Análisis y Síntesis

Introducción a los Sistemas Lógicos y Digitales

Análisis

Análisis:
Es el procedimiento por el cual mediante alguna técnica de inspección se trata de:
- Conocer qué hace ó cómo funciona un sistema ó circuito.

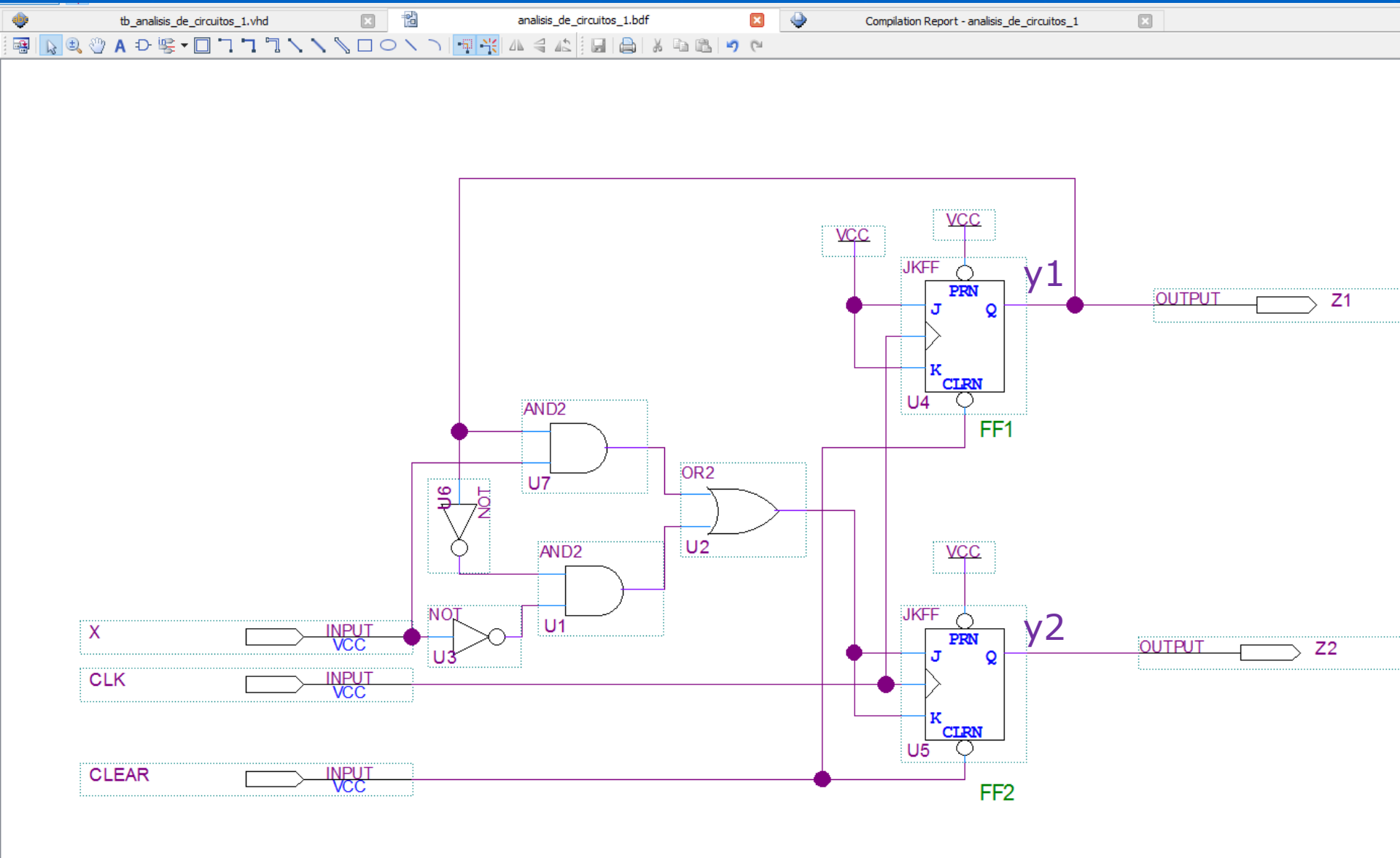
- Verificar su performance ó encontrar la causa de su mal funcionamiento.

Métodos de análisis:

- Tabla de verdad.
- Ecuaciones lógicas.
- Diagramas de estado.
- Simulación.
- Test del hardware.
- etc.....

Análisis

ANÁLISIS mediante el desarrollo de ecuaciones



Análisis

ANÁLISIS mediante el desarrollo de ecuaciones

Para un Flip-Flop JK con entradas J y K y salida "y" se plantea la siguiente ecuación que rige su funcionamiento:

Ecuación general del flip-flop JK: $y(n+1) = J\,\overline{y(n)} + \overline{K}\,y(n)$

Esta ecuación dictamina como se va a comportar su salida "y" (ó Q) cuando venga un flanco de reloj activo. Su valor no sólo puede depender de J y K, sino de lo que previamente valía la salida antes de recibir un nuevo flanco activo.
A la salida actual se la denomina con el sufijo (n) y a la que se actualiza con un nuevo flanco, (n+1).

Del circuito anterior se puede deducir lo siguiente:

$$\text{Si } J = K \Rightarrow y(n+1) = J \cdot \overline{y(n)} + \overline{J} \cdot y(n) = J \oplus y(n)$$

$$J1 = K1 = 1 \Rightarrow y1(n+1) = \overline{y1(n)}$$

$$J2 = K2 = \overline{x \oplus y1(n)} = H \begin{cases} x = 0 \Rightarrow H = \overline{y1(n)} \\ x = 1 \Rightarrow H = y1(n) \end{cases}$$

$$y2(n+1) = H \cdot \overline{y2(n)} + \overline{H} \cdot y2(n) = H \oplus y2(n)$$

$$y2(n+1) = \overline{y1(n)} \oplus y2(n) \text{ para } x = 0$$

$$y2(n+1) = y1n) \oplus y2(n) \text{ para } x = 1$$

Análisis

ANÁLISIS mediante el desarrollo de ecuaciones

De las ecuaciones de y1(n+1) e y2(n+1) se desprende que:

Independiente de X, la salida Z1 = y1, cambia de estado cada vez que se detecta un flanco activo de CLK: y1(n+1) = $\overline{y1(n)}$.

Cuando X=0, la salida y2 negará su estado anterior cada vez que venga un flanco activo de CLK y sea y1=0: (y2(n+1)=$\overline{y2(n)}$), pero mantendrá su estado anterior cuando sea y1=1: ($y2(n+1) = y2(n)$).

Cuando X=1, la salida y2 negará su estado anterior cada vez que venga un flanco activo de CLK y sea y1=1: (y2(n+1)=$\overline{y2(n)}$), pero mantendrá su estado anterior cuando sea y1=0: ($y2(n+1) = y2(n)$).

Para ambos casos de X, y2 dará como resultado una onda cuadrada de doble de período que la salida z1=y1. La diferencia entre X=0 y X=1 es que la salida z2=y2 en un caso cambiará de estado cada vez que baje la salida y1 (X=1) y en el otro caso (X=0) al subir dicha salida y1.

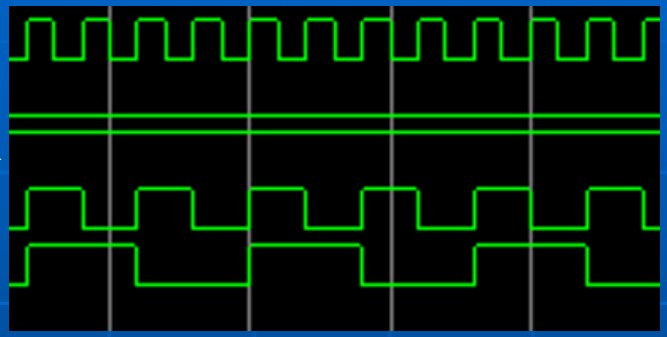
Se logra entonces un desfase de 90 grados entre ambos casos.

Análisis

ANÁLISIS mediante el desarrollo de ecuaciones

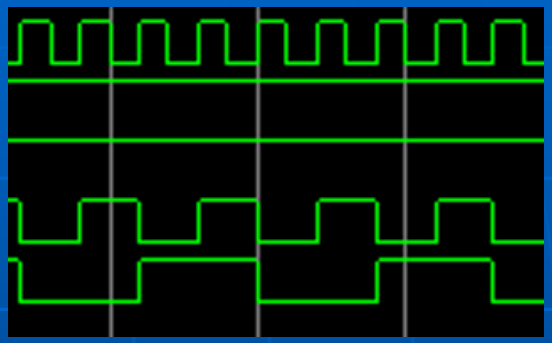
CLK
X=0
CLEAR

Z1
Z2



CLK
X=1
CLEAR

Z1
Z2



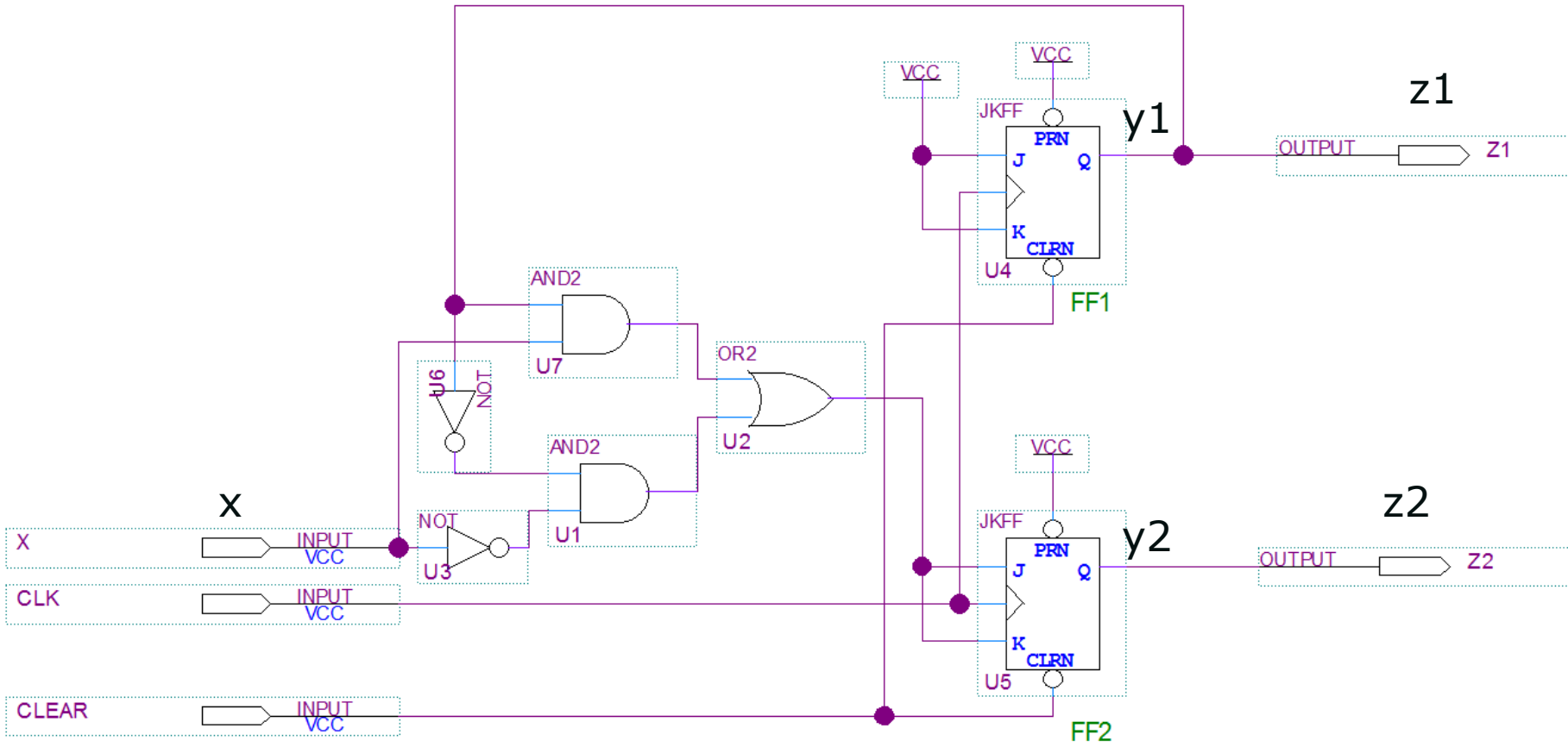
Se puede concluir que el circuito representa un contador binario de dos bits Z2 Z1, con entrada de reset asincrónico (CLEAR) y una entrada de control "X" que define el sentido de conteo en forma ascendente (X=1) ó descendente (X=0).

Análisis

ANÁLISIS mediante herramientas de software (ejemplo empleando Quartus II y Modelsim)

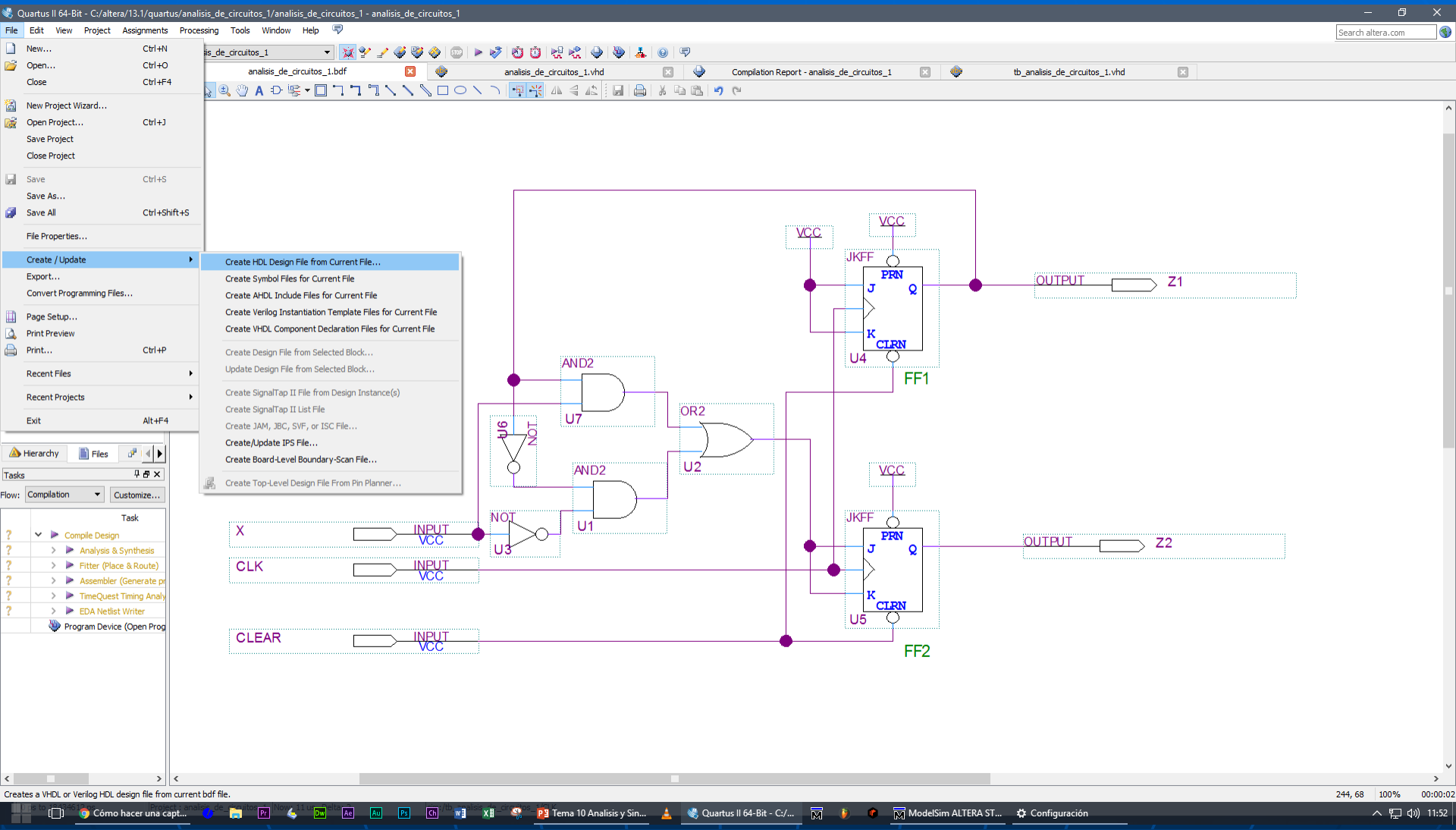
tb_analisis_de_circuitos_1.vhd | analisis_de_circuitos_1.bdf | Compilation Report - analisis_de_circuitos_1

El mismo circuito anterior se lo analizará empleando herramientas de captura y simulación.



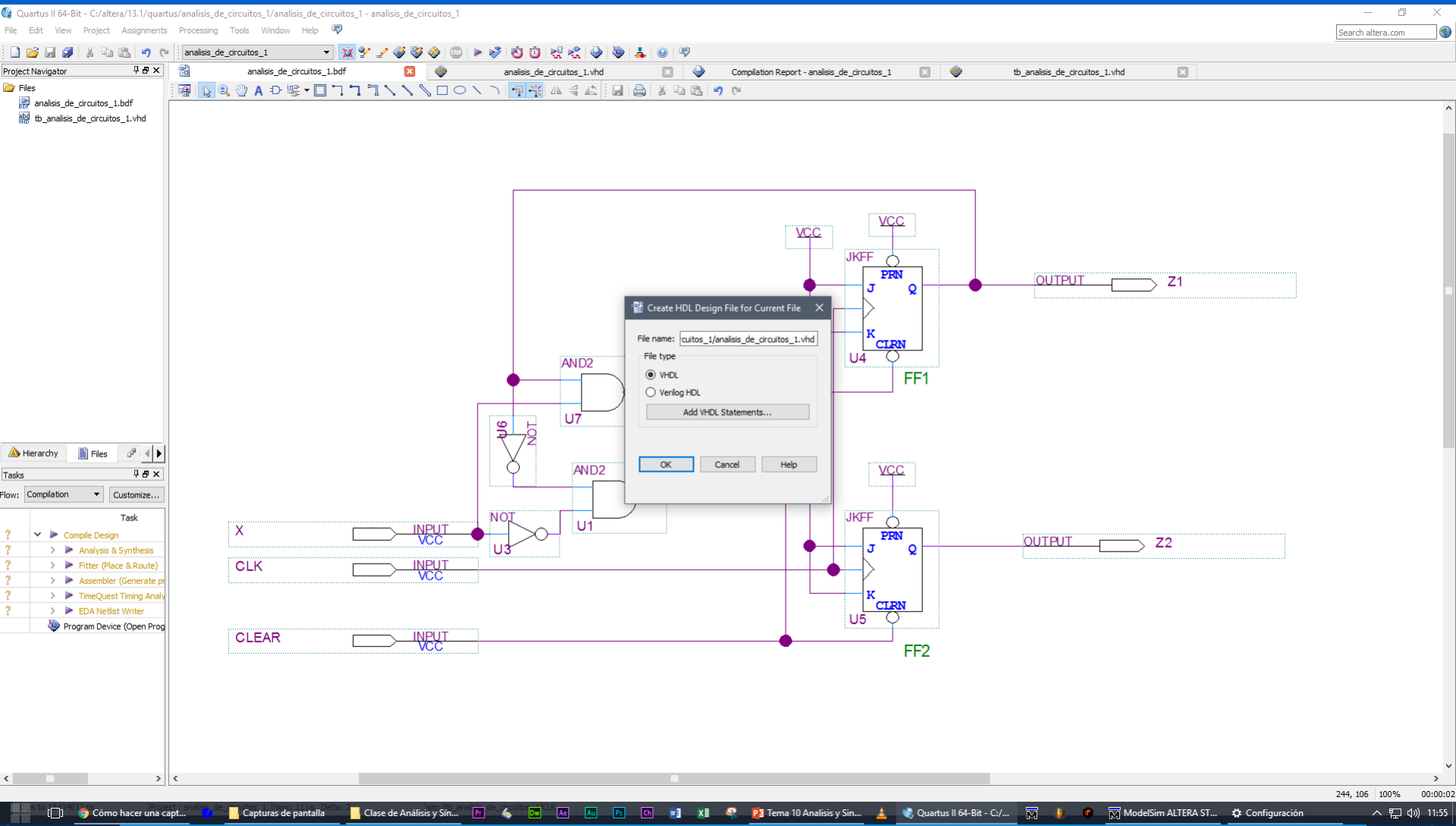
Análisis

Uso de Quartus II y Modelsim



Generación de esquemático y posterior conversión a formato HDL

Análisis



Generación de esquemático y posterior conversión a formato HDL

Análisis

analisis_de_circuitos_1.vhd | Compilation Report - analisis_de_circuitos_1

```
1  -- Copyright (C) 1991-2013 Altera Corporation
2  -- Your use of Altera Corporation's design tools, logic functions
3  -- and other software and tools, and its AMPP partner logic
4  -- functions, and any output files from any of the foregoing
5  -- (including device programming or simulation files), and any
6  -- associated documentation or information are expressly subject
7  -- to the terms and conditions of the Altera Program License
8  -- Subscription Agreement, Altera MegaCore Function License
9  -- Agreement, or other applicable license agreement, including,
10 -- without limitation, that your use is for the sole purpose of
11 -- programming logic devices manufactured by Altera and sold by
12 -- Altera or its authorized distributors.  Please refer to the
13 -- applicable agreement for further details.
14
15 -- PROGRAM     "Quartus II 64-Bit"
16 -- VERSION     "Version 13.1.0 Build 162 10/23/2013 SJ Web Edition"
17 -- CREATED     "Fri Apr 17 10:38:02 2020"
18
19 LIBRARY ieee;
20 USE ieee.std_logic_1164.all;
21
22 LIBRARY work;
23
24 ENTITY analisis_de_circuitos_1 IS
25    PORT
26    (
27       CLK :  IN  STD_LOGIC;
28       X :  IN  STD_LOGIC;
29       CLEAR :  IN  STD_LOGIC;
30       Z1 :  OUT  STD_LOGIC;
31       Z2 :  OUT  STD_LOGIC
32    );
33 END analisis_de_circuitos_1;
34
35 ARCHITECTURE bdf_type OF analisis_de_circuitos_1 IS
36
37 SIGNAL   SYNTHESIZED_WIRE_0 :  STD_LOGIC;
38 SIGNAL   SYNTHESIZED_WIRE_1 :  STD_LOGIC;
39 SIGNAL   SYNTHESIZED_WIRE_2 :  STD_LOGIC;
40 SIGNAL   SYNTHESIZED_WIRE_3 :  STD_LOGIC;
41 SIGNAL   SYNTHESIZED_WIRE_4 :  STD_LOGIC;
42 SIGNAL   SYNTHESIZED_WIRE_10 :  STD_LOGIC;
43 SIGNAL   SYNTHESIZED_WIRE_7 :  STD_LOGIC;
44 SIGNAL   SYNTHESIZED_WIRE_11 :  STD_LOGIC;
45 SIGNAL   SYNTHESIZED_WIRE_12 :  STD_LOGIC;
46
47
48 BEGIN
49 Z1 <= SYNTHESIZED_WIRE_12;
50 SYNTHESIZED_WIRE_4 <= '1';
51 SYNTHESIZED_WIRE_10 <= '1';
52 SYNTHESIZED_WIRE_7 <= '1';
53
```

Archivo VHDL generado por Quartus II en base al esquemático "analisis_de_circuitos_1.bdf"

Análisis

analisis_de_circuitos_1.vhd | Compilation Report - analisis_de_circuitos_1 | tb_analisis_de_circuitos_1.vhd | analisis_de_circuitos_1.bdf

```
50  SYNTHESIZED_WIRE_4 <= '1';
51  SYNTHESIZED_WIRE_10 <= '1';
52  SYNTHESIZED_WIRE_7 <= '1';
53
54
55
56
57
58
59  SYNTHESIZED_WIRE_2 <= SYNTHESIZED_WIRE_0 AND SYNTHESIZED_WIRE_1;
60
61
62  SYNTHESIZED_WIRE_11 <= SYNTHESIZED_WIRE_2 OR SYNTHESIZED_WIRE_3;
63
64
65  SYNTHESIZED_WIRE_1 <= NOT(X);
66
67
68
69  PROCESS(CLK,CLEAR,SYNTHESIZED_WIRE_4)
70  VARIABLE synthesized_var_for_SYNTHESIZED_WIRE_12 : STD_LOGIC;
71  BEGIN
72  IF (CLEAR = '0') THEN
73     synthesized_var_for_SYNTHESIZED_WIRE_12 := '0';
74  ELSIF (SYNTHESIZED_WIRE_4 = '0') THEN
75     synthesized_var_for_SYNTHESIZED_WIRE_12 := '1';
76  ELSIF (RISING_EDGE(CLK)) THEN
77     synthesized_var_for_SYNTHESIZED_WIRE_12 := (NOT(synthesized_var_for_SYNTHESIZED_WIRE_12) AND SYNTHESIZED_WIRE_10) OR (synthesized_var_for_SYNTHESIZED_WIRE_12 AND (NOT(SYNTHESIZED_WIRE_10)));
78  END IF;
79     SYNTHESIZED_WIRE_12 <= synthesized_var_for_SYNTHESIZED_WIRE_12;
80  END PROCESS;
81
82
83  PROCESS(CLK,CLEAR,SYNTHESIZED_WIRE_7)
84  VARIABLE synthesized_var_for_Z2 : STD_LOGIC;
85  BEGIN
86  IF (CLEAR = '0') THEN
87     synthesized_var_for_Z2 := '0';
88  ELSIF (SYNTHESIZED_WIRE_7 = '0') THEN
89     synthesized_var_for_Z2 := '1';
90  ELSIF (RISING_EDGE(CLK)) THEN
91     synthesized_var_for_Z2 := (NOT(synthesized_var_for_Z2) AND SYNTHESIZED_WIRE_11) OR (synthesized_var_for_Z2 AND (NOT(SYNTHESIZED_WIRE_11)));
92  END IF;
93     Z2 <= synthesized_var_for_Z2;
94  END PROCESS;
95
96
97  SYNTHESIZED_WIRE_0 <= NOT(SYNTHESIZED_WIRE_12);
98
99
100
101 SYNTHESIZED_WIRE_3 <= SYNTHESIZED_WIRE_12 AND X;
102
103
104 END bdf_type;
```

Archivo VHDL generado por Quartus II en base al esquemático "analisis_de_circuitos_1.bdf"

Análisis

tb_analisis_de_circuitos_1.vhd | analisis_de_circuitos_1.bdf

```
1  --Test-bench del proyecto analisis_de_circuitos_1 Sergio Noriega 2020
2  --Archivo: tb_analisis_de_circuitos_1.vhd.
3
4   library ieee;
5   use ieee.std_logic_1164.all;
6
7  entity tb_analisis_de_circuitos_1 is
8  end tb_analisis_de_circuitos_1;
9
10 architecture test of tb_analisis_de_circuitos_1 is
11    component analisis_de_circuitos_1
12    PORT
13    (
14       CLK :  IN  STD_LOGIC;
15       X :  IN  STD_LOGIC;
16       CLEAR :  IN  STD_LOGIC;
17       Z1 :  OUT  STD_LOGIC;
18       Z2 :  OUT  STD_LOGIC
19    );
20    end component;
21
22    signal CLK, X, CLEAR : STD_LOGIC;
23    signal Z1, Z2 : std_LOGIC;
24
25
26  begin
27
28    uut: analisis_de_circuitos_1 port map( CLK => CLK, X => X, CLEAR => CLEAR,
29                                           Z1 => Z1, Z2 => Z2);
30
31
32    gen_reloj : process -- Reloj de 200 ns de periodo y 50% de ciclo de trabajo
33       begin
34
35          CLK <= '0';
36          wait for 100 ns;
37          CLK <= '1';
38          wait for 100 ns;
39    end process gen_reloj;
40
41    estimulos : process
42    begin
43       X <= '0';
44       CLEAR <= '1';
45       wait for 100 ns;
46       CLEAR <= '0';
47       wait for 500 ns;
48       CLEAR <= '1';
49       wait for 5 us;
50       X <= '1';
51       wait for 5 us;
52    end process estimulos;
53 end test;
```

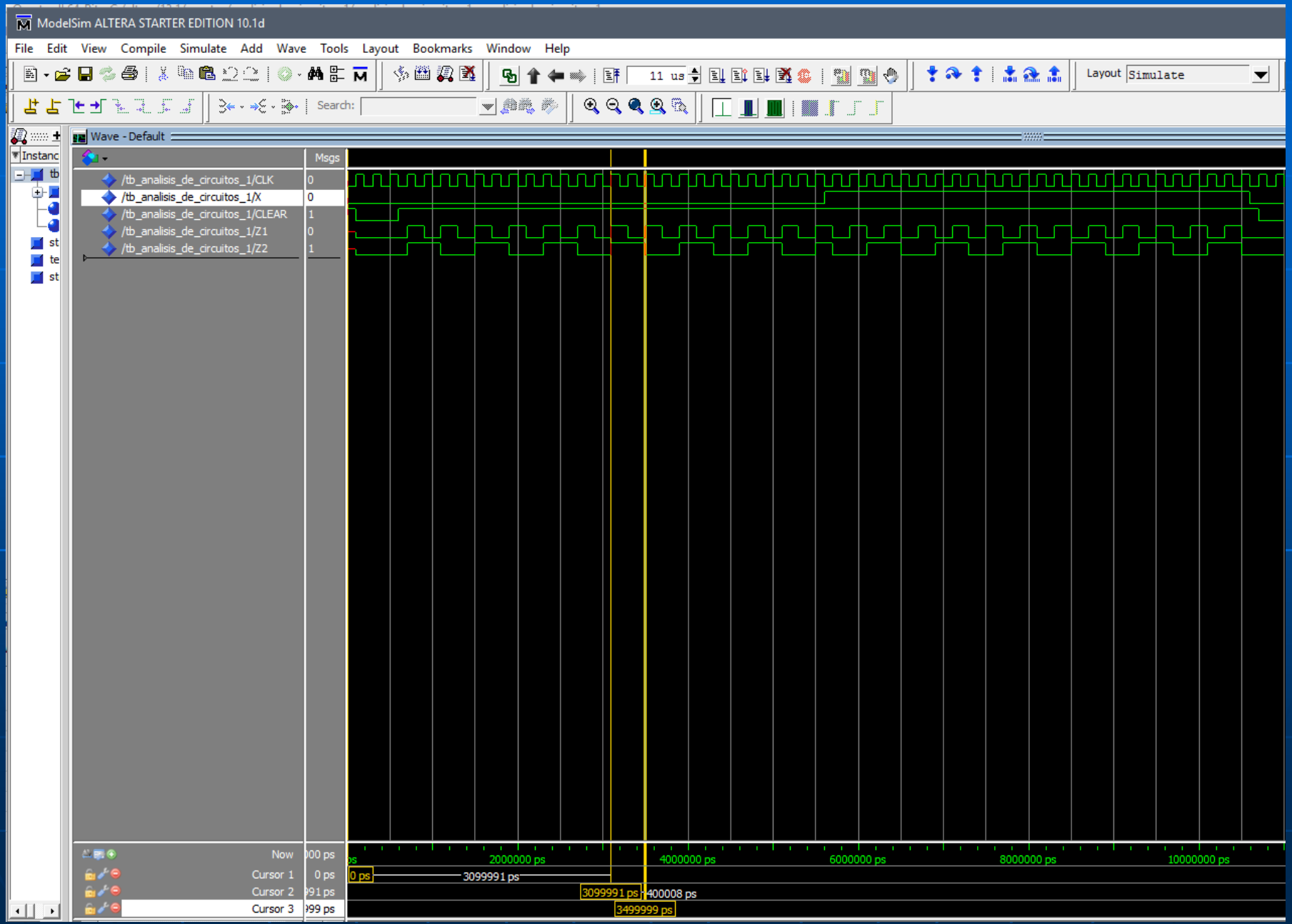
Archivo VHDL creado para hacer el test bench junto con el descripto en "analisis_de_circuitos_1.vhd".

