UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Facultad de Ingenieria

División de Ingeniería Eléctrica

Reporte de Práctica 03

Construción de Máquinas de estados Usando Memorias Direccionamiento por Trayectoria

Grupo 05

Grupo de Teoría 02

Semestre 2022-2

LABORATORIO DE ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORAS

Profesor: Ing. Julio Cesar Cruz Estrada

Integrantes:
Vivanco Quintanar, Diego Armando

11 de marzo de 2022

1. Introducción

El direccionamiento por trayectoria guarda el estado siguiente y las salidas de cada estado de la carta ASM en una localidad de memoria. La porción de la memoria que indica el estado siguiente es llamada LIGA, mientras que la segunda porci´on de la memoria es usada para las SALIDAS.

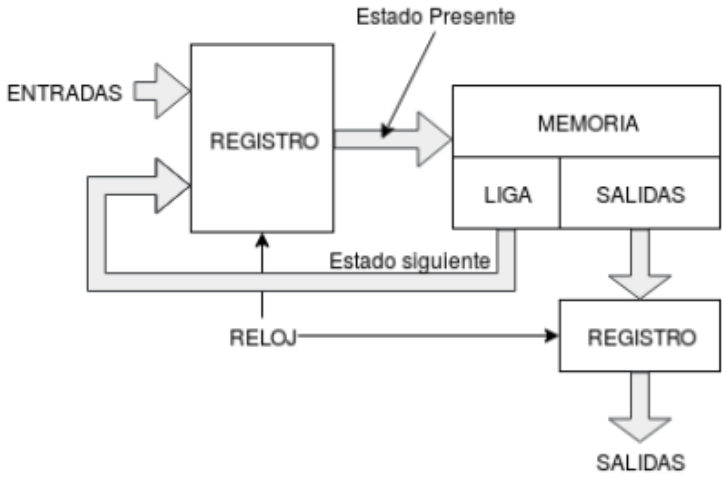

Figura 1: Direccionamiento por Trayectoria.

Cálculo del tamaño de la memoria ROM:

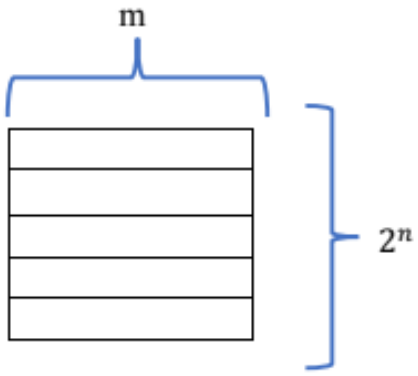

Figura 2: Memoria ROM.

n = bits de las entradas + bits de los estados

m = bits de la liga + bits de las salidas

Tamaño de memoria = $2^n * m$

Características del direccionamiento por trayectoria.

1. Soporta cualquier carta ASM.
2. Soporta salidas condicionales.
3. Se puede evaluar más de una condición en un estado.
4. Se desperdicia mucha memoria.
5. Elementos de Hardware.
    a Memoria ROM.
    b Registro.

2. Objetivo

- Familiarizar al alumno en el conocimiento de construcción de máquinas de estados usando direccinamiento de memorias con el método de direccionamiento por trayectoria.

3. Duración

- 1 semana.

4. Desarrollo

4.1. Actividad 1

Como parte de los requerimientos solicitados por el profesor, se nos pide crear una carta ASM que tenga los siguientes elementos y características:

- 6 estados
- 4 entradas
- 4 salidas
- La carta ASM debe tener salidas condicionales.
- La carta ASM debe tener al menos un estado con entradas en cascada

A partir de esa carta ASM encuentre el contenido de memoria utilizando el direccionamiento por trayectoria. Recuerde que antes de construir la tabla se debe asignar a cada estado de la carta ASM una representación binaria. Así mismo, recuerde que para cada estado es necesario considerar todas las posibles combinaciones de las variables de entrada, aun cuando algunas de ellas no se utilicen en determinado estado.

4.1.1. Creación de la carta ASM

La carta ASM que hice (figura 3) contiene los 6 estados solicitados, asi como el nímero de entradas y de salidas, dos estados contienen dos entradas en cascada y algunos tienen salidas condicionales.

