Morse Code Chat

Author: Alexander Powers
Team Member: Benjamin Mitchinson
May 3, 2019, ECE:3360 Embedded Systems

1. Introduction

Our project was the design and creation of a discrete Morse Code communication system programmed on two ATmega88PA development boards. Using our solution, two users are able to communicate with one another by tapping out Morse Code on a button, which is then transmitted over Bluetooth to the LCD of the other user. This project binds the two devices together, so that they only communicate with one another.

1.A Background

Morse Code is a series of short and long pulses canonically known as 'dit' and 'dah' for short and long. The entire alphabet is composed of these dits and dahs in varying combinations and lengths, as seen to the right. This language is perfect for discrete communication, as it is non-verbal and can be easily decomposed for serial transmission.

A	•-	N	-•	0	-----
B	-•••	O	---	1	•----
C	-•-•	P	•--•	2	••---
D	-••	Q	--•-	3	•••--
E	•	R	•-•	4	••••-
F	••-•	S	•••	5	•••••
G	--•	T	-	6	-••••
H	••••	U	••-	7	--•••
I	••	V	•••-	8	---••
J	•---	W	•--	9	----•
K	-•-	X	-••-	.	•-•-•-
L	•-••	Y	-•--	,	--••--
M	--	Z	--••	?	••--••

Figure 1. Morse Code Alphabet

1.B Goals & Specification

Our goal was to quickly and accurately interpret Morse Code and transmit a full message in both directions a distance of 10 meters. Our acceptance criteria consisted of both qualitative and quantitative performance evaluations. We placed requirements on the detection rate of dits and dahs, requiring that the Farnsworth unit be used as the standard. We also had qualitative expectations that the user interface would be intuitive and responsive to user taps. Another

qualitative expectation was automatic pairing, such that the user only has to power on the devices, and they will transmit correctly.

2. Implementation

This system has 3 sub components:

1. The Morse Code reading and decoding
2. The LCD display functionalities
3. The serial communication & configuration of the HC-05 Bluetooth module

Modeling this project as a set of subcomponents that all communicate, enabled us to test and debug individual components and validate their correctness before integrating the entire system. This led to more effective debugging and cleaner functionalities.

2.A Overview

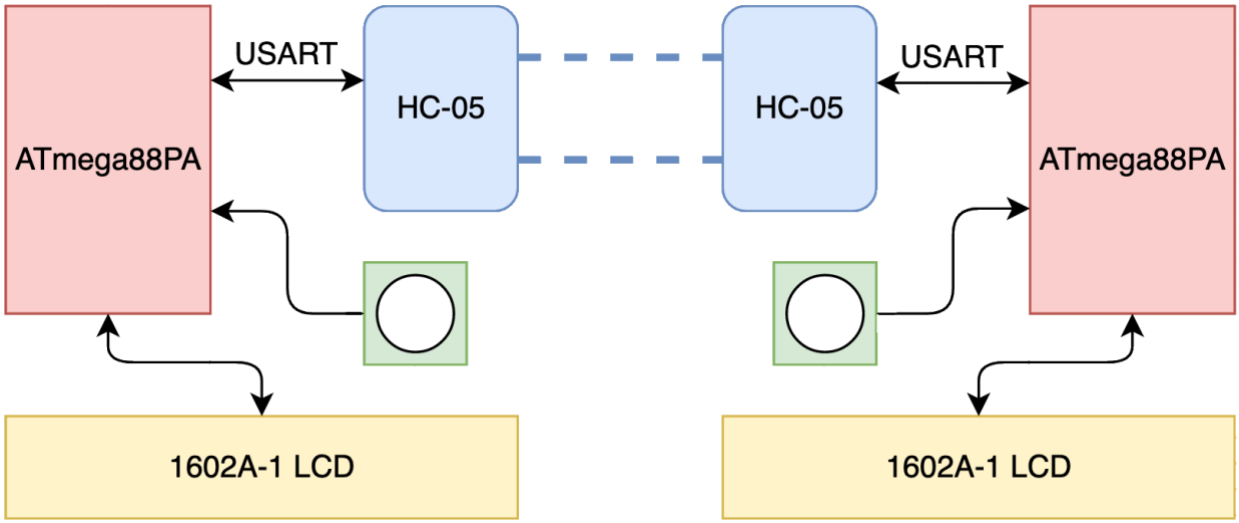

Figure 2. Whole System Dataflow
(Button: Green, ATmega88PA: Red, LCD: Yellow, Bluetooth: Blue)

