Arquitectura de Computadoras
Practica No. 2
Introducción al lenguaje VHDL

Objetivo: Familiarizar al alumno en el conocimiento de los algoritmos de las máquinas de estados utilizando el lenguaje VHDL.

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

1. En la figura 1 se muestra un algoritmo o carta ASM.

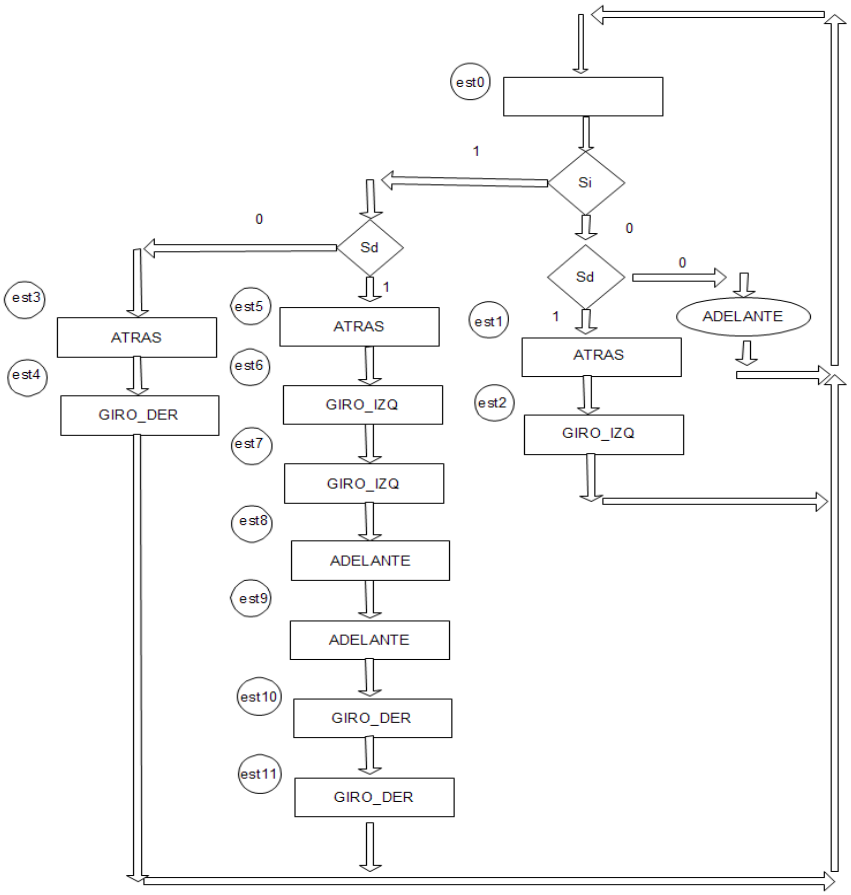

Figura 1. Algoritmo usando notación de carta ASM

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

A continuación se presenta este algoritmo usando el lenguaje de programación VHDL.

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

entity carta_asm_2 is

   Port ( RELOJ : in  STD_LOGIC;

                        RESET : in  STD_LOGIC;

                        S : in  std_logic_vector (1 downto 0); --DONDE EL BIT MAS SIGNIFICATIVO
ES SI Y EL MENOS EL SD
                        atras : out  STD_LOGIC;
                        adelante : out  STD_LOGIC;
                        giro_der : out  STD_LOGIC;
                        giro_izq : out  STD_LOGIC;
                        out_epresente :out std_logic_vector (3 downto 0));

end carta_asm_2;

architecture Behavioral of carta_asm_2 is

signal esiguiente :  std_logic_vector (3 downto 0) := B"0000";

