Kadeem Samuel
9/7/2018
2559-2971

Basic Micro-Controller Applications (IO, LCD Interface, AD) –
Part 2 Micro-Controller ADC and LCD Implementation of Ohmmeter

Introduction

The processor used for this module is the Atmel ATMega324P. This is an 8-bit processor that is programmed through the usage of the Atmel Ice, using the in-system programming method (functionally an SPI system). This microcontroller has a 1 MHz clock, 1k bytes of EEPROM, and 2K bytes of internal SRAM, in addition to 4 I/O ports, multiple timer/counters, and more. The goal of the design with this microcontroller is to measure the value of an arbitrary resistor, and output to the LCD screen. If the resistance is within 1K and 1M, then the resistance value will be output on the screen, otherwise "OUT OF RANGE" should be output on the screen.

Design

For the hardware, the Atmel Ice is used as the programmer for the ATMega device. This is connected to the ATMega as shown in Figure 1 in the appendix. A 4.7k pull-up resistor is used to connect the reset in place of a wire. In Atmel Studio 7, the interface mode is set to ISP for serial communication. A 5V supply is used to supply power to the circuit, which also represents the voltage level for logical high values. A layout of the design is provided in Figure 2, with the actual board shown in Figure 3. A potentiometer is used to adjust the contrast of the LCD screen. A voltage divider circuit is used to measure the voltage at ADC0, which is used to calculate the value of the measured resistor.

For the software, a flowchart of the code is shown in Figures 4 through 6. It works as follows: the ADC and the LCD are initialized. While the ATMega is on, the LCD is cleared, then the ADC0 voltage value (in 10-bit) is measured five times, then averaged. This value is converted into a floating-point voltage value, which is then used to calculate the resistance of the measured resistor. Then, if the resistance is less than 1K or more than 1M, the LCD displays "OUT OF RANGE" by using the provided lcd_char command. Otherwise, the resistance value is converted to a string from a numerical value, and output to the LCD. The function out_string outputs each character of a string to the LCD, for simplicity. The full code is provided in Figures 7 – 11. The bill of materials is shown in Table 1, with the total cost coming to $106.08.

Conclusion

For lower resistance values, around 1-10K, the measured resistance was relatively accurate, within about +1% of the actual value. For larger resistance values, such as those around but less than 1M, the measured resistance values were within around -3% of the actual value. Between these extremes, the measured resistance value was accurate to less than 1% error. The difficulty with measuring such a large range of values is that, for a known resistance value of 10K, low resistance values will be measured accurately, whereas higher resistance values will be somewhat inconsistent. This is due to the nature of the voltage divider equation: Two close resistance values can be measured with accuracy if they are low, because the difference in measured voltage will be significant enough to distinguish the two. On the other hand, if the two resistance values are large, the change in the measured voltage could be too small for the 10-bit ADC to measure, resulting in an incorrect resistance measurement. Capping the resistance at 1M prevents this effect from being too prevalent, but its effects can still be seen be using resistors of resistance 1M, 1.05M, and 0.995M. Furthermore, the LCD must have a resistor between Vcc and the LED+ terminal to stabilize the current to the LCD, otherwise the circuit will fail.

