CENTRO UNIVERSITARIO DE AUTOMACIÓN Y ROBÓTICA

Facultad Regional Córdoba

TECNICAS DIGITALES III

Agenda

-Process.
-Maquina de estados.
-Depuración de programas en VHDL utilizando archivos de disco.

process

```
-- Usage of Asynchronous resets may negatively impact FPGA resources
-- and timing. In general faster and smaller FPGA designs will
-- result from not using Asynchronous Resets. Please refer to
-- the Synthesis and Simulation Design Guide for more information.
process (<clock>,<async_reset>)
begin
   if <async_reset> = '1' then
      <statements>;
   elsif (<clock>'event and <clock> = '1') then
      if <sync_reset> = '1' then
         <statements>;
      else
         <statements>;
      end if;
   end if;
end process;
```

process

```
-- Usage of Asynchronous resets may negatively impact FPGA resources
-- and timing. In general faster and smaller FPGA designs will
-- result from not using Asynchronous Resets. Please refer to
-- the Synthesis and Simulation Design Guide for more information.
process (<clock>,<async_reset>)
begin
   if <async_reset> = '0' then
      <statements>;
   elsif (<clock>'event and <clock> = '1') then
      if <sync_reset> = '0' then
         <statements>;
      else
         <statements>;
      end if;
   end if;
end process;
```

process

```
-- Usage of Asynchronous resets may negatively impact FPGA resources
-- and timing. In general faster and smaller FPGA designs will
-- result from not using Asynchronous Resets. Please refer to
-- the Synthesis and Simulation Design Guide for more information.
process (<clock>,<reset>)
begin
   if <reset> = '1' then
      <statements>;
   elsif (<clock>'event and <clock> = '1') then
      <statements>;
   end if;
end process;
```

process

```
-- Usage of Asynchronous resets may negatively impact FPGA resources
-- and timing. In general faster and smaller FPGA designs will
-- result from not using Asynchronous Resets. Please refer to
-- the Synthesis and Simulation Design Guide for more information.
process (<clock>,<reset>)
begin
   if <reset> = '1' then
      <statements>;
   elsif (<clock>'event and <clock> = '1') then
      if <clock_enable> = '1' then
         <statements>;
      end if;
   end if;
end process;
```

If..then

```
if <condition> then
    <statement>
elsif <condition> then
    <statement>
else
    <statement>
end if;
```

case…when

```
case (<3-bit select>) is
    when "000" =>
        <statement>;
    when "001" =>
        <statement>;
    when "010" =>
        <statement>;
    when "011" =>
        <statement>;
    when "100" =>
        <statement>;
    when "101" =>
        <statement>;
    when "110" =>
        <statement>;
    when "111" =>
        <statement>;
    when others =>
        <statement>;
end case;
```

case…when

```
case (<3-bit select>) is
    when "000" =>
        <statement>;
    when "001" =>
        <statement>;
    when "010" =>
        <statement>;
    when "011" =>
        <statement>;
    when "100" =>
        <statement>;
    when "101" =>
        <statement>;
    when "110" =>
        <statement>;
    when "111" =>
        <statement>;
    when others =>
        <statement>;
end case;
```

Diagrama General

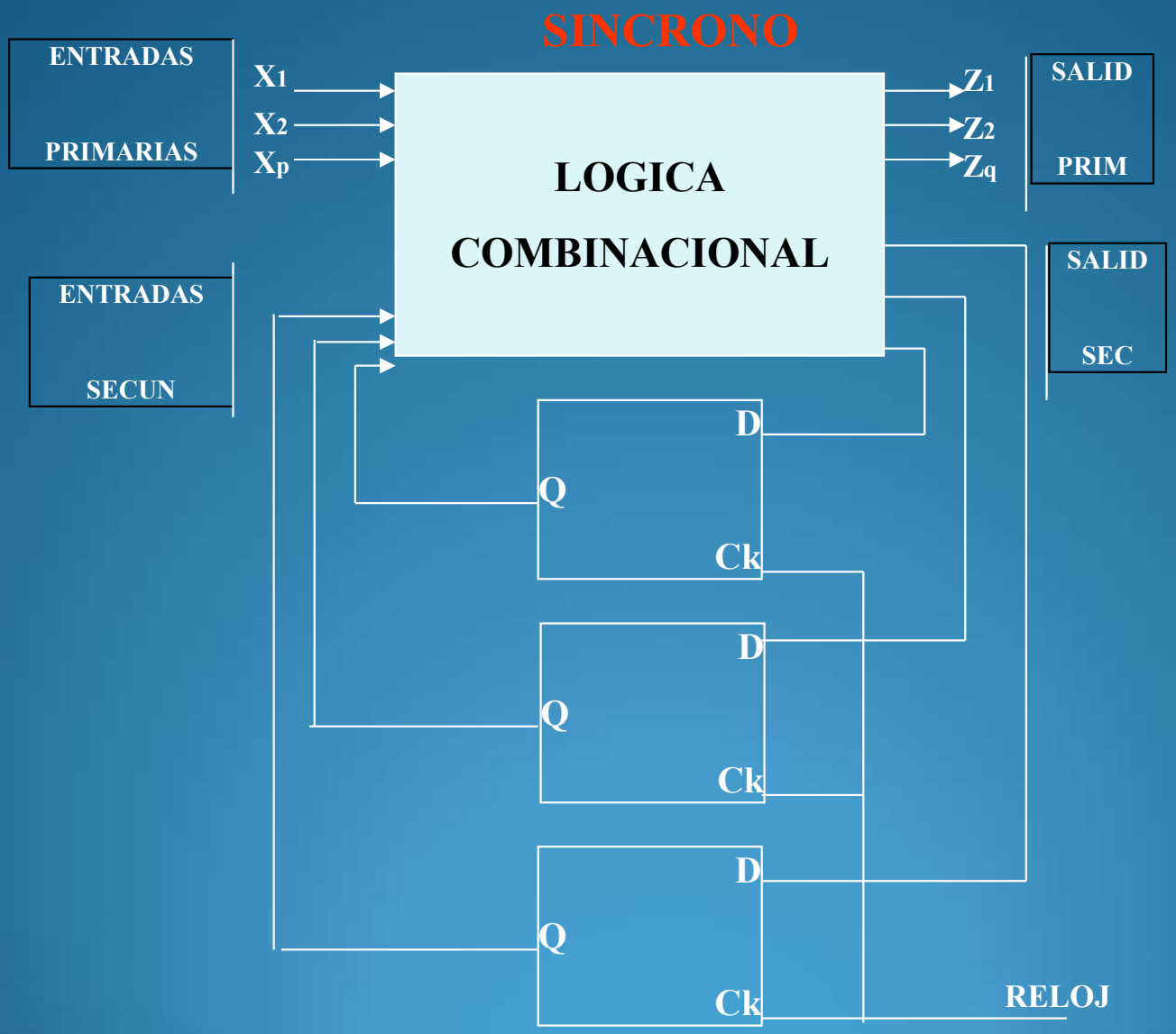
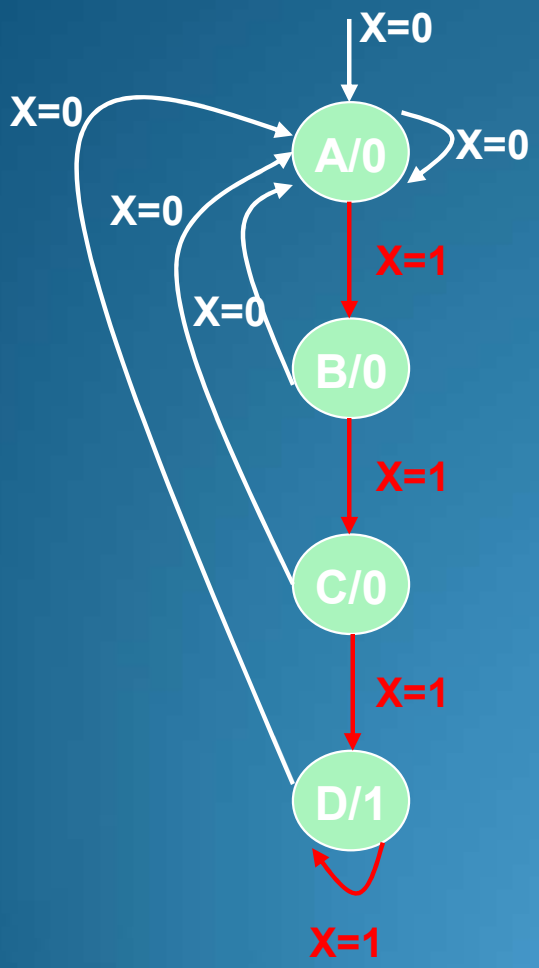
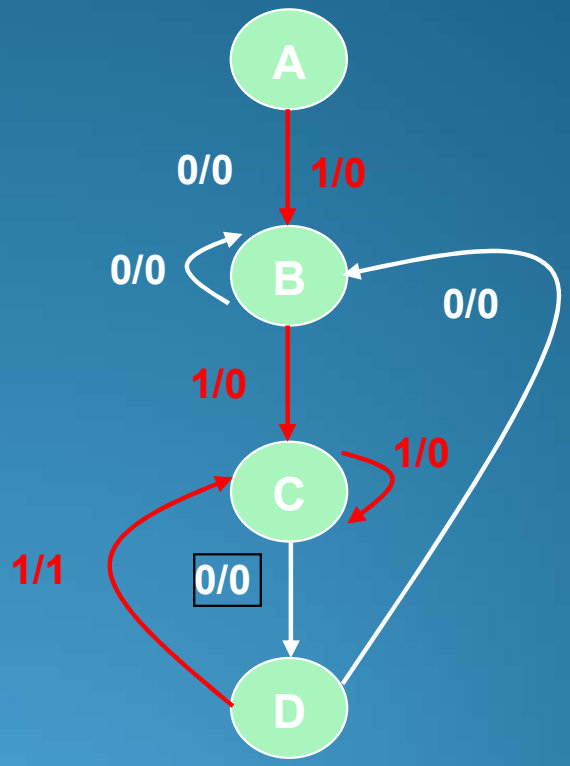
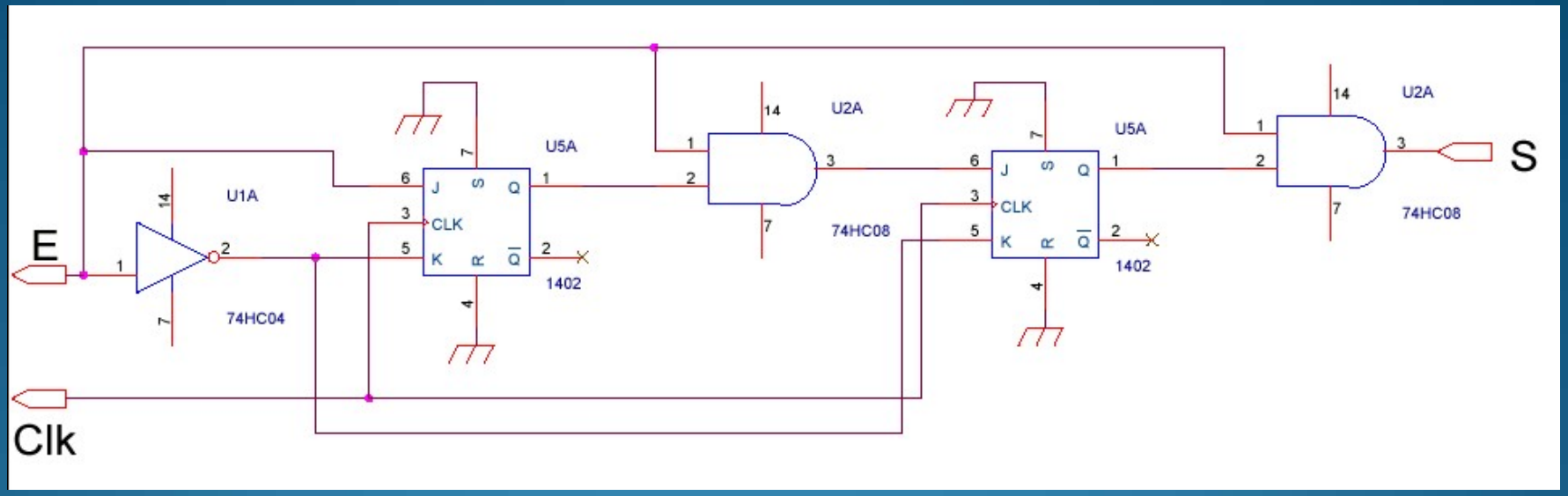


