Appendix A: Hardware and Software Figures

Figure A.1: Haptic Array Hardware Schematic
Figure A.2: Software Flowchart
Figure A.3: Power Circuit PCB
Figure A.4: Power Circuit Diagram
Appendix B: Simulation and Verification Figures

Figure B.1: Motor circuit without flyback diode.

Figure B.2: Plot of the voltage across the supply and the drain, without flyback diode. The motor loses its external bias at 100us.
Figure B.3: Motor circuit with flyback diode.

Figure B.4: Plot of the voltage across the supply and the drain, with flyback diode. The motor loses its external bias at 100us.
Figure B.5: Motor circuit with flyback diode and series resistance.

Figure B.6: Plot of the voltage across the supply and the drain, with flyback diode and series resistance. The motor loses its external bias at 100us.
<table>
<thead>
<tr>
<th>Module</th>
<th>Requirement</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Supply</td>
<td>1. A 9V battery must supply the Arduino. It must supply 200mA and not drop below 7V.</td>
<td>1. a. A 45 Ohm resistor will be connected from power to ground for the 9V power supply.</td>
</tr>
<tr>
<td></td>
<td>2. A 4.8V portable power supply for the TLC5940 and motor array must supply up to 2A and maintain a voltage between 4.7V and 5V.</td>
<td>b. The power supply will be tested with a digital multimeter to check if they can maintain a voltage +/- 5% of their expected value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. a. The power supply will be tested with a digital multimeter to check if it can maintain a voltage within the given range of 4.7V – 5V. It will be tested after charging, after 10 minutes of use, and after 20 minutes of use to make sure that it maintains 4.8V.</td>
</tr>
<tr>
<td>Haptic Driver</td>
<td>1. The Haptic Driver must drive a serial communication pin of 5V +/- 0.1 V.</td>
<td>1. a. The Arduino will be powered with 9V from the bench to VCC.</td>
</tr>
<tr>
<td></td>
<td>2. The Arduino must be able to respond to serial communication.</td>
<td>b. The Arduino will be programmed with a “serial communication on” program, that constantly modulates pins 3, 5, 10, 11, and 13 on the Arduino.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c. An oscilloscope will measure the voltage from each pin above to ground to verify that each pin drives at 5V +/- 0.1 V.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. A computer will send serial data to the Haptic Driver via USB to RS232 to TTL converters. The Arduino will confirm via LED that it is receiving data.</td>
</tr>
<tr>
<td>Power Circuit</td>
<td>1. The TLC5940 chip requires that a current reference resistor will be selected at 650 Ohms +/- 10% to ensure current sink draws 60mA.</td>
<td>1. An ohmmeter will verify the resistance of the IREF resistor to be at 650 ohms +/- 10%.</td>
</tr>
<tr>
<td></td>
<td>2. The TLC5940 must supply 60mA +/- 10% at each output, when each output is turned on.</td>
<td>2. a. 4.8 V will be supplied from the bench to VCC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b. A resistor of value 650 Ohms will be connected between pin IREF to GND.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c. The haptic driver will turn each output on individually via serial communication to the Power Circuit.</td>
</tr>
<tr>
<td>Motor Array</td>
<td>1. The motors must maintain vibration when driven with 60mA.</td>
<td>1. 60mA will be supplied from the benchtop source. Each motor will tested via a voltmeter and by feel.</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>2. The flyback diode circuit must dissipate power from the stored current in the motor when the constant current is cut off from the motors.</td>
<td>2. a. Measure the voltage across a motor with an oscilloscope. Wire a diode in parallel with a resistor in series with the motor. 60mA will be supplied to the motor, reverse biasing the diode.</td>
</tr>
</tbody>
</table>

b. The voltage induced by the motor when the current is terminated can be no greater than 17V.
Appendix C: Software Plots and Source Code

Following is a plot that demonstrates the three mapping functions: linear, exponential, and logarithmic.

