Scott Leonard
Mattics Phi
CPE 329-03
Spring 2012
4/30/12
John Oliver

Project 4: SD Card Multi-Meter Data Logger

AKA: The 35v Max AC/DC Hybrid Synergy Power Logger 5000:
Secure Digital Black-Ops Edition

Brief Statement of the Project

The purpose of this project is to construct a multi-meter data logging module to measure voltage and current readings from external circuits, and then store this information on a SD card for convenience and practically unlimited storage space.

System Requirements

The central requirement of our system is to combine an SD card storage system with a multimeter. Multimeters are very useful in electrical design and testing because they combine a number of measurement tools together in a cheap, portable, and reasonably accurate package. The ability to combine this portable, easy to use instrument with a data logging system is the central goal of our system.

System Specification

In order for this system to be useful, the multi-meter component must be reasonably precise. The system doesn't need the precision of a bench top tool, but it should be consistent in measurements. In addition, the data logging feature will need to interface with an SD card in such a way that collected data can easily viewed by the user through a card reader without special software.

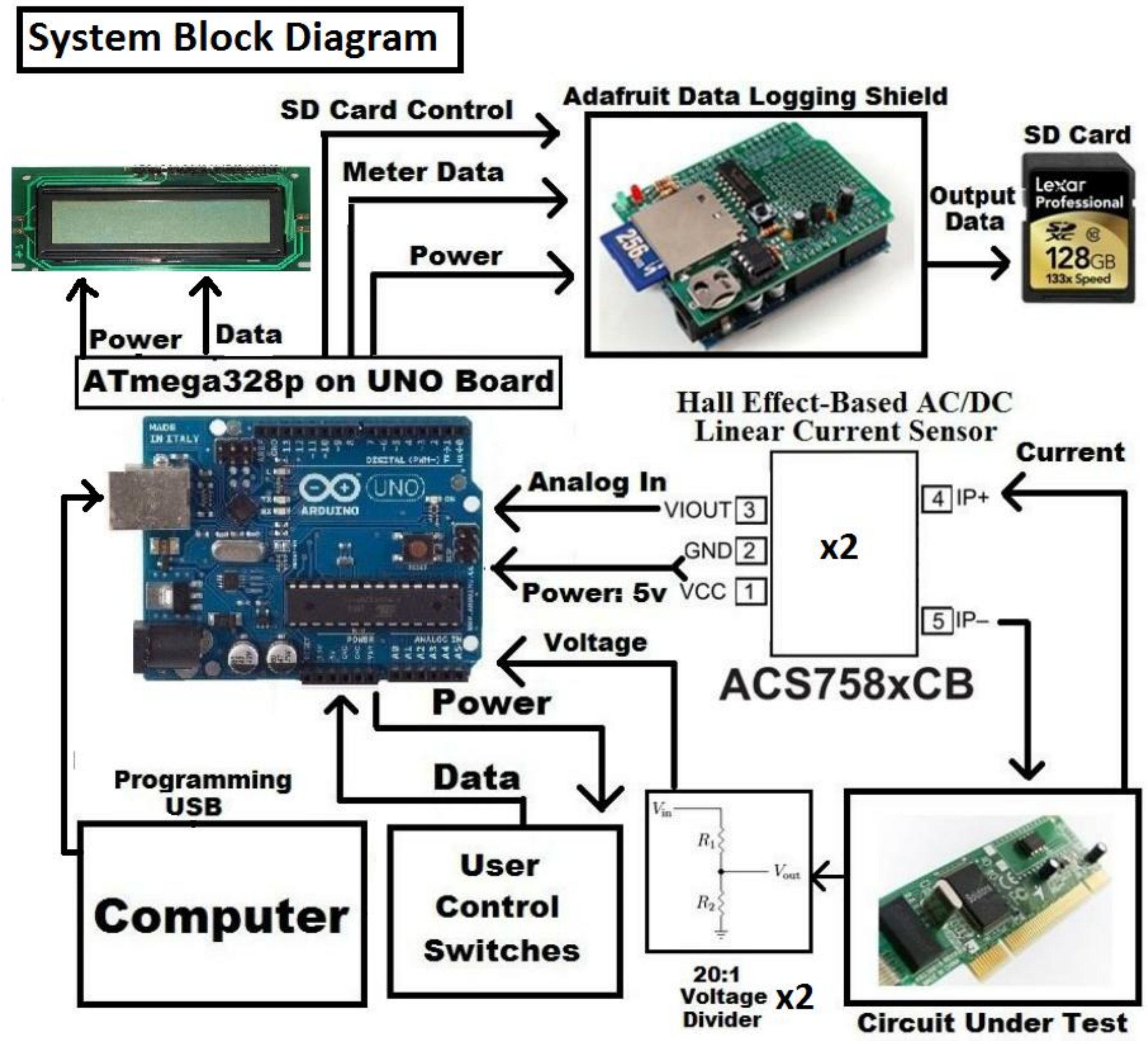
Specific Parameters

The system must have an accuracy of +/- 0.3 volts and +/- 0.3 amps, which will be verified by instantaneous readings on a bench top supply. Measurements must be taken at least every 3 seconds. The maximum voltage for measurements must be at least 20 volts (maximum output of the Agilent bench top supply). The maximum current measurement must be 50 amps for power and energy projects. The SD needs to use a FAT32 file structure for data logging for convenient access from a personal computer so collected data can easily be imported into a program like Excel from a text file.

System Architecture

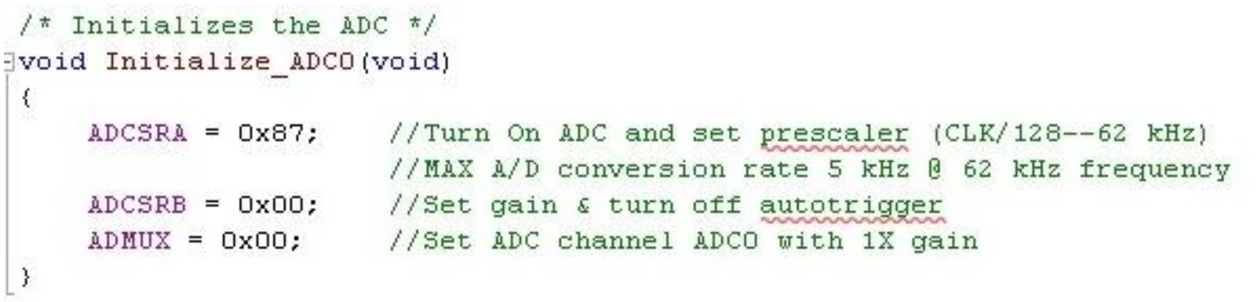
The project work was partitioned between both partners. Scott did the hardware aspect as well as the data logger, adding functionality to read voltage readings as well as present time stamps in the logger files. He also bought and implemented the various peripherals that went into this project. Mattics did the software for the voltage readings and the current readings. He also calibrated the readings to an accuracy of 3 significant figures, as seen on the Liquid Crystal Display. The hardware block diagram can be seen below in **Figure 1**, which shows how the design was implemented and carried out.

