Instituto Politécnico Nacional

Centro de Investigación en Computación

Lenguajes de descripción de hardware

Práctica 5 - Máquinas de estados finitos

Profesor:

M. en C. Osvaldo Espinosa Sosa

Por:

Ing. Ricardo Aldair Tirado Torres

Ciudad de México, 2 de junio de 2024

Tabla de contenido

1. Objetivos **2**

2. FSM con flip-flops en estilo *One-Hot* **3**

3. FSM con flip-flops en estilo *One-Hot* modificado **5**

4. FSM por comportamiento (estilo *Minimal Bits*) **7**

5. FSM por comportamiento (estilo *One-Hot*) **10**

6. FSM con registros de corrimiento **13**

6.1. Dos registros de corrimiento 13

6.2. Un registro de corrimiento 13

7. Conclusiones **16**

8. Anexos **17**

8.1. Descripciones del hardware 17

8.2. Bancos de pruebas (*Test Benches*) 25

1. Objetivos

- Implementar una máquina de estados finitos, utilizando lenguajes de descripción de hardware.
- Utilizar las ecuaciones de los flip-flops para codificar en estilo *One-Hot* original y modificado, y observar la implementación de hardware del visor RTL.
- Conocer algunos de los estilos de codificación para máquinas de estados finitos, como *Minimal Bits* y *One-Hot*.
- Describir por comportamiento a una máquina de estados finitos y analizar, por medio del *State Machine Viewer*, como es que el software codifica los estados, así como el diagrama de estados arrojado por Quartus.
- Diseñar una máquina de estados finitos empleando otras técnicas diferentes a los estilos de codificación y ecuaciones de flip-flops. Diferenciar la implementación de hardware entre este diseño y los anteriores.

2. FSM con flip-flops en estilo *One-Hot*

Actividad 1

Describir una FSM que resuelva el problema de detectar una secuencia de cuatro unos seguidos o cuatro ceros seguidos incluso secuencias traslapadas. Usar el estilo *"ONE - HOT"* utilizando 9 flip-flops para el estado:

Estado	y8	y7	y6	y5	y4	y3	y2	y1	y0
A	0	0	0	0	0	0	0	0	1
B	0	0	0	0	0	0	0	1	0
C	0	0	0	0	0	0	1	0	0
D	0	0	0	0	0	1	0	0	0
E	0	0	0	0	1	0	0	0	0
F	0	0	0	1	0	0	0	0	0
G	0	0	1	0	0	0	0	0	0
H	0	1	0	0	0	0	0	0	0
I	1	0	0	0	0	0	0	0	0

Diseñar utilizando las ecuaciones de los flip-flops. Compilar y simular. Observar el resultado en el visor RTL.

La visualización RTL de la máquina de estados finitos o FSM (siglas en inglés de *Finite-State Machine*), descrita en estilo *One-Hot*, se muestra en la Figura 1. La implementación en hardware utiliza a los 9 flip-flops descritos en el código y los interconecta de acuerdo a la ecuaciones descritas, haciendo uso de múltiples compuertas lógicas. Las simulaciones se visualizan en la Figura 2. Se observa que primero detecta 4 ceros consecutivos y pone en alto a S, aunque W inicia con "01". Después de restablecer el sistema (utilizando a RST), la FSM detecta 4 unos consecutivos y pone nuevamente en alto a S, para después volver a ponerla en bajo, ya que la secuencia continúa con "01".

En los Anexos se localiza la descripción de la FSM con estilo *One-Hot*. Además de las entradas y salida, se declararon 9 flip-flops y con una lista sensible a los flancos de subida de CLK y de bajada de RST, se describió el comportamiento de cada flip-flop, de acuerdo a su ecuación obtenida. Al momento de restablecer el sistema, se inicializa a cada flip-flop, tomando el estilo *One-Hot*.

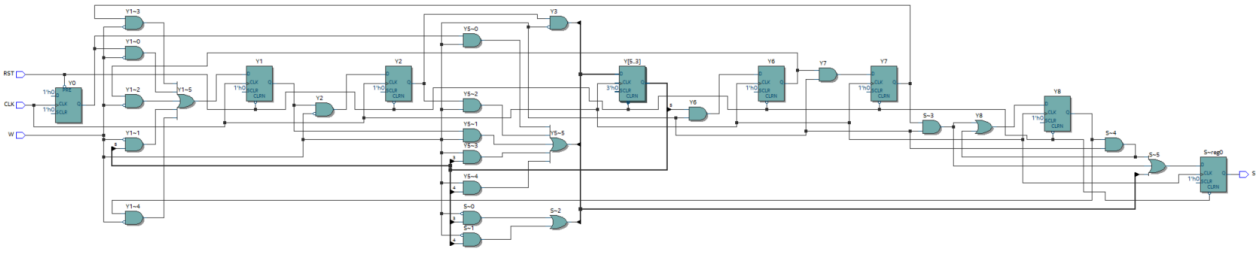

Figura 1: Diagrama RTL de la FSM, descrita con las ecuaciones de los flip-flops codificados en *One-Hot*.

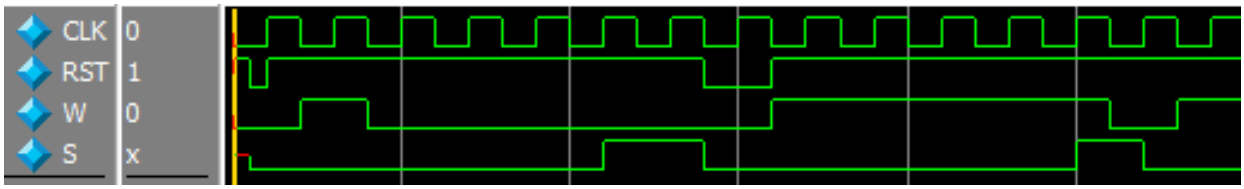

Figura 2: Simulación de la FSM, descrita con las ecuaciones de los flip-flops codificados en *One-Hot*, en el visor de formas de onda de ModelSim.

