Harvey Mudd College

Engineering 155
Microprocessor Systems: Design and Application

Internet-Controlled AGV

Authors:
Aaron Rosen
Alex Rich

Professors:
David Harris
Matthew Spencer

December 11, 2015

Abstract

Contents

1 Introduction

The Internet-Controlled Autonomous Ground Vehicle is a treaded chassis that holds an FPGA with bluetooth listening capabilities. A companion website allows a user to interact with the vehicle by commanding it to travel it to various places. The motivation behind this project was to build a tank that could navigate within a map and shoot NERF darts. By building a tank that is remotely controlled over the internet, the more difficult part of this concept is realized.

The project involves a website hosted by Apache2 on the Raspberry Pi 2, which sends commands from the a bluetooth dongle to a bluetooth receiver (BlueSMiRF) that is connected to the FPGA. The FPGA drives two motors. Each component is described further in the following sections.

2 New Hardware

2.1 BlueSMiRF

The Raspberry Pi and FPGA communicate with each other using a set of Bluetooth modules. The FPGA is wired to a SparkFun BlueSMiRF module. The Raspberry Pi has a Belkin FT8001 Bluetooth dongle

The BlueSMiRF is a Bluetooth RF module that has a UART pinout to interface with. To set up the interface, the device is powered with a single-sided +5V/GND power source, in

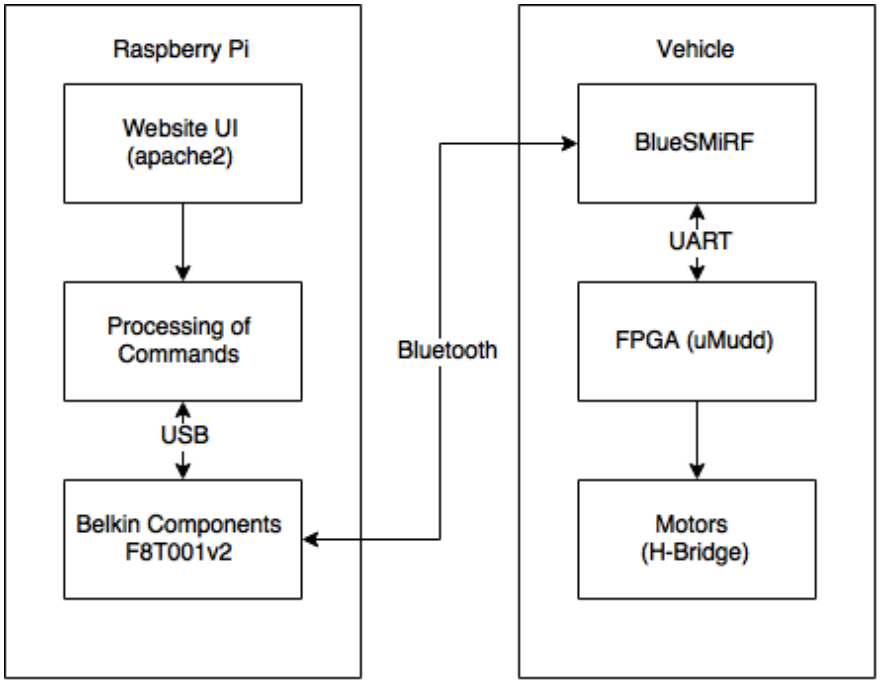


Figure 1: An overview of the system. The system is comprised of two major subsystems: the Raspberry Pi 2 controller and the vehicle, which is controlled by the μMudd board.

this case the same 5V output from the voltage regulator and ground line on the breadboard. RTS and CTS are wired together, and the TX and RX pins on the BlueSMiRF are connected to the pins of the FPGA that are assigned to RX and TX, respectively. (how to find address and how to config)

The Belkin FT8001 is a USB Bluetooth dongle that can be attached directly to the Raspberry Pi's USB port. (how to config)

To power the vehicle, the FPGA's H-Bridge outputs 1/2 and 3/4 are connected to two Pololu Brushed DC motors. The motors operate at 3-12V volts, which allows the same voltage regulator to power the input of the FPGA and the motors.

2.2 Vehicle

The vehicle is a Tamiya Tracked vehicle modified to accept a double gear box. The motors are wired directly to the H-Bridge outputs. The battery case is wired directly to a 5V regulator, which powers the system. The breadboard is mounted on three sheet metal stands.

3 Schematics

Figure 2 shows the layout of the breadboard. The two main components are the FPGA and the BlueSMiRF. The 7.4 V power source is comes from two TrustFire 3.7 V batteries, and is regulated with the 5V regulator. The switch makes for easy powering on and off of the vehicle.

4 Raspberry Pi

The Raspberry Pi 2 serves several functions. It hosts an Apache2 website accessible via the internet and contains code to submit commands to a bluetooth dongle. These two functions are integrated such that a user interacts with the website, indirectly sending commands over bluetooth. Figure 3 show the flow of data and control through the Pi.

The languages in use on the Pi are HTML, JavaScript, C, and Python.

4.1 Website

The website 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 commands currently buffered for sending. The code for the website is shown in Appendix 9.3.3. 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 calculates 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. The current algorithm involves first moving north/south, turning, moving east/west, and finally turning to reorient itself vertically. Once the commands are generated, 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.

