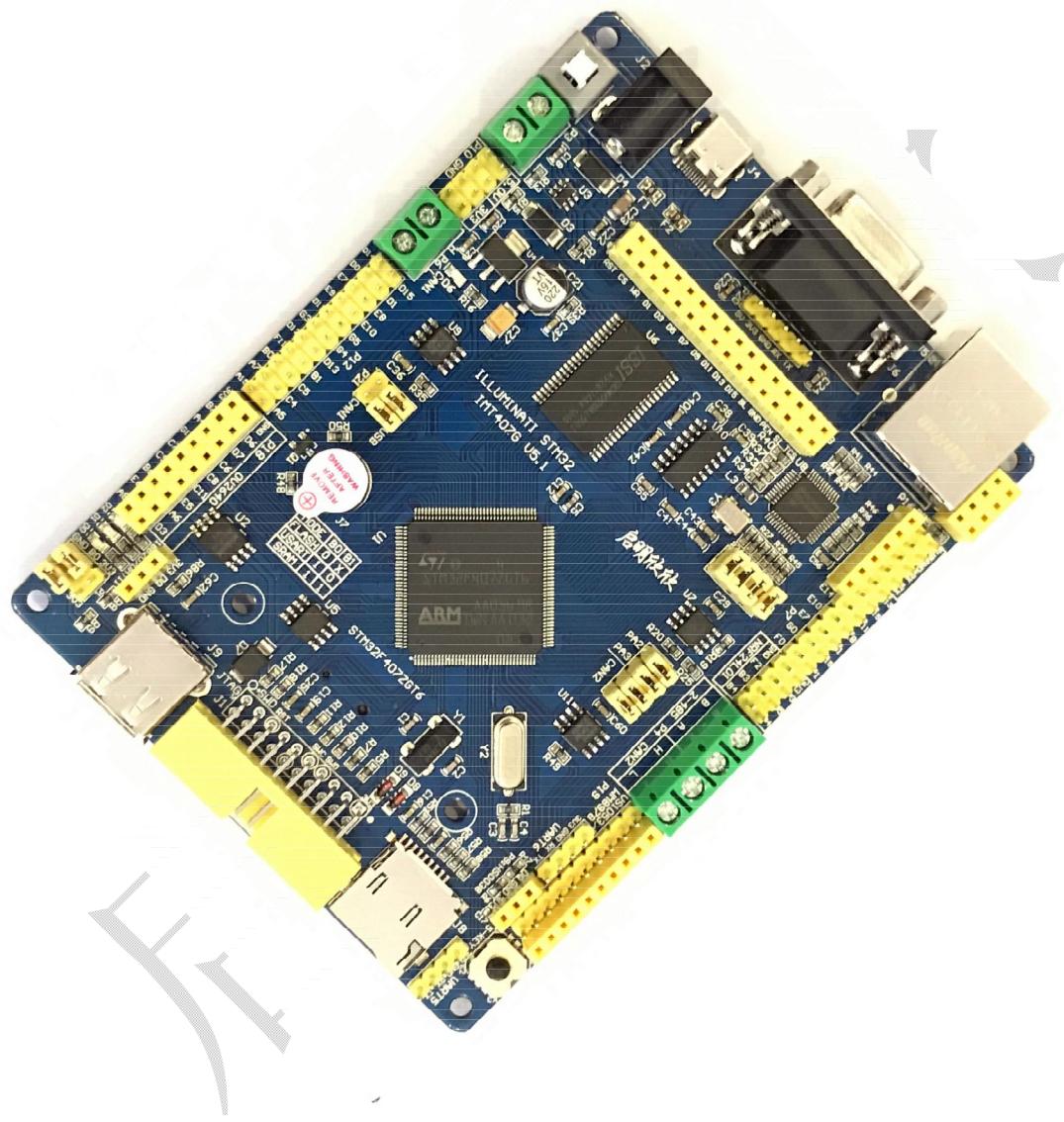


启明欣欣 STM32F407 开发板(高配版) V5.1

——使用手册



官 方 店 铺 : <https://shop125046348.taobao.com>

技术支持: QQ: 3231824766 QQ 交流群: 572537310

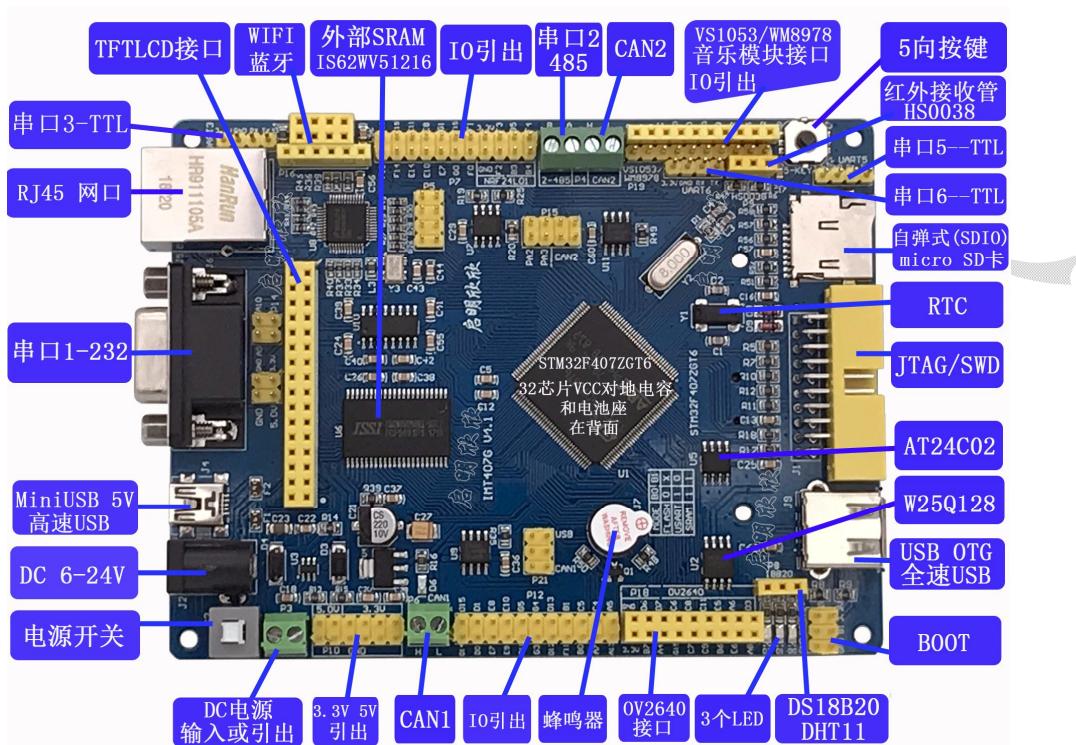
声明: 硬件与赠送资料中所有的软件、文档与手册、原理图和例程代码仅用于学习参考, 我们不承担客户将我们板子和相关资料用于产品, 所产生的一切问题(包括产品质量问题与任何专利、版权问题)。

目 录

第一篇 硬件资源介绍.....	1
第二篇 开发工具与工程架构解析.....	2
第三篇 例程使用详解.....	3
第 1 章 LED 跑马灯.....	3
第 2 章 蜂鸣器使用	4
第 3 章 按键使用	5
第 4 章 外部中断实验.....	6
第 5 章 TFTLCD 显示.....	8
第 6 章 串口 1-RS232 实验	9
第 7 章 定时器中断.....	11
第 8 章 PWM 输出.....	11
第 9 章 独立看门狗	12
第 10 章 模数转换 ADC.....	13
第 11 章 数模转换 DAC.....	14
第 12 章 串口 2-485.....	15
第 13 章 各串口 TTL 实验	17
第 14 章 IIC_24C02	17
第 15 章 SPI_W25Qxx	19
第 16 章 CAN1 与 CAN2 通信.....	20
第 17 章 DMA 实验.....	22
第 18 章 RTC 实时时钟.....	23
第 19 章 汉字显示	24
第 20 章 RTC 农历显示.....	25
第 21 章 内部温度传感器实验	26
第 22 章 DS18B20 温度传感器实验.....	27
第 23 章 红外传感器 HS0038.....	29
第 24 章 触摸屏使用	30
第 25 章 232_485_can 数据转换通信.....	32
第 26 章 USB U 盘(Host).....	33
第 27 章 TCP 服务器数据收发实验.....	35
第 28 章 TCP 客户端数据收发实验.....	40
第 29 章 UDP 客户端数据收发实验.....	45
第 30 章 串口 1-232 与 TCP 服务器双向通信	49

第一篇 硬件资源介绍

1.1 板载资源与接口介绍



1.2 板子供电介绍

电源电路部分 (可不是简单DC进来就直接1117了哦, 有电源芯片哦)

为什么要加电源芯片呢?

