Práctica No. 7.
Memorias.
Memories.

Haz que lo Bueno sea Mejor y lo Mejor Excelente

Unidad Temática: IV

Duración y entrega: indicadas en la plataforma.

En esta práctica usted aprenderá a implementar/simular distintas aplicaciones con circuitos de memoria en el FPGA de la tarjeta de desarrollo, como se lista a continuación:

- Implementar en el FPGA aplicaciones que utilizan circuitos de memoria para controlar un display matricial y un LCD.

In this practice you will learn to implement/simulate different applications using memory circuits in the FPGA development board, as listed below:

- *Implement in the FPGA applications using memory circuits to control a matrix display and LCD.*

Introducción.
Introduction.

Memorias.

El almacenamiento temporal o permanente de datos es una de las características más importantes de los sistemas secuenciales desarrollados, ya que permiten conservar valores o datos nuevos o estados anteriores y tenerlos disponibles para cuando se necesiten. Las Memorias digitales o simplemente memorias son los elementos que guardan información de forma temporal para procesar.

Existen diferentes formas de clasificar a las memorias, por el tipo de funcionamiento, tecnología de fabricación, por la capacidad de almacenamiento, etc., siendo la más común la que se muestra a continuación (ver un ejemplo en la figura 7.1):

- RAM (random-access-memory) dinámica o estática
- ROM (read-only-memory)
- PROM (programmable- read-only-memory)
- EPROM (erasable-programmable- read-only-memory)
- EEPROM (electrically-erasable-programmable- read-only-memory)
- FLASH (≈EEPROM)

Una memoria muy utilizada últimamente es la memoria FLASH, la cual es encontrada en las memorias USB por su gran capacidad y reducido tamaño y consumo de energía.

Todas las memorias contienen terminales de control (bus de control, selección del chip, dirección o address, etc.) y terminales de manejo de datos (bus de datos), que se utilizan para gobernar a dicho elemento, siendo la forma típica de encontrarlas como se muestra en la tabla 7.1 y 7.2.

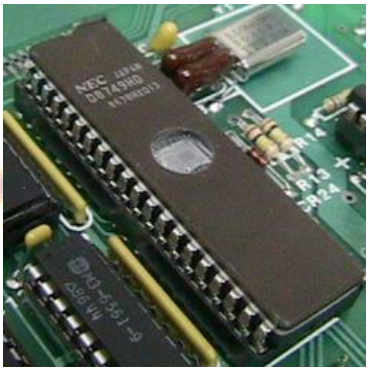

Figura 7.1. Ejemplo de una memoria EPROM borrable por luz ultravioleta.

Tabla 7.1. Nombres comunes de las terminales o señales en las memorias.

A0-A1	Bus de Direcciones (Address Bus) Input
D0-D3	Bus de Datos (Data Bus) Input/Output
$\overline{WE}$,WE’	Habilitador de escritura (Write Enable)
$\overline{CE}$, CE’	Habilitador del chip (Chip Enable)
$\overline{OE}$, OE’	Habilitador de salida (Output Enable)
$R/\overline{B}$, R/B’	Listo/Ocupado’ (Ready/Busy’)
V_{CC}	Alimentación +V
V_{SS}	Alimentación GND

Nota: Cuando en la memoria se está leyendo o escribiendo un dato se dice que está ocupada y RB’ daría un cero, de lo contrario, la memoria está lista para hacer alguna operación.

Tabla 7.2. Modos de operación de las memorias.

Modo	$\overline{CE}$	$\overline{OE}$	$\overline{WE}$	D0-D3
Chip deshabilitado (Standby)	1	X	X	Alta Impedancia Z
Salida deshabilitada (Output Disable)	0	1	X	Alta Impedancia Z
Leer datos (Read)	0	0	1	Entran datos a la memoria (Data Input)
Escribir datos (Write)	0	1	0	Salen datos de la memoria (Data Output)

Nota: 1 = V_{IH} = voltaje de entrada en alto, 0 = V_{IL}, X = V_{IH}, o V_{IL}.

A continuación, se presenta un en la figura 7.2 un diagrama de tiempos donde se muestran los ciclos de lectura y escritura para una memoria comercial. Cuando se lee (RD=0) el bus de datos es entrada, cuando se escribe (WR=0) el bus de datos es salida.

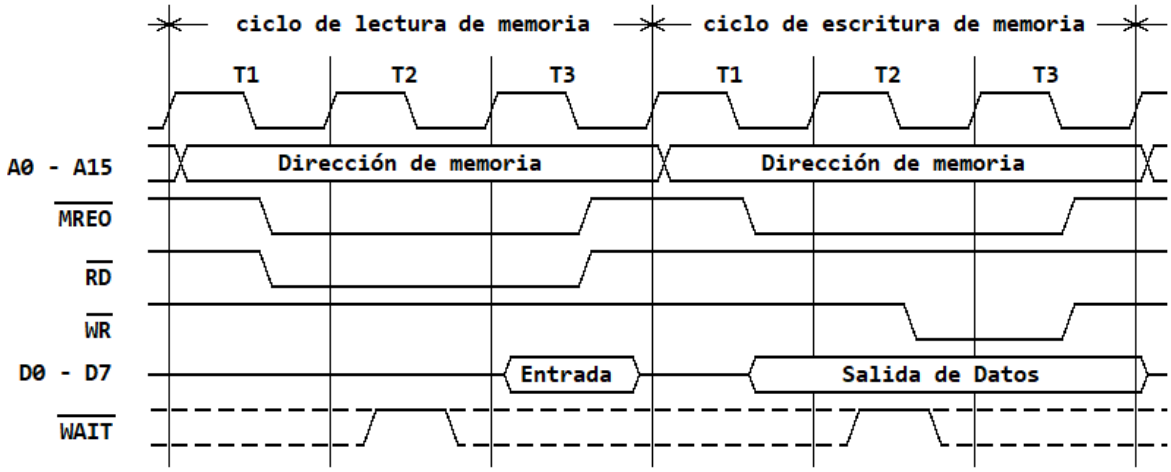

Figura 7.2. Ciclos de lectura y escritura de una memoria.

