PRACTICA: 4

COHERENCIA DE CACHE EN UN MULTIPROCESADOR IDEALIZADO

En esta práctica se modela un multiprocesador con cache privadas y un bus como red de interconexión, siendo el objetivo implementar un protocolo de coherencia. El modelado es ideal ya que sólo puede haber un acceso a memoria por ciclo. El objetivo de esta simplificación es centrarse en las transiciones entre estados, denominados estables, de un bloque almacenado en cache.

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 1). Sólo los bloque almacenados en cache tienen un hardware que representa su estado.

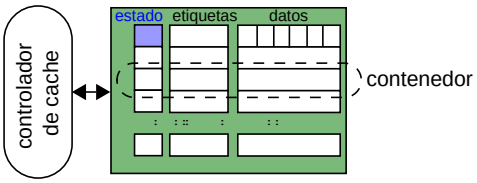


Figura 1 Memoria cache y campos en un contenedor de cache.

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. También actualiza el estado de los bloques en respuesta a transacciones del bus, si es el caso.

Organización del multiprocesador

La cache privada de un procesador es bloqueante, con escritura inmediata y sin asignación de contenedor en el caso de fallo en escritura. En la Figura 2 se muestra la organización del multiprocesador y las interfaces con memoria y con el generador de direcciones. En la parte superior derecha de la misma figura se muestra el componente etiquetado como regT. Este componente incluye un registro y un buffer de tres estados. Cuando una cache accede a memoria el buffer de tres estados se comporta como un cortocircuito. En caso contrario los cables quedan desconectados del bus (alta impedancia). La señal de control del buffer de tres estados proviene del generador de peticiones. Esta señal no se muestra en la Figura 2. La funcionalidad de las señales se describe en la tabla de la Figura 3.

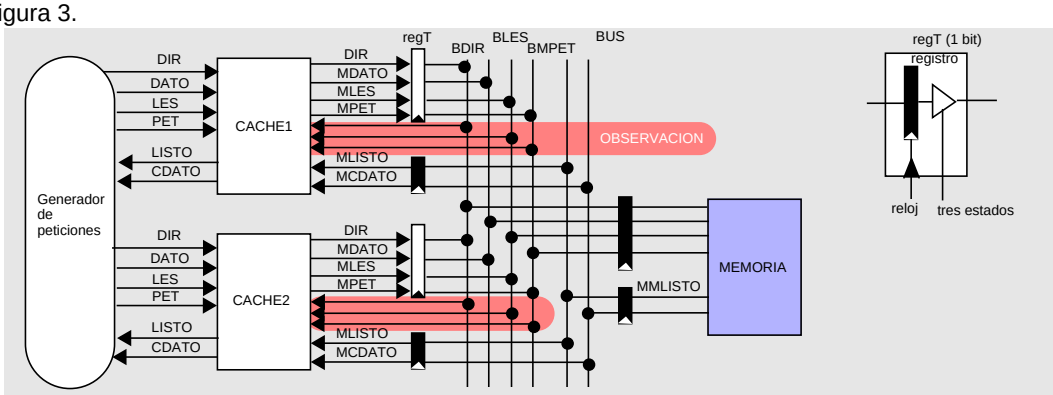


Figura 2 Interfaces entre generador de peticiones, caches privadas y memoria.

Las interfaces procesador/cache y cache/memoria relativas al servicio de una petición de un procesador son idénticas al caso uniprocesador. Las señales BDIR, BLES y BMPE, de la interface cache/bus, son utilizadas por una cache privada para observar la transaccción que ocupa el bus.

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

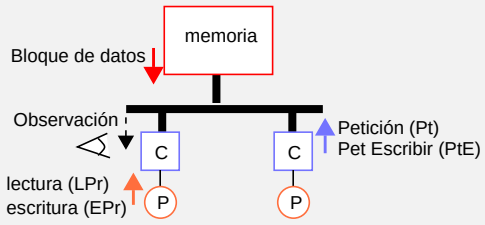
interface cache / bus (observación)

	Descripción de las señales
BDIR	dirección en un acceso a memoria
BMLES	indicación de lectura o escritura
BMPET	transacción en el bus

Figura 3 Descripción de las señales de las interfaces entre procesador, cache, bus y 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 proc.**	**Acciones cont.**	**Acciones Obs. bus**
LPr : lectura del procesador	Pt : petición de bloque	CcRe: reemplazo de un bloque	Escritura y acierto: Invalidar bloque
EPr : escritura del procesador	PtE : actualización de memoria		

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

Diagrama de transiciones y estados de un bloque en cache

En la Figura 5 se muestra, para un uniprocesador, el diagrama de transiciones entre estados de un bloque en cache. 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.

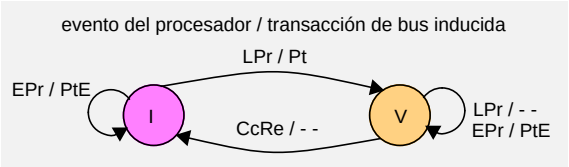


Figura 5 Eventos del procesador y acciones del controlador de cache.

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.

En el multiprocesador descrito hay que observar las transacciones de bus para actualizar el estado de los bloques almacenados en cache, si es el caso. Las transiciones entre estados inducidas por transacciones de bus se muestran en la Figura 6. Para atender la observación de una escritura, sólo es necesario añadir una transición entre los estados de un bloque en cache. Un bloque en estado V pasa al estado I cuando se observa en el bus una petición de escritura a un elemento del bloque. La observación de la misma transacción de bus en el estado I u otras transacciones de bus no modifica el estado del bloque en cache.

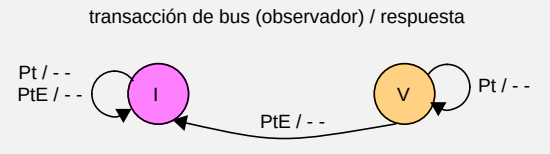


Figura 6 Eventos del procesador y acciones del controlador de cache.

