

Práctica de Laboratorio N.º 2

“Dispositivos Lógico Programables”

Alumno:

- Dante Agustín Cecchetti

Asignatura:

- Técnicas y Dispositivos Digitales 2

Departamento de Ingeniería Electrónica y Computación

Área Digitales

Facultad de Ingeniería

Universidad Nacional de Mar del Plata

Fecha de realización: 18/11/2022

Fecha de entrega: 25/11/2022

1) Identifique qué elementos constituyen los LEs de la FPGA Cyclone III a emplear en el laboratorio y qué estructura tienen las LABs.

Logic Elements (LEs):

Los LEs están constituidos por:

- Lookup table (LUT) de 4 entradas.
- Registro programable.
- Conexiones IN/OUT de carry en cadena.
- Conexión a 'cadena de registros' (permite que todas las LEs de una LAB operen en cascada, y el uso de los registros como shift register).
- Multiplexores.
- Conexiones:
 - ❖ Locales.
 - ❖ En columna.
 - ❖ En fila.
 - ❖ Directas.

Logic Array Block (LABs):

Las LABs están compuestas por:

- 16 LEs interconectadas localmente.
- Bloque MK9 de SRAM (256x36) que puede ser configurado como
 - ❖ True dual port.
 - ❖ Dual port
 - ❖ Single port
 - 8192x1
 - 4096x1
 - 2000x4
 - 1024x9
 - 512x18
 - 256x36.

la SRAM puede ser usada como ROM, shift register, FIFO

- Multiplicadores embebidos.

Las LABs generan conexiones entre ellas a través de una matriz de interconexión.

2) ¿De qué se trata el Nios® II?

El Nios® II es un procesador programado con los elementos de la FPGA (Flip flops, RAM, ROM, ALUs), con una estructura tipo RISC (reduced instruction set) de 32 bits. Al ser programado se lo denomina como un procesador softcore, una ventaja de que sea programado es la posibilidad de generar "custom cores".

3) ¿Qué diferencia existe entre IP Cores y los bloques embebidos (ej multiplicador embebido) disponibles en la FPGA?

Los IPcore son bloques pre armados, preprogramados, que permiten su utilización en diseños más grandes, estos bloques son parametrizados para adaptarse al diseño que se quiere implementar.

Un ejemplo de IPcore es la función FFT de la parte de DSP IPcores, este bloque ya está preprogramado, no hay una sección de la FPGA que se encarga exclusivamente de hacer FFT, este IPcore genera las conexiones necesarias para que la FFT se calcule.

En resumen, un IPcore es un “Plano” o instrucción de cómo construir un bloque lógico. Por otro lado los bloques embebidos están implementados y optimizados para hacer una tarea específica , pueden ser “programados”, pero con limitaciones, por ejemplo un multiplicador de 8 entradas puede ser concatenado con otro para hacer uno de 16 entradas programando las conexiones, pero va a seguir siendo un multiplicador, no se puede programar para hacer otras operaciones, estos bloques ya están implementados en el silicio, al estar ya implementados en el silicio mejoran la eficiencia y velocidad de operación.

4) ¿Qué tipo de celda de programación posee el dispositivo FPGA Cyclone III?

La FPGA Cyclone III utiliza celdas SRAM como celdas de programación.

5) Realice la descripción en VHDL de un Flip Flop JK.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity FFJK is
5  port ( J,K,CLOCK : in std_logic ;
6        Q_O        : out std_logic);
7  end FFJK;
8
9  architecture behavioral of FFJK is
10   signal Q :std_logic ;
11   begin
12   FFJK:
13   process (CLOCK)
14   begin
15   if rising_edge (CLOCK) then
16   Q <= (J and not Q) or (not K and Q) ;
17   end if;
18
19   end process FFJK ;
20
21   Q_O <= Q;
22
23   end behavioral;
```

6) Realice la descripción en VHDL de un restador completo de un bit.

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity restadorcompleto is
5  port ( A,B,Cin : in std_logic ;
6        Cout,Y: out std_logic);
7  end restadorcompleto;
8
9  architecture behavioral of restadorcompleto is
10 begin
11
12     Y <= ((A xor B) xor Cin);
13     Cout <= (not A and Cin) or (B and Cin) or (not A and B);
14
15 end behavioral;

```

7) Realice la descripción en VHDL del test bench del restador completo de un bit.

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity test_rest is
5  end test_rest;
6
7  architecture behavioral of test_rest is
8
9  component restadorcompleto
10 port ( A,B,Cin : in std_logic ;
11        Cout,Y: out std_logic);
12 end component;
13
14 signal A : std_logic:= '0';
15 signal Cout: std_logic:= '0';
16 signal B : std_logic:= '0';
17 signal Cin : std_logic:= '0';
18 signal Y: std_logic:= '0';
19
20 begin
21
22 uut:restadorcompleto port map(
23
24     A=>A,
25     B=>B,
26     Cin=>Cin,
27     Cout=>Cout,
28     Y=>Y);
29
30 process
31 begin
32     A <= '0' ; B <= '0' ; Cin <= '0' ; wait for 30ns;
33     A <= '0' ; B <= '0' ; Cin <= '1' ; wait for 30ns;
34     A <= '0' ; B <= '1' ; Cin <= '0' ; wait for 30ns;
35     A <= '0' ; B <= '1' ; Cin <= '1' ; wait for 30ns;
36     A <= '1' ; B <= '0' ; Cin <= '0' ; wait for 30ns;
37     A <= '1' ; B <= '0' ; Cin <= '1' ; wait for 30ns;
38     A <= '1' ; B <= '1' ; Cin <= '0' ; wait for 30ns;
39     A <= '1' ; B <= '1' ; Cin <= '1' ; wait for 30ns;
40 end process;
41
42 end behavioral;

