

《嵌入式系统》

(实验三 STM32、ARM裸机设计实验)
(答案)

厦门大学信息学院软件工程系 曾文华

2024年10月22日

目录

• 一、STM32设计实验

- 1-1 小键盘控制步进电机
- 1-2 小键盘控制舵机
- 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
- 1-7 电子钟
- 1-8 在数码管上显示温度采集值

• 二、ARM裸机设计实验

- 2-1 LED灯（混合编程）
- 2-2 呼吸灯（汇编语言）
- 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）

一、STM32设计实验

设计实验1-1：小键盘控制步进电机

- 请编写程序，实现通过小键盘控制步进电机的转动；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。
- 思路：
 - 1、在直流电机“3_DC_Motor”实验工程的基础上修改程序，将新的实验工程命名为“19_Key_Stepper_Motor”
 - 2、去掉“19_Key_Stepper_Motor”实验工程中的Dc_motor.c、tim.c、Dc_motor.h、tim.h文件
 - 3、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
 - 4、综合“3_DC_Motor”和“5_Stepper_Motor”实验工程的主.c，得到新的main.c
 - 5、在MDK中打开“19_Key_Stepper_Motor”，在工程中去掉Dc_motor.c、tim.c文件
 - 6、编译、下载、运行程序

```

/***** 该程序实现通过小键盘控制步进电机转到的功能 *****/

```

```

*****/
按小键盘 "1" 则顺时针转 按 "2" 则逆时针转 按其它14个键则停转 ****/

```

```

#include "main.h"
#include "usart.h"
#include "gpio.h"
#include "zlg72128.h"
#include "stdio.h"
#include "i2c.h"

```

```

#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          //显示缓冲区首地址
#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_WritePin(GPIOH, GPIO_PIN_13,GPIO_PIN_RESET);HAL_GPIO_WritePin(GPIOE, GPIO_PIN_4,GPIO_PIN_RESET);HAL_GP
#define DE_B                    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12,GPIO_PIN_RESET);HAL_GPIO_WritePin(GPIOH, GPIO_PIN_13,GPIO_PIN_SET);HAL_GPIO_WritePin(GPIOE, GPIO_PIN_4,GPIO_PIN_RESET);HAL_GP
#define DE_C                    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12,GPIO_PIN_RESET);HAL_GPIO_WritePin(GPIOH, GPIO_PIN_13,GPIO_PIN_RESET);HAL_GPIO_WritePin(GPIOE, GPIO_PIN_4,GPIO_PIN_SET);HAL_GP
#define DE_D                    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12,GPIO_PIN_RESET);HAL_GPIO_WritePin(GPIOH, GPIO_PIN_13,GPIO_PIN_RESET);HAL_GPIO_WritePin(GPIOE, GPIO_PIN_4,GPIO_PIN_RESET);HAL_
#define DE_AB                   HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12,GPIO_PIN_SET);HAL_GPIO_WritePin(GPIOH, GPIO_PIN_13,GPIO_PIN_SET);HAL_GPIO_WritePin(GPIOE, GPIO_PIN_4,GPIO_PIN_RESET);HAL_GPIO
#define DE_BC                   HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12,GPIO_PIN_RESET);HAL_GPIO_WritePin(GPIOH, GPIO_PIN_13,GPIO_PIN_SET);HAL_GPIO_WritePin(GPIOE, GPIO_PIN_4,GPIO_PIN_SET);HAL_GPIO
#define DE_CD                   HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12,GPIO_PIN_RESET);HAL_GPIO_WritePin(GPIOH, GPIO_PIN_13,GPIO_PIN_RESET);HAL_GPIO_WritePin(GPIOE, GPIO_PIN_4,GPIO_PIN_SET);HAL_GP
#define DE_DA                   HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12,GPIO_PIN_SET);HAL_GPIO_WritePin(GPIOH, GPIO_PIN_13,GPIO_PIN_RESET);HAL_GPIO_WritePin(GPIOE, GPIO_PIN_4,GPIO_PIN_RESET);HAL_GP

```

```

uint8_t flag = 0xff;           //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 SystemClock_Config(void);
void swtich_key(void);
void swtich_key_func(void);
void switch_flag(void);
void delay_my(uint8_t time);

```

答案


“19_Key_Stepper_Motor”实验工程main.c文件的开头部分


答案

“19_Key_Stepper_Motor”实验工程main.c文件的main函数

```
int main(void)
{
    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

    printf("FS-STM32开发板小键盘控制步进电机测试实验\r\n"); //在串口调试助手上显示打印的内容

    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_Buffer_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
    }
}
```

答案



```
if(flag == 1)
{
```

```
    /******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);
```

“19_Key_Stepper_Motor”实验工程main.c文件的main函数



```
    }
    if(flag == 2)
    {
```

```
        /******八拍方式***** 逆时针转    DA D CD C BC B AB A
        DE_DA;
        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_D;
        HAL_Delay(1);
        DE_CD;
        HAL_Delay(1);
        DE_C;
        HAL_Delay(1);
        DE_BC;
        HAL_Delay(1);
        DE_B;
        HAL_Delay(1);
        DE_AB;
        HAL_Delay(1);
        DE_A;
        HAL_Delay(1);
```

```
    }
```

```
}
```

```
}
```

设计实验1-2：小键盘控制舵机

- 请编写程序，实现通过小键盘控制舵机的转动；按“1”键，舵机转动；按其它15个键，舵机不转。
- 思路：
 - 1、在直流电机“3_DC_Motor”实验工程的基础上修改程序，将新的实验工程命名为“20_Key_Steering_Engine”
 - 2、去掉“20_Key_Steering_Engine”实验工程中的Dc_motor.c、Dc_motor.h文件
 - 3、用“4_Steering_Engine”实验工程中的gpio.c、stm32f4xx_it.c、tim.c、gpio.h、stm32f4xx_it.h、tim.h，代替“20_Key_Steering_Engine”实验工程中的gpio.c、stm32f4xx_it.c、tim.c、gpio.h、stm32f4xx_it.h、tim.h
 - 4、用“11_ZLG72128”实验工程中的i2c.c、i2c.h，代替“20_Key_Steering_Engine”实验工程中的i2c.c、i2c.h
 - 5、综合“3_DC_Motor”和“4_Steering_Engine”实验工程的主.c，得到新的main.c
 - 6、在MDK中打开“20_Key_Steering_Engine”，在工程中去掉Dc_motor.c文件
 - 7、编译、下载、运行程序

答案

main.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
#include "main.h"
#include "i2c.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"
#include "zlg72128.h"
#include "stdio.h"
```

“20_Key_Steering_Engine”实验工程main.c文件的开头部分

```
#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          //显示缓冲区首地址
#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)))
```

```
uint8_t flag = 0xff;                //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 SystemClock_Config(void);
void swtich_key(void);
void swtich_key_func(void);
void switch_flag(void);
void delay_my(uint8_t time);
```

答案

“20_Key_Steering_Engine”实验工程main.c文件的main函数

