Lab: 5
Nicholas Imamshah
Section: 6957

b) Prelab Questions

1. Assume you wanted a voltage reference range from -3V to 1V, with an unsigned 12-bit ADC. What are the voltages if the ADC output is 0xA92 and 0x976?
    a. 0xA92 => -0.3575V; 0x976 => -0.6348V
2. What is the difference in conversion ranges between 12-bit unsigned and signed conversion modes? List both ranges.
    a. 12-bit: signed = -2048 to 2047, unsigned = 0 to 4095
3. If you were working on another microcontroller and you wanted to add an 8-bit LCD to it, what is the minimum amount of signals required from the microcontroller to get it working?
    a. WE, RS (Assuming always in need of Command and Data differentiation), 8 Data lines, V_{DD}, V_O, V_{SS}.
4. In this lab our reference range is ideally from 0V to 5V. If the range was 0 to 2.0625V (a possible internal reference) and 12-bit signed mode was used, what is the resolution (volts/bit) and what is the digital value for a voltage of 0.42V.
    a. $2.0625V/2^{12} = 0.0005035$. DV = 834_{10}.

c) Problems Encountered

With the LCD: Since I figured we'd want to expand on Lab 3's EBI, I copied the Quartus project with the decoding logic; this resulted in the programmer trying to use the .pof from lab 3 and not the newly generated one. As a result, for a number of hours I was wondering why my control signals were all wrong, when they were never actually going into any kind of decoding logic in the first place.

With the ADC: Initially experienced issues with understanding how to use the ADC and configure it in C. After gaining some understanding I encountered trouble with reading the value; when using prescaler as DIV512, the values being recorded were wildly variable, but with DIV4 it became much more consistent.

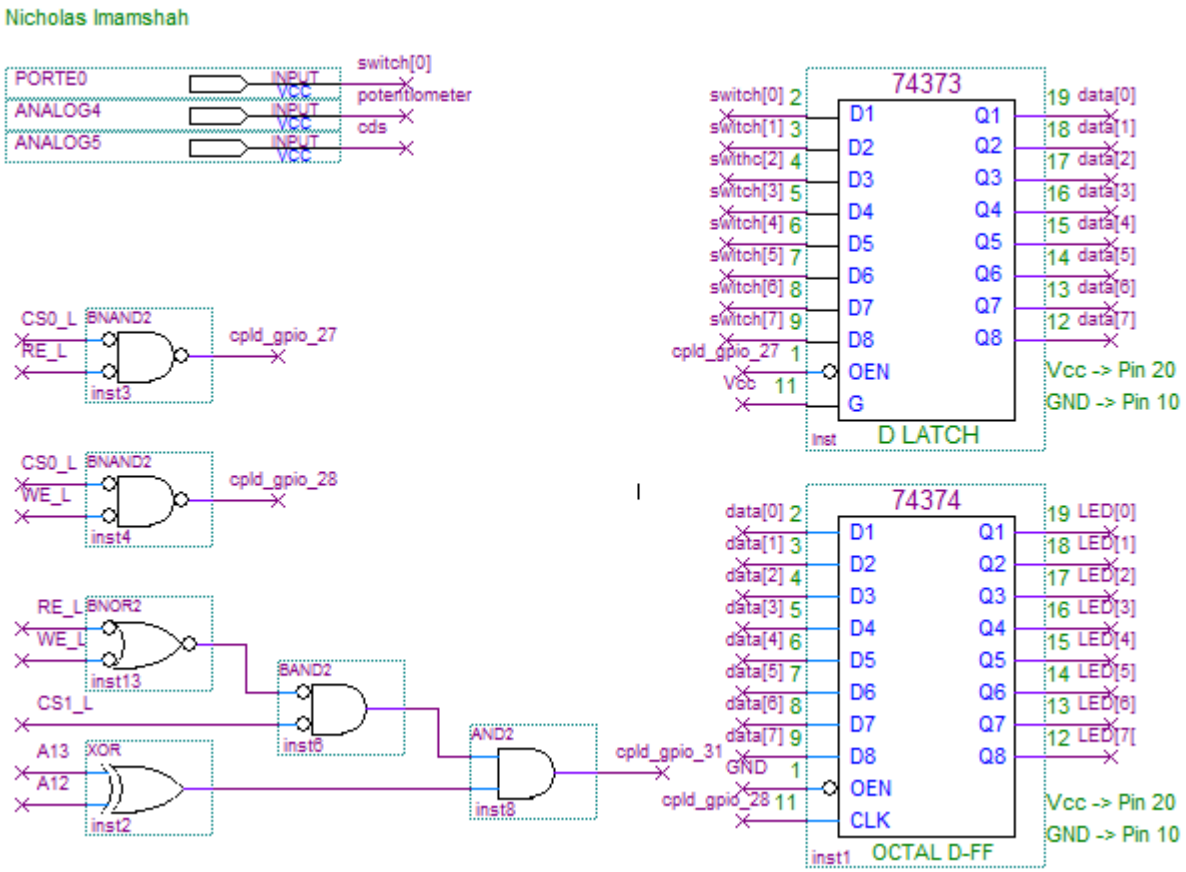
With the CdS: After understanding the potentiometer, I thought the process would be mostly identical, but missed the usage of the EVCTRL to sweep the now interesting channel 1.

