

Worcester Polytechnic Institute

Infrared Multitouch Interface

MQP Final Report

Sean Thulin
1/13/2010

Abstract

Touch based computing (also known as a Natural User Interface) is a rapidly growing trend. While touch based screens have been around for several years now, they had been limited to a single touch at a time. Recently, support has been building for a new type of system that allows multiple touches at the same time. This allows more than 1 person to interact with the screen at the same time as well as support for gestures commands such as rotate, scale, and others. Currently, the majority of these systems are based on using a projector and a camera. This requires a large area to house the enclosure. With the recent decrease in costs, flat panel displays have become very affordable. The goal of this project is to design a multi touch system that would work with an existing flat panel display.

Table of Contents

Abstract.....	2
Introduction	5
Initial Design and Research.....	7
Initial Approach.....	7
Component Research and Selection.....	9
Testing Procedure and Results	11
Circuit Design and Layout	13
Transmitter and Receiver Board	14
Microcontroller	18
Printed Circuit Board Layout.....	19
PCB Assembly.....	21
Testing.....	24
C Programming.....	26
Detection processing	27
Revision history.....	29
Final Thoughts and Lessons Learned	32
Conclusion.....	34
Appendix A – Code.....	35
Appendix B – Layouts and Schematics.....	48
Appendix C – Total Cost	50
Appendix D – Bibliography.....	51

Table of Figures

Figure 1 - Frustrated Total Internal Reflection (FTIR)	5
Figure 2 - Infrared Matrix.....	7
Figure 3 - Infrared Side Scanning Matrix	8
Figure 4 - Design Flow Chart	13
Figure 5 - Transmitter Board Schematic	15
Figure 6 - Reciever Board Schematic	15
Figure 7 - Transmitter Breadboard Mockup	16
Figure 8 - Receiver Breadboard Mockup	17
Figure 9 - MSP-EXP430F5438 Experimenter Board	18
Figure 10 - Transmitter Board.....	20
Figure 11 - Receiver Board.....	20
Figure 12 - Receiver Board Top.....	21
Figure 13 - Receiver Board Bottom.....	22
Figure 14 - Transmitter Board Top.....	22
Figure 15 - Transmitter Board Bottom.....	23
Figure 16 - Corner Connectors.....	23
Figure 17 - Infrared LED Illuminated.....	24
Figure 18 - Fully Assembled Layout	25
Figure 19 - Object Detection Flowchart.....	27
Figure 20 - Data Output via Serial.....	29
Table 1 - Infrared Transmitters	10
Table 2- Infrared Receivers	10

Introduction

Preliminary background research has shown that this type of a system is most commonly accomplished with the use of a layer of infrared light and a camera positioned to pick up the object reflection as shown in Figure 1. While this works well with a high performance computer with customized object detection software, it has its draw backs in that the camera has to be mounted some distance from the surface and usually behind a translucent panel so that no objects block the light.

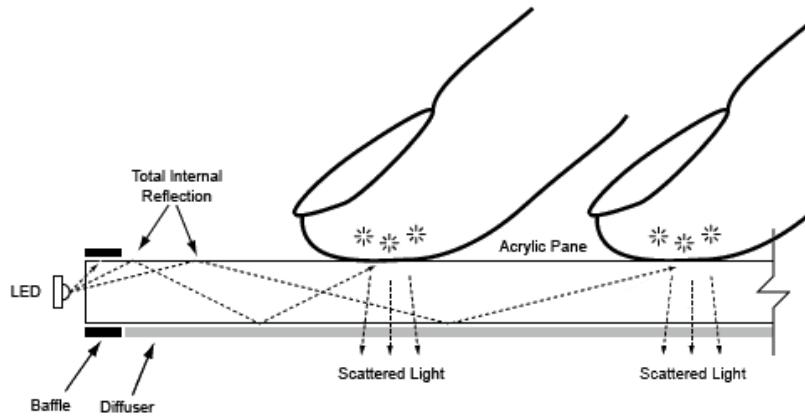


Figure 1 - Frustrated Total Internal Reflection (FTIR)

This detection method does not lend itself very well to modern flat panel displays as there is no open space behind the display layer to mount cameras. Other systems involve a special layer built into the screen. Micro thin wires are laid out in a grid and the capacitance of human fingers causes a detectable change across them. This works well also, but is not easily scalable and cannot be adapted to existing displays as a custom solution for each sized display would be required.

Given the limitations of the methods discussed above, a better method for detecting fingers “touching” an existing flat panel display should be developed. This system needs to offer some sort of invisible detection as it will be interacting with a light emitting display and there should be no interference with it. This is often achieved by using an infrared light detection source since these beams are not part of the visible light spectrum.

Initial Design and Research

Initial Approach

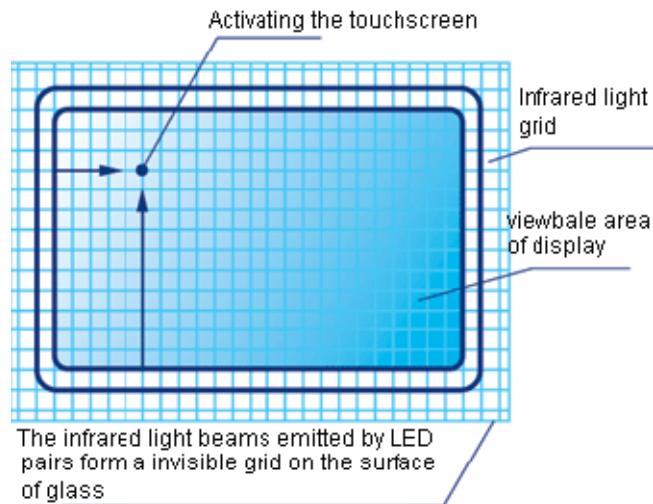


Figure 2 - Infrared Matrix

One commonly used configuration is to layout an infrared beam grid. As illustrated in Figure 2 a row of infrared LEDs would be placed along the top and sides of the display with matching receivers along opposing sides. When an object breaks the field, the sensors will see the shadows and be able compute the X-Y coordinates of the object.

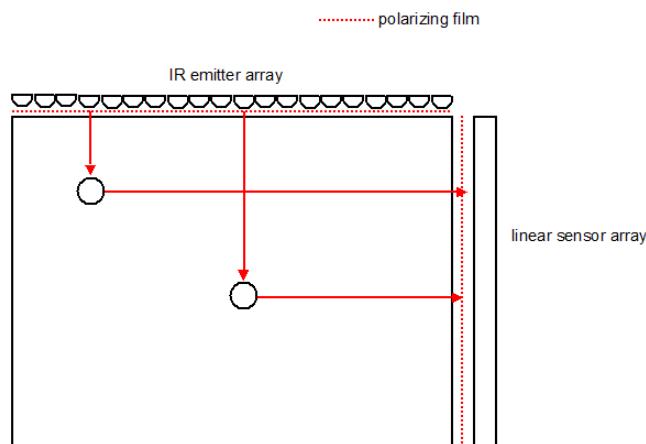


Figure 3 - Infrared Side Scanning Matrix

Another design option that also uses infrared emitters and sensors would require a large piece of acrylic to serve as an overlay of the existing panel. A row of LEDs placed along the top edge would shine into the clear acrylic and be trapped inside. This LED light would pass through a vertical polarizing filter to make it travel straight down. When a finger presses along the upper plane, the light would scatter from that point in all directions as seen in Figure 3. Receivers along the side would pick up this scattered light and measure intensity. Another polarizing filter, placed in front of the sensors, will be required to orient the scattered light and prevent it from going off axis. This design option would reduce the number of electrical components, but may not work with large displays as the polarizing film does reduce the light output by 50% each time it passes through the filter.

Unfortunately, the design option using an acrylic substrate was stopped before it was started. The requirements for the acrylic sheet should be at least a half inch thick and more than

an inch longer than the size of the display on all sides. However, polycarbonate and acrylic sheets are commonly sold in sizes of 4 feet by 8 feet. This would produce several pieces of adequate size for production, but is cost prohibitive for prototyping. Attempts to find a more suitable sized piece also proved difficult as the markup and shipping prices made the cost jump to over \$100. Inquiries made to local supply stores such as Lowes and Home Depot did not yield pieces that are thick enough to be a suitable transmission surface. Products were not of sufficient strength and thickness to resist warping from finger presses. Local picture frame stores were also contacted, but produced similar results. Sufficient products could be ordered, but would incur extra costs because of ordering and distribution charges. This also meant that small samples could not be obtained. Because of these problems, the decision was made to abandon this approach and focus on the other infrared based object detection option that is described in Figure 2.

Component Research and Selection

Unfortunately, the intriguing side scan method described above was not economically feasible. However, other the method under consideration will still require a large number of infrared diodes and detectors. Following the practices from projects in the past, orders were placed with both Mouser Electronics and Digi-Key Corporation. Seven different LED transmitters and six different receivers were selected for testing based on price and availability. 5 of each product were ordered for the purpose of testing and selection. Table 1 and Table 2 detail several different aspects of each product including viewing angle, costs, and power.

Manufac	Part#	Size	Degrees	Wavel	V.Max	C (mA)	Power	Price 1X	Price 500X	Site
Lite-On	LTE-4208	5mm	20	940	1.6	50	0.8	0.24	0.14	Digikey
Kingbright	WP7104F3C	3mm	34	940	1.6	50	0.8	0.2	0.0936	Digikey
Fairchild	QEC112	3mm	16	940	1.5	50	0.75	0.23	0.103	Mouser
Fairchild	QED234	5mm	40	940	1.6	100	1.6	0.2	0.104	Mouser
Fairchild	QED523	5mm	28	880	1.8	100	1.8	0.38	0.197	Mouser
Vishay	TSAL7200	5mm	34	940	1.6	100	1.6	0.21	0.14	Mouser
Vishay	TSAL6100	5mm	20	940	1.6	100	1.6	0.28	0.142	Mouser

Table 1 - Infrared Transmitters

Manufac	Part#	Size	Wavel	RiseTime	FallTime	Price 1X	Price 500X	Site
Lite-On	LTR-3208	5mm	940	10 us	15 us	0.23	0.13126	Digikey
Fairchild	QSC112	3mm	880	5 us	5 us	0.18	0.099	Mouser
Fairchild	QSD122	5mm	880	7 us	7 us	0.28	0.138	Mouser
Fairchild	QSD722	5mm	880	N/A	N/A	0.48	0.16146	Digikey
Vishay	BPV10NF	5mm	940	2.5 us	2.5 us	1.11	0.494	Digikey
Vishay	BPV11F	5mm	930	6 us	5 us	0.57	0.4	Mouser

Table 2- Infrared Receivers

Testing Procedure and Results

The layout of the final product will require a large array of transmitters and receivers flashing very quickly and spaced at a maximum of a few feet apart. A test was devised to simulate this condition with all the different possible permutations of transmitter/receiver pairs. Each set LEDs and Phototransistors was fed a 5 volt DC oscillating signal. This square wave signal would vary in frequency from 1 KHz all the way up to 25 KHz to see if there were any problems with output voltage. The test rigs were spaced roughly 3 feet apart to allow an object to be placed in the middle to block light. The output of the photo transistor was observed with a graphic multimeter and compared against the original signal. This test was required to measure the turn on time of the receiver as well as the time it takes to get a consistent and stable output.

Testing yielded better than expected results since all pairs handled the 25 KHz signal without a problem. Most receivers stabilized faster than the manufacturer stated in the specification documents. However, voltage output varied between manufacturers. While all of the Fairchild phototransistors output voltage levels similar to the input voltage levels, the Lite-On was only able to produce a maximum of 2.5 Volts, and the Vishay diodes only dispensed a little over 1 Volt. This whittled down the list of choices to a Fairchild device. Since all 3 tested devices passed with flying colors, the decision to use the cheapest product was made. However an executive decision to use only 5 millimeter LEDs with the final product changed the decision to the Fairchild QSD-122 Phototransistor.