答: 因为我们的板子不仅仅能用来学习, 重点是可以用来开发应用的, 有了电源芯片, 可以使板子可靠的长时间工作, 也可以提高电源利用率, 节省电量, 非常适合那种使用电池低功耗的场合。



为什么要引出DC的电压呢?

答: 为了更好的扩展应用, 工业上应用的模块供电多为12V或者24V的电压, 比如GPRS DTU一般用12V电压

这边注意下, DC 座供电用 6V 供电也行, 但是为了供电稳定建议用 6V 以上的。MINI USB 接口接于 STM32 高速 usb IO 口, 用于供

电和进行 **USB** 通信，不能用于下载程序。由于没有加高速芯片，所以 **USB** 也只能运行于全速模式。

第二篇 开发工具与工程架构解析

2.1 开发环境

例程使用 KEIL5(MDK5) 作为开发环境，资料启明欣欣 STM32F407(高配版)资料\常用软件\MDK5 中有安装文件和安装教程。

2.2 程序下载

启明欣欣 STM32F407开发板(高配版)可以使用串口线或者仿真器下载程序，具体怎么下载程序，请看资料启明欣欣 STM32F407(高配版)资料\启明 F407开发板(高配版)程序下载教程。

2.3 程序工程架构解析

The screenshot shows the Keil uVision IDE interface with the following details:

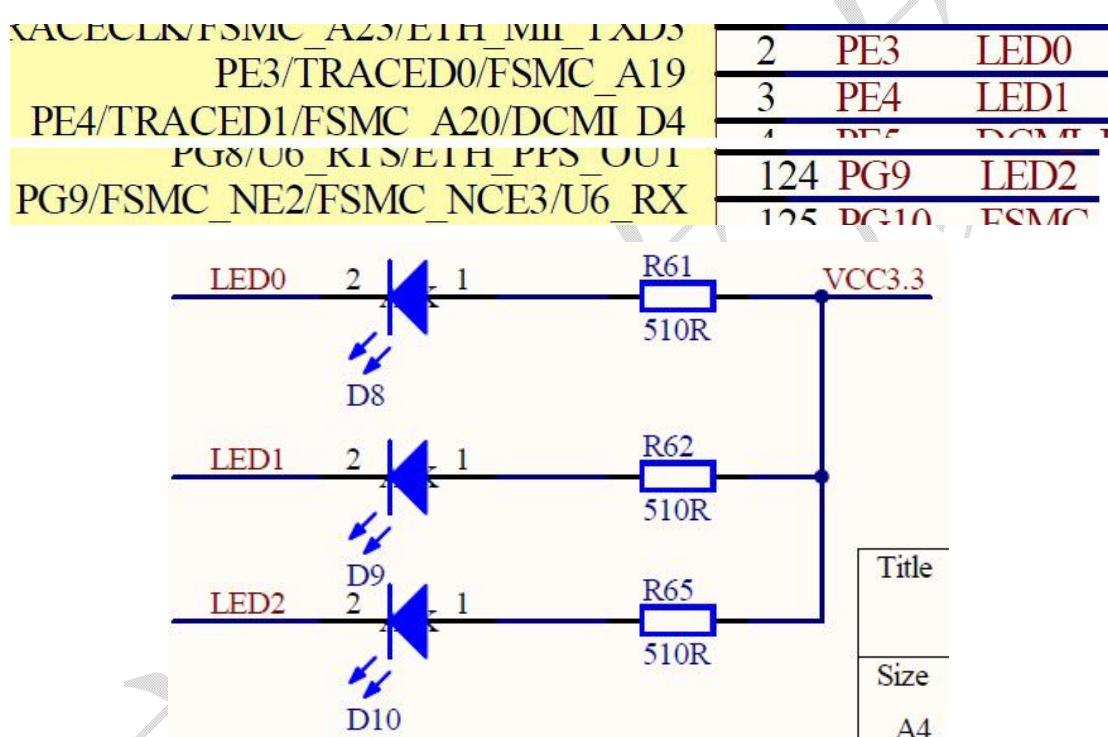
- Project Explorer:** Shows the project structure under "Target 1". Key files include:
 - Main: main.c, stm32f4xx_it.c
 - Startup_config: startup_stm32f407.c, system_stm32f4xx.c
 - USER: lcd.c, led.c, key.c
 - STM32F4_FWLIB: misc.c, stm32f4xx_gpio.c, stm32f4xx_fsmc.c, stm32f4xx_rcc.c, stm32f4xx_usart.c
 - Common: common.c
- Code Editor:** Displays the content of the file "key.h". Red annotations with arrows explain specific parts of the code:
 - Annotations point to the "main.c" file in the project tree and the "main.c" code in the editor, stating: "工程主函数main.c是内核中断函数" (The main function in the project is the interrupt function of the kernel).
 - Annotations point to the "stm32f4xx_it.c" file in the project tree and the "stm32f4xx_it.c" code in the editor, stating: "固体库核心文件与启动文件, 用户无需修改" (Solid library core files and startup files, users do not need to modify).
 - Annotations point to the "lcd.c" file in the project tree and the "LCD_Draw_Rectangle" function in the editor, stating: "用户程序运用区, 就是需要用到什么外设硬件, 就将其添加在此, 基本外设硬件都有自己的一个.C和.H" (User program application area, add what external hardware is needed here, basic peripheral hardware has its own .C and .H files).
 - Annotations point to the "common.c" file in the project tree and the "LCD_Draw_Circle" function in the editor, stating: "库函数源文件与其头文件, 外设需要用到的, 就将对应的库函数添加在此并且在STM32f10x_conf.h中将其对应的头文件去掉注释" (Library function source file and its header file, add the corresponding library functions here and remove the header file comments in STM32f10x_conf.h).
 - Annotations point to the "common.c" file in the project tree and the "LCD_Init" function in the editor, stating: "工程共用的文件, 里面有系统延时函数, 位定义" (Common file for the project, contains system delay functions and bit definitions).

第三篇 例程使用详解

此篇会结合开发板原理图和程序源码，讲解怎么使用各个例程。

第 1 章 LED 跑马灯

3.1.1 硬件介绍



3.1.2 核心代码解析

代码是通过位带操作实现 IO 口控制

```
int main(void)
{
    delay_init(); // 初始化延时函数
    LED_Init(); // 初始化 LED 端口

    while(1)
    {
        LED0=0; //LED0 亮
        LED1=1; //LED1 灭
        LED2=1; //LED2 灭
        delay_ms(500);
        LED0=1; //LED0 灭
        LED1=0; //LED1 亮
    }
}
```