Our system consists of two symmetrical, ATmega88 development boards, with their corresponding LCDs, Bluetooth modules and buttons, as shown above. This system allows for asynchronous communication between the two boards. The Bluetooth modules act as the pipe through which all data flows. The buttons are the sources of new information. And the LCDs are the final display. The ATmega88PA directs all of this traffic and keeps the whole system coordinated.

2.B Hardware Description

We decided to incorporate a new system peripheral for our final project, which was the HC-05 Bluetooth module. In addition to this new peripheral hardware, we used the 1602A-1 LCD, the ATmega88PA development board, and a hardware debounced button. All of these elements are picture below in the single device hardware schematic.

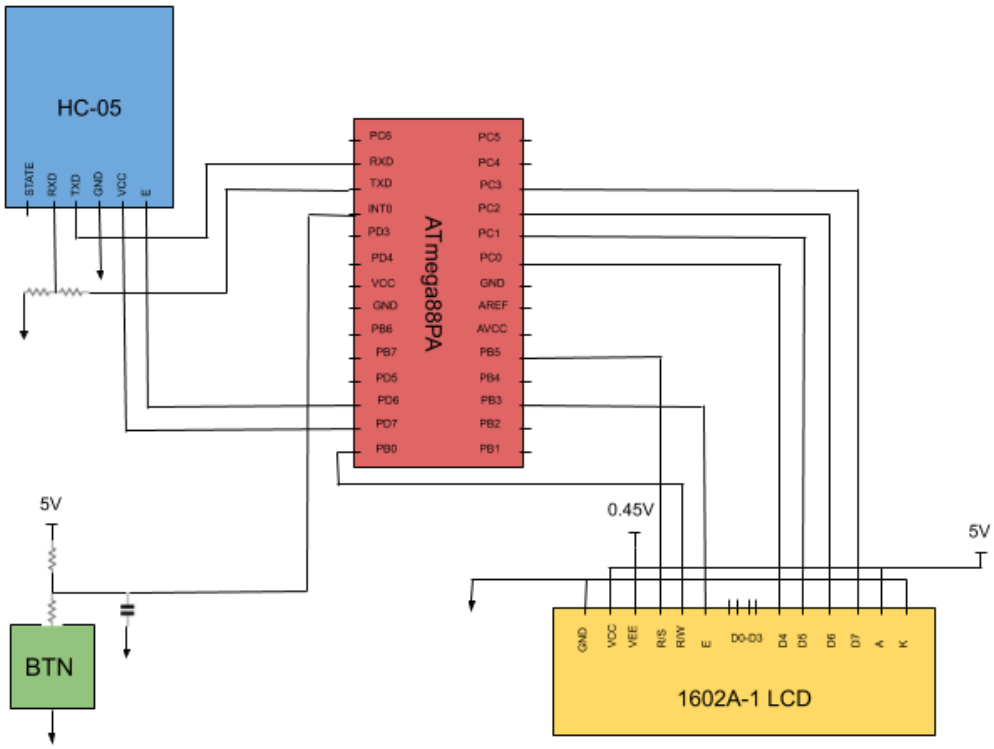

Figure 3. Single Device Hardware Schematic

Our LCD was configured in 4-bit data transmission mode. The LCD and ATmega88PA configuration were the same as lab4 but with the addition of both read and write capabilities. The HC-05 Bluetooth configuration was mainly software; however, it is important to note that we decided to use a voltage divider on the RXD pin of the HC-05 in order to drop the voltage from 5V down to 3V. The button was hardware debounce to avoid noise introduced by the users taps.

All of the subcomponents for one device were linked directly to the ATmega88PA, which acted as the communication hub of sorts. The button was interpreted and read by the ATmega88PA via interrupts. The USART transmission also happened via the interpretation interrupt. This left the body of the program free to wait for communications from the other device. The microcontroller's 16bit timer and pin change interrupts were crucial to the success of this multi-component system.

