4/9/2017

ELEX 7660: Digital System Design

Self-Driving Car

Kostiantyn Yushchak & Davneet Singh

Table of Contents

Kostiantyn Yushchak & Davneet Singh 0

1 Acknowledgments 3

2 Overview 3

2.1 Goals and Motivation 3

2.2 Background 3

2.3 Hardware and Sensors Descriptions 3

3 Outcome 4

3.1 Final Result 5

3.2 Future Improvements 6

4 Technical Details 6

4.1 Design Block Diagrams 6

4.1.1 Block Diagram of the overall design 6

4.1.2 Steering flowchart (implemented in `direction.sv`) 7

4.1.3 Flow Chart for the QTR-8RC Sensor (implemented in `ledsarray.sv`) 9

4.1.4 PWM Motor Control (implemented in `motorcontrol.sv`) 10

4.2 Circuit diagrams 11

4.2.1 5V the step-down regulator 11

4.2.2 Motor Control IC: L293D 11

4.2.3 Sensor Array 12

4.3 Verilog Code 12

4.3.1 Steering control 12

4.3.2 Receiving and Interpreting Data from the QTR-8RC Sensor 16

4.3.3 PWM to control the motor 19

4.3.4 Top Level Module for this project 23

5 References 24

Table of Figures

Figure 1: The car (front view) 4
Figure 2: The car (side view) 4
Figure 3: The car on the track 5
Figure 4: Overall Design Block Diagram 6
Figure 5: Sensor Array Flow Chart 8
Figure 6: QTR-8RC Reflectance Sensor Operation Flowchart 9
Figure 7: PWM flow chart 10
Figure 8: 5V, 1A Step-Down Voltage Regulator D24V10Fx 11
Figure 9: L293D H-Bridge Circuit 11
Figure 10: QTR - 8RC Sensor Array 12

1 Acknowledgments

We would like to thank our instructor, Ed Casas, for mentoring us in this project and providing us with the parts required. We would also like to thank our friends and classmates for every bit of help they provided.

2 Overview

2.1 Goals and Motivation

Our objective for this project was to build an autonomous car which follows a track using a reflectance sensor array [1].

The self-driving car industry is rapidly developing, and it was our main motivation behind this project. We wanted to recreate the product of this exciting industry on a small scale. It gave us an insight into how much work, knowledge, and variables goes into building self-driving cars.

2.2 Background

In addition, we chose this project because it encompassed applications of theory that we learned during lectures. We used an Altera DE0 Nano board to interface the sensors, and control the DC motors using a H-Bridge.

2.3 Hardware and Sensors Descriptions

QTR-8RC Reflectance Sensor Array was used in the project to track the location of the black line and ultimately steer the car in the appropriate direction. This sensor module has 8 IR LED/phototransistors, and each sensor provides a separate digital I/O-measurable output. Each IR LED has an emitter and a receiver which are controlled by a MOSFET. Digital I/O line can drive the output line high, and then measure the time for the output voltage to decay. The decay is slow with no reflectance, and fast with high reflectance. The black tape provides low reflectance [2].

After the position on the line is determined, the algorithm provides “turn left” and “turn right” signals to the motors to manoeuver the car back on track. Four DC motors are driven using L293D Dual H-Bridge Motor Driver, and the speed is slowed down using PWM.

Components used:

1. Altera Cyclone IV FPGA
2. QTR-8RC Reflectance Sensor Array
3. 5V, 1A Step-Down Voltage Regulator D24V10Fx
4. L293D Dual H-Bridge Motor Driver
5. Four 12V DC Motors
6. 12V, 1.2 Ah Battery
7. Car frame and wheels

3 Outcome

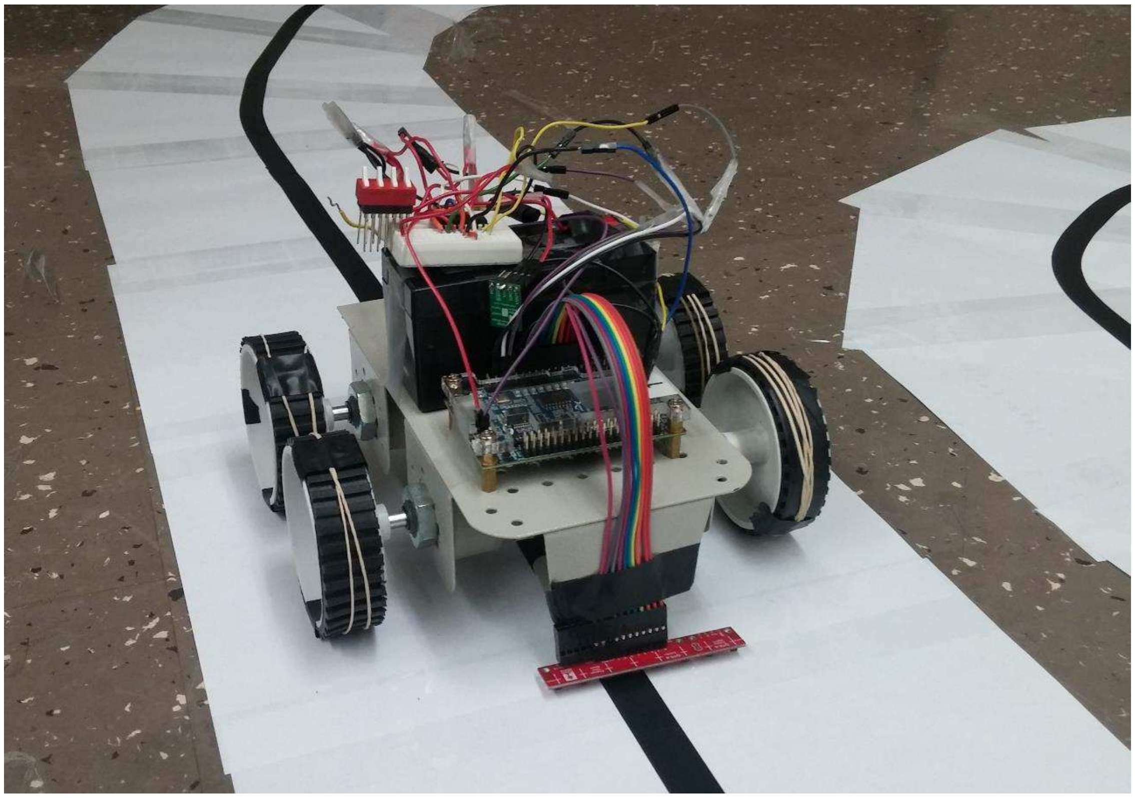


Figure 1: The car (front view)

