4-2017

Educational LED Board

Nathan Ericksen
Montana Tech of the University of Montana

Tyana Rasmusan
Montana Tech of the University of Montana

Ashtyn Aumueller
Montana Tech of the University of Montana

Recommended Citation

Advisor: Bryce Hill

Nathan Ericksen
Tyana Rasmusan
Ashtyn Aumueller
Senior Design 2016-2017
Mentor: Bryce Hill

Educational LED Board

Introduction

The goal of this project is to develop a programmable LED board to be used for educational purposes and encourage students to pursue degrees in the science, technology, engineering, and mathematics (S.T.E.M.) fields. When the board is complete, it will be brought into middle and high school classrooms for use as an interactive activity to demonstrate the basics of memory and programming. When students are finished programming their boards, they can mount it on a stick and swing it in circles to display their message in the air. The goal is to make the average production cost of each board under $5 so that the students can take them home and demonstrate the project to their friends and parents.

The LED board will communicate with a user interface that will allow students to program the LEDs to light up in a desired design with dipswitches. The user interface features four pushbuttons, a bank of eight dipswitches, a 3-D printed case, and an LCD display. For each "bit" the students program into the LED board their selection will be displayed on an LCD display. Eventually after programming in their select word or character the LCD will display their whole series. After the student moves from each memory location to the next, the sequence will be saved into the LED board for when it is mounted. The LED board will be mounted on a 3D-printed box attached to a piece of PVC pipe. After being disconnected from the user interface, the LED board will be powered with AA batteries. The teacher will receive a user interface to keep in the classroom for future use.

Design

Our design consists of a user interface, an LED board, and a mounting system for the LED board. The user interface was designed to be as simple and easy to understand as possible. It consists of eight dipswitches, an LCD screen, and four pushbuttons. The dipswitches are used to turn on the LEDs of the LED board. The LCD screen displays which bits or LEDs will be turned on and the current memory location. The green pushbutton accepts the data and sends it to the LED board's random access memory (RAM), as well as interrupting the LED display sequence when held for approximately three seconds. The red pushbutton sends a command to the LED board to move the ram data to the flash memory as well as the current memory location that determines the length of the design that will displayed. The red pushbutton also starts the LED display sequence. The two black buttons allow the student to increment and decrement the memory location. Although the flash memory on the microcontroller is able to store thousands of bytes, we have limited storage to just 64 bytes. The PCB for the user interface was built using ExpressPCB and is shown in Appendix A. This PCB includes a 330Ω bias resistor used by the LCD screen, pins that will be used to program the LED board, a 3.3V regulator, an oscillator, and both USB and barrel jack connections. A cover and box for the user interface was designed in SolidWorks and then 3-D printed. This design can be seen in Appendix D.

The LED board consists of a one square inch PCB, shown in Appendix B. This board has eight LEDs along one side. Each of these LEDs has a 10kΩ pull-up resistor. The LED boards have programming pins to connect to and receive data from the user interface. The board PCBs also have places for female sockets and male pins headers so that in future project development, multiple boards can be docked together to expand student design possibilities.

The LED board will be mounted and spun on a rotating platform so that the student's final design can be seen. The mounting system was initially designed in AutoCAD and then built in SolidWorks to be 3-D printed. These designs can be seen in Appendix E. The 3-D printed box has a slot to hold a battery clip that will be used to power the LED board after it is disconnected from the user interface. The box has a hole for a piece of PVC pipe to be inserted and capped in place on either side. The LED board will be attached on top of the flat portion of the box to be spun.

Pinout

Both the user interface and LED board use a 28-pin Texas Instruments MSP430G2553IPW28 microcontroller. The pinouts for both the user interface and the LED board are given in Appendix C. The pinouts had to be chosen based on pins needed for communication between the two boards and ease of use. On the user interface microcontroller, two pins are required for USB to serial communication. Two pins are also required for I^2C communication. The LCD screen requires four pins. All of port two is dedicated to the eight dipswitches. The pushbuttons pins are on port 3. The LED board also uses two pins for I^2C, but has four extra pins dedicated to this for future development. All of port three on this microcontroller is dedicated to the LEDs. The microcontrollers on both boards have a pin dedicated to clock.

Financing and Components List

The biggest constraint on our design was component cost. Because an LED board will be given to each student in a classroom workshop, this board needed to be designed using the most cost-effective part choices. The user interface was designed to be reliable and simple, so parts for this board were chosen based on functionality. Fewer of these boards will be needed and only one will be given away per classroom.

As the boards were designed, a list of necessary components was compiled for each. Table 1 gives this list for the user interface. This list includes the part number for each component from the Digikey website, where parts were ordered from. The final cost of this board was determined to be $48.38.

Table 1. User Interface Parts List for One Board.

User Interface				
Item	**Part Number**	**Number Needed per Board**	**Price per Part**	**Total Cost**
8 Position Dipswitch	GH7177-ND	1	$1.47	$1.47
Black Pushbutton	EG4791-ND	2	$0.53	$1.06
Green Pushbutton	EG4793-ND	1	$0.53	$0.53
Red Pushbutton	EG4792-ND	1	$0.53	$0.53
LCD Display	NHD-C0216CZ-NSW-BBW-3V3-ND	1	$11.00	$11.00
3.3V Regulator	AZ1117CH-3.3TRG1DICT-ND	2	$0.38	$0.76
Pull Up Resistor - 10kΩ	311-10KJRCT-ND	4	$0.10	$0.40
USB to Serial	MCP2200-I/SS-ND	1	$1.94	$1.94
Barrel Jack	CP-202A-ND	1	$0.93	$0.93
Oscillator	490-7848-1-ND	1	$0.67	$0.67
USB Jack	609-4613-1-ND	1	$0.46	$0.46
Microcontroller	296-33466-5-ND	1	$2.63	$2.63
Female 6-Pin Socket	609-3558-ND	1	$0.90	$0.90
PCB	n/a	1	$25.00	$25.00
Bias Resistor - 330Ω	311-330JRCT-ND	1	$0.10	$0.10
			Total	$48.38

The goal for the LED Board was to make the final price of an individual board less five dollars. This would make it realistic to put ten to twenty boards into a classroom and allow students to take them home with them once programmed. The cost of one, 10, 100, and 1,000 boards were compared to give us an idea of how much a bulk order would save us. In the end, the cost for 1,000 boards was by far the cheapest. The price went down to $4.18, meeting the price goal for the LED board. The price for one board without bulk ordering would be $10.12, which is more than double our goal amount. Tables 2 through 5 show the necessary components for the LED board and cost for an order of the respective size.

Table 2. LED Board Parts List for One Board.

LED Board				
Item	**Part Number**	**Number Needed per Board**	**Price per Part**	**Total Cost**
Green LED	160-1131-ND	8	$0.27	$2.16
Microcontroller	296-33466-5-ND	1	$2.63	$2.63
10kΩ Resistor	311-10KJRCT-ND	8	$0.10	$0.80
Female 6-Pin Socket	609-3558-ND	1	$0.90	$0.90
Male Pin Heads	SAM8918-ND	1	$1.91	$1.91
Battery Clip	BC22AAW-ND	1	$0.99	$0.99
PCB	n/a	1	$0.73	$0.73
			Total	$10.12

Table 3. LED Board Parts List for 10 Boards.

LED Board - Bulk Order (10 Boards)					
Item	**Part Number**	**Number Needed per Board**	**Price per Part**	**Total Needed**	**Total Cost**
Green LED	160-1131-ND	8	$0.14	80	$11.58
Microcontroller	296-33466-5-ND	1	$2.36	10	$23.61
10kΩ Resistor	311-10KJRCT-ND	8	$0.0084	80	$0.67
Female 6-Pin Socket	609-3558-ND	1	$0.79	10	$7.92
Male Pin Heads	SAM8918-ND	1	$1.58	10	$15.84
Battery Clip	BC22AAW-ND	1	$0.92	10	$9.20
PCB	n/a	1	$0.73	10	$7.31
				Total	$76.13
				Cost per Board	$7.61

Table 4. LED Board Parts List for 100 Boards.

LED Board - Bulk Order (100 Boards)					
Item	**Part Number**	**Number Needed per Board**	**Price per Part**	**Total Needed**	**Total Cost**
Green LED	160-1131-ND	8	0.0621	800	$49.68
Microcontroller	296-33466-5-ND	1	$1.90	100	$189.78
10kΩ Resistor	311-10KJRCT-ND	8	$0.0029	800	$2.32
Female 6-Pin Socket	609-3558-ND	1	$0.68	100	$68.35
Male Pin Heads	SAM8918-ND	1	$1.19	100	$118.80
Battery Clip	BC22AAW-ND	1	$0.83	100	$83.00
PCB	n/a	1	$0.73	100	$73.08
				Total	$585.01
				Cost per Board	$5.85

Table 5. LED Board Parts List for 1,000 Boards.

LED Board - Bulk Order (1,000 Boards)					
Item	**Part Number**	**Number Needed per Board**	**Price per Part**	**Total Needed**	**Total Cost**
Green LED	160-1131-ND	8	$0.04	8000	$331.20
Microcontroller	296-33466-5-ND	1	$1.11	1000	$1,113.75
10kΩ Resistor	311-10KJRCT-ND	8	$0.00153	8000	$12.24
Female 6-Pin Socket	609-3558-ND	1	$0.50	1000	$497.12
Male Pin Heads	SAM8918-ND	1	$0.88	1000	$877.80
Battery Clip	BC22AAW-ND	1	$0.62	1000	$620.00
PCB	n/a	1	$0.73	1000	$730.83
				Total	$4,182.94
				Cost per Board	$4.18

Printed Circuit Board Design

The Printed Circuit Boards (PCB) were designed using the ExpressPCB software. The size constrains for the LED board were to keep the footprint under 1”x1”. This maximized the amount of LED boards that could be cut from one PCB wafer while also minimizing the cost. The user interface board needed to be built within a 2.8”x3.5” footprint so the budget ordering option could be used and keep ordering prices down. The most important component on both of the PCB’s is the MSP430 microcontroller, so the trace layout was designed around the pinout for the chip. On the user interface an intuitive design was desired so the through holes for buttons and the dipswitch were placed on the footprint first. Then the through hole layout for the LCD screen was placed on the footprint. All component pads were arranged for convenience in the future when traces are being laid out. Traces for ground and power were placed on the PCB footprint before any other traces to ensure no grounding or power errors on the microcontroller or floated inputs. Then the traces for all the microcontroller inputs were laid out to their respective buttons, switches, and pins. The trace building was taken with great care to avoid crossing traces and shorting any inputs.

The LED board was designed with a compact package in mind, for the design phase the footprint for the LED board was adjusted to 1”x1”. Pads for the I^2C pin header, pulling resistors, LED’s, and MSP430 microcontroller were all placed first. The traces to power and ground were placed first and connected into a loop around the perimeter of the board. Then traces for the LED’s were placed to allow more direct paths and prevent unintended crossing. The clock and slave pads for I^2C were connected as well to allow synchronicity between communicating boards.

