Genís Bosch Pou
Nils Pachler de la Osa
03/12/2018 - MP - FIB - UPC

Multiprocesadores: Práctica 3

Coherencia de cache en un multiprocesador idealizado

Objetivo: En esta práctica se diseñará y analizará el comportamiento de un controlador de cache en un entorno multiprocesador idealizado.

El controlador de cache se basa en el diseñado en la práctica 2: un autómata con 4 estados correspondientes a: preparado, fallo lectura, acierto escritura y fallo escritura. Recordemos que esta cache era una cache con escritura inmediata.

En el entorno multiprocesador se convierte esta escritura inmediata en un protocolo de observación con invalidación. Es decir, los controladores estarán observando las peticiones del bus para saber si deben invalidar la propia cache. Con esta configuración, se distinguen 2 elementos principales en el controlador: el agente procesador y el agente observador. El primero es el responsable de atender las peticiones a la cache por parte del procesador. El diseño es idéntico al visto en la práctica 2. El segundo elemento se corresponde con el elemento responsable de observar por el bus las peticiones de otros procesadores.

El agente observador, al tratarse de un multiprocesador idealizado con invalidación, solo tiene que actuar cuando observa una petición de escritura que acierta en su propia cache. Es decir, el único momento donde entra en acción es cuando debe invalidar la copia la cache a causa de otra cache. Por ello, este autómata solo tiene dos estados: preparado y acierto escritura.

Las señales de entrada al circuito son simplemente las que dan la información mencionada: AF (acierto/fallo), BMPET (petición en una cache) y BMLES (tipo de petición en el bus).
La primera señal indica si se tiene el bloque en cache. Esta señal es, por tanto, una señal de estado, ya que da información del estado actual de la cache.
Las otras dos señales, en cambio, son señales de control, ya que dan información sobre lo que está pasando en el bus y, por extensión, en las otras caches.

Una vez establecida la naturaleza de estas señales, se puede proceder a diseñar el circuito. Este circuito no requiere mucha complejidad y se basa en 3 bloques: estado actual, determinación de estado siguiente y lógica de salida. En el apéndice 1 se muestra la descripción en VHDL del controlador descrito. El diseño es muy parecido al agente procesador. También es necesario añadir la lógica que distingue que elemento de los dos mencionados actúa (el agente procesador tiene prioridad).

A continuación, se muestra, de una forma esquemática, la lógica del agente observador, así como el diseño lógico del circuito.

Próximo estado	$\overline{\text{BMLES}}$ &BMPET&AF/MLISTO			
Estado	00	01	11	10
DESO	DESO	DESO	ESPO	ESPO
ESPO	ESPO	DESO	DESO	ESPO

Tabla 1. Lógica del próximo estado del agente observador

	Salida		
Estado / MLISTO	PEVO	VAO	LISO
DESO	0	0	1
ESPO/0	0	0	0
ESPO/1	1	1	0

Tabla 2. Lógica de salida del agente observador

En la tabla 1 se observa como el cambio de estado solo depende de la aparición de una petición de escritura en el bus que acierta en cache y de la señal MLISTO.
La lógica de salida (tabla 2) solo depende del estado y MLISTO, y se basa en activar las señales de escritura cuando hay que invalidar (MLISTO esta activada pero todavía no se ha cambiado el estado). La señal LISO indica si la cache, por parte del agente observador, está libre para recibir nuevas peticiones.

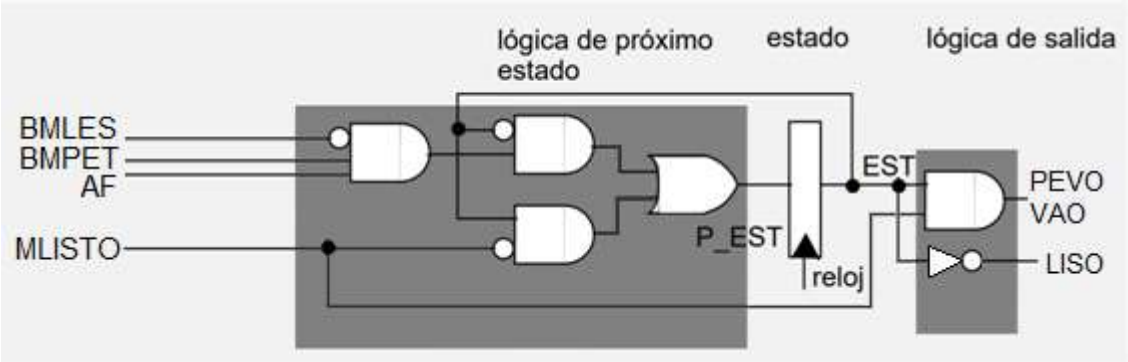

Figura 1. Diseño lógico del agente observador

Como se puede observar, las transiciones entre estados no requieren de mucha complejidad, ya que solo hay 2 estados y las transiciones solo dependen de 2 señales.

