Laboratory Assignments to Teach the Basics of Programmable Logic Applied to Mobile Robots

Jesus Savage, Francisco Dorantes, Alejandra Sanchez,
Ruben Anaya, Norma Chavez, Rodrigo Savage, Boris Escalante
Bio-Robotics Laboratory
School of Engineering FI, UNAM
Mexico City, Mexico
Email: robotssavage@gmail.com

Marco Morales
Robotics Laboratory
Department of Digital Systems, ITAM
Mexico City, Mexico
Email: marco.morales@itam.mx

***Abstract*—In this paper is presented a set a laboratory assignments that teach engineering students the basics of the design of state machines using programmable logic devices. The system developed at the end of these assignments is a small mobile robot that contains a behavior, based on a state machine, to avoid obstacles. These assignments have been tested with mechatronic, electrical and computer engineering students, and they considered that the material presented in them were sufficient to understand the basics of how to build state machines using programmable logic devices. The material is also useful for students interested on mobile robots, specifically in the area of subsumption theory using behaviors to control a mobile robot.**

***Keywords*-State Machines; Mobile Robots Behaviors; FPGs**

I. Introduction

There is a need of good and interesting laboratory assignments for students that teach them the basics of digital design with sequential logic, including the concept of state machines.

In recent years with the availability of affordable FPGAs systems, as the ones developed by Altera and Xilinx, the quality of the projects that can be done by students has been increased considerably.

Thus, in this paper is presented a set a laboratory assignments that teach engineering students the basics of the design of state machines using programmable logic devices. The system developed at the end of these assignments is a small mobile robot that contains a behavior, based on a state machine, to avoid obstacles. These assignments have been tested with mechatronic, electrical and computer engineering students and they considered that the material presented in them were sufficient to understand the basics of how to build state machines using programmable logic devices.The material is also useful for students interested of mobile robots, specifically in the area of subsumption theory using behaviors to control a mobile robot.

II. Laboratory Assignments

The assigments are taught during a course semester in laboratory sessions that last 2hrs, once a week, and each of them are completed in average in two weeks. The material used in these assignments consist of the Altera's TerAsic board, the MAX II Micro Kit [1], see figure 1, small dc motors, protoboards and power electronic modules.

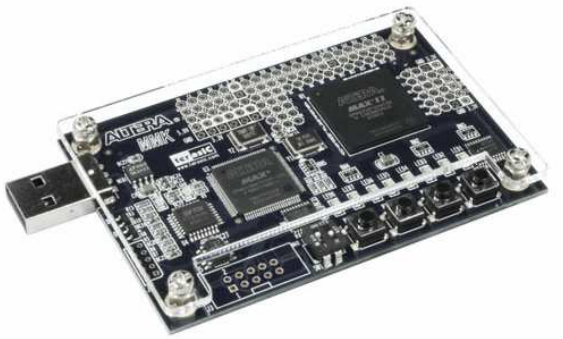

Figure 1. MAX II Micro Kit

A. Introductory Assignment

The first laboratory assignment is an introduction of the software, Quartus from Altera, and hardware tools to be used during the course. In this assignment the students are asked to design a simple system with a binary counter. They use the buttons included in the board to increment the counter and to reset it, the outputs of it are shown using the board's leds, Figure 1 shows the layout of this design.

Also in this assignment the students are introduced to the simulator, figure 2 shows a graphic of the design's simulation.

Due to the high speed of the board's clock, the students are asked also to design a clock divider to see visualy the perfomance of the counter using the leds of the board, see figure 4.

In the design of this divider they are introduced to VHDL coding, figure 5 shows its code.

