# MP Lab 3

Victor Mena y Rubén Juánez

16 de mayo de 2023

# Índice

# 1 CACHE CON ESCRITURA INMEDIATA

## 1. Trabajo 1

```
architecture estructural of arbitro is
    signal t_arb_concesion: std_logic;
    signal reg_fin_trans: std_logic;
    signal t_arb_conc_and: st_arb_concesion;
    signal prx_estado, estado: tipoestadoarb;
begin

    -- interface con el bus. Senyal de finalizacion de transaccion
    reg_mval: RD_1_arbi port map (reloj => reloj, pcero => pcero,
                                  e => fin_trans, s => reg_fin_trans);

    -- arbitraje
    arbi: arbitraje port map (arb => arb_control, peticion => arb_peticion,
                              concesion => t_arb_concesion);

    --logica de salida. Interface con el CC
    reg_conc: RD_1_arbi port map (reloj => reloj, pcero => pcero,
                                  e => t_arb_conc_and, s => arb_concesion);

    -- registro de estado
    reg_estado: process(reloj, pcero)
            variable v_estado: tipoestadoarb;
    begin
            if rising_edge(reloj) then
                    v_estado := prx_estado;
            elsif pcero = '1' then
                    v_estado := ARB;
            end if;
            estado <= v_estado;
    end process;

    -- logica de proximo estado
    prx_esta: process(estado, t_arb_concesion, reg_fin_trans, pcero)
            variable v_prxestado: tipoestadoarb;
    begin
            v_prxestado := estado;
            if(pcero /= '1') then
                    case estado is
                            when ARB =>
                                    if t_arb_concesion = '1' then
                                            v_prxestado := ESPARB;
                                    end if;
                            when ESPARB =>
                                    if reg_fin_trans = '0' then
                                            v_prxestado := ARB;
                                    end if;
                    end case;
            else
                    v_prxestado := ARB;
            end if;
            prx_estado <= v_prxestado;
    end process;

    -- logica se salida
    logi_sal: process(estado, t_arb_concesion, pcero)
            variable v_t_arb_concesion: st_arb_concesion;
    begin
            if(pcero /= '1') then
                    case estado is
                            when ARB =>
                                    v_t_arb_concesion := t_arb_concesion;
                            when ESPARB =>
                                    v_t_arb_concesion := '0';
                    end case;
            end if;
            t_arb_concesion <= v_t_arb_concesion;
    end process;
end;
```

## 2. Trabajo 2

Sirve para decidir si el valor del bus se tiene que guardar en el registro. Usa las dos señales **X.val** y **X.esc** para en el caso de un **PtE** que como se muestar en la figura 8 es la única transacción que modifica el autómata del observador.

## 3. Trabajo 10

| | procesador | | | Señales de estado (AF, est.) | | | | memoria | | árbitro | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | petición | | !peticón | lectura | | escritura | | resp. memoria | | concesión | | |
| | inicio | ! inicio | | si | no | si | no | si | no | si | no | |
| DES0 | INI<br>**pc_listo** | CMPETIQ<br>**ET_acc, EST_acc, pc_listo** | DES0<br>**pc_listo** | | | | | | | | | |
| INI | | | | | | | | | | | | ESCINI |
| ESCINI | | | | | | | | | | | | HECHOE<br>**ET_acc, ET_esc, EST_DE, DAT_acc** |
| DES | | CMPETIQ<br>**ET_acc, EST_acc, pc_listo** | DES<br>**pc_listo** | | | | | | | | | |
| CMPETIQ | | | | LEC | PML | PMEA<br>**arb_pet** | PMEF | | | | | |
| LEC | | | | | | | | | | | | HECHOL<br>**DAT_acc** |
| HEHCOL | | | | | | | | | | | | DES<br>**rc_val** |
| PML | | | | | | | | | | ESPL<br>**m_acc, m_pet, trans_bus** | PML<br>**arb_pet** | |
| ESPL | | | | | | | | ESB<br>**trans_bus** | ESPL | | | |
| ESB | | | | | | | | | | | | LEC<br>**ET_acc, ET_esc, EST_DE, DAT_acc, DAT_esc, muxE** |
| PMEA | | | | | | | | | | ESPEA<br>**pm_val, pm_esc, trans_bus** | PMEA<br>**arb_pet** | |
| ESPEA | | | | | | | | ESCP<br>**trans_bus** | ESPEA | | | |
| ESCP | | | | | | | | | | | | HECHOE<br>**DAT_acc, DAT_esc** |
| PMEF | | | | | | | | | | ESPEF<br>**pm_val, pm_esc, trans_bus** | PMEF<br>**arb_pet** | |
| ESPEF | | | | | | | | HECHOE<br>**trans_bus** | ESPEF | | | |
| HEHCOE | | | | | | | | | | | | DES |

