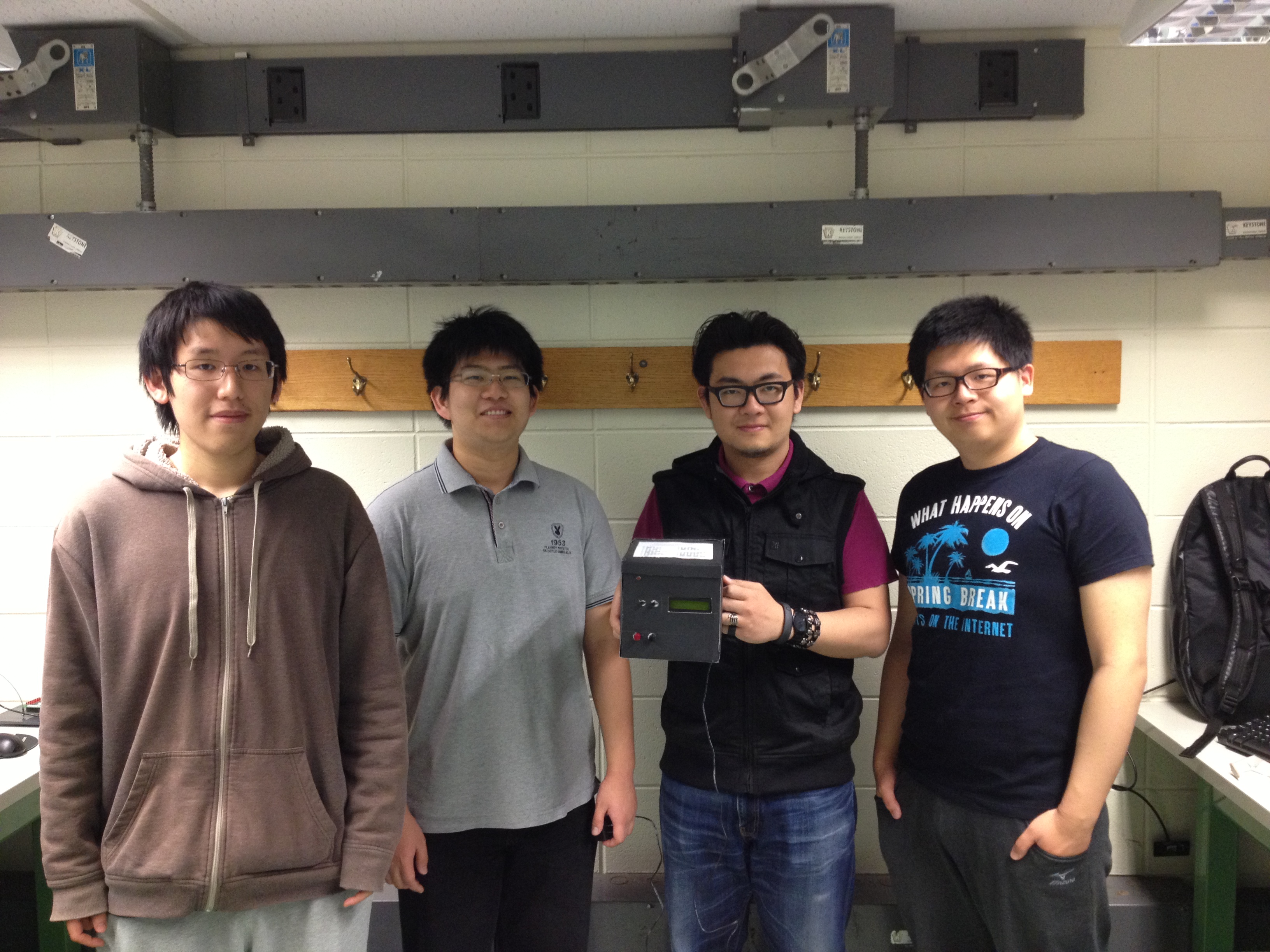
**<Morse Codes Translator>: Team 17**

****

Team members, left to right are: Daniel Li, Zihao Lu, Jiyuan Zhao, and Jinglun Huang.

|  |  |
| --- | --- |
| *Name* | *Class No.* |
| Daniel Li | 6014-L |
| Zihao Lu | 8445-L |
| Jinglun Huang | 0389-H |
| Jiyuan Zhao | 7815-Z |

|  |  |
| --- | --- |
| GRADING CRITERIA | MAX POINTS |
| Originality, creativity, level of project difficulty | 20 |
| Technical content, succinctness of report | 10 |
| Writing style, professionalism, references/citations | 10 |
| Demonstration of functionality | 20 |
| Overall quality/integration of finished product | 10 |
| Effective utilization of microcontroller resources | 10 |
| Significance of individual contributions\* | 20 |
| Poster bonus | 10 |
| Video bonus | 10 |
| Royal Showcase participation bonus (poster and video required) | 10 |

##### \**scores assigned to individual team members may vary*

##### Scoring Multiplier:

|  |  |
| --- | --- |
| 1.0 | *Excellent – among the very best projects/reports completed this semester* |
| 0.8 - 0.9 | *Good – all requirements were amply satisfied* |
| 0.6 - 0.7 | *Average – some areas for improvement, but all basic requirements were satisfied* |
| 0.4 - 0.5 | *Below average – some basic requirements were not satisfied* |
| 0.1 - 0.3 | *Poor – very few of the project requirements were satisfied* |

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| 1.0 Introduction | 1 |
| 2.0 Interface Design | 2 |
| 3.0 Peripheral Utilization | 3 |
| 4.0 Software Narrative | 4 |
| 5.0 Packaging Design | 6 |
| 6.0 Summary and Conclusions | 7 |
| 7.0 References | 8 |
| Appendix A: Individual Contributions and Activity Logs | 9 |
| Appendix B: Interface Schematic | 18 |
| Appendix C: Software Flowcharts | 20 |
| Appendix D: Packaging Design | 36 |

1. **Introduction**

The invention of electrical telegraph represents the biggest improvement on communication of human kind. Before the era of electrical telegraph, the fastest and longest communication method was semaphore, which uses arms or flags to transmit signal between stations that are within the range of telescope. The world prior to the early of 19th century seek for a new type of communication until the innovative discoveries were made in connection between electricity and magnetism by Danish physicist Hans Christian Oersted (1777-1851) in the year of 1820 [1] as well as British inventor William Sturgeon (1783-1850) in 1825 [2].

Two groups of people, Sir William Cooke (1806-1879) and Sir Charles Wheatstone (1802-1875) in England, and Samuel Morse (1791-1872), Leonard Gale (1800-1883) and Alfred Vail (1807-1859) were known for inventing telegraph around the same time in 1920s and 30s [1]. “What hath God wrought!” brings us the most successful and commonly used telegraph code system, the Morse code, in Baltimore, Maryland on May 24, 1844. This historical first message was sent by Morse from Washington, DC to Vail in Baltimore.

Morse code assigned numbers and letters with sets of dots (short mark) and dashes (long mark). Later, beeping sound was added the transmitter and receiver to give a faster conversion. Since our group was greatly interested by the improvement of telegraph on modern communication, we decided to build a Morse code translator that can convert between string of letters and Morse code corresponded to it. Our code translator operates in two modes. Mode 1 converts letters inputted, using pushbuttons to select desired letters in alphabet displayed on LCD, to Morse code outputted in a red LED and a speaker. Mode 2 reads the Morse codes user inputted using pushbutton and converts them into letters.

