《微机原理与接口技术》

实验指导书

实验二 汇编和 C 语言的交互编程及 GPIO 的应用

“微机原理与接口技术”课程教学团队

北京航空航天大学

仪器科学与光电工程学院

2022 年 11 月

一、实验目的

1. 了解 ARM 汇编语言和 C 语言交互编程和调用的方法和机制。
2. 掌握汇编和 C 语言变量和函数调用时的规则和注意事项。
3. 掌握多个源文件构成工程项目的软件开发模式。
4. 掌握 STM43H745 的 GPIO 的使用方法。
5. 学习并使用延时程序，并利用板载按键实现对 LED 的控制。

二、实验设备

1. PC 计算机；
2. Keil for ARM(MDK)开发环境；
3. NUCLEO 开发板，或 Disco 开发板，主要查看相应的开发板电路图。

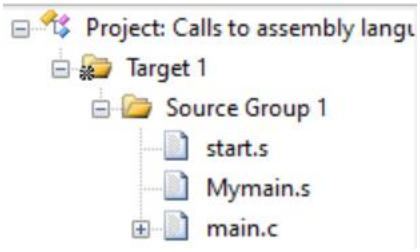
三、实验内容

1. C 程序调用汇编程序；
2. 汇编程序调用 C 程序；
3. GPIO 基础实验；
4. GPIO 提高实验。

四、实验步骤

4.1 C 程序调用汇编程序

1. 运行 Keil uVision5，按照实验一的项目创建方式建立一个新的项目，并在其中添加两个汇编文件（start.s 和 Mymain.s）和一个.c 文件（main.c），其中 start.s 文件作为启动文件，Mymain.s 文件保存编写的汇编程序，main.c 文件中写入 main 函数并在 main 函数中调用汇编程序，将通过 C 调用汇编实现字符串的拼接功能。



2. start.s 文件的内容如图所示，图中横线处为待填写的内容

```
1  Stack_Size      EQU     0x00000400
2
3                  AREA    STACK, NOINIT, READWRITE, ALIGN=3
4  Stack_Mem       SPACE   Stack_Size
5  __initial_sp
6                  PRESERVE8
7                  THUMB
8
9  ; Vector Table Mapped to Address 0 at Reset
10                 AREA    RESET, DATA, READONLY
11                 EXPORT  __Vectors
12
13 __Vectors       DCD     __initial_sp                ; Top of Stack
14                 DCD     Reset_Handler               ; Reset Handler
15
16
17                 AREA Init, CODE, READONLY
18                 xxxxxxxxxx
19                 EXPORT  Reset_Handler
20                 ENTRY
21 Reset_Handler
22
23                 LDR     R0, xxxxxxx
24                 BX      R0
25                 END
```