En la figura 1 hay que tener en cuenta que se ha diseñado el sistema con la base que DESO corresponde a valor lógico 0 y ESPO al valor 1.

Las señales de salida que se observan en la tabla 2 y figura 1 son:
- PEVO: activación del cambio de estado en la cache.
- VAO: invalidación del bloque.
- LISO: agente observador listo.

Para analizar el correcto funcionamiento del circuito se ha sometido a la cache a una prueba de test, que se corresponde con el análisis del agente observador en sus propias observaciones (apéndice 2). A continuación, se muestra la ventana de tiempos correspondiente a dicho análisis:

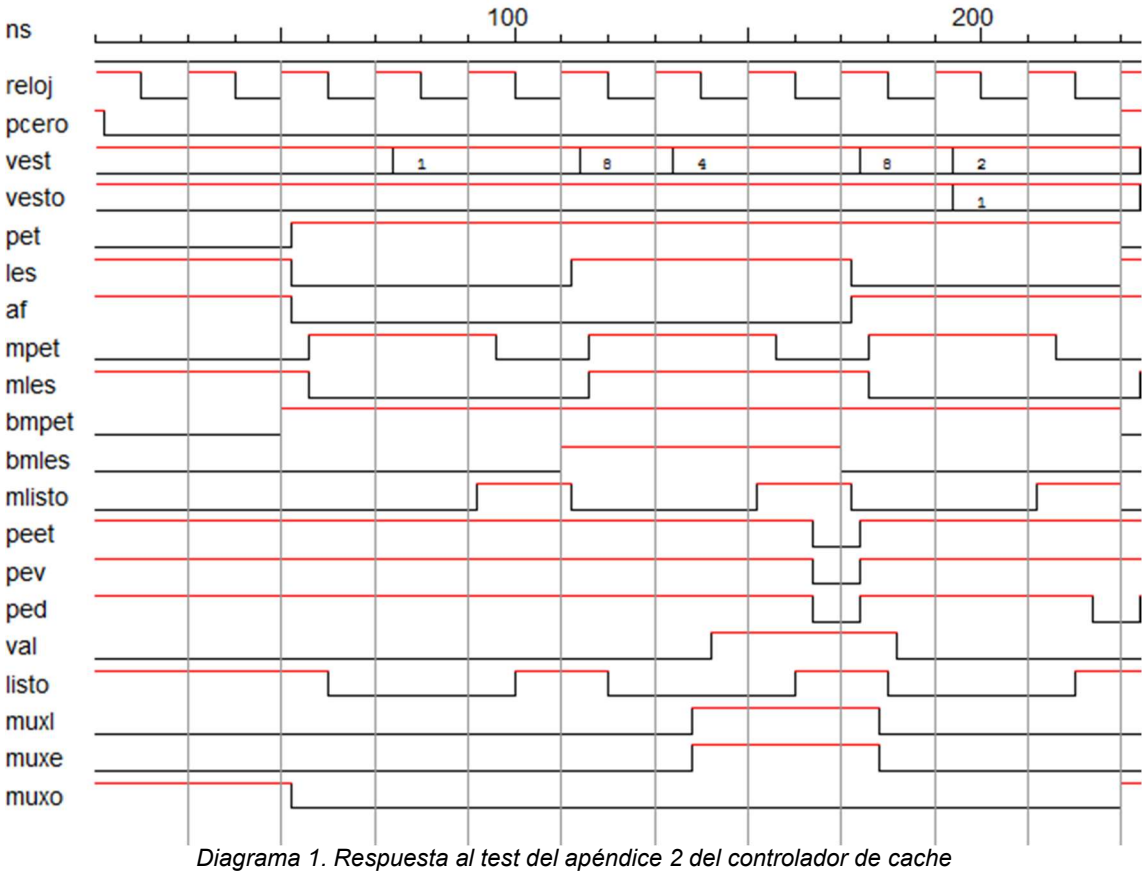

Diagrama 1. Respuesta al test del apéndice 2 del controlador de cache

Tal y como se observa en el apéndice 2, primero se ejecuta una fallada de escritura, seguido de una fallada en lectura y un acierto en escritura. Los aciertos en lectura no modifican el estado del controlador de cache, ya que se sirven en 1 ciclo.

