Contenido

Práctica 3
Núcleo del camino de datos de un procesador y buffer circular

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.

Una especificación de un circuito como el descrito se efectúa utilizando una descripción de transferencia entre registros (RTL, “Register Transfer Level”). En concreto, se describe la transferencia de datos y su temporalidad en componentes lógicos simples (registros, banco de registros, ALU, etc). En una descripción RTL se separa el camino de datos del control del mismo. El camino de datos incluye componentes lógicos y su interconexión. La unidad o módulo de control genera las señales de control con la temporalidad adecuada. Esta temporalidad se adecua a la metodología de reloj utilizada.

Además, se diseña un buffer circular, analizando el protocolo de las interfaces y el control del mismo.

Respecto al lenguaje VHDL se introduce la descripción de un modelo de comportamiento 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. En el Apéndice 3.3 se describe la especificación en VHDL de circuitos secuenciales básicos, distinguiendo la lógica de estado, de salida y de próximo estado. Como ejemplos se muestran un registro y un contador binario. En esta práctica se utiliza, por ejemplo, para diseñar el control de un camino de datos.

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.

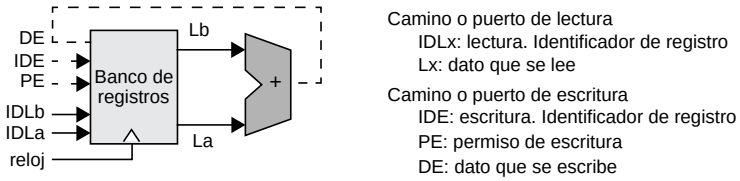


Figura 3.1 *Banco de registros: Entradas y salidas.*

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 estabilice 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. El almacenamiento es síncrono con la señal de reloj.

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, que utilizaremos, tiene dos caminos de lectura y uno de escritura. El camino de datos de escritura, incluyendo 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 (puerto de lectura, Figura 3.1): como entrada se utiliza el identificador del registro que se quiere leer (IDLx) y como salida se obtiene el valor que almacena el registro (Lx). El número de bits del identificador depende del número de registros. La lectura es combinacional.
- Camino de acceso de escritura (puerto de escritura, Figura 3.1): como entradas se utilizan el identificador del registro que se quiere escribir (IDE) y el valor que se quiere almacenar (DE). 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 0 y en el otro subintervalo de tiempo el nivel lógico es 1.

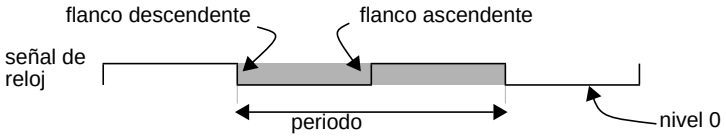


Figura 3.2 *Señal de reloj.*

Un cambio de nivel lógico en la señal de reloj se denomina flanco. El flanco se denomina ascendente cuando se pasa de nivel 0 a nivel 1 y descendente en caso contrario.

Forma o aspecto de la señal de reloj. La forma de la señal de reloj se relaciona con la duración de los subintervalos de tiempo en que permanece la señal de reloj en el nivel lógico 0 y en el nivel lógico 1 (Figura 3.3). Cuando los dos subintervalos de tiempos son de la misma duración se denomina una señal de reloj cuadrada. En caso contrario se denomina rectangular.

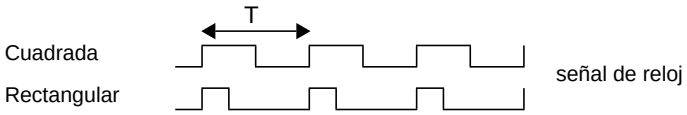


Figura 3.3 *Aspecto de la señal de reloj.*

Una operación típica en un camino de datos (Figura 3.1) lee dos registros del banco de registros, utiliza los valores leídos como entradas del sumador y almacena el resultado en un registro del banco de registros.

El funcionamiento del núcleo del camino de datos de un procesador es síncrono. Esto es, la escritura en el banco de registros está bajo el control de una señal de reloj. La decisión del instante en el cual se actualiza el estado de los elementos de almacenamiento se denomina metodología de reloj.

Metodología de reloj. Define cuándo los elementos de memorización pueden leerse y cuándo pueden escribirse.

Es importante distinguir los instantes en que se lee un registro de los instantes en que se escribe. Si las dos acciones se efectúan de forma concurrente, el valor de la lectura puede ser el valor antiguo, el que se está escribiendo o una mezcla de ambos. Esta impredecibilidad no es tolerable y la metodología de reloj se diseña para prevenir esta circunstancia.

La metodología de reloj que utilizaremos es por flanco. Esto es, se utiliza uno de los instantes de cambio de nivel lógico en la señal de reloj para actualizar los elementos de memorización de un circuito secuencial. En concreto utilizaremos el instante de cambio de nivel lógico 0 a nivel lógico 1.

Registro

Un registro es un elemento básico de almacenamiento que actualiza su estado en el flanco ascendente de la señal de reloj. Un registro elemental almacena un bit. Entonces, para construir un registro de n bits se agrupan n registros elementales y se accede a ellos como una unidad. Esto es, cuando se accede al registro se leen o escriben los n bits.

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.

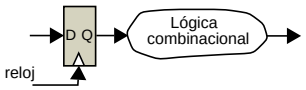


Figura 3.4 *Registro de 1 bit.*

Parámetros de tiempo asociados a un registro. Existen tres parámetros importantes, dos de ellos son relativos a que la señal de entrada debe ser estable un intervalo de tiempo antes y después del flanco de la señal reloj (Figura 3.5). Estos intervalos de tiempo se denominan respectivamente tiempo de estabilidad (t_e, "set_up time") y tiempo de mantenimiento (t_m, "hold time") y supondremos que su valor es cero[1].

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.

1. La ventana temporal que determinan estos intervalos de tiempo, alrededor del flanco de la señal de reloj que se utiliza para actualizar el estado, garantiza un funcionamiento correcto del dispositivo de almacenamiento. En el Apéndice 3.4 se efectúa un análisis detallado de los mismos.

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.

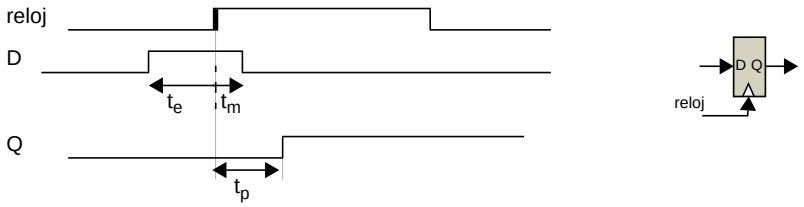


Figura 3.5 *Escritura en un registro. El retardo de propagación es t_p. Los tiempos de estabilidad y mantenimiento son t_e y t_m respectivamente.*

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 y en particular RTL. En esta sección se presenta la utilización del constructor “process” para describir el comportamiento de circuitos, en terminos de sentencias secuenciales y obtener un modelo que describe de forma abstracta el comportamiento[2].

Recordatorio. 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];
```

2. El constructor “process” y las sentencias secuenciales que se pueden utilizar en el cuerpo del mismo han sido descritas en la Práctica 2. Todos los processos se ejecutan una vez al iniciarse la simulación.

Modelo de comportamiento de un elemento combinacional: multiplexor

Un constructor “process” puede utilizarse para describir lógica combinacional. Para ello, en la lista de activación deben incluirse todas las señales de entrada. En la Figura 3.6 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 2.9 de la Práctica 2.

Notemos que la especificación VHDL de la Figura 3.6 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 compor 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 compor;
```

Figura 3.6 *Especificación VHDL, mediante el constructor “process”, de un multiplexor.*

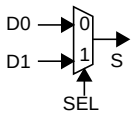
Cuando se describe un circuito combinacional debe cuidarse que se establezca un valor para todas las señales, que aparecen a la izquierda de una sentencias de asignación de señal, en todas las posibles ramas del código. En concreto, en la especificación de un multiplexor, si no se asigna un valor en todos los caminos se infiere una celda[3].

La diferencia entre las dos codificaciones del multiplexor de la Figura 3.7, está en la especificación de una de las asignaciones, cuando la señal SEL es igual a cero. La especificación de la parte derecha es factible, ya que el valor de una señal se actualiza únicamente al final de la ejecución del proceso, o cuando se suspende la ejecución y la asignación por defecto (antes de la sentencia if) es modificada por la especificada cuando se cumple la condición (SEL =1).

3. Posteriormente se describe en detalle la descripción de elementos de almacenamiento.

Multiplexor1

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is
port (   D0, D1: in  std_logic ;
         SEL :  in   std_logic ;
         S: out      std_logic );
end mux;
architecture compor of mux is
begin
process (D0, D1, SEL)
begin
        if (SEL ='1' ) then
            S <= D1 after 10 ns;
        else
            S <= D0 after 10 ns;
        end if;
end process;
end compor;
```



Multi9plexor2

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is
port (   D0, D1: in   std_logic ;
         SEL : in    std_logic ;
         S: out      std_logic );
end mux;
architecture compor of mux is
begin
process (D0, D1, SEL)
begin
        S <= D0 after 10 ns;
        if (SEL ='1' ) then
            S <= D1 after 10 ns;
        end if;
end process;
end compor;
```

Figura 3.7 *Dos forma de especificar un multiplexor.*

Simulación de una descripción VHDL de un circuito combinacional. La lista de actividad de un proceso que describe un circuito combinacional debe incluir todas las señales leídas dentro del proceso. Durante la simulación, un proceso sólo será ejecutado si se produce un evento en al menos una de las señales presentes en la lista de actividad.

Síntesis de una descripción VHDL de un circuito combinacional[4]. Una herramienta de síntesis se centra en el diseño lógico y tiene en cuenta tres tipos básicos de lógica: lógica combinacional, b) celda (elemento de almacenamiento gobernado por nivel), c) registro (elemento de almacenamiento gobernado por flanco) y d) algunos tipos de memoria estática (RAM). Durante el proceso de síntesis, la descripción VHDL se traduce en un interconexionado de elementos lógicos. La herramienta de síntesis infiere los componentes hardware analizando la descripción VHDL. Dada una descripción en VHDL de un circuito combinacional, una herramienta de síntesis, usualmente, no tiene en cuenta la lista especificada por el usuario y supone que todas las señales que se leen (entradas) están presentes en la lista de actividad. La herramienta genera una descripción en VHDL del circuito que ha inferido[5].

4. Cuando se escribe código VHDL hay que tener en cuenta que se está diseñando hardware. No se está escribiendo un programa para un computador. Las sentencias en VHDL implican un determinado hardware. En este sentido, si el código VHDL no es “bueno”, el hardware que elabore la herramienta de síntesis puede ser poco eficiente y costoso. Este curso no se centra en la síntesis, pero es necesario tener en cuenta las consideraciones que se describen para un correcto funcionamiento del circuito.

Diferencias en los resultados de simulación. Un simulador de VHDL tiene en cuenta la lista de actividad. Entonces, el simulador se ajusta a la especificación en VHDL. Por tanto, se puede producir una disfunción entre la simulación del código VHDL escrito por el ususario y el código, también VHDL, generado en el proceso de síntesis[6].

Por ejemplo, en el multiplexor de la Figura 3.7, si sólo se especifica la señal SEL en la lista de actividad, la simulación de la descripción VHDL, efectuada por el ususario, sólo produce resultados esperados cuando el cambio de una señal de entrada (D0, D1) coincide con un cambio en la señal SEL. Los eventos debidos a las señales de entrada no serán tenidos en cuenta en la simulación. Sin embargo, el código VHDL generado por el proceso de síntesis, tiene en cuenta todas las señales leídas y el resultado de la simulación es el esperado. Por tanto, es necesario seguir una serie de pautas para que exista coincidencia, en cuanto al comportamiento, entre el código de usuario y el código que genera la herramienta de síntesis[7].

Modelo de comportamiento de un elemento secuencial: celda

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 es inferido 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 memorización[8].

Consideremos la especificación de un multiplexor en VHDL mostrada en la parte izquierda de la Figura 3.8. En la especificación del multiplexor, a la señal Q se le asigna un valor en todas las posibles ramas del constructor “if”. En la parte derecha de la misma

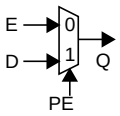
5. Por ejemplo, la palabra clave “after” tampoco es tenida en cuenta por la herramienta de síntesis. Entre otras, tampoco se tienen en cuenta inicializaciones de señales y variables y las operaciones de acceso a ficheros. Otro ejemplo es una señal declarada como “integer”. La herramienta de síntesis debe inferir un hardware que efectúe una representación binaria equivalente. Si no se especifica el rango, la herramienta de síntesis supone el máximo rango, el cual es usualmente 32 bits.
6. Recordemos que una herramienta de síntesis sólo sintetiza un subconjunto de las especificaciones que se pueden efectuar en VHDL.
7. En la lista de activación deben incluirse todas las señales de entrada. Para los resultados intermedios deben utilizarse variables. Una variable debe escribirse antes de ser leída en el cuerpo de un proceso.
8. De forma no consciente se pueden crear elementos de memorización (celda, “latch”). Una posibilidad para que no ocurra es asignar a todas las señales valores por defecto en el inicio del cuerpo de la arquitectura

Figura 3.8 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 3.8 se denomina celda. 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)[9].

Multiplexor

```
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 compor 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 compor;
```



Celda

```
library ieee;
use ieee.std_logic_1164.all;
entity celda_D is
port (    D : in     std_logic ;
          PE : in    std_logic ;
          Q: out     std_logic );
end D_latch;
architecture compor of celda_D is
begin
process (D, PE)
begin
        if (PE ='1' ) then
            Q <= D after 10 ns;
        end if;
end process;
end compor;
```

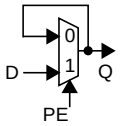


Figura 3.8 *Especificación VHDL de un multiplexor y de una celda ("latch").*

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".

9. Observemos que la lista de activación del proceso incluye las dos señales (D y PE). Cuando cualquiera de estas señales se modifica, el proceso se vuelve a ejecutar.

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')[10].

En la parte izquierda de la Figura 3.9 se muestra la especificación VHDL de un registro. En la lista de activación sólo está la señal reloj[11]. Por otro lado, 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[12]. 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)

```
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 compor 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 compor;
```

Registro con señal de puesta a cero síncrona

```
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 compor 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 compor;
```

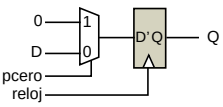


Figura 3.9 *Especificación VHDL de un registro. Parte derecha, registro con señal de puesta a cero síncrona.*

10. En la librería std_logic_1164 se dispone de las funciones "rising_edge" y "falling_edge" que detectan respectivamente un flanco ascendente o descendente cuando el objeto es de tipo std_logic.
11. El proceso se ejecuta en cada cambio de la señal de reloj.
12. Si no se especifica la condición reloj'event la herramienta de síntesis puede inferir una celda en lugar de un registro.

En la parte derecha de la Figura 3.9 se muestra la especificación VHDL de un registro con señal de puesta a cero síncrona con la señal de reloj[13]. El sincronismo se determina imbricando la condición de puesta a cero con la condición de flanco ascendente.

En la Figura 3.10 se muestra la especificación VHDL de un registro con señal de puesta a cero síncrona con la señal de reloj y señal de permiso de escritura.

Registro con señal de puesta a cero síncrona y permiso de escritura

```
library ieee;
use ieee.std_logic_1164.all;
entity reg is
port (  D : in                  std_logic_vector (3 downto 0);
        reloj, pcero, PE: in    std_logic ;
        Q: out                  std_logic_vector (3 downto 0) );
end reg;
architecture compor of reg is
begin
process (reloj)
        begin
        if (reloj='1' and reloj'event) then
            if pcero='1' then
                Q <= (others => '0') after 10 ns;
            elsif PE = '1' then
                Q <= D after 10 ns;
            end if;
        end if;
end process;
end compor;
```

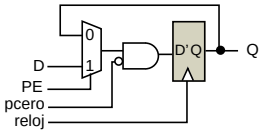


Figura 3.10 *Especificación VHDL de un registro con señal de puesta a cero síncrona y señal de permiso de escritura.*

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).

13. Si la puesta a cero es asíncrona, hay que incluir la señal pcero en la lista de activación. Adicionalmente, no se puede prescindir de la condición reloj'event, ya que el proceso es activado al activarse pcero.

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 2.9 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. En el Apéndice 3.13 se presenta un esquema estructural de un banco de registros utilizando componentes lógicos.

La salida de un registro es un vector de n bits, por tanto es necesario un multiplexor por cada bit. En la Figura 3.11 se muestra un esquema de un banco de registros con 32 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.

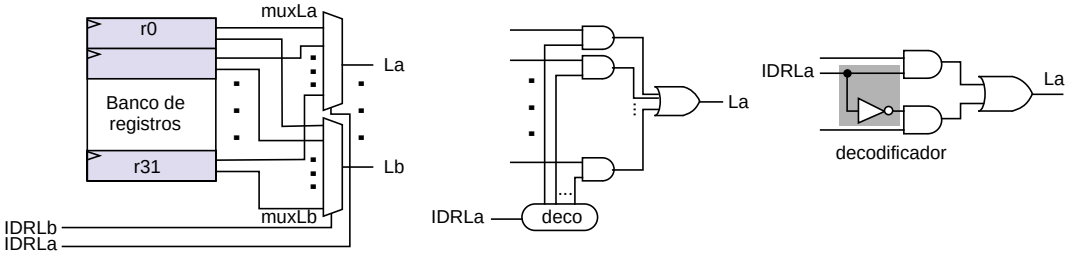


Figura 3.11 *Banco de registros. En el centro se muestra la lógica de lectura. En la parte derecha se muestra el detalle del multiplexor utilizado en la lectura de un banco de registros con dos registros.*

En la lógica de un multiplexor se incluye la función de decodificación del identificador de registro. En la parte central de la Figura 3.11 se muestra el detalle del multiplexor al tener en cuenta el decodificador. En la parte derecha de la Figura 3.11 se muestra el detalle del decodificador para el caso 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 3.12 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 de 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 de 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 de reloj, como la señal PE está desactivada, no se almacena el valor de la entrada D en el registro.

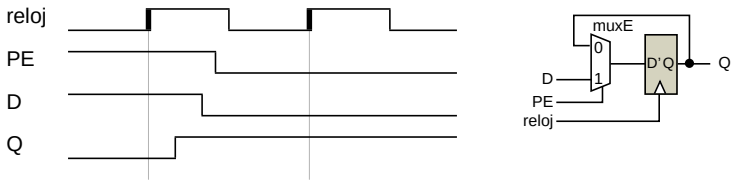


Figura 3.12 *Banco de registros. Lógica de escritura.*

En la Figura 3.13 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, la señal de permiso de escritura debe estar activada.

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

Figura 3.13 *Registro con permiso de escritura síncrono.*

Circuito secuencial. 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 3.3 de esta práctica se muestra la descripción VHDL del circuito de la parte derecha de la Figura 3.12, identificando de forma explícita estos elementos (Figura 3.61, página 219).

En la Figura 3.14 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 de reloj indica el instante en el que se actualiza el registro, si la señal permiso de escritura (PE) está activada. En Apéndice 2.9 de la Práctica 2 se describe la función lógica de un decodificador. Así mismo se muestran varias especificaciones en VHDL.

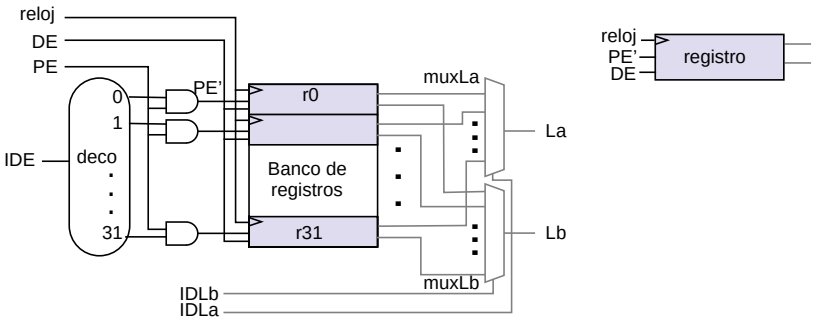


Figura 3.14 *Detalle de los elementos de un banco de registros.*

Parámetros de tiempo

En el diagrama de tiempos de la Figura 3.15 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_{muxLa}) 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 3.15.

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. Por tanto, el retardo de propagación de la lógica previa debe finalizar como muy tarde en ese instante de tiempo[14].

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. Posteriormente hay que esperar el retardo de lectura.

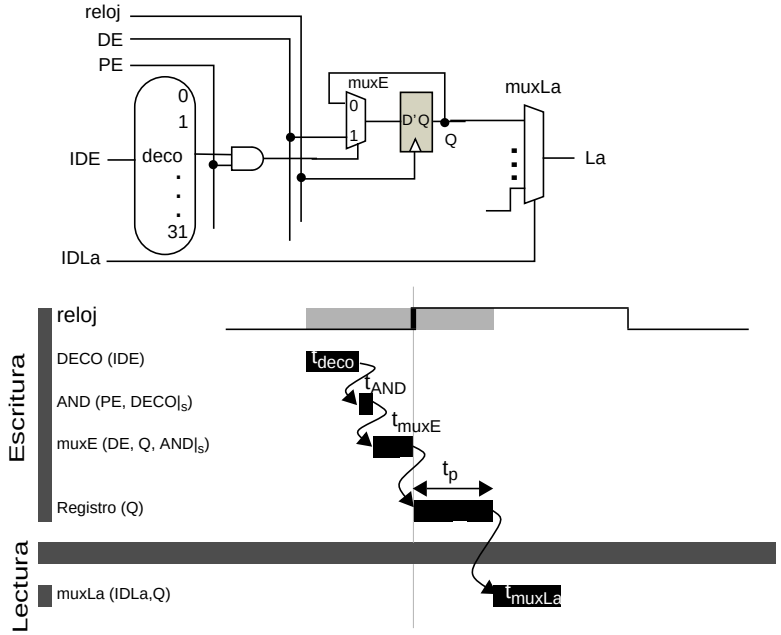


Figura 3.15 *Retardos en una operación de escritura en un registro del banco de registros. $DECO|_s$ y $AND|_s$ son las salidas del decodificador y la puerta AND respectivamente.*

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 la señal de la entrada seleccionada en el multiplexor muxE sean estables[15].

