Práctica 5

- Sistema para jugar al *Black Jack*

Práctica 5

- Desarrollar en VHDL un sistema para jugar al popular juego de cartas *Black Jack*.
  - El jugador pide cartas y debe intentar acercarse a una puntuación de 21. Si se pasa, pierde la partida.
  - Existe un contador que SIEMPRE está contando de 0 a 51 (tiene reset, pero su señal de "cuenta" está siempre conectada a '1'). El jugador, cuando pide una carta, pulsa un botón y lo detiene. El valor devuelto por el contador indica la posición de memoria donde se leerá la carta que el sistema la da al jugador.
  - Cada carta numérica tiene su valor, y las figuras tienen valor 10.
  - El jugador tiene 1 baraja a su entera disposición, con 52 cartas (para cada palo, cartas 1-10 y 3 figuras que valen 10 puntos).
  - El jugador puede decidir plantarse en cualquier momento. Pero si pide carta y su nueva puntuación es mayor que 21, pierde la partida.
  - El sistema funciona con el reloj @1Hz (con divisor de frecuencias).

Práctica 5

- Simplificaciones que haremos en este sistema:
  - El jugador juega contra sí mismo, no hay banca.
  - Los ases no tienen doble valor (1 u 11); sólo valen 1.
  - No se pueden “separar cartas” cuando el jugador tenga 2 cartas iguales.
  - No se pueden doblar apuestas a medida que el juego avanza. Sólo hay 2 posibles resultados: o plantarse con una puntuación dada o perder la partida.

Práctica 5

- Las cartas que quedan en la baraja del jugador se guardarán en una memoria.
- Esta memoria tendrá 64 posiciones, y las 52 primeras deben inicializarse a los valores de estas cartas.
  - MEM(0) = 1; MEM(1) = 2; …, MEM(12) = 10;
  - MEM(13) = 1; ….
- Cuando el sistema dé una carta al jugador, ésta se debe eliminar de la memoria.
  - El contenido de la posición de memoria seleccionada se actualizará a (others => ‘0’).
- Si, tras haber eliminado alguna carta (y tras haberla borrado de la memoria), el contador vuelve a devolver otra vez la misma carta (es decir, el contenido de la dirección de memoria que el contador proporciona es 0), el juego debe indicar esta situación al jugador encendiendo un LED e indicarle que vuelva a pedir una nueva carta.

