Lab 1: Input/Output Interfacing and Traffic Light Controller, FSMs

Goals

1. Familiarity with IAR workbench software development environment
2. Programming the EK-TM4C123 board
3. Learning basics of general-purpose input and output (GPIO)
4. Performing simple digital I/O input (interfacing push button) and output (interfacing LED)
5. Understanding the power of pointers in C and how to use them in addressing GPIO registers
6. Using the datasheet to extract information necessary for addressing the LaunchPad parts.
7. Understand debouncing
8. Designing a finite state machine (FSM) controller

What you need for the lab

1. The EK-TM4C123 Launchpad (http://www.ti.com/tool/EK-TM4C123GXL)
2. TM4C123 data sheet (https://canvas.uw.edu/courses/1205180/files/folder/Ek-TM4C123GXL?preview=49165887)
3. IAR workbench or other IDE
4. IDE installation guide (https://canvas.uw.edu/courses/1205180/files/folder/Ek-TM4C123GXL?preview=49716060)
5. Tutorial video (https://www.youtube.com/watch?v=cLbaSFKdAho)
   Since we use different versions of the IDE in this video, please consult the above IDE installation guide if you find any differences.
6. Oscilloscope and some cables
7. LEDs (https://learn.adafruit.com/all-about-leds/the-led-datasheet)
8. Push buttons
   (https://www.alps.com/prod/info/E/HTML/Tact/SnapIn/SKHH/SKHHAKA010.html)
9. Debouncing (https://canvas.uw.edu/courses/1205180/files/folder/labs?preview=49719211)
10. Lab write up specifications
    (https://canvas.uw.edu/courses/1205180/files/folder/labs?preview=49165850)

The IDE

1. IAR workbench is installed on the lab machines in the lab and the installation instructions (https://canvas.uw.edu/courses/1205180/files/folder/Ek-TM4C123GXL?preview=49716060) are posted on Canvas. If you want to download the software on your laptop please note that IAR works only on Windows OS so if your computer runs Linux or Mac OS, you will need to install a virtual machine. You are free to use other IDEs such as Keil uVision or Code Composer but make sure that the TM4C123 is supported (the board is supported by Keil 4.7 but not Keil 5).
2. The free version of IAR limits your data and code to 32 KB, which is not an issue for the lab assignments in this course.

TM4C123GXL board

Figure 1shows an overview of the different parts of the Launchpad, refer to the user manual (posted on canvas) for more information on the board's features