Las aplicaciones más comunes de las memorias se encuentran en los equipos como:

-- Computadoras	-- Impresoras	-- Cámaras digitales
-- Escáner	-- Osciloscopios digitales	-- Estéreos
-- Ultrasonidos	-- Visualizadores	-- Teléfonos

por mencionar solo algunas. Y sea cual fuere la aplicación, se recomienda consultar los manuales de los fabricantes para tener un panorama más amplio de los tipos y capacidades de memorias existentes en el mercado.

A continuación, se muestran unos ejemplos para una memoria RAM y una ROM junto con sus simulaciones en las figuras 7.3 a 7.7.

VHDL

```
--código de una memoria RAM de 16 bits
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity RAM IS
generic (bits : integer:=4);                              --tamaño del bus de datos
port (DataIN     : in std_logic_vector (bits-1 downto 0); --bus de entrada
      Address    : in std_logic_vector (1 downto 0);      --bus de direcciones
      DataOUT    : out std_logic_vector (bits-1 downto 0); --bus de salida
      CS         : in std_logic;                          --chip select
      RW         : in std_logic);                         --control de lectura/escritura
```

```
end RAM;

architecture Arq_RAM of RAM is
type mem is array (3 downto 0) of std_logic_vector (bits-1 downto 0);
signal memory : mem;
signal addr: integer range 0 to 3;
begin

convertir: process (address)
    begin
      addr <= conv_integer(address); --convierte a entero el dato binario "address"
end process convertir;

asigna: process(CS,RW,datain,address,memory)
    begin
if CS'event and CS='1' then            --chip select se habilita con flancos
        if RW='0' then memory(addr) <= datain;
--      con addr=0 asigna datain a memory(0), con addr=1 asigna datain a memory(1),
--      con addr=2 asigna datain a memory(2), con addr=3 asigna datain a memory(3).
        elsif RW='1' then dataout <= memory(addr);
--      con addr=0 manda memory(0) a dataout, con addr=1 manda memory(1) a dataout,
--      con addr=2 manda memory(2) a dataout, con addr=3 manda memory(3) a dataout.
        else dataout <= "ZZZZ";
        end if;
end if;
end process asigna;
end Arq_RAM; --código
```

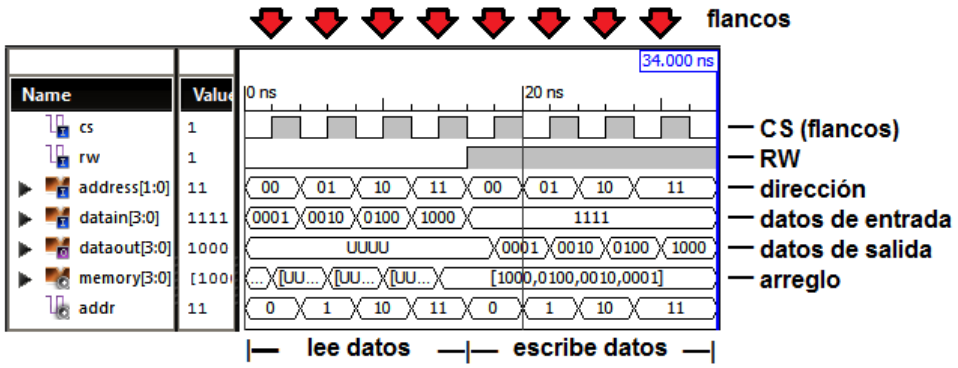

Figura 7.3. Simulación de una RAM, habilitando a CS con flancos, RW=’0’ para que la memoria lea un dato (guardar dato) y RW=’1’ para que la memoria escriba un dato (pase al dato a la salida).

Verilog

```
//Memoria RAM funcionando OK
module ram (
clk        , // Reloj o clock
```

```
address     , // Dirección o Address
data        , // Dato bi-directional
cs          , // Selección del chip o Chip Select
we          , // lectura/escritura Read/Write
oe            // Habilitador de salida
);

parameter DATA_BITS = 4 ; //número de bits del dato
parameter ADDR_WIDTH = 2 ; //número de bits de dirección
parameter SIZE_RAM = 1 << ADDR_WIDTH;//dobla el tamaño de la dirección

// entradas
input               clk         ;
input [ADDR_WIDTH-1:0] address     ;
input               cs          ;
input               oe          ;
input               we          ;

//salidas
inout [DATA_BITS-1:0]  data       ;

//registros
reg [DATA_BITS-1:0] data_out ; //registro de 8 bits
reg [DATA_BITS-1:0] mem [0:SIZE_RAM-1]; //matríz de datos
reg                 oe_r; //habilitador de la salida

// salida triestado
// cuando cs = 1, oe = 1, we = 0 y oe_r = 1
assign data = (cs && oe && !we && oe_r) ? data_out : 4'bz;

// Memoria
always @ (posedge clk)
begin : MEMORIA_RAM
// cuando cs=0 la salida es alta impedancia
  if ( cs == 0 ) begin data_out = 4'bz; end
// cuando está habilitado cs y we se escribe en la memoria
        else if (we) begin mem[address] = data; end
// cuando cs=1, oe=1 y we =0 se lee la memoria
        else if (oe) begin data_out = mem[address]; oe_r = 1; end
        else begin oe_r = 0; end
end //termina el always

endmodule //fin del modulo
```

