# Facultad de Ingeniería


Lab. Organización y Arquitectura de Computadoras

# Practica No. 4
# Construcción de Máquinas de estados Usando Memorias Direccionamiento Entrada - Estado

Alumnos

- Monsalvo Bolaños Melissa Monserrat
- Romero Andrade Cristian

Grupo: 01

Profesor
Ing. Adrian Ulises Mercado Martinez

Semestre
**2022–1**

Fecha de Entrega
14 de octubre de 2021

# Practica 4

Monsalvo Bolaños Melissa Monserrat y Romero Andrade Cristian

## Índice

## 1. Introducción

El direccionamiento entrada-estado 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, encogiéndose uno por el selector de liga, en 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. Este método tiene grandes ventajas como el ahorro de memoria, que cuenta con pocos elementos de hardware y que representa un sistema muy versátil.

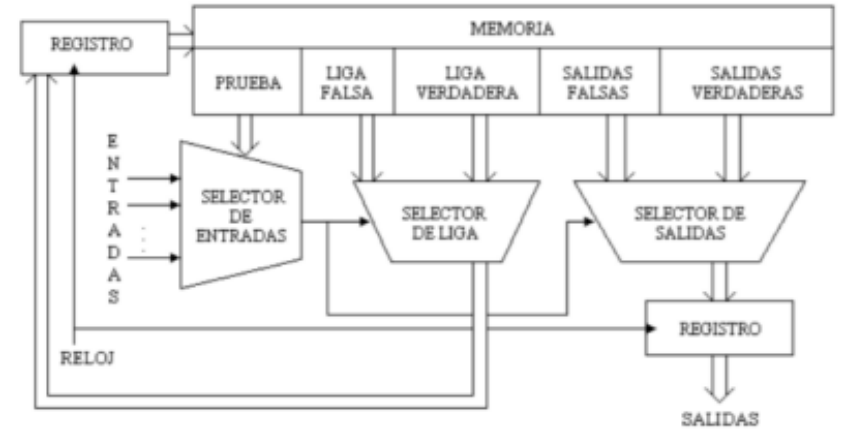


Figura 1: Direccionamiento Entrada-Estado

## 2. Objetivo

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

## 3. Desarrollo

Empezamos analizando siguiente figura, donde esta es una carta ASM donde secuencialmente otorgamos los valores binarios de los estados y para los estados de prueba

