Lab 4: Interrupts and Serial Communication

OBJECTIVES
• Explore and understand microprocessor interrupts. In part A of this lab, you will use XMEGA external interrupt system.
• Learn how to utilize asynchronous serial communication. In part C of this lab, you will use XMEGA’s USART system to transmit/receive characters to/from computer terminal as well as write interrupt driven USART routines and explore multitasking.

REQUIRED MATERIALS
• uPAD and Proto Base kit and tools
• DAD or NAD
• No new hardware
• XMEGA documents
  • doc8331, section 12, 13, and 23
  • doc8385, section 33 (pinout and pin functions)

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

PRELAB REQUIREMENTS

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

PART A: EXTERNAL INTERRUPTS
In Lab 2 of this course, you added switches to your board. You had to continuously poll the input port 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. The program for this section should be called Lab4_ext_intr.asm.

1. Read doc8331, sections 13.5-13.7 regarding external interrupts on the XMEGA chip. Then read section 13.13, which describes the various registers you will need to use.
2. Read doc 8331, chapter 12 about the Programmable Multilevel Interrupt Controller (PMIC). The PMIC_CTRL is a register that must be set using the table in section 12.5 regarding interrupt levels. As always with configuring a peripheral, pay close attention to section 12.8, which describes the registers associated to the PMIC.
3. Connect a single switch to any unused pin of your choice (PORTs E or F). Add this switch to your circuit diagram from the last lab and submit this in your lab document. Be sure that your keypad circuit remains functional for future labs.
4. You will initialize XMEGA to trigger an interrupt when a falling edge is detected on the input. The interrupt service routine (ISR) should only execute when the pin value changes from to Vcc to GND. However, because we are unable to debounce the switches on our board, the ISR will fire both when you toggle the switch on and off (due to the switch bouncing during both times). Therefore, inside the ISR you will have to test the value of the pin and only complete the desired function if the pin is the correct level.

5. Your initializations should proceed as follows:
   a. Select an interrupt priority level in the interrupt control register (PORTx_INTCTRL) for a port of your choosing.
   b. Select a pin on that same port as a source for the interrupt in one of the interrupt mask registers (PORTx_INTnMASK).
   c. Be sure to select the data direction for the input pin. Do not make assumptions.
   d. Select the input/sense configuration for the pin you selected in the PORTx_INTnCTRL register. Make sure you select sense falling edge.
   e. Turn on appropriate PMIC interrupt level in the (PMIC_CNTL) control register.
   f. 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.

6. 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 this interrupt is 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.)

7. Write an ISR to display a count of how many times interrupt has been executed, on your LEDs. (Be sure to initialize the count to zero and LED pin directions to output). The label of the ISR must be the same as what you used in the interrupt jmp 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 jmp/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 0), i.e., place an rjmp/jmp at the proper location to jump to the ISR (e.g., ISR_LED_COUNT).

8. You must, as always, put the origin of your program at 0x100 or beyond. This is necessary because of the interrupt vectors, which we will use in this lab, are located in the lower 506 addresses of program memory (0x000-0xFD). You would also still need an appropriate jmp or rjmp instruction at .org 0. The 0 address is the reset interrupt vector, which cause your program to start executing at the address corresponding to the label for this jmp or rjmp instruction.

9. Make sure your interrupt routine is working correctly by putting a breakpoint inside of it. (Remember the interrupt will seem to fire on both edges because of bouncing.)

10. Real switches bounce, so the count will probably be wrong! Record the number of bounces that you see when the switch goes from low to high and from high to low. (Do this a few times to see if it is always the same.) Put the shortest possible delay routine inside your ISR. (See the Appendix B for a discussion of placing delay routines inside interrupts, which should generally be avoided.) The delay should be near the start of the ISR, in order to give the pin time to stop bouncing before you read its value. Set your breakpoint after you check the pin’s level to make sure you are executing the counting only on a falling edge.

11. Use your DAD/NAD’s oscilloscope to record the bouncing (used in part 10). Submit a screenshot in your lab document.

PART B: ASYNCHRONOUS SERIAL COMMUNICATION

Asynchronous serial communication can be done entirely in software by “bit banging.” For example, if you understand the format of asynchronous serial data (see class notes and Figure 23-5 in the doc8331), and you want to send data, then all you have to do is to transmit a block of data on a GPIO pin. You would output the following bits at the required frequency or baud rate: start bit, 8 data bits, no parity, 1 start bit, and 1 stop bit. This bit banging is NOT part of the lab requirements but is always an option if the processor has no support for serial communication. Similarly, a receiver can be made by sampling inputs at the appropriate times. NOTE: This section is informational only, i.e., no deliverables are required.

PART C: XMEGA USART System

The XMEGA has several universal synchronous asynchronous receiver transmitter (USART) peripherals. One of them, PORTD’s USARTD0, is connected to the USB port on the uPAD through a FTDI USB bridge chip (FT232RL). See the uPAD schematic and user manual for more information. Determine which pins on PORTD are used for USARTD0 using table 33-3 in doc8385 (alternate pin functions). (You won’t need this info, but include this answer in your prelab.)