With the receiver selected, it was a simple process for choosing a transmitter. All 7 transmitters were tested to see which would produce the highest output signal with the phototransistor. As to be expected, the matched infrared LED from Fairchild performed the best and was also one of the cheapest. This was the driving factor in the decision to use the Fairchild QED-234 as the transmitter.

Circuit Design and Layout

For this project, circuit design was done in several phases. The initial planning phase revolved around the approach. In thinking ahead with the design, the decision was made to make the size of the overlay scalable. The frame would be broken down into smaller segments that could then be strung together in any order and still function correctly. This meant that the design would have to incorporate some sort of power and signal pass through system as well as the ability to know where each board is in relation to the end. At the end of each side will be right angled connectors to pass all the signals around.

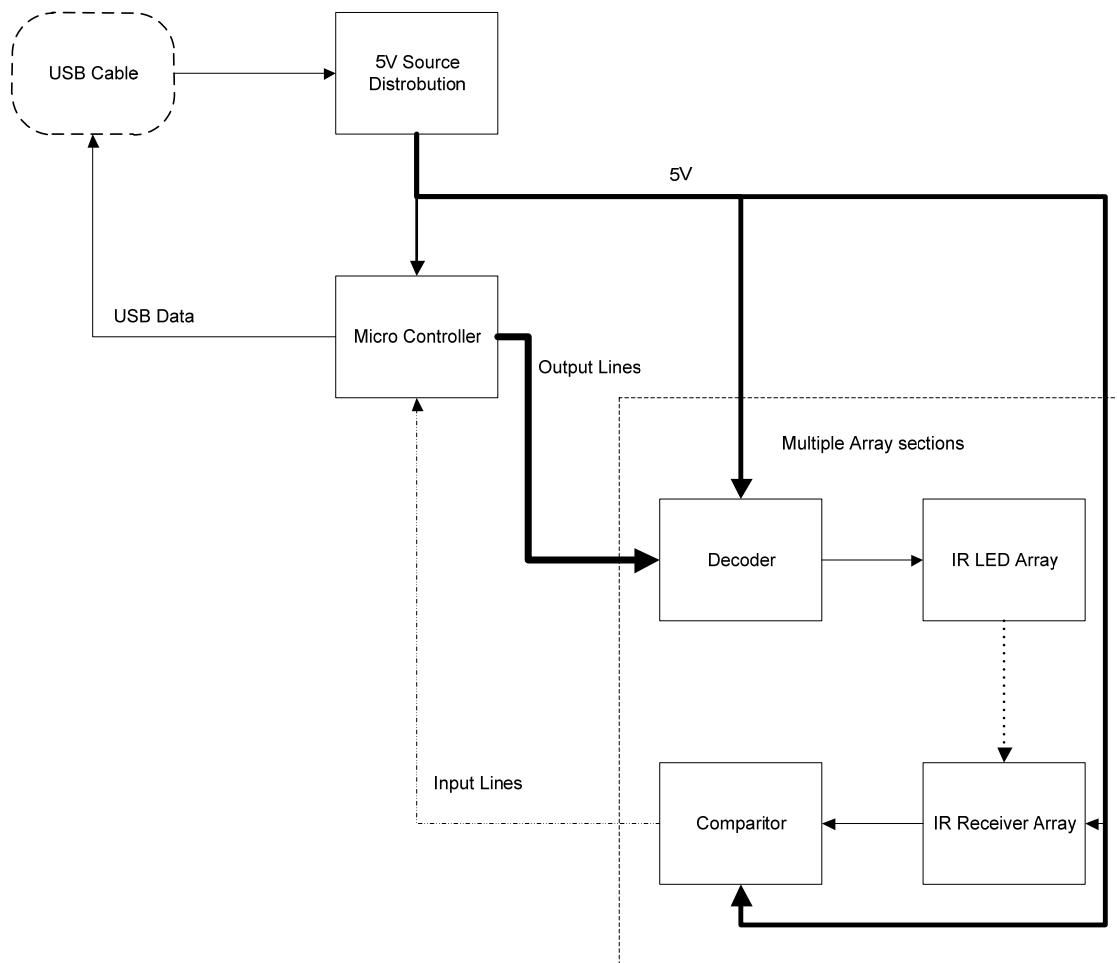


Figure 4 - Design Flow Chart

The design was first conceived using National Instruments MultiSim program. Two different layouts (a transmitter and receiver) were devised. To keep the total number of wires down, a series of decoders would be used to control the system. As described in Figure 4, each board would have power, ground, a clock signal, a chip select, and 4 address lines. These could be passed around to every board as this sequence was not dependent on their position along the frame. The receiver board would need an extra data bus to be able to pass information about each board. A group of 8 traces would be passed along each board. At one end of the board, all the lines would be shifted over by one. This means that the trace for pin one would become pin two on the second board and so on. This way all the data would be transmitted properly and one can reference the position of the board based on the pin they chose.

Transmitter and Receiver Board

Each transmitter board (as seen in Figure 7) is equipped with two 4-to-16 decoders. This would allow a total of 32 individual outputs to be selected. Since the LEDs being used can draw a maximum of 100mA each, and this value is much greater than the total power the decoder could supply safely, the decoder was hooked into an array of darlington pair transistors to handle the switching from low current to high current. After a cost benefit analysis, it was determined that the cheapest way to keep a small footprint was to use a 7 count Darlington array. This meant that my total number of individual selectable outputs would shrink to 28. The array was wired in line with the cathode side of the LEDs to provide a ground switching

setup. All the LEDs would be powered from a central 5V line that was then fed through a 33 Ohm resistor to ensure proper voltage regulation. Larger versions of Figure 5 and Figure 6 can be found in Appendix B.

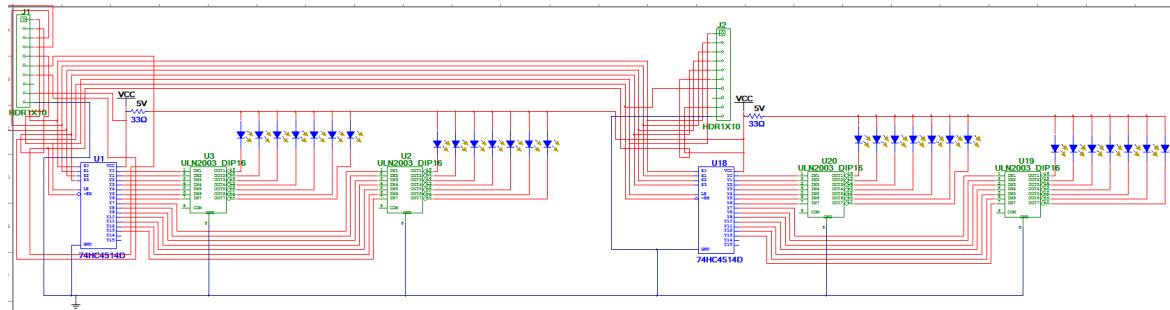


Figure 5 - Transmitter Board Schematic

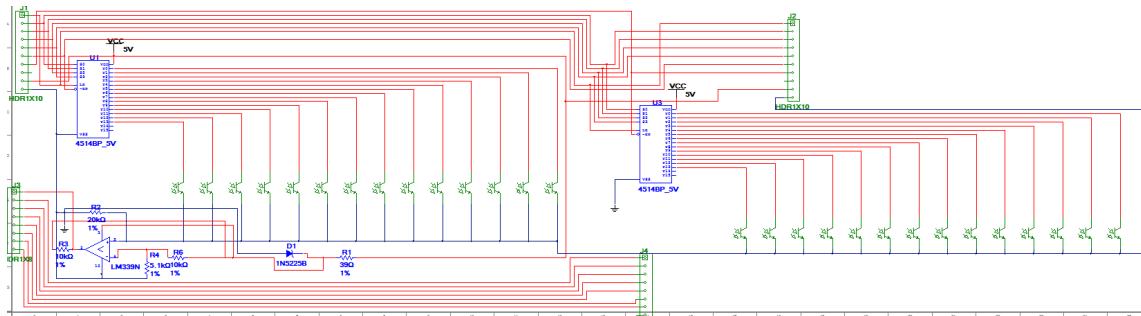


Figure 6 - Reciever Board Schematic

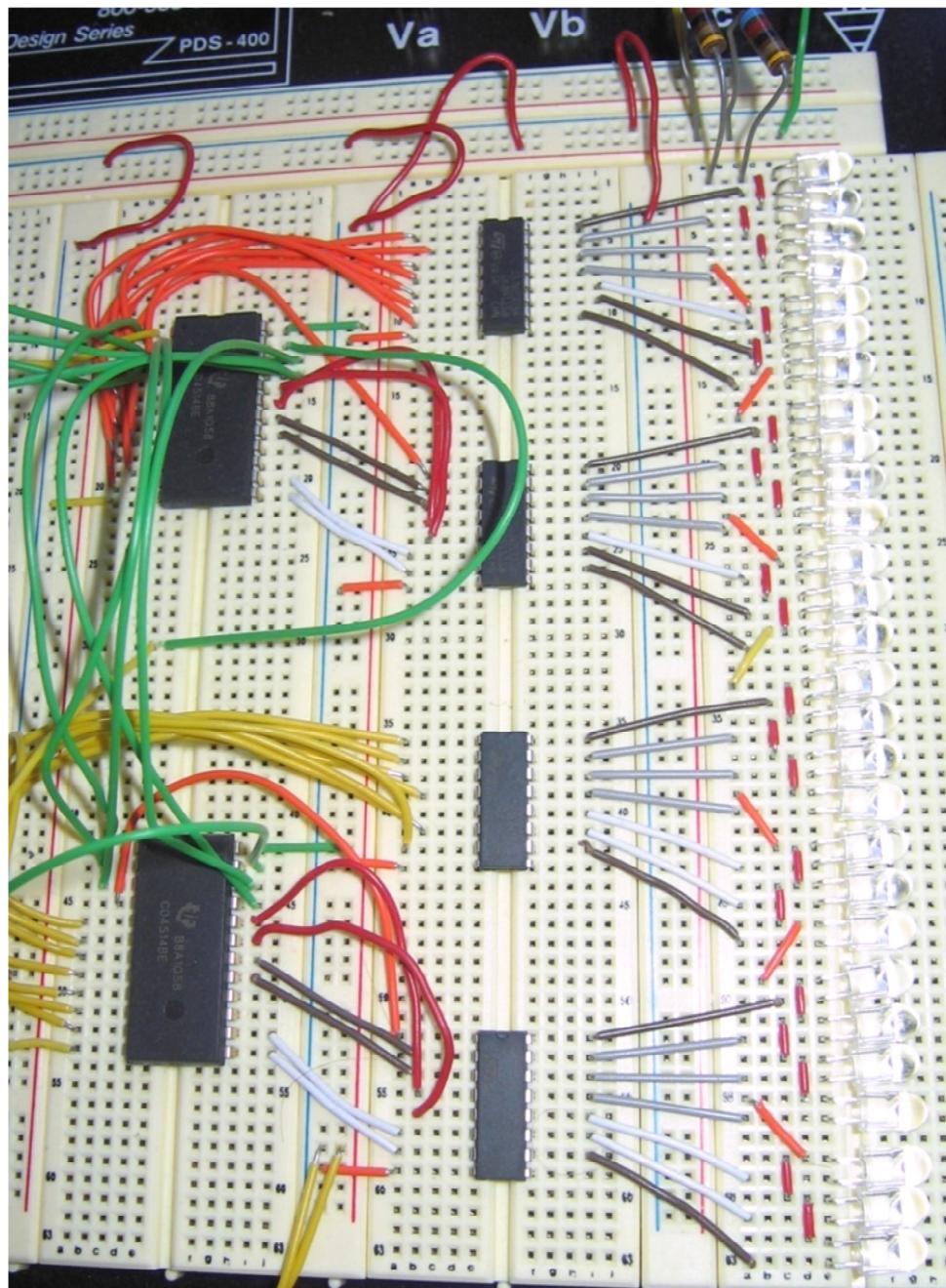


Figure 7 - Transmitter Breadboard Mockup

On the receiver board (as seen in Figure 8), a similar setup to the transmitter board was used. However, since this is a low power circuit, each phototransistor was wired directly to the

decoder to source power. The outputs of the phototransistors were all tied to a single comparator. Since only 1 output would be active at any given time, no isolation is needed. The comparator was wired in reverse output, so that when the input voltage dropped below the threshold, the comparator would output a logic high signal. This was then fed through a zener diode to regulate it to a maximum of 3 volts before reaching the input on a microcontroller.