**2.0 Interface Design**

The 9S12C microcontroller is connected to several components, including push buttons, the 74HC164 Shift Register, the HD44780U LCD, and a 273-092 8-Ohm Mini Speaker. The power for all of these devices is provided by the WDU9-300. Schematics for the entire design can be seen in Appendix B. The datasheets for the components can be seen in Section 7.0.

The HD44780U LCD is connected to the microcontroller with the 74HC164 Shift Register and Port T. The shift register is used along with SCI to quickly send 8 bits of data using only two pins. The MOSI pin of the microcontroller sends the data signal to the 74HC164, and the CLK pin sends the clocking signal. Port T pins are used as control signals to the LCD. PTT2 is connected to the RS pin, which controls whether the data is used as a character or a command. PTT3 is connected to the R/W pin, which controls if the LCD will read or write data. PTT4 is connected to the LCDCLK pin, which clocks the registers in the LCD. Each bit in the shift register is connected to the corresponding bit in the LCD data in. The datasheets for the shift register and LCD show the numbering of these pins.

Four push buttons are connected to Port AD for the user interface. They are used as active-high and connected to ground with a pull-down 10k resistor. Another push button is used to the reset the microcontroller. It is active-low and connected to the RES pin (Pin 30) of the microcontroller.

The red LED and the speaker are connected to Port T of the microcontroller. Both of these are directly connected between the microcontroller and ground. The LED is used as a standard output of the microcontroller, but the speaker connected to PWM Channel 0 at PTT0.

Since the LCD needed to be on the front panel of the device, it was placed on a miniature breadboard along with the shift register. The miniature breadboard was placed on its side and taped to the back of the front panel. This design allowed us to solder a minimal amount of wires together.**3.0 Microcontroller Resource Utilization**

The SPI peripheral is used by the microcontroller to interface with the 74HC164 Shift Register. Based on the shift register datasheet, the maximum frequency is always higher than 6 MHz at room temperature. Therefore, the SPI is initialized for 6 Mbps, and most significant bit first. Since the data is only being written to the shift register and not read, the SPI is initialized for master mode, single direction, and no interrupts. When data is written into the SPI data register, the SPI automatically shifts out the data to the shift register.

The timer peripheral is used for precise delays and counting how long a push button is asserted. Timer channel 7 is used for interrupts only, at a rate of 1 ms. First, channel 7 is enabled and set to output compare mode with interrupts enabled. The timer is initialized for a prescale of 16, and TC7 = 1500, with reset when output compare channel 7 is asserted. Morse code is heavily dependent on the timing of signals, so the timer is necessary to ensure that time periods are accurate. With an interrupt rate of 1 ms, the program can easily and accurately wait certain amounts of time.

The PWM peripheral is used to create a waveform for the speaker. The PWM will output on Channel 0, so PWM Channel 0 is enabled, using Clock A and MODRR is used to route it to PTT0. The frequency of the input square wave should be between 1 kHz and 2 kHz to be an audible sound. Therefore the PWM is initialized for active-high, no concatenate, left-aligned, with a prescale of 64 and a period of 255. This results in a frequency of about 24000000/255/64 = 1470 Hz. The PWM and speaker can be easily turned off by setting the duty cycle to 0 and turned on by setting it to a nonzero value such as 0x7F.

The RTI system is used to sample push buttons. The interrupt rate is set to 2.048 ms to be fast as well as accurate. The previous state of the push buttons is stored in a variable and compared to the current state. If the buttons changed from inactive to active, then that button’s flag is set.

**4.0** **Software Narrative**

The software in the Morse Code Converter is best described as a state machine (flag-driven) approach. The program initializes all flags, tables, and peripherals before entering the main loop. The main loop of the program checks for push button flags set by the RTI and performs the appropriate action. A more detailed flowchart can be found in Appendix C.

If the mode1 flag is set and the program is currently in mode 2, the input string is reset and mode 1 is entered. If the flag is set while it is in mode 1, the currently selected character is entered into the input string. If the leftpb flag is set, the cursor for the currently selected character moves left. Similarly, the rghtpb flag causes the cursor to move right. After each cursor move, if the cursor is outside the bounds of the screen, the page is changed. If BS (backspace) is selected, then the last entered character is deleted. If EN (enter) is selected, then the program begins conversion from the input string to Morse code. During the conversion, the LED on the front panel will light up for 0.25 seconds for a dot and 0.75 seconds for a dash. While the LED is lit, a speaker connected to the PWM is activated. The amount of time in between actions is controlled by timer interrupts. The program waits for 250 interrupts for 0.25 seconds, and 750 interrupts for 0.75 seconds. In mode 1, the LCD displays a character menu on the top row and the input string on the bottom row.

If the mode2 flag is set, the program enters mode 2 and clears the input string. The program takes in Morse input from the mode2 button until either 16 characters have been entered or user breaks the loop by changing to mode1. Timer interrupts constantly sample the mode2 push button, which allows the program to detect how long the button has been asserted or deserted. These values are converted into a Morse sequence of dots and dashes. Also, the timer interrupts light the LED and activates the speaker when the mode2 button is asserted. After a Morse sequence is entered, it will be converted into a letter and entered into the input string. If an unrecognized sequence of Morse code is entered, such as ‘-...’, then the LCD displays an error and discards that sequence. While in mode 2, if the left pushbutton flag is set, the last entered character will be deleted. In mode 2, the LCD displays the current Morse sequence on the top row and the input string on the bottom row.

Lookup tables are used to convert from Morse code to characters and from characters to Morse code. Morse code is converted into a base 4 number with the most significant digit being the first input. A dot in Morse code is converted to a 1, a dash is converted to a 2. If the entry does not contain a total of four dots and dashes, the rest of the values are filled with 0s. For example, ‘--..’ would convert to 2211 (165 in decimal) and ‘-.-’ would convert to 2120 (152 in decimal). For the table that converts characters into Morse code, morsetable[0] corresponds to A, morsetable[1] corresponds to B, etc. Since A is ‘.-’ in Morse code, morsetable[0] is 1200 base 4 (94 in decimal). B is ‘-...’, so morsetable[1] is 2111 base 4 (149 in decimal). The inverse table is morsetable2, which converts morse code into characters. morsetable2[94] is equal to ‘A’ and morsetable2[149] is equal to ‘B’.

**5.0 Packaging Design**

It was decided at the beginning of the project on the group meeting that a carbon box was used to house all electrical components and wires. The pushbuttons, LED, LCD screen and power supply could be easily mounted. The breadboard was used for all the internal wiring and the microcontroller and all the wires were simply plugged on the board. So a carbon box with length of 21 inches, and width and height of 15 inches was chosen.

The speaker was wired and hidden inside the box since user was no need to see it. One small and four larger holes were drilled on a display panel for fitting the red LED showing the signal and four pushbuttons, two for choosing left and right, two for Mode 1 and 2. On the right side of the panel, a rectangular shaped area was cut to show the LCD screen. All parts were glued tightly on the panel to both save some space on the breadboard and prevent falling. After that, the display panel would be tapped on the front of the box. On the left side of the box, a hole was drilled to pass the reset pushbutton. The breadboard was tapped inside the box preventing moving. The 5 volts power supply was connected to the microcontroller directly and came out at the back of the box. The box was closed on the back and only left a small hole to let the cable on

When all components were fixed in the box, the box was sealed and a sheet of hard black cardboard was used to cover the whole box with pushbuttons and LCD screen shown outside. A white paper printed the Morse code was pasted on top of the box for user to find the corresponding English letters.

