ECE 271 Final Project

Anthony Grana
Felipe Orrico Scognamiglio

December 2019

Contents

1 Introduction

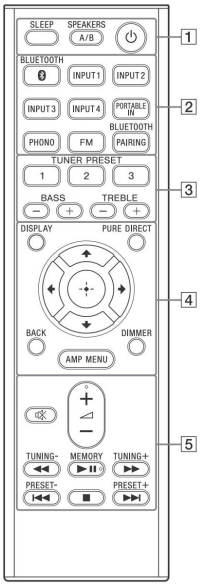


Figure 1: Sony RMT-AA400U Remote @Sony.com

This project is the condensation of all knowledge gathered over this term about FPGAs and HDLs, where the application is the implementation of a digital logic design that will allow us to remotely control a robot (speed, and direction of movement) through the use of a VCR remote.
With the use of basic logical components (such as AND, OR and NOT gates) put together to create larger logic blocks that enable the use of more advanced logic, and memory-based systems. Decoders, shift registers, and IR receivers allow the reception, storage, and decoding, and interpretation of the VCR remote's signals. This information is sent to decoders that will relay the commands to DC motors to control the robot.
Infrared is a cheap and robust standard that has existed in robotics for decades. It is commonly used for toy robots intended for indoor use where cost and manufacturability are a priority. The main cons of infrared are its limited range, its

lack of object penetration, and its vulnerability to noise. This makes infrared a poor standard with regards to reliability.

1.1 Project Deliverables

- Supports IR communication protocol from Sony RMT-AA400U remote.
- Can control DC motors with commands from the remote.
- Beep at a tone depending on the current speed of the robot.
- Display Signal data and speed on seven segment displays.

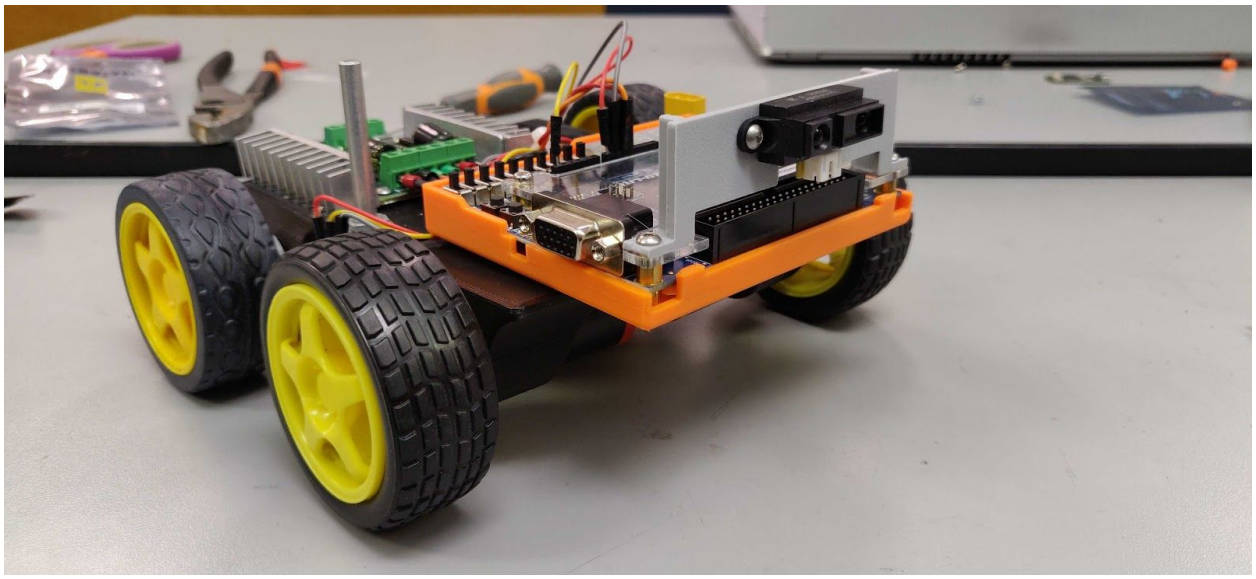


Figure 2: ECE271 Final Project Robot “Junkbot V3.0”

2 High-Level Description

2.1 Top-Level Hardware Diagram

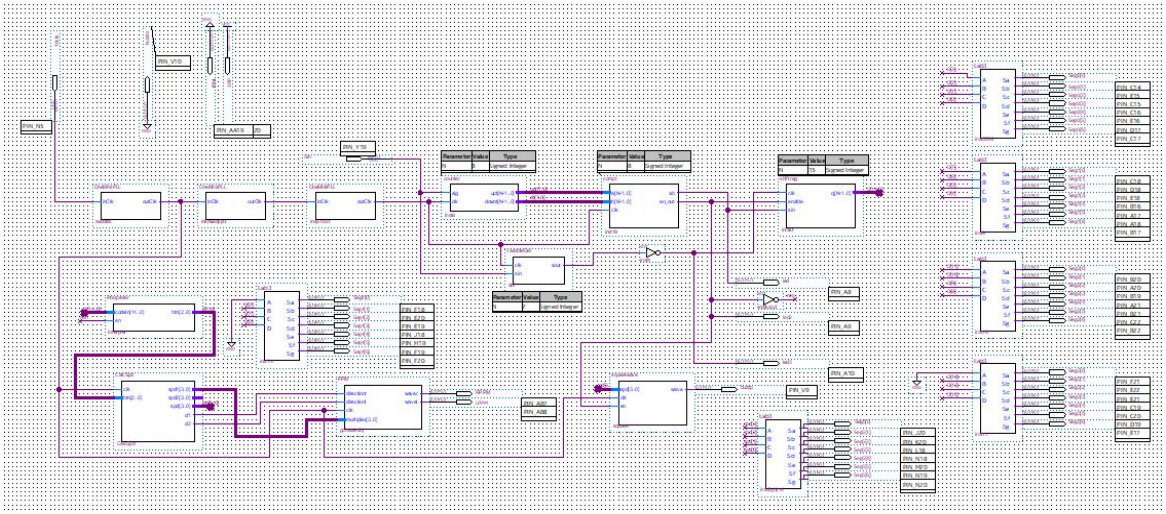


Figure 3: High Level Block Diagram

Inputs:

- VCR remote

Outputs:

- Output signal to control the motors (PPM protocol)
- 7-Segment Displays
- Square Wave buzzer

Connections:

- The IR receiver is connected to the IO pins PIN_Y9 of the FPGA, this device will receive the signal sent by the VCR remote and relay it to the data pin. The DC motor controller is connected to two of the IO pins PIN_AB7 and

PIN_AB8. Each of the pins control the direction and speed of the motors of their respective side. The Buzzer is connected to the IO pins PIN_V9 and PIN_V10 and is activated by a square wave generator tied to the enable of a shift register.