Coding

This project involved writing code to light LEDs, use dipswitches and pushbuttons, light and display information on an LCD screen, save information to FLASH memory, and send data from master pins to slave pins using I^2C. The code for each of these separate tasks had to be integrated. The commented final version of this code is given in Appendices G and H. Code was first written that was able to turn on and off eight LEDs at a variable frequency. Next, code was written to turn these same LEDs on and off with dipswitches. The LCD screen was used to display both the dipswitch positions and the memory location. First the LCD screen had to be initilized and then the dipswitches were added onto the screen with a star character for an 'on' dipswitch and a blank for an 'off'. Next the values of 0 to 63 were added to show the position in the memory location to make this project functional. After the LCD screen was properly working, this code was integrated with previous code to display which LEDs had been turned on by the switches. I^2C is used to communicate between the two boards. The user interface is the master and the LED board is the slave. Both microcontrollers have an SDA and SLC connection in which the information is sent back and forth. The addresses for each of the boards were found and the activation values for reading and writing were tested. The final piece of code that was written individually was for FLASH memory. Our project needs to utilize this kind of storage so that the LED board will retain the data that was sent by the user interface after the two boards are disconnected. The MSP430Gxx microcontroller series has a very limited amount of RAM, however there is much more flash storage available. There are two downsides to flash memory though, it is much slower to read/write to and from than RAM and it can only read/written a few thousand times before it becomes unreliable. The biggest challenge is moving data from the RAM to flash. This can be done by having the microcontroller specifically index data to a certain memory location, e.g. 1140. Later on when the microcontroller wants to read the code back it will look at the memory location and start reading values until it is instructed to stop. Values for the memory location and what is going into the memory location are received from the user interface over I^2C. When the microcontroller senses any sort of I^2C data it automatically goes into write mode which will then overwrite any data in the desired memory location with the most current data. The microcontroller can then take the values it read back and use them to fill the LED array when the board is powered back on. The final integration of the coding was simple as it all worked in parts which made the troubleshooting for the small sections minimal.

Assembly

After the necessary components were received from Digikey, they had to be mounted on the PCB. On the user interface, the microcontroller, voltage regulator chips, pulling resistors, capacitors, oscillator, and USB to serial chip were soldered on and then the board was baked in the reflow oven. After the completion of this process, the LCD screen, buttons, dipswitches, USB jack, barrel jack, and programming pins were soldered on to the board by hand. The same technique was used with the LED board, with the microcontroller and pulling resistors being placed prior to baking and the LEDs, pin sockets, and pin headers being soldered on afterwards. The SolidWorks files for the user interface enclosure and mounting setup were transferred to the program MakerBot and then loaded on to an SD card. This card was plugged into the printer and the designs were 3-D printed. PVC pipe and stoppers were purchased at Ace Hardware. The user interface was placed in its box, the LCD board was glued to the rotating platform and the battery clip was secured inside. The PVC pipe was inserted into the hole in the rotation setup and stoppers were added to either end. Photos of the completed assemblies are in Appendix I.

Programming with the User Interface

Our design allows for 64 memory locations to be programmed. Each memory location contains a byte, or eight bits, of information. The student will use the design worksheet given in Appendix F to plan out their design. The worksheet shows a grid with 8 rows and 64 columns. The student will fill in boxes in the grid to create a design. They will then use the user interface to program their board. Starting with memory location 00, they will turn on the dipswitches indicated in the vertical columns on the worksheet. Once the column has been pressed into the dipswitches, the data is sent to the LED board using the green pushbutton. The green pushbutton also makes the LED board light up the corresponding pattern just programmed into the board. They will then move to the next memory location using the right pushbutton. If a mistake was made they can press the left black pushbutton to go to the previous memory location and reprogram. This will be repeated until data has been entered into each of the memory locations on the worksheet. To ensure that the information will remain on the LED board's microcontroller after it is disconnected from the user interface, it must be moved from the RAM to the flash memory. This is done using the red pushbutton. When the red button is pushed, it also sends the current memory location over to the led board to determine the length of the design that is displayed. The red pushbutton then starts the cycle for displaying the message programmed into the LED board. If revisions are desired, press and hold the green button for three seconds, which stops the pattern from displaying and puts the LED board in programming mode. When the LED board is disconnected from the user interface, connected to power from the batteries, and spun, the board will move through length of the design and light up the corresponding LEDs.

Problems and Troubleshooting

We encountered several problems while completing this project. Our design had to be adjusted throughout the year as issues arose. We initially planned for our red pushbutton to be used to clear all of the data from the memory locations. After realizing we needed to utilize the microcontroller's flash memory, we decided that it was more logical to use this button to commit the data to the flash memory.

Our PCBs required several revisions. Changes had to be made to ensure traces were adequately spaced and via holes were placed correctly. Our first complete user interface PCB design was missing several things. After receiving these boards, we realized that our dipswitches were not mapped properly and the pins for programming and the LCD screen were not spaced properly. We redesigned this board to fix these problems and also moved the barrel jack and programming pins to the back of the board for ease of use. On our first LED board PCB, the microcontroller test pin was not connected and this had to be corrected.

While assembling the PCBs, we had issues with building bridges while baking in the reflow oven. This means that the solder holding down one pin came in contact with another pin, shorting them together. These bridges were discovered during the coding phase and were corrected using solder wick to remove the excess solder.

The 3-D printed portions of our project also took several redesigns. We initially planned for both the battery clip and the LED board to be attached on top of the rotating platform. We decided to move the battery clip into the box to add weight at the far end of the box, allowing it to swing more easily. We also decided to build a cover for the bottom of this platform to ensure that the battery clip would stay inside. The first prints for both parts of the user interface box were incorrectly sized and the holes for the components were misaligned. After these holes were repositioned, the outside of the enclosure was adjusted to make sure the user interface was secure inside. A lip was added to the cover and the outside margins were increased slightly so it would fit tightly. While printing, we also experienced issues with the plastic warping, which caused us to reprint several times.

The majority of our problems came while coding. We began coding using the MRT Development board provided to us by our mentor, Bryce Hill, and MSP430G2553 microcontroller launch pads. The Development boards feature LEDs, pushbuttons, and dipswitches. We were able to get each of these components to work with some troubleshooting. Some issues were experienced when coding for the LCD screen such as displaying the right information in the right place on the screen. These issues were resolved using the data sheet for the LCD screen provided on DigiKey. Another issue for the LCD screen was adjusting the code so that the screen would only update the dipswitch and position values on the screen when that value had changed. Before this issue was solved, the screen blinked quickly which was distracting. This was easily fixed by adding a simple loop onto the existing code. The biggest issues came once we began coding for I^2C to communicate between our two boards. Most of this code was provided by Bryce Hill but the troubleshooting did prove to be difficult and tedious. The fixes to these problems, once determined, were very simple. Changes as simple as commenting out one line of code fixed all of the slave code for the LED board to communicate. This line that was taken out was forcing the I^2C flag value to zero after each execution, and therefore would not allow the values to reach the final destination. This code was also the most sensitive to even the smallest changes, so again troubleshooting took the longest for this part of the code. The last piece of code that was written individually was to use flash memory. The issue with using flash memory is how the microcontroller finds memory indexes and reads them. Initially, a value would be sent to a specific location but when read back it would be hex 0xFF. There is actually an offset on the microcontroller when it comes to reading memory locations, but this also comes with the issue of validating where the values are actually being written. The microcontroller can only read back memory in even numbered locations, so this actually doubled the amount of space in the flash memory required for a 64 character array. With some tweaking to the provided example code we were able to have the memory read and write back from the same locations without overwriting or reading back the default 0xFF value. Once all of the code was integrated for the final design to perform, small tweaks throughout the code were made to improve the quality and functionality for our final design.

One of the last design flaws that was discovered was that when programming the LED board, a memory location that is included in the design cannot be left empty. If a blank or all the LEDs off is wanted, a value of zero must be entered into that memory location. If a location is left empty, the program will not work properly and will show a random design that was not entered. On top of the random design, the green button, which is regularly used to break out of running mode, does not function properly. Therefore, a value must be entered into every location within the desired design length.

Potential Project Impacts

As the United States continues to grow and progress, careers in STEM are becoming increasingly important. According to the US Department of Education, the number of available STEM jobs will increase by 14% between 2010 and 2020 [1]. In comparison, the US Department of commerce reports that other fields are growing at 9.8% [2]. This growth must be met with growth in the number of qualified STEM employees. Adecco Engineering and Technology has found that by 2018 there by be 2.4 million unfilled positions in STEM industries [3]. This is where STEM education comes into play. Exposing students to the STEM may get them interested in pursuing a career in this field. Thousands of careers fall under the STEM umbrella, creating possibilities for students of many different backgrounds and interests. We hope that hands-on learning, like what will be done with our project, will show students a side of science, technology, engineering, and mathematics that they may not have known about before and will spark an interest in learning more. To be continued…¯_(ツ)_/¯

References

[1]. “Science, Technology, Engineering and Math: Education for Global Leadership,” *Science, Technology, Engineering and Math: Education for Global Leadership | U.S. Department of Education*. [Online]. Available: http://www.ed.gov/stem. [Accessed: 20-Nov-2016].

[2]. “Why Is STEM Education So Important,” *Engineering For Kids*. [Online]. Available: http://engineeringforkids.com/article/02-02-2016_importanceofstem. [Accessed: 21-Mar-2017].

[3]. “Infographic: STEM Skills Are Driving Innovation,” *Infographic: STEM Skills Are Driving Innovation*, 04-Oct-2016. [Online]. Available: https://www.adeccousa.com/employers/resources/infographic-stem-skills-are-driving-innovation/. [Accessed: 20-Nov-2016].

Appendix A. *User Interface PCB Design.*

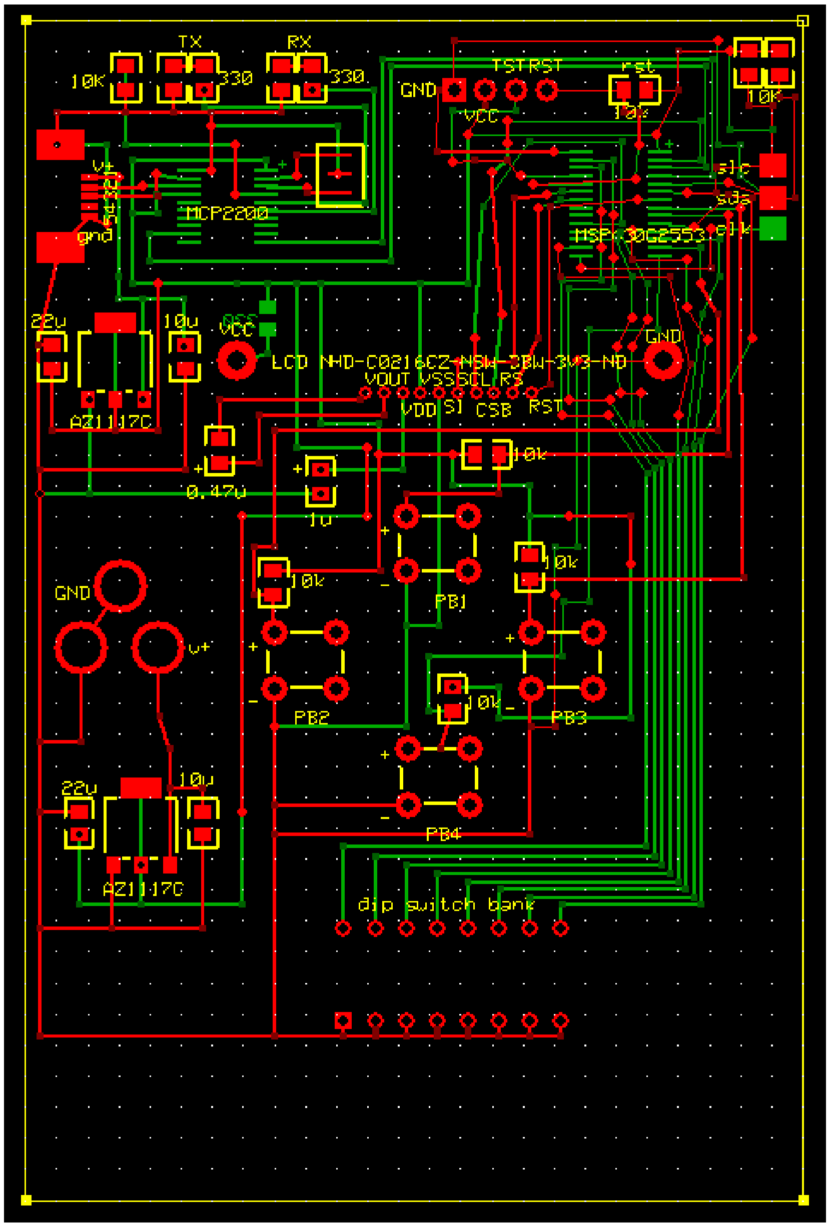

Figure 1A. User Interface PCB.

