CAPÍTULO II

MÁQUINAS DE ESTADOS Y SU CONSTRUCCIÓN

2.1 MÁQUINAS DE ESTADOS

El modelo de máquina de estados contiene los elementos necesarios para describir la conducta de un sistema en términos de entradas, salidas y del tiempo.

El siguiente diagrama presenta el modelo general de una Máquina de Estados.

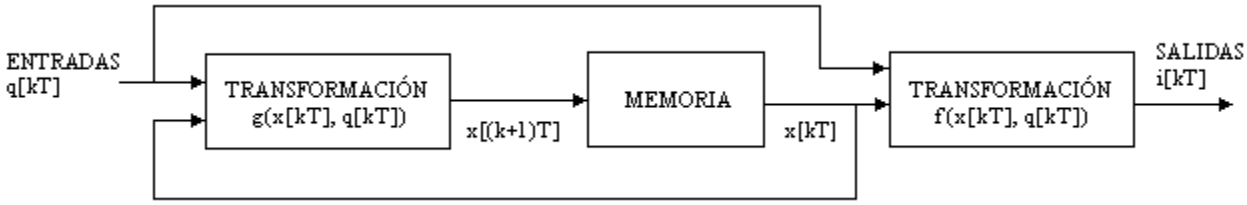


Figura. 2.1. Modelo general de una máquina de estados.

x[kT], representa el estado en el tiempo kT.
x[(k+1)T] = g(x[kT] , q[kT]), representa el siguiente estado.
q[kT], representa las entradas en el tiempo kT.
i[kT] = f(x[kT] , q[kT]), representa las salidas en el tiempo kT.

En donde T es el período de duración de cada estado y k es un contador entero.

2.1.1 EL ALGORITMO DE LA MÁQUINA DE ESTADOS

El algoritmo de la máquina de estados juega un papel muy importante en el diseño de sistemas digitales. Para circuitos síncronos la técnica de la carta ASM (Algorithm State Machine / Algoritmo de la Máquina de Estados) es la mejor notación, por lo tanto, se adoptará para el resto de la obra.

También existen otras técnicas como la de los diagramas de estados que son diagramas muy parecidos a las cartas ASM. Los diagramas de estados muestran gráficamente la secuencia de estados en un sistema y las condiciones de cada estado y de las transiciones entre cada uno de ellos.

Como ejemplo, en la figura 2.2 se describe el comportamiento de un contador binario de 3 bits mediante un diagrama de estados. Este circuito en particular no tiene ninguna entrada aparte de la de reloj, y ninguna otra salida más que las que se toman en cada flip-flop del contador.

2.1.2 ESTADOS Y RELOJ

El algoritmo de la máquina de estados se mueve a través de una secuencia de estados con base en la posición del estado presente y las variables de entrada. Los tiempos del estado están determinados por un reloj maestro.

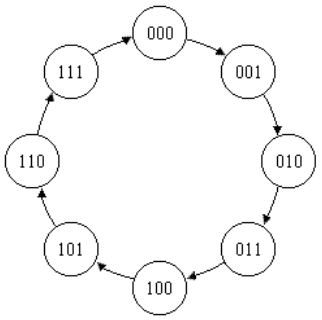


Figura. 2.2. Diagrama de estados para un contador binario de 3 bits.

2.2 NOTACIÓN DE LA CARTA ASM

2.2.1 REPRESENTACIÓN DE ESTADOS

El estado de una máquina de estados es la memoria de la historia pasada, suficiente para determinar las condiciones futuras. En la siguiente figura se muestra la representación del estado. Un estado se representa con un rectángulo y con su nombre simbólico en el extremo superior, encerrado en un círculo.

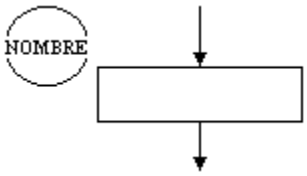


Figura 2.3. Representación del estado.

2.2.2 REPRESENTACIÓN DE DECISIONES

Las decisiones permiten seleccionar el camino que el algoritmo de la máquina de estados debe tomar de acuerdo a la variable o variables de entrada evaluadas. Las decisiones se representan mediante un rombo con el nombre de la variable a probar o una función que evalúe varias variables.

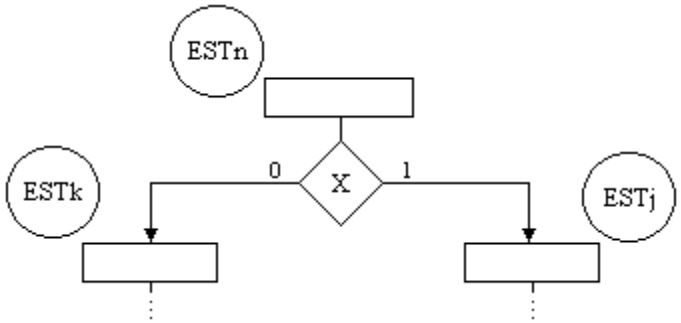


Figura 2.4. Representación de las decisiones.

2.2.3 REPRESENTACIÓN DE SALIDAS

Salidas no condicionales. Sirven para indicar la activación de una variable de salida. Para representarlas, se escriben dentro del rectángulo de estado, los nombres de las variables de salida que se activan en ese estado. Las salidas no condicionales no dependen de las condiciones de entrada, sólo dependen del estado actual. La figura 2.5 muestra la activación de las salidas VAR1 y VAR2 en el estado EST1.

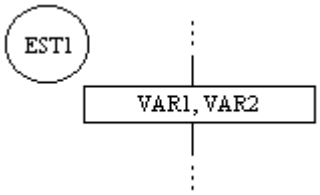


Figura 2.5. Representación de las salidas no condicionales.

Salidas Condicionales. Estas salidas se presentan solamente cuando ciertas condiciones de entrada existen. Se representan con un óvalo y los nombres de las salidas dentro de él.

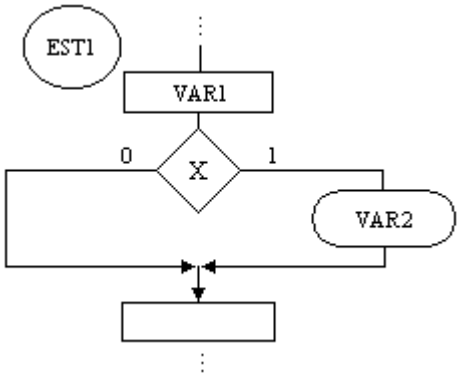


Figura 2.6. Representación de las salidas condicionales.

Para este ejemplo, si en el estado EST1 la variable de entrada X vale uno, entonces las salidas VAR1 y VAR2 serán activadas. Si X es igual a cero solamente se activará VAR1.

2.3 EJEMPLOS DE CARTAS ASM

A continuación se presentan algunos ejemplos para aclarar las ideas antes expuestas.

1) Diseñe un dispositivo que genere cierta secuencia binaria sólo cuando la variable INICIO sea igual a uno. Además esta secuencia dependerá del valor de la entrada X. Si X=0 la secuencia

binaria que se genera es la siguiente: 11, 10, 01, por el contrario, si X=1 la secuencia es: 01, 10, 11. Considere que cada pareja binaria se genera con un ciclo de reloj de diferencia.

Cuando se hace el diseño digital de un sistema es necesario hacer un diagrama de bloques que clarifique cuáles son las señales de entrada, cuáles las señales de salida, quién es el controlador y qué se está controlando.

Para este ejemplo, las señales de entrada son INICIO y X, y las señales de salida son las que representan la secuencia que se quiere generar, nombrémoslas VAR1 y VAR0.

El diagrama de bloques queda de la siguiente manera.

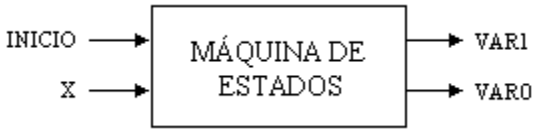


Figura 2.7. Diagrama de bloques para el ejemplo 1.

Y la carta ASM para esta máquina de estados se muestra en la figura 2.8.

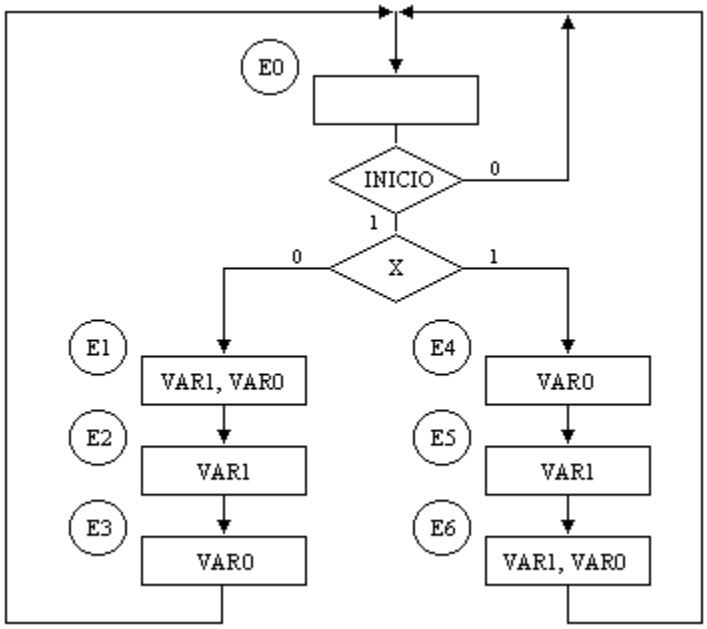


Figura 2.8. Carta ASM para el ejemplo 1.

