PRÁCTICA 4:

CONSTRUCCIÓN DE MÁQUINAS DE ESTADOS USANDO MEMORIAS Y DIRECCIONAMIENTO ENTRADA-ESTADO

INTEGRANTES:

BARRON PEREZ MARIAN ANDREA

PADILLA CASTILLO AARON SAMIR

GRUPO 3

Índice

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

Direccionamiento Entrada-Estado

Este tipo de direccionamiento se restringe a cartas ASM con una sola entrada por estado. Una nueva porción de la palabra de memoria contiene una representación binaria de la entrada a probar en cada estado, esta parte es llamada “la parte de prueba”. Con esta representación binaria un selector de entrada elige una de las variables de entrada. La parte de liga tiene dos estados siguientes, escogiéndose uno por el selector de liga, en base a la entrada seleccionada por la parte de prueba. La figura 1 muestra el diagrama de bloques de este método:

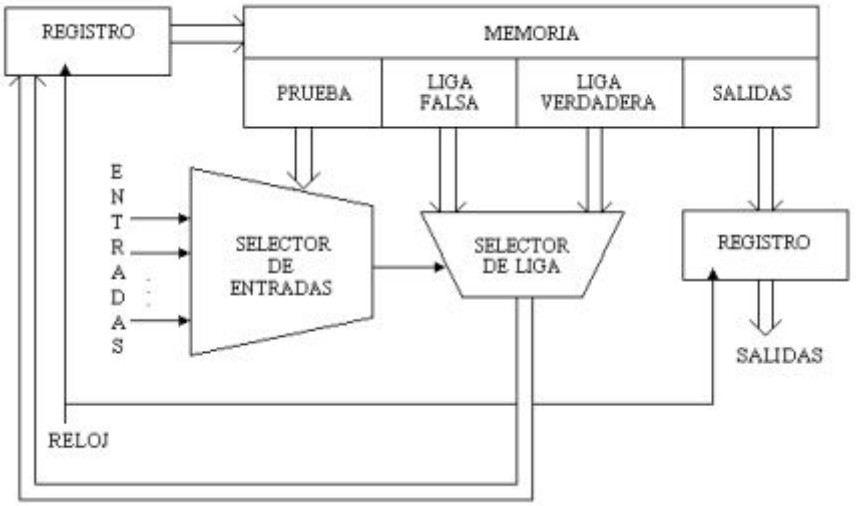

Figura 2: Diagrama de bloques para el método entrada-estado.

Si el valor de la entrada seleccionada por el selector de entradas es igual a cero, entonces el selector de liga elegirá la liga falsa, en caso contrario se seleccionará la liga verdadera; este caso se muestra en la figura 2.

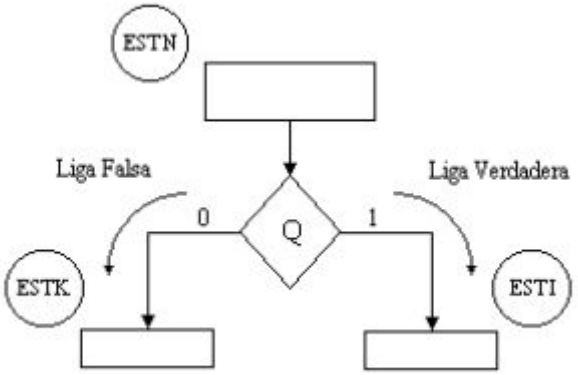

Figura 2: La liga falsa es el estado ESTK, mientras que la liga verdadera es el estado ESTI.

En esta práctica, dada una carta ASM se hará la construcción de máquinas de estado usando el direccionamiento de entrada-estado, así como la implementación en el software Quartus con sus respectivas simulaciones para probar el funcionamiento

Desarrollo

1. Dada la carta ASM de la figura 3, encuentre el contenido de memoria utilizando el direccionamiento entrada-estado. Recuerde que antes de construir la tabla se debe asignar a cada estado de la carta ASM una representacion bi- naria. Asimismo, no olvide asignar una representación binaria a las entradas.