Appendix

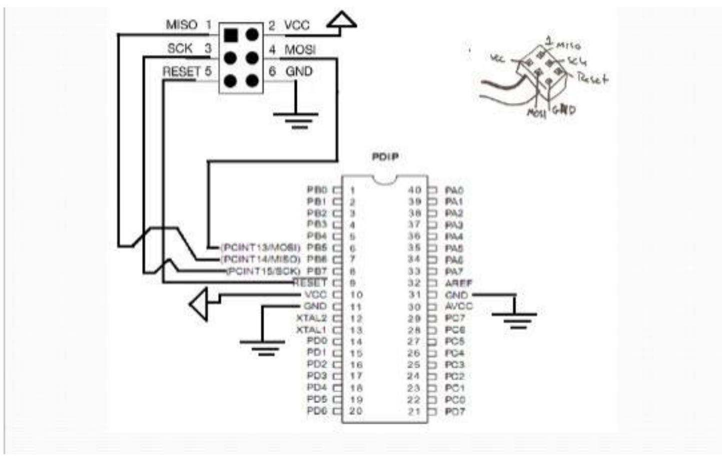

Figure 1: Pin Layout for Connecting Atmel Ice to ATMega324P, Red Wire is SCK

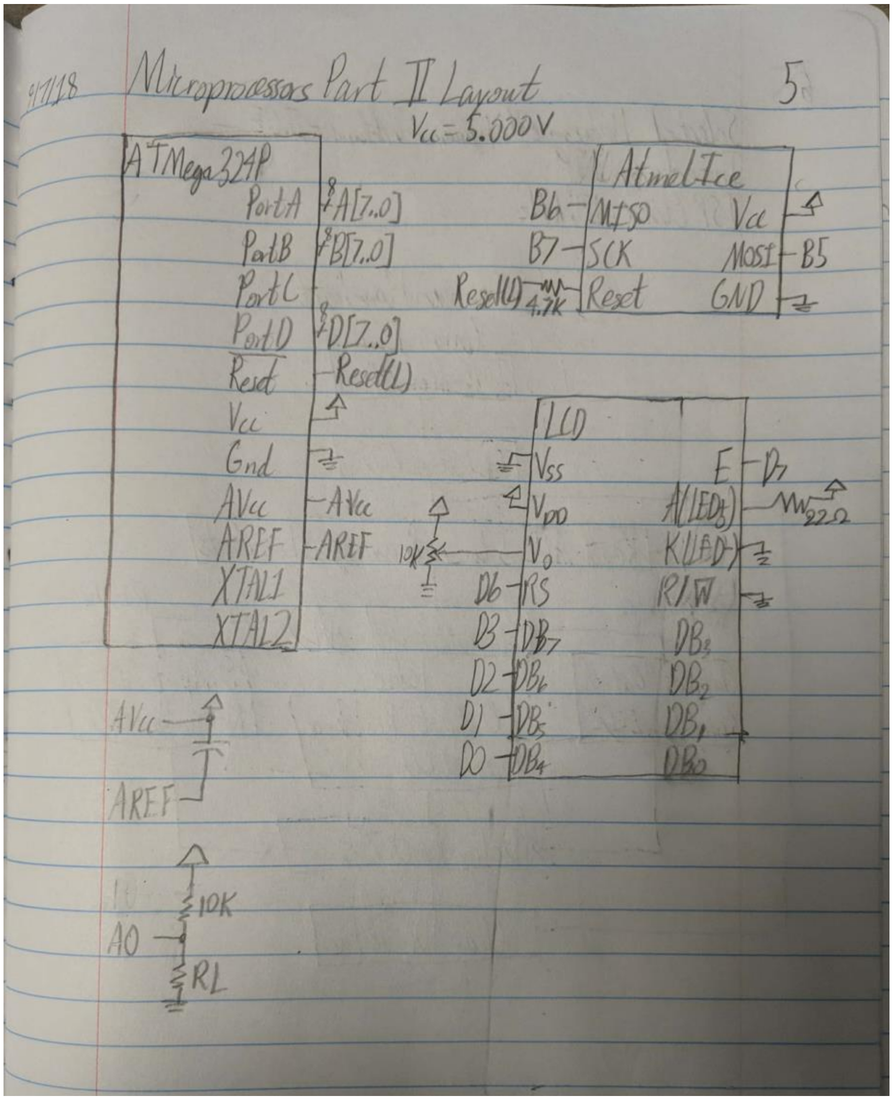

