Universidad Nacional Autónoma de México

Facultad de Ingeniería

Ingeniería en Computación

Laboratorio de Organización y Arquitectura de Computadoras

Práctica 05: Construcción de Máquinas de estado usando Memorias Direccionamiento Implícito

Alumnos:

- Murrieta Villegas Alfonso
- Reza Chavarria Sergio Gabriel
- Valdespino Mendieta Joaquin

Profesora: Ayesha Sagrario Román García

Grupo: 7

Fecha de entrega: 15 de octubre de 2021

Práctica 05: Construcción de Máquinas de estado usando Memorias Direccionamiento Implícito

Objetivo

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

Introducción

El direccionamiento implícito es una forma para implementar las máquinas de estados, sin embargo, esta tiene como característica principal que está restringida aquellas representaciones ya sea en ASM, donde solamente se valide una sola entrada por cada estado y además la estructura de saltos este de una determinada forma, se le denomina de esta forma por que aplica una propiedad implícita sobre un contador para hacer cargas en paralelo o autoincremento.

Este tipo de direccionamiento cuenta con 5 elementos de hardware, que son: memoria ROM, pequeño circuito lógico (multiplexor y 2 compuertas) y un contador, para el caso base, como se muestra a continuación.

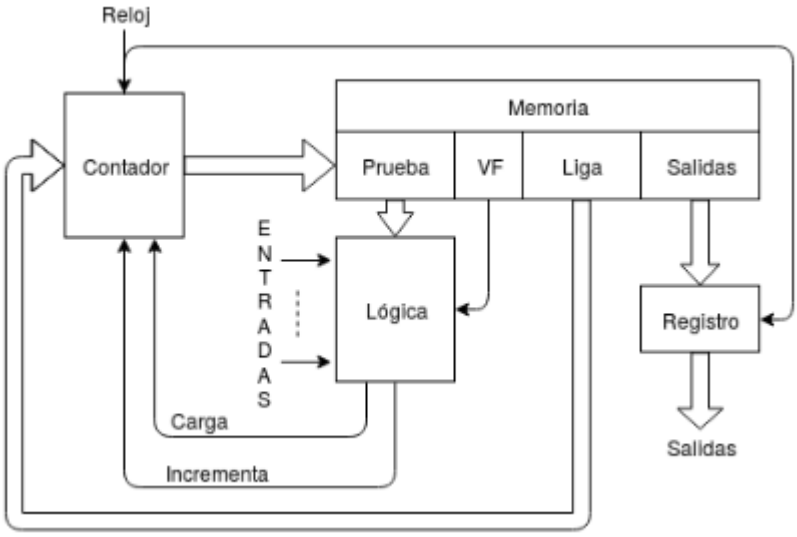


Ilustración 1: Direccionamiento Entrada Estado

La transición entre estados se da por la prueba (valores binarios de las entradas), la liga, y el valor VF, dependiendo la evaluación de dichas entradas dentro del circuito lógico se decide la carga de la liga en el contador o incrementar el valor del contador en uno.

Las ventajas claras de este tipo de direccionamiento son:

- Es el método que más ahorro de memoria tiene
- Solo hay cinco elementos de hardware
- Es versátil, para crear diversos sistemas con el mismo circuito basta con borrar y reprogramar memoria, usando

Sin embargo, la clara desventaja es que solo puede evaluar una entrada por estado, además de que está condicionado a aceptar determinados arreglos de estados al tomar una decisión, ya que, en evaluación de una entrada, forzosamente un estado de la decisión debe ser el subsecuente, Y para el caso base, no permite salidas condicionales, por lo que se tiene que cambiar la estructura, quedando de la siguiente forma