De la figura 2.8 se puede observar que para activar una señal de salida incondicional, en un estado en particular, es necesario colocar su nombre dentro del rectángulo del estado.

2) Convertir el siguiente código en lenguaje ‘C’ a una carta ASM.

```
for (x = a; x ≤ b; x = x + c) {
      var1 = 1;  var2 = 0;
}
var1 = 0;
```

El diagrama de bloques de este sistema es el siguiente.

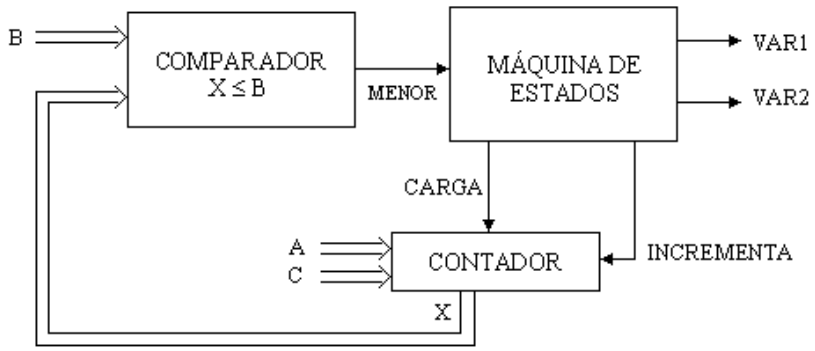


Figura 2.9. Diagrama de bloques para el ejemplo 2.

Se utiliza un contador para cargar el valor inicial de X o incrementar su valor en C unidades. La activación de la señal CARGA inicializará el valor de X con A, mientras que la activación de la señal INCREMENTA incrementará el valor de X en C unidades. También se cuenta con un comparador que evalúa la condición X ≤ B. Si X es menor o igual a B, el resultado es la activación de la señal MENOR, en caso contrario, la señal MENOR permanece en cero.

El algoritmo de la máquina de estados es el siguiente.

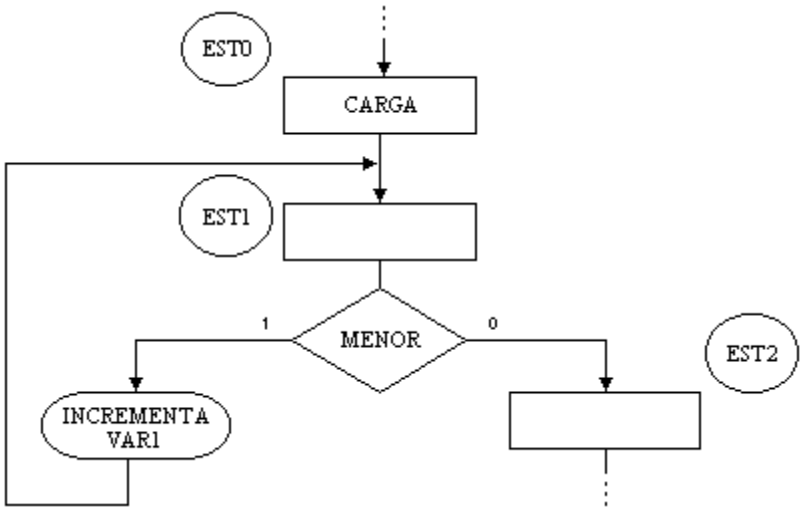


Figura 2.10. Carta ASM para el ejemplo 2.

En el estado EST0 se activa la señal CARGA con el fin de cargar en el contador el valor inicial de X. En el estado EST1 se pregunta por la variable de entrada MENOR, si ésta es igual a cero, la condición X ≤ B es falsa. Si MENOR es igual a uno, la condición es verdadera y por tanto, son activadas las señales INCREMENTA y VAR1 como salidas condicionales.

3) Convertir el siguiente código en lenguaje ‘C’ a una carta ASM.

```
while( x==0 ) {
      var5 = 1;
      var2 = 1;
      if( z==0 ) {
            x = 1;
            var5 = 0;
      }
}
var5 = 0;
var2 = 0;
```

En este ejemplo el valor de la variable X puede ser modificado por la lógica externa o por la máquina de estados. Por ello, para representar a X, utilizaremos un flip-flop cuyo valor será puesto a uno ó a cero dependiendo de las señales internas y externas.

El diagrama de bloques de este sistema es el siguiente.

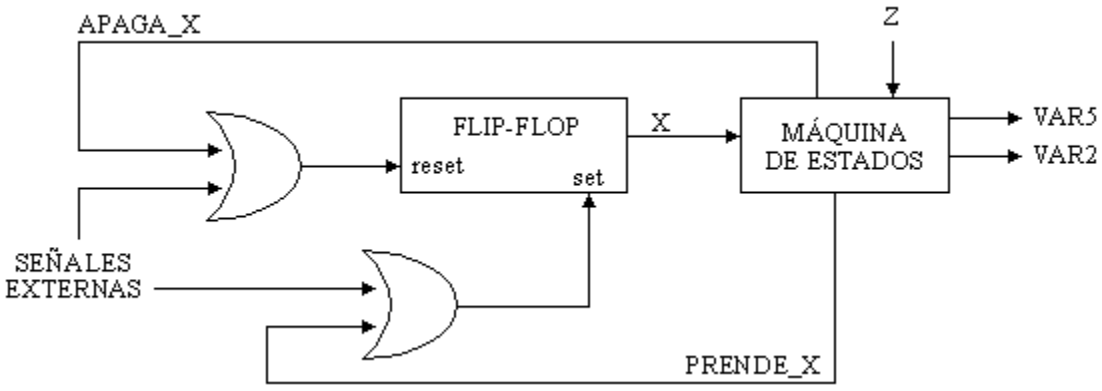


Figura 2.11. Diagrama de bloques para el ejemplo 3.

Y el algoritmo de la máquina de estados queda de la siguiente manera.

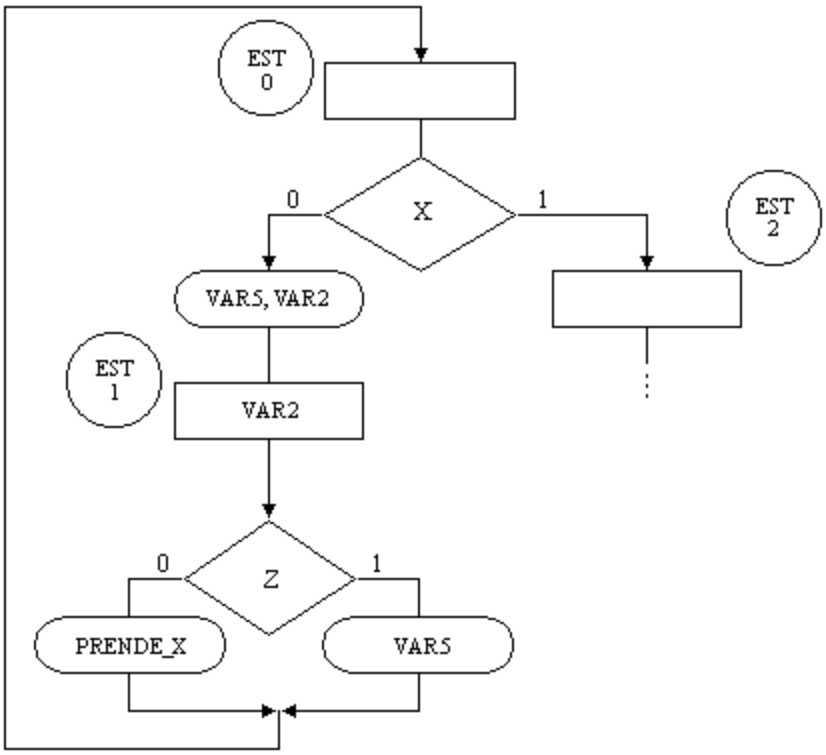


Figura 2.12. Carta ASM para el ejemplo 3.

4) Convertir el siguiente código en lenguaje ‘C’ a una carta ASM.

```
if ( x==n ) {
     var1 = 1;        var2 = 0;
} else {
     var1 = 0;        var2 = 1;
}
var1 = 0;
var2 = 0;
```

En este ejemplo las variables de entrada **x** y **n** están definidas como variables de un sólo bit. Para hacer la comparación de las variables **x** y **n** se usa la función lógica XOR, que valdrá cero cuando **x** y **n** sean iguales, y uno, cuando sean diferentes. La tabla XOR se presenta a continuación.

Entradas		**Salida**
x	n	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 2.1. Función lógica XOR.

El diagrama de bloques que ejecuta el código anterior en C se presenta enseguida.

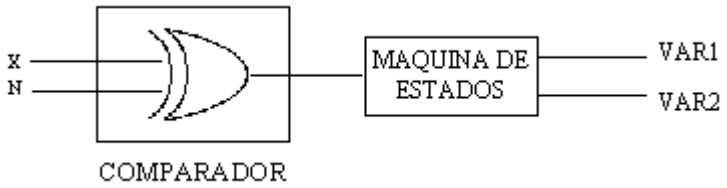


Figura 2.13. Diagrama de bloques para el ejemplo 4.

Y el algoritmo de esta máquina de estados es el siguiente.

