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

Fecha de Entrega
21 de octubre de 2021

Índice

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.

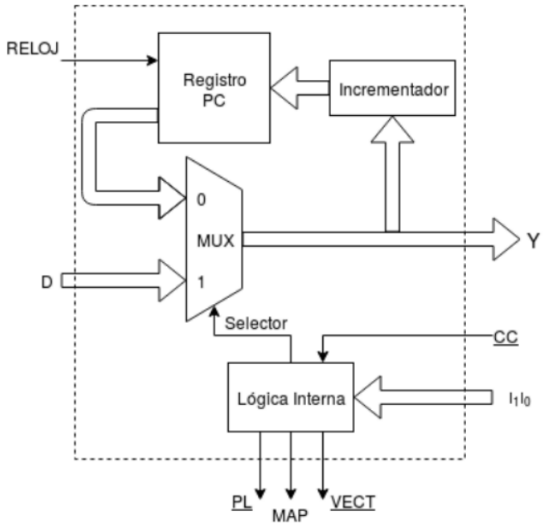

Figura 1: Diagrama de bloques interno de un secuenciador básico

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 $\underline{CC}$, la lógica es capaz de seleccionar entre la salida del registro μPC o la entrada D.

La lógica interna también genera las líneas $\underline{PL}$, $\underline{MAP}$ y $\underline{VECT}$, 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.

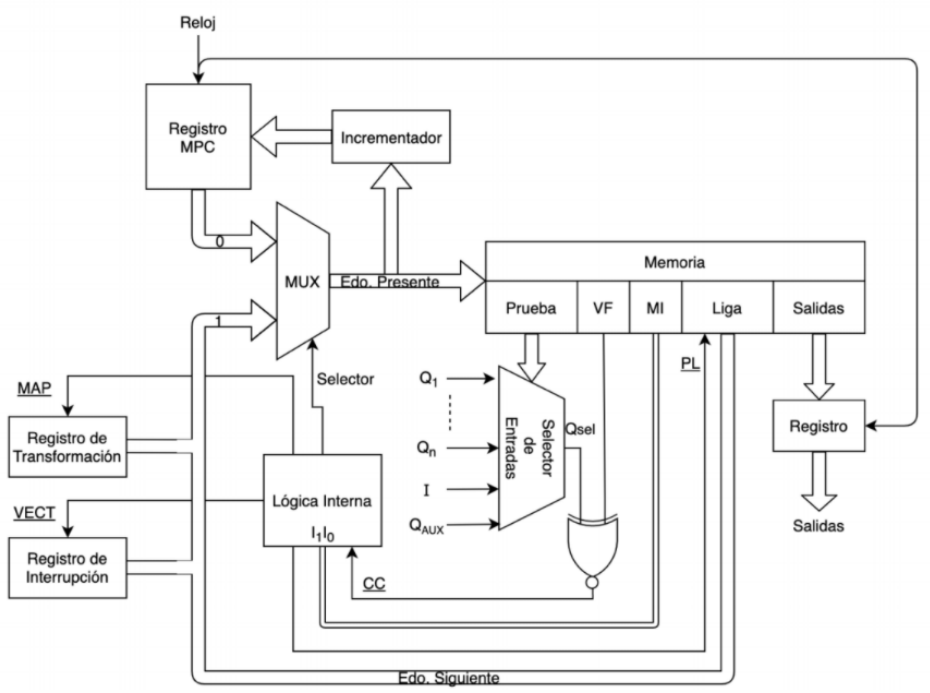

Figura 2: Diagrama de bloques interno de un secuenciador básico conectado a memoria

2.1. Paso Contiguo (C) [00]

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

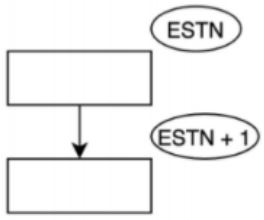

Figura 3: Carta ASM de Paso contiguo

2.2. Salto Condicional (SC) [01]

En esta instrucción se revisa el valor de la línea $\underline{CC}$, 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 $\underline{PL}$, ingresa a través de la entrada D.

