PRÁCTICA 9: INTRODUCCIÓN A LAS MÁQUINAS DE ESTADO. CERRADURA ELECTRÓNICA

Alumnos: Pablo Menéndez Ruiz de Azúa y Carles Olucha Royo
Fecha de realización: 26/11/2020
Fecha de entrega: 3/12/2020
Asignatura: Sistemas digitales I
Profesores: Julian Spahr y Álvaro Padierna

Índice

Objetivos

El propósito de esta práctica es:

- Comprender el funcionamiento de las máquinas de estado
- Diseñar y simular un detector de secuencia para una cerradura electrónica.
- Aprender a describir circuitos secuenciales con lenguajes de descripción de hardware VHDL.

Introducción

En esta práctica haremos un circuito digital secuencial para implementar una sencilla cerradura electrónica.

El circuito detectará una secuencia de pulsación de teclas y activará una salida cuando la secuencia sea completa. Para arrancar el sistema hay que apretar el pulsador que hace de reset. A continuación, los pulsadores que activan las señales p0 y p1 sirven para introducir una secuencia al circuito.

Cuando éste detecte la secuencia **p0 – p1 – p1 – p0**, entre las pulsaciones del usuario, debe iluminar un LED verde (**VALID**) para indicar que se ha recibido la secuencia anterior. Se permite el solapamiento de secuencias, de tal forma que si se pulsa **p0 – p1 – p1 – p0 – p1 – p1 – p0** se darán por válidas dos secuencias.

Dicho LED verde permanecerá iluminado hasta que se pulse cualquiera de los pulsadores, pudiendo volver a iluminarse si se vuelve a introducir de nuevo la secuencia correcta.

Mientras no exista una secuencia correcta, se iluminará un LED rojo (**ESPERA**).

El alumno debe tener en cuenta que pulsar uno de los interruptores genera un pulso que dura muchos ciclos de reloj. Por tanto, es necesario diseñar un detector de flanco para cada pulsador, en un bloque independiente llamado **DetectorFlancoBajada**. Además, queremos que sea de bajada porque los pulsadores son activos a nivel bajo.

Cálculos previos

Antes de empezar con la práctica, debemos programar todos los componentes que van a formar parte de nuestro detector de secuencia. En este caso, únicamente tenemos que programar dos máquinas de estado. En primer lugar, una máquina que actúe como un detector de flanco de bajada y, en segundo lugar, una máquina que va a ser el propio detector de secuencia.

Detector de flanco de bajada

Antes de empezar a programar este bloque, debemos hacernos un diagrama de estados ya que nos va a ayudar a entender cómo funciona y además nos va a facilitar mucho el trabajo a la hora de realizar la programación en VHDL.

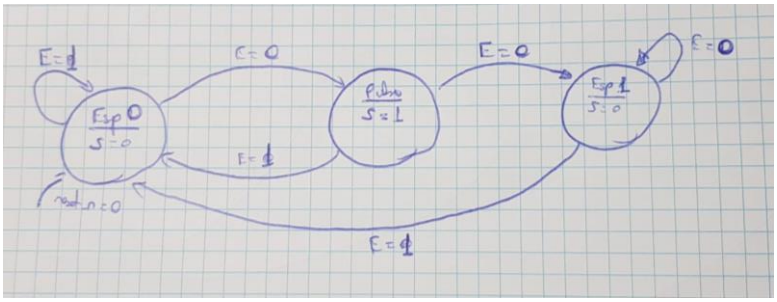

Figura 1 – Diagrama de estados del detector de flanco de bajada

Podemos ver que en este diagrama de estados tenemos tres estados: Esp0, Pulso y Esp1. Inicialmente, si reseteamos el circuito, nuestra entrada vale 1 y por tanto no pasa nada (Estado Esp0). En el momento, en el que nuestra entrada vale 0 y el reloj se encuentra en un flanco, cambiamos al estado pulso y nuestra salida vale 1. Este pulso dura un ciclo y vuelve a 0. Si cuando vuelve a 0 la salida, E sigue valiendo 0, esperamos a que se ponga en 1 (Esp1). Por último, si estamos en Pulso o en Esp1 y la entrada vale 1, volvemos al estado inicial.