Para mantener la coherencia, las caches privadas incluyen la funcionalidad de observación de las transacciones que se transmiten por el bus. Los recursos necesarios para efectuar esta operación están disponibles en la cache: campo etiquetas de la cache y comparador.

Como se modela un multiprocesador ideal, en un ciclo determinado sólo existe un acceso a memoria. Por tanto, no existe competencia para acceder a estos recursos entre la parte del controlador de cache que atiende las peticiones del procesador y la parte del controlador de cache que atiende las peticiones de observación. En el procesador que efectúa la petición son de interés las señales de salida de la parte del controlador de cache que atiende las peticiones del procesador. En los otros procesadores, que están observando el bus, son de interés las señales de salida de la parte del controlador de cache que atiende las operaciones de observación.

Transacción de bus y controlador de coherencia

La red de interconexión utilizada en el multiprocesador es un bus y el aspecto de una transacción de bus se muestra en la Figura 7, siendo los ciclos sólo indicativos. En este diseño simplificado, la operación de observación se efectúa en el mismo ciclo que se transmite la petición por el bus.

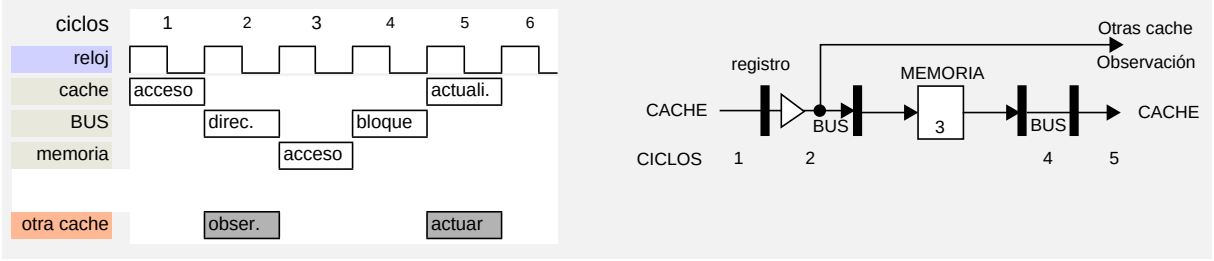


Figura 7 Esquema de una transacción de bus.

La actuación a que de lugar una observación (actuar) se efectúa en el mismo ciclo en el cual una petición de lectura actualiza la cache con el bloque de datos (actuali.).

Camino de datos

En la Figura 8 se muestra el camino de datos de la cache. Este camino de datos es idéntico al caso uniprocesador con la excepción del camino necesario para efectuar la operación de observación. En la misma figura se muestran, con trazo más grueso, las conexiones necesarias en el camino de datos para efectuar la operación de observación. En concreto, se añade el multiplexor muxO en la entrada de direcciones de la cache. Una de las entradas de este multiplexor proviene del procesador y la otra entrada proviene de los cables que transportan la dirección en el bus.

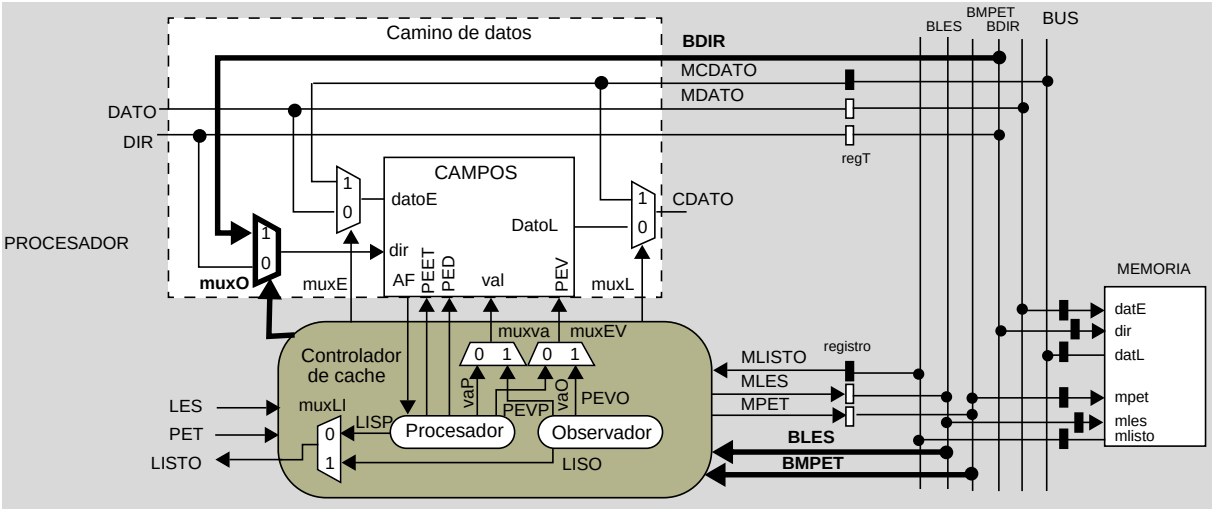


Figura 8 Camino de datos de un procesador del multiprocesador.

Controlador de cache

En el controlador de cache distinguimos dos autómatas que cooperan (Figura 8). Uno de ellos atiende las peticiones del procesador y el otro autómata observa las transacciones de bus. A estos autómatas los denominaremos, respectivamente, autómata del procesador y autómata del observador.

Como se esta modelando un multiprocesador idealizado, no es necesaria una comunicación explícita entre los dos autómatas. En un ciclo determinado se tienen en cuenta sólo las señales de uno de los autómatas. La selección se efectúa mediante los multiplexores muxva, muxEV y muxLI. Estos multiplexores se controlan teniendo en cuenta si la cache está efectuando una petición o está observando el bus.

