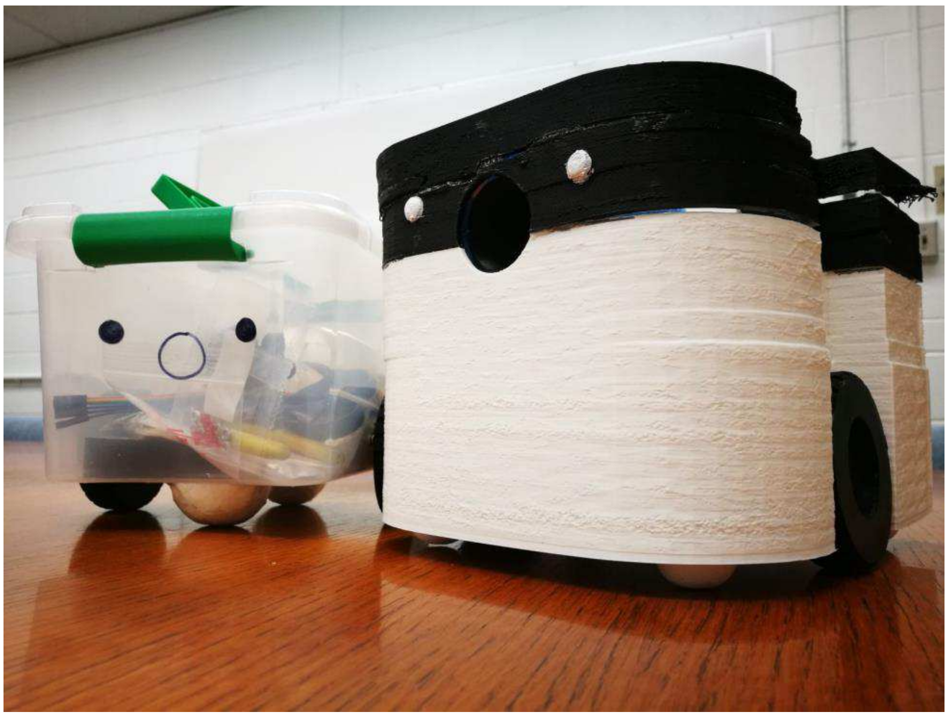
Ree-Ro

A Responsive Autonomous Robot



Ree-Ro is a little robot that does not like loud noises or sounds. It will "eat" the sound, and attempt to find where the sound is. In this implementation. It works best with sine waves.

Ree-Ro incorporates hardware, software, and mechanical parts. Inside Ree-Ro's is a sound envelope detector circuit that is connected to the ADC of the DE0-Nano FPGA board. Ree-ro's mobility is supported by two stepper motors connected to wheels.

The focus of this project is to give basic autonomy to a moving robot.

Bea Venzon, Lulu Li

4/16/2017

Contents

Figures

Introduction

The purpose of this project is to construct a mobile robot that can locate the source of a stationary sound of a known frequency autonomously, while avoiding obstacles. One can conceive of many ways that audio source localization on an autonomously roving robot can be applied in doing tasks that are dangerous to the average human, such as locating the source of a ticking time bomb or finding the source of the suspicious shuffling in the basement late at night. This robot accomplishes neither of these.

Introducing Ree-ro: a Responsive Robot with a vengeance. Ree-ro is equipped with two wheels that gives it the ability to move forward and turn left or right. The wheels are powered by 28-BYJ-48 motors. To explore the environment, Ree-ro is equipped with a sound sensor. A combination of hardware and software is used to process the sound. The sound is pre-amplified and filtered, until ultimately passing through an envelope detector an averager. If the sound is a sine wave, the expected waveform is a relatively flat envelope with a voltage level that varies with sound level. Within the FPGA, some further averaging is done in code.

Project Goals

Baseline Goals

High Level

- A robot that responds to sound and locate the direction of the loudest sound, given that the source is a sine wave.
    - Turn left or right by comparing sound averages.

Low-Level

- Design and build a circuit that can detect the envelope of a sine wave
- Get the motors to run
- Write a simple algorithm for approaching a sound source

Stretch Goals

High Level

- The robot (Ree-ro) should be able to avoid obstacles using ultrasonic sensors
    - The sensor will detect the distance, and sends a stop signal
- When sound level goes below a certain threshold for a certain amount of time, Ree-ro will go into a sleep state
- Have a personality
    - Play prerecorded voice clips for example:
        - Upon finding sound source: FIGHT ME!
        - Detect no sound: Good-night
- Have two displays that are the eyes
    - Have different expressions depending on the situation
        - When approaching an obstacle: O O
        - Upon finding the sound source: ≧ ≦
        - When going to sleep: *‾‾ ‾‾*

Suggestions for Future Work

- Implementing the stretch goals.
- Experiment with using different mic shapes. Consider using a parabolic mic for better directional properties.
- Consider using multiple mics to triangulate the sound.
- Instead of having Ree-ro react to sound level, make Ree-ro react to the tempo of the sound. The sound envelope detection circuit would not be in use for this implementation, and the tempo of the sound must be derived by some method with hardware and/or software.

POWER CONNECTIONS

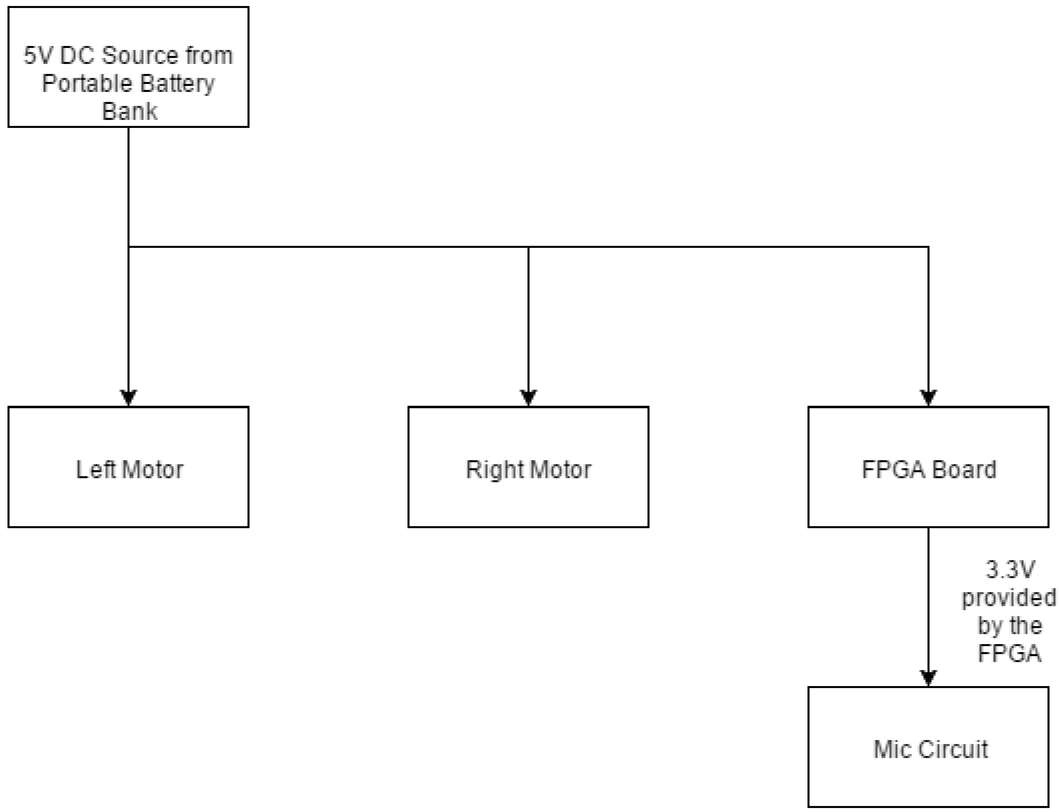


Figure 1: Power Connection to Parts of Ree-Ro

Ree-ro is powered by a portable battery charger for phones. To interface this, we used a stripped end of an old USB cable. The left motor, right motor, and the FPGA then gets power from the prototype board.

We would recommend using a female micro-USB break out board to avoid stripping the wire of the USB cable.

The FPGA provides a 3.3V output, which was used to power the mic circuit. The wire bringing out the power from our FPGA is white and black, corresponding to 3.3V and ground respectively.

SYSTEM DESIGN OVERVIEW

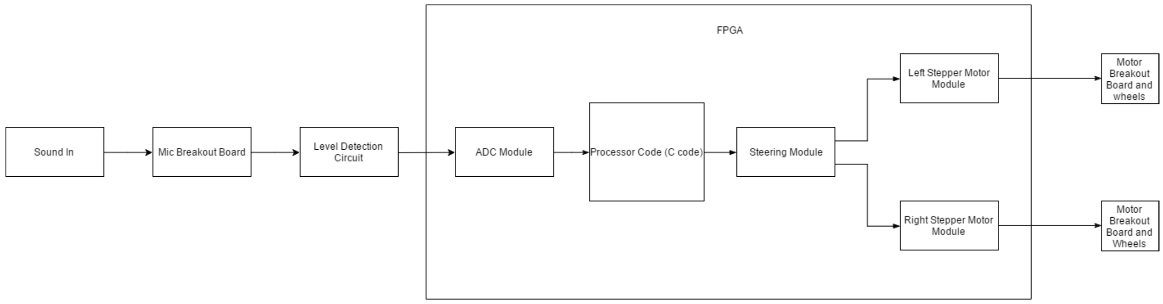


Figure 2: Top Level Overview of Ree-Ro

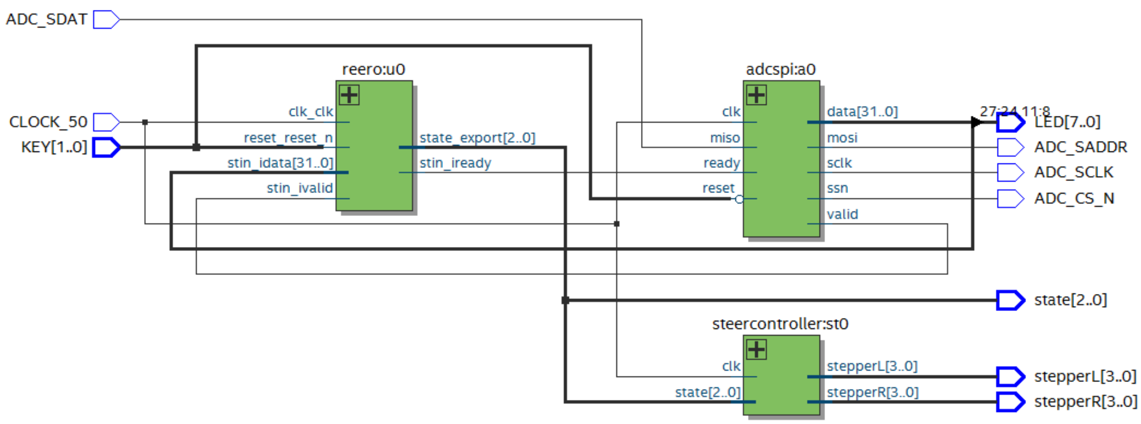


Figure 3: RTL View of Reero

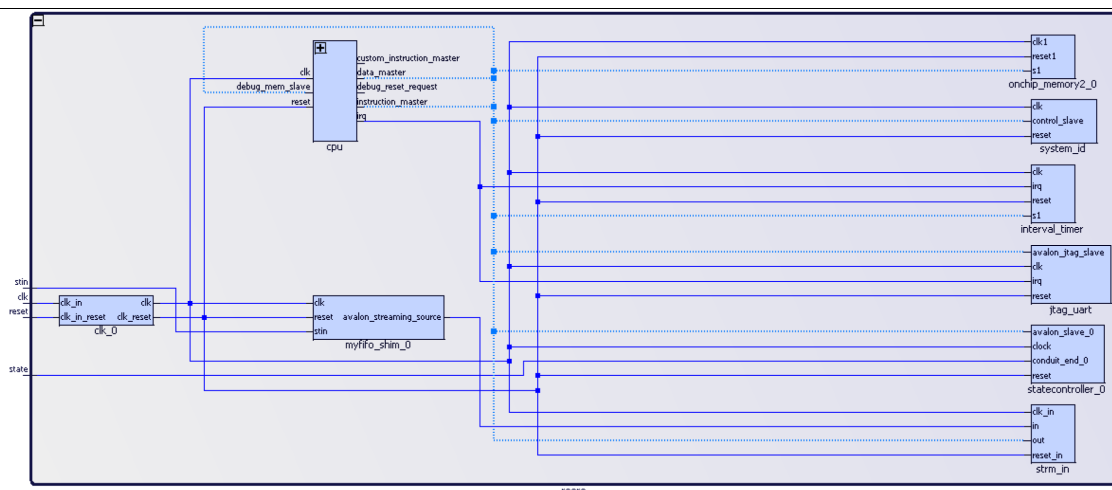


Figure 4: Reero IP Block

Module Descriptions

<table>
<tr><th>Module</th><th>Description</th></tr>
<tr><td>Processor Code & ADC Module</td><td>This module handles taking samples from the ADC and sends control signals to the steering module (rest, move forward, left, right).

