



Memoria

Sistemas Electrónicos para la Automatización

Docente: Jorge Romero Sánchez

Trabajo Final de la asignatura: Análisis, Simulación y Síntesis de un PLC

Jorge Benavides Macías: jorge2@uma.es

Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Contenido

1	Especificaciones	4
2	Estudio del código - FSM Mealy	4
2.1	CKE_Gen.vhd	4
2.2	Reg_Des.vhd	4
2.3	Debouncer.vhd	5
2.4	Sincronizador.vhd	5
2.5	Mis_Tipos_PLC.vhd	5
2.6	MUX_PLC.vhd	5
2.7	Reg_PLC.vhd	5
2.8	FSM_PLC.vhd	5
2.8.1	Entidad	6
2.8.2	Arquitectura	6
2.9	Conclusión	9
3	Diseño del Test Bench	9
4	Simulación	11
4.1	Test Bench	11
4.2	Fichero de estímulos Mealy	11
4.3	Fichero CSV generado	12
4.4	Cronograma de simulación	12
5	Síntesis	13
6	Anexos	13
6.1	Test Bench completo	13



Lista de figuras

1	Esquemático RTL de la práctica	6
2	Esquemático RTL de Ecuación de Transición 1	7
3	Esquemático RTL de Ecuación de Transición 2	7
4	Esquemático RTL de Ecuación de Salida 1 Mealy	8
5	Esquemático RTL de Ecuación de Salida 2 Mealy	8
6	Diagrama de la FSM de Mealy	9
7	Fichero CSV Generado	12
8	Cronograma de simulación Mealy	13
9	Esquemático RTL de la práctica	13

Lista de códigos

1	Tabla de estado en Test Bench	10
2	Tabla de salida en Test Bench	11
3	Test Bench Completo	17

1. Especificaciones

- **Estudio previo:** Describid en detalle el PLC cuyo código se suministra, y que no puede ser modificado una vez seleccionado el estilo de FSM: Moore o Mealy, que elegisteis en la 2ª práctica.
- Como parámetros fijos el sistema tendrá $k_Max = 3$ entradas, $p_Max = 4$ salidas y $m_Max = 4$ biestables tal y como aparece en el package *Tipos_FSM_PLC*. O de otra forma, con independencia de la FSM que se programe, todos tendréis que simular y sintetizar a nivel RTL el PLC con $k=3(=k_Max)$ entradas aunque vuestra FSM de ejemplo tenga $k=1$ entrada; $p=4(=p_Max)$ salidas aunque vuestra FSM de ejemplo tenga $p=2$ salidas; y $m=4(=m_Max)$ biestables aunque vuestra FSM de ejemplo tenga $m=2$ biestables.
- Generar un único proyecto en Vivado que implemente dicho PLC.
- Para la fase de simulación diseñad un único Test Bench con manejo de ficheros (modelo en Tema 5) que ilustre el funcionamiento correcto del sistema con los parámetros fijos para todos $k=k_Max$, $p=p_Max$ y $m=m_Max$: para ello se programarán en las tablas, la FSM de la 2ª práctica “Diseño, Simulación y Síntesis de una FSM en VHDL” que os corresponda.
- Generad los rebotes en la señal de Trigger como si fuese un conmutador externo de Pull-Up desde un proceso. El único código VHDL que habrá que incluir en la memoria será el del Test Bench diseñado, comentado y correctamente tabulado: cuidad el estilo de programación.
- **Sintetizad** el PLC a nivel RTL (RTL Analysis) para $k=k_Max$, $p=p_Max$ y $m=m_Max$, para comprobar que efectivamente se corresponde con lo que representa.

2. Estudio del código - FSM Mealy

En esta sección se llevará a cabo un análisis detallado del código VHDL para la implementación de una máquina de estado finito (FSM) del tipo Mealy. Se describirán todas las instancias de los componentes y sus respectivos subcomponentes.

2.1. CKE_Gen.vhd

El CKE_Gen se utiliza para generar un pulso de reloj con una duración de un ciclo de reloj a partir de una señal lógica continua. Esta señal se utiliza a menudo como una señal de habilitación para controlar la operación de otros componentes en un sistema digital.

2.2. Reg_Des.vhd

Permite almacenar temporalmente los valores de entrada de una señal, lo que permite comparar valores consecutivos y determinar si un cambio de estado ha sido estable, típicamente se usa el valor de 32 bits como tamaño del filtro, pero es ajustable.

2.3. Debouncer.vhd

Este dispositivo elimina el “rebote” o “ruido” en una señal de entrada, típicamente asociada a dispositivos mecánicos como botones. Filtra estos cambios de estado no deseados, permitiendo que solo los cambios de estado estables sean detectados y procesados por el sistema.

2.4. Sincronizador.vhd

Este componente agrupa los anteriores para sincronizar la señal de entrada con el reloj de sistema, permitiendo que solo los cambios de estado estables sean detectados y procesados.

2.5. Mis_Tipos_PLC.vhd

El paquete *Tipos_FSM_PLC* define diferentes tipos de datos y constantes que se utilizarán en el diseño de una máquina de estado finito (FSM) para un sistema PLC.

- Se define la constante *N_Bits_Dato* para especificar la anchura del dato de la tabla.
- El tipo *Tabla_FSM* es un arreglo de vectores lógicos estándar (STD_LOGIC_VECTOR) que se utilizará para generar una ROM o un multiplexor.
- Se definen las constantes *k_Max*, *p_Max* y *m_Max* para especificar el número máximo de entradas, salidas y biestables, respectivamente.

