Alex Rich and Aaron Rosen
Engineering 155
Final Project Status Report
November 23, 2015

Project Status Report: A Robot Controlled Over The Internet

The team has made progress on a robotic vehicle that receives user input from a website and executes the commands. The website is hosted by an E155 Raspberry Pi2 Apache2 server, and contains a visual grid interface. The grid represents different locations on the ground with respect to the vehicle. To send the vehicle to the desired location, the user clicks on the corresponding point on the grid, which inputs commands into a buffer to be sent to the vehicle. The Pi will parse the commands and send them wirelessly using a Belkin FT8001 Bluetooth dongle to a BlueSMiRF located on the vehicle. The vehicle's controller (the E155 μMudd board) reads these commands and executes them. An overview of this system is shown in Figure 1. Below we discuss the current status for each part (website, Pi, FPGA, vehicle, new hardware) in detail.

Website

The website currently contains a visual user interface that contains instructions for use, a grid on which to input locations for the robot to maneuver to, a list of the locations currently buffered for sending, and an option to clear the buffer. The code for the website is shown in Appendix A. The website was built using HTML, Javascript, and Bootstrap CSS.

The Pi hosts the website. Upon clicking in a grid space, the page's javascript will eventually calculate the left and right tread speeds and duration of movement required to get the tank to move from its original position to the new position. Currently, it creates a random command to be sent to the FPGA. Next, the javascript makes an HTTP GET request to the inputChars resource of the Pi. When the request is completed, the page updates, either submitting the next command in the buffer or waiting for another input from the user.

Raspberry Pi 2

After receiving a GET request, the common gateway interface (CGI) reads the input parameters and calls a Python script. The Python script utilizes the `bluetooth` module to allow sending data using the bluetooth dongle. Since the robot has a constant bluetooth device address, this address is hard coded into the Python script. When called, the python script sends the commands, one at a time, to the robot. Currently, the system sleeps for an amount of time to theoretically give the vehicle enough time to move. However, in the future, the Pi will wait for acknowledgement from the FPGA. The code on the Pi is shown in Appendix B. Figure 2 show the flow of data and control through the Pi.

FPGA

The FPGA reads data from a BlueSMiRF using UART hardware coded in SystemVerilog and then processes and executes the command. It is constructed as a controller-datapath pair with three main submodules - receiveMSG, executeCommand, and sendAck. The SystemVerilog code running on the FPGA is shown in Appendix C. The FPGA and BlueSMiRF are the only two electrical components on the breadboard. The schematic is shown in figure 4

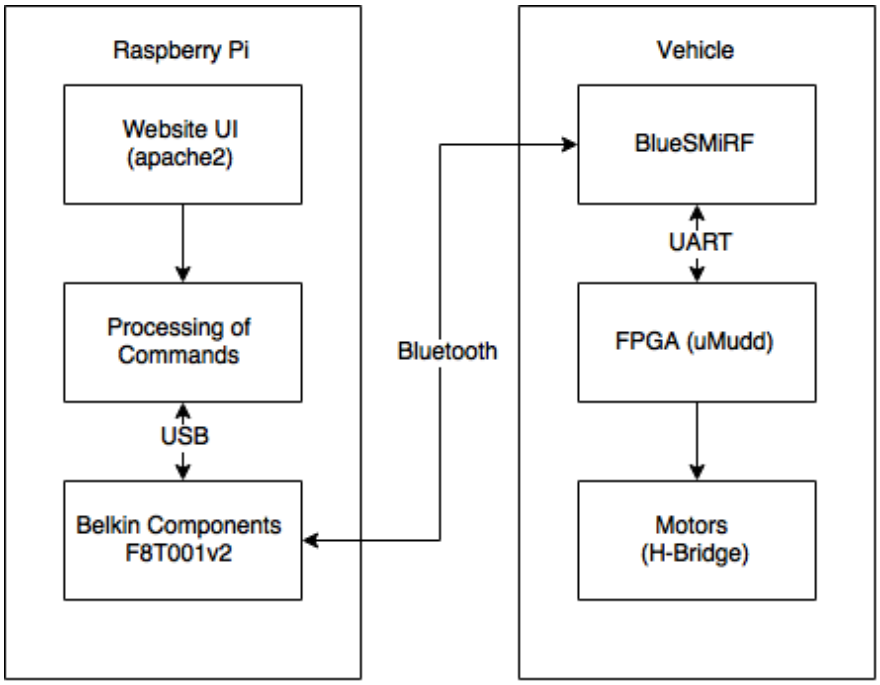


Figure 1: An overview of the system. The system is comprised of two major subsystems: the Raspberry Pi 2 controller and the vehicle, which is controlled by the μMudd board.