This is a modified version of Lab 5 (FIFO), with added outputs for the Steering Module.</td></tr>
<tr><td>State Control</td><td>State control is a register that holds the value that corresponds to the motion and direction of the motor.

<table>
<tr><th>Value</th><th>Action</th></tr>
<tr><td>000</td><td>Rest</td></tr>
<tr><td>001</td><td>Forward</td></tr>
<tr><td>010</td><td>Turn Left</td></tr>
<tr><td>011</td><td>Turn Right</td></tr>
<tr><td>1XX</td><td>Reset</td></tr>
</table>
</td></tr>
<tr><td>Steering</td><td>The steering module is an interface between the processor and the left and right stepper motor modules.</td></tr>
<tr><td>Stepper</td><td>This module goes through the stepper sequence for each motor.</td></tr>
</table>

AUDIO DETECTION

Mic Module

The sound module that was used is an Electret Microphone Amplifier breakout board which uses the MAX 4466 with an adjustable gain potentiometer. More gain was needed to maximize the resolution of the detected sound, so additional circuitry was added (see section: Sound Envelope Detection Circuit).

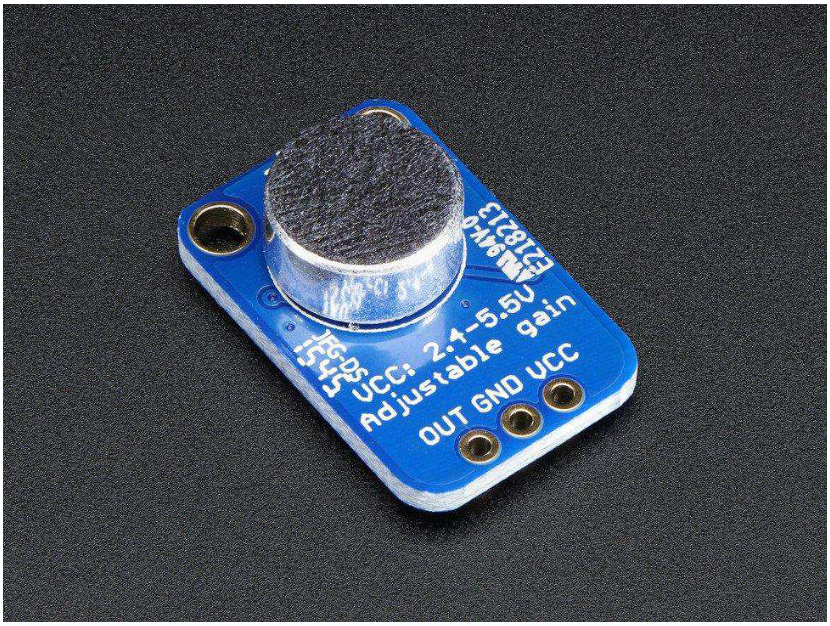


Figure 5: Mic Module [1]

Cone

To make the sound more directional, a 3D printed cone was used to direct the sound to the mic. Around the mic, foam was used to further insulate the cone, and in the final build of Ree-ro, the mic is placed inside the body to reduce the sound coming from the back.

The dimensioned mechanical drawings are included in the end.

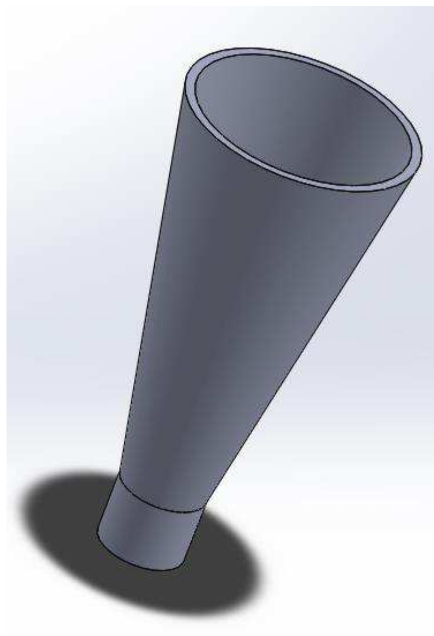


Figure 6: Cone – Isometric View

Sound Envelope Detection Circuit

Below is a functional block diagram of the sound level detection circuit, followed by the actual circuit. The sound level detection circuit detects the envelope of the audio signal from the mic to a corresponding DC level between 0 V to 3.3V.

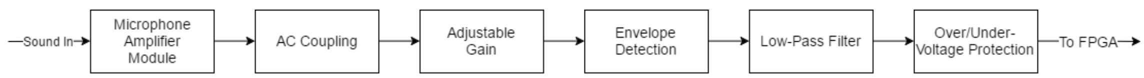


Figure 7: Sound Level Detection Block Diagram

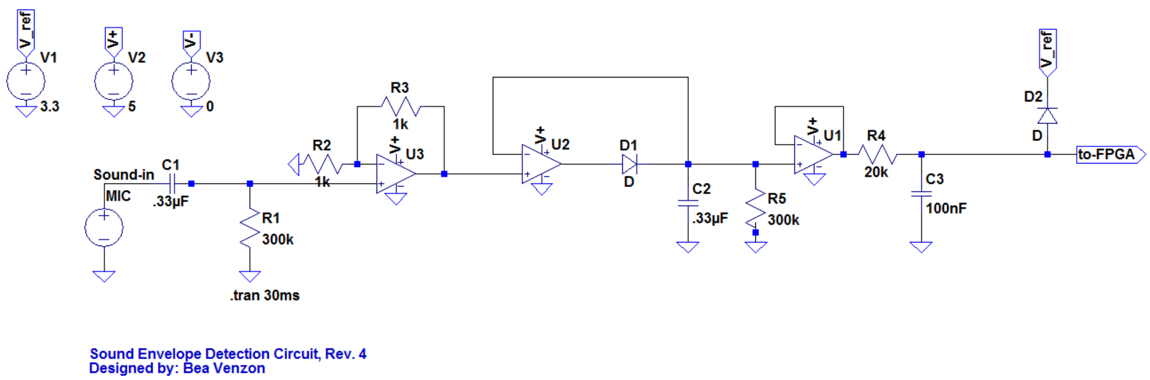


Figure 8: Sound Level Detection Circuit*

* Application Notes:

- R3 or R2 is a potentiometer to make the gain adjustable. R4 is also a potentiometer, to make the low pass filter adjustable.
- Ceramic capacitors were used; there's no guarantee that an electrolytic one would work.
- C1 just has to be a suitably large capacitor, to be used for AC coupling. (Note that C1 and R1 form a high pass filter).
- The low pass filter stage at the final stage is not as important; only use it if the diode detector circuit does not filter enough. If a larger capacitor than .33uF is available, add one too.
- The FPGA's 3.3V output is sufficient for powering this circuit. If, for whatever reason, the user decides to increase the supply voltage of the mic and the op amps to a value greater than 3.3 V (the mic can have an input voltage between 2.7 V to 5.5V), it will be important to have the over/under voltage protection diodes at the final stage of the circuit, with the positive reference tied to a 3.3V source.

BODY

Ree-ro's was modeled in CAD and was 3D printed. The features were designed to have an anthropomorphic appearance – having eyes and a mouth, to show an expressive personality.

The following figures below show a functional overview of Ree-ro's mechanical build. The dimensioned mechanical drawings are shown in the end.

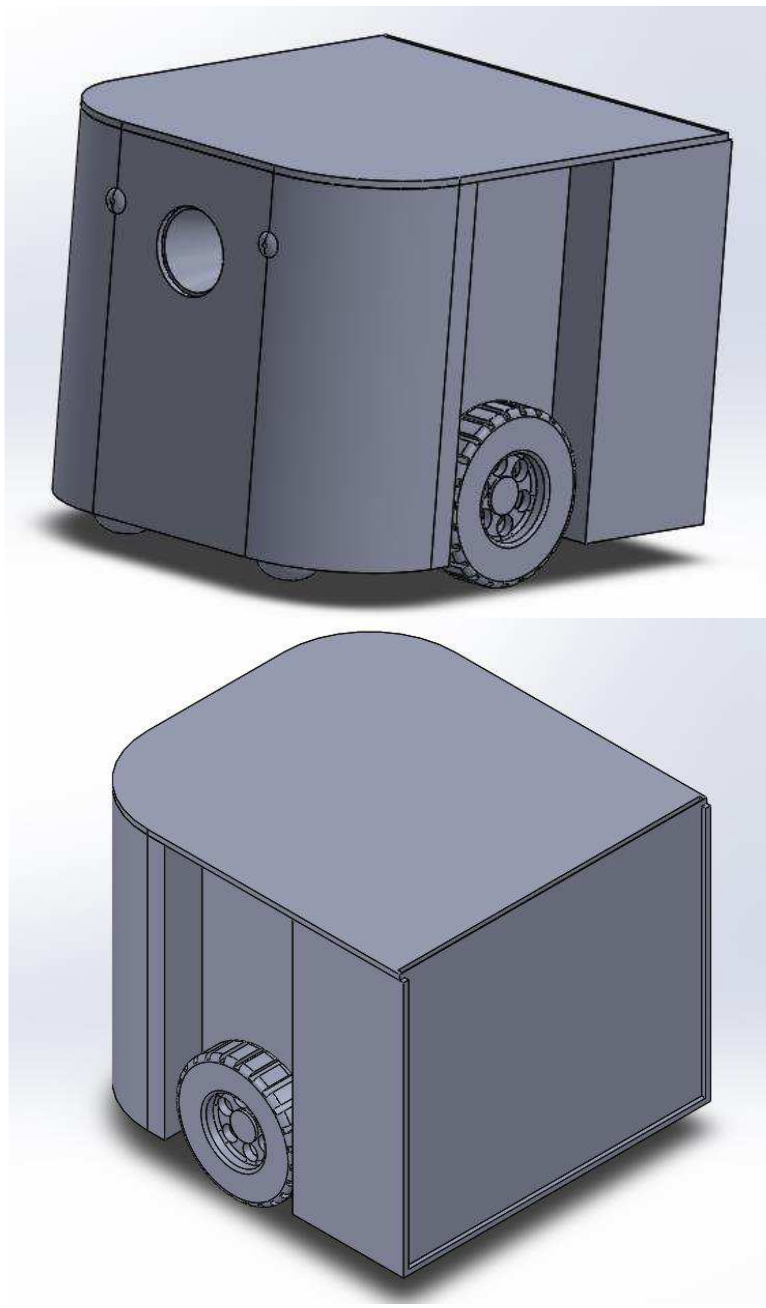


Figure 9: Ree-ro – Complete View (Outside)

Inside Ree-ro, there are two levels. The bottom level houses the charger and the stepper motors. There are mounting holes for the stepper motors, which are secured with a screw and a nut. The stepper motor drivers breakout boards, which are not shown, were placed on standoffs and attached on the side walls.