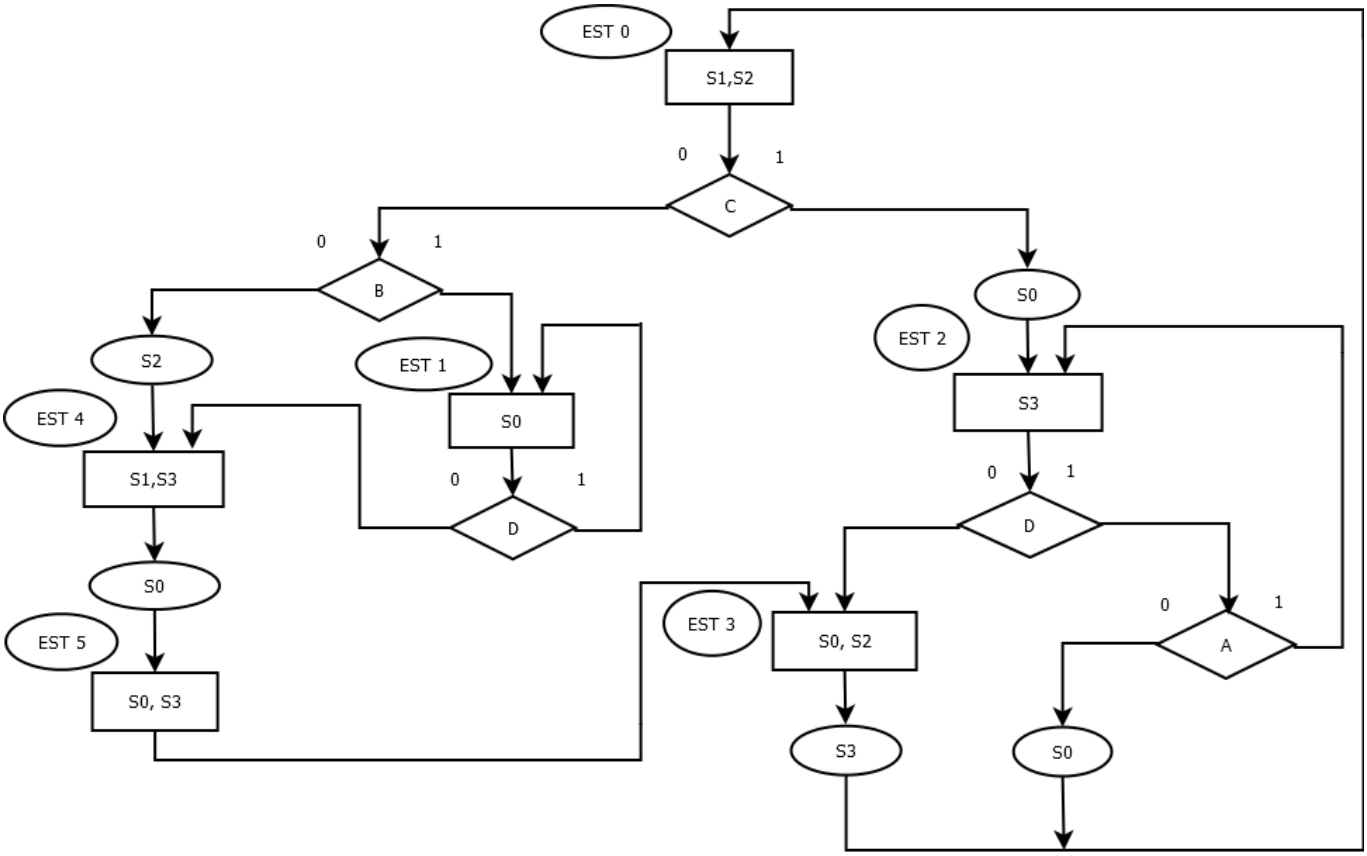

Figura 3: Carta ASM creada.

