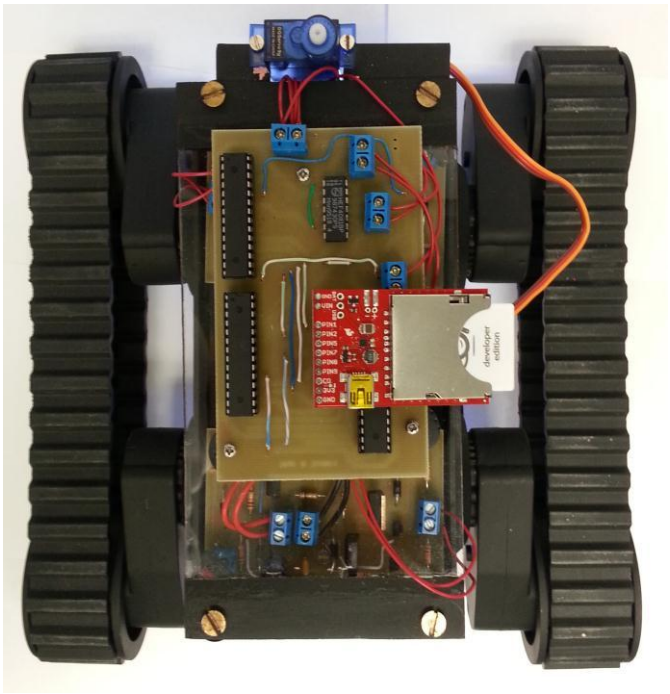


User Guide
and Technical Documentation for the
G.O.S.T© (Glove Operated Smart Tank)

May, 2013 **Edition**
Revision 1.0

Masi & Scalera
Innovations
THINK BIG... WE'LL DO THE REST

Electronic Engineering Technology
Dawson College
3040 Sherbrooke St W Montreal, QC H3Z 1A4
(514) 931-8731

www.gost.**.com**

Printed in the Canada
Created and Edited by Michael Scalera and Matthew Masi

First Edition

Table of Contents

About This Guide

This document is designed to help a wide range of knowledge customer in operating and troubleshooting the G.O.S.T ©. This document aims at helping a non-technical user in setting up the G.O.S.T©, and with simple troubleshooting should a problem arise. It also aims at explaining how the G.O.S.T© functions to more technical users, technicians, further developers. It also includes an in depth troubleshooting guide for each module in the G.O.S.T©.

This document includes the following:

- Setup Guide
- User Manual
- Technical Documentation
- System Diagrams
- Troubleshooting Guide

Overview

Smart Tank features:

- Internet connected (Electric IMP)
- Web-based programming
- Pan/Tilt mechanism
- 4WD, 4 geared motors on tank threads
- 3 dedicated microprocessors
- 6 AA-battery pack (batteries not included)
- PWM controlled H-Bridges
- Headlights and Under glow

Sensory Glove features:

- Internet connected (Electric IMP)
- Web-based programming
- 4 flex sensors for finger virtualization
- Accelerometer to monitor position of hand
- Smart transmission (Wi-Fi will only be used if new data is being sent to save battery)

The G.O.S.T © is a combination of a sensory glove and a tank in the form of a rover. The glove holds 4 flex-sensors that monitor the position of the thumb, index, middle, and ring finger as well as a 3-axis accelerometer (we only use 2 axis). Theses sensors allow us to operate the tank. The tank is a 4 wheel drive, thread driven rover. It holds a pan/tilt mechanism that allows the user to mount a cell phone (cellphone mount include) or any other camera or sensor module they wish. The glove controls the pan/tilt using the accelerometer, and uses the other flex sensors to move the rover and control the LEDs.

This product is intended to show how a sensory glove can control multiple different devices. Compared to a remote controller, the smart glove is only one hand and can control, at a minimum, the same amount of things of a remote controller.

1

Setting Up Your G.O.S.T

This section will help you setup your GOST for operation. The only thing that needs to be done is connect your Sensory Glove and Smart Tank to the internet. Once this is done, they are automatically connected to each other.

*Note: For the setup of your new G.O.S.T, you will need an Android or Apple mobile device.

To perform the setup procedure you will need to have a few things:

- Electric IMP Breakout Board from the Rover
- Electric IMP from Sensory Glove
- Electric IMP from Rover
- Mini USB
- Apple or Android Smartphone

1-1 *Connecting Sensory Glove and Smart Tank to the Internet*

The Sensory Glove and Smart Tank uses a module called the Electric Imp to connect to the internet. To get this module connected to the internet you will need to perform the "blink up". The following steps will explain how to perform the "blink up".

Step 1:

You must download the smartphone app called: Electric Imp, on your Android or Apple device.

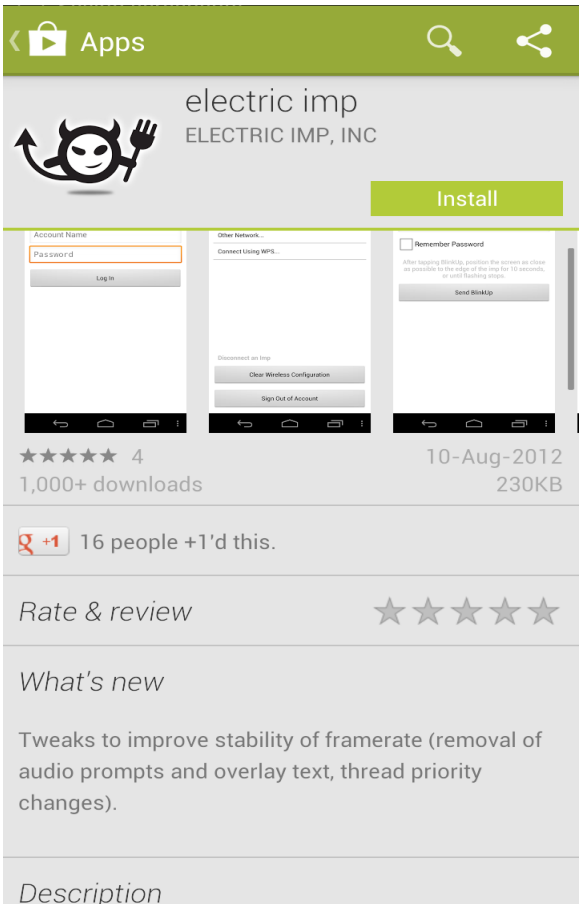
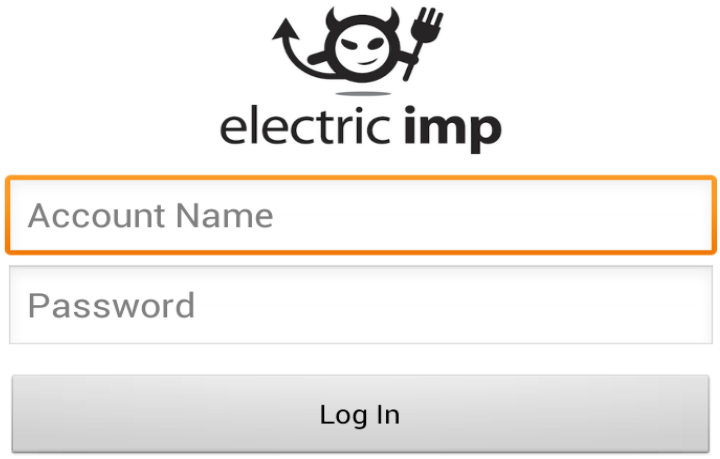


Figure 1: Electric IMP smartphone app

Step 2:

Once the application is downloaded and installed, you can open the app. You will be prompted with a login page.



To login in you can use the universal account:

Username: Gost_user@outlook.com

Password: Publicvoid

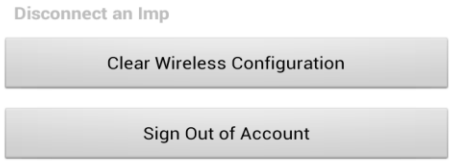
*Note: If this account is not working, please notify us by emailing or calling us at the information located at our contact page.

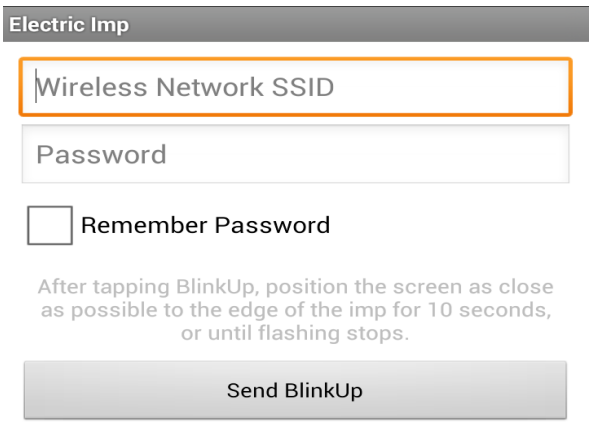
Step 3:

Once logged, you will be presented with the following page:

Select the outlined area.

Here you can put your WIFI information.





Step 4:

To perform the blink up you need to have three things ready:

- Electric Imp breakout board (Rover)
- Electric Imp (Rover)
- Mini USB cable

Connect the Mini USB cable to the Electric Imp breakout board. Put the Electric Imp in the slot with the Electric Imp logo facing up and the cut corner facing the breakout board. Connect the USB cable to any powered USB plug.

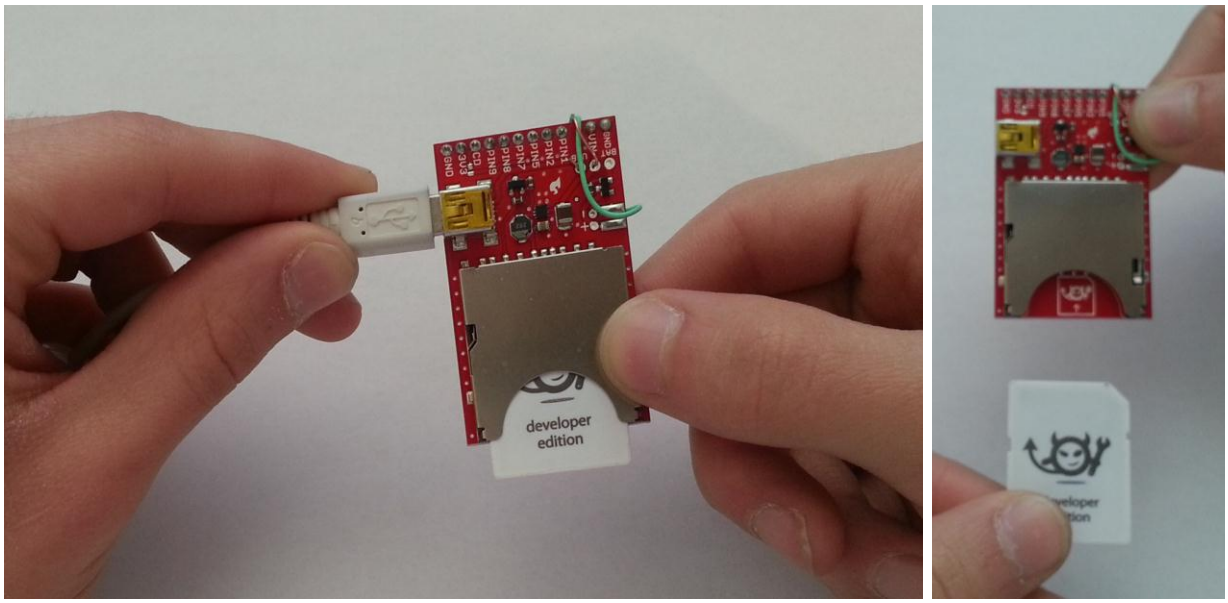
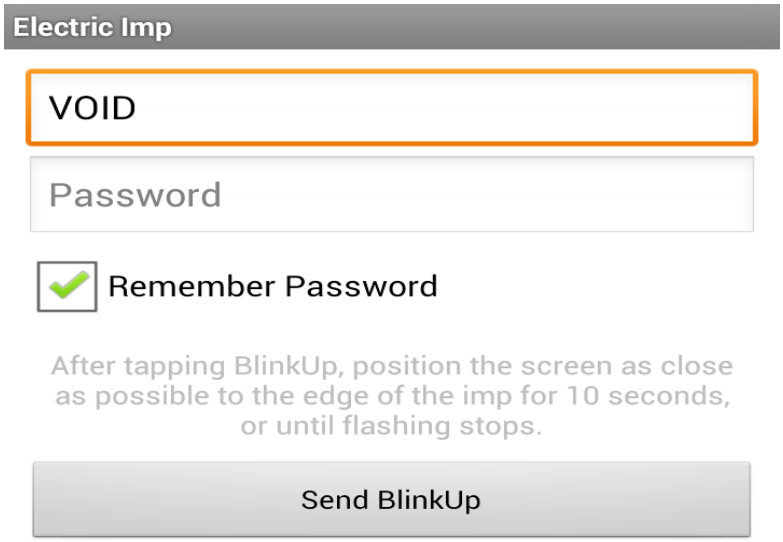


Figure 2: Connecting mini USB cable to breakout board

Step 5:

*Note: Once all that is set up, you should see the electric imp led flashing orange.

Insert your WIFI settings in the app.



Click “Send BlinkUp” on the screen and within 2 seconds place the breakout board with electric imp sticking out on the screen.

Wait for the flashing to end (approx. 10 seconds)

Step 6:

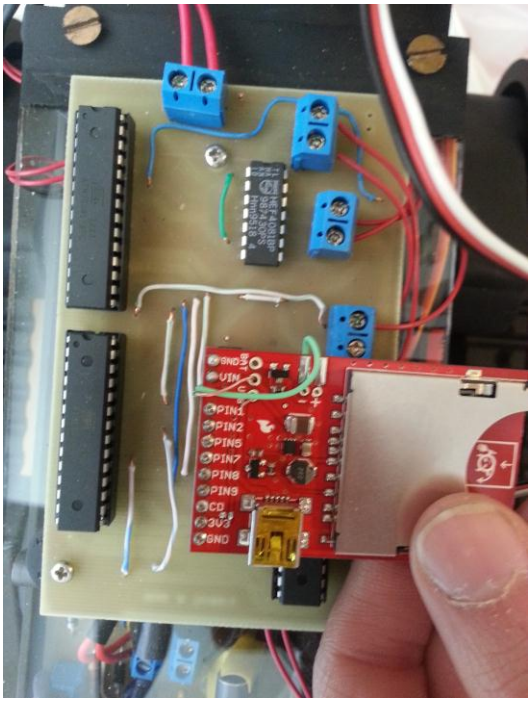
Once completed you should see the LED on the electric imp flashing green (after a little time of flashing red/orange). This means the electric imp is connected to the internet!

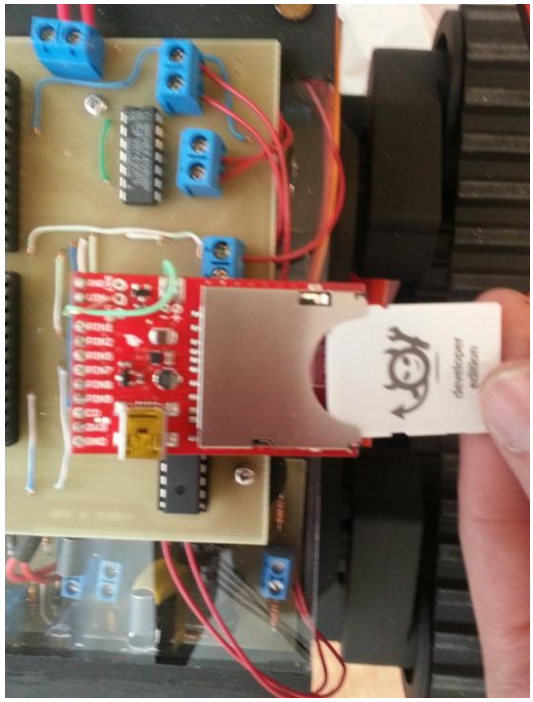
Step 7:

Repeat steps 1 – 6 on the second electric imp.

Step 8:

Once both of the electric imps are connected to the internet, we can now place one in the Sensory Glove. The breakout board used to commission these IMPs will go back on the rover and the second IMP will go inside the breakout board.





It doesn't matter which one goes where because the breakout board on each device has a unique ID code the will help distinguish the two from the Electric Imp cloud server.

1-2 *Placing the batteries*

Step 1:

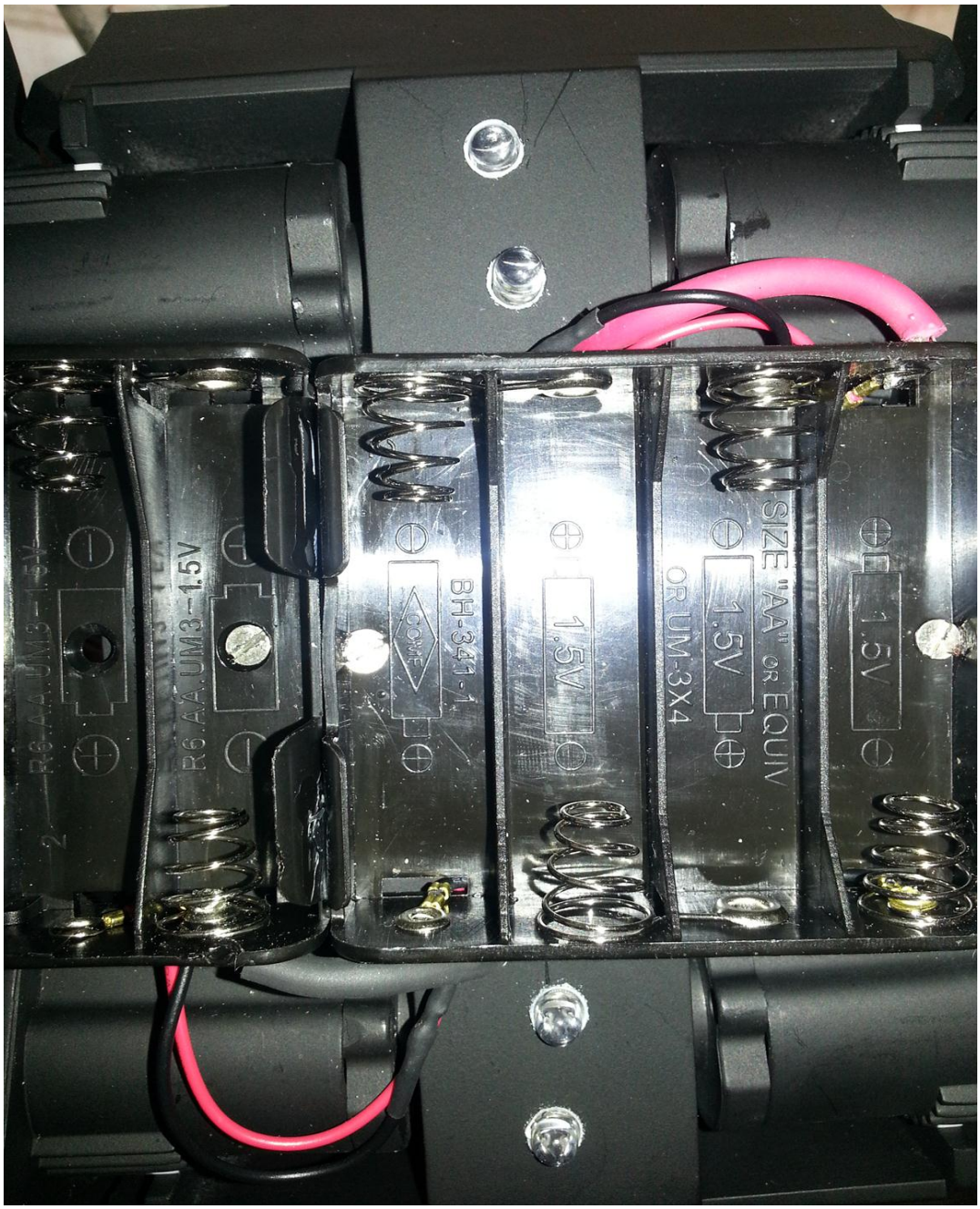
*Note: The sensory glove can take 3X any type of AA-batteries.

Place batteries in the Sensory Glove by following the diagrams on the battery holder:

Step 3:

*Note: The Smart Tank requires 6X NiMH type batteries.

Place batteries in the Smart Tank by following the diagrams on the battery holder:



2

User Guide

2-1 *The ControlCenter*

Overview -

The ControlCenter is a restricted area located on our webpage.

URL: http://www.gost.web44.net/

In the ControlCenter users and admins will have access to the Smart Tank CAMVIEW, and admins will have access to the Electric IMP planner. *Note: The electric imp planner should only be view and edited by technicians and developers. Any change to this area can effectually disable the G.O.S.T.

Step 1:

Start by visiting our webpage. http://www.gost.web44.net/

Figure 3: G.O.S.T webpage

Then you can click on ControlCenter. You will be prompted for your login credentials for the restricted area.

Figure 4: Logging into ControlCenter

Username: Admin
Password: Publicvoid

Step 2:

You will then select Smart Tank CAMVIEW.

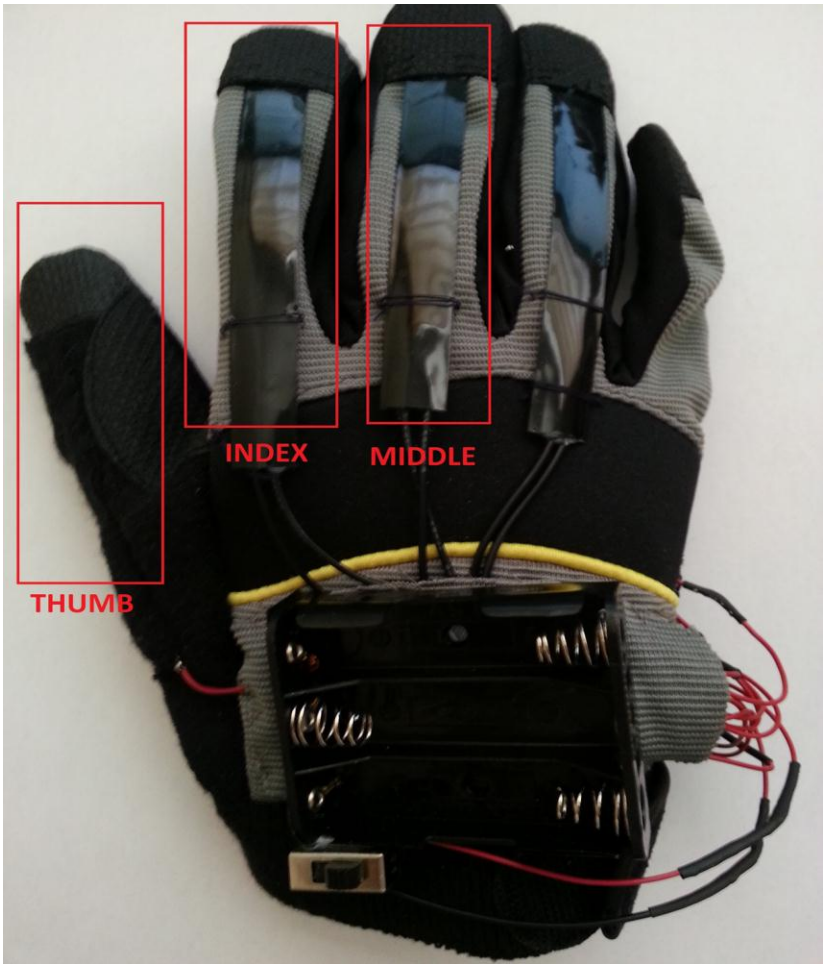
You will be brought to a page where you can see the feed of the IP cam on the rover, as well as the camera controls for things like the LED on the camera and to take a high resolution picture.

2-2 *Controlling the G.O.S.T system*

2-2-1 *Basic Smart Tank Movement*

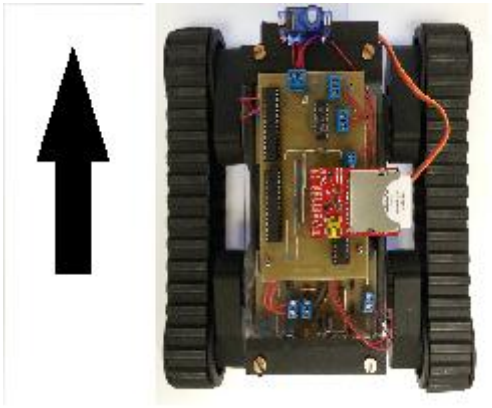
Here is a list of what every finger does for basic movement:

Thumb	Control forward / reverse
Index	Direct Control of speed of left thread of Smart Tank
Middle	Direct Control of speed of right thread of Smart Tank



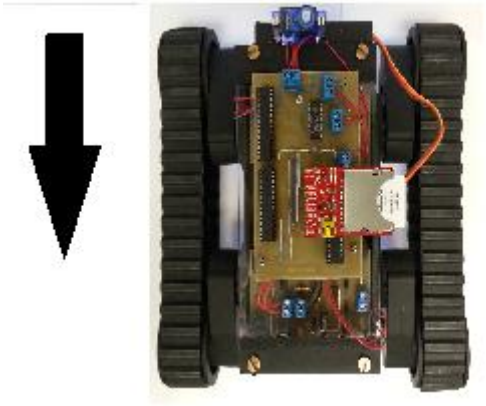
Full forward:

Both Index and Middle fingers bent:



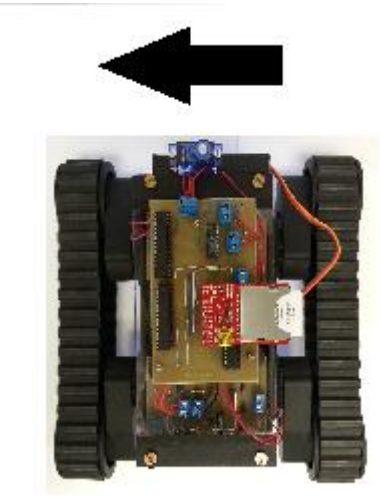
Full Reverse

Thumb down, and both index and middle fingers bent:



Turn Left

Middle finger bent:



Turn Right

Index finger bent:

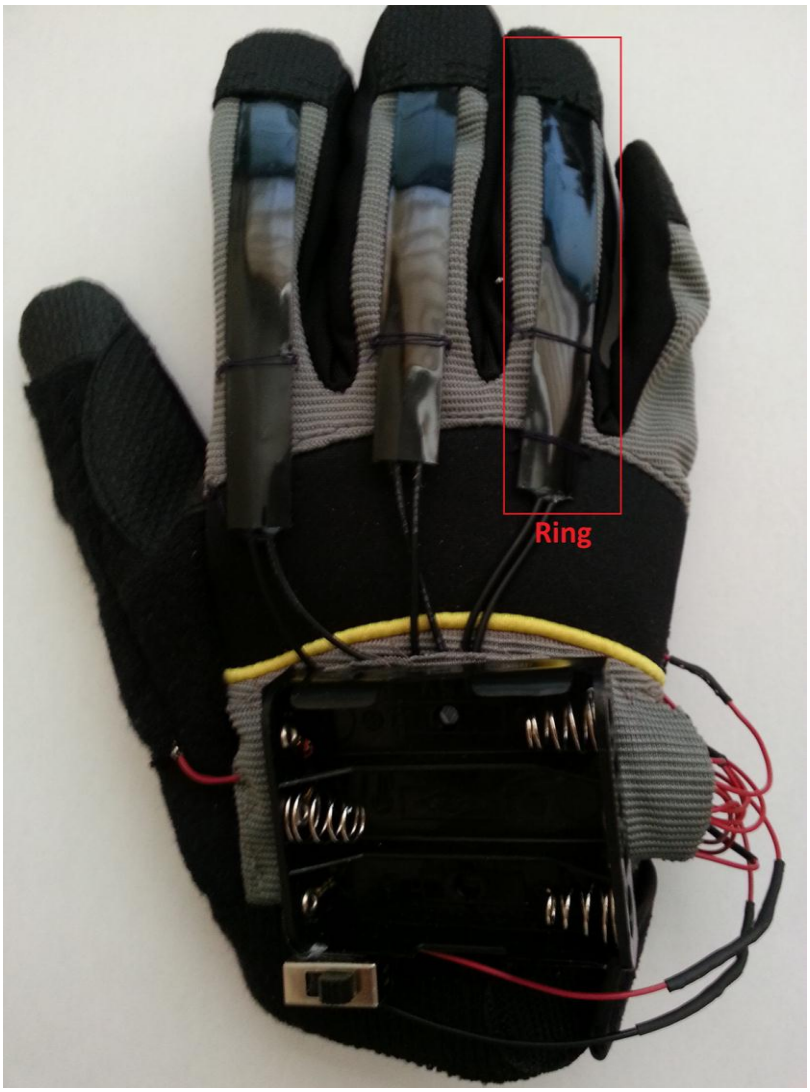
2-2-2 *Advanced Smart Tank Movement*

The Smart Tank will also allow you to a 360 degree spin. To use this feature first you must place your hand in a fist:

By tilting your hand left or right you will be able to spin the tank in that respective direction.

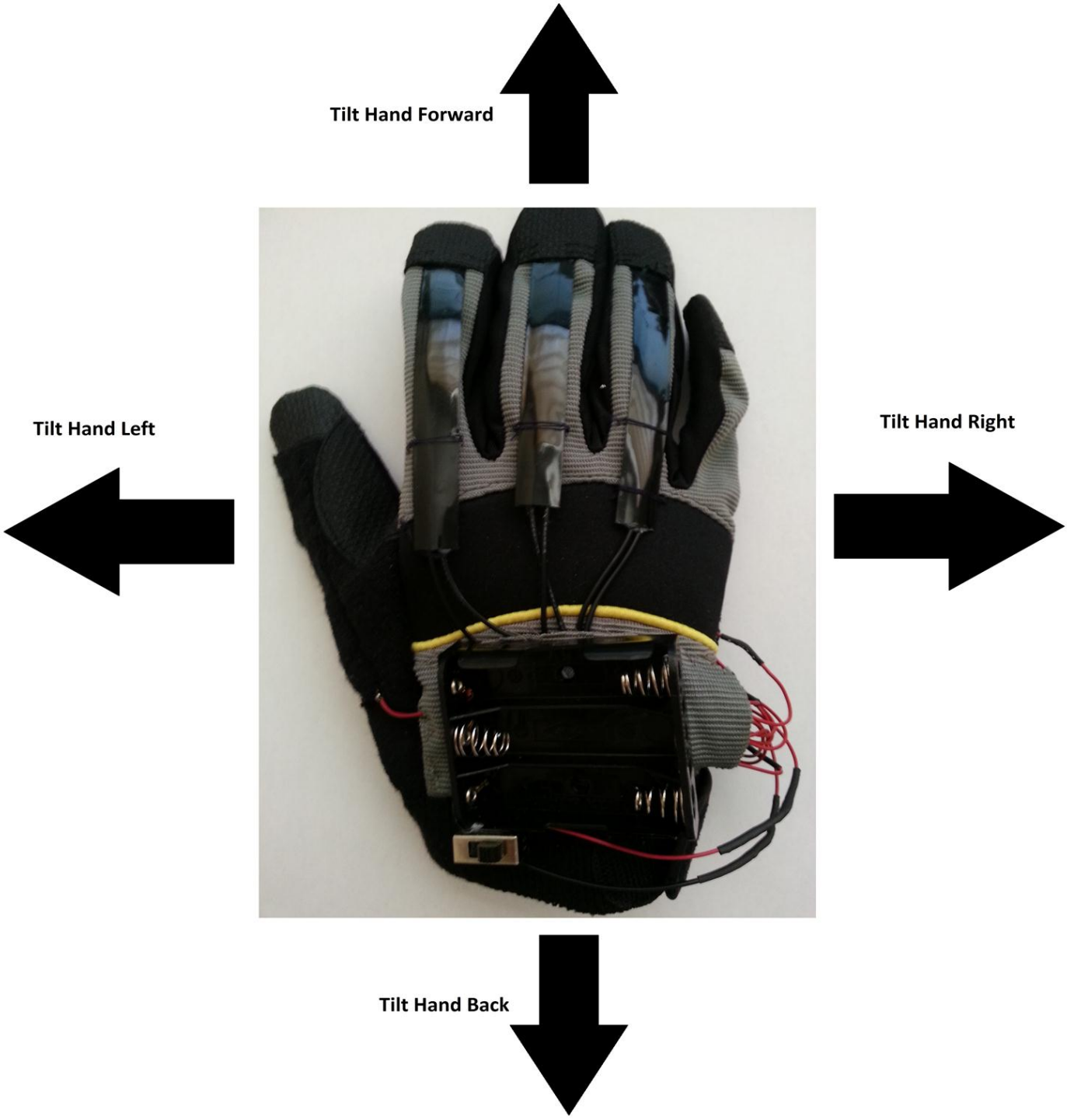
2-2-3 *Camera Control*

To enable camera control you will first need to bend your ring finger.



