

# ECE 362 Experiment 3: General Purpose I/O

## 1.0 Introduction

*In this experiment, you will learn how to attach simple input devices (pushbuttons) and simple output devices (LEDs) to an STM32 development board. You will learn how to configure the General Purpose Input/Output (GPIO) subsystems. You will read values from and write values to these pins.*

Step		Points
4.2	Prelab	30 points
6.1	Enabling the Clock to Ports	10 points
6.2	Configuring Pins as Outputs	10 points
6.3	Turning Output Pin On	10 points
6.4	Reading Input Pins	10 points
6.5	Wiring	10 points
6.6	Sequencing	10 points
6.7	Material Return	Points contingent on return
6.8	Post lab submission	10 points
	<b>Total</b>	<b>100 points</b>

**All lab points are contingent on submission of code completed in lab and return of materials.**

## 2.0 Objectives

1. Connect LEDs and buttons to the STM32F0 GPIO pins.
2. Configure the GPIO systems.
3. Read values from and write values to the GPIO pins.
4. Attach external circuitry to the GPIO pins.

## 3.0 Equipment and Software

1. Standard ECE362 Software Toolchain (Installation instructions available in Experiment 0: Getting Started)
2. STM32F0-DISCOVERY development board (For procurement instructions, see Experiment 0: Getting Started)

## 4.0 Prelab

### 4.1 Read section 5 of this document for background information

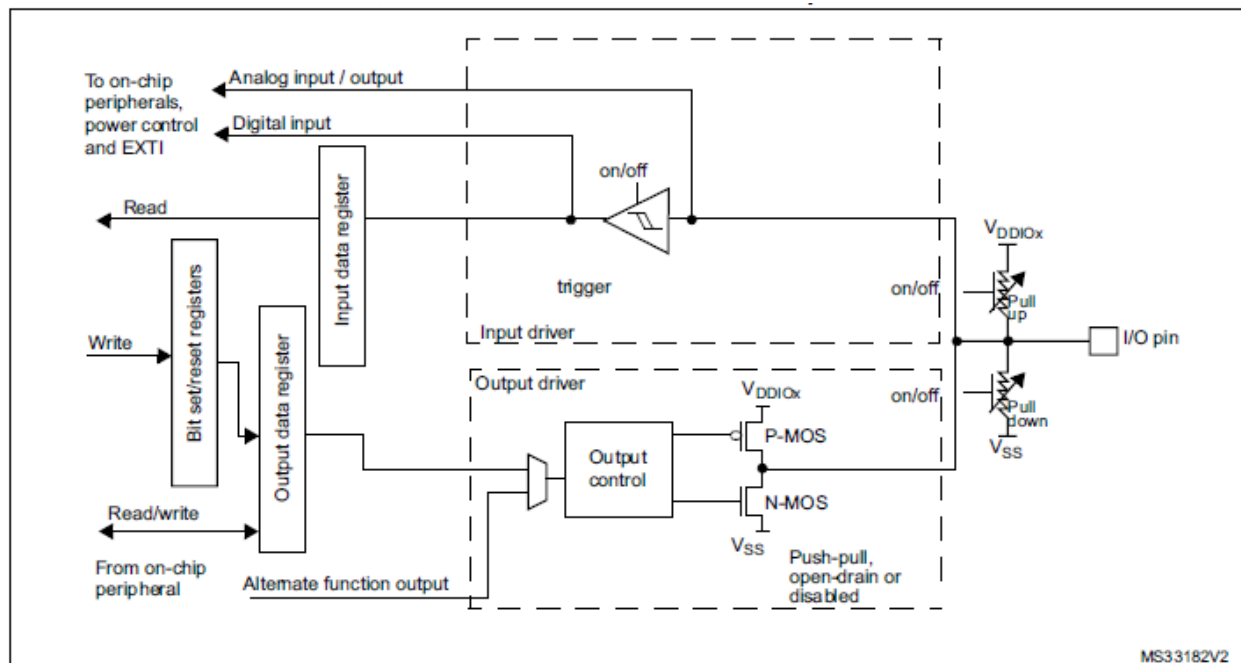
### 4.2 Complete the prelab questions on the course website

