	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio

Profesor: ING. JULIO CESAR CRUZ ESTRADA

Asignatura: LABORATORIO DE ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORAS

Grupo: 07

No de Práctica: 4

Integrante(s): Jiménez Treviño Emilio Cristóbal

Martinez Perez Brian Erik

Robles Jiménez Marco Antonio

No. de Equipo de cómputo empleado: *s/n*

Semestre: *2026-1*

Fecha de entrega: *26/09/2025*

Observaciones:

CALIFICACIÓN: _____

Práctica 4. Construcción de Máquinas de estados Usando Memorias Direccionamiento Entrada - Estado

Objetivo

Familiarizar al alumno en el conocimiento de construcción de máquinas de estados usando direccionamiento de memorias con el método de direccionamiento entrada-estado.

Introducción

En esta práctica se abordará la construcción de máquinas de estados usando memorias Direccionamiento Entrada - Estado. Este enfoque permite organizar el diseño en etapas bien definidas: la elaboración de la carta ASM, la obtención de la tabla de verdad y la codificación de los distintos bloques en VHDL, como el registro, concatenador, memoria ROM, divisor de datos y divisor de frecuencia. Se detallarán los pasos para la creación de un proyecto en Quartus y la asignación de pines para su posterior simulación y carga en un dispositivo FPGA que en nuestro caso fue la DE10-LITE debido a su capacidad y accesibilidad.

Desarrollo

Para comenzar primero se propuso una carta ASM con 5 estados, 3 entradas y 4 salidas. Las restricciones eran, al menos dos salidas en lógica negada, al menos un estado con salidas condicionales.

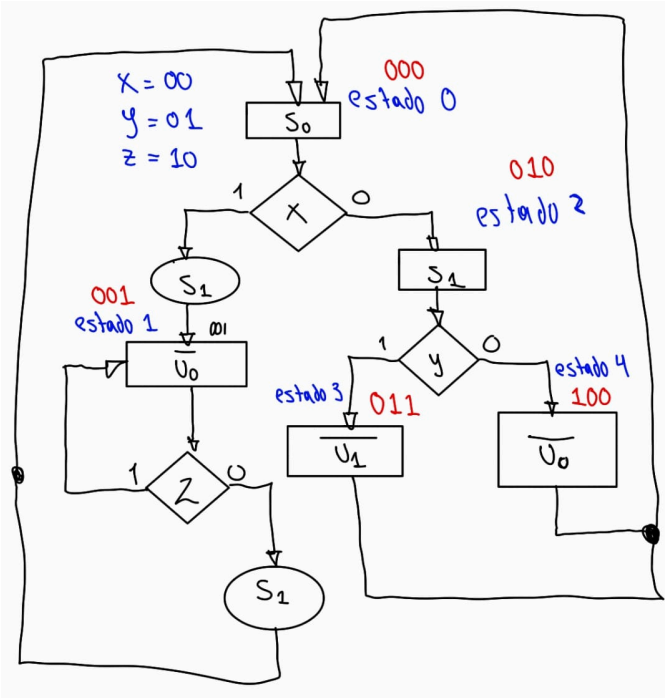


Figura 1. Carta ASM propuesta.

Después de obtener la carta ASM, obtenemos la tabla de verdad, donde debemos de tomar en cuenta que utilizamos el método de direccionamiento entrada-estado. Debemos tener en la cabecera el estado presente, prueba, liga falsa, liga verdadera, salida falsa y salida verdadera.

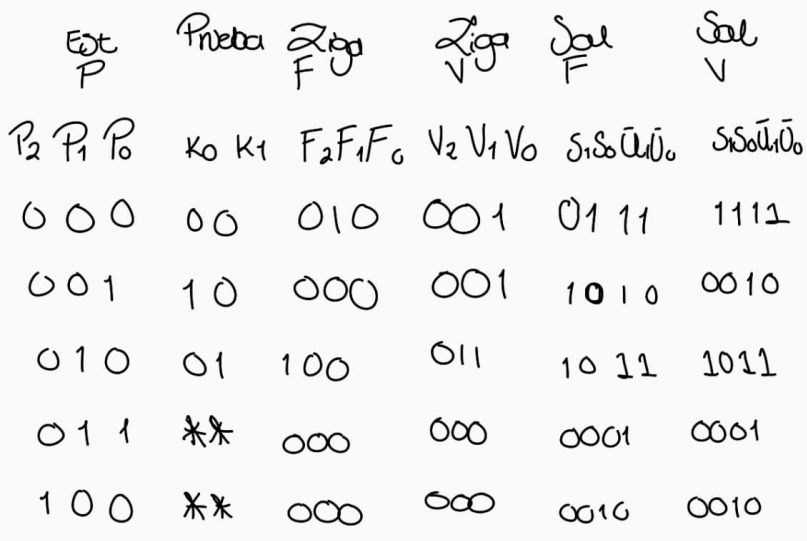


Figura 2. Tabla de verdad, obtenida por el método de direccionamiento entrada-estado.

Cuando ya tenemos la tabla de verdad, podemos implementar el direccionamiento entrada-estado utilizando el software de desarrollo Quartus y escribir el contenido de memoria obtenido.

