

FACULTAD DE
INGENIERÍA Y CIENCIAS



UNIVERSIDAD DIEGO PORTALES

Laboratorio 3
Modelado VHDL y Síntesis Secuencial Diseño
Lógico

Arquitectura y Organización de Computadores

Matías Antonio Astudillo Astete

Matías Ignacio Rivera Sáez

Marcos David Valderrama Carrasco

Profesor: Jorge Tobar Pinto

Jueves 30 de Mayo de 2024

Índice

1. Introducción	2
2. Marco teórico	3
3. Procedimiento teórico	4
4. Simulación	5
5. Experimentación Física	7
5.1. Configuración de Software y Hardware	7
5.2. Programación de circuito secuencial	8
6. Resultados experimentales	11
7. Comparación de resultados	12
8. Conclusiones	12
9. Referencias	13

1. Introducción

Para esta experiencia se propone el objetivo de diseñar un autómata, el cual sea capaz de detectar una secuencia de bits provenientes de una entrada, para que luego, una vez ingresada dicha secuencia, se active su correspondiente salida, todo esto mientras se tienen en cuenta los posibles solapamientos que podrían llegar a suceder al ingresar bits no correspondientes en la secuencia.

Para cumplir con este objetivo, se desarrollará un modelo VHDL del diseño en el software vivado, el cual será implementado en la placa FPGA NEXYS 4 DDR .

Inicialmente, se profundiza en el uso del software Boole para generar un diagrama de Moore y así obtener el código y circuito correspondiente para la secuencia deseada. Posteriormente, se recrea el circuito en un simulador para comprobar su correcto funcionamiento. Finalmente, se programa la placa para verificar el funcionamiento práctico en un entorno real.

2. Marco teórico

Un circuito secuencial es un tipo de circuito digital cuya salida depende no solo de las entradas actuales, sino también del historial de entradas anteriores. Esto se debe a que los circuitos secuenciales tienen *flip-flops*, chips que permiten almacenar bits de memoria. Esto les permite almacenar información sobre estados pasados.

Un *flip-flop* es un tipo de circuito digital de memoria diseñado para almacenar un bit de información. Los *flip-flops* son dispositivos biestables, lo que significa que pueden mantener uno de dos estados estables indefinidamente, hasta que se les proporcione una señal en su entrada de reloj para cambiar de estado. Estos dispositivos se utilizan en sistemas secuenciales para almacenar estados y, dependiendo del estado actual del circuito y de sus entradas, generar el siguiente estado.

A continuación, se explica la diferencia entre dos tipos comunes de *flip-flops*:

- **Flip-flop D:** Este tipo de *flip-flop* tiene una entrada de datos (*D*) y una entrada de reloj (*clock*). El valor en la entrada de datos se transfiere a la salida (*Q*) durante la activación del reloj. Es sencillo de utilizar, ya que la salida sigue o copia la señal de entrada en el instante en que el reloj se activa, y mantiene esa señal mientras el reloj está desactivado.
- **Flip-flop JK:** Este *flip-flop* tiene dos entradas de datos (*J* y *K*) y una entrada de reloj. Su funcionamiento es más complejo que el del *flip-flop D*, ya que permite más operaciones: cuando *J* y *K* son ambos 1, la salida invierte su estado. Si *J* es 1 y *K* es 0, la salida se pone en 1. Si *J* es 0 y *K* es 1, la salida se pone en 0. Cuando ambos son 0 la salida se mantiene sin cambios.

3. Procedimiento teórico

En esta experiencia se busca diseñar un autómata capaz de detectar, desde una entrada "X", la secuencia de bits 1011101, y de esta manera activar una salida "Z".

Para empezar, es necesario que identifiquemos la cantidad de estados posibles. Estos van a ser un total de 8, dado a que tenemos 7 estados dados por la cantidad de bits y 1 un estado inicial.

A continuación, es necesario construir las tablas que nos permitan identificar las transiciones entre estados (considerando solapamientos) en base a la entrada, y la salida de cada uno de ellos.