Figura 1: Tabla de transiciones agente procesador del *Proyecto 1*

# 4. Trabajo 12

| | procesador | | | Señales de estado | | | |
|---|---|---|---|---|---|---|---|
| | petición | | !petición | lectura | | escritura | |
| | lectura | escritura | | si | no | si | no |
| DESO | DESO | CMPETO<br>**observación, listo** | DESO | | | | |
| CMPETO | | | | | | EEST | DESO |
| | | | | | | **observación, EST_acc, EST_esc, EST_DE** | |
| EEST | | | | | | | DESO |

Figura 2: Tabla de transiciones agente observador del *Proyecto 1*

# 5. Trabajo 28

| | Antes | | | | | | Después | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | C 0 | | | | C 1 | | | | C 0 | C 1 |
| acceso | C 0 est. | C 1 est. | ARB conc | Bus trans. | mem. var. | mem. val. | cont. | var. | val. | est. | cont. | var. | val. | est. | Secuencia de estados del observador | Secuencia de estados del observador |
| 0. inicio | | | | | A | X | 0 | A | 8 | V | | | | | | |
| 0. inicio | | | | | B | X | | | | | 1 | B | 10 | V | | |
| 1. P0 load A<br>1. P1 no acceso | V | | | | A | X | 0 | A | 8 | V | | | | | DESO | DESO |
| 2. P0 no acceso<br>2. P1 load B | | V | | | B | X | | | | | 1 | B | 10 | V | DESO | DESO |
| 3. P0 store A(3)<br>3. P1 no acceso | V | | C | PtE | A | 3 | 0 | A | 8 | I | | | | | DESO, CMPTEO, EEST | DESO, CMPTEO |
| 4. P0 no acceso<br>4. P1 store A(5) | I | I | C | PtE | A | 5 | | | | | | | | | DESO, CMPTEO | DESO, CMPTEO |
| 5. P0 no acceso<br>5. P1 load A | I | I | C | Pt | A | 5 | | | | | 0 | A | 5 | V | DESO | DESO |
| 6. P0 store A(13)<br>6. P1 no acceso | I | V | C | PtE | A | 13 | | | | | 0 | A | 5 | I | DESO, CMPTEO | DESO, CMPTEO, EEST |

Figura 3: Secuencia de accesos del *Proyecto 2*

# 6. Trabajo 31

| | procesador | | | | Señales de estado | | | |
|---|---|---|---|---|---|---|---|---|
| | petición | | | !petición | lectura | | escritura | |
| | lectura | escritura | | | si | no | si | no |
| | | trans_bus | !trans_bus | | | | | |
| DESO | DESO | **listo** | CMPETO<br>**observación, listo** | DESO | | | | |
| CMPETO | | | | | | | EEST | DESO |
| | | | | | | | **observación, EST_acc, EST_esc, EST_DE** | |
| EEST | | | | | | | | DESO |

Figura 4: Tabla de transiciones agente observador del *Proyecto 2*

# 7. Trabajo 37

En la figura 5 se muestra la ventana temporal antes de aplicar la mejora de **trans_bus**. En la cual se puede observar como ambos controladores de cache ejecutan un proceso de observació en el bus. En concreto no encontramos en el ciclo 23 donde el procesador 0 hace una escritura, pero como el bloque no se encuentra en ninguna de las cache los observadores no realizan ninguna operación de invalidación.

Por otra parte, la figura 6 nos muestra la ventana temporal una vez añadida la nueva mejora. Si nos fijamos bien, podemos ver como ahora solo el controlador de cache de P1 activa el proceso de observación del bus con el agente observador.

