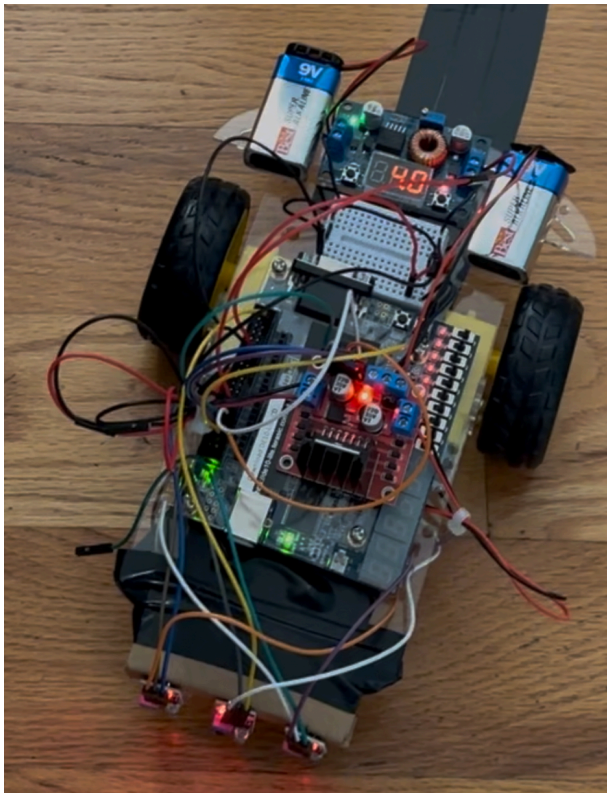
The City College of New York

Grove School of Engineering

EE 47200 Digital Design using Verilog

Fall 24 Semester



Line Following Robot

Instructor: Hakan Pekcan

Names: Annah Singh & Matthew Ching

Introduction

The basic instructions for the assignment required a project with a working simulation. For our project we decided to leverage our electronics background, in conjunction with our knowledge of Verilog HDL and FPGA boards. We decided to further expand and test the limits of FPGA by designing a line following robot which uses three infrared sensors to track a line. This project also serves as a baseline for further robotics projects, like our senior design project, which will involve numerous components that we have already used in this project.

Process

The first steps to getting a robot to work is researching and buying all the necessary components. A line following robot is usually the introductory robot for enthusiasts and this is no different for us. When doing our research we found that the best way to start was by buying and testing the motors and infrared sensors in Arduino first to get a better idea of how each component works in code. Below is our parts list and an explanation of what each part does.

- Smart Car Chassis: Used to mount the components onto and have a vehicle that can run
- TT Gearbox Motors: Basic motors with 2 leads that drive the motor forward or backward
- Infrared Sensors: Infrared sensors emit infrared radiation which is absorbed by electrical tape
- L298N Motor Driver: Splits our power to the 2 motors and allows the use of pulse width modulation
- LM2596 with Display: Used to take a 9V battery output and step down to 4.8V for FPGA board
- Terasic DE10-Lite: Max 10 FPGA board recommended by Professor Pekcan

- Cardboard, Wires, Batteries, Electrical Tape, and Zip Ties: Used to connect our components, give power, insulate exposed wires and make our track and cable manage

After purchasing all of our components our next task was to test our motors by applying power from our batteries to make sure the motors would spin correctly with enough speed out of the box. Since both motors worked we attached them to the chassis and continued to the next part.

Figure 1: TT Gearbox Motor (Amazon)

