OBJECTIVES

- Develop a basic understanding of converting signals from the analog continuous-time domain to the digital discrete-time domain using the ADC system within the *ATxmega128A1U*.
- Use and understand the purpose of the event system within the *ATxmega128A1U*
- Create a basic oscilloscope program by using the ADC and event systems within the XMEGA, as well as the *SerialPlot* program

INTRODUCTION

As you have seen throughout the previous labs, your microcontroller can produce and detect discrete binary values, i.e., '0' corresponding to a voltage value close to your processor's ground reference (normally 0 V), and '1' corresponding to a voltage value close to your processor's V_{CC} reference (e.g., 3.3 V). However, the world is also filled with analog signals, i.e., signals that have more than two discrete values, such as those that represent velocity, temperature, sound and light intensities, etc. Thankfully, it is possible for your microcontroller to both interpret and generate these non-digital signals, through two separate systems: An **A**nalog-to-**D**igital **C**onverter (**ADC**), and a **D**igital-to-**A**nalog **C**onverter (**DAC**).

NOTE: Many sensors output analog signals; an ADC is often used to convert their outputs to a digital representation. Some sensors perform this conversion themselves.

With an ADC system, an analog signal can be sampled and converted to a digital value. This digital value, whose limits are determined by the hardware and software configurations of the ADC system, can then be interpreted by your microcontroller.

LAB STRUCTURE

In this lab, we will leverage the ADC system within the XMEGA to ultimately create a program that functions like a simple oscilloscope. An analog input will be displayed on your computer screen via *SerialPlot*.

REQUIRED MATERIALS

- Atmel *ATxmega128A1U* AU Manual (doc8331)
- Atmel *ATxmega128A1U* Manual (doc8385)
- *OOTB µPAD*, with USB A/B cable
- *OOTB Analog Backpack*, with accompanying schematic
- **D**igital **A**nalog **D**iscovery (**DAD**) kit, with *WaveForms*

SUPPLEMENTAL MATERIALS

- AVR1300: Using the AVR XMEGA ADC
- AVR1001: Getting Started with the Event System
- *SerialPlot* source website
  - *SerialPlot* for 64- and 32-bit Windows

PRE-LAB PROCEDURE

REMINDER OF LAB POLICY

As required, you must re-read the *Lab Rules and Policies* before submitting any pre-lab assignment and before attending any lab.

1. USING THE ADC SYSTEM

In this section, you will write a program, **`lab7_1.c`**, to sample data from the CdS cell located on the *OOTB Analog Backpack*.

NOTE: A Cadmium-Sulfide (*CdS*) photocell (also known as a photoresistor, or a light-dependent resistor [*LDR*]) is a resistor whose resistance is variable to the amount of light present on the surface of the photo-conductive cell (see Figure 1). The CdS cell located on your *OOTB Analog Backpack* is labeled *CDS1*.

Figure 1: Two CdS photocells

However, before interfacing with the CdS photoresistor cell, you must familiarize yourself with the ADC system within the *ATxmega128A1U*.

1.1. Read the *AVR1300* Application Note describing the ADC system within XMEGA microcontrollers. Re-enforce the concepts from this document by reading § 28 (*ADC – Analog-to-Digital Converter*) of the doc8331 manual.

PRE-LAB EXERCISES

i. Why must we use the ADCA module as opposed to the ADCB module?

As alluded to above, an ADC system within your microcontroller will be used to sample analog voltage values generated by the CdS cell located on the *OOTB Analog Backpack*. Below, you will write software to interface with the CdS cell, although it will first be necessary to identify which ADC system must be used within the XMEGA.

1.2. Determine which ADC module must be used to interface with the CdS cell located on the *OOTB Analog Backpack*. Refer to the *OOTB Analog Backpack Schematic*.

1.3. Write a "C" function, ***void adc_init(void)***, to initialize the ADCA module as follows:

- 12-bit signed, right-adjusted
- Normal, i.e., not *freerun* mode
- 2.5 V voltage reference

Only enable the module after all ADC initializations have been made, and do *not* start a conversion within the initialization function.

In the *MUXCTRL* register, select the appropriate combination of positive and negative inputs to measure the voltage at the CdS cell. The CDS+ and CDS- signals on the *OOTB Analog Backpack* schematic should be used.

PRE-LAB EXERCISES

ii. Would it be possible to use any other ADC configurations such as single-ended, differential, differential with gain, etc. with the current pinout and connections of the *OOTB Analog Backpack*? Why or why not?

iii. What would the main benefit be for using an ADC system with 12-bit resolution, rather than an ADC system with 8-bit resolution? Would there be any reason to use 8-bit resolution instead of 12-bit resolution? If so, explain.