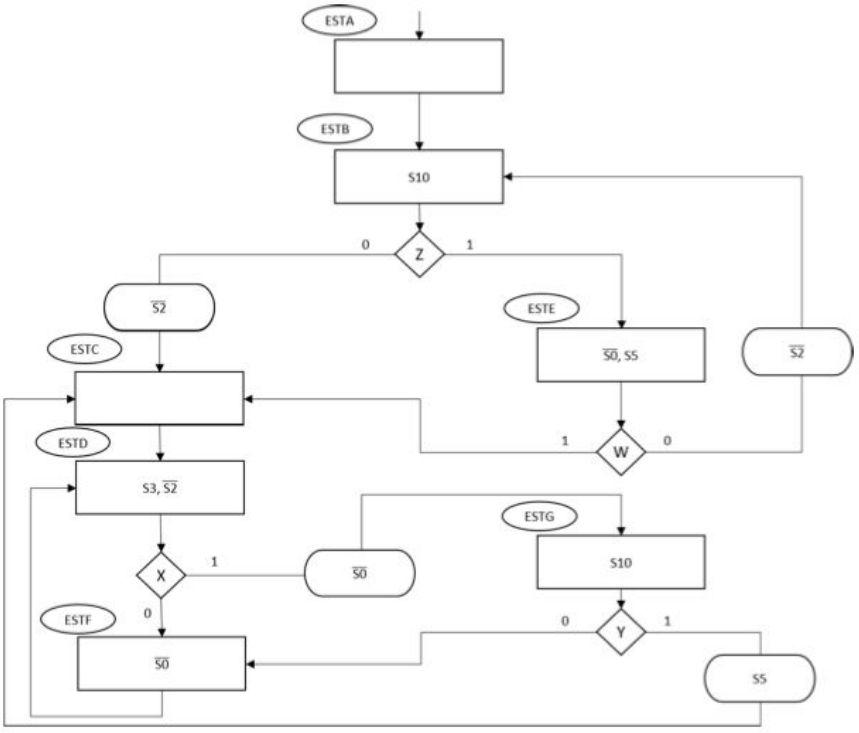

Figura 3: Carta ASM

El primer paso para comenzar a trabajar con el direccionamiento de trayectoria, es enumerar cada uno de los estados, por la implementación del mismo direccionamiento el orden que tenga no importa. A continuación se muestra nuestra propuesta:

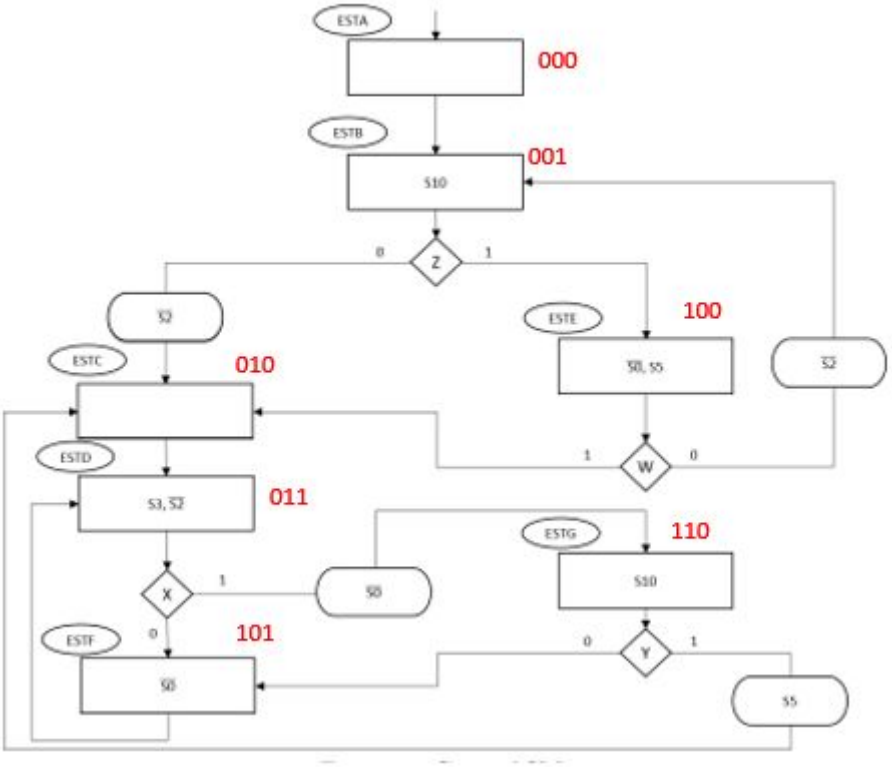

Figura 4: Carta ASM con enumeración

Entradas	
w	00
x	01
y	10
z	11

Salidas
s0
s2
s3
s5
s10

La figura anterior ya se encuentra codificada, esto con el fin de poder realizar la tabla con el contenido de estados. Para poder resolver la carta ASM por trayectoria es necesario poder ayudarnos de la tabla en donde encontramos cada uno de los registros.

Memoria Teórica

	Estado presente			Prueba		Liga falsa			Liga Verdadera			Salida Falsa					Salida verdadera				
												s0	s2	s3	s5	s10	s0	s2	s3	s5	s10
EstA	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	1	1	0	0	0
EstB	0	0	1	1	1	0	1	0	1	0	0	1	0	0	0	1	1	1	0	0	1
EstC	0	1	0	0	0	0	1	1	0	1	1	1	1	0	0	0	1	1	0	0	0
EstD	0	1	1	0	1	1	0	1	1	1	0	1	0	1	0	0	0	0	1	0	0
EstE	1	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	0
EstF	1	0	1	0	0	0	1	1	0	1	1	0	1	0	0	0	0	1	0	0	0
EstG	1	1	0	1	0	1	0	1	0	1	0	1	1	0	0	1	1	1	0	1	1

2. Una vez que haya obtenido el contenido de memoria, implemente el direccionamiento por entrada-estado utilizando el software de desarrollo Quartus y escriba el contenido de memoria obtenido.

Implementación

Código ROM

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.ALL;

ENTITY practica4 IS
	PORT(
		addr : IN std_logic_vector(2 DOWNTO 0); --estado presente, 3bits
		prueba : OUT std_logic_vector(1 DOWNTO 0); --va a multiplexorEntradas
		ligafalsa : OUT std_logic_vector(2 DOWNTO 0); --va a multiplexorSelectorLigas
		ligaverdadera : OUT std_logic_vector(2 DOWNTO 0); --va a multiplexorSelectorLigas
		salidafalsa : OUT std_logic_vector(4 DOWNTO 0); --va a multiplexorSalidas
		salidaverdadera : OUT std_logic_vector(4 DOWNTO 0) --va a multiplexorSalidas
	);
END practica4;

ARCHITECTURE behavioral OF practica4 IS
	TYPE mem_rom IS ARRAY(0 TO 7) OF std_logic_vector(17 DOWNTO 0); --memoria/matriz/ROM
	SIGNAL data_out : mem_rom; --declara data_out del tipo memoria rom
	--SIGNAL prueba : std_logic_vector(1 DOWNTO 0);
BEGIN
	data_out(0) <= "000010011100011000";
	data_out(1) <= "110101001000111001";
	data_out(2) <= "000110111100011000";
	data_out(3) <= "011011101010000100";
	data_out(4) <= "000010100001001010";
	data_out(5) <= "000110110100001000";
	data_out(6) <= "101010101100111011";

	PROCESS(addr) BEGIN
		prueba <= data_out(conv_integer(addr))(17 DOWNTO 16);
		ligafalsa <= data_out(conv_integer(addr))(15 DOWNTO 13);
		ligaverdadera <= data_out(conv_integer(addr))(12 DOWNTO 10);
		salidafalsa <= data_out(conv_integer(addr))(9 DOWNTO 5);
		salidaverdadera <= data_out(conv_integer(addr))(4 DOWNTO 0);
		--salidas <= data_out(conv_integer(addr))(5 DOWNTO 0);
	END PROCESS;
END ARCHITECTURE behavioral;
```

Código Registro

```
library ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;

