

● ● ● ● ● ●

-
-
-
-
-

Práctica 3

Núcleo del camino de datos de un procesador

.....

En esta sesión se trata de consolidar los conocimientos adquiridos sobre circuitos secuenciales, ya que son de primordial relevancia en la construcción de procesadores segmentados. Para ello utilizaremos el núcleo del camino de datos de un procesador que incluye un banco de registros y un sumador.

Se analiza el funcionamiento síncrono de un banco de registros y la ubicación en el periodo de la señal de Reloj de las fases de operaciones típicas que se efectúan utilizando el camino de datos. También se analiza el retardo de los componentes del camino de datos y se determina el tiempo de ciclo mínimo de la señal de Reloj.

Se introduce la descripción de un modelo de comportamiento en VHDL mediante la sentencia “process” que permite efectuar una especificación secuencial, de forma similar a un lenguaje de programación clásico, para describir el comportamiento de un sistema en función del tiempo. Este método de descripción lo utilizaremos para introducir la especificación de circuitos secuenciales. Además, en el apéndice se describe la especificación en VHDL de circuitos secuenciales básicos, como son el registro y el contador binario.

Descripción del núcleo del camino de datos de un procesador

En la Figura 3.1 se muestra el núcleo de un camino de datos de un procesador el cual contiene un banco de registros y un sumador.

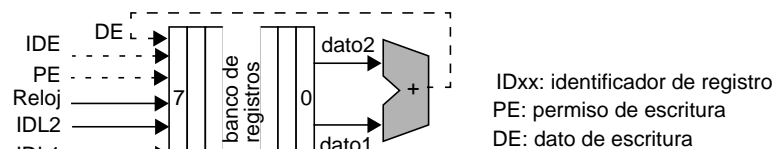


Figura 3.1 Banco de registros: Entradas y salidas.

Sumador. Es un circuito combinacional; la salida del sumador depende únicamente de los valores de las entradas actuales, suponiendo que ha transcurrido un tiempo suficiente para que se establezca la salida.

Banco de registros. Es un dispositivo de almacenamiento. Su salida depende del identificador de registro (entrada) y del valor almacenado previamente en el registro que se quiere leer. Este circuito lógico se denomina circuito secuencial ya que utiliza la historia previa para determinar su salida.

Entonces, el camino de datos de un procesador es una máquina de estados finitos que incluye lógica combinacional y registros que almacenan el estado del sistema.

El banco de registros tiene dos caminos de lectura y uno de escritura. El camino de datos de escritura y la señal de permiso de escritura (PE) se muestran en trazo discontinuo y los caminos de datos de lectura en trazo continuo. Así mismo, se muestra la señal Reloj que sincroniza el funcionamiento del camino de datos.

Cada registro del banco de registros almacena un número determinado de bits que se leen o escriben como una unidad. Los caminos de lectura y escritura al banco de registros se utilizan para leer o escribir un registro concreto. En cada camino de acceso al banco de registros se distinguen las siguientes señales:

- Camino de acceso de lectura: como entrada se utiliza el identificador del registro que se quiere leer y como salida se obtiene el valor que almacena el registro. El número de bits del identificador depende del número de registros.
- Camino de acceso de escritura: como entradas se utilizan el identificador del registro que se quiere escribir y el valor que se quiere almacenar. La escritura de un registro está sincronizada con una señal de Reloj que se utiliza para indicar el instante de tiempo en que se actualiza el registro. Además tenemos otra señal de entrada, denominada permiso de escritura (PE), que se utiliza para inhibir la escritura cuando sea necesario.

Reloj. Es un dispositivo que produce una señal repetitiva de forma permanente y se caracteriza por el intervalo de tiempo de una repetición. A este intervalo de tiempo se le denomina periodo o tiempo de ciclo (Figura 3.2).

En un periodo de reloj se distinguen dos subintervalos de tiempo consecutivos. En uno de ellos el nivel lógico es el cero y en el otro subintervalo de tiempo el nivel lógico es 1.

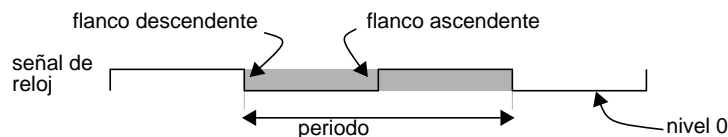


Figura 3.2 Señal de reloj.

En la Figura 3.4 la etiqueta D se corresponde con la entrada de datos del registro y la etiqueta Q se corresponde con la salida del registro. La señal Reloj se utiliza para sincronizar el instante de actualización del registro. Para simplificar los dibujos se representa usualmente un registro de 1bit (rebanada de 1 bit) y la parte asociada del camino de datos. Cuando sea necesario para la comprensión de la exposición, se detallarán los n bits.

Lectura. La salida del registro (Q) puede utilizarse en cualquier instante de tiempo como entrada de un circuito combinacional y esta acción no modifica el estado del registro.

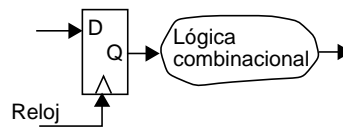


Figura 3.4 Registro de 1 bit.

Parámetro de tiempo asociado a un registro. Existen tres parámetros importantes, dos de ellos son relativos a que la señal de entrada debe ser válida un intervalo de tiempo antes y después del flanco de la señal Reloj. Estos intervalos de tiempo se denominan respectivamente tiempo de estabilización (“set_up time”) y tiempo de mantenimiento (“hold time”) y supondremos que su valor es cero.

El tercer parámetro es el retardo con que se observa en la salida la actualización del registro (retardo de propagación).

- Retardo de propagación (t_p) es el intervalo de tiempo desde el flanco de la señal Reloj, que se utiliza para actualizar el estado, hasta la estabilización del nuevo estado del registro. Como en el caso de puertas lógicas, el retardo de propagación depende de la carga (número de entradas de puertas lógicas) conectada a la salida del registro.

Escritura. La Figura 3.5 muestra la evolución temporal de las señales en una acción de escritura en un registro. Previo al flanco ascendente de la señal de Reloj suponemos que el registro almacena el valor 0. El valor de la señal de entrada (D) en el flanco ascendente de la señal Reloj se observa en la salida (Q) después de un retardo de propagación de la señal (t_p), que es función de los dispositivos lógicos utilizados para construir el registro.

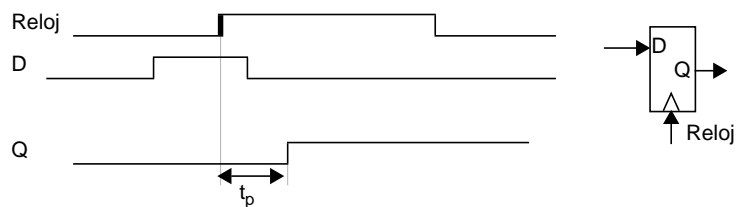


Figura 3.5 Escritura en un registro. El retardo de propagación es t_p .

Modelo de comportamiento en VHDL

Las sentencias de asignación de señal permiten especificar fácilmente el comportamiento de puertas lógicas en circuitos digitales. Sin embargo, sistemas digitales de mayor envergadura necesitan descripciones de comportamiento más complejas. Por ejemplo, modelos con lógica combinacional compleja o modelos que utilizan información de estado. Esto es, lógica secuencial (registros, contadores, ...), máquinas de estados.

VHDL permite representar circuitos digitales en diferentes niveles de abstracción, tales como comportamiento, flujo de datos o estructural. En esta sección se presenta otro constructor para describir el comportamiento de circuitos en términos de sentencias secuenciales y obtener un modelo que describe de forma abstracta el comportamiento. El elemento base del modelado secuencial o de comportamiento es el constructor “process”.

Proceso (“Process”)

Básicamente un proceso (“process”) es una secuencia de sentencias de asignación que se ejecutan en secuencia, siendo todo el conjunto de sentencias secuenciales una sentencia concurrente. A este conjunto de sentencias de asignación se le denomina cuerpo del proceso.

El cuerpo de un proceso se estructura de forma semejante a un lenguaje de programación clásico. Se declaran y utilizan variables y se dispone de constructores como “if-then”, “if-then-else”, “case”, “for” y “while”. Además un proceso puede contener sentencias de asignación de señal.

Un proceso se ejecuta concurrentemente con otras sentencias de asignación de señal concurrentes. Desde el punto de vista de tiempo de simulación no transcurre tiempo al ejecutarse. En este sentido, un proceso es una sentencia de asignación de señal compleja.

Nota: Todos los procesos se ejecutan una vez al iniciarse la simulación.

Recopilando, una sentencia proceso (“process”) es el constructor básico para un modelado que permite utilizar sentencias secuenciales en la especificación del comportamiento de un sistema en función del tiempo. La sintaxis de un proceso es la siguiente:

```

Sintaxis
[process_label:] process [ (sensitivity_list) ] [ is
  [ process_declarations]
  begin
    -- sequential statements
end process [process_label];
```

Núcleo del camino de datos de un procesador

Un proceso puede tener una etiqueta única con la opción “process_label”. La palabra clave “process” sigue a la etiqueta. Después de la palabra clave “process” sigue la lista de activación (sensivity list). La lista de activación es un conjunto de señales, separadas por coma, que determinan cuándo el proceso se ejecuta. Un cambio en el valor de una señal en la lista de activación determina que el proceso se ejecute.

Seguidamente se declaran las variables y constantes que se utilizan en el cuerpo del proceso (“process_declarations”), antes de la palabra clave “begin”. La palabra clave “begin” indica la parte de inicio de cálculos del proceso (cuerpo). Un proceso finaliza con las palabras clave “end process”.

