
Arquitectura de Computadoras

Practica No. 3

Construcción de Máquinas de Estados Usando Memorias

Objetivo: Familiarizar al alumno en el conocimiento construcción de máquinas de estados usando direccionamiento de memorias.

Desarrollo: Para cada uno de los siguientes apartados, realizar los diseños electrónicos que se piden.

1.- El direccionamiento por trayectoria guarda el estado siguiente y las salidas de cada estado de una carta ASM en una localidad de memoria. La porción de la memoria que indica el estado siguiente es llamada “la liga”, mientras que la porción que indica las salidas es llamada “la parte de las salidas”, como se muestra en la figura 1.

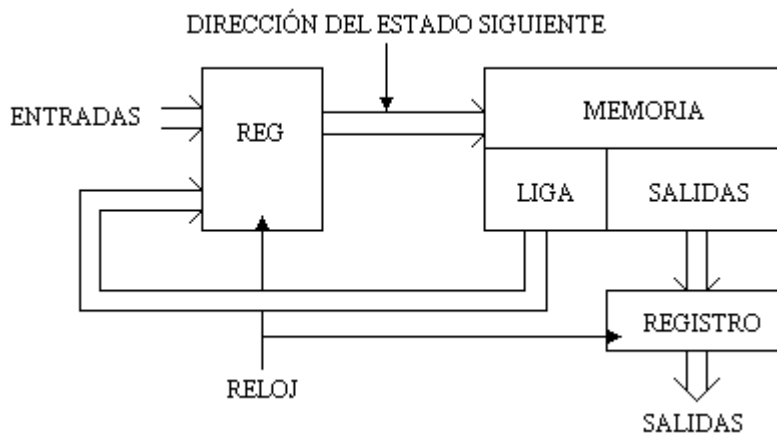


Figura 1. Direccionamiento por trayectoria.

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 1. La señal de reloj conectada a los registros es la que indica la velocidad de la máquina de estados. Usando un direccionamiento por trayectoria, construya una máquina de estados como la que se vio en la practica anterior. En el apéndice A se muestra el la versión en VHDL del contenido de la memoria del algoritmo de la máquina de estados.

* Se agradece el apoyo otorgado para el desarrollo de esta practica a DGAPA-UNAM PAPIME PE102213

APENDICE A

Instrumentación de la memoria usando VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity memoria is
  Port ( direccion : in STD_LOGIC_VECTOR (5 downto 0);
        output : out STD_LOGIC_VECTOR (7 downto 0));
end memoria;

architecture Behavioral of memoria is
begin
  process(direccion)

  begin
    -- La dirección se forma con la concatenación de la Liga y las entradas Si y Sd.
    -- El contenido de la memoria esta formado por los 4 bits mas significativos que son la LIGA y los 4 menos las salidas:
    -- adelante, atras, giro_der y giro_izq.
    --
    -- Localidades de memoria que representan el estado 0

    if(direccion="000000") then output <= "00001000";
    elsif(direccion="000001") then output <= "00010000";
    elsif(direccion="000010") then output <= "00110000";
    elsif(direccion="000011") then output <= "01010000";
    --
    -- Localidades de memoria que representan el estado 1

    elsif(direccion="000100") then output <= "00100100";
    elsif(direccion="000101") then output <= "00100100";
    elsif(direccion="000110") then output <= "00100100";
    elsif(direccion="000111") then output <= "00100100";

    --
    -- Localidades de memoria que representan el estado 2

    elsif(direccion="001000") then output <= "00000001";
    elsif(direccion="001001") then output <= "00000001";
    elsif(direccion="001010") then output <= "00000001";
    elsif(direccion="001011") then output <= "00000001";

    --
    -- Localidades de memoria que representan el estado 3

    elsif(direccion="001100") then output <= "01000100";
    elsif(direccion="001101") then output <= "01000100";
    elsif(direccion="001110") then output <= "01000100";
    elsif(direccion="001111") then output <= "01000100";
    --
    -- Localidades de memoria que representan el estado 4

    elsif(direccion="010000") then output <= "00000010";
    elsif(direccion="010001") then output <= "00000010";
    elsif(direccion="010010") then output <= "00000010";
```

```

elsif(direccion="010011") then output <= "00000010";
--
-- Localidades de memoria que representan el estado 5

elsif(direccion="010100") then output <= "01100100";
elsif(direccion="010101") then output <= "01100100";
elsif(direccion="010110") then output <= "01100100";
elsif(direccion="010111") then output <= "01100100";
--
-- Localidades de memoria que representan el estado 6

elsif(direccion="011000") then output <= "01110001";
elsif(direccion="011001") then output <= "01110001";
elsif(direccion="011010") then output <= "01110001";
elsif(direccion="011011") then output <= "01110001";

--

-- Localidades de memoria que representan el estado 7

elsif(direccion="011100") then output <= "10000001";
elsif(direccion="011101") then output <= "10000001";
elsif(direccion="011110") then output <= "10000001";
elsif(direccion="011111") then output <= "10000001";
--
-- Localidades de memoria que representan el estado 8

elsif(direccion="100000") then output <= "10011000";
elsif(direccion="100001") then output <= "10011000";
elsif(direccion="100010") then output <= "10011000";
elsif(direccion="100011") then output <= "10011000";
--

-- Localidades de memoria que representan el estado 9
elsif(direccion="100100") then output <= "10101000";
elsif(direccion="100101") then output <= "10101000";
elsif(direccion="100110") then output <= "10101000";
elsif(direccion="100111") then output <= "10101000";
--

-- Localidades de memoria que representan el estado 10

elsif(direccion="101000") then output <= "10110010";
elsif(direccion="101001") then output <= "10110010";
elsif(direccion="101010") then output <= "10110010";
elsif(direccion="101011") then output <= "10110010";

--

-- Localidades de memoria que representan el estado 11
elsif(direccion="101100") then output <= "00000010";
elsif(direccion="101101") then output <= "00000010";
elsif(direccion="101110") then output <= "00000010";
elsif(direccion="101111") then output <= "00000010";

end if;

end process;

end Behavioral;

```

APENDICE B

Registro de 6 bits en VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity register6 is
    Port ( clk : in STD_LOGIC;
          reset : in STD_LOGIC;

          D : in std_logic_vector (5 downto 0);
          Q : out std_logic_vector (5 downto 0));
end register6;

architecture Behavioral of register6 is

constant s0 :    std_logic_vector(5 downto 0) := B"000000";

begin

process (clk,reset)
    begin
        if reset='0' then Q <=s0;
        elsif rising_edge (clk) then
            Q <= D;
        end if;
    end process;
end Behavioral;
```

Conector de 4x2 a 6 bits en VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity conector4x2a6 is

    Port ( liga : in STD_LOGIC_VECTOR (3 downto 0);

          s : in STD_LOGIC_VECTOR (1 downto 0);

          output : out STD_LOGIC_VECTOR (5 downto 0));

end conector4x2a6;

architecture Behavioral of conector4x2a6 is

begin
```

```
process(liga,s)
begin
    output <= liga&s;
end process;
end Behavioral;
```

Conector de 8 a 4x4 bits en VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity conector8a4x4 is
    Port ( valor8 : in  STD_LOGIC_VECTOR (7 downto 0);
          liga : out STD_LOGIC_VECTOR (3 downto 0);
          salidas : out STD_LOGIC_VECTOR (3 downto 0));
end conector8a4x4;

architecture Behavioral of conector8a4x4 is
begin
    process(valor8)
begin
    liga <= valor8(7 downto 4);
    salidas <= valor8(3 downto 0);

end process;
end Behavioral;
```