

Universidad Nacional Autónoma de México

Facultad de Ingeniería

Ingeniería en Computación

Laboratorio de Organización y Arquitectura de Computadoras

Práctica 06

Alumnos:

- Murrieta Villegas Alfonso
- Reza Chavarria Sergio Gabriel
- Valdespino Mendieta Joaquin

Profesora: Ayesha Sagrario Román García

Grupo: 7

Fecha de entrega: 5 de noviembre de 2021

Práctica 06

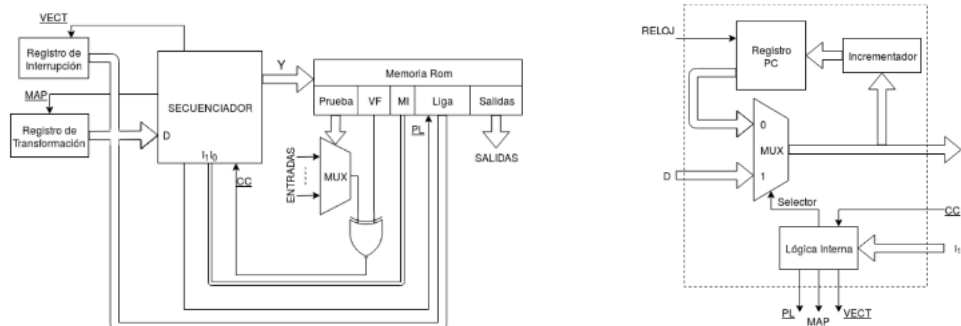
Objetivo

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

Introducción

El secuenciador básico es una forma ampliada para implementar máquinas de estados, cabe recalcar que está basada en el direccionamiento implícito agregando elementos en memoria y elementos de hardware, como los bits asignados para los microinstrucciones que son operaciones específicas para llevar a cabo la transición entre estados.

El secuenciador básico de cuatro microinstrucciones cuenta con una estructura de memoria, una estructura principal del secuenciador que contiene el registro del Program Counter, incrementador y un multiplexor, además de dos registros para las interrupciones y transformaciones (direcciones), observe la figura 1



(Secuenciador básico de 4 micro instrucciones)

Como se menciona con tal estructura en memoria, con dos bits podemos representar un total de 4 microinstrucciones diferentes, estas instrucciones se encuentran listadas a continuación dadas la combinación de I1 e I0.

00 – Paso Continuo: esta instrucción procede a transicional a la dirección del estado siguiente dada por el registro del Microprograma

01- Salto Condicional: esta instrucción con la evaluación de la entrada y la dirección asignada de la liga, además del valor falso pasa a una al estado siguiente dado por el valor del registro de la microprograma o un salto a la dirección dada por la secuencia de bits de la liga (\overline{PL}).

10- Salto de Transformación: esta instrucción, realiza una transición al estado determinado por el registro de transformación, A través el bus \overline{MAP} , tiene la capacidad de ir a cualquier estado marcado.

11- Salto de Interrupción: esta instrucción, con el valor \overline{CC} , evalúa, si es 0, transición al estado siguiente dado por el registro de microprograma, si no, entonces cambia al estado dado por el registro de interrupción, a través del bus \overline{VECT} .

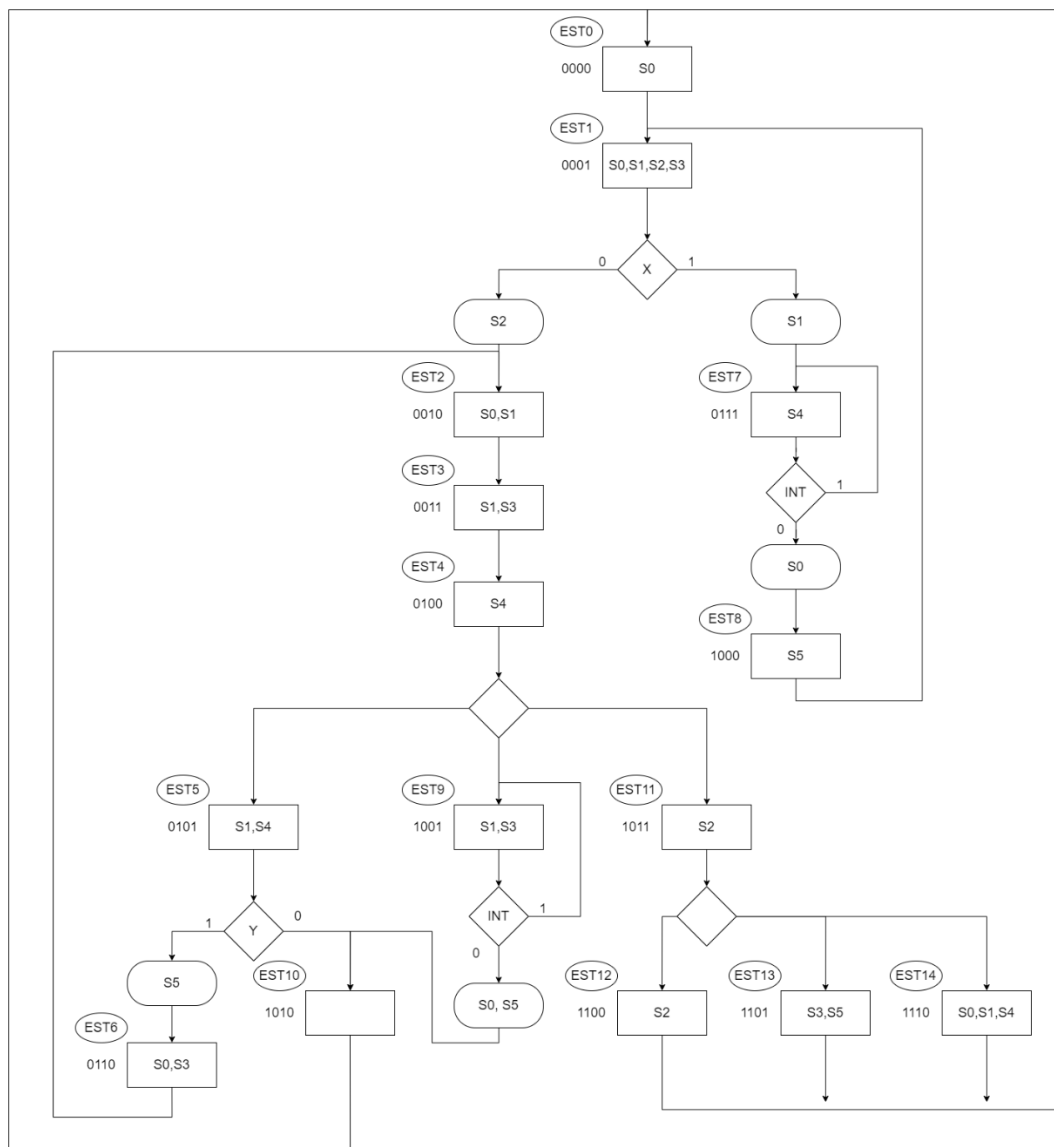
El comportamiento de este secuenciador se puede observar en la siguiente tabla de verdad, tomando en cuenta que \overline{CC} es el resultado de una XOR.