**6.0 Summary and Conclusions**

The Morse Code Translator implemented the Morse code translation process by converting text to Morse code and vice versa. Through this project, we gained an intimate knowledge of how pushbuttons, 74HC164 shift registers, HD44780U LCD and the 273-092 8-Ohm Mini speaker work with the microcontroller. The most direct learning experience was hardware selection. For instance, the 8-Ohm Mini speaker was chosen since it worked under our voltage restriction and could produce a sound with frequency from 600 Hz to 3000 Hz, which fitted our designed frequency (2000 Hz). The SPI peripheral was used by the microcontroller to interface with the 74HC164 shift register. When data was written into the SPI data register, the SPI automatically shifted out the data to the shift register. The timer peripheral was used for precise delays and counting how long a push button was asserted. The PWM peripheral was used to create a waveform for the speaker. LCD screen need to be wired on a separate small breadboard and hanged in the space for a limited space.

Our Morse code translator could convert Morse code and English letters in both directions. It also utilizes both audio and visual aids to illustrate the translated text. In addition, we integrated a backlit LCD for easy visibility in dark and low light conditions.

The consequence is good, but that as spectacular as it could be, we will try to improve the product if time is permitted. For example, we could use VGA cable to connect two translators so that user would be able to communicate with each other. Furthermore, we could install a wireless module so that the Morse code information can be transmitted wirelessly.

**7.0 References**

1. History.com (No date) Morse Code and the Telegraph.   
   Retrieved from <http://www.history.com/topics/telegraph>
2. Mary Bellis. (No date) The History of the Electric Telegraph and Telegraphy. Retrieved from <http://inventors.about.com/od/tstartinventions/a/telegraph.htm>
3. 74HC164. Retrieved from <http://www.nxp.com/documents/data_sheet/74HC_HCT164.pdf>
4. HD44780U (LCD-II). Retrieved from <http://www.serialwombat.com/parts/hd44780.pdf>
5. WDU9-300. Retrieved from <http://www.mouser.com/ds/2/410/WDU9-300-224465.pdf>

**Appendix A:**

**Individual Contributions**

**and**

**Activity Logs**

**Activity Log for:** Daniel Li **Role:** Software Leader

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Activity*** | ***Date*** | ***Start Time*** | ***End Time*** | ***Time Spent*** |
| Team meeting to plan project | 4/5 | 8:00 PM | 10:00 PM | 2 hours |
| Wrote the main program | 4/6 | 5:00 PM | 9:00 PM | 4 hours |
| Debugging of mode 1 | 4/7 | 5:00 PM | 6:00 PM | 1 hour |
| Wrote mode 2, code commenting | 4/8 | 7:00 PM | 10:00 PM | 3 hours |
| In lab discussion for particular parts needed | 4/9 | 7:00 PM | 12:00 PM | 5 hours |
| Debugging of mode 2 | 4/10 | 7:00 PM | 10:00 PM | 3 hours |
| Team meeting to discuss the final version of project | 4/14 | 7:00 PM | 10:00 PM | 3 hours |
| Started wiring the main circuit and LCD display screen | 4/15 | 4:00 PM | 12:00 PM | 8 hours |
| Finished wiring | 4/16 | 12:00 AM | 8:00 PM | 8 hours |
| Finished packaging | 4/18 | 1:00 PM | 7:00 PM | 6 hours |
| Demo | 4/19 | 10:30 PM | 10:45 PM | 15 minutes |
| Wrote parts of the report. | 4/20 | 8:00 PM | 10:00 PM | 2 hours |
|  |  |  | Totoal: | 45 hours |

**Written Summary of Technical Contributions: Daniel Li**

aniel Li proposed the original idea for the project. He discussed with his teammates what the program would do and how the program would function. Then, he wrote the main part of the program with the help of the other members. This part that he worked on includes the functions that output Morse code, and the functions that convert an input button into a character. He also had the idea to use a lookup table to convert between the two. Because Daniel finished the software quickly, the team had enough time to complete the hardware portion of the lab.

**Activity Log for:** Jinglun Huang **Role:** Peripheral Leader

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Activity*** | ***Date*** | ***Start Time*** | ***End Time*** | ***Time Spent*** |
| Team meeting to plan project | 4/5 | 8:00 PM | 10:00 PM | 2 hours |
| Went to Radio shack to buy speaker, pushbuttons and PCB | 4/9 | 7:30 PM | 8:00 PM | 30 minutes |
| In lab discussion for particular parts needed and modules going to be used | 4/9 | 7:00 PM | 12:00 PM | 5 hours |
| Team meeting to discuss the final version of project: a Morse code converter with LCD display and pushbutton inputs | 4/14 | 7:00 PM | 10:00 PM | 3 hours |
| Helped debugging the codes for generating beeping sound with PWM and successfully tested the circuit on breadboard | 4/15 | 4:00 PM | 12:00 PM | 8 hours |
| Soldered LCD and shift register to PCB but failed | 4/16 | 10:00 AM | 12:00 AM | 2 hours |
| Second trial on soldering PCB but failed again. Due to limitation on PCB, used breadboard to connect the circuits. | 4/16 | 12:00 AM | 8:00 PM | 8 hours |
| Built the front of converter with carbon boards. | 4/16 | 8:00 PM | 10:00 PM | 2 hours |
| Started packaging the body of converter also using carbon board | 4/17 | 4:00 PM | 9:00 PM | 5 hours |
| Finished packaging and embellishment | 4/18 | 1:00 PM | 7:00 PM | 6 hours |
| Demo | 4/19 | 10:30 AM | 10:45 AM | 15 minutes |
|  |  |  | Total: | 41 hours |

**Written Summary of Technical Contributions:** Jinglun Huang

Jinglun Huang organized the members together and planed the meeting schedules. He determined parts required to achieve desired functionalities. During the coding of main function, Jinglun helped software leader Daniel to come up with better solution and also debugged several mistakes. He also played an important role on the hardware and packaging. He successfully connected the circuit on breadboard with Daniel, soldered most of components and, although failed, soldered two sets of PCB with shift register and LCD. Jinglun also helped Zihao with packaging the final body of the converter.

**Activity Log for:** Jiyuan Zhao **Role:** Interface Leader

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Activity*** | ***Date*** | ***Start Time*** | ***End Time*** | ***Time Spent*** |
| Team meeting to plan project | 4/5 | 8:00 PM | 10:00 PM | 2 hours |
| Went to Radio shack to buy speaker, pushbuttons and PCB | 4/9 | 7:30 PM | 8:00 PM | 30 minutes |
| In lab discussion for particular parts needed and modules going to be used | 4/9 | 7:00 PM | 12:00 PM | 5 hours |
| Team meeting to discuss the final version of project: a Morse code converter with LCD display and pushbutton inputs | 4/14 | 7:00 PM | 10:00 PM | 3 hours |
| Started packaging the body of converter also using carbon board | 4/17 | 4:00 PM | 9:00 PM | 5 hours |
| Finished packaging | 4/18 | 1:00 PM | 7:00 PM | 6 hours |
| Demo | 4/19 | 10:30 AM | 10:45 AM | 15 minutes |
| Made poster | 4/20 | 6:00 PM | 11:00 PM | 5 hours |
| Made video | 4/21 | 6:00 PM | 11:00 PM | 5 hours |
|  |  |  | Total: | 32 hours |

**Written Summary of Technical Contributions:** Jiyuan Zhao

During the hardware wiring and interfacing phase, Jiyuan helped peripheral leader Jinglun and TDP leader Zihao and also debugged several mistakes. He also helped on the hardware and packaging. He helped Jinglun connected the circuit on breadboard as well as Zihao with packaging the final body of the converter. Furthermore, Jiyuan was in charge of posters and videos.

