厦门大学

《嵌入式系统》

课程实验报告

姓名：苏一涵

学院：信息学院

系：软件工程

专业：软件工程

学号：36720232204041

2025 年 10 月

第 3 次实验　STM32、ARM 裸机设计实验

1. 实验设备

（1）PC 微机

（2）嵌入式系统综合实验箱（FS3399M4）

2. 实验内容

2.1 实验要求

1、请从设计实验 1-1、1-2、1-3、1-4 中任意选择一个（小键盘控制）

2、请从设计实验 1-5、1-6 中任意选择一个（红外遥控器控制）

3、请从设计实验 1-7、1-8 中任意选择一个（数码管显示）

4、设计实验 2-1 必做（混合编程）

5、请从设计实验 2-2、2-3 中任意选择一个（汇编语言）

6、请从设计实验 2-4、2-5、2-6 中任意选择一个（混合编程）

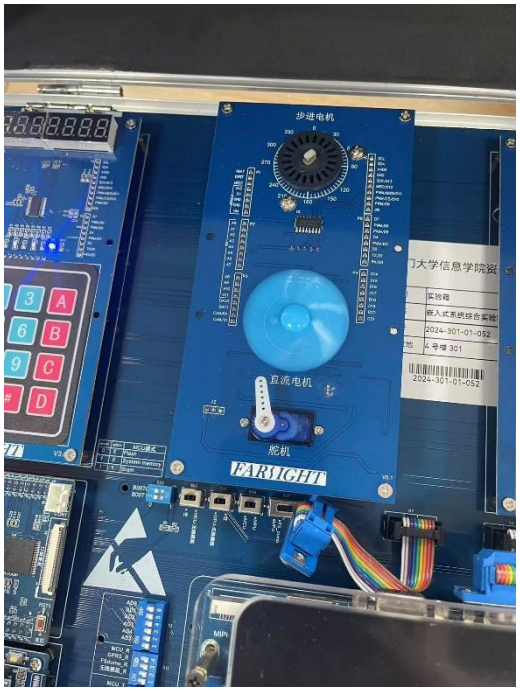
STM32 设计实验

1-1　小键盘控制步进电机

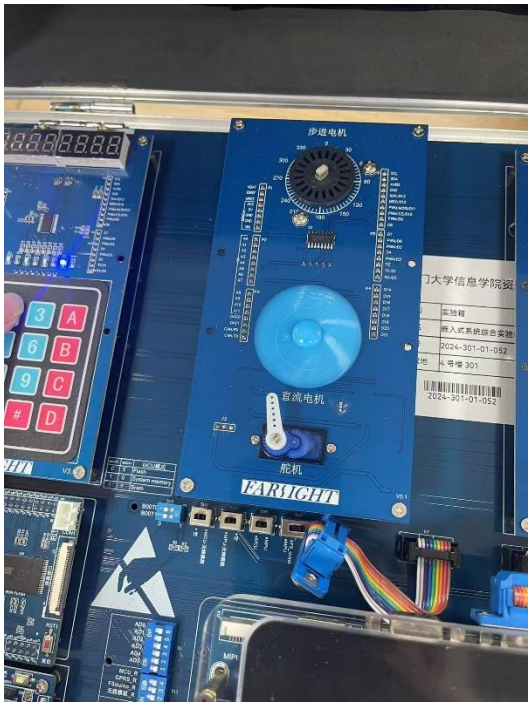
按 “1”键，步进电机顺时针转；按“2”键，步进电机逆时针转；按其它 14 个键，步进电机不转。