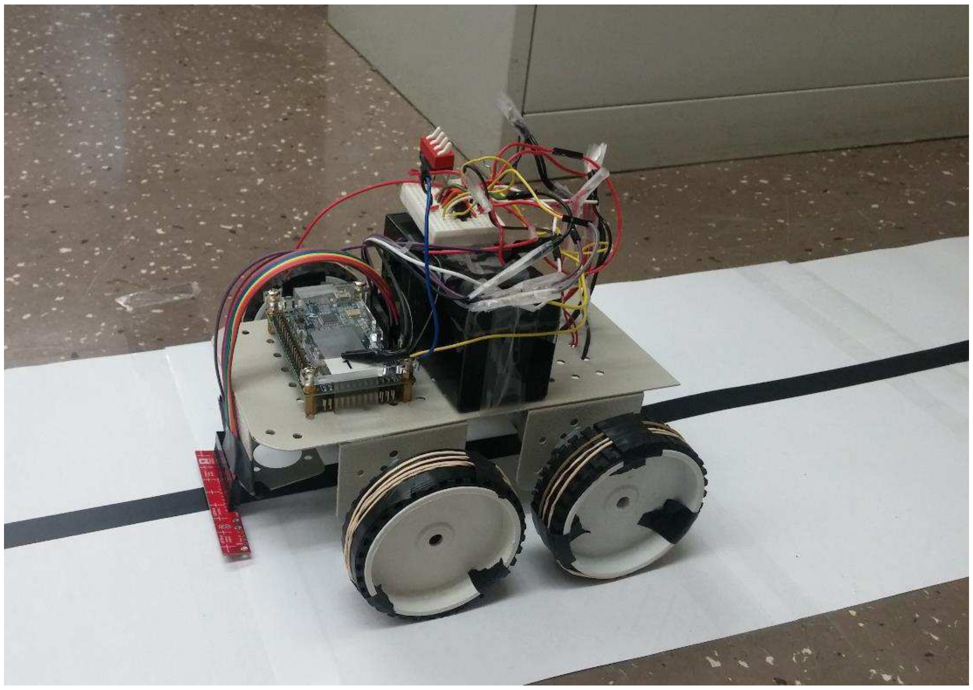


Figure 2: The car (side view)

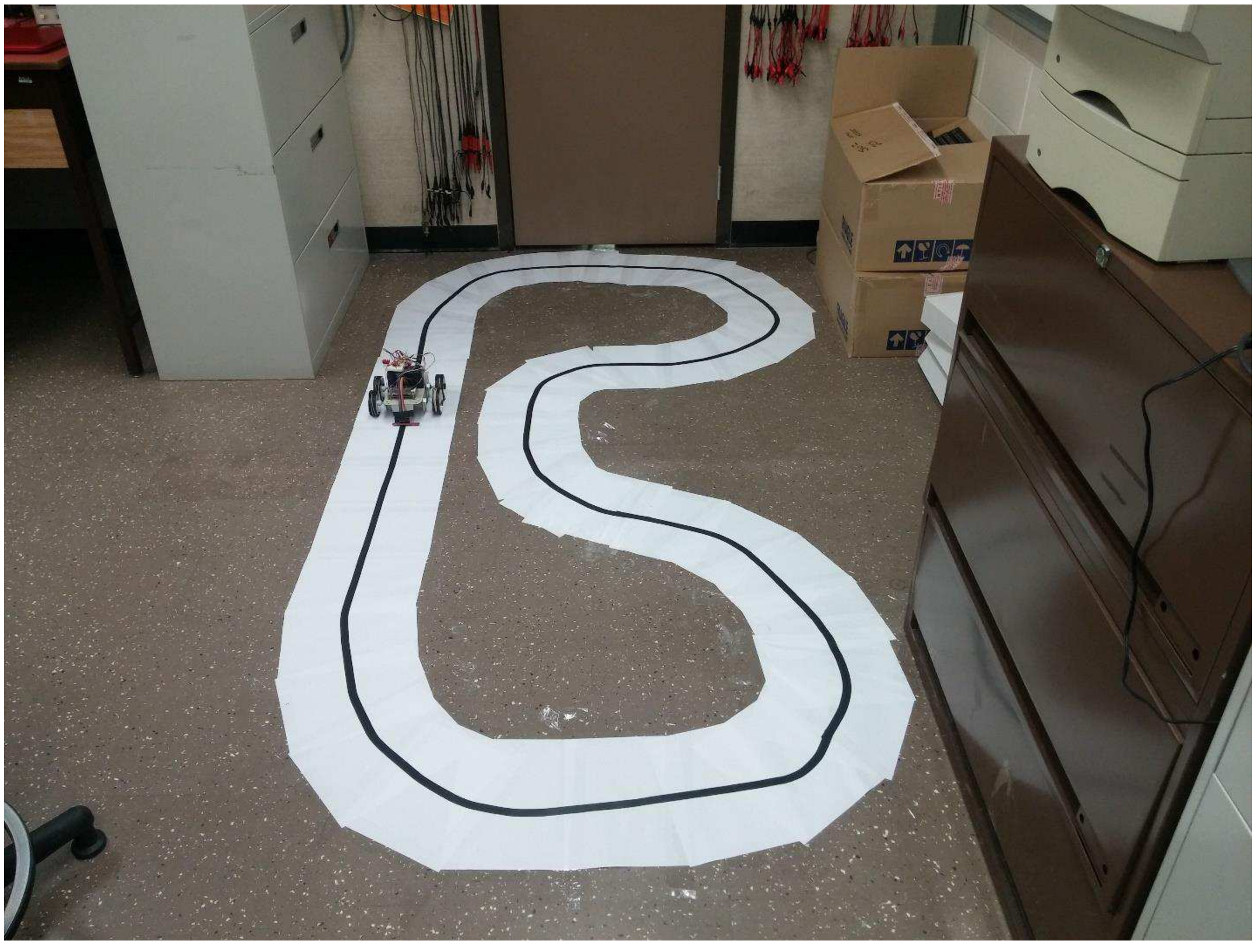


Figure 3: The car on the track

The above figures show our final product which includes a fully functional car on the track.