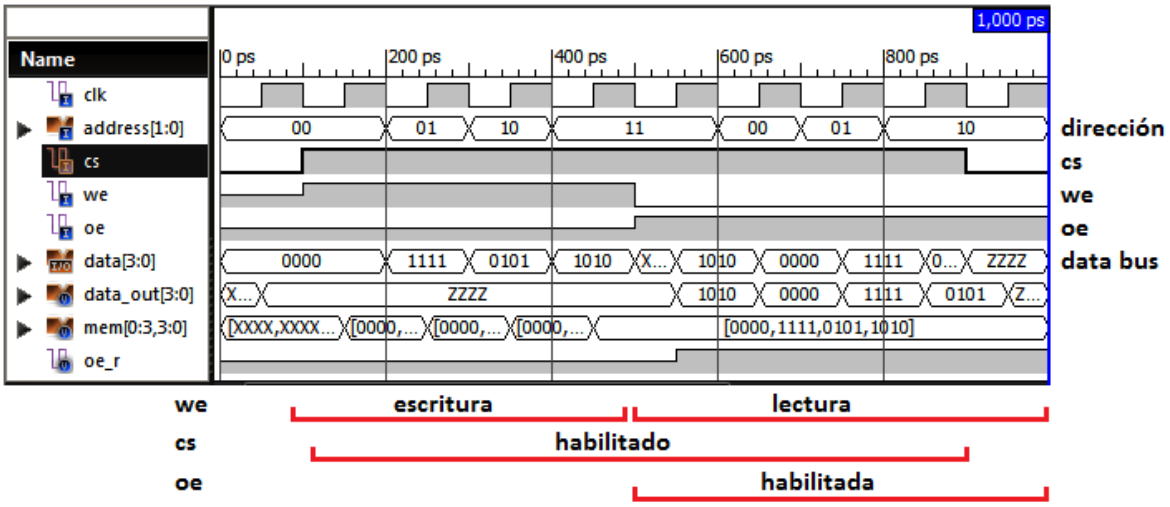

Figura 7.4. Simulación de una RAM, con reloj, chip selec “cs”, write enable “we”, y output enable “oe”.

VHDL

```
--código de una ROM de 16x8 (128bits) donde se guardan
--los números hexadecimales para un display de 7 segmentos
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity rom is
generic (bits: integer:=8);
port(address: in std_logic_vector(3 downto 0);
dataout: out std_ulogic_vector(bits-1 downto 0));
end entity rom;

architecture mem of rom is
type rom_array is array (0 to 15) of std_ulogic_vector (bits-1 downto 0);
-- los datos de los segmentos del display tienen el orden "abcdefgP" para ánodo común
constant rom: rom_array := (
 "00000001", "10011111", "00100101", "00001101",        -- 0, 1, 2, 3
 "10011001", "01001001", "01000001", "00011111",        -- 4, 5, 6, 7
 "00000001", "00001001", "00010001", "11000001",        -- 8, 9, A, b
 "01100011", "10000101", "01100001", "01110001");       -- C, d, E, F
signal addr: integer range 0 to 15;
begin
addr <= conv_integer(address); -- convierte el dato binario de address a entero
dataout <= rom(addr);
end architecture;
```

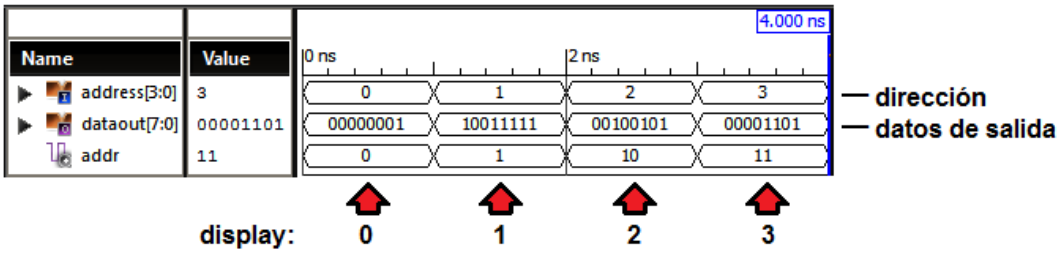

Figura 7.5. Simulación de una ROM, mostrando los primeros 4 datos. En cada dirección se obtiene un número para visualizarlo en un display de ánodo común.

Verilog

```
// memoria ROM de 16 bits
// 4 datos (2 bits de dirección) de 4 bits
// con datos cargados por archivo de texto
module ROM_4x4 (ROM_data, ROM_addr);
 output [3:0] ROM_data;
 input [1:0] ROM_addr;
 reg [3:0] ROM [3:0]; // 4 registros de 4 bits

// Carga el contenido de la rom desde el archivo ROM_data.txt
// guardado en el mismo directorio del proyecto ISE
 initial $readmemb ("ROM_data.txt", ROM, 0, 3); //
// lectura de la memoria ROM a partir de cada dirección dada
 assign ROM_data = ROM[ROM_addr];
endmodule
```

Archivo de texto: ROM_data.txt

```
1000
0100
0001
0010
```

La simulación de estos archivos se muestra en la siguiente figura 7.7.

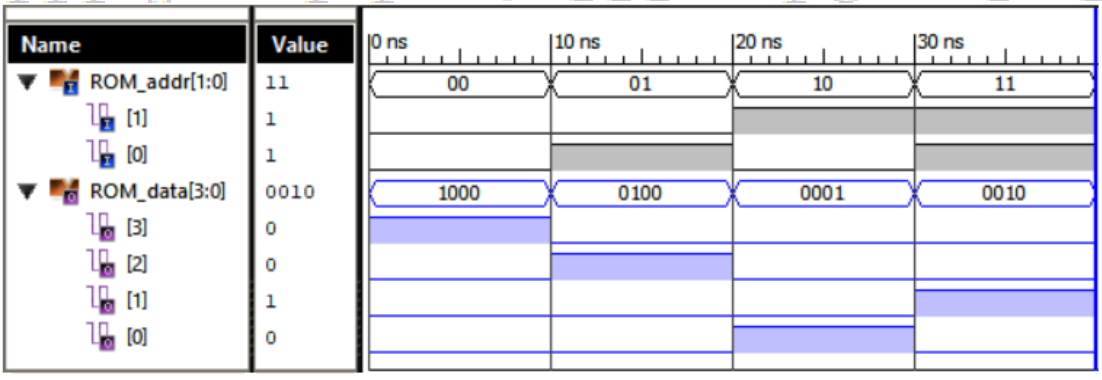

Figura 7.6. Simulación de la memoria ROM.