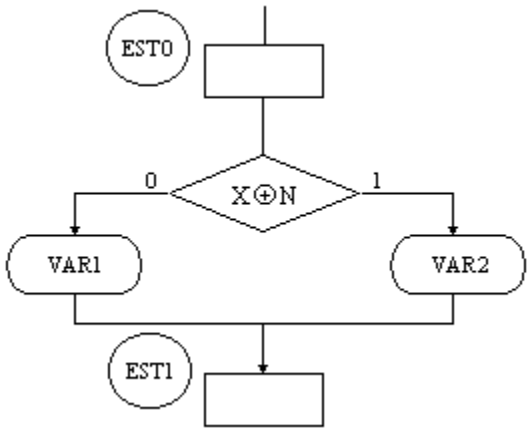


Figura 2.14. Carta ASM para el ejemplo 4.

En el diagrama se observa que si deseamos activar una señal de salida condicional, en un estado en particular, es necesario colocar su nombre dentro de un óvalo.

5) Usando un diagrama de tiempos mostrar la diferencia entre las cartas ASM de la figura 2.15, en donde la variable de salida VAR2 está como salida condicional en la carta ASM1 y como salida incondicional en la carta ASM2.

Si observa detenidamente el diagrama de tiempos de la figura 2.17 observará que la diferencia principal entre las dos cartas ASM es el tiempo cuando se activa la salida VAR2. Cuando está como salida condicional se activa en el estado EST1 junto con VAR1, y cuando está como salida incondicional se activa un flanco positivo después que VAR1.

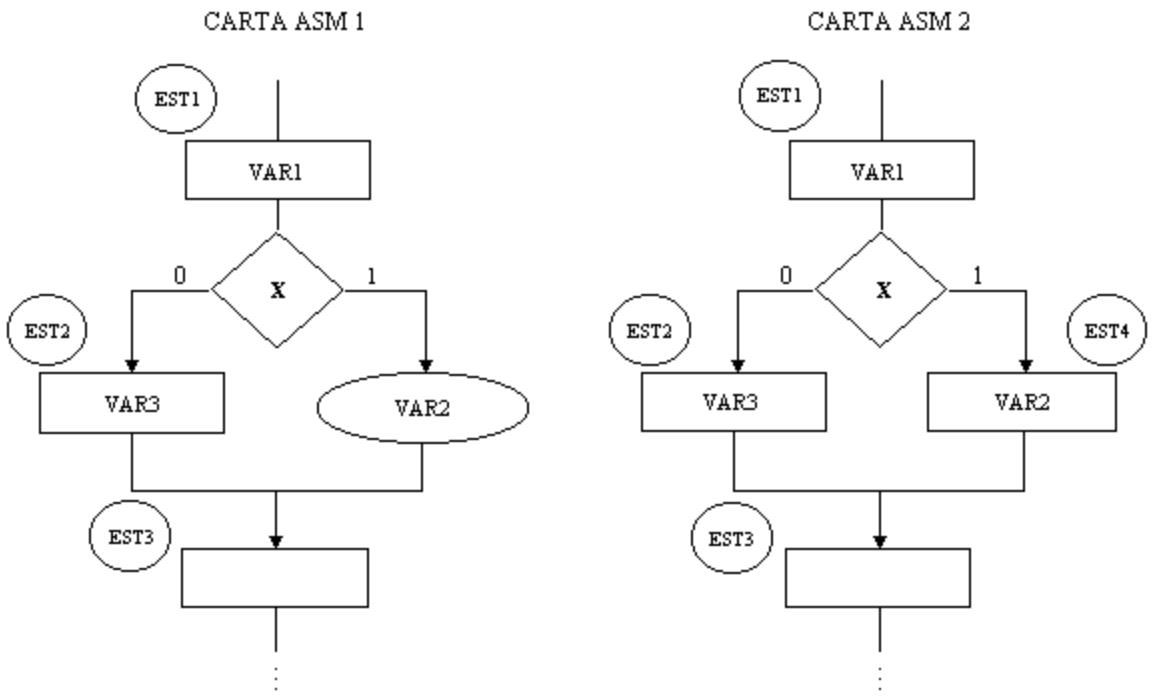


Figura 2.15. La salida VAR2 en la carta ASM1 se presenta como salida condicional, mientras que en la carta ASM2 se presenta como no condicional.

A continuación se muestra el diagrama de tiempos para la carta ASM1 cuando la entrada X es igual que cero. Este diagrama de tiempos es idéntico para la carta ASM2.

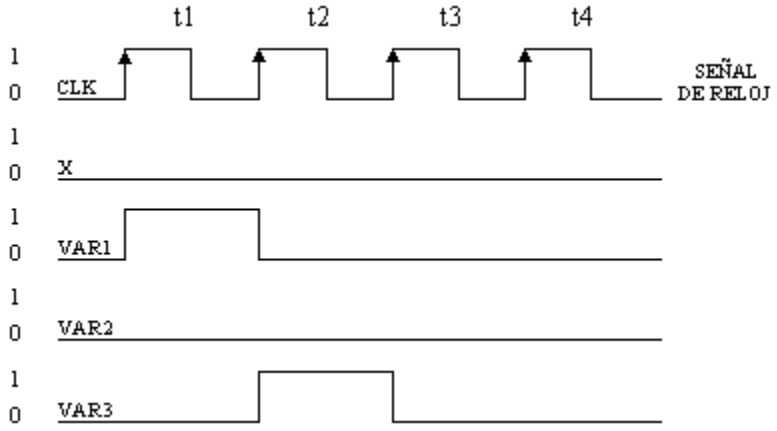


Figura 2.16. Diagrama de tiempos para las cartas ASM1 y ASM2, con x=0.

A continuación se muestran los diagramas de tiempos cuando X es igual a 1.

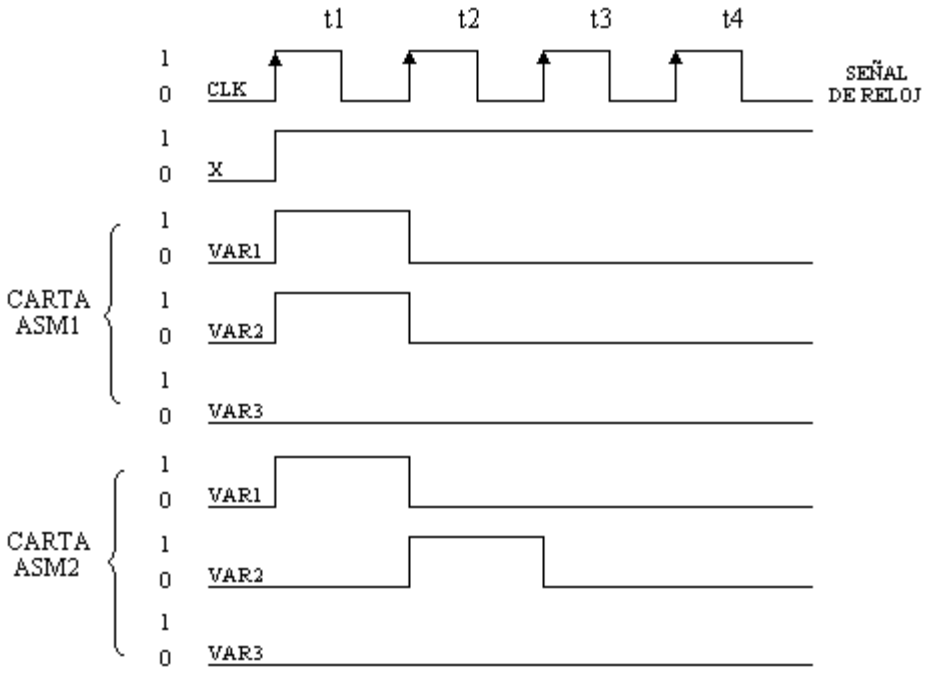


Figura 2.17. Diagrama de tiempos para las cartas ASM1 y ASM2, con x=1.

6) La siguiente figura muestra de manera simple la configuración de N estaciones del metro.

Figura 2.18. Configuración de las N estaciones del Metro.

El objetivo es diseñar un sistema digital, usando máquinas de estados, que mueva al tren de derecha a izquierda sobre la línea. En cada estación hay unos sensores que detectan la entrada de un tren, de manera que cuando arriba a una de ellas, hace una parada de dos minutos.

Además, existe un botón de emergencia en los vagones que hace que el tren se detenga un minuto extra en la estación, si así se requiriera.

A continuación se muestra el diagrama de bloques de este sistema:

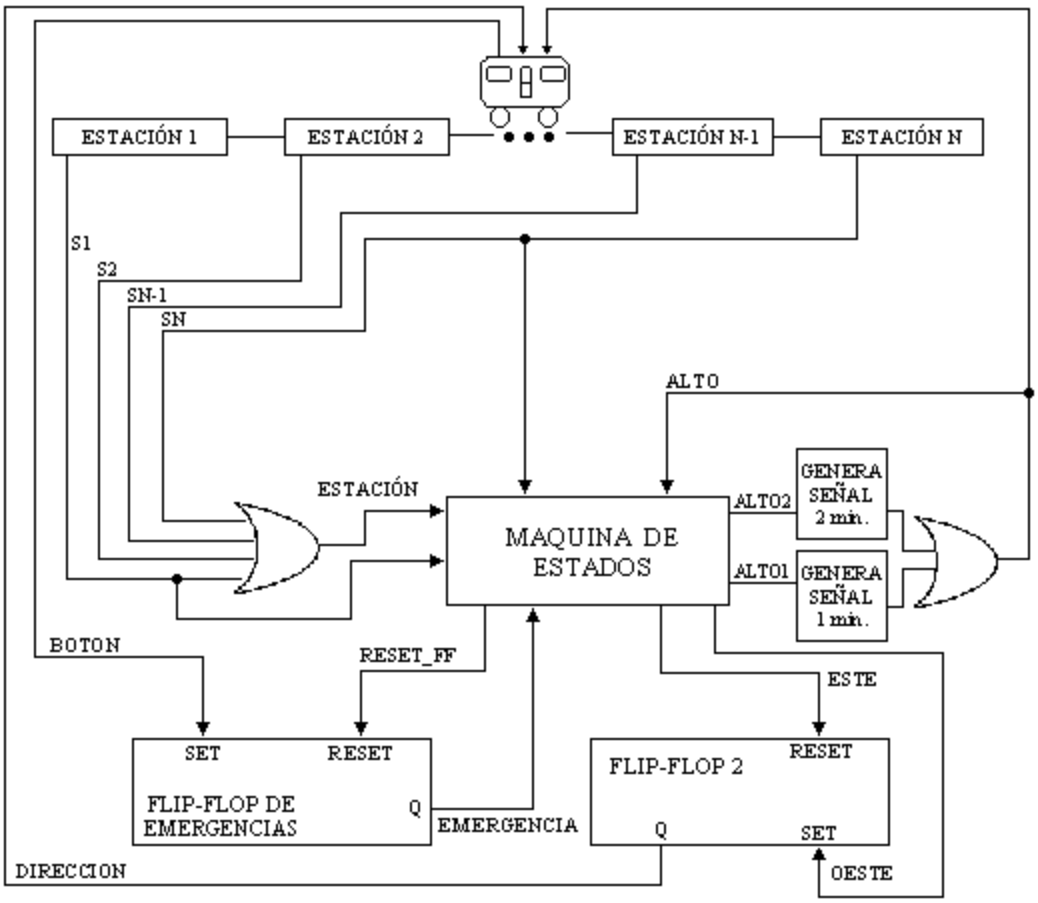


Figura 2.19. Diagrama de bloques para el ejemplo 6.

En el diagrama podemos observar que los sensores que detectan la presencia del tren en la estación están conectados a una compuerta OR. La salida de la compuerta OR, llamada ESTACIÓN, indica si un tren ha entrado en una estación, sin importar a qué estación entró. Sólo interesa saber a qué estación entra si se trata de las estaciones terminales S1 y SN, con el fin de cambiar la dirección de movimiento del tren.

Además, se tiene un módulo que genera una señal de salida de 2 minutos cuando la señal ALTO2 es activada. De manera similar, se tiene otro módulo que genera una señal de 1 minuto cuando ocurre una emergencia, es decir, se activa la señal ALTO1. Ambas señales de espera están conectadas a una compuerta OR. La salida de esta compuerta, llamada ALTO, se encarga de detener al tren durante el tiempo necesario. La salida ALTO también se retroalimenta a la máquina de estados para saber si el tren continúa parado.

También se tienen dos flip-flops, uno indica la dirección de movimiento del tren, y el otro la activación de la señal de emergencia. El flip-flop de emergencias es puesto a uno cuando se oprime el botón de emergencia en el tren, por ello, se conecta la línea del botón de emergencia en el SET del flip-flop. La salida de este flip-flop, denominada EMERGENCIA, entra a la máquina de estados. El

flip-flop de emergencias debe ser puesto a cero nuevamente para permitir otra emergencia, esto se hace por medio de la línea RESET-FF.

Por otra parte, el flip-flop de direcciones le indica al tren la dirección a seguir. Si la salida del flip-flop es igual a cero el tren irá hacia el este, si es uno irá hacia el oeste. El tren estará en movimiento todo el tiempo a menos que la señal de ALTO esté activada.

En la figura 2.20 se muestra la carta ASM de este ejemplo.

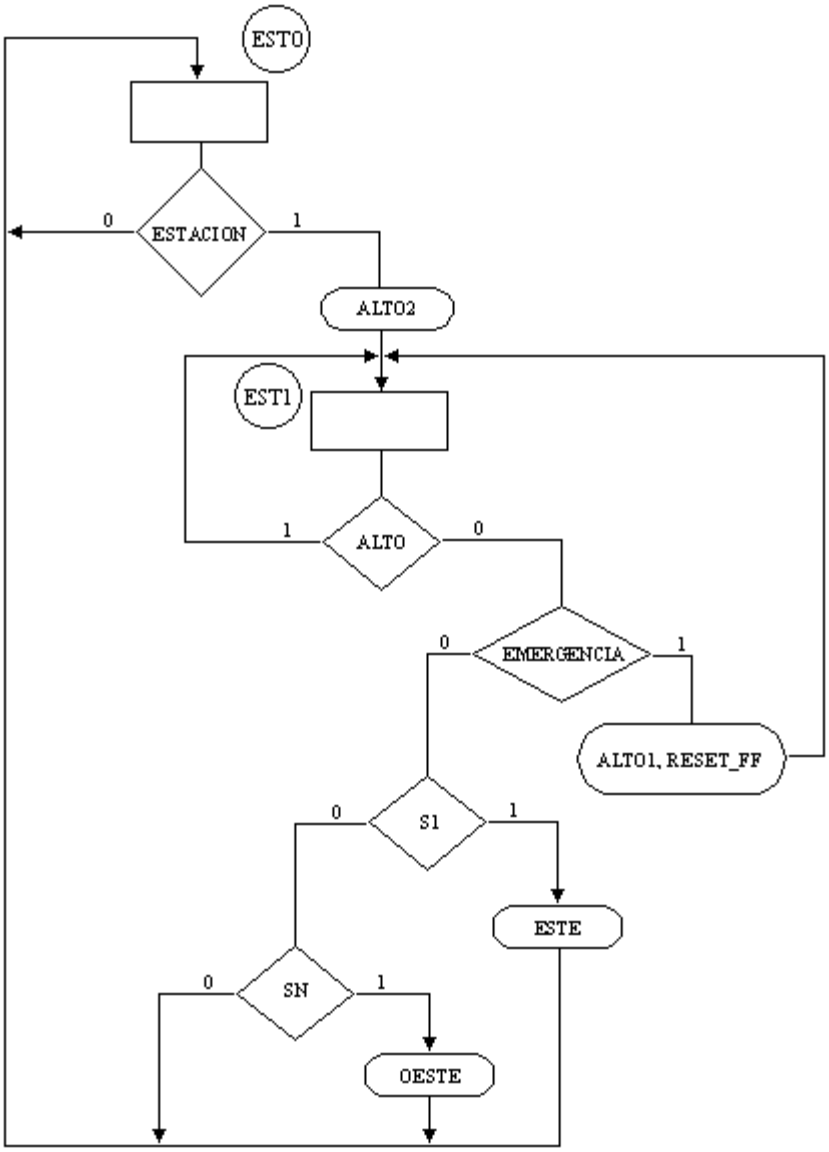


Figura 2.20. Carta ASM para el ejemplo 6.

En el estado EST0 se revisa el valor de la variable ESTACIÓN, si ésta es igual a cero indica que el tren no ha entrado a la estación, es decir, continúa avanzando. Si ESTACIÓN vale uno, entonces se activa la señal ALTO2, la cual genera la señal de 2 minutos que detiene al tren.

En el estado EST1 se revisa la variable ALTO, si ésta vale uno, el algoritmo permanece en ese estado hasta que ALTO valga cero. Cuando ALTO vale cero se revisa la señal de EMERGENCIA. Si el valor de EMERGENCIA es uno, se debe detener el tren por un minuto adicional. Para ello se activa la señal ALTO1 y se limpia el flip-flop de emergencias utilizando la señal de salida RESET_FF, de esta manera nuevas peticiones de emergencia serán permitidas.

Si no hay emergencias, se revisa en qué estación se encuentra el tren para saber si se ha alcanzado una de las estaciones terminales. Si S1 vale uno entonces el tren debe ir al este, por lo tanto, se activa la señal ESTE que pone en cero al flip-flop de dirección. Si SN es igual a uno, entonces se activa la señal OESTE para colocar en uno al flip-flop. Si no se ha alcanzado alguna de las estaciones terminales, el tren continuará avanzando siguiendo la dirección indicada por el flip-flop de direcciones.

7) Diseñe un sistema digital que reconozca la siguiente secuencia binaria.

0 1 0 1 1 1 0 1 1

Cuando haya llegado una secuencia igual, la señal de salida de reconocimiento deberá activarse. Tenga en cuenta que el primer bit de la secuencia es el que se localiza a la derecha, mientras que el último bit de la secuencia es el que se localiza a la izquierda.

La figura 2.21 muestra el diagrama de bloques de este problema en donde X representa cada uno de los bits de la secuencia binaria que hay que reconocer.

Figura 2.21. Diagrama de bloques para el ejemplo 7.

En la figura 2.22 se muestra la carta ASM de esta máquina de estados. Como puede ver, en cada estado se pregunta por el valor de entrada de la secuencia teniéndose una sincronización entre el tiempo de cada estado y las entradas.

Si en algún punto la secuencia es incorrecta, la máquina de estados regresaría a un estado en donde se tomarían en cuenta los valores de las entradas, que incluyendo el valor actual incorrecto, forman una secuencia válida.