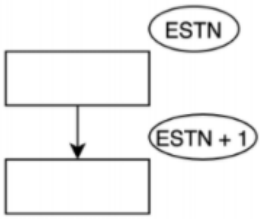

Figura 4: Carta ASM de Salto Condicional

2.3. Salto de Tranformación (ST) [10]

La dirección del estado siguiente se obtiene del registro seleccionado por la línea de $\underline{MAP}$. 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 $\underline{MAP}$.

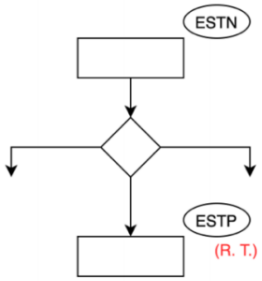

Figura 5: Carta ASM de Salto de Transformación

2.4. Salto de Interrupción (SI) [11]

En esta instrucción se revisa el valor de $\underline{CC}$, 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 $\underline{VECT}$, ingresa a través de la entrada D.

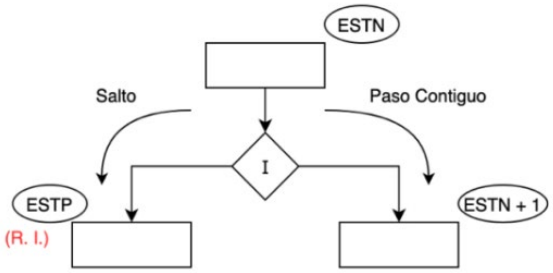

Figura 6: Carta ASM de Salto de Interrupción

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 microinstrucciónes 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';
```

```
    elsif (mi = "11" and cc = '1') then
           pl <= '0';map_li <= '0';vect <= '1';sel <= '1';
    end if;
  end process;
end Behavioral;
```

Código 2: `mux_sec.vhd`

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux_sec is
        Port (  SEL : in STD_LOGIC;
                        E0 : in STD_LOGIC_VECTOR(3 DOWNTO 0);
                        E1 : in STD_LOGIC_VECTOR(3 DOWNTO 0);
                        SALIDA  : out STD_LOGIC_VECTOR(3 DOWNTO 0));
end mux_sec;

architecture Behavioral of mux_sec is
begin
        process (SEL, E0, E1)
        begin
                if SEL = '0' then
                        SALIDA <= E0;
                elsif SEL = '1' then
                        SALIDA <= E1;
                end if;
        end process;
end Behavioral;
```

Código 3: `incrementador.vhd`

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity incrementador is
        Port (  ENTRADA : in STD_LOGIC_VECTOR(3 downto 0);
                        SALIDA  : out STD_LOGIC_VECTOR(3 downto 0));
end incrementador;

architecture Behavioral of incrementador is
begin
        process (ENTRADA)
        begin
                SALIDA <= ENTRADA + 1;
        end process;
end Behavioral;
```

Código 4: `registro.vhd`

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

```

```vhdl
entity registro is
        Port (  RELOJ : in STD_LOGIC;
                        RESET : in STD_LOGIC;
                        ENTRADA  : in STD_LOGIC_VECTOR(3 downto 0);
                        SALIDA : out STD_LOGIC_VECTOR(3 downto 0));
end registro;

architecture Behavioral of registro is
signal valor_interno : std_logic_vector (3 downto 0);
begin
        process (RELOJ, RESET, ENTRADA)
        begin
                if RESET = '1' then
                        valor_interno <= B"0000";
                elsif rising_edge (RELOJ) then
                        valor_interno <= ENTRADA;
                end if;
        end process;

        process (valor_interno)
        begin
                SALIDA <= valor_interno;
        end process;
end Behavioral;
```

