.1.1. Crear y Utilizar un bloque

Para crear un bloque se deben seguir los siguientes pasos:

- Crear un archivo nuevo: File → New.
- Crear el circuito que se desea se encuentre dentro del bloque.
- Añadir los pines de entrada y salida de datos
    - Para pines de entrada: Barra de diseño →Basic → Input.
    - Si se desea que el pin esté conectado a un pin físico de la FPGA se deja marcada la opción “FPGA pin”, en caso contrario se desmarca y solo se le coloca el nombre deseado al pin de entrada.
    - Luego de colocar el pin de entrada se conecta a la sección del circuito que se desee. Si se necesita asociar a un pin físico de la FPGA, basta con seleccionar de la lista desplegable el pin físico deseado.
    - Para pines de salida: Barra de diseño → Basic → Output

- De la misma manera que con los pines de entrada, los pines de salida también pueden o no asociarse a un pin físico de la FPGA, marcando o desmarcando la opción “FPGA pin”.

- Añadir información necesaria al bloque.

  - Barra de tareas → Edit → Project information

  - En la ventana emergente se puede agregar información que describe al bloque: Nombre, versión, una descripción de qué función realiza el bloque, Autor y una imagen .SVG para identificar el bloque.

- Como último paso se guarda el proyecto como archivo *.ice.

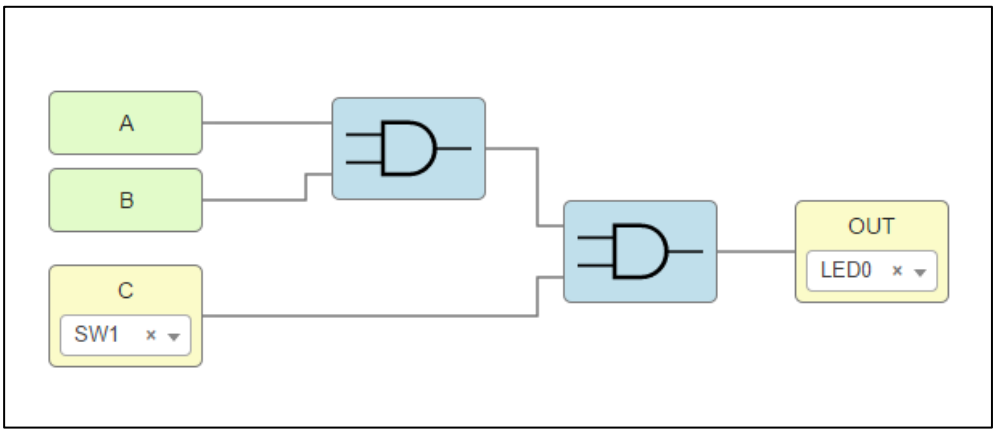
Una vez creado el bloque, se puede utilizar en otro proyecto siguiendo los pasos siguientes:

- Crear un nuevo proyecto: Barra de tareas → File → New.

- Abrir el archivo *.ice que se desea utilizar como bloque: Barra de tareas → File → Add as block.

- Colocar el bloque en el área de trabajo y hacer las conexiones que se desean para el bloque.

Figura 18. **Bloque para AND de 3 entradas y 1 salida**



Fuente: elaboración propia utilizando IceStudio.

La figura 18 resume los pasos expuestos para la creación de un bloque que es la conexión de una AND de 3 entradas y 1 salida. Del lado izquierdo se observan las 3 entradas de datos, dos de ellas (A y B) son entradas libres para conexión y la otra esta asignada a un switch físico de la FPGA. Del lado derecho se encuentra la salida que también esta asignada a un pin de la FPGA, en este caso, un led físico.

5.2. Compuertas lógicas

IceStudio brinda un catálogo con 7 compuertas básicas que pueden utilizarse en los diseños digitales para la FPGA libre Alhambra II. La figura 7 muestra la barra de diseño que contiene todas las herramientas disponibles, las compuertas lógicas se encuentran en: barra de diseño → logic → gate. Para utilizar las compuertas basta con seleccionar el elemento a utilizar y posicionarlo en el área de trabajo donde se desea.

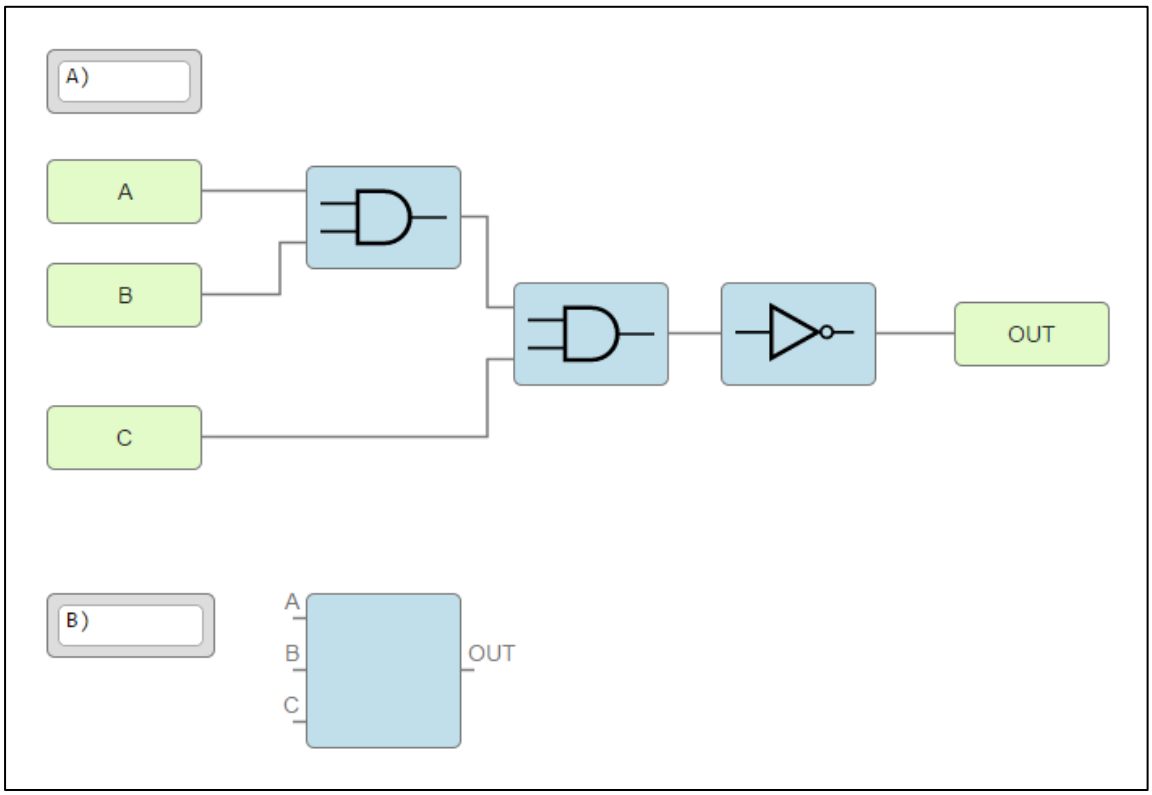
Si se hace doble click sobre la compuerta, IceStudio hace un acercamiento mostrando el contenido del bloque donde puede observarse los elementos y conexiones existentes.

5.2.1. Compuertas mixtas utilizando bloques

El catálogo de componentes digitales disponibles en IceStudio no son más que bloques que cumplen una función definida, por lo que, si se necesita una compuerta mixta (NAND, NOR, etc), pero con características que no están en la colección presente de IceStudio, la solución más simple es crear un bloque que desempeñe la función necesaria utilizando los bloques o colecciones ya disponibles en IceStudio.

En el caso de la compuerta NAND, IceStudio tiene un bloque de 2 entradas, una sección de código Verilog y una salida que realiza la función lógica de una compuerta NAND. Si como usuario se necesita una compuerta NAND con n entradas y 1 salida, queda en el usuario el descargar un bloque creado por alguien más que cumpla con los requisitos o crear el bloque por sí mismo utilizando las herramientas disponibles en IceStudio.

Figura 19. **Compuerta NAND de 3 entradas y 1 salida utilizando bloques**



Fuente: elaboración propia utilizando IceStudio.

La figura 19.A es una compuerta NAND de 3 entradas y 1 salida creada con 3 bloques lógicos: 2 compuertas AND y una compuerta NOT. Posee 3 entradas y una salida las cuales no están vinculadas a ningún pin físico de la FPGA.

La figura 19.B muestra el esquema de la figura 19.A cargado en IceStudio como un Bloque. Se puede observar que el bloque no posee un ícono que lo identifique como es el caso de las compuertas AND y NOT de IceStudio (Lo cual puede cambiarse como se expuso en la sección 5.1.1), así también, del lado izquierdo aparecen las entradas nombradas “A”, “B” y “C” y la salida “OUT”.

Al hacer doble click sobre el bloque de la figura 19.B se puede observar la figura 19.A, mostrando así la característica de la agrupación de circuitos por medio de bloques. Se demuestra de esta manera lo simple que es crear bloques con diferentes tipos de compuertas lógicas que cumplan con los requerimientos del diseñador de circuitos digitales.

5.2.2. Ejemplo de aplicación

Puede utilizarse el método de mapas de Karnaugh (expuesto en el capítulo anterior) o simple inspección para diseñar el circuito combinacional requerido.