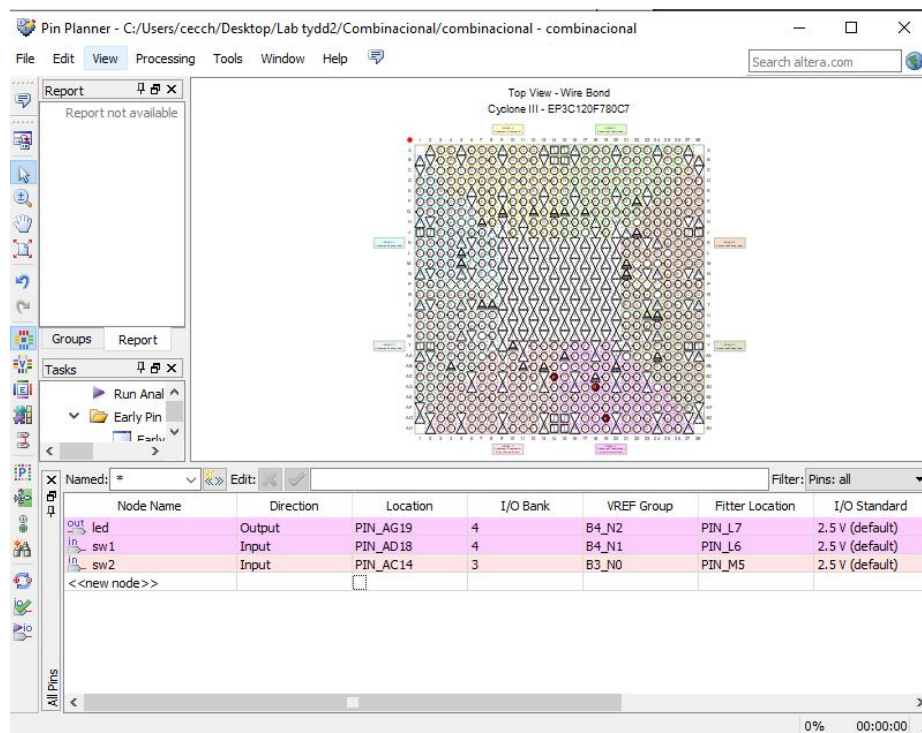
```

PARTE A: Implementación de un circuito combinacional en FPGA

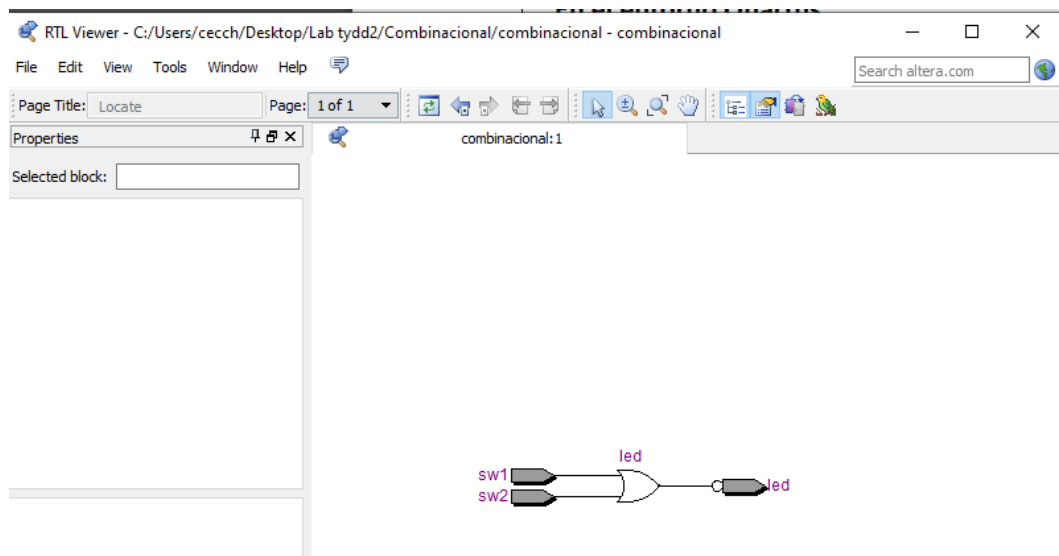
- 1) En un archivo VHDL describa el circuito combinacional mostrado:



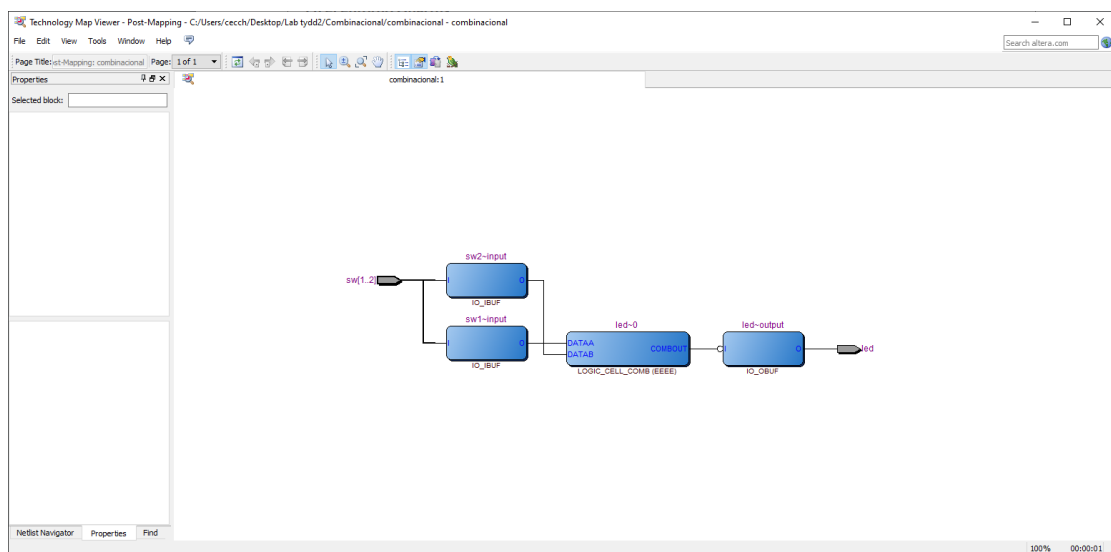
- 2) Asigne los pines de entrada y salida del diseño



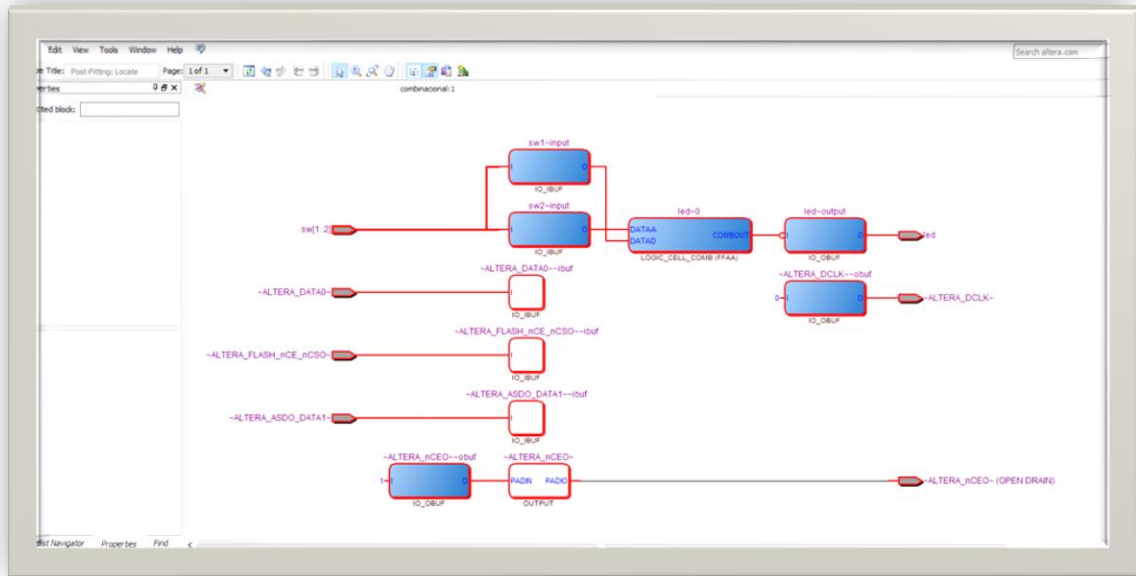
3) Verifique el circuito implementado mediante el visor de RTL



4) Verifique el circuito implementado luego del mapeo en el dispositivo.

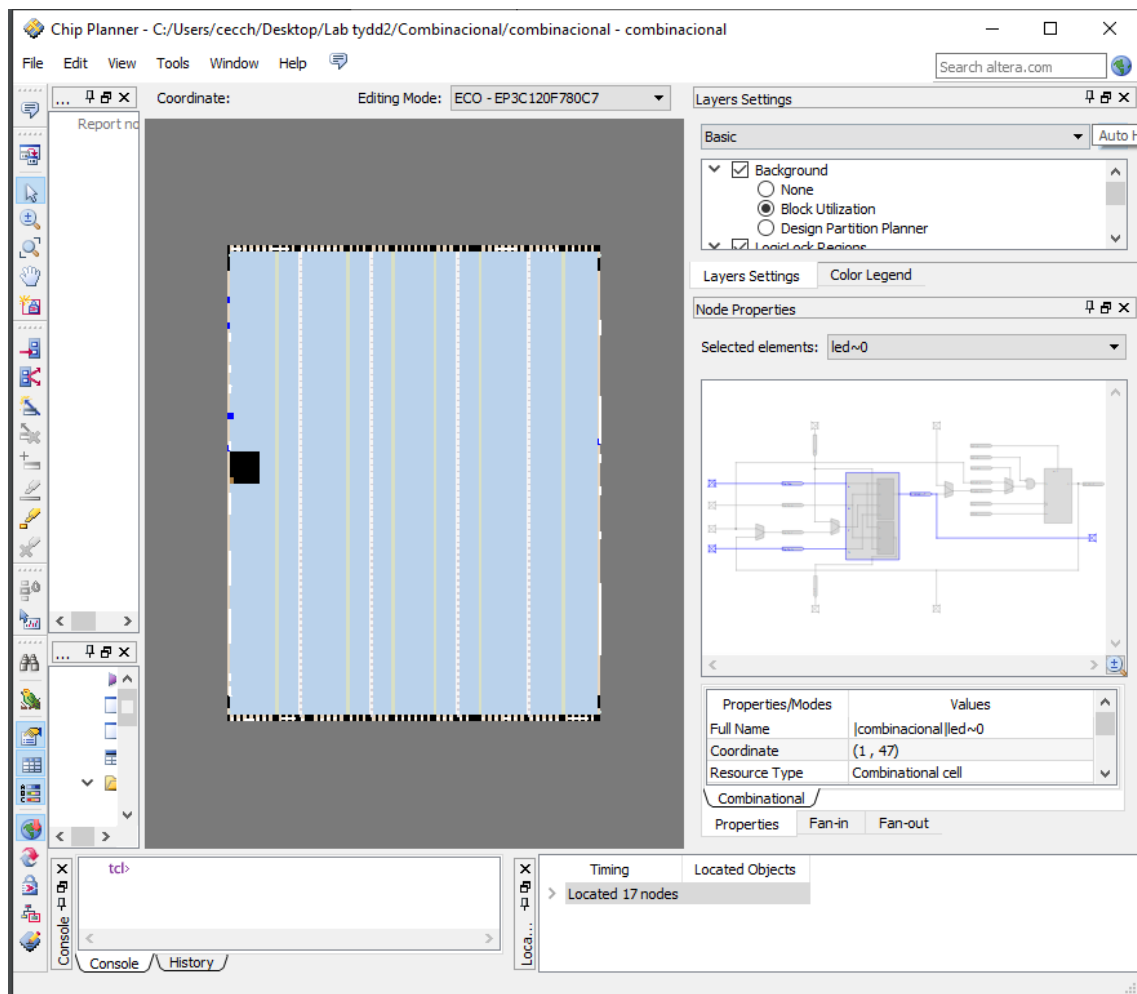


Map Viewer-Post Mapping

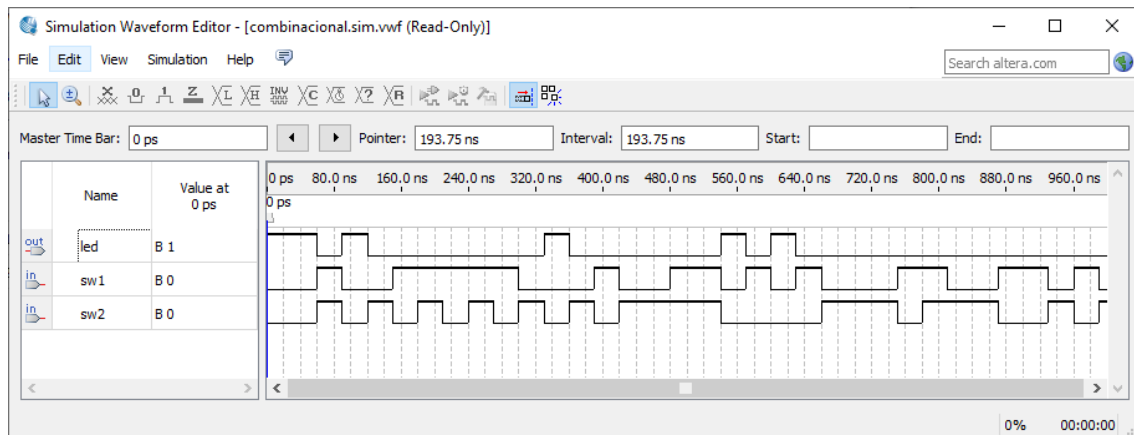


Map Viewer-Post fitting

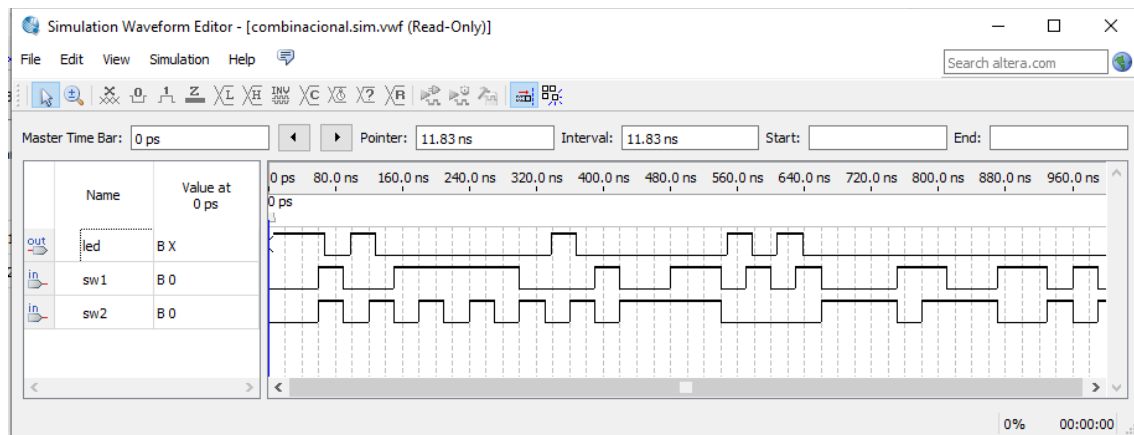
5) Verifique la implementación en el chip



6) Realice la simulación funcional y temporal del circuito mediante el simulador de Quartus.

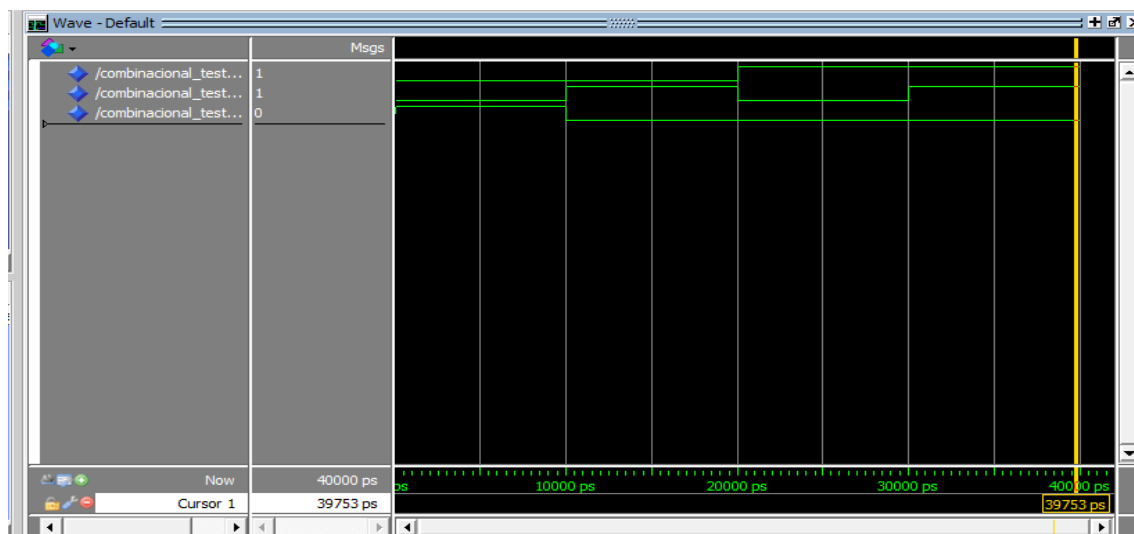


Simulación con Run Functional Simulation.

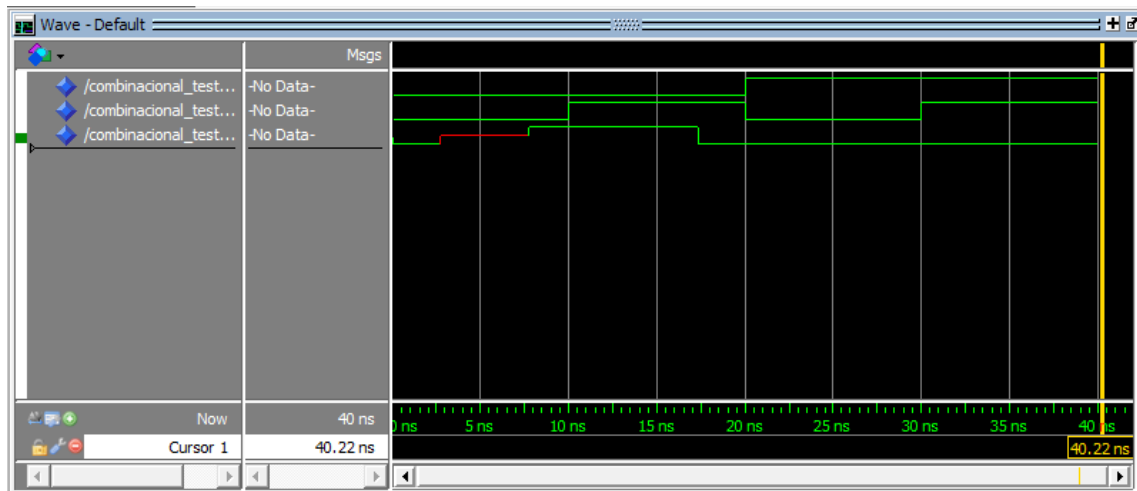


Simulación con Run Timing Simulation.

7) Realice la simulación del circuito mediante la simulación de Modelsim.

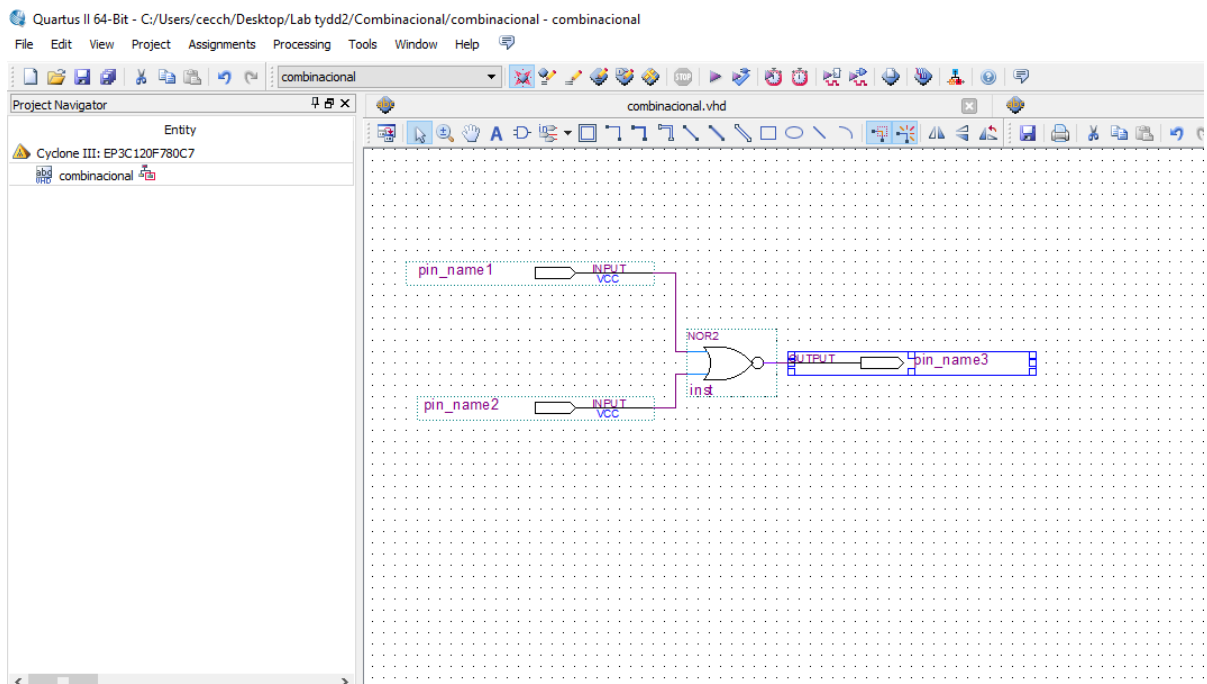


Simulación con RTL Simulation.



Simulación con Gate Level Simulation.

8) Genere un archivo esquemático



PARTE B: Implementación de un sumador completo con salidas registradas en VHDL

- 1) Genere un archivo VHDL en el cual describa un Flip Flop D