实验结果如下：

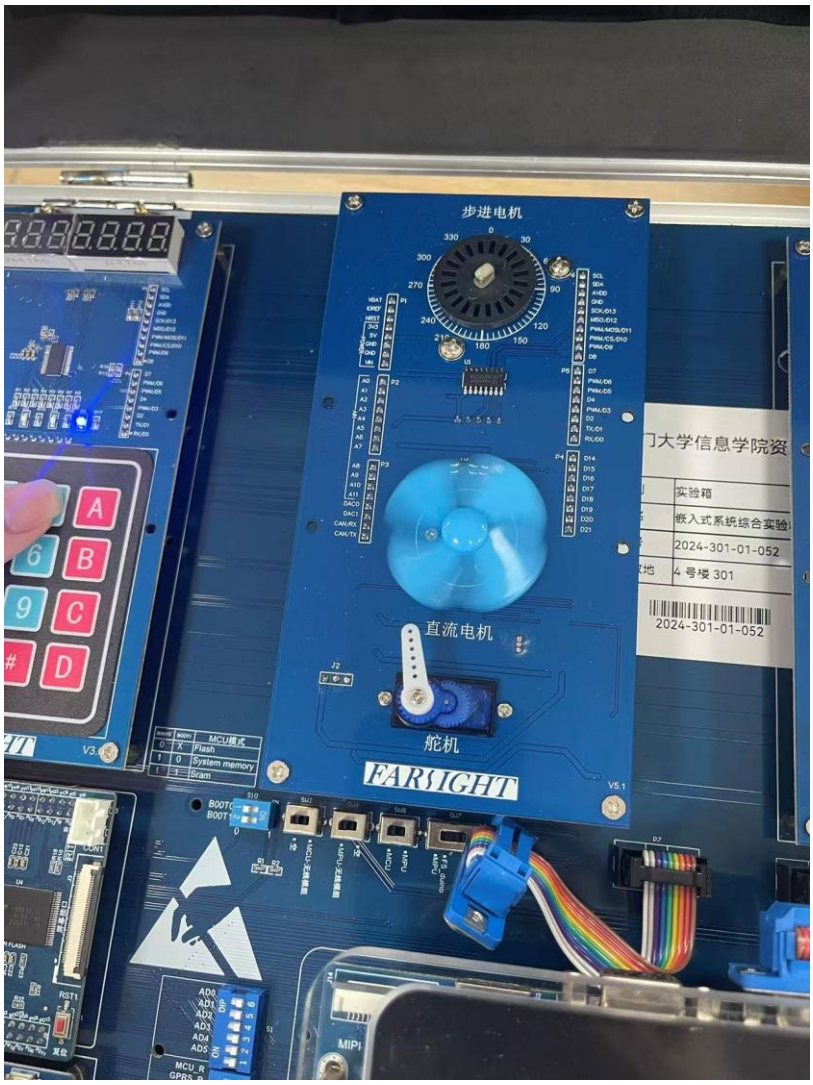
1 键：顺时针转



2 键：逆时针转



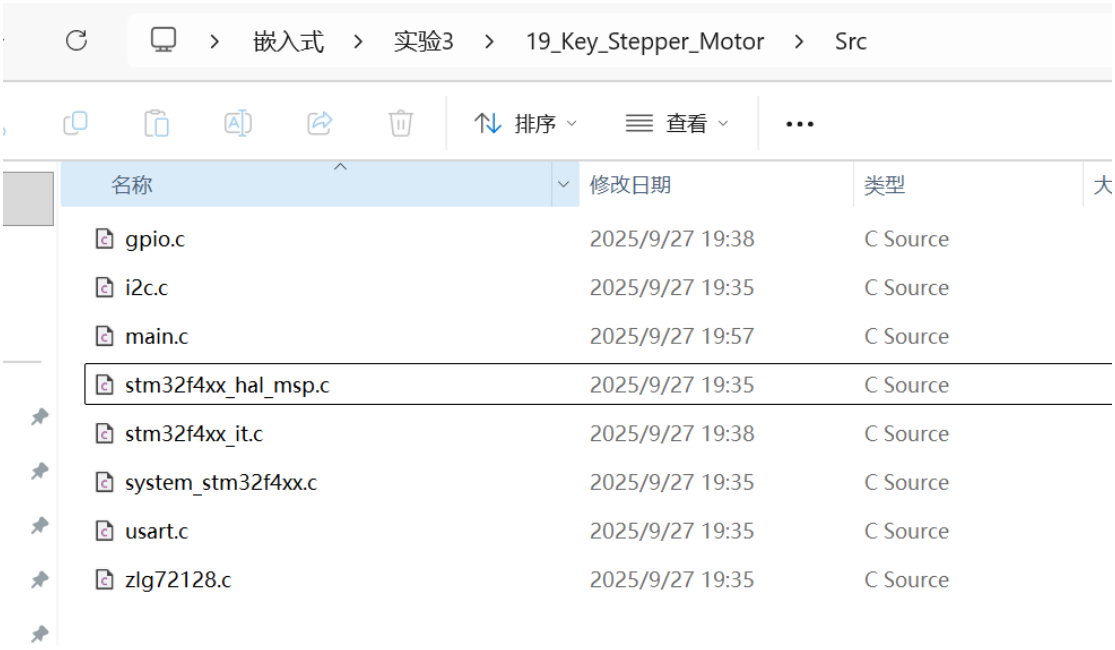
3 键：停止转



其余键：停止转

实验思路：

用“5_Stepper_Motor”实验工程中的 gpio.c、stm32f4xx_it.c、gpio.h、stm32f4xx_it.h，代替“19_Key_Stepper_Motor”实验工程中的 gpio.c、stm32f4xx_it.c、 gpio.h、 stm32f4xx_it.h



然后综合“3_DC_Motor”和“5_Stepper_Motor” 实验工程的 main.c，得到新的 main.c。

Main.c 文件详情如下：

首先定义一些变量和声明一些函数

```
#define ZLG_READ_ADDRESS1              0x01    //键值寄存器
#define ZLG_READ_FUNCTION_ADDRESS   0x03    //功能键寄存器
#define ZLG_READ_ADDRESS2              0x10
#define ZLG_WRITE_ADDRESS1             0x17    //数码管显示末尾地址
#define ZLG_WRITE_ADDRESS2             0x16
#define ZLG_WRITE_FLASH     0x0B
#define ZLG_WRITE_SCANNUM 0x0D
#define BUFFER_SIZE1                   (countof(Tx1_Buffer))
#define BUFFER_SIZE2                   (countof(Rx2_Buffer))
#define countof(a)              (sizeof(a) / sizeof(*(a)))

#define DE_A        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12,GPIO_PIN_SET);HAL_GPIO_Writ
#define DE_B        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12,GPIO_PIN_RESET);HAL_GPIO_Wr
#define DE_C            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12,GPIO_PIN_RESET);HAL_GPI
#define DE_D            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12,GPIO_PIN_RESET);HAL_GPI
#define DE_AB       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12,GPIO_PIN_SET);HAL_GPIO_Writ
#define DE_BC       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12,GPIO_PIN_RESET);HAL_GPIO_Wr
#define DE_CD           HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12,GPIO_PIN_RESET);HAL_GPI
#define DE_DA           HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12,GPIO_PIN_SET);HAL_GPIO_

uint8_t flag;               //不同的按键有不同的标志位值
uint8_t flag_key = 0;//中断标志位，每次按键产生一次中断，并开始读取8个数码管的值
uint8_t Rx2_Buffer[8]={0};
uint8_t Tx1_Buffer[8]={0};
uint8_t clear[9] = { 0 };
uint8_t Rx1_Buffer[1]={0};
uint8_t Rx1_Buffer_P[1]={0};
uint8_t Rx1_Buffer_T[1]={0};
uint8_t reset[1]={0xff};
uint8_t Transmit_Buffer[2]={0x00,0x03};

void delay_my(uint8_t time);
void SystemClock_Config(void);
void swtich_key(void);
void swtich_key_func(void);
void switch_flag(void);
```

初始化各项进程

```
HAL_Init();      //HAL初始化
SystemClock_Config();    //系统时钟配置
MX_GPIO_Init();      //GPIO初始化
MX_I2C1_Init();      //I2C初始化
MX_USART1_UART_Init();  //串口初始化
```

然后进入主循环

```
while (1)
{
    I2C_ZLG72128_Read(&hi2c1, 0x61, 0x01, Rx1_Buffer_P, 1);//读普通按键值
    I2C_ZLG72128_Read(&hi2c1, 0x61, 0x03, Rx1_Bufer_T, 1);//读功能按键值
    if (Rx1_Buffer_P[0] != 0x0)//普通按键（12个）
        swtich_key();//键值转换
    if (Rx1_Buffer_T[0] != 0xff)        //功能按键（4个）
        swtich_key_func();//键值转换
    if(flag == 1)   //如果有按键
    {
        //*****八拍方式******   顺时针转       A  AB  B  BC  C  CD  D  DA
        DE_A;
        HAL_Delay(1);       //可进行调速，延时时间不能太短     3（最慢）、2（中等）、1（最快）

        DE_AB;
        HAL_Delay(1);

        DE_B;
        HAL_Delay(1);

        DE_BC;
        HAL_Delay(1);

        DE_C;
        HAL_Delay(1);

        DE_CD;
        HAL_Delay(1);
```