Se genera un reloj de 200 ns de período.
En paralelo se genera una entrada para CLEAR, generando un pulso negativo de 500 ns de duración.
La señal X, en t=0 ns se pone en "0" y luego en 5,6 us se pasa a "1".

Con ambos archivos se entra a Modelsim y se observa después el resultado de la simulación en los diagramas de tiempo.

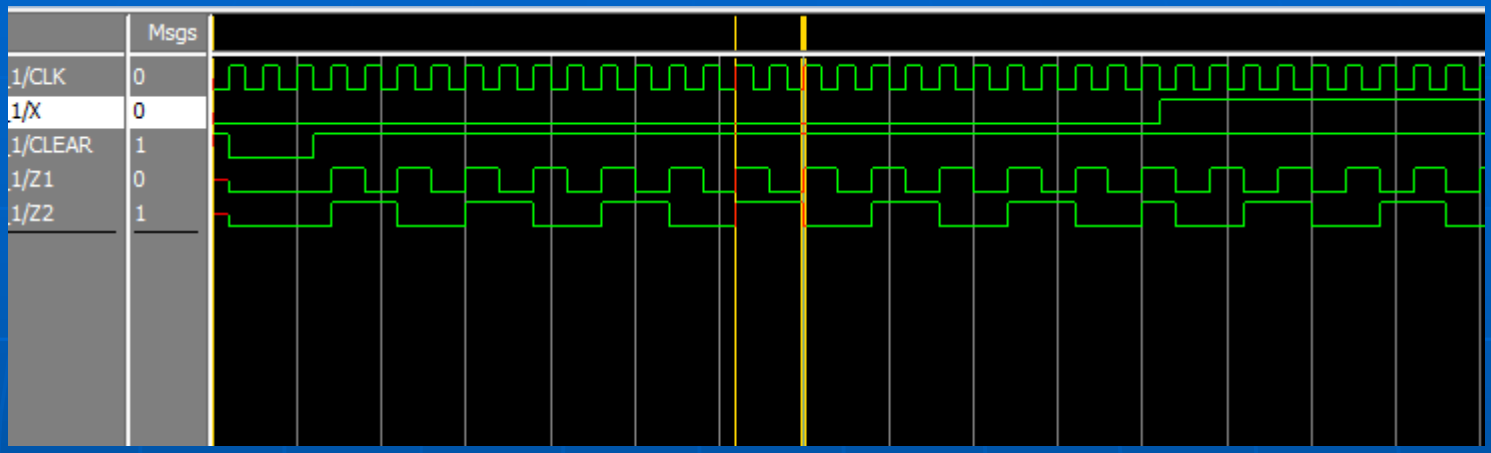
Análisis

Respuesta de Modelsim a "analisis_de_circuitos_1"



Análisis

Respuesta de Modelsim a "analisis_de_circuitos_1"



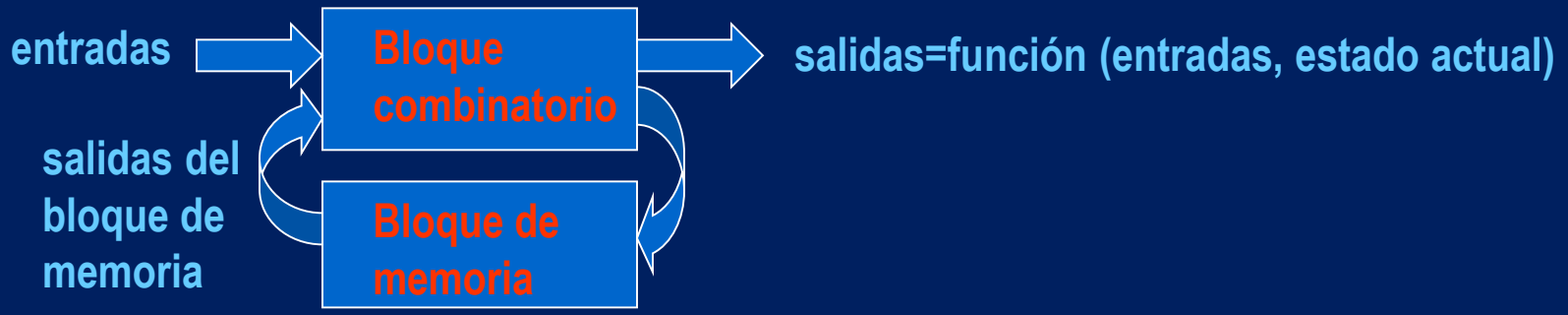
Por inspección de los diagramas de tiempo, el circuito parece ser un contador binario de 2 bits que cuenta los flancos de subida de una señal de reloj CLK y los refleja en las salidas Z2 Z1, siendo Z2 el bit MSB (más significativo).
Tiene una entrada de CLEAR que borra las salidas en estado bajo, en forma asincrónica y otra entrada X que controla el sentido de conteo:
si X=0 cuenta en forma regresiva y si X=1 en forma progresiva.

Análisis

Máquinas de estado

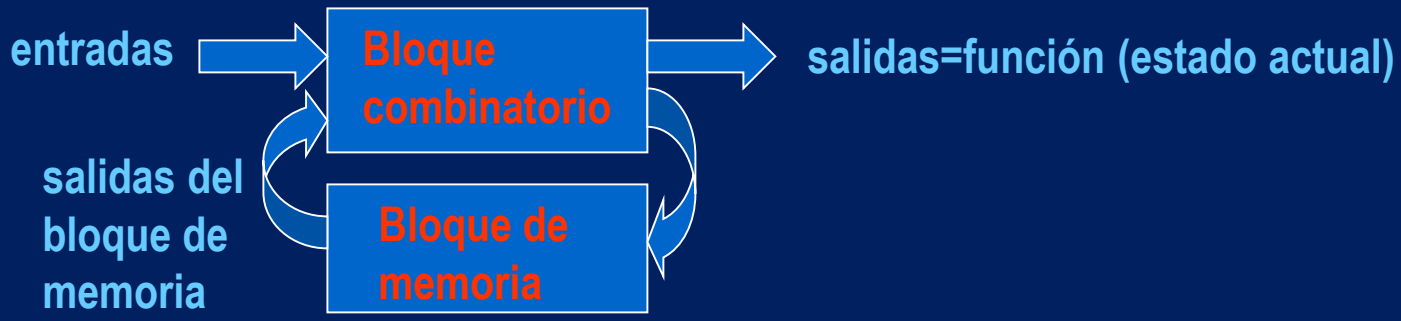
Modelo de Mealy

La(s) salida(s) depende(n) de la(s) entrada(s) y del estado actual.



Modelo de Moore

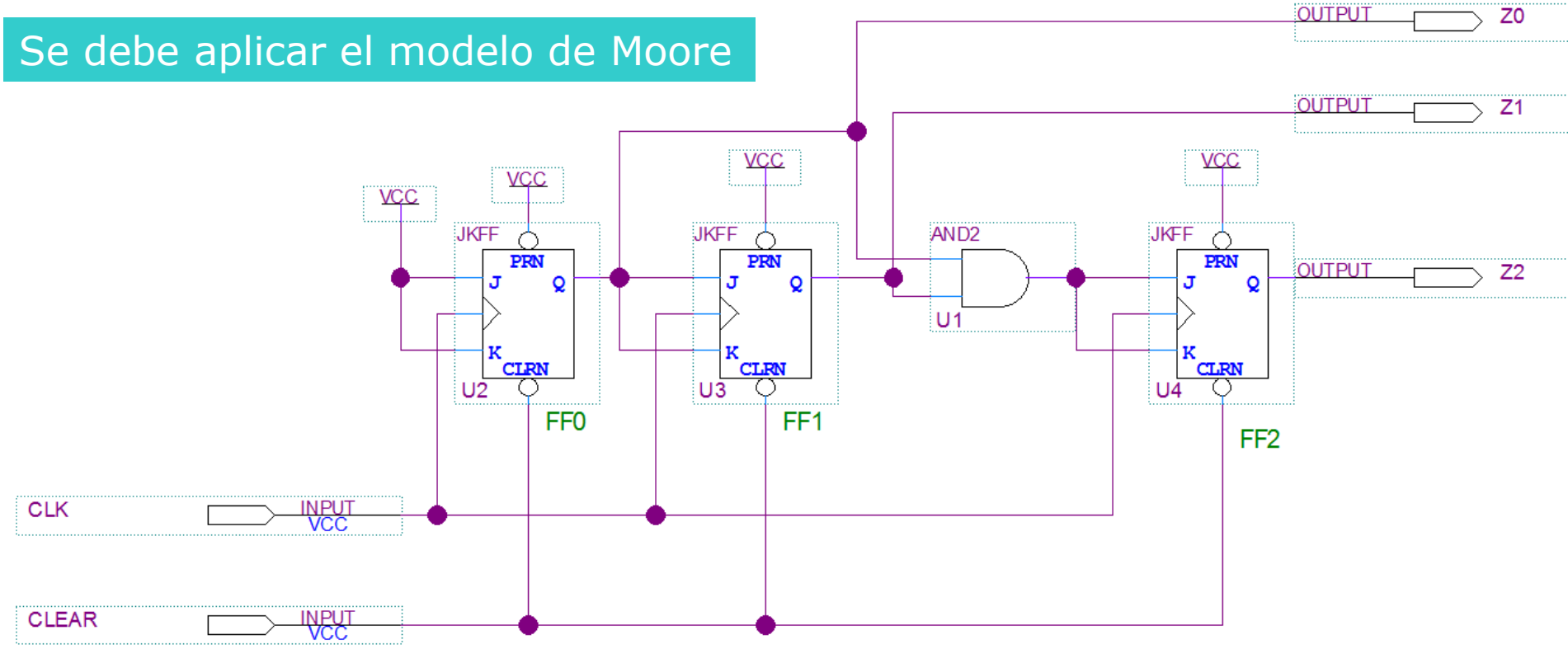
La(s) salida(s) depende(n) del del estado actual.



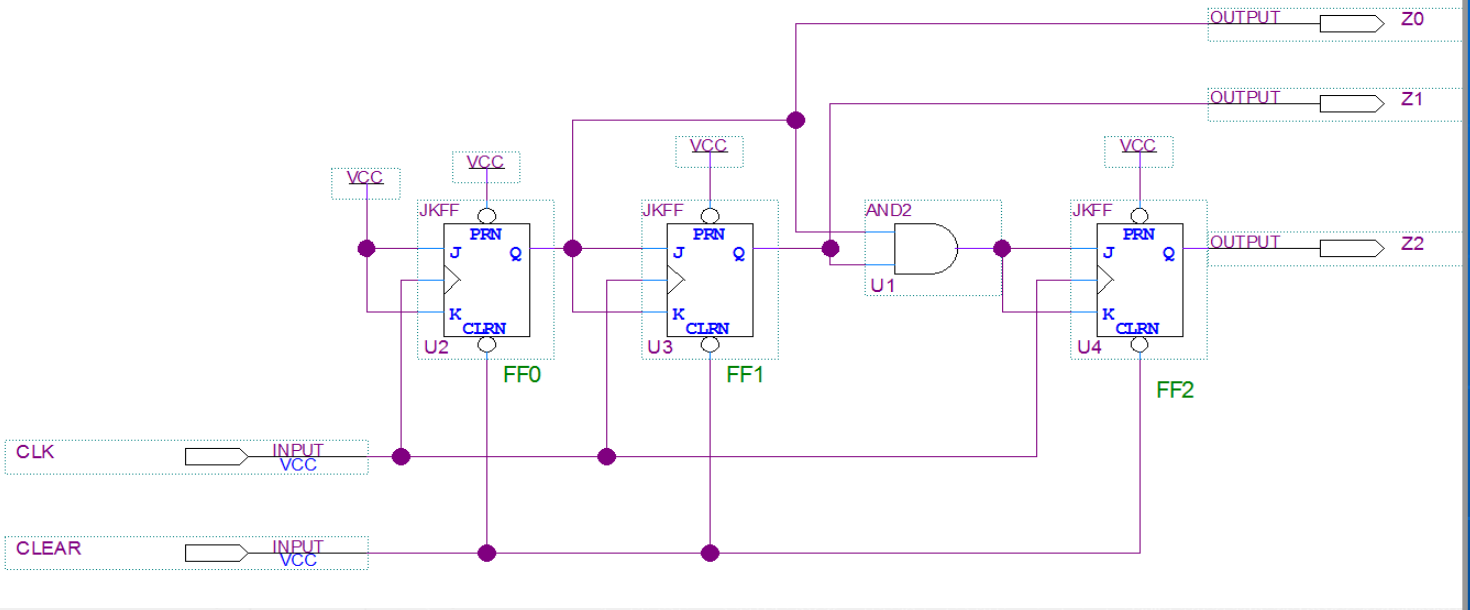
Análisis

Ejemplo de análisis por diagrama de estados

Se debe aplicar el modelo de Moore



Análisis



$$FF_0: \quad Q_0^{n+1} = \overline{Q_0^n} \cdot J_0 + Q_0^n \cdot \overline{K_0} = \overline{Q_0^n} \cdot J_0 + Q_0^n \cdot \overline{J_0} = Q_0^n \oplus J_0$$

$$J_0 = K_0 = 1 \Rightarrow Q_0^{n+1} = \overline{Q_0^n}$$

$$FF_1: \quad Q_1^{n+1} = \overline{Q_1^n} \cdot J_1 + Q_1^n \cdot \overline{K_1} = \overline{Q_1^n} \cdot J_1 + Q_1^n \cdot \overline{J_1} = Q_1^n \oplus J_1$$

$$J_1 = K_1 = Q_0^n \Rightarrow Q_1^{n+1} = Q_1^n \oplus Q_0^n$$

Si $Q_0^n = 0 \rightarrow Q_1^{n+1} = Q_1^n$

Si $Q_0^n = 1 \rightarrow Q_1^{n+1} = \overline{Q_1^n}$

Análisis

$$FF_2 : Q_2^{n+1} = \overline{Q_2^n} \cdot J_2 + Q_2^n \cdot \overline{K_2} = \overline{Q_2^n} \cdot J_2 + Q_2^n \cdot \overline{J_2} = Q_2^n \oplus J_2$$

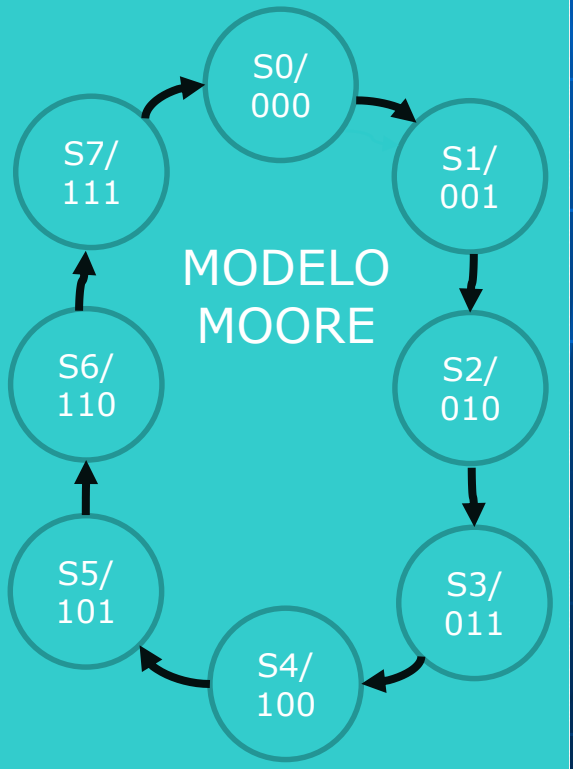
$$J_2 = K_2 = Q_0^n \cdot Q_1^n$$

$$Q_2^{n+1} = Q_2^n \oplus Q_0^n \cdot Q_1^n \Rightarrow \begin{cases} \text{Si } Q_1^n; Q_0^n \neq 11 \rightarrow Q_2^{n+1} = Q_2^n \\ \text{si } Q_1^n; Q_0^n = 11 \rightarrow Q_2^{n+1} = \overline{Q_2^n} \end{cases}$$

Tabla de Estados

	Q2n	Q1n	Q0n	Q2n+1	Q1n+1	Q0n+1
S0	0	0	0	0	0	1
S1	0	0	1	0	1	0
S2	0	1	0	0	1	1
S3	0	1	1	1	0	0
S4	1	0	0	1	0	1
S5	1	0	1	1	1	0
S6	1	1	0	1	1	1
S7	1	1	1	0	0	0

Diagrama de Estados



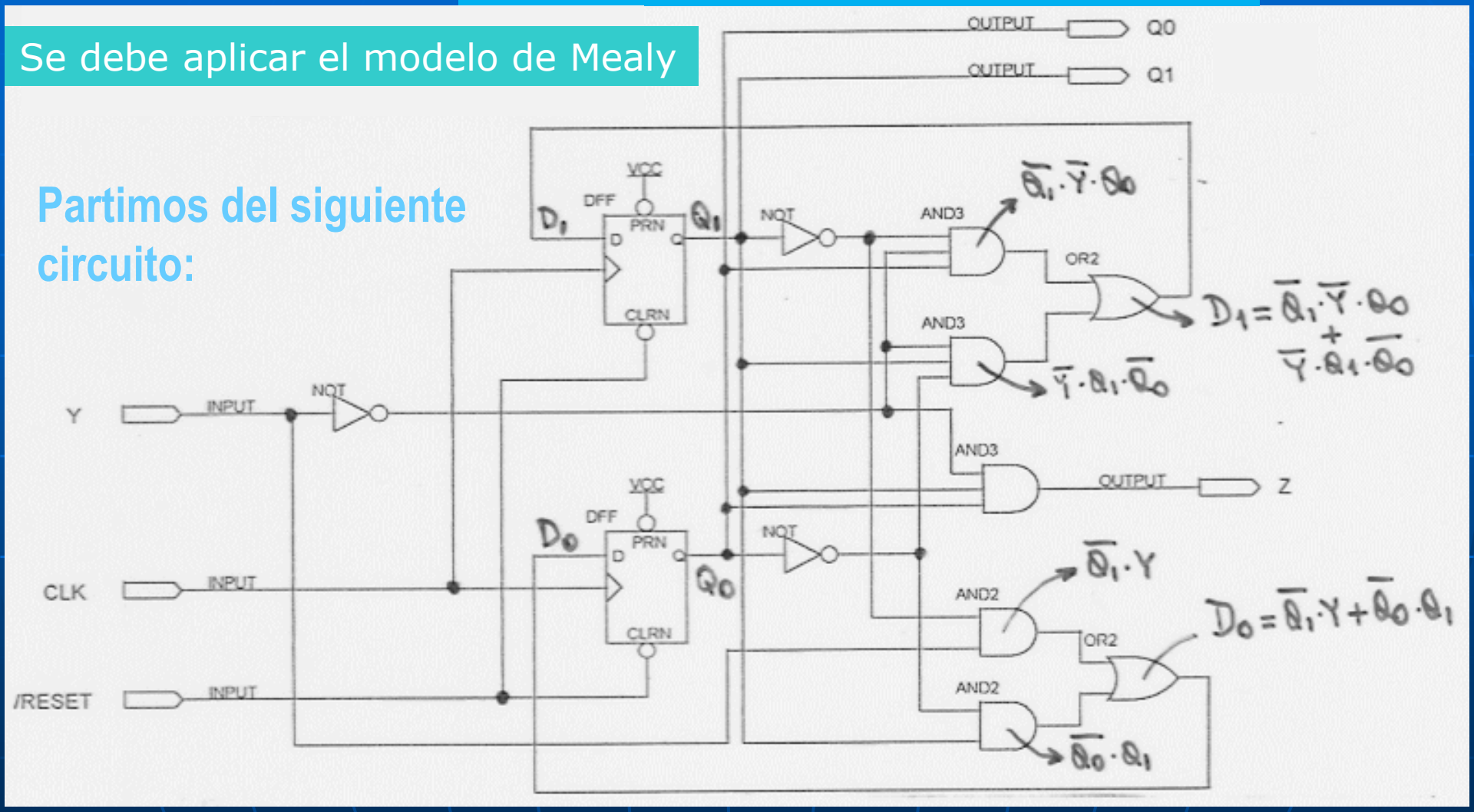
Se trata de un contador binario progresivo de 3 bits sincrónico con entrada de RESET asincrónico.

Análisis

Ejemplo de análisis por diagramas de estados

Se debe aplicar el modelo de Mealy

Partimos del siguiente circuito:



Análisis

$$Z = \bar{Y}\, Q1\, Q0$$

$$D0 = \overline{Q1}\, Y + Q1\, \overline{Q0}$$

$$D1 = \overline{Q1}\, \bar{Y}\, Q0 + \bar{Y}\, Q1\, \overline{Q0} = Y\,(Q1 \oplus Q0)$$

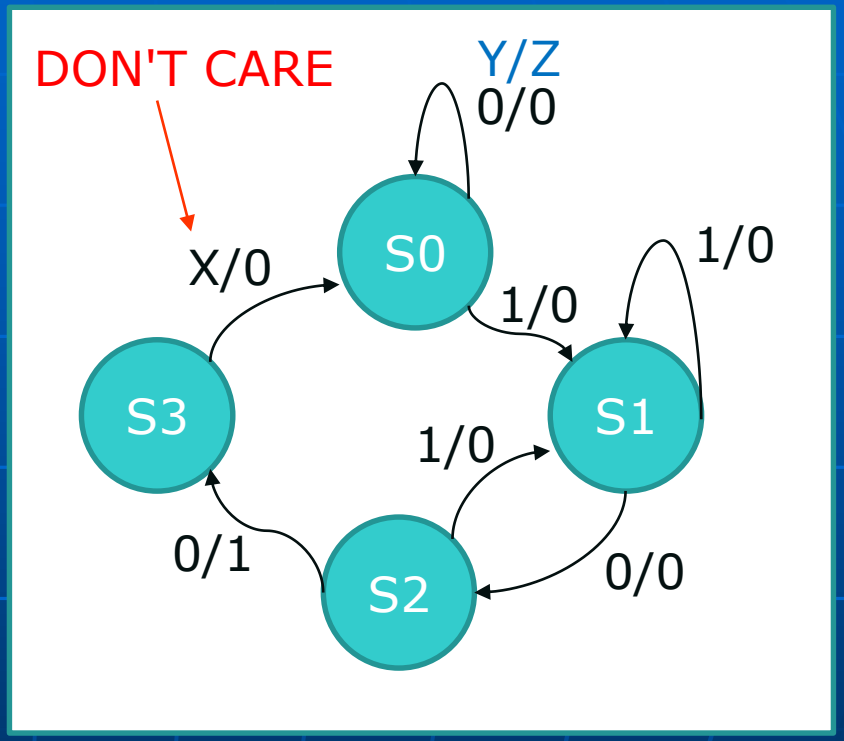
$$Y=0 \begin{cases} D0 = \overline{Q0}\, Q1 \\ D1 = Q1 \oplus Q0 \end{cases}$$

$$Y=1 \begin{cases} D0 = \overline{Q1} + \overline{Q0}\, Q1 \\ D1 = 0 \end{cases}$$

TABLA DE ESTADOS

	Q1n	Q0n	D1	D0	D1	D0	Z	
S0	0	0	0	0	0	1	0	0
S1	0	1	1	0	0	1	0	0
S2	1	0	1	1	0	1	1	0
S3	1	1	0	0	0	0	0	0
Y			0		1		0	1

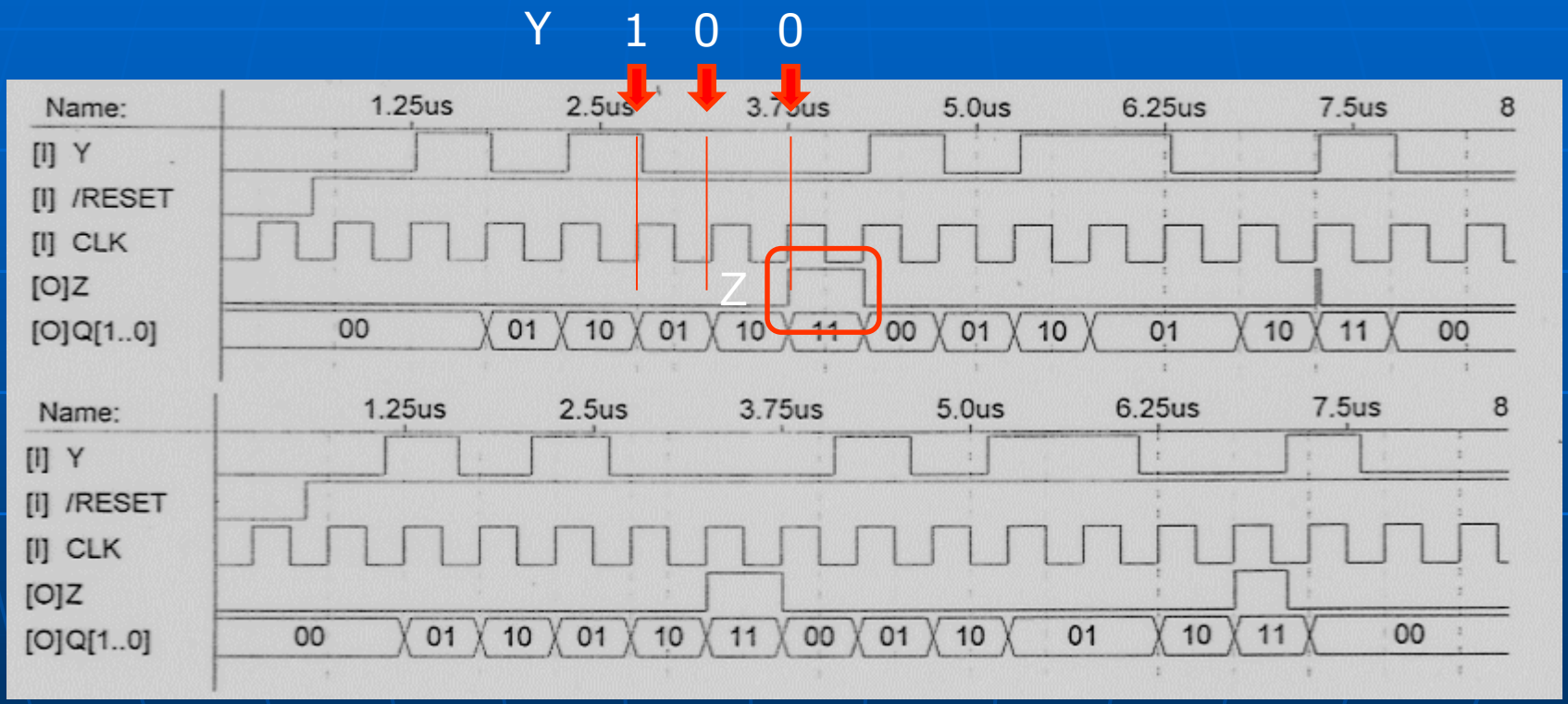
DIAGRAMA DE ESTADOS



Este circuito puede ser un detector de secuencia cíclico (se repite contínuamente) de una entrada serie "Y" tal que si la secuencia es "100", durante un ciclo de reloj la salida "Z" valdrá "1", caso contrario quedará siempre en 0.
Además, el siguiente bit después de detectar la secuencia correcta es descartado.....