```
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  entity FFD is
7  port ( D,CLOCK : in std_logic ;
8        Q        : out std_logic);
9  end FFD;
10
11 architecture behavioral of FFD is
12
13 begin
14 process(CLOCK)
15 begin
16 if (CLOCK = '1' and CLOCK'EVENT) then
17   Q <= D ;
18 end if;
19
20 end process;
21
22 end behavioral;
```

- 2) Genere un archivo VHDL en el cual describa el circuito sumador completo

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity sumador_FFD is
5  port ( in_a : in STD_LOGIC ;
6        in_b : in STD_LOGIC ;
7        in_cin : in STD_LOGIC ;
8        clk : in STD_LOGIC ;
9        o_f : out STD_LOGIC ;
10       o_cout : out STD_LOGIC
11       );
12 end sumador_FFD;
13
14 architecture Behavioral of sumador_FFD is
15
16 component FFD
17 port (D,CLOCK : in std_logic;
18       Q        : out std_logic
19       );
20 end component;
21
22 signal a,b,cin,cout,f : std_logic;
23 begin
24
25 f <= a xor b xor cin;
26 cout <= (a and b) or (cin and (a xor b) ) ;
27
28 D1 : FFD port map (in_a, clk, a) ;
29 D2 : FFD port map (in_b, clk, b) ;
30 D3 : FFD port map (in_cin, clk, cin) ;
31 D4 : FFD port map (cout, clk, o_cout);
32 D5 : FFD port map (f, clk, o_f);
33
34 end Behavioral;
```

3) Asigne los pines de entrada y salida del diseño

Pin Planner - C:/Users/usuario/Desktop/Laboratorio-TyDD2/sumador con FFD/Sumador - Sumador

File Edit View Processing Tools Window Help

Search altera.com

Report not available

Top View - Wire Bond
Cyclone III - EP3C120F780C7

Groups Report

Tasks

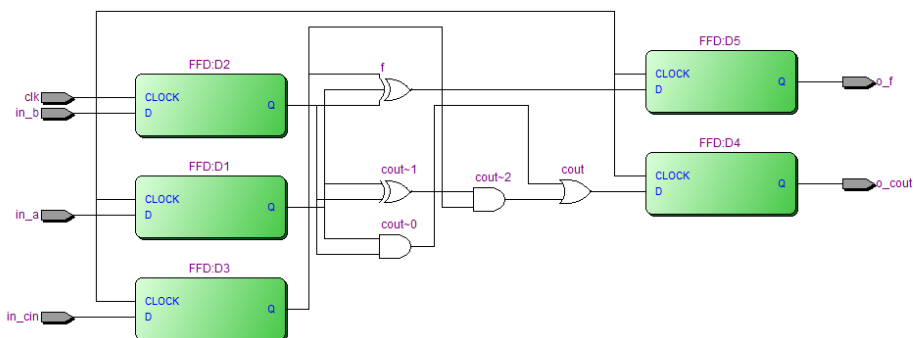
- Run Anal
- Early Pin
- Run
- Expo

Named: * Edit: Filter: Pins: all

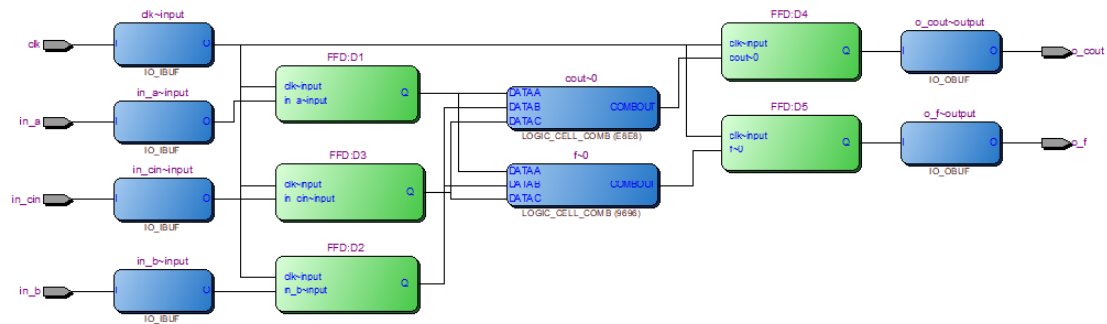
Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard
in clk	Input	PIN_A14	8	B8_N0	PIN_J2	2.5 V (default)
in_a	Input	PIN_AC14	3	B3_N0	PIN_G4	2.5 V (default)
in_b	Input	PIN_AD18	4	B4_N1	PIN_H4	2.5 V (default)
in_cin	Input	PIN_AG23	4	B4_N1	PIN_E1	2.5 V (default)
o_cout	Output	PIN_AF19	4	B4_N1	PIN_H3	2.5 V (default)
o_f	Output	PIN_AG19	4	B4_N2	PIN_G3	2.5 V (default)
<<new node>>						

Feedback 0% 00:00:00

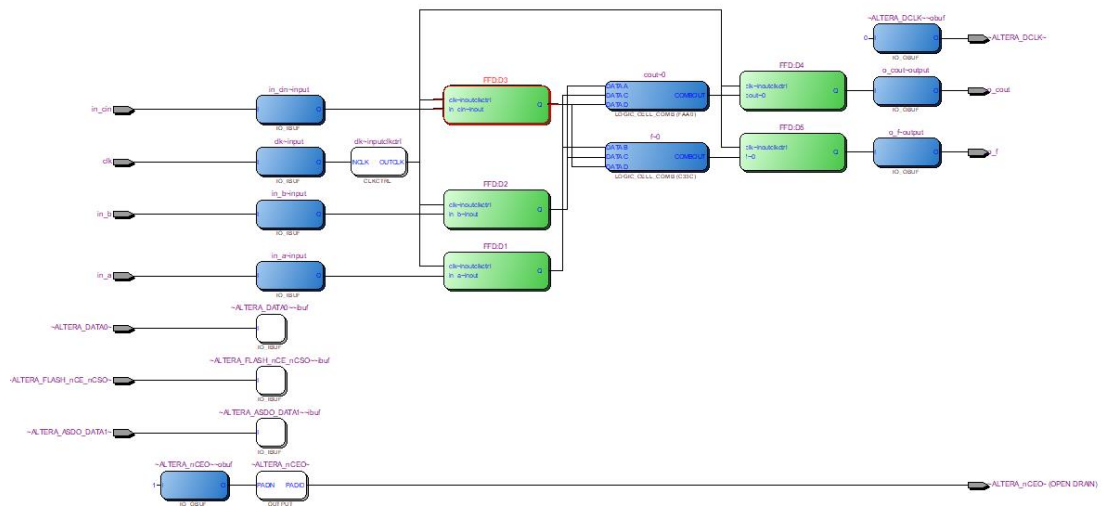
4) Verifique el circuito implementado mediante el visor de RTL