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.

14. Recordemos que estamos suponiendo que t_e y t_m son igual a cero.
15. Los retardos t_{deco}, t_{and} y t_{muxE} se indican en el instante más tardío en el cual se pueden producir. El retardo t_p está asociado a la propagación de la señal en el registro y empieza en el flanco de reloj. El retardo t_{muxLa} sigue en secuencia al retardo t_p.

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 ED 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 (muxLa). En la Figura 3.15 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 IDLx (IDLa) sea estable en el mismo instante de tiempo. Posteriormente se observa el retardo del multiplexor muxLa y la señal La es válida a partir de $t_p + t_{muxLa}$ 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** MY_WORD **is** array (0 to 15) of **std_logic**;	**type** array_name **is** array (indexing scheme) of type;

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 entero[16]). La sintaxis para especificar un rango es:

Sintaxis
simple_expresion (**to** | **downto**) simple_expresion

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

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

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.

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 3.16 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 un elemento (registro), de la señal que modela el banco de registros (“mem”), se efectúa una conversión de tipo” std_logic_vector” a entero.

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 que se modifica. 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).

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.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 compor of BR is
type ramtype is array (0 to nr-1 ) of std_logic_vector (n-1 downto 0);
signal mem: ramtype;
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((to_integer(unsigned(idedeco))) <= DE after 14 ns;
                  end if;
            end if;
            end process;
--          Lectura
            Le1 <= (others =>'0') when (to_integer(unsigned(IDL1)) = 0) else
                              mem(to_integer(unsigned(IDL1))) after 10 ns;
            Le2 <= (others =>'0') when (to_integer(unsigned(IDL2)) = 0) else
                              mem(to_integer(unsigned(IDL2))) after 10 ns;
end compor;
```

Figura 3.16 *Especificación en VHDL de un banco de registros.*

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

La lectura es asíncrona y se especifica mediante sentencias concurrentes[17]. Para tener en cuenta que siempre se lee el valor cero del registro cero la asignación concurrente es condicional. 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.

17. Las sentencias de asignación condicional que modelan las lecturas se evalúan cuando las señales IDLx o mem cambian.

Nota: Para esta práctica ha sido suministrado un fichero que al desempaquetarlo crea una estructura de directorios, incluyendo alguno de ellos ficheros. El directorio raíz es LAB3, el cual incluye dos directorios: NUCLEO_camino y buffer_circular. Cada uno de ellos incluye un arbol de directorios. En el Apéndice 3.6 y el Apéndice 3.8 se muestra la organización en directorios.

En cada uno de los directorios NUCLEO_camino y buffer_circular se dispone de un directorio denominado "documentacio" donde se ubica la documentación generada con Doxygen. En el Apéndice 3.7 y el Apéndice 3.9 se indica la forma de acceder a la documentación.

Trabajo: En el directorio NUCLEO_camino/COMPONENTES/BR/CODIGO está incluida una especificación de un banco de registros (fichero BR.vhd).

Las declaraciones de tamaño y de tipos están especificadas en el fichero NUCLEO_camino/tipos_ctes_pkg/cte_tipos_nucleo_pkg.vhd. Los retardos están especificados en el fichero retardos_nucleo_pkg.vhd ubicado en el mismo directorio.

Analice la especificación y los ficheros asociados.

El proyecto ya ha sido creado en el directorio QUARTUS. En el directorio PRUEBAS está ubicado el fichero prueba_BR.vhd que ayuda a comprobar el funcionamiento del banco de registros. En este fichero se utilizan procedimientos, ubicados en otro fichero (procedimientos_pkg.vhd), para estimular las entradas del banco de registros y comprobar las salidas. La utilización de procedimientos se expone en el Apéndice 3.10 y la utilización de procedimientos para escribir en la ventana textual o ficheros se expone en el Apéndice 3.11. Analice los ficheros y efectúe una simulación.

En el Apéndice 3.1 se describe cómo efectuar una simulación ciclo a ciclo o paso a paso. En el Apéndice 3.2 se describe cómo sincronizar las líneas vertical en la ventana de tiempo de Modelsim con un flanco de la señal de reloj.

Modifique el programa de prueba para comprobar el funcionamiento del puerto de lectura IDL2. Posteriormente, modifique el programa de prueba para comprobar la lectura concurrente de los dos puertos de lectura. En las pruebas utilice el mismo y distintos identificadores de registros.

Camino de datos y control del mismo

En la Figura 3.17 se muestra el núcleo de un camino de datos y un módulo de control para generar los identificadores de registro (lectura y escritura).

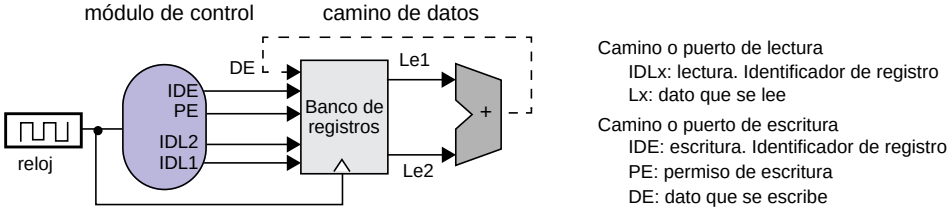


Figura 3.17 *Camino de datos y módulo control.*

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).

En la Figura 3.18 se muestran dos secuencias de identificadores de registros para realizar operaciones. En la secuencia de la izquierda, un registro destino (IDE) no se utiliza como registro fuente en una operación IDLx). En la secuencia de la derecha, el registro destino (IDE) se utiliza, en todas las operaciones, como uno de los registros fuente (IDL2) para la siguiente operación. En cada caso la secuencia de identificadores es generada por el módulo control.

		operación sin dependencias								
etiqueta	IDL1	0	1	2	3	4	5	6	7	8
	IDL2	9	10	11	12	13	14	15	16	17
	IDE	18	19	20	21	22	23	24	25	26

	operación con dependencias								
IDL1	0	1	2	3	4	5	6	7	8
IDL2	0	10	10	10	10	10	10	10	10
IDE	10	10	10	10	10	10	10	10	10

Figura 3.18 *Secuencia de identificadores de registro que genera el módulo control.*

Control del camino de datos

Podemos describir un sistema secuencial en términos de un camino de datos y el controlador, el cual es un autómata. El camino de datos está construido a partir de elementos combinacionales y secuenciales.

En al Figura 3.19 se muestra un modelo general de un sistema particionado siguiendo la idea descrita.

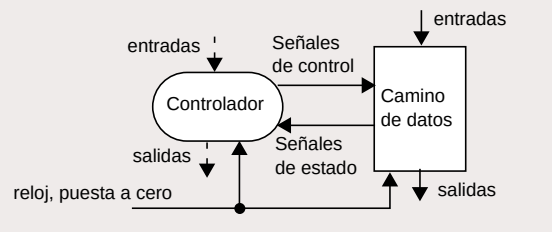


Figura 3.19 *Esquema general de un sistema con camino de datos y controlador específico.*

En general un camino de datos manipula y procesa datos. El controlador permite y determina el secuenciamiento de las operaciones en el camino de datos. Las señales de estado, provenientes del camino de datos, son utilizadas por el controlador como indicación de eventos que se producen en el camino de datos. Las señales de control, provenientes del controlador, se utilizan en el camino de datos para establecer las operaciones que quieren realizarse.

En el diseño que se está efectuando, el camino de datos es el banco de registros y el sumador. No se utilizan señales de estado y las señales de control son los identificadores de registro y la señal de permiso de escritura.

El módulo de control de la Figura 3.17 incluye tres contadores que generan una secuencia de identificadores de registro. Por ejemplo, una de las secuencias que se muestra en la Figura 3.18. Para generar estas secuencia se puede utilizar alguno de los contadores, con ligeras modificaciones, especificados en el Apéndice 3.3.

Ubicación de las fases de una operación aritmética en un ciclo

El núcleo del camino de datos de un 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 el cual 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 3.20 muestra las tres fases descritas previamente y su relación con un ciclo de la señal de reloj. La magnitud de los retardos es un ejemplo y no existe relación con el flanco descendente de la señal de 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.

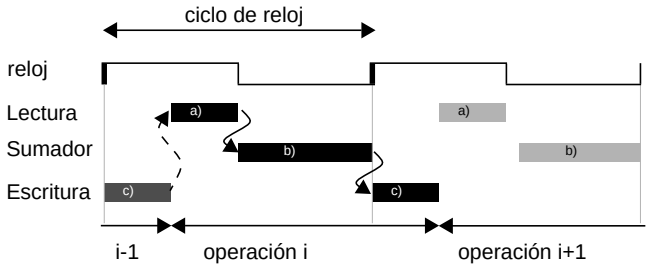


Figura 3.20 *Fases de una operación aritmética y su relación con el tiempo de ciclo. EL flanco descendente de la señal de reloj no está relacionado con ninguna actividad. Las flechas con trazo contínuo indican encadenamiento de retardos en una operación. Las flechas con trazo discontínuo indican dependencia entre operaciones (se lee un registro que se actualiza).*

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 (Figura 3.20). 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 3.21 se muestra un esquema simplificado del núcleo de un camino de datos (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.

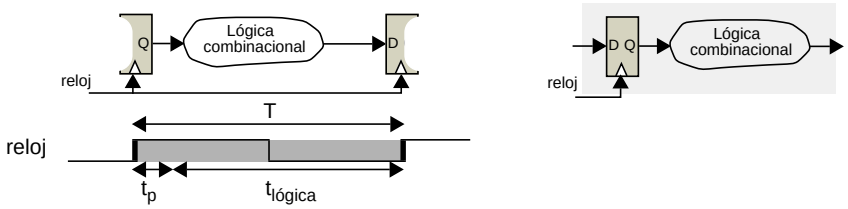


Figura 3.21 *Esquema simplificado del núcleo de un procesador.*

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.

$$T \geq t_p + t_{logica}$$

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

Evaluación del retardo del camino de datos

En la Figura 3.22 se muestran los componentes del camino de datos. En una entrada del sumador se distinguen de forma explícita los componentes, del banco de registros, involucrados en una acción de lectura y en la salida del sumador los componentes, del banco de registros, involucrados en una acción de escritura.

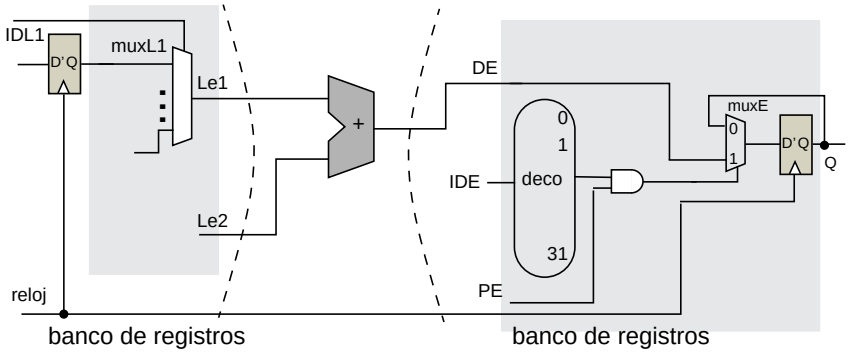


Figura 3.22 *Componentes del camino de datos.*

En la Figura 3.23 se indican los tiempos de retardo de los distintos elementos incluidos en el camino de datos y del módulo de control.

Componentes	Elementos	retardo (ns)	
Banco de registros	Decodificador (DECO)	t_{deco}	8
	Multiplexor (muxLx)[a]	t_{muxL}	10
	Registro	t_p	14
	Multiplexor (muxE)	t_{muxE}	0
	puerta AND	t_{AND}	0
Sumador		t_{sum}	16
Generador de identificadores de registro		t_{cont}	2

a. x = {1, 2}

Figura 3.23 *Retardos de los componentes del camino de datos y del módulo de control.*

Camino de datos simulado

En la Figura 3.24 se muestra el circuito que se utiliza en la simulación. En este circuito se distingue un banco de registros (BR), un sumador y un módulo de control. Adicionalmente, para inicializar el circuito y comprobar el resultado se dispone de las señales Pcero, Ini, PEini, DEini, IDEini, IDL1ini. El módulo control tiene como salida la señal selec, para seleccionar entre las entradas PEini, DEini, IDEini, IDL1ini y las salidas correspondientes generadas por él. El retardo de los multiplexores utilizados es cero.

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 Le1. El bus etiquetado como Le2 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.

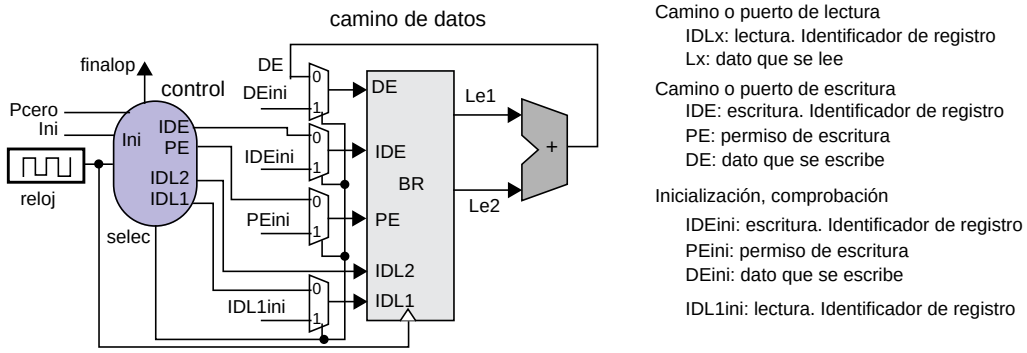


Figura 3.24 *Camino de datos utilizado en la simulación. Los identificadores de registros utilizan 5 bits y los datos se representan con 8 bis.*

El banco de registros dispone de 32 registros numerados del 0 al 31 y el número de bits que se almacena en cada uno de ellos es 8 bits. El registro cero está cableado a cero. Esto es, siempre se lee el valor cero.

Módulo de control

El módulo de control utiliza un autómata, o máquina de estados finitos, de dos estados, denominado principal y varios contadores (autómatas) subordinados al autómata principal[18]. Los autómatas subordinados son los que generan los identificadores de registro.

Los autómatas subordinados se activan cuando la señal Ini toma el valor uno. Cuando finaliza la secuencia programada se activa la señal finalop.

18. En el Apéndice 3.3 se describe el patrón de diseño de un autómata. También se muestran ejemplos sencillos.

El grafo de estados del autómata principal se muestra en la Figura 3.25. El diseño del autómata sigue el patrón descrito en el Apéndice 3.3. En particular, el diseño estructural.

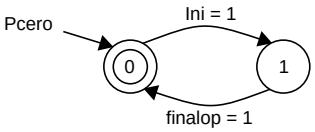


Figura 3.25 *Autómata principal del módulo control.*

Los autómatas subordinados (contadores) están especificados de forma similar: contador, IdenL1, IdenL2 y IdenE. En la Figura 3.26 se muestra el cuerpo de la arquitectura que describe al autómata.

