Universidad Nacional Autónoma de México
Facultad de Ingeniería-Ingeniería en Computación

Práctica No. 6
Secuenciador básico

Profesora: Ayesha Román García

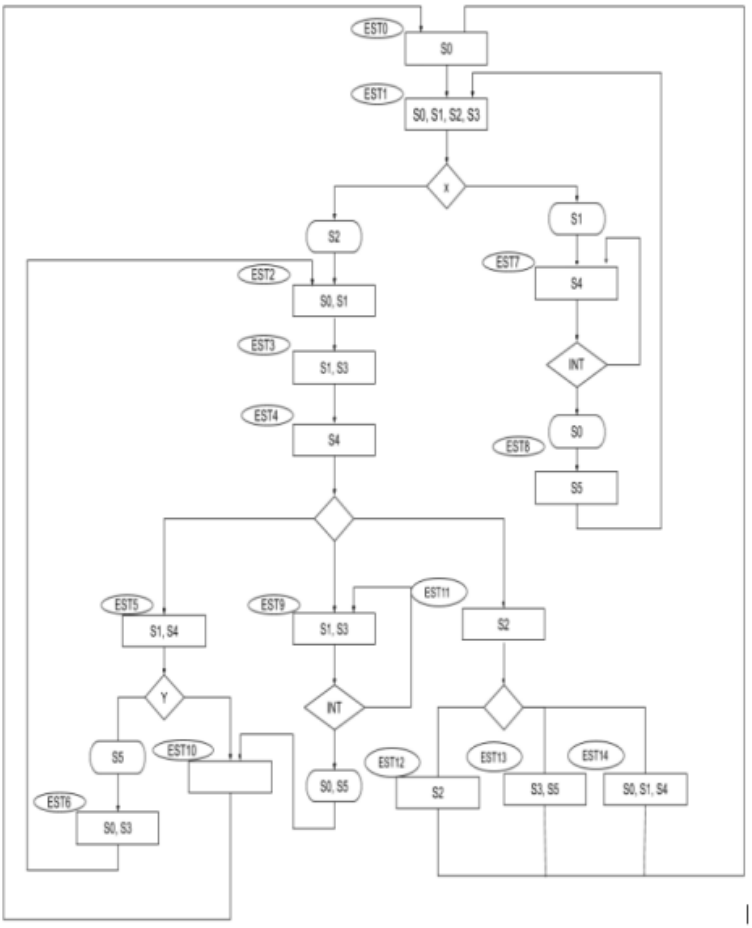
Lab. Organización y arquitectura de Computadoras

Alumnos: Cárdenas Avila Josué David
Huerta Rodriguez Ivan Ariel
Valdelamar Tamez Valeria

14/11/2022

Introducción:
En esta práctica diseñaremos un secuenciador básico, una de las estructuras más avanzadas para lo que llevamos del curso, un secuenciador controla todas las señales tanto internas como externas es el cerebro del microprocesador. Recibe la información en forma de combinaciones de bits que decodifica y ejecuta dentro de la unidad de control. En este ejercicio se ensambla uno para posteriormente poner a prueba su funcionamiento.
Objetivo
Familiarizar al alumno en el conocimiento del secuenciador básico, el cual es una parte fundamental del procesador.

Desarrollo:



Basándonos en la carta ASM lo que haremos será determinar el tamaño de la memoria:
Estados: 15 Entradas: 4 Salidas: 6
N=4
M=2+1+2+4+6+6
2^4*20=320
Prueba
X=00
Y=01
Int=10
Qaux=11
MI
00 Paso contiguo