3.Mymain.s 文件的内容如图所示，图中横线处为待填写的内容，并在矩形框中补充完整 my_strcat 代码段，通过汇编程序实现字符串拼接功能。

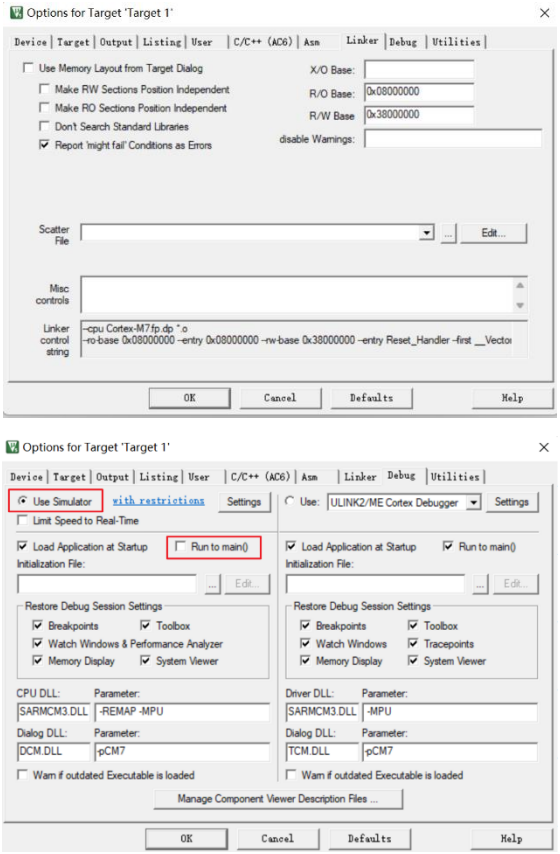
```
1                PRESERVE8
2                THUMB
3                AREA     |.text|,CODE,READONLY
4        IMPORT  |Image$$RO$$Limit|
5        IMPORT  |Image$$RW$$Base|
6        IMPORT  |Image$$ZI$$Base|
7        IMPORT  |Image$$ZI$$Limit|
8                xxxxxxxxxxxx
9   xxxxxx       FUNCTION
10               LDR R0,=|Image$$RO$$Limit|
11               LDR R1,=|Image$$RW$$Base|
12               LDR R3,=|Image$$ZI$$Base|
13  Init_RW
14               CMP R1, R3
15               LDRCC R2, [R0], #4
16               STRCC R2, [R1], #4
17               BCC Init_RW
18               LDR R1,=|Image$$ZI$$Limit|
19               MOV R2, #0
20  Init_ZI
21               CMP R3, R1
22               STRCC R2, [R3], #4
23               BCC Init_ZI
24
25               xxxxxxxxxx
26               xxxxxxxxxx
27  deadloop     B   deadloop
28               ENDFUNC
29
30               xxxxxxxxxxx
31  my_strcat    FUNCTION
32
33
34
35
36
37               ENDFUNC
38               NOP
39               END
```

4.main.c 文件的内容如图所示，first_sentence 字符数组中保留一句话的前半句，second_sentence 是指向后半句字符串常量的指针，通过调用 my_stract 汇编程序将 second_sentence 指向的字符串添加在 first_sentence 字符串数组的末尾，从而实现字符串的拼接，在横线 1 处完成 my_stract 函数的声明，并在横线 2 处调用 my_stract 函数实现拼接。

✧ 注：first_sentence 初始化时空间足够，在字符串的末尾编译器会自动添加空字符，空字符在汇编程序中的值为 0）

```
1  #include <stdio.h>
2
3  #define  SIZE     50
4
5
6
7  int main (void) {
8    const char * second_sentence = "but better carries it.";
9    char  first_sentence [SIZE] = "Good is good, ";
10
11   return 0;
12 }
13
```

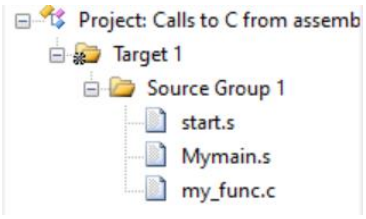
5. 程序编写完成后进行编译并调试，注意 Options>>Linker 选项卡的配置，如图所示，并在 Debug 选项卡中选中 Use simulator，为观察汇编程序的每一步运行，可以去掉 Debug 选项卡中的 Run to main()，否则调试时会直接跳到.c 文件中的 main 函数并向下运行。



6. 依据课上所学知识和教材将代码补充完整，观察寄存器和相关存储器在运行过程中和调用函数时的变化，以及完成拼接后 first_sentence 字符数组的内容，进行详细分析并撰写实验报告。

4.2 汇编程序调用 C 程序

1. 运行 Keil uVision5，按照实验一的项目创建方式建立一个新的项目，并在其中添加两个汇编文件 start.s 和 Mymain.s 和一个.c 文件 myfunc.c，其中 start.s 文件作为启动文件，Mymain.s 文件保存编写的汇编程序，my_func.c 文件定义 my_func 函数，并在 Mymain.s 中的汇编程序调用 C 程序。



2. start.s 文件的内容如图所示，图中横线处为待填写的内容。

```
Stack_Size      EQU     0x00000400

                AREA    STACK, NOINIT, READWRITE, ALIGN=3
Stack_Mem       SPACE   Stack_Size
__initial_sp
                PRESERVE8
                THUMB

; Vector Table Mapped to Address 0 at Reset
                AREA    RESET, DATA, READONLY
                EXPORT  __Vectors

__Vectors       DCD     __initial_sp                    ; Top of Stack
                DCD     Reset_Handler                   ; Reset Handler


                AREA Init, CODE, READONLY
                xxxxxxxxxx
                EXPORT  Reset_Handler
                ENTRY
Reset_Handler

                LDR     R0, xxxxxxx
                BX      R0
                END
```