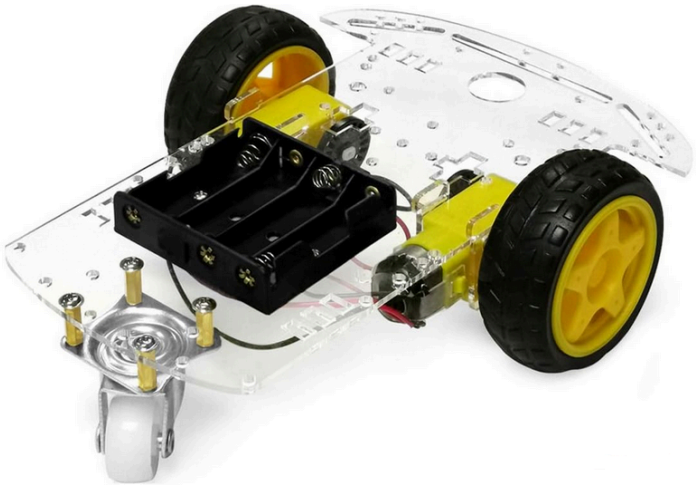


Figure 2: Smart Car Chassis Purchased from Amazon

Since we did not want our motors to spin at maximum speed and possibly miss turns, we decided to use the L298N motor driver which allows for PWM to be used as the driver provides enable pins. When testing this component we first screwed in all the wires from the motors and the batteries into the ports and used the 5V output from the driver to power the motors. With each pin successfully turning the motors forward and backwards we could then uncap the enable pins and test with that. Once all 6 input pins on the module were working we were able to move to testing our sensors.

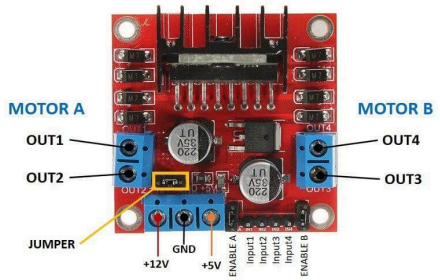


Figure 3: L298N Motor Driver (Xukyo, 2022)

The infrared sensors were tested using Arduino Uno to figure out how to use it and translate the code to Verilog HDL. Since infrared sensors are active low the sensor supplies 5V of power when nothing the sense or signal LED on the sensor is off (Figure 1). When the LED is on it means there is an object in its path and the sensor supplies 0V.

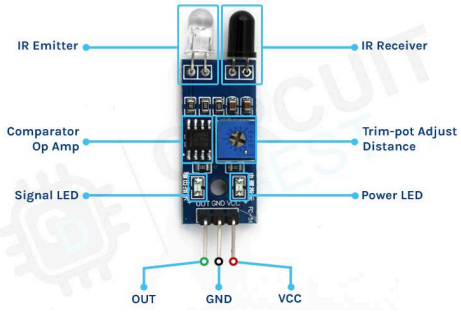


Figure 4: Parts of an Infrared Sensor (Das, 2022)

Since electrical tape absorbs infrared, the sensor acts as if there were nothing in front of it and supplies 5V. Using this we were able to test each sensor 1 by 1 and add them to the front of our chassis using cardboard to act as a front bumper.

After testing all those components we were finally able to start coding in Verilog. Since we had physical hardware to test our code on we decided to slowly implement each component of our robot and build up instead of coding everything at once and not being able to know what was the cause for our robot not working. The way our code works and what steps we took to arrive at our final module will be discussed in the next section. When powering our robot we quickly realized we had too many components to only use 4 AA batteries. We also realized that the motor driver was very power hungry and we decided it was best to separate our power between 2 9V batteries. The motor driver has a 5V output but we deemed it was best to avoid using it and instead use the 9V batteries, one for the DE10-Lite board and the other for the motor drivers. Since the FPGA board's pins are not rated for higher than 5V we needed to use the LM2596 voltage regulator to take the 9V power supply and turn it down to 4.8V which would be enough to power the board and the infrared sensors through it. Lastly we had to mount and wire everything using tie wraps, hot glue, electrical tape, cardboard, and double sided tape. We made a track for testing on the floor using electrical tape and a new track using copy paper for our demonstration.