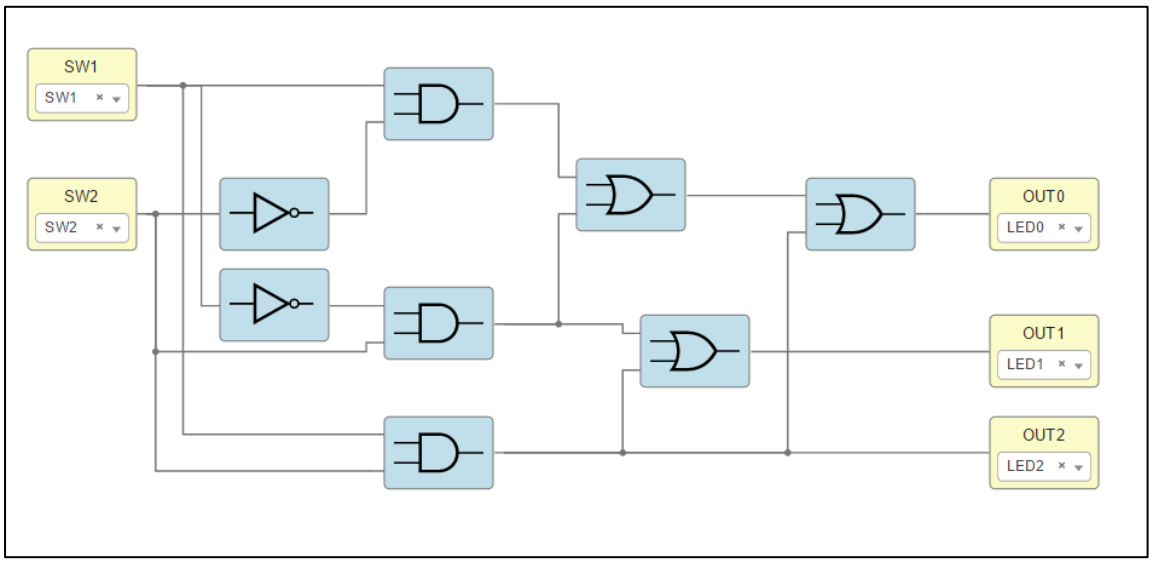
Tabla XXI. **Tabla de verdad ejemplo de aplicación de compuertas lógicas**

ENTRADAS		SALIDAS		
SW2	SW1	LED0	LED1	LED2
0	0	0	0	0
0	1	1	0	0
1	0	1	1	0
1	1	1	1	1

Fuente: elaboración propia.

La tabla XXI muestra el comportamiento esperado de 3 led de la FPGA libre cuando se presionan los switches de entrada. Para lograr que los leds se comporten de la manera descrita en esta tabla, se debe diseñar un circuito combinacional de 2 entradas y 3 salidas.

Figura 20. **Circuito combinacional para los leds de la FPGA**



Fuente: elaboración propia utilizando IceStudio.

La figura 20 muestra el circuito combinacional diseñado en IceStudio para los requerimientos de la tabla XXI. A la izquierda de la figura 20 se observan las 2 entradas digitales asociadas a los switches físicos y por la izquierda las 3 salidas digitales asociadas a 3 led físicos de la FPGA.

Para probar el correcto funcionamiento del diseño se debe cargar el diseño a la FPGA:

- Barra de tareas → Tools → Verify (Ctrl+R).
- Barra de tareas → Tools → Build (Ctrl+B).
- Barra de tareas → Tools → Upload (Ctrl+U).

IceStudio muestra un mensaje cuando el proyecto se ha cargado exitosamente a la FPGA. Se puede probar el funcionamiento del diseño presionando los switches de la FPGA y comprobando que el comportamiento de los leds coincida con los valores esperados según la tabla XXI.

5.3. Circuitos combinacionales

Los circuitos combinacionales son gobernados por las tablas de verdad, que indican los cambios en valores de salida según los cambios en las variables de entrada. Las tablas de verdad son un mapa que detalla el funcionamiento del circuito combinacional y en IceStudio también se aplican para el desarrollo de circuitos digitales.

5.3.1. Tablas de verdad en IceStudio

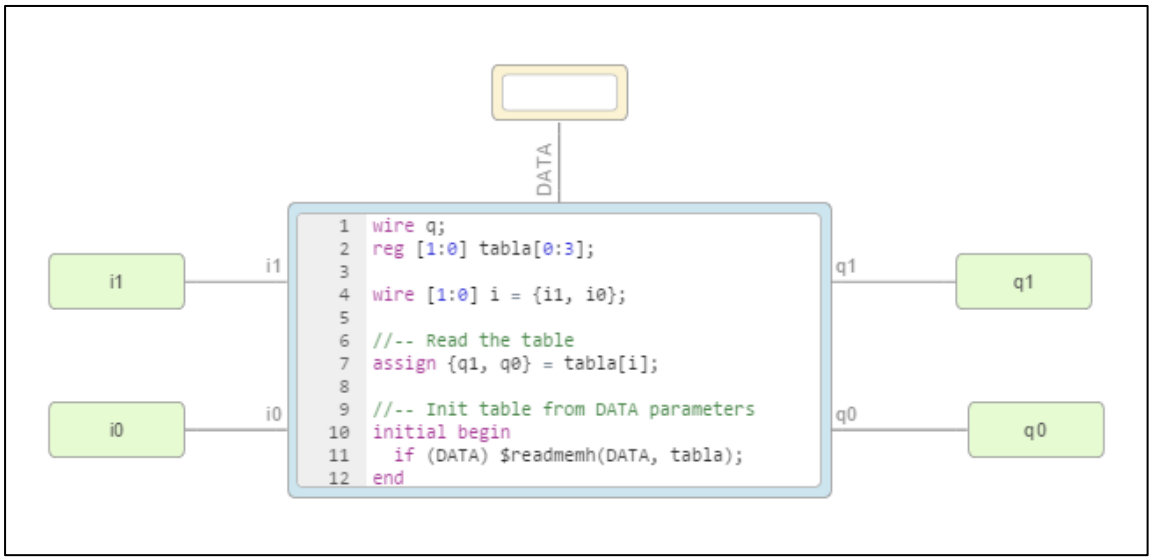
Para utilizar tablas de verdad en IceStudio se tienen dos opciones:

- La primera opción es agregar la colección de circuitos combinacionales disponible desde la página de tutoriales de la FPGA Alhambra II.
    - Descargar la colección desde: https://github.com/Obijuan/digital-electronics-with-open-FPGAs-tutorial/wiki/Video-5:-Colecciones-en-Icestudio#instalaci%C3%B3n.
    - Agregar la colección descargada: Barra de tareas → Tools –> Collections → Add.
    - Se selecciona el archivo *.zip descargado y darle un nombre a la colección (Combinacional1, por ejemplo).

- Seleccionar la colección para poder utilizarla: Barra de tareas → Select → Collection → Combinacional1.

- Seleccionar en la barra de diseño el bloque de tabla a utilizar: Barra de diseño → Comb → Tablas.

Figura 21. **Bloque para manejo de tablas de verdad creado con Verilog**



Fuente: Colección de IceStudio.

- La segunda opción es crear un bloque específico para utilizar tablas. La figura 21 muestra el interior del bloque tabla 2-2 que viene en el paquete de colecciones de la página de tutoriales de la FPGA libre. Para crear un bloque para manejo de tablas que se adecue a las necesidades del diseño a implementar se siguen los siguientes pasos:

    - Crear un archivo nuevo: barra de tareas → File → New.

- Agregar un bloque de código: Barra de diseño → Basic → Code.

- Se abre un cuadro de diálogo donde se ingresan los nombres de las entradas, salidas y parámetros a utilizar en el bloque de código que son los que se comunican a los bloques I/O.

- Agregar los puertos de entrada y salida del bloque: Barra de diseño → Basic → Input/Output.

- Agregar un bloque de “constantes”. Este bloque es el encargado de leer los datos de la tabla: Barra de diseño → Basic → Constant.

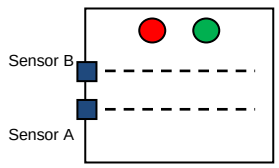
- Se procede a describir el hardware por medio del lenguaje verilog.

5.3.2. Ejemplo de aplicación

Se solicita diseñar un circuito combinacional que permita detectar si el niño(a) puede entrar a un juego mecánico dependiendo si cumple o no la altura indicada.

Tabla XXII. **Control de acceso a un juego mecánico**

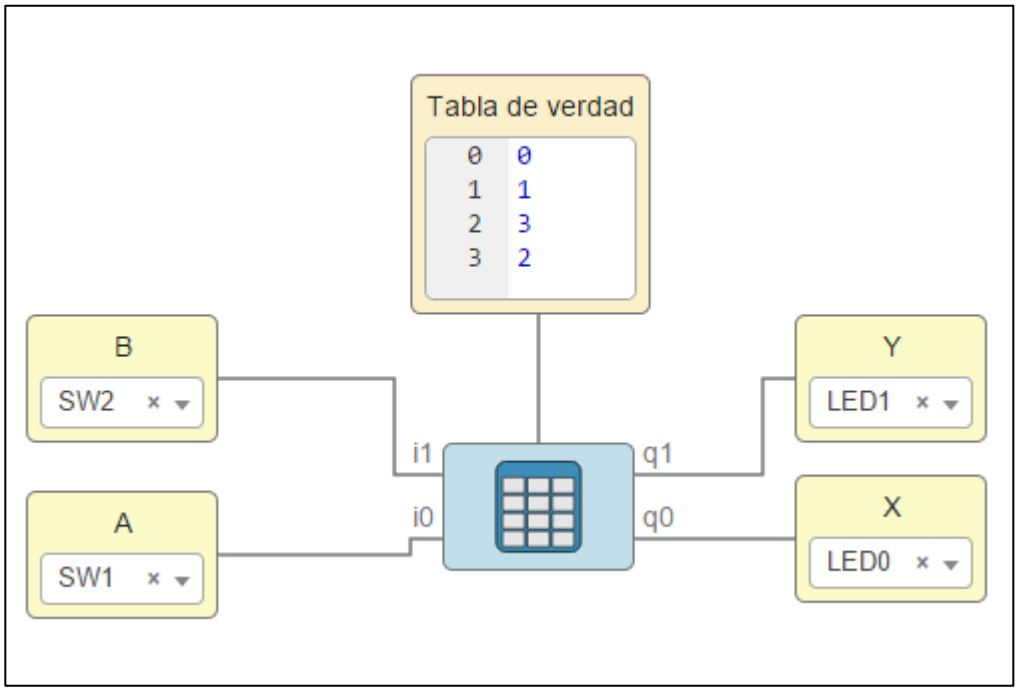
ENTRADAS		SALIDAS	
B	A	X	Y
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1



Fuente: elaboración propia.

