Memoria

Sistemas Electrónicos para la Automatización

Docente: *Jorge Romero Sánchez*

Diseño, Simulación y Síntesis de una FSM en VHDL

***Jorge Benavides Macías**: jorge2@uma.es*

Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Contenido

Lista de figuras

Lista de códigos

1. Especificaciones

- Diseñar un enunciado, simular y sintetizar una FSM de vuestra elección al estilo Moore ó Mealy con entre 3 y 8 estados y con sólo 2 procesos: uno secuencial y uno combinacional.
- Como máximo tendrá 2 entradas y 3 salidas.
- Elaborar un informe que ilustre el procedimiento de diseño, simulación y síntesis de todo el proceso seguido.
- Podéis utilizar capturas de pantalla (cronogramas de simulación, esquemático RTL), etc.
- Incluir en el PDF los códigos VHDL empleados y la captura del fichero .CSV generado y del fichero original de estímulos en función del retardo.

2. Diseño

2.1. Estudio teórico

Se pretende diseñar un circuito secuencial capaz de **detectar una secuencia de bits concreta**. El circuito secuencial se modela mediante una máquina de estados de tipo Mealy, como se muestra en la Figura 1, tiene como entrada un valor 'X' que intenta acertar el número correcto asociado a dicho estado y cuando completa la secuencia la 'Y' vale 1, en el resto de estados vale '0'. La secuencia correcta es ***101001***.

2.1.1. Diagrama de estados

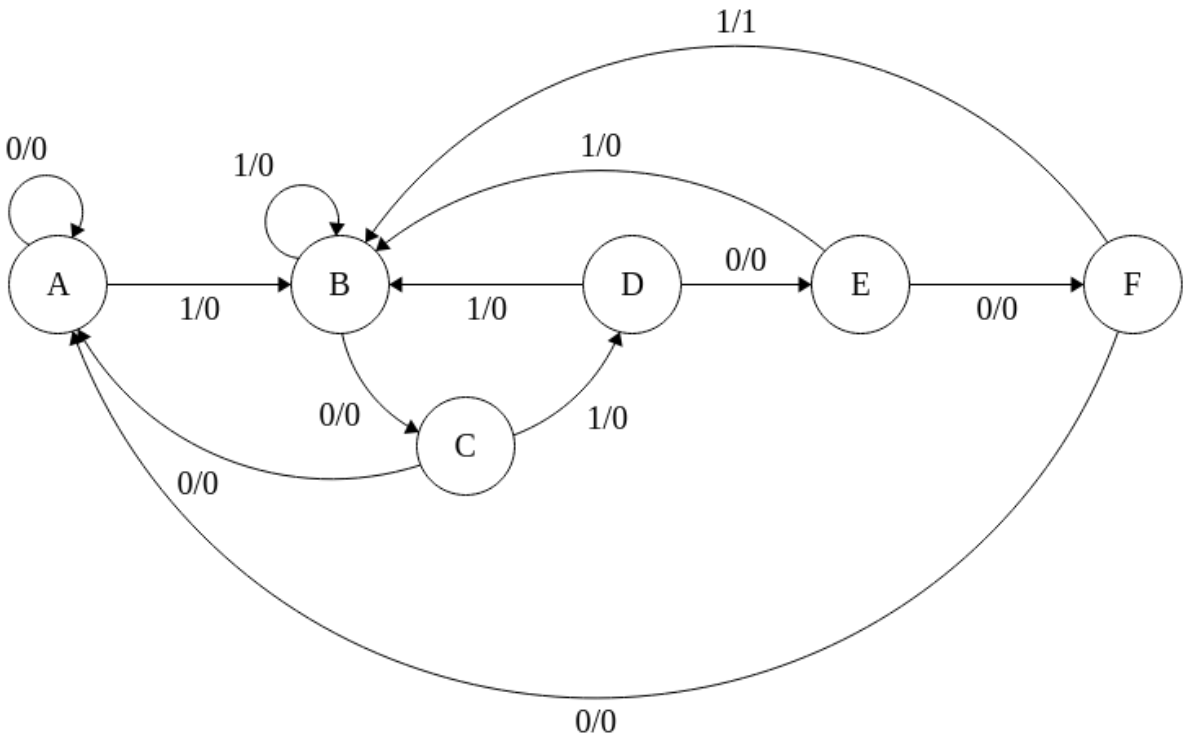

Figura 1: Máquina de estados de tipo Mealy

Explicación de las transiciones:

- **A**: Estado inicial. No hay bit correctos. (Y = 0)
- **A->B**: Un bit correcto. (Y = 0)
- **B->C**: Dos bits correctos. (Y = 0)

- **C->D**: Tres bits correctos. (Y = 0)
- **D->E**: Cuatro bits correctos. (Y = 0)
- **E->F**: Cinco bits correctos. (Y = 0)
- **F->B**: Seis bits y secuencia correcta. (Y = 1)

Cuando se introduce un bit erróneo, se comprueba si la cadena de bits que ha llegado puede ser parte de una nueva secuencia correcta, por lo que no se vuelve directamente al estado inicial.