constant s0 :       std_logic_vector(3 downto 0) := X"0";
constant s1 :       std_logic_vector(3 downto 0) := X"1";
constant s2 :       std_logic_vector(3 downto 0) := X"2";
constant s3 :       std_logic_vector(3 downto 0) := X"3";
constant s4 :       std_logic_vector(3 downto 0) := X"4";
constant s5 :       std_logic_vector(3 downto 0) := X"5";
constant s6 :       std_logic_vector(3 downto 0) := X"6";
constant s7 :       std_logic_vector(3 downto 0) := X"7";
constant s8 :       std_logic_vector(3 downto 0) := X"8";
constant s9 :       std_logic_vector(3 downto 0) := X"9";
constant s10 :      std_logic_vector(3 downto 0) := X"A";
constant s11 :      std_logic_vector(3 downto 0) := X"B";

begin

 process (RELOJ,reset,esiguiente, S)

  begin

        if reset='0' then esiguiente <=s0;

        elsif rising_edge (RELOJ)  then

                case esiguiente is

                        when s0 =>
```

```
                atras <= '0';
                giro_izq <= '0';
                giro_der <= '0';

        if  S =X"0"          then
                adelante  <= '1';
                esiguiente<= s0;

        elsif S =X"1"       then
                esiguiente<= s1;
                adelante  <= '0';

        elsif S =X"2"       then
                esiguiente<= s3;
                adelante  <= '0';

        elsif S =X"3"       then
                esiguiente<= s5;
                adelante  <= '0';

        end if;

when s1 =>
                adelante  <= '0';
                atras <= '1';
                giro_izq <= '0';
                giro_der <= '0';
                esiguiente<= s2;

when s2 =>
                adelante  <= '0';
                atras <= '0';
                giro_izq <= '1';
                giro_der <= '0';
                esiguiente<= s0;

when s3 =>
                adelante  <= '0';
                atras <= '1';
                giro_izq <= '0';
                giro_der <= '0';
                esiguiente<= s4;

when s4 =>
                adelante  <= '0';
                atras <= '0';
                giro_izq <= '0';
                giro_der <= '1';
                esiguiente<= s0;

when s5 =>
                adelante  <= '0';
                atras <= '1';
                giro_izq <= '0';
                giro_der <= '0';
                esiguiente<= s6;
```

```
                when s6 =>
                                adelante  <= '0';
                                atras <= '0';
                                giro_izq <= '1';
                                giro_der <= '0';
                                esiguiente<= s7;

                when s7 =>

                                adelante  <= '0';
                                atras <= '0';
                                giro_izq <= '1';
                                giro_der <= '0';
                                esiguiente<= s8;

                when s8 =>
                                adelante <= '1';
                                atras <= '0';
                                giro_izq <= '0';
                                giro_der <= '0';
                                esiguiente<= s9;

                when s9 =>
                                adelante <= '1';
                                atras <= '0';
                                giro_izq <= '0';
                                giro_der <= '0';
                                esiguiente<= s10;

                when s10 =>
                                adelante <= '0';
                                atras <= '0';
                                giro_izq <= '0';
                                giro_der <= '1';
                                esiguiente<= s11;

                when s11 =>
                                adelante <= '0';
                                atras <= '0';
                                giro_izq <= '0';
                                giro_der <= '1';
                                esiguiente<= s0;

                when others => null;

        end case;
        out_epresente <= esiguiente;

    end if;

