ECE3810 Microprocessor System Design Laboratory

Laboratory Report #1

Name: 胡瑞

Student ID: 122050014

Date: 2024.9.15

The Chinese University of Hong Kong, Shenzhen

During the lab, four experiments are involved for different purposes. Each experiment will be shortly introduced in the following section. Details of each experiment will be elaborated after the short introduction.

- Experiment 1: Set a GPIO as an output and drive a LED with standard peripheral library
- Experiment 2: Read a key from GPIO input and drive an LED with a standard peripheral library
- Experiment 3: Set a GPIO as an output and drive an LED with register setting
- Experiment 4: Read a Key from GPIO input and drive an LED with register setting
- Experiment 5: Create self-own library for the project board

1. Experiment 1

1.1 Design

In order to achieve the function of setting GPIO as an output, driving a LED with std peripheral library, we need to have Delay () function, finding the Clock signal that enable GPIO to work properly, and also the specific pin number and output mode. Source code is shown below.

```
#include "stm32f10x.h"

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

int main(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    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);

    while(1)
    {
        GPIO_ResetBits(GPIOB, GPIO_Pin_5);
        Delay(10000000);
        GPIO_SetBits(GPIOB, GPIO_Pin_5);
        Delay(10000000);
    }
}
```

Figure 1 Source code screenshot of experiment 1

This code is directly copied from the lab handout. To be noticed that LED0 is driven by output of PB5, which means GPIO port should be B, and the pin number should be 5.

1.2 Result

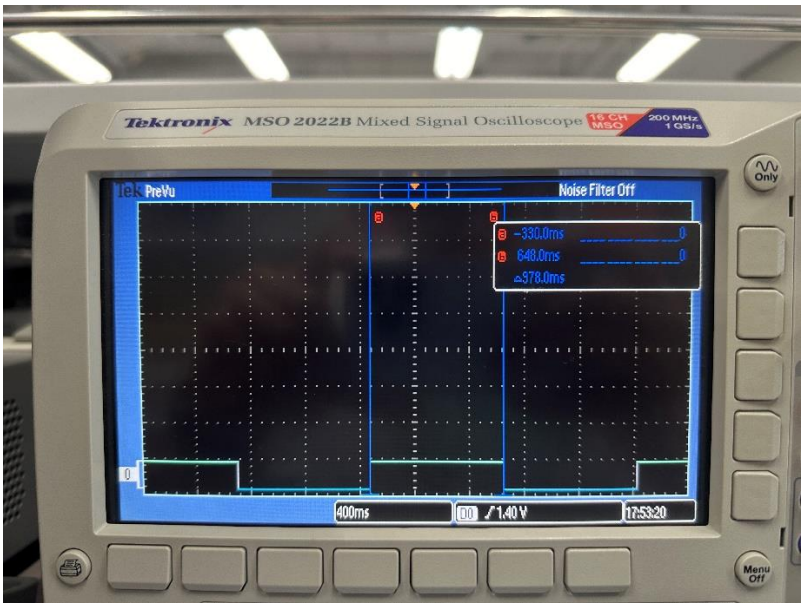

Figure 2 Logic analyzer waveform of experiment 1

As shown above in the result, the period of each high/low signal is around 978ms, which contains small error with 5% range of 1 sec.

2. Experiment 2

2.1 Design

In order to add a key function to turn on or turn off the LED0, we would need the input of one GPIO and set it to be pull-up input. Modifications are made to source code of experiment 1 based on new functions. Source code is shown below.

```c
#include "stm32f10x.h"

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

int main(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    u8 keyState;

    // Enable clock for GPIOB and GPIOE
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOE, ENABLE);

    // Configure PB5 (LED0) as output push-pull
    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);

    // Configure PE2 (Key2) as input with pull-up
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOE, &GPIO_InitStructure);

    while(1)
    {
        keyState = GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_2);
        if(keyState == 0) // Key2 is pressed (active low)
        {
            GPIO_ResetBits(GPIOB, GPIO_Pin_5); // Turn on LED0
        }
        else
        {
            GPIO_SetBits(GPIOB, GPIO_Pin_5); // Turn off LED0
        }
        Delay(100000); // Small delay to debounce
    }
}
```

Figure 3 Source code screenshot of experiment 2

The reason to select pull-up mode for Key2 is because we need to coordinate with the active-low mechanism of LED0. In pull-up mode, when the key is been pressed, the output signal will be logic LOW, driving the LED0 to be on; on the other hand, when the key is not pressed, the output signal will be logic HIGH, which turn off the LED0.