2.C Software Description

In order to accurately interpret the Morse Code, we used a sampling rate known as the Farnsworth timing in order to determine the difference between a dit and a dah. These taps were then stored in an array, where 1 is a dit, two is a dah, and the rest is padded with zeros. This array could then be read as a base three number, for example the character 'F' would be $(112100)_3$ which is $(387)_{10}$. This base ten number is a `key` for our decomposed map data structure. This base ten key is associated with exactly one Morse Code character, and all invalid Morse Code is caught and interpreted accordingly.

Our display utilizes an external library for LCD manipulation. This library allowed us to generate concise sub-routines for each component of the display. Every LCD related sub-routine utilizes a *lcd_gotoxy()* function before writing data. This avoids the need for managing the cursor position at all points in the program execution. This extra functionality cleaned up our code and made our solution more comprehendible.

Figure 4. LCD Display During Use

The above is an example of what the LCD looks like during use. The top line is dedicated to the incoming message from the other device. The bottom right shows you whether you just tapped a dit or a dah. This is particularly helpful while developing a feel for the timing of our device. The bottom left is the current interpretation of the character as a base three number, this six-character array updates in real-time as the dits and dahs are tapped.

We utilize the same serial communication techniques from lab five in our final project. The communication between the ATmega88PA and the Bluetooth HC-05 is USART. Once we completed the serial communication, and verified its functionality via SecureCRT, we moved on to the hardest aspect of the project; the HC-05 Bluetooth module configuration. The Bluetooth module is only able to be sent configuration commands in what is known as 'Command Mode'. In order to enter command mode, the enable pin, *E* above, must be toggled during the powering of the device. Once this is done, the module will respond to a variety of commands in the form of "AT+[COMMAND]?" and "AT+[COMMAND]=[VALUE]", where COMMAND and VALUE vary for different configurations and queries. These command strings allowed us to configure one HC-05 as a slave device and the other as a master device. This master slave relationship allowed the two devices to communicate. In addition, we were also able to set the "BIND" addresses of each of the modules, so that they were bound to each other. This binding was enforced by another configuration variable known as "CMODE". The CMODE configuration we chose forced the devices to only communicate with the address they were bound to.

Each of these software components worked independently thanks to our use of interrupts. This project would not have been nearly as successful or responsive without them. Interrupts allowed us to read the button press without breaking from our USART two-way communication pattern in main.

3. Experimental Methods

As with the nature of our project, most of our evaluation methods were qualitative. The system either performs as expected or it does not. We were able to evaluate each sub component of our system, as stated above, qualitatively. The button, LCD, and Bluetooth all work independently, and were tested independently. I briefly used an oscilloscope to validate the USART communication and button press debouncing functionalities.

4. Results

Our final product achieved all of our originally set expectations and specifications. We were able to attain fast, and accurate two-way communication, at the limiting distance of the Bluetooth module (10 meters). Ben and I were very happy with this result. We could validate that all of the sub systems worked independently as well as in conjunction, with qualitative testing, and oscilloscope measurements of the Morse Code taps, and USART serial transmissions. Below is a side by side picture of our final working product.

