Lab 5: EBI and ADC: Digital Voltmeter

OBJECTIVES

- Learn how to use C (as an alternative to Assembly) in your programs.
- Learn how to use an analog-to-digital conversion (ADC, also known as A/D) 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 a console (You are creating a simple voltage meter.)
- Explore and understand how the External Bus Interface works by storing ADC values into external SRAM memory.

REQUIRED MATERIALS

- uPAD 1.4 Board and Analog Backpack v1.2
- NAD/DAD (NI/Diligent Analog Discovery) kit
- You **WILL** need the following documentation:
  - Analog Backpack v1.2 documentation
  - uPAD 1.4 Memory Base v1.2
- XMEGA documents
  - doc8331: XMEGA Manual
  - doc8385
  - doc8032: Analog to Digital (ADC)
  - doc8075: Writing C-code for XMEGA
- Notes for A-to-D pertaining to uPAD
  - uPAD 1.4 documentation

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

PRELAB REQUIREMENTS

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 3744. For example, you are **NOT** allowed to use the `_delay_ms` or `_delay_us` functions. Also, do not use `sprintf`, `printf`, or any similar functions.

NOTE: Convert the 32 MHz clock configuration code that you previously used in Assembly to C.

PART A: External Bus Interface (EBI)

The EBI is crucial to the microcontroller in allowing data transfer between external peripherals and accessing external memory outside the processor. It's important to understand which ports/pins have access to the EBI, look at **doc8385 Section 33** and note which ports we will be using for this section. The XMEGA's External Bus Interface (EBI) system gives access to the XMEGA's address, data, and control busses. The external memory is part of the Data Memory section for the processor and happens right after the internal SRAM memory (0x2000-0x3FFF) ends.

1. Carefully read through the EBI section in the XMEGA AU Manual (doc8331, section 27). Make sure you understand the configuration registers, the chip select register, and how it generates the appropriate chip selects. In this lab we will configure XMEGA's CS0 (chip select 0) to enable external memory spaces at addresses that will be specified later.
2. Take a look at the Memory Base Board and you will notice you only have access to 8 bits of the address location. Figure 27-4 in doc 8331 shows the configuration assumed on your uPAD 1.4 board, i.e., the XMEAGA in SRAM 3-PORT ALE1 EBI configuration. Look at the Memory Base schematic to see the latch and SRAM devices on the Memory Base. In this configuration, the XMEGA needs the latch to hold addresses A15:A8 so that they are available (along with A7:0) when needed to access an external device, in this case an external SRAM that needs A14:0. These address pins are **time multiplexed** on one particular XMEGA port.
3. See the bottom two timing diagram in section 36.1 of doc8331, the write and read timing diagrams in the SRAM 3-PORT ALE1 EBI configuration.

In this lab you will write a C program (**`Lab5a.c`**) that configures the EBI at a memory location of $8000 to $9FFF **(How many addresses is this?)**. You must setup the EBI for SRAM 3-PORT ALE1 configuration, and properly configure the chip select and base address to accomplish the specifications. (You must also setup the direction of the specified ports pins for the various EBI signals.)

If you are trying to read or write to a specified EBI address ABOVE 0xFFFF in C, you will need dedicated software to accomplish this high-address access. There is a header file on the examples page of the class website called `ebi_driver.h`. There is an output function in this file called `__far_mem_write(address,data)` that is needed to write to addresses above 0xFFFF. There is also and input function in the same file that is needed to read form addresses above 0xFFFF called `__far_mem_read(address)`. To access these functions you will need to copy it you're your project and also include it in your program, i.e., include the following: `#include "ebi_driver.h"`.

However, since in the example specified earlier in this section, we are dealing with a 16-bit addresses, we do **NOT** need to use these "far" functions.

To access the external memory location you will need to **initialize a signed pointer to read/write values into the location.** A pointer is a variable that holds the address of another variable. To initialize a pointer you will need to use the "*" in front of the pointer variable name. For example:

```
  volatile int8_t *ptr = 0x4000;
```

Placing the volatile keyword in front of the variable tells the optimizer that the variable can change at any time.

