PRÁCTICA 4. Memoria RAM **FECHA: _____**

GRUPO: _____

ALUMNOS: _____

_____

OBJETIVO El alumno comprenderá el funcionamiento y manejo de las memorias en VHDL.

INTRODUCCION

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**. Una unidad completa de información se denomina **palabra** y está formada, generalmente, por uno o más bytes. Algunas memorias almacenan datos en grupos de 9 bits: un grupo de 9 bits consta de un byte más un bit de paridad.

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, si por ejemplo, la memoria está organizada en una matriz de 64 celdas, la capacidad total será de 64 bits.

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.

DESARROLLO

En esta práctica se diseñará una memoria rom y una memoria ram empleando los arreglos en VHDL para ello. Recordemos que los arreglos son una colección de elementos del mismo tipo a los que se accede mediante un índice. Su significado y uso no difiere mucho de la misma estructura presente en casi todos los lenguajes de programación.

La sintaxis para declarar un tipo de arreglo es la siguiente:

```
TYPE nombre_tipo IS ARRAY (especificacion) OF data_type;
```

- Donde nombre_tipo. Nombre para identificar el arreglo
- Especificacion. Es el tamaño de la dirección de la memoria
- Data_type. Es el tamaño de bus de datos

En el empleo de las distintas memorias es muy común el uso de constantes, que son un elemento que se inicializa en un determinado valor que no puede ser cambiado una vez inicializado, sino que se conserva para siempre. Un ejemplo se muestra a continuación:

```
Constant  e: real := 2.71;
```

También al trabajar con memorias se utilizará la conversión en el tipo de datos para la dirección ya que esta indicará la localidad de memoria a la cual se desea acceder; para ello se muestra a continuación la tabla de conversión entre los tipos de datos:

Tipo origen (a)	Tipo destino	Función de conversión / Type casting
unsigned, signed	std_logic_vector	std_logic_vector(a)
signed, std_logic_vector	unsigned	unsigned(a)
unsigned, std_logic_vector	signed	signed(a)
unsigned, signed	integer	to_integer(a)
natural	unsigned	to_unsigned(a, size)
integer	signed	to_signed(a, size)

Realizaremos una memoria RAM con 4 direcciones cuyo tamaño es de 4 bits. Debe tener las siguientes características:

- Su entrada de reloj.
- Una entrada de activación (enable).
- Una entrada de lectura.
- Una entrada de escritura.
- 2 entradas de dirección.
- 4 entradas de datos.
- 4 salidas de datos.

Para implementar la memoria ram se deberá agregar las siguientes librerías debido a que al trabajar con los arreglos llama algunas funciones que no se encuentran en la librería use ieee.std_logic_1164.all.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5  -- El paquete debe contener la función que pasa de std_logic_vector a
6  -- natural, para poder acceder al array
7
```

Cuando realizamos algún tipo de memoria podemos en la entidad (entity) un bloque generic, el cual cumple las mismas funciones las variables que se declaran que las constants.

```
7
8  entity SRAM is
9     generic( w: integer:=4; -- ancho de palabra
10             d: integer:=4; -- nº de palabras
11             a: integer:=2); -- ancho dirección
```

Para declarar el arreglo de la memoria se hace de la siguiente manera.

```
23 -- Utilizamos un array para guardar los valores de la memoria
24 type ram_type is array (0 to d-1) of std_logic_vector(w-1 downto 0);
25 signal tmp_ram: ram_type;
```

Una vez que ya se tenga establecida hay que declarar los ciclos de lectura y de escritura. Para cada uno de estos ciclos es necesario activarlo en un flanco de subida y que la entrada eneable este activa. Dependiendo de la entrada que se tenga (escritura o lectura) hará la función. A continuación, se muestra en VHDL el código.

```
27  -- Lectura
28  process(Clock, leer)
29  begin
30     if (Clock'event and Clock='1') then
31        if Enable='1' then
32           if leer='1' then
33              Data_out <= tmp_ram(conv_integer(adress));
34           else
35        Data_out <= (Data_out'range => 'Z');
36  -- Todos los bits de Data_out se ponen a 'Z'
37           end if;
38        end if;
39     end if;
40  end process;
```

```
41  -- Escritura
42  process(Clock, escribir)
43  begin
44     if (Clock'event and Clock='1') then
45        if Enable='1' then
46           if escribir='1' then
47              tmp_ram(conv_integer(adress)) <= Data_in;
48           end if;
49        end if;
50     end if;
```

INSTRUCCIONES

1. Una vez implementado el mismo código, modifíquelo para aumentar las direcciones a 16, para crear una memoria RAM de 4x4.
2. Realice una memoria ROM de de 16 direcciones con un ancho de palabra de 8 bits. Los datos que contendrá la memoria ROM podrá llenarlos como usted guste. Recuerde que debe llevar su señal de reloj.
3. Realice una memoria rom de 8 direcciones con un ancho de palabra de 4 bits. En este deberá implementar un contador para acceder a las direcciones de memoria. Para ello deberá conectar dos señales adicionales a su caja negra, una entrada será un botón que conectará a la tarjeta donde podrá activar o desactivar dicho contador, la segunda señal se utilizará para indicar si el conteo es ascendente o descendente. La frecuencia de reloj debe ser de ½ Hz. La señal de reloj de la memoria será la misma del contador.

NOTA: Registre todos los datos que considere necesarios para el desarrollo del reporte