Estos tipos de datos y constantes serán utilizados en el diseño de la entidad *FSM_PLC*, y estarán disponibles para su uso en el resto del código VHDL mediante la declaración *use work.Tipos_FSM_PLC.all*.

2.6. MUX_PLC.vhd

Como hemos visto en prácticas anteriores un multiplexor puede funcionar tanto como selector de señales o como dispositivo de memoria, en esta práctica se utiliza como un dispositivo de memoria, donde los datos se almacenan en una tabla ROM y se seleccionan a través de la señal de dirección *Direccion*.

2.7. Reg_PLC.vhd

Este registro se utiliza para almacenar una secuencia de bits y desplazarlos un bit a la vez a través de sus salidas.

2.8. FSM_PLC.vhd

La [Figura 1](#) es el RTL (*Register Transfer Level*) del PLC, se observa con facilidad que a nivel lógico el PLC no es complejo, cuenta con 6 módulos y una puerta lógica, sin embargo, la lógica o mejor dicho el código que hay detrás de cada módulo es complejo y hay que desgranarla poco a poco.

2.8.1. Entidad

La FSM tiene siete entradas y una salida; de las siete señales de entrada cuatro son señales de control (*clk*, *reset*, *cke*, *trigger*) y solo una de ella cuenta con un sincronizador, por lo tanto en simulación sería útil simular la pulsación de un botón mecánico.

Las otras señales de entrada son para introducir la tabla de estado, la x y la tabla de salida, correspondientes a la FSM que vamos a implementar.

La salida es un vector y este depende de las salidas y estados definidos.

2.8.2. Arquitectura

En la arquitectura del código se han definido una serie de señales internas para conectar los componentes y gestionar los distintos procesos de la arquitectura, además se han instanciado todos los componentes descritos en la sección anterior.

Procesos de verificaciones básicas de rangos y tiempos

El primer gran grupo de procesos tiene como finalidad comprobar tamaños de los datos y comprobar que se cumplen los tiempos de *Setup* y *Hold* y el ancho de pulso necesario, estos procesos ya se han hecho en otras prácticas y como sabemos no son sintetizables, pero a nivel de simulación son muy útiles.

Justo después en el código se instancian dos componentes, usando las señales internas definidas, y un proceso lógico para activar la FSM.

Ecuación de Transición de estado en los dos estilos: Mealy y Moore.

El proceso *asignacion_MUX* ajusta a la derecha con ceros de relleno, *m* bits del Estado a *N_Bits_Dato* del dato del MUX.

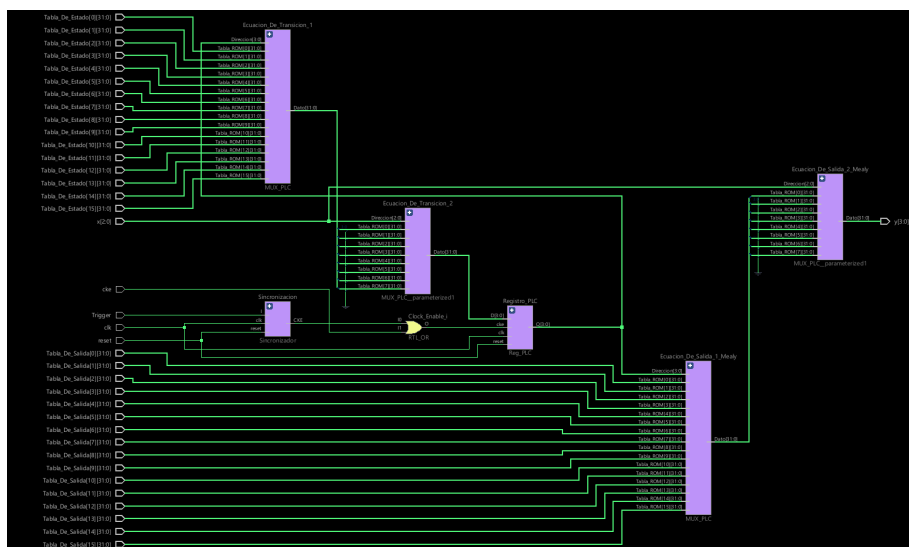


Figura 1: Esquemático RTL de la práctica

El primer módulo del esquemático se llama *ecuacion_de_transicion_1* (ver [Figura 2](#)), es un proceso dentro del código cuyo objetivo es seleccionar todos los posibles próximos estados a partir del actual. Es un multiplexor que funciona como memoria ROM y la información almacenada es la tabla de estado introducida por el puerto de entrada *Tabla_De_Estado*.