De esta manera, en los estados **B**, **D**, **E** y **F** al introducir un **uno**, se vuelve al estado B. Sin embargo, en los estados **C**, **A** y **F** vuelven al estado inicial **A**.

2.2. Entidad y arquitectura

La descripción de la Figura 1 en lenguaje hardware empieza por la entidad y los puertos; la entidad descrita en la Código 1 cuenta con 4 puertos de entrada y uno de salida, además de un tipo de variable definido llamado "Estado". Los puertos de entrada se dividen en datos y control, la *X* es de tipo binario como hemos visto en el diagrama de la Figura 1 y los 3 restantes son para el control que todo sistema secuencial necesita.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity detector_secuencia is
5
6    port(x     : in  std_logic;
7      clk   : in  std_logic;
8      reset : in  std_logic;
9      cke   : in  std_logic;
10     y     : out std_logic);
11   type Estado is (A, B, C, D, E, F);
12
13 end detector_secuencia;
```

Código 1: Entidad de la FSM

La arquitectura propuesta en la Código 2 cuenta con dos procesos muy bien definidos, uno combinacional y otro secuencial, el combinacional representa los distintos estados en los que se encuentra la máquina, la lista de sensibilidades es la que representa la dependencia con el estado actual y la entrada 'X'.

Como máquina Mealy la salida está dentro del condicional ya que depende de la entrada. El proceso secuencial tiene una lista de sensibilidades que depende del *clk*, *reset* y *cke*. Los cambios de estado se producen en los flancos de subida del reloj.

```
architecture Behavioral of detector_secuencia is

  signal Estado_Actual  : Estado := A;
  signal Proximo_Estado : Estado;

begin
  Combinacional : process(x, Estado_Actual)
  begin
    case Estado_Actual is
-------------------------------------------------
      when A =>
        -- Ecuacion de Transicion de Estado A:
        if x = '1' then
          Proximo_Estado <= B;
          -- Ecuacion de salida A:
          y <= '0';
        else
          Proximo_Estado <= A;
          y <= '0';
        end if;
-------------------------------------------------
      when B =>
        -- Ecuacion de Transicion de Estado B:
        if x = '0' then
          Proximo_Estado <= C;
          -- Ecuacion de salida B:
          y <= '0';
        else
          Proximo_Estado <= A;
          y <= '0';
        end if;
-------------------------------------------------
      when C =>
        -- Ecuacion de Transicion de Estado C:
        if x = '1' then
          Proximo_Estado <= D;
          -- Ecuacion de salida C:
          y <= '0';
        else
```

```
40            Proximo_Estado <= A;
41            -- Ecuacion de salida C:
42            y <= '0';
43          end if;
44  -------------------------------------------------
45        when D =>
46          -- Ecuacion de Transicion de Estado:
47          if x = '0' then
48            Proximo_Estado <= E;
49            -- Ecuacion de salida:
50            y <= '0';
51          else
52            Proximo_Estado <= B;
53            -- Ecuacion de salida:
54            y <= '0';
55          end if;
56  -------------------------------------------------
57        when E =>
58  -- Ecuacion de Transicion de Estado:
59          if x = '0' then
60            Proximo_Estado <= F;
61            -- Ecuacion de salida:
62            y <= '0';
63          else
64            Proximo_Estado <= B;
65            -- Ecuacion de salida:
66            y <= '0';
67          end if;
68  -------------------------------------------------
69        when F =>
70          -- Ecuacion de Transicion de Estado:
71          if x = '1' then
72            Proximo_Estado <= B;
73            -- Ecuacion de salida:
74            y <= '1'; -- La secuencia es correcta '101001'
75          else
76            Proximo_Estado <= A;
77            -- Ecuacion de salida:
78            y <= '0';
```

