ECE3810 Microprocessor System Design Laboratory

Lab 1. General Purpose Inputs and Outputs (GPIO)

School of Science and Engineering

The Chinese University of Hong Kong, Shenzhen

2024-2025 Term 1

1. Objectives

- To study the setting of GPIOs as inputs and outputs
- To study the project in C language with Standard Firmware Library for registers' setting
- To study the way for GPIOs setting by registers
- Create your own libraries for your project board

2. Basics

Peripherals mean things which are outside of the core microprocessor. They are very important for microprocessor systems. In this lab, we will start with some basic peripherals, named General Purpose Inputs and Outputs (GPIO). There are 7 sets of 16-bit GPIO ports in the microprocessor, from GPIOA to GPIOG. Each GPIOx has 16 pins, i.e. Px0-Px15 (x=A, B, C, …, G). Figure 1 illustrates the pin arrangement of STM32F103ZE.

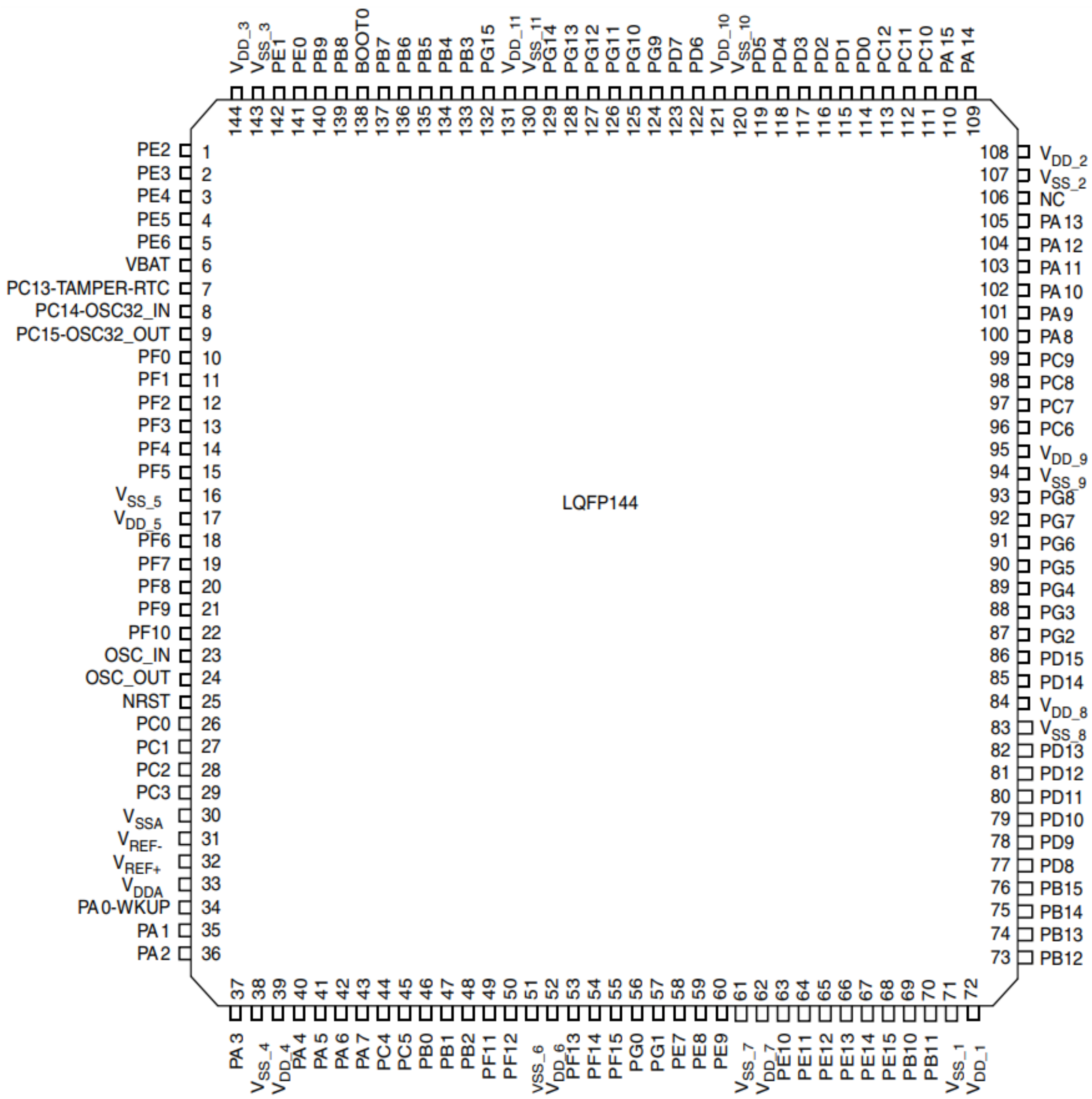


Figure 1. Pin arrangement of STM32F103ZE

Figure 2 shows the system architecture inside the STM32 chip. GPIOA-GPIOG are connected to APB2. APB means Advanced Peripheral Bus. APB1 is limited to 36 MHz, APB2 operates at full speed (up to 72 MHz depending on the device). If you want to use APB2, you have to turn on the clock of the peripheral, by RCC (Reset & Clock Control).