Another function selection is that we need to select “GPIO_ReadInputDataBit” function to read the bit input of PE2, which is the Key input. The reason for this selection is because PE2 itself is set to be input instead of output, and also the input is a single bit instead of several bits or a whole byte.

To be notice that we create a u8 keystate (unsigned 8-bit integer) to store the return value of “GPIO_ReadInputDataBit” function, because the return value itself is a unit8_t.

2.2 Result

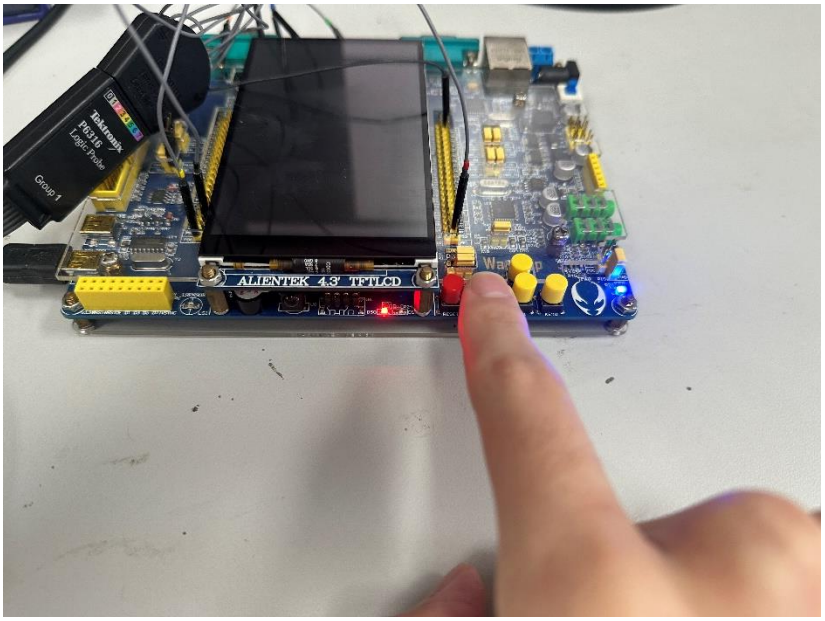

Figure 4 Pressing the Key2 will turn on the LED0

3. Experiment 3

3.1 Design

Using registers instead of peripheral library to achieve the function of experiment 1 requires us to match some related information of each compound with the content provided by the device manual. The source code is shown below.

```
1  #include "stm32f10x.h"
2
3  void Delay(u32 count)
4  {
5      u32 i = 0;
6      for(i=0; i<count; i++);
7  }
8
9  int main(void)
10 {
11     // Enable clock for GPIOB
12     RCC->APB2ENR |= 1<<3;
13
14     // Configure PB5 as output push-pull
15     GPIOB->CRL &= 0xFF0FFFFF;
16     GPIOB->CRL |= 0x00300000;
17
18     while(1)
19     {
20         GPIOB->BRR = 1<<5;  // Turn on LED0
21         Delay(10000000);
22         GPIOB->BSRR = 1<<5; // Turn off LED0
23         Delay(10000000);
24     }
25 }
26
```

Figure 5 Source code screenshot of experiment 3

For the 12th line code, it enables the clock for GPIOB. Because APB2ENR is the APB2 peripheral clock enable register, and bit 3 in this register corresponds to GPIOB, setting this bit enables the clock for GPIOB.

For the 15th line code, CRL is the Configuration Register Low for GPIOB. This operation clears the configuration bits for PB5 (bits 20-23), and then 0xFF0FFFFF is a mask that keeps all other bits unchanged except for the bit 5 in order to control the signal H/L.

