ECE-412
Spring 2022
Lab #3

Drew Branum, David Antosh, Calvin Lam, Ankit Kanotra

UART, ADC, EEPROM, GPIO, LCD Module, Mixing C and Assembly

ECE 412
Abstract

Headers are soldered onto the Xplained Mini evaluations board. The MCU is then connected to an external circuit constructed on a breadboard. Assembly language and C are pasted and built within Atmel Studio 7 to investigate features of the ATMega328P(B) MCU. A serial communication terminal, TeraTerm, is downloaded to interact with the MCU. The serial port settings are then set to the required settings and the ATmega328P(B) data sheet's information covering the USART peripheral is examined and analyzed. Atmel documentation pertaining to mixing C and Assembly language in a project is also examined and analyzed. Atmel Studio is then used to load the program onto the ATmega328P(B) and TeraTerm is used to examine the LCD, ADC, and EEPROM commands. Analysis of the commands takes place and is recorded. The voltage of the circuit is altered with a potentiometer to examine the ADC command and deeper examination of voltage conversion, ADC registers, circuitry, and code takes place using the ATmega328P(B) datasheet. The EEPROM command is then examined and tested with and without Atmel Studio. Using the ATmega328P(B) datasheet, the data, address, control registers, and code of the EEPROM is analyzed and recorded. The LCD command is examined and tested with alphanumeric graphics and text. The functionality of the LCD light working with the USART on the same port is examined and recorded. The wiring, backlight control, GPIO ports and contrast control of the LCD are analyzed using code and section 18 of the ATmega328P(B) data sheet. Conclusions are made about the C and Assembly language, the circuitry, the LCD, and the functionality of the given commands with respect to their hardware.

Body

The assembly instructions that initialize the USART are located in the global "Mega328P_Init:" function. These are the pertinent instructions:

```
out U2X0, r16
sts UBRR0H, r17
sts UBRR0L, r16
sts UCSR0B, r16
sts UCSR0C, r16
```

The UBBR0L and UBBR0H registers constitute the low and high bytes of the UBRR0 Baud Rate Register and the values stored in these registers are used by other components of the USART that need the baud rate, such as the down-counter. UCSRnA, UCSRnB, and USCR0C are control and status registers. Registers of this type allow you to choose between synchronous or asynchronous communication, and both possess unique bits that can be changed to alter the behavior of the USART. Additionally, the USCRnC register dictates the frame format as it is used to set the character size, parity mode, and stop bit select bits. Double speed transmission can be enabled when U2X0 is set to 1 when using asynchronous communication Otherwise, the U2X0 bit must be set to 0. The final USART register is the UDRn. Data that is written to this register is stored in the Transmit Data Buffer Register(TXB), and the data stored in the Receive Data Buffer Register(RXB) is returned upon reading this register. If the transmitter is enabled and data is written to the UDRn register, the data will eventually be transmitted to the TxDn pin.

The command line functions properly due to the "Mega328P_Init()" assembly function which initializes the necessary USART registers that are used in the "Command" C function such as UART_Puts and UART_Get(). In order to implement a different baud rate, the UBBRnL and UBBRnH registers would have to be changed in order to alter the UBBRn register. The appropriate bits of the UCSRnC register would have to be modified in order to change the number of data bits, parity, and number of stop bits. Bit UCZn determines the number of data bits, bit UPMn determines the type of parity bit, and bit USBSn determines the number of stop bits. Communication begins with the generation of the baud rate by the baud rate generator. The baud rate and other pertinent data then moves to the data bus. Due to the interrupt driven capability of the TXC flag, it is useful in half-duplex communication interfaces The RXC flag ensures that data received from the UDR is read in its entirety.

Before the ADC can be used, the power reduction bit in the power reduction register (PRR0) must be set to 0. Additionally, the ADEN (ADC enable) bit of the ADCSRA (ADC control and status register A) register must be set to 1 in order for the ADC to use voltage references and input channel selections. Any of the ADC input pins or analog input channel can be selected as single ended inputs. The ADC utilizes comparators to compare the unknown input voltage and the reference voltage(s) repeatedly until the unknown input voltage is determined.