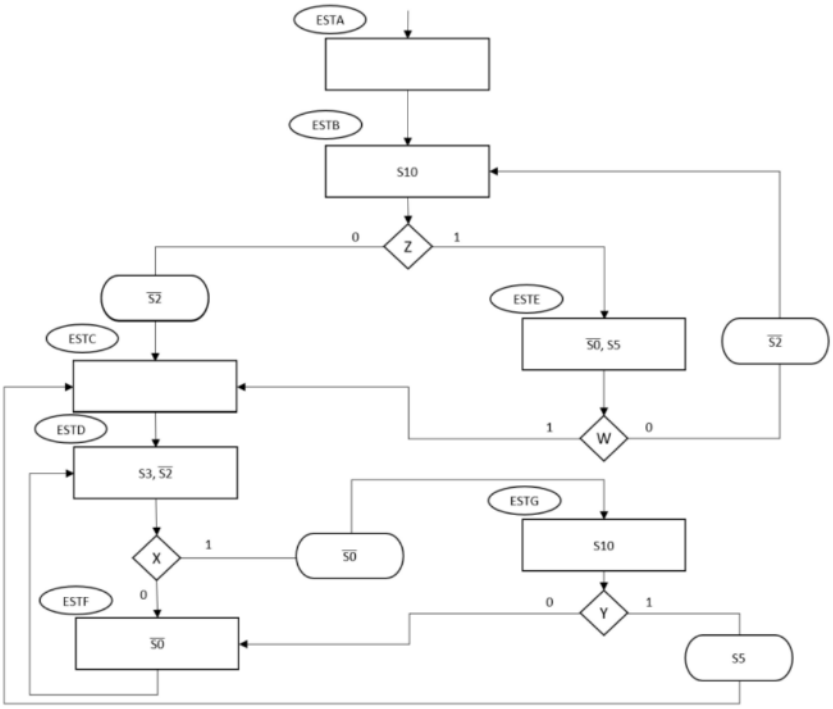


Figura 2: Carta ASM

Tabla 1: Valores binarios a estados

| Entradas | |
|---|---|
| ESTA | 000 |
| ESTB | 001 |
| ESTC | 010 |
| ESTD | 011 |
| ESTE | 100 |
| ESTF | 101 |
| ESTG | 110 |

Tabla 2: Valores binarios de las entradas

| Prueba | |
|---|---|
| x | 000 |
| y | 001 |
| z | 010 |
| w | 011 |
| aux | 100 |

Una vez especificados nuestros estados y entradas, resolvemos la tabla de verdad.

Tabla 3: Tabla de verdad obtenida

| Dirección de memoria | | | Contenido de la memoria | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Estado Presente | | | Prueba | | | Liga Falsa | | | Liga Verdadera | | | Salidas Falsas | | | | | Salidas Verdaderas | | | | |
| Q2 | Q1 | Q0 | | | | | | | | | | S5 | S3 | S2 | S1 | S0 | S5 | S3 | S2 | S1 | S0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |

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.numeric_std.all;
entity rom is
  generic(
    data_width : natural := 19;
    addr_length : natural := 3
    );
  port(
    clk: in std_logic;
    address: in std_logic_vector(addr_length-1 downto 0);
    data_out: out std_logic_vector(data_width-1 downto 0);
    mon_memoria: out std_logic_vector(data_width-1 downto 0)
    );
end rom;
architecture behavioral of rom is
  constant mem_size: natural := 7;
  type mem_type is array (mem_size-1 downto 0) of std_logic_vector(data_width-1 downto 0);
  constant mem : mem_type :=
    (
      0=>b"1000010010000000000",
      1=>b"0100101000001000010",
      2=>b"1000110110000000000",
      3=>b"0001011100100001000",
      4=>b"0110010101000010000",
      5=>b"1000110110000000000",
      6=>b"0011010100001010010",
      others=>b"1111111111111111111"
      );
begin
  rom : process(clk)
  begin
```

```
    if rising_edge(clk) then
      data_out <= mem(to_integer(unsigned(address)));
    end if;
  end process rom;
end architecture behavioral;
```

Código 1: `rom.vhd`

A continuación escribimos el bloque que realiza el direccionamiento entrada-estado.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity registro_entrada is port(
  clk, rst : in std_logic;
  estado_siguiente : in std_logic_vector(2 downto 0);
  data_out : out std_logic_vector(2 downto 0);
  mon_estado_siguiente: out std_logic_vector(2 downto 0)
  );
end registro_entrada;
architecture behavioral of registro_entrada is
  signal int_val : std_logic_vector(2 downto 0) := b"000";
begin
  process (clk, rst, estado_siguiente)
  begin
    if rst = '1' then
      int_val <= b"000";
    elsif rising_edge(clk) then
      mon_estado_siguiente <= estado_siguiente;
      int_val(2 downto 0) <= estado_siguiente;
    end if;
  end process;
  process(int_val)
  begin
    data_out <= int_val;
  end process;
end behavioral;
```

Código 2: `registro_entrada.vhd`

```
library IEEE;
use IEEE.std_logic_1164.all;
entity registro_salida is port(
  clk, rst : 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 int_val : std_logic_vector(4 downto 0) := b"00000";
begin
  process (clk, rst, data_in)
  begin
    if rst = '1' then
      int_val <= b"00000";
    elsif rising_edge(clk) then
      int_val(4 downto 0) <= data_in;
    end if;
  end process;
```

```
  process(int_val)
  begin
    salidas <= int_val;
  end process;
end behavioral;
```