Lo primero que debemos hacer es crear un proyecto en Quartus, seleccionamos la pestaña FILE -> New Project Wizard. En el asistente de creación de nuevos proyectos, en la Fig. 3 podemos observar seleccionado la familia del dispositivo en nuestro caso la MAX 10 y dentro de esta vendría a hacer la 10M50DAF484C7G dentro de esta familia.

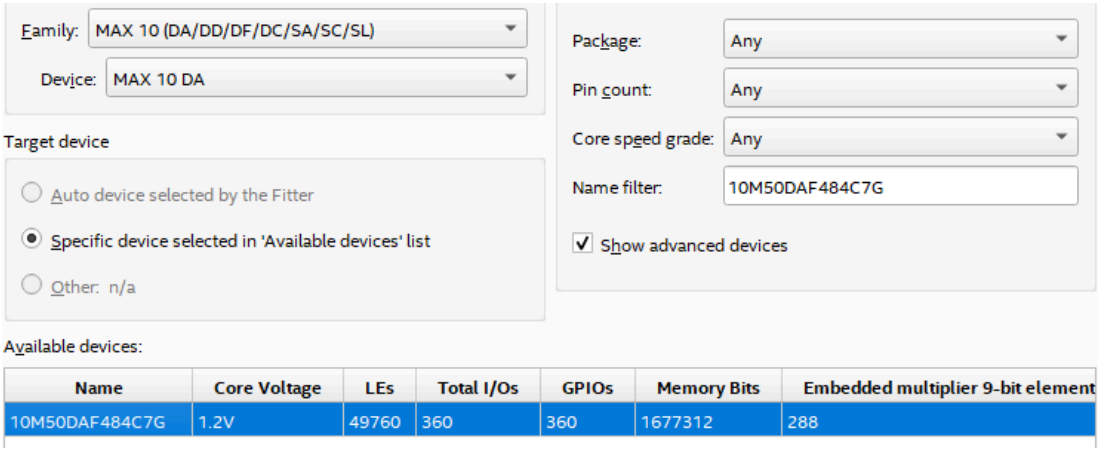


Figura 3. Device

Una vez creado nuestro proyecto, creamos un archivo de tipo .bdf dando click en la opción de Block Diagram/Schematic File como se muestra en la Fig. 4.

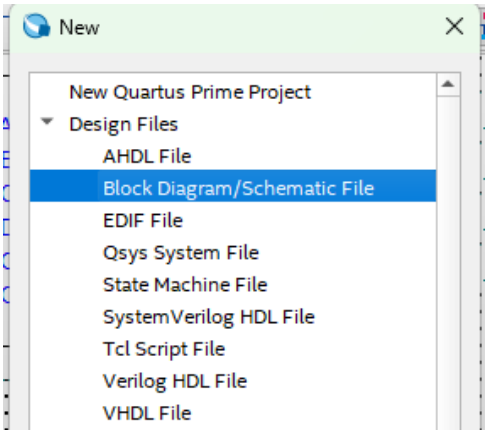


Figura 4. New File

Para implementar el circuito de direccionamiento entrada-estado, debemos tener en cuenta que tenemos un bloque llamado “registro”, el cual tiene como entradas el estado siguiente, el reloj y reset. El bloque de memoria almacena, la prueba, liga verdadera, liga falsa. salidas verdaderas y salidas falsas. Por último tenemos 3 multiplexores el primero el cual recibe a prueba obtiene a la salida “Qsel”. El segundo multiplexor el cual determina el estado siguiente dependiendo de la entrada de Qsel. El último multiplexor debe determinar cuál salida mostrar dependiendo la “Qsel” que tenga en la entrada.

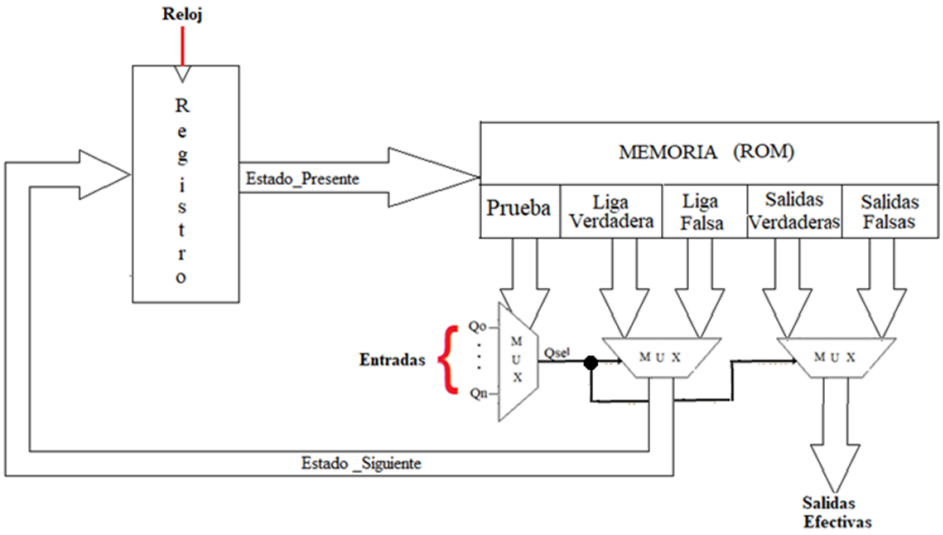


Figura 5. Diagrama circuito direccionamiento entrada-estado.