When dealing with a resolution of 10 bits, in order for the successive approximation circuitry to function properly, an input clock frequency between 50 and 200 kHz must be used in order to ensure the highest resolution. The clock prescaler helps ensure that the proper input clock frequency is used since it produces a specified clock frequency as long as the frequency of the CPU is above 100kHz. The prescaling is configured via the ADC Prescaler Selecter bits (ADPS) in the ADCSRA register. A 10 bit resolution has 2^10 (1024) discrete levels.

The ADC is in single conversion mode when the Power Reduction ADC bit (PRADC) in the Power Reduction Register (PRR) is set to 0 and the ADC Start Conversion bit (ADSC) in the ADC Control and Status Register A (ADCSRA) is set to 1. This bit remains the same for the duration of the conversion but changes to 0 after its completion. Thus, conversions can only be done one at a time in single conversion mode. Contrarily, the ADC is in free running mode when the ADC Interrupt flag is used as a trigger source because the ADC will start a new conversion as soon as the previous conversion has finished regardless of the status of the ADC Interrupt Flag (ADIF). Upon a successful conversion, a 10-bit digital result is produced and stored in the ADC Data Registers (ADCH and ADCL).

In TinyOS, the assembly "Mega328P_Init" initializes a few ADC registers such as ADCSRA, ADMUX, and ADCSRB. The only other assembly function that alters ADC registers is the ADC_Get function which is called by the ADC function in main.c. With differential input, quantization occurs whereas it doesn't with integral input. Additionally, deviation differs as well, since for differential input, deviation is present in the actual transition, whereas deviation is present in the actual code width for integral input.

The ATmega328P(B) contains 1 KB of EEPROM data memory. To access EEPROM from the CPU, the address, data, and control registers must be specified. The EEPROM address registers, EEARH and EEARL, must have an appropriate value written to them in order to be accessed since they are initially undefined. Within the given program, EEARH is offset to 0x22 and EEARL to 0x21. They are then used in EEPROM Read and Write to set up the address for which the input values can be read or written. The EEPROM data register, EEDR, contains the data to be written or read within the program at the address maintained by EEARH and EEARL. The EEDR is offset to 0x20 within the program to initialize its address. It is then used to read the written data or write the inputted data within the EEPROM in conjunction with register 16. The EEPROM control register, EECR, is offset 0x1F. Within the EECR, Bit 3, EEPROM Ready Interrupt Enable, Bit 2, EEPROM Master Write Enable, Bit 1 EEPROM Write Enable, and Bit 0 EEPROM Read Enable are utilized to implement read/write capabilities. When 1 is written to Bit 0, EEPROM read is triggered and the requested data is immediately available. When 0 is written to Bit 0, EEPROM read is turned off. When 1 is written to Bit 1 and Bit 2, EEPROM write is enabled. Bit 2 is the master write, so if Bit 1 is enabled and Bit 2 isn't, EEPROM write will still be disabled. Both must be enabled to write data. The following procedure is recommended by the ATmega328P(B) datasheet: "

1. Wait until EEPE becomes zero.
2. Wait until SPMEN in SPMCSR becomes zero.
3. Write new EEPROM address to EEAR (optional).
4. Write new EEPROM data to EEDR (optional).
5. Write a '1' to the EEMPE bit while writing a zero to EEPE in EECR.
6. Within four clock cycles after setting EEMPE, write a '1' to EEPE

".

Writing 1 to Bit 3 enables ready interrupt if the I bit is set in SREG. Ready Interrupt interrupts constantly when Bit 1 is cleared and is not generated during EEPROM write. Within the given software, Bits 0-3 are used to implement EEPROM read/write by equating them to their respective bits (0-3) at the beginning of the program. Throughout the program the bits are set using the SBI (Set Bit in I/O Register) in conjunction with the EECR. A general description of the given EEPROM_Read function: check to see if the EEPE is clear, if not loop infinitely, if so load r18 and r17 with 0x00 and 0x05 to set the address of EEARH (r18) and EEARL (r17). Load r16 with 0x00 and set the EERE bit

in the EECR so EEPROM read is enabled. Load the value in r16 to EEDR and store r16 in ASCII. EEPROM_Write is nearly the exact same, but different addresses are used for r17 and r18. Along with this r16 is set to 'F' in order to write it to EEPROM when Bits 1 and 2 are enabled.