Ambas capturas se han realizado en los mismos ciclos y mismo espacio temporal de la ventana.

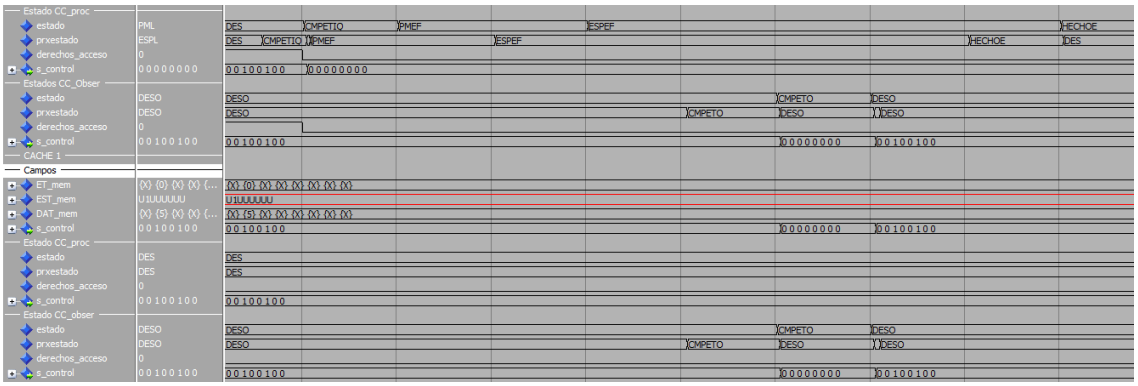

Figura 5: Ventana temporal de la simulación del Trabajo 26

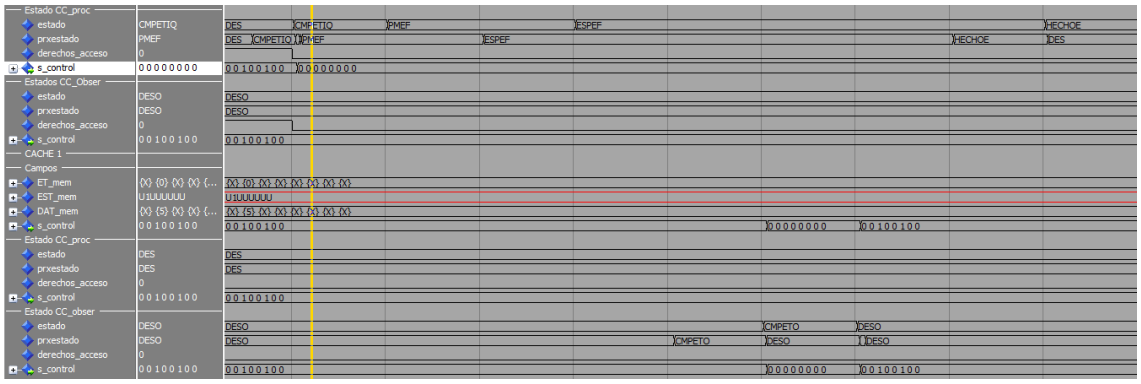

Figura 6: Ventana temporal de la simulación del Trabajo 35

# 8. Trabajo 41

Se usa la señal **trans_bus** debido a que ésta se mantiene activa durante toda la transacción que está pasando por el bus a diferencia de **arb_conc** que solo se activa durante un ciclo que es el mismo en el que se le concede el bus al CC respectivo.

# 9. Trabajo 42

| | ciclos | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Agentes | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| CC/proc0 | PMX | ESPX | ESPX | ESPX | ESPX | ESPX | ... | ... | ... | ... | ... | ... | ... |
| CC/proc1 | | DES | CMPET | PMX | PMX | PMX | PMX | ESPX | ESPX | ESPX | ESPX | ESPX | ... |
| BUS | | bus_I | bus_M | mem | bus_V | | | bus_I | bus_M | mem | bus_V | | |
| Arbitro | ARB | ESPARB | ESPARB | ESPARB | ESPARB | ESPARB | ARB | ESPARB | ESPARB | ESPARB | ESPARB | ESPARB | ARB |
| acciones | arb_pet | conc. | arb_pet | arb_pet | arb_pet | arb_pet | arb_pet | conc. | | | | | arbitraje |

