Lab 3: Interrupts and Timer/Counters

OBJECTIVES

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

REQUIRED MATERIALS

- uPAD 1.4 Development Board
- uPAD 1.4 Switch & LED Backpack
- NAD/DAD (NI/Diligent Analog Discovery) kit
- You WILL need the following documentation:
  - uPAD v1.4 Schematic
  - uPAD 1.4 Switch & LED Backpack Schematic
  - doc8331: XMEGA_AU_Manual

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.

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.

If a program/design does not work, utilize the debugging capabilities of your Atmel ICE 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!

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.

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).

NOTE: Your uPAD Development Board is equipped with an active-low RGB LED, labeled “D5” below the female headers at location “J4”.

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). Make sure you understand the configuration registers, the various waveform generation modes, and how the compare channels are used in the TC0 systems.

2. Explore the uPAD v1.4 Schematic. (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 [PORTx_REMAP] and 13.13.15 [PORTx_PINxCTRL]).

3. Write a program in assembly (lab3a.asm) to configure the necessary XMEGA systems and BLUE_PWM LED to output a blue hue of $0F. Remember that the RGB LEDs on the uPAD are active-low (and that 0xF0 \( \rightarrow \) 240/256 \( \rightarrow \) 94% duty cycle) and very much different than 0xF0 \( \rightarrow \) 15/256 \( \rightarrow \) 6% duty cycle)!
NOTES:
1. For all programs written in this lab, you **MUST** set the XMEGA clock frequency to 32 MHz.
2. When storing to the TC’s PER/CCs 16-bit registers in assembly, it is **ALWAYS** necessary to write to both the low AND high bytes (since the default values are 0xFF).

**Prelab Questions:**
1. How many TC0 channels are necessary to control all three LEDs in the uPAD’s RGB LED?
2. What would happen if the RGB period was $FFFF$ instead of $0000$?

**PART B: EXTERNAL INTERRUPTS**

In Lab 1, 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.

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 section 12.8: **Register Descriptions** to understand each of the registers associated with interrupts and the PMIC.

In this part, you will create an assembly program (**lab3b.asm**) to initialize the XMEGA to trigger an interrupt when tactile switch S1 (or S2) is pressed, i.e., chose one of the two tactile switches. Look at the switch’s circuit design to see which edge will first occur when the switch is pressed, *but do NOT trigger on that edge*. Create a level or edge triggered interrupt (your choice). (The tactile switch S1 is connect to PortF, pin 2; read the third paragraph of doc8331, section 13.6 to see why we should be very careful about using edge triggering for this switch. Note that we could use edge triggering with a more complex algorithm. Tactile switch S2 is connected to PortF, pin 3, which does not have the same hardware as PF2.) The interrupt service routine (ISR) should only execute when the pin value gets the trigger condition that you specified. However, because we are not willing to electrically debounce this switch, the external pin ISR will fire when the switch is pressed and maybe again when the switch is released (if the switch bounces upon release). Therefore, inside the ISR, you must test the value of the pin, and only complete the desired function if the pin is the correct level.

3. Create an assembly program (**lab3b.asm**) to configure an external interrupt for one of the two tactile switches and then continuously toggle on/off your BLUE_PWM RGB LED. Refer to the **Switch & LED Backpack Schematic** and **uPAD v1.4 Schematic** for information about the switch and RGB LEDs. The initializations in your program 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 (PORTx_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 (PORTx_INTnMASK).
   d. Be sure to select the data direction for the input pin. Do not make assumptions.
   e. Select the input/sense configuration for the pin you selected in the PORTx_PINnCTRL register. Make sure you select a proper trigger for the chosen interrupt.
   f. Turn on appropriate PMIC interrupt level in the (PMIC_CTRL) control register.
   g. In any interrupt driven application, the global interrupt flag should be the last thing you set during initializations. Simply use the **sei** instruction to set the global interrupt flag to enable the interrupt.
   h. In an infinite loop within the MAIN routine, toggle on/off your BLUE_PWM RGB LED AS QUICKLY AS YOU CAN.