Appendix B. *LED Board PCB Design.*

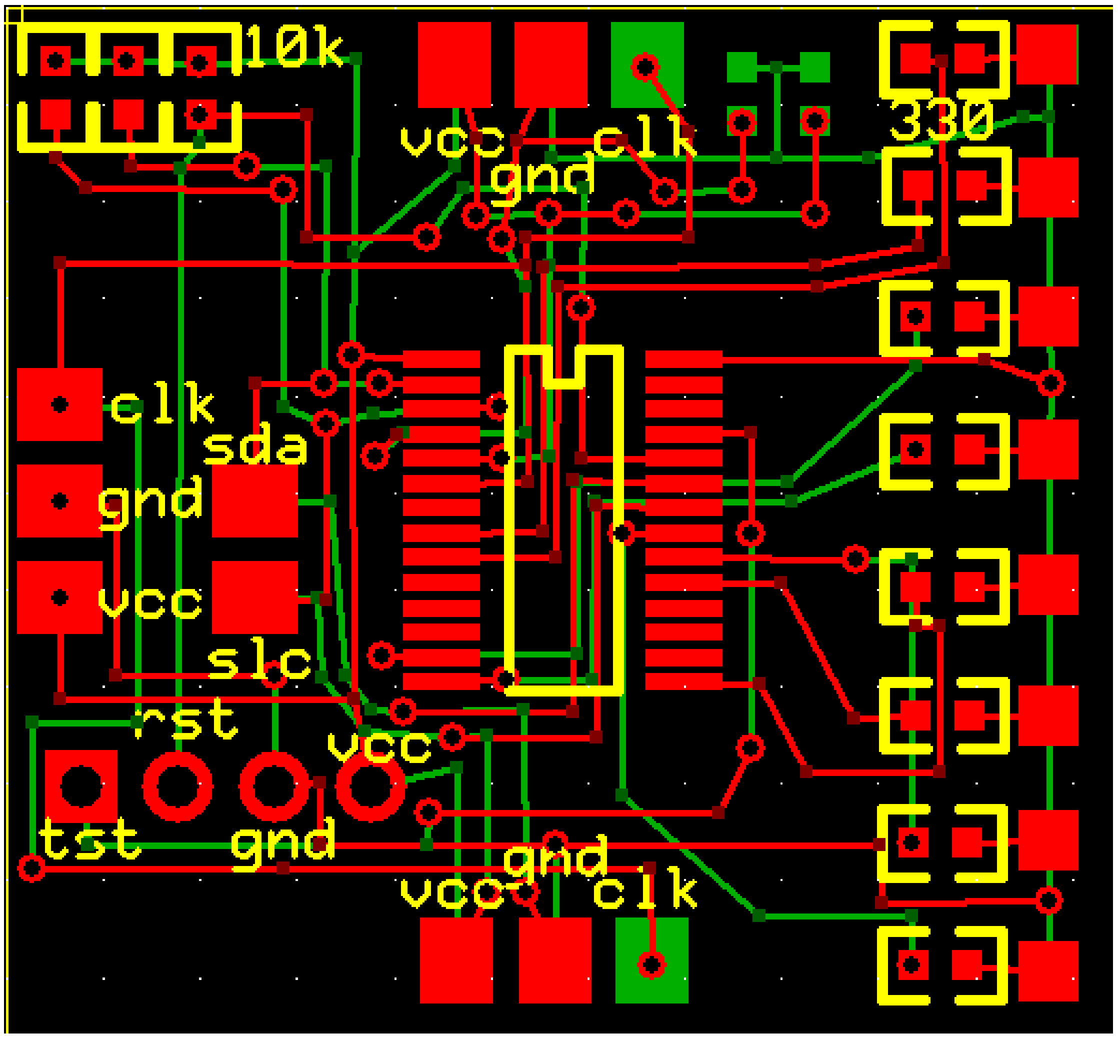

Figure 1B. LED Board PCB.

Appendix C. *Microcontroller Pinouts.*

Table 1C. User Interface Pinout.

USER INTERFACE: 28-PIN MICROCONTROLLER			
DVCC	(+)	DVSS	0
P1.0	LCD-CS	XIN/P2.6	DIPSWITCH-2
P1.1	TX(USB TO UART)	XOUT/P2.7	DIPSWITCH-1
P1.2	RX(USB TO UART)	TEST	(to programming pins)
P1.3	SDA(MASTER)	RST	(pullup)
P1.4	SLC(MASTER)	P1.7	LCD - MOSI
P1.5	LCD-RST	P1.6	LCD -RS
P3.1	BUTTON-RIGHT	P3.7	
P3.0	BUTTON-RED	P3.6	
P2.0	DIPSWITCH-8	P3.5	LCD-SCL
P2.1	DIPSWITCH-7	P2.5	DIPSWITCH-3
P2.2	DIPSWITCH-6	P2.4	DIPSWITCH-4
P3.2	BUTTON-LEFT	P2.3	DIPSWITCH-5
P3.3	BUTTON-GREEN	P3.4	GLOBAL CLK

Table 2C. LED Board Pinout.

LED BOARD: 28-PIN MICROCONTROLLER			
DVCC	(+)	DVSS	0
P1.0	SDA(EXTRA)	XIN/P2.6	
P1.1	SLC(EXTRA)	XOUT/P2.7	
P1.2	SDA(EXTRA)	TEST	(to programming pins)
P1.3	SLC(EXTRA)	RST	(pullup)
P1.4	CLK	P1.7	SDA(SLAVE)
P1.5		P1.6	SLC(SLAVE)
P3.1	LED	P3.7	LED
P3.0	LED	P3.6	LED
P2.0		P3.5	LED
P2.1		P2.5	
P2.2		P2.4	
P3.2	LED	P2.3	
P3.3	LED	P3.4	LED

Appendix D. *User Interface Enclosure Design.*

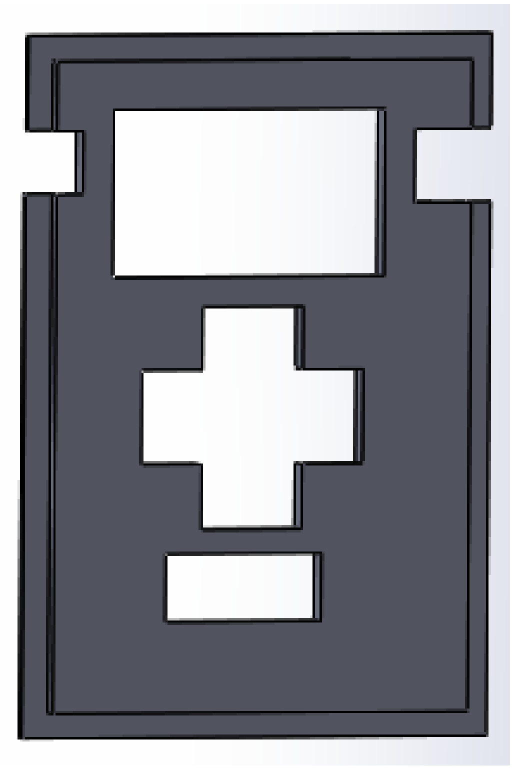

Figure 1D. SolidWorks Design for UI Front Cover.

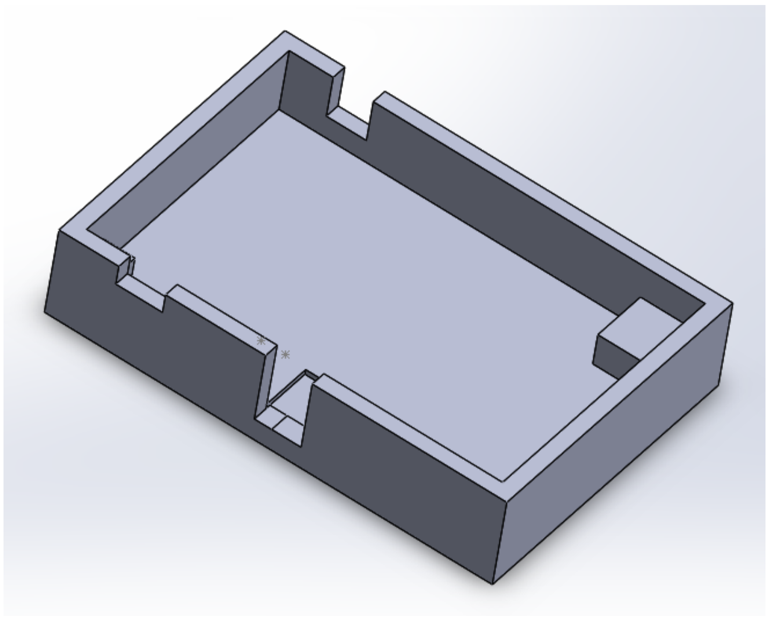

Figure 2D. SolidWorks Design for UI Enclosure.

Appendix E. *Mounting Design.*

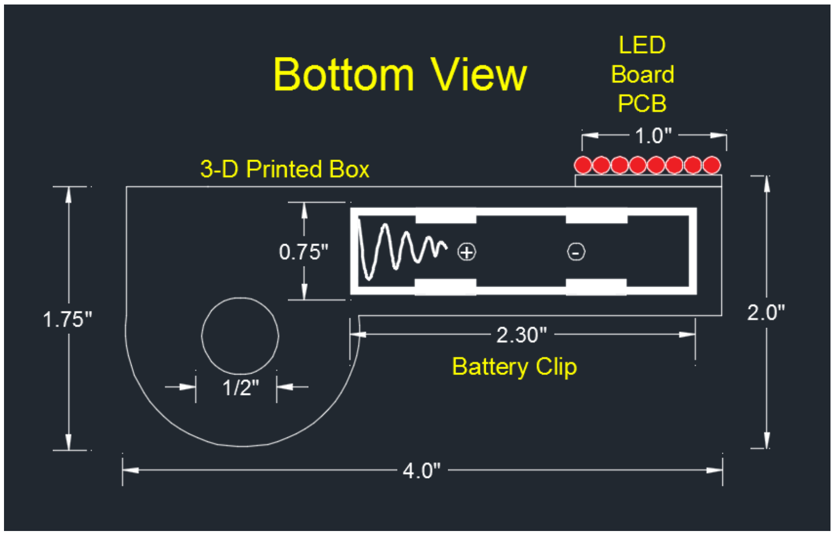

Figure 1E. Bottom View of Mounting Setup.

