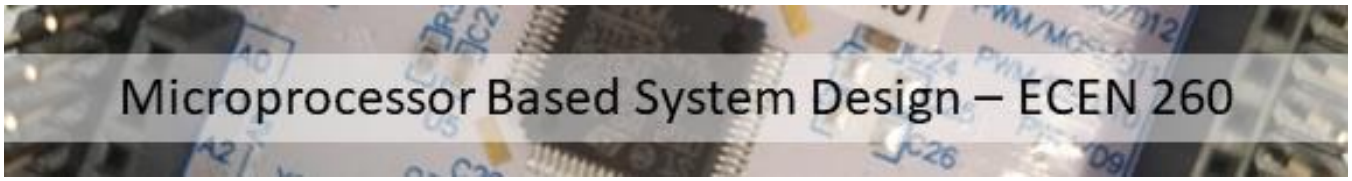


# ECEN 260 Lab 06
# *Device Drivers*

## Lab Objectives

- Learn how to create your own device drivers.
- Participate in code peer review. (This will happen during class next Monday.)
- Test each other's device drivers. (This will also happen during class next Monday.)

## Required Parts (same as last lab)

- 1 Nucleo-L476RG development board
- 1 USB cable
- 1 breadboard (protoboard)
- 3 reed switches (each has two parts)
- 3-4 LEDs (you choose the colors)
- 3-4 resistors
- 1-2 push buttons
- jumper wires

## Overview

For this lab, you will be creating your own GPIO drivers for the STM324L476RG on the Nucleo-L476RG board. Then, on Monday you will be peer reviewing and testing each other's drivers.

**Functions and header files:** When 'C' code is compiled, the complier needs to know what the input parameters and the return parameter of a function are before that function is called so that it can set up the assembly code correctly. We do this with a function *prototype*. Function prototypes should be declared *before* main. When you have code outside of main.c, it is common to put the function prototypes in a *header* (.h) file and include that file in main.c. Then, the function *implementations*–the code inside the function–will be written in a separate .c file of the same name as the header file.

## Lab Requirements

This lab has two parts: (1) write the code for the driver functions and submit them *by* Saturday, and (2) participate with a lab partner in a peer review of each other's driver functions *during* Monday's class and then resubmit your improved drivers, a demo video, and your peer review report.

## Part 1: Writing your Driver Code

This link will provide you with the complete drivers.h file from class on Wednesday. It includes the macros and function prototypes discussed in class. You will write the code implementations for each of those driver functions in a drivers.c file. Next week, you will test your drivers with this main.c file, which includes a adapted version of last week's alarm code using the driver functions instead of the

control registers. By Saturday, submit just your drivers.c file. You will later resubmit that that assignment with an updated version of your drivers after doing the peer review on Monday.

*To Complete by Saturday*

**Driver code:** Create a "drivers.c" code file that includes the implementations of the following driver functions:

- `void GPIOA_ClkEnable(void);`
- `void GPIOB_ClkEnable(void);`
- `void GPIOC_ClkEnable(void);`
- `void GPIO_Mode(Port* GPIOx, uint32_t pin, uint32_t mode);`
- `void GPIO_Pull(Port* GPIOx, uint32_t pin, uint32_t pull);`
- `void GPIO_TogglePin(Port* GPIOx, uint32_t pin);`
- `void GPIO_WritePin(Port* GPIOx, uint32_t pin, uint32_t state);`
- `uint32_t GPIO_ReadPin(Port* GPIOx, uint32_t pin);`

**Read function:** All the functions should return nothing (void), except for the `GPIO_ReadPin` function, which should return a `uint32_t`. For this function, you can decide whether to:

(A) return a copy of the IDR with all bits reset except for the bit for the corresponding pin, or
(B) return that value, but on the bit 0 position so the return value is either '0' or '1'.

Example: Assume the IDR = 0001 0010 1101 1010, and you are reading pin 4.
Option A: Return 0000 0000 0001 0000
Option B: Return 0000 0000 0000 0001
You should accurately document what your function does in the comments of the function.

**Write function:** The `GPIO_WritePin` function should use the third input parameter, `state`, to determine if the pin should be set or reset. If `state` is zero, then the pin should be reset. If the state is *anything* besides zero, the pin should be set.

**Default values:** Your driver functions should not assume any default values. For example, in the PUPDR, don't assume the bits start as zeros. In the MODER, don't assume the bits start as ones. Your functions should explicitly reset any bit that should be low and set any bit that should be high.

**Changing multiple bits:** If a function needs to change two bits in a control register, it should change both simultaneously to avoid any unintended intermediate state change. If the two changes can't be performed in a single operation (such as setting one bit and resetting the other), then the register value should be copied to a temporary variable where the changes can be made separately, and then the final value can be placed back into the register so both changes take affect simultaneously.

**Testing your drivers before Monday:** After completing your "drivers.c" file, perform a smaller test of your code by creating a main routine that turns LED2 on when B1 is pressed and off when B1 is not pressed. Then, during Monday's class, you will test it with the adapted alarm code.

**If you don't finish before Saturday:** Still come to Monday's class.

## Demonstration of Correct Operation – First Submission

**By Saturday:** Submit just a copy of your drivers.c file to the "Lab 06 Demo & Code" assignment. You will later do a second submission on Tuesday of next week after completing Part 2 (below) during class on Monday.