Figure 1: Hardware Block Diagram



Component Design

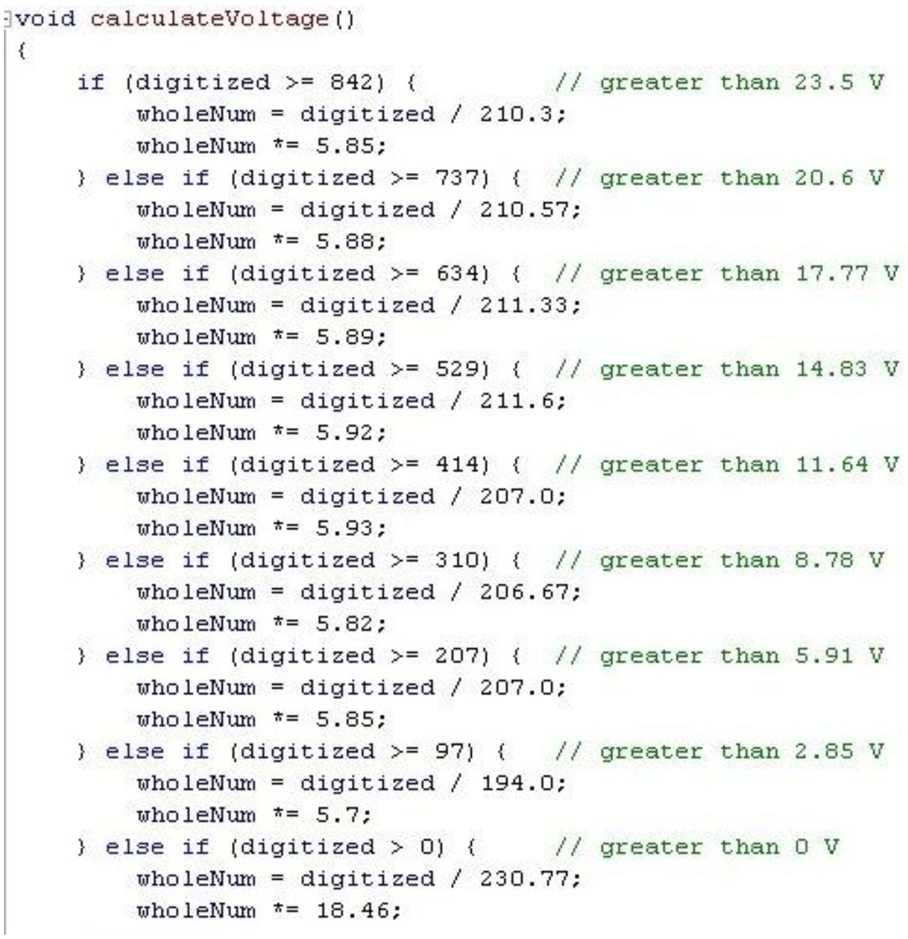
The coding portion of this project was very straightforward. The first things that had to be done was to be able to take input from the ADC. In order to do this, implementation of the ADC lab was in order, and in order to do this, being able to see what digital input was being seen had to be taken care of. By adding the LCD and outputting the digital input as seen from the ADC ports, implementing and error checking the voltage reader would be tolerable. The code to read the inputs can be seen below in **Figure 2**.



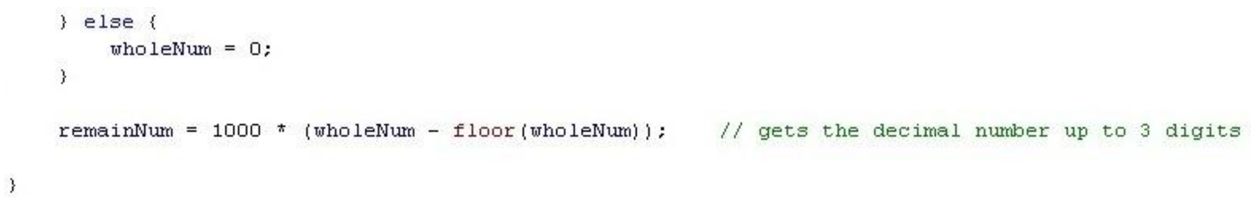
```
/* Initializes the ADC */
void Initialize_ADC0(void)
{
    ADCSRA = 0x87;    //Turn On ADC and set prescaler (CLK/128--62 kHz)
                      //MAX A/D conversion rate 5 kHz @ 62 kHz frequency
    ADCSRB = 0x00;    //Set gain & turn off autotrigger
    ADMUX = 0x00;     //Set ADC channel ADC0 with 1X gain
}
```

Figure 2: Code to Read Input From ADC

When testing, it was possible to see that the input from the ADC was not consistent, rather, it jumped around. By taking the average of the digital input and then doing some basic algebra to output the correct voltage. From testing and data recording, it's measured that some voltages require certain tweaking more than others. This can be seen in the code below in **Figure 3**.



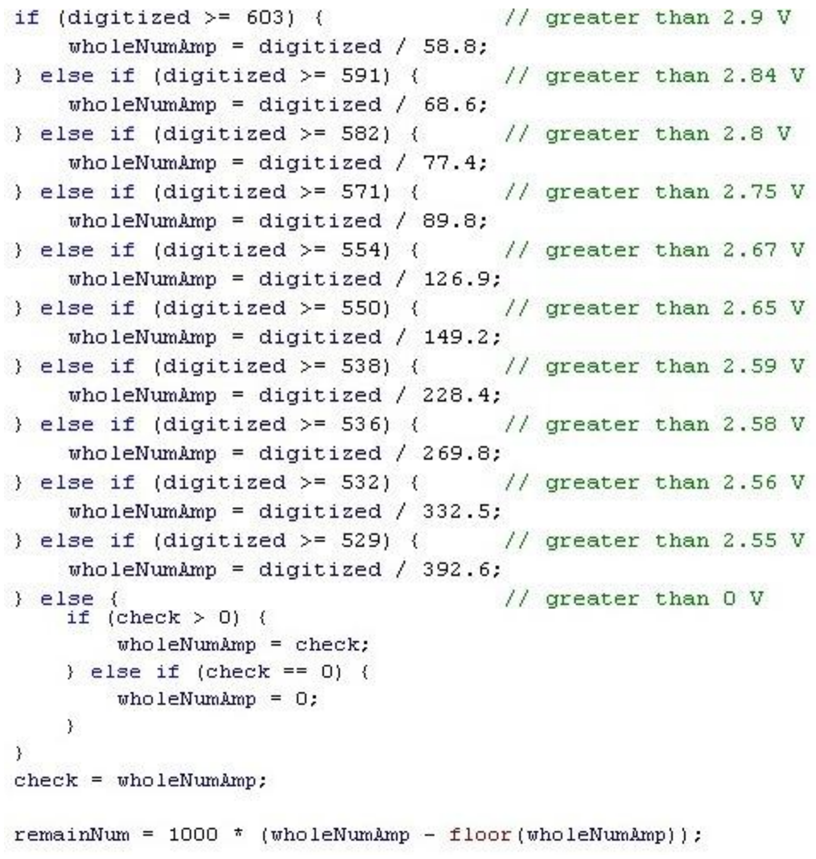
```
void calculateVoltage()
{
    if (digitized >= 842) {          // greater than 23.5 V
        wholeNum = digitized / 210.3;
        wholeNum *= 5.85;
    } else if (digitized >= 737) {   // greater than 20.6 V
        wholeNum = digitized / 210.57;
        wholeNum *= 5.88;
    } else if (digitized >= 634) {   // greater than 17.77 V
        wholeNum = digitized / 211.33;
        wholeNum *= 5.89;
    } else if (digitized >= 529) {   // greater than 14.83 V
        wholeNum = digitized / 211.6;
        wholeNum *= 5.92;
    } else if (digitized >= 414) {   // greater than 11.64 V
        wholeNum = digitized / 207.0;
        wholeNum *= 5.93;
    } else if (digitized >= 310) {   // greater than 8.78 V
        wholeNum = digitized / 206.67;
        wholeNum *= 5.82;
    } else if (digitized >= 207) {   // greater than 5.91 V
        wholeNum = digitized / 207.0;
        wholeNum *= 5.85;
    } else if (digitized >= 97) {    // greater than 2.85 V
        wholeNum = digitized / 194.0;
        wholeNum *= 5.7;
    } else if (digitized > 0) {      // greater than 0 V
        wholeNum = digitized / 230.77;
        wholeNum *= 18.46;
```



```
    } else {
        wholeNum = 0;
    }

    remainNum = 1000 * (wholeNum - floor(wholeNum));    // gets the decimal number up to 3 digits

}
```

