Objetivo

Familiarizar al alumno en el conocimiento de construcción de máquinas de estados usando direccionamiento de memorias con el método de direccionamiento por trayectoria.

Introducción

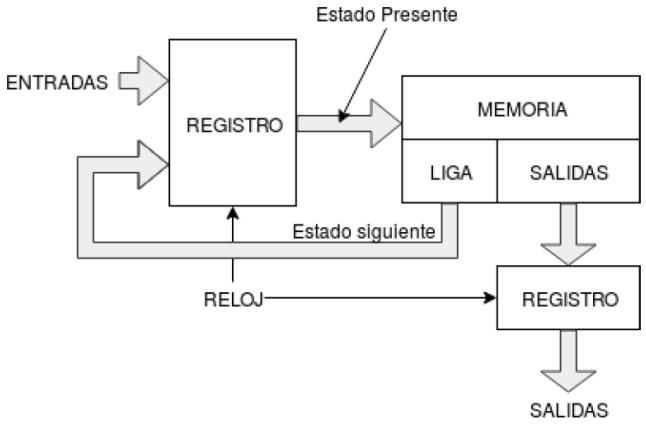
Existen dos limitantes básicas cuando se realizan diseños en dispositivos lógicos programables específicamente en FPGA, la primera es la cantidad de hardware utilizado en la aplicación y la otra es la frecuencia máxima a la cual puede ser usado el diseño, por dicha razón se debe buscar usar la menor 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 por trayectoria

Este tipo de direccionamiento guarda el estado siguiente y las salidas de cada estado de la carta ASM en una localidad de memoria. La porción de la memoria que indica el estado siguiente es llamada “LIGA”, mientras que la porción que indica las salidas es llamada “SALIDAS”.



Concatenando la liga del estado presente junto con las entradas se forma la dirección de memoria que contiene la dirección del estado siguiente. Esta dirección se guarda en un registro que está conectado a las líneas de dirección de la memoria, como se muestra en la figura. La señal de reloj conectada a los registros es la que indica la velocidad de la máquina de estados. Por lo tanto, para cada estado es necesario considerar todas las posibles combinaciones de las variables de entrada, aun cuando algunas de ellas no se utilicen.

Desarrollo

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

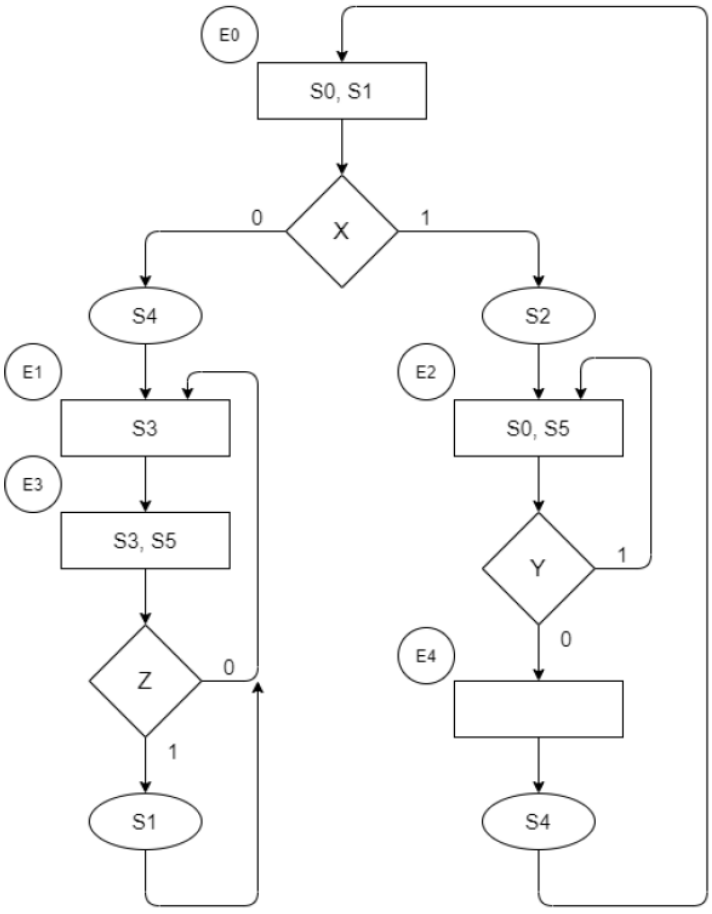


Figura 1.1 Tabla de verdad de la carta ASM.