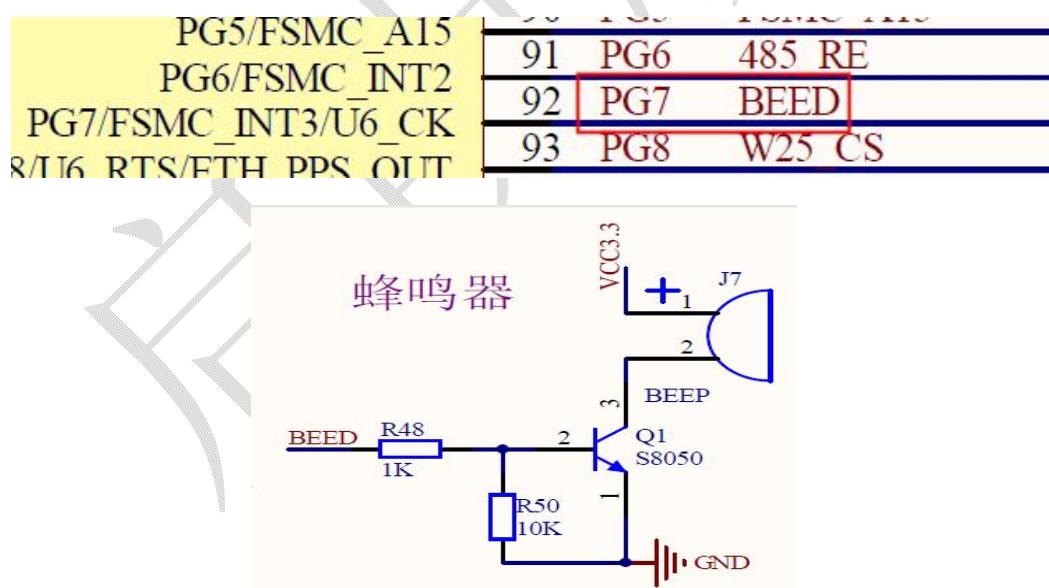
```
LED2=1;          //LED2 灭  
  
delay_ms(500);  
LED0=1;          //LED0 灭  
LED1=1;          //LED1 灭  
LED2=0;          //LED2 亮  
delay_ms(500);  
}  
}
```

3.1.3 实验操作与现象

开发板下入该 LDE 程序，LED 会从 LED0 至 LED2 依次亮灭，形成流水灯效果。

第 2 章 蜂鸣器使用

3.2.1 硬件设计



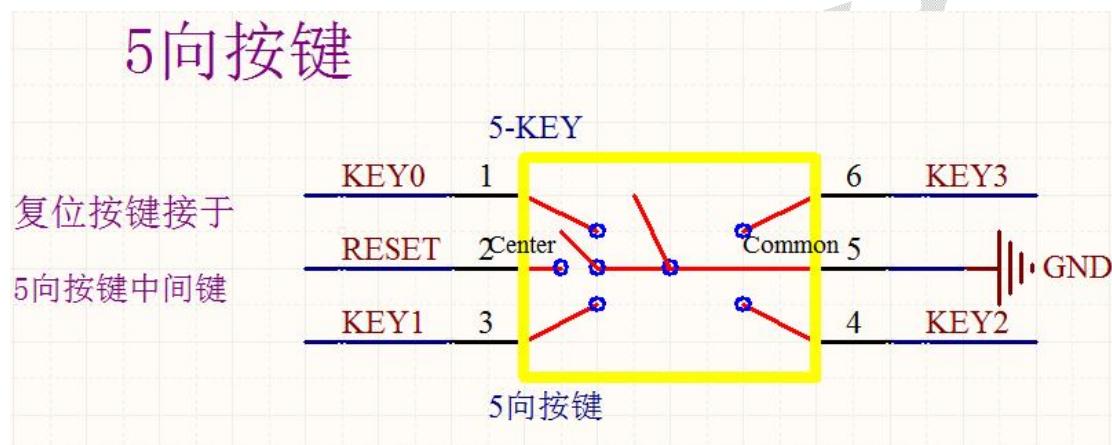
3.2.2 实验操作与现象

程序下进去后，蜂鸣器会一响一闭的。

第 3 章 按键使用

3.3.1 硬件设计

SS_SD	PF5/FSMC_AJ/ADC3_IN10	18	PF6	KEY3
PF6/TIM10_CH1/FSMC_NIORD/ADC3_IN4		19	PF7	KEY2
PF7/TIM11_CH1/FSMC_NREG/ADC3_IN5		20	PF8	KEY1
PF8/TIM13_CH1/FSMC_NIOWR/ADC3_IN6		21	PF9	KEY0
PF9/TIM14_CH1/FSMC_CD/ADC3_IN7		22	PE10	LCD_BT



3.3.2 核心代码解析

```

void key_scan(u8 mode)
{
    keyup_data=0;           //键抬起后按键值一次有效
    if (KEY0==0||KEY1==0||KEY2==0||KEY3==0)   //有键正按下
    {
        if (KEY0==0)      key_tem=1;
        else if (KEY1==0) key_tem=2;
        else if (KEY2==0) key_tem=3;
        else if (KEY3==0) key_tem=4;
        if (key_tem == key_bak) //有键按下后第一次扫描不处理,
与 else 配合第二次扫描有效, 这样实现了去抖动
        {
            key_time++; //有键按下后执行一次扫描函数, 该变
量加 1
            keydown_data=key_tem; //按键值赋予 keydown_data
        }
        if( (mode==0)&&(key_time>1) )//key_time>1 按键值无
效, 这就是单按, 如果 mode 为 1 就为连接
        keydown_data=0;
    }
    else //去抖动
}

```

```

    {
        key_time=0;
        key_bak=key_tem;
    }

}

else //键抬起
{
    if(key_time>2) //按键抬起后返回一次按键值
    {
        keyup_data=key_tem; // 键抬起后按键值赋予
keydown_data
    }
    key_bak=0; //要清零，不然下次执行扫描
程序时按键的值跟上次按的值一样，就没有去抖动处理了
    key_time=0;
    keydown_data=0;
}
}

```

该函数实现按键多种方式的扫描，也实现的多种组合运用，具体请看代码“按键模式剖析”。

3.3.3 实验操作与现象

程序下进去之后，按 KEY0，led 全亮，按 KEY1，蜂鸣器响，长按 KEY2 1 秒后，关闭蜂鸣器与 led0，长按 KEY3 2 秒后，灭掉 led1 和 led2。

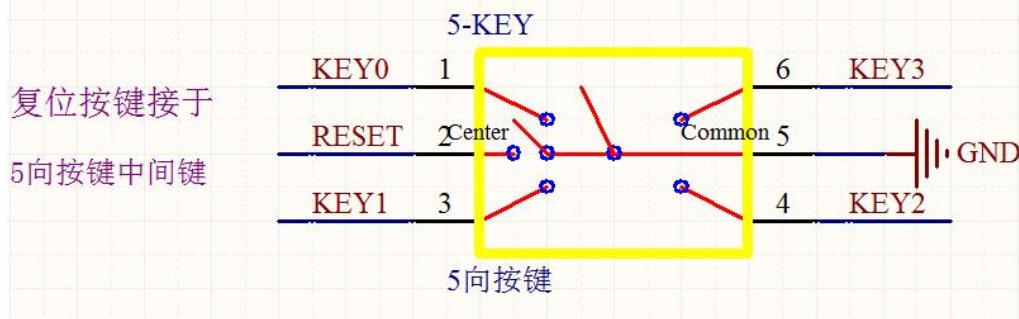
第4章 外部中断实验

使用按键作为外部中断输入源，按键按下触发中断，执行中断服务函数控制蜂鸣器与 led 灯。

3.4.1 硬件设计

SS_SD	PF5/FSMC_A5/ADC5_IN13	18 PF6 KEY3
PF6/TIM10_CH1/FSMC_NIORD/ADC3_IN4	19 PF7 KEY2	
PF7/TIM11_CH1/FSMC_NREG/ADC3_IN5	20 PF8 KEY1	
PF8/TIM13_CH1/FSMC_NIOWR/ADC3_IN6	21 PF9 KEY0	
PF9/TIM14_CH1/FSMC_CD/ADC3_IN7	22 PE10 T_CD_BT	