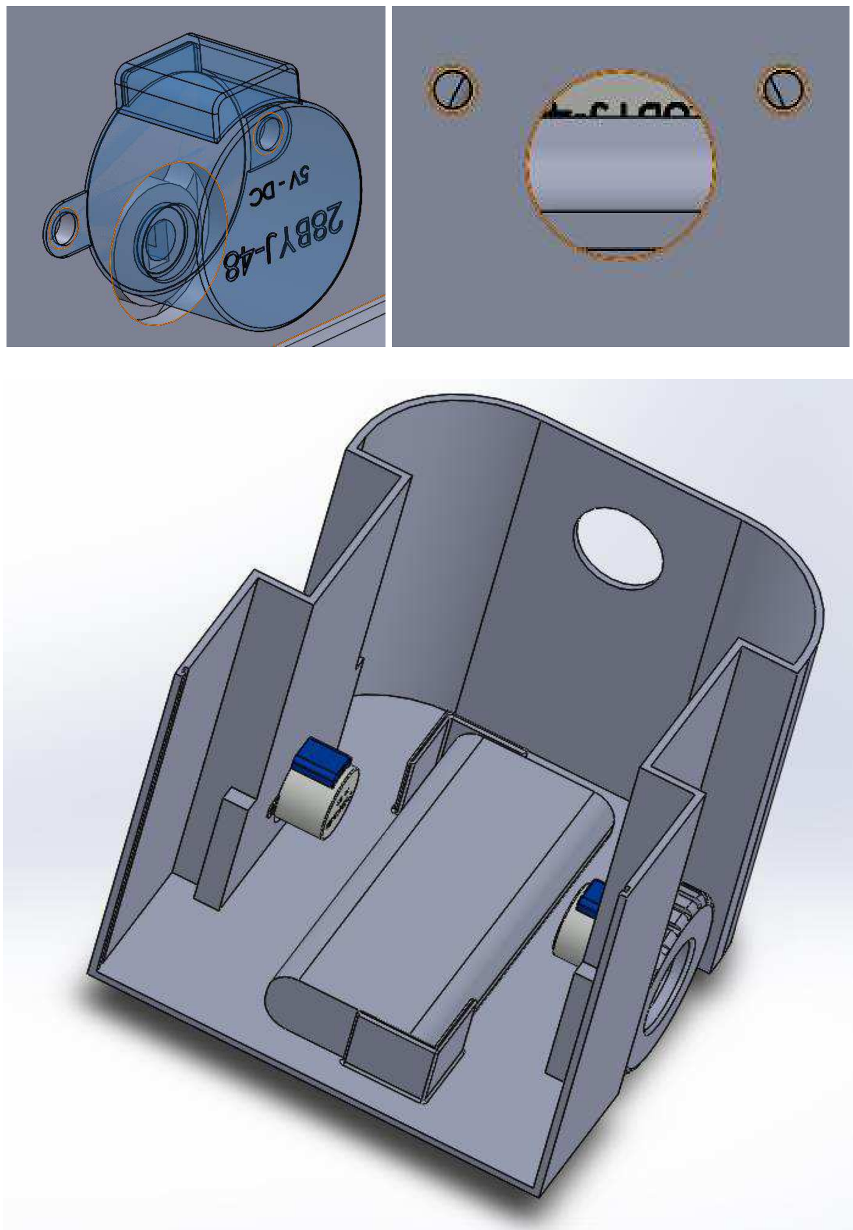


Figure 10: Ree-ro, Bottom Level

On the top level, there is a removable shelf where the FPGA and the sound level detection board is placed. The mounting holes are for the FPGA and the sound level detection board was placed on stand-offs and attached to the shelf with double sided tape. The mic is encased in cylindrical foam, and the pointed stand-offs are meant to pierce through the foam.

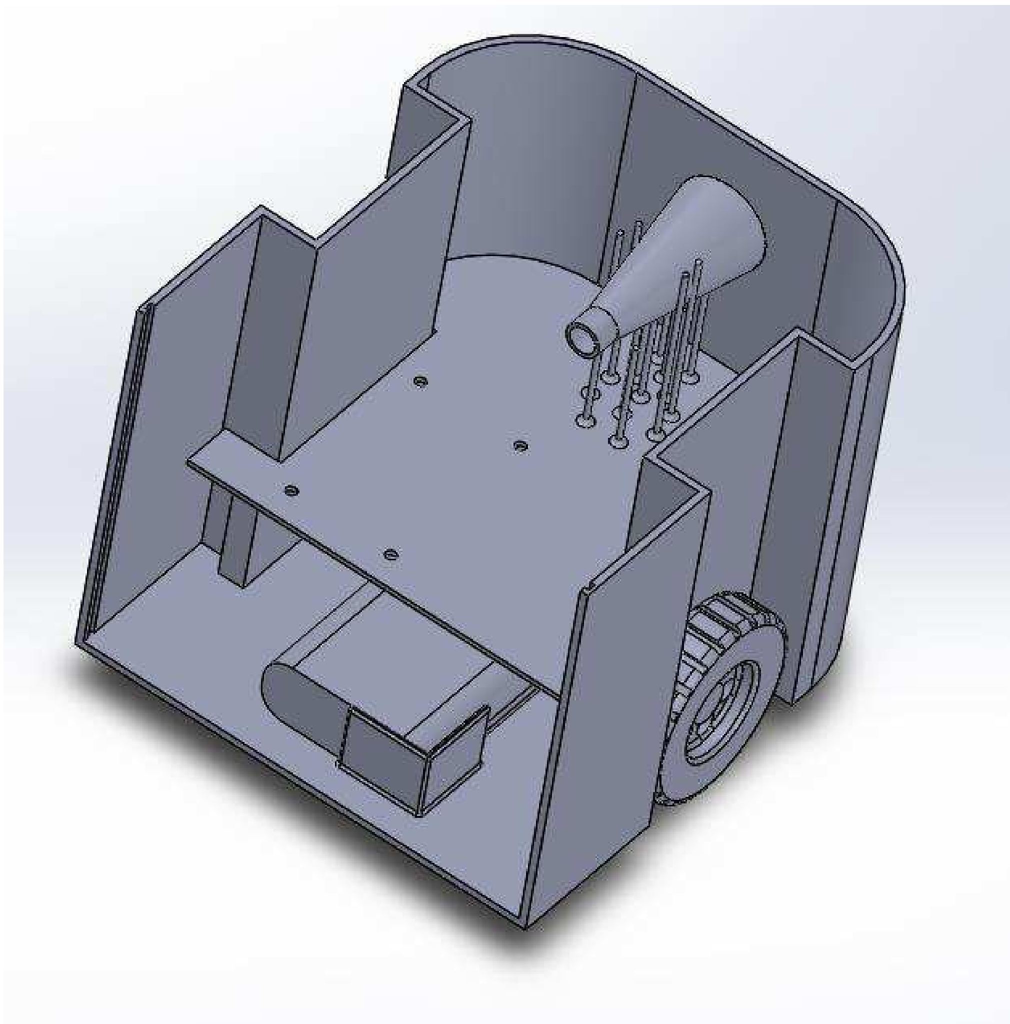


Figure 11: Ree-ro, Top Level

The width of the back cover panel fits into the slits in the back.

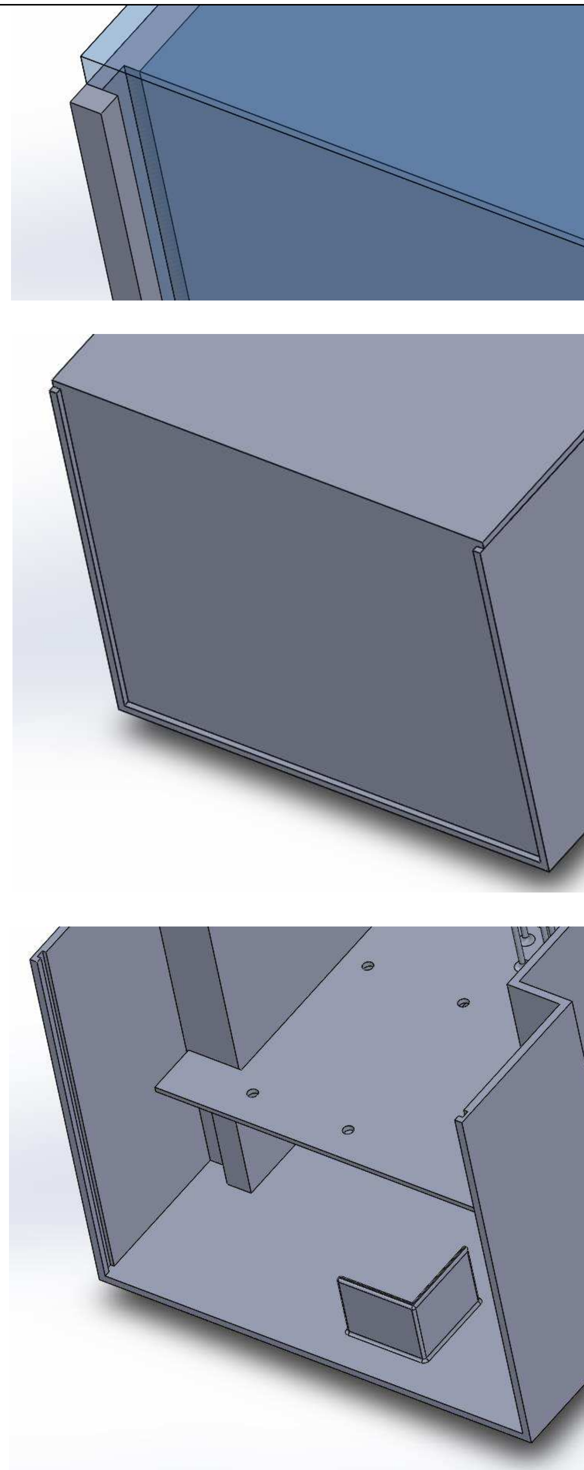


Figure 12: Ree-ro, Back Cover

Stand-offs were placed on the bottom for balance.

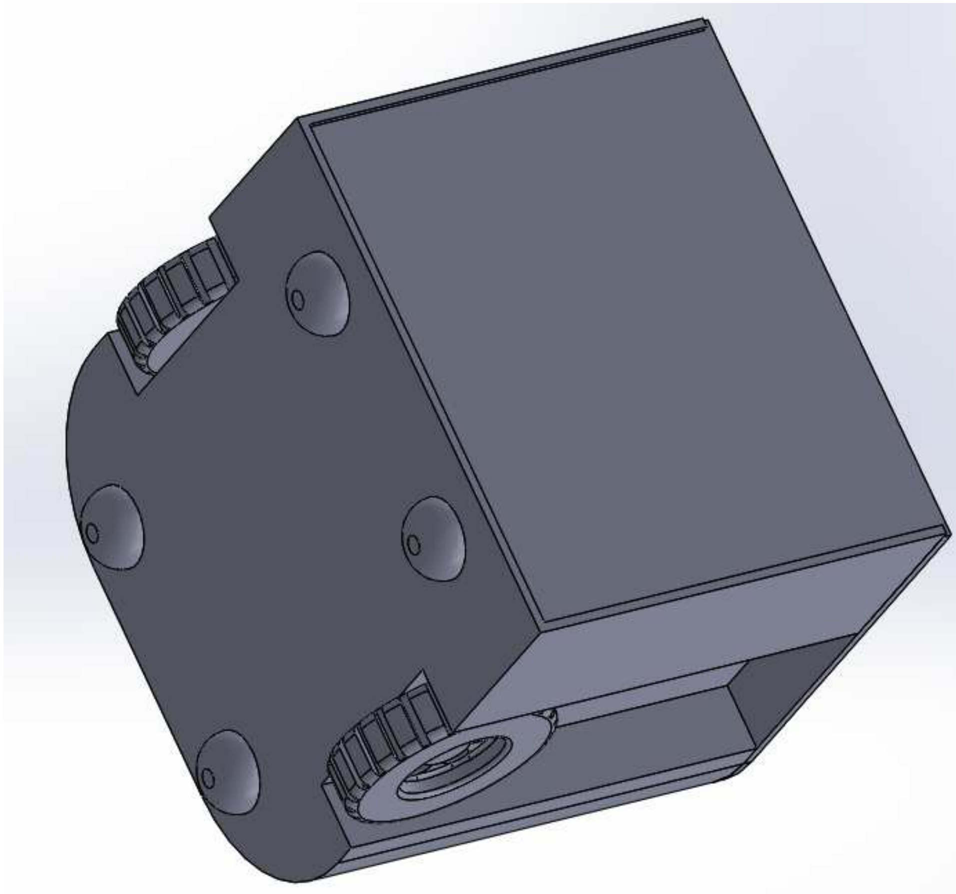


Figure 13: Reero, Bottom View

Mobilization

Stepper Motor

Ree-Ro's mobility comes from two wheels attached to its sides. The motors we used are two small 5V DC stepper motors that could be purchased from Amazon. The chip we use with the motor is ULN2003A. On Amazon, the breakout board mounted with the ULN2003A comes with the motors.