![Plot of Mapping Functions]

Figure C.1: Mapping the input values 1-5 to the output values 5-12

Following is the source code for the Haptic Driver, in its entirety:

```
#include "Tlc5940.h"
#include <math.h>

/* GLOBAL CONSTANTS */
const int NUM_POINTS = 2;                      // Number of data points displayed. Not tested above 3
const int COS_TABLE_SIZE = 200;                // Number of entries in the cosine lookup table
const int DATA_PT_SIZE = 4;                    // Number of values per data point (4 for intensity, x, y, freq)
const int NUM_TLC_OUTS = NUM_TLCS*16;          // Number of OUT pins (16 per TLC chip)
const float pi = 3.141593;
const float IN_INTENSITY_MIN = 0;
const float IN_INTENSITY_MAX = 1000;
const float OUT_INTENSITY_MIN = 400;
```
const float OUT_INTENSITY_MAX = 4095;

const float IN_X_MIN = 0;
const float IN_X_MAX = 200;
const float OUT_X_MIN = 0;             // 0 - 2 because x dimension is 3 motors wide
const float OUT_X_MAX = 2;

const float IN_Y_MIN = 0;
const float IN_Y_MAX = 900;
const float OUT_Y_MIN = 0;            // 0 - 9 because y dimension is 2x5 motors tall (2 arrays)
const float OUT_Y_MAX = 9;

const float IN_FREQ_MIN = 0;
const float IN_FREQ_MAX = 1000;
const float OUT_FREQ_MIN = 0;         // Value of 1 corresponds to ~0.25 Hz
const float OUT_FREQ_MAX = 100;       // Value of 100 corresponds to ~25 Hz

/* GLOBAL VARIABLES */
int tag = 0;
int intensity = 0;
float xloc = 0;
float yloc = 0;
float freq = 0;

/* CHANGE FUNCTIONAL_MAPPING HERE */
boolean functional_mapping = true;     //true for functional mapping, false for direct mapping
boolean new_points = false;            //true if getInput() returns a new set of data points to use

float cos_vals[COS_TABLE_SIZE];        //holds 1000 equally spaced samples of one cycle of cosine
float cos_ptrs[NUM_POINTS];            //for each data point, points to the current location in the cosine cycle
float cos_skip[NUM_POINTS];            //holds the number of cosine samples each data point's frequency skips

float points[NUM_POINTS][DATA_PT_SIZE]; //holds the current data points

float max_motor_ints[NUM_POINTS][NUM_TLC_OUTS]; //holds the max calculated intensities of every motor for every data point
float curr_motor_ints[NUM_TLC_OUTS]; //holds the current calculated motor intensities for every motor

void setup()
{
    // Call Tlc.init() to setup the tlc
    Tlc.init();

    // Set up Serial library at 9600 bps
    Serial.begin(9600);
//compute cosine values
for(int i = 0; i < COS_TABLE_SIZE; i++)
{
    cos_vals[i] = (cos(2*pi*i/COS_TABLE_SIZE)+1)/2;
}

//clear garbage from other arrays
for(int i = 0; i < NUM_POINTS; i++)
{
    for(int j = 0; j < DATA_PT_SIZE; j++)
    {
        points[i][j] = 0;
    }

    for(int j = 0; j < NUM_TLC_OUTS; j++)
    {
        max_motor_ints[i][j] = 0;
        curr_motor_ints[j] = 0;
    }

    cos_ptrs[i] = 0;
    cos_skip[i] = 0;
}

// Output some info to user
Serial.println("Haptic Array v1.0");
if(Functional_mapping)
{
    Serial.println("Currently using **functional mapping**");
    Serial.println("Number of TLC chips is ");
    Serial.println(NUM_TLCS, DEC);
    Serial.println("Input data to the array like so:");
    Serial.println("X - data point ID");
    Serial.println("XXXX - intensity");
    Serial.println("XXX - x location");
    Serial.println("XXX - y location");
    Serial.println("XXXX - frequency");
}
else // direct mapping
{
    Serial.println("Currently using **direct mapping**");
    Serial.println("Number of TLC chips is ");
    Serial.println(NUM_TLCS, DEC);
    Serial.println("Input data to the array like so:");
    Serial.println("XX - motor number");
    Serial.println("XXXX - intensity");
}

void loop()
{

// Check for serial input
getInput();

if(functional_mapping)
{
    // Check if new data points have been read and update
    if(new_points)
    {
        // Reset new_points
        new_points = false;

        // Calculate new motor max intensities for every data point and map freq
        for(int i = 0; i < NUM_POINTS; i++)
        {
            intensity = mapIntensity(points[i][0]);
xloc = mapX(points[i][1]);
yloc = mapY(points[i][2]);

            // Calculate the maximum intensities for each motor, given these points
            calcMaxMotorIntensities(i);

            freq = mapFreq(points[i][3]);