Figure 3: Code Handling to Output Correct Voltage

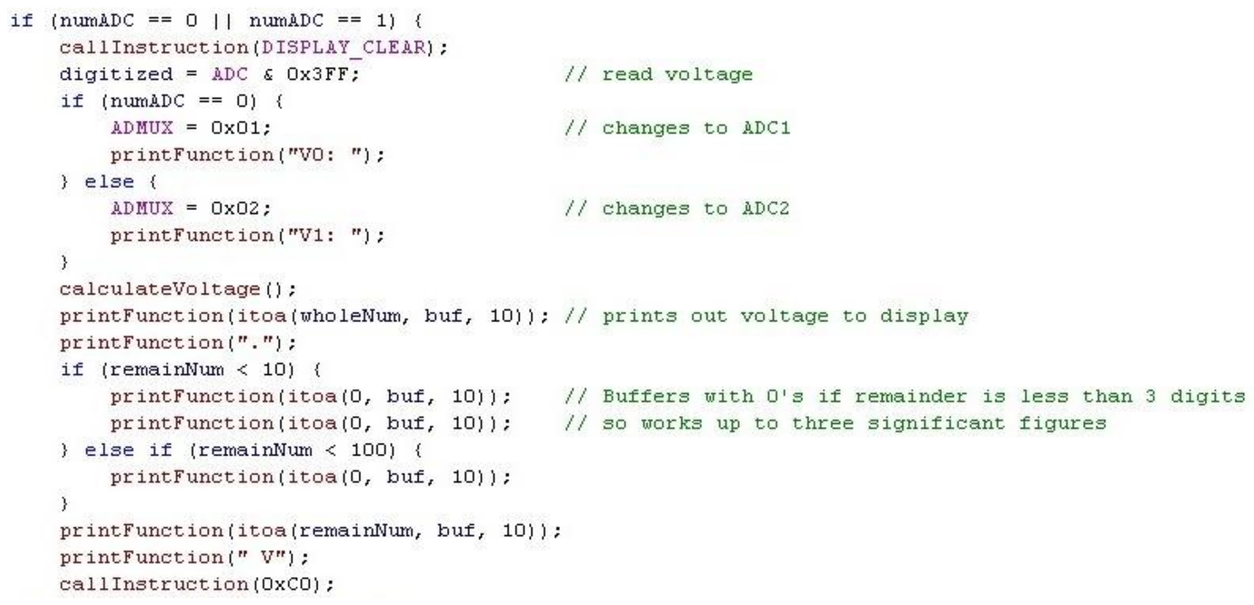
As always, with voltage readings also comes with current readings. The hall effect sensors used in this project output a voltage value which was then read through the ADC. Using a similar code design as the voltage readings, the current readings had to be averaged to a certain value when reading in certain ranges. The code can be seen below in **Figure 4**.



```
if (digitized >= 603) {              // greater than 2.9 V
    wholeNumAmp = digitized / 58.8;
} else if (digitized >= 591) {       // greater than 2.84 V
    wholeNumAmp = digitized / 68.6;
} else if (digitized >= 582) {       // greater than 2.8 V
    wholeNumAmp = digitized / 77.4;
} else if (digitized >= 571) {       // greater than 2.75 V
    wholeNumAmp = digitized / 89.8;
} else if (digitized >= 554) {       // greater than 2.67 V
    wholeNumAmp = digitized / 126.9;
} else if (digitized >= 550) {       // greater than 2.65 V
    wholeNumAmp = digitized / 149.2;
} else if (digitized >= 538) {       // greater than 2.59 V
    wholeNumAmp = digitized / 228.4;
} else if (digitized >= 536) {       // greater than 2.58 V
    wholeNumAmp = digitized / 269.8;
} else if (digitized >= 532) {       // greater than 2.56 V
    wholeNumAmp = digitized / 332.5;
} else if (digitized >= 529) {       // greater than 2.55 V
    wholeNumAmp = digitized / 392.6;
} else {                             // greater than 0 V
    if (check > 0) {
        wholeNumAmp = check;
    } else if (check == 0) {
        wholeNumAmp = 0;
    }
}
check = wholeNumAmp;

remainNum = 1000 * (wholeNumAmp - floor(wholeNumAmp));
```

Figure 4: Code Handling to Output Correct Current

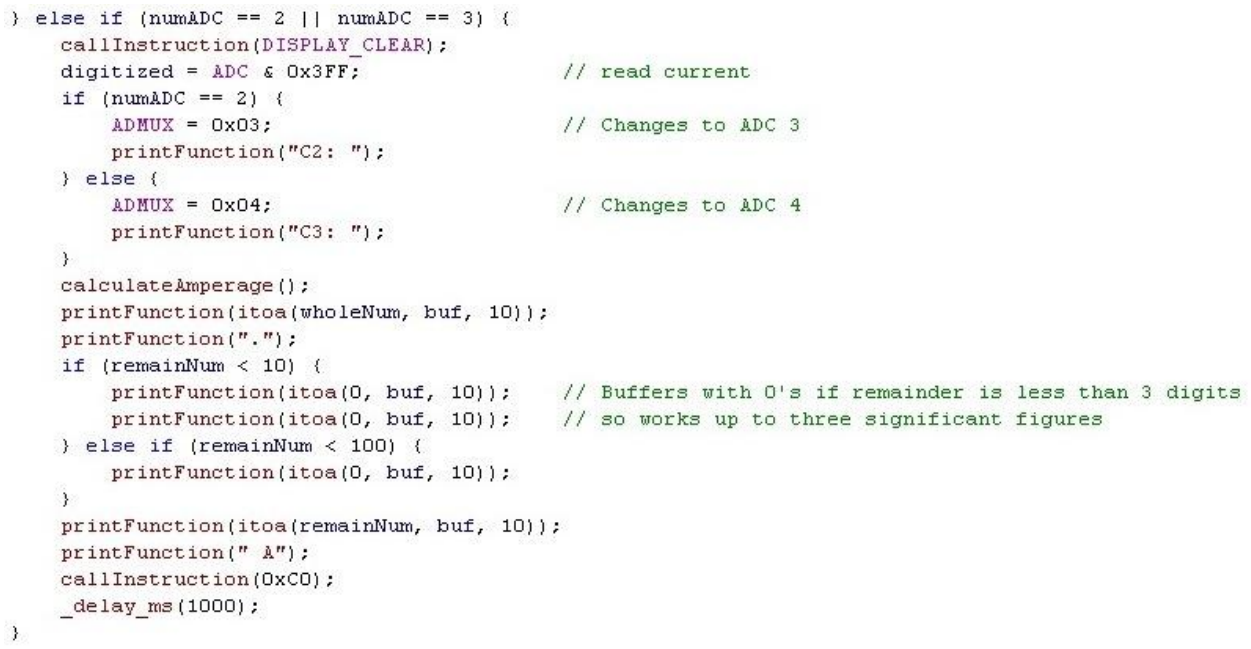
The reason that some voltage ranges differ from others is unknown, but by doing this, it is possible to get a more accurate voltage reading from multiple power supplies. The multiplication by ~5x is to output up to 35V. After that, it's just setting up the correct format on the LCD and then outputting the correct voltage. The code for that can be seen below in **Figure 5.**



```
if (numADC == 0 || numADC == 1) {
    callInstruction(DISPLAY_CLEAR);
    digitized = ADC & 0x3FF;                // read voltage
    if (numADC == 0) {
        ADMUX = 0x01;                       // changes to ADC1
        printFunction("V0: ");
    } else {
        ADMUX = 0x02;                       // changes to ADC2
        printFunction("V1: ");
    }
    calculateVoltage();
    printFunction(itoa(wholeNum, buf, 10)); // prints out voltage to display
    printFunction(".");
    if (remainNum < 10) {
        printFunction(itoa(0, buf, 10));    // Buffers with 0's if remainder is less than 3 digits
        printFunction(itoa(0, buf, 10));    // so works up to three significant figures
    } else if (remainNum < 100) {
        printFunction(itoa(0, buf, 10));
    }
    printFunction(itoa(remainNum, buf, 10));
    printFunction(" V");
    callInstruction(0xC0);
```

