**DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING**

**COLLEGE OF E&ME, NUST, RAWALPINDI**

**Computer System Architecture**

**Project Report**

**FPGA BASED CAR**

**(VARIABLE SPEED + LINE FOLLOWING + OBSTACLE DETECTION)**

**SUBMITTED TO:**

**Dr. Muhammad Yasin**

**LE Abu Bakar**

**SUBMITTED BY:**

**Muhammad Zakria Mehmood 391449**

**Faizan Ejaz 384102**

**Submission Date:**

7/6/2023

# Introduction

The FPGA-based car project aims to develop a semi-autonomous vehicle with specific features. The project involves incorporating three IR sensors, two for line detection and one for obstacle detection. The car is powered by a battery and utilizes an L298N motor driver to control two gear motors. The car structure consists of a base and extension acrylic plate to accommodate the FPGA board.

The primary objective of this project is to take a microcontroller-based project and transform it into an FPGA-based implementation. The motivation for this approach was inspired by the semi-autonomous rover project undertaken in Project Lab I at the Electrical and Computer Engineering Department of Texas Tech University during the summer of 2017. The original project was programmed using the Xilinx Vivado environment for their Basys3 FPGA board.

The problem statement for this project involves designing the FPGA-based car without relying on any pre-existing libraries or modules. Instead, the project aims to develop dedicated Verilog modules to address specific functionalities. One significant aspect is designing a module that generates PWM signals with different duty cycles to provide multiple speed options for the car.

The main goal of the project is to assemble all the hardware components and deliver a fully functional FPGA-based car within the specified deadline. This includes ensuring that the car successfully detects lines and obstacles while implementing the variable speed control feature.

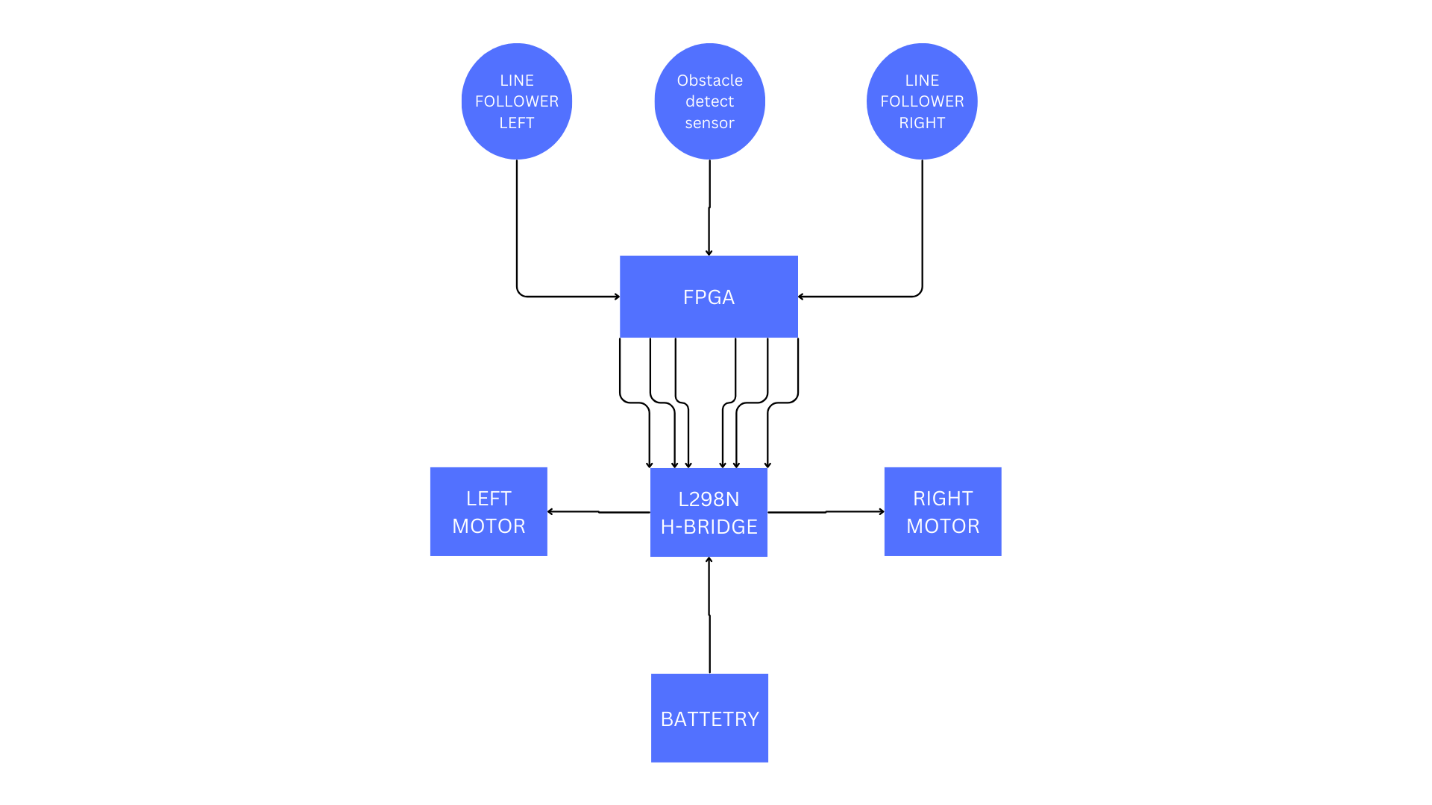
By undertaking this project, the group aims to enhance their understanding of FPGA design, Verilog programming, and hardware integration while also exploring the potential advantages of FPGA-based implementations in the field of autonomous vehicles.