Verilog

```
//memoria ROM de 16 bits
//4 datos (2 bits de dirección) de 4 bits
//con datos cargados por combinatorio en case
module ROM_4x4 (ROM_data, ROM_addr);
 output reg [3:0] ROM_data;
 input [1:0] ROM_addr;

// asignación de los datos en base a la dirección en la ROM
always @ (ROM_addr)
        begin
        case (ROM_addr)
                0: ROM_data <= 4'b1000;
                1: ROM_data <= 4'b0100;
                2: ROM_data <= 4'b0001;
                3: ROM_data <= 4'b0010;
                default ROM_data <= 4'bxxxx;
        endcase
        end
endmodule
```

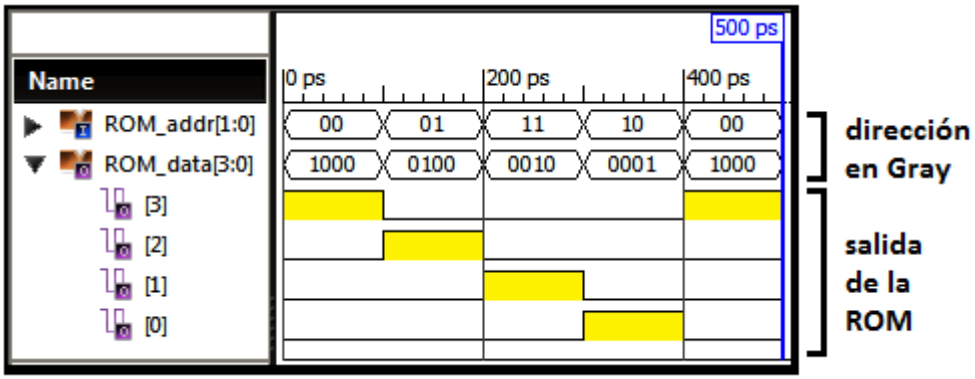

Figura 7.7. Simulación de la memoria ROM de 4 datos de 4 bits. Ambos códigos anteriores dan el mismo resultado.

Ejemplos de declaraciones en VHDL y Verilog para implementar arreglos y constantes:

VHDL: arreglo para display de ánodo común (AC)

```
type tabla is array (0 to 3) of std_logic_vector(7 downto 0);
constant datos_tabla : tabla :=          -- datos p/display ánodo común
(  "01001000",                           -- H = 48h
   "00000001",                           -- O = 01h
   "01110001",                           -- L = 71h
   "00001000"  );                        -- A = 08h
```

VHDL: constantes para display de ánodo común (AC)

```
        constant h:     std_logic_vector (7 downto 0) := "01001000",        -- H = 48h
```

```
constant o:     std_logic_vector (7 downto 0) := "00000001",     -- O = 01h
constant l:     std_logic_vector (7 downto 0) := "01110001",     -- L = 71h
constant a:     std_logic_vector (7 downto 0) := "00001000",     -- A = 08h
```

Verilog: constantes para display de ánodo común (AC)

```
wire [7:0] H, O, L, A;
assign H = 8'b01001000;
assign O = 8'b00000001;
assign L = 8'b01110001;
assign A = 8'b00001000;
```

VHDL: arreglo para display de cátodo común (CC)

```
type tabla is array (0 to 3) of std_logic_vector(7 downto 0);
constant datos_tabla : tabla :=          -- datos p/display cátodo común
(  "00110111",                           -- H = 37h
   "01111110",                           -- O = 7Eh
   "00001110",                           -- L = 0Eh
   "01110111"  );                        -- A = 77h
```

VHDL: constantes para display de cátodo común (CC)

```
constant h:     std_logic_vector (7 downto 0) := "00110111",     -- H = 37h
constant o:     std_logic_vector (7 downto 0) := "01111110",     -- O = 7Eh
constant l:     std_logic_vector (7 downto 0) := "00001110",     -- L = 0Eh
constant a:     std_logic_vector (7 downto 0) := "01110111",     -- A = 77h
```

Verilog: constantes para display de cátodo común (CC)

```
wire [7:0] H, O, L, A;
assign H = 8'b00110111;
assign O = 8'b01111110;
assign L = 8'b00001110;
assign A = 8'b01110111;
```

Matricial.

Los circuitos de manejo matricial se emplean para disminuir los terminales de conexión, por ejemplo, un elevador con 100 pisos necesitaría 100 hilos de control, mientras que con un diseño matricial de 10 columnas por 10 renglones se tienen los mismos 100 pisos, pero solo con 20 hilos de conexión. Los arreglos matriciales se utilizan tanto en elementos de entrada como en los teclados, como en los elementos de salida, displays y pantallas.

Dentro de los displays matriciales existen 2 tipos, los de ánodo y los de cátodo (Figura 7.8), esto se refiere a los terminales de la matriz de leds, que se encuentran conectados a los renglones, cuyo control típico es mandar datos en los renglones y barrer las columnas con un contador de anillo o mandar datos a las columnas y barrer los renglones.

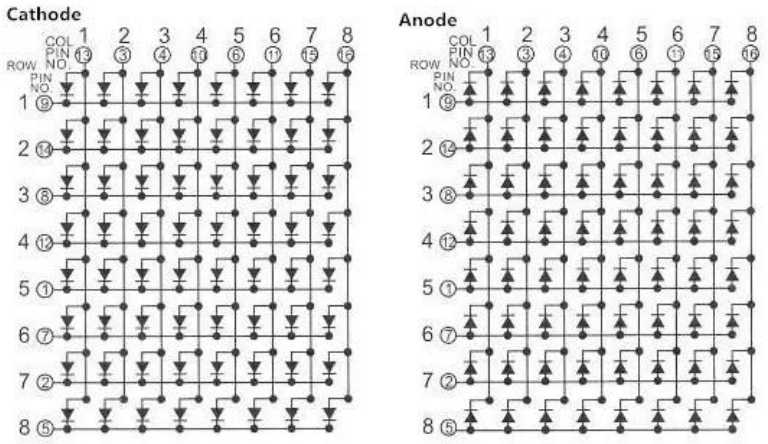

Figura 7.8. Display 8x8 cátodo (izquierda) y ánodo (derecha).

