**Multiprocesadores: Práctica 2**

***Cache con escritura inmediata***

*Objetivo:* En esta práctica se diseñará y analizará el comportamiento de un controlador de cache con escritura inmediata

El controlador se basa en un autómata con 4 estados: DES, ESP1, ESP2 y ESP3. El primer estado corresponde a un estado preparado: la cache esta lista para recibir peticiones. Los demás estados, en cambio, corresponden a una espera. ESP1 espera a recibir el bloque de memoria (fallo de lectura) mientras que ESP2 y ESP3, correspondientes respectivamente a acierto y fallo en escritura, esperan a recibir de memoria confirmación de escritura. El cambio de estado se efectúa cuando se recibe una petición en la que hay que acceder a memoria (todas las escrituras o fallo en lectura).

Las señales de salida del circuito son las que controlan el estado de la cache y sus bloques. También controlan las peticiones que se envían a memoria, así como la señal LISTO que indica si la cache está preparada para recibir peticiones.

En resumen, las señales de salida se agrupan en 3 grupos:

- LISTO: indica si la cache puede recibir nuevas peticiones

- Memoria: indica a memoria que debe efectuar una acción (MLES y MPET)

- Cache: señales y multiplexores para llevar el dato correspondiente a la cache o al procesador (val, PEET, PEV y PED)

Antes de diseñar el circuito, es necesario establecer distinciones entre estas señales. Principalmente, que señales dependen únicamente del estado y que señales dependen también de la entrada, lo que comúnmente se denominan autómatas de Moore en el primer caso, o de Mealy en el segundo.

Claramente, la señal LISTO depende únicamente del estado, ya que se activará cuando el controlador esté en el estado DES, con lo que es un autómata de Moore. La señal MPET actúa contrariamente a LISTO, con lo que también es considerado un autómata de Moore.

Las otras señales, en cambio, dependen también de la entrada, con lo que son autómatas de Mealy. En resumen:

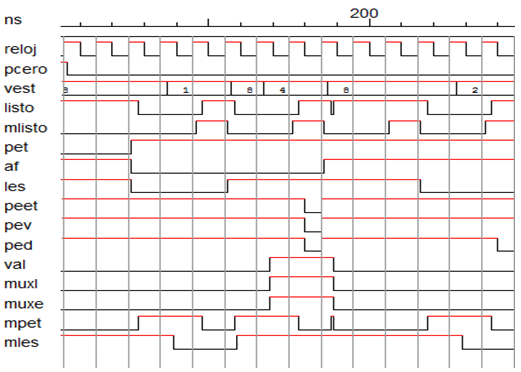
- MLES: depende de que se esté en estado DES y haya una petición así como del tipo de petición.

- val: solo se activará cuando memoria responda a una petición de lectura de bloque.

- PEET, PEV y PED: dependen del cambio de un bloque en cache (ya sea por fallada de escritura o acierto en lectura). Aunque esto depende solo del estado, también deben esperar a que la memoria responda para efectuar el cambio.

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.

Para analizar el correcto funcionamiento del circuito se ha sometido a la cache a una prueba de test, que se corresponde con el análisis de todos los casos posibles (apéndice 2). A continuación, se muestra la ventada 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.

En el diagrama 1 se observa cómo, al principio, la cache está lista. Entonces recibe una petición de escritura con fallo en cache. Inmediatamente la cache emite la petición a memoria y espera a la señal MLISTO que indica que la petición has sido procesada.

A continuación, recibe una petición de lectura que falla en cache, con lo que pide a memoria el bloque (110ns del diagrama). Cuando llega este bloque, se activan las señales necesarias que actualizarán el bloque en la cache (val, juntamente con PEET, PEV y PED).

Seguidamente llega un acierto en lectura que no modifica el estado (se observa en el cambio instantáneo de LISTO a los 175ns aproximadamente).

Finalmente llega una petición de escritura que acierta en cache, con lo que hay que pedir a memoria que actualice el estado (escritura inmediata) y hay que cambiar el dato en la cache (activación de PED a los 240ns).

La parte que determina el estado de un bloque, gestionada por el controlador de cache, y descrita en VHDL es:

val <= CERO after retardo;

…

if (pcero = CERO) then

case estado is

…

when ESP1 =>

val <= UNO after retardo;

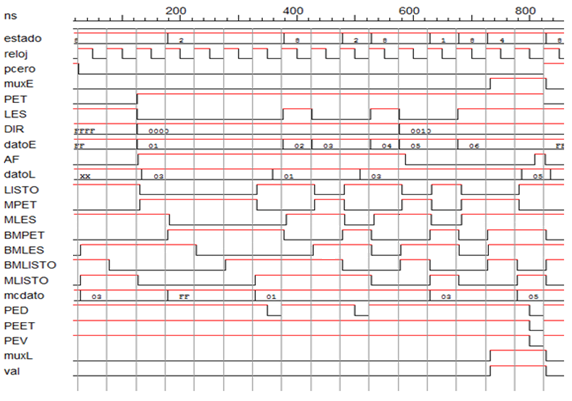
…

end case;

end if;

Donde, la validez de un bloque es 0 (no válido), y se pone a 1 (válido) en caso de una lectura con fallada, ya que el bloque es leído de memória a la cache.

Para determinar el correcto funcionamiento del controlador de caché diseñado, integramos el controlador de caché en el camino de datos y comprobamos el funcionamiento. Para ello, analizamos los casos de los que disponemos en el generador de peticiones. Obtendremos el cronograma del Diagrama 2.



*Diagrama 2. Respuesta al test del apéndice 3 del controlador de caché*

Primero, después de la puesta a 1 y posterior puesta a 0 de pcero, nos encontramos con una fallada de escritura, por lo que el estado pasa a ser ESP2, y vemos que se resuelve al escribir el dato y activar MLISTO. El estado vuelve a DES.

Después nos encontramos con una fallada de lectura sobre la dirección A. El estado cambia a ESP2, y se lee el dato, se carga en la caché y el estado vuelve a DES.

En el siguiente caso, obtendremos un acierto de escritura, así que el estado no va a cambiar y la petición va a realizarse sin ninguna acción del controlador de caché. Igual pasará con la siguiente, que produce un acierto de lectura.

Ahora, al generar una fallada en la escritura, el estado cambia de DES a ESP3. Se solicita escribir directamente a la memoria, por lo que el controlador de caché solo subministra el dato.

Finalmente, simulamos el caso de una fallada de lectura, que nos cambiará el estado a ESP1. El controlador de caché enviará una petición a memoria para obtener el dato sobre la dirección B. Lo escribirá en la caché activando el muxE y finalmente, con la señal de MLISTO, devolverá el estado a DES.

***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;