For the 16th line code, it sets the configuration for PB5. 0x00300000 sets bits 20-23 to be 0011. In the CRL register, each pin uses 4 bits. For PB5, these are bits 20-23. 0011 configures PB5 as an output with max speed of 50 MHz

Within the while loop, the 20th line code is related to the BRR. The BRR is the Bit Reset Register, which is used to reset a certain bit. This operation sets bit 5 in the BRR, which resets pin PB5. This implementation is used for the active-LOW mechanism of LED that connects to PB5, which turns on when the signal is LOW.

After a certain delay, in 22th line code, BSRR is used. BSRR is the Bit Set/Reset Register. This operation sets bit 5 in the BSRR, which sets pin PB5, turning off the LED that connects to PB5. Basically line 20 and line 22 are used to control the on and off of LED.

4. Experiment 4

4.1 Design

Using registers instead of peripheral library to achieve the function of experiment 2 requires us to match some related information of each compound with the content provided by the device manual. The source code is shown below.

```c
#include "stm32f10x.h"

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

int main(void)
{
    // Enable clock for GPIOA, GPIOB, and GPIOE
    RCC->APB2ENR |= (1<<2) | (1<<3) | (1<<6);

    // Configure PB5 (LED0) as output push-pull
    GPIOB->CRL &= 0xFF0FFFFF;
    GPIOB->CRL |= 0x00300000;

    // Configure PE5 (LED1) as output push-pull
    GPIOE->CRL &= 0xFF0FFFFF;
    GPIOE->CRL |= 0x00300000;

    // Configure PB8 (Buzzer) as output push-pull
    GPIOB->CRH &= 0xFFFFFFF0;
    GPIOB->CRH |= 0x00000003;

    // Configure PE2 (Key2) as input pull-up
    GPIOE->CRL &= 0xFFFFF0FF;
    GPIOE->CRL |= 0x00000800;
    GPIOE->ODR |= (1<<2);

    // Configure PE3 (Key1) as input pull-up
    GPIOE->CRL &= 0xFFFF0FFF;
    GPIOE->CRL |= 0x00008000;
    GPIOE->ODR |= (1<<3);

    // Configure PA0 (Key_Up) as input pull-down
    GPIOA->CRL &= 0xFFFFFFF0;
    GPIOA->CRL |= 0x00000008;
    GPIOA->ODR &= ~(1<<0);
```

Figure 6 Source code screenshot 1 of experiment 4

```c
    while(1)
    {
        // Key2 controls LED0
        if((GPIOE->IDR & (1<<2)) == 0)
        {
            GPIOB->BRR = 1<<5;  // Turn on LED0
        }
        else
        {
            GPIOB->BSRR = 1<<5; // Turn off LED0
        }

        // Key1 controls LED1
        if((GPIOE->IDR & (1<<3)) == 0)
        {
            GPIOE->BRR = 1<<5;  // Turn on LED1
        }
        else
        {
            GPIOE->BSRR = 1<<5; // Turn off LED1
        }

        // Key_Up controls Buzzer
        if(GPIOA->IDR & (1<<0))
        {
            GPIOB->BSRR = 1<<8; // Turn on Buzzer
        }
        else
        {
            GPIOB->BRR = 1<<8;  // Turn off Buzzer
        }

        Delay(100000); // Small delay to debounce
    }
}
```

Figure 7 Source code screenshot 1 of experiment 4

The basic logic of experiment 4 is similar to experiment 3, but we add another LED output, a buzzer output, and three keys input, and also connect inputs with outputs. The first step is to enable the clock of each output, and they are all control by APB2ENR, but with different bits. Next step is to configure all inputs and outputs to be modes that we want. For LEDs and buzzer, we want them to be push-pull mode; for Key 1 and Key 2, we want them to be pull-up mode because we need to reverse the input signal and coordinate with the active – LOW mechanism of LED; fir Key_Up, we want it to be pull down mode, which output HIGH and pressed, and this is because buzzer has active – HIGH working mechanism. We want that pressing the Key_Up will activate the buzzer.

One thing to be noticed that for the register used for configure the buzzer, we use CRH instead of CRL because the buzzer has its bin located at PB8 GPIO port, which should be controlled by CRH, Configure Register High. On the other hand, LED1 and LED2 are located at PB5 and PE5, which are low pins (pins 0 to 7) and should be controlled by CRL, Configure Register Low.