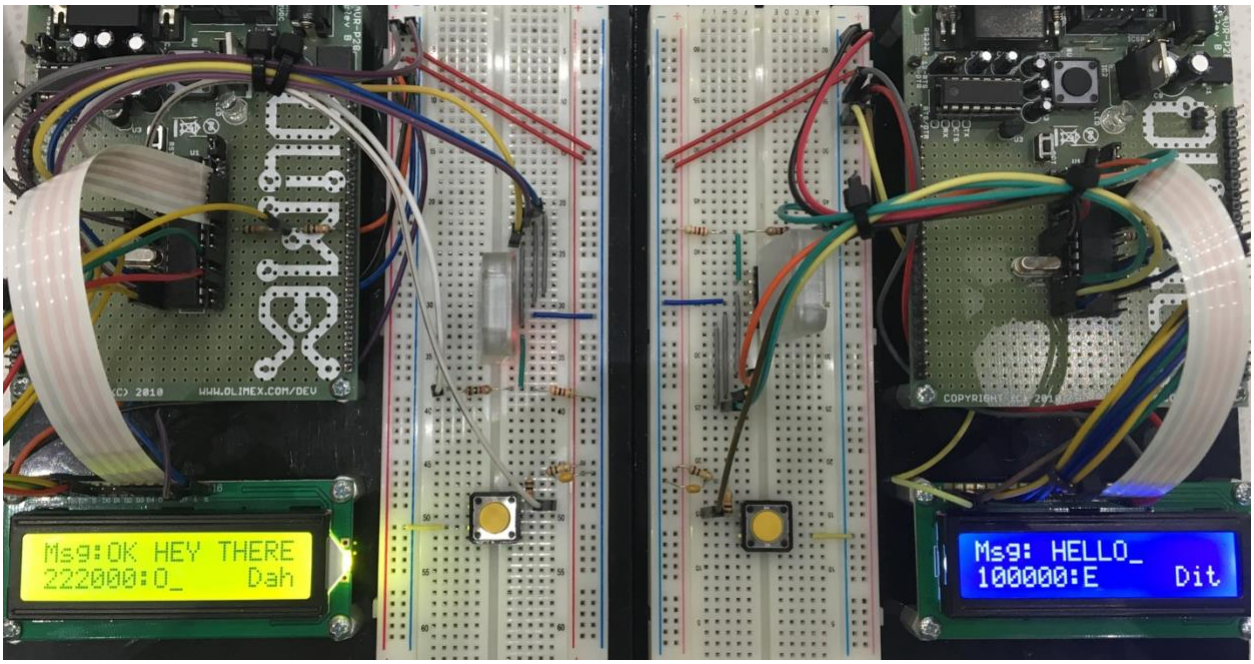

Figure 5. Final Product Side by Side

5. Discussion

This device met and exceeded all of our performance expectations. The Bluetooth connections were fast and accurate. We achieved every aspect of the design goal that we had set out

for ourselves. Our solution is not the only possible solution of course. We could have utilized any number of different adaptive sampling techniques to make the interpretation more responsive. We could have utilized a transmission mechanism such as Radio Frequency communication that may have been lighter weight than Bluetooth. Our current project is limited in that we can only display 12 characters along the top row of the LCD. If we were to continue this project, I would like to utilize EEPROM to store a message history, as well as introducing an adaptive sampling technique so that the Morse Code decoding occurs at any rate for new or experienced Morse Code users.

6. Conclusion

We were able to successfully execute our initial project design without the need for additional modifications or simplifications. This project has given me a lot of confidence in my abilities when it comes to embedded systems, and I would like to continue this project in order to improve it by making it handheld and battery powered.

Our final project was the culmination of all of the skills that we have developed in the course so far. We have grown so much since the days of wiring up Lab Kit A for the first time. Our level of understanding and competency in embedded systems has increased drastically over the span of this course. We are now able to integrate multiple C and AVR libraries, learn to configure new hardware and develop a feasible product in nothing more than a couple of weeks. Going forward the lessons that I have learned from this project and course will influence my design choices and goals as an engineer.

