Lab 5
Jineidy Mak
Secton 7F31

Pre-Lab Questions:

1. What is the difference in conversion ranges between 12- bit unsigned and signed conversion modes? List both ranges.

Since 2^12 is 2000 different resolutions. A signed 12-bit number can range from -2k to 2k. A unsigned number can go from 0 to 4k.

2. Assume you wanted a voltage reference range from -2 V to 1 V, with a signed 12-bit ADC. What are the voltages if the ADC output is 0xA25 and 0xB42?

(VREF/2048) - 1. VREF is decimal equivalent of 0xA25 and B42. 2597/2048 - 1 = .268 V
2882/2048 - 1 = .407 V

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?

R/~W, D7:0,

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 unsigned mode was used, what is the resolution (volts/bit)

(2.0625 + 1) * 4096 = 12544.

Problems Encountered:

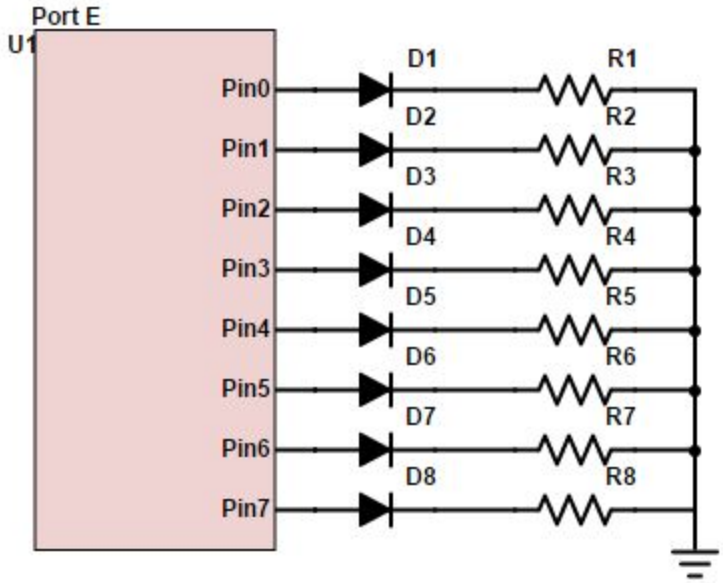
Flip-Flop and Tri-state can interfere with LCD data lines. Make sure they are disabled when using the LCD. Proper timing must be met.

Future Work/Applications:

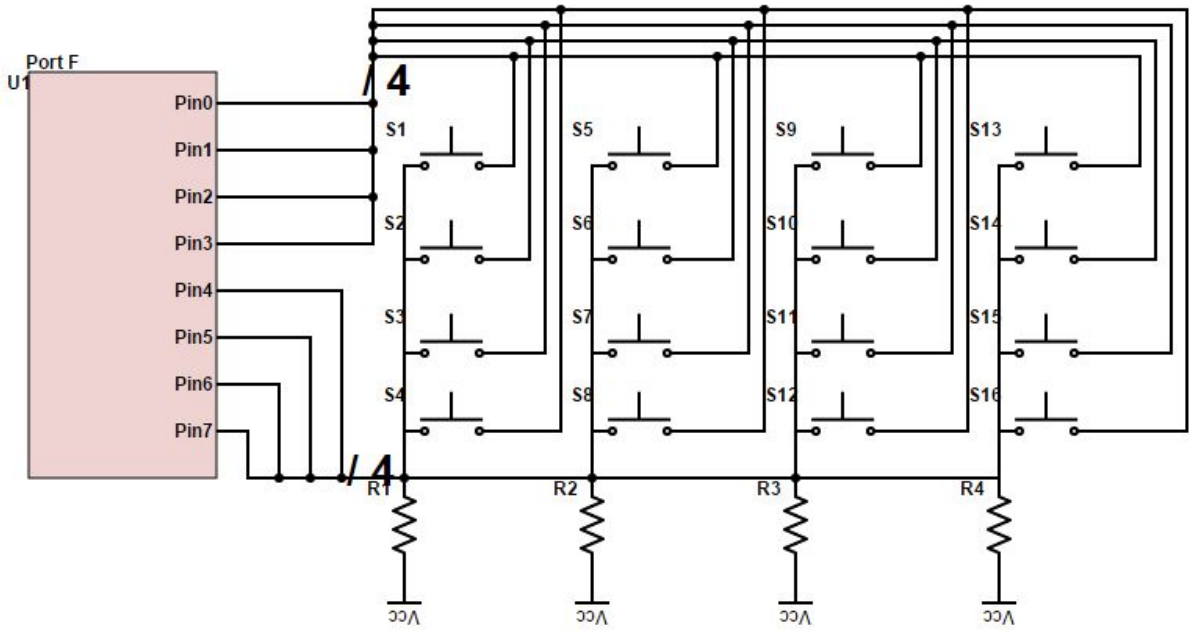
Something tells me this is how a lot of the LCDs we see on machines and electronics are interfaced.

