**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?
   1. 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.
   1. 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?
   1. WE, RS (Assuming always in need of Command and Data differentiation), 8 Data lines, VDD, VO, VSS.
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.
   1. 2.0625V/212 = 0.0005035. DV = 83410.

**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.

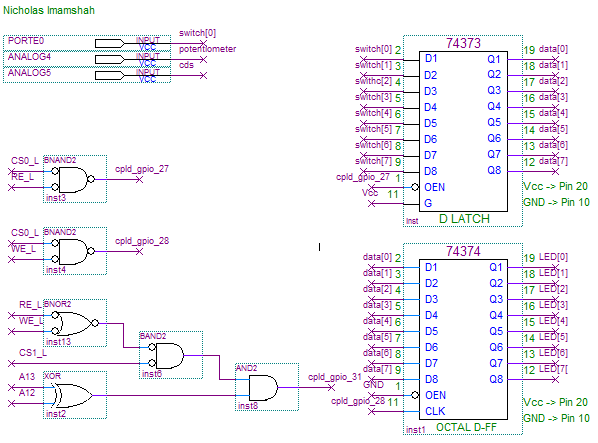
With the ADC: Initially experienced issues with understanding how to use the ADC and configure it in C. After gaining some understanding I encountered trouble with reading the value; when using prescaler as DIV512, the values being recorded were wildly variable, but with DIV4 it became much more consistent.

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.