```
begin
estadoreg: reg1 port map(reloj => reloj, e => prxestado, s => estado);   | Autómata principal
prxestado <= '0' when pcero ='1' or t_finalop = '1' else
             '1' when (ini = '1' and estado = '0') else
             '1' when estado = '1'
             else '0';
contador: registro port map(reloj => reloj, e => prxcnt, s => cnt);       | Autómatas secundarios
prxcnt <= -- contar el número de operaciones de la secuencia
IdenL1: registro port map(reloj => reloj, e => prxIDL1, s => IDL1);
prxIDL1 <= -- generación del identificador de registro
IdenL2: registro port map(reloj => reloj, e => prxIDL2, s => IDL2)
prxIDL2 <= -- generación del identificador de registro
IdenE: registro port map(reloj => reloj, e => prxIDE, s => IDE);
prxIDE <= -- generación del identificador de registro
t_finalop <= '0' when pcero = '1' or estado = '0' else
             '1' when cnt = "00000" and estado = '1'
             else '0';
finalop <= t_finalop;
selec <= not estado;
PE <= estado;
IDL1 <= t_IDL1;
IDL2 <= t_IDL2;
IDE <= t_IDE;
end;
```

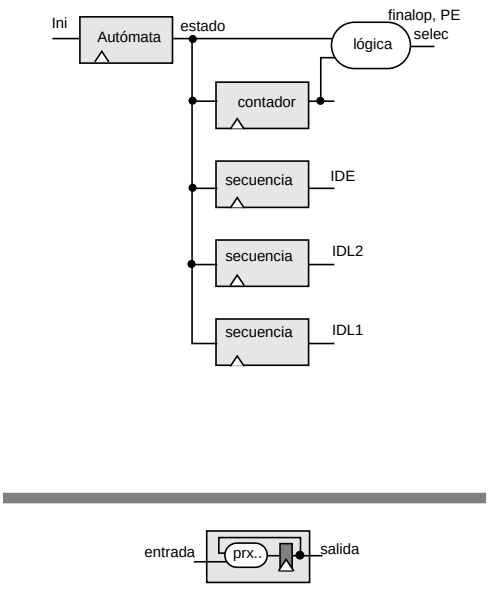


Figura 3.26 *Autómata de control. Automa principal y 4 autómatas subordinados. Registro y reg1 son registros de n bits y de 1 bit respectivamente.*

Trabajo: En el directorio NUCLEO_camino/ENSAMBLADO/CODIGO está ubicado el fichero que contiene la descripción estructural del camino de datos con un control. Analice la especificación y los ficheros asociados[19]. Para ello dispone, como ayuda, de la documentación generada utilizando "Doxygen".

El módulo control (NUCLEO_camino/COMPONENTES/control/control_indep/CODIGO/control.vhd) genera la secuencia de identificadores de registro mostrada en la parte izquierda de la Figura 3.18.

El proyecto ya ha sido creado en el directorio QUARTUS_control_indepe asociado al proyecto NUCLEO_camino cuando no hay dependencia en la secuencia de operaciones.

Análisis de los retardos en el diagrama temporal

En la Figura 3.27 se muestra un ciclo de simulación cuando se está efectuando la operación programada (parte izquierda de la Figura 3.18). En este ejemplo, cuando se ha activado la simulación, el contenido de cada registro es el valor del ordinal que identifica al registro.

En el caso concreto de un bus, el valor del grupo de señales que constituyen el bus se muestra como “unsigned”.

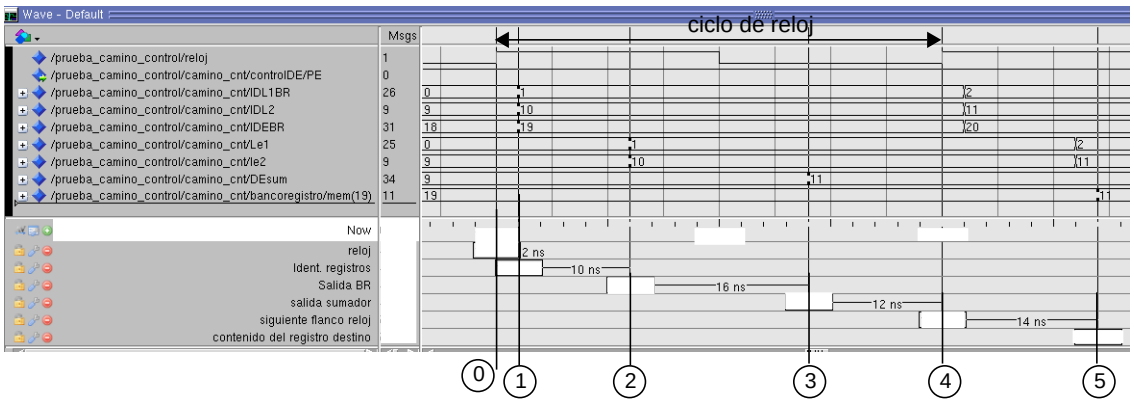


Figura 3.27 *Evolución de las señales en la ventana temporal de Modelsim. Un grupo se señales se muestra como “unsigned”.*

Los retardos de los componentes del camino de datos no se visualizan de forma explícita, hay que determinarlos a partir de cambios de valor en las señales[20].

19. En la sentencia de asignación de la señal de salida del sumador (señal suma en el fichero sumador.vhd) se ha especificado “transport” para que el simulador utilice este modo de simulación en esta sentencia. Por defecto, en una simulación funcional, Quartus activa el modo “inertial” en Modelsim. En el modo “transport” se observa, en la señal de salida, cualquier modificación que determine un cambio de valor en una entrada, después del retardo especificado. Utilizando el modelo inertial algunos cambios en las señales de entrada, menores que el retardo especificado en el componente, no se observan en la salida.

Seguidamente se muestra un análisis del ciclo de reloj identificado en la parte superior de la Figura 3.27. Las etiquetas de las señales se observan a la izquierda de la figura[21]. 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 3.27 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 ⓪ en la Figura 3.27. En los siguientes párrafos, el número que inicia el párrafo se corresponde con la marca numérica en la Figura 3.27. Los retardos de los elementos están especificados en el fichero NUCLEO_camino/tipos_ctes_pkg/cte_tipos_nucleo_pkg.vhd.

- 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 y escriben. Los identificadores de los registros que se leen son el 1 y el 10 (IDL1, IDL2). El valor del identificador del registro donde se escribe (IDE) es 19. El lapso de tiempo transcurrido entre el flanco ascendente de la señal de reloj y la observación de los identificadores de registro, indicado por los cursores, es 2 ns, el cual debe corresponderse con el valor establecido en el fichero control.vhd[22].
- 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 Le1 y Le2) el valor que se ha leído (1 y 10) de los registros 1 y 10 respectivamente. El lapso de tiempo (10 ns) entre el establecimiento de valores en los identificadores de registro (IDL1 e IDL2, ①) y la observación de los valores leídos, del banco de registros, debe corresponderse con t_{muxL}, el cual está especificado en el fichero BancoReg.vhd.
- ③ 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 DEsum) se observa el valor 11, que es la suma de los valores 1 (Bus Le1) y 10 (Bus Le2). El lapso de tiempo (16 ns) transcurrido desde que los valores de los registros alimentan al sumador (②) hasta que se obtiene el resultado se corresponde con t_{sum}, el cual está especificado en el fichero sumador.vhd. El valor 11 se conoce antes del flanco ascendente de la señal de reloj. Por tanto, en el registro 19 (IDE) se almacena el valor 11.
- ④ Flanco de la señal de reloj en el siguiente ciclo.

20. Efectuamos una simulación funcional con el mismo retardo para los componentes de un bus, el cual es salida de un elemento. Entonces, todos los cambios de señales se producen en el mismo instante de tiempo.
21. En este ejemplo solo se muestran las señales de interés para la explicación.
22. El retardo de los multiplexores utilizados para inicializar el banco de registros y comprobar el resultado es cero (Figura 3.24).

- ⑤ El valor 11 se puede observar en el siguiente ciclo analizando el contenido del registro 19 (mem(19)). El lapso de tiempo (14 ns) transcurrido desde el flanco de reloj se corresponde con el retardo para almacenar en un registro, el cual está especificado en el fichero BancoReg.vhd.

Trabajo: El fichero de prueba es PRUEBAS_control_indepe/prueba_camino_control.vhd. En el fichero de prueba, después de la puesta a cero, se inicializa el contenido de los registros del banco de registros. Posteriormente se indica al módulo control (Ini = 1) que efectúe la secuencia de operaciones que tiene programada de forma cableada. Posteriormente se comprueba el contenido de todos los registros, teniendo en cuenta la operación realizada. Para realizar esta última operación se utiliza una funcionalidad de VHDL-2008, denominada referencia jerárquica (Apéndice 3.5), que permite observar señales en diseños jerárquicos.

En el proyecto de Quartus ya está inicializada la funcionalidad "Nativelink simulation", para que se efectúe una simulación con Modelsim. Las señales que se visualizan en la ventana temporal están especificadas en el fichero wave.do, ubicado en el mismo directorio PRUEBAS_control_indepe y el fichero que contiene las órdenes para efectuar la simulación es formato_ventanas.do.

El programa de prueba imprime en la ventana textual de Modelsim un mensaje que indica el instante en el cual: a) se inicializan los registros del banco de registros, b) se inicia el cálculo programado y c) se inicia la comprobación. Además, para cada mensaje, Modelsim muestra el instante de tiempo mediante otro mensaje. Pulsando con el ratón en este último mensaje, el cursor seleccionado se ubica en el instante de tiempo correspondiente.

Por otro lado, en el directorio RESULTADOS_control_indepen se escribe un fichero que muestra el contenido de los registros al finalizar la simulación. Para ello se utilizan los procedimiento especificados en el fichero impri_BR_pkg.vhd ubicado en el directorio LAB3/UTILIDADES_pkg/impri_BR_pkg. La activación de almacenar los resultados dependen de un genérico de la entidad del programa de prueba.

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 puede ser distinto al descrito en el análisis previo.

Descripción de un funcionamiento incorrecto. En un funcionamiento correcto el valor de la suma (etiqueta DEsum) 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 3.28.

En la Figura 3.28 la descripción de los instantes de tiempo con las marcas ① y ② es igual a la efectuada en la Figura 3.27, que mostraba la evolución de las señales.

La marca de interés es la ④. Está después del flanco ascendente de la señal reloj (③) y el valor 11 es la suma del contenido de los registros 1 (IDL1) y 10 (IDL2), el cual es respectivamente 1 (Bus Le1) y 10 (Bus Le2).

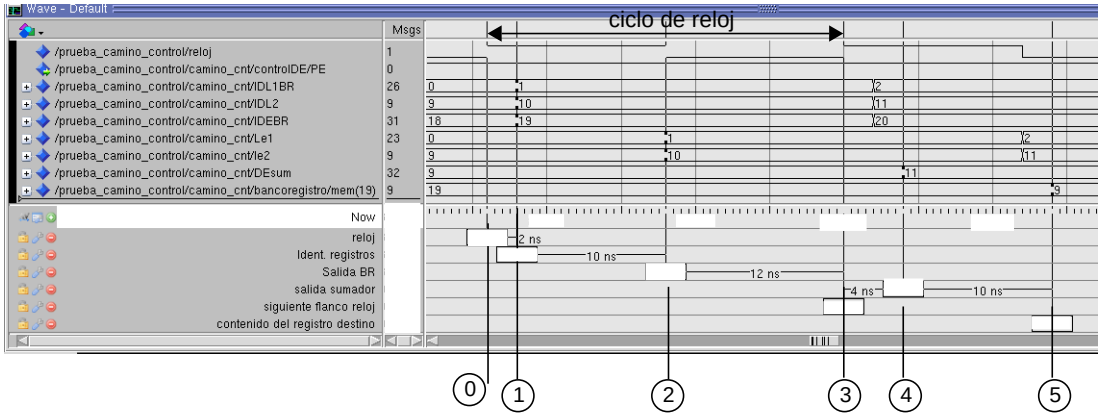


Figura 3.28 *Observación de un funcionamiento incorrecto. Un grupo se señales se muestra como "unsigned".*

Como el valor de la suma (Bus DEsum) en el flanco ascendente de la señal de reloj es 9, en el registro 18 se almacena un valor incorrecto. Esto puede observarse en el ciclo siguiente al que se está analizando (marca ⑤).

Trabajo: La señal de reloj está especificada en el programa de prueba prueba_camino_control.vhd. Para modificar la señal de reloj hay que editar el fichero. En el proceso, correspondiente a la señal de reloj, se indica la duración de cada semiperiodo de la señal de reloj. Se utiliza una señal cuadrada[23] y el semiperiodo se especifica como un parámetro en el programa de prueba ("generic").

Modifique el periodo de la señal de reloj para visualizar un funcionamiento incorrecto, semejante al mostrado en la Figura 3.28.

Una vez modificado el periodo puede activar otra vez la simulación desde Quartus.

Trabajo: Abra el proyecto QUARTUS_control_depen/camino_control.qpf. Este proyecto es idéntico al anterior, excepto en la secuencia de identificadores de registros, la cual debe programarse.

El fichero que genera la secuencia de identificadores de registro se encuentra en NUCLEO_camino/COMPONENTES/control/control_depen/CODIGO/control.vhd. Edite el fichero y diseñe los autómatas para que generen la secuencia de identificadores de

23. Esto es, el mismo lapso de tiempo en cada nivel de la señal.

registro mostrada en la parte derecha de la Figura 3.18. Para comprobar el diseño puede utilizar el programa de pruebas, el está ubicado en el directorio "PRUEBAS_control_depen" asociado al proyecto.

Determine el periodo mínimo para un funcionamiento correcto. Para ello analice el diagrama temporal. Corrobore el resultado dibujando un diagrama temporal de retardos utilizando los retardos[24] especificados en la Figura 3.23.

Buffer circular

Para la comunicación entre un productor y un consumidor puede utilizarse una cola FIFO (First-In First-Out). Esta cola permite absorver ráfagas del productor antes de que las procese el consumidor, incrementando la productividad del sistema[25]. La cola se utiliza para almacenar los datos que no pueden ser consumidos durante un bloqueo del consumidor, si no está lleno. Alternativamente, el consumidor puede extraer datos del buffer mientras el productor está produciendo un dato, si no está vacío

Una forma usual de construir una FIFO es utilizar un conjunto de posiciones de almacenamiento con dos puertos y reutilizar una posición cuando es consumida. Este tipo de buffer FIFO se denomina buffer circular (Figura 3.29).

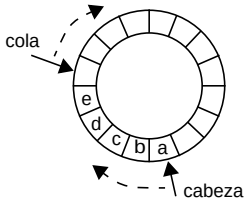


Figura 3.29 *Esquema de un buffer circular.*

Uno de los puertos del buffer circular es utilizado por el productor (escritura) y el otro puerto es utilizado por el consumidor (lectura). Cada uno de los puertos tiene asociado un contador (o puntero) para conocer la posición de almacenamiento que se escribe (productor) o que se lee (consumidor).

La estructura de almacenamiento es un banco de registros o memoria RAM. Como punteros, una posibilidad es utilizar contadores binarios. Cuando un contador llega al máximo valor, el siguiente valor es cero.

24. El retardo de un componente se empieza a marcar a partir del instante de tiempo en el cual todas las señales de entrada, que determinan el valor de su salida, son válidas.
25. En media la rapidez de producción es menor igual que la rapidez de consumo.

El contador asociado al puerto de escritura, al cual denominamos cola, indica la posición de almacenamiento donde el productor almacena la información en una acción de inyección (escritura). El contador asociado al puerto de lectura, al cual denominamos cabeza, indica la posición de almacenamiento de donde el consumidor extrae información (lectura).

El productor almacena información en entradas del buffer, utilizando el puntero cola, el cual indica la primera entrada libre. De forma similar, el consumidor extrae información de entradas del buffer, utilizando el puntero cabeza, el cual indica la primera entrada con información[26].

Cuando la cola recorre las posiciones de almacenamiento y encuentra la cabeza, el buffer está lleno y no se pueden almacenar datos. Al contrario, cuando la cabeza recorre las posiciones de almacenamiento y encuentra la cola, el buffer está vacío y no se puede extraer información. En la Figura 3.30 se muestra un buffer circular y el control asociado al mismo.

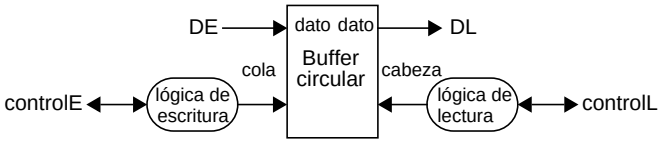


Figura 3.30 *Control en un buffer circular.*

Los elementos lógica de escritura y lógica de lectura de la Figura 3.30 incluyen circuitos lógicos para acceder al buffer circular y controlar las entradas ocupadas del mismo.

Control en un buffer circular

Para leer y escribir en un buffer circular se utilizan los punteros cabeza y cola. La cabeza indica la posición de almacenamiento que se lee y la cola indica la posición de almacenamiento que se escribe. El estado inicial es que los dos punteros (cabeza y cola) indiquen la misma posición de almacenamiento.

Partiendo de estas hipótesis, hay que determinar las condición de buffer lleno y buffer vacío.

La condición de que cola sea igual a cabeza no es una condición suficiente para distinguir si el buffer está lleno o vacío. Ahora bien, para una gestión simple del buffer es suficiente considerar que el buffer está lleno cuando hay N-1 entradas ocupadas, siendo N el número de entradas del buffer. Esto es, el buffer puede contener como máximo N-1 elementos sin extraer antes de que el productor se bloquee.

26. En el caso general es suficiente que productor y consumidor recorran la secuencia de posiciones de almacenamiento en el mismo orden.

Protocolo de comunicación

En el módulo buffer el control de escritura es la interface con el módulo productor y el control de lectura es la interface con el módulo consumidor (Figura 3.31).

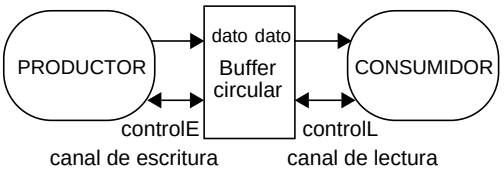


Figura 3.31 *Interface entre módulo productor, módulo buffer circular y módulo consumidor. No se muestra la lógica de escritura y de lectura.*

Por canal o enlace de comunicación entre dos módulos entendemos las conexiones entre los dos módulos. El módulo buffer tiene dos canales. El canal de escritura y el canal de lectura. Seguidamente se describe el protocolo en un canal de comunicación.

Protocolo en un canal de comunicación: listo/válido

En la Figura 3.32 se muestra un canal de comunicación entre un emisor y un receptor. En el canal se distingue una línea de comunicación de datos entre un emisor y un receptor. Adicionalmente existen dos líneas de control en sentidos opuestos: a) válido y b) listo. El protocolo se utiliza para controlar el flujo de información entre el emisor y el receptor.

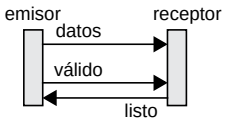


Figura 3.32 *Canal de comunicación.*

La línea de control denominada listo, del receptor al emisor, indica al emisor que el receptor acepta información. La línea de control denominada válido, del emisor al receptor, indica al receptor que hay información válida en el canal de comunicación.

En el canal se distinguen tres estados, que se muestran en la Figura 3.33.

Estado	válido	listo
Transferencia	1	1
Sin transferencia	0	{0, 1}
Espera	1	0

Figura 3.33 *Estados en un canal de comunicación.*

El emisor tiene un comportamiento persistente. Esto es, mantiene el dato mientras no reconoce que ha sido capturado por el receptor.

La transferencia o captura de un dato se produce en el flanco ascendente de la señal de reloj. Todos los componentes utilizan el mismo reloj.

El emisor activa la señal válido independientemente de la señal listo. Una vez ha activado la señal válido, no debe modificarla (persistente) hasta que se ha producido la transferencia (listo = 1 y flanco de reloj, Figura 3.34).

El receptor captura la información cuando la señal válido está activa y también está activada la señal listo. La captura se produce en el flanco de la señal de reloj.

Este protocolo permite que el emisor y el receptor bloqueen su “función” durante un tiempo arbitrario utilizando las señales válido y listo.

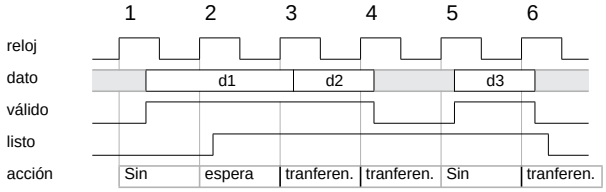


Figura 3.34 *Ejemplo del protocolo de comunicación.*

En el flanco del ciclo 2 de la Figura 3.34 la señal listo está desactivada y no se produce transferencia de información. El comportamiento del emisor es persistente. Por tanto, en el flanco del ciclo 3 aún mantiene activada la señal válido. En este flanco, la señal listo está activada. En consecuencia, se produce la transferencia de información (d1).

En el flanco del ciclo 4 se produce la misma situación que en el ciclo 3 y se efectúa otra transferencia de información (d2). En el flanco del ciclo 5, el emisor no tiene activada la señal válido. En consecuencia, no hay transferencia de información. En el flanco del ciclo 6, las señales válido y listo están activadas y se produce transferencia de información (d3).

Protocolo en los canales de comunicación de un buffer circular

El buffer circular se arropa con circuitería para implementar el protocolo de canal denominado listo/válido. En la Figura 3.35 se muestra la circuitería añadida, la cual utiliza las señales de control lleno y vacío del buffer circular.

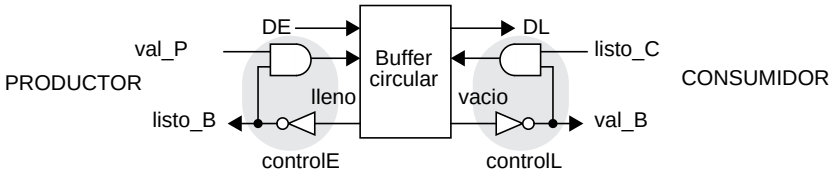


Figura 3.35 *Buffer circular con canales de comunicación listo/válido. No se muestra la lógica de escritura y de lectura.*

Con la información que se transmite por el canal, que conecta el productor y el buffer circular (receptor), se transmite información de control que coordina el flujo de información. La información de control son dos señales en sentidos opuestos denominadas: a) val_P y b) listo_B. En el caso del canal que conecta el consumidor con el buffer circular, el cual es el emisor, la información de control son las señales: a) listo_C y b) val_B.

Cuando el productor (emisor) quiere emitir información a través del canal, activa la señal val_P. El receptor (buffer circular) captura la información del canal cuando observa la señal val_P activa y la señal listo_B está activada. El productor (emisor) desactiva la señal val_P cuando no emite un dato. En el caso de que el receptor (buffer circular) no esté preparado para aceptar un dato desactiva la señal listo_B.

En el canal de comunicación entre el buffer circular (emisor) y el consumidor (receptor) el protocolo es parejo.

En resumen, solo existe comunicación cuando el receptor (buffer, consumidor) está dispuesto a aceptar información (listo_X = 1) y el emisor (productor, buffer) emite información (val_Y = 1). Cuando un receptor no está preparado para aceptar información (listo_X = 0), la información permanece en el canal (es persistente). El dato no debe ser modificado por el emisor hasta que el receptor esté preparado y consuma la información. Un emisor conoce que la información ha sido consumida cuando se cumple listo_X = val_Y = 1. En este caso puede inyectar más información en ciclos posteriores.

Buffer circular: camino de datos y control

En la Figura 3.36 se muestra un esquema lógico de un buffer circular, el cual consta de: a) un banco de registros (BR), b) dos módulos, denominados cola y cabeza y c) las interfaces correspondientes.

El módulo cabeza genera la dirección de lectura en el BR e indica si el buffer está vacío. El módulo cola genera la dirección de escritura en el BR, la señal de permiso de escritura (PE) e indica si el buffer está lleno.

Para generar la señal lleno, el módulo cola necesita la dirección de lectura generada en el módulo cabeza y viceversa, el módulo cabeza, para generar la señal vacío, necesita la dirección de escritura generada en el módulo cola.

Los módulos cabeza y cola modifican, si es el caso, sus salidas de forma síncrona con el flanco de la misma señal de reloj[27], siendo la excepción la señal PE[28]. El almacenamiento de información, por parte del productor, en el BR también se efectúa en el flanco de la señal de reloj.

El productor, después de activar la señal val_P, debe esperar a que la señal listo_B indique que el buffer acepta información. En el primer flanco de la señal de reloj se almacena la información en el buffer. De forma similar, el consumidor, cuando está listo (listo_C = 1), debe esperar a que la señal val_B indique que hay elementos en el buffer. En el primer flanco de la señal de reloj podrá extraer información.

El buffer almacena información, en el flanco de reloj, cuando la señal val_P está activada y el buffer no está lleno (listo_B = 1). De forma similar, el buffer considera que se ha leído información, en el flanco de reloj, si la señal listo_C está activada y el buffer no está vacío (val_B = 1).

Mientras el buffer no esté lleno ni vacío se puede almacenar y extraer información de forma concurrente.

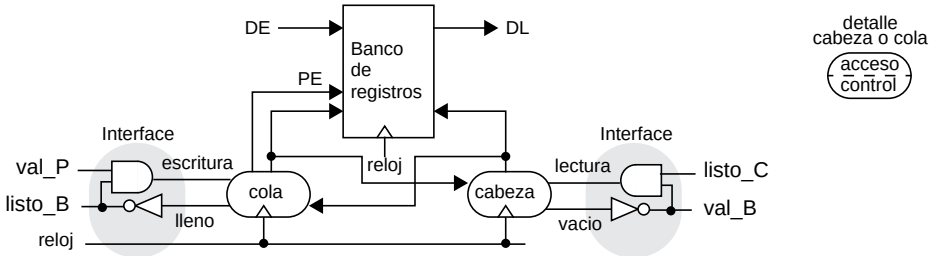


Figura 3.36 *Componentes en un buffer circular.*

En el esquema de la Figura 3.36 no se muestra la señal de puesta a cero (pcero) o inicialización. El diseño dispone de esta señal para inicializar los punteros cola y cabeza. Las señales lleno y vacío se inicializan al inicializar los punteros cola y cabeza. En consecuencia, también se inicializan las señales listo_B y val_B.

27. En este contexto entendemos que es el flanco ascendente de la señal de reloj.
28. Esta señal depende de la señal asíncrona “escritura”, inducida por el productor y debe estar activada antes del flanco de la señal de reloj para que se efectúe la escritura en el BR.

Cuando se activa la señal pcero ambos contadores se inicializan a cero[29]. Esto es, los punteros cola y cabeza indican el registro cero del BR[30]. La señal listo_B está activada e indica que el buffer acepta almacenar información. La señal val_B está desactivada e indica que no se puede extraer información del buffer.

Implementación. Al implementar el buffer circular de la Figura 3.36 la funcionalidad de los componentes cola y cabeza se divide en dos: a) acceso al BR y b) control. En estas condiciones la funcionalidad de acceso de los componentes cola y cabeza se ubica conjuntamente en un elemento denominado acceso. De forma similar, la funcionalidad de control de los dos componentes se ubica conjuntamente en el elemento denominado control (Figura 3.37).

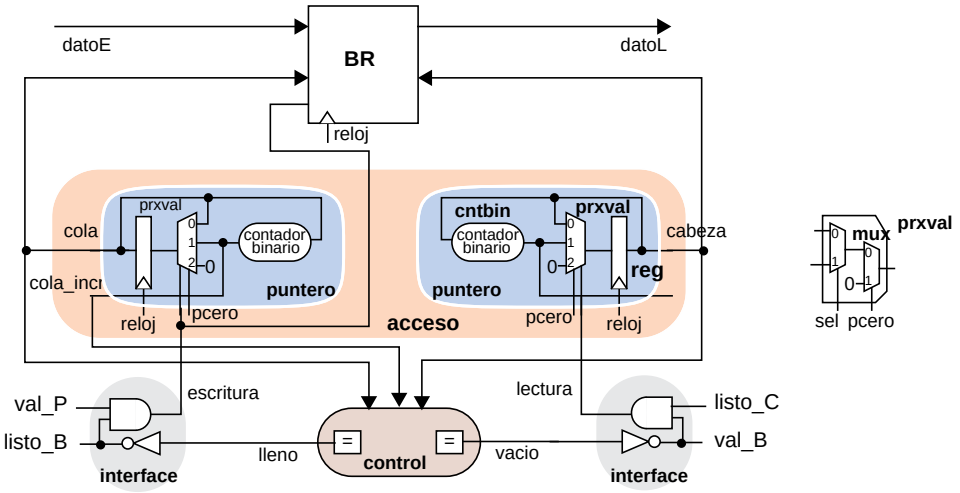


Figura 3.37 *Implementación del buffer circular. Elementos constituyentes.*

Seguidamente se describen algunos de los elementos utilizados en el diseño.

Punteros cola y cabeza. Son contadores binarios.

Condiciones de lleno y vacío. En la Figura 3.38 se muestran las relaciones entre los punteros cola y cabeza suponiendo que son contadores binarios.

29. La cola indica la entrada del buffer que se rellena, cuando el buffer no está lleno. La cabeza indica la entrada que se lee, cuando el buffer no está vacío.
30. Es suficiente que sea el mismo valor. Posteriormente los dos punteros deben de recorrer la misma secuencia de valores.

Condición	señal	relación entre punteros		
buffer vacío	vacio	cabeza = cola		
buffer lleno	lleno	cola = (cabeza - 1) mod N = (cabeza + N - 1) mod N	o	(cola + 1) mod N = cabeza

Figura 3.38 *Relaciones entre los punteros para determinar las condiciones de lleno y vacío.*

Elemento puntero. El elemento puntero es utilizado para construir la funcionalidad de acceso de los componentes cabeza y cola. Este elemento almacena y actualiza la dirección de la posición de memoria a la que se accede. Adicionalmente suministra información para determinar las condiciones lleno y vacío. En la Figura 3.39 se muestra un esquema del circuito.

El circuito puntero consta de un contador binario, dos multiplexores y un registro. El contador binario, una vez inicializado, genera de forma repetida la secuencia [0, . . ., N-1]. Mediante la señal pcero se establece el valor cero en el registro, de forma síncrona. El registro se actualiza con el valor del contador, de forma síncrona, cuando la señal sel está activada.

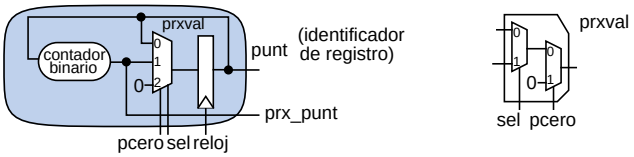


Figura 3.39 *Puntero. Estructura genérica para generar un identificador de registro.*

Elemento acceso. Este elemento contiene la funcionalidad de acceso al BR de los componentes cabeza y cola. En la Figura 3.40 se muestra la estructura. Se utilizan dos instancias del elemento puntero, una instancia para la cabeza y otra para la cola.

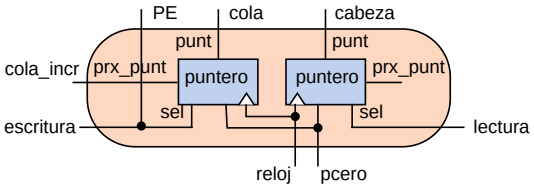


Figura 3.40 *Elemento acceso.*

Nota: En el Apéndice 3.8 se muestra la organización en directorios de los proyectos correspondientes al buffer circular.

En el Apéndice 3.9 se indica la forma de acceder a la documentación generada con Doxygen.

Nota: Las declaraciones de tamaño y de tipos de datos están especificadas en el fichero buffer_circular/tipos_ctes_pkg/cte_tipos_buffer_pkg.vhd. Los retardos están especificados en el fichero retardos_buffer_pkg.vhd ubicado en el mismo directorio.

Trabajo: En la tabla se muestra la ubicación ficheros del proyecto.

Directorios
buffer_circular
COMPONENTES
control_interace
COMPONENTES
acceso
ENSAMBLADO
CODIGO
acceso.vhd
puntero
ENSAMBLADO
CODIGO
puntero.vhd
componentes_acceso_pkg
componentes_acceso_pkg.vhd
COMPONENTES

El fichero acceso.vhd contiene la interface del elemento acceso. Este elemento debe incluir los punteros cola y cabeza.

Complete el elemento acceso, utilizando un diseño estructural, para su inclusión en el buffer circular. El elemento que debe utilizarse es el componente puntero cuya descripción está en el fichero "componentes_acceso_pkg.vhd". En el directorio COMPONENTES de la última fila de la tabla se ubican los subdirectorios con los elementos que se utilizan para el diseño del componente puntero.

En el directorio QUARTUS está creado el fichero acceso.qpf asociado al proyecto. Una vez completado el diseño, compruebe la elaboración RTL que efectúa Quartus.

En el directorio PRUEBAS, asociado al proyecto, hay un programa de pruebas.

Elemento control. Este elemento determina las condiciones de vacío y lleno del buffer. En la Figura 3.41 se muestra la estructura del elemento control sin las conexiones entre los elementos constituyentes y las entradas. El elemento básico es un comparador.

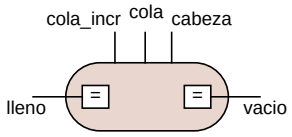


Figura 3.41 *Elemento control.*

Trabajo: El fichero control.vhd contiene la interface del elemento control, el cual se utiliza para determinar las señales de control vacío y lleno.

Directorios

buffer_circular

COMPONENTES

control_interace

COMPONENTES

control

CODIGO

control.vhd

Complete el elemento control para su inclusión en el buffer circular. Cualquier acción de comparación debe especificarse con una sentencia de asignación de señal condicional. Compile el elemento control con Quartus, analice la elaboración RTL y compruebe el funcionamiento con Modelsim. Para ello dispone de los ficheros oportunos en los directorios QUARTUS y PRUEBAS asociados al proyecto del componente control.

Elemento interface. Este elemento contiene la lógica que implementa el protocolo de canal listo/válido (Figura 3.42).

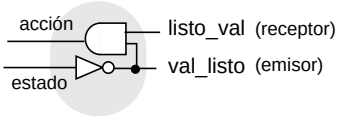


Figura 3.42 *Elemento interface.*

Módulo control interface. Para construir este módulo se ensamblan todos los elementos descritos previamente (Figura 3.43). .

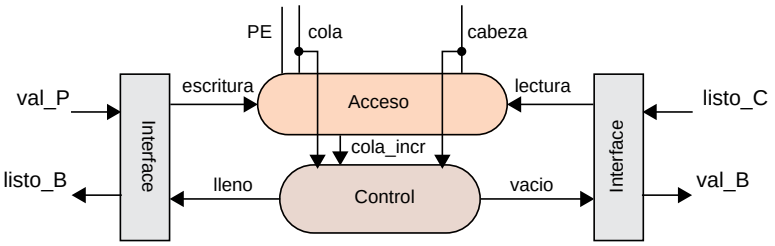


Figura 3.43 *Módulo control interface.*

Elemento banco de registros (BR). Este elemento de almacenamiento dispone de un puerto de escritura síncrono con la señal de reloj y de un puerto de lectura asíncrono (Figura 3.44).

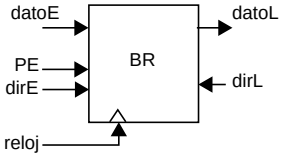


Figura 3.44 *Banco de registros. Un puerto de escritura y un puerto de lectura.*

Módulo buffer circular. Para construir este módulo se ensamblan todos los elementos descritos previamente (Figura 3.45).

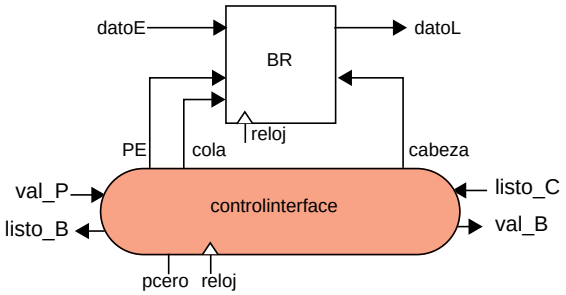


Figura 3.45 *Módulo buffer y elementos constituyentes.*

Trabajo: En el directorio buffer_circular/ENSAMBLADO se encuentran los directorios asociados al proyecto buffer circular.

Compile el proyecto y analice la elaboración RTL.

Diagramas temporales

En la Figura 3.46 se muestra el interconexionado del buffer circular con un productor y un consumidor (no se muestra la señal de reloj). El productor espera a que la señal listo_B esté activada, después de haber activado la señal val_P, para producir otro dato. Si el productor tarda más de 1 ciclo en producir el dato, éste desactiva la señal val_P.

El consumidor, una vez está listo (señal listo_C activada), espera a que esté activada la señal val_B. El conjunto de las dos señales se utiliza como permiso de escritura (pe) del registro que alimenta al consumidor. Si el consumidor tarda más de 1 ciclo en procesar el dato desactiva la señal listo_C.

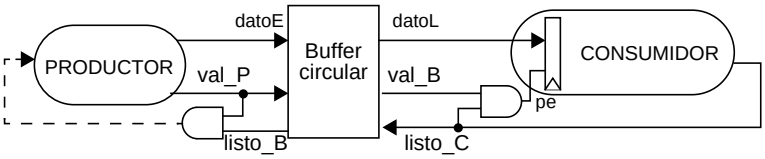


Figura 3.46 *Esquema del buffer circular y la comunicación con un productor y un consumidor.*

En la Figura 3.47 se muestra un funcionamiento en régimen permanente del buffer. En cada ciclo se inyecta un dato y también se extrae un dato. El acrónimo CONS indica el procesado del dato en el consumidor (retardo). Todas las señales de control val_X y listo_Y están activadas en los 3 ciclos.

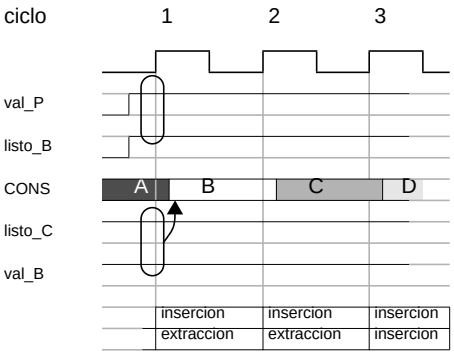


Figura 3.47 *Lecturas y escrituras concurrentes en el buffer.*

En la Figura 3.48 se muestran los casos de buffer lleno y escritura pendiente (productor) y buffer vacío y lectura pendiente (consumidor). En la parte izquierda de la Figura 3.48 se muestra que el buffer se llena en el ciclo 1 y que en el ciclo 2, el productor también quiere inyectar un dato en el buffer. Concurrentemente, en el ciclo 2, el consumidor (CONS) efectúa una extracción del buffer. Notemos que la actualización del buffer por parte del productor no se efectúa hasta el ciclo 3, cuando el buffer no está lleno.

En la parte derecha de la Figura 3.48 se muestra que el buffer está vacío en los ciclos 1 y 2. Durante estos ciclos el consumidor está esperando consumir un dato. En el ciclo 2 el productor añade un dato al buffer. El consumidor (CONS) consume este dato en el ciclo 3.

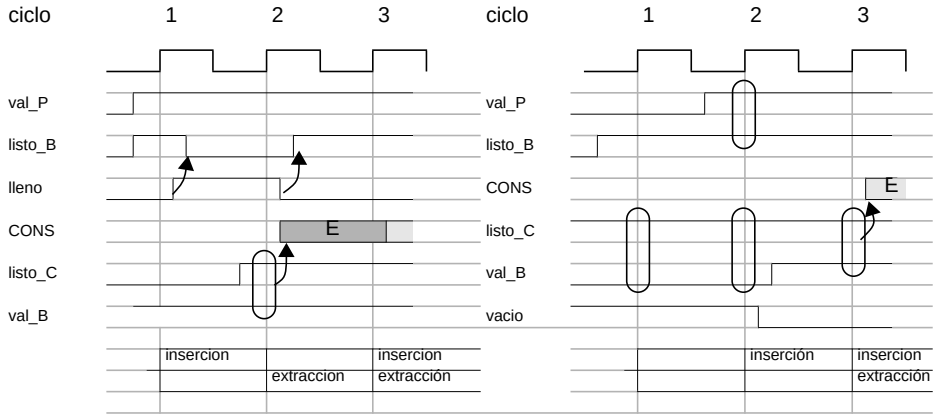


Figura 3.48 *Buffer lleno con escritura pendiente (izquierda) y vacío con lectura pendiente (derecha). Realización concurrente de una lectura o una escritura.*

Trabajo: Utilice el programa de prueba, ubicado en el directorio PRUEBAS asociado al proyecto de ensamblador del buffer_circular, para comprobar y observar el funcionamiento del buffer circular. Este programa incluye dos procesos que representan al productor y al consumidor (Figura 3.46).

El tiempo de generación de un dato por el productor puede modificarse mediante la variable tiempoproducir (valor mayor igual que 1) que se pasa como parámetro al procedimiento producir. El tiempo que el consumidor tarda en consumir un dato puede modificarse mediante la variable tiempoconsumir (valor mayor igual que 1) que se pasa como parámetro al procedimiento consumir.

En particular, identifique en el diagrama temporal los casos descritos en la Figura 3.47 y en la Figura 3.48. Para ello, modifique si es necesario el programa de prueba.

Utilización de todas las entradas del buffer

La igualdad de los punteros cola y cabeza no es suficiente para discernir si el buffer está lleno o vacio. Para ello es necesario utilizar información adicional cuando se quieren utilizar todas las entradas del buffer circular.

Autómata. Una alternativa es utilizar un indicador de si el buffer se está llenando o vaciando. Esto es, si la última acción en el buffer ha sido almacenar información o extraer información. En el caso de efectuarse las dos acciones concurrentemente, el efecto es nulo. En la Figura 3.49 se muestra un autómata de dos estados que indica la última acción realizada.

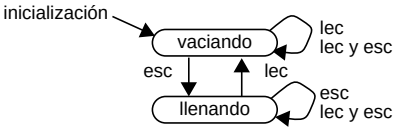


Figura 3.49 *Autómata para determinar la última acción realizada en el buffer circular: lectura o escritura.*

Cuando cola y cabeza indican la misma entrada del buffer y el buffer se está llenando, el buffer está lleno. En caso contrario, si el buffer se está vaciando, el buffer está vacio.

El autómata se incluye en el elemento control (Figura 3.50). En estas condiciones, el elemento control no requiere de la entrada cola_incr disponible en el diseño anterior (Figura 3.41), pero se le añaden las señales lec y esc.

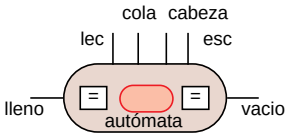


Figura 3.50 *Elemento control con autómata.*

Trabajo: Copie el fichero acceso.vhd, que ha creado previamente, en el directorio COMPONENTES/control_interface/COMPONENTES/acceso/ENSAMBLADO_automata/CODIGO. Edite el fichero y elimine el puerto de salida denominado colaincr. La nueva interface del componente está especificada en el fichero componentes_control_interface_automata_pkg.vhd almacenado en el directorio COMPONENTES/control_interface/componentes_control_interface_automata_pkg.

El fichero COMPONENTES/control_interface/COMPONENTES/acceso/control_automata/CODIGO/control.vhd contiene la interface del elemento control, el cual se utiliza para determinar las señales de control vacío y lleno. Los componentes que son visibles están especificados en el fichero ubicado en el directorio ../componentes_control-automata_pkg.

En el fichero control.vhd debe especificar el autómata que ha sido descrito de forma textual para determinar las señales de control vacío y lleno, junto con las comparaciones necesarias.

Para mantener el estado utilice el componente "reg_1", Especifique la lógica que determina el próximo estado mediante sentencias de asignación de señal condicional. De idéntica forma especifique la lógica de salida. Utilice la señal pcero para inicializar el estado.

Cualquier acción de comparación debe especificarse con una sentencia de asignación de señal condicional.

El retardo de evaluación de las salidas (lleno y vacío), a partir de las otras señales, es retcontrol.

En el directorio QUARTUS, asociado a este proyecto (control_automata), se incluye el fichero donde está creado el proyecto. Compruebe la elaboración RTL que efectúa Quartus.

Utilice el programa de prueba, ubicado en el directorio PRUEBAS, asociado al proyecto, para comprobar el funcionamiento del elemento.

Módulo buffer. En la Figura 3.51 se muestra el ensamblado de los componentes.

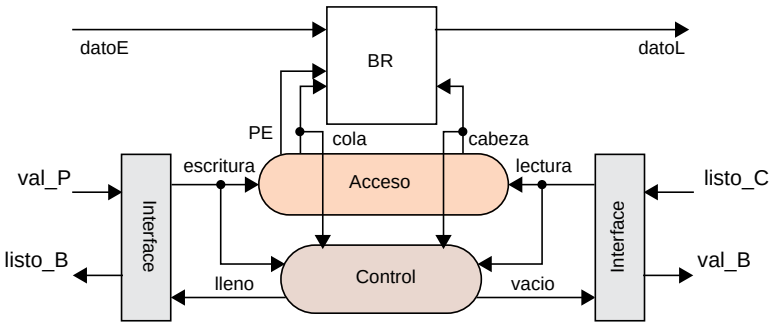


Figura 3.51 *Módulo buffer donde se pueden ocupar todas las entradas del buffer. Utilización de un autómata.*

Trabajo: En el directorio buffer_circular/ENSAMBLADO_automata se encuentran los directorios asociados al proyecto buffer circular ocupando todas las entradas del buffer.

Compile el proyecto y analice la elaboración RTL.

Utilice el programa de prueba, ubicado en el directorio PRUEBAS asociado al proyecto, para comprobar el funcionamiento del buffer circular, cuando se pueden ocupar todas las entradas del buffer.

Apéndice 3.1: Simulación paso a paso

El semiperiodo de la señal de reloj se especifica como un genérico (semiperiodo) en el programa de prueba. En la declaración de la entidad se utilizan otros genéricos (Figura 3.52). Mediante ellos, si existen, se controla la simulación.

Genérico	Descripción	Valor
semiperiodo	semiperiodo de la señal de reloj cuadrada	25 ns
pasoapaso	permite una simulación ciclo a ciclo	false
imprimir_BR	imprime el contenido del banco de registros al finaliza la simulación	true

Figura 3.52 *Parámetros de la simulación.*

Para modificar los valores por defecto utilizados en los genéricos hay que editar el fichero “prueba_*” ubicado en el directorio PRUEBAS asociado al proyecto.

Seguidamente se describe cómo efectuar una simulación paso a paso de forma manual o autómatica.

Manual

La simulación que se efectúa por defecto es “run-all”. Ahora bien, para efectuar una simulación paso a paso modifique la última sentencia del fichero formato_ventanas.do. Especifique solo run. Posteriormente modifique el parámetro de simulación “Default Run” (Simulate-> Runtime Options y en la pestaña “Defaults” el campo “Default Run”). Por ejemplo, puede utilizarse como valor un semiperiodo de la señal de reloj. Cada vez que de la orden “run” (icono en la parte superior de la ventana de Modelsim) se simula un paso.

Simulación con ModelSim. Para activar la simulación de Modelsim desde Quartus dé la orden que se muestra en la Figura 3.53 (RTL Simulation).

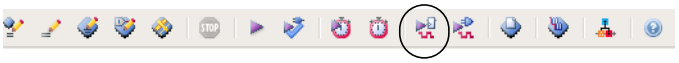


Figura 3.53 *Activación de simulador Modelsim desde la paleta de iconos de Quartus.*

Simulación paso a paso (ciclo a ciclo). Una vez activada la simulación utilice la orden “Run” para avanzar un ciclo de simulación (Figura 3.54).

Figura 3.54 *Paleta de iconos en ModelSim. Orden Run.*

Automática

En este caso se utiliza una funcionalidad disponible en VHDL_2008. Por tanto, hay que indicar que el programa de prueba se compile con la versión VHDL_2008.

Mediante el package entorno (environment) se puede controlar la simulación. Hay que declarar su utilización.

declaración