**Activity Log for:** Zihao Lu **Role:** TDP

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Activity*** | ***Date*** | ***Start Time*** | ***End Time*** | ***Time Spent*** |
| Team meeting to plan project | 4/5 | 8:00 PM | 10:00 PM | 2 hours |
| Went to Radio shack to buy speaker, pushbuttons and PCB | 4/9 | 7:30 PM | 8:00 PM | 30 minutes |
| In lab discussion for particular parts needed and modules going to be used | 4/9 | 7:00 PM | 12:00 PM | 5 hours |
| Team meeting to discuss the final version of project: a Morse code converter with LCD display and pushbutton inputs | 4/14 | 7:00 PM | 10:00 PM | 3 hours |
| Helped debugging the codes for generating beeping sound with PWM and successfully tested the circuit on breadboard | 4/15 | 4:00 PM | 12:00 PM | 8 hours |
| Purchased the carbon box, card board and other materials from Wal-Mart. | 4/16 | 5:00 PM | 6:00 PM | 1 hour |
| Built the front of converter with carbon boards. | 4/16 | 6:00 PM | 8:00 PM | 2 hours |
| Wired the main parts of the circuit on breadboard, and connected the LCD subcircuit. | 4/16 | 8:00 PM | 12:00 PM | 4 hours |
| Tapped and glued all components on the box, made sure that no mess-up occurs. | 4/17 | 12:00 AM | 4:00 PM | 4 hours |
| Started packaging the body of converter also using carbon board | 4/17 | 4:00 PM | 9:00 PM | 5 hours |
| Finished packaging and embellishment | 4/18 | 1:00 PM | 7:00 PM | 6 hours |
| Demo | 4/19 | 10:30 PM | 10:45 PM | 15 minutes |
| Integrated all the documents and pictures from each team members. | 4/22 | 1:00 PM | 3:00 PM | 2 hours |
|  |  |  | Total: | 42 hours |

**Written Summary of Technical Contributions:** Zihao Lu

Zihao was in charge of choosing and purchasing most parts of the electrical components and packaging materials. The team meeting and exchanging of documents were organized by him step by step. He connected the main circuit and wired the pushbuttons and display LCD, LED with help from his teammates. He, with Jinglun, did the whole packaging, including drilling holes on the carbon box, tapping and gluing all components inside the box, and covering the outside of the box with black cardboard, etc. It was him demonstrating the project to Professor Meyer.

**Appendix B:**

**Interface Schematic**

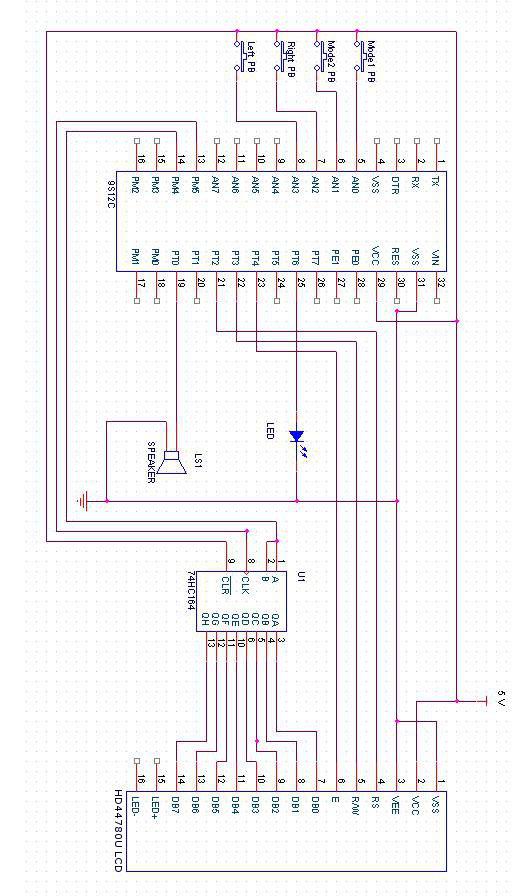


Figure 3.1: Morse Code Converter Schematic

This schematic shows how the 9S12C is connected to the Shift Register, speaker, push buttons, and LCD.

**Appendix C:**

**Software Flowcharts**

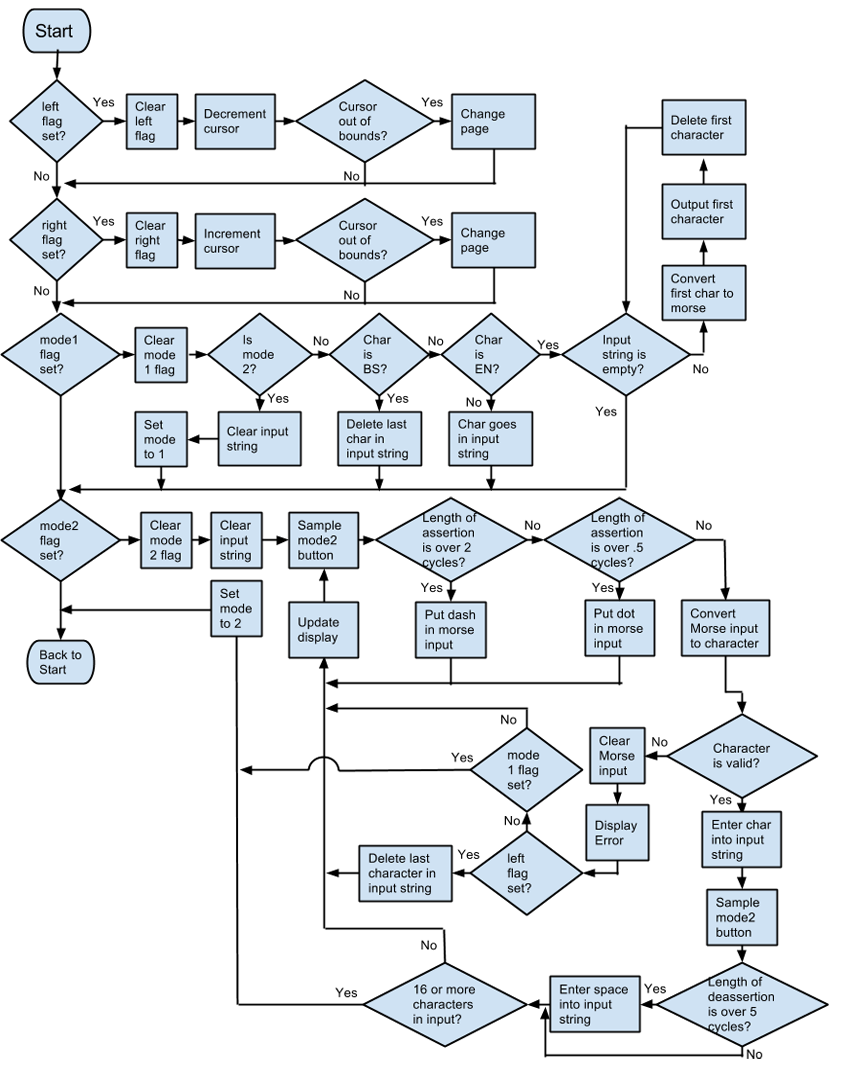


Figure 4.1: Software Flowchart

The figure above shows that the device follows a state machine (flag-driven) approach.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; ECE 362 - Mini-Project C Source File - Spring 2013