El autómata del procesador es el mismo que se ha diseñado para un sistema uniprocesador. Las señales de salida que se denominaban PEV, val y LISTO ahora se denominan PEVP, vaP y LISP (Figura 8).

Las señales de entrada del autómata de observación son: petición en el bus (BMPET), tipo de transacción (BMLES) y acierto o fallo en cache (AF). En una operación de observación, la señal de interes en el camino de datos es BDIR (Figura 8). Por tanto, en el multiplexor muxO se selecciona como salida la entrada uno.

ENTREGA. *Teniendo en cuenta la descripción del controlador del camino de datos de la cache, clasifique las señales BMPET, BMLES y AF como señales de control o señales de estado.*

Los dos autómatas actualizan las señales PEV, val y LISTO. Para seleccionar una de las dos posibles señales, generadas por cada autómata, se utilizan respectivamente los multiplexores muxEV, muxva y muxLI de la Figura 8.

Descripción textual del autómata de observación

Cuando la señal BMPET está activada y la transacción es una escritura (BMLES), el controlador de cache utiliza la señal AF para determinar si se ha producido acierto o fallo al comparar la dirección transportada por los cables BDIR y el campo etiqueta de un contenedor de cache (mapeo directo). Cuando se produce acierto el controlador debe invalidar el bloque de cache (ciclo 5 en la Figura 7). Esta acción la efectúa el autómata de observación después de que se active la señal MLISTO (finalización de la transacción).

```
while (true) {
    while (BMPET = NO) { };
    if (BMLES = escritura and AF = acierto) then
        while (MLISTO = NO) { };
            valO = invalidación;
            PEVO = actualización el estado;
     end if
}
```

Autómata de observación (observador de bus)

El autómata que se encarga de observar las transacciones de bus tiene dos estados: Desocupado (DESO) y Espera (ESPO). En el estado DESO se está preparado para atender una observación y en el estado ESPO se está atendiendo una observación. Las señales de salida del autómata de observación son PEVO, vaO y LISO. Estas señales son respectivamente, el permiso de escritura en el campo estado, el valor que se escribe en el campo estado y la indicación de que el autómata está preparado para atender otra observación. En la Figura 9 se muestra el diagrama de transiciones entre estados.

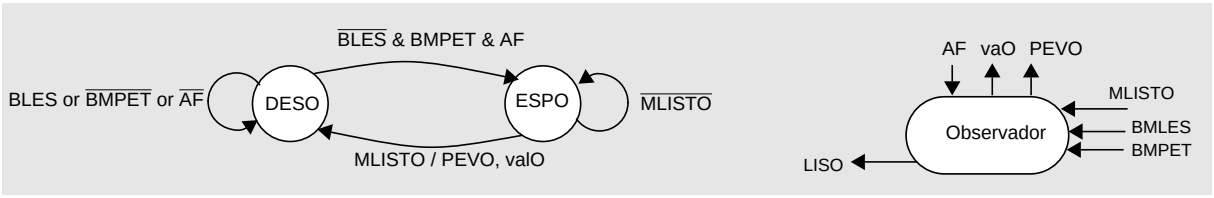


Figura 9 Autómata del controlador de cache.

Cuando existe una transacción de bus (BMPET = 1), es una petición de escritura (BLES = 0) y la cache tiene copia del bloque (AF = 1), se pasa del estado DESO al estado ESPO. En caso contrario el autómata permanece en el estado DESO. El autómata permanece en el estado ESPO hasta que finaliza la transacción de bus. La señal LISO está activada en el estado DESO.

ENTREGA. *Construya la tabla de transición entre estados del autómata de observación y diseñe el esquema de circuito con puertas y registros. Entregue la tabla que describe las transiciones entre estado. También, entregue las tablas de verdad de la lógica de próximo estado y de la lógica de salida. Así mismo, entregue el esquema de circuito con puertas y registros.*

Transacción de bus y controlador de coherencia

La red de interconexión utilizada en el multiprocesador es un bus y el aspecto de una transacción de bus y los estados de los autómatas del procesador y de observación se muestran en la Figura 10, siendo los ciclos sólo indicativos.

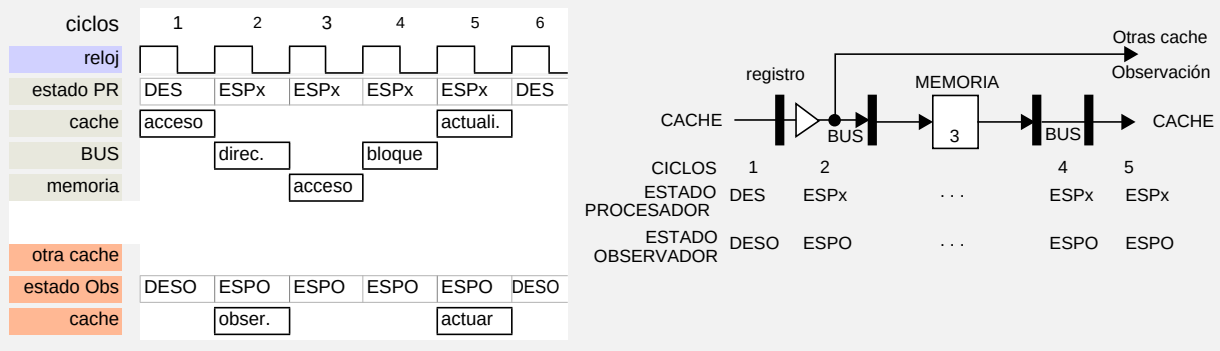


Figura 10 Transacción de bus y estados en el autómata del procesador y el autómata de observación.