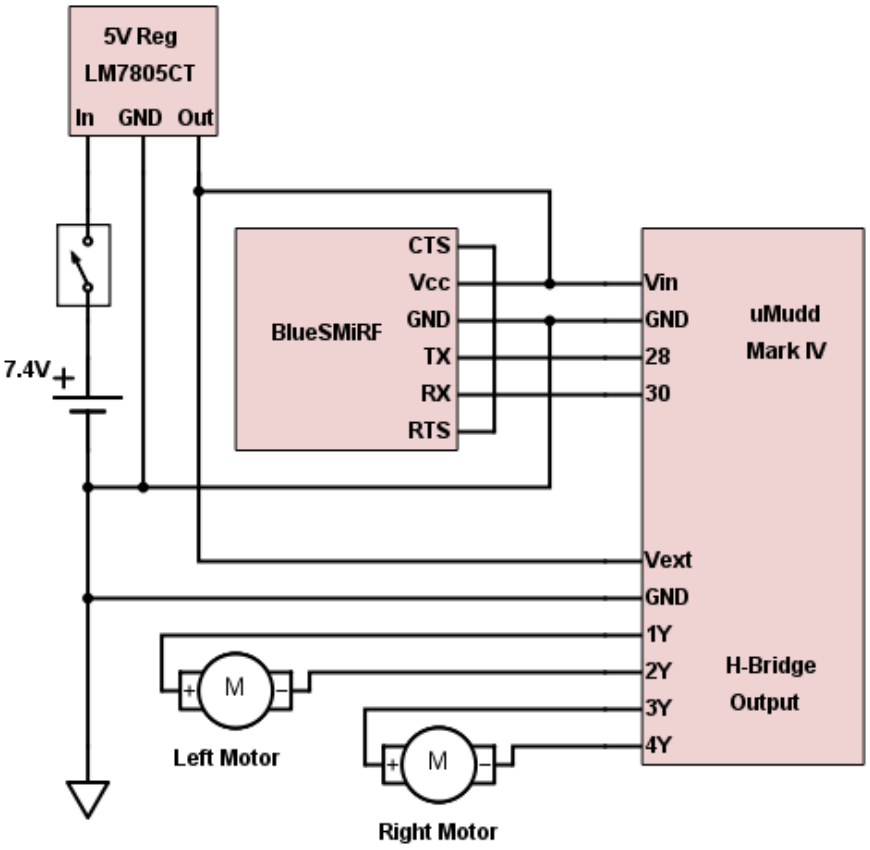


Figure 2: The components on the breadboard. The H-Bridge Output are PWM signals.

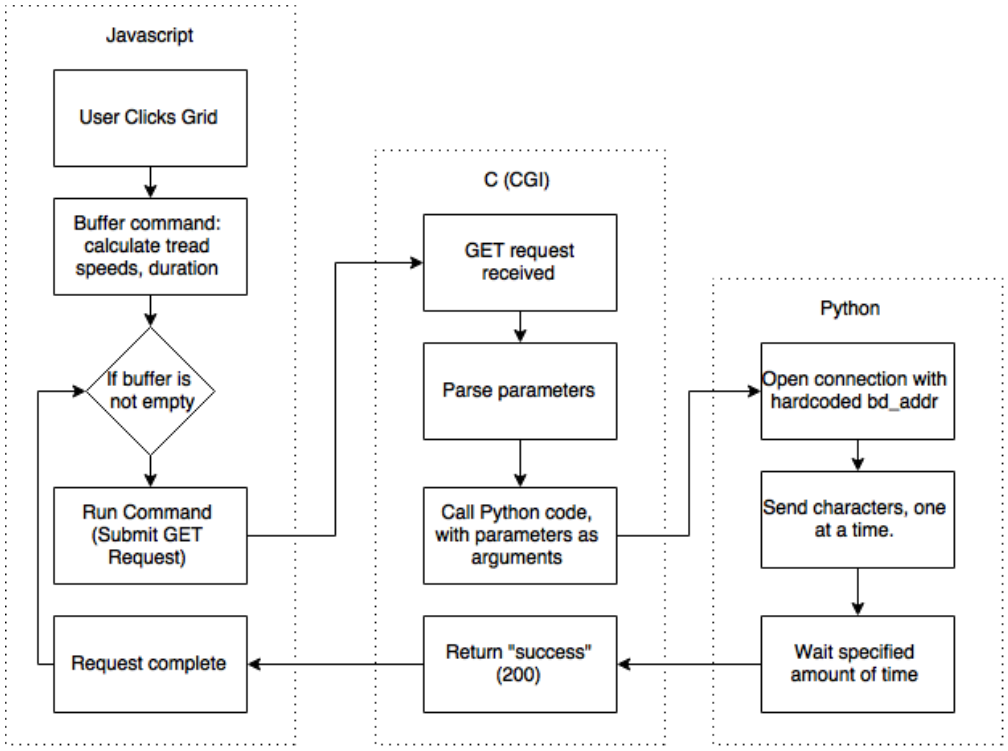


Figure 3: The flow of control and data through the Pi.

4.2 Python/C

After receiving a GET request, the common gateway interface (CGI) reads the input parameters (three integers) and converts them into a format that the vehicle understands. This involves converting the numbers to sign/magnitude bit representations. Because of how the UART works, the C also reverses the bits.

Then, the C program calls a Python script. The Python script utilizes the `bluetooth` module to allow sending data using the bluetooth dongle. When called, the Python script sends the commands, one at a time, to the robot. The system sleeps for the amount of time to give the vehicle enough time to move. The code on the Pi is shown in Appendix 9.3.

To send data over bluetooth, the address of the BlueSMiRF must be known. This can be done with a scan of all devices in the area. The scan is initiated with the terminal command

```
hcitool scan
```

Another way to do it is to scan in the Python script and match the name of each device in the area to the device's name. In any case, something has to be hardcoded into the Python script to make a connection with the BlueSMiRF. To save time when connecting to the vehicle, the address is hardcoded and thus there is no need to scan.