d) Future Work/Applications

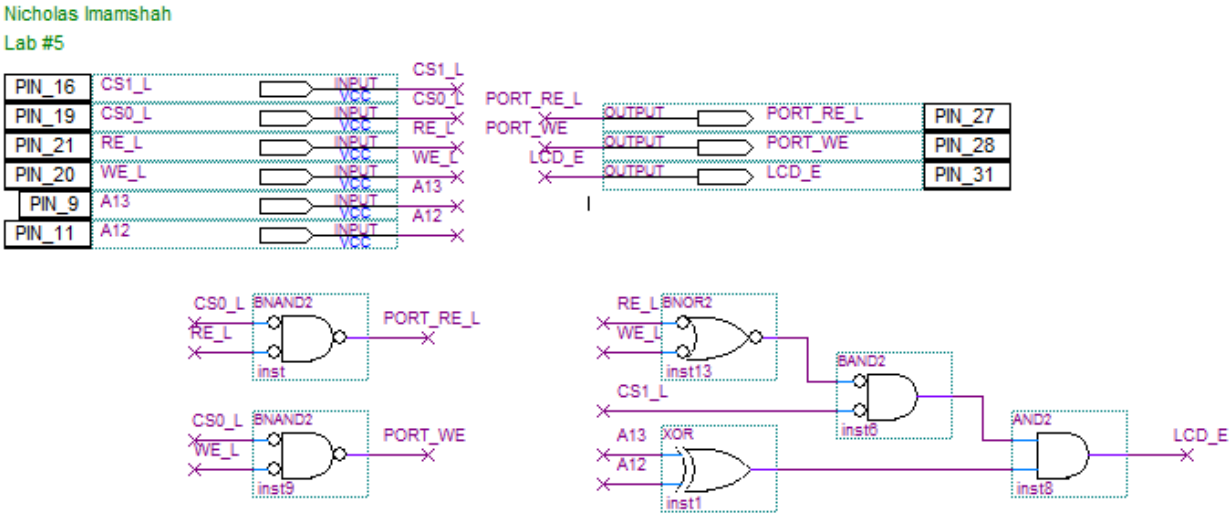
There are just so many things that can be done with the various parts of this lab. Recording the voltage of the potentiometer via ADC gives us dial form of control; for example, when a room has a light switch controlling a dimmable bulb, such a configuration could have been used to accomplish it (though I don't know how many rooms are controlled by a microprocessor). We also see LCDs in many applications, take for instance most vending machines now that present an LCD to provide information about the various items it offers. Finally, the CdS could be used

to sense the presence of a person; for instance, in a lighted setting, if a person were to enter and dim the light cast on the CdS, then that could trigger a function that responds to the person's presence.