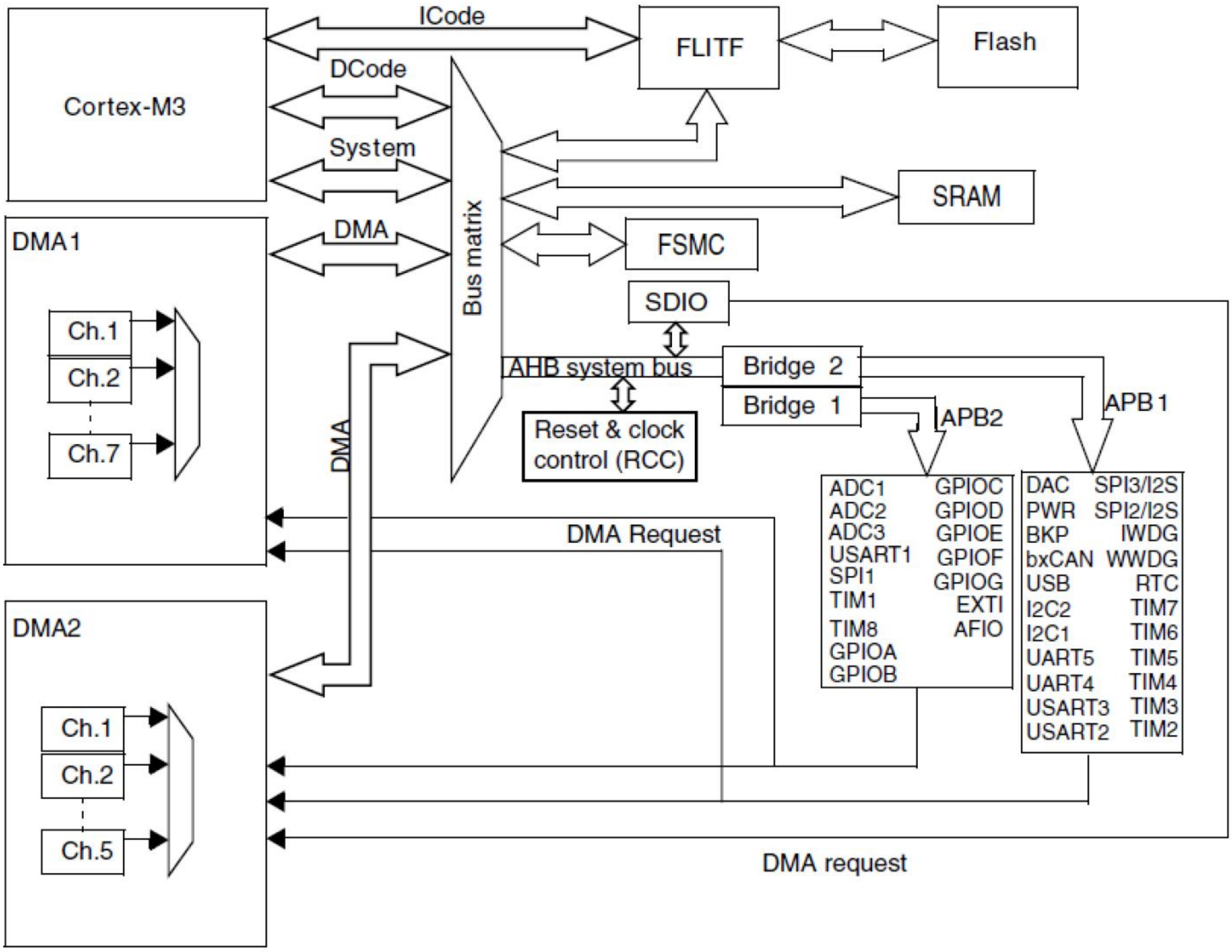


Figure 2. System Architecture

To turn on RCC, you need to know the RCC register map and reset values. Figure 3 gives the complete register map of RCC. There are 10 registers for RCC. In this lab, what you need to use is RCC_APB2ENR, which enables the corresponding peripherals.

Offset	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
0x00	RCC_CR	Reserved						PLL RDY	PLL ON	Reserved				CSSON	HSEBYP	HSERDY	HSEON	HSICAL[7:0]								HSITRIM[4:0]					Reserved	HSIRDY	HSION
	Reset value							0	0					0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0		1	1
0x04	RCC_CFGR	Reserved					MCO [2:0]			Reserved	USBPRE	PLLMUL[3:0]				PLLXTPRE	PLLSRC	ADC PRE [1:0]		PPRE2 [2:0]			PPRE1 [2:0]			HPRE[3:0]				SWS [1:0]		SW [1:0]	
	Reset value						0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	RCC_CIR	Reserved								CSSC	Reserved		PLLRDYC	HSERDYC	HSIRDYC	LSERDYC	LSIRDYC	Reserved			PLLRDYIE	HSERDYIE	HSIRDYIE	LSERDYIE	LSIRDYIE	CSSF	Reserved		PLLRDYF	HSERDYF	HSIRDYF	LSERDYF	LSIRDYF
	Reset value									0			0	0	0	0	0				0	0	0	0	0	0			0	0	0	0	0
0x0C	RCC_APB2RSTR	Reserved										TIM11RST	TIM10RST	TIM9RST	Reserved			ADC3RST	USART1RST	TIM8RST	SPI1RST	TIM1RST	ADC2RST	ADC1RST	IOPGRST	IOPFRST	IOPERST	IOPDRST	IOPCRST	IOPBRST	IOPARST	Reserved	AFIORST
	Reset value											0	0	0				0	0	0	0	0	0	0	0	0	0	0	0	0	0		0
0x010	RCC_APB1RSTR	Reserved		DACRST	PWRRST	BKPRST	Reserved	CANRST	Reserved	USBRST	I2C2RST	I2C1RST	UART5RST	UART4RST	USART3RST	USART2RST	Reserved	SPI3RST	SPI2RST	Reserved		WWDGRST	Reserved		TIM14RST	TIM13RST	TIM12RST	TIM7RST	TIM6RST	TIM5RST	TIM4RST	TIM3RST	TIM2RST
	Reset value			0	0	0		0		0	0	0	0	0	0	0		0	0			0			0	0	0	0	0	0	0	0	0
0x14	RCC_AHBENR	Reserved																					SDIOEN	Reserved	FSMCEN	Reserved	CRCEN	Reserved	FLITFEN	Reserved	SRAMEN	DMA2EN	DMA1EN
	Reset value																						0		0		0		1		1	0	0
0x18	RCC_APB2ENR	Reserved										TIM11 EN	TIM10 EN	TIM9 EN	Reserved			ADC3EN	USART1EN	TIM8EN	SPI1EN	TIM1EN	ADC2EN	ADC1EN	IOPGEN	IOPFEN	IOPEEN	IOPDEN	IOPCEN	IOPBEN	IOPAEN	Reserved	AFIOEN
	Reset value											0	0	0				0	0	0	0	0	0	0	0	0	0	0	0	0	0		0
0x1C	RCC_APB1ENR	Reserved		DACEN	PWREN	BKPEN	Reserved	CANEN	Reserved	USBEN	I2C2EN	I2C1EN	UART5EN	UART4EN	USART3EN	USART2EN	Reserved	SPI3EN	SPI2EN	Reserved		WWDGEN	Reserved		TIM14EN	TIM13EN	TIM12EN	TIM7EN	TIM6EN	TIM5EN	TIM4EN	TIM3EN	TIM2EN
	Reset value			0	0	0		0		0	0	0	0	0	0	0		0	0			0			0	0	0	0	0	0	0	0	0
0x20	RCC_BDCR	Reserved															BDRST	RTCEN	Reserved					RTC SEL [1:0]		Reserved					LSEBYP	LSERDY	LSEON
	Reset value																0	0						0	0						0	0	0
0x24	RCC_CSR	LPWRSTF	WWDGRSTF	IWDGRSTF	SFTRSTF	PORRSTF	PINRSTF	Reserved	RMVF	Reserved																						LSIRDY	LSION
	Reset value	0	0	0	0	1	1		0																							0	0