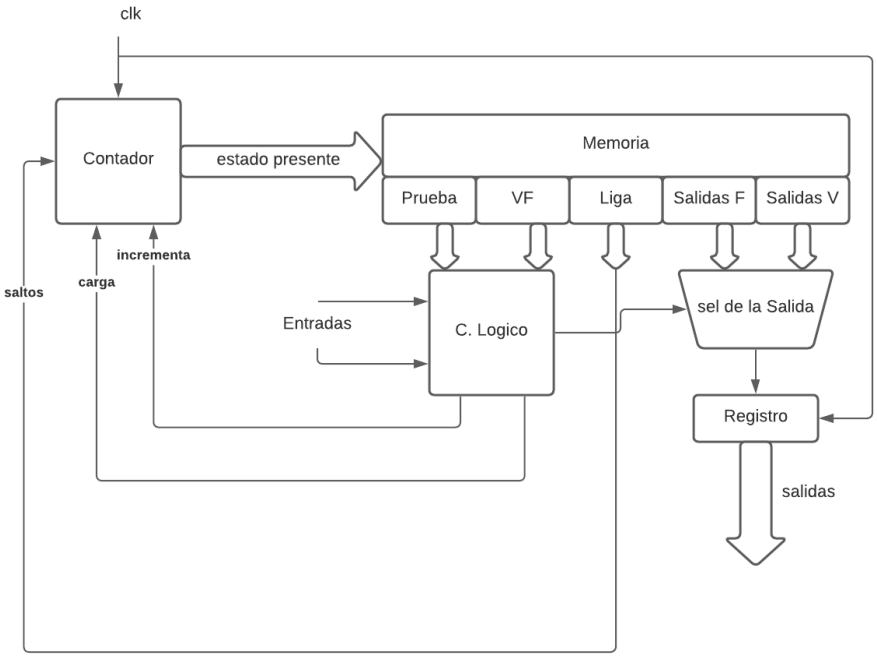
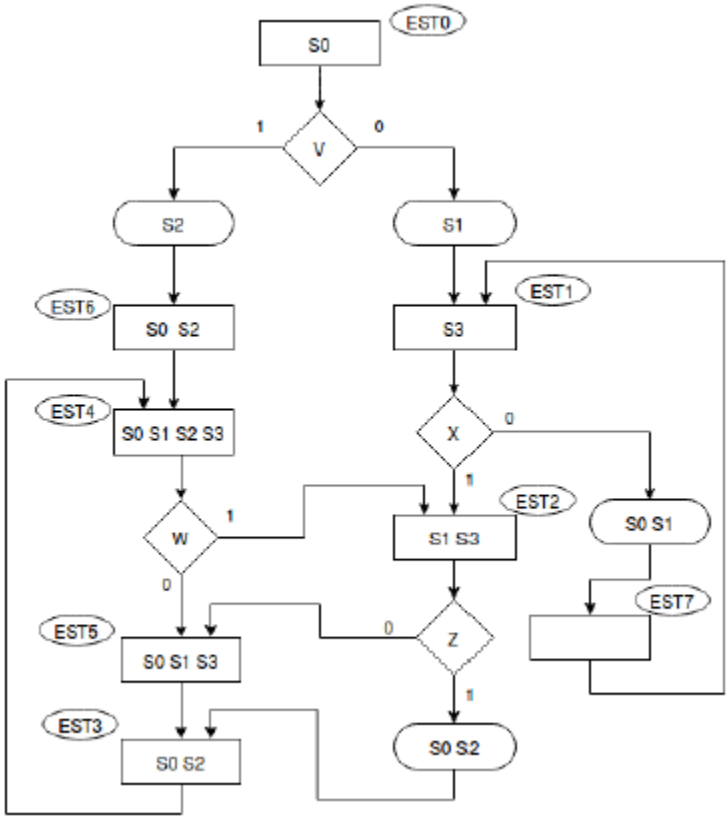


Ilustración 2: Direccionamiento Entrada Estado con salidas condicionales

En este caso, hay dos tipos de salidas, la salida observada depende si hay una salida condicional, es decir, que dependa de la toma de decisión o en otras palabras de la evaluación de la Entrada, en caso podrán variar entre salidas (Salidas F o V) o ser la misma si no se presentan las condicionales.

Desarrollo

1. Dada la carta ASM de la figura 4, encuentre el contenido de memoria utilizando el direccionamiento implícito. Recuerde que antes de construir la tabla se debe asignar a cada estado de la carta ASM una representación binaria. Así mismo, no olvide asignar una representación binaria a las entradas y la variable auxiliar.
2. Una vez que haya obtenido el contenido de memoria, implemente el direccionamiento implícito utilizando el software de desarrollo Quartus y escriba el contenido de memoria obtenido.
3. Simule su diseño para probar su funcionamiento. Sus simulaciones deben mostrar el contenido de la memoria, así como el estado presente.