Schematics:

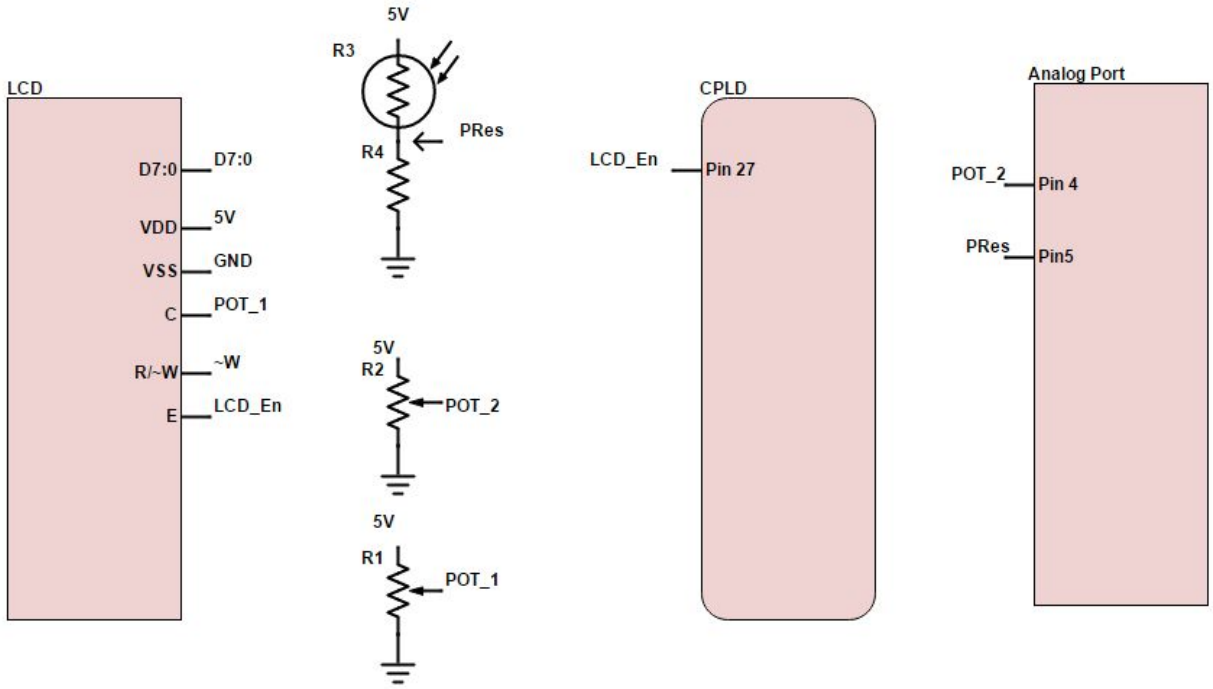
LEDs from PortE to Ground.



Keypad Wiring to Port F, with Pull-Up Resistors.



LCD and CPLD


LCD
D7:0
D7:0
VDD
5V
VSS
GND
C
POT_1
R/~W
~W
E
LCD_En
5V
R3
PRes
R4
5V
R2
POT_2
5V
R1
POT_1
CPLD
LCD_En
Pin 27
Analog Port
POT_2
Pin 4
PRes
Pin5

Decoding Logic:

```
library ieee;
use ieee.std_logic_1164.all;

entity lab3 is
  port (

    CS0_L   : in  std_logic;
    CS1_L   : in  std_logic;
    RE_L      : in  std_logic;
    WE_L      : in  std_logic;
    A15                 : in std_logic;
    A14                 : in std_logic;
    A13                 : in std_logic;
    A12                 : in std_logic;
    LCD_En  : out std_logic

    );

end lab3;


architecture BHV of lab3 is

begin

process(CS0_L, CS1_L, RE_L, WE_L, A15, A14, A13, A12)

            variable CS0_H : std_logic;
            variable CS1_H : std_logic;
            variable A15_L : std_logic;
            variable A14_L : std_logic;
            variable A13_L : std_logic;
            variable RE_H : std_logic;
            variable WE_H : std_logic;

begin

  CS0_H := not(CS0_L);
  CS1_H := not(CS1_L);
  RE_H := not(RE_L);
  WE_H := not(WE_L);
  A15_L := not(A15);
  A14_L := not(A14);
  A13_L := not(A13);

  LCD_En <= CS1_H and (RE_H or WE_H) and ((A15_L and A14_L and A13) or (A15_L and A14));

 end process;

end BHV;
```

Pseudocode/Flowcharts:

PartA:
Initialize EBI.
Initialize the LCD.
Send String to OUT_STRING fucntion.
OUT_STRING function calls OUT_CHAR function to print string according to string lenght.
If lenght is reach then exit the function OUT_STRING.
Delay_200uS.

