USC Viterbi
School of Engineering

FPGA Embedded Systems Final Project Report

University of Southern California

Team Members: Alexander Valente, Kivilcim Cumbul

Date of Report: 04/26/2019

3710 McClintock Ave, RTH Suite 403 · Los Angeles, CA 90089-1456
www.cs.usc.edu · Email: cumbul@usc.edu · valentea@usc.edu

Abstract

This paper describes our methods and thought process behind a Nexys3 FPGA board powered line following automated robot. We use a One-Hot State Machine to drive the logic of our hardware. We use PWM and an H-Bridge to act as a controller for the motors. Using light sensors on the bottom to detect the line (white tape) and send signals to the state machine.

3710 McClintock Ave, RTH Suite 403 · Los Angeles, CA 90089-1456
www.cs.usc.edu · Email: cumbul@usc.edu · valentea@usc.edu

Table of Contents

Introduction and Background

Our team was made up from two members, Alexander Valente and Kivilcim Cumbul. While Kivilcim Cumbul pursues a Computer Engineering and Computer Science degree, Alex double majors in both Electrical Engineering and Computer Science. We both have certain level of experiences in designing and building robots, and we combined our robotics knowledge with some of the digital design labs such as Number Lock, Picoblaze Keyboard, and EE354 homeworks such as RTL Design homeworks to build a line following robot.

The Design

1- Line Follower Description

Figure 1

Line following robot is an autonomous system which follows black line in white surface or white line in black surface. One of the most important application of a line follower is to supply materials in industry (Figure 1). To create a line follower, we used 4 different hardware components (Figure 2). We used Nexys 3 as the brain of the robot for decision making, 4 DC motors to move robot forward, left or right, and an H-bridge to supply 9 volts to DC motors and control the direction and speed of the motor via sending different percentages of duty cycles(Pulse Width Modulation). Lastly, we used digital light sensors to find the position of the line with respect to the robot.

