LAB 4: INTERRUPTS, MORE OF TIMERS/COUNTERS

OBJECTIVES

- Explore and understand microprocessor interrupts.
- Learn how external interrupts operate and how to configure the XMEGA's external interrupt system.
- Learn more about how a timer/counter operates and how to utilize the XMEGA's timer/counter systems for various applications.

REQUIRED MATERIALS

- Documentation:
  - *µPAD v2.0 Schematic*
  - µPAD *Switch & LED Backpack Schematic*
  - XMEGA_AU_Manual (doc8331)
  - ATxmega128A1U_Manual (doc8385)
  - Switch Debouncing using Software
- µPAD 2.0 Development Board
- µPAD Switch & LED Backpack
- NAD/DAD (NI/Diligent Analog Discovery) kit

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

PRE-LAB PROCEDURE

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

You should **ALWAYS** create a flowchart or pseudo-code of a desired algorithm **BEFORE** writing any code. In order to help you practice this habit, the flowchart(s) or pseudo-code for this and all subsequent labs in our course are due **72 hours** prior to the start of your lab.

If a program/design does not work, utilize the debugging capabilities of your board, along with your DAD/NAD and 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!

When storing to a 16-bit register (like the TC's PER, CC, and CNT registers) with assembly language, it is **ALWAYS** necessary to write to **BOTH** bytes of the register (low byte first then high byte) for the change to take effect.

1. Read sections 12 and 13 in doc8331, the first of which describes how interrupts work on the XMEGA, and the second of which specifically describes I/O Ports and how we can use them as interrupt sources.

PART A - RGB LEDS & PWM

An RGB LED is a combination of 3 LEDs in one collective package: one red LED, one green LED, and one blue LED. There are active-high RGB LEDs and active-low RGB LEDs: Active-high RGB LEDs share a common cathode (negative terminal), while active-low RGB LEDs (shown in Figure 1) share a common anode (positive terminal).

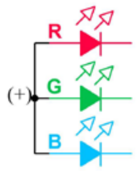


Figure 1: Active-low RGB LED

Note: Your µPAD Development Board is equipped with an active-low RGB LED, labeled "D4" below the female headers at location "J3".

Activating any one of the terminals of an RGB LED will display that specific terminal's color. To produce a wide array of other colors, you can activate multiple LEDs at once, using varying light intensities. If an LED is turned on and off rapidly, the percentage of time that each LED is powered on, or the duty cycle of the LED, determines that LED's light intensity. In computer displays, the RGB values are commonly stored as 8-bit numbers, one for each color (for a total of 24-bit color). This means that the range of intensity for each of the RGB values is from 0-255 in decimal, or \$00-\$FF in hexadecimal. Pulse Width Modulation can be used to control the light intensity of an RGB LED.

Pulse Width Modulation (PWM) is a technique for creating analog voltages with digital systems. Instead of providing a continuous range of voltage with a variable power supply, PWM can be utilized to vary the average output voltage between some minimum (usually 0 V) corresponding to a 0% duty cycle and some maximum (usually Vcc) corresponding to a 100% duty cycle.

The Compare or Capture (CC) channels in the XMEGA's TC system can be utilized to perform pulse width modulation.

1. Carefully read through the TC0/1 section in the XMEGA AU Manual (doc8331, section 14) and the doc8045 manual (especially sections 3, 4, and 6). Make sure you understand the configuration registers, the various waveform generation modes, and how the compare channels are used in the TC0 systems. Unlike the example in the 8045 manual, you should only set the CCx value ONCE in this part, and it is not necessary to do anything with or to the overflow interrupt flag.

2. Explore the µPAD v2.0 Schematic. **Note:** 3V3 on the schematic means 3.3 V. Determine the port/pins connected to the RGB LEDs. To control the RGB LED pins with PWM, will it be necessary to remap the specified port? Refer to the Alternate Pin Functions section in the 8385 manual (section 33). Also refer to the I/O register

descriptions in the I/O Ports section of the 8331 manual (section 13, especially sections 13.13.14 [PORTp_REMAP] and 13.13.15 [PORTp_PINxCTRL]).

3. Write a program in assembly language (`lab4a.asm`) to configure the necessary XMEGA systems and BLUE_PWM LED to output a blue hue of $0E (a light blue, with no red and no green). Remember that the RGB LEDs on the µPAD are active-low (and that 0x0E ➔ 14/256 ➔ 5.5% duty cycle is very much different than 0xE0 ➔ 224/256 ➔ 87.5% duty cycle)!

Notes:
1. For all programs written in this lab, you **MUST** set the XMEGA clock frequency to 32 MHz.
2. As mentioned before, when storing to the TC's PER/CCx 16-bit registers in assembly, it is **ALWAYS** necessary to write to both the low **AND** high bytes.