--entra CLK, RESET, ESTADO SIGUIENTE
--sale ESTADO PRESENTE
```

```
ENTITY registro IS
	PORT(
		clk : IN std_logic;
		reset : IN std_logic;
		liga : IN std_logic_vector(2 DOWNTO 0); --data_in
		estadopresente : OUT std_logic_vector(2 DOWNTO 0) --data_out
	);
END registro;

ARCHITECTURE behavioral OF registro IS
	SIGNAL internal_value : std_logic_vector(2 DOWNTO 0) := B"000"; --ceros
BEGIN
	PROCESS (clk, reset, liga) BEGIN
		IF reset = '1' THEN
			internal_value <= B"000"; --inicias en ceros
		ELSIF rising_edge(clk) THEN
			internal_value <= liga; --liga a auxiliar
		END IF;
	END PROCESS;

	PROCESS(internal_value) BEGIN
		estadopresente <= internal_value; --auxiliar a estado presente
	END PROCESS;
END behavioral;
```

Código Multiplexor Entradas

```
--3 multiplexores: entradas, ligas, salidas

--Multiplexor 4 a 1, multiplexorEntradas
library IEEE;
use IEEE.std_logic_1164.all;

entity multiplexorEntradas is
   port (
      entrada : out std_logic; --de todas las entradas solo debe salir una
      W,X,Y,Z : in std_logic; --todas las entradas
      prueba : in std_logic_vector(1 downto 0) --viene de la memoria
   ); --prueba solo me dira cual de las cuatro entradas es la que se necesita ahora,
      --independientemente de si su valor es positivo o negativo, la necesitamos
      --para saber cual es la liga/salida correspondiente (la falsa o la verdadera?)
end entity;

architecture arch of multiplexorEntradas is

   -- Señal auxiliar para hacer más legible la selección.
   signal auxSelVect : std_logic_vector (1 downto 0);

begin

   auxSelVect(1) <= prueba(1);
   auxSelVect(0) <= prueba(0);
   --auxSelVect (prueba) = 00, 01, 10, 11

   -- La selección de entrada a poner en la salida..
   entrada <= W when auxSelVect = "00" else
      X when auxSelVect = "01" else
      Y when auxSelVect = "10" else
```

```
        Z when auxSelVect = "11";
end architecture;
```

Código Multiplexor Ligas

```
--3 multiplexores: entradas, ligas, salidas

--Multiplexor 2 a 1, multiplexorLigas
library IEEE;
use IEEE.std_logic_1164.all;

entity multiplexorLigas is
    port (
        liga : out std_logic_vector(2 downto 0); --solo debe salir una
        ligafalsa, ligaverdadera : in std_logic_vector(2 downto 0); --todas las ligas
        entrada : in std_logic --viene del multiplexorEntradas
    ); --entrada solo me dira cual de las dos ligas es la que corresponde
end entity;

architecture arch of multiplexorLigas is

    -- Señal auxiliar para hacer más legible la selección.
    signal auxSel : std_logic;

begin

    auxSel <= entrada;
    --auxSel (entrada) = 0, 1

    --La selección de liga a poner en la salida..
    liga <= ligafalsa when auxSel = '0' else
        ligaverdadera when auxSel = '1';
end architecture;
```

Código Multiplexor Salidas

```
--3 multiplexores: entradas, ligas, salidas

--Multiplexor 2 a 1, multiplexorSalidas
library IEEE;
use IEEE.std_logic_1164.all;

entity multiplexorSalidas is
    port (
        salida : out std_logic_vector(4 downto 0); --solo debe salir una
        salidafalsa, salidaverdadera : in std_logic_vector(4 downto 0); --todas las salidas
        entrada : in std_logic --viene del multiplexorEntradas
    ); --entrada solo me dira cual de las dos salidas es la que corresponde
end entity;

architecture arch of multiplexorSalidas is

    -- Señal auxiliar para hacer más legible la selección.
    signal auxSel : std_logic;