Figure 5: Code Outputting the Voltage to the LCD

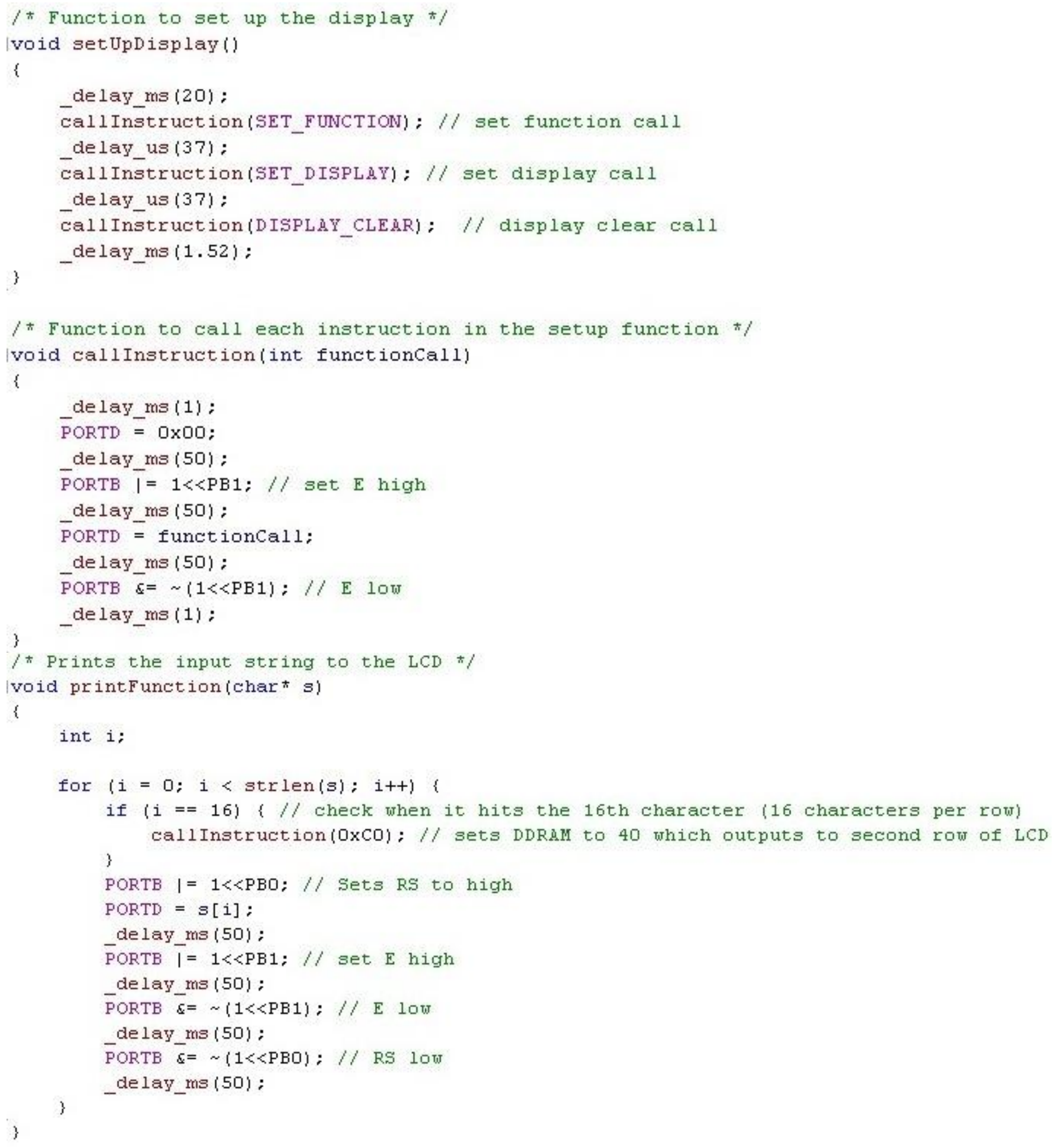
Setting up the current to print to the LCD was the same as the voltage. The current was to be presented at three significant figures, same as the voltage. The code can be seen below in **Figure 6**.



```
} else if (numADC == 2 || numADC == 3) {
    callInstruction(DISPLAY_CLEAR);
    digitized = ADC & 0x3FF;                // read current
    if (numADC == 2) {
        ADMUX = 0x03;                       // Changes to ADC 3
        printFunction("C2: ");
    } else {
        ADMUX = 0x04;                       // Changes to ADC 4
        printFunction("C3: ");
    }
    calculateAmperage();
    printFunction(itoa(wholeNum, buf, 10));
    printFunction(".");
    if (remainNum < 10) {
        printFunction(itoa(0, buf, 10));    // Buffers with 0's if remainder is less than 3 digits
        printFunction(itoa(0, buf, 10));    // so works up to three significant figures
    } else if (remainNum < 100) {
        printFunction(itoa(0, buf, 10));
    }
    printFunction(itoa(remainNum, buf, 10));
    printFunction(" A");
    callInstruction(0xC0);
    _delay_ms(1000);
}
```