1.4. Write a main routine for your program that *continually* does the following:

i. Start an ADC conversion on the proper ADC channel.

ii. Wait for the proper ADC interrupt flag to be set, indicating that the conversion has finished.

iii. Store the 12-bit signed conversion result into a signed 16-bit variable.

1.5. Verify that some ADC conversion results are accurate. This can easily be achieved by placing a breakpoint after you save the conversion result and viewing its contents in the *Watch* window.

To check if the result is correct, find an appropriate equation of a line that represents the linear relationship between the range of measurable analog voltage values and the range of digital values of the ADC system (e.g., $\mathbf{V_{ANALOG} = f(V_{DIGITAL})}$). For example, with an analog voltage range of –2.5 V to +2.5 V, if the analog voltage is 1V, the digital representation should be about 51_{10} = 0x33 for an 8-bit ADC system, and 819_{10} = 0x333 for a 12-bit ADC system; if the voltage is 1.5V, the digital representation should be about 77_{10} = 0x4D for an 8-bit ADC system, and 1228_{10} = 0x4CC for a 12-bit ADC system. You can measure the voltage difference across the CDS+ and CDS- pins using your multimeter and then compare that to the voltage value you calculate from your ADC result.

2. SAMPLING AT A SPECIFIC RATE USING EVENTS

In this section, you will write a program, **`lab7_2.c`**, that will sample the CdS cell six times per second (i.e., 6 Hz). A timer will be used to *automatically* trigger ADC conversions at a frequency of 6 Hz. For this to happen automatically, you *must* use the Event System.

2.1. Read § 6 (*Event System*) of the doc8331 manual as well as the *AVR1001* application note, which is provided in the *Supplemental Materials* section of this document.

When reading about events, you do not need to pay attention to any information regarding the quadrature decoder. It is not relevant for our course.

2.2. Write a "C" function, ***void tcc0_init(void)***, to initialize the TCC0 timer/counter module to overflow six times a second.

2.2.1 Use any valid prescaler/period combination to achieve an overflow time **of one-sixth** of a second.

2.2.2 Use the TCC0 overflow to trigger an event on *Event Channel 0*. This can be done using the *CH0MUX* register within the Event System.

2.3. Make the following additions to the ***adc_init*** function you wrote in § 0:

- Enable an ADC interrupt to be triggered when a conversion is complete.
- Using the *EVCTRL* register within the ADC module, make an ADC conversion start when *Event Channel 0* is triggered.

2.4. Write the ADC interrupt service routine that is executed when a conversion is complete. It should do the following:

i. Save the result into a signed 16-bit integer variable, just like you did in § (1.4)iii.

ii. Toggle the *RED_PWM* LED located on the *µPAD*.

NOTE: The rate at which the LED toggles should be identical to your sampling rate, i.e., 6 Hz.

3. OUTPUTTING SAMPLED DATA WITH UART

In this section, you will test the functionality of your current system by outputting the *analog voltage value* measured on the CdS cell every second, and then display the results, in terms of both decimal and hexadecimal, within a serial terminal program on your computer (e.g., PuTTY). More specifically, *the output displayed within your terminal program must include at least the following to describe the voltage measured:*

(+/-)voltage_decimal V (0xvoltage_hex)

where ***voltage_decimal*** is the measured ADC digital value corresponding to the voltage drop experienced by the CdS photocell, in terms of a decimal value with two decimal places, and where ***voltage_hex*** is the measured ADC digital value corresponding to the voltage drop experienced by the CdS photocell, in terms of a hexadecimal value with three digits.

For example, if an 8-bit ADC system with a voltage range of -5 V to +5 V was used, and the measured voltage was to correspond to a decimal value of 1.37 V, your output should include "**`+1.37 V (0x22)`**", without the quotes.

Moreover, if for the same system, a measured voltage value was to correspond to a decimal value of -2.52 V, your output should include "**`-2.52 V (0xC0)`**", also without the quotes.

PRE-LAB EXERCISES

iv. What is the decimal voltage value that is equivalent to a 12-bit signed result of 0x360, given a voltage range of -5V to +5V?

v. Given an 8-bit signed ADC system with a voltage reference range of -1V to +2V, express the expected digital value in terms of the analog input voltage, using the form $V_D = f(V_A)$.

3.1. Using everything from the previous two sections, create a voltmeter program, **`lab7_3.c`**, that outputs the voltage at the CdS cell to a serial terminal program every second.

3.2. In your ADC ISR, update a global variable with the new ADC conversion result, and set a global flag that indicates a new conversion has been made.