5向按键



3.4.2 核心代码解析

```

RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE); //使能 SYSCFG 时钟
//配置按键使用的 IO 口为外部中断线
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOF, EXTI_PinSource6); //PF6 连接
到中断线 6
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOF, EXTI_PinSource7); //PF7 连接
到中断线 7
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOF, EXTI_PinSource8); //PF8 连接
到中断线 8
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOF, EXTI_PinSource9); //PF9 连接
到中断线 9
    // 配置外部中断线 EXTI_Line6, 7, 8, 9
    EXTI_InitStructure.EXTI_Line = EXTI_Line6 | EXTI_Line7 | EXTI_Line8 |
EXTI_Line9;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt; //中断事件
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling; //下降沿触发
    EXTI_InitStructure.EXTI_LineCmd = ENABLE; //中断线使能
    EXTI_Init(&EXTI_InitStructure);

```

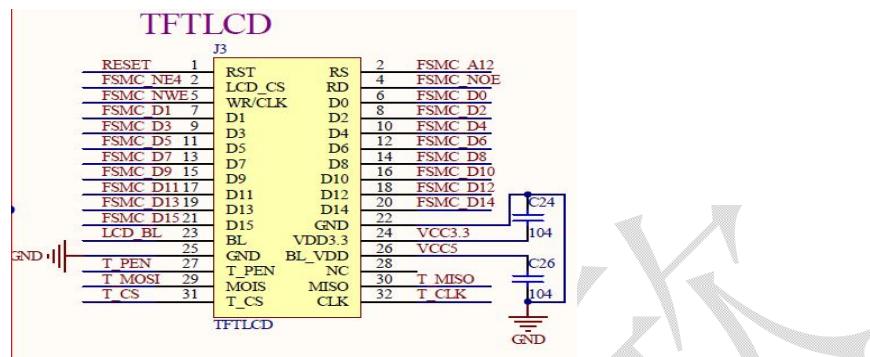
3.4.3 实验操作与现象

由于按键是接地有效，当按键按下时就是 IO 口从高电平变为低电平，下降沿触发中断。

程序下进去之后，按 KEY0，打开蜂鸣器，按 KEY1，LED0 亮，按 KEY2，LED1 亮，按 KEY3，关闭蜂鸣器，熄灭 LED0, LED1。LED2 系统运行提示灯。

第 5 章 TFTLCD 显示

3.5.1 硬件设计



3.5.2 核心代码解析

```
//使用 NOR/SRAM 的 Bank1. sector4, 地址位 HADDR[27, 26]=11
```

```
//A12 作为数据命令区分线
```

```
//注意设置时 STM32 内部会右移一位对其
```

```
#define CMD_BASE ((u32)(0x6C000000 | 0x00001FFE))
#define DATA_BASE ((u32)(0x6C000000 | 0x00002000))
#define LCD_CMD (* (u16 *) CMD_BASE )
#define LCD_DATA (* (u16 *) DATA_BASE)
```

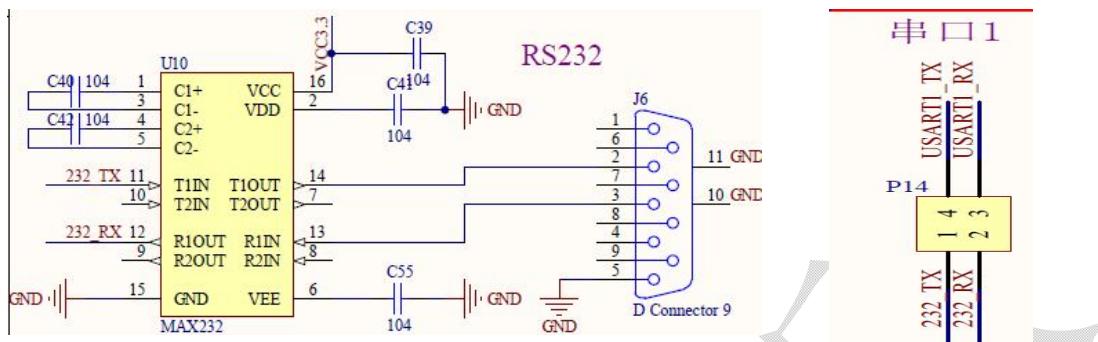
由于用 **FSMC A12** 作为数据命令选择线，所以数据与命令的地址线如上。

3.5.3 实验操作与现象

程序下进去之后，按 KEY0，依次进行各颜色清屏显示，按 KEY1，屏幕进行图形显示，长按 KEY2 进行数值显示，最高位有无填充 0 两种情况。

第 6 章 串口 1-RS232 实验

3.6.1 硬件设计



使用 DB9 针 232 串口时，需要将 P14 的短路帽接起来。由于使用的串口 1，所以需要对串口 1 进行初始化和编写需要的中断函数。

3.6.2 核心代码解析

串口发送函数：

```
void uart1SendChars(u8 *str, u16 strlen)
{
    u16 k= 0 ;
    do { uart1SendChar(*(str + k)); k++; } //循环发送,直到发送完毕
    while (k < strlen);
}
```

串口接收中断，里面定义接收命令的格式：S.....E

```
//串口 1 中断服务程序
void USART1_IRQHandler(void)
{
    u8 rec_data;
    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET) //接收中断
    {
        rec_data =(u8)USART_ReceiveData(USART1); // (USART1->DR) 读取接收到的数据
        if(rec_data=='S') //如果是 S, 表示是命令信息的起始位
        {
            uart_byte_count=0x01;
        }

        else if(rec_data=='E') //如果 E, 表示是命令信息传送的结束位
        {
```

```
if(strcmp("Light_led1", (char*)receive_str)==0)    LED1=0;      //点亮 LED1  
else if(strcmp("Close_led1", (char*)receive_str)==0)  LED1=1;//关灭 LED1  
else if(strcmp("Open_beep", (char *)receive_str)==0)  BEEP=1; //蜂鸣器响  
else if(strcmp("Close_beep", (char *)receive_str)==0) BEEP=0;//蜂鸣器不响  
for(uart_byte_count=0;uart_byte_count<32;uart_byte_count++)receive_str[uart  
_byte_count]=0x00;  
    uart_byte_count=0;  
}  
else if((uart_byte_count>0)&&(uart_byte_count<=USART1_REC_NUM))  
{  
    receive_str[uart_byte_count-1]=rec_data;  
    uart_byte_count++;  
}  
}  
}
```

3.6.3 实验操作与现象

程序下进去之后，接上 232 串口线，电脑端打开串口助手，发送控制命令。

1、串口助手接收开发板串口 1 发来的数据

按 KEY0 后，开发板会发送 UART1 TEST 给串口助手。

2、串口助手发送控制命令给开发板

发送命令格式为：S.....E为你需要控制的内容，具体看程序代码。

如：发送 SLight_led1E 为打开 LED1，发送后，此时 LED1 就会亮
其他命令同理。

第 7 章 定时器中断

3.7.1 硬件设计

STM32 内部集成定时器功能。

3.7.2 核心代码解析

```
void TIM2_Init(u16 auto_data, u16 fractional)
{
.....
}    定时器溢出时间计算方法:Tout=((auto_data+1)*(fractional+1))/Ft (us)
Ft 定时器时钟
TIM2_Init(4999, 8399); //定时器 2 时钟 84M, 分频系数 8400, 84M/8400=10K 所
以计数 5000 次为 500ms
```

3.7.3 实验操作与现象

程序下进去之后，每隔 500ms 蜂鸣器响闭 和 LED1 亮灭。

第 8 章 PWM 输出

3.8.1 硬件设计

利用 STM32 内部集成的定时器实现 PWM 输出。

3.8.2 核心代码解析

```
//初始化 TIM5 Channel1 PWM 模式
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1; //选择定时器模式:TIM
脉冲宽度调制模式 1
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; //比较
```

输出使能

```
TIM_OCIInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low; // 输出极性:TIM 输出比较极性低  
TIM_OC1Init(TIM5, &TIM_OCIInitStructure); //根据 T 指定的参数初始化外设  
TIM1 4OC1  
TIM_OC1PreloadConfig(TIM5, TIM_OCPreload_Enable); //使能 TIM5 在 CCR1 上的预装载寄存器 Ft 定时器时钟
```

TIM5_PWM_Init(499, 83); //84M/84=1Mhz 的计数频率, 重装载值 500, 所以 PWM 频率为 1M/500=2Khz.

