Técnicas y Dispositivos Digitales II	Dep. de Ing. Electrónica y Computación.
Laboratorio 2: *Dispositivos Lógico Programables*	Área Digitales Fac. de Ingeniería. U.N.M.D.P.

OBJETIVO

- Introducir en el empleo de programación digital en FPGA.

CUESTIONES PRELIMINARES

GENERALES

1) Identifique qué elementos constituyen los LEs de la FPGA Cyclone III a emplear en el laboratorio y qué estructura tienen las LABs.
2) ¿De qué se trata el Nios® II?
3) ¿Qué diferencia existe entre IP cores y los bloques embebidos (ej multiplicador embebido) disponibles en la FPGA?
4) ¿Qué tipo de celda de programación posee el dispositivo FPGA Cyclone III?
5) Realice la descripción en VHDL de un Flip Flop JK.
6) Realice la descripción en VHDL de un restador completo de un bit.
7) Realice la descripción en VHDL del test bench del restador completo de un bit.

MATERIALES E INSTRUMENTAL

GENERAL

- Software Quartus II versión 13.0
  LINK descarga:
  https://drive.google.com/file/d/1TP38yhMcN5oDTF68ga1Au9g96NaeENuL/view?usp=sharing

Hojas de datos de Cyclone III: "Cyclone III Device Handbook.pdf"
LINK: https://www.intel.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/cyc3/cyclone3_handbook.pdf

- PC

DESARROLLO DEL LABORATORIO

El laboratorio consta de cuatro partes:
PARTE A: Implementación de un circuito combinacional en FPGA.
PARTE B: Implementación de un sumador completo con salidas registradas en VHDL.
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.
PARTE D: Implementación de una máquina de estado.

La entrega del laboratorio sea via Github, en el siguiente formulario: https://forms.gle/AWo5Y13oEADif6S96
Tenga en cuenta que los nombres de las carpetas NO pueden tener espacios, ejemplo:

C:/ JUAN_PEREZ/ PARTE_A/proy_1 ✓

C:/ JUAN PEREZ/ PARTE A/proy 1 ✗

PARTE A: Implementación de un circuito combinacional en FPGA.

1) Utilice el entorno Quartus II para crear un nuevo proyecto.

AYUDA:

En el entorno Quartus:

File ➔ New Project Wizard ➔ Next➔

Elija el directorio del proyecto.

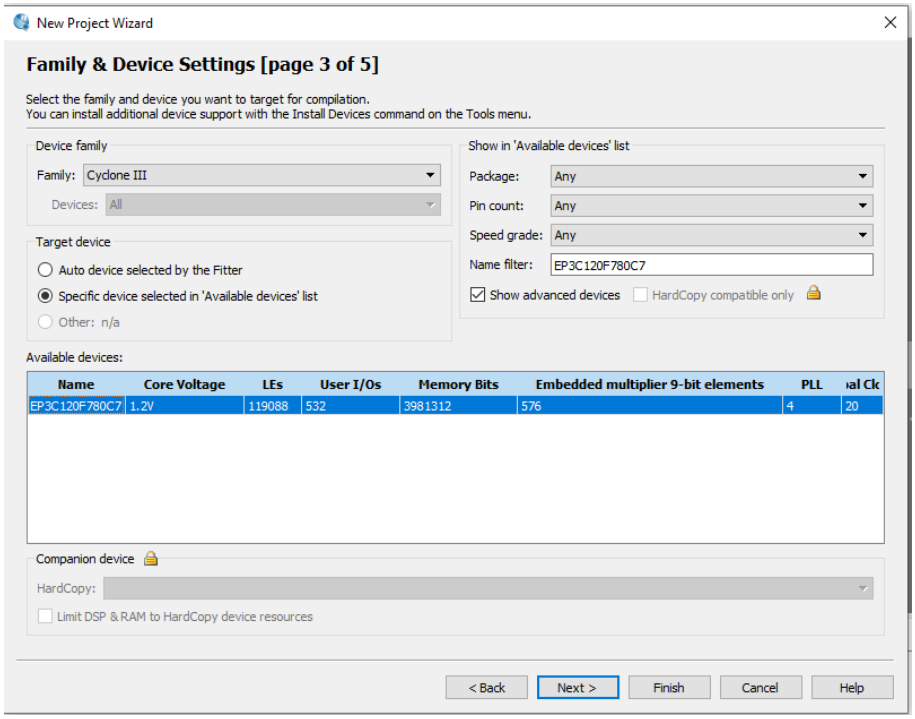
Elija el nombre del proyecto.

En Add files ➔ next (no agregamos ningún archivo)

Family & device settings: Family: Cyclone III , device: EP3C120F780C7➔ next

➔ next

➔ Finish

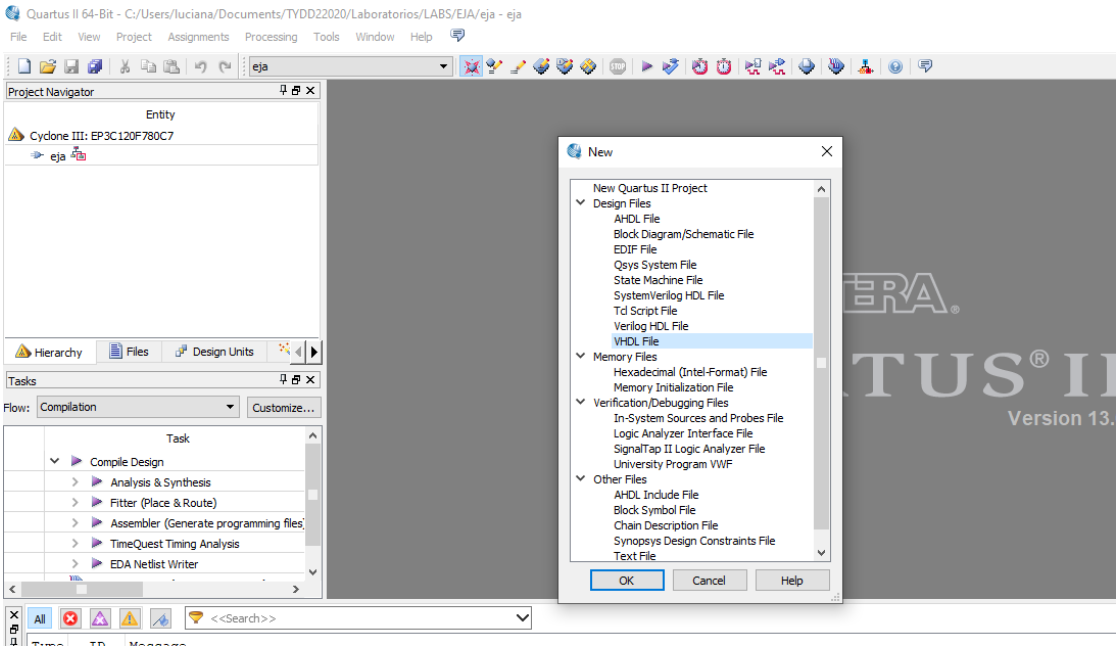


2) En el proyecto genere un archivo VHDL.

AYUDA:

En el entorno Quartus:

File➔ new➔ VHDL file ➔ OK



3) En el archivo VHDL describa el circuito combinacional mostrado en la Fig. 1, seteelo como Top Level entity compile.

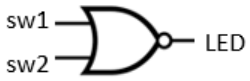


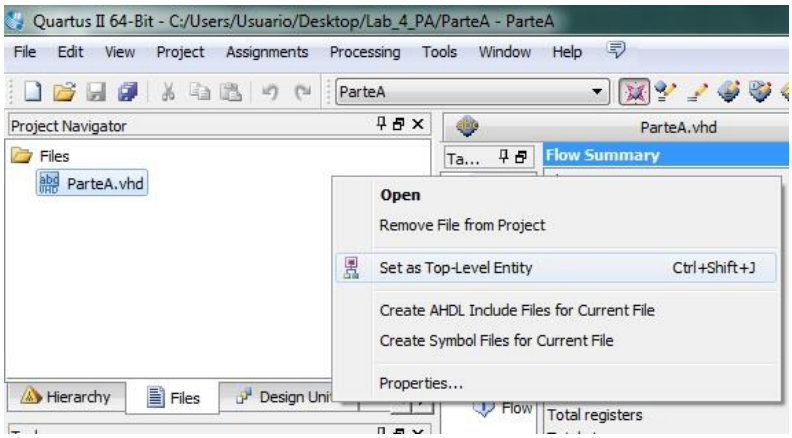
Fig. 1

AYUDA 1:

En el archivo VHDL deberá agregar el paquete IEEE.STD_LOGIC_1164.ALL de la librería IEEE, el cual contiene definiciones de tipos, subtipos y funciones.

AYUDA 2:

Para definir al archivo VHDL creado como Top Level entity:
En el Project Navigator vaya a la solapa Files, donde podrá ver al archivo VHDL creado. Seleccionando el archivo con el botón derecho del maus podrá setearlo como "Set as Top-Level Entity".



AYUDA 3:

Para compilar primero guarde el archivo VHDL y luego en Processing ➔ Start Compilation. (o CTR+L)
IMPORTANTE!!! El nombre de la entidad y el nombre del archivo vhdl deben ser iguales.