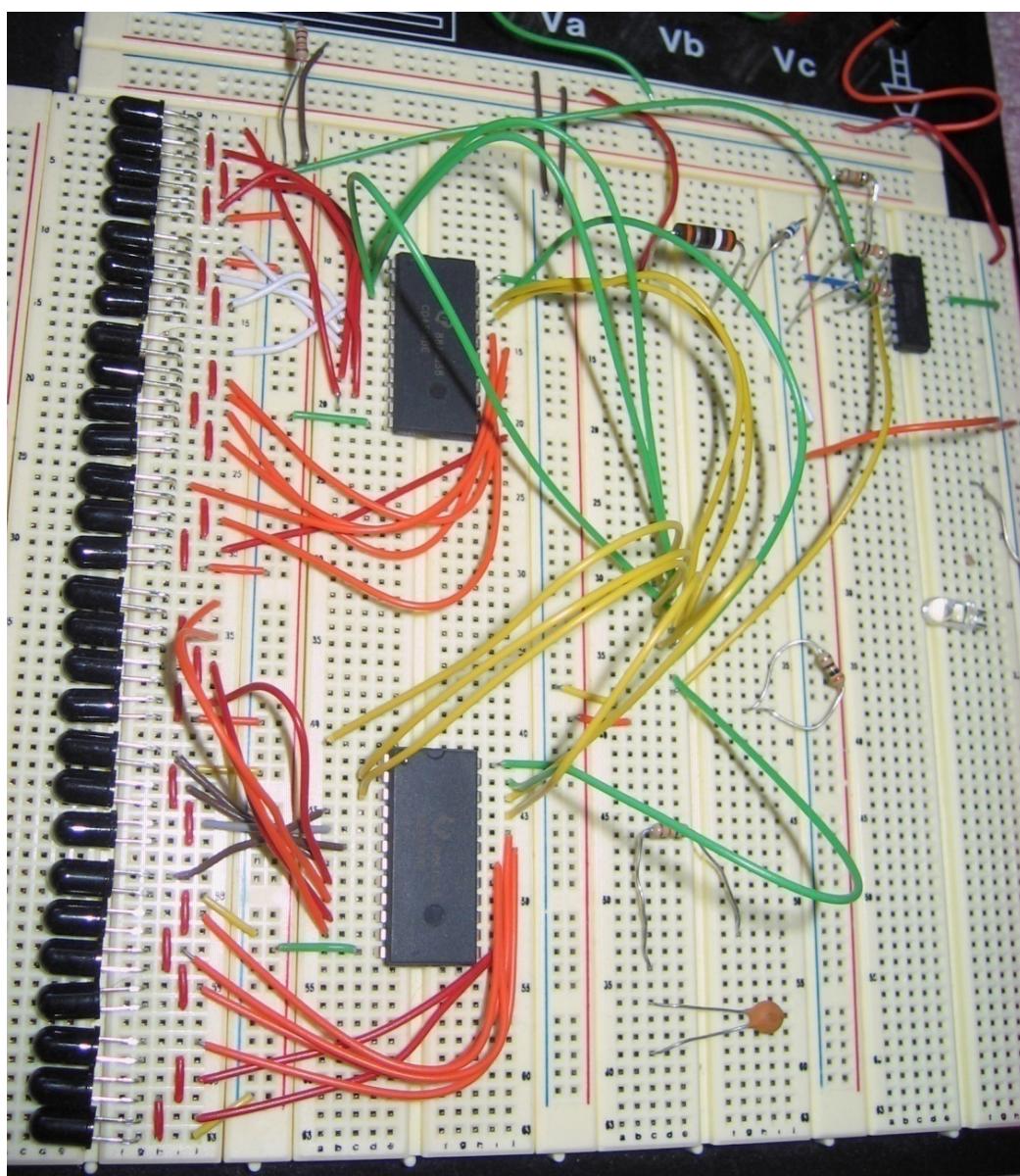


Figure 8 - Receiver Breadboard Mockup

Microcontroller

The microcontroller used was donated by Texas Instruments. The original design called for a chip that had on board USB as well as plenty of I/O ports. The MSP430 series was chosen due to familiarity as I have several years of experience with this chip. With these design constraints, an ideal choice would be TI's newest line, the F55XX series. However, due to unforeseen complications, TI was unable to release a proper development board and chip in time for this project. However, a backup solution was donated by Texas Instruments. The MSP-EXP430F5438 does not have USB built into the MSP430 processor, however this particular experimenter's board has an SPI-to-USB conversion chip onboard. This is obviously not an ideal solution for a final product as it has a lot of extras that were unneeded, but as this solution was donated, it made an adequate substitute.

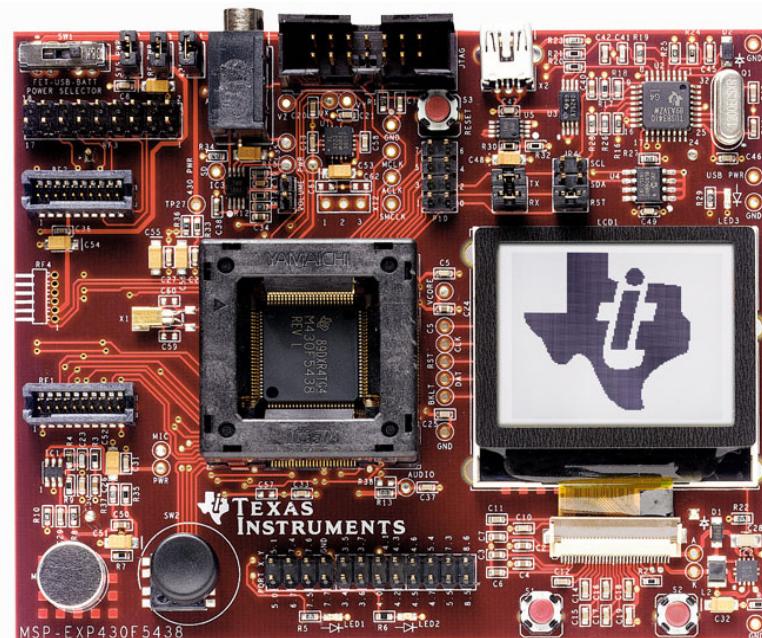


Figure 9 - MSP-EXP430F5438 Experimenter Board

Printed Circuit Board Layout

Circuit board layout design was done using National Instruments companion program called UltiBoard. This program has the ability to import the layout directly from MultiSim. First step in this process was to establish the guidelines of the board layout to minimize costs. After looking at several Printed Circuit Board manufacturing companies, the student discount from Advanced Circuits of Aurora, CO had the best price. To receive the discount, the design had to adhere to certain standards. The PCB could not be more than 60 square inches, the holes couldn't be smaller or bigger than certain sizes, and the board had to be a certain thickness. Sticking to these design constraints was not a problem, but there was no room for error as a 2nd production run was not in the budget.

With the design constraints taken care of, the layout started with the placement of the LEDs. They were spaced along the edge of the board and as close as possible to maximize resolution. With that taken care of, the rest of the components were placed to minimize electrical trace lengths and vias (connections between top and bottom of board). Each board had matching right angle male and female connectors. Main power and ground lines were made thicker than the signal lines to handle the extra current as shown in Figure 10 and Figure 11.

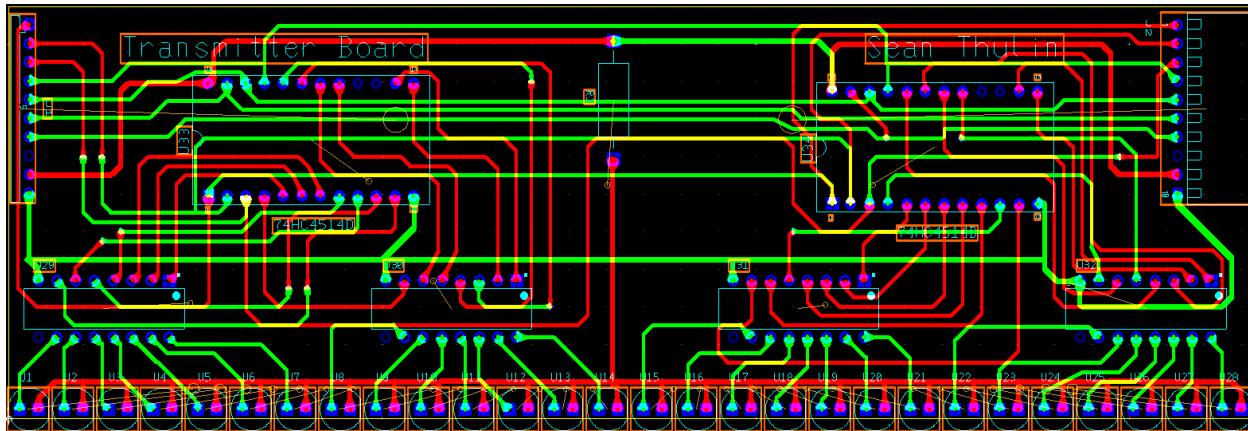


Figure 10 - Transmitter Board

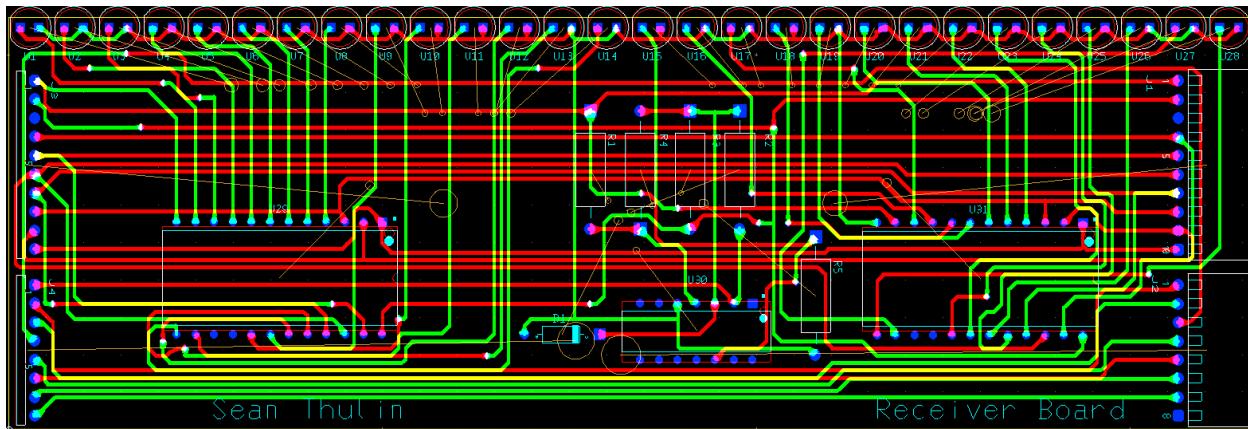


Figure 11 - Receiver Board

With the layout done, it was found that 4 of these could be placed on a single PCB and still fit within the 60 square inch design requirement. This would incur a small step-and-repeat charge, but still placed the total cost well below what other manufacturers offered. Final total count made these boards just over 6 inches long and allowed for 8 of each type of board. This would be more than enough to surround the 32 inch television that had been acquired for use

with this project. The corner connecting pieces were not designed through layout as it would not be cost efficient to have them made. Instead, it was chosen to have them hand assembled instead.

PCB Assembly

Doing PCB layout for the first time is always a learning process. With that being said, there were some issues with the boards that were traced back to problems with the layout in the first place. The clearance between the edge of the holes and the components that fit in them were very tight. All components did fit, but some were harder to place than others. This is just something that happens and extra precaution will be taken next time to increase the size of the drill holes next time. The other problem incurred was a layout issue of an unchecked ground path that was imported from MultiSim. An extra trace was placed in each receiver board design that required the trace to be cut and an extra ground wire to be rerun for one of the decoder chips as shown in Figure 12 and Figure 13.