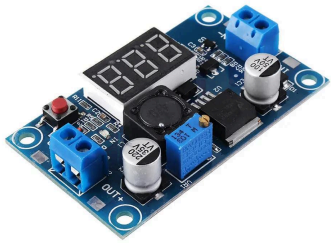


Figure 5: LM2596 Voltage Regulator (isakh1, 2021)

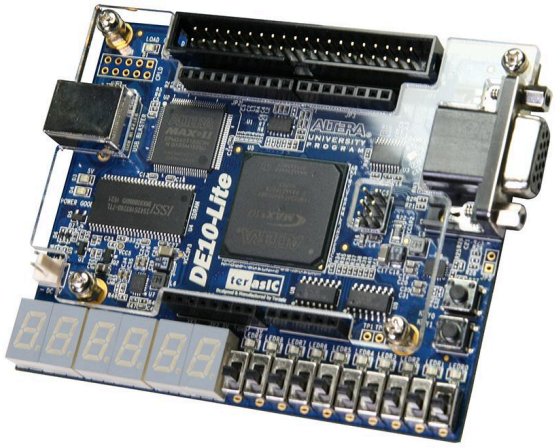


Figure 6: Terasic DE10-Lite FPGA Board (Terasic)

Code

The first step was to make our motors turn. We defined a new module called sensor1 and instantiated 4 output reg pins named left_motor1, left_motor2, right_motor1, right_motor2. The outputs that ended in 1 turned the motor forward and the outputs which ended in 2 turned the motor backwards. We then used dataflow modeling to just set the motor to drive forward by using

assign left_motor1, right_motor1 = 1;

After compiling and setting the pin assignments when we uploaded the code to the board using the .sof file the motors spun and we were able to implement the sensors. We knew that eventually we needed to implement all 3 sensors, so we decided to define them all in the port declaration, but comment them out as we tested. Since our middle sensor is the sensor which makes us go forward we decided to first code that as we already had a working basis for moving forward. We named the middle sensor “ir_sensor2” and the other 2 sensors followed the same naming scheme with ir_sensor1 on the left and ir_sensor3 on the right. When coding with the sensors we knew

we couldn't use blocking assignments as we needed to quickly change between the 3 ways we would move, so we used behavioral modeling.

sensor1.v — Compilation Report

```
1  module sensor1(output reg left_motor1, left_motor2, right_motor1, right_motor2,
2                 input wire /*ir_sensor1,*/ ir_sensor2 /*,ir_sensor3*/);
3
4
5      always @ (/*ir_sensor1,*/ ir_sensor2/*, ir_sensor3*/)
6         begin: sensors
7         if ( /*ir_sensor1 == 0 && */ ir_sensor2 == 1 /*&& ir_sensor3 == 0*/)
8            begin
9                     //go forward
10                    left_motor1 = 1;
11                    left_motor2 = 0;
12                    right_motor1 = 1;
13                    right_motor2 = 0;
14           end
15
16        else
17           begin
18                    left_motor1 = 0;
19                    left_motor2 = 0;
20                    right_motor1 = 0;
21                    right_motor2 = 0;
22           end
23
24        end
25     endmodule
26
27
```

Figure 7: Testing Motors with Middle IR Sensor

After seeing the success we had with 1 sensor we knew the motors were moving too fast and this caused us to overshoot the turns or completely go off track so we had to implement PWM. Pulse Width Modulation (PWM) is accomplished by taking into account the functionality of the enable pins on the motor driver. Turning on enable A will allow the left motor to spin when a signal is sent to either the forward or backwards pin, the same applies for enable B and the right motor. Meaning, even if there is a signal telling the motor to spin forwards or backwards, the motor will not spin unless the enable is set high. Using the built-in clock on the DE10-Lite, a counter is created that counts to 100, and when the counter is less than 55 the enable pins are high, creating a 55% duty cycle. In our code we initially chose 60, but realized it

was still slightly too fast and that 50 was too slow and would cause the motors to not spin. We eventually changed to 55 as seen in Figure 10.1.

sensor1.v — Compilation Report - sensor1

```
1  module sensor1(output reg left_motor1, left_motor2, right_motor1, right_motor2, ena, enb,
2                 input wire /*ir_sensor1,*/ ir_sensor2 /*,ir_sensor3*/, clk);
3
4
5     reg [7:0] counter = 100;
6
7        always @ (posedge clk)
8     begin: PWM
9        if (counter < 100)
10          counter <= counter + 1;
11       else
12          counter <= 0;
13    end
14
15    always @ (counter)
16    begin: PWM_counter
17       if (counter < 60)
18       begin
19          ena = 1;
20          enb = 1;
21       end
22       else
23       begin
24          ena = 0;
25          enb = 0;
26       end
27    end
28
```