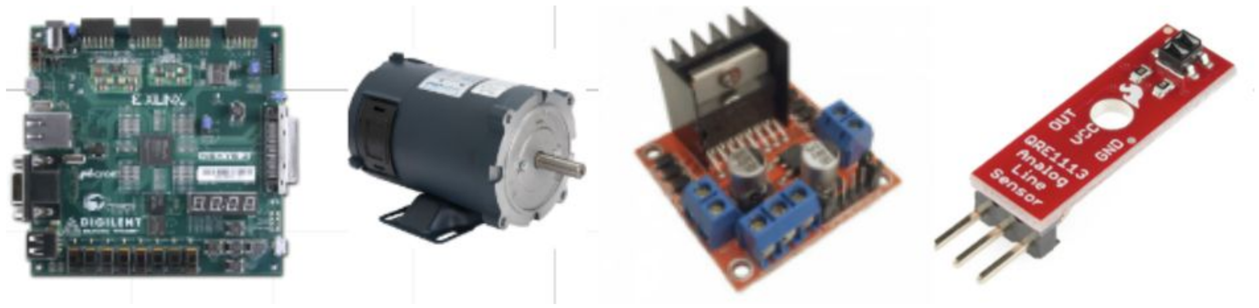
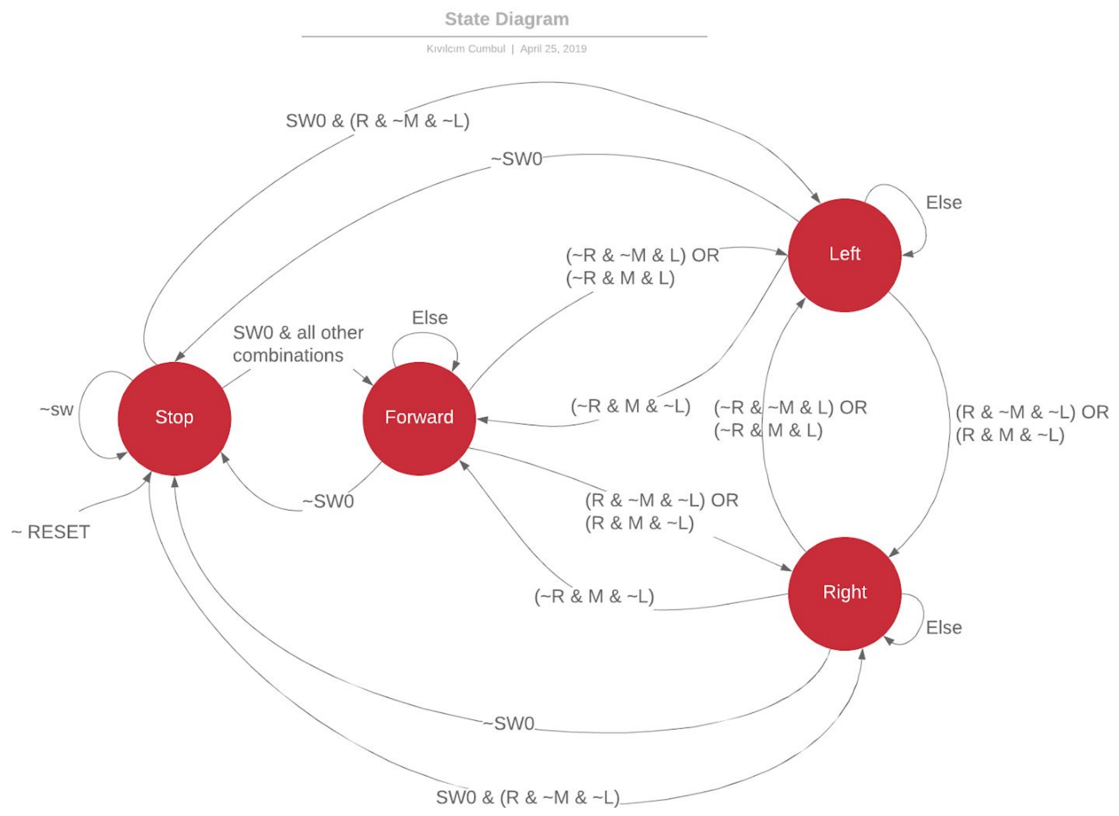


Figure 2: Nexys3, DC motor, H bridge motor controller and light sensors.