The inner dimensions of the holes in the wheel were sized to fit the stepper motor shaft.

Figure 14: Stepper Motor and Breakout Board

Stepper Motor Properties:

- Model: 28BYJ-48
- Rated Voltage: 5V DC
- Number of Phase: 4
- Speed Variation Ratio: 1/64
- Stride Angle: 5.625°/64
- Frequency: 100Hz
- DC resistance: 50Ω ±7%(25°C)
- Idle In-traction Frequency: > 600Hz
- Idle Out-traction Frequency: > 1000Hz
- In-traction Torque: > 34.3mN.m(120Hz)
- Self-positioning Torque > 34.3mN.m
- Friction torque: 600-1200 gf.com
- Pull in torque: 300gf.com
- Insulated resistance: >10MΩ(500V)
- Insulated electricity power: 600VAC/1mA/1s
- Insulation grade: A
- Rise in Temperature: <40K(120Hz)
- Noise: <35dB(120Hz, No load, 10cm)

Motor Basics

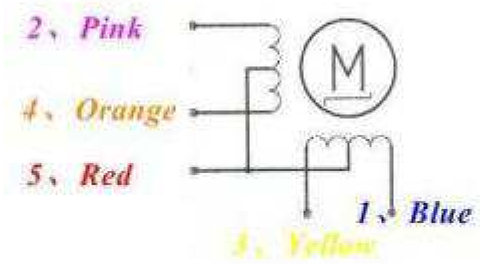


Figure 15: Motor wiring diagram

Table 1: Alternate Naming Used

Blue	A	Bit [0]
Yellow	A/	Bit [2]
Pink	B	Bit [1]
Orange	B/	Bit [3]

Table 2: Stepping Sequence

	Step							
	0:	1:	2:	3:	4:	5:	6:	7:
A	1	0	0	0	0	0	1	1
B	1	1	1	0	0	0	0	0
A/	0	0	1	1	1	0	0	0
B/	0	0	0	0	1	1	1	0

The stepping sequence displayed in Table 2 is used in our motor.sv module. Table 2 shows from top to bottom the least significant bit (LSB) to most significant bit (MSB) corresponding to the decoder in motor.sv.

This stepping sequence combined with the ULN2003A rotates the motor shaft clockwise. For Ree-Ro to move forward, the right wheel should turn clockwise while the left wheel should turn counter-clockwise. To do that, we need to reverse the step sequence in our left motor by counting downwards, instead of upwards.

Motor Connection Block Diagrams

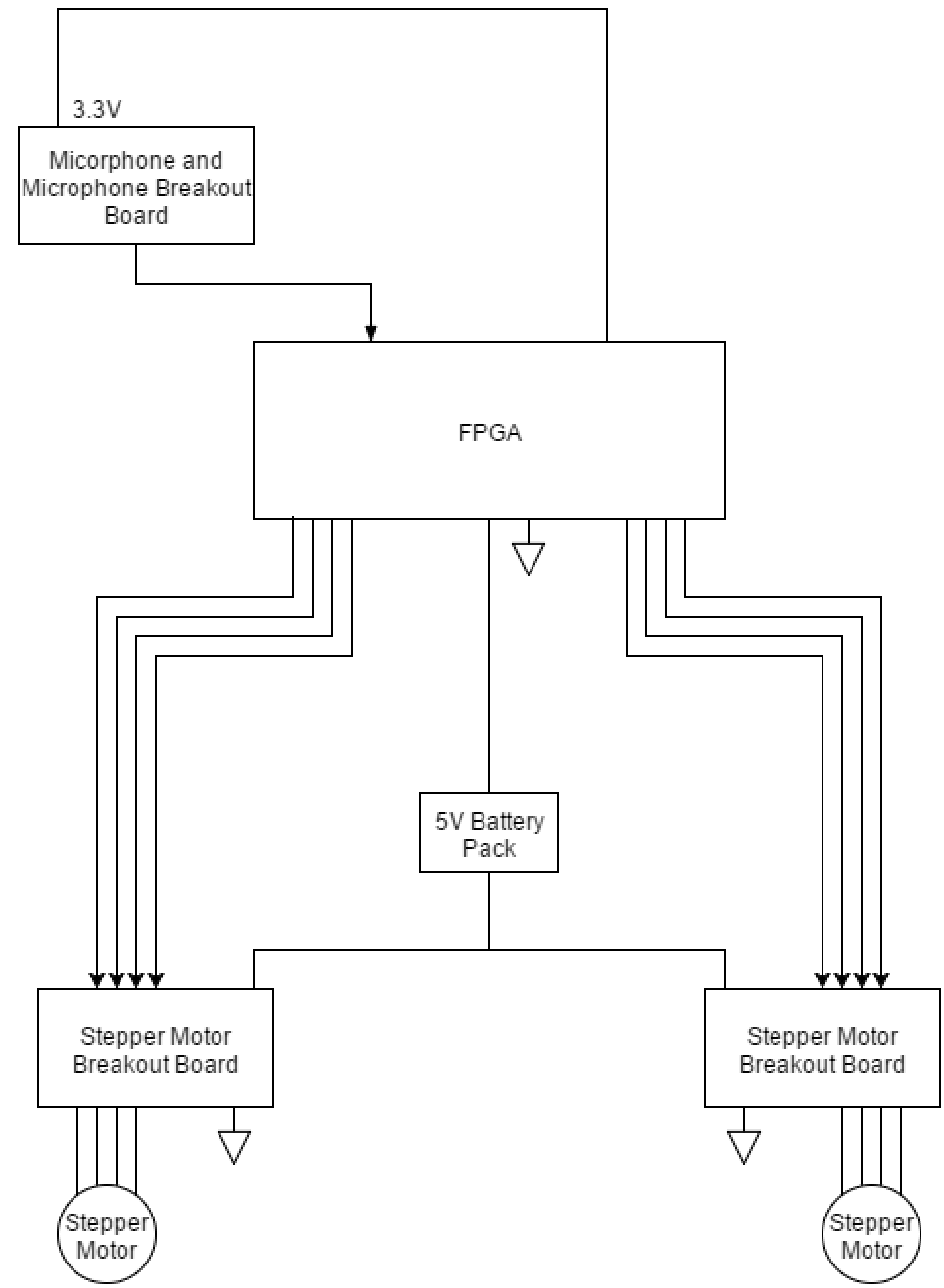


Figure 16: Block Diagram of Motor Connections

Motor Schematic

Table 3: Pin Connections

ULN2003 Pins	Breakout Board Pins
Pin 16:	IN1
Pin 15:	IN2
Pin 14:	IN3
Pin 13:	IN4

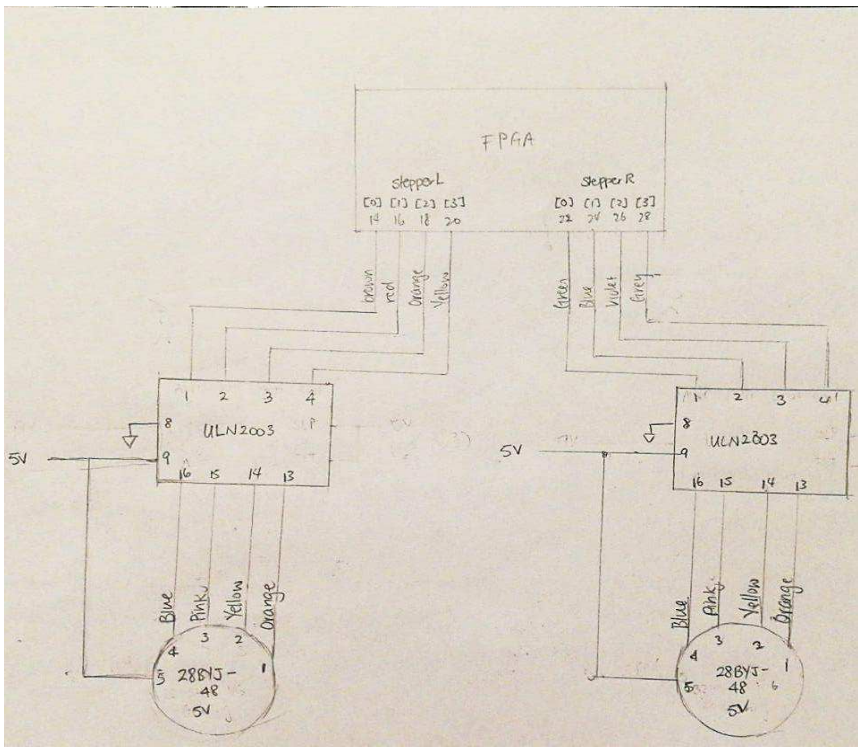


Figure 17: Schematic of motor connection to FPGA

The breakout board internally connects the common (Pin 9) of the chip to the positive power pin of the board. The red (common wire) of the motor is also internally connected to the power pin. So, by connecting 5V to the supply pins of the board, power is supplied to the board and the motor.
The connector on the board that is used to connect to the motor is keyed, so don't worry about connecting the motor backwards.

Simulation Results

Steering

Waveform

Figure 18: Steering Control Waveform

Note: for simulation, the clock division is only 2, as we only needed to test if the clock division works as it should be.

Stepper sequence is also simulated in the waveform above.

Testing

For testing, we brought out our data[2:0] and write signal out into four unused GPIO_0 pins for testing the logic of our steercontrol.sv. We then connected the four signals to a SPDT Grayhill switches to manually simulate the data signals.

CODE

Firmware

```
/* REERO CONTROLLER : reero.c
  Determines the direction that the motor will move by comparing a
  previous average of sound samples with the current one.

  Written by: Bea Venzon
  Date: March 20, 2017
  Modified: April 5, 2017
 */

#include <stdlib.h>
#include "unistd.h"             /* for usleep() */
#include "..\reero_c_bsp\system.h"

// Timing constants
#define             NSAMPLES            1000
                    // Do NOT exceed 65,535 ; will overflow !!!
#define             msDelay             1000
                    // usec --> 1000 usec = 1 msec
#define             MOTOR_RUNTIME       4000            // msec
#define             TURN_RUNTIME        1500            // msec
#define             WAIT                1000            // msec
#define             SAMP_usleep         100             // usec

// Control signals for steering module
#define             REST                0
#define             FORWARD                   1
#define             RIGHT               2
#define             LEFT                3
#define             RESET               4
#define             SETDIR(x)     (*(int*)STATECONTROLLER_0_BASE) = (x)

int main()
{
      SETDIR(RESET); // reset


      unsigned int n ;
      unsigned int totSound, currentSound, prevSound, currentDir,
                   prevDir ;
      totSound = currentSound = prevSound = 0 ;
      int *padc  = (int*) STRM_IN_BASE ;

      while ( 1 ) {
      currentDir = REST ;
      SETDIR(currentDir) ;

      // Allow robot to stabilize before taking samples
      for( n = 0; n < WAIT ; n++ )
            usleep(msDelay) ;

      // Sample and average sound
      totSound = currentSound = 0 ;
```

```
            for(  n = 0 ; n < NSAMPLES ; n++ ) {
                  int data = padc[0] ;
                  totSound += data&0xffff ;
                  usleep(SAMP_usleep) ;
            }

            currentSound = totSound / NSAMPLES ;

      // Direction control
            if( prevSound > currentSound ) {
                  if ( prevDir == LEFT ) {
                        currentDir = RIGHT ;
                        SETDIR(currentDir) ;
                  }
                  else{
                        currentDir = LEFT ;
                        SETDIR(currentDir) ;
                  }
            }
            else{
                  currentDir = FORWARD ;
                  SETDIR(currentDir) ;
            }

      if ((currentDir == LEFT) || (currentDir == RIGHT)) {
            for(n = 0; n < TURN_RUNTIME; n++)
                  usleep(msDelay) ;
      }

      SETDIR(FORWARD) ;

      // Store previous value before next cycle
      prevDir = currentDir ;
      prevSound = currentSound ;

      // The time that the motor is running
      for(n = 0; n < MOTOR_RUNTIME; n++)
            usleep(msDelay) ;

      }
      return 0;
}
```