## 5.0 Background

Two steps are needed to configure I/O pins for use. First, the Reset & Clock Control subsystem must be configured to enable a port for use. This is done by turning on the specific bit in the AHBENR (Advanced High-Performance Bus Enable Register). In doing so, be sure not to turn OFF important bits such as the SRAMEN bit. Use your lecture notes to determine how to do this. For this lab, in particular, you should enable Ports B and C by ORing the appropriate values into the AHBENR register.

Second, an enabled port must have its pins configured as either inputs or outputs (or some other functions that we will use in future labs). For this lab, we will use Port B pins as inputs and Port C pins as outputs. For both ports there are some pins whose function you should not modify. In particular, you should not change PA13, PA14, PC13, PC14, or PC15 since these are used for the External Oscillator, Debugging, and Programming interfaces. Instead, use code from your lecture notes to learn how to work around those.

A functional diagram for an STM32F0 GPIO port is shown in Figure 1, below:



**Figure 1. GPIO Pin Functional Diagram**

## 6.0 Experiment

For many lab experiments, we will provide a template project that you may incorporate into System Workbench to set up the software environment to build and test your code. This environment includes an *autotest* module that will evaluate the code you write. In this way, you can test your code to make sure it works before calling a teaching assistant to evaluate it. The process of setting up the environment is one you can go through yourself, or you can simply use the pre-packaged project.

### To use the pre-packaged project:

- Download the Lab03\_Template.zip from the lab web page.
- Unpack the zip file on your computer. You may change the name of the resulting directory to anything you want.
- Invoke System Workbench.
- Click on **File >> Open Projects from File System...** and select the directory that you unzipped.
- Fill in the main.s file and build your project.

### To build your own environment:

- Invoke System Workbench and create a project as you normally would.
- Create a main.s file and fill it with the skeleton file provided on the lab web page.
- Download the autotest.o compiled module and put it in your project's **src** directory.
- Select the **Project >> Properties** menu to open a dialog window...
  - Select the **C/C++ Build >> Settings** item on the left.
    - The dialog will change to show "Tool Setting" and other tabs.
    - Under Tool Settings, select **MCU GCC Linker >> Miscellaneous**
    - On the right will be a box titled "Other objects". Click on the green "+" icon.
      - In the new dialog that appears, select "Workspace..." and navigate to the name of your project, then the src directory, and select the autotest.o file. The result should appear in the "Add file path" dialog as:  
`${workspace_loc}/${ProjName}/src/autotest.o}`
  - Press OK for all of the dialogs.

Upon doing either of these actions, you will be able to link the code in your main.s file with the subroutines provided for you in autotest.o.

**These instructions will generally be the same for all assignments where we provide an autotest module for you.**

## 6.1 Enabling the Clock to Ports

Using the main.s template file provided for this lab, fill in the assembly language procedure for **enable\_ports**. This is should **enable the RCC clock** for both **Port C and Port B** without disabling other bits such as SRAMEN. Recall *what a subroutine is, how it needs to be written in assembly, do not forget to push and pop the right register(s)*.

To test your assembly subroutine, uncomment the line “*bl test\_enable\_ports*” in main.s. A working **enable\_ports** subroutine will blink the green LED once per second. Demonstrate your completed work to your TA.