For the main connection loop, GPIOE->IDR & (1<<2) and GPIOE->IDR & (1<<3) read the input state of Key2 and Key1. GPIOA->IDR & (1<<0) reads the input state of Key_Up. GPIOB->BRR = 1<<5 and GPIOB->BSRR = 1<<5 control LED0. GPIOE->BRR = 1<<5 and GPIOE->BSRR = 1<<5 control LED1. GPIOB->BSRR = 1<<8 and GPIOB->BRR = 1<<8 control the Buzzer.

4.2 Result

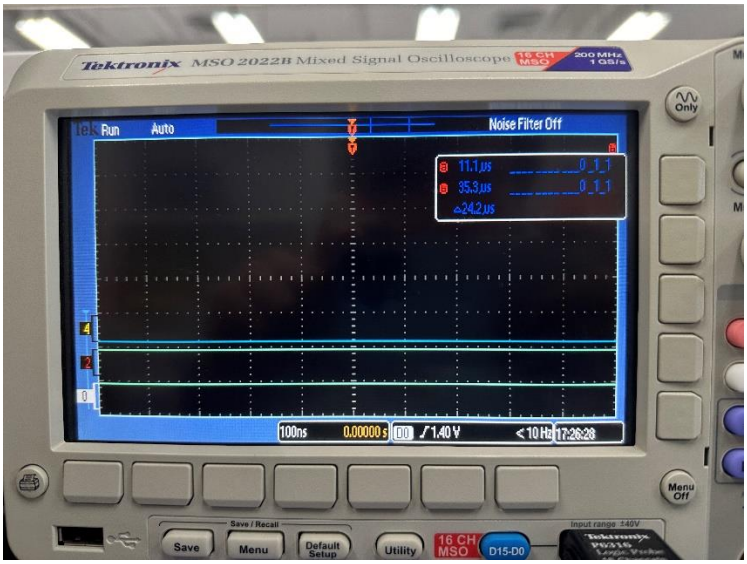

Figure 8 Logic analyzer waveform (No key is pressed)