Figura 7: Diagrama temporal de acciones de arbitraje y estados en los CC

# 10. Trabajo 44

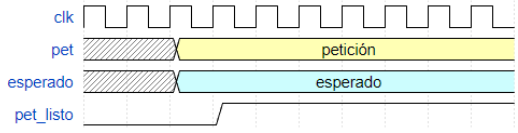


Figura 8: Diagrama temporal procedimiento *Plectura*

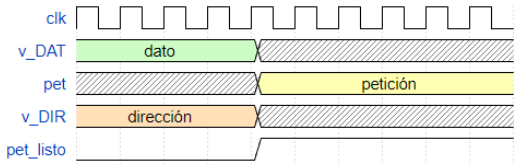


Figura 9: Diagrama temporal procedimiento *Pescritura*

Analizando el programa de pruebas podemos ver ocmo el proceso productor hace tres peticiones de escritura seguido de otras tres de lectura. Va lanzando las peticiones y el proceso consumidor las coge de un archvio y a la vez va comprobando que el resultado obtenido sea el esperado.

# 11. Trabajo 53

En las figuras 10 y 11 se muestra el cambio introducido en nuestro camino de datos. Las señales de control han sido desacopladas de tal forma que se usa un multiplexor para cada una de ellas para decidir cual se coge menos las señales **DAT** que siempre se cogen del agente procesador.