La respuesta de la cache es muy semejante a la práctica 2 hasta el acierto en escritura, donde el agente observador entra en acción. Esto se corresponde con el tiempo 190ns en el diagrama 1. En este momento, el observador ve una petición de escritura en el bus y acierta en su cache, con lo que se prepara para invalidar el dato (cambia al estado 1, correspondiente a “acierto escritura”).

En este estado, el agente prepara la cache para recibir la dirección del bus y poder invalidarlo. En este caso, como la observación es de una petición propia, no tiene ningún efecto sobre el resultado, ya que como se ha comentado, las peticiones del procesador tienen prioridad. Esto evita tener que diseñar lógica para evitar cambios en peticiones propias, ya que el agente observador no tendrá potestad para decidir sobre el estado de la cache.

La parte que determina el estado de un bloque, gestionada por el controlador de cache, está dividida en dos procesos, dependiendo de si se gestiona el estado del bloque del controlador de cache que ha hecho la petición o del observador. Así pues, se trata de dos secciones, tanto en el process del agente procesador como del process del agente observador, tal y cómo podemos ver a continuación en su descripción VHDL:

```
--PROCESADOR
process(estado, LES, PET, AF, MLISTO, pcero)
        begin
                vaP <= CERO after retardo;
                if (pcero = CERO) then
                        case estado is
                                when DES =>
                                …
                                when ESP1 =>
                                        …
                                        vaP <= UNO after retardo;
                                        …
                                when ESP2 =>
                                …
                                when ESP3 =>
                                …
                        end case;
                end if;
        end process;


--OBSERVADOR
sal_proc: process(estado_O, BMLES, BMPET, AF, MLISTO)
        begin
                …
                vaO <= CERO ;
                …
                case estado_O is
                        when DESO =>
                                …
                        when ESPO =>
                                if MLISTO = UNO then
                                        vaO <= UNO after retardo;
                                                                …
                                else
                                        …
                                end if;
                        when others =>
                                …
                end case;
        end process sal_proc;
```

Finalmente, el estado del bloque, “val”, se define con la seleccion de valO o valP dependiendo de muxva:

```
val <= vaP after retardo when muxva = CERO else vaO after retardo;
```

Para analizar el correcto funcionamiento del controlador de cache, este se ha incorporado en el camino de datos de un multiprocesador idealizado con dos procesadores interconectados por un bus. Para ello, utilizaremos el código VHDL proporcionado en el Apéndice 3. A continuación, se muestra la ventana de tiempos correspondiente a dicho análisis:

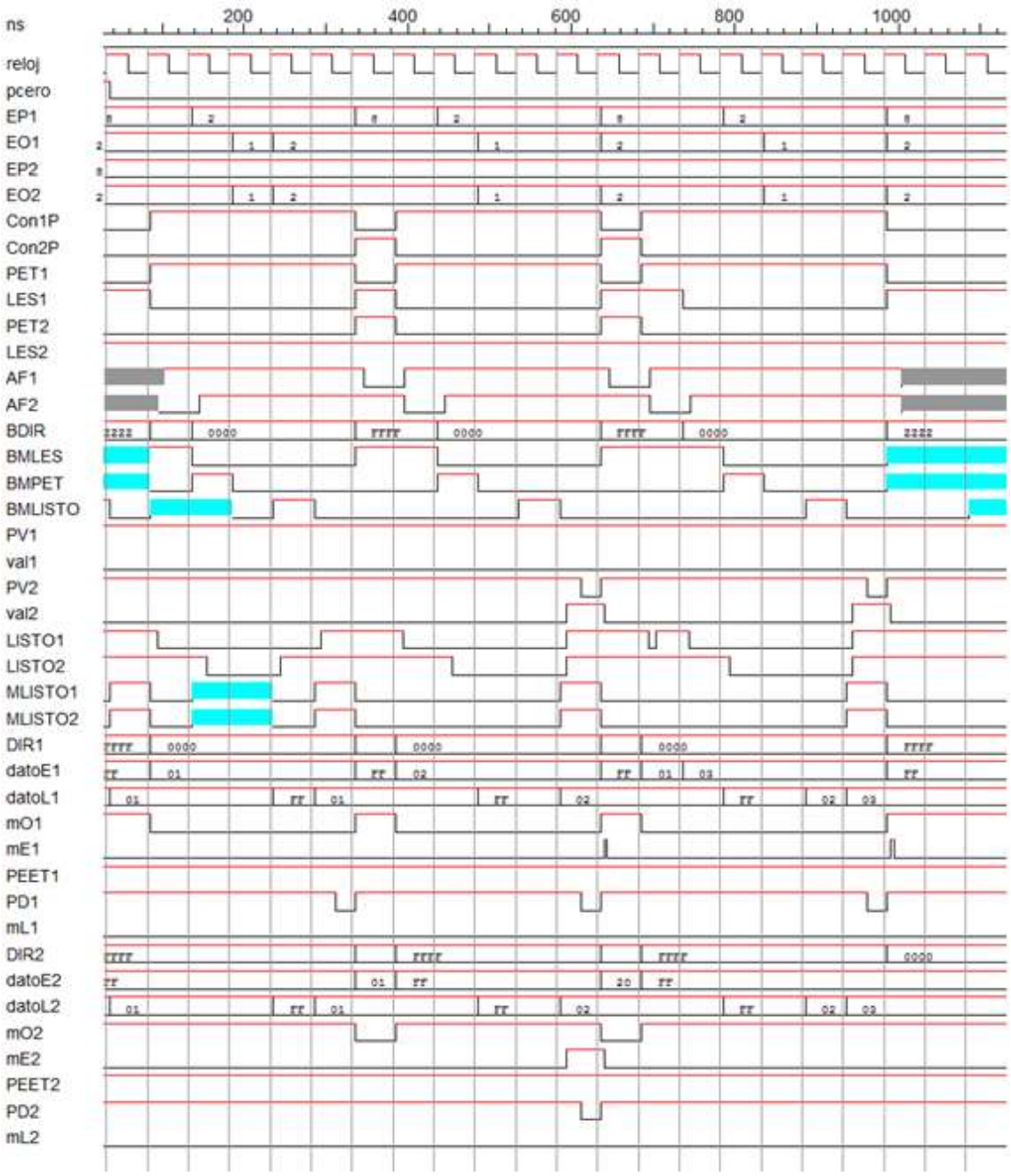

Diagrama 2. Respuesta incorporar el controlador de cache en el camino de datos

En este caso, nos fijamos que, dependiendo de Con1P o Con2P, es decir, a quien se concede el acceso al bus, las señales de LISTO1 y LISTO2 cambian según lo predicho.

Notamos que la primera acción que debe resolver el CC1 es una escritura con fallo sobre la dirección A, y, al tratarse de escritura immediata, el dato es escrito en memoria sin modificar el contenido de ninguna de las dos cachés, por lo que se mantiene su estado en inválido.

El controlador CC2, en su fase de observacion, obtiene una dirección que está como invàlida en su caché, por lo que no actua en consequencia.

El siguiente acceso a memoria es ralizado por P2, que obtiene un fallo de lectura sobre la dirección A, por lo que envia una petición a memória para obtener el bloque que contiene el dato A. El controlador de caché 1, que está observando, no actua, ya que no tiene el bloque en cache. Cuándo CC2 recibe la respuesta de memória con el bloque, finaliza la transacción.

En el siguiente acceso, P1 accede a memória con un fallo de lectura en la dirección A, por lo que enviará una petición a memória solicitando el bloque. Por otro lado, el controlador CC2 observará esta petición, y, aunque acertará sobre la dirección, no hará ningún cambio de estado al bloque, ya que permanece válido.

Finalmente, P2 hace una lectura del dato A, que ya tiene en la caché, por lo que obtiene un acierto de lectura. Así pues, el CC2 no envia ninguna petición a memória, por lo que el CC1 no observa nada, y el bus permanece libre.

Apéndice 1: Descripción 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 senal 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 modulo campos
		PED: out std_logic; -- y elementos perifericos
		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)
		); -- Observacion 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';
	-- observacion 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

-- senales temporales en el proceso de salida, sin sincronizar con la senal 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 after retardo;
			elsif (reloj = UNO and reloj'event) then
				estado <= prxestado after retardo;
				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
			prxestado <= estado after retardo;
			if pcero = CERO then
				case estado is
					when DES =>
						if PET = PETICION then
							if LES = escritura and AF = fallo then
								prxestado <= ESP3 after retardo;
							elsif LES = escritura and AF = acierto then
								prxestado <= ESP2 after retardo;
							elsif LES = lectura and AF = fallo then
								prxestado <= ESP1 after retardo;
							end if;
						end if;
					when ESP1 | ESP2 | ESP3 =>
						if MLISTO = UNO then
							prxestado <= DES after retardo;
						end if;
				end case;
			else
				prxestado <= DES after retardo;
			end if;
	end process;