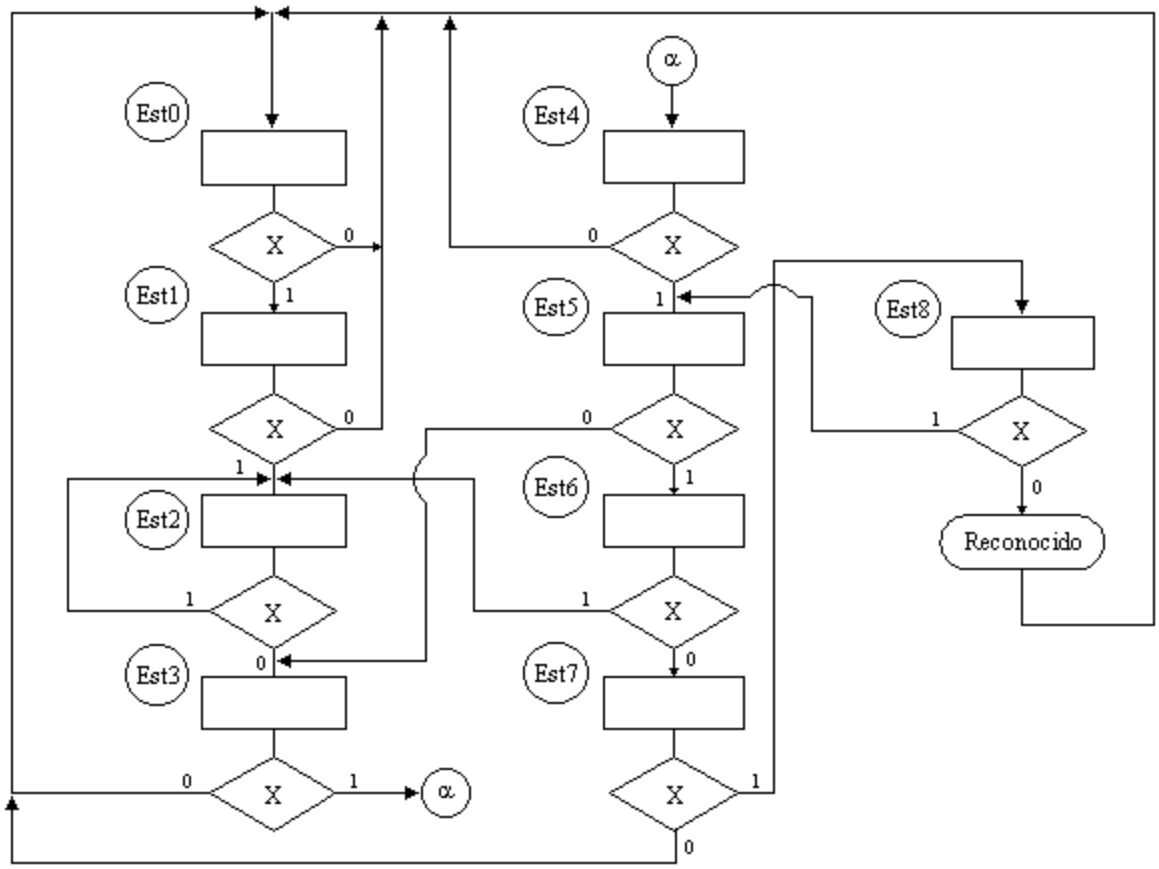


Figura 2.22. Carta ASM para el ejemplo 7.

Uno de los problemas de esta carta ASM es que si se quisiera cambiar la secuencia a reconocer se tendría que cambiar totalmente el algoritmo. Otra forma de resolver este problema sin utilizar cartas ASM se muestra en la figura 2.23, en donde la entrada X se introduce en un registro de corrimiento y se compara contra un registro que contiene el código a reconocer. Como puede ver, esta solución es mejor que el método en donde se utilizan cartas ASM, lo cual indica que no en todos los problemas se debe utilizar esta metodología.

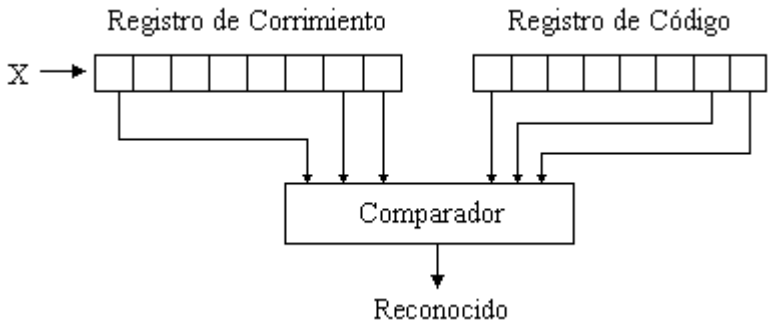


Figura 2.23. Solución al ejemplo 7 sin uso de cartas ASM.

2.4 CONSTRUCCIÓN DE MÁQUINAS DE ESTADOS USANDO LOS MÉTODOS TRADICIONALES

Antes de exponer una técnica de diseño específica, recordemos algunos conceptos de diseño digital como latches, flip-flops y circuitos secuenciales.

2.4.1 CIRCUITO SECUENCIAL

Un circuito secuencial está formado por una etapa de lógica combinacional, y una de etapa de memoria o flip-flops, que se utilizan para representar los estados.

2.4.2 UNIDAD BÁSICA DE ALMACENAMIENTO

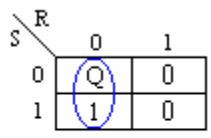
La unidad básica de almacenamiento es un dispositivo que almacena un dato binario, 0 ó 1, dependiendo de los valores de entrada. Este dispositivo presenta el siguiente comportamiento.

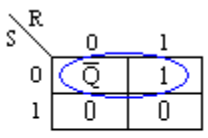
S	R	Q^+	Q^+	Observaciones
0	0	Q	Q	El dispositivo mantiene el valor que tenía guardado
0	1	0	1	Se almacena un cero
1	0	1	0	Se almacena un uno
1	1	0	0	Condición no válida

Tabla 2.2. Tabla de verdad para la unidad básica de almacenamiento.

Como se puede observar, esta unidad de almacenamiento cuenta con dos líneas de entrada, S (set) y R (reset), las cuales indican la manera en cómo deben operar las salidas Q^+ y Q^+. Las salidas de esta unidad siempre son complementarias una de la otra, es decir, cuando Q está a nivel alto, Q está a nivel bajo; y cuando Q está a nivel bajo, Q está a nivel alto. Debido a esta razón, la última condición de la tabla es inválida.

A partir de la tabla de verdad y utilizando mapas de Karnaugh es posible encontrar las expresiones lógicas para la unidad básica de almacenamiento.





$$Q^+ = \overline{R}\,Q = \overline{\overline{R}\,Q} = \overline{R + \overline{Q}}$$

$$\overline{Q}^+ = \overline{S}\,\overline{Q} = \overline{\overline{S}\,\overline{Q}} = \overline{S + Q}$$

Y con base en las expresiones lógicas se construye el diagrama lógico.

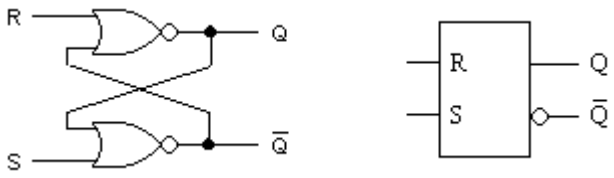


Figura 2.24. Diagrama y símbolo lógico para la unidad básica de almacenamiento.

2.4.3 LATCH TIPO D

El latch tipo D sólo tiene una entrada, además de la entrada de habilitación C, la cual está asociada a un reloj. El comportamiento de este dispositivo se muestra en la siguiente tabla y en el siguiente diagrama de tiempos.

D	C	Q^+	Q^+	Observaciones
*	0	Q	Q	El latch no cambia de estado
0	1	0	1	Latch en estado de Reset
1	1	1	0	Latch en estado de Set

Tabla 2.3. Tabla de verdad para el latch tipo D.

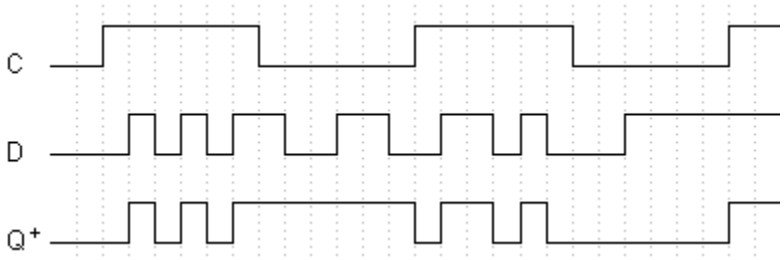


Figura 2.25. Diagrama de tiempos. Comportamiento del latch tipo D.

El diagrama de tiempos muestra de manera gráfica el funcionamiento del latch tipo D. En él se observa que mientras la señal de habilitación C vale uno, la salida Q^+ sigue al valor de la entrada D; y cuando C vale cero, la salida Q^+ mantiene el último valor de D antes de que C cambiara a cero.

El latch también es un dispositivo de almacenamiento de dos estados, por lo tanto, es posible diseñar un latch tipo D a partir de la unidad básica de almacenamiento. La siguiente figura ilustra este procedimiento.

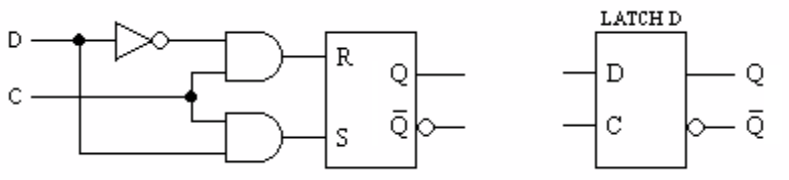


Figura 2.26. Diagrama y símbolo lógico para el latch tipo D.

Flip-flops

Los flip-flops son dispositivos biestables síncronos. En este caso, el término síncrono significa que la salida varía de estado únicamente en un instante específico de una entrada de disparo denominada reloj (clk). Es decir, un flip-flop cambia de estado con el flanco positivo (flanco de subida) o con el flanco negativo (flanco de bajada) del impulso de reloj y es sensible a sus entradas sólo en esta transición de reloj.