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 así identificarlos de manera correcta

E0 = 000
E1 = 001
E2 = 010
E3 = 011
E4 = 100

Con lo cual obtuvimos la siguiente tabla de verdad

Direccion de Memoria						Contenido de la memoria								
Estado Presente			Entradas			Ligas			Salidas					
P2	P1	P0	X	Y	Z	L2	L1	L0	S0	S1	S2	S3	S4	S5
0	0	0	0	0	0	0	0	1	1	1	0	0	1	0
0	0	0	0	0	1	0	0	1	1	1	0	0	1	0
0	0	0	0	1	0	0	0	1	1	1	0	0	1	0
0	0	0	0	1	1	0	0	1	1	1	0	0	1	0
0	0	0	1	0	0	0	1	0	1	1	1	0	0	0
0	0	0	1	0	1	0	1	0	1	1	1	0	0	0
0	0	0	1	1	0	0	1	0	1	1	1	0	0	0
0	0	0	1	1	1	0	1	0	1	1	1	0	0	0
0	0	1	0	0	0	0	1	1	0	0	0	1	0	0
0	0	1	0	0	1	0	1	1	0	0	0	1	0	0
0	0	1	0	1	0	0	1	1	0	0	0	1	0	0
0	0	1	0	1	1	0	1	1	0	0	0	1	0	0
0	0	1	1	0	0	0	1	1	0	0	0	1	0	0
0	0	1	1	0	1	0	1	1	0	0	0	1	0	0
0	0	1	1	1	0	0	1	1	0	0	0	1	0	0
0	0	1	1	1	1	0	1	1	0	0	0	1	0	0
0	1	0	0	0	0	1	0	0	1	0	0	0	0	1
0	1	0	0	0	1	1	0	0	1	0	0	0	0	1
0	1	0	0	1	0	0	1	0	1	0	0	0	0	1
0	1	0	0	1	1	0	1	0	1	0	0	0	0	1
0	1	0	1	0	0	1	0	0	1	0	0	0	0	1
0	1	0	1	0	1	1	0	0	1	0	0	0	0	1
0	1	0	1	1	0	0	1	0	1	0	0	0	0	1
0	1	0	1	1	1	0	1	0	1	0	0	0	0	1
0	1	1	0	0	0	0	0	1	0	0	0	1	0	1
0	1	1	0	0	1	0	0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	0	1	0	0	0	1	0	1
0	1	1	0	1	1	0	0	1	0	1	0	1	0	1
0	1	1	1	0	0	0	0	1	0	0	0	1	0	1
0	1	1	1	0	1	0	0	1	0	1	0	1	0	1
0	1	1	1	1	0	0	0	1	0	0	0	1	0	1
0	1	1	1	1	1	0	0	1	0	1	0	1	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	1	0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	0	0	0	0	0	0	0	1	0
1	0	0	0	1	1	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	1	0	0	0	0	0	0	0	1	0
1	0	0	1	1	0	0	0	0	0	0	0	0	1	0
1	0	0	1	1	1	0	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	1	0	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	1	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0	0	0
1	1	0	1	1	1	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0
1	1	1	0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	1	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0

Figura 1.1 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 (
       clk : IN std_logic;
       addr : IN std_logic_vector(5 DOWNTO 0);
       data : OUT std_logic_vector(8 DOWNTO 0)
   );
END rom;