-- logica de salida
	process(estado, LES, PET, AF, MLISTO, pcero)
		begin
			LISTO <= UNO after retardo;
			MPET <= CERO after retardo;
			TPED <= CERO after retardo;
			TPEET <= CERO after retardo;
			TPEV <= CERO after retardo;
			val <= CERO after retardo;
			muxL <= CERO after retardo;
			muxE <= CERO after retardo;
			MLES <= LES after retardo;
			if (pcero = CERO) then
				case estado is
					when DES =>
						if PET = PETICION and (AF = fallo or LES = escritura) then
							LISTO <= CERO after retardo;
							MPET <= UNO after retardo;
						end if;
					when ESP1 =>
						LISTO <= MLISTO after retardo;
						MPET <= not MLISTO after retardo;
						TPED <= MLISTO after retardo;
						TPEET <= MLISTO after retardo;
						TPEV <= MLISTO after retardo;
						val <= UNO after retardo;
						muxL <= UNO after retardo;
						muxE <= UNO after retardo;
					when ESP2 =>
						LISTO <= MLISTO after retardo;
						MPET <= not MLISTO after retardo;
						TPED <= MLISTO after retardo;
					when ESP3 =>
						LISTO <= MLISTO after retardo;
						MPET <= not MLISTO after retardo;
				end case;
			end if;
	end process;

-- actualizacion de cache
PED <= not (TPED and (not reloj));
PEET <= not (TPEET and (not reloj));
PEV <= not (TPEV and (not reloj));
end;
```

Apéndice 2: Descripción VHDL del generador de peticiones para comprobar el funcionamiento del controlador de cache

```
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 modulo 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 memori
			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 <= lectura 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;

                    LES <= escritura after retardo; -- lectura
                    PET<= PETICION after retardo;
                    AF <= ACIERTO after retardo; -- acierto 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 libre
                    MLISTO <= CERO after retardo;

            end process;
end;
```

Apéndice 3: Descripción VHDL del generador de peticiones para comprobar el funcionamiento de la cache

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.numeric_std.all;

library LogicWorks;
use LogicWorks.debug.all;

  entity cacheFSMtest is
                    generic (retardo: time := 2 ns);
      port ( pcero: out STD_LOGIC;
           LES: out STD_LOGIC;
           PET: out STD_LOGIC;
           LISTO: in STD_LOGIC;
           DIR: out std_logic_vector (15 downto 0);
           CDATO: in std_logic_vector (7 downto 0);
           DATO: out std_logic_vector (7 downto 0);

 --          acceso: out std_logic_vector (7 downto 0);
           reloj: in STD_LOGIC);
  end;

  architecture synth of cacheFSMtest is

    constant SI : std_logic := '1';
    constant NO : std_logic := '0';

    --constant A: std_logic_vector := "10100000";
    --constant F: std_logic_vector := "11110000" ;
    --constant AE: std_logic_vector := "10101110" ;
    --constant AF1: std_logic_vector := "11111110";

    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";

  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 <= SI;
         LES <= SI;
         PET <= NO;
         DIR <= DIRF;
         DATO <= DATF;

         wait until flanco_ascendente(reloj);
         pcero <= NO after retardo;
         wait until flanco_ascendente(reloj);
         wait until flanco_ascendente(reloj);
         LES <= NO after retardo;
         PET <= SI after retardo;
         DIR <= DIRA after retardo;
         DATO <= x"01" after retardo;
         --falla escriptura
```

```
        wait until flanco_ascendente(reloj) and LISTO = SI;
        LES <= SI after retardo;
        PET <= SI after retardo;
        DIR <= DIRA after retardo;
        DATO <= x"02" after retardo;
        --falla lectura;

       wait until flanco_ascendente(reloj) and LISTO = SI;
        LES <= NO after retardo;
        PET <= SI after retardo;
        DIR <= DIRA after retardo;
        DATO <= x"03" after retardo;
        --encert escriptura

       wait until flanco_ascendente(reloj) and LISTO = SI;
        LES <= SI after retardo;
        PET <= SI after retardo;
        DIR <= DIRA after retardo;
        DATO <= x"04" after retardo;
        --encert lectura

       wait until flanco_ascendente(reloj) and LISTO = SI;
        PET <= SI after retardo;
        LES <= NO after retardo;
        DIR <= DIRB after retardo;
        DATO <= x"05" after retardo;
        --falla escriptura

       wait until flanco_ascendente (reloj) and LISTO = SI;
        LES <= SI after retardo;
        PET <= SI after retardo;
        DIR <= DIRB after retardo;
        DATO <= x"06" after retardo;
        --falla lectura

        wait until flanco_ascendente(reloj) and LISTO = SI;

       --wait until flanco_ascendente (reloj);
      end process;
end;
```