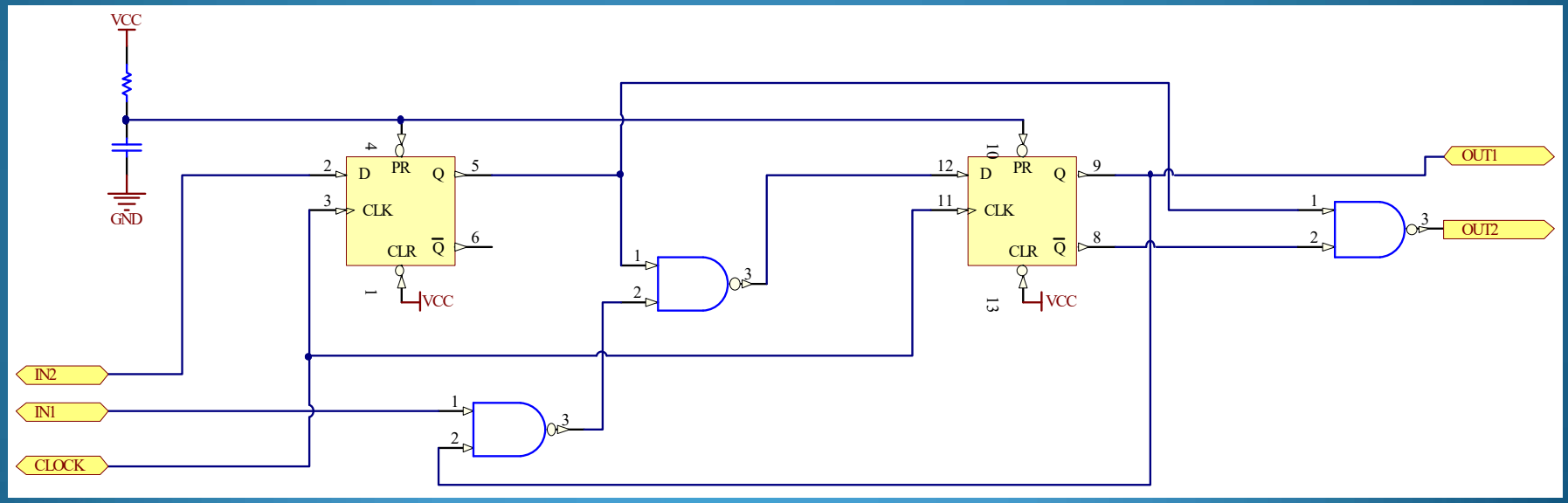
Diagrama de estados



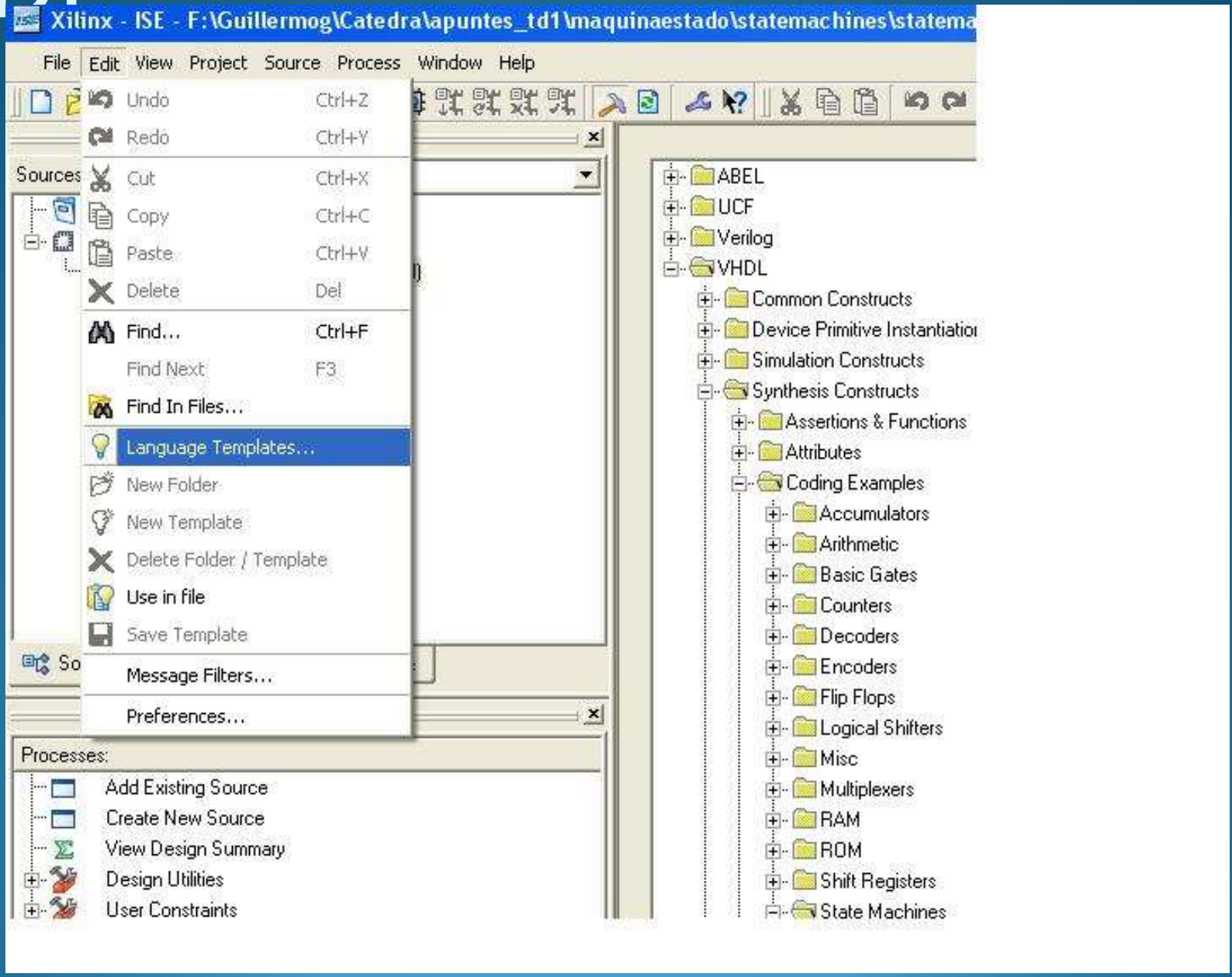


Moore y Mealy

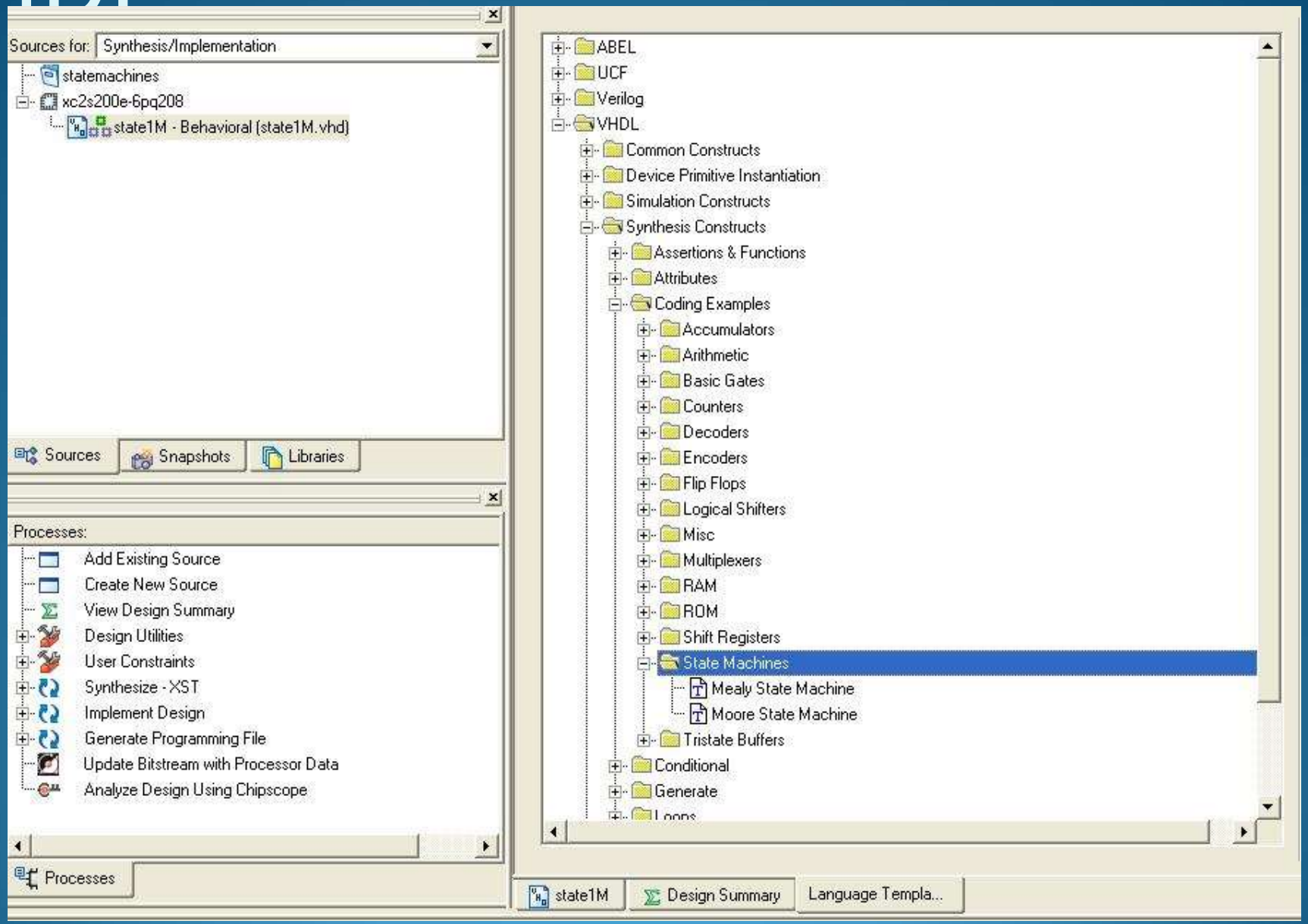




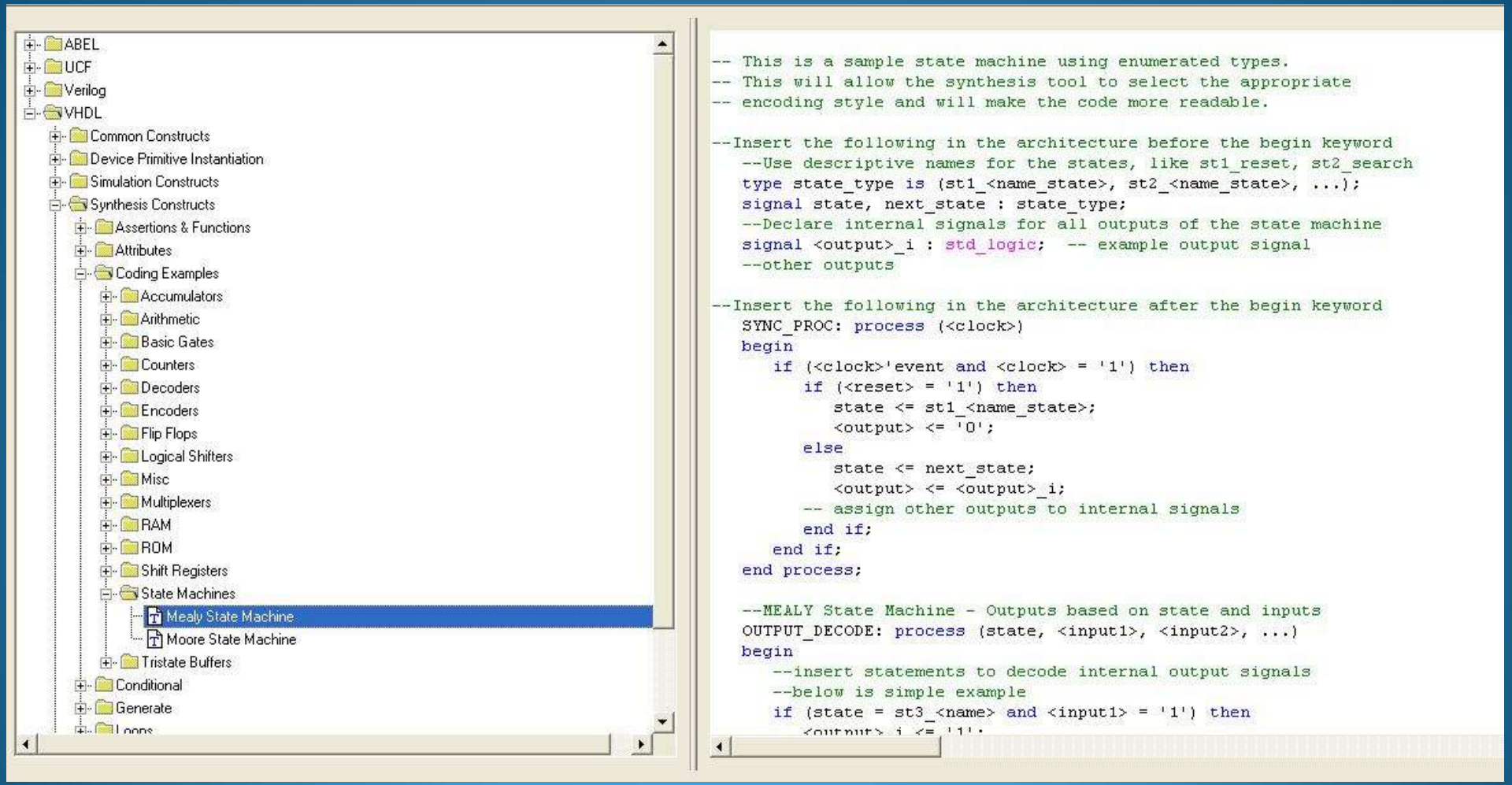
VHDL


Xilinx - ISE - F:\Guillermog\Catedra\apuntes_td1\maquinaestado\statemachines\statema
File
Edit
View
Project
Source
Process
Window
Help
Undo Ctrl+Z
Redo Ctrl+Y
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete Del
Find... Ctrl+F
Find Next F3
Find In Files...
Language Templates...
New Folder
New Template
Delete Folder / Template
Use in file
Save Template
Message Filters...
Preferences...
Sources
Processes:
Add Existing Source
Create New Source
View Design Summary
Design Utilities
User Constraints
ABEL
UCF
Verilog
VHDL
Common Constructs
Device Primitive Instantiatio
Simulation Constructs
Synthesis Constructs
Assertions & Functions
Attributes
Coding Examples
Accumulators
Arithmetic
Basic Gates
Counters
Decoders
Encoders
Flip Flops
Logical Shifters
Misc
Multiplexers
RAM
ROM
Shift Registers
State Machines

VHDL


Sources for: Synthesis/Implementation
statemachines
xc2s200e-6pq208
state1M - Behavioral (state1M.vhd)
Sources
Snapshots
Libraries
Processes:
Add Existing Source
Create New Source
View Design Summary
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Generate Programming File
Update Bitstream with Processor Data
Analyze Design Using Chipscope
Processes
ABEL
UCF
Verilog
VHDL
Common Constructs
Device Primitive Instantiation
Simulation Constructs
Synthesis Constructs
Assertions & Functions
Attributes
Coding Examples
Accumulators
Arithmetic
Basic Gates
Counters
Decoders
Encoders
Flip Flops
Logical Shifters
Misc
Multiplexers
RAM
ROM
Shift Registers
State Machines
Mealy State Machine
Moore State Machine
Tristate Buffers
Conditional
Generate
state1M
Design Summary
Language Templa...

VHDL


ABEL
UCF
Verilog
VHDL
Common Constructs
