CUDAR
CENTRO UNIVERSITARIO DE AUTOMACIÓN Y ROBÓTICA

UTN
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

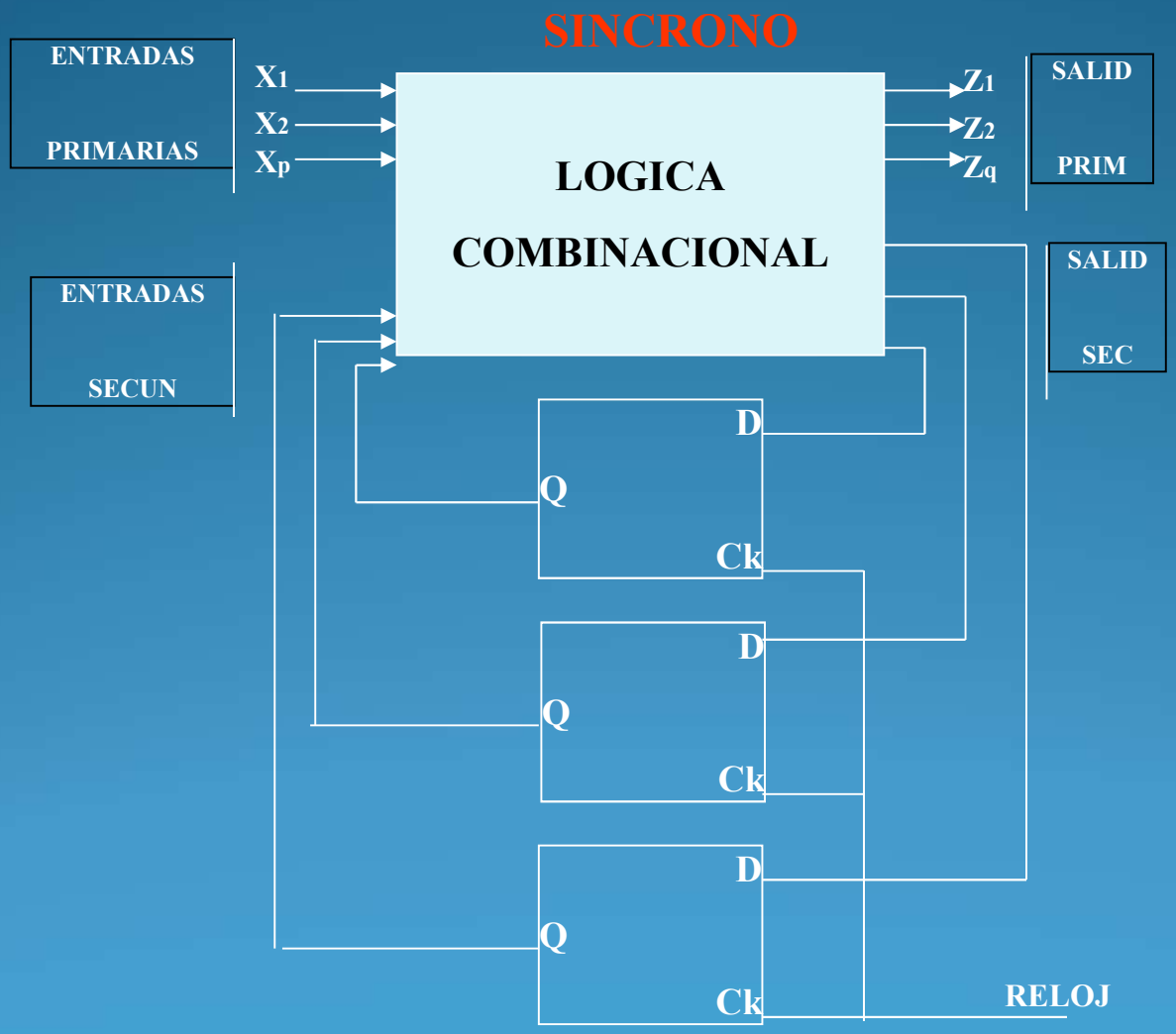
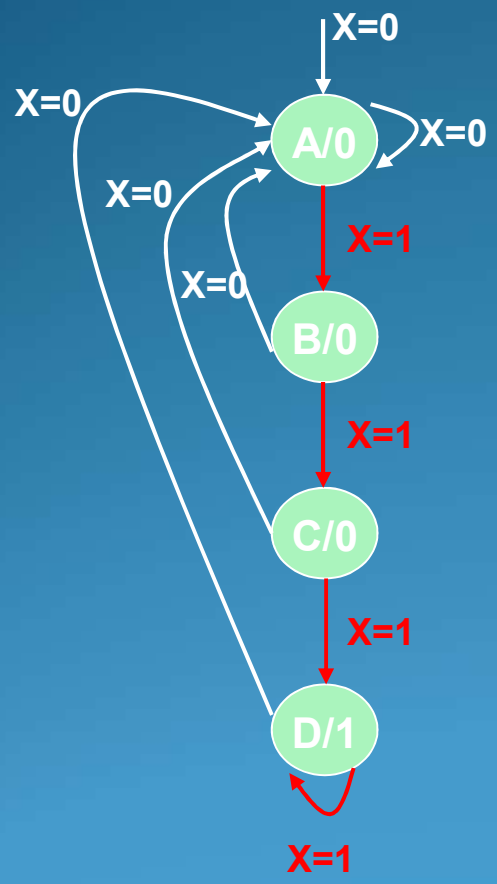