```
int main(void)
{
    uint16_t i;
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_TIM12_Init();
    MX_I2C1_Init();
    MX_USART1_UART_Init();
    HAL_TIM_PWM_Start(&htim12, TIM_CHANNEL_1);    //启动舵机

    printf("FS-STM32开发板舵机测试实验\r\n");      //在串口调试助手上显示打印的内容

    while (1)
    {
        for(i = 600; i < 2400; i++)
        {
            I2C_ZLG72128_Read(&hi2c1, 0x61, 0x01, Rx1_Buffer_P, 1); //读普通按键值
            I2C_ZLG72128_Read(&hi2c1, 0x61, 0x03, Rx1_Buffer_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)
            {
                __HAL_TIM_SetCompare(&htim12, TIM_CHANNEL_1, i);    //舵机从右到左转到
                HAL_Delay(1);    //延时1ms
            }
        }

        for(i = 2400; i > 600; i--)
        {
            I2C_ZLG72128_Read(&hi2c1, 0x61, 0x01, Rx1_Buffer_P, 1); //读普通按键值
            I2C_ZLG72128_Read(&hi2c1, 0x61, 0x03, Rx1_Buffer_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)
            {
                __HAL_TIM_SetCompare(&htim12, TIM_CHANNEL_1, i);    //舵机从左到右转动
                HAL_Delay(1);    //延时1ms
            }
        }
    }
}
```

设计实验1-3：小键盘控制LED灯

- 请编写程序，实现通过小键盘控制LED灯亮灭；按“1、2、3、A”键，对应的LED1、LED2、LED3、LED4灯亮；按“4、5、6、B”键，对应的LED1、LED2、LED3、LED4灯灭。
- 思路：
 - 1、在直流电机“3_DC_Motor”实验工程的基础上修改程序，将新的实验工程命名为“21_Key_Led”
 - 2、去掉“21_Key_Led”实验工程中的Dc_motor.c、tim.c、Dc_motor.h、tim.h文件
 - 3、用“1_Led”实验工程中的gpio.c、stm32f4xx_it.c、gpio.h、stm32f4xx_it.h，代替“21_Key_Led”实验工程中的gpio.c、stm32f4xx_it.c、gpio.h、stm32f4xx_it.h
 - 4、综合“3_DC_Motor”和“1_Led”实验工程的主.c，得到新的main.c
 - 5、在MDK中打开“21_Key_Led”，在工程中去掉Dc_motor.c、tim.c文件
 - 6、编译、下载、运行程序

答案

main.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
#include "stdio.h"
#include "main.h"
#include "i2c.h"
#include "usart.h"
#include "gpio.h"
#include "zlg72128.h"
```

“21_Key_Led”实验工程main.c文件的开头部分

```
#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      //显示缓冲区首地址
#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)))
```

```
uint8_t flag = 0xff;           //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 SystemClock_Config(void);
void swtich_key(void);
void swtich_key_func(void);
void switch_flag(void);
void delay_my(uint8_t time);
```

答案

```
int main(void)
{
    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
    printf("\n\r FS-STM32开发板通过小键盘控制LED的亮灭实验\r\n"); //在串口调试助手上显示打印的内容
    HAL_GPIO_WritePin(GPIOF,GPIO_PIN_10,GPIO_PIN_SET); //LED1灭
    HAL_GPIO_WritePin(GPIOC,GPIO_PIN_0,GPIO_PIN_SET); //LED2灭
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_SET); //LED3灭
    HAL_GPIO_WritePin(GPIOH,GPIO_PIN_15,GPIO_PIN_SET); //LED4灭
    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_Buffer_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
            printf("flag = %d\r\n", flag);
        }
        if (Rx1_Buffer_T[0] != 0xff) //功能按键 (4个)
        {
            swtich_key_func(); //键值转换
            printf("flag = %d\r\n", flag);
        }
        if(flag == 1)
            HAL_GPIO_WritePin(GPIOF,GPIO_PIN_10,GPIO_PIN_RESET); //LED1亮
        if(flag == 4)
            HAL_GPIO_WritePin(GPIOF,GPIO_PIN_10,GPIO_PIN_SET); //LED1灭
        if(flag == 2)
            HAL_GPIO_WritePin(GPIOC,GPIO_PIN_0,GPIO_PIN_RESET); //LED2亮
        if(flag == 5)
            HAL_GPIO_WritePin(GPIOC,GPIO_PIN_0,GPIO_PIN_SET); //LED2灭
        if(flag == 3)
            HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_RESET); //LED3亮
        if(flag == 6)
            HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_SET); //LED3灭
        if(flag == 10)
            HAL_GPIO_WritePin(GPIOH,GPIO_PIN_15,GPIO_PIN_RESET); //LED4亮
        if(flag == 11)
            HAL_GPIO_WritePin(GPIOH,GPIO_PIN_15,GPIO_PIN_SET); //LED4灭
    }
}
```

“21_Key_Led”实验工程main.c文件的main函数



设计实验1-4：小键盘控制蜂鸣器

- 请编写程序，实现通过小键盘控制蜂鸣器的响/不响；按“1”键，蜂鸣器响；按其它15个键，蜂鸣器不响。
- 思路：
 - 1、在直流电机“3_DC_Motor”实验工程的基础上修改程序，将新的实验工程命名为“22_Key_Beep”
 - 2、去掉“21_Key_Beep”实验工程中的Dc_motor.c、tim.c、Dc_motor.h、tim.h文件
 - 3、用“2_Beep”实验工程中的gpio.c、stm32f4xx_it.c、gpio.h、stm32f4xx_it.h，代替“21_Key_Beep”实验工程中的gpio.c、stm32f4xx_it.c、gpio.h、stm32f4xx_it.h
 - 4、综合“3_DC_Motor”和“2_Beep”实验工程的主.c，得到新的main.c
 - 5、在MDK中打开“22_Key_Beep”，在工程中去掉Dc_motor.c、tim.c文件
 - 6、编译、下载、运行程序

答案

main.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
#include "stdio.h"
#include "main.h"
#include "usart.h"
#include "gpio.h"
#include "zlg72128.h"
#include "i2c.h"
```

```
#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 BUFFER_SIZE1            (countof(Tx1_Buffer))
#define BUFFER_SIZE2            (countof(Rx2_Buffer))
#define countof(a)              (sizeof(a) / sizeof(*(a)))
```

```
uint8_t flag = 0xff;
uint8_t Rx2_Buffer[8]={0};
uint8_t Tx1_Buffer[8]={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};

//不同的按键有不同的标志位值
```

```
void SystemClock_Config(void);
void swtich_key(void);
void swtich_key_func(void);
void switch_flag(void);
```

“22_Key_Beep”实验工程main.c文件的开头部分

答案

```
int main(void)
```

```
{
```

```
    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
```

```
    printf("\n\r FS-STM32开发板小键盘控制蜂鸣器实验\r\n"); //在串口调试助手上显示打印的内容
```

```
    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_Buffer_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
            printf("flag = %d\n", flag);
```


```
        }
```

```
        if (Rx1_Buffer_T[0] != 0xff) //功能按键 (4个)
```

```
        {
```

```
            swtich_key_func(); //键值转换
            printf("flag = %d\n", flag);
```

```
        }
```

```
         if(flag == 1)
```

```
        {
```

```
            HAL_GPIO_WritePin(GPIOG,GPIO_PIN_6,GPIO_PIN_SET);
            HAL_Delay(1); //延时1ms
            HAL_GPIO_WritePin(GPIOG,GPIO_PIN_6,GPIO_PIN_RESET);
            HAL_Delay(1); //延时1ms
```

```
        }
```

```
    }
```

```
}
```

“22_Key_Beep”实验工程main.c文件的main函数

设计实验1-5： 红外遥控器控制LED灯

- 请编写程序，实现通过红外遥控器控制LED灯的亮/灭；按遥控器上的“1、2、3、4”键，对应的LED1、LED2、LED3、LED4灯亮；按遥控器上的“5、6、7、8”键，对应的LED1、LED2、LED3、LED4灯灭。
- 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“23_IR_Receive_Led”
 - 2、将“13_IR_Receive”和“1_Led”实验工程的gpio.c，结合起来，成为新的gpio.c
 - 3、综合“13_IR_Receive”和“1_Led”实验工程的main.c，得到新的main.c
 - 4、编译、下载、运行程序

#include "gpio.h"

“23_IR_Receive_Led”实验工程的gpio.c文件