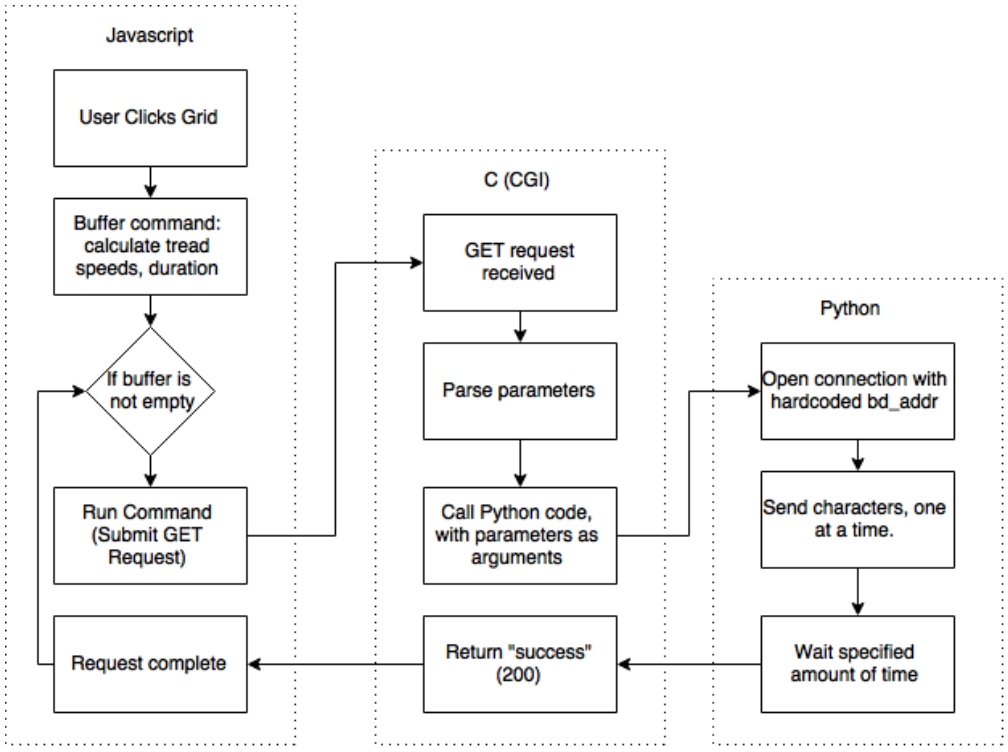


Figure 2: The flow of control and data through the Pi.

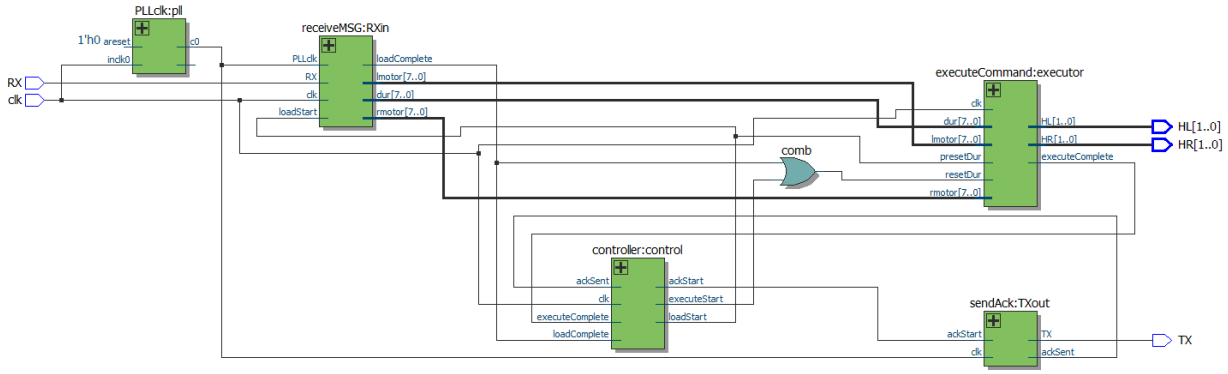


Figure 3: The schematic of the hardware in the FPGA. More detailed diagrams are shown in Appendix D.

The clock used to interface with the BlueSMiRF is implemented using a PLL that oversamples the 112.5 Kbaud frequency by a factor of 8. This oversampler determines if there is an incoming message. The actual sampling of the BlueSMiRF's TX line is accomplished using a frequency divider that allows for sampling at the correct rate. The divider's phase can be controlled in order to ensure that the sampling of the line is as close to the center of the transmission's clock as possible. As there are three characters in each command, the code holds the system in the receiveMSG state until all three characters have been loaded.

The FPGA executes the command received by controlling the two motors via the H-Bridges on the μMudd board. Each command consists of a PWM setting for each motor and a duration for which the motors should be running. A counter is used to create a reference clock for PWM; the power levels are referenced against this counter to determine the correct duty cycle. To prevent the vehicle from running indefinitely, the timer stops incrementing when the requested duration is reached, and a signal is output that is used to cut power to the motors. Each LSb of the duration char corresponds to roughly one-tenth of a second duration.

Once the requested duration has been reached and power to the motors cut, the FPGA transmits the character 'A' back to the BlueSMiRF as an ACK code. After this ACK has been sent, the FPGA will cycle around to the receiveMSG state again.

At this time, the μMudd board interfaces with the BlueSMiRF. All parts of the code have been written; however there are still critical bugs in the code to be worked out. The receiveMSG and controller modules have been independently verified in ModelSim. The other two main submodules will be independently verified shortly, and then the whole system will be tested together.

Figure 3 shows a high level block diagram of the FPGA, with more detailed diagrams in Appendix D.

Vehicle

All of the parts for the vehicle have been ordered and received. The vehicle has yet to be assembled, as there will be no machining or fabrication involved at this point.

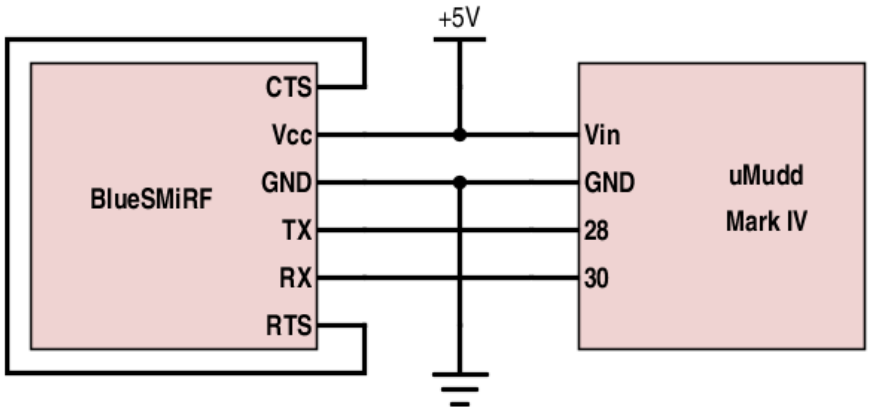


Figure 4: The schematic of the circuit on the breadboard. In simple applications, the CTS and RTS pins of the BlueSMiRF are connected together, and the only pins that are connected to the FPGA are TX and RX.