ARCHITECTURE behavioral OF rom IS
   TYPE mem_rom IS ARRAY(0 TO 63) OF std_logic_vector(8 DOWNTO 0);
   SIGNAL data_out : mem_rom;
BEGIN
   data_out(0) <= "001110010";
   data_out(1) <= "001110010";
   data_out(2) <= "001110010";
   data_out(3) <= "001110010";
   data_out(4) <= "010111000";
   data_out(5) <= "010111000";
   data_out(6) <= "010111000";
   data_out(7) <= "010111000";
   data_out(8) <= "011000100";
   data_out(9) <= "011000100";
   data_out(10) <= "011000100";
   data_out(11) <= "011000100";
   data_out(12) <= "011000100";
   data_out(13) <= "011000100";
   data_out(14) <= "011000100";
   data_out(15) <= "011000100";
   data_out(16) <= "100100001";
   data_out(17) <= "100100001";
   data_out(18) <= "010100001";
   data_out(19) <= "010100001";
   data_out(20) <= "100100001";
   data_out(21) <= "100100001";
   data_out(22) <= "010100001";
   data_out(23) <= "010100001";
   data_out(24) <= "001000101";
   data_out(25) <= "001010101";
   data_out(26) <= "001000101";
   data_out(27) <= "001010101";
   data_out(28) <= "001000101";
   data_out(29) <= "001010101";
   data_out(30) <= "001000101";
   data_out(31) <= "001010101";
   data_out(32) <= "000000010";
   data_out(33) <= "000000010";
   data_out(34) <= "000000010";
   data_out(35) <= "000000010";
   data_out(36) <= "000000010";
   data_out(37) <= "000000010";
   data_out(38) <= "000000010";
   data_out(39) <= "000000010";
   PROCESS (addr) BEGIN
       data <= data_out(conv_integer(unsigned(addr)));
   END PROCESS;
END behavioral;
```

Figura 1.2 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 por trayectoria, 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;
   entradas : IN std_logic_vector (2 DOWNTO 0);
   estado_siguiente : IN std_logic_vector (2 DOWNTO 0);
   data_out : OUT std_logic_vector (5 DOWNTO 0));
END registro_entrada;
ARCHITECTURE Behavioral OF registro_entrada IS
   SIGNAL internal_value : std_logic_vector(5 DOWNTO 0) := B"000000";
BEGIN
   PROCESS (clk, reset, entradas, estado_siguiente) BEGIN
       IF reset = '1' THEN
           internal_value <= B"000000";
       ELSIF rising_edge (clk) THEN
           internal_value(5 downto 3) <= estado_siguiente;
           internal_value(2 downto 0) <= entradas;
       END IF;
   END PROCESS;
   PROCESS (internal_value) BEGIN
       data_out <= internal_value;
   END PROCESS;
END Behavioral;
```

Figura 1.3 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 (5 DOWNTO 0);
   salidas : OUT std_logic_vector (5 DOWNTO 0));
END registro_salida;
ARCHITECTURE Behavioral OF registro_salida IS
   SIGNAL internal_value : std_logic_vector(5 DOWNTO 0) := B"000000";
BEGIN
   PROCESS (clk, reset, data_in) BEGIN
       IF reset = '1' THEN
           internal_value <= B"000000";
       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 1.4 Código registro_salida.vhd