Figure 6: Code Outputting the Current to the LCD

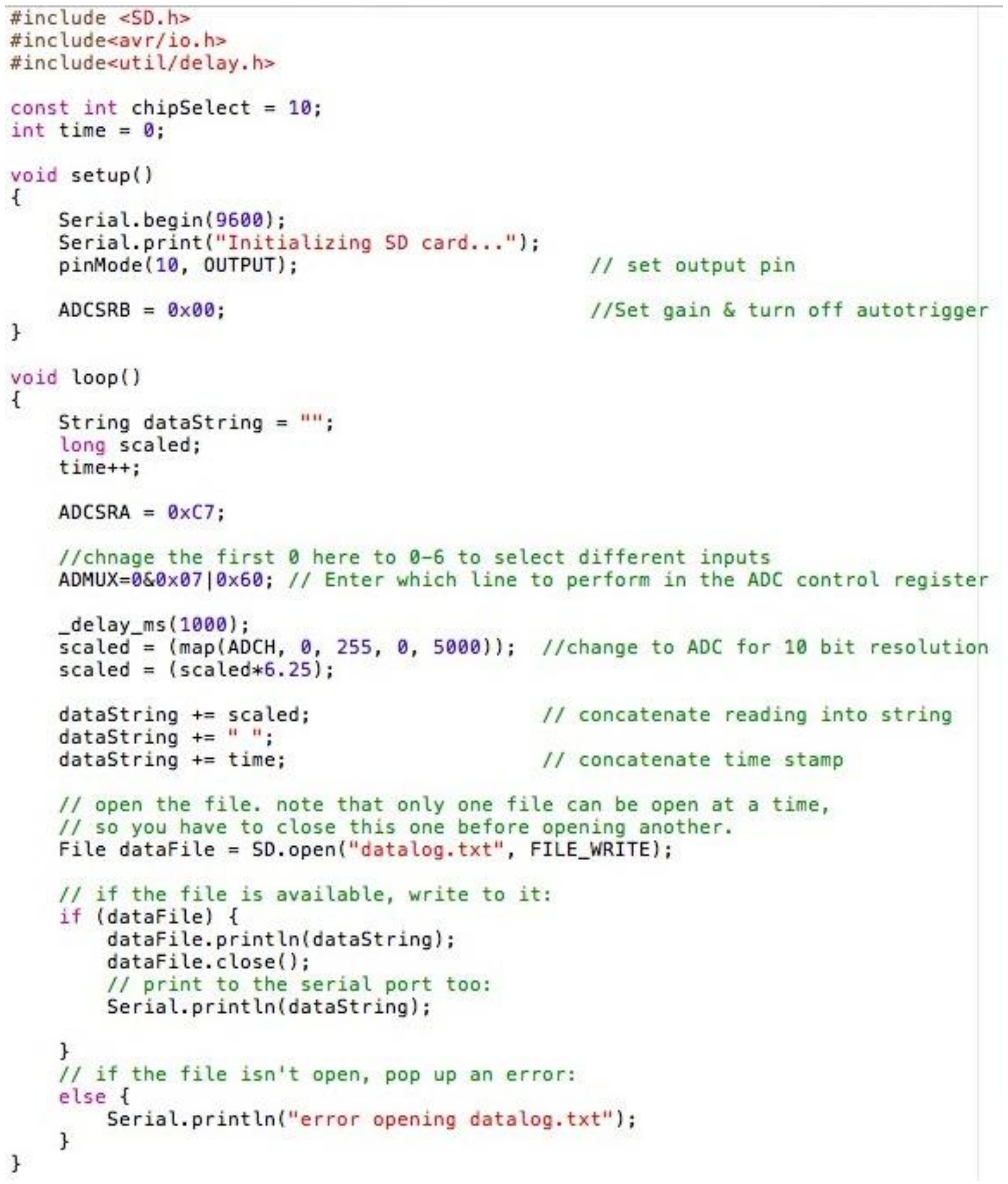
Printing to the LCD was just copied from previous programs. The code for this can be seen below in **Figure 7**.



```
/* Function to set up the display */
void setUpDisplay()
{
    _delay_ms(20);
    callInstruction(SET_FUNCTION); // set function call
    _delay_us(37);
    callInstruction(SET_DISPLAY); // set display call
    _delay_us(37);
    callInstruction(DISPLAY_CLEAR);  // display clear call
    _delay_ms(1.52);
}

/* Function to call each instruction in the setup function */
void callInstruction(int functionCall)
{
    _delay_ms(1);
    PORTD = 0x00;
    _delay_ms(50);
    PORTB |= 1<<PB1; // set E high
    _delay_ms(50);
    PORTD = functionCall;
    _delay_ms(50);
    PORTB &= ~(1<<PB1); // E low
    _delay_ms(1);
}
/* Prints the input string to the LCD */
void printFunction(char* s)
{
    int i;

    for (i = 0; i < strlen(s); i++) {
        if (i == 16) { // check when it hits the 16th character (16 characters per row)
            callInstruction(0xC0); // sets DDRAM to 40 which outputs to second row of LCD
        }
        PORTB |= 1<<PB0; // Sets RS to high
        PORTD = s[i];
        _delay_ms(50);
        PORTB |= 1<<PB1; // set E high
        _delay_ms(50);
        PORTB &= ~(1<<PB1); // E low
        _delay_ms(50);
        PORTB &= ~(1<<PB0); // RS low
        _delay_ms(50);
    }
}
```

Figure 7: Code to Print to the LCD

Aside from writing to the LCD and taking in ADC data, writing to an SD card data logger was also part of the project. The code was taken from an online source and implemented into the design. With the SD card, values can be written to it and then transferred and read from a text file on any computer. The simple code to do this can be seen below in **Figure 8.**



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

const int chipSelect = 10;
int time = 0;

void setup()
{
    Serial.begin(9600);
    Serial.print("Initializing SD card...");
    pinMode(10, OUTPUT);                        // set output pin

    ADCSRB = 0x00;                              //Set gain & turn off autotrigger
}

void loop()
{
    String dataString = "";
    long scaled;
    time++;

    ADCSRA = 0xC7;

    //chnage the first 0 here to 0-6 to select different inputs
    ADMUX=0&0x07|0x60; // Enter which line to perform in the ADC control register

    _delay_ms(1000);
    scaled = (map(ADCH, 0, 255, 0, 5000));  //change to ADC for 10 bit resolution
    scaled = (scaled*6.25);

    dataString += scaled;                   // concatenate reading into string
    dataString += " ";
    dataString += time;                     // concatenate time stamp

    // open the file. note that only one file can be open at a time,
    // so you have to close this one before opening another.
    File dataFile = SD.open("datalog.txt", FILE_WRITE);

    // if the file is available, write to it:
    if (dataFile) {
        dataFile.println(dataString);
        dataFile.close();
        // print to the serial port too:
        Serial.println(dataString);