**NOTE:** If you plan to make a subroutine for the external interrupt initializations in step 3, which is recommended, it is also recommended to only set the PMIC_CTRL register and global interrupt flag in the main routine. When writing multiple subroutines to initialize interrupts (which you will do later in this lab), it is ideal that no interrupts trigger before ALL necessary systems are configured.

The processor needs to know what code to execute when an interrupt occurs. After an interrupt occurs, there is a specific address to which the processor will jump. This address is called the interrupt vector address. The name of the vector for your external interrupt will be PORTx_INTn_vect where x is the port of your interrupt and n is interrupt source 0 or 1. (You have choices for x and n.) This vector’s value is defined in the include file at the top of every program you’ve written thus far. (In C, the vector’s values are available in the avr/io.h file.)

4. Write an ISR to display a count of how many times the external interrupt has been executed, on the LEDs located on the Switch & LED Backpack. (Be sure to initialize the count to zero and LED pin directions to output.)

**Notes:** The label of the ISR must be the same as what you used in the interrupt **rjmp** or **jmp** instruction after the .org to the
interrupt vector. The ISR must end with a `reti` instruction. The `reti` instruction is a special return statement reserved only for interrupt service routines and is (slightly) different from a `ret` instruction. By naming your ISR the same as the vector, the compiler knows how to put the appropriate `rjmp/jmp` instruction in the vector table, based on where it locates the actual routine in memory. With this jump instruction, the processor knows what to execute when your external interrupt occurs. The technique for setting the ISR address is the same as you have been using when setting the MAIN address (at address 0x0), i.e., place an `rjmp/jmp` at the proper location to jump to the ISR (e.g., ISR_LED_COUNT).

5. In the ISR, make sure to clear the interrupt flag by writing a one to the appropriate place in the `PORTx_INTFLAGS` register.

Notes: You must, as always, put the origin of your program at 0x100 or beyond. This is necessary because the interrupt vectors, which we will use in this lab, are located in the low addresses of program memory (0x00-0xFD). You therefore also still need an appropriate `jmp` or `rjmp` instruction at 0x0. The 0x0 address is the reset interrupt vector.

6. Make sure your interrupt service routine is working correctly by putting a breakpoint inside of it (remember the interrupt will seem to 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.

PART C: INTERRUPTS, CONTINUED

1. In this part of the lab you will create an assembly program (`lab3c.asm`) to debounce one of the two tactile switches. This switch will trigger an external interrupt (from Part B, i.e., copy all code from `lab3b.asm`) which in turn configure a timer/counter system to debounce this tactile switch. The flowchart in Figure 2 shows one possible technique (of many) for solving the problem. (You are free to choose another technique but be careful that it works and does not cause you to waste time in an ISR.)

Use the DAD/NAD’s oscilloscope to record the tactile switch’s bouncing, and then write an assembly program (`lab3c.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.

Since there is no available pin in which to insert into a DAD/NAD female header to directly measure the switch value on the Switch & LED backpack, you will need to insert a small wire or pin into one of the DAD/NAD female headers to touch to the appropriate pad on the chosen tactile switch. You can use the ground female header on the DAD/NAD to connect to the pin labeled GND on the J5 header of the Switch & LED backpack.

2. Measure the bouncing of chosen tactile switch with the DAD/NAD’s oscilloscope feature. It is NECESSARY AND REQUIRED that you measure the bouncing upon both pressing the push-button, and releasing it. Therefore, it is recommended to set both falling-edge and rising-edge triggers for the switch. (It might be necessary to set a level on the DAD/NAD to detect the edges. If so, select something within the 0 to 3.3 V range of the switch.) Record the amount of bounces for both pushing and releasing the tactile switch, and determine a reasonable amount of time that could be used to debounce your switch (i.e., this debounce time must be greater than the time of bouncing for both falling AND rising edges). Submit screenshots of the switch’s bouncing in your lab document, along with your recordings of the number of
bounces and your determined debounce time. In order to get enough resolution to see the bouncing, it is preferable to have two separate screen shots, one for when the switch is pressed and another for when it is released. Try to record the bouncing more than once to see if the bouncing is always the same.