The LCD possesses 11 commands. The first command, Clear Display, stores "20H" in DDRA and sets DDRAM address to "00H". The "Return Home" command also sets DDRAM address to "00H" but it does not alter its content and moves the cursor back to its initial position if changed. The "Entry Mode Set" command determines the direction in which the cursor will move and the way that the LCD will blink. The "Display ON/OFF control" command sets the following bits to 0 or 1: display bit, cursor bit, and blinking of cursor bit. The "Cursor or Display shift" command sets the cursor moving bit, display shift control bit, and the direction. This command can be used to fix issues with display data. The DDRAM remains unchanged after this command is executed. The "Function Set" command allows you to alter DL (Interface data length control bit), N and F (both are Display line number control bits). DL allows you to choose between 4-bit or 8-bit bus mode, N allows you to choose between 1 or 2 line display mode, and F allows you to choose the dots format display mode. The "Set CGRAM Address" and "Set DDRAM Address" commands allow you to set the CGRAM address and DDRAM addresses respectively. The "Read Busy Flag and Address" command allows you to determine if the LCD is in the middle of an internal operation by reading the busy flag. The address counter can also be read via this command. The "Write data to address" command stores data in internal RAM while the "Read data from RAM" command reads data from internal RAM. The LCD is capable of displaying numbers, symbols, letters, and Japanese characters.

11 pins (pin 4-14) of the LCD are connected to the Xplained Mini board. PBn and PDn pins are on the Xplained Mini Board. PB1 connects to RS, PB0 connects to RW, PB2 connects to E, PD0 connects to D0, PD1 connects to D1, PD2 connects to D2, PD3 connects to D3, PD4 connects to D4, PD5 connects to D5, PD6 connects to D6, and PD7 connects to D7. The backlight functions properly by connecting the cathode of the LCD to ground and the anode to a power source. The cathode and anode are the last two pins of the LCD. The brightness of the LCD screen is able to be changed by using the first three pins of the LCD and the use of a Single Turn 10K potentiometer.

All AVR ports possess read and write functionality as long as they are used as general digital I/O ports. Thus, the direction of a port pin can be changed as needed without being concerned about its effects on the direction of other port pins. A GPIO port pin is configured as input if its DDxn bit is set to 0. The port pin will be configured as output if the DDxn bit is set to 1. Some pins in Port C can be used as an external interrupt source. A port pin can be pulled high if its PORTxn bit is set to 1 and its DDxn bit is set to 0.

A TWI (Two-Wire Serial Interface) allows the interconnection of up to 128 different devices. A single pull-up resistor is necessary for each of the TWI bus lines in order to implement a TWI. The pull-up resistors in port pins can be activated by configuring the pin as an input pin and setting the PORTxn bit to 1. The SPI (Serial Peripheral Interface) allows synchronous data to be quickly transferred between a device and other units. The SPI must be enabled by setting the PRSPI bit in the Power Reduction Register to 0. The TC are timer counters that are enabled by setting the PRTIM0 bit in the Minimizing Power Consumption register to 0 and setting the PRTIM0 bit in the Power Reduction Register to 1. The AC (Analog Comparator) compares the input values on pins AIN0 and AIN1.

Source Code (Software)