3. FSM con flip-flops en estilo *One-Hot* modificado

Actividad 2

Considerar la tabla "*ONE – HOT* modificada":

Estado	y8	y7	y6	y5	y4	y3	y2	y1	y0
A	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	1	1
C	0	0	0	0	0	0	1	0	1
D	0	0	0	0	0	1	0	0	1
E	0	0	0	0	1	0	0	0	1
F	0	0	0	1	0	0	0	0	1
G	0	0	1	0	0	0	0	0	1
H	0	1	0	0	0	0	0	0	1
I	1	0	0	0	0	0	0	0	1

Diseñar utilizando las ecuaciones de los flip-flops de la tabla modificada. Observar el resultado en el visor RTL. Compilar y simular.

La visualización RTL de la FSM, descrita en estilo *One-Hot* modificado, se muestra en la Figura 3. La implementación en hardware utiliza a los 9 flip-flops descritos en el código y los interconecta de acuerdo a la ecuaciones descritas, haciendo uso de múltiples compuertas lógicas. Las simulaciones se visualizan en la Figura 4. Se observa que primero detecta 4 ceros consecutivos y pone en alto a S, aunque W inicia con "01". Después de restablecer el sistema (utilizando a RST), la FSM detecta 4 unos consecutivos y pone nuevamente en alto a S, para después volver a ponerla en bajo, ya que la secuencia continúa con "01".

En los Anexos se localiza la descripción de la FSM con estilo *One-Hot* modificado. Además de las entradas y salida, se declararon 9 flip-flops y con una lista sensible a los flancos de subida de CLK y de bajada de RST, se describió el comportamiento de cada flip-flop, de acuerdo a su ecuación obtenida. Al momento de restablecer el sistema, se inicializa a cada flip-flop, tomando el estilo *One-Hot* modificado. En comparación con la Actividad 1, el circuito tiene el mismo funcionamiento e implementa la misma cantidad de flip-flops, unicamente cambia el estilo de codificación alterando un poco las ecuaciones del estilo original.

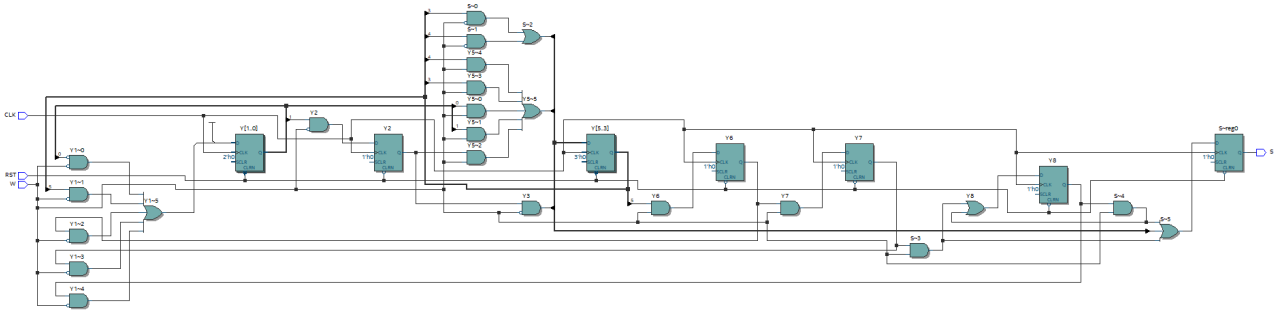

Figura 3: Diagrama RTL de la FSM, descrita con las ecuaciones de los flip-flops codificados en *One-Hot* modificado.

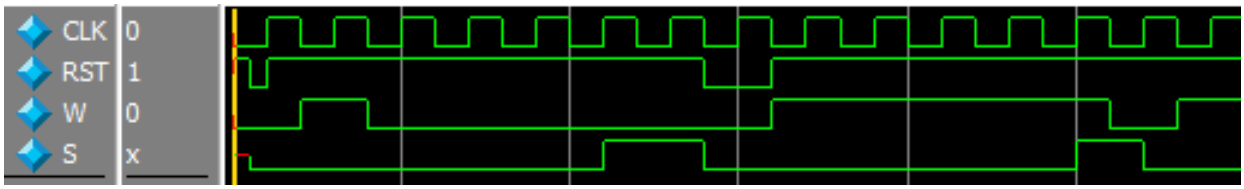

Figura 4: Simulación de la FSM, descrita con las ecuaciones de los flip-flops codificados en *One-Hot* modificado, en el visor de formas de onda de ModelSim.

4. FSM por comportamiento (estilo *Minimal Bits*)

Actividad 3

Describir por comportamiento la FSM del problema que está siendo considerado (como las descripciones vistas en clase). Indicar al compilador usar códigos del registro de estado en forma *"minimal bits"*. Compilar, simular y ver resultado en el visor RTL, ¿se genera el diagrama de estados y la tabla de códigos en los reportes? Para indicar al compilador qué

códigos del registro de estado debe usar se debe ir al menú de Quartus correspondiente:

Assignments > Settings > Analysis&sinthesis > StateMachineProcessing

La visualización RTL de la FSM, descrita por comportamiento en estilo *Minimal Bits*, se muestra en la Figura 5. En comparación a la Actividad 1 y 2, se implementan solo 2 compuertas lógicas OR, un multiplexor de 4 bits, un selector, un flip-flop D y dos registros de estados. En la Figura 6 se observa el diagrama de estados reportado por el *State Machine Viewer*, que corresponde con el diagrama visto en clase. Además, en la Figura 7 se tiene la tabla de códigos utilizada por el software de Quartus, que corresponde al estilo *One-Hot* modificado. Finalmente, en la Figura 8 se hallan todas las transiciones de los estados y que condición se debe dar para hacer el cambio del estado actual al estado destino.