3.1 Final Result

We successfully implemented the QTR-8RC sensor on the car and programmed it to follow the black line. It worked perfectly fine, and the car stayed on the track as planned. Some changes that we had to make in our initial design were to replace the defective motor controller board with an equivalent L293D IC to get the proper output voltage. With the original motor controller board, we were only getting 5V out, instead of 12, which was not enough to drive the 4 motors. The minimum voltage requirement for the motors was met using a L293D IC.

Overall, the result worked great and exceeded our expectations. Initially, we thought the car will have difficulties turning right or left due to high speed of the motors. But this issue was fixed by using Pulse Width Modulation (PWM) to control the speed of the motors. Also, the “tank” type steering was used, where a pair of wheels on one side turns in one direction, and the other side turns in the opposite direction. This type of steering allowed the car to steer almost on a 90-degree angle. We finished the major portion of the project and some left over add-on features could've been done if we had little more time. The team is proud of the result and considers this project a success.

3.2 Future Improvements

Although we finished the major objective of the project, which was making the car follow the line, we were unable to finish the add-on features. The features include connecting the distance sensor which checks for the obstacles ahead, and putting the speaker on the car which makes different sounds depending on the distance. Also, there could be an improvement of the steering algorithm by using PID control.

4 Technical Details

4.1 Design Block Diagrams

4.1.1 Block Diagram of the overall design

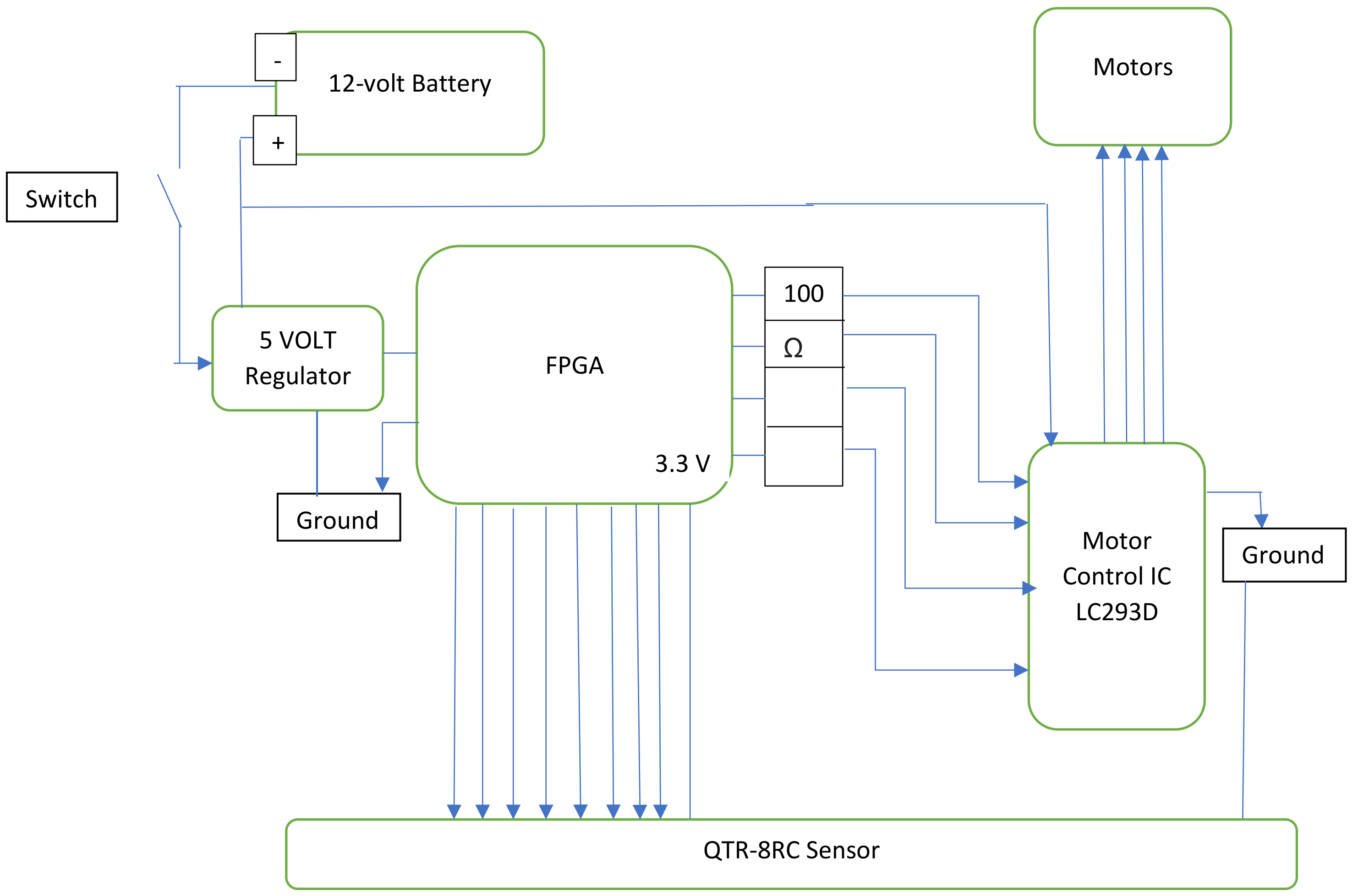
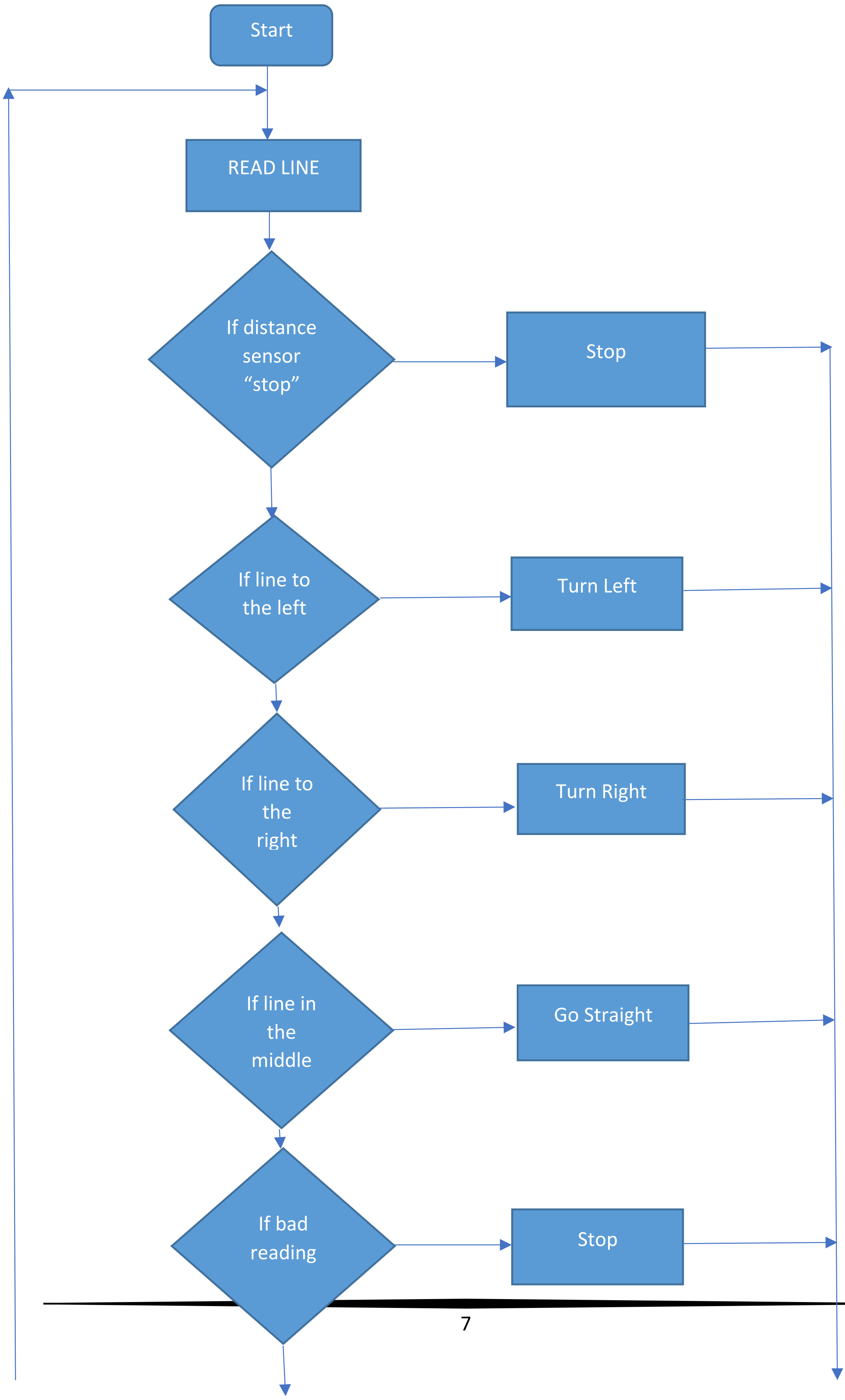