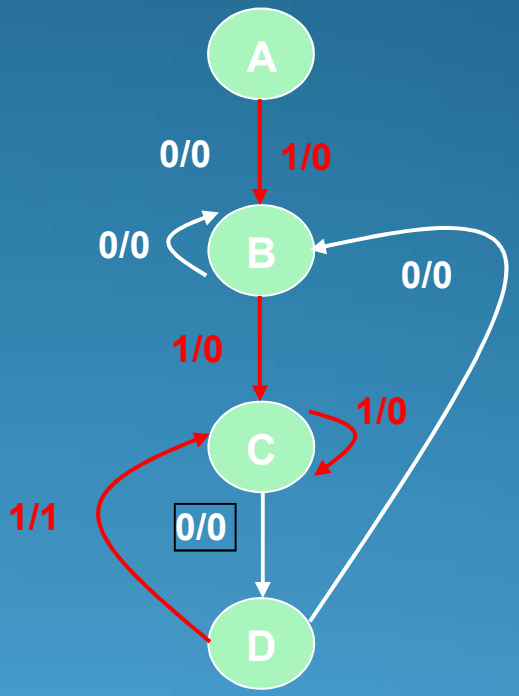
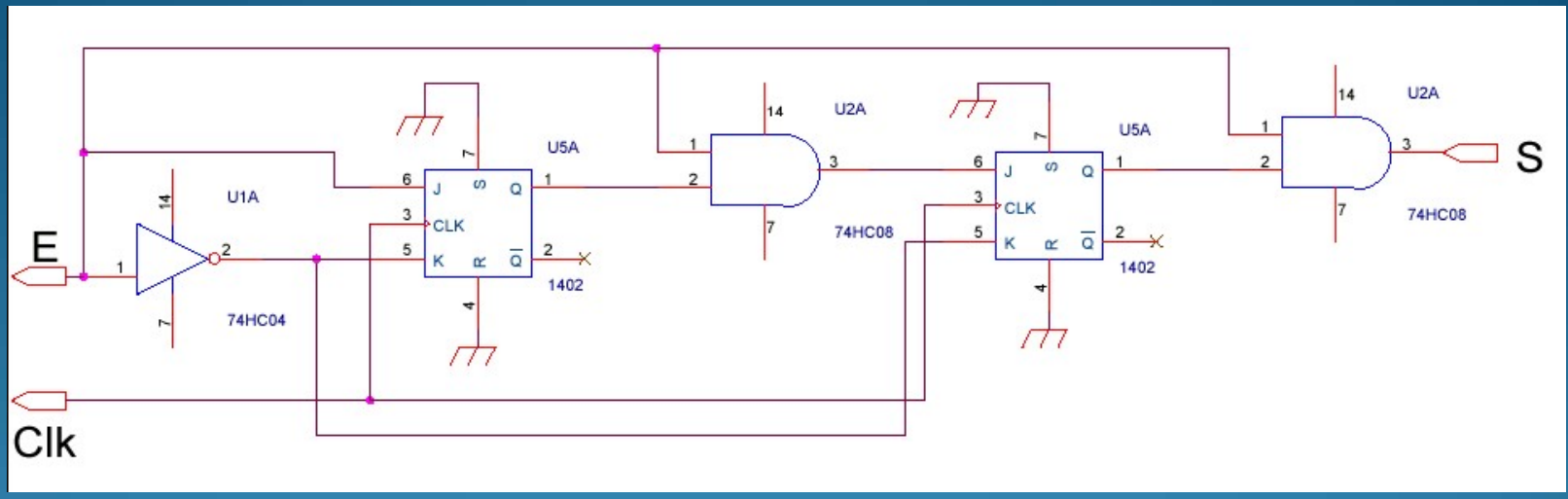


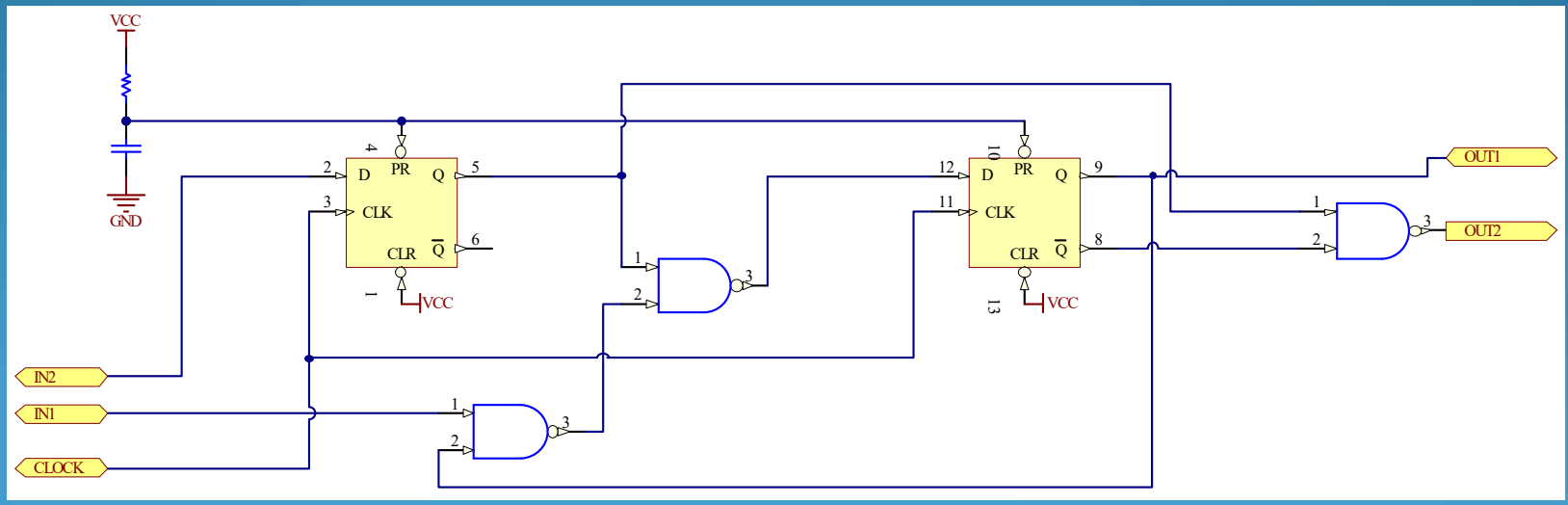
Diagrama de estados