Ejemplo de un archivo combinacional en vhd que implementa la función $LED = \overline{SW1.SW2}$, ud. Deberá ingresar la función de la Fig. 1 (LED <= SW1 nor SW2) :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ej_combinacional is
    Port ( SW1 : in  STD_LOGIC;
           SW2 : in  STD_LOGIC;
           LED : out  STD_LOGIC);
end ej_combinacional;

architecture Behavioral of ej_combinacional is


begin

LED <= not (SW1 and SW2);

end Behavioral;
```

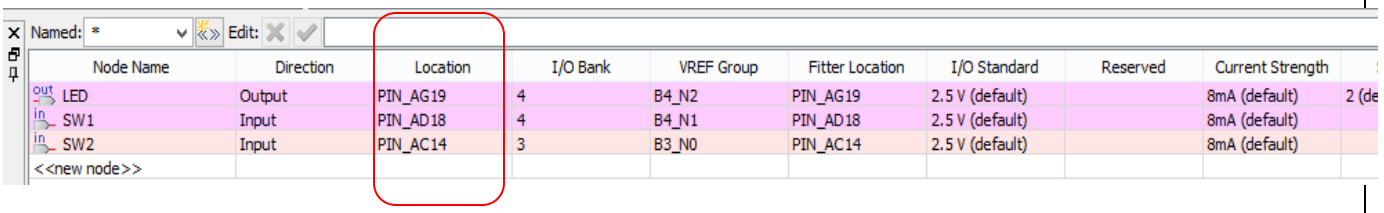
Fig. 2

4) Asigne los pines de entrada y salida del diseño. Compile nuevamente.

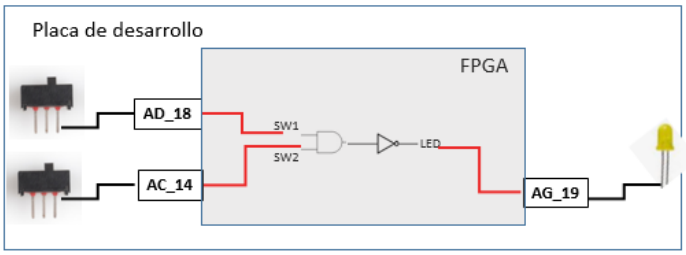
Asigne en este caso a las entradas SW1 y SW2 un switch a cada una y la salida LED un led.

AYUDA:

Busque los números de pines (necesitara dos switches de entrada y un led de salida) en el Manual de referencia "Cyclone III 3C120 Development Board", pags 37 y 38.
(https://www.intel.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/rm_cycloneiii_dev_kit_host_board.pdf)
También está en la pág. de la cátedra archivo rm_cycloneiii_dev_kit_host_board.pdf.
Luego, **en el entorno Quartus:** Assignments ➔ Pin Planner ➔ Location



Este paso genera las conecciones rojas que se ven en el siguiente diagrama:



5) Verifique el circuito implementado mediante el visor de RTL.

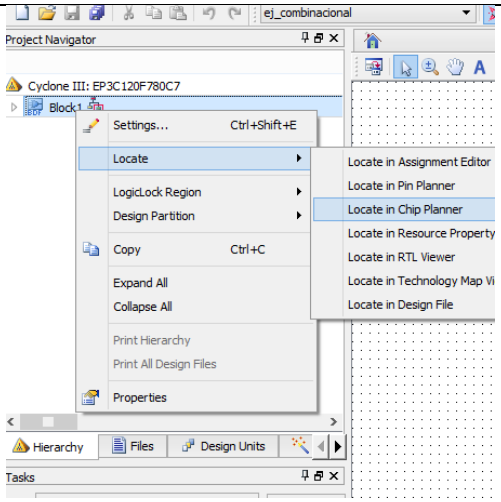
AYUDA:
En el entorno Quartus:
Tools➔ Netlist Viewer ➔ RTL view
¿Es el circuito implementado? ¿Qué hizo el sintetizador?

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

AYUDA:
En el entorno Quartus:
Tools ➔ Netlist Viewer ➔ Technology Map Viewer (past- Mapping y post-Fitting)

7) Verifique la implementación en el chip.

AYUDA:
En el entorno Quartus:
En la ventana: Project Navigator, solapa: Hierarchy, click derecho➔Locate➔Locate in Chip Planner.



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

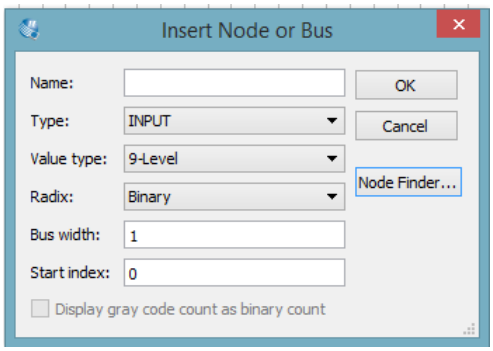
AYUDA:

En el entorno Quartus:

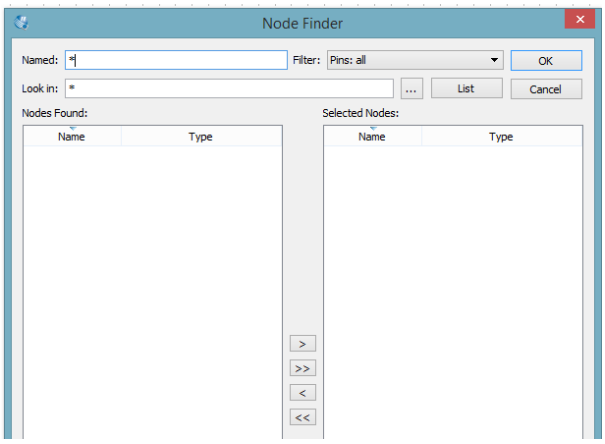
Cree un archivo de Estímulos (.vwf): File ➔ New ➔ University Program VWF

Guarde el archivo en el mismo directorio.

Agregue las señales de entrada y salida a simular: Edit ➔ Insert ➔ Insert Node or Bus



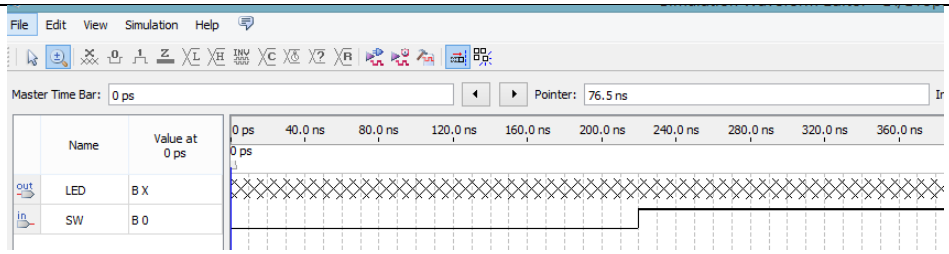
Node Finder➔List



Agregue las señales de entrada y salida que se deseen evaluar, luego OK.

En el Editor de forma de onda de Simulación (Simulation Waveform Editor):

Asigne valores a las entradas.



Realice la simulación funcional: Simulation➔Run Functional Simulation.
Realice la simulación temporal: Simulation➔Run Timing Simulation.

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

Escriba un archivo testbench VHDL y simúlelo en ModelSim.

PRIMERO CONFIGURE EL QUARTUS SEGÚN EL ARCHIVO lab3_2020_CONFIGURACION.PDF (2 configuraciones)

AYUDA:

En el entorno Quartus:

Cree archivo VHDL:

File ➔ new ➔ VHDL file

y escriba el archivo testbench**.**

Ejemplo de archivo testbench correspondiente al circuito de la Fig. 2:

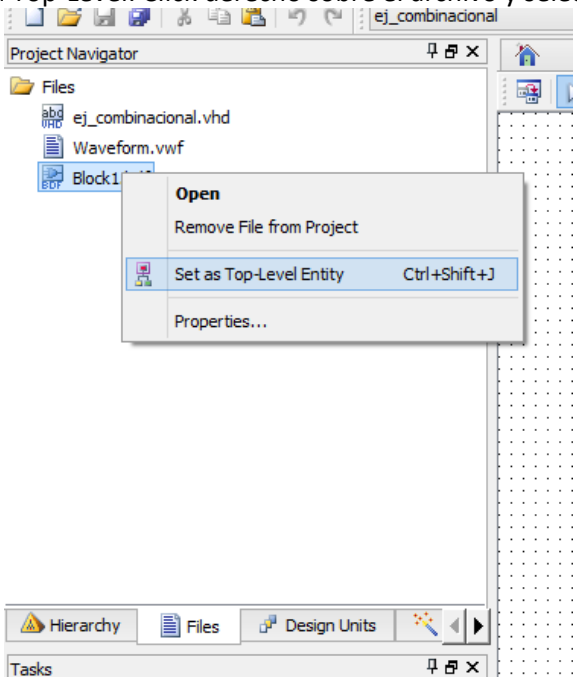
```
ARCHITECTURE behavior OF ej_combinacional_testbench IS

   COMPONENT ej_combinacional --component declaration
   PORT(
     SW1 : in  STD_LOGIC;
     SW2 : in  STD_LOGIC;
    LED : out  STD_LOGIC
     );

   END COMPONENT;

   --Inputs
   signal SW1 : std_logic := '0';
   signal SW2 : std_logic := '0';
   --Outputs
   signal LED : std_logic;