El primer código generado fue “registró”, su función es almacenar un valor de 3 bits (edos) en cada flanco de subida del reloj, a menos que se active una señal de reinicio, que lo inicializa a "000".

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

entity registro is
    Port ( reloj : in std_logic;
           reset : in std_logic;
           edos : in std_logic_vector (2 downto 0);
           edop : out std_logic_vector (2 downto 0)
    );
end registro;


architecture Behavioral of registro is

signal internal_value: std_logic_vector (2 downto 0) := B"000";
begin
    process(reloj, reset, edos)
    begin

        if reset = '0' then
            internal_value <= B"000";
        elsif rising_edge(reloj) then
            internal_value <= edos;
        end if;
    end process;

    process(internal_value)
    begin
        edop <= internal_value;
    end process;


end Behavioral;

```

Figura 6. Código VHDL, para generar el bloque “registro”.

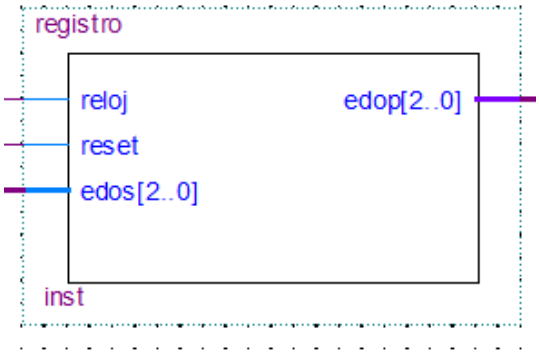


Figura 7. Bloque “registro”.

El segundo código fue “rom”, Almacena un conjunto de valores predefinidos. Cuando se le da una dirección de 3 bits que representa los estados de la carta ASM, devuelve el valor de 16 bits almacenado en esa posición de memoria, donde se encuentra la prueba, la liga verdadera, liga falsa, salidas verdaderas y salidas falsas.

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

entity rom is
    Port ( direccion : in std_logic_vector (2 downto 0);
           data : out std_logic_vector (15 downto 0)
    );
end rom;

architecture Behavioral of rom is

type mem is array (0 to 7) of std_logic_vector (15 downto 0);
signal internal_mem: mem;

begin
    -- prueba & LF & LV & SF & SV           s1s0u1u0
    internal_mem(0) <= "00"& "010" & "001" & "0111" & "1111";
    internal_mem(1) <= "10"& "000" & "001" & "1010" & "0010";
    internal_mem(2) <= "01"& "100" & "011" & "1011" & "1011";
    internal_mem(3) <= "--"& "000" & "000" & "0001" & "0001";
    internal_mem(4) <= "--"& "000" & "000" & "0010" & "0010";
    internal_mem(5) <= "--"& "000" & "000" & "0011" & "0011";
    internal_mem(6) <= "--"& "000" & "000" & "0011" & "0011";
    internal_mem(7) <= "--"& "000" & "000" & "0011" & "0011";
     process(direccion)
     begin
       data <= internal_mem(conv_integer(unsigned(direccion)));

    end process;

end Behavioral;
```

Figura 8. Código VHDL, para generar el bloque “rom”.

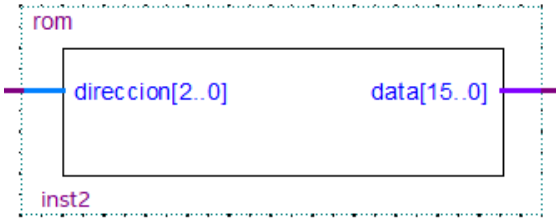


Figura 9. Bloque “rom”.

El tercer código fue “div_datos”, es un divisor de bits que toma un vector de 16 bits (data) y lo divide en cinco vectores de salida más pequeños: prueba (2 bits), LF (3 bits), LV (3 bits), SF (4 bits) y SV (4 bits).

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

entity div_datos is
    Port ( data : in std_logic_vector (15 downto 0);
           prueba : out std_logic_vector (1 downto 0);
           LF : out std_logic_vector (2 downto 0);
           LV : out std_logic_vector (2 downto 0);
           SF : out std_logic_vector (3 downto 0);
           SV : out std_logic_vector (3 downto 0)
    );
end div_datos;


architecture Behavioral of div_datos is
begin

    process(data)
    begin

          prueba <= data(15 downto 14);
          LF <= data(13 downto 11);
          LV <= data(10 downto 8);
          SF <= data(7 downto 4);
          SV <= data(3 downto 0);

    end process;