5) Verifique el circuito implementado luego del mapeo en el dispositivo

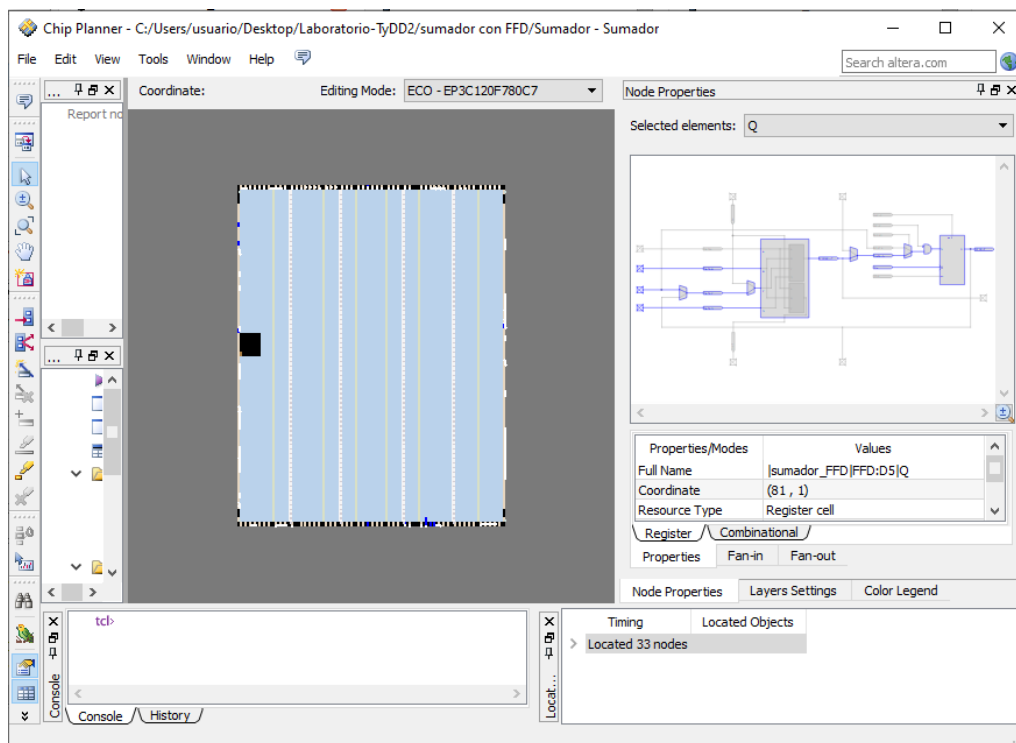


Map Viewer-Post Mapping

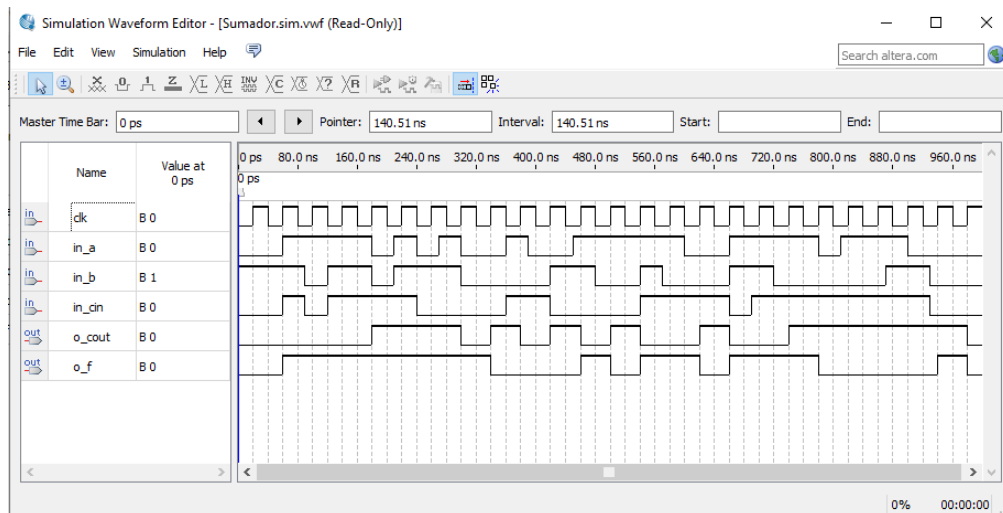


Map Viewer-Post fitting

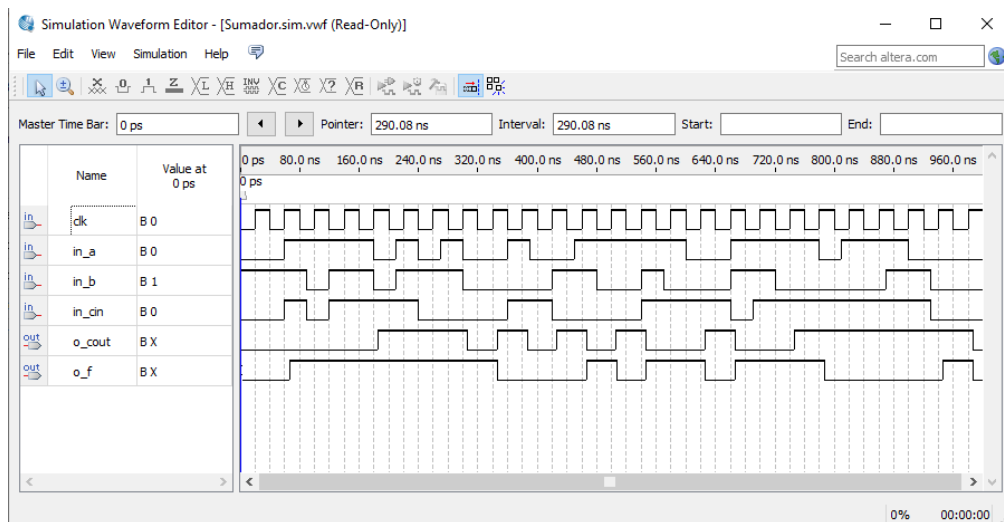
6) Verifique la implementación en el chip



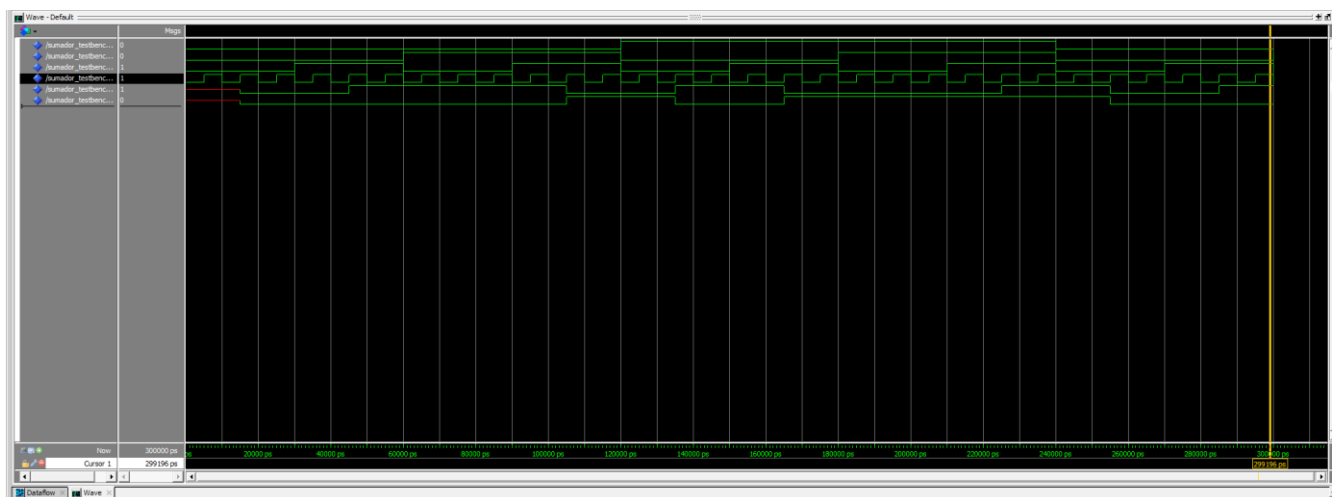
7) Verifique el funcionamiento funcional y temporal del circuito mediante la simulación de Quartus



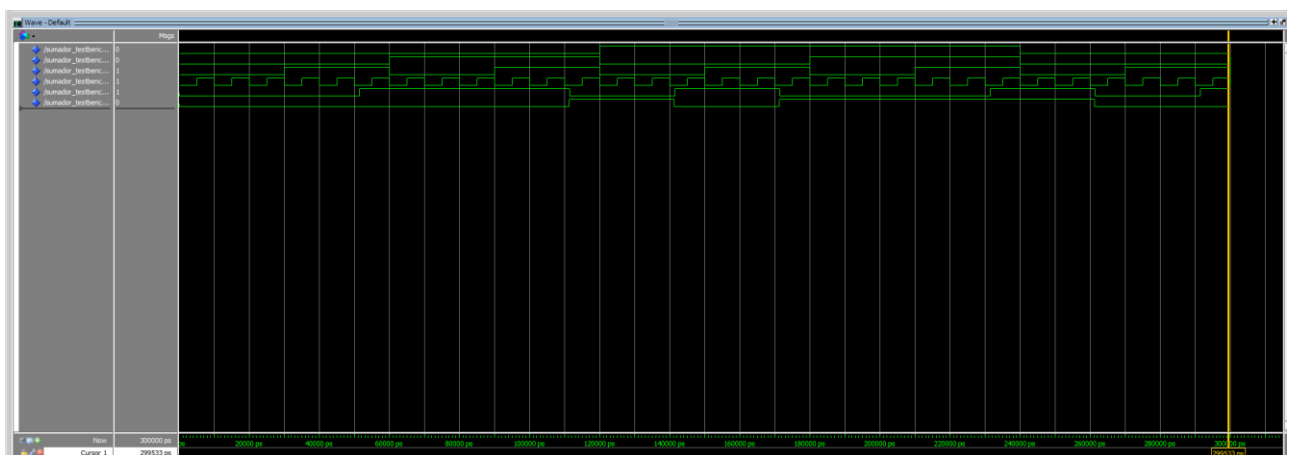
Simulación con Run Functional Simulation.



8) Verifique el funcionamiento del circuito mediante la simulación de Modelsim



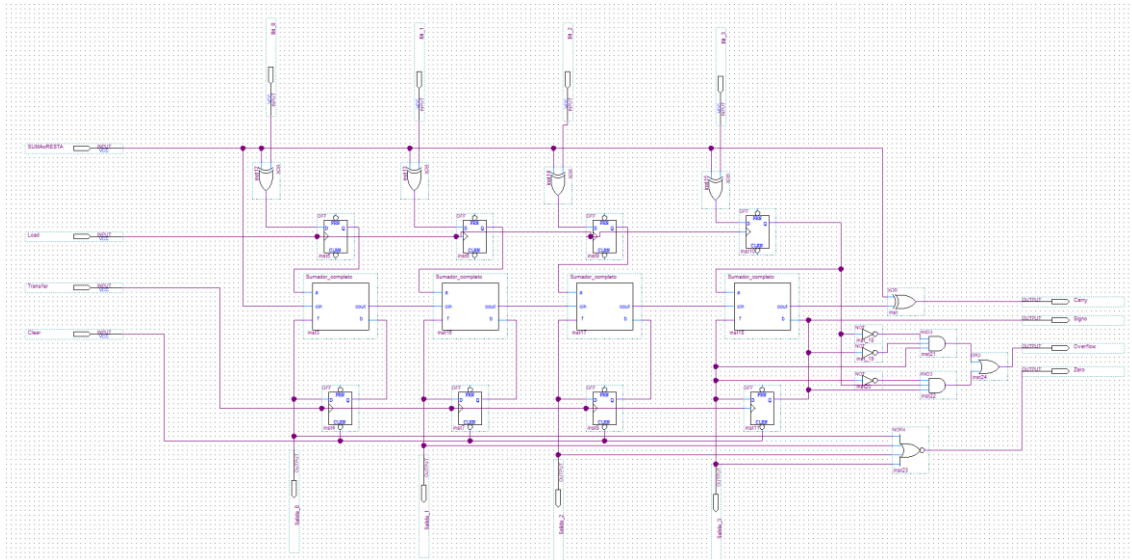
Simulación con RTL Simulation.



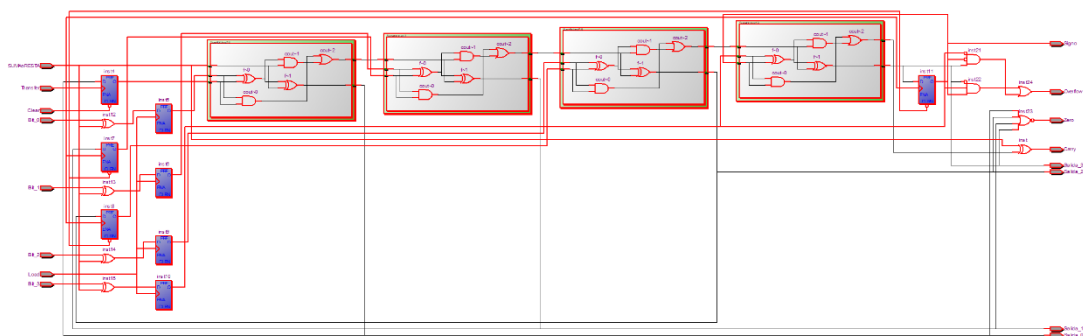
Simulación con Gate Level Simulation.