Creando finalmente la implementación del contenido de la memoria en donde los primeros tres bits representan el estado siguiente, los cuales se conectarán al registro de entrada, y el resto de los bits son las salidas, las cuales se conectan al registro de salida.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY separador IS PORT (
   clk : IN STD_LOGIC;
   reset : IN STD_LOGIC;
   data_in : IN std_logic_vector (8 DOWNTO 0);
   estado_siguiente: OUT std_logic_vector (2 DOWNTO 0);
   salidas : OUT std_logic_vector (5 DOWNTO 0));
END separador;
ARCHITECTURE Behavioral OF separador IS
   SIGNAL internal_value : std_logic_vector(8 DOWNTO 0) := B"000000000";
BEGIN
   PROCESS (clk, reset, data_in) BEGIN
       IF reset = '1' THEN
           internal_value <= B"000000000";
       ELSIF rising_edge (clk) THEN
           internal_value <= data_in;
       END IF;
   END PROCESS;
   PROCESS (internal_value) BEGIN
       estado_siguiente <= internal_value(8 downto 6);
       salidas <= internal_value(5 downto 0);
   END PROCESS;
END Behavioral;
```

Figura 1.4 Código separador.vhd

Una vez realizado cada uno de los componentes necesarios para llevar a cabo el direccionamiento por trayectoria, obtuvimos cada uno de sus simbolos para poder unirlos en un esquema, obteniendo el siguiente resultado:

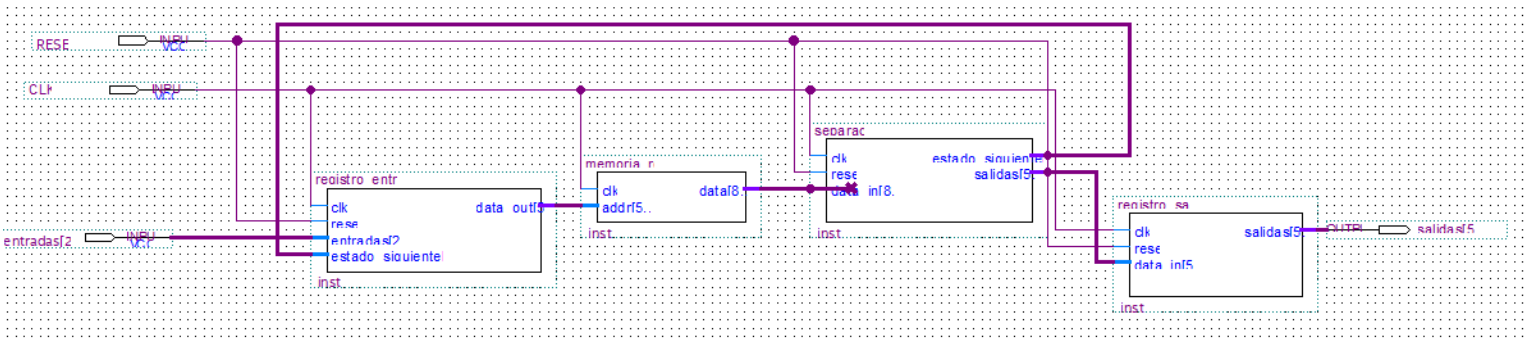