PartB-D:
Initialize EBI.
Initialize the LCD.
Initialize the ADC using unsigned 8-bit free-running mode. Use Pin 4 at Port Analog B as input.
Read value which is in binary. Convert to Hex.
Output each character at a time. 0 then x then Hex1 then Hex2. Hex1 and Hex2 are our converted numbers.
OUT_STRING function calls OUT_CHAR function to print string according to string lenght.
If lenght is reach then exit the function OUT_STRING.
Delay_200uS.

Initialize PortE for Part C.

PartE:
Initialize EBI.
Initialize the LCD.
Initialize PortF
Initialize PortE
Initialize the ADC using unsigned 8-bit free-running mode. Use Pin 4 at Port Analog B as input.
Read value which is in binary. Convert to Hex.
Use a switch statement. Use a function called Keypad_Poll to figure out which key is pressed similar to the way it is used in Lab2.
When Case 0,1,2,3 see Part A.
When Case 4,5 see PartD.
When Case 6,7 see PartC.
When Case *,# turn LCD off.
When Case 8 display special function.

Delay_200uS.

Program Code:

PartA

```
/*
 * lab5a.c
 *
 * Created: 7/11/2016 4:31:21 PM
 * Author : James Mak
 */

#include <avr/io.h>
#include "ebi_driver.h"

#define F_CPU 2000000
#define CS0_Start 0x8000
#define CS1_Start 0x420000
#define LCD_Start 0x422000
#define LCD_Write 0x422001

void Delay_1sec(void)
{
	volatile uint32_t count;

	for(count = 0; count < F_CPU; count++);

}

void Delay_XuS(int a) // Delay a certain amount of uS.
{
	for(int i = 0; i < a; i++);
}

void EBI_INIT(void)
{
	PORTH.DIR = 0x37;
	PORTH.OUT = 0x33;
	PORTK.DIR = 0xFF;

	EBI.CTRL = EBI_CS_MODE_SRAM_gc | EBI_IFMODE_3PORT_gc; //3 port (H,J,K) SRAM mode.

	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_16KB_gc; //SRAM mode 16k address space.

	EBI.CS1.BASEADDR = (uint16_t) (CS1_Start>>8) & 0xFFFF;
	EBI.CS1.CTRLA = EBI_CS_MODE_SRAM_gc | EBI_CS_ASPACE_64KB_gc; //SRAM mode 64k address space.
}

void BF_POLL(void)
{
	uint8_t BF = __far_mem_read(LCD_Start);

	while((BF & 0x80) == 0x80) //Poll the busy flag (bit 7 until it is clear.
	{
		BF = __far_mem_read(LCD_Start);
	}
```

```
}

void LCD_INIT(void)
{
	__far_mem_write(LCD_Start, 0x38); //Configure function.
	BF_POLL();
	__far_mem_write(LCD_Start, 0x0F); //Configure display.
	BF_POLL();
	__far_mem_write(LCD_Start, 0x01); //Clear display.
	BF_POLL();
	__far_mem_write(LCD_Start, 0x06); //Configure entry mode.
	BF_POLL();
}

void OUT_CHAR(char character)
{
	__far_mem_write(LCD_Write, character);
	BF_POLL();
	Delay_XuS(400);
}

void OUT_STRING(char str[]) //Store the characters in strings.
{
	__far_mem_write(LCD_Start, 0x01);    // Select function clear display, clear display.
	BF_POLL();
	Delay_XuS(400);

	for(int i = 0; i < strlen(str); i++)
	{
		OUT_CHAR(str[i]);
	}
}

int main(void)
{
	EBI_INIT();
	LCD_INIT();
	OUT_STRING("James is awesome!");
	while(1)
	{
		// Infinite loop.
	}
}
```