首先读入键盘的值

如果是数字案件转入对应的转换函数，如下

```
void swtich_key(void)       //普通按键值转换（12个键）————0、4、5、6、7、8、9、11、12、13、14、15
{
    switch(Rx1_Buffer_P[0])
    {
        case 0x01:
            flag = 13;
            break;
        case 0x02:
            flag = 15;
            break;
        case 0x03:
            flag = 0;
            break;
        case 0x04:
            flag = 14;
            break;
        case 0x09:
            flag = 12;
            break;
        case 0x0a:
            flag = 9;
            break;
        case 0x0b:
            flag = 8;
            break;
        case 0x0c:
            flag = 7;
            break;
        case 0x11:
            flag = 11;
            break;
        case 0x12:
            flag = 6;
            break;
        case 0x13:
            flag = 5;
            break;
        case 0x14:
            flag = 4;
            break;
        default:
            break;
    }
}
```

如果是功能按键则进入另一个函数：

```
void swtich_key_func(void)      //功能按键值转换（4个）————1、2、3、10
{
    switch (Rx1_Buffer_T[0])
    {
        case 0xfe:
            flag = 10;
            break;
        case 0xfd:
            flag = 3;
            break;
        case 0xfb:
            flag = 2;
            break;
        case 0xf7:
            flag = 1;
            break;
        default:
            break;
    }
}
```

如果按的是 1 键，则顺时针旋转

```
if(flag == 1)   //如果有按键
{
    //*****八拍方式******   顺时针转      A  AB  B  BC  C  CD  D  DA
    DE_A;
    HAL_Delay(1);       //可进行调速，延时时间不能太短     3（最慢）、2（中等）、1（最快）

    DE_AB;
    HAL_Delay(1);

    DE_B;
    HAL_Delay(1);

    DE_BC;
    HAL_Delay(1);

    DE_C;
    HAL_Delay(1);

    DE_CD;
    HAL_Delay(1);

    DE_D;
    HAL_Delay(1);

    DE_DA;
    HAL_Delay(1);
}
```

按的是 2 则逆时针旋转

```
if (flag == 2) {
    //*****八拍方式******   逆时针转      DA  D  CD  C  BC  B  AB  A
    DE_DA;
    HAL_Delay(3);       //可进行调速，延时时间不能太短     3（最慢）、2（中等）、1（最快）

    DE_D;
    HAL_Delay(3);

    DE_CD;
    HAL_Delay(3);

    DE_C;
    HAL_Delay(3);

    DE_BC;
    HAL_Delay(3);

    DE_B;
    HAL_Delay(3);

    DE_AB;
    HAL_Delay(3);

    DE_A;
    HAL_Delay(3);
}

HAL_Delay(100); //延时100ms
```

1-2　小键盘控制陀机

1-3　小键盘控制 LED 灯

1-4　小键盘控制蜂鸣器

1-5　红外遥控器控制 LED 灯

1-6　红外遥控器控制蜂鸣器

通过红外遥控器控制蜂鸣器的响/不响；按遥控器上的“1”键，蜂鸣器响；按遥控器上的其它键，蜂鸣器不响

实验思路：

1、在红外接收 “13_IR_Receive”实验工程的基础上修改程序，将新的实验工程命名为“24_IR_Receive_Beep”

2、将“13_IR_Receive”和“2_Beep”实验工程的 gpio.c，结合起来，成为新的 gpio.c

3、综合“13_IR_Receive”和“2_Beep”实验工程的 main.c，得到新的 main.c

结合得到的 gpio.c 情况如下

```
#include "gpio.h"

void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOF_CLK_ENABLE();
  GPIO_InitStruct.Pin = GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOF, &GPIO_InitStruct);
  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 2, 2);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOG_CLK_ENABLE();

  HAL_GPIO_WritePin(GPIOG, GPIO_PIN_6, GPIO_PIN_RESET);

  GPIO_InitStruct.Pin = GPIO_PIN_6;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);

}

```

将 GPIOF.15 配置为中断输入引脚（上升 / 下降沿触发中断），用于检测外部信号的电平变化（如按键、传感器信号）。

将 GPIOG.6 配置为推挽输出引脚（初始低电平），用于输出控制信号（如驱动 LED、继电器等外设）。

Main.c 代码设计如下：

```c
#include "main.h"
#include "usart.h"
#include "gpio.h"
#include "stdio.h"
#include "RemoteInfrared.h"

__IO uint32_t GlobalTimingDelay100us;
void SystemClock_Config(void);
uint8_t flag;

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_USART1_UART_Init();

    printf("\n\r FS-STM32开发板 IR红外线接收实验程序\n\r");

    while (1)
    {
        flag = Remote_Infrared_KeyDeCode(); //等待按红外遥控器的按键

        if (flag == 8)
        {
            HAL_GPIO_WritePin(GPIOG, GPIO_PIN_6, GPIO_PIN_SET);
            HAL_Delay(5);       //延时2ms
            HAL_GPIO_WritePin(GPIOG, GPIO_PIN_6, GPIO_PIN_RESET);
            HAL_Delay(5);       //延时2ms
        }
    }
}
```

初始化函数

```c
HAL_Init();
SystemClock_Config();
MX_GPIO_Init();
MX_USART1_UART_Init();
```

红外按键解码：读取遥控器按键，返回解码结果 flag

```c
flag = Remote_Infrared_KeyDeCode(); //等待按红外遥控器的按键
```

若解码结果为 8（对应遥控器某特定按键，如“OK 键”“数字键 8”）

控制蜂鸣器响动

```
if (flag == 8)
{
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_6, GPIO_PIN_SET);
    HAL_Delay(5);       //延时2ms
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_6, GPIO_PIN_RESET);
    HAL_Delay(5);       //延时2ms
}
```

