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

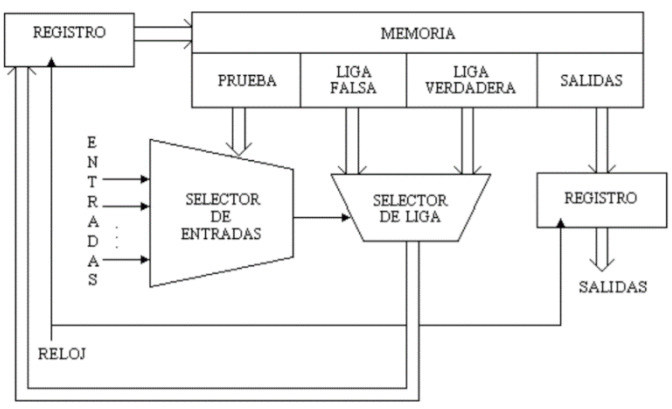
Existen dos limitantes básicas cuando se realizan diseños en dispositivos lógicos programables específicamente en FPGA, la primera es la cantidad de hardware utilizado en la aplicación y la otra es la frecuencia máxima a la cual puede ser usado el diseño, por dicha razón se debe buscar usar la menor cantidad de recursos y además de ello verificar que la implementación tenga un retardo de propagación tolerable.

Las máquinas de estados finitos se utilizan para describir cualquier tipo de sistema secuencial como control de otros bloques digitales o simplemente con descripción de muchas otras funciones digitales.

La descripción de estas máquinas de estados finitos, en un lenguaje de alto nivel se realiza de una forma sencilla, pero en ciertas ocasiones dicha descripción utiliza bastantes recursos del dispositivo, y además el tiempo de respuesta dado por los retardos de propagación es difícil de controlar. De aquí la necesidad de aprender a describirlas de formas diferentes buscando el diseño más eficiente, por lo cual es fundamental usar una arquitectura que toma como base el uso de un bloque ROM de la FPGA.

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 “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, con base a la entrada seleccionada por la parte de prueba.

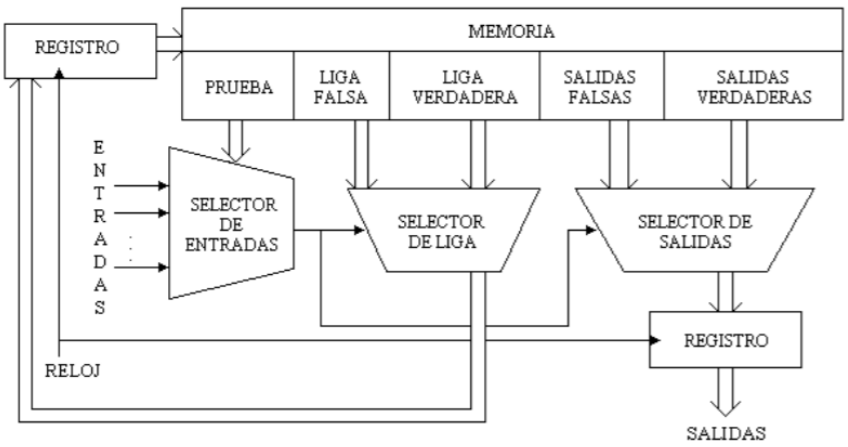


Si el valor de la entrada seleccionada por el selector de entradas es igual a cero, entonces el selector de liga elegirá la liga falsa, en caso contrario se seleccionará la liga verdadera.

Además de asignar una representación binaria a cada estado, también a cada variable de entrada se le asignará una.

Se utiliza también una variable auxiliar que nos sirve para los estados que no tengan variable de entrada, de manera que cuando en un estado no exista variable de entrada se probará la variable auxiliar, la cual tiene un valor preestablecido de cero ó uno. Así, si quisiéramos forzar a la máquina de estados para que salte a un estado específico se escogería la variable auxiliar Qx y dependiendo del valor de ésta (0 ó 1), en la liga falsa o verdadera según sea el caso, se tendría la dirección del estado a saltar.