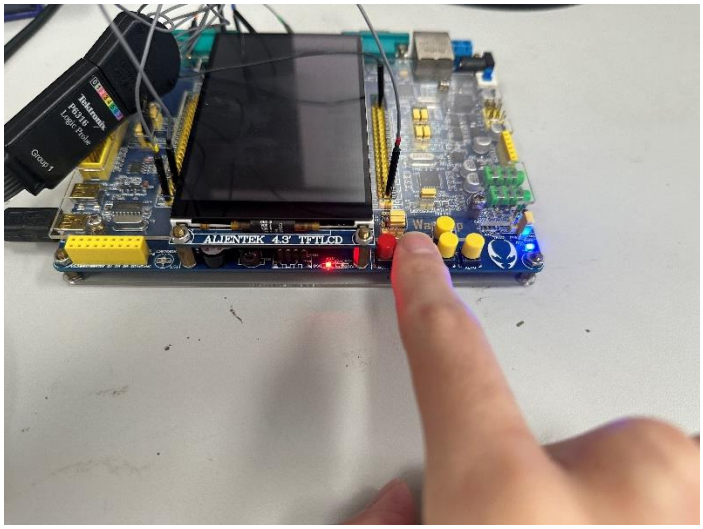

Figure 9 Key 2 is pressed, LED0 is on

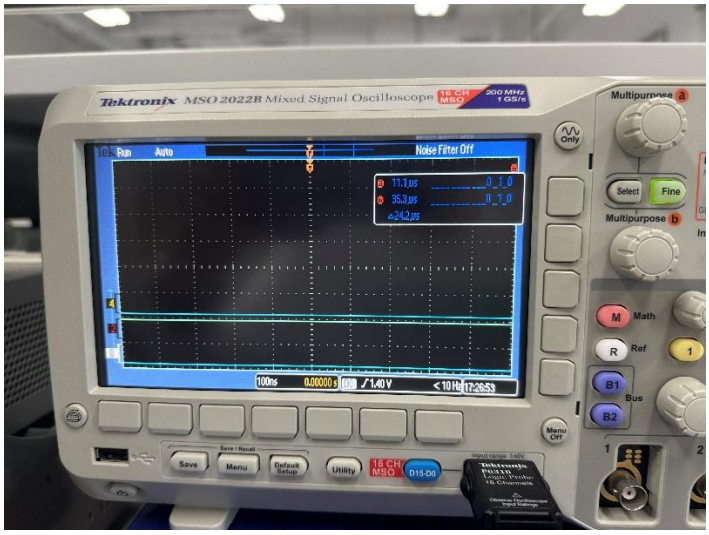

Figure 10 Logic analyzer waveform (Key2 is pressed)

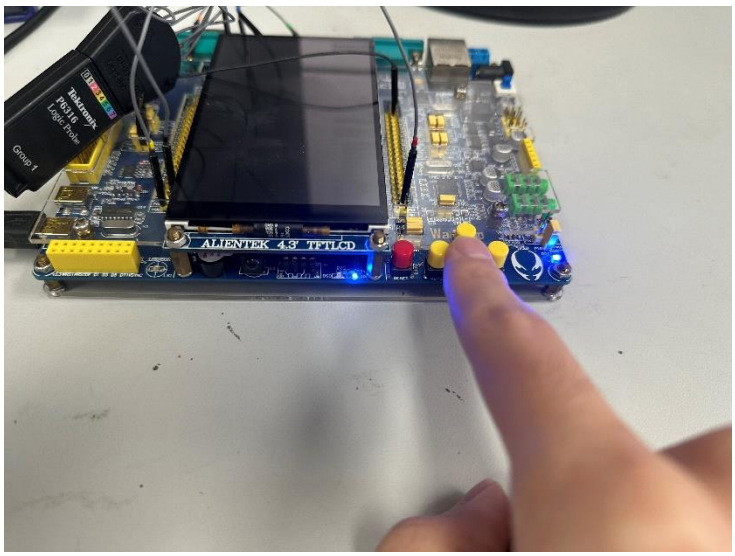

Figure 11 Key1 is pressed, LED1 is on

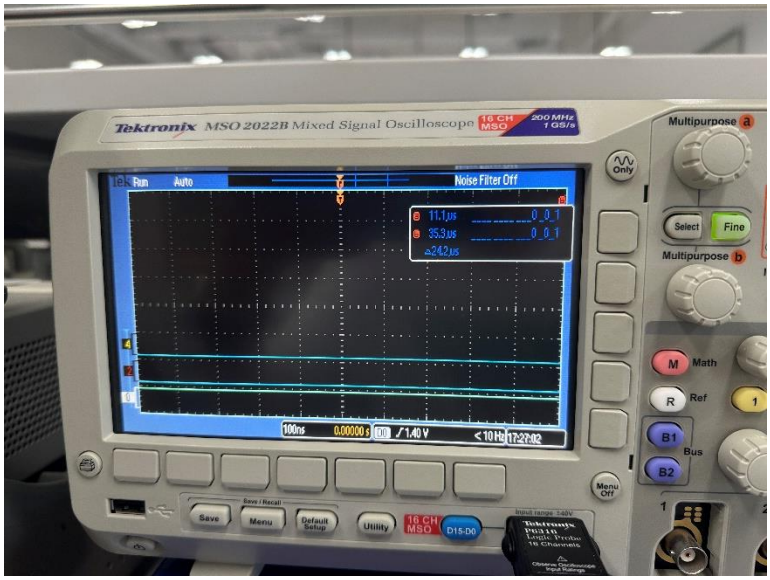

Figure 12 Logic analyzer waveform (Key1 is pressed)

