

Facultad de Ingeniería



Lab. Organización y Arquitectura de Computadoras

Práctica No 6 Secuenciador básico

Alumnos

- Monsalvo Bolaños Melissa Monserrat
- Romero Andrade Cristian

Grupo: 01

Profesor
Ing. Adrian Ulises Mercado Martinez

Semestre
2022-1



Índice

1	Objetivo	2
2	Introducción	2
2.1	Paso Contiguo (C) [00]	4
2.2	Salto Condicional (SC) [01]	4
2.3	Salto de Transformación (ST) [10]	5
2.4	Salto de Interrupción (SI) [11]	5
3	Desarrollo	6
3.1	Secuenciador	6
3.2	Carta ASM	9
3.3	Simulación	14
4	Conclusiones	15
	Referencias	15

1. Objetivo

Familiarizar al alumno en el conocimiento del secuenciador básico, el cual es una parte fundamental del procesador.

2. Introducción

Para el diseño de los módulos de control de una computadora se requieren máquinas de estados que sean capaces de ejecutar algoritmos más complejos. Haciendo modificaciones y agregando componentes a la variante del direccionamiento implícito se pueden crear máquinas de estados que efectúen cartas ASM con llamadas a subrutinas, estructuras DO WHILE, iteraciones tipo FOR, entre otras. Los dispositivos que son capaces de efectuar este tipo de operaciones son llamados secuenciadores.

A continuación, se muestra el diagrama de bloques de un secuenciador básico. Como puede observar en el diagrama, la dirección del estado siguiente, dada por el bus Y , puede venir de dos lugares posibles: Del registro μPC o de la entrada D .

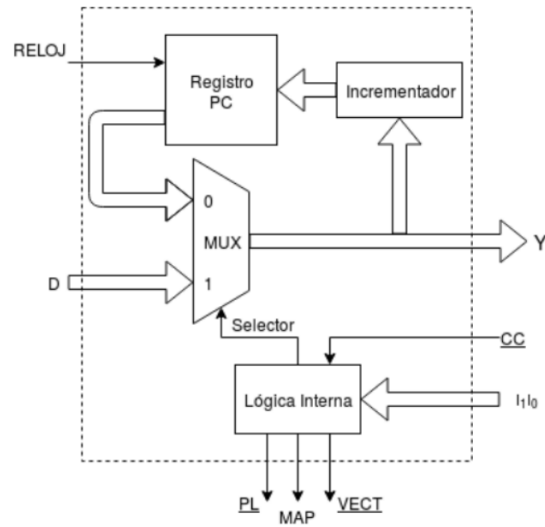


Figura 1: Diagrama de bloques interno de un secuenciador básico

El secuenciador cuenta con una lógica interna que se encarga de generar las señales que controlan al multiplexor. Dependiendo de la instrucción dada por las líneas I_1 e I_0 y de la línea CC , la lógica es capaz de seleccionar entre la salida del registro μPC o la entrada D .

La lógica interna también genera las líneas PL , MAP y $VECT$, las cuales seleccionan unos registros cuyas salidas están conectadas a la entrada D del secuenciador. De esta forma la dirección de salto puede venir de tres lugares distintos.