I_1, I_0, \overline{CC}	Selector	\overline{PL}	\overline{MAP}	\overline{VECT}	Comentarios
00*	0	1	1	1	Paso continuo
010	1	0	1	1	Salto Condicional con 0
011	0	1	1	1	Paso continuo
10*	1	1	0	1	Salto de Transformación
110	1	1	1	0	Salto por interrupción
111	0	1	1	1	Paso Continuo

Cabe aclarar que los registros \overline{PL} , \overline{MAP} , \overline{VECT} no pueden estar en bajo a la vez, ya que ocasionaría un corto circuito al secuenciador físico.

Desarrollo

En la figura 7 se presenta una carta ASM en donde se hace uso de todas las instrucciones que un secuenciador básico puede ejecutar. En el estado EST4 y EST11 la dirección del estado siguiente está determinada por el registro de transformación, seleccionado cuando el secuenciador ejecuta la instrucción ST. En el estado EST7 y EST9 la dirección del estado siguiente la proporciona el registro de interrupción.



Construya el secuenciador descrito en el punto 1 y conéctelo a una memoria tal como se muestra en el punto 2, utilizando VHDL y componentes estándares del software de desarrollo. Diseñe la lógica interna para que su secuenciador pueda ejecutar las instrucciones descritas anteriormente. Obtenga el contenido de la memoria de la carta ASM de la figura 7 y impleméntelo en su secuenciador. Utilice el simulador para probar su implementación.

Primero, obtuvimos la información de la memoria que tendrá el sistema. Esto al obtener su tabla de verdad con la información proporcionada por la carta ASM.

Para la representación del estado actual y las ligas se utilizarán 4 bits, esto para contar los 15 estados de la carta ASM.

Para la representación de las entradas se utilizaron 2 bits, esto para representar la entrada X, Y, la interrupción y Qaux (para los saltos auxiliares). Para las salidas (S0-S5) se utilizarán 6 bits, uno correspondiente a cada salida. Debido al uso de salidas condicionales se recurrirá a la modificación del sistema para obtener salidas verdaderas y salidas falsas.

Tabla de verdad

Para la representación de las entradas se utilizará la siguiente codificación binaria:

- $Q_{aux} = 00$
- $X = 01$
- $Y = 10$
- $INT = 11$

Para los microinstrucciones se utilizó la siguiente codificación:

- 00 Paso Continuo
- 01 Salto Condicional
- 10 Salto de Transformación
- 11 Salto por Interrupción

A continuación, se presenta la tabla de verdad obtenida.

Estado	Estado Presente P_3, P_2, P_1, P_0	Liga L_3, L_2, L_1, L_0	Micro Instrucción I_1, I_0	Prueba E_1, E_0	VF	Salida verdadera $S_5, S_4, S_3, S_2, S_1, S_0$	Salida Falsa $Z_5, Z_4, Z_3, Z_2, Z_1, Z_0$
0	0000	****	00	**	*	000001	000001
1	0001	0111	01	01	1	001111	001111
2	0010	****	00	**	*	000011	000011
3	0011	****	00	**	*	001010	001010
4	0100	****	10	**	*	010000	010000
5	0101	1010	01	10	0	110010	010010
6	0110	0010	01	00	0	001001	001001
7	0111	****	11	11	1	010000	010001
8	1000	0001	01	00	0	100000	100000
9	1001	****	11	11	1	001010	101011
10	1010	0000	01	00	0	000000	000000
11	1011	****	10	**	*	000100	000100
12	1100	0000	01	00	0	000100	000100
13	1101	0000	01	00	0	101000	101000
14	1110	0000	01	00	0	010011	010011
15	1111	****	00	**	*	000000	000000

Nota: Al asignar en la memoria, los asteriscos de la tabla se asignarán como 0. La memoria contendrá 15 registros de 21 bits.

Una vez obtenida la tabla de verdad, se programó en VHDL los componentes para la creación del secuenciador, la memoria ROM y el sistema en general.

Para el secuenciador se crearon 4 componentes. El bloque de lógica interna nos permitirá la obtención de información por las salidas del selector, \overline{PL} , \overline{MAP} y \overline{VECT} . El multiplexor nos permitirá seleccionar la respuesta correspondiente, si viene de los registros de externos o del contador, a la lógica interna. El incrementador aumentará el valor de los estados si se hace un paso continuo. El registro que nos permitirá obtener la información de estado, esto dependiendo del reloj interno.

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity secuenciador is port(
5      clk: in std_logic;
6      cc: in std_logic;
7      i: in std_logic_vector(1 downto 0);
8      D: in std_logic_vector(3 downto 0);
9      PL: out std_logic;
10     MP: out std_logic;
11     vect: out std_logic;
12     Y: out std_logic_vector(3 downto 0)
13 );
14 end secuenciador;
15
16 architecture arc_sec of secuenciador is
17     signal sel: std_logic;
18     signal entry: std_logic_vector(3 downto 0);
19     signal sal: std_logic_vector(3 downto 0);
20     signal y_inc: std_logic_vector(3 downto 0);
21 begin
22     log: entity work.internal_logic(arq_int_log) port map(i,cc,sel,PL,MP,vect);
23     reg: entity work.registro(arq_reg) port map(entry,clk,'1',sal);
24     mux: entity work.mux2(arq_mux) port map (sal, D, sel,y_inc);
25     inc: entity work.incrementador(arq_inc) port map(y_inc,entry);
26
27     Y<=y_inc;
28 end architecture;

```

Código 1: Bloque correspondiente al secuenciador

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_UNSIGNED.all;
4
5  entity incrementador is port(
6      ent: in std_logic_vector(3 downto 0);
7      sal: out std_logic_vector(3 downto 0)
8  );
9  end incrementador;
10
11 architecture arq_inc of incrementador is
12 begin
13     sal<=ent+1;
14 end architecture;

