4. Máquinas de estados finitos

En esta práctica se realizarán ejemplos sencillos de máquinas de estados finitos.

4.1 Encender un LED con un pulsador

En un sistema combinacional las salidas dependen exclusivamente del valor de las entradas. En los sistemas secuenciales las salidas no sólo dependen de las entradas, sino que también del estado interno. Este estado interno se consigue gracias a que el circuito tiene elementos de memoria que almacenan el estado. El estado interno del circuito evoluciona según el estado en que se encuentra y el valor de las entradas.

Veamos un ejemplo sencillo: queremos controlar el encendido de un LED con un pulsador, de modo si está apagado, al pulsar una vez se encienda y se mantenga encendido hasta que se vuelva a apretar el pulsador, que hará que se apague. Por lo tanto, las funciones del circuito son:

- El circuito deberá saber si se encuentra encendido o apagado: Deberá recordar en qué **estado** está.
- Consecuentemente si se encuentra encendido, deberá mantener el LED encendido, y si está apagado, el LED no debe lucir: Según el estado en que se encuentre deberá activar las **salidas**.
- Por último, al recibir la señal del pulsador, si estaba encendido deberá apagarse, y si está apagado deberá encenderse. Así pues, según el estado en que esté, y las **entradas** que reciba, deberá cambiar de estado.

Para esquematizar las funciones del circuito dibujaremos un diagrama de transición de estados, en el que se especifica de manera gráfica los puntos anteriormente citados.

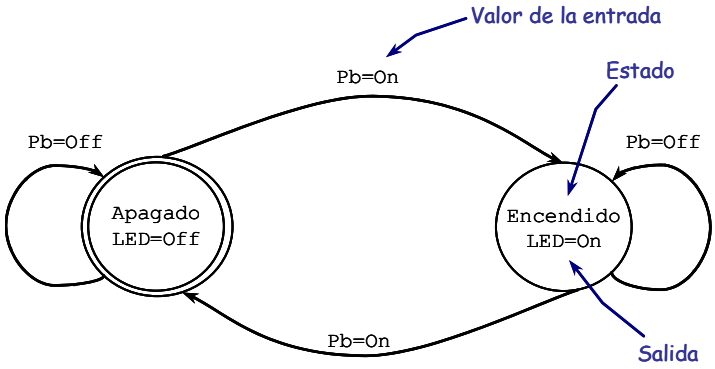

Figura 4-1: Diagrama de transición de estados

En el diagrama (figura 4-1) cada estado se representa dentro de un círculo. A veces, el estado inicial se representa en un círculo doble. En este caso, el estado inicial es `Apagado`. De cada estado salen flechas que indican la transición del estado según el valor de las entradas. En nuestro caso sólo hay una entrada: `Pb`. Si `Pb` se pulsa (`Pb=On`) habrá cambio de estado. Mientras no se pulse, se permanece en el mismo estado. La única salida que hay: `LED`, depende exclusivamente del estado y no de las entradas, y por tanto, se incluye dentro del estado y no en las flechas. Las máquinas de estado que sus **salidas dependen exclusivamente del estado** sin depender de las entradas se llaman máquinas de **Moore**. Cuando las salidas también dependen de las entradas además de los estados se llaman máquinas de **Mealy**.

El diagrama de bloques de nuestro circuito se muestra en la figura 4-2, que es la manera general de representar las máquinas de Moore (cambiando los nombres de las entradas y salidas).