Figure 3. RCC Register Map

Figure 4 is the clock tree. With the peripheral clock enabled, APB2 clock can be output to PCLK2. More details in setting the clock tree will be introduced in other labs. In this lesson, we will use the default values.

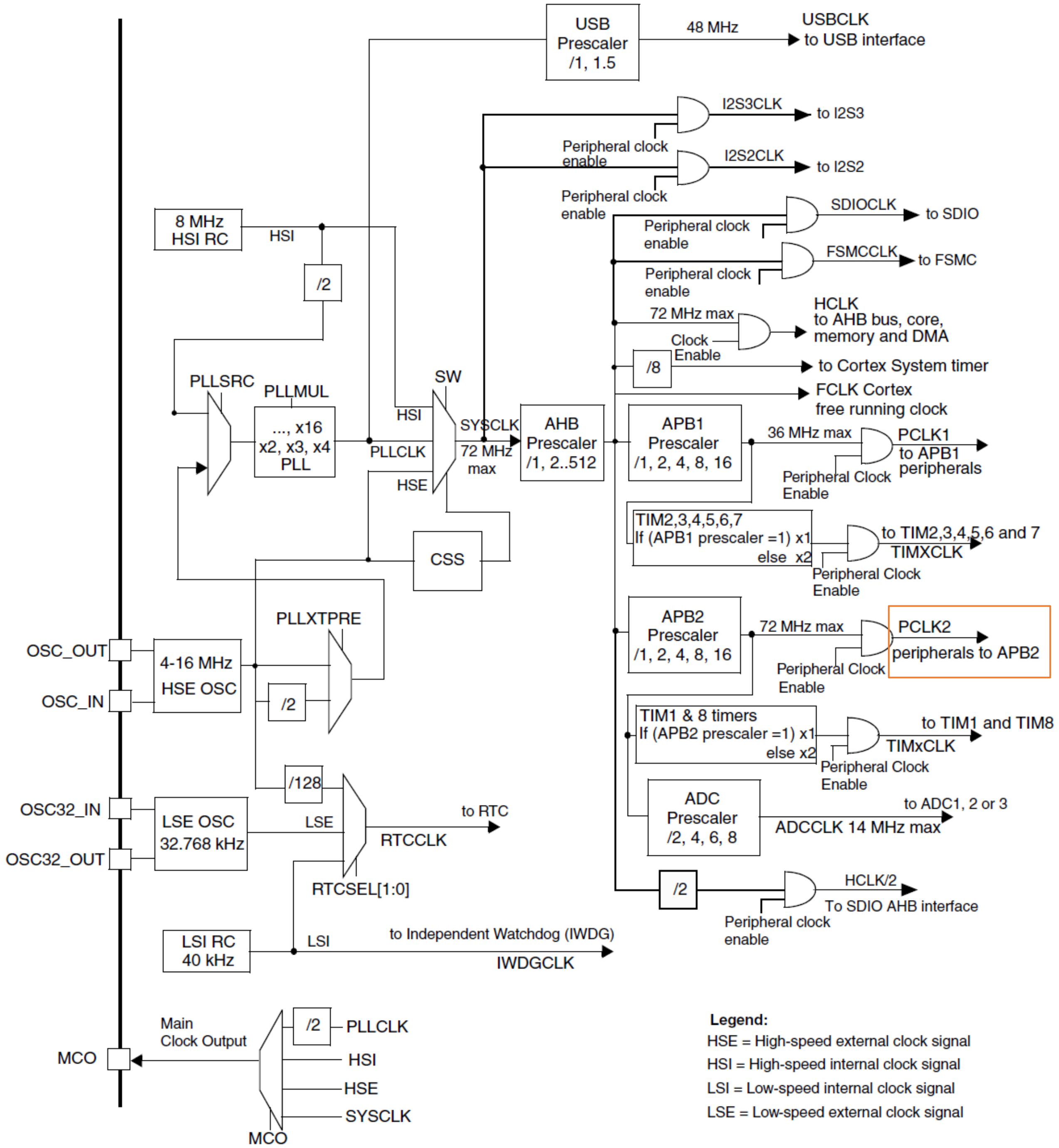


Figure 4. Clock Tree

In this lab, we use keys, LEDs and buzzer on the board. Try to locate them on your board. All of them are connected to the microprocessor's GPIOs.