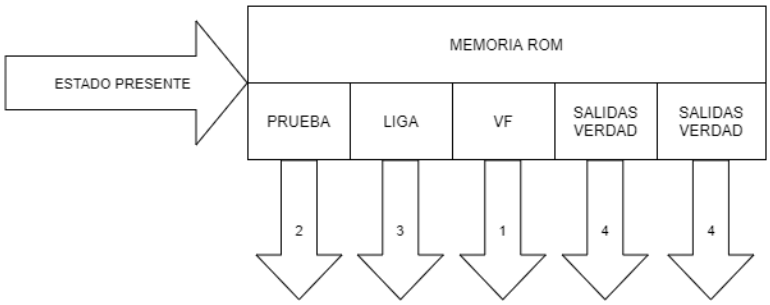


Desarrollo

Para la asignación de los estados se utilizarán 3 bits para la representación. Para las entradas se utilizaron 3 bits y se utilizará esta representación:

- $V = 000$
- $W = 001$
- $X = 010$
- $Z = 011$
- $Qaux = 100$

Se toma la dimensión de la memoria de la siguiente manera.



- Estado presente y liga: 3 bits
- Prueba: 3 bits
- VF: 1 bit
- Salidas verdaderas y falsas: 4 bits

Para el direccionamiento implícito se necesita tomar en cuenta en la toma de decisiones el salto del estado que no es continuo. Además, se necesita considerar las diferentes salidas por el uso de salidas condicionales. Para la resolución se necesita del siguiente encabezado para la tabla de verdad.

Estado Presente	**Prueba**	**Liga**	**VF**	**Salida Verdadera**	**Salida Falsa**
$\boldsymbol{P_2, P_1, P_0}$	K_2, K_1, K_0	V_2, V_1, V_0	VF	S_3, S_2, S_1, S_0	Z_3, Z_2, Z_1, Z_0

La tabla de verdad queda de la siguiente manera

Estado Presente P_2, P_1, P_0	Prueba K_2, K_1, K_0	Liga V_2, V_1, V_0	VF VF	Salida Verdadera S_3, S_2, S_1, S_0	Salida Falsa Z_3, Z_2, Z_1, Z_0
000	000	110	1	0101	0011
001	010	111	0	1000	1011
010	011	101	0	1111	1010
011	100	***	1	0101	0101
100	001	100	1	1111	1111
101	100	011	0	1011	1011
110	100	100	0	0101	0101
111	100	001	0	0000	0000

Ejercicio 2

Para la implementación de la máquina de estados se realizó con el manejo de los archivos *.smf. Creamos el archivo por medio de la configuración de la carta de manera gráfica.