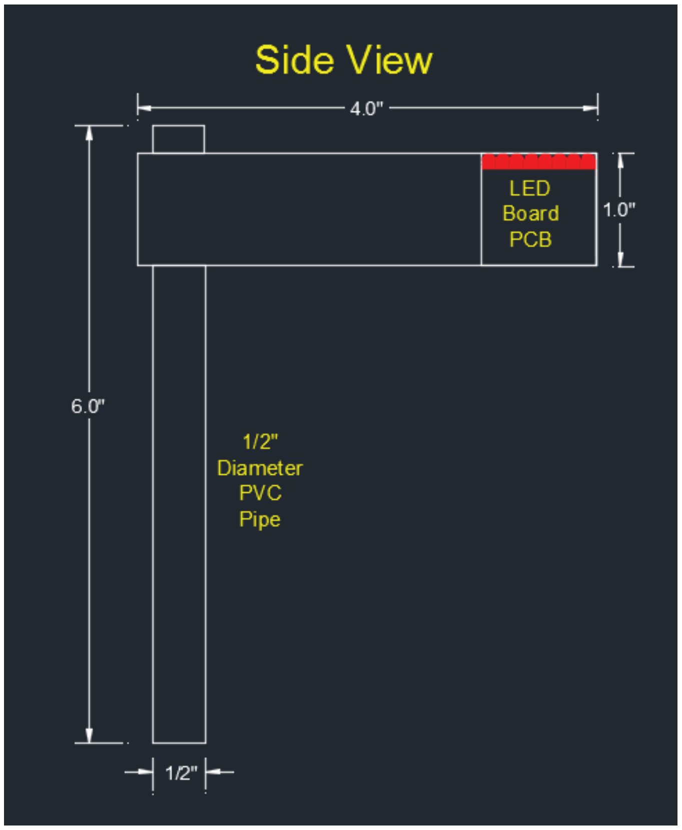

Figure 2E. Side View of Mounting Setup.

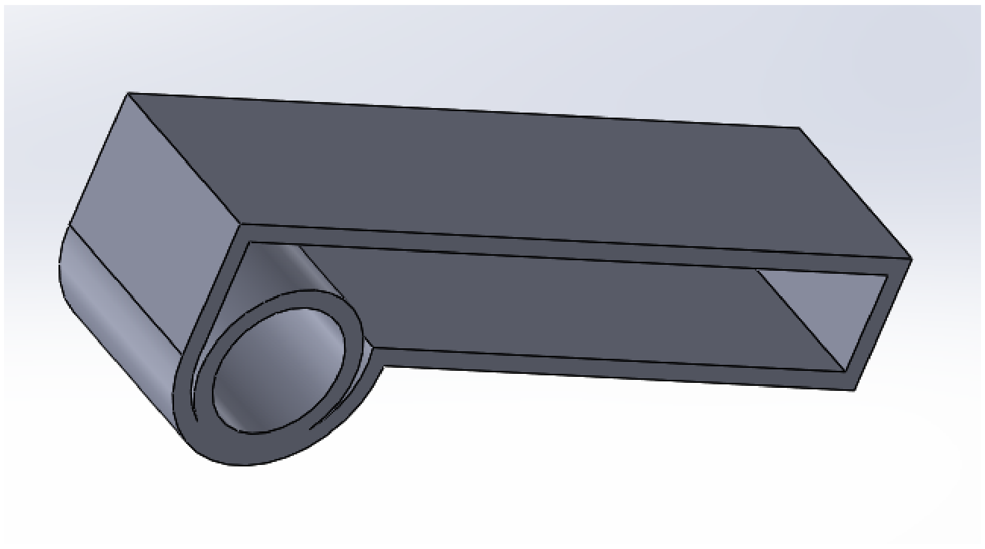

Figure 3E. SolidWorks Design of Rotation Platform.

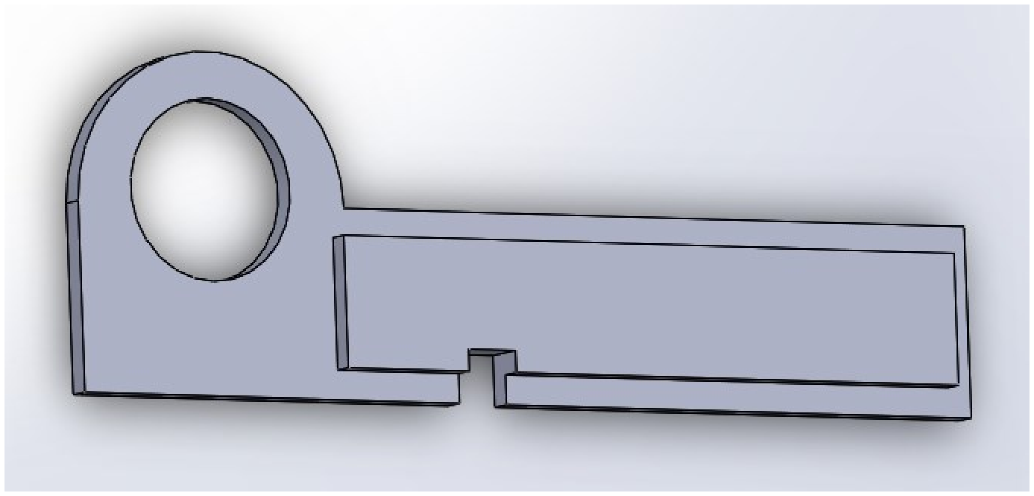

Figure 4E. SolidWorks Design of Rotation Platform Cover.

Appendix F. *Design Worksheet*

Memory Position->	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
LED Bit Number 0																																
1																																
2																																
3																																
4																																
5																																
6																																
7																																

Memory Position->	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
LED Bit Number 0																																
1																																
2																																
3																																
4																																
5																																
6																																
7																																

Figure 1F. Blank Design Worksheet.

Memory Position->		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
LED Bit Number	0	🟥				🟥			🟥	🟥	🟥			🟥				🟥		🟥	🟥	🟥	🟥	🟥			🟥	🟥	🟥			🟥	
	1	🟥	🟥		🟥	🟥		🟥				🟥		🟥	🟥			🟥				🟥				🟥				🟥		🟥	🟥
	2	🟥		🟥		🟥		🟥				🟥		🟥	🟥			🟥				🟥				🟥				🟥		🟥	🟥
	3	🟥				🟥		🟥				🟥		🟥		🟥		🟥				🟥				🟥				🟥		🟥	
	4	🟥				🟥		🟥				🟥		🟥		🟥		🟥				🟥				🟥	🟥	🟥	🟥	🟥		🟥	
	5	🟥				🟥		🟥				🟥		🟥			🟥	🟥				🟥				🟥				🟥		🟥	
	6	🟥				🟥		🟥				🟥		🟥			🟥	🟥				🟥				🟥				🟥		🟥	
	7	🟥				🟥			🟥	🟥	🟥			🟥				🟥				🟥				🟥				🟥		🟥	

Memory Position->		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
LED Bit Number	0			🟥			🟥	🟥	🟥			🟥	🟥	🟥	🟥	🟥		🟥	🟥	🟥	🟥		🟥	🟥	🟥	🟥		🟥				🟥	
	1			🟥		🟥				🟥				🟥				🟥					🟥					🟥				🟥	
	2			🟥		🟥				🟥				🟥				🟥					🟥					🟥				🟥	
	3	🟥		🟥		🟥				🟥				🟥				🟥	🟥	🟥	🟥		🟥					🟥	🟥	🟥	🟥	🟥	
	4	🟥		🟥		🟥	🟥	🟥	🟥	🟥				🟥				🟥					🟥					🟥				🟥	
	5		🟥	🟥		🟥				🟥				🟥				🟥					🟥					🟥				🟥	
	6		🟥	🟥		🟥				🟥				🟥				🟥					🟥					🟥				🟥	
	7			🟥		🟥				🟥				🟥				🟥	🟥	🟥	🟥		🟥	🟥	🟥	🟥		🟥				🟥	

Figure 2F. Example of Completed Design Worksheet.