PartB and D

```
/*
 * lab5d.c
 *
 * Created: 7/11/2016 4:31:21 PM
 * Author : James Mak
 */

#include <avr/io.h>
#include "ebi_driver.h"
#include <string.h>

#define F_CPU 2000000
```

```
#define CS0_Start 0x8000
#define CS1_Start 0x420000
#define LCD_Start 0x422000
#define LCD_Write 0x422001

void Delay_1sec(void)
{
	volatile uint32_t count;

	for(count = 0; count < F_CPU; count++);

}

void Delay_XuS(int a) // Delay a certain amount of uS.
{
	for(int i = 0; i < a; i++);
}

void EBI_INIT(void)
{
	PORTH.DIR = 0x37;
	PORTH.OUT = 0x33;
	PORTK.DIR = 0xFF;

	EBI.CTRL = EBI_CS_MODE_SRAM_gc | EBI_IFMODE_3PORT_gc; //3 port (H,J,K) SRAM mode.

	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_16KB_gc; //SRAM mode 16k address space.

	EBI.CS1.BASEADDR = (uint16_t) (CS1_Start>>8) & 0xFFFF;
	EBI.CS1.CTRLA = EBI_CS_MODE_SRAM_gc | EBI_CS_ASPACE_64KB_gc; //SRAM mode 64k address space.
}

void ADC_INIT(void)
{
	PORTB_DIRCLR = 0x04; //Pin 4 to input.
	ADCB.CTRLA = 0x07; //Channel 0, Flush, Enable ADC.
	ADCB.CTRLB = 0xCC; //Free-running, Unsigned, 8-bit.
	ADCB.REFCTRL = 0x30; //Port B as reference.
	ADCB.PRESCALER = 0x07; //Div/32 Pre-scaler.
	ADCB.EVCTRL = 0x00;
	ADCB.CH0.CTRL = 0x81;
	ADCB.CH0.MUXCTRL = 0x20; //ADC pin 7.
}

void BF_POLL(void)
{
	uint8_t BF = __far_mem_read(LCD_Start);

	while((BF & 0x80) == 0x80) //Poll the busy flag (bit 7 until it is clear.
	{
		BF = __far_mem_read(LCD_Start);
	}
}

/*int8_t fetch_bin_voltage()
```

```
{
	while((ADCB.CH0.INTFLAGS & 0x01) == 0)
	return
}*/

float bin_to_decimal(uint8_t bin_voltage)
{
	float conv_voltage = (float)(bin_voltage * 5) / 255;
	return conv_voltage;
}

void LCD_INIT(void)
{
	__far_mem_write(LCD_Start, 0x38); //Configure function.
	BF_POLL();
	__far_mem_write(LCD_Start, 0x0F); //Configure display.
	BF_POLL();
	__far_mem_write(LCD_Start, 0x01); //Clear display.
	BF_POLL();
	__far_mem_write(LCD_Start, 0x06); //Configure entry mode.
	BF_POLL();
}

void OUT_CHAR(char character)
{
	__far_mem_write(LCD_Write, character);
	BF_POLL();
	Delay_XuS(400);
}

void OUT_STRING(char str[]) //Store the characters in strings.
{
	__far_mem_write(LCD_Start, 0x01);    // Select function clear display, clear display.
	BF_POLL();
	Delay_XuS(400);

	for(int i = 0; i < strlen(str); i++)
	{
		OUT_CHAR(str[i]);
	}
}

void print_voltage(float dec_voltage, uint8_t binary_voltage)
{
	char volt_display[10];
	int first = 0;
	int second = 0;
	int third = 0;
	char hexa = 'x';
	char volts = 'V';
	char decimal = '.';
	char space = ' ';
	char zero = '0';

	first = (int)dec_voltage;
	second = (int)((dec_voltage - first) * 10);
	third = (int)((((dec_voltage - first) * 10) - second) * 10);
```

```
first = first + 0x30;
second = second + 0x30;
third = third + 0x30;

volt_display[0] = first;
volt_display[1] = decimal;
volt_display[2] = second;
volt_display[3] = third;
volt_display[4] = volts;
volt_display[5] = space;
volt_display[6] = zero;
volt_display[7] = hexa;

uint8_t first_hex = (binary_voltage >> 4) & 0x0F;
if(first_hex > 9)
{
	switch(first_hex)
	{
		case 10: volt_display[8] = 'A';
		break;
		case 11: volt_display[8] = 'B';
		break;
		case 12: volt_display[8] = 'C';
		break;
		case 13: volt_display[8] = 'D';
		break;
		case 14: volt_display[8] = 'E';
		break;
		case 15: volt_display[8] = 'F';
		break;
	}
}
else
{
	volt_display[8] = (first_hex + 0x30);
}

uint8_t second_hex = (binary_voltage & 0x0F);
if(second_hex > 9)
{
	switch(second_hex)
	{
		case 10: volt_display[9] = 'A';
		break;
		case 11: volt_display[9] = 'B';
		break;
		case 12: volt_display[9] = 'C';
		break;
		case 13: volt_display[9] = 'D';
		break;
		case 14: volt_display[9] = 'E';
		break;
		case 15: volt_display[9] = 'F';
		break;
	}
}
```

```
		else
		{
			volt_display[9] = (second_hex + 0x30);
		}

		for(int i = 10; i < strlen(volt_display); i++)
		{

			volt_display[i] = NULL;
		}

		OUT_STRING(volt_display);

}

int main(void)
{
		EBI_INIT();
		LCD_INIT();
		ADC_INIT();

		OUT_STRING("James is awesome!");

		while(1)
		{
			uint8_t bin_voltage = ADCB.CH0.RES;
			Delay_XuS(200);
			float dec_voltage = bin_to_decimal(bin_voltage);
			print_voltage(dec_voltage, bin_voltage);
			Delay_XuS(20000);
			Delay_XuS(20000);
		}
}
```