Table 1. Keys, LEDs and Buzzer pin assignment

Item	Input / Output	Operation	Connect to Pins
Key_Up	Input	Press=High	PA0
Key0	Input	Press=Low	PE4
Key1	Input	Press=Low	PE3
Key2	Input	Press=Low	PE2
DS0 (LED0)	Output	Low=Lit	PB5
DS1 (LED1)	Output	Low=Lit	PE5
Buzzer	Output	High=Sound	PB8

For programming microprocessor, you can use C or Assembly language. C language is popular and easy to understand. Assembly language is very close to machine code and is suitable for making device driver and solving the processing task with fast time constraint (like real-time applications).

Cortex-M3 has more than thousand peripheral registers, and you can reset them by assembly language. Chip's manufacturer also provides standard firmware library (e.g. STM32F10x_StdPeriph_Lib_V3.5.0) that you can call to reset registers.

3. Experiments

3.1 Experiment 1: Set a GPIO as an output and drive a LED with standard peripheral library

For the 7 sets of 16-bit GPIO, each port has the following registers:

- Two 32-bit configuration registers
  - GPIOx_CRL: configuration register low (for lower 8 bits)
  - GPIOx_CRH: configuration register high (for higher 8 bits)
- Two 32-bit data registers
  - GPIOx_IDR: input data register
  - GPIOx_ODR: output data register
- A 32-bit set/reset register
  - GPIOx_BSRR: bit set/reset register
- A 16-bit reset register
  - GPIOx_BRR: bit reset register
- A 32-bit locking register
  - GPIOx_LCKR

Each port bit of GPIOx can be individually configurated by software in several modes:

- Input floating
- Input pull-up (GPIO_Mode_IPU)
- Input pull-down (GPIO_Mode_IPD)
- Analog
- Output open-drain
- Output push-pull (GPIO_Mode_Out_PP)
- Alternate function push-pull
- Alternate function open-drain

In this experiment, we will use the 3 modes in reading. Firstly, we will set PB5 (connected to LED0) as output push-pull, and drive LED0 at DS0. As shown in Figure 5, output push-pull means the I/O pin will output HIGH or LOW by turning on the P-MOS or N-MOS FET, respectively.

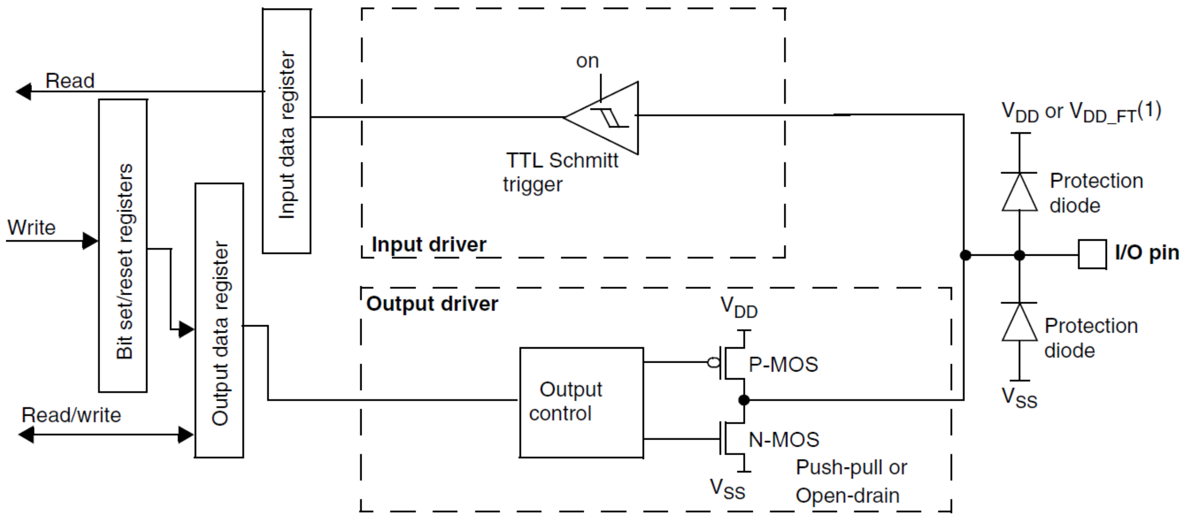


Figure 5. Output Configuration of A GPIO Pin

To complete this task, we need to go through some steps for setting up PB5.

- Enable the peripheral clock of GPIOB;
- Set GPIOB peripheral maximum speed to 50 MHz;
- Set PB5 as a push-pull mode output;
- Set PB5 output to “0” (lit LED) or to “1” (off LED).

The source code is provided as below. In this experiment, we use the standard peripheral library. All codes are easy to understand. If you want to know the definition of an item in the standard peripheral library, highlight the item and use the mouse right click, select “Go To Definition of …”, or read the document “stm32f10x_stdperiph_lib_um.chm” in folder “STM32F10x_StdPeriph_Lib_V3.5.0”.