Básicamente, los latches son similares a los flip-flops, sin embargo, la diferencia principal entre ambos tipos de dispositivos está en el método empleado para cambiar de estado.

2.4.4 EL FLIP-FLOP TIPO D

Un flip-flop tipo D disparado por flanco negativo presenta el siguiente comportamiento.

D	C	Q^+	Q^+
0	↓	0	1
1	↓	1	0
*	*	Q	Q

Tabla 2.4. Tabla de verdad para el flip-flop D disparado por flanco negativo.

Si existe un nivel bajo en la entrada D cuando se aplica el impulso de reloj, el flip-flop se pone en estado de reset y almacena el nivel bajo de la entrada durante el flanco de bajada del reloj. Si cuando se aplica el impulso de reloj la entrada D está a nivel alto, el flip-flop se pone en estado set y almacena el nivel alto de la entrada durante el flanco de bajada del reloj. El flip-flop mantendrá su valor mientras no exista una transición de alto a bajo en la señal de reloj.

La siguiente figura muestra cómo construir un flip-flop tipo D a partir de dos latches tipo D.

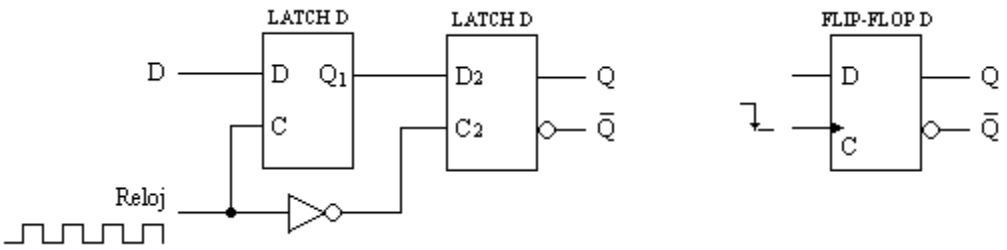


Figura 2.27. Diagrama y símbolo lógico para el flip-flop D.

También se anexa un diagrama de tiempos que muestra de manera gráfica el comportamiento de cada uno de los latches que constituyen al flip-flop tipo D.

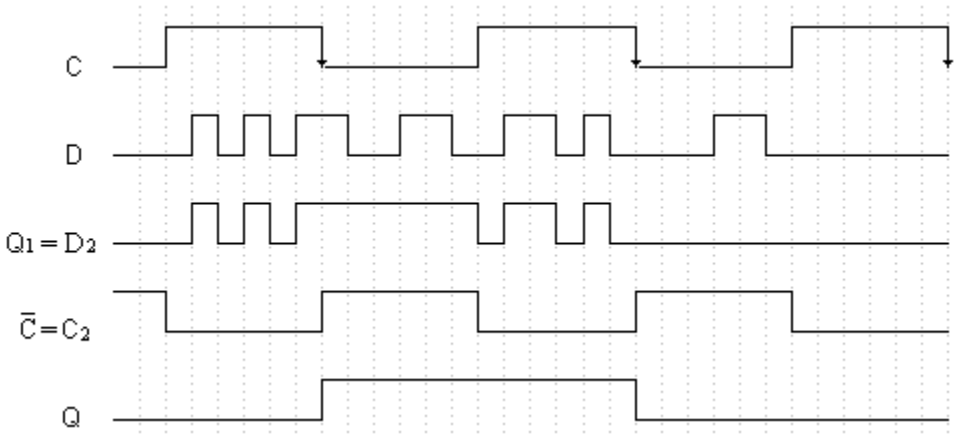


Figura 2.28. Diagrama de tiempos. Comportamiento del flip-flop tipo D.

Del diagrama se puede observar que tener el reloj negado en el segundo latch hace que la entrada D_2 se mantenga fija con el valor de salida del latch 1, el cual tendrá guardado el valor de la entrada D en el momento de la transición del reloj hacia cero.

2.4.5 PROCEDIMIENTO PARA EL DISEÑO DE CIRCUITOS SECUENCIALES

El método tradicional que se utilizará plantea los siguientes pasos para el diseño de un circuito secuencial:

1. Construir un Diagrama de Estados. Un diagrama de estados muestra la progresión de estados por los que el diseño avanza cuando se aplica una señal de reloj. Recuerde que en esta obra, el diagrama de estados corresponde a la carta ASM.

2. A partir del diagrama de estados, desarrollar una tabla con las transiciones entre estados y las salidas para cada estado. El número de flip-flops necesarios para representar todos los estados de la red secuencial está dado por:

$$\text{Número de Flip-flops} = n = \log_2 S$$

donde S es el número total de estados.

3. Transferir los valores para cada entrada D de los flip-flops a un mapa de Karnaugh. A partir de los mapas de Karnaugh podremos determinar las expresiones lógicas para las entradas de los flip-flops.

4. Transferir cada salida a un mapa de Karnaugh y obtener sus expresiones lógicas.

5. Implantación del circuito secuencial. Las expresiones lógicas obtenidas nos permitirán construir el diagrama lógico del circuito.

2.4.6 EJEMPLO

Para ilustrar este procedimiento se desarrollará un ejemplo: Diseñe un circuito secuencial a partir de la siguiente carta ASM.

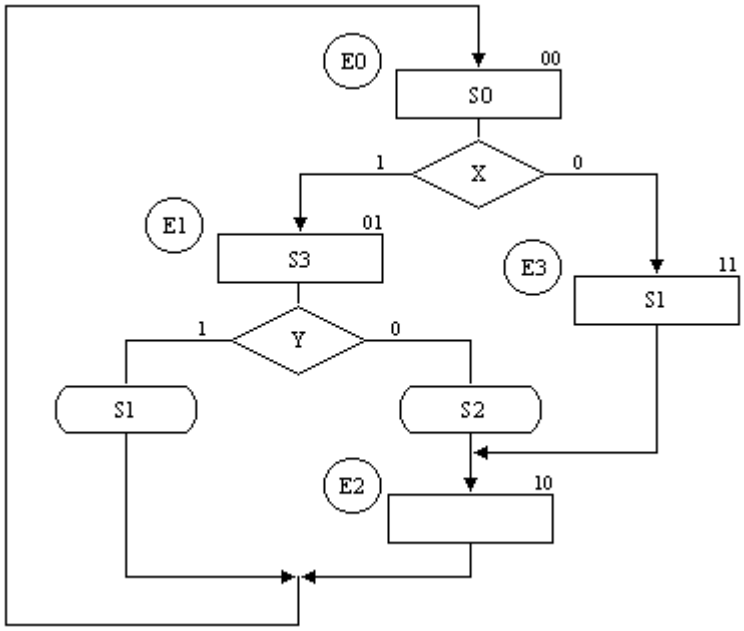


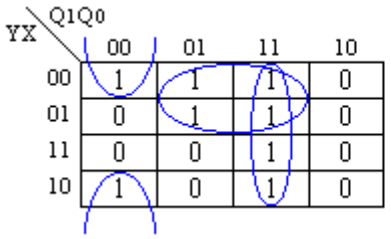
Figura 2.29. Carta ASM.

La tabla de transiciones de estados y de señales de salida es la siguiente.

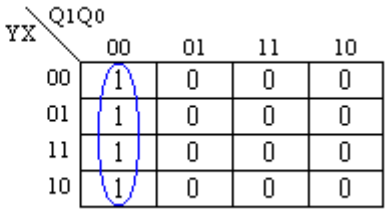
Estado Presente		*Entradas*		*Estado Siguiente*		*Salidas*			
Q_1	Q_0	Y	X	Q_1^+	Q_0^+	S_3	S_2	S_1	S_0
0	0	0	0	1	1	0	0	0	1
0	0	0	1	0	1	0	0	0	1
0	0	1	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1
0	1	0	0	1	0	1	1	0	0
0	1	0	1	1	0	1	1	0	0
0	1	1	0	0	0	1	0	1	0
0	1	1	1	0	0	1	0	1	0
1	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0
1	1	0	0	1	0	0	0	1	0
1	1	0	1	1	0	0	0	1	0
1	1	1	0	1	0	0	0	1	0
1	1	1	1	1	0	0	0	1	0

Tabla 2.5. Tabla de transiciones de estados y de salidas.

Se emplearon dos flip-flops tipo D para representar las transiciones de los cuatro estados que componen a la carta ASM. Los mapas de Karnaugh y las expresiones lógicas para las entradas de los flip-flops son los siguientes.



$$D_1 = Q_1^+ = Q_1 Q_0 + Q_0 Y + Q_1 Q_0 X$$



$$D_0 = Q_0^+ = Q_1 Q_0$$

El mapa de Karnaugh para la entrada D_1 se construye con los valores del estado Q_1^+, mientras que el mapa para la entrada D_0 se construye con los valores del estado Q_0^+.

De manera similar, se obtienen las expresiones lógicas para las señales de salida.