Figure 8: PWM counter with enable ports in port declaration

We also realized that the robot would slightly favor turning right so we had to start balancing the robot back to the middle. We started by making ir_sensor1 turn left if it was over the electrical tape. When coding turns with wheels that can only move 2 ways, we had to be creative and realize that if one motor spun backwards and the other spun forwards it would slightly turn the robot and help it stay on track. Since our robot turned right slightly we had to turn left meaning that we had to spin the left motor backwards and the right motor forward which can be seen in the code above. We mirrored this to turn right as seen below.

```
else if (ir_sensor1 == 1 && ir_sensor2 == 0 && ir_sensor3 == 0)
   begin
            // go left
            left_motor1 = 0;
            left_motor2 = 1;
            right_motor1 = 1;
            right_motor2 = 0;


   end

else if (ir_sensor1 == 0 && ir_sensor2 == 0 && ir_sensor3 == 1)
   begin
            // go right
            left_motor1 = 1;
            left_motor2 = 0;
            right_motor1 = 0;
            right_motor2 = 1;


   end

else
   begin
            left_motor1 = 0;
            left_motor2 = 0;
            right_motor1 = 0;
            right_motor2 = 0;
```

Figure 9: Turning mechanisms

There may be a case when neither or all 3 of the sensors are on the line of electrical tape, when this happens we need to make the robot stop moving which is where the final else block comes in to make sure the robot won't do anything we did not ask it to do.

Lastly, we added "blinkers" like seen in a car for visuals on the board for the direction we would be turning. As there are 10 LEDs available on the board, there is an always on LED which means our robot is ready to be placed on the track, and the rest are split into 3 LEDs per direction. When the robot is driving it looks as if it has a slider on it as the LEDs rapidly change due to FPGAs nearly instantaneous change. Our complete module is below along with the pin planner for the DE10-Lite board.

sensor1.v

```
module sensor1(output reg left_motor1, left_motor2, right_motor1, right_motor2, ena, enb, LED_ON,
               output reg [2:0] LED_LEFT, LED_RIGHT, LED_STR,
               input wire ir_sensor1, ir_sensor2, ir_sensor3, clk);

   initial
      begin
      LED_ON = 1;
      LED_STR = 3'b000;
      LED_LEFT = 3'b000;
      LED_RIGHT = 3'b000;
      end



   reg [7:0] counter = 100;

   always @ (posedge clk)
   begin: PWM
      if (counter < 100)
      begin
         counter <= counter + 1;
      end
      else
      begin
         counter <= 0;
      end
   end

   always @ (counter)
   begin: PWM_counter
      if (counter < 55)
      begin
         ena = 1;
         enb = 1;
      end
      else
      begin
         ena = 0;
         enb = 0;
      end
   end

   always @ (ir_sensor1, ir_sensor2, ir_sensor3)
      begin: sensors
      if ( ir_sensor1 == 0 &&  ir_sensor2 == 1 && ir_sensor3 == 0)
         begin
                  //go forward
                  left_motor1 = 1;
                  left_motor2 = 0;
                  right_motor1 = 1;
                  right_motor2 = 0;

                  //Check for direction
                  LED_ON = 1;
                  LED_STR = 3'b111;
         end

      else if (ir_sensor1 == 1 && ir_sensor2 == 0 && ir_sensor3 == 0)
         begin
                  // go left
                  left_motor1 = 0;
                  left_motor2 = 1;
                  right_motor1 = 1;
                  right_motor2 = 0;
```