This points to the external memory address 0x4000 in Data Memory. To read in data from memory you'll need to dereference the pointer by:

```
  (some   variable)   =   *ptr;   //contents   of
address
```

To write to memory:

```
  *ptr = (some data);
```

Once you initialize the pointer, set up an infinite loop and constantly write 0x37 into the memory location 0x8500 and 0x73 into memory location 0x8501. Use the pinouts on the bottom of the memory board, to view the signals (A3:0, D7:0, ALE, CS0, WE) using the LSA tool on your DAD/NAD. You won't have enough wires for all the signals (this is why A7:4 were left out above). Save a screenshot of the DAD/NAD LSA output, annotate the output describing the various things happening during each of the two write cycles.

It will be helpful for you to just create a function to set up EBI since you will need this setup later on for the lab.

PART B: USING AN ADC TO SENSE LIGHT

In this part of the lab, you will use XMEGA's ADC to detect light conditions using a CdS cell. CdS (Cadmium Sulfide) cells are a type of photoresistors, electronic components sensitive to incident light. You will write program **`Lab5b.c`**.

Properly install the Analog Backpack onto the uPAD. Read the Analog Backpack schematic sheet before you continue to determine the ports and pins that will be used with this device.

In this part of the lab you will be using the XMEGA's ADC (Analog to Digital converter) to detect light conditions using a CdS cell. CdS cells are a type of photo-resistors, electronic components sensitive to incident light. The CdS cell is wired (see Figure 1) in a Wheatstone bridge configuration. (You can learn how a Wheatstone bridge circuit works at https://en.wikipedia.org/wiki/Wheatstone_bridge. EEL 3111C has a lab that explores this circuit.) The output voltage is measured between CdS+ and CdS-. (See the Analog Backpack schematic for pin outputs.) When the light on the CdS cell increases, the voltage increases; as the light decreases, the voltage decreases. In a balanced Wheatstone

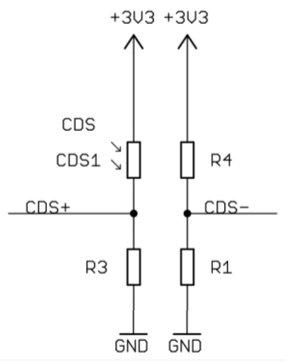

Figure 1: CdS cell configuration on Analog Backpack (on page 3 of Analog Backpack schematic).

bridge, the resistance of the CdS cell would be approximately the same as that of R4, R3, and R1; this would result in a 0 V output. Because the resistance of the CdS in complete darkness is **NOT** the same as the other resistors in the bridge, the differential voltage between the pins will be approximately -0.6 V. You can measure this with your multimeter or DAD/NAD (before creating your XMEGA program) to verify the values and also, later, to compare your XMEGA program to what you measure with these other devices.

Write a C program, **`Lab5b.c`**, that reads in the differential output of CdS+ and CdS-. Use the ADC on Port A for the pins of the CdS cell.

Cover and uncover the CdS cell and observe the changes in voltages across CDS+ and CDS-; this will give you an idea of the values to be expected with your board. To get the maximum voltage shine a light (perhaps from your cell phone) directly at the CdS cell. To obtain the minimum voltage completely cover the CdS cell.

Carefully read Section 28 of doc8331 of the XMEGA manual to learn about how to configure and start a conversion of the ADC. You will also need to look at the Analog backpack documentation to see which Port/Pins are connected to the analog pins on the XMEGA to properly configure them as input pins. Below are the basic guidelines needed to set up the ADC:

1. Use the external reference of AREFB in your ADC register initializations. This will set your ADC voltage span to be between -2.5 V and +2.5 V.
2. Configure the ADC system for **signed 8-bit (or signed 12-bit, right-adjusted), differential mode, with a gain of 1**. **Use Channel 0 for this measurement!** You must set the direction of the ADC pin that you are as an input and also enable the ADC module.

3. Test the voltage at the ADC input with your multimeter (or DAD/NAD) and then compare it with the values you get from your ADC system. Include breakpoints at the end of each analog conversion to compare the results to your multimeter/DAD/NAD reading.

   a. Find the equation ,$V_{ANALOG} = f(V_{DIGITAL})$, of a line in order to convert the digital value (between -128 and 127 for eight bits or -2048 and 2047 for 12 bits) to an analog value (–2.5 V to +2.5 V). Put this equation in your lab report. For example if the voltage is 1V, the digital representation should be about 0x33 (for 8-bit) or 0x333 (for 12-bit). If the voltage is 1.5V the digital representation should be about 0x4C (for 8-bits) or 0x4CC (for 12-bits).

   b. Now compare multimeter/DAD/NAD readings to the values calculated by using the formula determined above with the digital readings. The values may differ by as much as 10%.

