



INFORME DEL LABORATORIO 3

Vazquez Leonardo David, (vazquezleonardodavid@outlook.com)

TEMA: Dispositivos lógico-Programables
ASIGNATURA: Técnicas y Dispositivos Digitales II
ÁREA: Digitales

Departamento de Ingeniería Electrónica y Computación

Facultad de Ingeniería

Universidad Nacional de Mar del Plata

Fecha de entrega: 15/12/2020

ÍNDICE

Resumen.....	3
Objetivos Generales	3
Cuestiones Preliminares	4
Materiales e Instrumental	6
1. PARTE A: Implementación de circuito combinacional completo en FPGA en VHDL	7
2. PARTE B: Implementación de un sumador completo en FPGA en VHDL	12
3. PARTE C: Implementación de un sumador completo en FPGA empleando entorno esquemático	18
4. PARTE D: Implementación de máquinas de estado	25
Conclusiones generales.....	33
Referencias.....	33

Resumen

En el presente trabajo, se aborda como temática principal los dispositivos lógico-programables. La práctica de laboratorio se divide en cuatro partes. Tanto en la primera como en la segunda parte, se desea implementar dos circuitos combinacionales en FPGA mediante el entorno del programa de simulación Quartus II. Ya en la tercera parte, se realiza un circuito sumador completo empleando un entorno esquemático del simulador. Finalmente, en la cuarta parte, se desea implementar dos circuitos máquinas de estado para generar sus respectivos archivos VHDL.

Objetivos Generales

- Introducir el empleo de programación digital en FPGA.
- Realizar descripciones en VHDL de distintos circuitos lógicos.
- Desarrollar habilidades de escritura de reportes de resultados e informes técnicos.
- Desarrollar habilidades para el desarrollo de criterios y evaluación en la solución de problemas.

Cuestiones Preliminares

Cyclone III [1] es una familia de dispositivos de pruebas experimentales que contienen una FPGA y que ofrecen una combinación de alta funcionalidad, bajo consumo de potencia y bajo costo.

La arquitectura de la familia de dispositivos Cyclone III posee como la unidad lógica más pequeña a los Elementos Lógicos (LE). Cada LE, básicamente, posee 4 entradas, 4 entradas look-up table, un registro y una salida lógica. Las entradas look-up-table se utilizan para implementar cualquier función de cuatro variables. En la figura 1 se observa un esquemático completo de un LE.

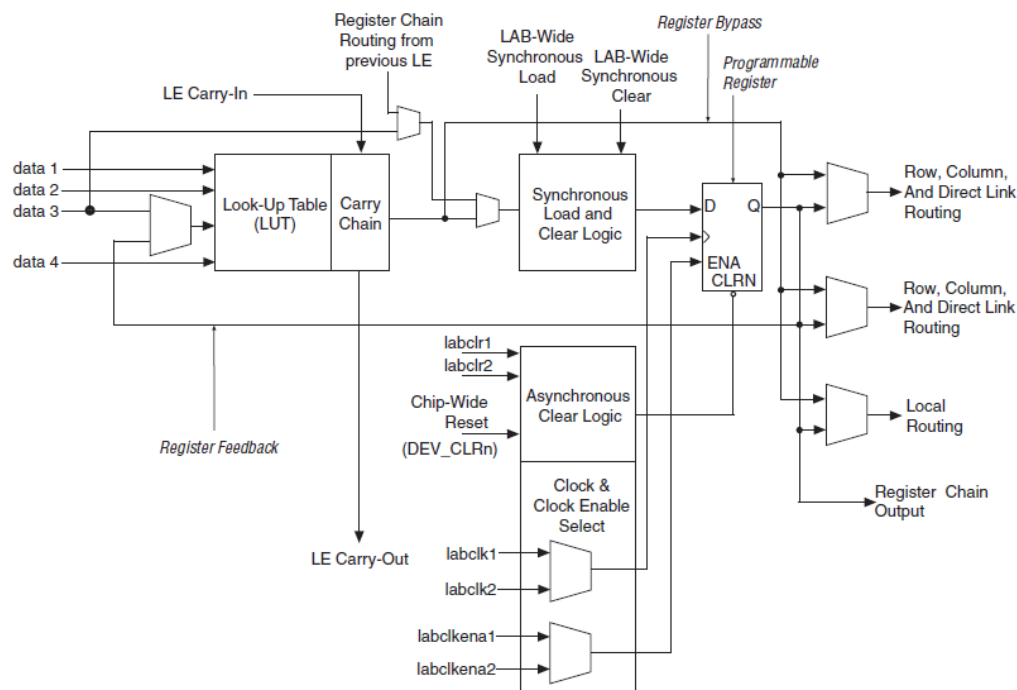


Figura 1. Esquemático completo de un LE.

Un bloque de arreglos lógicos (LAB) contiene varios LE's en conjunto, además de señales de control. La estructura general de un LAB consta de interconexiones locales entre las LE's que la conforman, para mejorar la eficiencia en el rendimiento y el área utilizada. En la figura 2 se observa la estructura general de los LAB's.

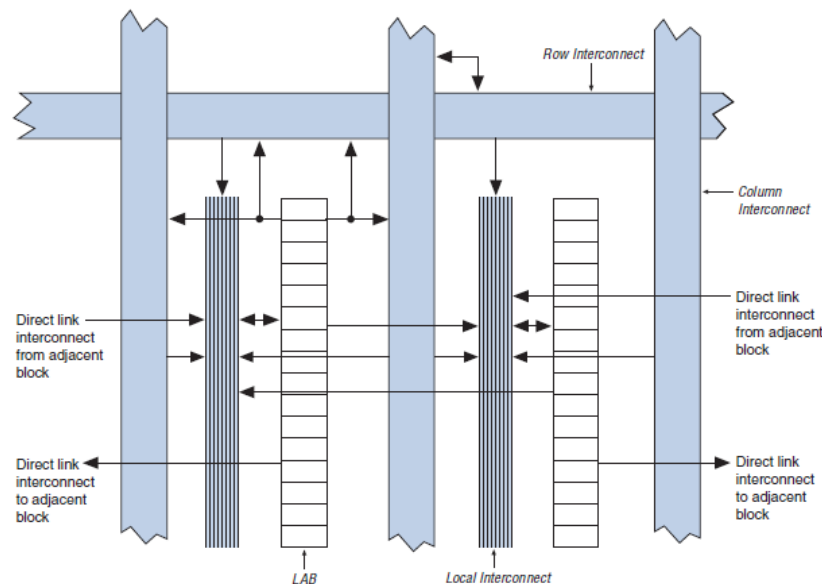
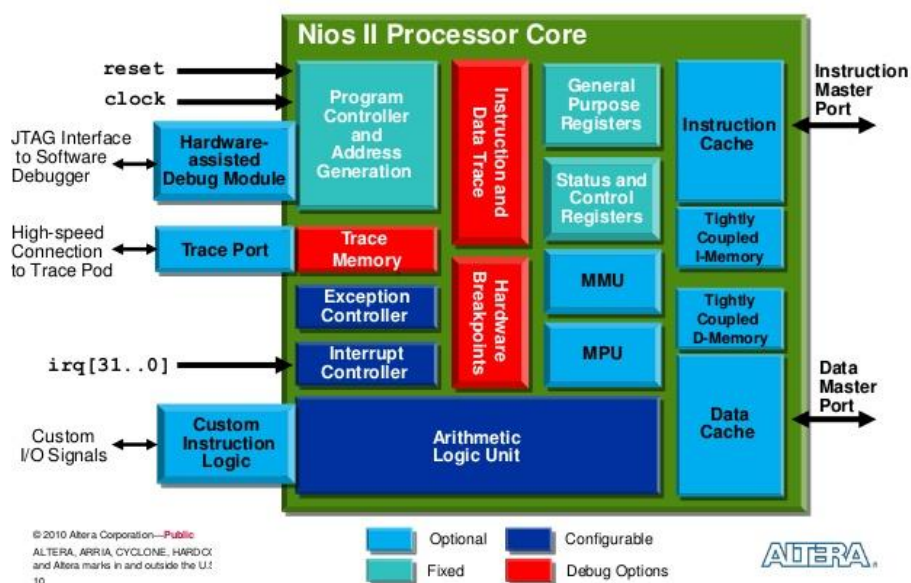


Figura 2. Estructura de los LAB's.

Algunos chips FPGA de la familia de dispositivos Cyclone III utilizan como arquitectura de procesador a NIOS[®] II [2]. En la figura 3 se observa el diagrama en bloques correspondiente.

Nios II Processor Configuration

Figura 3. Configuración de la arquitectura de procesador NIOS[®]II.

Las FPGA disponen en su estructura a bloques de unidades lógicas configurables llamadas 'IP Cores'. Estas unidades son prácticamente flexibles ya que se pueden configurar sus conexiones internas. La desventaja del uso de las 'IP Cores' es el desaprovechamiento en cuanto a espacio de silicio, debido a la no conexión de ciertas compuertas lógicas. En cambio, los sistemas embebidos son diseñados para cierta aplicación en específico (por ejemplo, multiplicadores 18x18 bits), aumentando así el rendimiento del bloque a utilizar debido al bajo consumo de potencia y menor utilización de área en cuanto a espacio del chip.

Materiales e Instrumental

- Software de simulación QUARTUS II[3].
- Datasheet Cyclone III: "Cyclone III Devide Handbook".
- PC.



Figura 4. Cyclone III.