e) Schematics

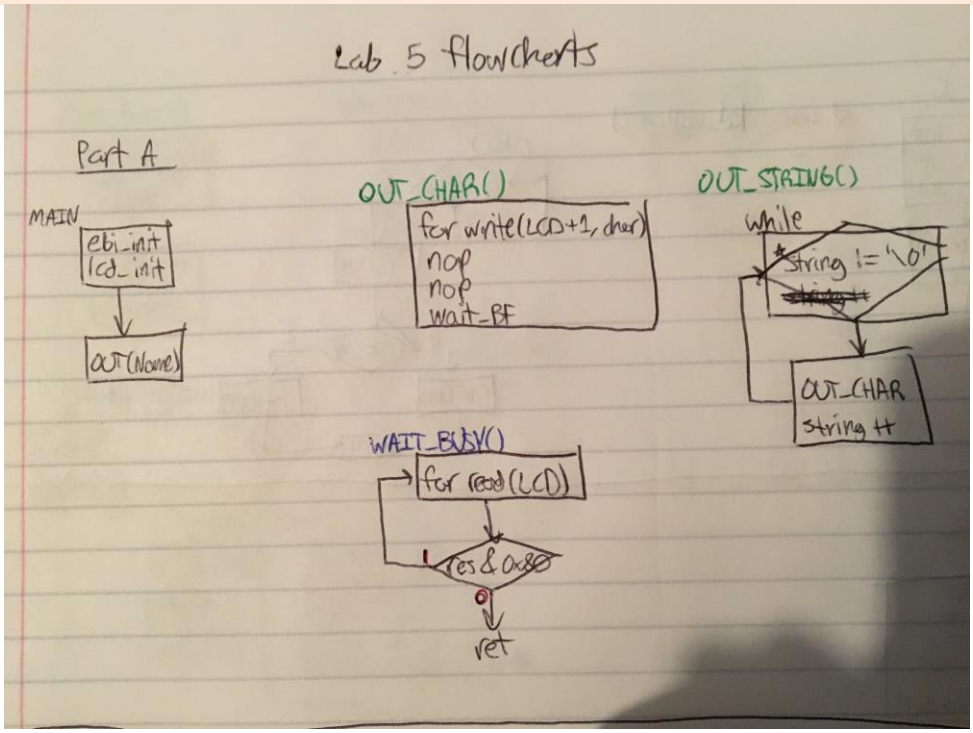


f) Decoding Logic

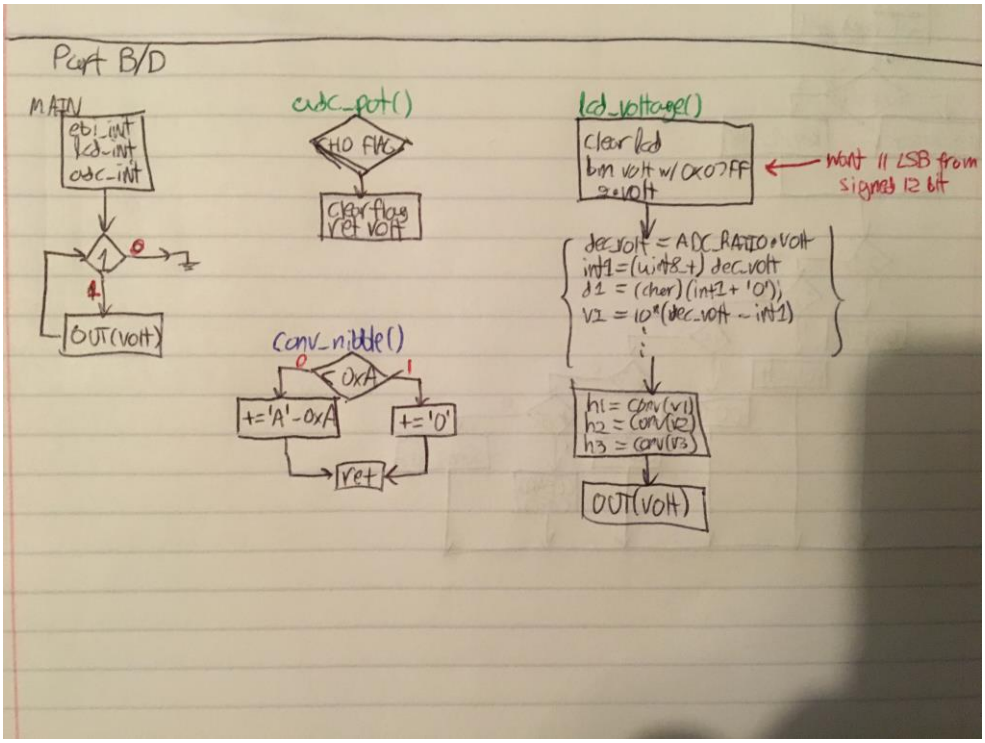


g) Pseudocode/Flowcharts

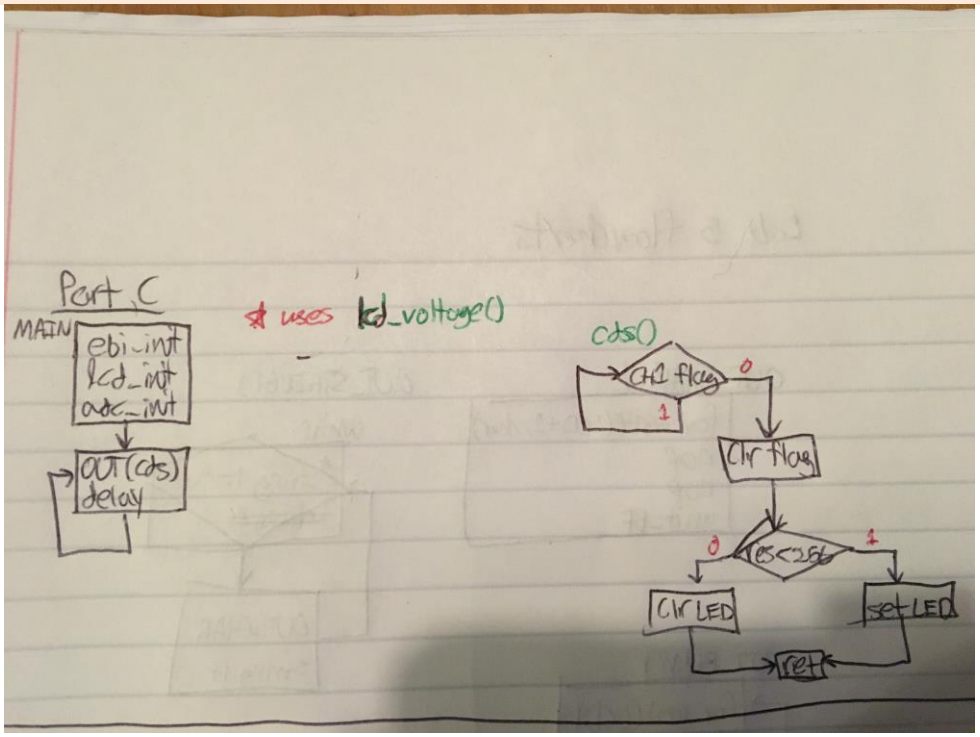
Part (a)



Part (b/d)

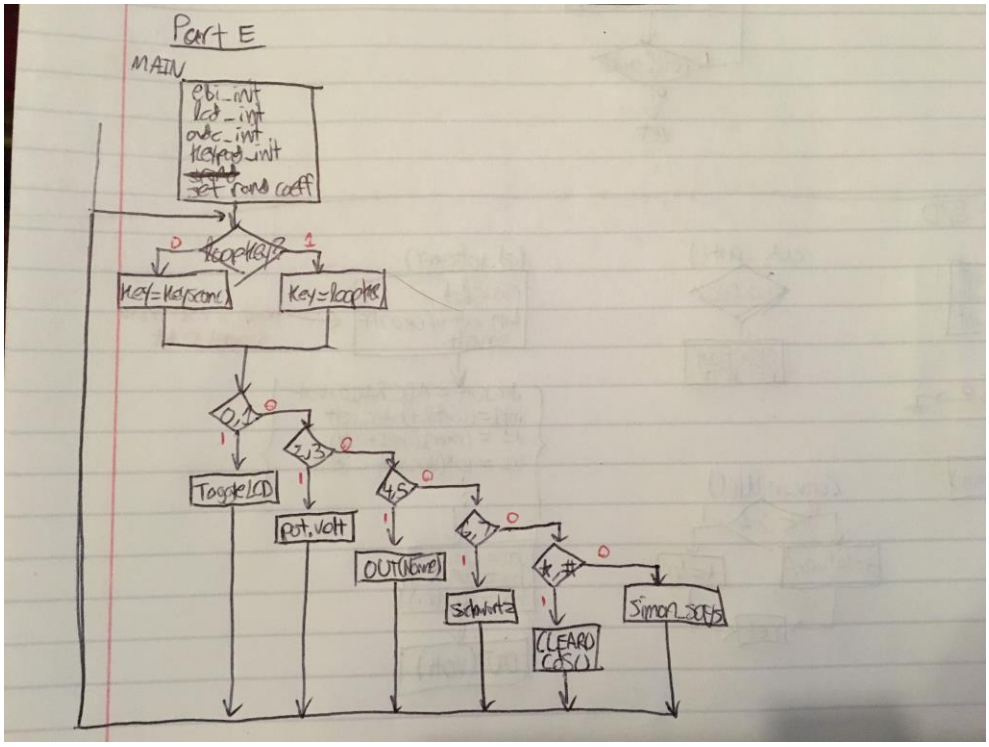

Part B/D
MAIN
ebi_init
lcd_init
adc_init
OUT(volt)
adc_pot()
HO FLAG
clear flag
ret volt
conv_nibble()
0
< 0xA
1
+='A'-0xA
+='0'
ret
lcd_voltage()
clear lcd
bin volt w/ 0x07FF
want 11 LSB from signed 12 bit
dec_volt = ADC_RATIO*volt
int1=(uint8_t) dec_volt
d1 = (char)(int1+'0');
v1 = 10*(dec_volt - int1)
h1 = conv(v1)
h2 = conv(v2)
h3 = conv(v3)
OUT(volt)