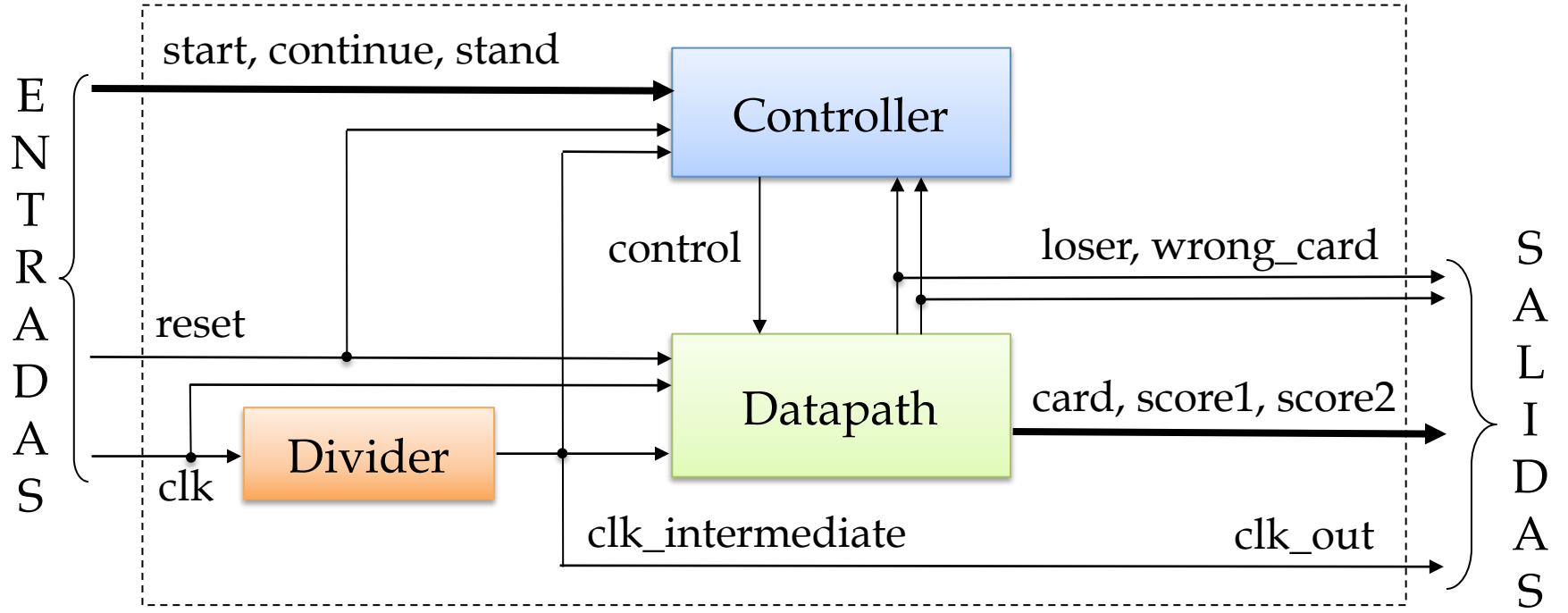
Práctica 5

- Entradas:
  - Reset.
  - Reloj.
  - Comenzar juego (“start”, switch).
  - Seguir jugando (“continue”, botón).
  - Plantarse (“stand”, botón).
- Salidas:
  - Carta incorrecta (“wrong_card”, 1 LED).
  - Valor de la carta actual (“card”, 1 display 7 segmentos).
  - Perder partida (“loser”, 1 LED).
  - Puntuación acumulada jugador (“score1” y “score2”, 2 displays 7 segmentos).
  - Reloj (a 1 Hz) para depurar (“clk_out”, 1 LED).

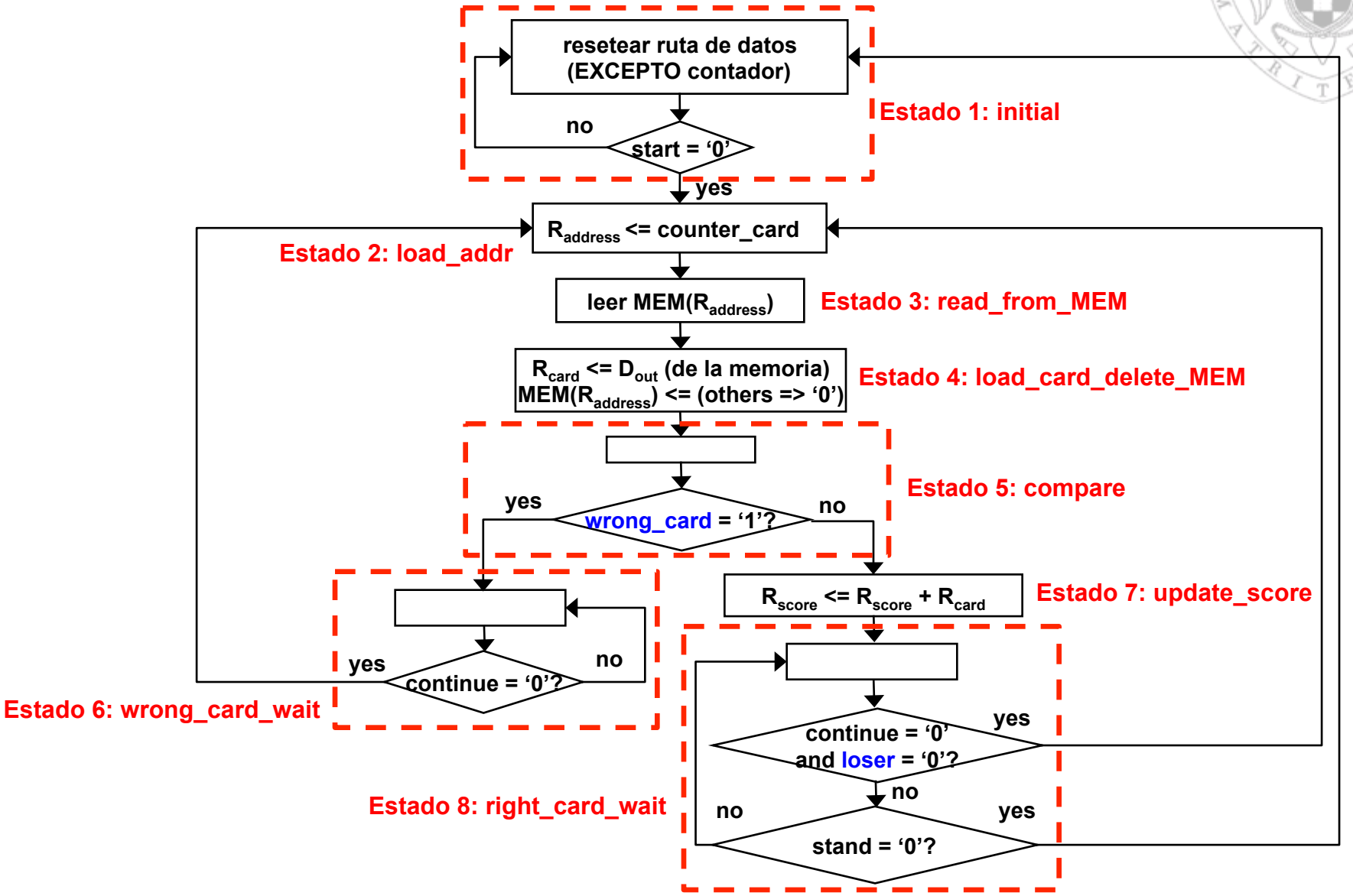
Los pulsadores son activos a baja: cuando se pulsan generan un ‘0’