Sending data in Python with the `bluetooth` module is fairly simple.

```
sock = BluetoothSocket( RFCOMM )
```

```
try:
    sock.connect((bd_addr, port))
except:
    # Address is likely in use, so close it.
    sock.close()
    return

# Send each character individually
for c in data:
    sock.send(c)

sock.close()
```

First a connection is made, data is sent over it, and the connection is closed. This could be done by only connecting once, however the nature of the website and the desired interaction made connecting and disconnecting for each command a simpler and understandable solution.

5 FPGA Design

The FPGA reads data from the BlueSMiRF using UART hardware coded in SystemVerilog, processes and executes the command, and then sends an acknowledgment back to the Raspberry Pi. It is constructed as a controller-datapath pair with three main submodules in the datapath - receiveMSG, executeCommand, and sendAck. The SystemVerilog code installed on the FPGA is shown in Appendix 9.2. The FPGA and BlueSMiRF are the only two electrical components on the breadboard. Two motors are connected to the μMudd board's H-bridge screw terminals. The schematic is shown in figure.

The clock used to interface with the BlueSMiRF is implemented using a PLL that oversamples the 115.2 Kbaud UART frequency at 921.6kHz, or 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 frozen when the start bit has not yet been detected. This ensures that the sampling of the line is as close to the center of the transmission's clock as possible. The Raspberry Pi sends three characters, which are flushed by the Bluetooth module's buffer at the same time, so the command appears as a 30-bit message. The sampler stops sampling when it sees a stop bit, and either begins a new message if the next bit is a start bit, or signals to the controller that a command has been received of an entire command if the line has remained high.

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 and rotation direction for each motor and a duration for which the motors should be turning. 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. Multiplexers are used to route the power to the correct pins on the H-Bridge, allowing for both forward and backwards movement. To prevent the vehicle from running indefinitely, the timer stops incrementing when the requested duration

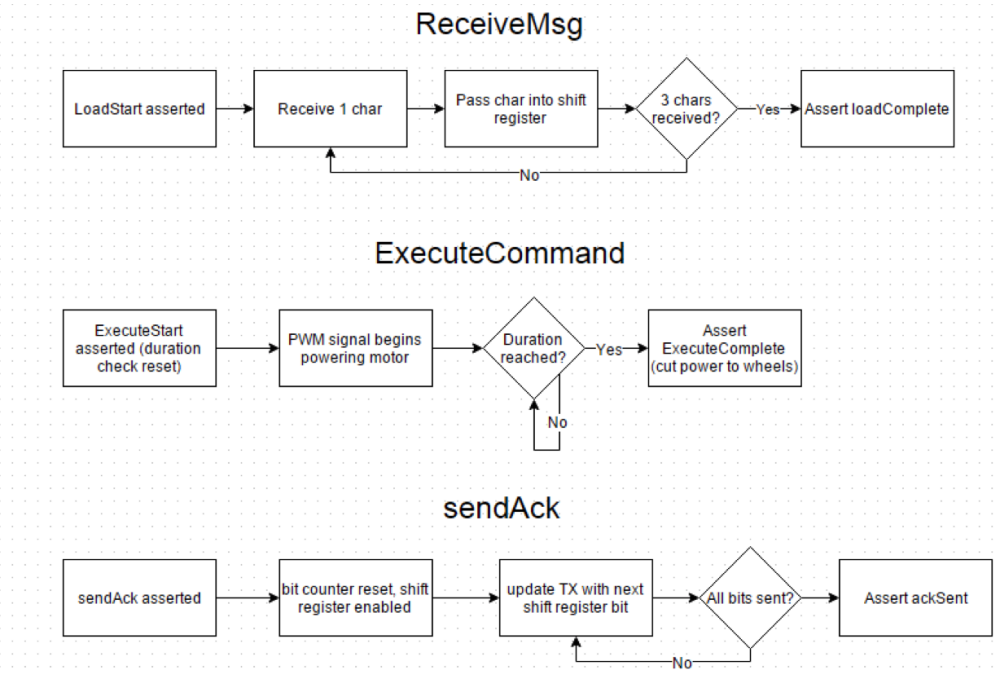


Figure 4: Schematic overview of the FPGA.

is reached, and a signal is output that is used to cut power to the motors. Each LSb of the duration character corresponds to roughly one-tenth of a second.

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 return to the receiveMSG state and start to sample for a new command.

A flowchart detailing each state of the FPGA is shown in Figure 4.

6 Results

The system has met all of the proposals, but does not always meet each one consistently. There are a few bugs of as-of-yet unidentified origin, most likely in the FPGA's SystemVerilog code, that occasionally cause the system to miss commands or fail to send an ack. These bugs are not easily reproducible.

As proposed, the Raspberry Pi hosts a website using an apache2 server. The website contains a grid with clickable cells, each of which correspond to a location on the ground. Clicking the grid triggers Javascript that updates the UI and submits code that calls C code in the cgi-bin. This C code processes the submission and converts it to the necessary command(s) to be sent to the FPGA, and then calls a Python script to send the data. The website is then updated with a "receipt" of the transmission (the HTTP response). When multiple commands need to be sent, unsent commands fill a queue. Refreshing the page will clear the buffer. In addition to what was proposed, the user is able to send arbitrary commands to the FPGA, and can also click on buttons to nudge the robot both rotationally

and translationally.

The Raspberry Pi itself is connected to a USB Bluetooth dongle that it can send data to in order to transmit the proper commands to the FPGA. The C code in the Pi has the ability to wait for an ack before sending the next command. However, this feature is currently disabled because a bug in the FPGA seems to sometimes cause the system to skip the ack stage in the controller's loop. Instead, the Pi uses only the timeout for the ack (ack timeout at one second longer than the command's duration) as a signal to send the next command.