Main.c:

```
// Lab3P1.c
//
//
// Author : Eugene Rockey
//

//no includes, no ASF, no libraries

const char MS1[] = "\r\nECE-412 ATMega328PB Tiny OS";
const char MS2[] = "\r\nby Eugene Rockey Copyright 2022, All Rights Reserved";
const char MS3[] = "\r\nMenu: (L)CD, (A)DC, (E)EPROM\r\n";
const char MS4[] = "\r\nReady: ";
const char MS5[] = "\r\nInvalid Command Try Again...";
const char MS6[] = "Volts\r";



void LCD_Init(void);                    //external Assembly functions
void UART_Init(void);
void UART_Clear(void);
void UART_Get(void);
void UART_Put(void);
void LCD_Write_Data(void);
void LCD_Write_Command(void);
void LCD_Read_Data(void);
void Mega328P_Init(void);
void ADC_Get(void);
void EEPROM_Read(void);
void EEPROM_Write(void);

unsigned char ASCII;                    //shared I/O variable with Assembly
unsigned char DATA;                     //shared internal variable with Assembly
char HADC;                              //shared ADC variable with Assembly
char LADC;                              //shared ADC variable with Assembly

char volts[5];                          //string buffer for ADC output
int Acc;                                //Accumulator for ADC use
double temp1;                           //Temperature conversion variables
double temp2;
double temp3;
double resist1 = 10000.0;               //The resistance of the resistor
double resist2;                         //Resistance of the thermistor
double B = 3950.0;                      //B value of the thermistor
char temp[3];
double v0 = 0.0;                        //Volt double placeholder
```

```
int a;
int b;
int c;                                          //end of temperature variables


void UART_Puts(const char *str)      //Display a string in the PC Terminal Program
{
        while (*str)
        {
                ASCII = *str++;
                UART_Put();
        }
}

void LCD_Puts(const char *str)       //Display a string on the LCD Module
{
        while (*str)
        {
                DATA = *str++;
                LCD_Write_Data();
        }
}


void Banner(void)                           //Display Tiny OS Banner on Terminal
{
        UART_Puts(MS1);
        UART_Puts(MS2);
        UART_Puts(MS4);
}

void HELP(void)                 //Display available Tiny OS Commands on Terminal
{
        UART_Puts(MS3);
}

void loop(void)                         //Loop Function For LCD Scrolling Display
{
        int c, d;
        for (c = 1; c <= 32; c++){
                for (d = 1; d<= 3276; d++)
                {}
        }
        return;
}


void LCD(void)                                          //Lite LCD demo
{
        DATA = 0x34;                                //Loads 0x00100010 into PortD
        LCD_Write_Command();
```

```
        DATA = 0x08;                                    //Loads 0x00001000 into PortD
        LCD_Write_Command();
        DATA = 0x02;                                    //Loads 0x00000010 into PortD
        LCD_Write_Command();
        DATA = 0x06;                                    //Loads 0x00000110 into PortD
        LCD_Write_Command();
        DATA = 0x0f;                                    //Loads 0x00001111 into PortD
        LCD_Write_Command();

        //While Loop for Infinite Scrolling Unless ASCII Changes
        ASCII = '\0';
        while(ASCII == '\0'){
                LCD_Puts(" Team 1");                    //Puts Team 1 onto LCD screen
                loop();                                 //Calls Loop function for Delay
        }
}

void ADC(void)                          //Lite Demo of the Analog to Digital Converter
{
        volts[0x1]='.';
        volts[0x3]=' ';
        volts[0x4]= 0;
        ADC_Get();
        Acc = (((int)HADC) * 0x100 + (int)(LADC))*0xA;
        volts[0x0] = 48 + (Acc / 0x7FE);
        Acc = Acc % 0x7FE;
        volts[0x2] = ((Acc *0xA) / 0x7FE) + 48;
        Acc = (Acc * 0xA) % 0x7FE;
        if (Acc >= 0x3FF) volts[0x2]++;
        if (volts[0x2] == 58)
        {
                volts[0x2] = 48;
                volts[0x0]++;
        }
        //UART_Puts(volts);
        //UART_Puts(MS6);
        v0 = volts[0x0] - 48 + (volts[0x2] - 48) / 10.0;        //converts volts from a string
                                                                //into a double

        resist2 = (v0 * resist1) / (5.0 - v0);                  //calculate the resistance of
                                                                //the thermistor

        //temperature calculation and conversion into Celsius
        temp1 = (B * 298.15) / (298.15 * log(resist2 / resist1) + B);
        temp2 = temp1 - 273.15;

        //temp3 = ((temp2 * 9.0) / 5.0) + 32.0;         //Celsius to Fahrenheit conversion

        //ASCII conversion of temperature(Celsius) double
        b = floor(fmod(temp2, 10.0));
        c = b + 48;
```

```
        a = floor(((temp2 - b) / 10.0)) + 48;

        temp[0x0] = a;
        temp[0x1] = c;
        temp[0x2] = 0;

        UART_Puts(temp);
        UART_Puts(" Celsius");

}

void Read()
{
        //Asks User for Input
        UART_Puts("\r\nEnter valid address or X to exit");
        //Initialize ASCII Variable
        ASCII = '\0';
        //Busy Loop Until User Input
        while (ASCII == '\0')
        {
                UART_Get();
        }
        //Set DATA Variable as ASCII User Input
        DATA = ASCII;
        //Call Read Assembly Function
        EEPROM_Read();
}

void Write()
{
        //Asks User for Input
        UART_Puts("\r\n Enter valid address or X to exit");
        //Initialize ASCII Variable
        ASCII = '\0';
        //Busy Loop Until User Input
        while (ASCII == '\0')
        {
                UART_Get();
        }
        //Set DATA Variable as ASCII User Input
        DATA = ASCII;
        //Call Write Assembly Function
        EEPROM_Write();
}


void EEPROM(void)
{
        //Asks User for Input
        UART_Puts("\r\nEnter W to write data or R to read data or X to exit");
        //Initialize ASCII Variable
```

```
	ASCII = '\0';
	//Busy Loop Until User Input
	while (ASCII == '\0')
	{
		UART_Get();
	}
	//Switch Case for possible Menu Options
	switch (ASCII)
	{
		case 'W' | 'w': Write();
		break;
		case 'R' | 'r': Read();
		break;
		case 'X' | 'x': Command();
		break;
	}

}


void Command(void)				//command interpreter
{
	UART_Puts(MS3);
	ASCII = '\0';
	while (ASCII == '\0')
	{
		UART_Get();
	}
	switch (ASCII)
	{
		case 'L' | 'l': LCD();
		break;
		case 'A' | 'a': ADC();
		break;
		case 'E' | 'e': EEPROM();
		break;
		default:
		UART_Puts(MS5);
		HELP();
		break;
	}
}

int main(void)
{
	Mega328P_Init();
	Banner();
	while (1)
	{
		Command();			//infinite command loop
	}
```

```
}
```