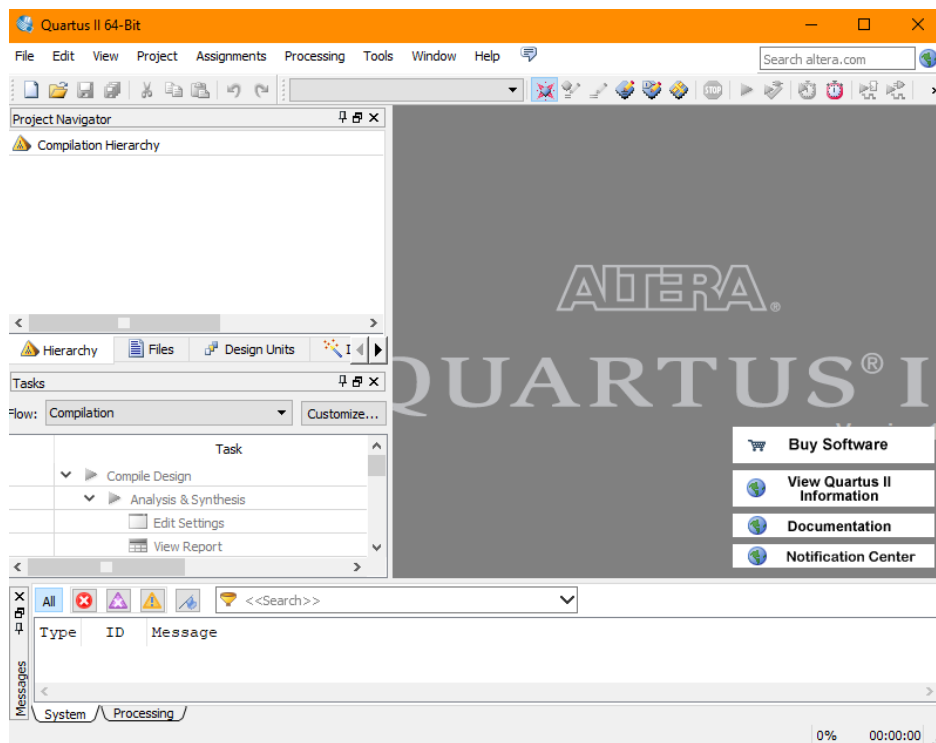


Figura 5. Interfaz Quartus II.

1. PARTE A: Implementación de circuito combinacional completo en FPGA en VHDL

Se desea implementar el siguiente circuito combinacional:



Figura 6. Circuito combinacional a implementar.

Para ello, se crea un nuevo proyecto de Cyclone III en Quartus II y se genera un archivo VHDL. En tal archivo se describe el circuito combinacional propuesto, definiendo las librerías a utilizar y declarando la entidad (definición de entradas y salidas) y la arquitectura (definiendo la función lógica a realizar):

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY PARTEA is
5  Port ( SW1 : in STD_LOGIC;
6        SW2 : in STD_LOGIC;
7        LED : out STD_LOGIC);
8  end PARTEA;
9
10 architecture COMPORTAMIENTO of PARTEA is
11
12 begin
13
14     LED <= NOT (SW1 and SW2);
15
16 end COMPORTAMIENTO;
  
```

Figura 7. Archivo VHDL del circuito combinacional.

Una vez que se compila el archivo, se recurre a la hoja de datos del dispositivo Cyclone III y se asignan los pines de entrada y salida del circuito:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard
out LED	Output	PIN_AG19	4	B4_N2	PIN_AG19	2.5 V (default)
in SW1	Input	PIN_AD18	4	B4_N1	PIN_AD18	2.5 V (default)
in SW2	Input	PIN_AD14	3	B3_N0	PIN_AD14	2.5 V (default)
<<new node>>						

Message: Quartus II 64-Bit Netlist Viewers Preprocess was successful. 0 errors, 0 warnir

Figura 8. Asignación de pines.

Luego, se verifica el circuito implementado mediante el visor RTL, y los mapeos post-Mapping y post-Fitting.

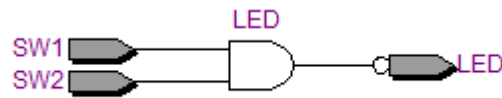


Figura 9. Visor RTL.

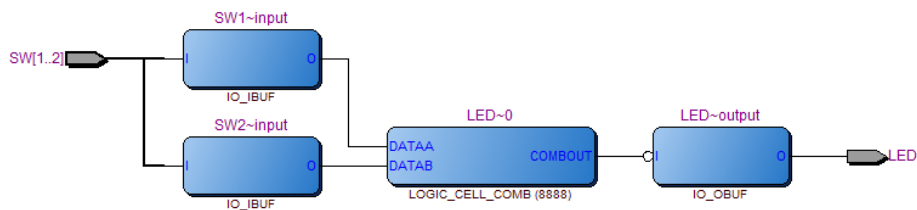


Figura 10. Post-Mapping.

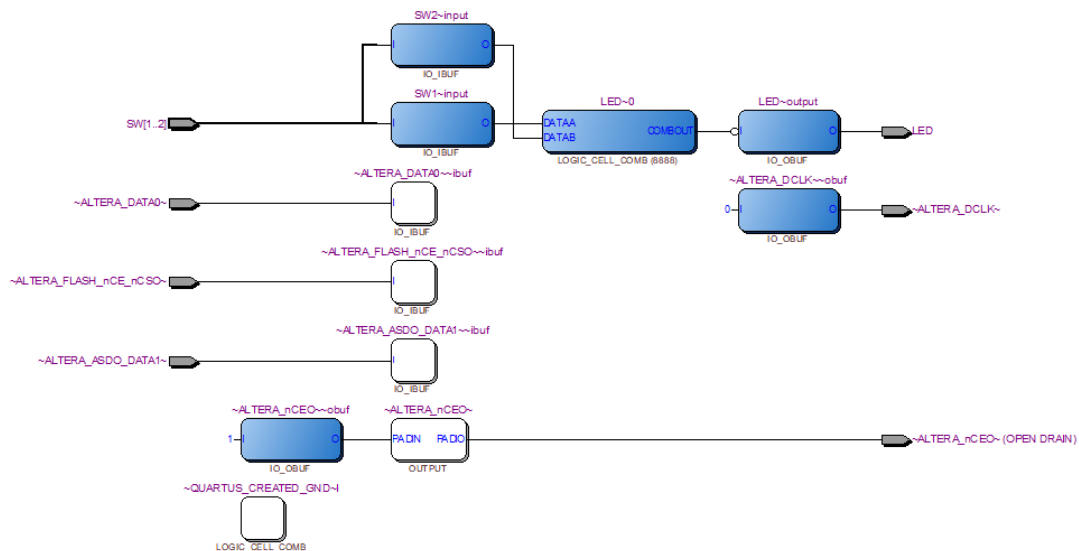


Figura 11. Post-Fitting.

Se verifica entonces la implementación del archivo VHDL en el chip FPGA:

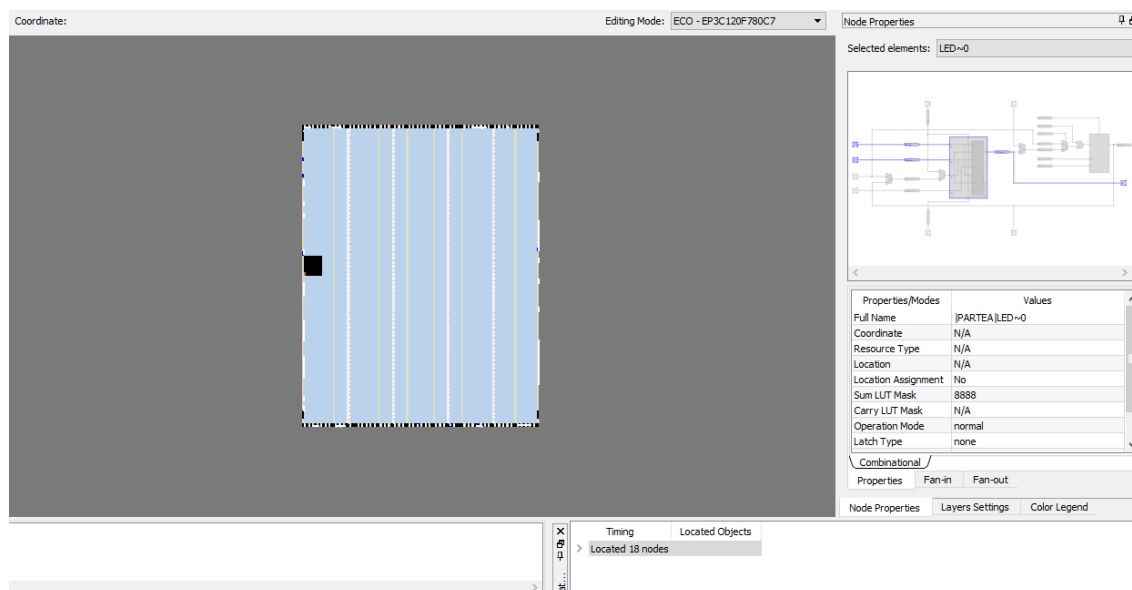


Figura 12. Implementación del archivo VHDL en el chip FPGA.

Mediante el simulador de Quartus II se realiza la simulación temporal y funcional del circuito, asignando valores arbitrarios a las entradas para observar la salida:

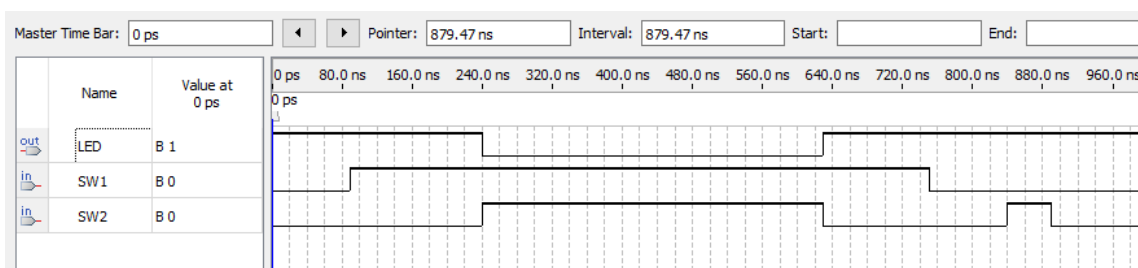


Figura 13. Simulación en Quartus II.

Para realizar la simulación en el programa ModelSim (Incluido en Quartus II), se escribe un archivo testbench VHDL. El mismo consta de la definición de las librerías a utilizar, la declaración de la entidad del archivo de prueba, la declaración de la componente del archivo VHDL correspondiente al circuito combinacional, la definición de las señales de entrada y salida, y finalmente el proceso de estímulo el cual se encarga de asignarle valores a las señales de entrada cada 10 ns.