	X=0	X=1
Q0	Q0	Q1
Q1	Q2	Q1
Q2	Q0	Q3
Q3	Q2	Q4
Q4	Q2	Q5
Q5	Q6	Q1
Q6	Q0	Q7
Q7	Q2	Q4

Cuadro 1: Tabla de transición.

	Z
Q0	0
Q1	0
Q2	0
Q3	0
Q4	0
Q5	0
Q6	0
Q7	1

Cuadro 2: Tabla de salida.

Luego, con los datos de las tablas, procederemos a generar un diagrama de Moore a través del programa *Boole*, donde se representa el funcionamiento lógico del autómata:

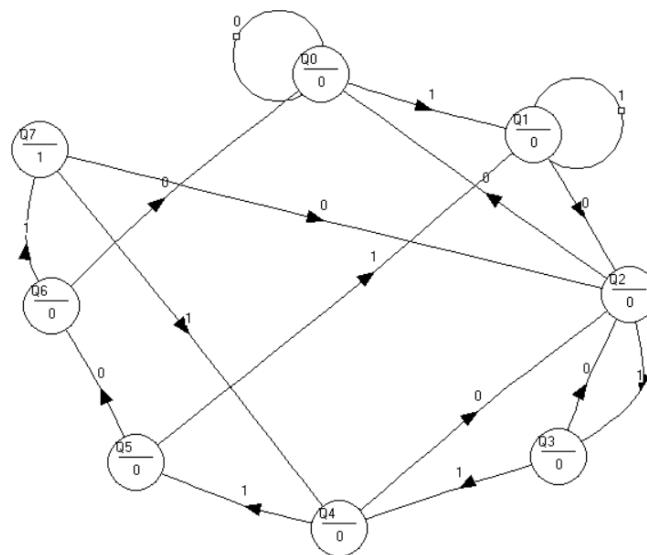


Figura 1: Diagrama de Moore sobre funcionamiento del autómata.

4. Simulación

Luego de construir el diagrama de Moore con el software *Boole*, se debe comprobar dentro de este mismo si la secuencia es correcta y determinista. Esto significa que el diagrama debe producir la misma salida para un mismo conjunto de entradas, dependiendo del estado actual del circuito. Para cualquier estado y entrada, la transición al estado siguiente y la salida deben estar completamente definidas. Además, el circuito debe ser capaz de soportar solapamientos, lo que implica que debe poder gestionar situaciones en las que las entradas cambien de manera que las nuevas entradas se solapen con el conjunto de estados anterior, manteniendo la coherencia y el orden de los estados.

Una vez comprobado que la secuencia es correcta y determinista, se procede a generar el circuito. El software ofrece la creación de un circuito utilizando Flip-flop D o Flip-flop JK.

Para esta experimentación se elige el flip-flop D, ya que, a pesar de que ambos tipos pueden lograr el mismo resultado, en este caso el flip-flop D requiere una configuración de cableado más sencilla en comparación al flip-flop JK.

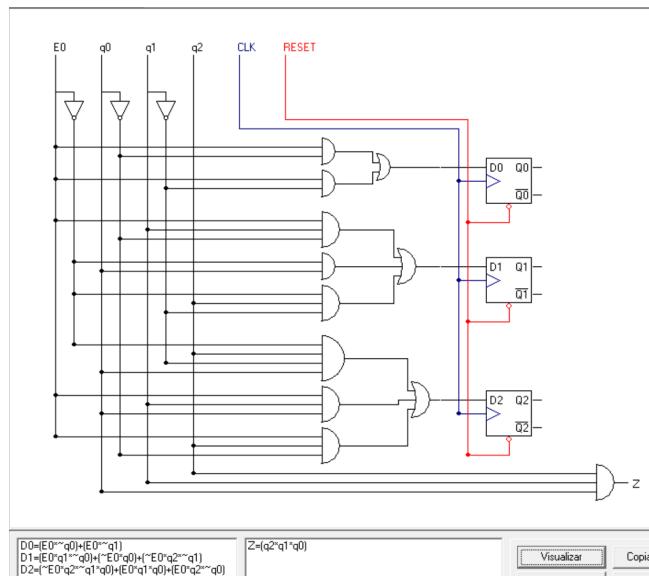


Figura 2: Circuito secuencial generado por software Boole.