1-7　电子钟

编写程序，实现在数码管上显示时、分、秒，每 1 秒变化 1 次

设计思路：修改“11_ZLG72128”实验工程的 main.c，得到新的 main.c

Main.c 如下：

```
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_I2C1_Init();
    MX_USART1_UART_Init();

    printf("=======>电子钟测试程序<=======\n");

    while (1)
    {
        second++;
        if (second == 60)
        {
            second = 0;
            minute++;
            if (minute == 60)
            {
                minute = 0;
                hour++;
                if (hour == 24)
                    hour = 0;
            }
        }
        hour_high = hour / 10;
        hour_low = hour - hour_high * 10;
        minute_high = minute / 10;
        minute_low = minute - minute_high * 10;
        second_high = second / 10;
        second_low = second - second_high * 10;
        // 显示“时”高位
        Tx1_Buffer[0] = convert(hour_high);
        I2C_ZLG72128_Write_char(&hi2c1, 0x60, ZLG_WRITE_ADDRESS8, Tx1_Buffer);
        // 显示“时”低位
        Tx1_Buffer[0] = convert(hour_low);
        I2C_ZLG72128_Write_char(&hi2c1, 0x60, ZLG_WRITE_ADDRESS7, Tx1_Buffer);

        // 显示“-”（时与分之间）
        Tx1_Buffer[0] = 0x40;
        I2C_ZLG72128_Write_char(&hi2c1, 0x60, ZLG_WRITE_ADDRESS6, Tx1_Buffer);

        // 显示“分”高位
```

先初始化关键的功能

```
HAL_Init();
SystemClock_Config();
MX_GPIO_Init();
MX_I2C1_Init();
MX_USART1_UART_Init();
```

然后用 **second** 这个变量来统计秒数，没六十秒加一分钟，六十分钟加一小时

```
second++;
if (second == 60)
{
    second = 0;
    minute++;
    if (minute == 60)
    {
        minute = 0;
        hour++;
        if (hour == 24)
            hour = 0;
    }
}
hour_high = hour / 10;
hour_low = hour - hour_high * 10;
minute_high = minute / 10;
minute_low = minute - minute_high * 10;
second_high = second / 10;
second_low = second - second_high * 10;
```

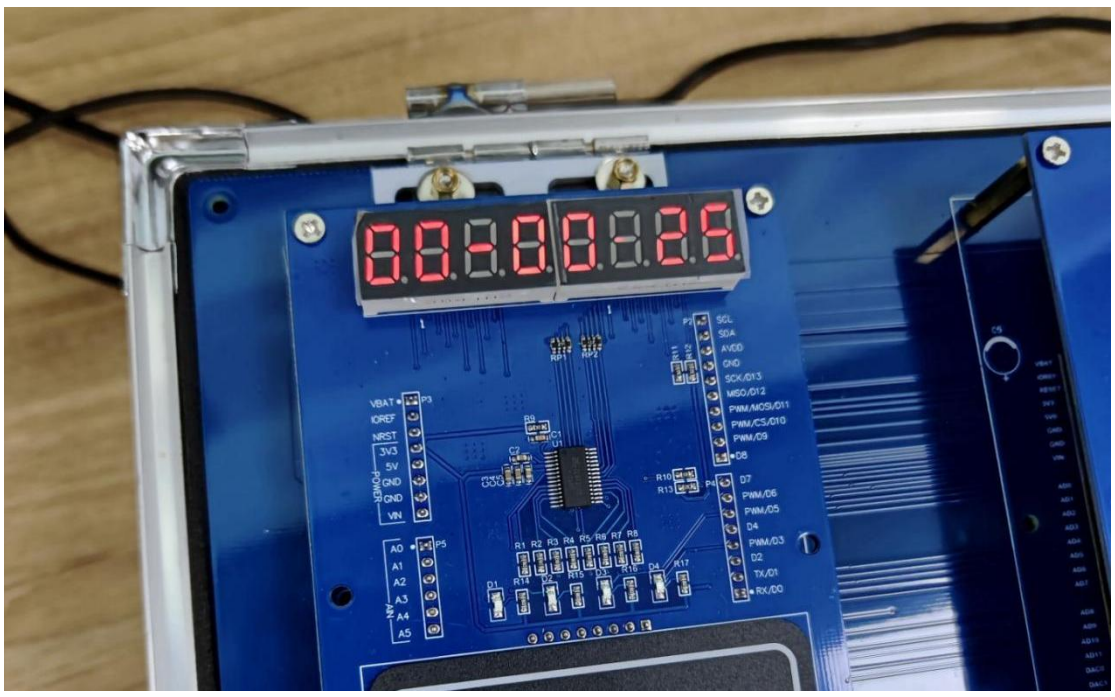
再根据以下内容修改“**11_ZLG72128**”原有程序

3、要在8个数码管的某一位（某一个）上显示某个字符的程序如下：

```
#define ZLG_WRITE_ADDRESS1                0x17          //最右边的数码管（显示缓冲区首地址）
#define ZLG_WRITE_ADDRESS2                0x16          //左数第7个数码管
#define ZLG_WRITE_ADDRESS3                0x15          //左数第6个数码管
#define ZLG_WRITE_ADDRESS4                0x14          //左数第5个数码管
#define ZLG_WRITE_ADDRESS5                0x13          //左数第4个数码管
#define ZLG_WRITE_ADDRESS6                0x12          //左数第3个数码管
#define ZLG_WRITE_ADDRESS7                0x11          //左数第2个数码管
#define ZLG_WRITE_ADDRESS8                0x10          //最左边的数码管

Tx1_Buffer[0] = 0x06;
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS7,Tx1_Buffer);     //在左数第2个数码管上显示“1”
```

```
    // 显示“时”高位
    Tx1_Buffer[0] = convert(hour_high);
    I2C_ZLG72128_Write_char(&hi2c1, 0x60, ZLG_WRITE_ADDRESS8, Tx1_Buffer);
    // 显示“时”低位
    Tx1_Buffer[0] = convert(hour_low);
    I2C_ZLG72128_Write_char(&hi2c1, 0x60, ZLG_WRITE_ADDRESS7, Tx1_Buffer);

    // 显示“-”（时与分之间）
    Tx1_Buffer[0] = 0x40;
    I2C_ZLG72128_Write_char(&hi2c1, 0x60, ZLG_WRITE_ADDRESS6, Tx1_Buffer);

    // 显示“分”高位
    Tx1_Buffer[0] = convert(minute_high);
    I2C_ZLG72128_Write_char(&hi2c1, 0x60, ZLG_WRITE_ADDRESS5, Tx1_Buffer);
    // 显示“分”低位
    Tx1_Buffer[0] = convert(minute_low);
    I2C_ZLG72128_Write_char(&hi2c1, 0x60, ZLG_WRITE_ADDRESS4, Tx1_Buffer);

    // 显示“-”（分与秒之间）
    Tx1_Buffer[0] = 0x40;
    I2C_ZLG72128_Write_char(&hi2c1, 0x60, ZLG_WRITE_ADDRESS3, Tx1_Buffer);

    // 显示“秒”高位
    Tx1_Buffer[0] = convert(second_high);
    I2C_ZLG72128_Write_char(&hi2c1, 0x60, ZLG_WRITE_ADDRESS2, Tx1_Buffer);
    // 显示“秒”低位
    Tx1_Buffer[0] = convert(second_low);
    I2C_ZLG72128_Write_char(&hi2c1, 0x60, ZLG_WRITE_ADDRESS1, Tx1_Buffer);

    HAL_Delay(1000); // 延时1000ms
```