            // Convert the frequency value to a skip value
            cos_skip[i] = freq;

            Serial.print("New data mapped for data point ");
            Serial.println(i, DEC);
            Serial.print("intensity = ");
            Serial.println(intensity, DEC);
            Serial.print("xloc = ");
            Serial.println(xloc, DEC);
            Serial.print("yloc = ");
            Serial.println(yloc, DEC);
            Serial.print("freq = ");
            Serial.println(freq, DEC);

            // Set cos_ptrs to 0 if frequency is 0
            if(cos_skip[i] == 0)
            {
                cos_ptrs[i] = 0;
            }
        }
    }
}

// Calculate changes from frequency and superimpose data points
calcCurrMotorIntensities();

// Send new motor intensity values to the TLC5940 chips
Tlc.update();

else // direct mapping
{  
    // Tlc.set is called in getInput for direct mapping
    if(new_points)
    {
        new_points = false;
        Tlc.update();
    }
}
delay(20);

/******************************************************************************
******
getInput
For functional mapping, if serial data is available, tries to read a full "message" (15 bytes) from
serial input. The order of bytes is:
1    tag
2-5  intensity
6-8  x location
9-11 y location
12-15 frequency

For direct mapping, if serial data is available, tries to read a full "message" (6 bytes) from
serial input. Also, Tlc.set is called here for convenience. The order of bytes is:
1-2  motor # (00 through 15 or 31)
3-6  motor intensity (0000 through 4095)

If serial data is read, the appropriate global variables are updated. The points array is
updated with the new data points.
******************************************************************************
******
void getInput()
{
    int int1 = 0;
    int int2 = 0;
    int int3 = 0;
    int int4 = 0;
    int x1 = 0;
    int x2 = 0;
    int x3 = 0;
    int y1 = 0;
    int y2 = 0;
    int y3 = 0;
    int freq1 = 0;
    int freq2 = 0;
    int freq3 = 0;
    int freq4 = 0;
    int ser_int = 0;
    int ser_x = 0;
int ser_y = 0;
int ser_freq = 0;

if(functional_mapping)
{
    // Only read if we know full sets of new data points are stored in the receive buffer
    if(Serial.available() && Serial.available()%15 == 0)
    {
        Serial.println("Reading serial data...");

        while(Serial.available())
        {
            tag = Serial.read()-'0'; // The data point tag
            int1 = Serial.read()-'0'; // Most significant byte of intensity
            int2 = Serial.read()-'0';
            int3 = Serial.read()-'0';
            int4 = Serial.read()-'0'; // Least significant byte of intensity
            x1 = Serial.read()-'0'; // Most significant byte of x location
            x2 = Serial.read()-'0';
            x3 = Serial.read()-'0'; // Least significant byte of x location
            y1 = Serial.read()-'0'; // Most significant byte of y location
            y2 = Serial.read()-'0';
            y3 = Serial.read()-'0'; // Least significant byte of y location
            freq1 = Serial.read()-'0'; // Most significant byte of frequency
            freq2 = Serial.read()-'0';
            freq3 = Serial.read()-'0';
            freq4 = Serial.read()-'0'; // Least significant byte of frequency

            ser_int = 1000*int1 + 100*int2 + 10*int3 + int4;
            ser_x = 100*x1 + 10*x2 + x3;
            ser_y = 100*y1 + 10*y2 + y3;
            ser_freq = 1000*freq1 + 100*freq2 + 10*freq3 + freq4;
        }

        if(tag < NUM_POINTS)
        {
            Serial.print("tag = ");
            Serial.println(tag, DEC);
            Serial.print("ser_int = ");
            Serial.println(ser_int, DEC);
            Serial.print("ser_x = ");
            Serial.println(ser_x, DEC);
            Serial.print("ser_y = ");
            Serial.println(ser_y, DEC);
            Serial.print("ser_freq = ");
            Serial.println(ser_freq, DEC);

            points[tag][0] = ser_int;
            points[tag][1] = ser_x;
            points[tag][2] = ser_y;
            points[tag][3] = ser_freq;
        }
    }
}
// Set flag to indicate that new data has been read
new_points = true;
}
}
}
else // direct mapping
{
    // Only read if we know full sets of new data points are stored in the receive buffer
    if(Serial.available() && Serial.available()%6 == 0)
    {
        Serial.println("Reading serial data...");

        while(Serial.available())
        {
            x1 = Serial.read() - '0';  // Most significant byte of x location
            x2 = Serial.read() - '0';
            int1 = Serial.read() - '0';  // Most significant byte of intensity
            int2 = Serial.read() - '0';
            int3 = Serial.read() - '0';
            int4 = Serial.read() - '0';  // Least significant byte of intensity

            ser_int = 1000*int1 + 100*int2 + 10*int3 + int4;
            ser_x = 10*x1 + x2;