El circuito y las ecuaciones correspondientes a cada *flip-flop* y salida se muestran en la Figura 2. En esta figura, se observan un total de tres *flip-flops*, esto se debe a que el número de estados posibles es ocho, y por lo tanto, estos estados pueden representarse mediante la combinación de tres bits.

Para la simulación del circuito se utiliza el software *logisim-evolution* [3]. Un software que permite recrear y simular circuitos combinacionales y secuenciales, ofreciendo una alta gama de configuraciones. Además, permite recrear el diagrama de tiempo de ejecución del circuito.

En la figura 3 se presenta el circuito en su estado final. Solo en este estado el led de la salida se

encuentra encendido, ya que se requiere que los tres *Flip-flop D* se encuentren con su salida activada en el mismo instante de reloj.

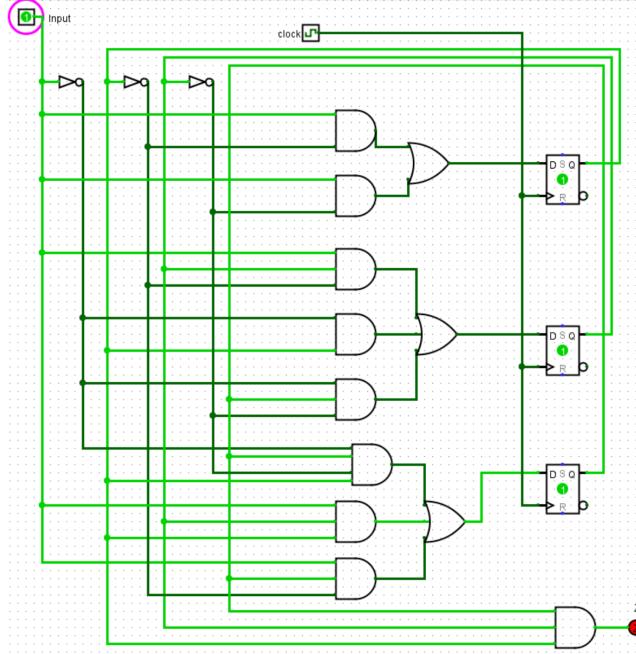


Figura 3: Simulación de circuito en logisim-evolution.

A continuación, se adjunta un link con el ingreso completo de la secuencia de bits 1011101:

<https://drive.google.com/file/d/1hethg6ysc2zdeGFDRValQ2ejTCyqiCSw/view?usp=sharing>

Posteriormente, en base a los resultados de la simulación, se procede a construir la correspondiente tabla de transición y tabla de salida.

	Input=0	Input=1
Q0	Q0	Q1
Q1	Q2	Q1
Q2	Q0	Q3
Q3	Q2	Q4
Q4	Q2	Q5
Q5	Q6	Q1
Q6	Q0	Q7
Q7	Q2	Q4

	Z
Q0	0
Q1	0
Q2	0
Q3	0
Q4	0
Q5	0
Q6	0
Q7	1

Cuadro 3: Tabla de transición simulación. Cuadro 4: Tabla de salida simulación.

Además, en la Figura 4 se muestra el comportamiento de las señales de los *flip-flops* y la salida al

activar la cadena de entrada de bits **1011101**, y posteriormente, al solaparla con **1101** en los instantes precisos del reloj, configurado a una velocidad de 0.5 Hz. A esto se le conoce como diagrama de tiempo.