Description:

- The IR receiver receives the information sent by the VCR remote and passes that into a serial bus in the FPGA. After the FPGA receives the serial input, it passes it through a decoder and a shift register that will decode and store the command. After that command is stored, it is passed to another block using a parallel bus that determines which button was pushed, and from that, it sends the information to the DC motor decoder that will decode that input and send the command to the motors. The buzzer gets activated by any button push of the remote and outputs a square wave that has its pitch dependent on the current speed.

3 Controller Description

3.1 VCR Remote

The VCR remote communicates button pushes using the SIRC protocol. This protocol defines the behavior of the remote, how it sends the information in a serial format. The decoder knows the signal has started when active period of 2.5ms happens, after that, the remote sends active signals of 0.5-0.7ms that are considered as logic zeros or 1.2ms that are considered as a logic one, with an inactive time of 0.5ms between each signal.

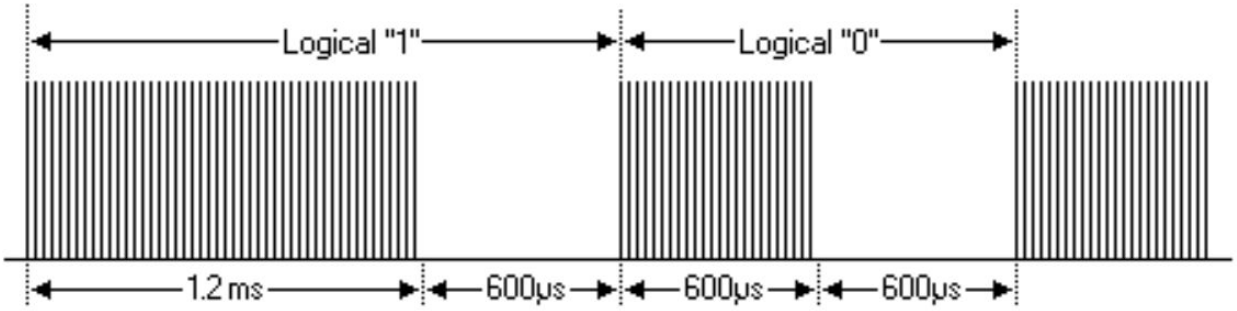


Figure 2: SIRC logic modulation @SB-Projects

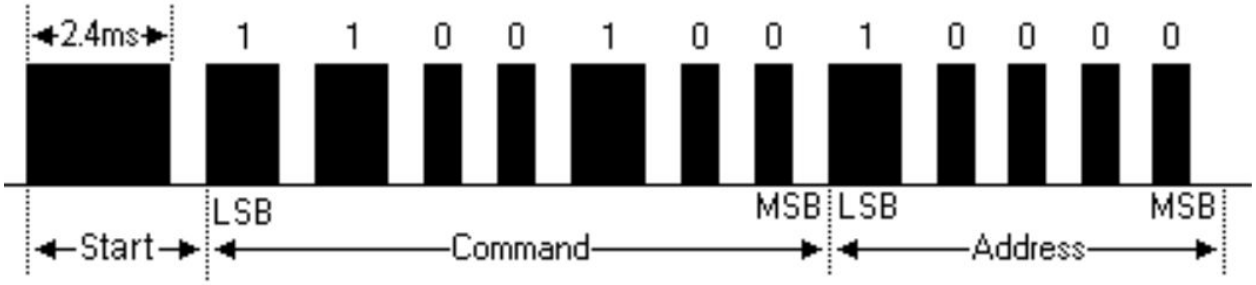


Figure 3: SIRC Protocol @SB-Projects

4 HDL Components

4.1 Top Module Components

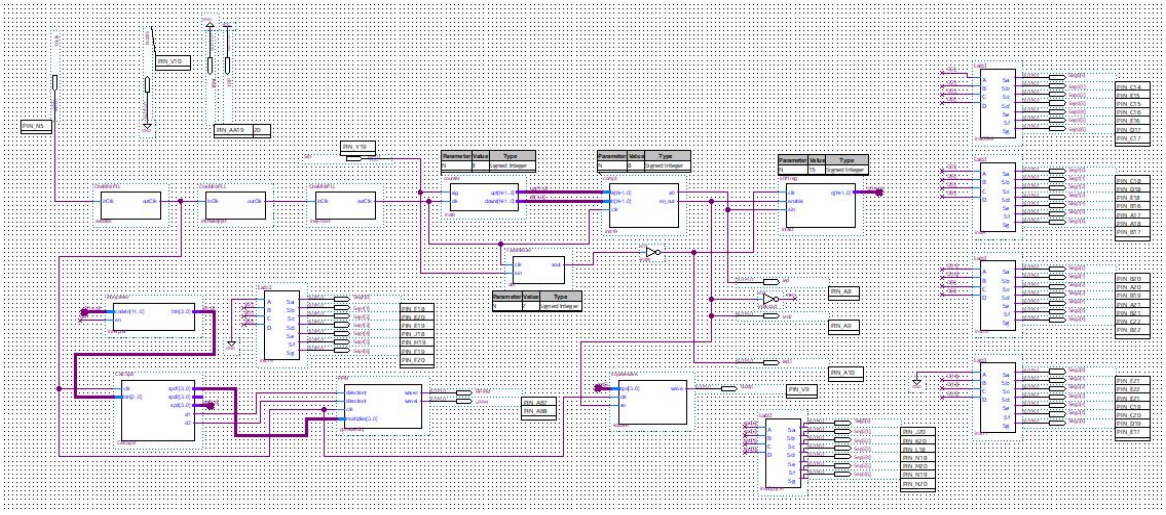


Figure 4: Top Model for the Entire Project.

Input: VCR remote.

Outputs:

- HEX code of the pressed VCR remote button to the Seven Segment Display decoders through 16bit bus.
- The commands for the motor controller module.
- The motor controller module outputs a PPM signal wave.
- The square wave signal to the buzzer.
- The Current Speed setting to a Seven Segment Display.

4.1.1 7-Segment Display Decoder Module (lab3)

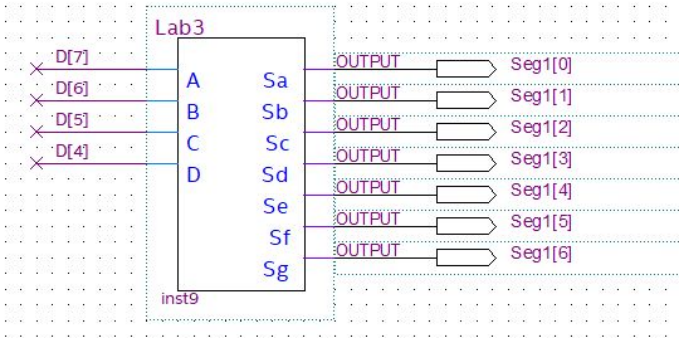


Figure 5: 7-Segment Display Decoder Module

Inputs: A,B,C,D specifies the 4 bits that constitute the number that will be displayed (also includes HEX values).