PRE-LAB QUESTIONS:
1. What would happen if the RGB period was $FFFF instead of $FF?
2. How many TC0 channels are necessary to control all three LEDs in the µPAD's RGB LED?

PART B - SWITCH BOUNCING
Real switches bounce. To debounce a switch, it is possible to create hardware circuitry or create a software delay. In this course, we will debounce switches by creating an appropriate software delay.

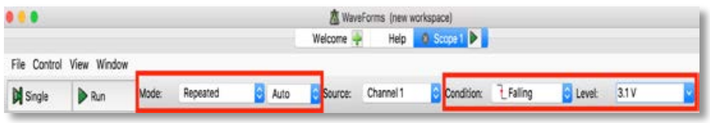
In Lab 2, you created a delay loop to force your processor to wait a pre-determined amount of time. During the delay, no other progress was made in your program. Timers/Counters **with interrupts** provide a better means to create delays, as they do not need to constantly check if the intended time has expired, i.e., other code can run during the time of the delay.

In this part of the lab, you will measure the bouncing of one your DIP switches as well as one of your tactile switches, both of which are located on the *Switch & LED Backpack*. You will use the DAD/NAD's oscilloscope to record both the DIP switch's **AND** the tactile switch's bouncing. (In Part C, you will use your first interrupt, an external pin interrupt; in Part D, you will create a software delay to debounce a tactile switch.)

Since there is no available female header on the *Switch & LED Backpack* for which to insert a DAD/NAD pin to directly measure a switch's output voltage, for simplicity we will measure the necessary switch pins after removing the Switch & LED backpack, and after providing the backpack with external power from the DAD/NAD's voltage supplying pins. **Note:** You can ground your DAD/NAD by connecting the appropriate DAD/NAD wire to the pin labeled GND on the J5 header of the *Switch & LED backpack*.

1. Stop debugging your program, and unplug power from your µPAD. Remove the Switch & LED backpack from the Development Board. Connect the correct voltage supplying pin from the DAD/NAD to the "3V3" pin on the Switch & LED backpack. Ground the backpack with the DAD/NAD's internal ground. Provide 3.3V to the backpack by using the **Supplies** feature within *Waveforms* **NOTE:** If your DAD/NAD's software is incapable of providing 3.3V, it is also okay to power your backpack with 5V.

2. Measure the bouncing of one of your DIP switches and of one of your tactile switches with the DAD/NAD's *Scope* feature in *WaveForms*. It is **NECESSARY AND REQUIRED** that you measure the bouncing upon both opening and closing each type of switch. Therefore you should set both falling-edge and rising-edge triggers for each switch. It is recommended that you use the ***Normal*** setting within the ***Repeated*** mode of your *Scope* program. Make sure that you are using the appropriate time base for your waveform. It is also recommended that you set a voltage level trigger to detect both edge types (something within the 0 to 3.3 V range of the switch). See Figure 2 for more details. Submit screenshots in your lab document for **each** switch type that show the bouncing upon opening, and upon closing, the switch. In order to get enough resolution to see the bouncing, it is preferable to have **two** separate screenshots, one for when a switch is closing and another for when it is opening. Record the amount of bounces for both opening and closing each switch, and then determine a reasonable amount of time that could be used to debounce your switches (i.e., this debounce time must be greater than both the time of bouncing for both a falling AND a rising edge). Try to record the bouncing more than once to see if the bouncing is always the same.



Mode: Repeated Auto
Source: Channel 1
Condition: Falling
Level: 3.1 V

Figure 2: Examples of recommended waveform settings within the *Scope* feature of *Waveforms*.

Note: It is *possible* that the falling edge of your tactile switch will appear to have almost no bouncing, i.e., the bounce time will occur in the order of microseconds. This should **NOT** be the case for any rising edge obtained from your tactile switch, i.e., the bounce time should be in the order of milliseconds.

PART C - EXTERNAL INTERRUPTS
In Lab 2, you had to continuously poll the input port of your tactile switches to know the state of the switches. Polling wastes processor clock cycles and results in a slow response time if the processor is engaged in another instruction/execution task. What if you wanted the processor to respond instantly to any changes on an input switch? The answer is to utilize external interrupts. A properly written

interrupt driven program should have the format in shown Figure 3:

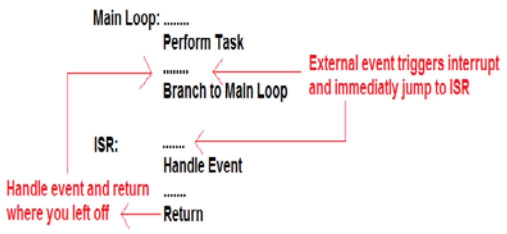

Main Loop: ....
Perform Task
....
Branch to Main Loop
External event triggers interrupt and immediatly jump to ISR
ISR:
....
Handle Event
Handle event and return where you left off
....
Return

Figure 3: Proper program flow for an interrupt-driven program.