This will activate the accelerometer which controls the pan/tilt.

Once the accelerometer is activated you will be able to control the pan/tilt with the following movements:



Here is how the camera pan/tilt module will react to the movements above.

Hand centered	Pan/Tilt centered
Tilt Hand Right	Pan Right
Tilt Hand Left	Pan Left
Tilt Hand Forward	Tilt Down
Tilt Hand Back	Tilt up

2-2-4 *Underglow Toggle*

To toggle the Smart Tank's blue led underglow you need to bend the thumb and ring finger at the same time.

2-3 *Simple Troubleshooting*

Problem	**Possible Solution**
Glove Not Turning on	- Batteries are low, change them
G.O.S.T not responding to movements	- Take IMPs out, and put them back in to reset
G.O.S.T not responding to movements but LEDs on tank are on	- Batteries on Smart Tank low
G.O.S.T moving by itself	- SPI transmission failed because of vibration, reset the IMPs.
Some movement not working	- Either wiring, or component bad on glove
Cannot see camera on webpage, but Smart Tank is moving fine	- IP cam not online, or not connected to network
Batteries are OK, nothing is working	- Network failure

3

Techinical Manual

3.1 *Webpage*

3.1.1 *Web hosting service*

To web hosting site that we decided to use is called 000webhost.com
This hosting company offers free webhosting, the only limitation is that you have a subdomain as your URL.

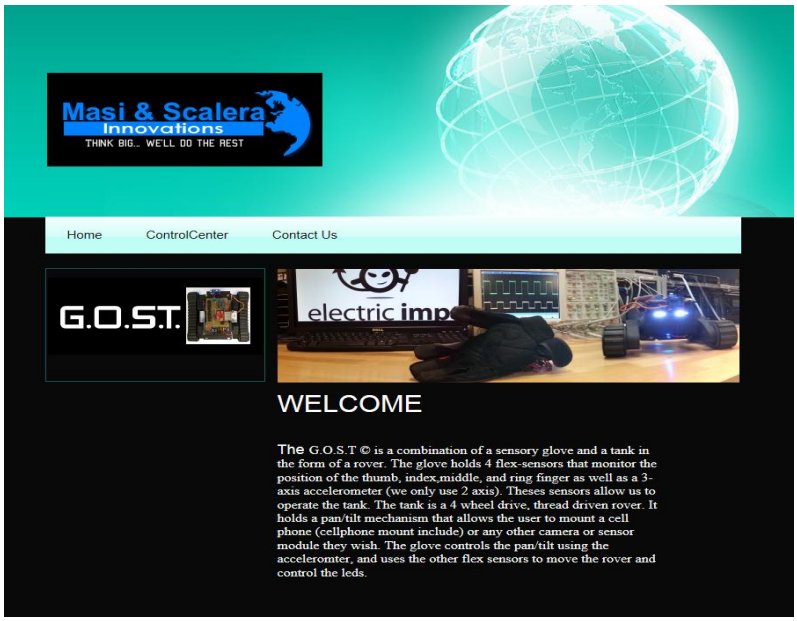
This hosting company also offers a control panel to edit your webpage:

We used 000webhost.com's GUI to build our webpage.

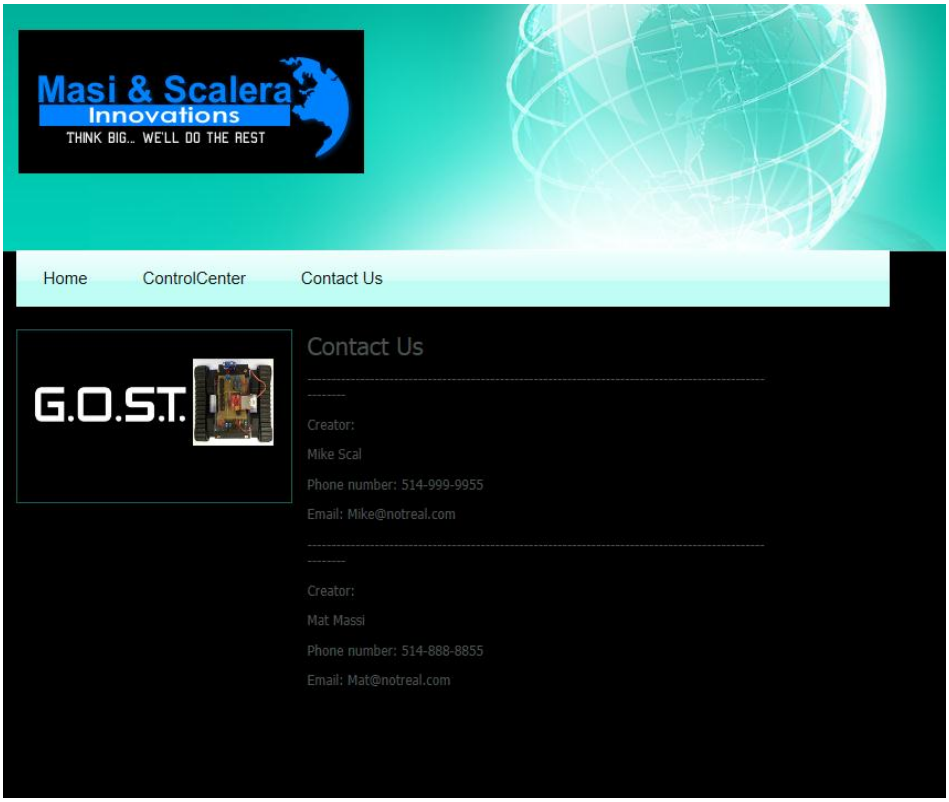
Here you can drag and drop elements of the webpage. You can also add and delete other pages.

3.1.2 *Webpage Navigation*

Home page:



From here you can navigate to two other areas of the webpage. First is the ControlCenter, which is a restricted area for users of the GOST. The ControlCenter will be discussed in section 3.1.3. The second tab in the navigation bar is a contact page:

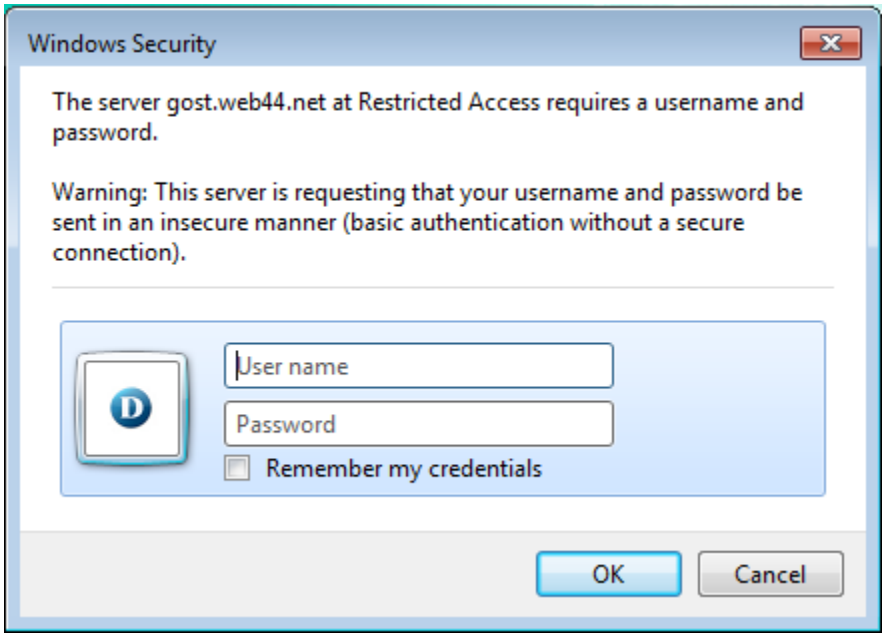


3.1.3 *ControlCenter*

The ControlCenter is an area restricted to only users of the G.O.S.T. To gain access to this area you need to login in with these credentials:

Username: Admin
Password: Publicvoid

The login will look like this:

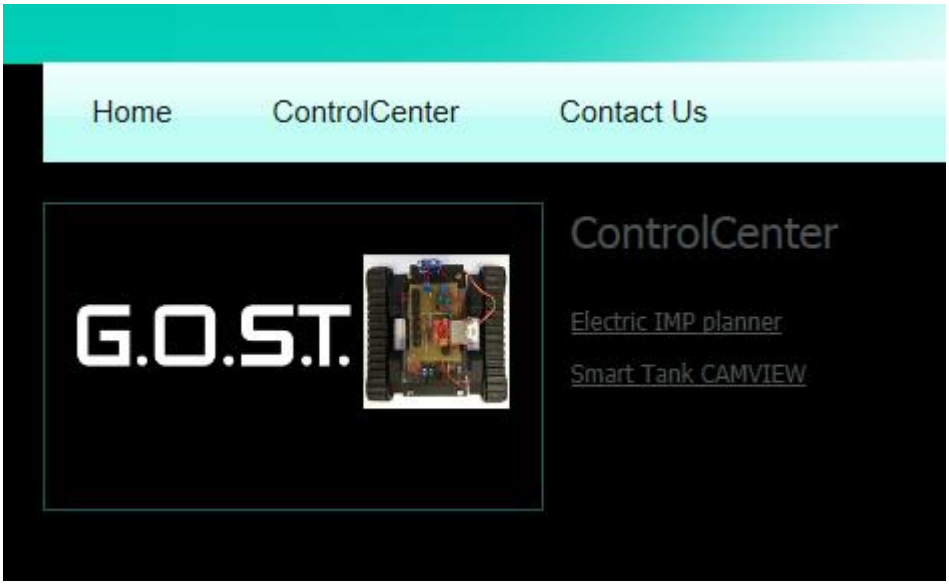


*Note: This login script is using something called htaccess. This tool allows a directory level permission control. Meaning this login will allow access to the directory itself. We chose to use this because this will secure the directory containing the IP address of our IP cam, and an attacker would not be able to gain access to this directory by putting the extension in the URL.

Once logged in you will be presented with two links:

Link 1: To the electric imp planner. (Communication cloud for Electric IMP)
Link 2: To the Smart Tank CAMVIEW (IP camera mounted on rover)

Looks like this:



The Electric IMP planner will be explained further on in the manual. But the SmartTank CAMVIEW is a link that will bring you to a page that will display the IP camera feed and IP camera controls for the IP camera located on the Smart Tank. The CAMVIEW will be explained further in the next section.

3.2 *CAMVIEW*

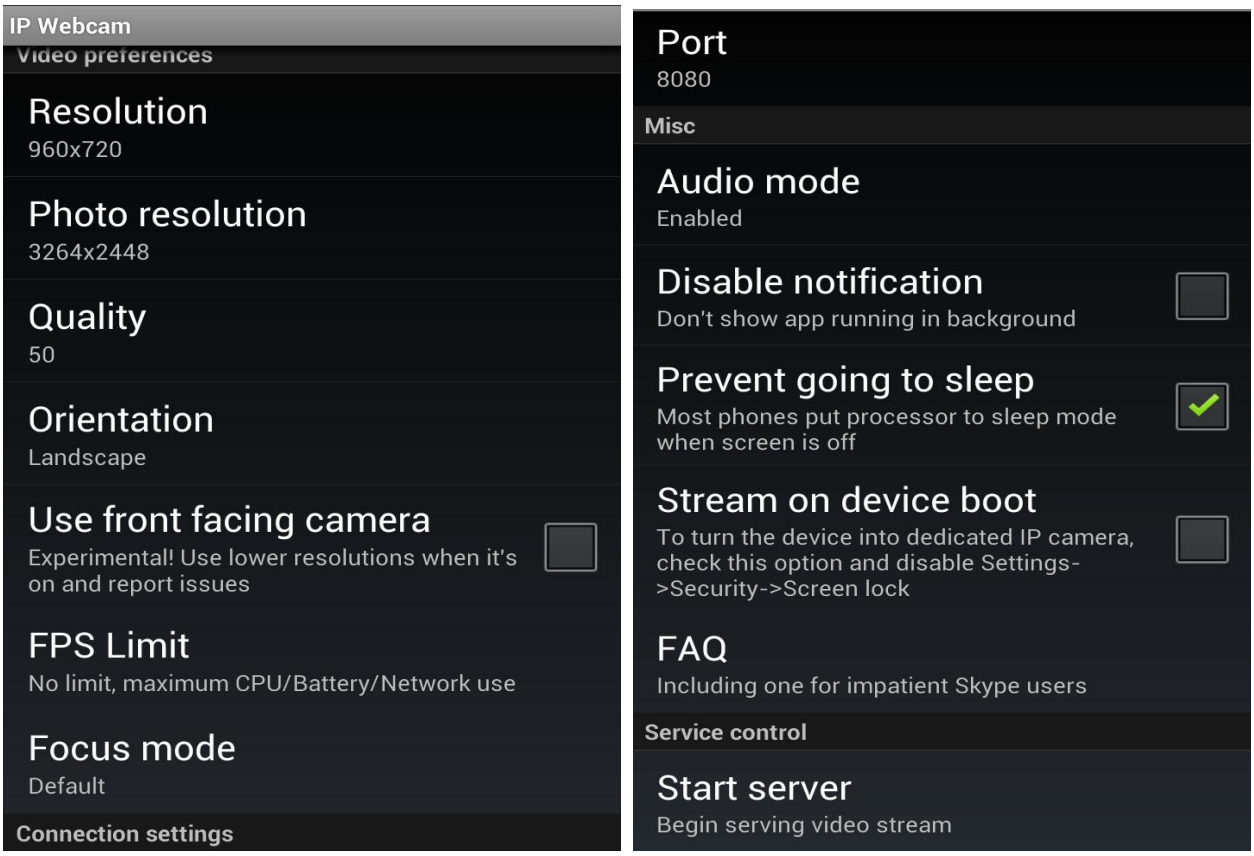
Overview -

The G.O.S.T natively does not come with an IP camera. The G.O.S.T offers a pan/tilt mechanism to allow for the user to place any sensor or camera they wish. For demonstrational purposes we included an IP on the rover to show how this would work. The IP camera we used is a dedicated Smartphone app running on an Android device. What this allows is the camera to have its own server as well as its own controls and battery.

3.2.1 *IP Webcam*

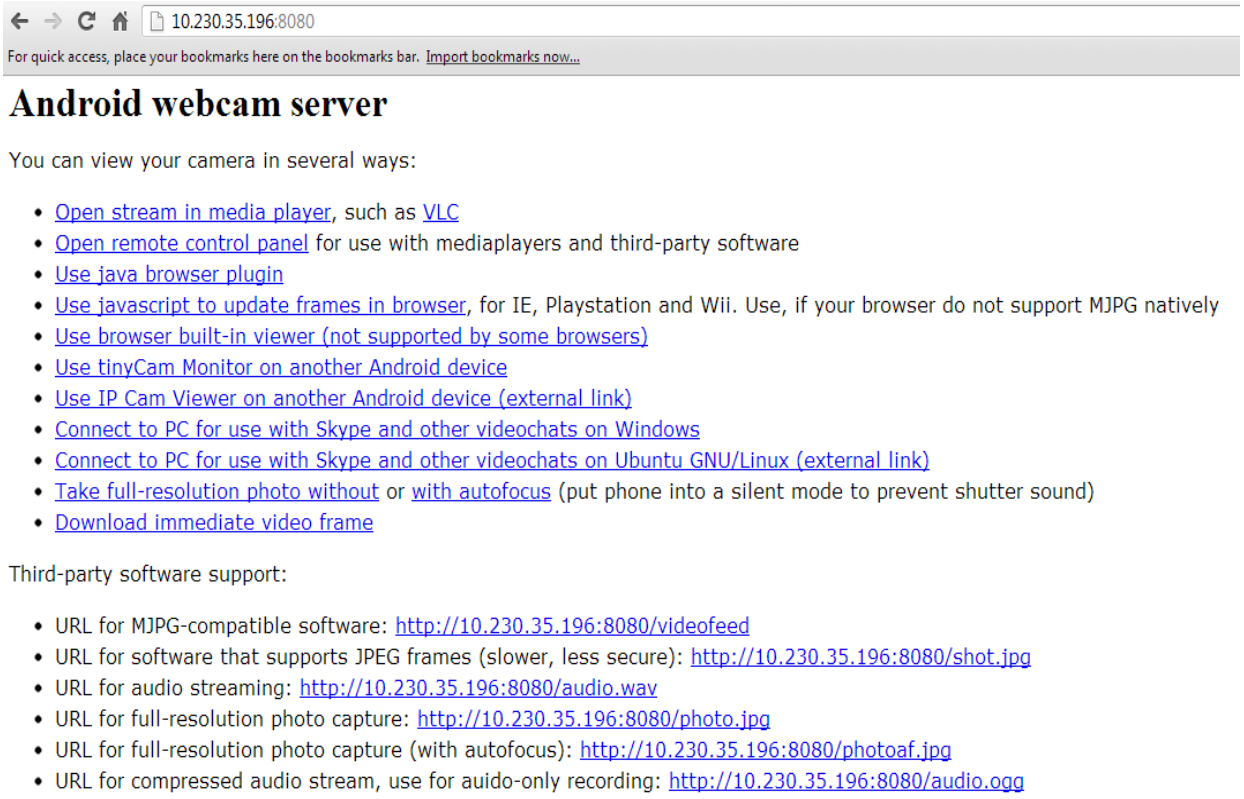
The Smartphone app we used to accomplish this was IP Webcam. It is easy to use, with a simple GUI.

We changed a few things like the resolution (so it fights native in our webpage) and the quality (to allow for smoother video and less delay). Nothing else was changed, so we can go ahead and start the server.



3.2.2 *Accessing the IP camera*

The app makes it easy to access the IP camera. Also it offers many different types of connection to the server. To access the IP camera menu, you need to have a computer connected to the same network as the Smart Phone, and point the browser to the IP of the phone, with port 8080(default). Here is what it looks like:



3.2.3 *Displaying the IP camera on the webpage*

When displaying the IP camera feed on the webpage, we wanted to accomplish to things. We wanted to offer the controls of the IP camera on the side along with the actual feed, and we wanted to hide the IP address from normal users who should not be accessing the IP cam server directly.

To accomplish this we used a new HTML5 tag. The tag is called iframe. Here is a snippet of the source code:

```
<iframe src="http://192.168.0.106:8080/videofeed" width="960" height="720"></iframe>
```


The tag uses the conventional open and closing format. What the actual tag does is display a webpage in a frame within another webpage. As you can see in the snippet, there is a "src=" statement, and then after a URL for where the frame can fetch the other webpage.

You have optional tags like width and height that was used in this case to get the full video within the frame.

Here is the completed code for this page:

```
<html>

<iframe src="http://192.168.0.106:8080/videofeed" width="960" height="720"></iframe>

<iframe src="http://192.168.0.106:8080/remote.html" width="250" height="720"></iframe>

<!html>
```

There are two frames on this page, the first frame is the video feed the second frame are the camera control.

This is how it looks. Very clean, and the URL at the top does not change to the IP address of the IP camera because the page is being served by the same server as the webpage.

3.3 *Electric IMP*

3.3.1 *Introduction*

The Electric IMP is a small device that allows a digital or analog circuit to connect to the Internet in a very easy way. The Electric IMP is a small device featuring a microprocessor and a Wi-Fi card built in. The electric IMP is programmed using and browser based compiler, and the code is written in a language called Squirrel (looks a lot like C). The Electric IMP also offers debugging messages and errors in the code to be read back to the browser where the developer can see what is going on with his device.

Features:

- Cortex-M3 Microprocessor
- 802.11b/g/n Wi-Fi
- WEP, WPA, WPA2
- Browser based compiler and programming
- 6 Buffered I/O

3.3.2 *Electric IMP API*

The Electric IMP offers a built in library of APIs. Here are a few that are needed to get the IMP functional and that were used in the making of the G.O.S.T.

```
imp.configure("Name", [InputPorts], [Outputports]);
```

This will configure the IMP with the cloud server. This also includes Input Ports and Output Ports which are used to send data back and forth from the cloud server.

```
hardware.pin7.configure(DIGITAL_OUT);
```

This will configure a pin (I.e. Pin 1, Pin 2, Pine 5, Pin 7, Pin 8, or Pin9). You can configure the pins to be anything, in this case it's a regular digital output (on or off), but can be set as an input, ADC input, SPI output, SPI input, etc...

```
imp.wakeup(5.0, awake);
```

This will configure the IMP to sleep for a certain amount of time, in this case 5 seconds, and the call a function once it is awake. It is very necessary that the code have at least one of this, with a minimum time of 20ms. This is because the Electric IMP does it communication with the cloud server while it is sleeping, and there is a keep-alive that is done on the IMP. If the IMP does not sleep, it will not respond to the cloud server keep-alive pings and it will disconnect from the server. This is also important because when the IMP is sleeping is when your data is transferred to the server.

```
hardware.pin1.read()
hardware.pin1.write()
```

Will respectively read or write from a pin specified.

```
local my_output = OutputPort("Val");
```

This will create an output port which is in charge of pushing data to the cloud server. It will also tie that output port to a locally significant variable.

```
class Input extends InputPort
{

	type = "number"
	name = "Input"

	function set(value)
	{
		DO something;
	}
}
```

This class statement will handle the input variables that it receives from the cloud server. This class will be called up whenever it receives data from the server.

3.4 *Sensory Glove*

3.4.1 *Overview*

The sensory glove uses the Electric IMP as the central point. Below is a diagram of how data is read within the Sensory Glove. The Sensory glove consists of:

- 4 Flex sensors, 1 for each finger except for the pinky
- 1 3-axis Accelerometer
- Electric IMP to read sensors and send data to internet

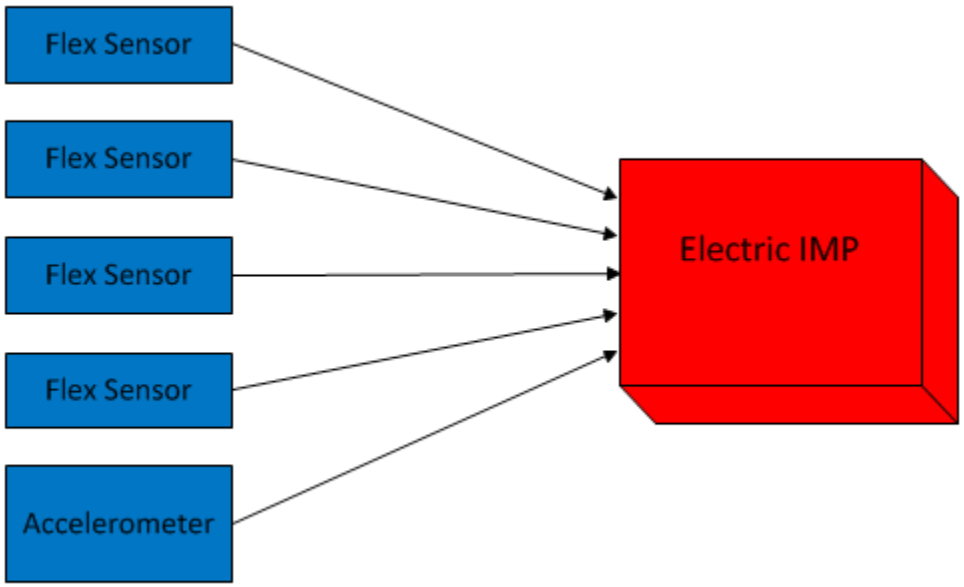
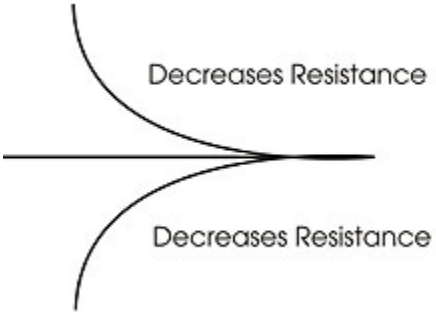


Figure 5: Sensory glove block diagram

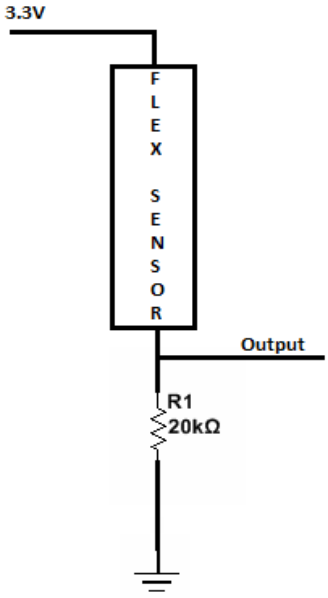
3.4.2 *Flex Sensors*

To accurately measure the bending radius of the finger, flex sensors are used. Flex sensors are in simple form, a potentiometer. The resistance changes as you bend the sensor.

The specific sensor that was used in the Sensory Glove is the FLX-01-L by Images Scientific. This specific sensor ranges in resistance from 20k Ohm to 1k Ohm. The resistance decreases as the sensor is bent.



These flex sensors are then used in a voltage divider circuit and feed into the Electric IMP for it to be read by an Analog-to-Digital Converter.



The voltage divider will then output a voltage that will start at ~1.65V and increase from there as the finger is bent. The Electric IMP then reads this value through its ADC input.

3.4.3 *Accelerometer*

The accelerometer used in the Sensory Glove is the 1.5 / 6 g Triple Axis Accelerometer Breakout Board (MMA7361L). The accelerometer takes an input voltage anywhere from 2.2 – 16 V, perfect for our 3.3V coming from the Electric IMP breakout board power supple. This accelerometer also has a power supply that offers a regulated 3.3V.

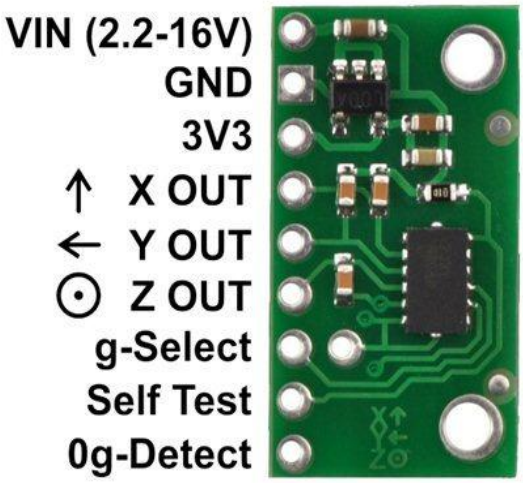


Figure 6: Accelerometer Breakout Board

The sensor will output a voltage based on the tilt. Although it is a 3-axis accelerometer, we only us 2-axis (X and Y). When both axis are centered, the output voltage will be ~1.65V (half of 3.3v). Each axis has sensitivity (at default) of 800mv/g. By tilting the accelerometer in a certain direction the corresponding axis will increase or decrease in voltage at its output. The X and Y OUT pins are directly tied to the Electric IMP ADC input pin.