Outputs: Sa, Sb, Sc, Sd, Se, Sf, Sg specify each of the 7 segments of the display.

Description: This module will receive a 4 bit binary input (number) and output the corresponding character in Hexadecimal to the desired display.

4.1.2 15-bit Shift Register (Serial-in-Parallel-out) Module (shiftreg)

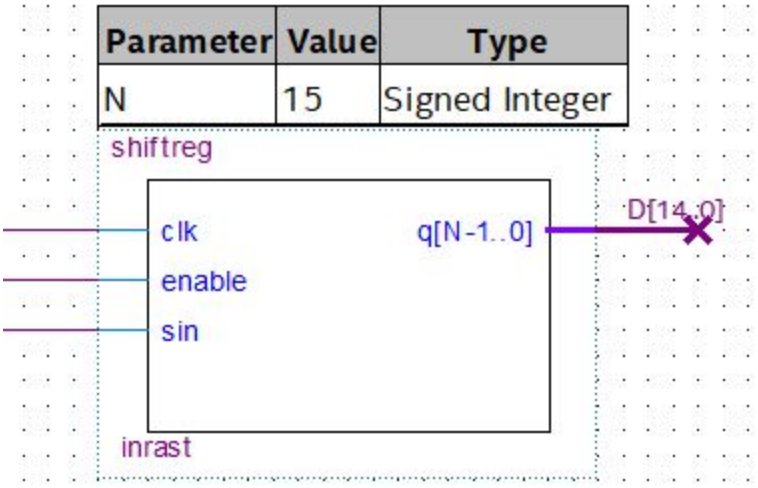


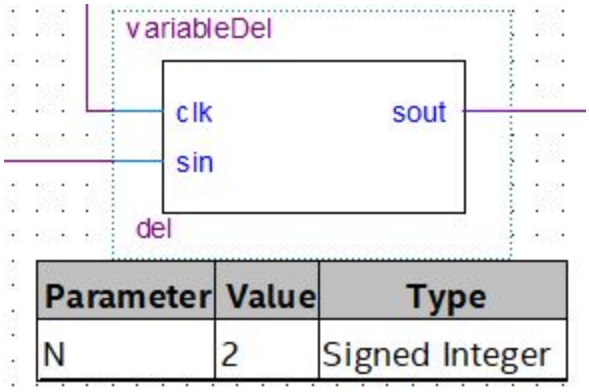
Figure 6: 15-Bit Shift Register for the IR Interpretation Module

Inputs: clk, enable, sin. Specify the clock input (inverted remote signal, with a 200μs time shift); the enable input, that enables the shifting feature in the register; the sin (serial input) input will input linearly into each register.

Outputs: q specifies the parallel output of the register as a 15-bit bus.

Description: This 15-bit shift register, receives the VCR remote input and shifts each bit by 1 at every clock cycle. Then, when enable is turned off, it holds that value until a new command is received.

4.1.3 variableDel Module



Parameter	Value	Type
N	2	Signed Integer

Figure 7: Variable Delay Module Block

Inputs: clk, sin. Clk specifies the clock input (10KHz) and sin (serial in) specifies the serial input of the shift register.

Outputs: sout. It specifies the serial output of the valiableDel block.

Description: This block is a shift register that delays the input by the parameter assigned to it. In this case, the input is delayed by 2 clock cycles, so that there is time for all blocks to process the information before they have to read the remote signal.

4.1.4 1MHzPLL Module

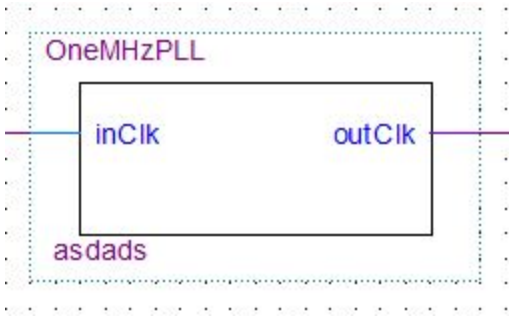


Figure 8: 1MHzPPL (clock divider) Module Block

Inputs: inClk, defines the clock (wave) input to the module.

Outputs: outClk, defines the clock (wave) output of the module.

Description: This is a divide by 10 clock module where a 10MHz clock is inputted and a 1MHz is outputted.

4.1.5 Counter Module

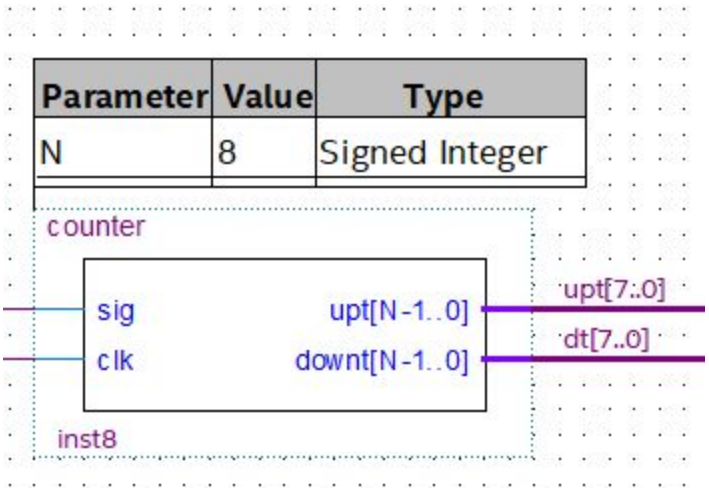


Figure 9: Counter Module Block

Inputs: sig, clk. Sig defines the VCR remote signal input to the counter. Clk defines the clock (10KHz) input to the module.

Outputs: upt, downt. Upt defines the time of the positive cycle (positive edge to negative edge) of the signal. Downt defines the time of the negative cycle (negative edge to positive edge) of the signal.

Description: This block counts how long the up time and down time of a bit is and outputs that on a 8-bit bus.

4.1.6 compt Module

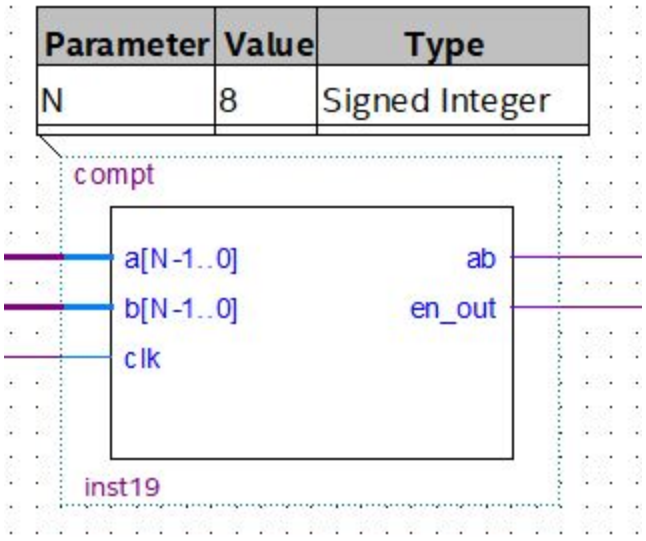


Figure 10: Comparator Module Clock

Inputs: a, b , clk. A is an input that reads an 8-bit number. B is an input that reads and 8-bit number. Clk is the clock input (1MHz).