4.1.2. Cálculo del tamaño de la memoria ROM

Como se estan usando 6 estados, se requieren de 2^3 estados, es decir que necesitamos 3 bits para representar 8 estados, de los cuales solo 6 serán ocupados.
Considerando que tenemos 4 entradas, se ucupan 4 bits, por lo que:

$$n=4+3=7$$

En este caso el número de bits empleados para los estados es el mismo para la liga y considerando que son 4 salidas las que se ocupan, se requieren de 4 bits más.

$$m=3+4=7$$

Finalmente:

$$\text{Tamaño de memoria} = 2^7 * 7bits$$

4.1.3. Asignación binaria de los estados

Estado	Valor Binario
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

4.1.4. Construir la tabla con el contenido de la memoria

A continuación se muestran las tablas con la dirección y el contenido de la memoria con el direccionamiento por trayectoria.

Dirección							Contenido						
Entradas				Edo. Presente			Liga			Salidas			
A	B	C	D	P2	P1	P0	L2	L1	L0	S3	S2	S1	S0
0	0	0	0	0	0	0	1	0	0	0	1	1	0
0	0	0	1	0	0	0	1	0	0	0	1	1	0
0	0	1	0	0	0	0	0	1	0	0	1	1	1
0	0	1	1	0	0	0	0	1	0	0	1	1	1
0	1	0	0	0	0	0	0	0	1	0	1	1	0
0	1	0	1	0	0	0	0	0	1	0	1	1	0
0	1	1	0	0	0	0	0	1	0	0	1	1	1
0	1	1	1	0	0	0	0	1	0	0	1	1	1
1	0	0	0	0	0	0	1	0	0	0	1	1	0
1	0	0	1	0	0	0	1	0	0	0	1	1	0
1	0	1	0	0	0	0	0	1	0	0	1	1	1
1	0	1	1	0	0	0	0	1	0	0	1	1	1
1	1	0	0	0	0	0	0	0	1	0	1	1	0
1	1	0	1	0	0	0	0	0	1	0	1	1	0
1	1	1	0	0	0	0	0	1	0	0	1	1	1
1	1	1	1	0	0	0	0	1	0	0	1	1	1

Cuadro 1: Dirección y contenido de memoria para el estado 0

Dirección							Contenido						
Entradas				Edo. Presente			Liga			Salidas			
A	B	C	D	P2	P1	P0	L2	L1	L0	S3	S2	S1	S0
0	0	0	0	0	0	1	1	0	0	0	0	0	1
0	0	0	1	0	0	1	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1	0	0	0	0	0	1
0	0	1	1	0	0	1	0	0	1	0	0	0	1
0	1	0	0	0	0	1	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0	0	1	0	0	0	1
0	1	1	0	0	0	1	1	0	0	0	0	0	1
0	1	1	1	0	0	1	0	0	1	0	0	0	1
1	0	0	0	0	0	1	1	0	0	0	0	0	1
1	0	0	1	0	0	1	0	0	1	0	0	0	1
1	0	1	0	0	0	1	1	0	0	0	0	0	1
1	0	1	1	0	0	1	0	0	1	0	0	0	1
1	1	0	0	0	0	1	1	0	0	0	0	0	1
1	1	0	1	0	0	1	0	0	1	0	0	0	1
1	1	1	0	0	0	1	1	0	0	0	0	0	1
1	1	1	1	0	0	1	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	1	1	1	0	0	0
0	0	0	1	0	1	0	0	0	0	1	0	0	1
0	0	1	0	0	1	0	0	1	1	1	0	0	0
0	0	1	1	0	1	0	0	0	0	1	0	0	1
0	1	0	0	0	1	0	0	1	1	1	0	0	0
0	1	0	1	0	1	0	0	0	0	1	0	0	1
0	1	1	0	0	1	0	0	1	1	1	0	0	0
0	1	1	1	0	1	0	0	0	0	1	0	0	1
1	0	0	0	0	1	0	0	1	1	1	0	0	0
1	0	0	1	0	1	0	0	1	0	1	0	0	0
1	0	1	0	0	1	0	0	1	1	1	0	0	0
1	0	1	1	0	1	0	0	1	0	1	0	0	0
1	1	0	0	0	1	0	0	1	1	1	0	0	0
1	1	0	1	0	1	0	0	1	0	1	0	0	0
1	1	1	0	0	1	0	0	1	1	1	0	0	0
1	1	1	1	0	1	0	0	1	0	1	0	0	0