Device Primitive Instantiation
Simulation Constructs
Synthesis Constructs
Assertions & Functions
Attributes
Coding Examples
Accumulators
Arithmetic
Basic Gates
Counters
Decoders
Encoders
Flip Flops
Logical Shifters
Misc
Multiplexers
RAM
ROM
Shift Registers
State Machines
Mealy State Machine
Moore State Machine
Tristate Buffers
Conditional
Generate
-- This is a sample state machine using enumerated types.
-- This will allow the synthesis tool to select the appropriate
-- encoding style and will make the code more readable.
--Insert the following in the architecture before the begin keyword
--Use descriptive names for the states, like st1_reset, st2_search
type state_type is (st1_<name_state>, st2_<name_state>, ...);
signal state, next_state : state_type;
--Declare internal signals for all outputs of the state machine
signal <output>_i : std_logic; -- example output signal
--other outputs
--Insert the following in the architecture after the begin keyword
SYNC_PROC: process (<clock>)
begin
if (<clock>'event and <clock> = '1') then
if (<reset> = '1') then
state <= st1_<name_state>;
<output> <= '0';
else
state <= next_state;
<output> <= <output>_i;
-- assign other outputs to internal signals
end if;
end if;
end process;
--MEALY State Machine - Outputs based on state and inputs
OUTPUT_DECODE: process (state, <input1>, <input2>, ...)
begin
--insert statements to decode internal output signals
--below is simple example
if (state = st3_<name> and <input1> = '1') then

Declaraciones

```
--Insert the following in the architecture before the begin keyword
   --Use descriptive names for the states, like st1_reset, st2_search
   type state_type is (st1_<name_state>, st2_<name_state>, ...);
   signal state, next_state : state_type;
   --Declare internal signals for all outputs of the state machine
   signal <output>_i : std_logic;  -- example output signal