Appendix A: Website

Code

```
<!DOCTYPE html>
<html>
<head>
  <title>E155 Final Project</title>
  <meta http-equiv="content-type" content="text-html;charset=utf-8">
  <link rel="stylesheet"
    href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/
      css/bootstrap.min.css">
  <script type="text/javascript"
    src="http://code.jquery.com/jquery.min.js"></script>
  <style>
    .grid { margin:1em auto; border-collapse:collapse }
    .grid td {
      cursor:pointer;
      width:60px; height:60px;
      border:1px solid #ccc;
      text-align:center;
      font-family:sans-serif; font-size:13px
    }
    .grid td.clicked {
      background-color:yellow;
      font-weight:bold; color:red;
    }
  </style>
</head>
<body>
```

```
<div id="bg"></div>
<div class="container">
  <div class="jumbotron">
    <h1>E155 Final Project</h1>
    <p>Microprocessors at Harvey Mudd College, Fall 2015</p>
    <p>Aaron Rosen and Alex Rich</p>
  </div>
  <div class="row">
    <div class="col-sm-4">
      <div class="panel">
        <h3>Send Command</h3>
        <form action="cgi-bin/inputChars" method="GET">
          <div class="form-group">
            <label for="l">Left Power:</label>
            <input type="text" class="form-control"
              name="l" placeholder="[-127, 127]">
          </div>
          <div class="form-group">
            <label for="r">Right Power:</label>
            <input type="text" class="form-control"
              name="r" placeholder="[-127, 127]">
          </div>
          <div class="form-group">
            <label for="l">Time</label>
            <input type="text" class="form-control"
              name="t" placeholder="1/10 seconds, [0, 255]">
          </div>
            <input type="submit" class="btn btn-default"
              value="Submit Command">
          </form>
        </div>
      </div>
      <div class="col-sm-4">
        <div class="panel">
          <h3 id="pac">Point and Click</h3>
          <div id="g"></div>
        </div>
      </div>
      <div class="col-sm-4">
        <div class="panel">
          <h3 id="commandListHeader">Commands</h3>
          <div id="commands"></div>
          <!--<a href="index.html">Go Back >></a>-->
        </div>
      </div>
    </div>
  </div>
</div>
```

```
</body>

<script>
var commandList = [];
var isExecuting = false;

var lastClicked;

var grid = clickableGrid(5,5,function(el, row, col, i){
  var l = Math.floor((Math.random() * 255) + 1);
  var r = Math.floor((Math.random() * 255) + 1);
  var t = Math.floor((Math.random() * 255) + 1);

  el.className = 'clicked';
  if (el == lastClicked) {
    return;
  }
  if (lastClicked) lastClicked.className = '';
  lastClicked = el;

  commandList.push({l:l,r:r,t:t});
  executeCommand();
})

function executeCommand() {
  updateList();
  if (commandList.length == 0) {
    return;
  } else if (isExecuting) {
    // If we're currently executing a command, check back in 0.1 sec
    setTimeout(function () {
          executeCommand();
      }, 100);
    return;
  }
  isExecuting = true

  var command = commandList.shift();
  console.log("sending command: ");
  console.log(commandToString(command));

  var xmlHttp = new XMLHttpRequest();
  xmlHttp.onreadystatechange = function() {
    if (xmlHttp.readyState == 4 && xmlHttp.status == 200) {
      isExecuting = false;
      executeCommand();
    } else if (xmlHttp.readyState == 4) {
      document.getElementById('pac').innerHTML =
```

```
        xmlHttp.status.toString().concat(": There was an error!");
      document.getElementById('pac').style.color = 'red';
      setTimeout(function () {
          document.getElementById('pac').innerHTML = "Point and Click";
          document.getElementById('pac').style.color = 'black';
      }, 2000);
    }
  }
  var url = "cgi-bin/inputChars";
  url = url.concat("?l=", command.l.toString(), "&r=",
    command.r.toString(), "&t=", command.t.toString());
  xmlHttp.open("GET", url, true); // true for asynchronous
  xmlHttp.send(null);
}

function updateList() {
  var str = "Current commands in buffer: <br>"
  for(var i=0;i<commandList.length;i++){
    str = str.concat((i+1).toString(), " -- ",
      commandToString(commandList[i]), "<br>");
  }
  str = str.concat("");
  document.getElementById('commands').innerHTML = str;
}

function commandToString(command) {
  var str = ""
  str = str.concat("L: ", command.l.toString(), ", R: ",
    command.r.toString(), ", T: ", command.t.toString());
  return str;
}

document.getElementById('g').appendChild(grid);

function clickableGrid(rows, cols, callback) {
  var i = 0;
  var grid = document.createElement('table');
  grid.className = 'grid';
  for (var r=0;r<rows;++r){
    var tr = grid.appendChild(document.createElement('tr'));
    for (var c=0;c<cols;++c){
      var cell = tr.appendChild(document.createElement('td'));
      cell.innerHTML = 'X';
      if (i== Math.floor(rows * cols / 2)) {
        cell.className = 'clicked';
        lastClicked = cell;
      }
      ++i;
```

```
      cell.addEventListener('click',(function(el,r,c,i){
        return function(){
          callback(el,r,c,i);
        }
      })(cell,r,c,i),false);
    }
  }
  return grid;
}
</script>
</html>
```

Result

Figure 5 shows the website user interface that appears when accessing the raspberry pi. The grid is clickable, and the buffer of commands is listed on the right side of the page. A user can also submit a direct command using the form on the right.