Para manejar salidas condicionales es necesario modificar el diagrama de bloques del direccionamiento entrada-estado. Esta modificación consiste en tener dos campos de salidas: el campo de salidas falsas y el campo de salidas verdaderas. En el primero, están las salidas que se activan cuando la variable de entrada censada vale cero, y en el segundo, están las salidas que se activan cuando la variable de entrada censada vale uno



En la presente práctica se desarrollará una máquina de estados con direccionamiento Entrada – Estado dada una carta ASM, con la cual a través del programa Quartus la programaremos en el lenguaje VHDL y así mismo la simularemos para corroborar su funcionamiento.

Desarrollo

Como primer punto, analizamos la carta ASM que se muestra a continuación, otorgando los valores binarios para cada uno de los estados.

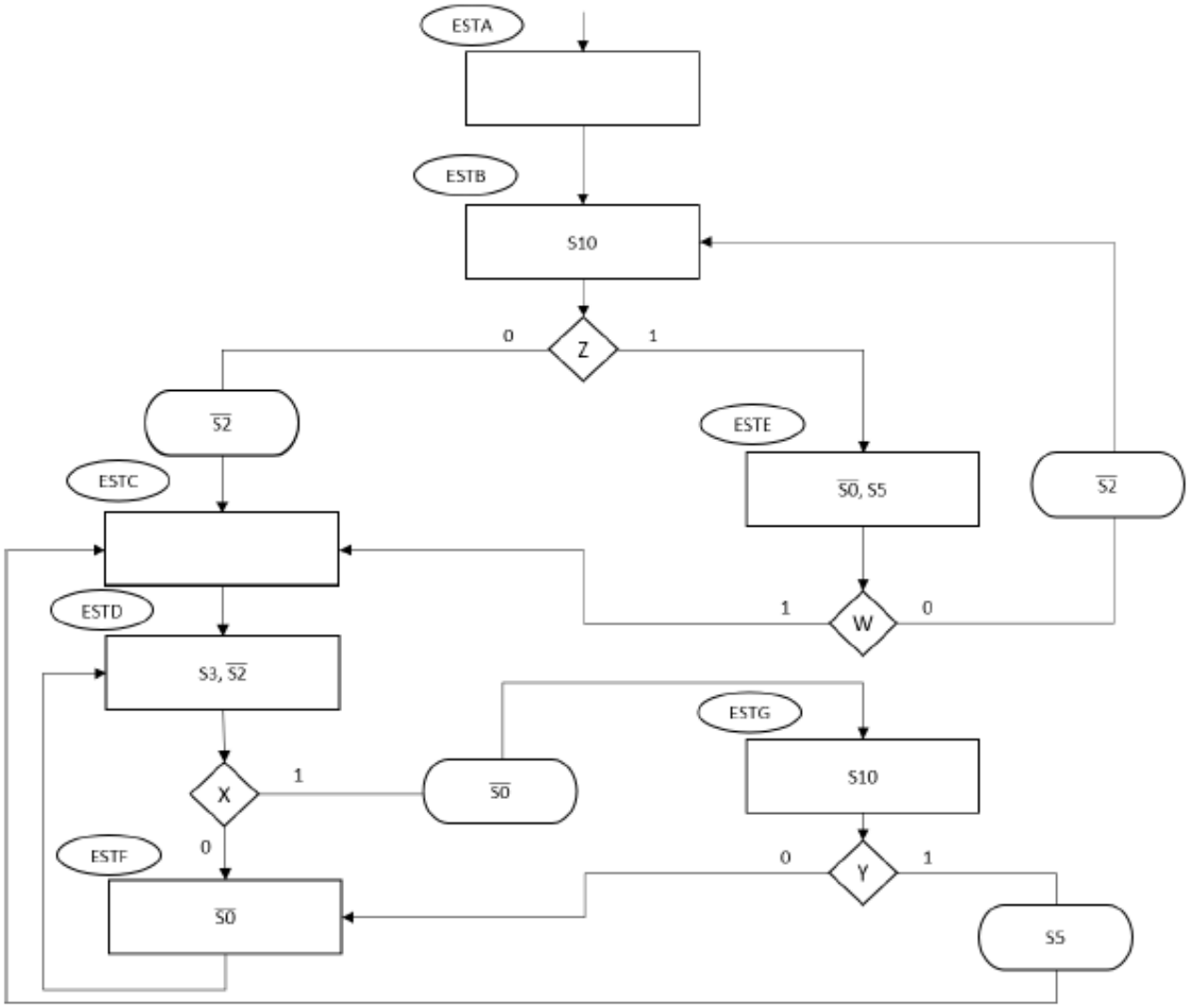


Figura 1.0 Carta ASM.

Analizando primeramente la carta ASM, realizamos la tabla de verdad correspondiente al comportamiento que esta tiene. Asignando inicialmente los valores binarios para cada uno de los estados y de los valores de prueba para identificarlos de manera correcta

Estados	
ESTA	000
ESTB	001
ESTC	010
ESTD	011
ESTE	100
ESTF	101
ESTG	110

Prueba	
W	00
X	01
Y	10
Z	11

Figura 1.1 Valores binarios de los estados y las entradas.

Con lo cual obtuvimos la siguiente tabla de verdad

Direccion			Contenido																	
Edo.Presente			Prueba		Liga Falsa			Liga Verdadera			Salidas Falsas					Salidas Verdaderas				
Q2	Q1	Q0	K1	K0	F2	F1	F0	V2	V1	V0	S0'	S2'	S3	S5	S10	S0'	S2'	S3	S5	S10
0	0	0	*	*	0	0	1	0	0	1	1	1	0	0	0	1	1	0	0	0
0	0	1	1	1	0	1	0	1	0	0	1	0	0	0	1	1	1	0	0	1
0	1	0	*	*	0	1	1	0	1	1	1	1	0	0	0	1	1	0	0	0
0	1	1	0	1	1	0	1	1	1	0	1	0	1	0	0	0	0	1	0	0