01 Salto condicional
10 Salto de transición
11 Salto de Interrupción

Estados

Est0=00, Est1=01,Est02=10…
Est14=1110

Tabla Verdad

Dirección				Contenido																					
Edo. Presente				Prueba		VF	MI		Liga				SF						SV						
P3	P2	P1	P0	K2	K1				P3	P2	P1	P0	s5	s4	S3	S2	S1	S0	S5	S4	S3	S2	S1	S0
0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
0	0	0	1	0	0	1	0	1	0	1	1	1	0	0	1	1	1	1	0	0	1	1	1	1
0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1
0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0
0	1	0	0	1	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	1	1	0	1	1	0	1	0	1	1	0	0	1	0	0	1	0	0	1	0
0	1	1	0	1	1	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	1	0	0	1
0	1	1	1	1	0	1	1	1	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0
1	0	0	0	1	1	0	0	1	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0
1	0	0	1	1	0	0	1	1	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	1
1	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
1	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
1	1	0	1	1	1	0	0	1	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0
1	1	1	0	1	1	0	0	1	0	0	0	0	0	1	0	0	1	1	0	1	0	0	1	1

Similar a prácticas anteriores, lo primero que haremos será el registro, aquí recibiremos el valor del reloj y de reset, si el valor de reset se encuentra en cero no podremos avanzar al siguiente estado, pero una vez que el reset se encuentre en uno podremos avanzar al estado N dependiendo del valor que tengamos en las entradas.

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

entity registroIns is
    Port ( clk : in std_logic;
           reset : in std_logic;
           data_in : in std_logic_vector (1 downto 0);
           data_out : out std_logic_vector (1 downto 0);
           ena:in std_logic);

end registroIns;

architecture Behavioral of registroIns is

    signal internal_value : std_logic_vector (1 downto 0) := B"00";
    constant alta_impedancia: std_logic_vector (1 downto 0) := "ZZ";
    constant zero: std_logic_vector (1 downto 0) := B"00";

    begin
    process (clk, reset, data_in)
    begin
       if reset = '0' then
          internal_value <= zero;
       elsif rising_edge (clk) then
          internal_value <= data_in;
       end if;
    end process;

    process (internal_value,ena)
    begin
       if ena ='1' then
          data_out <= alta_impedancia;
       else
          data_out <= internal_value;
       end if ;
    end process;
end Behavioral;
```

Posteriormente haremos una estructura llamada incrementador, la cual nos ayudará a pasar del estado n al estado n más uno, esto nos permitirá hacer pasos continuos.

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

entity incrementador  is
    Port (Estado : in std_logic_vector (3 downto 0);
          Sal: out std_logic_vector (3 downto 0));

end incrementador;

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

Mediante el contador vamos a seleccionar si se hará un paso contiguo para incrementar n o en caso de ser un salto, seleccionar el valor de la liga a la que hará el salto,

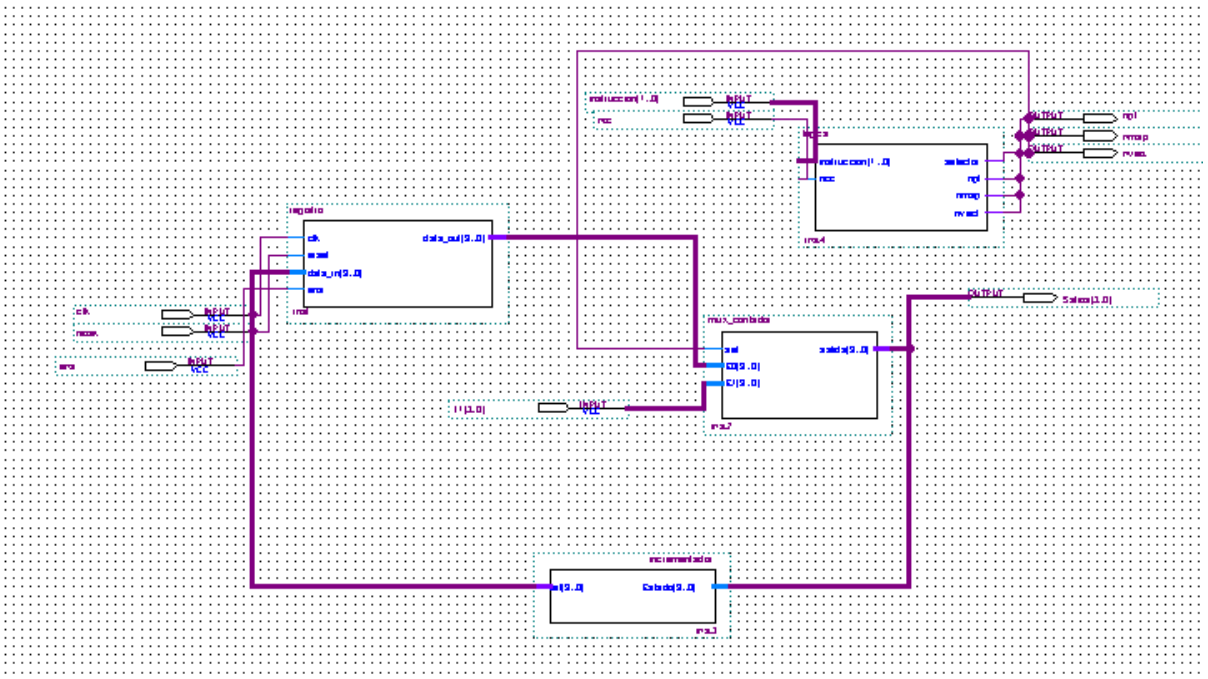
```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux_contador is
    Port ( sel : in std_logic;--seleccionar si es n+1 o p
           E0 : in std_logic_vector (3 downto 0);---N+1 O INCREMENTADOR
           E1: in  std_logic_vector (3 downto 0);---liga
           salida : out std_logic_vector (3 downto 0));--salida mux