Part C

```
/*
 * lab5c.c
 *
 * Created: 7/11/2016 4:31:21 PM
 * Author : James Mak
 */

#include <avr/io.h>
#include "ebi_driver.h"
#include <string.h>

#define F_CPU 2000000
#define CS0_Start 0x8000
#define CS1_Start 0x420000
#define LCD_Start 0x422000
#define LCD_Write 0x422001

void Delay_1sec(void)
{
		volatile uint32_t count;
```

```
	for(count = 0; count < F_CPU; count++);

}

void Delay_XuS(int a) // Delay a certain amount of uS.
{
	for(int i = 0; i < a; i++);
}

void EBI_INIT(void)
{
	PORTH.DIR = 0x37;
	PORTH.OUT = 0x33;
	PORTK.DIR = 0xFF;

	EBI.CTRL = EBI_CS_MODE_SRAM_gc | EBI_IFMODE_3PORT_gc; //3 port (H,J,K) SRAM mode.

	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_16KB_gc; //SRAM mode 16k address space.

	EBI.CS1.BASEADDR = (uint16_t) (CS1_Start>>8) & 0xFFFF;
	EBI.CS1.CTRLA = EBI_CS_MODE_SRAM_gc | EBI_CS_ASPACE_64KB_gc; //SRAM mode 64k address space.
}

void ADC_INIT(void)
{
	PORTB_DIRCLR = 0x03; //Pin 4 to input.
	ADCB.CTRLA = 0x07; //Channel 0, Flush, Enable ADC.
	ADCB.CTRLB = 0xCC; //Free-running, Unsigned, 8-bit.
	ADCB.REFCTRL = 0x30; //Port B as reference.
	ADCB.PRESCALER = 0x07; //Div/32 Pre-scaler.
	ADCB.EVCTRL = 0x00; //No events, sweep channel 0.
	ADCB.CH0.CTRL = 0x81; //Positive signal input.
	ADCB.CH0.MUXCTRL = 0x28; //ADC pin 5.
}

void BF_POLL(void)
{
	uint8_t BF = __far_mem_read(LCD_Start);

	while((BF & 0x80) == 0x80) //Poll the busy flag (bit 7 until it is clear.
	{
		BF = __far_mem_read(LCD_Start);
	}
}

/*int8_t fetch_bin_voltage()
{
	while((ADCB.CH0.INTFLAGS & 0x01) == 0)
	return
}*/

float bin_to_decimal(uint8_t bin_voltage)
{
	float conv_voltage = (float)(bin_voltage * 5) / 255;
	return conv_voltage;
```

```
}

void LCD_INIT(void)
{
	__far_mem_write(LCD_Start, 0x38); //Configure function.
	BF_POLL();
	__far_mem_write(LCD_Start, 0x0F); //Configure display.
	BF_POLL();
	__far_mem_write(LCD_Start, 0x01); //Clear display.
	BF_POLL();
	__far_mem_write(LCD_Start, 0x06); //Configure entry mode.
	BF_POLL();
}

void OUT_CHAR(char character)
{
	__far_mem_write(LCD_Write, character);
	BF_POLL();
	Delay_XuS(400);
}

void OUT_STRING(char str[]) //Store the characters in strings.
{
	__far_mem_write(LCD_Start, 0x01);    // Select function clear display, clear display.
	BF_POLL();
	Delay_XuS(400);

	for(int i = 0; i < strlen(str); i++)
	{
		OUT_CHAR(str[i]);
	}
}

void print_voltage(float dec_voltage, uint8_t binary_voltage)
{
	char volt_display[10];
	int first = 0;
	int second = 0;
	int third = 0;
	char hexa = 'x';
	char volts = 'V';
	char decimal = '.';
	char space = ' ';
	char zero = '0';

	first = (int)dec_voltage;
	second = (int)((dec_voltage - first) * 10);
	third = (int)((((dec_voltage - first) * 10) - second) * 10);

	first = first + 0x30;
	second = second + 0x30;
	third = third + 0x30;

	volt_display[0] = first;
	volt_display[1] = decimal;
	volt_display[2] = second;
	volt_display[3] = third;
```

```
volt_display[4] = volts;
volt_display[5] = space;
volt_display[6] = zero;
volt_display[7] = hexa;

uint8_t first_hex = (binary_voltage >> 4) & 0x0F;
if(first_hex > 9)
{
        switch(first_hex)
        {
                case 10: volt_display[8] = 'A';
                break;
                case 11: volt_display[8] = 'B';
                break;
                case 12: volt_display[8] = 'C';
                break;
                case 13: volt_display[8] = 'D';
                break;
                case 14: volt_display[8] = 'E';
                break;
                case 15: volt_display[8] = 'F';
                break;
        }
}
else
{
        volt_display[8] = (first_hex + 0x30);
}

uint8_t second_hex = (binary_voltage & 0x0F);
if(second_hex > 9)
{
        switch(second_hex)
        {
                case 10: volt_display[9] = 'A';
                break;
                case 11: volt_display[9] = 'B';
                break;
                case 12: volt_display[9] = 'C';
                break;
                case 13: volt_display[9] = 'D';
                break;
                case 14: volt_display[9] = 'E';
                break;
                case 15: volt_display[9] = 'F';
                break;
        }
}
else
{
        volt_display[9] = (second_hex + 0x30);
}

for(int i = 10; i < strlen(volt_display); i++)
{

        volt_display[i] = NULL;
```

```
		}

		OUT_STRING(volt_display);

}

int main(void)
{
		EBI_INIT();		//All initializations.
		LCD_INIT();
		ADC_INIT();
		PORTE.DIRSET = 0xFF;
		PORTE.OUT = 0x00;

		OUT_STRING("James is awesome!");

		while(1)
		{
			PORTE.OUT = 0x00;
			uint8_t bin_voltage = ADCB.CH0.RES;
			Delay_XuS(200);
			float dec_voltage = bin_to_decimal(bin_voltage);
			print_voltage(dec_voltage, bin_voltage);
			if(bin_voltage <= 0x2D) // 0x2D is my reference for low-light.
			{

				PORTE.OUT = 0xFF;
			}
			Delay_XuS(20000);
			Delay_XuS(20000);
		}
}
```