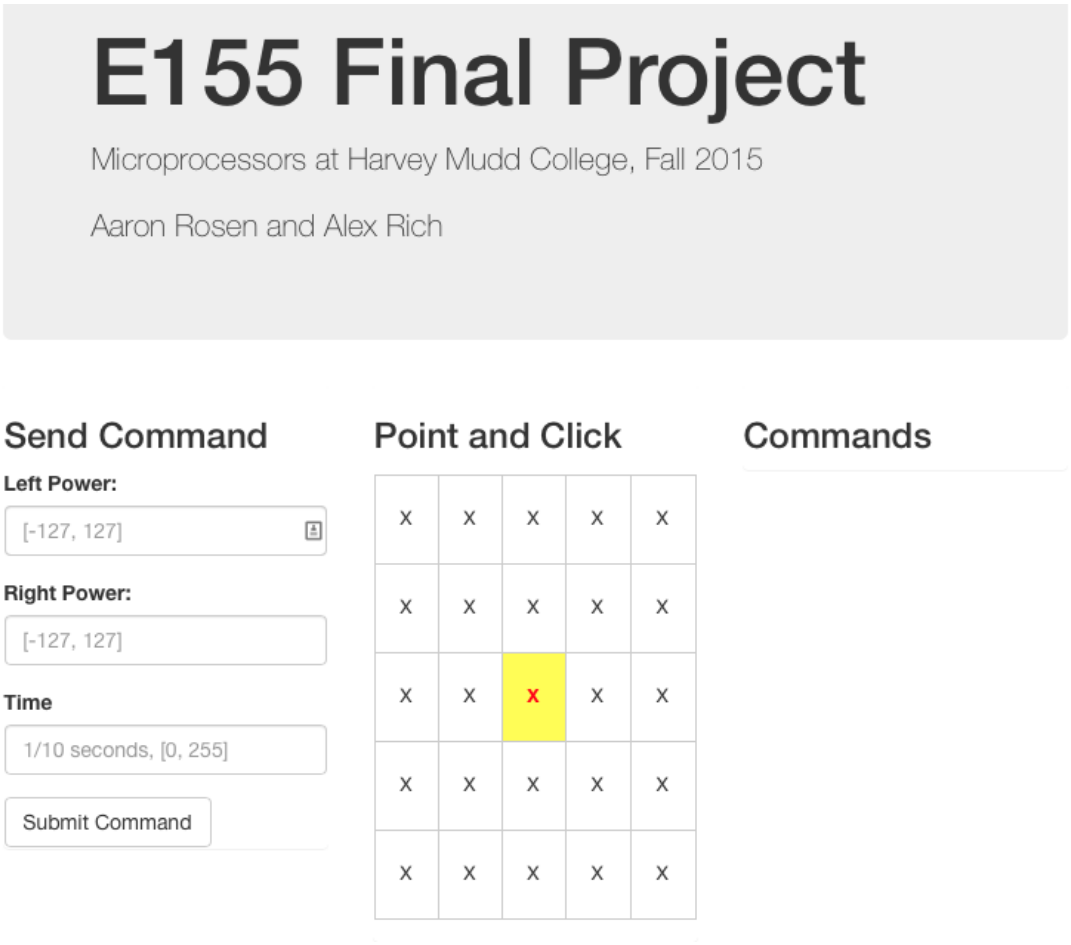


Figure 5: The website displayed to a user controlling the robot.

Appendix B: Raspberry Pi2 Code

CGI Code

```
// Control the tank given the basic commands.
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <python2.7/Python.h>

char convertToChar(int value)
{
  char byte;
  if (value < 0) {
    value = value * -1;
    byte = (char) value;
    byte = byte | 0x80;
  } else {
    byte = (char) value;
  }

  return byte;
}

void printBits(char byte)
{
  int i;
  for(i = 7; 0 <= i; i --)
    printf("%d", (byte >> i) & 0x01);
}

void callPython(char l, char r, char t)
{
  FILE* file;
  int argc;
  char * argv[4];

  char a[1] = {t};
  char b[1] = {r};
  char c[1] = {t};

  argc = 4;
  argv[0] = "sendData.py";
  argv[1] = a;
  argv[2] = b;
  argv[3] = c;

  Py_SetProgramName(argv[0]);
```

```
  Py_Initialize();
  PySys_SetArgv(argc, argv);
  file = fopen("sendData.py","r");
  PyRun_SimpleFile(file, "sendData.py");
  Py_Finalize();

  return;
}

bool sendData(char l, char r, char t)
{
  bool success = false;
  callPython(l, r, t);
  // TODO send the data over bluetooth!
  return success;
}

int main(void)
{
  char *data;
  int il, ir, it;
  char l, r, t;
  printf("%s%c%c\n", "Content-Type:text/html;charset=iso-8859-1",13,10);
  printf("<h1>Robot Under Control (Debug Page)</h1>");

  data = getenv("QUERY_STRING");

  if (data == NULL)
    printf("<p>Error! Error in passing data from form to script.</p>");
  else if (sscanf(data, "l=%d&r=%d&t=%d", &il, &ir, &it) != 3) {
    printf("<P>Error! Invalid data.</p>");
    printf("<p>Data Received: %s, Number of values located: %d</p>",
      &data, sscanf(data, "l=%d&r=%d&t=%d", &il, &ir, &it));
    printf("l: %d, r: %d, t: %d", il, ir, it);
  }
  else if (il > 127 || il < -127 || ir > 127 || ir < -127 || it > 255)
    printf("<p>The data you entered was invalid. Please remember size
      restrictions</p>");
  else {
    // TODO Convert these to sign magnitude!
    l = convertToChar(il);
    r = convertToChar(ir);
    t = (char) it;

    // Debug Output
    printf("<P>Raw values entered: <br>r: %d <br>l: %d <br>t: %d</p>",
     ir, il, it);
    printf("<P>Chars to be transmitted (may be invisible):
```

```
      <br>r: %c <br>l: %c <br>t: %c</p>", r, l, t);
    printf("<p>Bit Data: <br>r: ");
    printBits(r);
    printf("<br>l: ");
    printBits(l);
    printf("<br>t: ");
    printBits(t);
    printf("</p>");

    bool success = sendData(l, r, t);

    printf("Data sent: %s.", success ? "true" : "false");
  }

  printf("<br><br> <a href=\"../final.html\">Go Back</a>");
  return 0;
}

/*
sudo chown root:www-data /usr/lib/cgi-bin/inputChars
sudo chmod 010 /usr/lib/cgi-bin/inputChars
sudo chmod u+s /usr/lib/cgi-bin/inputChars

packages installed: bluez, python-dev, python-bluez
(maybe bluetooth is needed?)
*/
```

Python Code

```
from bluetooth import *
import sys
import time

def sendChars(data, bd_addr, port):
  sock = BluetoothSocket( RFCOMM )
  sock.connect((bd_addr, port))

  for c in data:
    sock.send(c)
    time.sleep(3)
  sock.close()

def printData(data):
  print 'Cleaned argument list:', str(data)
  print "<br>"
  dataToSend = ""
  for c in data:
    dataToSend += c
```

```
  print "Data to send: " + dataToSend + "<br>"

if len(sys.argv) > 1:
  print "<h4>ENTER PYTHON</h4>"
  data = [l[0] for l in sys.argv[1:]]
  printData(data)
  bd_addr = "00:06:66:4F:DC:B1"
  port = 1
  sendChars(data, bd_addr, port)
  print "<h4>EXITED PYTHON SUCCESSFULLY</h4>"

else:
  data = ["a"]
  bd_addr = "00:06:66:4F:DC:B1"
  port = 1
  sendChars(data, bd_addr, port)
```

Appendix C: SystemVerilog Code on FPGA

```
module VehicleControl(input logic clk,
                      input logic RX,
                      output logic TX,
                      output logic[1:0] HL,HR);
  //top level module
  logic[7:0] lmotor,rmotor,dur;
  logic loadStart,loadComplete;
  //executeStart should pulse when starting rather than staying high
  logic executeStart,executeComplete;
  logic ackStart,ackSent;
  logic pllclk;
  logic reset,locked;

  PLLclk pll(reset,clk,pllclk,locked);

  controller control(clk,loadComplete,executeComplete,ackSent,
    loadStart,executeStart,ackStart); //datapath controller

  receiveMSG RXin(clk,pllclk,RX,loadStart,lmotor,rmotor,dur,loadComplete);
  executeCommand executor(clk,(loadComplete | executeStart),loadStart,
    lmotor,rmotor,dur,executeComplete,HL,HR);
  sendAck TXout(pllclk,ackStart,ackSent,TX);

endmodule

module controller(input logic clk,
            input logic loadComplete,
            input logic executeComplete,
            input logic ackSent,
```

```
            output logic loadStart,
            output logic executeStart,
            output logic ackStart);
  //datapath controller
  reg[2:0] state;
  logic execute,executeDelayed;
  always_ff @(posedge clk)
    begin
      case(state)
        3'b001: begin
                if(loadComplete)    state<=3'b010; //state becomes execute
                else              state<=3'b001; //state remains load
              end
        3'b010: begin
                if(executeComplete)   state<=3'b100; //state becomes ack
                else              state<=3'b010; //state remains execute
              end
        3'b100: begin
                if(ackSent)         state<=3'b001; //state becomes load
                else              state<=3'b100; //state remains ack
              end
        default:                  state<=3'b001; //default to load
      endcase
    end
  assign {ackStart,execute,loadStart}=state; //delegate signals accordingly
  flop executeDelay(clk,execute,executeDelayed);
  assign executeStart = execute & ~ executeDelayed;
endmodule

module receiveMSG(input logic clk,PLLclk,
          input logic RX,loadStart,
          output logic[7:0] lmotor,rmotor,dur,
          output logic loadComplete);
  //top level for message recive subsystem
  logic[7:0] char;
  logic RXdone;
  logic resetTrigger;
  UARTRX receiveChar(clk,PLLclk,RX,loadStart,char,RXdone);
  shift3rst receiveCtr(loadStart,RXdone,resetTrigger,loadComplete);
  always_ff @(posedge RXdone)
    begin
      if(loadStart) {lmotor,rmotor,dur}={rmotor,dur,char};
    end
  flop #(1) loadCompleteDelay(clk,loadComplete,resetTrigger);
endmodule

module executeCommand(input logic clk,
               input logic resetDur,presetDur,
```

```
                input logic [7:0]lmotor,rmotor,dur,
                output logic executeComplete,
                output logic [1:0] HL,HR);
  //top level for message execute subsystem
  logic LPWM,RPWM;

  durcheck duration(dur,clk,resetDur,presetDur,executeComplete);
  pwm lmotorPWM(lmotor[6:0],clk,resetDur,LPWM);
  pwm rmotorPWM(rmotor[6:0],clk,resetDur,RPWM);
  hBridgeIn LHbridge(LPWM,executeComplete,lmotor[7],HL);
  hBridgeIn RHbridge(RPWM,executeComplete,rmotor[7],HR);
endmodule

module sendAck(input logic clk,
          input logic ackStart,
          output logic ackSent,
          output logic TX);
  //top level for ack send subsystem
  UARTTX sendChar(clk,ackStart,TX,ackSent);

endmodule

module UARTTX(input logic clk,
          input logic ackStart,
          output logic TX,
          output logic msgSent);
  // UART TX Pin
  // ACK is "A" (0x41),
  // msg is 11'b0_0100_0001_11 = 11'b001_0000_0111 = 11'h107
  logic resetTrigger;
  logic clk2;
  logic[10:0]msg;
  logic[3:0]count; //keeps track of the number of bits sent
  slowclk baudrate(clk,1'b1,clk2);
  always_ff @(posedge clk2)
    begin
      //this is an 11-bit shift register
      if(ackStart) {msg[9:0],msg[10]}={msg};
      //reset message loop to default position
      else {msg[9:0],msg[10]} = {10'h107,1'h0};
    end
  assign TX = msg[0];
  timer #(4) bitCount(clk2,resetTrigger,count);
  assign msgSent = (count == 11); //message send high after 11th bit sent
  flop #(1) msgSentDelay(clk,msgSent,resetTrigger);

endmodule
```

```
module UARTRX(input logic clk,PLLclk,
        input logic RX,
        input logic loadStart,
        output logic[7:0] char,
        output logic done);
  //UART RX Pin
  logic[3:0] validCheck;
  logic valid;
  logic UARTclk;
  logic[1:0] stopBits;
  logic resetTrigger;
  logic startBit;
  logic loadStartDelayed,doneDelayed;

  always_ff @(posedge clk,posedge done)
    begin
      if(done) valid <= 0;
      else valid <= valid | (~|(validCheck));
    end
  shift4 sampler(RX,PLLclk,validCheck);
  slowclk baudrate(PLLclk,valid,UARTclk);
  always_ff @(posedge UARTclk,posedge resetTrigger)
    begin
      if(resetTrigger) {done,startBit,char,stopBits} = 12'h001;
      //this is an 12-bit shift register
      else {done,startBit,char,stopBits}={startBit,char,stopBits,RX};
    end
  delay #(1) doneDelay(clk,done,doneDelayed);
  delay #(1) loadStartDelay(clk,loadStart,loadStartDelayed);
  assign resetTrigger = doneDelayed | (~loadStartDelayed & loadStart);
endmodule

module hBridgeIn(input logic pwr,done,direction,
         output logic[1:0] out);
  //cuts power to H-Bridge when done is asserted
  logic[1:0]sig;
  assign sig[0]=0;
  assign sig[1] = pwr & ~done;
  //direction is sign in sign/mag number
  assign out = direction?{sig[0],sig[1]}:sig;
endmodule

module pwm(input logic [6:0] power,
       input logic clk,reset,
       output logic wave);
  //Takes in an input signal and outputs corresponding PWM signal
  logic [6:0] count;
  timer #(7) pwmTimer(clk,reset,count);
```

```
  assign wave = (power > count);
endmodule

module durcheck(input logic[7:0] dur,
        input logic clk,reset,preset,
        output logic done);
  //checks the duration and cuts power to the wheels when done
  logic[29:0] durTime;
  always_ff @(posedge clk,posedge reset,posedge preset)
    begin
      if(reset) durTime <=0;
      else if(preset) durTime <= {dur,22'b0};
      else if(done) durTime <= durTime;
      else durTime <= durTime + 1'b1;
    end
  assign done = &(dur & durTime[29:22]);
endmodule

module shift3rst(input logic in,clk,reset,
        output logic out);
  //3-register shift register with reset
  logic d1,d2;
  always_ff @(posedge clk,posedge reset)
    if(reset)
      begin
        d1 <=0;
        d2 <=0;
        out <=0;
      end
    else
      begin
        d1<=in;
        d2<=d1;
        out<=d2;
      end
endmodule

module shift4(input logic in,clk,
        output logic[3:0] out);
  //4-register shift register, outputs all shifted bits
  always_ff @(posedge clk)
    begin
      out[0]<=in;
      out[1]<=out[0];
      out[2]<=out[1];
      out[3]<=out[2];
    end
endmodule
```

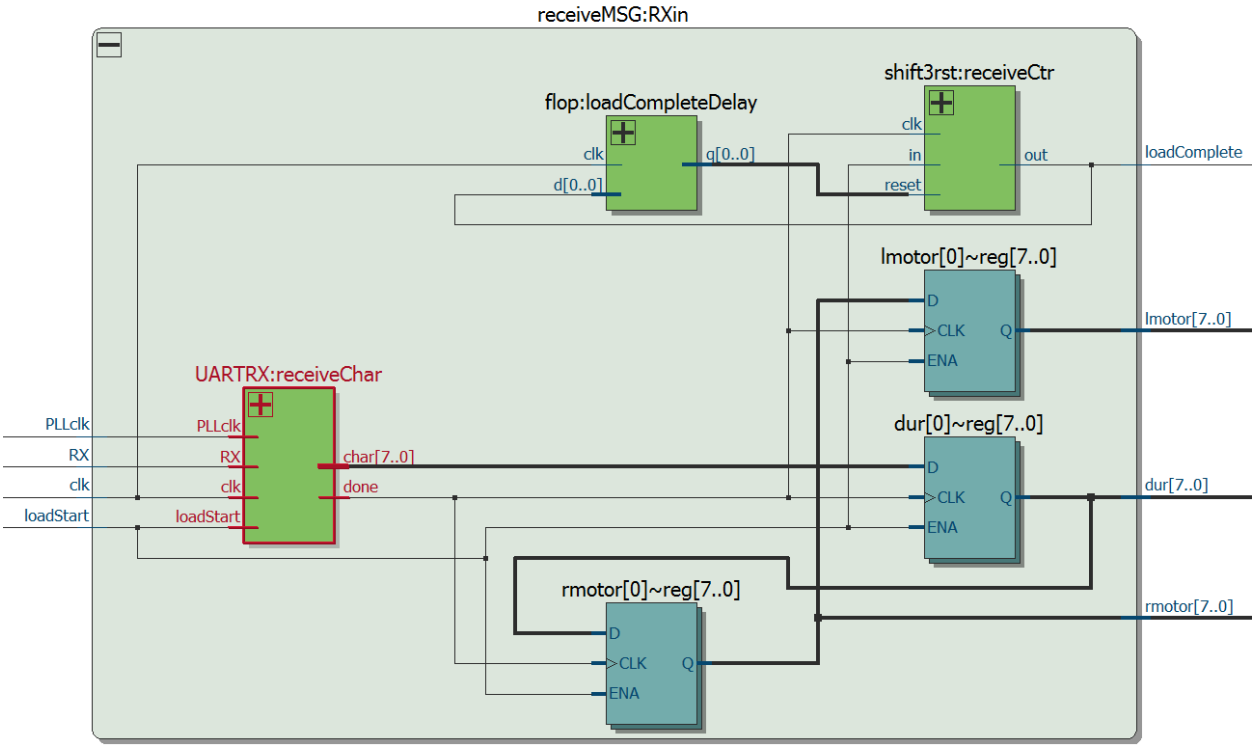
```
module slowclk(input logic clk,valid,
          output logic clk2);
  //creates a second slow timer that is reliant on valid for centering
  logic [2:0]count;
  always_ff  @(posedge clk)
    begin
      //if the signal is valid,
      //increment the counter normally
      if(valid) count <= count + 3'h1;
      else
        // if the signal is not valid, hold the slow clock
        // right before the transition
        begin
          count[2] <= 0;
          count[1] <= 1;
          count[0] <= 1;
        end
    end
  assign clk2=count[2];
endmodule


module timer #(parameter WIDTH=8)
        (input logic clk,reset,
         output logic [WIDTH-1:0] timeout);
  //a WIDTH-bit timer
  always_ff @(posedge clk,posedge reset)
    begin
      if(reset) timeout <= 0;
      else timeout <= timeout + 1'b1;
    end
endmodule

module flop #(parameter WIDTH=1)
         (input logic clk,
          input logic [WIDTH-1:0] d,
          output logic [WIDTH-1:0] q);
  always_ff @(posedge clk)
    begin
      q <= d;
    end
endmodule

module delay #(parameter WIDTH=1)
         (input logic clk,
          input logic [WIDTH-1:0] d,
          output logic [WIDTH-1:0] q);
```