end mux_contador;

architecture Behavioral of mux_contador 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;
```

Como última parte de este segmento tenemos la lógica, esta entidad nos permite determinar que va a activar el multiplexor y en consecuencia a esto mostrar la salida.

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

entity logica is
port (instruccion: in std_logic_vector (1 downto 0);
         ncc: in std_logic;
         selector: out std_logic;
         npl:  out std_logic;
         nmap:  out std_logic;
         nvect: out std_logic);
end logica;

architecture Behavioral of logica is
begin
   process (instruccion, ncc)
      begin
         if instruccion ="00" then
               selector <= '0';
               npl <= '1';
               nmap <= '1';
               nvect <= '1';
         elsif instruccion ="01" then
               if ncc='0' then
               selector <= '1';
               else selector <='0';
                end if ;

               npl <= '0';
               nmap <= '1';
               nvect <= '1';
         elsif instruccion ="10" then
               selector <= '1';
               npl <= '1';
               nmap <= '0';
               nvect <= '1';
         elsif instruccion ="11" then
               if ncc='0' then
               selector <= '1';
               else selector <='0';
                end if ;
```

```
            npl <= '1';
            nmap <= '1';
            nvect <= '0';
        end if ;
    end process;
end behavioral;
```

Todo lo anterior representa un segmento de nuestro secuenciador final y similar al de la práctica anterior tendría la siguiente estructura:



Una vez teniendo esta parte del programa lo que haremos será programar la memoria, divisor, mux entrada, mux salida y registros faltantes.

Comenzaremos por lo tanto con la memoria, esta entidad recibe el valor obtenido previamente en decimal y a partir de él determina el valor de la liga.