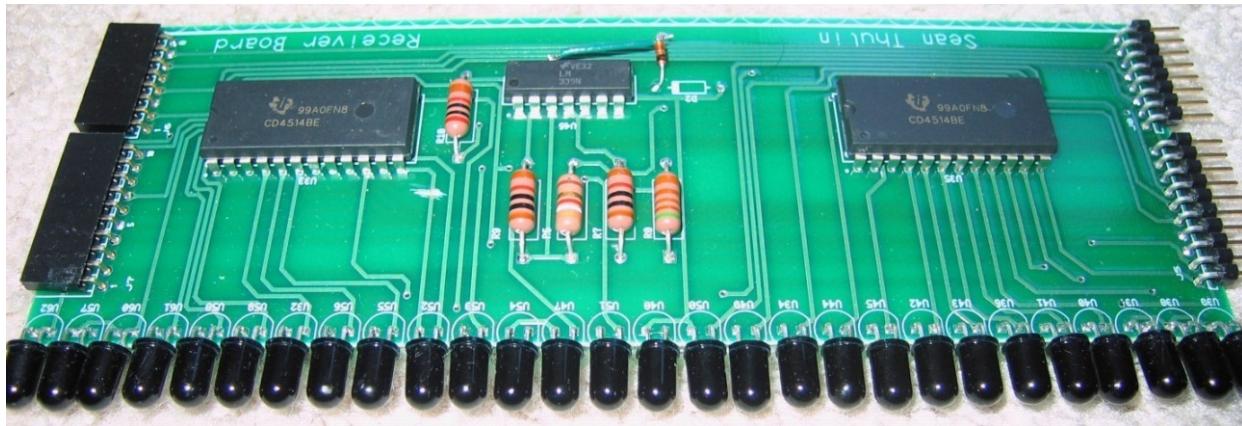


Figure 12 - Receiver Board Top

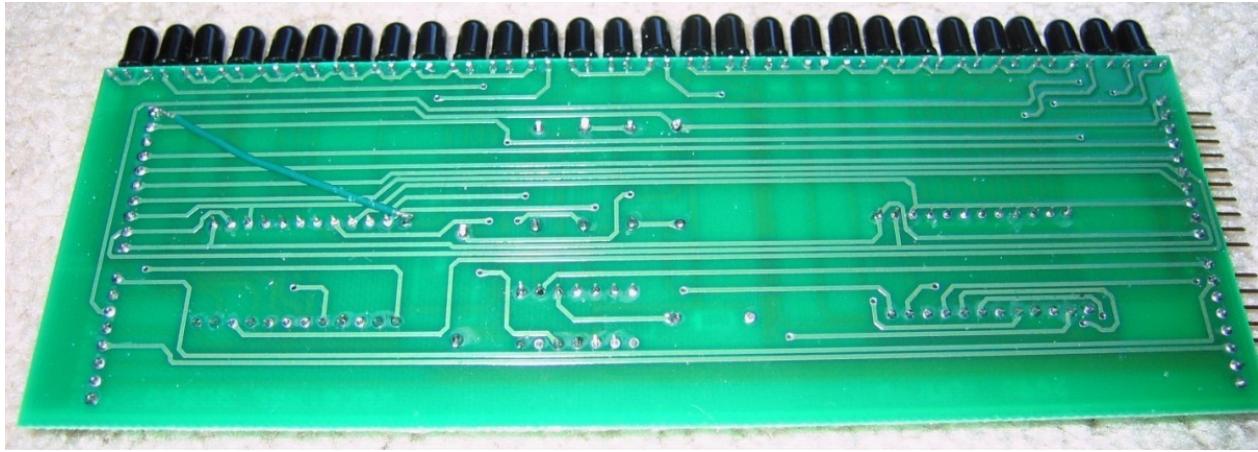


Figure 13 - Receiver Board Bottom

Despite the issues mentioned above, the rest of the assembly went smoothly. All 500+ components were soldered by hand. The transmitter board did not have any design flaws other than the tight fit on the holes for the LEDs. While assembling the boards was a tedious process, it did re-enforce proper soldering techniques as seen in Figure 14 and Figure 15. Extra components were ordered to cover the possible event that one was damaged during assembly. Basic continuity tests were performed on each board to test that a solid connection is made between the component and the circuit board.

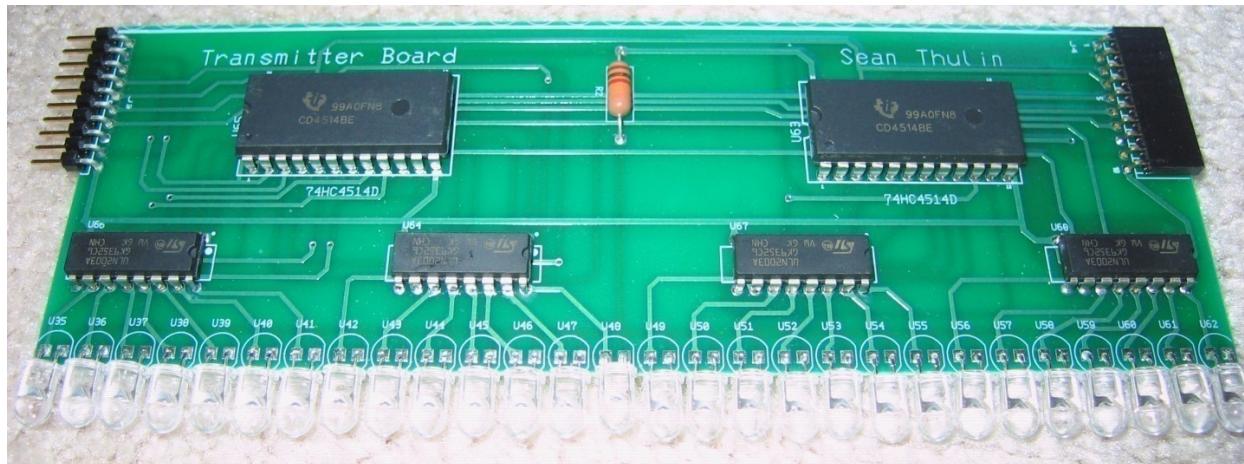


Figure 14 - Transmitter Board Top

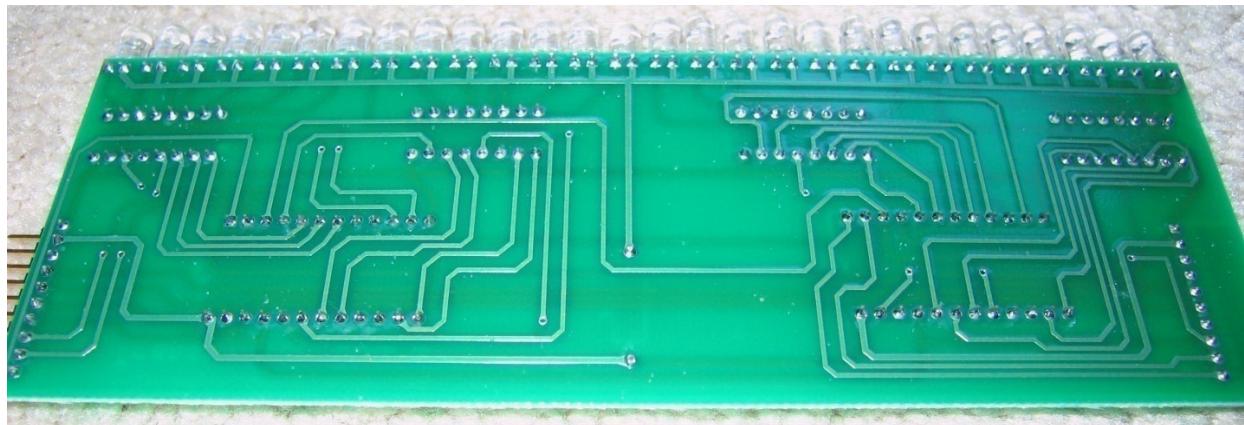


Figure 15 - Transmitter Board Bottom

The right angle pieces were designed to simply pass along the signal wires between the boards at a right angle and also provide a power and data hookup from the microcontroller board. Each of the 4 boards is unique so that it only fits in a specific corner. This design allows for less operator assembly error as the parts won't fit together in any other way. Ideally, they would be manufactured just like the rest of the boards, but for cost reasons, it was simpler to design and assemble these by hand from existing prototype boards as seen in Figure 16.

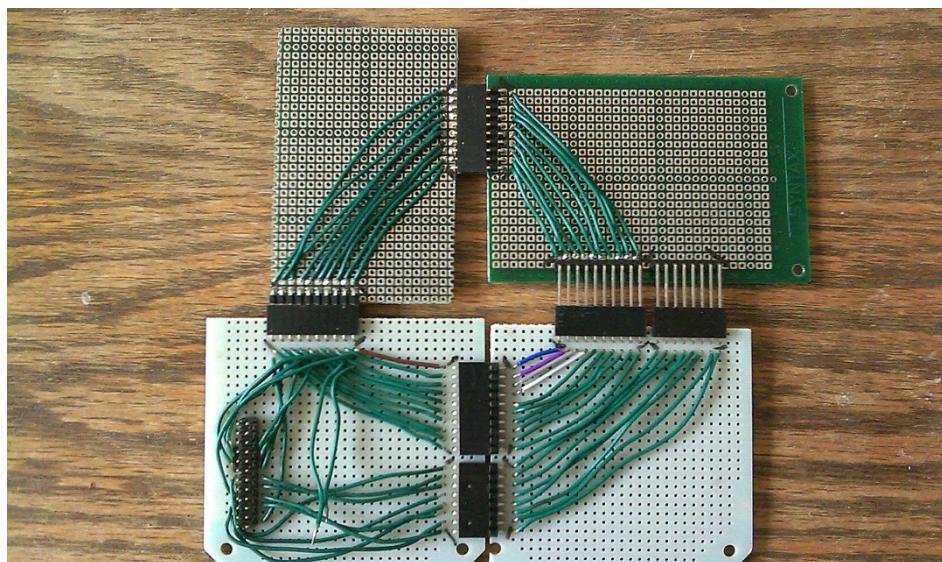


Figure 16 - Corner Connectors

Testing

As discovered earlier, a simple way to track the infrared LEDs is to use a cell phone camera as seen in Figure 17. This method was used again to make sure that each individual LED lights up and to full brightness.