BEGIN
    -- Instantiate the Unit Under Test (UUT)
   uut: ej_combinacional PORT MAP (
      SW1    => SW1,
      SW2    => SW2,
      LED => LED
   );

   stim_proc: process -- Stimulus process
   begin
      --stimulus
        SW1 <= '0'; SW2 <= '0'; wait for 10ns;
      SW1 <= '0'; SW2 <= '1'; wait for 10ns;
        SW1 <= '1'; SW2 <= '0'; wait for 10ns;
        SW1 <= '1'; SW2 <= '1'; wait for 10ns;

      wait;
   end process;
END;
```

Fig. 3

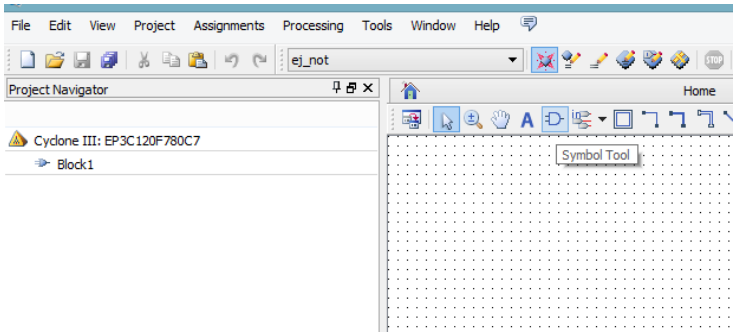
Setee el archivo testbench a emplear por ModelSim
Luego realice la simulación funcional:
Desde Quartus: Tools➔Run Simulation Tool➔RTL Simulation
Luego realice la simulación temporal:
Desde Quartus: Tools➔Run Simulation Tool➔ Gate Level Simulation

10) En el mismo proyecto genere un archivo esquemático.

AYUDA:
En el entorno Quartus:
File ➔ new ➔ Block Diagram/Schematic File
Setee este archivo como entidad Top-Level: Click derecho sobre el archivo y seleccione "Set as Top Level-Entity".

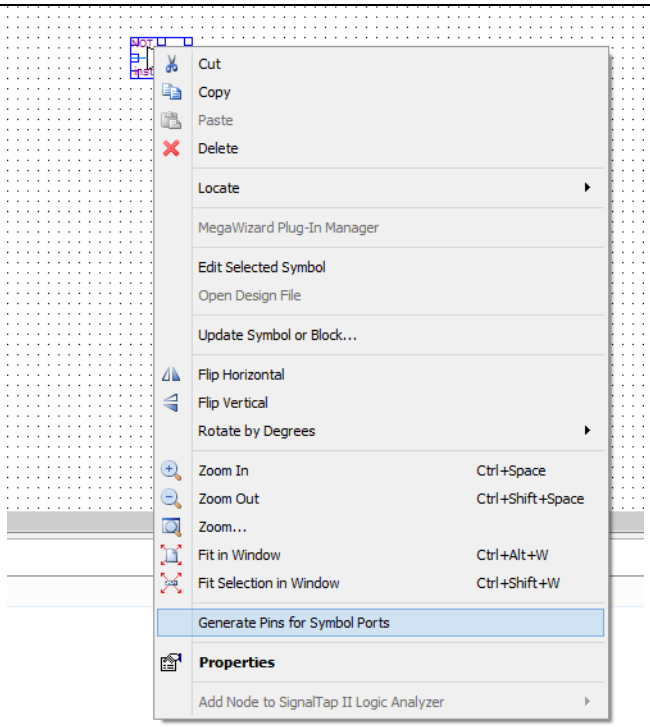


10.a En el archivo esquemático agregue los símbolos de los componentes necesarios (Symbol Tool), realice las conexiones necesarias.



Genere los puertos de entrada y salida.

AYUDA:
En el entorno Quartus:
Automáticamente Quartus genera los puertos, luego haciendo doble click sobre cada uno es posible modificar el nombre asignado.
Click derecho sobre el componente ➔ Generate Pins for Symbol Ports.



Asigne los pines de entrada y salida del diseño. Compile nuevamente.
Verifique.
Simule.

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

A partir del funcionamiento de un sumador completo cree en Quartus II un proyecto en el cual implemente un sumador completo. Latchee las señales de entrada y salida mediante Flip-Flops D.

Realice los siguientes paso:

1) Utilice el entorno Quartus II para crear un nuevo proyecto.

2) En el proyecto genere un archivo VHDL en el cual describa un Flip Flop D.

Ejemplo:

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

entity D_FF is
PORT( D,CLOCK: in std_logic;
Q: out std_logic);
end D_FF;

architecture behavioral of D_FF is
begin
process(CLOCK)
begin
if(CLOCK='1' and CLOCK'EVENT) then
Q <= D;
end if;
end process;
end behavioral;
```

3) En el proyecto genere un archivo VHDL en el cual describa el circuito sumador completo (defina como Top Level a este archivo). Compile.

3.1) Dentro de este VHDL deberá declarar al módulo Flip Flop y señales para conectar el circuito combinacional con los Flip Flops y las salidas/entradas. Luego en la arquitectura deberá instanciar la cantidad de Flip Flops necesarias y realizar las conexiones.

Ejemplo:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sumador_completo is
    Port (
        in_a    : in STD_LOGIC;
        in_b    : in STD_LOGIC;
        in_cin  : in STD_LOGIC;
          clk   : in STD_LOGIC;
        o_f     : out STD_LOGIC;
        o_cout  : out STD_LOGIC);
end sumador_completo;

architecture Behavioral of sumador_completo is

component D_FF
    port(D,CLOCK: in std_logic;
         Q: out std_logic);
    end component;

    signal a,b,cin,cout,f: std_logic;
begin

   f <= a xor b xor cin;
   cout <= (a and b) or (cin and (a xor b));

D1: D_FF port map (in_a,clk,a);
D2: D_FF port map (in_b,clk,b);
D3: D_FF port map (in_cin,clk,cin);
D4: D_FF port map (cout,clk,o_cout);
D5: D_FF port map (f,clk,o_f);
end Behavioral;
```

4) Asigne los pines de entrada y salida del diseño. Compile nuevamente.

5) Verifique el circuito implementado mediante el visor de RTL.

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

7) Verifique la implementación en el chip.

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

9) Verifique el funcionamiento del circuito mediante la simulación de Modelsim.

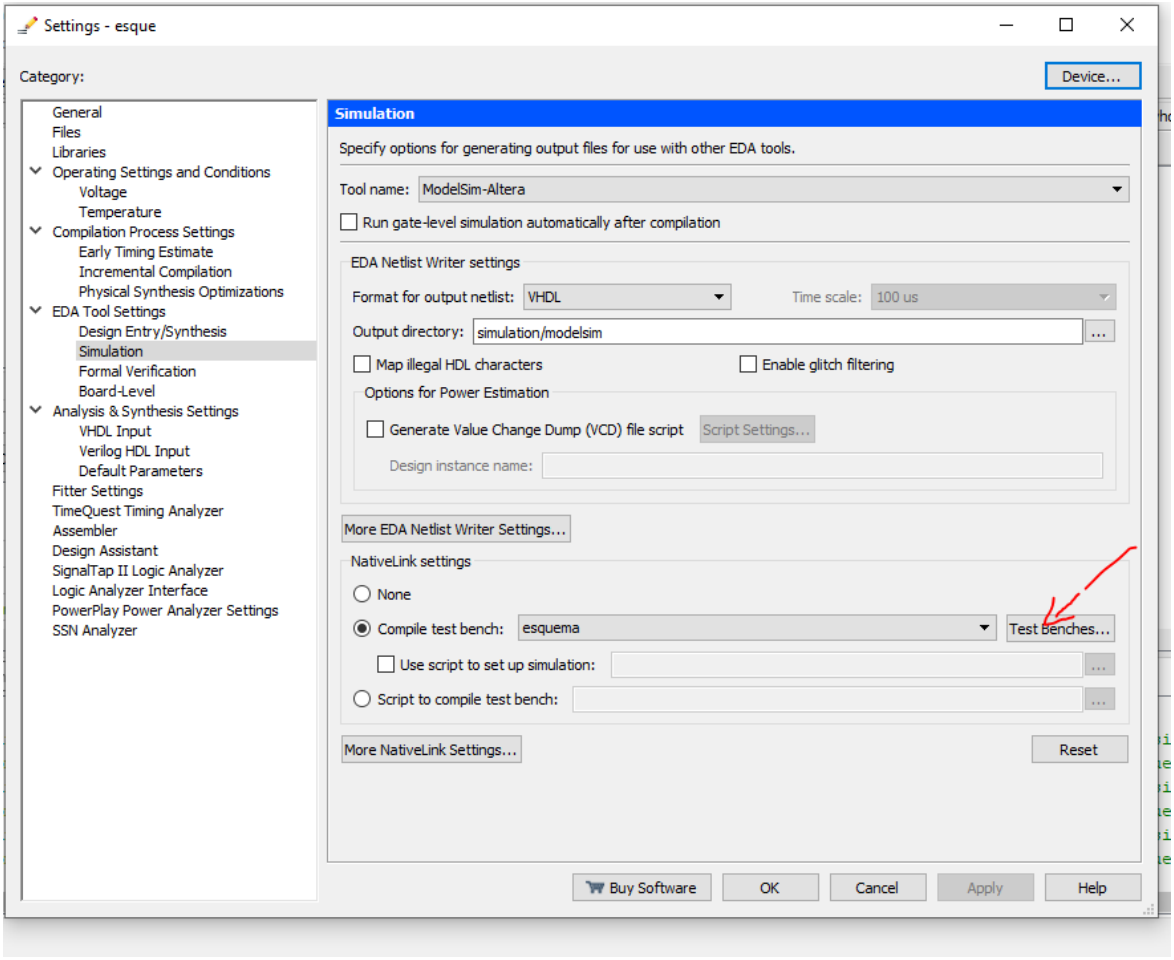
Ejemplo testbench de sumador completo:

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY sumador_completo_testbench IS
END sumador_completo_testbench;

ARCHITECTURE behavior OF sumador_completo_testbench IS

    COMPONENT sumador_completo --component declaration
    PORT(
         in_a    : in STD_LOGIC;
         in_b    : in STD_LOGIC;
         in_cin  : in STD_LOGIC;
           clk   : in STD_LOGIC;
         o_f     : out STD_LOGIC;
         o_cout  : out STD_LOGIC
        );
    END COMPONENT;

    --Inputs
    signal in_a : std_logic := '0';
    signal in_b : std_logic := '0';
    signal in_cin : std_logic := '0';
     signal clk : std_logic := '0';
    --Outputs
    signal o_f : std_logic;
    signal o_cout : std_logic;

--Clock period definition
constant clock_period : time := 20ns;
```