```
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_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    HAL_GPIO_WritePin(GPIOH, GPIO_PIN_15, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOF, GPIO_PIN_10, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_RESET);
```

```
GPIO_InitStruct.Pin = GPIO_PIN_15;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
```

```
HAL_GPIO_Init(GPIOH, &GPIO_InitStruct);
```

```
GPIO_InitStruct.Pin = GPIO_PIN_10;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
```

```
HAL_GPIO_Init(GPIOF, &GPIO_InitStruct);
```

```
GPIO_InitStruct.Pin = GPIO_PIN_0;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
```

```
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

```
GPIO_InitStruct.Pin = GPIO_PIN_15;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
```

```
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

}

答案

```
main.c - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
#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
```

```
//    printf("flag = %d\n",flag);
```



```
        if(flag == 8)
            HAL_GPIO_WritePin(GPIOF,GPIO_PIN_10,GPIO_PIN_RESET);           //LED1亮
```

```
        if(flag == 40)
            HAL_GPIO_WritePin(GPIOF,GPIO_PIN_10,GPIO_PIN_SET);              //LED1灭
```

```
        if(flag == 136)
            HAL_GPIO_WritePin(GPIOC,GPIO_PIN_0,GPIO_PIN_RESET);             //LED2亮
```

```
        if(flag == 168)
            HAL_GPIO_WritePin(GPIOC,GPIO_PIN_0,GPIO_PIN_SET);               //LED2灭
```

```
        if(flag == 72)
            HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_RESET);            //LED3亮
```

```
        if(flag == 232)
            HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_SET);              //LED3灭
```

```
        if(flag == 200)
            HAL_GPIO_WritePin(GPIOH,GPIO_PIN_15,GPIO_PIN_RESET);            //LED4亮
```

```
        if(flag == 24)
            HAL_GPIO_WritePin(GPIOH,GPIO_PIN_15,GPIO_PIN_SET);              //LED4灭
```

```
    }
```

```
}
```

“23_IR_Receive_Led”实验工程的main.c文件

设计实验1-6： 红外遥控器控制蜂鸣器

- 请编写程序，实现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。
- 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”实验工程的主.c，得到新的main.c
 - 4、编译、下载、运行程序

答案

gpio.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
#include "gpio.h"
```

“23_IR_Receive_Beep”实验工程的gpio.c文件

```
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);
}
```

答案

main.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

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

“23_IR_Receive_Beep”实验工程的main.c文件

```
_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
```

```
        }
```

```
    }
```

```
}
```

设计实验1-7： 电子钟

- 请编写程序，实现在数码管上显示时、分、秒，每1秒变化1次。



- 思路：

- 1、在小键盘/数码管 “11_ZLG72128”实验工程的基础上修改程序，将新的实验工程命名为“25_Clock”
- 2、“0-9”的七段码为：0x3f、0x06、0x5b、0x4f、0x66、0x6d、0x7d、0x07、0x7f、0x6f；“-”的七段码为0x40
- 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”

- 4、修改“11_ZLG72128”实验工程的main.c，得到新的main.c
- 5、编译、下载、运行程序

答案

main.c - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
#include "main.h"  
#include "i2c.h"  
#include "usart.h"  
#include "gpio.h"  
#include "zlg72128.h"  
#include "stdio.h"
```

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

```
uint8_t flag = 0xff;      //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};
```

```
uint8_t hour = 23;  
uint8_t minute = 59;  
uint8_t second = 50;
```

```
uint8_t hour_high = 0;  
uint8_t hour_low = 0;  
uint8_t minute_high = 0;  
uint8_t minute_low = 0;  
uint8_t second_high = 0;  
uint8_t second_low = 0;
```

“25_Clock”实验工程main.c文件的开头部分

答案

“25_Clock”实验工程main.c文件的main函数

```
void SystemClock_Config(void);
void switch_key(void);
void switch_key_func(void);
void switch_flag(void);
void delay_my(uint8_t time);

uint8_t convert(uint8_t hour_minute_second);

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;
    }
}
```

答案

“25_Clock”实验工程main.c文件的main函数

显示“时”



```
Tx1_Buffer[0] = convert(hour_high);  
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS8,Tx1_Buffer);  
Tx1_Buffer[0] = convert(hour_low);  
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS7,Tx1_Buffer);
```

显示“-”



```
Tx1_Buffer[0] = 0x40;  
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS6,Tx1_Buffer);
```

显示“分”



```
Tx1_Buffer[0] = convert(minute_high);  
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS5,Tx1_Buffer);  
Tx1_Buffer[0] = convert(minute_low);  
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS4,Tx1_Buffer);
```

显示“-”



```
Tx1_Buffer[0] = 0x40;  
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS3,Tx1_Buffer);
```

显示“秒”



```
Tx1_Buffer[0] = convert(second_high);  
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS2,Tx1_Buffer);  
Tx1_Buffer[0] = convert(second_low);  
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS1,Tx1_Buffer);
```

```
HAL_Delay(1000);
```

```
//延时1000ms
```

```
}
```

```
}
```

“25_Clock”实验工程main.c文件的convert函数

答案

```
→ uint8_t convert(uint8_t hour_minute_second)
{
    switch(hour_minute_second)
    {
        case 0:
            return 0x3f;
        case 1:
            return 0x06;
        case 2:
            return 0x5b;
        case 3:
            return 0x4f;
        case 4:
            return 0x66;
        case 5:
            return 0x6d;
        case 6:
            return 0x7d;
        case 7:
            return 0x07;
        case 8:
            return 0x7f;
        case 9:
            return 0x6f;
    }
}
```

将0至9转换为相应的七段码

设计实验1-8：在数码管上显示温度采集值

- 请编写程序，实现在数码管上显示采集到的温度传感器值。



- 思路：

- 1、在温度采集“14_Temp”实验工程的基础上修改程序，将新的实验工程命名为“26_Display_Temp”
- 2、将小键盘/数码管“11_ZLG72128”实验工程中的zlg72128.c、zlg72128.h文件，拷贝到“23_Display_Temp”实验工程中
- 3、“0.-9.”的七段码为：0xbf、0x86、0xdb、0xcf、0xe6、0xed、0xfd、0x87、0xff、0xef
- 4、综合“11_ZLG72128”和“14_Temp”实验工程的main.c，得到新的main.c
- 5、在MDK中打开“21_Key_Led”，在工程中增加zlg72128.c文件
- 6、编译、下载、运行程序

答案

main.c - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
#include "stdio.h"
#include "main.h"
#include "i2c.h"
#include "usart.h"
#include "gpio.h"
#include "LM75A.h"
#include "zlg72128.h"
```

```
uint16_t Temp;
```

```
uint8_t Temp_integerPart1;
uint8_t Temp_integerPart2;
```

```
uint8_t Temp_fractionalPart1;
uint8_t Temp_fractionalPart2;
uint8_t Temp_fractionalPart3;
uint8_t Temp_fractionalPart4;
uint8_t Temp_fractionalPart5;
uint8_t Temp_fractionalPart6;
```

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

#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)))
```

“26_Display_Temp”实验工程main.c文件的开头部分

答案

```
uint8_t flag = 0xff;           //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 SystemClock_Config(void);
void swtich_key(void);
void swtich_key_func(void);
void switch_flag(void);
void delay_my(uint8_t time);

uint8_t convert(uint8_t temp_display);
uint8_t convert_dot(uint8_t temp_display);