```
79          end if;
80      end case;
81    end process Combinacional;
82
83    Secuencial : process(clk, reset, cke)
84    begin
85      if reset = '1' then
86        Estado_Actual <= A;
87      elsif rising_edge(clk) then
88        if cke = '1' then
89          Estado_Actual <= Proximo_Estado;
90          end if;
91      end if;
92    end process Secuencial;
93  end Behavioral;
```

Código 2: Arquitectura de la FSM

3. Simulación

3.1. Test Bench

Para realizar la simulación del dispositivo hemos usado la plantilla del campus "Test_Bench_Fichero" con una serie de modificaciones, se observa claramente en el Código 3 que se ha instanciado el componente detector_secuencia (Código 1), se han declarado una serie de señales internas, inicializadas con el valor 'U' (*"uninitialized "*) para trabajar en simulación, y que en el *DUT* (*Device Under Test*) se ha asociado a cada puerto del componente una señal interna.

En el proceso "estímulos_Desde_Fichero" también hemos modificado el tamaño del input data, ya que solo necesitamos un estímulo.

La estimulación de las señales de control se ha hecho mediante procesos ya que es más sencillo de condificar, la entrada de datos se ha hecho en el fichero de estímulos.

```
1  ----------------------------------------------------------------------------------
2  -- Company: Universidad de Málaga
3  -- Engineer: Izan Amador, Jorge L. Benavides
4  --
5  -- Create Date: 23.11.2022 17:47:41
6  -- Design Name: FSM - Mealy
7  -- Module Name: Test_Bench - Behavioral
8  -- Project Name: Sincronizador
9  -- Target Devices: Zybo
10 -- Tool Versions: Vivado 2022.1
11 -- Description: Debouncer for a button.
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 ----------------------------------------------------------------------------------
19
20
21 library IEEE;
22 use IEEE.STD_LOGIC_1164.ALL;
23 use STD.textIO.ALL;                    -- Se va a hacer uso de ficheros.
24
25 entity Test_Bench_Fichero is
26 --  Port ( );
```

```
27 end Test_Bench_Fichero;
28
29 architecture Comportamiento of Test_Bench_Fichero is
30
31   component detector_secuencia
32     port(x  : in  std_logic;
33          clk   : in  std_logic;
34          reset : in  std_logic;
35          cke   : in  std_logic;
36          y     : out std_logic);
37   end Component detector_secuencia;
38
39
40   constant semiperiodo : time    := 10 ns;
41   signal x_interno, y_interno, reset_interno, cke_interno : std_logic := 'U';
42   signal clk_interno : std_logic := '0';
43
44
45 begin
46
47   DUT : detector_secuencia
48     port map(
49       x => x_interno,
50       y => y_interno,
51       reset => reset_interno,
52       clk => clk_interno,
53       cke => cke_interno);
54
55 -- Taken from The Student Guide to VHDL, Peter J.Asheden
56   clock_gen : process (clk_interno) is
57   begin
58     if clk_interno = '0' then
59       clk_interno <= '1' after semiperiodo,
60                      '0' after 2*semiperiodo;
61     end if;
62   end process clock_gen;
63
64   cke_interno <= '1';
65
```

```
66    reset : process
67    begin
68      reset_interno <= '0';
69      wait for 5 ns;
70      reset_interno <= '1';
71      wait for 5 ns;
72      reset_interno <= '0';
73      wait;
74    end process reset;
75
76    Estimulos_Desde_Fichero : process
77
78      file Input_File  : text;
79      file Output_File : text;
80
81      variable Input_Data   : BIT_VECTOR(0 downto 0) := (OTHERS => '0');
82      variable Delay        : time                   := 0 ms;
83      variable Input_Line   : line                   := NULL;
84      variable Output_Line  : line                   := NULL;
85      variable Std_Out_Line : line                   := NULL;
86      variable Correcto     : Boolean                := True;
87      constant Coma         : character              := ',';
88
89
90    begin
91
92  -- detector_secuencia_mealy_Estimulos.txt contiene los estímulos y los tiempos de
     ↪  retardo para el semisumador.
93      file_open(Input_File,
     ↪  "C:\Users\izana\Documents\GitHub\SEA\Estimulos\detector_secuencia_mealy_Estimulos.txt",
     ↪  read_mode);
94  -- detector_secuencia_mealy_Estimulos.csv contiene los estímulos y los tiempos de
     ↪  retardo para el Analog Discovery 2.
95      file_open(Output_File,
     ↪  "C:\Users\izana\Documents\GitHub\SEA\CSV\detector_secuencia_mealy_Estimulos.csv",
     ↪  write_mode);
96
97  -- Titles: Son para el formato EXCEL *.CSV (Comma Separated Values):
98      write(Std_Out_Line, string'("Retardo"), right, 7);
```