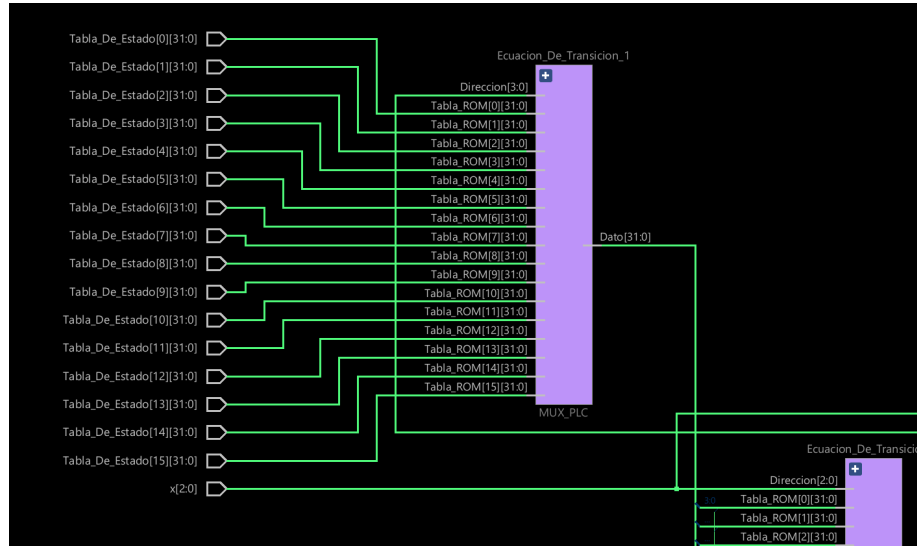


Figura 2: Esquemático RTL de Ecuación de Transición 1

El segundo módulo del esquemático se llama *ecuacion_de_transicion_2* (ver [Figura 3](#)), es un proceso dentro del código cuyo objetivo es seleccionar a partir de la entrada actual el próximo estado. El módulo también es un multiplexor que funciona como una memoria ROM que almacena los *Estados_con_formato*.

Estados con formato es una señal en la que se realiza una serie de asignaciones en el proceso *Asignacion_MUX*, este proceso divide una cadena de 32 bits en una columna de 8 filas con 4 bits por fila, esos 4 bits representan los posibles estados de la FSM.

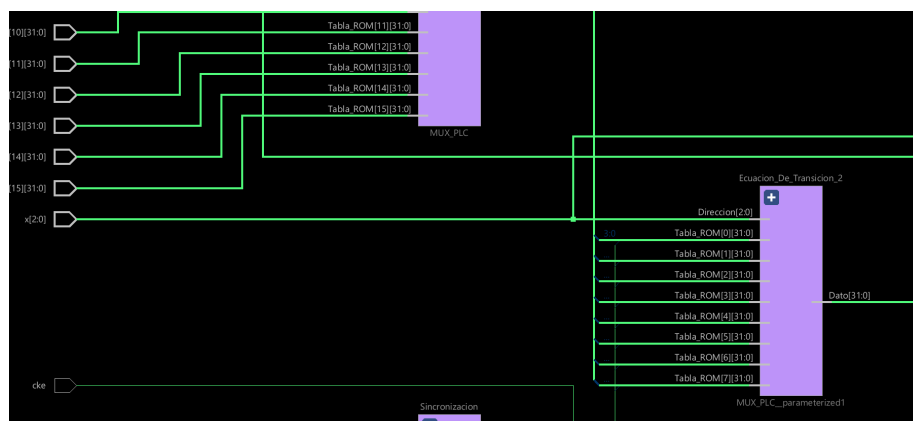


Figura 3: Esquemático RTL de Ecuación de Transición 2

Existe un proceso adyacente denominado *Asignacion_Proximo_Estado* que asigna a la señal *Proximo_Estado* el estado siguiente; este proceso al mismo tiempo ajusta a *N_Bits_Dato* el dato del MUX a *m* bits del estado.

El proceso *Ecuacion_De_Salida_1_Mealy* (ver [Figura 4](#)) es en esencia, otro multiplexor, en el que se selecciona

todas las posibles **salidas** a partir del estado actual.

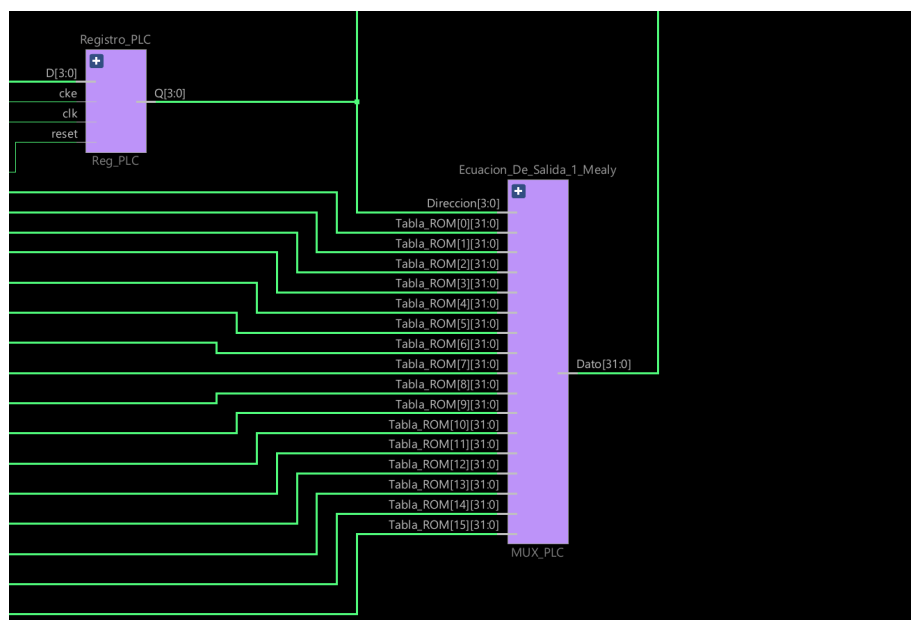


Figura 4: Esquemático RTL de Ecuación de Salida 1 Mealy

El proceso *Ecuacion_De_Salida_2_Mealy* (ver [Figura 5](#)) seleccionamos la salida a partir de la entrada. Este proceso incluye una señal llamada *Salidas_Con_Formato* que divide una cadena de 32 bits en una de 4 bits, consiguiendo 8 filas.