```

ENTITY testbench is
    end testbench;

architecture comportamiento of PARTEATEST is

    component PARTEA --declaracion de componente
    port (
        SW1 : IN STD_LOGIC;
        SW2 : IN STD_LOGIC;
        LED : OUT STD_LOGIC
    );
end component;

--ENTRADAS
signal SW1 : STD_LOGIC := '0';
signal SW2 : STD_LOGIC := '0';
--SALIDA
signal LED : STD_LOGIC;

BEGIN
    --UNIT UNDER TEST (UUI)
    uut: PARTEA port map (
        SW1 => SW1,
        SW2 => SW2,
        LED => LED );

    STIM:PROC: PROCESS --STIMULUS PROCESS
        SW1 <= '0' ; SW2 <= '0'; wait for 10ns;
        SW1 <= '0' ; SW2 <= '1'; wait for 10ns;
        SW1 <= '1' ; SW2 <= '0'; wait for 10ns;
        SW1 <= '1' ; SW2 <= '1'; wait for 10ns;
        wait;
    end process;

END;

```

Figura 14. Archivo VHDL del testbench.

En la figura 15 se observa la simulación en Modelsim:

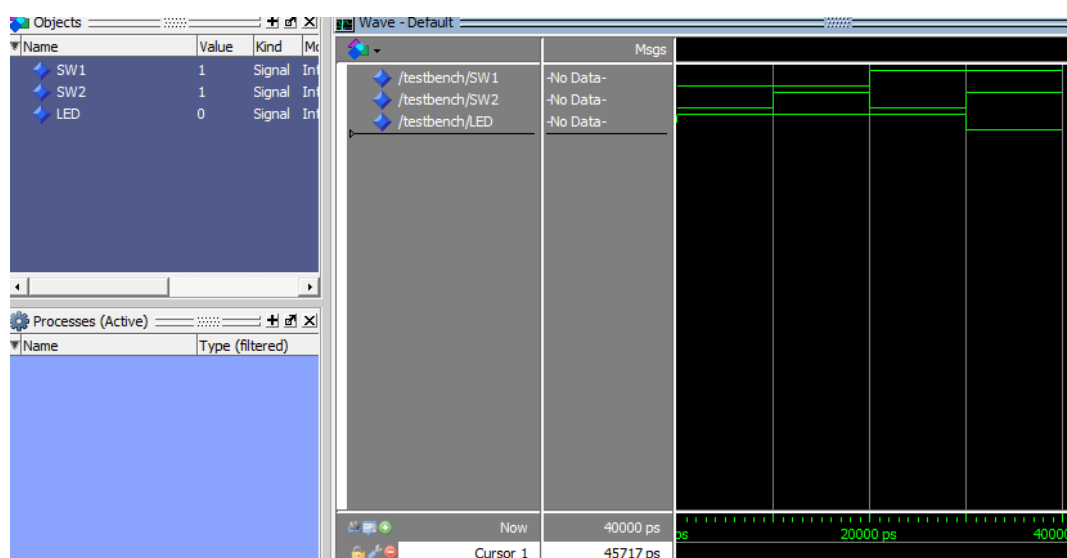


Figura 15. Simulación en Modelsim.

Además, se realiza el esquemático del circuito combinacional, con sus respectivas asignaciones de pines:

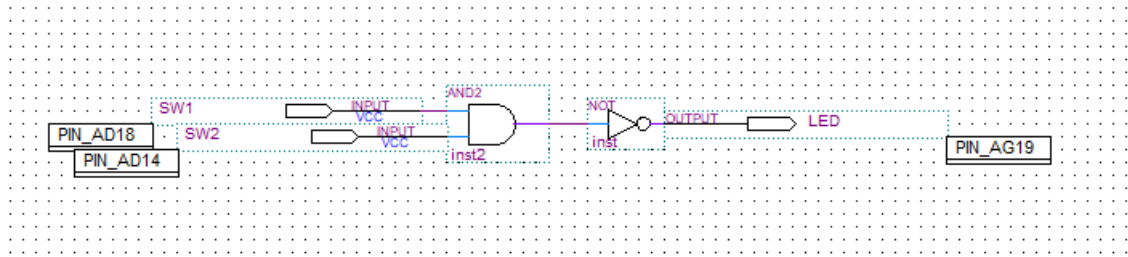


Figura 16. Esquemático del circuito combinacional propuesto.

2. PARTE B: Implementación de un sumador completo en FPGA en VHDL

Se desea implementar un sumador completo en el cual se lathce las señales de entrada y salida mediante Flip-Flop's D.

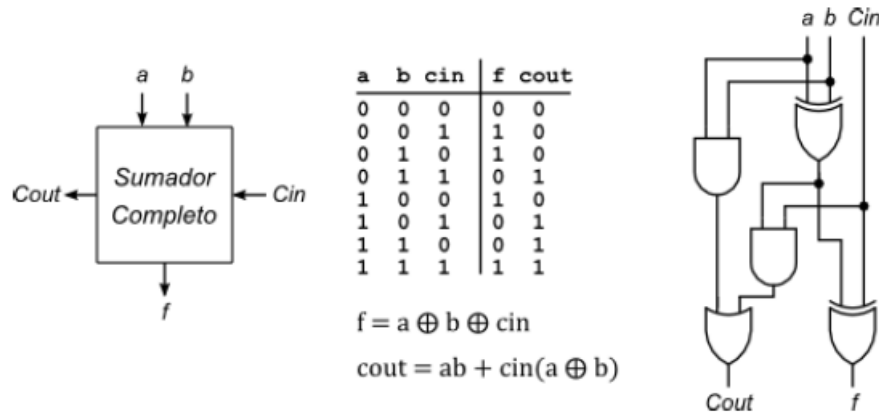


Figura 17. Sumador completo.

Para ello, se crea un nuevo proyecto de Cyclone III en Quartus II y se genera un archivo VHDL. En tal archivo se describe el funcionamiento del Flip-Flop D, definiendo las librerías a utilizar y declarando la entidad (definición de entradas y salidas) y la arquitectura (definiendo la función lógica a realizar mediante el proceso de clock):

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity FLIPFLOPD is
port ( D, clock: in std_logic;
      Q: out std_logic);
end FLIPFLOPD;

architecture behavior of FLIPFLOPD is
begin
process(clock)
begin
if (clock='1' and clock'EVENT) then
Q <= D;
end if;
end process;
end behavior;
```

Figura 18. Archivo VHDL del Flip-Flop D.

Una vez que se compila, se genera un nuevo archivo VHDL. En tal archivo se describe el funcionamiento del sumador completo, definiendo las librerías a utilizar y declarando la entidad (definición de entradas y salidas) y la arquitectura, el cual se utiliza la componente del archivo VHDL del Flip-Flop D para lathcear las señales de entrada y salida para luego declarar la función lógica a realizar.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sumador_completo is
port ( Ain : in STD_LOGIC;
      Bin : in STD_LOGIC;
      cinin : in STD_LOGIC;
      clk : in STD_LOGIC;
      Fout : out STD_LOGIC;
      coutout : out STD_LOGIC);
end sumador_completo;

architecture behavior of sumador_completo is

component FLIPFLOPD
port (D, clock : in std_logic;
      Q: out std_logic);
end component;

--entradas
signal A,B,cin : std_logic;
--salidas
signal F,cout : std_logic;

begin

    F <= A xor B xor cin ;
    cout <= (A and B) or (cin and (A xor B));

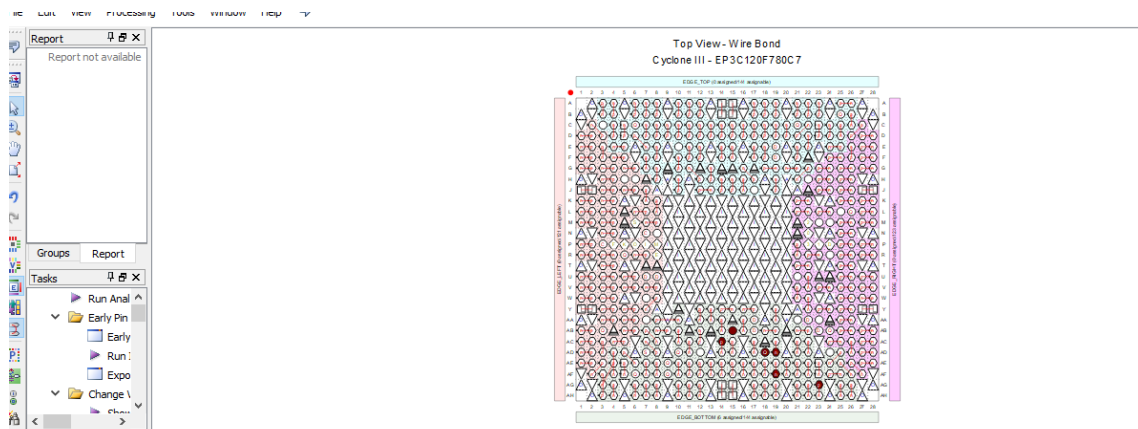
    D1: FLIPFLOPD port map (Ain, clk, A);
    D2: FLIPFLOPD port map (Bin, clk, B);
    D3: FLIPFLOPD port map (cinin, clk, cin);
    D4: FLIPFLOPD port map (cout, clk, coutout);
    D5: FLIPFLOPD port map (F, clk, Fout);