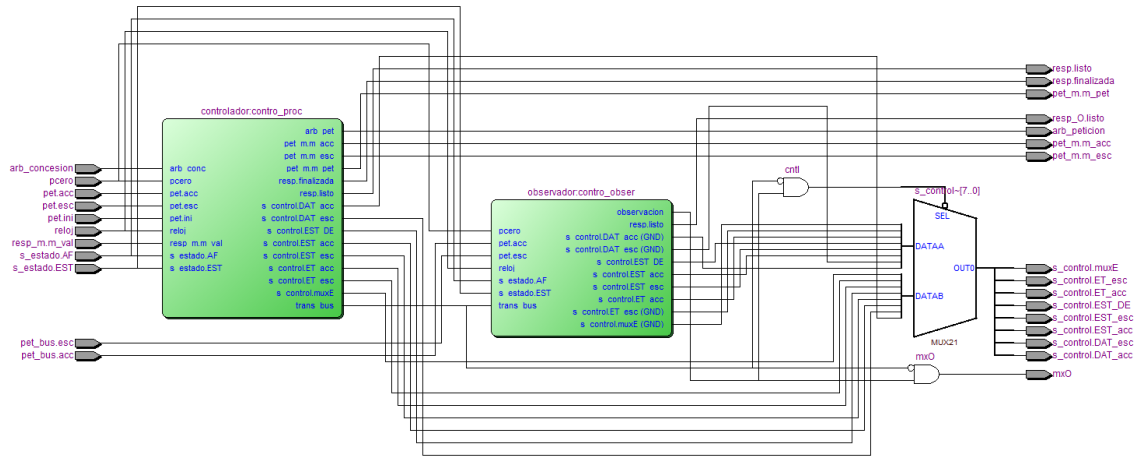


Figura 10: Diagrama RTL diseño original

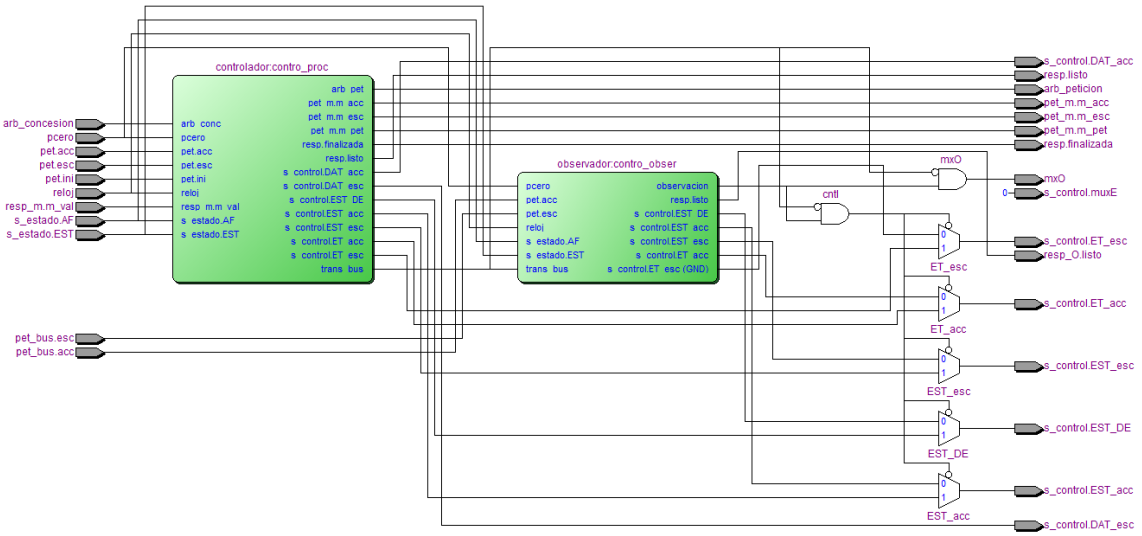


Figura 11: Diagrama RTL diseño nuevo con acceso desacoplado

# 12. Trabajo 56

Prestando atención a las señales **s_control** de las figuras 12 y 13 (sobretodo en el ciclo que viene después del marcador) vemos como antes de desacoplar las señales la elección de ellas se decidía del bloque completo, es decir, procedentes del agente procesador o del observador. En cambio al desacoplar las señales la elección de éstas puede ser una mezcla de las obtenidas por los dos agentes.