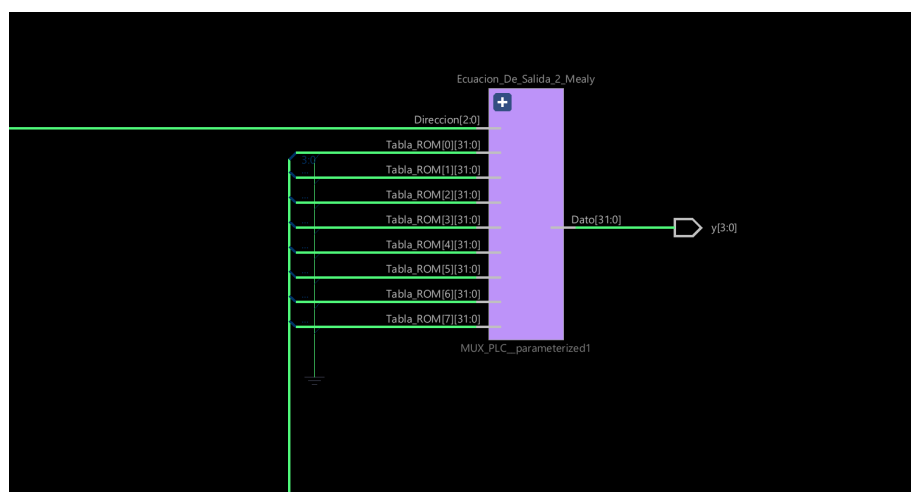


Figura 5: Esquemático RTL de Ecuación de Salida 2 Mealy

Para terminar existe un proceso más que asigna a *y* la *Salida*.

2.9. Conclusión

La FSM, ya sea de estilo Moore o Mealy, consta de dos etapas: selección y actualización del estado y selección y actualización de la salida. La primera etapa es común para ambas arquitecturas, mientras que la segunda depende del tipo de máquina que se implemente.

3. Diseño del Test Bench

Para realizar un banco de pruebas del PLC es primordial partir del diagrama de la FSM realizada. En mi práctica analicé un máquina de estados de tipo Mealy, es un “detector de secuencia” que si la secuencia es correcta el “jugador” ha ganado y la salida es un 1; la máquina cuenta con 6 estados, una entrada y una salida.

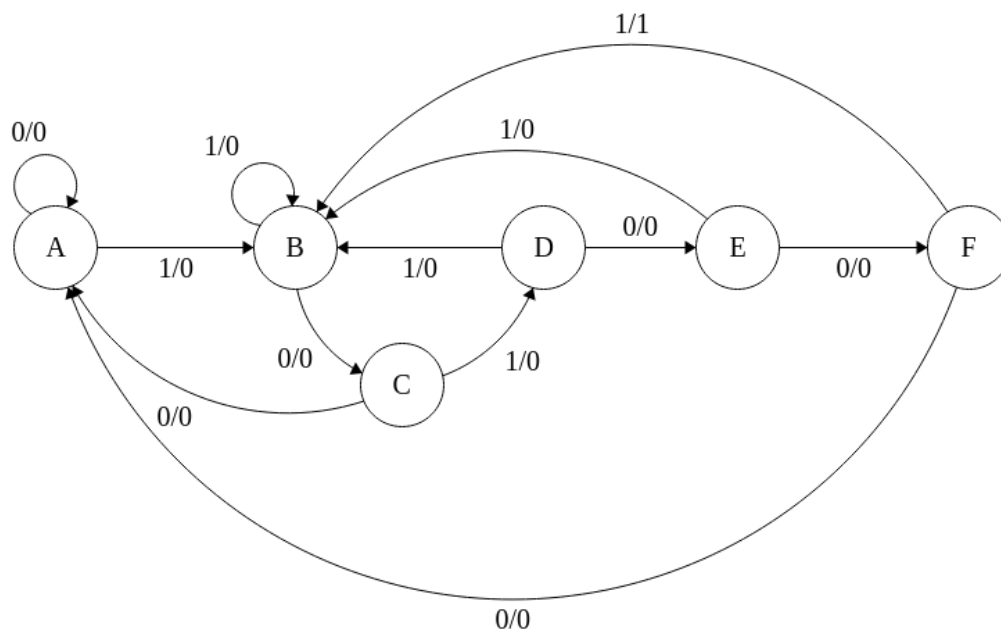


Figura 6: Diagrama de la FSM de Mealy

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**.

La tabla de estados posibles está descrita en el [Código 1](#), desde la línea 1 a la 13. La codificación que nos interesa es la $X(n)$ que establece el posible próximo estado según la entrada.

```

1  -----
2  -- MEALY
3  -----
4  -- Estos estados recogida del ejercicio 2
5  -- Tabla_De_estados_internos
6  -- | Z(n) | X (n) |
7  -- |      | 1 | 0 |
8  -- | 000 ( A ) | 0001 | 0000 |
9  -- | 001 ( B ) | 0001 | 0010 |
10 -- | 010 ( C ) | 0011 | 0000 |
11 -- | 011 ( D ) | 0001 | 0100 |
12 -- | 100 ( E ) | 0001 | 0101 |
13 -- | 101 ( F ) | 0001 | 0000 |
14
15 signal Tabla_De_estado_interno : Tabla_FSM(0 to 2**m_interno-1) :=
16   ((4 => '1', others => '0'), -- Estado 0
17    (4 => '1', others => '0'), -- Estado 1
18    (4|5 => '1', others => '0'), -- Estado 2
19    (4 => '1', 2 => '1', others => '0'), -- Estado 3
20    (4 => '1', 2 => '1', 0 => '1', others => '0'), -- Estado 4
21    (4 => '1', others => '0'),
22   -----
23   (others => '0'), -- Estado 5
24   (others => '0'), -- Estado 6
25   (others => '0'), -- Estado 7
26   (others => '0'), -- Estado 8
27   (others => '0'), -- Estado 9
28   (others => '0'), -- Estado 10
29   (others => '0'), -- Estado 11
30   (others => '0'), -- Estado 12
31   (others => '0'), -- Estado 13
32   (others => '0'), -- Estado 14
33   );