YX \ Q1Q0	00	01	11	10
00	0	1	0	0
01	0	1	0	0
11	0	1	0	0
10	0	1	0	0

$S_3 = Q_1 Q_0$

YX \ Q1Q0	00	01	11	10
00	0	1	0	0
01	0	1	0	0
11	0	0	0	0
10	0	0	0	0

$S_2 = Q_1 Q_0 Y$

YX \ Q1Q0	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	1	1	0
10	0	1	1	0

$S_1 = Q_1 Q_0 + Q_0 Y$

YX \ Q1Q0	00	01	11	10
00	1	0	0	0
01	1	0	0	0
11	1	0	0	0
10	1	0	0	0

$S_0 = Q_1 Q_0$

Finalmente, el diagrama lógico de esta máquina de estados es,

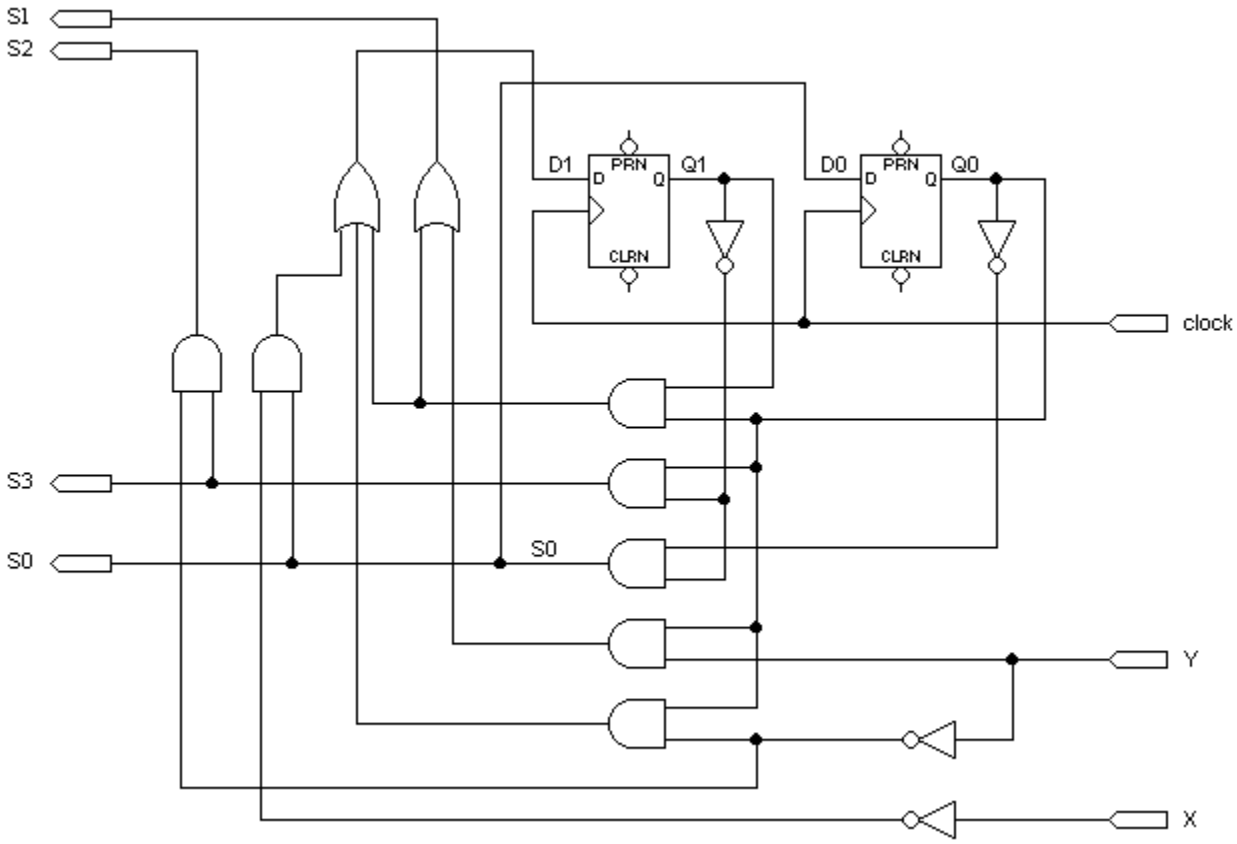


Figura 2.30. Diagrama lógico.

2.6 DISEÑO DIGITAL UTILIZANDO DISPOSITIVOS LÓGICOS PROGRAMABLES

Esta sección intenta explicar brevemente las ventajas del diseño utilizando dispositivos lógicos programables (PLDs, por sus siglas en inglés) sobre los métodos de diseño tradicionales. Esto no significa que los métodos tradicionales ya no sirvan, sin embargo, debido al avance tecnológico, los métodos tradicionales se han visto desplazados por nuevos métodos de diseño.

Para entender esta idea se desarrolla el ejemplo de la sección anterior utilizando un lenguaje de programación para PLDs. Para ello, utilizaremos el software MAX+PLUS II de Altera y alguno de los lenguajes de descripción de hardware que soporta.[1] Entre estos lenguajes están AHDL (Altera Hardware Description Language), VHDL (Very High Speed Integrated Circuit Hardware Description Language) y Verilog HDL. En este caso utilizaremos Verilog por su gran difusión en el campo de los dispositivos lógicos programables.

El siguiente programa implanta la carta ASM de la figura 2.29.

```
module Carta_ASM(clk, Y, X, S);

input               clk, Y, X;
output      [3:0]   S;
reg         [1:0]   currentState;
reg         [3:0]   S;

always @ (currentState) begin
    case (currentState)
            2'b00:          S = 4'b0001;
            2'b01: if (Y)   S = 4'b1010;
                   else     S = 4'b1100;
            2'b10:          S = 4'b0000;
            2'b11:          S = 4'b0010;
    endcase
end

always @ (posedge clk) begin
    case (currentState)
            2'b00: if (X)   currentState = 1;
                   else     currentState = 3;
            2'b01: if (Y)   currentState = 0;
                   else     currentState = 2;
            2'b10:          currentState = 0;
            2'b11:          currentState = 2;
    endcase
end

endmodule
```

El programa comienza declarando el nombre del módulo y las señales de entrada y salida que lo integran. La señal de reloj es *clk*, las señales *X* e *Y* son las variables de entrada, la señal *S* es un

[1] Consulte el apéndice A para cualquier duda acerca del entorno de MAX+PLUS II y del lenguaje Verilog HDL.

vector de variables de salida (S_3-S_2-S_1-S_0), y *currentState* es una variable interna que mantiene el valor del estado actual.

Se emplean dos construcciones always, la primera genera las señales de salida y la segunda calcula las transiciones entre estados.

La primera construcción always tiene como evento de control a la variable *currentState*. Este evento hace que la construcción se ejecute sólo si *currentState* cambia de valor. Cada vez que la construcción se ejecuta, el valor de *currentState* y el de las señales de entrada es revisado, y en base a ellas se asigna un valor al vector de salida *S*.

La segunda construcción always utiliza la señal de reloj como evento de control, esto hace que la construcción sea ejecutada sólo cuando se registra un flanco de subida en la señal de reloj. En esta construcción se genera el valor del estado siguiente en base al valor de la variable *currentState* (el estado actual) y al valor de las entradas.

Las Ecuaciones Lógicas

Cada vez que se compila un proyecto en MAX+PLUS II se genera un archivo con extensión '**.rpt**' con bastante información acerca del proyecto, como por ejemplo, el nombre del dispositivo físico en el cual fue acomodado el diseño, el porcentaje de utilización de este dispositivo, el número de celdas lógicas utilizadas, la asignación de señales de entrada/salida a pines o puertos de entrada/salida en el dispositivo, y quizá la más interesante, las ecuaciones lógicas del proyecto.

Adicionalmente, se crea otro archivo con la información necesaria para programar físicamente el dispositivo lógico programable. Para los PLDs de la familia MAX este archivo tiene extensión '**.pof**' y para los PLDs de la familia FLEX el archivo tiene extensión '**.sof**'.

A continuación se presenta un fragmento del archivo '*Carta_ASM.rpt*' con las ecuaciones lógicas del diseño:

```
** EQUATIONS **

clk      : INPUT;
X        : INPUT;
Y        : INPUT;


-- Node name is ':51' = 'currentState0'
-- Equation name is 'currentState0', location is LC4_A1, type is buried.
currentState0 = DFFE( _EQ001, GLOBAL( clk),  VCC,  VCC,  VCC);
_EQ001 = !currentState0  &  !currentState1;


-- Node name is ':50' = 'currentState1'
-- Equation name is 'currentState1', location is LC2_A1, type is buried.
currentState1 = DFFE( _EQ002, GLOBAL( clk),  VCC,  VCC,  VCC);
```

```
_EQ002 = !currentState0  &  !currentState1  &  !X  #  currentState0  &  currentState1
         #  currentState0  &  !Y;

-- Node name is 'S0'
-- Equation name is 'S0', type is output
S0      =  _LC7_A1;

-- Node name is 'S1'
-- Equation name is 'S1', type is output
S1      =  _LC5_A1;

-- Node name is 'S2'
-- Equation name is 'S2', type is output
S2      =  _LC3_A1;

-- Node name is 'S3'
-- Equation name is 'S3', type is output
S3      =  _LC1_A1;

-- Node name is ':101'
-- Equation name is '_LC1_A1', type is buried
_LC1_A1  = LCELL( _EQ003);
_EQ003 =  currentState0  &  !currentState1;

-- Node name is ':102'
-- Equation name is '_LC3_A1', type is buried
_LC3_A1  = LCELL( _EQ004);
_EQ004 =  currentState0  &  !currentState1  &  !Y;

-- Node name is ':103'
-- Equation name is '_LC5_A1', type is buried
_LC5_A1  = LCELL( _EQ005);
_EQ005 =  currentState0  &  currentState1  #  currentState0  &  Y;

-- Node name is ':104'
-- Equation name is '_LC7_A1', type is buried
_LC7_A1  = LCELL( _EQ006);
_EQ006 = !currentState0  &  !currentState1;
```

Si compara estas expresiones con las calculadas en la sección anterior notará que son muy parecidas, salvo por algunos detalles como los nombres de algunas variables y la notación para las operaciones lógicas **and** (&), **or** (#) y **not** (!).