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

entity memoria is
port( dir: in std_logic_vector(3 downto 0);
      data: out std_logic_vector(20 downto 0));
end memoria;

architecture behavior of memoria is
   type mem is array (0 to 14) of std_logic_vector(20 downto 0);
   signal s: mem;

   begin
            -- p    vf  mi    L     sf       sv
      s(0) <= "11"&"0"&"00"&"0000"&"000001"&"000001";
      s(1) <= "00"&"1"&"01"&"0111"&"001111"&"001111";
      s(2) <= "11"&"0"&"00"&"0000"&"000011"&"000011";
      s(3) <= "11"&"0"&"00"&"0000"&"001010"&"001010";
      s(4) <= "11"&"0"&"10"&"0000"&"010000"&"010000";
      s(5) <= "01"&"1"&"01"&"1010"&"110010"&"010010";
      s(6) <= "11"&"0"&"01"&"0010"&"001001"&"001001";
      s(7) <= "10"&"1"&"11"&"0000"&"010001"&"010000";
      s(8) <= "11"&"0"&"01"&"0001"&"100000"&"100000";
      s(9) <= "10"&"0"&"11"&"0000"&"001010"&"101011";
      s(10) <= "11"&"0"&"01"&"0000"&"000000"&"000000";
      s(11) <= "11"&"0"&"10"&"0000"&"000100"&"000100";
      s(12) <= "11"&"0"&"01"&"0000"&"000100"&"000100";
      s(13) <= "11"&"0"&"01"&"0000"&"101000"&"101000";
      s(14) <= "11"&"0"&"01"&"0000"&"010011"&"010011";


   process(dir)
      begin
         data <= s(conv_integer(unsigned(dir)));
   end process;
end behavior;
```

Ahora creamos el divisor, esta entidad cumple la función de separar los bits de la salida de los bits de la liga, esto con la finalidad de poder diferenciarlos con mayor facilidad al simular.

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

entity divisor is
   port( entrada: in std_logic_vector(20 downto 0);
         prueba: out std_logic_vector(1 downto 0);--prueba
         liga: out std_logic_vector(3 downto 0);--liga
         vf: out std_logic;  -------------------- vf
         instruccion: out std_logic_vector(1 downto 0);--MI
         sf: out std_logic_vector(5 downto 0);---salidas falsas
         sv: out std_logic_vector(5 downto 0));---salidad verdaderas
   end divisor;

architecture behavior of divisor is
   begin

      process(entrada)
      begin
         sv <= entrada(5 downto 0);
         sf <= entrada( 11 downto 6 );
         liga <= entrada(15 downto 12);
         instruccion <= entrada(17 downto 16);
         vf <= entrada(18);
         prueba <= entrada(20 downto 19);
      end process;
end behavior;
```

Mediante una entidad llamada multiplexor de prueba vamos a seleccionar si nuestras pruebas fueron activadas o no, si se da el caso de que se activen vamos a

mandar el valor a la siguiente instancia como un uno o un cero y dependiendo de ese valor veremos qué entradas son falsas y cuáles verdaderas.

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

entity mux_entrada is
Port( p: in std_logic_vector(1 downto 0);
      x: in std_logic;
      y: in std_logic;
      int: in std_logic;
      qaux:in std_logic;
      VoF: out std_logic);
end mux_entrada;

architecture behavior of mux_entrada is
begin
   process(p,x,y,int)
   begin
      if p = "00" then --x
         VoF <= x;
      elsif p = "01" then --y
         VoF <= y;
      elsif p = "10" then --int
         VoF <= int;
      elsif p = "11" then --
         VoF <= qaux;
      end if;
   end process;
end behavior;
```

Lo siguiente que tendríamos es un multiplexor para las salidas falsas y verdaderas, si obtenemos un valor cero de la prueba veremos las salidas falsas en la simulación pero si el valor es uno veremos las salidas verdaderas.

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

entity mux_sal is
Port( seleccion: in std_logic;
      sf: in std_logic_vector(5 downto 0);
      sv: in std_logic_vector(5 downto 0);
      salida: out std_logic_vector(5 downto 0));
end mux_sal;

architecture behavior of mux_sal is
begin
   process(seleccion,sf,sv)
   begin
      if seleccion = '0' then
         salida <= sv;
      elsif seleccion = '1' then
         salida <= sf;
      end if;
   end process;
end behavior;
```