```

Código 1: Tabla de estado en Test Bench

La tabla de salidas posibles está descrita en el [Código 2](#), desde la línea 1 a la 8. La codificación que nos interesa es la $Y(n)$ que establece el valor de salida asociado a cada estado.

Como se puede observar en ambos códigos la transcripción en la tabla es literal y prácticamente directa.

```

1  -----
2  -- MEALY
3  -----
4  -- Tabla_De_Salida_interno
5  -- | Z(n) | Y(n) |
6  -- | 000 ( A ) | 0 |
7  -- | 001 ( B ) | 0 |
8  -- | 010 ( C ) | 0 |
9  -- | 011 ( D ) | 0 |
10 -- | 100 ( E ) | 0 |
11 -- | 101 ( F ) | 1 |
12

```

```
13 signal Tabla_De_Salida_interno : Tabla_FSM(0 to 2**m_interno-1) :=
14   ((others => '0'), -- Salida: 0
15   (others => '0'), -- Salida: 0
16   (others => '0'), -- Salida: 0
17   (others => '0'), -- Salida: 0
18   (others => '0'), -- Salida: 0
19   (0 => '1', others => '0'), -- Salida: 1
20   -----
21   (others => '0'), -- Salida: 0
22   (others => '0'), -- Salida: 0
23   (others => '0'), -- Salida: 0
24   (others => '0'), -- Salida: 0
25   (others => '0'), -- Salida: 0
26   (others => '0'), -- Salida: 0
27   (others => '0'), -- Salida: 0
28   (others => '0'), -- Salida: 0
29   (others => '0'), -- Salida: 0
30   (others => '0')
31 );
```

Código 2: Tabla de salida en Test Bench

4. Simulación

Luego de codificar las tablas de estado y de salida, el resto del banco de pruebas es trivial; generamos 4 procesos para las señales de control del PLC. Un *clk*, un *reset*, un *cke* a 1 durante toda la simulación y como he comentado antes un *trigger* que simule la pulsación de un botón mecánico.

4.1. Test Bench

El fichero de estímulos son los posibles valores de la *X*, los cuales representan una secuencia correcta y dos incorrectas, con estos estímulos puedo recorrer todo los estados posibles y sus diferentes transicciones.

4.2. Fichero de estímulos Mealy

#Fichero de Estímulos de práctica 3 de Mealy.

#Device Name: Discovery2NI

#Nombre: Izan Amador, Jorge Benavides

#Fecha: 11 de Enero de 2023.

#

Delay Time (ns) Input (I).

Wait for reset

40 ns 000

Secuencia correcta

20 ns 001

20 ns 000

20 ns 001

20 ns 000

20 ns 000

20 ns 001

```
# Secuencias aleatorias (erróneas)
```

```
# Primera secuencia
```

```
25 ns 000
```

```
25 ns 000
```

```
25 ns 001
```

```
25 ns 001
```

```
25 ns 000
```

```
25 ns 001
```

```
# Segunda secuencia
```

```
25 ns 001
```

```
25 ns 001
```

```
25 ns 000
```

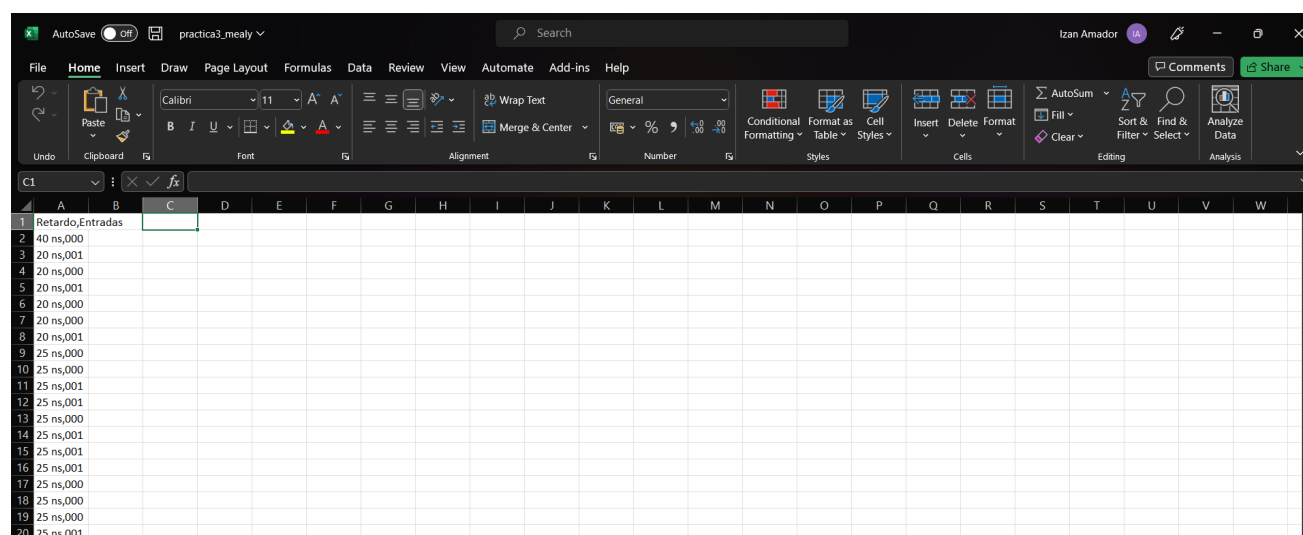
```
25 ns 000
```

```
25 ns 000
```

```
25 ns 001
```