Esquema RTL

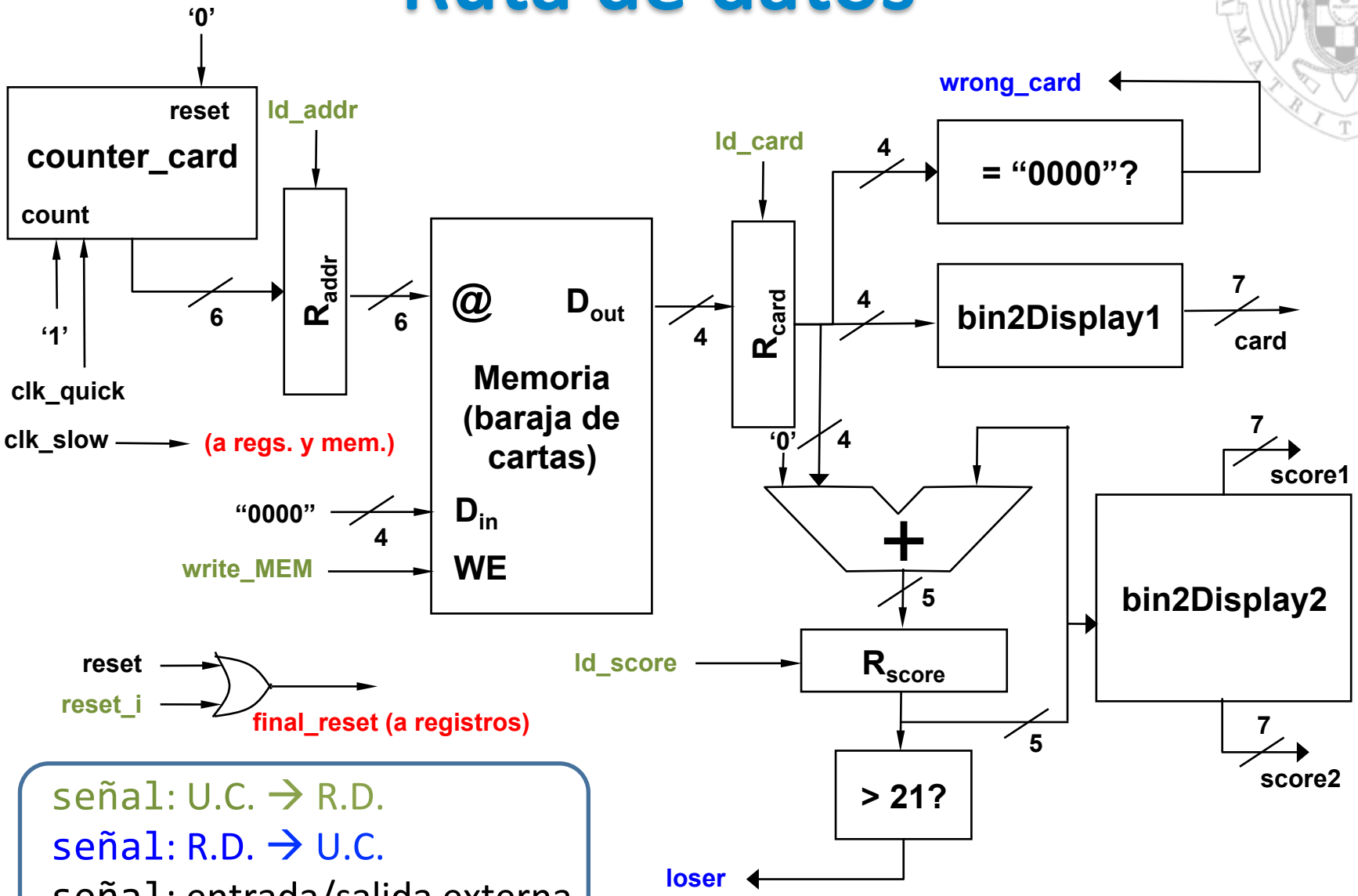
```
entity blackjack is
   port (clk, reset, start, continue, stand: in std_logic;
         loser, wrong_card: out std_logic;
         card, score1, score2: out std_logic_vector (6 downto 0);
         clk_out: out std_logic);
end blackjack;
```



