Práctica 6: Memorias

Arnold T. Maldonado

Instituto Politécnico Nacional, Unidad Profesional Interdisciplinaria de Ingeniería campus Zacatecas, Zacatecas, México

tm02arnold@gmail.com

***Resumen*— En esta práctica el objetivo fue diseñar e implementar en VHDL una memoria RAM de 16 direcciones, una memoria ROM de 16 direcciones con un ancho de palabra de 8 bits y una memoria ROM de 8 direcciones con un ancho de palabra de 4 bits. Aquí se plasman lo conceptos básicos y razonamientos que hicieron posible completar esta tarea.**

I. INTRODUCCIÓN

La memoria es la parte de un sistema que almacena datos binarios en grandes cantidades. Las memorias semiconductoras están formadas por matrices de elementos de almacenamiento que pueden ser latches o condensadores. Como regla general, las memorias almacenan datos en unidades que tienen de uno a ocho bits. La unidad menor de datos binarios es, como ya sabemos, el bit. En muchas aplicaciones, se tratan los datos en unidades de 8 bits, denominados bytes o en múltiplos de unidades de 8 bits. El byte se puede dividir en dos unidades de 4 bits, que reciben el nombre de nibbles.

Cada elemento de almacenamiento en una memoria puede almacenar un 1 o un 0 y se denomina celda. Las memorias están formadas por matrices de celdas, cada bloque de la matriz de memoria representa una celda de almacenamiento y su situación se puede especificar mediante una fila y

una columna. La capacidad de una memoria es el número total de unidades de datos que puede almacenar, la cual depende del número de celdas involucradas en la matriz de memoria.

Las dos principales categorías de memorias semiconductoras son las memorias RAM y ROM. La memoria RAM (Random Access Memory, memoria de acceso aleatorio) es un tipo de memoria en la que se tarda lo mismo en acceder a cualquier dirección de memoria y éstas se pueden seleccionar en cualquier orden, tanto en una operación de lectura como de escritura.

Todas las RAM poseen la capacidad de lectura y escritura. Debido a que las memorias RAM pierden los datos almacenados cuando se desconecta la alimentación, reciben el nombre de memorias volátiles.

La memoria ROM (Read Only Memory, memoria de sólo lectura) es un tipo de memoria en la que los datos se almacenan de forma permanente o semipermanente. Los datos se pueden leer de una ROM, pero no existe la operación de escritura como en las RAM. La ROM, al igual que la RAM, es una memoria de acceso aleatorio, pero tradicionalmente el término RAM se reserva para las memorias de acceso aleatorio de lectura/escritura.

II. DESARROLLO DE LA PRÁCTICA

a) Diseño de la RAM de 16 direcciones

Primero se completó el código en VHDL que venía en el guión de la práctica (se muestra en el Apéndice A). Se completó lo siguiente:

1. A la entidad SRAM se le añadieron los valores:
    a. **w: integer:=4** para representar el ancho de palabra.
    b. **d: integer:=4** para representar el número de palabras.
    c. **a: integer:=2** para representar el ancho de dirección.
2. Se declaró la señal **tmp_ram** como de tipo RAM.

Después se cambiaron algunos valores. El valor de **w** es 8 y el de **a** es 2^4 = 16 y el de **d** se mantuvo en 4. El código completo se muestra en el Apéndice B.

b) Diseño de la ROM de 16 direcciones

Al código de las dos primeras RAM se le modificó lo siguiente:

1. El valor de **a** ahora fue 4, para las d = 2^4 = 16 direcciones requeridas.
2. Se quitaron las entradas escribir y **data_in**.
3. Se quitó el proceso escribir.
4. Se declaró la señal **tmp_rom** y se le asignaron los valores que yo elegí (ver Tabla 1).
5. Se añadió el proceso reloj para que visualizar el contenido de la memoria automáticamente.

El código se muestra en el Apéndice C.