4.3. Fichero CSV generado

El CSV generado por el banco de pruebas para una posible verificación con Analog Discovery.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	Retardo,Entradas																						
2	40 ns,000																						
3	20 ns,001																						
4	20 ns,000																						
5	20 ns,001																						
6	20 ns,000																						
7	20 ns,000																						
8	20 ns,001																						
9	25 ns,000																						
10	25 ns,000																						
11	25 ns,001																						
12	25 ns,001																						
13	25 ns,000																						
14	25 ns,001																						
15	25 ns,001																						
16	25 ns,001																						
17	25 ns,000																						
18	25 ns,000																						
19	25 ns,000																						
20	25 ns,001																						

Figura 7: Fichero CSV Generado

4.4. Cronograma de simulación

El cronograma resultante solo tiene una secuencia correcta, que coincide con el fichero de estímulos y que además es el único código que pasa por todos los estados disponibles tal y como se ha definido en la [Figura 8](#). La simulación ha sido un éxito.

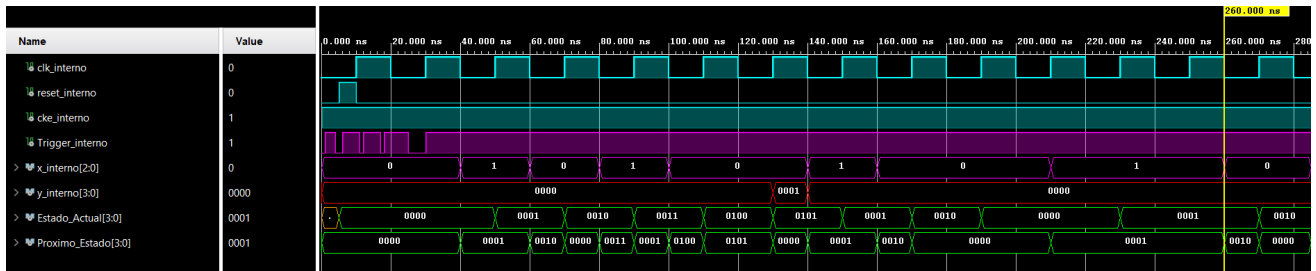


Figura 8: Cronograma de simulación Mealy

5. Síntesis

La síntesis del dispositivo ha generado un RTL esperado y comentado en secciones anteriores, por lo tanto hemos completado el proceso de diseño con éxito.

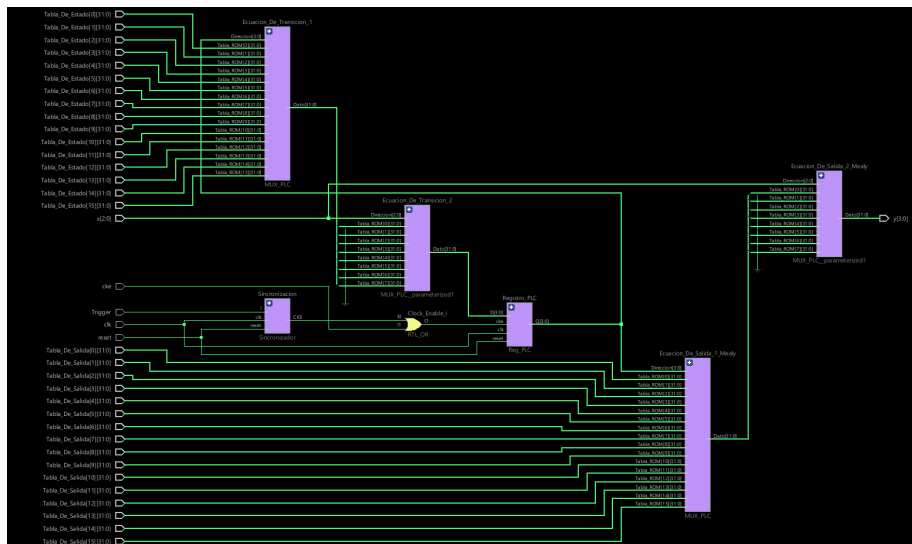


Figura 9: Esquemático RTL de la práctica

6. Anexos

6.1. Test Bench completo

```

1  -----
2  -- Company: Universidad de Málaga
3  -- Engineer: Izan Amador, Jorge L. Benavides
4  --
5  -- Create Date: 11-01-2023
6  -- Design Name: Test_Bench
7  -- Module Name: Test_Bench - Behavioral
8  -- Project Name: PLC
9  -- Target Devices: Zybo
10 -- Tool Versions: Vivado 2022.1
11 -- Description: Banco de pruebas para un PLC genérico.