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_I2C1_Init();
    MX_USART1_UART_Init();
    LM75SetMode(CONF_ADDR,NORMOR_MODE);

    while (1)
    {
        if((Temp = LM75GetTempReg()) != EVL_ER)
        {
            LM75GetTempValue(Temp);           //获取温度传感器的值
        }
    }
}
```

“26_Display_Temp”实验工程main.c文件的开头部分

```

Temp_integerPart1 = Temp*0.125/10;           //温度值整数部分的十位
Temp_integerPart2 = Temp*0.125 - Temp_integerPart1*10; //温度值整数部分的个位

Temp_fractionalPart1 = (Temp - (Temp_integerPart1*10 + Temp_integerPart2)*8)*0.125*10; //温度值小数点后第1位
Temp_fractionalPart2 = ((Temp - (Temp_integerPart1*10 + Temp_integerPart2)*8)*0.125*10)*10 - Temp_fractionalPart1*10; //温度值小数点后第2位
Temp_fractionalPart3 = (((Temp - (Temp_integerPart1*10 + Temp_integerPart2)*8)*0.125*10)*10 - Temp_fractionalPart1*10 - Temp_fractionalPart2*10; //温度值小数点后第3位
Temp_fractionalPart4 = ((((Temp - (Temp_integerPart1*10 + Temp_integerPart2)*8)*0.125*10)*10 - Temp_fractionalPart1*10)*10 - Temp_fractionalPart2*10 - Temp_fractionalPart3*10;
Temp_fractionalPart5 = ((((((Temp - (Temp_integerPart1*10 + Temp_integerPart2)*8)*0.125*10)*10 - Temp_fractionalPart1*10)*10 - Temp_fractionalPart2*10)*10 - Temp_fractionalPart3*10 - Temp_fractionalPart4*10;
Temp_fractionalPart6 = (((((((Temp - (Temp_integerPart1*10 + Temp_integerPart2)*8)*0.125*10)*10 - Temp_fractionalPart1*10)*10 - Temp_fractionalPart2*10)*10 - Temp_fractionalPart3*10 - Temp_fractionalPart4*10 - Temp_fractionalPart5*10;

```



```

Tx1_Buffer[0] = convert(Temp_integerPart1);
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS8,Tx1_Buffer);

```

```

Tx1_Buffer[0] = convert_dot(Temp_integerPart2);
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS7,Tx1_Buffer);

```

```

Tx1_Buffer[0] = convert(Temp_fractionalPart1);
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS6,Tx1_Buffer);

```

```

Tx1_Buffer[0] = convert(Temp_fractionalPart2);
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS5,Tx1_Buffer);

```

```

Tx1_Buffer[0] = convert(Temp_fractionalPart3);
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS4,Tx1_Buffer);

```

```

Tx1_Buffer[0] = convert(Temp_fractionalPart4);
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS3,Tx1_Buffer);

```

```

Tx1_Buffer[0] = convert(Temp_fractionalPart5);
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS2,Tx1_Buffer);

```

```

Tx1_Buffer[0] = convert(Temp_fractionalPart6);
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS1,Tx1_Buffer);

```

```

}
HAL_Delay(1000);

```

```

}
}

```

前3个数码管不亮

→ Tx1_Buffer[0] = 0x00;
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS8,Tx1_Buffer);

→ Tx1_Buffer[0] = 0x00;
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS7,Tx1_Buffer);

→ Tx1_Buffer[0] = 0x00;
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS6,Tx1_Buffer);

Tx1_Buffer[0] = convert(Temp_integerPart1);
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS5,Tx1_Buffer);

Tx1_Buffer[0] = convert_dot(Temp_integerPart2);
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS4,Tx1_Buffer);

Tx1_Buffer[0] = convert(Temp_fractionalPart1);
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS3,Tx1_Buffer);

Tx1_Buffer[0] = convert(Temp_fractionalPart2);
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS2,Tx1_Buffer);

Tx1_Buffer[0] = convert(Temp_fractionalPart3);
I2C_ZLG72128_Write_char(&hi2c1,0x60,ZLG_WRITE_ADDRESS1,Tx1_Buffer);

}
HAL_Delay(1000);

}
}

答案

“26_Display_Temp”实验工程main.c文件的main函数

→ uint8_t convert(uint8_t temp_display)
{

//不带小数点的显示

答案

“26_Display_Temp”实验工程main.c文件的convert函数

switch(temp_display)
{

case 0:

return 0x3f;

case 1:

return 0x06;

case 2:

return 0x5b;

case 3:

return 0x4f;

case 4:

return 0x66;

case 5:

return 0x6d;

case 6:

return 0x7d;

case 7:

return 0x07;

case 8:

return 0x7f;

case 9:

return 0x6f;

}

}



```
uint8_t convert_dot(uint8_t temp_display)
```

```
//带小数点的显示
```

```
{
```

```
    switch(temp_display)
```

```
    {
```

```
        case 0:
```

```
            return 0xbf;
```

```
        case 1:
```

```
            return 0x86;
```

```
        case 2:
```

```
            return 0xdb;
```

```
        case 3:
```

```
            return 0xcf;
```

```
        case 4:
```

```
            return 0xe6;
```

```
        case 5:
```

```
            return 0xed;
```

```
        case 6:
```

```
            return 0xfd;
```

```
        case 7:
```

```
            return 0x87;
```

```
        case 8:
```

```
            return 0xff;
```

```
        case 9:
```

```
            return 0xef;
```

```
    }
```

```
}
```

答案

“26_Display_Temp”实验工程main.c文件的convert_dot函数

二、ARM裸机设计实验

设计实验2-1： LED灯（混合编程）

- 请采用汇编语言与C语言混合编程的方式编写LED灯实验程序： 01-fs_led
- 编程思路： 请参考03-fs_beep实验工程的程序， 只需要编写（修改） 以下4个程序， 其余的程序与03-fs_beep实验工程中的相同：
 - ① main.c
 - ② fs3399_led.c
 - ③ fs3399_led.h
 - ④ Makefile

01-fs_led实验工程文件夹

- 汇编语言与C语言混合编程的程序

The image shows a Windows File Explorer window displaying the directory structure of the '01-fs_led' project. A red arrow points to the project name in the address bar. Red arrows also point from various files and folders in the left pane to their corresponding entries in the right pane's file list.

Local Disk (C:) > FS3399M4 > ARM裸机实验源码 > 01-fs_led >

名称	修改日期
common	2024/7/11 10:34
include	2024/7/12 17:40
output	2024/6/17 8:34
source	2024/7/11 10:55
start	2024/7/11 10:34
main.c	2024/7/12 17:40
Makefile	2024/7/11 16:49
map.lds	2024/6/17 8:34
rk3399.init	2024/6/17 8:34

Files and folders shown in the left pane:

- gic.h
- macro.h
- exceptions.S
- fs3399_gpio.h
- fs3399_led.h
- fs3399_timer.h
- fs3399_led.c
- fs3399_timer.c
- start.S

01-fs_led实验程序 (1)

- 1、main.c

main.c - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
#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();  
  
    for(;;)  
    {  
        //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);

}

while(1)
{
}

return 0;

}

3个函数（FsLedInit、FsLedOn、FsLedOff）由fs3399_led.c提供

01-fs_led实验程序 (2)

• 2、fs3399_led.c

fs3399_led.c - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
#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;
}

初始化LED灯

→ int FsLedOn(int n)
{
 switch(i)
 {
 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;
 defaule:
 return 0;
 }
}

点亮LED灯

→ int FsLedOff(int n)
{
 switch(i)
 {
 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;
 defaule:
 return 0;
 }
}

熄灭LED灯

01-fs_led实验程序 (3)

- 3、fs3399_led.h

fs_led.h - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
#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__ */
```


01-fs_led实验程序 (4)

• 4、Makefile

Makefile - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

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

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)

OBJS := \$(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)

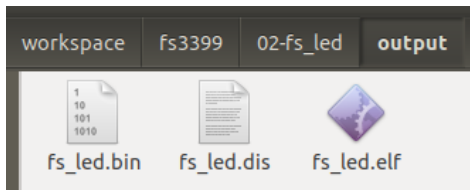
distclean clean:

@ echo " CLEAN complete."