La tabla XXII presenta los valores de la tabla de verdad con los posibles valores obtenidos por los sensores “A” y “B”. Las salidas “X” y “Y” se utilizan para encender/apagar los leds indicadores: X=led rojo y Y=led verde. El led verde indica que se cumple con la altura mínima requerida y que el niño(a) puede ingresar al juego mecánico. El led rojo indica que no se cumple con la altura mínima requerida por lo que no se puede acceder al juego. Cuando solo se activa el sensor B, ambos leds se encienden para indicar que algún sensor no funciona o que el sensor B está siendo obstaculizado por algo. Si ningún sensor se activa, ambos leds permanecen apagados.

Figura 22. **Circuito de control de acceso**



Fuente: elaboración propia utilizando IceStudio.

La figura 22 muestra el circuito de control de acceso aplicando la tabla de verdad y el bloque de circuito combinacional de IceStudio. Los bloques de

entrada “A” y “B” están asociados a los switches físicos de la FPGA y de igual forma, los bloques de salida “X” y “Y” están asociados a los leds físicos de la FPGA (led0 y led1).

El bloque combinacional utilizado es “tabla 2-2” de la colección de tutoriales disponibles en la página de la FPGA libre Alhambra II. La entrada de data se conecta al bloque memory disponible en: barra de diseño → Basic → Memory. Este bloque es el encargado de contener los valores de la tabla de verdad. El bloque memory cuenta con 2 columnas que manejan valores en decimal, pero los interpreta en binario por lo cual el ingreso de los valores de la tabla XXII se presenta en decimal en la figura 22.

Una vez realizada la conexión de los bloques como aparece en la figura 22 y se ingresan los datos a la memoria, se procede a cargar el diseño a la FPGA: Barra de tareas → Tools → Verify → Build → Upload.

5.4. Multiplexor y demultiplexor

El multiplexor y el demultiplexor son elementos digitales muy útiles al momento de diseñar un circuito digital. Existen muchas maneras de utilizar estos componentes en IceStudio e implementarlos en la FPGA.

Una forma sencilla para utilizar los multiplexores (y una de las ventajas de la FPGA libre) es descargar la colección que contiene los multiplexores desde la página de tutoriales de la FPGA libre Alhambra II.

- Descargar el archivo *.zip que contiene la colección de multiplexores desde: https://github.com/Obijuan/digital-electronics-with-open-FPGAs-tutorial/raw/master/wiki/Tutorial-14/Collection/Academia-Jedi-HW-14.zip.

- Sin descomprimir el archivo, se agrega la colección: Barra de tareas → Tools → Collections → Add.

- IceStudio abre un cuadro de diálogo en el cual se debe elegir el archivo con *.zip descargado anteriormente y colocarle el nombre deseado a esta colección nueva (Multiplexores, por ejemplo).

- Para poder utilizar esta colección: barra de tareas → Select → Collection → Multiplexores.

- Como último paso basta con seleccionar el tipo de multiplexor que se desea utilizar desde: Barra de diseño → Varios → Mux.

Como es de esperarse, no siempre los bloques disponibles en las colecciones de IceStudio podrán satisfacer las necesidades de diseño, por lo cual también es posible crear un bloque para un multiplexor que satisfaga las necesidades del diseñador digital.

5.4.1. Creación de multiplexor y demultiplexor utilizando tablas de verdad.

Debido a que el multiplexor y el demultiplexor son circuitos combinacionales, es posible crear un bloque utilizando los bloques de tabla y memoria de IceStudio para diseñar un multiplexor o demultiplexor que satisfaga las necesidades de un diseño digital en particular.

Tabla XXIII. **Tabla de verdad para MUX 2:1**

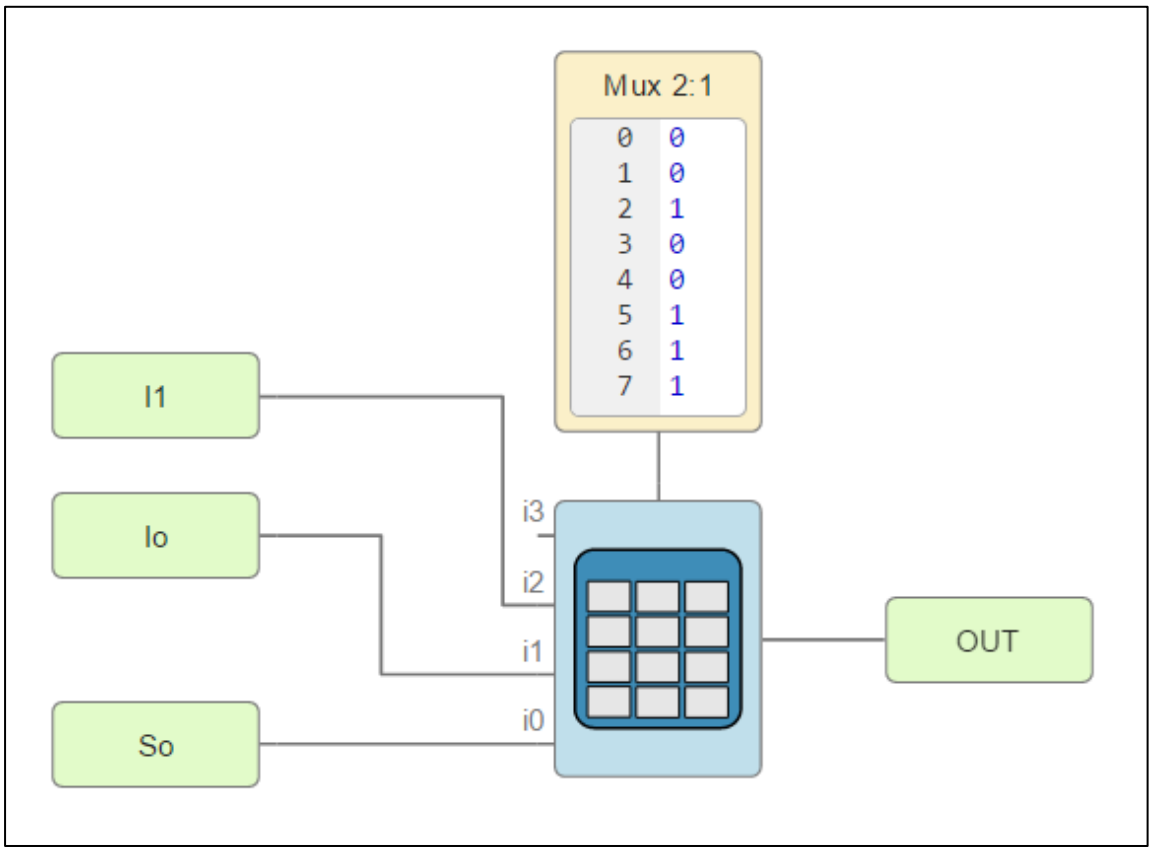
ENTRADAS			SALIDA	
I_1	I_0	S_0	OUT	
0	0	0	0	I0
0	0	1	0	I1
0	1	0	1	I0
0	1	1	0	I1
1	0	0	0	I0
1	0	1	1	I1
1	1	0	1	I0
1	1	1	1	I1

Fuente: elaboración propia.

Para crear un multiplexor de 2:1, se necesita 1 variable de control que permita seleccionar la entrada que se desea conectar con la salida. La tabla XXIII corresponde a la tabla de verdad de un multiplexor 2:1. Las entradas de datos o canales son: I_1 e I_0, las cuales pueden tener 2 valores posibles: 0 o 1 lógico. La entrada de selección "S_0" que también puede tomar solo dos valores (0 o 1 lógico) permite la selección del canal conectado a la salida.

Cuando S_0=0, independiente del valor de I_1, en la salida se obtiene el valor de I_0 en ese momento. Cuando S_0=1, independiente del valor de I_0, en la salida se obtiene el valor de I_1 en ese momento.

Figura 23. **Implementación de multiplexor utilizando tablas**



Fuente: Elaboración propia utilizando IceStudio.

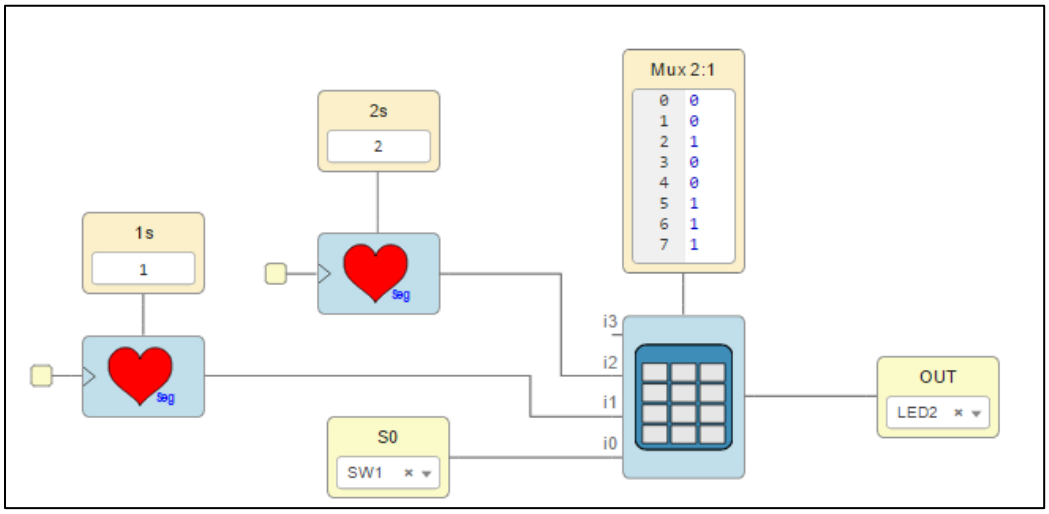
La figura 23 muestra el circuito interno de un bloque de multiplexor 2:1 implementado utilizando el bloque de “Tabla” y “Memory” de IceStudio. El bloque de Memory contiene los valores de entrada y salida (valores en decimal) de la tabla XXIII, los cuales viajan al bloque “Tabla 4-1” por medio de la entrada “DATA”. Los bloques de entrada I_1 e I_0 están conectados al bloque “Tabla 4-1” por medio de las entradas i_1 e i_2 respectivamente. La entrada de selección S_0 está conectada a la entrada i0 ya que es el menos significativo según la tabla XXIII.