3.Mymain.s 文件的内容如图所示，图中横线处为待填写的内容。

```
1                PRESERVE8
2                THUMB
3                AREA     |.text|,CODE,READONLY
4        IMPORT  |Image$$RO$$Limit|
5        IMPORT  |Image$$RW$$Base|
6        IMPORT  |Image$$ZI$$Base|
7        IMPORT  |Image$$ZI$$Limit|
8                xxxxxxxxxx
9    xxxxxx      FUNCTION
10               LDR R0,=|Image$$RO$$Limit|
11               LDR R1,=|Image$$RW$$Base|
12               LDR R3,=|Image$$ZI$$Base|
13   Init_RW
14               CMP R1, R3
15               LDRCC R2, [R0], #4
16               STRCC R2, [R1], #4
17               BCC Init_RW
18               LDR R1,=|Image$$ZI$$Limit|
19               MOV R2, #0
20   Init_ZI
21               CMP R3, R1
22               STRCC R2, [R3], #4
23               BCC Init_ZI
24
25
26
27
28
29
30
31
32   deadloop    B   deadloop
33               NOP
34
35   Class       DCD     xxxxxxxxxx
36
37               AREA    DATA, DATA , xxxxxxxx
38   Result      DCD     0
39               END
```

在矩形框中编写如下程序：

（1） 实验过程中赋值时仅允许使用 6 个寄存器 R0~R5，首先将寄存器赋初值 R0~R5 的初始值分别为 1~6，在程序结束时应保证 6 个寄存器的初始值不变（不允许再重新赋初值）。

（2） 将你的学号分为 4 个部分，每个部分两位数，如 20374210->20-37-42-10，将这 4 个两位数保存在 4 个寄存器中将会作为前 4 个参数传递至 C 函数 my_func 中。

（3） 定义 Class 标号用于存放你的专业班号如 201713，并将班号读入寄存器内，将作为第 5 个参数传递给 C 函数。

（4） 调用 C 函数，将 C 函数的输出结果放在 Result 中保存。

（5） 恢复现场，复原 6 个寄存器的初始值，观察寄存器初始值在程序开始时和结束时是否发生变化。

```
1                PRESERVE8
2                THUMB
3                AREA     |.text|,CODE,READONLY
4       IMPORT  |Image$$RO$$Limit|
5       IMPORT  |Image$$RW$$Base|
6       IMPORT  |Image$$ZI$$Base|
7       IMPORT  |Image$$ZI$$Limit|
8                xxxxxxxxxx
9   xxxxxx       FUNCTION
10               LDR R0,=|Image$$RO$$Limit|
11               LDR R1,=|Image$$RW$$Base|
12               LDR R3,=|Image$$ZI$$Base|
13  Init_RW
14               CMP R1, R3
15               LDRCC R2, [R0], #4
16               STRCC R2, [R1], #4
17               BCC Init_RW
18               LDR R1,=|Image$$ZI$$Limit|
19               MOV R2, #0
20  Init_ZI
21               CMP R3, R1
22               STRCC R2, [R3], #4
23               BCC Init_ZI
24
25
26
27
28
29
30
31
32  deadloop     B   deadloop
33               NOP
34
35  Class        DCD     xxxxxxxxxx
36
37               AREA    DATA, DATA , xxxxxxxx
38  Result       DCD     0
39               END
```

4.my_func.c 文件的内容如图所示，将会返回表达式的结果。

```
1  int my_func (int , int , int , int , int);
2
3  int my_func (int x1 , int x2 , int x3 , int x4 , int x5)  {
4    return  x1*x2*x3*x4 + x5;
5  }
```

以 20374210，201713 为例，my_func 将返回 20*37*42*10+201713

5. 依据课上所学知识和教材将代码补充完整，观察寄存器在运行过程中和调用函数时的变化，以及保存在 Result 中的结果是否正确，进行详细分析，在实验报告中计算理论结果并验证实际输出结果是否正确，撰写实验报告。

4.3 GPIO 基础实验

1. 了解 GPIO

GPIO(General Purpose Input Output)通用输入输出，是计算机系统的一种典型外设，可以实现输入输出等功能。STM32H745 系列芯片共有 11 组 GPIO 外设端口，即 GPIOA~GPIOK，每组 GPIO 外设端口共有 16 个引脚，定义为 GPIO_PIN_0~GPIO_PIN_15。