end Behavioral;
```

Figura 10. Código VHDL, para generar el bloque “div_datos”.

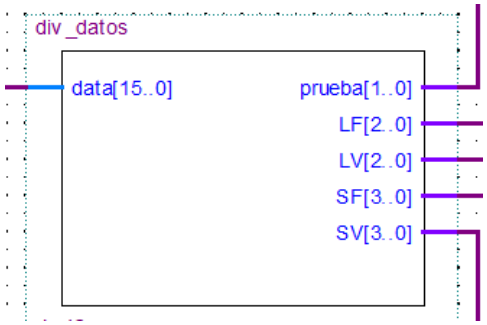


Figura 11. Bloque “div_datos”.

El cuarto código fue “mux_prueba”, es un multiplexor (MUX) que toma un vector de entrada de 2 bits (prueba) para seleccionar entre tres señales de entrada (x, y, z) y asigna la señal seleccionada a la salida qsel.

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

entity mux_prueba is
    Port ( prueba : in std_logic_vector (1 downto 0);
           x: in std_logic;
           y: in std_logic;
           z: in std_logic;
           qsel: out std_logic
    );
end mux_prueba;

architecture Behavioral of mux_prueba is
begin

    process(prueba, x, y, z)
    begin

    if prueba = "00" then
       qsel <= x;
    elsif prueba = "01" then
       qsel <= y;
    elsif prueba = "10" then
       qsel <= z;
    end if;

    end process;

end Behavioral;
```

Figura 12. Código VHDL, para generar el bloque “mux_prueba”.

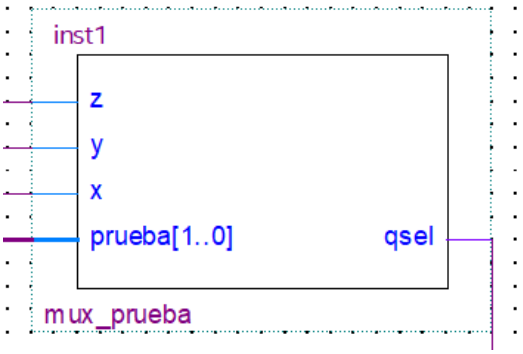


Figura 13. Bloque “mux_prueba”.

El quinto código fue “mux_liga”, es un multiplexor (MUX) de 2 a 1 para vectores de 3 bits. La señal de control qsel determina si la salida (edos) será el vector LF (si qsel es '1') o el vector LV (si qsel es '0').

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

entity mux_liga is
    Port (    LF : in std_logic_vector (2 downto 0);
              LV : in std_logic_vector (2 downto 0);
              qsel: in std_logic;
              edos : out std_logic_vector (2 downto 0)
    );
end mux_liga;

architecture Behavioral of mux_liga is
begin

    process(qsel, LF, LV)
    begin

    if qsel = '1' then
          edos <= LF;
    else
          edos <= LV;
    end if;

    end process;

end Behavioral;
```

Figura 14. Código VHDL, para generar el bloque “mux_liga”.

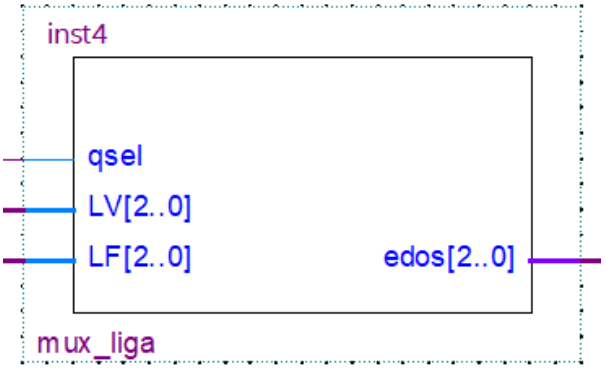


Figura 15. Bloque “mux_liga”.

El sexto código fue “mux_salida”, es un multiplexor (MUX) de 2 a 1, pero para vectores de 4 bits. La señal de control qsel selecciona si la salida (salida) será el vector SF o SV.

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

entity mux_salida is
    Port (   SF : in std_logic_vector (3 downto 0);
             SV : in std_logic_vector (3 downto 0);
             qsel: in std_logic;
             salida : out std_logic_vector (3 downto 0)
    );
end mux_salida;

architecture Behavioral of mux_salida is
begin

    process(qsel, SF, SV)
    begin

    if qsel = '1' then
          salida <= SF;
    else
          salida <= SV;
    end if;

    end process;

end Behavioral;
```

Figura 16. Código VHDL, para generar el bloque “mux_salida”.

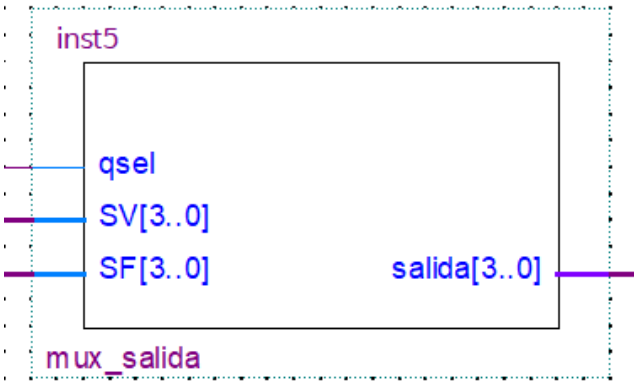


Figura 17. Bloque “mux_salida”.

