Universidad Nacional Autónoma de M éxico

Facultad de Ingeniería

Arquitectura de Computadoras

Práctica 3

M.I. José Luis Cruz Mora

Integrantes:

Barriga Martínez Diego Alberto
Cabrera López Oscar Emilio
Oropeza Vilchis Luis Alberto

Grupo: 3

10 Abril del 2018

1. Direccionamiento Entrada-Estado

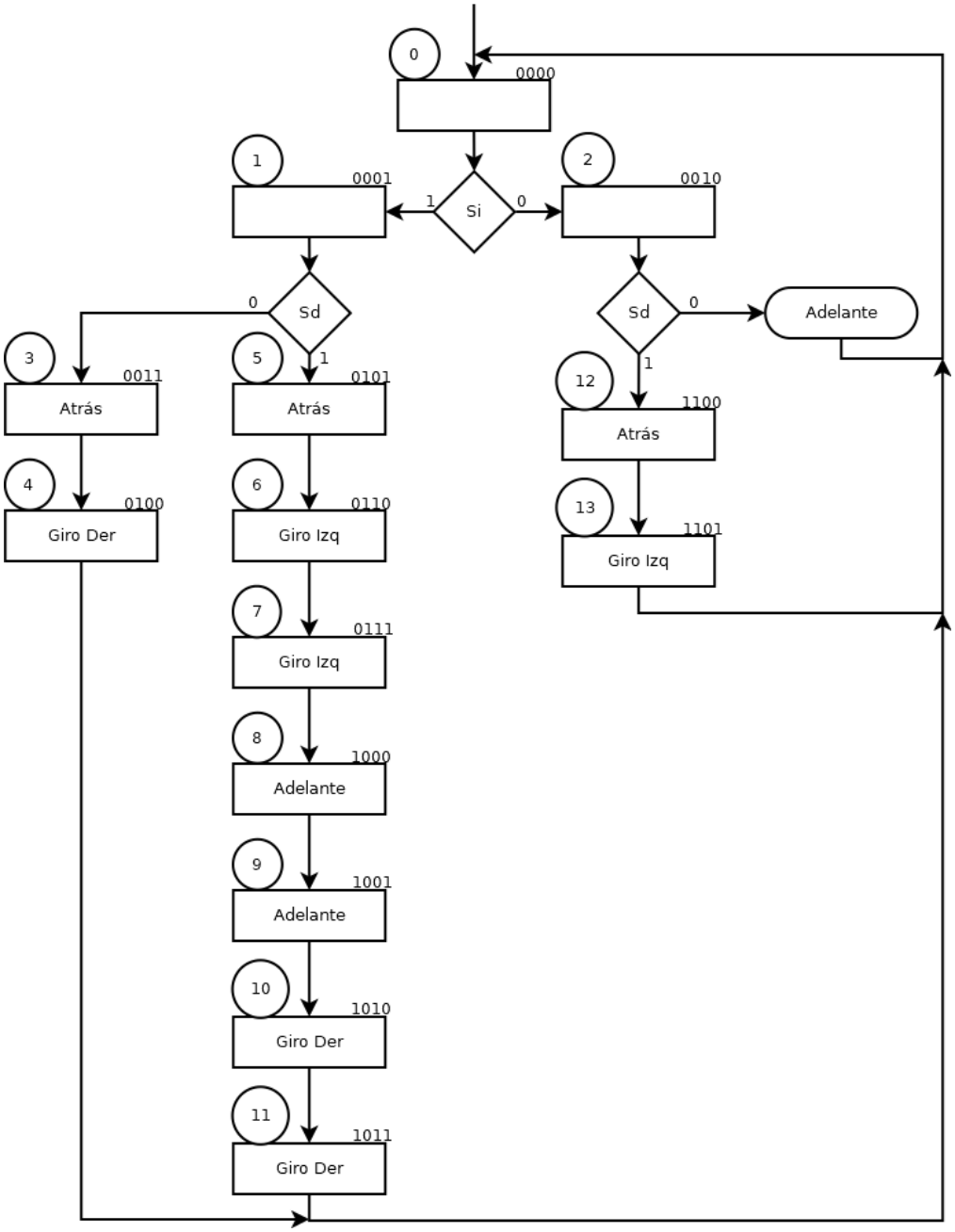

Figura 1: Carta ASM modificada

Número de estados: 14, por lo que se utilizan 4 bits para representarlos
Entradas: S_i y S_d
Salidas: Adelante (Ad), Atrás (At), Giro Izquierda (GI), Giro Derecha (GD)
Prueba:

Entrada	Código
S_i	0 0
S_d	0 1
$\uparrow Aux$	1 1

Contenido de la memoria:

Estado		Prueba	LV	LF	SV	SF
					Ad At GI GD	Ad At GI GD
0	0 0 0 0	0 0	0 0 0 1	0 0 1 0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 1	0 1 0 1	0 0 1 1	0 0 0 0	0 0 0 0
2	0 0 1 0	0 1	1 1 0 0	0 0 0 0	0 0 0 0	1 0 0 0
3	0 0 1 1	1 1	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0
4	0 1 0 0	1 1	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1
5	0 1 0 1	1 1	0 1 1 0	0 1 1 0	0 1 0 0	0 1 0 0
6	0 1 1 0	1 1	0 1 1 1	0 1 1 1	0 0 1 0	0 0 1 0
7	0 1 1 1	1 1	1 0 0 0	1 0 0 0	0 0 1 0	0 0 1 0
8	1 0 0 0	1 1	1 0 0 1	1 0 0 1	1 0 0 0	1 0 0 0
9	1 0 0 1	1 1	1 0 1 0	1 0 1 0	1 0 0 0	1 0 0 0
10	1 0 1 0	1 1	1 0 1 1	1 0 1 1	0 0 0 1	0 0 0 1
11	1 0 1 1	1 1	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1
12	1 1 0 0	1 1	1 1 0 1	1 1 0 1	0 1 0 0	0 1 0 0
13	1 1 0 1	1 1	0 0 0 0	0 0 0 0	0 0 1 0	0 0 1 0

Código:

Para la máquina de estados:

```
library ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;

entity entrada_estado is
        Port (
                        reloj: in std_logic;
                        entradas: in unsigned(1 downto 0); -- Bit menos sign
                        salidas: out unsigned(3 downto 0)
                        );
end entity;

architecture Behavioral of entrada_estado is
signal prueba: unsigned(1 downto 0);
signal sv, sf, lf, lv, salidas_mem, registro_mem, selector_liga, selector_sa
signal selector_entradas: std_logic;
signal mem_salida : unsigned(17 downto 0);

begin

-- Instancia de la memoria
memoria: entity work.memoria_ee
        port map(
                direccion => to_integer(registro_mem),
                salidas => mem_salida
        );

-- Multiplexor que elige la entrada
```

```
mux_entradas: selector_entradas <= entradas(0) when prueba = "00" else


-- Multiplexor que elige entre liga falsa y verdadera
mux_liga: selector_liga <= lf when selector_entradas = '0' else lv;
-- Multiplexor que elige entre salidas verdaderas o falsas
mux_salidas: selector_salidas <= sf when selector_entradas = '0' else sv;

-- Asignaciones de la salida de la memoria
-- Salidas falsa y verdadera
sf <= mem_salida(3 downto 0);
sv <= mem_salida(7 downto 4);
-- Liga falsa y verdadera
lf <= mem_salida(11 downto 8);
lv <= mem_salida(15 downto 12);
-- Prueba
prueba <= mem_salida(17 downto 16);

-- Registro que direcciona la memoria
reg_mem: process(reloj)
begin
        if rising_edge(reloj) then
                registro_mem <= selector_liga;
        end if;
end process;

-- Registro de las salidas
reg_salida: process(reloj)
begin
        if rising_edge(reloj) then
                salidas <= selector_salidas;
        end if;
end process;

end Behavioral;
```

Para la memoria:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity memoria_ee is

        generic
        (
                TAM_PALABRA : natural := 18;
                TAM_MEMORIA : natural := 14
        );
```