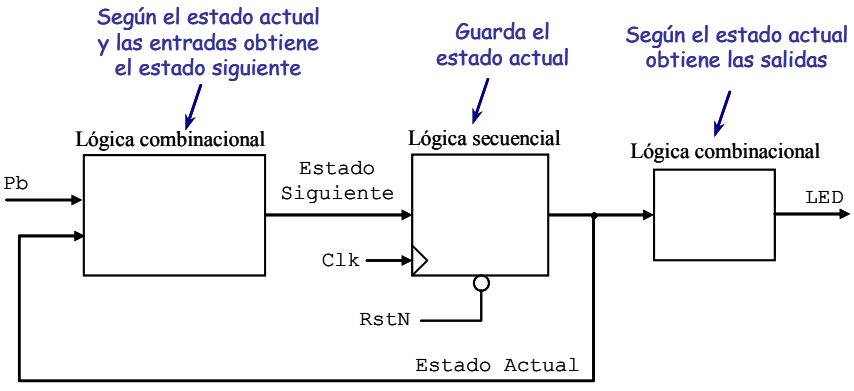

Figura 4-2: Diagrama de bloques del circuito

En este diagrama de bloques hay:

- Un bloque de lógica combinacional que a partir del estado en que se encuentre (`EstadoActual`) y el valor de las entradas (`Pb`) se obtiene el estado al que se mueve la máquina (`EstadoSiguiente`)
- Un bloque de lógica secuencial que guarda el estado (`EstadoActual`) y recibe el estado al que se debe mover en el siguiente ciclo de reloj (`EstadoSiguiente`). Este bloque es secuencial, y por tanto recibe la señal de reloj y de reset.
- Por último, puede tener un bloque de lógica combinacional que genera las salidas (`LED`) a partir del estado (`EstadoActual`)

La implementación de este diseño en VHDL es sencilla y no hace falta realizarla con las estructuras típicas de las máquinas de estados. Sin embargo, se hará así con carácter didáctico.

Antes de empezar a implementar la máquina de estados debemos acondicionar la entrada del pulsador. Debido a que los tiempos de pulsado son mucho mayores que el ciclo de reloj, debemos detectar el flanco de subida del pulsador y convertirla a un único pulso de ancho un ciclo de reloj (esto se explica en el capítulo 0 de repaso de VHDL, ver código 0-16).

```
P_DetectaFlanco: Process (RstN, Clk)
begin
  if RstN = '0' then
    PbReg1   <= cPbOff;
    PbReg2   <= cPbOff;
  elsif Clk'event and Clk='1' then
    PbReg2   <= PbReg1;
    PbReg1   <= PbDcha;
  end if;
end process;

PulsoPb <= '1' when ((PbReg1 = cPbOn) and (PbReg2 = cPbOff)) else '0';
```

Código 4-1: Detector de flanco del pulsador

El código 4-1 no pertenece a la descripción de la máquina de estados y por tanto no tiene representación en la figura 4-2. La salida de este código es la señal `PulsoPb` que se corresponde con la entrada de la máquina de estados. En la figura 4-2 se correspondería con la señal `Pb`.

Antes de pasar a implementar la máquina de estados es útil introducir los enumerados. En VHDL se pueden definir tipos de datos enumerados, que son útiles para describir los estados. Éstos tipos se definen en la parte declarativa de la arquitectura (también se puede hacer en los paquetes). Las señales que se declaren de ese tipo, sólo tomarán esos posibles valores. El uso de tipos enumerados es una alternativa al uso de constantes, como las que se usaron en el código 3-6: `cDirIzq`, `cDirDcha`,...

```
type   estados is (eApagado, eEncendido);   -- definicion del enumerado estados
signal EstadoActual     : estados;          -- senales de tipo estado
signal EstadoSiguiente  : estados;
```

Código 4-2: Declaración de tipo enumerado

Ahora implementaremos en VHDL los tres bloques de la figura 4-2.

```
---------------- Proceso secuencial -----------------------------------------
P_SEQ: Process (RstN, Clk)
begin
  if RstN='0' then
    EstadoActual <= eApagado;
  elsif Clk'event and Clk = '1' then
    EstadoActual <= EstadoSiguiente;
  end if;
end process;
------- Proceso combinacional que obtiene el estado siguiente ----------------
P_Comb: Process (EstadoActual, PulsoPb)
begin
  case EstadoActual is
    when eApagado =>
      if PulsoPb = '1' then
        EstadoSiguiente <= eEncendido;
      else
        EstadoSiguiente <= eApagado;
      end if;
    when eEncendido =>
      if PulsoPb = '1' then
        EstadoSiguiente <= eApagado;
      else
        EstadoSiguiente <= eEncendido;
      end if;
  end case;
end process;

--------- Obtencion de las salidas segun el estado en que se esta ---------------
```

```
Led <= cLedOn when (EstadoActual = eEncendido) else cLedOff;
```