@ rm \$(OBJS) \$(NAME).* main.o -rf

install:

cp \$(NAME).bin /mnt/hgfs/share



编译后生成上述3个文件，
fs_led.elf为可
执行文件

设计实验2-2：呼吸灯（汇编语言）

- 请采用汇编语言的方式编写呼吸灯实验程序：02-fs_assembly_pwm
- 思路：请参考01-fs_assembly_led实验工程的程序，需要编写（修改）以下3个程序，其它程序与01-fs_assembly_led实验工程相同：
 - ① fs_assembly_pwm.s
 - ② Makefile
 - ③ map.lids

02-fs_assembly_pwm实验工程文件夹

- 用汇编语言编写的程序



al Disk (C:) > FS3399M4 > ARM > ARM裸机设计实验 (答案) > 02-fs_assembly_pwm

<input type="checkbox"/> 名称	修改日期
 core	2024/6/17 8:34
 fs_assembly_pwm.s	2024/7/10 22:15
 fs3399.init	2024/6/17 8:34
 Makefile	2024/7/11 10:33
 map.lds	2024/7/11 10:33

02-fs_assembly_pwm实验程序 (1)

- 1、fs_assembly_pwm.s

```
fs_pwm.s - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
.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]
```

答案

//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

    ldr w2,=1000

LOOP3:
    //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
```

02-fs_assembly_pwm实验程序 (2)

- 2、Makefile

```
Makefile - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
#
# 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.lids -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
```

02-fs_assembly_pwm实验程序 (3)

- 3、map.lids

map.lids - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
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) }
```

```
}
```

设计实验2-3：蜂鸣器（汇编语言）

- 请采用汇编语言的方式编写蜂鸣器实验程序：03-fs_assembly_beep
- 思路：请参考01-fs_assembly_led实验工程的程序，只需要编写（修改）以下3个程序，其它程序与01-fs_assembly_led实验工程相同：
 - ① fs_assembly_beep.s
 - ② Makefile
 - ③ map.ld

03-fs_assembly_beep实验工程文件夹

- 用汇编语言编写的程序



Disk (C:) > FS3399M4 > ARM > ARM裸机设计实验 (答案) > 03-fs_assembly_beep		
名称	修改日期	类型
core	2024/6/17 8:34	文件
fs_assembly_beep.s	2024/7/11 14:26	S 文件
fs3399.init	2024/6/17 8:34	INIT 文件
Makefile	2024/7/11 10:31	文件
map.lds	2024/7/11 10:32	LDS 文件

03-fs_assembly_beep实验程序 (1)

• 1、fs_assembly_beep.s

```
fs_beep.s - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
.text
.global _start

_start:
    //设置 蜂鸣器(GPIO1_C7) 为输出模式
    ldr x0,=0xFF730004
    ldr w1,[x0]
    orr w1,w1,#(0x1 << 23)    //w1的第23位置1
    str w1,[x0]

//-----LED1(GPIO4_C6)-----
    //设置蜂鸣器(GPIO1_C7)输出高电平 (蜂鸣器响)
    ldr x0,=0xFF730000
    ldr w1,[x0]
    orr w1,w1,#(0x1 << 23)    //w1的第23位置1
    str w1,[x0]

    //延时
    ldr w3,=0x0FFFFFFF        //设置一个计数值
LOOP1:
    sub w3,w3,#1
    cmp w3,#0
    bne LOOP1

    //设置蜂鸣器(GPIO1_C7)输出高电平 (蜂鸣器不响)
    ldr x0,=0xFF730000
    ldr w1,[x0]
    ldr w2,=0xFF7FFFFFFF
    and w1,w1,w2              //w1的第23位置0
    str w1,[x0]

    //延时
    ldr w3,=0x0FFFFFFF        //设置一个计数值
LOOP2:
    sub w3,w3,#1
    cmp w3,#0
    bne LOOP2
//-----
```

```
//-----LED1(GPIO4_C6)-----
    //设置蜂鸣器(GPIO1_C7)输出高电平 (蜂鸣器响)
    ldr x0,=0xFF730000
    ldr w1,[x0]
    orr w1,w1,#(0x1 << 23)    //w1的第22位置1
    str w1,[x0]

    //延时
    ldr w3,=0x0FFFFFFF        //设置一个计数值
LOOP5:
    sub w3,w3,#1
    cmp w3,#0
    bne LOOP5

    //设置蜂鸣器(GPIO1_C7)输出高电平 (蜂鸣器不响)
    ldr x0,=0xFF730000
    ldr w1,[x0]
    ldr w2,=0xFF7FFFFFFF
    and w1,w1,w2              //w1的第23位置0
    str w1,[x0]

    //延时
    ldr w3,=0x0FFFFFFF        //设置一个计数值
LOOP6:
    sub w3,w3,#1
    cmp w3,#0
    bne LOOP6
//-----

stop:
    b stop
```

.text

.global _start

_start:

```
//设置 蜂鸣器(GPIO1_DDR_C7) 为高电平 (输出模式)
ldr x0,=0xFF730004
ldr w1,[x0]
orr w1,w1,#(0x1 << 23)    //w1的第23位置1
str w1,[x0]
```

//-----LED1(GPIO4_C6)-----

```
//设置蜂鸣器(GPIO1_DR_C7)输出高电平 (蜂鸣器响)
ldr x0,=0xFF730000
ldr w1,[x0]
orr w1,w1,#(0x1 << 23)    //w1的第23位置1
str w1,[x0]
```

```
//延时
ldr w3,=0x0FFFFFFF        //设置一个计数值
```

LOOP1:

```
sub w3,w3,#1
cmp w3,#0
bne LOOP1
```

```
//设置蜂鸣器(GPIO1_DR_C7)输出低电平 (蜂鸣器不响)
ldr x0,=0xFF730000
ldr w1,[x0]
ldr w2,=0xFF7FFFFFFF
and w1,w1,w2                //w1的第23位置0
str w1,[x0]
```

```
//延时
ldr w3,=0x0FFFFFFF        //设置一个计数值
```

LOOP2:

```
sub w3,w3,#1
cmp w3,#0
bne LOOP2
```

//-----

03-fs_assembly_beep实验程序 (2)

- 2、Makefile

```
Makefile - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
#
# System environment variable.
#

NAME                = fs_assembly_beep
CROSS                = ~/toolchain/6.4-aarch64/bin/aarch64-linux-
CC                   = $(CROSS)gcc
LD                   = $(CROSS)ld
OBJCOPY              = $(CROSS)objcopy
OBJDUMP              = $(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
    $(OBJDUMP) -D $(NAME).elf > $(NAME).dis

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

.PHONY: all clean
```

03-fs_assembly_beep实验程序 (3)

- 3、map.lds

map.lds - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
OUTPUT_FORMAT("elf64-littlearch64", "elf64-littlearch64", "elf64-littlearch64")
```

