Lab 6: LCD and A/D: Digital Voltmeter

OBJECTIVES

- Learn how to use C (as an alternative to Assembly) in your programs.
- Learn how to control and interface an LCD panel to a microprocessor.
- Learn how to use analog-to-digital conversion (ADC) system on a microcontroller.
- Use the ADC on your XMEGA to sample an analog input, convert the binary value to decimal, and display the value on an LCD. (You are creating a simple voltage meter.)

REQUIRED MATERIALS

- EEL4744 Board kit and tools
- DAD (Diligent Analog Discovery) kit
- 1 – Female Header(16-pins) for LCD mounting
- 2 – Male Header (16-pins) for LCD mounting
- 1 – LCD with 8-bit data (16-pins)
- 2– Potentiometers (pots)
- You **WILL** need the following documentation:
  - LCD Panel Notes (8-bit data)
  - Spec sheet for a Crystalfontz LCD panel
- XMEGA documents
  - doc8331: XMEGA Manual
  - doc8032: Analog to Digital (ADC)
  - doc8075: Writing C-code for XMEGA
- Lecture 13 notes for A-to-D pertaining to uTinkerer
  - uTinkerer documentation

YOU WILL NOT BE ALLOWED INTO YOUR LAB SECTION WITHOUT THE REQUIRED PRE-LAB.

PRELAB REQUIREMENTS

REMEMBER:
You must adhere to the *Lab Rules and* Policies document for **every** lab.

NOTE: All software in this lab should be written in C. If you cannot get your programs working in C, you can write it in Assembly for partial credit.

NOTE: Although the C language has a multitude of built-in functions, you are **NOT** permitted to use any of them in EEL 4744. For example, you are **NOT** allowed to use the `_delay_ms` or `_delay_us` functions.

PART A: LCD DISPLAY

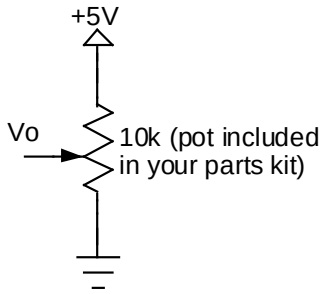
In this section you will add an LCD display to your uTinkerer PCB. The LCD module included in your kit can display 2 lines with up to 16 characters on each line (2x16), has an 8-bit data bus, and can operate in 4- and 8-bit modes. The LCD has 2 registers, command and data, that are used for issuing commands/writing (or reading) characters respectively. Refer to the *LCD Panel Notes (8-bit data)* document or device datasheet for pin-out and command information.

From a previous lab, you already configured CS1 to enable on addresses 0xA3 0000 - 0xA3 FFFF. In this lab, you will use also use additional address decoding to place an LCD at addresses 0xA3 **1**000 –0xA**3** **3**FFF, just below the address range of your SRAM (which starts at 0xA3 **4**000). You **must** use CS1 and the required address lines to access both the SRAM and the LCD. You SRAM **must** still work.

1. Consult the LCD datasheet for pin-out information and an example circuit. You will interface the LCD to the XMEGA pins as follows.

uTinkerer Pins	LCD Pins
A0	RS
CPLD Pin	E
D7:0	DB7:0
/WE	R/~W
5V pot (middle pin)	Vo

Note: You are given a 5V LCD in your kit, which means that you should connect VSS to GND and VDD to **5V** (**IMPORTANT: NOT 3.3V**!). We are currently using old inventory and future kits will eventually get a 3.3V LCD. Fortunately, you can still interface this LCD to XMEGA's 3.3V logic. Vo should be connected to a 5V potentiometer (pot) for contrast adjustment, as shown below



I suggest that you place the LCD to the right of your keypad, placed in a similar manner to what you did with the keypad. Solder a female header onto your board and a male header onto the LCD for easy removal and storage.

Note that if the LCD is placed so that the LCD overlaps the board as the keypad does, the display will show characters upside down. Make sure to place the LCD such that the side with the header is on the top so that the characters will appear right side up. If you place the LCD so that it dangles off the bottom edge of the board, the display will show characters right-side up (but be sure to remove the LCD while not in use). Wire wrap to the LCD as necessary while remember the proper pin 1.

2. Unlike the 4-bit LCD modules used in the past, the 8-bit LCD's timing characteristics are compatible XMEGA's external data bus (EBI). Aside from CS and address lines, your LCD enable signal (E) also needs to include the /WE **and** /RE signals from your processor. If you are confused about how to configure the LCD enable signal, look at the reading/writing diagrams for the processor (Appendix A in doc8331) along with the reading/writing diagrams for the LCD (datasheet).

3. RS signal is used to distinguish between LCD commands and characters sent to the LCD. **Since RS is attached to the XMEGA's A0 pin, you can use any even address to set RS to 0 and any odd address to set RS to 1. All addresses must be within LCD enable range**.