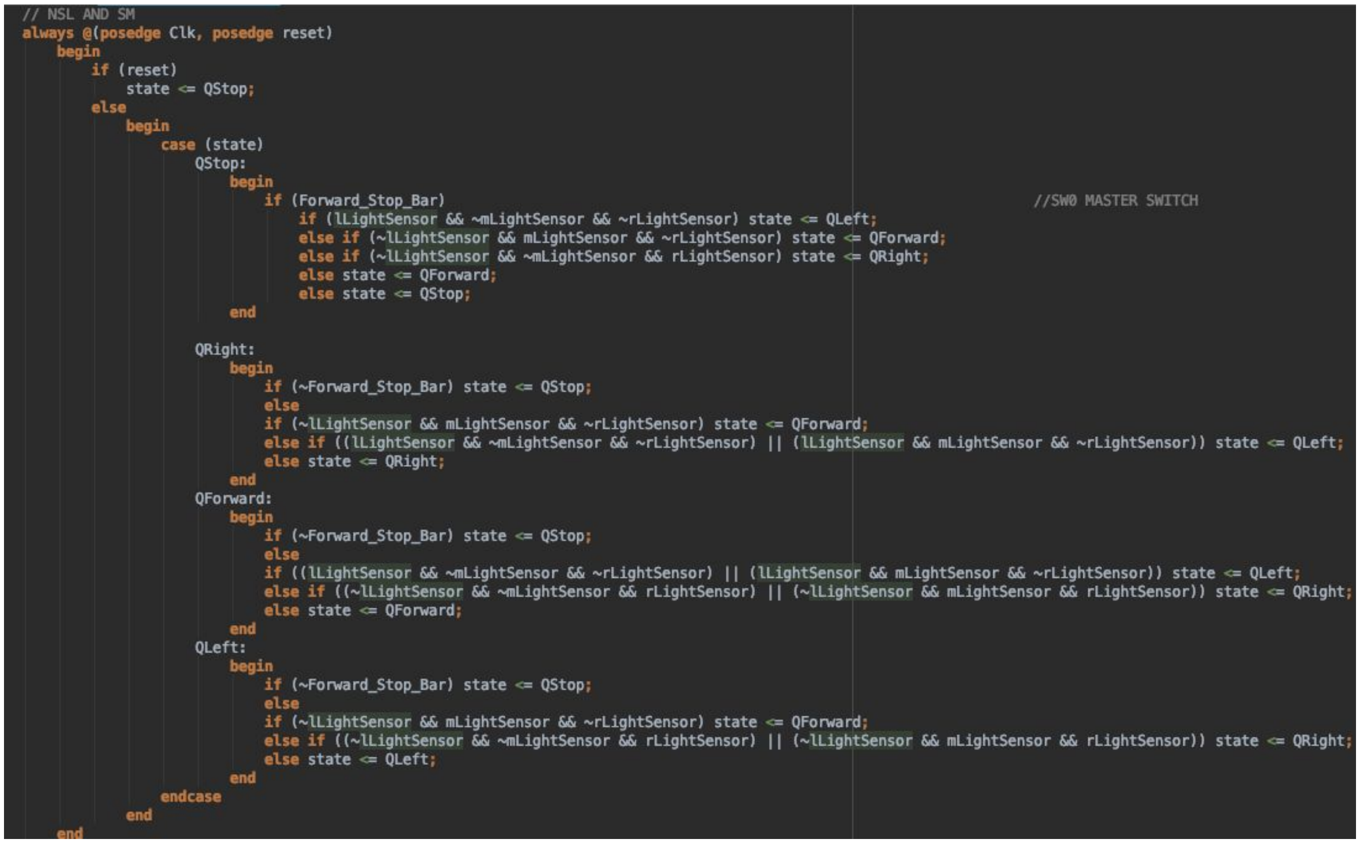
2- Explanation of the State Machine

After connecting all hardware components and assembling the robot, we started to create a state machine diagram in order to provide robot with decision making algorithm. The state machine has 4 states. Stop, Forward, Left and Right.

We used the Nexys3 FPGA as the controlling unit in our robot. We used the S0 Switch physically on the board do act as a master switch that changes the state of the robot between Stop (where it stays stationary regardless of anything else) and the movement case switching. Depending on the inputs of our three light sensors we either move the robot in place rotating left or right, or we move the robot forward. If we ever reached a place where there was no tape the robot continues in the same state that it was last in.