Cuando vemos un detector de flanco, generalmente vemos que las entradas del diagrama de estados son al revés. Es decir, donde aparecen 1 deberían ser 0 y viceversa. Sin embargo, en esta práctica estamos trabajando con pulsadores que son activos a nivel bajo.

Una vez realizado nuestro diagrama de estados, nos disponemos a programar la máquina en VHDL.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity DetectorFlancobajada is
5     port(
6        e        : in std_logic;
7        reset_n  : in std_logic;
8        clk      : in std_logic;
9        s        : out std_logic
10    );
11 end DetectorFlancobajada;
12
13 architecture behavioral of DetectorFlancobajada is
14    type t_estado is (Esp0,Pulso,Esp1);
15
16    signal estado_act,estado_sig : t_estado;
17
18 
19 TransicionEstados: process(estado_act,e)
20 begin
21    estado_sig <=estado_act;
22
23    case estado_act is
24       when Esp0 =>
25          if e = '0' then
26             estado_sig <= Pulso;
27          end if;
28       when Pulso =>
29          if e = '0' then
30             estado_sig <= Esp1;
31          end if;
32          if e = '1' then
33             estado_sig <= Esp0;
34          end if;
35       when Esp1 =>
36          if e = '1' then
37             estado_sig <= Esp0;
38          end if;
39       when others =>
40          estado_sig <=Esp0;
41    end case;
42 end process;
```