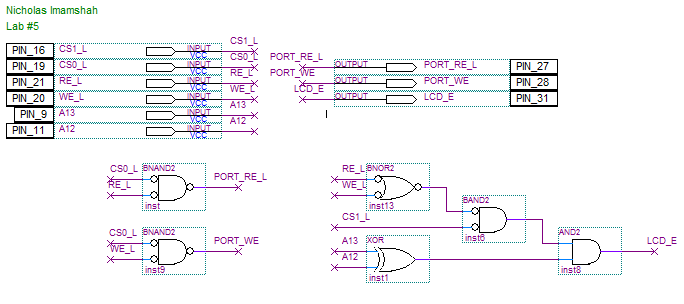
**d) Future Work/Applications**

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**

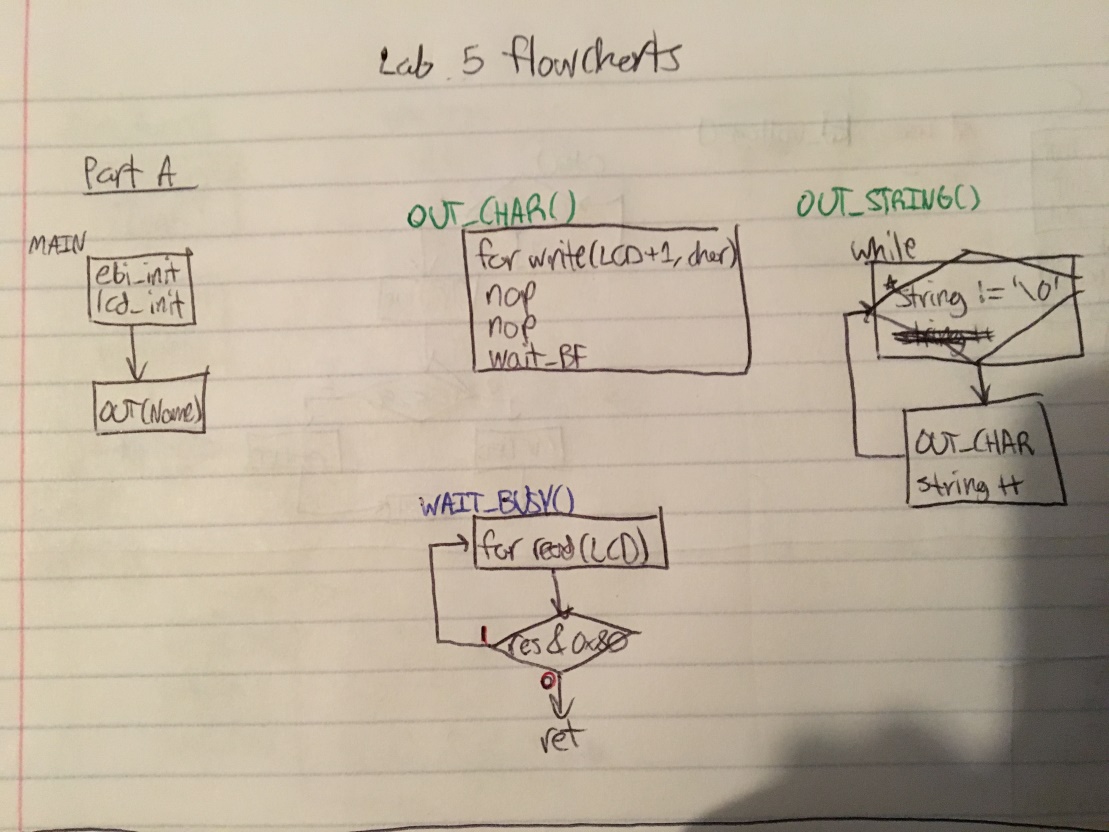