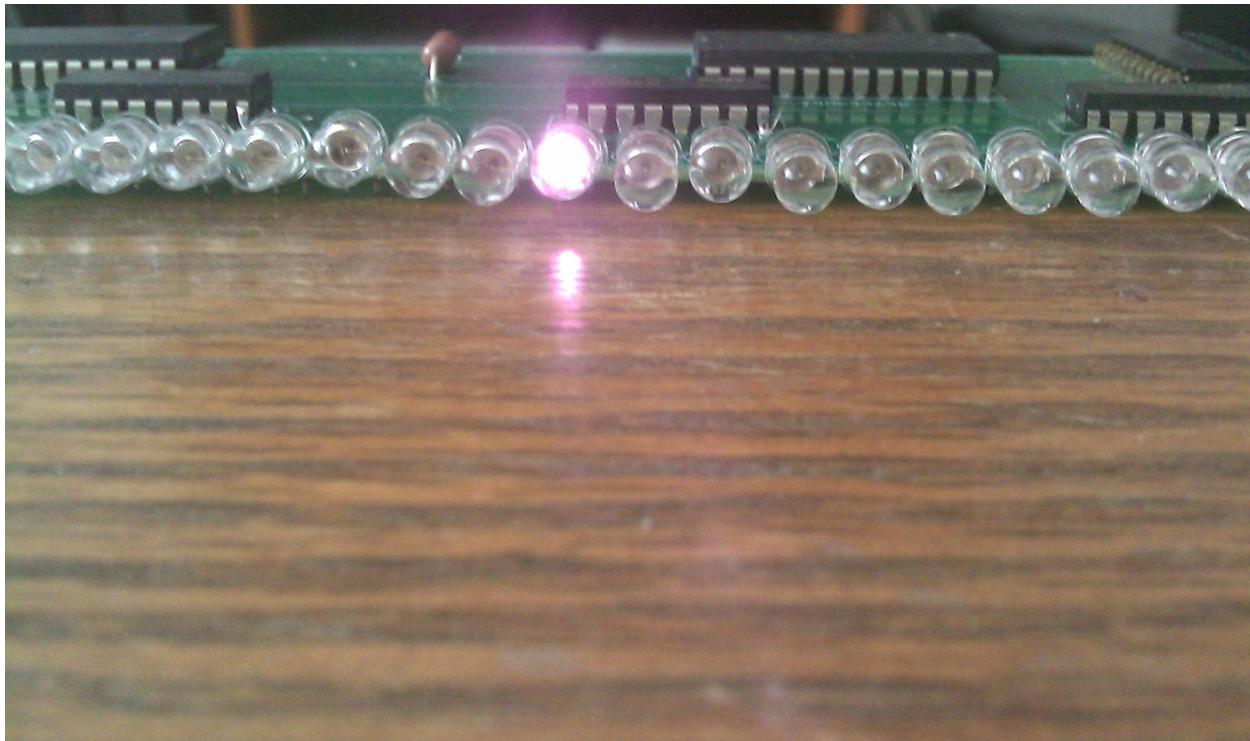


Figure 17 - Infrared LED Illuminated

Since the new development board had a different pin out structure than the testing board used for the prototype, modifications were needed to allow the existing microprocessor code to work with the new setup. Once these were taken care of, it was discovered that not all of the components were aligned properly and needed to be adjusted to achieve a maximum signal to noise ratio. A fully assembled layout can be seen in Figure 18.

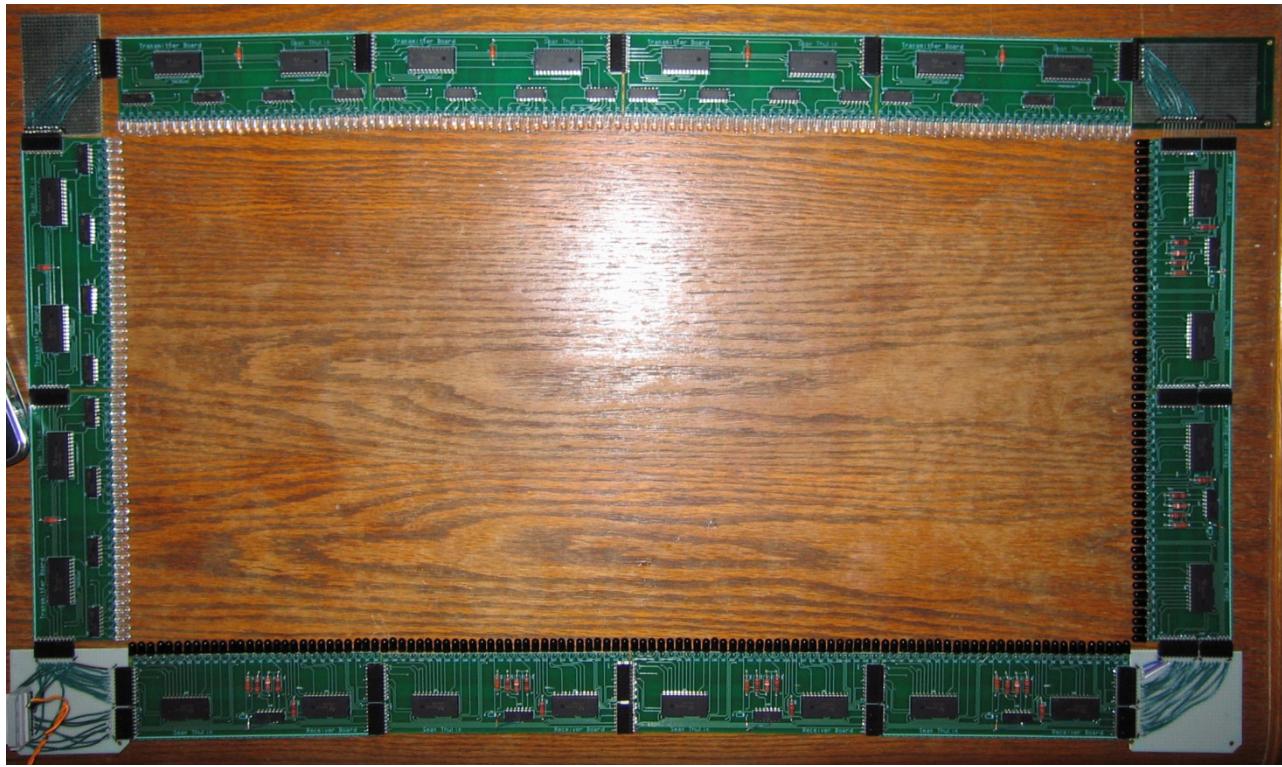


Figure 18 - Fully Assembled Layout

C Programming

Once the hardware testing had been completed, the code and algorithms needed to be tested. Coding for the project will be done using IAR's Embedded Workbench. Due to the amount of code required, the limitations of the free "kickstart edition" (which has a code size limit of 4 KB) would not allow the use of the demo code provided by Texas Instruments, so a full version of the product was used for the project.

The initial code was borrowed from the test program included by Texas Instruments. This was used to initialize the development board and get it into a low power state. This code was provided publicly under Gnu Public License. All the code for the excess components on the MSP-430EXP5438 board (the LCD, accelerometer, microphone, etc...) were removed to cut down on processor usage and power consumption. The next step was to implement a proper scanning procedure. Since each board has a total of 28 LEDs, a simple count up pattern can be used. By using a single register to control the clock and address lines, one can just add one a single number, and it will trigger the clock in between incrementing the address lines.

The clock and address lines are also passed on to the receiver boards so that the matching photodiode is activated at the same time. At this point, each receiver board has a single data line that can be read once the receiver is activated. So the code can simply read from that line and store it in a shift register before moving on to the next cycle. This repeating

pattern is created from a simple “for” loop function that handles incrementing 28 times before resetting.

Each board has 28 bits of data per full cycle and is stored in its own 32 bit variable. This makes it easy to reference exact data points. The input lines are attached to two different registers since TI didn’t provide a full register of expansion pins. The data is read in on each clock cycle. Then it is inverted since a logic low means that an object was detected, but the original specifications called for object detection to be logic high. At the end of that cycle, each bit is appended to the variable for its corresponding board, and then the cycle repeats.

Detection processing

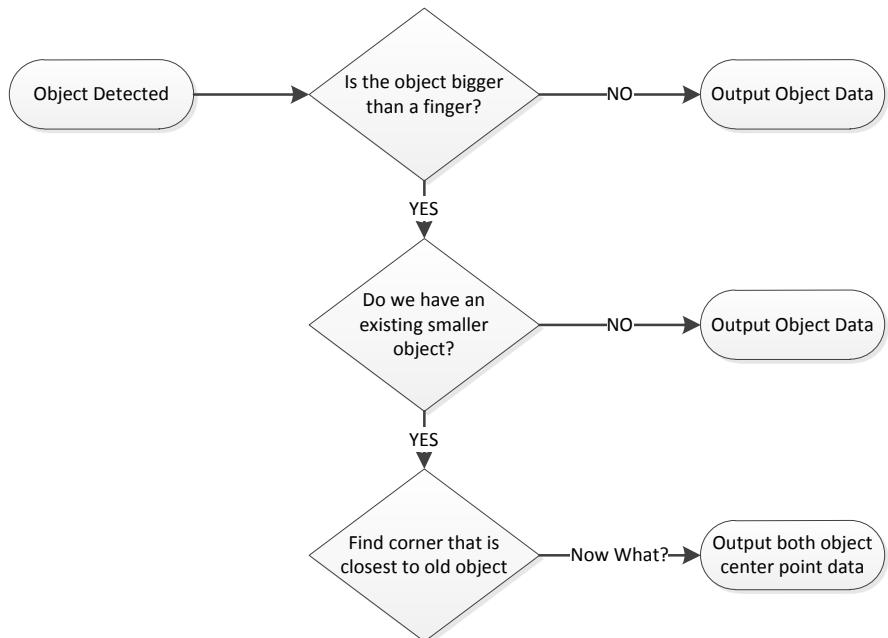


Figure 19 - Object Detection Flowchart

Once a full cycle of all 28 pairs of transmitters and receivers has completed, it's time to run detection and output the results. As seen in Figure 19, the first thing the code does is check to see if there are any hits on a given receiver board. If there aren't then it skips to the next one. This makes the scan rate much faster when the board is idling. If there is a hit, then processing begins. The smallest and largest x and y values are recorded, and then averaged to get a center point and a radius. If the object is small like the size of a finger, it is treated as 1 object. Once the size changes to something bigger, it is assumed that it can't be a single finger. The corners of this new larger object are compared against the center point of the previous pass. The corner that is not close to the previous point is then chosen to be a 2nd object and is treated as such. Once the two points are found, the system refines the center point calculations.

Regardless if there are one or two objects present in the touch field, the output is then sent out via a UART-to-USB converter where it is translated on a computer via a USB-to-Serial adapter. This allows the raw data to be viewed in a terminal screen. This was simply a formality as this is where the user interface ends. Normally, this step would be bypassed as a 2nd piece of software would be created to pass this data to applications. Writing programs for windows proved to be very problematic given the lack of computer science background, so this step was agreed to be skipped. A final output of the sequence can be seen in Figure 20.

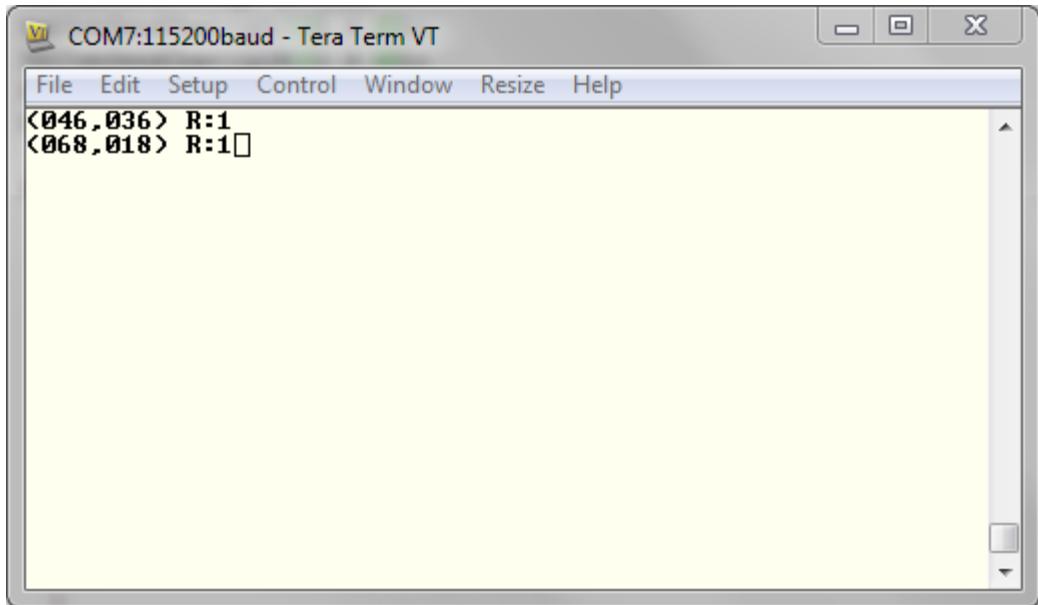


Figure 20 - Data Output via Serial

Revision history

The code for the project was written in several stages starting at the bare basics. After it was established that the MSP430 Experimenter Board worked via the Texas Instruments demo program, most of the program was removed to make room for the sequencer algorithm used from the initial prototype. This was constantly checked with a digital scope that can handle multiple data channels at the same time. After several attempts, a proper algorithm was devised to turn on each LED one at a time and then move to the next one as fast as possible. The speed on this was slowed to make sure that every component had plenty of setup and hold time before moving on to the next step.