Ecuaciones obtenidas manualmente	*Ecuaciones de MAX+PLUS II*
	_EQ001 = !currentState0 & !currentState1

$D_1 = Q_1 Q_0 + Q_0 Y + Q_1 Q_0 X$	_EQ002 = !currentState0 & !currentState1 & !X # currentState0 & currentState1 # currentState0 & !Y
$S_3 = Q_1 Q_0$	_EQ003 = currentState0 & !currentState1
$S_2 = Q_1 Q_0 Y$	_EQ004 = currentState0 & !currentState1 & !Y
$S_1 = Q_1 Q_0 + Q_0 Y$	_EQ005 = currentState0 & currentState1 # currentState0 & Y
$S_0 = Q_1 Q_0$	_EQ006 = !currentState0 & !currentState1

Tabla 2.6. Ecuaciones lógicas.

Por último, se presenta un diagrama de simulación del circuito, es decir, cómo se comporta el circuito a lo largo del tiempo. Note los cambios en las variables *currentState*, *S3*, *S2*, *S1* y *S0*, dependiendo de las señales de entrada y del estado presente.

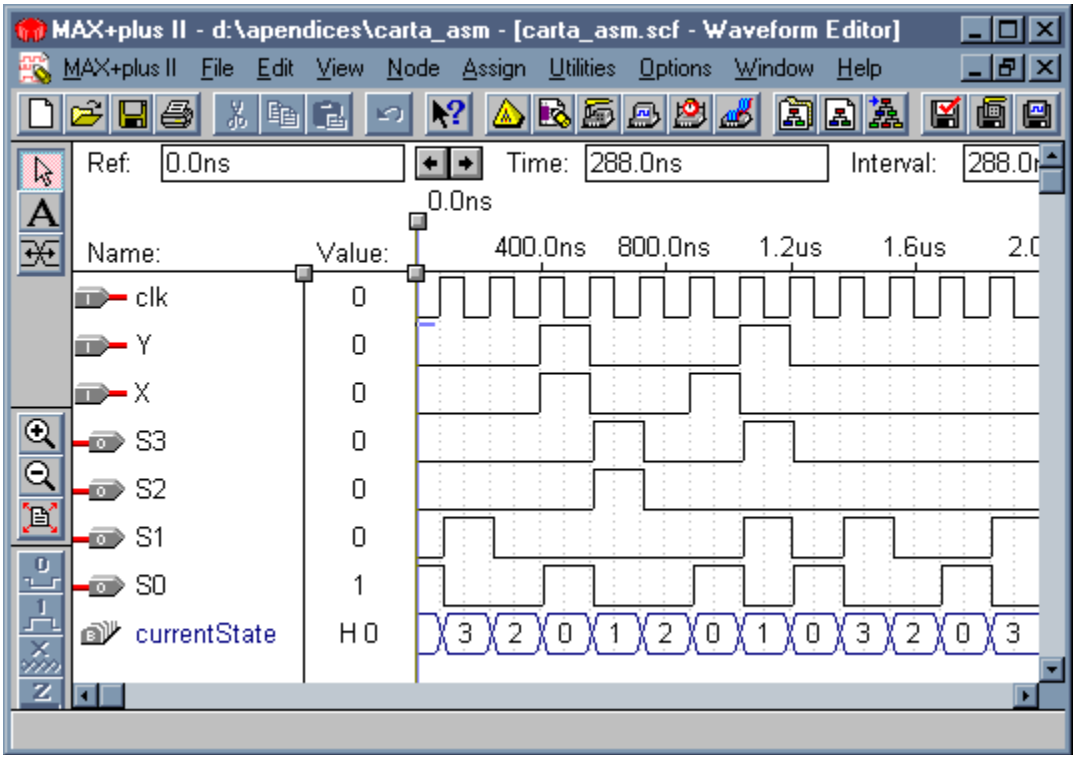


Figura 2.31. Diagrama de simulación.

Del ejemplo anterior podemos concluir lo siguiente:

- Los lenguajes de programación de los PLDs permiten diseñar fácilmente cualquier tipo de sistema digital, además de que permiten hacer modificaciones en el diseño de manera rápida. En cambio, si utilizamos los métodos tradicionales de diseño, una pequeña modificación

implicaría recalcular las expresiones lógicas del circuito, y por lo tanto, necesitaríamos dedicar mayor tiempo al diseño.

- Si trabaja con funciones lógicas muy complejas o muy grandes resultará más barato adquirir un PLD en lugar de adquirir los circuitos integrados por separado. Además de que el espacio ocupado por el PLD en su sistema será menor que el espacio ocupado por todos los circuitos individuales.

- Un PLD será de gran utilidad cuando requiera un sistema digital re-configurable, es decir, que la lógica del sistema necesite modificarse regularmente.

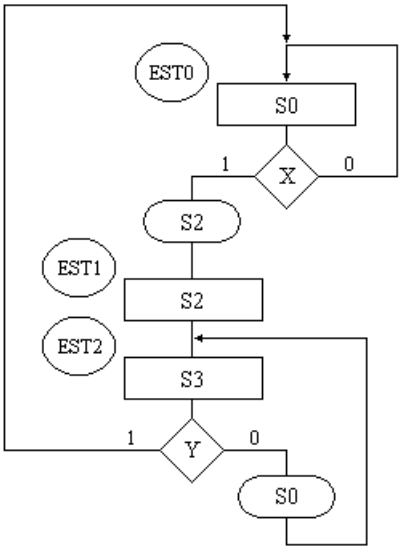
- Si requiere diseñar un sistema de alto rendimiento, utilizar un PLD es una buena opción, ya que estos dispositivos tienen tiempos de respuesta muy pequeños comparados con otros circuitos integrados.

PROBLEMAS

1. Diseñe el algoritmo de una máquina de estados, carta ASM, que genere la siguiente secuencia binaria: 000, 010, 101, 111, 110, 011, 100, 001.

2. Diseñe un sistema digital que reconozca la siguiente secuencia binaria: 1 0 1 1 1 0 1 0. Cuando haya llegado una secuencia igual, la señal de salida de reconocimiento deberá activarse. El primer bit de la secuencia es el que se localiza a la izquierda, mientras que el último bit de la secuencia es el que se localiza a la derecha.

3. Indique el diagrama de tiempos para las variables de salida de la siguiente carta ASM. Suponga que X toma el valor de uno e Y toma el valor de cero.

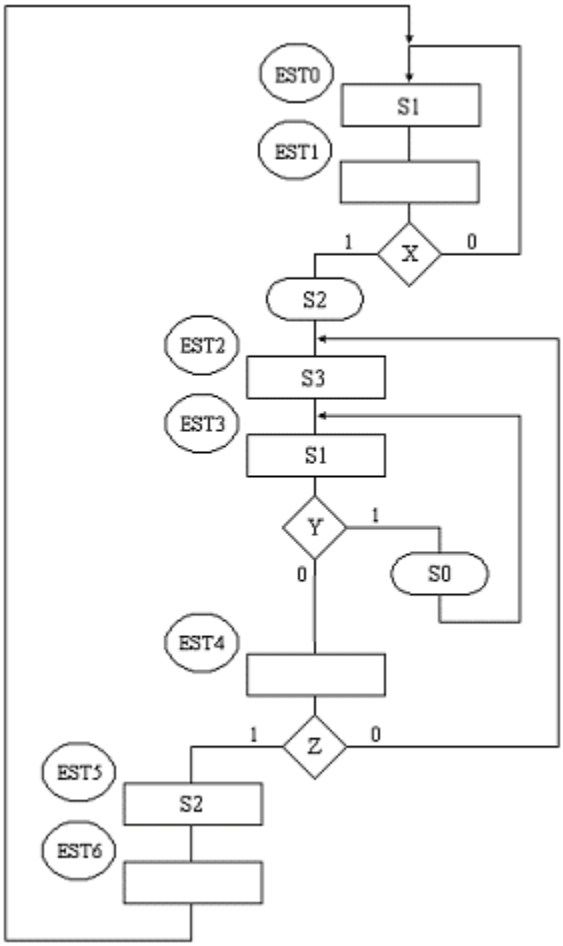


4. Se desea controlar la operación de un elevador para un edificio de ocho pisos. En cada piso, excepto el primero y el último, se cuenta con dos botones que indican la dirección hacia donde se desea ir, es decir, existe un botón para subir y otro para descender. En el primer piso sólo existe el botón para subir y en el último piso sólo existe el botón para bajar. Dentro del elevador se encuentra un tablero con botones que indican el piso al que se desea ir, además, cuenta con botones de abrir y cerrar la puerta y de un botón de emergencia. En los marcos de las puertas se tienen sensores de paso que indican si hay gente entrando o saliendo del ascensor. Diseñe un algoritmo de máquina de estados que controle la operación del elevador, indicando el diagrama de bloques del sistema así como la carta ASM.

5. Modifique el ejemplo de la figura 2.20 para que en lugar de salidas condicionales se tengan solamente salidas no condicionales, ¿qué afecto tiene esto en el desempeño del sistema?.

6. Diseñe una cerradura digital que acepte hasta cinco combinaciones diferentes. Indique el diagrama de bloques del sistema y la carta ASM si se requiriera.

7. Para la siguiente carta ASM encuentre las ecuaciones booleanas de los flip-flops, así como las ecuaciones booleanas de las salidas.



8. Codifique la carta ASM del ejercicio 7 usando el lenguaje de descripción de hardware Verilog HDL.