Análisis

Simulación del circuito anterior



Análisis

SignalTap II Logic Analyzer

Herramienta empleada cuando es necesario realizar una ***verificación funcional*** del diseño empleando el hardware sintetizado en la FPGA.
Funciona como un analizador lógico donde es posible seleccionar las señales que servirán para disparar el análisis y las que se desean testear.
Las señales de trigger pueden ser internas ó externas de entrada a la FPGA bajo estudio (lo mismo que las señales bajo análisis).
Dado que la información se almacena en memoria se requieren recursos tanto de lógica como de memoria dedicada de la propia FPGA en estudio.
Los resultados pueden verse en el mismo ambiente de diseño en el Quartus y almacenados luego en un archivo para posterior tratamiento.

Ventajas:
Permite realizar un análisis de la respuesta del hardware empleando eventos de disparo provenientes de señales internas ó externas al dispositivo.

Desventajas:
No permite realizar una verificación rigurosa del TIMING. Para ello existen otras herramientas de análisis como por ej. TimeQuest Timing Analyzer.

SIGNAL TAP II

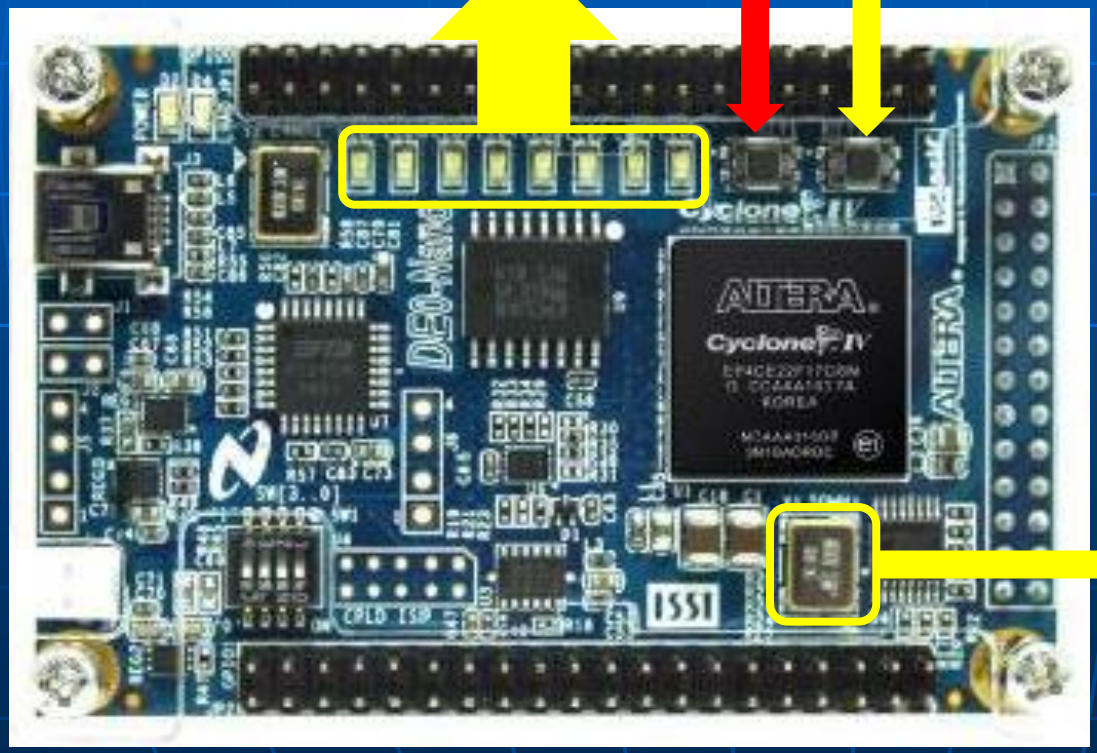
En el siguiente ejemplo se analizará la respuesta de los bits de salida de un contador binario sincrónico de 8 bits, el cual es sintetizado en la FPGA Cyclone IV de la placa DE0-Nano basada el el chip EP4CE22F17C6.
Todo el proceso de análisis se realiza con la placa conectada a la PC.
Se utiliza como trigger la señal de RESET la cual es externa y asignada al pulsador KEY0.
Los 8 bits de salida del contador serán almacenadas en memoria de la FPGA y posteriormente grabados en un archivo.

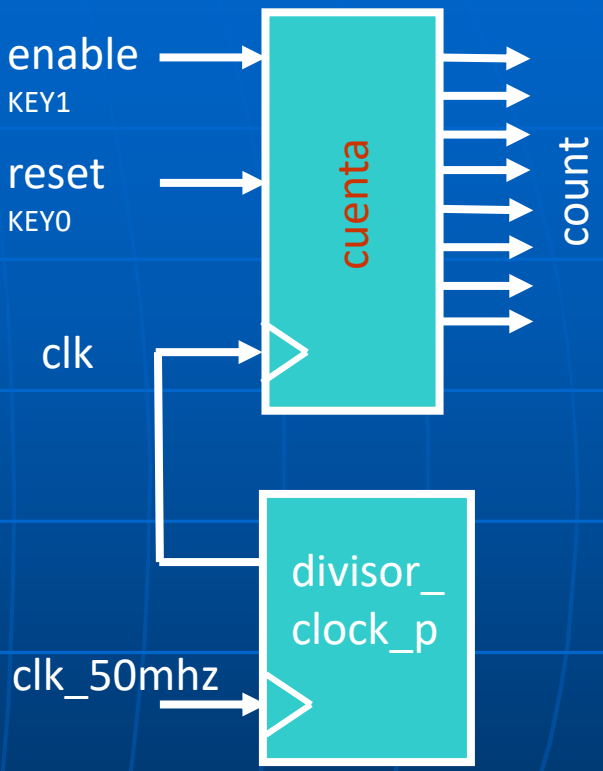
Al iniciar el análisis en el mismo ambiente del Quartus, es posible ver el diagrama temporal con la evolución de las señales a testear.

La cantidad de muestras se puede ajustar y depende de los recursos de la FPGA.
En este caso se eligen 16 Kmuestras.

Para definir el tiempo de muestreo se usa el reloj de 50 MHz de la entrada.

SIGNAL TAP II



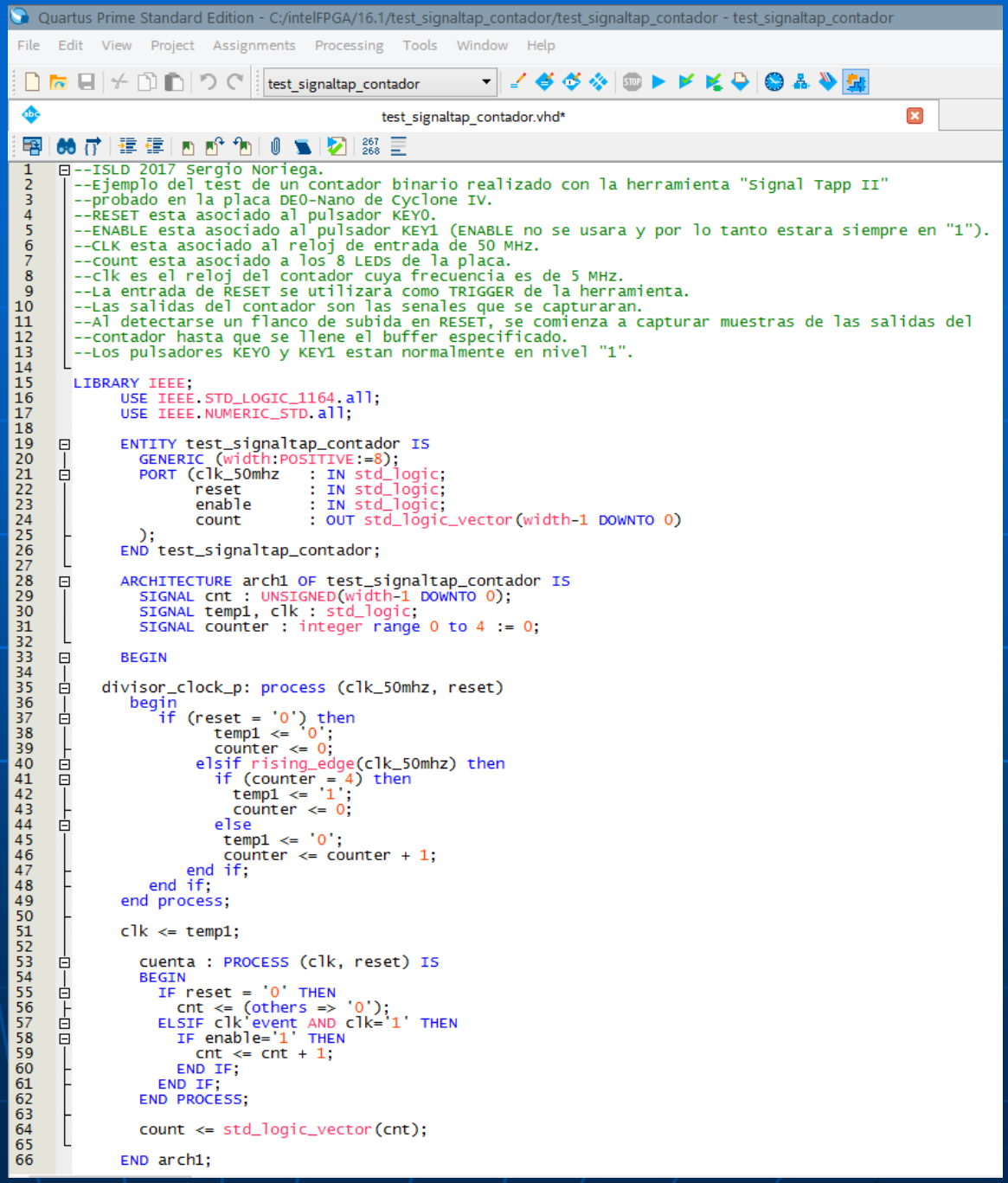


SIGNAL TAP II

A este contador se le hacen las siguientes asignaciones en la placa DE0-Nano:

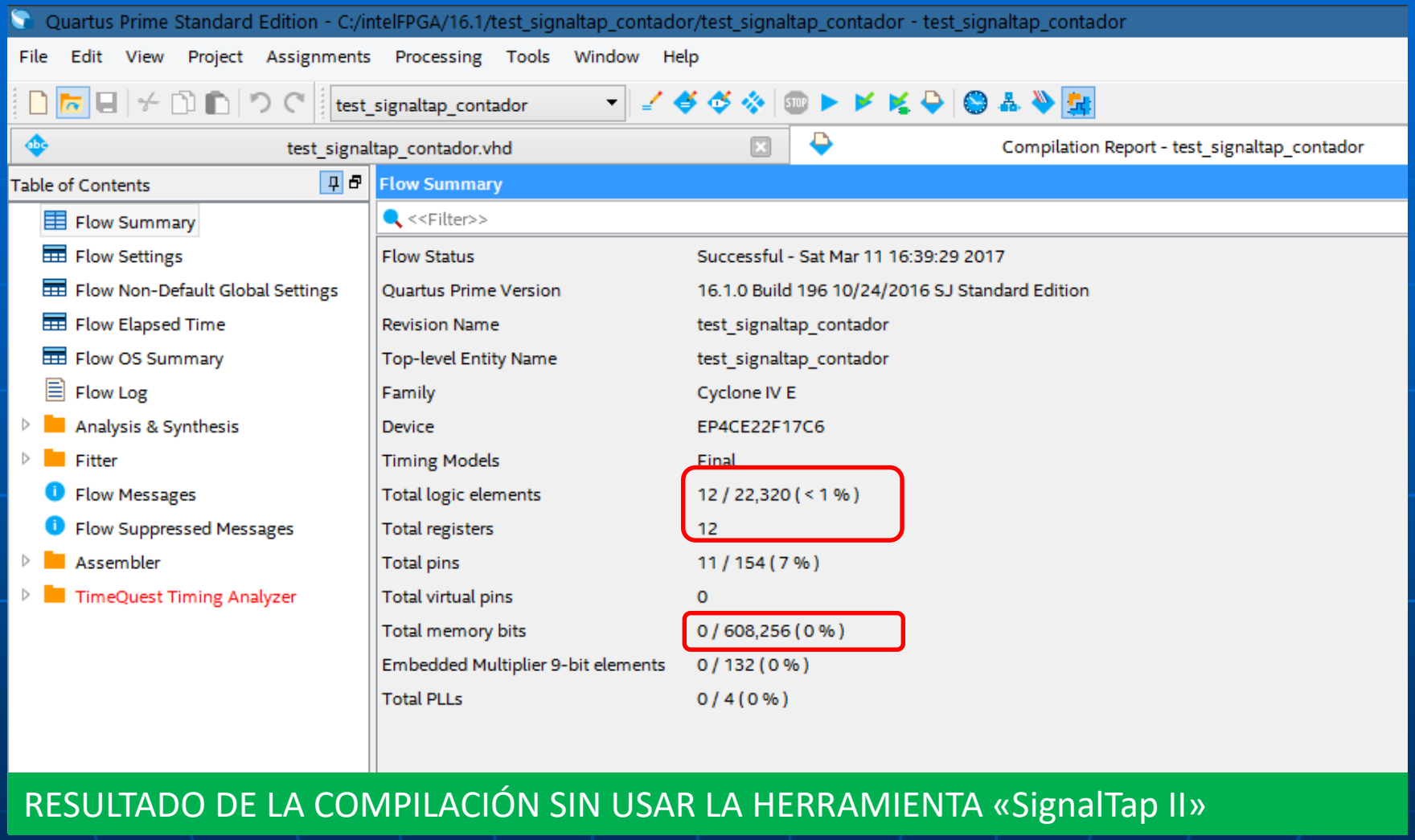
RESET a pulsador KEY0.
ENABLE a pulsador KEY1.
CLK_50MHZ a oscilador 50MHz.
Las 8 Salidas a los 8 LED´s.

KEY0 y KEY1 están siempre en «1» cuando NO se presionan (son pulsadores sin retención).




```
1  --ISLD 2017 Sergio Noriega.
2  --Ejemplo del test de un contador binario realizado con la herramienta "Signal Tapp II"
3  --probado en la placa DE0-Nano de Cyclone IV.
4  --RESET esta asociado al pulsador KEY0.
5  --ENABLE esta asociado al pulsador KEY1 (ENABLE no se usara y por lo tanto estara siempre en "1").
6  --CLK esta asociado al reloj de entrada de 50 MHz.
7  --count esta asociado a los 8 LEDs de la placa.
8  --clk es el reloj del contador cuya frecuencia es de 5 MHz.
9  --La entrada de RESET se utilizara como TRIGGER de la herramienta.
10 --Las salidas del contador son las senales que se capturaran.
11 --Al detectarse un flanco de subida en RESET, se comienza a capturar muestras de las salidas del
12 --contador hasta que se llene el buffer especificado.
13 --Los pulsadores KEY0 y KEY1 estan normalmente en nivel "1".
14
15 LIBRARY IEEE;
16      USE IEEE.STD_LOGIC_1164.all;
17      USE IEEE.NUMERIC_STD.all;
18
19      ENTITY test_signaltap_contador IS
20        GENERIC (width:POSITIVE:=8);
21        PORT (clk_50mhz    : IN std_logic;
22              reset        : IN std_logic;
23              enable       : IN std_logic;
24              count        : OUT std_logic_vector(width-1 DOWNTO 0)
25        );
26      END test_signaltap_contador;
27
28      ARCHITECTURE arch1 OF test_signaltap_contador IS
29        SIGNAL cnt : UNSIGNED(width-1 DOWNTO 0);
30        SIGNAL temp1, clk : std_logic;
31        SIGNAL counter : integer range 0 to 4 := 0;
32
33      BEGIN
34
35     divisor_clock_p: process (clk_50mhz, reset)
36       begin
37          if (reset = '0') then
38                temp1 <= '0';
39                counter <= 0;
40             elsif rising_edge(clk_50mhz) then
41               if (counter = 4) then
42                 temp1 <= '1';
43                 counter <= 0;
44               else
45                temp1 <= '0';
46                counter <= counter + 1;
47            end if;
48         end if;
49      end process;
50
51      clk <= temp1;
52
53        cuenta : PROCESS (clk, reset) IS
54        BEGIN
55          IF reset = '0' THEN
56            cnt <= (others => '0');
57          ELSIF clk'event AND clk='1' THEN
58            IF enable='1' THEN
59              cnt <= cnt + 1;
60            END IF;
61          END IF;
62        END PROCESS;
63
64        count <= std_logic_vector(cnt);
65
66      END arch1;
```

SIGNAL TAP II

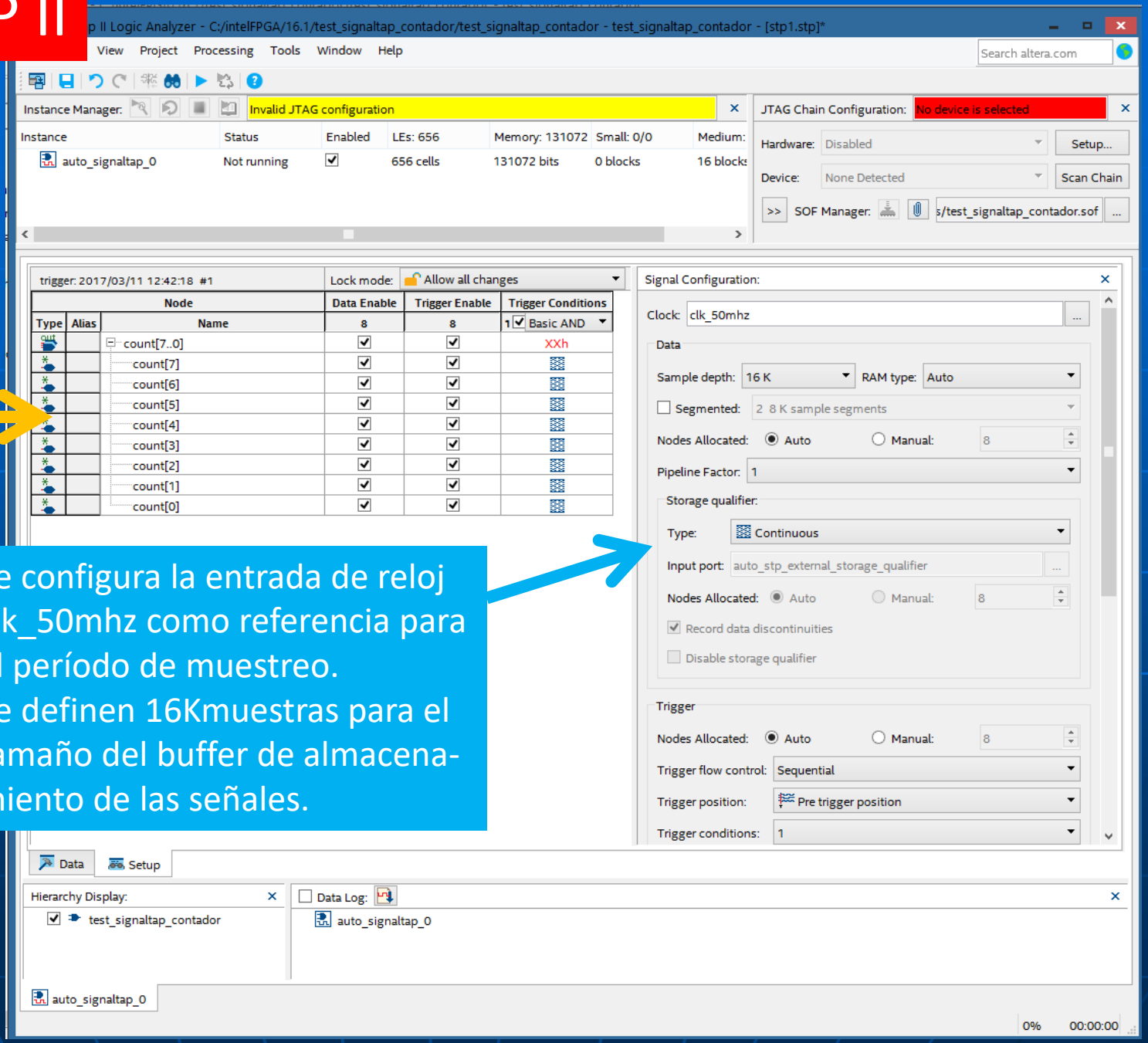


RESULTADO DE LA COMPILACIÓN SIN USAR LA HERRAMIENTA «SignalTap II»

SIGNAL TAP II

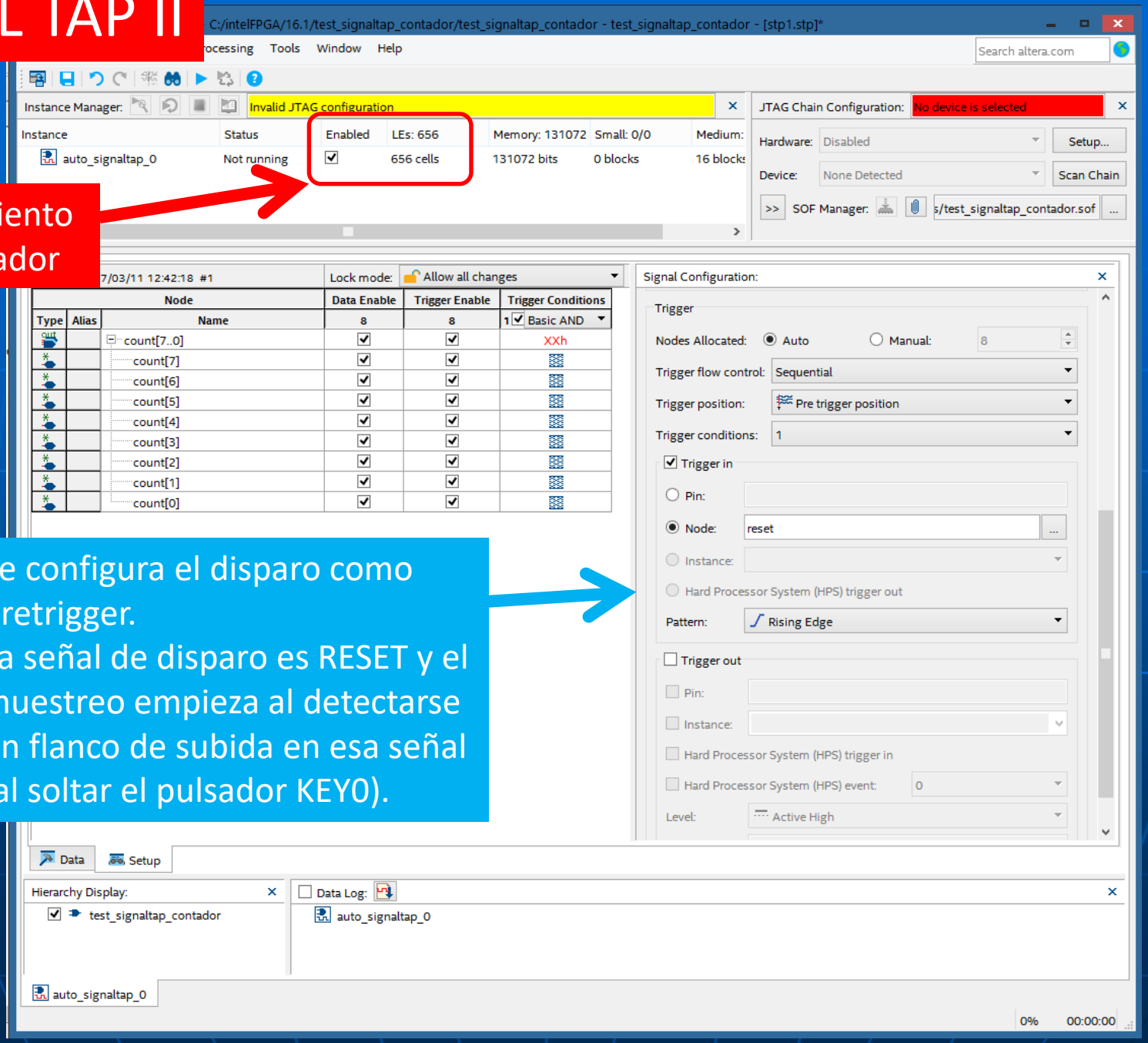
Se definen las señales usadas para su análisis que son las 8 salidas del contador count

Se configura la entrada de reloj clk_50mhz como referencia para el período de muestreo.
Se definen 16Kmuestras para el tamaño del buffer de almacena-miento de las señales.