Las simulaciones se visualizan en la Figura 9. El comportamiento es el mismo que el descrito en actividades anteriores.

En los Anexos se localiza la descripción de la FSM descrita por comportamiento y en estilo *Minimal Bits*. Además de las entradas y salida, se declararon a los 9 estados de la FSM como parámetros, así como dos registros, uno para almacenar el estado actual y otro para el siguiente estado. En una lista sensible se detectan a los flancos de subida de CLK y de bajada de RST y dentro de esta, empleando una estructura *case*, se describió el comportamiento de cada estado y el estado destino de acuerdo al valor de la señal entrada W.

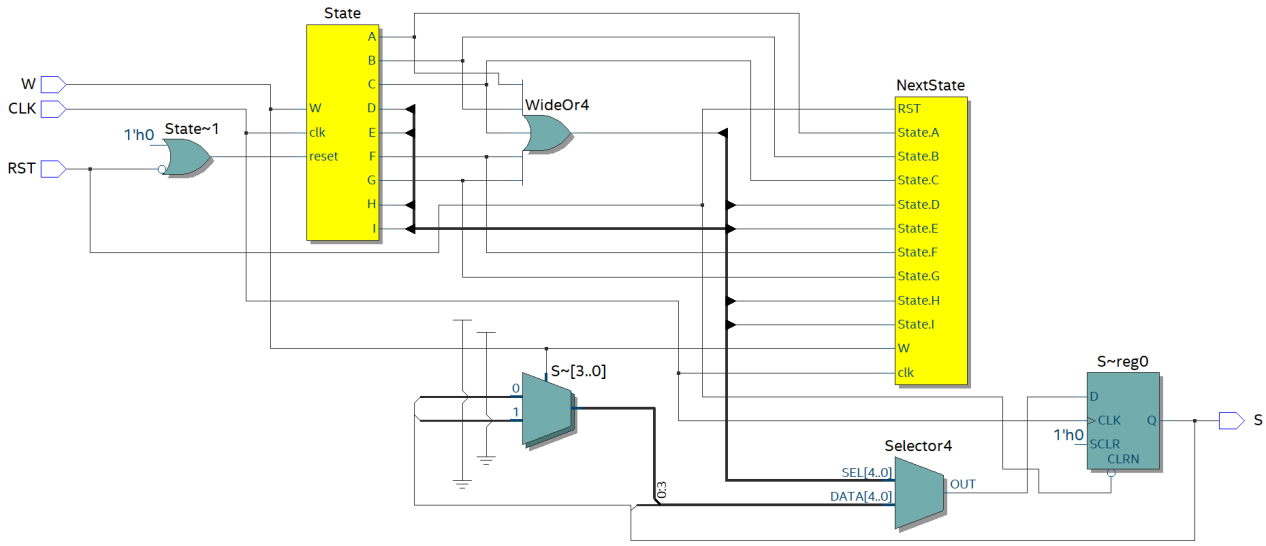

Figura 5: Diagrama RTL de la FSM, descrita por comportamiento en codificación *Minimal Bits*.

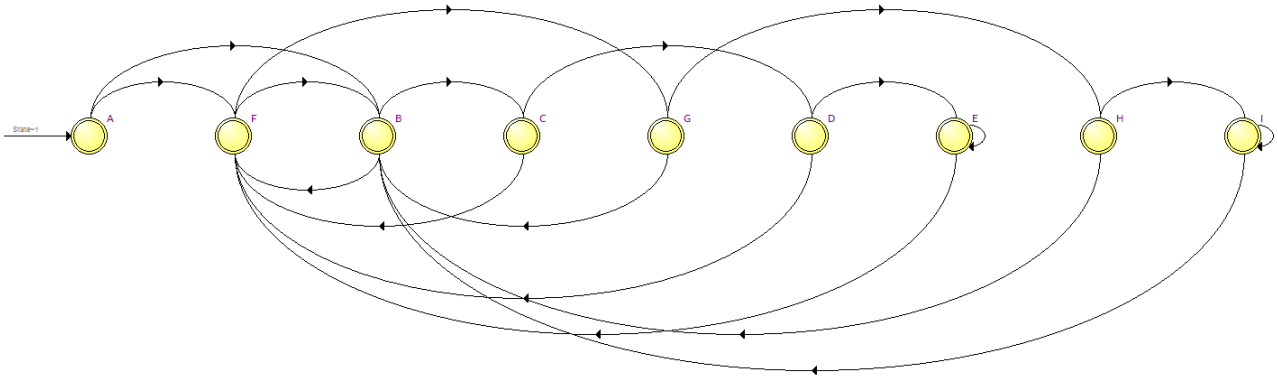

Figura 6: Diagrama de estados generado por el *State Machine Viewer* de la FSM estilo *Minimal Bits*.

	Name	I	H	G	F	E	D	C	B	A
1	A	0	0	0	0	0	0	0	0	0
2	B	0	0	0	0	0	0	0	1	1
3	C	0	0	0	0	0	0	1	0	1
4	D	0	0	0	0	0	1	0	0	1
5	E	0	0	0	0	1	0	0	0	1
6	F	0	0	0	1	0	0	0	0	1
7	G	0	0	1	0	0	0	0	0	1
8	H	0	1	0	0	0	0	0	0	1
9	I	1	0	0	0	0	0	0	0	1

Figura 7: Tabla de códigos (tipo *One-Hot* modificado) generada por el *State Machine Viewer* de la FSM estilo *Minimal Bits*.