Outputs: ab, en_out. Ab is defined by (a > 1.1ms). En_out is an output that outputs a 1 or a zero depending on the state of the block.

Description: This block is a comparator and an FSM where the block outputs an enable signal when the FMS is on state 2 (Packet currently being received) and a one or a zero to ab when a > 1.1ms.

4.1.7 Interpreter Module

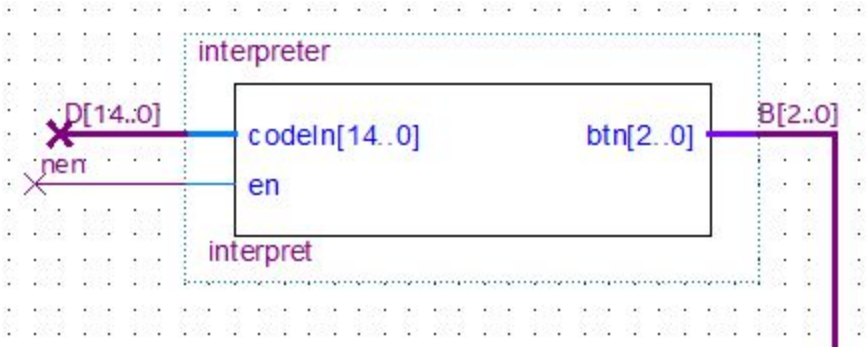


Figure 11: Interpreter Module Block

Inputs: codeIn, en. CodeIn is the input for the parallel signal from the VCR remote. En is an enable to convert that number and send to the output.

Outputs: btn outputs a binary number 0-6_{10} based on input.

Description: This module converts the 15-bit input into a 3-bit output based on HEX values.

4.1.8 CalcSpeed Module

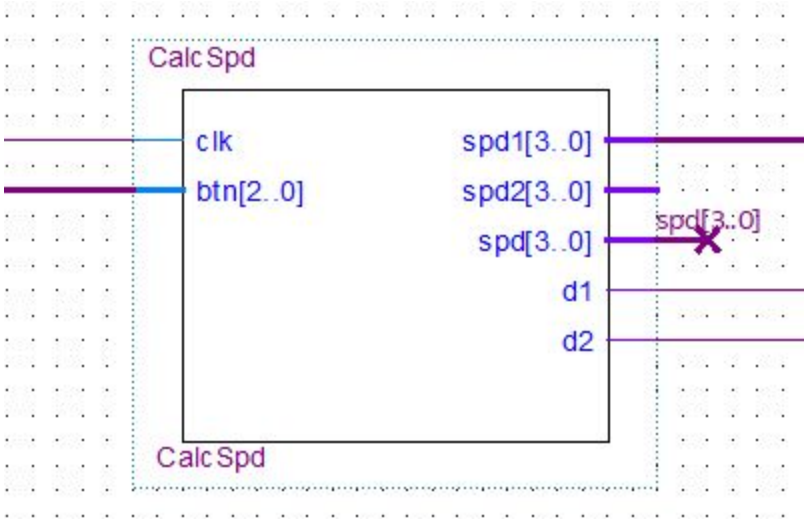


Figure 12: CalcSpeed Module Block

Inputs: clk, btn. Clk is defined by a 1MHz clock. Btn is a 3-bit value between 1-6_{10}.

Outputs: spd1, spd2, spd, d1, d2. D1 is the direction of the right side wheels, D2 is the direction of the left side wheels, spd is the speed of the square wave. Spd1 and Spd2 are the same speed value but for different side wheels.

Description: This block receives a clock and a button input and outputs the speed, and direction for both sides of the robot and the speed of the square wave.

4.1.9 PPM Module

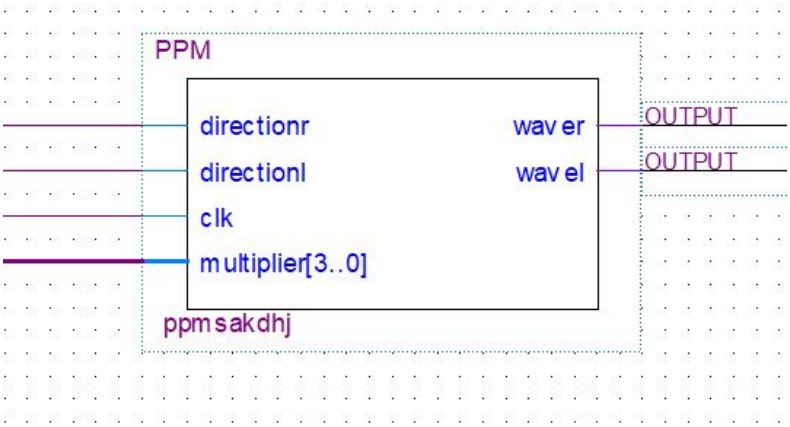


Figure 13: PPM (Motor decoder) Module Block

Inputs: directionr, directionl, clk, multiplier. Directionl and Directionr are the directions of the left side and right side of the robot. Clk is a 1MHz clock signal and multiplier is the speed multiplier for both sides.

Outputs: waver, wavel. Those outputs define the PPM wave that controls both sets of motors, right and left side.

Description: This block receives the direction and multiplier for both sets of motors in the robot, and creates a PPM wave that will control those motors.

4.1.10 Square Wave Module

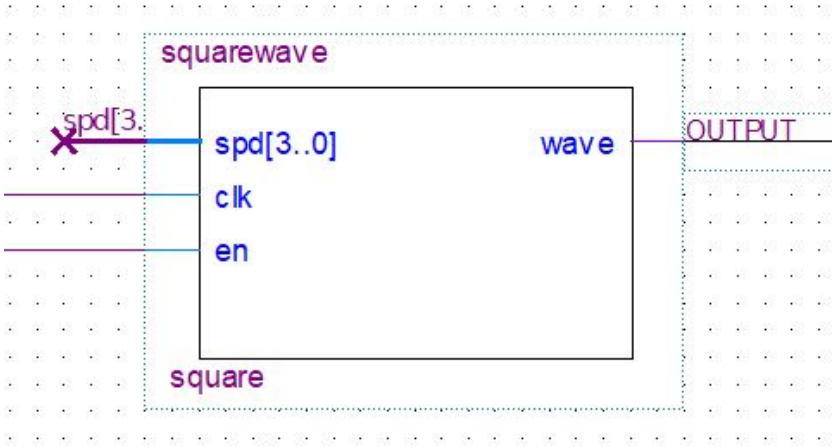


Figure 14: Square Wave Module Block

Inputs: spd, clk, en. Spd is the speed multiplier for the wave (used for frequency modulation). Clk is the clock input, and en is the input to enable the creation of the square wave.