Una forma de conectarlos es directamente a los terminales del FPGA, lo cual drenará la corriente del mismo dispositivo. Otra forma es con el uso de transistores, lo que permite el uso de fuente externa, resaltando que los renglones (ROW) en el display de cátodo son del tipo NPN y para los de ánodo son del tipo PNP, como se muestra en la figura 7.9. En la figura 7.10 se muestra físicamente el de cátodo en los renglones.

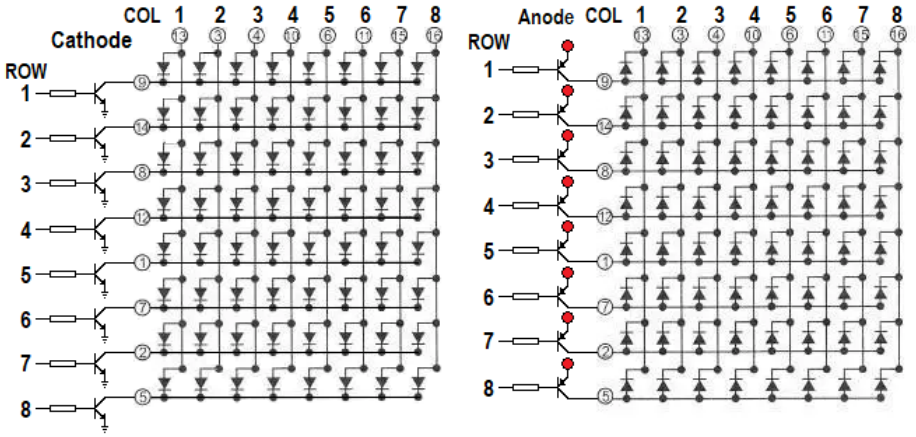

Figura 7.9. Conexión del display con transistores.

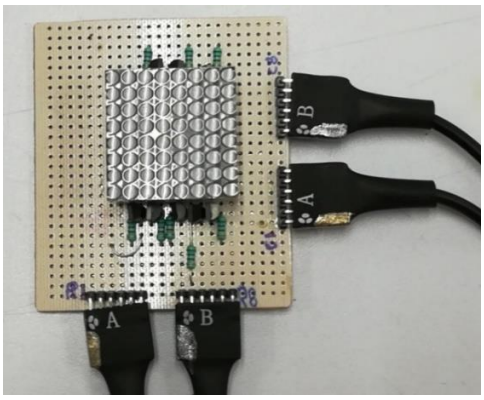

Figura 7.10. Foto del display de cátodo con transistores NPN en los renglones.

A continuación, se presentan las fotos en las figuras 7.11 a 7.13, así como un código ejemplo para mandar una flecha a un display matricial conectado de forma directa.

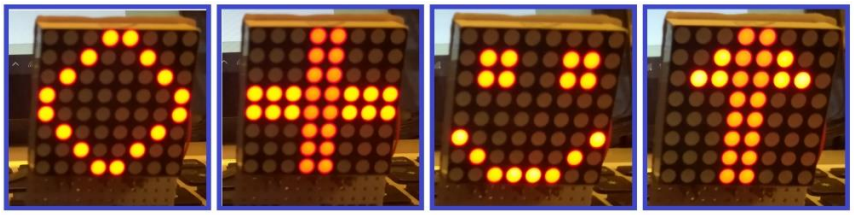

Figura 7.11. Símbolos del rombo, suma, carita feliz y flecha arriba.

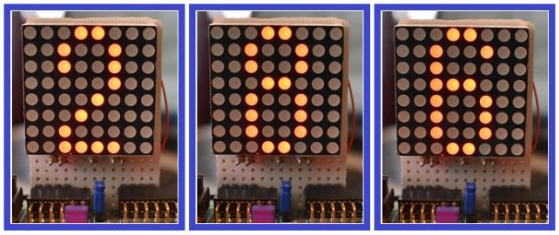

Figura 7.12. Números.

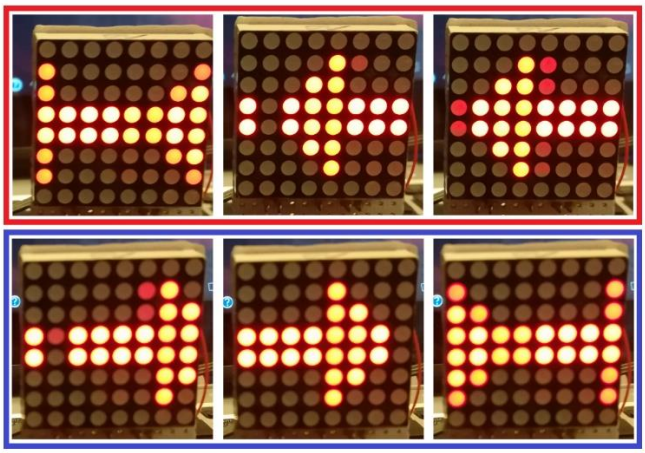

Figura 7.13. Imágenes del funcionamiento de una flecha en movimiento izquierda y derecha.