	Source State	Destination State	Condition
1	A	B	(!W)
2	A	F	(W)
3	B	C	(!W)
4	B	F	(W)
5	C	D	(!W)
6	C	F	(W)
7	D	F	(W)
8	D	E	(!W)
9	E	F	(W)
10	E	E	(!W)
11	F	G	(W)
12	F	B	(!W)
13	G	H	(W)
14	G	B	(!W)
15	H	I	(W)
16	H	B	(!W)
17	I	I	(W)
18	I	B	(!W)

Figura 8: Tabla de transiciones generada por el *State Machine Viewer* de la FSM estilo *Minimal Bits*.

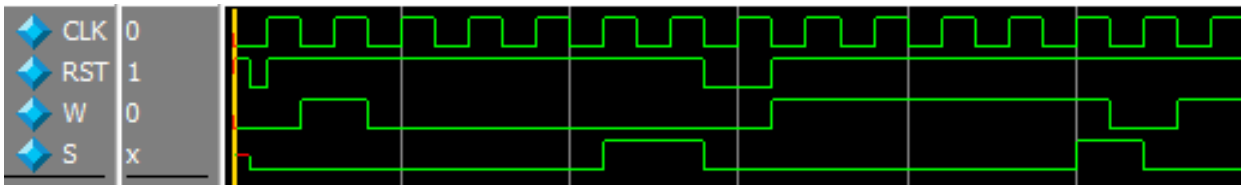

Figura 9: Simulación de la FSM estilo *Minimal Bits*, en el visor de formas de onda de ModelSim.

5. FSM por comportamiento (estilo *One-Hot*)

Actividad 4

Repetir el inciso 3 pero indicando codificación *"ONE - HOT"*. ¿Qué códigos *ONE-HOT* fueron utilizados, como los de la tabla normal o la modificada?

La visualización RTL de la FSM, descrita por comportamiento en estilo *One-Hot*, se muestra en la Figura 10. En comparación a la Actividad 1 y 2, se implementan solo 2 compuertas lógicas OR, un multiplexor de 4 bits, un selector, un flip-flop D y dos registros de estados. En la Figura 11 se observa el diagrama de estados reportado por el *State Machine Viewer*, que corresponde con el diagrama visto en clase. Además, en la Figura 12 se tiene la tabla de códigos utilizada por el software de Quartus, que corresponde al estilo *One-Hot* modificado. Finalmente, en la Figura 13 se hallan todas las transiciones de los estados y que condición se debe dar para hacer el cambio del estado actual al estado destino.

Las simulaciones se visualizan en la Figura 14. El comportamiento es el mismo que el descrito en actividades anteriores.

En los Anexos se localiza la descripción de la FSM descrita por comportamiento y en estilo *One-Hot*. La única diferencia de este código con el de la Actividad 3, es el tipo de codificación utilizada en parámetros, ya que se utilizó el *One-Hot*, no obstante, Quartus implementará siempre el estilo *One-Hot* modificado.

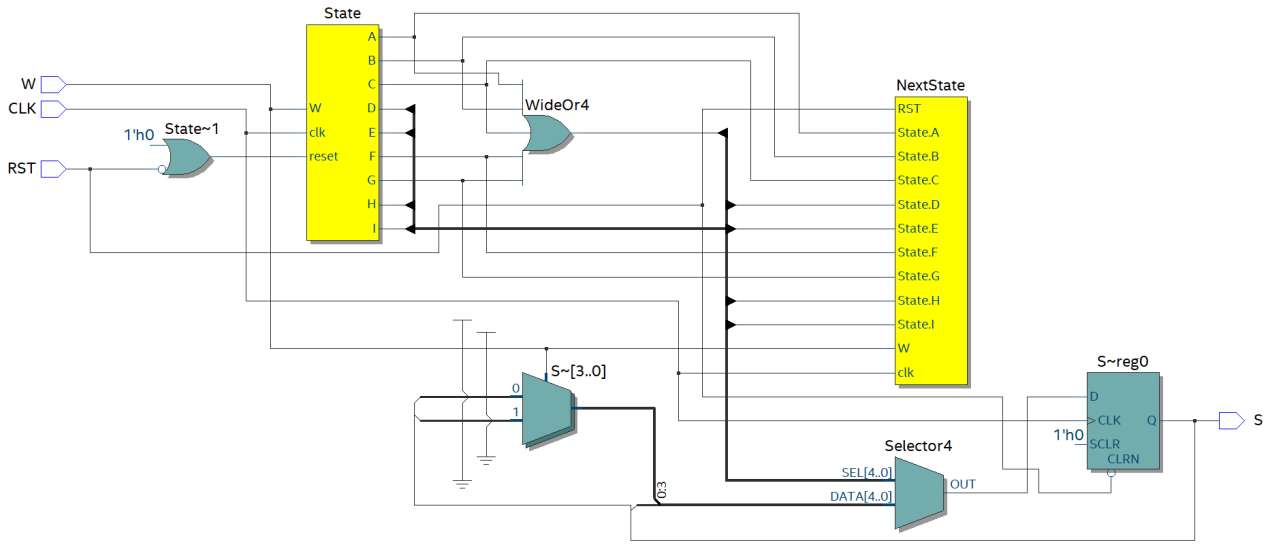

Figura 10: Diagrama RTL de la FSM, descrita por comportamiento en codificación *One-Hot*.