Figure 4: Overall Design Block Diagram

4.1.2 Steering flowchart (implemented in `direction.sv`)



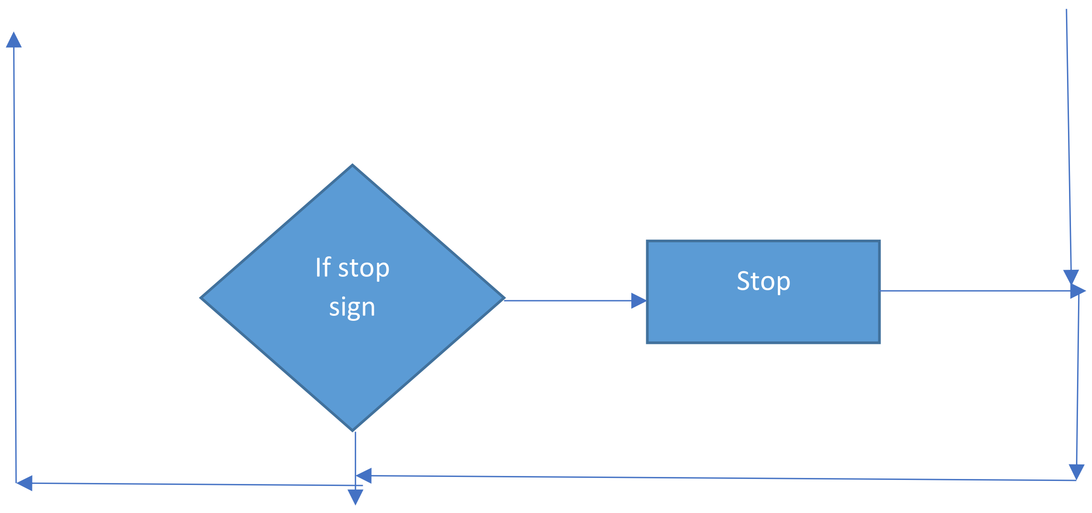


Figure 5: Sensor Array Flow Chart

4.1.3 Flow Chart for the QTR-8RC Sensor (implemented in `ledsarray.sv`)

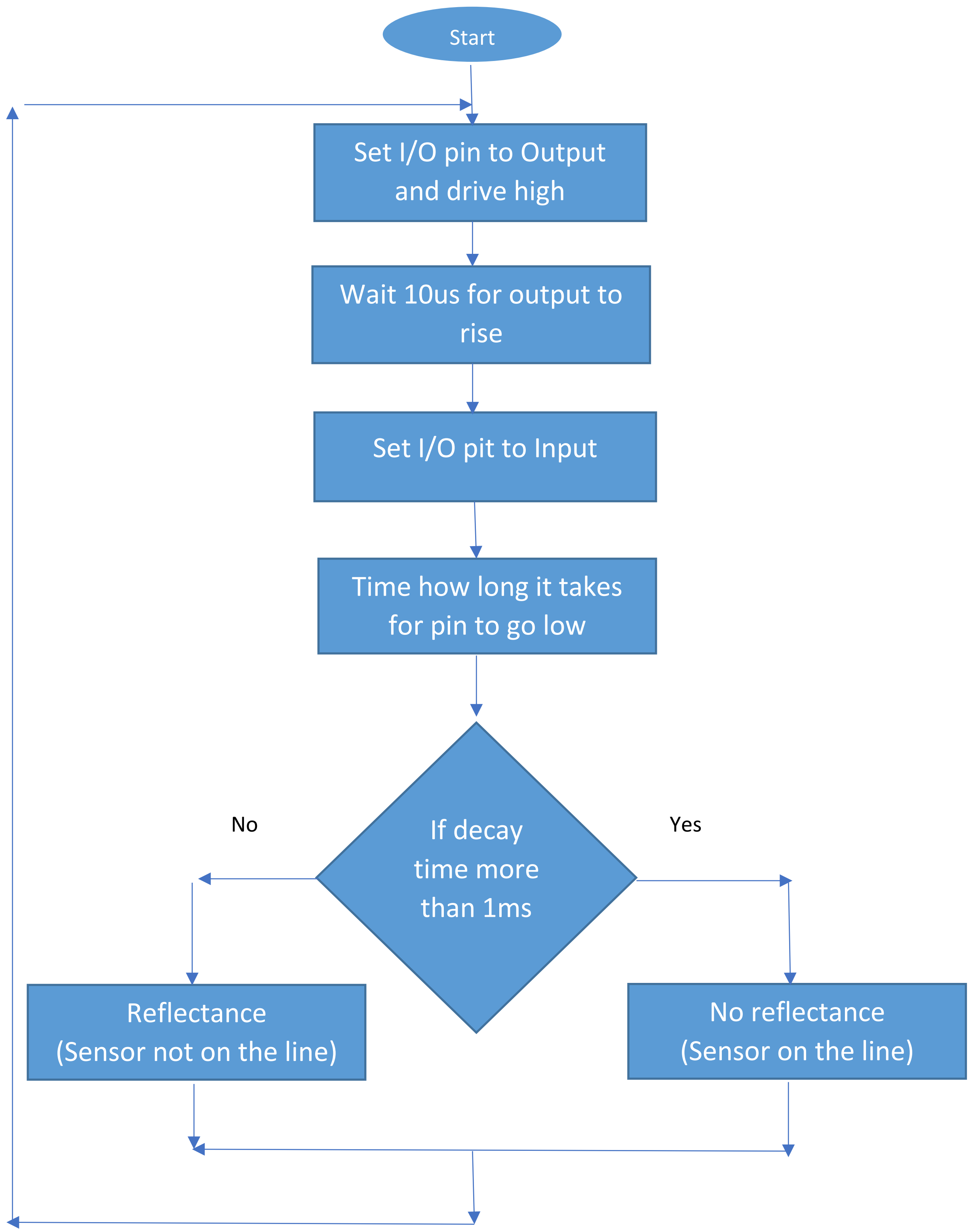

Start
Set I/O pin to Output and drive high
Wait 10us for output to rise