# Methodology

## System Architecture:

The overall system architecture of the FPGA-based car consists of several hardware components. These include:

* L298N H-bridge dual-channel motor driver: This motor driver module is responsible for controlling the direction and speed of the motors.
* Two gear motors: These motors are used to drive the wheels of the car.
* Batteries: The car is powered by batteries to supply the necessary electrical power.
* Three IR sensors: These sensors are utilized for line detection and obstacle detection.
* Spartan 6 FPGA board: The FPGA board serves as the core component for implementing the control and decision-making logic of the car.



## Techniques:

### Speed Control:

To achieve variable speed control, a counter is created that increments on each positive edge of the clock signal. The duty cycle of the PWM signal is determined based on the selected speed control option (1-3), where 1 represents slow and 3 represents fast. Additionally, the duty cycle is adjusted depending on the motion of the car (forward/turn). In the forward motion, the duty cycle is set to 50%, while in turning maneuvers, it is set to 75% to achieve better turning effects. The duty cycle is then divided by the speed control factor (1, 1.5, or 2) corresponding to the selected speed option. The PWM generator module generates the PWM signal, where the output is 1 during the period when the counter is less than or equal to the duty cycle, and 0 for the remaining time. The generated PWM signal is then assigned to the right and left motor speed output wires, which are connected to the enable A and enable B pins of the motor driver.

### Direction Control:

The direction control algorithm utilizes the line detection IR sensors. When a change is detected in the left sensor, a left turn is executed. Similarly, a right turn is executed when a change is detected in the right sensor. If both sensors indicate a line (both 0), it indicates no turn, and forward motion is executed. The resulting direction signals (LB\_direct1, LB\_direct2, RB\_direct1, and RB\_direct2) are fed into the four direction input pins (1-4) of the L298N motor driver, which controls the motor rotation to achieve the desired direction.