TIM_SetCompare1(TIM5, pwmzkb); 占空比为 pwmzkb/500.

3.8.3 实验操作与现象

程序下进去之后, 按 KEY0 占空比减少 10%, 按 KEY1 占空比增加 10%。

第 9 章 独立看门狗

3.9.1 硬件设计

利用 STM32 内部集成的看门狗实现。

3.9.2 核心代码解析

看门狗配置函数, 配置多长时间喂一次狗, 保证系统运行下去不会软件复位

```
// void IWDG_Init(u8 prer, u16 rlr)  
{  
    IWDG_WriteAccessCmd(IWDG_WriteAccess_Enable); //使能对 IWDG->PR IWDG->RLR  
    的写  
    IWDG_SetPrescaler(prer); //设置 IWDG 分频系数  
    IWDG_SetReload(rlr); //设置 IWDG 装载值
```

```
IWDG_ReloadCounter();  
IWDG_Enable(); //使能看门狗  
}  
IWDG_Init(4, 500); //与分频数为 64, 重载值为 500, 溢出时间为 1s
```

3.9.3 实验操作与现象

程序开始有点灯 LED0=0，当看门狗时间溢出，也就是说时间到，看门狗会使系统复位，灯会闪(先灭后亮)

如果 KEY0 按下，则喂狗 如果一直按 KEY0 一直喂狗(一直重载赋值，时间就无法到达 系统就不会复位)，灯不会闪。

第 10 章 模数转换 ADC

3.10.1 硬件设计

STM32 内部集成 ADC 模块。

3.10.2 核心代码解析

ch：通道值 0~16 取值范围为：ADC_Channel1_0~ADC_Channel1_16

```
u16 Get_Adc(u8 ch)  
{  
    //设置指定 ADC 的规则组通道，一个序列，采样时间  
    ADC-RegularChannelConfig(ADC1, ch, 1, ADC_SampleTime_480Cycles );  
    //ADC1, ADC 通道, 480 个周期, 提高采样时间可以提高精确度  
    ADC_SoftwareStartConv(ADC1); //使能指定的 ADC1 的软件转换启动功能  
    while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC )); //等待转换结束  
    return ADC_GetConversionValue(ADC1); //返回最近一次 ADC1 规则组的转换结果  
}
```

该函数返回测的通道的模拟电压值。

3.10.3 实验操作与现象

程序下进去之后，PA6(在 OV2640 那边，可以看丝印)接入所测量的模拟电压，屏幕会显示接入的模拟电压的电压值，会随着输入的电压的改变而实时显示电压化。

第 11 章 数模转换 DAC

3.11.1 硬件设计

STM32 内部集成 DAC 模块。

3.11.2 核心代码解析

```
/**************************************************************************  
* 名    称: void Dac1_Set_Vol(u16 vol)  
* 功    能: 设置通道 1 输出电压  
* 入口参数: vol:0~3300, 代表 0~3.3V  
* 返回参数: 无  
*****  
void Dac1_Set_Vol(u16 vol)  
{  
    double temp=vol;  
    temp/=1000;  
    temp=temp*4096/3.3; //12 位的缘故为 4096  
    DAC_SetChannel1Data(DAC_Align_12b_R, temp); //12 位右对齐数据格式设  
置 DAC 值  
}
```

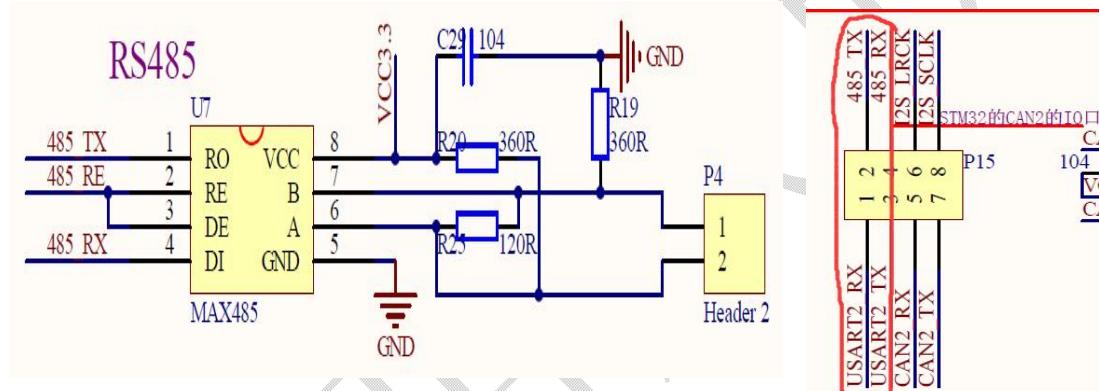
3.11.3 实验操作与现象

该例程为 PA4 作为 DAC 输出电压，PA6 作为 ADC 模拟电压输入，实验时需要用一根杜邦线连接 PA4 与 PA6(两个 IO 口都在 OV2640 接

口上), 此时 DAC(PA4)输出的电压值输入到 PA6, 屏幕也会实时显示输出的电压值的数字值、模拟值大小和测得的 ADC 输入电压的模拟值。按 KEY0 DAC 输出电压增加, 按 KEY1 DAC 输出电压减少。

第 12 章 串口 2-485

3.12.1 硬件设计



由于 485 接于串口 2 并且中间用跳针隔开, 所以在使用串口 2 的 485 时, 请将 P15 的短路帽接上。

3.12.2 核心代码解析

485 发送函数

```
void RS485_Send_Data(u8 *buf, u8 len)
{
    u8 t;
    RS485_TX_EN=1;           //设置为发送模式
    for(t=0;t<len;t++)       //循环发送数据
    {
        while(USART_GetFlagStatus(USART2, USART_FLAG_TC)==RESET); //等待
        USART_SendData(USART2, buf[t]); //发送数据
    }
    while(USART_GetFlagStatus(USART2, USART_FLAG_TC)==RESET); //等待
```

发送结束

```
uart_byte_count=0;  
RS485_TX_EN=0; //发送完设置为接收模式  
}
```

RS485_TX_EN 一般设置为“0”，处于**接收状态**，当什么时候需要发送的才设置为发送，发送完又设置成接收，使 485 一直处于接收状态，不会遗漏接收信息。

RS485 查询接收到的数据

```
void RS485_Receive_Data(u8 *buf, u8 *len)  
{  
    u8 rxlen=uart_byte_count;  
    u8 i=0;  
    *len=0; //默认为 0  
    delay_ms(10); //等待 10ms, 连续超过 10ms 没有接收到一个数据, 则认为  
    接收结束  
  
    if(rxlen==uart_byte_count&&rxlen) //接收到了数据, 且接收完成了  
    {  
        for(i=0;i<rxlen; i++)  
        {  
            buf[i]=RS485_receive_str[i];  
        }  
        *len=uart_byte_count; //记录本次数据长度  
        uart_byte_count=0; //清零  
    }  
}
```