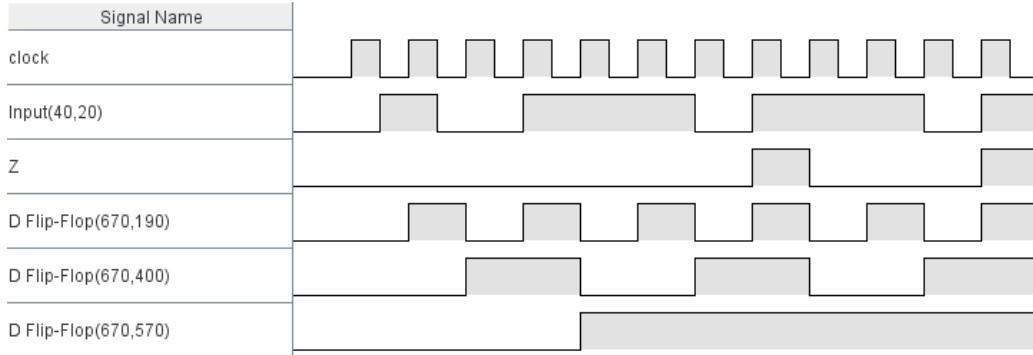


Figura 4: Diagrama de tiempo de circuito secuencial simulado en logisim.

5. Experimentación Física

5.1. Configuración de Software y Hardware

La presente sección expone de manera detallada el procedimiento de simulación llevado a cabo mediante el uso del hardware Nexys 4 DDR [2] (figura 7). Esta tarjeta de desarrollo está equipada con una FPGA (Field Programmable Gate Array), un circuito integrado que puede ser programado post-fabricación para ejecutar una amplia gama de funciones digitales. Adicionalmente, se empleó el software Xilinx Vivado [1], el cual provee un conjunto integral de herramientas para el diseño, síntesis, colocación, enrutamiento, verificación y simulación de circuitos. Mediante la configuración adecuada de ambos componentes, se desea replicar físicamente los resultados obtenidos en la simulación virtual de la sección 4.

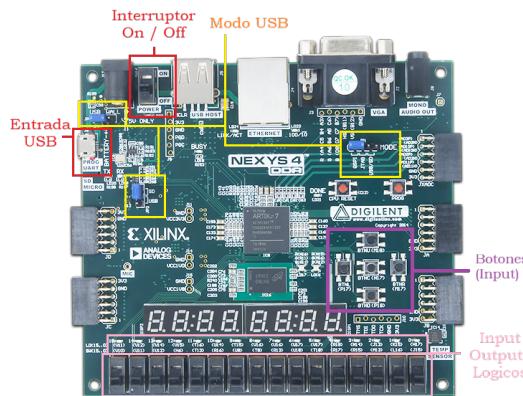


Figura 5: Placa Diligent Nexys 4 DDR.

5.2. Programación de circuito secuencial

Una vez verificamos que el comportamiento del circuito secuencial funciona correctamente durante la simulación, en el mismo software Boole solicitamos el código en lenguaje *vhdl* para replicar dicho comportamiento utilizando la placa Nexys 4. Es importante comentar de que a la hora de implementar el código generado por Boole en Vivado, se pudo observar que el código presentaba algunos errores de formato, por lo que se procedió a hacer una corrección del mismo, quedando de la siguiente manera:

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_ARITH.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 -- Especifica los puertos de entrada y salida
7 entity Sin_Titulo is
8   Port (
9     inicio: in std_logic;
10    ck: in std_logic;
11    E0: in std_logic;
12    Z: out std_logic
13  );
14 end Sin_Titulo;
15
16 architecture behavioral of Sin_Titulo is
17
18 type nombres_estados is (Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7);
19 signal estado: nombres_estados;
20 signal entrada_aux: std_logic_vector (0 downto 0);
21
22 begin
23
24 entrada_aux<=E0
25 -- Proceso para la transición de estados
26 process(inicio, ck)
27 begin
28 if inicio='1' then
29   estado<=Q0; -- Si hay un pulso de inicio, volver al estado Q0
30 elsif ck='1' and ck'event then -- Si hay un cambio en el flanco de subida de la señal de reloj
31   case estado is
32     when Q0 =>
33       case entrada_aux is
34         when '0' => estado<=Q0; -- Mantenerse en Q0 si la entrada es '0'
35         when '1' => estado<=Q1; -- Ir a Q1 si la entrada es '1'
36         when others => estado<=Q0; -- Volver a Q0 para otros valores de entrada
37       end case;
38     when Q1 =>
39       case entrada_aux is
40         when '0' => estado<=Q2; -- Ir a Q2 si la entrada es '0'
41         when '1' => estado<=Q1; -- Mantenerse en Q1 si la entrada es '1'
42         when others => estado<=Q0; -- Volver a Q0 para otros valores de entrada
43       end case;
44     when Q2 =>

```