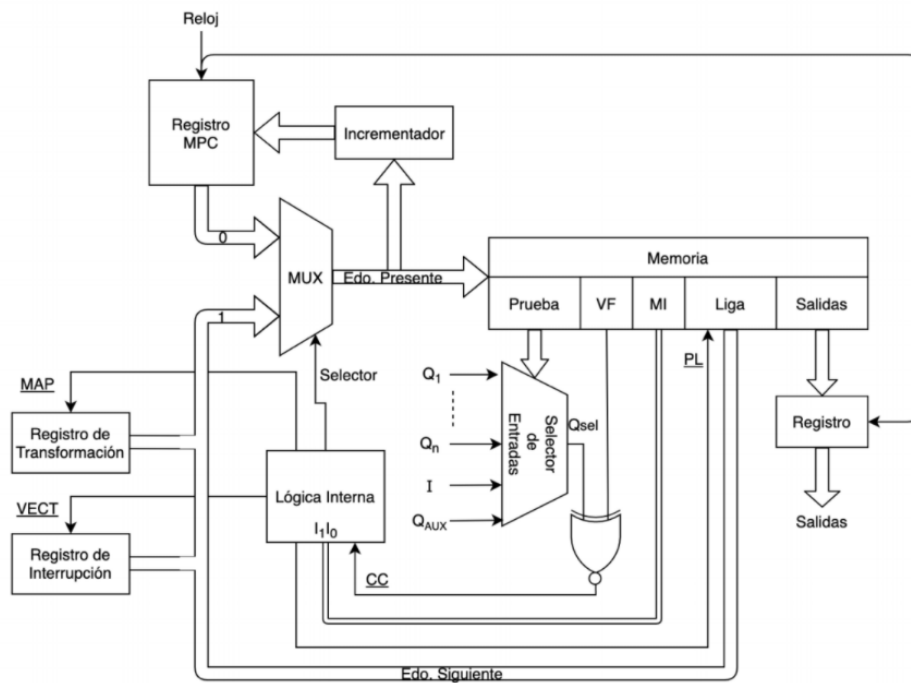


Figura 2: Diagrama de bloques interno de un secuenciador básico conectado a memoria

2.1. Paso Contiguo (C) [00]

En la instrucción continúa la dirección del estado siguiente la proporciona el registro μPC .

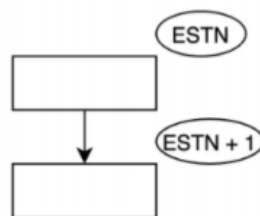


Figura 3: Carta ASM de Paso contiguo

2.2. Salto Condicional (SC) [01]

En esta instrucción se revisa el valor de la línea CC , si es igual a uno, la dirección del estado siguiente la proporciona el registro μPC ; si es igual a cero, la dirección del estado siguiente, contenida en el registro seleccionado por PL , ingresa a través de la entrada D .

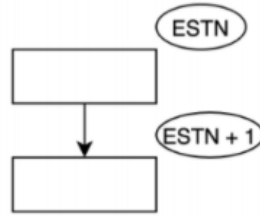


Figura 4: Carta ASM de Salto Condicional

2.3. Salto de Transformación (ST) [10]

La dirección del estado siguiente se obtiene del registro seleccionado por la línea de MAP. Este registro también está conectado a la entrada D. Aquí se introduce una nueva notación de carta ASM: un rombo con varias bifurcaciones. La bifurcación que se elija dependerá del contenido del registro seleccionado por MAP.

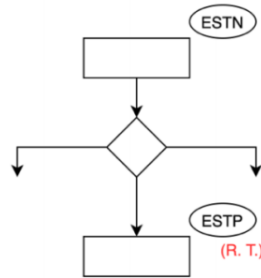


Figura 5: Carta ASM de Salto de Transformación

2.4. Salto de Interrupción (SI) [11]

En esta instrucción se revisa el valor de CC, si es igual a uno, la dirección del estado siguiente proviene del registro μPC ; si es igual a cero, la dirección del estado siguiente, contenida en el registro seleccionado por VECT, ingresa a través de la entrada D.

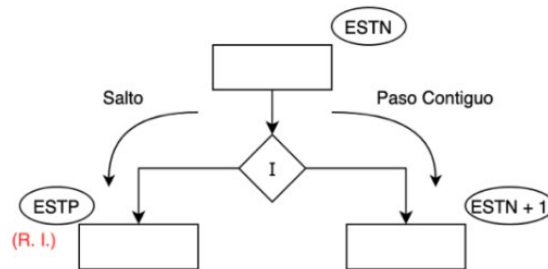


Figura 6: Carta ASM de Salto de Interrupción

3. Desarrollo

3.1. Secuenciador

Primeramente realizamos la tabla del secuenciador basado en la figura 1. en la cual debemos crear la lógica interna la cual escribimos la tabla 1 de las microinstrucciones en el siguiente bloque en VHDL. Donde después creamos el multiplexor que selecciona la secuencia si esta es contigua, usando el incrementador (código 2) o la que nos manda la memoria (código 5) ya sea por transferencia o por interrupción. Nos auxiliamos con otro bloque de incrementador (código 3) y un registro (código 4) para pasar los bits.