```

```

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 use work.Tipos_FSM_PLC.all;   -- Se importa la librería Tipos_FMS_PLC
25                               -- para trabajar con el tipo Tabla_FSM,
26                               -- el cual es un array de 32 bits
27
28 entity Test_Bench is
29 -- En el banco de pruebas no se definen puertos.
30 end Test_Bench;
31
32 architecture Comportamiento of Test_Bench is
33     component FSM_PLC         -- Se instancia el componente FSM_PLC para realizar el test
34
35         generic(k             : natural := 32;    -- k entradas.
36                 p             : natural := 32;    -- p salidas.
37                 m             : natural := 32;    -- m biestables. (Hasta 16 estados)
38                 T_DM : time    := 10 ps;         -- Tiempo de retardo desde el cambio de dirección del MUX hasta la actualización de la salida Q.
39                 T_D  : time    := 10 ps;         -- Tiempo de retardo desde el flanco activo del reloj hasta la actualización de la salida Q.
40                 T_SU : time    := 10 ps;         -- Tiempo de Setup.
41                 T_H  : time    := 10 ps;         -- Tiempo de Hold.
42                 T_W  : time    := 10 ps);        -- Anchura de pulso.
43
44         port (x               : in  STD_LOGIC_VECTOR(k - 1 downto 0); -- x es el bus de entrada.
45               y               : out STD_LOGIC_VECTOR(p - 1 downto 0); -- y es el bus de salida.
46               Tabla_De_Estado : in  Tabla_FSM(0 to 2**m - 1); -- Contiene la Tabla de Estado estilo Moore: Z(n+1)=T1(Z(n),x(n))
47               Tabla_De_Salida : in  Tabla_FSM(0 to 2**m - 1); -- Contiene la Tabla de Salida estilo Moore: Y(n)=T2(Z(n))
48               clk              : in  STD_LOGIC; -- La señal de reloj.
49               cke               : in  STD_LOGIC; -- La señal de habilitación de avance: si vale '1' el autómata avanza a ritmo de clk y si vale '0' manda Trigger.
50               reset             : in  STD_LOGIC; -- La señal de inicialización.
51               Trigger           : in  STD_LOGIC); -- Señal de disparo (single shot) asíncrono y posiblemente con rebotes para hacer un avance único. Lleva un sincronizador.
52     end component FSM_PLC;
53
54
55     constant semiperiodo : time := 10 ns; -- Constante para el tiempo del reloj
56     constant k_interno   : natural := 3;  -- Número de entradas
57     constant p_interno   : natural := 4;  -- Número de salidas
58     constant m_interno   : natural := 4;  -- Número de biestables (16 estados
59                                         -- posibles
60
61     signal x_interno : std_logic_vector(k_interno - 1 downto 0) := (others => 'U'); -- Entrada
62     signal y_interno : std_logic_vector(p_interno - 1 downto 0) := (others => 'U'); -- Salida
63     signal reset_interno, cke_interno, Trigger_interno : std_logic := 'U'; -- Señales de control
64     signal clk_interno : std_logic := '0'; -- Señal del reloj
65
66     -----
67     -- MEALY
68     -----
69     -- Estos estados recogidos del ejercicio 2
70     -- Tabla_De_estados_internos
71     -- | Z(n) | X(n) |
72     -- |-----|
73     -- | 000 (A) | 0001 | 0000 |
74     -- | 001 (B) | 0001 | 0010 |
75     -- | 010 (C) | 0011 | 0000 |
76     -- | 011 (D) | 0001 | 0100 |
77     -- | 100 (E) | 0001 | 0101 |
78     -- | 101 (F) | 0001 | 0000 |
79
80     signal Tabla_De_Estado_interno : Tabla_FSM(0 to 2**m_interno-1) :=
81     ((4 => '1', others => '0'), -- Estado 0
82      (4 => '1', 1 => '1', others => '0'), -- Estado 1
83      (4|5 => '1', others => '0'), -- Estado 2
84      (4 => '1', 2 => '1', others => '0'), -- Estado 3
85      (4 => '1', 2 => '1', 0 => '1', others => '0'), -- Estado 4
86      (4 => '1', others => '0'),
87
88      (others => '0'), -- Estado 5
89      (others => '0'), -- Estado 6
90      (others => '0'), -- Estado 7