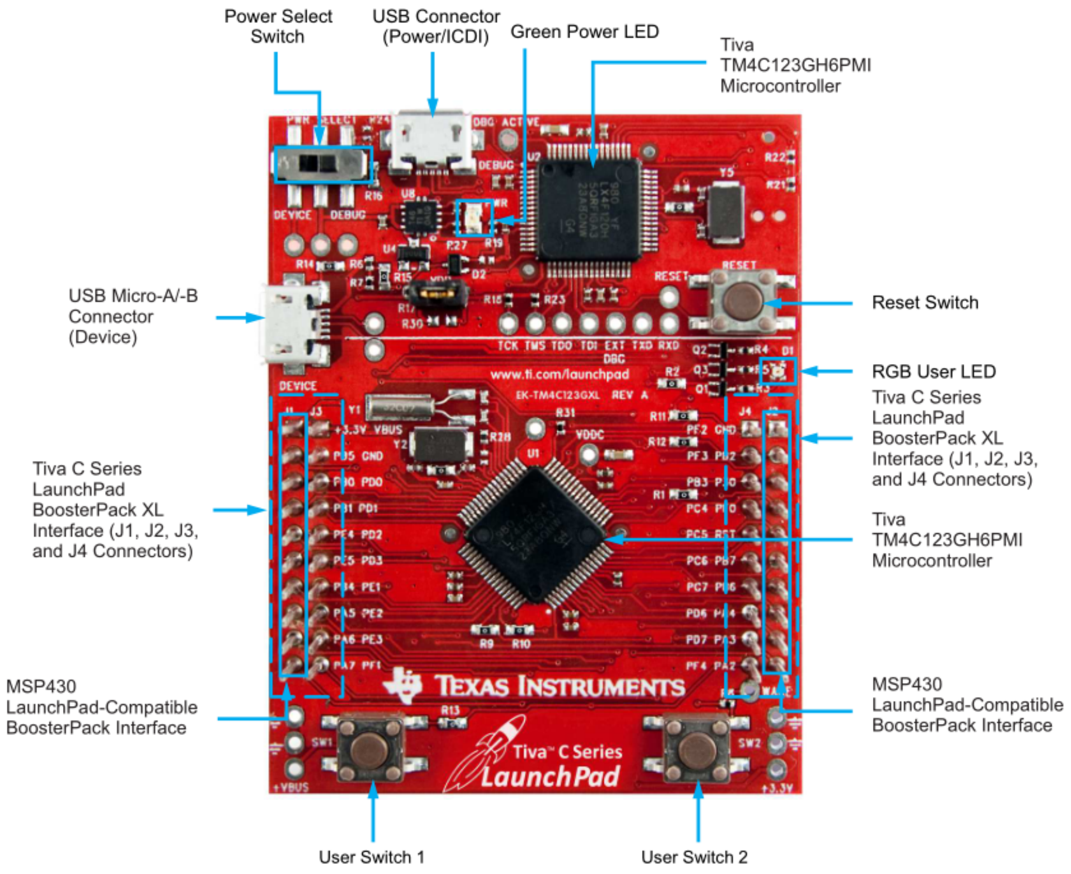

Power Select Switch
USB Connector (Power/ICDI)
Green Power LED
Tiva TM4C123GH6PMI Microcontroller
USB Micro-A/-B Connector (Device)
Reset Switch
RGB User LED
Tiva C Series LaunchPad BoosterPack XL Interface (J1, J2, J3, and J4 Connectors)
Tiva C Series LaunchPad BoosterPack XL Interface (J1, J2, J3, and J4 Connectors)
Tiva TM4C123GH6PMI Microcontroller
MSP430 LaunchPad-Compatible BoosterPack Interface
MSP430 LaunchPad-Compatible BoosterPack Interface
User Switch 1
User Switch 2

Figure 1 TM4C123G LaunchPad

The RGB user LED consists of three LEDs Red, Green, and Blue and are connected to the processor through Port F (as shown in Figure 2 and Figure 3)