实验结果如下：



1-8　在数码管上显示温度采集值

ARM 裸机设计实验

2-1　LED 灯（混合编程）

采用汇编语言与 C 语言混合编程的方式编写 LED 灯实验程序。

Main.c 代码

```c
#include "fs_led.h"
#include "fs3399_timer.h"

/*--------------------------MAIN FUNCTION--------------------------*/
/**
 * @brief  Main program body
 * @param[in]  None
 * @return int
 */
int main()
{
    //设置LED1(GPIO4_C6)、LED2(GPIO0_A2)、LED3(GPIO0_B4) 为输出模式
    FsLedInit();

    while (1)
    {
        //打开LED1
        FsLedOn(1);
        //延时100ms
        fs_delay_ms(100);
        //关闭LED1
        FsLedOff(1);
        //延时100ms
        fs_delay_ms(100);

        //打开LED2
        FsLedOn(2);
        //延时100ms
        fs_delay_ms(100);
        //关闭LED2
        FsLedOff(2);
        //延时100ms
        fs_delay_ms(100);

        //打开LED3
        FsLedOn(3);
        //延时100ms
        fs_delay_ms(100);
        //关闭LED3
        FsLedOff(3);
        //延时100ms
        fs_delay_ms(100);
    }
    return 0;
}
```

Start.s 代码如下：

```
#include "macro.h"


.globl  _start
_start:
    b   reset

    .align 3

reset:
#if 1
    /*
     * Could be EL3/EL2/EL1, Initial State:
     * Little Endian, MMU Disabled, i/dCache Disabled
     */
    //adr   x0, vectors
    switch_el x1, 3f, 2f, 1f
3:  msr vbar_el3, x0
    mrs x0, scr_el3
    orr x0, x0, #0xf            /* SCR_EL3.NS|IRQ|FIQ|EA */
    msr scr_el3, x0
    msr cptr_el3, xzr           /* Enable FP/SIMD */
    ldr x0, =240000000
    msr cntfrq_el0, x0          /* Initialize CNTFRQ */
    b   0f
2:  msr vbar_el2, x0
    mov x0, #0x33ff
    msr cptr_el2, x0            /* Enable FP/SIMD */
    b   0f
1:  msr vbar_el1, x0
    mov x0, #3 << 20
    msr cpacr_el1, x0           /* Enable FP/SIMD */
0:


    /*
     * Cache/BPB/TLB Invalidate
     * i-cache is invalidated before enabled in icache_enable()
     * tlb is invalidated before mmu is enabled in dcache_enable()
     * d-cache is invalidated before enabled in dcache_enable()
     */

    bl  lowlevel_init

    branch_if_master x0, x1, master_cpu

    /*
     * Slave CPUs
     */
slave_cpu:
    wfe
    ldr x1, =0x41c00000
    ldr x0, [x1]
    cbz x0, slave_cpu
    br  x0          /* branch to the given address */
master_cpu:
    /* On the master CPU */
#endif
```

```
58      #endif
59      _main:
60          mov x0, #1;
61          mov x1, #0;
62          bl main
63          b _main
64
65
66      ENTRY(lowlevel_init)
67          mov x29, x30            /* Save LR */
68
69          branch_if_master x0, x1, 2f
70
71          /*
72           * Slave should wait for master clearing spin table.
73           * This sync prevent salves observing incorrect
74           * value of spin table and jumping to wrong place.
75           */
76
77          /*
78           * All slaves will enter EL2 and optionally EL1.
79           */
80          bl  armv8_switch_to_el2
81
82      2:
83          mov x30, x29            /* Restore LR */
84          ret
85      ENDPROC(lowlevel_init)
86
87      ENTRY(armv8_switch_to_el2)
88      switch_el x0, 1f, 0f, 0f
89      0:  ret
90      1:  armv8_switch_to_el2_m x0
91      ENDPROC(armv8_switch_to_el2)
92
93
```

fs3399_led.c 代码如下：

```c
#include "fs_led.h"

int FsLedInit()
{
    GPIO4->SWPORTA_DDR |= (0x1 << 22);      //LED1输出模式——GPIO4的DDR的C6置1
    GPIO0->SWPORTA_DDR |= (0x1 << 2);       //LED2输出模式——GPIO0的DDR的A2置1
    GPIO0->SWPORTA_DDR |= (0x1 << 12);      //LED3输出模式——GPIO0的DDR的B4置1

    return 0;
}

int FsLedOn(int n)
{
    switch (n)
    {
    case 1:
        GPIO4->SWPORTA_DR |= (0x1 << 22);  //LED1灯亮——GPIO4的DR的C6置1
        return 0;
    case 2:
        GPIO0->SWPORTA_DR |= (0x1 << 2);   //LED2灯亮——GPIO0的DR的A2置1
        return 0;
    case 3:
        GPIO0->SWPORTA_DR |= (0x1 << 12);  //LED3灯亮——GPIO0的DR的B4置1
        return 0;
    default:
        return 0;
    }
}

int FsLedOff(int n)
{
    switch (n)
    {
    case 1:
        GPIO4->SWPORTA_DR &= ~(0x1 << 22); //LED1灯灭——GPIO4的DR的C6置0
        return 0;
    case 2:
        GPIO0->SWPORTA_DR &= ~(0x1 << 2);  //LED2灯灭——GPIO0的DR的A2置0
        return 0;
    case 3:
        GPIO0->SWPORTA_DR &= ~(0x1 << 12); //LED3灯灭——GPIO0的DR的B4置0
        return 0;
    default:
        return 0;
    }
}
```

fs3399_led.h 代码如下：

```c
#ifndef __FS_LED_H__
#define __FS_LED_H__

#include "fs3399_gpio.h"

#define PMUCRU_BASE                0xFF750000

//pclk_gpio1_en
#define PMUCRU_CLKGATE_CON1        (*((volatile unsigned int *)(PMUCRU_BASE+0x0104)))

int FsLedInit(void);

int FsLedOn(int);

int FsLedOff(int);

#endif /* __FS_LED_H__ */
```

