PRACTICA: 3

CACHE CON ESCRITURA INMEDIATA

En esta práctica se diseñará una cache bloqueante, con escritura inmediata y sin asignación de contenedor en el caso de fallo en escritura.

En la Figura 1 se muestran las interfaces de la cache con el procesador y la memoria. La funcionalidad de las señales se describe en la tabla de la Figura 2.

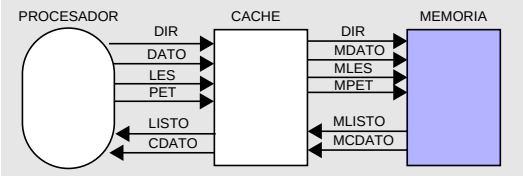


Figura 1 Interfaces entre procesador, cache y memoria.

Interface procesador / cache

	Descripción de las señales
DIR	dirección generada por el procesador
DATO	dato en una instrucción store
LES	indicación de lectura o escritura
PET	petición de acceso por parte del procesador
LISTO	operación finalizada
CDATO	dato suministrado por cache en una operación load

interface cache / memoria

	Descripción de las señales
DIR	dirección en un acceso a memoria
MDATO	dato en un acceso de escritura
MLES	indicación de lectura o escritura
MPET	petición de acceso por parte de la cache
MLISTO	operación finalizada
MCDATO	dato suministrado por memoria en una operación de lectura

Figura 2 Descripción de las señales de las interfaces entre procesador, cache y memoria.

Organización de la cache

Como es usual, una dirección de memoria referencia una posición que almacena un byte. Para simplificar el diseño, todos los accesos a memoria generados por el procesador son a byte.

La cache que se diseña es de mapeo directo y tiene 16 contenedores. En un contenedor se distinguen los campos: etiqueta, estado y bloque de datos (Figura 3). Sólo los bloque almacenados en cache tienen un hardware que representa su estado.

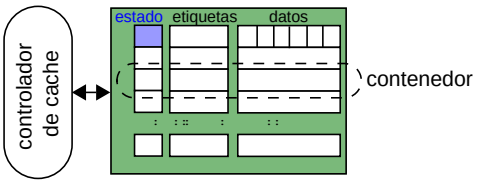
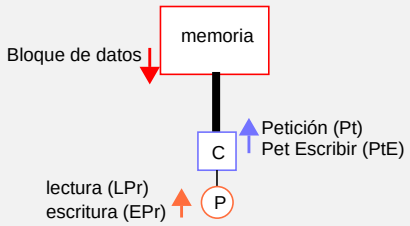


Figura 3 Memoria cache y campos en un contenedor de cache.

Para centrarnos en las acciones que debe realizar el controlador de cache cuando recibe peticiones del procesador, supondremos que el tamaño de bloque es igual a la granularidad de acceso del procesador: 1 bytes. El controlador de cache actualiza el estado de los bloques en respuesta a eventos del procesador y genera accesos a memoria.

Eventos del procesador y peticiones de acceso

En la tabla de la Figura 4 se describen los eventos del procesador y las acciones del controlador de cache.



Procesador	Controlador de cache
Peticiones	**Acciones**
LPr : lectura del procesador	Pt : petición de bloque
EPr : escritura del procesador	PtE : actualización de memoria
	CcRe: reemplazo de un bloque

Figura 4 Eventos del procesador y acciones del controlador de cache.

Diagrama de estados y transiciones de un bloque en cache

En la Figura 5 se muestran los dos estados de un bloque en cache y las transiciones entre ellos. El estado de un bloque sólo es explícito cuando está almacenado en cache. El estado inválido (I) indica que el contenedor no almacena un bloque válido. Un bloque que no está almacenado en un contenedor de cache se dice que no está presente (fallo de cache) y por tanto está en estado inválido. El estado válido (V) indica que el contenedor almacena un bloque válido. Una operación de reemplazo del controlador de cache selecciona un bloque y habilita el contenedor que lo almacena para otro uso (el bloque se invalida y se indica en el contenedor). En el caso que nos ocupa, escritura inmediata, esta operación no tiene asociada ninguna acción, ya que la memoria siempre está actualizada.

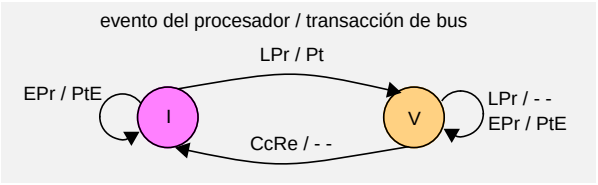


Figura 5 Diagrama de estados y transiciones de un bloque en cache.

Si en una operación de lectura (LPr) se produce acierto en cache se suministra el dato. En caso contrario se genera una petición de lectura del bloque, que contiene el dato accedido, a memoria (Pt). En una operación de escritura se genera una petición para actualizar memoria (PtE) y si es acierto, se actualiza el campo de datos en cache. La granularidad de una actualización de memoria es un byte. Notemos la diferencia en la granularidad de los accesos a memoria (bloque y byte). Para simplificar, en esta práctica se utiliza la misma granularidad.

Camino de datos y controlador de cache

En la Figura 6 se muestran las interfaces del camino de datos de la cache, el controlador de cache, el procesador y la memoria. Las señales entre el controlador de cache y el módulo campos se describen posteriormente.

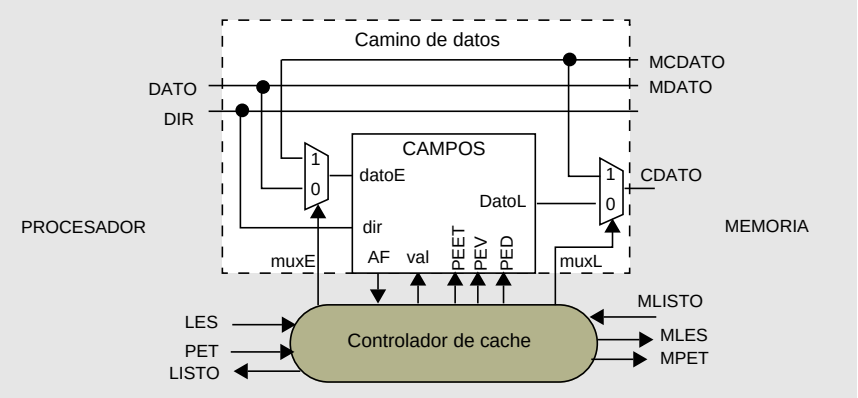


Figura 6 Memoria cache: módulo campos y controlador de cache.