El diseño de la figura 23 se puede guardar y utilizarse como un bloque de Mux 2:1 agregándolo a otro proyecto como un bloque.

5.4.2. Ejemplo de aplicación.

Se necesita que un led parpadee a 2 velocidades diferentes dependiendo de si un botón se mantiene o no presionado. Cuando el botón esta presionado (sw=1) el led debe parpadear cada 2 segundos y si el botón no está presionado (sw=0) el led debe parpadear cada segundo.

Figura 24. **Circuito para el parpadeo de Led**



Fuente: elaboración propia utilizando IceStudio.

La figura 24 muestra el circuito para el parpadeo del led2 de la FPGA. Se puede observar que en el diseño de este circuito se utiliza el bloque de la figura 23 sustituyendo los bloques de entrada I_0 e I_1 por los bloques de generación de pulsos y el bloque de salida por un bloque asociado al led físico de la FPGA.

Para generar la frecuencia de 2s y 1s, se utilizan los bloques de generación de pulsos: Barra de diseño → Varios → Bombeo → Corazón_seg. Cada bloque de generación de pulso se conecta a un bloque con una constante, la cual, indica la frecuencia. Los bloques para la constante se encuentran en: Barra de diseño → Basic → Constant.

La variable de selección S_0 es el botón “SW1” de la FPGA. Para esto se necesita colocar un bloque de entrada asociado al botón SW1 de la FPGA: Barra de diseño → Basic → Input (FPGA pin).

El led2 de la FPGA será utilizado como salida del circuito: Barra de diseño → Basic → Input (FPGA pin) → se selecciona el led2.

Se realizan las conexiones entre bloques como aparece en la figura 23 y se procede a cargar el diseño a la FPGA: Barra de tareas → Tools → Verify → Build → Upload.

5.5. Buses

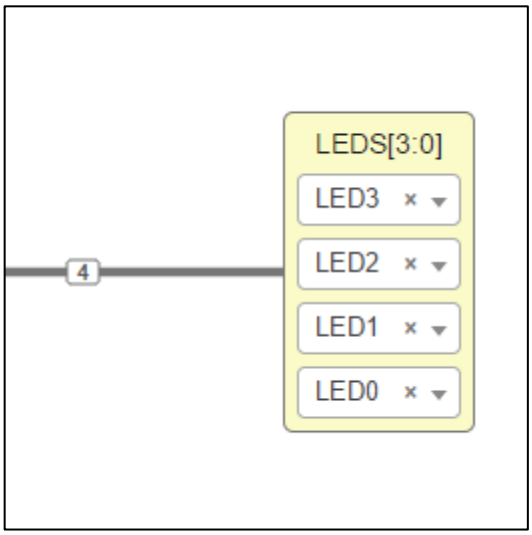
El empleo de buses en IceStudio ayuda a disminuir el número de elementos en el diseño y con esto, a tener una mejor estética y orden en el área de trabajo. De forma sencilla un bus en IceStudio es un conjunto de cables que están dentro de una sola línea.

Para utiliza un bus en IceStudio, solo debe indicarse como sigue: K[1:0]. Donde K puede ser cualquier nombre para identificar el bus y dentro de los corchetes le sigue el número del cable que transporta el bit más significativo y el número del cable que transporta el bit menos significativo.

Como ejemplo, para utilizar un bus de datos de 4 bits de nombre “CNX”, se le indica a IceStudio de la siguiente manera: CNX[3:0]. “CNX” es el nombre del bus que contiene 4 “cables” o líneas de datos. El número 3 indica que el cable número 3 transporta el bit más significativo y el cable número 0 transporta el bit menos significativo.

Para utilizar un bus en un bloque, basta con indicarlo ya sea en los puertos de entrada o salida. Por ejemplo, si se quieren utilizar 4 led de la FPGA, en vez de poner los bloques de forma individual y conectarlos por separado, se puede colocar un solo bloque de salida: Barra de diseño → Basic → Output. En la ventana emergente se marca la opción “FPGA pin” y en el nombre del bloque se coloca LEDS[3:0].

Figura 25. **Bloque de salida con un bus de 4 cables**



Fuente: elaboración propia utilizando IceStudio.

La figura 25 muestra el bloque de salida colocado en el área de trabajo. Como se puede observar, se tienen 4 listas para seleccionar el pin de la FPGA que se utilizará de salida (en este caso, los primeros 4 leds disponibles). Del lado izquierdo del bloque se encuentra el bus, en el cual, el número 4 indica que es un bus de 4 cables.

Se encuentran disponibles otros bloques para el manejo de buses, los cuales se encuentran en la colección del tutorial más reciente en la página de tutoriales y se agrega de la misma manera que con las colecciones de las secciones anteriores.

5.6. Comparador

El comparador de magnitudes es un circuito combinacional muy utilizado en la electrónica digital. Para utilizar el comparador de magnitudes en IceStudio para sintetizarlo en la FPGA se tienen varias opciones disponibles, desde diseñar uno propio que se adapte a las necesidades del diseño, hasta utilizar los comparadores creados por otros usuarios en la red.

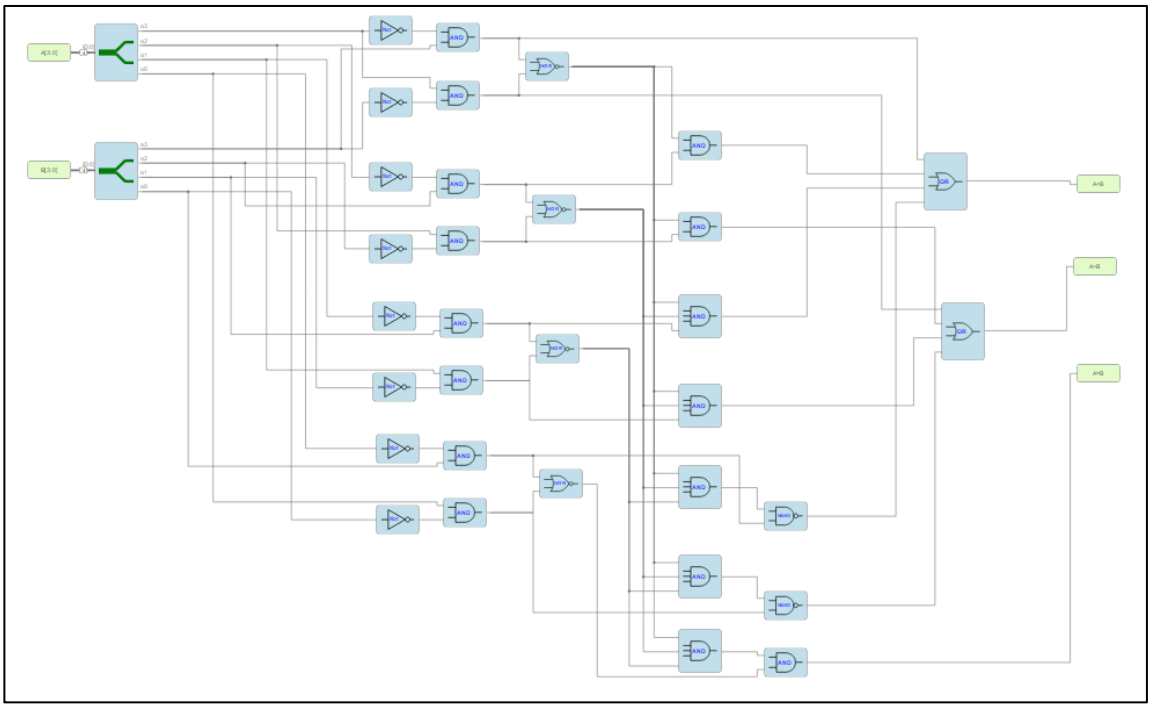
5.6.1. Comparador utilizando bloques de compuertas lógicas

Como se ha expuesto anteriormente, se puede diseñar un comparador de magnitudes de n bits utilizando compuertas lógicas utilizando los algoritmos expresados en las ecuaciones 2, 3, 4 y 5. Al aplicar los algoritmos para diseñar un comparador para dos números de 4 bits, se obtiene el circuito de la figura 10.

El circuito de la figura 10 se puede diseñar en IceStudio utilizando los bloques de compuertas lógicas disponibles. Al diseñar el circuito en IceStudio, se puede sintetizar y utilizar como un bloque de 8 entradas y 3 salidas para su uso

posterior en cualquier otro proyecto que se desee trabajar. Para este objetivo, las entradas y salidas del circuito deben ser bloques de entrada y salida desvinculados de los pines de la FPGA (deshabilitar la opción “FPGA pin” al colocar los bloques).

Figura 26. **Comparador de magnitudes con bloques de compuertas lógicas**



Fuente: elaboración propia utilizando IceStudio.

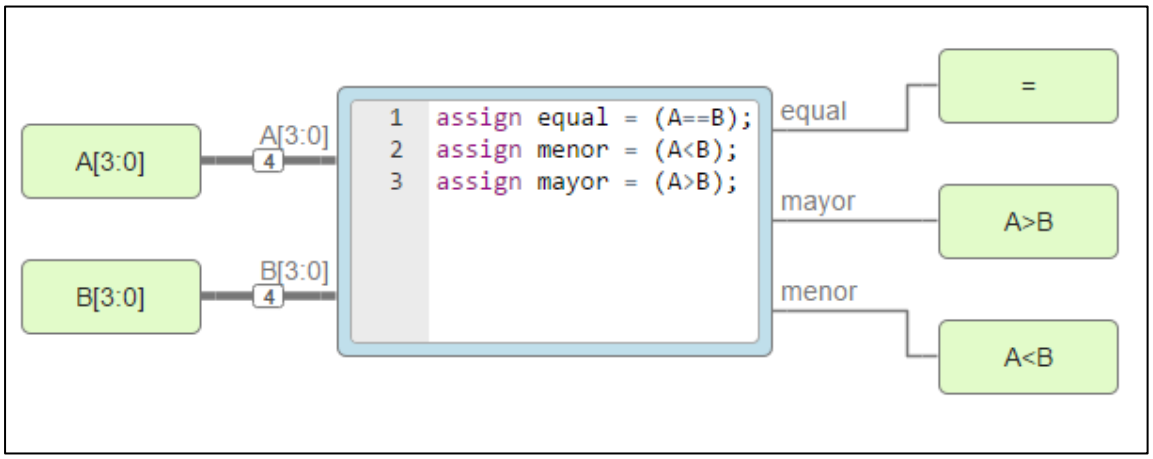
La figura 26 muestra el circuito del comparador de magnitudes de la figura 10 elaborado en IceStudio. Del lado izquierdo de la figura, se pueden observar 2 bloques de entrada (FPGA pin deshabilitado) de 4 cables cada uno que son las entradas para el número A y B a comparar. Seguido a los bloques de entrada, se encuentra un separador de bus, encargado de pasar de un bus a 4 cables

separados para poder manipular bit por bit los números ingresados. Al centro se encuentra el circuito combinacional formado por los bloques de compuertas lógicas de IceStudio seguidos por la parte final del circuito que son los pines de salida.