Por último creamos el código de “div_frec”, toma una señal de reloj de entrada (reloj) y genera una señal de salida (div_clk) con una frecuencia mucho menor. Esto nos ayuda a observar ,los efectos de los leds en la salida.

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

entity div_frec is
    Port ( reloj : in std_logic;
    div_clk : out std_logic);
end div_frec;

architecture Behavioral of div_frec is
begin
    process (reloj)
    variable cuenta: std_logic_vector (27 downto 0):=x"0000000";
    begin

        if rising_edge (reloj) then
            if cuenta = x"2FAF080" then --25M pulsos
                cuenta:= x"0000000";
            else
                cuenta:= cuenta+1;
            end if;
        end if;
        div_clk <= cuenta(25);
    end process;
end Behavioral;
```

Figura 18. Código VHDL, para generar el bloque “div_frec”.

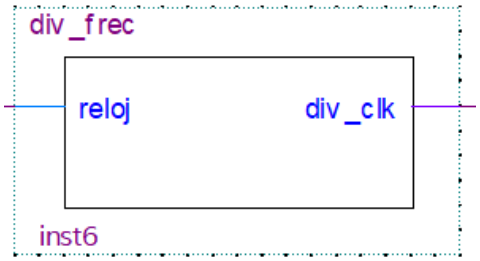


Figura 19. Bloque “div_frec”.

En el archivo esquemático, tenemos todos los bloques generados a partir del código vhdl, en este tenemos 5 entradas, “Reset”, “entrada[0]”, “entrada[1]”, “entrada[2]” y “CLK”. Además tenemos 6 salidas que serán asignadas a los leds de la tarjeta. “salida[0]”, “salida[1]”, “salida[2]”, “salida[3]”, “edop[0]”, “edop[1]”, “edop[2]”.

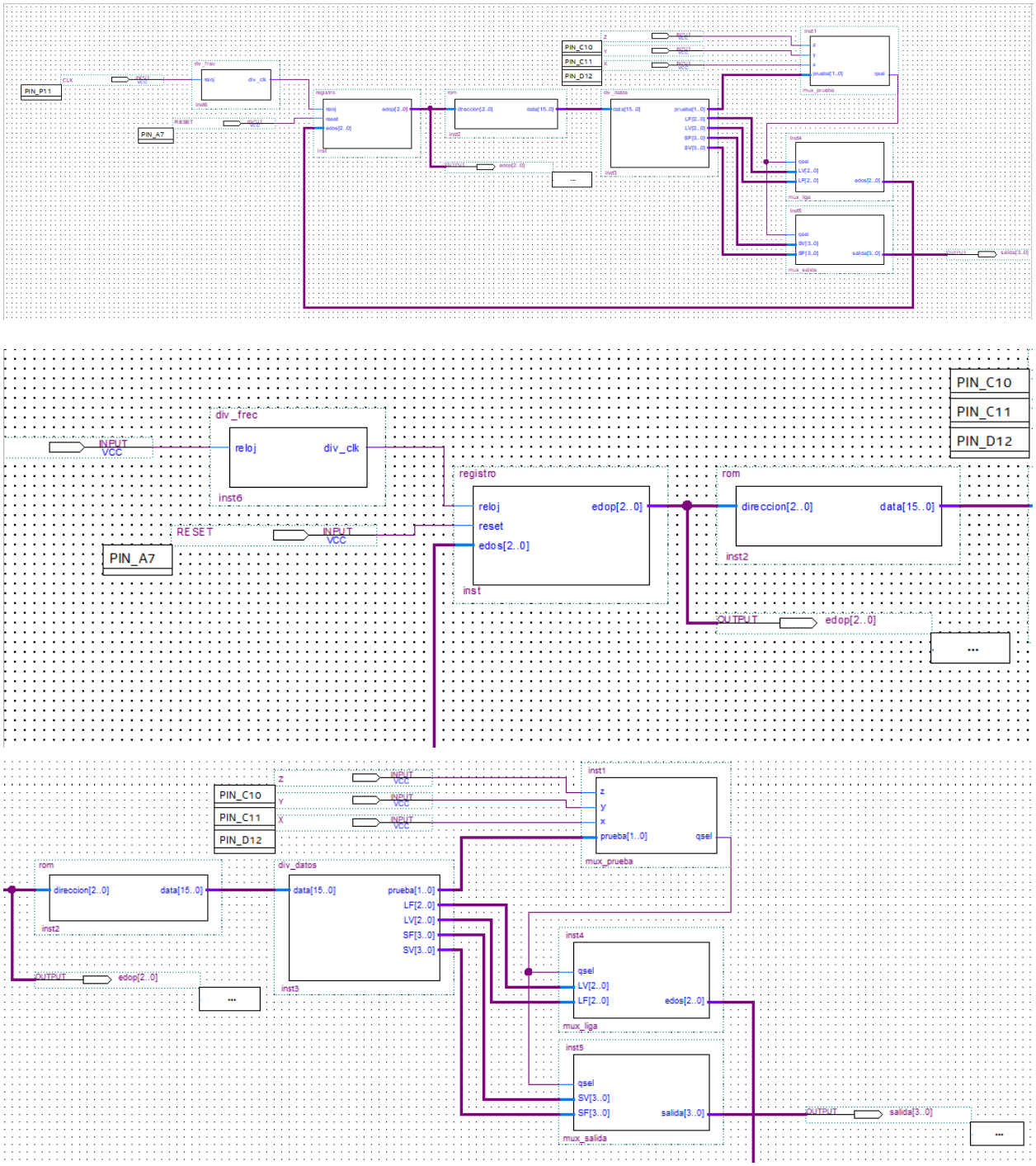


Figura 20. Diagrama de conexiones

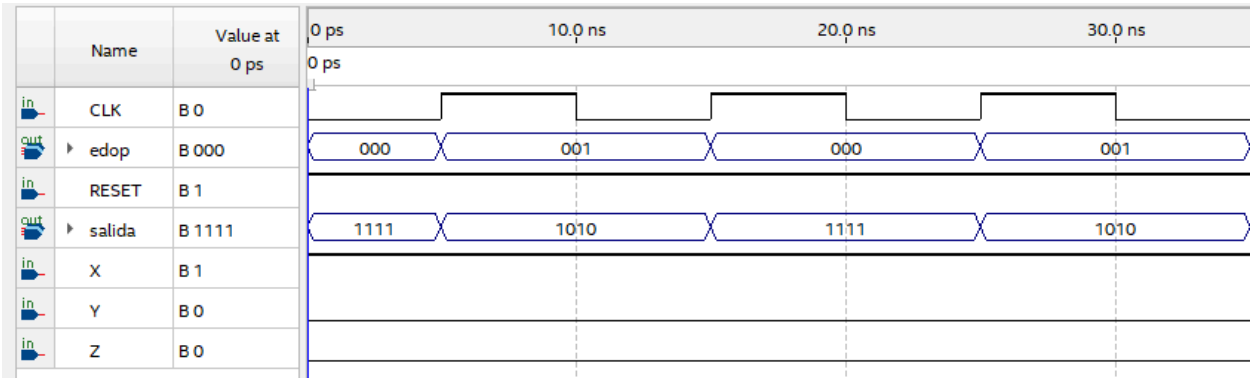