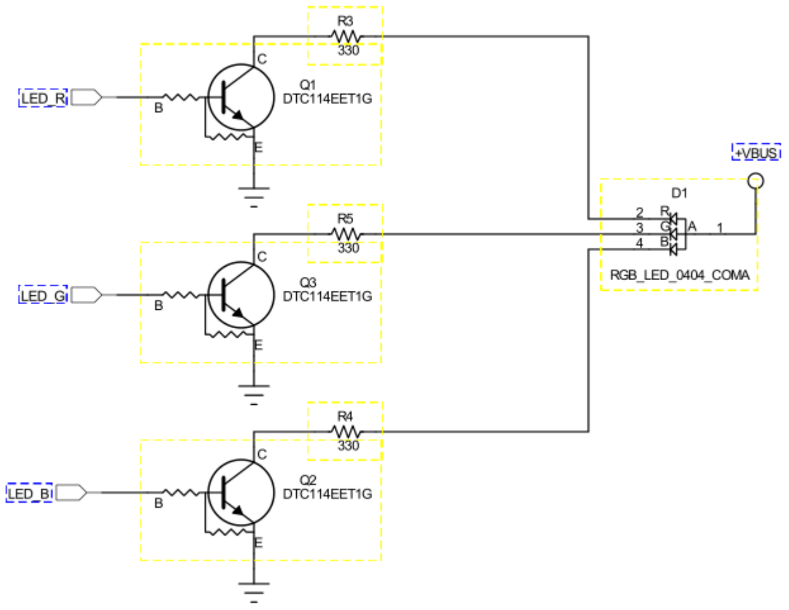


Figure 2 Three LEDs

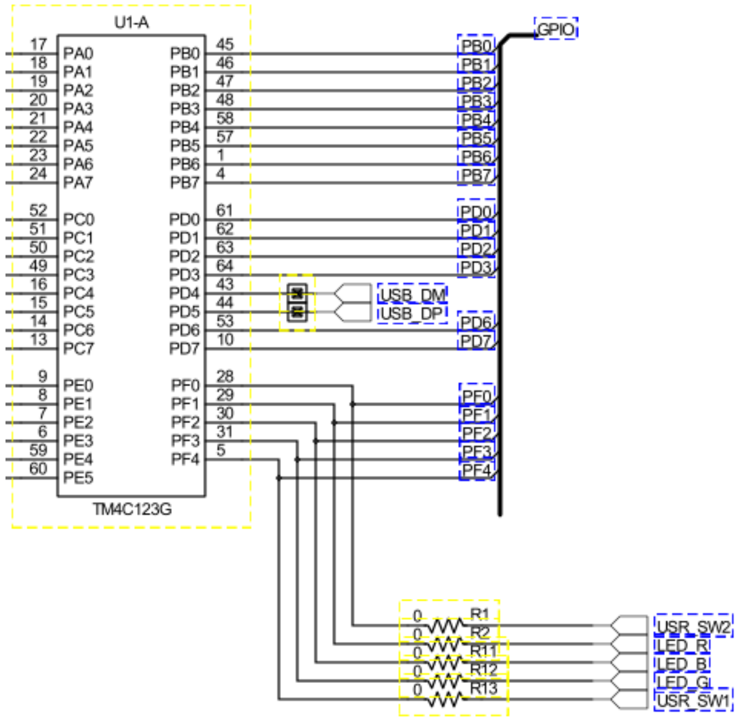


Figure 3 PortF

To turn on an LED, the following operations must be performed:

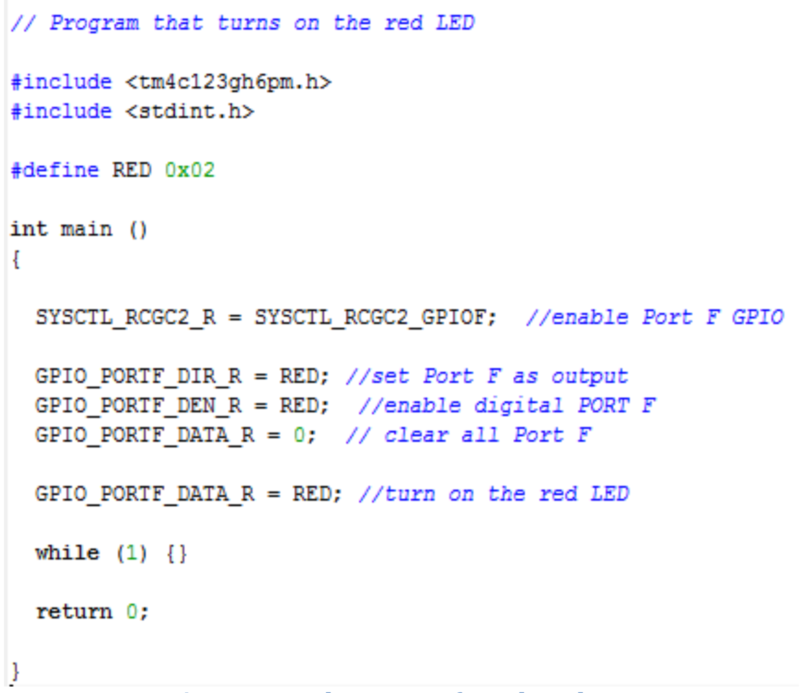
1. Configure the **RCGCGPIO** register to turn on the GPIO port F by enabling its clock. By default, the clock to all peripherals, including GPIO ports, are turned off in order to improve the energy efficiency.
2. Configure the **GPIODEN** register to set Port F as digital (by default, the mode of all GPIO pin is **analog**)
3. Configure the **GPIODIR** register to set the PF1, PF2, and PF3 pins as outputs
4. Configure the **GPIODATA** register to set the output ***value*** of the corresponding GPIO pin to ***1***. Port F is one byte (8 bits) wide so to turn on the red LED (as an example), PF1 must be set to 1 so the value of the port is 0x02 (hexadecimal).