Outputs: wave. Outputs the square wave generated.

Description: This block will receive a multiplier and use that to generate a wave with a specific frequency.

5 Appendix

5.1 Design Synthesis and Analysis

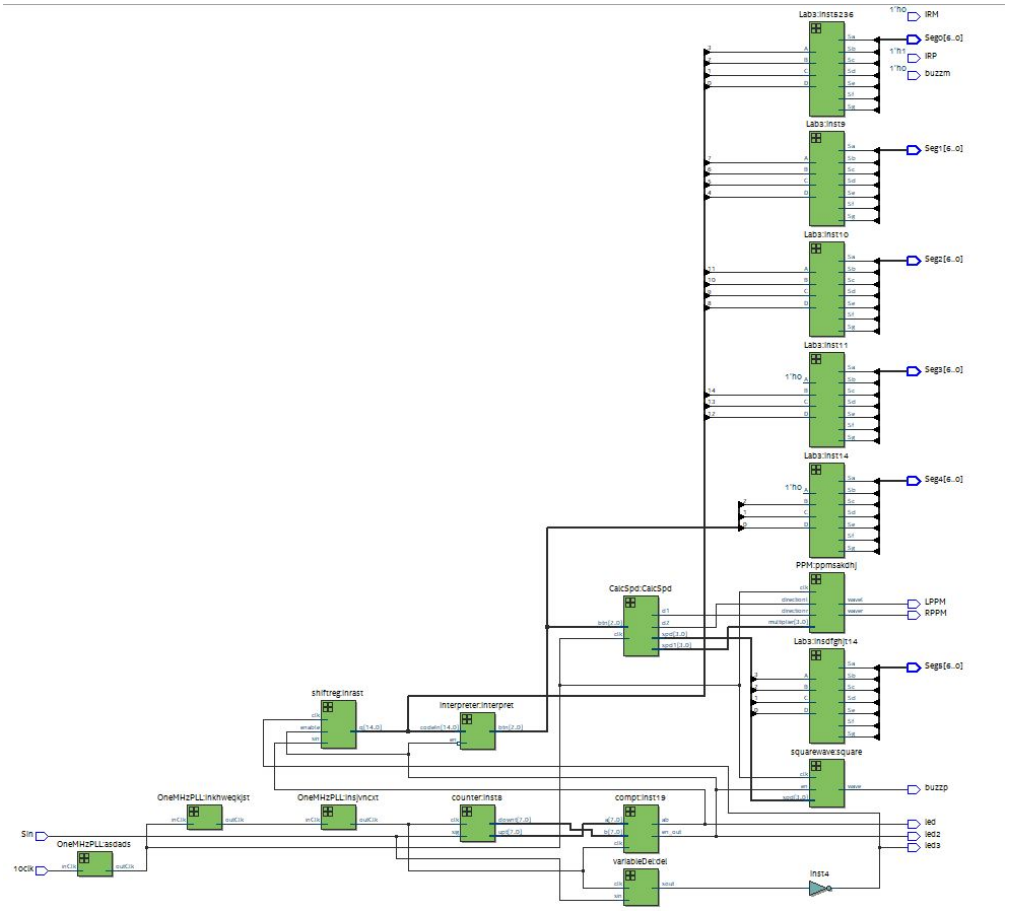


Figure 15: Design Synthesis By RTL Viewer

5.2 Source Code

5.2.1 7-Segment Display Decoder Module (lab3)

```
module Lab3(
   A,
   B,
   C,
   D,
   Sa,
   Sb,
   Sc,
   Sd,
   Se,
   Sf,
   Sg
);


input wire  A;
input wire  B;
input wire  C;
input wire  D;
output wire Sa;
output wire Sb;
output wire Sc;
output wire Sd;
output wire Se;
output wire Sf;
output wire Sg;

wire  An;
wire  Bn;
wire  Cn;
wire  Dn;



assign   An =  ~A;


Sf b2v_inst10(
   .A(A),
   .B(B),
   .C(C),
   .D(D),
   .An(An),
   .Bn(Bn),
   .Cn(Cn),
   .Dn(Dn),
   .Sf(Sf));


Sg b2v_inst11(
   .A(A),
   .B(B),
   .C(C),
   .D(D),
   .An(An),
   .Bn(Bn),
   .Cn(Cn),
   .Dn(Dn),
   .Sg(Sg));

assign   Bn =  ~B;
```

```
assign   Cn =  ~C;

assign   Dn =  ~D;


Sa b2v_inst5(
   .A(A),
   .B(B),
   .C(C),
   .D(D),
   .An(An),
   .Bn(Bn),
   .Cn(Cn),
   .Dn(Dn),
   .Sa(Sa));


Sb b2v_inst6(
   .A(A),
   .B(B),
   .C(C),
   .D(D),
   .An(An),
   .Bn(Bn),
   .Cn(Cn),
   .Dn(Dn),
   .Sb(Sb));


Sc b2v_inst7(
   .A(A),
   .B(B),
   .C(C),
   .D(D),
   .An(An),
   .Bn(Bn),
   .Cn(Cn),
   .Dn(Dn),
   .Sc(Sc));


Sd b2v_inst8(
   .A(A),
   .B(B),
   .C(C),
   .D(D),
   .An(An),
   .Bn(Bn),
   .Cn(Cn),
   .Dn(Dn),
   .Sd(Sd));


Se b2v_inst9(
   .A(A),
   .B(B),
   .C(C),
   .D(D),
   .An(An),
   .Bn(Bn),
   .Cn(Cn),
   .Dn(Dn),
   .Se(Se));
```

Figure 16: 7-Segment Display Module Verilog

5.2.2 15-bit Shift Register (Serial-in-Parallel-out) Module (shiftreg)

```
module shiftreg #(parameter N = 8) (input logic clk,
                                    input logic enable,
                                    input logic sin,
                                    output logic [N-1:0] q);

   always_ff @(posedge clk)
      if (enable) q<={q[N-2:0], sin};

endmodule
```

Figure 17: 15-bit Shift Register Module System Verilog

5.2.3 variableDel Module

```
module variableDel #(parameter N=8) (input logic clk, sin,
                    output logic sout);

                    logic [N-1:0] delay;

      always_ff@(posedge clk) begin
         sout <=delay[N-1];
         delay <= {delay[N-2:0],sin};
      end
endmodule
```

Figure 18: variableDel Module System Verilog

5.2.4 1MHzPLL Module

```
module OneMHzPLL(input logic inClk,
                 output logic outClk);

   logic [3:0]cnt;

   always_ff@(posedge inClk) begin

      cnt++;

      if(cnt == 10) begin
         outClk <= 1;
         cnt <= 0;
      end

      else outClk <= 0;

   end
endmodule
```