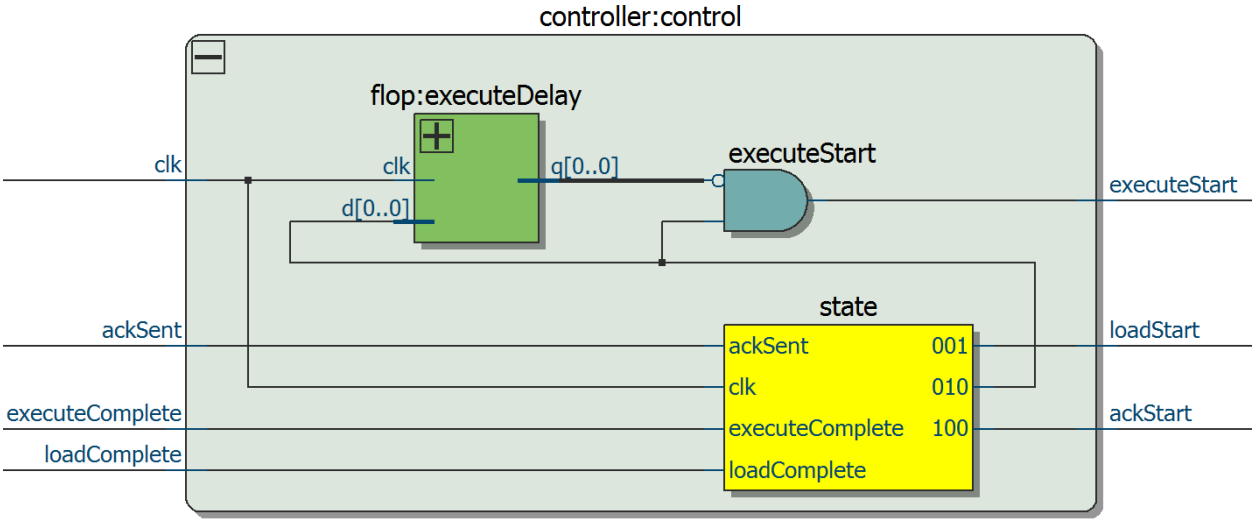
```
  logic[WIDTH-1:0] p;
  always_ff @(posedge clk)
    begin
      p <= d;
      q <= p;
    end
endmodule

/*
module Testbench();

  //TEST: receiveMSG
  //standalone module verified: 11/21 @23:07
  //logic needed:
  logic clk,PLLclk;
  logic RX;
  logic loadStart;
  logic[7:0] lmotor,rmotor,dur;
  logic loadComplete;
  receiveMSG dut(clk,PLLclk,RX,loadStart,lmotor,rmotor,dur,loadComplete);
  //chars: 0x95, 0xB6, 0x35
  always
    begin
      clk <=1; #50; clk <=0; #50;
    end

  always
    begin
      PLLclk <=1; #1100; PLLclk <=0; #1100;
    end

  initial
    begin
      RX=1;
    end

  always
    begin
      #100;
      loadStart=0;
      #100
      loadStart=1;
      #1000;
      RX=0; //start
      #17600;
      RX=1; //1
      #17600;
```