PARTE C: Implementación de un sumador/restador de 4 bits mediante un sumador completo y los bits C, V, N y Z, empleando el entorno esquemático.

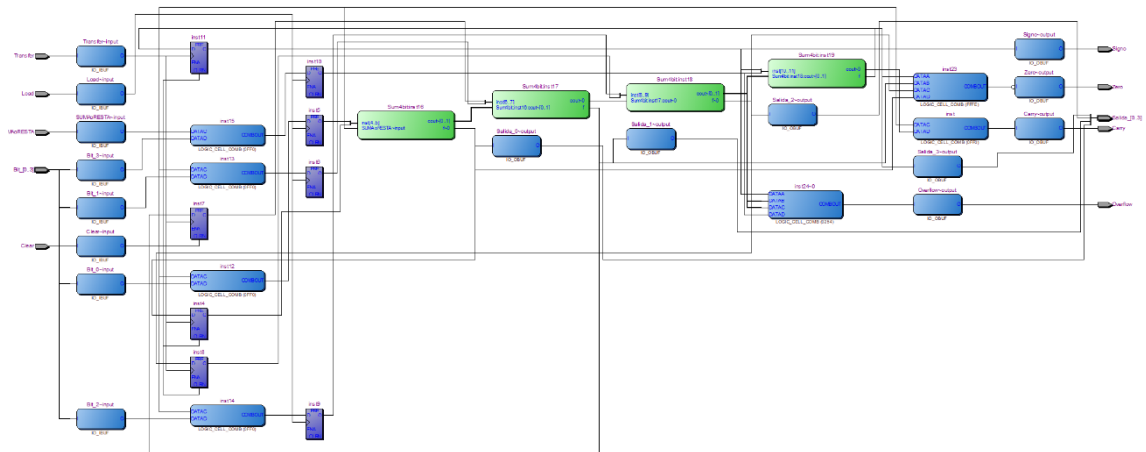
- 1) Genere un archivo esquemático



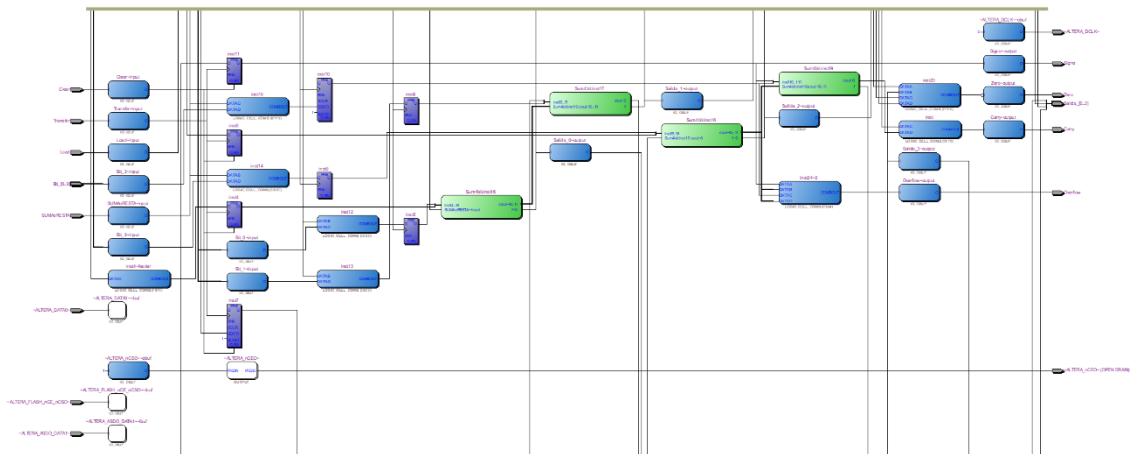
- 2) Verifique el circuito implementado mediante el visor de RTL.



3) Verifique el circuito implementado luego del mapeo en el dispositivo



Map Viewer-Post mapping



Map Viewer-Post fitting

4) Verifique la implementación en el chip

Pin Planner - C:/Users/cecch/Desktop/Laboratorio-TyDD2/Sumador de 4 bit con banderas/Sum4bit - Sum4bit

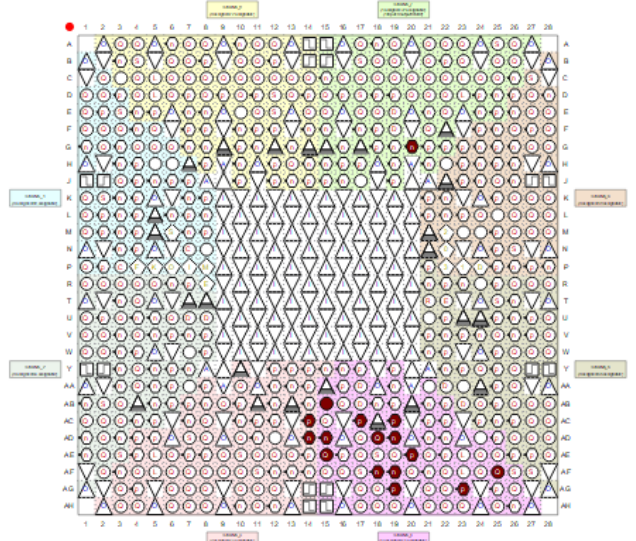
File Edit View Processing Tools Window Help Search altera.com

Report Report not available

Groups Report

Tasks Run Anal Early Pin Early Run

Top View - Wire Bond
Cyclone III - EP3C120F780C7

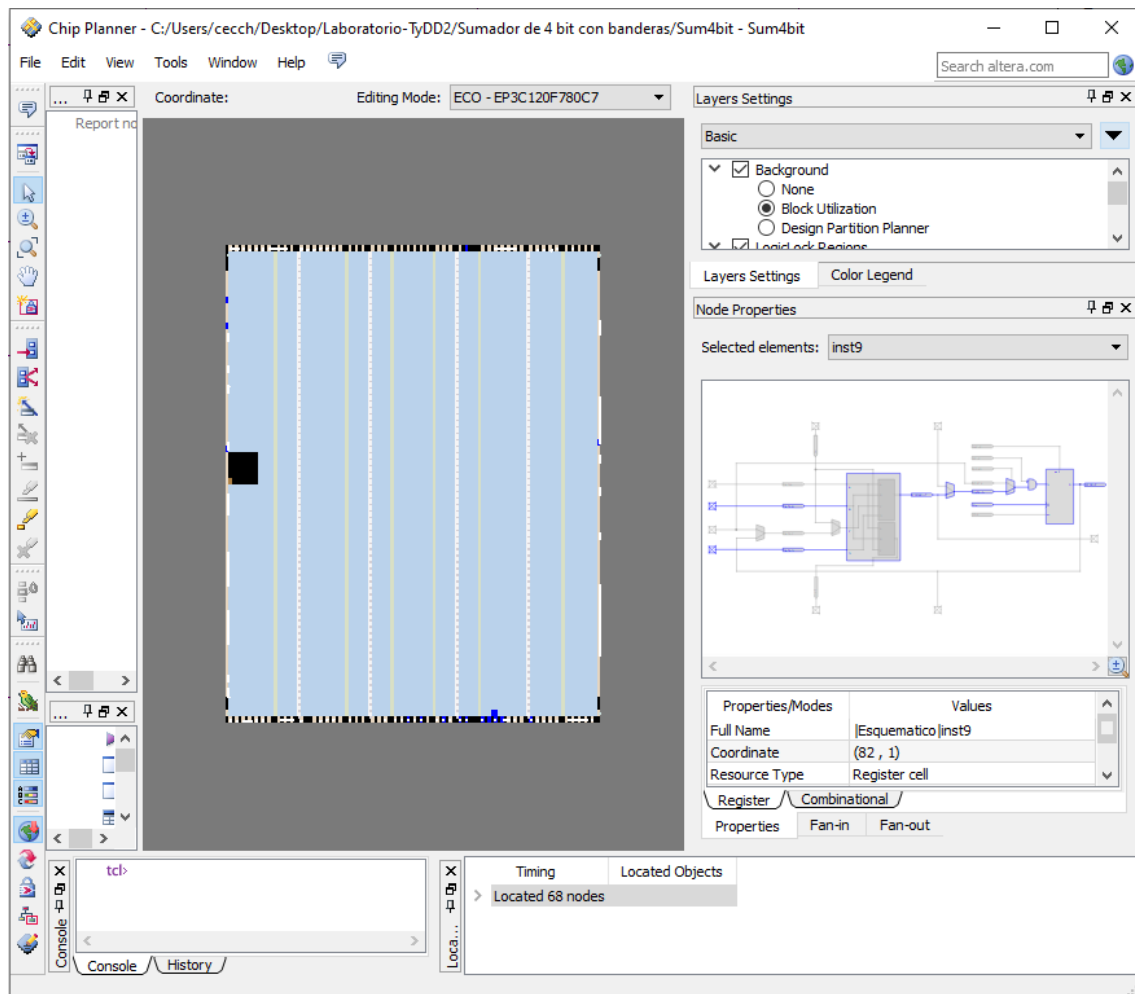


Named: * Edit: Filter: Pins: all

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard
in Bit_0	Input	PIN_AC14	3	B3_N0	PIN_J5	2.5 V (default)
in Bit_1	Input	PIN_AD18	4	B4_N1	PIN_J7	2.5 V (default)
in Bit_2	Input	PIN_AG23	4	B4_N1	PIN_L3	2.5 V (default)
in Bit_3	Input	PIN_AC19	4	B4_N0	PIN_M3	2.5 V (default)
out Carry	Output	PIN_AF19	4	B4_N1	PIN_K2	2.5 V (default)
in Clear	Input	PIN_AD14	3	B3_N0	PIN_Y2	2.5 V (default)
in Load	Input	PIN_G20	7	B7_N1	PIN_J2	2.5 V (default)
out Overflow	Output	PIN_AG19	4	B4_N2	PIN_J6	2.5 V (default)
out Salida_0	Output	PIN_AC17	4	B4_N2	PIN_M4	2.5 V (default)
out Salida_1	Output	PIN_AE15	4	B4_N2	PIN_K3	2.5 V (default)
out Salida_2	Output	PIN_AD19	4	B4_N0	PIN_L4	2.5 V (default)
out Salida_3	Output	PIN_AF18	4	B4_N1	PIN_K1	2.5 V (default)
out Signo	Output	PIN_AE20	4	B4_N1	PIN_K4	2.5 V (default)
in SUMAorESTA	Input	PIN_AB15	4	B4_N2	PIN_G2	2.5 V (default)
in Transfer	Input	PIN_AF25	4	B4_N1	PIN_J1	2.5 V (default)
out Zero	Output	PIN_AD15	4	B4_N2	PIN_G1	2.5 V (default)