            Tcl.set(ser_x, intensityBounds(ser_int));

            // Set flag to indicate that new data has been read
            new_points = true;
        }
    }
}

/******************************************************************************************
******
Channel Mapping
These functions call the mapping functions for each input value of intensity, x location, y location, and frequency. One of the three mapping functions (linear, exponential, and logarithmic) are chosen and uncommented.
*******************************************************************************************/
int mapIntensity(float intensity)
{
    // Check input bounds
    if(intensity < IN_INTENSITY_MIN)
        intensity = IN_INTENSITY_MIN;
    else if(intensity > IN_INTENSITY_MAX)
        intensity = IN_INTENSITY_MAX;

    // Perform one of the three mappings
return (int)linearMapping(intensity, IN_INTENSITY_MIN, IN_INTENSITY_MAX, OUT_INTENSITY_MIN, OUT_INTENSITY_MAX);
//return (int)exponentialMapping(intensity, IN_INTENSITY_MIN, IN_INTENSITY_MAX, OUT_INTENSITY_MIN, OUT_INTENSITY_MAX);
//return (int)logarithmicMapping(intensity, IN_INTENSITY_MIN, IN_INTENSITY_MAX, OUT_INTENSITY_MIN, OUT_INTENSITY_MAX);
}

float mapX(float xloc)
{
    // Check input bounds
    if(xloc < IN_X_MIN)
        xloc = IN_X_MIN;
    else if(xloc > IN_X_MAX)
        xloc = IN_X_MAX;

    // Perform one of the three mappings
    return linearMapping(xloc, IN_X_MIN, IN_X_MAX, OUT_X_MIN, OUT_X_MAX);
    //return exponentialMapping(xloc, IN_X_MIN, IN_X_MAX, OUT_X_MIN, OUT_X_MAX);
    //return logarithmicMapping(xloc, IN_X_MIN, IN_X_MAX, OUT_X_MIN, OUT_X_MAX);
}

float mapY(float yloc)
{
    // Check input bounds
    if(yloc < IN_Y_MIN)
        yloc = IN_Y_MIN;
    else if(yloc > IN_Y_MAX)
        yloc = IN_Y_MAX;

    // Perform one of the three mappings
    return linearMapping(yloc, IN_Y_MIN, IN_Y_MAX, OUT_Y_MIN, OUT_Y_MAX);
    //return exponentialMapping(yloc, IN_Y_MIN, IN_Y_MAX, OUT_Y_MIN, OUT_Y_MAX);
    //return logarithmicMapping(yloc, IN_Y_MIN, IN_Y_MAX, OUT_Y_MIN, OUT_Y_MAX);
}

float mapFreq(float freq)
{
    // Check input bounds
    if(freq < IN_FREQ_MIN)
        freq = IN_FREQ_MIN;
    else if(freq > IN_FREQ_MAX)
        freq = IN_FREQ_MAX;

    // Perform one of the three mappings
    return linearMapping(freq, IN_FREQ_MIN, IN_FREQ_MAX, OUT_FREQ_MIN, OUT_FREQ_MAX);
    //return exponentialMapping(freq, IN_FREQ_MIN, IN_FREQ_MAX, OUT_FREQ_MIN, OUT_FREQ_MAX);
    //return logarithmicMapping(freq, IN_FREQ_MIN, IN_FREQ_MAX, OUT_FREQ_MIN, OUT_FREQ_MAX);
}
Mapping Functions

These functions mathematically map their input values to output values, using the parameters set in the global variables. These functions are called by the Channel Mapping functions.