end behavior;

```

Figura 19. Archivo VHDL del sumador completo.

Una vez que se compila el archivo, se recurre a la hoja de datos del dispositivo Cyclone III y se asignan los pines de entrada y salida del circuito:



Top View - Wire Bond
Cyclone III - EP3C120F780C7

Node Name	Direction	Location	I/O Bank	VREF Group	Filter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair
Ain	Input	PIN_AC14	3	B3_N0	PIN_AC14	2.5 V (default)		8mA (default)		
Bin	Input	PIN_AD18	4	B4_N1	PIN_AD18	2.5 V (default)		8mA (default)		
cinin	Input	PIN_AG23	4	B4_N1	PIN_AG23	2.5 V (default)		8mA (default)		
clk	Input	PIN_AB15	4	B4_N2	PIN_AH14	2.5 V (default)		8mA (default)		
coutout	Output	PIN_AF19	4	B4_N1	PIN_AF19	2.5 V (default)		8mA (default)	2 (default)	
Fout	Output	PIN_AD19	4	B4_N0	PIN_AD19	2.5 V (default)		8mA (default)	2 (default)	

Figura 20. Asignación de pines.

Luego, se verifica el circuito implementado mediante el visor RTL, y los mapeos post-Mapping y post-Fitting.

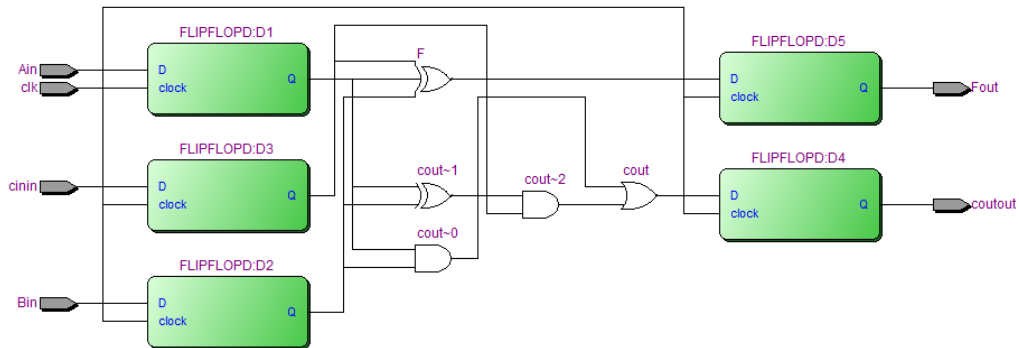


Figura 21. Visor RTL.

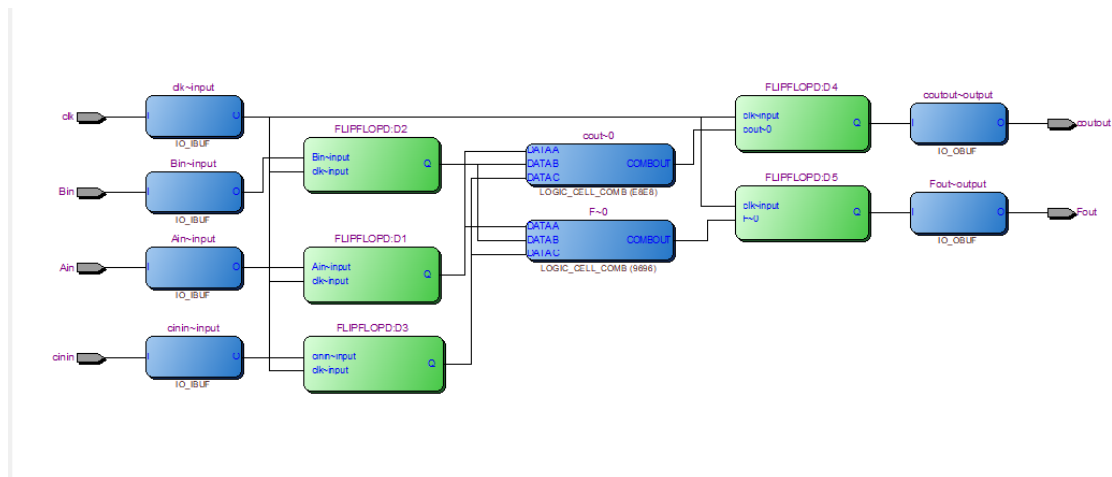


Figura 22. Post-Mapping.

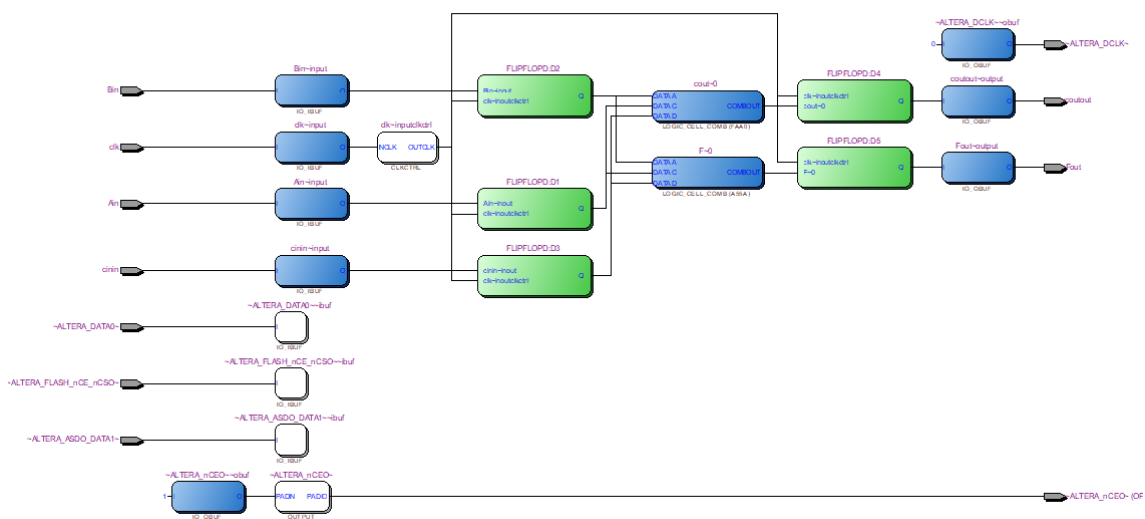


Figura 23. Post-Fitting.

Se verifica entonces la implementación del archivo VHDL en el chip FPGA:

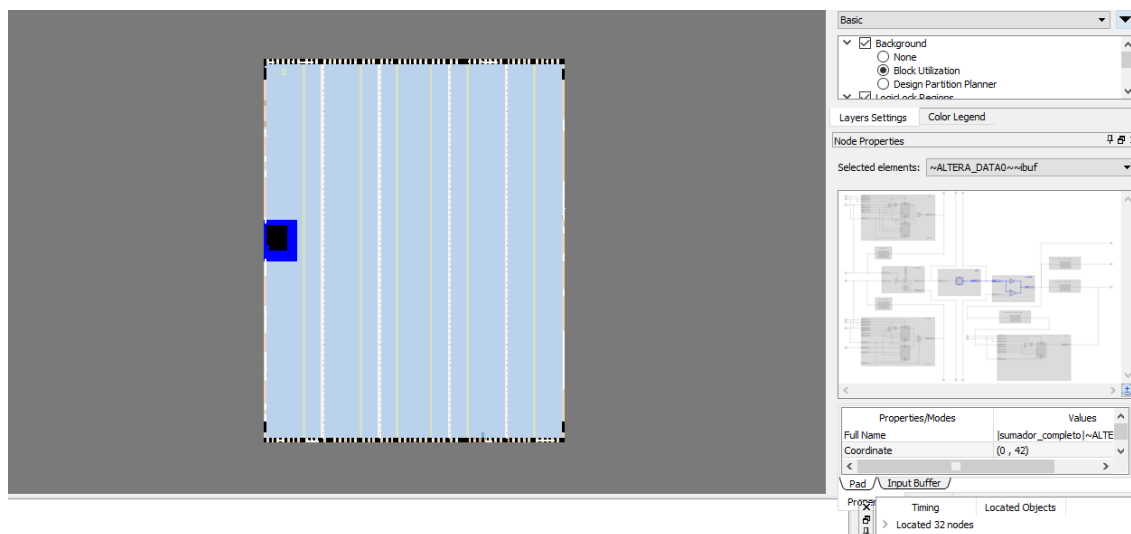


Figura 24. Implementación del archivo VHDL en el chip FPGA.

Mediante el simulador de Quartus II se realiza la simulación temporal y funcional del circuito, asignando valores arbitrarios a las entradas para observar la salida:

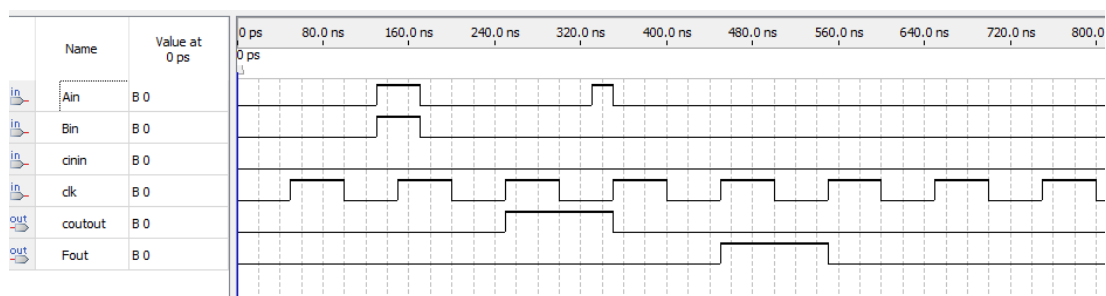


Figura 25. Simulación en Quartus II.

Para realizar la simulación en el programa ModelSim (Incluido en Quartus II), se escribe un archivo testbench VHDL. El mismo consta de la definición de las librerías a utilizar, la declaración de la entidad del archivo de prueba, la declaración de la componente del archivo VHDL correspondiente al circuito sumador completo, la definición del periodo del clock y las definiciones de las señales de entrada y salida, y finalmente el proceso del clock y del estímulo; el cual se encarga de asignarle valores a las señales de entrada cada 30 ns.

```

library ieee;
USE ieee.std_logic_1164.ALL;

entity sumador_completo_testbench is
end sumador_completo_testbench;

architecture behavior of sumador_completo_testbench is

component sumador_completo --declaracion de componente
port ( Ain : in STD_LOGIC;
      Bin : in STD_LOGIC;
      cinin : in STD_LOGIC;
      clk : in STD_LOGIC;
      Fout : out STD_LOGIC;
      coutout : out STD_LOGIC);
end component;

--entradas
SIGNAL Ain : std_logic:= '0';
SIGNAL Bin : std_logic:= '0';
SIGNAL cinin : std_logic:= '0';
SIGNAL clk : std_logic:= '0';

--salidas
SIGNAL Fout : std_logic;
SIGNAL coutout : std_logic;

    --definition de periodo de clock

    constant clock_period : time := 20ns;

begin
    --UNIT UNDER TEST (UUT)

    uut: sumador_completo port map (
        Ain => Ain,
        Bin => Bin,
        cinin => cinin,
        clk => clk,
        Fout => Fout,
        coutout => coutout
    );

    --definiciones de procesos de clock
    clock_process: process
    begin
        clk<='0';
        wait for clock_period/2;
        clk<='1';
        wait for clock_period/2;
    end process;

```

Figura 26a. Archivo VHDL del testbench.