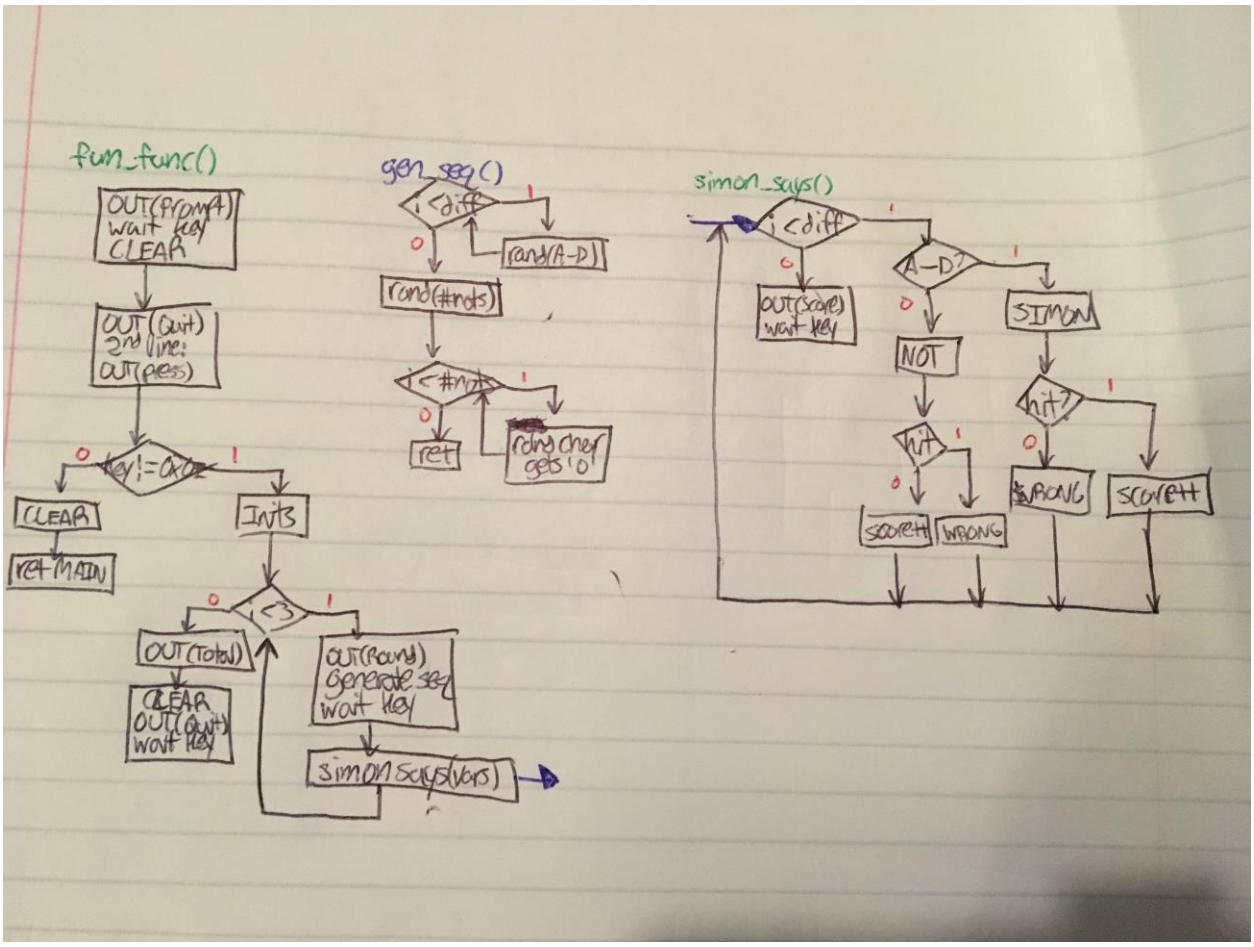
Part (c)


Part C
MAIN
ebi_init
lcd_init
adc_init
OUT(cds)
delay
uses lcd_voltage()
cds()
CH1 flag
0
1
Clr flag
0
cs<256
1
Clr LED
set LED
ret

Part (e)

Lab: 5
Nicholas Imamshah
Section: 6957





h) Program Code

Part (a)

```
/* Lab5_lcd_name.c
 *
 * Lab 5 LCD Name in C
 * Name: Nicholas Imamshah
 * Section: 6957
 * TA Name: Daniel Gonzalez
 * Description: The purpose of this program is to send out my name to the LCD.
 */
////////////////////////////////////////INCLUDES/////////////////////////////////////////
#include <avr/io.h>
#include "ebi_driver.h"

////////////////////////////////////INITIALIZATIONS////////////////////////////////////
#define F_CPU 2000000
#define CS0_Start 0x288000
#define CS0_End     0x289FFF
#define CS1_Start 0x394000
#define CS1_End 0x397FFF
#define LCD_BASEADDR 0x395000

////////////////////////////////////PROTOTYPES/////////////////////////////////////
void ebi_init();
void lcd_init();
void wait_busy();
void OUT_CHAR(char character);
void OUT_STRING(char *string);

////////////////////////////////////MAIN FUNCTION/////////////////////////////////////
int main(void)
{
        ebi_init();
        lcd_init();
        OUT_STRING("Nick Imamshah");
}

////////////////////////////////////FUNCTIONS/////////////////////////////////////
void ebi_init()
{
        PORTH.DIR = 0x37;
        PORTH.OUT = 0x33;
        PORTK.DIR = 0xFF;

        EBI.CTRL = EBI_SRMODE_ALE1_gc | EBI_IFMODE_3PORT_gc;

        EBI.CS0.BASEADDRH = (uint8_t) (CS0_Start>>16) & 0xFF;
        EBI.CS0.BASEADDRL = (uint8_t) (CS0_Start>>8) & 0xFF;
        EBI.CS0.CTRLA = EBI_CS_MODE_SRAM_gc | EBI_CS_ASPACE_8KB_gc;
```

```
	EBI.CS1.BASEADDR = (uint16_t) (CS1_Start>>8) & 0xFFFF;
	EBI.CS1.CTRLA = EBI_CS_MODE_SRAM_gc | EBI_CS_ASPACE_16KB_gc;
}

void lcd_init()
{
	__far_mem_write(0x288000, 0x00);
	wait_busy();
	__far_mem_write(LCD_BASEADDR, 0x38);		// Two lines
	asm volatile ("nop");
	asm volatile ("nop");
	wait_busy();

	__far_mem_write(LCD_BASEADDR, 0x0F);		// Display on; Cursor on; Blink on
	asm volatile ("nop");
	asm volatile ("nop");
	wait_busy();

	__far_mem_write(LCD_BASEADDR, 0x01);		// Clear screen; Cursor home
	asm volatile ("nop");
	asm volatile ("nop");
	wait_busy();
}

void wait_busy()
{
	uint8_t result = 0;
	do
	{
		result = __far_mem_read(LCD_BASEADDR);
	} while (result & 0x80);
}

void OUT_CHAR(char character)
{
	__far_mem_write(LCD_BASEADDR+1, character);
	asm volatile ("nop");
	asm volatile ("nop");
	wait_busy();
}

void OUT_STRING(char *string)
{
	while (*string != '\0')
	{
		OUT_CHAR(*string);
		string++;
	}
}
```