## Part 2: Participating in a Code Review on Monday (June 3)

*To Complete During Monday's class*

**During Monday's Class:**

1. Get with a lab partner and email each other your "drivers.c" file. (To preserve formatting, it is better to find the file and attach it to the email than to copy and paste the code into the email.)
2. Peer review each other's drivers.c file according to the instructions provided below.
3. Test the code by making a new project with their drivers.c file, the provided drivers.h file and this adapted main.c alarm code, which was changed to use the drivers.
4. Send your "Peer Review Report" to your lab partner so they can improve their code.
5. Update your own code based on your lab partner's peer review of your code.
6. Test your own drivers file in a new project with the full alarm code. Run all your previous tests.

**If you miss the Monday:** Email your instructor ASAP so that we can arrange a way for you to still participate in the peer review process.

## Peer Review Instructions

For the peer review, get a copy of your lab partner's drivers.c code. Make a new project with their code. If you are working in a group of 3, then Person A should review Person B's code; Person B should review Person C's code; and Person C should review Person A's code.

Create a peer review report in Word of your lab partner's drivers.c file. Include a section in your report for each of the following:

A. A copy of **their code.**
B. Your review of **the structure** of their code. Is the formatting professional? Are the comments useful for understanding the flow of the program (and not just restating what each individual line of code does)?
C. Your review of **the correctness** of their code. Are there any typos or bugs? Do the configuration functions work in all situations? Did they follow all guidelines on page 2 of these instructions?
D. Your results of **running their code** with the adapted alarm code. Does it execute as expected? If not, identify the unexpected behavior.
E. Any other **suggestions** for improving the code.

Then, send a copy of your peer review report back to your lab partner by Monday night.

## Demonstration of Correct Operation – Second Submission

**By Tuesday (6/4):** After Monday's peer review, make any desired changes to your main.c. Once you believe your drivers are written correctly, test them with the adapted alarm code. Record a video of your alarm working with your drivers. Then, resubmit to the same "Lab 06 Demo & Code" assignment with: (1) the video demo, (2) your updated copy of your drivers.c file – even if no changes were needed, and (3) a copy of your Peer Review Report that you did of your lab partner's code.

## Lab Report Instructions

The Lab Report for this Lab will be combined with next week's Lab 7 and won't be due until the end of Week 7. Starting with this lab report, your lab reports should now always include a *schematic*. A schematic is a symbolic diagram showing how the components are connected.

For this report, your schematic should show each of the four LEDs as outputs with current-limiting resistors, each of the two push buttons (arm/disarm) as inputs with pull resistors, and each of the three reed switches (front/back/window) as inputs with pull resistors. You may use any schematic tool to create your schematic. If you'd like, you may use the partial schematic in Figure 1 on the next page as a starting point. It was created with www.circuit-diagram.org. You may upload this partial Lab6_schematic.cddx file from I-Learn to that website. Then, you can complete the schematic on that website. This partial schematic assumes you are using Button 1 (PC13) as your "Arm" button and LED 2 (PA5) as your "Armed Status" indicator.

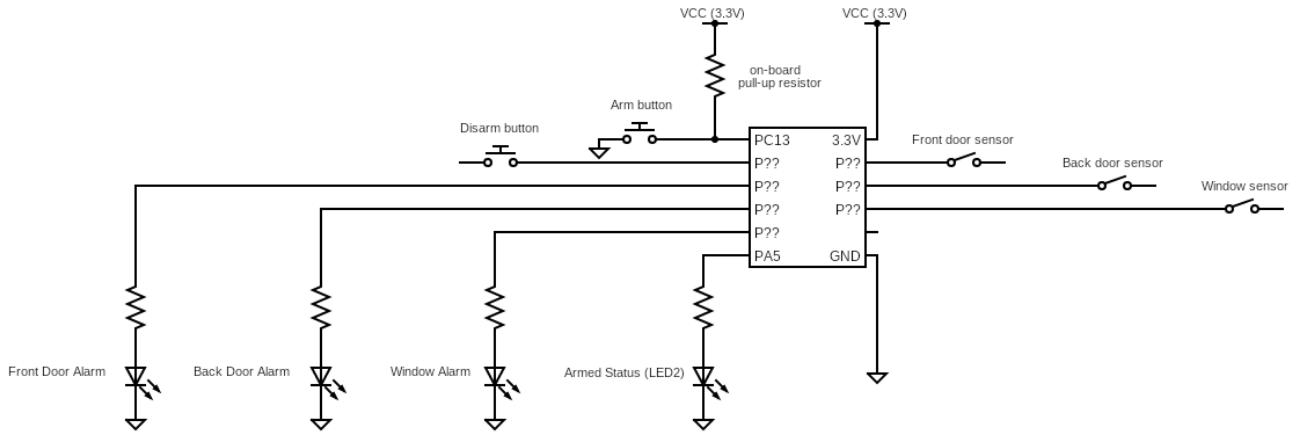


Figure 1: Partial Schematic (you must complete it)

Depending on if you choose to connect a particular button or switch to a high voltage 

or to low voltage  , that will determine if you need a pull-down resistor or pull-up resistor, respectively. You must add a "ground" or "VCC" connection to the other buttons and switches to indicate if they are wired as active-low or active-high, respectively, and then add a pull-up or pull-down resistor as appropriate for each. Also change the pin names on the chip icon from P?? to the appropriate names.

Include the schematic in your report using the Lab Report Template provided in the "Lab 06 & 07 Report" assignment in I-Learn, and then submit your report there.