4. Your LCD runs much slower than XMEGA. As a result, **you must wait some time (see datasheet) between consecutive instructions**. An alternative (and **better** solution) is to poll the LCD's Busy Flag (BF on DB7) to avoid delays. After writing a command or character, **read the BF using LCD command address (not LCD data)** to determine whether the LCD is still processing your last transmission. When the BF becomes false (low), the LCD is ready for the next character/command. Make sure you wait at least two cycles after a write before reading so the LCD has time to turn on its busy flag. The fastest that the (E) signal can be toggled, according the LCD documents, is 1 MHZ. Our board runs at 2 MHZ which violates this constraint; so we use the two cycle delay. (One cycle should also work, but we use two cycles just to be sure.) You may insert NOP instructions between a write and read to implement the delay.

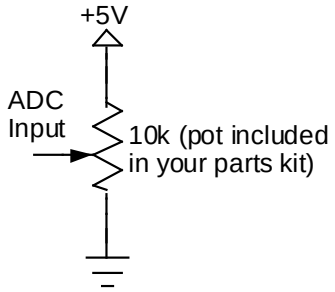
5. As a simple first test for writing to the LCD, write code to send out your name to the LCD. There is a character output function supplied to you on the examples page of our class website called `__far_mem_write(address,data)`. To access this function you need to include the `"ebi_driver.h"` header file at the top of your code. Save the above file and place it in the same folder as your program. You will then be able to use the `.include` to have access to it in the program.

   NOTE: It's **STRONGLY** suggested that you write a subroutines like OUT_CHAR and OUT_STRING (similar to the subroutines for outputs to your serial port that you wrote in homework 5). The proposed subroutines should take in as a parameter any string and send it to the LCD, making sure all LCD timing requirements are satisfied. This is almost **ESSENTIAL** to complete this lab on time. **You must initialize your LCD to 8-bit mode first before you can use the BF.**

PART B: USING AN A/D TO CREATE A MULTIMETER

In this part of the lab, you will use XMEGA's ADC to display the voltage from center tap of a second potentiometer on the LCD. Carefully read sections 28.1 - 28.6, 28.8, 28.16 - 28.17 in XMEGA doc8331 manual.

1. Connect a potentiometer (as shown below) to any of the eight open ADC channels (labeled AD0 through AD7) on your uTinkerer PCB. Note that channels AD8:AD11 on your uTinkerer PCB are not conditioned to accept 5V input. In order to reduce noise in the system, place the potentiometer as close to the analog input as possible.



2. Use the internal reference of VCC/1.6 in your ADC register initializations. This will set your ADC voltage span to be from 0V to 2.0625V. (There is voltage divider and op-amp circuitry on your board to divide the 0 to 5V potentiometer voltage to meet this constraint.)

3. There are several ways to configure the ADC system. For example, you can use 12-bit signed, right adjusted, single-ended mode with no gain, with continuous conversion on channel 0. In this case, bit 11 will be zero and you can use bits 10 to 0 (11 bits) or 10 to 3 (8 bit) for calculations. It is your choice to use signed or unsigned mode. (Do not forget to set the direction of the ADC pin you are using to input and enable the ADC module.)

   If you used 12-bit **un**signed mode, when the input is ground (0V), the digital value will be approximately 200. (See doc8032, Figure 3-2.) NOTE: When a prior version of the board was configured as unsigned 12-bit we saw an offset in the digital value, giving a value of approximately 130 (-70 from theoretical) for a 0V input when using the ADC, but 5V appeared as 4095 (as it should).

4. The ADC value will need to be converted into the decimal value voltage that it represents. You will use a algorithmic conversion (mathematical calculations) to find the decimal value from the analog voltage. The alternative is to use a lookup table (LUT), as discussed in the Appendix.

5. You must display the voltage of an ADC input pin as both a decimal number, e.g., 4.37 V, and as a hex number, e.g., 0xDE (if you use unsigned, 8-bit). For example, the LCD might display **`2.50V (0x7F)`** for unsigned, 8-bit. The hex value and its corresponding voltage will vary depending on your implementation. I suggest that you use signed mode for reasons apparent in Figure 3-2 of doc8032.

6. Determine a formula that converts the ADC value (unsigned 8-bit: 0 to 255, unsigned 12-bit: 0 to 4095, signed: 8-bit: -128 to 127, or singed 12-bit: -2048 to 2047). Note that our circuit will only allow positive input values. Essentially, you are just finding the equation of a line (with the ADC value on the horizontal access and the corresponding voltage on the vertical axis). Output 3 digits to the LCD for your decimal voltage, e.g., 3.14 V.

7. The hex values for the ASCII characters for the digits 0 through 9 are 0x30 through 0x39, i.e., just add 0x30 to the digit to find the ASCII representation of a digit. You will also need the hex values for the ASCII equivalents of the decimal point, a space, the letters "V" and "x," and both the left and right parenthesis.