   --other outputs
```

Transición de estado

```
--Insert the following in the architecture after the begin keyword
   SYNC_PROC: process (<clock>)
   begin
      if (<clock>'event and <clock> = '1') then
         if (<reset> = '1') then
            state <= st1_<name_state>;
            <output> <= '0';
         else
            state <= next_state;
            <output> <= <output>_i;
         -- assign other outputs to internal signals
         end if;
      end if;
   end process;
```

Lógica de salida

```
--MEALY State Machine - Outputs based on state and inputs
OUTPUT_DECODE: process (state, <input1>, <input2>, ...)
begin
   --insert statements to decode internal output signals
   --below is simple example
   if (state = st3_<name> and <input1> = '1') then
      <output>_i <= '1';
   else
      <output>_i <= '0';
   end if;
end process;
```

```
--MOORE State Machine - Outputs based on state only
OUTPUT_DECODE: process (state)
begin
   --insert statements to decode internal output signals
   --below is simple example
   if state = st3_<name> then
      <output>_i <= '1';
   else
      <output>_i <= '0';
   end if;
end process;
```

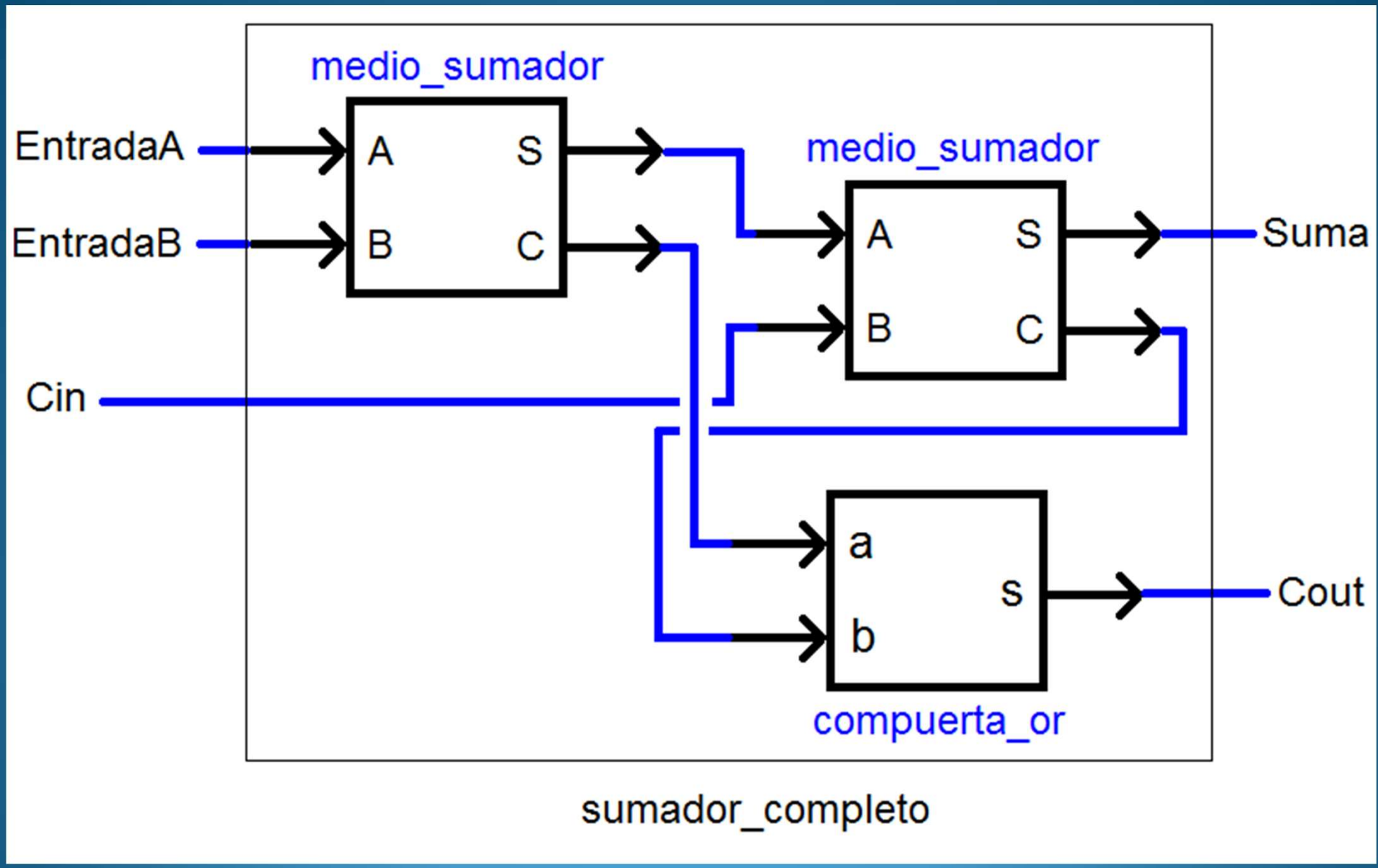
Lógica de transición

```
NEXT_STATE_DECODE: process (state, <input1>, <input2>, ...)
begin
   --declare default state for next_state to avoid latches
   next_state <= state;  --default is to stay in current state
   --insert statements to decode next_state
   --below is a simple example
   case (state) is
      when st1_<name> =>
         if <input_1> = '1' then
            next_state <= st2_<name>;
         end if;
      when st2_<name> =>
         if <input_2> = '1' then
            next_state <= st3_<name>;
         end if;
      when st3_<name> =>
         next_state <= st1_<name>;
      when others =>
         next_state <= st1_<name>;
   end case;
end process;
```