Nucleo 开发板：

实验平台由意法半导体公司（ST）的 STM32H745 Nucleo 开发板和自研底板组成。

本次实验使用的 LED 与引脚的对应关系：

- 开发板 USER LED1 绿色：PB0,
- 开发板 USER LED2 黄色：PE1,
- 开发板 USER LED3 红色：PB14,
- 开发板上按键 B1 蓝色：PC13 （特别提醒，另外一个黑色为复位按键!!!）
- 底板上 LED1 绿色：PE11,
- 底板上 LED2 蓝色：PB11.
- 底板上按键 KEY1:

底板上按键 KEY2:

Disco 开发板：

实验平台为意法半导体公司（ST）的 STM32H745 Discovery 开发板。

本次实验使用的 LED 与引脚的对应关系：

- 开发板 USER LED1：PB0 LED1：PJ2
- 开发板 USER LED2：PI13
- 开发板上按键 B1 蓝色：PC13 （特别提醒，另外一个黑色为复位按键!!!）

与 GPIO 相关的函数存放于 HAL 库的 GPIO 函数库中，打开 stm32h7xx_hal_gpio.h 文件可以看到 GPIO 引脚的宏定义，在 stm32h745xx.h 文件中可以看到 GPIO 端口的宏定义。

```
/** @defgroup GPIO_pins_define  GPIO pins define
  * @{
  */
#define GPIO_PIN_0                 ((uint16_t)0x0001)  /* Pin 0 selected    */
#define GPIO_PIN_1                 ((uint16_t)0x0002)  /* Pin 1 selected    */
#define GPIO_PIN_2                 ((uint16_t)0x0004)  /* Pin 2 selected    */
#define GPIO_PIN_3                 ((uint16_t)0x0008)  /* Pin 3 selected    */
#define GPIO_PIN_4                 ((uint16_t)0x0010)  /* Pin 4 selected    */
#define GPIO_PIN_5                 ((uint16_t)0x0020)  /* Pin 5 selected    */
#define GPIO_PIN_6                 ((uint16_t)0x0040)  /* Pin 6 selected    */
#define GPIO_PIN_7                 ((uint16_t)0x0080)  /* Pin 7 selected    */
#define GPIO_PIN_8                 ((uint16_t)0x0100)  /* Pin 8 selected    */
#define GPIO_PIN_9                 ((uint16_t)0x0200)  /* Pin 9 selected    */
#define GPIO_PIN_10                ((uint16_t)0x0400)  /* Pin 10 selected   */
#define GPIO_PIN_11                ((uint16_t)0x0800)  /* Pin 11 selected   */
#define GPIO_PIN_12                ((uint16_t)0x1000)  /* Pin 12 selected   */
#define GPIO_PIN_13                ((uint16_t)0x2000)  /* Pin 13 selected   */
#define GPIO_PIN_14                ((uint16_t)0x4000)  /* Pin 14 selected   */
#define GPIO_PIN_15                ((uint16_t)0x8000)  /* Pin 15 selected   */
#define GPIO_PIN_All               ((uint16_t)0xFFFF)  /* All pins selected */

#define GPIO_PIN_MASK              (0x0000FFFFU) /* PIN mask for assert test */
/**
```

```
#define GPIOA                ((GPIO_TypeDef *) GPIOA_BASE)
#define GPIOB                ((GPIO_TypeDef *) GPIOB_BASE)
#define GPIOC                ((GPIO_TypeDef *) GPIOC_BASE)
#define GPIOD                ((GPIO_TypeDef *) GPIOD_BASE)
#define GPIOE                ((GPIO_TypeDef *) GPIOE_BASE)
#define GPIOF                ((GPIO_TypeDef *) GPIOF_BASE)
#define GPIOG                ((GPIO_TypeDef *) GPIOG_BASE)
#define GPIOH                ((GPIO_TypeDef *) GPIOH_BASE)
#define GPIOI                ((GPIO_TypeDef *) GPIOI_BASE)
#define GPIOJ                ((GPIO_TypeDef *) GPIOJ_BASE)
#define GPIOK                ((GPIO_TypeDef *) GPIOK_BASE)
```