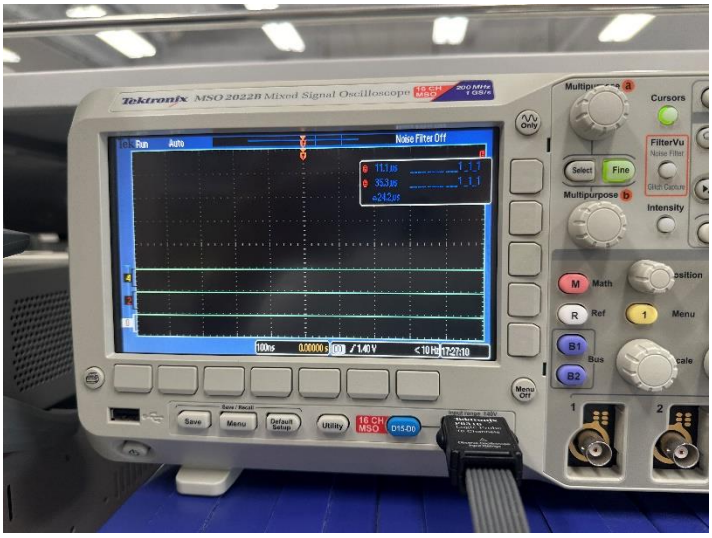

Figure 13 Logic analyzer waveform (Key_Up is pressed)

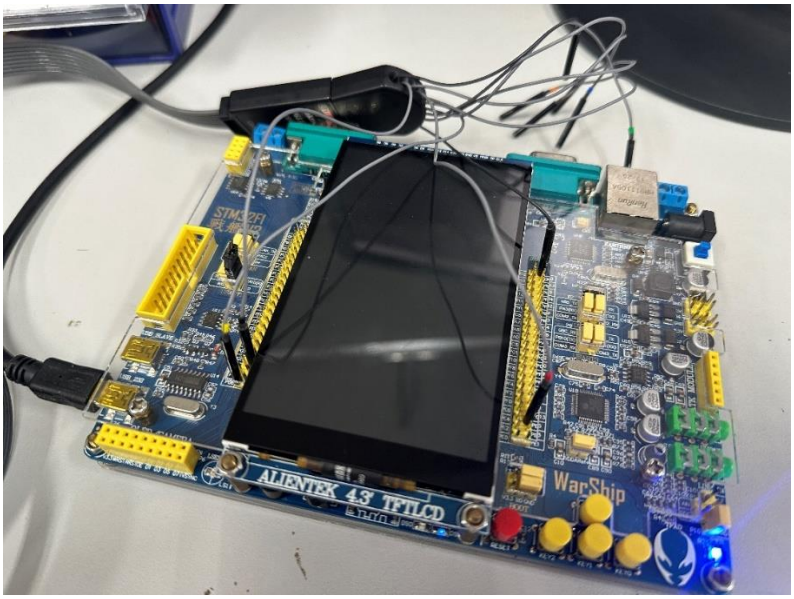

Figure 14 Specific wiring

5. Experiment 5

5.1 Design

To create an own library that contains these functions, we need to sperate each function into a new file, and also another header file for reference in the main code. The source codes are shown below

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

void EIE3810_Buzzer_Init(void)
{
    // Enable clock for GPIOB
    RCC->APB2ENR |= (1<<3);

    // Configure PB8 (Buzzer) as output push-pull
    GPIOB->CRH &= 0xFFFFFFF0;
    GPIOB->CRH |= 0x00000003;
}

void EIE3810_Buzzer_On(void)
{
    GPIOB->BSRR = 1<<8;
}

void EIE3810_Buzzer_Off(void)
{
    GPIOB->BRR = 1<<8;
}
```

Figure 15 Buzzer_Init file

```c
#ifndef __EIE3810_BUZZER_H
#define __EIE3810_BUZZER_H

#include "stm32f10x.h"

void EIE3810_Buzzer_Init(void);
void EIE3810_Buzzer_On(void);
void EIE3810_Buzzer_Off(void);

#endif
```

Figure 16 Buzzer head file

```c
#include "stm32f10x.h"
#include "EIE3810_Key.h"

void EIE3810_Key_Init(void)
{
    // Enable clock for GPIOA and GPIOE
    RCC->APB2ENR |= (1<<2) | (1<<6);

    // Configure PE2 (Key2) as input pull-up
    GPIOE->CRL &= 0xFFFFF0FF;
    GPIOE->CRL |= 0x00000800;
    GPIOE->ODR |= (1<<2);

    // Configure PE3 (Key1) as input pull-up
    GPIOE->CRL &= 0xFFFF0FFF;
    GPIOE->CRL |= 0x00008000;
    GPIOE->ODR |= (1<<3);