Set I/O pit to Input
Time how long it takes for pin to go low
If decay time more than 1ms
No
Yes
Reflectance (Sensor not on the line)
No reflectance (Sensor on the line)

Figure 6: QTR-8RC Reflectance Sensor Operation Flowchart

4.1.4 PWM Motor Control (implemented in `motorcontrol.sv`)

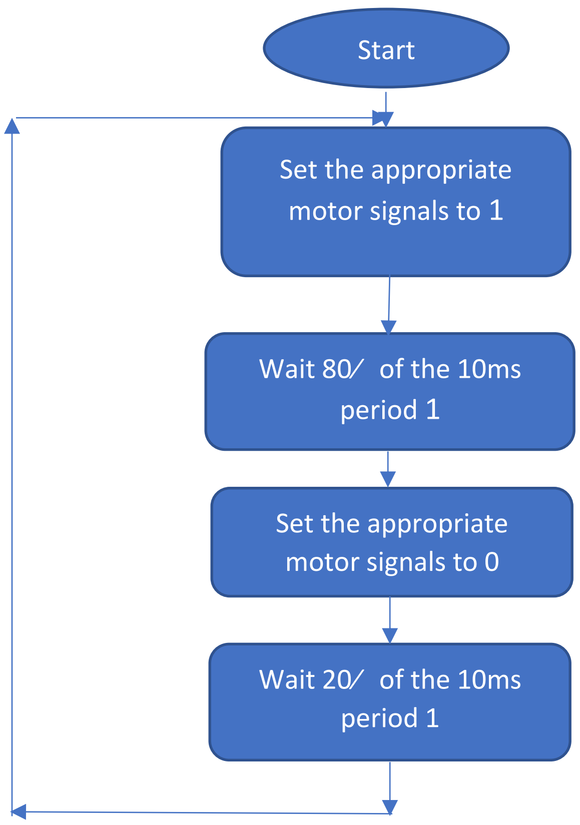
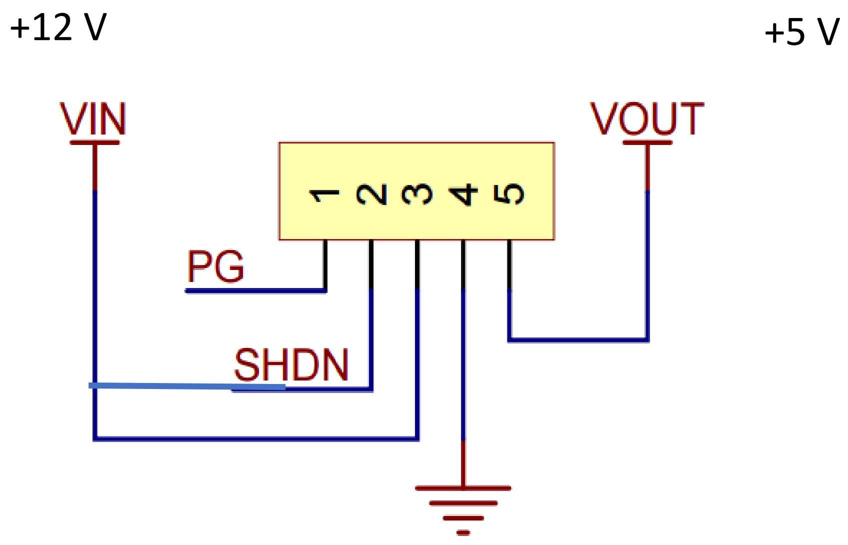