Appendix G. *User Interface Master Code.*

I. main.c

```
//TYANA RASMUSAN 4/8/17-- MASTER USERINTERFACE
// LED EDUCATIONAL BOARD -- SENIOR DESIGN SPRING 2017
// GROUP: NATHAN ERICKSEN & ASHTYN AUMUELLER

#include <msp430.h>
#include "i2c.h"

/*
 * main.c
 */
//LCD PINS
#define MOSI BIT7       //Master-out Slave-in
#define CS BIT0         //CHIP SELECT
#define RS BIT6             //REGISTER SELECT
//#define SCL BIT5 on port 3     // SERIAL CLOCK
#define RST BIT5            //RESET
#define tidelay 1600

void init_lcd();            //INITIALIZE LCD
void writecom();            //WRITE TO ADDRESS
void writedata();           //WRITE CHARACTER TO ADDRESS
void writelcd();            //WRITE TO LCD
void writeswitches();       //SEND SWITCH PATTERN TO LCD
void lcdmemloc();           //MEMORY LOCATION TO WRITE TO LCD
void buttons();                    //BUTTON FUNCTIONS

volatile int value;         //POINTS TO ARRAY TO DISPLAY ON LCD
volatile int row;           //POINTS TO TOP OR BOTTOM ROW ON LCD
int previousswitch=0;       //PREVIOUS SWITCH POSITION
int memorylocation=0;       //MEMORY LOCATION VALUE TO SEND TO LED BOARD
int fullval;                //VALUE USED TO GET TENS & ONES PLACE FOR PO. ON LCD


char bottomrow[16];                                 //SWITCH POSTIIONS AND
POSITION NUMBER
char toprow[16]={"01234567   POS."};             //SWITCH NUMBERS AND POSITION (NEVER
CHANGES)
char pnumber[] = {"00   "};                            //START POSITION NUMBER AT 00 ON
LCD
char sendbuf[3] = {0x90, 0x00, 0x00};     //90 TO WRITE TO LED BOARD I2C
//char writebuf[] = {0x91};                        //91 TO RECEIVE FROM LED BOARD I2C
char ramtoflash[3] ={0x90, 0xFF, 0x00};         //SEND COMMAND TO SAVE CURRENT RAM TO
FLASH I2C


void init_lcd(){                   //INITIALIZE LCD

       __delay_cycles(tidelay);
       writecom(0x30);      //WAKE UP
       __delay_cycles(tidelay);
       writecom(0x30);//WAKE UP
       __delay_cycles(tidelay);
```

```
        writecom(0x39);      //FUNCTION SET
        __delay_cycles(tidelay);
        writecom(0x14);      //INTERNAL OSC FREQUENCY
        __delay_cycles(tidelay);
        writecom(0x56);//POWER CONTROL
        __delay_cycles(tidelay);
        writecom(0x6D);      //FOLLOWER CONTROL
        __delay_cycles(tidelay);
        writecom(0x70);//CONTRAST
        __delay_cycles(tidelay);
        writecom(0x0C);//DISPLAY ON
        __delay_cycles(tidelay);
        writecom(0x06);//ENTRY MODE
        __delay_cycles(tidelay);
        writecom(0x01);      //CLEAR
        __delay_cycles(tidelay);
}


void writecom(int d){                                   //WRITE TO ADDRESS
        P1OUT &= ~CS;                                   //CS LOW
        P1OUT &= ~RS;                                   //RS LOW FOR INITIALIZATION MODE
        __delay_cycles(tidelay);
        unsigned int SC = 0;                            //A0 = COMMAND
        for(SC = 1;SC <= 8; SC++){              //SEND 8 BITS
                if((d&0x80)==0x80){                     //GET ONLY THE MSB
                        P1OUT |= MOSI;                          //IF 1,THEN MOSI = 1
                }
                else {
                        P1OUT &= ~MOSI;                         //IF 0, THEN MOSI = 0
                }
                d = (d<<1);                                     //SHIFT DATA BYTE LEFT
                P3OUT &= ~(BIT5);                       //SCL
                P3OUT |= (BIT5);                        //SCL
                __delay_cycles(tidelay);
                P3OUT &= ~(BIT5);                       //SCL
                __delay_cycles(tidelay);
        }
        P1OUT |= CS;                                    //CS HIGH
}


void writedata(int d){                                  //WRITE CHARACTER TO ADDRESS
        P1OUT &= ~CS;                                   //CS LOW
        P1OUT |= RS;                                    //RS HIGH FOR WRITING MODE
        __delay_cycles(tidelay);
        unsigned int SC = 0;                            //A0 = DATA
        for(SC = 1; SC <= 8; SC++){             //SEND 8 BITS
                if((d&0x80)==0x80){                     //GET ONLY THE MSB
                        P1OUT |= MOSI;                          //IF 1, THEN MOSI = 1
                }
                else{
                        P1OUT &= ~MOSI;                         //IF 0, THEN MOSI = 0
                }
                d=(d<<1);                                       //SHIFT DATA BYTE LEFT
                P3OUT &= ~(BIT5);                       //SCL
                P3OUT |= (BIT5);                        //SCL
                __delay_cycles(tidelay);
```

```
            P3OUT &= ~(BIT5);                       //SCL
            __delay_cycles(tidelay);
        }
        P1OUT |= CS;                                //CS HIGH
}


void writelcd(int address, char* value){        //WRITE TO LCD
        P1DIR |= (BIT5 + BIT6 + BIT7 + BIT0);           //SET CS RS MOSI & RST TO OUPUTS
        P1OUT |= (BIT5 + BIT6 + BIT7 + BIT0);           //SET ALL HIGH
        P3DIR |= (BIT5);                                        //SET SCL TO OUTPUT
        P3OUT |= (BIT5);                                        //SET HIGH

        P1OUT |= CS;                                            //CS HIGH

        __delay_cycles(tidelay);
        __delay_cycles(tidelay);

        int k;

        writecom(address);                                  //WRITE TO ADDRESS
SPECIFIED

        for(k=0;k<16;k++){
                writedata(value[k]);                    //WRITE CHARACTERS IN VALUE ARRAY
SEPCIFIED
                __delay_cycles(tidelay);
        }

}

void writeswitches(char* row){          //SEND SWITCH PATTERN TO LCD
        int ledbit;                                     //VALUE TO FILL BOTTOM ROW ARRAY
WITH SWITCH PATTERN
        P2DIR &=~ (BIT0 + BIT1 + BIT2 + BIT3 + BIT4 + BIT5 + BIT6 + BIT7);    //ENABLE
INPUT DIRECTORY FOR SWITCHES
        P2REN |= BIT0 + BIT1 + BIT2 + BIT3 + BIT4 + BIT5 + BIT6 + BIT7;
//ENABLE INTERNAL PULLING RESISTORS
        P2SEL &=~ (BIT6 + BIT7);


        if(previousswitch != P2IN){                             //IF PREVIOUS SWITCH
PATTERN DOESN'T EQUAL NEW SWITCH PATTERN
                previousswitch = P2IN;                                  //SET NEW
PATTERN

                for(ledbit=0; ledbit<8; ledbit++){              //BIT COMPARISON
                        if(previousswitch&(BIT7)){
                                row[ledbit] = ' ';                      //SWITCH OFF
                        }
                        else {
                                row[ledbit] = '*';                      //SWITCH ON
                        }
                        previousswitch=(previousswitch<<1);             //BIT SHIFT TEMP

                }}
        else{                                                           //IF
PREVIOUS SWITCH PATTERN DOES EQUAL NEW SWITCH PATTERN CONTINUE
```

```
        }


}

void lcdmemloc(){                                                     //MEMORY LOCATION TO
WRITE TO LCD

        int div = 10;                                                 //DIVISION VALUE
        int val;

        if(fullval <= 0x09){                                   //FOR VALUES 0-9
                pnumber[0]= 0x30;                                     //TENS PLACE IS 0 ON
LCD
                pnumber[1]=fullval+0x30;                       //ONES PLACE ON LCD
        }
        else{                                                                //FOR VALUES
10-63
                val = fullval/div;                                    //DIVIDE THE FULL
VALUE BY 10
                pnumber[0]= val + 0x30;                               //TENS PLACE ON LCD
                fullval = fullval - (val*div);                 //SUBTRACT THE FULL VALUE
BY THE TENS PLACE
                pnumber[1]= fullval + 0x30;                    //ONES PLACE ON LCD
        }

}

void buttons(){             //BUTTON FUNCTIONS

        P3DIR &=~ (0xFF);                                             //INPUT DIRECTORY
FOR BUTTONS 0=RED 1=RIGHT 2=LEFT 3=GREEN
        P3IN &=~ (0xFF);                                              //SET ALL INPUTS LOW

        int oldfullval;

        fullval = memorylocation;                              //DECLARE MEMORY LOCATION
        oldfullval = memorylocation;                           //SET OLD FULL VALUE TO THE
MEMORY LOCATION FOR COMPARISON


        if((P3IN&BIT2) ==0){                                   //LEFT BUTTON HIT
                __delay_cycles(tidelay);
                memorylocation = memorylocation-1;      //SHIFT IN MEMORY LOCATION VECTOR
                if (memorylocation < (0x00)){
                        memorylocation = (0x3F);               //IF MEMORY LOCATION HITS 0
ROLL OVER TO 63
                }
                fullval = memorylocation;                      //FULL VALUE FOR LCD SCREEN
        }
        else if((P3IN&BIT3) ==0){                              //GREEN BUTTON HIT
                __delay_cycles(tidelay);
                sendbuf[1] = memorylocation;                   //MEMORY LOCATION TO SEND
TO MAP FLASH MEMORY
                sendbuf[2] = ~(P2IN);                                 //DIPSWITCHES INTO
SENDBUF VECTOR
                i2c_rx_bb(sendbuf,0x03, 0x00);                 //SEND TO LED BOARD I2C
```

```
	}
	else if((P3IN&BIT1) ==0){					//RIGHT BUTTON HIT
		__delay_cycles(tidelay);
		memorylocation = memorylocation+1;		//SHIFT IN VECTOR
		if(memorylocation == (0x40)){
			memorylocation = (0x00);			//IF MEMORY LOCATION HITS
63 ROLL OVER TO 0
		}
		fullval = memorylocation;				//FULL VALUE FOR LCD SCREEN
	}
	else if((P3IN&BIT0) ==0){					//RED BUTTON HIT
		__delay_cycles(tidelay);
		ramtoflash[2] = memorylocation;			//SEND NUMBER OF MEMORY
LOCATIONS TO CYCLE THROUGH ON LED BOARD
		i2c_rx_bb(ramtoflash, 0x03, 0x00);		//WRITE CURRENT RAM TO FLASH ON
LED BOARD I2C & SEND NUMBER FOR DESIGN
	}
	else{										//IF NO
BUTTONS HIT CONTINUE


	}

	if(fullval!= oldfullval){					//IF FULL VALUE HAS
CHANGED CONTINUE TO REWRITE TO LCD
		lcdmemloc();							//MEMORY
LOCATION TO WRITE TO LCD
		writelcd(0xCB, pnumber);				//WRITE MEMORY
LOCATION OF LED DESIGN ON LCD
	}
}


int main(void) {
	WDTCTL = WDTPW | WDTHOLD;	// Stop watchdog timer



	while(1){

		init_lcd();		//INITIALIZE LCD
		i2c_init();		//INITIALIZE I2C

		writeswitches(bottomrow);		//WRITE SWITCH PATTERN TO DISPLAY
		writelcd(0x80,toprow);			//WRITE 0-7 & POS.
		writelcd(0xC0, bottomrow);		//WRITE SWITCH PATTERN
		writelcd(0xCB, pnumber);		//WRITE INITIAL MEMORY LOCATION ON LCD
		buttons();						//BUTTON FUNCTIONS



	}
}
```

II. i2c.c

```
/*
 * i2c.c
 *
 *  Created on: Jul 28, 2014
 *      Author: BHill
 */
#include "msp430.h"
#define tide 800
#define SDA BIT3
#define SCL BIT4
//volatile char i2cbuf[16];

void i2c_init(void){
       BCSCTL1 = CALBC1_16MHZ;                   // Set DCO
       DCOCTL = CALDCO_16MHZ;
       P1DIR |= (SDA+SCL);
       P1OUT |= (SDA+SCL);
}


void wait_burn(int cycles){
       volatile int  k;
       for(k=cycles;k>0;k--){
              __delay_cycles(5000);
       }

}

int i2c_rx_bb(char *i2cbuf,int txnum, int rxnum){
       volatile unsigned int i,k, temp;
       //            wait_burn(100);
       P1OUT|=SCL;                        //CLK High
       P1DIR |=SDA;                       //Data Output

       P1OUT &=~SDA;                      //Data low
       __delay_cycles(tide);
       P1OUT&=~SCL;                       //Clk low
       __delay_cycles(2);
       for (i=0;i<txnum;i++){
              temp=i2cbuf[i];
              for (k=0;k<8;k++){
                     __delay_cycles(tide);
                     if ((temp&0x80)==0x80){
                            P1OUT |=SDA;
                     }
                     else{
                            P1OUT&=~SDA;
                     }
                     P1OUT|=SCL;          //CLK High
                     __delay_cycles(tide);
                     temp<<=1;
                     P1OUT&=~SCL;         //CLK Low

              }
              P1DIR &=~SDA;                      //Set input on data for ACK
              P1OUT &=~SDA;
              __delay_cycles(tide);
              P1OUT|=SCL;                        //CLK high
              __delay_cycles(tide);
              if (P1IN&SDA){                           //  Acknowledge missed, stop
condition ensues
                     P1DIR|=SDA;                 // Set data as output
                     P1OUT&=~SDA;                // Data low
                     P1OUT&=~SCL;                //CLK Low
                     __delay_cycles(tide);
                     P1OUT|=SCL;                 //CLK High
                     __delay_cycles(tide);                    //Stop Condition
                     P1OUT |=SDA;                //Data High
                     return 1;
              }
              P1OUT &=~SCL;                      //CLK Low
              P1DIR|=SDA;                        //Set direction to output for data
              __delay_cycles(tide);
              P1OUT&=~SDA;
       }

       P1DIR&=~SDA;
       P1OUT&=~SDA;
       if (rxnum==-1)
              rxnum=100;

       for (i=(txnum); i<(txnum+rxnum); i++){
              temp=0x00;
              P1DIR&=~SDA;
              for (k=0; k<8; k++){
                     temp<<=1;
                     __delay_cycles(tide);

                     P1OUT|=SCL;          //CLK High
                     __delay_cycles(tide);
                     if ((P1IN&SDA)==SDA){
                            temp|=0x01;
                     }
                     else{
                            temp&=~0x01;
                     }
                     //                          __delay_cycles(7);
                     P1OUT&=~SCL;         //CLK Low

                     P1OUT |=SDA;
              }

              P1DIR |=SDA;                       //Set output for Master ACK
              if (i==(txnum+rxnum-1)){           // make NACK
                     P1OUT |=SDA;                      //Data High for NACK
              }
              else{                              //Master send ACK
                     P1OUT &=~SDA;                     //Data Low
                     __delay_cycles(tide);

              }
              if (i==txnum){
                     if (rxnum==100)
                            rxnum=temp;
              }

              i2cbuf[i]=temp;


              __delay_cycles(tide);
              P1OUT|=SCL;                        //CLK high
              __delay_cycles(tide*3);



              P1OUT &=~SCL;                      //CLK Low

              __delay_cycles(tide);
              P1OUT &=~SDA;                      //Data Low
              P1DIR&=~SDA;



       }
       P1DIR |=SDA;                       //Set output
       P1OUT &=~SDA;                      //Data low
       __delay_cycles(tide);
       P1OUT|=SCL;                        //CLK high
       __delay_cycles(tide);
       P1OUT|=SDA;                        // Data high: Stop bit
       return 0;


}
```