The FPGA implements UART in order to communicate with the BlueSMiRF. Data is clocked out by the BlueSMiRF to the FPGA, where it is read into a series of shift registers, one which is bit-wide for reading the RX line and one which is byte-wide to store the received characters. Upon completion of the transmission, the FPGA sends the entire contents of the byte-wide shift register to the modules that control the PWM signal and signal routing to the H-Bridge. The commands that are successfully read from the BlueSMiRF are executed correctly.

The vehicle was built from a kit, but was modified to accept a Tamiya double gearbox. This allowed for two motors to independently drive the treads of the vehicle. The motors are wired to the H-Bridge output pins, and the breadboard on which the circuitry is located is supported by sheet metal supports.

7 Parts List

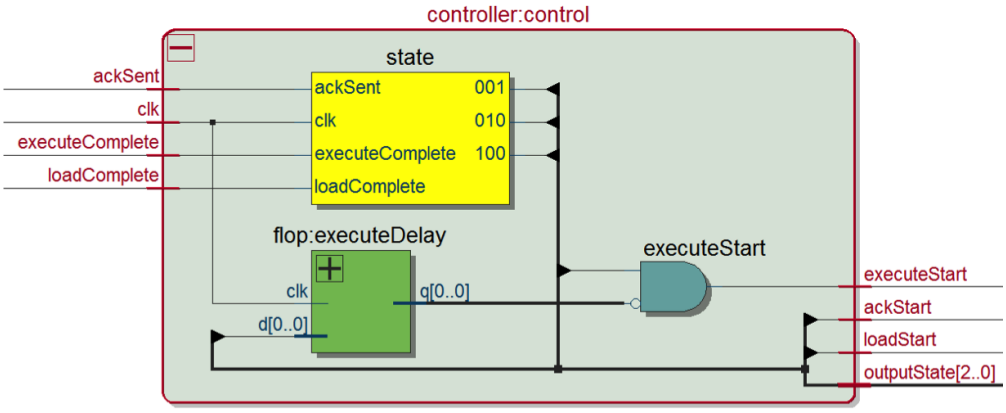
This is the bill of materials for the project.

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
Brushed DC Motor	Pololu 1117. Size: 130	Aaron Rosen	0.00
LM 7805 CT	5V Regulator	Stockroom	0.00

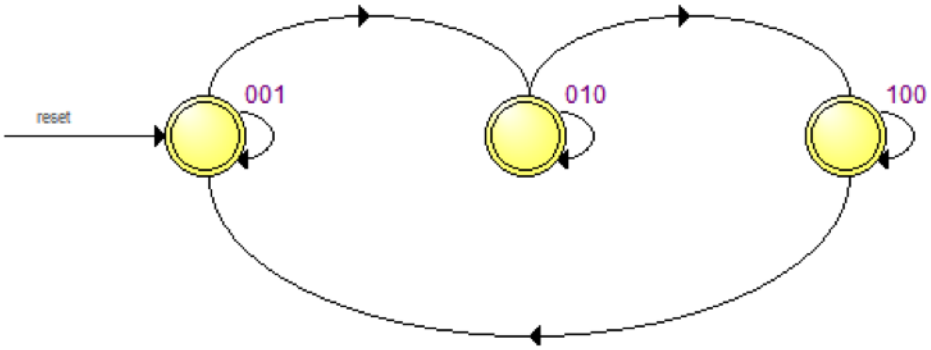
8 Appendices

8.1 FPGA Diagrams

8.1.1 Controller



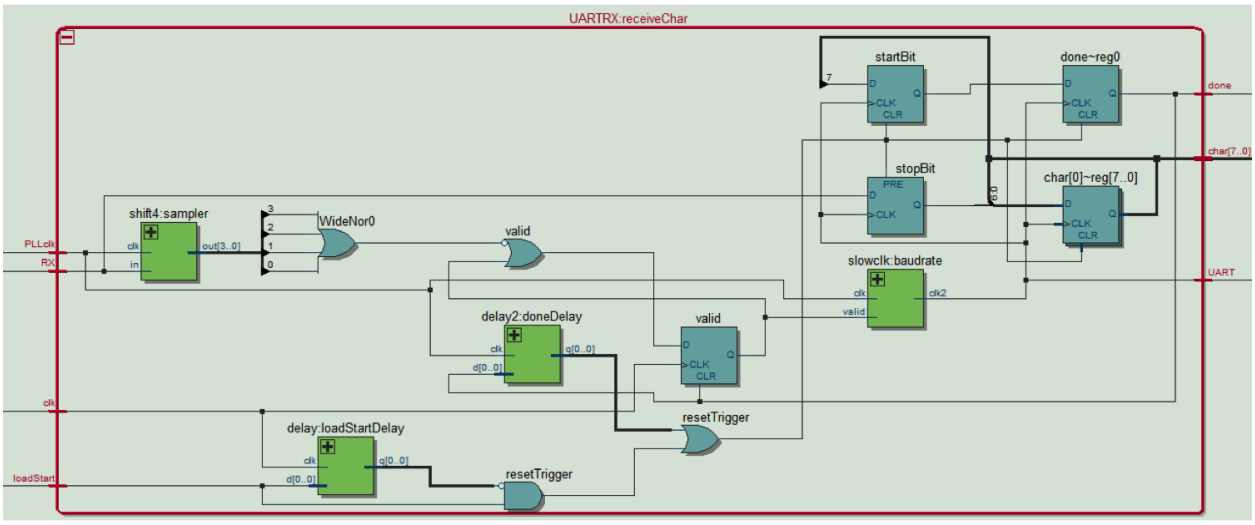
`clk` - 40MHz input clock
`loadComplete` - command loaded
`executeComplete` - successful vehicle travel
`ackSent` - ack sent successfully
`loadStart` - enables reception of commands
`executeStart` - pulse that resets duration timer
`ackStart` - enables sending of ack
`outputState` - output state of FSM (for scope only)



Controller State Machine

8.1.2 Receive A Char

This module reads one 8 bit input from the BlueSMiRF.