En el camino de datos se distinguen dos multiplexores. El multiplexor muxE se utiliza para actualizar el módulo campos en fallos de lectura y aciertos de escritura. El multiplexor muxL se utiliza en un fallo de cache para suministrar el dato al procesador, en paralelo con el almacenamiento en el módulo campos.

Elementos en el módulo campos

En la Figura 7 se muestran los elementos del módulo campos. Se distinguen tres elementos de almacenamiento que se utilizan para implementar los tres campos de los contenedores de cache. Cada campo se implementa con un elemento de memoria que tiene una entrada de 4 bits (Dir) para indicar el contenedor accedido (campo índice de una dirección de memoria). Otra entrada es el camino de acceso para escrituras (datE), cuyo número de bits depende del campo de cache. En el caso del campo etiquetas son 12 bits, ya que suponemos direcciones de 16 bits. En el campo estado es un bit y en el campo datos es un byte. Además cada uno de los elementos de almacenamiento tiene un entrada de un bit para indicar si se quiere efectuar una operación de escritura (permiso de escritura, PEET, PEV, PED). Adicionalmente todos los elementos de almacenamiento tienen un camino de lectura. En estas condiciones, sólo hay un puerto de acceso a cache y el contenedor al que se accede se está leyendo y puede escribirse si está activado el permiso de escritura.

Los elementos de memorización utilizados son asíncronos. Esto es, no tienen como entrada la señal de reloj. Desde el controlador de cache sincronizaremos la actualización de los elementos de memorización con la señal de reloj. En concreto, se actualizarán en el nivel bajo de la señal de reloj.

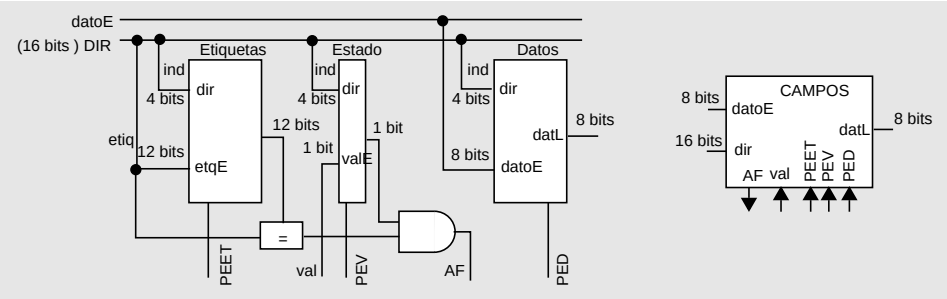


Figura 7 Elementos de almacenamiento y circuitos adicionales en el módulo campos.

Al implementar los elementos de memorización en LogicWorks la señal de permiso de escritura es negada (círculo en la entrada correspondiente de la RAM). Esto es, se escribe cuando la señal tiene el valor cero.

Flujo de información en el camino de datos

En este apartado se describe la utilización de los elementos del camino de datos para cada evento del procesador y la detección de acierto o fallo en cache. El camino de datos y la interface con memoria se muestra en la Figura 8. En los esquemas del camino de datos los multiplexores muxE y muxL de la Figura 6 están ubicados cerca del elemento de memorización que almacena el campo datos. En ocasiones las etiquetas de las señales en las Figura 8, Figura 9, Figura 10, Figura 11 y Figura 12 pueden ser distintas de las figuras previas.

Para simplificar la implementación supondremos que en la interface con memoria existe un bus que transporta la dirección a la cual se accede y dos buses para transportar datos. En uno de los buses el flujo de datos es de la cache a memoria (operación de esritura) y en el otro bus de datos el sentido es el contrario (operación de lectura). Las señales de la interface con memoria se detallan posteriormente.

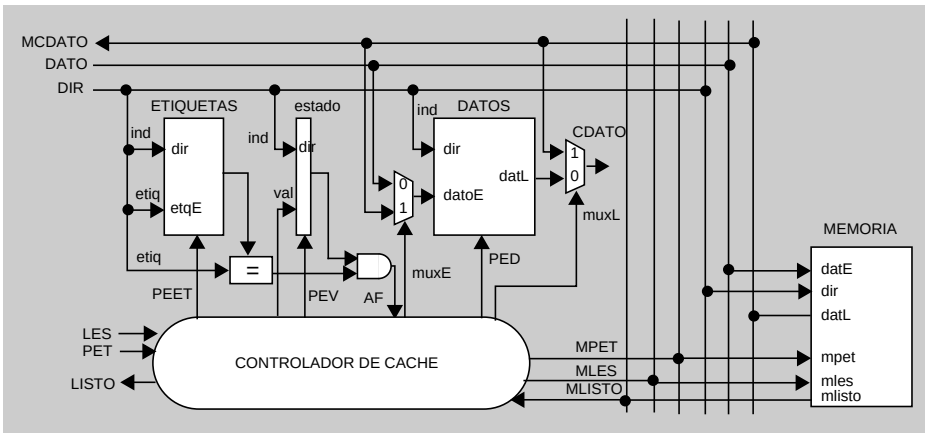


Figura 8 Memoria cache y campos en un contenedor de cache.

En las siguientes figuras se muestra la parte del camino de datos utilizada en cada una de los cuatro casos que se pueden producir en un acceso a cache.

- Petición de lectura y acierto en cache (Figura 9).