En al Figura 11 se muestra como ejemplo un acceso de escritura, con acierto en cache y acierto de observación en otra cache. En la última fila se indican las ciclos en que se detecta el tipo de acceso, el acceso a memoria y la actualización de la cache. En la primera columna de la izquierda se identifica el elemento al que está asociada la señal que se muestra. En la primera fila, identificada como “cnt. cache”, se muestra el estado del controlador de cache que efectúa la petición (autómata del procesador) y está accediendo a memoria. En la fila que tiene como identificador “cont. Observador” se muestra el estado del observador de un procesador que no está accediendo a memoria. En la Figura 11 se utilizan líneas con flecha para mostrar algunas precedencias entre las señales. Entre los distintos grupos de señales, para facilitar la comprensión, se ha replicado el diagrama temporal de la Figura 7 que muestra una transacción de bus.

En el primer ciclo se detecta acierto en la cache y las señales de interés son las que tienen a su izquierda, la primera aparición de arriba hacia abajo, del identificador cache (cache (P0)). En el segundo ciclo las señales de interés son las identificadas por BUS (BUS petición) y las señales identificadas por “otra cache (P1)”. Observe que en la otra cache se detecta un acierto de observación.

En el tercer ciclo se muestra el acceso a memoria. En el cuarto ciclo las señales de interés son las identificas por BUS (BUS respuesta). En el quinto ciclo se actualiza la cache (segunda aparición del identificador “cache (P0)”) y la cache que ha detectado acierto en la observación invalida el bloque (identificador “otra cache (P1)”). En el siguiente ciclo, la cache que ha efectuado la petición está lista para aceptar una nueva petición (señal LISTO, en la segunda aparición del identificador “cache (P0)”).

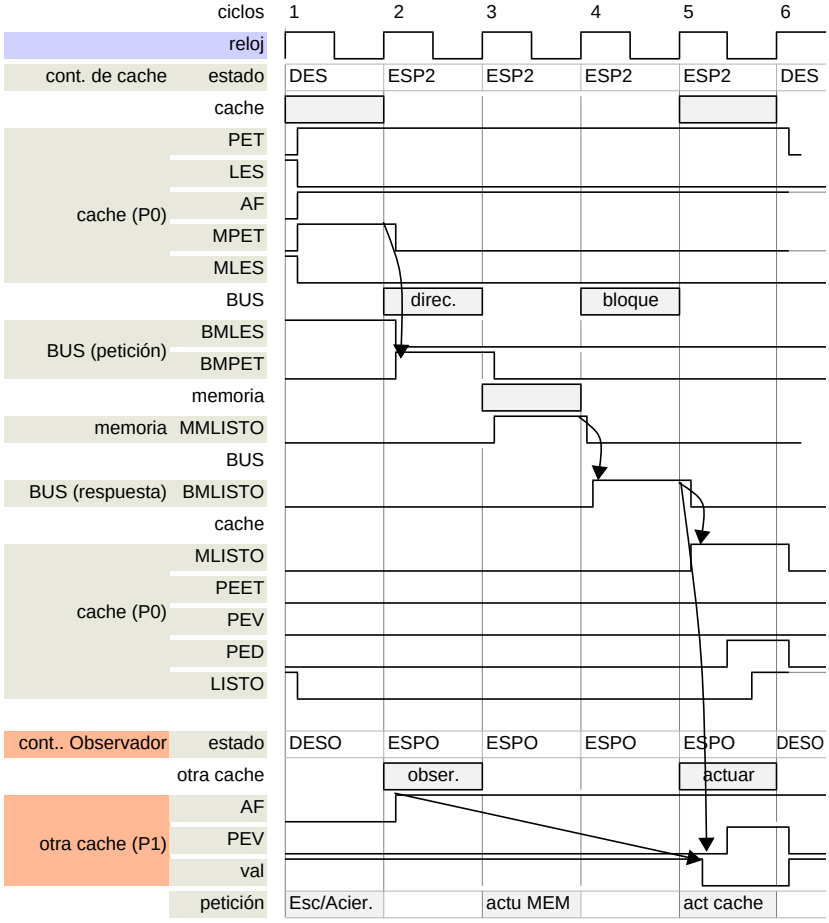


Figura 11 Acceso de escritura, con acierto en cache y acierto de observación en otra cache. Diagrama de algunas señales.

Integración del autómata del procesador y el autómata de observación

Una vez se dispone del autómata de observación hay que incluirlo en la descripción VHDL del controlador de cache.

Partimos del controlador de cache para un único procesador. En primer lugar hay que modificar la interface del controlador de cache incluyendo las señales BMPET y BLES. Así mismo, hay que incluir la descripción de un multiplexor para seleccionar entre las salidas PEVP y PEVO del autómata del procesador y del autómata de observación, con el objetivo de disponer de la señal PEV (muxEV). De forma similar hay que actuar para obtener las señales val y LISTO a partir de las señales vaP y vaO y LISP y LISO, que son salidas del autómata del procesador y del autómata de observación respectivamente (muxva, muxLI). Note que debe modificar el nombre de las señales PEV y LISTO en el autómata que ha utilizado en el caso uniprocesador para atender las peticiones del procesador. También hay que añadir la señal muxO para seleccionar la entrada del multiplexor muxO en el camino de datos.

La selección de la entrada en los multiplexores (muEV, muxva y muxLI) es función de si el controlador de cache está sirviendo una petición del procesador o está efectuando una observación. Como modelamos un multiprocesador idealizado, la señal de selección de estos multiplexores es la señal de entrada petición (PET) negada, ya que hay garantía de que sólo existe una petición de memoria en curso.

Interface. En la interface de la descripción VHDL se distinguen tres partes, a las que también denominamos interfaces. Las dos primeras interfaces: a) procesador y b) memoria son para servir los accesos a memoria. La tercera interface es el bus y se utiliza para efectuar la operación de observación.