Start
Set the appropriate motor signals to 1
Wait 80⁄ of the 10ms period 1
Set the appropriate motor signals to 0
Wait 20⁄ of the 10ms period 1

Figure 7: PWM flow chart

4.2 Circuit diagrams

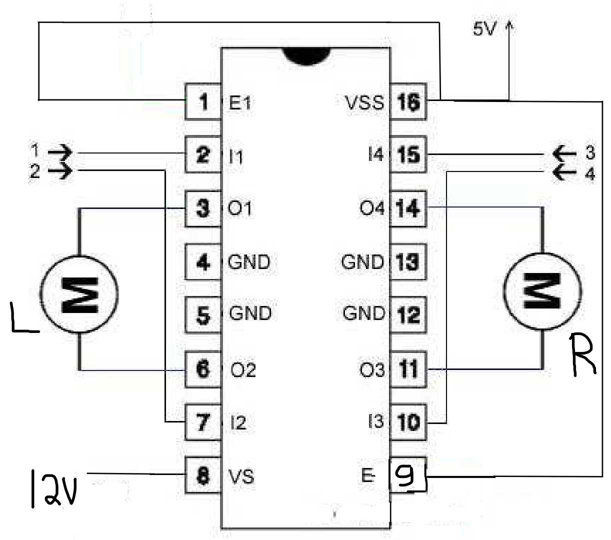
4.2.1 5V the step-down regulator



Scematic credit: Pololu Datasheet [3]

Figure 8: 5V, 1A Step-Down Voltage Regulator D24V10Fx

4.2.2 Motor Control IC: L293D



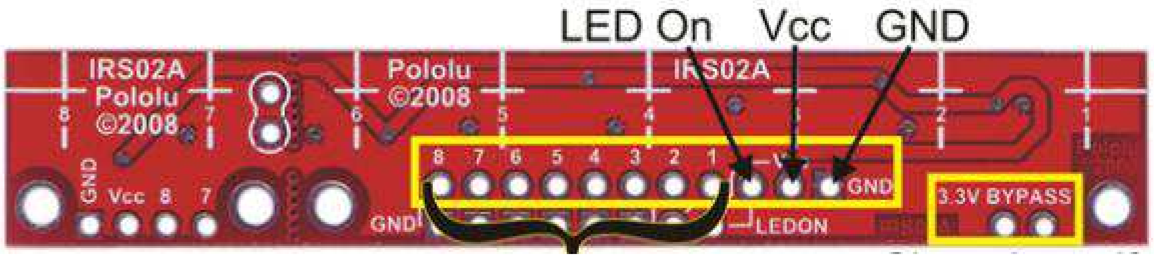
Note: 1,2,3,4 are the control logic signals from the FPGA

Left and Right motors encompass 2 motors in parallel

Schematic credit [4]

Figure 9: L293D H-Bridge Circuit

4.2.3 Sensor Array



Sensor inputs/outputs

Picture Credit: Pololu Datasheet [2]

Figure 10: QTR - 8RC Sensor Array

4.3 Verilog Code

4.3.1 Steering control

The purpose of this code is to examine the output of each of the reflectance sensor, and tell how motors need to engage to get the car back on track. Depending on the relative position to the black line, the code will decide which way to turn. For instance, if the right most sensor is engaged, it means that the car is on the left side of the track, and needs to turn right to stay on track. The steering is done by one side of the car going in one direction, and the other side going in the opposite direction (like a tank).

This part helps us achieve the correct motion of the car by telling which motors need to engage, and in what direction. This is a decision-making algorithm for the car, and is equivalent to a human steering the wheel in the appropriate direction.

```
// Kostiantyn Yushchak & Davneet Singh
//
// April 4, 2017
//
// direction.sv - module for direction control.
//
// Depending on which sensors are engaged, determine the
// steering direction required.

module direction( input logic [7:0] ledso, // reflectance sensor binary output
                  output logic [1:0] Lmotor, Rmotor, // motor binary signals
                  input logic clk, reset // clock, reset
                 );

      // Direction variables.

      logic straight, turnRight, turnLeft, stop;

      // Output motor directin bianry signals.

      always_ff@(posedge clk)
      begin

            if(straight)begin
                  Lmotor <= 1;
                  Rmotor <= 1;

            end
            else if(turnRight)begin
                  Rmotor <= 0;
                  Lmotor <= 1;
            end
            else if(turnLeft)begin
                  Rmotor <= 1;
                  Lmotor <= 0;
            end
            else if(stop) begin
                  Rmotor <= 0;
                  Lmotor <= 0;
            end
            else begin
                  Rmotor <= Rmotor;
                  Lmotor <= Lmotor;
            end
      end

      // Output the direction based on the reflectance sensor outputs.

      always_comb begin
            if (ledso == 8'b00011000 || ledso == 8'b00010000 || ledso ==
8'b00001000)
            begin
                  straight = 1;
                  turnRight = 0;
                  turnLeft = 0;
                  stop = 0;
            end
            else if (ledso == 8'b00001100 || ledso == 8'b00000100 || ledso ==
8'b00001000)
            begin
```

```
                turnRight = 1;
                straight = 0;
                turnLeft = 0;
                stop = 0;
        end
        else if (ledso == 8'b00000110 || ledso == 8'b00000100 || ledso ==
8'b00000010 )
        begin
                turnRight = 1;
                straight = 0;
                turnLeft = 0;
                stop = 0;
        end
        else if (ledso == 8'b00000011)
        begin
                turnRight = 1;
                straight = 0;
                turnLeft = 0;
                stop = 0;
        end
        else if (ledso == 8'b00000001)
        begin
                turnRight = 1;
                straight = 0;
                turnLeft = 0;
                stop = 0;
        end
        else if (ledso == 8'b00110000 || ledso == 8'b00010000 || ledso ==
8'b00100000)
        begin
                turnLeft = 1;
                turnRight = 0;
                straight = 0;
                stop = 0;
        end
        else if (ledso == 8'b01100000 || ledso == 8'b00100000 || ledso ==
8'b01000000)
        begin
                turnLeft = 1;
                turnRight = 0;
                straight = 0;
                stop = 0;
        end
        else if (ledso == 8'b11000000 || ledso == 8'b01000000)
        begin
                turnLeft = 1;
                turnRight = 0;
                straight = 0;
                stop = 0;
        end
        else if (ledso == 8'b10000000)
        begin
                turnLeft = 1;
                turnRight = 0;
                straight = 0;
                stop = 0;
        end

        // 3 SENSORS ENGAGED
        else if (ledso == 8'b00011100)
```

```
            begin
                  straight = 1;
                  turnRight = 0;
                  turnLeft = 0;
                  stop = 0;
            end
            else if (ledso == 8'b00001110)
            begin
                  turnRight = 1;
                  straight = 0;
                  turnLeft = 0;
                  stop = 0;
            end
            else if (ledso == 8'b00000111)
            begin
                  turnRight = 1;
                  straight = 0;
                  turnLeft = 0;
                  stop = 0;
            end
            else if (ledso == 8'b00111000)
            begin
                  turnLeft = 1;
                  turnRight = 0;
                  straight = 0;
                  stop = 0;
            end
            else if (ledso == 8'b01110000)
                  begin
                  turnLeft = 1;
                  turnRight = 0;
                  straight = 0;
                  stop = 0;
            end
            else if (ledso == 8'b11100000)
            begin
                  turnLeft = 1;
                  turnRight = 0;
                  straight = 0;
                  stop = 0;
            end
            else
            begin
                  turnLeft = 0;
                  turnRight = 0;
                  straight = 0;
                  stop = 1;
            end
      end
endmodule
```