```
99    write(Std_Out_Line, Coma, right, 1);
100   write(Std_Out_Line, string'("Entradas"), right, 8);
101
102   Output_Line := Std_Out_Line;
103
104   writeline(output, Std_Out_Line);
105   writeline(Output_File, Output_Line);
106
107   while (not endfile(Input_File)) loop
108
109     readline(Input_File, Input_Line);
110
111     read(Input_Line, Delay, Correcto);  -- Comprobación de que se trata de un
      ↪ texto que representa
112     -- el retardo, si no es así leemos la siguiente línea.
113     if Correcto then
114
115       read(Input_Line, Input_Data);  -- El siguiente campo es el vector de
      ↪ pruebas.
116       x_interno <= TO_STDLOGICVECTOR(Input_Data)(0);
117
118       -- De forma simultánea lo volcaremos en consola en csv.
119       write(Std_Out_Line, Delay, right, 5);  -- Longitud del retardo, ej. "20 ms".
120       write(Std_Out_Line, Coma, right, 1);
121       write(Std_Out_Line, Input_Data, right, 2);  --Longitud de los datos de
      ↪ entrada.
122
123       Output_Line := Std_Out_Line;
124
125       writeline(output, Std_Out_Line);
126       writeline(Output_File, Output_Line);
127
128       wait for Delay;
129     end if;
130   end loop;
131
132   file_close(Input_File);             -- Cerramos el fichero de entrada.
133   file_close(Output_File);            -- Cerramos el fichero de salida.
134   wait;
```

```
135   end process Estimulos_Desde_Fichero;
136
137
138 end Comportamiento;
```

Código 3: Test bench de la FSM

3.2. Fichero de estímulos

En el fichero de estímulo hemos descrito una serie de secuencias para probar todos los estados, en algunos casos son correctas y en otros incorrectas para observar los distintos saltos.

```
#Fichero de Estímulos de detector secuencia mealy.
#Device Name: Discovery2NI
#Nombre: Izan Amador, Jorge Benavides
#Fecha: 29 de Noviembre de 2022.
#
# Delay Time (ns) Input (I).

# Wait for reset
    40 ns 0
# Secuencia correcta
    20 ns 1
    20 ns 0
    20 ns 1
    20 ns 0
    20 ns 0
    20 ns 1

