

# Guía 8: Máquinas de Estado

Escuela de Ingeniería Eléctrica

Universidad de Costa Rica

## 1 Introducción

### 1.1 Objetivos del Laboratorio

El objetivo de este laboratorio es introducir a las personas estudiantes al diseño y la implementación de máquinas de estado finito (FSM) utilizando Verilog conductual. A través del desarrollo de un detector de secuencia y un sistema de telégrafo, las personas estudiantes explorarán cómo modelar sistemas secuenciales y cómo estos pueden ser implementados y verificados en una FPGA. Esta práctica permitirá comprender la importancia de las FSM en el diseño digital, así como desarrollar habilidades para identificar estados y transiciones en sistemas más complejos.

A través de esta actividad, los estudiantes lograrán:

- Diseñar y modelar una máquina de estado finito en Verilog para resolver un problema específico.
- Comprender la estructura básica de una FSM y cómo utilizar estados y transiciones para controlar el flujo de un sistema digital.
- Implementar el comportamiento de una FSM en una FPGA utilizando Intel Quartus Prime.
- Desarrollar habilidades en la depuración de FSM, identificando posibles errores en la transición de estados o en la lógica de control.

## 2 Conceptos teóricos necesarios

### 2.1 Máquinas de Estado Finito

Las máquinas de estado finito (FSM, por sus siglas en inglés) son modelos matemáticos ampliamente utilizados en el diseño de sistemas digitales. Se emplean para describir circuitos que dependen no solo de las entradas actuales, sino también de su historial de estados previos. Son fundamentales en el diseño de controladores digitales, sistemas de comunicación, procesadores, y muchas otras aplicaciones de hardware y software.

En una FSM, el comportamiento del sistema se modela como un conjunto de estados y transiciones entre ellos, que ocurren en respuesta a señales de entrada y están gobernadas por una señal de reloj.

Una máquina de estados se compone los siguientes elementos:

- Un conjunto de Estados ( $S$ ).
- Un alfabeto de Entradas ( $X$ ).
- Un alfabeto de Salidas ( $Y$ ).
- Una función de Próximo Estado, la cual, para cada combinación de estado actual y entradas, calcula a cuál estado deberá transicionar,  $\lambda : X \times S \rightarrow S$ . Esta función está dada por una serie de condiciones, las cuales son las reglas que determinan los cambios.
- Una función de Salida, la cual, para cada combinación de estado actual y entradas, calcula la salida del sistema,  $\delta : X \times S \rightarrow Y$ .

Las máquinas de estado son esenciales en prácticamente todos los sistemas digitales, algunas de sus aplicaciones incluyen:

- Controladores de tráfico: Semáforos que cambian de estado en función del tiempo o sensores de vehículos.
- Decodificadores de protocolos: Interpretación de señales en sistemas de comunicación (como UART, I2C, SPI).
- Sistemas embebidos: Controladores en electrodomésticos, ascensores, dispensadores automáticos.
- Detectores de secuencia: Verificación de patrones de bits en una señal digital.
- Procesadores: Gestión del ciclo de instrucción en CPUs y microcontroladores.

### 3 Ejemplo introductorio - Detector de secuencia

En esta sección se presentará el proceso para diseñar e implementar un detector de secuencia en la FPGA de la tarjeta de desarrollo. Un detector de secuencia es un circuito digital que tiene una entrada por la cual recibe datos y que, al detectar que se recibió una secuencia particular, produce una salida determinada.

Para este ejemplo la entrada será de 1 bit, el cual ingresará por medio del interruptor **SW0** de la tarjeta de desarrollo. El estado del interruptor se capturará al presionarse el botón **Key1**.

La salida del detector de secuencia se mostrará por medio del LED **LED0**. Mientras la secuencia no haya sido detectada el LED deberá estar apagado y si la secuencia es detectada el LED deberá iluminarse.

Adicionalmente, el botón **Key0** funcionará como un *reset* del detector de secuencia. Si este botón es presionado, el detector deberá descartar todos los elementos que haya recibido hasta el momento y volver a iniciar.

Si se detectó la secuencia el LED deberá permanecer encendido aunque se intenten capturar más elementos, el LED solo se apagará si se presiona el botón de *reset*.

La secuencia a detectar para este ejemplo será **1,0,0,1,1**, siendo el bit más a la izquierda el primero en ser recibido.

#### 3.1 Secuencia de funcionamiento del detector de secuencia

A continuación se describe cada paso que el usuario debería realizar para que el detector de secuencia considere la secuencia como detectada:

1. Presionar el botón **Key0**. (Esto se hace para reiniciar el detector y asegurar que la secuencia a ingresar comenzará a partir de este paso)
2. Poner **SW0** en estado **alto**.
3. Presionar el botón **Key1** para capturar el estado de **SW0**.
4. Poner **SW0** en estado **bajo**.
5. Presionar el botón **Key1** para capturar el estado de **SW0**.
6. Mantener **SW0** en estado **bajo**.
7. Presionar el botón **Key1** para capturar el estado de **SW0**.
8. Poner el botón **SW0** en estado **alto**.
9. Presionar el botón **Key1** para capturar el estado de **SW0**.
10. Mantener **SW0** en estado **alto**.