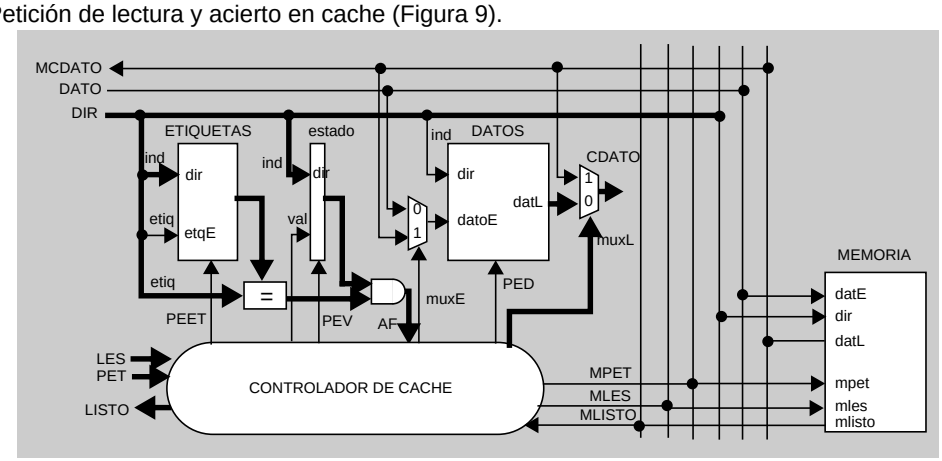


Figura 9 Acierto en lectura.

- Petición de lectura y fallo en cache (Figura 10).

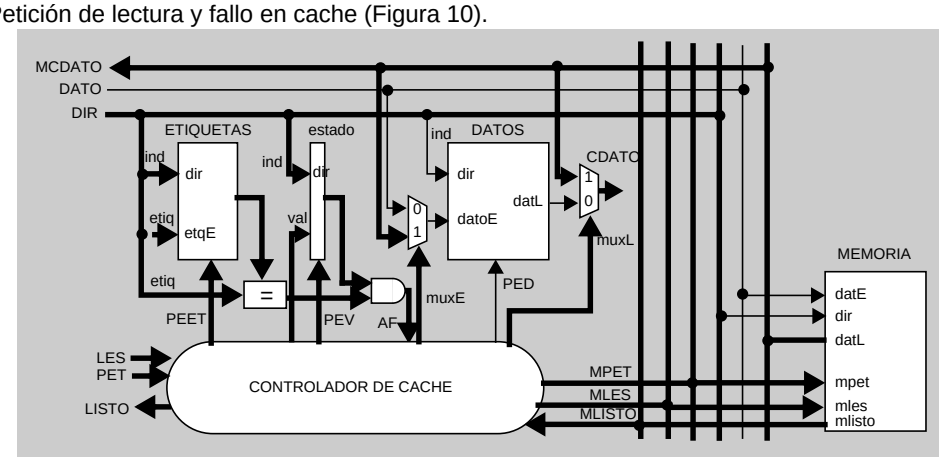


Figura 10 Fallo en lectura.

- Petición de escritura y acierto en cache (Figura 11).

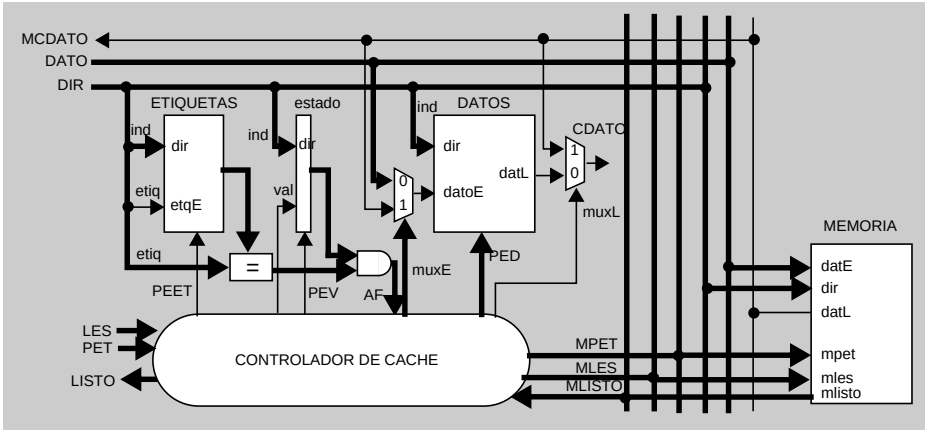


Figura 11 Acierto en escritura.

- Petición de escritura y fallo en cache (Figura 12).

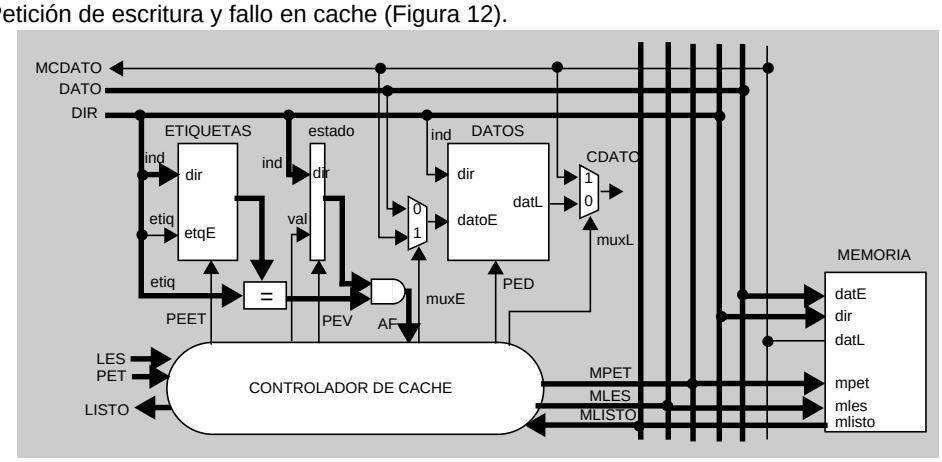
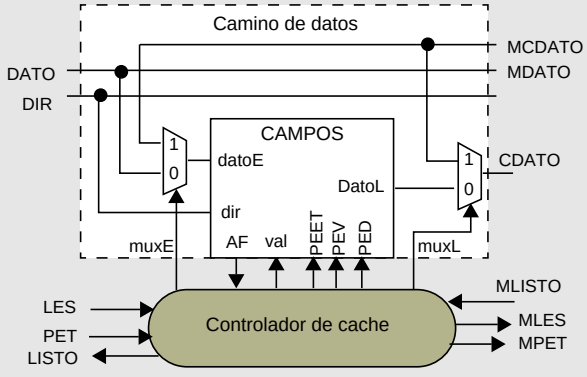


Figura 12 Fallo en escritura.

Controlador de cache

En la Figura 13 se muestra la interface entre el controlador de cache y el módulo campos, distinguiéndose las señales de estado y de control que se muestran en la tabla de la misma figura.