Figure 10.1: Complete Code for Line Following Robot

```
67
68                      //Check for direction
69                      LED_ON = 1;
70                      LED_LEFT = 3'b111;
71
72              end
73
74          else if (ir_sensor1 == 0 && ir_sensor2 == 0 && ir_sensor3 == 1)
75              begin
76                      // go right
77                      left_motor1 = 1;
78                      left_motor2 = 0;
79                      right_motor1 = 0;
80                      right_motor2 = 1;
81
82                      //Check for direction
83                      LED_ON = 1;
84                      LED_RIGHT = 3'b111;
85              end
86
87          else
88              begin
89                      left_motor1 = 0;
90                      left_motor2 = 0;
91                      right_motor1 = 0;
92                      right_motor2 = 0;
93
94                      //Check for direction
95                      LED_STR = 3'b000;
96                      LED_LEFT = 3'b000;
97                      LED_RIGHT = 3'b000;
98                      LED_ON = 1;
99              end
100
101         end
102     endmodule
103
104
```

Figure 10.2: Complete Code for Line Following Robot

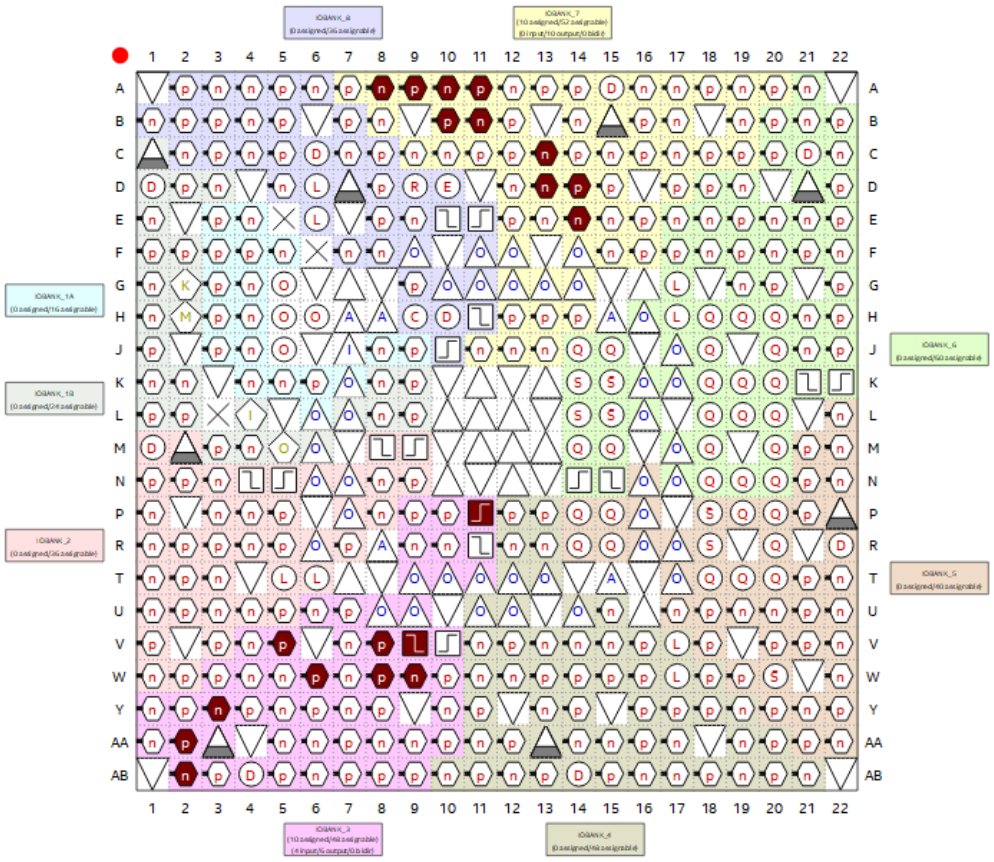


Figure 11: Pin Planner for DE10-Lite

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Strict Preservation
LED_LEFT[2]	Output	PIN_B10	7	B7_N0	PIN_B10	2.5 V		12mA (default)	2 (default)		
LED_LEFT[1]	Output	PIN_A10	7	B7_N0	PIN_A10	2.5 V		12mA (default)	2 (default)		
LED_LEFT[0]	Output	PIN_A9	7	B7_N0	PIN_A9	2.5 V		12mA (default)	2 (default)		
LED_ON	Output	PIN_A8	7	B7_N0	PIN_A8	2.5 V		12mA (default)	2 (default)		