4.3.2 Receiving and Interpreting Data from the QTR-8RC Sensor

The purpose of this code was to drive the QTR-8RC Reflectance Sensor Array. It drives each of the sensor pins high, and then measures the time it takes for each pin to decay. Short decay time indicates good reflectance (white surface), and long decay time indicates poor reflectance (black surface). Ultimately it provides an 8-bit array that is composed of binary bits, where 0 stands for a sensor being on the white surface, and 1 stands for the sensor being on the black surface.
This piece of code is a vital component of the project since it provides the information about the position of the car. All the later algorithms are based on the information from the 8-bit array outhunted by this code.

```
// Kostiantyn Yushchak & Davneet Singh
//
// April 4, 2017
//
// ledsarray.sv - code for the led array sensor
//
// Determines if each sensor is on the line or not.
// If on the line, outputs a 1 on ledsout[sensor#].

module ledsarray( inout tri [7:0] leds,      // reflectance sensor inout
                  output logic [7:0] ledso, // reflectance sensor binary signal
                  input logic reset, clk);  // reset and clock

      enum logic [2:0] // enumeration for the states
            {
             ledsout,    // set pins as output
             ledsin,     // set pins as input
             wait1,      // wait 10 us for sensors to charge
             wait2,            // wait x amount before reading sensors
             test        // read sensors
            } state, state_next;

      enum logic [1:0]
            {infinite,
             in,
             out
            } direction; // enum for pin direction, and loop the program forever

      logic signed [31:0] count1, count2; // count register

      parameter tenu = 500;   // 10us dealy for sensors to chrage up. (500 clk
cycles)
      parameter hunu = 20000; // 100us dealy before checking sensors. (5000 clk
cycles)

    // Register

      always_ff@(posedge clk)
      begin

            if (state == ledsout)begin
                  leds <= '1; // set pins high
                  direction = out;
```

```
end
else if (state == ledsin)begin
      leds <= 'z; // read from pins
      direction = in;
end

if (state == wait1) // 10us counter
      count1 <= count1 + 1'b1;
else
      count1 <= 0; // if not counting counter should remain at zero


if (state == wait2) // 100us counter
      count2 <= count2 + 1'b1;
else
      count2 <= 0; // if not counting counter should remain at zero

if (state == test)
begin
direction = infinite;

// Tell the posiotion of each sensor (if on the line or not)

      if(leds[0] ==1'b1)
            ledso[0] <= 1'b1;
      else
            ledso[0] <= 1'b0;

            if(leds[1] ==1'b1)
            ledso[1] <= 1'b1;
      else
            ledso[1] <= 1'b0;

            if(leds[2] ==1'b1)
            ledso[2] <= 1'b1;
      else
            ledso[2] <= 1'b0;

            if(leds[3] ==1'b1)
            ledso[3] <= 1'b1;
      else
            ledso[3] <= 1'b0;

            if(leds[4] ==1'b1)
            ledso[4] <= 1'b1;
      else
            ledso[4] <= 1'b0;

            if(leds[5] ==1'b1)
            ledso[5] <= 1'b1;
      else
            ledso[5] <= 1'b0;

            if(leds[6] ==1'b1)
            ledso[6] <= 1'b1;
      else
            ledso[6] <= 1'b0;

            if(leds[7] ==1'b1)
            ledso[7] <= 1'b1;
```

```
                else
                        ledso[7] <= 1'b0;

        end


    end

    // Controller register

    always_ff@(posedge clk)
    begin
        state <= state_next;
    end

    always_comb begin

        state_next = state;

        if ((reset == 0) || (direction == infinite))
            state_next = ledsout;
        if (direction == out)
            state_next = wait1;
        if (count1 >= tenu)
            state_next = ledsin;
        if (direction == in)
            state_next = wait2;
        if (count2 >= hunu)
            state_next = test;
    end
endmodule
```