接收中断触发后，在中断服务函数中，就会把接收的数值存在 RS485_receive_str[] 数组。

3.12.3 实验操作与现象

P4 接口为 485 对外的 A B 接口，请将外部 485 设备，按 A 接 A，B 接 B 的形式接起来。程序下进去就可以跟板子进行 485 通信。

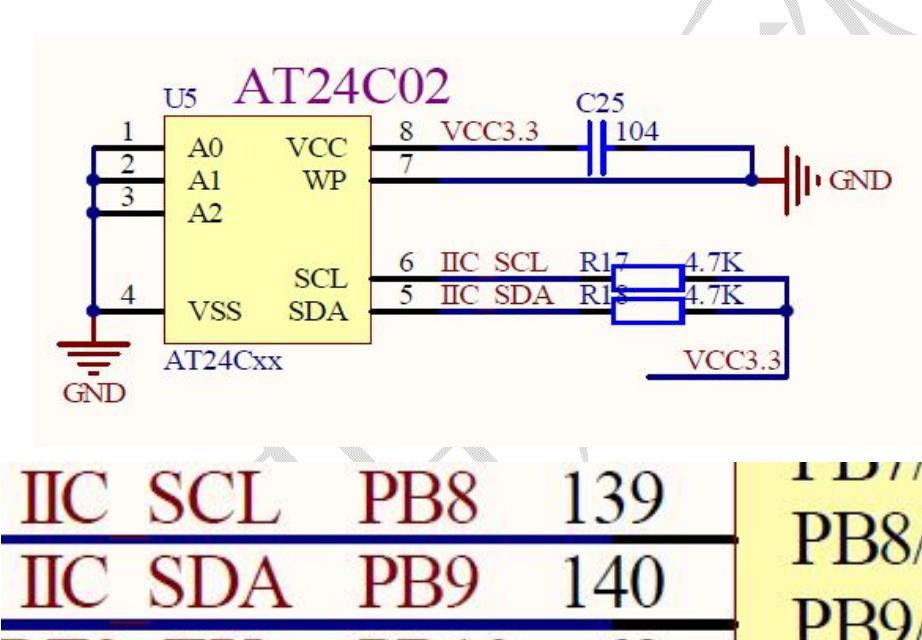
发什么控制命令格式跟第五章一样，请参考第五章的使用方法。

第 13 章 各串口 TTL 实验

本章实验跟第6章的使用操作一模一样，只是程序程序上需要对应串口号即可。

第 14 章 IIC_24C02

3.14.1 硬件设计



PB8 与 PB9 为原生的硬件 IIC。由于 STM32 原生的 IIC 不稳定，所以本例程程序使用普通 IO 口进行模拟 IIC 通信协议。

3.14.2 核心代码解析

```
//IIC_SDA 线 IO 方向配置
#define SDA_IN() {GPIOB->MODER&= ~(3<<18);GPIOB->MODER|=0<<18;} //PB9 输入模式
#define SDA_OUT() {GPIOB->MODER&= ~(3<<18);GPIOB->MODER|=1<<18;} //PB9 输出模式
```

AT24C02 读函数：

```
*****
* 名    称: void AT24C02_Read(u8 ReadAddr, u8 *pBuffer, u8 ReadNum)
* 功    能: 从 AT24C02 里面的指定地址开始读出指定个数的数据
```

* 入口参数: ReadAddr :开始读出的地址 0~255
pBuffer :数据数组首地址
ReadNum:要读出数据的个数

* 返回参数: 无

```
*****  
void AT24C02_Read(u8 ReadAddr, u8 *pBuffer, u8 ReadNum)
```

```
{
```

```
    while (ReadNum--)
```

```
{
```

```
        *pBuffer++=AT24C02_ReadByte (ReadAddr++);
```

```
}
```

```
}
```

AT24C02 写函数:

```
*****
```

* 名 称: void AT24C02_Write(u8 WriteAddr, u8 *pBuffer, u8 WriteNum)

* 功 能: 从 AT24C02 里面的指定地址开始写入指定个数的数据

* 入口参数: WriteAddr :开始写入的地址 0~255

pBuffer :数据数组首地址

WriteNum:要写入数据的个数

* 返回参数:

```
*****
```

```
void AT24C02_Write(u8 WriteAddr, u8 *pBuffer, u8 WriteNum)
```

```
{
```

```
    while (WriteNum--)
```

```
{
```

```
        AT24C02_WriteByte (WriteAddr,*pBuffer);
```

```
        WriteAddr++;
```

```
        pBuffer++;
```

```
}
```

3.14.3 实验操作与现象

该例程需要配合串口 1 通信来完成。

1、接收串口 1 发来的数据:

程序开始运行后，屏幕会显示请用户发送数据到串口1

Please send data to usart1, 此时用户就要通过串口助手向开发板发送数

据，发送格式为前面串口例程那样S.....E。“.....”为用户所需要发送的数据，发送后数据存在 receive_str[]数值中并在显示屏显示。

2、将接收到的数据写入AT24C02：

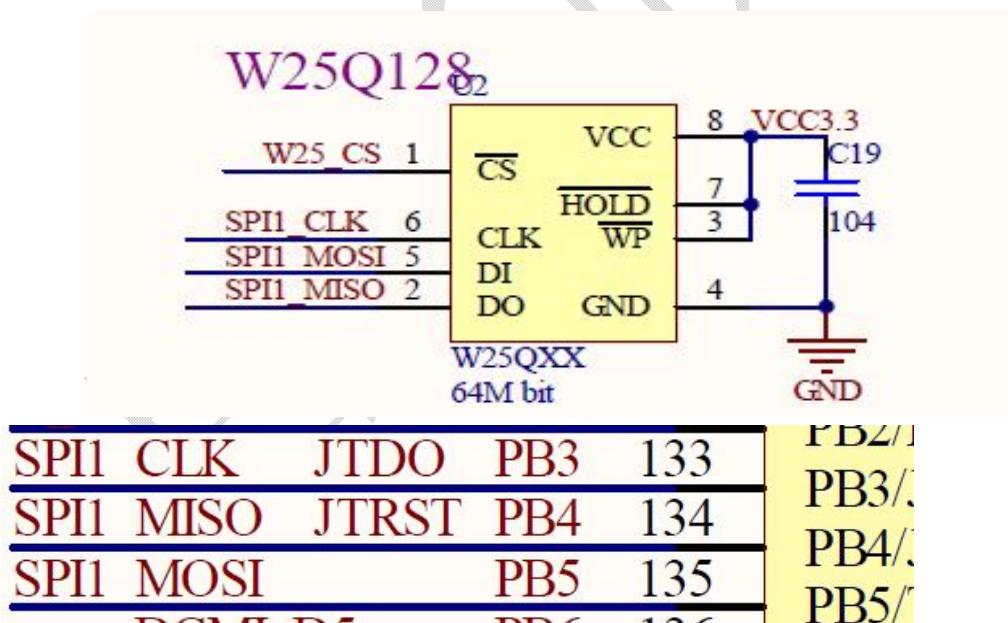
KEY0按下，将串口1接收到的数据(receive_str[])中数据)写入24C02

3、将写入到24C02的数据读出并显示

KEY1按下，将写入到24C02中的数据读出并显示

第 15 章 SPI_W25Qxx

3.15.1 硬件设计



W25Q128 接于 STM32 原生硬件 SPI1 处。

3.15.2 核心代码解析

```
//读写函数
```

3.15.3 实验操作与现象

该例程需要配合串口 1 通信来完成。

1、接收串口 1 发来的数据：

程序开始运行后，屏幕会显示请用户发送数据到串口1

Please send data to usart1, 此时用户就要通过串口助手向开发板发送数据，发送格式为前面串口例程那样S.....E。“.....”为用户所需要发送的数据，发送后数据存在 receive_str[] 数值中并在显示屏显示。

2、将接收到的数据写入W25Q128：

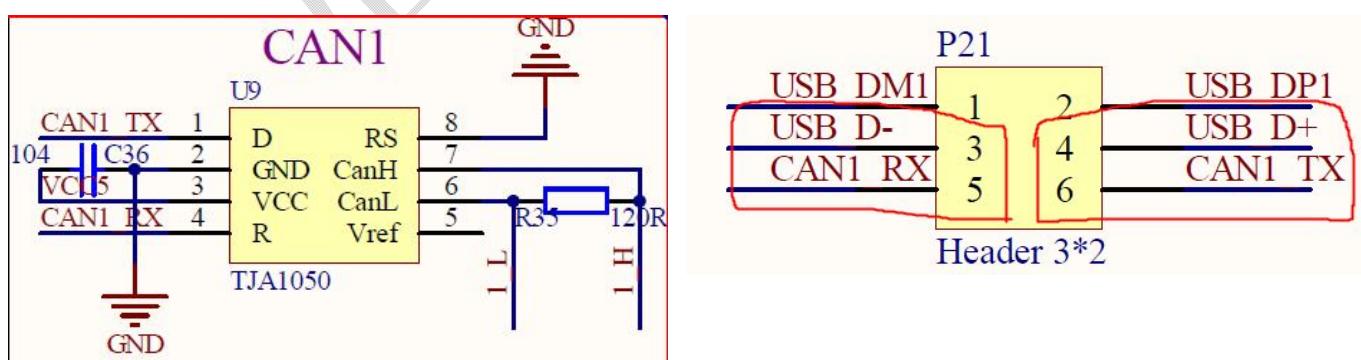
KEY0按下，将串口1接收到的数据 (receive_str[]) 中数据) 写入25Q128