SystemVerilog/Verilog Code

Top Level Module

```
// lab5top.sv - top-level module for ELEX 7660 lab 5
// Ed.Casas 2017-2-14

// This module was modified to be the top level module for Reero
// Modified by: Bea Venzon
// Modified date: April 4, 2017

module lab5top
  (
   input  logic CLOCK_50,
   output logic [7:0] LED,
   input  logic [1:0] KEY,

   // ADC SPI interface

   output logic ADC_CS_N,       // ssn
   output logic ADC_SADDR,      // mosi
   output logic ADC_SCLK,       // sclk
   input  logic ADC_SDAT,       // miso

   output logic [3:0] stepperL,
   output logic [3:0] stepperR,
      output logic [2:0] state      // state output
   ) ;

   // instantiates a Nios 2 processor with SDRAM memory and
   // ready/valid input for 'myfifo'. System defined in
   // lab5.qsys.

   reero u0
     (
      .clk_clk  (clk),              //  clock_50.clk
      .reset_reset_n   (reset_n), //  reset.reset
      .stin_idata(data),       // connected to myfifoshim stream inputs
      .stin_iready(ready),
      .stin_ivalid(valid),
      .state_export (state)     // state register
      );

      steercontroller st0
      (
       .clk             (clk),             // connect clock
       .state      (state),
       .stepperL   (stepperL),
       .stepperR   (stepperR)
      );

   // Instatiate an SPI master for the DE0-Nano ADC. Reads from
   // ADC SPI pins, outputs to a ready/valid interface that feeds
      // 'myfifo'.

   logic ready ;
   logic valid ;
```

```
   logic [31:0] data ;
   logic reset_n, clk ;

   assign clk = CLOCK_50 ;
   assign reset_n = KEY[0] ;

   adcspi a0
     (
      .sclk(ADC_SCLK), .mosi(ADC_SADDR), .ssn(ADC_CS_N), // SPI master
      .miso(ADC_SDAT),

      .ready(ready),
      .valid(valid),         // data out
      .data(data),

      .clk(clk), .reset(~reset_n) ) ;

   // copy MS ADC bits to LEDs for debug
   assign LED = { data[27:24], data[11:8] } ;


endmodule

// -- start of adcspi.sv ---

// SPI master interface for TI ADC128S022
// for ELEX 7660 201710 Lab 5
// Ed.Casas 2017-2-16

// reads channels 0 and 1
// sclk is clk is divided by 16
// output is 16-bit samples from channels 0 and 1
// samples packed into 32 bits (ch 0 in MS byte)

// ADC128S0022 interface:
// 16 bit transfers

// mosi and cs* change on falling edge of sclk
// mosi bits 13:11 are (next) channel number

// miso sampled on rising edge of sclk
// miso data is on ls 12 bits of miso

// sample rate is sclk rate / 16
// sample rate must be 50 to 200 kHz
// sclk rate must be 800 kHz to 3.2 MHz
// e.g. 50 MHz / 32 = 1.5625 MHz sclk, ~98kHz sampling

// mosi timing relative to rising edge of sclk:
// setup is >10ns, hold  >10ns

// miso timing is relative to falling edge of sclk:
// access is <27ns, hold ~4ns

module adcspi
   (
    output logic sclk, mosi, ssn, // SPI master
```

```
    input logic miso,
    input logic ready,          // ready/valid data out
    output logic valid,
    output logic [31:0] data,

    input logic clk, reset ) ;

   parameter MISO = {5'b00001,27'b0} ;

   // clock/bit counter
   struct packed {
      logic wordcnt ;
      logic [3:0] bitcnt ;
      logic sclk ;
      logic [3:0] clkcnt ; } cnt, cnt_next ;

   logic [31:0] sr ;             // shift register

   logic rising, falling, done ;

   assign sclk = cnt.sclk ;

   // done all bits
   assign done = cnt ==? '{'1,'1,'1,'1} ;

   // clock/bit counter
   assign cnt_next = ( reset || done ) ? '0 : cnt+1'b1 ;
   always@(posedge clk)
     cnt <= cnt_next ;

   assign rising  =  cnt_next.sclk && ~cnt.sclk ;
   assign falling = ~cnt_next.sclk &&  cnt.sclk ;

   always@(posedge clk) begin

      if ( falling )             // shift mosi out
        mosi <= sr[31] ;

      if ( rising )              // shift miso in
        sr <= {sr[30:0],miso} ;

      if ( done ) begin
         data <= sr ;            // copy to parallel out
         sr <= MISO ;            // channel select serial out
         mosi <= MISO[31] ;
         valid <= '1 ;           // data ready
      end

      if ( ready && valid )      // data was read
        valid <= '0 ;

   end

   always@(posedge clk)          // run continously
     ssn <= reset ;

endmodule
```

Interface Logic

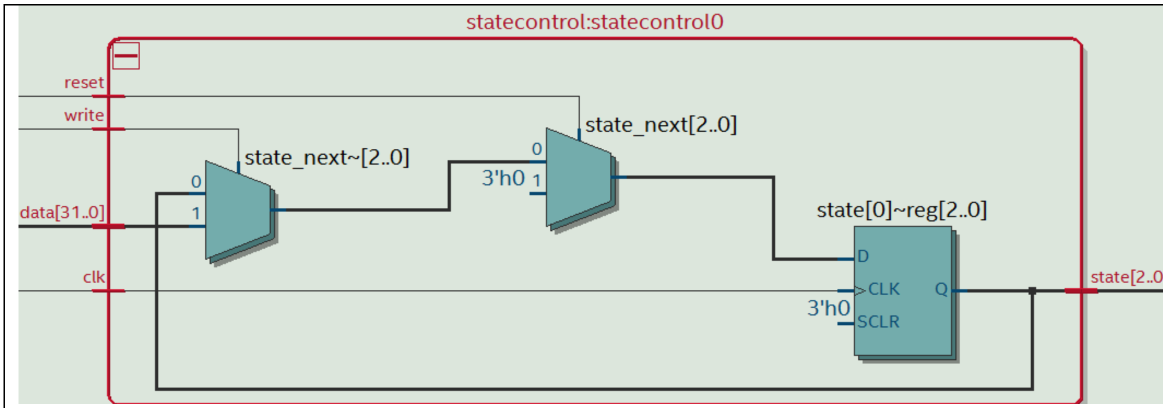


Figure 19: State Controller Schematic

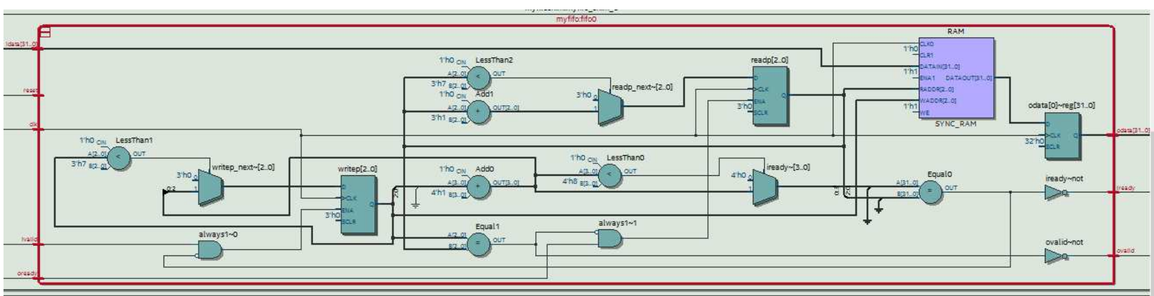


Figure 20: FIFO Schematic

State Register

```
// statecontrol.sv
// Statecontrol is a register that holds the value of the next
// state for Reero. This is for interfacing between the IP block
// containing the CPU and the steering module.

// Author: Bea Venzon
// Date: April 6, 2017

module statecontrol     ( input logic [31:0] data,  // from databus
                          input logic clk,
                          input logic write,
                          input logic reset,
                          output logic [2:0] state );

      logic [2:0] state_next ;

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

      always_comb begin
            if ( reset ) state_next = '0;
            else state_next <= write ? data[2:0] : state ;
      end

endmodule
```

State Register Shim

```
// statecontrolshim.v

// This is a shim that defines the ports of the state module in
// Verilog-2001 syntax and instantiates it.

// Written by: Bea Venzon
// Date: April 4, 2017
module statecontrolshim
  #( parameter fclk = 50000000 )
   (
    input  wire        avs_write,       // avalon_slave.write
    input  wire [31:0] avs_writedata,   //             .writedata
    input  wire        clk,             //   clock_sink.clk
    input  wire        reset,           //   reset_sink.reset
    output wire [2:0]  coe_state        //  conduit_end.state
    );

   statecontrol #( fclk ) statecontrol0  // fclk is clock frequency, Hz
   (
     .data(avs_writedata),          // Avalon MM bus, data
     .write(avs_write),             // " write strobe
     .state(coe_state),             // on/off output for audio
     .reset(reset),
       .clk(clk) ) ;

endmodule
```

FIFO (from Lab 5)

```
// myfifo.sv - FIFO with ready/valid input and output
// for ELEX 7660 201710 lab 5

// Author: Bea Venzon
// Date: March 9, 2017

module myfifo
   (
     output logic iready,         // ready/valid input
     input  logic ivalid,
     input  logic [31:0] idata,

     input  logic oready,        // Avalon-ST output
     output logic ovalid,
     output logic [31:0] odata,

     input logic reset, clk ) ;

   parameter W = 3 ;             // bit width of address pointer
   parameter N = 8 ;             // depth of RAM

   logic [31:0] RAM [0:(N-1)] ;
   logic [(W-1):0] readp, readp_next, writep, writep_next ;

   always_ff @ ( posedge clk ) begin
      readp <= readp_next ;
        writep <= writep_next ;
        RAM[writep] <= idata ;      // read
        odata <= RAM[readp] ;       // write
      end

   always_comb begin
      if ( reset ) begin
            writep_next = 0;
            readp_next = 0;
      end

     // FIFO status bits
      iready = (readp != ( ( writep + 1 ) < N ? ( writep + 1 ) : 0 ) )
? 1'b1 : 1'b0 ;
      ovalid = (readp != writep) ? 1'b1 : 1'b0 ;

      // Increment write pointer
        if ( ivalid && iready )
                   writep_next = ( writep < (N - 1) ) ? ( writep + 1 )
: 0 ;
        else
               writep_next = writep;

        // Increment read pointer
        if ( oready && ovalid )
             readp_next = ( readp < (N - 1) ) ? ( readp + 1 ) : 0 ;
        else
              readp_next = readp;
```

```
    end

endmodule
```

FIFO Shim (from Lab 5)

```
// Author: Ed Casas

module myfifoshim
    (
      output wire iready,    // "conduit" for input from ADC
      input  wire ivalid,
      input  wire [31:0] idata,

      input  wire ready,     // Avalon ST output to strm_in
      output wire valid,
      output wire [31:0] data,

      input  wire reset, clk ) ;

   // instantiate the System Verilog (.sv) implementation

   myfifo fifo0
     (
      .iready(iready),
      .ivalid(ivalid),
      .idata  (idata),

      .oready(ready),
      .ovalid(valid),
      .odata  (data),

      .reset(reset), .clk(clk) ) ;

endmodule
```