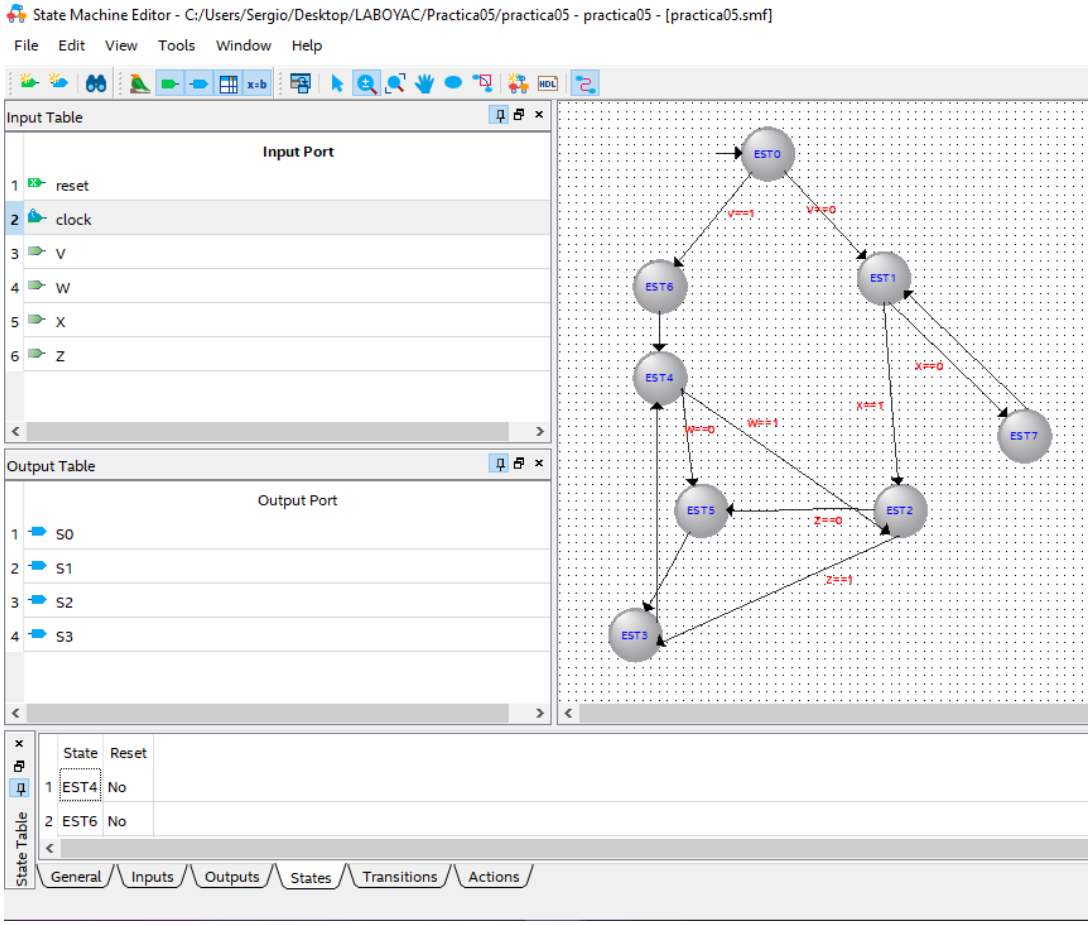


Ilustración 1: Creación de la máquina de estados por medio de State Machine File (.smf)*

Se implementaron los estados, entradas correspondientes, las transiciones con las condiciones correspondientes, las salidas y salidas condicionales. A continuación, se anexará la máquina de estados junto con el código generado en VHDL.

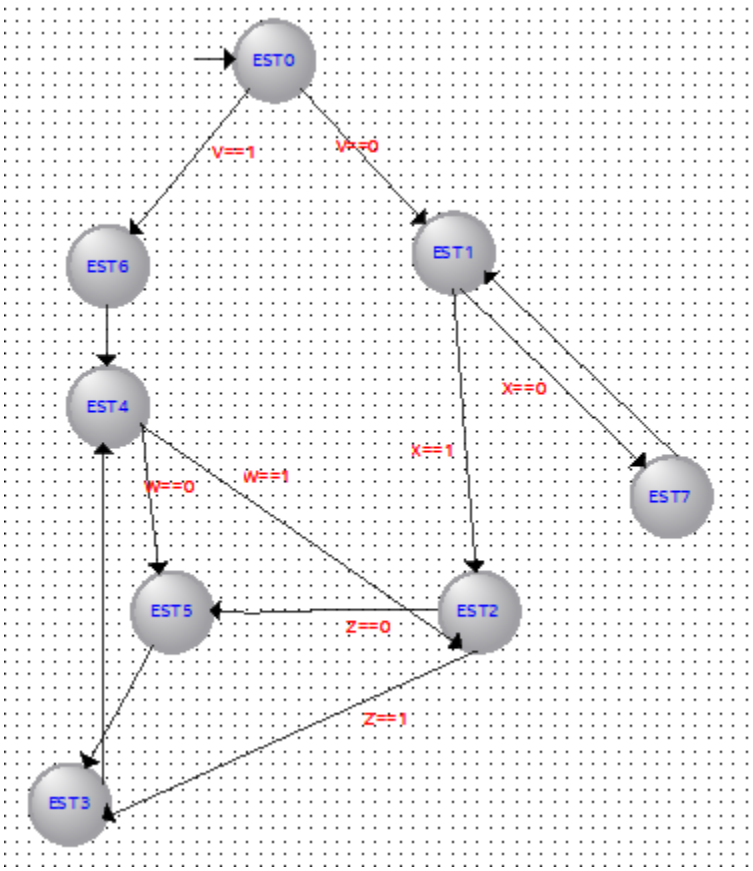


Ilustración 2: Máquina de estados generadas

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY practica05 IS
    PORT (
        reset : IN STD_LOGIC := '0';
        clock : IN STD_LOGIC;
        V : IN STD_LOGIC := '0';
        W : IN STD_LOGIC := '0';
        X : IN STD_LOGIC := '0';
        Z : IN STD_LOGIC := '0';
        S0 : OUT STD_LOGIC;
        S1 : OUT STD_LOGIC;
        S2 : OUT STD_LOGIC;
        S3 : OUT STD_LOGIC
    );
END practica05;