Código 4-3: Descripción de la máquina de estados con dos procesos (tres bloques si se cuentan las salidas): secuencial, combinacional que obtiene el estado y combinacional que obtiene las salidas

Existe una alternativa a la manera de describir una máquina de estados. En vez de usar dos procesos, uno secuencial y otro combinacional, se pueden unir en uno solo. Así, los dos procesos del código 4-3 se pueden implementar en uno sólo (incluso las salidas se podía haber incluido en el mismo).

```
---------------- Maquina de estados en un proceso ------------------
P_FSM: Process (RstN, Clk)
begin
  if RstN='0' then
    Estado <= eApagado;
  elsif Clk'event and Clk = '1' then
    case Estado is
      when eApagado =>
        if PulsoPb = '1' then
          Estado <= eEncendido;
        end if;
      when eEncendido =>
        if PulsoPb = '1' then
          Estado <= eApagado;
        end if;
    end case;
  end if;
end process;

--------- Obtencion de las salidas segun el estado en que se esta
Led <= cLedOn when (Estado = eEncendido) else cLedOff;
```

Código 4-4: Descripción de la máquina de estados con un proceso (dos bloques incluyendo las salidas)

En la versión con un sólo procesos (código 4-4) no hace falta crear dos señales para el estado: `EstadoSiguiente` y `EstadoActual`, sino que basta con una única señal: `Estado`. La síntesis de los códigos 4-3 y 4-4 produce resultados equivalentes. Y la elección de la forma de codificación depende de los gustos del diseñador. Normalmente se considera que con un sólo proceso (código 4-4) es más entendible y más parecido al diagrama de transición de estados (figura 4-1). Por otro lado, la implementación en dos procesos (códigos 4-3) suele ser mejor para detectar errores.

4.2 Clave electrónica

En este ejercicio diseñaremos un circuito que detectará una secuencia determinada de pulsadores. Como si fuese una clave que habría que introducir por medio de los pulsadores. Después de introducir la clave se encenderá un LED como indicador que es correcta, como si el LED fuese un actuador que abriese una puerta.

Los cuatro pulsadores de los extremos serán los utilizados para introducir la clave (`PbUp`, `PbDown`, `PbRight`, `PbLeft`). Se deja el pulsador central como la señal de reset. Consideramos que la clave es `PbUp`, `PbUp`, `PbDown` y `PbRight` (para resumir: `UUDR`). Se considera que se puede introducir la clave correcta en cualquier momento en que entre esa secuencia, esto es, no es necesario que sólo se pueda introducir después del reset. Así pues, al final de la secuencia `LUURLU`**`UUDR`** se lograría *abrir la puerta*, independientemente de que hayamos introducido una secuencia errónea al principio.

Es importante resaltar que si no se recibe ninguna pulsación el estado se mantiene, lo que sucederá a menudo ya que la frecuencia del reloj es mucho mayor que la frecuencia con la que podamos pulsar. Es muy difícil que el usuario pueda presionar dos pulsadores exactamente en el mismo tiempo, si eso sucediese, se debe decidir qué pulsador tiene prioridad.

El diagrama de transición de estados se muestra en la figura de la derecha. (figura 4-3)