Motor Control Modules

Steering Control

```
// steercontrol.sv
// Top module to turn on and off the left and right wheel of ReeRo.
// The counter for the stepper sequence is included in this module as
// both the left and right wheel are using the same counter.

// Author: Lulu Li
// Date: March 29, 2017
// Modified Date: April 4, 2017

module steercontroller ( input logic [2:0] state, // signal from data bus
                         output logic [3:0] stepperL, stepperR,
                         input logic clk ) ;

// count from 0-7 corresponding to stepper sequence
      logic [2:0] seqcntL;
      logic [2:0] seqcntL_next;
      logic [2:0] seqcntR;
      logic [2:0] seqcntR_next;


      // Divide 50Mhz FPGA clock to 5kHz for the stepper motor.
      logic [23:0] clkdiv, clkdiv_next;
      parameter clkdivmax = 24'd249999; // running at 200Hz
      //parameter clkdivmax = 23'd2;        // for testbench

      motor motor_L (.count(seqcntL), .stepper(stepperL));
      motor motor_R (.count(seqcntR), .stepper(stepperR));

      always_ff @(posedge clk)begin
            clkdiv <= clkdiv_next;
            seqcntL <= seqcntL_next;
            seqcntR <= seqcntR_next;
      end

      always_comb begin
            // Reset initial values.
            if(state[2]) begin
                  seqcntL_next = '0;
                  seqcntR_next = '0;
                  clkdiv_next = clkdivmax;
            end
            else begin
                  clkdiv_next = !clkdiv? clkdivmax: clkdiv - 24'b1;
                  case(state[1:0])
                        0: begin // Stopped
                              seqcntL_next = seqcntL;
                              seqcntR_next = seqcntR;
                        end

// Moving forward.
// Left wheel turn ccw (counting down); right wheel turn cw (counting up)
                        1: begin
                              seqcntL_next = !clkdiv? (seqcntL - 1'b1) :
seqcntL;
```

```
                              seqcntR_next = !clkdiv? (seqcntR + 1'b1) :
seqcntR;
                        end
// Turning right.
// Left wheel on, right wheel off.
                        2: begin
                              seqcntL_next = !clkdiv? (seqcntL - 1'b1):
seqcntL;
                              seqcntR_next = seqcntR;
                        end
// Turning left.
// Left wheel off, right wheel on.
                        3: begin
                              seqcntL_next = seqcntL;
                              seqcntR_next = !clkdiv? (seqcntR + 1'b1):
seqcntR;
                        end
                  endcase
            end
      end
endmodule
```

Motor

```
// steercontrol.sv
// This module is a lookup table for the step sequence of the 28byj-48
// unipolar stepper motor.
	// Stepper[0]: Blue wire
	// Stepper[1]: Pink wire
	// Stepper[2]: Yellow wire
	// Stepper[3]: Orange wire
	// Blue and yellow wire makes one coil.
	// Pink and orange wire makes one coil.


// Author: Lulu Li
// Date: March 26, 2017
// Modified Date: April 4, 2017


module motor(input logic [2:0] count,
		output logic [3:0] stepper);

	always_comb begin
		case (count)
			0: stepper = 4'b0011;
			1: stepper = 4'b0010;
			2: stepper = 4'b0110;
			3: stepper = 4'b0100;
			4: stepper = 4'b1100;
			5: stepper = 4'b1000;
			6: stepper = 4'b1001;
			7: stepper = 4'b0001;
		endcase
	end
endmodule
```

Pin Assignments

```
# -------------------------------------------------------------------------- #
#
# Copyright (C) 2016  Intel Corporation. All rights reserved.
# Your use of Intel Corporation's design tools, logic functions
# and other software and tools, and its AMPP partner logic
# functions, and any output files from any of the foregoing
# (including device programming or simulation files), and any
# associated documentation or information are expressly subject
# to the terms and conditions of the Intel Program License
# Subscription Agreement, the Intel Quartus Prime License Agreement,
# the Intel MegaCore Function License Agreement, or other
# applicable license agreement, including, without limitation,
# that your use is for the sole purpose of programming logic
# devices manufactured by Intel and sold by Intel or its
# authorized distributors.  Please refer to the applicable
# agreement for further details.
#
# -------------------------------------------------------------------------- #
#
# Quartus Prime
# Version 16.1.0 Build 196 10/24/2016 SJ Lite Edition
# Date created = 00:42:31  February 14, 2017
#
# -------------------------------------------------------------------------- #
#
# Notes:
#
# 1) The default values for assignments are stored in the file:
#           lab5_assignment_defaults.qdf
#    If this file doesn't exist, see file:
#           assignment_defaults.qdf
#
# 2) Altera recommends that you do not modify this file. This
#    file is updated automatically by the Quartus Prime software
#    and any changes you make may be lost or overwritten.
#
# -------------------------------------------------------------------------- #


set_global_assignment -name FAMILY "Cyclone IV E"
set_global_assignment -name DEVICE EP4CE22F17C6
set_global_assignment -name TOP_LEVEL_ENTITY lab5top
set_global_assignment -name ORIGINAL_QUARTUS_VERSION 16.1.0
set_global_assignment -name PROJECT_CREATION_TIME_DATE "00:42:31
FEBRUARY 14, 2017"
set_global_assignment -name LAST_QUARTUS_VERSION "16.1.0 Lite Edition"
set_global_assignment -name PROJECT_OUTPUT_DIRECTORY output_files
set_global_assignment -name MIN_CORE_JUNCTION_TEMP 0
set_global_assignment -name MAX_CORE_JUNCTION_TEMP 85
set_location_assignment PIN_R8 -to CLOCK_50
set_location_assignment PIN_A15 -to LED[0]
```

```
set_location_assignment PIN_A13 -to LED[1]
set_location_assignment PIN_B13 -to LED[2]
set_location_assignment PIN_A11 -to LED[3]
set_location_assignment PIN_D1 -to LED[4]
set_location_assignment PIN_F3 -to LED[5]
set_location_assignment PIN_B1 -to LED[6]
set_location_assignment PIN_L3 -to LED[7]

set_location_assignment PIN_D5 -to GPIO_0[9]
set_location_assignment PIN_A6 -to GPIO_0[11]
set_location_assignment PIN_D6 -to GPIO_0[13]
set_location_assignment PIN_C6 -to GPIO_0[15]
set_location_assignment PIN_E6 -to GPIO_0[17]
set_location_assignment PIN_D8 -to GPIO_0[19]
set_location_assignment PIN_E9 -to GPIO_0[23]
set_location_assignment PIN_F8 -to GPIO_0[21]

set_location_assignment PIN_D5 -to stepperL[0]
set_location_assignment PIN_A6 -to stepperL[1]
set_location_assignment PIN_D6 -to stepperL[2]
set_location_assignment PIN_C6 -to stepperL[3]
set_location_assignment PIN_E6 -to stepperR[0]
set_location_assignment PIN_D8 -to stepperR[1]
set_location_assignment PIN_F8 -to stepperR[2]
set_location_assignment PIN_E9 -to stepperR[3]

set_location_assignment PIN_J15 -to KEY[0]
set_location_assignment PIN_E1 -to KEY[1]
set_location_assignment PIN_P2 -to DRAM_ADDR[0]
set_location_assignment PIN_N5 -to DRAM_ADDR[1]
set_location_assignment PIN_N6 -to DRAM_ADDR[2]
set_location_assignment PIN_M8 -to DRAM_ADDR[3]
set_location_assignment PIN_P8 -to DRAM_ADDR[4]
set_location_assignment PIN_T7 -to DRAM_ADDR[5]
set_location_assignment PIN_N8 -to DRAM_ADDR[6]
set_location_assignment PIN_T6 -to DRAM_ADDR[7]
set_location_assignment PIN_R1 -to DRAM_ADDR[8]
set_location_assignment PIN_P1 -to DRAM_ADDR[9]
set_location_assignment PIN_N2 -to DRAM_ADDR[10]
set_location_assignment PIN_N1 -to DRAM_ADDR[11]
set_location_assignment PIN_L4 -to DRAM_ADDR[12]
set_location_assignment PIN_M7 -to DRAM_BA[0]
set_location_assignment PIN_M6 -to DRAM_BA[1]
set_location_assignment PIN_L7 -to DRAM_CKE
set_location_assignment PIN_R4 -to DRAM_CLK
set_location_assignment PIN_P6 -to DRAM_CS_N
set_location_assignment PIN_G2 -to DRAM_DQ[0]
set_location_assignment PIN_G1 -to DRAM_DQ[1]
set_location_assignment PIN_L8 -to DRAM_DQ[2]
set_location_assignment PIN_K5 -to DRAM_DQ[3]
set_location_assignment PIN_K2 -to DRAM_DQ[4]
set_location_assignment PIN_J2 -to DRAM_DQ[5]
set_location_assignment PIN_J1 -to DRAM_DQ[6]
set_location_assignment PIN_R7 -to DRAM_DQ[7]
set_location_assignment PIN_T4 -to DRAM_DQ[8]
set_location_assignment PIN_T2 -to DRAM_DQ[9]
set_location_assignment PIN_T3 -to DRAM_DQ[10]
```

```
set_location_assignment PIN_R3 -to DRAM_DQ[11]
set_location_assignment PIN_R5 -to DRAM_DQ[12]
set_location_assignment PIN_P3 -to DRAM_DQ[13]
set_location_assignment PIN_N3 -to DRAM_DQ[14]
set_location_assignment PIN_K1 -to DRAM_DQ[15]
set_location_assignment PIN_R6 -to DRAM_DQM[0]
set_location_assignment PIN_T5 -to DRAM_DQM[1]
set_location_assignment PIN_L1 -to DRAM_CAS_N
set_location_assignment PIN_L2 -to DRAM_RAS_N
set_location_assignment PIN_C2 -to DRAM_WE_N
set_global_assignment -name PARTITION_NETLIST_TYPE SOURCE -section_id
Top
set_global_assignment -name PARTITION_FITTER_PRESERVATION_LEVEL
PLACEMENT_AND_ROUTING -section_id Top
set_global_assignment -name PARTITION_COLOR 16764057 -section_id Top
set_global_assignment -name POWER_PRESET_COOLING_SOLUTION "23 MM HEAT
SINK WITH 200 LFPM AIRFLOW"
set_global_assignment -name POWER_BOARD_THERMAL_MODEL "NONE
(CONSERVATIVE)"
set_location_assignment PIN_A10 -to ADC_CS_N
set_location_assignment PIN_B10 -to ADC_SADDR
set_location_assignment PIN_B14 -to ADC_SCLK
set_location_assignment PIN_A9 -to ADC_SDAT



set_global_assignment -name QIP_FILE reero/synthesis/reero.qip
set_global_assignment -name SYSTEMVERILOG_FILE statecontrol.sv
set_global_assignment -name SYSTEMVERILOG_FILE steercontroller.sv
set_global_assignment -name SOURCE_FILE lab5.qsf
set_global_assignment -name SYSTEMVERILOG_FILE motor.sv
set_global_assignment -name SYSTEMVERILOG_FILE lab5top.sv
set_global_assignment -name SDC_FILE lab5.sdc
set_instance_assignment -name PARTITION_HIERARCHY root_partition -to |
-section_id Top
```

Testbench

Steering Control Test Bench

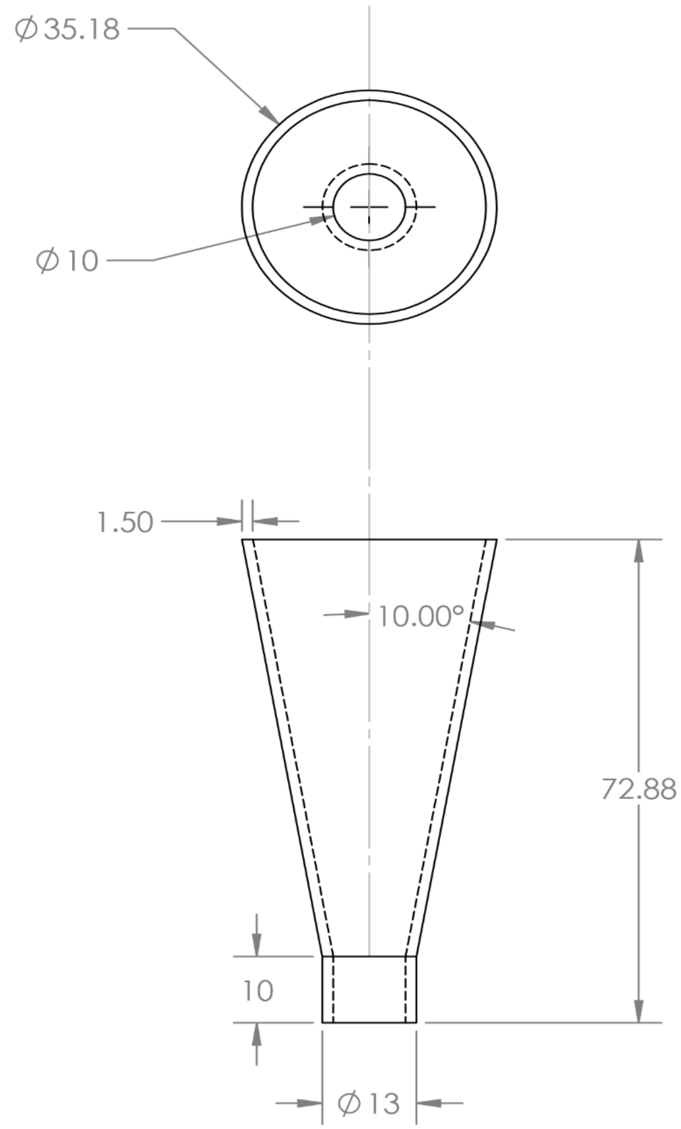
```
// steercontrol_tb.sv
// Test bench code for steercontrol.sv

// Author: Lulu Li
// Date: March 29, 2017

module steercontrol_tb();

	logic clk;
	logic reset;
	logic write;
	logic [31:0] data;
	wire [3:0] stepperL, stepperR;

	steercontrol dut(.*);

	initial begin

		// initial state
		reset = 0;
		clk = 0;
		write = 0;
		data = '0;

		repeat(2) begin
			#10ns; clk = ~clk;
		end

		// write a reset
		write = 1;
		data = 32'd4;
		repeat(2) begin
			#10ns; clk = ~clk;
		end
		write = 0;
		data = 'x;

		// idle for a bit
		repeat(20) begin
			#10ns; clk = ~clk;
		end

		// write a direction
		write = 1;
		data = 32'd2;
		repeat(2) begin
			#10ns; clk = ~clk;
		end
		write = 0;
		data = 'x;

		// idle for a bit
		repeat(20) begin
```

```
                #10ns; clk = ~clk;
            end

            // write a direction
            write = 1;
            data = 32'd1;
            repeat(2) begin
                #10ns; clk = ~clk;
            end
            write = 0;
            data = 'x;

            // idle for a bit
            repeat(20) begin
                #10ns; clk = ~clk;
            end
        end

endmodule
```

Mechanical Drawings

1. Cone
2. Wheel
3. Body
4. Shelf
5. Cover Plate

Note that all dimensions are stated in mm.



	NAME	DATE
DRAWN	Bea Venzon	2017-03-14
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		

COMMENTS:

DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL

FINISH

NEXT ASSY | USED ON

APPLICATION

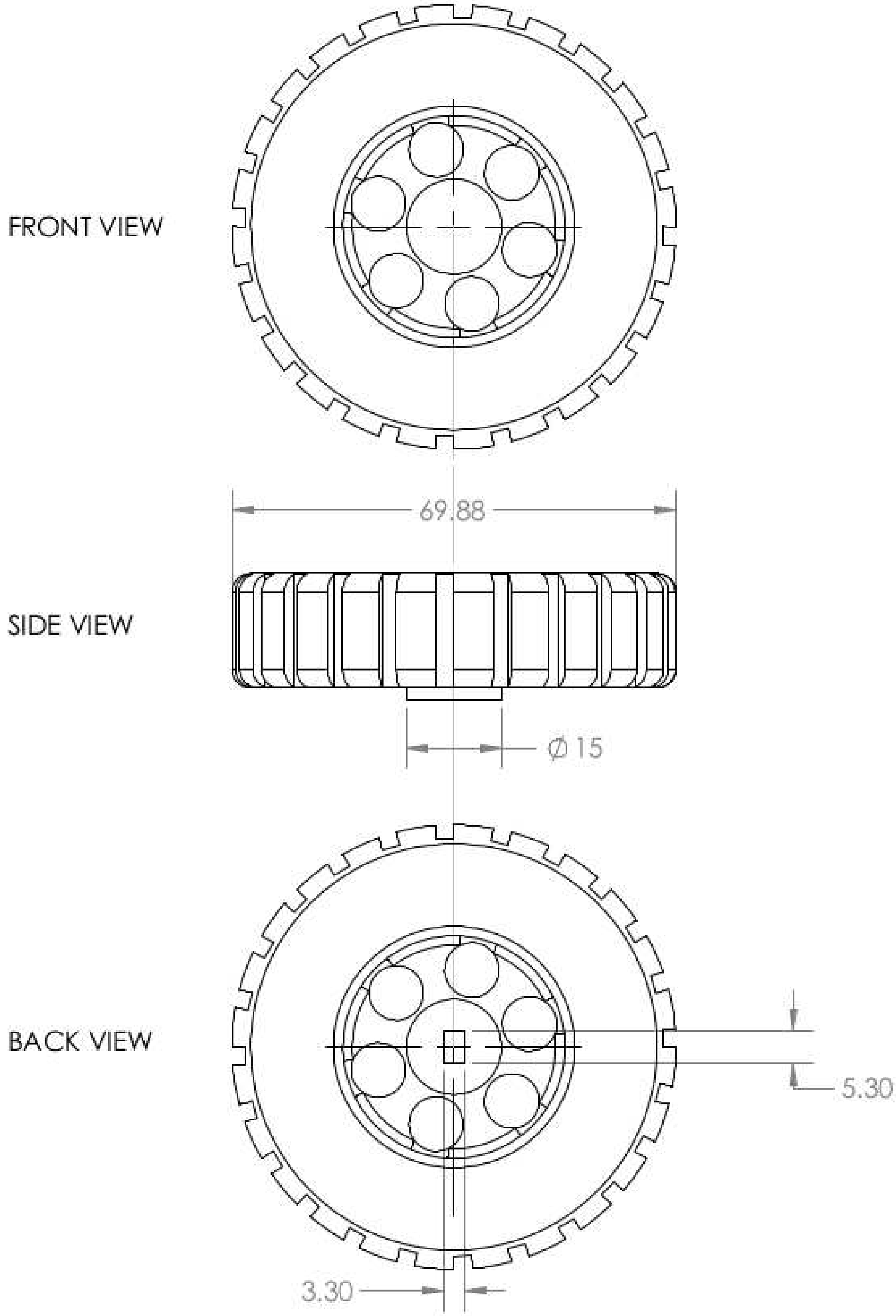
DO NOT SCALE DRAWING

PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <COMPANY NAME >. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <COMPANY NAME> IS PROHIBITED.

SIZE **A** | DWG. NO. Mic_Cyl1 | REV. 1

SCALE:1:1 | WEIGHT: | SHEET 1 OF 1



PROPRIETARY AND CONFIDENTIAL

NEXT ASSY	USED ON	
APPLICATION		DO NOT SCALE DRAWING

DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL

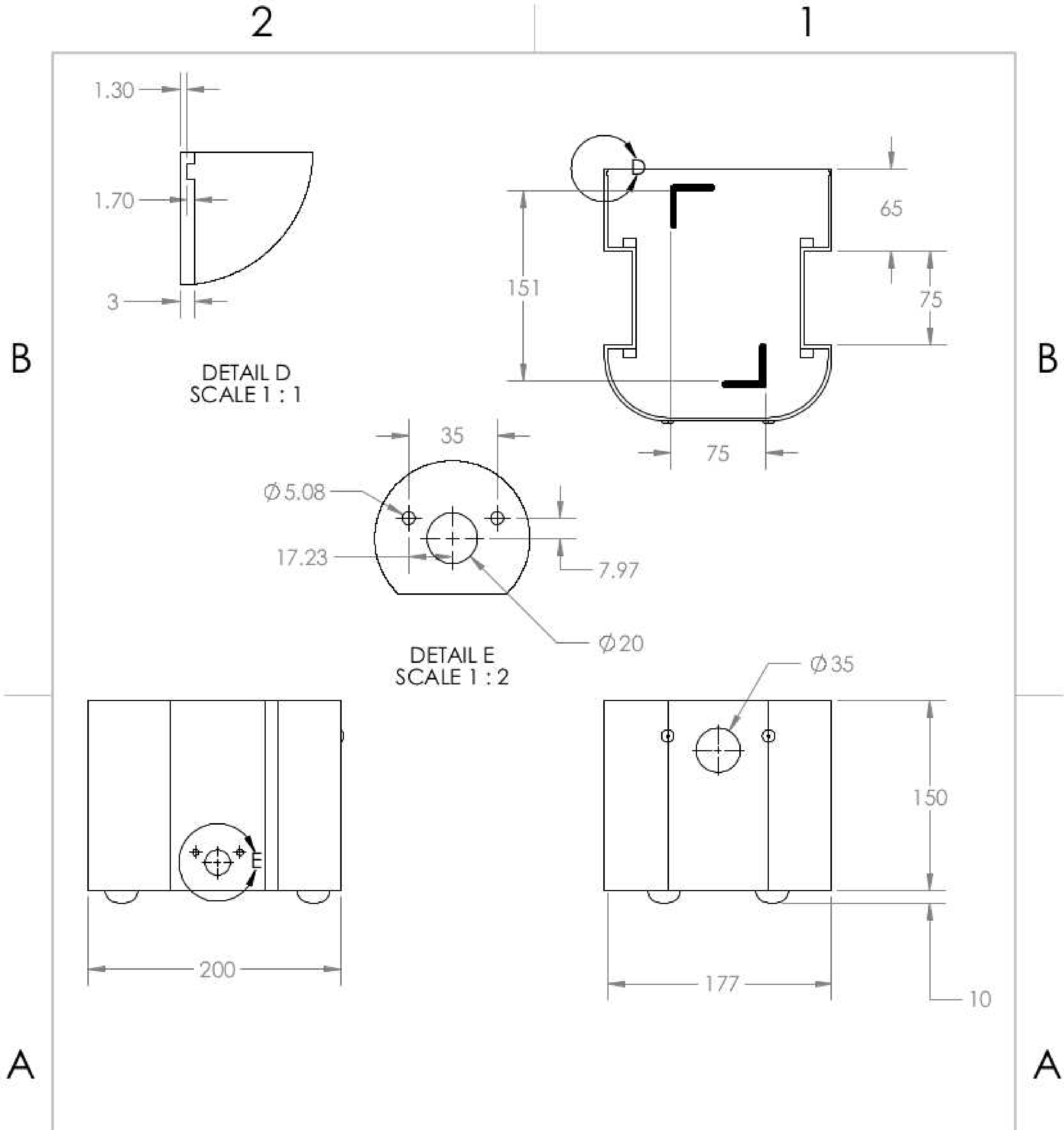
FINISH

	NAME	DATE
DRAWN	Bea Venzon	2017-03-14
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		

COMMENTS:

SIZE	DWG. NO.	REV.
A	Wheel	1

SCALE:1:2 WEIGHT: SHEET 2 OF 2



PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <COMPANY NAME>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <COMPANY NAME> IS PROHIBITED.