Un proceso se declara dentro del cuerpo de la arquitectura y es una sentencia concurrente. Sin embargo, las sentencias del cuerpo del proceso se ejecutan en secuencia, de forma similar a un lenguaje de programación clásico. Por tanto, el orden de las sentencias es importante a diferencia del orden de las sentencias de asignación de señales. De forma semejante a las sentencias concurrentes, un proceso lee y escribe señales ya sean internas o de la interface. En particular, se pueden efectuar asignaciones a señales que se han declarado externamente al proceso.

Los procesos se comunican mediante señales. Un proceso A puede actualizar una señal que está en la lista de activación de otro proceso B. Esto produce que el proceso B se ejecute después de que el proceso A haya actualizado la señal. A su vez el proceso B puede actualizar una señal que esta en la lista de activación de A.

Sentencias secuenciales (“sequential statements”)

El término sentencia secuencial es debido al hecho de que las sentencias en el cuerpo de un proceso se ejecutan en secuencia. La ejecución de las sentencias secuenciales se inicia cuando se produce un cambio en una señal contenida en la lista de activación. La ejecución de las sentencias dentro de un proceso continua hasta la última sentencia. Después de la ejecución de la última sentencia, el control vuelve al inicio del proceso.

Un tipo de sentencia secuencial es la sentencia de asignación de señal que hemos estado utilizando hasta ahora. Ahora bien, esta sentencia se ejecuta en el orden secuencial en que está escrita. Recordemos que un proceso sólo se ejecuta si se modifica una señal en la lista de activación. Por tanto, la sentencia de asignación de señal sólo se ejecutará en este caso.

Seguidamente describiremos otros tipos de sentencias secuenciales.

Sentencia de asignación de variables. Las sentencias de asignación de variables en un proceso se ejecutan inmediatamente y la acción de asignación se indica mediante el operador “:=” . Este comportamiento es distinto del comportamiento en una sentencia de

asignación de señal (" \leq ") donde el cambio se produce después del retardo especificado¹. Por tanto, los cambios que se efectúan en una variable son visibles (están disponibles) en sentencias posteriores del mismo proceso².

Sentencia "if". En una sentencia "if" la secuencia de sentencias que se ejecuta depende de una o más condiciones. La sintaxis es la siguiente.

Sintaxis
<pre> if [() condition [)] then sequential statements [elsif condition then sequential statements] [else sequential statements] end if;</pre>

Cada sentencia "if" tiene asociada la palabra clave "then". Cada sentencia "if" finaliza con un "end if". Si se quiere utilizar un constructor "else if" la versión VHDL es "elsif". La cláusula final "else" no tiene asociada con ella la palabra clave "then". La cláusula "else" se ejecuta cuando no se cumple ninguna de las condiciones previas. En estas condiciones la sentencia "if" garantiza que al menos una de las secuencias de sentencias se ejecuta. La cláusula final "else" es opcional. Su no inclusión introduce la posibilidad de que no se ejecute ninguna de las secuencias de sentencias asociadas con las condiciones evaluadas.

Cada condición es una expresión lógica (booleana). La sentencia "if" comprueba cada condición en el orden que se han escrito hasta que se encuentra una que se cumpla.

Las sentencias "if" pueden imbricarse.

Tipos de datos en VHDL

El resultado de una comparación ($s = '0'$) es un valor boolean y se utiliza en sentencias condicionales. El lenguaje VHDL tiene predefinido el tipo boolean.

Tipo boolean. Se distinguen los valores: true y false.

-
1. Por defecto hay un retardo delta mayor que cero. Posteriormente se expone con mayor detalle la diferencia.
 2. En un constructor proceso, la utilización del operador " $:=$ " en una sentencia de asignación de variables también se denomina asignación bloqueante. En el caso de utilizar el operador " \leq " en una sentencia de asignación de señal se denomina asignación no bloqueante.

Los valores “true” y “false” de tipo boolean no son equivalentes (o intercambiables) con los valores ‘1’ y ‘0’ respectivamente de std_logic. Seguidamente se muestran dos ejemplos. El ejemplo que se muestra a la izquierda es incorrecto. Debemos efectuar la especificación de la forma que se indica a la derecha. El objeto d en el ejemplo es de tipo std_logic.

Ejemplo incorrecto	Ejemplo correcto
if d then	if d = '1' then

Objetos de datos en VHDL: variables

La sintaxis para la declaración de un objeto variable es la siguiente.

Ejemplos	Sintaxis
variable var: std_logic ; variable index: integer range (0 to 255) := 0; variable index2: integer := -34;	variable variable_name: type := initial value;

Notemos la similitud con las declaraciones de los tipos “signal” y “constant”. En el segundo ejemplo se indica un rango de enteros. Las variables se declaran dentro de un proceso. Fuera de un proceso no se pueden utilizar variables.

Operador de asignación de variables. El operador de asignación de variables es “:=”. Notemos que es el mismo operador que se utiliza para inicializar señales, variables y constantes.

Señales versus variables

Podemos decir que una señal representa un cable o algún tipo de conexión física en un diseño. Una señal toma valores en instantes específicos de tiempo en la simulación. Una variable toma el valor inmediatamente cuando es asignada. Seguidamente se utilizan dos ejemplos para mostrar la diferencia entre la asignación de variables y señales en el cuerpo de un proceso (Figura 3.6).

En el ejemplo de la izquierda de la Figura 3.6 el objeto v_a se declara como una variable. Entonces, la sentencia S2 utiliza el valor establecido en v_a en la sentencia S1.

Ejemplo: variables	Ejemplo: señales
p_var: process (c, d) is variable v_a: std_logic; begin S1: v_a := c and b; S2: ... := v_a ... S3: ... <= v_a ... end process p_var;	signal s_a: std_logic; p_señ: process (c, d) is begin S1: s_a <= c and b; S2: ... := s_a ... S3: ... <= s_a ... end process p_señ;

Figura 3.6 Ejemplos de asignación de variables y señales.

En el ejemplo de la derecha de la Figura 3.6 el objeto `s_a` se declara como una señal. La sentencia S2 no utiliza el valor establecido en la sentencia S1. Utiliza el valor que la sentencia S1 ha establecido en la ejecución previa del proceso. Ello es debido a que una señal toma el valor asignado después de un retardo. Por defecto, este retardo es un valor denominado delta que es mayor que cero. Por tanto, como un proceso se ejecuta en tiempo cero, el valor asignado a una señal no es observable en una sentencia que se especifica posteriormente en el cuerpo del proceso. En otras palabras, se ignoran dependencias textuales entre señales. En la Figura 3.7 se ilustra de forma esquemática el proceso de actualización de señales en un proceso.

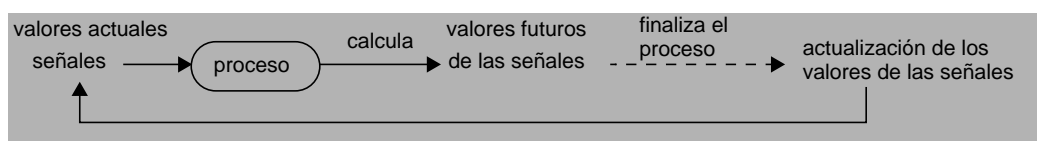


Figura 3.7 Actualización de señales en un proceso.

Modelo de comportamiento de un elemento combinacional: multiplexor

En la Figura 3.8 se muestra la especificación en VHDL de un multiplexor de 4 entradas utilizando un proceso. La especificación lógica de un multiplexor se describe en el Apéndice 2.2 de la Práctica 2.

Notemos que la especificación VHDL de la Figura 3.8 es una traducción directa de la tabla de verdad de un multiplexor. Las señales de control se especifican utilizando un bus.

Señales de 1 bit y un bus como señal de control

```
entity mux is
  port (d0, d1, d2, d3: in std_logic;
        sel : in std_logic_vector (1 downto 0);
        Y: out std_logic);
end mux;

architecture behP of mux is
begin
  proc_mux: process (d0, d2, d2, d3, sel)
  begin
    if (sel = "00") then Y <= d0 ;
    elsif (sel = "01") then Y <= d1 ;
    elsif (sel = "10") then Y <= d2 ;
    else Y <= d3;
    end if;
  end process proc_mux;
end behP;
```

Figura 3.8 Especificación VHDL mediante el constructor "process" de un multiplexor.

Modelo de comportamiento de un elemento secuencial: registro

El lenguaje VHDL no tiene un objeto específico para definir un elemento de memorización. En su lugar, la semántica del lenguaje permite que las señales sean interpretadas como un elemento de memorización. En otras palabras, el elemento de memorización está declarado dependiendo de cómo estas señales son asignadas.

La clave es que la semántica del lenguaje VHDL estipula que, en casos donde el código no especifica un valor de una señal, la señal mantiene su valor actual. En otras palabras, la señal debe recordar su valor actual y para hacerlo implícitamente se necesita un elemento de memorización¹.

Consideremos la especificación de un multiplexor en VHDL mostrada en la parte izquierda de la Figura 3.9. En la especificación del multiplexor, a la señal Q se le asigna un valor en todos los posibles casos de la comprobación en la sentencia "if". En la parte derecha de la misma Figura 3.9 se muestra un código en VHDL donde no se asigna valor cuando la señal PE toma el valor cero. En consecuencia, la señal Q mantiene su valor actual utilizando un elemento de memoria.

