EEL 3744 – Spring 2018
Revision **0**

Dr. Eric M. Schwartz

9-Apr-18

LAB 7: DMA OSCILLOSCOPE

OBJECTIVES

- Understand how to free up CPU utilization by using events and the DMA system.
- Learn how to make your own oscilloscope using the ADC system and Atmel Studio's Data Visualizer.

REQUIRED MATERIALS

- DAD/NAD Analog Discovery board
- µPAD Development Board
- Documentation:
  - XMEGA AU Manual (doc8331)
  - ATxmega128A1U Manual (doc8385)
  - Atmel 8046D - AVR1304 (DMA Controller)

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. All code must be written in C. your clock speed must be 32 MHz, and **for this lab you MUST use group configurations for all register initializations**!!

GETTING STARTED

ALWAYS create a flowchart or pseudo-code of the desired algorithm **BEFORE** writing any code. Remember to include all flowcharts/pseudocode in your lab document.

If a program/design does not work, utilize the debugging capabilities of your Atmel Debugger (i.e., emulator) and your DAD/NAD board, along with your prior electrical and computer engineering knowledge to fix any errors in your hardware and/or software (code). This should occur **BEFORE** you come to lab. If necessary, visit a TA or Dr. Schwartz, but come to lab prepared!

As you have been doing all semester long (and hopefully will continue to do in your engineering career) you should slowly build up your design, adding a new piece and testing that new piece before adding anything more. I have recommended you do just that through all the labs this semester, and hope you do this as well for this lab.

Note: The amount of information given in this lab is much less than in previous labs. Do **NOT** procrastinate!

OVERVIEW

In this lab, you will use the ADC system to sample an analog waveform and display it to a Data Visualizer Graph, like the way a real oscilloscope behaves.

The **D**irect **M**emory **A**ccess **C**ontroller (DMAC) and the Event System will be used to reduce the workload of the CPU.

PART A: CONFIGURING THE ADC

In Lab 6, you used the ADC system to create a voltmeter, like one that would be found on a multimeter. In this lab, you will be doing something very similar except instead of outputting the static values to the terminal, you will be outputting sampled ADC data to a *Data Visualizer Graph*. Remember that the *Graph* can be fed a stream of serial data. For this lab, we will send a stream of data directly over UART using USARTD0 (instead of using the data streamer protocol).

Since you aren't using the data streamer protocol, there isn't an easy way to specify the format of the data.

The *Graph* will interpret the data as unsigned, and you, or rather the DMA, will send individual bytes of ADC data via the UART. Create a function, ***void init_adc()***, that configures **ADCA CH0** in the following way:

- **8-bit** results
- Use **AREFB** (2.5V) as the reference
- Do **NOT** use freerun mode
- Use an ADC prescaler that will generate at MOST a **2 MSPS** (mega samples per second) ADC clock speed. This is required, because the XMEGA's ADC is not able to sample accurately above this rate.
- As mentioned above, the data must be **unsigned**.
- Because we are not using the same sampling method as the previous lab, you will NOT be using the Analog Backpack header (J3) to input the voltage waveform. Instead, you may use **any pin on Port A that is available**. To access Port A, you must have no backpack on the µPAD.

It would be a good idea to test this new configuration with your code from the previous lab to make sure it works before continuing.

PART B: CONFIGURING THE TIMER AND EVENTS

In this part, you will configure a timer (**TCC0**) to overflow at a rate of **2 kHz**. This timer will be configured the same way you have been doing all semester, with one exception. Now you will use the **Event System** instead of interrupts.

Events in the Event System are like interrupts in the sense that they occur when something happens in one of the XMEGA's peripherals, i.e., in the context of this lab, when a timer overflows, or when an ADC conversion completes. Events are different, however, because when they occur, no code needs to be executed such as an ISR when an interrupt happens. This means that the CPU is not needed at all in the case of an event. Read *sections 6.2 and 6.3* in the doc8331 manual for a very concise overview of the event system.