También se definen dos señales de salidas que se utilizan para conocer el estado de los dos autómatas: a) procesador (VEST) y b) observador (VESTO).

```
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;
        muxO: out std_logic;

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

        BLES: in std_logic;                              -- Interface con el bus para observar
        BMPET: in std_logic;

                                                         -- Observación externa del estado
        VEST: out std_logic_vector (3 downto 0);         -- procesador
        VESTO: out std_logic_vector (1 downto 0));       -- observador
end;
```

En el apéndice 1 se muestra una descripción VHDL del controlador de cache. La descripción del autómata del procesador se ha omitido ya que se utiliza la descripción del controlador de cache uniprocesador. Sólo se muestra la definición de los tres procesos y las señales de activación de los mismos.

La descripción del autómata de observación se efectúa mediante tres procesos: a) estado, b) lógica de próximo estado y c) lógica de salida.

Después de la descripción del autómata de observación, se describen los multiplexores muxva, muxEV y muxL utilizando sentencias de asignación de señal condicional. Las siguientes sentencias de asignación de señal se utilizan para sincronizar la actualización de los campos de la cache en el nivel bajo de la señal de reloj.

ENTREGA. *Modifique el controlador de cache diseñado en la práctica anterior añadiendo el autómata de observación y teniendo en cuenta las indicaciones previas.*

ENTREGA. *Compruebe el funcionamiento del controlador de cache utilizando un generador de peticiones. Para ello 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.*

Multiprocesador idealizado

En la Figura 12 se muestra un multiprocesador idealizado con dos procesadores interconectados por un bus.

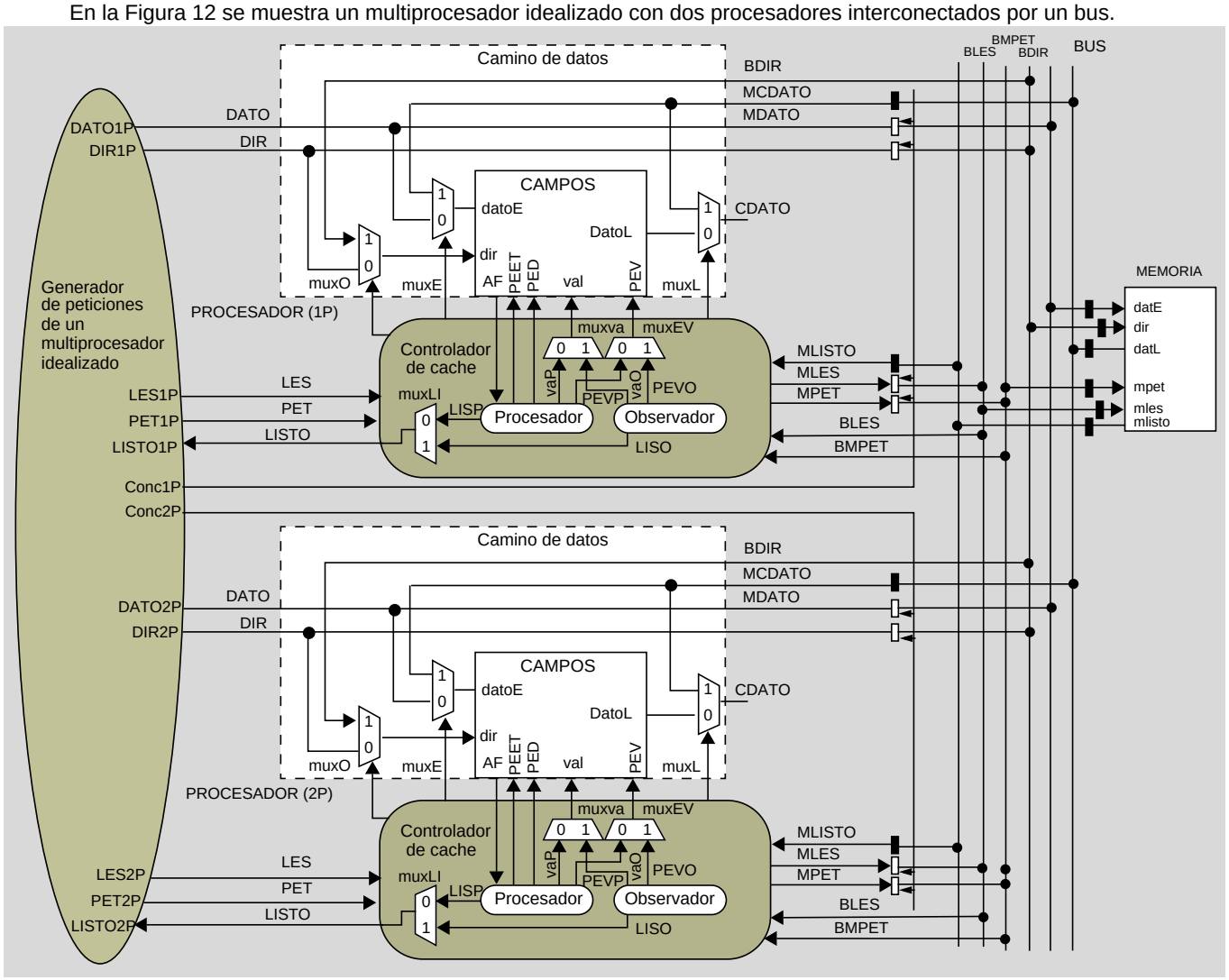


Figura 12 Multiprocesador con dos procesadores que utiliza como red de interconexión un bus.