1	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	0
1	0	1	*	*	0	1	1	0	1	1	0	1	0	0	0	0	1	0	0	0
1	1	0	1	0	1	0	1	0	1	0	1	1	0	0	1	1	1	0	1	1
1	1	1	*	*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 1.2 Tabla de verdad de la carta ASM.

Con los valores obtenidos, implementamos la memoria en Quartus con la programación VHDL obteniendo el siguiente código:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY rom IS
   PORT (
       addr : IN std_logic_vector(2 DOWNTO 0);
       data : OUT std_logic_vector(17 DOWNTO 0)
   );
END rom;

ARCHITECTURE behavioral OF rom IS
   TYPE mem_rom IS ARRAY(0 TO 7) OF std_logic_vector(17 DOWNTO 0);
   SIGNAL data_out : mem_rom;
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
       data <= data_out(conv_integer(unsigned(addr)));
   END PROCESS;
END behavioral;
```

Figura 1.3 Código rom.vhd

Posteriormente, como se mencionó en la introducción es necesario hacer el uso de registros para poder implementar de manera correcta el direccionamiento entrada-estado, teniendo así un registro de entrada y uno de salida.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY registro_entrada IS PORT (
   clk : IN STD_LOGIC;
   reset : IN STD_LOGIC;
   estado_siguiente : IN std_logic_vector (2 DOWNTO 0);
   data_out : OUT std_logic_vector (2 DOWNTO 0));
END registro_entrada;
ARCHITECTURE Behavioral OF registro_entrada IS
   SIGNAL internal_value : std_logic_vector(2 DOWNTO 0) := B"000";
BEGIN
   PROCESS (clk, reset, estado_siguiente) BEGIN
       IF reset = '1' THEN
           internal_value <= B"000";
       ELSIF rising_edge (clk) THEN
           internal_value(2 downto 0) <= estado_siguiente;
       END IF;
   END PROCESS;
   PROCESS (internal_value) BEGIN
       data_out <= internal_value;
   END PROCESS;
END Behavioral;
```