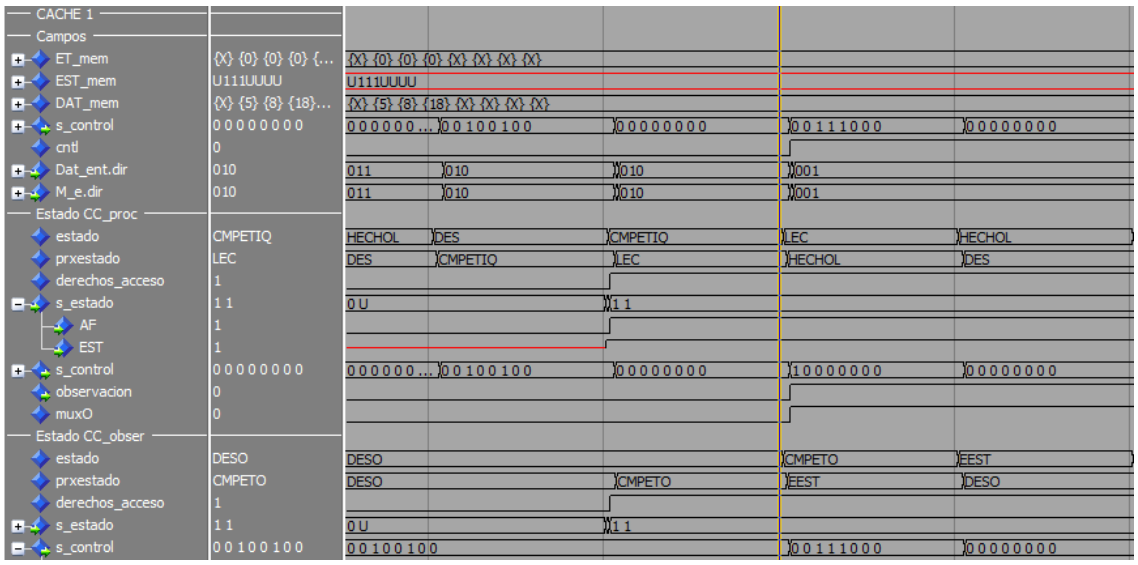


Figura 12: Ventana temporal simulación Modelsim diseño original

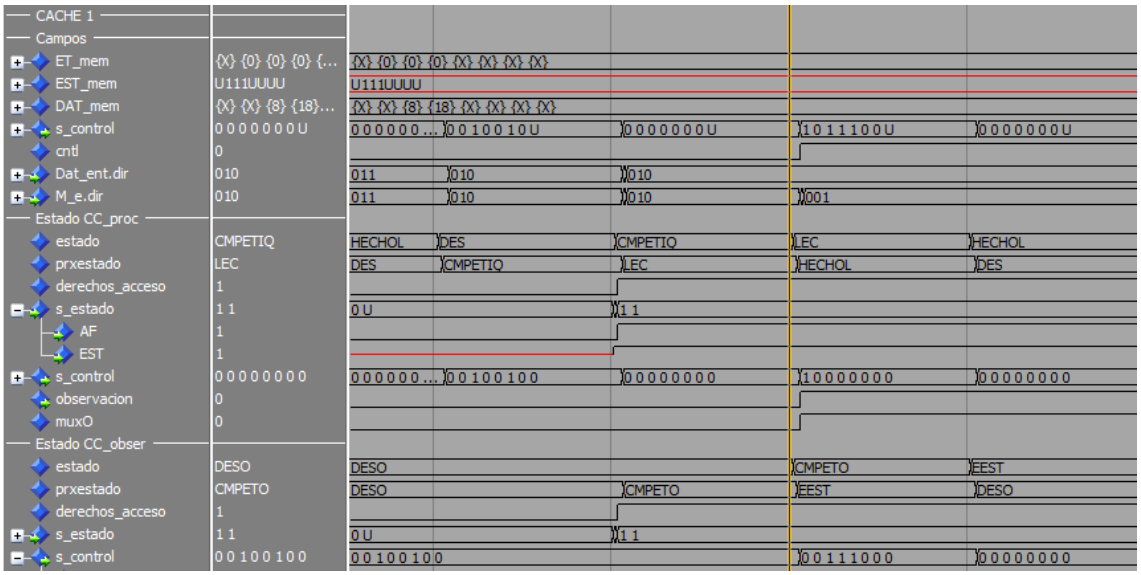


Figura 13: Ventana temporal simulación Modelsim nuevo con acceso desacoplado

# 13. Trabajo 57

En la siguiente figura se muestra una secuencia temporal de accesos y transacciones que verifican el afrimación del enunciado. El procesador 1 va a invalidar el bloque que va a expulsar y esto supone un grave problema.

| | ciclos | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Agentes | 0 | 1 | 2 | 5 | 6 | 7 | 8 | 9 | 10 |
| CC/proc0 | DES | CMPETIQ | PML | PML | PML | PML | PML | PML | PML |
| CC/obs1 | DESO | DESO | DESO | DESO | CMPETO | EEST | DESO | DESO | DESO |
| CC/proc1 | DES | CMPETIQ | PMEX | ESPEX | ESPEX | ESPEX | ESPEX | ESPEX | ESPEX |
| Arbitro | | ARB | ESPARB | ESPARB | ESPARB | ESPARB | ESPARB | ESPARB | ARB |
| acciones | | arb_pet | conc. | | | | | | arb_pet |

Figura 14: Diagrama temporal expulsión de bloque por fallo de lectura

# 14. Trabajo 74

En el siguiente diagrama temporal se puede observar como se realiza una operación de escritura que da acierto y pide concesión del bus. Antes de que se le conceda se efectúa una observación que provoca una invalidación de dicho contenedor. Más tarde la cache consigue acceso al bus y hace la petición. Al finalizar la petición, se actualiza el campo de datos porque originalmente era un acierto auqnue el agente observador haya invalidado ese contenedor.