```

Código 2: Incrementador del estado actual

```

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 internal_logic is port(
7      i: in std_logic_vector(1 downto 0);
8      ncc: in std_logic;
9      selector: out std_logic;
10     npl: out std_logic;
11     nmap: out std_logic;
12     nvect: out std_logic
13 );
14 end internal_logic;
15
16 architecture arq_int_log of internal_logic is
17     signal info: std_logic_vector(3 downto 0);
18 begin
19     process (i, ncc)
20     begin
21         --Paso continuo
22         if (i="00" and ncc='0') then
23             info <= "0111";
24         --Salto condicional
25         elsif (i="01" and ncc='0') then
26             info <= "1011";
27         --Paso continuo
28         elsif (i="01" and ncc='1') then
29             info <= "0111";
30         --Salto de Transformacion
31         elsif (i="10") then
32             info <= "1101";
33         --Salto por interrupcion
34         elsif (i="11" and ncc='0') then
35             info <= "1110";
36         --Paso Continuo
37         elsif (i="11" and ncc='1') then
38             info <= "0111";
39
40         else
41             info <= "0111";
42         end if;
43     end process;
44
45     process(info)
46     begin
47         selector<=info(3);
48         npl<= not(info(2));
49         nmap<= not(info(1));
50         nvect<= not(info(0));
51
52     end process;
53 end architecture;
54
55

```

Código 3: Lógica Interna del secuenciador


```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity mux2 is port(
5      sal,D: in std_logic_vector(3 downto 0);
6      sel: in std_logic;
7      y: out std_logic_vector(3 downto 0)
8  );
9  end mux2;
10
11  architecture arq_mux of mux2 is
12  begin
13      with sel select
14          y<=      sal      when '0',
15                  D        when '1',
16                  "0000"   when others;
17  end architecture;

```

Código 4: Multiplexor del secuenciador

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity registro is port(
5      entrada: in std_logic_vector(3 downto 0);
6      clk: in std_logic;
7      reset: in std_logic;
8      salida: out std_logic_vector(3 downto 0)
9  );
10  end registro;
11
12  architecture arq_reg of registro is
13  signal info: std_logic_vector(3 downto 0):=B"0000";
14  begin
15      process (clk, reset, entrada)
16      begin
17          if reset='0' then
18              info <= "0000";
19          elsif reset='1' then
20              if (clk' event and clk='1') then
21                  info <= entrada;
22              end if;
23          end if;
24      end process;
25
26      process (info)
27      begin
28          salida <= info;
29      end process;
30  end architecture;

```

Código 5: Registro del secuenciador (Paso Continuo)

Una vez obtenida la conexión entre los componentes del secuenciador se implementaron los elementos como la memoria ROM, donde se guardará la información de la tabla de verdad. Esto se hizo a partir de la configuración de un archivo externo (*.mif) para la información.

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_UNSIGNED.all;
4
5
6 entity ROM is port(
7     address: in std_logic_vector(3 downto 0);
8     cs: in std_logic;
9     data: out std_logic_vector(20 downto 0)
10 );
11 end ROM;
12 architecture arcROM of ROM is
13     type memory is array (14 downto 0) of std_logic_vector(20 downto 0);
14     signal mem: memory;
15     attribute ram_init_file: string;
16     attribute ram_init_file of mem: signal is "memoria.mif";
17
18     signal dato: std_logic_vector(20 downto 0);
19
20 begin
21     process(address)
22     begin
23         dato <= mem(conv_integer(address));
24     end process;
25
26     process(dato, cs)
27     begin
28         if (cs='1') then
29             data<= dato;
30         else
31             data<=(others=>'z');
32         end if;
33     end process;
34 end arcROM;
35

```

Código 6: Memoria ROM con archivo externo

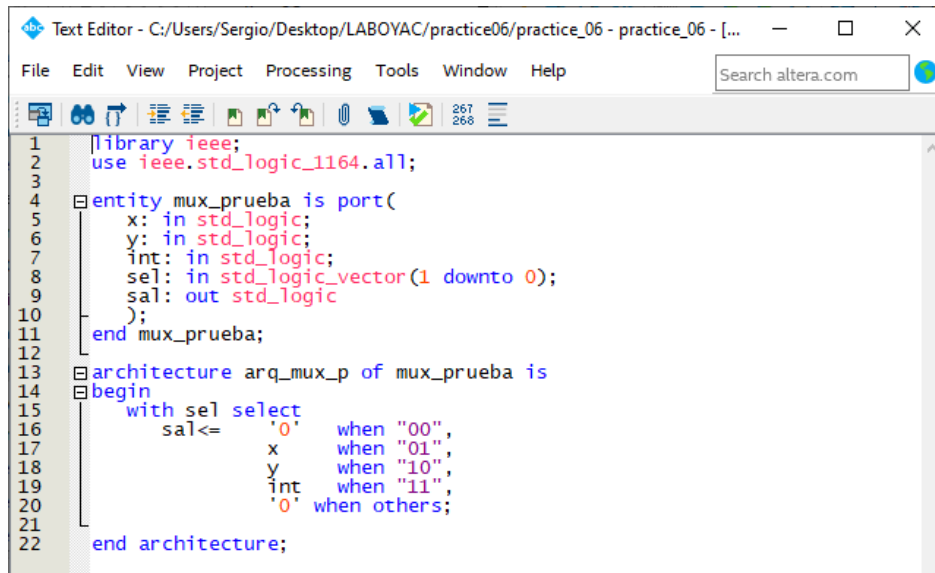
```

1 -- Copyright (C) 2020 Intel Corporation. All rights reserved.
2 -- Your use of Intel Corporation's design tools, logic functions
3 -- and other software and tools, and any partner logic
4 -- functions, and any output files from any of the foregoing
5 -- (including device programming or simulation files), and any
6 -- associated documentation or information are expressly subject
7 -- to the terms and conditions of the Intel Program License
8 -- Subscription Agreement, the Intel Quartus Prime License Agreement,
9 -- the Intel FPGA IP License Agreement, or other applicable license
10 -- agreement, including, without limitation, that your use is for
11 -- the sole purpose of programming logic devices manufactured by
12 -- Intel and sold by Intel or its authorized distributors. Please
13 -- refer to the applicable agreement for further details, at
14 -- https://fpgasoftware.intel.com/eula.
15
16 -- Quartus Prime generated Memory Initialization File (.mif)
17
18 WIDTH=21;
19 DEPTH=15;
20
21 ADDRESS_RADIX=BIN;
22 DATA_RADIX=BIN;
23
24 CONTENT BEGIN
25     0000 : 00000000000001000001;
26     0001 : 0111011001111001111;
27     0010 : 0000000000011000011;
28     0011 : 00000000001010001010;
29     0100 : 00001000001000010000;
30     0101 : 101001100110010010010;
31     0110 : 00100100001001001001;
32     0111 : 00001111101000010001;
33     1000 : 000101000100000100000;
34     1001 : 000011111001010101011;
35     1010 : 00000100000000000000;
36     1011 : 000010000000100000100;
37     1100 : 000001000000100000100;
38     1101 : 000001000101000101000;
39     1110 : 000001000010011010011;
40 END;
41

```

*Código 7: Archivo *.mif con la información de la memoria ROM*

Se realizaron 2 multiplexores, uno para la selección de las entradas al sistema y otro para la selección entre la salida verdadera y falsa. El selector del primer multiplexor estará conectado con la sección de prueba de la memoria, donde se tendrá Q_{Aux} , X , Y e *Interrupción*.