Figura 1.4 Código registro_entrada.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY registro_salida IS PORT (
   clk : IN STD_LOGIC;
   reset : IN STD_LOGIC;
   data_in : IN std_logic_vector (4 DOWNTO 0);
   salidas : OUT std_logic_vector (4 DOWNTO 0));
END registro_salida;
ARCHITECTURE Behavioral OF registro_salida IS
   SIGNAL internal_value : std_logic_vector(4 DOWNTO 0) := B"00000";
BEGIN
   PROCESS (clk, reset, data_in) BEGIN
       IF reset = '1' THEN
           internal_value <= B"00000";
       ELSIF rising_edge (clk) THEN
           internal_value <= data_in;
       END IF;
   END PROCESS;
   PROCESS (internal_value) BEGIN
       salidas <= internal_value;
   END PROCESS;
END Behavioral;
```

Figura 1.5 Código registro_salida.vhd

Posteriormente se creó un separador de los datos almacenados en memoria para así poder asignar el estado siguiente, el valor de prueba, las ligas falsas y verdaderas, y finalmente las salidas de ambas ligas. Dichos valores se pasaron a los correspondientes selectores, los cuales se indicarán más adelante.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY separador IS PORT (
   data_in : IN std_logic_vector (17 DOWNTO 0);
   prueba: OUT std_logic_vector (1 DOWNTO 0);
   liga_falsa : OUT std_logic_vector (2 DOWNTO 0);
   liga_verdadera : OUT std_logic_vector (2 DOWNTO 0);
   salida_falsa : OUT std_logic_vector (4 DOWNTO 0);
   salida_verdadera : OUT std_logic_vector (4 DOWNTO 0));
END separador;
ARCHITECTURE Behavioral OF separador IS
BEGIN
   PROCESS (data_in) BEGIN
       prueba <= data_in(17 downto 16);
       liga_falsa <= data_in(15 downto 13);
       liga_verdadera <= data_in(12 downto 10);
       salida_falsa <=  data_in(9 downto 5);
       salida_verdadera <=  data_in(4 downto 0);
   END PROCESS;
END Behavioral;
```

Figura 1.6 Código separador.vhd

Como se mencionó anteriormente, para poder llevar a cabo la correcta selección de las salidas, dado el valor de las entradas, se generó un selector de entrada para poder enviar el valor binario correspondiente a cada una de las entradas; también se generó un selector de liga para poder escoger entre liga falsa o liga verdadera y por ende un selector de salida para poder arrojar la salida correspondiente según la liga que se seguía.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY selector_entrada IS PORT (
   prueba : IN std_logic_vector (1 DOWNTO 0);
   W : IN std_logic;
   X : IN std_logic;
   Y : IN std_logic;
   Z : IN std_logic;
   valor_entrada : OUT std_logic);
END selector_entrada;
ARCHITECTURE Behavioral OF selector_entrada IS
BEGIN
   PROCESS (prueba) BEGIN
       CASE(prueba) IS
           WHEN "00" =>
           valor_entrada <= W;
           WHEN "01" =>
           valor_entrada <= X;
           WHEN "10" =>
           valor_entrada <= Y;
           WHEN "11" =>
           valor_entrada <= Z;
           WHEN OTHERS =>
           valor_entrada <= W;
       END CASE;
   END PROCESS;
END Behavioral;
```

Figura 1.7 Código selector_entrada.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY selector_liga IS PORT (
   valor_entrada : IN std_logic;
   liga_falsa : IN std_logic_vector (2 DOWNTO 0);
   liga_verdadera : IN std_logic_vector (2 DOWNTO 0);
   liga : OUT std_logic_vector (2 DOWNTO 0));
END selector_liga;
ARCHITECTURE Behavioral OF selector_liga IS
BEGIN
   PROCESS (valor_entrada) BEGIN
       CASE(valor_entrada) IS
           WHEN '1' =>
           liga <= liga_verdadera;
           WHEN '0' =>
           liga <= liga_falsa;
       END CASE;
   END PROCESS;
END Behavioral;
```