5.6.2. Comparador utilizando un bloque de código

IceStudio brinda la opción de combinar bloques de código en Verilog para desarrollar parte del hadware de una forma más sencilla. El bloque de código se encuentra en: Barra de diseño → Basic → Code. Al momento de colocarlo, abre un cuadro de diálogo en el que se solicita ingresar 3 datos: Ingresar el nombre de las entradas, el nombre de las salidas y el nombre de las líneas para los parámetros.

Figura 27. **Comparador con bloque de código**



Fuente: elaboración propia utilizando IceStudio.

La figura 27 presenta un comparador de magnitudes diseñado con un bloque de código. El circuito está diseñado para funcionar como un bloque y

poder utilizarlo en otro proyecto junto a otros bloques, es por esa razón que los puertos de entradas y salida no están asociados a un pin físico de la FPGA.

Para elaborar el diseño de la figura 27, primero se procede a colocar el bloque de código: Barra de tareas → Basic → Code. En la ventana emergente se ingresan los campos solicitados:

- Enter the Input ports: A[3:0], B[3:0].
- Enter the Output ports: equal, mayor, menor.
- Enter the parameters: esta sección se deja en blanco ya que no se utilizan parámetros.

Al presionar "OK" aparece el cuadro azul con los puertos de entrada y salida nombrados tal y como se ingresan en la ventana emergente. Es en este bloque donde se ingresa la descripción de hardware para el comparador y es el corazón de todo el bloque.

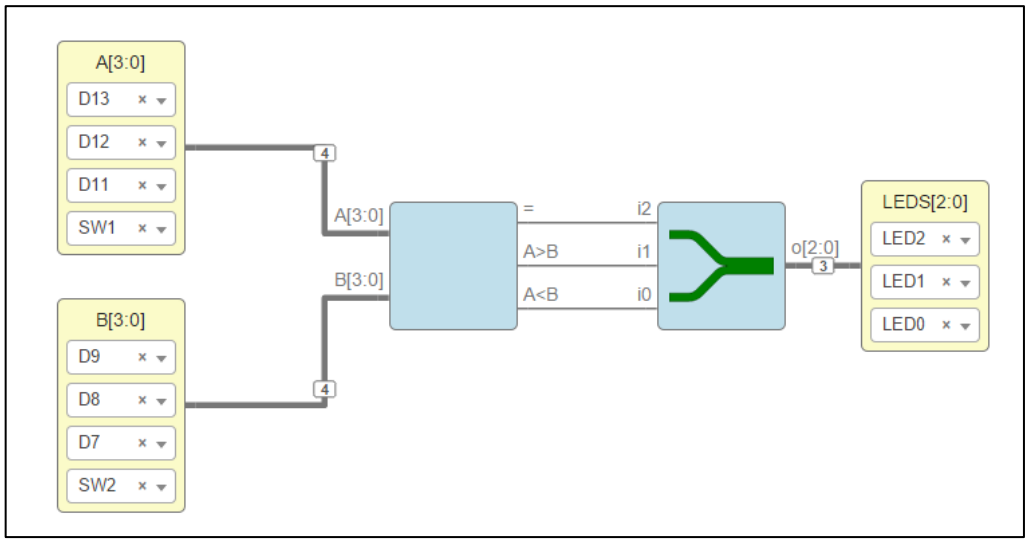
Los puertos de entrada se colocan como se ha hecho anteriormente: Barra de diseño → Basic →input ("FPGA pin" desmarcado). Los puertos de entrada se configuran para ser de bus de 4 cables como se observa en la figura 27. Los puertos de salida son bloques individuales para cada variable de salida del bloque code y se encuentran en: Barra de diseño → Basic → Output ("FPGA pin" desmarcado).

Por último, se guarda el diseño para poder utilizarlo como bloque en cualquier otro proyecto: Barra de tareas → File → Save.

5.6.3. Ejemplo de aplicación

Comparar 2 números (A y B) de 4 bits. Si el número A es igual al número B se debe encender el led2 de la FPGA. Si el número A es mayor al número B, se debe encender el led1 de la FPGA. Si el número A es menor al número B, se debe encender el led0 de la FPGA.

Figura 28. **Ejemplo de aplicación del comparador de magnitudes**



Fuente: elaboración propia utilizando IceStudio.

El circuito de la figura 28 muestra la aplicación del comparador de magnitudes de la figura 27. Si se hace doble click sobre el bloque del comparador, se abre una ventana con el diseño de la figura 27, por lo que en este ejemplo se demuestra que un bloque diseñado en un proyecto previo puede utilizarse en uno nuevo, simplemente agregándolo como bloque teniendo en cuenta la correcta configuración de los puestos de entrada y salida (habilitando o no la opción de “FPGA pin”).

Analizando la figura 28, se puede observar del lado izquierdo dos bloques de entrada para los números A y B, configurados para conectarse por medio de un bus de 4 líneas: Barra de diseño → Basic → Input (FPGA pin habilitado) → A[3:0] y B[3:0]. Los bloques de entrada se conectan al bloque del comparador por medio de los puertos de entrada para bus de 4 líneas.

Del lado derecho de se presenta un bloque que agrupa 3 líneas de datos correspondientes a las 3 salidas del comparador en un bus de 3 líneas que va conectado con un puerto de salida de 3 leds con un bus de entrada de 3 líneas. El bloque para agrupar las líneas de datos se encuentra en: Barra de diseño → Varios → Bus → 03_bits → Agregador. Por otra parte el bloque de salida de los led: Barra de diseño → Basic → Output (FPGA pin habilitado) → LEDS[2:0].

Después de conectar todos los componentes como muestra la figura 28, se procede a cargar el proyecto en la FPGA: Barra de tareas → Tools → Verify → Build → Upload.

Para probarlo, basta con presionar los switches para cambiar el valor de ambos números y verificar la salida del comparador en los leds. Si se desea probar con diferentes valores, se puede cambiar los switches por pines de la FPGA y conectarlos a dip-switches para ingresar cualquier número de 4 bits que se desee en las entradas para el número A y B.

5.7. Control de tiempo

Como se expone en el capítulo 4, los circuitos secuenciales sincrónicos abordados en este trabajo de investigación manejan información de entrada en instantes de tiempo discretos. Los cambios en la salida, así como el ingreso y

proceso de los datos se lleva a cabo en armonía con los pulsos de reloj. Por lo que es importante comprender como se maneja el tiempo en la FPGA.

La FPGA posee un reloj del sistema que es de 12Mhz con un período de 83.3ns aproximadamente. Estos datos son muy importantes para el control de tiempo en cualquier circuito que se desea diseñar utilizando la FPGA.

5.7.1. Tics

En la colección de bloques de IceStudio se maneja el tiempo de uno o varios bloques utilizando una medida de tiempo denominada “Tic”. El tic se define como el pulso mínimo (83.3ns) que se puede producir o manejar en un circuito utilizando la FPGA.

Debido a su aplicación en los circuitos, los tics se encuentran en bloques especiales en la colección de IceStudio. Los bloques generadores de tics son:

- Pulsador de tics: se utiliza principalmente con elementos de entrada como pulsadores/botones o cualquier otro elemento que genere ruido a al momento de utilizarlo. El bloque se encarga de generar un tic limpio cuando recibe un cambio en la entrada. Éste bloque se encuentra en: Barra de diseño → Varios → Pulsadores → Pulsador-tic.

- IR de tics: el IR de tics realiza la misma función que el pulsador de tics con la diferencia que se utiliza con sensores ópticos, generando un tic cuando se detecta un objeto. Éste bloque se encuentra en: Barra de diseño → Varios → IR → IR-tic.

- Corazón de tics: es un bloque que una serie de tics con un tiempo de espaciamiento T. Éste bloque se encuentra en: Barra de diseño → Varios → Bombeo → Tics.

5.7.2. Temporizador

El bloque de temporizador puede encontrarse en: Barra de diseño → Varios → Timer. El bloque de timer se encarga de llevar el control de un tiempo específico utilizando sus dos salidas para generar un pulso de longitud T o para generar un tic cuando el tiempo ya ha finalizado.

El bloque de timer cuenta con 3 puertos de entrada que cuentas con las siguientes características:

- Pin de start: se encuentra en el lado inferior izquierdo del bloque y es el encargado de recibir un tic de disparo, que indica que se debe iniciar con el control de tiempo especificado.

- Una entrada para el CLK o reloj del sistema que se encuentra del lado superior izquierdo.

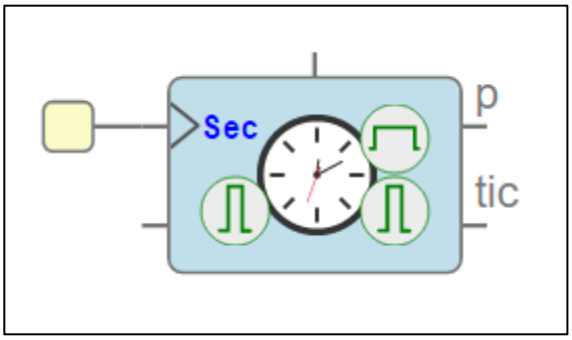
- Pin de parámetro: se encuentra en el lado superior del bloque y se encarga de leer el valor del parámetro de tiempo especificado. Según el tipo de bloque de timer que se utiliza, el parámetro estará en segundos, microsegundos o milisegundos.