```

45      case entrada_aux is
46          when '0' => estado<=Q0; -- Volver a Q0 si la entrada es '0'
47          when '1' => estado<=Q3; -- Ir a Q3 si la entrada es '1'
48          when others => estado<=Q0; -- Volver a Q0 para otros valores de entrada
49      end case;
50  when Q3 =>
51      case entrada_aux is
52          when '0' => estado<=Q2; -- Volver a Q2 si la entrada es '0'
53          when '1' => estado<=Q4; -- Ir a Q4 si la entrada es '1'
54          when others => estado<=Q0; -- Volver a Q0 para otros valores de entrada
55      end case;
56  when Q4 =>
57      case entrada_aux is
58          when '0' => estado<=Q2; -- Volver a Q2 si la entrada es '0'
59          when '1' => estado<=Q5; -- Ir a Q5 si la entrada es '1'
60          when others => estado<=Q0; -- Volver a Q0 para otros valores de entrada
61      end case;
62  when Q5 =>
63      case entrada_aux is
64          when '0' => estado<=Q6; -- Ir a Q6 si la entrada es '0'
65          when '1' => estado<=Q1; -- Volver a Q1 si la entrada es '1'
66          when others => estado<=Q0; -- Volver a Q0 para otros valores de entrada
67      end case;
68  when Q6 =>
69      case entrada_aux is
70          when '0' => estado<=Q0; -- Volver a Q0 si la entrada es '0'
71          when '1' => estado<=Q7; -- Ir a Q7 si la entrada es '1'
72          when others => estado<=Q0; -- Volver a Q0 para otros valores de entrada
73      end case;
74  when Q7 =>
75      case entrada_aux is
76          when '0' => estado<=Q2; -- Volver a Q2 si la entrada es '0'
77          when '1' => estado<=Q4; -- Volver a Q4 si la entrada es '1'
78          when others => estado<=Q0; -- Volver a Q0 para otros valores de entrada
79      end case;
80  when others => estado<=Q0;
81 end case;
82 end if;
83 end process;
84
85 -- Proceso para determinar la salida basado en el estado actual
86 process(estado)
87 begin
88 case estado is
89     when Q0 =>Z<='0'; -- Salida '0' para el estado Q0
90     when Q1 =>Z<='0'; -- Salida '0' para el estado Q1
91     when Q2 =>Z<='0'; -- Salida '0' para el estado Q2
92     when Q3 =>Z<='0'; -- Salida '0' para el estado Q3
93     when Q4 =>Z<='0'; -- Salida '0' para el estado Q4
94     when Q5 =>Z<='0'; -- Salida '0' para el estado Q5
95     when Q6 =>Z<='0'; -- Salida '0' para el estado Q6
96     when Q7 =>Z<='1'; -- Salida '1' para el estado Q7
97 end case;

```

```

98 end process;
99
100 end behavioral;
```

Código 1: Archivo de configuracion vhd para el circuito secuencial entregado por Boole.

Posteriormente, se definen las restricciones. Esto implica modificar el archivo con formato “xdc”, en el cual se especifica la relación entre las señales de las entradas/salidas lógicas del código 1 con los pines del FPGA. El archivo utilizado para vincular las entradas y salidas se muestra en el código 2.