Cuadro 2: Dirección y contenido de memoria para el estado 1 y 2

Dirección							Contenido						
Entradas				Edo. Presente			Liga			Salidas			
A	B	C	D	P2	P1	P0	L2	L1	L0	S3	S2	S1	S0
0	0	0	0	0	1	1	0	0	0	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	1	0	1
0	0	1	0	0	1	1	0	0	0	1	1	0	1
0	0	1	1	0	1	1	0	0	0	1	1	0	1
0	1	0	0	0	1	1	0	0	0	1	1	0	1
0	1	0	1	0	1	1	0	0	0	1	1	0	1
0	1	1	0	0	1	1	0	0	0	1	1	0	1
0	1	1	1	0	1	1	0	0	0	1	1	0	1
1	0	0	0	0	1	1	0	0	0	1	1	0	1
1	0	0	1	0	1	1	0	0	0	1	1	0	1
1	0	1	0	0	1	1	0	0	0	1	1	0	1
1	0	1	1	0	1	1	0	0	0	1	1	0	1
1	1	0	0	0	1	1	0	0	0	1	1	0	1
1	1	0	1	0	1	1	0	0	0	1	1	0	1
1	1	1	0	0	1	1	0	0	0	1	1	0	1
1	1	1	1	0	1	1	0	0	0	1	1	0	1
0	0	0	0	1	0	0	1	0	1	1	0	1	1
0	0	0	1	1	0	0	1	0	1	1	0	1	1
0	0	1	0	1	0	0	1	0	1	1	0	1	1
0	0	1	1	1	0	0	1	0	1	1	0	1	1
0	1	0	0	1	0	0	1	0	1	1	0	1	1
0	1	0	1	1	0	0	1	0	1	1	0	1	1
0	1	1	0	1	0	0	1	0	1	1	0	1	1
0	1	1	1	1	0	0	1	0	1	1	0	1	1
1	0	0	0	1	0	0	1	0	1	1	0	1	1
1	0	0	1	1	0	0	1	0	1	1	0	1	1
1	0	1	0	1	0	0	1	0	1	1	0	1	1
1	0	1	1	1	0	0	1	0	1	1	0	1	1
1	1	0	0	1	0	0	1	0	1	1	0	1	1
1	1	0	1	1	0	0	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1	0	1	1	0	1	1
1	1	1	1	1	0	0	1	0	1	1	0	1	1

Cuadro 3: Dirección y contenido de memoria para el estado 3 y 4

Dirección							Contenido						
Entradas				Edo. Presente			Liga			Salidas			
A	B	C	D	P2	P1	P0	L2	L1	L0	S3	S2	S1	S0
0	0	0	0	1	0	1	0	1	1	1	0	0	1
0	0	0	1	1	0	1	0	1	1	1	0	0	1
0	0	1	0	1	0	1	0	1	1	1	0	0	1
0	0	1	1	1	0	1	0	1	1	1	0	0	1
0	1	0	0	1	0	1	0	1	1	1	0	0	1
0	1	0	1	1	0	1	0	1	1	1	0	0	1
0	1	1	0	1	0	1	0	1	1	1	0	0	1
0	1	1	1	1	0	1	0	1	1	1	0	0	1
1	0	0	0	1	0	1	0	1	1	1	0	0	1
1	0	0	1	1	0	1	0	1	1	1	0	0	1
1	0	1	0	1	0	1	0	1	1	1	0	0	1
1	0	1	1	1	0	1	0	1	1	1	0	0	1
1	1	0	0	1	0	1	0	1	1	1	0	0	1
1	1	0	1	1	0	1	0	1	1	1	0	0	1
1	1	1	0	1	0	1	0	1	1	1	0	0	1
1	1	1	1	1	0	1	0	1	1	1	0	0	1
0	0	0	0	1	1	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	0	0
0	0	1	0	1	1	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	0	0	0	0	0	0	0
0	1	0	1	1	1	0	0	0	0	0	0	0	0
0	1	1	0	1	1	0	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	0	0	0	0	0	0
1	0	0	0	1	1	0	0	0	0	0	0	0	0
1	0	0	1	1	1	0	0	0	0	0	0	0	0
1	0	1	0	1	1	0	0	0	0	0	0	0	0
1	0	1	1	1	1	0	0	0	0	0	0	0	0
1	1	0	0	1	1	0	0	0	0	0	0	0	0
1	1	0	1	1	1	0	0	0	0	0	0	0	0
1	1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0	0