3.4.4 *Electric IMP*

Sensory glove wiring diagram:

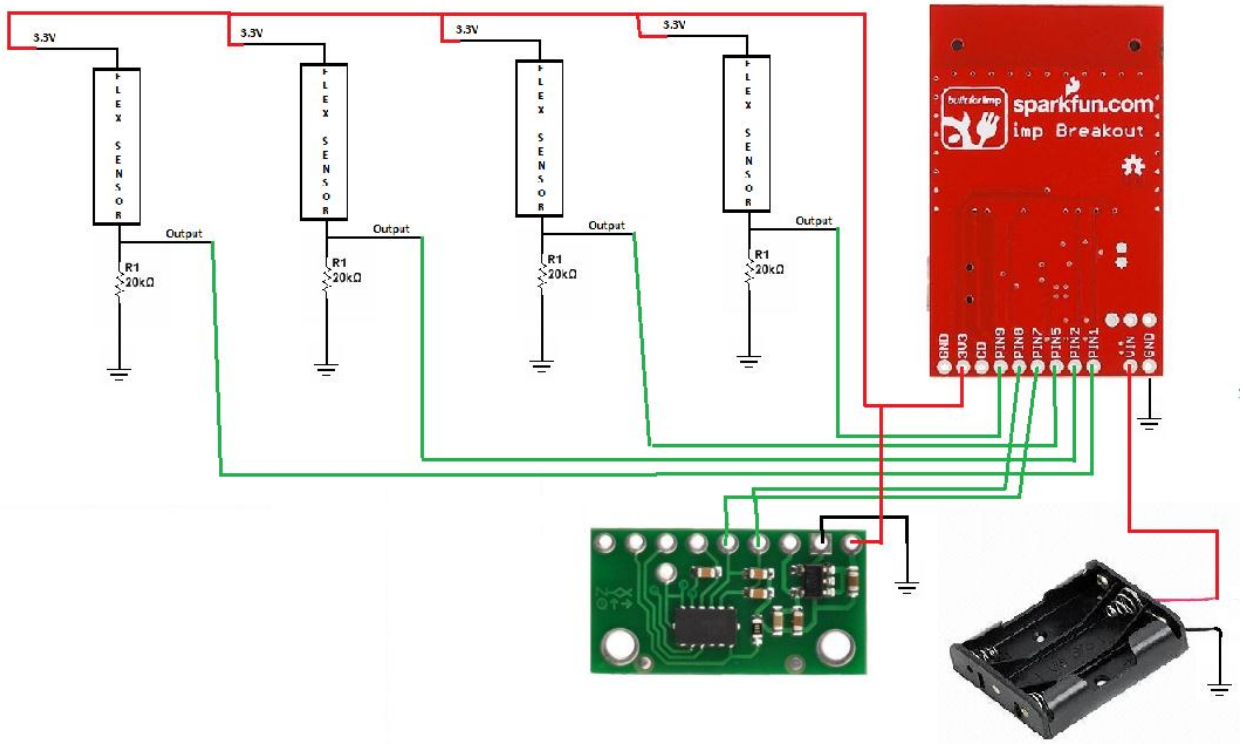


Figure 7: Sensory glove wiring diagram

Electric IMP pin connection:

Pin 1: Ring Finger (flex sensor voltage divider)
Pin 2: Middle Finger (flex sensor voltage divider)
Pin 5: Index Finger (flex sensor voltage divider)
Pin 7: X – axis (accelerometer)
Pin 8: Y – axis (accelerometer)
Pin 9: Thumb Finger (flex sensor voltage divider)

```
////////////////////////////////////////////////////////////////////////
// G.O.S.T System                                                     //
// Created by: Michael Scalera and Matthew Masi                       //
// Masi & Scalera Innovations                                         //
// Sensory Glove Code                                                 //
// Revision 1                                                         //
//                                                                    //
// This code is open source, and may be edited.                       //
// Any change to this code may hinder the performance of your         //
// G.O.S.T System. We are not responsible for any damage you cause.//
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////

// Declaring Output Variables

local output_index = OutputPort("Index");
local output_middle = OutputPort("Middle");
local output_x_axis = OutputPort("X-axis");
local output_y_axis = OutputPort("Y-axis");

// Declaring Local Variables

local thumb;
local index;
local middle;
local ring;
local x;
local y;

local battery;
local wifi;

// Declaring Temporary Storage Variables

local buffer;
local buffer_index;
local buffer_middle;
local buffer_x;
local buffer_y;
```

Above we just declare all our variables. The important one to notice is the Output Port variables. Those statements are tying local significant variables to the output port variables which are sent to the cloud server. This is the first step to transferring data between the IMPs

```
//*************>> MAIN CODE <<*************//

server.log("SENSORY GLOVE NOW ONLINE");
server.show("SENSORY GLOVE NOW ONLINE");

function read()
{
    battery = hardware.voltage();       // Measure Regulated Voltage
    wifi = imp.rssi();                  // Measure Signal strength in DBM

    //*************Thumb Finger*************//

    thumb = hardware.pin9.read();       // Read Pin 9

    thumb = ((thumb/1000)-38);          // Adjust value for tolerance
    thumb = thumb.tointeger();          // Convert from float to integer

    if(thumb <= -1)                     // Filter negative numbers
    {                                   // and keep it in range
        thumb = 0;
    }
    else if(thumb >= 6)
    {
        thumb = 5;
    }

    //*************Index Finger*************//

    buffer_index = index;               // Save previous value
    index = hardware.pin5.read();       // Read Pin 5

    index = (((index/1000)-35)/2.3);    // Adjust value for tolerance
    index = index.tointeger();          // Convert from float to integer

    if(index <= -1)                     // Filter negative numbers
    {                                   // and keep it in range
        index = 0;
    }
    else if(index >= 11)
    {
        index = 10;
    }

    if(thumb >= 3)                      // Test for reverse or forward (thumb status)
    {
        index = index + 64;             // It thumb is bent, bit tag "64" for reverse
```

Important code to notice above are statements were we read specific pins like "hardware.pin9.read", this in fact is reading the value at pin 9. Each pin is set up to be an ANALOG pin (ADC).

```
}
else
    index = index + 32;             // If not, bit tag "32" for forward

//*************Middle Finger*************//

buffer_middle = middle;             // Save previous value
middle = hardware.pin2.read();      // Read pin 2

middle = (((middle/1000)-40)/2);    // Adjust value for tolerance
middle = middle.tointeger();        // Convert from float to integer

if(middle <= -1)                    // Filter negative numbers
{                                   // and keep it in range
    middle = 0;
}
else if(middle >= 11)
{
    middle = 10;
}

if(thumb >= 3)                      // Test for reverse or forward (thumb status)
{
    middle = middle + 64;           // It thumb is bent, bit tag "64" for reverse
}
else
    middle = middle + 32;           // If not, bit tag "32" for forward

//*************Ring Finger***************//

ring = hardware.pin1.read();        // Read pin 1

ring = ((ring/1000)-30);            // Adjust value for tolerance
ring = ring.tointeger();            // Convert from float to integer

if(ring <= -1)                      // Filter negative numbers
{                                   // and keep it in range
    ring = 0;
}
else if(ring >= 11)
{
    ring = 10;
}
```

There are a few other things to notice. First we save values into a buffer value for each pin. What we do with this buffered data is test the new data to see if it changed. Only if there is a change is the data sent out. This will allow us to save battery power, and resources over Wi-Fi.

Each pin has mathematical calculations that are used to adjust the value. This mathematical equation will give us the range we want (0 – 10), and also adjust the value for the tolerance error of the electrical components (resistor and flex sensor).


```
//*************X-Axis*******************//

buffer_x = x;                       // Save previous value
x = hardware.pin7.read();           // Read pin 7

x = (((x/1000)-26)/2);              // Adjust value for tolerance
x = x.tointeger();                  // Convert from float to integer
x=10-x;                             // Reverse the numbers to correct direction

if(x <= -1)                         // Filter negative numbers
{                                   // and keep it in range
    x = 0;
}
else if(x >= 11)
{
    x = 10;
}

if(x <= 3 && x >= 0)                // If hand is tilted left
    x = 1;                          // Set output value to 1 (atemga significance) (camera left)
else if(x <= 10 && x >= 7)          // If hand is tilte right
    x = 10;                         // Set output value to 10 (atmega significance) (camera right)
else                                // If not, hand is centered
    x = 5;                          // Set output value to 10 (atmega significance) (camera centered)

//*************Y-Axis*******************//

buffer_y = y;                       // Save pervious value
y = hardware.pin8.read();           // Read pin 8

y = (((y/1000)-22)/2);              // Adjust value for tolerance
y = y.tointeger();                  // Convert float to integer
y = 10 - y;                         // Reverse the numbers to correct direction

if(y <= -1)                         // Filter negative numbers
{                                   // and keep it in range
    y = 0;
}
else if(y >= 11)
{
    y = 10;
}

if(y <= 4 && y >= 0)                // If hand is tilted forward
    y = 10;                         // Set output value to 10 (atmega significance) (camera down)
else if(y <= 10 && y >= 6)          // If hand is tilted backward
```

In the code we also include limitations so that when someone with a different size hand wears the glove, the size of hand will not affect the data.


```
    y = 5;                          // Set output value to 5 (atmega significance) (camera up)
else                                // Hand is centered
    y = 10;                         // Set output value to 10 (atmega significance) (camera down)

 //************Output Values***************//

if (thumb == 2 && ring == 10 && y == 5) // Test for led toggle
{
    buffer = 128;                   // Bit tag "128"
    output_index.set(buffer);       // Send value to left atmega
    output_middle.set(buffer);      // Sent value to right atmega
}
else if(thumb >= 3 && index == 74 && middle == 74 && ring == 10)
{                                   // Test for fist
    if(x == 1)                      // If hand tilted left
    {
       buffer = 74;                 // Bit tag "64" (reverse) at full speed 10 -> 74
       output_index.set(buffer);    // Output that to left atmega
       buffer = 42;                 // Bit tag "32" (forward) at full speed 10 -> 42
       output_middle.set(buffer);   // Output that to right atmega
    }
    else if (x == 10)               // If hand tilted right
    {
        buffer = 42;                // Bit tag "32" (forward) at full speed 10 -> 42
        output_index.set(buffer);   // Output that to left atmega
        buffer = 74;                // Bit tag "64" (reverse) at full speed 10 -> 74
        output_middle.set(buffer);  // Output that to right atmega
    }
    else                            // If hand is centered during fist
    {
        buffer = 32;                // Bit tag "32" (forward) stopped at 0 -> 32
        output_index.set(buffer);   // Output to left atmega
        output_middle.set(buffer);  // Output to right atmega
    }

}
else                                // If led toggle gesture not sensed, and fist not sensed
{

    if(buffer_index != index)       // Test to see if index finger value changed
    {
        output_index.set(index);    // If changed send index value to left atmega
    }

    if(buffer_middle != middle)     // Test to see if middle finger value changed
    {
```

This section of the code we output the values and also test for the advanced controls of the G.O.S.T. What the sensory glove will do is test for the LED toggle hand gesture first, if it is sensed it will act accordingly. Then if that is false it will test for the first hand gesture, if this is true it will spin the Smart Tank in a direction based on the tilt of the hand. Finally if those two gestures are not found, the Sensory Glove will send the raw data, (if it changed) to the cloud server.


```
            output_middle.set(middle);      // If changed send middle value to right atmega
        }

        if(ring >= 5 && buffer_x != x)      // Test to see if ring finger is bent, and x axis value changed
        {
            output_x_axis.set(x);           // If both true, send to camera atmega (pan)
        }

        if(ring >= 5 && buffer_y != y)      // Test to see if ring finger is bent, and y axis value changed
        {
            output_y_axis.set(y);           // If both true, send to camera atmega (tilt)
        }

    }
    //Display values in log

   server.log(format("Thumb >> ( %d ) Index >> ( %d ) Middle >> ( %d ) Ring >> ( %d ) X-Axis >> ( %d ) Y-Axis >>

    // Schedule the next read (every 2ms)

   imp.wakeup(0.2, read);

}


// Register with the server
imp.configure("read ADC", [], [output_index,output_middle,output_x_axis,output_y_axis]);

 // Configure pin 9 as an ADC Pin
hardware.pin9.configure(ANALOG_IN);

// Configure pin 8 as an ADC Pin
hardware.pin8.configure(ANALOG_IN);

// Configure pin 7 as an ADC Pin
hardware.pin7.configure(ANALOG_IN);

// Configure pin 5 as an ADC Pin
hardware.pin5.configure(ANALOG_IN);

// Configure pin 2 as an ADC Pin
hardware.pin2.configure(ANALOG_IN);

// Configure pin 1 as an ADC Pin
hardware.pin1.configure(ANALOG_IN);

// Start blinking
read();

// End of code.
```

First thing to notice on this screen is the server.log. This statement allows us to put debug messages on the online compiler of the Electric IMP. This is very useful to see what is going on with the IMP. In this case we show the values of each sensor on the log screen.

Another thing to notice is the wakeup function. This function will sleep the IMP for 20ms to allow for data transfer. Once 20ms is up, it will restart the whole process.

Then we have the hardware.pinx.configure(ANALOG_IN); This sets the corresponding pin to an ADC pin, allowing use to read the analog value. The IMP has a 16 bit resolution for its ADC inputs. This means 65535 possibilities.

3.5 *Smart Tank*

3.5.1 *Electric IMP*

Electric IMP wiring diagram:

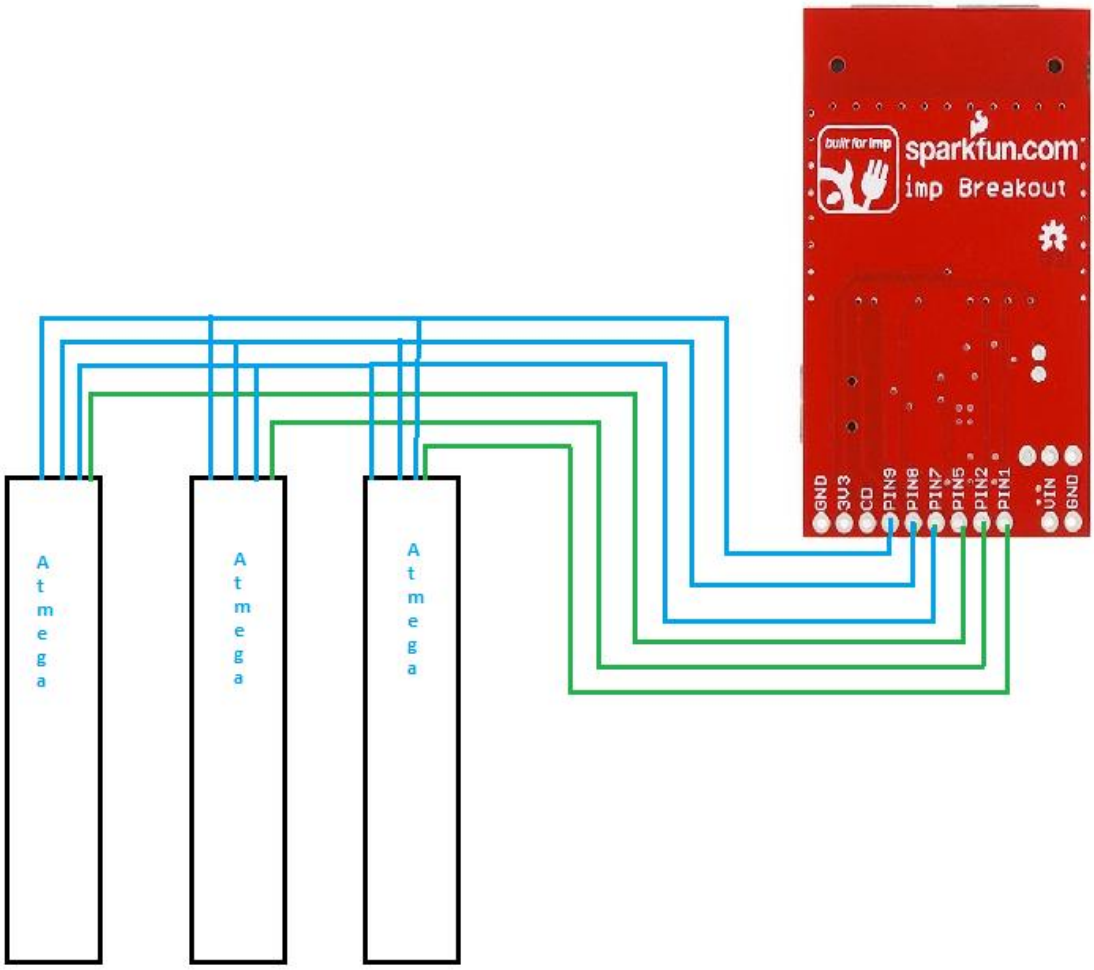


Figure 8: Electric IMP wiring diagram

```
////////////////////////////////////////////////////////////////////
// G.O.S.T System                                                 //
// Created by: Michael Scalera and Matthew Masi                   //
// Masi & Scalera Innovations                                     //
// Smart Tank Code                                                //
// Revision 1                                                     //
//                                                                //
// This code is open source, and may be edited.                   //
// Any change to this code may hinder the performance of your     //
// G.O.S.T System. We are not responsible for any damage you cause.//
//                                                                //
//                                                                //
////////////////////////////////////////////////////////////////////


//**************>> MAIN CODE <<**************//

//****LEFT TRACK ATMEGA HANDLER****//

class index_input_handler extends InputPort
{
    name = "index"
    type = "number"

    function set(value)
    {

        if(value >= 32 && value <= 63)              // Test for bit tag "32"
        {
            if (value >= 33)
                server.log("Left Track Forward");   // Display left track going forward if there is a speed value
            else
                server.log("Left Track Stop");      // If value is 0, display left track as stopped
        }
        else if(value >= 64)                        // Test for bit tag "64"

            if (value >= 65)
                server.log("Left Track Reverse");   // Display left track going backward if there is a speed value
            else
                server.log("Left Track Stop");      // If value is 0, display left track as stopped

        else
            server.log("LED Toggle")                // If not LED is being toggled (bit tag "128")


        value = value.tochar();                     // Change value to character
                                                    // That is how the atmega needs it to be read properly
        hardware.pin5.write(0);                     // Ground chip select on Left Track Atmega

        hardware.spi189.write(value);               // Write value to SPI
        hardware.spi189.read(1);                    // Bogus read, for IMP purposes
```

Most of what was seen in the Sensory glove can be applied here. The new element is the class function. This function is in other words a handler for the input values it receives from the cloud server. The class function will receive and hand the data off in a function within the class function which then can run a set of commands.


```
        hardware.pin5.write(1);                     // Bring chip select back high

    }
}

//****RIGHT TRACK ATMEGA HANDLER****//

class middle_input_handler extends InputPort
{
    name = "middle"
    type = "number"

    function set(value)
    {
        if(value >= 32 && value <= 63)              // Test for bit tag "32"
        {
            if (value >= 33)
                server.log("Right Track Forward");  // Display right track going forward if there is a speed value
            else
                server.log("Right Track Stop");     // If value is 0, display right track as stopped
        }
        else if(value >= 64)                        // Test for bit tag "64"

            if (value >= 65)
                server.log("Right Track Reverse");  // Display right track going backward if there is a speed value
            else
                server.log("Right Track Stop");     // If value is 0, display right track as stopped

        else
            server.log("LED Toggle")

        value = value.tochar();                     // Change value to character
                                                    // That is how the atmega needs it to be read properly
        hardware.pin2.write(0);                     // Ground chip select on Right Track Atmega

        hardware.spi189.write(value);               // Write value to SPI
        hardware.spi189.read(1);                    // Bogus read, for IMP purposes

        hardware.pin2.write(1);                     // Bring chip select back high

    }
}

//****CAMERA PAN ATMEGA HANDLER****//

class x_input_handler extends InputPort
{
    name = "x"
    type = "number"

    function set(value)
```

What is being done in this code is that, every time data is being received from a certain input port, it will be sent out SPI with the corresponding ATMEGA8L chip grounded.

```
    {
        if (value == 1)                         // Test value for left tilt
            server.log("Pan -> Left");          // Display panning left
        else if(value == 10)                    // Test value for right tilt
            server.log("Pan -> Right")          // Display panning right
        else                                    // Value is centered
            server.log("Pan -> Center")         // Display centered

        value = value + 64;                     // Bit tag "64" to allow atmega to distinguish between pan and tilt
        value = value.tochar();                 // Convert to character for atmega

        // Display on our Planner node
        hardware.pin7.write(0);                 // Ground chip select for camera atmega

        hardware.spi189.write(value);           // Write value to SPI
        hardware.spi189.read(1);                // Bogus read for IMP purposes

        hardware.pin7.write(1);                 // Bring chip select for camera atmega back high

    }
}

//****CAMERA TILT ATMEGA HANDLER****//

class y_input_handler extends InputPort
{
    name = "y"
    type = "number"

    function set(value)
    {
        if (value == 10)                        // Test for centering
            server.log("Tilt center");          // Display tilt is centered
        else                                    // If not camera is up
            server.log("Tilt Up");              // Display tilt is up

        value = value + 32;                     // Bit tag "32" to distinguish pan/tilt
        value = value.tochar();                 // Convert to character for atmega

        hardware.pin7.write(0);                 // Ground chip select for camera atmega

        hardware.spi189.write(value);           // Write value to SPI
        hardware.spi189.read(1);                // Bogus rad for IMP purposes

        hardware.pin7.write(1);                 // Bring chip select for camera atmega back high

    }
}
```

```
// Configure SPI parameters
hardware.spi189.configure(MSB_FIRST, 3000);

// Configure Pin 5 as a digital out for Atmega chip select
hardware.pin5.configure(DIGITAL_OUT);
hardware.pin5.write(1);

// Configure Pin 2 as a digital out for Atmega chip select
hardware.pin2.configure(DIGITAL_OUT);
hardware.pin2.write(1);

// Configure Pin 7 as a digital out for Atmega chip select
hardware.pin7.configure(DIGITAL_OUT);
hardware.pin7.write(1);

// Tie input port values to handlers
// These handlers will be called once data is received on the resepective value
local index = index_input_handler();
local middle = middle_input_handler();
local x = x_input_handler();
local y = y_input_handler();

// Configure IMP with input port values
imp.configure("Rover", [index,middle,x,y], []);
```

Here two new types of pins are configured. DIGITAL_OUT type of pin is for pins for logical high or low, simply enough. The second type of pin seen above is the hardware.spi189.configure(MSB_FIRST, 3000). What this does is set pins 1, 8, and 9 as SPI communication pins, and then sets parameters for the SPI transmission. In this case we use MSB_FIRST, which means that the most significant bit will be sent out SPI first and 3000 means the SPI transfer rate will be close to 3 KHz.

3.5.2 *Motor Control Circuit Overview*

Component	Description
ATMEGA8L	Controls left DC motors
ATMEGA8L	Controls right DC motors
ATMEGA8L	Controls left DC motors
4081 Quad Input CMOS AND Gate	Selects H-Bridge Terminal

Table 1: Motor Control Circuit component List

ATMEGA8L DC Motor Controller Operation

Two out of the three ATMEGA8L microprocessors are used to control left and right DC motors. Data is received by these ATMEGA8L microprocessors through SPI communication. The ATMEGA8L, enabled as a slave, receives data from the Wi-Fi enabled Electric Imp microprocessor which is enabled as the master in the communication. The data is processed and converted into different PWM signals according to the received value. The PWM signals have ten steps and are varied from 55% to 100% at a 35 kHz frequency. The PWM signal is fed to two different two input CMOS AND gates and are each ANDed with addition control signals coming from the ATMEGA8L. The control signals are determined by bit tagged data that is received by the ATMEGA8L. One of the control signals is on at a time which enables the output of only one of the CMOS AND gate outputs at a time. The two outputs of the CMOS AND gates are then fed to two inputs of an H-bridge circuit to enable the motors in a forward or reverse state. These two motor controllers also control whether a set of LEDs are turned on or off using bit tagging to multiplex the LEDs with the PWM signals.

ATMEGA8L Servomotor Controller Operation

The third ATMEGA8L used in the motor control circuit is used to control a servomotor which holds a camera. The servomotor has pan ad tilt (x-axis and y-axis movements) functionalities used to position the camera at different angles. The data received by this ATMEGA8L is also received through SPI by the Electric Imp microprocessor. The data is processed and converted into signals with different pulse lengths according to the received value. There are 10 different signal pulses ranging from 180us to 1.8ms which correspond to a certain angle that the servomotor will be positioned at. To determine whether the servomotor will be panning or tilting, the received data is bit tagged. If the data is tagged with a value of 64, the servomotor rotates along the x-axis. If the data is tagged with a value of 32, the servomotor rotates along the y-axis.

3.5.3 *SPI Communication*

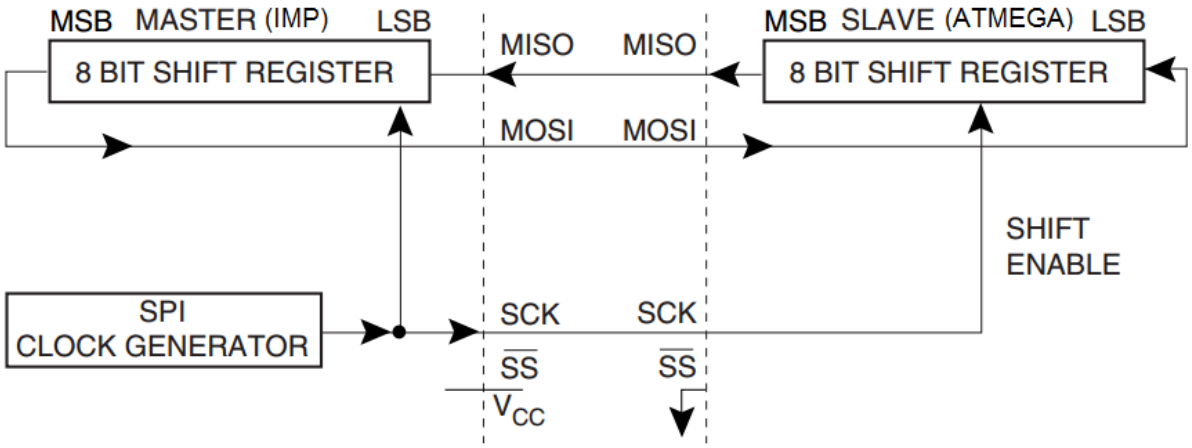


Figure 9: SPI Master-Slave Interconnection

The SPI master (IMP) initiates the communication cycle by pulling low the slave select (~SS) pin of one of the three desired slaves (ATMEGA8Ls). The Master and Slave prepare the data to be sent in their respective Shift Registers, and the Master generates the required clock pulses on the SCK line to interchange data. The data is shifted out from the Master to the Slave on the Master Out – Slave In (MOSI) line. After one byte has been completely shifted, the end of Transmission Flag, SPIF is set. The SPI interrupt enable bit, SPIE, in the SPCR Register is set, so an interrupt is then requested.

Pin	Direction, Slave SPI
MOSI	Input
MISO	Output
SCK	Input
~SS	Input

Table 2: SPI slave pin configuration

The ATMEGA8Ls on the motor control circuit are all configured as slaves. The MOSI pin is left as an input because it receives the data from the master on this pin. The SCK pin is configured as an input so it can receive the clock from the master. The ~SS pin is set as an input because it is waiting for the master to set it low for communication to start. The MISO must be configured as an output on a slave device but in this case, it is not used.

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 10: SPI Control Register – SPCR

Two bits are enabled in the SPCR control register for SPI to function between the ATMEGA8L and the Electric IMP:

Bit 7 – SPIE: SPI Interrupt Enable

This bit causes the SPI interrupt to be executed if SPIF bit in the SPSR Register is set and if the global interrupt enable bit in SREG is set.

Bit 6 – SPE: SPI Enable

When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

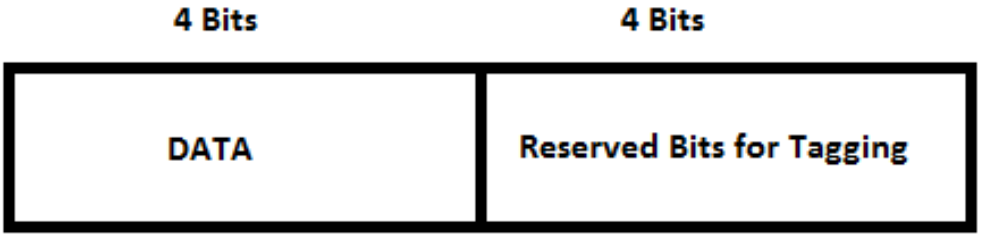


Figure 11: SPI Data Frame

The SPI data frame consists of 4 bits for the data, and 4 bits reserved for tagging to enable different functions.

DC Motor Controller SPI data frames:

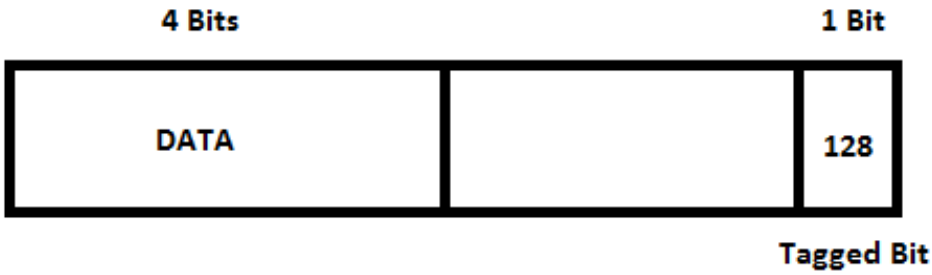


Figure 12: Data frame with eighth bit tagged

The eight bit (128) is tagged to enable the LEDs.

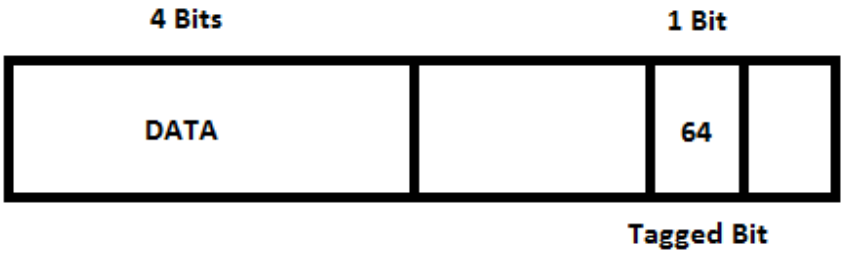


Figure 13: Data frame with seventh bit tagged

The seventh bit (64) is tagged to enable a pulse to put the motors in a forwarding motion.

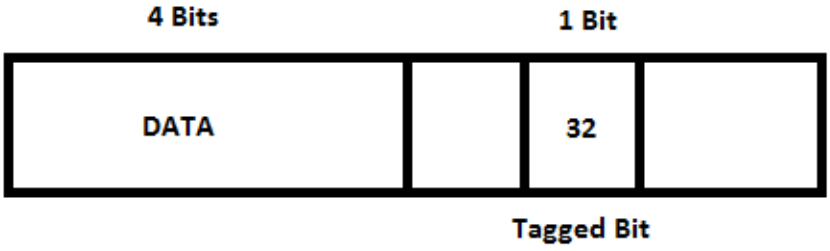


Figure 14: Data frame with sixth bit tagged

The sixth bit (32) is tagged to enable a pulse to put the motors in a reversing motion.

Servomotor Controller SPI Waveforms:

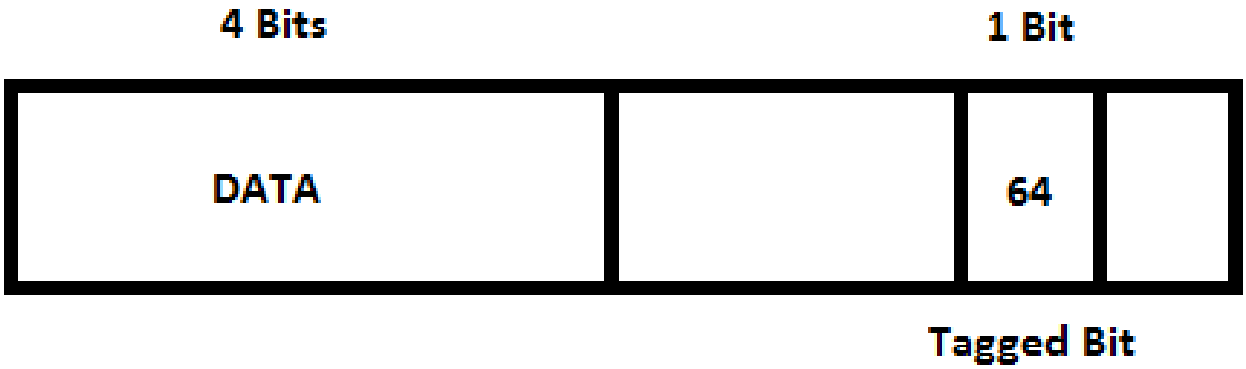


Figure 15: Data frame with seventh bit tagged

The seventh bit (64) is tagged to enable the x-axis movement of the servomotor.

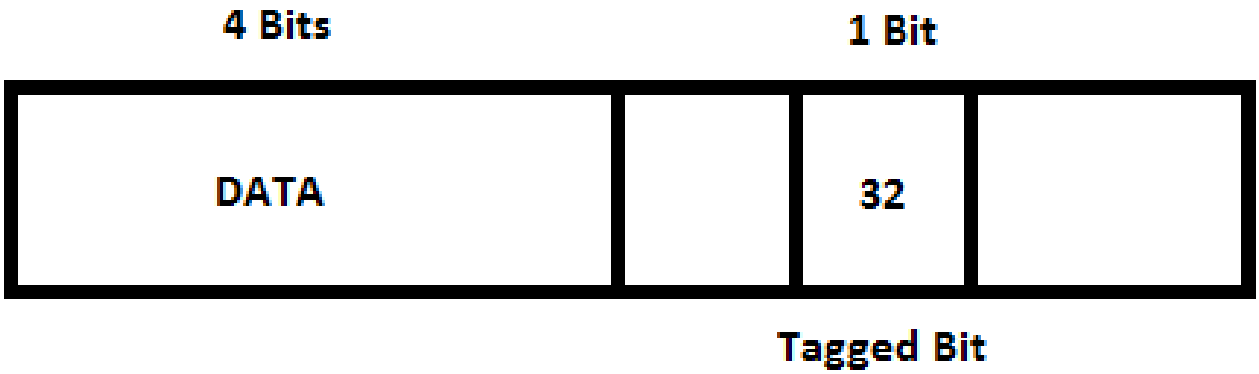


Figure 16: Data frame with sixth bit tagged

The sixth bit (32) is tagged to enable the y-axis movement of the servomotor.