# Secuencias aleatorias (erróneas)
# Primera secuencia

    25 ns 0
    25 ns 0
    25 ns 1
    25 ns 1
    25 ns 0
    25 ns 1

# Segunda secuencia

    25 ns 1
    25 ns 1
    25 ns 0
    25 ns 0
```

```
25 ns 0
25 ns 1
```

3.3. Cronograma de simulación

El cronograma resultante solo tiene una secuencia correcta (señal 'Y' en rojo del cronograma) que coincide con el fichero de estímulos y que además es el único código que paso por todo los estados disponibles tal y como se ha definido en la Figura 1.

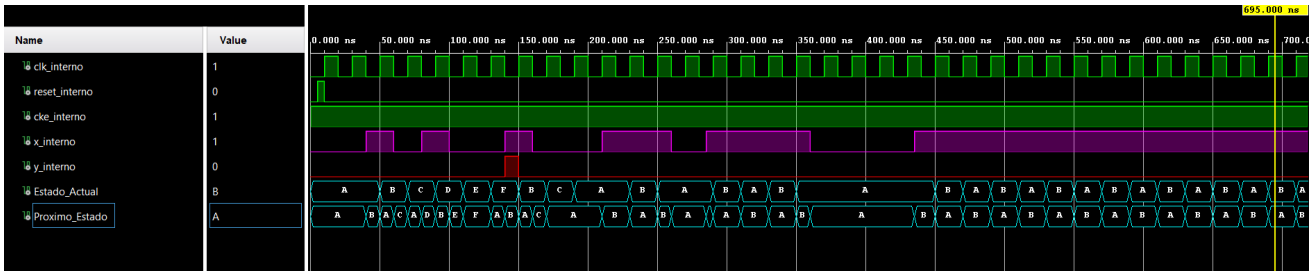

Figura 2: Cronograma de simulación de la FSM

3.4. Fichero CSV generado

El CSV generado para una posible implementación en el Analog Discovery.

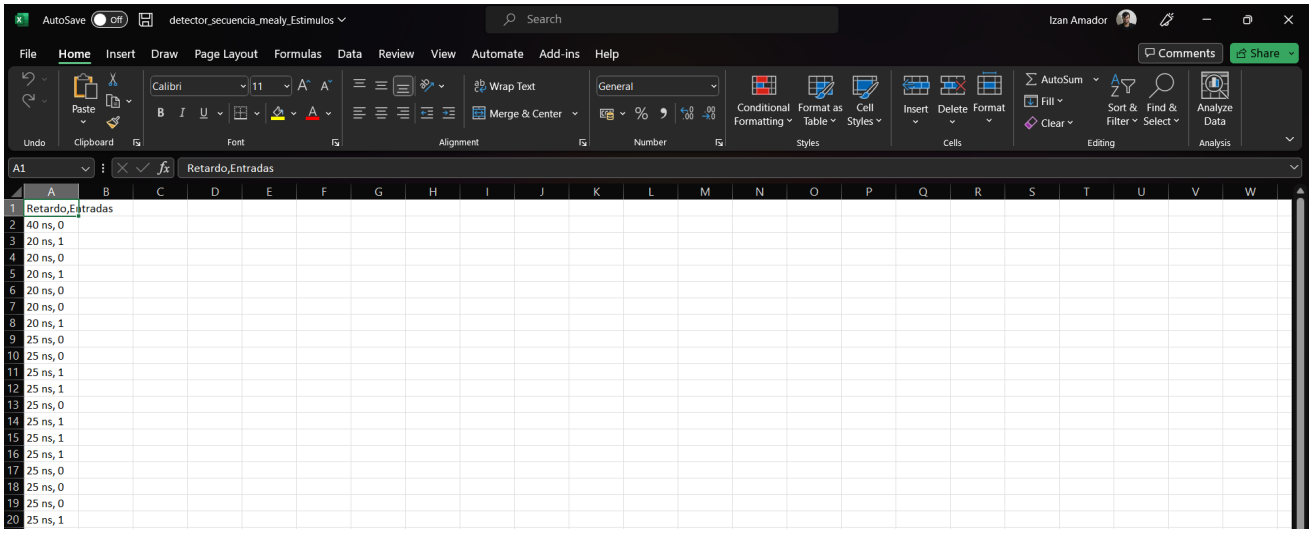

Figura 3: Fichero CSV generado para la FSM

4. Síntesis

4.1. Esquemático RTL

La síntesis del diseño ha sido todo un éxito, cuenta con 5 multiplexores y un registro, como se puede ver en la Figura 4 no hay ningún *latch* por la tanto la síntesis es correcta.

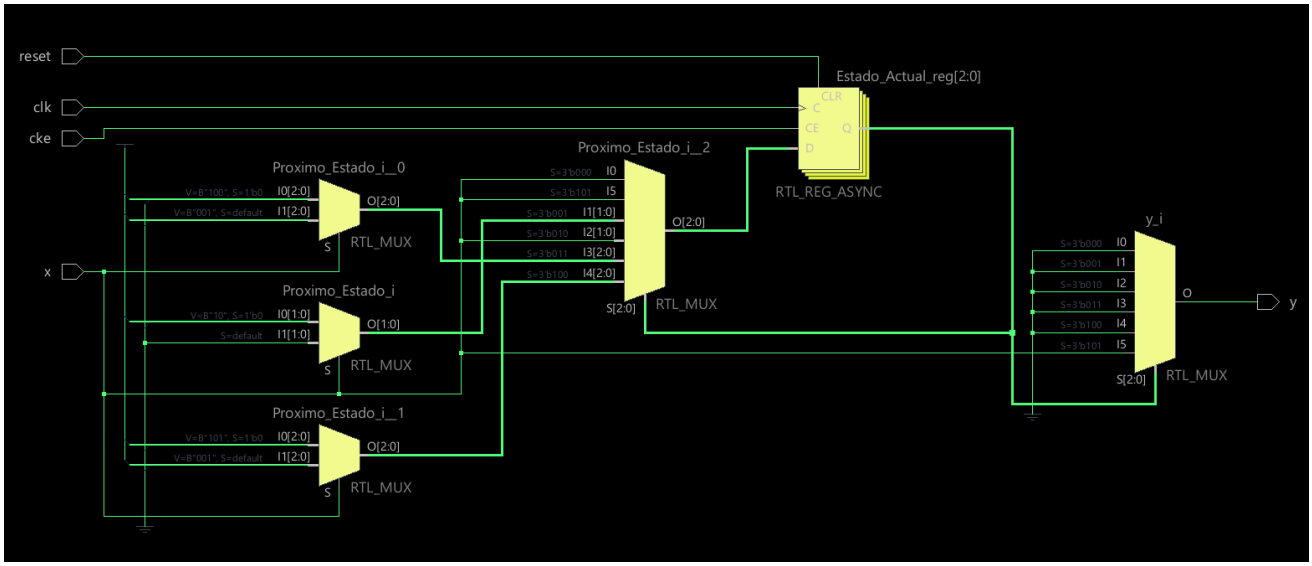

Figura 4: Esquemático RTL de la FSM