El multiprocesador que se modela es ideal en el sentido de que, en un ciclo determinado, sólo hay un acceso a memoria en curso. Esta idealización se modela utilizando un único generador de peticiones para todas las caches privadas del multiprocesador.
El generador de direcciones es el encargado de idealizar el funcionamiento. Genera peticiones de acceso sólo a un procesador y concede acceso al bus (ConcxP, donde x es el número de procesador) al mismo procesador. El otro procesador no recibe petición de acceso y por tanto, está en modo de observación. Esto es, observa la transacción que transportada el bus.

Generador de peticiones de un multiprocesador idealizado

El generador de peticiones, en un ciclo determinado, genera una petición a una cache (PET). En este mismo ciclo la señal de petición de las otras caches está desactivada y deben actuar en modo observación.
La señal PET también se utiliza para controlar los buffer de tres estados de los cables que alimentan al bus con señales. Por tanto, al bus sólo está alimentado por las señales de la cache que efectúa una petición. Las otras caches observan la transacción que se transmite por el bus (ConcxP = not PET).
La interface del generador de direcciones para el caso de dos procesadores es la siguiente.

```
entity pruebacontroladormulti is
    generic (retardo: time := 4 ns);
    port ( pcero: out std_logic;
    LES1P: out std_logic;                          -- Interface con el controlador de cache 1P
    PET1P: out std_logic;
    LISTO1P: in std_logic;
    DIR1P: out std_logic_vector (15 downto 0);     -- Interface con módulo campos de cache 1P
    DATO1P: out std_logic_vector (7 downto 0);
    Conc1P: out std_logic;                         -- Interface con el bus

    LES2P: out std_logic;                          -- Interface con el controlador de cache 2P
    PET2P: out std_logic;
    LISTO2P: in std_logic;
    DIR2P: out std_logic_vector (15 downto 0);     -- Interface con módulo campos de cache 2P
    DATO2P: out std_logic_vector (7 downto 0);
    Conc2P: out std_logic;                         -- Interface con el bus

    reloj: in std_logic);
end;
```

En el apéndice 3 se muestra un ejemplo de generación de accesos a memoria para el multiprocesador idealizado. Antes de iniciar un acceso se espera a que todos los procesadores estén preparados.

Conexión del controlador de cache y el camino de datos

En el fichero CACHE.cct y la librería libmultiIDEAL.clf, accesible desde el Raco, se dispone respectivamente del camino de datos y de los elementos utilizados en su construcción. En el camino de datos se distinguen los elementos mostrados en la Figura 12.
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 como se muestra en la Figura 10.
Cuando un controlador de cache solicita un acceso a memoria debe generar un pulso en la señal MPET. Cuando ha finalizado el servicio solicitado, la memoria contesta con un pulso en la señal MLISTO.

ENTREGA. *Incluya el controlador de cache en el camino de datos. Compruebe el funcionamiento de la cache. Entregue un diagrama temporal donde se observe el funcionamiento.*

Apéndice 1: Esqueleto en VHDL del controlador de cache. Especificación del autómata de observación

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

library LogicWorks;
use LogicWorks.debug.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;
        muxO: out std_logic;

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

        BMLES: in std_logic;                             -- Interface con el bus para observar
        BMPET: in std_logic;

                                                         -- Observación externa del estado
        VEST: out std_logic_vector (3 downto 0);         -- procesador
        VESTO: out std_logic_vector (1 downto 0));       -- observador
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';

-- PROCESADOR
                                                         -- Observación externa del estado del proc.
    constant VESTDESO: std_logic_vector :=x'1000';       -- Estado DESO
    constant VESTESP1: std_logic_vector :=x'0100';       -- Estado ESP1
    constant VESTESP2: std_logic_vector :=x'0010';       -- Estado ESP2
    constant VESTESP3: std_logic_vector :=x'0001';       -- Estado ESP3

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

-- señales temporales en el proceso de salida del procesador, sin sincronizar con la señal de reloj,
```

```
-- correspondientes a las PED, PEV y PEET
    signal TPED: std_logic :='1';
    signal PEVP: std_logic :='1';                          -- en uniprocesador se llamaba TPEV
    signal TPEET: std_logic :='1';

-- procesador: señal para establecer el estado (listo y val)
    signal LISP: std_logic :='1';
    signal vaP: std_logic :='1';

-- OBSERVADOR
                                                           -- Observación externa del estado del Obs.
    constant VESTDESOB: std_logic_vector :="10";           -- Estado DESO
    constant VESTESPOB: std_logic_vector :="01";           -- Estado ESPO

    type tipoestadoO is (DESO, ESPO);                      -- Estados del automata del observador
    signal estado_O, prxestado_O: tipoestadoO;             -- Registro de estado del observador

-- señales temporales en el proceso de salida del observador, sin sincronizar con la señal de reloj,
-- correspondiente a PEVO
    signal PEVO: std_logic :='1';

-- observador: señal para establecer el estado (listo y val)
    signal LISO: std_logic :='1';
    signal vaO: std_logic :='1';

-- multiplexores. Seleccion entre la salida del procesador y la salida del observador
    signal muxva: std_logic;                               -- seleccion de la señal val
    signal muxEV: std_logic;                               -- seleccion de la señal PEV
    signal muxLI: std_logic;                               -- selec. de la señal LISTO del proc. u obs.
    signal TPEV: std_logic;                                -- señal de salida del multiplexor muxEV

-- 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

-- PROCESADOR
-- registro de estado del procesador
est_proc: process (reloj, pcero)
        begin
            . . .
    end process est_proc;

-- logica de proximo estado del procesador
prox_proc: process(estado, LES, PET, AF, MLISTO, pcero)
        begin
            . . .
    end process prox_proc;

-- logica de salida del procesador
sal_proc: process(estado, LES, PET, AF, MLISTO, pcero)
        begin                                              -- la señal LISTO debe denominarse LISP
            vaP <= CERO;                                   -- hay que definidr la señal vaP
            . . .                                          -- la señal TPEV debe denominarse PEVP
    end process sal_proc;
```