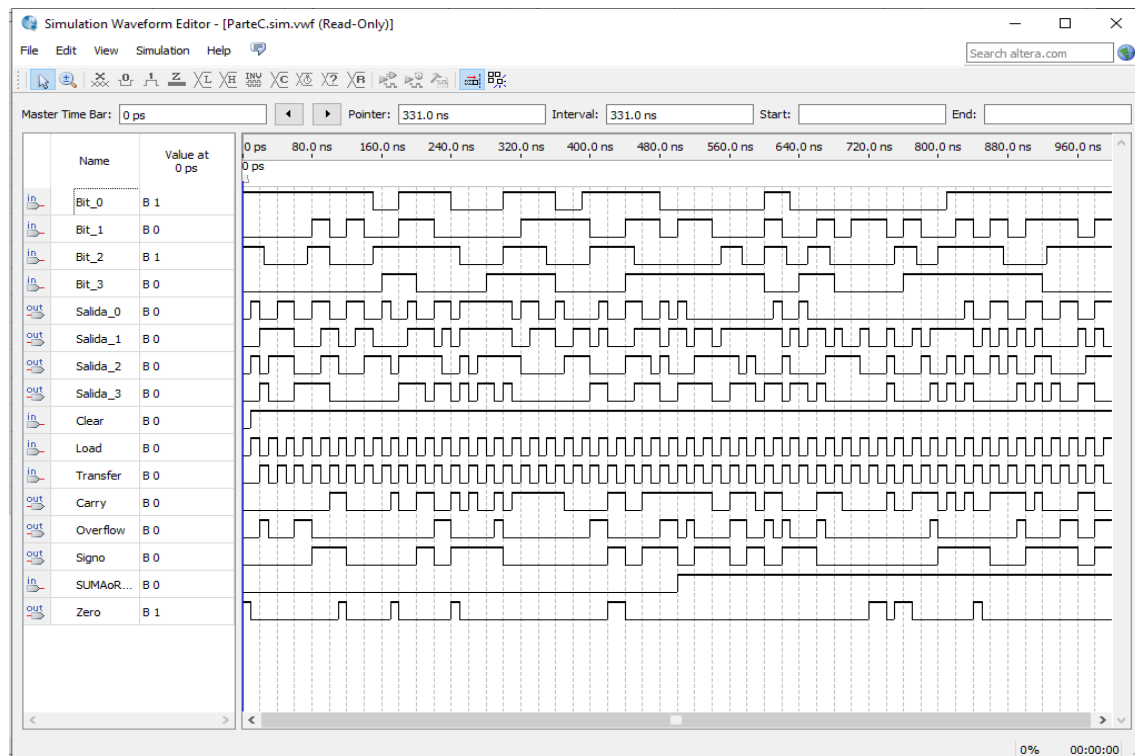
0% 00:00:00

Asignación de pines

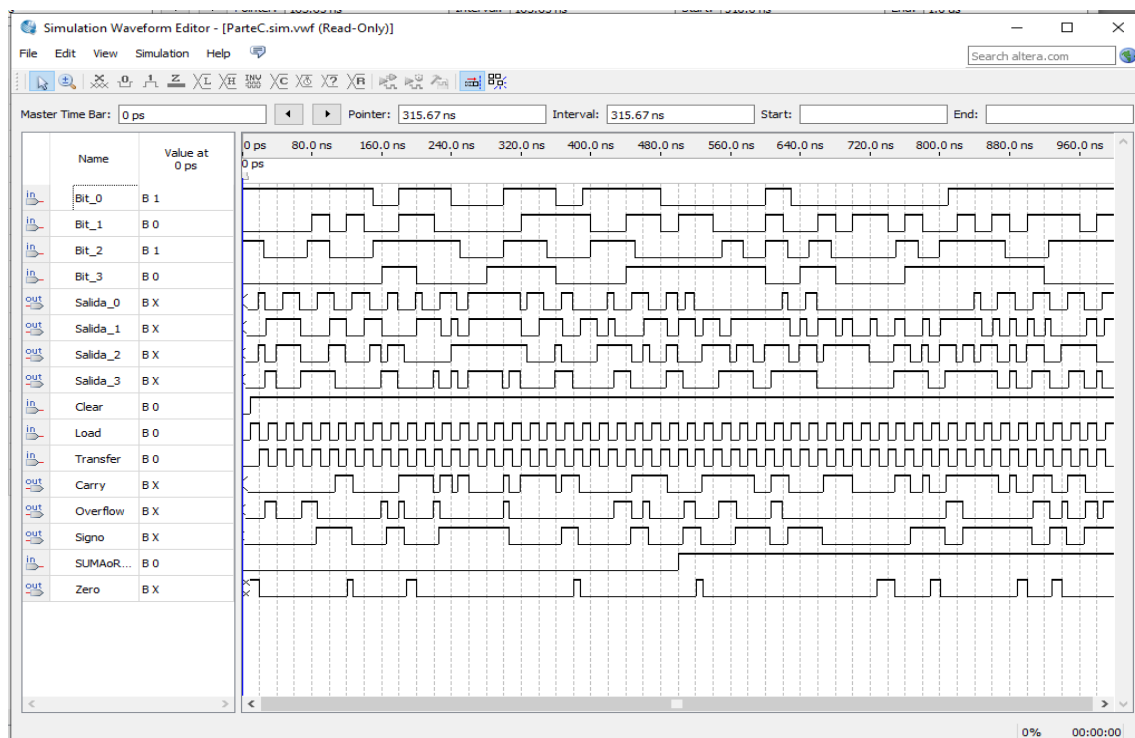


Visualización de la implementación física en el chip

- 5) Verifique el funcionamiento funcional y temporal del circuito mediante la simulación de Quartus

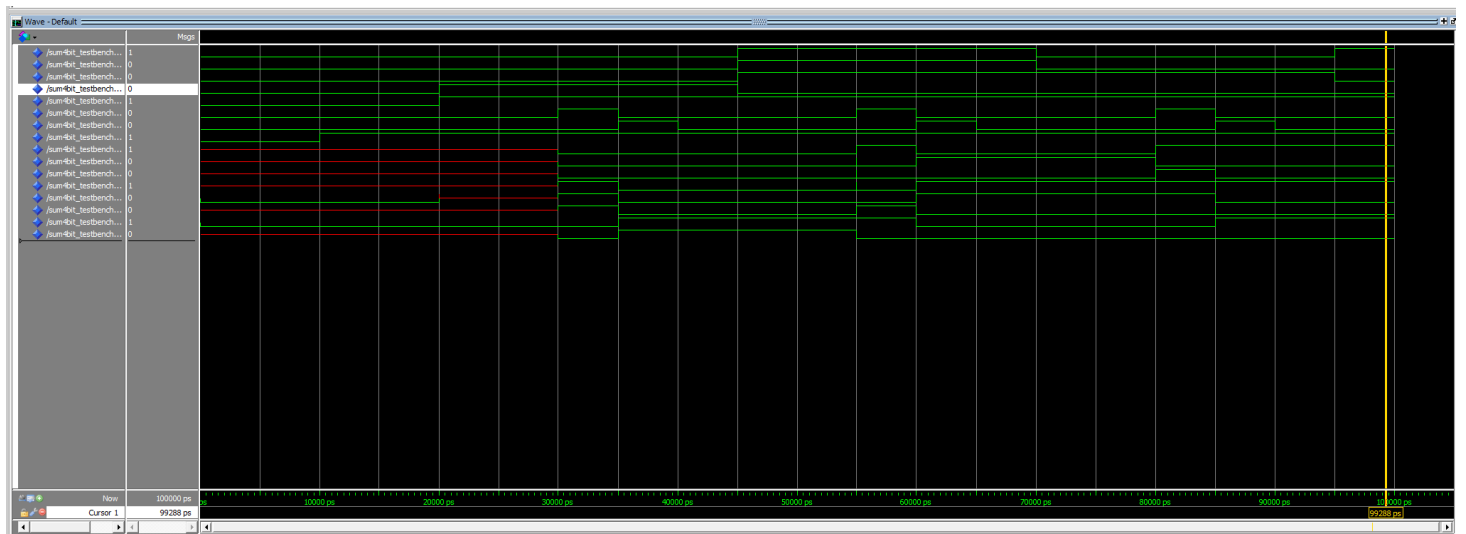


Simulación con Run Functional Simulation.

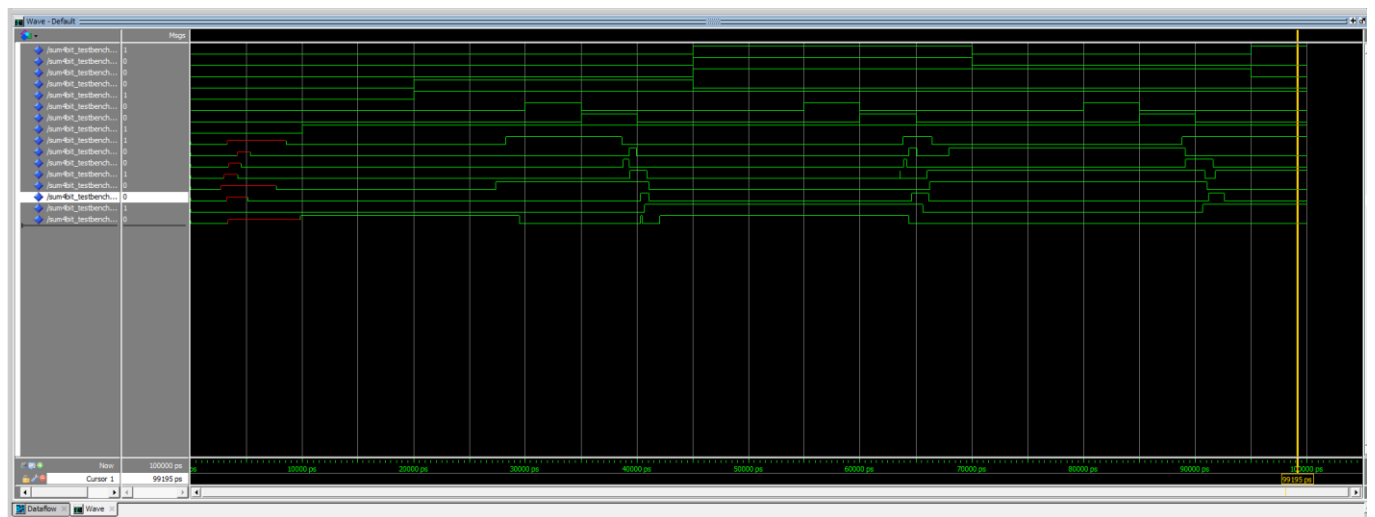


Simulación con Run Timing Simulation.

6) Verifique el funcionamiento del circuito mediante la simulación de Modelsim



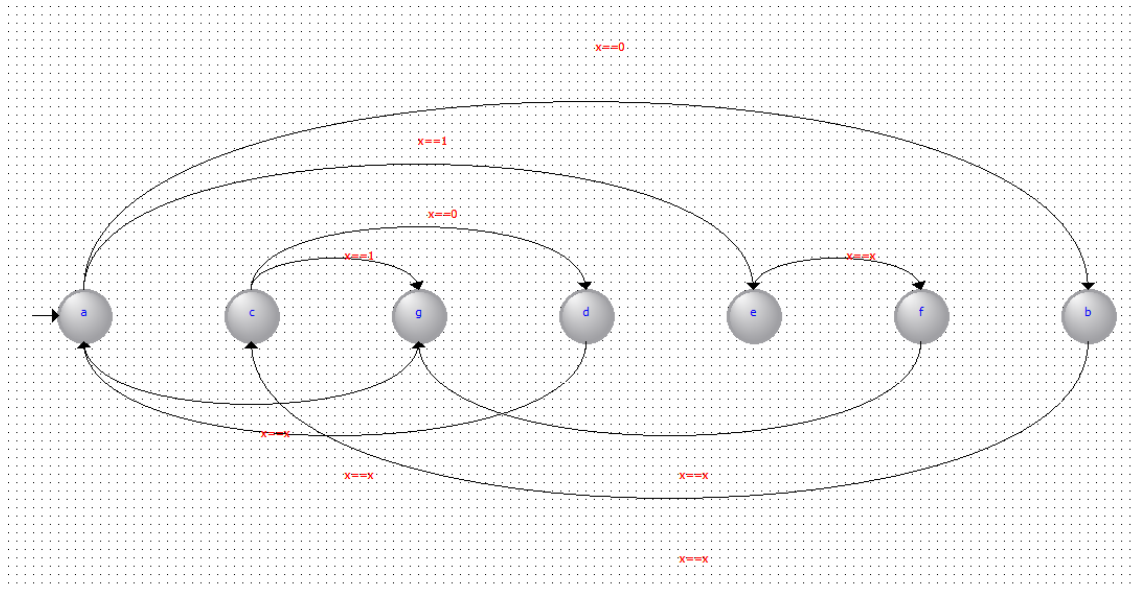
Simulación con RTL Simulation



Simulación con Gate Level Simulation.