Cuadro 4: Dirección y contenido de memoria para el estado 5 y 6

Dirección							Contenido						
Entradas				Edo. Presente			Liga			Salidas			
A	B	C	D	P2	P1	P0	L2	L1	L0	S3	S2	S1	S0
0	0	0	0	1	1	1	0	0	0	0	0	0	0
0	0	0	1	1	1	1	0	0	0	0	0	0	0
0	0	1	0	1	1	1	0	0	0	0	0	0	0
0	0	1	1	1	1	1	0	0	0	0	0	0	0
0	1	0	0	1	1	1	0	0	0	0	0	0	0
0	1	0	1	1	1	1	0	0	0	0	0	0	0
0	1	1	0	1	1	1	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0	0	0	0	0	0	0
1	0	0	0	1	1	1	0	0	0	0	0	0	0
1	0	0	1	1	1	1	0	0	0	0	0	0	0
1	0	1	0	1	1	1	0	0	0	0	0	0	0
1	0	1	1	1	1	1	0	0	0	0	0	0	0
1	1	0	0	1	1	1	0	0	0	0	0	0	0
1	1	0	1	1	1	1	0	0	0	0	0	0	0
1	1	1	0	1	1	1	0	0	0	0	0	0	0
1	1	1	1	1	1	1	0	0	0	0	0	0	0

Cuadro 1: Dirección y contenido de memoria para el estado 7

Como podemos observar se ocupa mucha memoria ya que en algunos casos tanto la liga como la salida se repiten porque algunas entradas no afectan la transición de un estado a otro.

5. Actividad 2

Una vez que haya obtenido el contenido de memoria, implemente el direccionamiento por trayectoria utilizando el software de desarrollo Quartus y escriba el contenido de memoria obtenido.
Como se explicó en la introducción la arquitectura del direccionamiento por trayectoria se compone de dos elementos, los cuales son una memoria ROM y un registro.

5.1. Memoria ROM

A continuación se muestra el código que nos ayuda a crear una memoria ROM, en el primer bloque de código podemos ver las bibliotecas que se ocupan, posteriormente vemos la definición de la entidad, declaramos dos varibales como genericas, siendo estas las que nos indican los bits que se necesitan tanto las direcciones de memoria como el tamaño del contenido de la memoria.
Finalmente declaramos las entradas y las salidas del bloque de la memoria, siendo las entradas un reloj (clk) y los bits que nos apuntan a las direcciones de la memoria (address) y como salida tenemos a la concatenación de los bits de la liga y de las salidas (data_out).

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity mem_rom is
  generic(data_width : natural :=7;  --Contenido de la memoria
        addr_length : natural :=7 --Direccion de memoria
        );
  port(
      clk: in std_logic; --Reloj
      address: in std_logic_vector(addr_length-1 downto 0); --Edo. Presente
      data_out: out std_logic_vector(data_width-1 downto 0)  --Edo.
         Siguiente
     );
end mem_rom;
```

Listing 1: Definición de la entidad de la memoria ROM.

En el siguiente bloque podemos ver la estrutura de la arquitectura de la memoria en donde se calcula el tamaño de la memoria, para definir una variable de tipo array a la que le asociamos el vector del contenido de la memoria, finalmente asociamos lo anterior a una variable de tipo constante para cargar el contenido de la memoria obtenido en la actividad anterior.