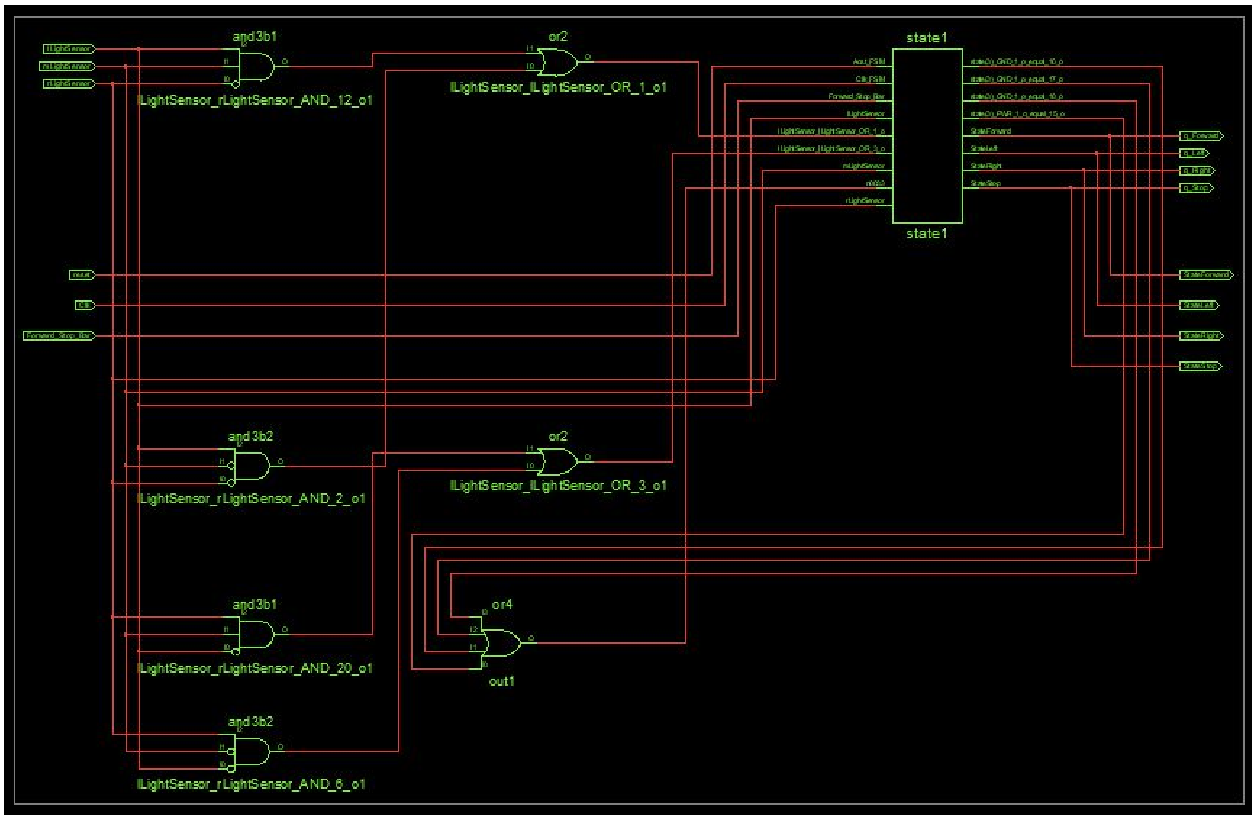
In the verilog code, we used the same technique we used in the RTL design homework. Once we understood the main logic to power the state machine we went to coding it. We started with just having the master switch change the state from Stop State, to the case switching. In the case switching we change the state depending on the inputs from the sensors. A challenge we faced was then if the path of the tape was too dramatic, both the middle and one of the side sensors would be active-high and that originally would break our state machine. So to fix this we added statements that if any two sensors were active then we would change state to the direction of the corresponding outer led.



```verilog
// NSL AND SM
always @(posedge Clk, posedge reset)
    begin
        if (reset)
            state <= QStop;
        else
            begin
                case (state)
                    QStop:
                        begin
                            if (Forward_Stop_Bar)                                                   //SW0 MASTER SWITCH
                                if (lLightSensor && ~mLightSensor && ~rLightSensor) state <= QLeft;
                                else if (~lLightSensor && mLightSensor && ~rLightSensor) state <= QForward;
                                else if (~lLightSensor && ~mLightSensor && rLightSensor) state <= QRight;
                                else state <= QForward;
                                else state <= QStop;
                        end

                    QRight:
                        begin
                            if (~Forward_Stop_Bar) state <= QStop;
                            else
                            if (~lLightSensor && mLightSensor && ~rLightSensor) state <= QForward;
                            else if ((lLightSensor && ~mLightSensor && ~rLightSensor) || (lLightSensor && mLightSensor && ~rLightSensor)) state <= QLeft;
                            else state <= QRight;
                        end
                    QForward:
                        begin
                            if (~Forward_Stop_Bar) state <= QStop;
                            else
                            if ((lLightSensor && ~mLightSensor && ~rLightSensor) || (lLightSensor && mLightSensor && ~rLightSensor)) state <= QLeft;
                            else if ((~lLightSensor && ~mLightSensor && rLightSensor) || (~lLightSensor && mLightSensor && rLightSensor)) state <= QRight;
                            else state <= QForward;
                        end
                    QLeft:
                        begin
                            if (~Forward_Stop_Bar) state <= QStop;
                            else
                            if (~lLightSensor && mLightSensor && ~rLightSensor) state <= QForward;
                            else if ((~lLightSensor && ~mLightSensor && rLightSensor) || (~lLightSensor && mLightSensor && rLightSensor)) state <= QRight;
                            else state <= QLeft;
                        end
                endcase
            end
    end
```