Part (b/d)

```
/* Lab5_lcd_voltage.c
 *
 * Lab 5 LCD Voltage in C
 * Name: Nicholas Imamshah
 * Section: 6957
 * TA Name: Daniel Gonzalez
 * Description: The purpose of this program is to
 */
////////////////////////////////////////INCLUDES/////////////////////////////////////////
#include <avr/io.h>
#include "ebi_driver.h"
#include "ebi_init.h"
#include "lcd_init.h"

////////////////////////////////////INITIALIZATIONS////////////////////////////////////
#define F_CPU 2000000
#define ADC_RATIO 0.001221

////////////////////////////////////PROTOTYPES////////////////////////////////////
void adc_init(void);
uint16_t adc_pot(void);
void lcd_voltage(uint16_t volt);
uint8_t conv_nibble(uint8_t nib);

////////////////////////////////////MAIN FUNCTION////////////////////////////////////
int main(void)
{
	ebi_init();
	lcd_init();
	adc_init();
	while (1)
	{
		lcd_voltage(adc_pot());
	}
}

////////////////////////////////////FUNCTIONS////////////////////////////////////
void adc_init()
{
	// General ADC Configuration
	ADCB.CTRLB = ADC_CONMODE_bm | ADC_FREERUN_bm;		// High Z, No Limit, Signed, Free Run,
12 Bit
	ADCB.REFCTRL = ADC_REFSEL_AREFB_gc;				// Ext. Ref. from AREFB
	ADCB.PRESCALER = ADC_PRESCALER_DIV4_gc;

	// ADC Channel Configuration
	ADCB.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN4_gc;		// Pin 4
	ADCB.CH0.INTCTRL = ADC_CH_INTLVL_LO_gc;			// Enable low-level interrupts
	ADCB.CH0.CTRL = ADC_CH_INPUTMODE_SINGLEENDED_gc;	// Single-ended
```

```
	// Begin Conversions
	ADCB.CTRLA = ADC_CH0START_bm | ADC_ENABLE_bm;		// Start Conversion on channel 0, Enable
ADC

	PORTB.DIRCLR = 0x13;
}

uint16_t adc_pot(void)
{
	while (!ADCB.CH0.INTFLAGS);
	ADCB.CH0.INTFLAGS = 0x01;
	return ADCB.CH0.RES;
}

void lcd_voltage(uint16_t volt)
{
	CLEAR_SCREEN();

	// Convert ADC value to Decimal Voltage
	volt = volt & 0x07FF;					// We can assume positive, so ignore sign bit.
	volt *= 2;
	float dec_volt = ADC_RATIO*volt;		// Apply formula.

	uint8_t int1, int2, int3, h1, h2, h3;
	char d1, d2, d3;
	float volt2, volt3;

	int1 = (uint8_t) dec_volt;				// Determine Decimal representation
	d1 = (char) (int1 + '0');
	volt2 = 10*(dec_volt - int1);
	int2 = (uint8_t) volt2;
	d2 = (char) (int2 + '0');
	volt3 = 10*(volt2 - int2);
	int3 = (uint8_t) volt3;
	d3 = (char) (int3 + '0');

	h1 = conv_nibble(volt>>8);				// Obtain ASCII for Hex representation
	h2 = conv_nibble(volt>>4 & 0x0F);
	h3 = conv_nibble(volt & 0x0F);

	char string[] = {d1, '.', d2, d3, ' ', 'V', ' ', '(', '0', 'x', h1, h2, h3, ')', '\0'};
	OUT_STRING(string);					// Output Voltmeter reading
}

uint8_t conv_nibble(uint8_t nib)
{
	if (nib < 0xA)
	{
		nib += '0';					// Offset by ASCII '0'
	} else
	{
```

```
                nib += 'A' - 0xA;                                   // Subtract out 0xA so that 0xA => 0, 0xB => 1, etc.,
then offset by ASCII 'A'
        }
        return nib;
}
```

Part (c)

```
/* .c
 *
 *  C
 * Name: Nicholas Imamshah
 * Section: 6957
 * TA Name: Daniel Gonzalez
 * Description: The purpose of this program is to
 */
////////////////////////////////////////INCLUDES/////////////////////////////////////////
#include <avr/io.h>
#include "ebi_driver.h"
#include "ebi_init.h"
#include "adc_init.h"
#include "lcd_init.h"
#include "adc_pot.h"

////////////////////////////////////INITIALIZATIONS//////////////////////////////////////
#define F_CPU 2000000

//////////////////////////////////////PROTOTYPES/////////////////////////////////////////
uint16_t cds(void);
void rough_delay(void);

////////////////////////////////////MAIN FUNCTION//////////////////////////////////////
int main(void)
{
        ebi_init();
        adc_init();
        lcd_init();
        while (1)
        {
                lcd_voltage(cds());
                rough_delay();
        }
}

////////////////////////////////////FUNCTIONS//////////////////////////////////////
uint16_t cds(void)
{
        while (!ADCB.CH1.INTFLAGS);
        ADCB.CH1.INTFLAGS = 0x01;
        if (ADCB.CH1.RES < 256)
```

```
		{
			__far_mem_write(CS0_Start, 0x01);
		} else
		{
			__far_mem_write(CS0_Start, 0x00);
		}
		return ADCB.CH1.RES;
}

void rough_delay(void)
{
		for (int i = 0; i < 15000; i++)
		{
			asm volatile ("nop");
		}
}
```