There are 8 event channels on your XMEGA, and each one can be configured in a unique way. For this lab, you will have **Event Channel 0** occur when your timer (TCC0) overflows. **Event Channel 1** will occur when a conversion

LAB 7: DMA OSCILLOSCOPE

on ADCA CH0 completes. Event CH1 will be used later in the lab to trigger DMA.

There are very few registers in the Event System, and you will only need to configure the CHnMUX register (one for each channel). This register determines what on the XMEGA should trigger each event channel. Read *section 6.8.1* in the doc8331 manual for all the possible trigger sources for the event channels.

Create a function, ***void init_tcc0()***, that initializes TCC0 to overflow at a rate of **2 kHz**. Do NOT enable interrupts for this timer. In this function, you must also select **TCC0_OVF** as the event source for Event CH0. Group configurations are very convenient when configuring the **CHnMUX** register! Don't forget to use group configurations when configuring registers.

Now, every time TCC0 overflows, an event is generated on Event CH0. What do we want to happen every time the timer overflow? **We want to sample the ADC!** The beautiful thing about events is that you can trigger one peripheral to do something when an event is generated by another peripheral. All of this can happen, and the CPU is doing **NOTHING**, not even processing an interrupt.

To make Event CH0 trigger a conversion on ADCA CH0, you will need to add something to your ***init_adc()*** function. An ADC register you may have not used in the last lab, **EVCTRL**, is used to tell the ADC system which channel(s) to start a conversion on when certain event channels are generated. This register is described in *section 28.16.4* of the doc8331 manual. Configure this register such that a conversion is started on ADCA CH0 when Event CH0 occurs, and make sure it is **ONLY** when Event CH0 occurs.

At this point, if you have configured everything correctly, a conversion on ADCA CH0 is started at a rate of 2 kHz, all without any CPU intervention!

PART C: UART

To be able to eventually get the ADC data to the Data Visualizer Graph, you must use a UART, as you did in the accelerometer lab.

Write a function (or use one from a previous lab) that initializes USARTD0 with a baud rate of 115,200 bps, one start, one stop bit, and no parity. As you will see in the next part, you will not need to use functions such as ***out_char*** or ***out_string***.

PART D: DMA WITH EVENTS

The DMA system allows data to be moved from one peripheral to another via the data bus, without processor intervention. You will use a timer to accurately "trigger" the DMA system to move data from our look-up table (data memory) to the DAC system, as to avoid CPU utilization.

When configuring the DMA, there are a few things that need to be considered.

- How many bytes will be transferred in a burst (one piece of data)?
- How many bytes are in the complete transfer (maybe there is more than one burst)? Is it just a single burst?
- How many times do you want to move this data? Do you want to move it forever?
- From where should the DMA get the necessary data? How many bytes will be transferred? If more than one byte is to be transferred, after the first byte is retrieved, where should the DMA look next?
- Where should the DMA put the data? Again, if there is more than one byte to be sent to the destination, where should the DMA put each additional byte after the first?
- Regarding the source and destination of the DMA transfer, does the DMA need to point back to its original memory locations after a transfer?
- What will trigger the DMA transfer? A completed ADC transfer? An interrupt? An event?

As you can see, there are many settings to consider when configuring DMA. Be careful to note that when configuring the DMAC, there are several registers that can **only** be configured when the DMA is disabled.

Re-read Section 5 of the 8331 manual and the Atmel 8046D - AVR1304 (DMA Controller Application Note) document. Write a function, ***void init_dma()***, to initialize DMA CH0 to transfer data from the ADCA CH0 result register to the USARTD0 data register when an ADC conversion is complete. Here are some tips:

- To determine when an ADC conversion is complete, you will use Event CH1, as mentioned in Part B of this lab.
- You must configure the **CHnMUX** register for Event CH1 in your ***init_adc()*** function to generate an event on CH1 when a conversion completes on **ADCA_CH0**. Again, group configurations are very convenient when configuring this register!
- You must select Event CH1 as the trigger source for DMA, so that every time a new ADC conversion is completed, it is sent via UART to the Data Visualizer.