在使用 GPIO 前，需要对 GPIO 进行初始化配置，在 stm32h7xx_hal_gpio.h 文件中可以看到配置结构体，如下所示，

```
typedef struct
{
  uint32_t Pin;       /*!< Specifies the GPIO pins to be configured.
                           This parameter can be any value of @ref GPIO_pins_define */

  uint32_t Mode;      /*!< Specifies the operating mode for the selected pins.
                           This parameter can be a value of @ref GPIO_mode_define */

  uint32_t Pull;      /*!< Specifies the Pull-up or Pull-Down activation for the selected pins.
                           This parameter can be a value of @ref GPIO_pull_define */

  uint32_t Speed;     /*!< Specifies the speed for the selected pins.
                           This parameter can be a value of @ref GPIO_speed_define */

  uint32_t Alternate;  /*!< Peripheral to be connected to the selected pins.
                            This parameter can be a value of @ref GPIO_Alternate_function_selection */
} GPIO_InitTypeDef;
```

在 GPIO_InitTypeDef 结构体中，可以设置 GPIO 的引脚（Pin），引脚的工作模式（Mode，配置输入输出等），用于输入模式的上拉下拉选择（Pull），引脚最大输出速度（Speed），引脚复用（Alternate）等。在声明结构体的下方，可以找到其各成员取值范围的宏定义。

2. 了解 GPIO 相关函数

（1）__HAL_RCC_GPIOX_CLK_ENABLE();

在使用 GPIO 之前需要先使能相应端口的时钟，X 即组号或端口编号，可以选择的范围 A~K，__HAL_RCC_GPIOX_CLK_ENABLE 的定义可以在 stm32h7xx_hal_rcc.h 中找到，不使用某个端口时，可以通过 __HAL_RCC_GPIOX_CLK_DISABLE 关闭 X 端口的时钟，实现降低功耗的目的。

（2）void HAL_GPIO_Init(GPIO_TypeDef*GPIOx, GPIO_InitTypeDef *GPIO_Init);

HAL_GPIO_Init 函数用于 GPIO 的初始化，函数的定义可以在 stm32h7xx_hal_gpio.c 中找到，输入的第一个参数代表端口号（GPIOA~GPIOK），第二个参数为上文提到的 GPIO 配置结构体，结构体中的成员可以自行赋值，例如，对 A 端口第 0 个引脚初始化：

```
GPIO_InitTypeDef  GPIO_InitStruct;

GPIO_InitStruct.Pin = GPIO_PIN_0;                 /* Pin0 */
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;       /* 推挽输出 */
GPIO_InitStruct.Pull = GPIO_NOPULL;                /* 无上拉和下拉电阻 */
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH; /* GPIO速度等级最高 */

HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

（3）void HAL_GPIO_WritePin(GPIO_TypeDef*GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)；

HAL_GPIO_WritePin 函数用于设置 GPIO 引脚的值，输入的第一个参数代表端口号（GPIOA~GPIOK），第二个参数代表端口上的引脚，第三个参数指定要写入的值，可选择 GPIO_PIN_RESET 清除端口 PIN 或 GPIO_PIN_SET 设置端口 PIN。

（4）GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)；

HAL_GPIO_ReadPin 函数用于读取 GPIO 引脚的输出值，返回值 GPIO_PinState。

3. GPIO 基础实验

GPIO 基础实验将分为以下 5 个部分逐步进行（注意每个步骤的代码均需要保留）：

（1） 通过配置 GPIO，点亮 LED 灯（完成此步骤后请将点亮 LED，Nucleo 板三个 LED，Disco 板二个 LED，灯拍照并附在实验报告中）；

本次实验用到的 LED 灯共有 3 个，相关的宏定义可以在 stm32h7xx_nucleo.h 文件中找到，如图所示，要求在 CM7 的 main.c 里定义 Nucleo 板三个 LED，Disco 板二个 LED，灯的初始化函数，通过初始化 GPIO，完成 LED 灯的初始化操作。

```
#define LEDn                                    3U

#define LED1_PIN                                GPIO_PIN_0
#define LED1_GPIO_PORT                          GPIOB
#define LED1_GPIO_CLK_ENABLE()                  __HAL_RCC_GPIOB_CLK_ENABLE()
#define LED1_GPIO_CLK_DISABLE()                 __HAL_RCC_GPIOB_CLK_DISABLE()