```
OUTPUT_ARCH(aarch64)
```

```
ENTRY(_start)
```

```
SECTIONS
```

```
{
```

```
    . = 0x00280000;
```

```
    . = ALIGN(8);
```

```
    .text :
```

```
    {
```

```
        fs_assembly_beep.o(.text)
```

```
        *(.text)
```

```
    }
```

```
    . = ALIGN(8);
```

```
    .rodata :
```

```
    { *(.rodata) }
```

```
    . = ALIGN(8);
```

```
    .data :
```

```
    { *(.data) }
```

```
    . = ALIGN(8);
```

```
    .bss :
```

```
    { *(.bss) }
```

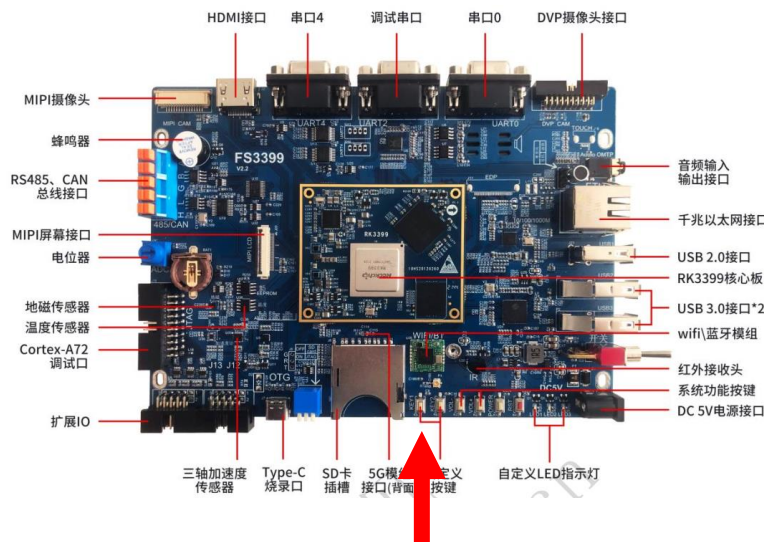
```
}
```

设计实验2-4： 查询方式按键控制蜂鸣器 (混合编程)

- 请采用汇编语言与C语言**混合编程**的方式编写**查询方式按键控制蜂鸣器**的程序（按KEY1键，蜂鸣器响；再按KEY1键，蜂鸣器不响）：**11_fs_loop_key_beep**
- 编程思路：请在**04-fs_loop_key**实验工程的基础上修改程序，并参考**03-fs_beep**实验工程的程序。只需要修改以下2个程序，其它程序与**04-fs_loop_key**实验工程相同：

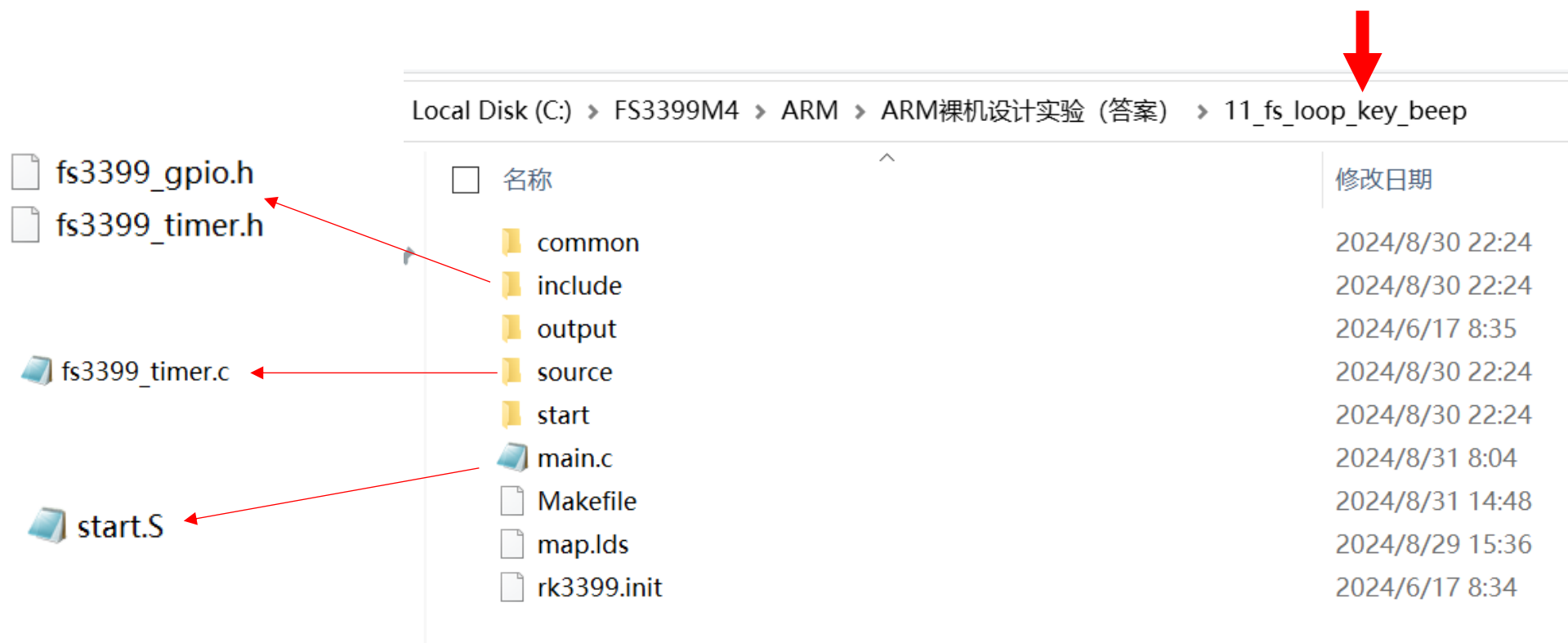
① main.c

② Makefile



11-fs_loop_key_beep实验工程文件夹

- 汇编语言与C语言**混合编程**的程序



11-fs_loop_key_beep实验程序 (1)

main.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
#include "fs3399_gpio.h"
```

```
#include "fs3399_timer.h"
```

```
//按键轮询检测: key1->beep
```

```
int main()
```

```
{
```

```
    //灯的状态标志位
```

```
    int beep_flag = 1;
```

```
    //设置GPIO1_B2(key1) 为输入模式
```

```
    GPIO1->SWPORTA_DDR |= (~0x1 << (1*8 + 2));
```

```
    //设置GPIO1_C7(beep)为输出模式
```

```
    GPIO1->SWPORTA_DDR |= (0x1 << 23);
```

```
    while(1)
```

```
    {
```

```
        //轮询检测KEY1状态
```

```
        if (0 == (GPIO1->EXT_PORTA & (0x1 << (1*8 + 2))))
```

```
        {
```

```
            fs_delay_ms(100); //软件消抖
```

```
            if (0 == (GPIO1->EXT_PORTA & (0x1 << (1*8 + 2))))
```

```
            {
```

```
                if (1 == beep_flag)
```

```
                {
```

```
                    GPIO1->SWPORTA_DR |= (0x1 << 23);
```

```
                    beep_flag = 0;
```

```
                }
```

```
            } else
```

```
            {
```

```
                GPIO1->SWPORTA_DR &= (~0x1 << 23);
```

```
                beep_flag = 1;
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

```
}
```

```
//判断 GPIO1_B2 KEY1 是否按下?
```

```
//判断 GPIO1_B2 KEY1 是否按下? 确定key1确实按下
```

```
//设置GPIO1_C7 输出高电平 (蜂鸣器响)
```

```
//设置GPIO1_C7 输出低电平 (蜂鸣器不响)
```



答案

11-fs_loop_key_beep实验程序 (2)