Dirección	*En memoria*
0000	10001000
0001	00010001
0010	11001100
0011	00110011
0100	10101010
0101	01010101
0110	01110111
0111	10001000
1000	00100010
1001	11001100
1010	01000100
1011	10001000
1100	00000000
1101	00100010
1110	10001000
1111	00000000

Tabla 1: Organización de la ROM de 16 direcciones

c) Diseño de la ROM de 8 direcciones

Los requerimientos de la memoria fueron:

1. ROM de 8 direcciones con un ancho de palabra de 4 bits.
2. Debía implementar un contador para acceder a las direcciones de memoria.
3. Una entrada será un botón que conectará a la tarjeta donde podrá activar o desactivar el contador
4. Una salida para indicar si el conteo es ascendente o descendente.
5. La frecuencia de reloj debía ser de 1⁄2 Hz. La señal de reloj de la memoria debía ser la misma del contador.

Se usó el código de la memoria ROM anterior, se le modificaron las siguientes cosas:

1. Se añadió el proceso **reloj** que establece la frecuencia del reloj especificada.
2. Se añadió la entrada **funciona** para controlar el cuándo funciona del contador.
3. Se añadió la salida **X** para indicar el funcionamiento del contador

La ROM contiene los valores de las direcciones de la Tabla 2. El código completo se muestra el código en el Apéndice D.

Dirección	*En memoria*
000	1000
001	0001
010	1100
011	0011
100	1010
101	0101
110	0111
111	1000

Tabla 2: Organización de la ROM de 8 direcciones

III. SIMULACIONES

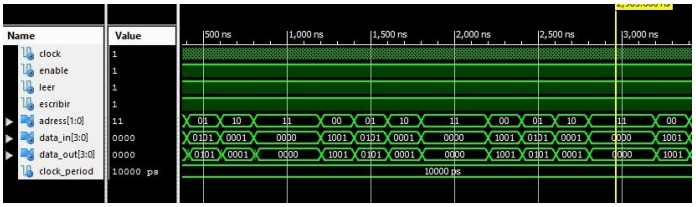

Figura 1: Simulación de la RAM de 4 direcciones

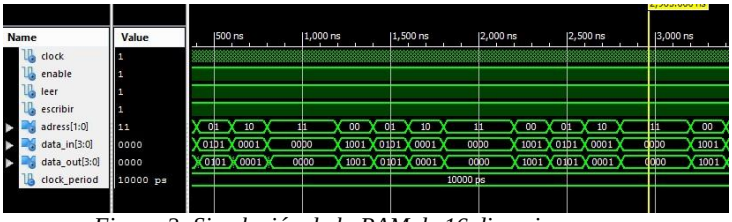

Figura 2: Simulación de la RAM de 16 direcciones

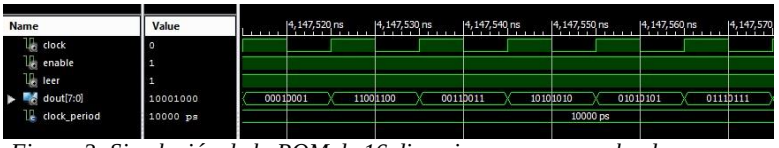

Figura 3: Simulación de la ROM de 16 direcciones con un ancho de palabra de 8 bits

IV. IMPLEMENTACIONES EN EL FPGA

Figura 4: RAM de 16 direcciones

Figura 5: ROM de 16 direcciones

Figura 6: ROM de 8x4 con el contador ascendente

Figura 7: ROM de 8x4 con el contador descendente

V. CONCLUSIONES

Dado que las memorias RAM tienen tantos usos resultó importante para mí el conocer cómo funciona y cómo se organiza una, así como verla en acción usando los LED de FPGA.

Aquí se ven reflejados la gran mayoría los conceptos que vimos en el curso y me agradó poder diseñar las memorias usándolos.

APÉNDICE A

CÓDIGO DE LA RAM DE 4 DIRECCIONES

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

--Bloque generic: Cumple las mismas funciones
--las variables que se declaran que las constants
ENTITY SRAM IS
    GENERIC
    (
        w : INTEGER := 4; --ancho de palabra
        d : INTEGER := 4; --n° de palabras
    a : INTEGER := 2); --ancho de dirección

    PORT
    (
        Clock : IN std_logic;
        Enable : IN std_logic;
        leer : IN std_logic;
        escribir : IN std_logic;
        adress : IN std_logic_vector (a - 1 DOWNTO 0);
        Data_in : IN std_logic_vector (w - 1 DOWNTO 0);
        Data_out : OUT std_logic_vector (w - 1 DOWNTO 0)
    );
END SRAM;
```

Figura 8: RAM con 4 direcciones (1/2)