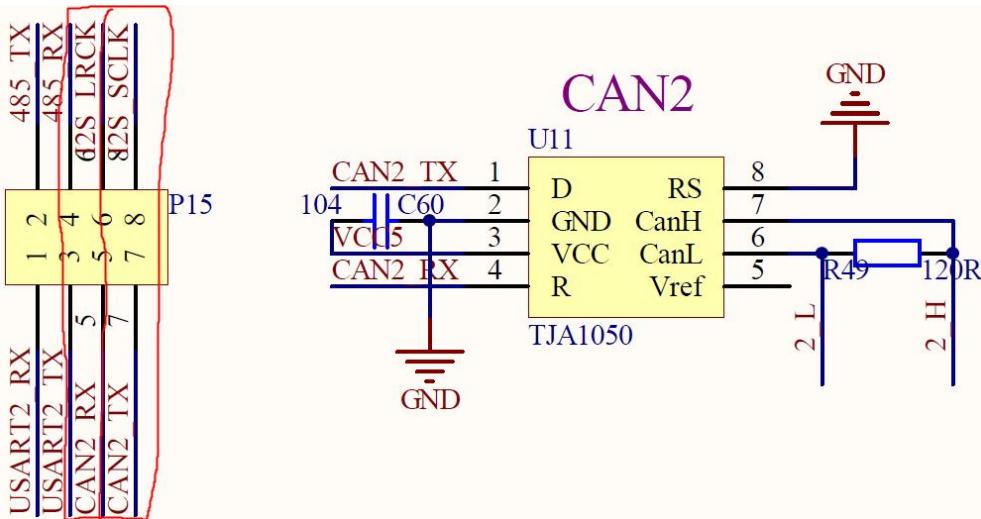
3、将写入到W25Q128的数据读出并显示：

KEY1按下，将写入到W25Q128中的数据读出并显示

第 16 章 CAN1 与 CAN2 通信

3.16.1 硬件设计





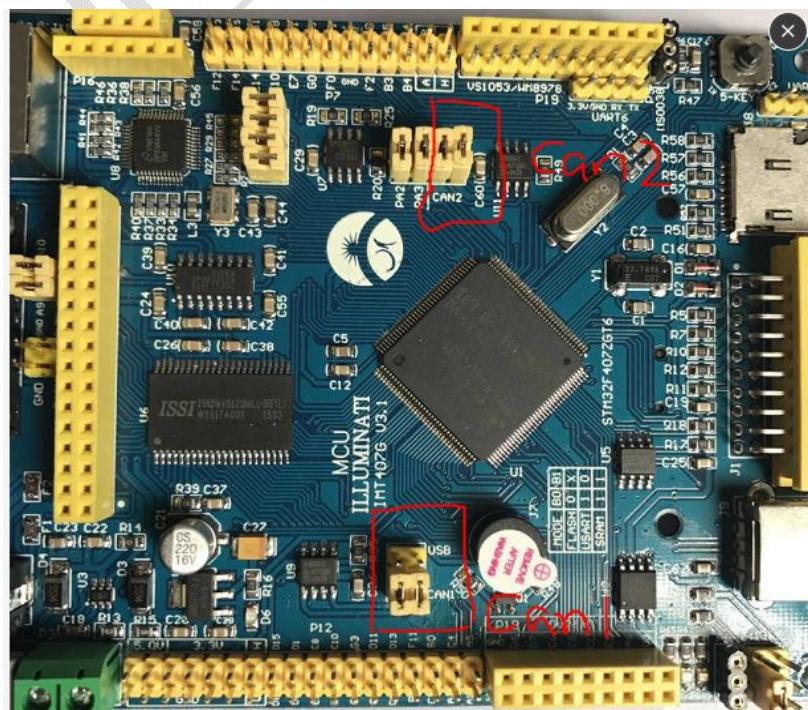
CAN1 和 CAN2 时，记得短路帽都接于 CAN 端。

3.16.2 核心代码解析

```
//CAN 模式选择
u8 CAN1_mode=1; //CAN 工作模式;0,普通模式;1,环回模式
u8 CAN2_mode=1; //CAN 工作模式;0,普通模式;1,环回模式
CAN1_Mode_Init(CAN_SJW_1tq,CAN_BS2_6tq,CAN_BS1_7tq,6,CAN_Mode_Normal); //CAN 初始化回环模式,波特率 500Kbps
CAN2_Mode_Init(CAN_SJW_1tq,CAN_BS2_6tq,CAN_BS1_7tq,6,CAN_Mode_Normal); //CAN 初始化回环模式,波特率 500Kbps
```

3.16.3 实验操作与现象

确保短路帽都接到两个 CAN 端。如下图接法：



1、CAN 模式设置：

CAN的工作模式有回环模式和普通模式。

回环模式只要一个can自己就能工作，自发自收。

普通模式，需要两个CAN。这时需要连接两个CAN的H和L接口。

程序初始化时工作在普通模式。

2、发送与接收数据：

程序下载运行后，两个can会相互给对方发数据，收到数据后都会显示在屏幕上，并通过串口1发出。

第 17 章 DMA 实验

3.17.1 硬件设计

本章使用串口 1 来实验 DMA 发送，硬件跟第 6 章一样。

3.17.2 核心代码解析

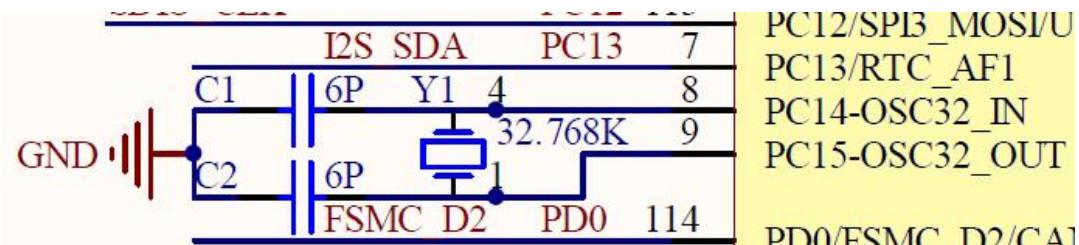
```
//配置串口 1DMA 输出  
USER_DMA_Config(DMA2_Stream7, DMA_Channel1_4, (u32)&USART1->DR, (u32)SEND  
_TestDate, sizeof(SEND_TestDate));  
//DMA2, STEAM7, CH4, 外 设 为 串 口 1, 存 储 器 为 SendBuff, 长 度  
为:sizeof(SEND_TestDate). }
```

3.17.3 实验操作与现象

程序下载运行后，KEY0按下 DMA发送。在CPU没有调用串口发送函数 DMA自己把从串口数据发出。

第 18 章 RTC 实时时钟

3.18.1 硬件设计



STM32F4 的实时时钟 (RTC) 相对于 STM32F1 来说，改进了不少，带了日历功能了，STM32F4 的 RTC，是一个独立的 BCD 定时器/计数器。RTC 提供一个日历时钟（包含年月日时分秒信息）、两个可编程闹钟（ALARM A 和 ALARM B）中断，以及一个具有中断功能的周期性可编程唤醒标志。RTC 还包含用于管理低功耗模式的自动唤醒单元。