#if defined (USE_NUCLEO_H745ZI_Q) || defined (USE_NUCLEO_H743ZI2) || defined (USE_NUCLEO_H7A3ZI_Q) || defined (USE_NUCLEO_H723ZG)
#define LED2_PIN                                GPIO_PIN_1
#define LED2_GPIO_PORT                          GPIOE
#define LED2_GPIO_CLK_ENABLE()                  __HAL_RCC_GPIOE_CLK_ENABLE()
#define LED2_GPIO_CLK_DISABLE()                 __HAL_RCC_GPIOE_CLK_DISABLE()
#else /* USE_NUCLEO_H743ZI */
#define LED2_PIN                                GPIO_PIN_7
#define LED2_GPIO_PORT                          GPIOB
#define LED2_GPIO_CLK_ENABLE()                  __HAL_RCC_GPIOB_CLK_ENABLE()
#define LED2_GPIO_CLK_DISABLE()                 __HAL_RCC_GPIOB_CLK_DISABLE()
#endif

#define LED3_PIN                                GPIO_PIN_14
#define LED3_GPIO_PORT                          GPIOB
#define LED3_GPIO_CLK_ENABLE()                  __HAL_RCC_GPIOB_CLK_ENABLE()
#define LED3_GPIO_CLK_DISABLE()                 __HAL_RCC_GPIOB_CLK_DISABLE()
```

函数的原型可声明为 void led_init (void);

函数的定义：