Señales de estado	Señales de control
AF : acierto / fallo	val : próximo estado del bloque
	PEET : permiso de escritura del campo etiqueta
	PEV : permiso de escritura del campo estado
	PED: permiso de escritura del campo datos
	muxE: selección del dato que se escribe
	muxL : selección del dato suministrado

Figura 13 Interfaces del controlador de cache y el módulo datos.

Descripción textual del controlador de cache

Cuando la señal PET se activa el controlador de cache utiliza la señal AF para determinar si se ha producido acierto o fallo. La señal LES indica si el acceso es de tipo lectura o escritura. Si la instrucción es un load y se detecta acierto, el dato que se ha leído del campo de datos se suministra al procesador. El procesador reconoce que ha finalizado el servicio observando la señal LISTO. En el caso de una escritura o un fallo de lectura es nece-

sario interactuar con memoria, lo cual se realiza mediante la señal de petición MPET. El tipo de petición a memoria se indica mediante la señal MLES.

Transcurrido el tiempo de acceso a memoria se activa la señal MLISTO. Si es una operación de lectura se escribe el bloque en cache, se suministra el dato al procesador y se activa la señal LISTO. Si es una operación de escritura y ha sido acierto en cache se actualiza el campo de datos de cache al recibir la confirmación de memoria (MLISTO). Tanto si ha sido acierto o fallo en escritura el servicio ha finalizado y se activa la señal LISTO. Seguidamente se presenta la descripción en pseudo código.

```
while (true) {
    while (PET = NO) { };
    if (AF = acierto and LES = lectura) then
        LISTO = SI;
        muxL = contenedor;
    else
        MPET = peticion a memoria;
        MLES = LES;
        while (MLISTO = NO) { };
        if (LES = lectura) then
            muxE = memoria;
            muxL = memoria;
            PED = escribir campo datos;
            PEV = escribir estado;
            PEET = escribir etiquetas;
        else if (AF = acierto and LES = escritura) then
                muxE = procesador;
                PED = escribir campo datos;
        end if
    end if
}
```

Autómata del controlador de cache

En el autómata distinguimos cuatro estados, denominados DES, ESP1, ESP2 y ESP3 (Figura 14). En las transiciones que salen del estado DES las condiciones se muestran de forma textual y utilizando variables (entre paréntesis). Además, por claridad, en el diagram de estados no se ha especificado la señal PET en ninguna de las transiciones desde el estado DES. En las transiciones que finalizan en otro estado, la condición completa es la condición mostrada (X) AND la señal PET. En la transición que finaliza en el mismo estado, la condición completa es la condición mostrada OR la señal $\overline{PET}$. En la parte superior derecha de la Figura 14 se muestran de forma esquemática las condiciones completas de las transiciones que salen del estado DES.

En el estado DES se aceptan peticiones del procesador y el autómata permanece en este estado si es un acierto de lectura. En un fallo de lectura o una petición de escritura se pasa a uno de los otros tres estados, donde hay que esperar a que la memoria efectúe el servicio. Al estado ESP1 se llega en un fallo de lectura. En una operación de escritura la transición es del estado DES al estado ESP2 o ESP3 en función de si se determinna un acierto o un fallo.

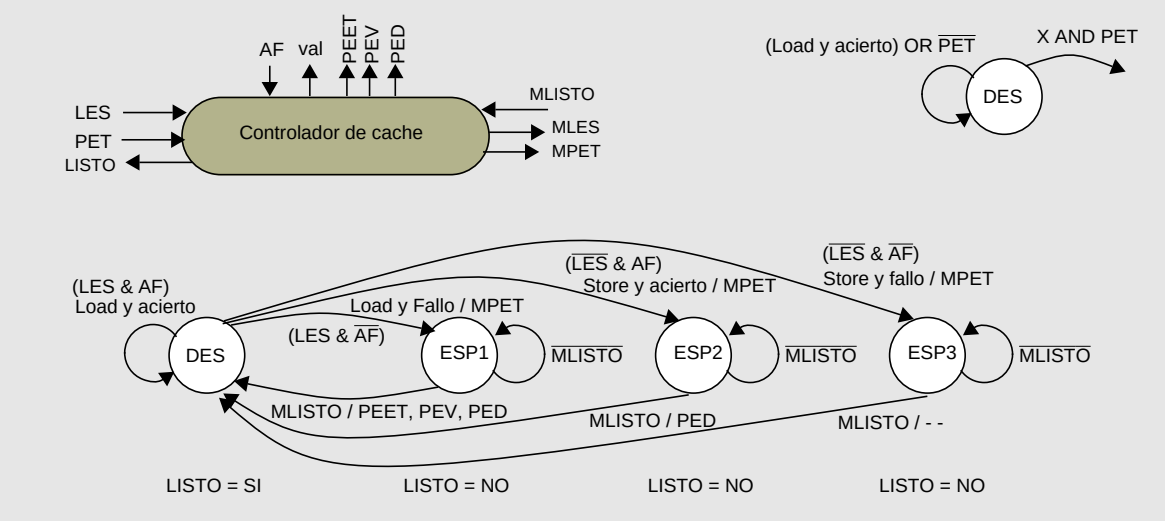


Figura 14 Autómata del controlador de cache.

Diseño lógico del controlador de cache

En las tablas de la Figura 15 se muestra para cada estado las señales de entrada que deben tenerse en cuenta para determinar las señales de salida. La existencia de una petición se codifica como PET =1. Una lectura se codifica como LES = 1. Un acierto se codifica como AF = 1. La finalización del servicio de memoria se codifica como MLISTO =1. El permiso de escritura se codifica como PEx = 1 donde x puede ser D, V o ET.

ENTREGA. *Clasifique las señales de salida en tipo Moore o Mealy.*

En todos los estados la señal MLES es igual a la señal LES. En el estado DES sólo hay que considerar las señales de salida LISTO y MPET, todas las otras señales deben estar desactivadas. En el estado ESP1 sólo hay que considerar las señales de salida LISTO, PED, PEV y PEET, las otras señales deben estar desactivadas (toma el valor cero). En el estado ESP2 sólo hay que considerar las señales de salida LISTO y PED, las otras señales deben de estar desactivadas. En el estado ESP3 sólo hay que considerar la señal de salida LISTO, las otras señales deben de estar desactivadas.