Figure 2: Hardware Connection Block Diagram of the Complete Circuit

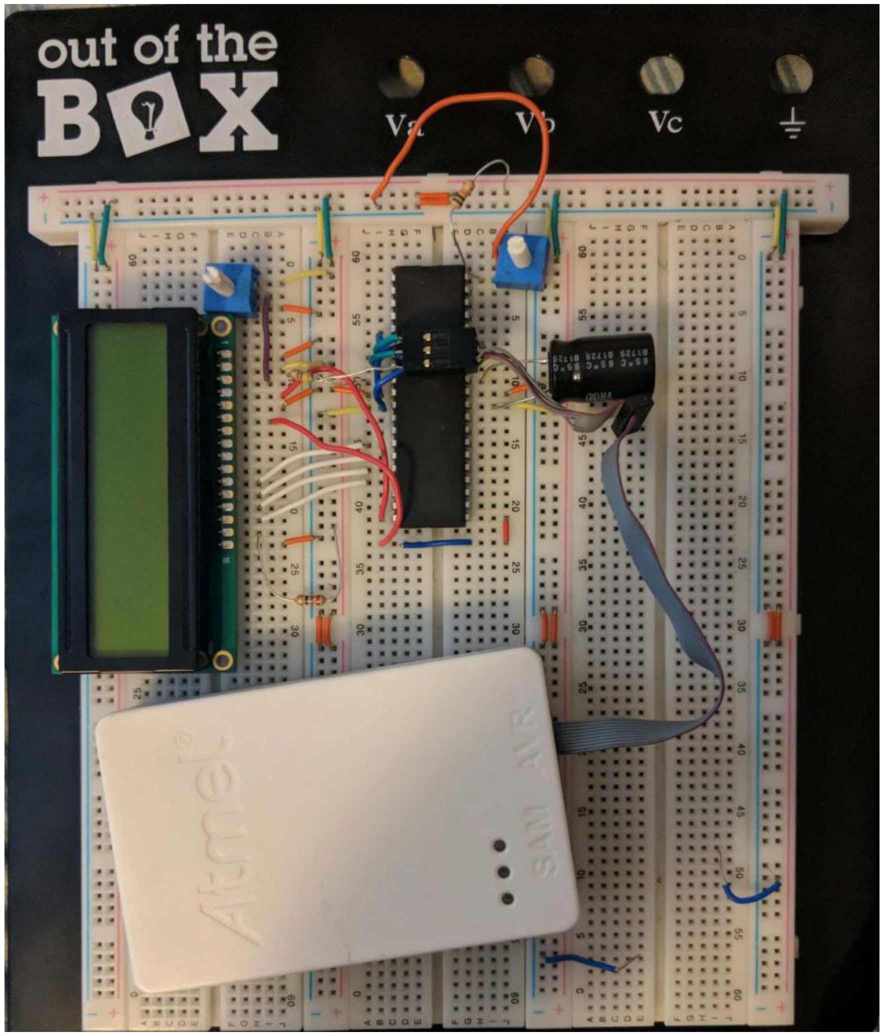

Figure 3: Actual Circuit Showing Atmel Ice (Bottom), ATMega324P (Center), and LCD (Left)