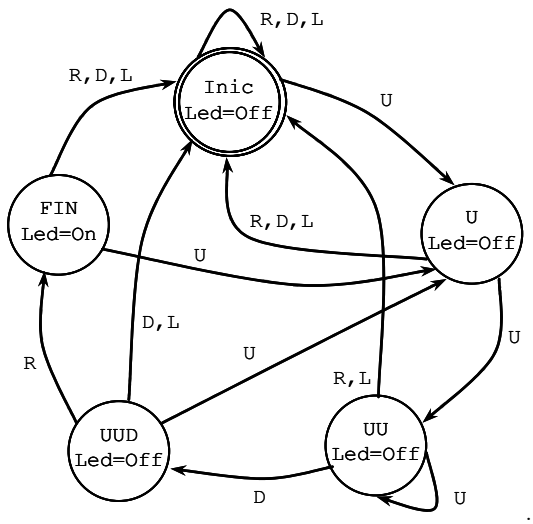

Figura 4-3: Diagrama de transición de estados para la clave electrónica

A continuación se muestra la parte del código VHDL que implementa la máquina de estados, tanto en la versión de dos procesos (a la izquierda, código 4-5) como la de un proceso (a la derecha, código 4-6).

```
P SEQ: Process (RstN, Clk)
begin
  if RstN='0' then
    EstadoAct   <= eInic;
  elsif Clk'event and Clk = '1' then
    EstadoAct   <= EstadoSig;
  end if;
end process;

P COMB: Process (RstN, Clk)
begin
 -- Para evitar poner "else" en todos
 EstadoSig <= EstadoAct;
 case EstadoAct is
  when eInic =>
   if PulsoUp = '1' then
     EstadoSig <= eU;
   end if;
  when eU =>
   if PulsoUp = '1' then
     EstadoSig <= eUU;
   elsif (PulsoRght or PulsoDown or
          PulsoLeft) = '1' then
     EstadoSig <= eInic;
   end if;
  when eUU =>
   if PulsoDown = '1' then
     EstadoSig <= eUUD;
   elsif PulsoUp = '1' then
     EstadoSig <= eUU;
   elsif (PulsoRght or PulsoLeft)= '1' then
     EstadoSig <= eInic;
   end if;
  when eUUD =>
   if PulsoRght = '1' then
     EstadoSig <= eFin;
   elsif PulsoUp = '1' then
     EstadoSig <= eU;
   elsif (PulsoDown or PulsoLeft)= '1' then
     EstadoSig <= eInic;
   end if;
  when eFin =>
   if PulsoUp = '1' then
     EstadoSig <= eU;
   elsif (PulsoRght or PulsoDown or
          PulsoLeft) = '1' then
     EstadoSig <= eInic;
   end if;
 end case;
end process;

Led <= cLedOn when EstadoAct = eFin else
       cLedOff;
```

Código 4-5: Máquina de estados con dos procesos

```
P_FSM: Process (RstN, Clk)
begin
 if RstN='0' then
   Estado <= eInic;
 elsif Clk'event and Clk = '1' then
  case Estado is
   when eInic =>
    if PulsoUp = '1' then
      Estado <= eU;
    end if;
   when eU =>
    if PulsoUp = '1' then
      Estado <= eUU;
    elsif (PulsoRght or PulsoDown or
           PulsoLeft) = '1' then
           Estado <= eInic;
    end if;
   when eUU =>
    if PulsoDown = '1' then
      Estado <= eUUD;
    elsif PulsoUp = '1' then
      Estado <= eUU;
    elsif (PulsoRght or PulsoLeft)='1' then
      Estado <= eInic;
    end if;
   when eUUD =>
    if PulsoRght = '1' then
      Estado <= eFin;
    elsif PulsoUp = '1' then
      Estado <= eU;
    elsif (PulsoDown or PulsoLeft)='1' then
      Estado <= eInic;
    end if;
   when eFin =>
    if PulsoUp = '1' then
      Estado <= eU;
    elsif (PulsoRght or PulsoDown or
           PulsoLeft) = '1' then
      Estado <= eInic;
    end if;
  end case;
 end if;
end process;

Led <= cLedOn when Estado = eFin else
       cLedOff;
```

Código 4-6: Máquina de estados con un proceso

Se deja como ejercicio realizar el mismo circuito con la variante de que cuando llegue al estado final, se mantenga durante 5 segundos, y después de ese tiempo pase al estado inicial.