    // Configure PA0 (Key_Up) as input pull-down
    GPIOA->CRL &= 0xFFFFFFF0;
    GPIOA->CRL |= 0x00000008;
    GPIOA->ODR &= ~(1<<0);
}

u8 EIE3810_Read_Key2(void)
{
    return (GPIOE->IDR & (1<<2)) == 0;
}

u8 EIE3810_Read_Key1(void)
{
    return (GPIOE->IDR & (1<<3)) == 0;
}

u8 EIE3810_Read_Key_Up(void)
{
    return (GPIOA->IDR & (1<<0)) != 0;
}
```

Figure 17 Key_Init file

```c
#ifndef __EIE3810_KEY_H
#define __EIE3810_KEY_H

#include "stm32f10x.h"

void EIE3810_Key_Init(void);
u8 EIE3810_Read_Key2(void);
u8 EIE3810_Read_Key1(void);
u8 EIE3810_Read_Key_Up(void);

#endif
```

Figure 18 Key header file

```c
#include "stm32f10x.h"
#include "EIE3810_Led.h"

void EIE3810_LED_Init(void)
{
    // Enable clock for GPIOB and GPIOE
    RCC->APB2ENR |= (1<<3) | (1<<6);

    // Configure PB5 (LED0) as output push-pull
    GPIOB->CRL &= 0xFF0FFFFF;
    GPIOB->CRL |= 0x00300000;

    // Configure PE5 (LED1) as output push-pull
    GPIOE->CRL &= 0xFF0FFFFF;
    GPIOE->CRL |= 0x00300000;
}

void EIE3810_LED0_On(void)
{
    GPIOB->BRR = 1<<5;
}

void EIE3810_LED0_Off(void)
{
    GPIOB->BSRR = 1<<5;
}

void EIE3810_LED1_On(void)
{
    GPIOE->BRR = 1<<5;
}

void EIE3810_LED1_Off(void)
{
    GPIOE->BSRR = 1<<5;
}
```

Figure 19 LED_Init file

```c
#ifndef __EIE3810_LED_H
#define __EIE3810_LED_H

#include "stm32f10x.h"

void EIE3810_LED_Init(void);
void EIE3810_LED0_On(void);
void EIE3810_LED0_Off(void);
void EIE3810_LED1_On(void);
void EIE3810_LED1_Off(void);

#endif
```

Figure 20 LED header file

```c
#include "stm32f10x.h"
#include "EIE3810_Key.h"
#include "EIE3810_Buzzer.h"
#include "EIE3810_Led.h"

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

int main(void)
{
    EIE3810_Key_Init();
    EIE3810_Buzzer_Init();
    EIE3810_LED_Init();

    while(1)
    {
        // Key2 controls LED0
        if(EIE3810_Read_Key2())
        {
            EIE3810_LED0_On();
        }
        else
        {
            EIE3810_LED0_Off();
        }

        // Key1 controls LED1
        if(EIE3810_Read_Key1())
        {
            EIE3810_LED1_On();
        }
        else
        {
            EIE3810_LED1_Off();
        }

        // Key_Up controls Buzzer
        if(EIE3810_Read_Key_Up())
        {
            EIE3810_Buzzer_On();
        }
        else
        {
            EIE3810_Buzzer_Off();
        }

        Delay(100000); // Small delay to debounce
    }
}
```

Figure 21 main.c file

Each initialization function enables the necessary clock and configures the appropriate GPIO pins. Also, each component has its own initialization function and control functions, referring to corresponding header file. In the main () function, these initialization functions are called at the beginning, replacing the previous initialization code. The main file now only contains header file reference and directly use defined functions to control signal and device without any direct register operation. The main loop remains the same, controlling the LEDs and buzzer based on the key inputs.

x. Conclusion

In this lab, we have conducted five experiments with different purposes. We have learnt set a GPIO as an output and drive a LED with standard peripheral library; read a key from GPIO input and drive an LED with a standard peripheral library; set a GPIO as an output and drive an LED with register setting; read a Key from GPIO input and drive an LED with register setting and create self-own library for the project board