SIGNAL TAP II

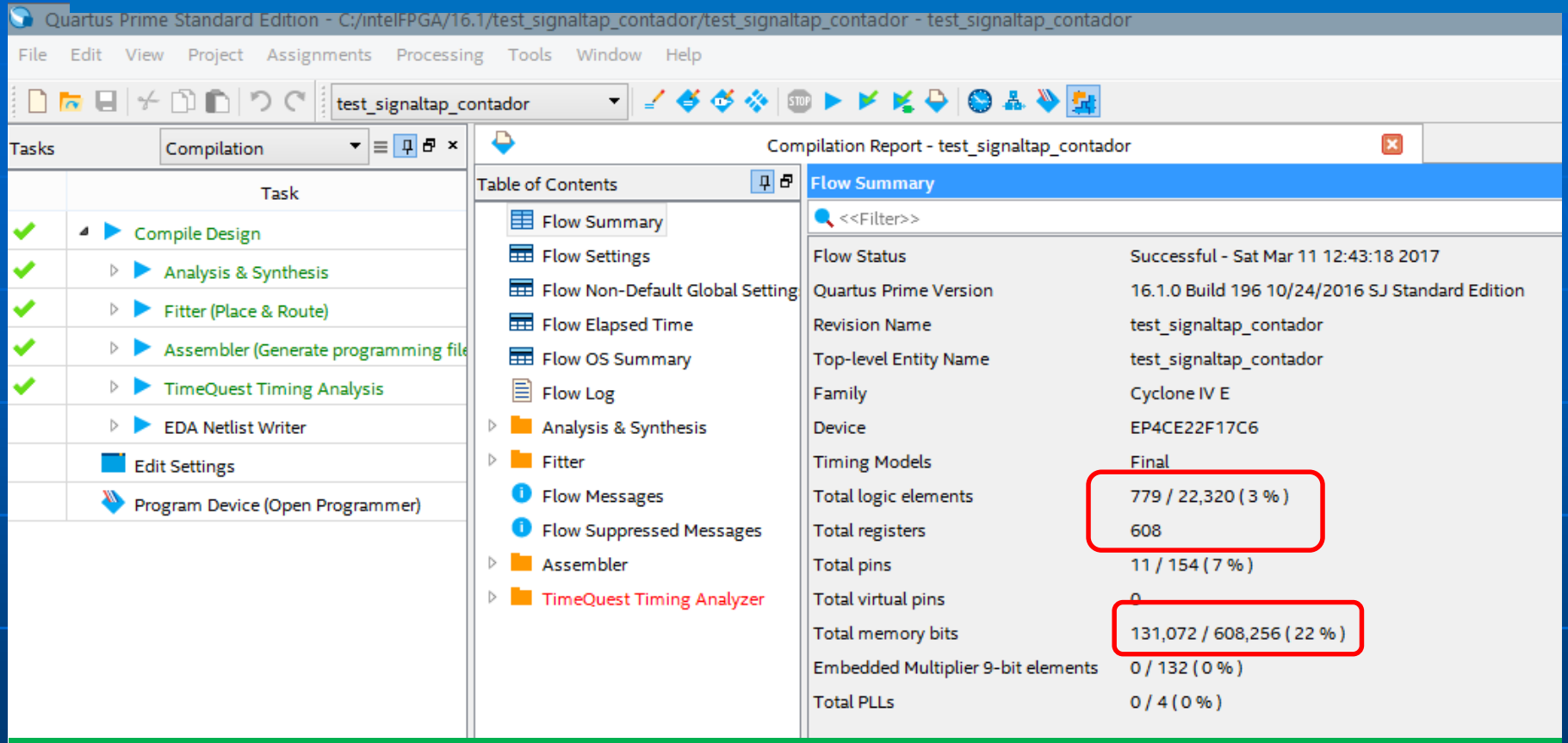
Requerimiento del analizador



Se configura el disparo como pretrigger.
La señal de disparo es RESET y el muestreo empieza al detectarse un flanco de subida en esa señal (al soltar el pulsador KEY0).

SIGNAL TAP II

RESULTADO DE LA COMPILACIÓN AL USAR LA HERRAMIENTA «SignalTap II»



Si bien el circuito contador no emplea memoria dedicada, las 16 Kmuestras solicitadas para hacer el análisis, utiliza el 22% de los recursos disponibles con el modelo de la Cyclone IV que viene en la DE0-Nano.

Por otro lado, el proyecto requiere un total de 11 Elementos Lógicos para implementar el contador, mientras que el SignalTap, necesita 656.

SIGNAL TAP II

PASOS A REALIZAR:

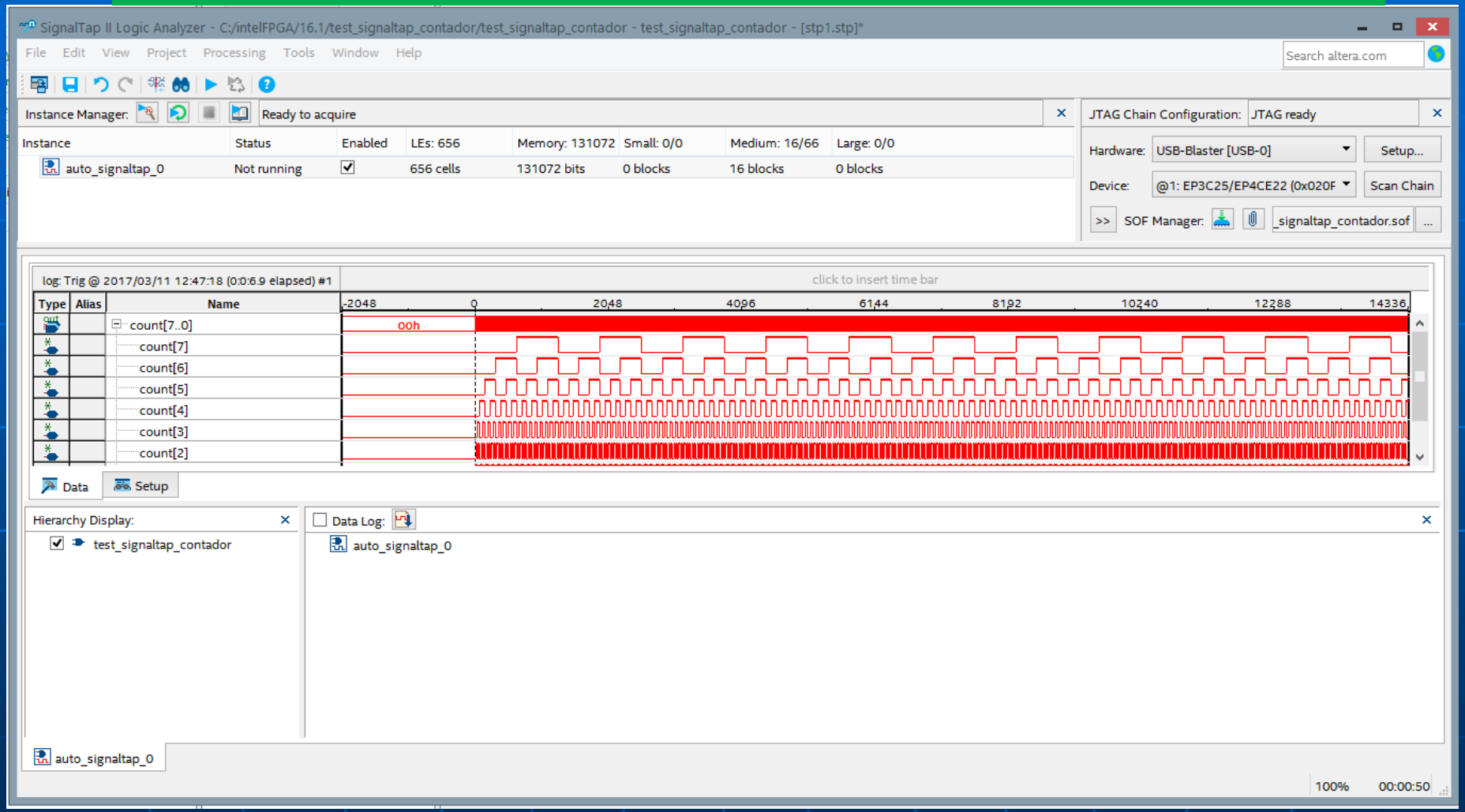
1)
El análisis se inicia al presionar el botón de Analysis de la ventana del SignalTap.

2)
Como la entrada de RESET es la que origina el comienzo de la adquisición de muestras, se debe presionar KEY0.
Al soltar KEY0 (flanco de subida) comienza el proceso el cual se detendrá al llenar el buffer.

3)
Al termino de la captura, aparecerá una ventana con un diagrama temporal con las señales configuradas para su análisis.
Se puede operar sobre ese diagrama o posteriormente dado que la información se almacena en un archivo de datos.

SIGNAL TAP II

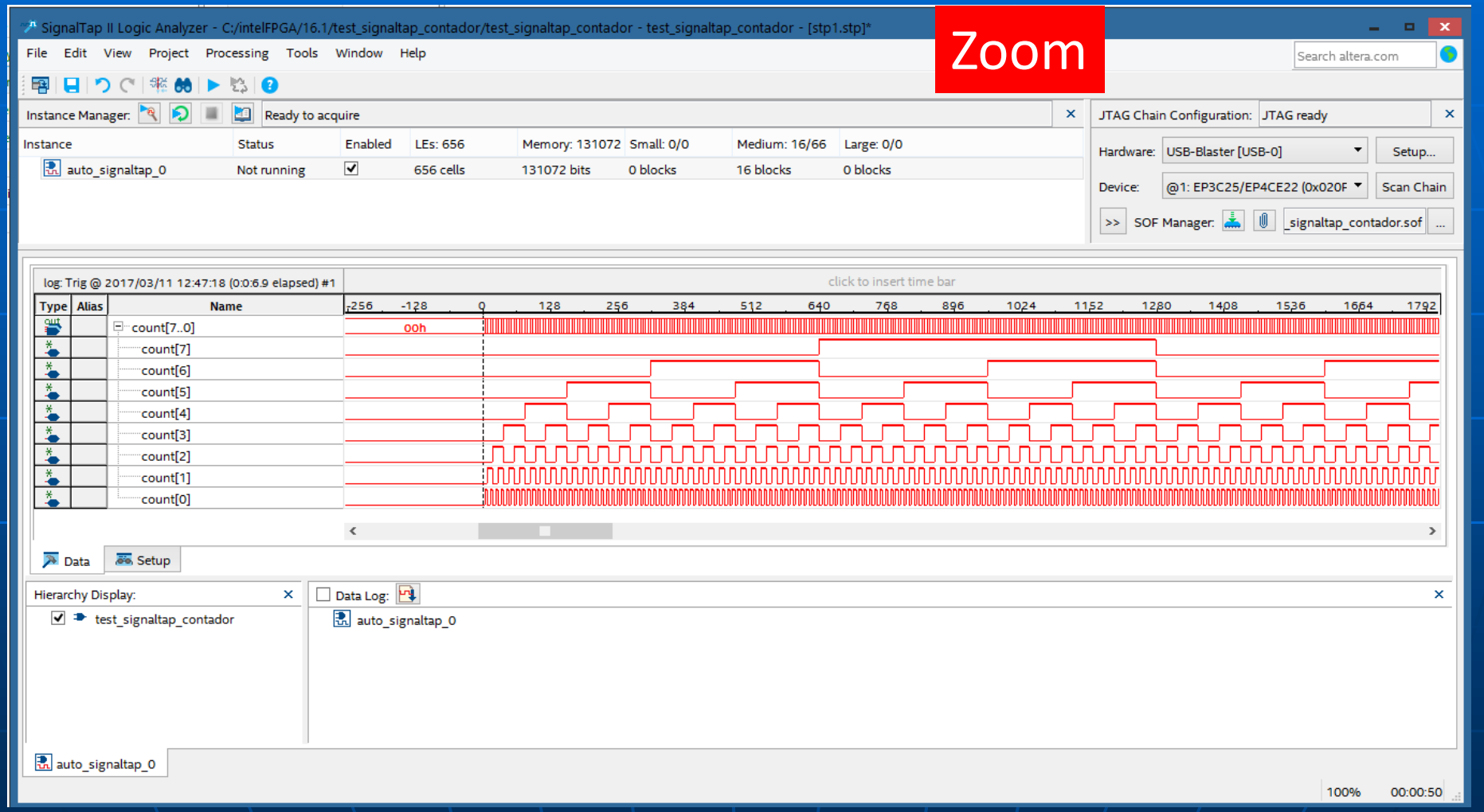
Diagrama temporal con las 16Kmuestras adquiridas de las 8 salidas del contador



SIGNAL TAP II

Diagrama temporal con las 16Kmuestras adquiridas de las 8 salidas del contador

Zoom



Síntesis

Síntesis:
Es el procedimiento por el cual mediante alguna técnica de fabricación se trata de generar hardware a través de la descripción de un sistema ó circuito.
Las posibles vías de desarrollo son al menos dos:
-Empleo de lógica standard.
-Empleo de lógica programable.

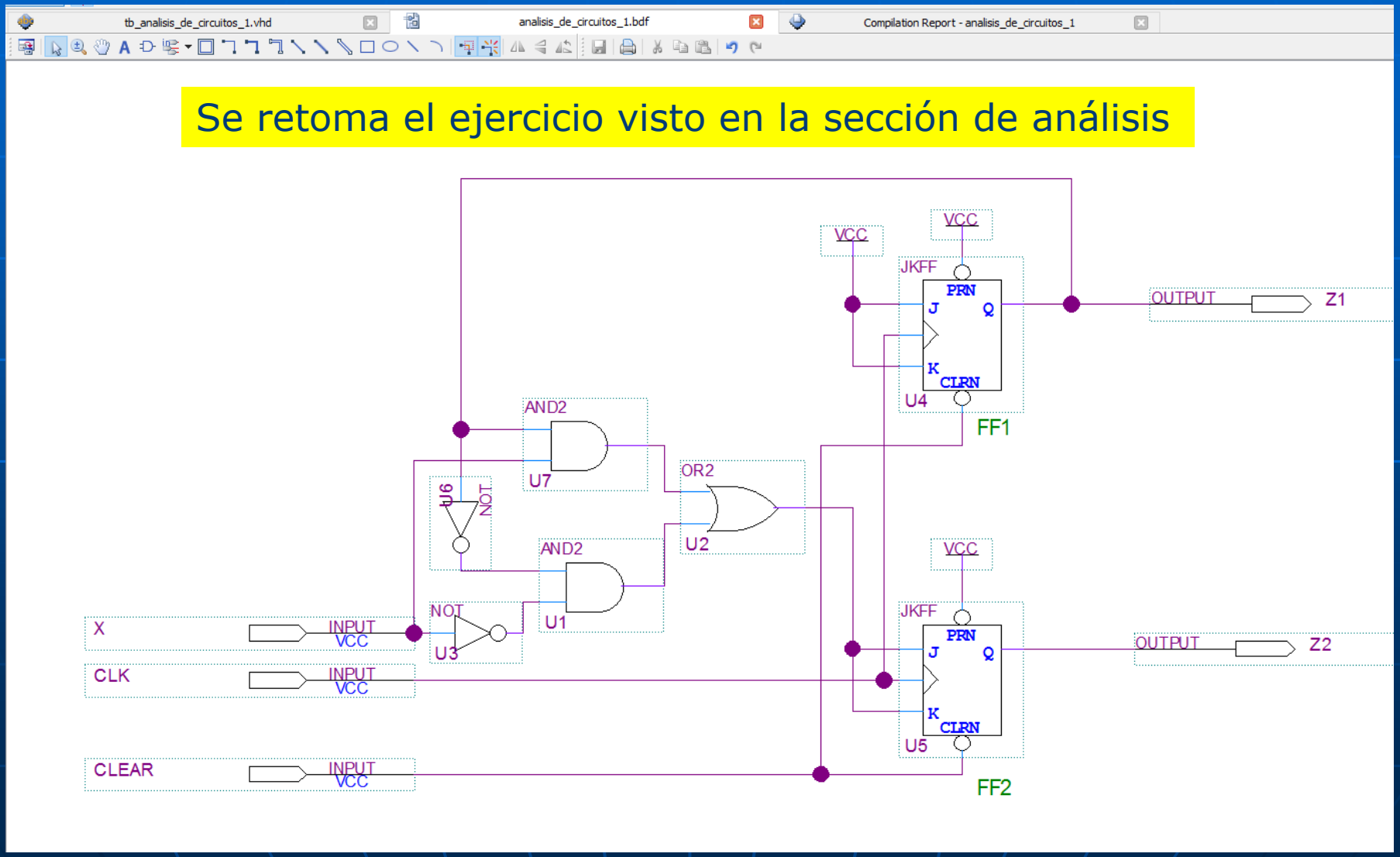
Métodos de desprición:

- **Tabla de verdad.**
- **Ecuaciones lógicas.**
- **Diagramas de estado.**
- **Algoritmos de síntesis.**
- **Diseño basado en HDL.**
- **etc.....**

Descripción + Simulación + Producción de hardware

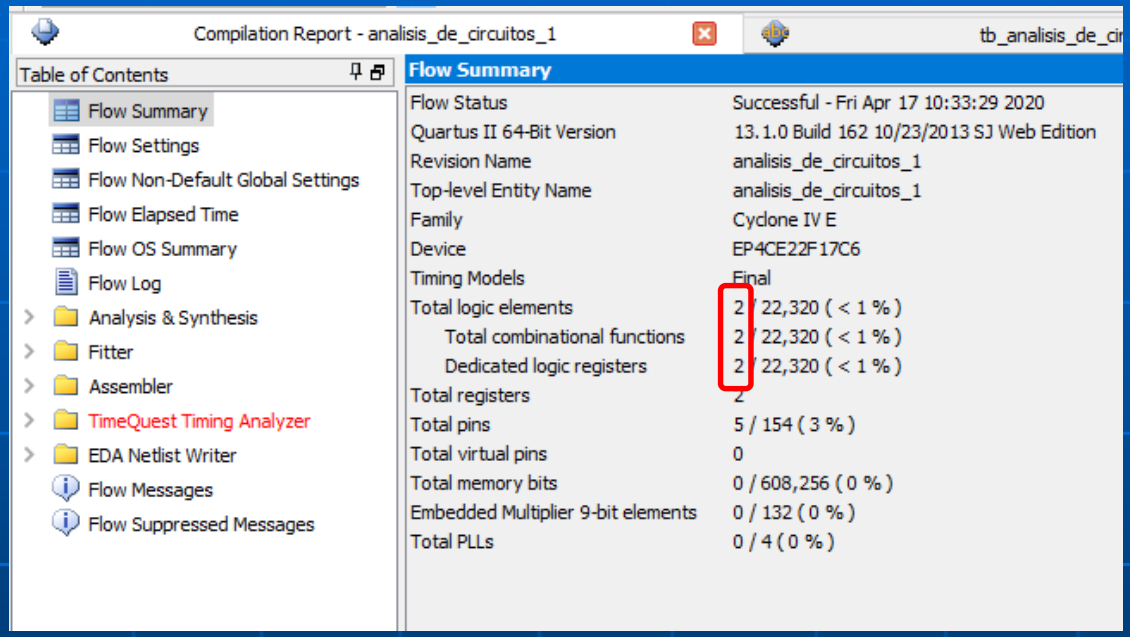
Síntesis

Diseño de un circuito en base a entrada por esquemático (ejemplo un contador sincrónico de 2 bits con entrada de RESET asincrónico y control de conteo progresivo-regresivo)



Síntesis

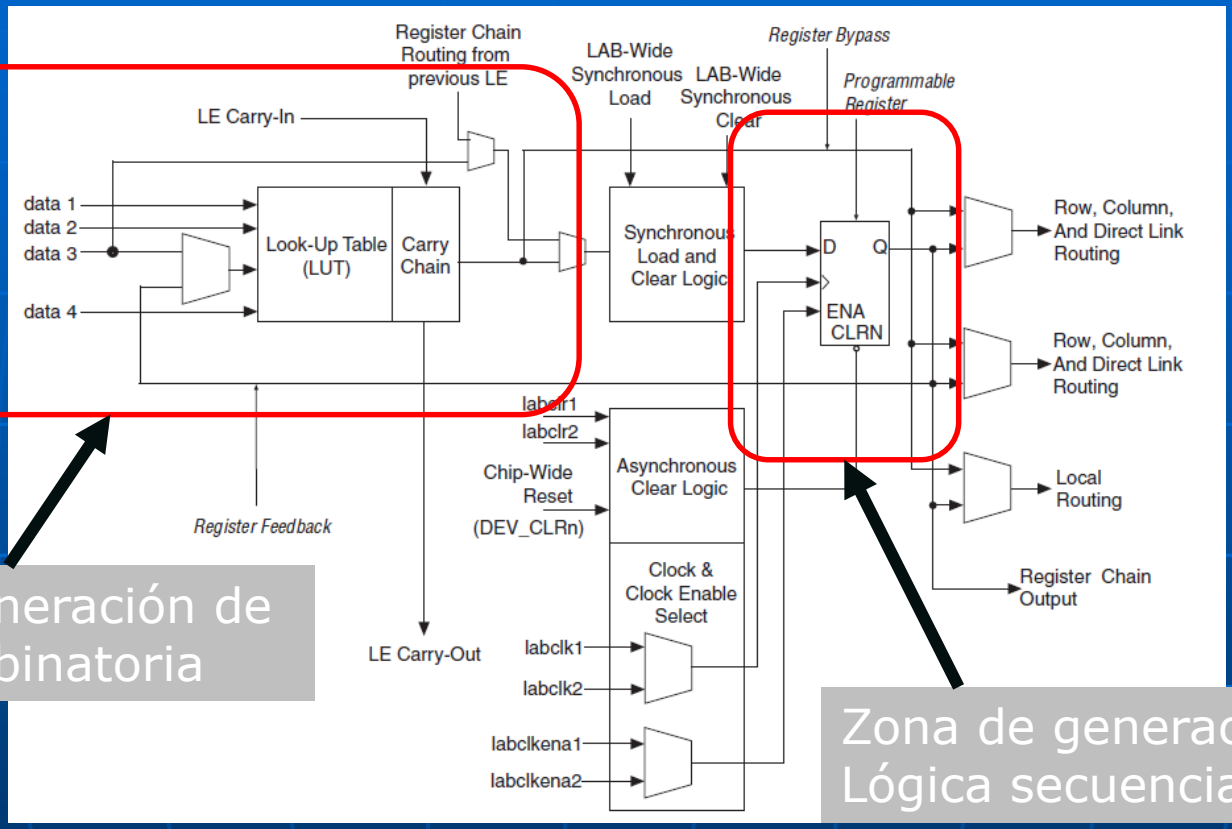
Resultado de compilar en Quartus II la entrada del circuito como esquemático previa conversión a descripción como VHDL.



Habiendo elegido la FPGA Cyclone IV modelo EP4CE22F17C6, se utilizaron sólo dos Elementos Lógicos para construir el circuito

Síntesis

Esquemático de un Elemento Lógico (LE) de la FPGA Cyclone IV "E"



Zona de generación de Lógica combinatoria

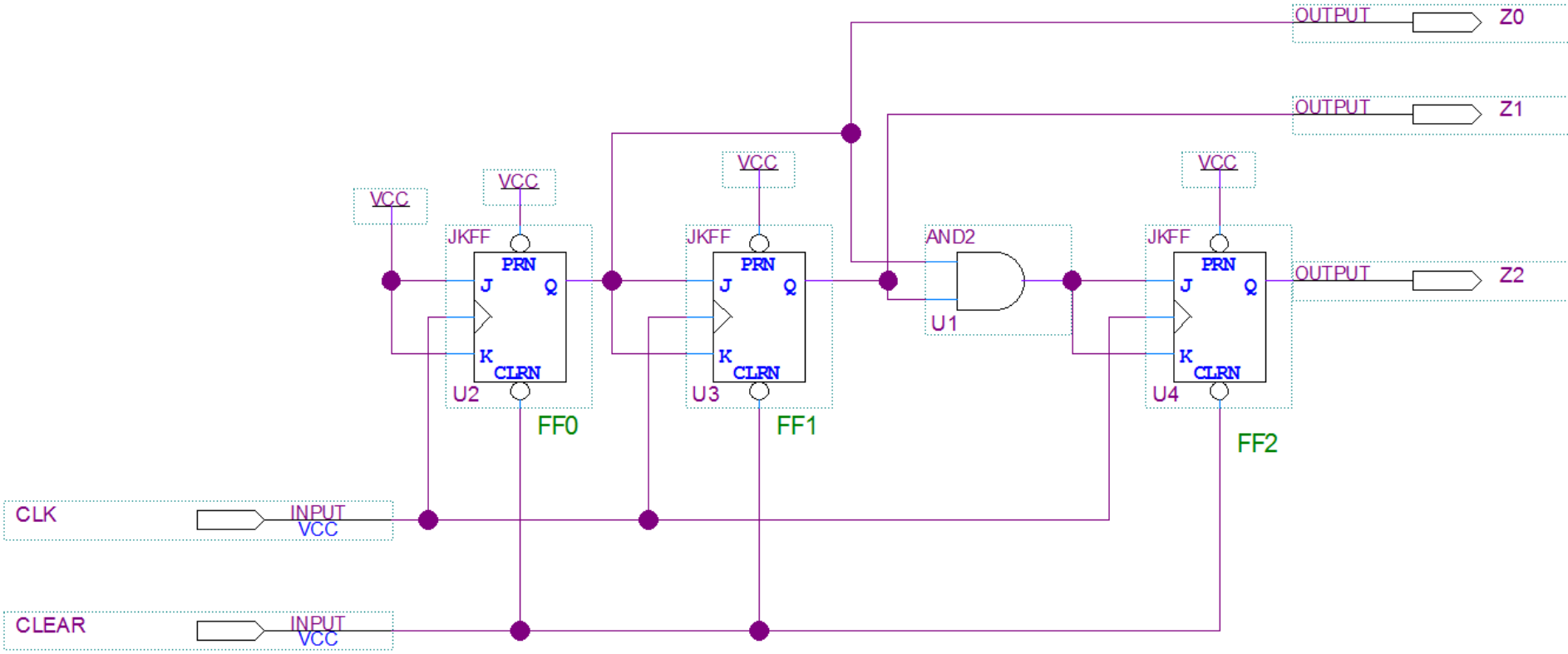
Zona de generación de Lógica secuencial

El dispositivo FPGA tipo Cyclone IV modelo EP4CE22Fxxxx dispone de 22.320 LEs. Para este diseño sólo requiere de 2 LEs

Síntesis

Entrada de diseño por esquemático

Ejemplo 2



Síntesis

Conversión a VHDL desde archivo BDF

analisis_de_circuitos_2.vhd | Compilation Report - analisis_de_circuitos_2

```
1  -- Copyright (C) 1991-2013 Altera Corporation
2  -- Your use of Altera Corporation's design tools, logic functions
3  -- and other software and tools, and its AMPP partner logic
4  -- functions, and any output files from any of the foregoing
5  -- (including device programming or simulation files), and any
6  -- associated documentation or information are expressly subject
7  -- to the terms and conditions of the Altera Program License
8  -- Subscription Agreement, Altera MegaCore Function License
9  -- Agreement, or other applicable license agreement, including,
10 -- without limitation, that your use is for the sole purpose of
11 -- programming logic devices manufactured by Altera and sold by
12 -- Altera or its authorized distributors.  Please refer to the
13 -- applicable agreement for further details.
14
15 -- PROGRAM     "Quartus II 64-Bit"
16 -- VERSION     "Version 13.1.0 Build 162 10/23/2013 SJ Web Edition"
17 -- CREATED     "Fri Apr 17 20:08:10 2020"
18
19 LIBRARY ieee;
20 USE ieee.std_logic_1164.all;
21
22 LIBRARY work;
23
24 ENTITY analisis_de_circuitos_2 IS
25    PORT
26    (
27       CLK :  IN  STD_LOGIC;
28       CLEAR :  IN  STD_LOGIC;
29       Z1 :  OUT  STD_LOGIC;
30       Z2 :  OUT  STD_LOGIC;
31       Z0 :  OUT  STD_LOGIC
32    );
33 END analisis_de_circuitos_2;
34
35 ARCHITECTURE bdf_type OF analisis_de_circuitos_2 IS
36
37 SIGNAL   SYNTHESIZED_WIRE_7 :  STD_LOGIC;
38 SIGNAL   JKFF_U3 :  STD_LOGIC;
39 SIGNAL   SYNTHESIZED_WIRE_0 :  STD_LOGIC;
40 SIGNAL   SYNTHESIZED_WIRE_8 :  STD_LOGIC;
41 SIGNAL   SYNTHESIZED_WIRE_3 :  STD_LOGIC;
42 SIGNAL   SYNTHESIZED_WIRE_4 :  STD_LOGIC;
43 SIGNAL   SYNTHESIZED_WIRE_9 :  STD_LOGIC;
44
45
46 BEGIN
47 Z1 <= JKFF_U3;
48 Z0 <= SYNTHESIZED_WIRE_7;
49 SYNTHESIZED_WIRE_0 <= '1';
50 SYNTHESIZED_WIRE_8 <= '1';
51 SYNTHESIZED_WIRE_3 <= '1';
52 SYNTHESIZED_WIRE_4 <= '1';
53
```