A picture containing diagram, screenshot, line, electric blue

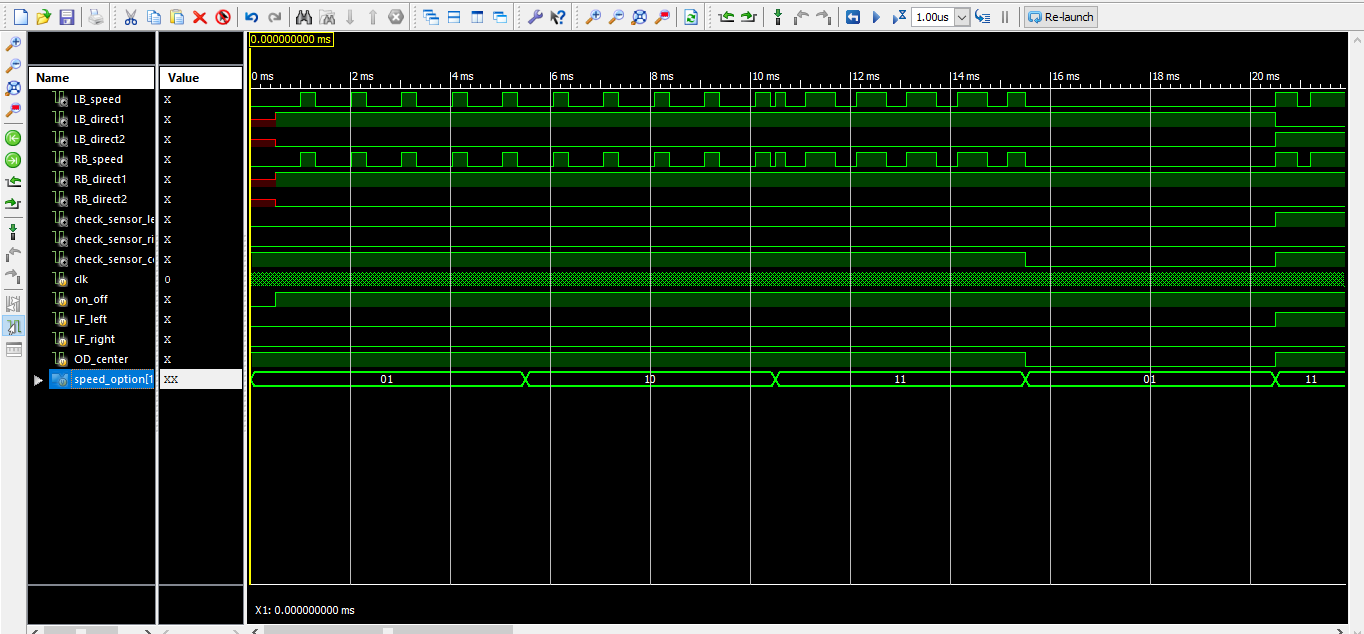
Description automatically generated

By implementing these techniques and algorithms, the FPGA-based car achieves the functionality of line following, obstacle detection, and variable speed control. The design considerations and algorithm implementations are customized to suit the project requirements and are fine-tuned for optimal performance.

# Experimental Setup and Results

## Experimental Setup:

The experimental setup involved testing the FPGA-based car in both simulation and on the FPGA board. For simulation, the code was run using Xilinx to observe the expected behavior. Test bench includes all the possible inputs of sensors and speed controls.



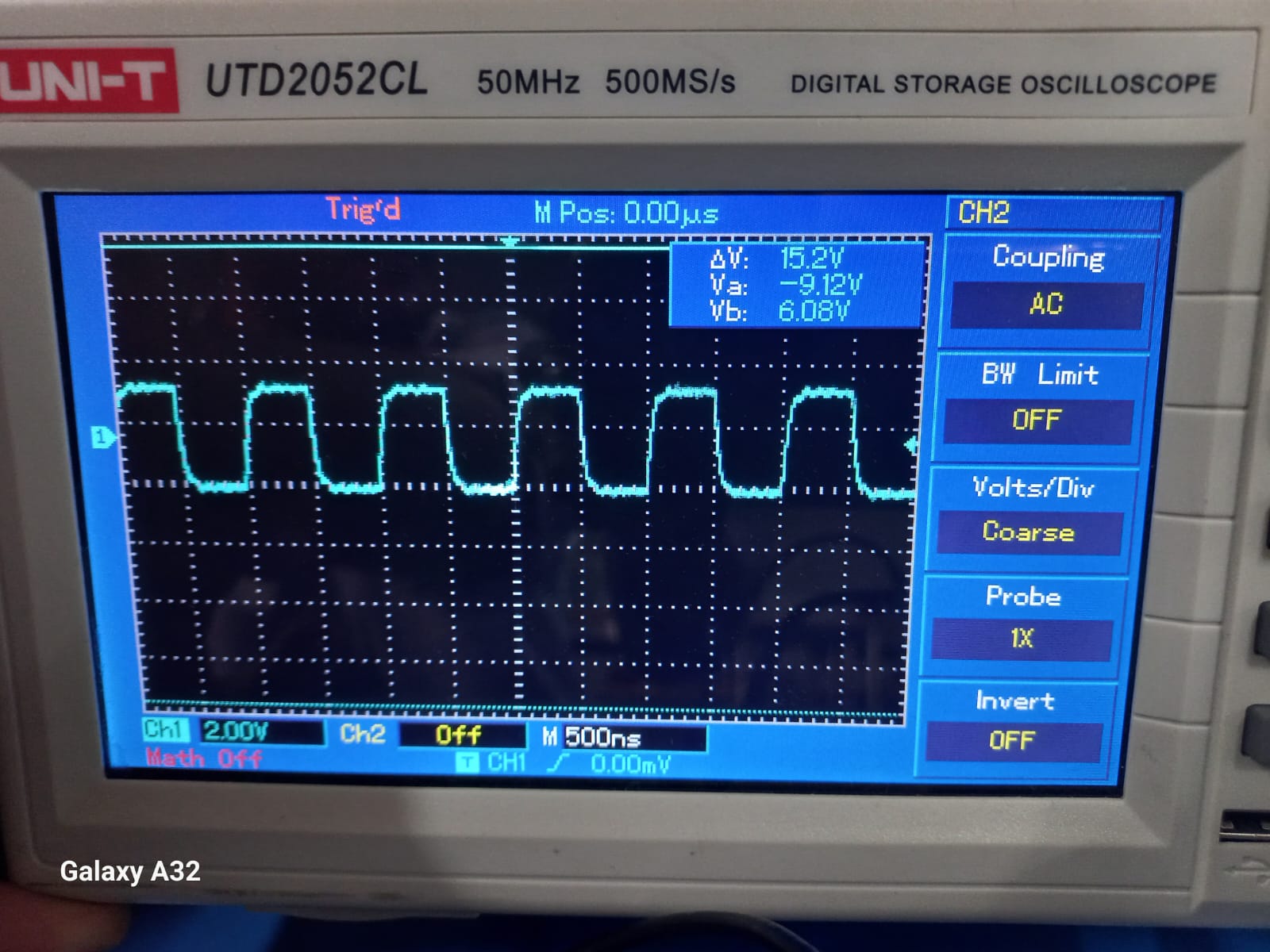
A screenshot of a computer

Description automatically generated

## Results:

In the physical testing on the FPGA board, manual inputs were provided instead of using sensors to simulate different scenarios. The oscilloscope was used to observe the output PWM signals and verify the functionality of the direction algorithm. The results of the experiments demonstrated the successful implementation of the FPGA-based car. The PWM signals were generated accurately, and the direction algorithm effectively controlled the motor rotation to achieve the desired turns. The oscilloscope observations provided visual confirmation of the expected behavior.

MEDIUM SPEED:



SLOW SPEED:

A picture containing text, display device, monitor, screenshot

Description automatically generated

FAST SPEED:

A picture containing text, display device, electronics, screenshot

Description automatically generated

## Analysis and Interpretation:

The FPGA-based car exhibited strengths and limitations in its performance. One of the strengths was its ability to execute sharp turns by utilizing motor direction control instead of traditional steering mechanisms. This allowed for greater maneuverability and flexibility in navigating tight corners.

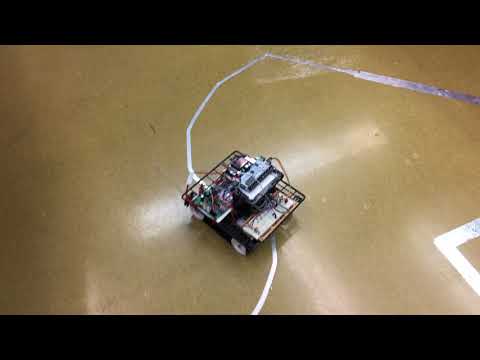
However, a limitation of the system was identified regarding the sensitivity of the FPGA board to input voltage. Due to this sensitivity, the car could not be made completely mobile by relying solely on battery power. The FPGA board required a highly regulated and precise voltage supply to avoid any potential damage. This limitation prevented the car from operating independently without the USB cord connected to the laptop and the FPGA board.

To overcome this limitation, a highly regulated voltage regulator would be necessary to regulate the voltage from the battery and ensure it matches the required voltage of the FPGA (which is 5 volts for the Nexys 3 Spartan 6 board).

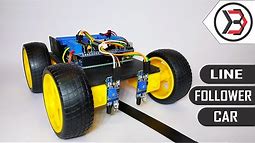
Overall, the experimental results demonstrated the successful implementation of the FPGA-based car, showcasing its capabilities in line following, obstacle detection, and variable speed control. The strengths in maneuverability and the identified limitation regarding voltage sensitivity provide valuable insights for further improvement and future iterations of the project.

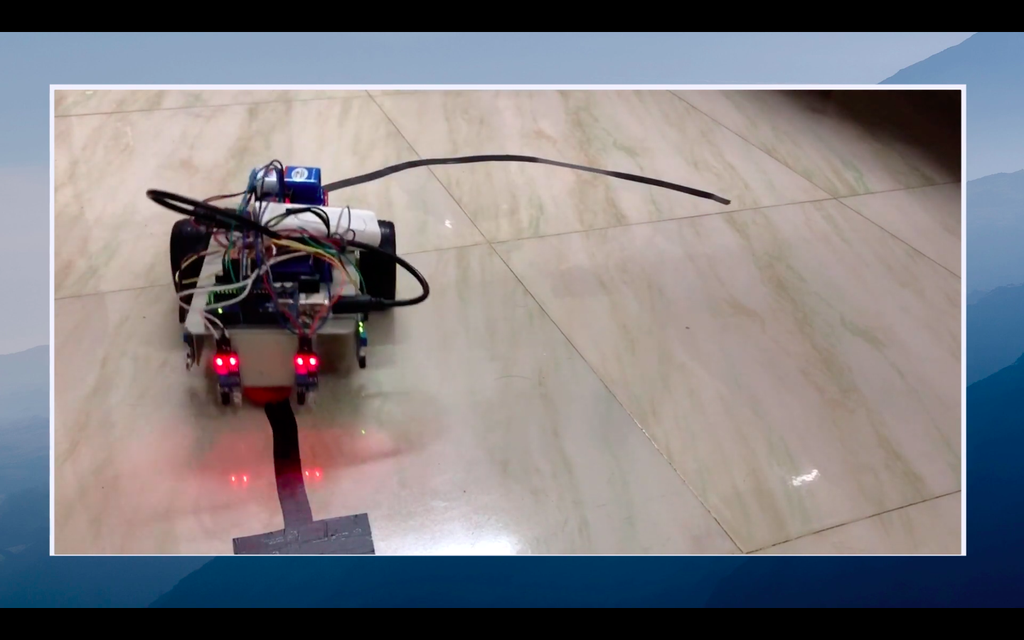
# References

1. Alexandre Soares Silva. (2017). FPGA-based Rover - Verilog. [Online]. Available at: https://github.com/alexandresoaresilva/FPGA-based-rover.Verilog [Accessed 19th April 2023].



1. DIY King. (2019, July 3). How To Make A DIY Arduino Line Follower Car At Home [Video]. YouTube. Available at: https://www.youtube.com/watch?v=t7k9D1jDEtk [Accessed May, 2nd 2023].





# Appendices

## Base module:

`timescale 1ms / 1us

module Base(

input clk,

input [1:0] speed\_option,

input on\_off,

input LF\_left,

input LF\_right,

input OD\_center,

output reg val,

output LB\_speed,

output LB\_direct1,

output LB\_direct2,

output RB\_speed,

output RB\_direct1,

output RB\_direct2,

output check\_sensor\_left,

output check\_sensor\_right,

output check\_sensor\_center

);

wire brake;

assign brake = (!OD\_center || !on\_off) ?1:0;

reg left\_row, right\_row;

reg [1:0] speed;

wire [1:0] speedcontrol;

assign speedcontrol = speed;

always@(posedge clk)begin

val = clk;

if(!brake)begin

// Forward movement

if(!LF\_left && !LF\_right)begin // both sensor output zero

speed <= 1;

left\_row <= 1; //motor spin forward direction

right\_row <= 1; //motor spin forward direction

end

// Left turn

else if(LF\_left && !LF\_right)begin // left = 1 & righ = 0

speed <= 2;

left\_row <= 0; //motor spin backward direction

right\_row <= 1; //motor spin forward direction

end

// Right turn

else if(!LF\_left && LF\_right)begin // left = 0 & right = 1

speed <= 2;

left\_row <= 1; //motor spin forward direction

right\_row <= 0; //motor spin backward direction

end

end

else begin

// obstacle infront! STOP

speed <= 0;

end

end

wire pwm;

PWM\_GENERATOR pulse(

.clk(clk),

.option(speed\_option),

.speedControl(speedcontrol),

.pwm(pwm)

);

assign LB\_speed = pwm;

assign RB\_speed = pwm;

assign LB\_direct1 = left\_row;

assign LB\_direct2 = !left\_row;

assign RB\_direct1 = right\_row;

assign RB\_direct2 = !right\_row;

assign check\_sensor\_left = LF\_left;

assign check\_sensor\_right = LF\_right;

assign check\_sensor\_center = OD\_center;

endmodule

PWM Generator:

`timescale 1ms / 1us

module PWM\_GENERATOR(

input clk,

input [1:0] option,

input [1:0] speedControl,

output pwm

);

reg [7:0] counter;

reg [7:0] duty\_cycle;

reg [1:0] factor;

always@(speedControl or option)begin

case(option)

2'b00: factor = 0; // stop

2'b11: factor = 1; // full speed

2'b10: factor = 1.5; // medium speed

2'b01: factor = 2; // half speed

default: factor = 0; // stop

endcase

case(speedControl)

2'b00: duty\_cycle = 0; // stop

2'b01: duty\_cycle = 60; // duty cycle 50%

2'b10: duty\_cycle = 75; // duty cycle 75%

default: duty\_cycle = 0;

endcase

end

always@(posedge clk)begin

if(counter < 100)begin

counter <= counter + 1;

end

else begin

counter <= 0;

end

end

assign pwm = (counter < (duty\_cycle / factor)) ? 1:0;

endmodule

Constraints file:

NET clk LOC = V10;

NET val LOC = T11;

NET RB\_direct1 LOC = K2;

NET RB\_direct2 LOC = K1;

NET RB\_speed LOC = L4;

NET LB\_direct1 LOC = T12;

NET LB\_direct2 LOC = V12;

NET LB\_speed LOC = N10;

NET speed\_option[1] LOC = T5;

NET speed\_option[0] LOC = V8;

NET on\_off LOC = U8;

NET LF\_left LOC = M10;

NET LF\_right LOC = N9;

NET OD\_center LOC = U11;

NET check\_sensor\_left LOC = R11;

NET check\_sensor\_right LOC = N11;

NET check\_sensor\_center LOC = M11;

## Test bench:

`timescale 1us / 1ns

module test\_base;

// Inputs

reg clk;

reg on\_off;

reg LF\_left;

reg LF\_right;

reg OD\_center;

reg [1:0] speed\_option;

// Outputs

// LEFT MOTOR

wire LB\_speed;

wire LB\_direct1;

wire LB\_direct2;

// RIGHT MOTOR

wire RB\_speed;

wire RB\_direct1;

wire RB\_direct2;

// CHECK LIGHTS ON-BOARD

wire check\_sensor\_left;

wire check\_sensor\_right;

wire check\_sensor\_center;

// Instantiate the Unit Under Test (UUT)

Base uut (

.clk(clk),

.speed\_option(speed\_option),

.on\_off(on\_off),

.LF\_left(LF\_left),

.LF\_right(LF\_right),

.OD\_center(OD\_center),

.LB\_speed(LB\_speed),

.LB\_direct1(LB\_direct1),

.LB\_direct2(LB\_direct2),

.RB\_speed(RB\_speed),

.RB\_direct1(RB\_direct1),

.RB\_direct2(RB\_direct2),

.check\_sensor\_left(check\_sensor\_left),

.check\_sensor\_right(check\_sensor\_right),

.check\_sensor\_center(check\_sensor\_center)

);

initial begin

clk = 0; #5;

// FORWARD NO OBSTACLE BUT SWITCH OFF speed select = 1 (slowest)

speed\_option = 1;

on\_off = 0;

LF\_left = 0;

LF\_right = 0;

OD\_center = 1;

#500;

// FORWARD NO OBSTACLE speed select = 1 (slowest)

speed\_option = 1;

on\_off = 1;

LF\_left = 0;

LF\_right = 0;

OD\_center = 1;

#5000;

// FORWARD NO OBSTACLE speed select = 2 (medium)

speed\_option = 2;

on\_off = 1;

LF\_left = 0;

LF\_right = 0;

OD\_center = 1;

#5000;

// FORWARD NO OBSTACLE speed select = 3 (fastest)

speed\_option = 3;

on\_off = 1;

LF\_left = 0;

LF\_right = 0;

OD\_center = 1;

#5000;

// FORWARD OBSTACLE speed select = 1 (slowest)

speed\_option = 1;

on\_off = 1;

LF\_left = 0;

LF\_right = 0;

OD\_center = 0;

#5000;

// LEFT NO OBSTACLE speed select = 3 (fastest)

speed\_option = 3;

on\_off = 1;

LF\_left = 1;

LF\_right = 0;

OD\_center = 1;

#5000;

// RIGHT NO OBSTACLE speed select = 3 (fastest)

speed\_option = 3;

on\_off = 1;

LF\_left = 0;

LF\_right = 1;

OD\_center = 1;

#5000;

end

always#5 clk=~clk;

endmodule