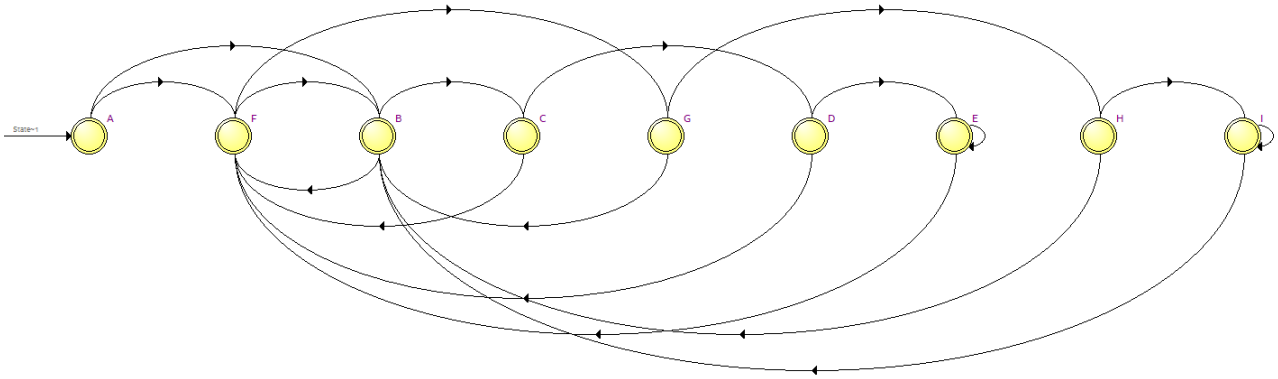

Figura 11: Diagrama de estados generado por el *State Machine Viewer* de la FSM estilo *One-Hot*.

	Name	I	H	G	F	E	D	C	B	A
1	A	0	0	0	0	0	0	0	0	0
2	B	0	0	0	0	0	0	0	1	1
3	C	0	0	0	0	0	0	1	0	1
4	D	0	0	0	0	0	1	0	0	1
5	E	0	0	0	0	1	0	0	0	1
6	F	0	0	0	1	0	0	0	0	1
7	G	0	0	1	0	0	0	0	0	1
8	H	0	1	0	0	0	0	0	0	1
9	I	1	0	0	0	0	0	0	0	1

Figura 12: Tabla de códigos (tipo *One-Hot* modificado) generada por el *State Machine Viewer* de la FSM estilo *One-Hot*.

	Source State	Destination State	Condition
1	A	B	(!W)
2	A	F	(W)
3	B	C	(!W)
4	B	F	(W)
5	C	D	(!W)
6	C	F	(W)
7	D	F	(W)
8	D	E	(!W)
9	E	F	(W)
10	E	E	(!W)
11	F	G	(W)
12	F	B	(!W)
13	G	H	(W)
14	G	B	(!W)
15	H	I	(W)
16	H	B	(!W)
17	I	I	(W)
18	I	B	(!W)

Figura 13: Tabla de transiciones generada por el *State Machine Viewer* de la FSM estilo *One-Hot*.

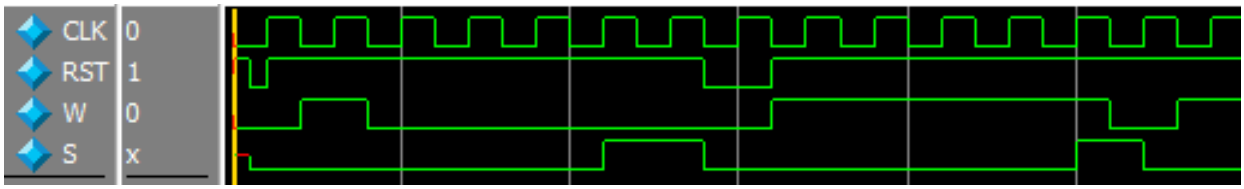

Figura 14: Simulación de la FSM estilo *One-Hot*, en el visor de formas de onda de ModelSim.

6. FSM con registros de corrimiento

Actividad 5

En ocasiones se puede resolver un problema utilizando un método diferente. ¿Se podrá resolver el problema utilizando dos registros de corrimiento de 4 bits?, ¿Uno para detectar los ceros y otros para detectar los unos consecutivos?, ¿Se podrá resolver con un solo registro de corrimiento? Compilar y simular.

6.1. Dos registros de corrimiento

La visualización RTL de la FSM, utilizando 2 registros de corrimiento, se muestra en la Figura 15. El circuito hace uso de dos registros de 3 bits (el bit faltante corresponde al valor de W), dos comparadores, una compuerta OR y un registro a la salida para S.

Las simulaciones se visualizan en la Figura 16. El comportamiento es el mismo que el descrito en actividades anteriores, siendo la única diferencia que se visualiza el corrimiento de bits en la simulación. Cuando el registro "Zeros" o "Ones" es exactamente igual a "1111", significa que hubo una secuencia de 4 ceros o 4 unos, consecutivos.

En los Anexos se localiza la descripción de la FSM utilizando 2 registros de corrimiento. Además de las entradas y salida, se declararon dos registros de 4 bits. En una lista sensible se detectan a los flancos de subida de CLK y de bajada de RST y dentro de esta se realiza el corrimiento de bits, de acuerdo al valor de la entrada W. Una vez que se detectan 4 valores consecutivos (ya sean unos o ceros), con una estructura *if*, se pone la salida S en alto, de lo contrario, S estará en bajo.

6.2. Un registro de corrimiento

La visualización RTL de la FSM, utilizando un registro de corrimiento, se muestra en la Figura 17. El circuito hace uso de un registro de 4 bits para corrimiento, un registro de 1 bit para la diferenciación de la secuencia de ceros y unos, dos multiplexores, un comparador y un registro a la salida para S.