In this part of the lab you will write a program (called Lab4_serial.asm) to send and receive data between uPAD and your laptop using USARTD0.

You will need to install a terminal program on your laptop to communicate with your board. Some examples are Bray Terminal (also known as Br@y++ Terminal), PuTTY, X-CTU, RealTerm, and HyperTerminal. Install and familiarize yourself with one of them. Information on how to use Bray Terminal is given in the Appendix A. The example program used in class is available on our web site (SCI_Polling.asm) and can be used as a starting point for your program.

To successfully complete this portion of the lab, I suggest that you create the subroutines/functions described below.

1. USART Initialization function (USART_INIT). This function will take care of all of USART initializations.
   a. Set the data direction of the PORTD USART bits to the corresponding values for Tx and Rx.
   b. The uPAD includes USB support, but we will instead use the UART (called USART in the XMEGA) to communicate with USB (through a chip from FTDI) in order to pass data between the XMEGA and your laptop. See the Appendix D and Figures A1 and A2 for more information on these circuits. PortQ bits 1 and 3 are critical and MUST be configured correctly. Initialize these pins as outputs and output the appropriate values to these pins. Complete uPAD schematics are available on our website under Software/Docs.
   c. Set the baud rate to 38,400 Hz (bits per second) by storing the appropriate value in the baud rate registers. (You can use a program at http://tinyurl.com/lhunuy2 to verify your baud rate calculation, but be sure that you know how to calculate these for your lab quiz and exams.) See the Appendix C for a discussion of the definition of “baud.”
   d. Set the USART for asynchronous mode, 8 data bits, no parity, 1 start bit, and 1 stop bit.

2. Character output subroutine (OUT_CHAR). This subroutine will output a single character to the transmit pin of the XMEGA’s USART system.
   a. Check if the previously transmitted character has been completed; if not, keep checking until it has been completed.
   b. Transmit the character passed to the subroutine.

3. String output subroutine (OUT_STRING). This routine will output character strings stored in memory.
   a. Read the character pointed to by Z and increment the pointer.
   b. If this character is not null, call OUT_CHAR, and repeat for each non-null character that follows; otherwise return from the subroutine.
4. **Character input subroutine (IN_CHAR).** This subroutine will receive a single character from the receiver pin of the XMEGA’s USART system.
   a. Check if a character has been received, if not, keep checking until it has been received.
   b. Read the character from the receive buffer.

**PART D: Baud Rate Verification**
To verify that your baud rate is working correctly, modify your previous program (`Lab4_serial.asm`) to confirm the baud rate and to determine its accuracy. Unfortunately, there is no easy access to the XMEGA’s transmit and receive pins for PortD on either uPAD or the uPAD Proto Base, so you should test your baud rate using either PortE or PortF. Call this program `Lab4_serial_baud_test.asm`.
   a. Transmit the character “U”, which is also the number 0x55.
   b. Delay 500 μs.
   c. Repeat.
This will allow you to measure the width of one bit on your DAD/NAD board. Use your DAD/NAD board to measure the width of this bit and show that it is the correct length for 38,400 baud transmission.

**PART E: Serial Menu**
Modify a previous program of yours to write an interactive program, called `Lab4_serial_menu.asm`. The purpose of this program is to display one of favorite things on your computer screen when prompted to do so. The menu you will create should look like this:

```plaintext
your_name's favorite:
1. Book
2. Pizza toppings
3. Ice cream/yogurt flavor
4. Food
5. Actor/Actress/Reality “Star”
6. Re-display menu
ESC: exit
```

When your program first starts it should display the above menu using OUT_STRING subroutine and then wait for an incoming character. When a 1, 2, 3, 4, 5 or 6 is received, output the corresponding message, e.g., for 4 you program might respond, “*Cookie Monster’s favorite food is cookies,*” and redisplay the menu. Replace the items in italics with your name, favorite item selected, and your favorite of that item, respectively. If the received character is an escape (ESC) character, output “Done!” and terminate execution. If any other character is received, ignore it. For readability, add a blank line between every message.

You will need two special characters to cause the cursor to move down a line. These characters should occur in order at the end of every line. In addition, an escape character (ESC) will be used to terminate your program. **Note:** On some tablet PCs last semester, the ESC key code did NOT return a $1B. If this is the case for your computer, just pick another key and let Dr. Schwartz and your TA know by sending them emails (and telling them what type of computer you are using). Also document this in your pre-lab report.) Add the following equates to your assembly file:

```assembly
.equ CR = 0x0D
.equ LF = 0x0A
.equ ESC = 0x1B
```

ASCII characters can be represented as a single character in single quotes (e.g., ‘A’ or a string of characters in double quotes (e.g., “this is a string of ASCII characters”).