11. Presionar el botón **Key1** para capturar el estado de **SW0**.

12. Tras presionar el botón **Key1** en el paso anterior se considerará la secuencia **10011** como correctamente detectada y en consecuencia el LED **LEDRO** deberá encenderse.

En caso de que, en algún momento, el valor recibido no sea correcto, se debe configurar la máquina de estados para que continúe esperando la secuencia. Si en algún momento se introduce un "0" de forma equivocada, el patrón deberá iniciarse desde el inicio. Pero, si se ingresa un "1" de forma equivocada, el patrón deberá considerar ese "1" como parte de un nuevo intento.

### 3.2 Estados del detector de secuencia

Este detector de secuencia se puede describir como una máquina de estados en la cual los estados representan cada elemento recibido en el orden correcto de la secuencia esperada, empezando con no haber recibido ningún elemento y finalizando con haber recibido la secuencia completa. Por lo tanto, los estados serán los listados a continuación:

1. Estado 0 (S0): No se ha recibido ningún elemento de la secuencia.
2. Estado 1 (S1): Se ha recibido un '1'.
3. Estado 2 (S1): Se ha recibido la secuencia '10'.
4. Estado 3 (S3): Se ha recibido la secuencia '100'.
5. Estado 4 (S4): Se ha recibido la secuencia '1001'.
6. Estado 5 (S5): Se ha recibido la secuencia completa '10011' (Encender **LEDRO**).

Para esta máquina de estado, el evento que producirá una posible transición de estado será el presionado de los botones, ya sea para capturar el estado del interruptor **SW0** mediante el botón **Key1** o para reiniciar el estado de la máquina por medio de **Key0**.

Es importante considerar que, si se recibe un elemento incorrecto después de haber recibido varios elementos correctos no se deberá regresar necesariamente al estado inicial (esto depende de la secuencia a detectar y del elemento que se haya recibido). Por ejemplo, si se recibe un '1' estando en el Estado S2, el próximo estado será el Estado S1. En la Figura 1 se presenta el diagrama que representa la máquina de estados del detector de secuencia.

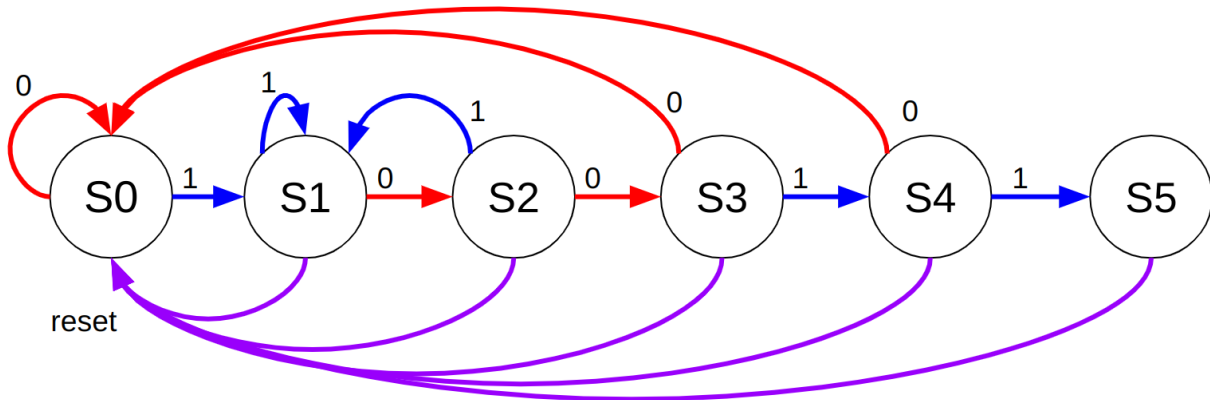


Figura 1: Diagrama de estados del detector de secuencia

Las flechas en rojo indican transiciones cuando la entrada detectada en **SW0** fue un "0", mientras que las flechas en azul indican la transición cuando se detectó un "1". Las transiciones moradas corresponden a cuando se presionó el reset.

### 3.3 Desarrollo del módulo de Verilog del sistema a implementar

Como el detector de secuencia es un módulo simple, se desarrollará en un único módulo de Verilog. Para este, aparte de las entradas y salidas mencionadas anteriormente se incluirá una entrada para el reloj y se definirán dos señales adicionales: `state` y `next_state`. La primera almacenará el estado actual y la segunda almacenará el estado al que deberá transicionar en el próximo flanco del reloj. En cada flanco de reloj se asignará la variable de próximo estado a la variable de estado. Para los estados se utilizó la codificación de la Tabla 1.

Estado	Codificación
S0	000
S1	001
S2	010
S3	011
S4	100
S5	101

Tabla 1: Codificación de estados para el detector de secuencia.