7. Acknowledgements

7.A Source Code

```
// ////////////////////////////////////////////////////////////////
// Main C Final Project file for ECE:3360 (Embedded Systems)
// Spring 2019, The University of Iowa.
//
// Desc: Morse Code Transmission Device (Client + Receiver Pair)
//
// Authors: B. Mitchinson, A. Powers
// ////////////////////////////////////////////////////////////////
// Setting CPU Clock Speed
#ifndef F_CPU
#define F_CPU 8000000UL // 8 MHz -> CProgramming Notes, Slide 10
#endif
// ///////////////////////
// Imports
// ///////////////////////////////
```

```
#include <avr/io.h>
#include <stdio.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "lcd.h"
#include "USART_RS232_H_file.h"
// ////////////////////////////////
// LCD #define statements
// ////////////////////////////////
// Command mode vs Data mode for LCD
#ifndef RS
#define RS  5
#endif
// Enable pin of LCD
#ifndef E
#define E   3
#endif
// Data pin 4 of LCD
#ifndef D4
#define D4  0
#endif
// Data pin 5 of LCD
#ifndef D5
#define D5  1
#endif
// Data pin 6 of LCD
#ifndef D6
#define D6  2
#endif
// Data pin 7 of LCD
#ifndef D7
#define D7  3
#endif
// LED is on PORTC pin 5
#ifndef LED_PORT
#define LED_PORT  5
#endif
// maximal morse code array length
#ifndef MORSE_ARR_LEN
#define MORSE_ARR_LEN 6
```

```
#endif
// Baud rate of USART communication
#ifndef BAUDRATE
#define BAUDRATE 9600
#endif
// Port that the bluetooth pins are located on
#ifndef BLUE_TOOTH_PORT
#define BLUE_TOOTH_PORT PORTD
#endif
// Voltage pin of the HC-05 bluetooth module
#ifndef BLUE_TOOTH_VCC
#define BLUE_TOOTH_VCC 7
#endif
// Enable pin of the HC-05 bluetooth module
#ifndef BLUE_TOOTH_E
#define BLUE_TOOTH_E 6
#endif
// ////////////////////////////////
// Globals
// ////////////////////////////////
// Farnsworth unit speed
#ifndef F_UNIT
#define F_UNIT 3.0 // 4 units per second (250ms)
#endif
// samplings per unit
#ifndef SAMPLES_PER_UNIT
#define SAMPLES_PER_UNIT 26 // @ 80/second samples 80/fUnit = 26
#endif
// Character mapping (dictionary out of two array cross order indexed)
#ifndef TOTAL_CHARS
#define TOTAL_CHARS 40
#endif
int keys[TOTAL_CHARS] = {243, 486, 405, 360, 324, 648, 567, 594, 675, 621, 702, 432, 351, 378,
459, 603, 630, 387, 477, 441, 468, 693, 369, 612, 639, 684, 726, 483, 402, 375, 366, 363, 606, 687,
714, 723, 692, 455, 400, 364};
char chars[TOTAL_CHARS] = {'E', 'T', 'A', 'H', 'I', 'M', 'N', 'D', 'G', 'K', 'O', 'R', 'S', 'U', 'W', 'B', 'C',
'F', 'J', 'L', 'P', 'Q', 'V', 'X', 'Y', 'Z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ',', '.', '?', ' '};
// Input
volatile int input[MORSE_ARR_LEN] = {0,0,0,0,0,0};
volatile int inputIndex = 0;
```

```
char too_long = 0x00;
float downSamples = 0;
float upSamples = 0;
// State for edge detection
char prevButtonDown = 0;
// Boolean to improve entry
char justFinishedChar = 1;
//  Button booleans
char buttonDown = 0;
// ////////////////////////////////
// Prototypes
// ////////////////////////////////
char hash_inputs(void);
void lcd_show_input_arr(void);
void lcd_clr_top(void);
void lcd_clr_bot(void);
void lcd_dah(void);
void lcd_dit(void);
void lcd_bad(void);
void reset_input_arr(void);
// bluetooth functions
void power_on_hc05(void);
void power_off_hc05(void);
void enable_on_hc05(void);
void enable_off_hc05(void);
void blue_tooth_to_command_mode(void);
////////////////////////////////////
// Timer Config (For button sampling to register dits and dahs)
// ////////////////////////////////
void timer1_init(void){
        // 8,000,000 / 256 = 31250 ticks per second.
        // We need timer to range from 0 to 31250
        //    @ a 1/256. It's overflow then occurs
        //    once every second.
        // 1  time  a second:            31250 : 0x7A12
        // Other scaled calculations