Part E

```
/*
 * lab5e.c
 *
 * Created: 7/11/2016 4:31:21 PM
 * Author : James Mak
 */

#include <avr/io.h>
#include "ebi_driver.h"
#include <string.h>

#define F_CPU 2000000
#define CS0_Start 0x8000
#define CS1_Start 0x420000
#define LCD_Start 0x422000
#define LCD_Write 0x422001

void Delay_XuS(int a) // Delay a certain amount of uS.
{
		for(int i = 0; i < a; i++);
}
```

```
void Delay_Xsec(int b)
{

	uint32_t ticks = 1000;

	for(int i = 0; i < b; i++)
	{

		for(int j = 0; j < ticks; j++)
		{

		}

	}

}

void EBI_INIT(void)
{
	PORTH.DIR = 0x37;
	PORTH.OUT = 0x33;
	PORTK.DIR = 0xFF;

	EBI.CTRL = EBI_CS_MODE_SRAM_gc | EBI_IFMODE_3PORT_gc; //3 port (H,J,K) SRAM mode.

	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_16KB_gc; //SRAM mode 16k address space.

	EBI.CS1.BASEADDR = (uint16_t) (CS1_Start>>8) & 0xFFFF;
	EBI.CS1.CTRLA = EBI_CS_MODE_SRAM_gc | EBI_CS_ASPACE_64KB_gc; //SRAM mode 64k address space.
}

void ADC_INIT(void)
{
	PORTB_DIRCLR = 0x04; //Pin 4 to input.
	ADCB.CTRLA = 0x07; //Channel 0, Flush, Enable ADC.
	ADCB.CTRLB = 0xCC; //Free-running, Unsigned, 8-bit.
	ADCB.REFCTRL = 0x30; //Port B as reference.
	ADCB.PRESCALER = 0x07; //Div/32 Pre-scaler.
	ADCB.EVCTRL = 0x00;
	ADCB.CH0.CTRL = 0x81;
	ADCB.CH0.MUXCTRL = 0x20; //ADC pin 7.
}

void ADC_INIT_2(void)
{
	PORTB_DIRCLR = 0x03; //Pin 4 to input.
	ADCB.CTRLA = 0x07; //Channel 0, Flush, Enable ADC.
	ADCB.CTRLB = 0xCC; //Free-running, Unsigned, 8-bit.
	ADCB.REFCTRL = 0x30; //Port B as reference.
	ADCB.PRESCALER = 0x07; //Div/32 Pre-scaler.
	ADCB.EVCTRL = 0x00; //No events, sweep channel 0.
	ADCB.CH0.CTRL = 0x81; //Positive signal input.
	ADCB.CH0.MUXCTRL = 0x28; //ADC pin 5.
}
```

```
void BF_POLL(void)
{
	uint8_t BF = __far_mem_read(LCD_Start);

	while((BF & 0x80) == 0x80) //Poll the busy flag (bit 7 until it is clear.
	{
		BF = __far_mem_read(LCD_Start);
	}
}

/*int8_t fetch_bin_voltage()
{
	while((ADCB.CH0.INTFLAGS & 0x01) == 0)
	return
}*/

float bin_to_decimal(uint8_t bin_voltage)
{
	float conv_voltage = (float)(bin_voltage * 5) / 255;
	return conv_voltage;
}

void LCD_INIT(void)
{
	__far_mem_write(LCD_Start, 0x38); //Configure function.
	BF_POLL();
	__far_mem_write(LCD_Start, 0x0F); //Configure display.
	BF_POLL();
	__far_mem_write(LCD_Start, 0x01); //Clear display.
	BF_POLL();
	__far_mem_write(LCD_Start, 0x06); //Configure entry mode.
	BF_POLL();
}

void OUT_CHAR(char character)
{
	__far_mem_write(LCD_Write, character);
	BF_POLL();
	Delay_XuS(400);
}

void OUT_STRING(char str[]) //Store the characters in strings.
{
	__far_mem_write(LCD_Start, 0x01);    // Select function clear display, clear display.
	BF_POLL();
	Delay_XuS(400);

	for(int i = 0; i < strlen(str); i++)
	{
		OUT_CHAR(str[i]);

		if(i == 16)
		{
			__far_mem_write(LCD_Start, 0xC0); // Go to second line.
			BF_POLL();
			Delay_XuS(400);
		}
```

```
		}
}

void print_voltage(float dec_voltage, uint8_t binary_voltage)
{
		char volt_display[10];
		int first = 0;
		int second = 0;
		int third = 0;
		char hexa = 'x';
		char volts = 'V';
		char decimal = '.';
		char space = ' ';
		char zero = '0';



		first = (int)dec_voltage;
		second = (int)((dec_voltage - first) * 10);
		third = (int)((((dec_voltage - first) * 10) - second) * 10);

		first = first + 0x30;
		second = second + 0x30;
		third = third + 0x30;

		volt_display[0] = first;
		volt_display[1] = decimal;
		volt_display[2] = second;
		volt_display[3] = third;
		volt_display[4] = volts;
		volt_display[5] = space;
		volt_display[6] = zero;
		volt_display[7] = hexa;

		uint8_t first_hex = (binary_voltage >> 4) & 0x0F;
		if(first_hex > 9)
		{
			switch(first_hex)
			{
				case 10: volt_display[8] = 'A';
				break;
				case 11: volt_display[8] = 'B';
				break;
				case 12: volt_display[8] = 'C';
				break;
				case 13: volt_display[8] = 'D';
				break;
				case 14: volt_display[8] = 'E';
				break;
				case 15: volt_display[8] = 'F';