The linearMapping function will map the input value (first forcing it to be within the input range set by iMin and iMax) to the output range set by oMin and oMax using a linear relationship. exponentialMapping and logarithmicMapping work in a similar fashion. Note that logarithmicMapping also forces the input to be greater than or equal to one.

```c
float linearMapping(float value, float iMin, float iMax, float oMin, float oMax)
{
    if(iMin != iMax) //avoid dividing by zero
        value = (value - iMin)/(iMax - iMin)*(oMax - oMin)+oMin;
    else
        value = oMin;
    return value;
}

float exponentialMapping(float value, float iMin, float iMax, float oMin, float oMax)
{
    if(iMin != iMax) //avoid dividing by zero
    {
        value = (exp(value/100)-exp(iMin/100))/(exp(iMax/100)-exp(iMin/100))*(oMax-oMin)+oMin;
        // divide by 100 to keep values from overflowing
    }
    else
        value = oMin;
    return value;
}

float logarithmicMapping(float value, float iMin, float iMax, float oMin, float oMax)
{
    if(iMin != iMax && value > 0 && iMin > 0 && iMax > 0) //avoid dividing by zero and undefined logs
    {
        value = (log(value)-log(iMin))/(log(iMax)-log(iMin))*(oMax-oMin)+oMin;
    }
    else
        value = oMin;
    return value;
}
```

*********************************************************************************

******

calcMaxMotorIntensities

This function takes a data_point as input and iterates through every motor, calculating the maximum vibrating intensity for each motor, taking into consideration location, which can fall between two motors. The intensities are then put into an array called max_motor_ints.
In the case of a location that falls between neighboring motors, linear interpolation is used to create a natural stimulus.

Note that the calculated intensity is referred to as the "maximum" because frequency will cause the intensity to fall below this value during operation.

```c
/**
  * void calcMaxMotorIntensities(int data_point)
  * {
  *     int curr_x = 0;
  *     int curr_y = 0;
  *     float xdiff = 0;
  *     float ydiff = 0;
  *     float dist = 0;
  *     int new_intensity = 0;
  *     for(int i = 0; i < 32; i++)
  *     {
  *         if(i%16 != 15) //the 16th output of chips aren't used for the 3x5 array
  *         {
  *             if(i < 15)
  *             {
  *                 curr_x = i%3;
  *                 curr_y = i/3;
  *             }
  *             else
  *             {
  *                 curr_x = (i%16)%3;
  *                 curr_y = (i-1)/3;
  *             }
  *             dist = sqrt(pow(xloc-curr_x,2)+pow(yloc-curr_y,2));
  *             if(dist < 1)
  *             {
  *                 new_intensity = intensity/(1-dist);
  *             }
  *             else
  *             {
  *                 new_intensity = 0;
  *             }
  *             max_motor_ints[data_point][i] = intensityBounds(new_intensity);
  *         }
  *     }
  * }
  * *****/

/**
  * intensityBounds
  * This function takes a value as an input and returns that value bounded between 0 and 4095. The Tlc.set function requires the intensity input to be between 0 and 4095.
  */
```
```c
int intensityBounds(int value)
{
    if(value < 0)
        return 0;
    if(value > 4095)
        return 4095;
    return value;
}
```

```c
void calcCurrMotorIntensities()
{
    int new_intensity = 0;
    // Iterate through all motors
    for(int i = 0; i < NUM_TLC_OUTS; i++)
    {
        if(i%16 != 15) //the 16th output of chips aren't used for the 3x5 array
        {
            new_intensity = 0;
            // Iterate through all data points
            for(int j = 0; j < NUM_POINTS; j++)
            {
                // Superimpose all data points and use frequency for intensities, accounting for
                // minimum intensity
                new_intensity += (max_motor_ints[j][i] - OUT_INTENSITY_MIN)*cos_vals[(int)cos_ptrs[j]]
                + OUT_INTENSITY_MIN;
            }
            curr_motor_ints[i] = new_intensity;
        }
        // Set new superimposed intensity for the motor
        Ttc.set(i,intensityBounds(new_intensity));
    }
```

This function iterates through every motor and data point, superimposing intensities from multiple data points. It also accounts for frequency for every data point using a raised cosine lookup table.

New intensities for every data point are calculated by multiplying the maximum intensity by the value in the lookup table.

Once the actual intensity is calculated, Tlc.set is called and each data point's pointer in the lookup table is advanced.


}  

// Update all frequency variables  
for(int i = 0; i < NUM_POINTS; i++)  
{
    // Advance cos_ptrs, making sure to wrap around cos_vals table (can't use % since we have floats)  
    cos_ptrs[i] = (cos_ptrs[i] + cos_skip[i]);  
    while(cos_ptrs[i] > COS_TABLE_SIZE)  
        cos_ptrs[i] -= COS_TABLE_SIZE;  
}  
}