PARTE D: Implementación de una máquina de estado.

1) Implemente un diagrama de estados



2) Genere el código VHDL

```

18  LIBRARY ieee;
19  USE ieee.std_logic_1164.all;
20
21  ENTITY maquina_de_estados IS
22  PORT (
23      reset : IN STD_LOGIC := '0';
24      clock : IN STD_LOGIC;
25      x : IN STD_LOGIC := '0';
26      Z : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
27  );
28  END maquina_de_estados;
29
30  ARCHITECTURE BEHAVIOR OF maquina_de_estados IS
31      TYPE type_fstate IS (a,b,c,d,e,f,g);
32      SIGNAL fstate : type_fstate;
33      SIGNAL reg_fstate : type_fstate;
34  BEGIN
35      PROCESS (clock,reg_fstate)
36      BEGIN
37          IF (clock='1' AND clock'event) THEN
38              fstate <= reg_fstate;
39          END IF;
40      END PROCESS;
41
42      PROCESS (fstate,reset,x)
43      BEGIN
44          IF (reset='1') THEN
45              reg_fstate <= a;
46              Z <= "0000";
47          ELSE
48              Z <= "0000";
49              CASE fstate IS
50                  WHEN a =>
51                      IF ((x = '0')) THEN
52                          reg_fstate <= b;
53                      ELSIF ((x = '1')) THEN
54                          reg_fstate <= e;
55                      -- Inserting 'else' block to prevent latch inference
56                      ELSE
57                          reg_fstate <= a;
58                      END IF;
59
60                      Z <= "0000";
61                  WHEN b =>
62                      reg_fstate <= c;
63
64                      Z <= "0110";
65                  WHEN c =>
66                      IF ((x = '0')) THEN
67                          reg_fstate <= d;
68                      ELSIF ((x = '1')) THEN
69                          reg_fstate <= g;
70                      -- Inserting 'else' block to prevent latch inference
71                      ELSE
72                          reg_fstate <= c;
73                      END IF;
74
75                      Z <= "1111";
76                  WHEN d =>
77                      reg_fstate <= a;
78
79                      Z <= "1001";
80                  WHEN e =>
81                      reg_fstate <= f;
82
83                      Z <= "1000";
84                  WHEN f =>

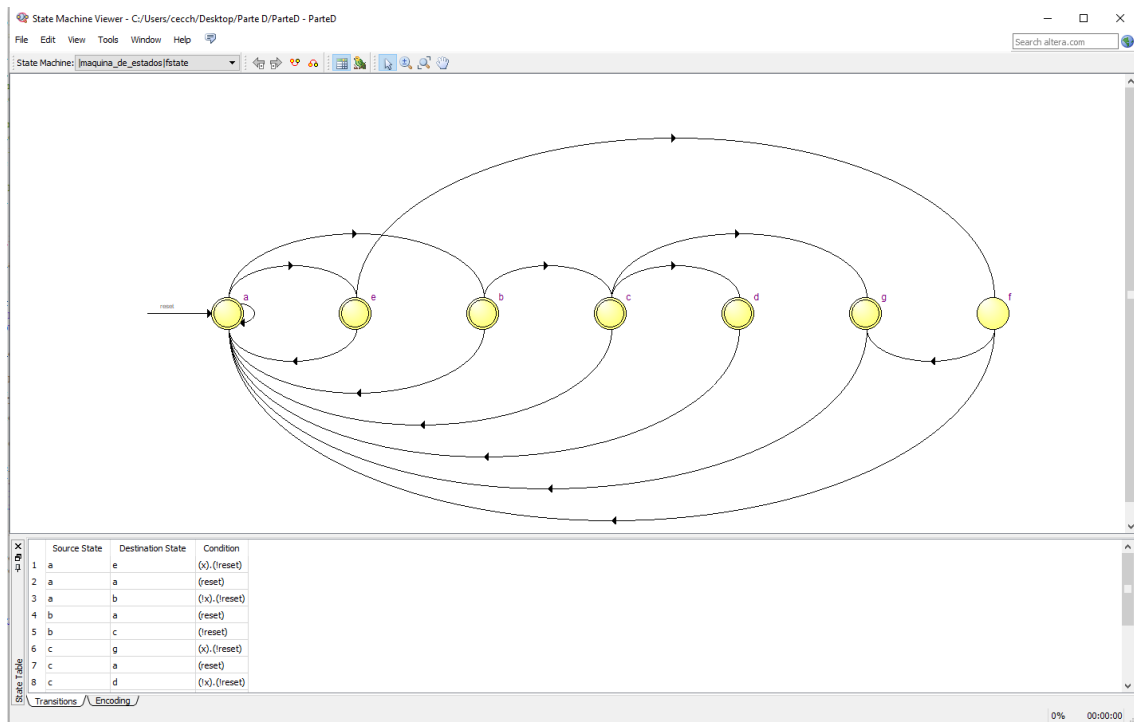
```

```

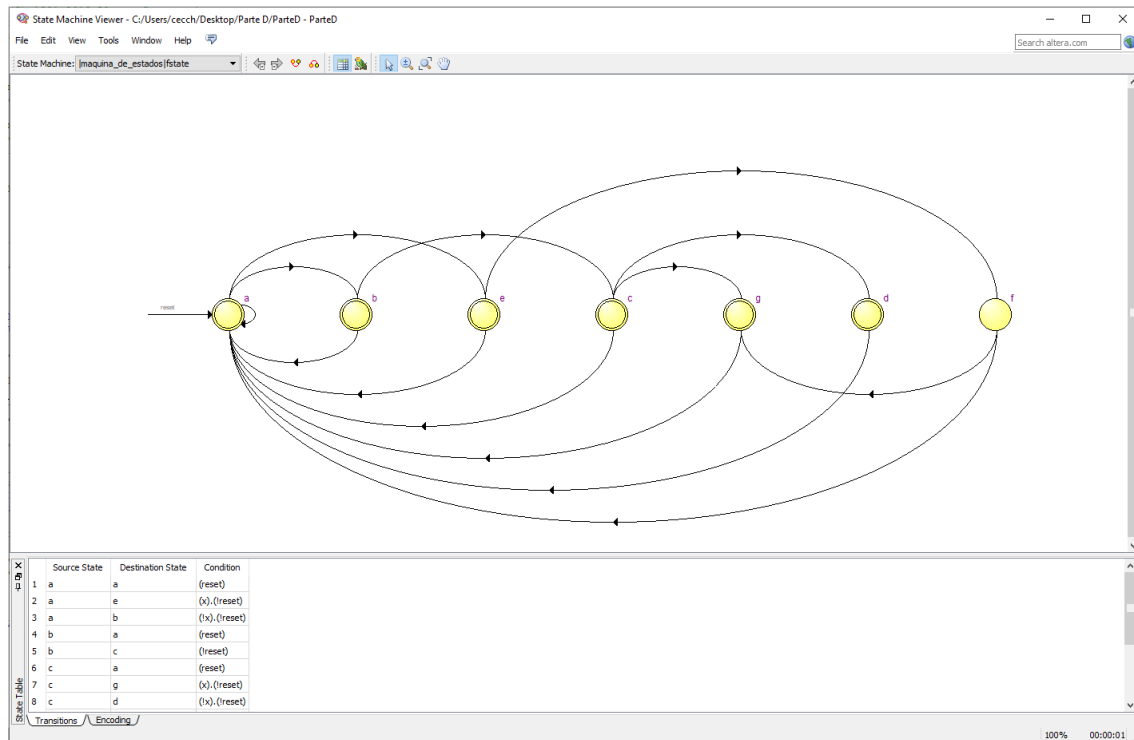
85         reg_fstate <= g;
86
87         Z <= "1100";
88         WHEN g =>
89             reg_fstate <= a;
90
91         Z <= "1110";
92         WHEN OTHERS =>
93             Z <= "XXXX";
94             report "Reach undefined state";
95         END CASE;
96     END IF;
97 END PROCESS;
98 END BEHAVIOR;
99

```

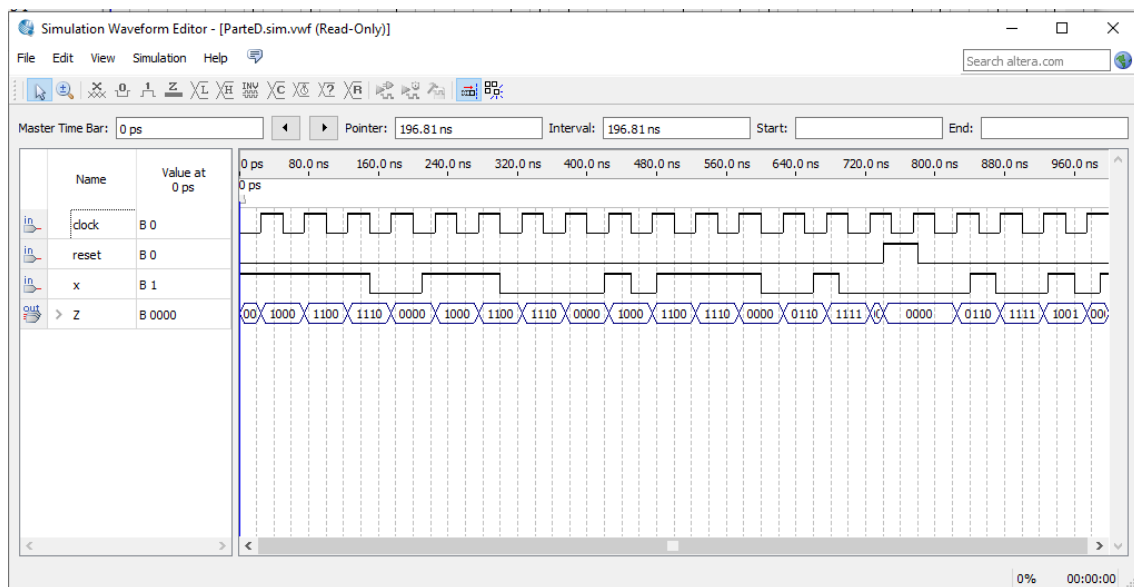
3) Vea la máquina implementada



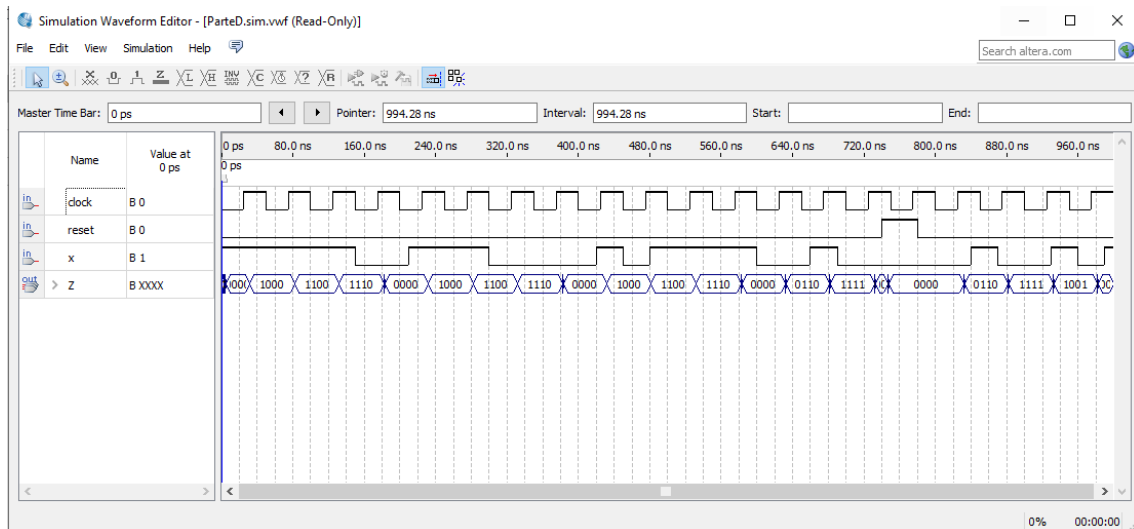
- 4) Cambie el estilo de procesamiento de la máquina de estado, para que el compilador asigne otra codificación. Compile nuevamente y vea el circuito implementado



- 5) Realice las simulaciones para verificar el correcto funcionamiento.