Código 3: `registro_salida.vhd`

Ahora escribimos el separador de datos almacenados en memoria para así asignar el estado siguiente, valor de prueba, ligas falsas y las ligas verdaderas, donde estas dos últimas son la salida del bloque.

```
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 (18 downto 0);
  prueba:
  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);
  liga_falsa:  out std_logic_vector(2 downto 0);
  liga_verdadera:  out std_logic_vector(2 downto 0)
  );
end separador;
architecture behavioral of separador is
begin
  process (data_in)
  begin
    prueba <= data_in(18 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;
```

Código 4: `separador.vhd`

Para realizar la elección correcta de las salidas, teniendo en cuenta el valor de las entradas, se generó el selector de entrada para poder enviar un valor binario correspondiente a cada entrada; El selector de la liga también se generó para poder elegir entre una liga falsa o una liga real, y, por lo tanto, un selector de inicio para obtener un rendimiento adecuado de acuerdo con la liga que ha ocurrido

```
library IEEE;
use IEEE.std_logic_1164.all;
entity selector_entrada is
  port (
    valor_salida : out std_logic;
    prueba: in std_logic_vector(2 downto 0);
    w, x, y, z, aux : in std_logic
    );
end entity;
architecture arch_selector_entrada of selector_entrada is
```

```
  signal selInt : std_logic_vector (2 downto 0);
begin
  selInt <= prueba;
  valor_salida <= x when selInt = "000" else
                  y when selInt = "001" else
                  z when selInt = "010" else
                  w when selInt = "011" else
                  aux when selInt = "011" else
                  'U';
end architecture;
```

Código 5: `selector_entrada.vhd`

```
library IEEE;
use IEEE.std_logic_1164.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 entity;
architecture arch_selector_liga 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 architecture;
```

Código 6: `selector_liga.vhd`

```
library IEEE;
use IEEE.std_logic_1164.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 entity;
architecture arch_selector_salida 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 architecture;
```