Assembler1.s

```
// Lab3P1.s
//
//
// Author : Eugene Rockey
// Copyright 2022, All Rights Reserved


.section ".data"                    //sets up constants in the memory
.equ    DDRB,0x04                   //Port B Data Direction Register as I/O, sets port to
                                    //0x04 location
.equ    DDRD,0x0A                   //DDRD = 10
.equ    PORTB,0x05                  //PORTB = 5
.equ    PORTD,0x0B                  //PORTD = 11
.equ    U2X0,1                      //U2X0 = 1
.equ    UBRR0L,0xC4                 //UBRR0L = 196
.equ    UBRR0H,0xC5                 //UBRR0H = 197
.equ    UCSR0A,0xC0                 //UCSR0A = 192
.equ    UCSR0B,0xC1                 //UCSR0B = 193
.equ    UCSR0C,0xC2                 //UCSR0C = 194
.equ    UDR0,0xC6                   //UDR0 = 198
.equ    RXC0,0x07                   //RXC0 = 7
.equ    UDRE0,0x05                  //UDRE0 = 5
.equ    ADCSRA,0x7A                 //ADCSRA = 122
.equ    ADMUX,0x7C                  //ADMUX = 124
.equ    ADCSRB,0x7B                 //ADCSRB = 123
.equ    DIDR0,0x7E                  //DIDR0 = 126
.equ    DIDR1,0x7F                  //DIDR1 = 127
.equ    ADSC,6                      //ADSC = 6
.equ    ADIF,4                      //ADIF =4
.equ    ADCL,0x78                   //ADCL = 120
.equ    ADCH,0x79                   //ADCH = 121
.equ    EECR,0x1F                   //EECR = 31
.equ    EEDR,0x20                   //EEDR = 32
.equ    EEARL,0x21                  //EEARL = 33
.equ    EEARH,0x22                  //EEARH = 34
.equ    EERE,0                      //EERE = 0
.equ    EEPE,1                      //EEPE = 1
.equ    EEMPE,2                     //EEMPE = 2
.equ    EERIE,3                     //EERIE = 3

//variables accessible by the C code
.global HADC                        //high bit of ADC for C code
.global LADC                        //low bit of ADC for C code
.global ASCII                       //numbers will be stored to match the ASCII table
.global DATA                        //Used for communication between C and
Assembly instructions
```