```
architecture arch_rom of mem_rom is
  constant mem_size : natural := 2**addr_length;
  type mem_type is array(mem_size-1 downto 0) of std_logic_vector(
     data_width-1 downto 0);
  constant mem : mem_type :=
    (
      -- Contenido para estado 0
      0 => b"1000110", 1 =>b"1000110", 2 => b"0100111", 3 =>b"0100111",
      4 => b"0010110", 5 =>b"0010110", 6 => b"0100111", 7 =>b"0100111",
      8 => b"1000110", 9 =>b"1000110", 10 => b"0100111", 11 =>b"0100111",
      12 => b"0010110", 13 =>b"0010110", 14 => b"0100111", 15 =>b"0100111",

      --Contenido para estado 1
      16 => b"1000001", 17 =>b"0010001", 18 => b"1000001", 19 =>b"0010001",
      20 => b"1000001", 21 =>b"0010001", 22 => b"1000001", 23 =>b"0010001",
      24 => b"1000001", 25 =>b"0010001", 26 => b"1000001", 27 =>b"0010001",
      28 => b"1000001", 29 =>b"0010001", 30 => b"1000001", 31 =>b"0010001",

      --Contenido para estado 2
      32 => b"0111000", 33 =>b"0001001", 34 => b"0111000", 35 =>b"0001001",
      36 => b"0111000", 37 =>b"0001001", 38 => b"0111000", 39 =>b"0001001",
      40 => b"0111000", 41 =>b"0101000", 42 => b"0111000", 43 =>b"0101000",
      44 => b"0111000", 45 =>b"0101000", 46 => b"0111000", 47 =>b"0101000",
```

Listing 2: Arquitectura de la memoria ROM y una primera parte de su contenido.

En el siguiente bloque de código se muestran las siguientes direcciones de memoria a las que se les asocio el contenido de la memoria obtenido de la actividad anterior.

```
      --Contenido para estado 3
      48 => b"0001101", 49 =>b"0001101", 50 => b"0001101", 51 =>b"0001101",
      52 => b"0001101", 53 =>b"0001101", 54 => b"0001101", 55 =>b"0001101",
      56 => b"0001101", 57 =>b"0001101", 58 => b"0001101", 59 =>b"0001101",
      60 => b"0001101", 61 =>b"0001101", 62 => b"0001101", 63 =>b"0001101",


      --Contenido para estado 4
      64 => b"1011011", 65 =>b"1011011", 66 => b"1011011", 67 =>b"1011011",
      68 => b"1011011", 69 =>b"1011011", 70 => b"1011011", 71 =>b"1011011",
      72 => b"1011011", 73 =>b"1011011", 74 => b"1011011", 75 =>b"1011011",
      76 => b"1011011", 77 =>b"1011011", 78 => b"1011011", 79 =>b"1011011",


      --Contenido para estado 5
      80 => b"0111001", 81 =>b"0111001", 82 => b"0111001", 83 =>b"0111001",
      84 => b"0111001", 85 =>b"0111001", 86 => b"0111001", 87 =>b"0111001",
      88 => b"0111001", 89 =>b"0111001", 90 => b"0111001", 91 =>b"0111001",
      92 => b"0111001", 93 =>b"0111001", 94 => b"0111001", 95 =>b"0111001",


       --Contenido para estados 6 y 7
      others =>b"0000000"
    );
```

Listing 3: Segunda parte del contenido de la memoria ROM.

Una vez que hemos definido el contenido de la memoria, procedemos a crear el proceso de direccionamiento, para ello utilizamos el reloj, por cada flanco de subida, estaremos recorriendo las direcciones de memoria mediante el casteo de la entrada (liga -> address).

```
begin
  process (clk)
  begin
    if rising_edge(clk) then
      data_out <= mem(to_integer (unsigned(address)));
    end if;
  end process;
end architecture arch_rom;
```

Listing 4: Proceso de acceso mediante las direcciones al contenido de la memoria ROM.

5.2. Registro

En el siguiente bloque de código podemos ver la implementación de un registro en VHDL, como primer punto colocamos las bibliotecas básicas, posteriormente declaramos una variable generica llamada data_width la cual definira el número de bits para la salida, posteriormente definimos las entradas y las salidas del registro. Tenemos en las entradas el reset, el reloj (clk), entradas, la liga y como salida tenemos lo que es el estado presente.

Dentro de la arquitectura tenemos un proceso, el cual verifica el estado del reset, si esta activado el registro manda a su salida el estado 0 y las salidas de la carta ASM desactivadas. Cuando el reset esta desactivado y cuando se tenga un flanco de subida en la señal de reloj se concatenará la liga y la entrada para ser asignado a la salida.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity registro is
  generic(
    data_width : natural :=7
  );
  port(
    q : out std_logic_vector(data_width-1 downto 0);
    entradas: in std_logic_vector(3 downto 0);
    liga : in std_logic_vector(2 downto 0);
    rst: in std_logic;
    clk: in std_logic
  );
end entity registro;

architecture arch_reg of registro is
begin
  process (clk)
  begin
    if rst = '1' then
      q <= "0000000";
    elsif rising_edge(clk) then
      q <= liga & entradas;
    end if;
  end process;
end architecture arch_reg;
```

Listing 5: Código en VHDL para el registro.