The same steps apply for reading user switches except that GPIODIR register should be configured to set pins PF0 and PF4 as inputs. Please note that PF0 has special considerations. It is configured as a GPIO by default but is locked and can be reprogrammed by unlocking the pin in the **GPIOLOCK** register and un-committing it by setting the **GPIOCR** register. Therefore you need to configure GPIOLOCK, GPIOCR, and GPIOPUR as well.

Section A. Input/Output Interfacing

Example Program

The program shown in figure 4 is a program that turns on the red LED.



```
// Program that turns on the red LED

#include <tm4c123gh6pm.h>
#include <stdint.h>

#define RED 0x02

int main ()
{

  SYSCTL_RCGC2_R = SYSCTL_RCGC2_GPIOF;  //enable Port F GPIO

  GPIO_PORTF_DIR_R = RED; //set Port F as output
  GPIO_PORTF_DEN_R = RED;  //enable digital PORT F
  GPIO_PORTF_DATA_R = 0;  // clear all Port F

  GPIO_PORTF_DATA_R = RED; //turn on the red LED

  while (1) {}

  return 0;

}
```

Figure 4 sample program for onboard LED

Note that it includes the tm4c123gh6pm.h file which is included in a library specific to the TM4C123GXL board. This can be downloaded from http://www.ti.com/tool/sw-ek-tm4c123gxl (available on canvas too). The "installing header files" document on canvas refers to installing the header files for the TIVA series and shows how to set the path in IAR. You can follow the same steps for setting the path of the TM4C123GXL library.

Section A Required Tasks:

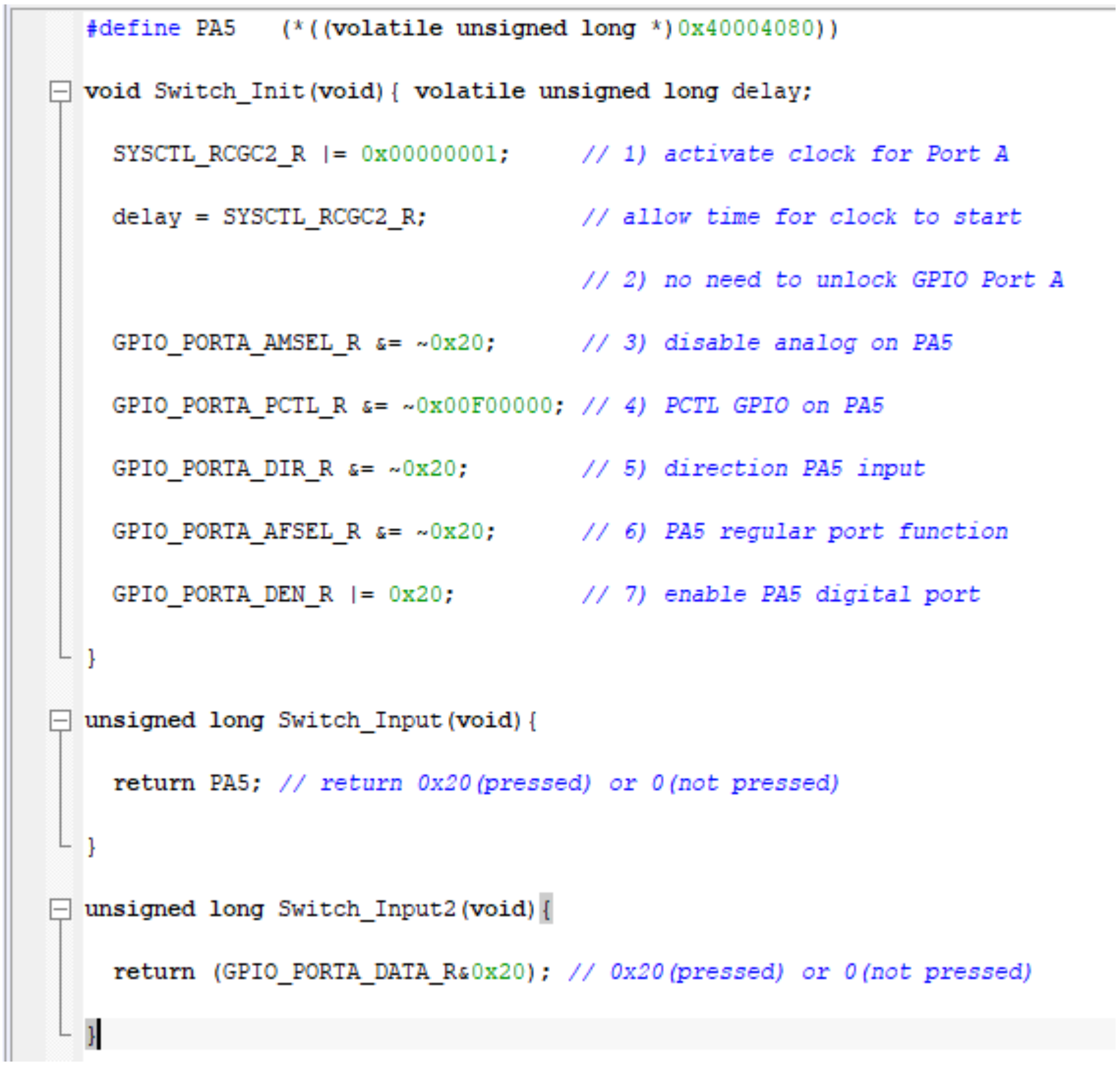
1. Try the program of figure 4 to turn on the red LED on your board.
2. Modify the program to exclude using macros definitions and eliminate the use of the TIVA and Launchpad libraries. You need to directly address the registers based on their base and offset addresses as described in the datasheet.
3. Expand the program to light up all LEDs. They should be flashing continuously.
4. Modify the program to define and use your own macros for better readability. You do not need to include the LaunchPad libraries.
5. Write another C program to incorporate the user input from the switches. You should turn on the red light when user switch 1 is pressed and turn on the green light when the user switch 2 is pressed.
6. Use an oscilloscope to show the voltage output of the LED and the voltage signal of the pin connected to the onboard pushbutton. Find out the latency between the onboard buttons pressed and LED lighting up.

Section B. Traffic Light Controller, FSMs

Button Debouncing

Check the correct pinout for the push buttons. Wire one pin of a button to the ground, and the other pin to PA5 in the TIVA. Note here, PA5 is a positive-logic button input, where 1 means pressed, and 0 means not pressed. You may use the breadboard and jumper wires from EE 215.

Please try the program of figure 5 to interface a switch to TIVA.



```
#define PA5   (*((volatile unsigned long *)0x40004080))
void Switch_Init(void){ volatile unsigned long delay;
  SYSCTL_RCGC2_R |= 0x00000001;     // 1) activate clock for Port A
  delay = SYSCTL_RCGC2_R;           // allow time for clock to start
                                    // 2) no need to unlock GPIO Port A
  GPIO_PORTA_AMSEL_R &= ~0x20;      // 3) disable analog on PA5
  GPIO_PORTA_PCTL_R &= ~0x00F00000; // 4) PCTL GPIO on PA5
  GPIO_PORTA_DIR_R &= ~0x20;        // 5) direction PA5 input
  GPIO_PORTA_AFSEL_R &= ~0x20;      // 6) PA5 regular port function
  GPIO_PORTA_DEN_R |= 0x20;         // 7) enable PA5 digital port
}
unsigned long Switch_Input(void){
  return PA5; // return 0x20(pressed) or 0(not pressed)
}
unsigned long Switch_Input2(void){
  return (GPIO_PORTA_DATA_R&0x20); // 0x20(pressed) or 0(not pressed)
}
```

