ÍNDICE

Finalmente, el **quinto paso** consiste en sintetizar el circuito secuencial síncrono que implemente las funciones lógicas anteriores. Si se usan flip-flops D, las entradas al flip-flop corresponden con el nuevo estado ($D_A = NA$, $D_B = NB$), mientras que las salidas del flip-flop corresponden con el estado actual ($Q_A = A$, $Q_B = B$). En la Figura 6.3 se muestra el circuito.

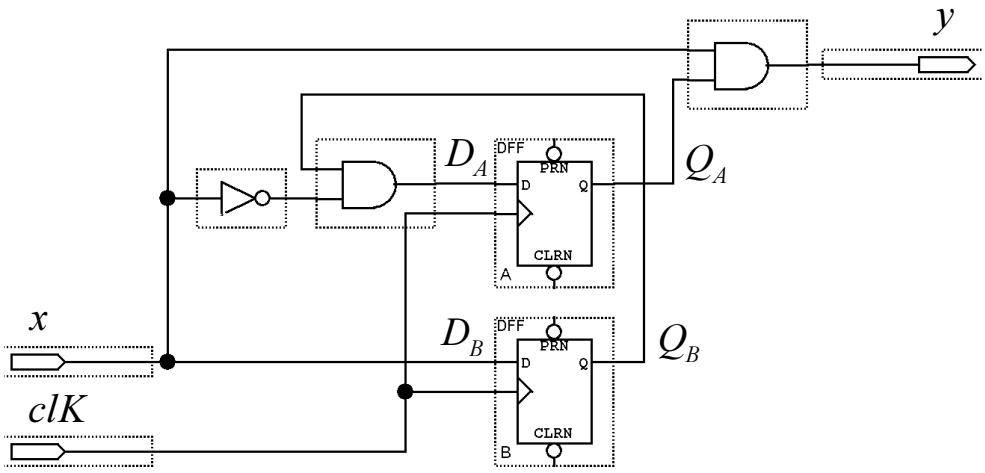

Figura 6.3: Circuito detector de la secuencia 101.

6.3. SÍNTESIS DE LÓGICA SECUENCIAL

El comportamiento y la salida de un bloque de lógica secuencial depende del valor actual de sus entradas y de sus variables de estado. El valor de las variables de estado se almacena típicamente en elementos de memoria, ya sea en *latches* o en *flip-flops*.

Se llama *latches* a aquellos circuitos cuyo estado cambia únicamente cuando su entrada *enable* está a un determinado nivel ('0' ó '1'). A continuación, se muestra parte de la **architecture** de un latch-D.

```
process (enable, D) is
begin
   if (enable = '1') then
      q <= D;
   end if;
end process;
```

Asimismo, se llama *flip-flops* a aquellos circuitos cuyo estado cambia o bien en el flanco de subida del reloj, o bien en el flanco de bajada del reloj. En la Sección 2.7 se describió el diseño de un flip-flop D.

6.3.1. Sentencias condicionales incompletas

Obsérvese que el latch se define empleando una cláusula **if-then** sin **else**. Se denomina *sentencias condicionales incompletas* a este tipo de sentencias. Este punto es importante, ya que muestra que las sentencias condicionales incompletas (**if**, **case**, etc. sin **else**) son sintetizadas mediante latches. Esto es debido a que la salida debe mantener su valor si ninguna de las condiciones de la cláusula se satisface.

Un motivo por el cual podría pensarse en omitir el caso **else**, es que sea indiferente el valor que se asigne a las señales en este caso (ya que en la práctica nunca se producirá). Si no deseamos que el circuito se sintetice mediante un latch, debemos incluir el caso **else** en la sentencia condicional y asignar a las señales el valor `'X'`.

6.3.2. Sentencias condicionales completas

Una característica inherente a las sentencias **if-elsif-else** es que las condiciones no tienen forzosamente que ser excluyentes entre sí. La herramienta se síntesis asume, por tanto, que deben comprobarse las condiciones en un cierto orden de prioridad, generando la lógica para ello. Esto da lugar a circuitería innecesaria en el caso en que las condiciones sean excluyentes.

En aquellos casos en que las condiciones de la sentencia condicional completa sean excluyentes entre sí, es preferible emplear sentencias **case** o **with**. Esto es debido a que las sentencias **case** y **with** se sintetizan de manera natural en multiplexores o estructuras equivalente a multiplexores, que son rápidas y ocupan relativamente poco área.