ESTADOS

DES

PET	AF	LES	LISTO	MPET
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

ESP1

MLISTO	LISTO	PED	PEV	PEET
0	0	0	0	0
1	1	1	1	1

ESP2

MLISTO	LISTO	PED
0	0	0
1	1	1

ESP3

MLISTO	LISTO
0	0
1	1

Figura 15 Tablas de verdad para el diseño de la lógica de salida.

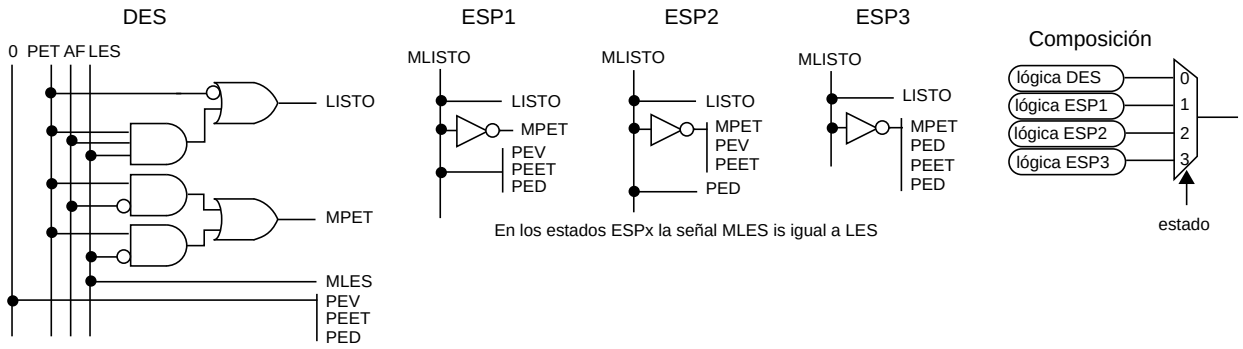


Figura 16 Izquierda: en función del estado, diseño con puertas de la lógica de salida. Derecha: Composición de diseños parciales en función del estado.

Para el diseño completo de la lógica de salida utilizaremos un multiplexor de cuatro entradas para cada señal (mostrado en la parte derecha de la Figura 16). Las entradas de un multiplexor se corresponden con la misma señal de cada uno de los diseños de la Figura 16. La salida del multiplexor se controla mediante la señal estado. Las tablas de la Figura 17 se utilizan para diseñar la lógica de próximo estado. En el estado DES hay que considerar las señales PET, AF y LES. En los estados ESP1, ESP2 y ESP3 sólo hay que considerar la señal MLISTO. Los estados se codifican de la siguiente forma: DES = 00, ESP1= 01, ESP2 = 10, ESP3 = 11. En la Figura 17 se muestra el diseño con puertas lógicas correspondiente a la tabla de la misma figura. El diseño se ha efectuado de la misma forma que se ha descrito para la lógica salida.

ESTADOS

DES

estado	PET	AF	LES	próximo
DES	0	0	0	DES
DES	0	0	1	DES
DES	0	1	0	DES
DES	0	1	1	DES
DES	1	0	0	ESP3
DES	1	0	1	ESP1
DES	1	1	0	ESP2
DES	1	1	1	DES

ESPx

estado	MLISTO	próximo
ESPx	0	ESPX
ESPx	1	DES

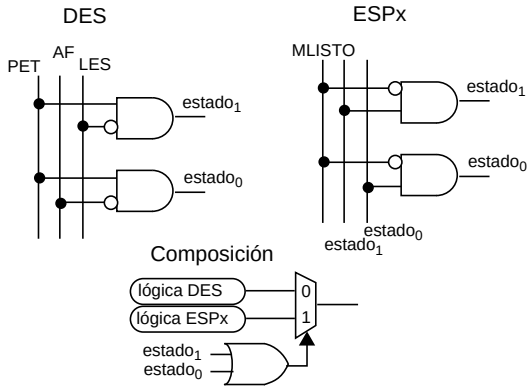


Figura 17 Izquierda: tablas de verdad para el diseño de la lógica de próximo estado. Derecha: diseño con puertas de la lógica de próximo estado en función del estado y su composición.

En la Figura 18 se muestra la relación entre la señal de reloj y los retardos de la lógica.

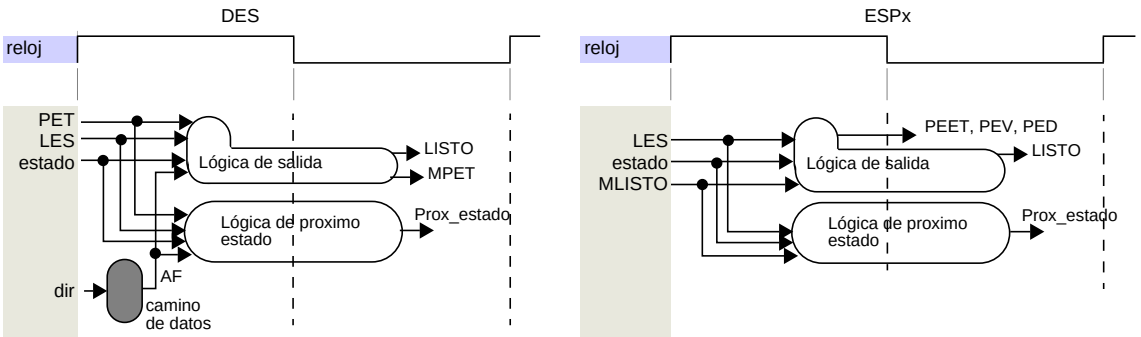


Figura 18 Relación funcional entre la señal de reloj y retardos de la lógica.

En las Figura 19 se muestran diagramas temporales para los cuatro casos de acceso a cache. En la parte izquierda de la Figura 19 se muestran accesos de lectura y en la parte derecha accesos de escritura. Los retardos que se observan en las señales y el número de ciclos en los estados ESPx no se corresponden con la implementación solicitada. Se utilizan sólo para visualizar los cambios de valor en las señales y para que el diagrama temporal ocupe pocos ciclos.