Tabla 1: Lógica interna (Microinstrucciones)

Entradas			Salidas				
I1	I0	CC	PL	MAP	VECT	Selector	Y
0	0	0	1	1	1	0	uPC
0	0	1	1	1	1	0	uPC
0	1	0	0	1	1	0	uPC
0	1	1	0	1	1	1	D
1	0	0	1	0	1	1	D
1	0	1	1	0	1	1	D
1	1	0	1	1	0	0	uPC
1	1	1	1	1	0	1	D

Código 1: logica_interna.vhd

```
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
6  entity logica_interna is
7      Port ( mi : in STD_LOGIC_vectOR(1 downto 0);
8            cc : in STD_LOGIC;
9            sel : out STD_LOGIC;
10           pl : out STD_LOGIC;
11           map_li : Out STD_LOGIC;
12           vect : out STD_LOGIC);
13 end logica_interna;
14
15 architecture Behavioral of logica_interna is
16 begin
17     process (mi, cc)
18     begin
19         if mi = "00" then
20             pl <= '0';map_li <= '0';vect <= '0';sel <= '0';
21
22         elsif (mi = "01" and cc = '0') then
23             pl <= '1';map_li <= '0';vect <= '0';sel <= '0';
24         elsif (mi = "01" and cc = '1') then
25             pl <= '1';map_li <= '0';vect <= '0';sel <= '1';
26
27         elsif mi = "10" then
28             pl <= '0';map_li <= '1';vect <= '0';sel <= '1';
29
30         elsif (mi = "11" and cc = '0') then
31             pl <= '0';map_li <= '0';vect <= '1';sel <= '0';
```

```

32     elsif (mi = "11" and cc = '1') then
33         pl <= '0';map_li <= '0';vect <= '1';sel <= '1';
34     end if;
35 end process;
36 end Behavioral;

```

Código 2: mux_sec.vhd

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity mux_sec is
5      Port ( SEL : in STD_LOGIC;
6            E0 : in STD_LOGIC_VECTOR(3 downto 0);
7            E1 : in STD_LOGIC_VECTOR(3 downto 0);
8            SALIDA : out STD_LOGIC_VECTOR(3 downto 0));
9  end mux_sec;
10
11 architecture Behavioral of mux_sec is
12 begin
13     process (SEL, E0, E1)
14     begin
15         if SEL = '0' then
16             SALIDA <= E0;
17         elsif SEL = '1' then
18             SALIDA <= E1;
19         end if;
20     end process;
21 end Behavioral;

```

Código 3: incrementador.vhd

```

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
6  entity incrementador is
7      Port ( ENTRADA : in STD_LOGIC_VECTOR(3 downto 0);
8            SALIDA : out STD_LOGIC_VECTOR(3 downto 0));
9  end incrementador;
10
11 architecture Behavioral of incrementador is
12 begin
13     process (ENTRADA)
14     begin
15         SALIDA <= ENTRADA + 1;
16     end process;
17 end Behavioral;

```

Código 4: registro.vhd

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3

```

```

4  entity registro is
5      Port ( RELOJ : in STD_LOGIC;
6             RESET : in STD_LOGIC;
7             ENTRADA : in STD_LOGIC_VECTOR(3 downto 0);
8             SALIDA : out STD_LOGIC_VECTOR(3 downto 0));
9  end registro;
10
11  architecture Behavioral of registro is
12  signal valor_interno : std_logic_vector (3 downto 0);
13  begin
14      process (RELOJ, RESET, ENTRADA)
15      begin
16          if RESET = '1' then
17              valor_interno <= B"0000";
18          elsif rising_edge (RELOJ) then
19              valor_interno <= ENTRADA;
20          end if;
21      end process;
22
23      process (valor_interno)
24      begin
25          SALIDA <= valor_interno;
26      end process;
27  end Behavioral;

```

Después tenemos como resultado (después de conectar cada bloque) el siguiente diagrama (figura 7).