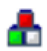
```c
#include "stm32f10x.h"

void Delay(u32 count) //Delay subroutine generate soem time delay by looping
{
  u32 i;
  for (i=0;i<count;i++);
}

int main(void)
{
  GPIO_InitTypeDef GPIO_InitStructure; //Define a structure to configurate the GPIO
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE); //Enable GPIOB
  GPIO_InitStructure.GPIO_Pin=GPIO_Pin_5; //Pin5
  GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP; //Output Push-Pull mode
  GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz; //50MHz maximal speed
  GPIO_Init(GPIOB, &GPIO_InitStructure);//Set Pin5 of GPIOB with the parameters above
  GPIO_SetBits(GPIOB, GPIO_Pin_5); //Set Pin5 of GPIOB: to high
  while(1)
  {
    GPIO_ResetBits(GPIOB, GPIO_Pin_5); //Reset Pin5 of GPIOB: to low
    Delay(1000000);
    GPIO_SetBits(GPIOB, GPIO_Pin_5); //Set Pin5 of GPIOB: to high
    Delay(1000000);
  }
}

```

Figure 6. Source Code of Experiment 1

The function Delay(u32 count) is a subroutine that will generate some time delay by looping. This time delay is not accurate, but changing the input parameter count can still adjust the delay time.

u32 means a 32-bit unsigned number, and the maximum of this number is 2^32=4,294,967,296.

Procedures:

1. Type in the codes into your project “main.c”. Do not forget how to build up the project. Recall Lab 0.
2. “stm32f10x_rcc.c” and “stm32f10x_gpio.c” must be included in your project “Fw_lib”. Click  on the toolbar and fill the Manage Project Items to include the two files.
3. Compile the program and download the binary output file to your project board.
4. Modify the parameter value of “Delay” function and let the LED0 flash one second alternatively (one second on, and one second off; with less than 5% of error).

5. Find the GND and PB5 pins on the board and use the logic analyzer from the oscilloscope Tektronix MSO2022B to measure the signal of PB5. (Refer to user manual of the oscilloscope Tektronix MSO2022B).

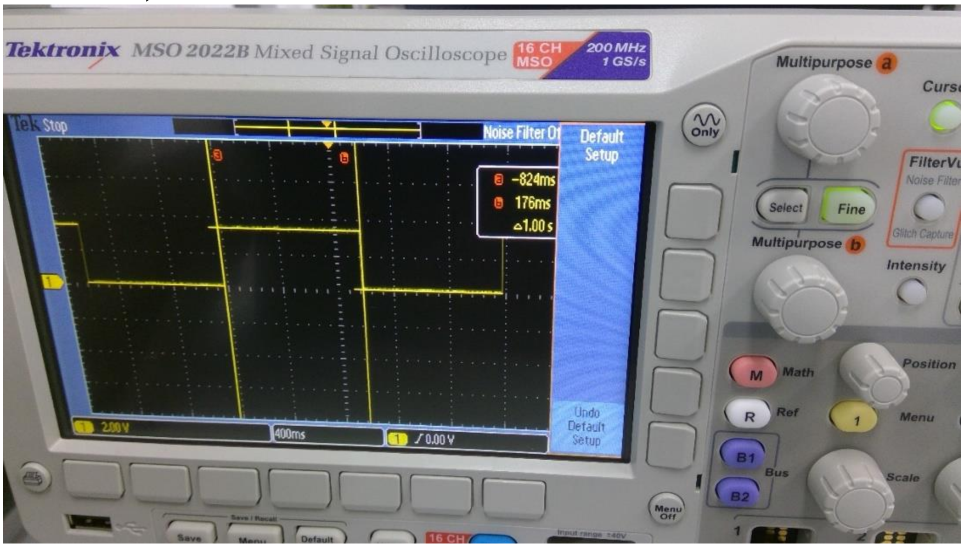


Figure 7. A sample curve on PB5

Figure 8. Buttons for turning on the Logic Analyzer and socket for the Probes.

Figure 9. Probes in Logic Analyzer. The ground signal of the development board (GND) should be connected to Ground of the logic probe.

[Demonstration] Demonstrate that you have realized Step 4 and 5.

[In Report] Include your source code and the test result.

3.2 Experiment 2: Read a key from GPIO input and drive an LED with a standard peripheral library

The "Key2" switch on board is connected to PE2 (GPIOE, bit-2). When it is pressed, the signal to PE2 will be LOW. Hence, in the internal part of the microprocessor, it needs an internal pull-up, otherwise, the signal of PE2 will not be HIGH, when Key2 is not pressed.

For each pin of the GPIO, there are two internal resistors, which you can choose to connect by setting that pin. As shown in Figure 8, if you connect the upper resistor to V_{DD}, it pulls up, and thus when Key2 is not pressed, the signal to PE2 is high. When Key2 is pressed, based on Table 1, the signal to on Key2 is low, and then PE2 is also low.