4.3.3 PWM to control the motor

The purpose of this code is to control the speed of the motors using pulse width modulation (PWM), and send those signals to the L293D dual H-bridge motor driver. By trial, we chose a 10ms period with 80⁄ duty cycle as the parameters for the adequate maneuvering of the car on the track.
This code helps us achieve a smooth car motion. If we were to directly connect the motors to the 12-V battery, the speed would be too fast, and the car would be moving in the "zig-zag" motion, or completely drive off the track.

```
// Kostiantyn Yushchak & Davneet Singh
//
// April 4, 2017
//
// motorcontrol.sv - motor speed controller
//
// Control the speed of left and right car motors using PWM.
// Set "highduty" and "lowduty" for number of cycles for each portion
// of the period.

module motorcontrol(
                    output logic [1:0] LmotorPW, RmotorPW, // motor PWM signals
                    input logic [1:0] Lmotor, Rmotor,      // motor logic signals
                    input logic reset, clk ) ;             // reset and clock

      // Use 100HZ frequency. 10ms
      // 50% dcycle
      // CLock is 50MHz =  20n. 10ms/20ns = 500 000 cycles in 10 ms
      // 5ms/20ns = 250 000 cycles 1.

      logic [1:0] LmotStatus, RmotStatus; // Tells if motors are engaged
      logic signed [31:0] count1, count2; // count register

      parameter highduty = 400000;        // Number of cycles output is 1
      parameter lowduty = 100000;         // Number of cycles output is 0

      enum logic [2:0] // enumeration for the states
            {
             high,       // positive duty cycle
             low,        // 0 duty cycle
             wait1,      // positive duty cycle wait
             wait2             // 0 duty cycle wait
            } state, state_next;

      enum logic [1:0] // enumeration to swith between states. names are not
significant
            {infinite,
             in,
             out
            } direction;

      always_ff@(posedge clk)
      begin

            // Applly positive PWM to the appropriate motor.
```

```
if (state == high)begin
      direction = out; // to move to the next state

      // Left motor

      if(Lmotor == 1) // left motor forward
      begin
            LmotorPW[0] <= 1'b1;
            LmotorPW[1] <= 1'b0;
            LmotStatus <= 1;
      end
      else if(Lmotor == 2) // left motor backwards
      begin
            LmotorPW[0] <= 1'b0;
            LmotorPW[1] <= 1'b1;
            LmotStatus <= 2;
      end
      else if(Lmotor == 0) //left motor don't move
      begin
            LmotorPW[0] <= 1'b0;
            LmotorPW[0] <= 1'b0;
            LmotStatus <= 0;
      end

      // Right motor

      if(Rmotor == 1) // left motor forward
      begin
            RmotorPW[0] <= 1'b1;
            RmotorPW[1] <= 1'b0;
            RmotStatus  <= 1;
      end
      else if(Rmotor == 2) // left motor backwards
      begin
            RmotorPW[0] <= 1'b0;
            RmotorPW[1] <= 1'b1;
            RmotStatus  <= 2;
      end
      else if(Rmotor == 0) //left motor don't move
      begin
            RmotorPW[0] <= 1'b0;
            RmotorPW[0] <= 1'b0;
            RmotStatus  <= 0;
      end

end

// Applly 0 to the appropriate motor.

if (state == low)begin

      direction = in; // to move to the next step

      // Left motor

      if(LmotStatus == 1)      // left motor forward
      begin
            LmotorPW[0] <= 1'b0;
            LmotorPW[1] <= 1'b0;
            end
            else if(LmotStatus == 2) // left motor backwards
            begin
                  LmotorPW[0] <= 1'b0;
                  LmotorPW[1] <= 1'b0;
            end
            else if(LmotStatus == 0) //left motor don't move
            begin
                  LmotorPW[0] <= 1'b0;
                  LmotorPW[0] <= 1'b0;
            end

            // Right motor

            if(RmotStatus == 1)       // left motor forward
            begin
                  RmotorPW[0] <= 1'b0;
                  RmotorPW[1] <= 1'b0;
            end
            else if(RmotStatus == 2)  // left motor backwards
            begin
                  RmotorPW[0] <= 1'b0;
                  RmotorPW[1] <= 1'b0;
            end
            else if(RmotStatus == 0)  // left motor don't move
            begin
                  RmotorPW[0] <= 1'b0;
                  RmotorPW[0] <= 1'b0;
            end

      end

      if (state == wait1)                  // high duty cycle counter
            count1 <= count1 + 1'b1;
      else
            count1 <= 0; // if not counting counter should remain at zero


      if (state == wait2)                  // low duty cycle counter
            count2 <= count2 + 1'b1;
      else
            count2 <= 0; // if not counting counter should remain at zero
end

// Controller register

always_ff@(posedge clk)
begin
      state <= state_next;
end

always_comb begin

      state_next = state;

      if (reset == 0)
            state_next = high;
      if (direction == out)
            state_next = wait1;
      if (count1 >= highduty)
```

```
                    state_next = low;
            if (direction == in)
                    state_next = wait2;
            if (count2 >= lowduty)
                    state_next = high;
      end
endmodule
```

4.3.4 Top Level Module for this project

The code instantiates all the modules for proper operation, as well as defines the common clock and reset.

```
// Kostiantyn Yushchak & Davneet Singh
//
// April 4, 2017
//
// cartop.sv - top-level module for ELEX 7660 Project.

module cartop (
                input logic CLOCK_50,    // clock
                input logic [1:0] KEY,   // reset
                inout tri [7:0] leds,    // reflectance sensor inout
                output logic [1:0] LmotorPW, RmotorPW // motor PWM signals
               );

      logic [7:0] ledso; // reflectance sensor binary signal
      logic [1:0] Lmotor, Rmotor; // motor binary signals

   // Instantiate reflectance sensor module.

   ledsarray u0 (
            .clk(CLOCK_50),
            .leds(leds),
                  .ledso(ledso),
            .reset(KEY[0])
            );

      // Instantiate PWM module.

      motorcontrol u1 (
                        .clk(CLOCK_50),
                        .reset(KEY[0]),
                        .Lmotor(Lmotor),
                        .Rmotor(Rmotor),
                        .LmotorPW(LmotorPW),
                        .RmotorPW(RmotorPW)
                        );

      // Instantiate direction module.

      direction u2 (
                        .ledso(ledso),
                        .Lmotor(Lmotor),
                        .Rmotor(Rmotor),
                        .clk(CLOCK_50),
                        .reset(KEY[0])
                        );

endmodule
```

5 References

[1] "Line Following Robot," [Online]. Available: https://circuitdigest.com/microcontroller-projects/line-follower-robot-using-arduino.

[2] "QTR-8RC Reflectance Sensor Data Sheet," [Online]. Available: https://www.pololu.com/docs/pdf/0J12/QTR-8x.pdf.

[3] "1A Step-Down Voltage Regulator," [Online]. Available: https://www.pololu.com/file/0J843/d24v10fx-schematic.pdf.

[4] "H-Bridge Circuit," [Online]. Available: http://www.ti.com/lit/ds/symlink/l293.pdf.