El elemento de memorización especificado en la parte derecha de la Figura 3.9 se denomina "latch". Cuando la señal PE toma el valor '1' la entrada se propaga a la salida (modo transparente). Cuando la señal PE toma el valor '0' se mantiene el valor de la señal de salida aunque se modifique el valor de la señal de entrada (modo opaco).

1. De forma no consciente se pueden crear elementos de memorización ("latch"). Una posibilidad para que no ocurra es asignar a todas las señales valores por defecto en el inicio del cuerpo de la arquitectura

Multiplexor	Latch
<pre> library ieee; use ieee.std_logic_1164.all; entity mux is port (D, E: in std_logic ; PE: in std_logic ; Q: out std_logic); end mux; architecture beh of mux is begin process (D, E, PE) begin if (PE = '1') then Q <= D after 10 ns; else Q <= E after 10 ns; end if; end process; end beh; </pre>	<pre> library ieee; use ieee.std_logic_1164.all; entity D_latch is port (D: in std_logic ; PE: in std_logic ; Q: out std_logic); end D_latch; architecture beh of D_latch is begin process (D, PE) begin if (PE = '1') then Q <= D after 10 ns; end if; end process; end beh; </pre>

Figura 3.9 Especificación VHDL de un multiplexor y de un "latch".

Modelo de comportamiento de un registro

Un registro es un circuito secuencial en el que la señal de salida sólo puede cambiar en los flancos de subida o bajada de la señal de reloj.

Antes de mostrar la especificación en VHDL de un registro se describe una funcionalidad de VHDL para identificar flancos en el valor de una señal.

Atributos en VHDL. Los atributos se utilizan para obtener información sobre señales y variables. Los atributos predefinidos en VHDL se aplican a un prefijo tal como una señal o una variable. Hay varias clases de atributos. Por ahora nos centraremos en los atributos de función y en concreto en el atributo "event".

Attribute	Function
signal_name'event	devuelve el valor Booleano verdad ("true") si se produce un evento en la señal, en otro caso devuelve el valor falso ("false")

La utilización de un atributo invoca la llamada a una función que devuelve un valor. Un ejemplo de utilización del atributo es.

```
if (reloj'event and reloj='1') then ...
```

Esta expresión comprueba si se produce un flanco ascendente en la señal “reloj”. Se detecta si se produce un evento en la señal de reloj (reloj’event) y se comprueba si la consecuencia de este evento es que la señal toma el valor ‘1’ (reloj = ‘1’). El valor previo de la señal es cero ya que se identifica un evento cuando hay un cambio en el valor de una señal.

En la parte izquierda de la Figura 3.10 se muestra la especificación VHDL de un registro. Observemos que la condición en la sentencia “if” en el cuerpo del proceso detecta si se produce un flanco ascendente en la señal de reloj. La señal de entrada se transfiere a la salida sólo en este caso. Para otros valores o transiciones de la señal de reloj la salida no se modifica. En consecuencia se ha especificado un registro que almacena el valor que hay en la entrada cuando se produce un flanco ascendente en la señal de reloj.

Registro (flip-flop)	Registro con señal de puesta a cero síncrona
<pre> library ieee; use ieee.std_logic_1164.all; entity reg is port (D : in std_logic ; reloj : in std_logic ; Q : out std_logic); end reg; architecture beh of reg is begin process (reloj) begin -- flanco ascendente de la señal de reloj if (reloj='1' and reloj'event) then Q <= D after 10 ns; end if; end process; end beh; </pre>	<pre> library ieee; use ieee.std_logic_1164.all; entity reg is port (D : in std_logic_vector (3 downto 0); reloj, pcero : in std_logic ; Q : out std_logic_vector (3 downto 0)); end reg; architecture sync of reg is begin process (reloj) begin if (reloj='1' and reloj'event) then if pcero='1' then Q <= (others => '0') after 10 ns; else Q <= D after 10 ns; end if; end if; end process; end sync; </pre>

Figura 3.10 Especificación VHDL de un registro. Registro con señal de puesta a cero síncrona.

En la parte derecha de la Figura 3.10 se muestra la especificación VHDL de un registro con señal de puesta a cero síncrona con la señal de reloj. El sincronismo se determina imbricando la condición de puesta a cero con la condición de flanco ascendente.

Módulo de estimulación

El constructor proceso se puede utilizar para describir en VHDL la generación de señales de estimulación de un circuito. En particular se puede utilizar para generar una señal de reloj y adicionalmente delimitar el número de ciclos. Antes de mostrar la especificación introduciremos otra funcionalidad disponible en VHDL.

Sentencia secuencial en VHDL: wait

En un proceso puede no especificarse una lista de activación. Ahora bien, en este caso es necesario que el proceso tenga alguna sentencia con la palabra clave “wait”.

Sentencia “wait”. La sentencia “wait” suspende la ejecución de un proceso hasta que se produce un evento. Algunas de las posibilidades son:

Ejemplos	Sintaxis
<code>wait for 10 ns;</code> <code>wait until x = '1';</code>	<code>wait for time expression;</code> <code>wait until condition;</code> <code>wait ;</code>

La condición en la sentencia “wait until” debe cumplirse (“true”) para que el proceso reanude la ejecución. En el primer ejemplo, el proceso se espera hasta que hayan transcurrido 10 ns. Si sólo se especifica la sentencia wait el proceso queda suspendido indefinidamente.

En un proceso, si se especifica una lista de activación no se puede especificar una sentencia con la palabra clave “wait” y viceversa.

Ejemplo

En la Figura 3.11 se muestra la generación de una señal de reloj cuadrada con un periodo de 10 ns. Se generan seis periodos de reloj.

```
Señal de reloj
library ieee;
use ieee.std_logic_1164.all;

entity reloj is
port (   reloj : out         std_logic );
end reloj;

architecture beh of reloj is
begin
reloj: process
    variable iter : integer :=0;
    begin
        reloj <= '1'; wait for 5 ns;
        reloj <= '0'; wait for 5 ns;
        iter := iter +1;
        if (iter = 6) then wait;
        end if;
    end process;
end beh;
```

Figura 3.11 Especificación en VHDL de una señal de reloj cuadrada con un periodo de 10 ns durante 6 periodos.

Estructura de un banco de registros

Un banco de registros es un conjunto de elementos de almacenamiento a los que se accede individualmente. El elemento básico de un banco de registros es el registro. Los otros elementos son lógica combinacional para determinar el registro que se quiere leer o escribir y para inhibir la escritura cuando se produce el flanco ascendente de la señal de reloj (permiso de escritura, PE).

Lectura del banco de registros.

La lectura de un registro no modifica el estado, entonces es suficiente con seleccionar el registro que se quiere leer. La selección se efectúa mediante un multiplexor cuyas entradas son las señales de salida de los registros y el identificador de registro. En el Apéndice 2.2 de la Práctica 2 se describe la función lógica de un multiplexor y varias formas de especificarlo en VHDL. También se muestran varias formas de especificar un

multiplexor en VHDL utilizando sentencias condicionales de asignación de señal. Otros elementos combinacionales relacionados con el circuito multiplexor son el decodificador y la puerta de tres estados, que también se describen en el mismo apéndice.

La salida de un registro es un vector de n bits, por tanto es necesario un multiplexor por cada bit. En la Figura 3.12 se muestra un esquema de un banco de registros con 8 registros y dos caminos de lectura. Puede considerarse que se muestra una rebanada de 1 bit del camino de datos o que los buses y multiplexores transportan n bits.

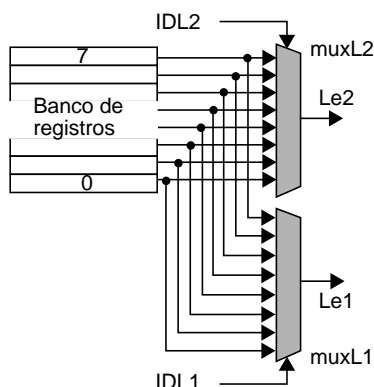


Figura 3.12 Banco de registros. Lógica de lectura.

En la lógica de un multiplexor se incluye la función de decodificación del identificador de registro. En la Figura 3.13 se muestra el caso de selección de un registro de 1 bit en un banco de registros con dos registros.

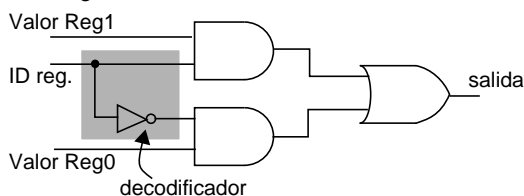


Figura 3.13 Detalle del multiplexor utilizado en la lectura de un banco de registros con dos registros.

Escritura en el banco de registros

La escritura de un registro actualiza el estado. Entonces, además del identificador de registro, el valor que se quiere almacenar y el permiso de escritura, es necesaria una señal que indique el instante de tiempo en que se actualiza.

En la Figura 3.14 se muestra el componente elemental de un registro en un banco de registros. El multiplexor se utiliza para seleccionar entre el contenido del registro (Q) y un dato (D) de entrada. Cuando la señal PE tiene el valor 1 se selecciona la entrada D como

salida del multiplexor. Entonces, cuando se produce el flanco ascendente de la señal Reloj se almacena el valor de la entrada D en el registro. En caso contrario, señal PE igual a 0, la entrada seleccionada es la Q. En consecuencia, cuando se produce un flanco ascendente de la señal Reloj, en el registro se almacena el valor que había previamente. En el diagrama temporal de la figura se observa que en el primer flanco ascendente de la señal Reloj se almacena el valor de la entrada D en el registro, ya que la señal PE está activada. En el segundo flanco ascendente de la señal Reloj, como la señal PE está desactivada no se almacena el valor de la entrada D en el registro.