Makefile 文件如下：

将 name 改成 fs_led

```
# CORTEX-A53 PERI DRIVER CODE
# VERSION 3.0
# ATHUOR www.hqyj.com
# MODIFY DATE
# 2020.04.12  Makefile
# SHELL=C:/Windows/System32/cmd.exe

CROSS_COMPILE   := ~/toolchain/6.4-aarch64/bin/aarch64-linux-

CFLAGS += -g -O0 -fno-builtin -nostdinc -I./common/include -I./include

CC =        $(CROSS_COMPILE)gcc
LD =        $(CROSS_COMPILE)ld
OBJCOPY =   $(CROSS_COMPILE)objcopy
OBJDUMP =   $(CROSS_COMPILE)objdump
OUTPUT_DIR = ./output

NAME    = $(OUTPUT_DIR)/fs_led

OBJSss := $(wildcard ./start/*.S) $(wildcard ./common/src/*.S) \
       $(wildcard ./source/*.S) $(wildcard ./*.S) \
       $(wildcard ./start/*.c) $(wildcard ./common/src/*.c) \
       $(wildcard ./source/*.c) $(wildcard ./*.c)
OBJSs  := $(patsubst %.S,%.o,$(OBJSss))
OBJS   := $(patsubst %.c,%.o,$(OBJSs))

all:$(OBJS)
    @ echo "  LD   Linking  $(NAME).elf"
    @ $(LD) -Tmap.lds -o $(NAME).elf $^
    @ echo "  OBJCOPY Objcopying  $(NAME).bin"
    @ $(OBJCOPY) -O binary -S $(NAME).elf $(NAME).bin
    @ echo "  OBJDUMP Objdumping  $(NAME).dis"
    @ $(OBJDUMP) -D $(NAME).elf > $(NAME).dis

%.o : %.S
    @ echo "  AS   $@"
    @ $(CC) -o $@ $< -c $(CFLAGS)

%.o : %.c
    @ echo "  CC   $@"
    @ $(CC) -o $@ $< -c $(CFLAGS)
distclean clean:
    @ echo "  CLEAN  complete."
    @ rm  $(OBJS) $(NAME).* main.o -rf

install:
    cp $(NAME).bin /mnt/hgfs/share
```

实验结果如下：

2-2　呼吸灯（汇编语言）

用汇编语言的方式编写呼吸灯实验程序：　02-fs_assembly_pwm

fs_assembly_pwm.s 代码如下

```
.text
.global _start
_start:
//-------------------------初始化PWM-------------------------
    //设置GPIO4_C6（LED1灯）为第二功能：PWM1
    ldr x0, =0xFF77E028          //GRF_GPIO4C_IOMUX寄存器的地址=0xFF77E028
    ldr w1, =(0x3 << 28) | (0x1 << 12)
    str w1, [x0]

    //先关掉PWM1输出
    ldr x0, =0xFF42001C          //PWM1->CTRL寄存器的地址=0xFF42001C
    ldr w1, [x0]
    and w1, w1, #~(0x1 << 0)
    str w1, [x0]

    //设置时钟源
    ldr x0, =0xFF42001C
    ldr w1, [x0]
    orr w1, w1, #(0x1 << 9)
    str w1, [x0]

    //设置分频因子
    ldr x0, =0xFF42001C
    ldr w1, [x0]
    orr w1, w1, #(0x1 << 12)
    str w1, [x0]

    //设置预分频
    ldr x0, =0xFF42001C
    ldr w1, [x0]
    orr w1, w1, #(0x1 << 16)
    str w1, [x0]

    //设置PWM的模式
    ldr x0, =0xFF42001C
    ldr w1, [x0]
    orr w1, w1, #(0x1 << 1)
    str w1, [x0]

    //设置PWM输出波形起始极性
    ldr x0, =0xFF42001C
    ldr w1, [x0]
    orr w1, w1, #(0x1 << 3)
    str w1, [x0]

    //设置PWM对齐方式
    ldr x0, =0xFF42001C
    ldr w1, [x0]
    and w1, w1, #~(0x1 << 5)
    str w1, [x0]
```

```
    //使能PWM
    ldr x0, =0xFF42001C
    ldr w1, [x0]
    orr w1, w1, #(0x1 << 0)
    str w1, [x0]
  //-------------------------------------------------------------

LOOP0:
    ldr w2, =0
LOOP1:
    //置PWM1->PERIOD_HPR = 1000
    ldr x0, =0xFF420014           //PWM1->PERIOD_HPR寄存器的地址=0xFF420014
    ldr w1, =1000
    str w1, [x0]

    //置PWM1->DUTY_LPR = w2
    ldr x0, =0xFF420018           //PWM1->DUTY_LPR寄存器的地址=0xFF420018
    str w2, [x0]

    //延时
    ldr w3, =0x00FFFFFF           //设置一个计数值
LOOP2:
    sub w3, w3, #1
    cmp w3, #0
    bne LOOP2

    add w2, w2, #1
    cmp w2, #1000
    bne LOOP1

LOOP3:
    ldr w2, =1000
    //置PWM1->PERIOD_HPR = 1000
    ldr x0, =0xFF420014           //PWM1->PERIOD_HPR寄存器的地址=0xFF420014
    ldr w1, =1000
    str w1, [x0]

    //置PWM1->DUTY_LPR = w2
    ldr x0, =0xFF420018           //PWM1->DUTY_LPR寄存器的地址=0xFF420018
    str w2, [x0]

    //延时
    ldr w3, =0x00FFFFFF           //设置一个计数值
```

```
  LOOP4:
      sub w3, w3, #1
      cmp w3, #0
      bne LOOP4

      sub w2, w2, #1
      cmp w2, #0
      bne LOOP3

      b LOOP0
    //-------------------------------------------------------------
  stop:
      b stop
```