Las simulaciones se visualizan en la Figura 18. El comportamiento es el mismo que el descrito en actividades anteriores, siendo la única diferencia que se visualiza el corrimiento de bits en la simulación. Cuando el registro "ZerosOnes" es exactamente igual a "1111", significa que hubo una secuencia de 4 ceros o 4 unos, consecutivos.

En los Anexos se localiza la descripción de la FSM utilizando un registro de corrimiento. Además de las entradas y salida, se declararon dos registros de 4 bits. En una lista sensible se detectan a los flancos de subida de CLK y de bajada de RST y dentro de esta se realiza el corrimiento de bits, de acuerdo al valor de la entrada W. El método difiere del circuito con 2 registros de corrimiento, debido a que, en lugar de usar un registro para detectar un tipo de secuencia, ahora se utiliza el mismo regreso para detectar ambas secuencias, siendo el corrimiento hacia la izquierda para detectar unos y hacia la derecha para detectar ceros. Una vez que se detectan 4 valores consecutivos (ya sean unos o ceros), con una estructura *if*, se pone la salida S en alto, de lo contrario, S estará en bajo.

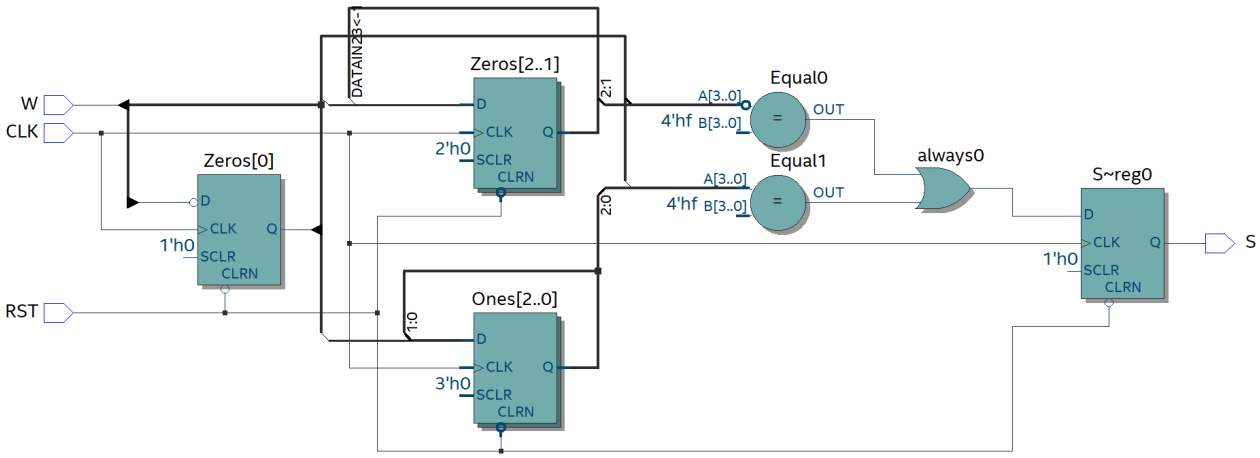

Figura 15: Diagrama RTL de la FSM, utilizando 2 registros de corrimiento.

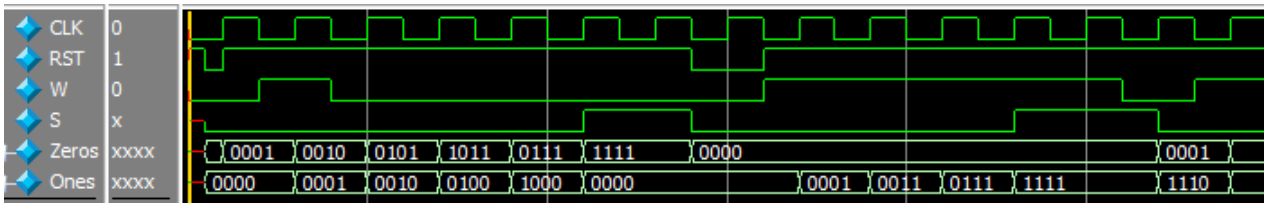

Figura 16: Simulación de la FSM, utilizando 2 registros de corrimiento, en el visor de formas de onda de ModelSim.

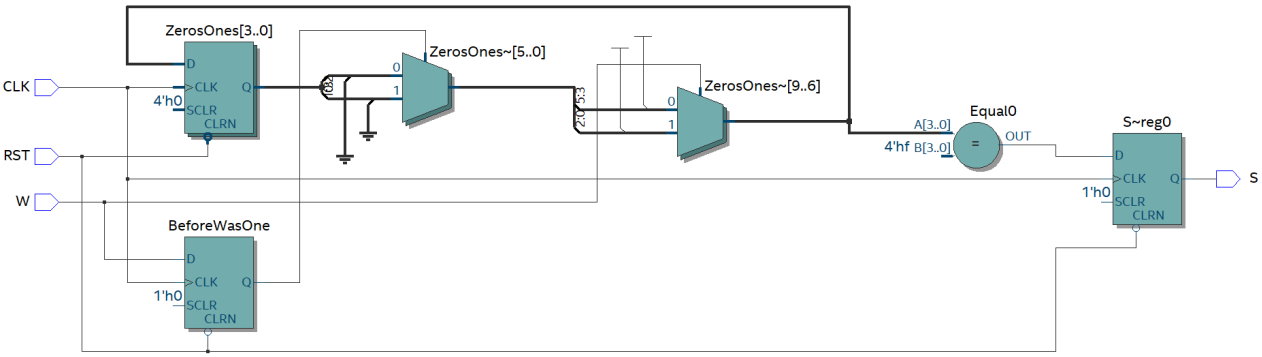

Figura 17: Diagrama RTL de la FSM, utilizando un solo registro de corrimiento.