VHDL

```
-- Este código presenta una flecha en un display matricial de 8x8.
-- Los renglones van a resistencia y de ahi a los ánodos.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity matrixArrow is
        Generic (N: integer:=15); -- M: integer:=26); -- N valor de bits del divisor
  Port ( clk : in  STD_LOGIC; -- reloj de 50MHz
```

```
        R,C : out STD_LOGIC_VECTOR (8 downto 1)); -- Renglones y Columnas
end matrixArrow;

architecture matrixArrow of matrixArrow is
--divisor de M+1 bits
signal clkdiv: std_logic_vector (N downto 0);
--contador de tres bits que sirve para el barrido
signal barrido: std_logic_vector (2 downto 0);
--arreglo de datos para visualizar una flecha
type arreglo is array (1 to 8) of std_logic_vector(7 downto 0);
constant tabla : arreglo :=(  -- datos de flecha arriba
                              "00000000",
                              "00000100",
                              "00000110",
                              "11111111",
                              "11111111",
                              "00000110",
                              "00000100",
                              "00000000");
--signal tabla : arreglo;
begin

-- proceso del divisor cldiv
divisor: process (clk)
begin
        if clk'event and clk='1' then
        clkdiv <= clkdiv + 1;
        end if;
end process divisor;

--manda los datos del display
asigna: process (clkdiv(N-1), barrido) --, clkdiv(M))

begin

--      esta asignación funciona igual que clkdiv <= clkdiv + 1
        barrido <= clkdiv(N downto N-2);

--se mandan los datos a los renglones y las columnas con el contador barrido
                case barrido is
                when o"0" =>  R <= tabla(1); C <= "01111111";
                when o"1" =>  R <= tabla(2); C <= "10111111";
                when o"2" =>  R <= tabla(3); C <= "11011111";
                when o"3" =>  R <= tabla(4); C <= "11101111";
                when o"4" =>  R <= tabla(5); C <= "11110111";
                when o"5" =>  R <= tabla(6); C <= "11111011";
                when o"6" =>  R <= tabla(7); C <= "11111101";
```

```
when o"7" =>  R <= tabla(8); C <= "11111110";
when others =>R <= tabla(1); C <= "00000000";
end case;

end process asigna;

end matrixArrow;
```

LCD.

Un display de cristal líquido es otro visualizador (figura 7.14) comunmente utilizado en muchos dispositivos y cambian en tamaño y número de caracteres distribuidos por renglones, existen con y sin luz de fondo pudiéndose cambiar el color si se utiliza un led RGB. Lo principal para poder manejar un LCD es inicializarlo y posteriormente mandar los datos ya sean fijos como un mensaje o aviso o bien mandar un dato variable como la temperatura, presión, número de hojas restantes, etc. El número de datos a manejar en el bus puede ser de un nibble (4 bits) o un byte (8 bits). A continuación, se presenta el listado de puntos que se tienen que considerar para realizar la inicialización, también llamada Power-On Initialization y en la figura 7.15 se muestra su diagrama de estados.

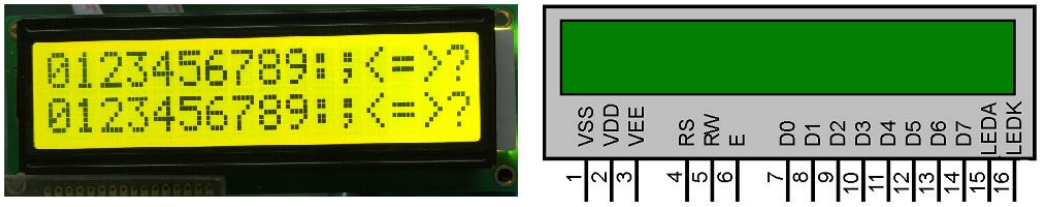

Figura 7.14. Imagen y diagrama LCD.

Para la tarjeta Spartan3E de Xilinx que trabaja con 4 bits se hacen los siguientes pasos:

- Se establece el número de bits a utilizar del bus de datos
- Esperar 15 ms o más, son 750,000 ciclos de reloj a 50 MHz.
- Escribir SF_D<11:8> = 0x3, pulso alto a LCD_E por 12 ciclos de reloj.
- Esperar 4.1 ms o más, son 205,000 ciclos de reloj a 50 MHz.
- Escribir SF_D<11:8> = 0x3, pulso alto a LCD_E por 12 ciclos de reloj.
- Esperar 100 µs o más, son 5,000 ciclos de reloj a 50 MHz.
- Escribir SF_D<11:8> = 0x3, pulso alto a LCD_E por 12 ciclos de reloj.
- Esperar 40 µs o más, son 2,000 ciclos de reloj a 50 MHz.
- Escribir SF_D<11:8> = 0x2, pulso alto a LCD_E por 12 ciclos de reloj.
- Esperar 40 µs o más, son 2,000 ciclos de reloj a 50 MHz.

Posteriormente, se debe configurar el display (Display Configuration):

- Mandar el commando de ajuste de función (Function Set command), 0x28, para configurar el display para operación.
- Mandar el commando de ajuste de modo de entrada (Entry Mode Set command), 0x06, para ajustar el apuntador de dirección con incremento automático.
- Mandar el commando de encendido/apagado del display (Display On/Off command), 0x0C, para encender el display y desabilitar el tintileo del cursor.

• Finalmente, mandar el commando de limpiar el display (Clear Display command), por lo menos 1.64 ms (82,000 ciclos de reloj) despues de mandarlo.

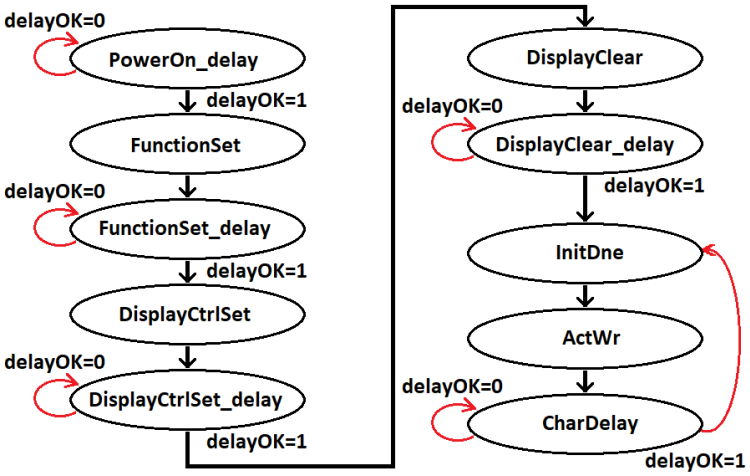

Figura 7.15. Diagrama de estado de la inicialización y el mando de comandos.

Por último, en la tabla 7.3 se muestran los comandos básicos de control para un LCD y en la tabla 7.4 los códigos ASCII (American Standar Code for Interchange Information) que se utilizan para el envío de datos al LCD.