3.5.4 *DC Motor Controller - PWM Output Signals*

SPI Data (after tagged bit is removed)	ON variable	OFF variable	Duty Cycle	Frequency
0	-	-	0%	35khz
1	11	9	55%	35khz
2	12	8	60%	35khz
3	13	7	65%	35khz
4	14	6	70%	35khz
5	15	5	75%	35khz
6	16	4	80%	35khz
7	17	3	85%	35khz
8	18	2	90%	35khz
9	19	1	95%	35khz
10	-	-	100%	35khz

Table 3: Relation between received data and outputted PWM signal

After the ATMEGA8L selects whether the motors will be forwarding or reversing, the tagged bit associated with the data is stripped away leaving the data to be a value of 0 to 10. When the data is 0, the signal is a constant low, and when the data is 10, the signal is a constant high. When the data is between 0 and 10, the duty cycle of the PWM signal is varied at nine different steps starting at 55% up until 95%.

PWM SIGNAL Step 0 – 0% Duty Cycle:

SPI Data (after tagged bit is removed)	ON variable	OFF variable	Duty Cycle	Frequency
0	-	-	0%	35khz

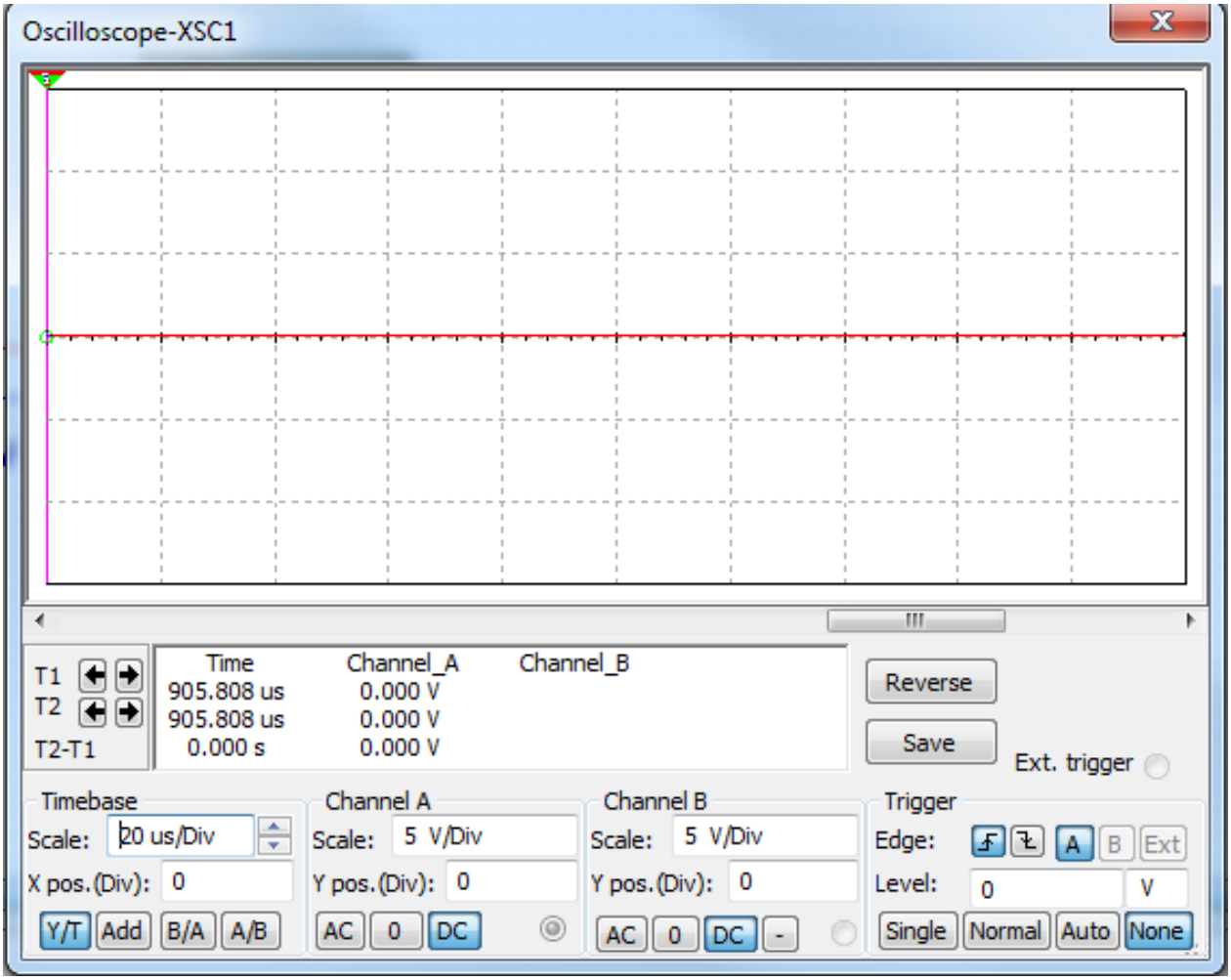


Figure 17: PWM signal with 0% duty cycle

PWM SIGNAL Step 1 – 55% Duty Cycle:

SPI Data (after tagged bit is re-moved)	ON variable	OFF variable	Duty Cycle	Frequency
1	11	9	55%	35khz

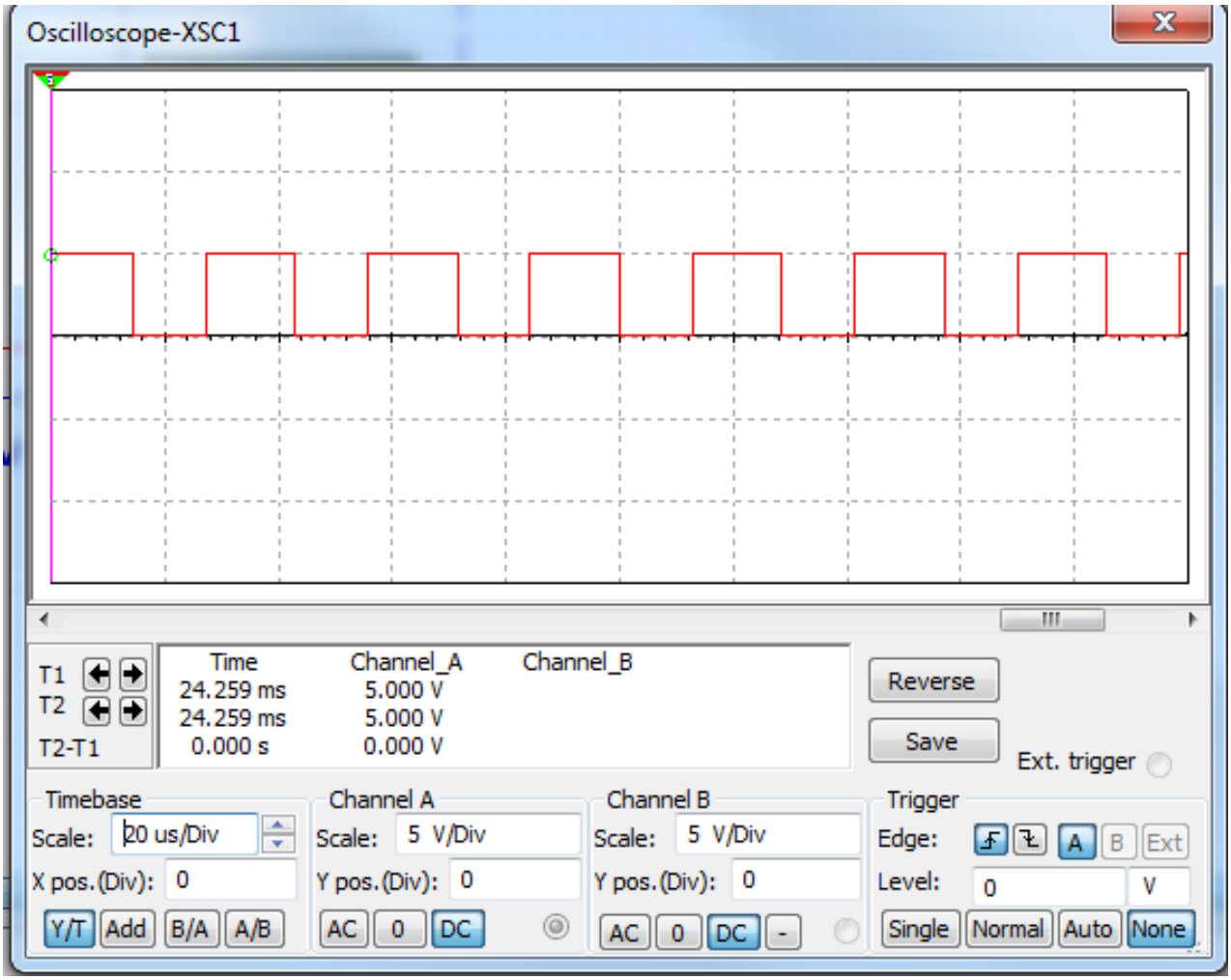


Figure 18: PWM signal with 55% duty cycle

PWM SIGNAL Step 2 – 60% Duty Cycle:

SPI Data (after tagged bit is removed)	ON variable	OFF variable	Duty Cycle	Frequency
2	12	8	60%	35khz

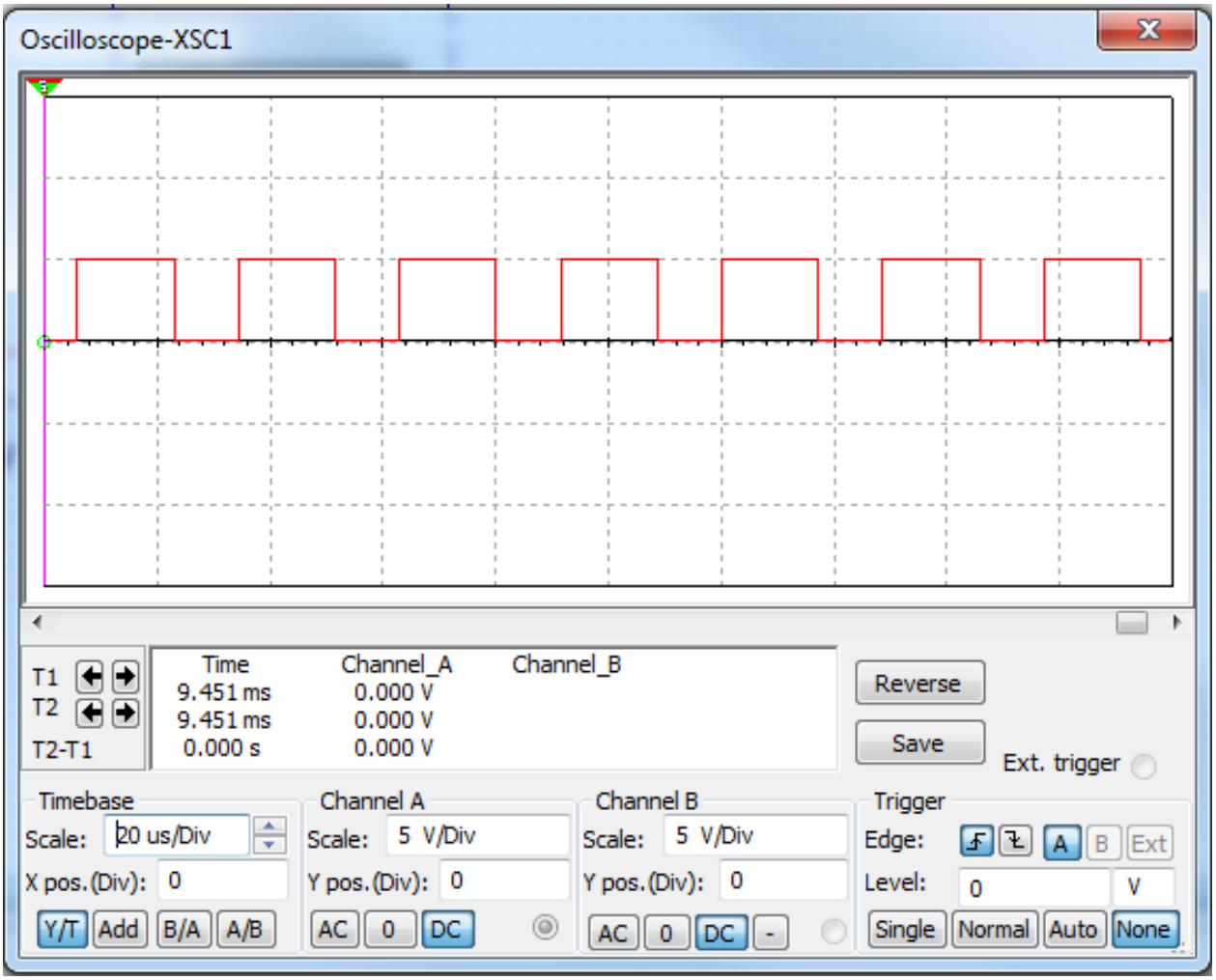


Figure 19: PWM signal with 60% duty cycle

PWM SIGNAL Step 3 – 65% Duty Cycle:

SPI Data (after tagged bit is removed)	ON variable	OFF variable	Duty Cycle	Frequency
3	13	7	65%	35khz

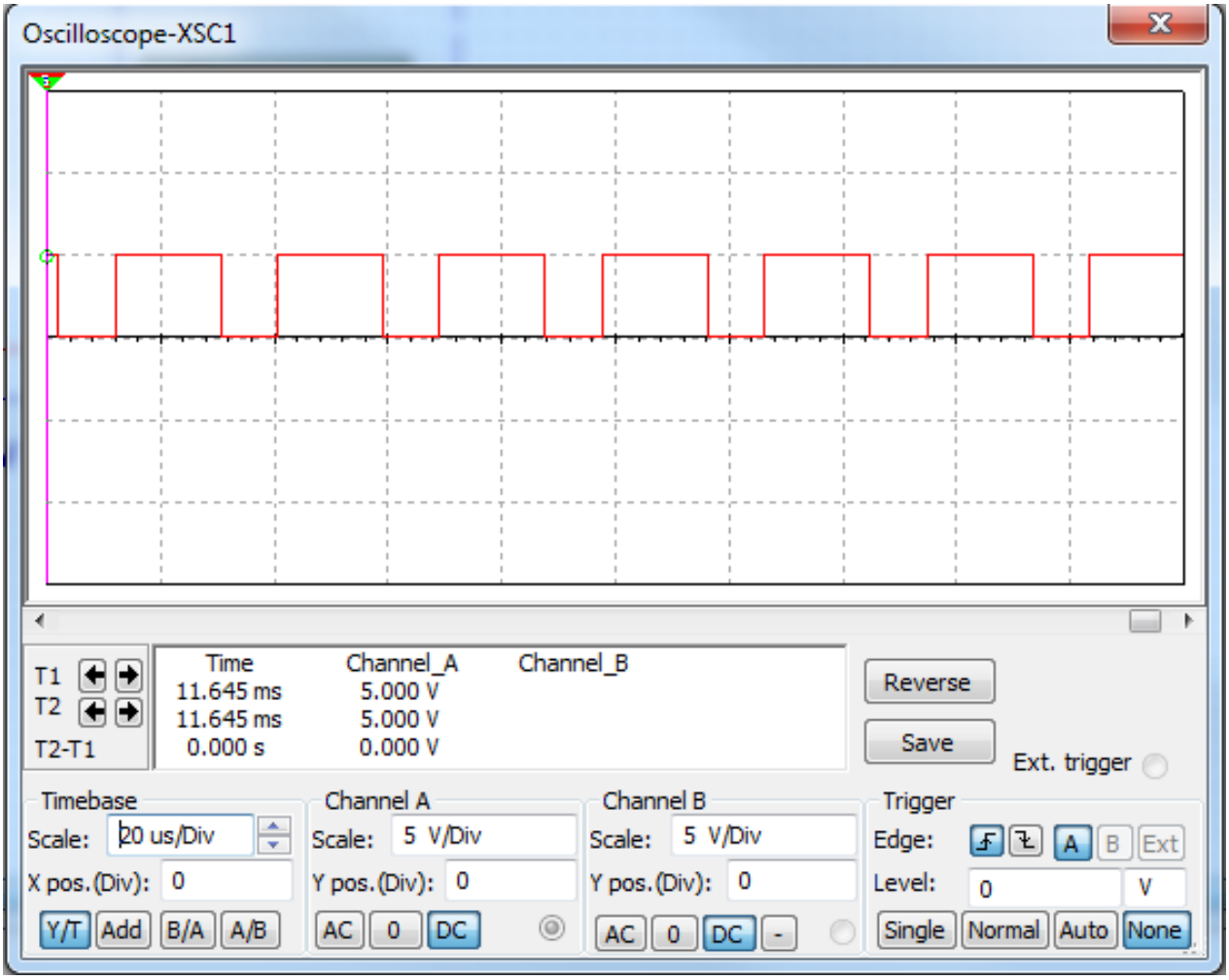


Figure 20: PWM signal with 65% duty cycle

PWM SIGNAL Step 4 – 70% Duty Cycle:

SPI Data (after tagged bit is removed)	ON variable	OFF variable	Duty Cycle	Frequency
4	14	6	70%	35khz

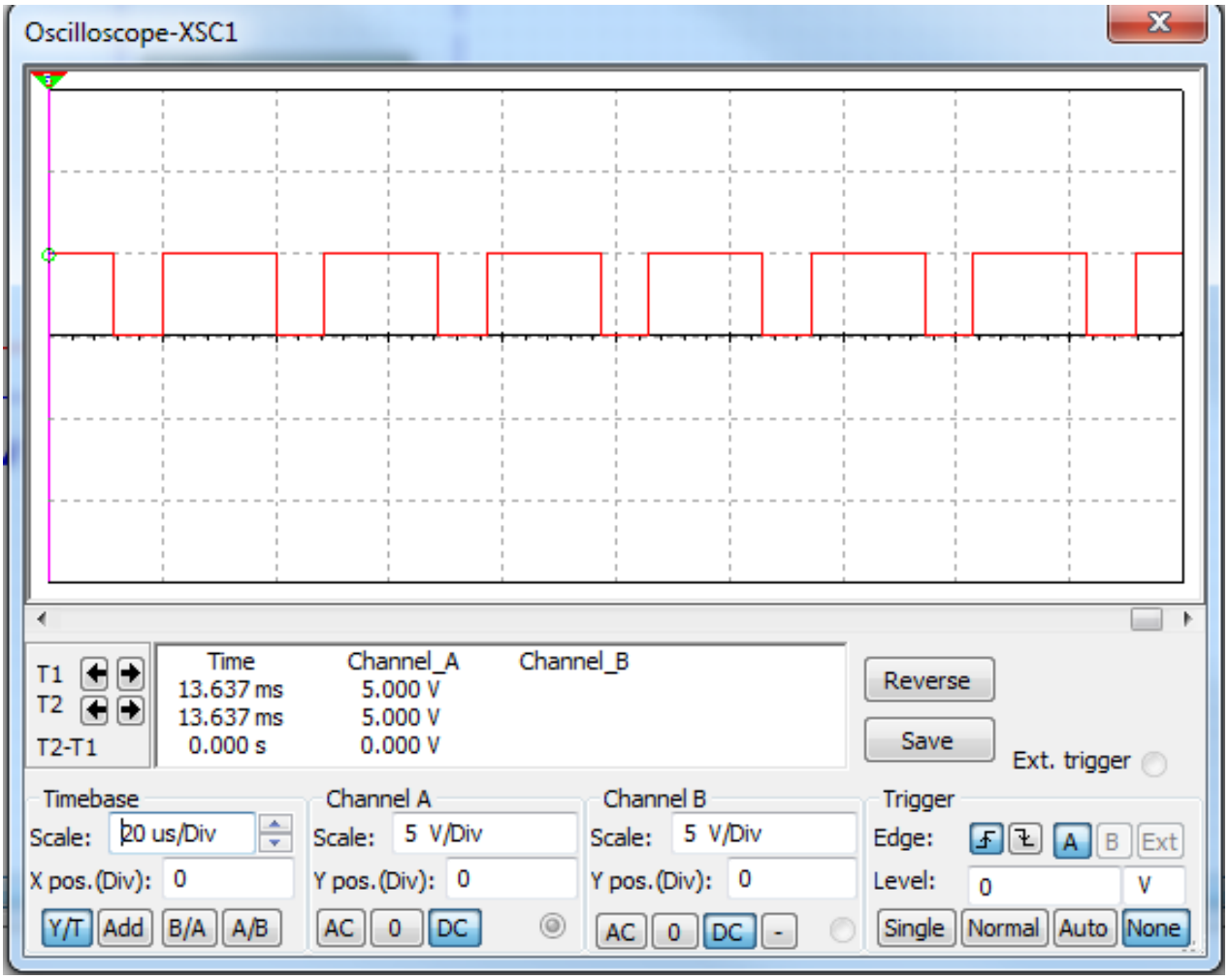


Figure 21: PWM signal with 70% duty cycle

PWM SIGNAL Step 5 – 75% Duty Cycle:

SPI Data (after tagged bit is removed)	ON variable	OFF variable	Duty Cycle	Frequency
5	15	5	75%	35khz

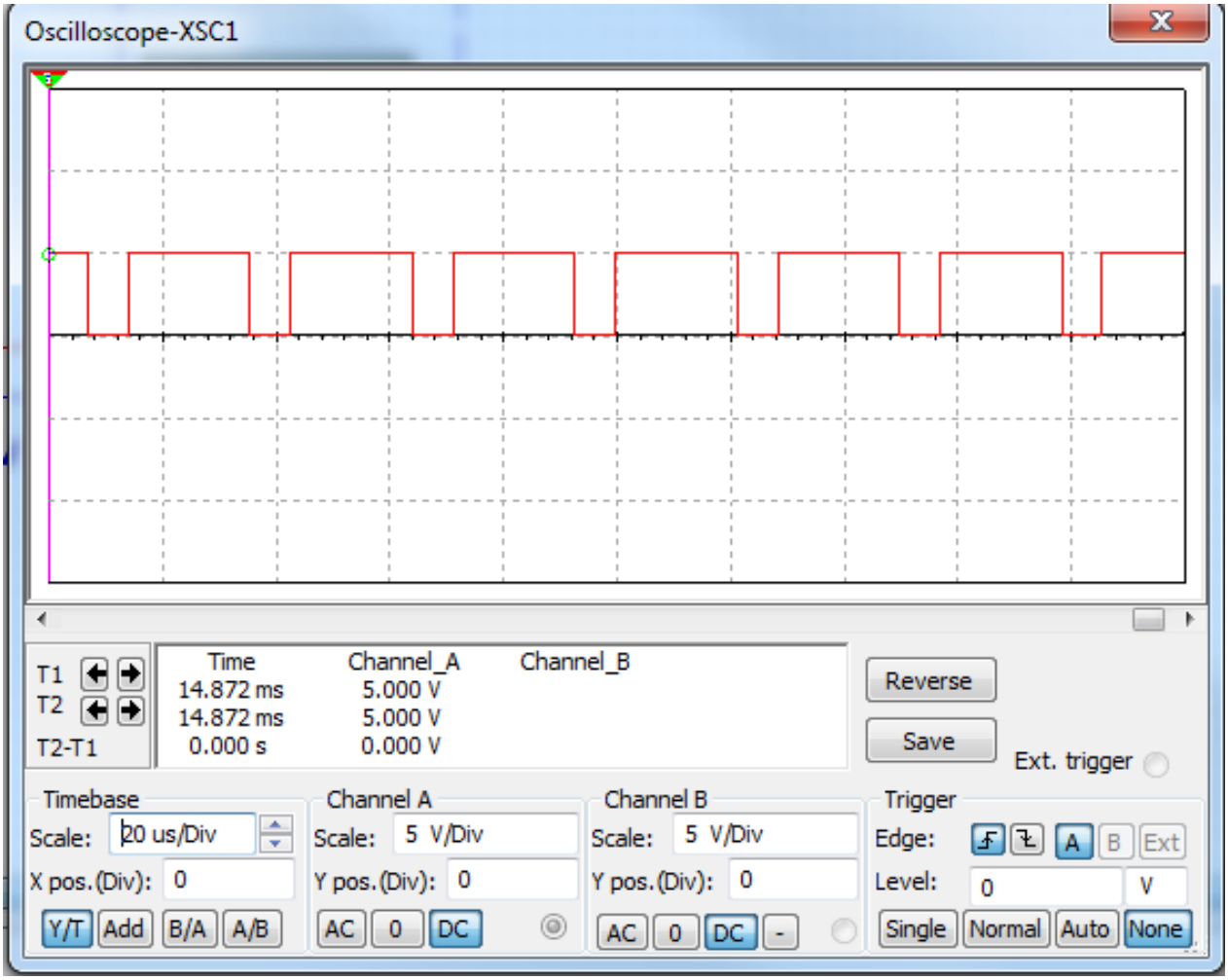


Figure 22: PWM signal with 75% duty cycle

PWM SIGNAL Step 6 – 80% Duty Cycle:

SPI Data (after tagged bit is removed)	ON variable	OFF variable	Duty Cycle	Frequency
6	16	4	80%	35khz

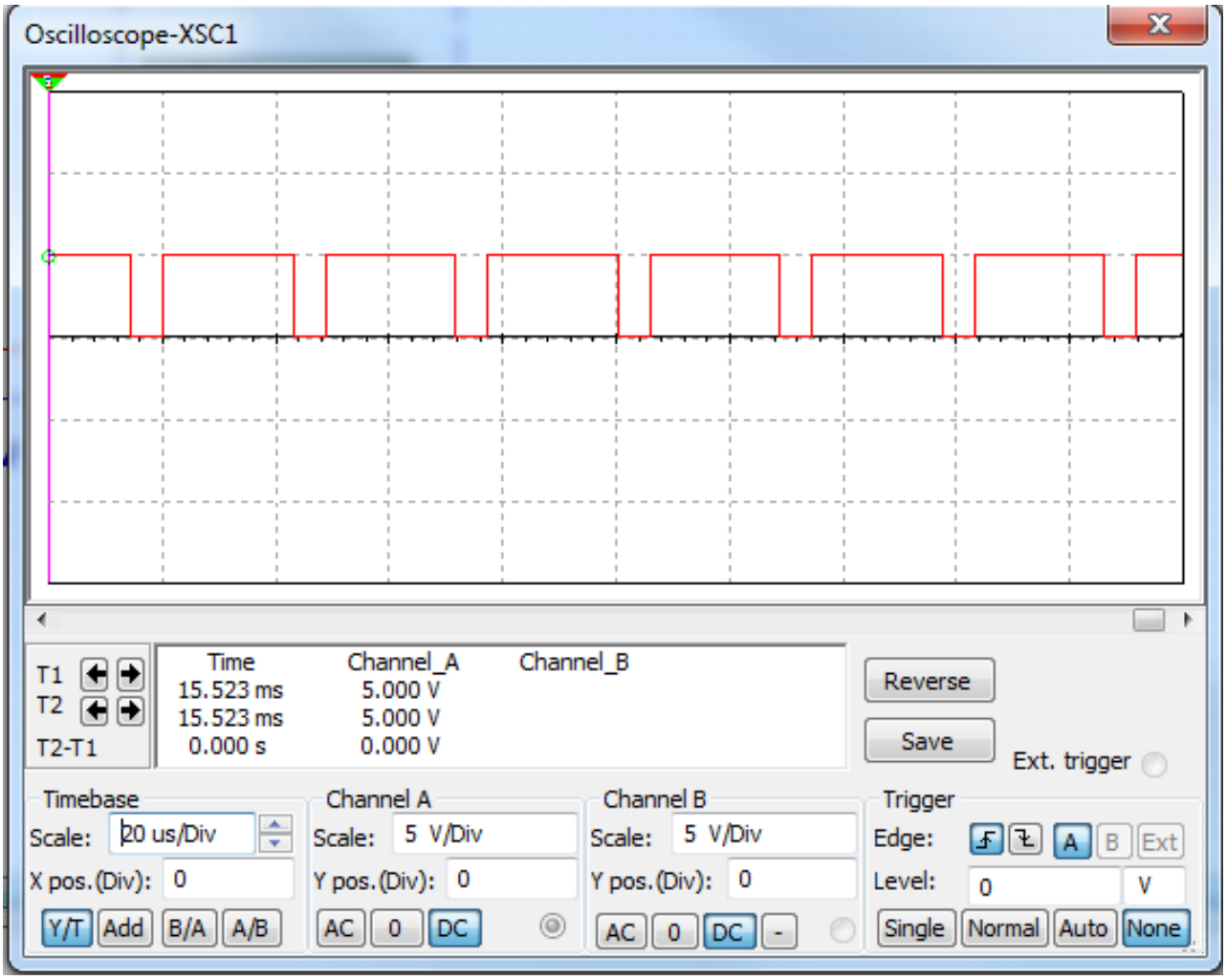


Figure 23: PWM signal with 80% duty cycle

PWM SIGNAL Step 7 – 85% Duty Cycle:

SPI Data (after tagged bit is removed)	ON variable	OFF variable	Duty Cycle	Frequency
7	17	3	85%	35khz