After a proper sequence was created (and confirmed via a camera that can see infrared), attention was paid to the receivers. Each one had to be fine-tuned to make sure that it was seeing a proper signal. Due to a design flaw, there wasn't as strong an output as there was in the bread boarded design, so some of the receivers would trigger a false positive. Since the transmitters and receivers were highly directional, some simple realignment fixed all the problems. This data was then fed to the development board's LCD screen for simple raw data analysis. However the code to drive the screen drastically reduced the performance of the system, so it was scrapped as soon as possible.

The next step in this process was to get the USB communication between the board and the computer working. Fortunately, the Texas Instruments demo program had a basic system setup to send and receive data. However, the bit rate on this was set so low that it was slowing down the overall system performance. Due to a problem in the documentation of the code from TI, it took several hours to rewrite the clock to run faster as the math they had used was incorrect and produced an invalid and uneven clock signal. Once this problem was remedied, the output of raw data could be seen in a terminal view.

These paved the way for the final phase of refining the touch detection algorithm and optimize code to squeeze extra speed out of the system. Several revisions were made and sometimes undone to get this working correctly. In hindsight, some sort of check in system

would be best to use next time since it makes it easy to view changes. A final version of the code can be found in the appendix.

Final Thoughts and Lessons Learned

If I had to do the project over again, I'd make some changes. First and foremost, I would definitely take on a partner or two. Due to the start date of the project, having a partner wasn't going to be possible because of scheduling conflicts. This meant that there was a large amount of work that was to be done by just 1 person and I think a better job and more creativity would come from a group environment. A CS major would have also been very helpful. When it came down to it, my windows programing skills are just nonexistent. It would have been nice to get this project working with actual graphical demos instead of just text displays.

Extra attention needed to be paid to the board layout. Using the design import functions in UltiBoard combined with the automatic layout features meant that little things could be overlooked. Because of this, the comparator was wired backwards and didn't end up giving me the extra sensitivity that I required. And the holes in the boards were too small, so I need to make sure that I give components more clearance the next time I do layout.

I have learned a lot from this project. It did seem like I bit off more than I could chew, but I am ultimately satisfied by the results. I gained a large amount of experience in design and

implementation as well as learning to rely on only myself and work independently. The project was a success in my book and that's what counts the most.

Conclusion

Overall, the project was a success. The system was able to detect and track multiple (more than one) touches on the screen at a given time. The frame was designed in a method that is expandable and reproducible and it works with existing surfaces. One could even use this on non-screen surfaces, but it seems as though this would be an overly expensive and complicated solution.

By starting with initial planning and research, continuing through design and testing, and resulting in a working prototype, this MQP has been an accurate representation of my understanding of the total design process. This project, at the culmination of my educational experience at Worcester Polytechnic Institute, gave me a proper understanding of real world design situations and I will take the knowledge learned to a future job.

Appendix A - Code

```
//*****  
**  
// MSP-EXP430F5438 Experimenter's Board - User Experience Demo  
//  
// Main Project File: UserExperience.c  
//  
// D. Dang  
// Texas Instruments Inc.  
// Ver 1.00 - May 2008  
// Built with Code Composer Essentials CCE v3.2  
//  
// Modified and improved upon by Sean Thulin for MQP  
//  
// This program is free software: you can redistribute it and/or modify  
// it under the terms of the GNU General Public License as published by  
// the Free Software Foundation, either version 3 of the License, or  
// (at your option) any later version.  
//  
// This program is distributed in the hope that it will be useful,  
// but WITHOUT ANY WARRANTY; without even the implied warranty of  
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
// GNU General Public License for more details.  
//  
// You should have received a copy of the GNU General Public License  
// along with this program. If not, see <http://www.gnu.org/licenses/>.  
//*****  
**  
#include <msp430x54x.h>  
#include "MSP-EXP430F5438 HAL\hal_MSP-EXP430F5438.h"  
#include "UserExperienceDemo\UserExperienceGraphics.h"  
#include "UserExperienceDemo\flashUtils.h"  
  
//--Calibration constants and user configuration values stored in INFOB Flash  
// at production; --  
__no_init unsigned char boardMode @ 0x1900;  
__no_init unsigned char lcdBackLightLevelSetting @ 0x1908;  
__no_init unsigned long lastAudioByteFlash @ 0x1940;  
__no_init unsigned char temperatureConversion @ 0x1928;  
__no_init long temperatureCalibrationC @ 0x1920;  
__no_init unsigned char wakeUpOnAcc @ 0x1978;  
__no_init unsigned char lcdContrastSetting @ 0x1910;  
__no_init int Acc_X_Calibrated_Offset @ 0x1950;  
__no_init int Acc_Y_Calibrated_Offset @ 0x1960;  
__no_init int Acc_Z_Calibrated_Offset @ 0x1970;  
  
#define TIME_OUT 10  
#define TIME_OUT2 3  
#define MENU_MAX 6  
#define SETTING_MENU_MAX 6  
#define MENU_ITEM_WIDTH 14  
  
enum{ LPM4_MODE, LPM3_MODE, ACTIVE_MODE, APPLICATION_MODE } ;  
enum{ APP_CLOCK, APP_BAL_BALL, APP_USB, APP_AUDIO, PMM_MCLK, MENU_SETTING};
```

```

enum{  SET_TIME, SET_CONTRAST, SET_BACKLIGHT, SET_TEMPERATURE_CONVERSION,
CONFIGURE_ACCELEROMETER, EXIT_SETTING};

unsigned char boardModeLOCAL;
int Acc_X_Calibrated_OffsetLOCAL;
int Acc_Y_Calibrated_OffsetLOCAL;
int Acc_Z_Calibrated_OffsetLOCAL;
unsigned char lcdBackLightLevelSettingLOCAL;
unsigned char lcdContrastSettingLOCAL;
unsigned char temperatureConversionLOCAL;
long temperatureCalibrationCLOCAL;
unsigned char wakeUpOnAccLOCAL;

/*static char menuText[]={
"MSP-EXP430F5438\0"
" 1. Clock \0"
" 2. UniBall \0"
" 3. USB-UART \0"
" 4. Voice Rec\0"
" 5. PMM-MCLK \0"
" 6. Settings \0"
}; */

unsigned char menuPos, settingMenuPos;
unsigned char timeOutCounter = 0;
unsigned char CpuMode, accWake=0, menuPos, settingMenuPos;

volatile unsigned char buttonsPressed;
volatile unsigned char buttonDebounce;

char TemperatureStr[]= "\0\0\0\0\0\0";
char VccStr[] = "0.0V";
unsigned char RTCAccHalfSec = 0, RTCExit64Hz= 0 , RTCExitSec= 0, RTCAccSec =
0;

#include "UserExperienceDemo/clock.c"
#include "UserExperienceDemo/balanceBall.c"
#include "UserExperienceDemo/usbTest.c"
#include "UserExperienceDemo/audio.c"
#include "UserExperienceDemo/menuSetting.c"
#include "UserExperienceDemo/PMM.c"

void setupRTC();

*****/**/
* @brief Checks for the board revision and returns a value < 0 if wrong
*        revision is specified in main.c
*
* @param none
*
* @return Whether or not the board revision matches the software
* - 0 - The board revision does not match the software
* - 1 - The board revision matches the software
*****/
unsigned char assert_board_version( void )
{
    P8DIR &= ~BIT7;                                // Set P8.7 input

```

```

P8OUT |= BIT7;                                // Set pullup resistor
P8REN |= BIT7;                                // Enable pull up resistors

#ifdef REV_02
    if (!(P8IN & BIT7))                      // Board rev = 0_02?
        return 0;
#else
    if ((P8IN & BIT7))                        // Board rev = 0_03?
        return 0;
#endif

P8DIR |= BIT7;                                // Set P8.7 output
P8OUT &= ~BIT7;                               // Set P8.7 = 0
P8REN &= ~BIT7;                               // Disable pull up resistors

return 1;
}

/***************** swDelay() *****/
void swDelay(unsigned int max_cnt)
{
    unsigned int cnt1=0, cnt2;

    while (cnt1 < max_cnt)
    {
        cnt2 = 0;
        while (cnt2 < 125)
            cnt2++;
        cnt1++;
    }
}

/***************** abs() *****/
int abs (int i) {
    return i < 0 ? -i : i;
}

/***************** lowPowerMode3() *****/
* @brief Enters LPM3 and waits for either accelerometer tilt or a button
*        tilt
*        to activate the board.
*
* @param none
*
* @return none
*/
void lowPowerMode3(void)
{
    int accX, accY, accZ;

    CpuMode = LPM3_MODE;

    halLcdClearScreen();
    halLcdImage(TI_BUG, 14, 106, 10, 0);

    halLcdSetBackLight(0);
}

```

```

    halLcdStandby();

    accWake = 0;
    if (wakeUpOnAcc)
    {
        RTCCTL0 |= RTCRDYIE;                                //Enable interrupt
        RTCAccSec = 1;
        halAccelerometerInit();
        halAdcSetQuitFromISR( 1 );
    }

    halBoardSetSystemClock( SYSCLK_12MHZ );

    /* Either a button press or an RTC interrupt will wake the CPU
     * from LPM3 mode. The RTC interrupt will periodically enable and
     * re-initialize the accelerometer to see if the user has registered
     * a change in the tilt that is greater than the accelerometer thresholds.
     * If so, accWake is set and the board is activated
    */
    do
    {
        halAccelerometerShutDown();

        __bis_SR_register(LPM3_bits + GIE);      // Enter LPM3
        __no_operation();                         // For debugger only

        if (!buttonsPressed)
        {
            halAccelerometerRead( &accX, &accY, &accZ );
            accWake = ( accX > ACC_X_THRESHOLD || accX < -ACC_X_THRESHOLD ||
                        accY > ACC_Y_THRESHOLD || accY < -ACC_Y_THRESHOLD );
        }
    }
    while (accWake == 0 && buttonsPressed == 0);

    halBoardSetSystemClock( SYSCLK_16MHZ );

    RTCCTL0 |= RTCRDYIE;                                //Enable interrupt
    RTCAccSec = 0;
    halAccelerometerShutDown();

    halLcdInit();
    halLcdInit();
    halLcdClearScreen();
    halLcdSetBackLight(lcdBackLightLevelSettingLOCAL );
    CpuMode = ACTIVE_MODE;
}

// ----- Active Mode with Menu-----
-

/********************************************/ /**
* @brief Enters LPM3 and waits for either accelerometer tilt or a button
tilt
*       to activate the board.
*
* @param menuText The text that constitutes the application menu.

