ECE 1770

ELECTRONIC MICROPROCESSOR SYSTEMS

Lab 5: Interfacing Push-button and LED in Assembly

TASK#	Grading criteria	Instructor Initial
1	Your program runs successfully without any compilation error when you demo it. (10)_____ The behavior of the LED follows the requirement of this task. (20)_____ Submit your complete and correct assembly codes on Courseweb. (20)_____	
2	Your program runs successfully without any compilation error when you demo it. (10)_____ The behavior of the LED follows the requirement of this task. (20)_____ Submit your complete and correct assembly codes on Courseweb. (20)_____	

In order to receive credit for this laboratory exercise, please submit this handout to your lab instructor when you are finished.

Team Members: Group Number:	TA:

ECE1770

Lab 5: Interfacing Push-button and LED in Assembly

Jingtong Hu
February 24, 2019

1. Objective

1) Learn the basics of GPIO input and output configuration: input/output, push-pull/open-drain, pull up/down.
2) Program GPIO registers to perform simple digital I/O input (interfacing push button, i.e. the center key of the joystick) and output (interfacing LED).
3) Understand polling I/O (busy waiting) and its inefficiency.

2. Lab Assignments

1) Follow Chapter 14 of your textbook and implement an assembly program that toggles the Green LED every 1 second.
2) Toggle the Red LED when the center key of the joystick is pressed.

Note: When you submit your codes on Courseweb, please upload only two separate assembly files called *lab5_task1.s*, *lab5_task2.s*, which contains the complete codes of task 1 and task 2, respectively.

3. LEDs on the Board

There are **two LEDs** on the STM32L4 discovery board (shown in Figure 1), which are connected to the GPIO Port B Pin 2 (PB2) and the GPIO Port E Pin 8 (PE8) pin of the STM32L4 processor, respectively. To light up a LED, software must at least perform the following three operations:

1. Enable the clock of the corresponding GPIO port. (By default, the clock to all peripherals, including GPIO ports, are turned off to improve the energy efficiency.)
2. Set the mode of the corresponding GPIO pin. It must be set as output (By default, the mode of all GPIO pin is analog)
3. Set the output value of the corresponding GPIO pin to 1. (When the output value is 1, the voltage on the GPIO pin is 3V. When the output value is 0, the voltage is 0V.)

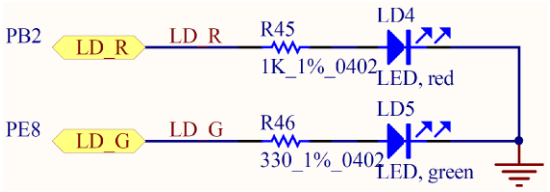

Figure 1. Two LEDs on the STM32L4 Discovery Board

The joystick (MT-008A) has five keys, including *up, down, left, right,* and *center*. Each key has an output pin and all of them are connected to a common pin, as shown in Figure 2. In this lab, the push-button we use is the *center* key of the joystick.

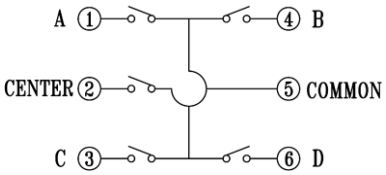

Figure 2. The Keys for Joysticks

The joystick is connected to the GPIO pins PA0, PA1, PA5, PA2, and PA3. The detail of the connection between them is shown in Figure 3. A capacitor and a resistor are added for each GPIO pin to perform ***hardware debouncing***.

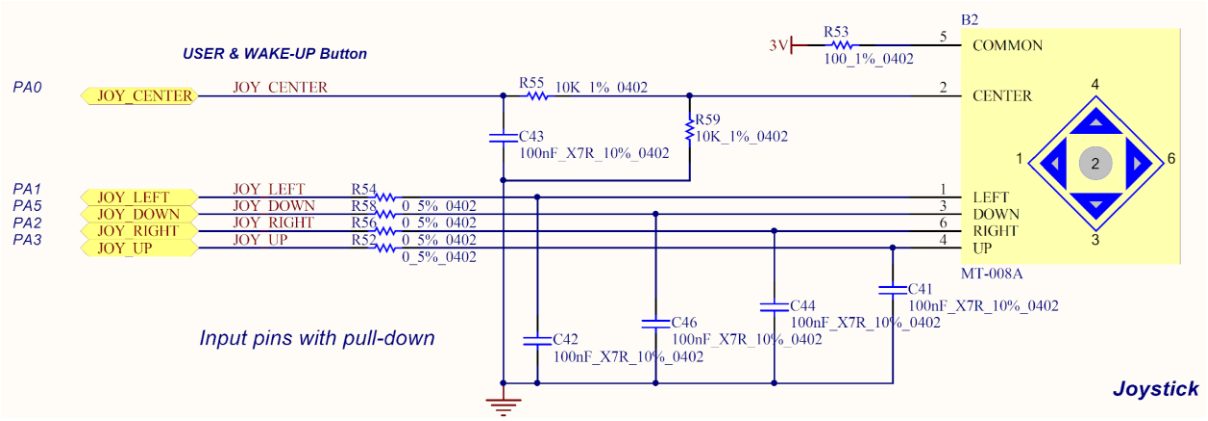

Figure 3. The Connection between the Joystick and GPIO Pins

Note: At ambient temperature, GPIO pin (general purpose input/outputs) can sink or source up to **±8 mA**.

4. Pin Connections

STM32L476VGT6 microcontroller features 1 Mbyte of Flash memory, and 128 Kbytes of RAM in LQFP100 package (with 100 pins). The onboard peripherals are connected as follow (shown in Table 1).

Table 1. The Mapping of Onboard Peripherals and GPIO Pins

Peripheral	Peripheral's Interface	Pin	Peripheral	Peripheral's Interface	Pin
Joystick (MT-008A)	Center	PA0	LCD	VLCD	PC3
	Left	PA1		COM0	PA8
	Right	PA2		COM1	PA9
	Up	PA3		COM2	PA10
	Down	PA5		COM3	PB9
User LEDs	LD4 Red	PB2		SEG0	PA7
	LD5 Green	PE8		SEG1	PC5
CS43L22 Audio DAC Stereo I2C address 0x94	SAI1_MCK	PE2		SEG2	PB1
	SAI1_FS	PE4		SEG3	PB13
	SAI1_SCK	PE5		SEG4	PB15
	SAI1_SD	PE6		SEG5	PD9
	I2C1_SCL	PB6		SEG6	PD11
	I2C1_SDA	PB7		SEG7	PD13
	Audio_RST	PE3		SEG8	PD15
MP34DT01 MEMS MIC	Audio_DIN	PE7		SEG9	PC7
	Audio_CLK	PE9		SEG10	PA15
LSM303C eCompass	MAG_CS	PC0		SEG11	PB4
	MAG_INT	PC1		SEG12	PB5
	MAG_DRDY	PC2		SEG13	PC8
	MEMS_SCK	PD1 (SPI2_SCK)		SEG14	PC6
	MEMS_MOSI	PD4 (SPI2_MOSI)		SEG15	PD14
	XL_CS	PE0		SEG16	PD12
	XL_INT	PE1		SEG17	PD10
L3GD20 Gyro	MEMS_SCK	PD1 (SPI2_SCK)		SEG18	PD8
	MEMS_MOSI	PD4 (SPI2_MOSI)		SEG19	PB14
	MEMS_MISO	PD3 (SPI2_MISO)		SEG20	PB12
	GYRO_CS	PD7		SEG21	PB0
	GYRO_INT1	PD2		SEG22	PC4
	GYRO_INT2	PB8		SEG23	PA6
ST-Link V2	USART_TX	PD5	USB OTG	OTG_FS_PowerSwitchOn	PC9
	USART_RX	PD6		OTG_FS_OverCurrent	PC10
	SWDIO	PA13		OTG_FS_VBUS	PC11
	SWCLK	PA14		OTG_FS_ID	PC12
	SWO	PB3		OTG_FS_DM	PA11
	3V3_REG_ON	PB3		OTG_FS_DP	PA12
Quad SPI Flash Memory	QSPI_CLK	PE10(QUADSPI_CLK)	Clock	OSC32_IN	PC14
	QSPI_CS	PE11(QUADSPI_NCS)		OSC32_OUT	PC15
	QSPI_D0	PE12(QUADSPI_BK1_IO0)		OSC_IN	PH0
	QSPI_D1	PE13(QUADSPI_BK1_IO1)		OSC_OUT	PH1
	QSPI_D2	PE14(QUADSPI_BK1_IO2)			
	QSPI_D3	PE15(QUADSPI_BK1_IO3)			

5. Clock Configuration

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

System Clock

In order to meet the requirement of performance and energy-efficiency for different applications, the processor core can be driven by four different clock sources, including, ***HSI*** (high-speed internal) oscillator clock, ***HSE*** (high-speed external) oscillator clock, **PLL** clock, and **MSI** (multi-speed internal) oscillator clock. A faster clock provides better performance but usually consumes more power, which is not appropriate for battery-powered systems.

Peripheral Clock

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

Figure 4 shows the clock tree of ***STM32L476VGT6***, the processor used in the STM32L4 Discovery kit. 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 it is not used. The software can select various clock sources and scaling factors to achieve desired clock speed, depending on the application's needs.

In this lab, we will use the default settings of the system clock. The clock frequency is 4 MHZ.

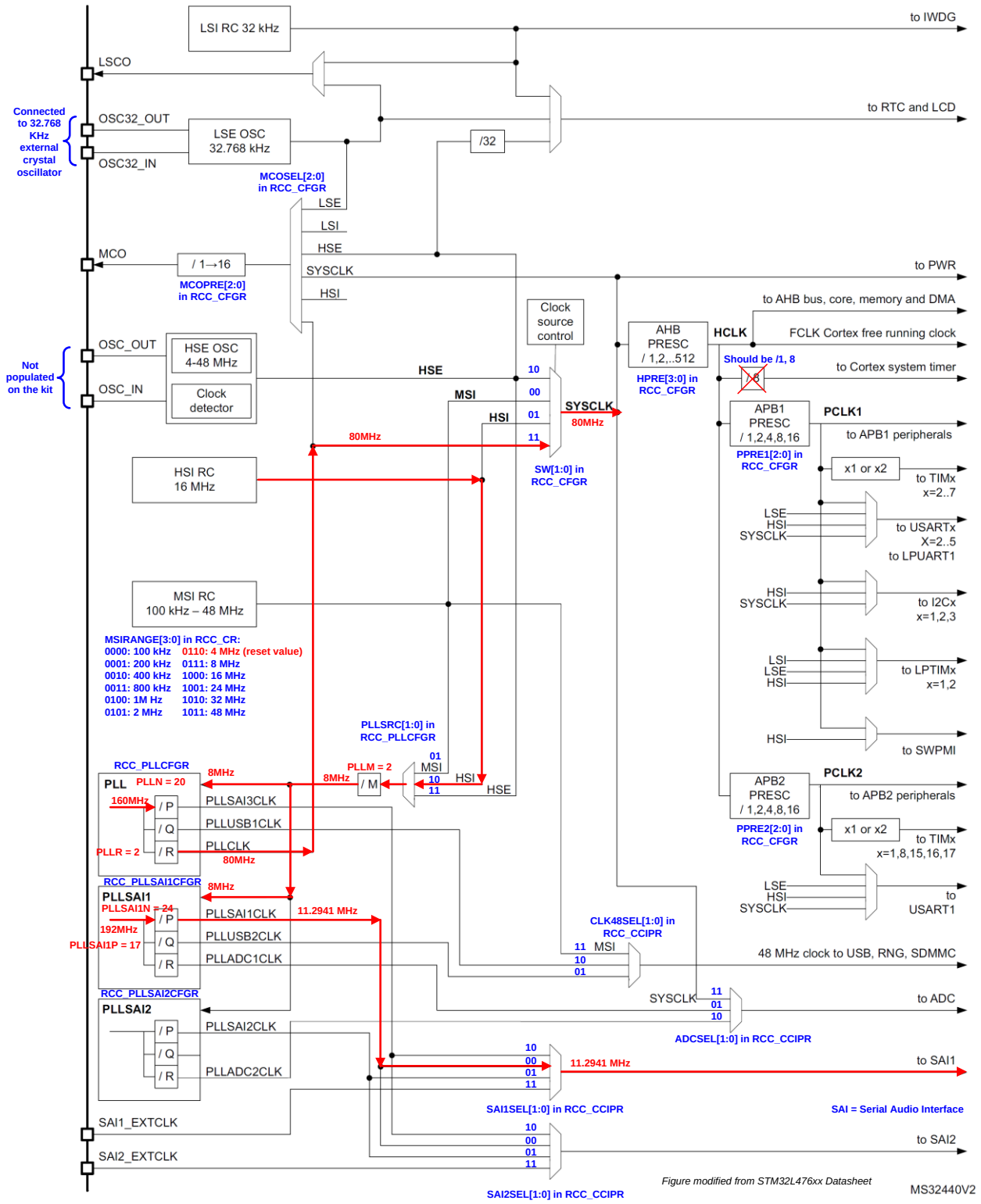

Figure 4. The Clock Tree of *STM32L476VGT6*

6. GPIO Pins Configuration

Each of the GPIO pins can be configured by software as output (push-pull or open-drain), as input (with or without pull-up or pull-down) or as peripheral alternate function. In this lab, we will configure PB2 and PE8 as push-pull output. Each general-purpose I/O port x (x = A, B, C, …, H) has:

- four 32-bit configuration registers
  - GPIOx_MODER (mode register)
  - GPIOx_OTYPER (output type register)
  - GPIOx_OSPEEDR (output speed register)
  - GPIOx_PUPDR (pull-up/pull-down register)
- two 32-bit data registers
  - GPIOx_IDR (input data register)
  - GPIOx_ODR (output data register)
- a 32-bit set/reset register (GPIOx_BSRR)
- a 32-bit locking register (GPIOx_LCKR)
- two 32-bit alternate function selection registers
  - GPIOx_AFRH (alternative function high register)
  - GPIOx_AFRL (alternative function low register)

The detail of these configurations is shown in Figure 5.

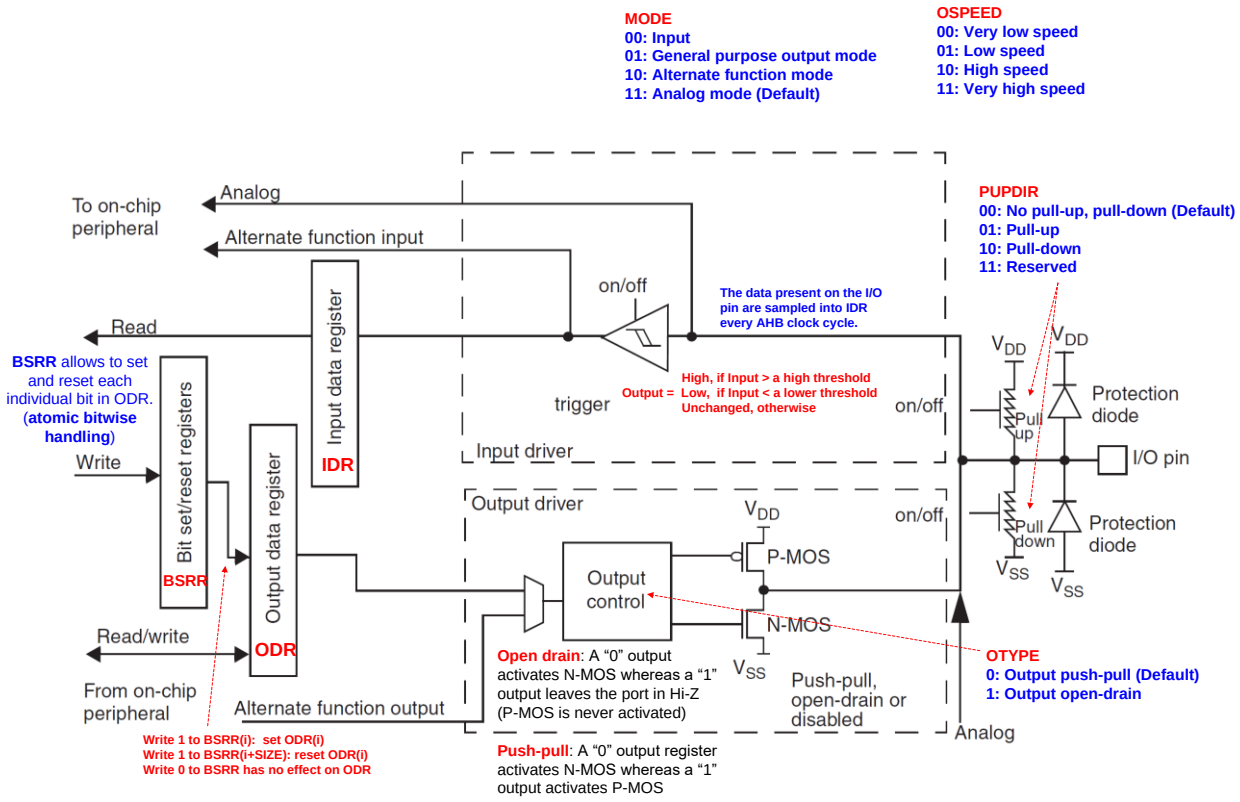

Figure 5. The Detail of GPIO Pin Configuration

7. Pre-Lab

Please read this part and fill in the blanks before coding. It is conducive to finishing your lab tasks.

Usage of the mask:

When we toggle, set, or reset specific bits of a word (4 bytes), we have to keep the other bits of the word unchanged. For example, we want to set bit 2 of the value in R1, the following code is incorrect because it resets all the other bits in this word.

```
MOV R1, 0x00000004
```

The correct approach is:

```
ORR R1, 0x00000004
```

Typically we use *MASK* to facilitate the operations of toggling, setting or resetting a group of bits in a variable. For example, now you have declared a variable in the data section:

```
MASK DCD 0x00008004
```

Assume the value of the *MASK* has been loaded to R2. Then we can manipulate the bits of value in R1 using the following instructions:

```
; Set bit 15 and bit 2
ORR R1, R2

; Reset bit 15 and bit 2
BIC R1, R2

; Toggle bit 15 and bit 2
EOR R1, R2
```

1. Enable the clock of GPIO Port A (for joystick), Port B (for Red LED) and Port E (for Green LED).

1.1 Enable the clock of GPIO Port A

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					GPIOPHEN	GPIOPGEN	GPIOPFEN	GPIOPEEN	GPIOPDEN	GPIOPCEN	GPIOPBEN	GPIOPAEN
Mask																																

Value of AHB2 Enable Register MASK for Port A = 0x_____ (in HEX)

1.2 Enable the clock of GPIO Port B

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					GPIOPHEN	GPIOPGEN	GPIOPFEN	GPIOPEEN	GPIOPDEN	GPIOPCEN	GPIOPBEN	GPIOPAEN
Mask																																

Value of AHB2 Enable Register MASK for Port B = 0x_____ (in HEX)

1.3 Enable the clock of GPIO Port E

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					GPIOPHEN	GPIOPGEN	GPIOPFEN	GPIOPEEN	GPIOPDEN	GPIOPCEN	GPIOPBEN	GPIOPAEN
Mask																																

Value of AHB2 Enable Register MASK for Port E = 0x_____ (in HEX)

2. Pin Initialization for Red LED (PB 2)

2.1 Configure PB 2 as 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	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																																

GPIOB Mode Register MASK Value = 0x_____(in HEX)

2.2 Configure PB 2 Output Type as Push-Pull

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																																

GPIOB Output Type Register MASK Value = 0x_____ (in HEX)

2.3 Configure PB 2 Output Type as No Pull-up No Pull-down

NO PUPD (00, reset), Pullup (01), Pulldown (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																																

GPIOB Pull-up Pull-down Register MASK Value = 0x_____(in HEX)

3. Pin Initialization for Green LED (PE 8)

3.1 Configure PE 8 as 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	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																																

GPIOE Mode Register MASK Value = 0x_____ (in HEX)

3.2 Configure PE 8 Output Type as Push-Pull

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																																

GPIOE Output Type Register MASK Value = 0x_____ (in HEX)

3.3 Configure PE 8 Output Type as No Pull-up No Pull-down

NO PUPD (00, reset), Pullup (01), Pulldown (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																																

GPIOE Pull-up Pull-down Register MASK Value = 0x_____ (in HEX)

4. Pin Initialization for Joy Stick

4.1 Configure PA0 (Center), PA1 (Left), PA2 (Right), PA3 (Up), and PA5 (Down) as 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																																
Value																																

GPIOA Mode Register MASK Value = 0x_____ (in HEX)

4.2 Configure PA0 (Center), PA1 (Left), PA2 (Right), PA3 (Up), and PA5 (Down) as Pull-down

NO PUPD (00, reset), Pullup (01), Pulldown (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																																
Value																																

GPIOA Pull-up Pull-down Register MASK Value = 0x_____ (in HEX)

5. Pre-defined Constants for GPIO

In the template of project for lab 5 provided on Courseweb, you may notice that there is a file called *stm32l476xx_constants.s.* This file contains a lot of constans predefined for GPIO. For example, we have the following codes in the *main.s* provieded:

```
; Enable clock of GPIO Port A
LDR   r0, =RCC_BASE
LDR   r1, [r0, #RCC_AHB2ENR]
;  0x0001 is the value of AHB2 Enable Register MASK for Port A, you can also
;  use the constant RCC_AHB2ENR_GPIOAEN to replace 0x 0001 here.
ORR   r1, r1, #0x0001
STR   r1, [r0, #RCC_AHB2ENR]
```

In the above codes, *RCC_BASE, RCC_AHB2ENR* are the constants pre-defined in *stm32l476xx_constants.s*. *RCC_BASE* reprensents the base address for RCC (Reset and Clock Control). And *RCC_AHB2ENR* represents the **offset** of AHB2ENR (AHB2 Enable Register) compared to *RCC_BASE*. In order to get the accurate address of AHB2 Enable Register, you need to add *RCC_AHB2ENR* to *RCC_BASE* and the sum of the addition is the address you want.

If you want to see the concrete value of *RCC_AHB2ENR,* you can open the file *stm32l476xx_constants.s* in Keil and then use ***Ctrl + F***. You will see the window shown as Figure 6. Type the name of the constants you want to search in the first blank and then click *Find Next*. You can see the matching value in the highlighted line.

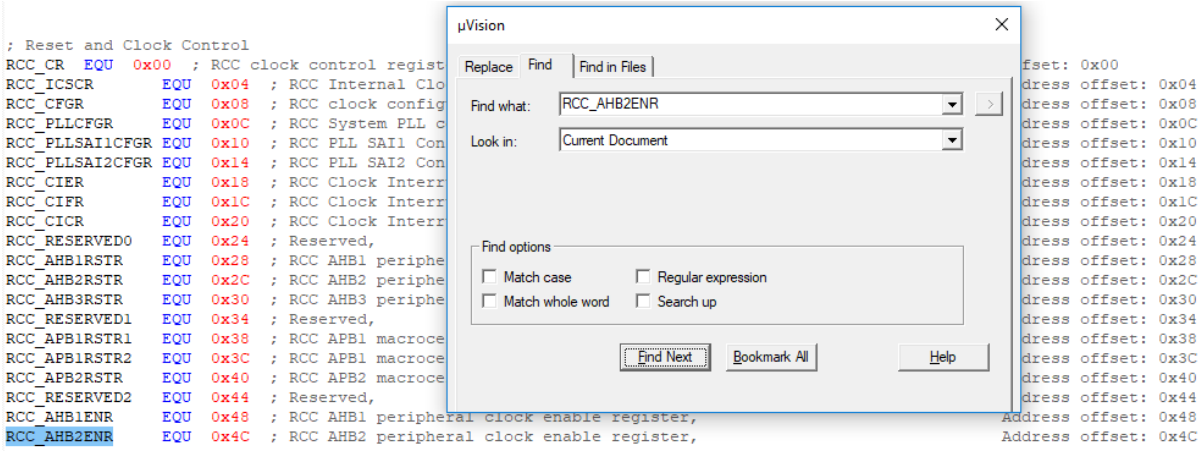

Figure 6. The Window for Search Constants in Files

Note: If you want to use these constants predefined in the *stm32l476xx_constants.s* , you must add the following line in the head of *main.s*, which we have already done in the template project for lab5 provided on Courseweb.

INCLUDE stm32l476xx_constants.s ; Load Constant Definitions

Tasks

Note: For the following two tasks, we have already provided part of the codes in the template projects for lab 5 on Courseweb. Please download it to your computer. Try to imitate the existing codes in *main.s* to finish the rest part of it. The comments are crucial hints for coding. Make full use of them. In some lines, you may need some value as mask for specific registers, which can be found in the **Pre-Lab** section if you have finished them. You may also need some other constants predefined in *stm32l476xx_constants.s*. Refer to the slides of the related class may help you to do this.

Task 1: Toggle the Green LED Every 1 Second

In this task, we are going to toggle the **Green LED** every 1 second. In the main function, we initialize the pin for Green LED as push-pull output. Then we enter the loop and start toggling the LED. In the loop, we first delay for 1 second by calling the function *Delay1Second*. After that we toggle the Green LED by calling *Toggle_GREEN_LED*.

Task 2: Toggle the Red LED When the Center Button is Pressed

In this task, we are going to toggle the **Red LED** when the center key of the joystick is pressed. In the main function, we initialize the pin for Red LED as push-pull output. Then we enter the loop and start checking whether the center key is pressed. If it is not pressed, we jump back to the loop; otherwise we wait until the center key is released before toggling the Red LED. The waiting is necessary because if we skip the waiting, our program will toggle the LED and jump back to check the button again, after which the LED will be toggled one more time. The flowchart is shown in Figure 7.

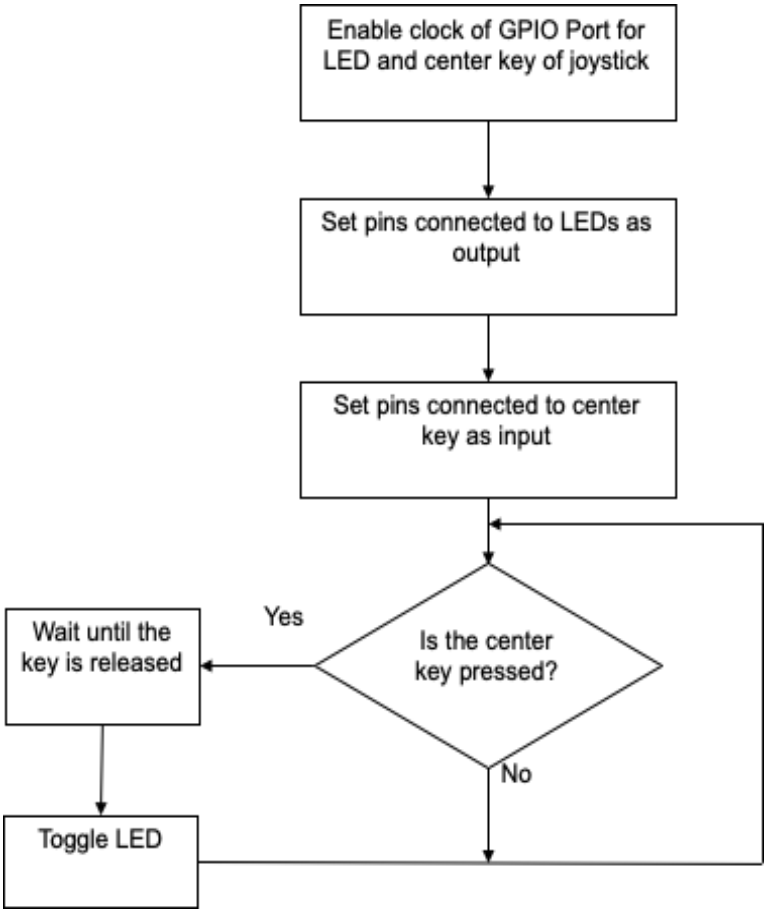

Figure 7. The Flowchart of Task 2