Part (e)

```
/* Lab5_lcd_keypad.c
 *
 * Lab 5 LCD Function using a Keypad
 * Name: Nicholas Imamshah
 * Section: 6957
 * TA Name: Daniel Gonzalez
 * Description: The purpose of this program is to control the LCD's function with a keypad
 */
////////////////////////////////////////INCLUDES/////////////////////////////////////////
#include <avr/io.h>
#include "ebi_driver.h"
#include "ebi_init.h"
#include "lcd_init.h"
#include "adc_init.h"
#include "adc_pot.h"
#include "adc_cds.h"
#include "keypad.h"
#include <time.h>
#include <stdlib.h>

////////////////////////////////////INITIALIZATIONS//////////////////////////////////////
#define F_CPU 2000000

//////////////////////////////////////PROTOTYPES/////////////////////////////////////////
void rough_delay(void);
void long_delay(void);
void fun_func(void);
void gen_seq(uint8_t diff, char *says);
void wait_key(void);
uint8_t simon_says(uint8_t diff, uint8_t speed, char *says);

char *prompt = "Simon Says!";
```

```
char *start = "Press any key";
char *quit = "Press * to quit.";

////////////////////////////////////MAIN FUNCTION/////////////////////////////////////
int main(void)
{
	ebi_init();
	lcd_init();
	adc_init();
	keypad_init();
	srand(time(NULL));


	uint8_t key, loop_key = 0xFF;
	while (1)
	{
		if (loop_key == 0xFF)				// If no loop key, scan for key
		{
			do
			{
				key = keyscan();
			} while (key == 0xFF);
		}
		else
		{					// Else a loop key has been read, use it
			key = loop_key;
			loop_key = 0xFF;
		}

		if (key <= 0x01)				// Toggle LCD On/Off
		{
			lcd_toggle();
		}
		else if (key <= 0x03)			// Display ADC Voltage reading from Potentiometer
		{
			do
			{
				lcd_voltage(adc_pot());
				rough_delay();
				loop_key = keyscan();
			} while (loop_key == 0xFF);
		}
		else if (key <= 0x05)			// Send "Nick Imamshah" to LCD Screen
		{
			OUT_STRING("Nick Imamshah");
		}
		else if (key <= 0x07)			// Send phrase to LCD Screen on both lines
		{
			OUT_STRING("May the Schwartz");
```

```
                        OUT_COMMAND(0xC0);
                        OUT_STRING("be with you!");
                }
                else if (key >= 0x0E && key < 0xFF)                     // Clear the LCD Screen and control LED with CdS ADC
reading
                {
                        CLEAR_SCREEN();
                        uint16_t cds_volt = cds();
                        if (cds_volt < 256)
                        {
                                __far_mem_write(CS0_Start, 0x01);
                        } else
                        {
                                __far_mem_write(CS0_Start, 0x00);
                        }
                }
                else
                {                                                                                                       //
Perform a custom operation
                        fun_func();
                }

                keyhold();
        }
}

////////////////////////////////////FUNCTIONS/////////////////////////////////////
void rough_delay(void)
{
        for (int i = 0; i < 15000; i++)
        {
                asm volatile ("nop");
        }
}

void long_delay(void)
{
        for (int i = 0; i < 20; i++)
        {
                for (int j = 0; j < 5000; j++)
                {
                        asm volatile ("nop");
                }
        }
}

void fun_func(void)
{
        OUT_STRING(prompt);
        wait_key();
```

```
	CLEAR_SCREEN();

	uint8_t key = 0xFF, diff = 5;

	OUT_STRING(quit);
	OUT_COMMAND(0xC0);
	OUT_STRING("Press any key");
	do
	{
		key = keyscan();
	} while (key == 0xFF);
	//wait_key();
	key == 0xFF;

	while (key != 0x0E)
	{
		CLEAR_SCREEN();

		uint8_t total_score = 0;				// Initialize the total score
		char *says = 0;

		for (int i = 0; i < 3; i++)				// Play 3 rounds
		{
			OUT_STRING("Round ");
			OUT_CHAR((char) (i+1) + '0');
			gen_seq(diff + i*2, says);
			wait_key();
			CLEAR_SCREEN();
			total_score += simon_says(diff + i*2, 14-i*3, says);	// Play game & sum scores
			keyhold();						// Wait for key press to
continue to next round
			CLEAR_SCREEN();
		}

		OUT_STRING("Total Score: ");		// Output total score
		if (total_score >= 20)				// Handles various ranges of numbers
		{
			uint8_t d1 = total_score/10;
			uint8_t d2 = total_score - 20;
			OUT_CHAR(conv_nibble(d1));
			OUT_CHAR(conv_nibble(d2));
		}
		else if (total_score > 9 && total_score < 20)
		{
			uint8_t d1 = total_score/10;
			uint8_t d2 = total_score - 10;
			OUT_CHAR(conv_nibble(d1));
			OUT_CHAR(conv_nibble(d2));
		} else
		{
			OUT_CHAR((char) total_score + '0');
```

```
                }
                long_delay();                                   // Wait before moving on to options

                CLEAR_SCREEN();
                OUT_STRING(quit);
                wait_key();

                do
                {                                                                       // Scan for keys
                        key = keyscan();
                } while (key == 0xFF);
        }
        CLEAR_SCREEN();
}

void gen_seq(uint8_t diff, char *says)
{
        for (int i = 0; i < diff; i++)
        {
                says[i] = (rand() % 4) + 'A';
        }
        int nulls = rand() % diff;
        for (int i = 0; i < nulls; i++)
        {
                says[rand() % diff] = '0';
        }
        asm volatile ("nop");
}

void wait_key(void)
{
        rough_delay();
        OUT_COMMAND(0xC0);                      // Break line first
        OUT_STRING(start);                      // Output 'start' string
        uint8_t key;
        do
        {                                                       // Wait for a key press
                key = keyscan();
        } while (key == 0xFF);
        keyhold();
}

uint8_t simon_says(uint8_t diff, uint8_t speed, char *says)
{
        uint8_t key = 0xFF, score = 0;          // Initialize variables
        for (uint8_t i = 0; i < diff; i++)    // 'diff'iculty determines length of Round
        {
                if (says[i] >= 'A' && says[i] <= 'D')                   // Simon said
                {
                        OUT_STRING("Simon says: ");
                        OUT_CHAR(says[i]);
```

```
                keyhold();
                for (int i = 0; i < speed; i++)                 // Speed determines how long player
                {
        //      has to enter a key
                        for (int j = 0; j < 1000; j++)
                        {
                                key = keyscan();
                                if (key != 0xFF) break;
                        }
                }
                if (conv_nibble(key & 0x0F) == says[i])    // Check if correct action
                {
                        OUT_COMMAND(0xC0);
                        OUT_STRING("CORRECT! :-)");
                        score += 1;
                }
                else
                {
                        OUT_COMMAND(0xC0);
                        OUT_STRING("WRONG! :-(");
                }
                long_delay();                                                   // Wait before
moving to next letter
                CLEAR_SCREEN();
        }
        else
        {
        // Simon DID NOT say
                char c = (rand() % 4) + 'A';
                OUT_STRING("Press this: ");
                OUT_CHAR(c);
                keyhold();
                for (int i = 0; i < speed; i++)                 // Speed determines how long player
                {
        //      has to enter a key
                        for (int j = 0; j < 1000; j++)
                        {
                                key = keyscan();
                                if (key != 0xFF) break;
                        }
                }
                if (key == 0xFF)                                                // Check if correct action
                {
                        OUT_COMMAND(0xC0);
                        OUT_STRING("Good Job!");
                        score += 1;
                }
                else
                {
                        OUT_COMMAND(0xC0);
                        OUT_STRING("Simon didn't say");
```

```
                                }
                                long_delay();                                   // Wait before
moving to next letter
                                CLEAR_SCREEN();
                        }
                }
                OUT_STRING("Score: ");                                  // Output Round score
                OUT_CHAR((char) score + '0');
                OUT_COMMAND(0xC0);
                wait_key();                                                     // Wait
for player input before continuing
                return score;
}
```