1. Read doc8331, sections 13.5 through 13.7 regarding external interrupts on the XMEGA chip. Then read section 13.13, which describes many of the various registers you will need to use. Read the first four paragraphs of section 13.6 very carefully.

2. Read doc8331, section 12: *Interrupts and Programmable Multilevel Interrupt Controller* (PMIC). The PMIC_CTRL register determines the level of interrupt(s) that are enabled (see section 12.5 and Table 12-1). When configuring a peripheral, pay close attention to the registers associated with that peripheral. See *Register Descriptions* (section 12.8) to understand each of the registers associated with interrupts and the PMIC.

In this part, you will create an assembly program (`lab4c.asm`) to initialize the XMEGA to trigger an interrupt when **tactile switch S1** is pressed. Look at the switch's circuit design (on the *Switch & LED Backpack Schematic*) to see which edge will first occur when the switch is pressed, and **use** that edge. The interrupt service routine (ISR) should only execute when the pin value changes as determined above. However, because we are not willing to electrically debounce this switch (as we did in EEL 3701), the ISR will fire both when the switch is pressed and released (assuming that the switch bounces during the release time). Therefore, inside the ISR, you must test the value of the pin, and only complete the desired function if the pin's voltage is at the correct level.

Note: You must (as I have always done in my examples) put the origin of your main routine at 0x100 or beyond. (My examples have all skipped addresses from 0x0 to 0x100, or 0x0 to 0x200.) This is necessary because the interrupt vectors (which will be used in this lab and all future labs) are located in the low addresses of program memory, i.e., 0x00 through 0xFD.

3. Create an assembly program (`lab4c.asm`) to configure an external interrupt for tactile switch S1. To prove that the interrupt is completely independent from other code running on the XMEGA, set up an endless loop to continuously toggle on/off your BLUE_PWM **(the name given in the µPAD schematic)** RGB LED as quickly as possible, i.e., do **NOT** use a timer system; instead, just use a few lines of code within an endless loop. Refer to the *Switch & LED Backpack Schematic* and the *µPAD v2.0 Schematic* for information about the switch and RGB LEDs. The initializations in your MAIN routine should proceed as follows:
   a. Give the pins appropriate default values and then set the pin directions for your RGB LEDs.
   b. Select an interrupt priority level in the interrupt control register (PORTp_INTCTRL) for a port of your choosing.
   c. Select the appropriate pin on that same port as a source for the interrupt in one of the interrupt mask registers (PORTp_INTnMASK).
   d. Be sure to select the data direction for the input pin.
   e. Select the input/sense configuration for the pin (that you selected in the PORTp_PINnCTRL register) that corresponds to the edge that first occurs when the tactile switch is pressed.
   f. Turn on the appropriate PMIC interrupt level in the PMIC_CTRL control register.
   g. In any interrupt driven application, enabling the global interrupt enable (I bit) should be the last thing you do during initializations. Simply use the `sei` instruction to enable interrupts (to set the I bit).
   h. Within an endless loop in the MAIN routine, toggle on/off your BLUE_PWM RGB LED as quickly as you can (without using a timer system).

Now you will create the necessary interrupt service routine that the processor will execute shortly after the respective interrupt occurs. The ISR is the code that the processor will execute shortly after the respective interrupt occurs. After an interrupt occurs, there is a specific address to which the processor will jump, often called the interrupt vector address. The name of the vector used in this part of the lab for your external interrupt will be PORTp_INTn_vect, where p is the port of your interrupt and n is interrupt source 0 or 1. (You have choices for both p and n.) This vector's address value is defined in the include file of every program you have written thus far. (In C, which we will use later this semester, the vector's values are available in the avr/io.h file.)

After determining the correct vector address label, you must tell the processor where to go if it encounters that vector address. This is accomplished with an `rjmp` or `jmp` instruction, just as you should have done for the highest priority interrupt, RESET, at the start of every program. Just like the RESET interrupt vector is defined with an `rjmp MAIN` following an `.org 0`, the interrupt vector must be defined similarly with an `.org` <**interrupt_vector**> followed by `rjmp` **<ISR_LABEL>**, where **interrupt_vector** is found in the include file as x_vect for interrupt type x, and where **ISR_LABEL** is the ISR's label.