				break;
			}
		}
		else
		{
			volt_display[8] = (first_hex + 0x30);
		}
```

```
	uint8_t second_hex = (binary_voltage & 0x0F);
	if(second_hex > 9)
	{
		switch(second_hex)
		{
			case 10: volt_display[9] = 'A';
			break;
			case 11: volt_display[9] = 'B';
			break;
			case 12: volt_display[9] = 'C';
			break;
			case 13: volt_display[9] = 'D';
			break;
			case 14: volt_display[9] = 'E';
			break;
			case 15: volt_display[9] = 'F';
			break;
		}
	}
	else
	{
		volt_display[9] = (second_hex + 0x30);
	}

	for(int i = 10; i < strlen(volt_display); i++)
	{

		volt_display[i] = '\0';
	}

	OUT_STRING(volt_display);

}

uint8_t voltmeter(void)
{
	uint8_t bin_voltage = ADCB.CH0.RES;
	Delay_XuS(200);
	float dec_voltage = bin_to_decimal(bin_voltage);
	print_voltage(dec_voltage, bin_voltage);

	return bin_voltage;
}


void special_function(void)
{
	while(1)
	{
		char game[32];

		for(int i = 0; i < 33; i++)
		{
			game[i] = '*'; // Dots
		}
```

```
                for(int i = 33; i < strlen(game); i++)
                {

                        game[i] = '\0';
                }

                OUT_STRING(game);
                Delay_Xsec(1);

                __far_mem_write(LCD_Start, 0x0C); //Cursor and cursor blink off.
                BF_POLL();
                __far_mem_write(LCD_Start, 0x02); //Clear display.
                BF_POLL();
                Delay_XuS(2000);


                for(int i = 0; i < 33;)
                {
                        Delay_Xsec(1);;

                        game[i] = 0x3C; //Open mouth.
                        OUT_CHAR(game[i]);
                        BF_POLL();
                        __far_mem_write(LCD_Start, 0x10); // Keep cursor on he same position.
                        BF_POLL();
                        Delay_Xsec(40);
                        game[i] = 0x20; //Clear position.
                        OUT_CHAR(game[i]);
                        Delay_Xsec(40);
                        game[i] = 0x2D;//Close mouth.
                        OUT_CHAR(game[i]);
                        BF_POLL();
                        __far_mem_write(LCD_Start, 0x10);
                        BF_POLL();
                        Delay_Xsec(20);


                        if(i == 15)
                        {
                                        __far_mem_write(LCD_Start, 0xC0); // Go to second line.
                                        BF_POLL();
                                        Delay_XuS(400);
                        }

                        i++;

                }

                Delay_Xsec(20);


        }
}

void PORTF_INIT(void)
{
```

```
	PORTF.DIRSET = 0x0F;	//Configure the I/O pins of Port F.

	PORTF.PIN7CTRL = 0x18;	//Set pull-up resistor to pin 7.
	PORTF.PIN6CTRL = 0x18;	//Set pull-up resistor to pin 6.
	PORTF.PIN5CTRL = 0x18;	//Set pull-up resistor to pin 5.
	PORTF.PIN4CTRL = 0x18;		//Set pull-up resistor to pin 4.
}

char keypad_decode()
{
	switch(PORTF_IN)
	{
		case 0x77: return '1';
		break;
		case 0xB7: return '2';
		break;
		case 0xD7: return '3';
		break;
		case 0xE7: return 'A';
		break;
		case 0x7B: return '4';
		break;
		case 0xBB: return '5';
		break;
		case 0xDB: return '6';
		break;
		case 0xEB: return 'B';
		break;
		case 0x7D: return '7';
		break;
		case 0xBD: return '8';
		break;
		case 0xDD: return '9';
		break;
		case 0xED: return 'C';
		break;
		case 0x7E: return 'E';
		break;
		case 0xBE: return '0';
		break;
		case 0xDE: return 'F';
		break;
		case 0xEE: return 'D';
		break;
		default: return '\0';
		break;
	}
}

char keypad_poll()
{
	PORTF_INIT();
	char key = '\0';

	while(key == '\0')
	{
```

```
		PORTF.OUT = 0x07;
		key = keypad_decode();
		if(key != '\0')
		{
			return key;
		}
		PORTF.OUT = 0x0B;
		key = keypad_decode();
		if(key != '\0')
		{
			return key;
		}
		PORTF.OUT = 0x0D;
		key = keypad_decode();
		if(key != '\0')