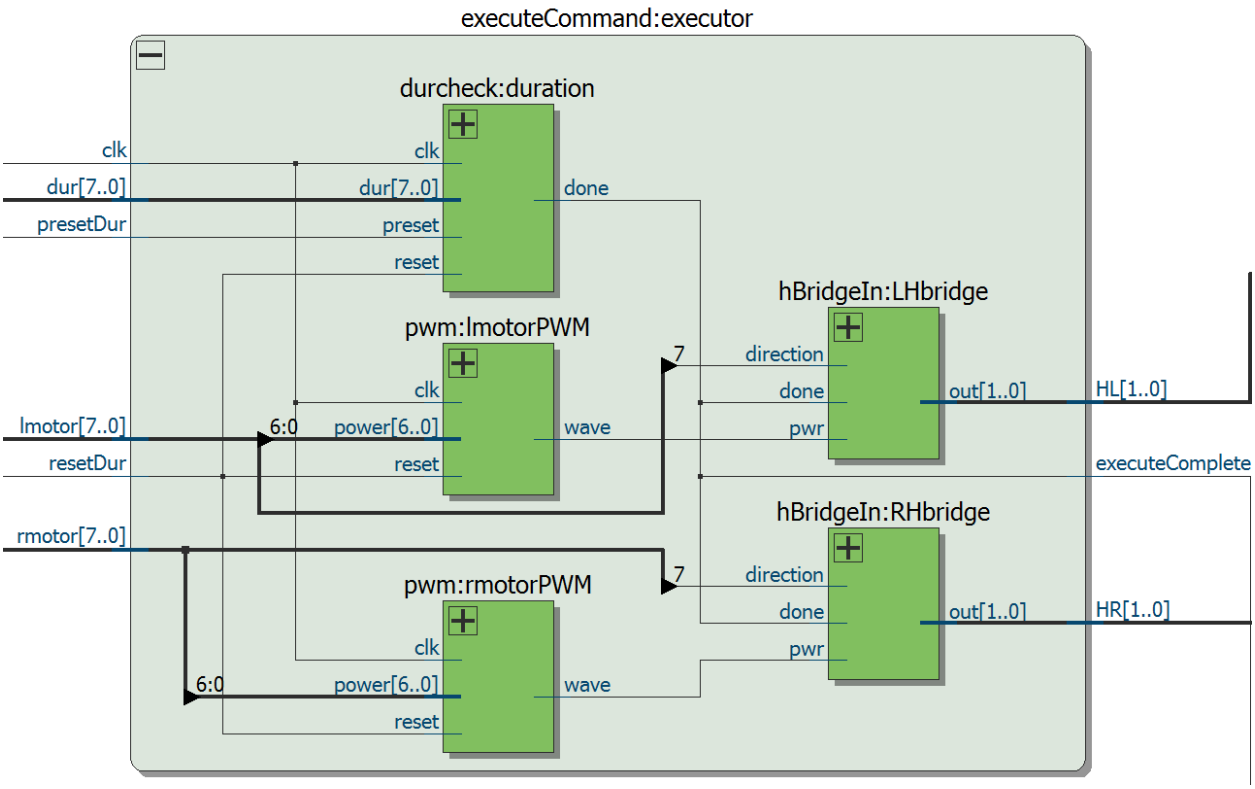
```
RX=0; //2
#17600;
RX=0; //3
#17600;
RX=1; //4
#17600;
RX=0; //5
#17600;
RX=1; //6
#17600;
RX=0; //7
#17600;
RX=1; //8
#17600;
RX=1; //stop
#17600;
#17600;

#100000;
RX=0; //start
#17600;
RX=1; //1
#17600;
RX=0; //2
#17600;
RX=1; //3
#17600;
RX=1; //4
#17600;
RX=0; //5
#17600;
RX=1; //6
#17600;
RX=1; //7
#17600;
RX=0; //8
#17600;
RX=1; //stop
#17600;
#17600;

#100000;
RX=0; //start
#17600;
RX=0; //1
#17600;
RX=0; //2
#17600;
```

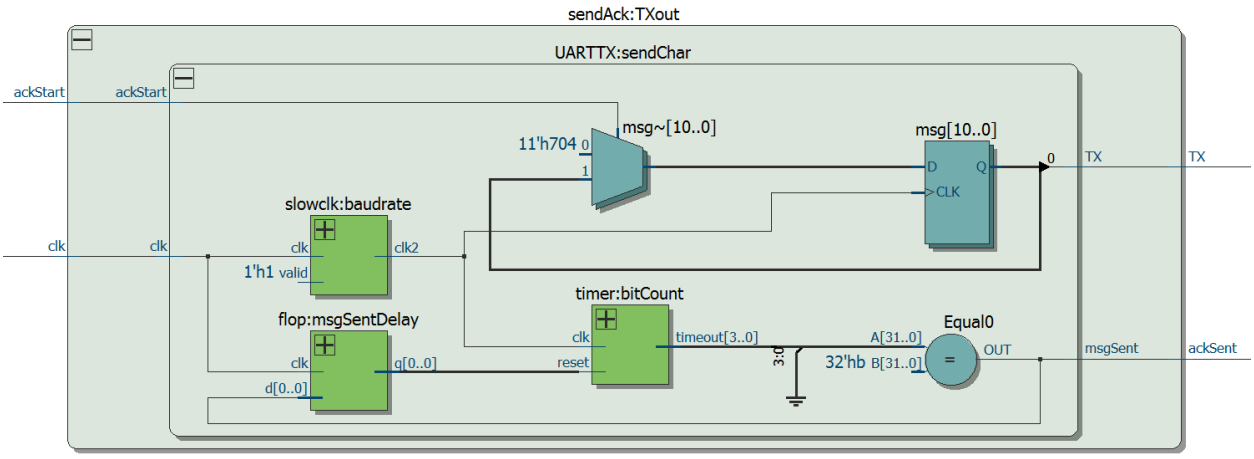
```
        RX=1; //3
        #17600;
        RX=1; //4
        #17600;
        RX=0; //5
        #17600;
        RX=1; //6
        #17600;
        RX=0; //7
        #17600;
        RX=1; //8
        #17600;
        RX=1; //stop
        #17600;
        #17600;
        #100000;
    end
endmodule
*/
```

Appendix D: FPGA Diagrams




controller:control
flop:executeDelay
clk
clk
d[0..0]
q[0..0]
executeStart
executeStart
state
ackSent
ackSent
001
clk
010
executeComplete
100
loadComplete
loadStart
ackStart


executeCommand:executor
durcheck:duration
clk
dur[7..0]
presetDur
clk
dur[7..0]
preset
reset
done
hBridgeIn:LHbridge
pwm:lmotorPWM
7
direction
clk
done
out[1..0]
HL[1..0]
lmotor[7..0]
6:0
power[6..0]
wave
pwr
resetDur
reset
executeComplete
hBridgeIn:RHbridge
rmotor[7..0]
7
direction
pwm:rmotorPWM
done
out[1..0]
HR[1..0]
clk
pwr
6:0
power[6..0]
wave
reset



Appendix E: Bill of Materials

This is the bill of materials for the project.

Item	Description	Source	Cost
Tracked Vehicle Chassis Kit	Chassis for the tank, includes treads and frame.	Amazon	15.39
Tamiya 70168 Double Gearbox	Gives tank flexibility to turn by controlling each tread independently.	Amazon	11.99
μMudd Board	Controls the vehicle.	E155	0.00
Raspberry Pi 2	Provides website interface and sends commands to vehicle.	E155	0.00
BlueSMiRF	Wirelessly communicate via Bluetooth between Pi and μMudd board.	E155	0.00
Belkin Components FT8001	Send bluetooth data from Pi.	E155	0.00
2X TrustFire 14500	Li Ion battery, 3.7 V	Aaron Rosen	0.00