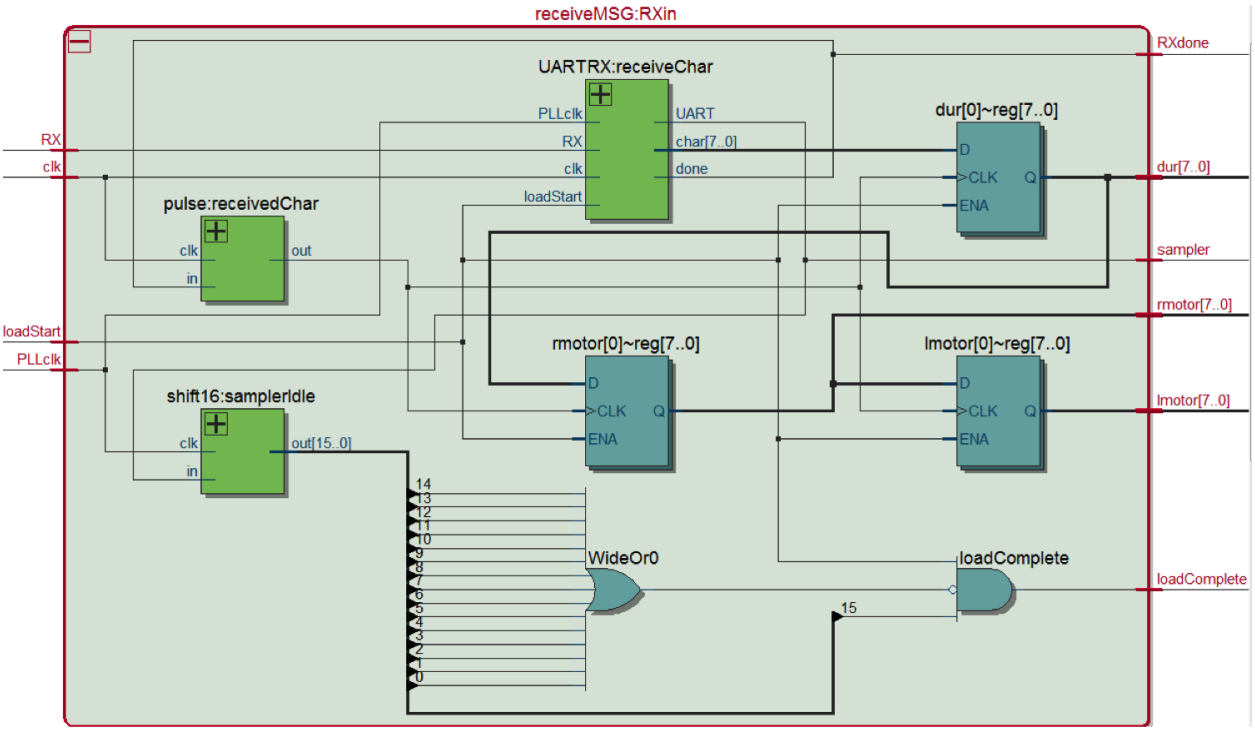


`loadStart` - Resets shift register
`clk` - 40MHz input clock
`PLLclk` - 921.6KHz UART oversampler
`RX` - BlueSMiRF TX/ FPGA RX

`done` - signals end of char received
`char` - the char received
`UART` - the 115.2KHz clk that samples the FPGA RX line (for scope only)

8.1.3 Receive A Message

This module reads three 8 bit inputs from the BlueSMiRF.

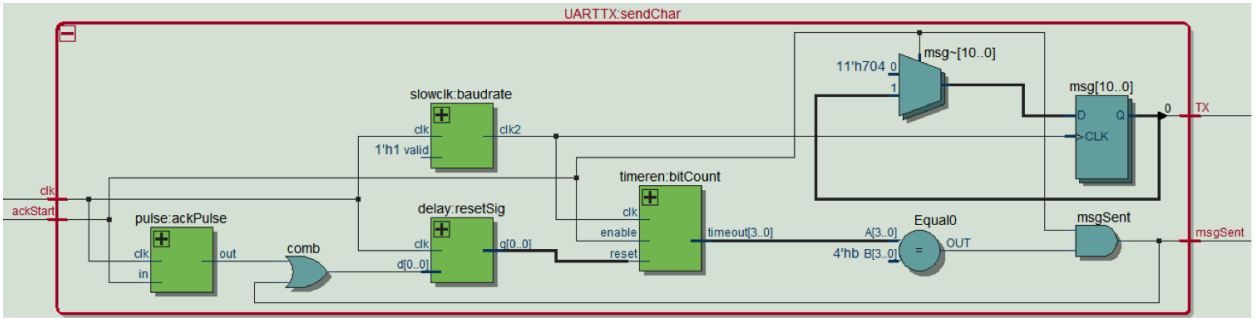


`loadStart` - Enables receiveChar and shift registers
`clk` - 40MHz input clock
`PLLclk` - 921.6KHz UART oversampler
`RX` - BlueSMiRF TX/ FPGA RX

`lmotor` - left motor PWM/direction
`rmotor` - right motor PWM/direction
`dur` - command duration
`loadComplete` - signals end of command
`sampler` - UART sampler (for scope only)
`RXdone` - signals end of char received (for scope only)

8.1.4 Send A Char

This module sends one 8 bit output to the BlueSMiRF.