```
        port
        (
                direccion       : in natural range 0 to 2**TAM_MEMORIA - 1;
                salidas         : out unsigned((TAM_PALABRA -1) downto 0)
        );

end entity;

architecture rtl of memoria_ee is
        subtype palabra_t is unsigned((TAM_PALABRA-1) downto 0);
        type memoria_t is array(TAM_MEMORIA-1 downto 0) of palabra_t;

        signal mem : memoria_t := (
                0 =>  "0000010010000000000",
                1 =>  "0101010011000000000",
                2 =>  "0111000000000001000",
                3 =>  "1101000100010001000",
                4 =>  "1100000000000010001",
                5 =>  "1101100110010001000",
                6 =>  "1101110111001000010",
                7 =>  "1110001000001000010",
                8 =>  "1110011001100010000",
                9 =>  "1110101010100010000",
                10 => "1110111011000100001",
                11 => "1100000000000100001",
                12 => "1111011101010001000",
                13 => "1100000000001000010"
        );
begin
        salidas <= mem(direccion);
end rtl;
```

Simulación:

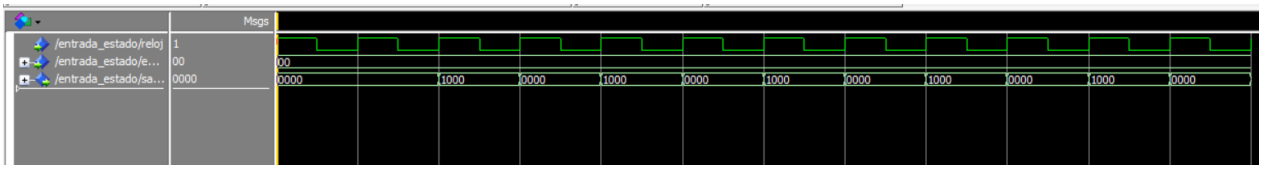

Figura 2: Entradas: 00

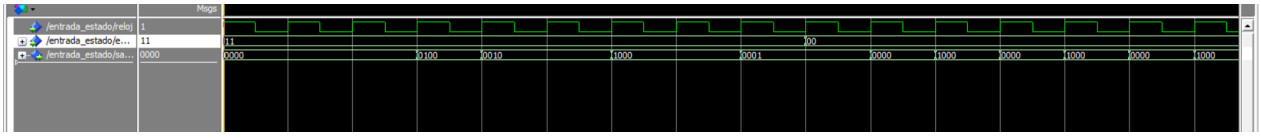

Figura 3: Entradas: 11

2. Direccionamiento Implícito

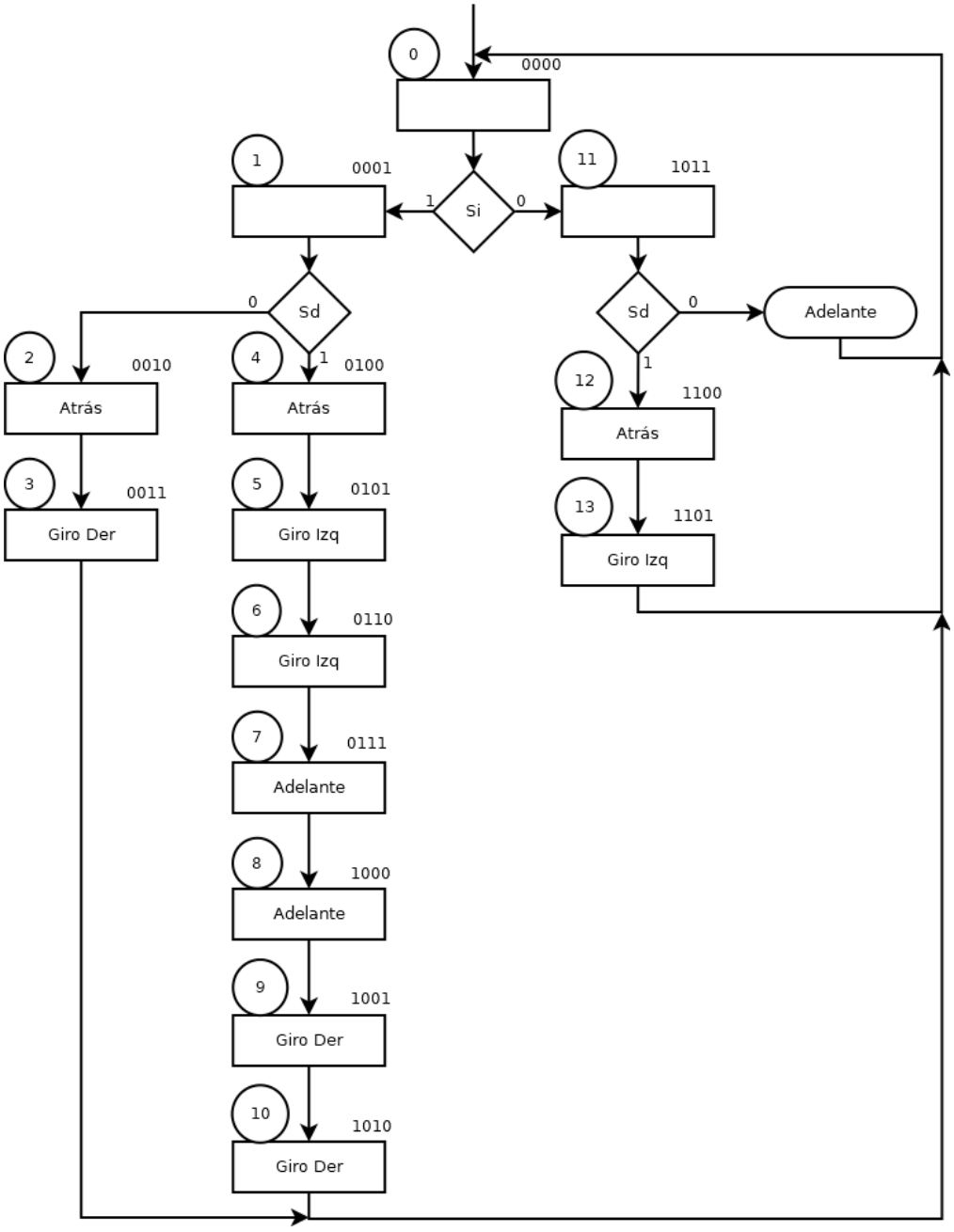

Figura 4: Carta ASM modificada

Número de estados: 14, por lo que se utilizan 4 bits para representarlos
Entradas: S_i y S_d
Salidas: Adelante (Ad), Atrás (At), Giro Izquierda (GI), Giro Derecha (GD)
Prueba:

Entrada	Código
S_i	0 0
S_d	0 1
$\uparrow Aux$	1 1

Carga:

Entrada	VF	Carga
0	0	1
0	1	0
1	0	0
1	1	1

Contenido de la memoria:

Estado		Prueba	VF	Liga	SV	SF
					Ad At GI GD	Ad At GI GD
0	0 0 0 0	0 0	0	1 0 1 1	0 0 0 0	0 0 0 0
1	0 0 0 1	0 1	1	0 1 0 0	0 0 0 0	0 0 0 0
2	0 0 1 0	1 1	1	0 0 1 1	0 1 0 0	0 1 0 0