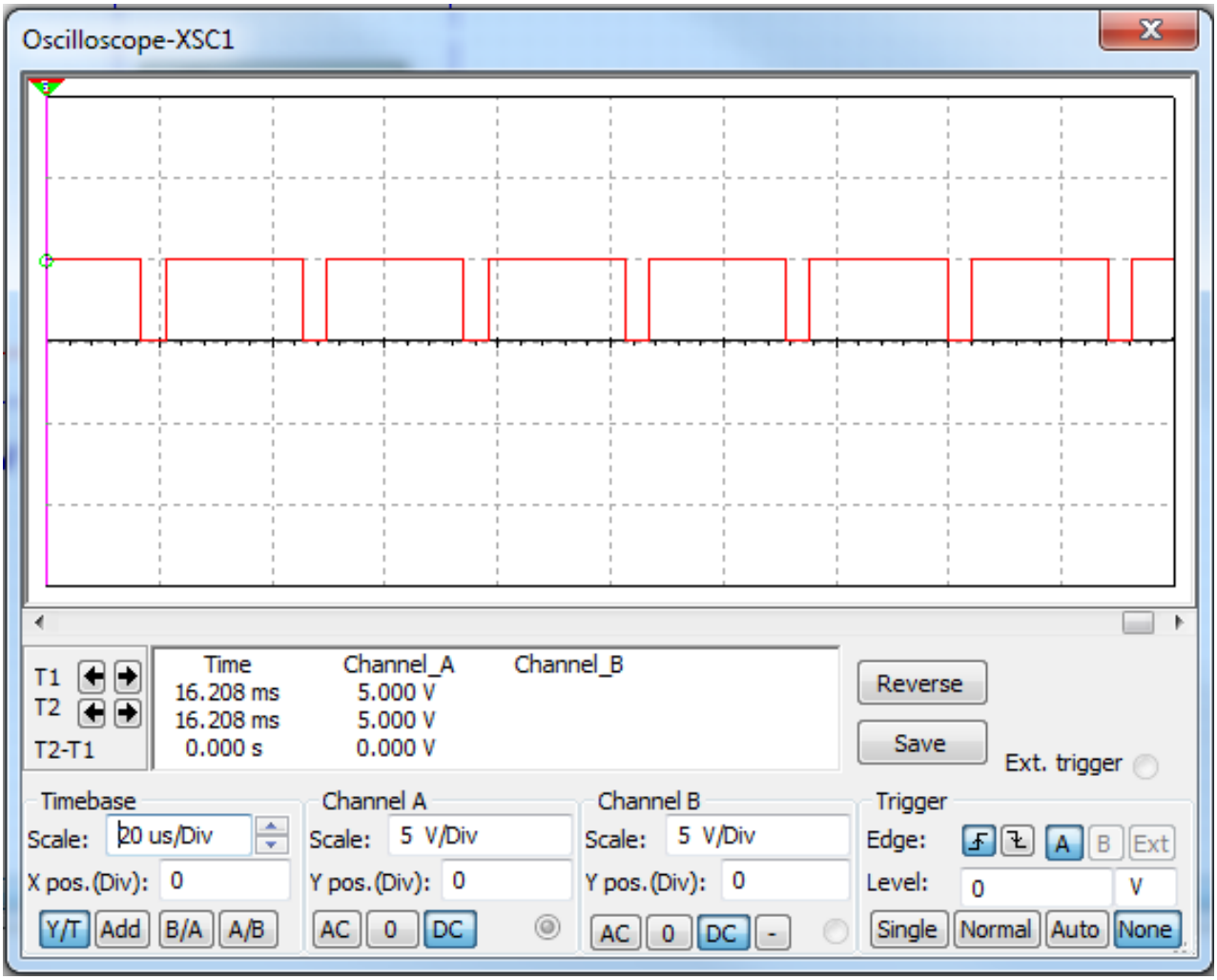


Figure 24: PWM signal with 85% duty cycle

PWM SIGNAL Step 8 – 90% Duty Cycle:

SPI Data (after tagged bit is removed)	ON variable	OFF variable	Duty Cycle	Frequency
8	18	2	90%	35khz

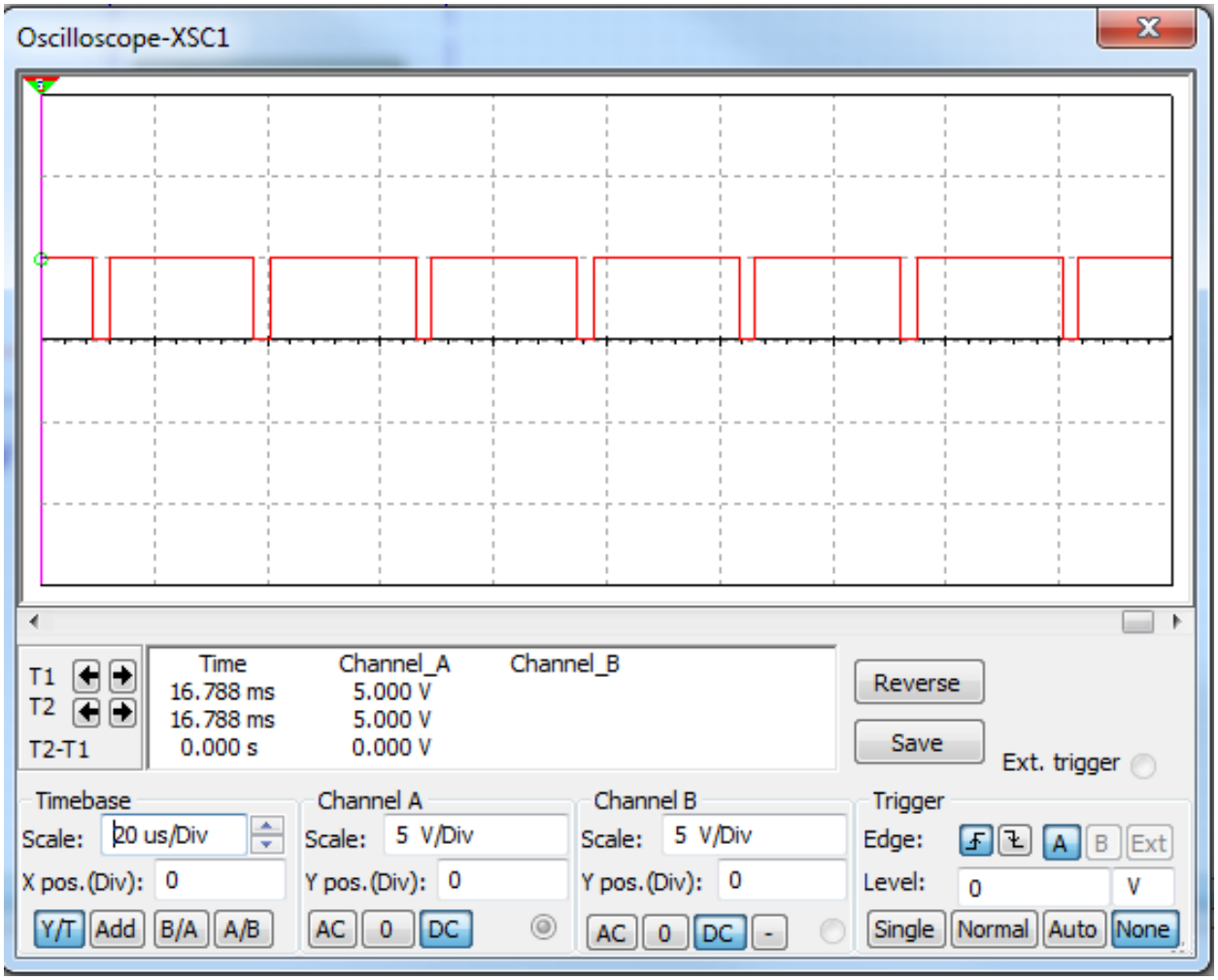


Figure 25: PWM signal with 90% duty cycle

PWM SIGNAL Step 9 – 95% Duty Cycle:

SPI Data (after tagged bit is removed)	ON variable	OFF variable	Duty Cycle	Frequency
9	19	1	95%	35khz

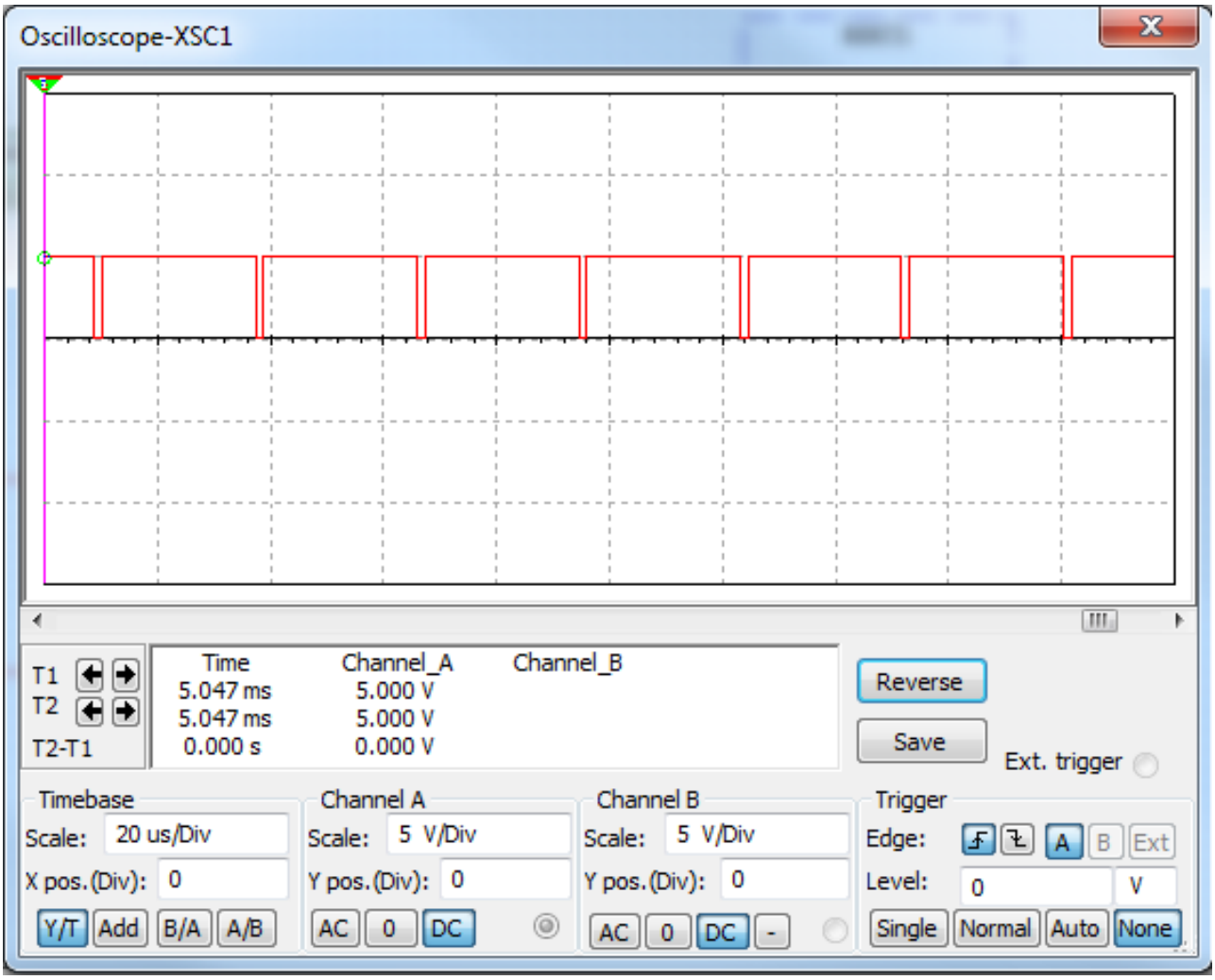


Figure 26: PWM signal with 95% duty cycle

PWM SIGNAL Step 10 – 100% Duty Cycle:

SPI Data (after tagged bit is re-moved)	ON variable	OFF variable	Duty Cycle	Frequency
10	-	-	100%	35khz

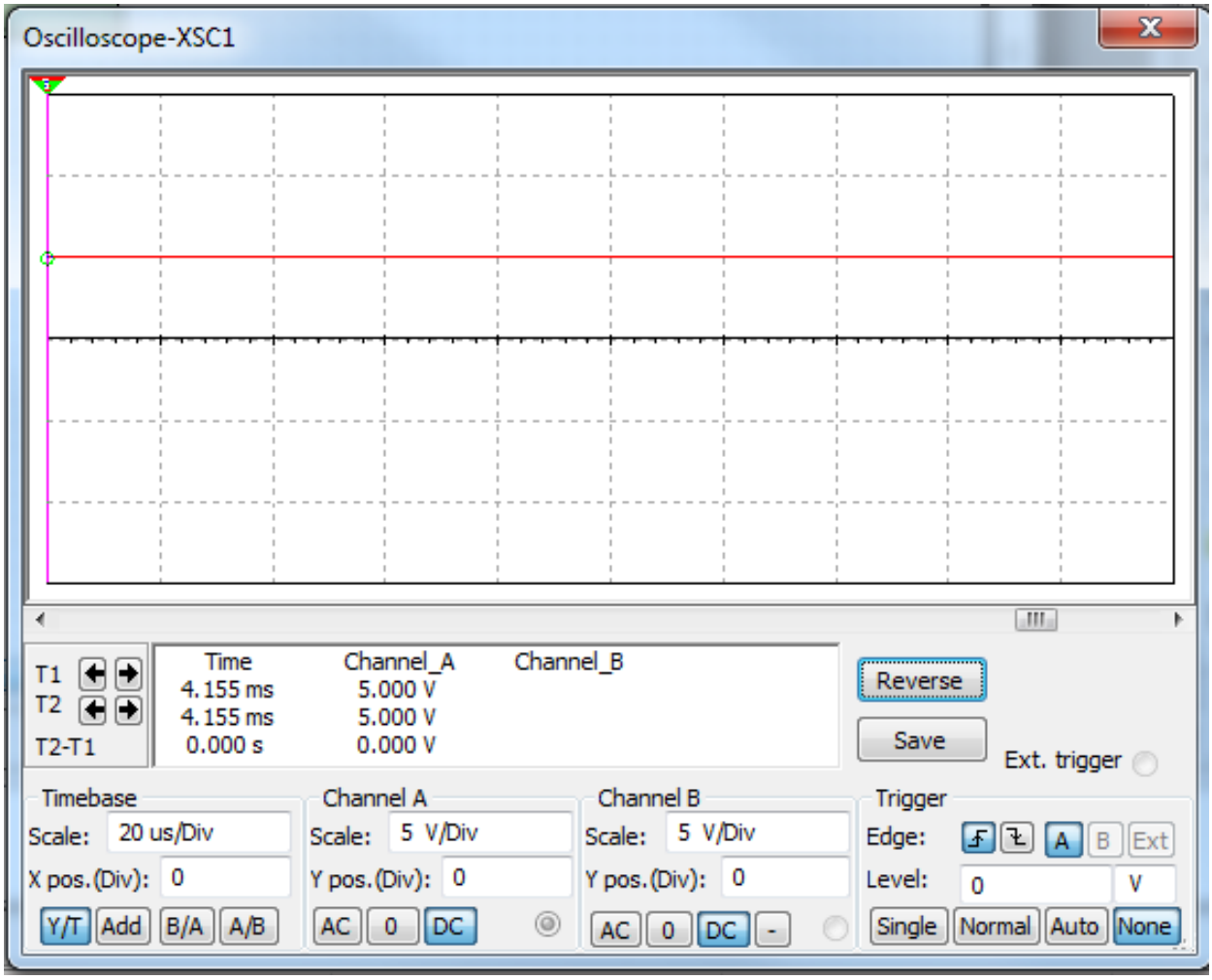


Figure 27: PWM signal with 100% duty cycle

3.5.5 *CMOS AND Gate & LED Output Signals*

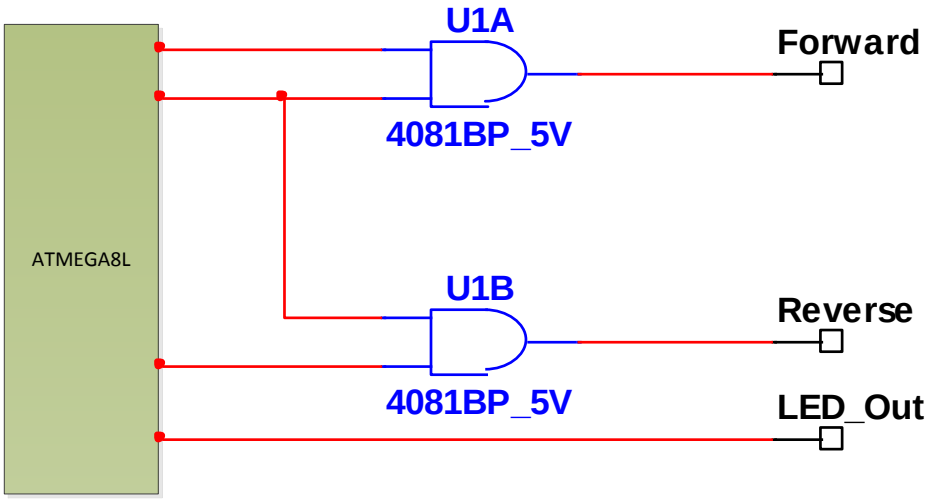


Figure 28: ATMEGA8L DC Motor Controller output signals

The CMOS AND gate is used in this control circuit to output the PWM signal at either the forward or reverse terminal of the H-bridge circuit. The PWM output signal is fed to one of the two inputs of two different AND gates. The other input of each AND gate is fed a control signal from the ATMEGA8L which determines which AND gate will output a signal. On control signal is enabled at a time enabling a PWM signal to either the forward or reverse terminals of the H-bridge circuit. The control signals are determined by data with a tagged bit at the end. Data with a tagged bit of 32 enables the motor in forward mode. Data with a tagged bit of 64 enables the motor in reverse mode.

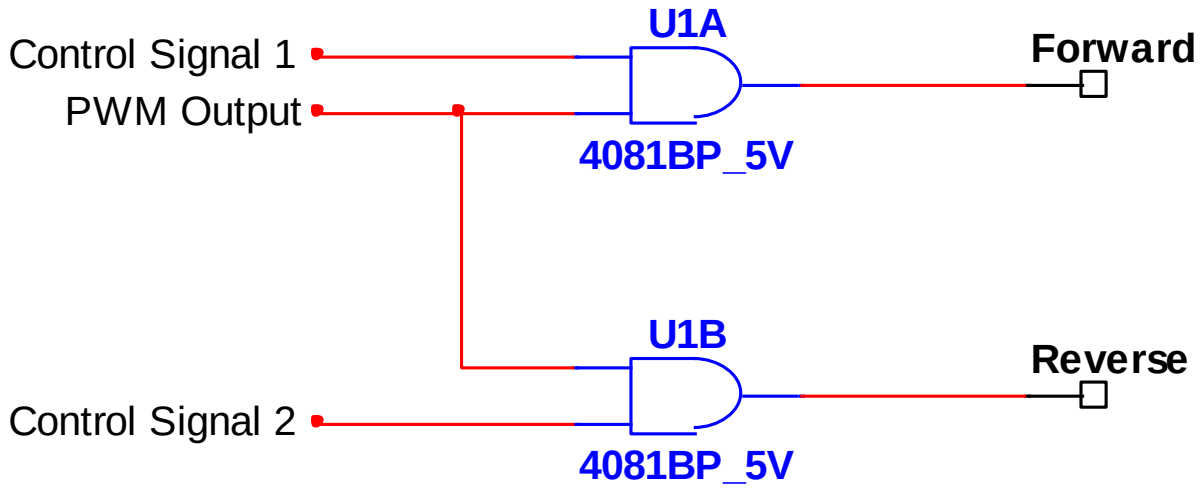


Figure 29: CMOS AND gates for direction control

The LEDs are controlled with tagged data as well; a tagged bit of 128 enables the LED. They can be enabled by themselves, while the motors are forwarding, or while the motors are reversing. The led signal is multiplexed by adding the value 2 to the data when data is tagged with a bit of 128. The following table explains when the LEDs are enabled, and when the motors are enabled for forwarding or reversing.

Control Signal 2	Control Signal 1	LED Signal	PWM Signal	Port Value	Output
0	0	0	0	0	No Signal
0	0	0	1	1	n/a
0	0	1	0	2	LED On
0	0	1	1	3	n/a
0	1	0	0	4	n/a
0	1	0	1	5	Forward
0	1	1	0	6	n/a
0	1	1	1	7	Forward + LED On
1	0	0	0	8	n/a
1	0	0	1	9	Reverse
1	0	1	0	10	n/a
1	0	1	1	11	Reverse + LED On

Table 4: ATMEGA8L port output values

3.5.6 *Servomotor Controller – Output Signals*

The function of the servo is to receive a control signal that represents a desired output position of the servo, and apply power to its DC motor until its shaft turns to that position. The servo has a 3 wire connection: power, ground, and control. The power source must be constantly applied; the servo draws current from the power lead to drive the motor. The control signal is pulse width modulated (PWM), but the duration of the high pulse determines the position of the servo shaft. A longer pulse makes the servo turn to a clockwise-from-center position, and a shorter pulse makes the servo turn to a counter-clockwise-from-center position. The servo control pulse is repeated every 20 milliseconds.

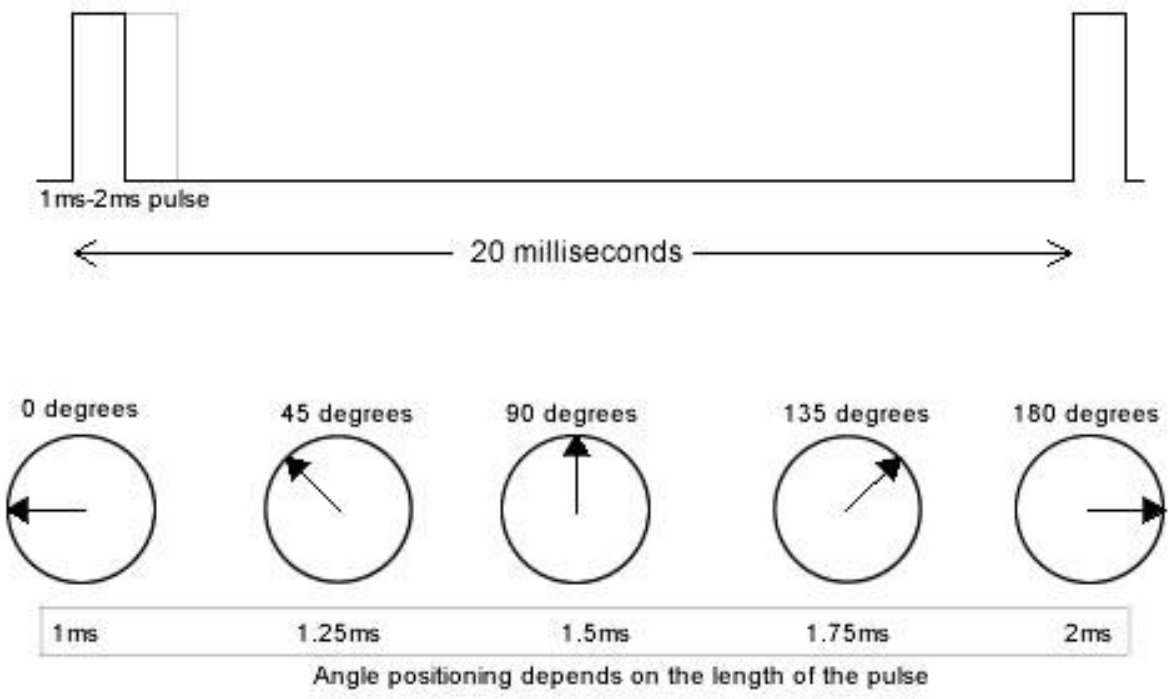


Figure 30: Servomotor angle at different pulse lengths

SPI Data (after tagged bit is removed)	Pulse Length (us)	Off Time (ms)	Angle (®)
0	1800	10	165
1	1620	10	150
2	1440	10	135
3	1260	10	120
4	1080	10	105
5	900	10	90
6	720	10	75
7	540	10	60
8	360	10	45
9	180	10	30
10	0	10	-

Table 5: Relationship between received data and output signal pulses

After the ATMEGA8L has selected which axis it will be using (x-axis or y-axis), the tagged bit is stripped off and a signal pulse is outputted according to the data. Figure 17 above shows the relation between the length of the outputted pulse and the angle at which the servomotor positions itself.

Servo Pulse 0 – 1800us:

SPI Data (after tagged bit is removed)	Pulse Length (us)	Off Time (ms)	Angle (®)
0	1800	10	165

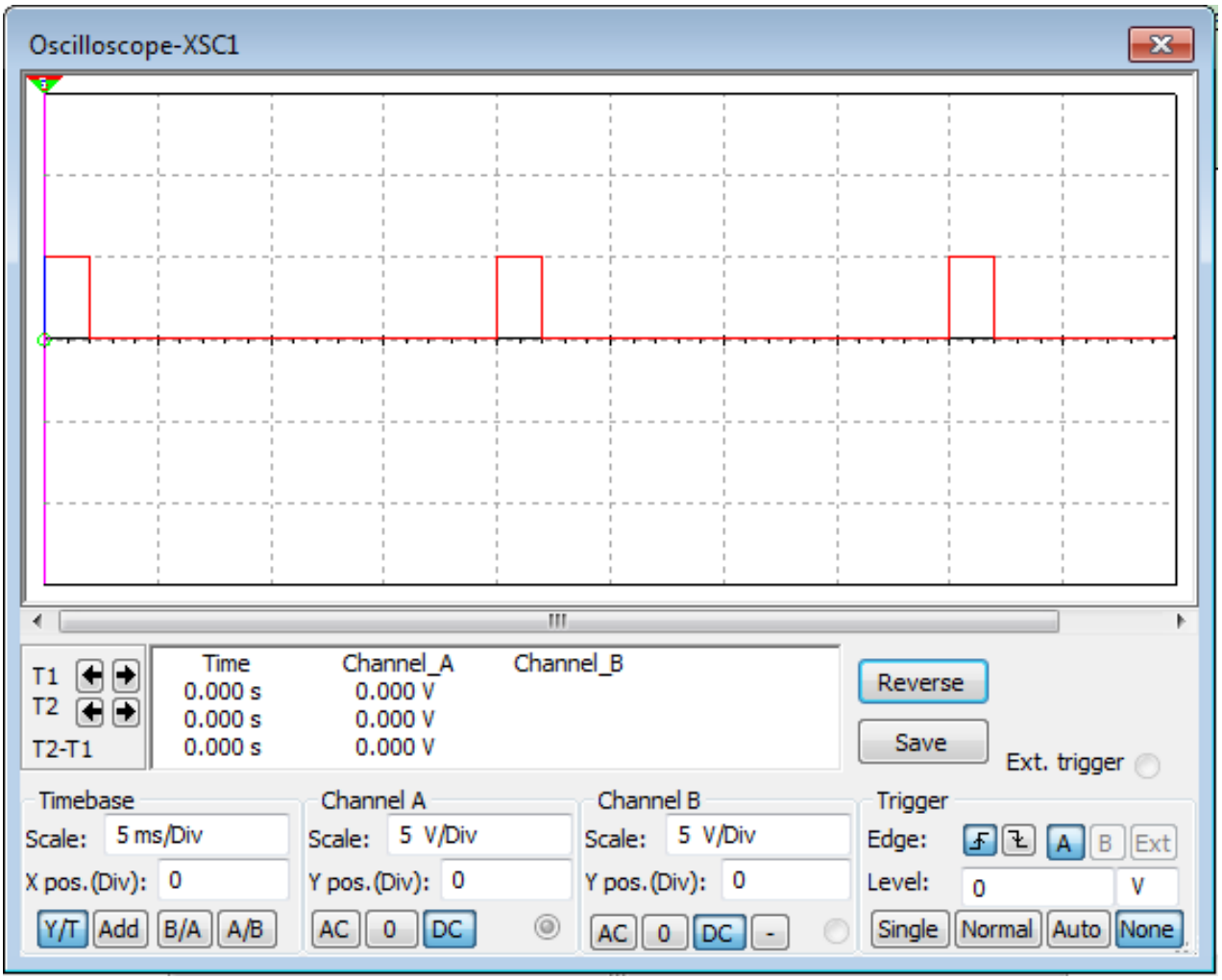


Figure 31: 1800us signal pulse

Servo Pulse 1 – 1620us:

SPI Data (after tagged bit is removed)	Pulse Length (us)	Off Time (ms)	Angle (®)
1	1620	10	150

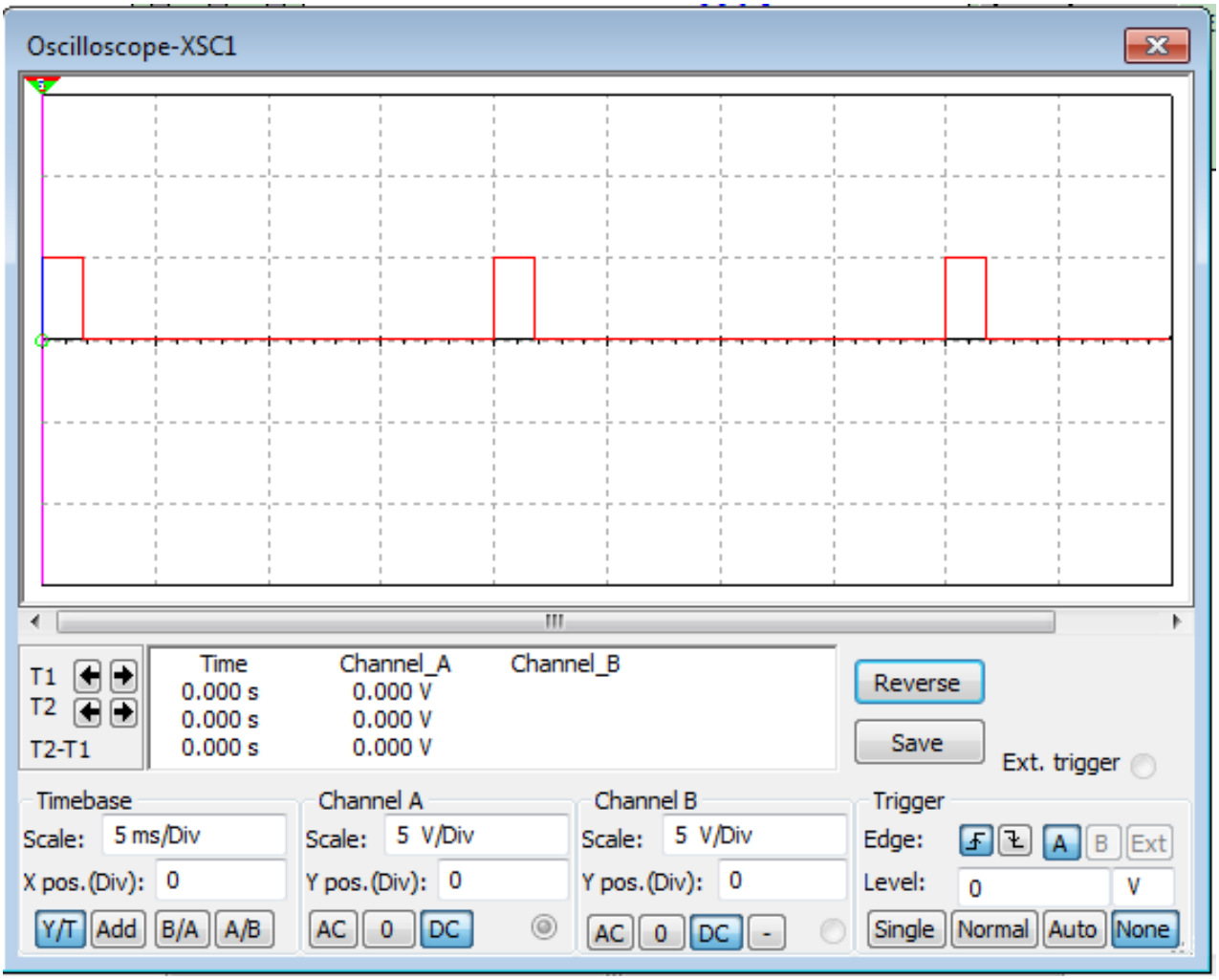


Figure 32: 1620us signal pulse

Servo Pulse 2 – 1440us:

SPI Data (after tagged bit is removed)	Pulse Length (us)	Off Time (ms)	Angle (®)
2	1440	10	135