Figure 19: 1MHzPPL Module System Verilog

5.2.5 Counter Module

```
module counter #(parameter N = 8) (input logic sig, clk, output logic [N-1:0] upt, downt);

    logic flag;

    always_ff@(posedge clk)
    begin
        if (sig)
            begin
                if (flag)
                    begin
                        flag <= 0;
                        upt = 0;
                    end
                upt++;
                downt<=0;
            end
        else if (!sig)
            begin
                downt<= downt + 1;
                flag <= 1;
            end
    end

endmodule
```

Figure 20: Counter Module System Verilog

5.2.6 compt Module

```
module compt #(parameter N = 8) (input logic [N-1:0]a, input logic [N-1:0]b,input logic clk, output logic ab, output logic en_out);

    //a is uptime counter
    //b is down time counter

    assign ab = (a > 11); //if the uptime is greater than 1.1ms, this is 1

    typedef enum logic [1:0] {S0, S1, S2} statetype;

    statetype state;//might have problem here

    always_ff@(posedge clk)
        begin

            if (b > 8 & state == S2) state <= S0; //reset state
            if (a > 20 & state == S0) state <= S1; //stand by state
            if (b > 0 & state == S1) state <= S2; //data state


        end

    assign en_out=(state==S2);


endmodule
```

Figure 21: compt Module System Verilog

5.2.7 Interpreter Module

```
module interpreter(input logic [14:0] codeIn,
                   input logic en,
                    output logic [2:0] btn);

   always@(codeIn)begin
      if(en)
      case(codeIn)
         15'h0F0D: btn <= 0;
         15'h4F0D: btn <= 1;
         15'h2F0D: btn <= 2;
         15'h6F0D:btn <= 3;
         15'h180C: btn <= 4;
         15'h240C: btn <= 5;
         15'h640C:btn <= 6;
         default: btn <= 4;
      endcase
   end
endmodule
```

Figure 22: Interpreter Module System Verilog

5.2.8 CalcSpeed Module

```
module CalcSpd(input logic clk,
               input logic [2:0]btn,
               output logic [3:0] spd1, spd2, spd,
               output logic d1, d2);

   logic spdflg;

   always_ff@(posedge clk) begin
      case(btn)
         0: begin
            spd1 <= spd;
            spd2 <= spd;
            d1 <= 1;
            d2 <= 1;
            spdflg <= 0;
         end

         1: begin
            spd1 <= spd;
            spd2 <= spd;
            d1 <= 0;
            d2 <= 0;
            spdflg <= 0;
         end

         2: begin
            spd1 <= spd*2;
            spd2 <= spd*2;
            d1 <= 1;
            d2 <= 0;
            spdflg <= 0;
         end

         3: begin
            spd1 <= spd*2;
            spd2 <= spd*2;
            d1 <= 0;
            d2 <= 1;
            spdflg <= 0;
         end

         4: begin
            spd1 <= 0;
            spd2 <= 0;
            d1 <= 1;
            d2 <= 1;
            spdflg <= 0;
         end

         5: begin
            if(~spdflg)begin
               spd++;
               spdflg <= 1;
            end
         end

         6: begin
            if(~spdflg)begin
               spd <= spd - 1;
               spdflg <= 1;
            end
         end
      endcase
   end
```

Figure 23: CalcSpeed Module System Verilog

5.2.9 PPM Module

```
module PPM (input logic directionr, directionl, clk,input logic [3:0] multiplier, output logic waver, wavel);

   //left side
   logic [13:0]countl;
   logic [10:0]high_timel;
   logic [1:0] direction1;

   always_ff@(posedge clk)
   begin
      if (directionl)
      high_timel <= (1500 + (multiplier*62)/2);
      else
      high_timel <= (1500 - (multiplier*62)/2);

   end

   always_ff@(posedge clk)
   begin
      countl++;
      if(countl < high_timel) wavel <=1;
      else if (countl < 10000 + high_timel) wavel<=0;
      else countl <= 0;
   end

   //right side
   logic [13:0]countr;
   logic [10:0]high_timer;
   logic [1:0] direction2;

   always_ff@(posedge clk)
   begin
      if (directionr)
      high_timer <= (1500 + (multiplier*62)/2);
      else
      high_timer <= (1500 - (multiplier*62)/2);
   end

   always_ff@(posedge clk)
   begin
      countr++;
      if(countr < high_timer) waver <=1;
      else if (countr < 10000 + high_timer) waver<=0;
      else countr <= 0;
   end


endmodule
```

Figure 24: PPM Module System Verilog

5.2.10 Square Wave Module

```
module squarewave (input logic [3:0]spd, input logic clk,en, output logic wave);

   logic [13:0]count;
   logic [10:0]high_time;
   assign high_time = 1000000/(1000+(spd*312));

   always_ff@(posedge clk)
   begin
      if(en)begin
         count++;
         if(count < high_time) wave <=1;
         else if (count < 2*high_time) wave<=0;
         else count <= 0;
      end
      else wave <= 0;
   end


endmodule
```

Figure 25: Square Wave Module System Verilog

5.3 Simulation Results

5.3.1 7-Segment Display Decoder Module (lab3)

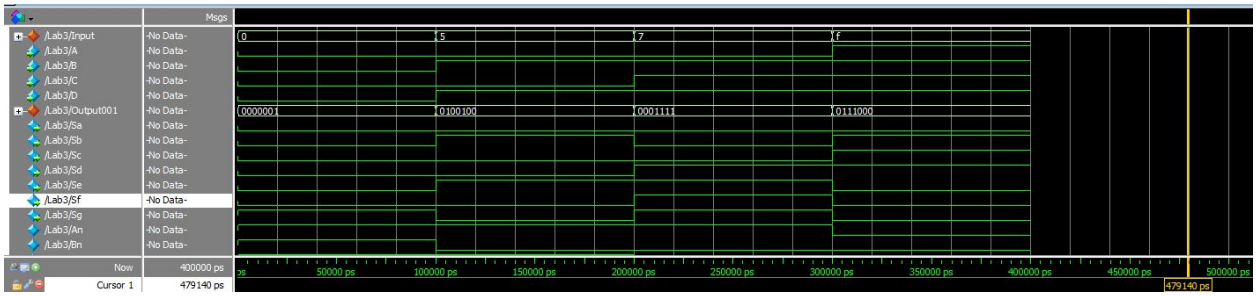


Figure 26: 7-Segment Display Decoder Module ModelSim Simulation

For this Simulation we are interested in seeing the change in the outputs to the displays based on the 4-bit input number.