8. If we assume that the input voltage calculated in part 6 was 3.14V, the below algorithm describes how to send that value to the LCD, one character at a time. Note that using the type casting operation in C is very helpful for this algorithm. Type casting converts a value of one type to a value in another type. For example, if I is an integer equal to 3 and F is a floating point number, then F = (float) 3; will result in F = 3.0. Similarly, if Pie = 3.14159265 (approximately), then I = (int) Pie, with result in I = 3.

   - Pi = 3.14159… //variable holds original value
   - Int1 = (int) Pi = 3 ➔ 3 is the first digit of Pi
   - Send this Int1 digit to the LCD, then send "."
   - Pi2 = 10*(Pi - Int1) = 1.4159…
   - Int2 = (int) Pi2 = 1 ➔ 1 is the second digit of Pi
   - Send this Int2 digit to the LCD
   - Pi3 = 10*(Pi2 – Int2) = 4.159…
   - Int3 = (int) Pi3 = 4 ➔ 4 is the third digit of Pi
   - Send this Int2 digit to the LCD
   - …

   Send a space, then a "V (" to the LCD. Then send the two or three hex digits corresponding to the ADC value to the LCD, i.e., 2 hex digits if you use 8-bit mode and 3 hex digits if you use 12-bit mode. Finally send a ")" to the LCD, resulting in something like **`3.14 V (0xA0)`**, for an unsigned, 8-bit.

9. Testing: Use your DAD and multimeter to verify that you XMEGA-based voltmeter is functioning properly. Set the potentiometer at five different positions across the entire range of voltages and record the readings from your DAD, your XMEGA-based voltmeter, and you multimeter. The values will not agree perfectly, and may be as much as (5-10%) different.

PART C: SELECTING LCD FUNCTION USING A KEYPAD

In this part of the lab you will use your keypad to select different functions for displaying on your LCD. The functions are described as follows:

Function	Keypad Keys	LCD Function
1	0,1,2	Clear LCD and blink cursor at home
2	3,4	Display the following on **2** lines: **`May the Schwartz`** **`be with you!`**
3	5,6	**Continuously** display the pot tap voltage, e.g., **2.37 V (0x79)**
4	7,8,9	Display your name on LCD
5	others	Create your own. **Be creative!** (worth +3 extra credit points added to this lab's grade)

If any key from function 1 is pressed, clear the LCD, return the cursor to home (the top left element of the LCD) and make it blink. If any key from function 2 is pressed, the string "May the Schwartz be with you!" should be displayed on the LCD. If the user presses any key from function 3, the LCD should display the voltmeter reading in the same format as described in parts 5-8 of this lab; the LCD should constantly update the voltmeter reading until a different key is pressed. If a key from function 4 is pressed, the LCD should display your name, just like in Part A of this lab. If any key from function 5 is pressed, the LCD should display anything you want. Note that the better and more original function you create for function 5, the more points you will get (up to 3% of the lab). Until a new key is pressed, the last function pressed should determine what is shown on the LCD. For instance, if a 7 is pressed, your name should be sent out on the LCD and displayed until a different function is selected on the keypad.

PRE-LAB QUESTIONS

1. 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?
2. In this lab our reference range is ideally from 0V to 5V. If the range was 0 to 2.0625V and 12-bit unsigned mode was used, what is the resolution and what is the digital value for a voltage of 0.37 V.
3. Assume you wanted a voltage reference range from 1 V to 2 V, with a signed 12-bit ADC. What are the voltages if the ADC output is 0xD42 and 0xF37?

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

PRE-LAB REQUIREMENTS

1. Answer all pre-lab questions
2. Add an LCD panel to your uTinkerer PCB.
3. Use the ADC to sample the analogue voltage and display the results on the LCD.
4. Write an interactive menu, using your keypad, LCD, and ADC. You may use your computer keyboard and your serial port in lieu of your keypad.

IN-LAB REQUIREMENTS

1. Demonstrate parts B and C.
2. If parts B and C do not work, demonstrate part A.

APPENDIX

Part B, number 4 could be accomplished using a Table Look Up (LUT) technique, described below. If there is no division available, then this is a good technique. Since there is no division instruction in assembly language, than this technique would be necessary unless a division subroutine was created. In C, division is accomplished with the appropriate library included.

4. The ADC value will need to be converted into the decimal value voltage that it represents. You could use algorithmic conversion, however your processor does not have a divide instruction, and you cannot use C-generated division routines. The alternative is to use a lookup table (LUT). An example lookup table (LUT) may look like the following at successive memory locations, where the values are in stored in ASCII:

```
0.00V (0x00), 0.02V (0x01), 0.04V (0x02),
0.06V (0x03), 0.08V (0x04), 0.10V (0x05), …
```

   Note that the actual value corresponding to 0x01 for 8-bit unsigned is 0.0196V. But it is a waste to store the ASCII for the decimal point, the parentheses, and the 0x, since those will always need to be displayed.

   A table with the above structure would simplify the lookup process. One only has to calculate the address by using the ADC's hex value to retrieve the voltage equivalent. There are multiplication assembly instructions which should be used for the lookup process:

   Addr = SizeOfText*ADC_value + TableAddr