```
use std.env.all;
```

En concreto utilizaremos el procedimiento que permite parar la simulación y continuar posteriormente.

procedimiento

```
stop (status: integer)
```

Como parámetro del procedimiento utilizaremos el valor cero (0).

Para activar este modo de simulación ha sido declarado un genérico en la declaración de la entidad del programa de prueba (pasoapaso).

Genérico	Descripción (activo: valor true)	Valor
pasoapaso	permite una simulación ciclo a ciclo	true

Este procedimiento se utiliza en el proceso reloj (Figura 3.55).

Proceso reloj

```
rlj: process
begin
. . .
      reloj <= '0';
      wait for semiperiodo;
. . .
      reloj <= '1';
      wait for semiperiodo;
      if pasoapaso then
            stop(0);
      end if;
. . .
end process ;
```

Figura 3.55 *Utilización del procedimiento stop para controlar la simulación.*

Una vez se para la simulación hay que utilizar la orden "Run-All" para avanzar otro ciclo de simulación (Figura 3.56).

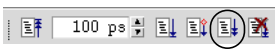


Figura 3.56 *Paleta de iconos en ModelSim. Orden Run-All.*

Apéndice 3.2: Marcas verticales en la ventana temporal de Modelsim

Para establecer líneas verticales en cada inicio de ciclo de la señal de reloj deben efectuarse los siguientes pasos. Seleccione la ventana temporal. Posteriormente, dé la orden “Wave -> Wave Preferences ...”. En la ventana emergentes (Figura 3.57), pestaña Grid&Time”, estableza como valor en “Grid Offset” el valor, en nanosegundos, del semiperiodo de la señal de reloj y en “Auto Period” el valor del periodo de la señal de reloj (2 x semiperiodo, Figura 3.57).

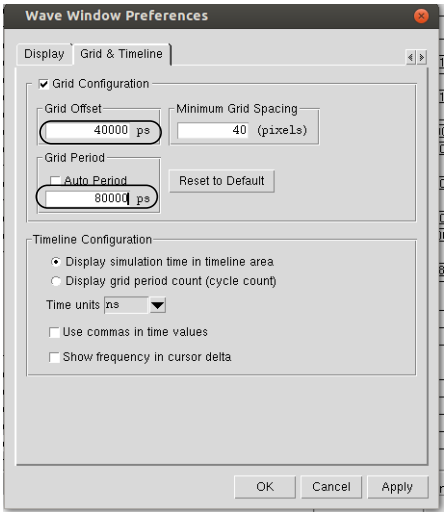


Figura 3.57 *Pasos para establecer marcas en la ventana temporal. En el ejemplo, el periodo de la señal de reloj es 80 ns.*

Una vez establecidas las marcas en la ventana temporal se pueden almacenar para simulaciones posteriores. Para ello, seleccione la ventana temporal y posteriormente dé la orden “File -> Save Format”. En la ventana emergente pulse en el botón “Browse...” y muévase por la jerarquía de directorios. El fichero wave.do debe almacenarse en el directorio PRUEBAS. En caso de utilizarse otro nombre debe modificarse el fichero formato_ventanas.do, que está ubicado en el mismo directorio.

Apéndice 3.3: 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 3.58. 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 combinacional que determina el próximo estado. La lógica salida es otro circuito combinacional.

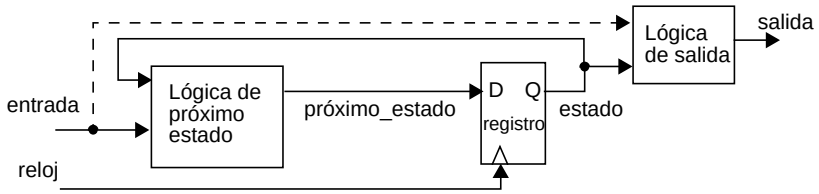


Figura 3.58 *Modelo esquemático de un circuito secuencial síncrono.*

En la Figura 3.58 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 combinacionales 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 3.59 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.

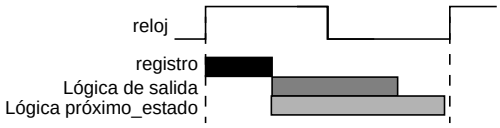


Figura 3.59 *Diagrama temporal, en un periodo de la señal de reloj, de un circuito síncrono. Relaciones de dependencia entre los retardos.*

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 (autómata subordinado).

Diseños simples

Registro con puesta a cero asíncrona. En la tabla de la Figura 3.60 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

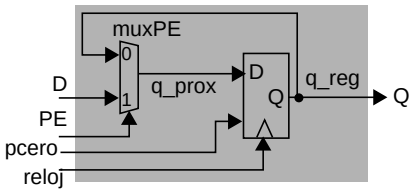


Figura 3.60 *Registro con permiso de escritura síncrono.*

En la parte derecha de la Figura 3.60 se muestra el diagrama de un registro con permiso de escritura. Observemos que podemos asimilar elementos de este diagrama con elementos mostrados en la Figura 3.58. 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 3.61 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 condicional[31], las cuales se describen en un Apéndice 2.9 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;
```

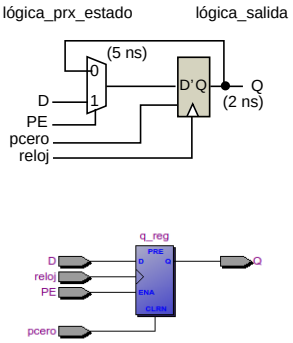


Figura 3.61 *Descripción VHDL de un registro con puesta a cero asíncrona, distinguiendo los elementos combinacionales y secuenciales.*

En la descripción VHDL de la Figura 3.61 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 3.61 sigue el modelo de la Figura 3.58. 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 componentes de forma aislada. En la Figura 3.62 se muestra una descripción VHDL del mismo tipo de registro donde se entremezclan los distintos elementos, lo cual dificulta la comprobación y verificación.

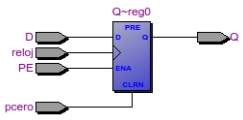
31. Esta sentencia de asignación de señal de forma condicional puede sustituirse por un proceso que describa un circuito combinacional.

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 compor 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 compor;
```



Elaboración efectuada por QUARTUS

Figura 3.62 *Descripción VHDL de un registro con permiso de escritura utilizando un único proceso.*

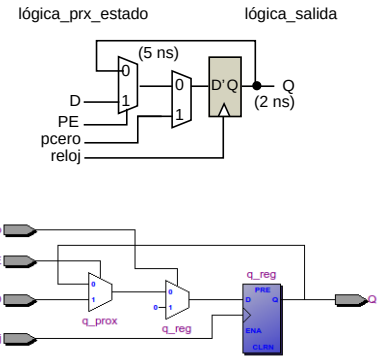
Puesta a cero síncrona. La puesta a cero puede efectuarse de forma síncrona. En la Figura 3.63 se muestra la descripción VHDL.

Registro con puesta a cero síncrona

```
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 (reloj'event and reloj = '1') then
          if (pcero ='1') then
             q_reg <= '0' after 2 ns;
          else
             q_reg <= q_prox after 2 ns;
          end if ;
        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;
```



Elaboración efectuada por QUARTUS

Figura 3.63 *Descripción VHDL de un registro con puesta a cero síncrona, distinguiendo los elementos combinacionales y secuenciales.*

Descripción estructural de un registro con permiso de escritura y puesta a cero síncrona. En la Figura 3.64 se muestra la especificación del circuito de la Figura 3.63 utilizando un registro especificado en VHDL, sin permiso de escritura ni puesta a cero. En la descripción queda explícito el elemento que almacena el estado.

Descripción estructural. Registro con puesta a cero síncrona

```
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 estruc of regen is
component registro is
  generic (retardo: time := 2 ns);
  port ( reloj, D: in std_logic;
      Q: out std_logic);
end component;
signal q_reg: std_logic;
signal q_prox: std_logic;
begin
--Estado
reg:     registro generic map (retardo => 2 ns)
         port map (reloj => reloj, D => q_prox, Q => q_reg);
-- Lógica de próximo estado
         q_prox <= '0' when pcero = '1' else D when PE = '1' else q_reg;
-- Lógica de salida
         Q <= q_reg;
end estruc;
```

registro

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

entity registro is
  generic (retardo: time := 2 ns);
  port ( reloj, D: in std_logic;
      Q: out std_logic);

end;
architecture compor of registro is
begin
process (reloj)
begin
  if (reloj'event and reloj = '1') then
     Q <= D after retardo;
  end if ;
end process;
end compor;
```

Figura 3.64 *Descripción estructural en VHDL de un registro con puesta a cero síncrona.*

En la Figura 3.64 se muestra la elaboración efectuada por QUARTUS.

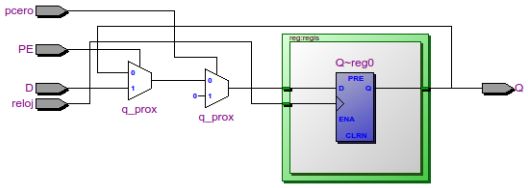


Figura 3.65 *Elaboración efectuada por Quartus a partir de la especificación de la Figura 3.64.*

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 3.66 se muestra un esquema de un contador binario. Comparando esta figura con la Figura 3.58 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.

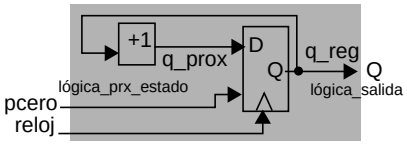


Figura 3.66 *Contador binario.*

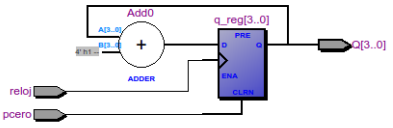
En la Figura 3.67 se muestra una descripción VHDL del contador binario donde se distinguen tres partes. El proceso denominado reg modela el elemento de memorización denominado registro. Después del proceso, la primera sentencia de asignación de señal modela el incrementador unidad. La última sentencia 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.numeric_std.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 <= std_logic_vector(unsigned(q_reg) +1) after 8 ns ;
-- Lógica de salida
         q <= q_reg;
end trespart;
```



Elaboración efectuada por QUARTUS

Figura 3.67 *Descripción VHDL de un contador binario.*

Contador módulo. En la Figura 3.68 se muestra el esquema de circuito de un contador módulo 7. La lógica próximo estado es un incrementador, un multiplexor y un comparador.

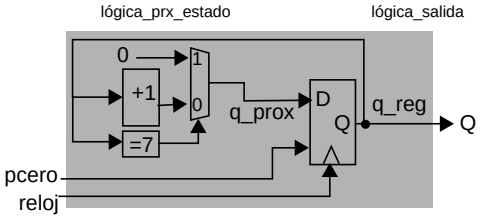


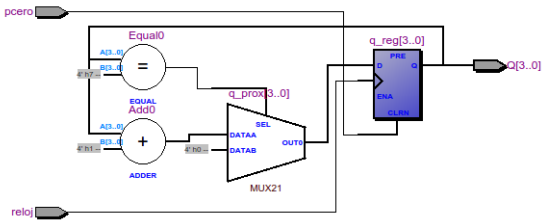
Figura 3.68 *Contador módulo.*

En la Figura 3.69 se muestra la descripción VHDL siguiendo el modelo de la Figura 3.58 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

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;
entity cntrmod is
port (   reloj, pcero : in    std_logic;
         Q: out               std_logic_vector (3 downto 0) );
end cntrmod;
architecture trespart of cntrmod 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 to_integer( unsigned(q_reg)) = modu else std_logic_vector(unsigned(q_reg) +1) after 10 ns ;
-- Lógica de salida
         q <= q_reg;
end trespart;
```



Elaboración efectuada por QUARTUS

Figura 3.69 *Descripción VHDL de un contador módulo.*

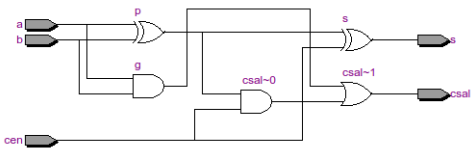
Apéndice 3.4: Síntesis de descripciones VHDL

Una simulación (Modelsim) de las especificaciones de la Figura 3.70 muestra resultados distintos. En la simulación del código de la izquierda se constata que el valor de las señales p y g, establecidas en una ejecución del proceso, no son observables hasta la siguiente ejecución. El comportamiento es similar a la existencia de un dispositivo de almacenamiento entre las señales p y g y las señales s y csal[32].

Circuito combinacional

```
library ieee;
use ieee.std_logic_1164.all;
entity s1bitS is
    port (a, b, cen: in std_logic;
          s, csal: out std_logic);
end s1bitS;
architecture compor of s1bitS is
signal p, g: std_logic;
begin
proc_s1bitS: process (a, b, cen)
begin
    p <= a xor b after 1 ns;
    g <= a and b after 1 ns;
    s <= p xor cen after 1 ns ;
    csal <= g or (p and cen) after 1 ns;
end process proc_s1bitS;
end compor;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity s1bitV is
    port (a, b, cen: in std_logic;
          s, csal: out std_logic);
end s1bitV;
architecture compor of s1bitV is
begin
proc_s1bitV: process (a, b, cen)
variable p, g: std_logic;
begin
    p := a xor b ;
    g := a and b s;
    s <= p xor cen;
    csal <= g or (p and cen) after 1 ns;
end process proc_s1bitV;
end compor;
```



Síntesis efectuada por QUARTUS (también la elaboración RTL que muestra)

Figura 3.70 *Especificaciones de un sumador de 1 bit.*

Por otro lado, las dos especificaciones de la Figura 3.70 son sintetizadas (Quartus) como un circuito combinacional. Esto es, el código VHDL generado por la herramienta de síntesis[33] se comporta como la especificación de la derecha de la Figura 3.70. Cuando se compila (elabora) el código de la izquierda, la herramienta Quartus indica que faltan las señales p y g en la lista de activación del proceso[34].

En los ejemplos de la Figura 3.71 existe una sentencia condicional y en una de las ramas no se efectúa ninguna asignación. Por otro lado, en la lista de activación sólo está especificada la señal utilizada en la sentencia condicional. La herramienta de síntesis infiere posiciones de almacenamiento. Para que se active el proceso debe producirse un cambio de nivel en la señal PE. Una vez se ha producido el cambio de nivel no se vuelve a activar el proceso hasta el siguiente cambio de nivel[35].

32. Recordemos que es el comportamiento especificado en el lenguaje VHDL. El valor establecido en una señal no es observable en la misma ejecución del proceso.
33. En este curso este código no se genera y por tanto, no se utiliza en la simulación con Modelsim.
34. Esta es una de las razones por las que utilizamos Quartus y posteriormente Modelsim. Quartus nos ayuda a comprobar si el código se corresponde con la especificación que queremos efectuar. Al activar la simulación con Modelsim se utiliza el código VHDL especificado por el usuario.

```
library ieee;
use ieee.std_logic_1164.all;

entity s1bitS is
    port (a, b, cen, PE: in std_logic;
          s, csal: out std_logic);
end s1bitS;

architecture compor of s1bitS is
signal p, g: std_logic;

begin
proc_s1bitS: process (PE)
begin
if (PE = '1') then
    p <= a xor b after 1 ns;
    g <= a and b after 1 ns;
    s <= p xor cen after 1 ns ;
    csal <= g or (p and cen) after 1 ns;
end process proc_s1bitS;
end compor;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity s1bitV is
    port (a, b, cen, PE: in std_logic;
          s, csal: out std_logic);
end s1bitV;

architecture compor of s1bitV is
begin

proc_s1bitV: process (PE)
variable p, g: std_logic;
begin
if (PE = '1') then
    p := a xor b ;
    g := a and b;
    s <= p xor cen after 1 ns;
    csal <= g or (p and cen) after 1 ns;
end process proc_s1bitV;
end compor;
```

Figura 3.71 *Sumador de 1 bit con registros.*

Las dos especificaciones de la Figura 3.71 son sintetizadas de forma diferente, lo cual es lo que se espera. La herramienta de síntesis traduce la especificación de la izquierda de tal forma que hay registros entre las señales p y g y las señales s y csal[36]. También hay registros después de la señales s y csal. En la síntesis de la especificación de la derecha sólo hay registro después de las señales s y csal[37].

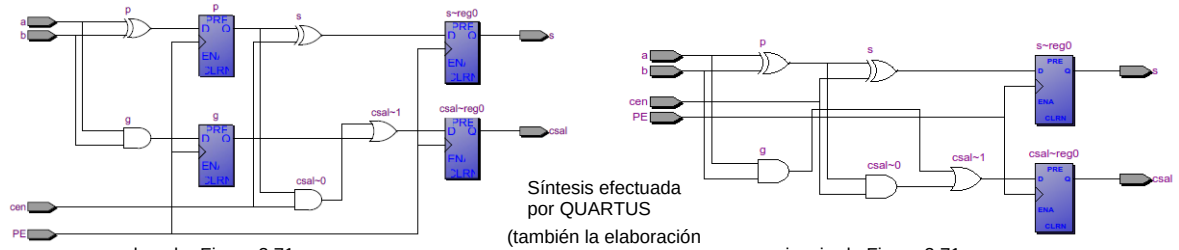


Figura 3.72 *Circuitos secuenciales sintetizado (elaborados) por QUARTUS partiendo de las especificaciones de la Figura 3.71.*

35. Al especificar PE como un objeto std_logic (9 posibles valores, 1, 0, x, H, L, . . .), en una simulación del código de usuario puede activarse el proceso en un cambio de indefinido al valor uno. Por ello, es recomendable utilizar la función “rising_edge” incluida en la package “std_logic_1164” de la librería ieee. También existe la función “rising_edge”. Estas funciones comprueban los metavalores definidos en la “std_logic” y además, se expresa de forma clara lo que se pretende. Otra posibilidad para la simulación de un código de usuario es añadir la condición reloj’last_value = ‘0’ en la sentencia condicional.
36. Se están utilizando sentencias de asignación de señal concurrentes.
37. Se utilizan variables en la evaluación de p y q.

Apéndice 3.5: Referencia jerárquica (VHDL-2008)

En la fase de comprobación o verificación interesa la observación o monitorización de señales u otros objetos utilizados en un diseño jerárquico desde el programa de prueba. La revisión de 2008 de VHDL pemite acceder a estos objetos. Esta característica se denomina nombres externos (“external names”). Un nombre externo especifica un camino jerárquico en un diseño jerárquico para acceder a un objeto.

Un nombre externo se escribe como[38]

Declaración

```
<< tipo_objeto camino_desde_exterior: subtipo >>
```

donde tipo_objeto indica una de las siguientes clases de objeto: constant, signal o variable. El subtipo especifica una visión del objeto.

El camino_desde_exterior está compuesto por: a) un delimitador “<<“, b) el tipo de objeto, c) los nombres (etiquetas) de los componentes instanciados. Cada nombre está seguido por un “.”. Al final se especifica el nombre del objeto y d) finalmente se especifica el delimitador “>>”.

En el contexto de un programa de prueba, el dispositivo está instanciado en el programa de prueba. El nombre_desde_exterior es un camino relativo. Se empieza especificando el nombre de la instancia[39].

38. Para más detalles consulte un manual de VHDL-2008.
39. Cuando se analiza el programa de prueba no se comprueba la existencia del objeto ni el tipo del mismo. Una vez ha sido elaborado el diseño jerárquico, el objeto debe existir y ser del tipo apropiado para corresponderse con el subtipo especificado en el nombre externo.

Apéndice 3.6: Organización de los directorios: NUCLEO de un camino de datos

En este apéndice muestra parte del árbol de directorios del núcleo de un camino de datos. En este árbol hay incluidos cinco proyectos en los subdirectorios indicados en cursiva. Los directorios en negrita son específicos al proyecto NUCLEO_camino en el que hay dependencias en la secuencia de operaciones. El directorio control_depen se corresponde con el proyecto donde en la secuencia de operaciones hay dependencias.

Arbol de directorios

- NUCLEO_camino
  - COMPONENTES
    - *BR*
    - componentes_nucleo_pkg
    - control
      - COMPONENTES
        - registros
      - componentes_control_pkg
      - ***control_depen***
      - *control_indep*
    - multiplexores
    - Sumador
  - ENSAMBLADO
    - ***QUARTUS_control_depen***
    - *QUARTUS_control_indepe*
  - tipos_ctes_pkg

Apéndice 3.7: Documentación: NUCLEO_camino

La documentación ha sido generada utilizando la herramienta Doxygen. El fichero que hay que abrir con un navegador es “index.html” ubicado en el directorio LAB3/NUCLEO_camino/documentacio/hmtl.

Las pestañas que se muestran son autoexplicativas.

Dado un módulo se muestra un grafo de dependencias jerárquicas con otros módulos. Pulsando en un nodo del grafo se observan el módulo o módulos que agrupa.

También se puede acceder al código VHDL que describe a un módulo.

Apéndice 3.8: Organización de los directorios: buffer circular

En este apéndice muestra parte del árbol de directorios del buffer circular. En este árbol hay incluidos dos proyectos del buffer circular. Los directorios en negrita son específicos al proyecto buffer circular en el que es factible llenar todas las entradas del buffer. Los directorios sin el sufijo "_automata" se corresponden con el proyecto buffer circular donde una entrada no se puede ocupar.

Arbol de directorios

buffer_circular
- COMPONENTES
  - BR
  - control_interface
    - COMPONENTES
      - acceso
        - componentes_acceso_pkg
        - ENSAMBLADO
        - puntero
          - COMPONENTES
            - cnt_bin
            - prxval
            - registro
            - registro_automata
          - componentes_puntero_pkg
          - ENSAMBLADO
        - **componentes_control_automata_pkg**
        - control
        - **control_automata**
        - interface
      - **componentes_control_interface_automata_pkg**
      - componentes_control_interface_pkg
      - ENSAMBLADO
      - **ENSAMBLADO_automata**
    - componentes_buffer_circular_pkg
  - ENSAMBLADO
  - **ENSAMBLADO_automata**
  - tipos_ctes_pkg

Apéndice 3.9: Documentación: buffer circular

La documentación ha sido generada utilizando la herramienta Doxygen. El fichero que hay que abrir con un navegador es “index.html” ubicado en el directorio LAB3/buffer_circular/documentacio/hmtl.

Las pestañas que se muestran son autoexplicativas.

Dado un módulo se muestra un grafo de dependencias jerárquicas con otros módulos. Pulsando en un nodo del grafo se observan el módulo o módulos que agrupa.

También se puede acceder al código VHDL que describe a un módulo, exceptuando los que se deben diseñar o completar.

Apéndice 3.10: Funciones y procedimientos en VHDL

En este apéndice se describe lo mínimo imprescindible para utilizar funciones y procedimientos.

Funciones

Cuando se llama a una función devuelve un resultado. En consecuencia, puede considerarse como un mecanismo para definir un operador.

Supondremos que una función se especifica en la parte declarativa del cuerpo de una arquitectura[40]. La sintaxis es la siguiente

Sintaxis	Ejemplo
function name_function [(formal_parameter_list)] **return** type **is** {declarative part} **begin** {sequential statements } **return** [expression] **end** [name_function];	**function** nand_f (**signal** a, b : std_logic) **return** std_logic **is** **begin** **return** (a **nand** b); **end** nand_f;

En primer lugar se describe la interface y el tipo de valor que se devuelve. Posteriormente se indica la lista de parámetros formales, los cuales se separan con ";". Todos los parámetros son de lectura (modo "in").

Sintaxis. Parámetros formales
[constant or **singal]** name_parameter [**in**] subtype]

En la parte declarativa se pueden declarar variables, las cuales son inicializadas en cada llamada a la función. En el cuerpo de la función se utilizan sentencias secuenciales, con la excepción de la sentencia wait.

Una función devuelve un valor del tipo especificado. Una función debe tener al menos una sentencia return.

Al llamar a una función, los parámetros actuales se pueden asociar con los parámetros formales utilizando asociación por posición o por nombre (semejante a la sentencia port map en la instanciación de un componente).

Un ejemplo de llamada a una función es el siguiente

llamada a una función	Comentario
c <= nand_f (a, b);	c ha sido declarada como std_logic

40. También se puede especificar en un "package" (Apéndice 2.6).

Procedimientos

Un procedimiento encapsula una secuencia de sentencias secuenciales.

Supondremos que un procedimiento se expecifica en la parte declarativa del cuerpo de una arquitectura[41]. La sintaxis es la siguiente

Sintaxis	Ejemplo

procedure name_procedure [(formal_parameter_list)] **is**
{declarative part}
begin
{sequential statements }
end [name_procedure];
	procedure nand_p (**signal** a, b : **in** std_logic; **signal** c **out** std_logic) **is**
	begin
	c <= a **nand** b;
	end nand_p;

En primer lugar se describe la interface y el tipo de valor que se devuelve. Posteriormente se indica la lista de parámetros formales, los cuales se separan con ";". La única forma de devolver un valor, calculado en un procedimiento, es la utilización de un parámetro. Para ello se utiliza el modo "out" o "inout" al declarar un parámetro. Un parámetro de modo "inout" puede leerse dentro del procedimiento.

Sintaxis. Parámetros formales

[constant or **singal** or **variable]** name_parameter [**in** or **out** or **inout**] subtype]

En la parte declarativa se pueden declarar variables, las cuales son inicializadas en cada llamada al procedimiento. Esto es, no son persistente entre llamadas. En el cuerpo de la función se utilizan sentencias secuenciales.

Un ejemplo de llamada a un procedimiento es el siguiente

llamada a un procedimiento

```
nand_p (a, b, c);
```

41. También se puede especificar en un "package" (Apéndice 2.6).

Apéndice 3.11: Acceso a ficheros en disco

Durante la comprobación de un circuito puede ser de utilidad utilizar ficheros de texto, ya sea para leer información o almacenarla.

Un fichero sólo puede contener un tipo de objetos. Nosotro utilizaremos el tipo string[42].

Ejemplos	Sintaxis
type text **is file of** string;	**type** data_type: **is file of** type_name;

La declaración de un fichero se efectúa de la siguiente forma[43]. Un fichero es un objeto de tipo “file”.

Ejemplos (fichero de texto)	Sintaxis
file S: text **open** write_mode **is** “resultados”;	**file** identifier: data_type [[**open** mode] **is** file_name]; mode = read_mode | write_mode | append_mode

La declaración de un fichero “file” puede incluirse en cualquier parte declarativa en la cual se crean los objetos: cuerpo de la arquitectura, procesos, “packages” o subprogramas. En “mode” se especifica el tipo de operación para la cual se abre el fichero físico asociado con el objeto “file”. “File_name” es una expresión del tipo predefinido “string”. Identifica un fichero en el sistema de ficheros, el cual está asociado con el objeto “file”.

Para entradas y salidas textuales se utiliza el “package” textio” de la libreria “std” (standar). Este “package hay que hacerlo visible.

Declaración (visibilidad)	Comentario
library std; **use** std:textio.all;	El “package” textio soporta los tipos: bit, bit_vector, boolean, character, string, integer, real, time

En este “package” están declarados los procedimientos que se utilizan para leer y escribir.

Las operaciones de lectura y escritura están orientadas a líneas de texto. Para leer o escribir una línea de texto hay que utilizar un objeto tipo “line” declarado como una variable.

Ejemplos
variable linea_S: line;

Escritura

Para escribir en una línea se parte de una línea vacía y se agrega información textual a la misma. Una vez ha sido construida una línea se escribe con el procedimiento “writeline” y la línea queda vacía. Esto es, una escritura se efectúa en dos pasos.

42. Este tipo está declarado en el “package” que se indica posterioremente (“textio”).
43. Las salidas estandar en VHDL se denominan “input” y “output”. Son dos variables de tipo “file” declaradas en el package “textio”.

La escritura de una linea en un fichero tiene el siguiente formato.

Ejemplos	Comentarios
writeline (S, linea_S);	S: identificador de fichero linea_S: variable de tipo “line”

Para agregar información a una línea se dispone de funciones en el “package” “textio”. Los dos primeros parámetros son la línea y el objeto.

Ejemplos	Sintaxis	Comentarios
variable var: integer; . . . write (linea_S, var , right, 8);	write (linea, objeto , [alineación], [anchura]); En el caso de un objeto de tipo real, se puede especificar, como último parámetro el número de dígitos después del punto. En el caso de un objeto de tipo time, se puede especificar como último parámetro las unidades de tiempo.	linea: variable de tipo “line”

Los argumentos “alineación” y “anchura” son opcionales. El argumento “alineación ” se utiliza para indicar si el texto se justifica a la derecha (“right”, es el defecto) o a la izquierda (“left”) del espacio utilizado para ubicar de forma textual el parámetro “valor”. El parámetro “ancho” (el defecto es cero y el texto ocupa el espacio necesario) se utiliza para indicar un espacio fijo en el cual se ubica el parámetro “valor” de forma textual. Las posiciones no utilizadas se rellenan con espacios. Su utilidad es para encolumnar datos de longitud variable como el tipo “integer”.

La función “write” se utiliza para escribir tipos de objetos distintos. Por ello, cuando se construye manualmente una cadena de valores para imprimir hay que indicar el tipo de los elementos que la componen.

Ejemplo

```
variable var: integer;
variable tiempo: time;
. . .
write (linea_S, var, rigth, 8);
write (linea_S, string’(“ , ”)); -- es util para importar el fichero a otro programa
write (linea_S, tiempo, rigth, 12, ns);
writeline (S, linea_S);
```

Ejemplo

```
variable var: integer;
variable tiempo: time;
. . .
write (linea_S, integer’image(var) & string’(“ , ”) & time’image(tiempo));
writeline (S, linea_S);
```

Para los tipos de objetos std_logic y std_logic_vector hay que utilizar el “package”.

Declaración (visibilidad)

```
use IEEE.std_logic_textio.all; -- está incluido en la librería IEEE
```

Este package dispone de procedimientos que permiten imprimir, además de en binario, en hexadecimal u octal. Los procedimientos correspondientes son:

Procedimientos para imprimir en hexadecimal u octal

```
hwrite (linea, valor , [alineación], [anchura]);
owrite (linea, valor , [alineación], [anchura]);
```

El siguiente código muestra un ejemplo donde se escribe una traza en un fichero.

Ejemplo

```
write (linea_S, string’(“ 0x”));
hwrite(linea_S, valor);
write (linea_S, string’(“ 0x”));
hwrite(linea_S, resultado);
write (linea_S, string’(“   ”));
write (linea_S, now); -- tiempo de simulación
writeline (S, linea_S);
```

Salidas

```
0x01   0x0111   5 ns
0x02   0x2341   12 ns
0x21   0x1357   20 ns
```

Lectura

En una operación de lectura se lee una línea completa. Posteriormente hay que obtener los elementos incluidos en la línea. La lectura también requiere dos pasos como la escritura.

El fichero se abre en modo lectura.

Ejemplos

```
file E: text open read_mode is “entradas”;
```

Se declara una variable de tipo “line”.

Ejemplos

```
variable linea_E: line;
```

La lectura de una linea de un fichero tiene el siguiente formato.

Ejemplos	Comentarios
readline (S, linea_E);	E: identificador de fichero linea_E: variable de tipo “line”

El acceso a los campos individuales de una línea se efectúa mediante el procedimiento “read” del “package” “textio”.

Ejemplos

```
variable var: bit_vector;
variable good: boolean;
. . .
read (linea_S, var, good);
if not good then
   assert (false) report (“ lectura erronea”) severity failure;
end if;
```

Sintaxis

```
read (linea, objeto , good);
```

La variable booleana good se utiliza para comprobar si el campo leído se corresponde con el tipo del objeto.
El “package” textio soporta los tipos: bit, bit_vector, boolean, character, string, integer, real, time.

linea: variable de tipo “line”

La utilización repetida del procedimiento “read” permite leer todos los campos de una línea. En el caso de objetos de tipo “bit_vector” e “integer” los distintos elementos deben estar separados por un espacio. Si la línea sólo contienen objetos de tipo “character” y “string” no se distinguen campos en la línea. Por ejemplo, si en un fichero hay lineas de comentario intercaladas y empiezan por “#” se pueden detectar de las siguiente forma.

Ejemplos

```
readline (S, linea_E);
if linea_E(linea_E’low) /= ‘#’ then
      read (linea_E, . . . );
. . .
```

Para leer la siguiente línea hay que llamar al procedimiento “readline”. Cualquier campo no leido en la línea previa queda descartado[44]. El procedimiento “readline” puede ser ejecutado hasta que se detecta la condición de “end-of-file” utilizando “endfile(E)”.

Ejemplos

```
while not endfile (E) loop
. . .
end loop;
```

44. Esta característica puede utilizarse para añadir comentarios en los ficheros de entrada.

Para el tipo de objetos std_logic y std_logic_vector hay que utilizar el “package”.

Declaración (visibilidad)

```
use IEEE.std_logic_textio.all; -- está incluido en la librería IEEE
```

Este package permite leer, además de en binario, en hexadecimal u octal. Los procedimientos correspondientes son:

Procedimiento para lee en hexadecimal u octal

```
hread (linea, elemento]);
oread (linea, elemento);
```

El siguiente código muestra un ejemplo de lectura de datos de un fichero.

Ejemplo

```
readline (E, linea_E);
hread (linea_E, var1);
hread(linea_E, var2);
```

Entradas

```
01    0111    primero
05    2341    quinto
09    1357    noveno
```

Abrir y cerrar ficheros

La especificación que se ha mostrado previamente, al declarar un fichero abría directamente el fichero. Ahora bien, esta acción se puede posponer. Por otro lado, después de abrir un fichero se puede cerrar explícitamente.

Para abrir un fichero se utiliza el procedimiento “file_open” y para cerrarlo “file_close”..

Ejemplos

```
file fichero : text;
. . .
variable status : file_open_status;
. . .
file_open(status, fichero, "Nombre", read_mode);
assert status=open_ok
      report "No se pudo abrir"
      severity failure;
-- lecturas
file_close(fichero);
```

Sintaxis

```
file_open(file_status, file, file_name, open_kind);

open_kind = read_mode, write_mode, append_mode

file_status = open_ok, status_error (fichero ya abierto), name_error, mode_error


file_close(file);
```

Acceso a ficheros de forma interactiva

En el siguiente ejemplo se muestra una forma de solicitar el nombre de un fichero de forma interactiva. Las salidas estandar en VHDL son “input” y “output”.

Ejemplo

```
process is
variable linea_F: line;
variable nombre: string(1 to 10);
. . .
write (output, “Indicar el fichero: “);
readline (input, linea_F);
read (linea_F, nombre);
. . .
end;
```

Apéndice 3.12: Implementación de elementos de almacenamiento

En este apéndice se expone la implementación de elementos de almacenamiento (celda y registro) utilizando puertas lógicas.

Implementación de un multiplexor

En la Figura 3.73 se muestra un multiplexor implementado con puertas lógicas. La señal sel se utiliza para seleccionar cuál de las entradas es seleccionada como salida. En la misma figura se muestran tres diagramas temporales. En el primer diagrama (a) el retardo del multiplexor son 2 retardos de puerta (t_p), mientras que en el segundo diagrama (b) son tres retardos de puerta.

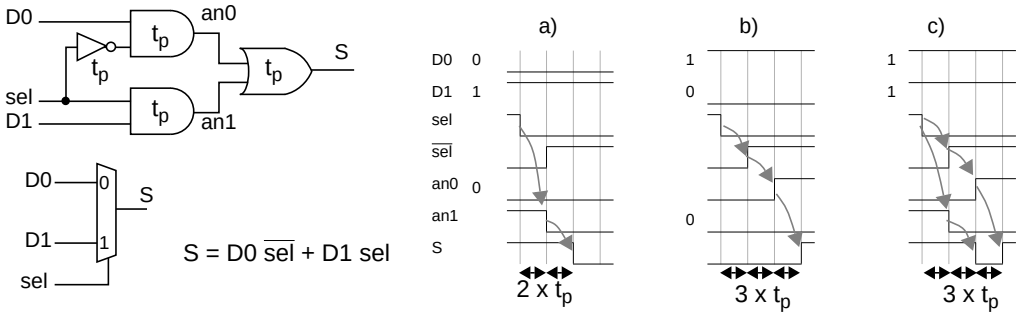


Figura 3.73 *Implementación de un multiplexor con puertas. Diagramas temporales.*

Transitorio estático. En el tercer diagrama (c) de la Figura 3.73 se observa un cambio espúrio transitorio de la señal. Este cambio transitorio está relacionado con el retardo del inversor. El retardo del camino de las entradas a la puerta OR es mayor en el camino que incluye el inversor.

El cambio transitorio espúrio es estático y puede eliminarse añadiendo lógica. En la Figura 3.74 se muestra que añadiendo una puerta and, cuyas entradas son D0 y D1, no se producen cambios transitorios estáticos espúrios[45]. La nueva puerta mantiene el valor uno en la salida mientras las otras puertas evalúan.

45. Un análisis de la tabla de Karnaugh permite determinar que puertas adicionales son necesarias para eliminar el transitorio estático. Observemos que el transitorio es debido únicamente a un cambio en una de las señales de entrada. El conjunto de entradas previo y el actual, en el cual sólo ha cambiado el valor de una señal, determinan el mismo valor de la salida. Ahora bien, durante un transitorio la salida tiene el valor complementario. Por otro lado, un transitorio espúrio es dinámico si la salida cambia más de una vez respondiendo a un cambio en una única entrada. Los niveles de salida son distintos antes y después de que se produzca el transitorio. Un transitorio dinámico se puede producir si existen varios caminos con retardos distintos desde la entrada que cambia de nivel a la salida.

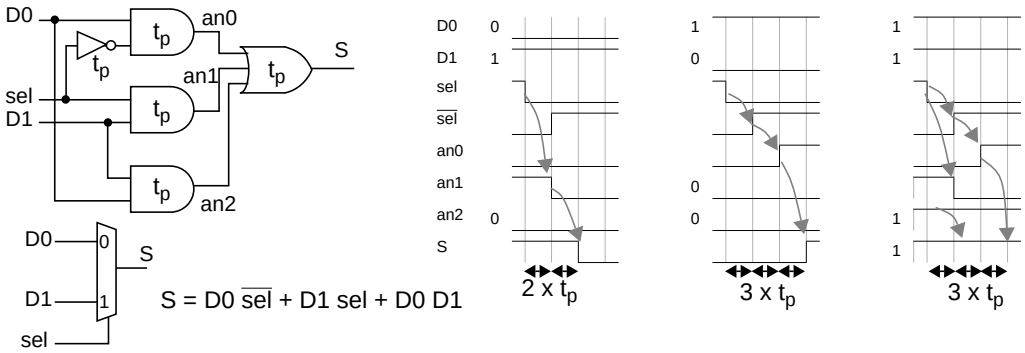


Figura 3.74 *Mutliplexor modificado libre de transitorios estáticos.*

Transitorio funcional (o de especificación). La modificación del diseño del multiplexor garantiza que no se producen cambios transitorios si sólo cambia una de las señales de entrada en un instante determinado. Ahora bien, en la figura se muestra un diagrama temporal donde hay dos señales (D1 y sel) que cambian concurrentemente. En la salida se produce un transitorio espúrio[46].

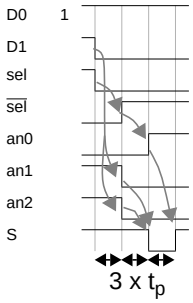


Figura 3.75 *Diagrama temporal con un transitorio funcional.*

Estos cambios transitorios no pueden elimarse de forma estática, ya que son debidos a la especificación del circuito. La forma de soportarlos o soslayarlos (resolver el problema) es restringir los cambios de las entradas, de forma que sólo una entrada pueda cambiar en un instante determinado.

46. Este transitorio es debido a que el retardo por los distintos caminos, desde las entradas a la salida, es distinto.

Implementación de una celda

La utilización de un multiplexor para implementar una celda o latch se muestra en la Figura 3.76. La señal sel es el permiso de escritura (PE), la entrada D0 se conecta con la salida y la entrada D1 (D) es la entrada de la celda. Existe un camino de realimentación de la salida (Q) a una de las entradas (D0). La celda es transparente mientras la señal PE tiene el valor uno. Esto es, la entrada se propaga a la salida y cambios de nivel en la entrada (D) se observan en la salida después de un retardo. Cuando el valor de la señal PE es cero la celda es opaca. Esto es, la entrada D0 = Q', que está conectada a la salida, se transmite a la salida y cambios de nivel en la señal D no son observados en la salida.

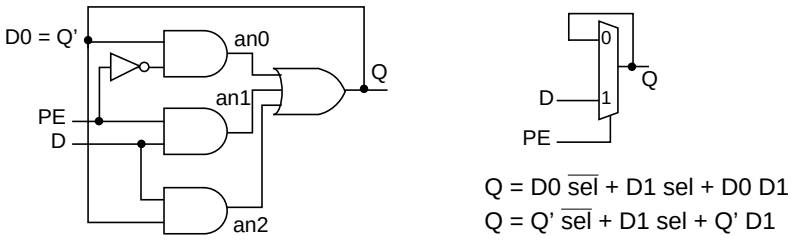


Figura 3.76 *Celda implementada con un multiplexor.*

Teniendo en cuenta el comportamiento del multiplexor a cambios concurrentes (Figura 3.77) podemos inferir que en la salida de la celda se observa un transitorio durante un periodo de tiempo, después de cambios concurrentes en las señales PE y D. Este transitorio hace oscilar al circuito. El pulso que se produce en la salida de la Figura 3.77 se propaga a la entrada (D0 = Q') y desde esta a la salida (Q, Figura 3.77), utilizando el camino de realimentación. La anchura del pulso es un retardo de puerta. Este transitorio persiste mientras no se modifique el valor de una entrada[47].

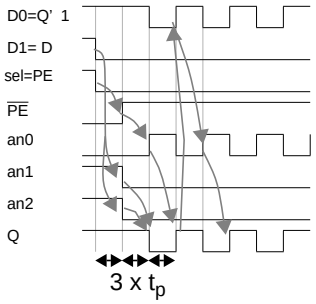


Figura 3.77 *Oscilación en la celda.*

47. Si el retardo entre los cambios de nivel de D1 y PE es un ciclo también se produce una oscilación.

Protocolo de cambios en las entradas. Por todo ello, se define un protocolo de actuación (restricciones) en las señales de entrada durante los periodos que pueden determinar cambios de estado en la celda. Esto es, la ventana temporal que abraza el cambio de transparente a opaca de la celda.

En primer lugar, sólo una de las entradas pueda modificar su valor en un instante determinado[48]. En concreto, se establece un tiempo de estabilidad en la señal D, antes de que la señal PE cambie (t_e). También se establece un tiempo de mantenimiento (t_m) en la señal D, después de que la señal PE haya modificado su valor. En resumen, la señal de entrada debe ser estable un tiempo t_e antes de un cambio de nivel de 1-> 0 en la señal PE y un tiempo t_m después del cambio de nivel.

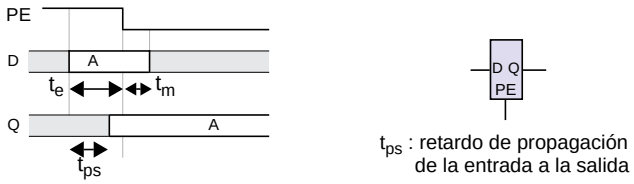


Figura 3.78 *Restricciones temporales de las señales de entrada en una celda.*

Tiempo de estabilidad (t_e). Garantiza que el valor de la entrada (D) se propaga a través del camino de realimentación, antes de que se desactive el permiso de escritura (PE). Esto es, la celda sea opaca.

En un primer lugar efectuamos un análisis suponiendo que el multiplexor es una caja negra. Sólo observamos el camino de realimentación de la sadlida Q a una entrada (D0 = Q'). Un cambio en la salida (Q), inducido por un cambio en la señal de entrada (D), debe de volver a propagarse a través del multiplexor (D0 = Q'). El retardo del multiplexor es t_{mux} = 3 x t_p (Figura 3.73). Entonces, la propagación de un cambio de nivel en una de las dos entradas (D, PE) requiere de un retardo t_{mux} (Figura 3.79). Posteriormente, la propagación de Q, utilizando el camino de realimentación a través del multiplexor, representa un retardo t_{mux}. En consecuencia, el tiempo de estabilidad es t_e = 2 x t_{mux}.

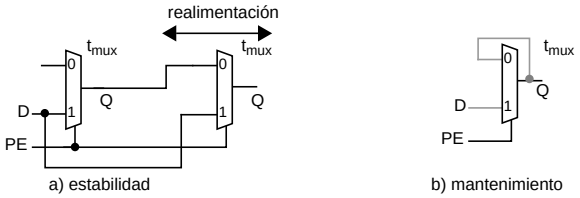


Figura 3.79 *Tiempo de estabilidad y tiempo de mantenimiento de una celda. El camino de realimentación se representa replicando el multiplexor.*

48. El efecto del cambio en la entrada debe propagarse a través del circuito. Además, un circuito debe ser estable antes de que se modifique una entrada.

Tiempo de mantenimiento (t_m). Garantiza que la celda es opaca y la salida (Q) es estable antes de que la entrada cambie.

El cambio en la señal PE debe propagarse a la salida. Durante este tiempo la señal D debe mantenerse estable, con el objetivo de que Q no se modifique (Figura 3.79). En consecuencia, el tiempo de mantenimiento es $t_m = t_{mux}$.

Seguidamente efectuamos un análisis más detallado de los retardos t_e y t_m. En concreto, dejamos de observar el multiplexor como una caja negra.

Respecto al tiempo de estabilidad hay que determinar cuanto tiempo, antes de que la celda se vuelva opaca (transición 1 -> 0), la entrada debe ser estable, para que la señal de entrada (D) se haya propagado.

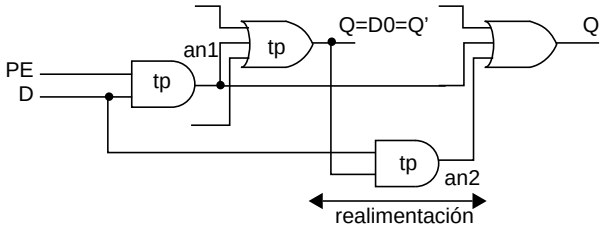


Figura 3.80 *Análisis del tiempo de estabilidad de una celda.*

La entrada D debe propagarse por las puertas AND cuyas salidas son an1 y an2. Posteriormente debe propagarse por la puerta OR. Finalmente, la salida Q = Q' = D0 debe propagarse por la puerta AND cuya salida es an2 (Figura 3.80). En este instante de tiempo se tiene garantia de que las señales an2 y Q tienen el mismo valor. Durante este intervalo de tiempo la señal PE debe mantener el nivel 1. La señal D no ha cambiado de nivel y por tanto, las señales an1 (D y PE) y Q tampoco han cambiado de nivel[49]. En consecuencia, $t_e = 3 \times t_p$.

En la Figura 3.81 se muestra la evolución de las señales cuando la celda es transparente y hay un cambio de nivel en la señal D. El valor de t_e es 3 x t_p [50].