5.3.2 15-bit Shift Register (Serial-in-Parallel-out) Module (shiftreg)

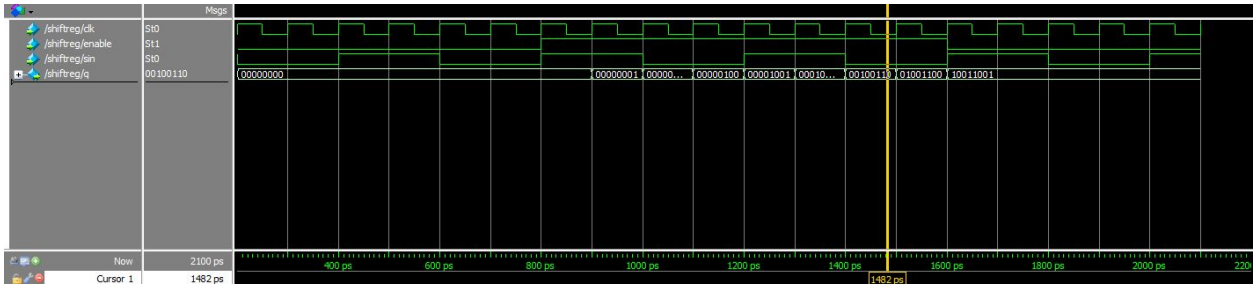


Figure 27: 15-bit Shift Register Module ModelSim Simulation

For this simulation we are interested in seeing how the serial input bits are shifted through the register.

5.3.3 variableDel Module

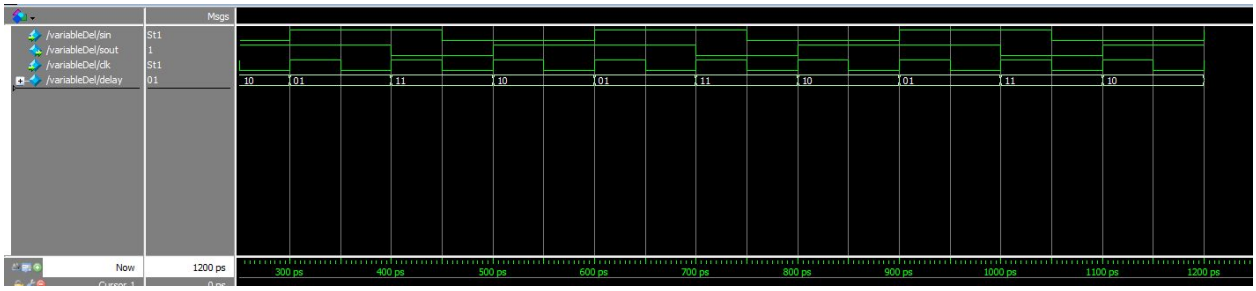


Figure 28: variableDel Module ModelSim Simulation

For this simulation we are interested in seeing how the module delays the serial input to the output by 2 clock cycles.

5.3.4 1MHzPLL Module

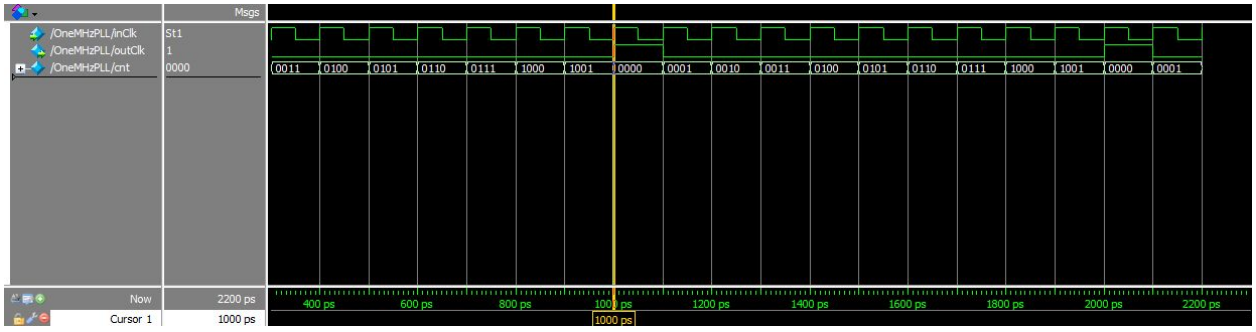


Figure 29: 1MHzPPL Module ModelSim Simulation

For this simulation we are interested in seeing how the clock's rising edge frequency is divided by 10 compared to the input clock.

5.3.5 Counter Module

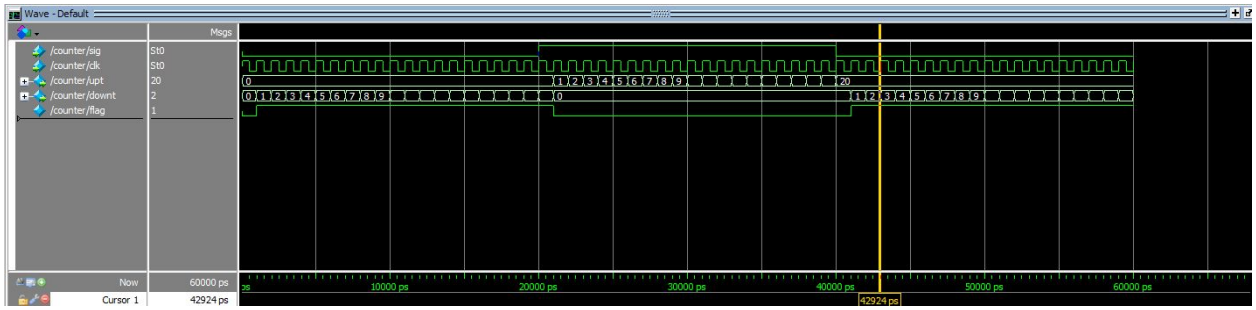


Figure 30: Counter Module ModelSim Simulation

For this simulation we are interested in seeing how the counter counts how long the up time and down time of the signal is.

5.3.6 compt Module

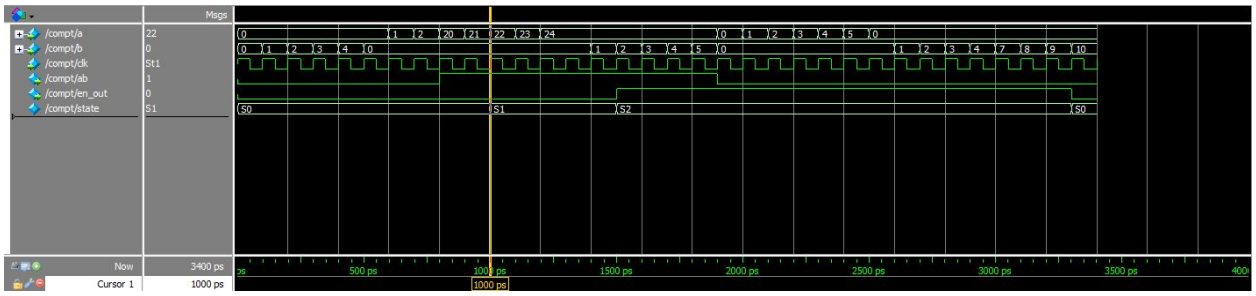


Figure 31: Compt Module ModelSim Simulation

For this simulation we are interested in checking if signal fed by the counter determines if the bit is a one or a zero.