i) Appendix

ebi_init.h

```
/* ebi_init.h
 *
 * EBI Initialization Header
 * Name: Nicholas Imamshah
 * Section: 6957
 * TA Name: Daniel Gonzalez
 * Description: The purpose of this program is to configure the EBI for I/O Ports and LCD.
 */
////////////////////////////////////////INCLUDES////////////////////////////////////////
#include <avr/io.h>

////////////////////////////////////INITIALIZATIONS/////////////////////////////////////
#define F_CPU 2000000
#define CS0_Start 0x288000
#define CS0_End     0x289FFF
#define CS1_Start 0x394000
#define CS1_End 0x397FFF

//////////////////////////////////////PROTOTYPES//////////////////////////////////////
void ebi_init();

//////////////////////////////////////FUNCTIONS//////////////////////////////////////
void ebi_init()
{
        PORTH.DIR = 0x37;
        PORTH.OUT = 0x33;
        PORTK.DIR = 0xFF;

        EBI.CTRL = EBI_SRMODE_ALE1_gc | EBI_IFMODE_3PORT_gc;

        EBI.CS0.BASEADDRH = (uint8_t) (CS0_Start>>16) & 0xFF;
        EBI.CS0.BASEADDRL = (uint8_t) (CS0_Start>>8) & 0xFF;
        EBI.CS0.CTRLA = EBI_CS_MODE_SRAM_gc | EBI_CS_ASPACE_8KB_gc;
```

```
	EBI.CS1.BASEADDR = (uint16_t) (CS1_Start>>8) & 0xFFFF;
	EBI.CS1.CTRLA = EBI_CS_MODE_SRAM_gc | EBI_CS_ASPACE_16KB_gc;
}
```

lcd_init.h

```
/* lcd_init.h
 *
 * LCD Initialization Header
 * Name: Nicholas Imamshah
 * Section: 6957
 * TA Name: Daniel Gonzalez
 * Description: The purpose of this program is to initialize the LCD.
 */
////////////////////////////////////INCLUDES/////////////////////////////////////
#include <avr/io.h>

////////////////////////////////INITIALIZATIONS//////////////////////////////////
#define F_CPU 2000000
#define LCD_BASEADDR 0x395000

//////////////////////////////////PROTOTYPES////////////////////////////////////
void lcd_init();
void wait_busy();
void OUT_CHAR(char character);
void OUT_STRING(char *string);
void OUT_COMMAND(char command);
void CLEAR_SCREEN(void);
void lcd_toggle(void);

uint8_t lcd_disp = 0x07;

//////////////////////////////////FUNCTIONS////////////////////////////////////
void lcd_init()
{
	wait_busy();
	__far_mem_write(LCD_BASEADDR, 0x38);		// Two lines
	asm volatile ("nop");
	asm volatile ("nop");
	wait_busy();

	__far_mem_write(LCD_BASEADDR, 0x0F);		// Display on; Cursor on; Blink on
	asm volatile ("nop");
	asm volatile ("nop");
	wait_busy();

	__far_mem_write(LCD_BASEADDR, 0x01);		// Clear screen; Cursor home
	asm volatile ("nop");
	asm volatile ("nop");
	wait_busy();
}
```

```
void wait_busy()
{
	uint8_t result = 0;
	do
	{
		result = __far_mem_read(LCD_BASEADDR);
	} while (result & 0x80);                                    // Poll the BF of the LCD
}

void OUT_CHAR(char character)
{
	wait_busy();
	__far_mem_write(LCD_BASEADDR+1, character);
	asm volatile ("nop");
	asm volatile ("nop");
	wait_busy();
}

void OUT_STRING(char *string)
{
	while (*string != '\0')                                     // Loop until null character is encountered
	{
		OUT_CHAR(*string);
		string++;
	}
}

void OUT_COMMAND(char command)
{
	wait_busy();
	__far_mem_write(LCD_BASEADDR, command);
	asm volatile ("nop");
	asm volatile ("nop");
	wait_busy();
}

void CLEAR_SCREEN(void)
{
	wait_busy();
	__far_mem_write(LCD_BASEADDR, 0x01);
	asm volatile ("nop");
	asm volatile ("nop");
	wait_busy();
}

void lcd_toggle(void)
{
	lcd_disp = lcd_disp ^ 0x07;
	uint8_t disp_comm = 0x08 | lcd_disp;
```

```
	wait_busy();
	OUT_COMMAND(disp_comm);
}
```