49. Tengamos en cuenta que los cambios de nivel de las señales no son abruptos (instantáneos). por otro lado, el retardo de una puerta depende del cambio de nivel que se produce. También, el retardo de dos puertas iguales no es exactamente el mismo.

50. En esta exposición se ha supuesto que las puertas se comportan como líneas de retardo. Esto es, cualquier cambio en la entrada de una puerta se propaga a la salida después del retardo de puerta. En una implementación física una puerta no se comporta de esta forma. Una puerta puede filtrar la propagación de cambios de nivel de pequeña duración en las entradas. Por otro lado, no es suficiente un único parámetro de retardo para caracterizar una puerta. Un análisis preciso requiere modelar las puertas con la implementación utilizada a nivel tecnológico (transistores e interconexión) y utilizar un simulador apropiado para este nivel de especificación.

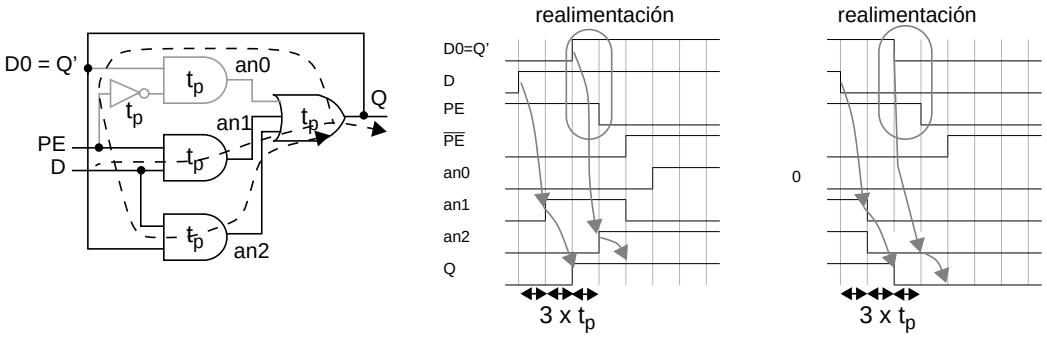


Figura 3.81 *Tiempo de estabilidad de una celda. Diagrama temporal.*

Respecto al tiempo de mantenimiento hay que determinar cuando tiempo, después de que la celda se vuelva opaca (transición 1 -> 0), la entrada D debe ser estable, para que la señal de entrada se almacene en la celda.

La entrada PE debe propagarse por el inversor y la puerta AND cuya salida es an0. En este instante de tiempo se tiene garantia de que la entrada an0 de la puerta OR y la salida de la puerta OR tienen el mismo nivel de señal.

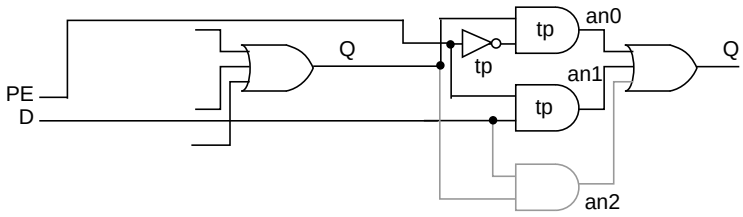


Figura 3.82 *Análisis del tiempo de mantenimiento de una celda.*

En la Figura 3.83 se muestra la evolución de las señales cuando la celda se vuelve opaca. Esto es, hay un cambio de nivel en la señal PE. El valor de t_m es 2 x t_p.

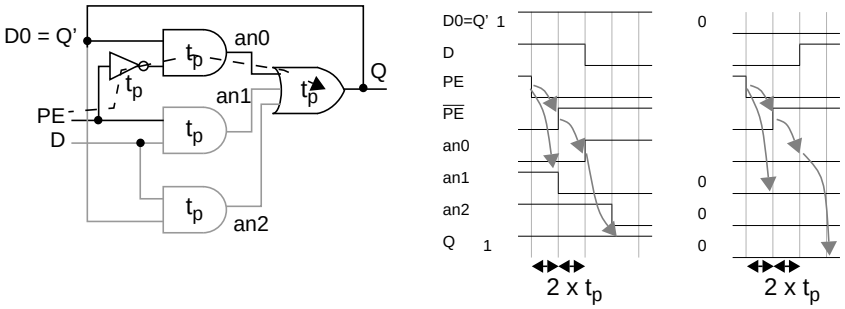


Figura 3.83 *Tiempo de mantenimiento de una celda. Diagrama temporal.*

En la Figura 3.84 se muestran los mismos casos, que en la Figura 3.81 y en la Figura 3.83, con un valor de t_e igual a 2 x t_p y un valor de $t_m = t_p$. Respecto al retardo t_e, notemos que en el diagrama de la izquierda las señales an1 y an2 cambian de nivel en el mismo instante en sentidos opuestos. Cualquier variación relativa en los retardos de las puertas AND entre sí puede producir un pulso en la salida, que puede propagarse a través del camino de realimentación. Respecto al retardo t_m, notemos que en el diagrama de la izquierda las señales an0 y an2 cambian de nivel en el mismo instante en sentidos opuestos. Cualquier variación relativa en los retardos de las puertas AND entre sí puede producir un pulso en la salida, que puede propagarse a través del camino de realimentación.

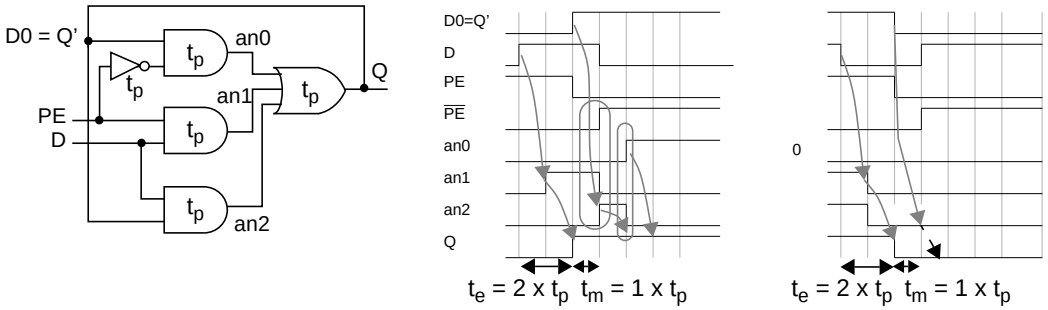


Figura 3.84 *Evolución de las señales en una celda. Tiempos de t_e y t_m menores que los necesarios.*

Retardo de propagación. Cuando la señal PE tiene el nivel 1 (celda transparente), el retardo de propagación t_{ps} (Figura 3.78) es 2 x t_p (Figura 3.84). En caso contrario (PE = 0), el retardo de propagación, al conmutar la celda a transparente, es 3 x t_p (Figura 3.85).

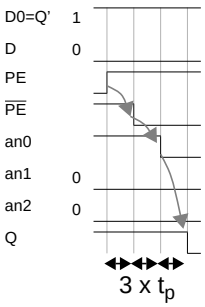


Figura 3.85 *Retardo de propagación cuando una celda conmuta a transparente.*

Circuito equivalente de una celda

Partiendo del circuito de la parte izquierda del la Figura 3.86 y utilizando el álgebra de Boole, se obtiene el circuito que se muestra en la parte central de la Figura 3.86. Seguidamente, reorganizando la disposición espacial de las puertas se obtiene el esquema de la derecha de la Figura 3.86. En este esquema se identifica una celda SR (basada en NAND), con una señal de control (PE) y la especialización para que se comporte como una celda D.

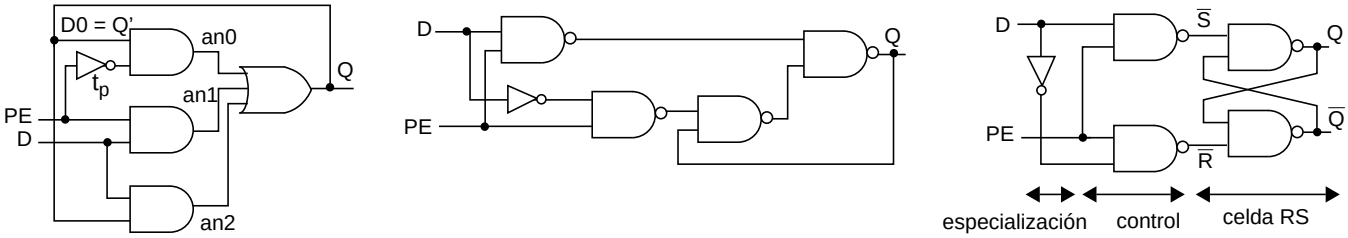


Figura 3.86 *Circuitos equivalentes de una celda de tipo D.*

Registro

Una celda propaga a la salida la información que hay en la entrada durante un nivel de la señal de reloj (transparente). En el otro nivel de la señal de reloj, la celda es opaca. En cambio, un registro almacenada la información que hay en la entrada en el flanco de reloj. En cualquier otro instante de tiempo es opaco. Por ello, es más sencillo diseñar circuitos síncronos con registros como elementos de desacoplo entre circuitos combinacionales.

Un registro se puede construir con dos celdas (Figura 3.88). La celda de entrada se denomina maestra y la celda de salida esclava. Mientras la celda de entrada es transparente la celda de salida es opaca y viceversa.

En el flanco ascendente de la señal de reloj la celda maestra es opaca hasta el flanco descendente. La celda esclava es transparente y se propada el valor almacenado en la celda maestra. La salida del registro es el valor almacenado en la celda maestra.

En el flanco descendente hasta el siguiente flanco ascendente la celda maestra es transparente y la celda esclava es opaca. La celda esclava tiene almacenado el valor que propagaba en el semiperiodo previo. En la Figura 3.87 se muestra un ejemplo donde sólo se considera retardo de propagación en la celda. El retardo del inversor (Figura 3.88) se considera igual a cero.

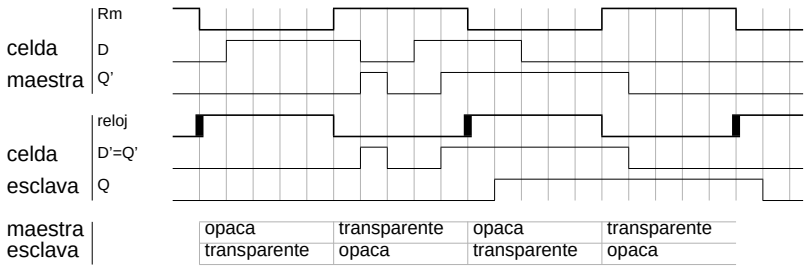


Figura 3.87 *Evolución de las señales en un registro.*

Las restricciones temporales en un registros emanan de los elementos que se utilizan en su construcción. El almacenamiento de un valor requiere de un tiempo de estabilidad (t_e) y de un tiempo de mantenimiento (t_m) en la entrada, los cuales abrazan al flanco de la señal de reloj. Estos valores deben garantizar que las restricciones de tiempos de estabilidad y manteniemiento de las celdas, con las que ha sido construido el registro, se cumplan.

En concreto, las restricciones de tiempos de estabilidad y mantenimiento del registro están asociadas a las restricciones con el mismo nombre en la celda maestra. La celda maestra asegura las restricciones de tiempo de estabilidad de la celda esclava.

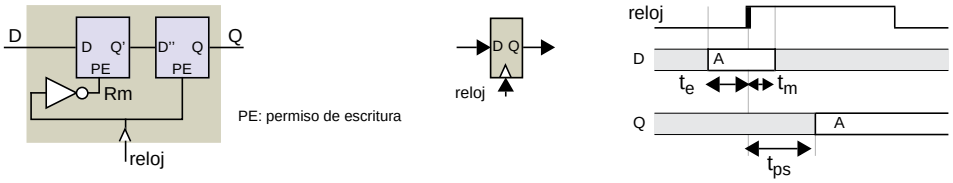


Figura 3.88 *Registro construido con dos celdas.*

La salida de la celda maestra debe mantenerse estable durante el tiempo de mantenimiento de la celda esclava. Esto es, el tiempo de mantenimiento en la celda esclava tiene que ser menor que el retardo del inversor más el retardo en el cual se observa un cambio en la salida de la celda maestra, aunque sea transitorio (retardo de contaminación).

En la Figura 3.89 y en la Figura 3.90 se muestran diagramas temporales de los retardos en los componentes de un registro. En los diagramas se identifican los tiempos de estabilidad y mantenimiento del registro (t_e, t_m) y de las celdas (t'_e, t'_m) con las cuales ha sido construido el registro. También se muestra el retardo de propagación en el registro (t_{ps}) y en una celda (t_{pcel}).

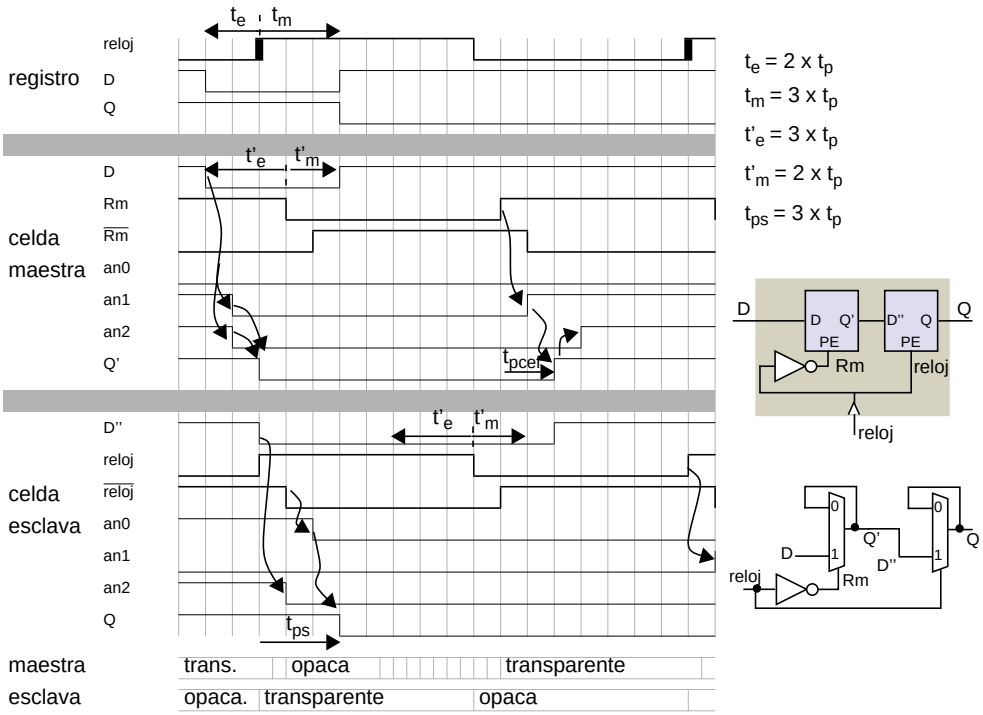


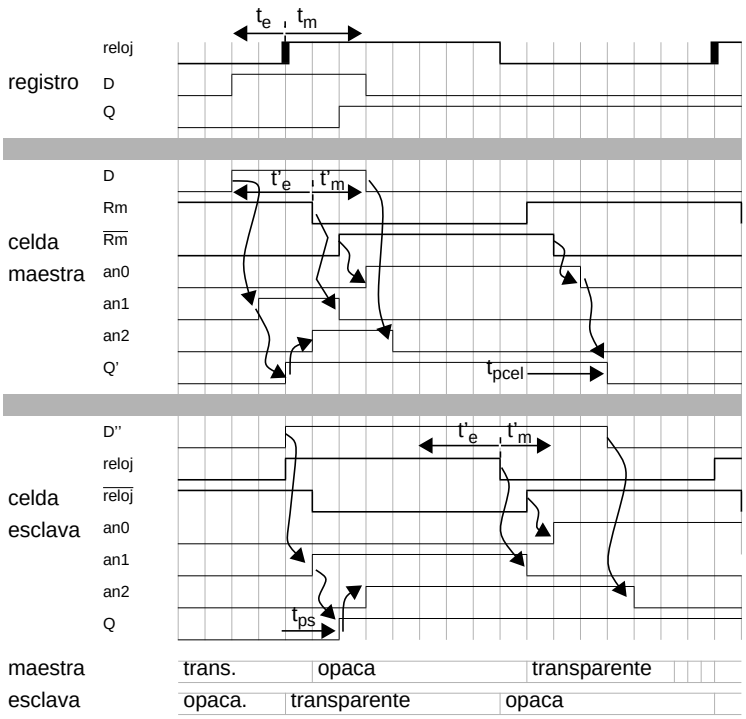
Figura 3.89 *Evolución temporal de las señales en un registro y en los elementos que lo constituyen. Transición 0 -> 1 en la entrada.*

En el diseño que se utiliza para construir un registro, los tiempos de estabilidad y mantenimiento de un registro están relacionados con los tiempos de estabilidad y mantenimiento de la celda maestra y el desplazamiento del reloj. La señal PE en la celda maestra es la señal de reloj desplazada un retardo de puerta (inversor). Entonces, el tiempo de estabilidad del registro (t_e) es un retardo menor que t'_e (celda) y el tiempo de mantenimiento del registro t_m es un retardo mayor que t'_m (celda), para que el t'_m sea igual a dos retardos de puerta.

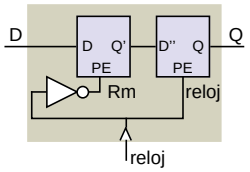
El tiempo de mantenimiento en la celda esclava se cumple, ya que el retardo de propagación de la celda maestra (t_{pcel}) es 2 x t_p y además hay que añadir el retardo del inversor.

El retardo de propagación del registro (t_{ps}) es igual al retardo de tres puertas. La celda esclava debe volverse transparente (inversor, puerta AND cuya salida es an0 y puerta OR).

Los retardos te y tm está relacionados con la anchura mínima de la señal de reloj en el nivel bajo y alto respectivamente. Esta duración debe permitir la propagación de la entrada respectiva a través de la celda y la estabilización de la salida.



$t_e = 2 \times t_p$
$t_m = 3 \times t_p$
$t'_e = 3 \times t_p$
$t'_m = 2 \times t_p$
$t_{ps} = 2 \times t_p$



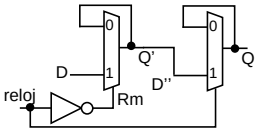


Figura 3.90 *Evolución temporal de las señales en un registro y en los elementos que lo constituyen. Transición 1-> 0 en la entrada.*

Observemos que durante un retardo t_p, después de los flancos de la señal de reloj, las dos celdas son transparentes u opacas. Este intervalo de tiempo está incluido en la restricción t'_m (celda).

La salida del registro mantiene el valor previo durante un tiempo mínimo de 2 x t_p, siendo el retardo 3 x t_p.

En la Figura 3.91 se muestra otra posible organización de las celdas para construir un registro. En este diseño el desplazamiento del reloj se observa en la celda esclava. Entonces, los tiempo de estabilidad y mantenimiento del registro son los respectivos tiempos de la celda maestra.

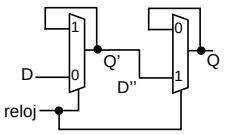


Figura 3.91 *Registro construido con dos celdas con distintas conexiones de realimentación.*

Celda controlada por pulso

La estructura de la celda se muestra en la Figura 3.92. Esta celda se utiliza como sustituto de un registro. La idea es que el pulso sea la suficientemente estrecho para que pueda considerarse como un flanco. Ahora bien, la anchura del pulso de ser suficiente para que la celda almacene el dato. Por otro lado, dado un diseño, el pulso se genera para cada celda o un grupo de celdas próximas. La señal que se distribuye es el reloj.

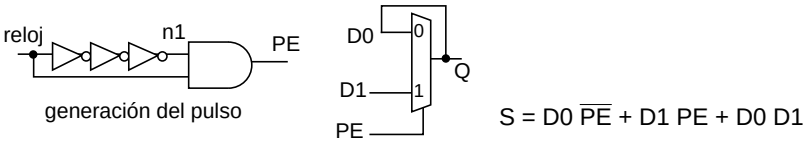


Figura 3.92 *Celda controlada por pulso.*

Generador de pulso. La Figura 3.93 muestra un diagrama temporal del pulso que se genera utilizando el circuito de la Figura 3.92. Se supone que los retardos de un inversor y una puerta AND son idéntico (t_p).

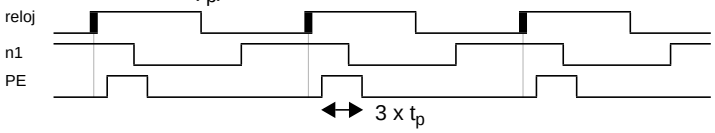


Figura 3.93 *Diagrama temporal de la generación del pulso.*

En la Figura 3.94 y en la Figura 3.95 se muestran diagramas temporales de la evolución de las señales en los distintos componentes. Durante el pulso la celda es transparente. Al finalizar el pulso se almacena el valor presente en la entrada.

El pulso debe tener una anchura suficiente para que se propage la entrada (D) cuando la celda es transparente. Notemos que el tiempo de estabilidad, teniendo como referencia la señal de reloj, es negativo. por otro lado, el tiempo de mantenimiento es mayor que la anchura del pulso.

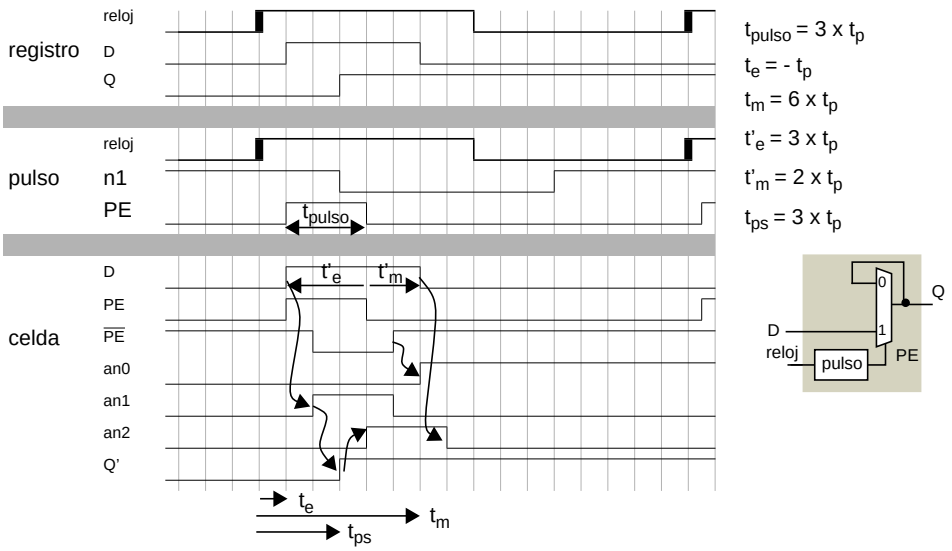


Figura 3.94 *Celda controlada por pulso. Cambio de nivel 0-> 1 en la señal D.*

Cuando se utiliza una celda controlada por pulso en un diseño segmentado, la anchura del pulso debe ser menor que el menor retardo a través de la lógica que alimenta a la celda.