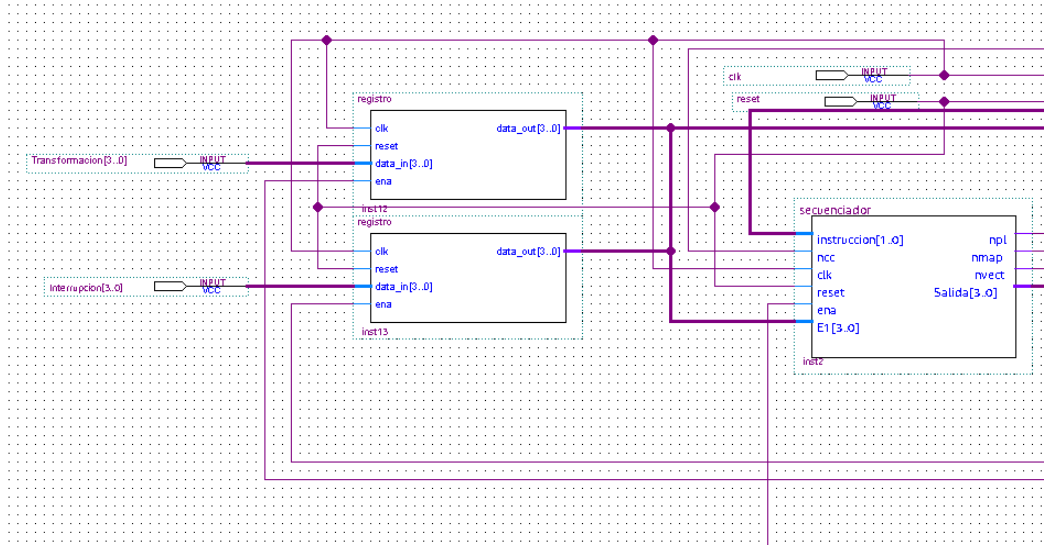
Finalmente tenemos unas entidades que sirven para los registros de entrada, vf y salidas.

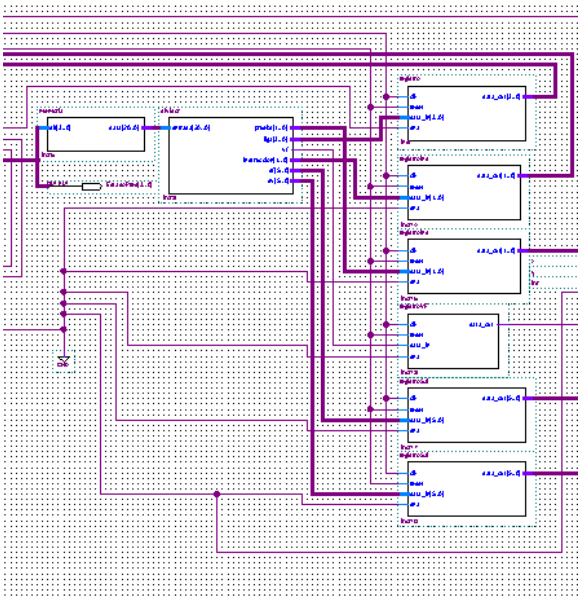
```vhdl
library IEEE;
Library ieee;
use IEEE.STD_LOGIC_1164.ALL;

entity registroIns is
    Port ( clk : in std_logic;
           reset : in std_logic;
           data_in : in std_logic_vector (1 downto 0);
           data_out : out std_logic_vector (1 downto 0);
           ena:in std_logic);

end registroIns;

architecture Behavioral of registroIns is

    signal internal_value : std_logic_vector (1 downto 0) := B"00";
    constant alta_impedancia: std_logic_vector (1 downto 0) := "ZZ";
    constant zero: std_logic_vector (1 downto 0) := B"00";

    begin
    process (clk, reset, data_in)
    begin
       if reset = '0' then
          internal_value <= zero;
       elsif rising_edge (clk) then
          internal_value <= data_in;
       end if;
    end process;

    process (internal_value,ena)
    begin
       if ena ='1' then
          data_out <= alta_impedancia;
       else
          data_out <= internal_value;
       end if ;
    end process;
end Behavioral;
```