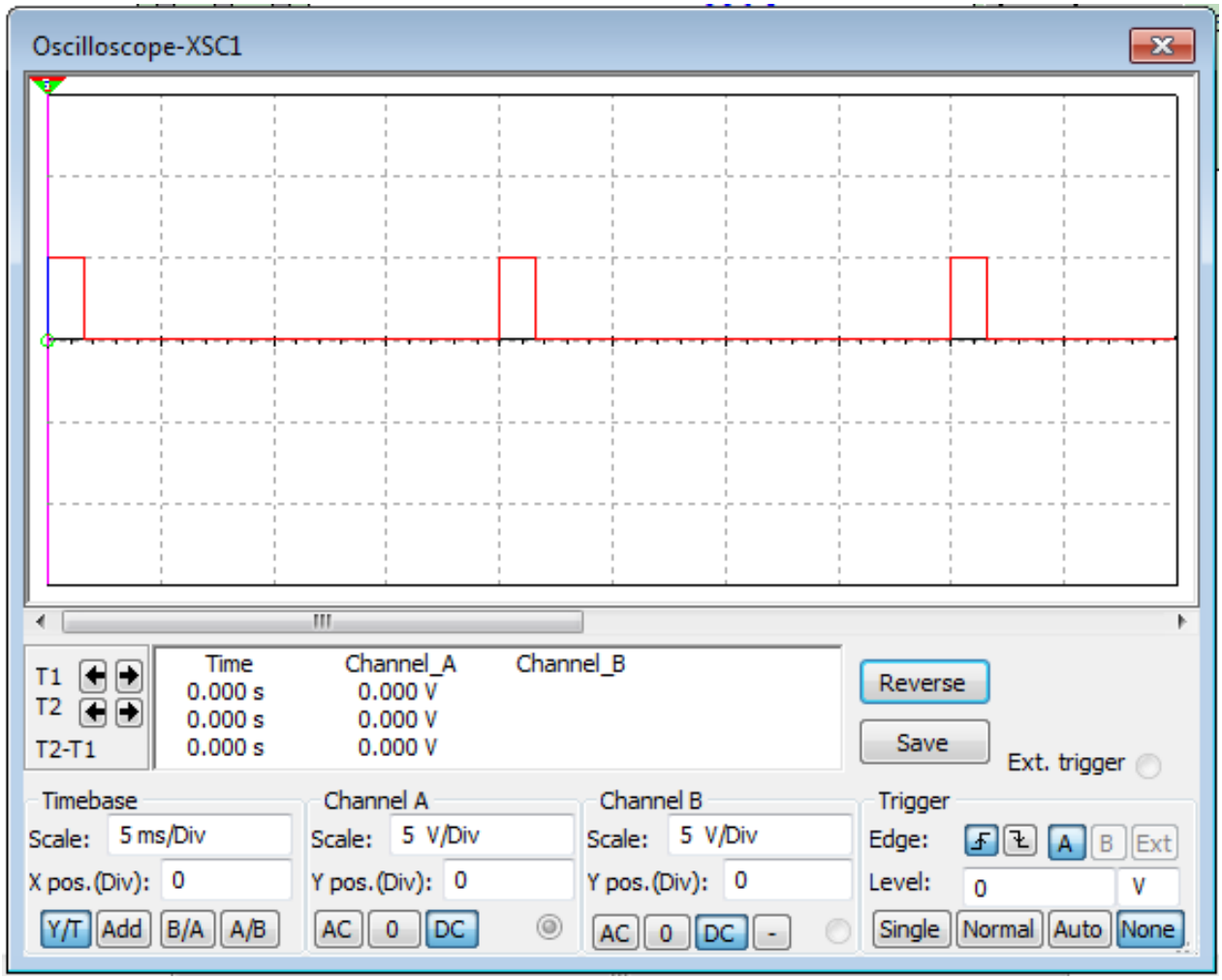


Figure 33: 1440us signal pulse

Servo Pulse 3 – 1260us:

SPI Data (after tagged bit is removed)	Pulse Length (us)	Off Time (ms)	Angle (®)
3	1260	10	120

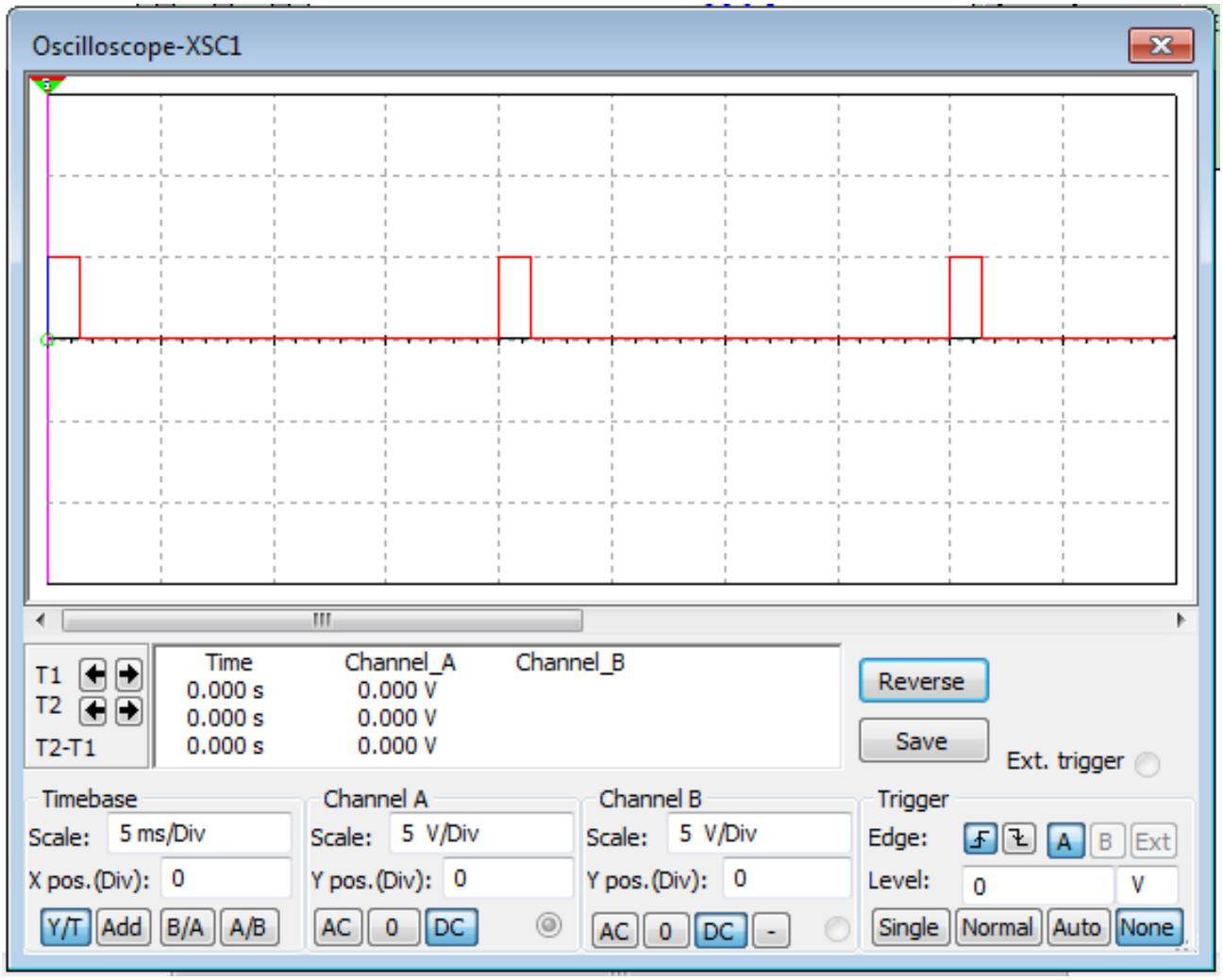


Figure 34: 1260us signal pulse

Servo Pulse 4 – 1080us:

SPI Data (after tagged bit is removed)	Pulse Length (us)	Off Time (ms)	Angle (®)
4	1080	10	105

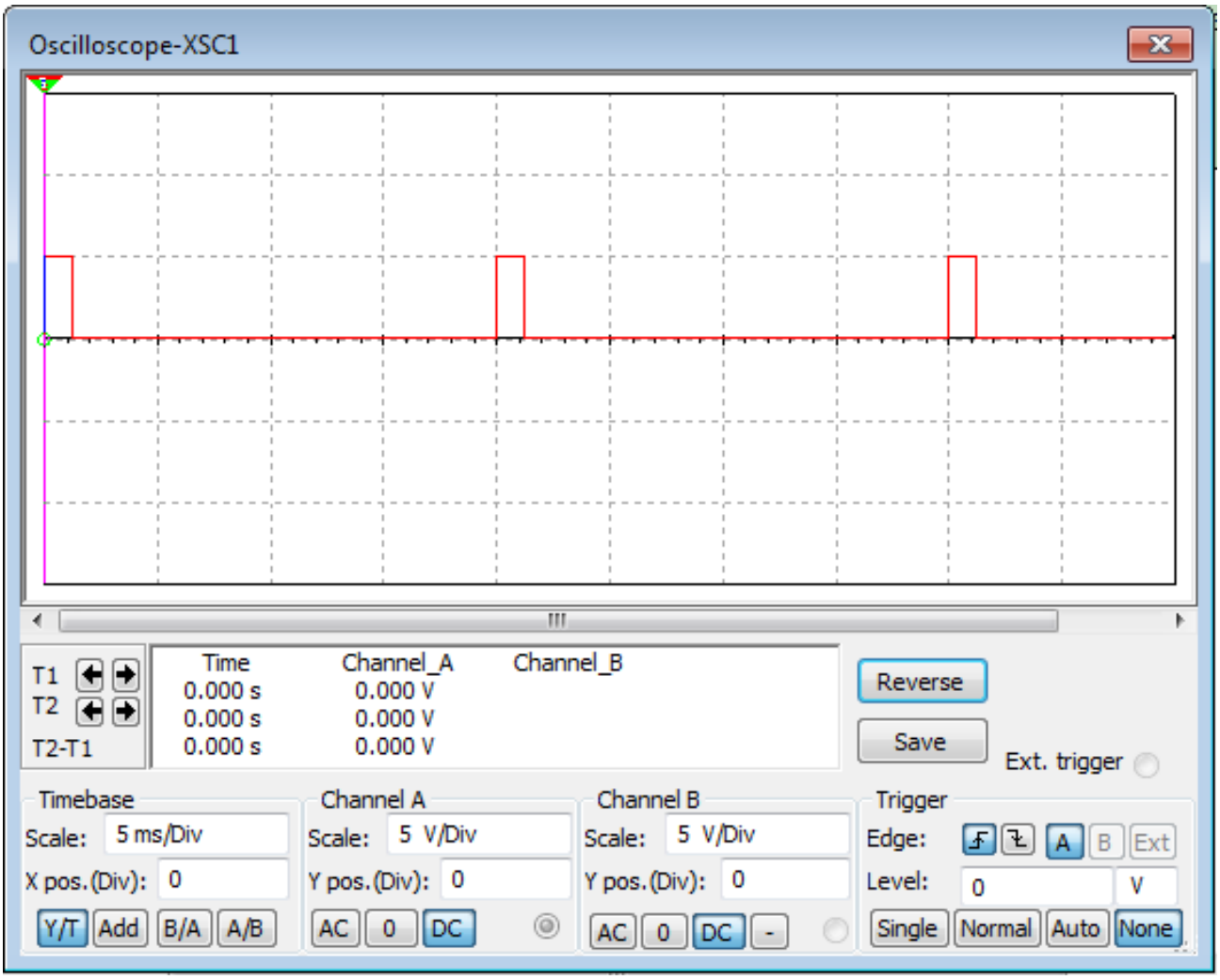


Figure 35: 1080us signal pulse

Servo Pulse 5 – 900us:

SPI Data (after tagged bit is removed)	Pulse Length (us)	Off Time (ms)	Angle (®)
5	900	10	90

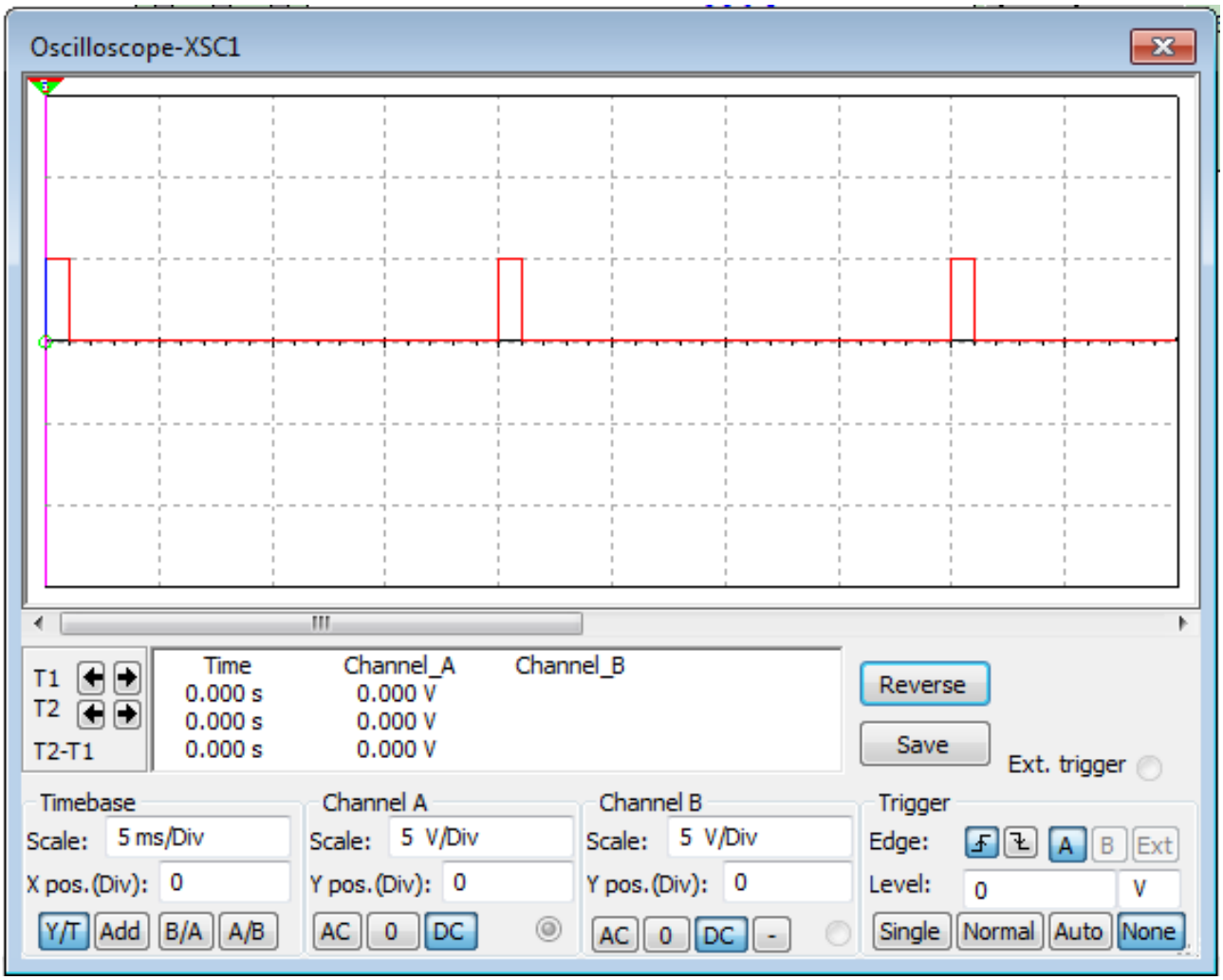


Figure 36: 900us signal pulse

Servo Pulse 6 – 720us:

SPI Data (after tagged bit is removed)	Pulse Length (us)	Off Time (ms)	Angle (®)
6	720	10	75

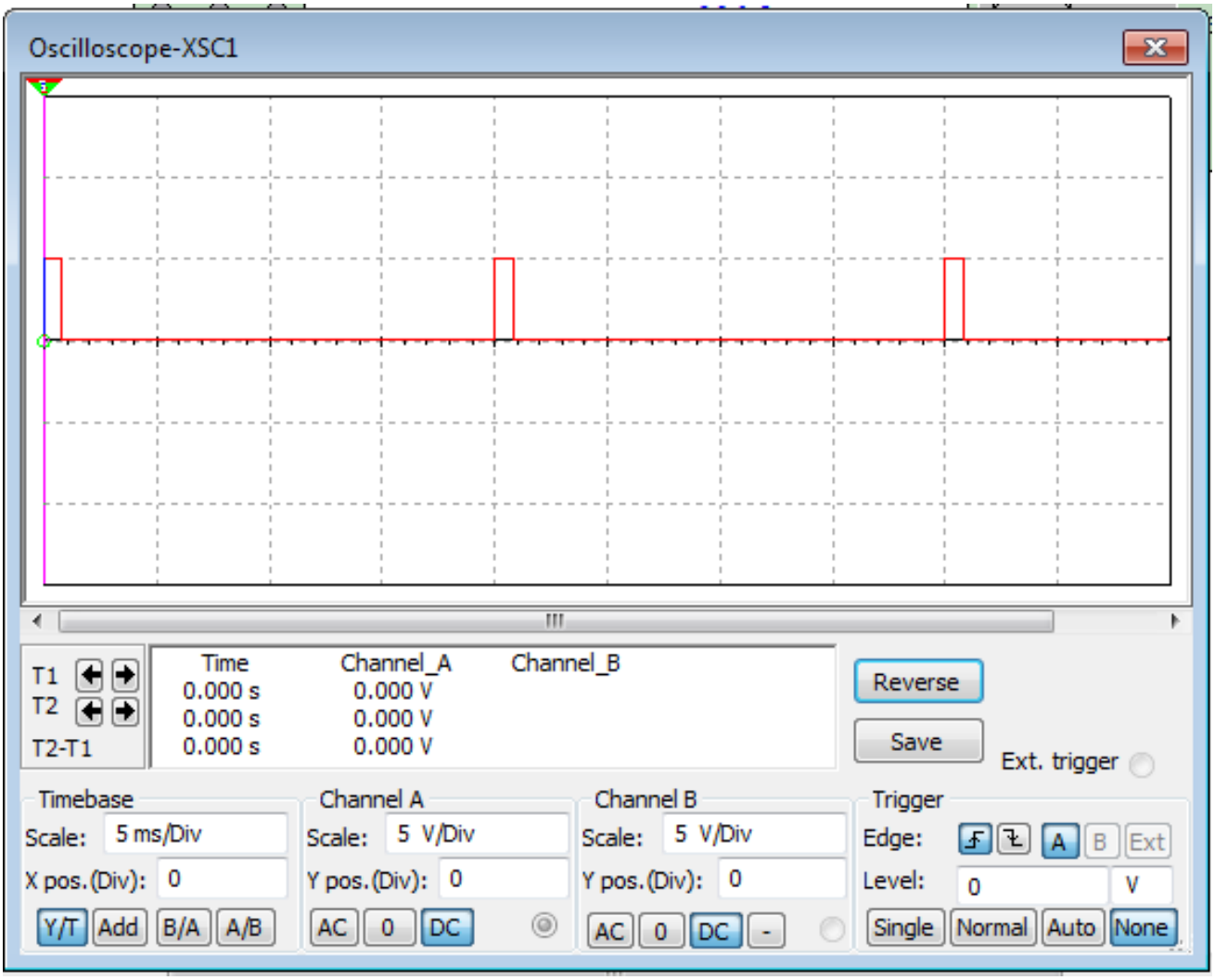


Figure 37: 720us signal pulse

Servo Pulse 7 – 540us:

SPI Data (after tagged bit is removed)	Pulse Length (us)	Off Time (ms)	Angle (®)
7	540	10	60

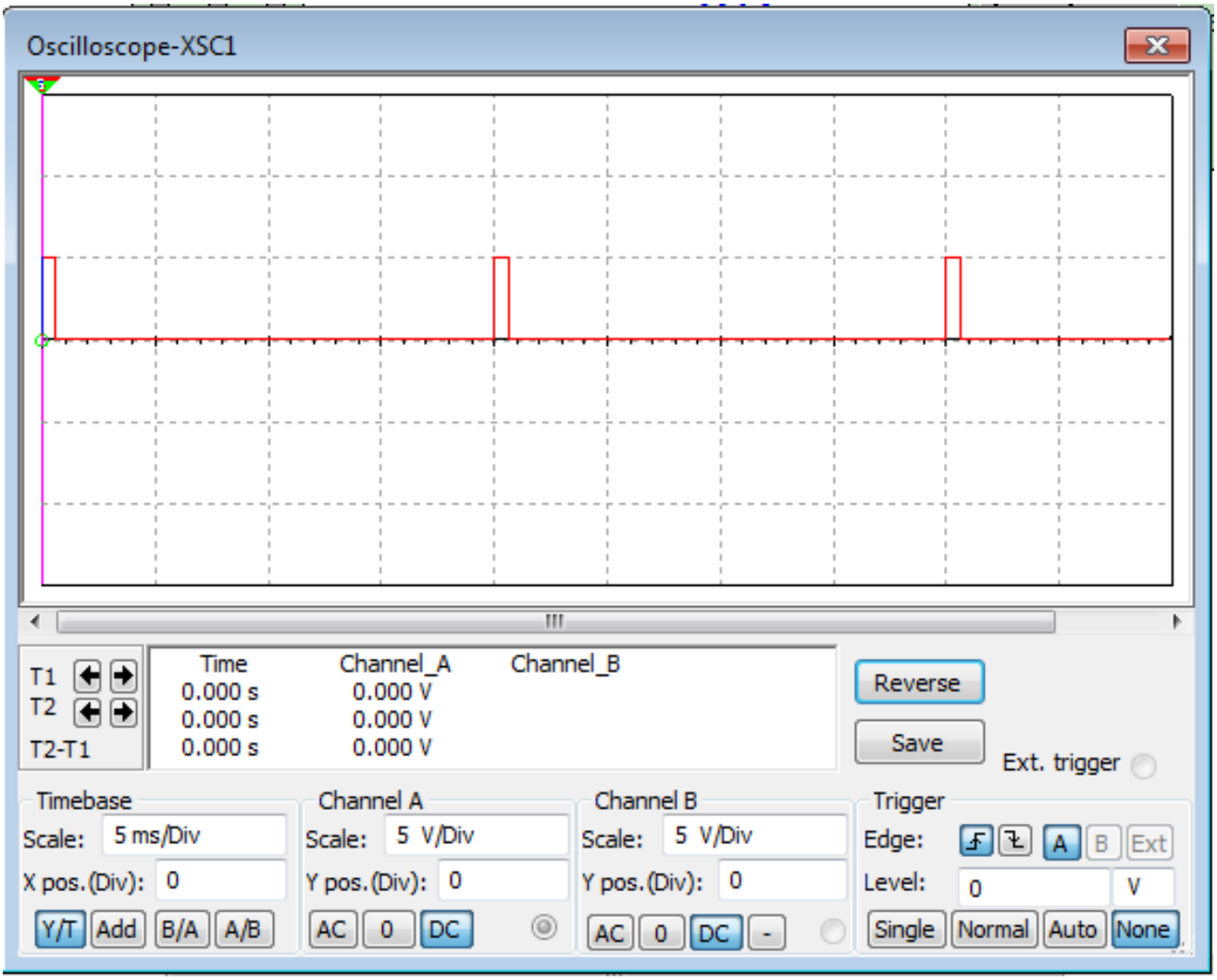


Figure 38: 540us signal pulse

Servo Pulse 8 – 360us:

SPI Data (after tagged bit is removed)	Pulse Length (us)	Off Time (ms)	Angle (®)
8	360	10	45

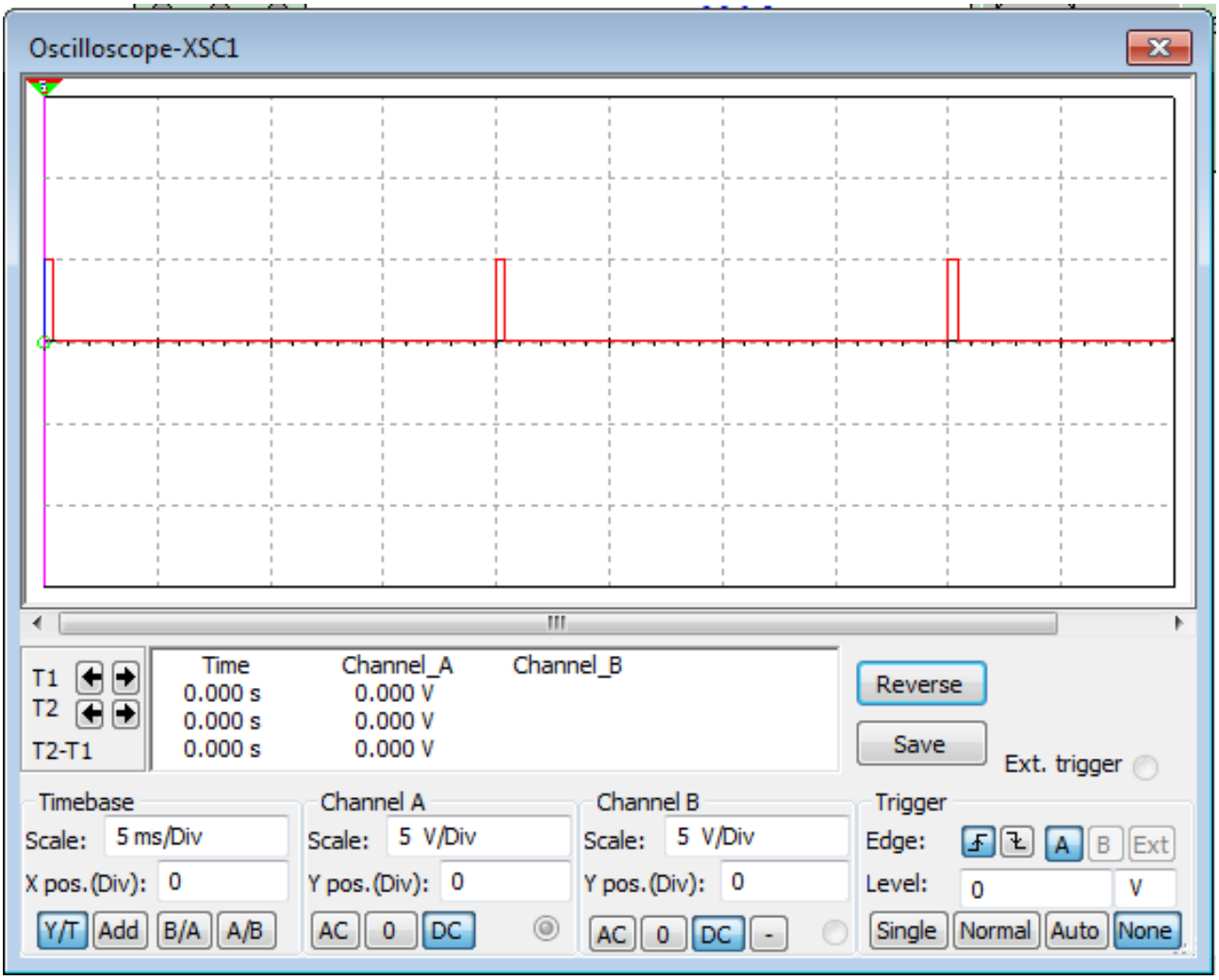


Figure 39: 360us signal pulse

Servo Pulse 9 – 180us:

SPI Data (after tagged bit is removed)	Pulse Length (us)	Off Time (ms)	Angle (®)
9	180	10	30

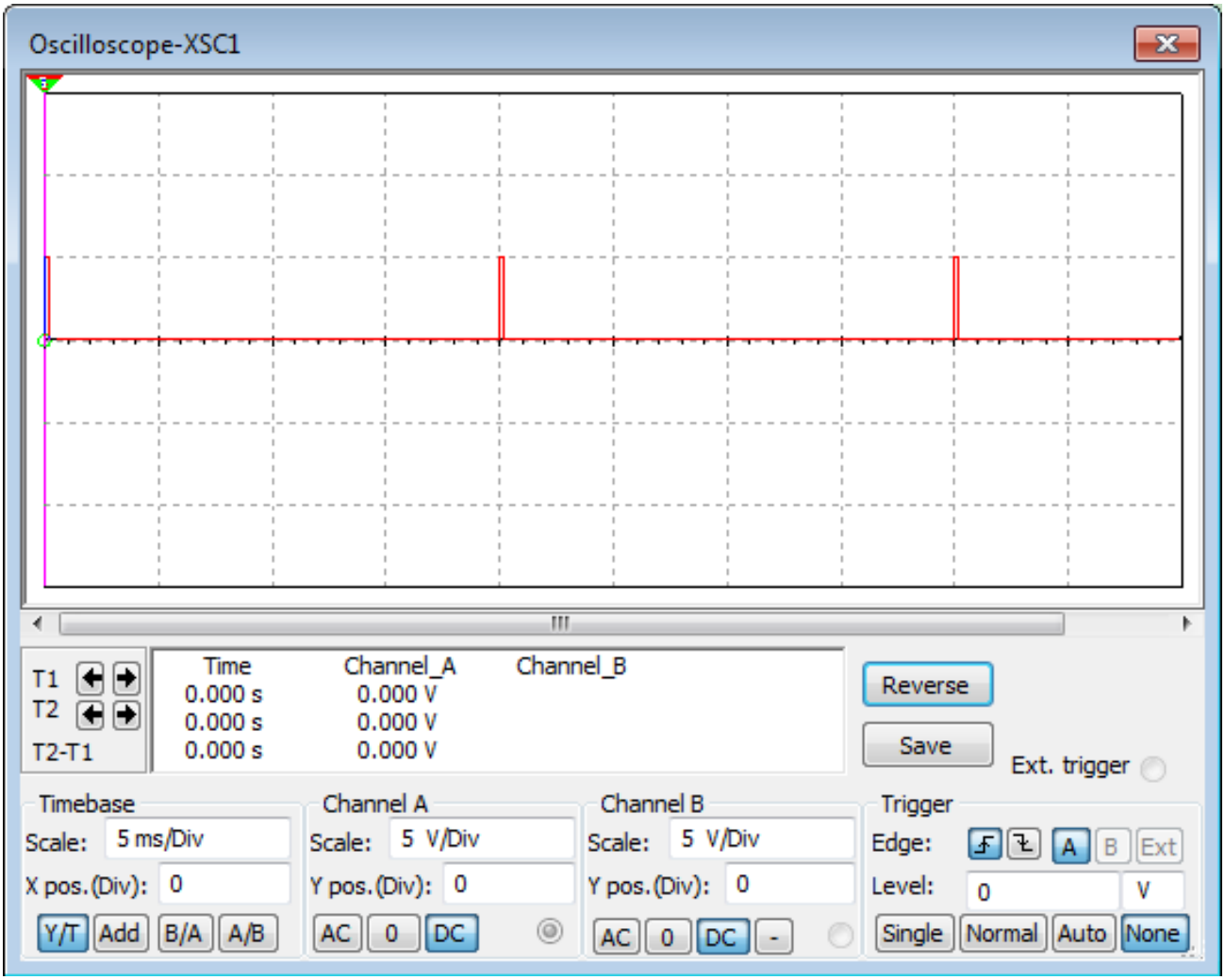


Figure 40: 180us signal pulse

Servo Pulse 10 – 0us:

SPI Data (after tagged bit is removed)	Pulse Length (us)	Off Time (ms)	Angle (®)
10	0	10	-

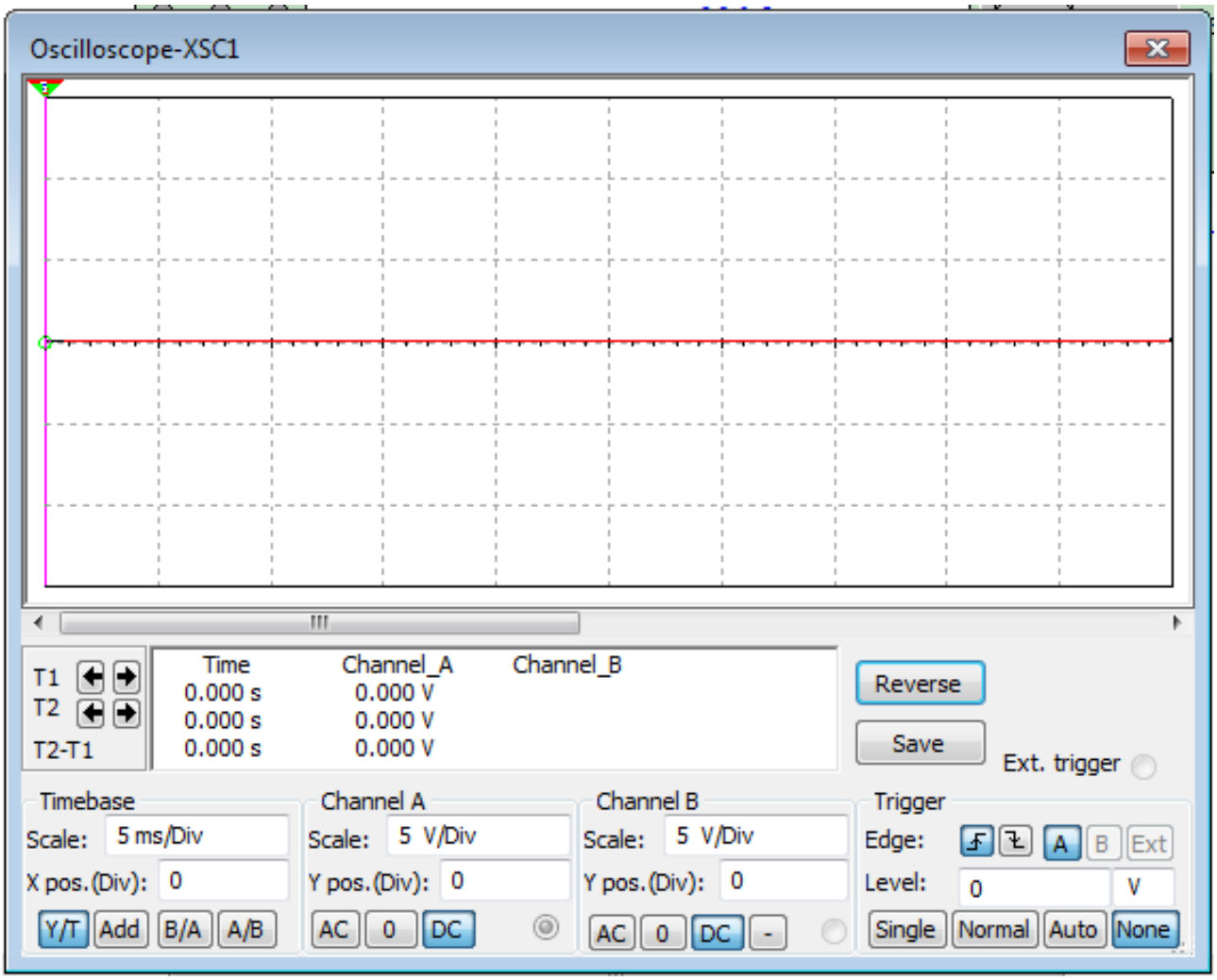


Figure 41: 0us signal pulse

3.5.7 *H-Bridge Circuit*

Component	Quantity
DC Motor	x2
1N4001 Diode	x4
0.1uf Capacitor	x1
470uf Capacitor	x1
1kohm Resistor	x2
10kohm Resistor	x6
220kohm Resistor	x2
2N3904 BJT	x2
IRF530 N-Channel MOSFET	x2
IRF9510 P-Channel MOSFET	x2

Table 6: H-Bridge Circuit component list

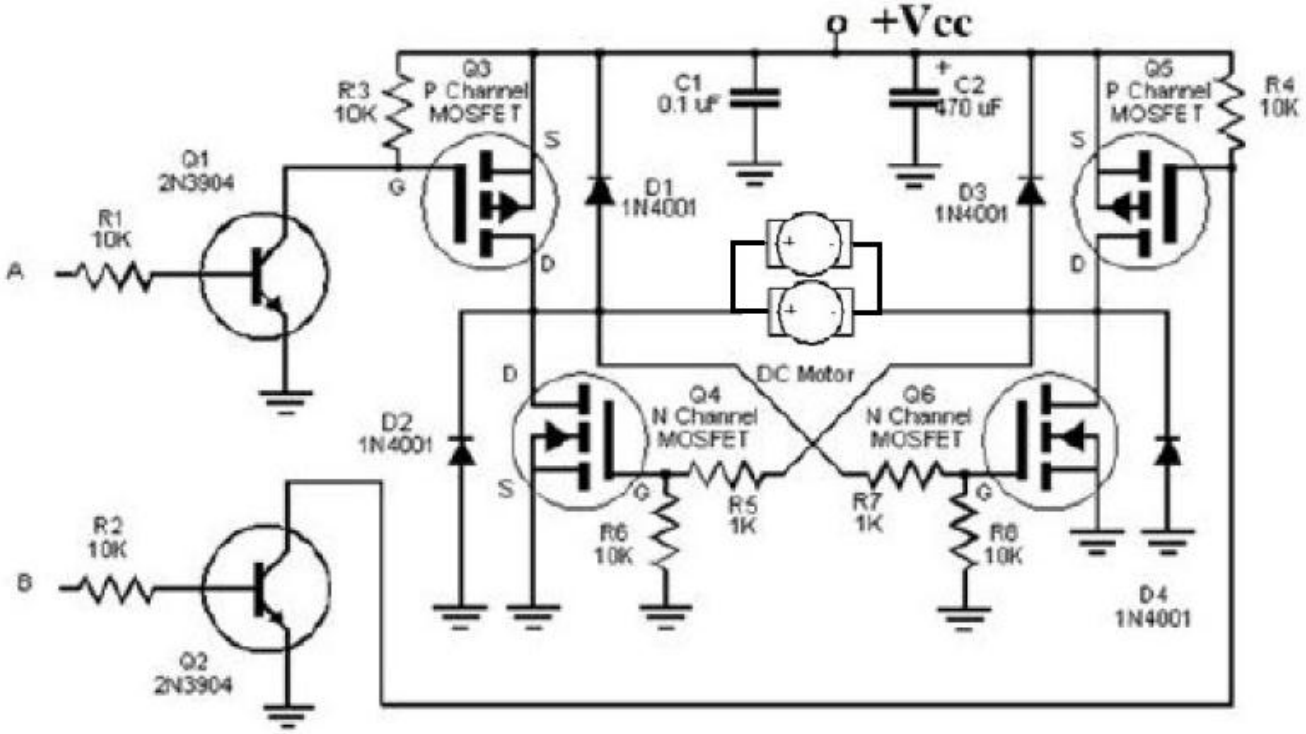


Figure 42: H-bridge circuit diagram

The two BJTs at the input terminals of the circuit are used as switches and are turned on when they sense current. The current is delivered through PWM signals from the motor control circuit. The BJTs will then control each MOSFET in the circuit. If a positive logic is sent to input BJT A, and negative logic to BJT B, then the top right MOSFET and the bottom left MOSFET will both be ON (conducting) allowing the motor to spin in a direction. If negative logic is sent to BJT A and positive logic to BJT B, the top left MOSFET, and bottom right MOSFET will turn on, allowing the motor to spin in another direction. By using P-Channel and N-channel MOSFETS, the circuit can control which direction to spin the motor based on the logical value at the BJT inputs. This is because P-Channel MOSFETS will be ON only when they have a negative voltage at their gate, while N-Channel requires a positive voltage to turn on.

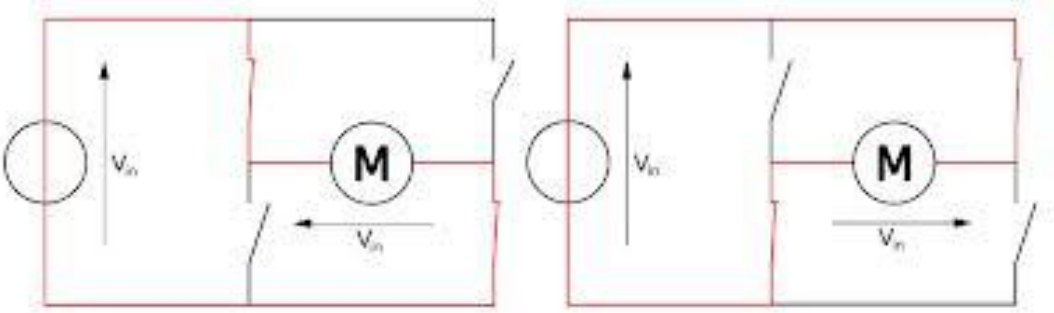


Figure 43: H-bridge conducting stages

3.5.8 *Voltage Regulator*

Figure 44: Step-down voltage regulator S7V7F5

To regulate the 7.2V we get from the battery pack down to 5V we use a step down/ step up voltage regulator. This regulator has a voltage input range of 2.7V to 11.8V, and outputs 5v with a common ground. This regulator not only protects the circuit from a brown out if the current is being drained heavily by the motors, but also allows the control board to stay on even if the battery starts to dip lower than 5V.

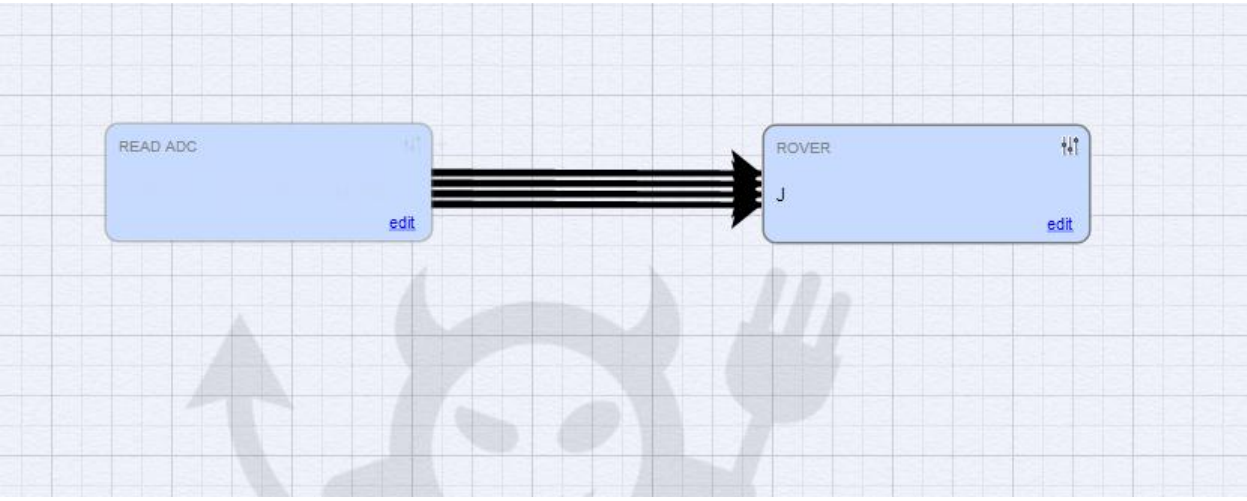
Here are the specs for the voltage regulator:

- Ability to convert both higher and lower input voltages
- Operating Voltage: 2.7V to 11.8 V
- Integrated over-temperature protection
- Default output voltage: 5 V

The **Step-Up / Step-Down Voltage Regulator S7V7F5** efficiently produces 5 V from input voltages between 2.7 and 11.8 V.

3.6 *Electric IMP planner*

We learned about Input Ports and Output Ports, which sends and receives data from the Electric IMP cloud server. But to get data to and from an imp from the server we need to use one more tool. This is called the planner, and it looks like this when there is a connection between two Electric Imps:

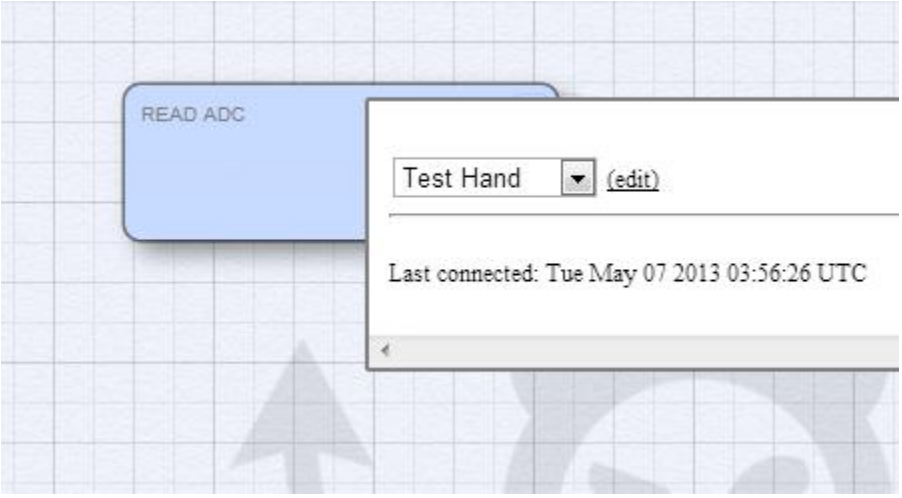


Note: To access the planner, you can go through our webpage. And use login:

Username: Gost_user@outlook.com
Password: Publicvoid

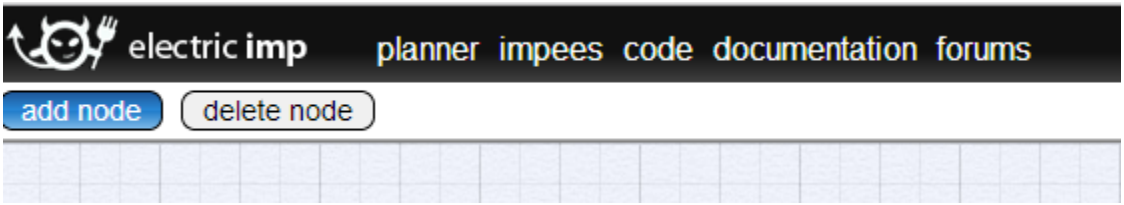
In the Electric IMP planner there are blue rectangles for each impee configured. (Impee = Imp + breakout board). The electric IMP is just a microprocessor and a Wi-Fi chip. For the server to know which code to push to it the breakout board will add a unique ID to distinguish each Electric IMP, the combination of the two is called an Impee.

In the planner you can choose the code that the electric IMP will run by clicking the small settings icon in the top right corner of the Electric IMP Impee node:



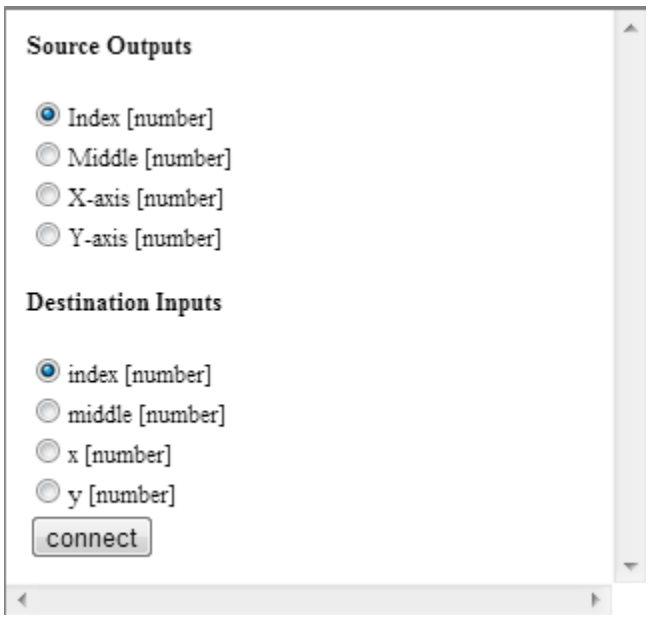
You are then given a drop down menu with a list of firmware codes you created and saved on the compiler in the website. You can choose one for the Impee to run.

To add, edit and delete code on the electric imp website you will go to the code navigation bar:



You are then given a list of all firmwares you created and have saved. Also there is a plus button where you can add a blank firmware to create a new one.

To make connections between the Impees is very simple. You click the small plus button sticking out of the node on the right side, and then point the arrow at the node you want the impee to connect to. Once this is down you are given a list of output ports from the source node, and a list of input ports from the destination node. You just click on the two variables you want to tie together to send data to each other:



Once they are connected it should look like this:

Colour	Speed	Meaning
Green	Pulse	Successfully received configuration via Blinkup. You will see a brief green pulse from the imp just as the BlinkUp flashing stops
Red	Fast	Failed to receive configuration via Blinkup
Orange	Slow	No WiFi settings, run Blinkup
Red	Slow	Attempting to join WiFi. If the imp stays in this state, check the wifi SSID you entered in the BlinkUp app is correct
Red, Orange, Off	Slow	Getting IP address (DHCP). If the imp stays in this state, generally your wifi password was typed incorrectly
Orange, Red, Off	Slow	Got IP address, connecting to server. If the imp stays in this state, often the issue is that the port the imp is using is being blocked by a firewall
Green	Very Slow	Connected to the cloud (turns off after 60 seconds)
Red	Fast	Connection lost, attempting to reconnect
Short green, off, Long green, off	Slow	Verifying firmware update
Green	On (not flashing)	Firmware upgrade in progress, please do not unplug your imp. Usually lasts less than a minute
None		Normal operation

3.7 Troubleshooting

3.7.1 *Sensory Glove*

This section will help you troubleshoot your Sensory Glove of the G.O.S.T system if it is not functioning properly. This section a continuation of the Simple troubleshooting guide that is located in section 2-3.

Step 1:

The first thing and easiest thing to check is the Electric IMP's debug LED. This LED can help pin point the problem. Here is a list of LED codes and what they mean:

Color	Speed	Meaning
Green	Pulse	Successfully received configuration via Blinkup. You will see a brief green pulse from the imp just as the BlinkUp flashing stops
Red	Fast	Failed to receive configuration via Blinkup
Orange	Slow	No WiFi settings, run Blinkup
Red	Slow	Attempting to join WiFi. If the imp stays in this state, check the wifi SSID you entered in the BlinkUp app is correct
Red, Orange, Off	Slow	Getting IP address (DHCP). If the imp stays in this state, generally your wifi password was typed incorrectly
Orange, Red, Off	Slow	Got IP address, connecting to server. If the imp stays in this state, often the issue is that the port the imp is using is being blocked by a firewall
Green	Very Slow	Connected to the cloud (turns off after 60 seconds)
Red	Fast	Connection lost, attempting to reconnect
Short green, off, Long green, off	Slow	Verifying firmware update
Green	On (not flashing)	Firmware upgrade in progress, please do not unplug your imp. Usually lasts less than a minute
None		Normal operation

If the Electric IMP led is blinking green, then the IMP is connected to the internet, and you can move on to the next step. If the LED is showing a different issue, then resolve the corresponding problem by following the table above (network, imp, etc...).

Step 2:

Check to see if values are changing in the Electric IMP planner:

```
5/7/2013 1:42:12 PM: Thumb >> ( 0 ) Index >> ( 32 ) Middle >> ( 32 ) Ring >> ( 7 ) X-Axis >> ( 10 ) Y-Axis >> ( 10 ) Battery >> ( 3.29 V) Signal >> ( -40 dBm)
5/7/2013 1:42:13 PM: Thumb >> ( 0 ) Index >> ( 32 ) Middle >> ( 32 ) Ring >> ( 7 ) X-Axis >> ( 10 ) Y-Axis >> ( 10 ) Battery >> ( 3.29 V) Signal >> ( -40 dBm)
5/7/2013 1:42:13 PM: Thumb >> ( 0 ) Index >> ( 32 ) Middle >> ( 32 ) Ring >> ( 7 ) X-Axis >> ( 10 ) Y-Axis >> ( 10 ) Battery >> ( 3.30 V) Signal >> ( -39 dBm)
5/7/2013 1:42:13 PM: Thumb >> ( 0 ) Index >> ( 32 ) Middle >> ( 32 ) Ring >> ( 7 ) X-Axis >> ( 10 ) Y-Axis >> ( 10 ) Battery >> ( 3.30 V) Signal >> ( -39 dBm)
5/7/2013 1:42:13 PM: Thumb >> ( 0 ) Index >> ( 32 ) Middle >> ( 32 ) Ring >> ( 7 ) X-Axis >> ( 10 ) Y-Axis >> ( 10 ) Battery >> ( 3.29 V) Signal >> ( -39 dBm)
5/7/2013 1:42:13 PM: Thumb >> ( 0 ) Index >> ( 32 ) Middle >> ( 32 ) Ring >> ( 7 ) X-Axis >> ( 10 ) Y-Axis >> ( 10 ) Battery >> ( 3.28 V) Signal >> ( -40 dBm)
5/7/2013 1:42:14 PM: Thumb >> ( 0 ) Index >> ( 32 ) Middle >> ( 32 ) Ring >> ( 7 ) X-Axis >> ( 10 ) Y-Axis >> ( 10 ) Battery >> ( 3.29 V) Signal >> ( -39 dBm)
5/7/2013 1:42:14 PM: Thumb >> ( 0 ) Index >> ( 32 ) Middle >> ( 32 ) Ring >> ( 7 ) X-Axis >> ( 10 ) Y-Axis >> ( 10 ) Battery >> ( 3.29 V) Signal >> ( -39 dBm)
```

These values change based on the position of the hand. The values will be changing every 20ms. If the values are not changing then the electric imp on the glove is the culprit of the problem. Also if the following message is displayed in the planner log:

```
5/7/2013 2:52:16 PM: Power state: online=>offline
```

Then the glove is offline.

If the values are changing, and the imp is online, then the Sensory Glove is working correctly, and you can move on to section 3.7.2.

Step 3:

The next step is to check if the time is changing. If it is but the values are not changing, then it may be a problem with the wiring, or one of the electrical components is not working correctly.

Next step would be to isolate the component that is not changing. You can do this by verifying out of all the sensors which value is not changing. This will isolate the actual component. Once isolate you may check the wiring. You can test the wiring by using a DMM. Measure the voltage at the corresponding pin (PIN out is located in section 3.4.4). Keep the glove centred on a flat surface, and the fingers should not be bent, you should then be receiving ~1.65V at each pin. If you are not receiving this voltage at one of the pins, then the wiring is to blame. If you are receiving a different voltage than this, than it may either be the flex sensor, accelerometer, or one of the resistors in the voltage divider that is to blame. You can either ship the Sensory Glove to us, or changing the suspected component yourself.

If you are receiving the right voltages, you can start bending the fingers one by one, and measuring the voltage at the same time. The voltages should increase when you bend the finger. To test the Accelerometer, at center, the voltage should be 1.65V. Just like an X – Y plane, the voltage will decrease when tilt to the negative side, and increase while tilting to the positive side.

If all the voltages are changing correctly then you can move on to the next section.

Step 4:

If:

- The voltages are changing
- The time is not changing for the logging messages
- The values are not changing in the logging messages

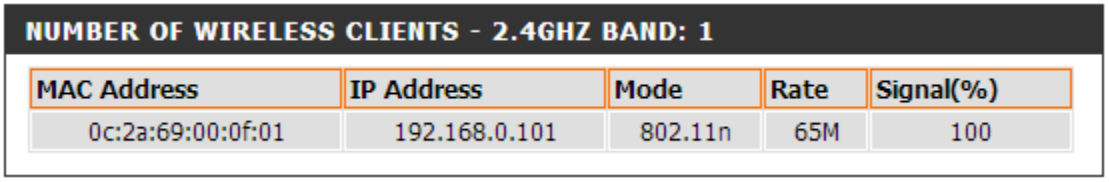
There is an error in the code. This needs to be reported to us. Screenshots and the method of which caused the problem can be emailed to us, at the information at our contact page.

3.7.2 *Smart Tank IMP*

This section will help troubleshoot strictly the Electric IMP on the rover. This part should only be done if the Sensory Glove is functioning properly. If you did not follow the Sensory Glove troubleshooting guide (section 3.7.1) then you should do that first before continuing with this guide. This section a continuation of the Simple troubleshooting guide that is located in section 2-3.

Step 1:

Check the LED codes in section 3.7.1. If all LED is green, and there is no issue with LED codes, then the IMP is connected to the network and you can move to the next step. If the LED is not green, then the most likely cause is batteries or network issues. Make sure the batteries are not drained, and the IMP is connected to the network. You can check the connected devices by looking at your routers wireless client list:



NUMBER OF WIRELESS CLIENTS - 2.4GHZ BAND: 1

MAC Address	IP Address	Mode	Rate	Signal(%)
0c:2a:69:00:0f:01	192.168.0.101	802.11n	65M	100

You can check the mac addresses of the IMP by looking at the Electric IMP card. The mac is listed on the back of the card. If the imp is connected to the network, and the Smart Tank is still not functioning properly then you can continue to the next part.

Step 2:

The next step would be to look at the Electric IMP planner for details of how the IMP is working. The log will detail what the IMP is sending out SPI (it will show an explanation not the actual data it is sending out).

```
5/7/2013 12:50:56 PM: Right Track Forward
5/7/2013 12:50:57 PM: Left Track Reverse
5/7/2013 12:50:57 PM: Right Track Reverse
5/7/2013 12:50:57 PM: Pan -> Center
5/7/2013 12:50:57 PM: Left Track Stop
5/7/2013 12:50:57 PM: Right Track Stop
5/7/2013 12:50:58 PM: Left Track Stop
5/7/2013 12:50:58 PM: Right Track Stop
5/7/2013 12:50:58 PM: Left Track Reverse
5/7/2013 12:50:58 PM: Right Track Reverse
5/7/2013 12:50:58 PM: Left Track Reverse
5/7/2013 12:50:58 PM: Right Track Reverse
5/7/2013 12:50:58 PM: Pan -> Left
5/7/2013 12:50:59 PM: Pan -> Center
5/7/2013 12:50:59 PM: Pan -> Left
5/7/2013 12:50:59 PM: Pan -> Center
5/7/2013 12:51:00 PM: Left Track Reverse
5/7/2013 12:51:00 PM: Left Track Reverse
5/7/2013 12:51:00 PM: Left Track Reverse
5/7/2013 12:51:01 PM: Left Track Reverse
5/7/2013 12:51:01 PM: Pan -> Right
```

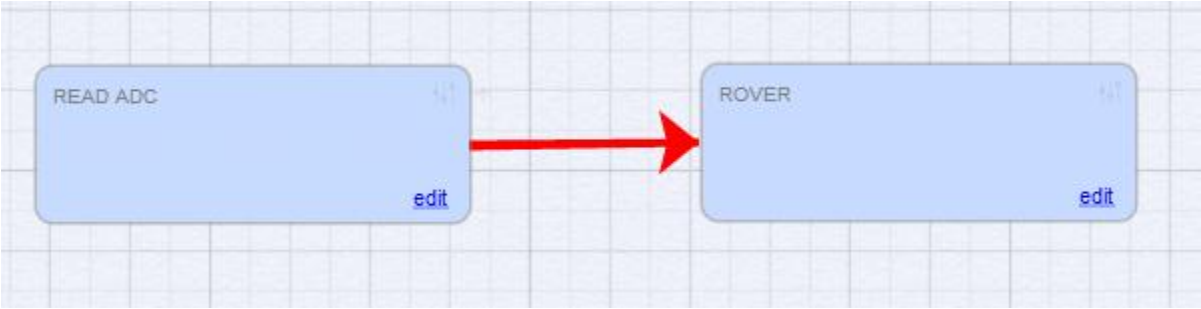
If these messages are being shown when the glove is being used, then the IMP is calling the handler functions. If the Handler functions are being called then the data is being sent out the SPI. The next step would be to test the data that is being sent out the SPI, by going to section 3.7.3 *Received SPI Data*.

If the there are no logging messages move to the next step.

Step 3:

If there are no logging messages than one issue could be the virtual connection between the two IMPS on the planners. To make things simpler, you can just remove all the connections between the two nodes and reconnect them.

You can delete a virtual connection by clicking on the arrow until it turns red.

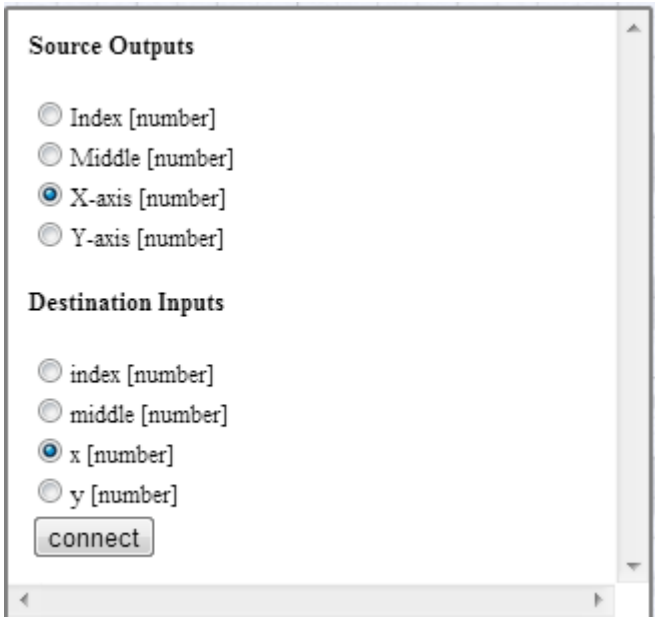


Then click the delete button.

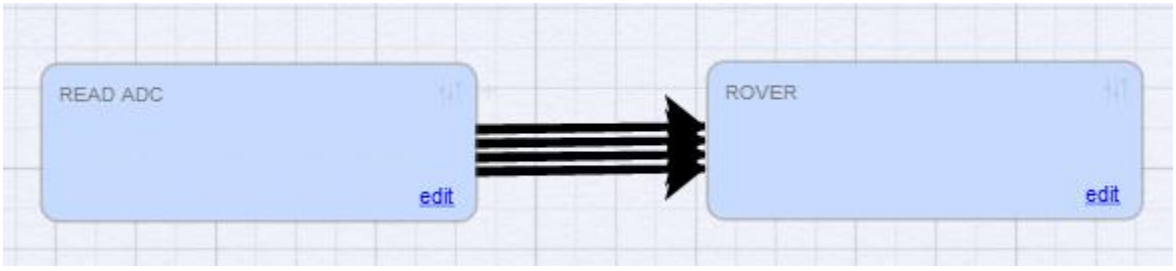
Cleared it should look like this:

Then reconnect each virtual connection. There are 4 connections, and to make things simpler, each variable is named the same on both sides, so the connections are just matching.