Hint: When debugging your code, you may consult the “I/O Registers” tab in the Monitor window. This tab provides a hierarchical listing of all of the various peripherals utilized by the microcontroller and their sub-registers. By default their values are not shown; double clicking on a given entry will cause the debugger to track the values of that entry and its lower elements. Use this to confirm that the various registers you are writing to are taking on the values which you expect. An example of this is shown in Figure 2, below:

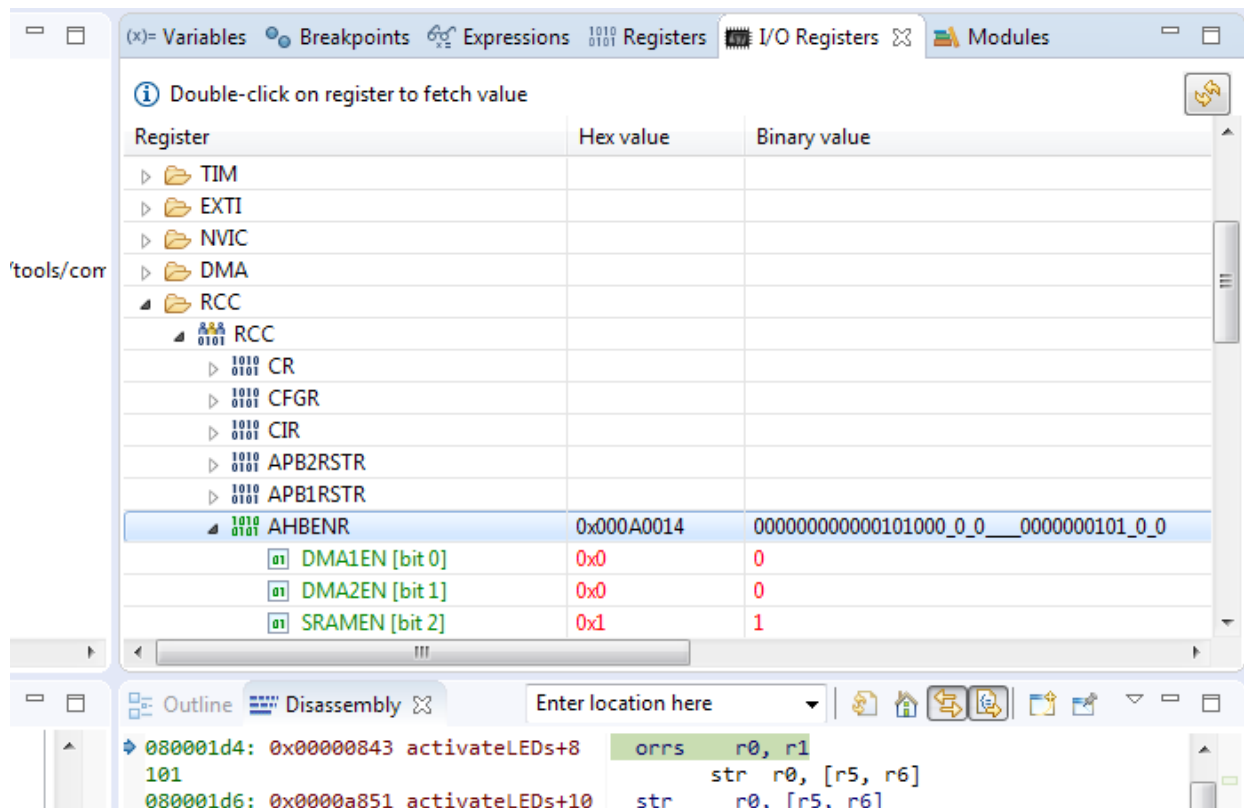


Figure 2. I/O Registers View

## 6.2 Configuring Pins as Outputs

For this section you will need to complete the subroutine ***port\_c\_output*** in main.s so that it configures **pins 0, 1, 2 and 3** of **Port C** as **outputs**. DO NOT change the configuration for pins 13, 14, and 15. *Once again, make sure to push and pop the right register(s).*

To test your assembly subroutine, uncomment the line “*bl test\_port\_c\_output*” in main.s. A working ***port\_c\_output*** subroutine will blink the blue LED once per second. Demonstrate your completed work to your TA.

## 6.3 Turning Output Pin On

For this section you will need to complete the subroutines ***setpin*** and ***clrpin*** in main.s. Each of these subroutines expects a pointer to the GPIOx register array to be passed as the first parameter (register R0). The second parameter (register R1) is the bit number to be set (by ***setpin***) or cleared (by ***clrpin***). The ***setpin*** subroutine sets the corresponding bit (given by register r1) in the GPIOx\_ODR register. And ***clrpin*** subroutine clears the corresponding bit (given by register r1) in the GPIOx\_ODR register.