`clk` - 921.6KHz UART oversampler
`ackStart` - resets counter to begin transmitting ack message

`TX` - the FPGA TX / BlueSMiRF RX
`msgSent` - indicates successful transmission of ACK

8.2 FPGA Code

VehicleControl.sv

```
//Aaron Rosen and Alex Rich
//E155 Final Project

module VehicleControl(input logic clk,
                      input logic RX,
                      output logic TX,
                      output logic[1:0] HL,HR,
                      output logic HbridgeEN,
                      output logic[2:0] state,
                      output logic[7:0] char,
                      output logic sampler,loadComplete,loadStart,
                          RXdone,ackSent);
```

```
    //top level module
    logic[7:0] lmotor,rmotor,dur;
    //logic loadStart,loadComplete;
    logic executeStart,executeComplete; //executeStart should pulse
       when starting rather than staying high
    logic ackStart;
    logic pllclk;
    logic reset,locked;
    logic[1:0] HLled,HRled;

    assign HbridgeEN = 1;
    assign HLled = HL;
    assign HRled = HR;
    assign char = lmotor;

    PLLclk2 pll(reset,clk,pllclk,locked); //sampler/UART clk

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

    receiveMSG RXin(clk,pllclk,RX,loadStart,lmotor,rmotor,dur,
       loadComplete,sampler,RXdone); //UART msg Receive
    executeCommand executor(clk,(loadComplete | executeStart),(~
       executeComplete & loadStart),lmotor,rmotor,dur,executeComplete,
       HL,HR); //powertrain control
    sendAck TXout(pllclk,ackStart,ackSent,TX); //UART msg transmit

endmodule

module controller(input logic clk,
            input logic loadComplete,
            input logic executeComplete,
            input logic ackSent,
            output logic loadStart,
            output logic executeStart,
            output logic ackStart,
            output logic[2:0] outputState);
  //datapath controller
  reg[2:0] state;
  assign outputState = 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,sampler,RXdone);
  //top level for message recive subsystem
  logic[7:0] char;
  logic[15:0]idle;
  logic shiftSig;
  UARTRX receiveChar(clk,PLLclk,RX,loadStart,char,RXdone,sampler);
  pulse receivedChar(clk,RXdone,shiftSig);
  shift16 samplerIdle(sampler,PLLclk,idle);
  assign loadComplete = loadStart & idle[15] & ~(|idle[14:0]);
  always_ff @(posedge shiftSig)
    begin
      if(loadStart) {lmotor,rmotor,dur}={rmotor,dur,char};
    end
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 #(30) 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
  //TODO: Change shift register to more directly include TX
  logic resetTrigger;
  logic ackStartPulse;
  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
      if(ackStart) {msg[9:0],msg[10]}={msg[10:0]}; //this is an 11-bit
         shift register
      else {msg[9:0],msg[10]} = {10'h107,1'h0}; //reset message loop to
          default position
    end
  assign TX = msg[0];
  timeren #(4) bitCount(clk2,resetTrigger,ackStart,count);
  assign msgSent = (count == 4'hB) & ackStart; //message send high
     after 11th bit sent
  delay #(1) resetSig(clk,(msgSent|ackStartPulse),resetTrigger);
```

```
  pulse ackPulse(clk,ackStart,ackStartPulse);

endmodule

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

  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,stopBit} = 11'h001;
      else {done,startBit,char,stopBit}={startBit,char,stopBit,RX}; //
          this is an 12-bit shift register
    end
  delay2 #(1) doneDelay(PLLclk,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;
  assign out = direction?{sig[0],sig[1]}:sig; //direction is sign in
     sign/mag number
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 #(parameter WIDTH=30)
           (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[WIDTH-1:0] durTime;
  always_ff @(posedge clk,posedge reset)
    begin
      if(reset) durTime <=0;
      else if(preset) durTime <= {dur,{WIDTH-8{1'b0}}};
      else if(done) durTime <= durTime;
      else durTime <= durTime + 1'b1;
    end
  assign done = (dur == durTime[WIDTH-1:WIDTH-8]);
endmodule

module shift3rst(input logic in,clk,reset,
           output logic[2:0] out);
  //3-register shift register with reset
  logic c,d,e;
  always_ff @(posedge clk,posedge reset)
    if(reset)
      begin
        c <=0;
        d <=0;
        e <=0;
      end
    else
      begin
        c<=in;
        d<=c;
        e<=d;
      end
  assign out = {e,d,c};
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 shift16(input logic in,clk,
        output logic[15:0] out);
  //4-register shift register, outputs all shifted bits
  always_ff @(posedge clk)
    begin
      out <= {out[14:0],in};
    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(valid) count <= count + 3'h1; //if the signal is valid,
         increment the counter normally
      else
        begin  //if the signal is not valid, hold the slow clock right
           before the transition
          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 timeren #(parameter WIDTH=8)
        (input logic clk,reset,enable,
         output logic [WIDTH-1:0] timeout);
  //a WIDTH-bit timer with enable
  always_ff @(posedge clk,posedge reset)
    begin
      if(reset) timeout <= 0;
      else if(enable) 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 flopen #(parameter WIDTH=1)
         (input logic clk,en,
          input logic [WIDTH-1:0] d,
          output logic [WIDTH-1:0] q);
  always_ff @(posedge clk)
    begin
      if(en) q <= d;
      else   q <= q;
    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 delay2 #(parameter WIDTH=1)
          (input logic clk,
           input logic [WIDTH-1:0] d,
           output logic [WIDTH-1:0] q);

  logic [WIDTH-1:0] p1,p2;
  always_ff @(posedge clk)
    begin
      p1 <= d;
      p2 <= p1;
      q <= p2;
    end
endmodule

module pulse(input logic clk,in,
          output logic out);
  //creates a pulse when the input signal goes high
  logic delayed;
  delay inDelay(clk,in,delayed);
  assign out = in & ~ delayed;
endmodule
```