```
void led_init (void) {
    ……
```

…… //函数内完成（Nucleo 板三个 LED，Disco 板二个 LED，）灯的初始化。

HAL_GPIO_WritePin(xxxx,xxxxx, GPIO_PIN_SET); //初始化后可以先熄灭 LED（GPIO_PIN_RESET），也可以点亮 LED（GPIO_PIN_SET）。

}

✧ 注：函数应先声明原型，放在 main()函数外的 main.c 文件的上方，否则编译器不知道函数导致报错，当然也可以直接在 main()函数上方定义函数再调用。

创建好后在 CM7 的 main.c 文件中的主函数 main()内调用定义好的 LED 初始化函数，编译下载观察效果。

（2） 用硬延时 delay，实现 LED1 的循环亮灭（闪烁）；

void HAL_Delay(uint32_t Delay); 延迟函数，以毫秒为单位，通过 HAL_Delay 函数和 HAL_GPIO_WritePin 的配合使用实现 LED1 的闪烁。

（3） 接收按键输入信号，当按键按下，点亮 LED1；当按键断开，LED1 熄灭。

与(1)步骤相同，在 CM7 的 main.c 里定义按键初始化函数，可定义为 void key_init(void)，user 按键的端口和引脚号定义如下所示，将工作模式 Mode 设定为输入 GPIO_MODE_INPUT，上拉下拉 Pull 设定为 GPIO_PULLDOWN，随后在 main()函数里调用 key_init 函数完成按键的初始化，随后完成按键控制 LED1 的亮灭操作。

（4） 按一下按键 LED 灯点亮，再按一下 LED 灯熄灭。

（5） 按键消抖实验：用硬延时 delay，检测按键是否按下。

思路：当第一次检测到按键按下时，延时 10ms-20ms 再检测一次按键是否按下，如果按下则确认按键按下，若否，则属于干扰信号。

4. 观察上述步骤中每一步下载调试时的现象，撰写的实验报告中应包括完成每一步骤的相应代码，以及 LED 初始化和 Key 初始化函数的代码。

4.4 GPIO 提高实验（扩展）

GPIO 提高实验为扩展部分，任选一个完成：

1. 通过按键切换，实现 LED 灯的快闪，慢闪，双闪等功能。

初始时刻，Nucleo 板三个 LED，Disco 板二个 LED 灯初始化保持在点亮状态，首次按下按键时，LED1 开始快速闪烁，其余 LED 灯保持点亮。再次按下按

键时，LED2 开始慢速闪烁，其余 LED 灯保持点亮。再次按下按键，LED 灯同时开始闪烁。再次按下按键恢复至初始点亮状态。

2. 实现长按住按键 5s 后，LED1 灯开始闪烁；短按按键时 LED1 灯实现亮灭切换，即短按按键 LED1 熄灭，再次短按按键 LED1 点亮，每次短按按键时长最长不超过 1s。

4.5 调试程序完成实验报告

调试程序，并完成实验报告。

实验报告每人提交一份，需要包含调试实验的具体步骤并进行相应的分析。

若出现极为明显的抄袭现象，所涉及的同学本次实验成绩为 0，提交报告时间另行通知。

提交内容：

1. 实验工程源代码；
2. 实验报告；

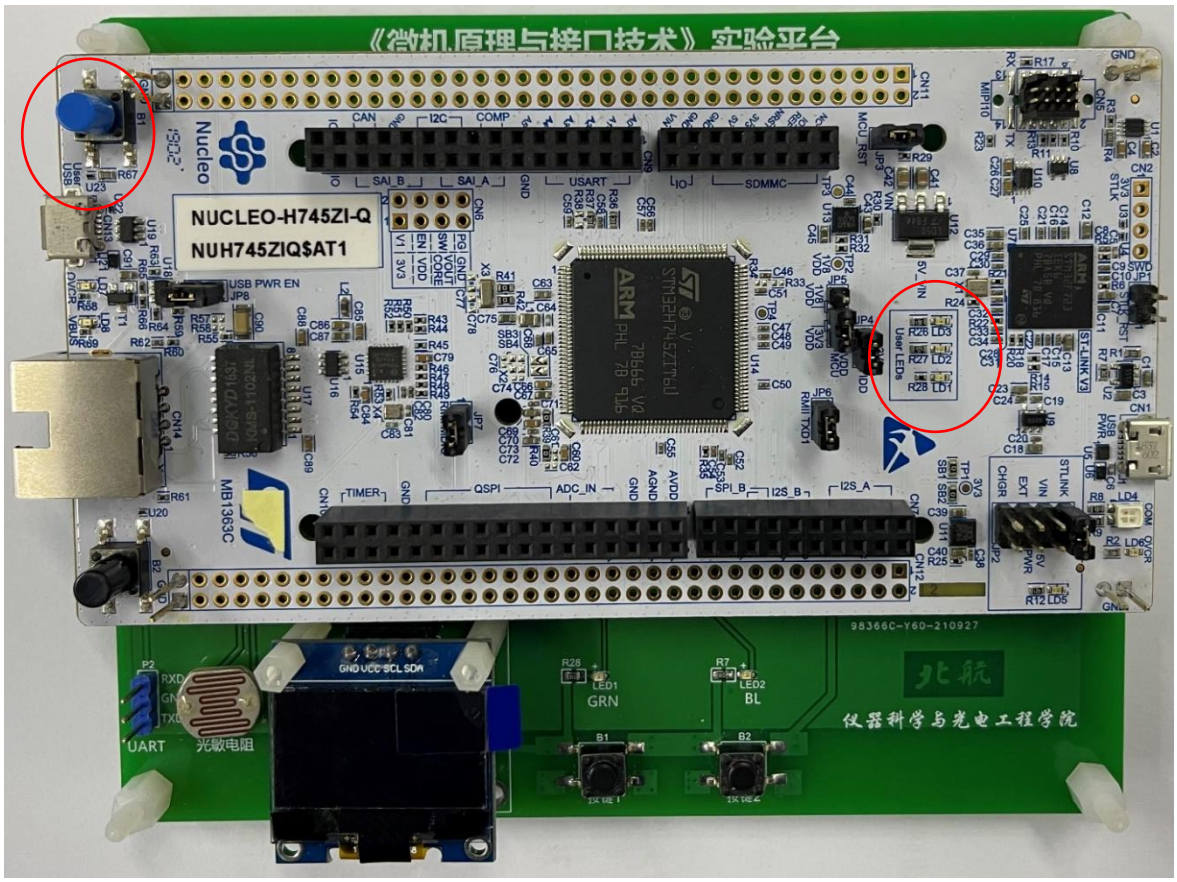
压缩打包，报告命名格式：“学号+姓名+实验 x+冯（于，吴）”

五、参考程序

见附件。

六、注意事项

（1）在 C 程序的调试过程中，如果遇到跳转指令执行时程序运行不正常的问题，请在跳转的目标指令地址处增加断点。



实验平台实物照片（Nucleo）

实验平台实物照片（Disco 版）