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;

; Team ID: 17

;

; Project Name: Morse Code Converter

;

; Team Members:

;

; - Team/Doc Leader: Zihao Lu Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

;

; - Software Leader: Daniel Li Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

;

; - Interface Leader: Jiyuan Zhao Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

;

; - Peripheral Leader: Jinglun Huang Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

;

;

; Academic Honesty Statement: In signing above, we hereby certify that we

; are the individuals who created this HC(S)12 source file and that we have

; not copied the work of any other student (past or present) while completing

; it. We understand that if we fail to honor this agreement, we will receive

; a grade of ZERO and be subject to possible disciplinary action.

;

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;

; The objective of this Mini-Project is to make a device that can convert

; user inputted characters into Morse code in the form of a blinking LED

; and a speaker. The device can also convert Morse code entered from a

; push button into characters.

;

;

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;

; List of project-specific success criteria (functionality that will be

; demonstrated):

;

; 1. The device will have a mode where the user can input characters in

;1 a menu shown on the LCD. The inputted characters will be converted

; into Morse code.

;

; 2. The device will have a mode where the user can input Morse code from

; a push button. The Morse code will be converted into characters and

; displayed on the LCD.

;

; 3. The user can change modes by pressing push button corresponding to

; each mode.

;

; 4. When Morse code is being output, an LED will blink the correct

; length and amount of time. While the LED is on, a speaker will also

; be turned on. Similarly, while morse code is being input, the LED

; and speaker will be on while the button is pressed.

;

; 5. When entering characters in the first mode, the user can perform

; a backspace operation by selecting "BS" and an enter operation by

; selecting "EN".

;

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;

; Date code started: 4/9/2013

;

; Update history:

;

; Date: 4/9/2013 Name: Daniel Li Update: Started coding, finished coding Mode 1

;

; Date: 4/13/2013 Name: Daniel Li Update: Finished Mode 2

;

; Date: 4/15/2013 Name: Daniel Li Update: Debugging

;

;

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <hidef.h> /\* common defines and macros \*/

#include "derivative.h" /\* derivative-specific definitions \*/

#include <mc9s12c32.h>

// All funtions after main should be initialized here

char inchar(void);

void outchar(char x);

void fdisp(void);

void morse(void);

void clearinput(void);

void morseout(int length);

void timerwait(int waitcount);

void timerwait2(int waitcount);

void getchar(void);

void getcharerror(void);

void checkend(void);

void shiftout(byte outbyte);

void lcdwait(void);

void send\_byte(byte outbyte);

void send\_i(byte inst);

void chgline(byte curpos);

void print\_c(byte outbyte);

void pmsglcd(char\* outString, int lneint);

// Variable declarations

int leftpb = 0; // left pushbutton flag, PAD3

int rghtpb = 0; // right pushbutton flag, PAD2

int morsepb = 0; // morse (mode2) pushbutton flag, PAD1

int charpb = 0; // char (mode1) pushbutton flag, PAD0

int prevpb = 0; // previous pushbutton state

int twait = 0; // amount of cycles the timer waits for

int waitmod = 250; // wait multiplier (250 ms)

int waitcnt = 0; // counter for timer wait

int enterflag = 0;

char cursor = 0; //current location of the cursor

int page = 0; //current page number

int i; //loop index variable

int endflag = 0;

int in[4]; //0 - none, 1 - dot, 2 - dash

char test;

int oncount; //amount of time the button was pressed

int offcount; //amount of time the button was off

int mode = 1;

char minput[17]; //string of morse input

char input[17]; //string input by the user

int index; //current index of input string

char morsetable[26]; //table of morse code values;

char morsetable2[256]; //dots and dashes -> letters

//;LCD COMMUNICATION BIT MASKS

int RS = 0x04; //;RS pin mask (PTT[2])

int RW = 0x08; //;R/W pin mask (PTT[3])

int LCDCLK = 0x10; //;LCD EN/CLK pin mask (PTT[4])

//;LCD INSTRUCTION CHARACTERS

char LCDON = 0x0F; //;LCD initialization command

char LCDCLR = 0x01; //;LCD clear display command

char TWOLINE = 0x38; //;LCD 2-line enable command

char CURMOV = 0xFE; //;LCD cursor move instruction

char LINE1 = 0x80; //;LCD line 1 cursor position

char LINE2 = 0xC0; //;LCD line 2 cursor position

int loop = 1;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Initializations

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void initializations(void) {

//; Set the PLL speed (bus clock = 24 MHz)

CLKSEL = CLKSEL & 0x80; //; disengage PLL from system

PLLCTL = PLLCTL | 0x40; //; turn on PLL

SYNR = 0x02; //; set PLL multiplier

REFDV = 0; //; set PLL divider

while (!(CRGFLG & 0x08)){ }

CLKSEL = CLKSEL | 0x80; //; engage PLL

// Disable watchdog timer (COPCTL register)

COPCTL = 0x40 ; //COP off; RTI and COP stopped in BDM-mode

// Add additional port pin initializations here

DDRT = 0x7F; //PTT7 is input, PTT6-0 are outputs

ATDDIEN = 0xCF; //program PAD7-6,3-0 pins as digital inputs

/\*

Initialize PWM for output of a square wave on PTT0

;\*/

MODRR = 0x01; //route PTT0 to PWM

PWME = 0x01; //enable ch 0

PWMPOL = 0x01; //active high

PWMCTL = 0x00; //no concatenate

PWMCAE = 0x00; //left aligned

PWMPRCLK = 0x06;// div 64

PWMCLK = 0x00; // use clock a on ch 0;

PWMPER0 = 0xFF; // div 255

PWMDTY0 = 0x00; // intially off

//; Initialize SPI for baud rate of 6.25MHz, MSB first

//; - Consult 3.1.3 of the SPI Block User Guide

//; < add SPI initialization code here >

SPIBR = 0x01; // divisor = (1 + b000) \* 2^(1 + b001) = 4

SPICR1 = 0x50;

SPICR2 = 0x00;

/\*; Initialize TIM Ch 7 (TC7) for periodic interrupts every 1.0 ms

; Enable timer subsystem

; Set channel 7 for output compare

; Set appropriate pre-scale factor and enable counter reset after OC7

; Set up channel 7 to generate 1 ms interrupt rate

; Initially disable TIM Ch 7 interrupts

\*/

TSCR1 = 0x80; //7 bit = enable

TIOS = 0x80; //channel 7 output compare

TSCR2 = 0x0C; //counter reset, prescale of 16

TC7 = 1500; // 1500 \* 16 = 0.001 \* 24000000

TIE = 0x00; //channel 7 disable

TIE = 0x80; //channel 7 enable interrupt

/\* Initialize the LCD

; - pull LCDCLK high (idle) PTT\_PTT4

; - pull R/W' low (write state) PTT\_PTT3

; - turn on LCD (LCDON instruction)