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

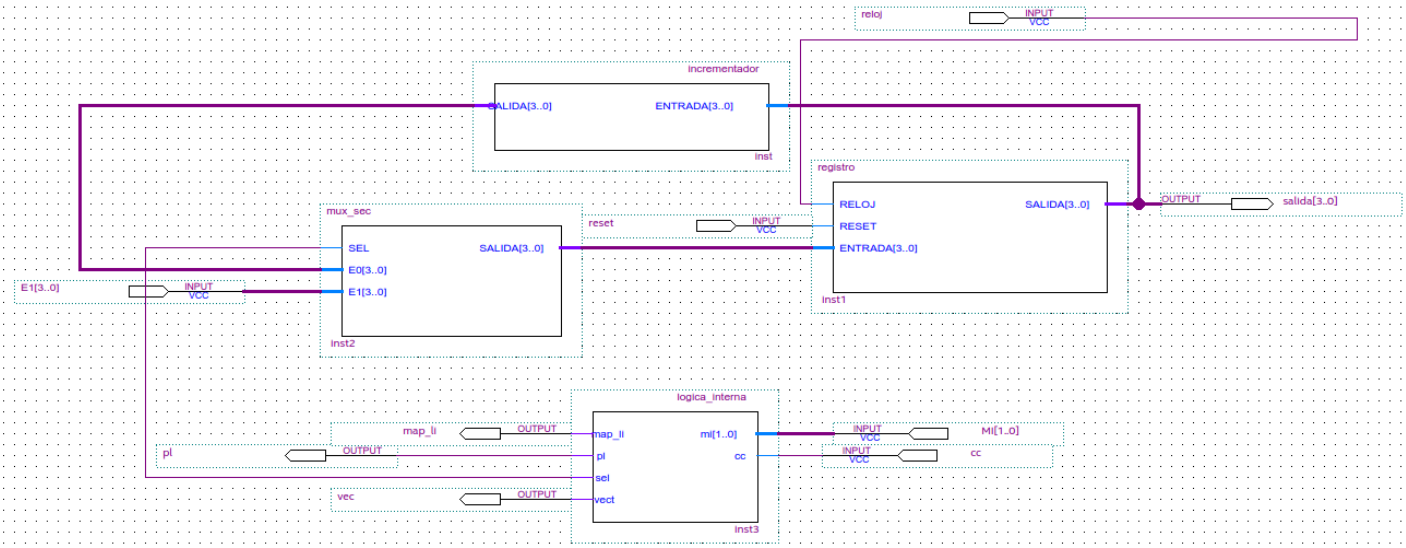

Figura 7: Diagrama de bloques del secuenciador

3.2. Carta ASM

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

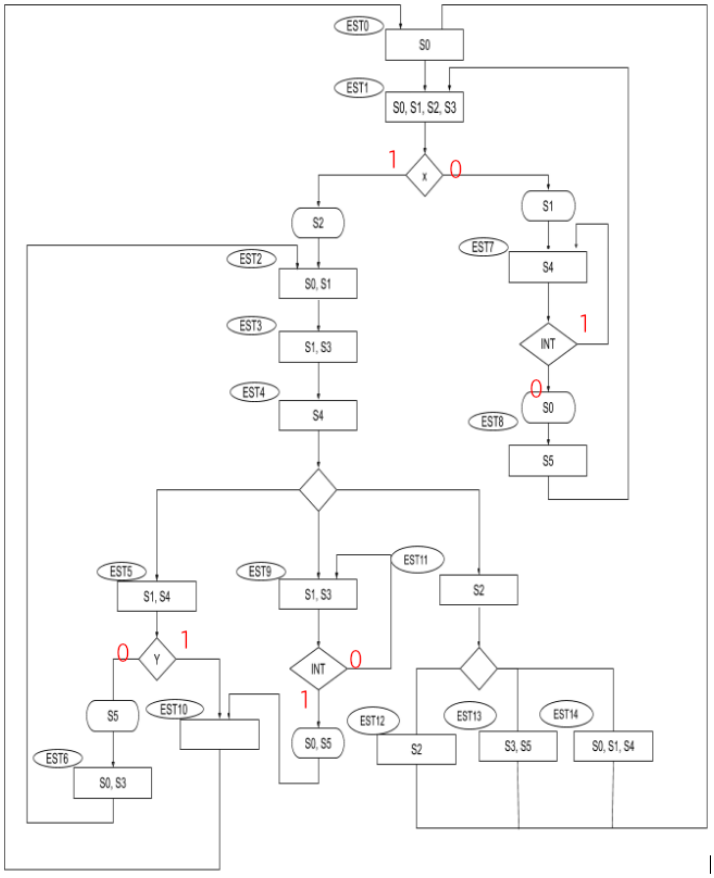

Figura 8: Carta ASM

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		
0	0	aux
0	1	X
1	0	Y
1	1	INT