```
-- OBSERVADOR
-- registro de estado del observador
est_obs: process (reloj, pcero)
        begin
            if (pcero = UNO) then
                estado_O <= DESO after retardo;
                VESTO <= VESTDESOB after retardo;
            elsif (flanco_ascendente(reloj)) then
                estado_O <= prxestado_O after retardo;
                case prxestado_O is
                    when DESO => VESTO <= VESTDESOB after retardo;
                    when ESPO => VESTO <= VESTESPOB after retardo;
                end case;
            end if;
    end process est_obs;

-- logica de proximo estado del observador
prox_obs: process(estado_O, BMLES, BMPET, AF, MLISTO)
        begin
                case estado_O is
                    when DESO =>
                        if (BMPET = PETICION and BMLES = escritura and AF = acierto) then
                            prxestado_O <= ESPO after retardo;
                        else
                            prxestado_O <= DESO after retardo;
                        end if;
                    when ESPO =>
                        if MLISTO = UNO then
                            prxestado_O <= DESO after retardo;
                        else
                            prxestado_O <= ESPO after retardo;
                        end if;
                    when others => prxestado_O <= DESO after retardo;
                end case;
    end process prox_obs;

-- logica de salida del observador
sal_proc: process(estado_O, BMLES, BMPET, AF, MLISTO)
        begin
            PEVO <= NO ;
            vaO <= CERO ;
                case estado_O is
                    when DESO =>
                        if (BMPET = PETICION and BMLES = escritura and AF = acierto) then
                            LISO <= CERO after retardo;
                        else
                            LISO <= UNO after retardo;
                        end if;
                    when ESPO =>
                        if MLISTO = UNO then
                            LISO <= UNO after retardo;
                            vaO <= UNO after retardo;
                            PEVO <= SI after retardo;
                        else
                            LISO <= CERO after retardo;
                        end if;
                    when others =>
                        LISO <= CERO after retardo;
```

```
                    vaO <= CERO after retardo;
                    PEVO <= CERO after retardo;
                end case;
    end process sal_proc;

-- selección de las señales del procesador o del observador
    muxva <= not PET;                                   -- modelado ideal
    muxEV <= not PET;                                   -- la cache atiendo al procesador u observa
    muxLI <= not PET;                                   -- se utiliza la señal PET como de concesion
    muxO <= not PET;                                    -- si PET activo los otros observan

    LISTO <= LISP after retardo when muxLI = CERO else  -- multiplexor muxLI
            LISO after retardo;
    val <= vaP after retardo when muxva = CERO else     -- multiplexor muxva
            vaO after retardo;
    TPEV <= PEVP after retardo when muxEV = CERO else   -- multiplexor muxEV
            PEVO after retardo;

-- actualizacion de cache
PED <= not (TPED and (not (reloj) ) after retardo;
PEET <= not (TPEET and (not (reloj) ) after retardo;
PEV <= not (TPEV and (not (reloj) ) or pcero after retardo;   -- no se actualiza el bit de validez si pcero
                                                              -- es igual a uno
end;
```

Apéndice 2: Descripción en VHDL de un generador de peticiones para comprobar el controlador de cache

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

library LogicWorks;
use LogicWorks.debug.all;

    entity genpruecontpet is
        generic (retardo: time := 2 ns);
        port ( pcero: out std_logic;
        LES: out std_logic;
        PET: out std_logic;
        LISTO: in std_logic;

        AF: out std_logic;
        MLISTO: out std_logic;

        BMLES: out std_logic;
        BMPET: out std_logic;

        reloj: in std_logic);
    end;

    architecture estimulos of genpruecontpet is
        constant SI : std_logic := '1';
        constant NO : std_logic := '0';

        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;
            LES <= lectura ;
            PET <= NOPETICION;

            AF <= acierto;
            MLISTO <= NO;

            BMLES <= escritura;
            BMPET <= NOPETICION;
```

```
        wait until flanco_ascendente(reloj);
        pcero <= CERO after retardo;
        wait until flanco_ascendente(reloj);
        wait until flanco_ascendente(reloj) and LISTO = SI;

        LES <= escritura after retardo ;                -- efectua peticiones. Observa su propia peticion.
        PET <= PETICION after retardo;

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

        BMLES <= escritura ;
        BMPET <= PETICION;

        wait until flanco_ascendente(reloj);
        wait until flanco_ascendente(reloj);
        MLISTO <= SI after retardo;

        wait until flanco_ascendente(reloj) and LISTO = SI;
        MLISTO <= NO after retardo;

        LES <= lectura after retardo ;                  -- efectua peticiones. Observa su propia peticion.
        PET <= PETICION after retardo;

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

        BMLES <= lectura ;
        BMPET <= PETICION;

        wait until flanco_ascendente(reloj);
        wait until flanco_ascendente(reloj);
        MLISTO <= SI after retardo;

        wait until flanco_ascendente(reloj) and LISTO = SI;
        MLISTO <= NO after retardo;

        LES <= escritura after retardo ;                -- efectua peticiones. Observa su propia peticion.
        PET <= PETICION after retardo;

        AF <= acierto after retardo;                    -- acierto de escritura

        BMLES <= escritura ;
        BMPET <= PETICION;

        wait until flanco_ascendente(reloj);
        wait until flanco_ascendente(reloj);
        MLISTO <= SI after retardo;

        wait until flanco_ascendente(reloj) and LISTO = SI;
        MLISTO <= NO after retardo;

        . . .
    end process;
end;
```

Apéndice 3: Descripción en VHDL de un generador de peticiones para un multiprocesador idealizado