; - enable two-line mode (TWOLINE instruction)

; - clear LCD (LCDCLR instruction)

; - wait for 2ms so that the LCD can wake up

\*/

PTT\_PTT4 = 1;

PTT\_PTT3 = 0;

send\_i(LCDON);

send\_i(TWOLINE);

send\_i(LCDCLR);

lcdwait();

// Initialize input strings

clearinput();

input[16] = 0; //end string with null

for(i = 0; i < 15; i++) //clear minput

minput[i] = ' ';

minput[16] = 0; //end string with null

// Initialize RTI for 2.048 ms interrupt rate

RTICTL = RTICTL | 0x1F;

CRGINT = CRGINT | 0x80;

// Initialize Morse Tables

morsetable[0] = (1 << 6) | (2 << 4) | (0 << 2) | (0); //A

morsetable[1] = (2 << 6) | (1 << 4) | (1 << 2) | (1); //B

morsetable[2] = (2 << 6) | (1 << 4) | (2 << 2) | (1); //C

morsetable[3] = (2 << 6) | (1 << 4) | (1 << 2) | (0); //D

morsetable[4] = (1 << 6) | (0 << 4) | (0 << 2) | (0); //E

morsetable[5] = (1 << 6) | (1 << 4) | (2 << 2) | (0); //F

morsetable[6] = (2 << 6) | (2 << 4) | (1 << 2) | (0); //G

morsetable[7] = (1 << 6) | (1 << 4) | (1 << 2) | (0); //H

morsetable[8] = (1 << 6) | (1 << 4) | (0 << 2) | (0); //I

morsetable[9] = (1 << 6) | (2 << 4) | (2 << 2) | (2); //J

morsetable[10] = (2 << 6) | (1 << 4) | (2 << 2) | (0); //K

morsetable[11] = (1 << 6) | (2 << 4) | (1 << 2) | (0); //L

morsetable[12] = (2 << 6) | (2 << 4) | (0 << 2) | (0); //M

morsetable[13] = (2 << 6) | (1 << 4) | (0 << 2) | (0); //N

morsetable[14] = (2 << 6) | (2 << 4) | (2 << 2) | (0); //O

morsetable[15] = (1 << 6) | (2 << 4) | (2 << 2) | (1); //P

morsetable[16] = (2 << 6) | (2 << 4) | (1 << 2) | (2); //Q

morsetable[17] = (1 << 6) | (2 << 4) | (1 << 2) | (0); //R

morsetable[18] = (1 << 6) | (1 << 4) | (1 << 2) | (0); //S

morsetable[19] = (2 << 6) | (0 << 4) | (0 << 2) | (0); //T

morsetable[20] = (1 << 6) | (1 << 4) | (2 << 2) | (0); //U

morsetable[21] = (1 << 6) | (1 << 4) | (1 << 2) | (2); //V

morsetable[22] = (1 << 6) | (2 << 4) | (2 << 2) | (0); //W

morsetable[23] = (2 << 6) | (1 << 4) | (1 << 2) | (2); //X

morsetable[24] = (2 << 6) | (1 << 4) | (2 << 2) | (2); //Y

morsetable[25] = (2 << 6) | (2 << 4) | (1 << 2) | (1); //Z

for(i = 0; i < 256; i++)

morsetable2[i] = 0;

morsetable2[(1 << 6) | (2 << 4) | (0 << 2) | (0)] = 0 + 'A';//A

morsetable2[(2 << 6) | (1 << 4) | (1 << 2) | (1)] = 1 + 'A';//B

morsetable2[(2 << 6) | (1 << 4) | (2 << 2) | (1)] = 2 + 'A';//C

morsetable2[(2 << 6) | (1 << 4) | (1 << 2) | (0)] = 3 + 'A';//D

morsetable2[(1 << 6) | (0 << 4) | (0 << 2) | (0)] = 4 + 'A';//E

morsetable2[(1 << 6) | (1 << 4) | (2 << 2) | (0)] = 5 + 'A';//F

morsetable2[(2 << 6) | (2 << 4) | (1 << 2) | (0)] = 6 + 'A';//G

morsetable2[(1 << 6) | (1 << 4) | (1 << 2) | (0)] = 7 + 'A';//H

morsetable2[(1 << 6) | (1 << 4) | (0 << 2) | (0)] = 8 + 'A';//I

morsetable2[(1 << 6) | (2 << 4) | (2 << 2) | (2)] = 9 + 'A';//J

morsetable2[(2 << 6) | (1 << 4) | (2 << 2) | (0)] = 10 + 'A';//K

morsetable2[(1 << 6) | (2 << 4) | (1 << 2) | (0)] = 11 + 'A';//L

morsetable2[(2 << 6) | (2 << 4) | (0 << 2) | (0)] = 12 + 'A';//M

morsetable2[(2 << 6) | (1 << 4) | (0 << 2) | (0)] = 13 + 'A';//N

morsetable2[(2 << 6) | (2 << 4) | (2 << 2) | (0)] = 14 + 'A';//O

morsetable2[(1 << 6) | (2 << 4) | (2 << 2) | (1)] = 15 + 'A';//P

morsetable2[(2 << 6) | (2 << 4) | (1 << 2) | (2)] = 16 + 'A';//Q

morsetable2[(1 << 6) | (2 << 4) | (1 << 2) | (0)] = 17 + 'A';//R

morsetable2[(1 << 6) | (1 << 4) | (1 << 2) | (0)] = 18 + 'A';//S

morsetable2[(2 << 6) | (0 << 4) | (0 << 2) | (0)] = 19 + 'A';//T

morsetable2[(1 << 6) | (1 << 4) | (2 << 2) | (0)] = 20 + 'A';//U

morsetable2[(1 << 6) | (1 << 4) | (1 << 2) | (2)] = 21 + 'A';//V

morsetable2[(1 << 6) | (2 << 4) | (2 << 2) | (0)] = 22 + 'A';//W

morsetable2[(2 << 6) | (1 << 4) | (1 << 2) | (2)] = 23 + 'A';//X

morsetable2[(2 << 6) | (1 << 4) | (2 << 2) | (2)] = 24 + 'A';//Y

morsetable2[(2 << 6) | (2 << 4) | (1 << 2) | (1)] = 25 + 'A';//Z

fdisp();

chgline(LINE1 + cursor);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Main

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main(void) {

DisableInterrupts;

initializations();

EnableInterrupts;

while(loop) {

//loop

//; If right pushbutton pressed, increment cursor

if(rghtpb == 1){

rghtpb = 0;

if(mode == 1) {

cursor++;

if(cursor >= 16) { //if cursor goes over the bound, change page

page = (page + 1) % 2;

cursor = 0;

}

fdisp();

chgline(LINE1 + cursor);

}

}

//; If left pushbutton pressed, decrement cursor

if(leftpb == 1){

leftpb = 0;

if(mode == 1) {

cursor--;

if(cursor < 0) { //if cursor goes over the bound, change page

page = (page + 1) % 2;

cursor = 15;

}

fdisp();

chgline(LINE1 + cursor);