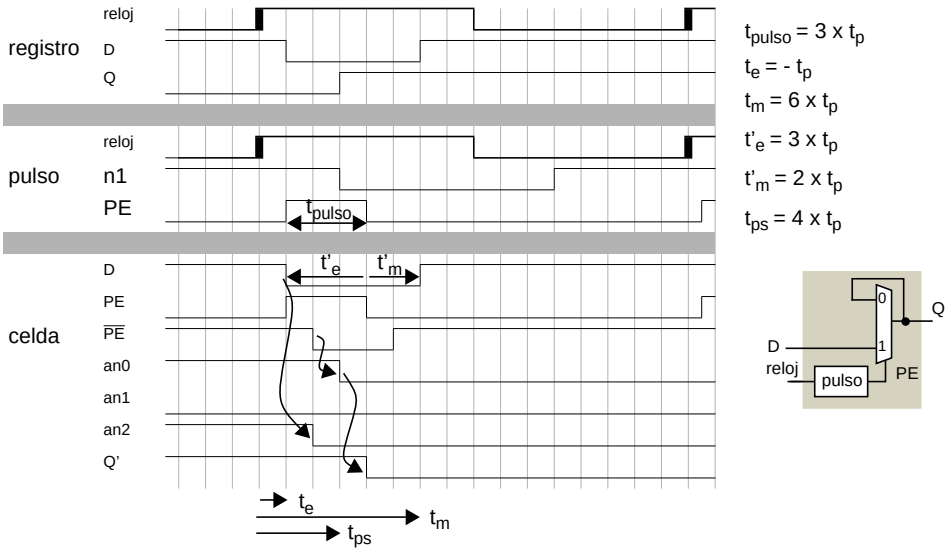


Figura 3.95 *Celda controlada por pulso. Cambio de nivel 1-> 0 en la señal D.*

Especificación funcional de componentes básicos de almacenamiento

En la Figura 3.96 se muestra la especificación funcional de una celda.

```
library ieee;
use ieee.std_logic_1164.all;
entity celda is
    generic (tp: time := 2 ns);
    port (D: in std_logic;
          PE: in std_logic;
           Q: out std_logic);
end celda;
architecture compor of celda is
begin
cel: process (D, PE) begin          -- cualquier evento en D o PE determina que el proceso evalúe
    if PE = '1' then                -- mientras PE = 1, la entrada se propaga a la salida
        Q <= D after tp;
    end if;
end process;
end;
```

Figura 3.96 *Especificación funcional en VHDL de una celda.*

En la Figura 3.97 se muestra la especificación funcional de un registro.

```
library ieee;
use ieee.std_logic_1164.all;
entity registro is
    generic (tp: time := 2 ns);
    port (D: in std_logic;
          reloj: in std_logic;
           Q: out std_logic);
end registro;
architecture compor of registro is
begin
reg: process (reloj) begin          -- un evento en la señal de reloj determina que el proceso evalúe
    if rising_edge(reloj) then      -- en el flanco ascendente, la entrada se propaga a la salida
        Q <= D after tp;            -- para detectar el flanco ascendente se puede utilizar la función
    end if;                         -- rising_edge(reloj)
end process;
end;
```

Figura 3.97 *Especificación funcional en VHDL de un registro.*

Comprobación de restricciones temporales. A las especificaciones previas se pueden añadir sentencias que comprueban las restricciones temporales.

Para ello se utilizan atributos de los objetos, en concreto, de los objetos tipo señal.

- S'delayed(t). Es el valor de la señal S en el tiempo now[51] - t.

- S'last_event. Es el tiempo transcurrido desde el último evento en la señal S.

La restricción de tiempo de estabilidad en un registro se puede comprobar de la siguiente forma en VHDL. Cuando el reloj efectúa la transición 0->1, si la entrada no ha estado estable (no se ha producido un evento) al menos un tiempo t_e, entonces no se ha cumplido la restricción de tiempo de estabilidad.

```
assert not (reloj'event and reloj ='1' and D'last_event < te)
    report "incumplimiento del tiempo de estabilidad"
    severity "warning";
```

La restricción de tiempo de mantenimiento se puede comprobar con la siguiente sentencia en VHDL. Cuando se produce un cambio en la entrada, si el reloj tiene el valor '1' y el reloj ha tomado este valor un tiempo previo menor que t_m, entonces no se ha cumplido la restricción.

```
assert not (reloj'delayed(tm)'event and reloj'delayed(tm) ='1' and D'last_event < tm)
    report "incumplimiento del tiempo de mantenimiento"
    severity "warning";
```

Los parámetros de restricción temporal se pueden incluir en la especificación de la inteface. En la Figura 3.98 se muestra su inclusión en la especificación de un registro.

```
library ieee;
use ieee.std_logic_1164.all;
entity registro is
   generic (tp: time := 2 ns; te: time := 0.5 ns: tm: time := 0.5 ns);
   port (D: in std_logic;
         reloj: in std_logic;
          Q: out std_logic);
   begin
   assert not (reloj'event and reloj ='1' and D'last_event < te)
       report "incumplimiento del tiempo de estabilidad"
       severity "warning";
   assert not (reloj'delayed(tm)'event and reloj'delayed(tm)= '1' and D'last_event < tm)
       report "incumplimiento del tiempo de mantenimiento"
       severity "warning";
end registro;

architecture compor of registro is
begin
reg: process (reloj) begin
    if rising_edge(reloj) then
        Q <= D after tp;
    end if;
end process;
end;
```

Figura 3.98 *Especificación en VHDlL de las sentencias que se utilizan para comprobar las restricciones temporales de un registro.*

51. now es una función de VHDL que suministra el tiempo de simulación actual.

Apéndice 3.13: Implementación de matrices de almacenamiento

En este apéndice se describen varias estructuras de almacenamiento. Se analiza la organización de las mismas y sus restricciones temporales.

Banco de registros

En la Figura 3.99 se muestra la implementación mediante registros de 1 bit de un banco de registros. La matriz que se muestra dispone de un puerto de lectura y un puerto de escritura. La escritura es síncrona con la señal de reloj y la lectura es asíncrona.

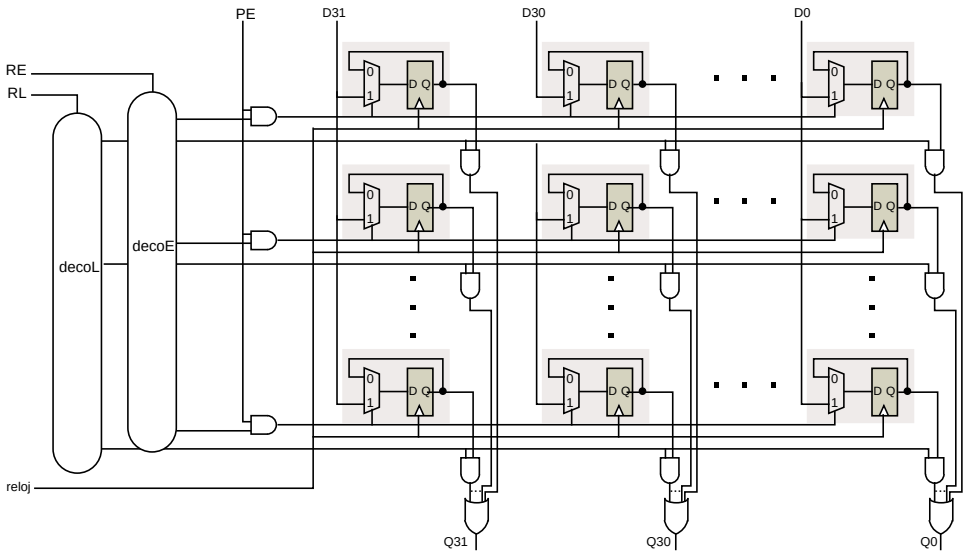


Figura 3.99 *Detalle de un banco de registros con un puerto síncrono de escritura y un puerto asíncrono de lectura.*

El diseño previo es factible cuando el número de elementos de almacenamiento es pequeño. Para cada bit de almacenamiento es necesario un cable que emerja de la retícula (matriz) de celdas. Una alternativa es distribuir la puerta OR (Figura 3.100). Cada celda tiene asociada una puerta AND y una puerta OR. El área necesaria en cada columna es equivalente a la de un cable. La reducción de cableado en las columnas es significativa.

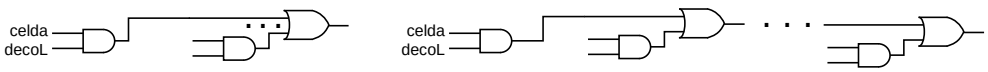


Figura 3.100 *Parte del camino de lectura.*

Otra alternativa es utilizar un buffer de tres estados (Figura 3.101). Este elemento, mediante una señal de control, conecta la entrada a la salida (conmutador cerrado, 0, 1) o la desconecta (conmutador abierto). La señal de control de los buffer de tres estados son las salidas del decodificador decoL. Por otro lado, en el cableado de las columnas no hay que utilizar puertas lógicas en cada celda.

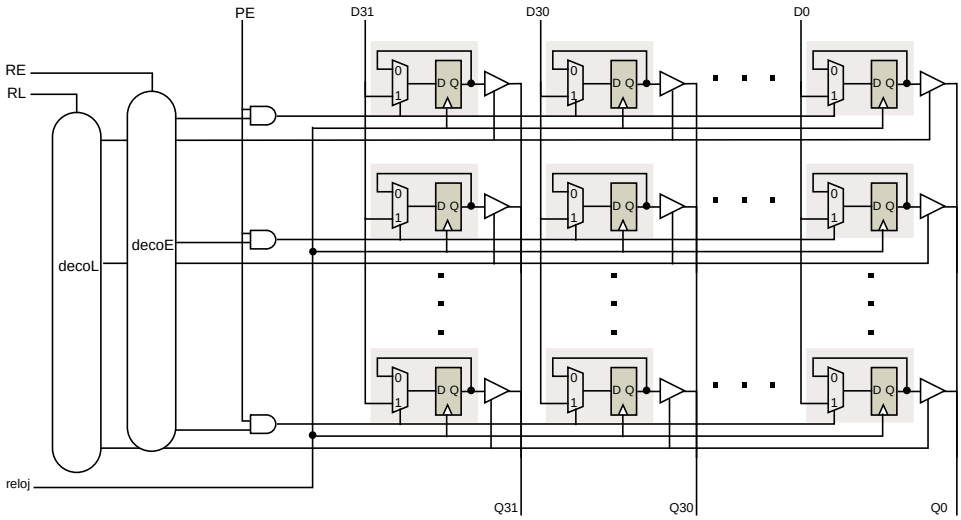


Figura 3.101 *Banco de registros. Buffer de tres estados para la lectura.*

Compartición de los cables de datos para lectura y escritura. En el diseño previo se utilizan cables de datos distintos para lecturas y escrituras. Una forma de reducir el área ocupada es utilizar un único cable. En estas condiciones se elimina un decodificador. El mismo puerto de acceso se utiliza para lecturas y escrituras de forma excluyente. En la Figura 3.102 se muestra un esquema. Para efectuar la compartición se utilizan buffer de tres estados. Estos buffer están controlados por las señales LE y PE.

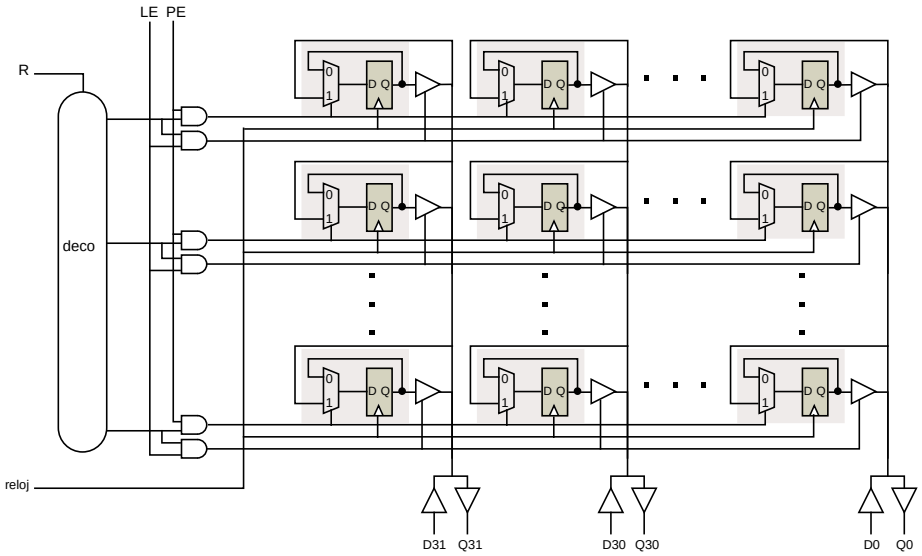


Figura 3.102 *Puerto de acceso compartido por lecturas y escrituras.*

Aspecto de una matriz de almacenamiento

La forma o aspecto u organización de la matriz de celdas influye en la longitud del cableado que se utiliza para su funcionamiento (Figura 3.103). La longitud de los cables determina el retardo de acceso. Por ello, es de interés independizar el aspecto de la matriz de celdas del número de bits que se quiere leer o escribir en un instante determinado.

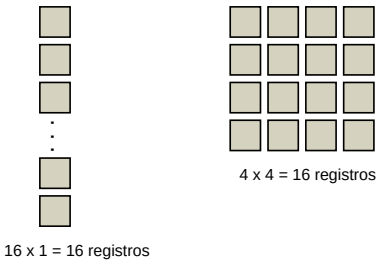


Figura 3.103 *Organización de una matriz de celdas.*

En estas condiciones, en una fila de la matriz hay más bits de los que se quiere leer o escribir (Figura 3.104). Por tanto, son necesarias señales de control para determinar las columnas a las que se accede.

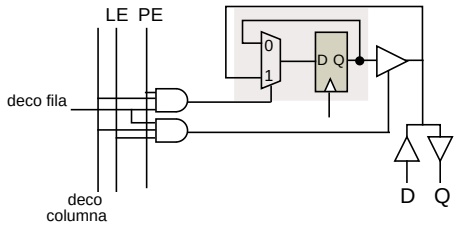


Figura 3.104 *Elemento y control del mismo en una matriz de celdas.*

En el siguiente ejemplo, un banco de memoria contiene 2^{31} bits y la matriz tiene el aspecto de un rectángulo de 2^{16} x 2^{15}. En un acceso se lee o escribe un bit. En consecuencia son necesarios 8 bancos para efectuar el acceso a un byte.

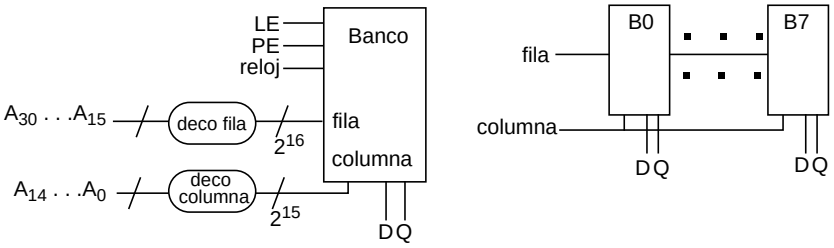


Figura 3.105 *Banco de memoria y organización de varios de ellos para leer un byte.*

Elemento básico de almacenamiento

En los diseño previos ha sido utilizado el registro como elemento de almacenamiento. Un registro almacena información en cada flanco de reloj.

Un registro se puede construir con dos celdas de almacenamiento que actúan por nivel. Una celda almacena la información que hay en la entrada cuando el nivel de la señal de reloj toma un valor determinado (cero o uno).

Para reducir el área que ocupa un conjunto de elementos de almacenamiento se pueden utilizar celdas en lugar de registros (celda de salida) y compartir la celda de entrada de un registro entre varias celdas. En concreto, todas las celdas. El conjunto de posiciones de almacenamiento sigue siendo síncrono.

Por otro lado, puede sustituirse esta celda compartida por un registro (Figura 3.106). De esta forma las señales de entrada al grupo de celdas funcionan de forma síncrona con el flanco de reloj. En contraposición, el grupo de celdas es asíncrono y queda sincronizado con el exterior por los registros que hay en las entradas.

Suponemos que las señales LE y PE no están activadas al unísono.

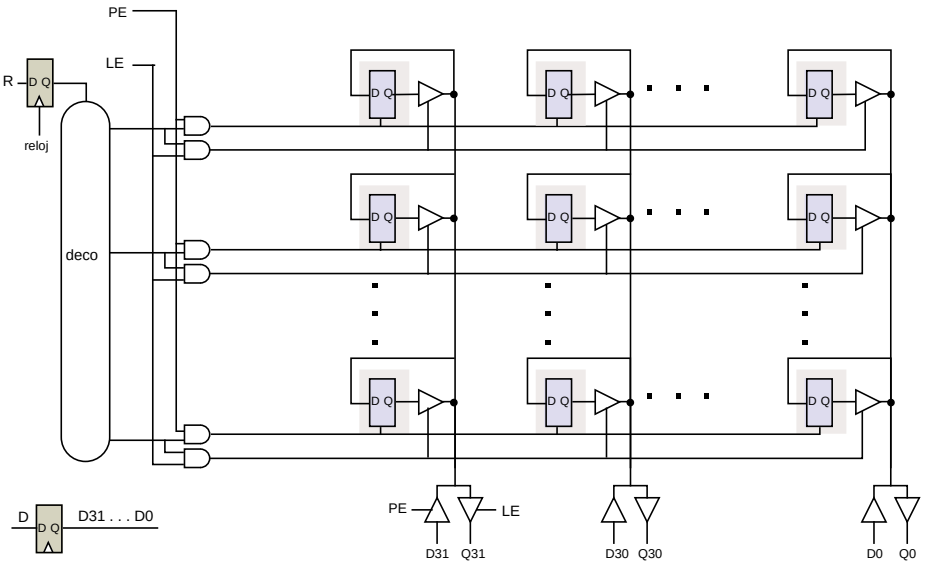


Figura 3.106 *Matriz de almacenamiento con celdas como elemento y registros en las entradas de dirección y datos.*

Memoria síncrona

Para gestionar el almacenamiento en las celdas es necesario un autómata de estados que genere las señales de control de las mismas y del encaminamiento de la información. La salida también se puede sincronizar con la señal de reloj (Figura 3.107).

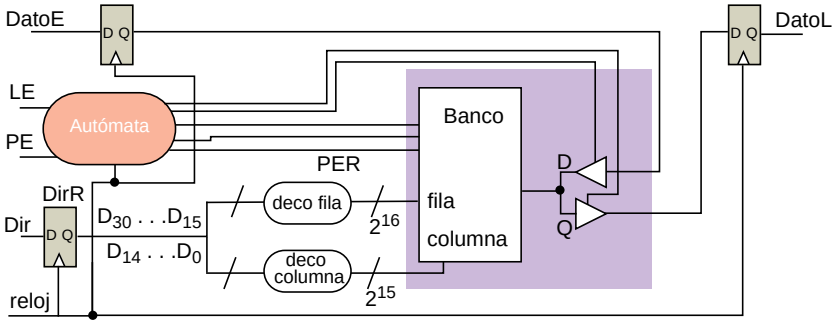


Figura 3.107 *memoria síncrona.*

Operación de lectura. En al Figura 3.108 se muestra la temporalidad en una operación de lactura. Al finalizar el ciclo en el cual ha sido muestreada la dirección se dispone del dato.

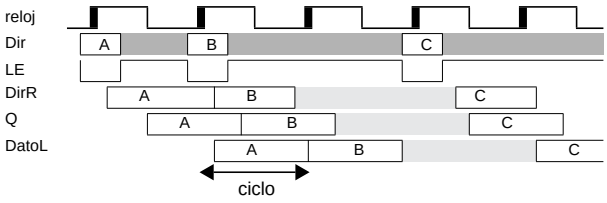


Figura 3.108 *Operaión de lectura.*

Operación de escritura. En la Figura 3.109 se muestra la temporalidad en una operación de escritura. Son necesarios dos ciclos para almacenar el dato. Es necesario retardar la llegada del dato.

La utilización de la misma temporalidad en una operación de lectura y escritura no es factible. El retardo de la dirección (selección de celdas) es mayor que el retardo del dato. Por tanto, es necesario retardar la llegada del dato.

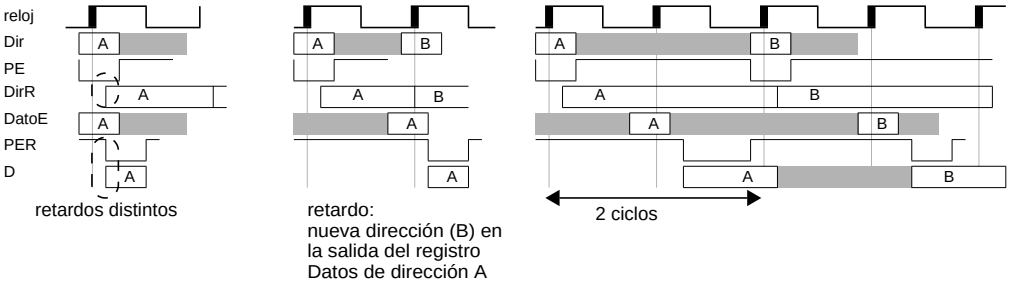


Figura 3.109 *Operación de escritura.*