1. Una puerta se abre al activar un pulsador P. La apertura se produce hasta que alcanza el tope de apertura detectado por el sensor A. A partir de ese momento se produce el cierre de la puerta, hasta alcanzar el topo de cierre detectado por el sensor C, y en ese momento se produce la parada.

   La puerta se controla por dos salidas, S1 Y S0 cuando S1=1 la puerta se abre, cuando S0=1 la puerta se cierra. Diseñar el autómata, dibujar el circuito, simularlo en CedarLogic y describirlo en Verilog con verificación.

Instanciación de componentes


medio_sumador
EntradaA
EntradaB
A
B
S
C
medio_sumador
A
B
S
C
Suma
Cin
a
b
s
Cout
compuerta_or
sumador_completo

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use ieee.std_logic_unsigned.all;

--! Simulacion del la entidad SPIuCFPGA.
ENTITY Test_SPIuCFPGA IS
END Test_SPIuCFPGA;

ARCHITECTURE behavior OF Test_SPIuCFPGA IS

    --! Component Declaration for the Unit Under Test (UUT)
    COMPONENT SPIucFPGA
    Port ( HGRANT   : in  STD_LOGIC;        --! Concesion del uso del bus (aRBITRO->MAESTRO)
           HREADY   : in  STD_LOGIC;        --! Respuesta del procesamiento de datos (ESCLAVO->MAESTRO)
              HRESET    : in  STD_LOGIC;        --! Reinicio del sistema (BUS->MAESTRO)
           HCLK         : in  STD_LOGIC;        --! Reloj del sistema con fase 0 (BUS->MAESTRO)
           HRDATA   : in  STD_LOGIC_VECTOR (7 downto 0);    --! Bus de lectura de datos (ESCLAVO->MAESTRO)

           HBUSREQ  : out  STD_LOGIC;       --! Peticion del bus (MAESTRO->aRBITRO)
              HLOCK         : out  STD_LOGIC;       --! Peticion privilegiada del bus (MAESTRO->aRBITRO)
              HWRITE    : out  STD_LOGIC;       --! Tipo de transferencia (1: Escritura | 0: Lectura) (MAESTRO->ESCLAVO)
           HTRANS   : out  STD_LOGIC_VECTOR (1 downto 0); --! Tipo de transferencia (MAESTRO->ESCLAVO,aRBITRO)
           HADDR        : out  STD_LOGIC_VECTOR (23 downto 0); --! Bus de salida de direcciones (MAESTRO->BUS)
           HWDATA   : out  STD_LOGIC_VECTOR (7 downto 0); --! Bus de salida de datos de escritura (MAESTRO->ESCLAVO)
           HSIZE        : out  STD_LOGIC_VECTOR (2 downto 0); --! Tamano del paquete a transferir (MAESTRO->ESCLAVO)
           HBURST   : out  STD_LOGIC_VECTOR (2 downto 0); --! Longitud de transferencia (MAESTRO->ESCLAVO)

              MOSI      : in STD_LOGIC; --! MOSI: Master Output Slave Input.
             SSEL       : in STD_LOGIC; --! SSEL: Slave SELect.
             SCLK       : in STD_LOGIC; --! SCLK: SPI CLocK.
              MISO      : out STD_LOGIC);   --! MISO: Master Input Slave Output
    END COMPONENT;
```

```
COMPONENT Tester_SPI_WR2
    Port ( Datos        : in   STD_LOGIC_VECTOR (7  downto 0);  --! Datos que provienen del archivo y se enviaran por MOSI.
           Direccion: in   STD_LOGIC_VECTOR (21 downto 0);  --! Direccion que proviene de archivo y se enviaran por MOSI.
              NumDatos  : in   STD_LOGIC_VECTOR (7  downto 0);  --! Cantidad de datos que se enviaran por MOSI.
           Modo         : in   STD_LOGIC;   --! Indica si es una transmision long o short. [1: Short | 0: Long].
           Transmit     : in   STD_LOGIC;   --! Indica que se tiene que realizar una transmision.
           Reset        : in   STD_LOGIC;   --! Reinicia el esclavo SPI manteniento SSEL en alto por un tiempo TBD.
           Reloj        : in   STD_LOGIC;   --! Reloj del sistema que ademas se utilizara para la senal SCKL.
           Ready        : out  STD_LOGIC;   --! Indica que se transmitieron todos los datos.
              MOSI      : out  STD_LOGIC;   --! Senal MOSI de transmision SPI.
           SSEL         : out  STD_LOGIC;   --! Senal SSEL de transmision SPI.
           SCLK         : out  STD_LOGIC;   --! Senal SCLK de transmision SPI.
              NAddr         : out  STD_LOGIC;   --! Senal que le indica al manejador de archivos que necesita otra direccion nuevo.
           NData        : out  STD_LOGIC);--! Senal que le indica al manejador de archivos que necesita otro dato nuevo.
END COMPONENT;
```

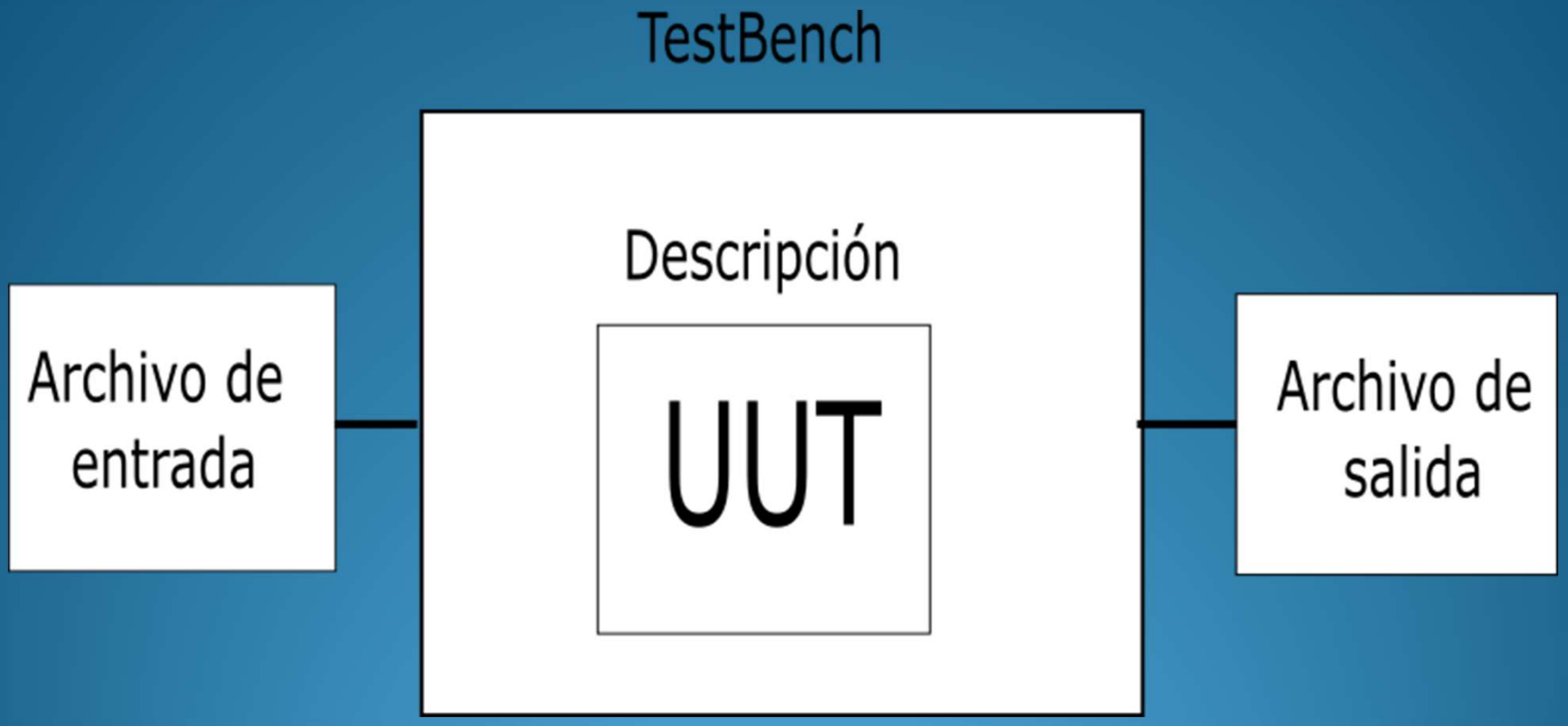
```
Inst_Tester_SPI_WR: Tester_SPI_WR2
    PORT MAP(
        Datos           => sigDatos_WR,
        Direccion   => sigDireccion_WR,
        NumDatos        => sigNumDatos,
        Modo            => sigModo,
        Transmit    => sigTransmit_WR,
        Reset       => sigReset,
        Reloj       => reloj,
        Ready       => sigReady_WR,
        MOSI            => sigMOSI_WR,
        SSEL            => sigSSEL_WR,
        SCLK            => sigSCLK_WR,
        NAddr       => sigNAddr_WR,
        NData       => sigNData_WR);
```