```

1 set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets ck_IBUF];
2 set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { inicio }];
3 set_property -dict { PACKAGE_PIN N17 IOSTANDARD LVCMOS33 } [get_ports { ck }];
4 set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { EO }];
5 set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { Z }];
```

Código 2: Archivo de configuracion xdc para el circuito secuencial.

En este caso fue necesario agregar la linea 1 en el código 2, debido a que sin esta ocurría un problema con el clock, el cual no permitía la implementación del código en la placa. Este problema ocurría, ya que se estaba ocupando un botón para controlar las señales del reloj en vez de usar las rutas dedicadas dentro del FPGA.

Todo el proceso de escritura e implementación del código se puede resumir en el siguiente esquema:

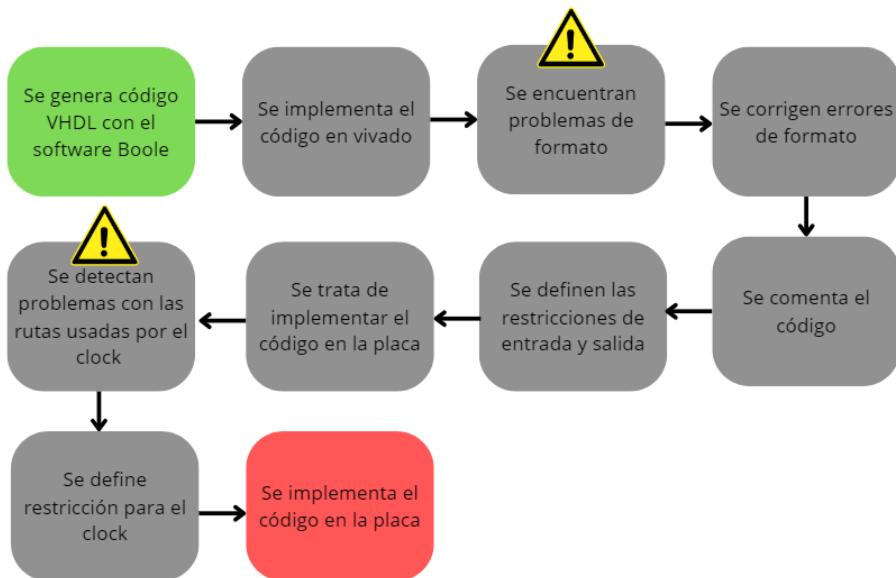


Figura 6: Esquema escritura e implementación del código.

6. Resultados experimentales

Se procede a mostrar una imagen de la placa al encontrarse en el estado Q7:

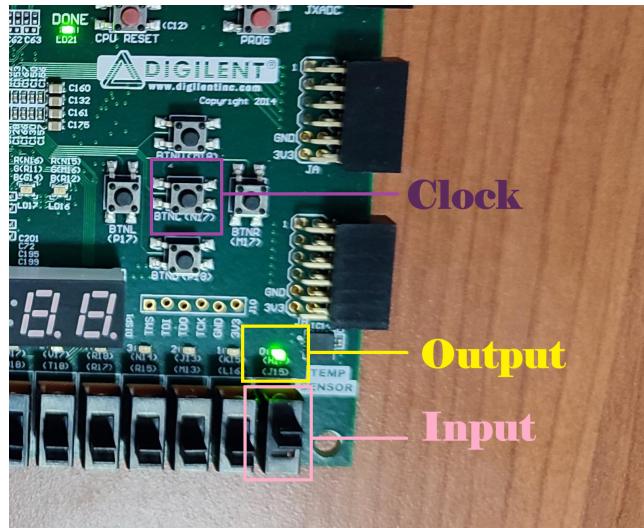


Figura 7: Pines usados en la placa Nexys.

Luego, se adjunta un link con el ingreso completo de la secuencia de bits 1011101:

<https://drive.google.com/file/d/1Ezj3tkm16XDPWoMBwWjNpUD8Y0xFuz2a/view>

Además, se adjunta un ejemplo de solapamiento, donde se pasa del estado Q7 al estado Q4 y luego se ingresa la cadena de bits necesaria para llegar al estado Q7 nuevamente:

https://drive.google.com/file/d/1cxAhnbx_uZrZpaEPEibTPuHlvj-2xUl6/view

Ahora, basado en lo observado en el video y lo experimentado en clase, construiremos la correspondiente tabla de transición y tabla de salida. Es importante tener en cuenta que la entrada **E0** se vincula al interruptor **J15**, que es el switch ubicado en el extremo derecho, y la salida **Z** se vincula al LED **H17**.

	E0=0	E0=1
Q0	Q0	Q1
Q1	Q2	Q1
Q2	Q0	Q3
Q3	Q2	Q4
Q4	Q2	Q5
Q5	Q6	Q1
Q6	Q0	Q7
Q7	Q2	Q4

Cuadro 5: Tabla de transición experimental.

	Z
Q0	0
Q1	0
Q2	0
Q3	0
Q4	0
Q5	0
Q6	0
Q7	1

Cuadro 6: Tabla de salida experimental.

7. Comparación de resultados

Ahora procederemos a hacer una comparación entre los valores teóricos definidos en el procedimiento teórico, y los valores experimentales obtenidos en el experimento.

	Exp	Teó	Exp	Teó
	E0=0	X=0	E0=1	X=1
Q0	Q0	Q0	Q1	Q1
Q1	Q2	Q2	Q1	Q1
Q2	Q0	Q0	Q3	Q3
Q3	Q2	Q2	Q4	Q4
Q4	Q2	Q2	Q5	Q5
Q5	Q6	Q6	Q1	Q1
Q6	Q0	Q0	Q7	Q7
Q7	Q2	Q2	Q4	Q4

Cuadro 7: Tabla de transición comparativa.

	Exp	Teó
	Z	Y
Q0	0	0
Q1	0	0
Q2	0	0
Q3	0	0
Q4	0	0
Q5	0	0
Q6	0	0
Q7	1	1

Cuadro 8: Tabla de salida comparativa.

Analizando las tablas de comparación, podremos ver que los resultados obtenidos experimentalmente son iguales a los valores teóricos, gracias a esto existirá un 0 % de error porcentual.

8. Conclusiones