NEXT ASSY | USED ON

APPLICATION

DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL

FINISH

DO NOT SCALE DRAWING

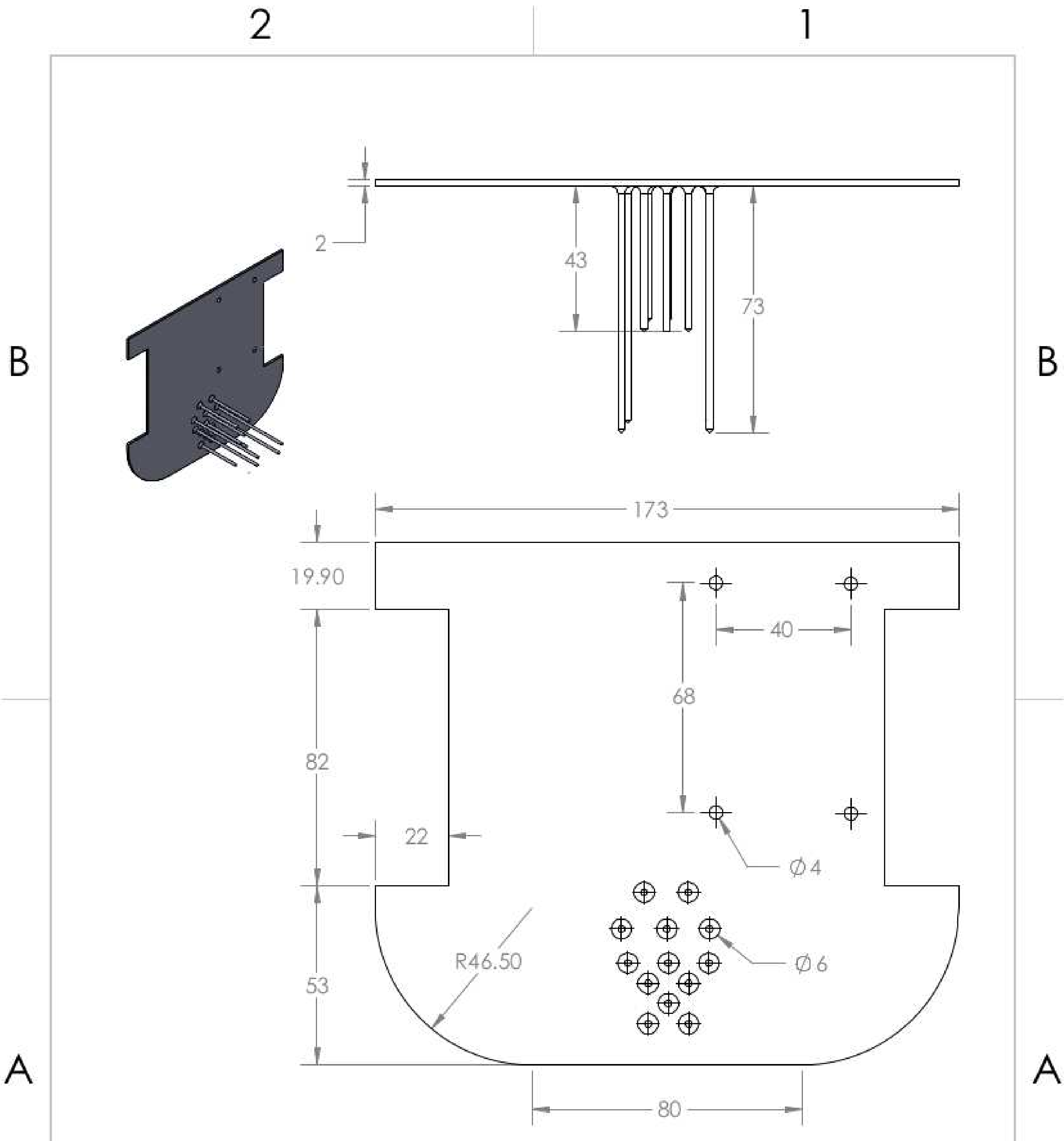
	NAME	DATE
DRAWN	Bea Venzon	2017-03-14
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		

COMMENTS:

SIZE A | DWG. NO. Body | REV. 1

SCALE 1:5 | WEIGHT: | SHEET 3 OF 3

2
1



PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <COMPANY NAME>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <COMPANY NAME> IS PROHIBITED.

NEXT ASSY	USED ON
APPLICATION	

DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL

FINISH

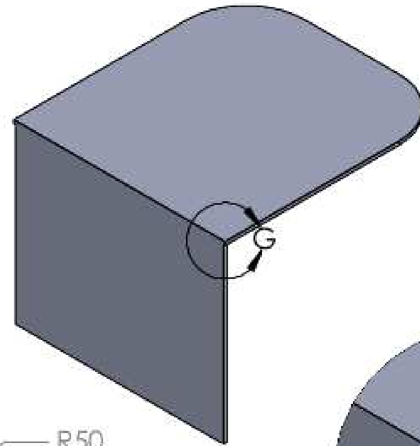
DO NOT SCALE DRAWING

	NAME	DATE
DRAWN	Bea Venzon	2017-03-14
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

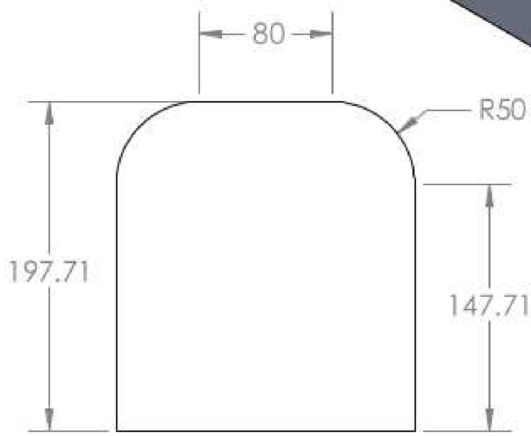
SIZE	DWG. NO.	REV
A	Shelf	1
SCALE: 1:5	WEIGHT:	SHEET 4 OF 4

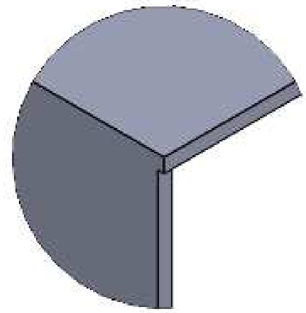
2 1

2 1

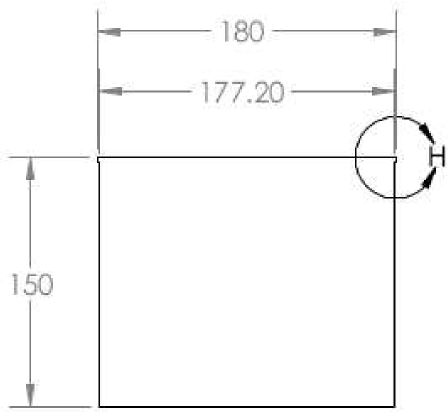


TOP VIEW

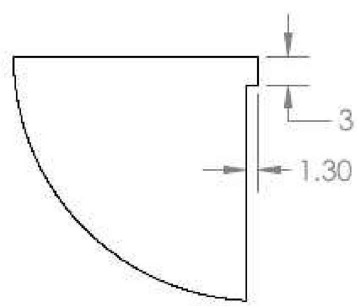




DETAIL G
SCALE 1 : 1



BACK VIEW



DETAIL H
SCALE 1.5 : 1

B A

PROPRIETARY AND CONFIDENTIAL

NEXT ASSY	USED ON	DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±		NAME	DATE
APPLICATION		MATERIAL	DRAWN	Bea Venzon	2017-03-14
		FINISH	CHECKED		
		DO NOT SCALE DRAWING	ENG APPR.		
			MFG APPR.		
			Q.A.		
			COMMENTS:		

SIZE	DWG. NO.	REV.
A	Cover Plate	1
SCALE: 1:5	WEIGHT:	SHEET 5 OF 5

2 1

PHOTOS

Printing Process

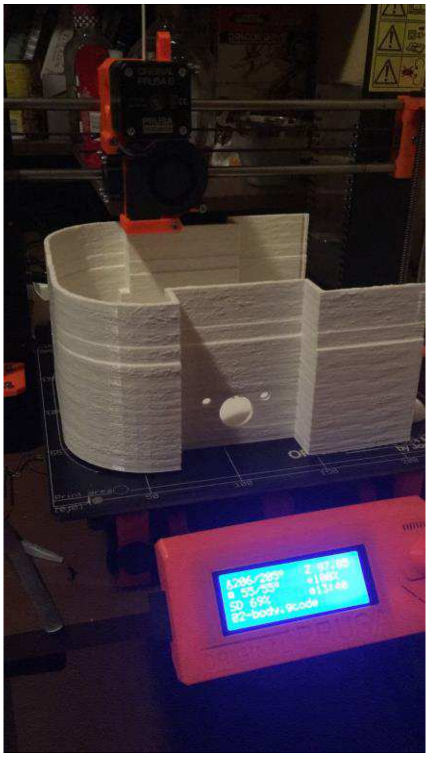


Figure 21: 3D Printing Process – Body

Figure 22: 3D Printing Process – Wheels

Final Results

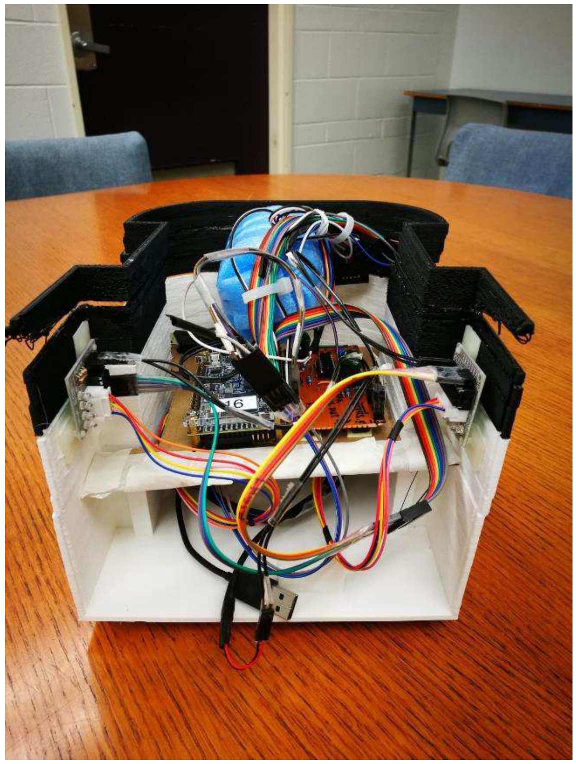


Figure 23: Complete Assembly

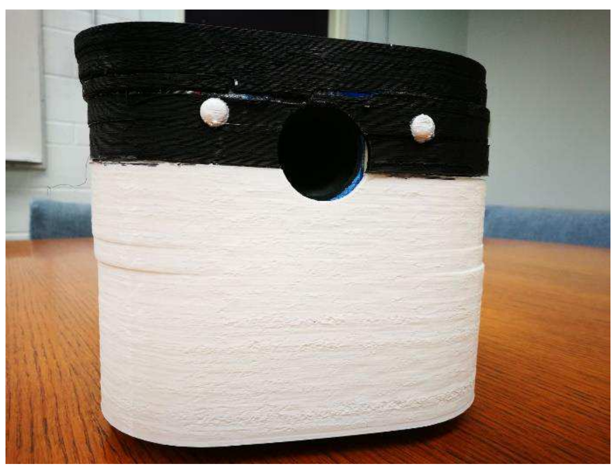


Figure 24: Ree-Ro Front View

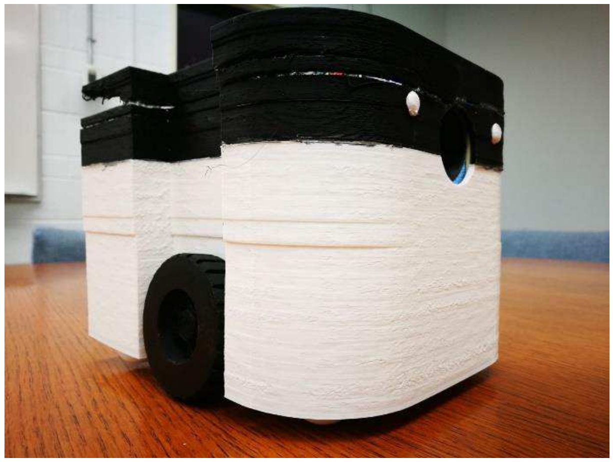


Figure 25: Ree-Ro Angle View

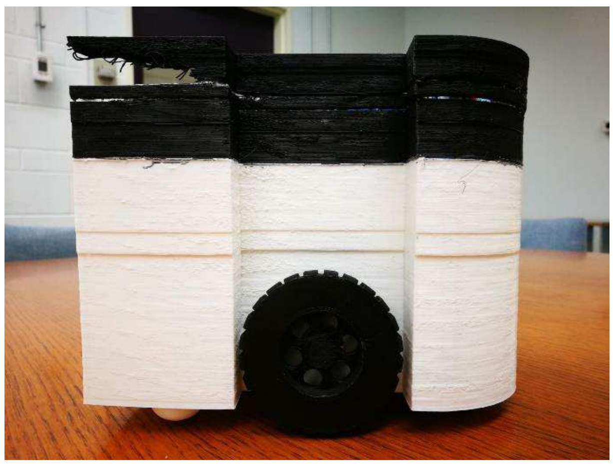


Figure 26: Ree-Ro Side View