III. i2c.h

```
/*
 * i2c.h
 *
 *  Created on: Jul 28, 2014
 *      Author: BHill
 */

#ifndef I2C_H_
#define I2C_H_
void i2c_init(void);
int i2c_rx_bb(char *,int, int);
void wait_burn(int);
//extern volatile char i2cbuf[16];



#endif /* I2C_H_ */
```

Appendix H. *LED Board Slave Code.*

I. main.c
```
/*
 * LED BOARD CODE -- SLAVE,FLASH MEMORY READ AND WRITE, AND DISPLAY LED PATTERN
 *
 *  Created on: April 8, 2017
 *  Author: Nathan Ericksen
 *
 *  LED EDUCATIONAL BOARD -- SENIOR DESIGN SPRING 2017
 *  GROUP: TYANA RASMUSAN & ASHTYN AUMUELLER
 */

#include <msp430.h>
#include "serial_handler.h"




char *Flash_ptr;
//char design[64] = {0X00, 0xFF, 0x02, 0x04, 0x02, 0xFF, 0x00, 0x00, 0x01, 0x01, 0xFF,
0x01, 0x01, 0x00, 0x00, 0xFF, 0x89, 0x89, 0x89, 0x00, 0x00, 0xFF, 0x81, 0x81, 0x81, 0x00,
0x00, 0xFF, 0x08, 0x08, 0x08, 0xFF};
char design[64];            //LED DESIGN TO BE FILLED
volatile int led[64];       //ARRAY FOR READING OUT OF FLASH
volatile char memloc, ledconfig;   //VARIABLES FOR MEMORY LOCATION AND LED CHARACTER
volatile int setparam = 0;
volatile int rtrn = 1;
volatile int stp_val;       //VALUE RECEIVED FROM USERINTERFACE FOR DESIGN LENGTH
void memory_mode();         //READ/WRITE RAM TO FLASH
void writemem();            //FUNCTION THAT DECIDES WHEN TO READ OR WRITE

//FUNCTION TO DISPLAY FLASHING LED PATTERN
void write_word(){

       volatile int i,j,l,m;

       //WHILE LOOP THAT CHECKS FOR ANY I2C INPUT AND DISPLAYS LED PATTERN
       while(1){

              //DELAY THAT CHECKS FOR BUTTON PRESS ON UI
              for(i=5800;i>0;i--);
              //IF LED BOARD RECIEVES ANY I2C SIGNAL IT ENDS THE FUNCTION AND RETURNS TO
PROGRAMMING MODE
              if(i2crxflag > 0){
                     return; //END FUNCTION
              }

              //OTHERWISE IF NO I2C SIGNAL IS DETECTED RUN TWO CYCLES OF THE LED PATTERN
              else{
                     for(l=0;l<2;l++){
                            //INDEXES THROUGH ALL DESIGN VALUES OF LED ARRAY DELAY ADDS A
SLOWER BLINK SPEED
                            for(j=0;j<(stp_val);j++){
                                   for(i=30000;i>0;i--){
                                          P3OUT = led[j];
                                   }
```

```
                                }
                        }
                }
        }
}

//FUNCTION FOR READING OR WRITING TO FLASH MEMORY FROM THE CONTROLLER'S RAM
void memory_mode(int setparam){

        volatile int k,l,p,z;


        Flash_ptr = (char *) 0x1040;
        if (setparam==-1){                                              // Read the current mode
from the memory

                for (k=0;k<(64);k++);
                design[k]=*(Flash_ptr+k+1);                     //Read stored flash memory
back into design matrix

        }
        else if (setparam){                                                     //Set the mode to
active mode
                //write to memory this mode
                FCTL1 = FWKEY + ERASE;                          // Set Erase bit
                FCTL3 = FWKEY;                                  // Clear Lock bit
                *Flash_ptr = 1;                                 // Dummy write to erase Flash
segment

                FCTL1 = FWKEY + WRT;                            // Set WRT bit for write
operation

                for (l=0;l<(stp_val);l++){
                        *Flash_ptr++=design[l];                         //write design
matrix to flash memory
                }

                FCTL1 = FWKEY;                                  // Clear WRT bit
                FCTL3 = FWKEY + LOCK;                           // Set LOCK bit
                rtrn = 0;
        }

}


void writemem(){

        volatile int z;

        if(memloc == 0xFF){         //RED BUTTON PRESS ON UI
                i2crxflag = 0;              //SET I2C FLAG LOW TO LOOK FOR ANOTHER I2C
INITIALIZATION
                stp_val = ledconfig;
                memory_mode(1);             //WRITE RAM TO FLASH
                write_word();       //CALL FUNCTION TO DISPLAY PATTERN ON LED'S
        }
```

```
	else{				//GREEN BUTTON PRESS ON UI OR NO BUTTON PRESS
		memory_mode(-1);	//READ FLASH
		design[memloc] = ledconfig; //UI SENDS MEMORY LOCATION NUMBER OVER, THE LED
DESIGN ARRAY READS IN THAT ONE LOCATION
		P3OUT = ledconfig;  //DISPLAYS SAMPLE LED PATTERN FOR MEMORY LOCATION
	}

	//FILLS LED ARRAY WITH DESIGN ARRAY FOR USE IN WRITE_WORD FUNCTION
	for(z=0;z<64;z++){
		led[z] = design[z];
	}

}



/*
 * main.c
 */
int main(void) {
	WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer

	volatile int extflg = 0;
	P3DIR |= 0xFF;				//SET PORT 3 TO OUTPUT MODE
	P3OUT &=~ 0xFF;				//SET ALL OF PORT 3 LOW
	P1DIR |= 0xFF;				//SET PORT 1 TO OUTPUT MODE
	P1OUT |= 0xFF;				//SET ALL OF PORT 1 HIGH

	i2c_slave_init(0x48);			//START I2C SLAVE INITIALIZATIONS
	uart_init(0x08);

//UI SENDS THREE I2C DATA VALUES STARTING WITH WRITE, MEMORY LOCATION, THEN LED PATTERN
VALUE

	//writemem();
//	write_word();


	while(1){



		if (i2crxflag>0){			//BOARD RECIEVES I2C SIGNAL
			memloc=i2cRXData[0]; //THE MEMORY LOCATION THE PATTERN IS TO BE
STORED IN
			ledconfig=i2cRXData[1];	//PATTERN OF LED'S STORED IN HEX VALUES
			writemem();			//CALL WRITEMEM FUNCTION TO READ
OR WRITE VALUES
			i2crxflag = 0;
			extflg = 0;
		}

		else{
			writemem();			//OTHERWISE READ FROM MEMORY AND
CONTINUE DISPLAYING PATTERN
		}

	}
```

```
       return 0;
}
```

II. serial_handler.c

```
/*
 * uart_control.c
 *
 *  Created on: Jul 28, 2014
 *      Author: BHill
 */
#include  "msp430.h"
#include "serial_handler.h"
#define uart_max 36
#define i2c_max  64

unsigned char tx_data_str[uart_max], rx_data_str[uart_max], rx_flag = 0, dec_str[6],
eos_flag=0;
char dec_char[6];
int tx_ptr,e_tx_ptr;
unsigned char i2cTXData[i2c_max],i2cRXData[i2c_max];
volatile int i2cTXData_ptr=0,i2cRXData_ptr=0,i2crxflag=0;
volatile int i2cmode=0;
volatile int address = 0x48;

void i2c_slave_init(int address){

       BCSCTL1 = CALBC1_16MHZ;                    // Set DCO
       DCOCTL = CALDCO_16MHZ;

       P1SEL |= BIT6 + BIT7;                     // Assign I2C pins to USCI_B0
       P1SEL2|= BIT6 + BIT7;                     // Assign I2C pins to USCI_B0


       UCB0CTL1 |= UCSWRST;                      // Enable SW reset
       UCB0CTL0 = UCMODE_3 + UCSYNC;             // I2C Slave, synchronous mode
       UCB0I2COA = address;                         // Own Address is input
       UCB0CTL1 &= ~UCSWRST;                     // Clear SW reset, resume operation
       UCB0I2CIE |= UCSTTIE;                     // Enable STT interrupt
       IE2 |= (UCB0TXIE+UCB0RXIE);                          // Enable TX interrupt
       i2cmode=0;
       P2OUT = P1SEL;
}

void uart_init(int br){
       volatile int temp=0;
       //Set baud rate to 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200,
230400, 460800, 921600
       // use index of 0 1 2 3... corresponding to the rates above
       volatile unsigned int brvec[]={0x55, 0x15, 0x0B, 0x05, 0x83, 0x41, 0xA1, 0x16,
139, 69, 35, 17};
       volatile unsigned int hrvec[]={0xD0,0x34,0x1A,0x0D,6,3,1,1,0,0,0,0};
       BCSCTL1 = CALBC1_16MHZ;                    // Set DCO
	DCOCTL = CALDCO_16MHZ;
	P1SEL |= (BIT1+BIT2);                              // P3.4,5 = USCI_A0 TXD/RXD
	P1SEL2 |= (BIT1+BIT2);
	//	UCA0CTL1 |= UCSWRST;
	UCA0CTL1 |= UCSSEL_2;                      // SMCLK
	UCA0BR0 = brvec[br];
	UCA0BR1 = hrvec[br];
	UCA0MCTL = UCBRS0;                         // Modulation UCBRSx = 1
	UCA0CTL1 &= ~UCSWRST;                      // **Initialize USCI state machine**
	IE2 |= UCA0RXIE;                           // Enable USCI_A0 RX interrupt
	__bis_SR_register(GIE);                          // interrupts enabled
}

void uart_write_string(int vals, int vale){
	int i;                                                 // writes a string
from global variable tx_data_str.  vals is starting pointer and vale is the ending value
	for(i=vals;i<vale;i++){
		while (!(IFG2&UCA0TXIFG));
		UCA0TXBUF=tx_data_str[i];
	}
	while (!(IFG2&UCA0TXIFG));
	UCA0TXBUF='\n';
	while (!(IFG2&UCA0TXIFG));
	UCA0TXBUF='\r';
}

void uart_write_fast_string(int vals, int vale){
	tx_ptr=vals;                                           // writes a string
from global variable tx_data_str.  vals is starting pointer and vale is the ending value
	e_tx_ptr=vale;                                                //  Uses
interrupts to send out bytes
	UCA0TXBUF=tx_data_str[tx_ptr];
	IE2 |= UCA0TXIE;

}


#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void)
{
	if (IE2&UCA0TXIE){
	//portion of uart_write_fast_string
		tx_ptr++;
		if (tx_ptr<e_tx_ptr)
			UCA0TXBUF=tx_data_str[tx_ptr];
		else{
			while (!(IFG2&UCA0TXIFG));
			UCA0TXBUF='\n';
			while (!(IFG2&UCA0TXIFG));
			UCA0TXBUF='\r';
			IE2 &=~ UCA0TXIE;
		}
	}
	else{

		//if (i2cmode){
		if (i2crxflag == 2){
```

```
                    UCB0TXBUF = i2cTXData[i2cTXData_ptr];                       // TX
data on i2c slave bus
                    i2cTXData_ptr++;
            }
            else{
                    i2cRXData[i2cRXData_ptr]=UCB0RXBUF;
            // rx data on i2c slave bus
                    i2cRXData_ptr++;
                    i2crxflag++;
            }
            //i2crxflag = 0;
        }



}

//  Place data in RX-buffer and set flag
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
        volatile char temp;
        if(IFG2 & UCA0RXIFG){                                                // Receive
data on UART
            rx_data_str[rx_flag]=UCA0RXBUF;                             // data is stored in
rx_data_str
            temp=rx_data_str[rx_flag];
            while (!(IFG2&UCA0TXIFG));
            //      UCA0TXBUF=temp;
            if (rx_data_str[rx_flag]=='\r')                             // new line or
carriage return set eos_flag global variable
                    eos_flag=1;
            if (rx_data_str[rx_flag]=='\n')
                    eos_flag=1;
            rx_flag++;
            if (rx_flag>uart_max){                                               //
maximum of characters starts at the beginning again
                    rx_flag=1;
            }
        }
        if(IFG2 & UCB0TXIFG){                                                // detect
beginning of i2c in slave-master mode
            i2cmode=1;
            if (UCB0STAT&UCSTTIFG){
                    UCB0STAT &= ~(UCSTPIFG + UCSTTIFG);        // Clear interrupt flags
                    i2cTXData_ptr=0;
            }       // Increme
        }
        if(IFG2&UCB0RXIFG){                                                  // detect
beginning of i2c in master-slave mode
            i2cmode=0;
            if (UCB0STAT&UCSTTIFG){
                    UCB0STAT &= ~(UCSTPIFG + UCSTTIFG);        // Clear interrupt flags
                    i2cRXData_ptr=0;
            }       // Increment data
        }

}
```

```c
char uart_get_char(int num){
	return rx_data_str[num];
}
void uart_set_char(char tx_data,int num){
	tx_data_str[num]=tx_data;
}


void conv_hex_dec(int val){
	volatile int temp,prev;
	unsigned int divider=10000;
	volatile int n=1,z=0,neg=0;
	dec_str[0]='0';
	if (val<0){
		neg=1;
		val=val*(-1);
		dec_str[0]='-';
	}
	prev=0;
	for(n=1;n<6;n++){
		temp=(val-prev)/divider;


		dec_str[n]=temp+0x30;
		prev=prev+(temp*divider);


		divider=divider/10;
	}

}

void unsigned_conv_hex_dec(int val){
	volatile unsigned int temp,prev;
	unsigned int divider=10000;
	volatile unsigned int n=1,z=0,neg=0;
	dec_str[0]='0';
	prev=0;
	for(n=1;n<6;n++){
		temp=(val-prev)/divider;


		dec_str[n]=temp+0x30;
		prev=prev+(temp*divider);


		divider=divider/10;
	}

}

int conv_dec_hex ( void ){
	volatile int num,k,temp;
	num=0;
	for (k=1;k<6;k++){
		num*=10;
		temp=dec_char[k];
```

```
            if (dec_char[k]>0x39)
                   return 0x7FFF;
            if (dec_char[k]<0x30)
                   return 0x7fff;
            num+=(dec_char[k]-0x30);
     }
     if (dec_char[0]=='-')
            num=num*(-1);
     return num;
}
```