Por ejemplo, de la tabla 7.3, primero poner RS=’0’ y RW=’0’ y para limiar el display se envía el comando (0x01); para posicionar el cursor en HOME (0x02); para colocar el cursor en la columna 1 de la:
línea 1 se envía el comando (0x80); // Sets cursor to line 1 of display
línea 2 se envía el comando (0xC0); // Sets cursor to line 2 of display
línea 3 se envía el comando (0x94); // Sets cursor to line 3 of display
línea 4 se envía el comando (0xD4); // Sets cursor to line 4 of display

Tabla 7.3. comandos de un LCD.

COMMAND	COMMAND CODE										COMMAND CODE
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
SCREEN CLEAR	0	0	0	0	0	0	0	0	0	1	Screen Clear, Set AC to 0 Cursor Reposition
CURSOR RETURN	0	0	0	0	0	0	0	0	1	*	DDRAM AD=0, Return, Content Changeless
INPUT SET	0	0	0	0	0	0	0	1	I/D	S	Set moving direction of cursor, Appoint if move
DISPLAY SWITCH	0	0	0	0	0	0	1	D	C	B	Set display on/off,cursor on/off, blink on/off
SHIFT	0	0	0	0	0	1	S/C	R/L	*	*	Remove cursor and whole display,DDRAM changeless
FUNCTION SET	0	0	0	0	1	DL	N	F	*	*	Set DL,display line,font
CGRAM AD SET	0	0	0	1	ACG						Set CGRAM AD, send receive data
DDRAM AD SET	0	0	1	ADD							Set DDRAM AD, send receive data
BUSY/AD READ CT	0	1	BF	AC							Executing internal function, reading AD of CT
CGRAM/ DDRAM DATA WRITE	1	0		DATA WRITE							Write data from CGRAM or DDRAM
CGRAM/ DDRAM DATA READ	1	1		DATA READ							Read data from CGRAM or DDRAM
	I/D=1: Increment Mode; I/D=0: Decrement Mode S=1: Shift S/C=1: Display Shift; S/C=0: Cursor Shift R/L=1: Right Shift; R/L=0: Left Shift DL=1: 8D DL=0: 4D N=1: 2R N=0: 1R F=1: 5x10 Style; F=0: 5x7 Style BF=1: Execute Internal Function; BF=0: Command Received **1+ADD: 80 Row=1 Col=1, 84 Row=1 Col=5, C0 Row=2 Col=1, C7 Row=2 Col=8**										DDRAM: Display data RAM CGRAM: Character Generator RAM ACG: CGRAM AD ADD: DDRAM AD & Cursor AD AC: Address counter for DDRAM & CGRAM

Por ejemplo, para mandar los caracteres ASCII de la tabla 7.4, primero poner RS='1' y RW='0', que es el DATA WRITE de la tabla 7.3. Para mandar la "S" mayúscula se envía el comando (0x53); para la "o" minúscula es (0x6F); para el número "1" es (0x31); para espacio es (0x20), etc.

NOTA: en ocasiones, dependiendo del origen del display, en los códigos (0x80) hasta la (0x9F) aparecen en blanco y en las líneas (0xB1) hasta la (0xDC) puede tener grabados caracteres chinos y los otros códigos pueden tener otros símbolos. Lo más parecido a grado Celsius "°" aparece en la (0xDF)

Tabla 7.4. ASCII para el envío de caracteres al LCD.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL	STX	SOT	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
80	א	ב	ג	ד	ה	ו	ז	ח	ט	י	ך	כ	ל	ם	מ	ן
90	נ	ס	ע	ף	פ	ץ	צ	ק	ר	ש	ת	¢	£	¥	₧	ƒ
A0	á	í	ó	ú	ñ	Ñ	ª	º	¿	⌐	¬	½	¼	¡	«	»
B0	░	▒	▓	│	┤	╡	╢	╖	╕	╣	║	╗	╝	╜	╛	┐
C0	└	┴	┬	├	─	┼	╞	╟	╚	╔	╩	╦	╠	═	╬	╧
D0	╨	╤	╥	╙	╘	╒	╓	╫	╪	┘	┌	█	▄	▌	▐	▀
E0	α	ß	Γ	π	Σ	σ	µ	τ	Φ	Θ	Ω	δ	∞	φ	ε	∩
F0	≡	±	≥	≤	⌠	⌡	÷	≈	°	∙	·	√	ⁿ	²	■	NBSP

Nota: Algunos LCD de China, contienen caracteres chinos en vez de estos.

“Si tienes alguna duda, apóyate con tu profesor”

Desarrollo.
Procedure.

Recuerde *MOSTRAR* sus circuitos funcionando a su profesor para la valoración del trabajo de laboratorio (TL). **Nota**: antes de pedir la revisión de cualquier punto, verificar que está funcionando como se solicita.

1. *Implement the code to program 2 images in a **matrix display**, one image is fixed and the other image must have movement (examples: happy face, arrows, numbers, letters, symbols, animals, sine signals, etc.) with a sensor activation and a voice message giving some indication (examples: “going up”, “going down”, “laughing”, etc.), illustrated in figure 7.16. An 8x8 or bigger matrix display will be used, energize with external source with or without power transistors or use a serial matrix display IC (i.e. MAX7219). The change between images is by means of a magnetic sensor. Report the codes (HDL & UCF) and several photos of fixed and moving images with explanatory text. Make a video (sent to the mail or put the link) explaining what is shown in the fixed image and the moving image.*

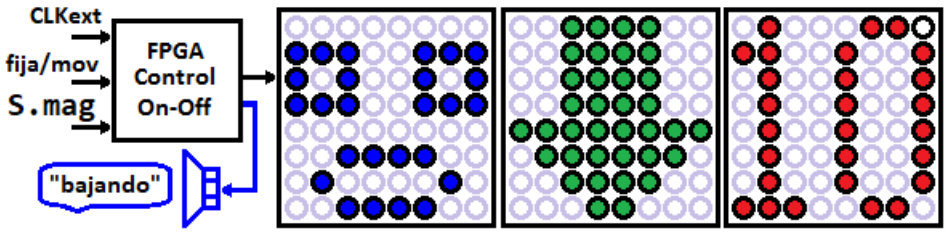

Figura 7.16. Figuras del display matricial.