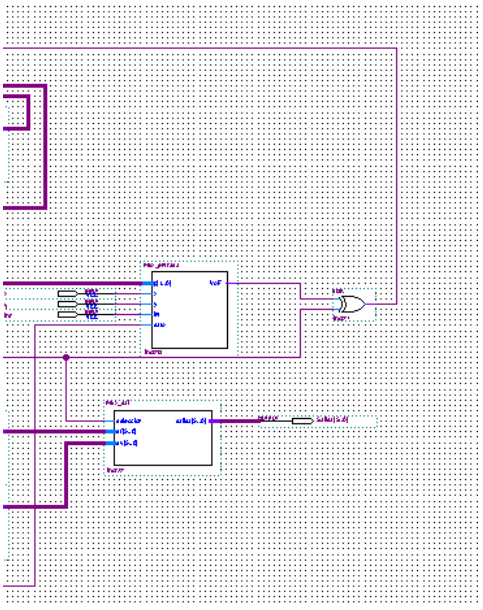
```vhdl
library IEEE;
Library ieee;
use IEEE.STD_LOGIC_1164.ALL;

entity registroVF is
    Port ( clk : in std_logic;
           reset : in std_logic;
           data_in : in std_logic;
           data_out : out std_logic;
           ena:in std_logic);

end registroVF;

architecture Behavioral of registroVF is

    signal internal_value : std_logic := '0';
    constant alta_impedancia: std_logic := 'Z';
    constant zero: std_logic := '0';

    begin
    process (clk, reset, data_in)
    begin
       if reset = '0' then
          internal_value <= zero;
       elsif rising_edge (clk) then
          internal_value <= data_in;
       end if;
    end process;

    process (internal_value,ena)
    begin
       if ena ='1' then
          data_out <= alta_impedancia;
       else
          data_out <= internal_value;
       end if ;
    end process;
end Behavioral;
```

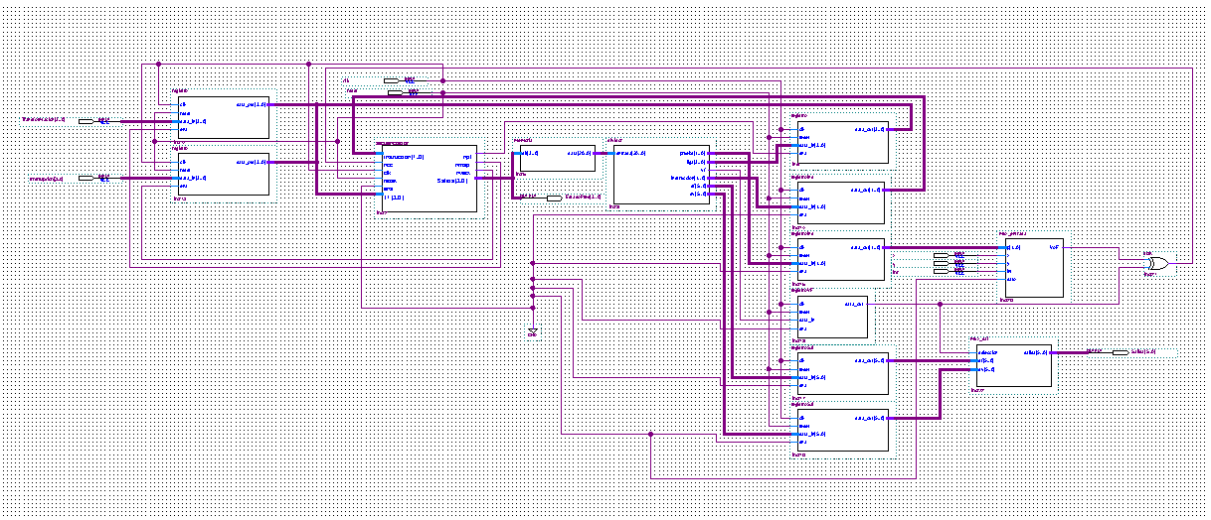
```vhdl
library IEEE;
Library ieee;
use IEEE.STD_LOGIC_1164.ALL;

entity registroSal is
    Port ( clk : in std_logic;
           reset : in std_logic;
           data_in : in std_logic_vector (5 downto 0);
           data_out : out std_logic_vector (5 downto 0);
           ena:in std_logic);

end registroSal;

architecture Behavioral of registroSal is

    signal internal_value : std_logic_vector (5 downto 0) := B"000000";
    constant alta_impedancia: std_logic_vector (5 downto 0) := "ZZZZZZ";
    constant zero: std_logic_vector (5 downto 0) := B"000000";

    begin
    process (clk, reset, data_in)
    begin
       if reset = '0' then
          internal_value <= zero;
       elsif rising_edge (clk) then
          internal_value <= data_in;
       end if;
    end process;

    process (internal_value,ena)
    begin
       if ena ='1' then
          data_out <= alta_impedancia;
       else
          data_out <= internal_value;
       end if ;
    end process;
end Behavioral;
```