Código 7: `selector_salida.vhd`

## 3.1. Diagrama

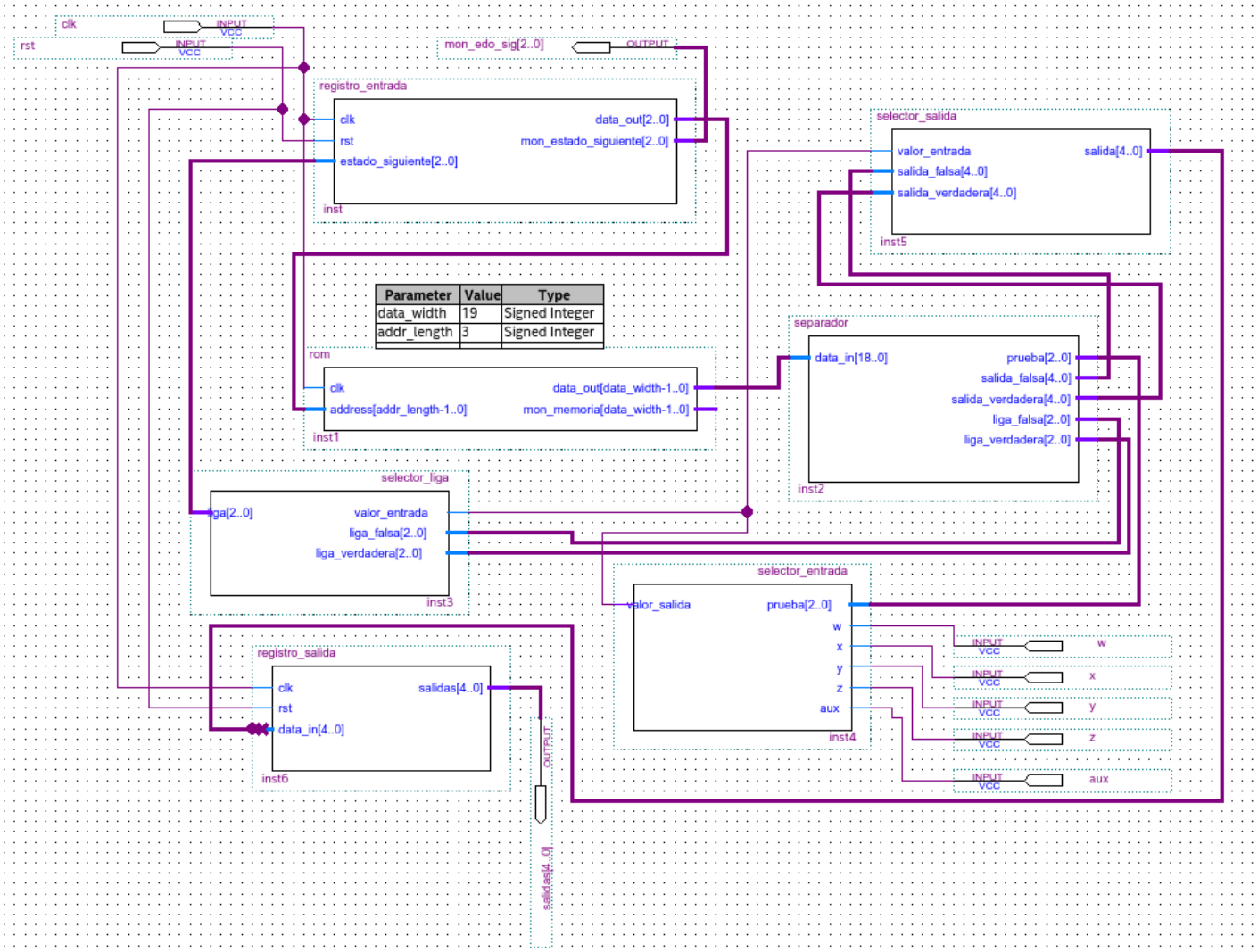


Figura 3: Diagrama de Bloques

## 3.2. Simulación

Una vez obtenido el diagrama esquemático proseguimos con la simulación. Para el siguiente ejemplo se considera que todas las entradas están en 0 menos W.

Sigamos la trayectoria en la carta ASM. Empezamos en el estado A y pasamos al B, en este primer caso, la salida corresponde a 00000. Posteriormente en el estado B se evalúa Z y pasa al estado C, este paso tiene como salida en alto a s1.

El siguiente paso es del estado C al D con salida 00000. Posteriormente, en el estado D se evalúa X, como está en 0 pasa al estado F y solo se activa s3.

Por último, del estado F regresa al estado D con ninguna salida en alto y como X no cambia su valor, se queda en un ciclo.