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

    entity pruebacontroladormulti is
        generic (retardo: time := 4 ns);
        port ( pcero: out std_logic;
        LES1P: out std_logic;                          -- Interface con el controlador de cache 1P
        PET1P: out std_logic;
        LISTO1P: in std_logic;
        DIR1P: out std_logic_vector (15 downto 0);     -- Interface con módulo campos de cache 1P
        DATO1P: out std_logic_vector (7 downto 0);
        DATO1L: in std_logic_vector (7 downto 0);      -- puede utilizarse para comprobar el funcionamiento
        Conc1P: out std_logic;                         -- Interface con el bus

        LES2P: out std_logic;                          -- Interface con el controlador de cache 2P
        PET2P: out std_logic;
        LISTO2P: in std_logic;
        DIR2P: out std_logic_vector (15 downto 0);     -- Interface con módulo campos de cache 2P
        DATO2P: out std_logic_vector (7 downto 0);
        DATO2L: in std_logic_vector (7 downto 0);      -- puede utilizarse para comprobar el funcionamiento
        Conc2P: out std_logic;                         -- Interface con el bus

        reloj: in std_logic);
    end;

    architecture estimulos of pruebacontroladormulti 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 CONCESION : std_logic := '1';
        constant NOCONCESION : 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 DIRF: std_logic_vector := x"FFFF";
        constant DATF: std_logic_vector := x"FF";

        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                                                   -- puesta a cero
   pcero <= UNO;                                        -- procesador 1P
   LES1P <= lectura ;
   PET1P <= NOPETICION;
   DIR1P <= DIRF;
   DATO1P <= DATF;
   Conc1P <= NOCONCESION after retardo;

   LES2P <= lectura ;                                   -- procesador 2P
   PET2P <= NOPETICION;
   DIR2P <= DIRF;
   DATO2P <= DATF;
   Conc2P <= NOCONCESION after retardo;

   wait until flanco_ascendente(reloj) and LISTO1P = UNO and LISTO2P = UNO;
   pcero <= CERO after retardo;
   wait until flanco_ascendente(reloj) and LISTO1P = UNO and LISTO2P = UNO;

   LES1P <= escritura after retardo;                    -- proc. 1P accede
   PET1P <= PETICION after retardo;                     -- escritura en direccion DIRA del dato “01”
   DIR1P <= DIRA after retardo;                         -- fallo de escritura
   DATO1P <= x”01” after retardo;
   Conc1P <= CONCESION after retardo;

   LES2P <= lectura after retardo;                      -- proc. 2P observa
   PET2P <= NOPETICION after retardo;
   DIR2P <= DIRF after retardo;                         -- fallo en observacion
   DATO2P <= DATF after retardo;
   Conc2P <= NOCONCESION after retardo;

                                                        -- esperar a que las 2 caches esten libres
   wait until flanco_ascendente(reloj) and LISTO1P = UNO and LISTO2P = UNO;

   LES1P <= lectura after retardo;                      -- proc. 1P observa
   PET1P <= NOPETICION after retardo;
   DIR1P <= DIRF after retardo;                         -- fallo de observación
   DATO1P <= DATF after retardo;
   Conc1P <= NOCONCESION after retardo;

   LES2P <= lectura after retardo;                      -- proc. 2P accede
   PET2P <= PETICION after retardo;                     -- lectura en direccion DIRA
   DIR2P <= DIRA after retardo;                         -- fallo de lectura
   DATO2P <= x”01” after retardo;
   Conc2P <= CONCESION after retardo;

                                                        -- esperar a que las 2 caches esten libres
   wait until flanco_ascendente(reloj) and LISTO1P = UNO and LISTO2P = UNO;

   LES1P <= lectura after retardo;                      -- proc. 1P accede
   PET1P <= PETICION after retardo;                     -- lectura en direccion DIRA
   DIR1P <= DIRA after retardo;                         -- fallo de escritura
   DATO1P <= x”02” after retardo;
   Conc1P <= CONCESION after retardo;

   LES2P <= lectura after retardo;                      -- proc. 2P observa
   PET2P <= NOPETICION after retardo;
   DIR2P <= DIRF after retardo;                         -- acierto en observacion, siendo una lectura
   DATO2P <= DATF after retardo;
   Conc2P <= NOCONCESION after retardo;

                                                        -- esperar a que las 2 caches esten libres
```

```
        wait until flanco_ascendente(reloj) and LISTO1P = UNO and LISTO2P = UNO;

        LES1P <= lectura after retardo;                    -- proc. 1P observa
        PET1P <= NOPETICION after retardo;
        DIR1P <= DIRF after retardo;                       -- fallo de observación
        DATO1P <= DATF after retardo;
        Conc1P <= NOCONCESION after retardo;

        LES2P <= lectura after retardo;                    -- proc. 2P accede
        PET2P <= PETICION after retardo;                   -- lectura en direccion DIRA
        DIR2P <= DIRA after retardo;                       -- acierto de lectura
        DATO2P <= x"20" after retardo;
        Conc2P <= CONCESION after retardo;

                                                           -- esperar a que las 2 caches esten libres
        wait until flanco_ascendente(reloj) and LISTO1P = UNO and LISTO2P = UNO;

        LES1P <= lectura after retardo;                    -- proc. 1P observa
        PET1P <= NOPETICION after retardo;
        DIR1P <= DIRF after retardo;                       -- fallo de observación
        DATO1P <= DATF after retardo;
        Conc1P <= NOCONCESION after retardo;

        LES2P <= lectura after retardo;                    -- proc. 2P observa
        PET2P <= NOPETICION after retardo;
        DIR2P <= DIRA after retardo;
        DATO2P <= DATF after retardo;
        Conc2P <= NOCONCESION after retardo;

                                                           -- esperar a que las 2 caches esten libres
        wait until flanco_ascendente (reloj) and LISTO1P = UNO and LISTO2P = UNO;
        . . .
    end process;
end;
```