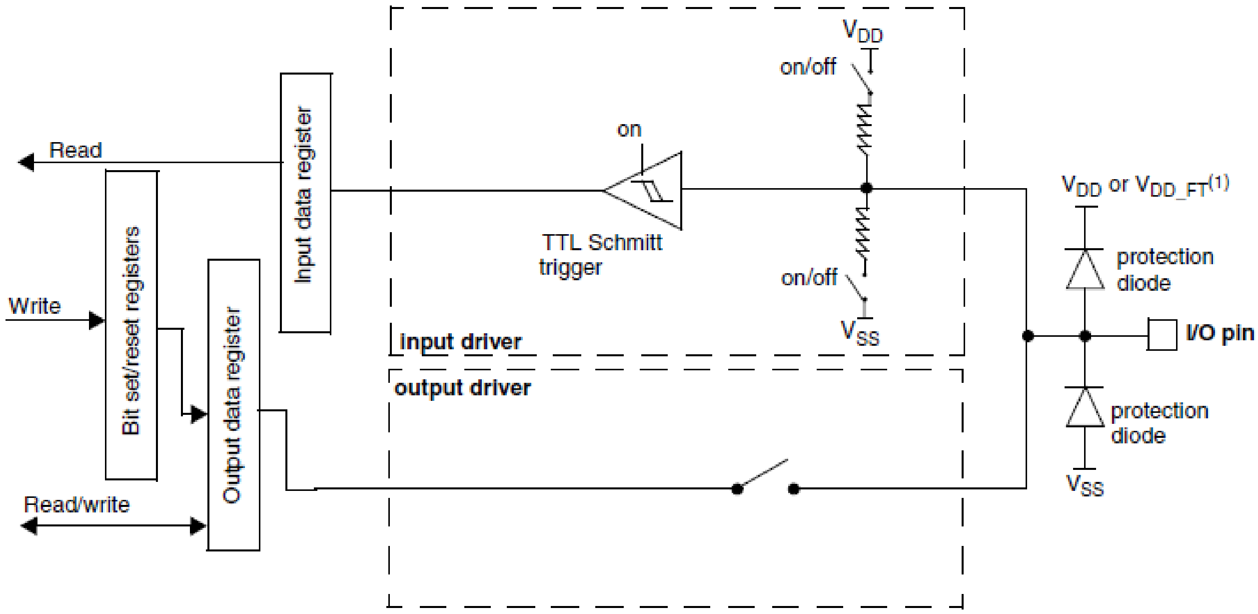


Figure 10. Input configuration

In this experiment, you need to read "“stm32f10x_gpio.h” and “stm32f10x_gpio.c” in Fw_lib to find out how to set the pin PE2 as input and find the suitable function that can read a bit from GPIO input pin.

To set PE2’s mode as internal pull-up, find a type in the list below.

```
typedef enum
{ GPIO_Mode_AIN = 0x0,
  GPIO_Mode_IN_FLOATING = 0x04,
  GPIO_Mode_IPD = 0x28,
  GPIO_Mode_IPU = 0x48,
  GPIO_Mode_Out_OD = 0x14,
  GPIO_Mode_Out_PP = 0x10,
  GPIO_Mode_AF_OD = 0x1C,
  GPIO_Mode_AF_PP = 0x18
}GPIOMode_TypeDef;
```

Figure 11. Modes of GPIO

Here below provides a list of functions, find the function to read PE2.

```
void GPIO_DeInit(GPIO_TypeDef* GPIOx);
void GPIO_AFIODeInit(void);
void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct);
void GPIO_StructInit(GPIO_InitTypeDef* GPIO_InitStruct);
uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
uint16_t GPIO_ReadInputData(GPIO_TypeDef* GPIOx);
uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
uint16_t GPIO_ReadOutputData(GPIO_TypeDef* GPIOx);
void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal);
void GPIO_Write(GPIO_TypeDef* GPIOx, uint16_t PortVal);
void GPIO_PinLockConfig(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void GPIO_EventOutputConfig(uint8_t GPIO_PortSource, uint8_t GPIO_PinSource);
void GPIO_EventOutputCmd(FunctionalState NewState);
void GPIO_PinRemapConfig(uint32_t GPIO_Remap, FunctionalState NewState);
void GPIO_EXTILineConfig(uint8_t GPIO_PortSource, uint8_t GPIO_PinSource);
void GPIO_ETH_MediaInterfaceConfig(uint32_t GPIO_ETH_MediaInterface);
```

Figure 12. Some Functions Affiliated with GPIO

Before you read the port, you should first define a variable to store the returned value. Here are a number of variable types you can choose from. Select the appropriate type.

- u8 = unsigned 8-bit integer
- u16 = unsigned 16-bit integer
- u32 = unsigned 32-bit integer

Procedures:

1. Modify the program of Figure 6. Add the pin setup procedures for Key2 and set it as input with internal pull-up. You can copy the entire folder of the previous project and rename it for Experiment 2. For the experiments hereafter, you can do it similarly.
2. Check whether Key2 is pressed. If it is pressed down, turn on LED0. Otherwise, turn off LED0.
3. Repeat 2.

[Demonstration] Demonstrate that you have realized this.

[In Report] Include your source code with adequate comments and the test result.

3.3 Experiment 3: Set a GPIO as an output and drive an LED with register setting

After using the standard peripheral library, you may not know how to set registers, even if you have read through the file "stm32f10x_gpio.c".

The first step is to understand the registers that control GPIOs. Here below, we provide the register information.

Hints: During the experiment, you need to check different register information a number of times. To help you from getting confused in different register configurations, I suggest that you take an A4 paper and put all the register information on one page as neat as possible. It does not need to contain complete information of the registers. This will help you speed up in the lab. You should also have a sketch notebook in the design lab to facilitate you to put down some notes.

*****Note: They are all in "*STM32F10x Reference Manual RM0008*". In later labs, we will not provide them. If you have questions later, refer to *RM0008*.**

7.3.7 APB2 peripheral clock enable register (RCC_APB2ENR)

Address: 0x18

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait states, except if the access occurs while an access to a peripheral in the APB2 domain is on going. In this case, wait states are inserted until the access to APB2 peripheral is finished.

Note: When the peripheral clock is not active, the peripheral register values may not be readable by software and the returned value is always 0x0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										TIM11 EN	TIM10 EN	TIM9 EN	Reserved		
										rw	rw	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC3 EN	USART 1EN	TIM8 EN	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	IOPG EN	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

Bit 3 **IOPBEN:** IO port B clock enable
Set and cleared by software.
0: IO port B clock disabled
1: IO port B clock enabled

Figure 13. RCC_APB2ENR (***RM0008, page 111***)

9.2.1 Port configuration register low (GPIOx_CRL) (x=A..G)

Address offset: 0x00

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 0 .. 7)

These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table*.