```
architecture Behavioral of slaveAMBAsimulacion is

component FileWRITER is
    Generic(    WriteFile: string := "Archivo.dat"; -- En el caso de que el archivo no este en la carpeta de
                InputWidth : integer := 8);

    Port    (   NewData     : in  STD_LOGIC;
                Input   : in  STD_LOGIC_VECTOR ((InputWidth-1) downto 0);
                FileON  : in  STD_LOGIC);
end component FileWRITER;

component FileREADER
    GENERIC(    ReadFile: string    := "AddressFile.dat";
                OutputWidth : integer := 8);
    PORT(   NewData : IN std_logic;
            FileON  : OUT std_logic;
            Output  : OUT std_logic_vector((OutputWidth-1) downto 0));
END component;

signal sgEscribirDato : std_logic;                  --!Senal que en alto indica que se debe escribir un dato
signal sgLeerDato : std_logic;                      --!Senal que en alto indica que se debe leer un dato
signal sgEnable : std_logic;                            --!senal de habilitacion
signal sgHWDATA : std_logiC_vector(7 downto 0); --!bus de escritura amba

begin
```

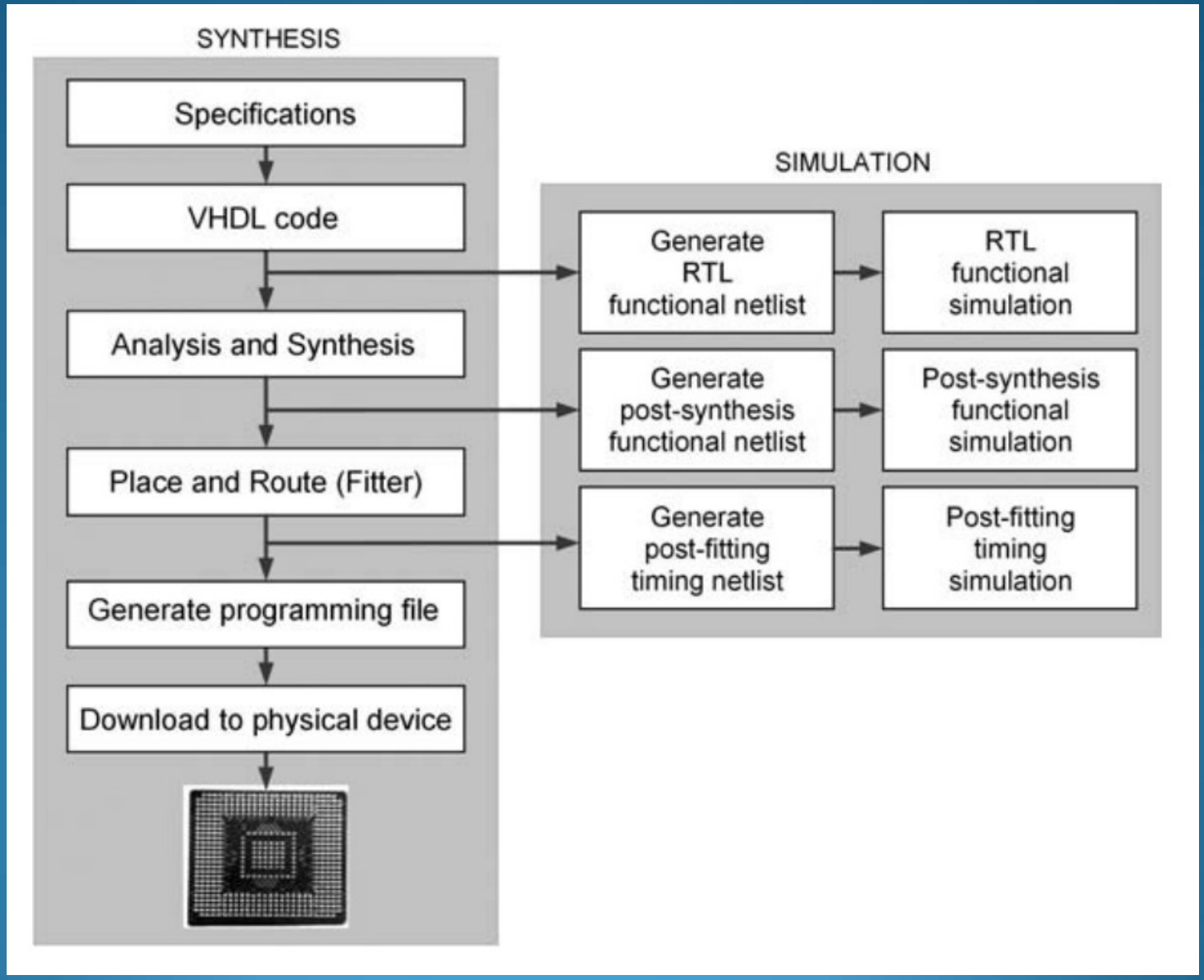
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use std.textio.all;
use work.txt_util.all;

entity FileWRITER is
    Generic(    WriteFile: string := "Archivo.dat"; -- En el caso de que el arc
                InputWidth : integer := 8);

    Port    (   NewData     : in  STD_LOGIC;
                Input   : in  STD_LOGIC_VECTOR ((InputWidth-1) downto 0);
                FileON  : in  STD_LOGIC);
end FileWRITER;
```