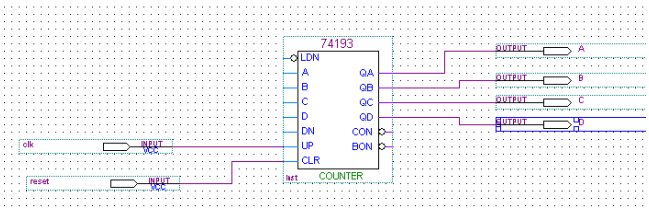

Figure 2. Assignment 1, implementation of a counter

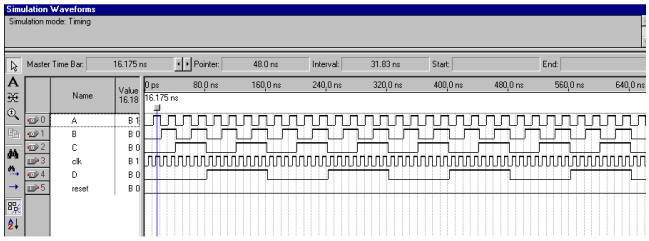

Figure 3. Simulator of the counter design

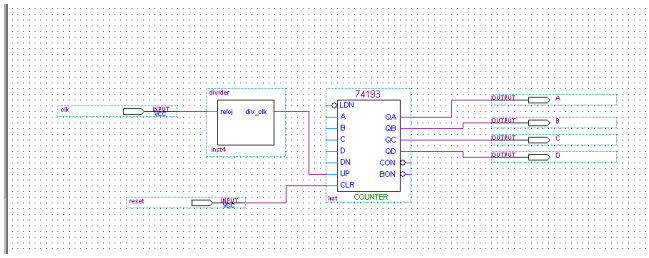

Figure 4. Clock divider included in the counter's design

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

entity divider is
   Port ( clk : in std_logic;
   div_clk : out std_logic);
end divider;

architecture Behavioral of divider is

begin

 process (clk)

 variable cnt: std_logic_vector (27 downto 0):

 begin
  if rising_edge (clk) then
   if cnt=X"4000000" then
      cnt:=X"0000000";
   else
     cnt:= cnt+1;
   end if;
  end if;

  div_clk <= cnt (25);

  end process;