Procedemos con nuestro asignación de pines, para esto ocuparemos nuestro data sheet de nuestra FPGA DE-10 Lite, para hacer esto daremos click al icono que dice pin planner o presionando "ctrl+shift+n" los pines se colocan en el apartado de Location en la Fig. 21 podremos ver cuales se asignaron para este caso, recordemos que deben asignarse desde el más significativo al menos, para una correcta secuencia.

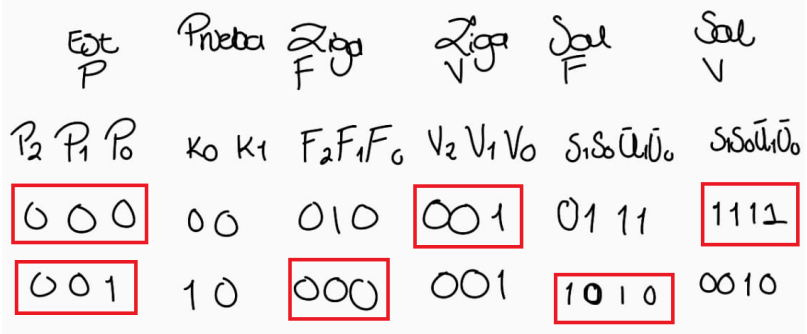
Node Name	Direction	Location
RESET	Input	PIN_A7
salida[0]	Output	PIN_A8
salida[1]	Output	PIN_A9
salida[2]	Output	PIN_A10
edop[1]	Output	PIN_A11
salida[3]	Output	PIN_B10
edop[2]	Output	PIN_B11
Z	Input	PIN_C10
Y	Input	PIN_C11
X	Input	PIN_D12
edop[0]	Output	PIN_D14
CLK	Input	PIN_P11

Figura 21. Asignación de pines

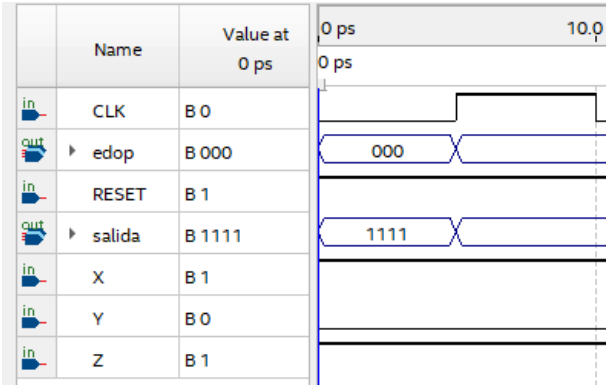
Simulación

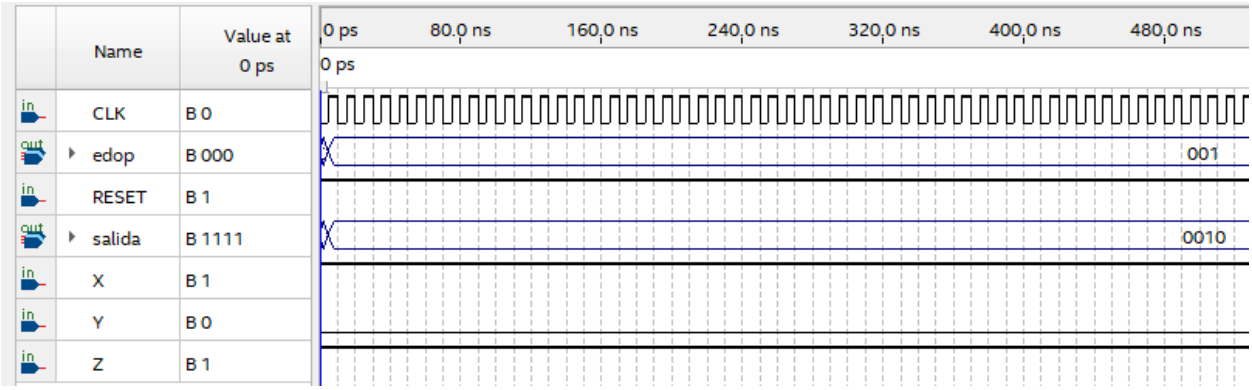
Secuencia con X=1, Z=0, estado 0 y estado 1.





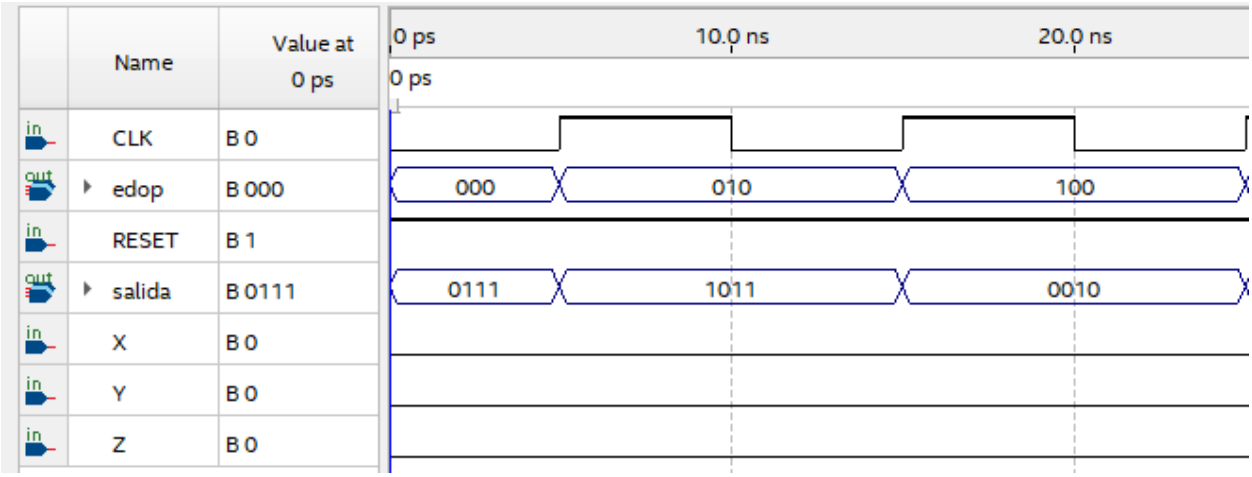
Secuencia con X=1, Z=1, estado 0 y estado 1.





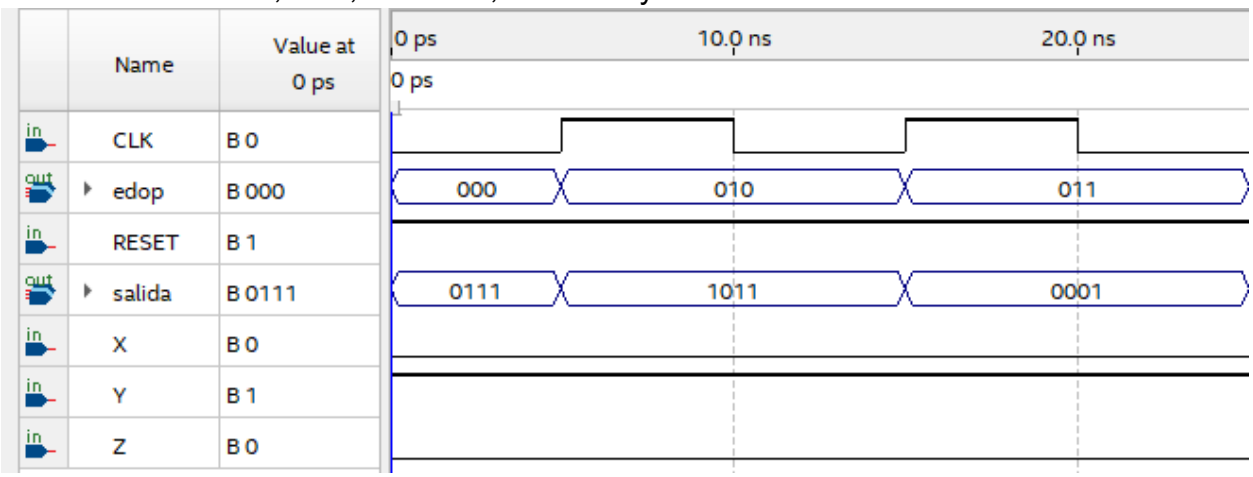
Est P	Prueba Ziga F	Ziga V	Sal F	Sal V
$P_2\ P_1\ P_0$	$K_0\ K_1$	$F_2F_1F_0$	$V_2\ V_1\ V_0$	$S_1S_0\bar{U}_1\bar{U}_0$	$S_1S_0\bar{U}_1\bar{U}_0$
0 0 0	0 0	0 1 0	0 0 1	0 1 1 1	1 1 1 1
0 0 1	1 0	0 0 0	0 0 1	1 0 1 0	0 0 1 0

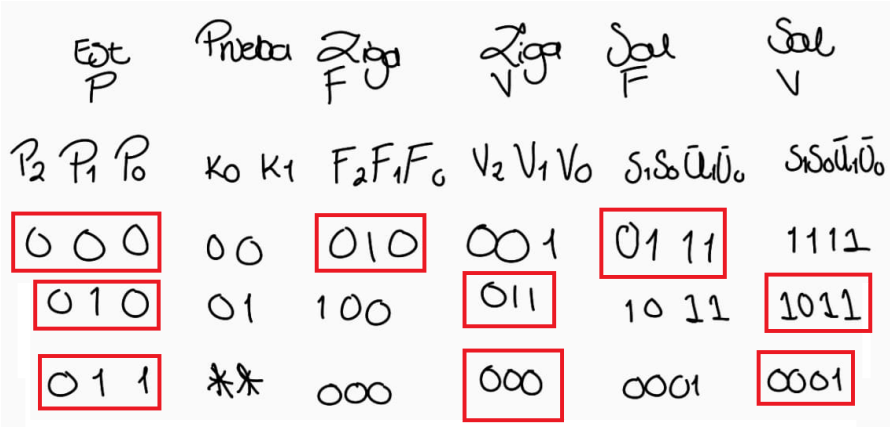
Secuencia con X=0, Y=0, estado 0, estado 2 y estado 4.