MI		
C	0	0
SC	0	1
ST	1	0
SI	1	1

Estados				
EST0	0	0	0	0
EST1	0	0	0	1
EST2	0	0	1	0
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				Contenido de memoria																						
Estado presente				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	1	0	0	0	0	0	1	0	0	0	0	0	1	C
0	0	0	1	0	1	0	0	1	0	1	1	1	0	0	1	1	0	1	0	0	1	0	1	1	SC
0	0	1	0	0	0	0	0	0	0	0	1	1	0	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	1	0	0	0	1	0	1	0	C
0	1	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	ST
0	1	0	1	1	0	1	0	1	1	0	1	0	0	1	0	0	1	0	1	1	0	0	1	0	SC
0	1	1	0	0	0	0	0	1	0	0	1	0	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	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	0	1	0	0	0	0	0	SC
1	0	0	1	1	1	0	1	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	0	SC
1	0	1	1	0	0	0	1	0	1	0	1	1	0	0	0	1	0	0	0	0	0	1	0	0	ST
1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	SC
1	1	0	1	0	0	0	0	1	0	0	0	0	1	0	1	0	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	0	1	1	0	1	0	0	1	1	SC

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

Código 5: `memoria.vhd`

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


entity memoria is
  generic(
    data_width : natural := 21;
    addr_length : natural := 3
    );
  port ( direccion : in std_logic_vector (3 downto 0);
         m : out std_logic_vector(20 downto 0);
         pl : in std_logic;
         prueba: out std_logic_vector(1 downto 0);
         vf : out std_logic;
         mi : out std_logic_vector(1 downto 0);
         liga : out std_logic_vector (3 downto 0);
         salidas : out std_logic_vector(5 downto 0)
                      );
end memoria;

architecture behavioral of memoria is
  signal datos: std_logic_vector (data_width-1 downto 0);

  constant mem_size : natural := 15;
  type mem_type is array (mem_size-1 downto 0) of std_logic_vector(data_width-1 downto 0);
  constant mem : mem_type :=
  (
    0 => b"000000001000001000001",
    1 => b"010010111001101001011",
    2 => b"000000011000011000011",
    3 => b"000000100001010001010",
    4 => b"000100100010000010000",
    5 => b"101011010010010110010",
    6 => b"000010010001001001001",
    7 => b"111110111010000010001",
    8 => b"000010001100000100000",
    9 => b"110111001101011001010",
```

```
    10=> b"00001000000000000000",
    11=> b"00010101100010000100",
    12=> b"00001000000100000100",
    13=> b"00001000010100010100",
    14=> b"00001000001001101001"
  );
  begin
  process(direccion,pl)
  begin

    -- entrada/prueba (20 - 19), vf (18), mi (17 - 16), liga (15 - 12), salidas_v(11-6), salida_f(5,0)



    datos <= mem(to_integer(unsigned(direccion)));
         m <= datos;


    prueba <= datos(20 downto 19);
    vf <= datos(18);
    mi <= datos(17 downto 16);
    if pl = '1' then
      liga <= datos(15 downto 12);
    else
      liga <= "ZZZZ";
    end if;
    if (datos(18)='0') then salidas<=datos(5 downto 0); else salidas<=datos(11 downto 6); end if;
  end process;
end behavioral;
```

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

Código 6: `mux_entradas.vhd`

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux_entradas is
        Port (  SEL : in STD_LOGIC_VECTOR(1 DOWNTO 0);
                        E0 : in STD_LOGIC;
                        E1 : in STD_LOGIC;
                        E2 : in STD_LOGIC;
                        E3 : in STD_LOGIC;
                        SALIDA  : out STD_LOGIC);
end mux_entradas;

architecture Behavioral of mux_entradas is
begin
        process (SEL, E0, E1, E2, E3)
        begin
                if SEL = "00" then
                        SALIDA <= E0;
                elsif SEL = "01" then
                        SALIDA <= E1;
                elsif SEL = "10" then
                        SALIDA <= E2;
                elsif SEL = "11" then
                        SALIDA <= E3;
                end if;
        end process;
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`

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity reg_transf is
        Port (  RELOJ : in STD_LOGIC;
                          RESET : in STD_LOGIC;
                          MAP_LI : in STD_LOGIC;
                          ENTRADA  : in STD_LOGIC_VECTOR(3 downto 0);
                          SALIDA : out STD_LOGIC_VECTOR(3 downto 0));
end reg_transf;

architecture Behavioral of reg_transf is
        signal valor_interno : std_logic_vector (3 downto 0) := B"0000";
begin
        process (RELOJ, RESET, ENTRADA)
        begin
                if RESET = '1' then
                        valor_interno <= "0000";
                elsif rising_edge (RELOJ) then
                        valor_interno <= ENTRADA;
                end if;
        end process;

        process (valor_interno, MAP_LI)
        begin
                if MAP_LI = '1' then
                        SALIDA <= valor_interno;
                else
                        SALIDA <= "ZZZZ";
                end if;

        end process;

end Behavioral;
```