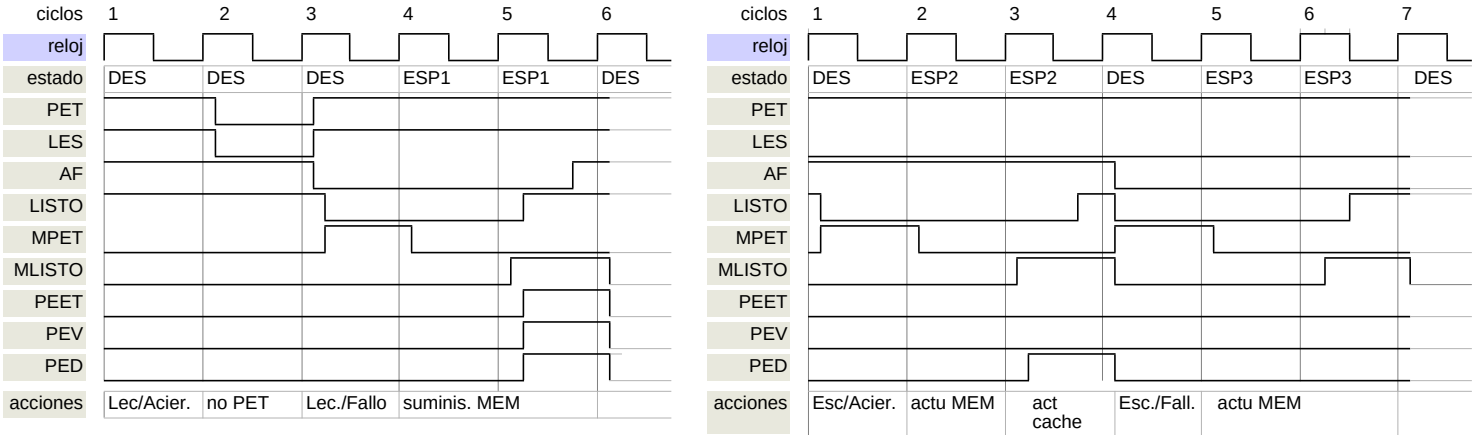


Figura 19 Diagramas temporales en los cuatro casos de acceso a cache.

Al diseñar el autómata del controlador de cache añada como salidas del autómata señales para identificar el estado en que se encuentra.

Generador de peticiones

En el apéndice 2 se muestra el esqueleto de un generador de peticiones para comprobar el controlador. Antes de iniciar un acceso se espera a que el procesador esté preparado. Después de iniciar el acceso y detectar una escritura o un fallo de lectura es necesario emular el funcionamiento de la memoria. Esta acción se efectúa esperando varios ciclos de reloj y activando la señal MLISTO. En el generador que se muestra en el apéndice 2 la memoria tarda un ciclo (Figura 19, dos flancos de la señal de reloj).

ENTREGA. *Diseñe una descripción VHDL del controlador de cache. En el apéndice 1 se muestra un esqueleto de la descripción del controlador de cache. La descripción debe constar de tres procesos: a) estado, b) lógica de próximo estado y c) lógica de salida. La actualización de los campos de cache se sincroniza con el nivel bajo de la señal de reloj. Esta sincronización debe efectuarse utilizando sentencias de asignación de señal fuera de los procesos.*

ENTREGA. *Compruebe el funcionamiento del controlador de cache utilizando un generador de peticiones. Para elo utilice el ejemplo de generador mostrado en el apéndice 2. Entregue la descripción VHDL del controlador de cache y un diagrama temporal donde se observe el funcionamiento.*

ENTREGA. *Extraiga de la descripción VHDL del controlador de cache la parte correspondiente al autómata que determina el estado de un bloque. Entregue esta descripción.*

Conexión del controlador de cache y el camino de datos

En el fichero cache.cct y la librería libuni.clf, accesibles en el Racó, se dispone respectivamente del camino de datos y de los elementos utilizados en su construcción. Los otros ficheros accesibles en el Racó contienen la descripción de los elementos en VHDL, la implementación mediante circuitos lógicos o una mezcla de ambas.

En el camino de datos se distinguen el módulo de campos y los multiplexores (Figura 20).

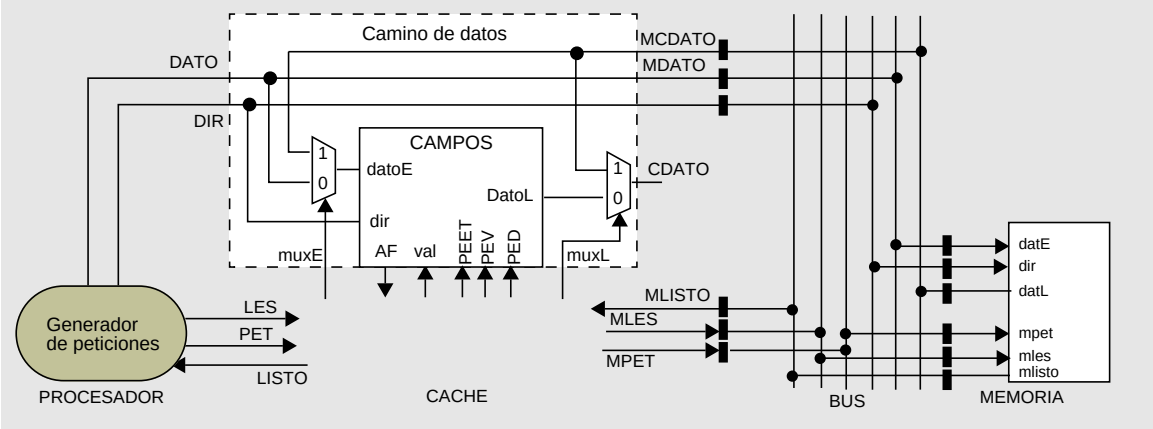


Figura 20 Esquema del camino de datos de la cache, acceso a memoria y generador de peticiones.

La cache tiene implementados 16 contenedores y se utiliza mapeo directo. La memoria sólo tiene implementadas 64 posiciones de almacenamiento. Esto es, los 8 bits más significativos de la dirección no se tienen en cuenta al acceder a memoria. Sin embargo, algunos de estos bits pertenecen al campo etiquetas de cache y por tanto, aunque se acceda a la misma dirección de memoria, en cache se consideran direcciones distintas.