```
43  
44  Salidas: process (estado_act)
45  begin
46  
47     s<='0';
48     case estado_act is
49        when Esp0 =>
50           null;
51        when Pulso =>
52           s<='1';
53        when Esp1 =>
54           null;
55        when others =>
56           null;
57     end case;
58  end process;
59  
60  VarEstado : process(clk,reset_n)
61  begin
62     if reset_n='0' then
63        estado_act <=Esp0;
64     else
65        if falling_edge(clk) then
66           estado_act <= estado_sig;
67        end if;
68     end if;
69  end process;
70  end behavioral;
```

Figura 2 – Código VHDL del detector de flanco de bajada

Una vez hecho el diagrama de estados, es muy fácil comprender el código. En primer lugar, tenemos que definirnos nuestros tres estados. Después vamos a programar 3 procesos: el primero para las transiciones entre estados dependiendo de la entrada, el segundo cuál es la salida dependiendo del estado actual y el último cómo se cambia de un estado a otro

Detector de secuencia

Al igual que con el detector de flanco, debemos hacernos un diagrama de estados ya que nos va a ayudar a entender cómo funciona este bloque y además nos va a facilitar mucho el trabajo a la hora de realizar la programación en VHDL.

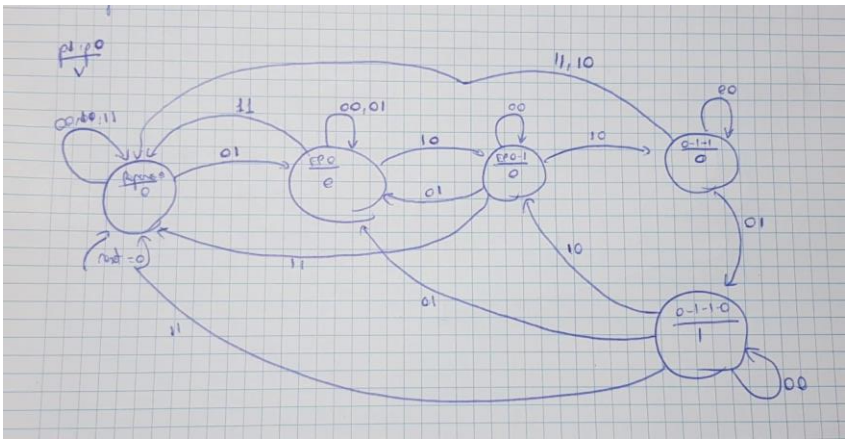

Figura 3 – Diagrama de estados del detector de secuencia (p0-p1-p1-p0)

En primer lugar, si pulsamos el botón de reset, nos vamos al estado inicial (reposo). Hasta que no pulsemos el p0, no cambiamos al estado Ep0. En todos los casos si pulsamos los dos pulsadores a la vez, se vuelve al estado inicial o de reposo.

Si estamos en el p0 y no pulsamos nada o volvemos a pulsar p0, nos quedamos en el mismo estado. Si pulsamos p1, nos movemos al estado Ep0p1.

Si estamos en el estado Ep0p1 y no pulsamos nada, nos mantenemos en él. Si pulsamos p0, volvemos al anterior (Ep0) y si pulsamos otra vez p1, nos movemos al estado Ep0p1p1.

Si estamos en el estado Ep0p1p1 y no pulsamos nada, nos mantenemos en él. Si pulsamos p1, volvemos al inicio (reposo) y si pulsamos p0, nos movemos al estado Ep0p1p1p0.

Si estamos en el estado Ep0p1p1p0 y no pulsamos nada, nos mantenemos en él. Si pulsamos p0, volvemos al estado Ep0 y si pulsamos p1 volvemos al estado Ep0p1.

Finalmente, si no nos encontramos en el estado Ep0p1p1p0, la salida VALID será 0, mientras que la salida ESPERA será 1. Y si estamos en este estado, VALID será 1 y ESPERA será 0.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity DetectorSecuencia is
5     port(
6        reset_n  : in std_logic;
7        clk      : in std_logic;
8        p1       : in std_logic;
9        p0       : in std_logic;
10       valid    : out std_logic;
11       espera   : out std_logic
12    );
13 end DetectorSecuencia;
14
15 architecture behavioral of DetectorSecuencia is
16
17    type t_estados is (Reposo,EP0,EP01,EP011,EP0110);
18
19    signal estado_act,estado_sig : t_estados;
20
21    signal entradas: std_logic_vector(1 downto 0);
22
23 begin
24
25 VarEstado : process(clk,reset_n)
26 begin
27    if reset_n = '0' then
28       estado_act <=Reposo;
29    else
30       if falling_edge(clk) then
31          estado_act <= estado_sig;
32       end if;
33    end if;
34 end process VarEstado;
```

```
36  valid <= '1' when estado_act = EP0110 else '0';
37  espera <= '0' when estado_act = EP0110 else '1';
38  entradas(1) <=p1;
39  entradas(0) <=p0;
40
41  TransicionEstados : process(estado_act,entradas)
42  begin
43
44     estado_sig<=estado_act;
45
46     case estado_act is
47        when Reposo =>
48           if entradas = "01" then
49              estado_sig <=EP0;
50           end if;
51        when EP0 =>
52           if entradas = "11" then
53              estado_sig <= Reposo;
54           elsif entradas = "10" then
55              estado_sig <= EP01;
56           end if;
57        when EP01 =>
58           if entradas = "11" then
59              estado_sig <= Reposo;
60           elsif entradas = "10" then
61              estado_sig <= EP011;
62           elsif entradas = "01" then
63              estado_sig <= EP0;
64           end if;
65        when EP011 =>
66           if entradas = "11" then
67              estado_sig <= Reposo;
68           elsif entradas = "10" then
69              estado_sig <= Reposo;
70           elsif entradas = "01" then
71              estado_sig <= EP0110;
72           end if;
73        when EP0110 =>
74           if entradas = "11" then
75              estado_sig <= Reposo;
76           elsif entradas = "10" then
77              estado_sig <= EP01;
78           elsif entradas = "01" then
79              estado_sig <= EP0;
80           end if;
81        when others =>
82           estado_sig <= Reposo;
83     end case;
84  end process;
85  end behavioral;
```

Figura 4 – Código VHDL del detector de secuencia

Desarrollo práctico

Nuestro circuito va a estar formado por dos detectores de flanco a la entrada, que serán usados para los pulsadores. Las salidas de estos detectores de flanco son las entradas del detector de secuencia. Además, tenemos una señal reset que entra y su función es devolver al estado inicial a nuestro circuito. Las salidas de todo nuestro circuito serán VALID y ESPERA, que ya comentamos anteriormente cuando están activas y cuando no. Para entender esto mejor, veamos el diagrama de bloques de la figura 5.