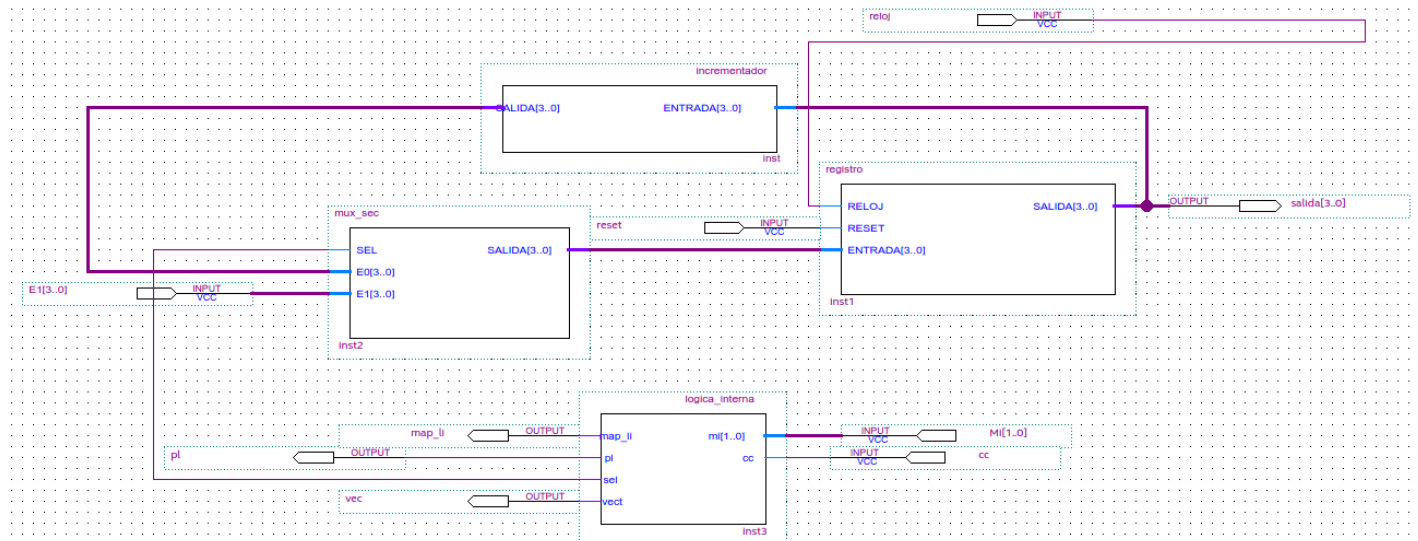


Figura 7: Diagrama de bloques del secuenciador

3.2. Carta ASM

Una vez obtenido el secuenciador, resolvemos la carta ASM objetivo de la practica (figura 8)

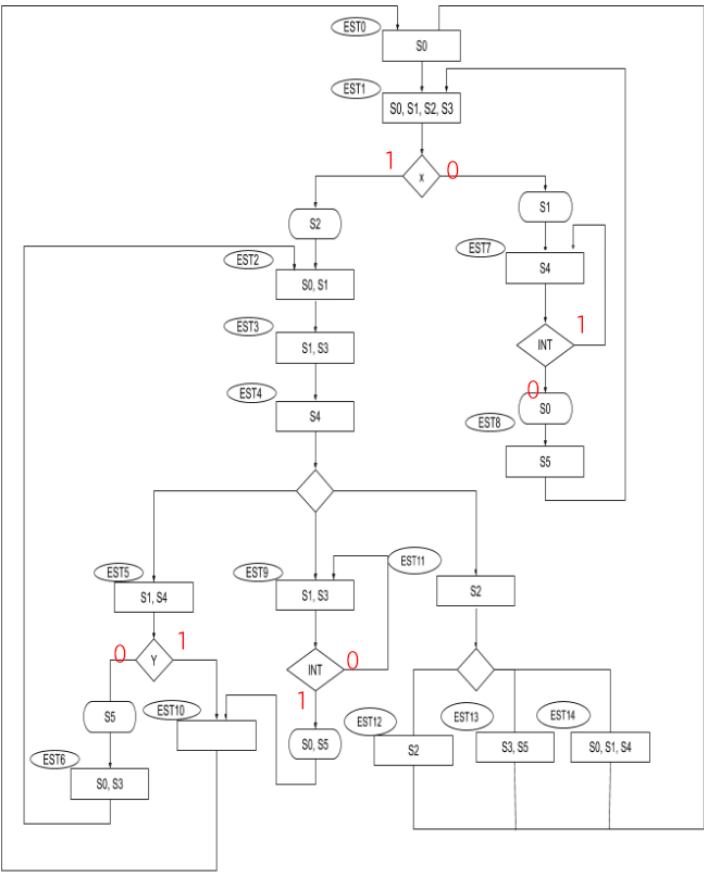


Figura 8: Carta ASM

Definimos las entradas, MI, estados (tabla 3) y la tabla de verdad de la carta ASM (tabla 5).