Once everything is all set up, the following should be happening:

1. TCC0 overflows, which generates the CH0 event.
2. The CH0 event triggers a conversion on ADCA CH0.
3. When the ADC conversion is completed, an event on CH1 is generated.
4. The CH1 event triggers a DMA transaction from the ADC result register to the USART data register.

In the next part, you will set up the Data Visualizer Graph to plot the data you are sampling with the ADC. For now, you can generate a sinusoid from your DAD board and connect it to the ADC pin you chose on Port A. If everything is working, you should be able to see values in PuTTY that correspond to a sinusoidal input.

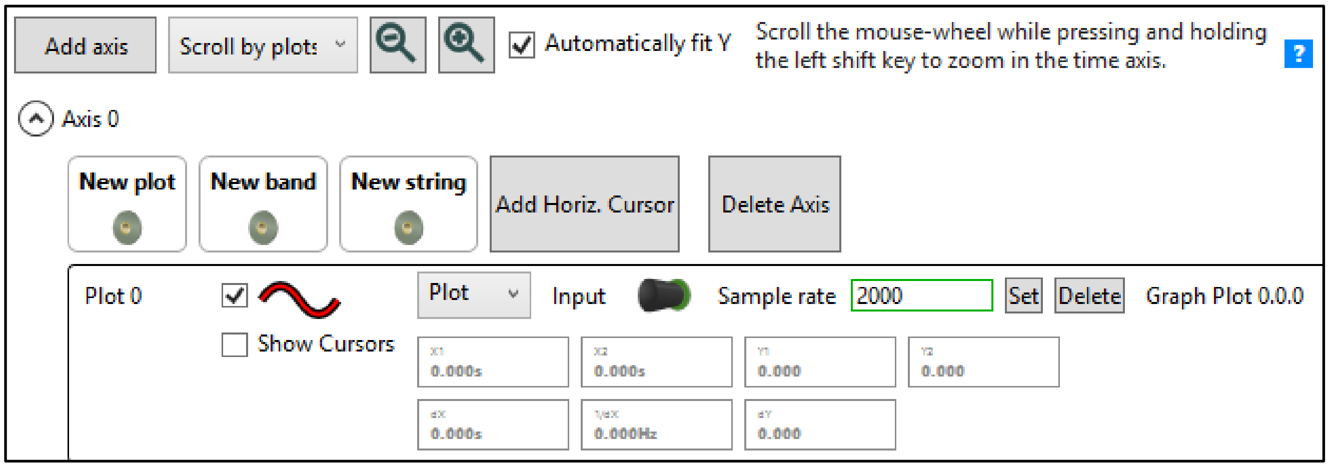
LAB 7: DMA OSCILLOSCOPE

PART E: OSCILLOSCOPE

In this part, you will configure the *Data Visualizer Graph* to view the sampled ADC data in real time.

While debugging, open the *Data Visualizer* as you have done in previous labs, and **open a Serial Port and a Graph**. Then, in the *Serial Port* menu, select your COM port, baud rate, parity, stop bits, and click "Connect". The main difference for this lab is that there is no data streamer protocol to worry about.

Next, connect your *Serial Port* directly to a "new plot" in the *Graph* window. Make sure you have the same settings as shown in Figure 1 below.



☑ Automatically fit Y

Scroll the mouse-wheel while pressing and holding the left shift key to zoom in the time axis.

Plot 0 ☑

☐ Show Cursors

Plot

Sample rate 2000

Graph Plot 0.0.0

X1 0.000s | X2 0.000s | Y1 0.000 | Y2 0.000

dX 0.000s | 1/dX 0.000Hz | dY 0.000

Figure 1: Graph Configuration

Test your program by inputting multiple waveforms using your DAD/NAD board. Test multiple frequencies.

PRE-LAB QUESTIONS:

1. How would you complete this lab without using the Event System? What is the major drawback of doing it without Events?
2. What is the range of digital and analog values you should be able to read with your ADC the way it is configured in this lab?
3. How many DMA channels are available on the XMEGA?
4. How many different *base value* trigger options are available within the XMEGA's DMA system?

IN-LAB REQUIREMENTS

1. Demonstrate Part E.