8.3 Raspberry Pi Code

8.3.1 CGI Resource

inputChars.c

```
/**
 * inputChars.c
 * Alex Rich and Aaron Rosen
 * E155 Final Project
 * Fall 2015
 *
 * This program is called with an HTTP Request with three parameters: l
    , r, and t.
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <python2.7/Python.h>

// Convert an integer into a sign-magnitude 8 bit number,
// then reverse the bits.
char convertToChar(int value)
{
  char byte;
```

```
  if (value < 0) {
    value = value * -1;
    byte = (char) value;
    byte = byte | 0x80;
  } else {
    byte = (char) value;
  }
  int i;
  char a;
  char newBite = 0x00;
  for (i = 0; i < 8; i++) {
    a = byte % 2;
    newBite = newBite | (a << (7 - i));
    byte = byte >> 1;
  }
  return newBite;
}

// Print the bits in a byte.
void printBits(char byte)
{
  int i;
  for(i = 7; 0 <= i; i --)
    printf("%d", (byte >> i) & 0x01);
}

// calls Python script sendData.py with the given input parameters
void callPython(char l, char r, char t)
{
  FILE* file;
  int argc;
  char * argv[4];

  char a[1] = {l};
  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;
}

// Sends the commands to the vehicle.
bool sendData(char l, char r, char t)
{
  bool success = false;
  callPython(l, r, t);
  return success;
}

// This is executed each time that a GET request is made.
int main(void)
{
  char *data;
  int il, ir, it;
  char l, r, t;

  // Grab the data
  printf("%s%c%c\n", "Content-Type:text/html;charset=iso-8859-1",13,10)
      ;
  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 {
    // Format data correctly and send it
    l = convertToChar(il);
    r = convertToChar(ir);
    t = convertToChar(it);

    // Debug Output
    printf("<P>Raw values entered: <br>r: %d <br>l: %d <br>t: %d</p>",
        ir, il, it);
    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("Successful Ack? %s.", success ? "true" : "false");
    printf("Successful? unknown (server is not listening for ack)");
  }

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

/*
Commands to make this an exebutable with correct permissions:

cd /usr/lib/cgi-bin/
sudo gcc -o inputChars inputChars.c -lpython2.7

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 with sudo apt-get (may not need bluetooth package)
bluez, python-dev, python-bluez, bluetooth
*/
```

8.3.2 Send Data over Bluetooth

sendData.py

```
# sendData.py
#
# Alex Rich and Aaron Rosen
# E155 Final Project
# Fall 2015
#
# This script is called with three commands: l, r, and t.

 from bluetooth import *
import sys
import time

def sendChars(data, bd_addr, port):
  print "setting up connection..."
```

```
  sock = BluetoothSocket( RFCOMM )
  print "connecting to" + bd_addr + "..."
  try:
    sock.connect((bd_addr, port))
  except:
    sock.close()
    print "<h3>Address in use!</h3>"
    return

  print "sending data..."
  for c in data:
    print c, '...'
    sock.send(c)

  sock.close()

  print "data sent."

  t = int('{:08b}'.format(int(ord(data[2])))[::-1], 2)

  t = 1 + t/10.0
  print "<br>seconds slept: " + str(t) + "..."
  time.sleep(t)

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 we've been called with arguments
if len(sys.argv) > 1:
  print "<h5>Enter Python</h5>"
  data = []
  for l in sys.argv[1:]:
    try:
      data.append(l[0])
    except:
      data.append("\x00")
  printData(data)
  # Hardcoded address of BlueSMiRF
  bd_addr = "00:06:66:4F:DC:B1"
  port = 1
  sendChars(data, bd_addr, port)
  print "<h5>Exited Python Succesfully</h5>"
```

```
# Run a simple test command
else:
  data = ["a"]
  bd_addr = "00:06:66:4F:DC:B1"
  port = 1
  sendChars(data, bd_addr, port)
  receiveAck(bd_addr)
```