```

stim_proc: process --stimulus process

begin
--stimulus

    Ain <='0'; Bin <= '0' ; cinin <= '0';wait for 30ns;
    Ain <='0'; Bin <= '0' ; cinin <= '1';wait for 30ns;
    Ain <='0'; Bin <= '1' ; cinin <= '0';wait for 30ns;
    Ain <='0'; Bin <= '1' ; cinin <= '1';wait for 30ns;
    Ain <='1'; Bin <= '0' ; cinin <= '0';wait for 30ns;
    Ain <='1'; Bin <= '0' ; cinin <= '1';wait for 30ns;
    Ain <='1'; Bin <= '1' ; cinin <= '0';wait for 30ns;
    Ain <='1'; Bin <= '1' ; cinin <= '1';wait for 30ns;
    wait;

end process;

END;

```

Figura 26b. Archivo VHDL del testbench.

En la figura 27 se observa la simulación en Modelsim:

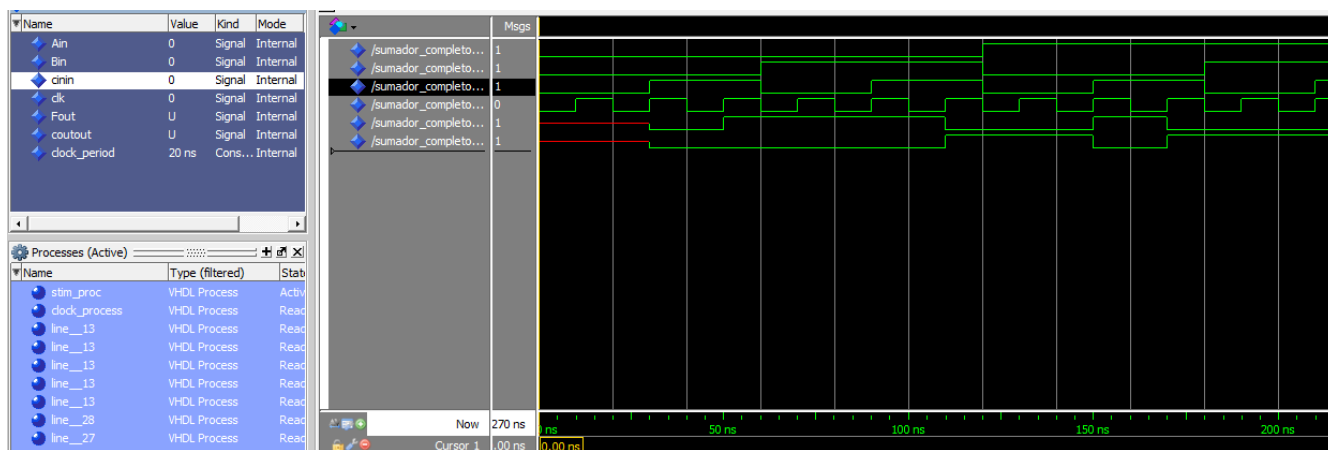


Figura 27. Simulación en Modelsim.

3. PARTE C: Implementación de un sumador completo en FPGA empleando entorno esquemático

Se desea implementar un sumador completo en el cual se lathce las señales de entrada y salida mediante Flip-Flop's D. Para ello se utiliza un entorno esquemático del simulador Quartus II, agregando los símbolos de los componentes necesarios. En la figura 28 se observa la implementación del circuito propuesto:

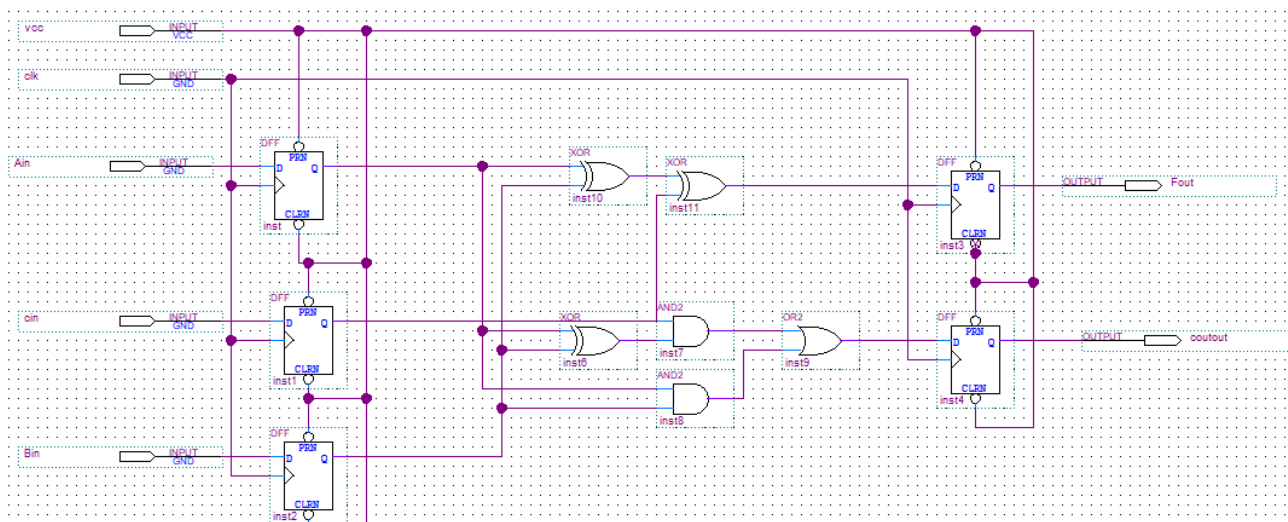


Figura 28. Circuito esquemático del sumador completo.

Una vez que se compila el archivo, se recurre a la hoja de datos del dispositivo Cyclone III y se asignan los pines de entrada y salida del circuito:

Report

Report not available

Groups Report

Tasks

- Run Anal
- Early Pin
- Early
- Run
- Expo
- Change

Named: *

Top View - Wire Bond

Cyclone III - EP3C120F780C7

Pin Assignment Table:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair
AIN	Input	PIN_AC14	3	B3_N0	PIN_L7	2.5 V (default)		8mA (default)		
BIN	Input	PIN_AD18	4	B4_N1	PIN_N3	2.5 V (default)		8mA (default)		
cin	Input	PIN_AG23	4	B4_N1	PIN_N4	2.5 V (default)		8mA (default)		
cout	Output	PIN_AC19	4	B4_N0	PIN_J2	2.5 V (default)		8mA (default)		
coutout	Output	PIN_AF19	4	B4_N1	PIN_L6	2.5 V (default)		8mA (default)	2 (default)	
Fout	Output	PIN_AG19	4	B4_N2	PIN_M5	2.5 V (default)		8mA (default)	2 (default)	
vcc	Input	PIN_AF25	4	B4_N1	PIN_J1	2.5 V (default)		8mA (default)		
<new node>										

Figura 29. Asignación de pines.

Luego, se verifica el circuito implementado mediante el visor RTL, y los mapeos post-Mapping y post-Fitting.

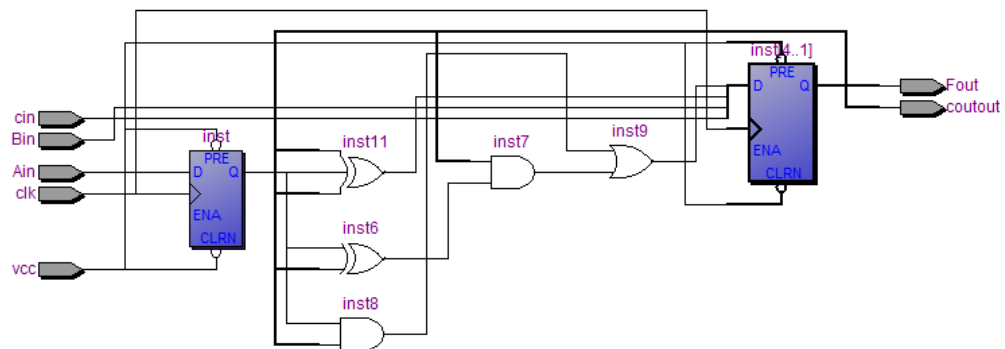


Figura 30. Visor RTL.

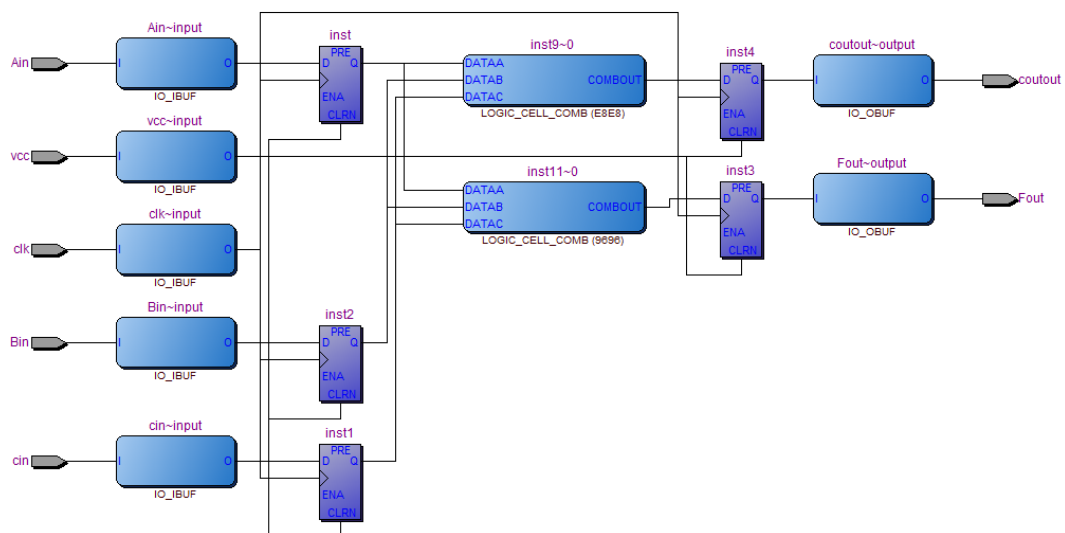


Figura 31. Post-Mapping.

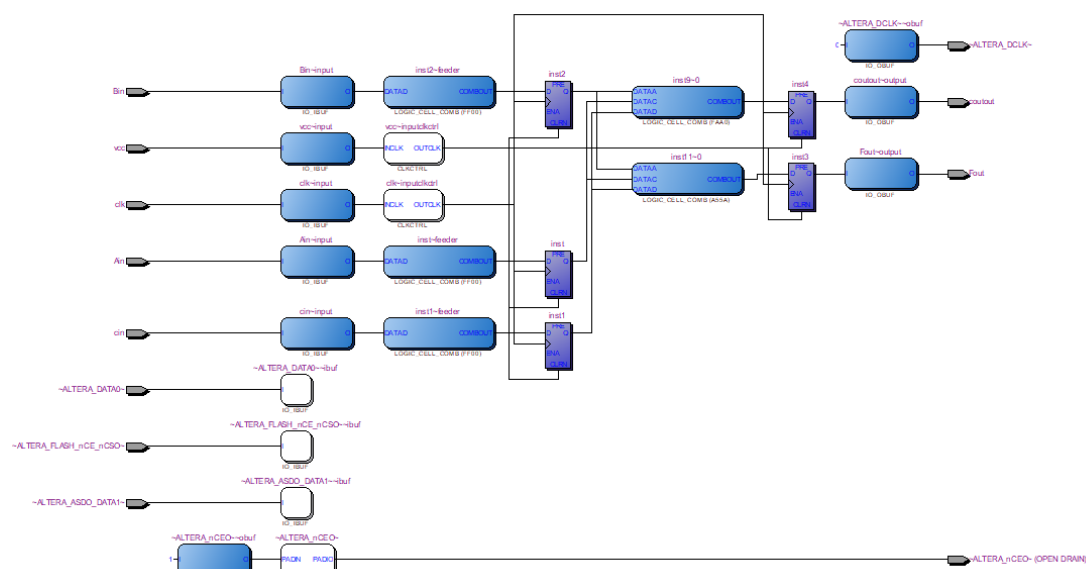


Figura 32. Post-Fitting.

Se verifica entonces la implementación del archivo VHDL en el chip FPGA:

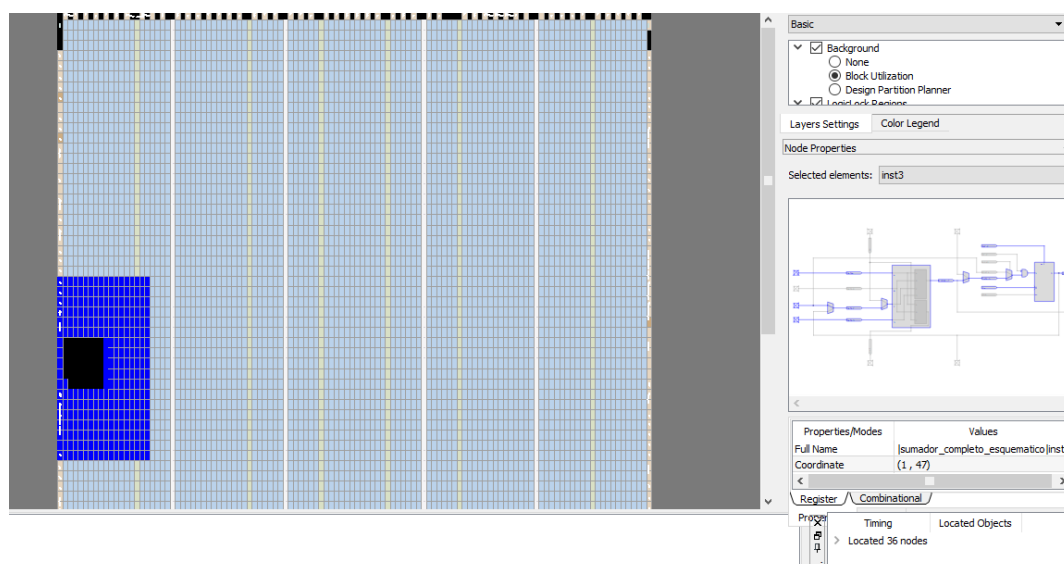


Figura 33. Implementación del archivo VHDL en el chip FPGA.

Mediante el simulador de Quartus II se realiza la simulación temporal y funcional del circuito, asignando valores arbitrarios a las entradas para observar la salida:

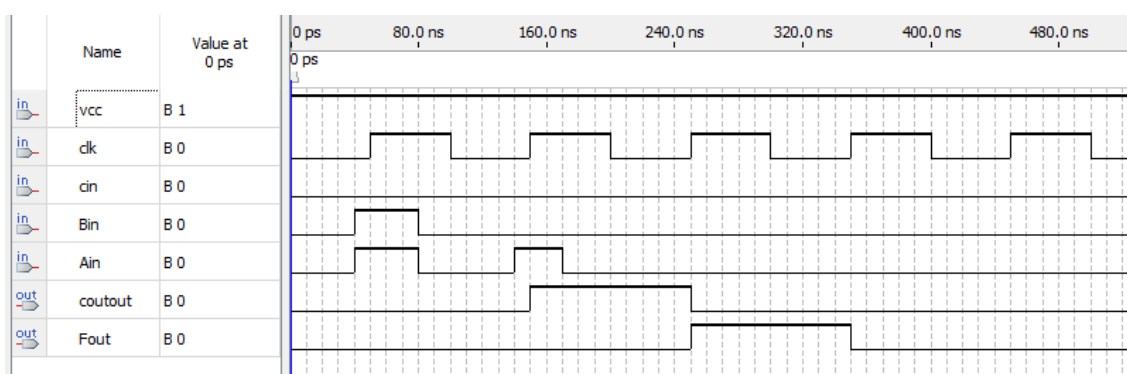


Figura 34. Simulación en Quartus II.

Para realizar la simulación en el programa ModelSim, se convierte el archivo esquemático a un archivo VHDL que sea reconocible por ModelSim:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY work;

]ENTITY sumador_completo_esquemático IS
  PORT
]  (
    cin : IN STD_LOGIC;
    Bin : IN STD_LOGIC;
    Ain : IN STD_LOGIC;
    clk : IN STD_LOGIC;
    vcc : IN STD_LOGIC;
    Fout : OUT STD_LOGIC;
    coutout : OUT STD_LOGIC
  );
END sumador_completo_esquemático;

ARCHITECTURE bdf_type OF sumador_completo_esquemático IS

SIGNAL SYNTHESIZED_WIRE_6 : STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_7 : STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_0 : STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_8 : STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_1 : STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_2 : STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_3 : STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_4 : STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_5 : STD_LOGIC;

BEGIN

PROCESS (clk, vcc, vcc)
BEGIN
IF (vcc = '0') THEN
  SYNTHESIZED_WIRE_6 <= '0';
ELSIF (vcc = '0') THEN
  SYNTHESIZED_WIRE_6 <= '1';
ELSIF (RISING_EDGE(clk)) THEN
  SYNTHESIZED_WIRE_6 <= Ain;
END IF;
END PROCESS;

PROCESS (clk, vcc, vcc)
BEGIN
IF (vcc = '0') THEN
  SYNTHESIZED_WIRE_8 <= '0';
ELSIF (vcc = '0') THEN
  SYNTHESIZED_WIRE_8 <= '1';
ELSIF (RISING_EDGE(clk)) THEN
  SYNTHESIZED_WIRE_8 <= cin;
END IF;
END PROCESS;