III. serial_handler.h

```
/*
 * serial_handler.h
 *
 *  Created on: Mar 3, 2016
 *      Author: BHill
 */

#ifndef SERIAL_HANDLER_H_
#define SERIAL_HANDLER_H_

extern unsigned char tx_data_str[36], rx_data_str[36],rx_flag ,dec_str[6],eos_flag;
extern char dec_char[6];
void uart_init(int);
void uart_write_string(int,int);
char uart_get_char(int);
void uart_set_char(char,int);
void conv_hex_dec(int);
void unsigned_conv_hex_dec(int);
int conv_dec_hex (void);
void  i2c_slave_init(int);
//void uart_write_fast_string(int, int);
extern unsigned char i2cTXData[64],i2cRXData[64];
extern volatile int i2cTXData_ptr,i2cRXData_ptr,i2crxflag;
extern volatile int i2cmode;


#endif /* SERIAL_HANDLER_H_ */
```

Appendix I. Photos of Completed Assemblies.

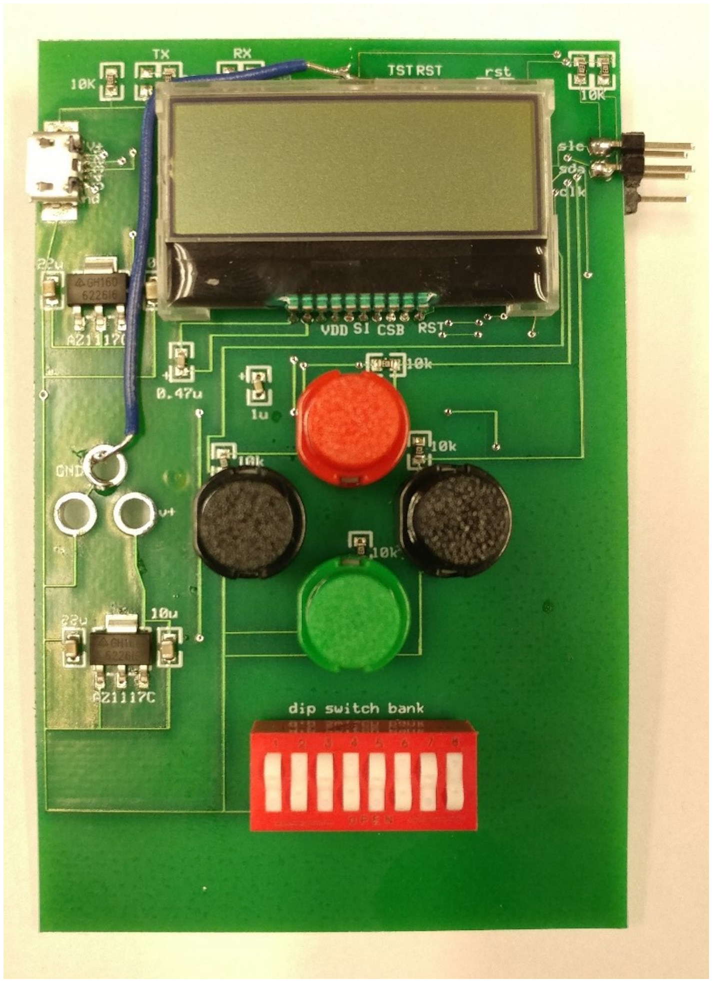

Figure 1I. Front of User Interface PCB.

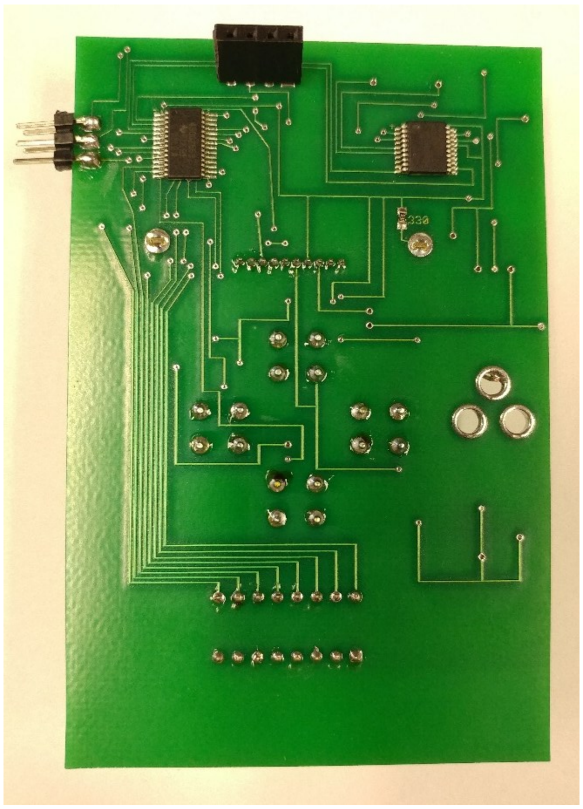

Figure 2I. Back of User interface PCB.

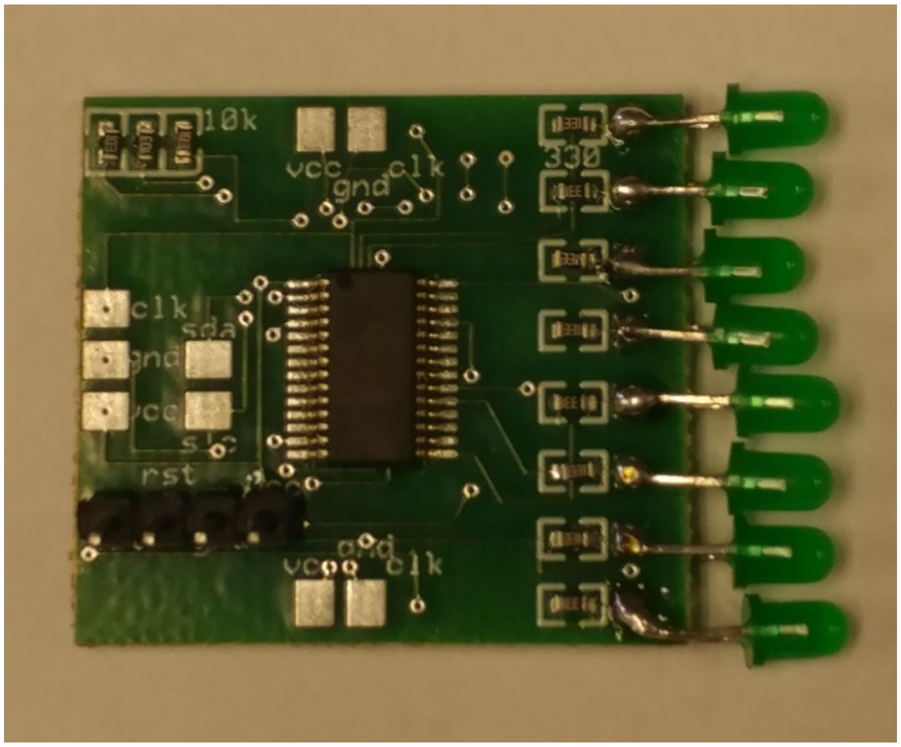

Figure 3I. Front of LED Board PCB.

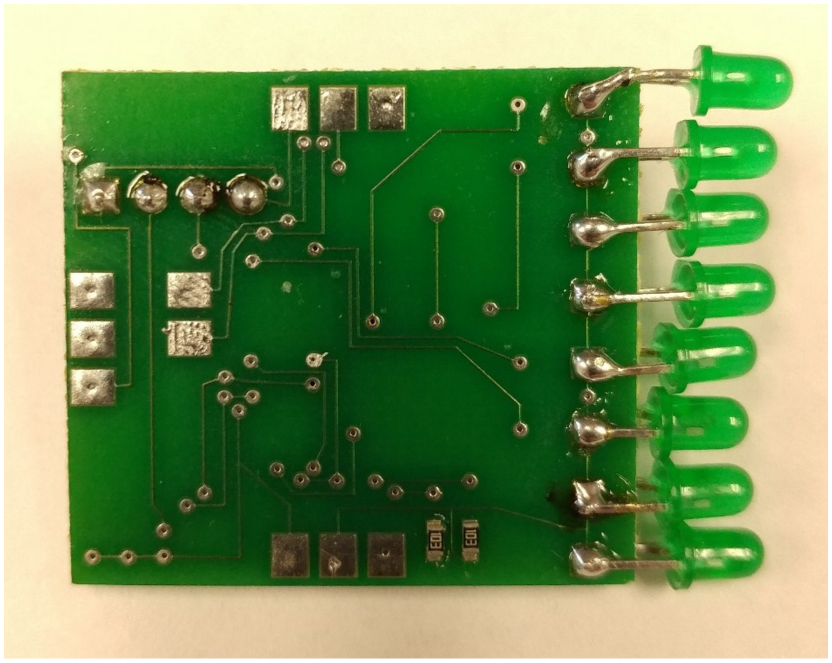
Figure 4I. Back of LED Board PCB.