```
ARCHITECTURE memoria_arc OF SRAM IS

    --Array para guardar los valores de memoria
    TYPE ram_type IS ARRAY (0 TO d - 1) OF std_logic_vector(w - 1 DOWNTO 0);
    SIGNAL tmp_ram : ram_type;

BEGIN
    --Declarando los ciclos de lectura y de escritura

    --lectura
    PROCESS (leer, Clock)
    BEGIN
        IF (Clock'EVENT AND Clock = '1') THEN
            IF (Enable = '1') THEN
                IF (leer = '1') THEN
                    Data_out <= tmp_ram(conv_integer(adress));
                ELSE
                    Data_out <= (Data_out'RANGE => 'Z');
                END IF;
            END IF;
        END IF;
    END PROCESS;

    --escritura
    PROCESS (escribir, Clock)
        BEGIN
            IF (Clock'EVENT AND Clock = '1') THEN
                IF (Enable = '1') THEN
                    IF (escribir = '1') THEN
                        tmp_ram(conv_integer(adress)) <= Data_in;
                    END IF;
                END IF;
            END IF;
        END PROCESS;

END memoria_arc;
```

Figura 9: RAM con 4 direcciones (2/2)

APÉNDICE B

CÓDIGO DE LA RAM DE 16 DIRECCIONES

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

--Bloque generic: Cumple las mismas funciones las variables que
--se declaran que las constants
ENTITY SRAM IS
    GENERIC
    (
        w : INTEGER := 8; --ancho de palabra
        d : INTEGER := 4; --n° de palabras
    a : INTEGER := 4); --ancho de dirección
    --
    PORT
    (
        Clock : IN std_logic;
        Enable : IN std_logic;
        leer : IN std_logic;
        escribir : IN std_logic;
        adress : IN std_logic_vector (a - 1 DOWNTO 0);
        Data_in : IN std_logic_vector (w - 1 DOWNTO 0);
        Data_out : OUT std_logic_vector (w - 1 DOWNTO 0)
    );
END SRAM;

ARCHITECTURE memoria_arc OF SRAM IS

    --Array para guardar los valores de memoria
    TYPE ram_type IS ARRAY (0 TO d - 1) OF std_logic_vector(w - 1 DOWNTO 0);
    SIGNAL tmp_ram : ram_type;

BEGIN
```

Figura 10: RAM con 16 direcciones (1/2)

```
    --Declarando los ciclos de lectura y de escritura

    --lectura
    PROCESS (leer, Clock)
    BEGIN
        IF (Clock'EVENT AND Clock = '1') THEN
            IF (Enable = '1') THEN
                IF (leer = '1') THEN
                    Data_out <= tmp_ram(conv_integer(adress));
                ELSE
                    Data_out <= (Data_out'RANGE => 'Z');
                END IF;
            END IF;
        END IF;
    END PROCESS;

    --escritura
    PROCESS (escribir, Clock)
        BEGIN
            IF (Clock'EVENT AND Clock = '1') THEN
                IF (Enable = '1') THEN
                    IF (escribir = '1') THEN
                        tmp_ram(conv_integer(adress)) <= Data_in;
                    END IF;
                END IF;
            END IF;
        END PROCESS;

END memoria_arc;
```

Figura 11: RAM con 16 direcciones (2/2)

APÉNDICE C

CÓDIGO DE LA ROM DE 16 DIRECCIONES

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY ROM IS
    GENERIC
    (
        w : INTEGER := 8; --ancho de palabra
        d : INTEGER := 16; --n° de palabras
    a : INTEGER := 4); --ancho de dirección