2. *Implement in the language indicated by the teacher (VHDL or Verilog) the codes to display on the **LCD**: (**i**) a fixed message (i.e. name of your team-company-members); (**ii**) a marquee with movement control and (**iii**) a fixed message with variable data input (i.e. numbers from 0 to 63 controlled by the 6 switches (or one encoder) or temperature with LM35 sensor and ADC, etc.). It is possible to work with parallel LCD (i.e. Pmod CLP) and work with bytes or nibles or use a serial LCD (i.e. Pmod CLS). Report the codes (HDL & UCF) used and photos of its operation, remembering that all codes always carry comments, as well as at least 3 photos with explanatory text.*

3. *Write your comments and conclutions (include "Thank you" in at least one paragraph, "Thanks to this practice ..."). All the codes are presented in only one column with comments by line or blocks.*

NOTA: Respetar la numeración de cada punto de este formato en el reporte escrito [**máximo 20 cuartillas** si se entrega impreso (Letra Times New Roman de 12ptos, interlineado sencillo)].

Referencias.
References.

Para mayor información de las memorias, consulte las siguientes ligas:
Archivos Hexadecimal Memorias (en Google) (consultados en marzo-2022)
http://www.ii.uam.es/~etc1lab/practicas2004/ETC1_practica4b.htm
http://www.ii.uam.es/~etc1lab/practicas2005/ETC1_P3.htm
http://www.ii.uam.es/~etc1lab/

Declaración de arreglos en Verilog, consultada en marzo-2022,
http://stackoverflow.com/questions/3011510/how-to-declare-and-use-1d-and-2d-byte-arrays-in-verilog

Tutorial de Verilog, consultada en marzo-2022,
http://www.iuma.ulpgc.es/~nunez/clases-FdC/verilog/Verilog%20Tutorial%20v1.pdf

Memorias RAM y ROM, consultada en marzo-2022,
http://vlsiworld-asic.blogspot.mx/2012/02/verilog-code-for-ram-rom.html

Multi-Output Circuits: Encoders, Decoders, and Memories Lab Workbook, consultada en marzo-2022, http://www.xilinx.com/support/documentation/university/ISE-Teaching/HDL-Design/14x/Nexys3/Verilog/docs-pdf/lab3.pdf

FIFO memory, consultada en marzo-2022, http://www.fpga4student.com/2017/01/verilog-code-for-fifo-memory.html

SD card, consultadas en marzo-2022, http://www.fpga4fun.com/SD1.html http://www.fpga4fun.com/SD2.html

FPGA VGA Graphics in Verilog Part 1, consultada en marzo-2022, https://timetoexplore.net/blog/arty-fpga-vga-verilog-01

Tutorial Verilog, con sultado en marzo-2022, disponible en: https://www.chipverify.com/verilog/verilog-tutorial

Guía de usuario de la tarjeta Spartan 3E, marzo-2022, disponible en: http://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf

Matriz de leds, marzo-2022, disponible en: https://forums.xilinx.com/t5/CPLDs-Archived/need-of-vhdl-code-for-led-matrix/td-p/169452

MAX7221 Serially Interfaced, 8-Digit, LED Display Drivers, marzo-2022, disponible en: https://www.maximintegrated.com/en/products/power/display-power-control/MAX7221.html

Información y códigos de LCD, marzo-2022, disponible en: https://eewiki.net/pages/viewpage.action?pageId=4096079

16x2 LCD interfacing with CPLD Development Kit, consultado en marzo-2022, disponible en: https://www.pantechsolutions.net/16x2-lcd-interface-with-cpld-development-kit

Librerías VHDL de Intesc, consultado en marzo-2022, disponible en: https://intesc.mx/librerias-vhdl/?v=0b98720dcb2c

Ejemplos para el LCD de Digilent “Pmod CLP, CLS” en el *resource center*, consultado en marzo-2022, disponible en: PmodCLP, https://reference.digilentinc.com/reference/pmod/pmodclp/start PmodCLS, https://store.digilentinc.com/pmod-cls-character-lcd-with-serial-interface/

Juan Antonio Jaramillo Gómez, Mirna Salmerón Guzmán, Brahim El Filali, USO DE UN DISPLAY MATRICIAL DE 8X8 CON VHDL, consultado en marzo-2022, disponible en:

http://www.boletin.upiita.ipn.mx/index.php/ciencia/745-cyt-numero-64/1454-uso-de-un-display-matricial-de-8x8-con-vhdl

Juan Antonio Jaramillo Gómez, Mirna Salmerón Guzmán, Brahim El Filali, MANEJO DE UN LCD CON VHDL PARA VISUALIZAR LOS CARACTERES ASCII, consultado en marzo-2022, disponible en: http://www.boletin.upiita.ipn.mx/index.php/ciencia/850-cyt-numero-76/1786-manejo-de-un-lcd-con-vhdl-para-visualizar-los-caracteres-ascii

Reyes Barranca M. A., Arellano Cárdenas O., Flores Nava L. M., Electrónica Digital, CINVESTAV, agosto 2013, pp. 17 (Diseño jerárquico), 26 (RS232), 31 (Sensores IR), 36 (Actuadores servo), 41 (LCD), 51 (DAC), 57 (ADC), consultado en marzo-2022, disponible en: http://www.vlsilab.cinvestav.mx/files/Practicas_Spartan_3E.pdf

LCD terminales y comandos, consultado en marzo-2022, disponible en: https://electronicsforu.com/resources/learn-electronics/16x2-lcd-pinout-diagram

Datasheet de un controlador de LCD, consultado en marzo-2022, disponible en: https://www.buydisplay.com/download/ic/SPLC780.pdf

VHDL Predefined Attributes, consultado en marzo-2022, disponible en: https://www.csee.umbc.edu/portal/help/VHDL/attribute.html

Ejemplos VHDL, consultado en marzo-2022, disponible en: https://intesc.mx/practicas-avanxe7/?v=0b98720dcb2c

"Imprime la parte UPIITA a tu trabajo, busca información en la red y compártela con tus compañeros".

Recuerda que:
"La práctica hace al maestro"

¡Si tienes alguna duda apóyate en tu profesor!