    }
    // if the file isn't open, pop up an error:
    else {
        Serial.println("error opening datalog.txt");
    }
}
```

Figure 8: Code to Write to the SD Card

<u>Hardware and Schematics</u>

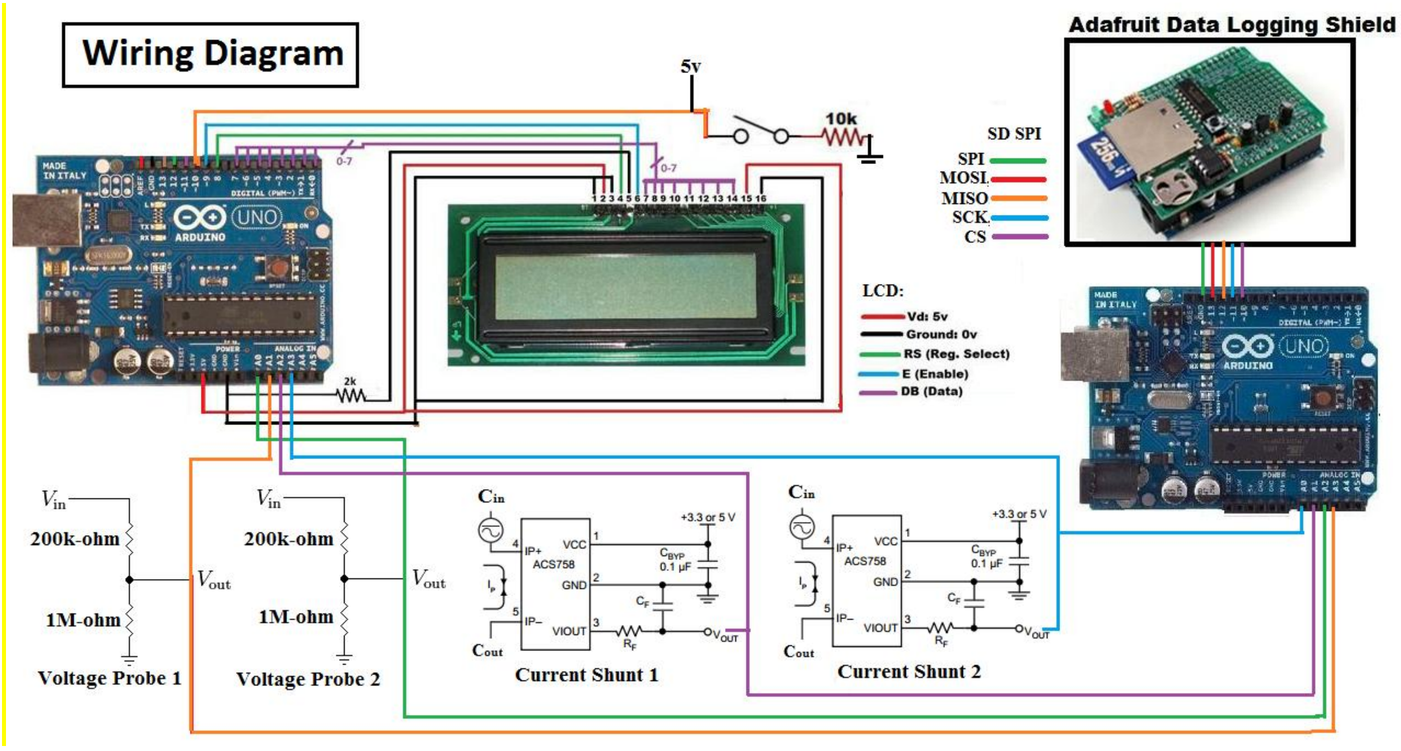
Three important external circuits were required to meet the requirements of our project. First, the current measurements needed to be taken through a Hall Effect sensor to measure both AC and DC inputs at high current. The pin out for these sensors is included below in **Figure 9** in our circuit diagram. The particular current sensor used in this project functioned by "outputting" half of its input voltage between two terminals. As current was shunted through the input terminals, the output voltage increased by 40mV for each 1 amp increase in current. Using a similar circuit as the voltage meter, this changing output voltage was received by the ADC and translated to the actual current rating in the circuit under test.

The second external component used in our circuit was a voltage divider, to extend the maximum input value of ADC from 5 volts to about 35 volts. This simple voltage divider was optimized to be as accurate as possible while drawing the smallest amount of current from the circuit under test. This was accomplished by choosing resistors values with a large enough value to avoid significant loading on the circuit, but a small enough value to avoid the effects of tolerance. For example, a 1 kilo-ohm resistor with a 5% tolerance will deviate from its rated value far less than a Mega-ohm resistor with the same tolerance. In the end, we chose a 1 Meg-ohm resistor and a 200k ohm resistor for the divider. Careful calibration of both dividers was required to account for the 5% tolerance of each resistor. Note that very low tolerance resistors are available with 1% and even 0.005% tolerances, but they come at a high price.

Finally, our last external circuit was the Adafruit Data Logger Shield. This shield was designed to fit directly over an Arduino Uno and featured an integrated SD card hold and real-time clock, making it ideal for this project. The full schematic for the data logger is included in the appendix. Communication between the SD and card Arduino was made through an array of buffers which linked the SPI, MOSI, MISO, SCK, and CS lines of the SD card to port B of the Arduino. A 3.3v power connection and ground line were also required. The maximum allowable SD card size for the logger is 2Gb. Considering a single voltage value is 4 bytes (4 text characters), a 2Gb SD card could store around 500 million data points, which was deemed acceptable for our purposes.

<u>Circuit Designed:</u>

Figure 9: Wiring Diagram



System Integration

Our multi-meter system will be calibrated and tested using bench top measurements tools, which will provide ample accuracy over a range of voltages and currents. Testing the system with a bench top instrument will be a straightforward procedure. A simple resistive circuit will be used as the circuit under test. This circuit will be powered by our Agilent Digital Power Supply through a range of voltages and a lead acid battery charger for a range of high currents.

The data logging system will then be configured to measure voltage and current while instantaneous measurements on the same component are recorded from a multimeter. These instantaneous measurements will be recorded by hand along with a time value for comparison with the resulting logged data on the SD card.

The timestamps produced by the data logging module will be checked against a digital clock to ensure accurate time keeping during testing. According to the datasheet for the module, incoming data is logged into the SD card in real-time, so calibration will be easy if access to an external clock is possible.

Verify Requirements

Our project met and exceeded all the requirements specified in our original proposal. The final product demonstrated a consistent resolution of 100 millivolts when measuring voltage over a range of 0 to 27 volts. Because of unexpected noise between the current sensors and our ADC, the current meter was less consistent with its readings, but a resolution of 0.3 amps was demonstrated as required. The SD card storage functionality was implemented without issue and performed as expected, allowing the logger to capture data at variable rates in a file format directly accessible by a PC card reader.

In order to meet our requirement for an “A” grade project, it was decided to implement a “high resolution” data logging mode. As described in our proposal, this mode of operation required that data be captured every 30 milliseconds instead of every 3 seconds. The final product exceeded this requirement by an order of magnitude by capturing data every 3 milliseconds. This high resolution mode is very useful for measuring high frequency signals without aliasing. Furthermore, this mode could be used for common AC waveform measurements with minimal software additions, like RMS and peak-to-peak measurements.

Problems Encountered

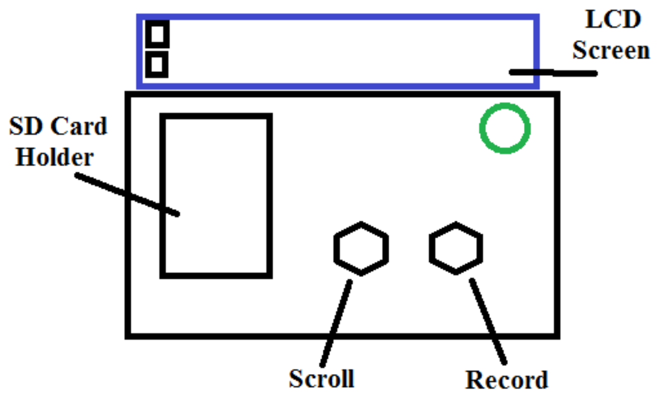
Many problems were encountered during this experiment. One important thing that was noticed was that when reading the voltages readings in digital form the ADC, the values jumped from number to number, and was never consistent. In order to resolve this, taking in a range of digital numbers at a time solved this problem. Through lots of data logging and calculations, the refined code for converting from digital back to analog was found and used. This can be seen in the code examples above. The same goes for the current values, except that the range of the digital input was far greater than the range of the voltage, between 80-90. This allowed the amperage reading to oscillate from normal values to 0 and back to normal values. In order to resolve this, implementation of a flag checker was put to use. When the previous reading was 0, and the current reading is 0, then it must not be plugged in so that the current amperage value must be 0. If the current amperage value is 0 and the previous amperage value was greater than 0, then it was a fluke so just print out what the previous current value is. This small bit of code fixed this issue.

Conclusion

When tested, our system displayed a reading from one of its four possible input probes on a LCD. Two input probes were used for current and two for voltage. These probes could be cycled through the display LCD through the press of a button. The data logging system could be configured to read any number of these inputs at time intervals down to 3 milliseconds, saving data to any number of text files on an SD card. Each reading could be coupled with a real-time stap for added utility. Critical shortcomings in the system were only created from noise on the current sensors that occasionally produced inaccurate readings. This issue could be solved by simply trying new sensors, perhaps with lower current ratings for higher accuracy. Decoupling capacitors may also have the ability to resolve this problem.

This system will be useful for future projects because one could log data over an extended amount of time without having to be present in a lab. This system could easily be extended by adding other sensors for different types of data collection. This platform would be very useful for future projects where accurate high speed data collection is required.

User Manual:



Thank you for purchasing the 35v Max AC/DC Hybrid Synergy Power Logger 5000: Secure Digital Black-Ops Edition. This brief manual will guide you through the use and functions of this device.

Specs:
Voltage Measurement: 0-35 volts DC, +/-0.3v
Current Measurement: 0-50 Amps AC or DC, +/-0.3 amps
Sensing Time Resolution: 3 milliseconds to 3 seconds, with real time stamps.

Set-Up:
Before turning your logger on, make sure the SD card is inserted inside the appropriate slot. Flip the power switch to turn on your device and watch the LCD screen for any errors. If the SD card has a problem, it will be displayed on the screen. The most common problem is “card not recognized”, which can be quickly rectified by inserting the card into any standard PC card reader and selecting the format option for the card, using the FAT32 file system. If no problems are detected, the screen will prompt you for a time measurement resolution. This value determines how often the card will save data from its probes. Scroll through the different time values by pressing the scroll button on the card. Once a desired value is selected, hit the record button to use that value. The logger is now ready for measurements.

Using the logger:
The voltage and current probes on the logger function like a multimeter. Place the voltage probes in parallel with the circuit under test and current probes in series. Confusing these connections on high power circuits is not recommended. To begin saving data, simply push the record button on the logger. A green LED will light up on the board while data is being saved to the SD card. Push the record button again to stop saving data. Before removing the SD card, turn the device off.

Appendices

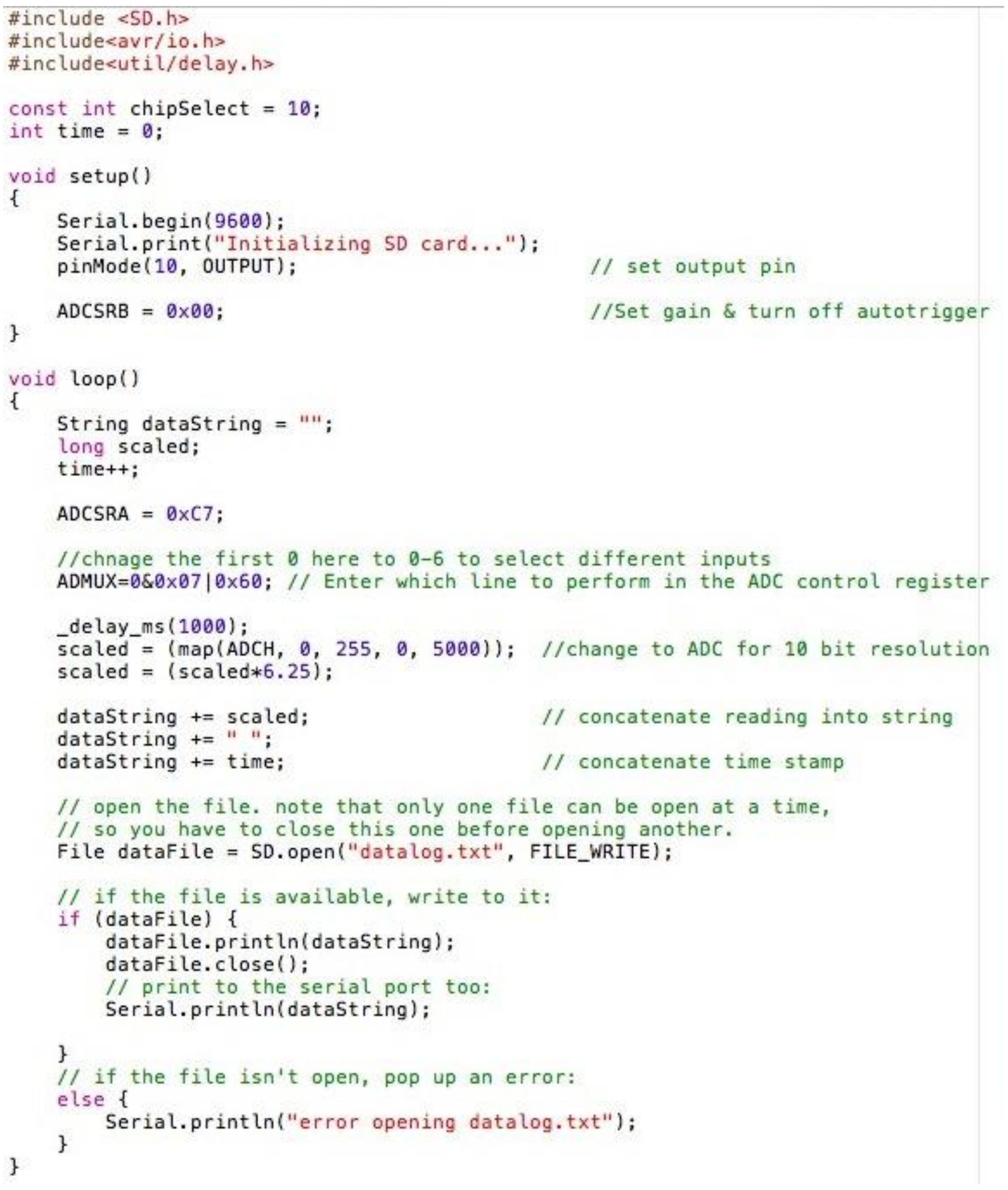
```
/*	Scott Leonard
	Mattics Phi
	Project 4 - Sensor Project
*/