****

**f) Decoding Logic**

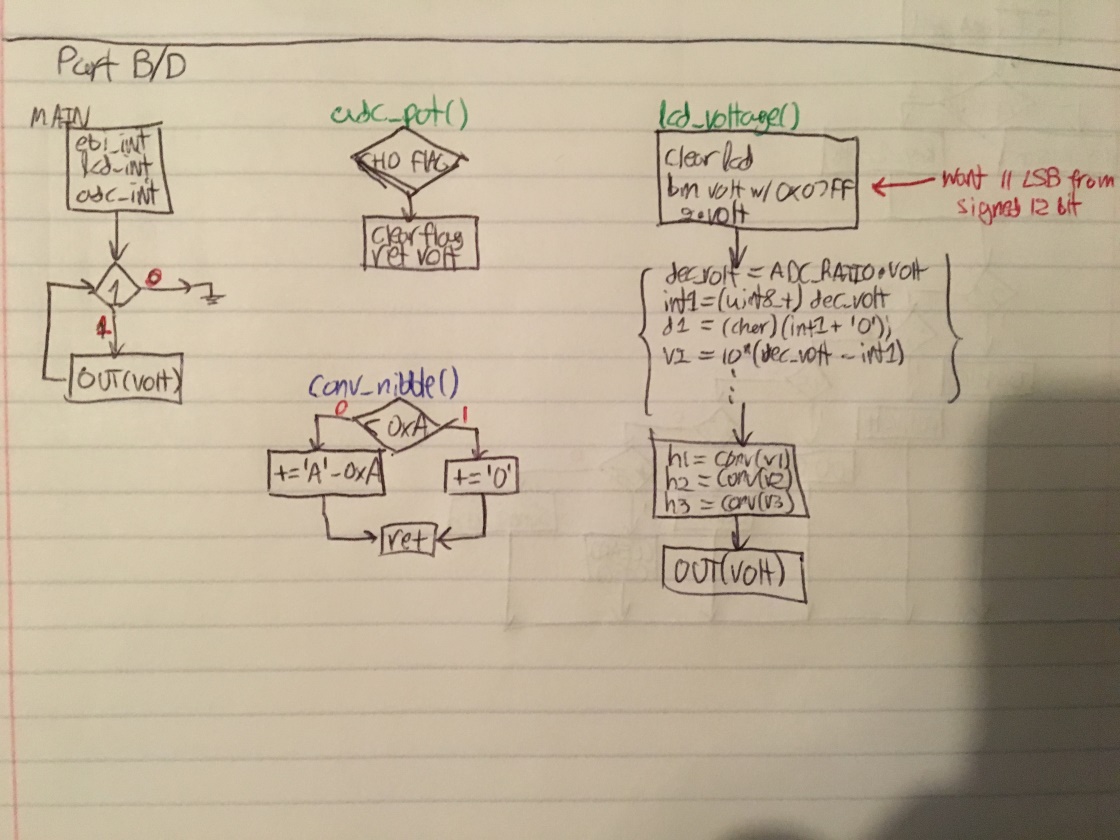
****

**g) Pseudocode/Flowcharts**

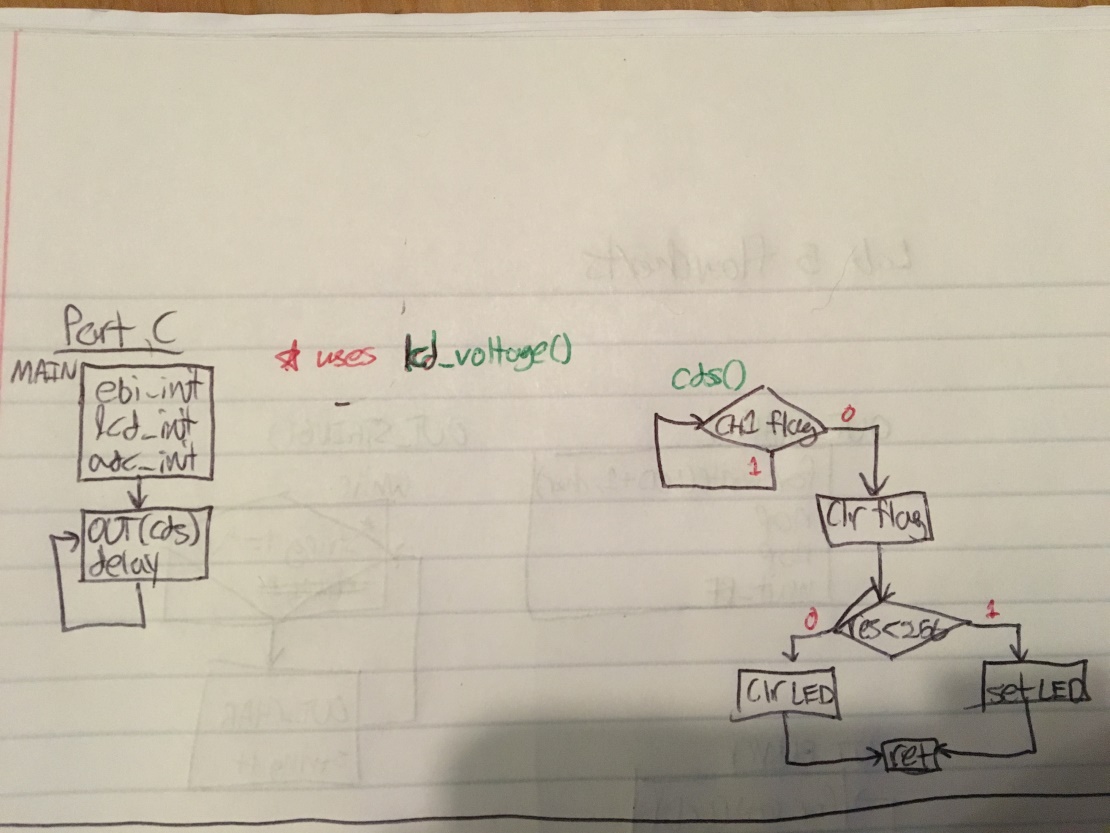
**Part (a)**

****

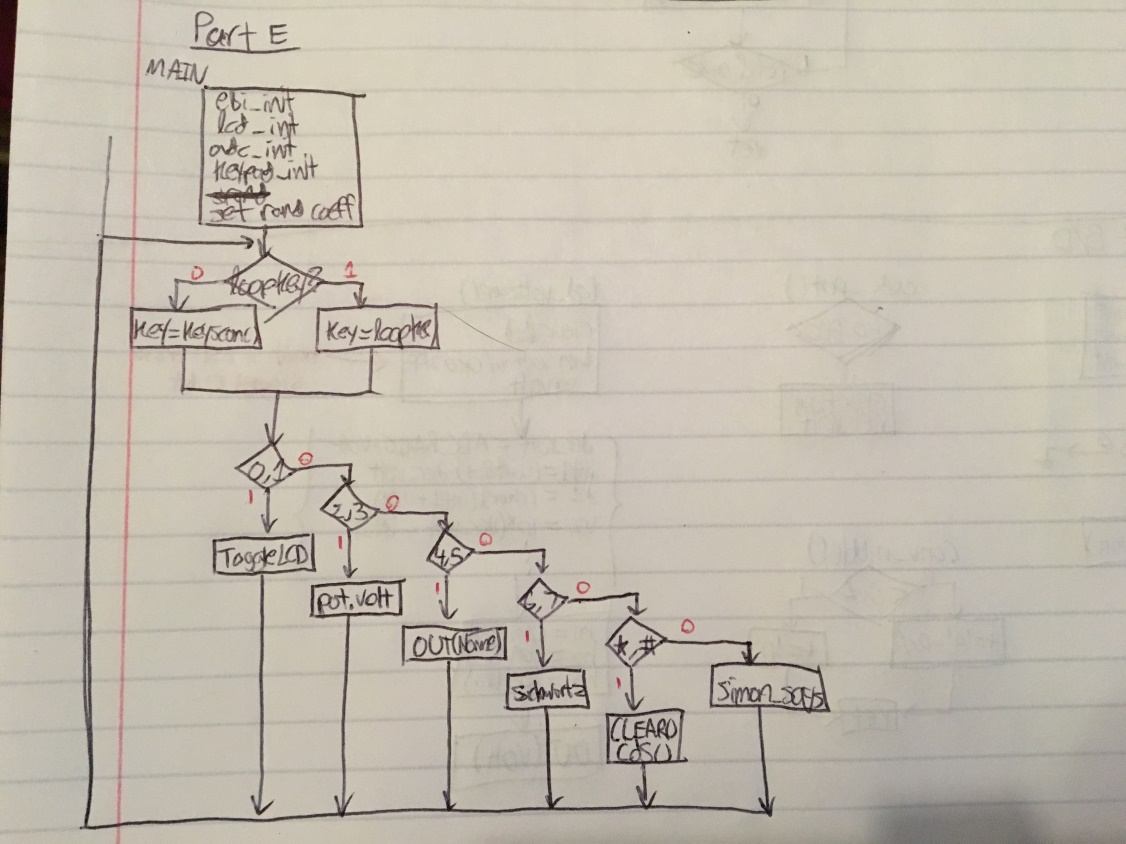
