Lab 2: Clock Configuration, Timers, RGB LED

OBJECTIVES
• Learn how to make adjustments to the XMEGA’s clock configuration
• Experiment with a basic timer counter and operate an RGB LED

REQUIRED MATERIALS
• uPAD v1.4 + Schematic
• XMEGA documents
• doc8331 (Sections 7, 13 and 14)
• doc8385
• doc8045: AVR1306: Using the XMEGA Timer/Counter
• Digilent/NI Analog Discovery (DAD/NAD) kit

You will also use the oscilloscope of your DAD/NAD board to measure the clock’s frequency.

Read over section 7 of the doc8331 manual, paying close attention to the register descriptions in sections 7.9 and 7.10. The registers you will need to focus on to change the clock speed are OSC_CTRL, OSC_STATUS, CPU_CCP, CLK_CTRL, and CLK_PSCTRL.

The OSC_CTRL register (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 (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 (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 there are three things to do:
1. Wait for the right flag to be set in the OSC_STATUS register.
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 1 (Figure 7-5 in the doc8331 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. So, to set prescalers for the clock:
1. Write the “IOREG” signature to the CPU_CCP register.
2. Write desired settings to the CLK_PSCTRL register.

In order 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 is to use the CLKEVOUT register (doc8331, Section 13.14.4). This is the method you must use for this lab.

ALWAYS create a flowchart or pseudo-code of the desired algorithm BEFORE writing any code.

For this part, you need to create a subroutine that does everything described above. It should enable and select...
Lab 2: Clock Configuration, Timers, RGB LED

the 32 MHz oscillator, as well as scale the clock if desired.

For this lab, set the clock to 8 MHz. Then, configure the CLKEVOUT register to output the system clock (8 MHz) to Port C pin 7.

Make sure there are no backpacks connected to the uPAD (You can leave the memory base attached.) You have access to Port C pin 7 via a female header on the top of the board. Use the oscilloscope to measure the clock signal. In the DAD/NAD 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 20ns/div as shown in Figure 2 below.

Figure 2: Oscilloscope frequency measurement time base.

Take a screenshot of your oscilloscope window with the frequency measurement in view. Submit this in the Appendix of your lab report; also (as usual) submit your program, in this case your lab2a.asm file.

Repeat this process for 32 MHz. Also submit this to your Appendix. Note that since the DAD/NAD’s ADC has a sampling frequency of 105 MHz, there will only be a few samples per period of this 32 MHz clock signal.

PART B – TIMER COUNTER

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 a program called lab2b.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), with a total of eight TCs. For this lab, you can use PORT C or F. By default, these TCs have 16-bit CNT registers. The CNT register stores the current count value of each individual TC. You can read the CNT register or write to it at any time.

When storing to a 16-bit register (like the TC’s PER, CC, and CNT) it is generally necessary to write to BOTH bytes of the register in order for the change to take effect.

Each timer module is comprised of several configuration registers. We are most interested in the TCxy_CTRLA, TCxy_CNT, and TCxy_PER registers (where x = C, D, E, or F [for Port] and y = 0, 1, or 2 [for Timer]).

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 enable specific functions.

The PER registers (e.g., TCC0_PER) are used to set the “TOP” of the timer’s count. Depending on the mode of operation, the timer’s CNT registers are constantly compared to the PER register. When the CNT value reaches the TOP value, the CNT value is reset. You can also choose to have events or interrupts occur when this happens. You will use interrupts in the next lab.

The CTRLA register determines how fast the CNT register is incremented. See section 14.12.1. 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 (DIV0).

Since the CNT registers are typically 16 bits, in general in assembly you would need to perform operations to the low and high CNT bytes separately. For this lab, however, we will only be using the low byte. This means the CNT register will start at 0 and count up to 255 (with the most significant byte always 0).


ALWAYS create a flowchart or pseudo-code of the desired algorithm BEFORE writing any code (in this case, lab2b.asm). For this part of the lab, you will do as follows:
1. With the XMEGA running at 2 MHz, configure a single 16-bit Timer/Counter that counts from 0 to 255. Use the slowest single prescaler available.

2. Continuously output the value of the lower byte of the CNT register to either PORT C or F. Make sure NO backpacks are connected.

3. Connect 8 of the digital I/O pins of your DAD/NAD board to female header of the port you chose.

4. Open a “Logic” analyzer window inside the WaveForms software. Then click the green “+” icon to add a “bus”. Select the 8 DIO signals that you connected to your port and then click the “+” to add them to the bus. You can name the bus if you would like (i.e. CNT, PORTC, etc). Change the “Format” to “Hexadecimal”. Make sure the MSB in WaveForms corresponds to the MSB of the CNT value on PORTC. Click “Ok” to add the bus.

5. Now you should see all 8 bit signals in the LSA window. You should also see a hexadecimal value which should be equivalent to the CNT value.

6. Submit a screenshot of your LSA window. It MUST show the CNT value increment between at least 0x00 and 0x0F. To accomplish this, you will need to set a trigger to freeze the display at 0x00. The easiest way to set a trigger in this case is to click the X in the “T” (trigger type) column next to the I/O signals. You will also probably need to adjust the time base and position values.

PART C – RGB LEDs

For this part, you will use what you know about timers to operate an RGB LED on your uPAD board in a program called lab2c.asm. Look at the uPAD v1.4 schematic and make sure you understand how it is connected. See the GPIO_Output.asm example if you aren’t sure how to turn on/off the LEDs.

From here on out, run your XMEGA at 32 MHz unless you are told otherwise.

LED RGB color codes are typically on a scale of 0 to 255 (0x00 – 0xFF) where 0 is none and 255 is the highest. For example, if you want a light red color you could use a value of 32 = 0x20 for red (and 0 for green and blue, usually referred to as 20-0-0). Since you cannot fine-tune the voltage that gets to the LEDs (since they are only on or off), you may adjust the time the LED is on/off. Using the previous example, if an LED is on for 32 units of time out of 255 units of time, it will appear to be dim! (Note: This 20-0-0 will not appear pink in regular RGB displays.) The voltage to the LED is still either high or low, but by changing the amount of time it is on/off you are tricking the human eyes into thinking it isn’t as bright.

Your task is to write a program that uses a timer to manipulate the RGB LED on the uPAD. You should have 3 values (red, green, and blue) that you can change in your program that determine the color of the RGB LED.

Like Part B, set the PER register equal to 0xFF (255). Use the CNT register to determine when to turn each LED on/off. If on, each LED should turn on at 0 and off at a chosen value. Do NOT use interrupts or the timer/counter’s pulse width modulation (PWM) system. Test your program with various values of RGB. Try 20-0-0, 80-0-0, and FF-0-0; then try 0-FF-0 and FF-FF-00. Describe each of these colors in your lab report.

PRELAB REQUIREMENTS SUMMARY

1. Learn how to configure the clock. Screenshot of oscilloscope window.
2. Setup a basic Timer/Counter. Screenshot of LSA window showing the CNT value increment.
3. Have operating RGB LED

IN-LAB REQUIREMENTS

1. Show the TA your LSA window from Part B with the program running.
2. Demonstrate functionality of your RGB LED. Your TA will specify specific color values.