#include <stdlib.h>
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <math.h>

#define F_CPU 16000000ULs
#define SET_FUNCTION 0x38
#define SET_DISPLAY 0x0F
#define SET_CGRAM 0x40
#define DISPLAY_CLEAR 0x01

/* Function Calls */
void printFunction(char*);
void callInstruction(int);
void setUpDisplay();
void Initialize_ADC0();
void printOutReading();
void calculateVoltage();
void calculateAmperage();

/* Global Variables */
char buf[10];
int numADC = 0;
int digitized = 0;
double wholeNum = 0;
double wholeNumAmp = 0;
double remainNum = 0;
double check = 0;

int main(void)
{
	Initialize_ADC0();                                          // Initializes the
ADC
	UCSR0B = 0;                                                        // TX/RX to
write I/O
	DDRB = 0x03;                                                // enables portB
output (PB1 PB0)
	DDRD = 0xFF;                                                // enables portD
output (All of PD 7:0)
	setUpDisplay();                                                    // function
to set up display calls
	_delay_ms(10);

	while (1) {
		if (PINB & 0x04) {                                  // button
interface
			numADC++;
			if (numADC == 4) {                          // Wraps around to
the first button
				numADC = 0;
			}
```

```
            }
            printOutReading();                                  // Prints out to
LCD
        }
    return 0;
}

void printOutReading()
{
        _delay_ms(1000);
        ADCSRA = 0xC7;                                                      // start
conversion
        _delay_us(260);                                                     // ensure
max sampling rate not exceeded

        if (numADC == 0 || numADC == 1) {
                callInstruction(DISPLAY_CLEAR);
                digitized = ADC & 0x3FF;                    // read voltage
                if (numADC == 0) {
                        ADMUX = 0x01;                                       // changes
to ADC1
                        printFunction("V0: ");
                } else {
                        ADMUX = 0x02;                                       // changes
to ADC2
                        printFunction("V1: ");
                }
                calculateVoltage();
                printFunction(itoa(wholeNum, buf, 10));     // prints out voltage to
display
                printFunction(".");
                if (remainNum < 10) {
                        printFunction(itoa(0, buf, 10));    // Buffers with 0's if
remainder is less than 3 digits
                        printFunction(itoa(0, buf, 10));    // so works up to three
significant figures
                } else if (remainNum < 100) {
                        printFunction(itoa(0, buf, 10));
                }
                printFunction(itoa(remainNum, buf, 10));
                printFunction(" V");
                callInstruction(0xC0);
        } else if (numADC == 2 || numADC == 3) {
                callInstruction(DISPLAY_CLEAR);
                digitized = ADC & 0x3FF;                    // read current
                if (numADC == 2) {
                        ADMUX = 0x03;                                       // Changes
to ADC 3
                        printFunction("C2: ");
                } else {
                        ADMUX = 0x04;                                       // Changes
to ADC 4
                        printFunction("C3: ");
                }
                calculateAmperage();
                printFunction(itoa(wholeNum, buf, 10));
                printFunction(".");
                if (remainNum < 10) {
                        printFunction(itoa(0, buf, 10));    // Buffers with 0's if
remainder is less than 3 digits
```