8.3.3 Webpage

final.html

```
<!--
 * final.html
 * Alex Rich and Aaron Rosen
 * E155 Final Project
 * Fall 2015
 *
 * This webpage allows a user to interact with our AGV.
-->

<!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> <!-- </Send Command> -->
        <div class="panel">
          <h3>Settings</h3>
          <button class="btn btn-default" onclick="resetSettings()">
            Reset Grid Size</button>
          <button class="btn btn-default" onclick="getGridSize()">
            Update Grid Size</button>
          <br>
          <br>
          <div class="row">
            <div class="col-xs-6 col-md-4"></div>
            <div class="col-xs-6 col-md-4 " style="text-align:center"
              >
              <button class="btn btn-default" onclick="nudge({l:
                forwardSpeed, r:forwardSpeed, t:5})">Forward</button
                >
            </div>
            <div class="col-xs-6 col-md-4"></div>
```

```
            </div>
            <div class="row">
              <div class="col-xs-6 col-md-4" style="text-align:right">
                <button class="btn btn-default" onclick="nudge({l:-
                    turningSpeed, r:turningSpeed, t:2})">Left</button>
              </div>
              <div class="col-xs-6 col-md-4" style="text-align:center">
                <h4>Nudge</h4>
              </div>
              <div class="col-xs-6 col-md-4" style="text-align:left">
                <button class="btn btn-default" onclick="nudge({l:
                    turningSpeed, r:-turningSpeed, t:2})">Right</button>
              </div>
            </div>
            <div class="row">
              <div class="col-xs-6 col-md-4"></div>
              <div class="col-xs-6 col-md-4" style="text-align:center">
                <button class="btn btn-default" onclick="nudge({l:-
                    forwardSpeed, r:-forwardSpeed, t:5})">Backward</
                    button>
              </div>
              <div class="col-xs-6 col-md-4"></div>
            </div>
          </div>
        </div>
        <div class="col-sm-4">
          <div class="panel">
            <h3 id="pac">Point and Click</h3>
            <div id="g">
              <!-- The grid ends up here. -->
            </div>
          </div>
        </div>
        <div class="col-sm-4">
          <div class="panel">
            <h3 id="commandListHeader">Commands</h3>
            <div id="commands"></div>
          </div>
          <div class="panel">
            <h3 id="resultsHeader">Results</h3>
            <div id="results"></div>
          </div>
        </div>
      </div>
    </div>
  </div>
</body>
```

```
<script>
// calibrated time (in tenths of seconds) it takes to move from one
   square to the next.
var forwardDur = 90;
// calibrated time (in tenths of seconds) it takes to turn 90 degrees
   in either direction.
var ninetyDur = 24;
// calibrated power level to move forward, [-127, 127]
var forwardSpeed = 127;
// calibrated power level to turn each tread when turning, [-127, 127]
var turningSpeed = 127;

// The command buffer queue
var commandList = [];
// Is the bot currently executing a command?
var isExecuting = false;

// The last clicked grid cell
var lastClicked;
var lastRow;
var lastCol;
// Creates the main grid.
var grid = clickableGrid(5,5,function(el, row, col, i){

  el.className = 'clicked';

  if (el == lastClicked) return; // The tank doesn't need to move.
  if (lastClicked) {
    lastClicked.className = '';
    pushCommands(row, col);
  }
  lastClicked = el;
  lastRow = row;
  lastCol = col;
  executeCommand();
})

// Allows user to change the grid size
function getGridSize() {
  var newSize = prompt("Please enter the grid size", "90");

  if (newSize != null) {
    forwardDur = newSize;
  }
}
```

```
function resetSettings() {
  forwardDur = 90;
}

// Given a command, this function adds it to the queue
function nudge(command) {
  commandList.push(command);
  executeCommand();
}

// Pushes commands from lastRow, lastCol to newRow, newCol
function pushCommands(newRow, newCol)
{
  var vDist = lastRow - newRow;

  // Move up and down first.
  if (vDist != 0) {
    var power = forwardSpeed;
    if (vDist < 0) power = -1 * power;
    var totalTime = Math.abs(vDist * forwardDur);
    if (totalTime > 255) {
      while (totalTime > 255) {
        commandList.push({l:power,r:0.85*power,t:255});
        totalTime = totalTime - 255;
      }
    }
    commandList.push({l:power, r:Math.floor(0.967*power), t:totalTime})
      ;
  }

  // Now we need to move left or right (if needed)
  var hDist = lastCol - newCol;
  if (hDist != 0) {
    // First turn 90 degrees

    // Set a positive tread and a negative tread:
    var pPower = turningSpeed;
    var nPower = -1 * turningSpeed;
    commandList.push({l:nPower, r:pPower, t:ninetyDur});

    var power = forwardSpeed;
    if (hDist < 0) power = -1 * power;
    var totalTime = Math.abs(hDist * forwardDur);
    if (totalTime > 255) {
      while (totalTime > 255) {
        commandList.push({l:power,r:power,t:255});
        totalTime = totalTime - 255;
```

```
      }
    }
    commandList.push({l:power, r:power, t:totalTime});

    // We're done. Turn back to face up and down (the treads are
       reverse this turn).
    commandList.push({l:pPower, r:nPower, t:ninetyDur});
  }
}

// Executes commands in the buffer. Will continue executing until
   buffer is empty.
function executeCommand() {
  // Display current commands on the webpage.
  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;
  }

  // We are executing a command.
  isExecuting = true

  // Pop the last command off the buffer
  var command = commandList.shift();
  console.log("sending command: ");
  console.log(commandToString(command));

  // create a GET request to submit this command.
  var xmlHttp = new XMLHttpRequest();
  xmlHttp.onreadystatechange = function() {
    if (xmlHttp.readyState == 4 && xmlHttp.status == 200) {
      // If the command is succesfful, we're done executing
      isExecuting = false;

      // Check if we need to execute anything else.
      executeCommand();
      document.getElementById('results').innerHTML = xmlHttp.
         responseText
    } else if (xmlHttp.readyState == 4) {
      // There was an error, put it in the header so it is noticed.
      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);
    }
  }

  // Add parameters
  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);
}

// Writes the buffer to the page.
function updateList() {
  var str = "Current commands in buffer (in order of execution): <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;
}

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

// Add the grid to the page.
document.getElementById('g').appendChild(grid);

// Generates HTML for the 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;
        lastRow = r;
        lastCol = c;
      }
      ++i;
      cell.addEventListener('click',(function(el,r,c,i){
        return function(){
          callback(el,r,c,i);
        }
      })(cell,r,c,i),false);
    }
  }
  return grid;
}

// When we start, send two test commands.
commandList.push({l:5, r:5, t:1});
commandList.push({l:10, r:10, t:1});
executeCommand();

</script>
</html>
```