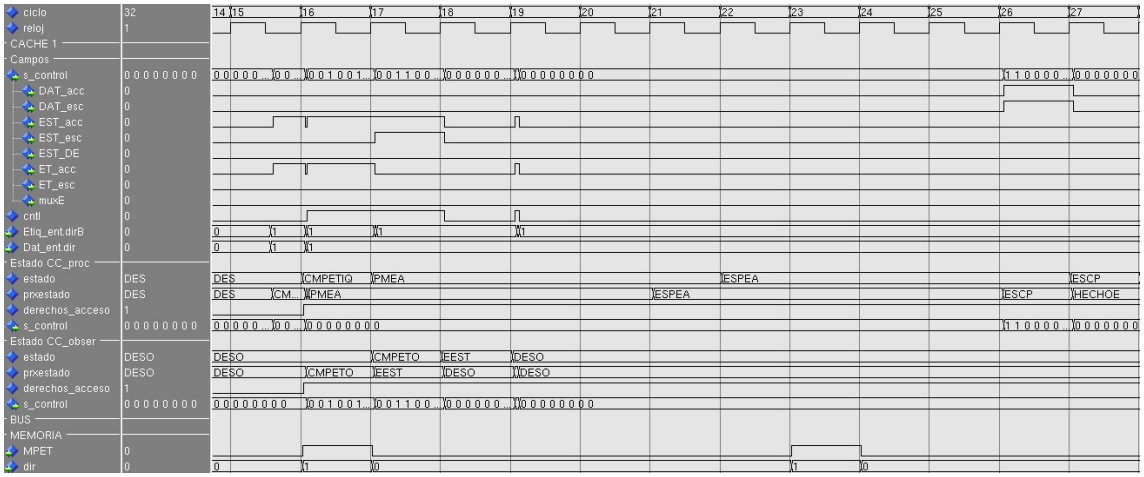


Figura 15: Ventana temporal simulación Modelsim observación *cruce*

## 15. Trabajo 75

La implementación actual ya garantiza la coherencia de cache haciendo uso de los dos agentes y permitiendo que cada procesador pueda visulaizar los accesos que hacen los otros. Además al usar un bus como red de interconexión nos aseguramos de que todos vean los accesos en el mismo orden, garantizando así también la consistencia de cache.

## 16. Trabajo 76

Añadir este buffer de escrituras a nuestro diseño actual mejora la cantidad de accesos por timepo. La coherencia se sigue manteniendo debido a que, como se especifica en el enunciado, si alguna de las lecturas coincide con un bloque que está en el buffer para escribirse se quedará bloqueada hasta que se resuelva la dependencia. No obstante, éste buffer no mantiene la secuencialidad de los accesos original, pero sí la consistencia y las dependencias que haya entre accesos.

## 17. Trabajo 77

Para hacer la lectura de los campos ET y EST, en peticiones de store, se nos tendrá que haber concedido el bus, ese será el momento en el que los podremos leer antes. Para las lecturas no puede ser igual porque sino tendríamos que solicitar el bus incluso para las peticiones que sean aciertos de lectura.

Así mismo, en el estado DES y DES0 podríamos leer los campos ET y EST si se trata de una lectura. Para las escrituras en el estado PME (juntamos fallos y aciertos) cuando se concediese el bus pasaríamos a un nuevo estado en el que comprobaríamos ET y EST (CMPESC) y a partir de ahí pasaríamos a PMEF o PMEA. De esta forma nos ahorraríamos la actualización innecesaria de DAT y además estaremos manteniendo la consistencia secuencial.