Once your subroutines are implemented, uncomment “*bl test\_set\_clrpin*”. A working subroutine will allow the code to turn the blue and green LEDs on and off, together, once per second. Demonstrate your completed work to your TA.

**6.3.1 Name two I/O registers other than the ODR that can be used to set the bit values of ODR. Type your answer in your post lab write up.**

## 6.4 Reading Input Pins

Complete the subroutine ***getpin*** in main.s. This function accepts the GPIOx base address as parameter 1 (register R0) and a bit number as parameter 2 (register R1). It returns the value (in register r0) in GPIOx\_IDR at the bit specified by r1. That is the subroutine should return 0x1 if the pin is high or 0x0 if the pin is low.

Once your subroutine is implemented, uncomment “*bl test\_getpin*”. A working subroutine will allow the code to turn on both the blue and green LEDs only when the user pushbutton is pressed. Demonstrate your completed work to your TA.

**Continues on next page**

## 6.5 Wiring

**Unplug the power from your STM32 first, THEN** wire your circuit as shown in Fig 3. When you have completed your wiring, show your circuit to your TA. Ensure that you connect the resistors for the buttons to the 3V (3.3V) power. In other words, **SHOW YOUR WIRING TO YOUR TA BEFORE YOU PLUG IT BACK IN.** To test your wiring uncomment “*bl test\_wiring*”. The right wiring will turn on the LED’s in a binary sequence, and will stop when any one of the wired buttons is pressed and restarts as soon as one it’s released.