Los puertos de salida del bloque de timer son 2 y tienen dos funciones diferentes:

- Salida de pulso: es la salida superior derecha del bloque y se encarga de mantener un pulso en alto durante el periodo de tiempo especificado por medio de la entrada de parámetro.

- Salida de tic: se encuentra en la parte inferior derecha del bloque y se encarga de enviar un tic cuando el periodo de tiempo especificado en la entrada de parámetro del bloque ha concluido.

Figura 29. **Bloque de temporizador**



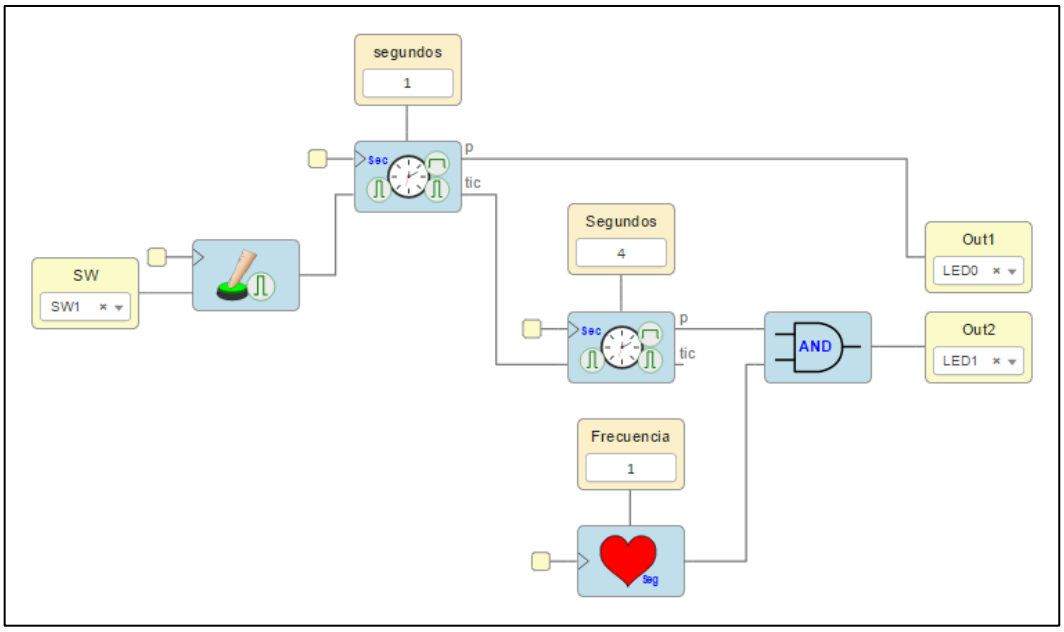
Fuente: Colección de IceStudio.

La figura 29 muestra el bloque timer de IceStudio. Su funcionamiento es muy simple, al recibir un tic por medio del pin de start, el bloque lee el valor del parámetro ingresado por el pin asignado a esta tarea. Inmediatamente el pin de salida de pulso pasa de estado bajo al estado en alto por un periodo de tiempo igual al valor del parámetro ingresado. Una vez se cumple el tiempo establecido, el pin de salida de pulso pasa de estado alto a un estado bajo. Por otro lado, el pin de salida de tic emite un tic de salida indicando que el periodo de tiempo determinado por el parámetro ha concluido.

5.7.3. Ejemplo de aplicación

Se necesita encender el led0 de la FPGA durante 1s y luego mantener parpadeando el led1 durante 4s. Se deben utilizar los bloques de tics y timers disponibles en la colección de IceStudio.

Figura 30. **Ejemplo de aplicación utilizando bloques de tics y timers**



Fuente: elaboración propia utilizando IceStudio.

La figura 30 muestra el circuito diseñado en IceStudio utilizando los bloques de timers y tics como se requiere para el ejemplo de aplicación.

De este ejemplo en adelante, se detalla el funcionamiento de los circuitos sin profundizar en como colocar los bloques debido a que, en la descripción de cada uno de ellos se indica dónde encontrarlos.

Comenzando desde el lado izquierdo de la figura 30, se observa un bloque de entrada asociado al switch1 de la FPGA. El switch1 se conecta a la entrada del bloque de pulsador de tics, que se encarga de enviar un tic cuando se presiona el switch1. A la derecha de este bloque, se encuentra el primer bloque de timer con un parámetro de 1s. La salida de pulso del timer se conecta al led0 de la FPGA para que se mantenga en estado alto durante el segundo en el que el timer se encuentra activo. La salida de tic se conecta a la entrada tic del segundo timer. El timer2 tiene un parámetro de 4 segundos teniendo conectada su salida de pulso a una compuerta AND que, a su vez tiene en su otra entrada un generador de pulsos de 1s. Mientras el timer se mantenga activo durante los 4s, la salida de pulso se encuentra en alto, por lo que la compuerta AND deja pasar los pulsos del generador haciendo parpadear el led1. Cuando el timer detecta el paso de tiempo de 4s la salida de pulso pasa a un estado bajo y los pulsos del generador ya no llegan al led1 manteniéndolo apagado.

5.8. Biestable

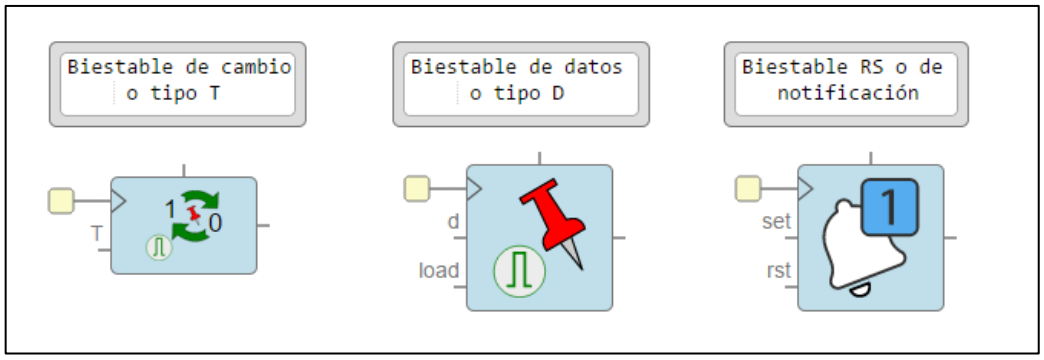
Dado que un elemento importante presente en los circuitos secuenciales son las unidades de memoria, IceStudio tiene bloques en su colección que están diseñados para cumplir las funciones de los flip-flops expuestos en el capítulo 4.

IceStudio emplea los biestables como unidades de memoria capaces de almacenar un bit. Los bloques de biestable tienen diferencias con los flip-flops expuestos en el capítulo 4 debido a que estos bloques no se rigen por tablas de verdad y están diseñados para realizar funciones específicas, por lo cual, se utilizan para diferentes propósitos según la necesidad del circuito que se desea diseñar.

Si se desea trabajar con los flip-flops, se puede crear un bloque específico que opere bajo los parámetros de una tabla de verdad y obtener el mismo comportamiento de los flip-flops tipo D, T y RS. El por qué no se encuentran estos bloques en IceStudio, es debido a que es más sencillo utilizar los bloques de biestable que vienen en la colección ya que elimina el diseño que conlleva el análisis de cada tabla de verdad para cada flip-flop al utilizar los diversos bloques que facilitan la creación de circuitos regidos por parámetros.

Para fines de aprendizaje, se pueden crear bloques de flip-flop convencionales utilizando el bloque de code, para sintetizar el circuito del flip-flop deseado empleando Verilog.

Figura 31. **Tipos de biestable**



Fuente: elaboración propia utilizando IceStudio.

La figura 31 muestra los 3 tipos de bloque de biestable disponibles en IceStudio y como se menciona anteriormente, cada uno se utiliza con diferentes propósitos.

5.8.1. Biestable tipo T

También llamado biestable de cambio. Su característica es cambiar de estado cada vez que recibe un tic en la entrada. Si el valor inicial es 0, al detectar un tic en la entrada, el valor de salida cambia a 1. Si el valor de la salida es 1 y se detecta un tic en la entrada, el valor de la salida cambia a 0.

Como se muestra en la figura 31, el bloque presenta 3 puertos de entrada y un puerto de salida. Los puertos y sus características son los siguientes:

- Entrada de tic: es el encargado de recibir los tics de entrada que permitirán el cambio de estado en la salida del biestable y se encuentra en la parte inferior izquierda del bloque.

- Reloj del sistema: conectado por default al reloj del sistema, pero puede configurarse para que el bloque trabaje a la frecuencia deseada y se encuentra en la parte superior izquierda del bloque.

- Parámetro: es una entrada para el valor inicial de la salida, puede ser 1 o 0. El pin se encuentra en la parte superior del bloque.

- Salida: puerto de salida de 1 bit que puede tomar el valor de 0 o 1. Se encuentra en la parte derecha del bloque.

El bloque del biestable tipo T o de cambio se encuentra en: Barra de diseño → Varios → Biestables → cambio.

5.8.2. Biestable de datos o tipo D

Los biestables de datos o tipo D se utilizan para el manejo y almacenamiento de bits de datos y retenerlos hasta que nuevos datos ingresen al bloque.

La figura 31 muestra el bloque de un biestable tipo D. Como se puede observar el bloque consta de 4 puertos de entrada y un puerto de salida. Los puertos y sus características son las siguientes:

- Tic de captura: es un puerto de entrada y se utiliza para detectar un tic que indica que se debe almacenar un bit en la entrada. El puerto se encuentra en la parte inferior izquierda del bloque.

- Entrada de dato: es el segundo puerto de entrada y es el encargado de recibir el bit que se desea guardar. El puerto se encuentra entre el puerto de tic de captura y el puerto del reloj del sistema.