Unidad de control



Ruta de datos



señal: U.C. → R.D.
señal: R.D. → U.C.
señal: entrada/salida externa

Unidad de control – Código VHDL

```
entity controller is
    port (clk, reset, start, continue, stand: in std_logic;
        loser, wrong_card: in std_logic;
        control: out std_logic_vector (4 downto 0));
end controller;

architecture ARCH of controller is

    type T_STATE is (initial, load_addr, read_from_MEM,
load_card_delete_MEM, compare, wrong_card_wait, update_score,
right_card_wait);
    signal STATE, NEXT_STATE: T_STATE;

    signal control_aux: std_logic_vector (4 downto 0);
    alias ld_addr: std_logic is control_aux (0);
    alias write_MEM: std_logic is control_aux (1);
    alias ld_Card: std_logic is control_aux (2);
    alias ld_Score: std_logic is control_aux (3);
    alias reset_i: std_logic is control_aux (4);
```

Unidad de control – Código VHDL

```
begin
    control <= control_aux;

    SYNC_STATE: process (clk, reset, STATE, NEXT_STATE)
    begin
        if reset ='1' then
            STATE <= initial;
        elsif clk'event and clk='1' then
            STATE <= NEXT_STATE;
        end if;
    end process SYNC_STATE;

    COMB: process (STATE, start, continue, stand, loser, wrong_card)
    begin
        ld_addr <= '0';
        write_MEM <= '0';
        ld_Card <= '0';
        ld_Score <= '0';
        reset_i <= '0';

        case STATE is
           when initial => --completar
           --completar resto de estados

        end case;

    end process COMB;
end ARCH;
```

Ruta de datos – Código VHDL

```
entity datapath is
    port (clk_quick, clk_slow, reset: in std_logic;
          control: in std_logic_vector (4 downto 0);
          loser, wrong_card: out std_logic;
          card, score1, score2: out std_logic_vector (6 downto 0));
end datapath;

architecture ARCH of datapath is

    --declaracion de componentes... (completar)

    signal control_aux: std_logic_vector(4 downto 0);
    alias ld_addr: std_logic is control_aux (0);
    alias write_MEM: std_logic is control_aux (1);
    alias ld_Card: std_logic is control_aux (2);
    alias ld_Score: std_logic is control_aux (3);
    alias reset_i: std_logic is control_aux (4);

    signal final_reset, loser_aux, wrong_card_aux: std_logic;
    --declaracion del resto de señales intermedias... (completar)
```