Testbench

Esquema general



Niveles de simulación



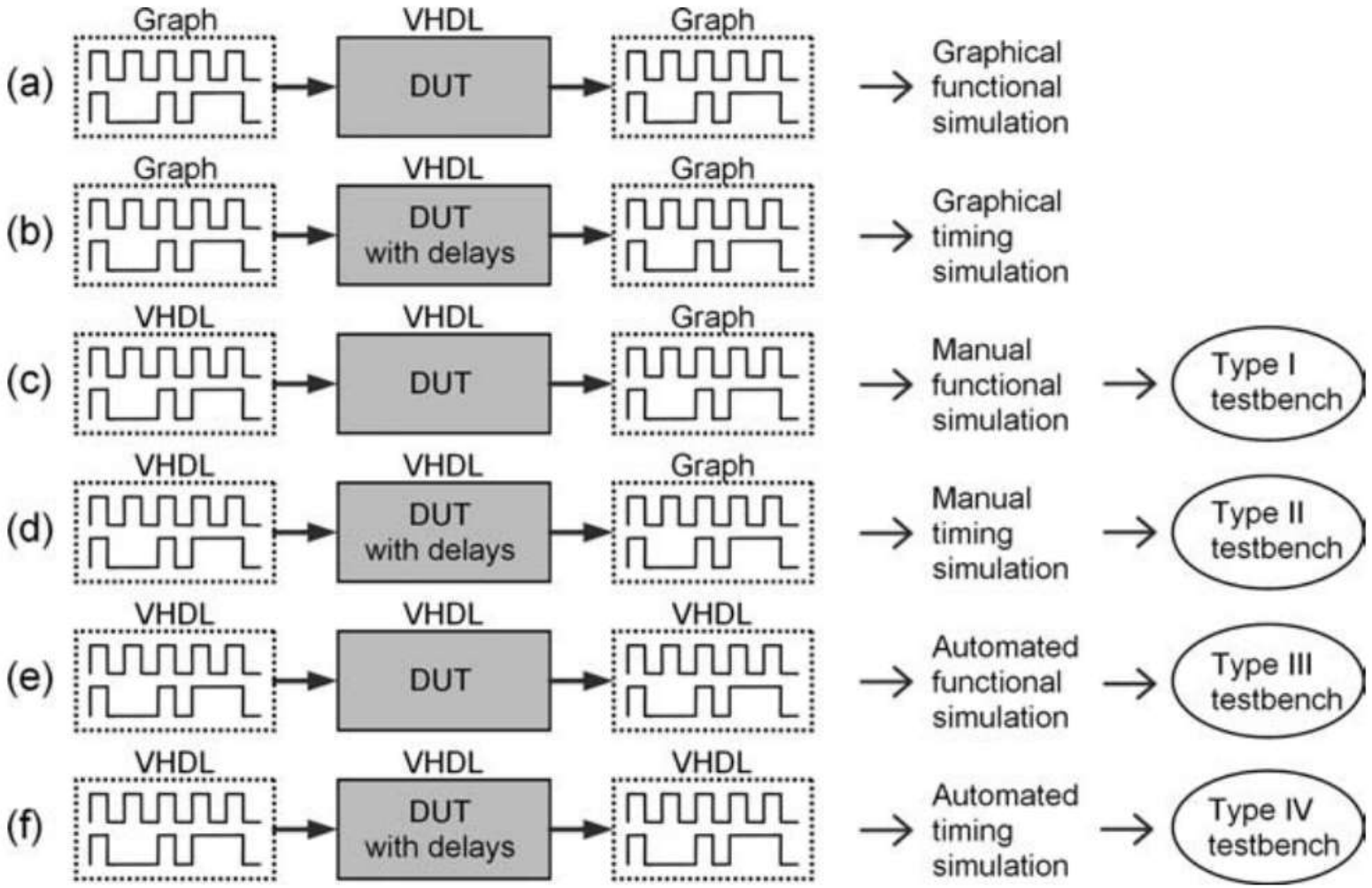
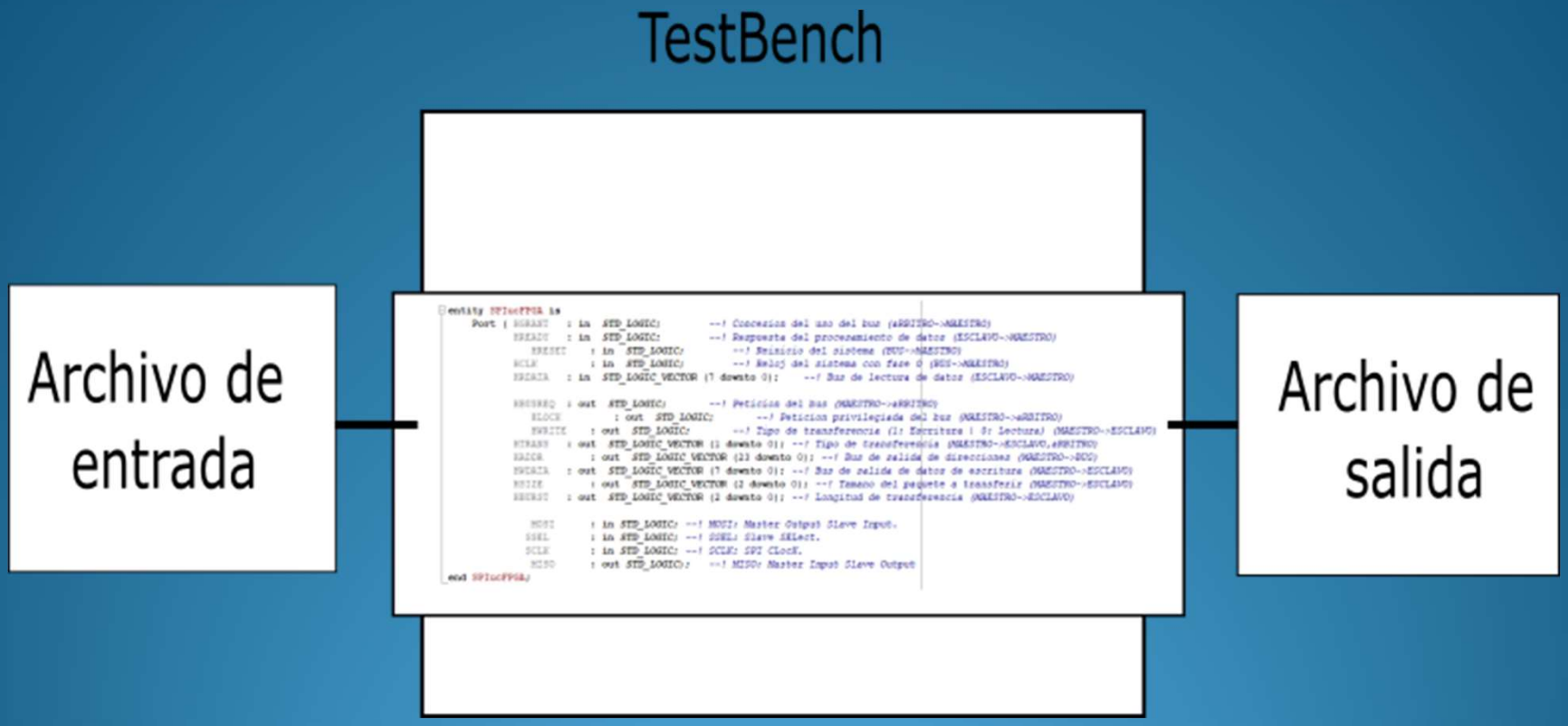


Figure 10.2
Simulation interface options.

Circuit Design and Simulation with VHDL. The MIT Press Cambridge, Massachusetts London, England