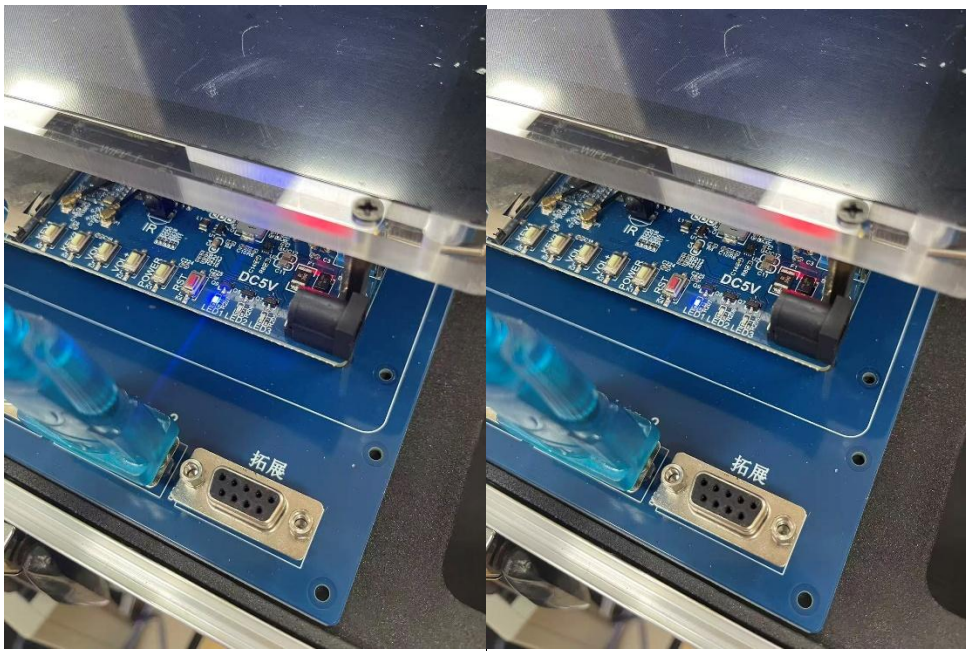
Makefile 代码如下：

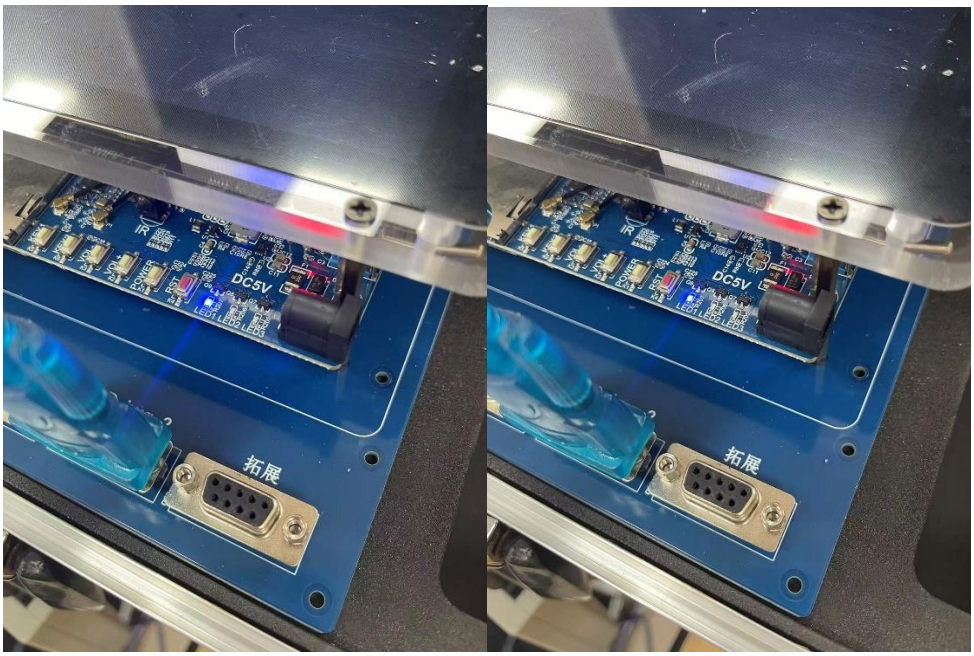
```
#
# System environment variable.
#

NAME    = fs_assembly_pwm
CROSS   = ~/toolchain/6.4-aarch64/bin/aarch64-linux-
CC      = $(CROSS)gcc
LD      = $(CROSS)ld
OBJCOPY = $(CROSS)objcopy
OBGJDOMP = $(CROSS)objdump
CFLAGS  = -O0 -g -c

all:
    $(CC) $(CFLAGS) -o $(NAME).o $(NAME).s
    $(LD) $(NAME).o -Tmap.lds -o $(NAME).elf
    $(OBJCOPY) -O binary -S $(NAME).elf $(NAME).bin
    $(OBGJDOMP) -D $(NAME).elf > $(NAME).dis

clean:
    rm -rf *.o *.bin *.elf *.dis

.PHONY: all clean
```

map.lds 代码如下：

```
OUTPUT_FORMAT("elf64-littleaarch64", "elf64-littleaarch64", "elf64-littleaarch64")

OUTPUT_ARCH(aarch64)
ENTRY(_start)
SECTIONS
{
    . = 0x00280000;
    . = ALIGN(8);
    .text :
    {
        fs_assembly_pwm.o(.text)
        *(.text)
    }
    . = ALIGN(8);
    .rodata :
    { *(.rodata) }
    . = ALIGN(8);
    .data :
    { *(.data) }
    . = ALIGN(8);
    .bss :
     { *(.bss) }
}
```

实验结果如下：





2-3　蜂鸣器（汇编语言

2-4　查询方式按键控制蜂鸣器（混合编程）

2-5　中断方式按键控制蜂鸣器（混合编程）

2-6　串口发送与接收（混合编程）

采用汇编语言与 C 语言混合编程的方式编写发送与接收的程序： 13-fs_uart_receive

Main.c 文件代码如下：

```c
#include "fs3399_uart.h"
#include "common.h"

int main()
{
    char str1[] = "FS3399 UART test string !";
    char str2[] = "Xiamen University niub666";

    char c;

    fs_uart_init(115200);              //串口初始化，115200 is baud rate

    //测试串口发送数据

    //发送字符
    fs_putc('A');
    fs_putc('B');
    fs_putc('C');
    fs_putc('1');
    fs_putc('2');
    fs_putc('3');
    fs_putc('a');
    fs_putc('b');
    fs_putc('c');

    fs_putc('\n');
    fs_putc('\r');

    //发送字符串
    fs_puts(str1);

    //发送字符
    fs_putc('\n');
    fs_putc('\r');

    //发送字符串
    fs_puts(str2);

    //printf函数测试
    printf("\n\r");
    printf("fs3399 test printf function\n\r");

    while (1)
    {
        c = fs_getc();
        fs_putc(c);
    }

    return 0;
}