Síntesis

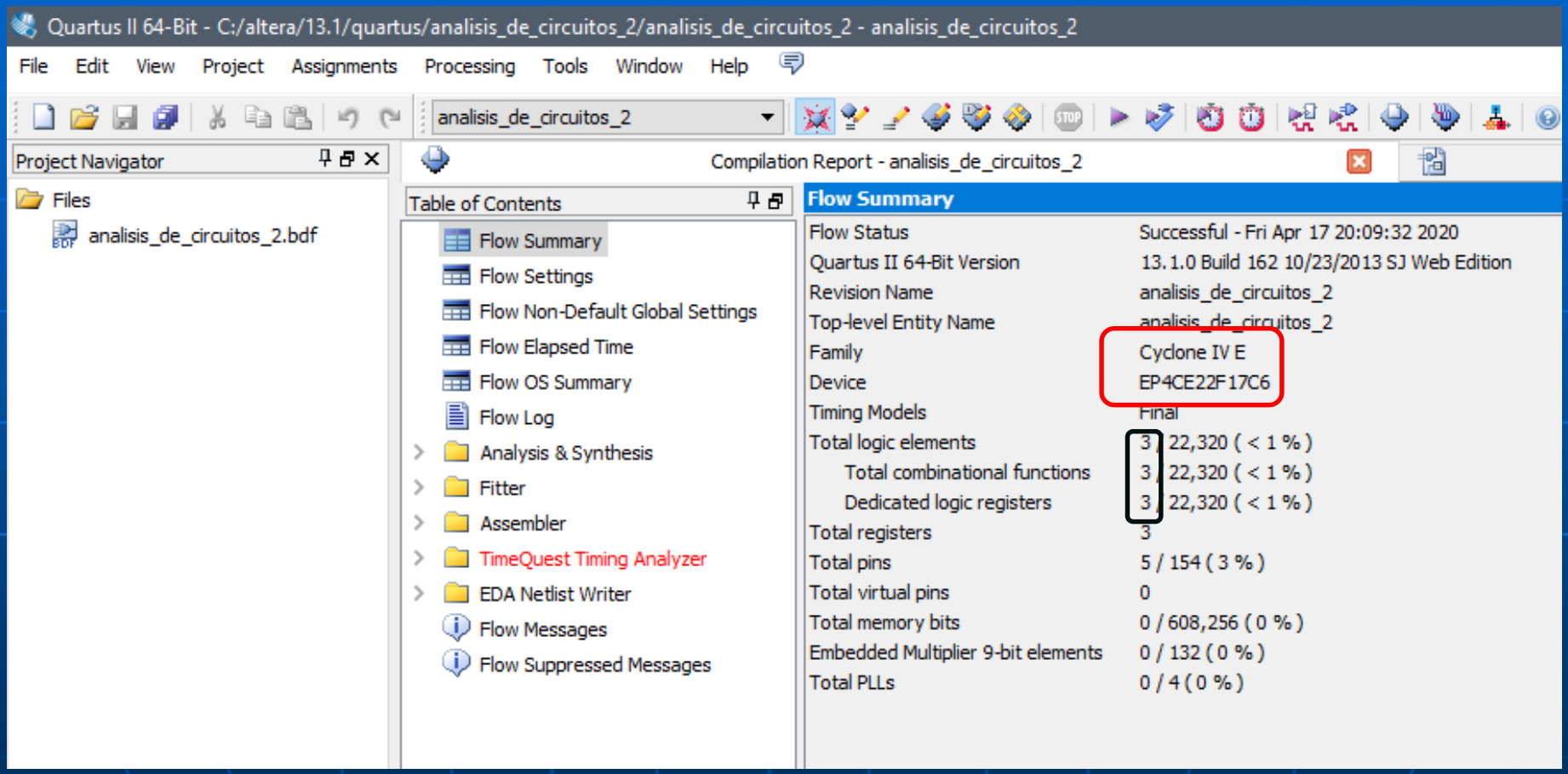
Conversión a VHDL desde archivo BDF

analisis_de_circuitos_2.vhd | Compilation Report - analisis_de_circuitos_2 | analisis_de_circuitos_2.bdf

```
49    SYNTHESIZED_WIRE_0 <= '1';
50    SYNTHESIZED_WIRE_8 <= '1';
51    SYNTHESIZED_WIRE_3 <= '1';
52    SYNTHESIZED_WIRE_4 <= '1';
53
54
55
56
57
58
59
60    SYNTHESIZED_WIRE_9 <= SYNTHESIZED_WIRE_7 AND JKFF_U3;
61
62
63    PROCESS(CLK,SYNTHESIZED_WIRE_0)
64    VARIABLE synthesized_var_for_SYNTHESIZED_WIRE_7 : STD_LOGIC;
65    BEGIN
66    IF (SYNTHESIZED_WIRE_0 = '0') THEN
67       synthesized_var_for_SYNTHESIZED_WIRE_7 := '1';
68    ELSIF (RISING_EDGE(CLK)) THEN
69       synthesized_var_for_SYNTHESIZED_WIRE_7 := (NOT(synthesized_var_for_SYNTHESIZED_WIRE_7) AND SYNTHESIZED_WIRE_8) OR (synthesized_var_for_SYNTHESIZED_WIRE_7 AND (NOT(SYNTHESIZED_WIRE_8)));
70    END IF;
71       SYNTHESIZED_WIRE_7 <= synthesized_var_for_SYNTHESIZED_WIRE_7;
72    END PROCESS;
73
74
75    PROCESS(CLK,CLEAR,SYNTHESIZED_WIRE_3)
76    VARIABLE synthesized_var_for_JKFF_U3 : STD_LOGIC;
77    BEGIN
78    IF (CLEAR = '0') THEN
79       synthesized_var_for_JKFF_U3 := '0';
80    ELSIF (SYNTHESIZED_WIRE_3 = '0') THEN
81       synthesized_var_for_JKFF_U3 := '1';
82    ELSIF (RISING_EDGE(CLK)) THEN
83       synthesized_var_for_JKFF_U3 := (NOT(synthesized_var_for_JKFF_U3) AND SYNTHESIZED_WIRE_7) OR (synthesized_var_for_JKFF_U3 AND (NOT(SYNTHESIZED_WIRE_7)));
84    END IF;
85       JKFF_U3 <= synthesized_var_for_JKFF_U3;
86    END PROCESS;
87
88
89    PROCESS(CLK,CLEAR,SYNTHESIZED_WIRE_4)
90    VARIABLE synthesized_var_for_Z2 : STD_LOGIC;
91    BEGIN
92    IF (CLEAR = '0') THEN
93       synthesized_var_for_Z2 := '0';
94    ELSIF (SYNTHESIZED_WIRE_4 = '0') THEN
95       synthesized_var_for_Z2 := '1';
96    ELSIF (RISING_EDGE(CLK)) THEN
97       synthesized_var_for_Z2 := (NOT(synthesized_var_for_Z2) AND SYNTHESIZED_WIRE_9) OR (synthesized_var_for_Z2 AND (NOT(SYNTHESIZED_WIRE_9)));
98    END IF;
99       Z2 <= synthesized_var_for_Z2;
100   END PROCESS;
101
102
103   END bdf_type;
```

Síntesis

Resultado de la compilación



Se emplean sólo 3 LEs para la síntesis del contador de 3 bits

Síntesis

Test Bench para la verificación del contador

tb_analisis_de_circuitos_2.vhd | analisis_de_circuitos_2.vhd

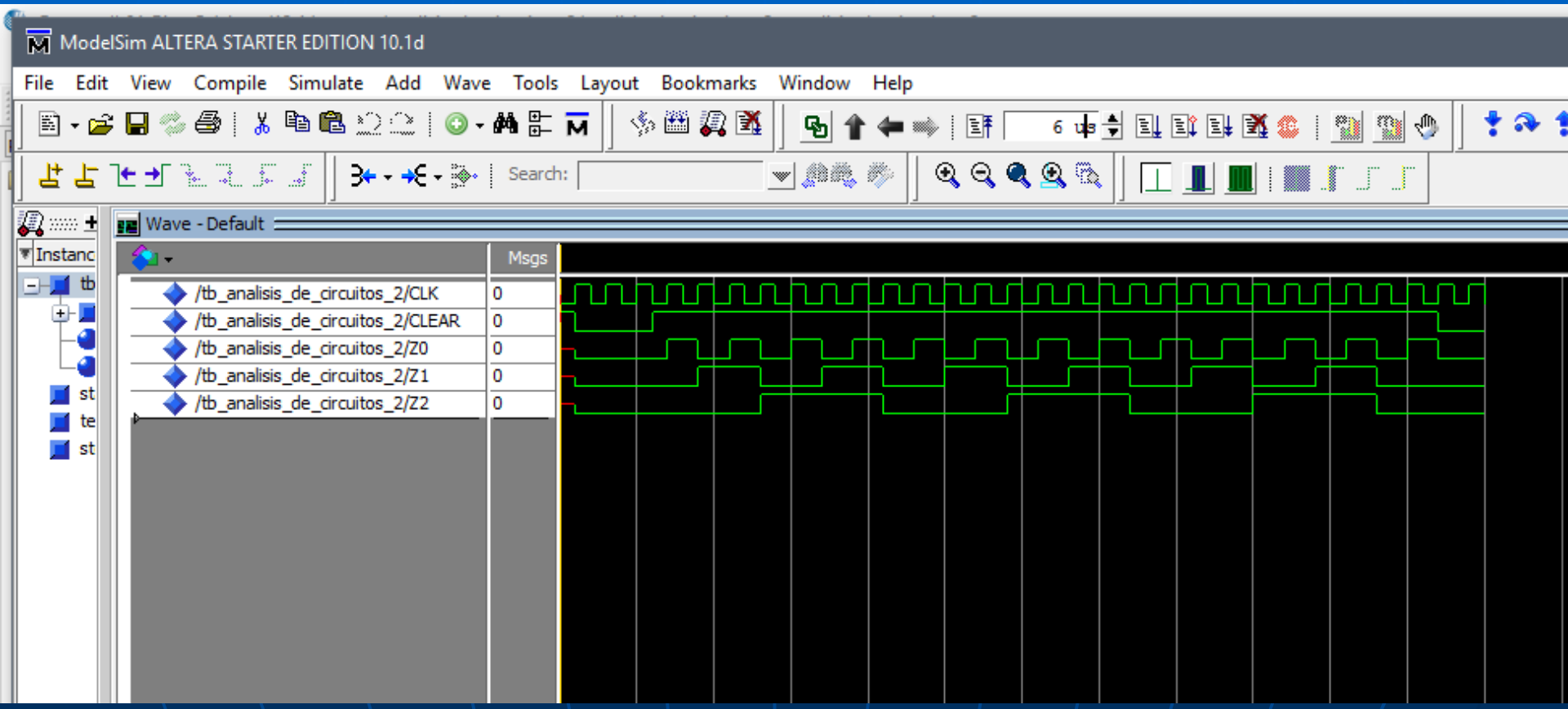
```
1  --Test-bench del proyecto analisis_de_circuitos_2 Sergio Noriega 2020
2  --Archivo: tb_analisis_de_circuitos_2.vhd.
3
4  library ieee;
5  use ieee.std_logic_1164.all;
6
7  entity tb_analisis_de_circuitos_2 is
8  end tb_analisis_de_circuitos_2;
9
10 architecture test of tb_analisis_de_circuitos_2 is
11    component analisis_de_circuitos_2
12    PORT
13    (
14       CLK :  IN  STD_LOGIC;
15       CLEAR :  IN  STD_LOGIC;
16       Z1 :  OUT  STD_LOGIC;
17       Z2 :  OUT  STD_LOGIC;
18       Z0 :  OUT  STD_LOGIC
19    );
20    end component;
21
22    signal CLK, CLEAR : STD_LOGIC;
23    signal Z0, Z1, Z2 : std_LOGIC;
24
25
26 begin
27
28   uut: analisis_de_circuitos_2 port map( CLK => CLK, CLEAR => CLEAR,
29                                          Z1 => Z1, Z2 => Z2, Z0 => Z0);
30
31
32    gen_reloj : process -- Reloj de 200 ns de periodo y 50% de ciclo de trabajo
33       begin
34
35          CLK <= '0';
36          wait for 100 ns;
37          CLK <= '1';
38          wait for 100 ns;
39    end process gen_reloj;
40
41    estimulos : process
42    begin
43       CLEAR <= '1';
44       wait for 100 ns;
45       CLEAR <= '0';
46       wait for 500 ns;
47       CLEAR <= '1';
48       wait for 5 us;
49    end process estimulos;
50 end test;
```

Generación de RELOJ.

Generación de RESET.

Síntesis

Resultado de la simulación con Modelsim del contador



Síntesis

Máquinas de estado

Se define estado a una dada combinación entre las salidas de los elementos de memoria (FFs) que conforman el circuito.

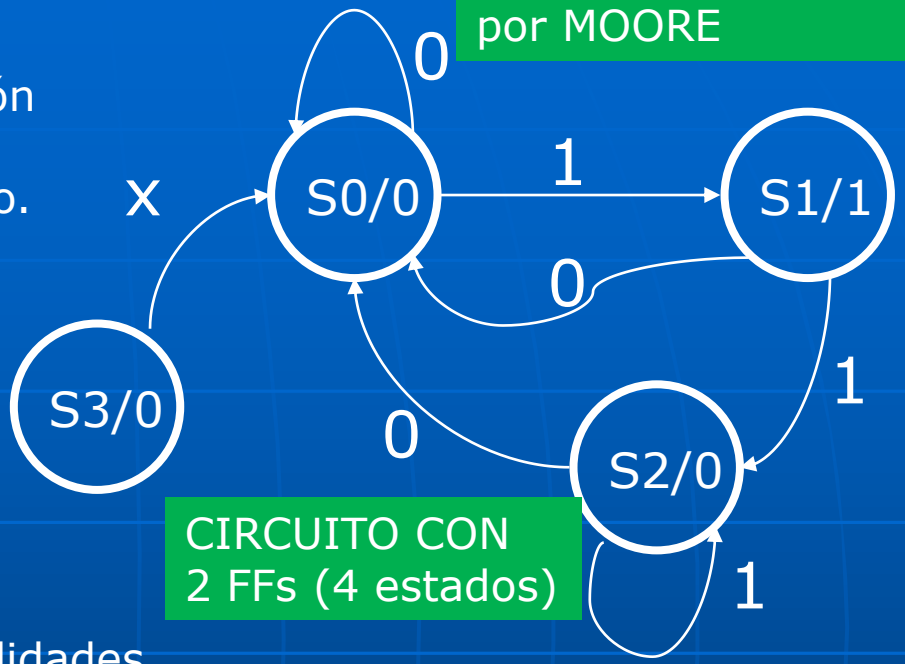
En Moore una salida NO puede cambiar su valor mientras esté en un dado estado.

En Mealy, en cambio, puede hacerlo.

En estos dos ejemplos hay una entrada entonces de cada estado hay dos posibilidades de acción.
Si hay dos entradas serán 4 y así sucesivamente.

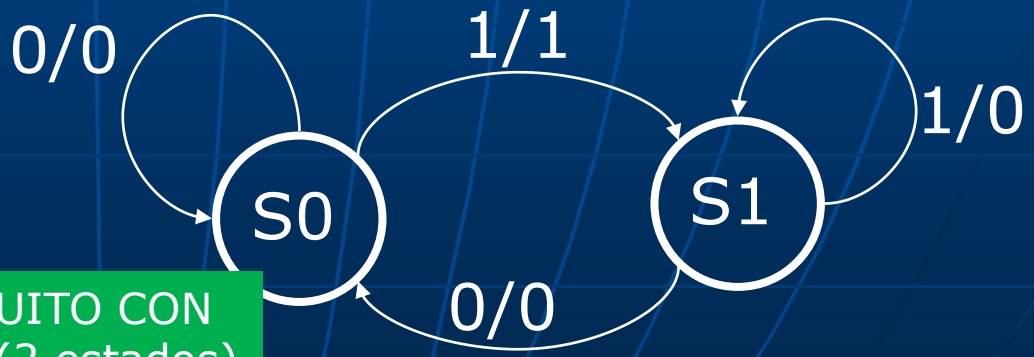
La referencia temporal en estos diagramas está implícita:
Para pasar de un estado a otro DEBE llegar un flanco activo de RELOJ.

Modelo de descripción por MOORE



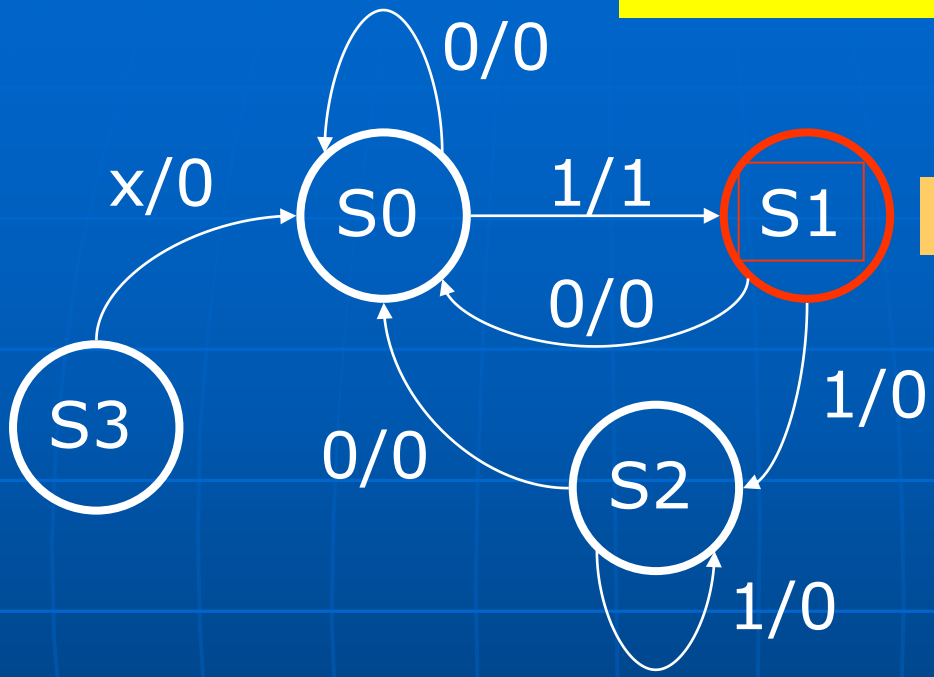
CIRCUITO CON
2 FFs (4 estados)

Modelo de descripción por MEALY



CIRCUITO CON
1 FF (2 estados)

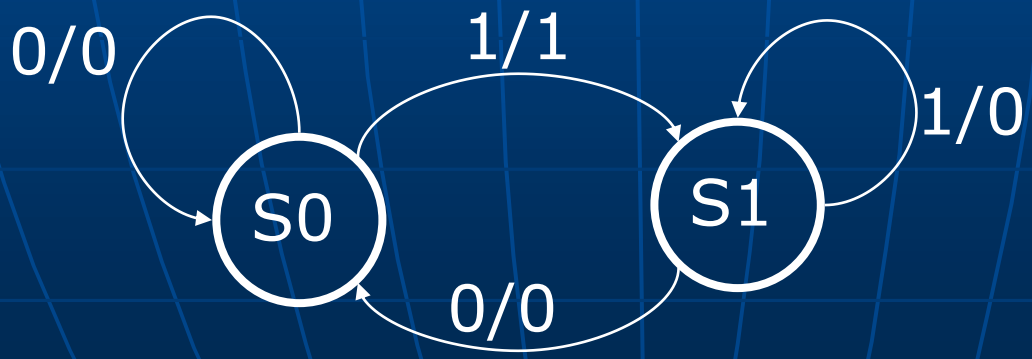
REDUNDANCIA



ESTADO REDUNDANTE

SE NECESITAN 2 FF

REORDENANDO .....



AHORA ....
SE NECESITA 1 FF

Síntesis

Máquinas de estado

Síntesis por Mealy

Contador binario progresivo-regresivo de 2 bits

Método para lógica standard

X=0 conteo regresivo y viceversa.

DIAGRAMA DE ESTADOS

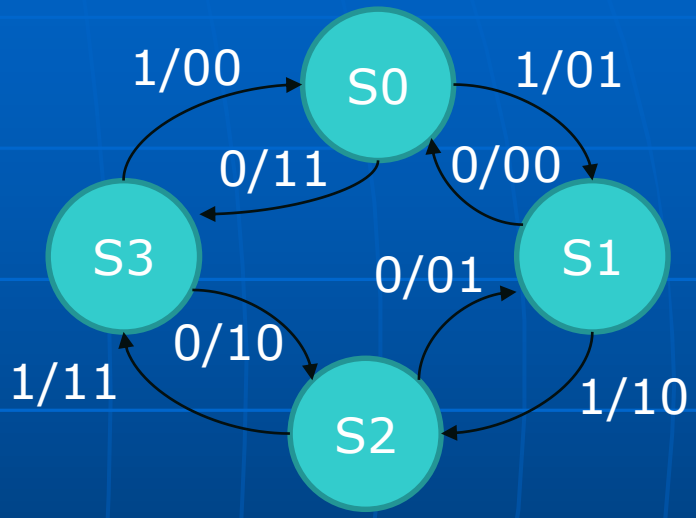


TABLA DE EXCITACIÓN

ESTADO ACTUAL		ESTADO SIGUIENTE		Q1Q0	Q1Q0	D1D0	D1D0
S0	00	S3	S1	11	01	11	01
S1	01	S0	S2	00	10	00	10
S2	10	S1	S3	01	11	01	11
S3	11	S2	S0	10	00	10	00
		0	1	0	1	0	1
	X						

Tablas de transición para generar el siguiente estado

TABLA DE TRANSICIÓN FLIP-FLOP TIPO "D"

Qn	Qn+1	D
0	0	0
0	1	1
1	0	0
1	1	1

TABLA DE TRANSICIÓN FLIP-FLOP TIPO "JK"

Qn	Qn+1	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Si quiero usar FFs tipo "D", simplemente debo copiar en "D" el valor futuro que espero de Q.

Si quiero usar FFs tipo "JK", tengo más opciones:
Por ejemplo si el Q actual es 0 (Qn=0) y quiero pasar a un Q de 0 (Qn+1=1).
entonces tengo dos opciones:
Pongo directamente JK=10 ó elijo la opción de negar el estado anterior de Q (JK=11), es decir queda JK=1X (don't care en la entrada "K").

Método para lógica standard

X \ Q1Q0	00	01	11	10
0	1 (0)	0 (1)	1 (3)	0 (2)
1	0 (4)	1 (5)	0 (7)	1 (6)

D1

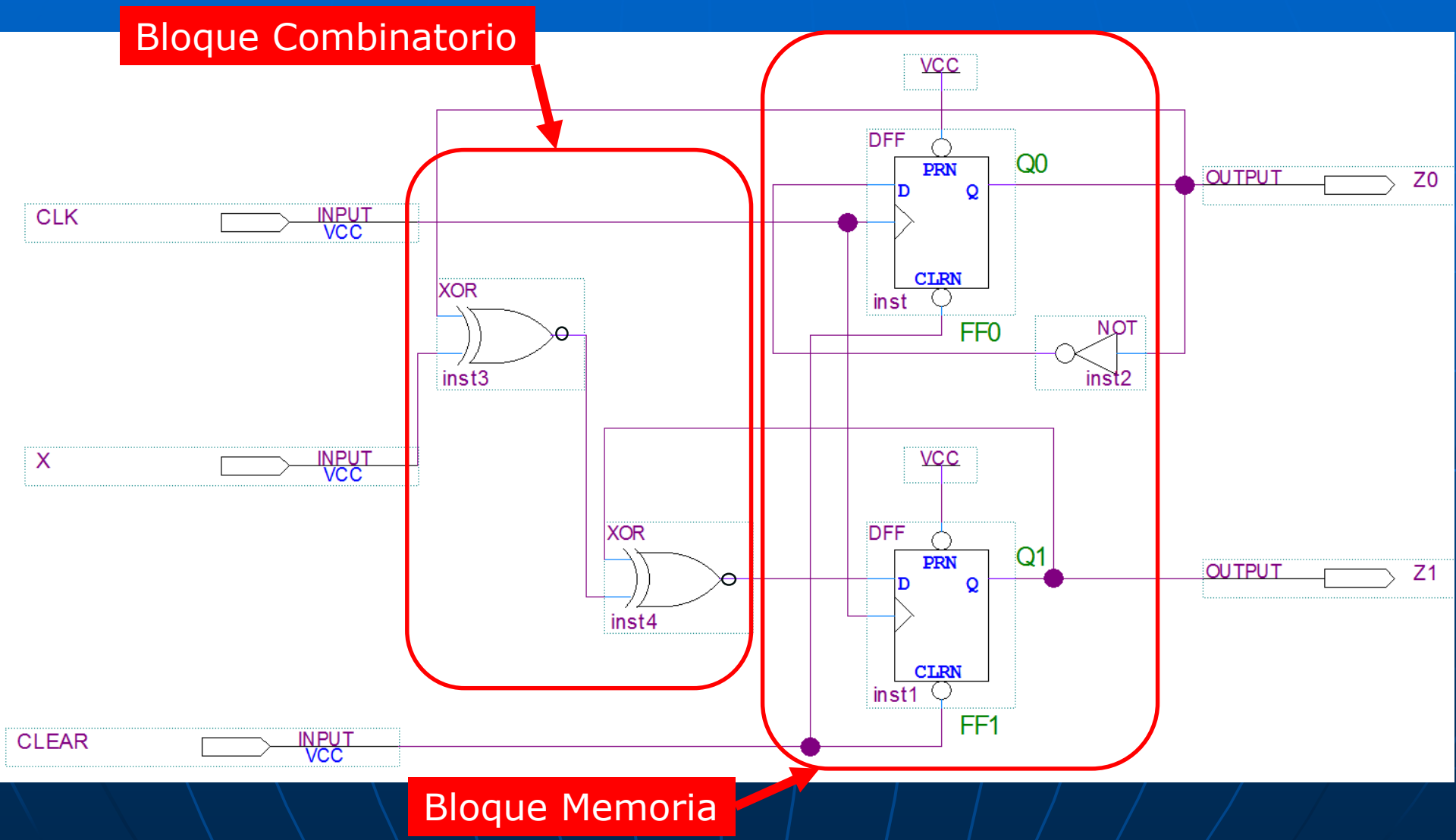
$$D1 = \overline{X \oplus Q1 \oplus Q0}$$

X \ Q1Q0	00	01	11	10
0	1 (0)	0 (1)	0 (3)	1 (2)
1	1 (4)	0 (5)	0 (7)	1 (6)

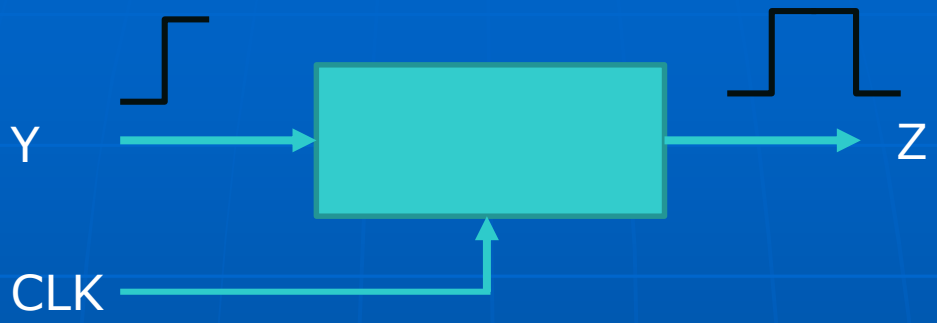
D0

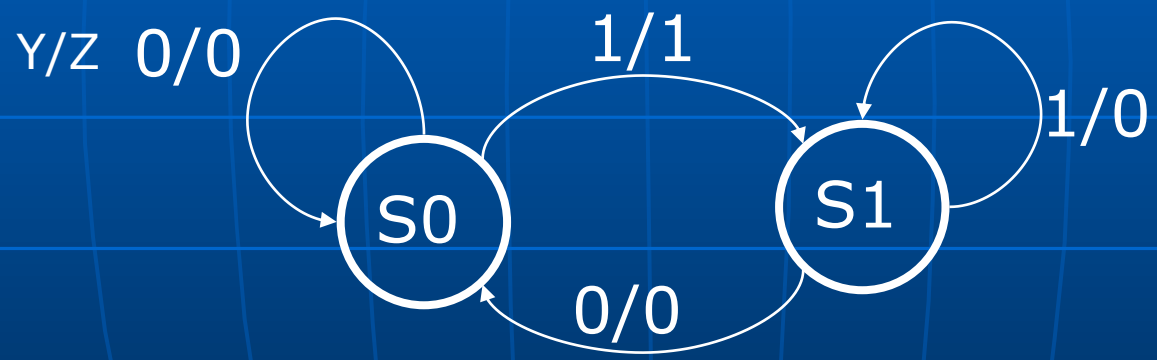
$$D0 = \overline{Q0}$$

Contador binario progresivo-regresivo de 2 bits



Diseño de circuito monoestable disparado por flanco ascendente





El circuito está a la espera de detectar un flanco de subida por la entrada Y. Cuando ello ocurra, la salida Z se pondrá en "1" durante un ciclo de reloj CLK.