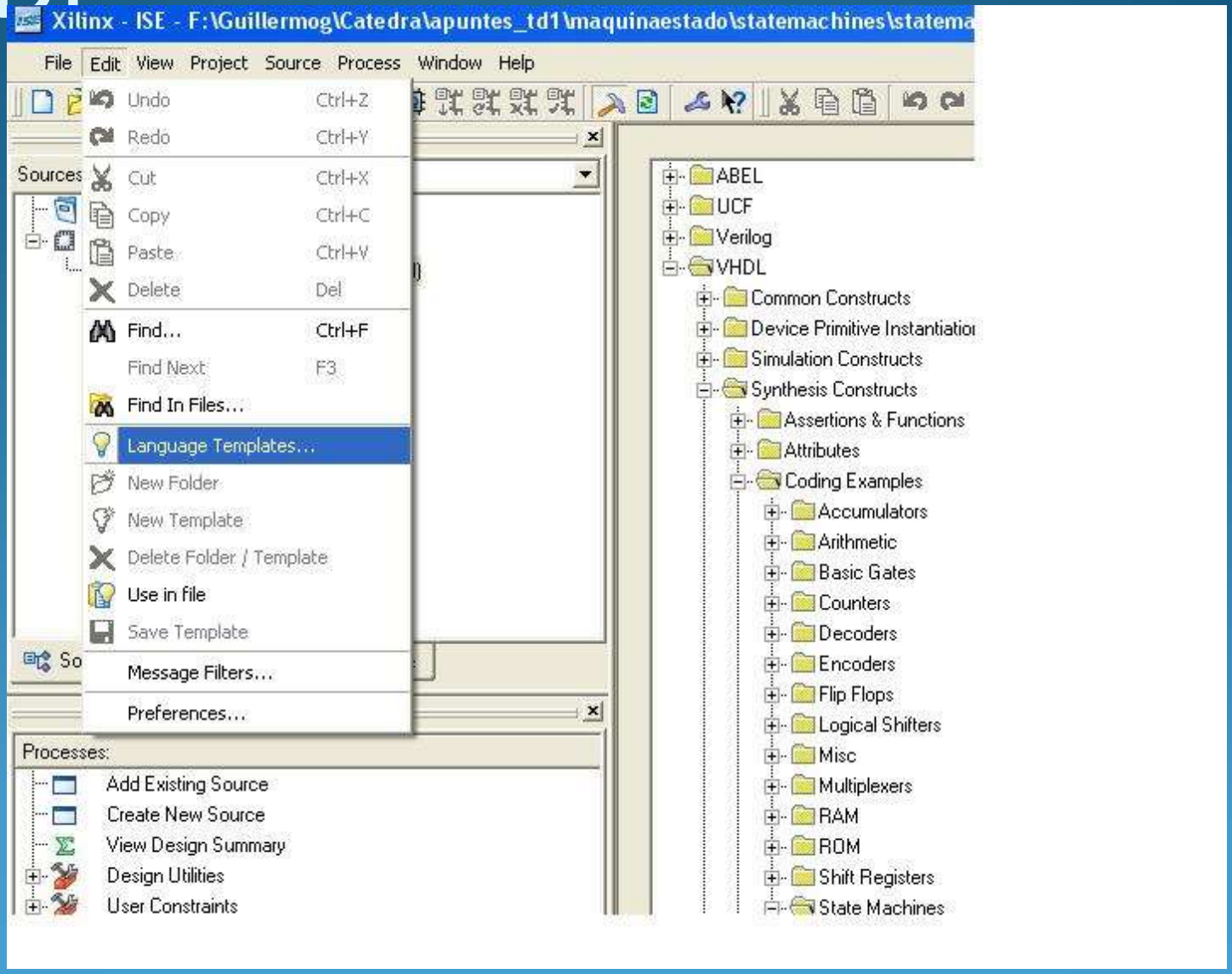


Moore y Mealy

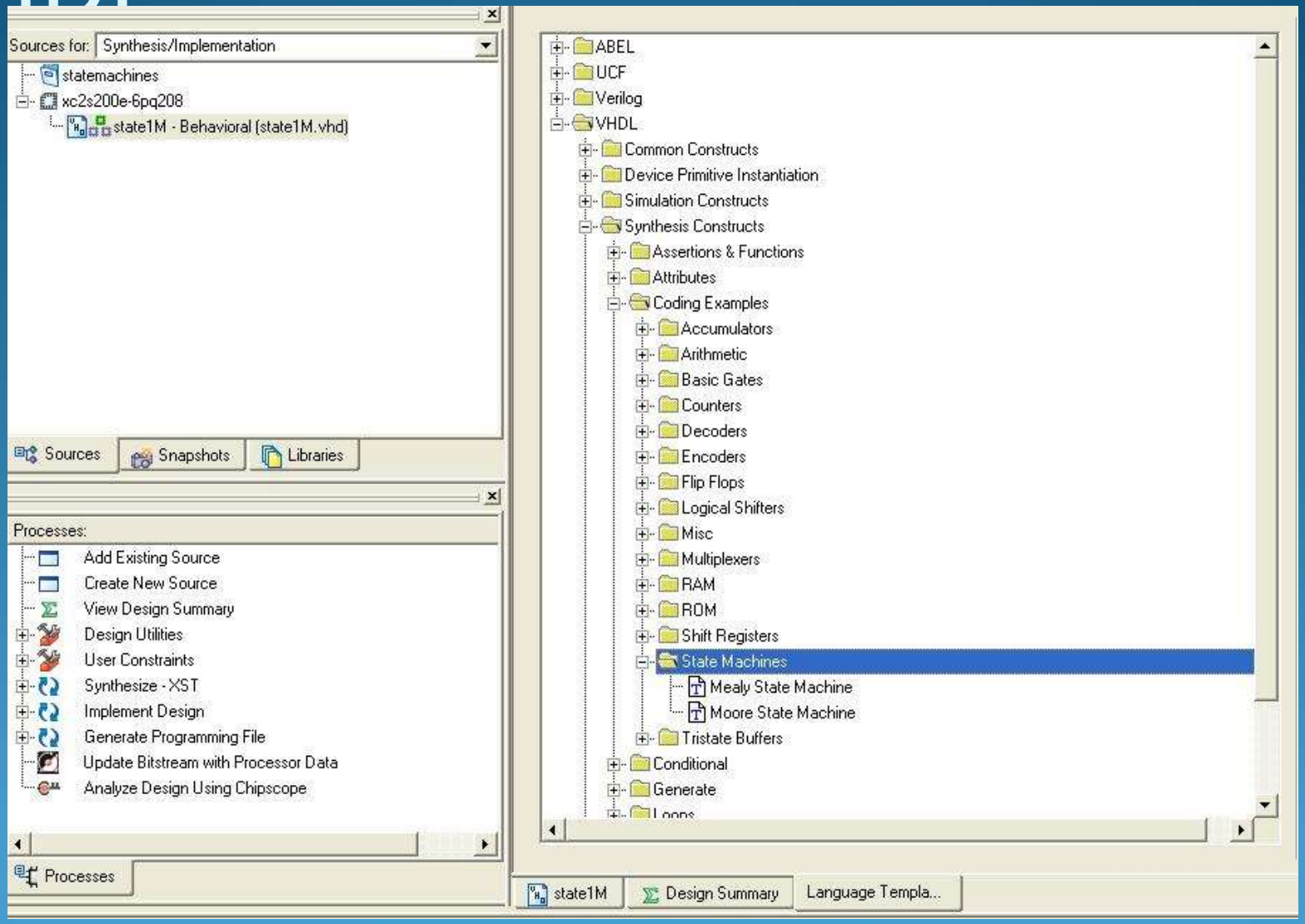




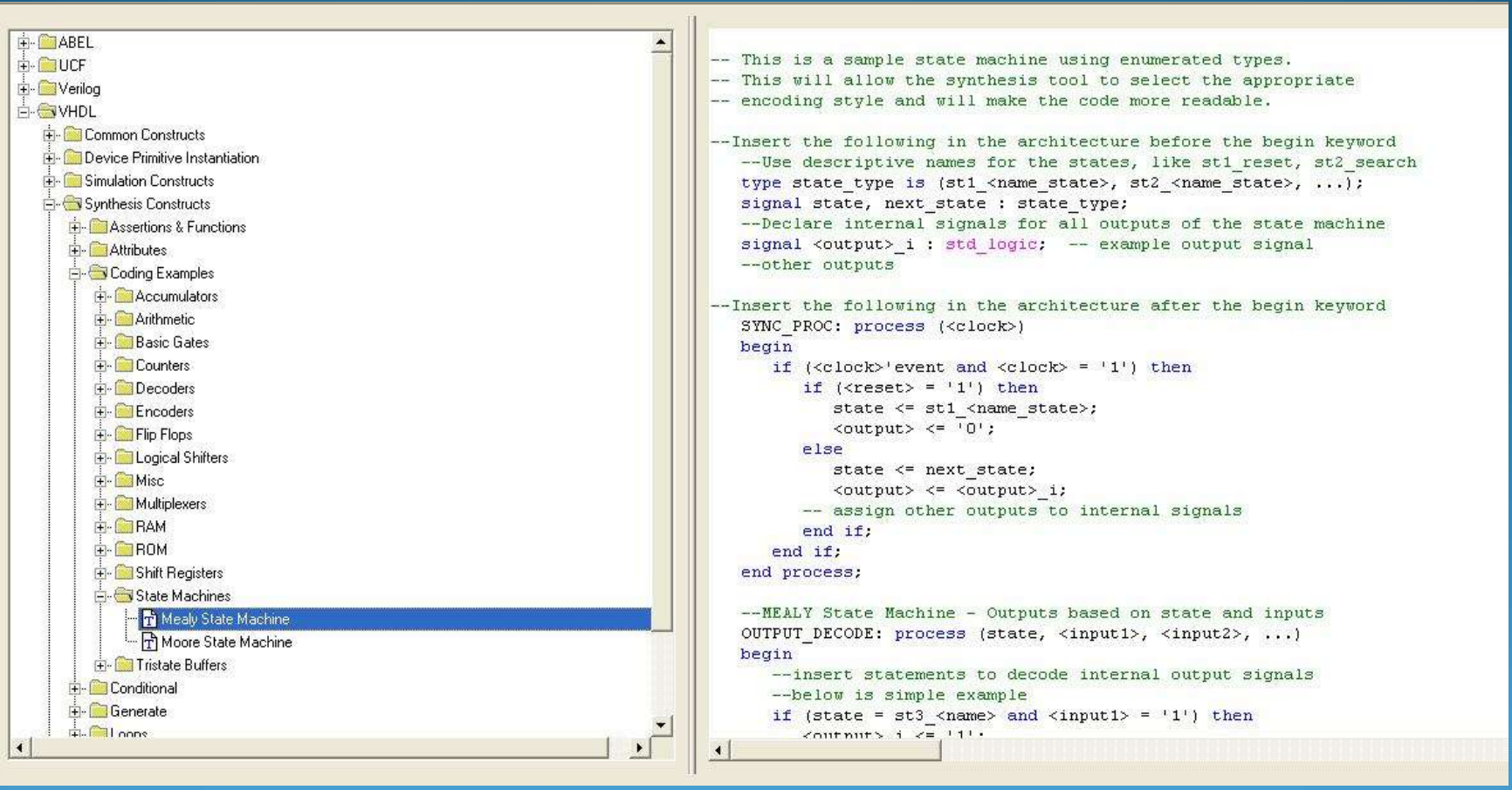
VHDL


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