PART C: CREATING A VOLTMETER

1. In this part of the lab you will take the ADC value from Part B into a voltmeter. Add the program you did in Part B to this part, call this program `Lab5c.c`. **Setup the UART for 8 data bits, a baud rate of 28,800 Hz, no parity bit, and 1 stop bit**. If you haven't already done so, translate the assembly program you did in Lab 4 to C. You will be outputting both the decimal and hexadecimal representations of the voltage from the CdS cell to your PC's screen using the Atmel Terminal.

2. You must display the voltage of the ADC input pins as both a decimal number, e.g., 4.37 V, and as a hex number, e.g., 0x6FD in signed 12-bit or 0xDF in unsigned 8-bit. You can use either signed 12-bit or signed 8-bit. An example output for a signed 8-bit ADC with a range of -5 V to 5 V is **`+4.37 V (0x6F)`**, i.e., the format for the voltage output is the decimal voltage with three digits, two to the right of the decimal point and then the hex value in either eight or 12 bits.

3. The ASCII characters of digits 0 through 9 are 0x30 through 0x39, i.e., just add 0x30 to the digit to find the ASCII representation of the 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.

4. If we assume that the input voltage calculated in part B was +3.14 V, the below algorithm describes how to send that value to the terminal, 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 Pi = 3.14159, then I = (int) Pi, with result in I = 3.

   First send the sign, either '+' or '+' out of the XMEGA's serial port, i.e., transmit it to your PC.

   The below algorithm describes how you could output the digits of a decimal number. (Remember that you are **not** allowed to use library functions like `sprint` or `printf` in this course.)

   - Pi = 3.14159…//variable holds original value
   - Int1 = (int) Pi = 3 ➔ 3 is the first digit of Pi
   - Transmit Int1 and then "."
   - Pi2 = 10*(Pi - Int1) = 1.4159…
   - Int2 = (int) Pi2 = 1 ➔ 1 is the second digit of Pi
   - Transmit the Int2 digit
   - Pi3 = 10*(Pi2 – Int2) = 4.159…
   - Int3 = (int) Pi3 = 4 ➔ 4 is the third digit of Pi
   - Transmit the Int2 digit, then a space, and then a 'V'

   Then transmit another blank, a starting parenthesis, the three hex digits (for 12-bit ADC) or two hex digits (for 8-bit ADC) corresponding to the ADC value, and then a ending parenthesis. Another example output for signed 8-bit (-5 V to 5 V) ADC is **3.14 V (0x50).** You will also need to transmit a new line character at the end of each result.

PART D: ADC Channel 1 with DAD/NAD

In this part of the lab, you will set up ADC Channel 1 to read voltages from your DAD/NAD. To start, you need to place the GND and waveform generator 1 pins to AIN- and AIN+ respectively. See Figure 2 for proper wiring. Keep the same set up as Part B. All you have to do is configure Channel 1 for ADCA. You will also need to display the hex and analog values as done in Part C. Name this file `Lab5d.c`.

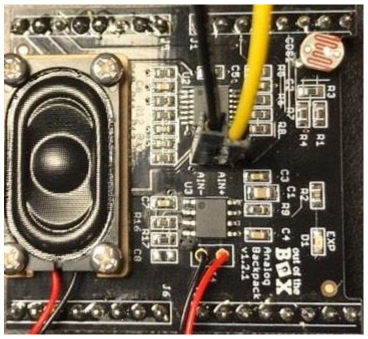

Figure 2: Connection to AIN+/AIN-.

View the Analog Backpack schematic to see where the AIN+/AIN- signal drive into. **You don't need to know how to analyze the Differential Op amps that drive those two signals.**

Using the DAD/NAD Board

1. In the Waveform's software, go to the "Wavegen" tab.
2. Switch the "Simple" menu to "Basic".
3. Click on DC now (See Figure 3).

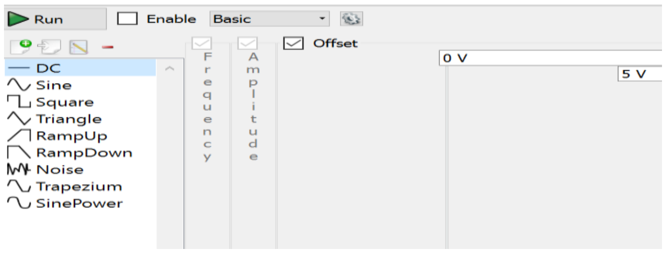

Figure 3: DAD/NAD Waveforms configuration

NOTE: To measure the voltage across soldered pins, use your multi-meter from EEL3701.