```
Makefile - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
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_loop_key_beep

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)

OBJS := $(patsubst %.S,%.o,$(OBJSS))

OBJS := $(patsubst %.c,%.o,$(OBJS))

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)

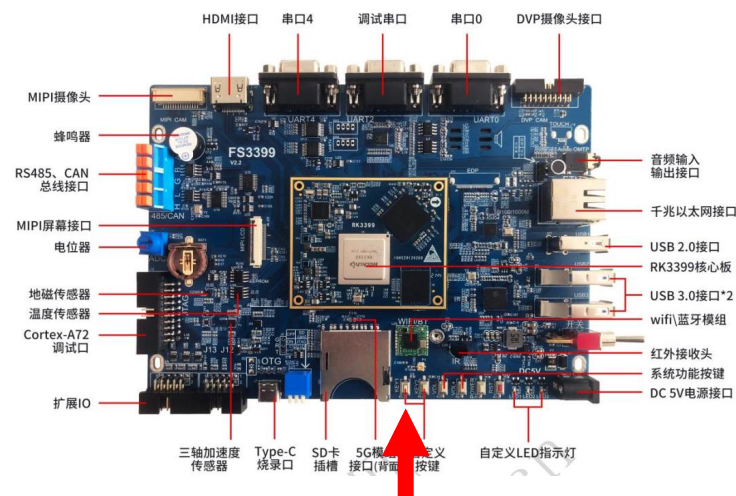
distclean clean:
    @ echo " CLEAN complete."
    @ rm $(OBJS) $(NAME).* main.o -rf

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

设计实验2-5： 中断方式按键控制蜂鸣器 (混合编程)

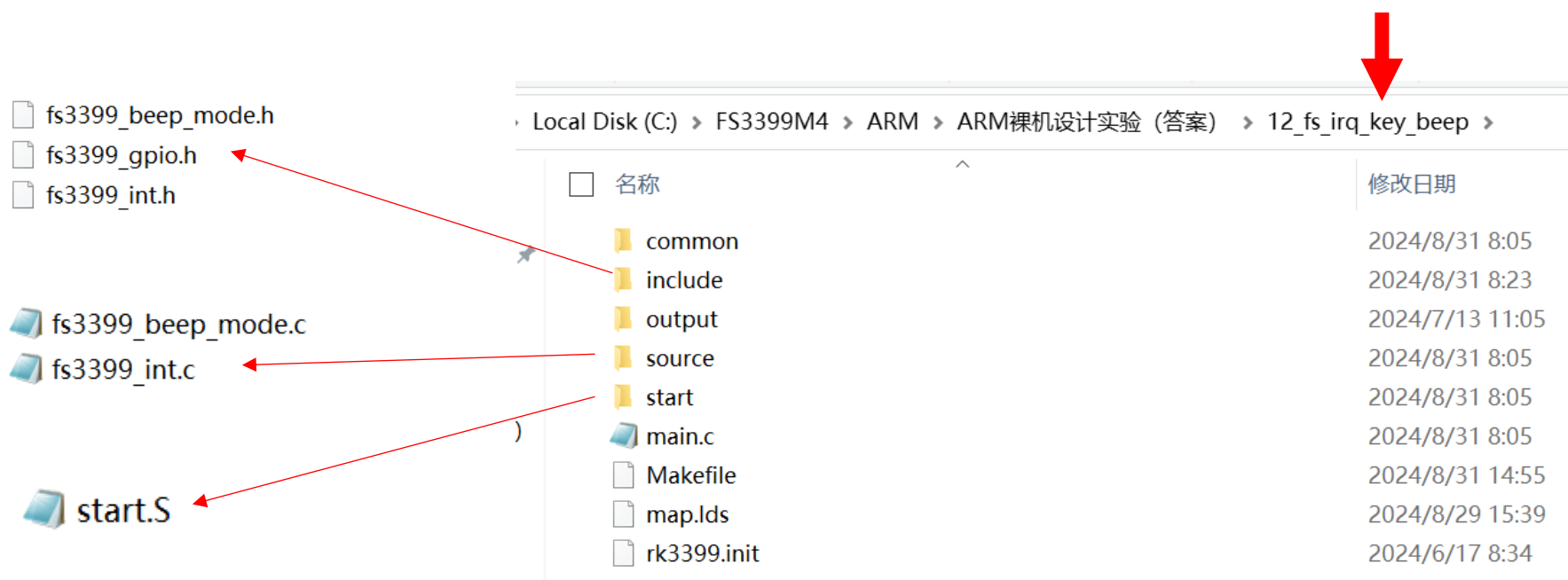
- 请采用汇编语言与C语言**混合编程**的方式编写**中断方式按键控制蜂鸣器**的程序（按KEY1键，蜂鸣器响；再按KEY1键，蜂鸣器不响）：**12_fs_irq_key_beep**
- 编程思路：请在**05-fs_irq_key**实验工程的基础上修改程序，并参考**03-fs_beep**实验工程的程序。只需要修改以下6个程序，其它程序与**05-fs_irq_key**实验工程相同：

- ① main.c
- ② fs3399_int.c
- ③ fs3399_beep_mode.c（修改了文件名）
- ④ fs3399_int.h
- ⑤ fs3399_beep_mode.h（修改了文件名）
- ⑥ Makefile




12-fs_irq_key_beep实验工程文件夹

- 汇编语言与C语言混合编程的程序



答案

12-fs_irq_key_beep实验程序 (1)

 main.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
#include "fs3399_int.h"
```

```
int main()
```

```
{
```

```
    key_beep_irq();
```



```
    while (1)
```

```
    {
```

```
    }
```

```
    return 0;
```

```
}
```

答案

12-fs_irq_key_beep实验程序 (2)

```
fs3399_int.c - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

return 0;
}

//irq 中断处理函数
void do_irq(void)
{
    unsigned long nintid;
    unsigned long long irqstat;
    asm volatile("mrs %0, " __stringify(ICC_IAR1_EL1) : "=r" (irqstat));
    nintid = (unsigned long)irqstat & 0x3FF;
    if (nintid < NR_GIC_IRQS)
        g_irq_handler[nintid].m_func((void *) (unsigned long) nintid);
    asm volatile("msr " __stringify(ICC_EOIR1_EL1) ", %0" :: "r" ((unsigned long long) nintid));
    asm volatile("msr " __stringify(ICC_DIR_EL1) ", %0" :: "r" ((unsigned long long) nintid));
    isb();
}

//KEY1中断响应处理逻辑：按键控制亮灭
static void key_beep_change(void)
{
    if (GPIO1->INT_STATUS & (0x01 << (8 + 2))) //Interrupt status
    {
        GPIO1->PORTA_EOI |= (0x01 << (8 + 2)); //Clear interrupt
        if (flag == 0) {
            beep_mode(1); //蜂鸣器响
            flag = 1;
        } else {
            beep_mode(0); //蜂鸣器不响
            flag = 0;
        }
    }
}