Simulación con Run Functional Simulation.



Simulación con Run Timing Simulation

PARTE D (Segunda parte): Implementación mediante Template de máquina de estado

- 1) Seleccione la máquina de estados que desee, Moore o Mealy, note que el template es genérico de 4 estados, Ud. deberá incluir o quitar estados según la máquina que desee implementar, además de modificar la cantidad de bits de las entradas y salidas.

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity ParteDb is
5
6     port(
7         clk          : in std_logic;
8         entrada      : in std_logic;
9         reset        : in std_logic;
10        salida       : out std_logic_vector(3 downto 0)
11    );
12
13 end entity;
14
15 architecture rtl of ParteDb is
16     type state_type is (s0, s1, s2, s3, s4, s5, s6);
17
18     signal state : state_type;
19
20 begin
21     process (clk, reset)
22     begin
23         if reset = '1' then
24             state <= s0;
25         elsif (rising_edge(clk)) then
26             case state is
27                 when s0 =>
28                     if entrada = '1' then
29                         state <= s4;
30                     else
31                         state <= s1;
32                     end if;
33                 when s1 =>
34                     if entrada = '1' then
35                         state <= s2;
36                     else
37                         state <= s2;
38                     end if;
39                 when s2 =>
40                     if entrada = '1' then
41                         state <= s6;
42                     else
43                         state <= s3;
44                     end if;
45                 when s3 =>
46                     if entrada = '1' then
47                         state <= s0;
48                     else
49                         state <= s0;
50                     end if;
51                 when s4 =>
52                     if entrada = '1' then
53                         state <= s5;
54                     else
55                         state <= s5;
56                     end if;
57                 when s5 =>
58                     if entrada = '1' then
59                         state <= s2;
60                     else
61                         state <= s2;
62                     end if;
63                 when s6 =>
64                     if entrada = '1' then
65                         state <= s0;
66                     else
67                         state <= s0;
68                     end if;
69             end case;

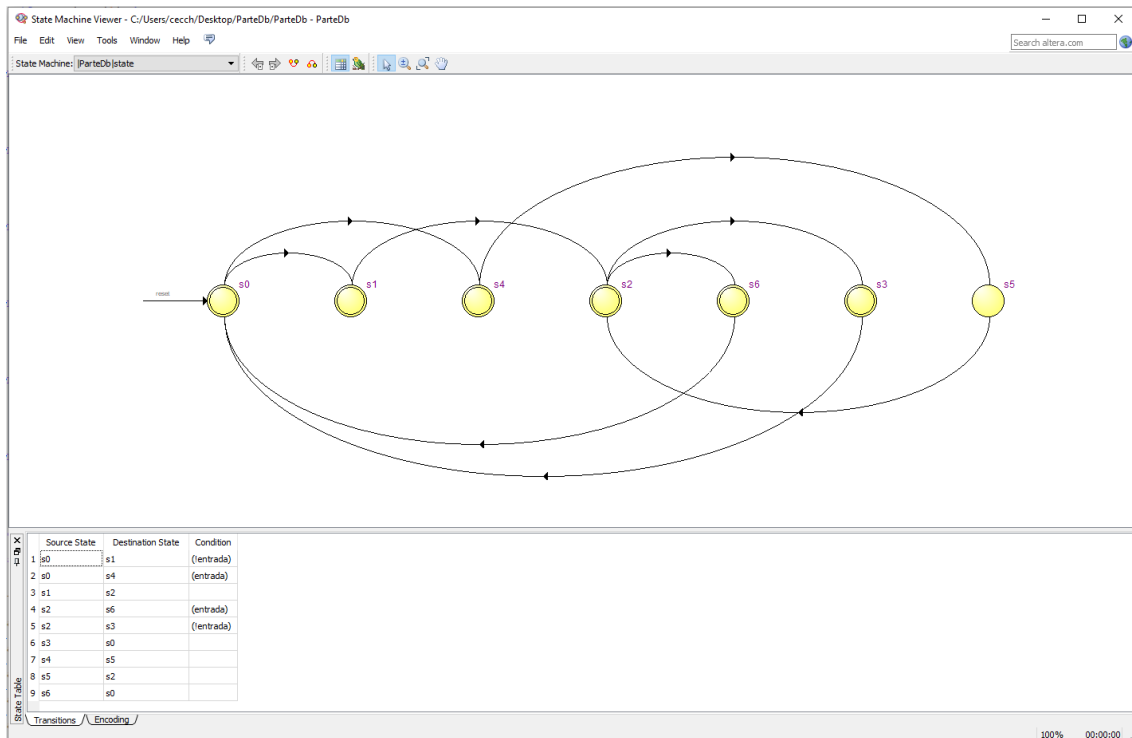
```

```

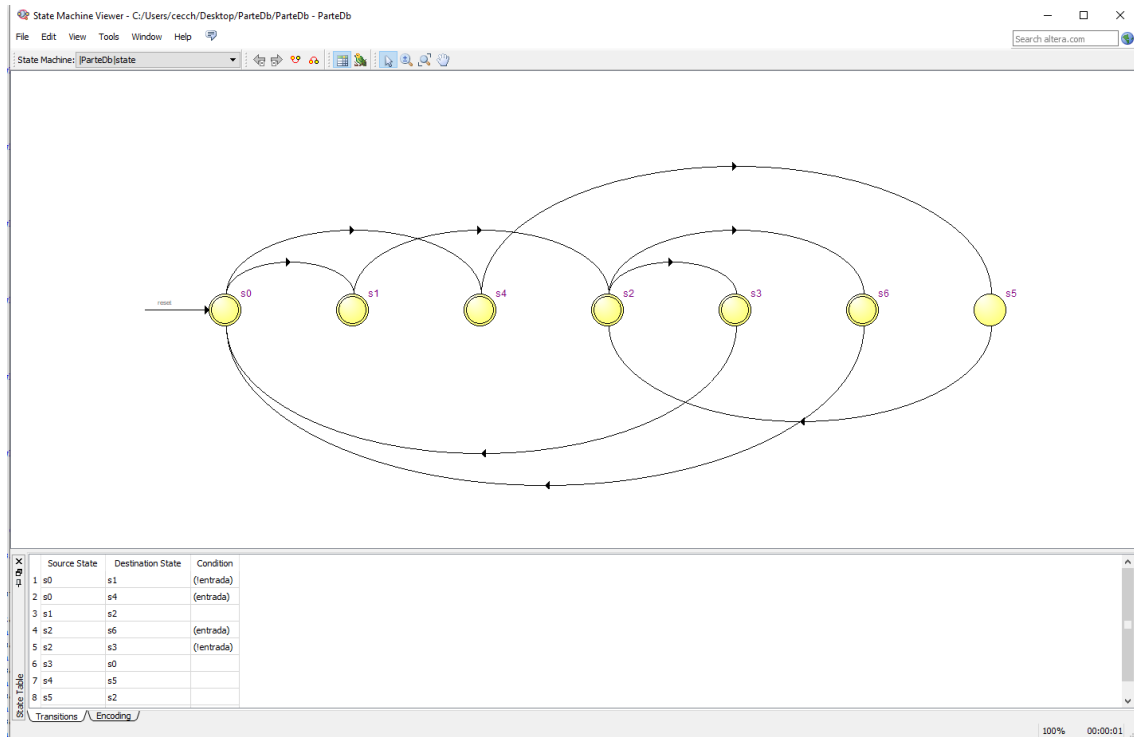
70     end if;
71 end process;
72
73
74 process (state)
75 begin
76     case state is
77     when s0 =>
78         salida <= "0000";
79     when s1 =>
80         salida <= "0110";
81     when s2 =>
82         salida <= "1111";
83     when s3 =>
84         salida <= "1001";
85     when s4 =>
86         salida <= "1000";
87     when s5 =>
88         salida <= "1100";
89     when s6 =>
90         salida <= "1110";
91
92     end case;
93 end process;
94
95 end rtl;
96

```

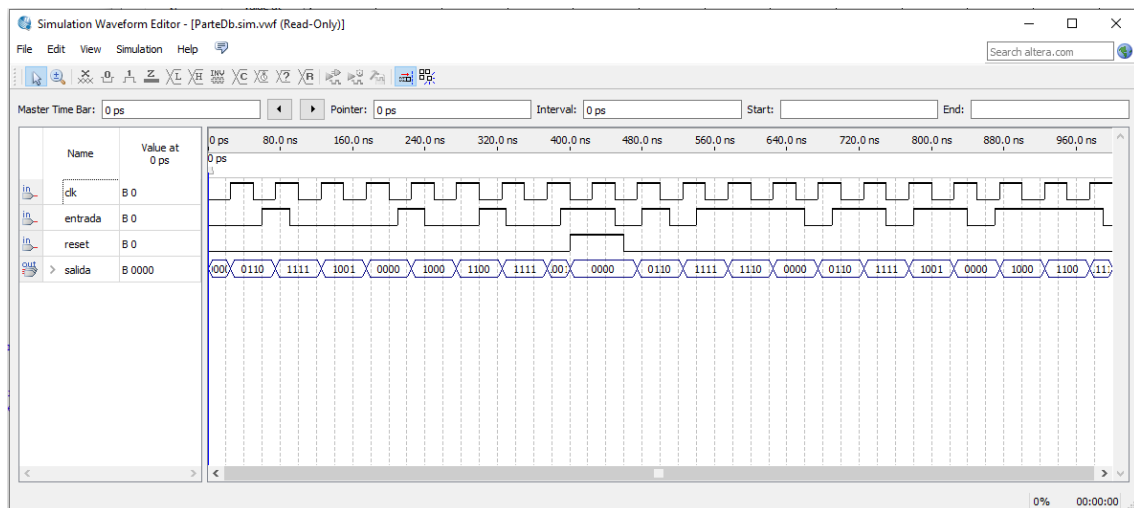
2) Vea la máquina implementada



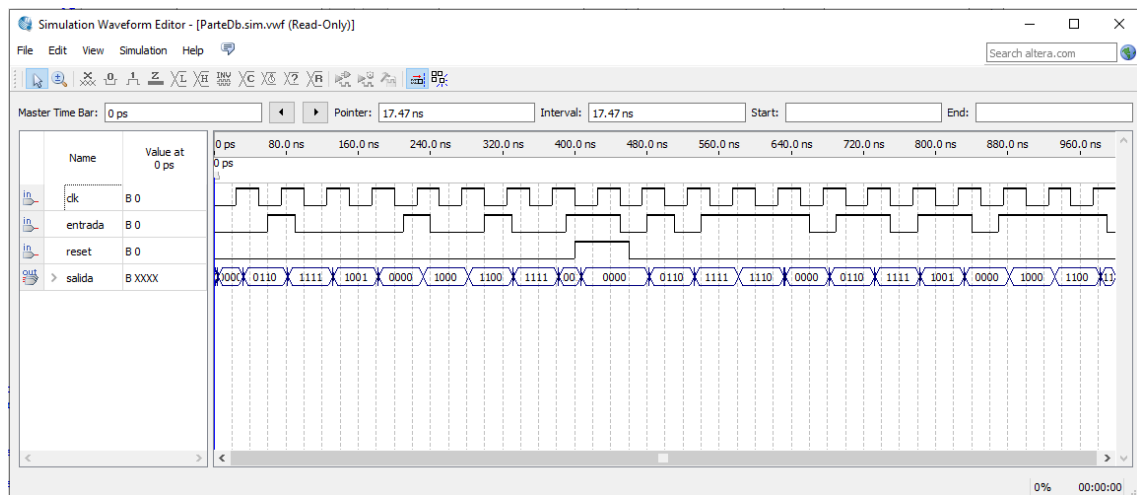
- 3) Cambie la codificación usada para cada estado. Compile nuevamente y vea el circuito implementado.



- 4) Simule para verificar el correcto funcionamiento



Simulación con Run Functional Simulation.



Simulación con Run Timing Simulation