LED_RIGHT[2]	Output	PIN_B11	7	B7_N0	PIN_B11	2.5 V		12mA (default)	2 (default)		
LED_RIGHT[1]	Output	PIN_A11	7	B7_N0	PIN_A11	2.5 V		12mA (default)	2 (default)		
LED_RIGHT[0]	Output	PIN_D14	7	B7_N0	PIN_D14	2.5 V		12mA (default)	2 (default)		
LED_STR[2]	Output	PIN_E14	7	B7_N0	PIN_E14	2.5 V		12mA (default)	2 (default)		
LED_STR[1]	Output	PIN_C13	7	B7_N0	PIN_C13	2.5 V		12mA (default)	2 (default)		
LED_STR[0]	Output	PIN_D13	7	B7_N0	PIN_D13	2.5 V		12mA (default)	2 (default)		
clk	Input	PIN_P11	3	B3_N0	PIN_P11	2.5 V		12mA (default)			
ena	Output	PIN_W6	3	B3_N0	PIN_W6	2.5 V		12mA (default)	2 (default)		
enb	Output	PIN_V5	3	B3_N0	PIN_V5	2.5 V		12mA (default)	2 (default)		
ir_sensor1	Input	PIN_Y3	3	B3_N0	PIN_Y3	2.5 V		12mA (default)			
ir_sensor2	Input	PIN_AB2	3	B3_N0	PIN_AB2	2.5 V		12mA (default)			
ir_sensor3	Input	PIN_AA2	3	B3_N0	PIN_AA2	2.5 V		12mA (default)			
left_motor1	Output	PIN_V9	3	B3_N0	PIN_V9	2.5 V		12mA (default)	2 (default)		
left_motor2	Output	PIN_V8	3	B3_N0	PIN_V8	2.5 V		12mA (default)	2 (default)		
right_motor1	Output	PIN_W9	3	B3_N0	PIN_W9	2.5 V		12mA (default)	2 (default)		
right_motor2	Output	PIN_W8	3	B3_N0	PIN_W8	2.5 V		12mA (default)	2 (default)		

Figure 12: Pin Assignments for DE10-Lite

After compiling the code and assigning the pins we were able to upload the code to the board using the provided USB-Blaster and the .pof file created by compiling the code. At first we would use .sof, but .sof is only used for when the board is connected to the device which uploaded to it. To have the board store the code in memory we have to use .pof which loads and configures the board to use the instructions until overwritten.