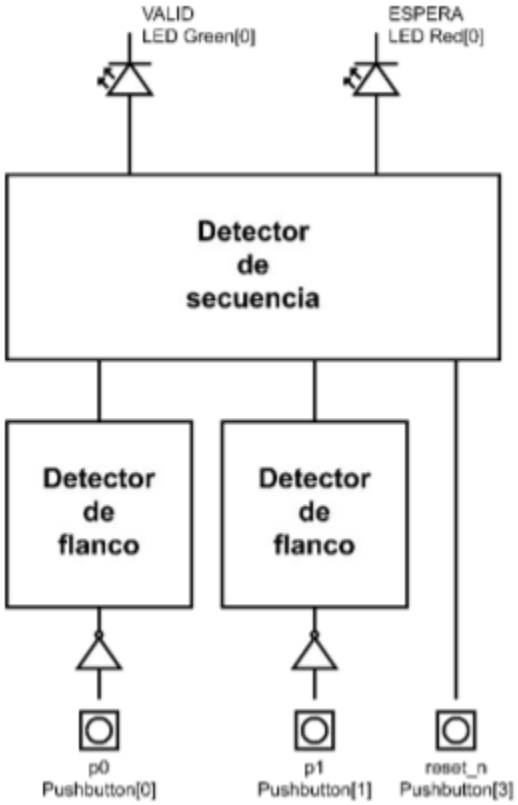

Figura 5 – Diagrama de bloques del circuito

Diseño del circuito en VHDL

Una vez que tenemos todos los componentes que necesitamos para programar nuestro circuito y entendiendo el diagrama de bloques de la figura 5, podemos escribir el código VHDL de nuestro circuito.

```vhdl
entity Practica9 is
    port(
        reset_n  : in std_logic;
        clk      : in std_logic;
        p1       : in std_logic;
        p0       : in std_logic;
        valid    : out std_logic;
        espera   : out std_logic
    );
end Practica9;

architecture behavioral of Practica9 is

    type t_estados is (Reposo,EP0,EP01,EP011,EP0110);

    signal estado_act,estado_sig : t_estados;

    signal entradas: std_logic_vector(1 downto 0);

    component DetectorFlancobajada
        port(
        e        : in std_logic;
        reset_n  : in std_logic;
        clk      : in std_logic;
        s        : out std_logic
    );
    end component;

    component DetectorSecuencia
        port(
        reset_n  : in std_logic;
        clk      : in std_logic;
        p1       : in std_logic;
        p0       : in std_logic;
        valid    : out std_logic;
        espera   : out std_logic
    );
    end component;

begin

DetectorFlanco_1 : DetectorFlancobajada
port map(
    e        => p1,
    reset_n  => reset_n,
    clk      => clk,
    s        => entradas(1)
);

DetectorFlanco_2 : DetectorFlancobajada
port map(
    e        => p0,
    reset_n  => reset_n,
    clk      => clk,
    s        => entradas(0)
);

Detector_secu  : DetectorSecuencia
port map(
    reset_n  => reset_n,
    clk      =>clk,
    p1       =>entradas(1),
    p0       =>entradas(0),
    valid    =>valid,
    espera   =>espera
);

end behavioral;
```

Figura 6 – Cógido VHDL del circuito

Finalmente, el último paso antes de realizar la simulación y la implantación física es la asignación de pines.

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strengtl	Differential Pair
clk	Input	PIN_L1	2	B2_N1	PIN_L1	3.3-V L...efault)		24mA (default)	
espera	Output	PIN_R20	6	B6_N0	PIN_R20	3.3-V L...efault)		24mA (default)	
p0	Input	PIN_R22	6	B6_N0	PIN_R22	3.3-V L...efault)		24mA (default)	
p1	Input	PIN_R21	6	B6_N0	PIN_R21	3.3-V L...efault)		24mA (default)	
reset_n	Input	PIN_T21	6	B6_N0	PIN_T21	3.3-V L...efault)		24mA (default)	
valid	Output	PIN_U22	6	B6_N1	PIN_U22	3.3-V L...efault)		24mA (default)	
<<new node>>									