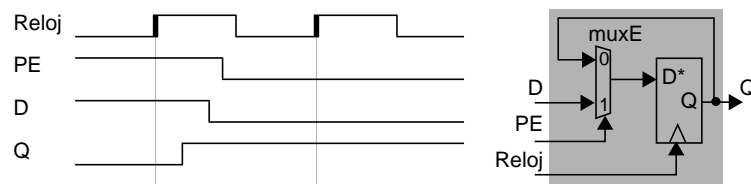


Figura 3.14 Banco de registros. Lógica de escritura.

En la Figura 3.15 se muestra la especificación VHDL de un registro con señal de permiso de escritura (PE). Notemos que el permiso de escritura es síncrono con la señal de reloj. Por tanto, para que se actualice el registro en el flanco ascendente de la señal de reloj debe estar activada la señal de permiso de escritura.

```

Registro con permiso de escritura síncrono
library ieee;
use ieee.std_logic_1164.all;

entity reg is
generic (n: natural :=2);
port ( D : in      std_logic_vector (n-1 downto 0);
      reloj, PE : in  std_logic ;
      Q: out      std_logic_vector (n-1 downto 0) );
end reg;

architecture behreg of reg is
begin
process (D, reloj, PE)
begin
    if (reloj='1' and reloj'event) then
        if PE = '1' then
            Q <= D after 10 ns;
        end if;
    end if;
end process;
end behreg;

```

Figura 3.15 Registro con permiso de escritura síncrono.

En general, en un circuito secuencial se identifican tres elementos: a) elemento de memorización, b) lógica de próximo estado y c) lógica de salida. En el Apéndice 3.1 de esta práctica se muestra la descripción VHDL del circuito de la parte derecha de la Figura 3.14, identificando de forma explícita estos elementos (Figura 3.34, página 129).

En la Figura 3.16 se muestran los elementos del banco de registros que participan en la escritura de un registro y se completa con los caminos de lectura. Para determinar el registro que se escribe se utiliza el decodificador (DECO), el cual tiene como entrada el identificador de registro (IDE). El flanco ascendente de la señal Reloj indica el instante en el que se actualiza el registro, si la señal permiso de escritura (PE) está activada. En Apéndice 2.2 de la Práctica 2 se describe la función lógica de un decodificador. Así mismo se muestran varias especificaciones en VHDL.

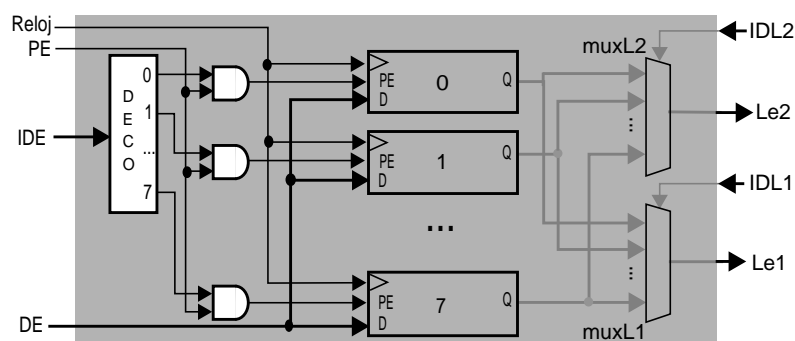


Figura 3.16 Detalle de los elementos de un banco de registros.

Parámetros de tiempo

En el diagrama de tiempos de la Figura 3.17 se observa el retardo de los componentes y la evolución de las señales en una operación de escritura en un registro del banco de registros y su posterior lectura.

Los retardos de la lógica (t_{deco} , t_{AND} , t_{muxE} , t_p , t_{muxL}) se han marcado con una trama negra y las señales de entrada de la lógica se especifican entre paréntesis junto con la etiqueta de la lógica en el margen izquierdo de la Figura 3.17.

Cuando una señal determina un instante de tiempo en el cual deben de ser estables otras señales, este instante de tiempo se utiliza como punto de partida para el análisis del retardo.

El flanco ascendente de la señal Reloj es el punto de partida cuando se analiza una acción de escritura, ya que las entradas de los registros deben ser estables en ese instante de tiempo. Por señal estable se entiende una señal que no modifica su valor.

En el análisis de una acción de lectura también es importante el flanco ascendente de la señal Reloj. En concreto, si se lee el mismo registro que acaba de actualizarse en el flanco ascendente previo de la señal Reloj hay que esperar un retardo igual a la propagación de la señal en el registro para que el registro esté actualizado.

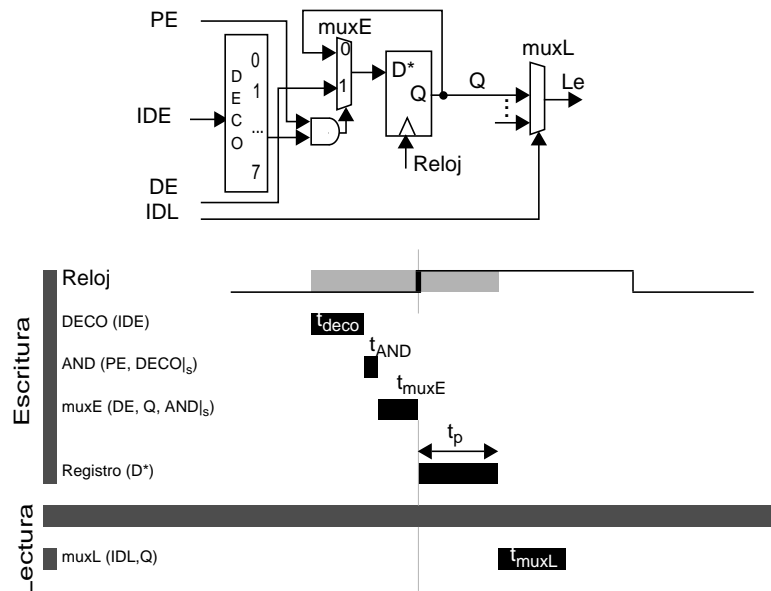


Figura 3.17 Retardos en una operación de escritura en un registro del banco de registros. $DECO|_s$ y $AND|_s$ son las salidas del decodificador y la puerta AND respectivamente.

La señal de entrada en un registro del banco de registros debe ser estable antes del flanco ascendente de la señal Reloj. Para ello es necesario que se haya seleccionado previamente la entrada del multiplexor (muxE), que la señal se transfiera a la salida del multiplexor y que las señales de entrada en el multiplexor muxE sean estables.

La señal de selección del multiplexor muxE es la salida de una puerta AND cuyas entradas son la señal de permiso de escritura (PE) y la salida del decodificador. Por tanto, la entrada del identificador de registro de entrada (IDE) debe ser estable antes de $t_{deco} + t_{AND} + t_{muxE}$ unidades de tiempo. Así mismo, la señal PE debe ser estable antes de $t_{AND} + t_{muxE}$ unidades de tiempo.

En cuanto al dato de entrada al banco de registros (DE) es suficiente que sea estable cuando la salida de la puerta AND es estable en la entrada de muxE. En otras palabras, puede ser estable $t_{deco} + t_{AND}$ unidades de tiempo después de la señal del identificador de registro.

El valor de la entrada DE se observa a la salida del registro (Q) después de un intervalo de tiempo t_p (tiempo de propagación del registro) desde el flanco ascendente del reloj.

El contenido de un registro puede observarse en la salida del banco de registros después del retardo introducido por el multiplexor de lectura (muxL). En la Figura 3.17 se muestra la lectura del mismo registro que se acaba de escribir en el ciclo previo. La señal Q es estable t_p unidades de tiempo después del flanco ascendente de la señal Reloj. Entonces, es suficiente que la señal IDL sea estable en el mismo instante de tiempo. Posteriormente se observa el retardo del multiplexor muxL y la señal Le es válida a partir de $t_p + t_{muxL}$ unidades de tiempo después del flanco ascendente de la señal Reloj.

Modelo de comportamiento de un banco de registros

Antes de describir el modelo de comportamiento de un banco de registros se describen las funcionalidades de VHDL que se utilizan en la descripción del modelo.

Tipos compuestos

Un objeto de datos compuesto consta de una colección de elementos de datos relacionados. Por ahora sólo consideraremos el objeto array.

Un “array” consta de una colección de valores, todos los cuales son del mismo tipo. La posición de cada elemento en un “array” está determinada por un valor escalar denominado índice.

Para crear un objeto tipo “array” definimos en primer lugar un tipo “array” en una declaración de tipo. La sintaxis para la declaración es:

Ejemplos	Sintaxis
type VAR is array (0 to 7) of integer ;	type array_name is array (indexing scheme) of type;
type MY_WORD is array (0 to 15) of std_logic ;	

Un tipo “array” se define especificando el rango del índice y el tipo de elemento. Un rango discreto es un subconjunto de valores de un tipo discreto (tipo entero¹). La sintaxis para especificar un rango es:

Sintaxis
simple_expresion (to downto) simple_expresion

En el último ejemplo previo se ha definido un vector (“array”) de elementos de tipo **std_logic** que se indexa desde 0 hasta 15.

1. Otro tipo discreto es el enumerado, que por ahora no utilizaremos.

Los tipos utilizados como ejemplos pueden usarse para especificar los siguientes objetos:

Ejemplos

```
signal MEM_ADDR: MY_WORD;
constant SETTING: VAR := (2, 4, 6, 8, 10, 12, 14, 16);
```

En el primer ejemplo MEM_ADDR es un vector de 16 bits. Para acceder a un elemento individual se especifica el índice. Por ejemplo MEM_ADDR(15) accede al bit más a la derecha del vector.