En el desarrollo de la actividad se lograron los objetivos propuestos, además de enlazar los conocimientos teóricos aprendidos en clase, como el uso de diagramas de Moore y máquinas de estados, con la experiencia práctica de laboratorio.

A diferencia de los circuitos combinacionales, los circuitos secuenciales presentan una mayor complejidad debido a la incorporación de memoria en sus componentes. Esta memoria permite la transición entre diferentes estados, dependiendo del estado actual del sistema y de la siguiente señal de entrada que influye en él.

Por esta razón, diseñar un circuito secuencial con las entradas y salidas deseadas implica una mayor complejidad en comparación con un circuito combinacional. Sin embargo, esta tarea se puede

simplificar gracias al uso de software como *Boole*, que no solo proporciona la ecuación del circuito, sino que también ofrece una representación gráfica de este con las conexiones correspondientes, permitiendo la elección de diferentes flip-flops según sea necesario.

A pesar de las ventajas, no se puede confiar completamente en este tipo de software, como es el caso de *Boole*. Aunque generó un código VHDL mayormente acertado, se presentaron errores al ejecutarlo en *vivado*. Esto se debe a que, aunque ambos programas utilizan un lenguaje común, están diseñados de manera diferente. Por lo tanto, es necesario interpretar y ajustar el código generado para asegurar que funcione según lo deseado en el entorno específico de *vivado*, en conjunto con las entradas y salidas porteadas con el hardware.

Otro problema que se presentó fue al intentar utilizar la señal de un pulso de los botones del hardware para simular el cambio de estado del reloj del sistema. A pesar de asignar la variable correspondiente al botón del hardware, surgieron problemas al compilar. Para solucionar esto, fue necesario deshabilitar el reloj interno del hardware en el código 2. Este tipo de errores son difíciles de interpretar, ya que el software solo indicó un problema con el “clock”, sin proporcionar detalles adicionales, o una sugerencia para solucionarlo. Una vez comprendido y solucionado este problema, la implementación se facilitó, permitiendo replicar este tipo de código de circuito secuencial con mayor facilidad a posterioridad.

9. Referencias

- [1] AMD. *Vivado™ ML*. URL: <https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html>.
- [2] Digilent. *Nexys 4 DDR Reference Manual*. URL: <https://digilent.com/reference/programmable-logic/nexys-4-ddr/reference-manual>.
- [3] *Logisim-Evolution Github*. URL: <https://github.com/logisim-evolution/logisim-evolution>.
- [4] M. Morris Mano. *Diseño Digital 3ra Edición*. Pearson Education, 2003.
- [5] William Stallings. *Computer Organization And Architecture Designing For Performance - Eight Edition*. Prentice Hall, 2010.