Ruta de datos – Código VHDL

```
begin

    control_aux <= control;
    final_reset <= reset_i OR reset;

    loser_aux <= '1' when <COMPLETAR> else '0';      --completar
    wrong_card_aux <= '1' when <COMPLETAR> else '0'; --completar

    loser <= loser_aux;
    wrong_card <= wrong_card_aux;

    --instanciacion del resto de componentes... (completar)

end ARCH;
```

Práctica 5 – Código VHDL

- Ficheros disponibles en el Campus Virtual:
  - `blackjack.vhd` (fichero "top" en la jerarquía, no modificar).
  - `divider.vhd` (divisor de frecuencias, no modificar).
  - `controller.vhd` (unidad de control, hay que completarla).
  - `datapath.vhd` (ruta de datos, hay que completarla).
  - Resto de componentes que hay que instanciar en la ruta de datos (no modificar):
    - `counter.vhd`
    - `bin2display1.vhd`
    - `bin2display2.vhd`
    - `register_n.vhd`
    - `ram_memory.vhd`

Memoria RAM

- Las cartas estarán almacenadas a partir de la posición 0x00 de una memoria SRAM **síncrona**.
- La memoria será de un solo puerto:
  - Se puede realizar de forma independiente una operación de lectura o una operación de escritura.
- La definición de puertos es:
  - `din`: bus de datos de entrada. 4 bits necesarios (datos de 0 a 10).
  - `addr`: bus de direcciones. 6 bits necesarios (memoria de 64 posiciones).
  - `we`: write enable. `we`='1' indica escritura; `we`='0' indica lectura.
  - `dout`: bus de datos de salida. Idem longitud que `din`. Este puerto conserva el valor leído por última vez hasta que se realice una nueva lectura.

Memoria RAM – Código VHDL

```
entity ram_memory is
    port (clk : in std_logic;
          we : in std_logic;
          addr : in std_logic_vector(5 downto 0);
          din : in std_logic_vector(3 downto 0);
          dout : out std_logic_vector(3 downto 0)
      );
end ram_memory;
```

Memoria RAM – Código VHDL

```
architecture ARCH of ram_memory is
    type ram_type is array (0 to 63) of std_logic_vector (3 downto 0);
    signal RAM : ram_type := (
          X"1", X"2", X"3", X"4", X"5", X"6", X"7", X"8", X"9", X"A", X"A", X"A", X"A",
          X"1", X"2", X"3", X"4", X"5", X"6", X"7", X"8", X"9", X"A", X"A", X"A", X"A",
          X"1", X"2", X"3", X"4", X"5", X"6", X"7", X"8", X"9", X"A", X"A", X"A", X"A",
          X"1", X"2", X"3", X"4", X"5", X"6", X"7", X"8", X"9", X"A", X"A", X"A", X”A",
          X"0", X"0", X"0", X"0", X"0", X"0", X"0", X"0", X"0", X"0", X"0", X"0");
begin

    memory_update: process (clk)
    begin
        if rising_edge(clk) then
            if we = '1' then
                RAM(to_integer(unsigned(addr))) <= din;
            else
                dout <= RAM(to_integer(unsigned(addr)));
            end if;
       end if;
    end process memory_update;
end ARCH;
```

Si copiáis directamente el código del pdf, puede que Xilinx no reconozca las doble comillas

Observaciones

- ATENCIÓN:
  - Los bloques de memoria de Xilinx son **síncronos** tanto para lectura como para escritura:
    - Para leer: la dirección del dato buscado tiene que estar en *addr* un ciclo antes de su utilización.
    - Para escribir: el dato no estará efectivamente escrito en la memoria hasta que no se produzca el siguiente flanco de subida en el reloj (igual que ocurre con los registros).
  - Nuestra unidad de control ha sido diseñada teniendo en cuenta estos dos puntos.