Para acceder a un subrango se especifica un rango de índices, siendo un ejemplo:

Ejemplo

```
MEM_ADDR (0 to 7);
```

De los 16 bits del vector MEM_ADDR se accede a los 8 bits más a la izquierda del vector.

Conversión de tipos.

Puesto que VHDL es un lenguaje fuertemente tipado, no se puede asignar un valor de un tipo a un objeto de diferente tipo. En general es preferible utilizar todos los objetos del mismo tipo. Sin embargo en ocasiones es de utilidad manipular objetos de tipos distintos. Para efectuar asignaciones de objetos de tipos distintos es necesario efectuar una conversión de tipo. En elementos denominados “package” de la librería IEEE existen funciones para efectuar los cambios de tipo.

Un “package” de IEEE que extiende los tipos de objetos es “numeric_std”. En este “package” se definen además operadores para los nuevos tipos de objetos. Ahora bien, por razones históricas existen otros “packages” suministrados por proveedores y vendedores de software que no son parte de los estándar de IEEE, pero que se almacenan en la librería IEEE. Uno de ellos es el “package” “std_logic_arith” que se utiliza en LogicWorks. Este “package” define también los tipos de objetos y operadores definidos en el “package” “numeric_std” de IEEE. En LogicWorks el “package” “std_logic_arith” define operadores aritméticos para objetos de tipo “std_logic_vector”. En este sentido, un objeto de tipo “std_logic_vector” se interpreta como un número binario sin signo.

Función	Parámetro (fuente)	Resultado (destino)	“package”
conv_integer	std_logic_vector	entero	std_logic_arith
to_stdlogicvector	bit_vector	std_logic_vector	std_logic_1164
to_bitvector	std_logic_vector	bit_vector	std_logic_1164

Especificación VHDL de un banco de registros

Un banco de registros es un vector de registros con lógica de decodificación para escribir, con una señal de permiso de escritura y un multiplexor para la lectura.

Los registros se estructuran como una matriz de elementos de almacenamiento de un bit. En VHDL, en los “packages” usuales como “std_logic_1164” no están predefinidos estos tipos de objetos. Entonces se crea un tipo de datos vector de vectores definido por el usuario.

Supongamos que el número de registros es NR y el número de bits que almacena cada registro es N. El tipo de datos y la declaración de señal pueden escribirse de la siguiente forma.

Declaración de tipo

```
type BRtipo is array (0 to NR - 1) of std_logic_vector (N - 1 downto 0);
signal BR: BRtipo;
```

Para acceder a un registro se utiliza:

BR(i), donde i es de tipo entero

En la Figura 3.18 se muestra la especificación VHDL de un banco de registros. El banco de registros tiene dos puertos de lectura y un puerto de escritura. Para acceder a la señal que modela el banco de registros (“mem”) se efectúa una conversión de tipo std_logic_vector a entero.

Nota: VHDL permite la definición de modelos parametrizados. Para especificar un parámetro se utiliza la palabra clave “generic” antes de la declaración de los puertos de entrada y de salida. Un genérico es un objeto constante y únicamente puede ser leído.

El proceso denominado escribir y la sentencia de asignación de señal anterior modelan el puerto de escritura. En esta sentencia de asignación se especifica el retardo de la decodificación del identificador del registro a modificar. En la lista de activación del proceso está la señal de reloj. El retardo de actualización de un registro en el banco de registros se especifica al efectuar la acción de escritura. Un registro se actualiza después de que hayan transcurrido 14 ns desde el flanco ascendente de la señal de reloj y además esté activado el permiso de escritura (PE).

El proceso denominado leer modela los dos puertos de lectura. La lectura es asíncrona. En la lista de activación están los identificadores de registro y la señal mem. Esta señal se incluye para el caso de que no se modifiquen los identificadores de registro y uno de los registros que se leen haya sido actualizado. El retardo de lectura, retardo del multiplexor utilizado para seleccionar el registro que se lee, se especifica al efectuar la acción de lectura. La lectura de un registro se observa después de que hayan transcurrido 10 ns.

El registro cuyo índice es cero es un registro especial. Siempre contiene el valor cero.

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity BR is -- banco de registros con tres puertos
generic (n : positive := 4; nr : positive := 4; nrb : positive := 2);
port (   reloj, PE : in      std_logic;
        IDL1, IDL2, IDE : in std_logic_vector (nrb-1 downto 0);
        DE: in          std_logic_vector (n-1 downto 0);
        Le1, Le2: out     std_logic_vector (n-1 downto 0) );
end BR;

architecture behav of BR is
type ramtype is array (0 to nr-1 ) of std_logic_vector (n-1 downto 0);
signal idedeco: std_logic_vector(nrb-1 downto 0);
begin -- Banco de registros de 3 puertos. Los puertos de lectura son combinacionales
      idedeco <= IDE after 8 ns; -- decodificación identificador registro de escritura
escribir: process (reloj)
begin
  if (reloj'event and reloj = '1') then
    if (PE = '1') then mem(conv_integer(idedeco)) <= DE after 14 ns;
    end if;
  end if;
end process;