//GPIO1_B2(KEY1)按键中断服务程序
void key_beep_irq(void)
{
    enable_interrupts(); //使能中断
    // GPIO设置
    GPIO1->SWPORTA_DDR &= ~(0x01 << (8 + 2)); //should be Input
    GPIO1->INTEN |= (0x01 << (8 + 2)); //Interrupt enable
    GPIO1->INTMASK &= ~(0x01 << (8 + 2)); //Interrupt bits are unmasked
    GPIO1->INTTYPE_LEVEL |= (0x01 << (8 + 2)); //Edge-sensitive
    GPIO1->INT_POLARITY &= ~(0x01 << (8 + 2)); //Active-low
    GPIO1->DEBOUNCE |= (0x01 << (8 + 2)); //Enable debounce
    // 注册中断函数并使能
    irq_install_handler(GPIO1_INTR, (interrupt_handler_t *) key_beep_change, (void *) (0));
    irq_handler_enable(GPIO1_INTR);
}
```

12-fs_irq_key_beep实验程序 (3)

fs3399_beep_mode.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
#include "fs3399_beep_mode.h"
```

```
void beep_mode(int mode)
```

```
{
```

```
    //设置GPIO1_C7为输出模式 (beep)
```

```
    GPIO1->SWPORTA_DDR |= (0x1 << 23);
```



```
    //设置GPIO1_C7 输出低电平 (蜂鸣器不响)
```

```
    GPIO1->SWPORTA_DR &= (~(0x1 << 23));
```

```
    if(0 == mode)
```

```
        //蜂鸣器不响
```

```
{
```

```
        //设置GPIO1_C7 输出低电平
```

```
        GPIO1->SWPORTA_DR &= (~(0x1 << 23));
```



```
}
```

```
    if(1 == mode)
```

```
        //蜂鸣器响
```

```
{
```

```
        //设置GPIO1_C7 输出高电平
```

```
        GPIO1->SWPORTA_DR |= (0x1 << 23);
```



```
}
```

```
}
```

12-fs_irq_key_beep实验程序 (4)

```
fs3399_int.h - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

unsigned int IACTIVER[32]; //0x0380
unsigned int IPRIORITYR[256]; //0x0400
unsigned int ITARGETSR[256]; //0x0800
unsigned int ICFGR[64]; //0x0c00
unsigned int IGROUPMODR[64]; //0x0d00
unsigned int NSACR[64]; //0x0e00
unsigned int SGIR; //0x0f00
unsigned int RESERVED8[3]; //0x0f04
unsigned int CPENDSGIR[4]; //0x0f10
unsigned int SPENDSGIR[4]; //0x0f20
unsigned int RESERVED9[5236]; //0x0f30
unsigned int IROUTER[1918]; //0x6100
} gicd_reg, *p_gicd;


#define GICD ((p_gicd)RKIO_GICD_PHYS)
#define ICC_IAR0_EL1 S3_0_C12_C8_0
#define ICC_IAR1_EL1 S3_0_C12_C12_0
#define ICC_EOIR0_EL1 S3_0_C12_C8_1
#define ICC_EOIR1_EL1 S3_0_C12_C12_1
#define ICC_HPPIR0_EL1 S3_0_C12_C8_2
#define ICC_HPPIR1_EL1 S3_0_C12_C12_2
#define ICC_BPR0_EL1 S3_0_C12_C8_3
#define ICC_BPR1_EL1 S3_0_C12_C12_3
#define ICC_DIR_EL1 S3_0_C12_C11_1
#define ICC_PMR_EL1 S3_0_C4_C6_0
#define ICC_RPR_EL1 S3_0_C12_C11_3
#define ICC_CTLR_EL1 S3_0_C12_C12_4
#define ICC_CTLR_EL3 S3_6_C12_C12_4
#define ICC_SRE_EL1 S3_0_C12_C12_5
#define ICC_SRE_EL2 S3_4_C12_C9_5
#define ICC_SRE_EL3 S3_6_C12_C12_5
#define ICC_IGRPEN0_EL1 S3_0_C12_C12_6
#define ICC_IGRPEN1_EL1 S3_0_C12_C12_7
#define ICC_IGRPEN1_EL3 S3_6_C12_C12_7
#define ICC_SEIEN_EL1 S3_0_C12_C13_0
#define ICC_SGI0R_EL1 S3_0_C12_C11_7
#define ICC_SGI1R_EL1 S3_0_C12_C11_5
#define ICC_ASGI1R_EL1 S3_0_C12_C11_6

extern void enable_interrupts(void);
extern void irq_install_handler(int irq, interrupt_handler_t *handler, void *data);
extern int irq_handler_enable(int irq);
extern void do_irq(void);
extern void key_beep_irq(void);

#endif
```

答案

12-fs_irq_key_beep实验程序 (5)

 fs3399_beep_mode.h - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
#ifndef __BEEP_H__  
#define __BEEP_H__
```

```
#include "fs3399_gpio.h"
```

```
void beep_mode(int mode);
```

```
#endif
```


答案

12-fs_irq_key_beep实验程序 (6)

```
Makefile - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
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_irq_key_beep
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)

OBJS := $(patsubst %.S,%.o,$(OBJSS))
OBJS := $(patsubst %.c,%.o,$(OBJS))

all:$(OBJS)
    @ echo " LD Linking $(NAME).elf"
    @ $(LD) -Tmap.ld -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
```

设计实验2-6： 串口发送与接收（混合编程）

- 实验工程**06-fs_uart**只完成了串口的发送功能，该工程的“fs3399_uart.c”文件中还提供了串口接收函数“char fs_getc()”。
- 请采用汇编语言与C语言**混合编程**的方式编写**发送与接收**的程序：**13-fs_uart_receive**
- 编程思路：请在**06-fs_uart**实验工程的基础上修改程序，只需要修改以下2个程序，其它程序与**06-fs_uart**实验工程相同：
 - ① main.c
 - ② Makefile

13-fs_uart_receive实验工程文件夹

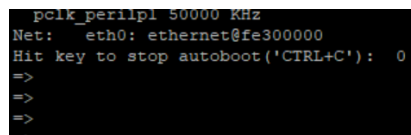
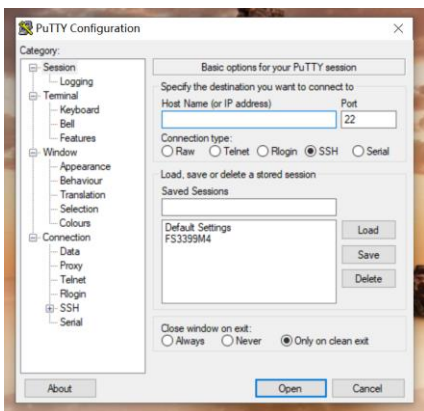
- 汇编语言与C语言混合编程的程序



Local Disk (C:) > FS3399M4 > ARM > ARM裸机设计实验 (答案) > 13-fs_uart_receive >			
<input type="checkbox"/> 名称	修改日期	类型	
common	2024/8/30 22:23	文件夹	
include	2024/8/30 22:23	文件夹	
output	2024/8/30 22:23	文件夹	
source	2024/8/30 22:23	文件夹	
start	2024/8/30 22:23	文件夹	
main.c	2024/8/31 12:42	C 文件	
Makefile	2024/8/29 16:03	文件	
map.lids	2024/8/29 16:04	LDS 文件	
rk3399.init	2024/6/17 8:34	INIT 文件	

13-fs_uart_receive实验工程的运行

- 1、运行PUTTY.EXE（串口连接工具），将其处于“=>”状态
- 2、关闭Putty
- 3、运行UartAssist.exe（串口调试助手），打开串口
- 4、在Ubuntu中运行程序，此时串口调试助手中显示程序发送的结果
- 5、在串口调试助手的“发送”框中输入“abc123”，点击发送，此时串口调试助手的“接收”框中显示“abc123”，表示程序运行正确



13-fs_uart_receive实验程序 (1)

```
main.c - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
{
    char str1[] = "FS3399 UART test string!";
    char str2[] = "Xiamen University";

    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;
}
```

答案


13-fs_uart_receive实验程序 (2)

```
Makefile - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
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.ld -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
```

Thanks