4.3 Cuatro modos para los LED con máquina de estados

En el capítulo anterior se realizó un diseño en que dos interruptores controlaban el modo de desplazamiento de 4 LED (apartado 3.4). Ahora se va a realizar el mismo diseño pero va a estar controlado por un pulsador. Cada vez que se presione el pulsador se cambiará de modo en este orden: desplazamiento a la izquierda, desplazamiento a la derecha, bidireccional y parpadeo (*flash*).

Inicialmente podemos pensar en una máquina de estados con el diagrama de la figura 4-4. Sin embargo, si hiciésemos el circuito con esta máquina de estado, algunos estados necesitarían guardar cierta información. Algunos de estos estados tendrían otras máquinas de estados internas.

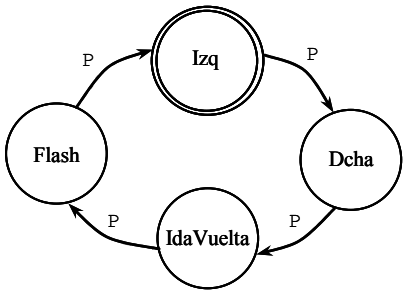

Figura 4-4: Primera versión del diagrama de transición de estados

Por ejemplo, cuando se está en el estado `IdaVuelta` se necesita saber cuál es el sentido de los LED: si están yendo a la izquierda o a la derecha. O por ejemplo, el estado `Flash`, necesitaría saber si deben lucir los LED pares o impares. Por último, también es necesario inicializar los LED al pasar del estado `Flash` al estado `Izq`. Ya que el estado `Flash` enciende dos LED, mientras que el estado `Izq` sólo enciende un LED a la vez.

En la figura 4-5 se muestra una versión más completa del diagrama de estados que a continuación se explica:

- Se parte del estado `Inic`, del que se sale en el siguiente ciclo de reloj. Por tanto, en este estado sólo se está un ciclo de reloj y por eso la flecha de salida no tiene ninguna entrada (siempre sale al estado `Izq`). El estado `Inic` se encarga de inicializar los LED, haciendo que sólo luzca `LED0`.
- De los estados `Izq` y `Dcha` se sale apretando el pulsador.
- El estado `IdaVuelta` se ha separado en dos. El estado `Ida` realiza un desplazamiento a la izquierda, que termina cuando el LED de la izquierda está luciendo. Entonces se pasa al estado `Vuelta`, que desplaza a la derecha. De manera análoga, estando en el estado `Vuelta`, cuando el LED de la derecha luzca, se pasa al estado `IDA`. Habrá transiciones entre estos estados hasta que se presione el pulsador, lo que hará pasar al estado `Flash1`.
- El estado `Flash` se ha dividido en los estados `Flash1` y `Flash2`. Que hacen lucir los LED pares o impares. Cada segundo se pasa de un estado a otro, hasta que se pulse el pulsador, que hará volver al estado `Inic`.