ARCHITECTURE BEHAVIOR OF practica05 IS
    TYPE type_fstate IS (EST4,EST6,EST0,EST1,EST5,EST3,EST7,EST2);
    SIGNAL fstate : type_fstate;
    SIGNAL reg_fstate : type_fstate;
BEGIN
    PROCESS (clock,reg_fstate)
    BEGIN
        IF (clock='1' AND clock'event) THEN
            fstate <= reg_fstate;
```

```
    END IF;
END PROCESS;

PROCESS (fstate,reset,V,W,X,Z)
BEGIN
    IF (reset='1') THEN
        reg_fstate <= EST0;
        S0 <= '0';
        S1 <= '0';
        S2 <= '0';
        S3 <= '0';
    ELSE
        S0 <= '0';
        S1 <= '0';
        S2 <= '0';
        S3 <= '0';
        CASE fstate IS
            WHEN EST4 =>
                IF ((W = '1')) THEN
                    reg_fstate <= EST2;
                ELSIF ((W = '0')) THEN
                    reg_fstate <= EST5;
                -- Inserting 'else' block to prevent latch inference
                ELSE
                    reg_fstate <= EST4;
                END IF;

                S2 <= '1';

                S3 <= '1';

                S1 <= '1';

                S0 <= '1';
            WHEN EST6 =>
                reg_fstate <= EST4;

                S2 <= '1';

                S3 <= '0';

                S1 <= '0';

                S0 <= '1';
            WHEN EST0 =>
                IF ((V = '1')) THEN
                    reg_fstate <= EST6;
                ELSIF ((V = '0')) THEN
                    reg_fstate <= EST1;
                -- Inserting 'else' block to prevent latch inference
                ELSE
                    reg_fstate <= EST0;
                END IF;

                IF ((V = '1')) THEN
                    S2 <= '1';
                -- Inserting 'else' block to prevent latch inference
                ELSE
                    S2 <= '0';
                END IF;
```

```
    S3 <= '0';

    IF ((V = '0')) THEN
        S1 <= '1';
    -- Inserting 'else' block to prevent latch inference
    ELSE
        S1 <= '0';
    END IF;

    S0 <= '1';
WHEN EST1 =>
    IF ((X = '1')) THEN
        reg_fstate <= EST2;
    ELSIF ((X = '0')) THEN
        reg_fstate <= EST7;
    -- Inserting 'else' block to prevent latch inference
    ELSE
        reg_fstate <= EST1;
    END IF;

    S2 <= '0';

    S3 <= '1';

    IF ((X = '0')) THEN
        S1 <= '1';
    -- Inserting 'else' block to prevent latch inference
    ELSE
        S1 <= '0';
    END IF;

    IF ((X = '0')) THEN
        S0 <= '1';
    -- Inserting 'else' block to prevent latch inference
    ELSE
        S0 <= '0';
    END IF;
WHEN EST5 =>
    reg_fstate <= EST3;

    S2 <= '0';

    S3 <= '1';

    S1 <= '1';

    S0 <= '1';
WHEN EST3 =>
    reg_fstate <= EST4;

    S2 <= '1';

    S3 <= '0';

    S1 <= '0';

    S0 <= '1';
WHEN EST7 =>
    reg_fstate <= EST1;
```

```
                    S2 <= '0';

                    S3 <= '0';

                    S1 <= '0';

                    S0 <= '0';
                WHEN EST2 =>
                    IF ((Z = '0')) THEN
                        reg_fstate <= EST5;
                    ELSIF ((Z = '1')) THEN
                        reg_fstate <= EST3;
                    -- Inserting 'else' block to prevent latch inference
                    ELSE
                        reg_fstate <= EST2;
                    END IF;

                    IF ((Z = '0')) THEN
                        S2 <= '0';
                    ELSIF ((Z = '1')) THEN
                        S2 <= '1';
                    -- Inserting 'else' block to prevent latch inference
                    ELSE
                        S2 <= '0';
                    END IF;