 end process;
end Behavioral;
```

Haga un proyecto nuevo en el sistema de desarrollo Quartus en el cual se compile el código anterior. En Quartus haga un símbolo de esta máquina de estados. Programe la tarjeta para que muestre, usando cuatro leds, el estado presente, así mismo muestre también las salidas: adelante, atras, giro_izq y giro_der.

Las entradas SI, SD y reset introduzcalas usando los botones de la tarjeta. Conecte al reloj maestro al divisor que diseño en la practica anterior para poder visualizar mejor la operación de la máquina de estados, como se muestra en la figura 3.

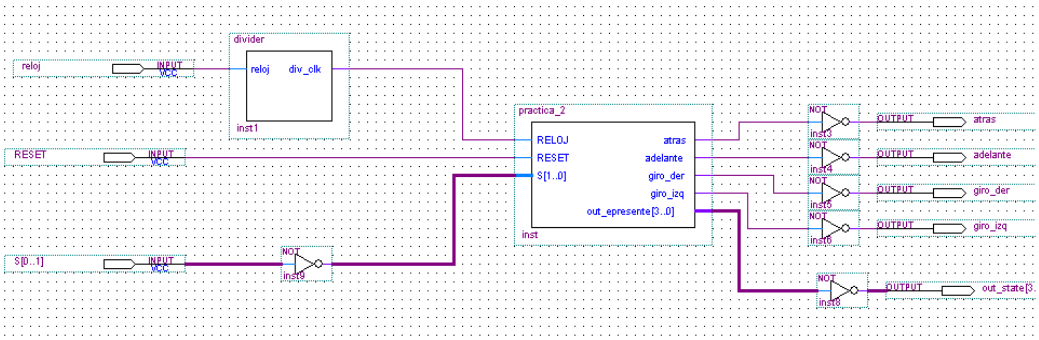

Figura 3. Diagrama de bloques de la máquina de estados

El siguiente código en VHDL es el de una máquina de estados, que a partir de una señal de reloj externa de alta frecuencia y la de un botón de entrada sirve para sensar cuando este es oprimido y liberado, enviando una señal que permite ver el desempeño de otra máquina de estados conectado a el paso a paso.

```
library IEEE;

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

entity sensa_boton is

   Port ( boton : in  STD_LOGIC;

                        clk : in  STD_LOGIC;
                        reloj : out  STD_LOGIC;
                        epresente:  buffer  STD_LOGIC);

end sensa_boton;

architecture Behavioral of sensa_boton is

signal esiguiente: STD_LOGIC;

 begin

  process (esiguiente,boton)
```

```
begin

        if rising_edge (clk)  then

         case esiguiente is

                when '0'    =>
                        reloj <= '0';
                        if  boton ='0' then
                                                esiguiente <= '0';
                                else
                                                esiguiente <= '1';
                        end if;

                when '1'      =>
                        if  boton ='1'        then
                                                esiguiente <= '1';
                                                reloj <= '0';
                                else
                                                esiguiente <= '0';
                                                reloj <= '1';
                                end if;

                when others   => null;

         end case;

        end if;

        epresente <= esiguiente;

 end process;

end Behavioral;
```

Compile este código e incluyalo en el diseño como se muestra en la figura 4.

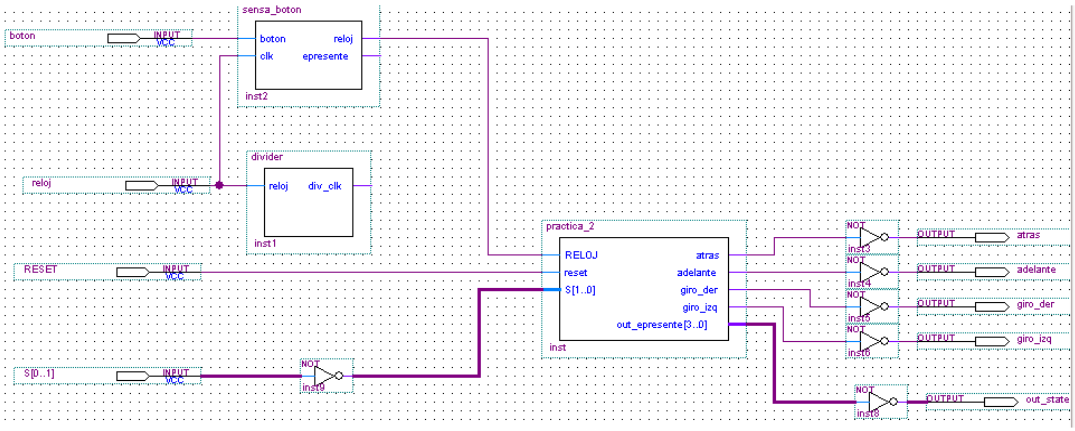

Figura 4. Incorporación del módulo que sensa cuando un botón es oprimido