Finalmente tendríamos que unir las instancias resultando en lo siguiente:

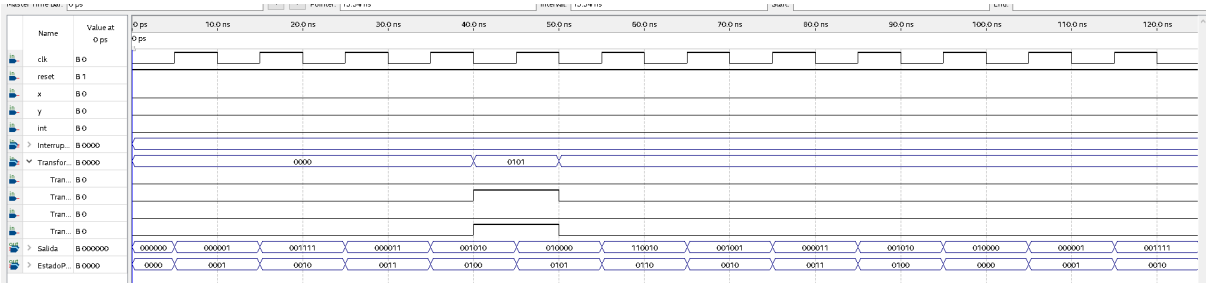








Finalmente al simular obtendremos los siguientes resultados:



Conclusiones:

Cárdenas Avila Josué David:

Mediante esta práctica pudimos estudiar el comportamiento de los secuenciadores básicos y su estructura con el fin de analizar cada una de ellas para poderlas comprender y posteriormente plasmar en la simulación de nuestro ejercicio práctico, es por ello que los objetivos han logrado ser concluidos de forma exitosa.

Huerta Rodríguez Iván Ariel:

Se entiende cómo funcionan los secuenciadores básicos a través de la simulación, además de la importancia de su uso y se logró culminar el entendimiento de lo que se vio en teoría.

Valdelamar Tamez Valeria:

Al realizar esta práctica pudimos comprender lo que es el comportamiento de los secuenciadores básicos, con ello se realizó un ejercicio donde pudimos poner a prueba los conocimientos adquiridos.

Referencias:

http://dea.unsj.edu.ar/mp1/APUNTES/Catedra%20Modos%20de%20Direccionamiento.pdf

https://www.infor.uva.es/~bastida/OC/modos.pdf

https://hopelchen.tecnm.mx/principal/sylabus/fpdb/recursos/r100112.PDF