                    S3 <= '1';

                    S1 <= '1';

                    IF ((Z = '1')) THEN
                        S0 <= '1';
                    ELSIF ((Z = '0')) THEN
                        S0 <= '0';
                    -- Inserting 'else' block to prevent latch inference
                    ELSE
                        S0 <= '0';
                    END IF;
                WHEN OTHERS =>
                    S0 <= 'X';
                    S1 <= 'X';
                    S2 <= 'X';
                    S3 <= 'X';
                    report "Reach undefined state";
            END CASE;
        END IF;
    END PROCESS;
END BEHAVIOR;
```

Ejercicio 3

Para la prueba en la simulación se utilizó la herramienta de Simulation Waveform Editor. Se realizó los recorridos posibles para la visualización de la activación de las salidas correspondientes a las entradas y estados condicionales.

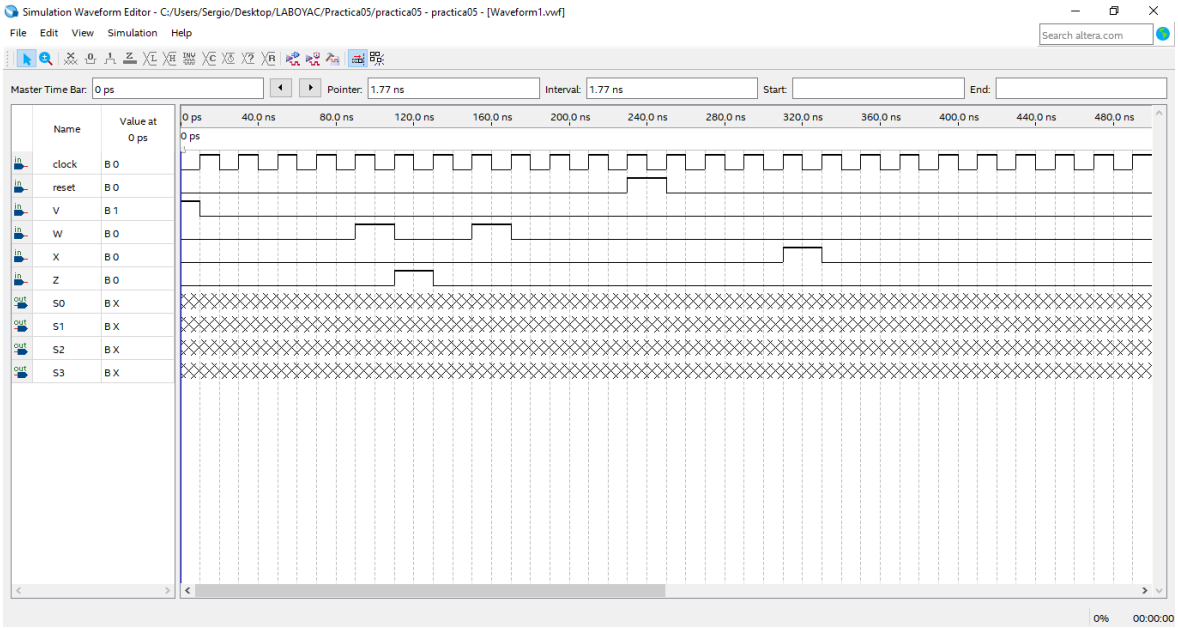


Ilustración 3: Asignación de entradas para la simulación.

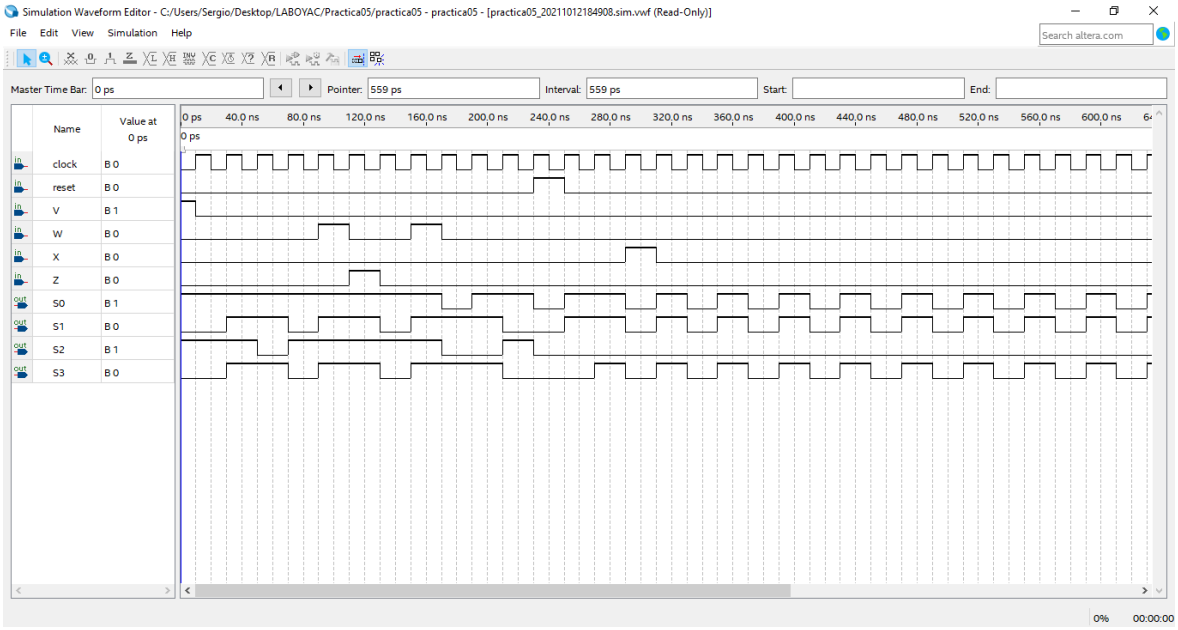


Ilustración 4: Obtención de comportamiento por las salidas