		{
			return key;
		}
		PORTF.OUT = 0x0E;
		key = keypad_decode();
		if(key != '\0')
		{
			return key;
		}
	}

	return key;
}

int main(void)
{
	EBI_INIT();
	LCD_INIT();
	PORTE.DIRSET = 0xFF;
	PORTE.OUT = 0x00;

	char key_pad = keypad_poll();

	while(1)
	{
		switch(key_pad)
		{
			case '0': OUT_STRING("James is awesome!");
			break;
			case '1': OUT_STRING("James is awesome!");
			break;
			case '2': OUT_STRING("May the Schwartz be with you!");
			break;
			case '3': OUT_STRING("May the Schwartz be with you!");
			break;
			case '4':
					while(1)
					{
						ADC_INIT();
							voltmeter();
							Delay_XuS(20000);
```

```
                                        Delay_XuS(20000);
                                        Delay_XuS(20000);
                                }
                        break;
                        case '5':
                                while(1)
                                {
                                        ADC_INIT();
                                        voltmeter();
                                        Delay_XuS(20000);
                                        Delay_XuS(20000);
                                        Delay_XuS(20000);
                                }
                        break;
                        case '6':
                                __far_mem_write(LCD_Start, 0X01);
                                BF_POLL();
                                Delay_XuS(400);

                                while(1)
                                {
                                        ADC_INIT_2();
                                        uint8_t bi_voltage = voltmeter();
                                        PORTE.OUT = 0x00;

                                        if(bi_voltage <= 0x2D) // 0x2D is my reference for
low-light.
                                        {

                                                PORTE.OUT = 0xFF;
                                        }

                                        Delay_XuS(20000);
                                        Delay_XuS(20000);
                                }

                        break;
                        case '7':
                                __far_mem_write(LCD_Start, 0X01);
                                BF_POLL();
                                Delay_XuS(400);

                                while(1)
                                {
                                        ADC_INIT_2();
                                        uint8_t bi_voltage = voltmeter();
                                        PORTE.OUT = 0x00;

                                        if(bi_voltage <= 0x2D) // 0x2D is my reference for
low-light.
                                        {

                                                PORTE.OUT = 0xFF;
                                        }

                                        Delay_XuS(20000);
                                        Delay_XuS(20000);
```

```
                                        }
                        break;
                        case 'E':
                                        __far_mem_write(LCD_Start, 0X0C);
                                        BF_POLL();
                                        Delay_XuS(400);
                        break;
                        case 'F':

                                        __far_mem_write(LCD_Start, 0X01);
                                        BF_POLL();
                                        Delay_XuS(400);
                        break;
                        case '8':
                                        special_function();

                        default:
                        break;

                        }


                Delay_XuS(20000);
                Delay_XuS(20000);
                Delay_XuS(20000);

        }
}
```

Appendix: