ECE 153B – Winter 2024

Sensor and Peripheral Interface Design

Lab 1 – Interfacing with LED using GPIO

Deadline: Jan 18, 2024, 10:00 PM

Objectives

1. Get familiar with the Keil μ Vision software development environment
2. Learn the basics of GPIO configurations: input/output, push-pull/open-drain, pull-up/down, GPIO speeds
3. Program GPIO registers to perform simple digital input (interfacing push button) and output (interfacing LED)
4. Understand polling I/O (busy waiting)

Lab Overview

Use polling method to set/toggle the green LED for user pushbutton clicks.

Grading

Part	Weight
Functionality	65 %
Checkoff Questions	20 %
Turn-In Questions	15 %

You must submit your code (src folder) and answers to turn-in questions (as PDF) in Gradescope by the specified deadline. In the week following the submission on Gradescope, you will demo your lab to the TA. Checkoff will also involve some questions from the TA.

Code submission should be done in group. Each group should create only 1 submission with both members added to the submission using GradeScope's team submission feature. Be sure to properly indent and comment your code to receive full credit.

Please note that we take the Honor Code very seriously, do not copy the code from others.

Contents

1 Necessary Supplies

- STM32 Nucleo-64 Board
- Type A Male to Mini B USB Cable

2 Introduction

2.1 GPIO – General Purpose Input/Output

On the STM32 Nucleo board, there are 8 GPIO ports $(A, B, C, D, \ldots, H)$ with 16 pins each. Figure 1 shows a diagram of a single GPIO pin. Each of the GPIO pins can be configured in the software as an output (push-pull or open-drain), as an input (with or without pull-up/down), or as a peripheral alternative function. Each GPIO port $x \in \{A, B, C, D, \ldots, H\}$ has

- Four 32 bit configuration registers
  - `MODER` (mode register)
  - `OTYPER` (output type register)
  - `OSPEEDR` (output speed register)
  - `PUPDR` (pull-up/pull-down register)
- Two 32 bit data registers
  - `IDR` (input data register)
  - `ODR` (output data register)
- One 32 bit set/reset register `BSRR`
- One 32 bit locking register `LCKR`
- Two 32 bit alternative function registers
  - `AFRH` (alternative function high register)
  - `AFRL` (alternative function low register)