In input mode (MODE[1:0]=00):

00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved

In output mode (MODE[1:0] > 00):

00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 0 .. 7)

These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table*.

00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

Figure 14. GPIOx_CRL (***RM0008, page 170)***

9.2.2 Port configuration register high (GPIOx_CRH) (x=A..G)

Address offset: 0x04

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 8 .. 15)

These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table*.

In input mode (MODE[1:0]=00):
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved

In output mode (MODE[1:0] > 00):
00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 8 .. 15)

These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table*.
00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

Figure 15. GPIOx_CRH (***RM0008, page 171***)

9.2.3 Port input data register (GPIOx_IDR) (x=A..G)

Address offset: 0x08h

Reset value: 0x0000 XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDRy:** Port input data (y= 0 .. 15)

These bits are read only and can be accessed in Word mode only. They contain the input value of the corresponding I/O port.

Figure 16. GPIOx_IDR ***(RM0008, page 171)***

9.2.4 Port output data register (GPIOx_ODR) (x=A..G)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy:** Port output data (y= 0 .. 15)

These bits can be read and written by software and can be accessed in Word mode only.

Note: For atomic bit set/reset, the ODR bits can be individually set and cleared by writing to the GPIOx_BSRR register (x = A .. G).

Figure 17. GPIOx_ODR (***RM0008, page 172***)

9.2.5 Port bit set/reset register (GPIOx_BSRR) (x=A..G)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BRy:** Port x Reset *bit y (y= 0 .. 15)*

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Reset the corresponding ODRx bit

Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BSy:** Port x *Set bit y (y= 0 .. 15)*

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Set the corresponding ODRx bit

Figure 18. GPIOx_BSRR (***RM0008, page 172***)

9.2.6 Port bit reset register (GPIOx_BRR) (x=A..G)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 Reserved

Bits 15:0 BRy: Port x Reset bit y (y= 0 .. 15)
These bits are write-only and can be accessed in Word mode only.
0: No action on the corresponding ODRx bit
1: Reset the corresponding ODRx bit

Figure 19. GPIOx_BRR (***RM0008, page173***)

Table 20. Port bit configuration table

Configuration mode		CNF1	CNF0	MODE1	MODE0	PxODR register
General purpose output	Push-pull	0	0	01 10 11 see *Table 21*		0 or 1
	Open-drain		1			0 or 1
Alternate Function output	Push-pull	1	0			don't care
	Open-drain		1			don't care
Input	Analog	0	0	00		don't care
	Input floating		1			don't care
	Input pull-down	1	0			0
	Input pull-up					1

Table 21. Output MODE bits

MODE[1:0]	Meaning
00	Reserved
01	Max. output speed 10 MHz
10	Max. output speed 2 MHz
11	Max. output speed 50 MHz

Figure 20. Port Bit Configuration Table (***RM0008, page 160***)

From Figure 11, RCC_APB2ENR, the RCC clock of Port-B enable is bit-3 of RCC_APB2ENR. You can directly set the value of RCC_APB2ENR with an instruction: "RCC->APB2ENR |= 1<<3".

"1<<3" means that setting "1" and shift it three times to the left. So, it becomes a binary number

0b0000 0000 0000 0000 0000 0000 0000 1000. When using OR "|", IOPB EN will be set to "1". This will enable the RCC clock of Port-B.

The syntax "A |= B" means "A = A|B". Similarly "A&=B" means "A=A&B". "&" is AND operation for every bit in A and B.

(More about bitwise operations: https://binaryupdates.com/bitwise-operations-in-embedded-programming)

Procedure:

1. You will need to modify the program of Figure 6. Replace the lines with codes using the register method.
2. Modify the lines of GPIOB pin-5 setting as follows.
   ~~GPIO_InitTypeDef GPIO_InitStructure;~~
   ~~RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);~~
   RCC->APB2ENR|=1<<3;
   ~~GPIO_InitStructure.GPIO_Pin=GPIO_Pin_5;~~
   ~~GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;~~
   ~~GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;~~
   ~~GPIO_Init(GPIOB, &GPIO_InitStructure);~~
   ~~GPIO_SetBits(GPIOB, GPIO_Pin_5);~~
   GPIOB->CRL &=0xFF0FFFFF;
   GPIOB->CRL |=0x00300000;
3. Modify the LED0 controlling as follows.
   ~~GPIO_ResetBits(GPIOB, GPIO_Pin_5);~~
   GPIOB->BRR = 1<<5;
   ~~GPIO_SetBits(GPIOB, GPIO_Pin_5);~~
   GPIOB->BSRR = 1<<5;
4. Compile the program and download it to the board to run.

[In Report] Understand the meaning of each line about register setting and describe them in your lab report.

[Demonstration] Show that your coding works appropriately.

3.4 Experiment 4: Read a Key from GPIO input and drive an LED with register setting

Add more functions below base on Experiment 3. In this experiment, use register setting method, NOT standard firmware library.

Procedures:

1. Based on the project in Experiment 3, you shall modify it to realize: when Key2 is pressed down, LED0 is lit; when Key2 is released, LED0 is off. Compile the program and download it to the board. *(Hints How to select Input pull-up or Input pull-down? Refer to Figure 20 ODR register)*
2. Apply the same logic of Step 1 to make Key1 control the LED1 on/off. Compile the program and download it to the board.
3. Apply the same logic of Step 1 to make Key_Up control the buzzer on/off. Compile the program and download it to the board.
4. Try to obtain the image by logic analyzer channel of the oscilloscope.

[In Report] Include coding with adequate comments, the photo of how you wire for Step 4, as well as result on the screen of the oscilloscope.