Figure 5 sample code for push button

Commonly, we would encounter a bouncing issue when a push button is used. When it happens, the GPIO pins would treat a single press as multiple presses. There are multiple ways to denounce the button, either by hardware approach or software approach. See the debouncing document in the Canvas for more details (https://canvas.uw.edu/courses/1205180/files/folder/labs?preview=49719211).

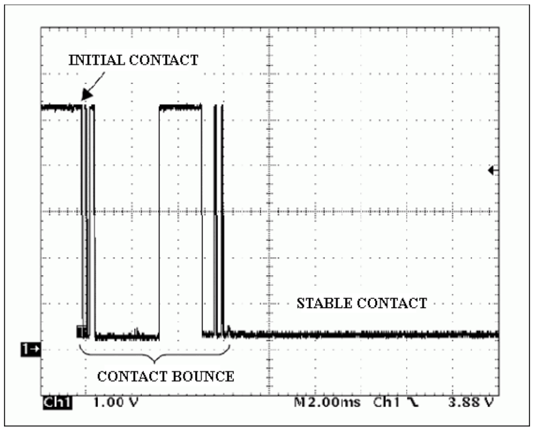


Figure 5 push button bounce

A common approach is using a delay. Essentially, when you press or not press a button, the contact is not immediately read. We just wait a few moments before reading the new value. Please try this code below:

```
//   SysCtlDelay(7000000); // uncomment me if you need debouncing
```

LED

LEDs emit light when the current passes through them. LEDs have polarity, the current flows from the anode (shorter lead) to the cathode (longer lead) to activate them. The LEDs won't be damaged if you reversed the direction, but they won't be turned on.

The brightness of a LED depends on the current flows through it. If the current is more than 8mA, it is not safe to connect it directly to a GPIO pin on the board. We suggest you use a 220Ω current limiting resistor, which connects to the cathode of the LEDs. The hardware configuration of this LED is shown in Figure 6. Note +5 V is VBUS on TIVA. Please try the code above to interface an LED. You need to modify the pin from PA2 to PE0 to match the hardware configuration. Alternatively, you could connect the pin 1 of U3a7406 (or any generic 7406 gates found in the kit) to PA2 on TIVA to match the sample code.