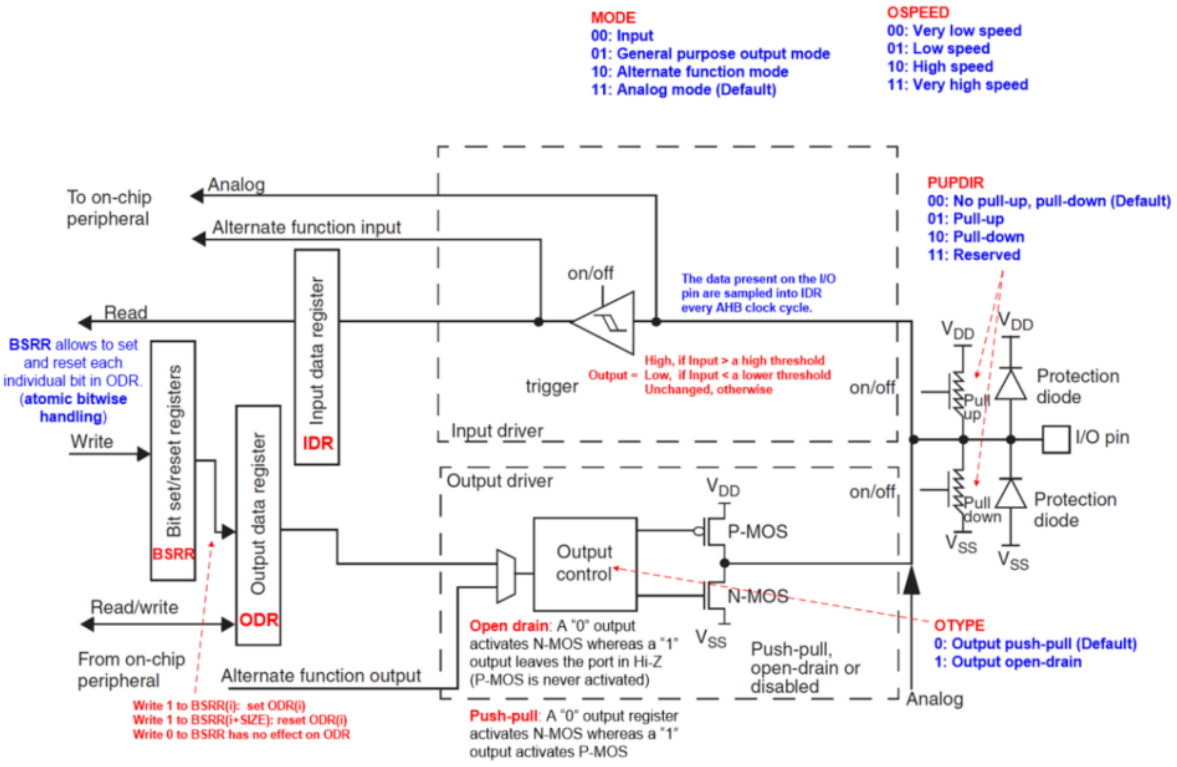

Figure 1: GPIO Pin

2.2 The LED and the Button on the Board

There is one user-controllable green LED (LD2) on the Nucleo board (see Figure 2). The LED LD2 is connected to **PA5** (GPIO Port A Pin 5). To light an LED, the software must perform at least the following three operations:

1. Enable the clock of the corresponding GPIO port. (Recall that all peripheral clocks are turned off by default to improve energy efficiency.)
2. Set the **mode** of the corresponding GPIO pin to **output**. (By default, the GPIO pin mode is **analog**.)
3. Set the push-pull/open-drain setting for the GPIO pins to **push-pull**.
4. Set the pull-up/pull-down setting for the GPIO pins to **no pull-up and no pull-down**.
5. Set the **output value** of the corresponding GPIO pin to `1`. (When the output value of a GPIO pin is `1`, the voltage on the GPIO pin is 3.3 V. On the other hand, when the output value of a GPIO pin is `0`, the voltage on the GPIO pin is 0 V.)

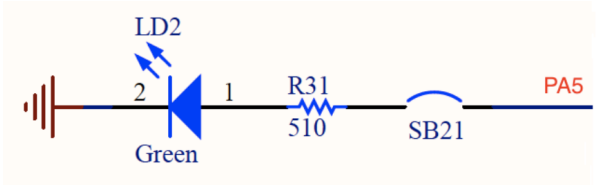

Figure 2: LED on the Nucleo Board

There is a blue user button on the Nucleo board. The button is connected to the GPIO pin PC13 (GPIO Port C Pin 13). Furthermore, a capacitor and resistor are connected to each GPIO pin for *hardware debouncing* (see Figure 3).

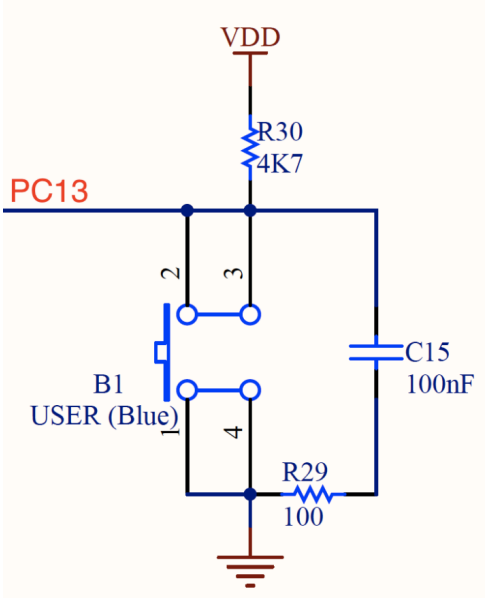

Figure 3: Hardware Debouncing on the User Button

3 Lab Exercise - Polling

In this part of the lab, you will toggle the green LED for alternate user button-clicks. For this part of the lab, you are required to use **polling** (waiting while constantly checking to see whether an event occurs) to implement this function.

The code you will write involves writing certain bits to different registers. To do this, we will use a technique called **masking**. Masking is a technique that lets us toggle, set, or reset specific bits of a word (4 bytes) while keeping the remaining bits unchanged.

Given a `mask` and a target variable `word`, we can perform the following operations using bitwise operators in C.

```
word |= mask;  // Set bits in word specified by mask
word &= ~mask; // Reset bits in word specified by mask
word ^= mask;  // Toggle bits in word specified by mask
```

Consider a simple example. Let variable `word` be 1 byte and let's say that we want to set bit 2 of the variable. We can do this with the following line of code

```
word |= 4;
```

Recall that $a \mid 0 = a$ and $a \mid 1 = 1$. Thus, using the OR operation with `mask` $= 4 =$ `0b0100` lets us set bit 2 without modifying the rest of the bits in the variable `word`. Note that

```
word = 4;
```

is not the correct approach because bits 0, 1, and 3 are reset in addition to setting bit 2.

In practice, we should use macro name instead of numbers to perform masking, for the sake of better readability and easier debugging. Macro names are predefined constants provided in header file. They are equivalent to numer literals, but conveys a meaning in their name. For example, when you see `GPIO_ODR_OD4`, you know that this macro is for the register `GPIOx->ODR`, and it represents the output of pin 4 in the corresponding GPIO.

3.1 Setting up the GPIO

Next, we need to determine the masks that we need for several registers so that we can perform different tasks such as configuration of different GPIO pins. Use the following steps to initialize/configure the registers for this part of the lab. Use the provided tables to help you determine what the mask should be (in hexadecimal).

Note: The register names and many macros are defined in the file `stm32l476xx.h`. You should be referring to this file to determine what registers you must modify (and what masks to apply).

3.2 Enable the Clock of GPIO Ports A (for Green LED) and Port C (for User Button)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AHB2ENR														RNGEN		AESEN			ADCEN	OTGFSEN					GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
Mask																																

AHB2ENR Mask (Macro Name) = _____

AHB2ENR Masking Operation:_____

3.3 Pin Initialization for Green LED (PA5)

(a) **Set the mode of PA5 to Output**
GPIO Mode: Input (00), Output (01), Alternative Function (10), Analog (11 – default)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER	MODE R15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]		MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
Mask																																

GPIO MODER Mask (Macro Names) = _____

GPIO A MODER Masking Operation:_____

(b) **Set the output type of PA 5 as Push-Pull**
Output Type: Push-Pull (0, reset), Open-Drain (1)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OTYPER	Reserved																OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
Mask																																

GPIO A OTYPER Mask (Macro Name) = _____

GPIO A OTYPER Masking Operation: _____

(c) **Set PA5 to No Pull-Up, No Pull-Down**
PUPD Type: No PUPD (00, reset), Pull-Up (01), Pull-Down (10), Reserved (11)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR	PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]		PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
Mask																																

GPIO A PUPDR Mask (Macro Name) = _____

GPIO A PUPDR Masking Operation: _____

3.4 Pin Initialization for User pushbutton (PC13)

(a) **Set mode of pin PC13 to Input**
GPIO Mode: Input (00), Output (01), Alternative Function (10), Analog (11 – default)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER	MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]		MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
Mask																																

(b) **Set PC13 to no pull-up and no pull-down**
PUPD Type: No PUPD (00, reset), Pull-Up (01), Pull-Down (10), Reserved (11)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR	PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]		PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
Mask																																

4 Checkoff Requirements

After completing initialization of all necessary GPIO pins, you need to continuously poll to check for button presses. Use the same masking technique to implement the features: Switch the green LED from ON to OFF or OFF to ON when button is clicked.

5 Turn-in Questions

Turn in these questions with your code submission. Please do not use the predefined masks in your answers.

1. What should my mask be if I want to clear the mode bits of pin PB11?

2. Write the line of code you would use to set PC7 to output mode (Assume the output register is already cleared for you)

3. Write the line of code you would use to read input data from PC3

6 Appendix

There are two major types of clocks: **system clock** and **peripheral clock**.

6.1 System Clock

To meet the requirement of performance and energy-efficiency for different applications, the processor core can be driven by four different clock sources including **High Speed Internal** (HSI) oscillator clock, **High Speed External** (HSE) oscillator clock, **Phase Locked Loop** (PLL) clock, and **Multispeed Internal** (MSI) oscillator clock. A faster clock provides better performance but usually consumes more power.

6.2 Peripheral Clock

All peripherals must be clocked to function. However, *the clocks of all peripherals are turned off by default to reduce power consumption.*

Figure 4 shows the clock tree of **STM32L476RGTx**, the processor used on the Nucleo board. This particular diagram shows the configuration of a 80 MHz SYSCLK and 11.2941 MHz SAI (SAI is not used in this lab). The clock sources in the domain of **Advanced High-Performance Bus** (AHB), low speed **Advanced Peripheral Bus 1** (APB1), and high speed **Advanced Peripheral Bus 2** (APB2) can be switched on or off independently when not in use. The software can select various clock sources and scaling factors to achieve the desired

clock speed depending on the application's needs.

The following definition of `void System_Clock_Init()` uses the 16 MHz HSI as the input to the PLL clock. Appropriate scaling factors have been selected to achieve the maximum allowed clock speed (80 MHz).

```
void System_Clock_Init(void) {
  // ...

  // Enable the HSI oscillator
  RCC->CR |= RCC_CR_HSION;
  while(!(RCC->CR & RCC_CR_HSIRDY));

  RCC->CR &= ~RCC_CR_PLLON;
  while((RCC->CR & RCC_CR_PLLRDY) == RCC_CR_PLLRDY);

  // Select PLL as the clock source
  RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLSRC;
  RCC->PLLCFGR |= RCC_PLLCFGR_PLLSRC_HSI; // 00 - No clock, 01 - MSI,
                                          // 10 - HSI, 11 - HSE

  // Make PLL 80 MHz
  // f(VCO clk) = f(PLL clk input) * PLLN / PLLM = 16 MHz * 20 / 2 = 160 MHz
  // f(PLL_R) = f(VCO clk) / PLLR = 160 MHz / 2 = 80 MHz
  RCC->PLLCFGR = (RCC->PLLCFGR & ~RCC_PLLCFGR_PLLN) | 20U << 8;
  RCC->PLLCFGR = (RCC->PLLCFGR & ~RCC_PLLCFGR_PLLM) | 1U << 4;
  RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLR; // 00 - PLLR = 2, 01 - PLLR = 4,
                                     // 10 - PLLR = 6, 11 - PLLR = 8
  RCC->PLLCFGR |= RCC_PLLCFGR_PLLREN; // Enable Main PLLCLK output
  RCC->CR |= RCC_CR_PLLON;
  while(!(RCC->CR & RCC_CR_PLLRDY));

  // Set PLL as the system clock
  RCC->CFGR &= ~RCC_CFGR_SW;
  RCC->CFGR |= RCC_CFGR_SW_PLL; // 00 - MSI, 01 - HSI, 10 - HSE, 11 - PLL

  // Wait until system clock is selected
  while((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL);

  // ...
}
```

7 References

[1] STM32L4x6 Advanced ARM-based 32-bit MCUs Reference Manual

[2] Yifeng Zhu, "Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C", ISBN: 0982692633