```

```

91     (others => '0'),           -- Estado 8
92     (others => '0'),           -- Estado 9
93     (others => '0'),           -- Estado 10
94     (others => '0'),           -- Estado 11
95     (others => '0'),           -- Estado 12
96     (others => '0'),           -- Estado 13
97     (others => '0')            -- Estado 14
98 );
99
100 -- Tabla_De_Salida_interno
101 -- |   Z(n)   | Y(n) |
102 -- | 000 ( A ) | 0   |
103 -- | 001 ( B ) | 0   |
104 -- | 010 ( C ) | 0   |
105 -- | 011 ( D ) | 0   |
106 -- | 100 ( E ) | 0   |
107 -- | 101 ( F ) | 1   |
108
109 signal Tabla_De_Salida_interno : Tabla_FSM(0 to 2**m_interno-1) :=
110 ((others => '0'),             -- Salida: 0
111  (others => '0'),             -- Salida: 0
112  (others => '0'),             -- Salida: 0
113  (others => '0'),             -- Salida: 0
114  (others => '0'),             -- Salida: 0
115  (0 => '1', others => '0'),    -- Salida: 1
116  -----
117  (others => '0'),             -- Salida: 0
118  (others => '0'),             -- Salida: 0
119  (others => '0'),             -- Salida: 0
120  (others => '0'),             -- Salida: 0
121  (others => '0'),             -- Salida: 0
122  (others => '0'),             -- Salida: 0
123  (others => '0'),             -- Salida: 0
124  (others => '0'),             -- Salida: 0
125  (others => '0'),             -- Salida: 0
126  (others => '0')             -- Salida: 0
127 );
128
129 begin
130
131     DUT : FSM_PLC             -- Device Under Test
132     generic map (             -- Asociación de constantes definidas en la arquitectura
133         k => k_interno,
134         p => p_interno,
135         m => m_interno)
136     port map(                 -- Asociación de los puertos con sus respectivas señales
137         -- para simulación
138         x => x_interno,
139         y => y_interno,
140         Tabla_De_Estado => Tabla_De_Estado_interno,
141         Tabla_De_Salida => Tabla_De_Salida_interno,
142         Trigger => Trigger_interno,
143         reset    => reset_interno,
144         clk      => clk_interno,
145         cke      => cke_interno);
146
147 -- Extraído de The Student Guide to VHDL, Peter J.Asheden
148 clock_gen : process (clk_interno) is -- Reloj del sistema
149 begin
150     if clk_interno = '0' then
151         clk_interno <= '1' after semiperiodo,
152         '0' after 2*semiperiodo;
153     end if;
154 end process clock_gen;
155
156 cke_interno <= '1'; -- Habilitación del PLC
157
158 trigger: process             -- Simulación de rebotes como si fuese un conmutador externo
159 begin
160     trigger_interno <= '0';
161     wait for 1 ns;
162     trigger_interno <= '1';
163     wait for 3 ns;
164     trigger_interno <= '0';
165     wait for 2 ns;
166     trigger_interno <= '1';
167     wait for 5 ns;
168     trigger_interno <= '0';
169     wait for 1 ns;

```

```

170 trigger_interno <= '1';
171 wait for 5 ns;
172 trigger_interno <= '0';
173 wait for 1 ns;
174 trigger_interno <= '1';
175 wait for 7 ns;
176 trigger_interno <= '0';
177 wait for 5 ns;
178 trigger_interno <= '1';
179 wait;
180 end process trigger;
181
182 reset : process          -- Señal de reset
183 begin
184     reset_interno <= '0';
185     wait for 5 ns;
186     reset_interno <= '1';
187     wait for 5 ns;
188     reset_interno <= '0';
189     wait;
190 end process reset;
191
192 Estimulos_Desde_Fichero : process
193
194     file Input_File : text;
195     file Output_File : text;
196
197     variable Input_Data : BIT_VECTOR(k_interno-1 downto 0) := (OTHERS => '0');    -- La entrada del fichero de estímulos está condicionada al tamaño de la
198                                                                                      -- entrada del PLC, que en este caso es de k_interno
199     variable Delay : time := 0 ms;
200     variable Input_Line : line := NULL;
201     variable Output_Line : line := NULL;
202     variable Std_Out_Line : line := NULL;
203     variable Correcto : Boolean := True;
204     constant Coma : character := ',';
205
206     begin
207         -- practica3_Estimulos.txt contiene los estímulos y los tiempos de retardo para el semisumador.
208         file_open(Input_File, "C:\Users\izana\Documents\GitHub\SEA\Estimulos\practica3_mealy_Estimulos.txt", read_mode);
209         --file_open(Input_File, "C:\Users\izana\Documents\GitHub\SEA\Estimulos\practica3_Estimulos.txt", read_mode);
210
211         -- practica_Estimulos.csv contiene los estímulos y los tiempos de retardo para el Analog Discovery 2.
212         file_open(Output_File, "C:\Users\izana\Documents\GitHub\SEA\CSV\practica3_mealy.csv", write_mode);
213         -- file_open(Output_File, "C:\Users\izana\Documents\GitHub\SEA\CSV\practica3.csv", write_mode);
214
215         -- Titles: Son para el formato EXCEL *.CSV (Comma Separated Values):
216         write(Std_Out_Line, string'("Retardo"), right, 7);
217         write(Std_Out_Line, Coma, right, 1);
218         write(Std_Out_Line, string'("Entradas"), right, 8);
219
220         Output_Line := Std_Out_Line;
221
222         writeline(output, Std_Out_Line);
223         writeline(Output_File, Output_Line);
224
225         while (not endfile(Input_File)) loop
226
227             readline(Input_File, Input_Line);
228
229             read(Input_Line, Delay, Correcto); -- Comprobación de que se trata de un texto que representa
230             -- el retardo, si no es así leemos la siguiente línea.
231             if Correcto then
232
233                 read(Input_Line, Input_Data); -- El siguiente campo es el vector de pruebas.
234                 x_interno <= TO_STDLOGICVECTOR(Input_Data);
235                 -- De forma simultánea lo volcaremos en consola en csv.
236                 write(Std_Out_Line, Delay, right, 5); -- Longitud del retardo, ej. "20 ms".
237                 write(Std_Out_Line, Coma, right, 1);
238                 write(Std_Out_Line, Input_Data, right, 2); --Longitud de los datos de entrada.
239
240                 Output_Line := Std_Out_Line;
241
242                 writeline(output, Std_Out_Line);
243                 writeline(Output_File, Output_Line);
244
245                 wait for Delay;
246             end if;
247         end loop;
248

```




```
249     file_close(Input_File);           -- Cerramos el fichero de entrada.
250     file_close(Output_File);          -- Cerramos el fichero de salida.
251     wait;
252     end process Estimulos_Desde_Fichero;
253
254
255     end Comportamiento;
```

Código 3: Test Bench Completo