6.3.3. Retardos

En la Sección 4.2 se explicó que en la descripción para síntesis de los circuitos combinacionales debe evitarse el uso de retardos. Lo mismo aplica a la descripción de circuitos secuenciales. Los retardos en el hardware son dependientes de la tecnología empleada en su fabricación y están sujetos a la variabilidad del proceso de fabricación, por ello es extremadamente difícil construir circuitos que presenten un determinado retardo.

Así pues, no es posible sintetizar sentencias tales como `wait for x ns`. Igualmente, no es posible sintetizar sentencias que empleen la cláusula **after**. Algunas

herramientas de síntesis ignoran estas sentencias, mientras que otras muestran mensajes de error.

Cuando es preciso emplear retardos para describir adecuadamente el circuito, puede definirse un retardo constante en la parte superior del código

```
constant DEL : time := 1 ns;
```

y usar este retardo en el código del circuito. Por ejemplo:

```
A <= B after DEL;
```

Puede asignarse un valor positivo a esta constante de retardo para realizar la simulación del circuito y posteriormente asignarle el valor cero cuando vaya a realizarse la síntesis.

Obsérvese que esta discusión acerca de los retardos aplica a los circuitos, no a los bancos de prueba, que no necesitan ser sintetizados.

6.3.4. Inicialización

Debe evitarse inicializar las variables y las señales al declararlas, ya que este tipo de inicialización no puede ser sintetizada.

La inicialización de una variable o señal implica una acción que se realiza únicamente una vez al comienzo de la simulación. Si es preciso realizar una acción al comienzo de la simulación, entonces debe situarse en la secuencia de acciones que se ejecutan cuando se activa la señal de reset. Estas acciones se definen típicamente dentro de un bloque **process** sensible a la señal de reset.

6.3.5. Bloques process

La descripción de la **architecture** de los circuitos secuenciales se basa fundamentalmente en el empleo de bloques **process**. Las asignaciones a señales dentro de los bloques **process** deben cumplir que:

– La señal en la parte izquierda de la asignación secuencial debe ser una señal definida dentro del bloque **process**, o una señal **out** o **inout** de la interfaz del circuito.

– Las señales en la parte derecha de la asignación secuencial deben ser señales definidas dentro del bloque **process**, o señales **in** o **inout** de la interfaz del circuito.

– No se debe asignar valor a una determinada señal dentro de más de un bloque **process**. Es decir, cada señal debe ser evaluada en un único bloque **process**. Aunque VHDL no impide que se asigne valor a una misma señal en varios bloques **process**, esta práctica da lugar a circuito difíciles de depurar y típicamente no es soportada por las herramientas de síntesis comerciales.

Para asignar valor a las salidas de los flip-flops y latches dentro de un bloque **process**, pueden emplearse sentencias de asignación a señal (usan el operador `<=`) y sentencias de asignación a variable (usan el operador `:=`). Las asignaciones a variable tienen efecto inmediatamente, mientras que las asignaciones a señal tienen efecto en un instante de tiempo δ (delta) unidades de tiempo posterior al tiempo simulado actual (suponiendo que no se ha especificado el retraso en la asignación empleando la cláusula **after**).

Cuando se emplean bloques **process** para describir un circuito secuencial, debe indicarse la lista de sensibilidad de cada bloque **process** con el fin de controlar cuándo se activa el bloque. Cuando se especifica una lista de sensibilidad, el simulador no ejecuta el bloque **process** hasta que no se produce un cambio en alguna de las señales que componen la lista.

Obsérvese que es posible controlar la ejecución del bloque **process** mediante la lista de sensibilidad o mediante el empleo de cláusulas **wait** dentro del cuerpo el bloque **process**. En cada bloque **process** debe optarse por uno de los dos métodos, ya que no es posible emplear simultáneamente una lista de sensibilidad y sentencias **wait**. Sin embargo, no es recomendable emplear este segundo método (uso de sentencias **wait** dentro del bloque **process**), ya que puede dar lugar a circuitos no sintetizables.

6.4. FLIP-FLOP JK

En esta sección se describe el diseño de un flip-flop JK con reset asíncrono activado al nivel LOW. Este circuito puede encontrarse en dos estados: Q='0' y Q='1'. La tabla de transición de estados, considerando únicamente las entradas J y K, así como el estado actual (Q_t) y el siguiente estado (Q_{t+1}), es la mostrada a continuación. Las transiciones de estado se producen en el flanco de subida de la señal de reloj.