Figura 7 – Asignación de pines

Resultados experimentales

Simulación

Antes de realizar la simulación es necesario crearse un archivo con extensión .vht en el que se indique los intervalos de tiempos y como cambia cada variable de entrada.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY Practica9_vhd_tst IS
END Practica9_vhd_tst;
ARCHITECTURE Practica9_arch OF Practica9_vhd_tst IS
SIGNAL clk : STD_LOGIC:='0';
SIGNAL espera : STD_LOGIC;
SIGNAL p0 : STD_LOGIC;
SIGNAL p1 : STD_LOGIC;
SIGNAL reset_n : STD_LOGIC;
SIGNAL valid : STD_LOGIC;
COMPONENT Practica9
PORT (
 clk : IN STD_LOGIC;
 espera : OUT STD_LOGIC;
 p0 : IN STD_LOGIC;
 p1 : IN STD_LOGIC;
 reset_n : IN STD_LOGIC;
 valid : OUT STD_LOGIC);
END COMPONENT;
BEGIN
i1 : Practica9
PORT MAP (
 clk => clk,
 espera => espera,
 p0 => p0,
 p1 => p1,
 reset_n => reset_n,
 valid => valid);
init : PROCESS
-- variable declarations
BEGIN
-- code that executes only once
WAIT;
END PROCESS init;
clk <= not clk after 50 ns;
always : PROCESS
BEGIN
 reset_n <= '0';
 p1 <= '1';
 p0 <= '1';
 wait for 100 ns;
 reset_n <= '1';
 wait for 10 ns;
 p0 <= '0'; -- Pulso p0
 wait for 200 ns;
 p0 <= '1';
 wait for 200 ns;
 p1 <= '0'; -- Pulso p1
 wait for 200 ns;
 p1 <= '1';
 wait for 300 ns;
 p1 <= '0'; -- Pulso p1
 wait for 200 ns;
 p1 <= '1';
 wait for 300 ns;
 p0 <= '0'; -- Fin de secuencia: pulso en p0
 wait for 200 ns;
 p0 <= '1';
 assert valid = '1'
 report "Error: No se activa la salida tras introducir una secuencia correcta"
 severity warning;
 -- Enlazamos con otra nueva secuencia
 wait for 200 ns;
 p1 <= '0'; -- Pulso p1
 wait for 200 ns;
 p1 <= '1';
 assert valid = '0'
 report "Error: No se desactiva la salida tras volver a pulsar otra tecla"
 severity warning;
 wait for 300 ns;
 p1 <= '0'; -- Pulso p1
 wait for 200 ns;
 p1 <= '1';
 wait for 300 ns;
 p0 <= '0'; -- Fin de secuencia: pulso en p0
 wait for 200 ns;
 p0 <= '1';
```