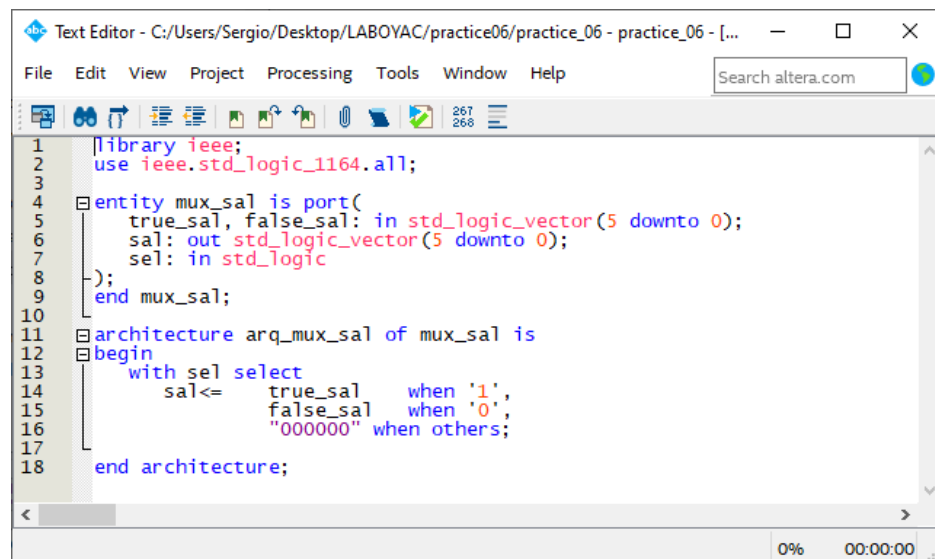
```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity mux_prueba is port(
5      x: in std_logic;
6      y: in std_logic;
7      int: in std_logic;
8      sel: in std_logic_vector(1 downto 0);
9      sal: out std_logic
10 );
11 end mux_prueba;
12
13 architecture arq_mux_p of mux_prueba is
14 begin
15     with sel select
16         sal <= '0'   when "00",
17                x     when "01",
18                y     when "10",
19                int    when "11",
20                '0'   when others;
21
22 end architecture;

```

Código 8: Multiplexor de las entradas (pruebas)

El otro multiplexor funcionará con la salida de la entrada seleccionada. Si la entrada es 1 seleccionará la salida verdadera, en caso de 0 se selecciona la falsa.



```

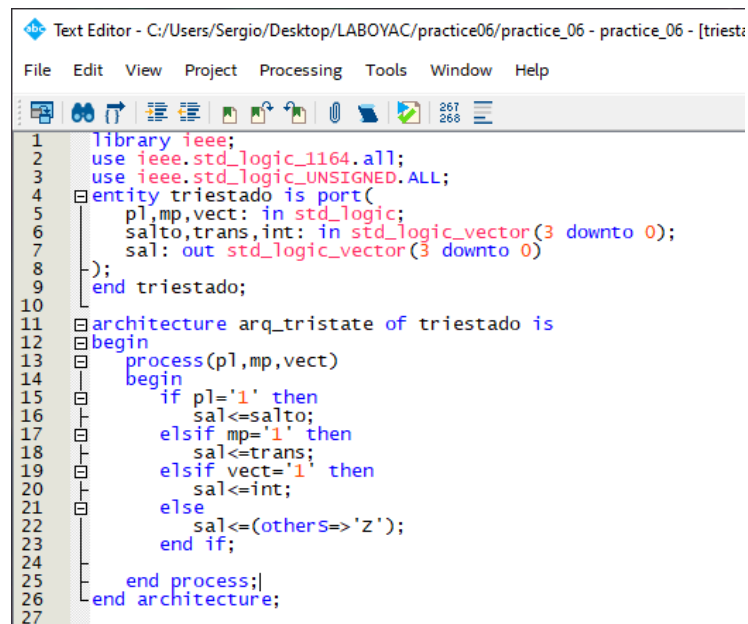
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity mux_sal is port(
5      true_sal, false_sal: in std_logic_vector(5 downto 0);
6      sal: out std_logic_vector(5 downto 0);
7      sel: in std_logic
8  );
9  end mux_sal;
10
11 architecture arq_mux_sal of mux_sal is
12 begin
13     with sel select
14         sal <= true_sal   when '1',
15                false_sal  when '0',
16                "000000"   when others;
17
18 end architecture;

```

Código 9: Multiplexor de la salida verdadera y falsa

Se utilizó el registro creado para el secuenciador para representar los registros de interrupción y de transformación. Estos tendrán una entrada para que en la simulación se seleccione el estado seleccionado.

Se creó un elemento extra para elegir la entrada de los registros dependiendo de las salidas de la lógica interna. Si \overline{PL} es activado la entrada del salto condicional, si es \overline{MAP} es activado la entrada del registro de transformación y si es \overline{VECT} es activado la entrada del registro de interrupción.



```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_UNSIGNED.ALL;
4  entity triestado is port(
5      pl,mp,vect: in std_logic;
6      salto,trans,int: in std_logic_vector(3 downto 0);
7      sal: out std_logic_vector(3 downto 0)
8  );
9  end triestado;
10
11 architecture arq_tristate of triestado is
12 begin
13     process(pl,mp,vect)
14     begin
15         if pl='1' then
16             sal<=salto;
17         elsif mp='1' then
18             sal<=trans;
19         elsif vect='1' then
20             sal<=int;
21         else
22             sal<=(others=>'Z');
23         end if;
24     end process;
25 end architecture;
```

Código 10: Lógica de las entradas dependiendo de la lógica interna

Por último, se crearon 4 registros para la inicialización del sistema y el guardado de información de la memoria. La lógica de estos registros es similar al registro del secuenciador, lo diferente es en la cantidad de bits de entrada y de salida.