        // 40 times a second: 31250/40 =   782 : 0x30D
        // 80 times a second: 31250/80 =   391 : 0x187
        TCCR1B = TCCR1B |  1 << CS12;   // 256 Pre-scale
        TCCR1B = TCCR1B |  1 << WGM12;  // CTC Mode: Clear timer on compare mode
```

```
	// When TCNT1 (Counter index) matches OCR1A, it's reset to zero
	// and the OCF1A Interrupt Flag is Set. OCF1A Automatically cleared
	OCR1A = 0x187; // Set the top
	TIMSK1 = TIMSK1 |  1 << OCIE1A; // Output Compare A Match Interrupt Enable
}
// ////////////////////////////////
// Interrupt Configuration
// ////////////////////////////////
// button press interrupt
ISR(INT0_vect){
	PINC |= (1<<5); // Toggle Light
}
// Timer1 Overflow ISR
ISR(TIMER1_COMPA_vect){
	buttonDown = !(PIND & (1<<2));
	if (buttonDown) {
		downSamples += 1;
		if (downSamples >= 6 * SAMPLES_PER_UNIT){
			justFinishedChar = 1;
			reset_input_arr();
			lcd_show_input_arr();
			too_long = 0x01;
		}
	}
	else {
		upSamples += 1;
		if ((upSamples >= 5 * SAMPLES_PER_UNIT) && (!justFinishedChar)) {
			lcd_show_input_arr();
			char inputChar = hash_inputs();
			lcd_gotoxy(6,1);
			lcd_putc(':');
			lcd_putc(inputChar);
			if (inputChar != '!'){
				USART_TxChar(inputChar);
			}
			justFinishedChar = 1;
			reset_input_arr();
		}
	}
	if ((buttonDown != prevButtonDown) && (buttonDown)) { // just pressed
```

```
                justFinishedChar = 0;
                downSamples = 0;
                upSamples = 0;
        }
        else if ((buttonDown != prevButtonDown) && (!buttonDown) && (inputIndex < 6)) { //
just released + have room for more entries
                if (too_long){
                        too_long = 0x00;
                }
                else if (downSamples >= 1 * SAMPLES_PER_UNIT){
                        lcd_dah();
                        input[inputIndex] = 2;
                        inputIndex++;
                        lcd_show_input_arr();
                }
                else { //(downSamples >= 1 * samplesPerUnit){
                        lcd_dit();
                        input[inputIndex] = 1;
                        inputIndex++;
                        lcd_show_input_arr();
                }
                downSamples = 0;
                upSamples = 0;
        }

        // store state for next operation, in order to identify pos and neg edges
        prevButtonDown = buttonDown;
        if (too_long == 0x01){
                reset_input_arr();
                lcd_show_input_arr();
        }
}
// ////////////////////////////////
// Main Loop
// ////////////////////////////////
int main(void)
{
   //LCD Data Direction Configuration
   DDRC |= (1<<D4) | (1<<D5) | (1<<D6) | (1<<D7) ;
        DDRB |= (1<<RS) | (1<<E);
```

```
        DDRD |= (1<<PD6) | (1<<PD7);
        // Initial LCD Config
        lcd_init(LCD_DISP_ON_CURSOR);
        lcd_gotoxy(0,0);
        lcd_puts("Msg:");
        // Turn off the LED to use for button feedback
        PORTC |= (1<<5) ;
    // set interrupt configurations
    DDRC |= 1 << 5;
    EICRA |= (1<<ISC00);
    EIMSK |= (1<<INT0);

    // enable interrupts
    _delay_ms(50);
        sei();
        // Turn on sampling timer
        timer1_init();
        USART_Init(BAUDRATE); // initialize USART with 9600 baud rate
        blue_tooth_to_command_mode();
        //SUSART_SendString("AT\r\n");
        //USART_SendString("AT+NAME=M\r\n");
        //USART_SendString("AT+NAME?\r\n");
        //USART_SendString("AT+ADDR?\r\n");
        //USART_SendString("AT+ROLE?\r\n");
        //USART_SendString("AT+ROLE=1\r\n");
        //USART_SendString("AT+BIND=14,3,5fa85\r\n"); // Addr of slave: used to program
master (ted's)
        //USART_SendString("AT+BIND=14,3,5f6f4\r\n"); // Addr of master: used to program
slave (ours)
        //USART_SendString("AT+BIND?\r\n");