Figura 1.8 Código selector_liga.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY selector_salida IS PORT (
   valor_entrada : IN std_logic;
   salida_falsa : IN std_logic_vector (4 DOWNTO 0);
   salida_verdadera : IN std_logic_vector (4 DOWNTO 0);
   salida : OUT std_logic_vector (4 DOWNTO 0));
END selector_salida;
ARCHITECTURE Behavioral OF selector_salida IS
BEGIN
   PROCESS (valor_entrada) BEGIN
       CASE(valor_entrada) IS
           WHEN '1' =>
           salida <= salida_verdadera;
           WHEN '0' =>
           salida <= salida_falsa;
       END CASE;
   END PROCESS;
END Behavioral;
```

Figura 1.9 Código selector_salida.vhd

Una vez realizado cada uno de los componentes necesarios para llevar a cabo el direccionamiento entrada-estado, obtuvimos cada uno de sus símbolos para poder unirlos en un esquema, obteniendo el siguiente resultado:

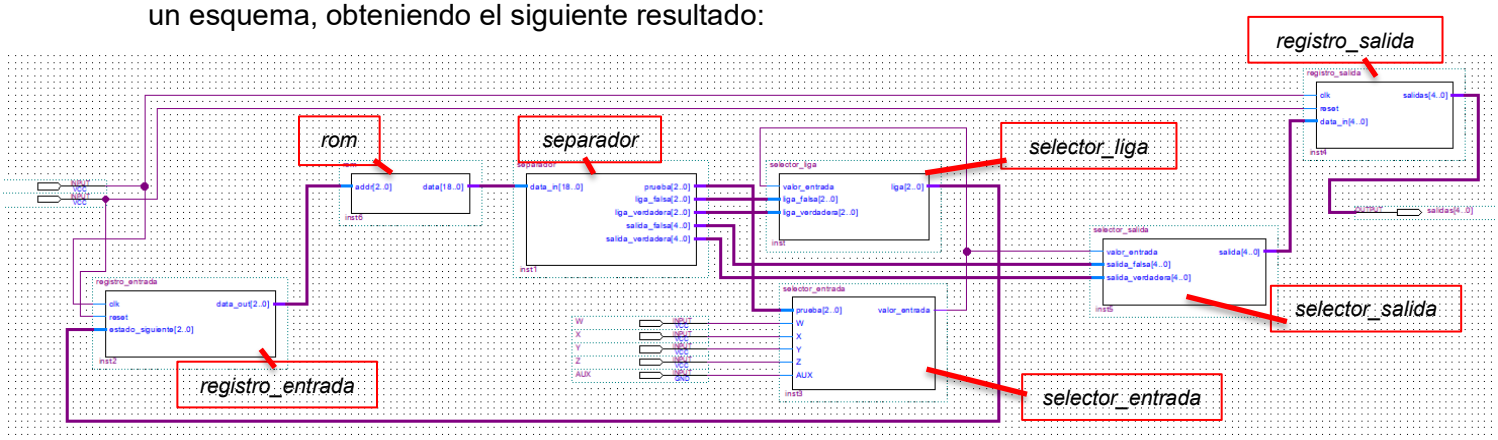


Figura 1.10 Esquema de la máquina de estados por direccionamiento entrada-estado

Como podemos observar se sigue prácticamente el diagrama mostrado en la introducción que pertenece al direccionamiento entrada-estado. Ahora bien, una vez compilado el bloque entero y asignado cada una de las entradas, así como la señal de reloj y de Reset, procedemos a simularlo