J	*K*	Nuevo estado
0	0	Q_{t+1}=Q_t
0	1	Q_{t+1}=0
1	0	Q_{t+1}=1
1	1	Q_{t+1}=not Q_t

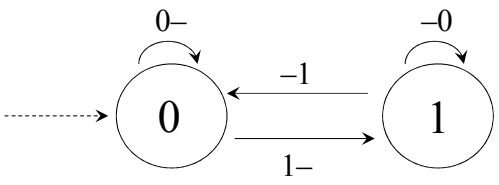

Figura 6.4: Transición de estados de un flip-flop JK. En los arcos del diagrama se muestra el valor de las señales JK. El bit '−' es "don't care" (por ejemplo, "0−" representa "00" ó "01"). La transición de reset se representa mediante una línea punteada.

Cuando la entrada de reset asíncrono pasa de valer '1' a valer '0', el circuito pasa al estado Q='0'. Se denomina *reset asíncrono* porque la transición al estado '0' se produce en el instante en que cambia la señal de reset, con independencia del valor de la señal de reloj. La transición de reset asíncrono se representa mediante una línea punteada en el diagrama situado en la parte derecha de la Figura 6.4.

El circuito tiene dos salidas: Q y $\overline{Q}$. La salida Q es igual al estado y la salida $\overline{Q}$ es su inversa: $\overline{Q} = \text{not } Q$.

6.4.1. Diseño del flip-flop

El Código VHDL 6.1 es el diseño de un flip-flop JK con reset asíncrono activado al nivel LOW. La **architecture** contiene un bloque **process** que es sensible a la señal de reset y a la señal de reloj. En el cuerpo del bloque **process** se examina primero la señal de reset, ya que en caso de estar activa esa señal (con valor '0') la máquina debe pasar al estado Q='0'.

6.4.2. Banco de pruebas

En el caso de los circuitos combinacionales, el objetivo del programa de test es comprobar (cuando esto es posible) todas las posibles combinaciones de valores de las entradas al circuito. En los circuitos secuenciales, el programa de test debe recorrer (cuando esto es posible) todos los arcos del diagrama de estado, para cada uno de los valores de las entradas que producen la transición.

En el caso del biestable JK, esto implica testear 8 transiciones. En la Figura 6.5 se indica el orden en que el programa de test recorrerá los arcos (número inscrito

```
-----------------------------------------------
-- Biestable JK con reset asíncrono en nivel LOW
-- Fichero: flipflop_JK.vhd
library IEEE;
use IEEE.std_logic_1164.all;

entity flipflop_JK is
   port ( q, q_n               : out std_logic;
          clk, J, K, reset_n   : in std_logic );
end entity flipflop_JK;

architecture flipflop_JK of flipflop_JK is
   signal q_interna : std_logic;
begin
   q   <= q_interna;
   q_n <= not q_interna;

   process (reset_n, clk) is
      variable JK : std_logic_vector(1 downto 0);
   begin
      if (reset_n = '0') then
          q_interna <= '0';
      elsif rising_edge(clk) then
          JK := J & K;
          case (JK) is
              when "01"   => q_interna <= '0';
              when "10"   => q_interna <= '1';
              when "11"   => q_interna <= not q_interna;
              when others => null;
          end case;
      end if;
   end process;

end architecture flipflop_JK;
-----------------------------------------------
```

Código VHDL 6.1: Diseño de un flip-flop JK con reset asíncrono.

en una circunferencia), y qué valores de las entradas se aplicará en cada caso para producir la transición. El Código VHDL 6.2 y 6.3 es el banco de pruebas.

En el banco de pruebas se ha definido un **procedure**, que comprueba que los valores actuales de la salida del flip-flop coinciden con los esperados, mostrando el correspondiente mensaje en caso de que no coincidan.

Obsérvese que la sentencia **wait** que hay al final del bloque **process** finaliza la ejecución de dicho bloque, pero no la simulación, ya que la simulación de las sentencias de asignación concurrente a las señales de reset y reloj se realiza indefinidamente. Por tanto, la simulación del banco de pruebas no finaliza por sí misma, siendo preciso fijar su duración: 900 ns. La simulación del banco de pruebas se realiza con cero errores. En la Figura 6.6 se muestra el resultado obtenido de la simulación.