adc_pot.h

```
/* Lab5_lcd_voltage.c
 *
 * Lab 5 LCD Voltage in C
 * Name: Nicholas Imamshah
 * Section: 6957
 * TA Name: Daniel Gonzalez
 * Description: The purpose of this program is to
 */
////////////////////////////////////////INCLUDES/////////////////////////////////////////
#include <avr/io.h>
#ifndef LCD_BASEADDR
#include "lcd_init.h"
#endif

////////////////////////////////////INITIALIZATIONS////////////////////////////////////
#define F_CPU 2000000
#define ADC_RATIO 0.001221

////////////////////////////////////PROTOTYPES////////////////////////////////////
uint16_t adc_pot(void);
void lcd_voltage(uint16_t volt);
uint8_t conv_nibble(uint8_t nib);

////////////////////////////////////FUNCTIONS////////////////////////////////////
uint16_t adc_pot(void)
{
	while (!ADCB.CH0.INTFLAGS);
	ADCB.CH0.INTFLAGS = 0x01;
	return ADCB.CH0.RES;
}

void lcd_voltage(uint16_t volt)
{
	CLEAR_SCREEN();

	// Convert ADC value to Decimal Voltage
	volt = volt & 0x07FF;                                   // We can assume positive, so ignore sign bit.
	volt *= 2;                                              // Multiply by 2 to account for ADC /2
	float dec_volt = ADC_RATIO*volt;                // Apply formula.

	uint8_t int1, int2, int3, h1, h2, h3;
	char d1, d2, d3;
	float volt2, volt3;

	int1 = (uint8_t) dec_volt;                              // Determine Decimal representation
```

```
	d1 = (char) (int1 + '0');
	volt2 = 10*(dec_volt - int1);
	int2 = (uint8_t) volt2;
	d2 = (char) (int2 + '0');
	volt3 = 10*(volt2 - int2);
	int3 = (uint8_t) volt3;
	d3 = (char) (int3 + '0');

	h1 = conv_nibble(volt>>8);				// Obtain ASCII for Hex representation
	h2 = conv_nibble(volt>>4 & 0x0F);
	h3 = conv_nibble(volt & 0x0F);

	char string[] = {d1, '.', d2, d3, ' ', 'V', ' ', '(', '0', 'x', h1, h2, h3, ')', '\0'};
	OUT_STRING(string);					// Output Voltmeter reading
}

uint8_t conv_nibble(uint8_t nib)
{
	if (nib < 0xA)
	{
		nib += '0';				// Offset by ASCII '0'
	} else
	{
		nib += 'A' - 0xA;			// Subtract out 0xA so that 0xA => 0, 0xB => 1, etc.,
then offset by ASCII 'A'
	}
	return nib;
}
```

adc_cds.h

```
/* cds.h
 *
 * CdS Header
 * Name: Nicholas Imamshah
 * Section: 6957
 * TA Name: Daniel Gonzalez
 * Description: The purpose of this program is to interface the CdS resistor
 */
////////////////////////////////////////INCLUDES////////////////////////////////////////
#include <avr/io.h>

////////////////////////////////////INITIALIZATIONS/////////////////////////////////////
#define F_CPU 2000000

////////////////////////////////////PROTOTYPES/////////////////////////////////////
uint16_t cds(void);

////////////////////////////////////FUNCTIONS/////////////////////////////////////
uint16_t cds(void)
{
```

```
	while (!ADCB.CH1.INTFLAGS);
	ADCB.CH1.INTFLAGS = 0x01;
	return ADCB.CH1.RES;
}
```

keypad.h

```
/* keypad.c
 *
 * Keypad in C
 * Name: Nicholas Imamshah
 * Section: 6957
 * TA Name: Daniel Gonzalez
 * Description: The purpose of this program is to interface the XMEGA processor
 *			with an external Keypad.
 */
////////////////////////////////////////INCLUDES/////////////////////////////////////////
#include <avr/io.h>

////////////////////////////////////INITIALIZATIONS////////////////////////////////////
#define F_CPU 2000000

uint8_t keys[] = {0x1, 0x4, 0x7, 0xE, 0x2, 0x5, 0x8, 0x0, 0x3, 0x6, 0x9, 0xF, 0xA, 0xB, 0xC, 0xD};

////////////////////////////////////PROTOTYPES/////////////////////////////////////
void keypad_init(void);
uint8_t keyscan(void);
void keyhold(void);

////////////////////////////////////FUNCTIONS/////////////////////////////////////
void keypad_init(void)
{
	PORTF.PIN7CTRL = PORT_OPC_PULLUP_gc;		// Set OPC to Pull-Up for all Keypad pins
	PORTF.PIN6CTRL = PORT_OPC_PULLUP_gc;
	PORTF.PIN5CTRL = PORT_OPC_PULLUP_gc;
	PORTF.PIN4CTRL = PORT_OPC_PULLUP_gc;
	PORTF.PIN3CTRL = PORT_OPC_PULLUP_gc;
	PORTF.PIN2CTRL = PORT_OPC_PULLUP_gc;
	PORTF.PIN1CTRL = PORT_OPC_PULLUP_gc;
	PORTF.PIN0CTRL = PORT_OPC_PULLUP_gc;

	PORTF.DIRSET = 0x0F;				// Set LSNibble of PortF as Output
}

uint8_t keyscan(void)
{
	uint8_t input, index, line, key = 0xFF, i = 0;
	for (i = 0; i < 4; i++)	// Iterate columns
	{
		line = ~(0x01 << i) & 0x0F;		// Iterate shift 0x08 by i and not to hit each col
		PORTF.OUT = line;			// Output value for col
```

```
                asm volatile ("nop");
                input = PORTF.IN & 0xF0;      // Read Input and bitmask off Output bits

                if (input < 0xF0)
                {
                        switch (input)
                        {
                                case 0xE0:
                                        index = 0x00;
                                        break;
                                case 0xD0:
                                        index = 0x01;
                                        break;
                                case 0xB0:
                                        index = 0x02;
                                        break;
                                case 0x70:
                                        index = 0x03;
                                        break;
                        }
                        key = keys[index+4*i];
                        return key;
                }
        }
        return key;
}

void keyhold(void)
{
        while ((PORTF.IN & 0xF0) < 0xF0);
}
```