leer: process (IDL1, IDL2, mem)
begin
  if (conv_integer (IDL1) = 0) then Le1 <= X"0" after 10 ns;
  -- registro 0 almacena valor 0
  else Le1 <= mem(conv_integer (IDL1)) after 10 ns;
  end if;
  if (conv_integer(IDL2) = 0) then Le2 <= X"0" after 10 ns;
  else Le2 <= mem(conv_integer(IDL2)) after 10 ns;
  end if;
end process;
end behav;

```

Figura 3.18 Especificación en VHDL de un banco de registros.

Ubicación de las fases de una operación aritmética en un ciclo

El núcleo del camino de datos del procesador se utiliza para efectuar operaciones aritméticas. Los registros almacenan los valores que alimentan las entradas del sumador y la salida del sumador se almacena en un registro del banco de registros.

En una operación aritmética se distinguen las siguientes fases: a) leer dos valores del banco de registros, b) efectuar la suma y c) almacenar el resultado de la suma en el banco de registros.

Nota: En general supondremos que las operaciones de lectura y escritura en el banco de registros no se pueden efectuar en paralelo, ya que en algún caso se puede querer leer el mismo registro que acaba de escribirse en el flanco ascendente de la señal de reloj.

La señal de reloj determina el instante en que se actualiza un registro del banco de registros. Supondremos que este instante de tiempo es el flanco ascendente de la señal de Reloj. Entonces, en un ciclo de reloj debe efectuarse una operación aritmética.

La Figura 3.19 muestra las fases descritas previamente y su relación con un ciclo de la señal Reloj. Suponemos el peor caso. Esto es, que una operación puede leer el contenido del registro que actualiza la operación previa. Unicamente se muestran los retardos de almacenar el resultado en el banco de registros, leer los datos del banco de registros y el retardo del sumador. Las fases de una operación se han marcado con el mismo tono en la trama y se identifican con una etiqueta. También suponemos que la señal PE está activada y permite la escritura en el banco de registros.

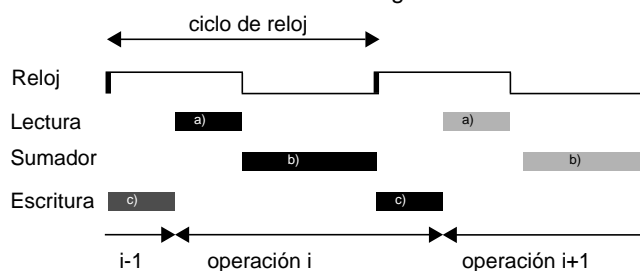


Figura 3.19 Fases de una operación aritmética y su relación con el tiempo de ciclo.

En el flanco ascendente de la señal de Reloj se actualiza el banco de registros, seguidamente se leen los datos del banco de registros y se efectúa la suma de estos datos. Posteriormente en el siguiente flanco ascendente de la señal de Reloj se efectúa la actualización del banco de registros con el resultado de la suma.

Restricciones en el tiempo de ciclo

En un circuito secuencial se distinguen elementos de memorización y lógica combinacional. Las entradas de la lógica combinacional son las salidas de los elementos de memorización. Si las salidas de los elementos de memorización permanecen estables durante el retardo de la lógica combinacional, se garantiza que se evalúa la función lógica para dichas entradas. Esto es, las señales de salida del circuito combinacional son estables mientras no se modifiquen las entradas.

En la Figura 3.20 se muestra un esquema simplificado del núcleo del camino de datos de un procesador (se muestra una rebanada de 1 bit). Para efectuar la descripción se utiliza la parte izquierda de la figura donde se ha linealizado el camino de datos, de forma que el flujo de información va de izquierda a derecha. En estas condiciones, el registro de la izquierda se asimila a la operación de lectura de un registro del banco de registros y el registro de la derecha se asimila a la operación de escritura en un registro del banco de registros. La lógica combinacional incluye el sumador y los elementos combinacionales del banco de registros.

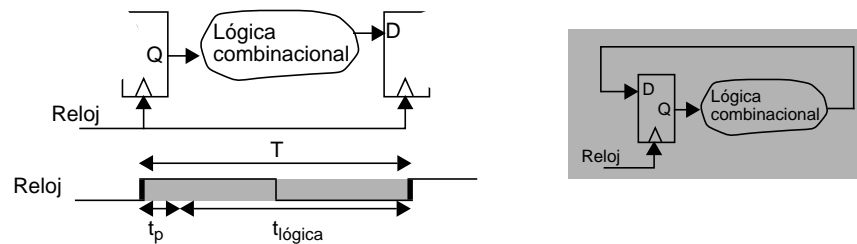


Figura 3.20 Esquema simplificado del núcleo de un procesador.

Los elementos de almacenamiento actualizan el estado en instantes prefijados de la señal de Reloj y el periodo de la señal de Reloj tiene una duración suficiente, para que la lógica combinacional evalúe señales estables en sus salidas a partir de las señales estables de sus entradas. Esta restricción establece un valor mínimo al periodo de la señal de Reloj. El tiempo mínimo de ciclo T , necesario para un funcionamiento correcto del circuito secuencial, está expresado por la siguiente relación.

$$T \geq t_p + t_{lógica}$$

donde t_p es el tiempo de propagación del registro y $t_{lógica}$ es el tiempo de retardo de la lógica combinacional. En $t_{lógica}$ se incluye el retardo de lectura cuando se accede a un banco de registros.

Características del camino de datos utilizado

En la Figura 3.21 se muestra el núcleo de un camino de datos y un módulo de control para generar los identificadores de registro (lectura y escritura).

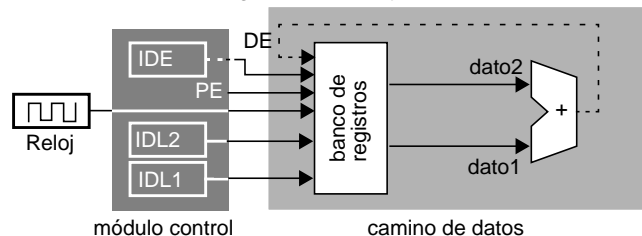


Figura 3.21 Camino de datos y módulo control.

El valor de los identificadores de registro se actualiza en cada flanco ascendente de la señal Reloj. El retardo, respecto al flanco ascendente, con que se observan los identificadores de registro en la salida del módulo control es t_{cont} . Así mismo, se supone que la señal PE está permanentemente en el nivel lógico 1 (se permite la escritura). La secuencia de identificadores de registro que genera el módulo control se muestra en la Figura 3.22.

etiqueta	secuencia repetitiva cada 4 ciclos			
	IDL1	0	1	2
	IDL2	4	5	5
	IDE	5	5	5

Figura 3.22 Secuencia de identificadores de registro que genera el módulo control.

En la Figura 3.23 se muestran los componentes del camino de datos. En una entrada del sumador se distinguen de forma explícita los componentes involucrados en una acción de lectura y en la salida del sumador los componentes involucrados en una acción de escritura.

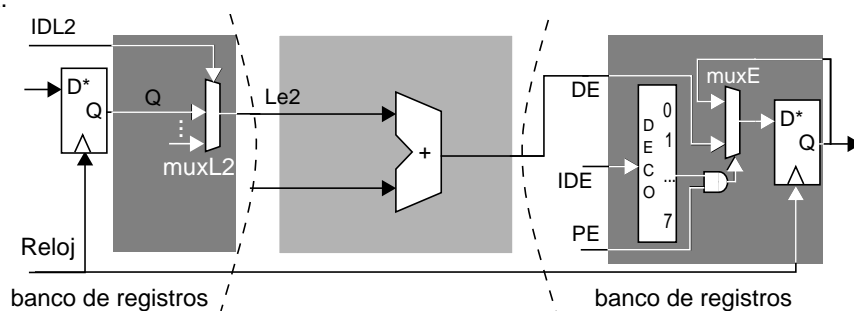


Figura 3.23 Componentes del camino de datos.

En la Figura 3.24 se indican los tiempos de retardo de los distintos elementos incluidos en el camino de datos y del módulo de control.

Componentes	Elementos	retardo
Banco de registros	Decodificador (DECO)	t_{deco}
	Multiplexor (muxL)	t_{muxL}
	Registro	t_p
	Multiplexor (muxE)	t_{muxE}
	puerta AND	t_{AND}
Sumador		t_{sum}
Generador de identificadores de registro		t_{cont}

Figura 3.24 Retardos de los componentes del camino de datos y del módulo de control.

Simulación del camino de datos

Trabajo: Abra el fichero NCD.cct

En la Figura 3.25 se muestra el circuito almacenado en el fichero NCD.cct. En este circuito se distingue un banco de registros (BR), un sumador y un módulo de control (CTR). Adicionalmente, en la parte derecha, hay ubicados elementos visualizadores que muestran las señales transportadas por los buses en representación hexadecimal.

Los identificadores de los registros de lectura se denominan IDL1 e IDL2. El contenido del registro cuyo identificador es IDL1 se transporta por el bus etiquetado como A. El bus etiquetado como B transporta el contenido del registro cuyo identificador es IDL2.

Las entradas al banco de registros para efectuar operaciones de escritura son: a) identificador de registro (IDE), b) valor (DE) y c) permiso de escritura (PE). Un registro del banco de registros se actualiza en el flanco ascendente de la señal Reloj si la señal PE está activada.

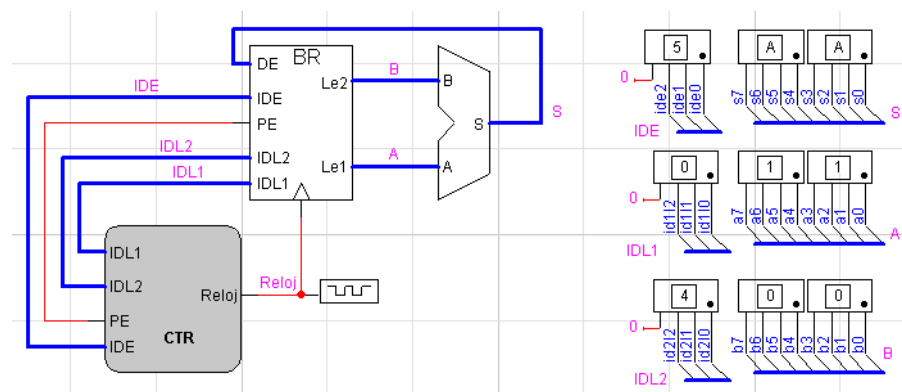


Figura 3.25 Camino de datos almacenado en el fichero NCD.cct. Los identificadores de registros utilizan 3 bits y los datos se representan con 8 bits.

El banco de registros dispone de 8 registros numerados del cero al siete y los valores que almacenan se representan con 8 bits.

Los ficheros BancoReg.dvw y control_camino.dvw contienen las especificaciones en VHDL del banco de registros y del módulo de control.

Simulación de una secuencia de operaciones

Para efectuar la simulación deben efectuarse los siguientes pasos:

- 1 Se establecen marcas en el área de señales de la ventana de tiempo. Para ello, seleccione en la paleta de herramientas de la ventana de tiempo la acción de establecer marcas ("trigger", Figura 3.26).



Figura 3.26 Paleta de herramientas de la ventana de tiempo.

Entonces se visualiza la ventana que se muestra en la Figura 3.27 y se rellena con la información que se observa en la figura.

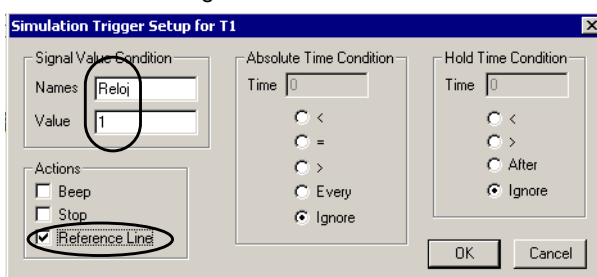


Figura 3.27 Ventana para establecer marcas en la ventana de tiempo.

Cuando se active la simulación se visualizarán marcas verticales que indican el flanco ascendente de la señal Reloj.

- 2 Se efectúa una puesta a cero (reset) utilizando la paleta de herramientas de la ventana de tiempos

Nota: Al efectuar una puesta a cero, los registros del banco toman los valores indicados en el fichero BancoReg.dvw y las salidas del módulo control son PE=0, IDL1=IDL2=IDE=7.

- 3 Desplazando el dial del orden de 1/3 de la longitud máxima se activa la simulación, estableciendo una velocidad ralentizada que ayuda a visualizar la evolución de las señales en la ventana de tiempo.

En el primer flanco ascendente de la señal de Reloj se observan los valores de los identificadores de registro que genera el módulo Control.

A partir de este instante, en el área de traza de señales de la ventana de tiempo se observa la evolución de las señales. En el caso concreto de un bus se muestra en hexadecimal el valor del grupo de señales que constituyen el bus. En la Figura 3.28 se muestra un ejemplo. Los retardos de los componentes del camino de datos no se visua-

lizan de forma explícita, hay que determinarlos a partir de cambios de valor en las señales. En este ejemplo, cuando se ha activado la simulación, los valores de los registros 0, 1, 2, 3 y 4 son, en representación hexadecimal, 11, 22, 33, 44 y 00 respectivamente.

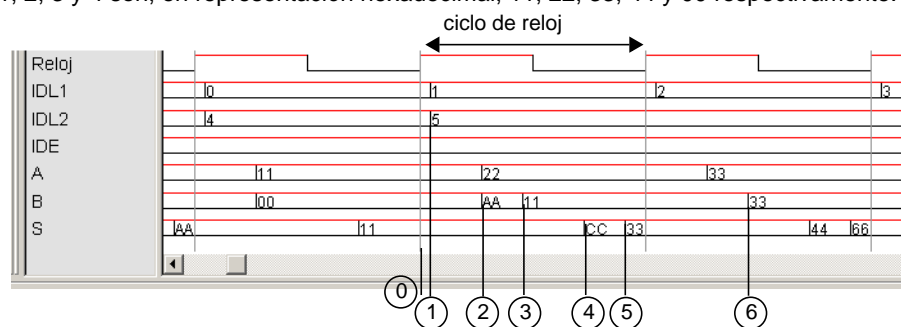


Figura 3.28 Evolución de las señales en la ventana de tiempos.


Seguidamente se muestra un análisis del ciclo de Reloj identificado en la parte superior de la Figura 3.28. Las etiquetas de las señales se observan a la izquierda de la figura. Todos los flancos ascendentes de la señal Reloj tienen como marca una línea vertical a trazos. La traza de una señal de bus es un par de líneas paralelas y una línea vertical cuando se modifica el valor previo. En la parte derecha de la línea vertical se visualiza el valor actual.