6. Actividad 3

Simule su diseño para probar su funcionamiento. recuerden que en sus simulaciones debe aparecer el contenido de la memoria adem´as del estado presente.

6.1. Esquemático

Para poder realizar la simulación debemos unir los dos componentes de hardware del direccionamiento por trayectoria, para estructurar la arquitectura en VHDL hacemos uso de esquemáticos, la ventaja de Quartus es que a partir del código se pueden crear símbolos esquematicos. El esquemático obtenido se muestra a continuación:

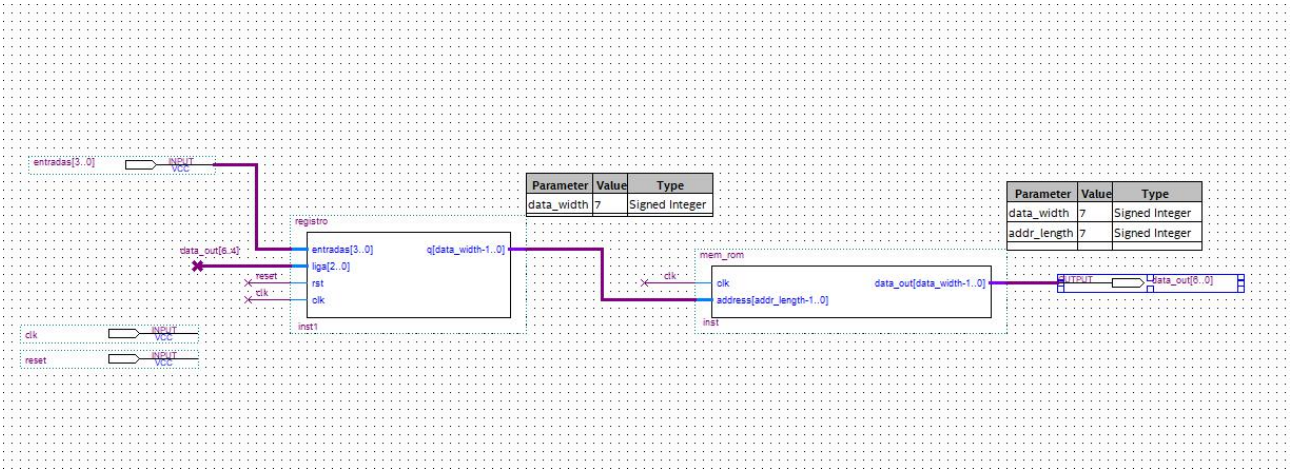

Figura 4: Arquitectura del direccionamiento por Trayectoría en VHDL.

Finalmente en la figura 5 tenemos el resultado de la simulación del proyecto.