3	0 0 1 1	1 1	1	0 0 0 0	0 0 1 0	0 0 0 1
4	0 1 0 0	1 1	1	0 1 0 1	0 1 0 0	0 1 0 0
5	0 1 0 1	1 1	1	0 1 1 0	0 0 1 0	0 0 1 0
6	0 1 1 0	1 1	1	0 1 1 1	0 0 1 0	0 0 1 0
7	0 1 1 1	1 1	1	1 0 0 0	1 0 0 0	1 0 0 0
8	1 0 0 0	1 1	1	1 0 0 1	1 0 0 0	1 0 0 0
9	1 0 0 1	1 1	1	1 0 1 0	0 0 0 1	0 0 0 1
10	1 0 1 0	1 1	1	0 0 0 0	0 0 0 1	0 0 0 1
11	1 0 1 1	0 1	0	0 0 0 0	0 0 0 0	1 0 0 0
12	1 1 0 0	1 1	1	1 1 0 1	0 1 0 0	0 1 0 0
13	1 1 0 1	1 1	1	0 0 0 0	0 0 1 0	0 0 1 0

Código:

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;

entity practica6 is
  port (
    clock     : in  std_logic;
    entradas  : in  unsigned(1 downto 0);
    salidas   : out unsigned(3 downto 0)
  );
end entity;

architecture arch of practica6 is
  signal carga_s, vf_s, ent_s: std_logic;
  signal prueba_s: unsigned(1 downto 0);
  signal liga_s, salv_s, salf_s, dir_s: unsigned(3 downto 0);
  signal sal_mem_s: unsigned(14 downto 0);
begin
  cont_e: entity work.contador port map(
    clock => clock,
```

```
    data  => liga_s,
    load  => carga_s,
    count => dir_s
  );

  rom_e: entity work.rom port map(
    cs       => '1',
    addr     => dir_s,
    data_out => sal_mem_s
  );

  reg_outv_p: process(clock) begin
    if rising_edge(clock) then
                        salv_s <= sal_mem_s(7 downto 4);
                end if;
        end process;

  reg_outf_p: process(clock) begin
    if rising_edge(clock) then
                        salf_s <= sal_mem_s(3 downto 0);
                end if;
        end process;

  prueba_s <= sal_mem_s(14 downto 13);
  vf_s     <= sal_mem_s(12);
  liga_s   <= sal_mem_s(11 downto 8);
  carga_s  <= ent_s xnor vf_S;

  with prueba_s select
    ent_s <=  entradas(0) when  "00",
              entradas(1) when  "01",
              '1'         when  "11",
              '1'         when  others;

  salidas <= salf_s when vf_s = '0' else salv_s;

end architecture;
```