begin
```

```
    auxSel <= entrada;
    --auxSel (entrada) = 0, 1

    --La selección de liga a poner en la salida..
    salida <= salidafalsa when auxSel = '0' else
        salidaverdadera when auxSel = '1';
end architecture;
```

Circuito Secuencial

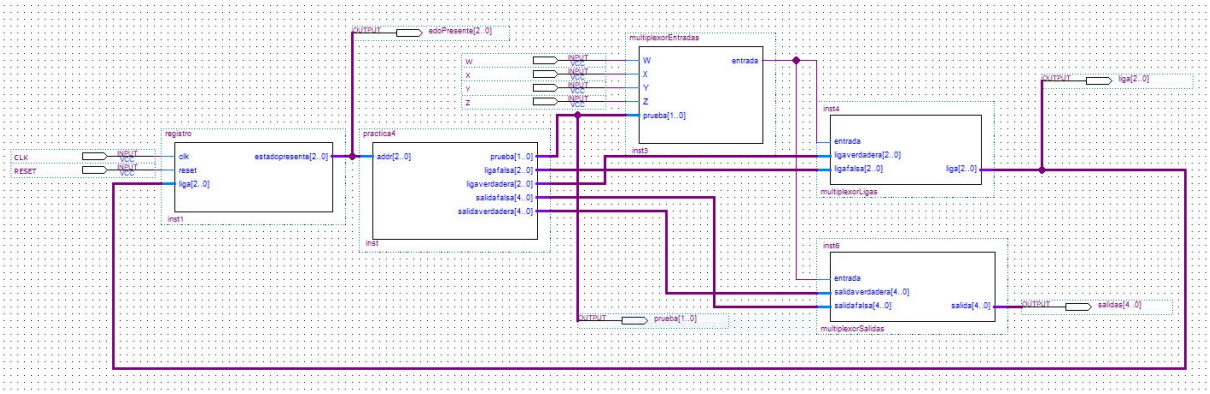

Figura 5: Circuito Secuencial

3. Simule su diseño para probar su funcionamiento. Recuerda mostrar el estado presente, el valor de la prueba, el estado siguiente y las salidas.

Simulación

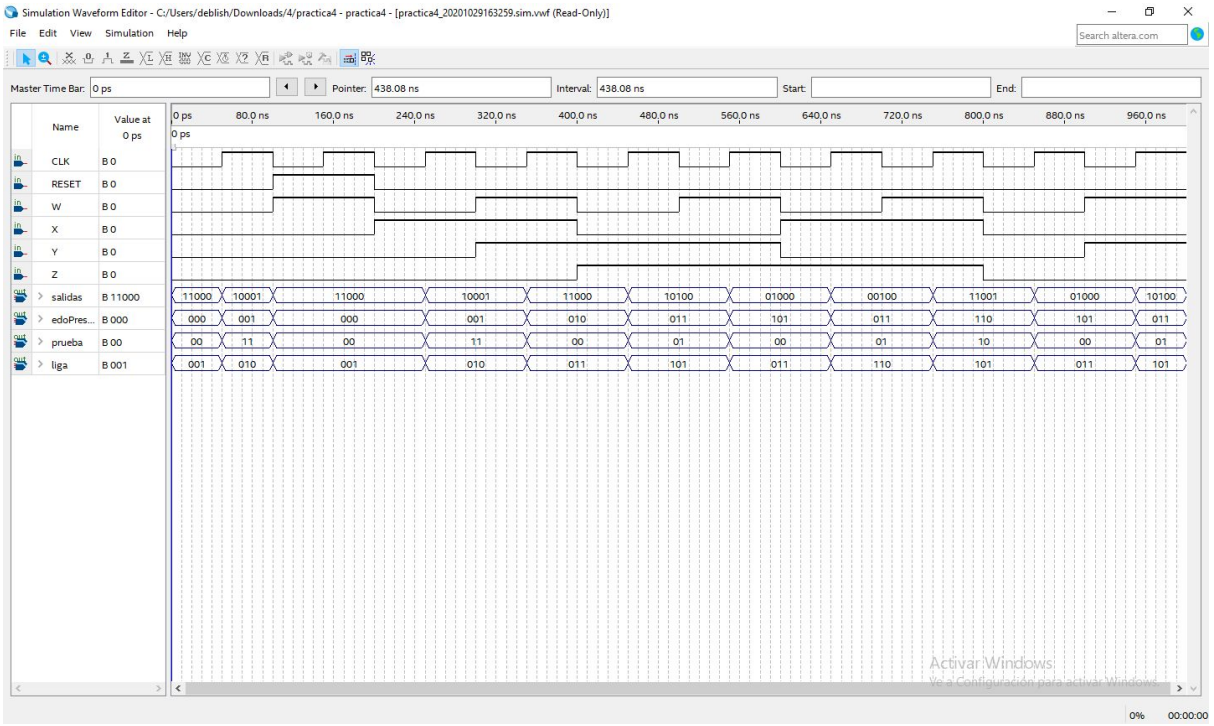

Figura 6: Captura de una simulación

Conclusiones

En esta práctica se cumplió con el objetivo de construcción de máquinas de estados usando direccionamiento de memorias con el método de direccionamiento entrada - estado. Así como también se pudo notar la eficiencia de este sobre el de trayectoria, porque es más óptimo en el ahorro de memoria al no tener que hacer tantas combinaciones para los diferentes casos de entrada con el estado presente.

Pero tiene un problema y es que no acepta entradas anidadas, dado que, al usar un multiplexor para la selección de prueba, analiza únicamente una entrada y esto al querer usar una decisión anidada es imposible de resolver. Sin embargo, se pudo solucionar la necesidad de esperar un ciclo para mostrar las salidas condicionales que dependen de las entradas.

Barron Perez Marian Andrea

El direccionamiento entrada-estado tiene una ventaja significativa sobre el de trayectoria debido a que se ahorra mucho espacio al colocar un multiplexor en las entradas, esto hace que en las tablas de transiciones no existan muchas combinaciones cuando se tiene entradas condicionales, ya que solo se toma el valor de salida, independientemente de la variable. Aunque si bien, es un buen método, aún no es posible manejar más de una entrada en un estado, pero es una buena aproximación a un buen modo de direccionamiento.

Padilla Castillo Aaron Samir