A continuación se muestra el módulo de Verilog que describe el detector de secuencia para este ejemplo:

## sequence\_detector.v

```

1  module lab8_ejemplo ( clk , rst , enter , in , out );
2      // Las entradas son :
3      // Reloj , reset , boton de enter y dato del switch
4      input clk , rst , enter , in ;
5      // La salida es el LED
6      output reg out ;
7      // Codificacion estados
8      localparam S0 = 3'b000;
9      localparam S1 = 3'b001;
10     localparam S2 = 3'b010;
11     localparam S3 = 3'b011;
12     localparam S4 = 3'b100;
13     localparam S5 = 3'b101;
14
15     reg [ 2 : 0 ] state , next_state ;
16
17     // Cuando llega un reset , o se presiona el boton
18     // Se calcula la l gica de pr ximo estado
19     always @ ( negedge enter , negedge rst ) begin
20         // Si hay reset , vuelva a S0
21         if ( rst == 0) next_state = S0 ;
22         // Sino , siga el diagrama de estados
23         else begin
24             case ( state )
25                 S0 : begin
26                     if ( in == 1) next_state = S1 ;
27                     else next_state = S0 ;
28                 end
29                 S1 : begin
30                     if ( in == 1) next_state = S1 ;
31                     else next_state = S2 ;
32                 end
33                 S2 : begin
34                     if ( in == 1) next_state = S1 ;
35                     else next_state = S3 ;
36                 end
37                 S3 : begin
38                     if ( in == 0) next_state = S0 ;
39                     else next_state = S4 ;
40                 end
41                 S4 : begin
42                     if ( in == 0) next_state = S0 ;
43                     else next_state = S5 ;
44                 end
45                 S5 : next_state = state ;
46                 default: next_state = state ;
47             endcase
48         end
49     end
50
51     // Cada flanco positivo del reloj
52     always @ ( posedge clk ) begin
53         // Actualice el estado
54         state <= next_state ;
55         // Si esta en S5 , encienda el LED
56         if ( state == S5 ) out <= 1;
57         // Sino apaguelo
58         else out <= 0;
59     end
60
61 endmodule

```

### 3.4 Creación del proyecto de Quartus Prime

Genere un proyecto nuevo en Intel Quartus Prime siguiendo la secuencia indicada en las guías anteriores. Incluya el módulo de Verilog descrito en la sección anterior en un archivo nuevo y agreguelo en el proyecto creado.

Programa la FPGA de la tarjeta de desarrollo y compruebe que el diseño funciona correctamente. Utilice la asignación de pines indicada en la Tabla 2.

Señal	Componente
clk	MAX10_CLK1_50
rst	Key0
enter	Key1
in	SW0
out	LEDRO

Tabla 2: Asignación de pines para el detector de secuencia.

## 4 Trabajo en Clase

**Recuerde que el dispositivo que se usará es el Max 10 10M50DAF484C7G.**

El circuito a implementar en este laboratorio corresponde a una máquina de estados que permite desplegar mensajes en los display de 7 segmentos. Los caracteres a desplegar serán ingresados en codificación ASCII utilizando los switches SW0-SW7. Una vez ingresado el código, el botón Key1 enviará el código a la máquina de estados. La máquina de estados continuará recibiendo caracteres hasta que se hayan ingresado seis. Una vez esto suceda, la palabra ingresada se desplegará en los displays de 7 segmentos HEX0-HEX5.

A continuación se detalla el funcionamiento:

- Los *displays* de siete segmentos deben permanecer apagados hasta que se haya finalizado de ingresar la palabra.
- El circuito recibirá **letras** codificadas en ASCII por medio de los interruptores SW7-SW0.
- El botón Key0 funcionará como un *reset* del circuito y lo llevarán a su estado inicial (*displays* apagados) y esperando a capturar la primera letra.
- El circuito capturará cada letra cuando se presione el botón Key1.
- El circuito capturará 6 letras y mantendrá el orden en que fueron ingresadas (la primera letra ingresada deberá desplegarse en el display más a la izquierda). Estas 6 letras serán interpretadas como una palabra.
- Cuando se capturen las 6 letras, el circuito mostrará la palabra capturada en los *displays* de siete segmentos usando la codificación indicada en la Tabla 4.
- Si la entrada seleccionada no corresponde a una letra válida, se deberá desplegar la palabra "ERROR" utilizando los displays.

Utilice la asignación de pines indicada en la Tabla 3. Las codificaciones en ASCII de cada letra se muestran en la Tabla 4. **No es necesario codificar todas las letras, únicamente las que se muestran en la Tabla.**

**Como parte de la demostración de este laboratorio deberá mostrar el procedimiento realizado para diseñar la máquina de estados.** No se revisarán demostraciones sin esto.

Señal	Componente
Caracter a desplegar	SW7-SW0
Reset	Key0
Capturar caracter	Key1
Despliegue texto	HEX5-HEX0

Tabla 3: Asignación de pines para este problema







Letra	ASCII	Representación en 7seg	Letra	ASCII	Representación en 7seg
C	8'b01000011		O	8'b01001111	
E	8'b01000101		R	8'b01010010	
I	8'b01001001		U	8'b01010101	

Tabla 4: Patrón del *display* de siete segmentos para algunas letras del alfabeto.