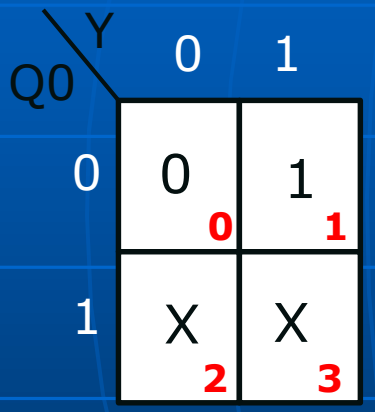
Síntesis

Síntesis por Mealy

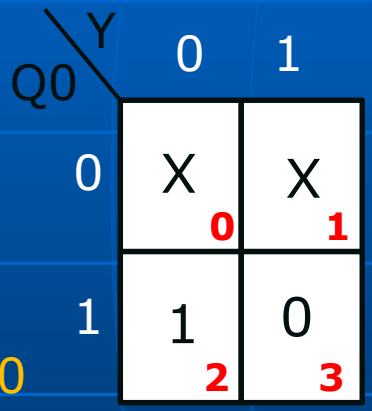
Método para lógica standard

TABLA DE EXCITACIÓN

Q0(n)	Q0(n+1)	J0	K0	Z	Q0(n+1)	J_0	K_0	Z
0	0	0	X	0	1	1	X	1
1	0	X	1	0	1	X	0	0
	Y=0				Y=1			

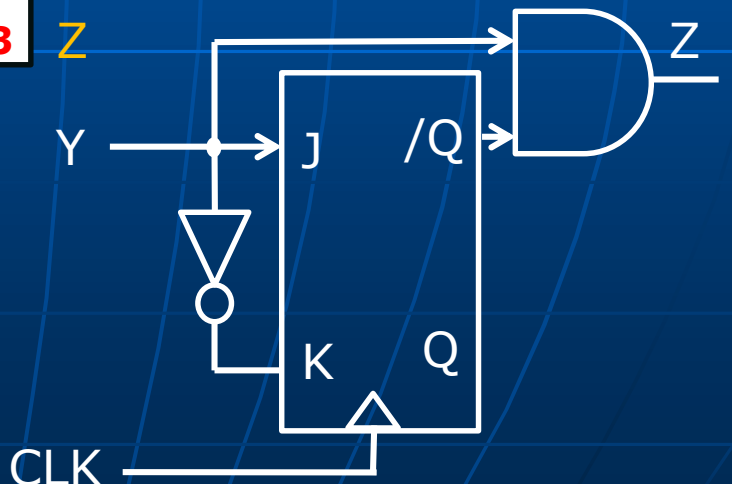


J0



K0

Z



Diseño de circuito monoestable disparado por flanco ascendente

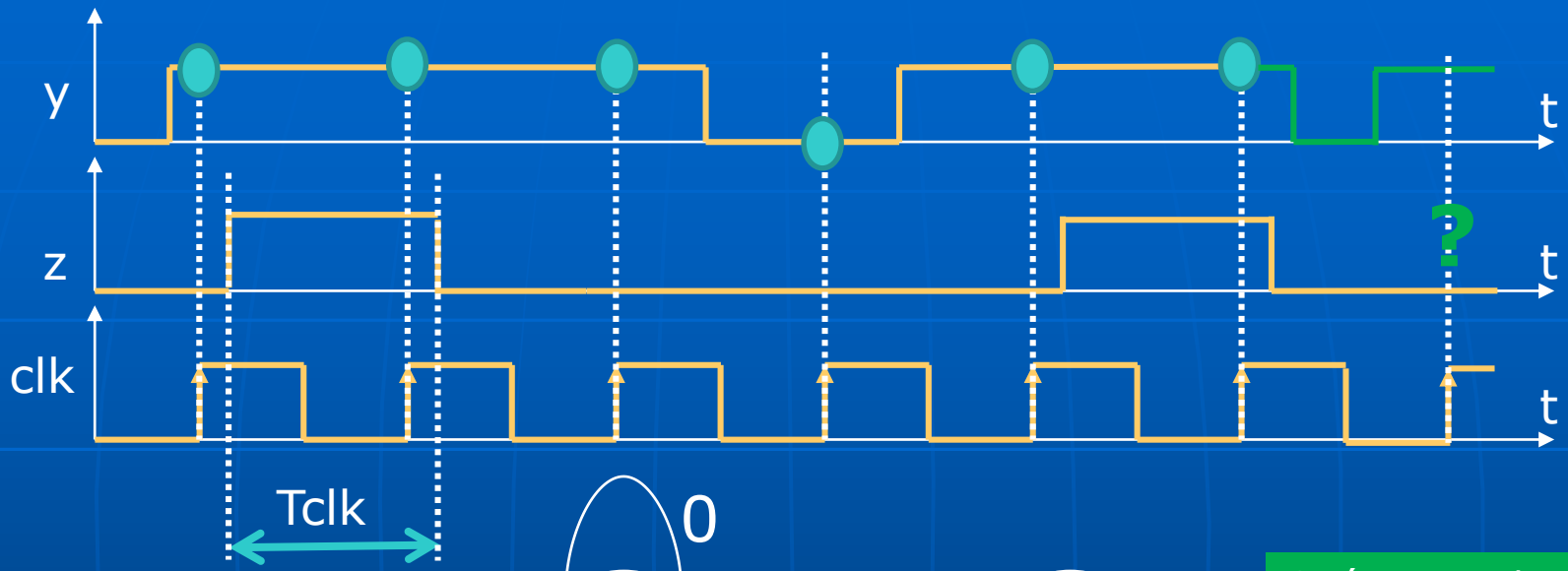


DIAGRAMA DE ESTADOS

Qué pasa si la entrada cambia dos veces entre dos flancos activos de reloj ???.
Cómo se evita eso ???.

Síntesis por Moore

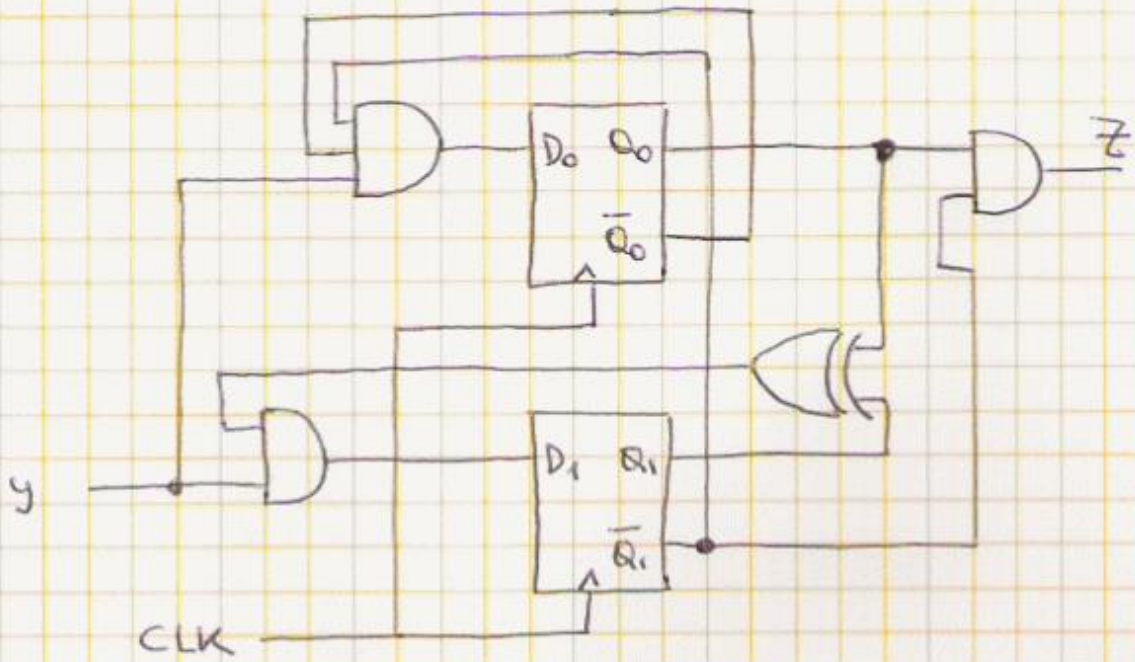
y \ Q_1Q_0	00	01	11	10
0	00	00	00	00
1	01	10	00	10

D_1D_0

$$D_1 = y \cdot (\bar{Q}_1 Q_0 + Q_1 \bar{Q}_0) = y \cdot (Q_0 \oplus Q_1)$$

$$D_0 = y \cdot \bar{Q}_1 \cdot \bar{Q}_0$$

$$z = \overline{Q_1 \cdot Q_0}$$



Por Moore el mismo problema generó un hardware un poco mas complejo pero con la salida sin depender de la entrada.

Monoestable disparado por flanco ascendente

monoestable_moore_1.vhd

```
1  library ieee; use ieee.std_logic_1164.all;
2  entity monoestable_moore_1 is
3     port
4     (
5        clk      : in  std_logic;
6        y        : in  std_logic;
7        reset    : in  std_logic;
8        z        : out std_logic
9     );
10 end entity;
11 architecture rtl of monoestable_moore_1 is
12
13    type state_type is (s0, s1, s2);
14    signal state : state_type;
15 begin
16    process (clk, reset, y)
17    begin
18       if reset = '1' then
19          state <= s0;
20       elsif (clk'event and clk = '1') then
21          case state is
22             when s0=>
23                if y = '1' then
24                   state <= s1;
25                else
26                   state <= s0;
27                end if;
28             when s1=>
29                if y = '1' then
30                   state <= s2;
31                else
32                   state <= s0;
33                end if;
34             when s2=>
35                if y = '1' then
36                   state <= s2;
37                else
38                   state <= s0;
39                end if;
40          end case;
41       end if;
42    end process;
43
44    process (state)
45    begin
46       case state is
47          when s0=>
48                z <= '0';
49          when s1=>
50                z <= '1';
51          when s2=>
52                z <= '0';
53       end case;
54    end process;
55 end rtl;
```

Sección que define a que estado iré, dependiendo del valor de la entrada.

Sección que define cuanto valdrá la salida, dependiendo del estado.

tb_monoestable_moore_1.vhd | monoestable_moore_1.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3
4  entity tb_monoestable_1 is
5  end tb_monoestable_1;
6
7  architecture test of tb_monoestable_1 is
8
9     component monoestable_moore_1
10       Port (   clk      : in std_logic;
11                y        : in std_logic;
12                reset    : in std_logic;
13                z        : out std_logic
14          );
15    end component;
16
17 signal clk           : std_logic;
18 signal reset         : std_logic;
19 signal y             : std_logic;
20 signal z             : std_logic;
21
22 begin
23
24   uut: monoestable_moore_1 port map (  clk => clk,
25                                        y => y,
26                                        reset => reset,
27                                        z => z
28                                        );
29
30    gen_reloj : process  -- Reloj de 200 ns de periodo y 50% de ciclo de trabajo
31       begin
32          clk <= '0';
33          wait for 100 ns;
34          clk <= '1';
35          wait for 100 ns;
36    end process gen_reloj;
37
38    estimulos : process
39    begin
40             y   <= '0';
41             reset <= '0';
42             wait for 250 ns;
43             reset <= '1';
44             wait for 250 ns;
45             reset  <= '0';
46             wait for  250 ns;
47             y <= '1';
48             wait for 551 ns;
49             y   <= '0';
50             wait for  253 ns;
51             y <= '1';
52             wait for  1 us;
53    end process estimulos;
54 end test;
```

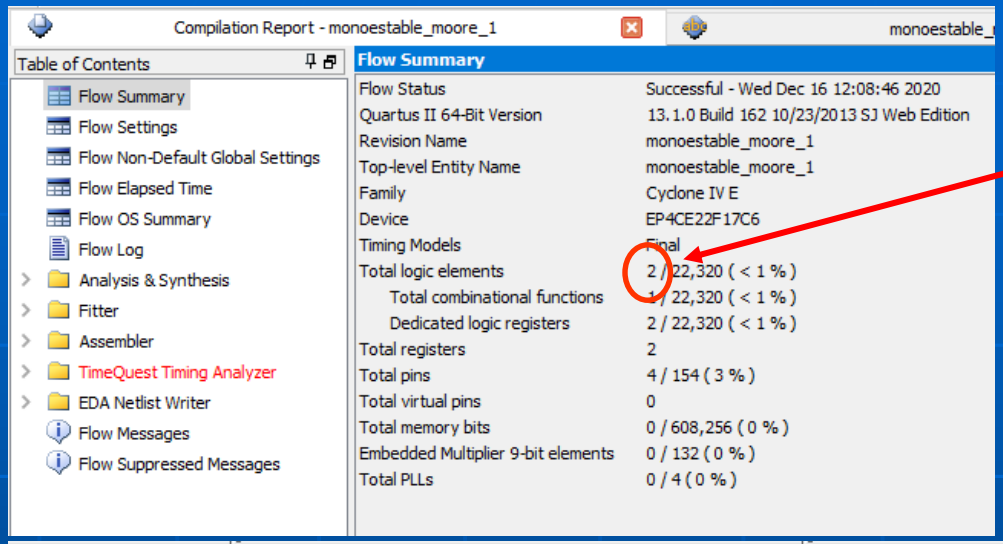
Monoestable disparado por flanco ascendente

Generación de los estímulos temporales de "clk", "reset" y la entrada "y" para emplear en el Modelsim y generar luego la gráfica temporal de la salida "z".

Generación de "clk".

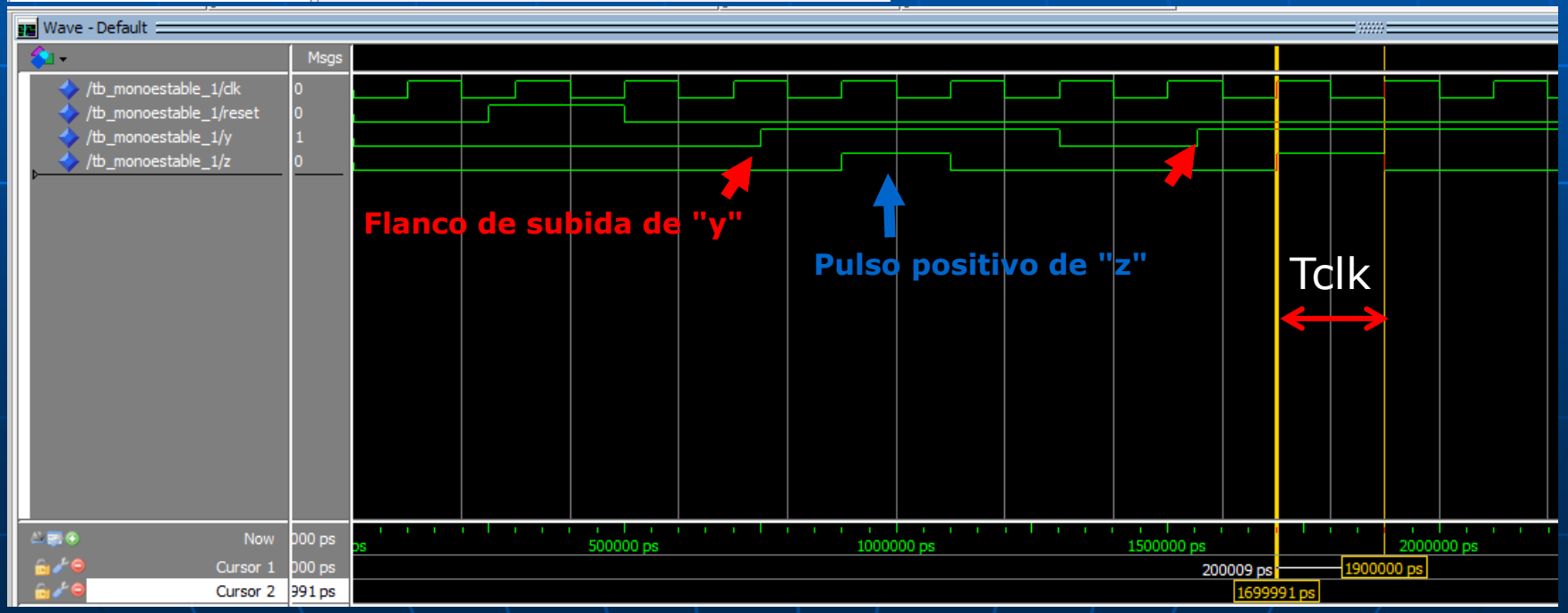
Generación de "reset" e "y".

Monoestable disparado por flanco ascendente

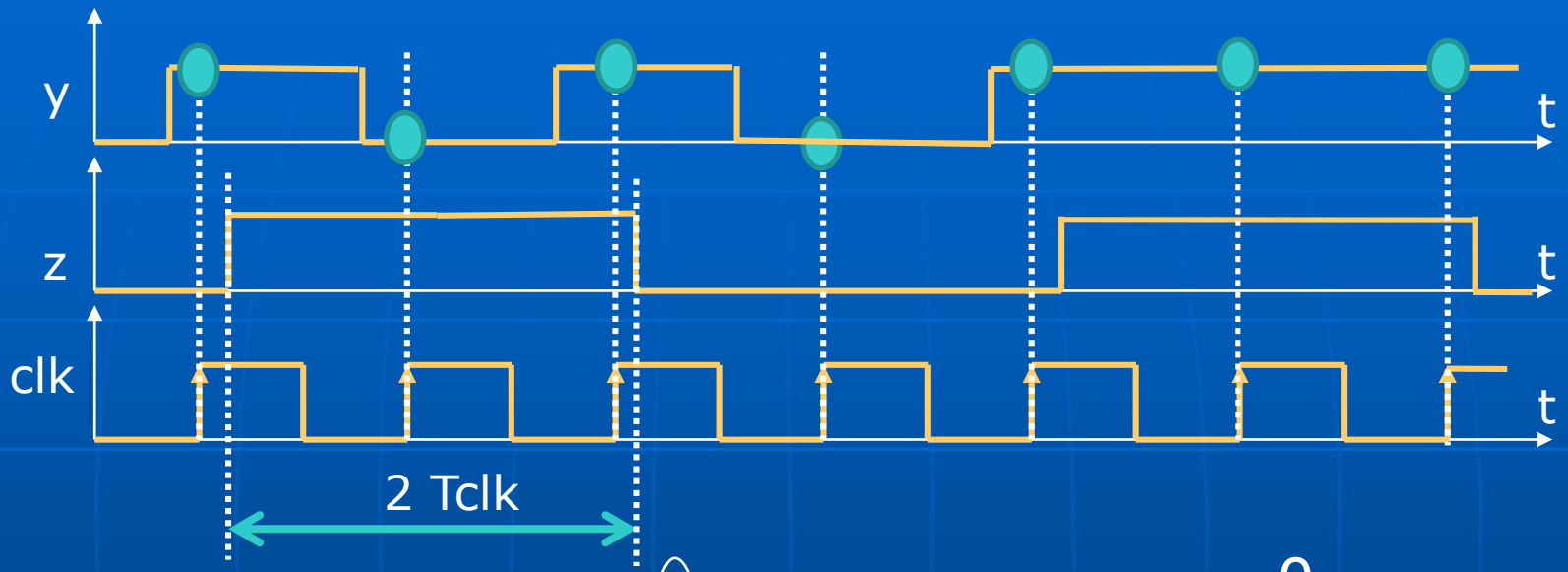


El proyecto insume sólo dos LE (Elementos Lógicos).

Dispositivo:
Cyclone IV E,
modelo EP4CE22F17C6.



Diseño de circuito monoestable disparado por flanco ascendente



Monoestable no-redisparable con duración de 2 ciclos de reloj.

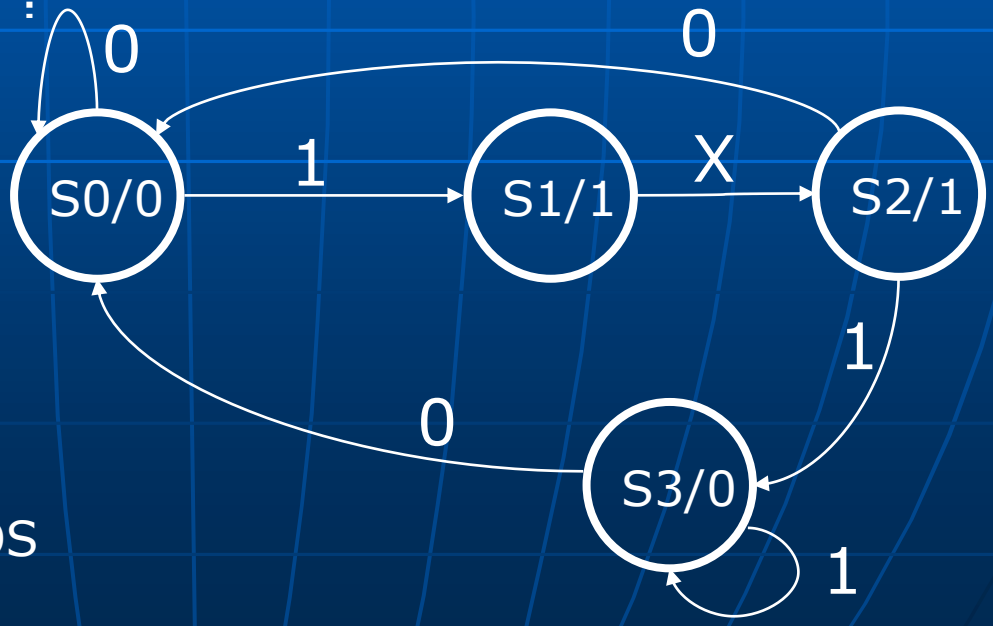


DIAGRAMA DE ESTADOS

Síntesis por Moore en VHDL

monoestable_moore_2.vhd — Compilation Report

```
1  library ieee; use ieee.std_logic_1164.all;
2  entity monoestable_moore_2 is
3     port
4     (
5        clk   : in  std_logic; y : in std_logic;
6        reset : in  std_logic; z : out   std_logic
7     );
8  end entity;
9  architecture rtl of monoestable_moore_2 is
10
11    type state_type is (s0, s1, s2, s3); signal state : state_type;
12 begin
13    process (clk, reset, y)
14    begin
15       if reset = '1' then state <= s0;
16       elsif (clk'event and clk = '1') then
17          case state is
18             when s0=>
19                if y = '1' then
20                   state <= s1;
21                else
22                   state <= s0;
23                end if;
24             when s1=>
25                if y = '1' then
26                   state <= s2;
27                else
28                   state <= s2;
29                end if;
30             when s2=>
31                if y = '1' then
32                   state <= s3;
33                else
34                   state <= s0;
35                end if;
36             when s3=>
37                if y = '1' then
38                   state <= s3;
39                else
40                   state <= s0;
41                end if;
42          end case;
43       end if;
44    end process;
45
46    process (state)
47    begin
48       case state is
49          when s0=> z <= '0';
50          when s1=> z <= '1';
51          when s2=> z <= '1';
52          when s3=> z <= '0';
53       end case;
54    end process;
55 end rtl;
```

Monoestable disparado por flanco ascendente
No-redisparable. Duración 2 ciclos de reloj.

Sección que define a que estado iré, dependiendo del valor de la entrada.

Sección que define cuanto valdrá la salida, dependiendo del estado.

tb_monoestable_moore_2.vhd | monoestable_moore_2.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3
4  entity tb_monoestable_2 is
5  end tb_monoestable_2;
6
7  architecture test of tb_monoestable_2 is
8
9     component monoestable_moore_2
10       Port (   clk      : in std_logic;
11                y        : in std_logic;
12                reset    : in std_logic;
13                z        : out std_logic
14           );
15    end component;
16
17 signal clk           : std_logic;
18 signal reset         : std_logic;
19 signal y             : std_logic;
20 signal z             : std_logic;
21
22 begin
23
24   uut: monoestable_moore_2 port map (  clk => clk,
25                                        y => y,
26                                        reset => reset,
27                                        z => z
28                                        );
29
30    gen_reloj : process  -- Reloj de 200 ns de periodo y 50% de ciclo de trabajo
31       begin
32          clk <= '0';
33          wait for 100 ns;
34          clk <= '1';
35          wait for 100 ns;
36    end process gen_reloj;
37
38    estimulos : process
39    begin
40             y   <= '0';
41             reset <= '0';
42             wait for 250 ns;
43             reset <= '1';
44             wait for 250 ns;
45             reset  <= '0';
46             wait for  250 ns;
47             y <= '1';
48             wait for 551 ns;
49             y   <= '0';
50             wait for  253 ns;
51             y <= '1';
52             wait for  1 us;
53   end process estimulos;
54 end test;
```

Monoestable disparado por flanco ascendente
No-redisparable. Duración 2 ciclos de reloj.

Usamos el mismo test-bench que para el ejercicio anterior.

Generación de los estímulos temporales de "clk", "reset" y la entrada "y" para emplear en el Modelsim y generar luego la gráfica temporal de la salida "z".

Generación de "clk".

Generación de "reset" e "y".

Compilation Report - monoestable_moore_2

Table of Contents

- Flow Summary
- Flow Settings
- Flow Non-Default Global Settings
- Flow Elapsed Time
- Flow OS Summary
- Flow Log
- Analysis & Synthesis
- Fitter
- Assembler
- TimeQuest Timing Analyzer
- EDA Netlist Writer
- Flow Messages
- Flow Suppressed Messages

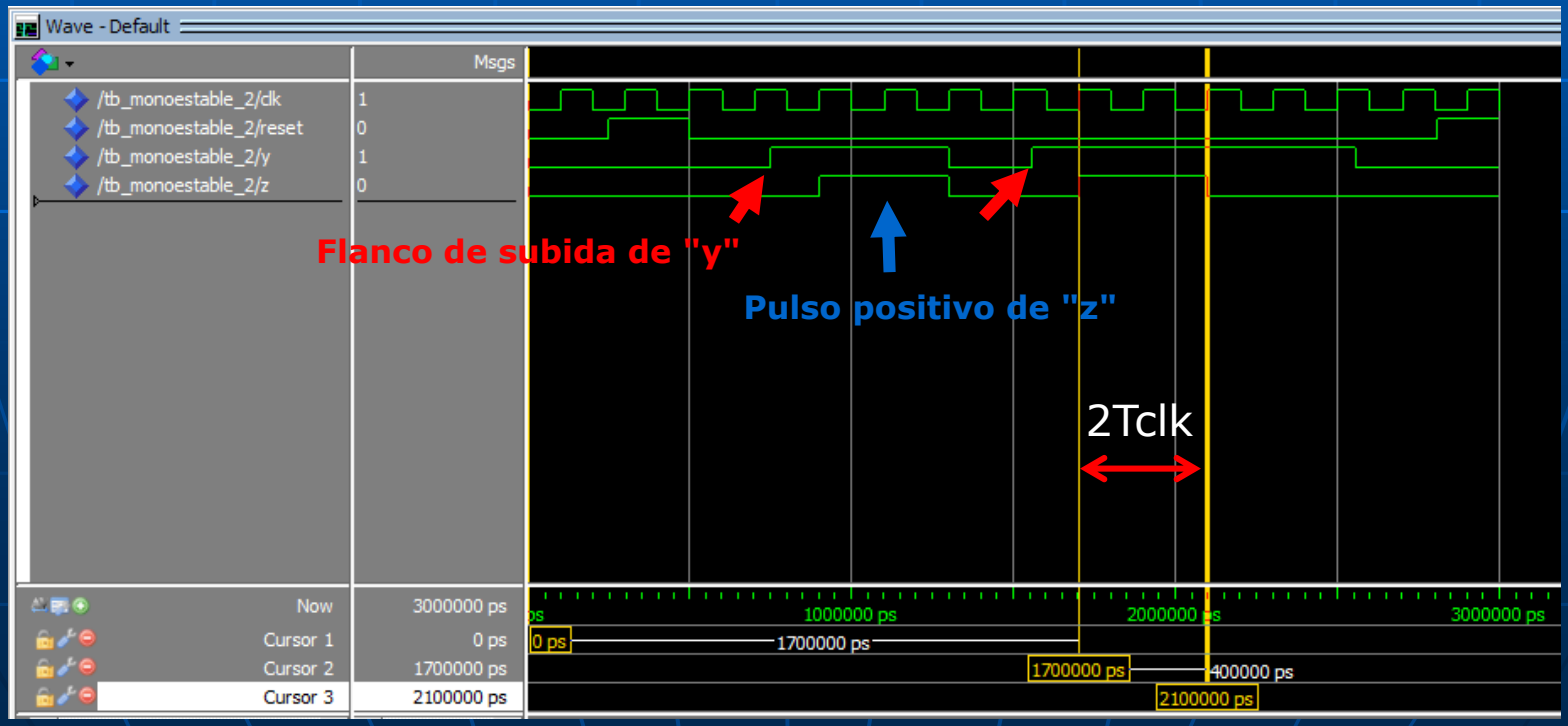
Flow Summary

Flow Status	Successful - Wed Dec 16 13:24:41 2020
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Web Edition
Revision Name	monoestable_moore_2
Top-level Entity Name	monoestable_moore_2
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	4 / 22,320 (< 1 %)
Total combinational functions	1 / 22,320 (< 1 %)
Dedicated logic registers	3 / 22,320 (< 1 %)
Total registers	3
Total pins	4 / 154 (3 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

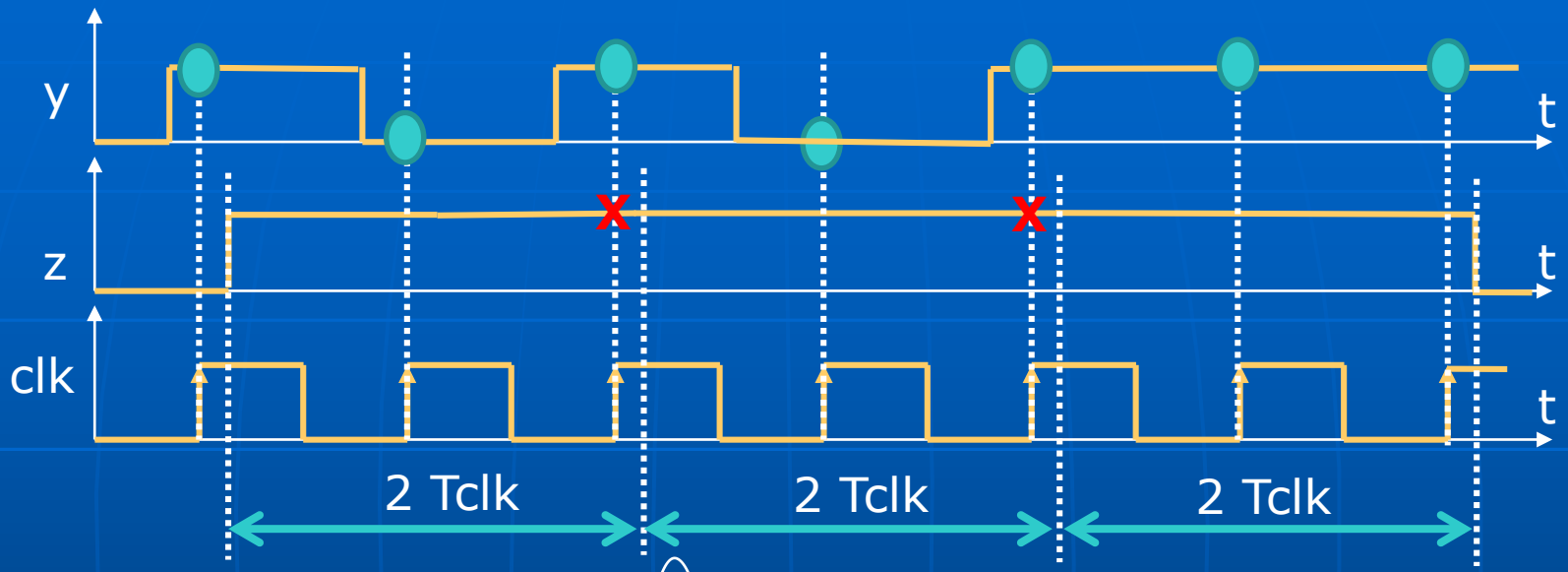
Monoestable disparado por flanco ascendente
No-redisparable. Duración 2 ciclos de reloj.

El proyecto insume 4 LE (Elementos Lógicos).

Dispositivo:
Cyclone IV E,
modelo EP4CE22F17C6.



Diseño de circuito monoestable disparado por flanco ascendente



Monoestable redisparable con duración de 2 ciclos de reloj.

En el ejemplo, el monoestable se redispara donde está la cruz roja.

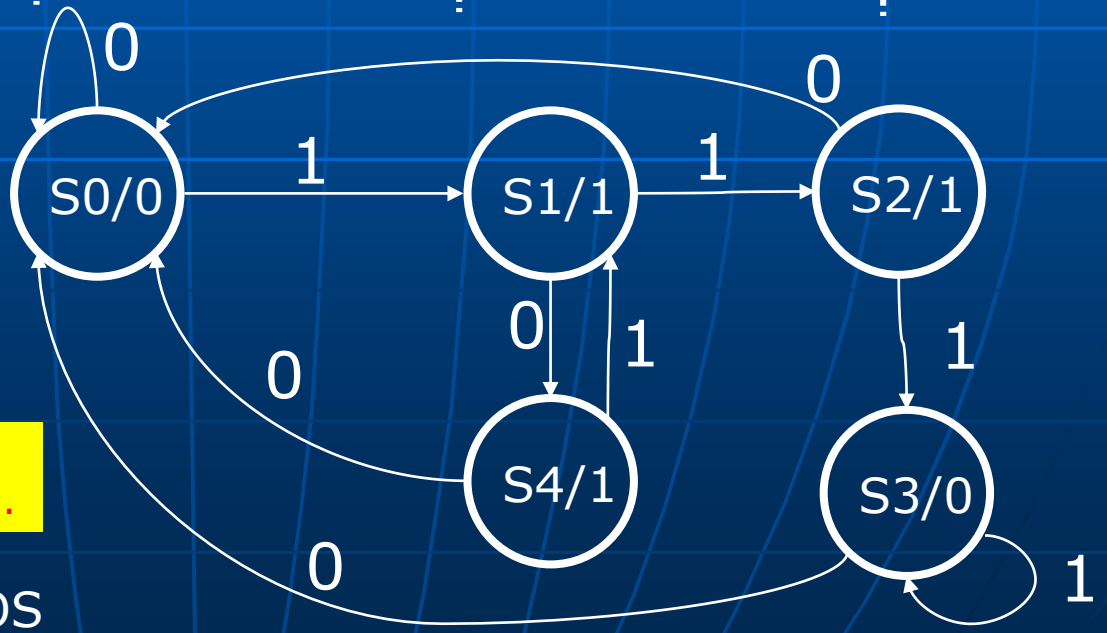


DIAGRAMA DE ESTADOS

monoestable_moore_3.vhd | tb_monoestable_

```
1  library ieee; use ieee.std_logic_1164.all;
2  entity monoestable_moore_3 is
3     port
4     (
5        clk   : in  std_logic; y : in std_logic;
6        reset : in  std_logic; z : out   std_logic
7     );
8  end entity;
9  architecture rtl of monoestable_moore_3 is
10
11    type state_type is (s0, s1, s2, s3, s4); signal state : state_type;
12 begin
13    process (clk, reset, y)
14    begin
15       if reset = '1' then state <= s0;
16       elsif (clk'event and clk = '1') then
17          case state is
18             when s0=>
19                if y = '1' then state <= s1;
20                else
21                   state <= s0;
22                end if;
23             when s1=>
24                if y = '1' then state <= s2;
25                else
26                   state <= s4;
27                end if;
28             when s2=>
29                if y = '1' then state <= s3;
30                else
31                   state <= s0;
32                end if;
33             when s3=>
34                if y = '1' then state <= s3;
35                else
36                   state <= s0;
37                end if;
38             when s4=>
39                if y = '1' then state <= s1;
40                else
41                   state <= s0;
42                end if;
43                end case;
44       end if;
45    end process;
46
47    process (state)
48    begin
49       case state is
50          when s0=> z <= '0'; when s1=> z <= '1';
51          when s2=> z <= '1'; when s3=> z <= '0';
52          when s4=> z <= '1';
53       end case;
54    end process;
55 end rtl;
```

Monoestable disparado por flanco ascendente
Redisparable. Duración 2 ciclos de reloj.

Sección que define a que estado iré, dependiendo del valor de la entrada.

Sección que define cuanto valdrá la salida, dependiendo del estado.

tb_monoestable_moore_3.vhd | monoestable_moore_3.vhd

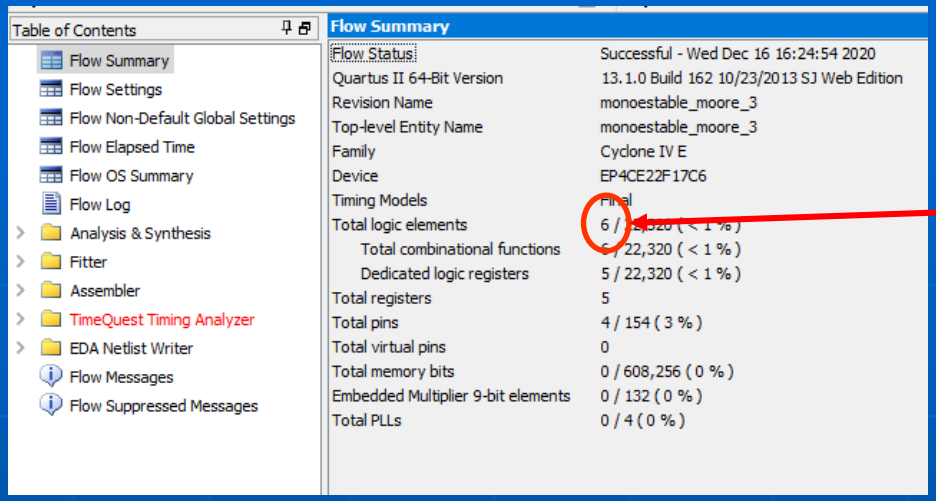
```
4   entity tb_monoestable_3 is
5   end tb_monoestable_3;
6
7   architecture test of tb_monoestable_3 is
8
9      component monoestable_moore_3
10        Port (   clk      : in std_logic;
11                 y        : in std_logic;
12                 reset    : in std_logic;
13                 z        : out std_logic
14        );
15     end component;
16
17  signal clk           : std_logic;
18  signal reset         : std_logic;
19  signal y             : std_logic;
20  signal z             : std_logic;
21
22  begin
23
24   uut: monoestable_moore_3 port map (  clk => clk,
25                                        y => y,
26                                        reset => reset,
27                                        z => z
28                                        );
29
30     gen_reloj : process  -- Reloj de 200 ns de periodo y 50% de ciclo de trabajo
31        begin
32           clk <= '0';
33           wait for 100 ns;
34           clk <= '1';
35           wait for 100 ns;
36     end process gen_reloj;
37
38     estimulos : process
39     begin
40              y   <= '0';
41              reset <= '0';
42              wait for 250 ns;
43              reset <= '1';
44              wait for 250 ns;
45              reset  <= '0';
46              wait for  250 ns;
47              y <= '1';
48              wait for 200 ns;
49              y   <= '0';
50              wait for  200 ns;
51              y <= '1';
52              wait for 600 ns;
53              y   <= '0';
54              wait for  200 ns;
55              y <= '1';
56              wait for  1 us;
57     end process estimulos;
58  end test;
```

Monoestable disparado por flanco ascendente Redisparable. Duración 2 ciclos de reloj.

Generación de los estímulos temporales de "clk", "reset" y la entrada "y" para emplear en el Modelsim y generar luego la gráfica temporal de la salida "z".

Generación de "clk".

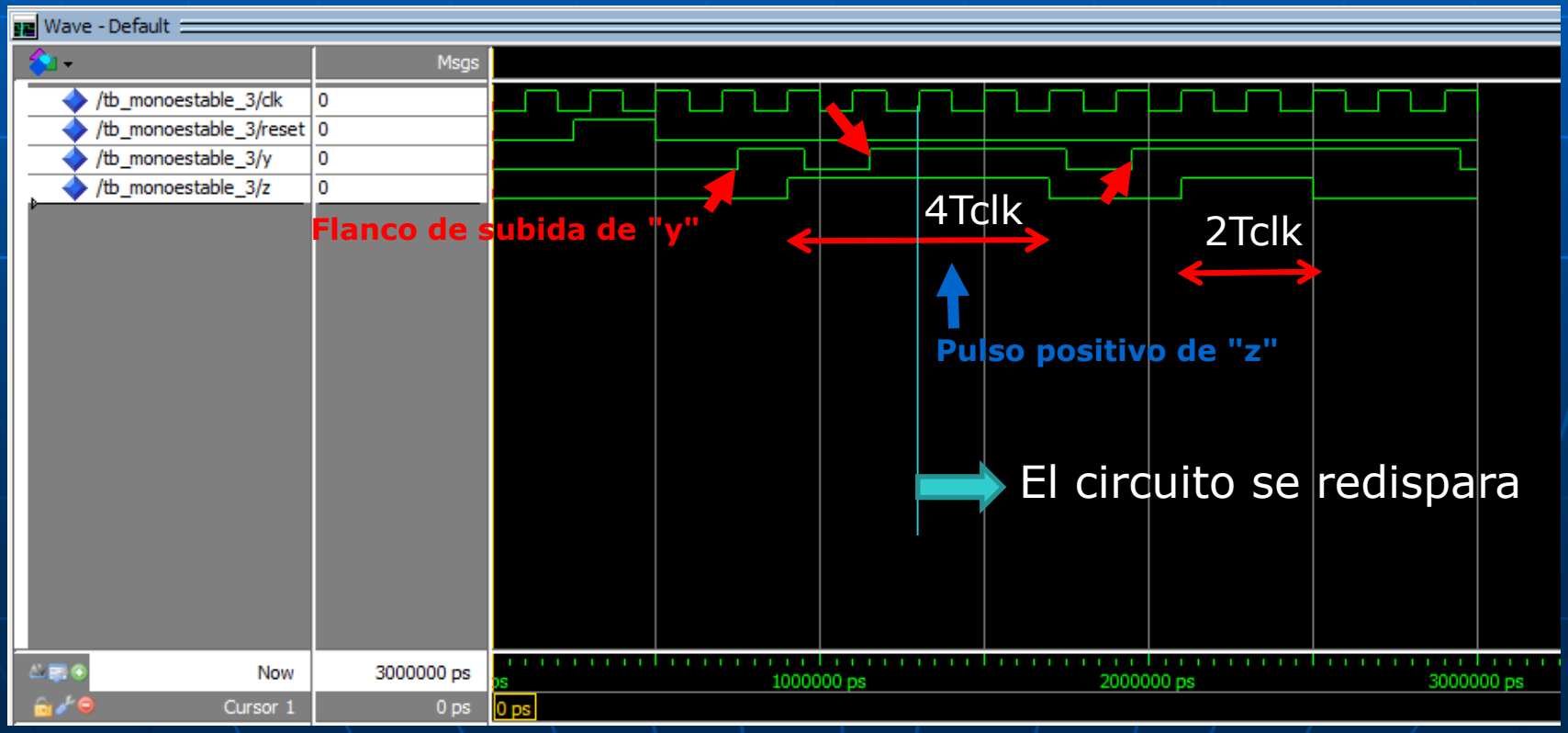
Generación de "reset" e "y".



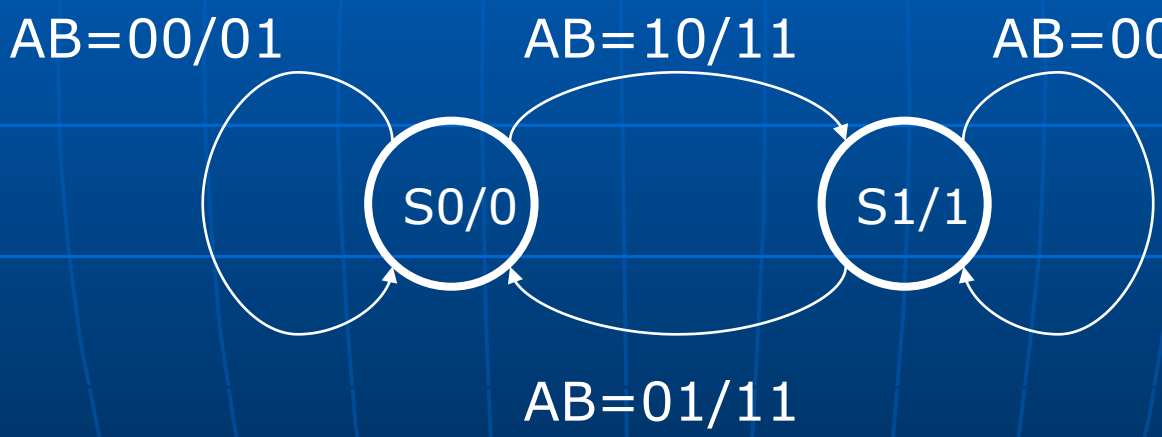
Monoestable disparado por flanco ascendente
Redisparable. Duración 2 ciclos de reloj.

El proyecto insume 6 LE (Elementos Lógicos).

Dispositivo:
Cyclone IV E,
modelo EP4CE22F17C6.



Diseñar en base a un flip-flop JK un circuito que responda con el siguiente diagrama de estados:



Método para lógica standard

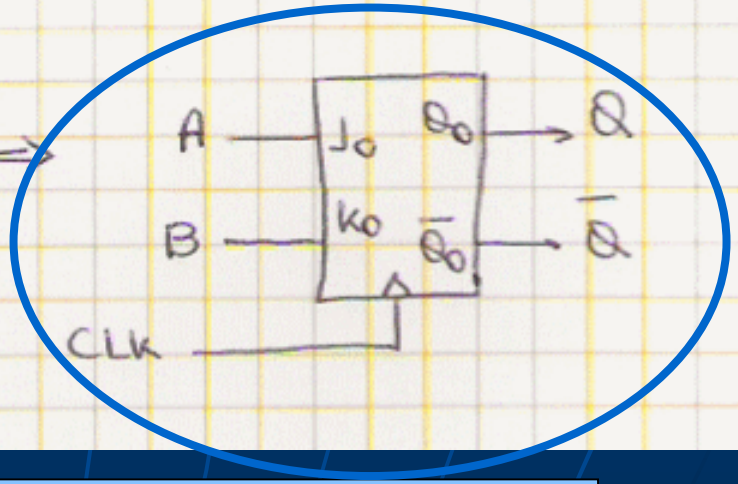
TABLA DE EXCITACIÓN

ESTADO ACTUAL	ESTADO SIGUIENTE				ENTRADAS ACTUALES J_0K_0			
0	0	0	1	1	0X	0X	1X	1X
1	1	0	1	0	X0	X1	X0	X1
	00	01	10	11	00	01	10	11
	AB				AB			

Q_{0_n} \ AB	$\bar{A}$ 00	$\bar{A}$ 01	A 11	A 10
$\bar{Q}_0$ 0	0X	0X	1X	1X
Q_0 1	X0	X1	X1	X0
	$\bar{B}$	B		$\bar{B}$

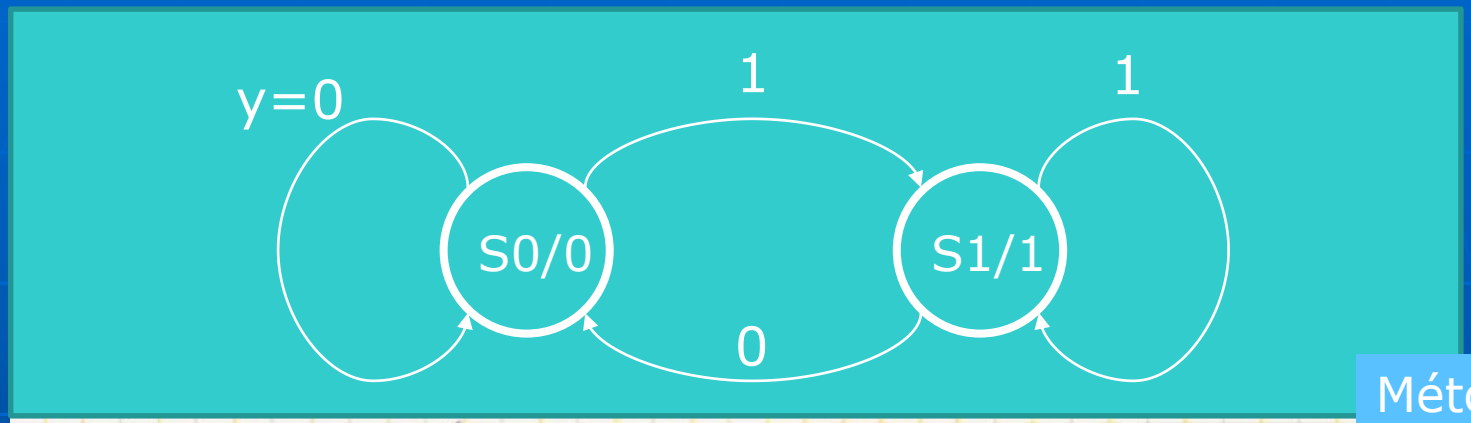
$J_0 = A$

$K_0 = B$



El diagrama de estados corresponde a un flip-flop "JK"..!!

Diseñar en base a un flip-flop D un circuito que responda con el siguiente diagrama de estados:



Método para lógica standard

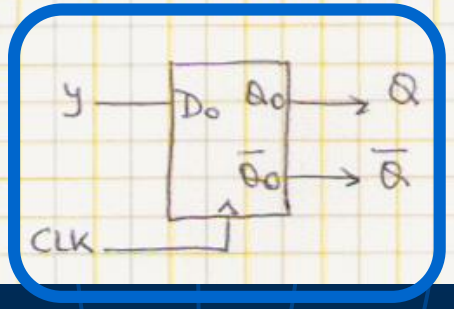
TABLA DE EXCITACIÓN

ESTADO ACTUAL	ESTADO SIGUIENTE		ENTRADA ACTUAL D_0	
	y = 0	y = 1	y = 0	y = 1
0	0	1	0	1
1	0	1	0	1

Q_{0_n} \ y	0	1
0	0	1
1	0	1

D_0

$D_0 = y$



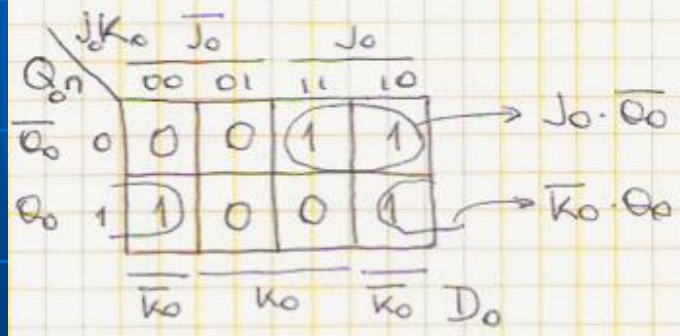
El diagrama de estados era el de un FF "D"

DISEÑO DE UN FF "JK" EN BASE A UNO TIPO "D"

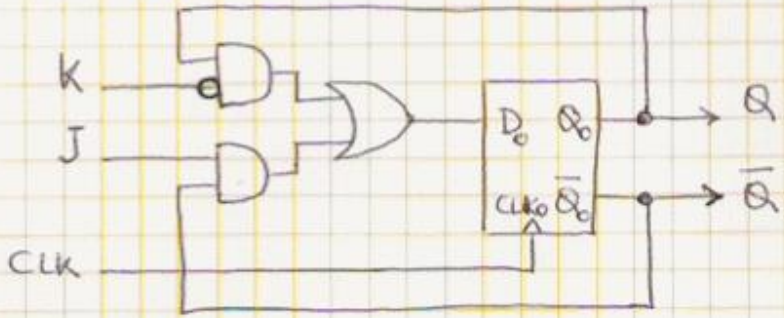
Método para lógica standard

TABLA DE EXCITACIÓN

Q_{0n}	Q_{0n+1}	J_0K_0	D_0
0	0	00	0
0	1	01	0
1	0	10	1
1	1	11	1

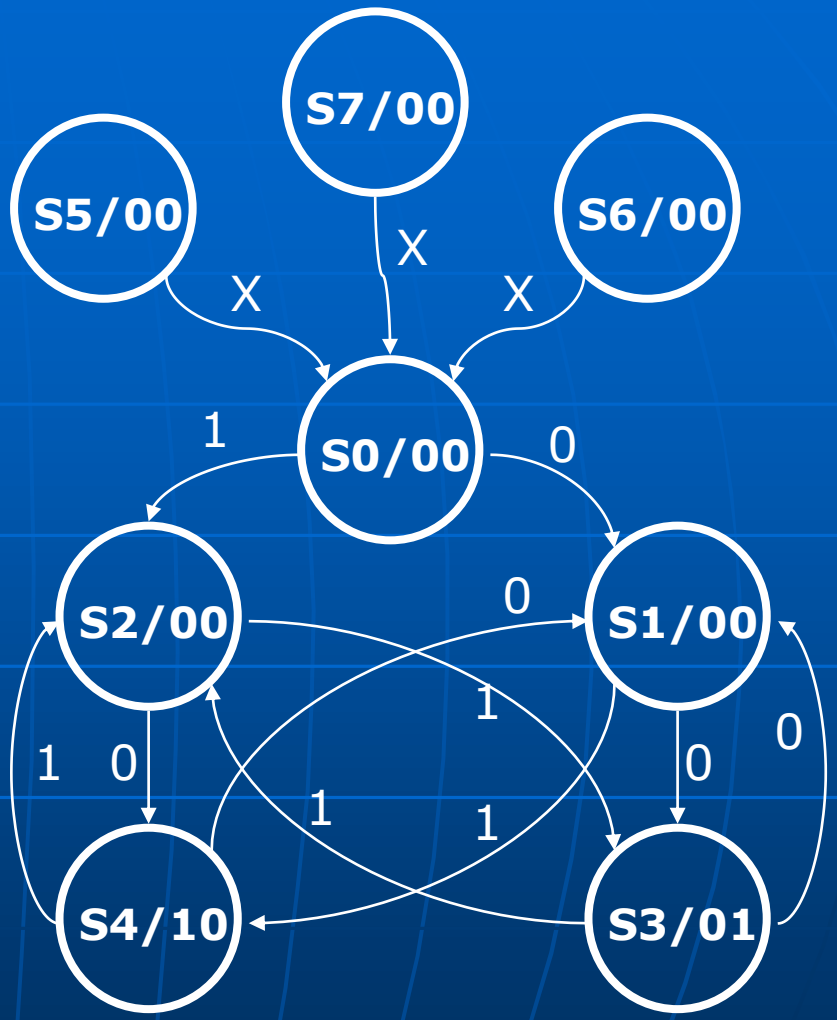


$$D_0 = J_0 \cdot \overline{Q_0} + \overline{K_0} \cdot Q_0$$



Detector de paridad par en formato serie de 2 bits de magnitud

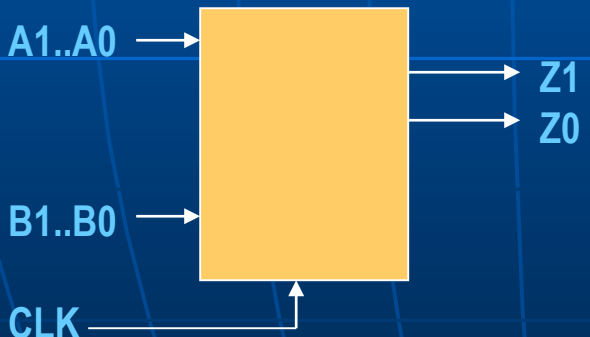
BITS ENTRANTES		SALIDAS	
D1	D0	Z1	Z0
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1
TRABAJANDO		0	0

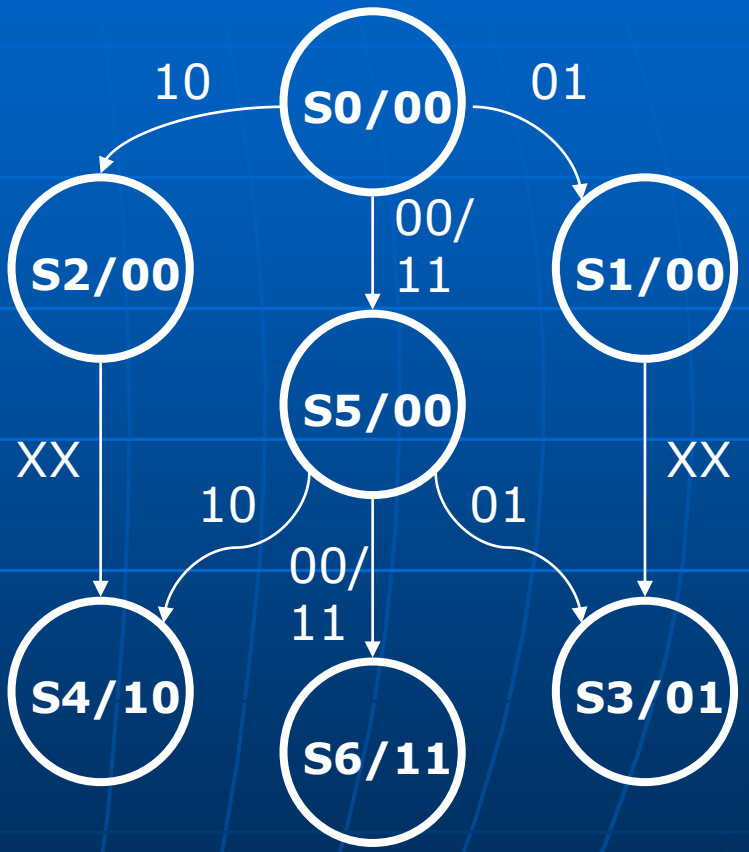


Diseño de comparador de magnitud serie de dos números sin signo (A y B) donde se transmite el bit mas significativo primero

Las salidas Z1Z0 deben cumplir con la siguiente tabla:

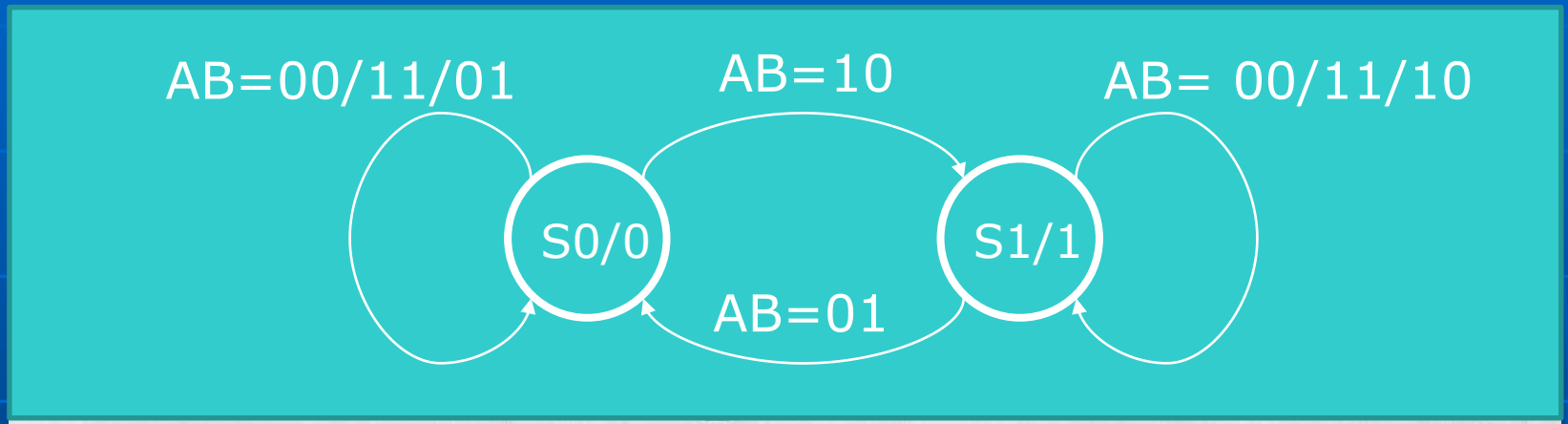
Si A>B => Z1Z0=10
Si A<B => Z1Z0=01
Si A=B => Z1Z0=11
Si se está comparando => Z1Z0=00





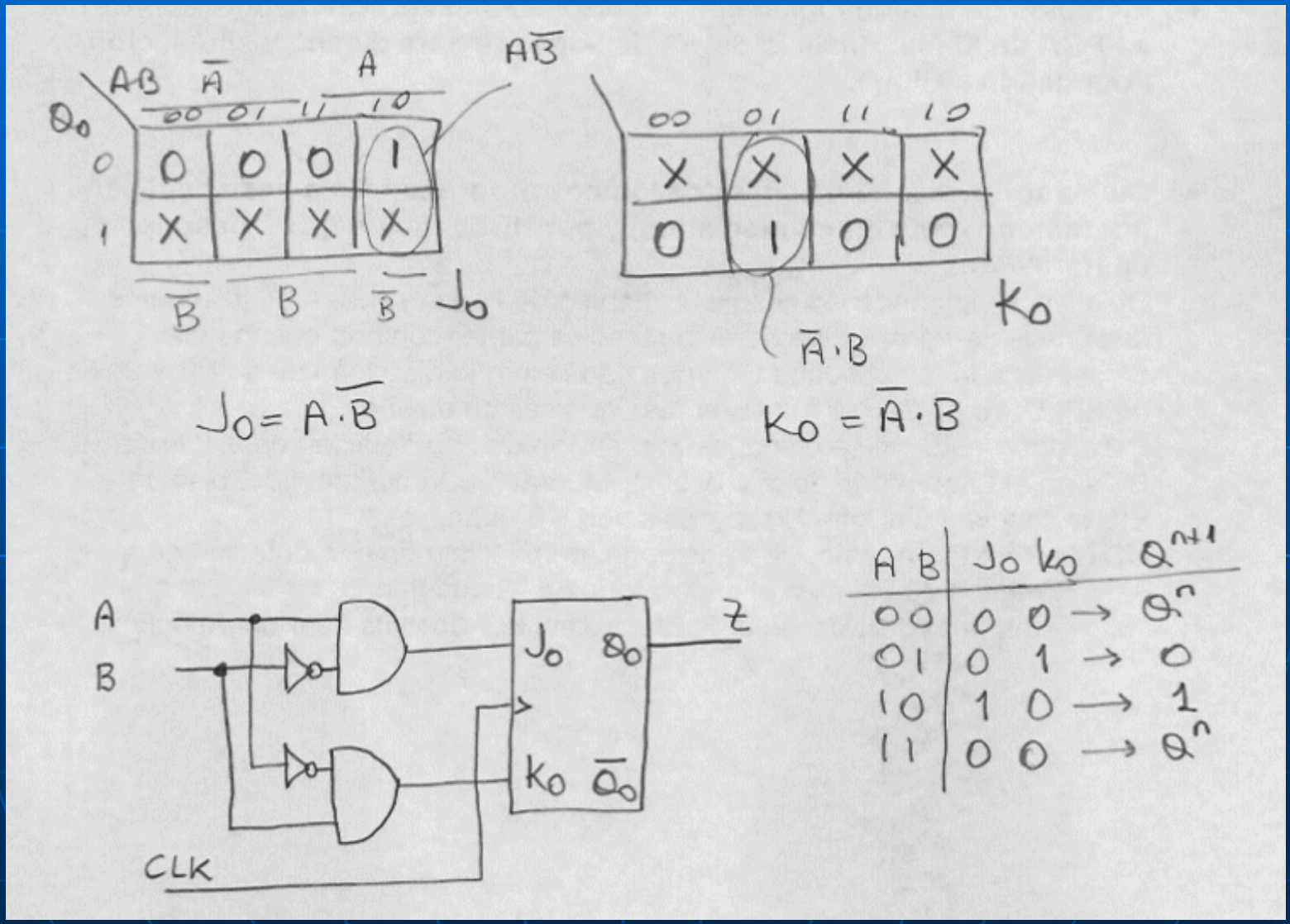
Comando de un motor con dos pulsadores A y B

Método para lógica standard



Q_0^n	E.A	E.S				z	Q_0^{n+1}				f_0 k_0			
0	S_0	S_0	S_0	S_1	S_0	0	0	0	1	0	0X	0X	1X	0X
1	S_1	S_1	S_0	S_1	S_1	1	1	0	1	1	X0	X1	X0	X0
		00	01	10	11		00	01	10	11	00	01	10	11
		AB					AB				AB			

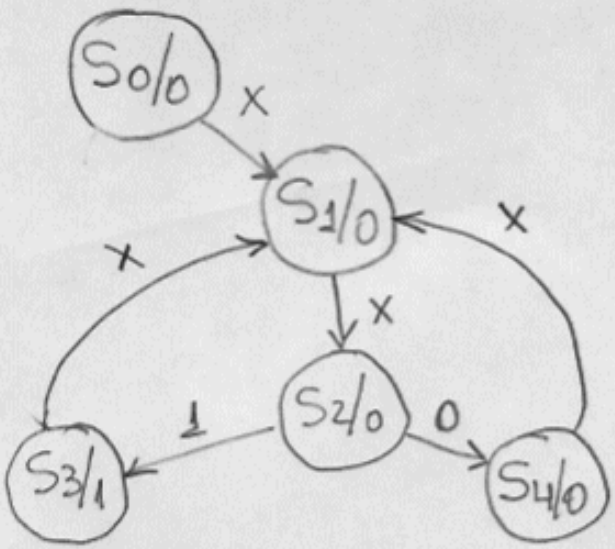
Comando de un motor con dos pulsadores A y B



Detector de número impar en formato serie cíclico, siendo el primer bit de entrada de cada secuencia, el MSB.

Sin solapamiento

1er bit MSB.

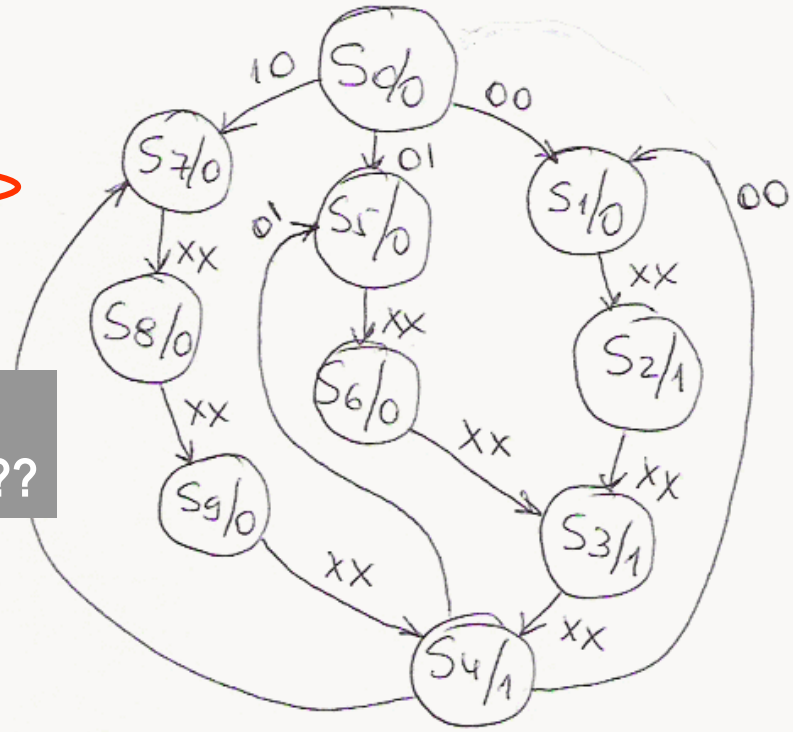


b_2 b_1 b_0	
000	impar => b0=1
001	
010	
011	
100	
101	
110	
111	

SINTETIZAR UN GENERADOR DE SEÑALES QUE GENERE LA SIGUIENTE SECUENCIA EN FORMA CÍCLICA:

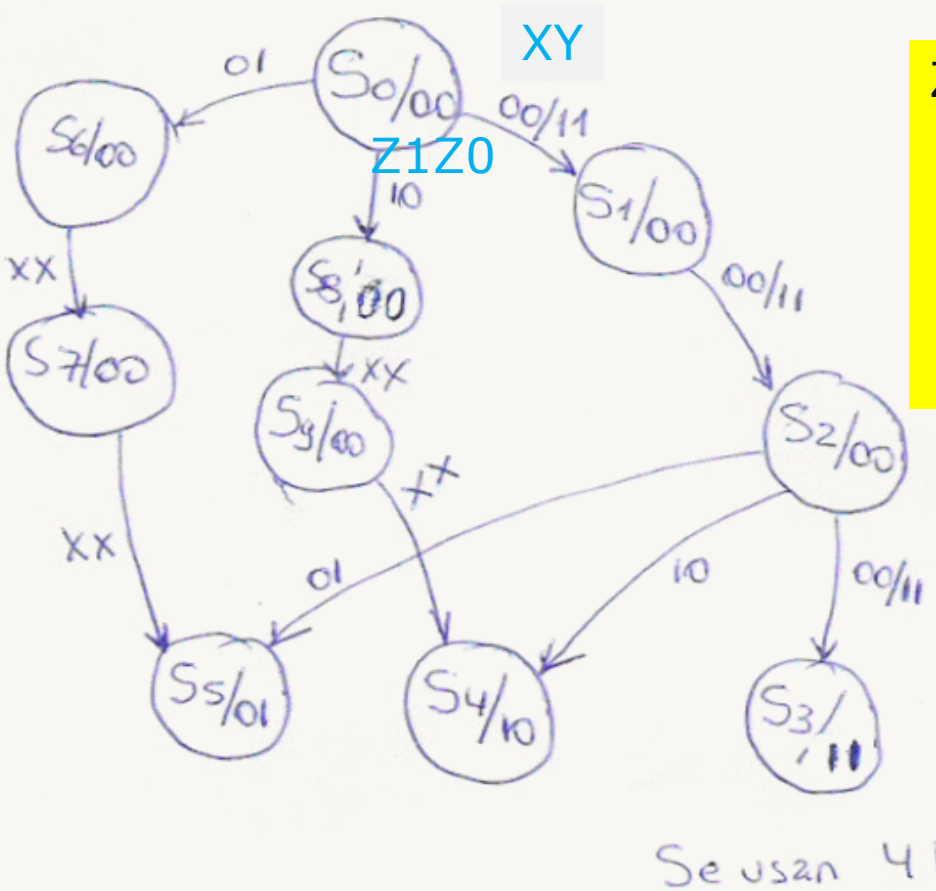
AB	secuencia
00	0 1 1 1
01	0 0 1 1
10	0 0 0 1
11	- - - -

Qué hacemos con esta combinación??



Las entradas se evalúan en el primer ciclo de cada nueva secuencia de entrada.

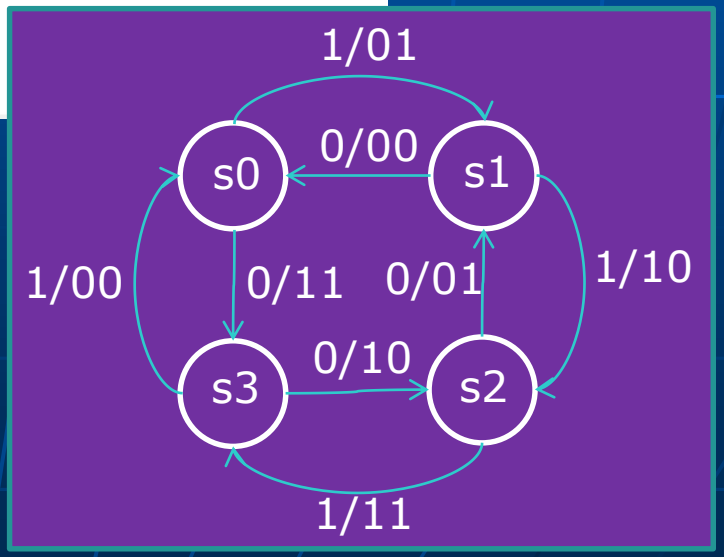
Sintetizar un circuito comparador de magnitud de dos números binarios sin signo de 3 bits en formato serie donde se manda primero el bit mas significativo MSB.



Z1	Z0	CONDICIÓN
0	0	en proceso
0	1	X < Y
1	0	X > Y
1	1	X = Y

Síntesis de contador up-down con máquina de Mealy. Descripción en VHDL

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity mealy_counter_2bits is

    port
    (
        clk      : in  std_logic;
        dir      : in  std_logic;
        reset    : in  std_logic;
        data_out : out std_logic_vector(1 downto 0)
    );

end entity;

architecture rtl of mealy_counter_2bits is

    type state_type is (s0, s1, s2, s3);

    signal state : state_type;

begin
    process (clk, reset, dir)
    begin
        if reset = '1' then
            state <= s0;
        elsif (rising_edge(clk)) then
            case state is
                when s0=>
                    if dir = '1' then
                        state <= s1;
                    else
                        state <= s3;
                    end if;
                when s1=>
                    if dir = '1' then
                        state <= s2;
                    else
                        state <= s0;
                    end if;
                when s2=>
                    if dir = '1' then
                        state <= s3;
                    else
                        state <= s1;
                    end if;
                when s3=>
                    if dir = '1' then
                        state <= s0;
                    else
                        state <= s2;
                    end if;
            end case;
        end if;
    end process;

    process (state, dir)
    begin
        case state is
            when s0=>
                if dir = '1' then
                    data_out <= "01";
                else
                    data_out <= "11";
                end if;
            when s1=>
                if dir = '1' then
                    data_out <= "10";
                else
                    data_out <= "00";
                end if;
            when s2=>
                if dir = '1' then
                    data_out <= "11";
                else
                    data_out <= "01";
                end if;
            when s3=>
                if dir = '1' then
                    data_out <= "00";
                else
                    data_out <= "10";
                end if;
        end case;
    end process;
end rtl;
```



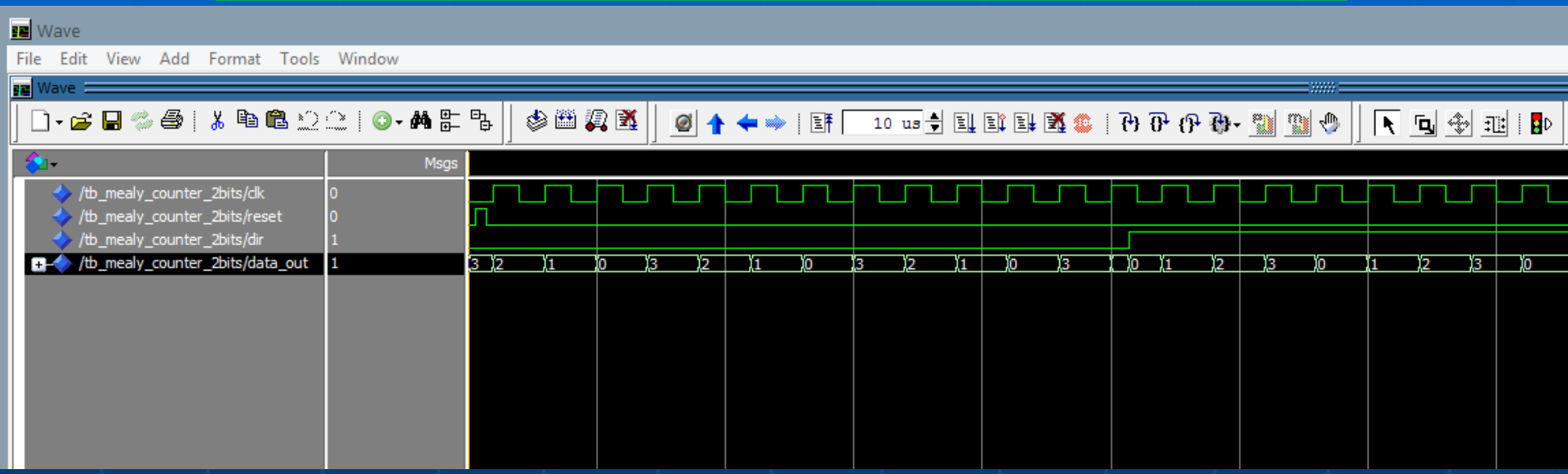
Síntesis por Mealy en VHDL

TEST BENCH DESCRIPTO EN VHDL del contador sintetizado con Mealy

```
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 tb_mealy_counter_2bits is
7  end tb_mealy_counter_2bits;
8
9  architecture test of tb_mealy_counter_2bits is
10
11    component mealy_counter_2bits
12       Port (   clk      : in std_logic;
13                dir      : in std_logic;
14                reset    : in std_logic;
15                data_out : out std_logic_vector(1 downto 0)
16           );
17    end component;
18
19 signal clk           : std_logic;
20 signal reset         : std_logic;
21 signal dir           : std_logic;
22 signal data_out      : std_logic_vector(1 downto 0);
23
24 begin
25
26  uut: mealy_counter_2bits port map (  clk => clk,
27                                       dir => dir,
28                                       reset => reset,
29                                       data_out => data_out
30                                    );
31
32    gen_reloj : process  -- Reloj de 200 ns de periodo y 50% de ciclo de trabajo
33       begin
34          clk <= '0';
35          wait for 100 ns;
36          clk <= '1';
37          wait for 100 ns;
38    end process gen_reloj;
39
40    estimulos : process
41
42    begin
43             dir   <= '0';
44             reset <= '0';
45             wait for 30 ns;
46             reset <= '1';
47             wait for 40 ns;
48             reset  <= '0';
49             wait for 2.5 us;
50             dir <= '1';
51             wait for 2.5 us;
52
53    end process estimulos;
54
55 end test;
```

Síntesis por Mealy en VHDL

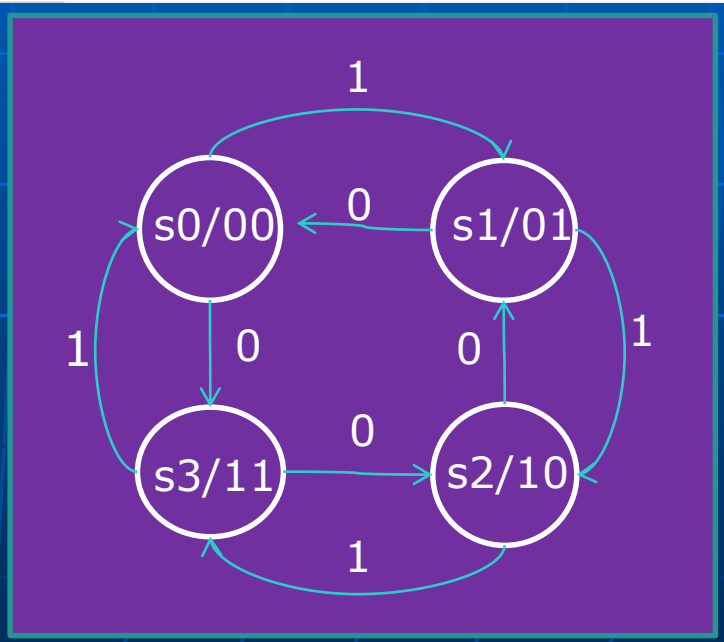
Simulación del contador implementado con máquina de Mealy



Síntesis de contador up-down con máquina de Moore. Descripción en VHDL

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity moore_counter_2bits is

    port
    (
       clk      : in  std_logic;
       dir      : in  std_logic;
       reset    : in  std_logic;
       data_out : out std_logic_vector(1 downto 0)
    );

end entity;

architecture rtl of moore_counter_2bits is

    type state_type is (s0, s1, s2, s3);

    signal state : state_type;

begin
    process (clk, reset, dir)
    begin
       if reset = '1' then
          state <= s0;
       elsif (rising_edge(clk)) then
          case state is
             when s0=>
                if dir = '1' then
                   state <= s1;
                else
                   state <= s3;
                end if;
             when s1=>
                if dir = '1' then
                   state <= s2;
                else
                   state <= s0;
                end if;
             when s2=>
                if dir = '1' then
                   state <= s3;
                else
                   state <= s1;
                end if;
             when s3=>
                if dir = '1' then
                   state <= s0;
                else
                   state <= s2;
                end if;
          end case;
       end if;
    end process;
```

```vhdl

    process (state)
    begin
       case state is
          when s0=>
                data_out <= "00";
          when s1=>
                data_out <= "01";
          when s2=>
                data_out <= "10";
          when s3=>
                data_out <= "11";
       end case;
    end process;
end rtl;

```



Síntesis por Moore en VHDL

TEST BENCH DESCRIPTO EN VHDL del contador sintetizado con Moore

```
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 tb_moore_counter_2bits is
7   end tb_moore_counter_2bits;
8
9   architecture test of tb_moore_counter_2bits is
10
11     component moore_counter_2bits
12        Port (   clk      : in std_logic;
13                 dir      : in std_logic;
14                 reset    : in std_logic;
15                 data_out : out std_logic_vector(1 downto 0)
16           );
17     end component;
18
19  signal clk           : std_logic;
20  signal reset         : std_logic;
21  signal dir           : std_logic;
22  signal data_out      : std_logic_vector(1 downto 0);
23
24  begin
25
26    uut: moore_counter_2bits port map (  clk => clk,
27                                         dir => dir,
28                                         reset => reset,
29                                         data_out => data_out
30                                      );
31
32     gen_reloj : process  -- Reloj de 200 ns de periodo y 50% de ciclo de trabajo
33        begin
34           clk <= '0';
35           wait for 100 ns;
36           clk <= '1';
37           wait for 100 ns;
38     end process gen_reloj;
39
40     estimulos : process
41
42     begin
43              dir   <= '0';
44              reset <= '0';
45              wait for 30 ns;
46              reset <= '1';
47              wait for 40 ns;
48              reset  <= '0';
49              wait for 2.5 us;
50              dir <= '1';
51              wait for 2.5 us;
52
53    end process estimulos;
54
55  end test;
```

Síntesis por Moore en VHDL

Simulación del contador implementado con máquina de Moore