- Reloj del sistema: entrada de reloj conectada por default al reloj del sistema, pero puede conectarse a un reloj con la frecuencia de operación deseada. El puerto se encuentra en la parte superior izquierda del bloque.

- Parámetro: es la cuarta entrada del bloque y se encarga de recibir el valor inicial que tendrá el biestable cuando inicie su operación.

- Salida de dato: es el único puerto de salida del bloque, e indica el valor del bit almacenado en el biestable. El puerto se encuentra en el lado derecho del bloque.

El funcionamiento del bloque biestable tipo D es muy simple: se presenta por la entrada de dato el bit (0 o 1) que se desea guardar. Cuando se detecta un tic por la entrada de tic de captura, el bit presente en la entrada de dato se guarda y pasa a la salida. El valor en la salida no cambia hasta que se guarde un bit diferente.

Para utilizar el biestable tipo D, basta con buscarlo en la colección de IceStudio: Barra de diseño → Varios → Biestables → Dato.

5.8.3. Biestable RS

El bloque de un biestable RS se muestra en la figura 31. Como se puede observar, IceStudio utiliza el símbolo de la campana y la notificación para describir la función de este bloque: enviar una notificación cuando ocurre un evento.

La figura 31 muestra la distribución de los puertos en el bloque del biestable RS, el cual presenta 4 puertos de entrada y 1 de salida. Los puertos y sus características son las siguientes:

- Reset: puerto de entrada ubicado en la parte inferior izquierda del bloque cuya función es poner en la salida el valor por defecto del biestable, que puede ser 0 o cualquier otro valor ingresado por medio de la entrada de parámetro.

- Set: es un puerto de entrada ubicado a inmediación del lado izquierdo del bloque y su función es colocar en la salida el valor contrario al valor por defecto del biestable, que puede ser 1 o el valor opuesto al ingresado por medio de la entrada de parámetro.

- Reloj del sistema: entrada de reloj conectada por default al reloj del sistema, pero puede conectarse a un reloj con la frecuencia de operación deseada. El puerto se encuentra en la parte superior izquierda del bloque.

- Parámetro: puerto de entrada utilizado para colocar el valor por default del biestable. El valor puede ser 0 o 1 e influye en el comportamiento del biestable ya que afecta el valor colocado a la salida por los puertos reset y set.

- Notificación: único puerto de salida ubicado en la parte derecha del bloque. Los valores de salida pueden ser 0 o 1 dependiendo de los cambios en los puertos de entrada.

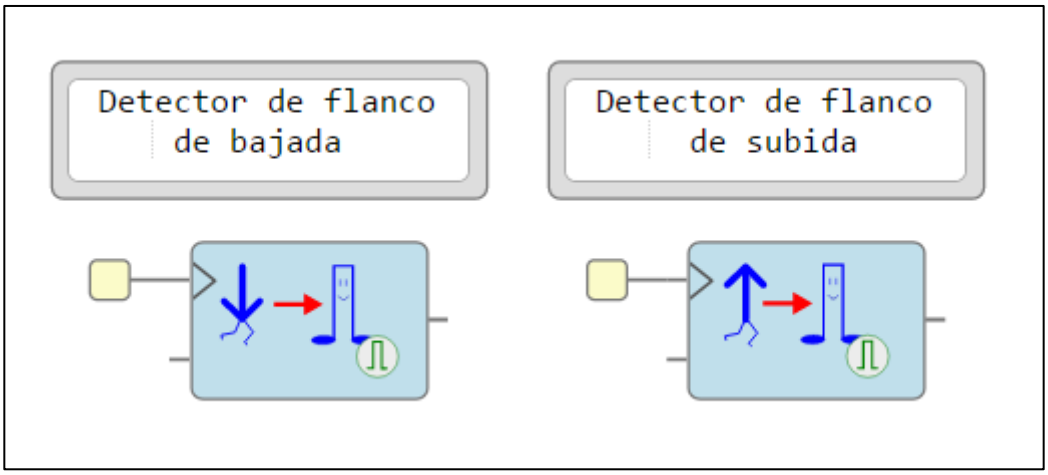
El funcionamiento del bloque de biestable RS es muy simple: suponiendo que en la entrada de parámetro se encuentra conectada a un bloque de constante con un valor de 0, esto indica que, si se presenta un bit de 1 en la entrada set, la salida notificación toma un valor de 1. Si en la entrada de reset se presenta un bit de 1, la salida notificación vuelva al valor presente en la entrada de parámetro que es 0 en este caso. Por otro lado, si en la entrada de parámetro el valor de la constante es 1. Al presentarse un bit de 1 en la entrada set, la salida notificación toma un valor de 0. Si se presenta un bit de 1 en la entrada reset, la salida notificación toma un valor de 1. Los valores en la salida se mantienen hasta que se presente un bit en la entrada opuesta del biestable.

Para utilizar el biestable tipo D, basta con buscarlo en la colección de IceStudio: Barra de diseño → Varios → Biestables → Set-Reset.

5.8.4. Detector de flancos

El detector de flancos como su nombre lo indica, se encarga de enviar un tic por la salida cuando detecta un flanco de subida o bajada dependiendo el tipo de bloque que se utiliza.

Figura 32. **Bloques detectores de flancos**



Fuente: elaboración propia utilizando IceStudio.

Los bloques mostrados en la figura 32 no son biestables, pero son bloques indispensables que se utilizan en conjunto con los bloques de biestable tipo D y de cambio debido a que emplean tics en su entrada para funcionar. En diferentes circuitos se necesita llevar a cabo una función cuando se detecta un flanco de subida o bajada y es para estos casos en los cuales los bloques de detección de flancos son útiles.

IceStudio cuenta con dos bloques detectores de flancos: detector de flanco de subida y detector de flanco de bajada. Como se muestra en la figura 32, ambos

bloques de detección de flancos cuentan con 2 puertos de entrada y 1 puerto de salida, cuyas características son:

- Reloj del sistema: entrada de reloj conectada por default al reloj del sistema, pero puede conectarse a un reloj con la frecuencia de operación deseada. El puerto se encuentra en la parte superior izquierda del bloque.

- Entrada: puerto por el cual ingresa la señal que se somete a análisis para detectar el flanco que se desea.

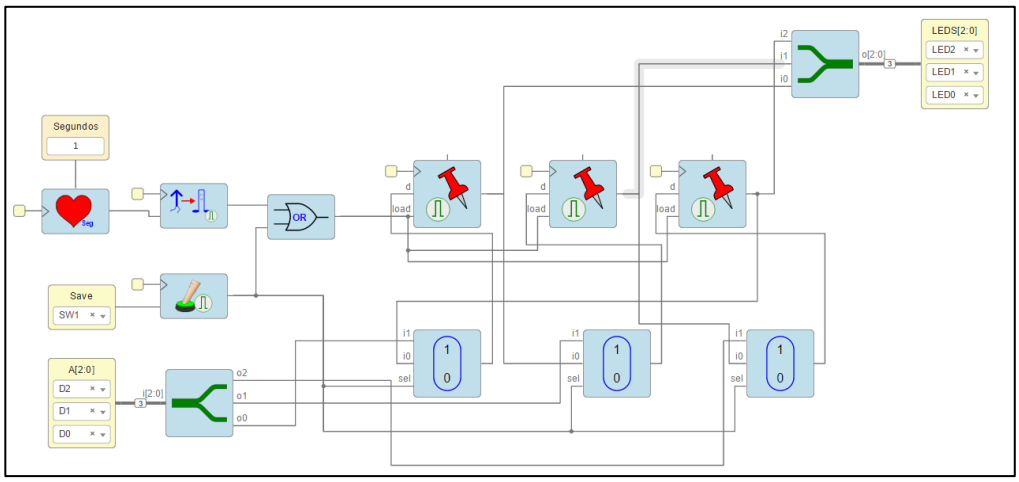
- Salida: puerto por el cual se envía un tic cada vez que se detecta un flanco de subida o un flanco de bajada según el tipo de bloque que se utiliza.

Los detectores de flancos se encuentran en: Barra de diseño → Varios → Flancos.

5.8.5. Ejemplo de aplicación

Se solicita diseñar un circuito de “luces fantásticas” en el cual se pueda ingresar la secuencia en la que los leds deben encenderse y apagarse. La secuencia se puede cambiar en cualquier momento y se debe colocar un botón para poder guardarla. El corrimiento debe de ser desde el led0 hacia el led2, teniendo un retraso de 1s entre cada bit de corrimiento.

Figura 33. **Ejemplo de aplicación utilizando bloques de biestables**



Fuente: elaboración propia utilizando IceStudio.

La figura 33 muestra el circuito que satisface la solicitud planteada en el párrafo anterior. El ingreso de bits se hace por medio de un bloque de 3 entradas asociadas a los primeros 3 pines del puerto D de la FPGA. La salida del bloque de entrada es un bus de 3 cables los cuales se separan utilizando un bloque separador. Cada cable se conecta a un multiplexor distinto de un arreglo de multiplexores que se encarga de dejar pasar los bits hacia el arreglo de biestables tipo D encargados del corrimiento.

Un bloque de entrada asociado al switch1 de la FPGA es el encargado de enviar una señal a un bloque de pulsador-tic para enviar un tic hacia la entrada de select de los multiplexores cuando el switch es presionado. Este mismo tic se utiliza como señal para guardar en la entrada load del arreglo de biestables tipo D cuando se suma con otra señal proveniente de un detector de flancos de subida.