Simulación:

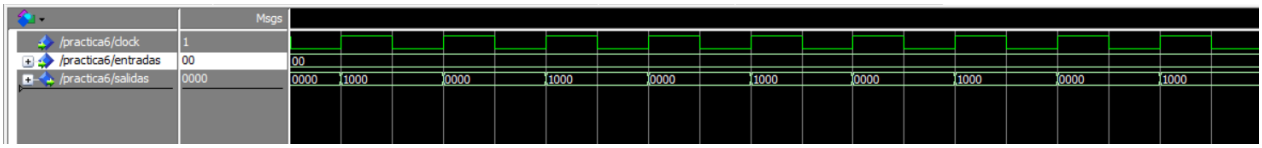

Figura 5: Entradas: 00

Figura 6: Entradas: 11

Conclusiones

Barriga Martínez Diego Alberto

La tercera práctica se ha vuelto más interesante ya que se sientan las bases con las que se trabajará a lo largo del semestre. El uso de direccionamientos es ampliamente utilizado porque reduce repetición y creación de tablas enormes de estados. Entrada-Estado y direccionamiento implícito son buenos acercamientos a lo que más adelante será el proyecto final que es mucho más cercano a un proyecto real que los ejercicios en clase. Con esto y los constantes ejercicios y tareas se reafirma el entendimiento de los direccionamientos.

Cabrera López Oscar Emilio

Con esta práctica aprendimos a implementar los modos de direccionamiento entrada-estado e implícito, lo que nos permite contruir máquinas más complejas aprovechando mejor la memoria. Esta práctica también nos permitió reforzar conocimientos de como implementar varios elementos VHDL como memorias, multiplexores, registros, etc. Todo esto nos permitirá realizar con mayor facilidad prácticas posteriores y el proyecto de la materia.

Oropeza Vilchis Luis Alberto

Hemos logrado implementar los dos modos de direccionamiento correctamente. Una vez que hemos entendido los conceptos nos ha sido sencillo implementar los diseños en VHDL. Sólo se debe tener cuidado al introducir los valores de la memoria ya que con alguno que esté mal el comportamiento podrá fallar en alguna parte, y podríamos no darnos cuenta, por lo que se debe verificar esta información una vez que se ha escrito, ya sea de manera manual o en algun archivo *.mif* que facilita la creación de memorias en Quartus.