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity reg_liga is port(
5      entrada: in std_logic_vector(3 downto 0);
6      clk: in std_logic;
7      reset: in std_logic;
8      salida: out std_logic_vector(3 downto 0)
9  );
10 end reg_liga;
11
12 architecture arq_reg_l of reg_liga is
13     signal info: std_logic_vector(3 downto 0) := B"0000";
14 begin
15     process (clk, reset, entrada)
16     begin
17         if reset='0' then
18             info <= "0000";
19         elsif reset='1' then
20             if (clk' event and clk='1') then
21                 info <= entrada;
22             end if;
23         end if;
24     end process;
25
26     process (info)
27     begin
28         salida <= info;
29     end process;
30 end architecture;
31
32

```

Código 11: Registro utilizado para la liga

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity reg_sal is port(
5      entrada: in std_logic_vector(5 downto 0);
6      clk: in std_logic;
7      reset: in std_logic;
8      salida: out std_logic_vector(5 downto 0)
9  );
10 end reg_sal;
11
12 architecture arq_reg_s of reg_sal is
13     signal info: std_logic_vector(5 downto 0) := B"000000";
14 begin
15     process (clk, reset, entrada)
16     begin
17         if reset='0' then
18             info <= "000000";
19         elsif reset='1' then
20             if (clk' event and clk='1') then
21                 info <= entrada;
22             end if;
23         end if;
24     end process;
25
26     process (info)
27     begin
28         salida <= info;
29     end process;
30 end architecture;
31
32

```

Código 12: Registro de las salidas

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity reg_1 is port(
5      entrada: in std_logic;
6      clk: in std_logic;
7      reset: in std_logic;
8      salida: out std_logic
9  );
10 end reg_1;
11
12 architecture arq_reg_01 of reg_1 is
13     signal info: std_logic := '0';
14 begin
15     process (clk, reset, entrada)
16     begin
17         if reset = '0' then
18             info <= '0';
19         elsif reset = '1' then
20             if (clk' event and clk = '1') then
21                 info <= entrada;
22             end if;
23         end if;
24     end process;
25
26     process (info)
27     begin
28         salida <= info;
29     end process;
30 end architecture;
31
32

```

Código 13: Registro para VF

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity reg_2 is port(
5      entrada: in std_logic_vector(1 downto 0);
6      clk: in std_logic;
7      reset: in std_logic;
8      salida: out std_logic_vector(1 downto 0)
9  );
10 end reg_2;
11
12 architecture arq_reg_2 of reg_2 is
13     signal info: std_logic_vector(1 downto 0) := B"00";
14 begin
15     process (clk, reset, entrada)