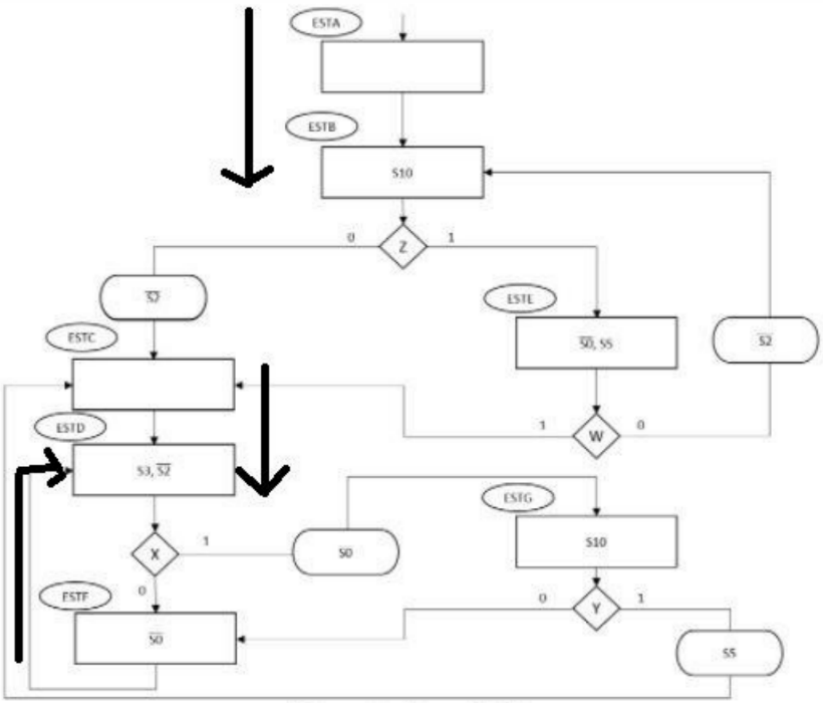


Figura 4: Ruta Carta ASM

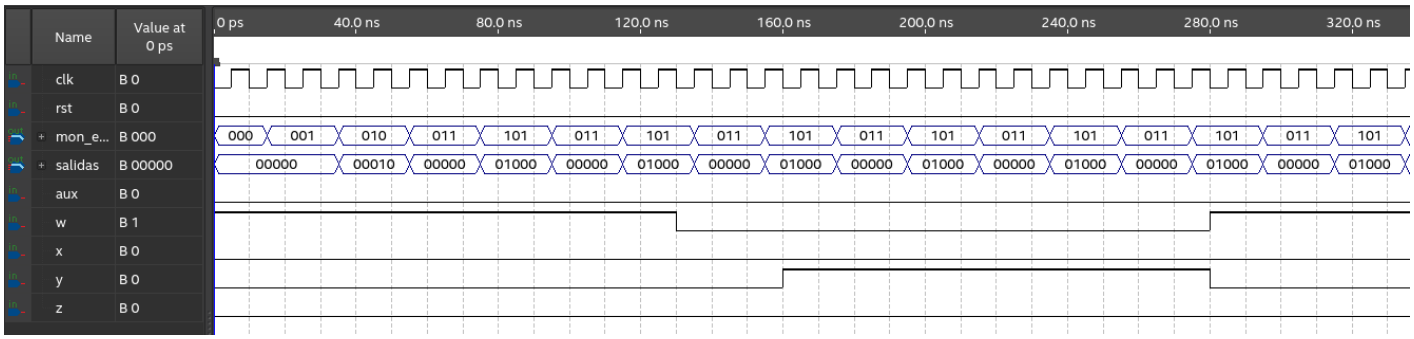


Figura 5: Simulación

## 4. Conclusiones

### Monsalvo Bolaños Melissa Monserrat

Con el desarrollo de esta práctica pudimos implementar la aplicación de una carta ASM por el método de direccionamiento de entrada estado. Pudimos observar que, a diferencia del método de direccionamiento por trayectoria, este método requiere de más elementos. Y

pudimos percatarnos de que a diferencia del método anterior aquí no podemos evaluar múltiples entradas, sin embargo observamos las bondades que nos ofrece, como la significativa reducción en la memoria.

**Romero Andrade Cristian**

La implementación de la carta ASM usando direccionamiento estrada-estado fue sencilla puesto que el uso de memorias "traduce" la tabla de verdad que se obtuvo al resolver la carta. Consecuentemente se logró desarrollar los bloques en código vhdl para lograr que concuerde la simulación con la carta y asu vez la tabla de verdad

**Conclusiones en equipo**

En la presente se logró implementar en VHDL la carta ASM usando direccionamiento entrada-estado, en mayor parte se utilizó el mismo diseño de la práctica anterior con unos cambios como por ejemplo la selección de liga y el de salidas para poder implementar dicho direccionamiento.

## Índice de tablas

## Índice de figuras

## Índice de Códigos