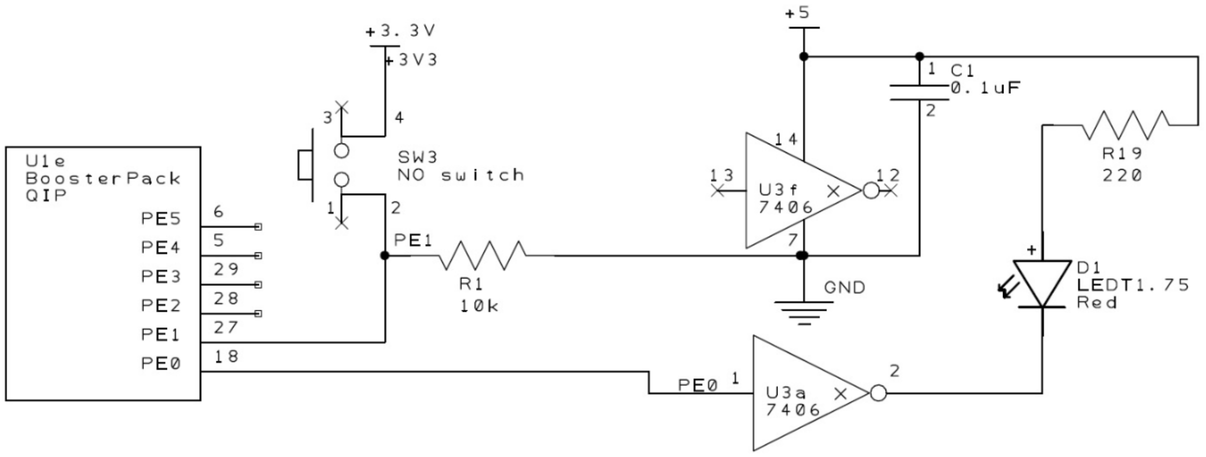
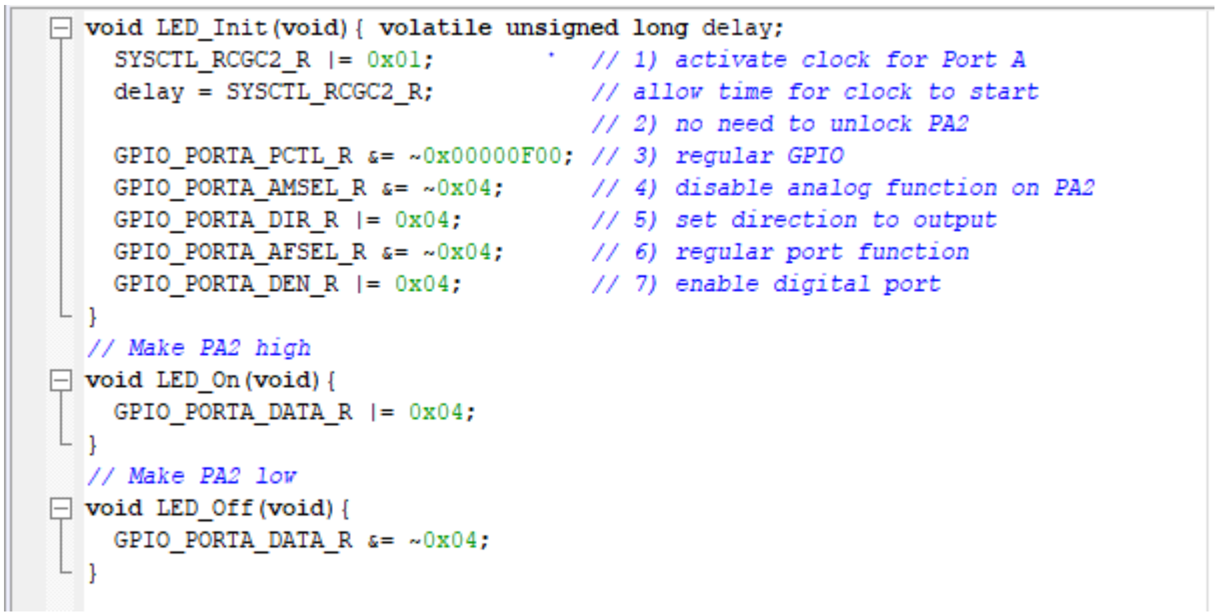


Figure 6 LED hardware setup



```
void LED_Init(void){ volatile unsigned long delay;
  SYSCTL_RCGC2_R |= 0x01;          // 1) activate clock for Port A
  delay = SYSCTL_RCGC2_R;          // allow time for clock to start
                                   // 2) no need to unlock PA2
  GPIO_PORTA_PCTL_R &= ~0x00000F00; // 3) regular GPIO
  GPIO_PORTA_AMSEL_R &= ~0x04;     // 4) disable analog function on PA2
  GPIO_PORTA_DIR_R |= 0x04;        // 5) set direction to output
  GPIO_PORTA_AFSEL_R &= ~0x04;     // 6) regular port function
  GPIO_PORTA_DEN_R |= 0x04;        // 7) enable digital port
}
// Make PA2 high
void LED_On(void){
  GPIO_PORTA_DATA_R |= 0x04;
}
// Make PA2 low
void LED_Off(void){
  GPIO_PORTA_DATA_R &= ~0x04;
}
```