16     begin
17         if reset = '0' then
18             info <= "00";
19         elsif reset = '1' then
20             if (clk' event and clk = '1') then
21                 info <= entrada;
22             end if;
23         end if;
24     end process;
25
26     process (info)
27     begin
28         salida <= info;
29     end process;
30 end architecture;
31
32

```

Código 14: Registro para prueba y microinstrucción

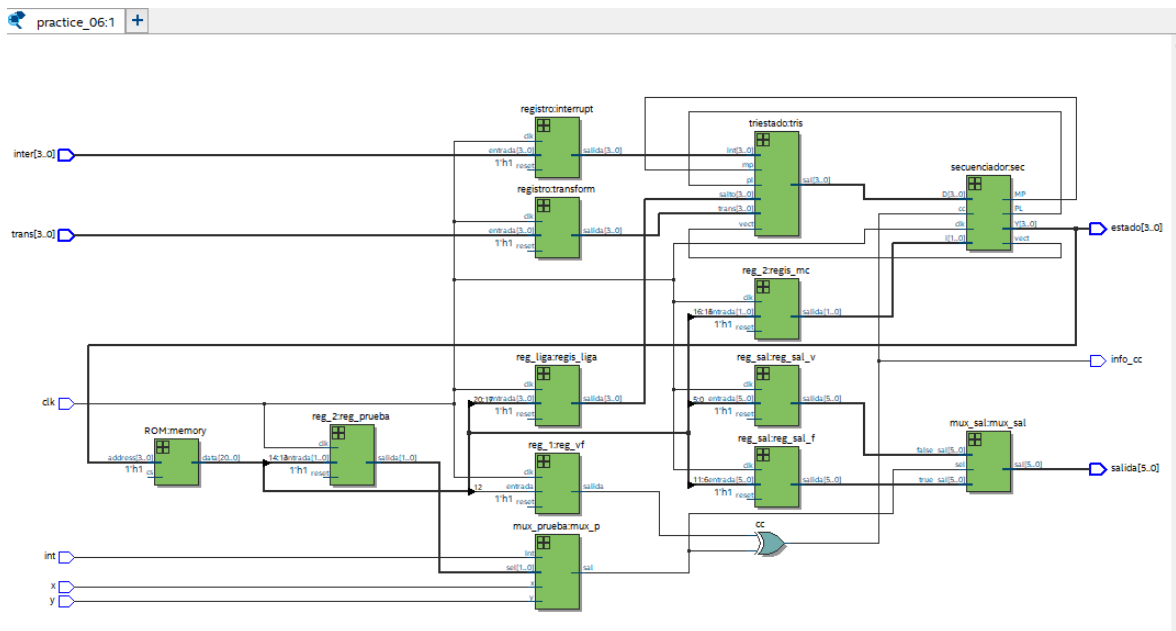
A continuación, se incluirá el archivo general del proyecto donde se conectan los elementos por VHDL y el RTL del proyecto.

```

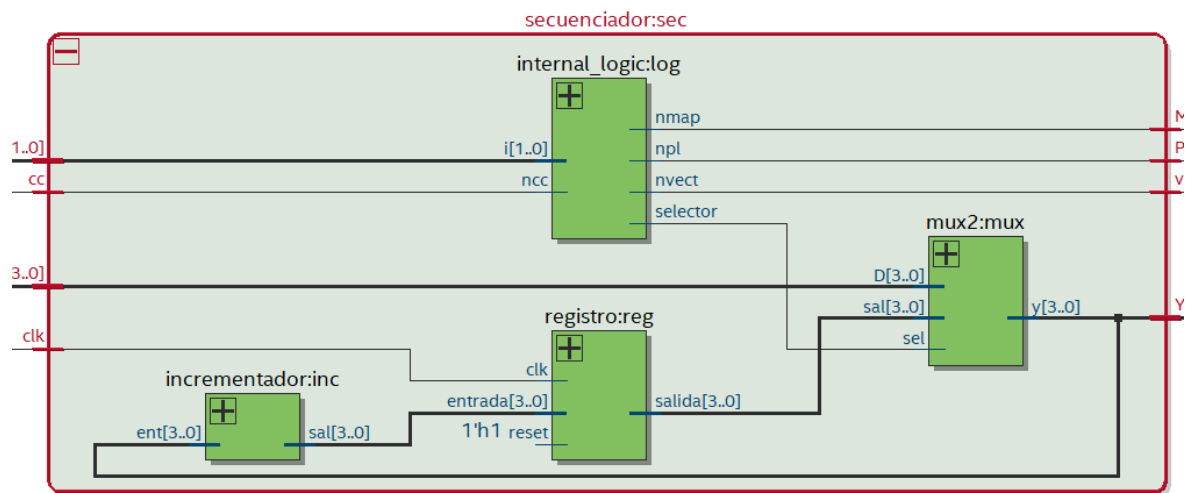
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_unsigned.all;
4
5 entity system is port(
6     x,y,int: in std_logic;
7     clk: in std_logic;
8     trans, inter: in std_logic_vector(3 downto 0);
9     estado: out std_logic_vector(3 downto 0);
10    info_cc: out std_logic;
11);
12 end system;
13
14 architecture arq_system of system is
15     signal info_rom: std_logic_vector(20 downto 0);
16     signal bus_D: std_logic_vector(3 downto 0);
17     signal bus_T_bus_I: std_logic_vector(3 downto 0);
18     signal Y_reg: std_logic_vector(3 downto 0);
19     signal prueba: std_logic_vector(1 downto 0);
20     signal cc: std_logic;
21     signal mp_vect, pl: std_logic;
22     signal salidas_v, salidas_f: std_logic_vector(5 downto 0);
23     signal vf: std_logic;
24     signal ligat: std_logic_vector(3 downto 0);
25
26     begin
27         memory: entity work.ROM(arc_rom) port map (Y_reg,'1',info_rom);
28         reg_liga: entity work.reg_liga(arc_reg_1) port map (info_rom(20 downto 17),clk,'1',liga);
29         reg_mc: entity work.reg_2(arc_reg_2) port map (info_rom(16 downto 15),clk,'1',micro);
30         reg_prueba: entity work.reg_3(arc_reg_3) port map (info_rom(14 downto 13),clk,'1',pru);
31         reg_vf: entity work.reg_4(arc_reg_4) port map (info_rom(12),clk,'1',vf);
32         reg_sal_f: entity work.reg_sal(arc_reg_5) port map (info_rom(11 downto 6),clk,'1',salidas_f);
33         reg_sal_v: entity work.reg_sal(arc_reg_5) port map (info_rom(5 downto 0),clk,'1',salidas_v);
34
35         cc<= mux_ent xor vf;
36         sec: entity work.secuenciador(arc_sec) port map (clk,cc,micro,bus_D,pl,mp_vect,Y_reg);
37         info_cc<=cc;
38         mux_sal: entity work.mux_sal(arc_mux_p) port map (x,y,int,pru,mux_ent);
39         interrupt: entity work.registro(arc_reg) port map (inter,clk,'1',bus_I);
40         transform: entity work.registro(arc_reg) port map (trans,clk,'1',bus_T);
41         tris: entity work.triestado(arc_tristate) port map (pl,mp_vect,liga,bus_T,bus_I,bus_D);
42         estado<=Y_reg;
43     end architecture;
44
45
46
47
48
49
50
51