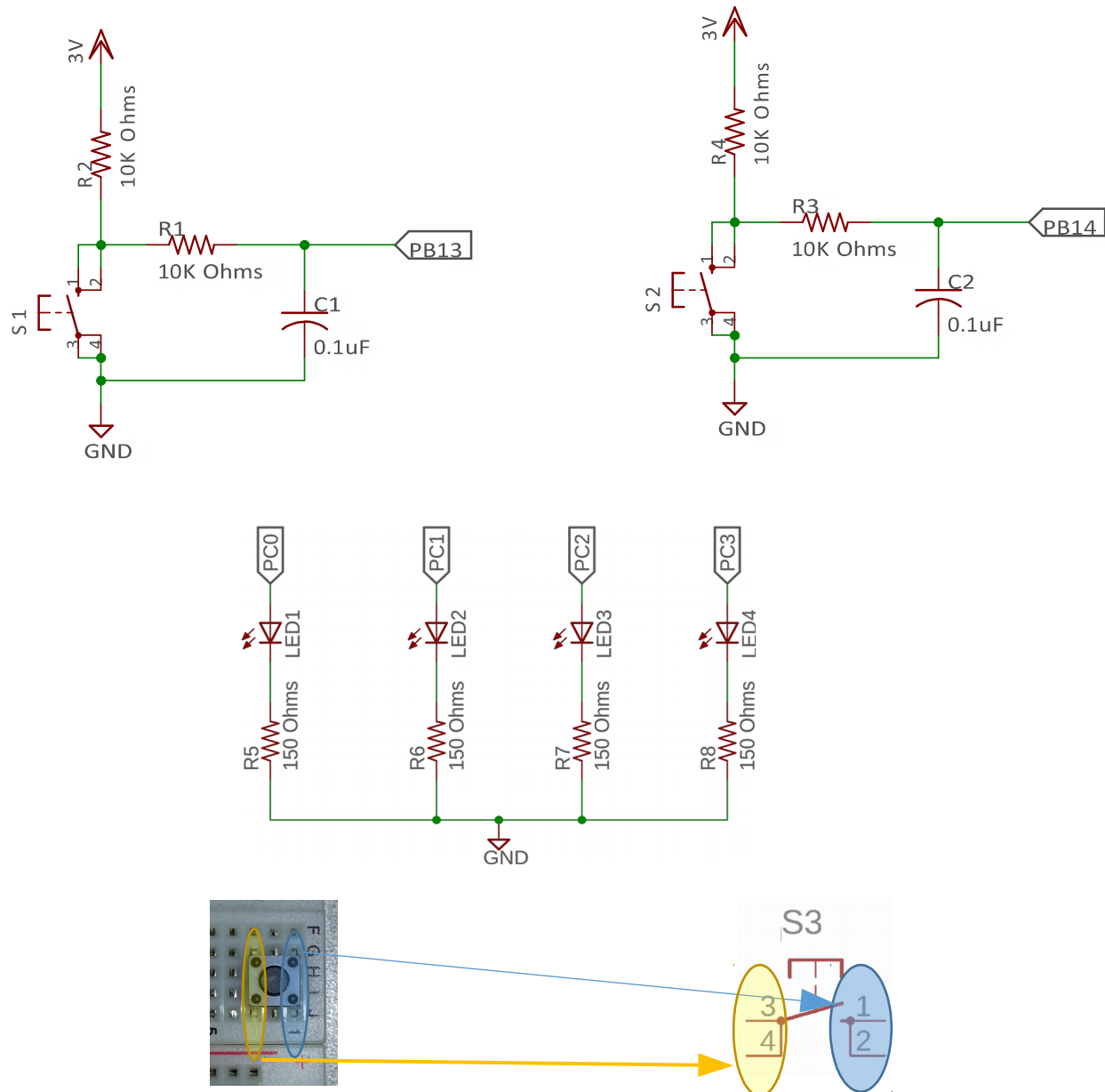


Figure 3. Schematic for wiring LEDs and push buttons

**6.5.1 Each I/O pin has an internal pull up, typically 40K Ohms. We could get rid of R2 and R4. Name the register that enables this internal pull up. Type your answer in your post lab write up.**

## 6.6 Sequencing

For this section you will need to complete the *seq\_led* subroutine. This subroutine should cycle/sequence through the LED's connected to PC0, PC1, PC2 and PC3. The subroutine must also be able to cycle in both directions, given the direction parameter.

This subroutine accepts two parameters (R0 and R1), R0 is a pointer to the GPIOx base address, R1 is the direction for the cycle/sequence. The *seq\_led* subroutine will perform the following operations:

- 1) Read the current content of GPIOx\_ODR
- 2) If R1 = 1
  - a. Left shift the content by 1
  - b. Check if value exceeds 8
  - c. If so set it to 0x1                      // Logic for cycling though the LED's
- 3) Else (R1 != 1)
  - a. Right shift the content by 1
  - b. Check if value is 0
  - c. If so set it to 0x8                      // Logic for cycling though the LED's
- 4) Store the new value in ODR

Once you complete this section, uncomment "*bl run\_seq*". A working subroutine will allow the code to turn on the next LED when you press on button, and turn on the previously lit LED when you press the other. The LEDs will cycle in a sequential fashion. Demonstrate your completed work to your TA.

**Refer to the wiring diagram for the lab and answer the following questions**

**6.6.1 Remove R1 and C1 and connect pins 1 and 2 of switch S1 directly to pin PB13, run the sequencing routine. Do the LEDs sequence correctly when you press the wired push button S1? Type your observation in your post lab write up.**

**6.6.2 What kind of signal filtering is circuit of R1 and C1 effectively performing? Type your answer in your post lab write up.**

**6.6.3 What is the logic level of the push button? Your answer must be in the following format: “Pressed: logic <X>, Released: logic <Y>” replace <X> and <Y> with the appropriate logic levels (logic 1 and logic 0). Type your answer in your post lab write up.**

## **6.7 Material Return**

Return the lab components provided to you. You will not receive any credit unless you return the components.

## **6.8 Complete post lab and submit code**

Submit your code and complete the post lab on the website. The portal for post-lab will close 10 minutes after your scheduled lab.