        //USART_SendString("AT+CMODE=0\r\n");
        char data_in;
        int num_chars = 4;
    // infinite loop
    while (1)
    {
                // this command is blocking, so all interrupts happen here
                data_in = USART_RxChar();
                // disable interrupts while writing to LCD
                cli();
```

```
                // check if we need to wrap around to the beginning of the line
                if (num_chars > 15){
                        lcd_clr_top();
                        num_chars = 4;
                }
                lcd_gotoxy(num_chars, 0);
                if (data_in != '\r' && data_in != '\n'){
                        lcd_putc(data_in);
                        num_chars++;
                } else {
                        lcd_putc(' ');
                        num_chars++;
                }
                sei();
    }
}
// ////////////////////////////////
// https://www.geeksforgeeks.org/convert-base-decimal-vice-versa/
int toDecimal(volatile int *arr, int base)
{
    int power = 1;
    int num = 0;
    for (int i = MORSE_ARR_LEN - 1; i >= 0; i--)
    {
        if (arr[i] >= base)
        {
            printf("Invalid Number");
            return -1;
        }
        num += arr[i] * power;
        power = power * base;
    }
    return num;
}
// ////////////////////////////////
// bluetooth functions
// ////////////////////////////////
void power_on_hc05(void) {
        BLUE_TOOTH_PORT |= (1<<BLUE_TOOTH_VCC);
}
```

```
void power_off_hc05(void) {
	BLUE_TOOTH_PORT &= ~(1<<BLUE_TOOTH_VCC);
}
void enable_on_hc05(void) {
	BLUE_TOOTH_PORT |= (1<<BLUE_TOOTH_E);
}
void enable_off_hc05(void) {
	BLUE_TOOTH_PORT &= ~(1<<BLUE_TOOTH_E);
}
void blue_tooth_to_command_mode(void) {
	enable_off_hc05();
	power_on_hc05();
	_delay_ms(2000);
	enable_on_hc05();
	_delay_ms(3000);
	enable_off_hc05();
}
// ////////////////////////////////
// Quick Printing Methods
// ////////////////////////////////
void lcd_clr_top(void){
	lcd_gotoxy(0,0);
	lcd_puts("Msg:            ");
}
void lcd_clr_bot(void){
	lcd_gotoxy(0,1);
	lcd_puts("               ");
}
void lcd_dah(void){
	lcd_gotoxy(13,1);
	lcd_puts("Dah");
}
void lcd_dit(void){
	lcd_gotoxy(13,1);
	lcd_puts("Dit");
}
void lcd_bad(void){
	lcd_gotoxy(13,1);
	lcd_puts("Bad");
}
```

```
void lcd_clear_bot_three(void){
	lcd_gotoxy(13,1);
	lcd_puts("   ");
}
// ////////////////////////////
// Reveal input array
void lcd_show_input_arr(void){
	// Showing the input array adds 20% to memory due to sprintf
	char buff[7];
	int i=0;
	int index = 0;
	for (i=0; i<6; i++)
		index += sprintf(&buff[index], "%d", input[i]);
	lcd_gotoxy(0, 1);
	lcd_puts("       ");
	lcd_gotoxy(0, 1);
	lcd_puts(buff);
}

// Get character from input array
char hash_inputs(void){
	int lookupKey = toDecimal(input, 3);
	int i = 0;
	while(keys[i] != lookupKey){
		i++;
		if (i >= TOTAL_CHARS){
			lcd_bad();
			return '!';
		}
	}
	return chars[i];
}
void reset_input_arr(void){
	for(int i = 0; i<6; i++){
		input[i] = 0;
	}
	inputIndex = 0;
}
/*
Farnsworth Timing (Altered a lot based on user preference. Pretty clear in the ISR)
```

dit: 1 unit (down)
dah: 3 units (down)
morse-character: 1 unit (up)
ascii-character: 3 units (up)
Word: 7 units or more (up)
*/

7.B External Libraries

- USART Library (https://www.electronicwings.com)
- LCD Library (http://tinyurl.com/peterfleury)
- Farnsworth Unit (https://morsecode.scphillips.com/timing.html)
- Standard AVR Libraries (interrupt.h/delay.h/io.h)

7.C Datasheets

- ATmega88PA Datasheet http://user.engineering.uiowa.edu/~rbeichel/lectures/es_s19/resources/ATmega88PA_Datasheet14.pdf
- HC-05 Datasheet http://www.electronicaestudio.com/docs/istd016A.pdf
- LCD Datasheet http://user.engineering.uiowa.edu/~rbeichel/lectures/es_s19/resources/1602A-1.doc.pdf