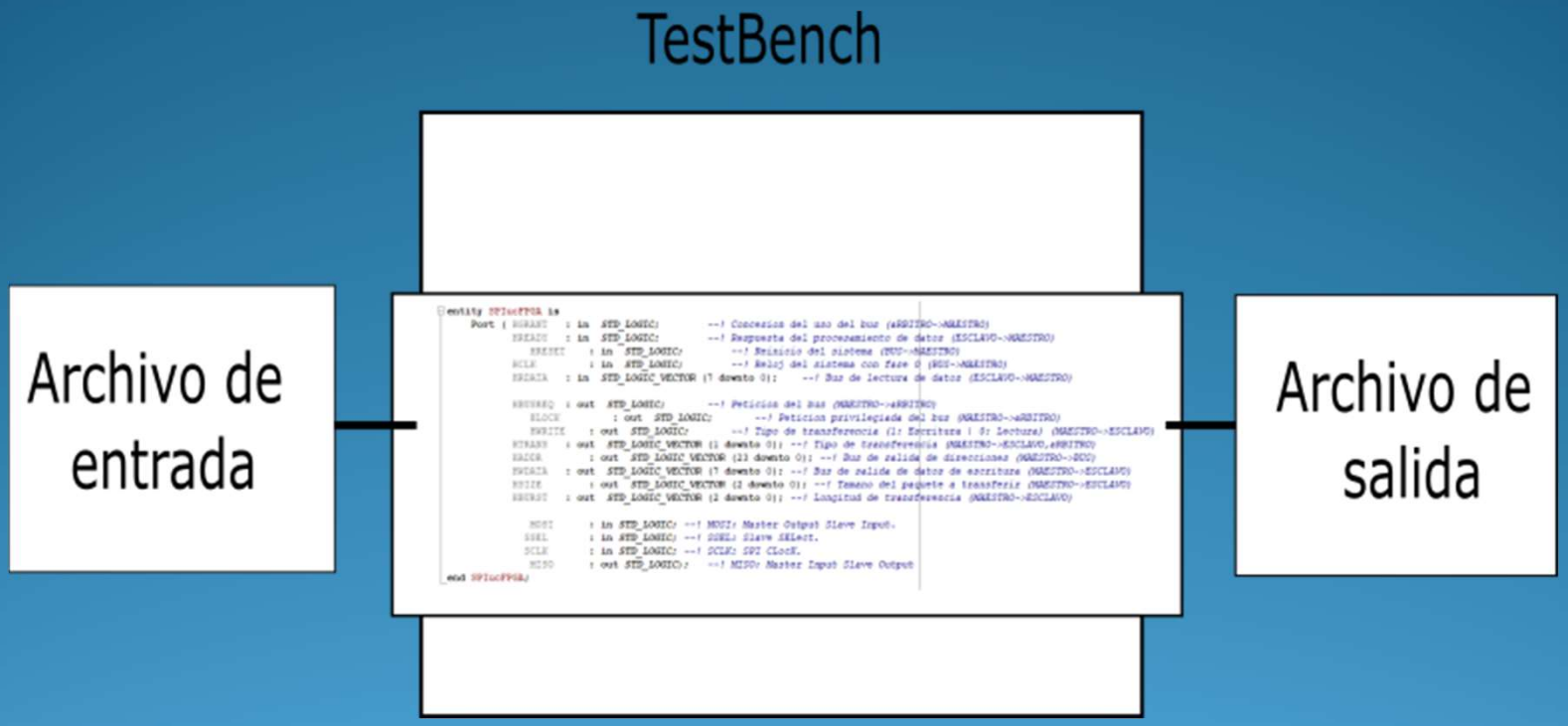
```
NEXT_STATE_DECODE: process (state, <input1>, <input2>, ...)
begin
   --declare default state for next_state to avoid latches
