LAB 3: CLOCK CONFIGURATION, TIMERS/COUNTERS, EBI

OBJECTIVES

- Learn how to adjust the XMEGA's clock (CLK) system.
- Learn how a timer/counter operates and how to utilize the XMEGA's timer/counter (TC) systems for various simple applications.
- Explore and understand the implementation of memory-mapped I/O. Interface with an external SRAM.
- Utilize the CLK, EBI, and TC systems to read and store data to an external memory location at certain time intervals.

REQUIRED MATERIALS

- Documentation:
  - µPAD v2.0 Schematic
  - µPAD Memory Base v1.4 Schematic
  - Atmel AVR XMEGA AU Manual (doc8331)
  - Atmel AVR XMEGA A1U Manual (doc8385)
  - AVR1306: Using the XMEGA Timer/Counter (doc8045)
- DAD/NAD board
- µPAD v2.0 Development Board
- Switch & LED Backpack

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

INTRODUCTION

In this lab, you will first explore the XMEGA's clock (CLK) system, and gain the ability to change its operating frequency. Next, you will learn the basics of the timer/counter (TC) system within the XMEGA, as well as explore the External Bus Interface (EBI) system. Finally, you will use your new knowledge about the XMEGA CLK, TC, and EBI systems to write a program to periodically store data to external memory (SRAM) that is read from the DIP switches on the Switch & LED Backpack.

PRE-LAB REQUIREMENTS

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

Note: This lab is time-intensive. Do **NOT** wait to start.

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; submit this **at least 72 hours** before the start of your lab.

PART A – CLOCK CONFIGURATION

Timing is critical in most embedded systems, no matter how simple or complex they may be. The clock is the basis of all synchronous operations in the XMEGA. In the previous lab, you wrote a crude delay routine. What would happen if you tried to run the same code with a different clock speed? It may still delay, but it would no longer be accurate.

In this part of the lab, you will write an assembly program (`lab3a.asm`) with a subroutine that adjusts the frequency of the XMEGA's clock. You will also use the oscilloscope of your DAD/NAD board to measure the clock's frequency.

1. Read the System Clock and Clock Options section (**Section 7**) in the 8331 manual. Pay close attention to the register descriptions in Sections 7.9 and 7.10. The registers you will need to use to change the clock speed are OSC_CTRL, OSC_STATUS, CPU_CCP, CLK_CTRL, and CLK_PSCTRL.

The OSC_CTRL register (section 7.10.1) enables the oscillators. You must enable the new oscillator before you can select it as the new clock source. After the desired oscillator is enabled, you must give it time to stabilize. The OSC_STATUS register (section 7.10.2) contains flags that are **set** only when the oscillator is **stable** and ready for use.

Before selecting the enabled oscillator as the new clock source, you must write the "IOREG" signature to the CPU_CCP register (section 3.14.1). CCP is short for "Configuration Change Protection". This step is necessary to protect important registers while the clock source is being switched.

The next step is to select the new source for the system clock. To do this, you must use the CLK_CTRL register (7.9.1).

After **enabling** the oscillator, you must do the following:

1. Wait for the right flag to be set in the OSC_STATUS register.
   - Use of the `sbrs/sbrc` instructions when polling bits on the XMEGA is recommended.
2. Write the "IOREG" signature to the CPU_CCP register.
3. Select the new clock source in the CLK_CTRL register.

After the new system clock source has been switched, you have the option to further divide it using the CLK_PSCTRL (7.9.2) register. See **Figure 7-5** in the 8331 manual for a visual representation of what is happening.

To successfully make changes to the CLK_PSCTRL register, you must **first** write the "IOREG" signature to the CPU_CCP register the same way you do to make changes to the CLK_CTRL register. To set prescalers for the clock, you must do the following:

LAB 3: CLOCK CONFIGURATION, TIMERS/COUNTERS, EBI

1. Write the "IOREG" signature to the CPU_CCP register.
2. Write desired settings to the CLK_PSCTRL register.

To analyze the XMEGA's clock, you need to make the clock signal output to an I/O pin. There are several ways to do this, but the most straightforward technique is to use the CLKEVOUT register (**Section 13.14.4 in the 8331 manual**). This is the method that you **must** use for this lab.

1. Remove any backpacks from the µPAD. (Leave the memory base attached.)
2. Create an assembly program (**`lab3a.asm`**). Within this program, **create a subroutine** that configures the system clock for 32 MHz, initializing everything described above. The subroutine should also allow the programmer to easily scale the system clock with a prescaler, if desired.
3. Within the MAIN routine, use this subroutine to configure the clock speed for **4 MHz.** Then, configure the CLKEVOUT register to **output** the system clock **(4 MHz)** to **PORTC pin 7 (**see Section 13.14.4 in the 8331 manual**)**. Don't forget to configure PORTC pin 7 as an output.

The clock should now be outputting from PORTC pin 7. We measure the clock frequency with the DAD/NAD's *Scope* feature in *WaveForms*. Again, make sure there are no backpacks connected to the µPAD (although you can leave the memory base attached). You can access PORTC pin 7 via a female header on the top of the board.

4. Use the *Scope* feature in *WaveForms* to measure the clock signal, and display the waveform frequency. Within the *Scope* window, display the frequency by selecting View > Measurements, then Add > Defined Measurement > Horizontal > Frequency. To yield an accurate frequency measurement, set your time base to an appropriate value, like 20ns/div as shown in **Figure 1** below.

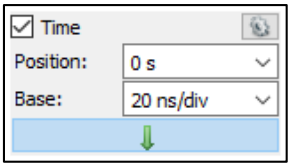

Figure 1: Oscilloscope frequency measurement time base.

5. Take a screenshot of your oscilloscope window, with the clock waveform AND frequency measurement in view. **Include** this screenshot in the Appendix at the end of your lab report.

6. Repeat this process for **32 MHz**. Take another screenshot and include this in your Appendix. **Note:** Since the DAD/NAD's ADC has a maximum sampling frequency of 100 MHz, there will only be a few samples taken per period for a 32 MHz clock signal.

Pre-Lab Questions

1. What is the default clock frequency of the XMEGA?
2. What would you need to do to run the XMEGA at 8 MHz?
3. The XMEGA is rated to run at frequencies up to 32 MHz, but some peripherals run up to either 2x or 4x of this frequency. How can the XMEGA be configured to run at 32 MHz, the 2x peripheral clock at 64 MHz, and the 4x peripheral clock at 128 MHz without using any external clock sources? (There may be multiple solutions)
4. Why are timers useful?

PART B – TIMERS & COUNTERS

In this section, you will learn the basics of XMEGA's Timer/Counter (TC) system. You will analyze a single timer/counter with the Logic State Analyzer feature of your DAD/NAD board and write an assembly program (**`lab3b.asm`**).

Timers are far more useful and accurate than software delays like from the previous lab. They work in tandem with the processor's clock. Your XMEGA has two TCs per port (PORTC through PORTF), for a total of eight TCs. For this lab, you can use either PORTC or PORTF for your timer/counter. By default, each TC has a 16-bit CNT register that stores the current count value. You can read the CNT register or write to it at any time. **Note:** It is possible to split one of the TC systems (TC0) into two 8-bit TCs or concatenate two 16-bit TCs into a single 32-bit TC, though neither of these will be utilized in this course.

Each timer/counter module is comprised of several configuration registers. We are most interested in the TCpn_CTRLA, TCpn_CNT, TCpn_PER, and TCpn_CCm registers (where p = C, D, E, or F [for each respective port], n = 0, 1, or 2 [for each timer counter within a given port], and m = A, B, C, or D).

The CTRL registers (CTRLA through CTRLE) are used for configuring the timers. You will use these registers to select the speed of the timer's clock, the mode the timer will operate in, and to configure specific TC functionality.

For this lab, we will be configuring a timer in normal mode. In normal mode, the PER registers (e.g., TCC0_PER) are used to set the "TOP" of the timer's count. Also in this mode, the timer's CNT register is constantly compared to the PER register. When the CNT value reaches the TOP value, the CNT value is reset back to zero. You can also choose to have events or interrupts triggered when this happens. You will use interrupts in the next lab. Do **NOT** use interrupts for this lab.

LAB 3: CLOCK CONFIGURATION, TIMERS/COUNTERS, EBI

The CTRLA register determines how fast the CNT register is incremented (**see section 14.12.1 of the 8331 manual).** For example, the DIV2 configuration means that the CNT register will be incremented every **two** clock cycles. The higher division you use, the slower the CNT register is incremented. This register also stops the timer (CLKSEL_OFF).

For this lab, we will be configuring a timer/counter's PER register for 0x00FF. This means that the CNT register will start at 0 and count up to 255 (with the most significant byte always 0). Since the CNT registers are typically configured for 16 bits, when writing in assembly, you will need to perform operations to the low and high bytes of the CNT register **separately**. When storing to a 16-bit register (like the TC's PER, CCX, and CNT registers), it is **ALWAYS** necessary to write to BOTH bytes of the register for the change to take effect (**low byte THEN high byte).** You MUST write to the low byte before the high byte.

1. Carefully read sections 14.1-14.4, 14.6, and 14.12 of the 8331 manual. You may also consult doc8045 (*AVR1306: Using the XMEGA Timer/Counter*) for general TC information.
2. Create an assembly program (**`lab3b.asm`**). With the XMEGA running at **2 MHz**, configure a single 16-bit Timer/Counter that counts from 0 to 255. Use the slowest **timer/counter prescaler** available.
3. **Continuously output** the value of the lower byte of the CNT register to either PORTC or PORTF. Make sure that NO backpacks are connected to your µPAD.
4. Connect 8 of the digital I/O pins of your DAD/NAD board to the female headers of your chosen port.
5. Open the Logic State Analyzer (*Logic)* feature inside the *WaveForms* software.
6. Click the green "+" icon to add a "**bus**". You can name the bus if you would like (i.e., CNT, PORTC, etc.). Select the eight DIO signals that are connected to your port and then click the "+" to add them to the bus. Change the "Format" to "**Hexadecimal**". Make sure the MSB in *WaveForms* corresponds to the MSB of the CNT value on your chosen port. Click "Ok" to add the bus.

You should now see all eight bit signals in the LSA window. Run the LSA. You should also see a hexadecimal value representing the TC's CNT value. **Note:** You can verify this CNT value with the I/O window in Atmel Studio.

7. **Submit a screenshot** of your LSA window. It **MUST** show the CNT value incrementing between **at least** 0x00 and 0x0F. To accomplish this, you can set a trigger to freeze the display when the LSA detects 0x00. The easiest way to set a trigger is to click the 'X' in the "T" (trigger type) column, next to the I/O signals. Before taking your screenshot, you will also probably need to adjust the time base and position values.

PART C – EBI

The XMEGA's External Bus Interface (EBI) system is needed to allow the microcontroller to transfer data between itself and external peripherals. The EBI system gives access to the XMEGA's address, data, and control busses. However, before we can interface with an external component, we need to know how to configure the EBI system, as well know how to access/control the necessary EBI busses/signals.

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 registers, and how the EBI generates its appropriate control signals. **Note:** There are several possible modes for which the EBI system can be configured. **Our µPAD is designed to interface with the XMEGA EBI system configured for 3-PORT ALE1 mode (see Figure 27-4).**

Pre-Lab Questions

5. Where does potential external memory space begin and end?

Since our µPAD is designed to interface with 3-PORT ALE1 mode, address bits A15:A8 and address bits A7:A0 of your XMEGA are **time-multiplexed** on an 8-bit bus (**see Figure 36-1 in the 8331 manual**). For these address bits to be time-multiplexed, it means that at certain times the processor outputs address bits A15:A8 on the 8-bit bus, and at other times the processor outputs address bits A7:A0. To be able to use the EBI system to access any specific external memory address location, it is necessary to "latch" the temporary address bits A15:A8. The XMEGA's ALE1 control signal is used to identify **when** to latch these 8 bits.

2. Study Figure 36-1 in the 8331 manual. Focus on the read/write cycles that use the ALE1 signal.

The *Memory Base Board* attached below our µPAD is equipped with a latch (labeled U2). The XMEGA's ALE1 control signal is physically connected to this latch's enable, which is used to properly identify when to latch address bits A15:A8.

3. View the µPAD Memory Base Schematic. Study the external latch (a 74'573 IC), labeled U2.

We should have already determined how to access/utilize address bits A15:A0 of any external memory address, using our XMEGA. Though, since the address space for the external memory in our XMEGA is selectable from 256 bytes (8-bit) up to 16 MB (24-bit), there is **ALWAYS**

LAB 3: CLOCK CONFIGURATION, TIMERS/COUNTERS, EBI

another byte of any memory address, namely the highest byte, which contains address bits A23:A16. To utilize this upper byte of any address (above 0xFFFF), we must use an internal chip select. (This is true because of the 3-Port ALE1 mode that we are using. In other modes, these address pins may be available, e.g., SRAM 3-Port ALE12 mode.)

4. Re-read the Chip Select section (**Section 27.3**) of the 8331 manual. Also re-read Section 27.11 of the 8331 manual (and the timing diagrams in Section 36 in Appendix A).

To utilize the EBI system, the address, data, and control bus pins need to be made available.

5. Carefully read through the Alternate Pin Functions section (**Section 33.2)** of the 8385 manual.

Pre-Lab Questions

6. Why is our EBI configuration named "3-PORT ALE1 mode"?
7. Which ports contain the necessary EBI signals? List each port and which pins are connected to each of the necessary EBI signals.

When configuring the EBI system, to prevent the system from reading/writing at improper times, the necessary control signals must be initialized as outputs, and defaulted to their **false** values.

6. Re-study Figure 36-1 in the 8331 manual. Focus on the read/write cycles that use the ALE1 signal. Identify which signals will need to be initialized, and what value they must be initialized to. **Note:** It will first be necessary to identify the activation-level of each control signal.

In current semesters of 3744, due to a compromise made for board size, there is now a lack of physical space for external components and pins. This leads us to have a more limited use for the EBI system. Though we are unable to easily add an input/output port (see Appendix for more details about what we used to do for EBI in 3744), we can now (and will in this lab) interface with an external SRAM built onto our *Memory Base Board* (mounted underneath our uPAD PCB, and labeled U4 on the Memory Base Schematic). The external SRAM is internally connected to important pins/signals on our XMEGA processor, including certain address/data lines, an XMEGA chip select used to enable the SRAM, as well as the XMEGA's read/write control signals.

7. View the µPAD Memory Base Schematic again. Study the external SRAM, labeled U4.

Pre-Lab Questions

8. What is the memory size of the external SRAM? How do you know?
9. Would our external SRAM be aliased (i.e., partially address decoded) if we chose our chip select size to be 32K? How about 64K?

8. Create an assembly program (**`lab3c.asm`**). In this program, first configure the EBI and place the SRAM at the external memory location 0x20 0000. You must setup the EBI for the SRAM 3-PORT ALE1 configuration, and properly configure the specified chip select and base address to accomplish these specifications. You must also set the direction of the necessary port pins for the various EBI signals, and initialize them accordingly. **Note:** It would be wise to make a subroutine to configure the EBI system and place the external SRAM at a specific memory location.
9. Upon configuring the EBI, write 0xA5 to any **single** memory location in the external SRAM. After writing, read back from the same memory location to verify that the value was properly written. You can use either a Watch Window or the Processor Status Window in Atmel Studio to view the necessary register. Have your program repeat this writing/reading process **forever**.
10. Use the pinouts on the bottom of the memory board, to view the signals (A2:0, D7:0, ALE, CS0, WE, RE) using the LSA tool on your DAD/NAD. You don't have enough DAD/NAD inputs for all of the address pins (this is why A7:3 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.

PART D – PUTTING IT ALL TOGETHER

In this final part, you will combine all of your new knowledge about the CLK, TC, and EBI systems to create an assembly program (**`lab3d.asm`**) to store data read from the DIP switches (located on the *Switch & LED Backpack*) to sequential external memory locations. Every one second, write a new value to the **next** sequential address of the SRAM. After **each** byte is written to the external SRAM, read it back (to verify that it was correctly written), and store the value read from the SRAM to the LED bank (also located on the *Switch & LED Backpack*).

1. Create an assembly program (**`lab3d.asm`**). First configure your XMEGA's clock to run at 32 MHz. **Note:** Unless otherwise noted, for all future programs, you must first configure the XMEGA clock to run at 32 MHz.
2. Configure the EBI system, as in Part C, placing the external SRAM at memory location 0x30 0000.

LAB 3: CLOCK CONFIGURATION, TIMERS/COUNTERS, EBI

3. Configure a timer/counter for normal mode, with a period that corresponds to 1 second. **Note:** the order of steps 2 and 3 are arbitrary.
4. Write the rest of your main program, so that every second, the byte value that is currently on your DIP switches is the value that you store to external memory. After writing the data to the SRAM, read it back and store it to the LED bank located on your *Switch & LED Backpack*. Verify that the stored/read-in value is correct.

Note: There are several ways to implement this program, though you must **NOT** use delay loops. To determine when one second has elapsed, it is recommended that you *poll* the correct timer/counter interrupt flag. If you use this method, make sure to clear the flag when necessary.

PRE-LAB PROCEDURE SUMMARY

1. Answer all pre-lab questions.
2. Learn how to configure the clock. Take screenshots of the DAD/NAD's oscilloscope window.
3. Setup a basic timer/counter. Take a screenshot of the DAD/NAD's LSA window, showing the CNT value properly incrementing.
4. Configure the EBI system, and place the on-board external SRAM at external memory location 0x20 0000.
5. Utilize the CLK, EBI, and TC systems to read and store data to external memory locations at the specified time intervals. Verify this stored data by reading from the same memory location and then storing the read-in data to the LED bank located on your *Switch & LED Backpack*.

IN-LAB PROCEDURE

- Demonstrate Part D of this lab.

REMINDER OF LAB POLICY

Please re-read the *Lab Rules & Policies* so that you are sure of the deliverables that are due prior to the start of your lab. Failure to follow these rules may cause you to earn a zero for this lab.

APPENDIX

A: INPUT/OUTPUT PORTS

As mentioned in Lab 1, The XMEGA has several input/output (I/O) ports (A-F, H, J, K, Q, and R). Each of the I/O pins on all the ports have additional features, some of which will be used in labs during this semester (including this lab!). In previous semesters of 3744, we would "free up" these ports (to be able to utilize their special uses) by adding an **additional** memory-mapped I/O ports. To save on address decoding logic, the memory-mapped I/O port would be aliased (i.e., partially address decoded) as much as possible, and the input and output port would share the same addresses. To create an input port, we would connect an **input source** (in our case, DIP switches) to the inputs of an external tri-state buffer device (with eight buffers), and then connect the outputs of the tri-state buffers to the XMEGA's data bus. To create an output port, we would connect the XMEGA's data bus to the input bus of external D flip-flop register (with eight D flip-flops), and then connect the outputs from the D flip-flop register to an **output source** (in our case, a LED switch bank). Make sure to understand these simple hardware expansion concepts (also discussed in class), as you will be expected to perform them on your exams.

B: USING THE EBI IN C

If you are trying to read or write to a specified EBI address **ABOVE** 0xFFFF in the C language (which will be necessary later in the semester), you will need dedicated software to accomplish this high-address access. There is a header file (header files will be discussed later in the semester) 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 include the header file in your project's dependencies, as well as include it in your program, i.e., include the following at the beginning of your program:

```
#include "ebi_driver.h".
```

Again, we will see more about this when we start programming in C.