```

```

/*
* @param menuPos The line of the current menu option.
*
* @param change
*
* - 0 - Move menu selection up
* - 1 - Move menu selection down
*
* @param menuNum The enumerated value that represents the current
*                 menu selection.
*
* @return none
***** */

void menuUpdate(char *menuText, unsigned char *menuPos, int change,
                unsigned char menuNum)
{
    halLcdPrintLine(&menuText[*menuPos*MENU_ITEM_WIDTH+16],
                    (*menuPos)+1, OVERWRITE_TEXT);

    if (change == 0)                                //Subtract
    {
        if (*menuPos > 0)
            (*menuPos)--;
        else
            *menuPos = (menuNum - 1);
    }
    else
    {
        if ((*menuPos) < menuNum - 1)
            (*menuPos)++;
        else
            *menuPos = 0;
    }

    halLcdPrintLine(&menuText[*menuPos*MENU_ITEM_WIDTH+16],
                    (*menuPos)+1, INVERT_TEXT | OVERWRITE_TEXT);
}

/** */
* @brief Draws and manages the selection of the menu options.
*
* @param menuText The text that constitutes the application menu.
*
* @param menuNum The enumerated value that represents the current
*                 menu selection.
*
* @return The updated, or latest, menu selection.
***** */

unsigned char activeMenuMode(char *menuText, unsigned char menuNum)
{
    unsigned char menuPosition, quit = 0;
    int i;

    halAccelerometerShutDown();
    halAdcInitTempVcc();
    RTCExitSec = 1;                                // To update digital clock

```

```

halButtonsInterruptEnable( BUTTON_ALL );

halLcdClearScreen();
halLcdImage(TI_TINY_BUG, 4, 32, 104, 12 );
menuPosition = 0;
//Print menu title
halLcdPrintLine(menuText, 0, 0 );

//Print menu items
for (i=1;i<menuNum;i++)
    halLcdPrintLine(&menuText[i*MENU_ITEM_WIDTH+16], i+1, OVERWRITE_TEXT );
//First line is inverted text, automatic selection
halLcdPrintLine(&menuText[0*MENU_ITEM_WIDTH+16], 1, \
    INVERT_TEXT | OVERWRITE_TEXT );
timeOutCounter = 0;
buttonsPressed = 0;

halAdcSetQuitFromISR( 0 );

while (CpuMode == ACTIVE_MODE && quit == 0)
{
    TA0CTL &= ~TAIFG;

    __bis_SR_register(LPM3_bits + GIE);           //Returns if button pressed or
clock ticks
    // __no_operation();                           // For debugger only

    if (buttonsPressed)
    {
        switch (buttonsPressed)
        {
            case BUTTON_UP:          menuUpdate(menuText, &menuPosition, 0, menuNum);
break;
            case BUTTON_DOWN:        menuUpdate(menuText, &menuPosition, 1, menuNum);
break;
            case BUTTON_SELECT:      CpuMode = APPLICATION_MODE; break;
            case BUTTON_S2:          CpuMode = APPLICATION_MODE; break;
            case BUTTON_S1:          quit = 1;
            default: break;
        }
        timeOutCounter = 0;
    }
    else
        //if no button pressed --> clock
ticks
    {
        halAdcStartRead();
        digitalClockDraw();
        halAdcReadTempVcc(TemperatureStr, VccStr);
        halLcdPrintLineCol(TemperatureStr, 7, 12, OVERWRITE_TEXT);
        halLcdPrintLineCol(VccStr, 8, 12, OVERWRITE_TEXT);
        if (++timeOutCounter > TIME_OUT)
            CpuMode = LPM3_MODE;
    }
    buttonsPressed = 0;
}
RTCExitSec = 0;
halAdcShutDownTempVcc();

```

```

    return menuPosition;
}

/****************************************************************************************************/ /**
* @brief This is the example code's main function.
*
* @param none
*
* @return none
*****************************************************************/
int MQPSETUP()
{
    /* Check for the version of the board */
    if(!assert_board_version())
        while(1);

    //Initialize clock and peripherals
    halBoardInit();
    halBoardStartXT1();
    halBoardSetSystemClock(SYSCLK_16MHZ);

    loadSettings();

    //Initialize buttons
    buttonDebounce = 1;
    halButtonsInit( BUTTON_ALL );
    halButtonsInterruptEnable( BUTTON_ALL );

    bis_SR_register(LPM3_bits + GIE);           // Enter LPM3
    //__no_operation();                         // For debugging only

    CpuMode = ACTIVE_MODE;
    //setupRTC();

    //variables used for detecting finger positions
    unsigned long tempinput = 0x0000;
    unsigned long tempinput2 = 0x0000;

    unsigned long input2[] = { 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000 };

    unsigned long tempa = 0x0000;
    unsigned long tempb = 0x0000;

    int x_min = 0;
    int x_max = 0;
    int x1 = 0;
    int x2 = 0;
    int x_old = 0;
    int xlr = 0;
    int y_min = 0;
    int y_max = 0;
    int y1 = 0;
    int y2 = 0;
    int y_old = 0;

```

```

int ylr = 0;
int r = 0;
int r_old = 0;
halUsbInit(); // start USB connection

while (1) // main loop
{
    //reset detection storage variable to 0
    for (int g = 0; g < 8; g++) {
        input2[g] = 0x0000;
    }
    // P4 BIT Outputs: ADDR1 - ADDR2 - ADDR3 - ADDR4 - NONE- CS1 - CS2
    P4DIR |= (BIT0|BIT1|BIT2|BIT3|BIT4|BIT5|BIT6); // P4.0 Decoder control
output
    P5DIR |= (BIT0); // clk
    P5SEL &= ~BIT0;
    P3DIR = 0;
    P3SEL = 0;
    P7DIR = 0;
    P7SEL = 0;
    P4SEL &= ~(BIT0|BIT1|BIT2|BIT3|BIT4|BIT5|BIT6);

    // Loop through decoder forever
    while(1) {
        for (int k = 0; k < 2; k++) { // Chip Select 0 or 1
            if (k == 0) { // Chip 0
                P4OUT &= ~BIT5;
                P4OUT |= BIT6;
            }
            else { // Chip 1
                P4OUT &= ~BIT6;
                P4OUT |= BIT5;
            }
            P4OUT &= (BIT6|BIT5); // clear all ADDR lines
            swDelay(1);
            for (int i = 0; i < 14; i++) { // loop 14 outputs of decoder chip
                P5OUT += 1; // CLK high
                swDelay(1);
                P5OUT -= 1; // CLK low
                swDelay(1);
                P4OUT += 1; // shift up to next segment
                swDelay(1);

                tempinput = P3IN; // read in part of the scan
                tempinput >>= 4; // shift it over to align it properly
                tempinput &= 0x000F; // zero out the rest that we don't care about
                tempinput2 = P7IN; // read in the other half of the scan
                tempinput2 &= 0x00F0; // zero out what we don't want
                tempinput |= tempinput2; // add them together to get 1 long reading
                tempinput ^= 0x00FF; // and zero out the rest

                // append the first byte and invert order
                for (int r = 0; r < 8; r++) {
                    input2[r] <= 1;
                    input2[r] &= 0xFFFFFFFF;
                    input2[r] += (tempinput & 0x00000001);

```

```

        tempinput >>= 1;
    }
} // loop back to top to switch decoder chips
} // At this point all 28 segments should be cycled

// start output for terminal readout
halUsbSendChar(27);
halUsbSendString("[2J", 3);
halUsbSendChar(27);
halUsbSendString("[1;1H", 5);

// clear variables incase they don't get overwritten
x_min = 0;
x_max = 0;
x2 = 0;
y_min = 0;
y_max = 0;
y2 = 0;

// loop for the 6 boards that we have
// this would be changed in a later version to make it user assigned
for (int z = 0; z < 6; z++) {
    tempa = input2[z]; // read in first board
    if (tempa != 0) { // do we have any hits?
        tempb = 0;
        for (int x = 0; x < 28; x++) { // invert the order of the bits
            tempb <= 1;
            tempb += (tempa & 0x00000001);
            tempa >>= 1;
        }
        // for each bit, check to see what position it is
        for (int x = 0; x < 28; x++) {

            if (((tempb & 0x00000001) == 1) {
                if ((x + (z*28)) > 111) { // if it's greater than 111, then it
must be a y
                    if( y_min == 0 ) { // check for the first hit or "edge"
                        y_min = (x + (z*28)-111);
                    }
                    if ( y_max < (x + (z*28)-111) ) { // check for the last hit
or "edge"
                        y_max = (x + (z*28)-111);
                    }
                } else {

                    if( x_min == 0 ) { // check for the first hit or "edge"
                        x_min = (x + (z*28));
                    }
                    if ( x_max < (x + (z*28)) ) { // check for the last hit or
"edge"
                        x_max = (x + (z*28));
                    }
                }
            }
            tempb >>= 1; // shift over 1 before starting again
        }
    }
}

```

```

        }

    // average the diameters to get the radius
    y1r = (y_max - y_min) / 2;
    x1r = (x_max - x_min) / 2;
    r = (y1r+x1r+1)/2; // + 1 is to make it round up instead of down

    if (r < 3) { // if the radius is smaller than 3, really only have 1
object
        // set midpoint to middle of triggers
        y1 = (y_min + y_max) / 2;
        x1 = (x_min + x_max) / 2;
        r_old = r;
    }
    // start of 2nd point detection
    else {
        //compare edges to see what is near old object and make 2nd object on
other edge
        if (abs(x_old - x_min) < 5) {
            x2 = (x_max - r_old);
            x1 = (x_old + x_min)/2;
        }
        else {
            x2 = (x_min + r_old);
            x1 = (x_old + x_max)/2;
        }
        if (abs(y_old - y_max) < 5) {
            y2 = (y_min + r_old);
            y1 = (y_old + y_max)/2;
        }
        else {
            y2 = (y_max - r_old);
            y1 = (y_old + y_min)/2;
        }
    }
    // output the x,y and radius of each point
    halUsbSendChar('(');
    if (x1 > 100) {
        halUsbSendChar('1');
    }
    else {
        halUsbSendChar('0');
    }
    halUsbSendChar((x1/10) + 48);
    halUsbSendChar((x1%10) + 48);
    halUsbSendString(",0",2);
    halUsbSendChar((y1/10) + 48);
    halUsbSendChar((y1%10) + 48);
    halUsbSendString(" R:",4);
    halUsbSendChar(r_old + 48);

    if ((x2 != 0) || (y2 != 0)) {
        halUsbSendChar(27);
        halUsbSendString("[2;1f", 5);
        halUsbSendChar(')');
        if (x2 > 100) {

```

```

        halUsbSendChar('1');
    }
    else {
        halUsbSendChar('0');
    }
    // output 2nd line
    halUsbSendChar((x2/10) + 48);
    halUsbSendChar((x2%10) + 48);
    halUsbSendString(",0",2);
    halUsbSendChar((y2/10) + 48);
    halUsbSendChar((y2%10) + 48);
    halUsbSendString(") R:",4);
    halUsbSendChar(r_old + 48);
}
x_old = x1;
y_old = y1;
}
}

//****************************************************************************
* @brief  Initializes the RTC calendar.
*
* Initial values are January 01, 2009, 12:30:05
*
* @param  none
*
* @return none
//****************************************************************************
void setupRTC(void)
{
    RTCCTL01 = RTCMODE + RTCBCD + RTCHOLD + RTCTEV_1;

    SetRTCHOUR(0x12);
    SetRTCMIN(0x30);
    SetRTCSEC(0x05);
    SetRTCDAY(0x01);
    SetRTCMON(0x01);
    SetRTCYEAR(0x2009);

    RTCCTL01 &= ~RTCHOLD;

    RTCPS1CTL = RT1IP_5;                                // Interrupt freq: 2Hz
    RTCPS0CTL = RT0IP_7;                                // Interrupt freq: 128hz

    RTCCTL0 |= RTCRDYIE + RTCTEVIE;                     // Enable interrupt
}

//****************************************************************************
* @brief  Sets up the WDT as a button debouncer, only activated once a
*         button interrupt has occurred.
*
* @param  none
*
* @return none
//****************************************************************************
void startWDT()