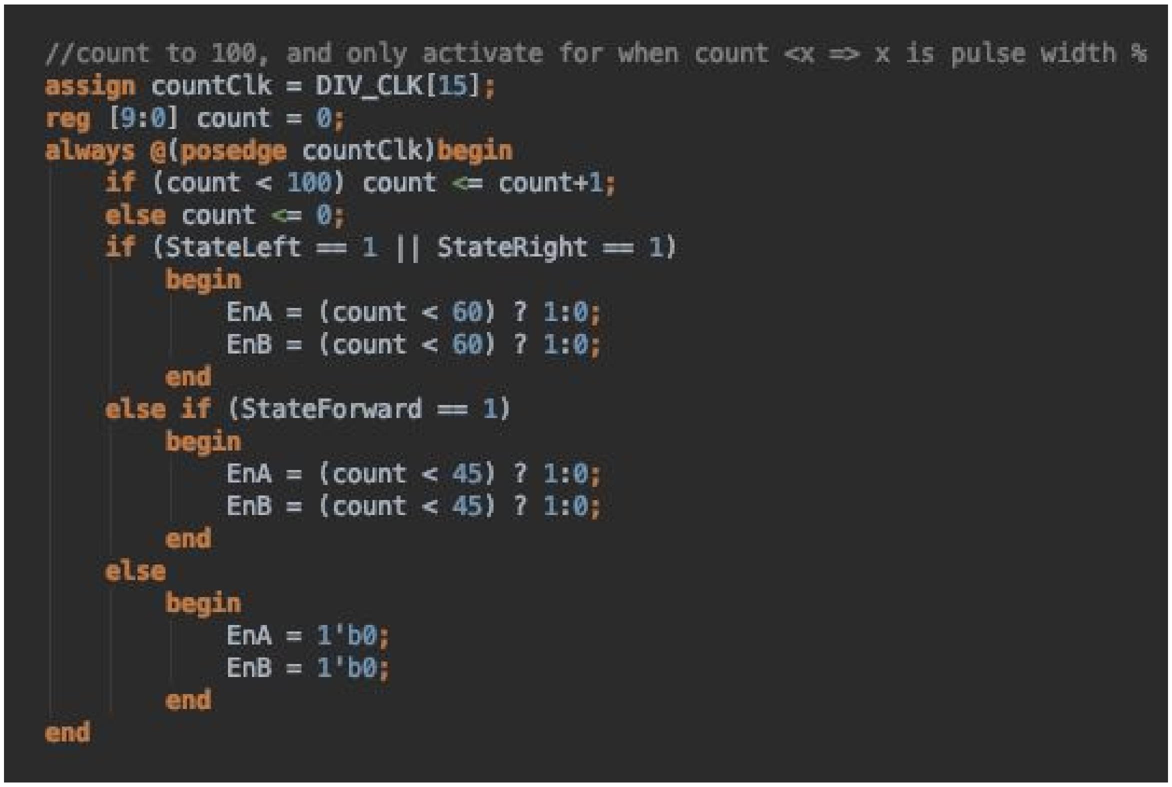
3- Block Level Description


and3b1
or2
state1
ILightSensor_rLightSensor_AND_12_o1
ILightSensor_ILightSensor_OR_1_o1
and3b2
or2
ILightSensor_rLightSensor_AND_2_o1
ILightSensor_ILightSensor_OR_3_o1
and3b1
or4
out1
ILightSensor_rLightSensor_AND_20_o1
and3b2
ILightSensor_rLightSensor_AND_6_o1