    PORT
    (
        Clock : IN std_logic;
        Enable : IN std_logic;
        Leer : IN std_logic;
        dout : OUT std_logic_vector(w - 1 DOWNTO 0)
    );
END ENTITY ROM;
ARCHITECTURE ROM_arch OF ROM IS
    SIGNAL cuenta : INTEGER := 0;
    SIGNAL direccion : std_logic_vector (a - 1 DOWNTO 0);
    TYPE MEMORY_16_8 IS ARRAY (0 TO d - 1) OF std_logic_vector(w - 1 DOWNTO 0);
    CONSTANT ROM_16_8 : MEMORY_16_8 := ("10001000", "00010001", "11001100", "00110011",
        "10101010", "01010101", "01110111", "10001000",
        "00100010", "11001100", "01000100", "10001000",
        "00000000", "00100010", "10001000", "00000000"
    );
BEGIN
```

Figura 12: ROM con 16 direcciones (1/4)

```
    --lectura
    PROCESS (Clock)
    BEGIN
        IF (Clock'EVENT AND Clock = '1') THEN
            dout <= ROM_16_8(to_integer(unsigned(direccion)));
        END IF;
    END PROCESS;

    reloj : PROCESS (Leer, Enable, direccion, Clock)
    BEGIN
        IF (Enable <= '1') THEN
            IF (Leer <= '1') THEN
```

Figura 13: ROM con 16 direcciones (2/4)

```
                IF (Clock'EVENT AND Clock = '1') THEN
                    CASE (cuenta) IS
                        WHEN 0 =>
                            direccion <= "0000";
                            cuenta <= cuenta + 1;
                        WHEN 1 =>
                            direccion <= "0001";
                            cuenta <= cuenta + 1;
                        WHEN 2 =>
                            direccion <= "0010";
                            cuenta <= cuenta + 1;
                        WHEN 3 =>
                            direccion <= "0011";
                            cuenta <= cuenta + 1;
                        WHEN 4 =>
                            direccion <= "0100";
                            cuenta <= cuenta + 1;
                        WHEN 5 =>
                            direccion <= "0101";
                            cuenta <= cuenta + 1;
                        WHEN 6 =>
                            direccion <= "0110";
                            cuenta <= cuenta + 1;
                        WHEN 7 =>
                            direccion <= "0111";
                            cuenta <= cuenta + 1;
                        WHEN 8 =>
                            direccion <= "1000";
                            cuenta <= cuenta + 1;
                        WHEN 9 =>
                            direccion <= "1001";
                            cuenta <= cuenta + 1;
                        WHEN 10 =>
                            direccion <= "1010";
                            cuenta <= cuenta + 1;
                        WHEN 11 =>
                            direccion <= "1011";
                            cuenta <= cuenta + 1;
                        WHEN 12 =>
                            direccion <= "1100";
                            cuenta <= cuenta + 1;
```

Figura 14: ROM con 16 direcciones (3/4)

```
                        WHEN 13 =>
                            direccion <= "1101";
                            cuenta <= cuenta + 1;
                        WHEN 14 =>
                            direccion <= "1110";
                            cuenta <= cuenta + 1;
                        WHEN 15 =>
                            direccion <= "1111";
                            cuenta <= cuenta + 1;
                        WHEN OTHERS =>
                            cuenta <= 0;
                    END CASE;
                END IF;
            END IF;
        END IF;
    END PROCESS;
END ARCHITECTURE ROM_arch;
```

Figura 15: ROM con 16 direcciones (4/4)

APÉNDICE D
CÓDIGO DE LA ROM DE 8 DIRECCIONES

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY ROM IS
    GENERIC (
        w : INTEGER := 4; --ancho de palabra
        d : INTEGER := 8; --n° de palabras
    a : INTEGER := 3); --ancho de dirección
    PORT (
        Clock : IN std_logic;
        Enable : IN std_logic;
        Leer : IN std_logic;
        funciona : IN std_logic;
        ad : IN std_logic; -- ascendente si ad = 1
        X : OUT std_logic;
        dout : OUT std_logic_vector(w - 1 DOWNTO 0)
    );
END ENTITY ROM;
ARCHITECTURE ROM_arch OF ROM IS
    CONSTANT max_count : INTEGER := 5000000;
    SIGNAL count : INTEGER RANGE 0 TO max_count;
    SIGNAL clk_state : std_logic := '0';
    SIGNAL clk : std_logic;
    SIGNAL cuenta : INTEGER := 0;
    SIGNAL direccion : std_logic_vector (a - 1 DOWNTO 0);
    TYPE MEMORY_8_4 IS ARRAY (0 TO d - 1) OF std_logic_vector(w - 1 DOWNTO 0);
    CONSTANT ROM_8_4 : MEMORY_8_4 := ("1000", "0001", "1100", "0011",
        "1010", "0101", "0111", "1000"
    );
BEGIN
    --lectura
    PROCESS (clk)
    BEGIN
        IF (clk'EVENT AND clk = '1') THEN
            dout <= ROM_8_4(to_integer(unsigned(direccion)));
        END IF;
    END PROCESS;
```