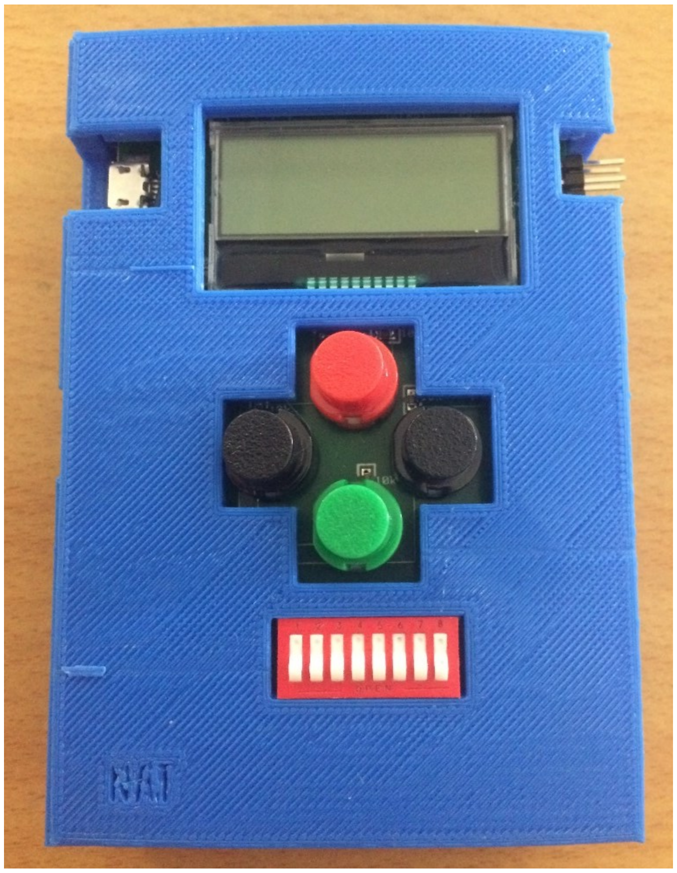
Figure 5I. User Interface with Enclosure.

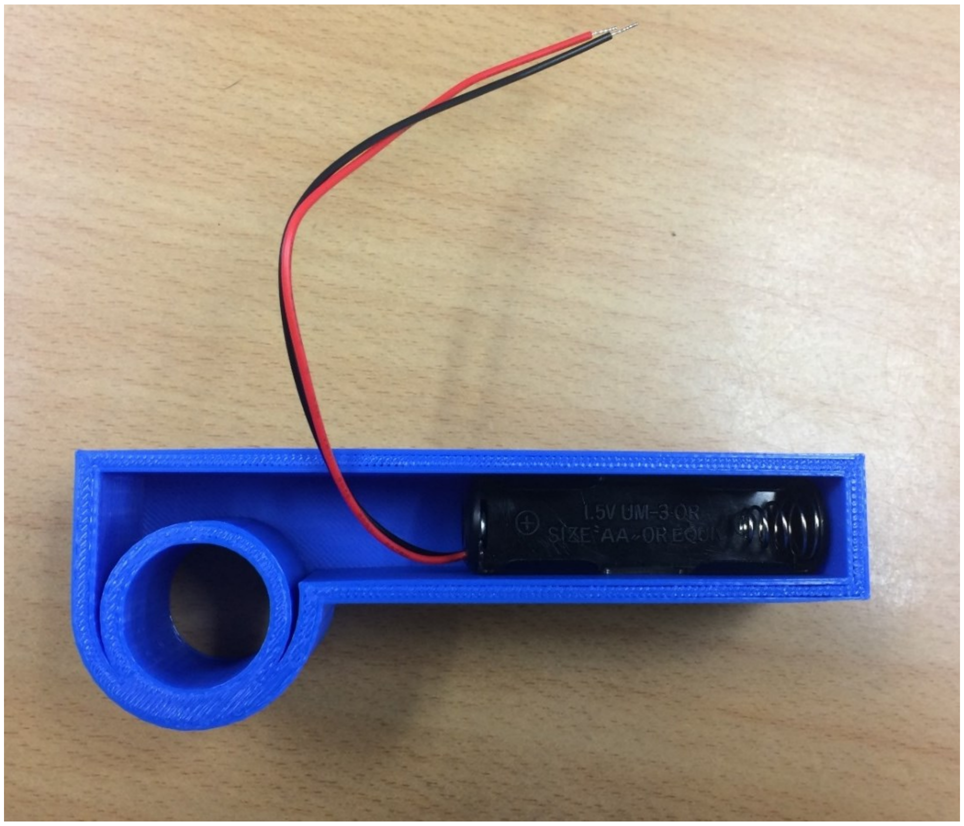

Figure 6I. Battery Box with Battery Clip.

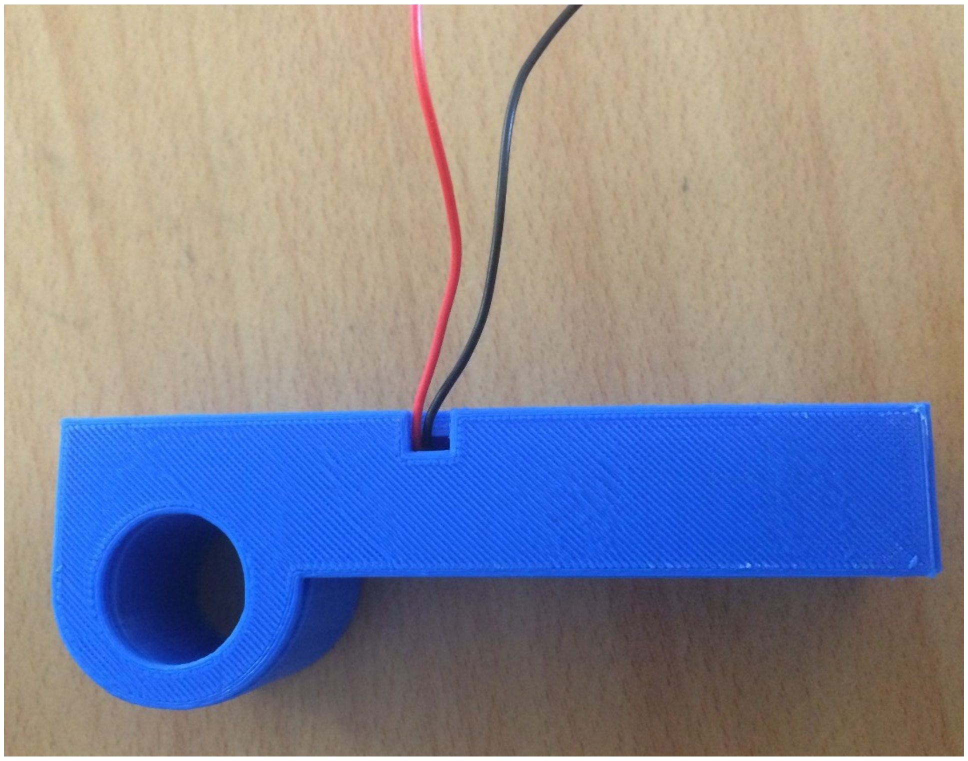

Figure 7I. Battery Box with Cover.

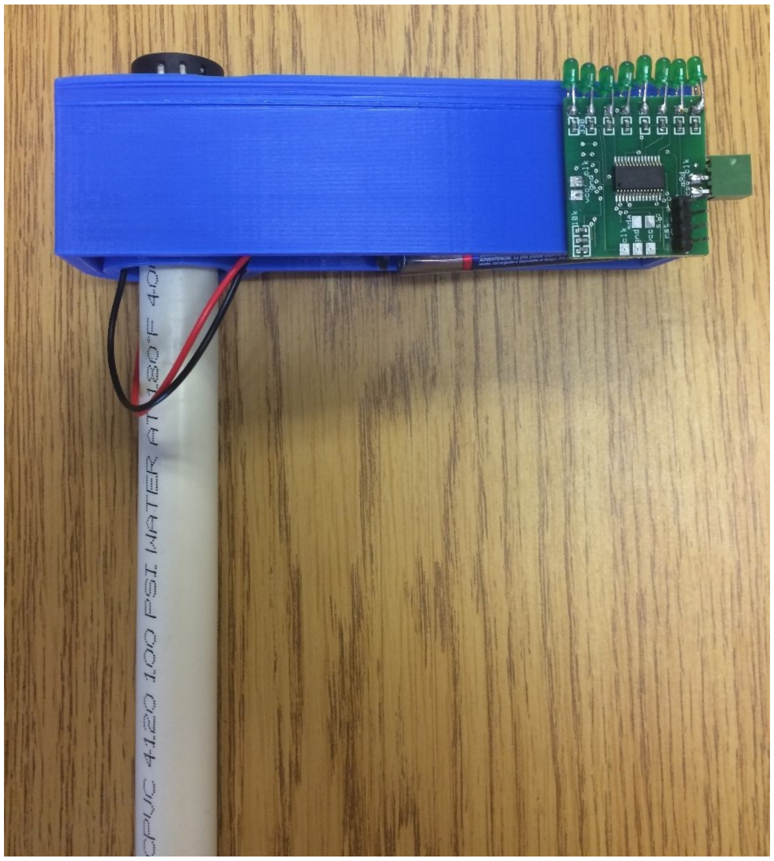
Figure 8I. Mounted LED Board.

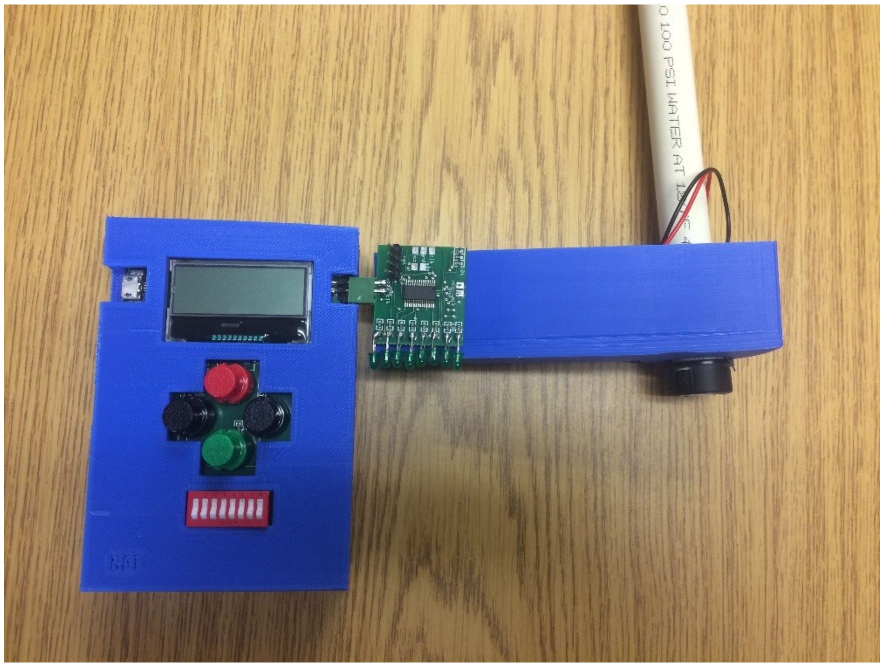
Figure 9I. Complete Assembly