Figura 1.5 Esquema de la maquina de estados por Direccionamiento por trayectoria

Como podemos observar se sigue prácticamente el diagrama mostrado en la introducción que pertenece al direccionamiento por trayectoria. 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.

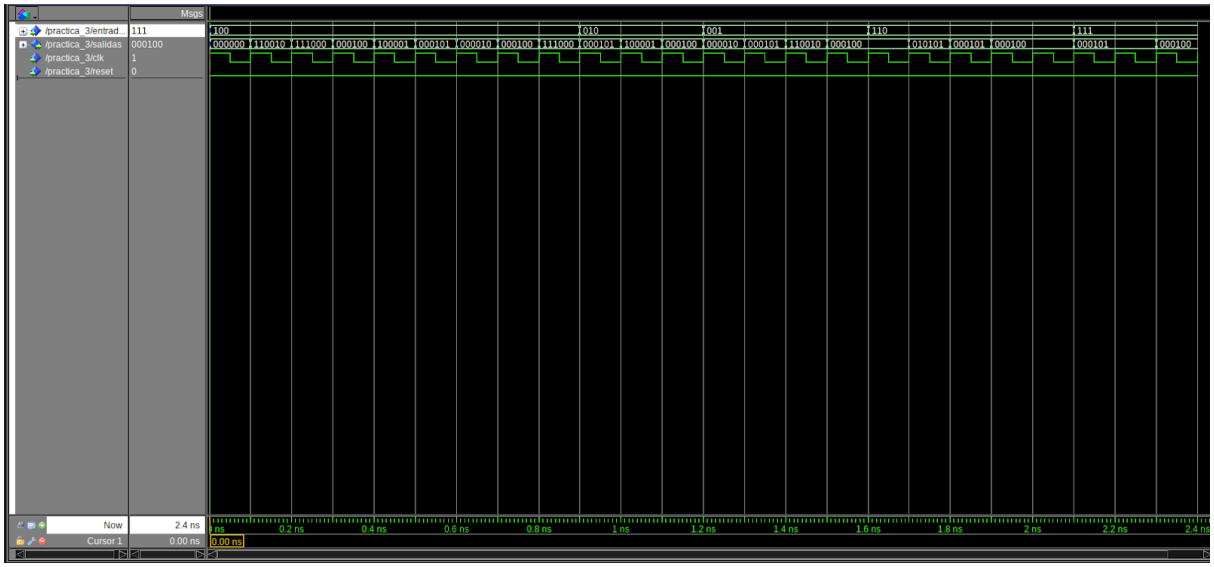


Figura 1.6 Simulación 01 del Esquema

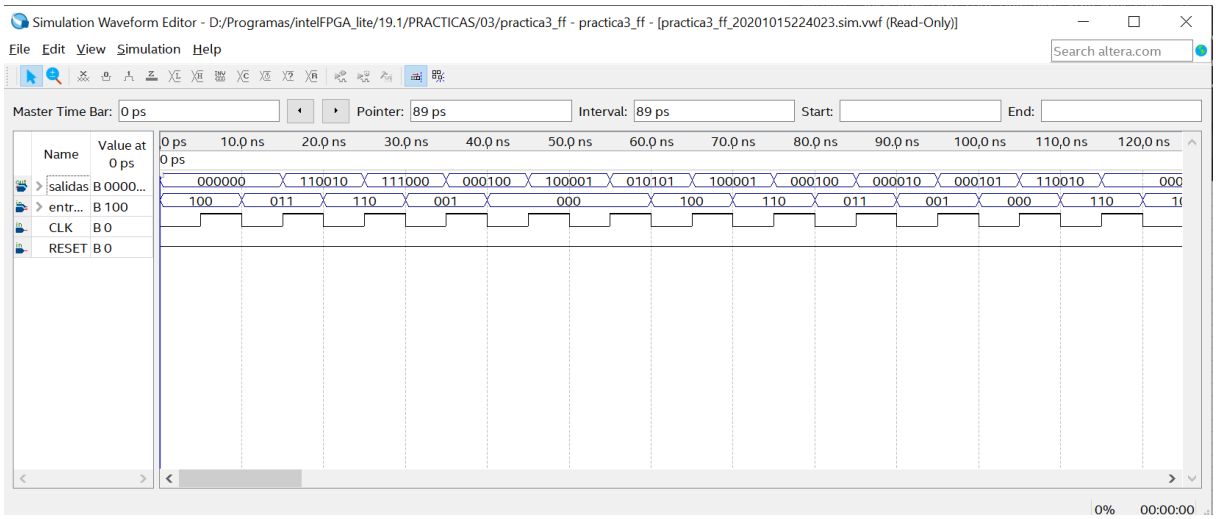


Figura 1.7 Simulación 02 del Esquema

Como logramos observar en las simulaciones se observa a la salida los valores que obtuvimos después del análisis de la carta ASM a través de la tabla de verdad, en donde efectivamente siguiendo las entradas que colocamos para cada una de las simulaciones se obtiene el valor esperado para cada uno de los estados en donde se encuentra el proceso, haciéndose efectivo el cambio de estado en cada ciclo de reloj, según está representado dentro de la carta ASM.

Fuentes de Consulta

- Jacinto, E. (2013) Implementación de máquinas de estados basadas en rom. Consultado el 19 de octubre del 2020 de: https://dialnet.unirioja.es/descarga/articulo/5038459.pdf
- Savage, J (S.F) Diseño de microprocesadores. Consultado el 19 de octubre del 2020 de: https://www.academia.edu/29332852/DISE%C3%91O_DE_MICROPROCESADORES
- Chávez, N (S.F) Construcción de máquinas de estados usando memorias. Consultado el 19 de octubre del 2020 de: http://profesores.fi-b.unam.mx/normaelva/Direccionamientos.pdf