In the sm file, we create get left sensor, middle sensor, right sensor inputs, and we decide which state to go depending on the input choices in each clock. First, we were using divclk(25), and the robot was not responding fast enough, than we changed our clock time into system clock. In the figure, generation of each State is shown, but after getting state outputs, in top file we created a OFL. In the OFL we also implemented PWM to change the duty cycle of each output and tune the robot in order to follow line smoothly.

4- Explanation of PWM

When first testing the motor control, we quickly realised that the speed of the robot was much too fast and that when turning to stay on the path of the tape we would get jittery movement form the robot. We decided the best way to mitigate this movement would be with Pulse width modulation. Pulse width modulation means changing the output voltage, namely the motor speed, with changing the duty cycle of the square waveform. We implemented PWM as follows



```
//count to 100, and only activate for when count <x => x is pulse width %
assign countClk = DIV_CLK[15];
reg [9:0] count = 0;
always @(posedge countClk)begin
    if (count < 100) count <= count+1;
    else count <= 0;
    if (StateLeft == 1 || StateRight == 1)
        begin
            EnA = (count < 60) ? 1:0;
            EnB = (count < 60) ? 1:0;
        end
    else if (StateForward == 1)
        begin
            EnA = (count < 45) ? 1:0;
            EnB = (count < 45) ? 1:0;
        end
    else
        begin
            EnA = 1'b0;
            EnB = 1'b0;
        end
end
```

We assign a clock at a slower speed than our universal state machine clock. We then use that clock to increment the count variable to 100. We used 'if' statements to state that the enable pins are active only when count is less than the desired duty cycle. The main challenge we faced when implementing this was the fact that our pulse width is completely based off the speed at which we increment the counter, and that is based off the clock we use. If we set that clock too fast the pulse width would be extremely fast and the motors would not be receiving enough power to even move. If we made the pulse width too slow, the robot would visibly move at full speed for a small portion of time and then would completely stop for the remainder of that pulse width. So we had to calculate the proper pulse width as determined by the clock and then also what percentage we wanted the duty cycle to be at. Once we were done tuning these values the robot movement along the line was smooth.

Test Methodology

We have used different test methodologies for both hardware and software part of the line follower robot. For the hardware portion of the testing, we built numerous circuits with leds, resistors, buttons and potentiometers. We checked motors, motor controllers, light sensors, and FPGA input output ports with different circuits to make sure they are working. For software (Verilog) testing, we have created 5 different projects in Xilinx ISE. First, after creating the state machine, we tested it with buttons, leds and switches by giving 3 different inputs. Then, we tested motor controller testing with buttons and switches by sending outputs from FPGA to the motor controller. Then, after implementing Pulse Width Modulation, we tested its functionality with leds, and checked if they get dimmer. Lastly, we checked light sensors input readings we get from FPGA with turning on and off leds. After testing each hardware and software component, we tested every module in robot, and tuned the line following robot with trying different clock dividing and PWM values.

Conclusion and Future Work

To conclude, even though we had several serious challenges while implementing the line follower robot, we successfully combined what we learned in Digital Design course with our previous robotics knowledge, and developed a deeper knowledge in Verilog coding. We were controlling line follower robot with light sensor inputs, but on top of that, FPGA buttons were also used to control the robot. As a future work, with adding a small piece of code, we can control robot from a remote controller. Also, with a similar hardware which includes ultrasound sensor and after adding small changes to the state machine, we can create a wall following robot. Further changes in the state machine would result in obstacle avoiding robot.