**Part (b/d)**

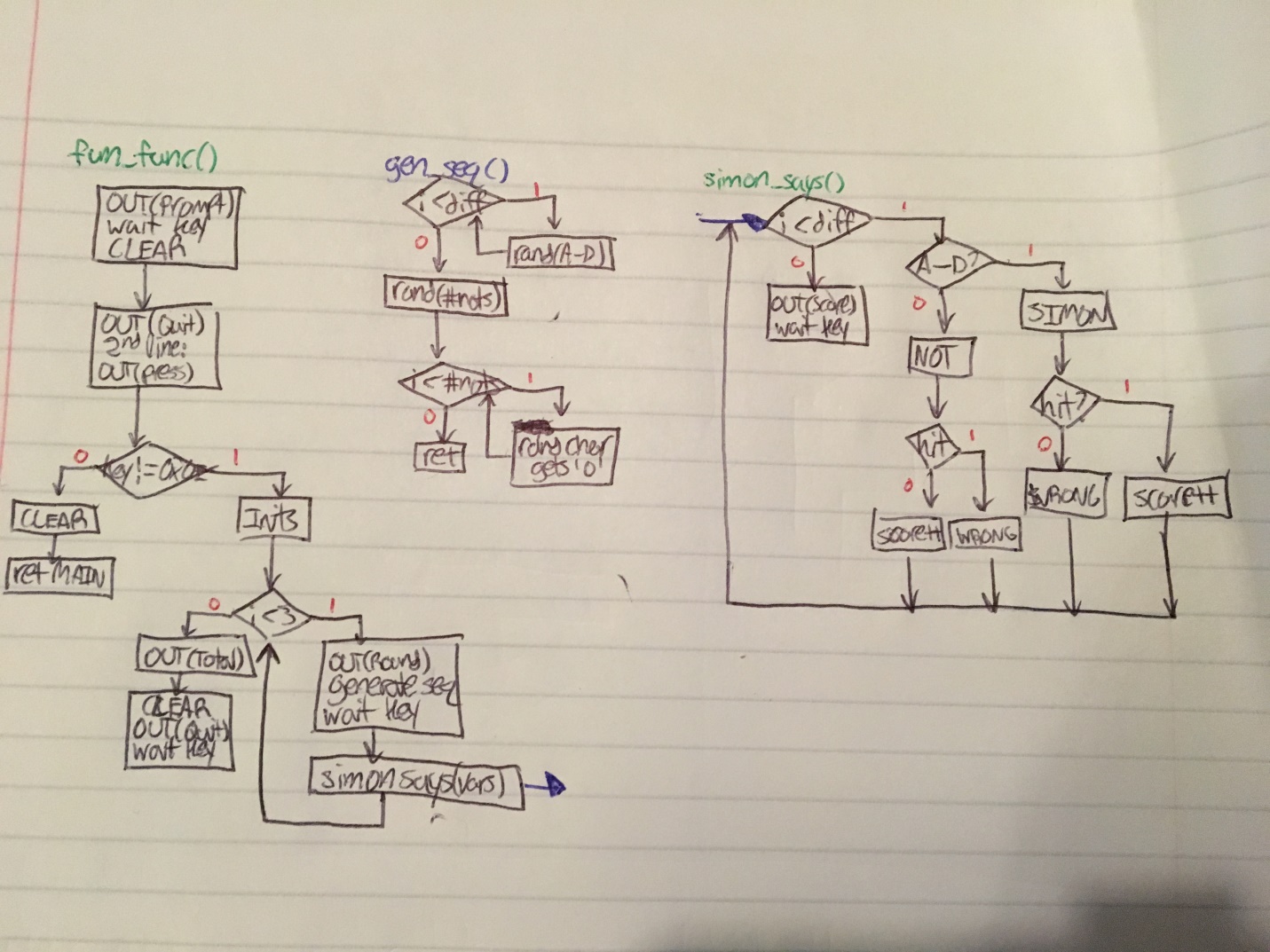
****

**Part (c)**

****

**Part (e)**

****

****

**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);

}