Tabla 3: Valores binarios de entradas y MI

Entrada			MI			Estados				
0	0	aux	C	0	0	EST0	0	0	0	0
0	1	X	SC	0	1	EST1	0	0	0	1
1	0	Y	ST	1	0	EST2	0	0	1	0
1	1	INT	SI	1	1	EST3	0	0	1	1
						EST4	0	1	0	0
						EST5	0	1	0	1
						EST6	0	1	1	0
						EST7	0	1	1	1
						EST8	1	0	0	0
						EST9	1	0	0	1
						EST10	1	0	1	0
						EST11	1	0	1	1
						EST12	1	1	0	0
						EST13	1	1	0	1
						EST14	1	1	1	0

Tabla 5: Tabla de verdad

Dirección de Memoria Estado presente				Contenido de memoria																				
				Prueba	VF	MI		Liga				Salidas Verdaderas						Salidas Falsas						
I1	I0	S5	S4			S3	S2					S1	S0	S5	S4	S3	S2	S1	S0					
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	C	
0	0	0	1	0	1	0	0	1	0	1	1	1	0	0	1	0	0	1	0	1	0	1	1	SC
0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	C
0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	1	0	0	C
0	1	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	ST
0	1	0	1	1	0	1	0	1	1	0	1	0	0	1	0	1	1	0	0	1	0	1	0	SC
0	1	1	0	0	0	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	SC
0	1	1	1	1	1	1	1	1	0	1	1	1	0	1	0	0	0	0	1	0	0	0	1	SCI
1	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	SC
1	0	0	1	1	1	0	1	1	0	0	1	1	0	1	0	1	1	0	0	1	0	1	0	SCI
1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SC
1	0	1	1	0	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	ST
1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	SC
1	1	0	1	0	0	0	0	1	0	0	0	0	1	0	1	0	0	1	0	1	0	0	0	SC
1	1	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0	1	0	0	1	1	SC

Con estos valores implementamos la memoria en Quartus (código 5):

Código 5: memoria.vhd

```

1  library ieee;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5
6  entity memoria is
7      generic(
8          data_width : natural := 21;
9          addr_length : natural := 3
10     );
11     port ( direccion : in std_logic_vector (3 downto 0);
12           m : out std_logic_vector(20 downto 0);
13           pl : in std_logic;
14           prueba: out std_logic_vector(1 downto 0);
15           vf : out std_logic;
16           mi : out std_logic_vector(1 downto 0);
17           liga : out std_logic_vector (3 downto 0);
18           salidas : out std_logic_vector(5 downto 0)
19     );
20 end memoria;
21
22 architecture behavioral of memoria is
23     signal datos: std_logic_vector (data_width-1 downto 0);
24
25     constant mem_size : natural := 15;
26     type mem_type is array (mem_size-1 downto 0) of std_logic_vector(data_width-1 downto 0);
27     constant mem : mem_type :=
28     (
29         0 => b"000000001000001000001",
30         1 => b"010010111001101001011",
31         2 => b"000000011000011000011",
32         3 => b"000000100001010001010",
33         4 => b"000100100010000010000",
34         5 => b"101011010010010110010",
35         6 => b"000010010001001001001",
36         7 => b"111110111010000010001",
37         8 => b"000010001100000100000",
38         9 => b"110111001101011001010",