}

}

//; If char pushbutton pressed, enter the value into the input string, enter mode 1

if(charpb == 1){

charpb = 0;

if(mode == 2) { //change mode and clear input

mode = 1;

clearinput();

}

if(page == 0) {

input[index] = 'A' + cursor; //insert character from first page

index++;

}

else if(page == 1 && (cursor == 10 || cursor == 13)) { //insert space

input[index] = ' ';

index++;

}

else if(page == 1 && (cursor == 11 || cursor == 12)) { //backspace

if(index > 0) {

input[index - 1] = ' ';

index--;

}

}

else if(page == 1 && (cursor == 14 || cursor == 15)) { //enter

enterflag = 1;

}

else if(page == 1) {

input[index] = 'Q' + cursor; //insert character from second page

index++;

}

if(index >= 16 || enterflag == 1) {

enterflag = 0;

morse(); //outputs the characters

clearinput();

}

fdisp();

chgline(LINE1 + cursor);

}

//records the morse input, outputs characters

if(morsepb == 1){

morsepb = 0; //intialize variables for mode2

endflag = 0;

TIE = 0x80;

mode = 2;

clearinput();

if(PTAD\_PTAD0 == 1) { //sound

PTT\_PTT6 = 1;

PWMDTY0 = 0x10;

}

while(index < 16 && endflag == 0) //get characters until full or break

getchar();

TIE = 0x00;

PTT\_PTT6 = 0; //sound off

PWMDTY0 = 0x00;

morsepb = 0;

charpb = 0;

}

\_FEED\_COP(); /\* feeds the dog \*/

} /\* loop forever \*/

/\* please make sure that you never leave main \*/

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; RTI interrupt service routine: RTI\_ISR

;

; Initialized for 2.048 ms interrupt rate

;

; Samples state of pushbuttons

; PAD0 - morsepb (mode2)

; PAD1 - charpb (mode1)

; PAD2 - rghtpb

; PAD3 - leftpb

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

interrupt 7 void RTI\_ISR(void)

{

// set CRGFLG bit

CRGFLG = CRGFLG | 0x80;

if((prevpb & 0x80) && !PTAD\_PTAD7) //mask PAD3

leftpb = 1;

if((prevpb & 0x40) && !PTAD\_PTAD6) //mask PAD3

rghtpb = 1;

if(!(prevpb & 0x08) && PTAD\_PTAD3) //mask PAD3

leftpb = 1;

if(!(prevpb & 0x04) && PTAD\_PTAD2) //mask PAD2

rghtpb = 1;

if(!(prevpb & 0x02) && PTAD\_PTAD1) //mask PAD1

charpb = 1;

if(!(prevpb & 0x01) && PTAD\_PTAD0) //mask PAD0

morsepb = 1;

prevpb = PTAD;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; TIM interrupt service routine

;

; Initialized for 1.0 ms interrupt rate

;

; mode1 - interrupts enabled only during wait loops, timer controls the wait length

; mode2 - interrupts always enabled, samples the PAD0 input every millisecond

;

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

interrupt 15 void TIM\_ISR( void)

{

// set TFLG1 bit

TFLG1 = TFLG1 | 0x80;

// No need to add anything in the .PRM file, the interrupt number is included above

if(mode == 1) {

waitcnt++;

if(waitcnt >= waitmod) {

waitcnt = 0;

twait--;

}

if(twait <= 0) {

TIE = 0x00; //channel 7 interrupt disable

}

}

else if(mode == 2) {

if(PTAD\_PTAD0 == 1) {

oncount++;

PTT\_PTT6 = 1; //sound on

PWMDTY0 = 0x10;

}

if(PTAD\_PTAD0 == 0) {

offcount++;

PTT\_PTT6 = 0; //sound off

PWMDTY0 = 0x00;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; fdisp: Display the current page of letters and the input

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void fdisp()

{

send\_i(LCDCLR);

chgline(LINE1);

if(mode == 1) {

if(page == 0)

pmsglcd("ABCDEFGHIJKLMNOP", 1); //page 0

else if(page == 1)

pmsglcd("QRSTUVWXYZ BS EN", 1); //page 1

}

else {

pmsglcd(minput, 1);

print\_c(' ');

}

chgline(LINE2);

pmsglcd(input, 2);

}

//morse - "input" is converted into morse code

void morse(void) {

char morsecode;

int part1; //splits the letter into 4 parts

int part2; //each part corresponds to a dot, dash, or nothing

int part3;

int part4;

for(i = 0; i < index; i++) {

morsecode = morsetable[input[i] - 'A']; //split char

part1 = (morsecode & 0xC0) >> 6;

part2 = (morsecode & 0x30) >> 4;

part3 = (morsecode & 0x0C) >> 2;

part4 = morsecode & 0x03;

if(input[i] == ' ')

timerwait(1); //delay for a space

else {

if(part1 != 0) { //outputs each part

morseout(part1);

}

if(part2 != 0) {

timerwait(1);

morseout(part2);

}

if(part3 != 0) {

timerwait(1);

morseout(part3);

}

if(part3 != 0) {

timerwait(1);

morseout(part4);

}

}

timerwait(3); //delay between chararacters

}

}

/\*morseout

1: led on for 1 .

2: led on for 3 -

\*/

void morseout(int length) {

if(length == 1) { // output '.'

PTT\_PTT6 = 1;

PWMDTY0 = 0x10;

timerwait(1);

PTT\_PTT6 = 0;

PWMDTY0 = 0x00;

}

if(length == 2) { // output '-'

PWMDTY0 = 0x10;

PTT\_PTT6 = 1;

timerwait(3);

PTT\_PTT6 = 0;

PWMDTY0 = 0x00;

}

}

//clearinput - clears input string

void clearinput(void) {

for(i = 0; i < 16; i++)

input[i] = ' ';

index = 0;

}

//timerwait - loop until a certain amount of interrupts occur

void timerwait(int cycle) {

twait = cycle;

TIE = 0x80; //enable timer interrupts

while(twait > 0) {

asm {

nop

}

}

}

//timerwait2 - loop and count how many cycles the button is pushed

void timerwait2(int cycles) {

oncount = 0;

offcount = 0;

TIE = 0x80;

while( (oncount + offcount) < cycles ) {

asm {

nop

}

}

}

//getchar - converts morse input to character

void getchar(void) {

in[0] = 0; //morse input is combined into a input for table2

in[1] = 0;

in[2] = 0;

in[3] = 0;

for(i = 0; i<4; i++)

minput[i] = ' ';

for(i = 0; i < 4; i++) {

if(charpb == 1) { //break if entering mode 1

endflag = 1;

break;

}

timerwait2(waitmod\*2);

if(oncount >= waitmod\*3/4 && oncount <= waitmod\*3/2) { //if the input is the length of a '.'

minput[i] = '.';

fdisp();

in[i] = 1;

}

else if(oncount >= waitmod\*3/2) { //if the input is the length of a '-'

timerwait2(waitmod\*2);

minput[i] = '-';

fdisp();

in[i] = 2;

}

else { //if no entry, then convert to character

test = morsetable2[(in[0] << 6) | (in[1] << 4) | (in[2] << 2) | in[3]];

if(test != 0) { //checks the entry matches a value in table2

input[index++] = test;

fdisp();

chgline(LINE2 + index);

checkend();

break;

}

else

getcharerror();