The best way to write and debug code is to test **as you write it.** I suggest that you start by writing a simple program to read a character (using IN_CHAR), then write a simple program to output a character (using OUT_CHAR), and then write program to echo the single character that is received with IN_CHAR by transmitting the read data using OUT_CHAR. Write a final simple program to test your OUT_STRING subroutine before finally writing the required program to display the menu and execute the menu specified tasks.

**PART F: Interrupt Driven Receiving**
We have already used interrupts in part A of this lab. Now you will use your knowledge to create an interrupt driven echo program. Call the program that you write in this section `Lab4_serial_int.asm`.

1. Initialize USART as in part C, except also enable low level receiver interrupt (using CTRLA).
2. Turn on appropriate PMIC interrupt level in the control (PMIC_CTRL) register.
3. Write an ISR (triggered on a USART receive) that echoes each received character to the transmitter. Writing an ISR is very similar to writing a subroutine, with the main differences of often needing to clear a flag (caused by the interrupt) and returning with an RETI instead of a RET. You will also generally need to push and pop several registers.
4. To demonstrate the concept of an interrupt driven program, after your initializations, toggle an LED “forever” with a 0.37 second delay in a loop in the main routine, and do nothing else! At the same time, if you type a key on your terminal, your program should echo back the single character inside an ISR. The interrupt should now do all the work for you. A properly interrupt driven program should have the following format:
Lab 4: Interrupts and Serial Communication

PRE-LAB QUESTIONS
1. List the XMEGA’s USART registers used in your programs and briefly describe their functions.
2. What is the difference between synchronous and asynchronous communication?
3. What is the difference between serial and parallel communication?
4. List the number of bounces from part A of this lab. How long (in ms) is your delay routine for debouncing?
5. What is the maximum possible baud you can use for asynchronous communication if your board runs at 2 MHz? Support your answer with the values you would place in any special registers that are needed.

IN-LAB REQUIREMENTS
1. Part A: Demonstrate your external interrupt program executing on your board.
2. Part C: Demonstrate your USART menu program executing on your board.
3. Part E: Demonstrate USART menu program executing on your board.
4. Part F: Demonstrate your interrupt driven echo program executing on your board.
5. Answer any questions your TA may ask you about your hardware and software.
APPENDIX

A: Using Bray Terminal

You will first need to download the Bray program from our website at http://mil.ufl.edu/3744/software/Bray.exe. More information about this program is available at the website https://sites.google.com/site/terminalbpp/.

After you write your program, you will need to configure the Br@y++ terminal as follows:

1. With your XMEGA Board powered you will need to determine the virtual serial port that was created. To do this:
   a. Right click Computer
   b. Select Properties
   c. Click Device Manager
   d. Check to see the Ports (COM & LPT) associated with the UART:

2. Open up Br@y++ terminal and match the settings that you wrote in your program, and the COM Port. Note: Set handshaking (not shown below) to none. Note: Select the COM port that you found in the Device Manager. The Atmel board will show up under USB Serial Port. It is COM4 in this screenshot. (Note that the values are not what you need for your lab.)

3. Select “Connect” on the top left. If everything is installed correctly, you should NOT get a warning, and be ready to transmit and receive data.

4. There are two ways you can transmit data to your XMEGA:
   a. Single Key Press: Place your cursor at the bottom of the Br@y++ terminal in the Transmit section and begin typing:

   b. Send Queued Data: Place your cursor in the white box and type a message; when done press “-> Send” on the right side (not shown below) to send the entire message:

   c. Send Queued Data:

   d. Receive:

5. When Br@y++ terminal receives data, it will be displayed in the receive section, the data can be viewed in ASCII or HEX. We will use ASCII.

B: Delay Loops in ISRs

Normally, interrupt service routines should be as short as possible. Designing a program that explicitly delays the processor while inside an ISR is bad practice because it means the processor is unable to service other interrupt routines (assuming that you haven’t explicitly allowed ISRs to nest/preempt each other, which is also generally discouraged). Better techniques that do not require delaying within the ISR could include additional hardware on the switch, or a software architecture that uses ‘flags’ that are set in the ISR and read in the main loop and processed. A timer system (which we will use in a future lab) could be used to trigger an interrupt after an appropriate delay.

C: Baud Rate v. Hz v. Bits/Second

If you search the term ‘baud’ you will find that it actually means ‘symbols per second’. A symbol is the unit that is being transmitted, so in our case, we are transmitting ‘bits’. Therefore, we can equate baud = bits/second. Because we transmit one bit per cycle and because hertz (Hz) is cycles per second, we can also say that the bits/second = Hz.

Learn more about communication theory in EEL 4514: Communication Systems and Components.
Lab 4: Interrupts and Serial Communication

D: uPAD and USB and Serial
The relevant uPAD schematics for our use of an XMEGA serial port interfacing with USB are shown in Figure A1.

Figure A2 shows a block diagram description of the proper functioning of the circuits shown in Figure A1.

**Figure A1:** uPAD schematics related to USART and USB.

**Figure A2:** uPAD block diagrams related to USART and USB.