In Lab 1, you created a delay loop to force your processor to wait a pre-determined amount of time. During the delay, no other progress is made in running your code. Interrupt service routines are a better means to create specific delays, since other code can run during the delay time. Interrupt service routines should generally be as short as possible so that this gain is not lost, i.e., designing a long ISR is bad practice because it means that the processor stops non-interrupt processing from moving forward and other interrupt routines (assuming that you haven’t explicitly allowed ISRs to nest/preempt each other, which is also generally discouraged) cannot be processed.

3. Configure the necessary timer/counter interrupt vector.

Follow the steps below in the flowchart of Figure 2:

4. Write a subroutine to configure a timer/counter system using normal mode. Set up the timer/counter to create a delay your determined previously to deal with the bouncing of your tactile switch. Do not turn on the timer/counter yet. (Use a different timer/counter than you used in Part A.)

5. Write a subroutine to configure an overflow interrupt (see TCxn_INTCTRLA) for your timer/counter system

6. Modify the external interrupt ISR that you wrote in Part B to enable the timer/counter overflow interrupt to create the necessary delay.

7. Write a timer/counter overflow ISR that disables the timer/counter overflow interrupt and re-enable the external pin interrupt. If the switch is in its pressed position, increment the counter and update the LEDs; otherwise, do not increment the counter.

8. The infinite loop in your MAIN routine from Part B should be unchanged. This routine can be replaced with almost any other code with NO EFFECT on the detection of the pressing of tactile switch.

9. Verify that you correctly debounced your switch by running your program and confirming that the LEDs indicate the actual number of times that you press the tactile switch.

### PART D: PUTTING IT ALL TOGETHER

You will now use all your knowledge about timer/counters and interrupts to write a program (lab3d.asm) that blinks your RGB LEDs between specified pairs of colors every 0.5 seconds. Table 1 indicates the configuration for your RGB LEDs based on the number of times that the switch is pressed.

<table>
<thead>
<tr>
<th>Setting</th>
<th># of times S1 pressed</th>
<th>RGB LEDs Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0, 4, …</td>
<td>RGB LEDs OFF</td>
</tr>
<tr>
<td>1</td>
<td>1, 5, …</td>
<td>RGB LEDs ON; Display “University of Florida Orange” (RGB: 0xFA4616) for 0.5 seconds; Display “University of Florida Blue” (RGB: 0x0021A5) for 0.5 seconds</td>
</tr>
<tr>
<td>2</td>
<td>2, 6, …</td>
<td>RGB LEDs ON; Display “Holiday Red” (RGB: 0xC21F1F) for 0.5 seconds; Display “Holiday Green” (RGB: 0x3C8D0D) for 0.5 seconds</td>
</tr>
<tr>
<td>3</td>
<td>3, 7, …</td>
<td>RGB LEDs ON; Display “The Incredible Hulk Purple” (RGB: 0x8A2C9A) for 0.5 seconds; Display “The Incredible Hulk Green” (RGB: 0x49FF07) for 0.5 seconds</td>
</tr>
</tbody>
</table>

When your program begins, your RGB LEDs must be off. Whenever the tactile switch is pressed, your RGB LEDs must alternately blink the two specified colors (in Table 1). Each color will remain on for 0.5 seconds.

1. Create a flowchart/write pseudocode for this program BEFORE writing an assembly program. Think about how many timer/counter 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 (lab3d.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 which you program. It is also HIGHLY recommended that you make subroutines for your initializations.

### NOTES:

1. The program for Part D 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 kept. Therefore, a force restart of the TC system may be necessary.

### PRE-LAB REQUIREMENTS SUMMARY

1. Answer all pre-lab questions.

2. Create lab3a.asm to set up a timer/counter system and control the BLUE_PWM RGB LED with PWM as described in Part A.
3. Configure an external interrupt for a tactile switch on your Switch & LED backpack and complete the assembly program `lab3b.asm` in Part B.

4. Measure and record the of bouncing of your tactile switch S1 as described in Part C.

5. Configure another timer/counter system to properly debounce your chosen tactile switch and complete the assembly program `lab3c.asm` in Part C.

6. Write an assembly program `lab3d.asm` for Part D to output/blink various specified color-pairs on the RGB LED on your uPAD.

**IN-LAB REQUIREMENTS**

1. Demonstrate tactile switch bouncing using your DAD/NAD.

2. Demonstrate Parts B, C, and D of this lab.