   next_state <= state;  --default is to stay in current state
   --insert statements to decode next_state
   --below is a simple example
   case (state) is
      when st1_<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.

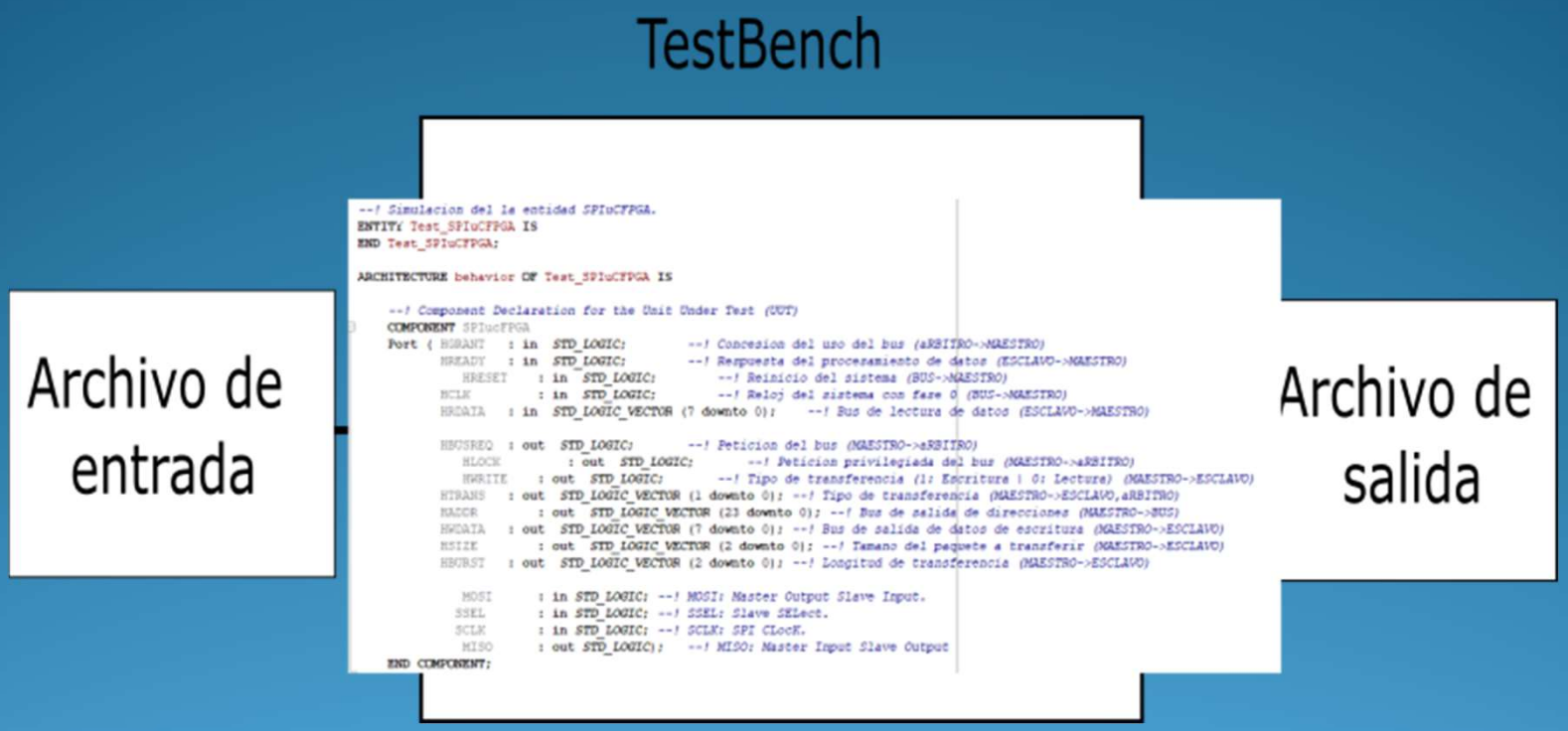
   La puerta se controla por dos salidas, S1 Y S0 cuando S1=1 la