Para el desarrollo del análisis, en la Figura 3.28 se han añadido marcas con números rodeados por círculos. Los intervalos de tiempo que se describen se contabilizan a partir del flanco ascendente de la señal Reloj, marcado con ① en la Figura 3.28. En los siguientes párrafos, el número que inicia el párrafo se corresponde con la marca numérica en la Figura 3.28.

- ① Después del retardo en la generación de los identificadores de los registros (t_{cont}), se observan los identificadores de los registros que se leen, los cuales son el 1 y el 5 (IDL1, IDL2). El valor del identificador del registro donde se escribe no se observa (IDE), ya que no se ha modificado respecto al ciclo previo; es el valor 5.
- ② Después del retardo en la generación de los identificadores de los registros y del retardo del elemento muxL1 ($t_{cont} + t_{muxL}$), se observa (Bus A) el valor que se ha leído (22) del registro 1. El valor que se lee del registro 5 es el que se ha calculado dos ciclos antes (Bus B). Este caso depende del ciclo que se analiza.
- ③ Después del retardo en la generación de los identificadores de los registros, el retardo del registro y el retardo del multiplexor muxL2 ($\max(t_{cont}, t_p) + t_{muxL}$), se observa (Bus B) que el valor del registro 5 es 11, el cual es el valor calculado en el ciclo previo.
- ④ Después del retardo en la generación de los identificadores de los registros, el

retardo del multiplexor muxL1 y el retardo del sumador ($t_{cont} + t_{muxL} + t_{sum}$), en la salida del sumador (Bus S) se observa el valor CC que es la suma de los valores 22 (Bus A) y AA (Bus B). La observación de este valor concreto es debido a que el retardo de este sumador es independiente de los valores de las entradas y cuando se modifica una entrada se observa una nueva salida después del retardo.

- ⑤ Después del retardo en la generación de los identificadores de los registros, el retardo del registro, el retardo del multiplexor muxL2 y el retardo del sumador ($[\max(t_{cont}, t_p)] + t_{muxL} + t_{sum}$), en la salida del sumador (Bus S), se observa el valor 33 que es la suma de los valores 22 (Bus A) y 11 (Bus B). Este valor se conoce antes del flanco ascendente de la señal de Reloj. Por tanto, en el registro 5 (IDE) se almacena el valor 33.
- ⑥ El valor 33 se puede observar en el siguiente ciclo en el Bus B ya que se suma al registro 2.

- 4 La simulación puede pararse en cualquier instante de tiempo seleccionado en la paleta de herramientas de la ventana de tiempo la siguiente opción .

Trabajo: Active la simulación siguiendo los pasos descritos y efectúe un análisis de las señales en la ventana de tiempos. El valor preestablecido del contenido de los registros 0, 1, 2 y 3 puede ser distinto al descrito en el análisis previo.

Modificación del periodo de la señal Reloj

Con la simulación desactivada se selecciona el elemento Reloj en el circuito y se efectúan los pasos que se muestran en la parte izquierda de la Figura 3.29.

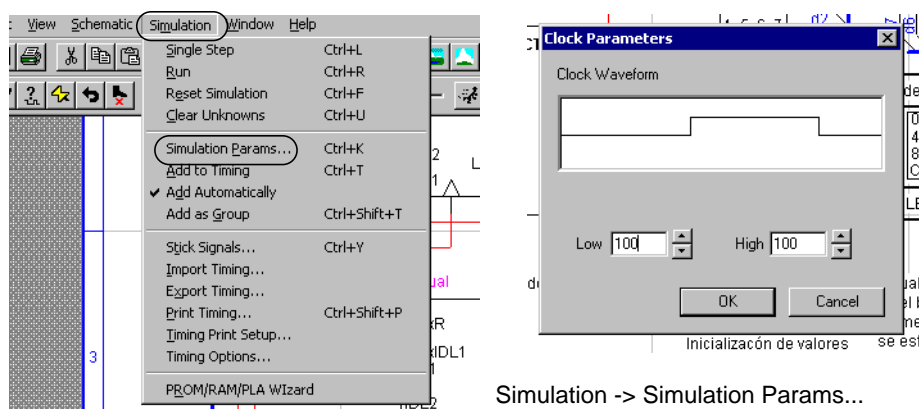


Figura 3.29 a) pasos para modificar el periodo de la señal Reloj y b) ventana donde se establece el valor del periodo.

Una vez efectuados los pasos se visualiza la ventana de la derecha de la Figura 3.29, que permite establecer independientemente la duración de tiempo en el nivel lógico 0 y en el nivel lógico 1. Mediante el teclado o mediante las pestañas de incremento o decremento de unidad en unidad se modifica el valor numérico de duración del nivel lógico que interese.

Posteriormente se activa la simulación y se simula con el periodo establecido para la señal Reloj.

Descripción de un funcionamiento incorrecto. En un funcionamiento correcto el valor de la suma (etiqueta S) debe visualizarse en la ventana de tiempos antes del flanco ascendente del siguiente ciclo. En caso contrario, debido a que el periodo establecido es menor que el necesario, una evolución posible de las señales se muestra en la Figura 3.30.

En la Figura 3.30 la descripción de los instantes de tiempo con las marcas ①, ②, ③ y ④ es igual a la efectuada en la Figura 3.28 que mostraba la evolución de las señales.

La marca de interés es la ⑤. Está después del flanco ascendente de la señal Reloj y el valor 33 es la suma del contenido de los registros 1 (IDL1) y 5 (IDL2), el cual es respectivamente 22 (Bus A) y 11 (Bus B).

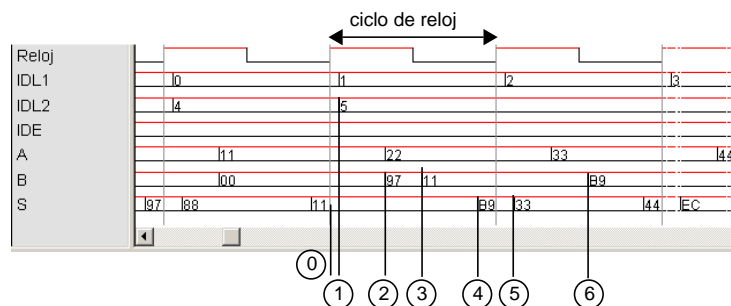


Figura 3.30 Observación de un funcionamiento incorrecto.

Como el valor de la suma (Bus S) en el flanco ascendente de la señal de Reloj es B9 (22 + 97), en el registro 5 se almacena un valor incorrecto. Esto puede observarse en el siguiente ciclo (marca ⑥). Se leen los registros 2 (IDL1) y 5 (IDL2) y su contenido es 33 (Bus A) y B9 (Bus B) respectivamente. Los valores que se deberían leer en un funcionamiento correcto son 33 y 33.

Trabajo: Compruebe el periodo calculado en las preguntas.

Apéndice 3.1: Circuitos secuenciales

Los circuitos secuenciales se pueden dividir en dos clases básicas: síncronos y asíncronos. Nosotros nos centraremos en los circuitos síncronos. Un circuito síncrono facilita la verificación y la comprobación del funcionamiento.

Circuito síncrono. Un circuito síncrono utiliza los registros como elementos de memorización y todos los registros están controlados por un único reloj.

El diagrama básico de un circuito síncrono se muestra en la Figura 3.31. El elemento de memorización (registro) conocido como registro de estado es un conjunto de registros, sincronizados por una única señal de reloj. La salida del registro es la señal de estado que representa el estado interno del sistema. La lógica próximo_estado es un circuito combinatorial que determina el próximo estado. La lógica salida es otro circuito combinatorial.

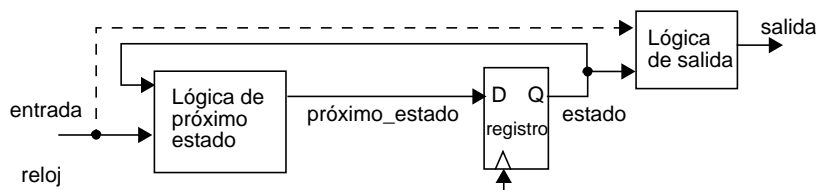


Figura 3.31 Modelo esquemático de un circuito secuencial síncrono.

En la Figura 3.31 las señales de salida dependen del estado y de las señales de entrada (línea de trazos). Este tipo de circuito síncrono se denomina autómata de Mealy. Si las señales de salida sólo dependen del estado el circuito síncrono se denomina autómata de Moore. Nosotros nos centraremos en estos últimos.

El funcionamiento del circuito es el siguiente.

- En el flanco ascendente de la señal de reloj, el valor de la señal próximo_estado (D) se muestrea y se propaga a la salida (Q), siendo ahora el nuevo estado. El valor se almacena en el elemento de memorización (registro) y durante todo el periodo de la señal de reloj no se modifica. Representa el estado del sistema.
- Las lógicas combinatoriales próximo_estado y salida determinan respectivamente el próximo estado y la salida.
- En el siguiente flanco ascendente de la señal de reloj se repite el proceso.

En la Figura 3.32 se muestra un diagrama temporal de las relaciones de dependencia entre los retardos de los componentes de un circuito secuencial síncrono. La magnitud de los retardos que se representa es un ejemplo.



Figura 3.32 Diagrama temporal, en un periodo de la señal de reloj, de un circuito síncrono. Relaciones de dependencia entre los retardos.

Dado un circuito síncrono podemos distinguir, de forma informal, varios tipos: a) circuito secuencial regular, b) circuito secuencial aleatorio y c) circuito secuencial combinado. La diferencia entre el primer tipo y el segundo es la complejidad de las transiciones entre estados y la complejidad de las lógicas combinacionales. En un circuito secuencial regular la representación binaria de los estados usualmente tiene una interpretación. Ejemplos del primer tipo son contadores y registros de desplazamiento. Los circuitos secuenciales del segundo tipo se denominan máquinas de estados finitos. En el tercer tipo se incluyen los circuitos que constan de elementos del primer tipo y del segundo tipo. En este tercer caso, la máquina de estados finitos se utiliza para controlar el circuito secuencial regular.

Diseños simples

Registro con puesta a cero asíncrona. En la tabla de la Figura 3.33 se representa el funcionamiento del circuito que se muestra en la parte derecha de la misma figura. El símbolo Q^* indica el futuro valor de la salida y el símbolo Q indica el valor actual de la salida. La señal PE (permiso de escritura) sólo se tiene en cuenta en el flanco ascendente de la señal de reloj. Esto significa que la señal PE está sincronizada con la señal de reloj. En el flanco ascendente de la señal de reloj se muestrean las señales PE y D. Si PE = 0 se mantiene el valor en el elemento de memorización. En caso contrario, cuando PE = 1, el funcionamiento es el de un registro. Esto es, la entrada se propaga a la salida.

reloj	PE	Q^*
0	-	Q
1	-	Q
flanco	0	Q
flanco	1	D

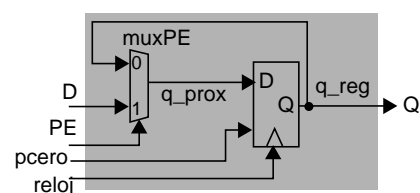


Figura 3.33 Registro con permiso de escritura síncrono.

En la parte derecha de la Figura 3.33 se ha mostrado el diagrama de un registro con permiso de escritura. Observemos que podemos asimilar elementos de este diagrama con elementos mostrados en la Figura 3.31. La lógica del próximo_estado es el multiplexor y no existe lógica para determinar la señal de salida, ya que es directamente la salida del registro.

En la Figura 3.34 se muestra una descripción VHDL del registro con permiso de escritura síncrono y puesta a cero asíncrona. Se distinguen tres partes. El proceso denominado reg modela el elemento de memorización denominado registro. El elemento mux se modela mediante una sentencia de asignación de señal de forma condicional¹, las cuales se describen en un Apéndice 2.2 de la Práctica 2. La sentencia de asignación de señal modela la lógica de salida. La lógica de salida es un cable que conecta la salida del registro con el puerto de salida.

Registro con puesta a cero asíncrona