4. Write an external pin interrupt ISR to display a count of how many times the external interrupt has been executed on your port of eight LEDs (located on the *Switch & LED Backpack*). Be sure to initialize the count to zero and the LED pin directions to output. For this and most interrupts, you must clear the interrupt flag before returning from the ISR. The external pin interrupt flags are in a PORTp_INTFLAGS register. Most flags are cleared by storing a 1 to the flag bit.

5. Make sure your interrupt service routine is working correctly by putting a breakpoint inside of it (remember the interrupt will likely fire on both edges because of bouncing). Real switches bounce, so the count will probably be wrong! This is okay in this part of the lab.

Table 1: Switch triggered RGB LED configurations

Setting	# of times S1 pressed	Functionality
0	0, 4, …	RGB LEDs OFF
1	1, 5, …	RGB LEDs ON; Display "The Incredible Hulk Purple" (RGB: 0x8A2C9A) for 0.1 seconds; Display "The Incredible Hulk Green" (RGB: 0x49FF07) for 0.1 seconds.
2	2, 6, …	RGB LEDs ON; Display "Holiday Red" (RGB: 0xC21F1F) for 0.1 seconds; Display "Holiday Green" (RGB: 0x3C8D0D) for 0.1 seconds
3	3, 7, …	RGB LEDs ON; Display "University of Florida Orange" (RGB: 0xFA4616) for 0.1 seconds; Display "University of Florida Blue" (RGB: 0x0021A5) for 0.1 seconds

PART D - INTERRUPTS, CONTINUED

In this section, you will create an assembly program (`lab4d.asm`) to debounce tactile switch S1. S1 will trigger an external interrupt which will be used to debounce tactile switch S1. There are many techniques possible to accomplish switch debouncing, as described in the *Switch Debouncing with Software* document on our course website. Use any method that does not waste time in an ISR and does not alter the code used in your MAIN routine for Part C. Start this program by copying all the code from `lab4c.asm`.

1. Create an assembly program (`lab4d.asm`) to debounce your S1 tactile switch, so that the count value displayed on your LEDs is a correct indication of the amount of times that the push-button has been pressed. **Note:** The infinite loop in your MAIN routine from Part B should be unchanged. This loop should be able to be replaced with almost any other code and have **NO EFFECT** on the detection of pressing tactile switch S1.

PART E - PUTTING IT ALL TOGETHER

You will now use all your knowledge about timers and interrupts to write a program (`lab4e.asm`) that blinks your RGB LEDs between specified pairs of colors every 0.1 seconds. Table 1 indicates the configuration for your RGB LEDs based on the number of times that S1 is pressed.

When your program begins, your RGB LEDs must be off. Whenever tactile switch S1 is pressed, your RGB LEDs must alternately blink between two specified colors (see Table 1). Each color will remain on for 0.1 seconds.

NOTE: The colors are likely to not appear accurate until they are blinking at the specified rate of 0.1 seconds!

1. Create a flowchart/write pseudocode for this program **BEFORE** writing any code. Think about how many timer systems you will need, as well as how many interrupts are necessary. Remember to include this and all other flowcharts/pseudocode in your lab document.
2. Create an assembly program (`lab4e.asm`) to perform the specified RGB LED blinking. **Note**: There are many ways to design this program. It is **HIGHLY** recommended that you create a table of RGB color data for use by your program. It is also **HIGHLY** recommended that you make subroutines for your initializations (rather than put your initializations in your main routine.

Notes:
1. The program for Part E should utilize code and concepts previously used in this lab. This part is very time-intensive, so do **NOT** wait to start.
2. When turning off the TC system that uses PWM, the waveforms currently being outputted on selected pins will be remain with their last value. Therefore, a force restart of the TC system may be necessary.

PRE-LAB PROCEDURE SUMMARY

1. Answer all pre-lab questions.
2. Create `lab4a.asm` to set up a timer system and control the BLUE_PWM RGB LED with PWM, as described in Part A.
3. Measure and record the bouncing of one of your DIP switches and of one of your tactile switches, as described in Part B. Include the necessary screenshots in your lab report.
4. Configure an external interrupt for tactile switch S1 on your Switch & LED backpack, and complete the assembly program **(**`lab4c.asm`**)** in Part C.
5. Configure another timer system to properly debounce your tactile switch S1, and complete the assembly program **(**`lab4d.asm`**)**, as specified in Part D.
6. Write an assembly program **(**`lab4e.asm`**)** for Part E, to output/blink various specified color-pairs on the RGB LED of your µPAD.

IN-LAB PROCEDURE

1. Demonstrate switch bouncing using your DAD/NAD, as in Part B.
2. Demonstrate Parts D and E 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.