Figure 7 sample code for LEDs

Traffic Light Controller

We provide three different LEDs and two push buttons in this lab. One red LED (for stop), one green LED (for go), and one yellow LED (for warn). The suggested pins are:

LEDs/ switch	Possibility 1	Possibility 2	Possibility 3
Button for start/stop	PA2	PB2	PE0
Button for passenger	PA3	PC4	PE1
Red LED (Stop)	PA7	PB5	PE5
Yellow LED (Warn)	PA6	PB4	PE4
Green LED (Go)	PA5	PB3	PE3

Finite State Machine Design

Design a finite state machine that implements a traffic light system. It should include two push buttons as an input, and three LEDs are an output. You may need to purchase one push button from EE store if you can only find one button in your lab kie.

Inputs:

1) Button for start/stop

   The user would press this button to start and stop the system. The system will always start from the go stage (where the green LED is on).

2) Button for passenger

   Once this button is pressed, it indicates there is a passenger that would like to cross the street. If the current stage is stopped (where the red LED is on), nothing will happen. If the current stage is going (where the green LED is on), the current stage will change to warn stage (where the yellow LED is on), then the current stage will change to stop stage (where the red LED is on) to allow the passenger safely across the street.

Outputs:

1) Red LED (stop)
   When the current stage is stop, the red LED is on, and all other LEDs are off.

2) Yellow LED (warn)
   When the current stage is warn, the yellow LED is on, and all other LEDs are off. It only happens when the button for the passenger is pressed.

3) Green LED (go)
   When the current stage is go, the green LED is on, and all other LEDs are off.

Section B Required Tasks:

1. Try the program to interface the button.
2. Try the program for the button debouncing.
3. Try the program for the LED.
4. Design the FSM for the traffic light system.
5. Draw the finite state machine system diagram.
6. Draw the schematics for the traffic light control system.

Good Coding Practices (Courtesy of Y.Zhu, author of "Embedded systems with ARM Cortex-M")

Program comments are used to improve code readability and to assist in debugging and maintenance. A general principle is "Structure and documents your program the way you wish other programmers would" (McCann, 1997).

The book titled "The Elements of Programming Style" by Brian Kernighan and P. J. Plauger gives good advice for beginners.

1. **Format your code well.** Make sure it's easy to read and understand. Comment where needed but don't comment obvious things it makes the code harder to read. If editing someone else's code, format consistent with the original author.

2. Every program you write that you intend to keep around for more than a couple of hours ought to have documentation in it. Don't talk yourself into putting off the documentation. A program that is perfectly clear today is clear only because you just wrote it. Put it away for a few months, and it will most likely take you a while to figure out what it does and how it does it. If it takes you a while to figure it out, how long would it take someone else to figure it out?

3. **Write Clearly -** don't be too clever **-** don't sacrifice clarity for efficiency.
4. **Don’t over a comment.** Use comments only when necessary.
5. Format a program to help the reader understand it. **Always Beautify Code.**
6. Say what you mean, **simply and directly**.
7. **Don't patch bad code** - rewrite it.
8. **Make sure comments and code agree.**
9. Don't just echo code in comments **- make every comment meaningful.**
10. **Don't comment bad code** - rewrite it.
11. **The single most important factor in style is consistency.** The eye is drawn to something that "doesn't fit," and these should be reserved for things that are actually different.

Deliverables:

1. A lab report that includes: abstract, introduction, Procedures, results, and conclusion sections. The report should address the required section A and B tasks and should include screenshots of your programs as well as pictures of oscilloscope output. Including the traffic light controller system schematic and finite state machine diagram in the report. Please check the lab write up specifications on the Canvas (https://canvas.uw.edu/courses/1205180/files/folder/labs?preview=49165850).
2. All source files with proper comments.
3. Demonstrate the output to your TA.
4. Upload the report and source files to the canvas. One submission is expected per team.