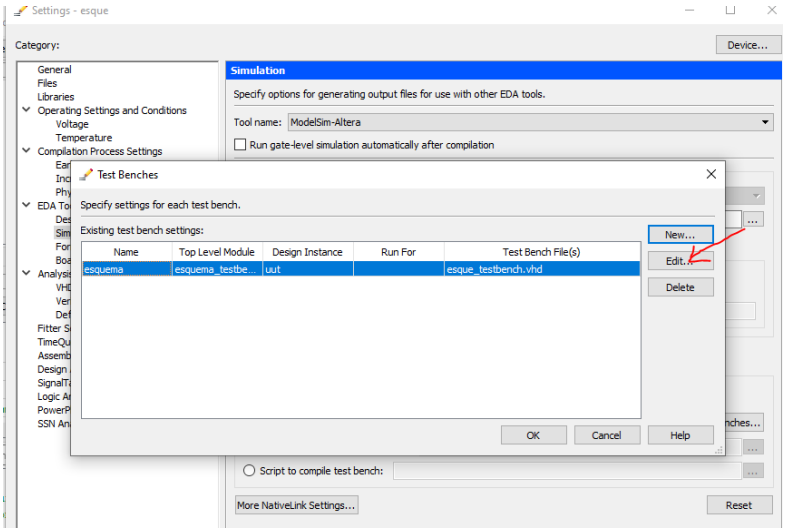
```vhdl
BEGIN
     -- Instantiate the Unit Under Test (UUT)
    uut: sumador_completo PORT MAP (
         in_a    => in_a,
         in_b    => in_b,
         in_cin  => in_cin,
           clk => clk,
         o_f     => o_f,
         o_cout => o_cout
    );

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

    stim_proc: process -- Stimulus process
    begin
         --stimulus
         in_a <= '0'; in_b <= '0'; in_cin <= '0'; wait for 30ns;
         in_a <= '0'; in_b <= '0'; in_cin <= '1'; wait for 30ns;
         in_a <= '0'; in_b <= '1'; in_cin <= '0'; wait for 30ns;
         in_a <= '0'; in_b <= '1'; in_cin <= '1'; wait for 30ns;
         in_a <= '1'; in_b <= '0'; in_cin <= '0'; wait for 30ns;
         in_a <= '1'; in_b <= '0'; in_cin <= '1'; wait for 30ns;
         in_a <= '1'; in_b <= '1'; in_cin <= '0'; wait for 30ns;
         in_a <= '1'; in_b <= '1'; in_cin <= '1'; wait for 30ns;
         wait;
    end process;
END;
```

Configuración del tiempo de simulación:

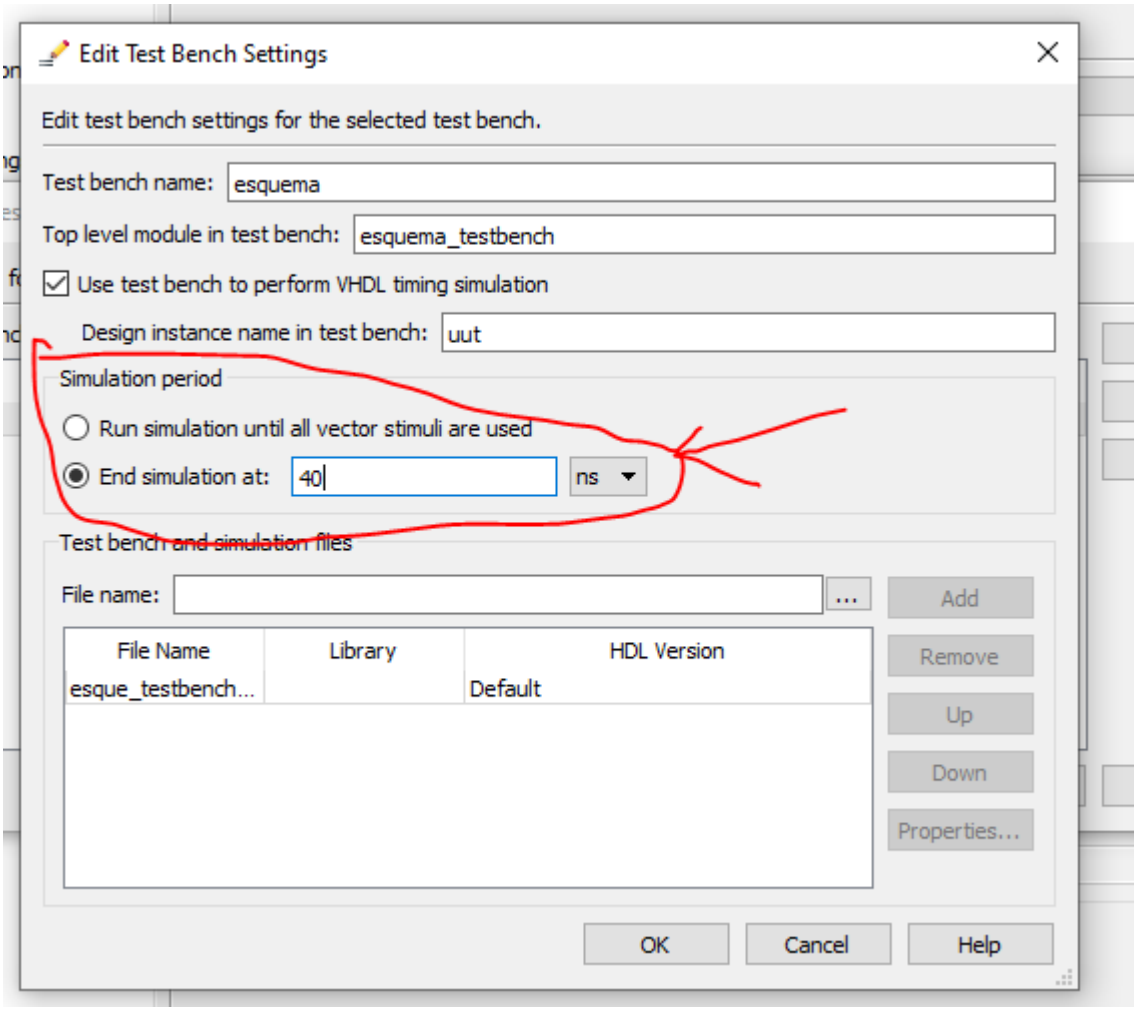
Assignemts=>Settings=>simulation=>



Compile testbench, boton test benches, ahi agregan el archivo testbench o editar el que ya se tenga cargado

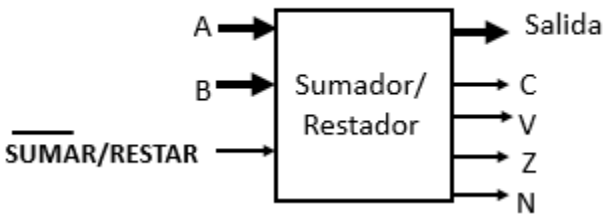


Se puede elegir un tiempo determinado o que se simule mientras haya estímulos:



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.

A partir del sumador completo creado en VHDL en el punto anterior cree en Quartus II un proyecto esquemático en el cual se implemente un sumador/restador de 4 bits y los bits de CCR (V, N, Z y C). Latchee las señales de entrada y salida mediante Flip-Flops D.



Realice los siguientes pasos:

1) Utilice el entorno Quartus II para crear un nuevo proyecto.

2) En el proyecto genere un archivo esquemático (defina como Top Level a este archivo).

3) Agregue en el proyecto un archivo VHDL que implemente el sumador completo, como el desarrollado en el punto anterior pero SIN registrar las entradas y salidas, ya que en este caso será un componente interno de un circuito mayor, por lo que se deben registrar las entradas y salidas del circuito final.