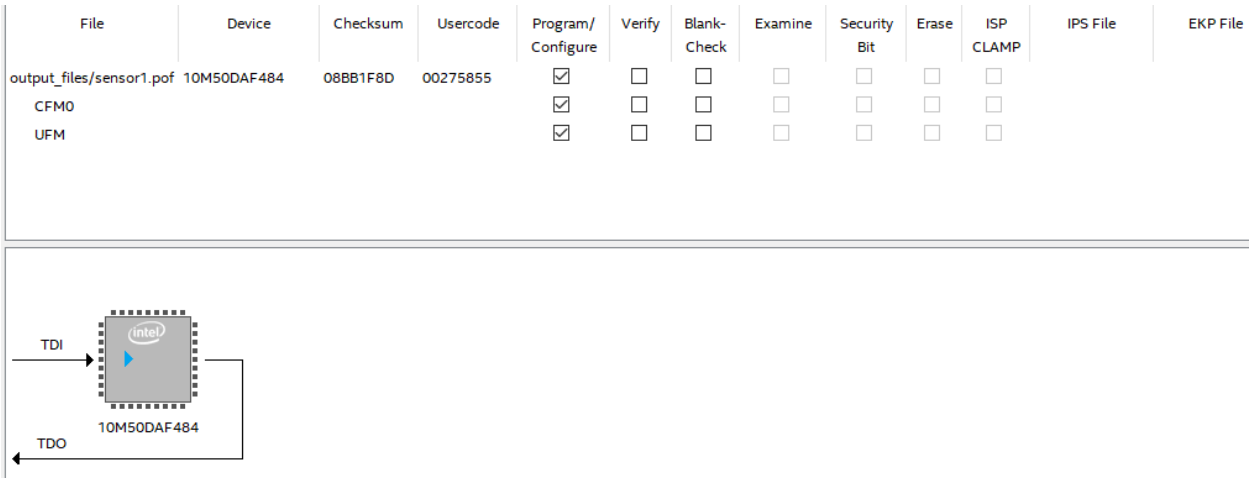


Figure 13: sensor1.pof being uploaded to the board

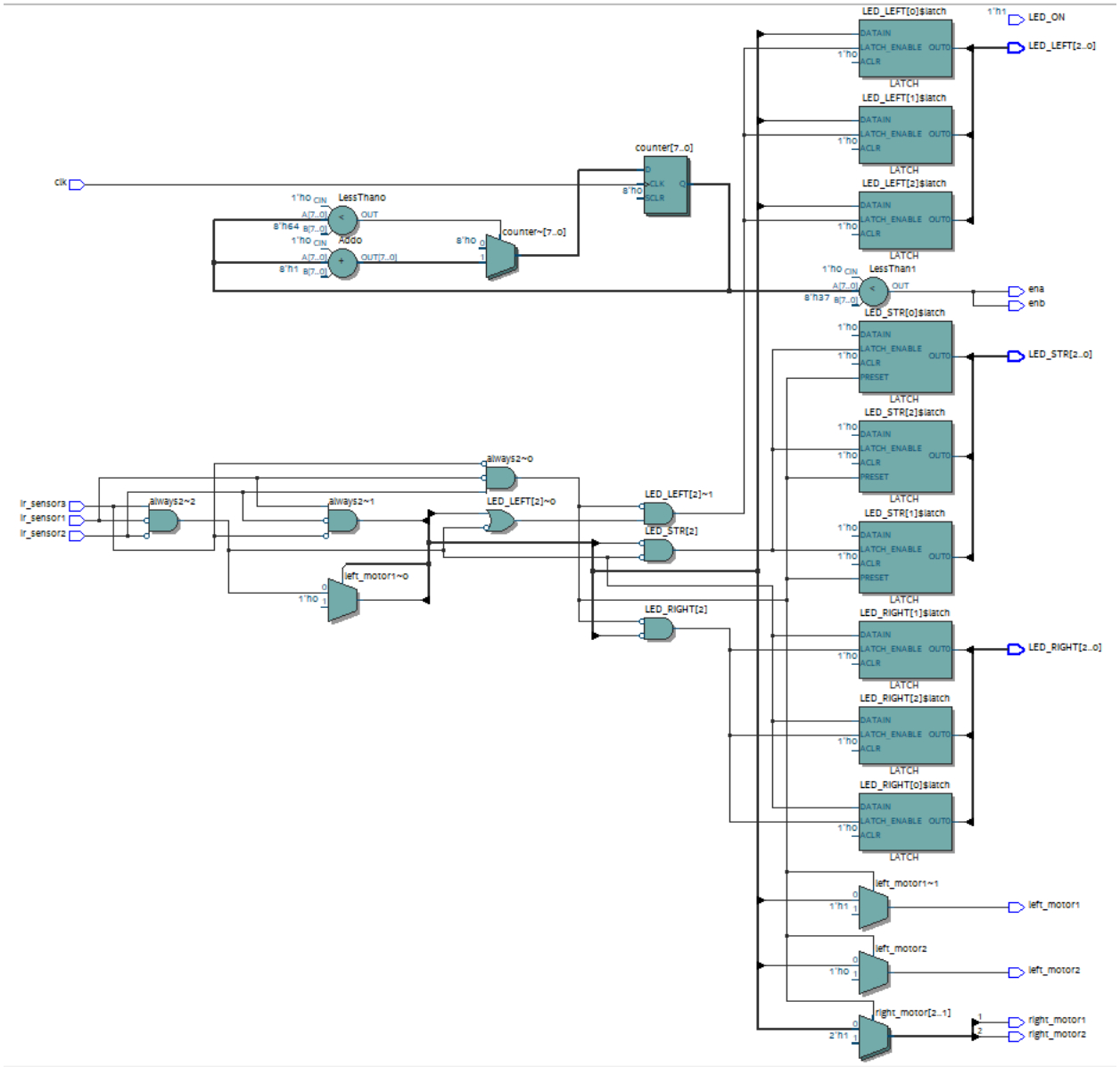


Figure 14: RTL Viewer of Verilog code for Robot

Troubleshooting

When building this robot there were some major hurdles to overcome. The first hurdle was getting the board to work on our devices. Although Terasic is now owned by Intel and carries Intel chips, the board still had to overcome security checks based on the OS being used. Since most Quartus only runs on Windows and Microsoft has been making security a #1 priority

on Windows 11 due to the AI capabilities of CoPilot, the board was not recognized in the device drivers. The first instructions online were to download the USB-Blaster drivers and see if it would work. Once those drivers were installed the board was recognized in the device manager, but still had a warning symbol and no connection was established. We then read that on Windows 11, memory integrity had to be turned off to use the board. This opened the devices up to more risks, but finally allowed us to use the board. Although we had to use 3 devices as 1 laptop did work after turning off memory integrity, but the other did not so in the end a device running Windows 10 worked a lot easier just after downloading the USB drivers.

Another hurdle to overcome was the board itself having pins that came dead on arrival (DOA). Since these pins were DOA when we tried assigning them, Quartus still allowed these pins to be assigned and when connecting everything to them the components on the pins surrounding it would act differently than what the code was instructing it to. We then sat and tested every individual GPIO pin on the board to confirm which pins were DOA and which ones were usable, we then marked down which were unusable and avoided those pins.

A smaller bump to overcome was some components malfunctioning or damaging as we used them. The first infrared sensor we tested was DOA and was extremely hot to touch as it had been shorted. We thought our code was messed up, but it was the component itself that didn't allow for us to have success on our initial trials. Another issue were the screws on