```

Figura 35a. Archivo VHDL correspondiente al esquemático.

```

SYNTHESIZED_WIRE_0 <= SYNTHESIZED_WIRE_6 XOR SYNTHESIZED_WIRE_7;

SYNTHESIZED_WIRE_1 <= SYNTHESIZED_WIRE_0 XOR SYNTHESIZED_WIRE_8;

PROCESS (clk, vcc, vcc)
BEGIN
IF (vcc = '0') THEN
    SYNTHESIZED_WIRE_7 <= '0';
ELSIF (vcc = '0') THEN
    SYNTHESIZED_WIRE_7 <= '1';
ELSIF (RISING_EDGE(clk)) THEN
    SYNTHESIZED_WIRE_7 <= Bin;
END IF;
END PROCESS;

    PROCESS (clk, vcc, vcc)
    BEGIN
    IF (vcc = '0') THEN
        Fout <= '0';
    ELSIF (vcc = '0') THEN
        Fout <= '1';
    ELSIF (RISING_EDGE(clk)) THEN
        Fout <= SYNTHESIZED_WIRE_1;
    END IF;
    END PROCESS;

    PROCESS (clk, vcc, vcc)
    BEGIN
    IF (vcc = '0') THEN
        coutout <= '0';
    ELSIF (vcc = '0') THEN
        coutout <= '1';
    ELSIF (RISING_EDGE(clk)) THEN
        coutout <= SYNTHESIZED_WIRE_2;
    END IF;
    END PROCESS;

SYNTHESIZED_WIRE_3 <= SYNTHESIZED_WIRE_6 XOR SYNTHESIZED_WIRE_7;

SYNTHESIZED_WIRE_5 <= SYNTHESIZED_WIRE_8 AND SYNTHESIZED_WIRE_3;

SYNTHESIZED_WIRE_4 <= SYNTHESIZED_WIRE_6 AND SYNTHESIZED_WIRE_7;

SYNTHESIZED_WIRE_2 <= SYNTHESIZED_WIRE_4 OR SYNTHESIZED_WIRE_5;

END bdf_type;

```

Figura 35b. Archivo VHDL correspondiente al esquemático.

Luego, se escribe el archivo testbench VHDL correspondiente. El mismo consta de la definición de las librerías a utilizar, la declaración de la entidad del archivo de prueba, la declaración de la componente del archivo VHDL correspondiente al circuito sumador completo, la definición del periodo del clock y las definiciones de las señales de entrada y salida; y finalmente el proceso del clock y del estímulo, el cual se encarga de asignarle valores a las señales de entrada cada 30 ns.

```
library ieee;
use ieee.std_logic_1164.ALL ;

]entity sumador_completo_esquematico_testbench is
end sumador_completo_esquematico_testbench;

]architecture comportamiento of sumador_completo_esquematico_testbench is

]component sumador_completo_esquematico --component declaration

]Port ( Ain : in STD_LOGIC;
      Bin : in STD_LOGIC;
      cin : in STD_LOGIC;
      clk : in STD_LOGIC;
      vcc : in STD_LOGIC;
      Fout : out STD_LOGIC;
      coutout : out STD_LOGIC);
end component;

      --entradas

      signal Ain : std_logic := '0';
      signal Bin : std_logic := '0';
      signal cin : std_logic := '0';
      signal clk : std_logic := '0';
      signal vcc : std_logic := '1';

      --salidas

      signal Fout : std_logic;
      signal coutout : std_logic;

      --clock period definition

      constant clock_period :time := 20ns;
```

Figura 36a. Archivo VHDL correspondiente al esquemático.

```

BEGIN

--UUT

uut: sumador_completo_esquematico
port map (
    Ain => Ain,
    Bin => Bin,
    cin => cin,
    clk => clk,
    vcc => vcc,
    Fout => Fout,
    coutout => coutout
);

--clock process definitions
clock_process:process
begin
    clk<='0';
    wait for clock_period/2;
    clk<='1';
    wait for clock_period/2;
end process;

stim_proc: process --stimulus process
begin
--stimulus

    Ain  <= '0'; Bin <= '0'; cin <= '0'; vcc <= '1'; wait for 30ns;
    Ain  <= '0'; Bin <= '0'; cin <= '1'; vcc <= '1'; wait for 30ns;
    Ain  <= '0'; Bin <= '1'; cin <= '0'; vcc <= '1'; wait for 30ns;
    Ain  <= '0'; Bin <= '1'; cin <= '1'; vcc <= '1'; wait for 30ns;

    Ain  <= '1'; Bin <= '0'; cin <= '0'; vcc <= '1'; wait for 30ns;
    Ain  <= '1'; Bin <= '0'; cin <= '1'; vcc <= '1'; wait for 30ns;
    Ain  <= '1'; Bin <= '1'; cin <= '0'; vcc <= '1'; wait for 30ns;
    Ain  <= '1'; Bin <= '1'; cin <= '1'; vcc <= '1'; wait for 30ns;
    wait;

end process;

END;

```

Figura 36b. Archivo VHDL correspondiente al esquemático.