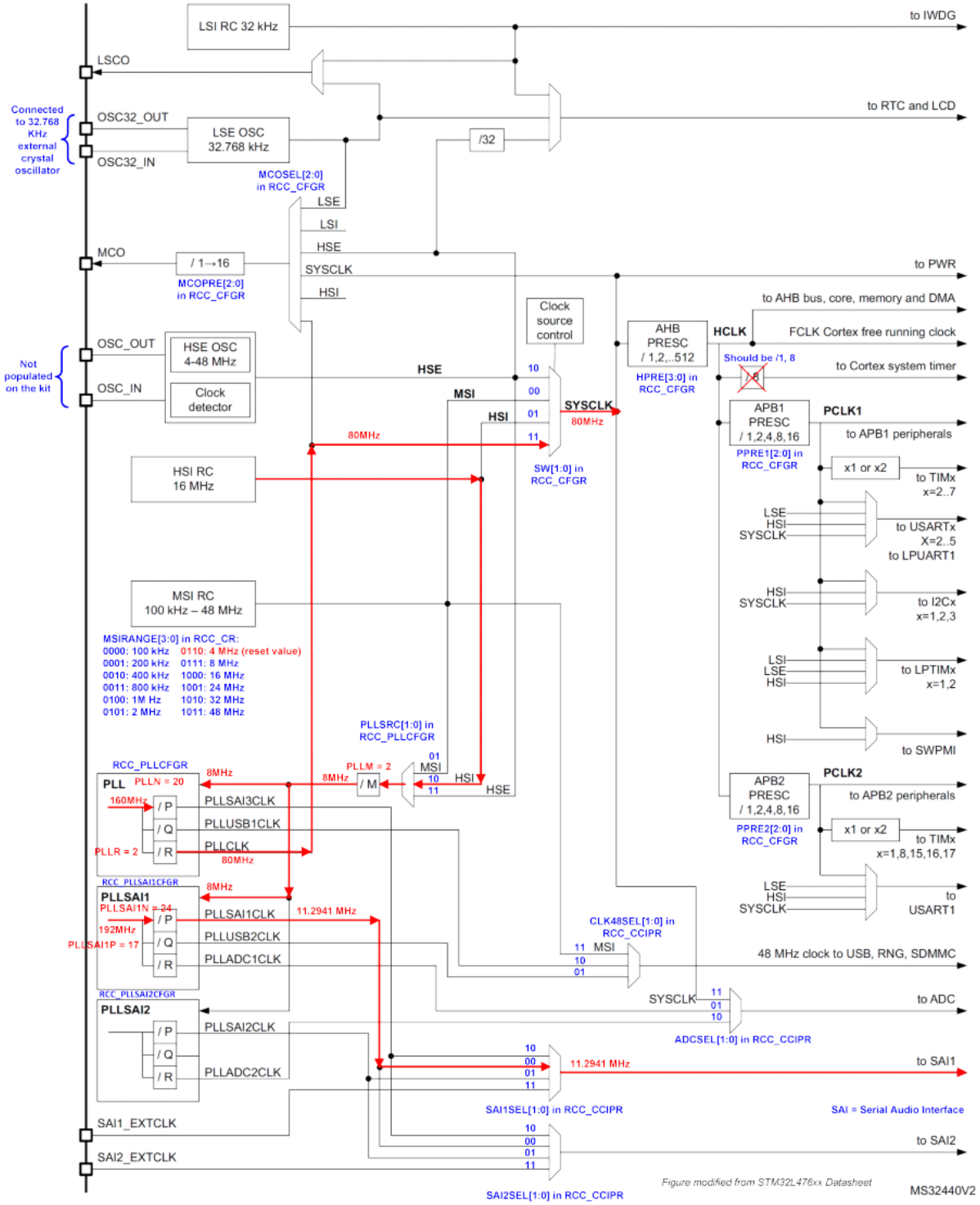

Figure 4: **STM32L476RGTx** Clock Tree