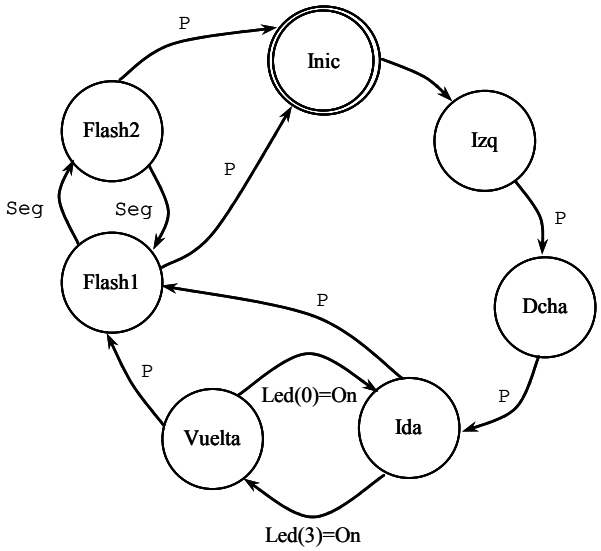

Figura 4-5: Diagrama de transición de estados inicial

El código VHDL con los procesos que implementan la máquina de estados se muestra a continuación.

```
------------------ Proceso secuencial -----------------------------
SEQ_FSM: Process (RstN, Clk)
begin
  if RstN='0' then
    EstadoAct <= INIC;
  elsif Clk'event and Clk='1' then
    EstadoAct <= EstadoSig;
  end if;
end process;
------------------ Proceso combinacional ----------------------------
COMB_FSM: Process (EstadoAct, PulsoPb, RegLed, Segundo)
begin
  case EstadoAct is
    when INIC =>
      if PulsoPb = '1' then
        EstadoSig <= DCHA;
      else
        EstadoSig <= IZQ;
      end if;
    when IZQ =>
      if PulsoPb = '1' then
        EstadoSig <= DCHA;
      else
        EstadoSig <= IZQ;
      end if;
    when DCHA =>
      if PulsoPb = '1' then
        EstadoSig     <= IDA;
      else
        EstadoSig     <= DCHA;
      end if;
    when IDA =>
      if PulsoPb = '1' then
        EstadoSig <= FLASH1;
      elsif (RegLed(3) = cLedOn) then
        EstadoSig <= VUELTA;
      else
        EstadoSig <= IDA;
      end if;
    when VUELTA =>
      if PulsoPb = '1' then
        EstadoSig <= FLASH1;
      elsif (RegLed(0) = cLedOn) then
        EstadoSig <= IDA;
```

```
        else
          EstadoSig <= VUELTA;
        end if;
      when FLASH1 =>   -- parapedo 2 a 2
        if PulsoPb = '1' then
          EstadoSig <= INIC;
        elsif Segundo = '1' then
          EstadoSig <= FLASH2;
        else
          EstadoSig <= FLASH1;
        end if;
      when FLASH2 =>   -- parapedo 2 a 2
        if PulsoPb = '1' then
          EstadoSig <= INIC;
        elsif Segundo = '1' then
          EstadoSig <= FLASH1;
        else
          EstadoSig <= FLASH2;
        end if;
    end case;
  end process;
  ----------------- Proceso de las salidas -----------------------------
  P LEDS: Process (RstN, Clk)
  begin
    if  RstN = '0' then
      RegLed      <= (others => cLedOff);
      RegLed(0)   <= cLedOn;                -- Todos apagados menos el cero
    elsif Clk'event and Clk = '1' then
      case EstadoAct is
        when INIC =>
          RegLed      <= (others => cLedOff);
          RegLed(0)   <= cLedOn;                -- Todos apagados menos el cero
        when IZQ =>
          if Segundo = '1' then
            RegLed <= RegLed(gNumLeds-2 downto 0) & RegLed(gNumLeds-1);
          end if;
        when DCHA =>
          if Segundo = '1' then
            RegLed <= RegLed(0) & RegLed(gNumLeds-1 downto 1);
          end if;
        when IDA =>
          if Segundo = '1' then
            RegLed <= RegLed(gNumLeds-2 downto 0) & RegLed(gNumLeds-1);
          end if;
        when VUELTA =>
          if Segundo = '1' then
            RegLed <= RegLed(0) & RegLed(gNumLeds-1 downto 1);
          end if;
        when FLASH1 =>   -- parapedo 2 a 2
          RegLed <= "1010";
        when FLASH2 =>   -- parapedo 2 a 2
          RegLed <= "0101";
      end case;
    end if;
  end process;
```

Código 4-7: Procesos VHDL que implementan la máquina de estados del control de cuatro modos de los LED

4.4 Ejercicios opcionales

Se dejan como ejercicios opcionales:

- Realizar el circuito anterior con dos pulsadores, de modo que con un pulsador se recorra la secuencia vista de los estados (figura 4-4), y que con el otro se recorra la secuencia inversa: `Izq` → `Flash` → `IdaVuelta` → `Dcha` → `Izq`.
- Añadir un pulsador de modo que haga variar el intervalo que duran los LED encendido: `1seg` → `0.5seg` → `0.25seg` → `1seg`