For this part, you're going to test the maximum/minimum constraints of your analog values. Move the marker up to 5 V. Measure the voltage on IN0+ and IN0-. (These are the outputs of the differential op amps you drove AIN- and AIN+ to). Now move, the marker to -5 V and measure the same pins.

Now that you have an idea of your analog value constraints, configure channel 1 to read in the values and print **both the hex and analog values** onto the terminal. You'll be scrolling through all voltage ranges from -5 to +5 to see how the output voltage changes by viewing the Hex/Analog values on the Atmel Terminal.

PART E: Toggle Menu
In this part of the lab, you will be integrating multiple peripherals together using interrupts, Timer Counters, and the UART system. Make sure, you're able to use channel 1 and 0 of the ADC system for both the DAD/NAD input and CdS cell. You'll be creating a menu that displays onto the terminal. When the program runs you need to display the following below onto the terminal:

Function	Keyboard Keys	Functionality
1	'a' or 'A'	Start conversion on CdS Cell and displays on Terminal
2	'b' or 'B'	Start conversion on DAD/NAD channel and displays on Terminal
3	'c' or 'C'	Start Timer Counter, after every 1 s the CdS starts a conversion (Interrupt Driven)
4	'd' or 'D'	Turn off Timer Counter
5	'e' or 'E'	Start CdS cell conversion, write this value into external SRAM location $8000.
6	'f' or 'F'	Read the external memory Location at $8000 and prints the value onto the Terminal

You will also need to set up the ADC compare interrupt for both channel 0 and 1. For channel 0, set up the interrupt to trigger for below threshold. For channel 1, set up the interrupt to trigger for above threshold. You will also need to ADCx_CMP register to 0 as well for this to work. Anytime you cover your hand fully on the CdS cell, the voltage should be negative and your program should go to the ISR. If you move the voltage marker on your DAD/NAD board to negative voltages, the value should be above 0, and trigger the ISR for channel 1. In the ISR you should do the following:

Channel 0 ISR: Turn on the RED RGB led; others should be off

Channel 1 ISR: Turn on the BLUE RGB led; others should be off.

NOTE: ALL interrupts should be HIGH LEVEL. To enable interrupts in C, use the function `sei()`. Also, you must use the below line in your program:

```
#include <avr/interrupt.h>.
```

- When you start the program, the terminal starts with a menu displaying all the options you can perform
- When you press 'a' on your keyboard, the terminal should display the current CdS voltage/hex values
- When you press 'b' on your keyboard, the terminal should display the current DAD/NAD voltage/hex values
- When you press 'c', this should turn on the timer counter. The easiest way to think about this, have the timer counter already initialized, besides the register that actually turns it on. Also, when the timer counter is on, there should be a continuing ADC conversion on the CdS cell every 1 s displaying on the terminal. Your program should also continue polling for any of the keys to be pressed as well.
- When you press 'd' the timer counter should be turned off and the CdS cell conversion should be halted.
- When you press 'e', you should start a conversion on the CdS cell and write this value into the memory location specified. (**Use memory window to see what happens**)
- When you press 'f', you should read in the value you last wrote to in the memory location and display it on the terminal.

NOTE: For the first time you do the conversion the result could be 0 until the next conversion, this is okay.

Name the file **`Lab5e.c`**, it will be helpful to create functions to reuse for later purposes.

PRE-LAB QUESTIONS

1. What is the difference in conversion ranges between 8-bit unsigned and signed conversion modes? List both ranges.
2. Assume you wanted a voltage reference range from -2 V to 3 V, with an unsigned 12-bit ADC. What are the voltages if the ADC output is 0xA92 and 0x976?
3. Write the address decode equation to put the input and output ports at addresses 0x2000 – 0xAFFF.

PRE-LAB REQUIREMENTS

1. Answer the pre-lab questions.
2. Setup the EBI for SRAM at memory locations $8000-$9FFF.
3. Configure the ADC properly for Channel 0 and 1 (CdS cell and DAD/NAD board).
4. Display proper voltages (in decimal and hex) on the terminal.
5. Write an interactive program that uses the serial communications, timer/counter, and ADC as described.

IN-LAB REQUIREMENTS

1. Demonstrate Part E.
2. If Part E does not work, demonstrate as much as you can from Parts A, B, C, and D.