AYUDA:

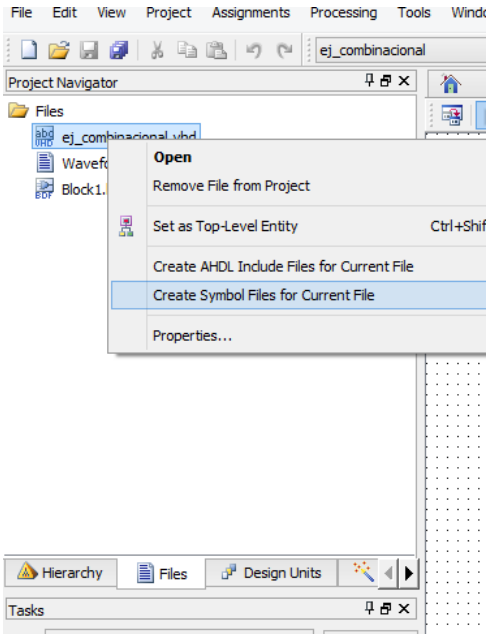
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sumador_completo is
    Port (
        a    : in STD_LOGIC;
        b    : in STD_LOGIC;
        cin  : in STD_LOGIC;
        f    : out STD_LOGIC;
        cout : out STD_LOGIC);
end sumador_completo;

architecture Behavioral of sumador_completo is

begin

    f <= a xor b xor cin;
    cout <= (a and b) or (cin and (a xor b));

end Behavioral;
```

4) Genere un símbolo esquemático a partir del archivo vhd desarrollado en el punto anterior.

AYUDA:

En el entorno Quartus:

En la ventana: Project Navigator, solapa: Files, click derecho sobre el archivo➔Create Symbol Files for Current File.



5) Agregue el componente simbólico al archivo esquemático.

6) En el archivo esquemático agregue los símbolos de los componentes necesarios (Symbol Tool), necesarias para conseguir el funcionamiento de un sumador/restador y los bits de CCR. Agregue FFD en las entradas y salidas del circuito.

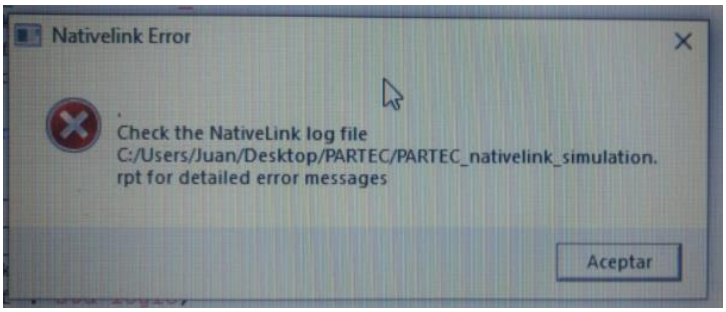
AYUDA:

Puede usar el archivo VHDL DFF desarrollado en el punto anterior, incluirlo en el proyecto, generar un símbolo esquemático a partir de este arhicvo VHDL y agregarlo al circuito esquemático.

Otra opción es usar el ffd que provee el Quartus en Simbol tool=> primitives=> storage=> dff

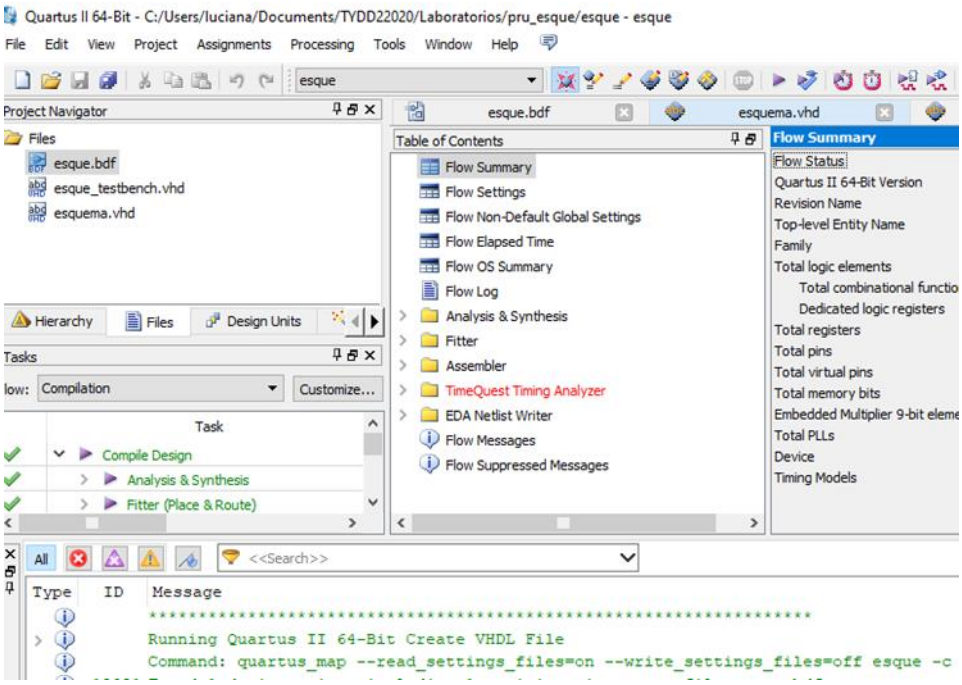
7) Realice las conexiones genere puertos de entrada/salida y compile.
8) Asigne los pines de entrada y salida del diseño. Compile nuevamente.
9) Verifique el circuito implementado mediante el visor de RTL.
10) Verifique el circuito implementado luego del mapeo en el dispositivo.
11) Verifique la implementación en el chip.
12) Verifique el funcionamiento funcional y temporal del circuito mediante la simulación de Quartus.
13) Verifique el funcionamiento del circuito mediante la simulación de Modelsim.

En este caso no se logra realizar la simulación con el entorno de simulación de Modelsim. El archivo testbench no logra instanciar el archivo esquemático creado. Aparece el siguiente error:

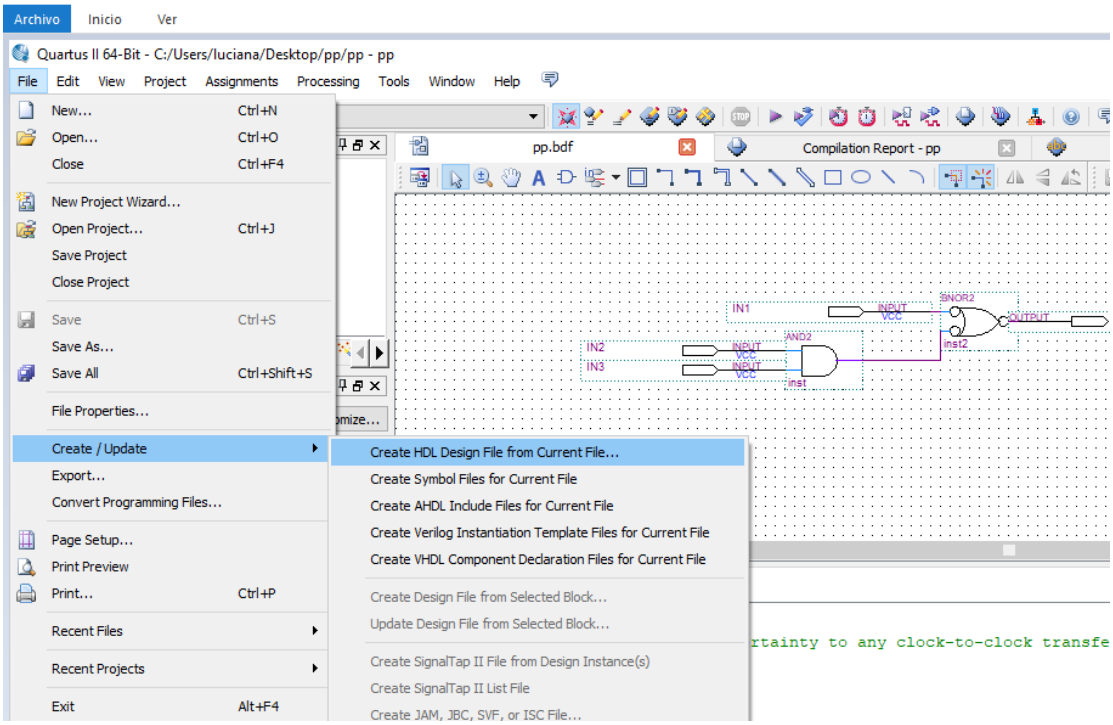


Para poder simular un archivo esquemático primero hay que pasar el esquemático a VHDL, porque Modelsim no soporta esquemático

Para solucionarlo: para convertir en VHDL hay que pararse en el archivo esquemático.



y elegir File=>create/update=>create HDL file from current file



Esto crea un archivo .vhdl en la carpeta donde se esta trabajando con el mismo nombre del esquemático. Sacar el esquemático, agregar el archivo vhdl, compilar y ahi es posible simular con Modelsim.

PARTE D: Implementación de una máquina de estado.

Implemente las máquinas de estados de los ejercicios 2 (secuencia de luces) de la Guía 6.

Implemente mediante la herramienta State Tool y mediante la inserción de un Template de máquina de estado:

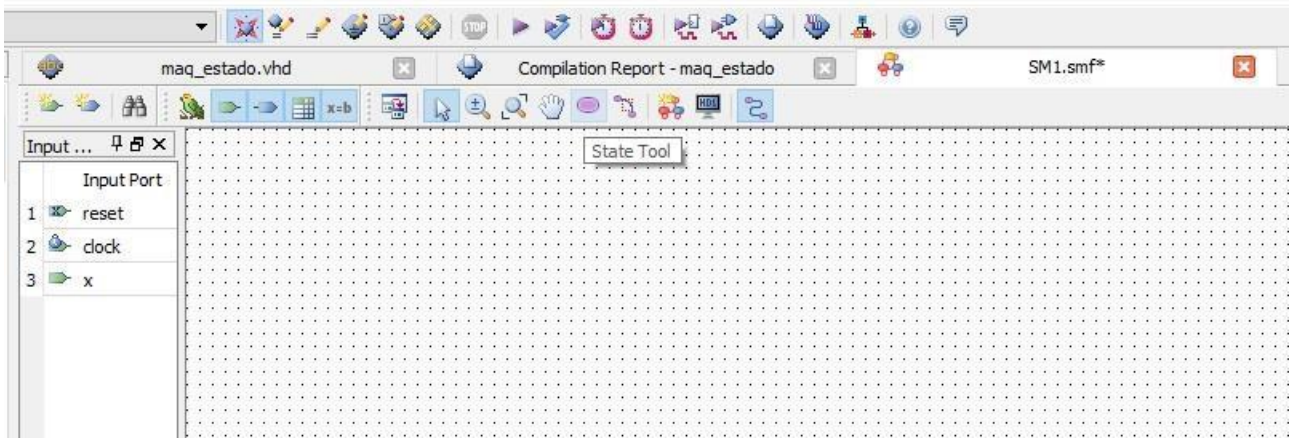
Implementación mediante State Tool:

Realice los siguientes pasos:

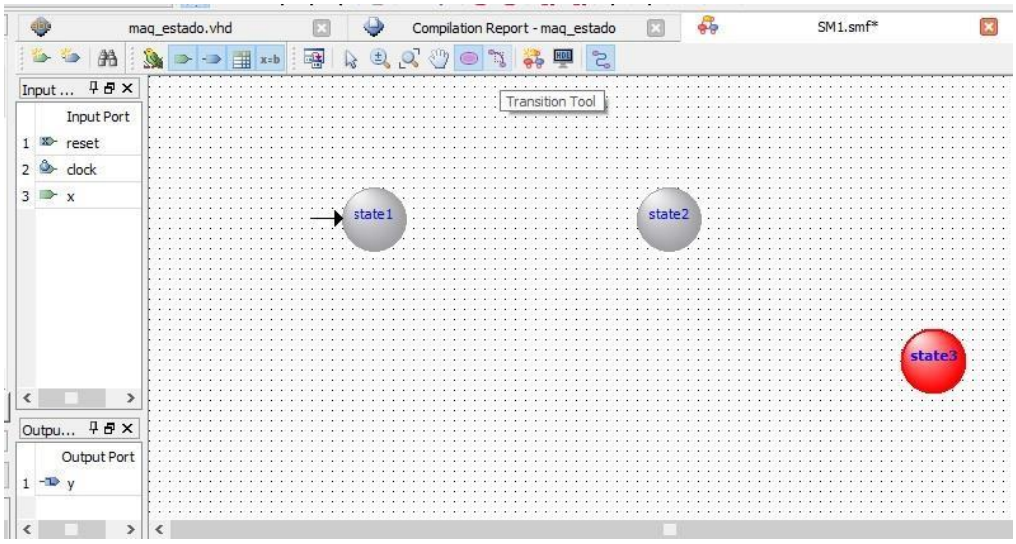
1) Utilice el entorno Quartus II para crear un nuevo proyecto.

2) En el proyecto genere un archivo *State machine file*.

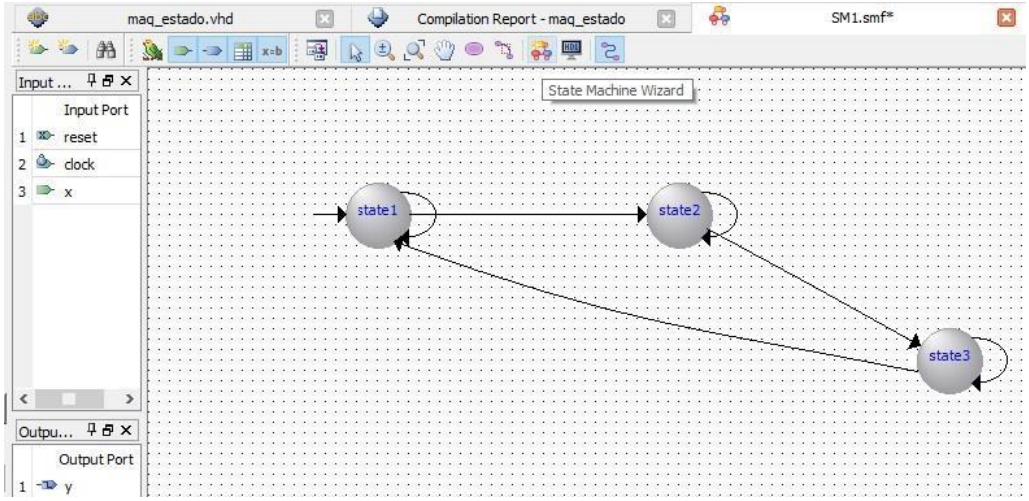
3) Con la herramienta *State tool* ingrese los estados.



4) Con la herramienta *Transition tool* ingrese las transiciones.

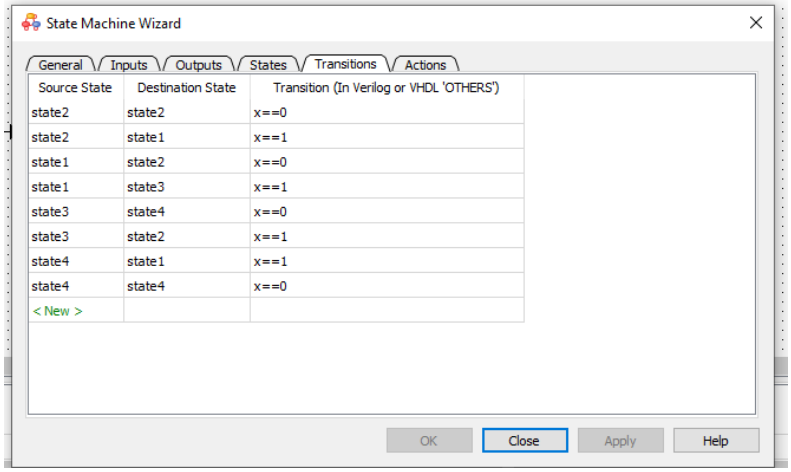


5) Con la herramienta *State machine wizard* seleccione Edit existing state machine.

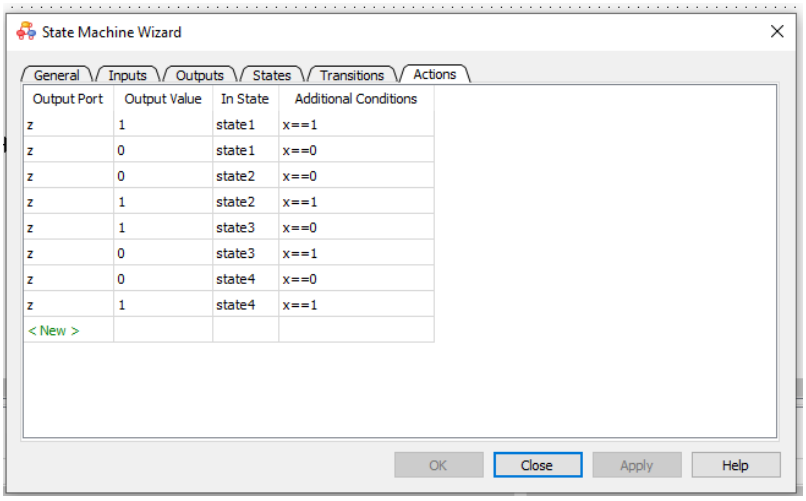


6) Con la herramienta State machine wizard seleccione Edit:

Ingrese las entradas, salidas, las condiciones de transición:

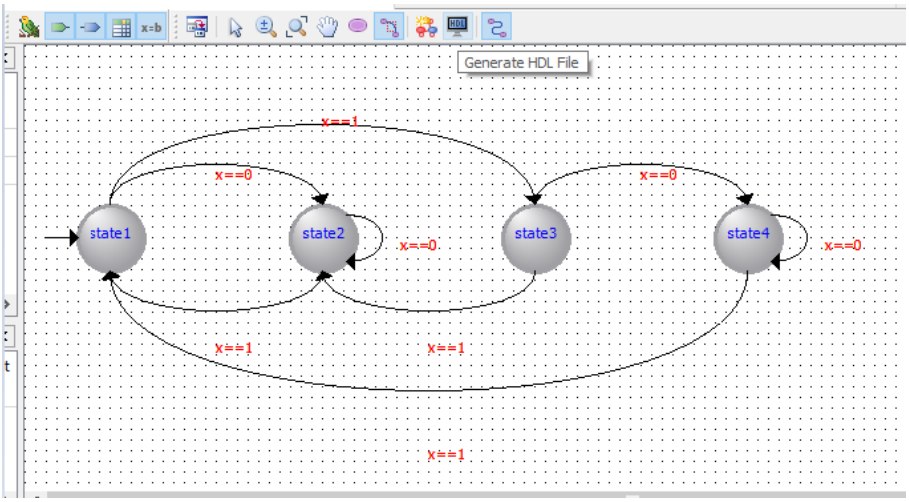


Y en la solapa *Actions* los valores de salida.

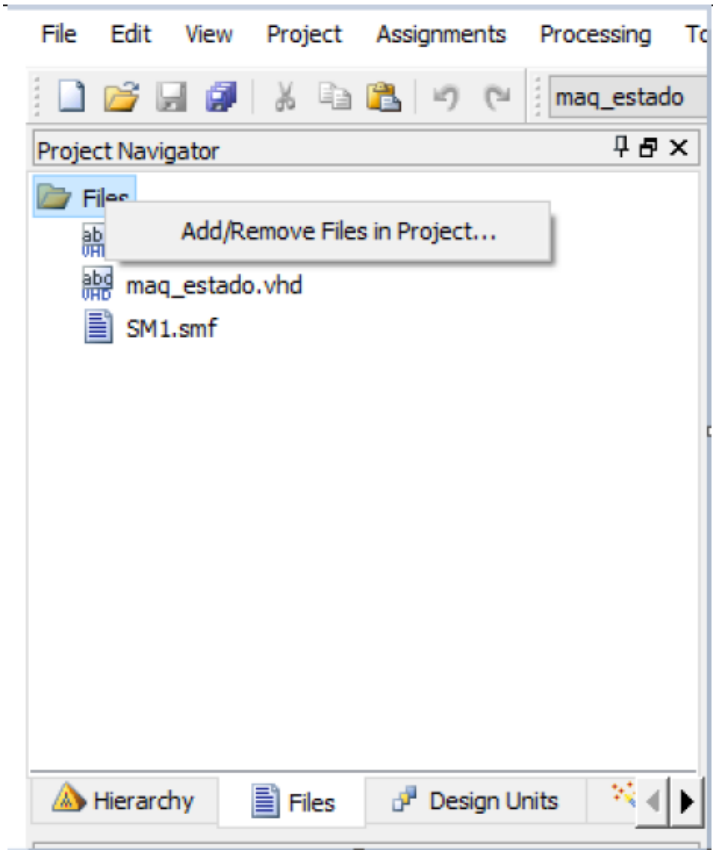


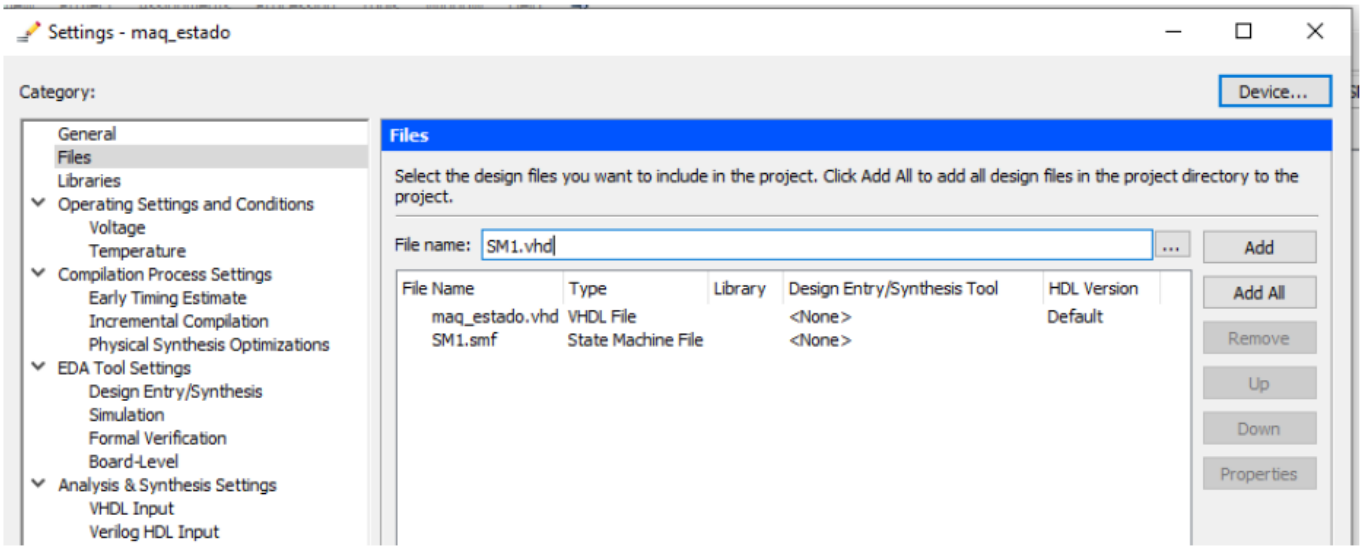
Note que si la maquina es Moore la columna “Additional Conditions” quedara en blanco, si es Mealy se deberá completar con la condición de la entrada.

8) Genere el código VHDL:



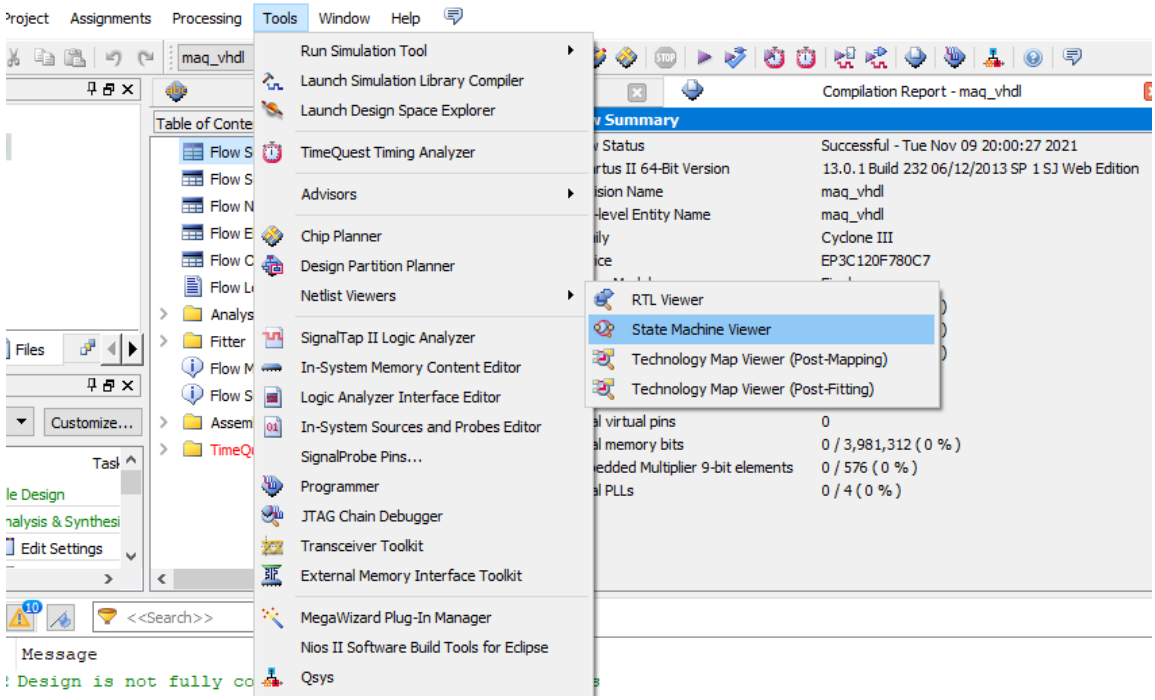
8) Guarde el archivo vhdl generado y agréguelo al proyecto.



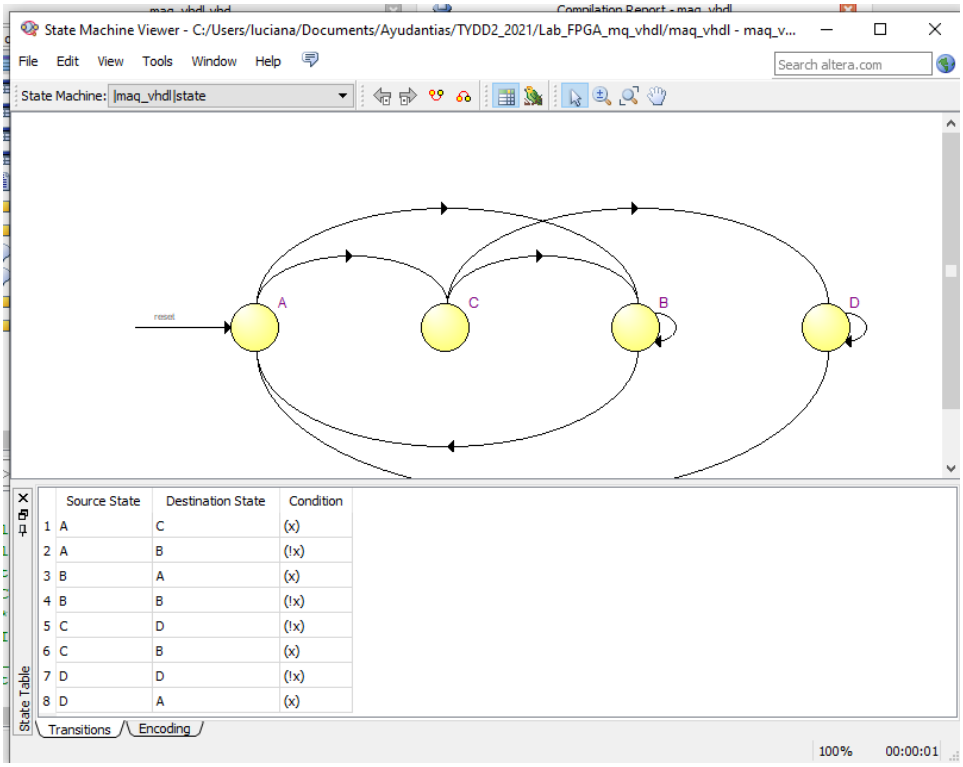


9) Compile.

10) Vea la máquina implementada seleccionando del Menu Tools=>Netlist Viewers=>State Machine Viewer



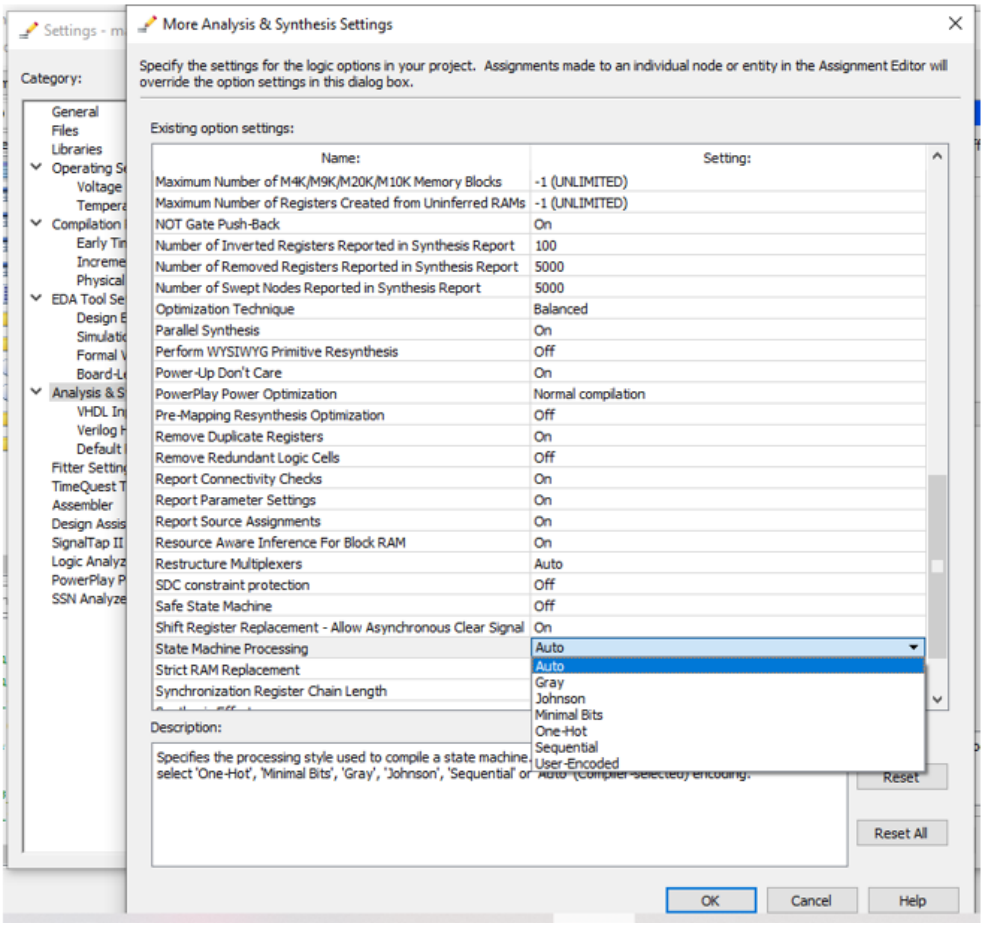
Verá algo como la siguiente figura, donde en la solapa “Encoding” podrá ver la codificación empleada.



11) Cambie el estilo de procesamiento de la máquina de estado, para que el compilador asigne otra codificción.

AYUDA:

Para cambiar la codificación usada para cada estado seleccione del menú Assignments=>Settings=> Analysis & Synthesis Settings=> More Settings=> State Machine Processings podrá elegir el estilo de procesamiento entre: auto, gray, Johnson, Minimal Bits, One-Hot, Sequential y User-Encoded



Auto	Allows the Compiler to choose the best encoding for the state machine.
Gray	Uses the minimal number of bits to encode the state machine, ceiling of 2 to the log of *n* states. This setting is difference from the **Minimal Bits** setting because each state has only one bit difference from its neighboring states.