```

```

39      10=> b"00001000000000000000",
40      11=> b"000101011000100000100",
41      12=> b"000010000000100000100",
42      13=> b"000010000101000101000",
43      14=> b"000010000010011010011"
44  );
45  begin
46  process(direccion,pl)
47  begin
48
49      -- entrada/prueba (20 - 19), vf (18), mi (17 - 16), liga (15 - 12), salidas_v(11-6), salida_f(5,0)
50
51
52
53      datos <= mem(to_integer(unsigned(direccion)));
54      m <= datos;
55
56
57      prueba <= datos(20 downto 19);
58      vf <= datos(18);
59      mi <= datos(17 downto 16);
60      if pl = '1' then
61          liga <= datos(15 downto 12);
62      else
63          liga <= "ZZZZ";
64      end if;
65      if (datos(18)='0') then salidas<=datos(5 downto 0); else salidas<=datos(11 downto 6); end if;
66  end process;
67  end behavioral;

```

Creamos un multiplexor de entradas (código 6) que nos ayuda a seleccionar las pruebas:

Código 6: mux_entradas.vhd

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity mux_entradas is
5      Port ( SEL : in STD_LOGIC_VECTOR(1 DOWNTO 0);
6            E0 : in STD_LOGIC;
7            E1 : in STD_LOGIC;
8            E2 : in STD_LOGIC;
9            E3 : in STD_LOGIC;
10           SALIDA : out STD_LOGIC);
11 end mux_entradas;
12
13 architecture Behavioral of mux_entradas is
14 begin
15     process (SEL, E0, E1, E2, E3)
16     begin
17         if SEL = "00" then
18             SALIDA <= E0;
19         elsif SEL = "01" then
20             SALIDA <= E1;
21         elsif SEL = "10" then
22             SALIDA <= E2;
23         elsif SEL = "11" then
24             SALIDA <= E3;
25         end if;
26     end process;
27 end Behavioral;

```

Finalizamos escribiendo los registros de transferencia y de interrupción (código 7 y 8 respectivamente) que nos ayudan a seleccionar la entrada en caso de que la prueba sea la especificada

Código 7: reg_transf.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity reg_transf is
5      Port ( RELOJ : in STD_LOGIC;
6             RESET : in STD_LOGIC;
7             MAP_LI : in STD_LOGIC;
8             ENTRADA : in STD_LOGIC_VECTOR(3 downto 0);
9             SALIDA : out STD_LOGIC_VECTOR(3 downto 0));
10 end reg_transf;
11
12 architecture Behavioral of reg_transf is
13     signal valor_interno : std_logic_vector (3 downto 0) := B"0000";
14 begin
15     process (RELOJ, RESET, ENTRADA)
16     begin
17         if RESET = '1' then
18             valor_interno <= "0000";
19         elsif rising_edge (RELOJ) then
20             valor_interno <= ENTRADA;
21         end if;
22     end process;
23
24     process (valor_interno, MAP_LI)
25     begin
26         if MAP_LI = '1' then
27             SALIDA <= valor_interno;
28         else
29             SALIDA <= "ZZZZ";
30         end if;
31     end process;
32 end Behavioral;
```

Código 8: reg_int.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity reg_int is
5      Port ( RELOJ : in STD_LOGIC;
6             RESET : in STD_LOGIC;
7             VECT : in STD_LOGIC;
8             ENTRADA : in STD_LOGIC_VECTOR(3 downto 0);
9             SALIDA : out STD_LOGIC_VECTOR(3 downto 0));
10 end reg_int;
11
12 architecture Behavioral of reg_int is
13     signal valor_interno : std_logic_vector (3 downto 0) := B"0000";
14 begin
15     process (RELOJ, RESET, ENTRADA)
16     begin
```