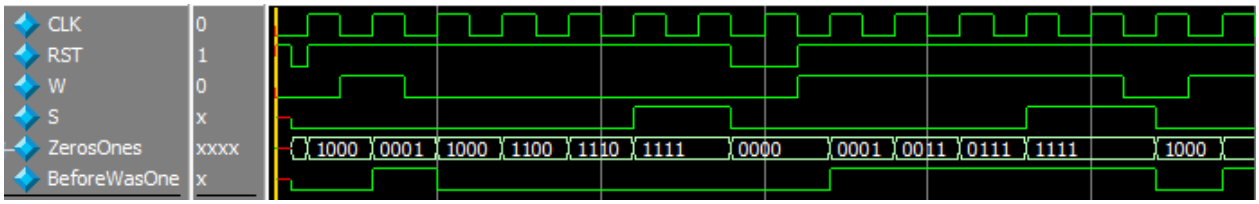

Figura 18: Simulación de la FSM, utilizando un solo registro de corrimiento, en el visor de formas de onda de ModelSim.

7. Conclusiones

En conclusión, se implementaron los 4 circuitos en lenguaje Verilog de manera exitosa.

Para los latches RS y D, se implementaron de manera correcta, usando la directiva *keep* para observar que, al mantenerse las señales internas en el módulo, se utilizan más compuertas lógicas, en el visor RTL, y más celdas lógicas, en el *Technology Map Viewer*. Esto puede resultar en una desventaja, ya que al realizar la simulación con retardos (modo lento de 85°C), se observa que los circuitos con directiva *keep*, son ligeramente más lentos.

Para el flip-flop D, se describió el módulo usando la configuración maestro-esclavo con 2 latches D, pero se realizó el mismo análisis que con los latches RS y D, concluyendo en que, las instancias mantienen las señales internas, generando más compuertas y celdas lógicas, y el retardo sigue siendo mayor cuando se emplea a la directiva *keep*.

Para el latch D y el flip-flop D, se observó con el *Chip Planner* que no se implementan en los mismos recursos, ya que el latch se implementa en la *Look Up Table* de un elemento lógico, mientras que el flip-flop se implementa de forma directa en el elemento lógico.

Con el uso de simulaciones con retardos, se visualizó que los circuitos implementados tienen una frecuencia máxima de operación, puesto que en el peor de los casos, la señal de salida va retrasando su respuesta a la señal de entrada

Finalmente, se asignaron los pines correspondientes en la placa de desarrollo para programar el dispositivo y realizar las pruebas pertinentes en hardware.

En los Anexos se pueden encontrar los códigos implementados junto con sus respectivos bancos de pruebas.

8. Anexos

8.1. Descripciones del hardware

```
module FSM_OneHot(
input    CLK, RST, W,
output  reg  S
);
reg Y0;
reg Y1;
reg Y2;
reg Y3;
reg Y4;
reg Y5;
reg Y6;
reg Y7;
reg Y8;
always @(posedge CLK, negedge RST)
begin
 if (!RST)
 begin
  Y0 <= 1;
  Y1 <= 0;
  Y2 <= 0;
  Y3 <= 0;
  Y4 <= 0;
  Y5 <= 0;
  Y6 <= 0;
  Y7 <= 0;
  Y8 <= 0;
  S  <= 0;
 end
 else
 begin
  Y0 <= 0;
  Y1 <= Y0 & ~W | Y5 & ~W | Y6 & ~W | Y7 & ~W | Y8 & ~W;
  Y2 <= Y1 & ~W;
  Y3 <= Y2 & ~W;
  Y4 <= Y3 & ~W | Y4 & ~W;
  Y5 <= Y0 &  W | Y1 &  W | Y2 &  W | Y3 &  W | Y4 &  W;
  Y6 <= Y5 &  W;
  Y7 <= Y6 &  W;
  Y8 <= Y7 &  W | Y8 & W;
  S  <= Y3 & ~W | Y4 & ~W | Y7 &  W | Y8 &  W;
```

```
 end
end
endmodule
```

Programa 1: Descripción en Verilog de la máquina de estados finitos, utilizando flip-flops codificados en *One-Hot*.

```
module FSM_OneHotM(
input    CLK, RST, W,
output  reg  S
);
reg Y0;
reg Y1;
reg Y2;
reg Y3;
reg Y4;
reg Y5;
reg Y6;
reg Y7;
reg Y8;
always @(posedge CLK, negedge RST)
begin
 if (!RST)
 begin
  Y0 <= 0;
  Y1 <= 0;
  Y2 <= 0;
  Y3 <= 0;
  Y4 <= 0;
  Y5 <= 0;
  Y6 <= 0;
  Y7 <= 0;
  Y8 <= 0;
  S  <= 0;
 end
 else
 begin
  Y0 <= 1;
  Y1 <= ~Y0 & ~W | Y5 & ~W | Y6 & ~W | Y7 & ~W | Y8 & ~W;
  Y2 <=  Y1 & ~W;
  Y3 <=  Y2 & ~W;
  Y4 <=  Y3 & ~W | Y4 & ~W;
  Y5 <=  Y0 &  W | Y1 &  W | Y2 &  W | Y3 &  W | Y4 &  W;
  Y6 <=  Y5 &  W;
  Y7 <=  Y6 &  W;
  Y8 <=  Y7 &  W | Y8 & W;
```

```
  S   <= Y3 & ~W | Y4 & ~W | Y7 &  W | Y8 &  W;
 end
end
endmodule
```