[Demonstration] Show that your coding works appropriately.

3.5 Experiment 5: Create your own library for the project board.

After Experiment 4, you should know how to initialize the keys, buzzer, and LEDs on the board. Separate those initialization steps with some subroutines and store them to the folder "Board" that you have created.

Procedures:

1. You can copy the project folder of Experiment 4 and rename it for Experiment 5.
2. Download Lab1_Empty_Coding_Files.rar from Blackboard, unzip and copy the 6 files (three .c files and three .h files) into the "Board" folder
3. Modify your main program from Experiment 4 and create three subroutine functions named **EIE3810_Key_Init()**, **EIE3810_Buzzer_Init()** and **EIE3810_LED_Init()**. You may need to revise your previous coding, as keys, buzzer, and LEDs should be initialized separately.
4. Move the initialization steps to those subroutines and store them to the files **EIE3810_Key.c**, **EIE3810_Buzzer.c** and **EIE3810_LED.c**.
5. Add your function declarations to those corresponding .h files. This is a basic skill in C programming. If you do not know how to do that, here is a reference.

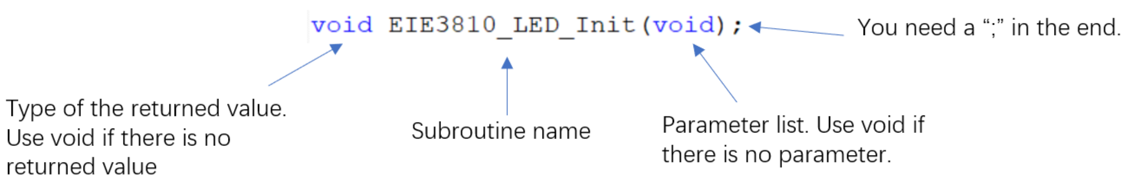


Figure 22. A sample of routine/function declaration

6. Add the three .c files in Board folder into Manage Project Items. How to do that? Click  on the toolbar.

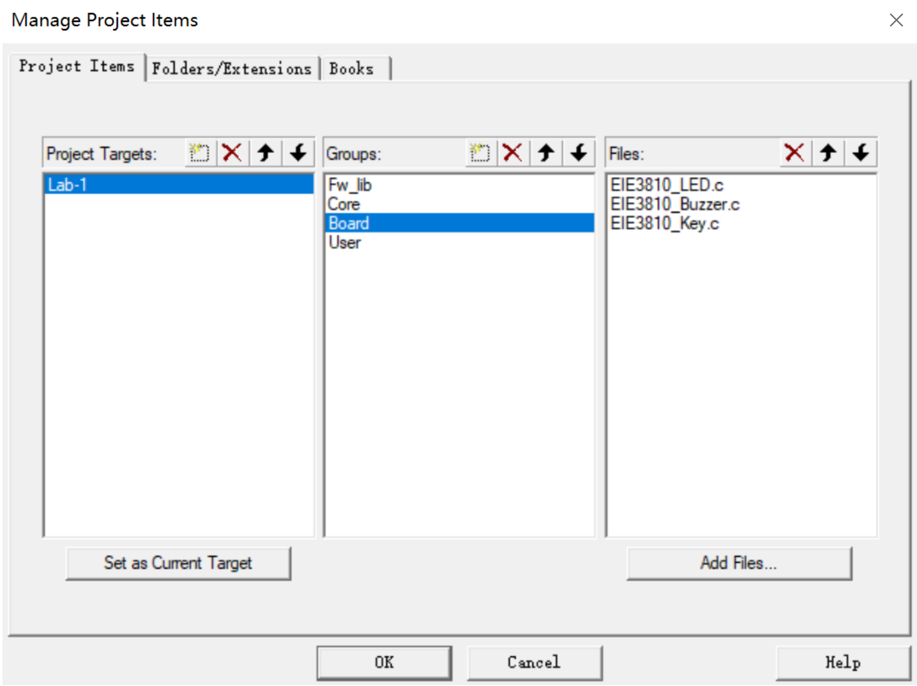


Figure 23. Manage Project Items

7. In main.c, add three lines to include the three .h files.

```c
#include "stm32f10x.h"
#include "EIE3810_LED.h"
#include "EIE3810_KEY.h"
#include "EIE3810_Buzzer.h"
```

Figure 24. #include coding.

8. Modify your program as in Figure 21 using initialization functions.

```c
#include "stm32f10x.h"
#include "EIE3810_LED.h"
#include "EIE3810_KEY.h"
#include "EIE3810_Buzzer.h"



void Delay(u32 count)
{
  u32 i;
  for (i=0;i<count;i++);
}



int main(void)
{
  EIE3810_KEY_Init();
  EIE3810_LED_Init();
  EIE3810_Buzzer_Init();

  //more codings below
```

Figure 25. Some sample code at the beginning of main.c

9. Compile the program and download it to the board. If it does not work, check problems and revise.
10. Create other subroutines you want for the board's LED, keys and buzzer, for example,
    a) to turn on LED0
    b) to turn off LED0
    c) to turn on LED1
    d) to turn off LED2
    e) to turn on Buzzer
    f) to turn off Buzzer
    g) to read Key_Up
    h) to read Key2
    i) to read Key1

    No direct register operation is allowed in main.c.

[In Report] Include coding with adequate comments.

[Demonstration] Show that your coding works appropriately.

4. Lab Report and Source Code

Submit the report softcopy and your code (complete project folder of each experiment) in zip format to Blackboard by the deadline below:

- L01: 15:00, Monday, September 23, 2024
- L02: 15:30, Wednesday, September 25, 2024

Each day of late submission will result in 10% deduction in the report and source code raw marks.