3.3. Create a "C" function, ***void usartd0_init(void)***, that initializes the USARTD0 module to operate at 116,500 bps, with eight data bits, odd parity, and one stop bit.

3.4. In your main routine, when the flag gets set, do the following:

3.4.1 Clear the global flag.

3.4.2 Output the voltage to the serial terminal.

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

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

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

Transmitted Result: **`3.14 V`**

Make sure that after every voltage transmission, you send a carriage return and line feed characters such that the next voltage is displayed on a new line.

NOTE: Although previously in this course, we could avoid floating point numbers, this would make it much more difficult to solve the given problems in this section and the subsequent sections of the lab. Therefore, you should use float in these sections of the lab.

4. VISUALIZING THE ADC CONVERSIONS

In this section, you will write a program, **`lab7_4.c`**, to visually display the CdS cell's voltage using *SerialPlot*, like you did in Lab 6 with the IMU's accelerometer. This will behave like a very basic oscilloscope.

4.1. Modify your ***tcc0_init*** function to make the timer overflow at 137 Hz instead of 6 Hz. You may need to make changes to your prescaler.

4.2. In your main routine, instead of outputting the decimal voltage value as done in § 3, you will output the raw 16-bit signed values via UART using *Simple Binary* data format that *SerialPlot* understands. Now, since you are only outputting one type of data (ADC channel conversion), you only need to use one channel in *SerialPlot*. This means that you only need to output two bytes via UART every time you get a new conversion result. Don't forget to update your *SerialPlot* configurations accordingly. Leave the variable type the same (*int16*) and change the number of channels to one.

You should be able to see the waveform in a the *SerialPlot* window, like when it was used with the accelerometer.

5. SWITCHING BETWEEN MULTIPLE INPUTS

Finally, you will write a program, **`lab7_5.c`**, to use UART to switch between two different analog inputs: the CdS cell and the analog input jumper labeled *J3* on the *OOTB Analog Backpack*. Note that the circuit on the last page of the *OOTB Analog Backpack* schematic that will be used in this section has a **gain of 0.4** (as stated in the text on the top of that page).

5.1. Modify your previous ***usartd0_init*** function to enable interrupts for the UART receiver.

NOTE: To send data from *SerialPlot* to your microcontroller, you will use the *Commands* tab within SerialPlot. Enter the character you wish to send in the field next to "Command 1" and then click *Send*. You can create and save multiple unique commands if desired.

When the character 'L' (for **L**ight) is received via your serial terminal connection, the program should switch to *continually* measuring and outputting the CdS cell data to *SerialPlot* at a rate of 137 Hz, where an error of up to 2% is permitted for the generated frequency.

When the character 'F' (for **F**unction generator) is received, the program should switch to measuring the result of the analog input jumper *J3*, located on the *OOTB Analog Backpack*.

If any other characters are received, don't change anything.

The decision to switch input sources should occur only within the receiver interrupt service routine.

NOTE: You will most likely be using the function generator of your DAD board to supply the ADC with an analog signal via the *J3* header. You should, however, be able to visualize waveforms from other entities, such as another circuit, a DAC (digital-to-analog converter), or some other source.

Be aware that this will not work over all input frequencies. You may want to explore the correlation between the frequency of an input signal and the affect it has on your program's output.

PRE-LAB PROCEDURE SUMMARY

1. Write lab7_1.c to ensure your ADC initializations are correct.
2. Write lab7_2.c to introduce the concept of sampling at specific intervals.
3. Write lab7_3.c to create a voltmeter-like application with UART.
4. Write lab7_4.c to create a single-input basic oscilloscope.
5. Write lab7_5.c to expand on the topics from § 4.

APPENDIX

A. TROUBLESHOOTING SERIAL PLOT

The goal of this section is to help you with some of the issues students typically have with the serial plot software.

Problem #1 (Flipped Low & High Bytes)

- Many students run into an issue where their data seems to fluctuate rapidly through a large range of values. This typically happens when the data from your *µPAD* is flipped. *SerialPlot* is taking in your low byte as a high byte and your high byte as the low byte. The simplest way to fix this is to restart you program and ensure that Serial Plot is running before you start the program on your *µPAD*.

Problem #2 (Baud Rate)

- If you are getting junk data from your *µPAD* on *SerialPlot*, this usually indicates that the Baud Rate is incorrectly input into *SerialPlot*. Check both your code and *SerialPlot* values to ensure that they match.

Problem #3 (PuTTY)

- If you are getting no data from your *µPAD*, ensure that you do not have PuTTY running. Only one program can have control over a serial connection of your *µPAD*, and PuTTY and *SerialPlot* don't get along, so only use one at a time.