Programa 2: Descripción en Verilog de la máquina de estados finitos, utilizando flip-flops codificados en *One-Hot* modificado.

```
module FSM_Behavior_MB(
input    CLK, RST, W,
output  reg  S
);

parameter A = 4'b0001;
parameter B = 4'b0010;
parameter C = 4'b0011;
parameter D = 4'b0100;
parameter E = 4'b0101;
parameter F = 4'b0110;
parameter G = 4'b0111;
parameter H = 4'b1000;
parameter I = 4'b1001;
reg [3:0] State;
reg [3:0] NextState;

always @(posedge CLK, negedge RST)
begin
 if (!RST)
 begin
  State = A;
  S = 0;
 end
 else
 begin
  case(State)
  A:
   if(W)
    NextState = F;
   else
    NextState = B;
  B:
   if(W)
    NextState = F;
   else
    NextState = C;
  C:
```

```
    if(W)
     NextState = F;
    else
     NextState = D;
  D:
    if(W)
     NextState = F;
    else
    begin
     S = 1;
     NextState = E;
    end
  E:
    if(W)
    begin
     S = 0;
     NextState = F;
    end
    else
     NextState = E;
  F:
    if(W)
     NextState = G;
    else
     NextState = B;
  G:
    if(W)
     NextState = H;
    else
     NextState = B;
  H:
    if(W)
    begin
     S = 1;
     NextState = I;
    end
    else
     NextState = B;
  I:
    if(W)
     NextState = I;
    else
    begin
     S = 0;
     NextState = B;
```

```
   end
  endcase
  State = NextState;
 end
end
endmodule
```

Programa 3: Descripción en Verilog de la máquina de estados finitos, utilizando codificación *Minimal Bits*.

```
module FSM_Behavior_OneHot(
input    CLK, RST, W,
output  reg  S
);

parameter A = 9'b000000001;
parameter B = 9'b000000010;
parameter C = 9'b000000100;
parameter D = 9'b000001000;
parameter E = 9'b000010000;
parameter F = 9'b000100000;
parameter G = 9'b001000000;
parameter H = 9'b010000000;
parameter I = 9'b100000000;
reg [8:0] State;
reg [8:0] NextState;

always @(posedge CLK, negedge RST)
begin
 if (!RST)
 begin
  State = A;
  S = 0;
 end
 else
 begin
  case(State)
  A:
   if(W)
    NextState = F;
   else
    NextState = B;
  B:
   if(W)
    NextState = F;
   else
```

```
    NextState = C;
  C:
   if(W)
    NextState = F;
   else
    NextState = D;
  D:
   if(W)
    NextState = F;
   else
   begin
    S = 1;
    NextState = E;
   end
  E:
   if(W)
   begin
    S = 0;
    NextState = F;
   end
   else
    NextState = E;
  F:
   if(W)
    NextState = G;
   else
    NextState = B;
  G:
   if(W)
    NextState = H;
   else
    NextState = B;
  H:
   if(W)
   begin
    S = 1;
    NextState = I;
   end
   else
    NextState = B;
  I:
   if(W)
    NextState = I;
   else
   begin
```

```
    S = 0;
    NextState = B;
   end
  endcase
  State = NextState;
 end
end
endmodule
```

Programa 4: Descripción en Verilog de la máquina de estados finitos, utilizando codificación *One-Hot*.

```
module FSM_2ShiftRegister(
input    CLK, RST, W,
output reg   S
);

reg [3:0] Zeros;
reg [3:0] Ones;

always @(posedge CLK or negedge RST)
begin
 if (!RST)
 begin
  Zeros = 4'b0000;
  Ones  = 4'b0000;
  S = 0;
 end
 else
 begin
  Zeros = {Zeros[2:0], ~W};
  Ones  = {Ones[2:0], W};
  if (Zeros == 4'b1111 || Ones == 4'b1111)
   S = 1;
  else
   S = 0;
 end
end
endmodule
```

Programa 5: Descripción en Verilog de la máquina de estados finitos, utilizando 2 registro de corrimiento.

```
module FSM_1ShiftRegister(
input    CLK, RST, W,
output reg   S
```

```
);

reg  [3:0]  ZerosOnes;
reg    BeforeWasOne;

always @(posedge CLK or negedge RST)
begin
 if (!RST)
 begin
  ZerosOnes = 4'b0000;
  S = 0;
  BeforeWasOne = 0;
 end
 else
 begin
  if (W)
  begin
   if (BeforeWasOne)
    ZerosOnes = {ZerosOnes[2:0], 1'b1};
   else
   begin
    ZerosOnes = 4'b0001;
    BeforeWasOne = 1;
   end
  end
  else
   if (BeforeWasOne)
   begin
    ZerosOnes = 4'b1000;
    BeforeWasOne = 0;
   end
   else
    ZerosOnes = {1'b1, ZerosOnes[3:1]};
  if (ZerosOnes == 4'b1111)
   S = 1;
  else
   S = 0;
 end
end
endmodule
```

Programa 6: Descripción en Verilog de la máquina de estados finitos, utilizando 1 registro de corrimiento.

8.2. Bancos de pruebas (*Test Benches*)

```
`timescale 1 ns/ 1 ps
module FSM_OneHot_vlg_tst();
 reg CLK;
 reg RST;
 reg W;
 wire S;

 FSM_OneHot i1 (
  .CLK(CLK),
  .RST(RST),
  .S(S),
  .W(W)
 );

 initial
 begin
  CLK = 0; W = 0; RST = 1;
  #5; RST = 0;
  #5; RST = 1;
  #10;  W = 1;
  #20;  W = 0;
  #100; RST = 0;
  #20;  RST = 1; W = 1;
  #100; W = 0;
  #20;  W = 1;
  $display("Running testbench at CIC");
 end

 always
 begin
  #10; CLK = ~CLK;
 end

endmodule
```

Programa 7: Banco de prueba para el Programa 1, Programa 2, Programa 3, Programa 4, Programa 5 y Programa 6.