5.3.7 Interpreter Module

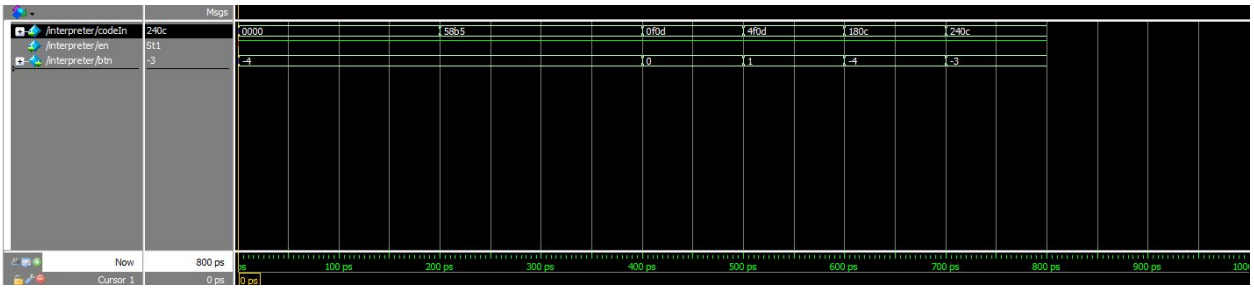


Figure 32: Interpreter Module ModelSim Simulation

For this simulation we are interested in seeing how the module decodes the input signal and relays the decoded numbers to the output port.

5.3.8 CalcSpeed Module

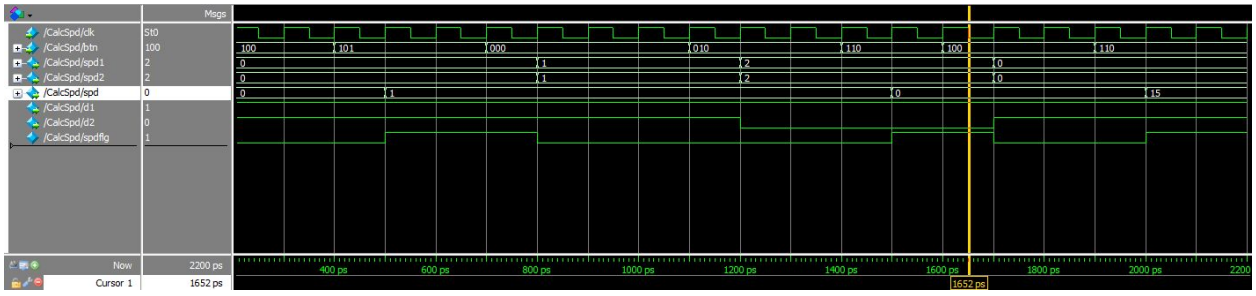


Figure 33: CalcSpeed Module ModelSim Simulation

For this simulation we are interested in seeing how, based on the inputs, this module outputs the direction and speed for each set of motors.

5.3.9 PPM Module

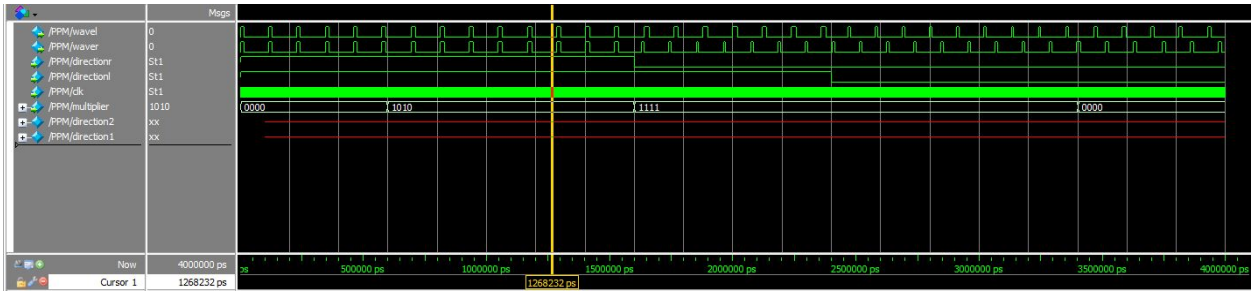


Figure 34: PPM Module ModelSim Simulation

For this simulation we are interested in seeing how, based on the direction and multiplier inputs, the waver and wavel signals must behave. Note that direction1, and direction2 are not required for the function of this block.

5.3.10 Square Wave Module

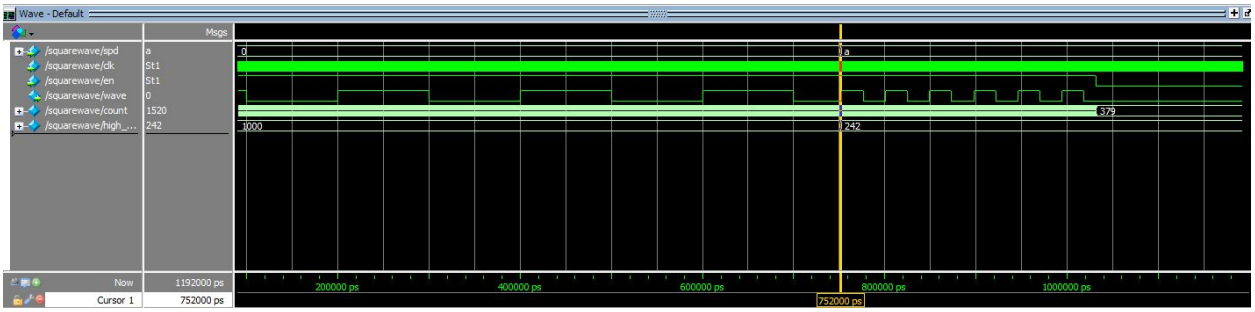


Figure 35: Square Wave Module ModelSim Simulation

For this simulation we are interested in seeing that the spd input determines the frequency of the square wave output.

5.4 Hardware Testing (Implementation)

5.4.1 Suspended Robot Function

5.4.1.1 Front Movement (YouTube Link)

5.4.1.2 Backwards Movement(YouTube Link)

5.4.1.3 Right turn(YouTube Link)

5.4.1.4 Left Turn(YouTube Link)

5.4.1.5 Speed Change(YouTube Link)

In case one of the links is broken, please use this one (Suspended Movement) for the full video.

5.4.2 Floor Robot Function

5.4.2.1 Front Movement(YouTube Link)

5.4.2.2 Backwards Movement(YouTube Link)

5.4.2.3 Stop(YouTube Link)

5.4.2.4 Doughnut Maneuver (YouTube Link)

5.4.2.5 Speed Change(YouTube Link)

In case one of the links is broken, please use this one (Floor Movement) or this one (Doughnut Maneuver) for the full videos.

6 References

1. IR Remote Protocol Documentation: https://www.sbprojects.net/knowledge/ir/sirc.php
2. Sony Remote Picture: https://docs.sony.com/release//Manual_4726907111.pdf