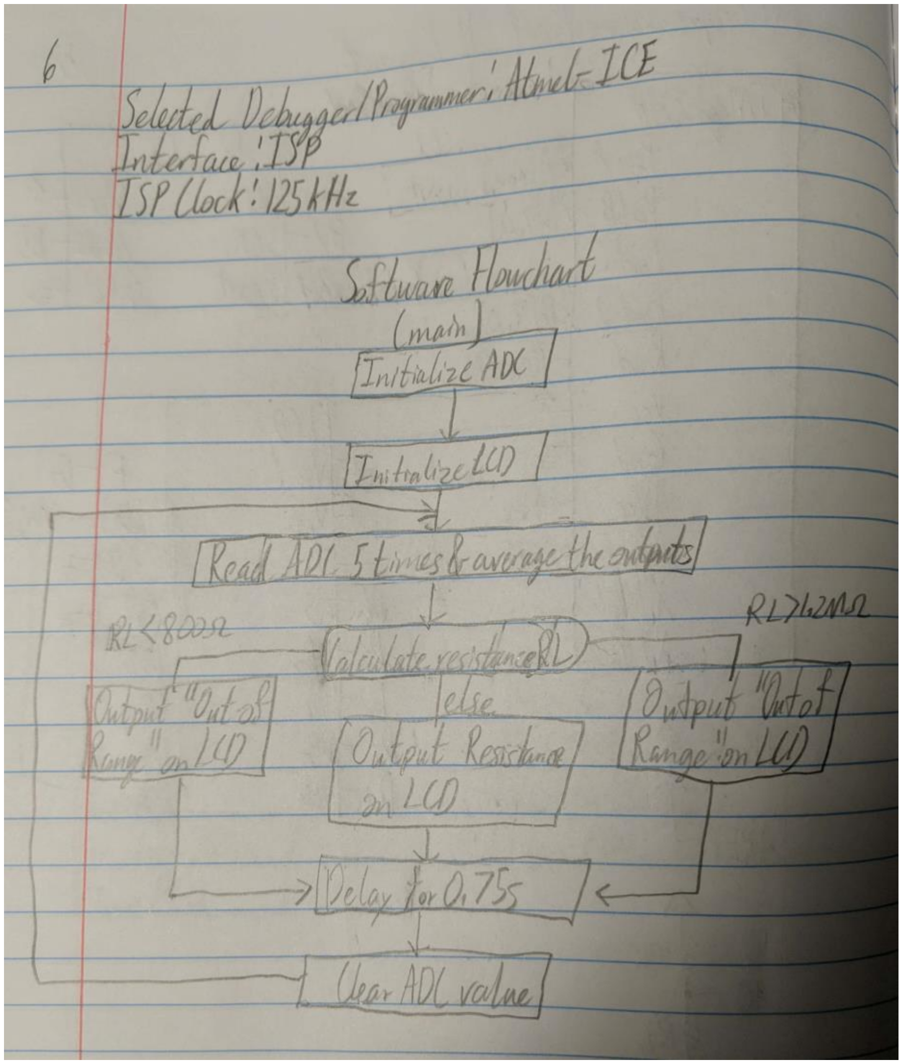

Figure 4: Software Flowchart, Page 1

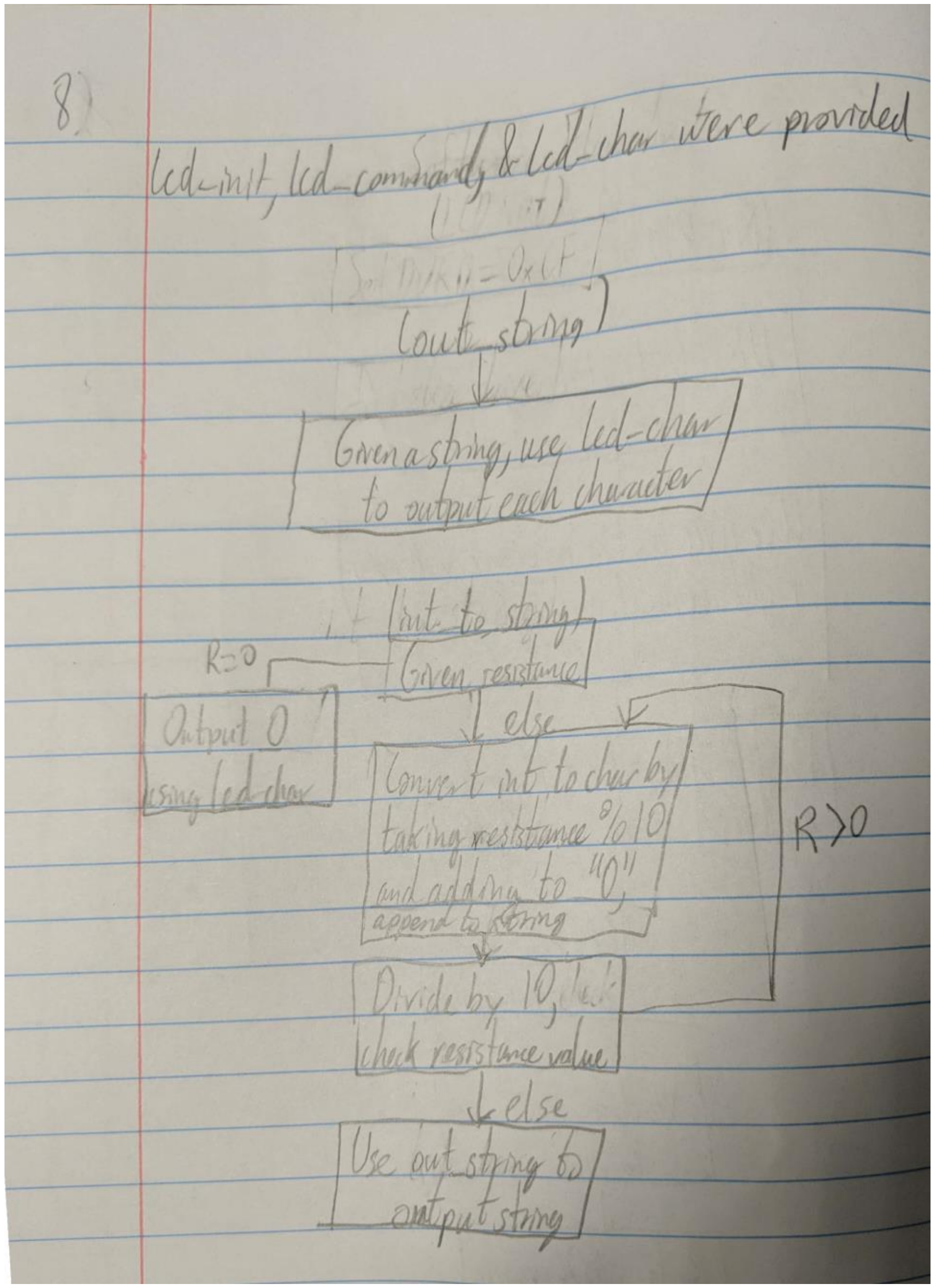

Figure 5: Software Flowchart, Page 2

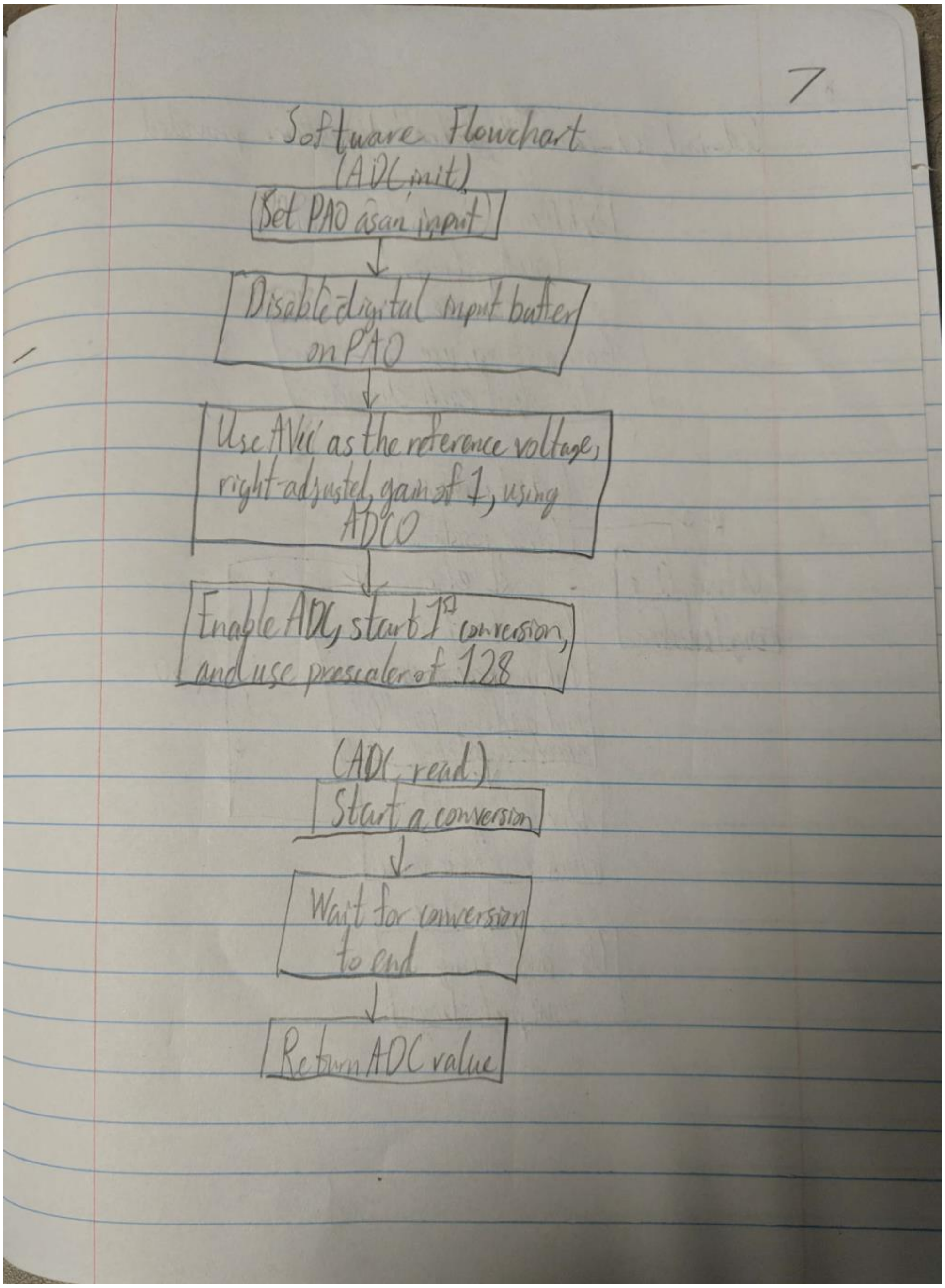

Figure 6: Software Flowchart, Page 3

```c
/*
 * Part2.c
 *
 * Created: 8/29/2018 4:55:48 PM
 *  Author: Kadeem Samuel
 *
 * The purpose of this program is to measure the resistance of a resistor, and display the resistance on an LCD screen
 */

#include <avr/io.h>
#include <util/delay.h>

void ADC_init(void);
void lcd_init(void);
void lcd_command(char);
void lcd_char(char);
void out_string(char s[]);
void int_to_string(unsigned long n);
uint16_t ADC_read(void);

//Math will be done in kOhms due to processor data size (8 bit), then numbers will be fed to LCD in Ohms
double volatile Rl = 9820.0; //Known resistance value, used in voltage divider, measured in kOhms