end Behavioral;
```

Figure 5. Clock divider programmed using VHDL

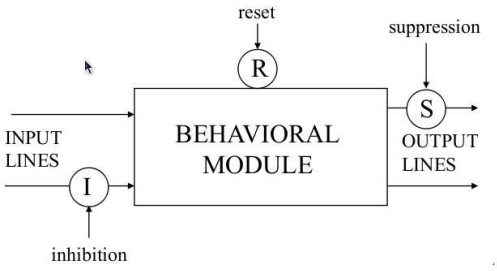

Figure 6. Augmented Finite State Machine

B. State Machine Assignment

In the second assignment the students are introduced to the implementation of state machines using VHDL [2]. Also they are introduced to the concept of simple mobile robot behaviors using state machines [3]. Ronald Brooks in the late 80's proposed a new robotics paradigm to control robots, in which their behaviors are build using augmented state machines (AFSM) shown in figure 6. In an AFSM some of its inputs and outputs can be substituted for other values externally by another AFSM.

Then by connecting together the AFSMs, each one containing a behavior, emergent intelligence can be achieved by a robot. In the subsumtion architecture each of the robot's behaviors, AFSMs, depending of their hierarchy in the system their inputs and outputs can the canceled by other AFSMs, figure 7 shows one example of this architecture.

In this laboratory session the students learn how to design a behavior for a robot that avoids obstacles. Figure 8 shows this behavior [4], the robot has two sensors in its left and right side, that allows it to detect obstacles. It has also two motors that move the robot forward and backward, if they move to the same direction, and they turn the robot in its on axis, if one motor is in one direction while the other goes to the contrary one.

Figure 9 shows the algorithm state machine (ASM) for the obstacle avoidance behavior.

In figure II-D it is shown the implementation of this ASM using VHDL. In this laboratory assignment the inputs of the state machine are introduced using the buttons of the Altera board and the outputs are visualized using the leds of it.

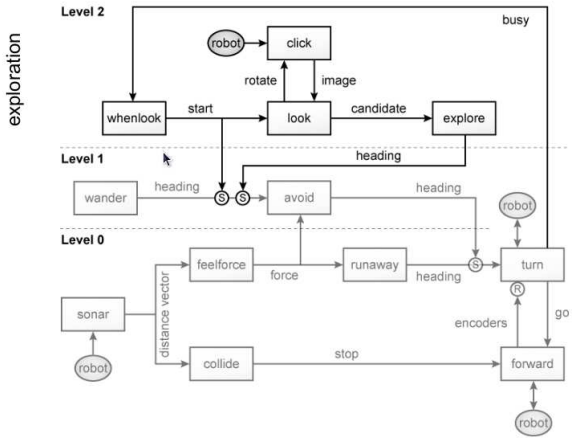

Figure 7. Brooks' subsumtion system to control a robot 6

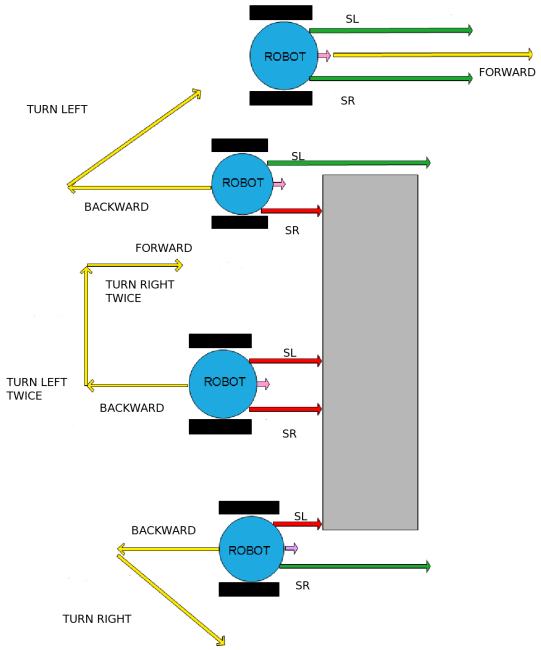

Figure 8. Robot avoiding an obstacle

C. Power Stage Assignment

In this laboratory assignment the students are required to build the power stage that controls the operation of the motors, figure 10 shows it.

This stage is connected to the FPGA board with the digital design shown in figure 11, which basically turns a motor on and off and set its direction.

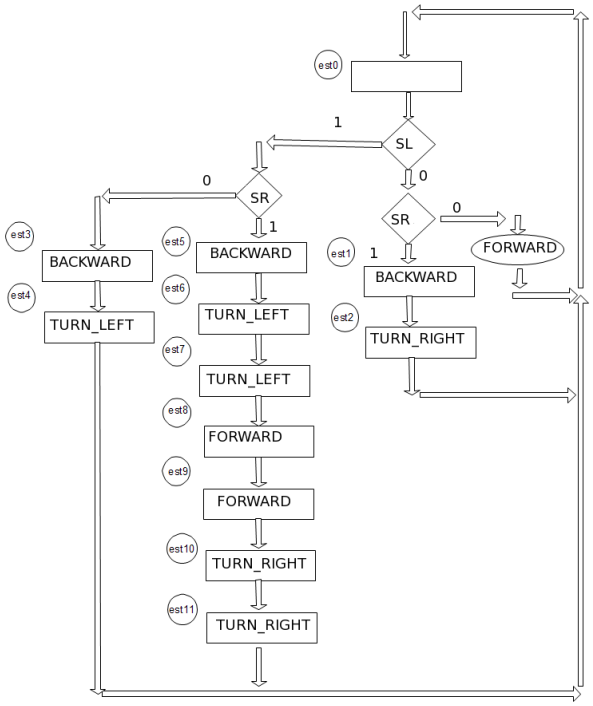

Figure 9. Algorithm State Machine for a mobile robot that avoids obstacles

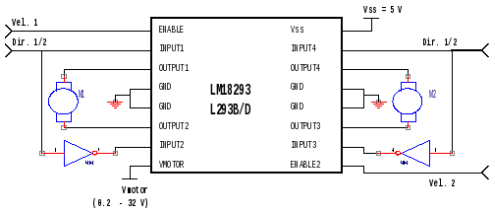

Figure 10. Power stage

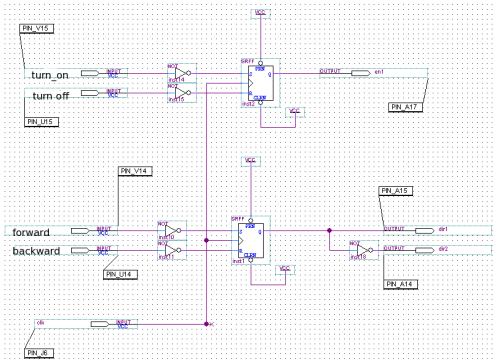

Figure 11. Digital design that turns the motor on and off

D. Construction of the robot assignments

In these laboratory assignments the students are asked to put together the state machine design together with the power stage, see figure 12.

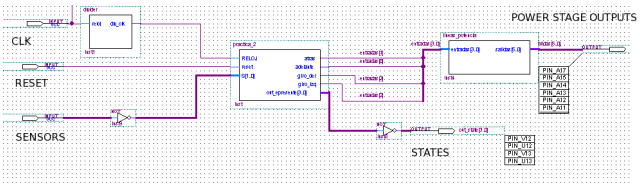

Figure 12. State Machine layout

Also they are asked to build the frame of the robot and put the motors and the sensors, see figure 13. The sensors that detect the obstacles are tactile or infrared ones. This assignments are distibuted in four weeks.

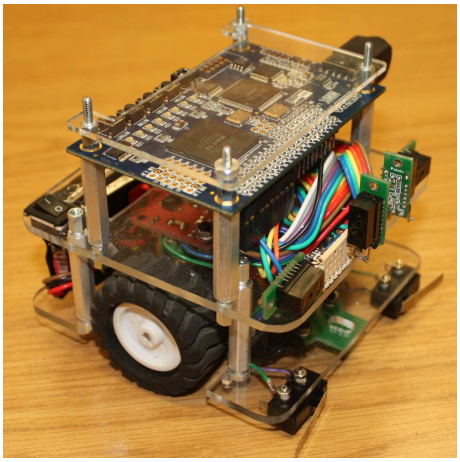

Figure 13. Robot with fpga

III. Conclusion

These laboratory assignments have been used by students of two of the leading engineering schools in Mexico at the ITAM and UNAM. The students that are using these assignments are from the electrical, mechatronics and computer engineering programs in both universities. The assignments catch the interest of the students by showing them an immediate feedback of their work. They are very satisfied not only because they learn how to design sequential systems using modern tools but also as a ”Ludic” experience, when they see the robot that they built really avoids obstacles. A video clip of one of the robots built with these assignments can be watched at the following address: http://www.youtube.com/user/BioroboticsUNAM

Also the laboratory assignments can be download from the following site: http://biorobotics.fi-p.unam.mx/

For future work more laboratory assignments will be developed in which more complex robots' behaviors will be programed.

Acknowledgment

This work was supported by PAPIME-DGAPA UNAM under Grant PE101107 and PAPIIT-DGAPA UNAM under Grant IN-107609

References

[1] Terasic Technologies, (2009). Max II Micro UserManual release v1.32.

[2] Sunggu Lee, (2005).Advanced Digital Logic: State Machine Design Using VHDL, Verilog, and Synthesis for FPGAS.

[3] Brooks, R. A. (1986). A robust layered control system for a mobile ribot. IEEE Journal of Robotics and Automation, RA.-2(1):14-23.

[4] Arkin, R.C., (1998). Behavior-Based Robotics- MIT Press, Cambridge, MA.