En la figura 37 se observa la simulación en Modelsim:

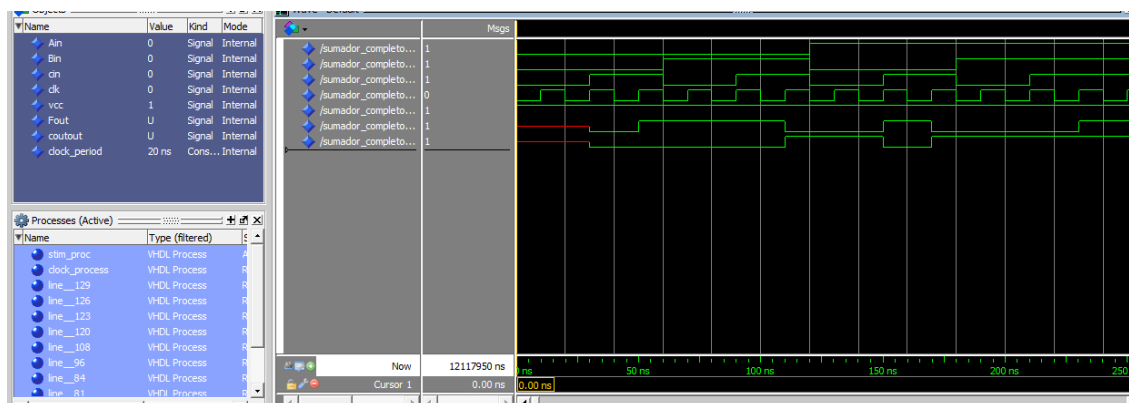


Figura 37. Simulación en Modelsim.

4. PARTE D: Implementación de máquinas de estado

Se desea implementar en un proyecto de Quartus II, dos máquinas de estado referentes a los ejercicios 2 y 4 de la guía de trabajos prácticos N°6, correspondientes a la secuencia de luces y el dado, respectivamente. Para ambas máquinas de estado, se definen, mediante la herramienta "State Machine Wizard", las entradas y salidas, los estados, la transición de los estados, y los valores de las salidas para cada estado:

Input Table		State				
Input Port				Source State	Destination State	Transition
1	reset	1	E1	1	E7	
2	clock	2	E2	2	E1	~x
3	x	3	E3	3	E2	
		4	E4	4	E1	x
		5	E5	5	E3	~x
		6	E6	6	E4	x
		7	E7	7	E5	
				8	E6	
				9	E3	x

Output Port	Output Value	In State	Additional Conditions	Output State
LA	0	E1		Current clock cycle
LA	0	E2		Current clock cycle
LA	1	E3		Current clock cycle
LA	1	E4		Current clock cycle
LA	1	E5		Current clock cycle
LA	1	E6		Current clock cycle
LA	1	E7		Current clock cycle
LB	0	E1		Current clock cycle
LB	1	E2		Current clock cycle

Figura 38. Parámetros de la máquina de estados correspondiente a la secuencia de luces.

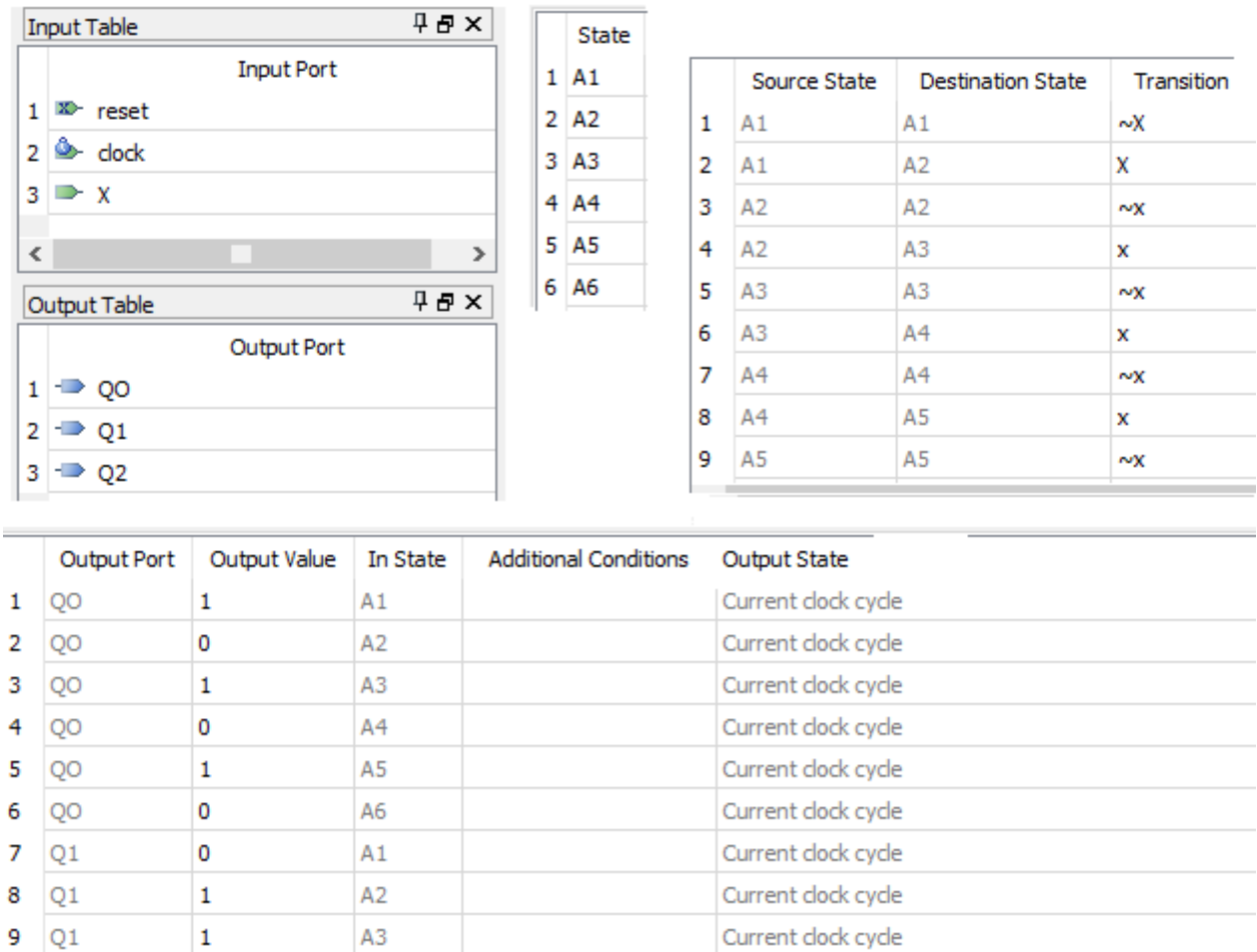


Figura 39. Parámetros de la máquina de estados correspondiente al dado.

De esta forma, se generan automáticamente los diagramas de estados correspondientes:

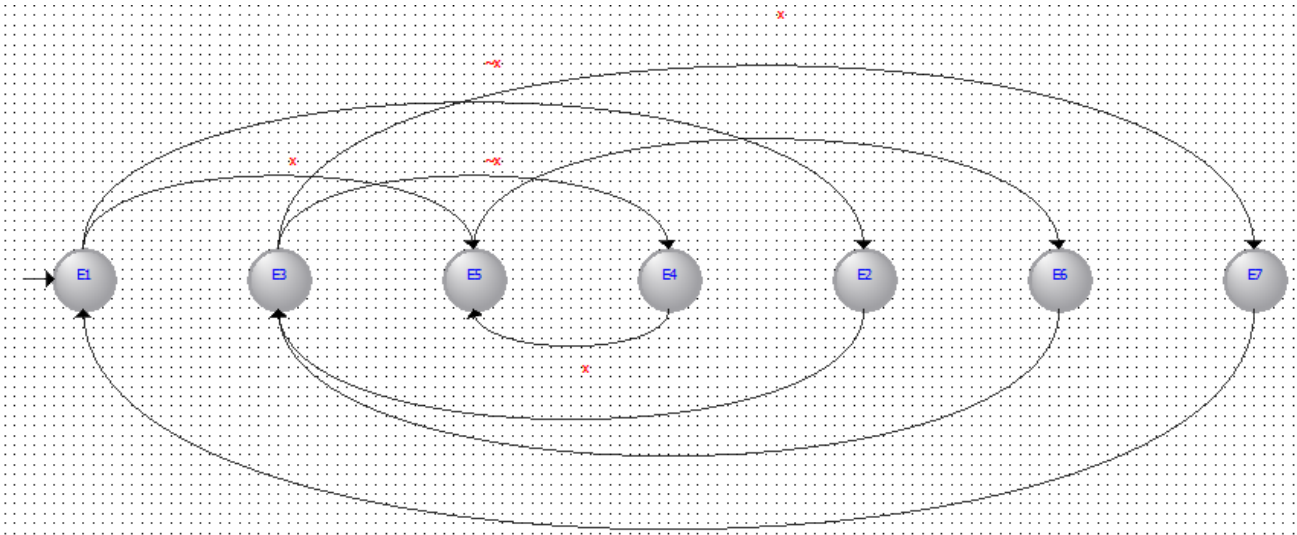


Figura 40. Diagrama de estados correspondiente a la secuencia de luces.