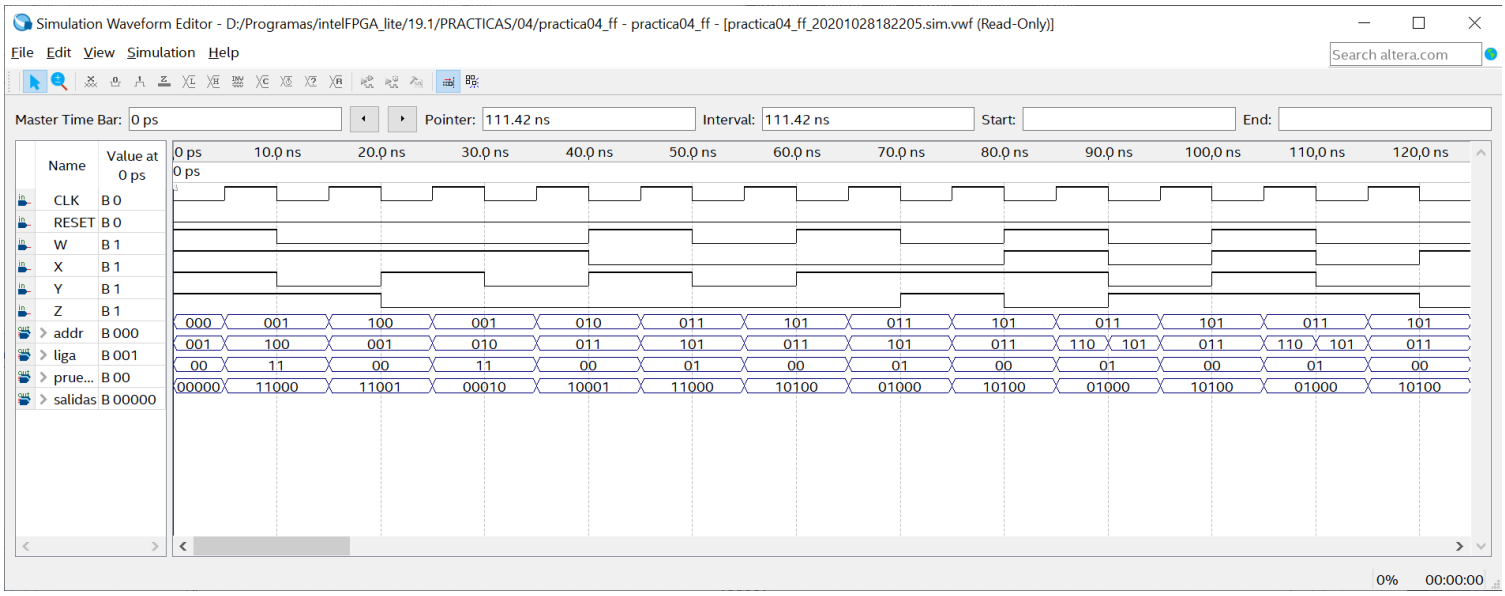


Figura 1.11 Simulación del Esquemático

Como podemos ver en la simulación se observan las señales de reloj (*clk*), reset, entradas, estado actual (*addr*), estado siguiente (*liga*), el valor de prueba (*prueba*) y finalmente las salidas (*salidas*).

Siguiendo la carta ASM y la tabla de verdad junto con los valores de las entradas, podemos comprobar que efectivamente se lleva a cabo de manera correcta el funcionamiento de la maquina de estados por direccionamiento entrada-estado.

Fuentes de Consulta

- Savage, J (2015) Diseño de microprocesadores. Facultad de Ingeniería. Universidad Nacional Autónoma de México. 482pp,
- Jacinto, E. (2013) Implementación de máquinas de estados basadas en rom. Consultado el 28 de octubre del 2020 de: https://dialnet.unirioja.es/descarga/articulo/5038459.pdf
- Chávez, N (S.F) Construcción de máquinas de estados usando memorias. Consultado el 28 de octubre del 2020 de: http://profesores.fi-b.unam.mx/normaelva/Direccionamientos.pdf