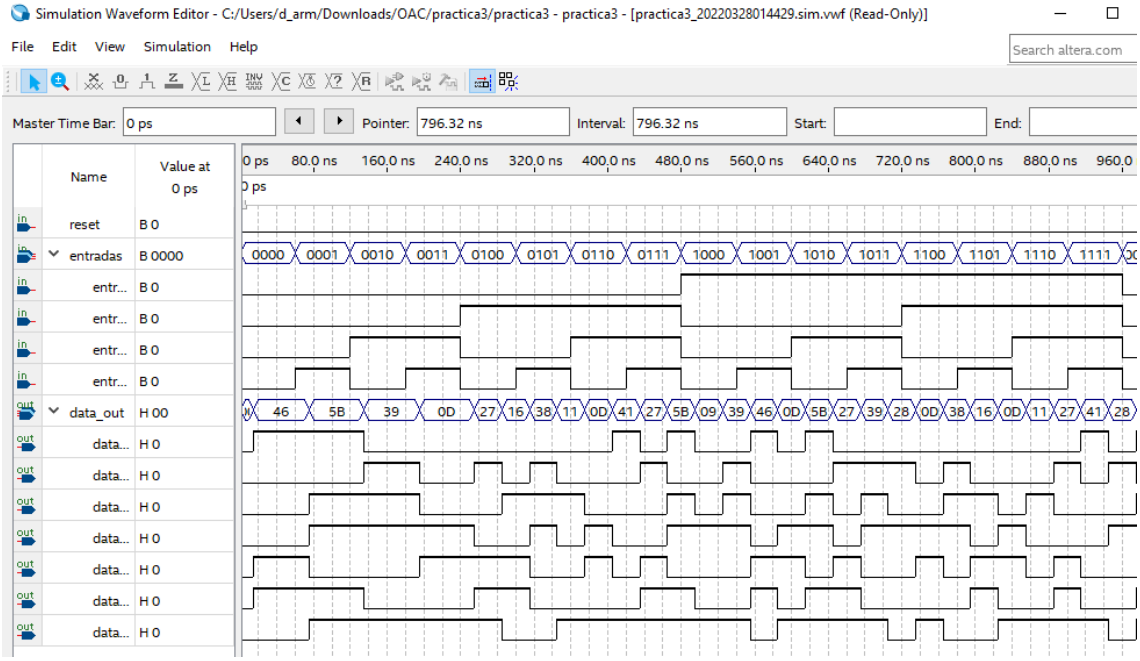

Figura 5: Arquitectura del direccionamiento por Trayectoría en VHDL

Como podemos ver en la figura 5 la simulación obtenida es correcta, en las entradas tenemos los valores binarios de todas las combinaciones asi como el estado en bajo y alto de cada entrada, lo mismo tenemos para la salida 'q', la cual es la concatenación del estado siguiente (liga) y las salidas activadas del estado presente.Por ejemplo en el tiempo 1, las entradas estan apagadas y como en el estado 0 nos vamos a un condicional que depende de la entrada C, y su valor es 0, nos lleva al estado 4 y en el recorrido se activan las salidas S1 y s2. Si se observa bien, en la salidas como valor agrupado se específico que los valores mostrados se representen en hexadecimal, ya que facilita la lectura.Del caso descrito anteriormente debo aclarar que para la salida, los bits más significativos representan el estado siguiente y los bits menos significativos indican las

salidas que se activan, para el estado 0 con las condiciones mencionadas nos manda al estado 4 (en hexadecimal es 4) y para las salidas activadas S1 y S2 (en hexadecimal es 6) tal y como se puede ver en la figura 5.

7. Conclusiones

El objetivo de la práctica se cumplió puesto que de manera teórica se obtuvo la tabla de valores para el direccionamiento por trayectoría de una Carta ASM, que en este caso se tuvo que trabajar con una totalmente diferente al del manual de la práctica. Respecto a la implementación en VHDL no se tuvo problema alguno, en mi caso lo nuevo fue hacer los módulos en forma de esquemático a partir de códigos en VHDL. Finalmente puedo decir que este direccionamiento tiene la desventaja de que se desperdicia mucha memoria para los casos en que se tenga un paso contiguo en la carta ASM, siendo que los valores para la liga y las salidas se repiten en varias direcciones de memoria.

Referencias

[1] Biorobotics UNAM.(2020) *Laboratorio de Organización y Arquitectura de Computadoras Práctica No. 3 Construción de Máquinas de estados Usando Memorias Direccionamiento por Trayectoria* Recuperado el 11 de marzo de 2022, de Biorobotics UNAM en https://biorobotics.fi-p.unam.mx/wp-content/uploads/Courses/laboratorio_de_organizacion_y_arquitectura_de_computadoras/2020-2/practicas/practica3_oac.pdf

[2] Savage Carmona Jesús, Vázquez Gabriel y Chávez Rodríguez Norma Elva . (2015) *DISEÑO DE MICROPROCESADORES* Recuperado el 11 de marzo de 2022, de Biorobotics UNAM. en https://biorobotics.fi-p.unam.mx/wp-content/uploads/Courses/arquitectura_de_computadoras/material_de_apoyo/diseo_de_procesadores.pdf