the motor drivers stripped really easily which caused us to have to swap drivers 2 times before we could finish our robot. Lastly, our caster wheel or weight distribution made the robot curve to the right instead of fully straight so we needed to ensure our sensors were far enough apart to account for the drift off the track and correct it as needed.

Conclusion

Although the project was new to both of us and came with many challenges, coding a line-following robot using FPGA and Verilog helps further our understanding of Verilog. This project was also very exciting to work on due to the nature of being able to build a robot for the first time and having the space to explore it with an open ended project. Seeing Verilog being used in a more physical application further enforced our likeness to Verilog and makes it seem like a more viable career option in the future.

References

Amazon.com: AEDIKO 8pcs TT Motor Dual DC 3-6V Gearbox Motor 200RPM Ratio 1:48 Shaft Motor with 2.54mm Wire for Arduino DIY Smart Car Robot : Toys & Games. (2024). Amazon.com. https://www.amazon.com/AEDIKO-Motor-Gearbox-200RPM-Ratio/dp/B09N6NXP4H *priority_high* Webpage author

Amazon.com: The perseids DIY Robot Smart Car Chassis Kit with Speed Encoder, 2 Wheels and and Battery Box for Arduino/Microbit/Raspberry Pi for Adult Age 16+ (2 Wheels) : Toys & Games. (2024). Amazon.com. https://www.amazon.com/gp/product/B07DNYQ3PX/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&th=1 *priority_high* Webpage author

Das, D. (2022, March 16). *Interfacing IR Sensor Module with Arduino*. Circuitdigest.com. https://circuitdigest.com/microcontroller-projects/interfacing-ir-sensor-module-with-arduino

Technologies, T. (n.d.). *Terasic - DE Boards - MAX - DE10-Lite Board*. Www.terasic.com.tw. https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=234&No=1021 *priority_high* Date published

Thingiverse.com. (2024). *LM2596 DC-DC Voltage Regulator with 7 segment LED display by isakh1*. Thingiverse. https://www.thingiverse.com/thing:4894286

Xukyo. (2022, April 11). *Using an L298N module with Arduino • AranaCorp*. AranaCorp. https://www.aranacorp.com/en/using-an-l298n-module-with-arduino/