puerta se abre, cuando S0=1 la puerta se cierra. Diseñar el autómata, dibujar el circuito, simularlo en CedarLogic y describirlo en Verilog con verificación.

Esquema general

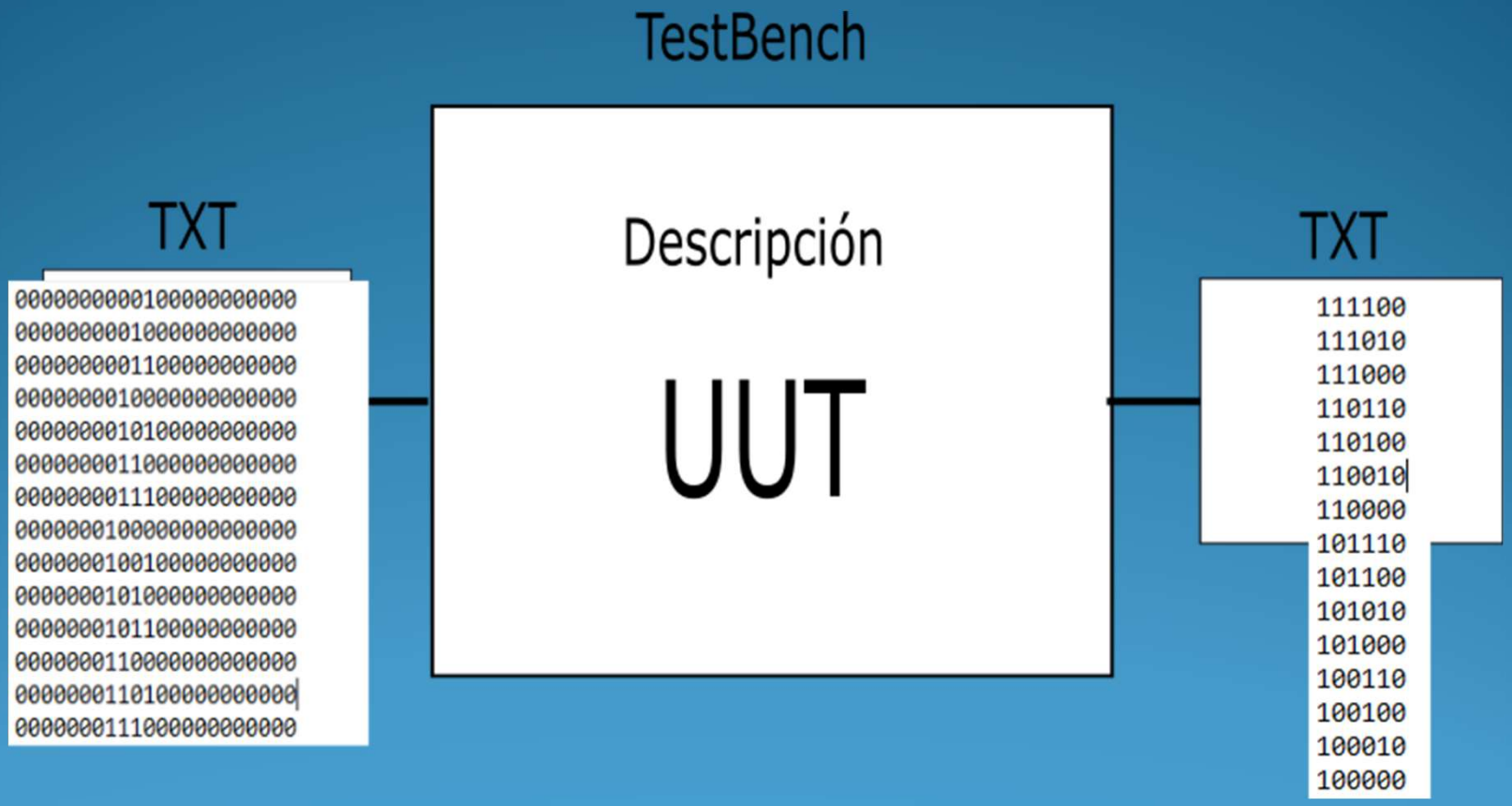
Esquema general



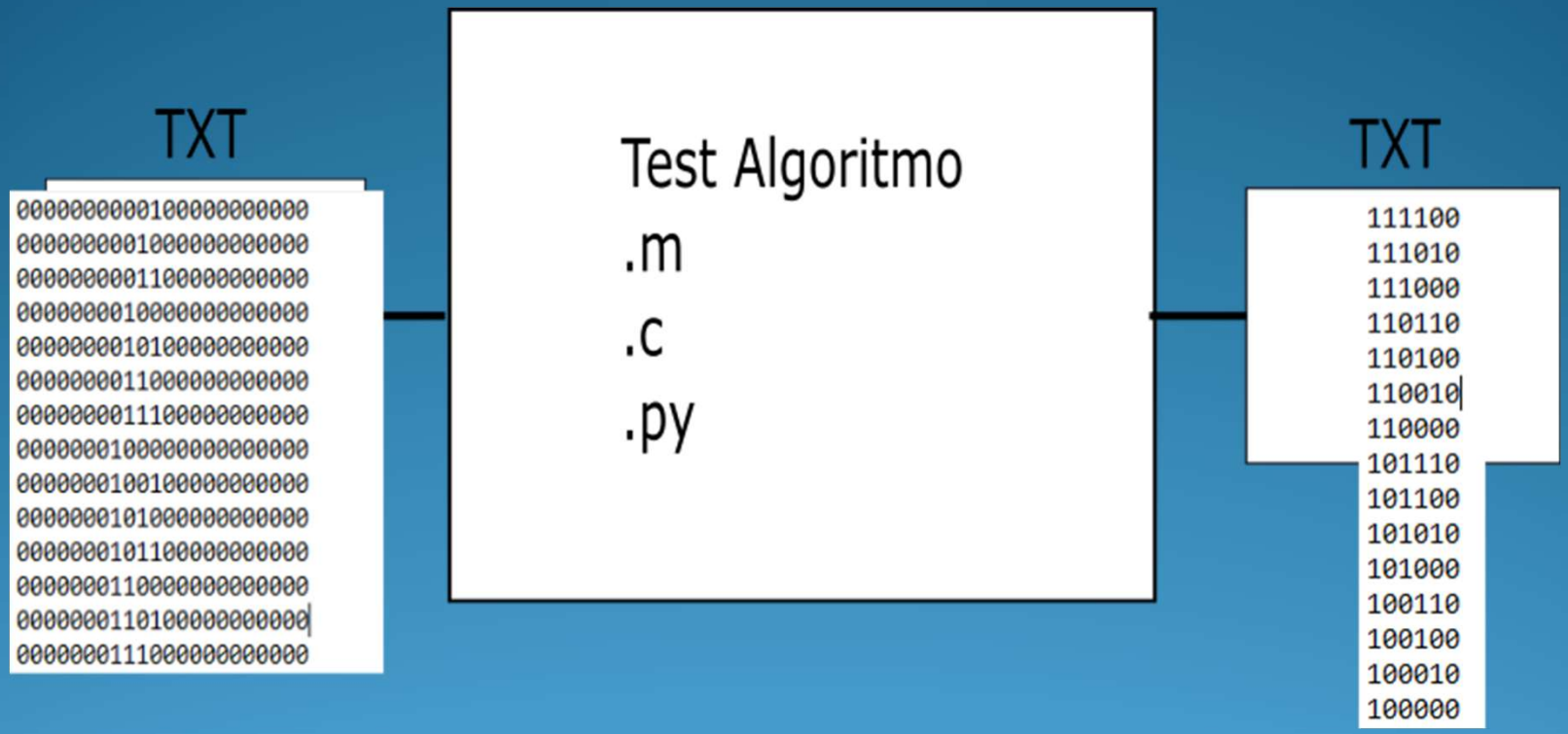
Esquema general



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;
```