}

}

}

//getcharerror - enter if the input doesn't match a value in the table

// shows error, restarts character entry

void getcharerror(void) {

pmsglcd("Error ", 1);

for(i = 0; i < 5; i++){

in[i] = 0;

minput[i] = ' ';

}

i = -1;

while( PTAD\_PTAD0 == 0 && !charpb ) {

if(leftpb == 1 && index > 0){

leftpb = 0;

input[--index] = ' ';

fdisp();

}

} //wait for user

pmsglcd(" ", 1);

}

//checkend - check for space or end of stream in morse input

void checkend(void) {

TIE = 0x80;

oncount = 0;

offcount = 0;

i = 1;

while(i < 40) { //counts number of cycles that the button is off

if(leftpb == 1 && index > 0){

leftpb = 0;

input[index - 1] = ' ';

index--;

fdisp();

i = 0;

}

timerwait2(waitmod/4);

if(oncount >= waitmod/8)

break;

else

i++;

}

if(i >= 39) //if over 10 cycles, break

endflag = 1;

else if(i >= 20) //if over 5 cycles, print space

input[index++] = ' ';

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; shiftout: Transmits the contents of register A to external shift

; register using the SPI. It should shift MSB first.

;

; MISO = PM[4]

; SCK = PM[5]

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void shiftout(byte outbyte)

{

while(!SPISR\_SPTEF){}

SPIDR = outbyte;

asm{ ldaa #11 ;//wait for 30 cycles for SPI data to shift out

spwait: dbne a , spwait

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; lcdwait: Delay for 2 ms

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void lcdwait()

{

asm{

psha

pshb

pshc

ldaa #$FF

ldab #63

lp2srt: dbeq b , lpend

lp1srt: dbne a , lp1srt

bra lp2srt

lpend: pulc

pulb

pula

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; send\_byte: writes contents of register A to the LCD

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void send\_byte(byte outbyte)

{

//Shift out character

//Pulse LCD clock line low->high

//Wait 2 ms for LCD to process data

shiftout(outbyte);

PTT\_PTT4 = 0;

PTT\_PTT4 = 1;

PTT\_PTT4 = 0;

lcdwait();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; send\_i: Sends instruction passed in register A to LCD

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void send\_i(byte inst)

{

PTT\_PTT2 = 0; //Set the register select line low (instruction data)

send\_byte(inst); //Send byte

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; chgline: Move LCD cursor to the cursor position passed in A

; NOTE: Cursor positions are encoded in the LINE1/LINE2 variables

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned char cc;

void chgline(byte curpos)

{

send\_i((char)CURMOV);

cc = curpos;

send\_i(curpos);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; print\_c: Print character passed in register A on LCD ,

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void print\_c(byte outbyte)

{

PTT\_PTT2 = 1; //Set the register select line back to high (actual data)

send\_byte(outbyte);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; pmsglcd: pmsg, now for the LCD! Expect characters to be passed

; by call. Registers should return unmodified. Should use

; print\_c to print characters.

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void pmsglcd(char \* outString, int lneint)

{

int cntt;

cntt = 0;

if(lneint == 1){chgline((char)LINE1);}

else if(lneint == 2){chgline((char)LINE2);}

while(outString[cntt] != 0){

print\_c(outString[cntt]);

cntt++;

}

}

**Appendix D:**

**Packaging Design**



Figure 5.1: Overview of the carbon box

The outside of the box was covered with black cardboard

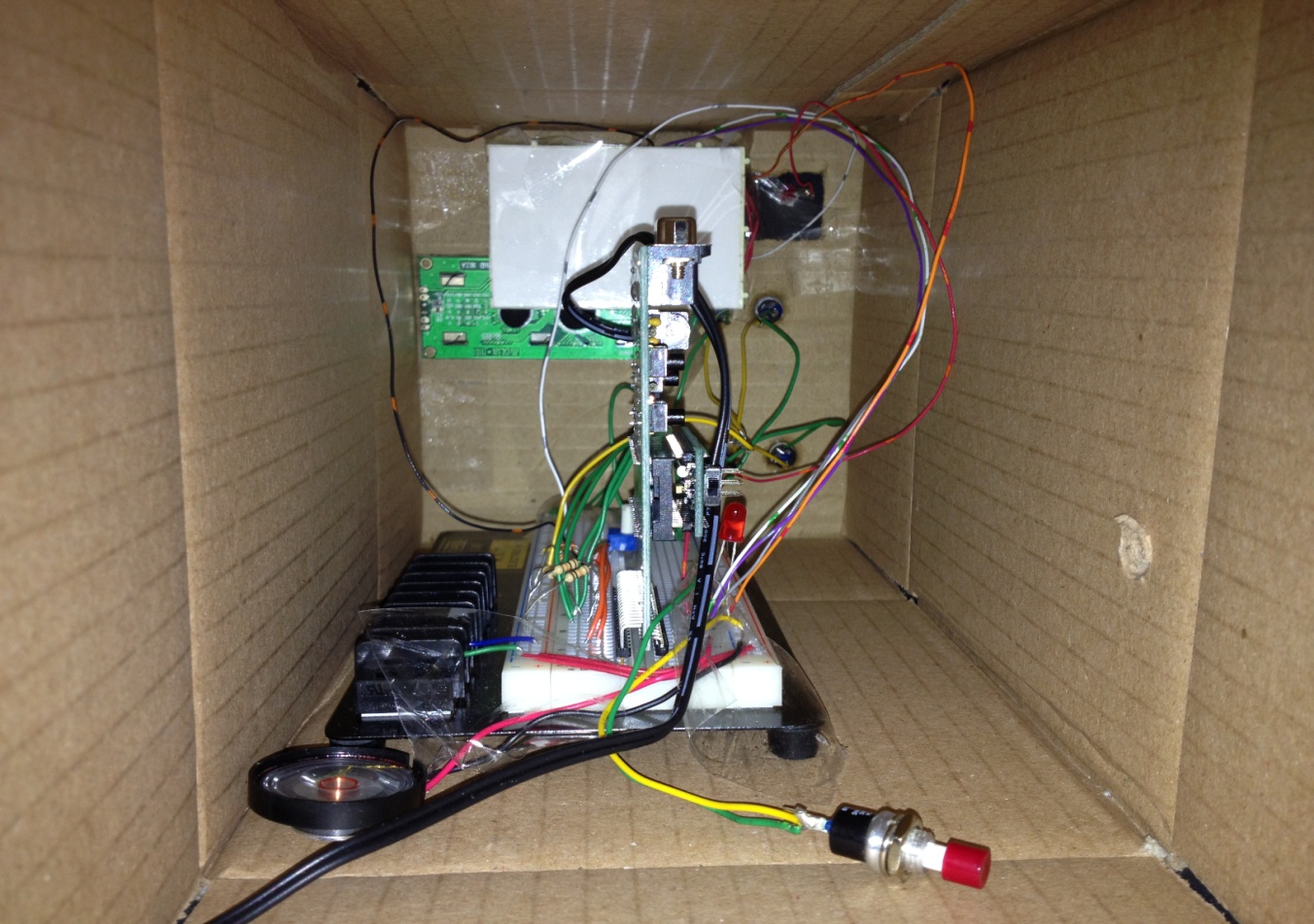


Figure 5.2: Inside wiring on the breadboard and microcontroller

Breadboard was tapped on the bottom, and the LCD screen and all pushbuttons were glued on the front display pannel