unsigned long volatile Rmeas = 0;//
char resistance[0];

int main (void)
{
    uint16_t volatile ADCValue = 0;
    ADC_init();
    lcd_init();
    while (1)
    {
        lcd_command(0x01); //Clear the LCD
        for(int volatile i = 1; i <= 5; ++i)
        {
            ADCValue += (uint16_t)(ADC_read()-1); //Store the value from the ADC, -1 used as a correction factor
        }
```

Figure 7: Program Code, Page 1

```c
        ADCValue = ADCValue/5;
        float volatile Voltage = (ADCValue * (5.0/1024.0)); //Determine the voltage value
        Rmeas = (unsigned long)(Voltage * Rl)/(5.0 - Voltage); //Measure the resistance value
        //The following are correction factors to account for inaccuracies made in different voltage ranges by the ADC:
        if(Rmeas < 800 || Rmeas > 1200000) //Values Shiming told me to use
        {
            out_string("OUT OF RANGE");
        }
        else
        {
            Rmeas = 1.001*Rmeas;
            out_string("R = ");
            int_to_string(Rmeas);
            out_string(" Ohms");
        }
        _delay_ms(750); //Delay so that the full message can appear on the LCD
        ADCValue = 0;
    }
}

void ADC_init(void)
{
    DDRA = 0x00; //PA is all inputs
    DIDR0 = (1<<ADC0D); //Disable digital input buffer on PA0 to reduce power consumption
    ADMUX = (1<<REFS0); //AVcc pin as reference, right adjusted format, gain of 1, using ADC0
    ADCSRA |= (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(ADPS0); //Enable the ADC, and start the first conversion, prescaler of 128
}

uint16_t ADC_read(void)
{
    ADCSRA |= (1<<ADSC);
    while(ADCSRA & (1<<ADSC));
    //Continuously poll the flag
    return (ADC);
}

void lcd_init(void)
```