# 18. Trabajo 80

| | | procesador | | | | Señales de estado (AF, est.) | | | | memoria | | árbitro | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | petición | | | !peticón | lectura | | escritura | | resp. memoria | | concesión | | |
| | | inicio | ! inicio | | | si | no | si | no | si | no | si | no | |
| | | | lectura | escritura | | | | | | | | | | |
| DES0 | | INI | CMPETIQ | PME | DES0 | | | | | | | | | |
| | | **pc_listo** | **ET_acc, EST_acc, pc_listo** | **arb_pet** | **pc_listo** | | | | | | | | | |
| INI | | | | | | | | | | | | | | ESCINI |
| ESCINI | | | | | | | | | | | | | | HECHOE |
| | | | | | | | | | | | | | | **ET_acc, ET_esc, EST_DE, DAT_acc** |
| DES | | | CMPETIQ | PME | DES | | | | | | | | | |
| | | | **ET_acc, EST_acc, pc_listo** | **arb_pet** | **pc_listo** | | | | | | | | | |
| CMPETIQ | | | | | | LEC | PML | | | | | | | |
| | | | | | | | **arb_pet** | | | | | | | |
| LEC | | | | | | | | | | | | | | HECHOL |
| | | | | | | | | | | | | | | **DAT_acc** |
| HEHCOL | | | | | | | | | | | | | | DES |
| | | | | | | | | | | | | | | **rc_val** |
| PML | | | | | | | | | | | | ESPL | PML | |
| | | | | | | | | | | | | **m_acc, m_pet, trans_bus** | **arb_pet** | |
| ESPL | | | | | | | | | | ESB | ESPL | | | |
| | | | | | | | | | | **trans_bus** | | | | |
| ESB | | | | | | | | | | | | | | LEC |
| | | | | | | | | | | | | | | **ET_acc, ET_esc, EST_DE, DAT_acc, DAT_esc, muxE** |
| PME | | | | | | | | | | | | COMPES | PME | |
| | | | | | | | | | | | | **ET_acc, EST_acc, ET_esc, ET_esc** | **arb_pet** | |
| ESPEA | | | | | | | | | | ESCP | ESPEA | | | |
| | | | | | | | | | | **trans_bus** | | | | |
| ESCP | | | | | | | | | | | | | | HECHOE |
| | | | | | | | | | | | | | | **DAT_acc, DAT_esc** |
| COMPESC | | | | | | | | ESPEA | ESPEF | | | | | |
| | | | | | | | | **pm_esc** | | | | | | |
| ESPEF | | | | | | | | | | HECHO | ESPEF | | | |
| | | | | | | | | | | **trans_bus** | | | | |
| HEHCOE | | | | | | | | | | | | | | DES |

Figura 16: Tabla de transiciones entre estados *Proyecto 5*

# 19. Trabajo 84

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

use work.componentes_arb_circular_pkg.all;

entity arbitraje is
        generic(n: natural:=2);
        port(reloj, pcero: std_logic;
                peticiones: in std_logic_vector(n-1 downto 0);
                concesiones: out std_logic_vector(n-1 downto 0));
end arbitraje;

architecture estructural of arbitraje is
signal prioridades: std_logic_vector(n-1 downto 0);
signal t_concesiones: std_logic_vector(n-1 downto 0);
signal or_reduce: std_logic_vector(n-1 downto 0);
signal or_reduc: std_logic;

component modulo is
        port(reloj, pini, u, c: in std_logic; sal: out std_logic);
end component;

type t_mat_ors is array (0 to n-1) of std_logic_vector(0 to n - 2);
type t_mat is array (0 to n-1) of std_logic_vector(0 to n-1)
signal ors_data: t_mat := (other => (others => '0'));
signal tempsalidas: std_logic_vector(0 to n-1);
signal ors: t_mat_ors;

begin

mods_outer: for i in 0 to n-1 generate
        signal tmpor: std_logic_vector(n-1 downto 0);
begin
        mods_salidas: for j in i+1 to n-1 generate
                signal tmp: std_logic;
        begin
                mymod: modulo port map(reloj => reloj, pini => pcero, u => tempsalidas(i),
                        c => tempsalidas(j), sal => tmp);
                ors_data(i)(j) <= tmp and peticiones(j);
                ors_data(j)(i) <= (not tmp) and peticiones(j);
        end generate;
        tmpor(0) <= ors_data(i)(0);
        or_gen: for k in 1 to n-1 generate begin
                tmpor(k) <= tmpor(k-1) or ors_data(i)(k);
        end generate;
        tempsalidas(i) <= peticiones(i) and not tmpor(n-1);
        t_concesiones(i) <= tempsalidas(i) after ret_arb;
end generate;
or_reduce <= (or_reduce(n-2 downto 0) & '0') or peticiones;
or_reduc <= or_reduce(n-1);

pri: RD_arbi_1 port map (reloj => reloj, pini => pcero, pe => or_reduc,
                        e => t_concesiones(n-1), s => prioridades(0));
rest:   for i in 1 to n-1 generate
ele:            RD_arbi_0 port map  (reloj => reloj, pini => pcero,
                        pe => or_reduc, e => t_concesiones(i-1), s => prioridades(i));
                end generate rest;

arbi:   arb_propa generic map (n => n)
                                port map (peticiones => peticiones, prioridades => prioridades,
                        concesiones => t_concesiones);

concesiones <= t_concesiones;
end;
```