EEL 3744 – Spring 2018
Revision **1**

Lab 2: GPIO (LEDs and Switches)

OBJECTIVES

In this lab, you will add the *Switch and LED Backpack* onto the µPAD. You will become familiar with using the I/O ports to control switch circuits and LED circuits. You will also cultivate your programming skills and utilize the DAD/NAD (Analog Discovery board) oscilloscope and logic analyzer functions.

REQUIRED MATERIALS

- Review *Lab Rules and Policies* document
- DAD/NAD board
- µPAD 2.0 Development Board
- Switch and LED Backpack + Schematic
- USB A to B Cable
- Atmel AVR XMEGA AU Manual (8331 and 8385)
- AVR Instruction Set Manual (0856)

PRELAB REQUIREMENTS

REMEMBER

You must adhere to the *Lab Rules and Policies* document for **every** lab. Re-read the document, if necessary.

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 and develop it as a 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 features of Atmel Studio such as breakpoints along with your DAD/NAD board and your prior electrical/computer engineering knowledge to fix any errors. This should occur **BEFORE** you come to lab. Visit a TA or Dr. Schwartz if necessary but come to lab prepared!

USING ATMEL STUDIO EFFICIENTLY

In Atmel Studio you can have multiple assembly files in the **SAME** project. It is highly recommended that you have one project per lab that contains multiple assembly files (one for each part that requires one). This makes your projects much more organized and easy to navigate. Here is a GIF that shows how to add a new file to your project and set it as the **entry file**:

https://mil.ufl.edu/3744/labs/multiple_asm_files.gif

In Atmel Studio, the entry file is the .asm file that is chosen to be assembled.

PART A: PRELIMINARY INFORMATION

All of the more than dozen XMEGA 8-bit ports can be used as general-purpose inputs or outputs (GPIO). In this section, you will ultimately design a program that outputs to the eight LEDs located on your *Switch and LED Backpack*. These LEDs are designed to connect to one of the ports on the XMEGA.

First, you **must** design the LED circuits. You should have learned how to design and construct LED circuits in EEL3701. If necessary, see the below document for a refresher:

http://mil.ufl.edu/3701/docs/hardware_get_started.pdf

1. Design (on paper or computer) eight LED circuits connected to the port meant to connect to the LEDs on the *Switch & LED* Backpack. Make sure you use the correct activation level **AND** label the correct port. See the *Switch and LED backpack* schematic for more details. Include this circuit design into your prelab submission.

To correctly add the *Switch and LED Backpack* (see Figure 1) onto the uPAD, match the J1, J2, J3, and J4 headers with their respective ports. The cutout at the top of the backpack should go over the button on the uPAD.

Figure 1: LED & Switch Backpack Orientation

Remember that before using an I/O port, you first have to configure the **necessary pins within that port** as either inputs or outputs. To configure a pin as an input, the appropriate bit in the **PORTx_DIR** register must be a zero (*cleared*). To configure a pin as an output, it must be a one (*set*).

The configuration registers of the XMEGA can thought of simply as data memory address locations that contain some data. For example, the DIR register for Port F (**PORTF_DIR**) is at address 0x6A0 = 1696_{10}. If you are curious, you can find this information in the *Register Descriptions* document on the website or in the include file listed under "Dependencies" in Atmel Studio's Solution Explorer:

https://mil.ufl.edu/3744/docs/XMEGA/ATxmega128A1U_Definitions.pdf

There are also registers such as DIRSET, DIRCLR, and DIRTGL for each port. Writing to these registers ultimately **modifies** the DIR register. Instead of directly writing to the DIR register, you should always utilize these registers (DIRSET, DIRCLR, or DIRTGL) when you do not intend to alter the pin direction of EVERY pin within a

given port (unless you do not need to configure **ALL** of the pins on the port). For example, if you **ONLY** want to select the data direction of PF6 (Port F, pin 6), without affecting the directions of the other pins on Port F, you should store 0100 0000_2 (0x40) to **PORTF_DIRSET** or to **PORTF_DIRCLR**, depending on the desired direction of the pin (i.e., input or output). This would either set or clear bit 6 in **PORTF_DIR,** without affecting its other bits.