Figure 8: Program Code, Page 2

```
void lcd_init(void)
{
    DDRD |= 0xCF;
    lcd_command(0x33); //Initialize LCD driver
    lcd_command(0x32); //Four bit mode
    lcd_command(0x2C); //2 Line Mode
    lcd_command(0x0C); //Display on, cursor off, blink off
    lcd_command(0x01); //Clear screen, cursor home
}

void lcd_command(char cmd)
{
    char temp = cmd;
    PORTD = 0; //Don't change the last bit to account for the ADC
    _delay_ms(5);
    cmd = ( (cmd & 0xF0) >> 4) | 0x80; //Write Upper Nibble (RS=0) E --> 1
    PORTD = cmd;
    _delay_ms(5);
    cmd ^= 0x80; //E = 0
    PORTD = cmd;
    _delay_ms(5);
    cmd = temp;
    cmd = ((cmd & 0x0F) | 0x80); //Write lower nibble (E = 1)
    PORTD = cmd;
    _delay_ms(5);
    cmd ^= 0x80;
    PORTD = cmd;
    _delay_ms(7);
}

void lcd_char (char data)
{
    char temp = data;
    PORTD = 0x40;
    _delay_us(100);
    data = ( (data & 0xF0) >> 4) | 0xC0;  //Write Upper Nibble (RS=1) E --> 1
    PORTD = data;
```

Figure 9: Program Code, Page 3

```
void lcd_char (char data)
{
    char temp = data;
    PORTD = 0x40;
    _delay_us(100);
    data = ( (data & 0xF0) >> 4) | 0xC0;  //Write Upper Nibble (RS=1) E --> 1
    PORTD = data;
    _delay_us(100);
    data ^= 0x80; //  E --> 0
    PORTD = data;
    _delay_us(100);
    data = temp;
    data = ( (data & 0x0F) ) | 0xC0; //Write Lower Nibble (RS=1) E --> 1
    PORTD = data;
    _delay_us(100);
    data ^= 0x80; //E --> 0
    PORTD = data;
    _delay_us(100);
}

void out_string(char s[])
{
    for (int volatile i = 0; i < strlen(s); ++i)
    {
        lcd_char(s[i]);
    }
}

void int_to_string(unsigned long n)
{
    //Using long data type to not run into math issues with 8 bit processor
    char a[7] = {0};
    int volatile i=0;
    if(n == 0)
    {
        //Case where the voltage is shorted
        lcd_char('0');
```

Figure 10: Program Code, Page 4

```c
void int_to_string(unsigned long n)
{
    //Using long data type to not run into math issues with 8 bit processor
    char a[7] = {0};
    int volatile i=0;
    if(n == 0)
    {
        //Case where the voltage is shorted
        lcd_char('0');
        return;
    }
    else

    {
        while(n > 0)
        {
            //Add the number at each location to the char form of 0, to generate char integers
            a[i++] = (n % 10) + '0';
            n = n / 10;
        }
        out_string(a);
    }
}
```

Figure 11: Program Code, Page 5

Part Number	Part Name	Cost per Part	Volume Discount (Price per unit for 100 units)	Source
1	ATMega324P	$4.87	$4.04	Mouser Electronics
2	Atmel Ice with Connector	$93.61	N/A	Digi-Key Electronics
3	4.7K Resistor	$0.10	$0.04	Jameco Electronics
4	Breadboard with Wires	$3.59	N/A	Amazon
5	1000 uF Capacitor	$0.22	$0.14	Mouser Electronics
6	10K Potentiometer (x2)	$0.57	$0.45	Mouser Electronics
7	10K Resistor	$0.25	$0.22	Jameco Electronics
8	22 Ohm Resistor	$0.12	N/A	Galco Industrial Electronics
9	LCD Screen	$2.18	$1.45	Newegg
Total		$106.08		

Table 1: Bill of Materials