```
library IEEE;
use IEEE.std_logic_1164.all;

entity regen is
port ( reloj, pcero, PE, D: in std_logic;
      Q: out std_logic );
end regen;

architecture trespart of regen is
signal q_reg: std_logic;
signal q_prox: std_logic;
begin
--Estado
reg: process (reloj, pcero)
begin
    if (pcero = '1') then
        q_reg <= '0' after 2 ns;
    elsif (reloj'event and reloj = '1') then
        q_reg <= q_prox after 2 ns;
    end if ;
end process;
-- Lógica de próximo estado
q_prox <= D after 5 ns when PE = '1' else
q_reg after 5 ns;
-- Lógica de salida
q <= q_reg;
end trespart;
```

Figura 3.34 Descripción VHDL de un registro con puesta a cero asíncrona distinguiendo los elementos combinacionales y secuenciales.

En la descripción VHDL de la Figura 3.34 el registro tiene un retardo de actualización de 2 ns y la lógica de próximo estado tiene un retardo de 5 ns.

La descripción VHDL de la Figura 3.34 sigue el modelo de la Figura 3.31. En este modelo se pueden identificar de forma clara los elementos combinacionales y los elementos secuenciales. Esto permite comprobar y verificar el funcionamiento de los

1. Esta sentencia de asignación de señal de forma condicional puede sustituirse por un proceso.

componentes de forma aislada. En la Figura 3.35 se muestra una descripción VHDL del mismo tipo de registro donde se entremezclan los distintos elementos, lo cual dificulta la comprobación y verificación.

```

Registro con permiso de escritura síncrono
library IEEE;
use IEEE.std_logic_1164.all;

entity regen is
port ( reloj, pcero, PE, D: in std_logic;
      Q: out std_logic );
end regen;

architecture beha of regen is
begin
  regen: Process (reloj, pcero)
  begin
    if (pcero = '1') then
      Q <= '0' after 2 ns;
    elsif (reloj'event and reloj = '1') then
      if PE = '1' then
        Q <= D after 5 ns;
      end if ;
    end if ;
  end process ;
end beha;

```

Figura 3.35 Descripción VHDL de un registro con permiso de escritura utilizando un único proceso.

Contador binario. Un contador binario pasa por una secuencia de estados cuya codificación en binario se puede interpretar como números naturales. Por ejemplo un contador binario de 2 bits repite indefinidamente la secuencia: 00, 01, 10, 11.

Un contador binario tiene un registro que almacena n bits y su salida se interpreta como un número natural codificado en base 2. El contador incrementa el contenido del registro en cada ciclo de la señal de reloj; contando desde 0 hasta $2^n - 1$. La cuenta se repite indefinidamente.

En la Figura 3.36 se muestra un esquema de un contador binario. Comparando esta figura con la Figura 3.31 podemos identificar que la lógica próximo_estado es un incrementador, el cual calcula el nuevo valor del próximo estado. No existe lógica para determinar la señal de salida, ya que es directamente la salida del registro.

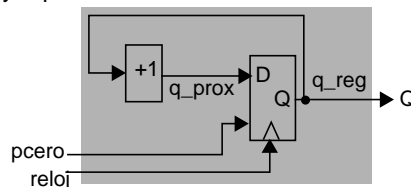


Figura 3.36 Contador binario.

En la Figura 3.37 se muestra una descripción VHDL del contador binario donde se distinguen tres partes. El proceso denominado reg modela el elemento de memorización denominado registro. Después del proceso, la primera sentencia de asignación de señal modela el incrementador unidad. La última sentencia de asignación de señal modela la lógica de salida. La lógica de salida es un cable que conecta la salida del registro con el puerto de salida.

```

Contador binario

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity contador is
port ( reloj, pcero : in  std_logic;
      Q: out          std_logic_vector (3 downto 0) );
end contador;

architecture trespart of contador is
signal q_reg: std_logic_vector (3 downto 0);
signal q_prox: std_logic_vector (3 downto 0);
begin
--Estado
reg: Process (reloj, pcero)
begin
    if (pcero ='1') then
        q_reg <= (others => '0') after 2 ns;
    elsif (reloj'event and reloj = '1') then
        q_reg <= q_prox after 2 ns;
    end if ;
end process ;
-- Lógica de próximo estado
q_prox <= q_reg +1 after 8 ns ;
-- Lógica de salida
q <= q_reg;
end trespart;

```

Figura 3.37 Descripción VHDL de un contador binario.

Contador módulo. En la Figura 3.38 se muestra el esquema de circuito de un contador módulo 7. La lógica próximo estado es un incrementador, un multiplexor y un comparador.

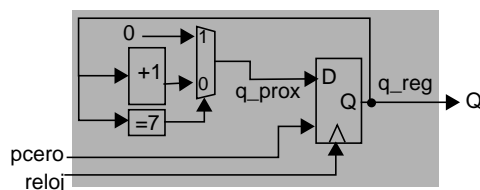


Figura 3.38 Contador módulo.

En la Figura 3.39 se muestra la descripción VHDL siguiendo el modelo de la Figura 3.31 donde se identifican tres partes: a) estado, b) lógica de próximo estado y c) lógica de salida. El límite del contador se describe mediante una constante. Para describir la lógica de próximo estado se utiliza un incrementador, un comparador y un multiplexor. La condición que se evalúa es el valor del contador (comparación).

Contador módulo
<pre> library IEEE; use IEEE.std_logic_1164.all; use IEEE.std_logic_arith.all; entity cntmod is port (reloj, pcero : in std_logic; Q: out std_logic_vector (3 downto 0)); end cntmod; architecture trespart of cntmod is constant modu: integer := 7; signal q_reg: std_logic_vector (3 downto 0); signal q_prox: std_logic_vector (3 downto 0); begin --Estado reg:Process (reloj, pcero) begin if (pcero ='1') then q_reg <= (others => '0') after 2 ns; elsif (reloj'event and reloj = '1') then q_reg <= q_prox after 2 ns; end if ; end process ; -- Lógica de próximo estado q_prox <= (others => '0') after 10 ns when conv_integer(q_reg) = modu else q_reg +1 after 10 ns ; -- Lógica de salida q <= q_reg; end trespart; </pre>

Figura 3.39 Descripción VHDL de un contador módulo.