```
 79     wait for 300 ns;
 80     assert valid = '1'
 81     report "Error: No se activa la salida tras introducir una secuencia correcta"
 82     severity warning;
 83     -- Volvemos a introducir una nueva secuencia
 84     p0 <= '0'; -- Pulso p0
 85     wait for 200 ns;
 86     p0 <= '1';
 87     wait for 200 ns;
 88     p1 <= '0'; -- Pulso p1
 89     wait for 200 ns;
 90     p1 <= '1';
 91     wait for 300 ns;
 92     p1 <= '0'; -- Pulso p1
 93     wait for 200 ns;
 94     p1 <= '1';
 95     wait for 300 ns;
 96     p0 <= '0'; -- Fin de secuencia: pulso en p0
 97     wait for 200 ns;
 98     p0 <= '1';
 99     assert valid = '1'
100     report "Error: No se activa la salida tras introducir una secuencia correcta"
101     severity warning;
102     assert false
103     report "Fin de la simulacion"
104     severity failure;
105   END PROCESS always;
106   END Practica9_arch;
```

Figura 8 – Código vhd para la simulación

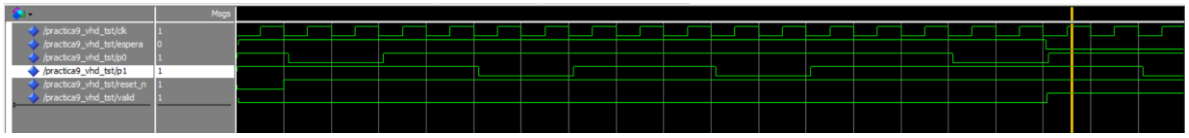

Figura 9 – Simulación de nuestro circuito (primeros tramos)

En la figura 9 se puede observar como cuando la secuencia se ha completado (donde tenemos colocada la línea amarilla), nuestra señal valid pasa a 1 mientras que la señal espera cambia a 0.

Implantación física

Tras haber realizado la simulación y haber comprobado que todo funcionaba correctamete, ya podemos implantar nuestro esquema en la placa y comprobar si nuestro circuito realiza la acción para la que lo hemos diseñado.

En la placa hemos utilizado, los pulsadores 0 y 1 como nuestras entradas p0 y p1, el pulsador 3 como nuestra entrada de reset, el led verde 0 para nuestra salida valid y el led rojo 0 para nuestra salida espera. Además, nuestra señal de reloj es una señal de 50 MHz de frecuencia.

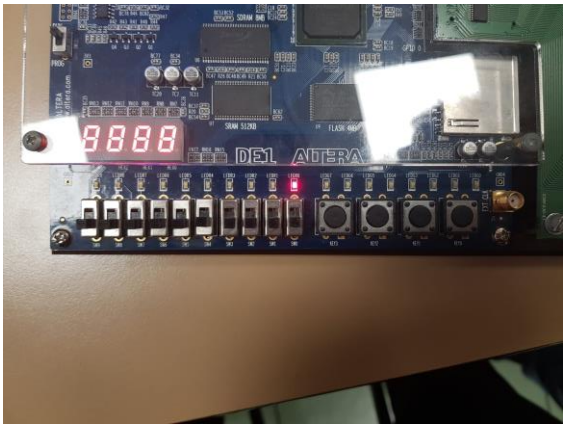

Figura 10 – Imagen de la salida cuando la secuencia no está completa

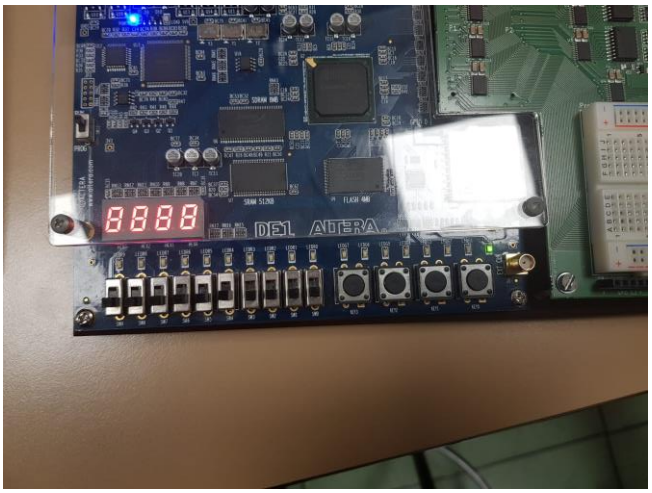

Figura 11 – Imagen de la salida cuando la secuencia se ha completado

Conclusiones y análisis de resultados

Al inicio de esta práctica teníamos tres objetivos principales. El primero era comprender el funcionamiento de las máquinas de estado. Tras haber realizado el diseño, incluido el diagrama de estados, y la programación de las dos máquinas de estado, tanto el detector de flanco como el detector de secuencia, podemos afirmar que hemos comprendido cómo funcionan estos componentes.

Nuestro segundo objetivo era diseñar y simular un detector de secuencia para una cerradura electrónica. Este era el objetivo principal de la práctica, por lo que como hemos conseguido realizar la simulación y la implantación física correctamente, podemos decir que hemos conseguido diseñar, simular e implantar en una placa FPGA un detector de secuencia y de esta forma demostrar que nuestro diseño funcionaba correctamente.

Nuestro tercer objetivo era aprender a describir circuitos secuenciales con lenguajes de descripción de hardware VHDL. Como se puede observar, hemos utilizado procesos que dependían de un reloj en todos nuestros diseños, tanto del detector de flanco como del detector de secuencia, por lo tanto hemos aprendido y dominado como describir circuitos secuenciales con VHDL.