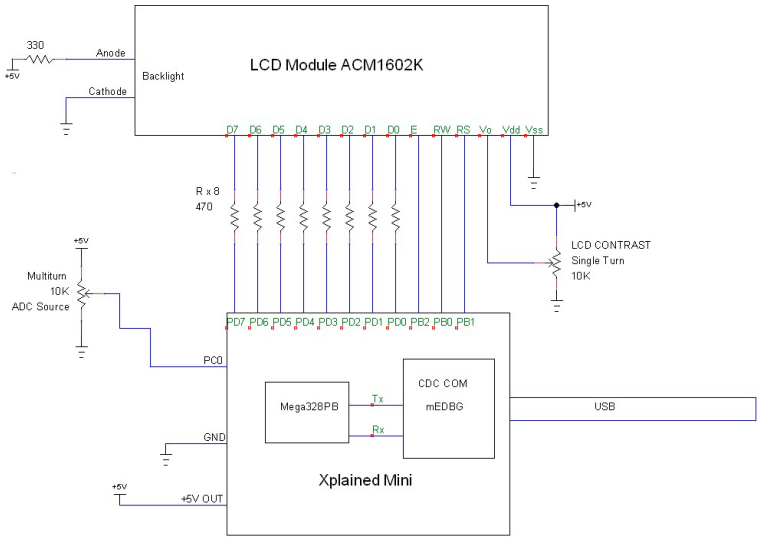
```
.set      temp,0                    //temp = 0

.section ".text"                    //Section used for program code and functions
.global Mega328P_Init
Mega328P_Init:
        ldi     r16,0x07            ;PB0(R*W),PB1(RS),PB2(E) as fixed outputs
        out     DDRB,r16            //writes value in r16 to DDRB port at 0x04
        ldi     r16,0               //loads r16 with 0
        out     PORTB,r16           //writes value (0) in r16 to PORTB
        out     U2X0,r16            ;initialize UART, 8bits, no parity, 1 stop, 9600
        ldi     r17,0x0             //loads r17 with 0x0
        ldi     r16,0x67            //loads r16 with 0x67
        sts     UBRR0H,r17          //stores value in r17 to data space at UBRR0H
                                    0xC5
        sts     UBRR0L,r16          //stores value in r16 to data space at UBRR0L 0xC4
        ldi     r16,24              //loads r16 with 24
        sts     UCSR0B,r16          //stores value in r16 to data space at UCSR0B 0xC1
        ldi     r16,6               //loads r16 with 6
        sts     UCSR0C,r16          //stores value in r16 to data space at UCSR0C 0xC2
        ldi     r16,0x87            //initialize ADC, loads r16 with 0x87
        sts     ADCSRA,r16          //stores value in r16 to data space ADCSRA 0x7A
        ldi     r16,0x40            //loads r16 with 0x40
        sts     ADMUX,r16           //stores value in r16 to data space ADMUX 0x7C
        ldi     r16,0               //loads r16 with 0
        sts     ADCSRB,r16          //stores value in r16 to data space ADCSRB 0x7B
        ldi     r16,0xFE            //loads r16 with 0xFE
        sts     DIDR0,r16           //stores value in r16 to data space DIDR0 0x7E
        ldi     r16,0xFF            //loads r16 with 0xFF
        sts     DIDR1,r16           //stores value in r16 to data space DIDR1 0x7F
        ret                         //returns to line after init was called

.global LCD_Write_Command
LCD_Write_Command:
        call    UART_Off            //calls UART_Off to turn off the UART
        ldi     r16,0xFF            ;PD0 - PD7 as outputs
        out     DDRD,r16            //Stores value of r16 into DDRD address in SRAM
        lds     r16,DATA            //loads value in data space DATA () into r16
        out     PORTD,r16           //Stores value of r16 into PORTD address in SRAM
        ldi     r16,4               //load indirect 4 into r16
        out     PORTB,r16           //Stores value of r16 into PORTB address in SRAM
        call    LCD_Delay           //calls LCD_Delay function
        ldi     r16,0               //load indirect 0 into r16
        out     PORTB,r16           //Stores value of r16 into PORTB address in SRAM
        call    LCD_Delay           //calls LCD_Delay function
        call    UART_On             //calls UART_On to turn the UART on
        ret                         //returns to main function

LCD_Delay:
        ldi     r16,0xFA            //loads r16 with 0xFA
D0:     ldi     r17,0xFF            //start of D0 subroutine, loads r17 with 0xFF
```

```
D1:     dec     r17             //start of D1 subroutine, decreases r17 by 1
        brne    D1              //Repeat until r17 is 0
        dec     r16             //decreases r16 by 1
        brne    D0              //Repeat until r16 is 0
        ret                     //returns to main function

.global LCD_Write_Data
LCD_Write_Data:
        call    UART_Off        //calls UART_Off subroutine to turn off UART
        ldi     r16,0xFF        //load indirect 0xFF into r16
        out     DDRD,r16        //writes value in r16 to DDRD (0x0A)
        lds     r16,DATA        //loads r16 with the value stored in DATA
        out     PORTD,r16       //writes value in r16 to PORTD (0x0B)
        ldi     r16,6           //load indirect 6 into r16
        out     PORTB,r16       //writes value in r16 to PORTB (0x05)
        call    LCD_Delay       //calls LCD_Delay subroutine
        ldi     r16,0           //load indirect 0 into r16
        out     PORTB,r16       //writes value in r16 to PORTB (0x05)
        call    LCD_Delay       //calls LCD_Delay subroutine
        call    UART_On         //calls UART_On subroutine
        ret                     //returns to main function

.global LCD_Read_Data
LCD_Read_Data:
        call    UART_Off        //calls UART_Off subroutine to turn off UART
        ldi     r16,0x00        //loads r16 with 0x00
        out     DDRD,r16        //writes value in r16 to DDRD port at 0x0A
        out     PORTB,4         //writes value 4 to PORTB port at 0x05
        in      r16,PORTD       //writes value at PORTD () to r16
        sts     DATA,r16        //stores the value in r16 to data space DATA ()
        out     PORTB,0         //writes value 0 to PORTB port at 0x05
        call    UART_On         //calls UART_On subroutine to turn the UART on
        ret                     //returns to main function

.global UART_On
UART_On:
        ldi     r16,2           //Load immediate r16 with 2
        out     DDRD,r16        //Copy bit 2 into port DDRD (port 10)
        ldi     r16,24          //Load immediate r24 with 24
        sts     UCSR0B,r16      //Copy r16(24) into UCSR0B SRAM Location
(193)
        ret                     //Return to Function this was called from

.global UART_Off
UART_Off:
        ldi     r16,0           //Load immediate r16 with 0
        sts     UCSR0B,r16      //Copy r16(0) into UCSR0B SRAM Location (193)
        ret                     //Return to Function this was called from

.global UART_Clear
UART_Clear:
```

```
        lds     r16,UCSR0A      //r16  is set with 192
        sbrs    r16,RXC0        //Skip next instruction if bit 7 of r16 is 1
        ret                     //return to main function
        lds     r16,UDR0        //loads r16 with the value stored in the data space at
                                UDR0 ()
        rjmp    UART_Clear      //relative jump to UART_Clear subroutine

.global UART_Get
UART_Get:
        lds     r16,UCSR0A      //r16 is set to 192
        sbrs    r16,RXC0        //Skip next instruction if bit 7 of r16 is 1
        rjmp    UART_Get        //jumps to UART_Get (unconditional jump)
        lds     r16,UDR0        //loads the value in data space at UDR0 to r16
        sts     ASCII,r16       //Store ASCII global with value of r16
        ret                     //return to main function

.global UART_Put
UART_Put:
        lds     r17,UCSR0A      //r17 is set to 192
        sbrs    r17,UDRE0       //checks if the bit register has been set
        rjmp    UART_Put        //restarts function
        lds     r16,ASCII       //grabs the stored value in ASCII and loads it into
                                //r16
        sts     UDR0,r16        //stores r16 into stack
        ret                     //return to main function

.global ADC_Get
ADC_Get:
        ldi     r16,0xC7        //Loads 199 to r16
        sts     ADCSRA,r16      //Stores r16 in stack
A2V1:
        lds     r16,ADCSRA      //Loads store value from stack into r16
        sbrc    r16,ADSC        //Skips next line if the Bit register was cleared
        rjmp    A2V1            //Loops back up to A2V1 it Bit register wasn't
                                //cleared
        lds     r16,ADCL        //loads 120 from .data section
        sts     LADC,r16        //loads 120 into LADC so C code can access
        lds     r16,ADCH        //loads 121 from .data section
        sts     HADC,r16        //loads 121 into HADC for C code
        ret                     //return to main function

.global EEPROM_Write
EEPROM_Write:
        sbic    EECR,EEPE
        rjmp    EEPROM_Write    ; Wait for completion of previous write
        ldi     r18,0x00        ; Set up address (r18:r17) in address register
        ldi     r17,0x05
        ldi     r16,DATA        ; Set up data in r16
        out     EEARH, r18
        out     EEARL, r17
        out     EEDR,r16        ; Write data (r16) to Data Register
```

```
        sbi     EECR,EEMPE      ; Write logical one to EEMPE
        sbi     EECR,EEPE       ; Start eeprom write by setting EEPE
        ret

.global EEPROM_Read
EEPROM_Read:
        sbic    EECR,EEPE
        rjmp    EEPROM_Read     ; Wait for completion of previous write
        ldi     r18,0x00        ; Set up address (r18:r17) in EEPROM address
                                ;register
        ldi     r17,0x05
        ldi     r16,0x00
        lds     r16,DATA
        out     EEARH, r18
        out     EEARL, r17
        sbi     EECR,EERE       ; Start eeprom read by writing EERE
        in      r16,EEDR        ; Read data from Data Register
        sts     ASCII,r16
        ret


        .end
```