Esquema general



Esquema general

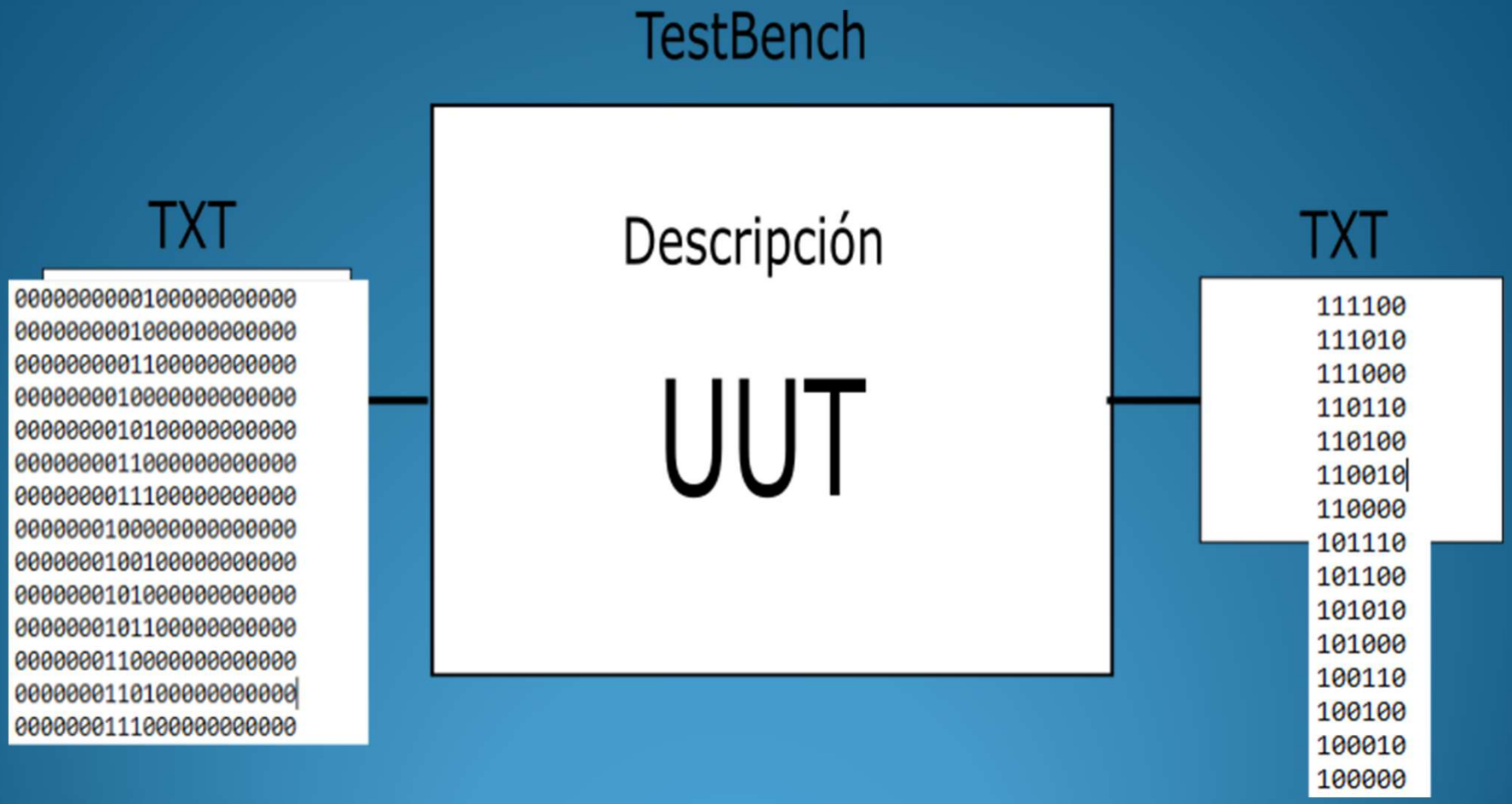
TestBench

Archivo de entrada

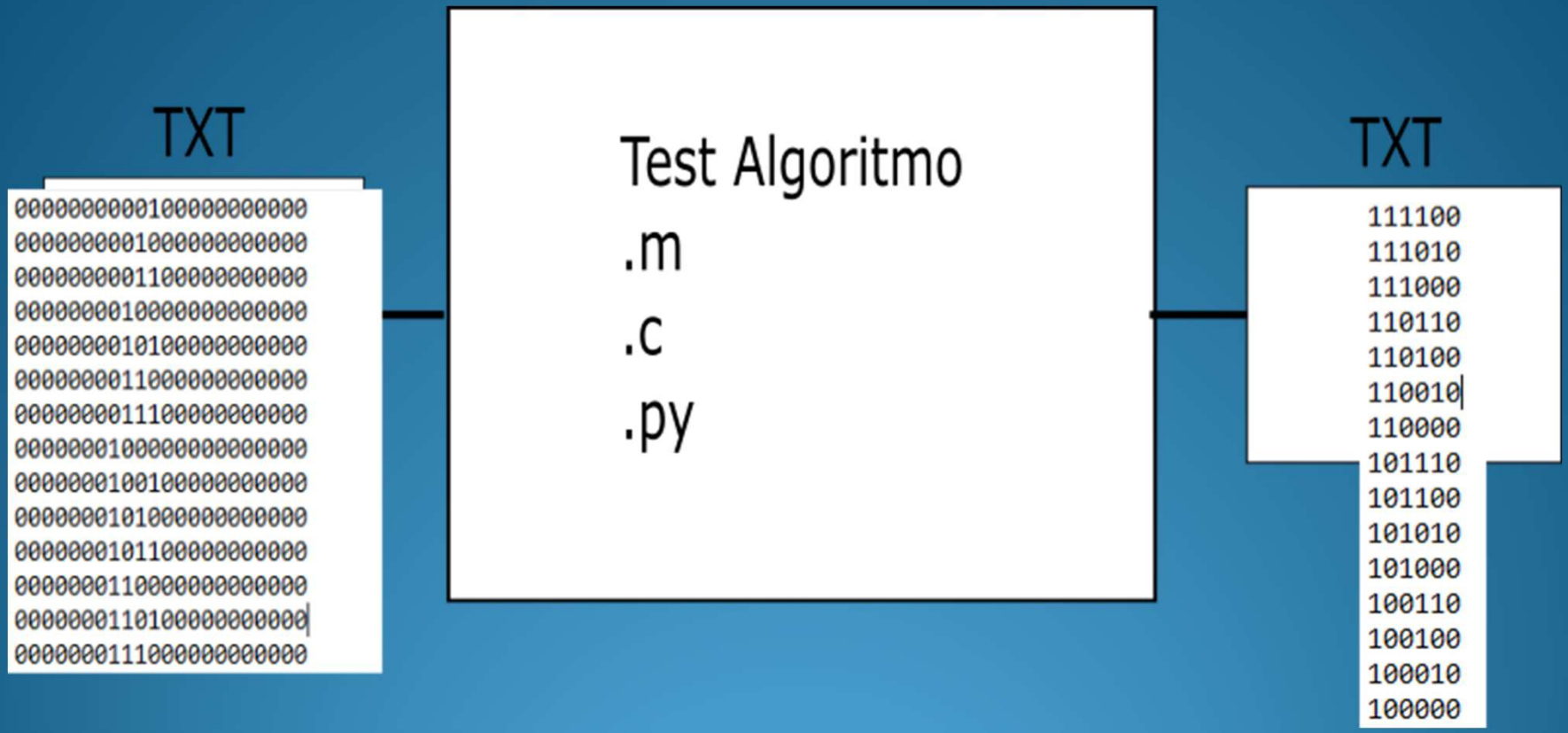
```
--! Simulacion del le entidad SPIuCFPGA.
ENTITY Test_SPIuCFPGA IS
END Test_SPIuCFPGA;

ARCHITECTURE behavior OF Test_SPIuCFPGA IS

    --! Component Declaration for the Unit Under Test (UUT)
    COMPONENT SPIucFPGA
    Port ( HGRANT   : in  STD_LOGIC;        --! Concesion del uso del bus (aRBITRO->MAESTRO)
           HREADY   : in  STD_LOGIC;        --! Respuesta del procesamiento de datos (ESCLAVO->MAESTRO)
             HRESET    : in  STD_LOGIC;        --! Reinicio del sistema (BUS->MAESTRO)
           HCLK         : in  STD_LOGIC;        --! Reloj del sistema con fase 0 (BUS->MAESTRO)
           HRDATA   : in  STD_LOGIC_VECTOR (7 downto 0);    --! Bus de lectura de datos (ESCLAVO->MAESTRO)

           HBUSREQ  : out  STD_LOGIC;       --! Peticion del bus (MAESTRO->aRBITRO)
              HLOCK         : out  STD_LOGIC;       --! Peticion privilegiada del bus (MAESTRO->aRBITRO)
              HWRITE    : out  STD_LOGIC;       --! Tipo de transferencia (1: Escritura | 0: Lectura) (MAESTRO->ESCLAVO)
           HTRANS   : out  STD_LOGIC_VECTOR (1 downto 0); --! Tipo de transferencia (MAESTRO->ESCLAVO,aRBITRO)
           HADDR        : out  STD_LOGIC_VECTOR (23 downto 0); --! Bus de salida de direcciones (MAESTRO->BUS)
           HWDATA   : out  STD_LOGIC_VECTOR (7 downto 0); --! Bus de salida de datos de escritura (MAESTRO->ESCLAVO)
           HSIZE        : out  STD_LOGIC_VECTOR (2 downto 0); --! Tamano del paquete a transferir (MAESTRO->ESCLAVO)
           HBURST   : out  STD_LOGIC_VECTOR (2 downto 0); --! Longitud de transferencia (MAESTRO->ESCLAVO)

             MOSI       : in STD_LOGIC; --! MOSI: Master Output Slave Input.
            SSEL        : in STD_LOGIC; --! SSEL: Slave SELect.
            SCLK        : in STD_LOGIC; --! SCLK: SPI CLocK.
             MISO       : out STD_LOGIC);    --! MISO: Master Input Slave Output
    END COMPONENT;
```

Archivo de salida

Esquema general



Esquema general



Lectura de estímulos

```
--! COMPONENTES PARA MANEJO DE ARCHIVOS
--! Entidad que escribe archivos.
COMPONENT FileWRITER
    Generic(    WriteFile: string := "Archivo.dat"; --! En el caso de que el archivo no este en la carpeta de trabajo: "C:\Carpeta/NombreAr
                InputWidth : integer := 8); --! Se carga el tamano del bus de entrada.

    Port    (   NewData     : in  STD_LOGIC;    --! Indica que hay un nuevo dato para guardar.
                Input   : in  STD_LOGIC_VECTOR ((InputWidth-1) downto 0);   --! Senal de entrada.
                FileON  : in  STD_LOGIC);   --! Senal que abre el archivo a escribir.
END COMPONENT;

--! Entidad que lee archivos.
COMPONENT FileREADER
    Generic(    ReadFile: string := "AddressFile.dat";  -- En el caso de que el archivo no este en la carpeta de trabajo: "C:\Carpeta/Nombr
                OutputWidth : integer := 8);    --! Se carga el tamano del bus de salida.
    Port(   NewData     : in   STD_LOGIC;   --! Indica que se quiere leer un nuevo dato desde el archivo.
            FileON  : out  STD_LOGIC;   --! Abre para leer el archivo.
            Output  : out  STD_LOGIC_VECTOR ((OutputWidth-1) downto 0));    --! Senal de salida.
END COMPONENT;
```