El direccionamiento por entrada-estado es un direccionamiento un poco más eficiente para implementar cartas ASM comparado con el direccionamiento por trayectoria, si bien tiene un uso menor de la memoria, la longitud de la pila se vuelve muy extensa. La ventaja que tiene en términos de facilidad de aplicación es muy notable, sin embargo, aplicar este tipo de direccionamiento físicamente también puede traer consigo costos elevados.

Participación en la práctica

Aportación teórica

Barron Perez Marian Andrea

Para la aportación teórica realice parte de la introducción, desarrollo y conclusiones junto con mi compañero Samir, ya que durante la videollamada de clase cada uno dio su punto de vista y su opinión, de tal manera que en conjunto se realizó cada punto anterior, ya que de esta manera fue más sencillo de realizar la práctica.

Padilla Castillo Aaron Samir

Las únicas partes que se hicieron de manera individual fueron las conclusiones, esta nueva sección de 'aportación' y la portada, el resto se hizo durante la videollamada de clase, donde se discutió cómo avanzar en el desarrollo y actividades de esta práctica (código y simulaciones).

Aportación experimental

Barron Perez Marian Andrea

Para la aportación experimental de igual manera en cómo se realizó la aportación teórica, se realizó en equipo de tal manera que hicimos una videollamada, para que se tuviera clara la idea de como realizar la tabla del contenido de memoria y para la programación en vhdl, ya que de esta manera no habria ningun problema al entender el código.

Padilla Castillo Aaron Samir

Como se dijo anteriormente, el aporte fue bilateral y secuencial vía videollamada. Hemos intentado no hacerlo independientemente para que ambos podamos observar los cambios y avances en tiempo real, ha tomado más tiempo pero nos hemos sentido cómodos con la manera de trabajo.

Referencias

Savage, J. (2015). Diseño de microprocesadores. En *Diseño de microprocesadores* (2015.ª ed., pp. 50-52). Facultad de ingeniería. https://classroom.google.com/c/MTYwNjg5NzQ1Mjgz/m/MTY5OTM1NDA5OTk4/details?hl=es