```

17         if RESET = '1' then
18             valor_interno <= "0000";
19         elsif rising_edge (RELOJ) then
20             valor_interno <= ENTRADA;
21         end if;
22     end process;
23
24     process (valor_interno, VECT)
25     begin
26         if VECT = '1' then
27             SALIDA <= valor_interno;
28         else
29             SALIDA <= "ZZZZ";
30         end if;
31     end process;
32
33 end Behavioral;

```

Finalizamos conectado todo registro al igual que el diagrama de la introducción (figura 2):

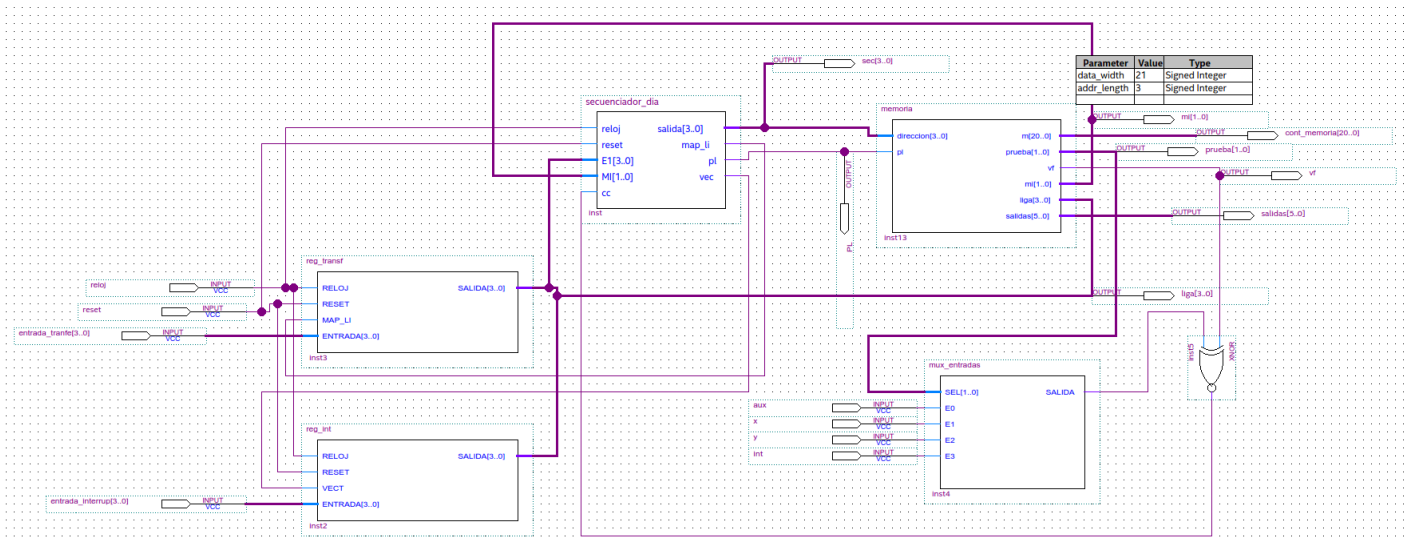


Figura 9: Esquema del secuenciador Básico

3.3. Simulación

Simulamos varias rutas para comprobar que la implementación de la carta ASM, probando los saltos de interrupción del estado 7 y el estado 9, y los saltos de transferencia de transferencia del estado 4:

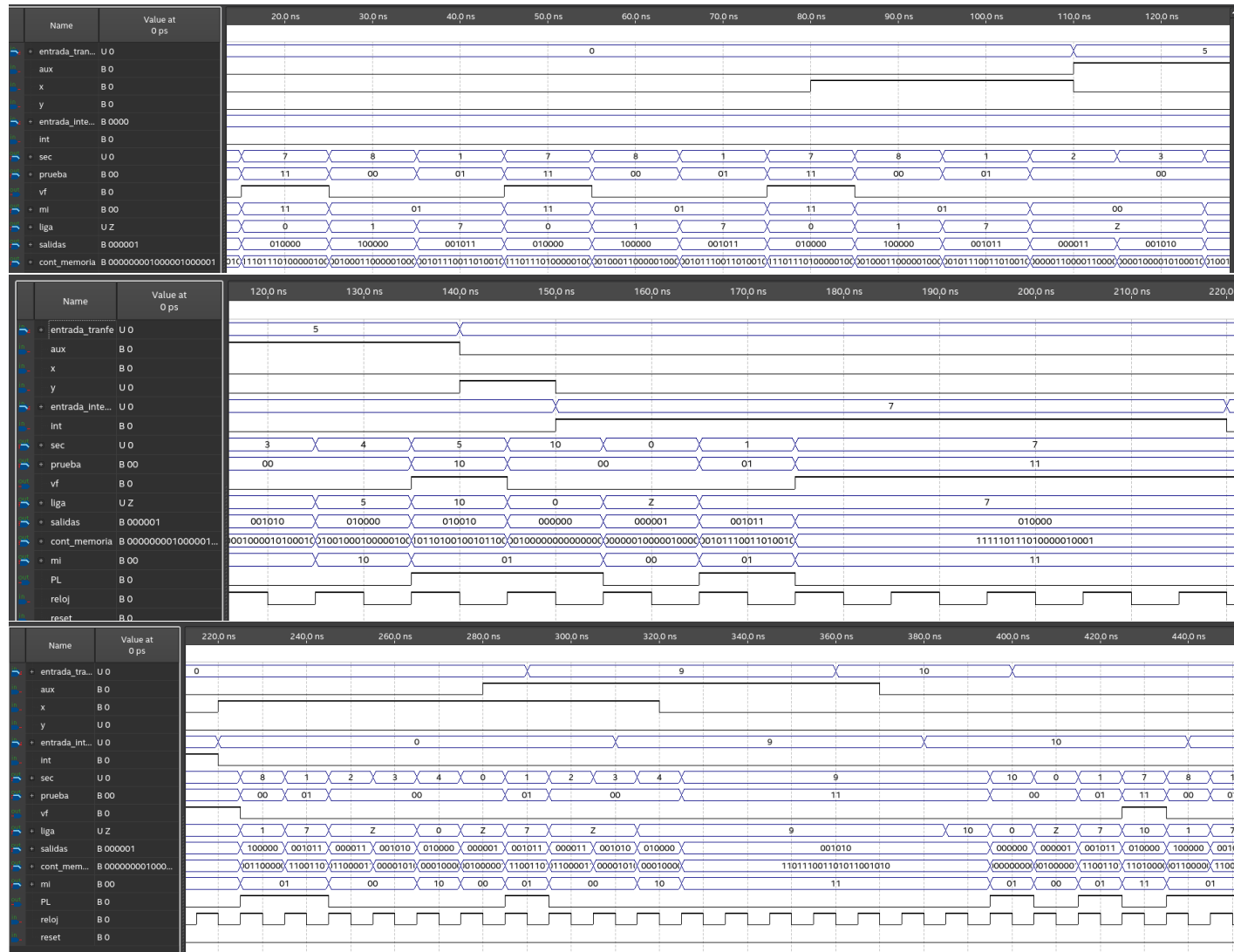


Figura 10: Simulaciones

4. Conclusiones

Monsalvo Bolaños Melissa Monserrat A través del desarrollo de esta práctica pudimos implementar un secuenciador básico. Determinamos la lógica para la asignación del estado siguiente con base en la micro instrucción seleccionada, activando el registro que provee el estado, ya sea el de interrupción, el Registro de micro-programa, el registro de transformación o la liga.