Each port also has an **OUT** register. If a pin is configured as an output, then the **OUT** register is used to control the digital voltage level that is present on the pin. If a bit in the **OUT** register is **set**, then the voltage on the respective pin will be driven **high** (Vcc). If a bit is **cleared**, its voltage will be driven **low** (GND).

Just like the DIR register,when applicable, the OUT register's bits should be modified by using the port's OUTSET, OUTCLR, and OUTTGL.

PART B: SWITCHES AND LEDS

In this part, you will write a program, `lab2b.asm`, that constantly reads the DIP switches on the *Switch and LED backpack* and outputs data that was read to the LEDs.

First, make sure that you have the *Switch and LED backpack* properly connected to your uPAD (see Figure 1). Reference the *Switch and LED backpack* schematic document to determine which ports connect to the switches and LEDs.

Ultimately, this program must do the following **FOREVER**, i.e., within an endless loop:

- Properly initialize the ports to which the switch and LED circuits are connected.
- Read the switch data.
- Output switch data to the LEDs.

1. Write the program (`lab2b.asm`), as described above.

PART C: DELAYS

In this part, you will write a program, `lab2c.asm`, that toggles an output pin at a specific rate using software delays. Software delays are useful for many things: blinking LEDs, waiting before performing certain actions, etc. (Later this semester, we will make more intelligent, non-software, delay functions.)

1. Create a subroutine (`DELAY_10ms`) that does nothing but delay 10 ms. Since your XMEGA's clock runs at 2 MHz by default (until we change this in Lab 3), it would be a good assumption to say that each instruction takes about 0.5 μs = 1/(2 MHz).

NOTE: Chances are, the first time you write your subroutine, it won't be close to 10 ms. This is okay; you will test it and adjust it when you write the rest of your program. Put the actual time that you measure the first time you write the subroutine into your lab report.

2. Put the delay time measured during your first attempt into the appendix of your lab report.
3. Use your `DELAY_10ms` subroutine to write a program, `lab2c.asm`, that toggles an output pin at a rate of 5 Hz. Remember that frequency is the reciprocal of period, i.e., f = 1/T, where T is the period between the toggling of the pin. Make sure you toggle an **accessible** pin! (Check the schematics if necessary.)

The output will be a square wave with a 50% duty cycle, as shown in Figure 2, where X is 100 ms. (The LED will be on for half the period and off for the other half.)

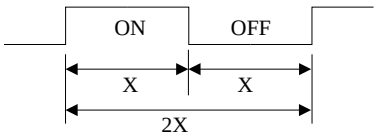

Figure 2: Blinking output, where X =100 ms

4. Observe this waveform using the oscilloscope of your DAD/NAD. Use the ***Measurements*** tool under View (or press Ctrl+M) to display the values of the waveform's average frequency and average period. Make a note of the average period of the generated waveform and adjust your `DELAY_10ms` subroutine until it is accurate to within ±3%.
5. Once your DELAY_10ms subroutine is within 3% of 10 ms, take a screenshot of your *Waveforms* window and submit it in the **appendix** of your pre-lab report (as stated in the *Lab Rules and Policies*, part 16, item h). (Remember that screenshots must **always** be place within the appendix!) The screenshot **MUST** display the waveform, the average period, and the average frequency. You will **not get credit** for the screenshot if it is not legible.
6. Now, make a subroutine that can delay a select multiple of 10 ms. Use your corrected `DELAY_10ms` subroutine to create a **separate** subroutine called `DELAYx10ms`, where "x" will be a number passed into the `DELAYx10ms` subroutine within a resister, e.g., R16. The delay should be the number in the register multiplied by 10 ms. It is okay if the largest allowable value is 127, giving a maximum delay of 1.27 s, but a maximum delay of 2.55 s is preferred (with an allowable value up to 255). The minimum delay should be 10 ms (remember to push/pop any registers used).