Example:



Once all 4 connections are made there should be 4 arrows:



Step 4:

If those steps did not solve the issue, then there is an error in the code. This needs to be reported to us by contacting us at our contact page. Please include screenshots (if needed) and how the problem came about.

3.7.3 *Received SPI Data*

The first measurement should be taken at the MOSI pin and the SCK pin on the ATMEGA8L. The SCK should be measured at 3 kHz and the MOSI pin should be measured for the following data frames:

	<----------Tagged Bits ---------->				<---------- Data ---------->			
Clock (3kHz)	1	0	1	0	1	0	1	0
Data 0	x	x	x	x	0	0	0	0
Data 1	x	x	x	x	0	0	0	1
Data 2	x	x	x	x	0	0	1	0
Data 3	x	x	x	x	0	0	1	1
Data 4	x	x	x	x	0	1	0	0
Data 5	x	x	x	x	0	1	0	1
Data 6	x	x	x	x	0	1	1	0
Data 7	x	x	x	x	0	1	1	1
Data 8	x	x	x	x	1	0	0	0
Data 9	x	x	x	x	1	0	0	1
Data 10	x	x	x	x	1	0	1	0

Figure 45: SPI data table

The four MSB bits should be looked at in more detail to make sure that the corresponding tagged bit performs the right function:

	<----------Tagged Bits ---------->				<---------- Data ---------->			
Clock (3kHz)	1	0	1	0	1	0	1	0
Tagged Bit: 128	1	0	0	0	x	x	x	x
Tagged Bit: 64	0	1	0	0	x	x	x	x
Tagged Bit: 32	0	0	1	0	x	x	x	x

Figure 46: Tagged data

For the DC Motor Controller, the data should be checked that it's tagged with 128 when the LEDs are enabled, 64 when the motors are forwarding, and 32 when reversing. For the Servomotor Controller, the data should be measured and checked that it's tagged with 64 when it is panning (x-axis), and when it's tagged with 32, it is tilting (y-axis).

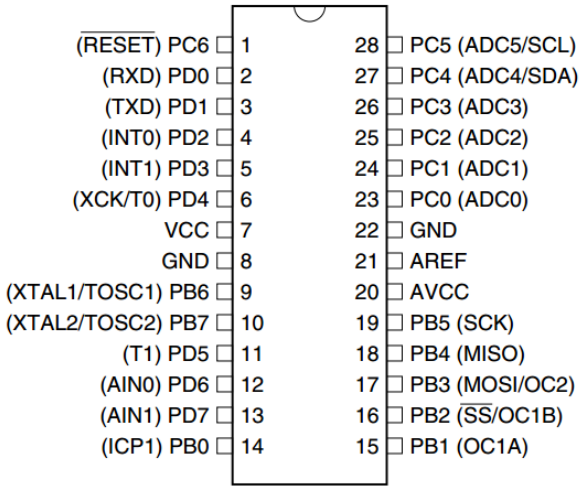


Figure 47: Atmega8L pinout

All three ATMEGA8Ls should be measured at the SS pin to make sure that it is receiving a low pulse when that ATMEGA is being used. If the pin is grounded, the chip does not receive the SPI data.

3.7.4 *Output Signals of the ATMEGA8Ls*

PC0 of PORTC should be measured first and checked for the appropriate PWM output signals. The signals should be measured at a frequency of approximately 35 kHz. The following table shows the relation between the data being received through SPI and the output PWM signals (refer to step 1 to measure the SPI data):

SPI Data (after tagged bit is removed)	Duty Cycle	Frequency
0	0%	35khz
1	55%	35khz
2	60%	35khz
3	65%	35khz
4	70%	35khz
5	75%	35khz
6	80%	35khz
7	85%	35khz
8	90%	35khz
9	95%	35khz
10	100%	35khz

Table 7: PWM output vs. SPI received SPI data

PC1, PC2, and PC3 of PORTC should also be measured to determine if the tagged data is performing the right operation. PC1 should follow the PWM signal of PC0 when the data is tagged 128 (refer to step 1). PC2 should follow the PWM signal of PC0 when the data is tagged with 64, and PC3 should follow the signal of PC0 when the data is tagged with 32.

Tagged Bit	PC0 (PWN Signal	PC1 (LED Control)	PC2 (Motor control 1	PC3 (Motor Control 2	Function
32	1	0	0	1	Forward
64	1	0	1	0	Reverse
128	1	1	0	0	Enable LED

Table 8: Tagged bit vs. output control signals

Appendix

Appendix I - *Diagrams*

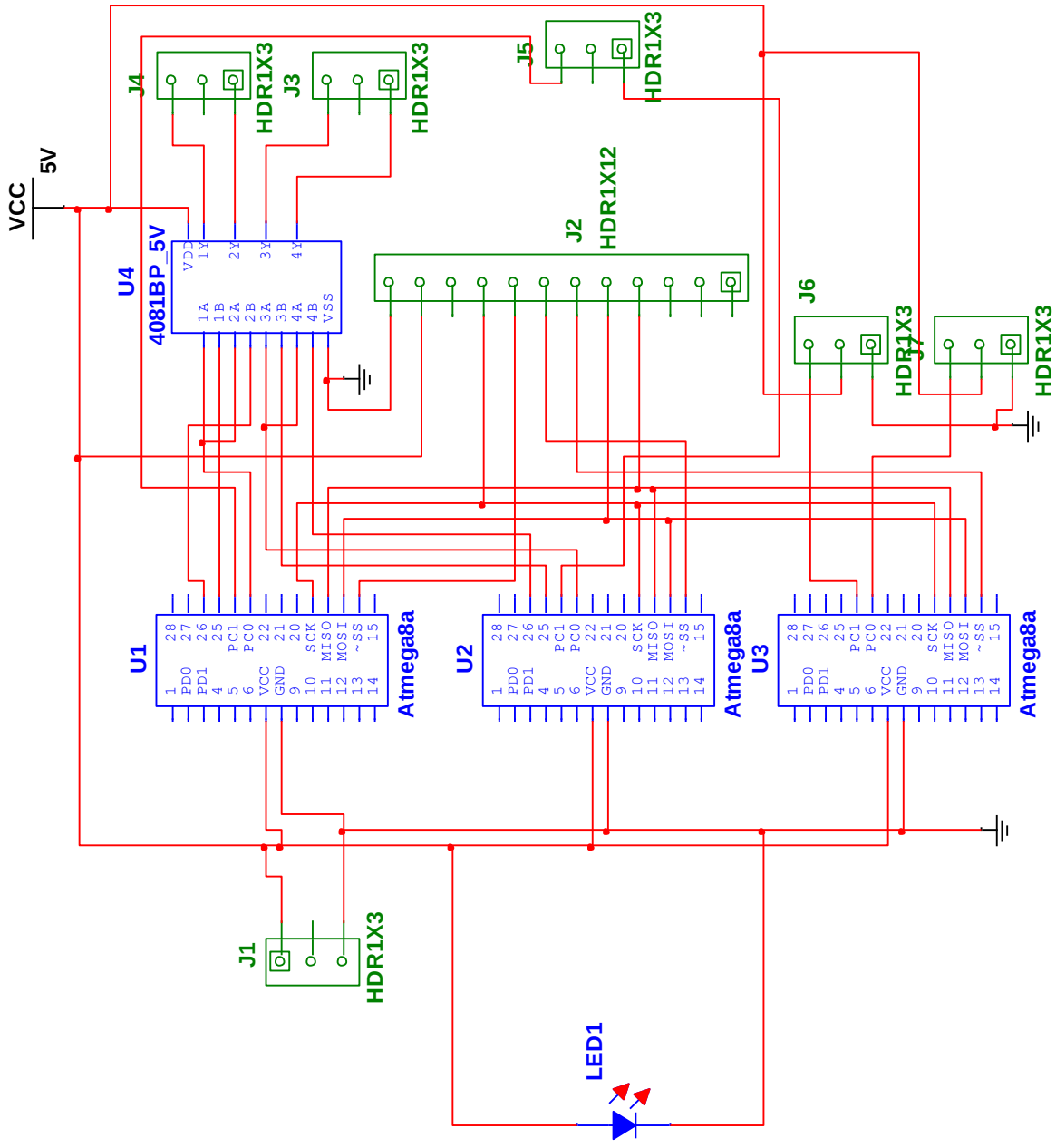


Figure 48: Motor controller circuit wiring diagram

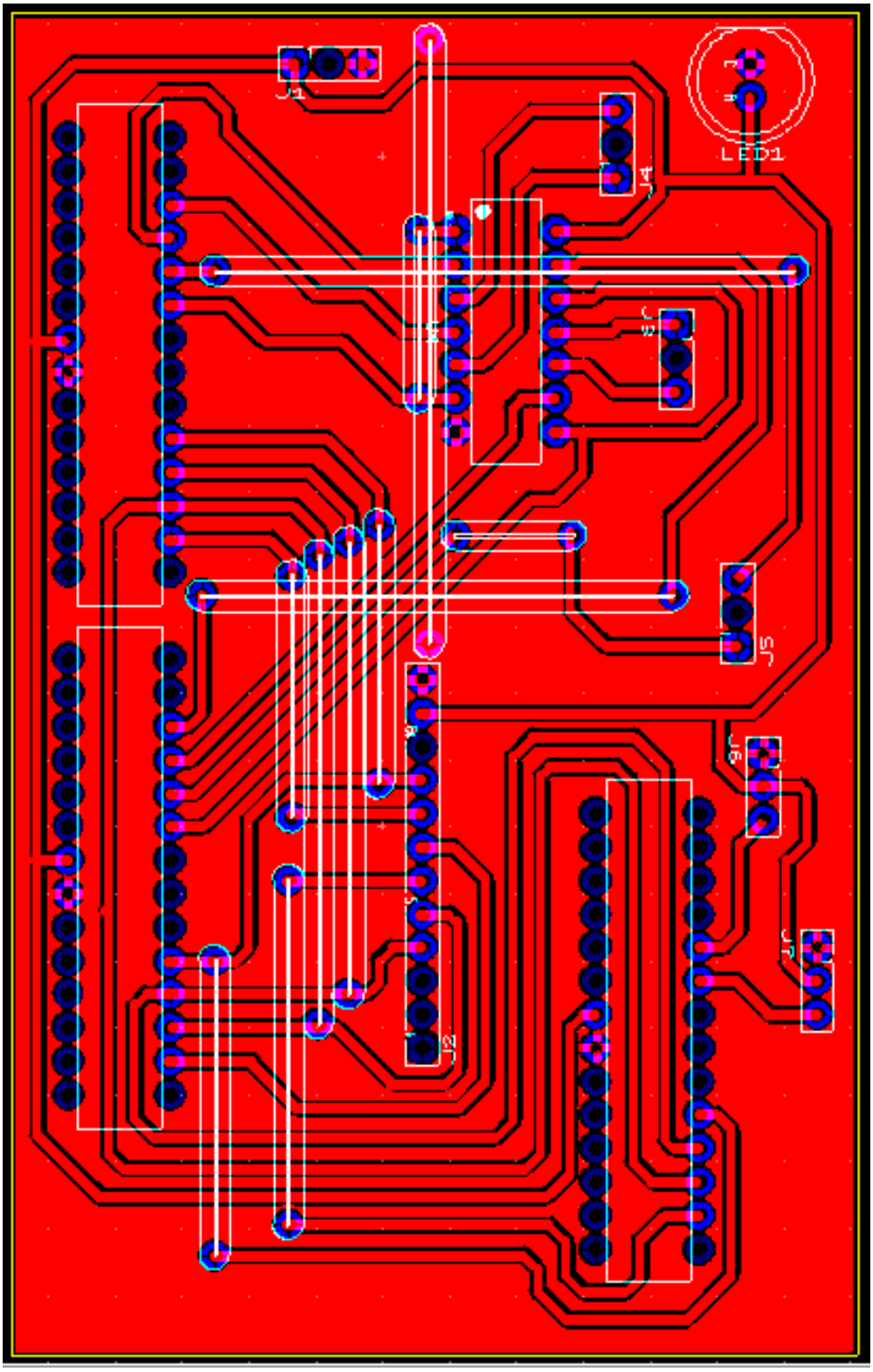


Figure 49: Motor controller circuit PCB design

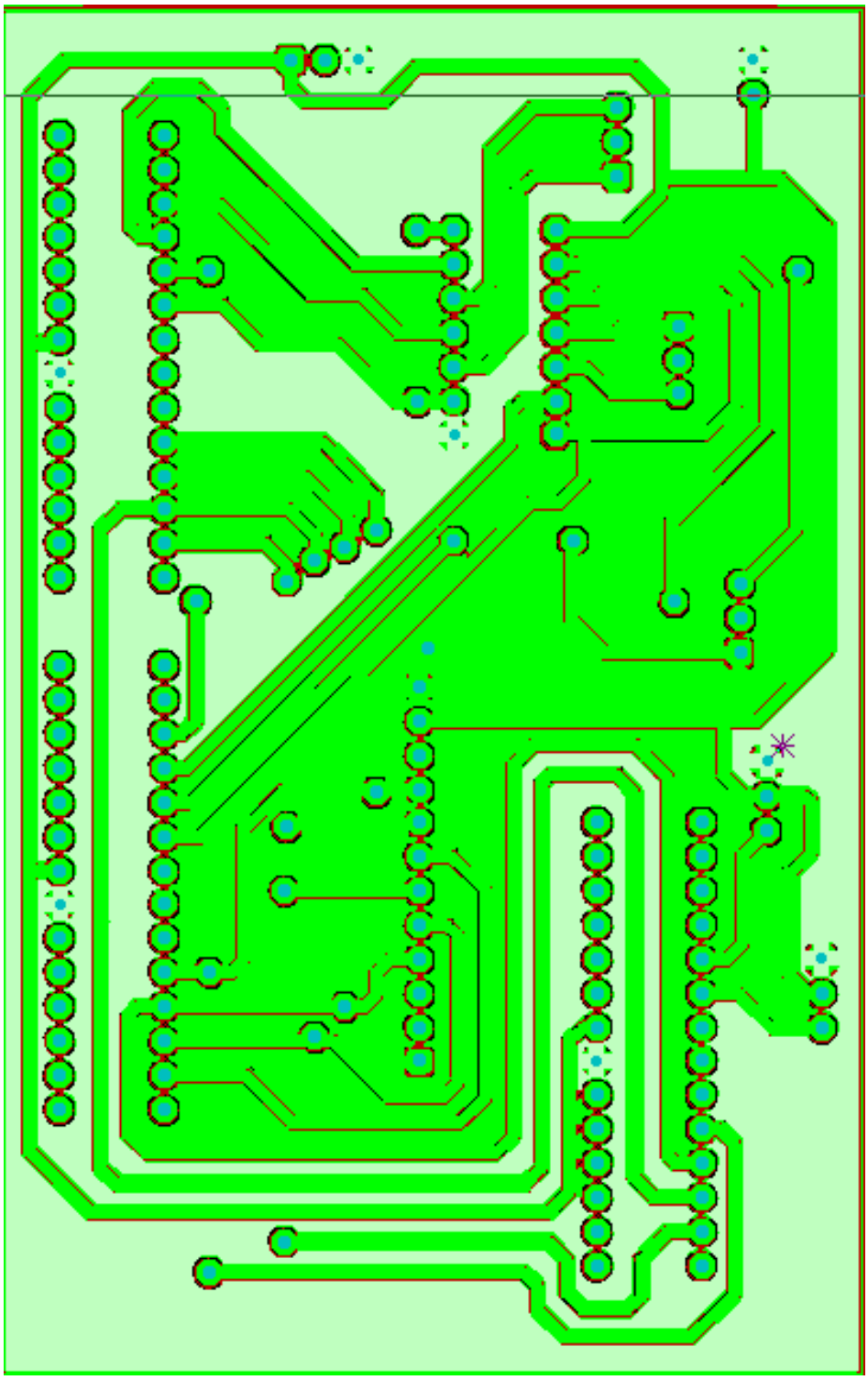


Figure 50: Circuit Cam design for motor controller circuit

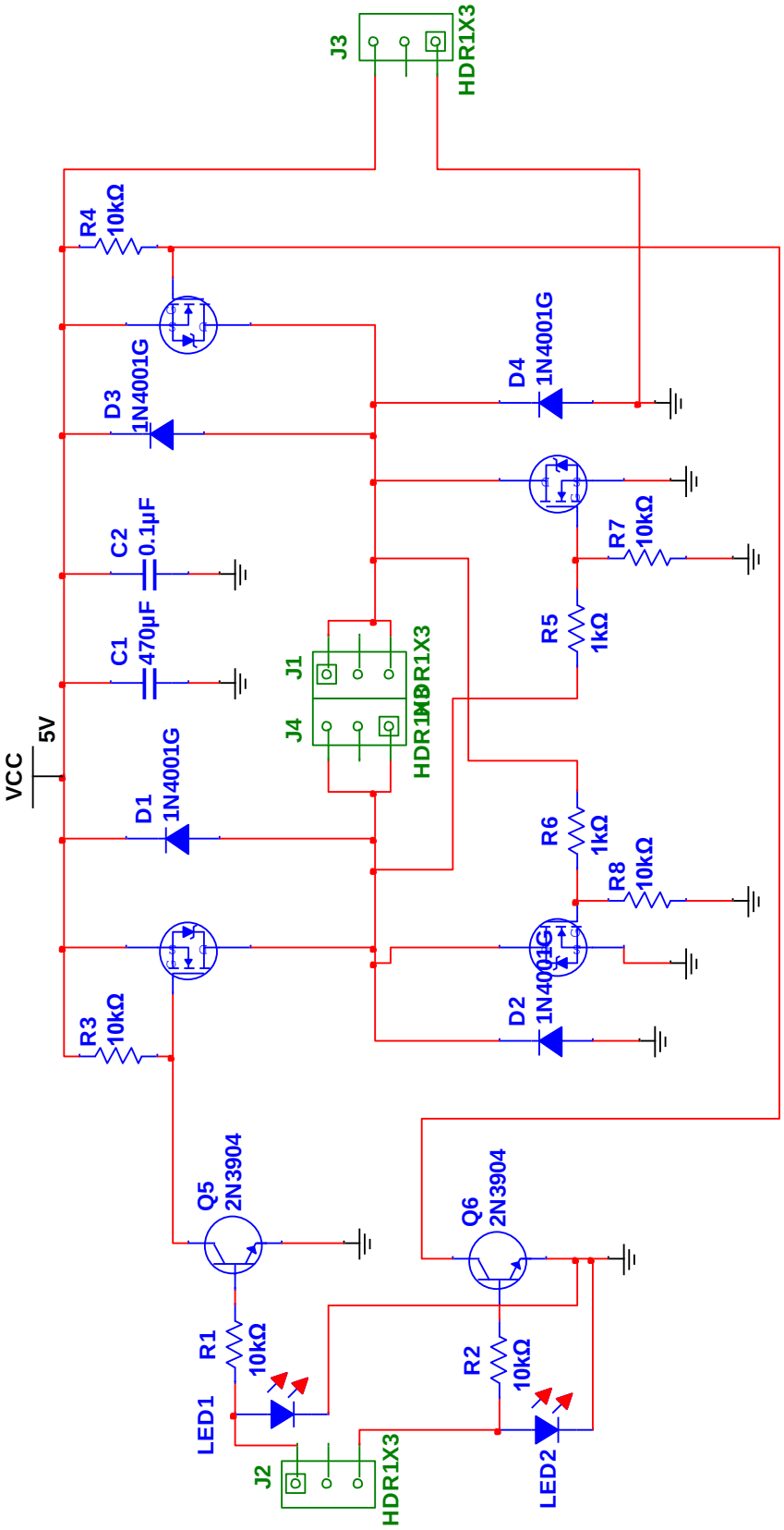


Figure 51: H-bridge wiring diagram

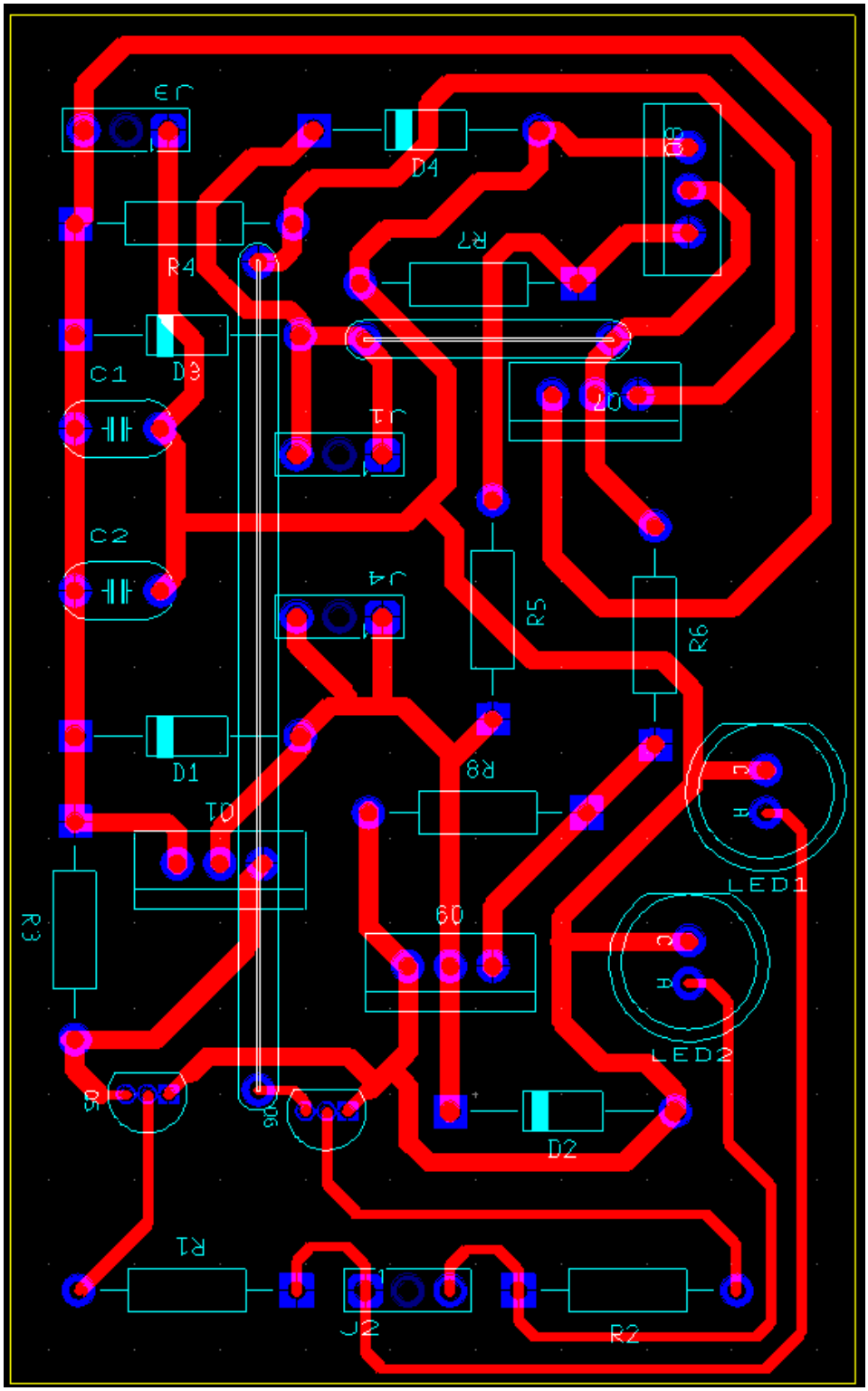


Figure 52: H-bridge PCB design

Appendix II - *Code*

```
/*************************************************
 * Title :      PWM Motor Controller             *
 *************************************************
 * Author :     Matthew Masi & Michael Scalera   *
 *************************************************
 * Chip :       ATMEGA8L                         *
 *************************************************
 * Created :    12/02/13                         *
 *************************************************
 * Modified :   24/04/13                         *
 *************************************************/


/*************************************************
 * The PWM Motor Controller program controls two
 * left side DC motors and two right side DC
 * Motors with PWM output signals, and controls
 * output LEDs. The output data for the PWM
 * signals and LEDs are controlled by the input
 * data received through SPI communication and is
 * triggered as an internal interrupt
 */

/* === PREPROCESSOR DIRECTIVES === */

#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 8000000UL
#include <util/delay.h>

/* === VOLATILE VARIABLE DECLARATION === */

volatile char data = 'a';       //SPI reception variable
volatile int on;                //on time of PWM signal
volatile int off;               //off time of PWM signal
volatile int var;               //port output value
volatile int ipt;               //check bit tags
volatile int led;               //enable pin for LEDs


int main (void)
{
    DDRC = 0xff;                //enable PORTC as output
    DDRB = (1<<DDB4);           //set MISO output, all others input
    SPCR = (1<<SPE)|(1<<SPIE);  //enable SPI & SPI interrupt
    SREG = 0x80;                //enable global interrupts

    var = 0;
    on = 10;
    off = 10;
    led = 2;
    int x = 0;
```



```
 sei();                        //set global interrupt enable

    while(1)
    {

    /* === PWM ON TIME === */

        if(on != 10)
        {
            PORTC = var + led;  // LED and direction select
            for(x=0;x!=on;x++)  //on time duration
            {
                x=x;            //bogus instruction
            }
        }

    /* === PWM OFF TIME === */

        if(on != 20)
        {
            PORTC = 0 + led;    //LED and direction select
            for(x=0;x!=off;x++) //off time duration
            {
                x=x;            //bogus instruction
            }

        }
    }
}

/************************************************************
 * Function : SPI Interrupt
 * -------------------------
 * This interrupt function is active whenever data is
 * recieved by the ATMEGA through SPI. Recieved data is valued
 * between 0 and 10 which coresponds to the duty cycle of a
 * pwm signal. All data is tagged with a bit to either enable
 * the motor in reverse mode (64), enable the motor in forward
 * mode (32), or turn on the LEDs (128).
 */

ISR(SPI_STC_vect)
{
    data = SPDR;              //read SPI data (char)
    ipt = data;

    /* === ENABLING LEDs === */

    if(ipt >= 128)
    {
        if(led == 2)          //disable LEDs
            led = 0;
        else
            led = 2;          //enable LEDs
    }
```



```
 /* === REVERSE MOTOR === */

    else if(ipt >= 64)
    {
        on = ipt - 64;      //removes tagged bit
        var = 5;            //PWM and reverse enabled

        on = on + 10;       //PWM on time
        off = 20 - on;      //PWM off time
    }

    /* === FORWARD MOTOR === */

    else
    {
        on = ipt - 32;      //removes tagged bit
        var = 9;            //PWM and forward enabled

        on = on + 10;       //PWM on time
        off = 20 - on;      //PWM off time
    }


}
```



```
/***********************************************
 * Title :      Servomotor Controller          *
 ***********************************************
 * Author :     Matthew Masi & Michael Scalera *
 ***********************************************
 * Chip :       ATMEGA8L                       *
 ***********************************************
 * Created :    12/02/13                       *
 ***********************************************
 * Modified :   24/04/13                       *
 ***********************************************/


/***********************************************
 * The Servomotor Controller program controls a
 * servomotor with different length pulses. The
 * duration of each pulse will put the servomotor
 * at a different angle. These pulses are affected
 * by the input data received through SPI
 * communication and are triggered as an internal
 * interrupt.
 */

/* === PREPROCESSOR DIRECTIVES === */

#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 8000000UL
#include <util/delay.h>

/* === VOLATILE VARIABLE DECLARATION === */

volatile char data = 'a';
volatile int on;
volatile int y;
volatile int x;
volatile int z;

int main (void)
{
    DDRC = 0xff;                    //enable PORTC as output
    DDRB = (1<<DDB4);               //set MISO output, all others input
    SPCR = (1<<SPE) | (1<<SPIE);    //enable SPI & SPI interrupt
    SREG = 0x80;                    //enable global interrupts

    on = 0;
    y = 0;
    x = 0;
    z = 0;

    sei();                          //set global interrupt enable

    while(1)
    {

        /* === X-AXIS ON-TIME PULSE LENGTH === */
```



```
        PORTC = 1;                  //pin connected to x-axis servo
        _delay_us(550);             //550us buffer between servomotors

        for(x=0;x<=y;x++)
        {
            _delay_us(180);         //pulse on-time for x-axis
        }

        /* === Y-AXIS ON-TIME PULSE LENGTH === */

        PORTC = 2;                  //pin connected to y-axis servo
        _delay_us(550);             //550us buffer between servomotors

        for(x=0;x<=z;x++)
        {
            _delay_us(180);         //pulse on-time for y-axis
        }

        /* === X & Y-AXIS OFF-TIME PULSE LENGTH === */

        PORTC = 0;
        _delay_ms(10);              //10ms pulse off-time
    }
}

/************************************************************
 * Function : SPI Interrupt
 * -------------------------
 * This interrupt function is active whenever data is
 * recieved by the ATMEGA through SPI. Recieved data is valued
 * between 0 and 10 which coresponds to the duration of a
 * signal pulse. All data is tagged with a bit to either enable
 * the y-axis motor (32), or enable the x-axis motor (64).
 */

ISR(SPI_STC_vect)
{
    data = SPDR;                    //read SPI data (char)
    on = data;

    /* === X-AXIS ROTATION OF SERVOMOTOR === */

    if(on >= 64)
    {
        on = on -64;                //removes bit tag
        z = 10 - on;                //on-time pulse length
    }

    /* === Y-AXIS ROTATION OF SERVOMOTOR === */

    else if(on >=32 && on <= 63)
    {
        on = on -32;                //removes bit tag
        y = 10 - on;                //on-time pulse length

    }
}
```


Appendix III – *Bill of Materials*

Part	**Cost**
Step-Up / Step-Down Voltage Regulator (1x)	$5.10
Bi-Directional Flexible Bend Sensor (4x)	$41.00 ($10.25 each)
Dagu Mini Pan and Tilt Kit (1x)	$14.95
Dagu Rover 5 4WD Tracked Chassis (1x)	$68.96
+/- 1.5 / 6g Triple Axis Accelerometer (1x)	$14.95
Electric Imp Breakout (2x)	$25.90 ($12.95 each)
Electric Imp (2x)	$59.90 (29.95 each)
Gloves (material)	$2.00
2x AA battery holder	$0.59
3x AA battery holder	$0.59
4x AA battery holder	$0.59
30ft 24 AWG wire	$3.59
3 meters 18AWG flexible wire (red)	$3.87
3 meters 18AWG flexible wire (black)	$3.87
8x AA 1.2V NiMH batteries	$14.99
48x AA 1.5V batteries	$4.99
Atmega8L (3x)	$12.69($4.23 each)
2 port blue headers (12x)	$11.88($0.99 each)
2x Copper Boards	$12.00 ($6.00 each)
All electronic components (resistors, capacitors, etc)	Approx. $5.00
Total cost	**$307.41**

Appendix IV – *Cost Analysis*

Week	Michael (hours)	Matthew(hours)
1	7	7
2	7	7
3	7	7
4	7	7
5	7	7
6	7	7
7	7	7
8	7	7
9	7	7
10	7	15
11	7	7
12	10	7
13	10	12
14	12	12
15	18	18
Total Hours	**127**	**134**
Total paid ($34/hr)	**$4,318.00**	**$4,556.00**

Appendix V –System Diagram