Así mismo, se decidió tomar el “camino largo” para el desarrollo de la práctica, de este modo, pudimos conocer a profundidad las partes involucradas en el desarrollo del secuenciador.

Romero Andrade Cristian Esta practica estaba dividida en dos partes, en la cual se tenia que diseñar la lógica del secuenciador y aparte pasar y decir que acciones tomar al secuenciador. Se implemento la carta ASM usando las MI que nosotros definimos e implementamos para desarrollar la practica.

Referencias

Chavez, N. E. (s.f). Construcción de máquinas de estados usando memorias. <http://profesores.fi-b.unam.mx/normaelva>
Savage, J. & Vázquez, G. (s.f). *Diseño de microprocesadores*. Facultad de Ingenieria.

Índice de tablas

1	Lógica interna (Microinstrucciones)	6
3	Valores binarios de entradas y MI	9
5	Tabla de verdad	10

Índice de figuras

1	Diagrama de bloques interno de un secuenciador básico	3
2	Diagrama de bloques interno de un secuenciador básico conectado a memoria	4
3	Carta ASM de Paso contiguo	4
4	Carta ASM de Salto Condicional	5
5	Carta ASM de Salto de Transformación	5
6	Carta ASM de Salto de Interrupción	5
7	Diagrama de bloques del secuenciador	8
8	Carta ASM	9
9	Esquema del secuenciador Básico	13
10	Simulaciones	14

Índice de Códigos

1.	logica_interna.vhd	6
2.	mux_sec.vhd	7

3.	incrementador.vhd	7
4.	registro.vhd	7
5.	memoria.vhd	10
6.	mux_entradas.vhd	11
7.	reg_transf.vhd	12
8.	reg_int.vhd	12