Código 8: `reg_int.vhd`

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity reg_int is
        Port (  RELOJ : in STD_LOGIC;
                          RESET : in STD_LOGIC;
                          VECT : in STD_LOGIC;
                          ENTRADA  : in STD_LOGIC_VECTOR(3 downto 0);
                          SALIDA : out STD_LOGIC_VECTOR(3 downto 0));
end reg_int;

architecture Behavioral of reg_int is
        signal valor_interno : std_logic_vector (3 downto 0) := B"0000";
begin
        process (RELOJ, RESET, ENTRADA)
        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):

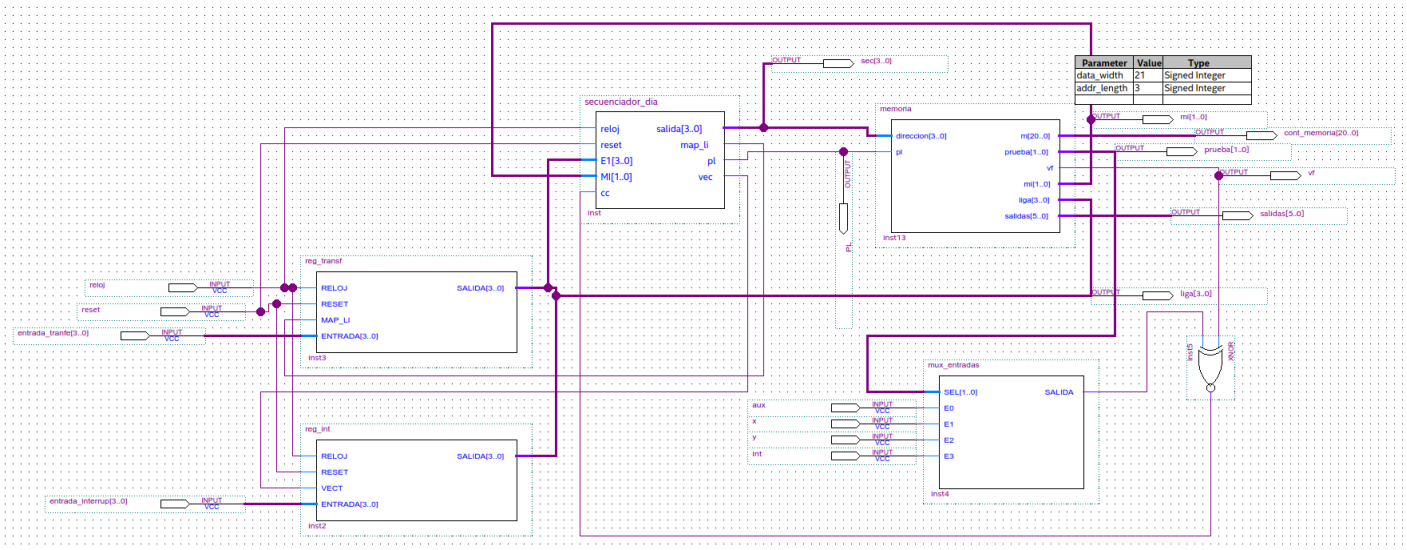

Figura 9: Esquema del secuenciador Básico

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:

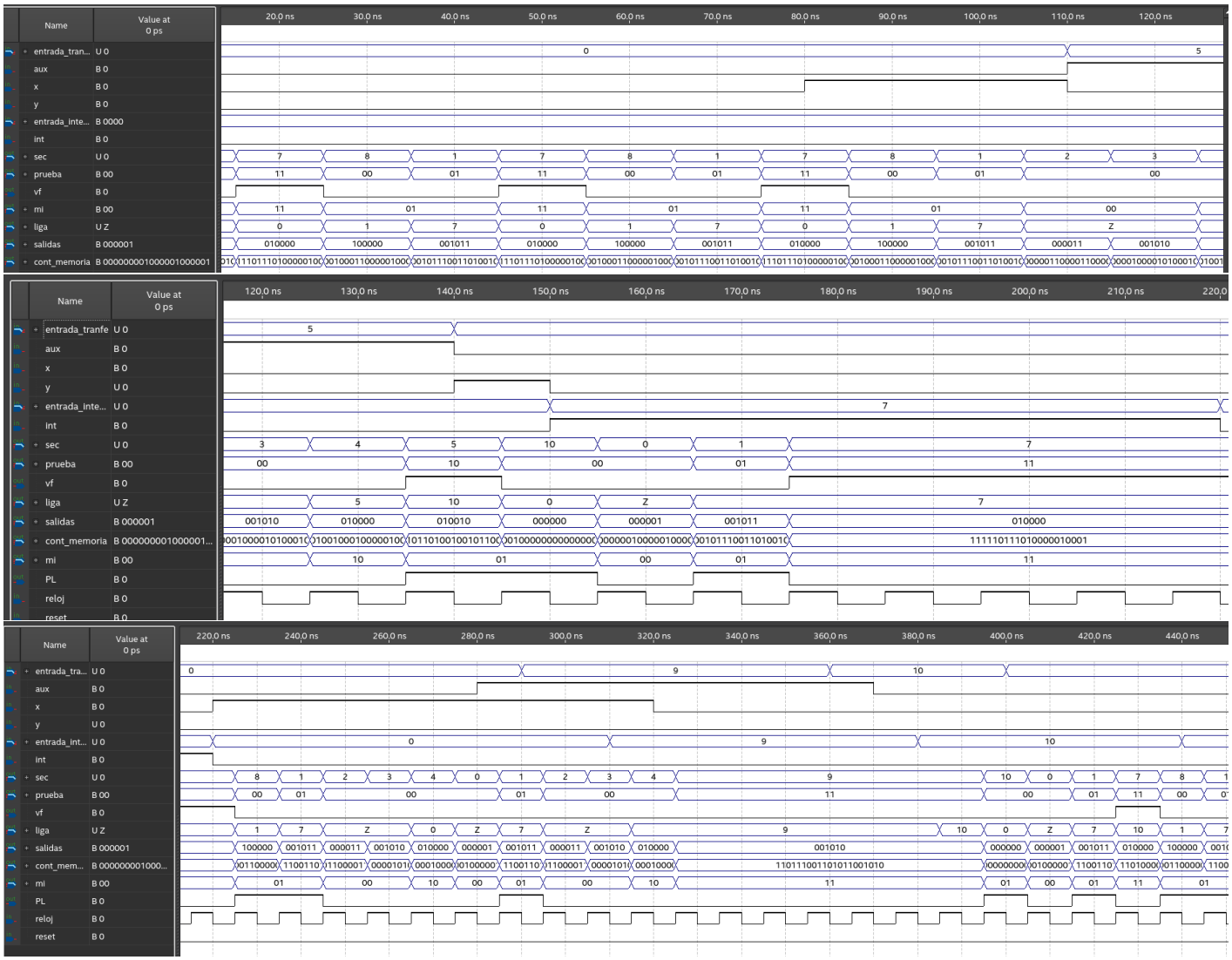

Figura 10: Simulaciones

4. Conclusiones

Monsalvo Bolaños Melissa Monserrat

Romero Andrade Cristian

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

Índice de figuras

Índice de Códigos