El acceso a memoria está segmentado. Los rectángulos negros en la Figura 20 representan registros de desacoplo. En el ciclo posterior a la detección de una escritura o un fallo de lectura se transmite por el bus la información oportuna. La memoria está ocupada durante un ciclos. En el ciclo posterior se ocupa el bus y en el siguiente ciclo se actualiza la cache, suministrando el dato al procesador, si es el caso.

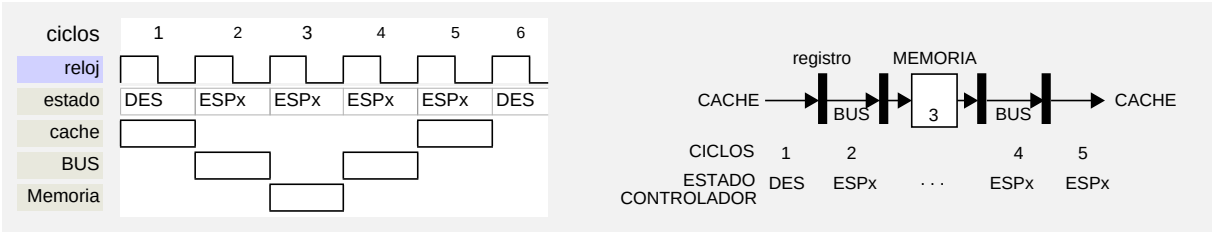


Figura 21 Ocupación del bus y memoria en una solicitud de acceso a memoria.

Cuando el controlador de cache solicita un acceso a memoria genera un pulso en la señal MPET. Cuando la memoria finaliza un servicio contesta con un pulso en la señal MLISTO (Figura 19).

Generador de peticiones

En el apéndice 3 se muestra un ejemplo de generador de peticiones de acceso a memoria. Antes de iniciar un acceso se espera a que el procesador esté preparado.

ENTREGA. *Incluya el controlador de cache que ha diseñado en el camino de datos. Compruebe el funcionamiento de la cache utilizando como base el ejemplo de generador de peticiones mostrado en el apéndice 3. Entregue un diagrama temporal donde se observe el funcionamiento.*

Apéndice 1: Esqueleto de una descripción en VHDL del controlador de cache

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

entity controlador is
    generic (retardo: time := 4 ns);
    port (reloj, pcero: in std_logic;                          -- Pcero señal de puesta a cero
        LES: in std_logic;                                     -- Interface con el procesador
        PET: in std_logic;
        LISTO: out std_logic;

        AF: in std_logic;                                      -- Interface con módulo campos
        PED: out std_logic;                                    -- y elementos periféricos
        PEET: out std_logic;
        PEV: out std_logic;
        val: out std_logic;
        muxL: out std_logic;
        muxE: out std_logic;

        MLES: out std_logic;                                   -- Interface con memoria
        MPET: out std_logic;
        MLISTO: in std_logic;

        VEST: out std_logic_vector (3 downto 0));              -- Observación externa del estado
end;

architecture comportamiento of controlador is
    constant UNO : std_logic := '1';                           -- Constantes para establecer
    constant CERO : std_logic := '0';                          -- y comprobar valores

    constant lectura : std_logic := '1';
    constant escritura : std_logic := '0';

    constant acierto : std_logic := '1';
    constant fallo : std_logic := '0';

    constant PETICION : std_logic := '1';
    constant NOPETICION : std_logic := '0';

                                                               -- observación externa del estado
    constant VESTDESO: std_logic_vector :="1000";              -- Estado DESO
    constant VESTESP1: std_logic_vector :="0100";              -- Estado ESP1
    constant VESTESP2: std_logic_vector :="0010";              -- Estado ESP2
    constant VESTESP3: std_logic_vector :="0001";              -- Estado ESP3

    type tipoestado is (DES, ESP1, ESP2, ESP3);                -- Estados
    signal estado, prxestado: tipoestado;                      -- Registro de estado

-- señales temporales en el proceso de salida, sin sincronizar con la señal de reloj,
-- correspondientes a las PED, PEV y PEET
    signal TPED: std_logic :='1';
    signal TPEV: std_logic :='1';
    signal TPEET: std_logic :='1';

-- deteccion de flanco ascendente
    function flanco_ascendente (signal reloj: std_logic) return boolean is
        variable flanco: boolean:= FALSE;
        begin
```

```
            flanco := (reloj = '1' and reloj'event);
            return (flanco);
        end flanco_ascendente;

    begin

    -- registro de estado
        process (reloj, pcero)
            begin
                if pcero = UNO then
                    . . .
                    VEST <= VESTDESO;
                elsif (flanco_ascendente(reloj)) then
                    . . .
                    case prxestado is
                        when DES => VEST <= VESTDESO after retardo;
                        when ESP1 => VEST <= VESTESP1 after retardo;
                        when ESP2 => VEST <= VESTESP2 after retardo;
                        when ESP3 => VEST <= VESTESP3 after retardo;
                    end case;
                end if;
        end process;

    -- logica de proximo estado
        process(estado, LES, PET, AF, MLISTO, pcero)
            begin
                . . .
                if (pcero = CERO then
                    case estado is
                        when DESO =>
                            . . .
                        when ESP1 | ESP2 | ESP3 =>
                            . . .
                    end case;
                else
                    prxestado <= DESO after retardo;
                end if;
        end process;

    -- logica de salida
        process(estado, LES, PET, AF, MLISTO, pcero)
            begin
                val <= UNO after retardo;
                . . .
                if (pcero = CERO) then
                    case estado is
                        when DESO =>
                            . . .
                        when ESP1 =>
                            . . .
                        when ESP2 =>
                            . . .
                        when ESP3 =>
                            . . .
                        when others =>
                            . . .
                    end case;
                else
                    . . .
        end process;

    -- actualizacion de cache
```

```
PED <= not (TPED and (not (reloj) );
PEET <=. . .
PEV <=. . .