Figura 16: ROM con 8 direcciones (1/5)

```
    PROCESS (funciona, Leer, Enable, direccion, clk)
        BEGIN
            IF (funciona <= '1') THEN
                IF (Enable <= '1') THEN
                    IF (Leer <= '1') THEN
                        IF (clk'EVENT AND clk = '1') THEN
                            IF (ad <= '1') THEN
                                CASE (cuenta) IS
```

Figura 17: ROM con 8 direcciones (2/5)

```
                WHEN 0 =>
                    direccion <= "000";
                    cuenta <= cuenta + 1;
                    X <= '1';
                WHEN 1 =>
                    direccion <= "001";
                    cuenta <= cuenta + 1;
                    X <= '1';
                WHEN 2 =>
                    direccion <= "010";
                    cuenta <= cuenta + 1;
                    X <= '1';
                WHEN 3 =>
                    direccion <= "011";
                    cuenta <= cuenta + 1;
                    X <= '1';
                WHEN 4 =>
                    direccion <= "100";
                    cuenta <= cuenta + 1;
                    X <= '1';
                WHEN 5 =>
                    direccion <= "101";
                    cuenta <= cuenta + 1;
                    X <= '1';
                WHEN 6 =>
                    direccion <= "110";
                    cuenta <= cuenta + 1;
                    X <= '1';
                WHEN 7 =>
                    direccion <= "111";
                    cuenta <= cuenta + 1;
                    X <= '1';
```

Figura 18: ROM con 8 direcciones (3/5)

```
                WHEN OTHERS =>
                    cuenta <= 0;
                    X <= '1';
            END CASE;
        END IF;
    ELSIF (ad <= '0') THEN
        CASE (cuenta) IS
            WHEN 0 =>
                direccion <= "111";
                cuenta <= cuenta - 1;
                X <= '0';
            WHEN 1 =>
                direccion <= "110";
                cuenta <= cuenta - 1;
                X <= '0';
            WHEN 2 =>
                direccion <= "101";
                cuenta <= cuenta - 1;
                X <= '0';
            WHEN 3 =>
                direccion <= "100";
                cuenta <= cuenta - 1;
                X <= '0';
            WHEN 4 =>
                direccion <= "011";
                cuenta <= cuenta - 1;
                X <= '0';
            WHEN 5 =>
                direccion <= "010";
                cuenta <= cuenta - 1;
                X <= '0';
            WHEN 6 =>
                direccion <= "001";
                cuenta <= cuenta - 1;
                X <= '0';
            WHEN 7 =>
                direccion <= "000";
                cuenta <= cuenta - 1;
                X <= '0';
            WHEN OTHERS =>
                cuenta <= 0;
```

Figura 19: ROM con 8 direcciones (4/5)

```
                        X <= '0';
                    END CASE;
                END IF;
            END IF;
        END IF;
    END PROCESS;

    gen_clock : PROCESS (Clock, count)
    BEGIN
        IF Clock' event AND Clock = '1' THEN
            IF count < max_count THEN
                count <= count + 1;
            ELSE
                clk_state <= NOT clk_state;
                count <= 0;
            END IF;
        END IF;
    END PROCESS;

    asignacion : PROCESS (clk_state)
    BEGIN
        clk <= clk_state;
    END PROCESS;

END ARCHITECTURE ROM_arch;
```

Figura 20: ROM con 8 direcciones (5/5)

REFERENCIAS

[1] M. M. Morris, *Diseño Digital* (3ª Ed.). Naucalpan de Juárez: Pearson Educación, 2003.

[2] T. L Floyd, *Fundamentos de Sistemas* Digitales (9ª Ed.). Pearson Educación, 2007.

[3] D. D. Gajski y P. G. Carlos, *Principios de Diseño Digital*. Prentice Hall, 2000.