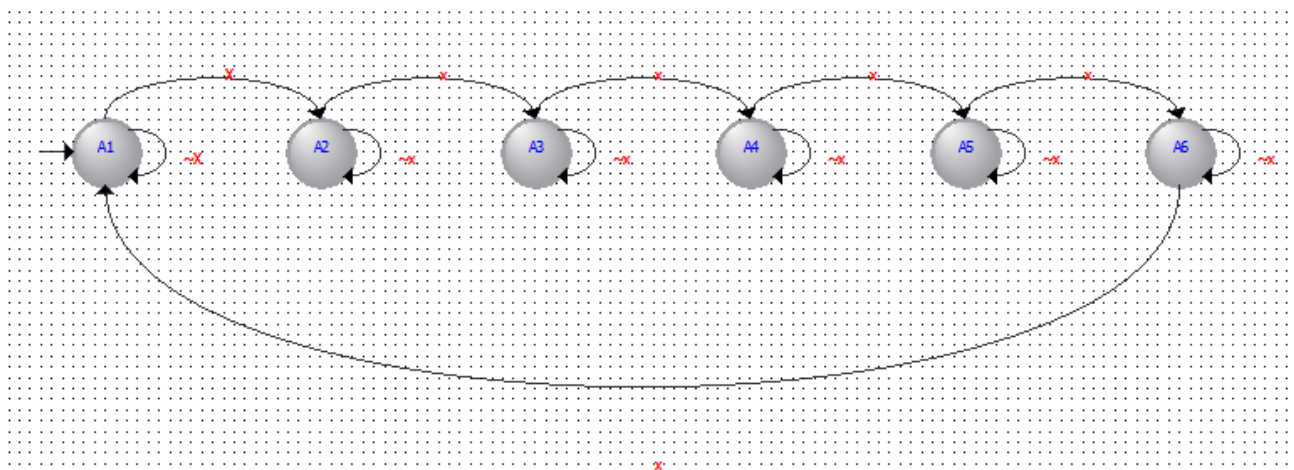


Figura 41. Diagrama de estados correspondiente al dado.

A partir de los diagramas de estados, se generan sus respectivos archivos VHDL para posibles simulaciones:

- SECUENCIA DE LUCES

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY secuenciadeluces IS
PORT (
    reset : IN STD_LOGIC := '0';
    clock : IN STD_LOGIC;
    x : IN STD_LOGIC := '0';
    LA : OUT STD_LOGIC;
    LB : OUT STD_LOGIC;
    LC : OUT STD_LOGIC;
    LD : OUT STD_LOGIC
);
END secuenciadeluces;

ARCHITECTURE BEHAVIOR OF secuenciadeluces IS
    TYPE type_fstate IS (E1,E2,E3,E4,E5,E6,E7);
    SIGNAL fstate : type_fstate;
    SIGNAL reg_fstate : type_fstate;
BEGIN
    PROCESS (clock,reg_fstate)
    BEGIN
        IF (clock='1' AND clock'event) THEN
            fstate <= reg_fstate;
        END IF;
    END PROCESS;

    PROCESS (fstate,reset,x)
    BEGIN
        IF (reset='1') THEN
            reg_fstate <= E1;
            LA <= '0';
            LB <= '0';
            LC <= '0';
            LD <= '0';
        
```

Figura 42a. Archivo VHDL correspondiente a la secuencia de luces.

```

ELSE
    LA <= '0';
    LB <= '0';
    LC <= '0';
    LD <= '0';
    CASE fstate IS
        WHEN E1 =>
            IF (NOT((x = '1')) THEN
                reg_fstate <= E2;
            ELSIF ((x = '1')) THEN
                reg_fstate <= E5;
            -- Inserting 'else' block to prevent latch inference
            ELSE
                reg_fstate <= E1;
            END IF;

            LC <= '0';

            LD <= '0';

            LB <= '0';

            LA <= '0';
        WHEN E2 =>
            reg_fstate <= E3;

            LC <= '1';

            LD <= '0';

            LB <= '1';

            LA <= '0';
        WHEN E3 =>
            IF (NOT((x = '1')) THEN
                reg_fstate <= E4;
            ELSIF ((x = '1')) THEN
                reg_fstate <= E7;
            -- Inserting 'else' block to prevent latch inference
            ELSE
                reg_fstate <= E3;
            END IF;
    
```

Figura 42b. Archivo VHDL correspondiente a la secuencia de luces.

```

        LC <= '1';
        LD <= '1';
        LB <= '1';
        LA <= '1';
    WHEN E4 =>
        IF ((x = '1')) THEN
            reg_fstate <= E5;
            -- Inserting 'else' block to prevent latch inference
        ELSE
            reg_fstate <= E4;
        END IF;
        LC <= '0';
        LD <= '1';
        LB <= '0';
        LA <= '1';
    WHEN E5 =>
        reg_fstate <= E6;
        LC <= '0';
        LD <= '0';
        LB <= '0';
        LA <= '1';

        WHEN E6 =>
            reg_fstate <= E3;
            LC <= '0';
            LD <= '0';
            LB <= '1';
            LA <= '1';
        WHEN E7 =>
            reg_fstate <= E1;
            LC <= '1';
            LD <= '0';
            LB <= '1';
            LA <= '1';
        WHEN OTHERS =>
            LA <= 'X';
            LB <= 'X';
            LC <= 'X';
            LD <= 'X';
            report "Reach undefined state";
        END CASE;
    END IF;
END PROCESS;
END BEHAVIOR;

```

Figura 42c. Archivo VHDL correspondiente a la secuencia de luces.

- DADO

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

]ENTITY DADO IS
]  PORT (
    reset : IN STD_LOGIC := '0';
    clock : IN STD_LOGIC;
    X : IN STD_LOGIC := '0';
    Q0 : OUT STD_LOGIC;
    Q1 : OUT STD_LOGIC;
    Q2 : OUT STD_LOGIC
    );
END DADO;

]ARCHITECTURE BEHAVIOR OF DADO IS
    TYPE type_fstate IS (A1,A2,A3,A4,A5,A6);
    SIGNAL fstate : type_fstate;
    SIGNAL reg_fstate : type_fstate;
BEGIN
    PROCESS (clock,reg_fstate)
    BEGIN
        IF (clock='1' AND clock'event) THEN
            fstate <= reg_fstate;
        END IF;
    END PROCESS;

    PROCESS (fstate,reset,X)
    BEGIN
        IF (reset='1') THEN
            reg_fstate <= A1;
            Q0 <= '0';
            Q1 <= '0';
            Q2 <= '0';
        ELSE
            Q0 <= '0';
            Q1 <= '0';
            Q2 <= '0';
            CASE fstate IS
                WHEN A1 =>
                    IF (NOT((X = '1')) THEN
                        reg_fstate <= A1;
                    ELSIF ((X = '1')) THEN
                        reg_fstate <= A2;
                    -- Inserting 'else' block to prevent latch inference
                    ELSE
                        reg_fstate <= A1;
                    END IF;

                    Q2 <= '0';

                    Q0 <= '1';

                    Q1 <= '0';

```

Figura 43a. Archivo VHDL correspondiente al dado.

```

WHEN A2 =>
    IF (NOT((X = '1')) ) THEN
        reg_fstate <= A2;
    ELSIF ((X = '1')) THEN
        reg_fstate <= A3;
    -- Inserting 'else' block to prevent latch inference
    ELSE
        reg_fstate <= A2;
    END IF;

    Q2 <= '0';

    Q0 <= '0';

    Q1 <= '1';
WHEN A3 =>
    IF (NOT((X = '1')) ) THEN
        reg_fstate <= A3;
    ELSIF ((X = '1')) THEN
        reg_fstate <= A4;
    -- Inserting 'else' block to prevent latch inference
    ELSE
        reg_fstate <= A3;
    END IF;

    Q2 <= '0';

    Q0 <= '1';

    Q1 <= '1';
WHEN A4 =>
    IF (NOT((X = '1')) ) THEN
        reg_fstate <= A4;
    ELSIF ((X = '1')) THEN
        reg_fstate <= A5;
    -- Inserting 'else' block to prevent latch inference
    ELSE
        reg_fstate <= A4;
    END IF;

    Q2 <= '1';

    Q0 <= '0';

    Q1 <= '0';

```

Figura 43b. Archivo VHDL correspondiente al dado.

```

WHEN A5 =>
    IF (NOT((X = '1')))) THEN
        reg_fstate <= A5;
    ELSIF ((X = '1')) THEN
        reg_fstate <= A6;
    -- Inserting 'else' block to prevent latch inference
    ELSE
        reg_fstate <= A5;
    END IF;

    Q2 <= '1';

    Q0 <= '1';

    Q1 <= '0';
WHEN A6 =>
    IF (NOT((X = '1')))) THEN
        reg_fstate <= A6;
    ELSIF ((X = '1')) THEN
        reg_fstate <= A1;
    -- Inserting 'else' block to prevent latch inference
    ELSE
        reg_fstate <= A6;
    END IF;

        Q2 <= '1';

        Q0 <= '0';

        Q1 <= '1';
    WHEN OTHERS =>
        Q0 <= 'X';
        Q1 <= 'X';
        Q2 <= 'X';
        report "Reach undefined state";
    END CASE;
END IF;
END PROCESS;
END BEHAVIOR;

```

Figura 43c. Archivo VHDL correspondiente al dado.

Conclusiones generales

Se concluye entonces que:

- Se pudo generar distintos proyectos en el entorno de simulación de Quartus II.
- Mediante la compilación de archivos VHDL se pudo solucionar distintos errores de escritura ocurridos durante el diseño de los circuitos.
- Se pudo analizar la hoja de datos del dispositivo Cyclone III para corroborar los pines de entrada y salida a utilizar.
- Se pudo corroborar, mediante la simulación, el funcionamiento de los circuitos implementados en la FPGA virtual.

Referencias

[1] Cyclone III. <https://www.intel.la/content/www/xl/es/products/programmable/fpga/cyclone-iii/features.html>.

https://www.intel.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/rm_cycloneiii_dev_kit_host_board.pdf

[2] NIOS[®]II. <https://www.intel.la/content/www/xl/es/products/programmable/processor/nios-ii.html>.

[3] Quartus II. <https://fpgasoftware.intel.com/13.0sp1/>