```

增加了收发字符的代码

```c
while (1)
{
    c = fs_getc();
    fs_putc(c);
}
```

Makefile 文件如下：

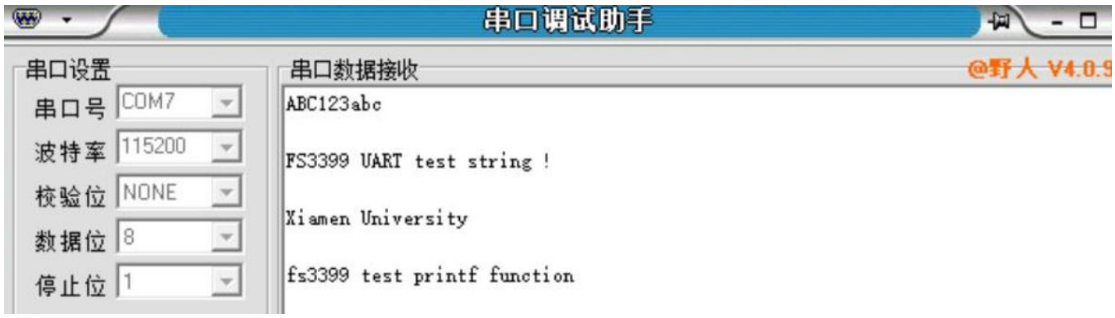
```
# CORTEX-A53 PERI DRIVER CODE
# VERSION 3.0
# ATHUOR www.hqyj.com
# MODIFY DATE
# 2020.04.12  Makefile
# SHELL=C:/Windows/System32/cmd.exe

CROSS_COMPILE   := ~/toolchain/6.4-aarch64/bin/aarch64-linux-

CFLAGS += -g -O0 -fno-builtin -nostdinc -I./common/include -I./include

CC =        $(CROSS_COMPILE)gcc
LD =        $(CROSS_COMPILE)ld
OBJCOPY =   $(CROSS_COMPILE)objcopy
OBJDUMP =   $(CROSS_COMPILE)objdump
OUTPUT_DIR = ./output

NAME    = $(OUTPUT_DIR)/fs_uart_receive

OBJSs := $(wildcard ./start/*.S) $(wildcard ./common/src/*.S) \
        $(wildcard ./source/*.S) $(wildcard ./*.S) \
        $(wildcard ./start/*.c) $(wildcard ./common/src/*.c) \
        $(wildcard ./source/*.c) $(wildcard ./*.c)
OBJSs  := $(patsubst %.S,%.o,$(OBJSs))
OBJS   := $(patsubst %.c,%.o,$(OBJSs))

all:$(OBJS)
    @ echo "  LD   Linking  $(NAME).elf"
    @ $(LD) -Tmap.lds -o $(NAME).elf $^
    @ echo "  OBJCOPY Objcopying  $(NAME).bin"
    @ $(OBJCOPY) -O binary -S $(NAME).elf $(NAME).bin
    @ echo "  OBJDUMP Objdumping  $(NAME).dis"
    @ $(OBJDUMP) -D $(NAME).elf > $(NAME).dis

%.o : %.S
    @ echo "  AS   $@"
    @ $(CC) -o $@ $< -c $(CFLAGS)

%.o : %.c
    @ echo "  CC   $@"
    @ $(CC) -o $@ $< -c $(CFLAGS)
distclean clean:
    @ echo "  CLEAN  complete."
    @ rm  $(OBJS) $(NAME).* main.o -rf

install:
    cp $(NAME).bin /mnt/hgfs/share
```

Start.s 代码如下：

```
 4      .globl  _start
 5      _start:
 6          b   reset
 7
 8          .align 3
 9
10      reset:
11      #if 1
12          /*
13          * Could be EL3/EL2/EL1, Initial State:
14          * Little Endian, MMU Disabled, i/dCache Disabled
15          */
16          //adr   x0, vectors
17          switch_el x1, 3f, 2f, 1f
18      3:  msr vbar_el3, x0
19          mrs x0, scr_el3
20          orr x0, x0, #0xf            /* SCR_EL3.NS|IRQ|FIQ|EA */
21          msr scr_el3, x0
22          msr cptr_el3, xzr           /* Enable FP/SIMD */
23          ldr x0, =240000000
24          msr cntfrq_el0, x0          /* Initialize CNTFRQ */
25          b   0f
26      2:  msr vbar_el2, x0
27          mov x0, #0x33ff
28          msr cptr_el2, x0            /* Enable FP/SIMD */
29          b   0f
30      1:  msr vbar_el1, x0
31          mov x0, #3 << 20
32          msr cpacr_el1, x0           /* Enable FP/SIMD */
33      0:
34
35
36          /*
37          * Cache/BPB/TLB Invalidate
38          * i-cache is invalidated before enabled in icache_enable()
39          * tlb is invalidated before mmu is enabled in dcache_enable()
40          * d-cache is invalidated before enabled in dcache_enable()
41          */
42
43          bl  lowlevel_init
44
45          branch_if_master x0, x1, master_cpu
46
47          /*
48          * Slave CPUs
49          */
50      slave_cpu:
51          wfe
52          ldr x1, =0x41c00000
53          ldr x0, [x1]
54          cbz x0, slave_cpu
55          br  x0          /* branch to the given address */
56      master_cpu:
57          /* On the master CPU */
58      #endif
```

```
58      #endif
59    _main:
60          mov x0, #1;
61          mov x1, #0;
62          bl main
63          b _main
64
65
66    ENTRY(lowlevel_init)
67          mov x29, x30            /* Save LR */
68
69          branch_if_master x0, x1, 2f
70
71          /*
72           * Slave should wait for master clearing spin table.
73           * This sync prevent salves observing incorrect
74           * value of spin table and jumping to wrong place.
75           */
76
77          /*
78           * All slaves will enter EL2 and optionally EL1.
79           */
80          bl  armv8_switch_to_el2
81
82    2:
83          mov x30, x29            /* Restore LR */
84          ret
85      ENDPROC(lowlevel_init)
86
87      ENTRY(armv8_switch_to_el2)
88      switch_el x0, 1f, 0f, 0f
89      0:  ret
90      1:  armv8_switch_to_el2_m x0
91      ENDPROC(armv8_switch_to_el2)
92
```

实验结果如下：



2.2 完成情况

请给出你完成的设计实验情况（实验效果，可以通过拍照或拍视频的方式），并贴上该设计实验的主要代码（STM32 实验只需要贴上 main.c 文件的代码，ARM 裸机实验只需要贴上自己编写的.c 文件、.s 文件的代码）。