Est P	Prueba Ziga F	Ziga V	Sal F	Sal V
$P_2\ P_1\ P_0$	$K_0\ K_1$	$F_2F_1F_0$	$V_2\ V_1\ V_0$	$S_1S_0\bar{U}_1\bar{U}_0$	$S_1S_0\bar{U}_1\bar{U}_0$
0 0 0	0 0	0 1 0	0 0 1	0 1 1 1	1 1 1 1
0 1 0	0 1	1 0 0	0 1 1	1 0 1 1	1 0 1 1
1 0 0	* *	0 0 0	0 0 0	0 0 1 0	0 0 1 0

Secuencia con X=0, Y=1, estado 0, estado 2 y estado 3.





Conclusiones

Jiménez Treviño Emilio Cristóbal

En esta práctica se logró implementar el direccionamiento entrada-estado con el cual en nuestra FPGA se pudo observar cómo es que dependiendo de nuestras entradas en XYZ recorría cierto camino o otro, con esta práctica por fin pude afianzar mis conocimientos de llenado de la memoria ROM que siempre me costó mucho trabajo debido a que tuvimos muchos problemas con eso en esta práctica ya que no entendíamos ni que observar ni si estaba bien o mal al final se soluciono con ayuda de un ayudante de profesor que va conmigo en servicio que me explico, tambien reduje un poco el reloj para mejorarlo y verlo más fácil porque iba muy rápido.

Martinez Perez Brian Erik

Logramos cumplir con el propósito de implementar el direccionamiento entrada-estado. Los módulos VHDL permitieron construir una máquina de estados microprogramada. La ROM almacenaba las instrucciones con campos de prueba, liga y salida. Los multiplexores seleccionaron el próximo estado según condiciones externas. El