Schematics (Hardware)



Title: ECE412 Lab 3 Circuit		
Author: Eugene Rockey		
File: C:\Users\Donkey\Desktop\test.dsn		Document
Revision: 1.0	Date	Sheets: 1 of 1

Analysis

Through the building of a circuit and re-engineering of code. Knowledge of the relationship between code and hardware was achieved. Additionally, the relationship of how C can be programmed to operate in conjunction with assembly routines. This understanding of how to set up components such as the LCD screen and how to read data from I/O can be applied to almost any problem as the process for doing so will be very similar. Additionally, through the use of TeraTerm, an understanding of the process for testing code from a PCB/Circuit was gained, as well as ways for debugging.

Conclusion

In conclusion, the wiring and coding of a circuit provided an understanding of the relationship between hardware and software. The skills to read data from I/O, store and read data from SRAM, and how to operate components such as an LCD screen were achieved. Additionally, the full process of using and working with higher level languages in conjunction with assembly to operate hardware, will be a beneficial tool in the understanding of more complex problems.

References

Atmel, AVR Assembler: User Guide.

AVR instruction set manual. (n.d.). Retrieved February 20, 2022, from http://atmel-studio-doc.s3-website-us-east-1.amazonaws.com/webhelp/GUID-0B644D8F-67E7-49E6-82C9-1B2B9ABE6A0D-en-US-1/index.html

Resistance - Adafruit Industries. (n.d.). Retrieved April 3, 2022, from https://cdn-shop.adafruit.com/datasheets/103_3950_lookuptable.pdf

Specifications of LCD Module. Retrieved March 2, 2022 from https://components101.com/sites/default/files/component_datasheet/16x2%20LCD%20Datasheet.pdf