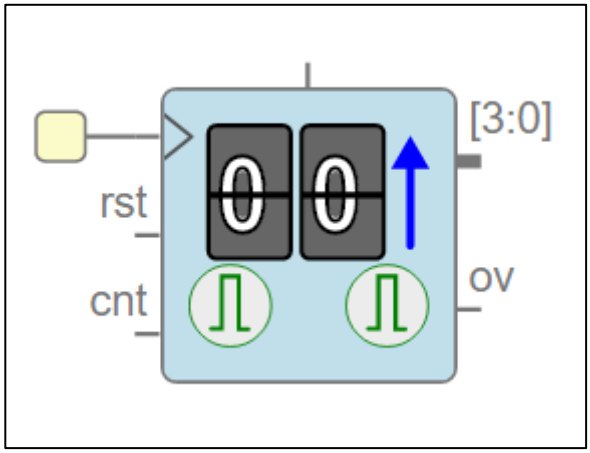
El arreglo de biestables tipo D está configurado para realizar un corrimiento perpetuo de bits ya que la salida del último biestable se conecta a la entrada del primer biestable. Un bloque de corazón es el encargado de enviar pulsos a cada segundo a un detector de flanco de subida el cual, se encarga de enviar un tic a un sumador, encargado de dejar pasar el tic proveniente del detector o del pulsador-tic. El tic de salida del sumador se conecta a la entrada load del arreglo de biestables tipo D para realizar el corrimiento, guardando la salida de un biestable en el siguiente. Por último, se tiene un bloque de salida de 3 leds conectado a un bus de 3 cables el cual, agrupa los 3 cables de salida de cada biestable del arreglo.

5.9. Contador

Los contadores son circuitos combinacionales muy útiles al momento de contar eventos. Un contador se puede diseñar utilizando compuertas lógicas y la respectiva tabla de verdad que es, el mapa de comportamiento del circuito.

En IceStudio se puede crear un bloque de contador que contenga el diseño de un contador como el expuesto en la figura 16 y tabla XX, pero es algo más complejo ya que IceStudio tiene bloques de contadores que se pueden utilizar y que ahorran tiempo en el diseño.

Figura 34. **Bloque de un contador**



Fuente: colección de IceStudio.

La figura 34 muestra el bloque de un contador ascendente de 4 bits disponible en la colección de bloques de IceStudio. Consta de 4 puertos de entrada y 2 puertos de salida que se describen a continuación:

- Entrada de tics: es el puerto denominado “cnt” en la figura 34. Es un puerto de entrada y su función es la de detectar los tics de entrada que son los que utiliza el bloque para aumentar la cuenta.

- Reset: es un puerto de entrada denominado “rst” en la figura 34. Su función es regresar la cuenta a 0 cuando llega un bit de 1 en el puerto.

- entrada de reloj conectada por default al reloj del sistema, pero puede conectarse a un reloj con la frecuencia de operación deseada. El puerto se encuentra en la parte superior izquierda del bloque.

- Módulo (cuenta máxima): puerto en el cual se ingresa un parámetro el cual, indica hasta que número llegará la cuenta para después regresar a 0 y generar un tic de desbordamiento (overflow) por el puerto “ov”. Si no se coloca un parámetro la cuenta llegará hasta 2^n donde n es el número de bits del contador.

- Desbordamiento (overflow): es un puerto de salida que indica cuando se ha llegado a la cuenta máxima (overflow) enviando un tic.

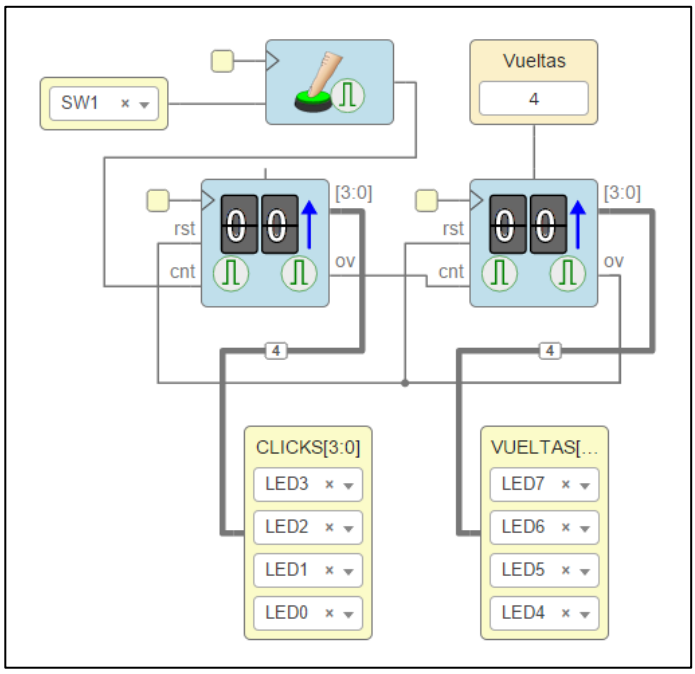
- Salida (bus de n bits): es un puerto de salida formado por un bus de n bits en el cual se envía la cuenta actual del contador. Se puede observar que el contador de la figura 34 es de 4 bits debido a que el bus de salida es de 4 bits.

Para utilizar un contador de n bits, basta con buscarlo en la colección de IceStudio: Barra de diseño → Varios → contadores.

5.9.1. Ejemplo de aplicación

Diseñar circuito que lleve el conteo del número de veces que se presiona el switch1 de la FPGA. El conteo debe llegar hasta 15 y se debe mostrar en los primeros 4 leds de la FPGA. En los 4 leds restantes se debe mostrar cuantas veces se ha reiniciado el conteo (vueltas). Cuando las vueltas lleguen a 4, ambos contadores deben reiniciarse a 0.

Figura 35. **Circuito con bloques de contadores**



Fuente: elaboración propia utilizando IceStudio.

La figura 35 muestra un circuito que cumple con lo que se solicita en el párrafo anterior. Como se puede observar, se emplean 2 bloques de contador de 4 bits los cuales llevan el conteo del número de veces en que se presiona el switch1 y el número de veces en que se ha reiniciado el conteo. Se tiene un bloque de entrada asociado al switch1 de la FPGA, cuya salida se conecta a un bloque pulsador-tic encargado de enviar un tic por su salida cada vez que se presiona el switch1.

El contador a la izquierda de la figura 35, es el encargado de llevar el conteo del número de veces que se presiona el switch1. El contador se incrementa cada vez que detecta en su entrada el tic que proviene al momento de pulsar el switch1 y al mismo tiempo envía un tic (por su salida de overflow) al segundo contador

cada vez que la cuenta supera las 15 pulsaciones y se reinicia el contador. La salida de este contador se conecta a un bloque de salida con los primeros 4 leds de la FPGA por medio de un bus de 4 cables.

El contador a la derecha de la figura 35, es el encargado de llevar el conteo del número de veces en que la cuenta de las pulsaciones se reinicia. La entrada de módulo está conectada a un bloque de constante, donde se coloca el valor de “4” correspondiente al número máximo de vueltas que se deben contar. El tic de overflow proveniente del contador anterior, se utiliza para aumentar el conteo. La salida de overflow se conecta en la entrada de reset de ambos contadores, logrando así, reiniciar a 0 ambos contadores al momento de enviar el tic de overflow cuando se sobrepasa el conteo de 4 vueltas. La salida de este contador se conecta a un bloque de salida con los 4 últimos leds de la FPGA por medio de un bus de 4 cables.

CONCLUSIONES

1. La FPGA libre Alhambra II es un dispositivo de apoyo para el estudiante debido a que es fácil de utilizar, con lo cual, se facilita el aprendizaje de la electrónica digital al tener una herramienta que permite de forma sencilla desarrollar circuitos digitales de aplicación a los temas que se estudian.

2. La plataforma de IceStudio demuestra ser una buena opción para el estudiante que inicia en la electrónica debido a que se puede describir hardware en una FPGA de forma gráfica. Para el estudiante a nivel intermedio, está la opción de crear los bloques necesarios para diferentes propósitos mediante otros bloques o también por medio de los bloques de código que permiten por medio de Verilog describir el hardware necesario.

3. Todo el software libre que existe actualmente para el desarrollo de hardware es de mucha ayuda para los estudiantes que no pueden pagar la licencia de un software. Por tal razón, IceStudio presenta una ventaja no solo económica, también una facilidad de empleo debido a todas las capacidades y herramientas disponibles para el desarrollo de cualquier circuito electrónico en una FPGA libre.

.

RECOMENDACIONES

1. Visitar la página oficial de la FPGA libre Alhambra II para contenido más detallado sobre las librerías de bloques de IceStudio.

2. Existe un tutorial en crecimiento en el github de la FPGA libre Alhambra II en el cual se puede encontrar en mayor detalle los temas expuestos en este trabajo de graduación.

3. Mantener actualizada la información del movimiento de FPGA´s libres debido a que se encuentran desarrollando toolchains para FPGA´s comerciales como Xilinx.

BIBLIOGRAFÍA

1. Apio. *Documentación,* ©2018. [en línea]. <https://apiodoc.readthedocs.io/en/stable/source/installation.html>. [consulta: junio 2019].

2. ARROYO, Jesus. *IceStudio User Guide*, ©2016-2018. [en línea]. <https://icestudio.readthedocs.io/en/latest/source/userguide.html>. [consulta: mayo 2019].

3. DONOVAN, James. *Electrónica digital*. 1ª ed. México: Continental, 1997. 691 p.

4. GNU. *Software libre*, ©1996, 2002, 2004-2007, 2009-2019. [en línea]. <https://www.gnu.org/philosophy/free-sw.es.html>. [consulta: junio 2019].

5. GONZALEZ, Juan. *Tutorial de electrónica digital para makers con FPGAs libres*, ©2019. [en línea]. <https://github.com/Obijuan/digital-electronics-with-open-FPGAs-tutorial/wiki>. [consulta: mayo 2019].

6. Lattice semiconductor*. iCE40™ LP/HX Family Data Sheet.* [en línea]. < http://www.latticesemi.com/~/media/LatticeSemi/Documents/DataSheets/iCE/iCE40LPHXFamilyDataSheet.pdf>. [consulta: abril 2019]

7. MANO, Morris. *Diseño digital*. 3ª ed. México: Pearson Educación, 2003. 536 p.

8. MOORE, Andrew. *FPGAs for dummies.* 2ª ed. Estados Unidos: John Wiley & Sons, 2017. 44 p.

9. WOLF, Clifford. *Project Icestorm.* [en línea]. <http://www.clifford.at/icestorm>. [consulta: junio 2019].

10. Xilinx. *Spartan-6 FPGA Configuration.* [en línea]. <https://www.xilinx.com/support/documentation/user_guides/ug380.pdf>. [consulta: abril 2019].