OBJECTIVES

In this lab you will add the *Switch and LED Backpack* onto the uPAD. You will become familiar with using the I/O functions ports to control switch circuits and LED circuits. You will also exercise 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
- uPAD 1.4 Development Board
- Switch and LED Backpack + Schematic
- USB A to B Cable
- Atmel AVR XMEGA AU Manual (8331 and 8385)
- AVR Instruction Set Manual

PRELAB REQUIREMENTS

REMEMBER

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

GETTING STARTED

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

READ the *Switch and LED Backpack* schematic thoroughly to see which orientation the Backpack will go on the uPAD 1.4 Board.

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!

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 to output to 8 LEDs connected 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

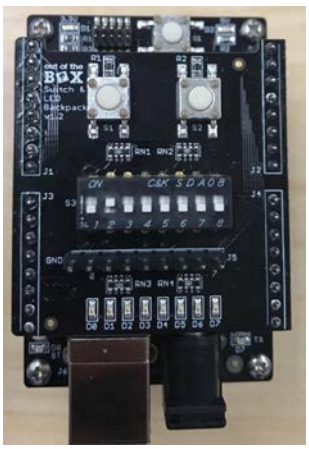

Figure 1: LED & Switch Backpack Orientation

Design (on paper or computer) eight LED circuits connected to a single port. 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 to your pre-lab 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.

Remember that before using a GPIO, you have to first configure the **pins that you need** 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 be simply thought of as 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 **modifies** the DIR register. You should always utilize these registers when you do not need to configure **ALL** of the pins on a 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 could 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 the 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 its pin will be **high** (Vcc). If a bit is **cleared**, its voltage will be driven **low** (GND).

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

PART A: SWITCHES AND LEDS

In this part, you will write a program, `lab1a.asm`, that constantly reads the DIP switches on the *Switch and LED backpack* and outputs the data to the LEDs.

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

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

You will need to do the following:
1. Properly initialize the ports to which the switch and LED circuits are connected.
2. Read the switch data.
3. Output switch data to the LEDs.

PART B: DELAYS

In this part you will write a program, `lab1b.asm`, that toggles an output pin at a specific rate using software delays. Delays are useful for many things: blinking LEDs, waiting before performing certain actions, etc.

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 assembled.

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 2), 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.

Use your `DELAY_10ms` subroutine to write a program, `lab1b.asm`, that toggles an output pin at a rate of 50 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 a pin that you have access to! Check the schematics if necessary.

The output is a square wave with a 50% duty cycle, as shown in Figure 2, where X is 10 ms. (The LED will be on for half the period and off for the other half.) Observe this waveform using the oscilloscope of your DAD/NAD.

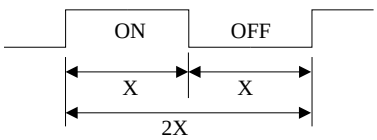

Figure 2: Blinking output, where X =10 ms

Use the ***Measurements*** tool under View (or press Ctrl+M) and display the values of the average frequency and the average period of the waveform. Make a note of the average period of the generated waveform, and adjust your `DELAY_10ms` subroutine until it is accurate to within ±3%.

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 you pre-lab report (as stated in the *Lab Rules and Policies*, part 14, item h). 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.

Now, make a subroutine that can delay a select multiple of 10 milliseconds. Use your corrected `DELAY_10ms` subroutine to create a subroutine called `DELAYx10ms`, where "x" will be a number passed into the `DELAYx10ms` subroutine in a resister, e.g., R16. The delay should be the number in the register × 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 also allowable (with an allowable value up to 255). The minimum delay should be 10 ms (remember to push/pop any registers used).

PART C: MAKING A GAME

In this part, you will make a game, `lab1c.asm` that utilizes all of the previous parts: inputs (switches), outputs (LEDs), as well as your delay subroutines.

To summarize the game, two LEDs will be shifting from the most significant bits [7:6] to the least significant bits [1:0]. This pattern will repeat. The goal of the game is to press one of the tactile switch buttons (S2) when the LEDs in the middle [4:3] 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, ONLY the **red** LED (connected to Port D) should turn on.

See the *uPAD v1.4* schematic for details about the RGB LEDs on Port D. When configuring Port D, make sure you **ONLY** change the direction/output value for the pins you are using. Do **NOT** modify the other pins on Port D.

Here are the specifics:
1. The two LEDs should shift **one** position at a time, i.e., [7:6] should be on, then [6:5], etc. They should shift every **80 ms**.
2. 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.
3. If you press S2 while the [4:3] LEDs are on, the game should stop and the green LED should turn on.
4. You should be able to reset the game by pressing the S1 button. You should be able to reset the game after winning OR losing.

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 Part A on your board.
- Demonstrate your Part C, and use the Logic Analyzer of your DAD/NAD board to see all 8 of the LEDs.
  - If you can't demonstrate Part C, then demonstrate Part B.

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.