PART D: MAKING A GAME

In this part, you will make a game, `lab2d.asm` that utilizes all the previous parts: inputs (switches), outputs (LEDs), and your delay subroutines.

To summarize the game, two LEDs will be shifting towards and then away from the center from each side. Each row of

Figure 3 shows a frame of the successive patterns of the eight LEDs. This pattern must repeats. The goal of the game is to press one of the tactile switch buttons (S2) when the LEDs in the middle (bits 4 and 3) are on. When the game is won, the LEDs should stop shifting, and **ONLY** the **green** LED (connected to Port D) should turn on. If S2 is pressed at the wrong time, i.e., NOT when 4 and 3 are on, the LEDs should stop shifting, and **ONLY** the **red** LED (connected to Port D) should turn on.

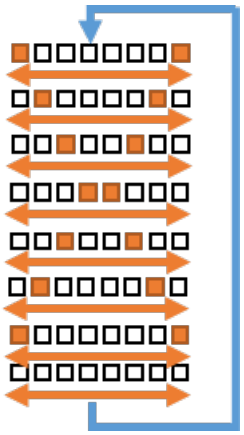

Figure 3: Game animation pattern

Remember that the tactile switches, unlike the DIP switches, are **NOT** the only thing connected to their port. Therefore, when you are reading them you must either mask away the other 6 bits by using some basic logical operations, or use must use another type of instruction, e.g., sbrs or sbrc. There are many ways to accomplish this, though just make sure you are reading only the necessary pins.

See the *µPAD v2.0* schematic for details about the RGB LEDs on Port D. When configuring Port D, make sure that you **ONLY** change the direction/output value for the pins you are using. Do **NOT** modify the other pins on Port D.

Here are further game specifics:

- The two LEDs should shift **one** position at a time, i.e., [7,0] LEDs should be on, then [6,1], etc. They should shift every 100 ms.
- You should NOT be able to hold S2 to win the game. If you press S2, and the [4:3] LEDs aren't on, the game should stop, and the red LED should turn on.
- If you press S2 while the [4:3] LEDs are on, the game should stop, and the green LED should turn on.
- You should be able to reset the game by pressing the S1 button. You should be able to reset the game **ONLY** after winning OR losing.

1. Write the program (`lab2d.asm`), as specified.
2. Use your DAD/NAD and the Logic Analyzer feature within *Waveforms* to analyze the animation pattern created, on Port C (the J5 Header on the *Switch and LED Backpack*).

ADDITIONAL NOTES:

When writing any programs, it is very useful to create equates to define commonly used values. For example:

```
.equ S2 = 0x08
```

If you add this to your .asm file you can then use the term "S2" throughout your program without having to remember that the S2 tactile switch is connected to pin 3 on its port. You can do this for many other things such as bit values:

```
.equ BIT3 = 0x08
```

We could then replace the previous equate with the following, as long as BIT3's definition appears first in program code:

```
.equ S2 = BIT3
```

Note: You will need to demonstrate understanding how to analyze the pattern signal using the Logic Analyzer in Waveforms on Port C (the J5 Header on the *Switch and LED Backpack*)**.**

LAB PROCEDURE

- Demonstrate your solution to Part B executing on your board.
- Demonstrate your solution Part D executing on your board, and then use the DAD/NAD's **Logic Analyzer** within *Waveforms* to demonstrate the specified game animation.
  - If you cannot demonstrate Part D, then demonstrate Part C.

REMINDER OF LAB POLICY

Please re-read the *Lab Rules & Policies* so that you are sure of the deliverables (both on paper and through Canvas) that are due prior to the start of your lab.