A continuación, se mencionarán los estados y las salidas correspondientes con respecto a la simulación realizada anteriormente.

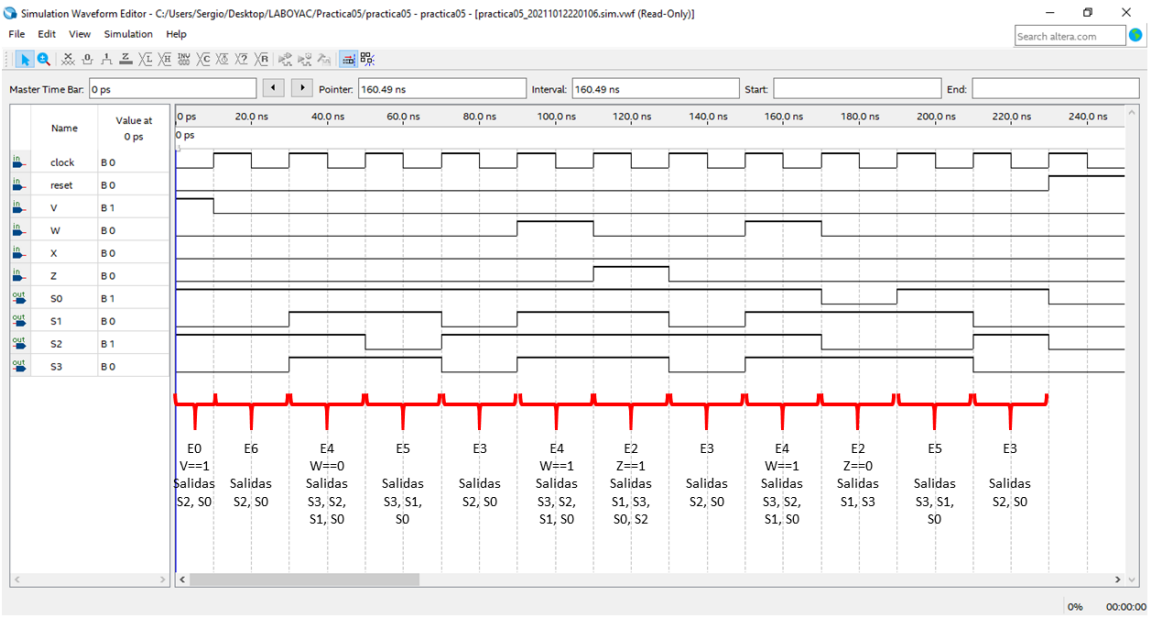


Ilustración 5: Simulación de la máquina de estados (parte 1)

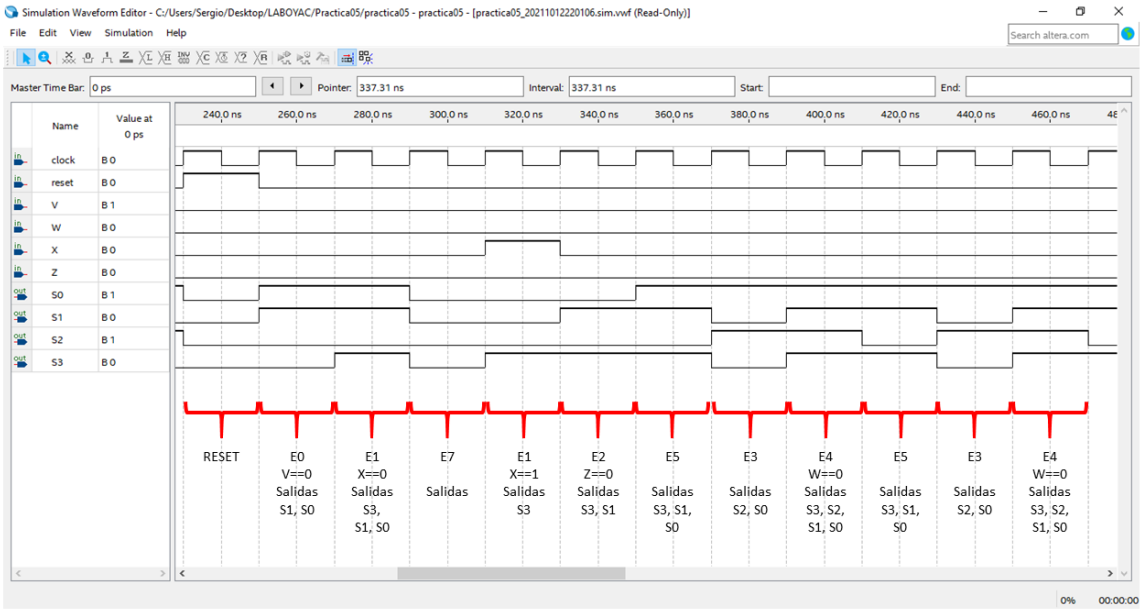


Ilustración 6: Simulación de la máquina de estados (parte 2)

Conclusiones

Murrieta Villegas Alfonso

En la presente práctica aprendimos a emplear el Método de Direccionamiento Implícito dentro de Máquinas de Estados con el objetivo de tener un manejo más simple en los saltos entre estados, además, y como parte de este método mejoramos el rendimiento de las instrucciones respecto al almacenamiento comparando con métodos previos.

Por último, y como parte principal de la práctica aprendimos a aterrizar mediante VHDL el direccionamiento de memoria lo cual es un gran avance para posteriores proyectos donde crearemos ya sea una unidad de memoria completa o incluso un procesador básico.

Reza Chavarria Sergio Gabriel

A partir del manejo de direccionamiento implícito de una máquina de estados se tiene un manejo más simple con respecto a los saltos entre estados. Esto mejora en el rendimiento y en el manejo de instrucciones con el almacenamiento de memoria menor en comparación entre los direccionamientos vistos anteriormente.

Valdespino Mendieta Joaquin

En la presente práctica, pudimos aplicar parte de los conceptos vistos acerca del direccionamiento implícito, mediante la creación de una máquina de estados, claramente tiene ventajas respecto a otros métodos, el más destacable es el uso de memoria, para el procesamiento de instrucciones ,sin embargo, también tiene limitaciones, relacionadas a la forma de transicionar, entre estados, ya que debe cumplir cierta estructura para no caer en errores, que podrían afectar posteriormente en la lógica de ejecución.

Bibliografía

- Savage J., Vázquez G & Chávez N. (2015). Diseño de Microprocesadores, Capítulo III Construcción De Máquinas de Estado usando Memorias. Encontrado el 9 de octubre del 2021. Sitio Web: https://biorobotics.fi-p.unam.mx/wp-content/uploads/Courses/arquitectura_de_computadoras/material_de_apoyo/diseo_de_procesadores.pdf