registro almacenó y actualizó el estado presente. La simulación verificó transiciones y salidas correctamente. Además se notó el cambio en el tamaño de la memoria ROM ya que ahora pasamos a tener solo 8 registros respecto al método anterior de trayectoria, en el cual teníamos 64 registros.

Robles Jimenez Marco Antonio

Esta práctica me permitió entender con claridad el enfoque entrada-estado: trasladar la complejidad de la lógica a una ROM que, a partir del par *(estado presente, entradas)*, devuelve los campos de prueba, ligas y salidas, mientras que el registro conserva el estado y los multiplexores coordinan la transición y la activación de salidas en función de *qsel*. Al recorrer el flujo ASM → tabla → VHDL (registro, ROM, div_datos, mux_prueba, mux_liga, mux_salida, div_frec) → integración en Quartus → asignación de pines → simulación/DE10-LITE, confirmé la trazabilidad entre la carta ASM y cada palabra de la ROM es lo que garantiza que las transiciones observadas en hardware coinciden con el diseño, y la codificación compacta de estados hace que, para este caso, el mapa en memoria sea mucho más pequeño que con el direccionamiento por trayectoria.

Bibliografía

Laboratorio de Organización y Arquitectura de Computadoras. (2020, octubre 11). *Práctica No. 4 Construcción de Máquinas de estados Usando Memorias Direccionamiento Entrada - Estado.*