Johnson	The number of bits used to encode the state machine is the ceilling of half of the states. Each state has only one bit difference from its neighboring states. Each state is generated by shifting the previous state's bits to the right by 1. The MSB of each state is the negation of the LSB of the previous state.
Minimal Bits	Uses the minimal number of bits to encode the state machine.
One-Hot	Encodes the state machine in the one-hot style. For one-hot encoding, the Quartus II software does not guarantee that each state has one bit set to one and all other bits to zero. Instead, the Quartus II software makes sure that the reset state is all-zeroes. All other states have two bits set to one and all others to zero. This is done so the state machine powers up in the reset state. This encoding has the same properties as true one-hot encoding: each state can be recognized by the value of one bit.
Sequential	Uses the minimal number of bits to encode the state machine; each state is the binary form of its state value.
User-Encoded	Encodes the state machine in the manner specified by the user.

12) Compile nuevamente y vea el circuito implementado.

13) Realice las simulaciones para verificar el correcto funcionamiento.

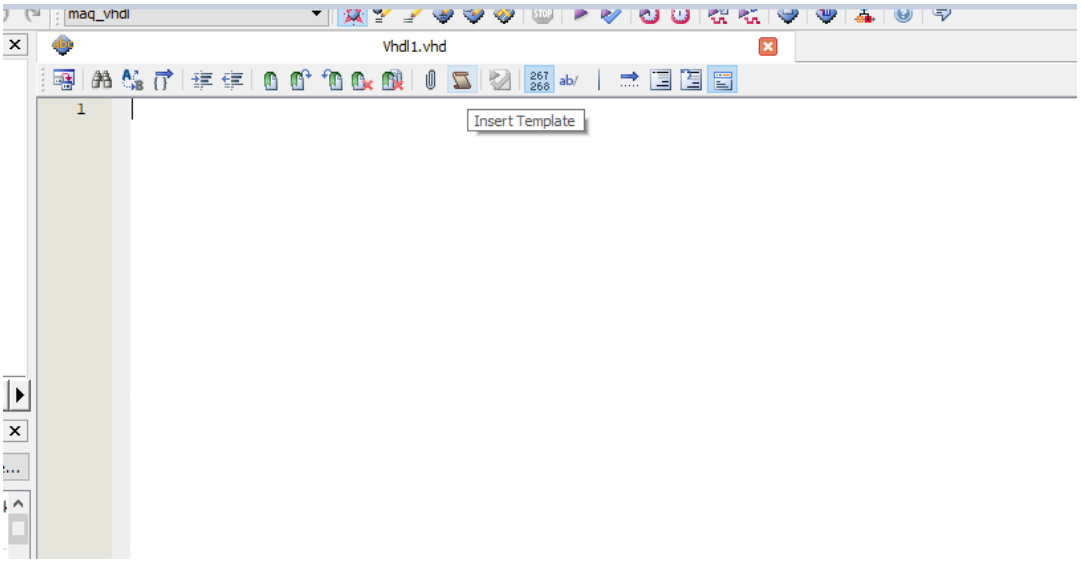
Implementación mediante Template de máquina de estado:

Realice los siguientes pasos:

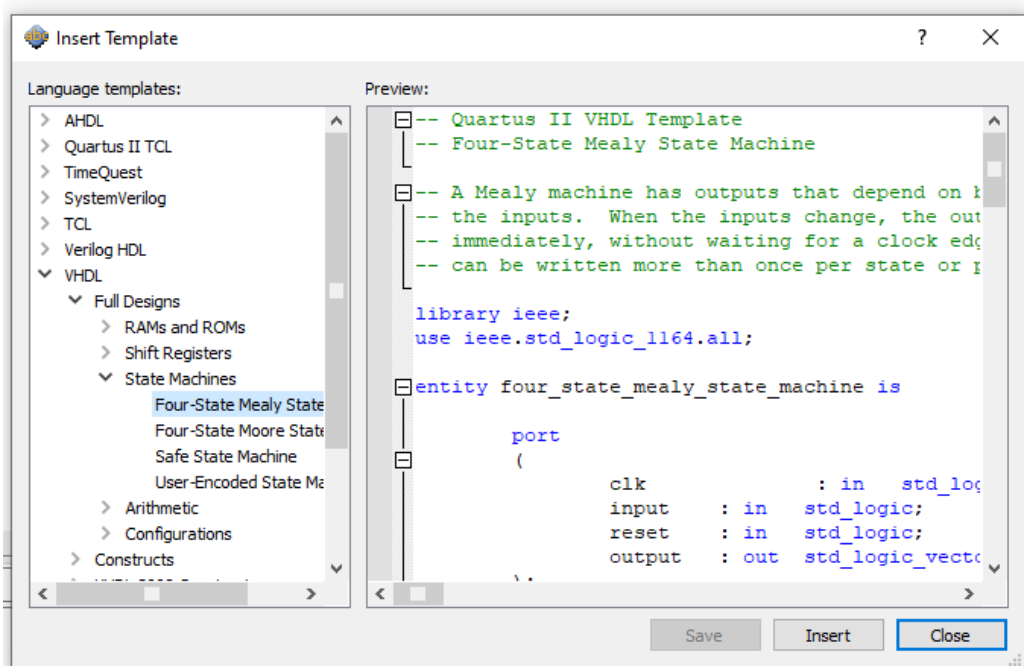
1) Utilice el entorno Quartus II para crear un nuevo proyecto.

2) En el proyecto genere un archivo *VHDL file*.

3) Seleccione Insert Template



4) 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.



5) Compile

6) Vea la máquina implementada seleccionando del Menu Tools=>Netlist Viewers=>State Machine Viewer

Verifique en la solapa “Encoding” la codificación empleada.

7) Cambie la codificación usada para cada estado.

AYUDA:

Seleccione del menú Assignments=>Settings=> Analysis & Synthesis Settings=> More Settings=> State Machine Processings podrá elegir el estilo de procesamiento entre: auto, gray, Johnson, Minimal Bits, One-Hot, Sequential y User-Encoded

8) Compile nuevamente y vea el circuito implementado.

9) Simule para verificar el correcto funcionamiento.