```

```

{
    //WDT as 250ms interval counter
    SFRIFG1 &= ~WDTIFG;
    WDTCTL = WDTPW + WDTSSEL_1 + WDTCNTCL + WDTIS_5;
    SFRIE1 |= WDTIE;
}

/*********************************************
#pragma vector=RTC_VECTOR
__interrupt void RTC_ISR(void)
{
    static unsigned char counter=0;
    switch (RTCIV)
    {
        case 0x02: if (RTCExitSec == 1)
            __bic_SR_register_on_exit(LPM3_bits);

        if (RTCAccSec == 1)
        {
            halAccelerometerInit();
            halAdcStartRead();
        }
        break;
        case 0x04: break;
        case 0x08: if (RTCExit64Hz == 1)
            if (++counter == 4)
            {
                counter = 0;
                __bic_SR_register_on_exit(LPM3_bits);
            }
            break;
        case 0x0A: if (RTCAccHalfSec == 1)
        {
            ADC12IFG = 0;
            ADC12CTL0 |= ADC12ENC | ADC12SC;
        }
        break;
    }
    RTCCTL0 &= ~RTCRDYIFG;
}

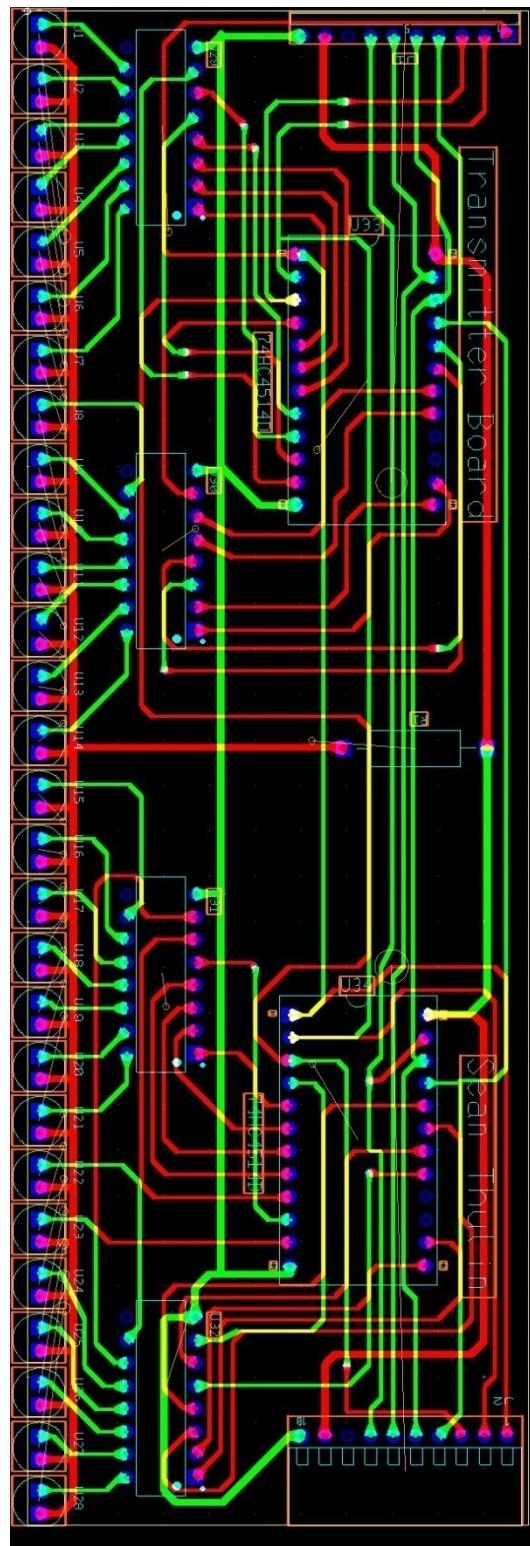
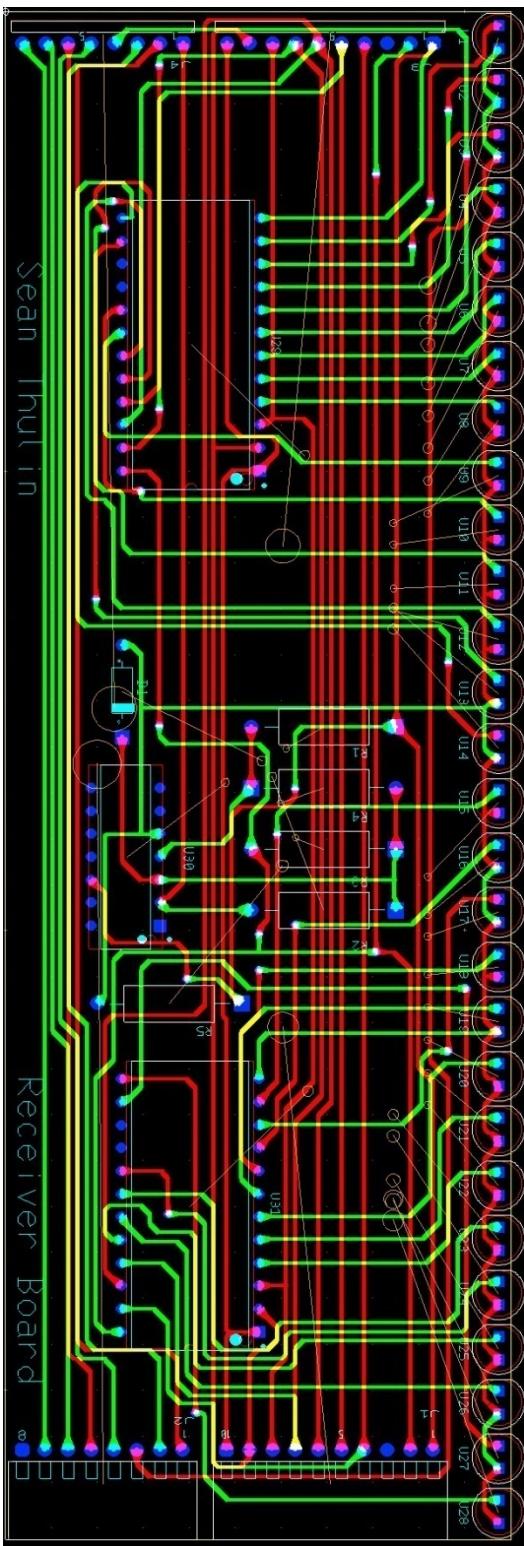
#pragma vector=WDT_VECTOR
__interrupt void WDT_ISR(void)
{
    if (buttonDebounce == 2)
    {
        buttonDebounce = 1;
        SFRIFG1 &= ~WDTIFG;
        SFRIE1 &= ~WDTIE;
        WDTCTL = WDTPW + WDTHOLD;
    }
}

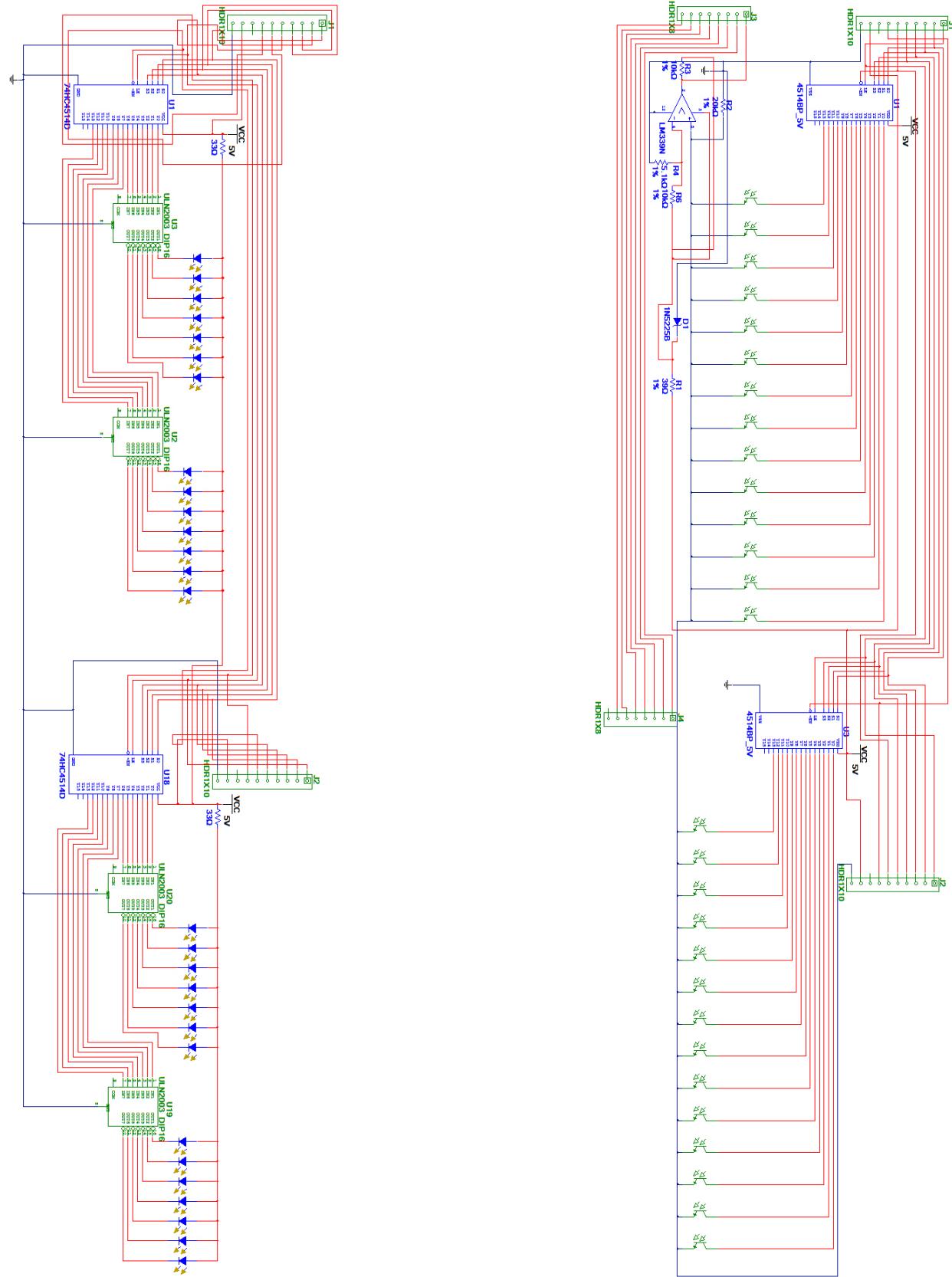
#pragma vector=PORT2_VECTOR
__interrupt void Port2_ISR(void)
{
}

```

```
if (buttonDebounce == 1)
{
    buttonsPressed = P2IFG;
    buttonDebounce = 2;
    startWDT();
    __bic_SR_register_on_exit(LPM3_bits);
}
else if (0 == buttonDebounce)
{
    buttonsPressed = P2IFG;
    __bic_SR_register_on_exit(LPM4_bits);
}
```

Appendix B – Layouts and Schematics





Appendix C – Total Cost

Initial product purchase for testing = \$28.35

PCB manufacturing: $\$33 \times 4 = \265.46

TI MSP-EXP430F5438: \$0

Phototransistor QSD-122: $\$0.20 \times 224 = \44.80

LED QED-234: $\$0.14 \times 224 = \31.36

RA Header Socket 575-502001: $\$10.22 \times 5 = \51.10

RA Pin Header 575-642001: $\$10.29 \times 4 = \41.16

Darlington Array ULN2003A: $\$0.235 \times 32 = \7.52

Decoder CDN4514BE: $\$0.95 \times 32 = \30.4

Zener 1N5225: $\$0.03 \times 8 = \0.48

Comparator LM339N: $\$0.27 \times 8 = \2.16

33 Ohm Resistor CCF60: $\$0.06 \times 8 = \0.48

39 Ohm Resistor CCF60: $\$0.06 \times 8 = \0.48

5.1K Ohm Resistor CCF60: $\$0.06 \times 8 = \0.48

10 Ohm Resistor CCF60: $\$0.06 \times 16 = \0.96

20K Ohm Resistor CCF60: $\$0.06 \times 8 = \0.48

Total: \$505.67

Appendix D – Bibliography

A D Metro Touch Screen Primer: http://www.admetro.com/pdfs/Touch_Screen_101.pdf

Advanced Circuits: <http://www.4pcb.com/>

Digikey Catalog: <http://dkc1.digikey.com/US/EN/PDF/US2010/Complete.html>

Fairchild QED234 datasheet: <http://www.fairchildsemi.com/ds/QE%2FQED234.pdf>

Fairchild QSD122 datasheet: <http://www.fairchildsemi.com/ds/QS%2FQSD124.pdf>

Mouser Electronics Catalog: <http://www.mouser.com/catalogviewer.aspx>

Multi-Touch Surfaces: A Technical Guide: <http://mi-lab.org/files/2009/10/bymultitouch.pdf>

Natural User Interface Group: <http://nuigroup.com/forums>

Texas Instruments MSP-EXP430F5438: <http://focus.ti.com/docs/toolsw/folders/print/msp-exp430f5438.html>

Texas Instruments CD74HC4515 datasheet: <http://focus.ti.com/lit/ds/symlink/cd74hc4515.pdf>

Tyco Technologies - A Survey of Mainstream and Emerging Touch Technologies: http://www.sidchapters.org/pacificnorthwest/meetings/dec03_08_presentation.pdf