```
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); - SL and  SR
                backward : out  STD_LOGIC;
                forward : out  STD_LOGIC;
                turn_right : out  STD_LOGIC;
                turn_left : out  STD_LOGIC;
                out_present_state :out std_logic_vector (3 downto 0));
end carta_asm_2;

architecture Behavioral of carta_asm_2 is
signal next_state :  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,next_state, S)
  begin
      if reset='0' then next_state <=s0;
      elsif rising_edge (RELOJ)  then
            case next_state is
                when s0 =>
                        backward <= '0';
                        turn_left <= '0';
                        turn_right <= '0';
                    if  S =X"0"      then
                        forward  <= '1';
                        next_state<= s0;
                    elsif S =X"1"   then
                        next_state<= s1;
                        forward  <= '0';
                    elsif S =X"2"   then
                        next_state<= s3;
                        forward  <= '0';
                    elsif S =X"3"   then
                        next_state<= s5;
                        forward  <= '0';
                    end if;
                when s1 =>
                        forward  <= '0';
                        backward <= '1';
                        turn_left <= '0';
                        turn_right <= '0';
                        next_state<= s2;
                when s2 =>
                        forward  <= '0';
                        backward <= '0';
                        turn_left <= '1';
                        turn_right <= '0';
                        next_state<= s0;
                when s3 =>
                        forward  <= '0';
                        backward <= '1';
                        turn_left <= '0';
                        turn_right <= '0';
                        next_state<= s4;
                when s4 =>
                        forward  <= '0';
                        backward <= '0';
                        turn_left <= '0';
                        turn_right <= '1';
                        next_state<= s0;
                when s5 =>
                        forward  <= '0';
                        backward <= '1';
                        turn_left <= '0';
                        turn_right <= '0';
                        next_state<= s6;
                when s6 =>
                        forward  <= '0';
                        backward <= '0';
                        turn_left <= '1';
                        turn_right <= '0';
                        next_state<= s7;
                when s7 =>
                        forward  <= '0';
                        backward <= '0';
                        turn_left <= '1';
                        turn_right <= '0';
                        next_state<= s8;
                when s8 =>
                        forward <= '1';
                        backward <= '0';
                        turn_left <= '0';
                        turn_right <= '0';
                        next_state<= s9;
                when s9 =>
                        forward <= '1';
                        backward <= '0';
                        turn_left <= '0';
                        turn_right <= '0';
                        next_state<= s10;
                when s10 =>
                        forward <= '0';
                        backward <= '0';
                        turn_left <= '0';
                        turn_right <= '1';
                        next_state<= s11;
                when s11 =>
                        forward <= '0';
                        backward <= '0';

                        turn_left <= '0';
                        turn_right <= '1';
                        next_state<= s0;
                when others => null;
            end case;
            out_present_state <= next_state;
      end if;

end process;
```

Figure 14. VHDL implementation of the ASM