```

Código 15: Código general del proyecto



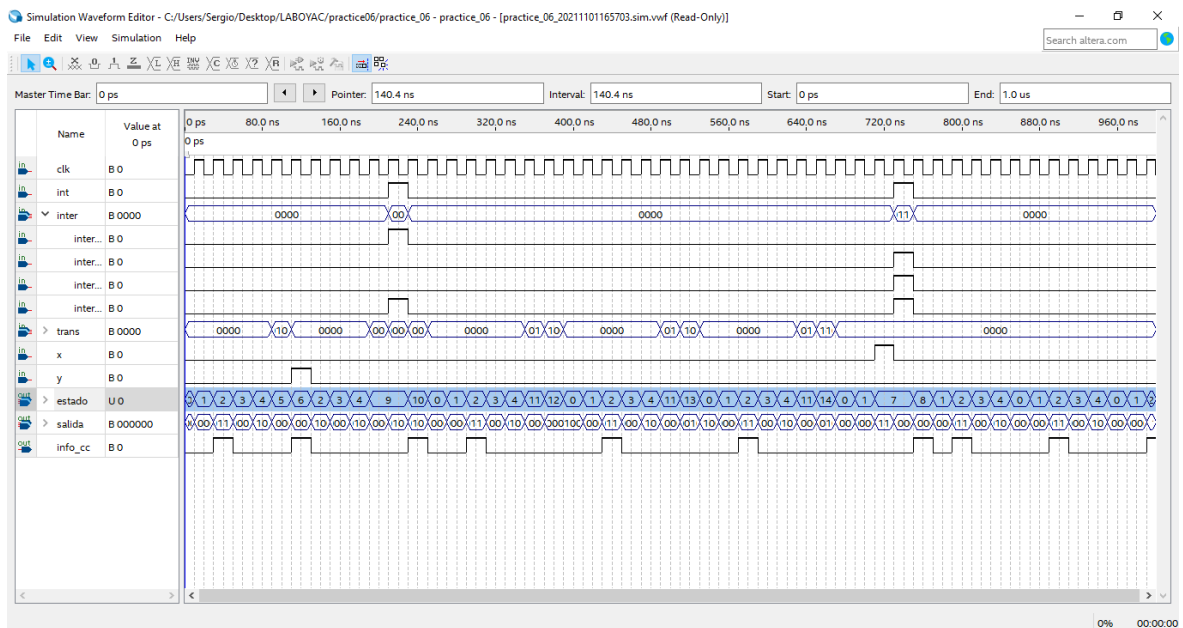
RTL 1: Proyecto general



RTL 2: Estructura del secuenciador

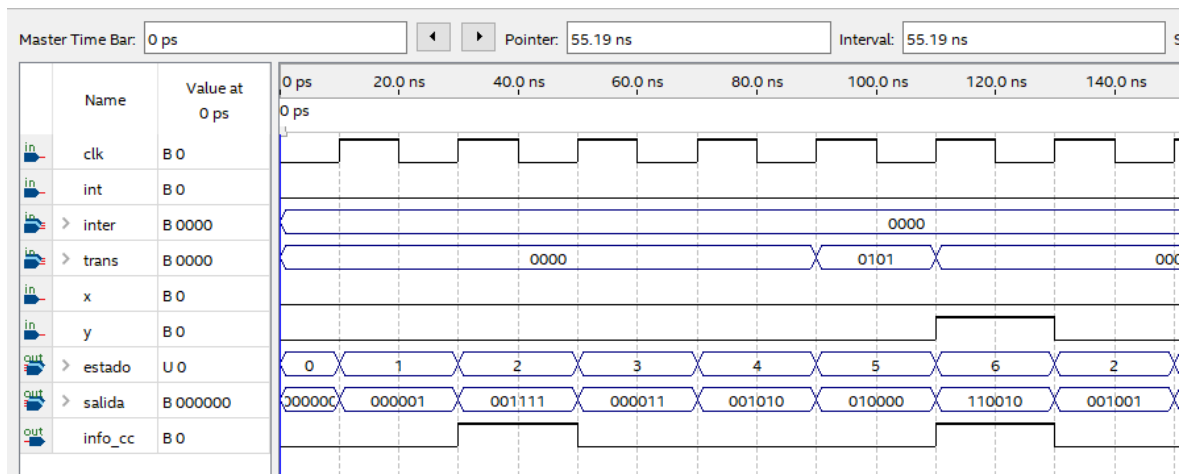
Simulación

Para la simulación se realizó con Waveform Editor que nos permitía la entrada de elementos en grupos. En los puntos de la carta ASM donde se realice una transformación o una interrupción se asigna el estado al que se quiere acceder como entrada. En la vista se observan como entradas la frecuencia del reloj, $X, Y, Interrupción$ y las entradas de los registros de interrupción y transformación. Como salidas se muestra el estado al que se dirige, la salida del estado presente y la información del CC.



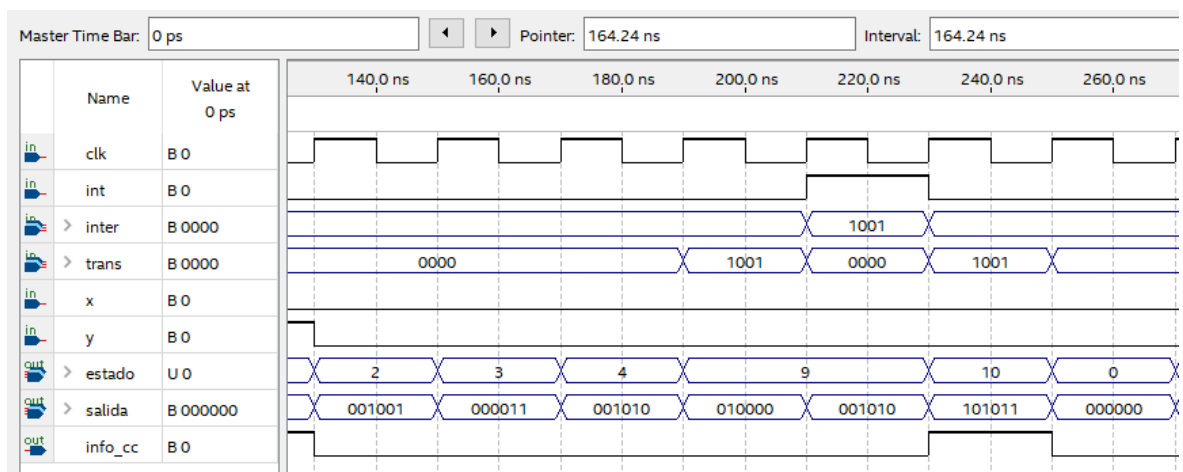
Simulación 1: Simulación general del proyecto

En la primera sección se realiza el recorrido del estado 0 al estado 5, utilizando el registro de transformación.



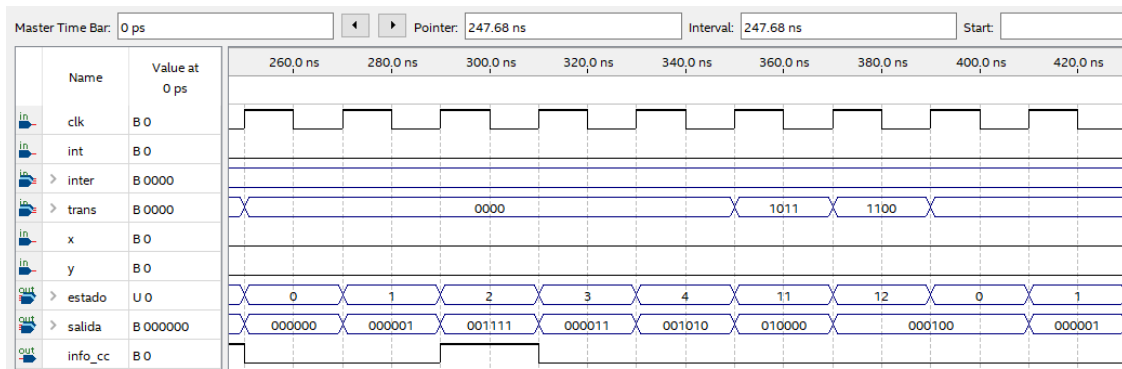
Simulación 2: Recorrido del estado 2 al 6

El siguiente recorrido se hizo del estado 2 al estado 10. Se utilizó el registro de transformación dirigido al estado 9 y el registro de interrupción activado para dirigirse de nuevo al estado 9.