end;
```

Apéndice 2: Esqueleto de una descripción en VHDL de un generador de peticiones para comprobar el funcionamiento del controlador

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

    entity genpruecontpet is
        generic (retardo: time := 2 ns);
        port ( pcero: out std_logic;
        LES: out std_logic;                                  -- Interface con el controlador de cache
        PET: out std_logic;
        LISTO: in std_logic;

        AF: out std_logic;                                   -- respuesta de módulo campos
        MLISTO: out std_logic;                               -- respuesta de memoria

        reloj: in std_logic);
    end;

    architecture estimulos of genpruecontpet is
        constant UNO : std_logic := '1';
        constant CERO : std_logic := '0';

        constant lectura: std_logic := '1';
        constant escritura : std_logic := '0';

        constant PETICION : std_logic := '1';
        constant NOPETICION : std_logic := '0';

        constant ACIERTO : std_logic := '1';
        constant FALLO : std_logic := '0';

        function flanco_ascendente (signal reloj: std_logic) return boolean is
            variable flanco: boolean:= FALSE;
            begin
                flanco := (reloj = '1' and reloj'event);
                return (flanco);
        end flanco_ascendente;

    begin
    process
        begin
            pcero <= UNO;                                    -- puesta a cero
            LES <= lectura;
            PET <= NOPETICION;                               -- no hay peticion

            AF <= ACIERTO;
            MLISTO <= CERO;

            wait until flanco_ascendente(reloj);             -- esperar 1 ciclo

            pcero <= CERO after retardo;                     -- final de puesta a cero

            wait until flanco_ascendente(reloj);             -- esperar 2 ciclo
            wait until flanco_ascendente(reloj) and LISTO = UNO;  -- esperar a que la cache este preparada

            LES <= escritura after retardo;                  -- escritura
            PET <= PETICION after retardo;

            AF <= FALLO after retardo;                       -- fallo de escritura
```

```
            wait until flanco_ascendente(reloj);                    -- esperar a memoria
            wait until flanco_ascendente(reloj);
            MLISTO <= UNO after retardo;                            -- finaliza acceso a memoria

            wait until flanco_ascendente(reloj) and LISTO = UNO;    -- esperar a que la cache este preparada
            MLISTO <= CERO after retardo;

            LES <= lectura after retardo;                           -- lectura
            PET<= PETICION after retardo;

            AF <= FALLO after retardo;                              -- fallo de lectura

            wait until flanco_ascendente(reloj);                    -- esperar a memoria
            wait until flanco_ascendente(reloj);
            MLISTO <= UNO after retardo;                            -- finaliza acceso a memoria

            wait until flanco_ascendente(reloj) and LISTO = UNO;    -- esperar a que la cache este libre
            MLISTO <= CERO after retardo;

            LES <= escritura after retardo;                         -- lectura
            PET <= PETICION after retardo;

            AF <= ACIERTO after retardo;                            -- acierto de lectura

            wait until flanco_ascendente(reloj);                    -- esperar a memoria
            wait until flanco_ascendente(reloj);
            MLISTO <= UNO after retardo;                            -- finaliza acceso a memoria

            wait until flanco_ascendente(reloj) and LISTO = UNO;    -- esperar a que la cache este libre
            MLISTO <= CERO after retardo;
            . . .
    end process;
end;
```

Apéndice 3: Esqueleto de una descripción en VHDL de un generador de peticiones para comprobar el funcionamiento de la cache

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

    entity pruebacontrolador is
        generic (retardo: time := 2 ns);
        port ( pcero: out std_logic;
        LES: out std_logic;                                  -- Interface con el controlador de cache
        PET: out std_logic;
        LISTO: in std_logic;
        DIR: out std_logic_vector (15 downto 0);             -- Interface con módulo campos
        DATO: out std_logic_vector (7 downto 0);
        DATOL: out std_logic_vector (7 downto 0);            -- puede utilizarse para comprobar el dato leido

        reloj: in std_logic);
    end;

    architecture estimulos of pruebacontrolador is
        constant UNO : std_logic := '1';
        constant CERO : std_logic := '0';

        constant lectura: std_logic := '1';
        constant escritura : std_logic := '0';

        constant PETICION : std_logic := '1';
        constant NOPETICION : std_logic := '0';

        constant DIRA: std_logic_vector := x"0000";
        constant DATA: std_logic_vector := x"AA";
        constant DIRB: std_logic_vector := x"0010";
        constant DATB: std_logic_vector := x"BB";

        constant periodo : time := 10 ns;

        function flanco_ascendente (signal reloj: std_logic) return boolean is
            variable flanco: boolean:= FALSE;
            begin
                flanco := (reloj = '1' and reloj'event);
                return (flanco);
        end flanco_ascendente;

    begin
    process
        begin
            pcero <= UNO after retardo;                      -- puesta a cero
            LES <= lectura after retardo;
            PET <= NOPETICION after retardo;
            DIR <= x"FFFF";
            DATO <=X"FF";

            wait until flanco_ascendente(reloj);             -- esperar 2 ciclos
            wait until flanco_ascendente(reloj);
            wait until flanco_ascendente(reloj) and LISTO = UNO;   -- esperar a que la cache este libre

            pcero <= CERO after retardo;                     -- finaliza la puesta a cero
            LES <= escritura after retardo;                  -- escritura en direccion DIRA del dato DATA
            PET <= PETICION after retardo;                   -- fallo de escritura
```

```
        DIR <= DIRA after retardo;
        DATO <= DATA after retardo;

        wait until flanco_ascendente(reloj) and LISTO = UNO;    -- esperar a que la cache este libre
        LES <= lectura after retardo;                           -- lectura en direccion DIRA
        PET<= PETICION after retardo;                           -- fallo de lectura
        DIR <= DIRA after retardo;

        wait until flanco_ascendente(reloj) and LISTO = UNO;    -- esperar a que la cache este libre
        LES <= escritura after retardo;                         -- escritura en direccion DIRA del dato DATB
        PET <= PETICION after retardo;                          -- acierto de escritura
        DIR <= DIRA after retardo;
        DATO <= DATB after retardo;

        wait until flanco_ascendente(reloj) and LISTO = UNO;    -- esperar a que la cache este libre
        LES <= lectura after retardo;                           -- lectura en direccion DIRA
        PET <= PETICION after retardo;                          -- acierto de lectura
        DIR <= DIRA after retardo;

        wait until flanco_ascendente(reloj) and LISTO = UNO;    -- esperar a que la cache este libre
        DIR <= x"FFFF" after retardo;
        PET <= NOPETICION after retardo;
        LES <= lectura after retardo;

        wait until flanco_ascendente(reloj) and LISTO = UNO;    -- esperar a que la cache este libre
        . . .
    end process;
end;
```