```
                printFunction(itoa(0, buf, 10));      // so works up to three
significant figures
            } else if (remainNum < 100) {
                printFunction(itoa(0, buf, 10));
            }
            printFunction(itoa(remainNum, buf, 10));
            printFunction(" A");
            callInstruction(0xC0);
            _delay_ms(1000);
        }
}

void calculateAmperage()
{
        if (digitized >= 603) {                         // greater than 2.9 V
            wholeNumAmp = digitized / 58.8;
        } else if (digitized >= 591) {          // greater than 2.84 V
            wholeNumAmp = digitized / 68.6;
        } else if (digitized >= 582) {          // greater than 2.8 V
            wholeNumAmp = digitized / 77.4;
        } else if (digitized >= 571) {          // greater than 2.75 V
            wholeNumAmp = digitized / 89.8;
        } else if (digitized >= 554) {          // greater than 2.67 V
            wholeNumAmp = digitized / 126.9;
        } else if (digitized >= 550) {          // greater than 2.65 V
            wholeNumAmp = digitized / 149.2;
        } else if (digitized >= 538) {          // greater than 2.59 V
            wholeNumAmp = digitized / 228.4;
        } else if (digitized >= 536) {          // greater than 2.58 V
            wholeNumAmp = digitized / 269.8;
        } else if (digitized >= 532) {          // greater than 2.56 V
            wholeNumAmp = digitized / 332.5;
        } else if (digitized >= 529) {          // greater than 2.55 V
            wholeNumAmp = digitized / 392.6;
        } else {                                              // greater than 0 V
            if (check > 0) {
                wholeNumAmp = check;
            } else if (check == 0) {
                wholeNumAmp = 0;
            }
        }
        check = wholeNumAmp;

        remainNum = 1000 * (wholeNumAmp - floor(wholeNumAmp));
}

void calculateVoltage()
{
        if (digitized >= 842) {                 // greater than 23.5 V
            wholeNum = digitized / 210.3;
            wholeNum *= 5.85;
        } else if (digitized >= 737) {  // greater than 20.6 V
            wholeNum = digitized / 210.57;
            wholeNum *= 5.88;
        } else if (digitized >= 634) {  // greater than 17.77 V
            wholeNum = digitized / 211.33;
            wholeNum *= 5.89;
        } else if (digitized >= 529) {  // greater than 14.83 V
            wholeNum = digitized / 211.6;
            wholeNum *= 5.92;
```

```
	} else if (digitized >= 414) {  // greater than 11.64 V
		wholeNum = digitized / 207.0;
		wholeNum *= 5.93;
	} else if (digitized >= 310) {  // greater than 8.78 V
		wholeNum = digitized / 206.67;
		wholeNum *= 5.82;
	} else if (digitized >= 207) {  // greater than 5.91 V
		wholeNum = digitized / 207.0;
		wholeNum *= 5.85;
	} else if (digitized >= 97) {   // greater than 2.85 V
		wholeNum = digitized / 194.0;
		wholeNum *= 5.7;
	} else if (digitized > 0) {           // greater than 0 V
		wholeNum = digitized / 230.77;
		wholeNum *= 18.46;
	} else {
		wholeNum = 0;
	}

	remainNum = 1000 * (wholeNum - floor(wholeNum));   // gets the decimal
number up to 3 digits

}

/* Function to set up the display */
void setUpDisplay()
{
	_delay_ms(20);
	callInstruction(SET_FUNCTION); // set function call
	_delay_us(37);
	callInstruction(SET_DISPLAY); // set display call
	_delay_us(37);
	callInstruction(DISPLAY_CLEAR);  // display clear call
	_delay_ms(1.52);
}

/* Function to call each instruction in the setup function */
void callInstruction(int functionCall)
{
	_delay_ms(1);
	PORTD = 0x00;
	_delay_ms(50);
	PORTB |= 1<<PB1; // set E high
	_delay_ms(50);
	PORTD = functionCall;
	_delay_ms(50);
	PORTB &= ~(1<<PB1); // E low
	_delay_ms(1);
}

/* Prints the input string to the LCD */
void printFunction(char* s)
{
	int i;

	for (i = 0; i < strlen(s); i++) {
		if (i == 16) { // check when it hits the 16th character (16
characters per row)
			callInstruction(0xC0); // sets DDRAM to 40 which outputs to
second row of LCD
```

```
            }
            PORTB |= 1<<PB0; // Sets RS to high
            PORTD = s[i];
            _delay_ms(50);
            PORTB |= 1<<PB1; // set E high
            _delay_ms(50);
            PORTB &= ~(1<<PB1); // E low
            _delay_ms(50);
            PORTB &= ~(1<<PB0); // RS low
            _delay_ms(50);
      }
}

/* Initializes the ADC */
void Initialize_ADC0(void)
{
      ADCSRA = 0x87;    //Turn On ADC and set prescaler (CLK/128--62 kHz)
                      //MAX A/D conversion rate 5 kHz @ 62 kHz frequency
    ADCSRB = 0x00;    //Set gain & turn off autotrigger
    ADMUX = 0x00;     //Set ADC channel ADC0 with 1X gain
}
```



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

const int chipSelect = 10;
int time = 0;

void setup()
{
    Serial.begin(9600);
    Serial.print("Initializing SD card...");
    pinMode(10, OUTPUT);                              // set output pin

    ADCSRB = 0x00;                                    //Set gain & turn off autotrigger
}

void loop()
{
    String dataString = "";
    long scaled;
    time++;

    ADCSRA = 0xC7;

    //chnage the first 0 here to 0-6 to select different inputs
    ADMUX=0&0x07|0x60; // Enter which line to perform in the ADC control register

    _delay_ms(1000);
    scaled = (map(ADCH, 0, 255, 0, 5000));  //change to ADC for 10 bit resolution
    scaled = (scaled*6.25);

    dataString += scaled;                         // concatenate reading into string
    dataString += " ";
    dataString += time;                           // concatenate time stamp

    // open the file. note that only one file can be open at a time,
    // so you have to close this one before opening another.
    File dataFile = SD.open("datalog.txt", FILE_WRITE);

    // if the file is available, write to it:
    if (dataFile) {
        dataFile.println(dataString);
        dataFile.close();
        // print to the serial port too:
        Serial.println(dataString);

    }
    // if the file isn't open, pop up an error:
    else {
        Serial.println("error opening datalog.txt");
    }
}
```

References

LCD: http://www.sparkfun.com/datasheets/LCD/st7066.pdf
Data Logger: http://www.ladyada.net/make/logshield/design.html
Current Sensor: http://www.alldatasheet.com/datasheet-pdf/pdf/299822/ALLEGRO/ACS758XCB.html
SD Card and FAT32 Tutorial for ARV: http://www.frank-zhao.com/cache/fat_sd.php
Program 1 for LCD code
ADC Lab