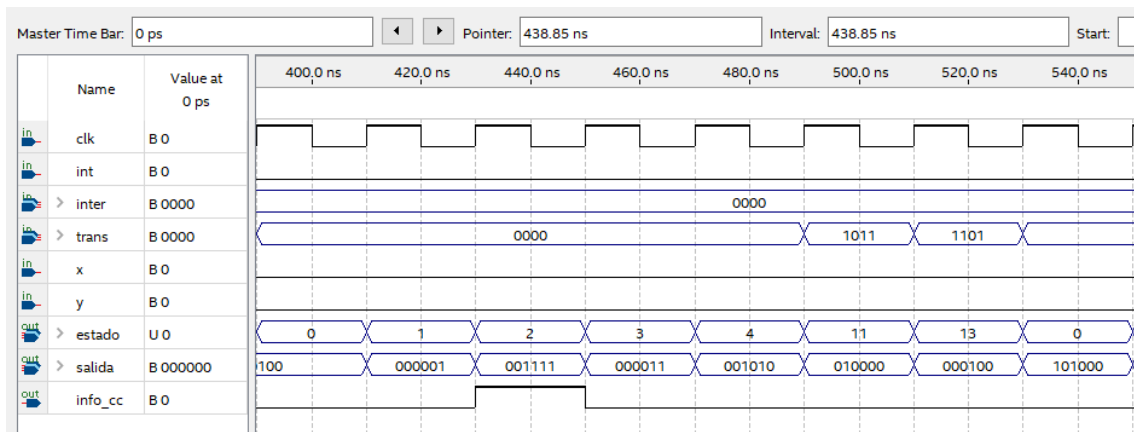


Simulación 3: Recorrido del estado 0 al 10, pasando por el estado 9

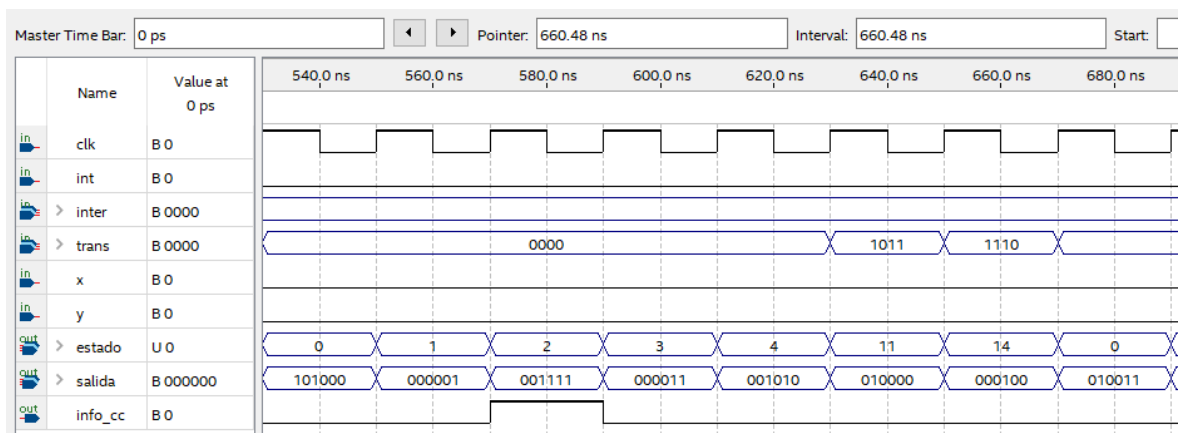
El siguiente recorrido se hizo del estado 0 al 11. Una vez llegado el 11 por registro de transformación se realizaron 3 transformaciones diferentes, una al estado 12, otra al 14 y la última al 14.



Simulación 4: Recorrido del estado 0 al 12, pasando por el estado 11

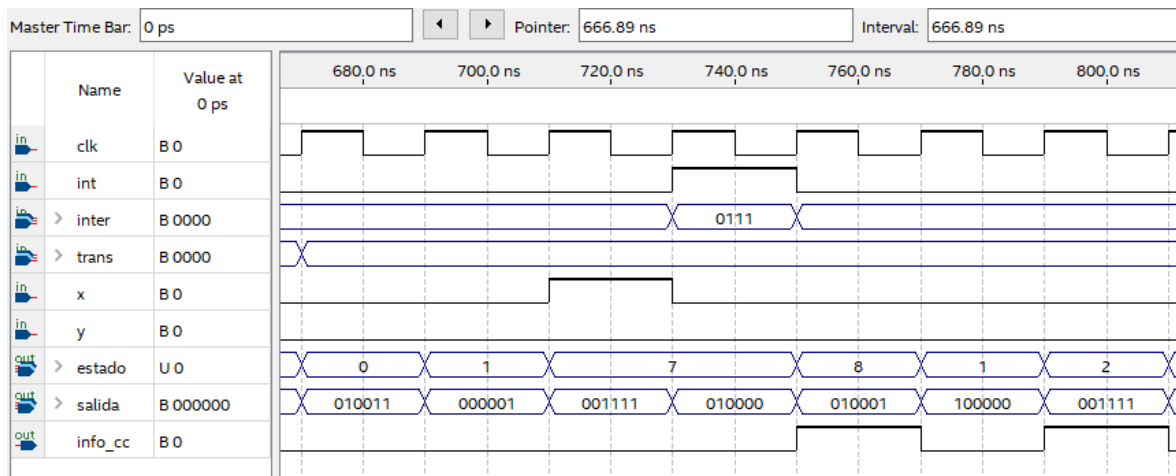


Simulación 5: Recorrido del estado 0 al 13, pasando por el estado 11



Simulación 6: Recorrido del estado 0 al 14, pasando por el estado 11

El último recorrido de la carta ASM fue el del estado 7, tomando la entrada X con 1 y activando la interrupción en el estado 7.



Simulación 7: Recorrido del estado 0 al 8, tomando decisión con X e interrupción en el estado 7

Conclusiones

Murrieta Villegas Alfonso

A lo largo de las prácticas previas hemos aprendido distintos conceptos relacionados con las máquinas de estado además de poder trabajar mediante VHDL este tipo de abstracción con el fin de llegar a una entidad mayor denominada procesador.

En la presente práctica es como a través de la creación de componentes de un secuenciador es como aprendimos el funcionamiento de la asignación de memoria que es un componente elemental de cualquier procesador, además, mediante la tabla de verdad determinamos 4 distintas instrucciones para determinar el uso de las entradas al mismo secuenciador, empleando a la par el uso de registros o direcciones de memoria.

Reza Chavarria Sergio Gabriel

A partir de la creación y configuración de la lógica y de los componentes que tiene un secuenciador se pudo entender el funcionamiento y comportamiento de este a partir de una asignación en memoria.

Con la obtención de los datos de la tabla de verdad se necesita tomar en cuenta las 4 diferentes instrucciones, el manejo de las entradas y la lógica interna del secuenciador para realizar el funcionamiento adecuado. Las diferentes instrucciones necesitan manejarse de manera cuidadosa en los registros utilizados.

Valdespino Mendieta Joaquin

En la presente práctica pudimos observar y comprender un poco más el funcionamiento del secuenciador básico de 4 microinstrucciones, basado en el direccionamiento implícito para su armado, para ello se tuvieron que tener en cuenta la creación de los módulos correspondientes a la arquitectura presentada, donde la parte esencial es la memoria y la estructura del secuenciador con sus respectivos buses, todo para poder implementar de manera correcta la transición de las cartas asm, que se presenten, además

la modularidad del proyecto permite la modificación y moldeo acorde a los dicho anteriormente.

Bibliografía

- Savage J., Vázquez G & Chávez N. (2015). Diseño de Microprocesadores, Capítulo III Construcción De Máquinas de Estado usando Memorias. Encontrado el 29 de octubre del 2021. Sitio Web: https://biorobotics.fi-p.unam.mx/wp-content/uploads/Courses/arquitectura_de_computadoras/material_de_apoyo/diseño_de_procesadores.pdf