3.18.2 核心代码解析

F407 的 RTC 就是一个时钟芯片，代码操作也是操作对应的寄存器，比如年月日时分秒寄存器。所以代码看起来简单。请用户看下例程代码。

3.18.3 实验操作与现象

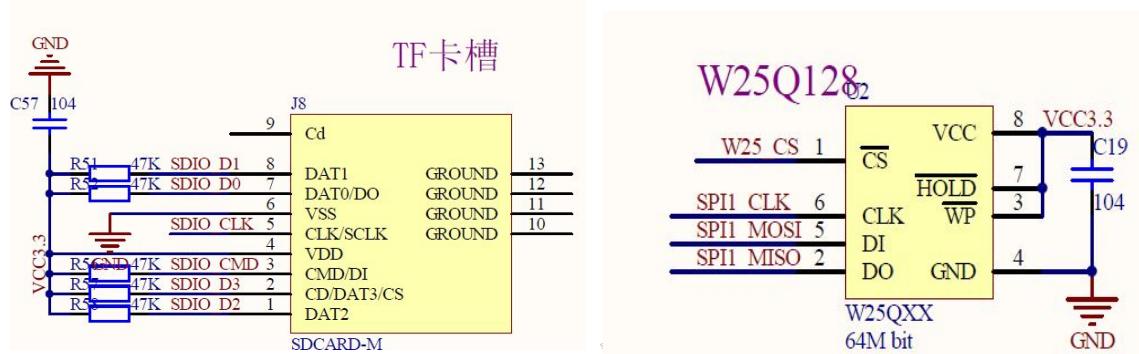
程序下载进去后，RTC时钟初始化后开始跑动，并在液晶屏上每秒更新显示。该时钟例程有调整设置时间功能。

- 1、长按KEY3 四秒进入调整时间
- 2、按KEY2右移选择你要调整的时间选项(年月日时分秒)
- 3、按KEY0数值加1，按KEY1数值减1
- 4、调整完时间，短按KEY3设置时间

第 19 章 汉字显示

3.19.1 硬件设计

硬件使用到 SD 卡和 FLASH W25Q128。



将 SD 卡中存放的字库，更新到 W25Q128 中，然后液晶屏显示汉字从 W25Q128 取字库。

3.19.2 核心代码解析

由于使用到 SD 卡，所以用文件系统操作比较方便。使用到文件系统需要先为其分别内存，用内存管理来分配。

```
*****HanZiUse.lib 调用 更新字库要用到的*****
Memory_Init(INSRAM); //初始化内部内存池
fsTF=(FATFS*)Mem_malloc(INSRAM,sizeof(FATFS)); //为文件系统分配
内存
f_mount(fsTF,"0:",1); //挂载 TF 卡
*****
```

3.19.3 实验操作与现象

程序下进去后，检测W25Q128里面有没字库，如没有字库，更新字库。按KEY0可以更新字库(从SD卡取字库更新到W25Q128)，按KEY1进行汉字显示测试。

第 20 章 RTC 农历显示

3.20.1 硬件设计

该例程为 RTC 和汉字显示结合运用。

3.20.2 核心代码解析

只要输入公历的日期就可以得到农历日期的字符串

```
/*
* 函数名称:void GetLunarCalendarStr(u16 year,u8 month,u8 day,u8 *str)
* 功能描述:输入公历日期得到农历字符串
* 入口参数: year      公历年
*           month     公历月
*           day       公历日
*           str        接收农历日期字符串地址
* 输出参数: 无
* 说明:      GetLunarCalendarStr(2015,03,15,str) 返回*str="乙未年正月廿五"
*/
void GetLunarCalendarStr(u16 year,u8 month,u8 day,u8 *str)
{
    u8 NLyear[4];
    u8 SEyear;
    StrCopyss(&str[0],(u8 *)"甲子年正月初一",15);
    if(GetChinaCalendar(year,month,day,(u8 *)NLyear)==0) return;
    GetSkyEarth(NLyear[0]*100+NLyear[1],&SEyear);
    StrCopyss(&str[0],(u8 *) sky[SEyear%10],2); // 甲
    StrCopyss(&str[2],(u8 *)earth[SEyear%12],2); // 子
    if(NLyear[2]==1) StrCopyss(&str[6],(u8 *)"正",2);
    else             StrCopyss(&str[6],(u8 *)monthcode[NLyear[2]-1],2);

    if(NLyear[3]>10) StrCopyss(&str[10],(u8 *)nongliday[NLyear[3]/10],2);
    else               StrCopyss(&str[10],(u8 *)"初",2);
    StrCopyss(&str[12],(u8 *)monthcode[(NLyear[3]-1)%10],2);
}
```

3.20.3 实验操作与现象

程序下进去后，检测W25Q128里面有没有字库，如没有字库，更新字库。接着就显示时间和公历农历的日期，还有节气。如需调整时间，请参照第18章的RTC实验，操作与其一模一样。

第 21 章 内部温度传感器实验

3.21.1 硬件设计

STM32F407 的内部温度传感器在内部和 ADC1_IN16 输入通道相连，此通道把传感器输出的电压转换成数字值。所以我们操作就跟前面说的操作 ADC 是一样的。

3.21.2 核心代码解析

```
//取 ADC 值
u16 Get_Adc_Average(u8 ch, u8 times)
{
    u32 temp_val=0;
    u8 t;
    for(t=0; t<times; t++)
    {
        temp_val+=Get_Adc(ch); //获取通道转换值
        delay_ms(5);
    }
    return temp_val/times;
}

//把 ADC 值转化成电压值并扩大 100 倍便于液晶屏显示
short Get_Temprate(void)
{
    u32 adcx;
    short result;
    double temperate;
    adcx=Get_Adc_Average(ADC_Channel1_16, 10); //读取通道 16 内部温度传感器通
```

道, 10 次取平均

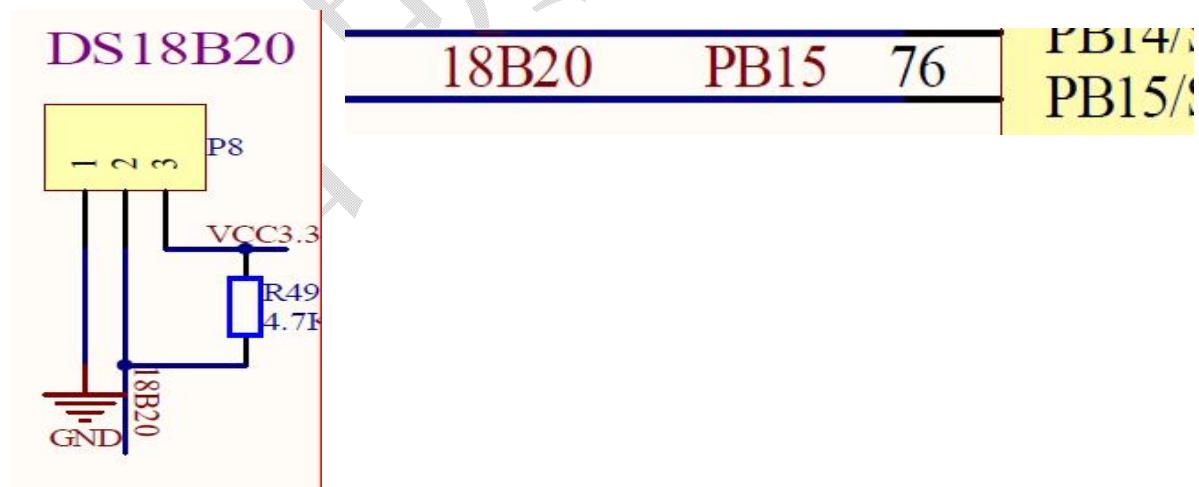
```
temperate=(float)adcx*(3.3/4096);           //电压值  
temperate=(temperate-0.76)/0.0025 + 25; //转换为温度值  
result=temperate*=100;                      //扩大 100 倍.  
return result;  
}
```

3.21.3 实验操作与现象

程序下载运行后, 屏幕显示温度值。内部温度传感器主要测量的是STM32芯片的温度, 所以在使用的时候, 会因为板子工作功耗, 而温度值变化。

第 22 章 DS18B20 温度传感器实验

3.22.1 硬件设计



3.22.2 核心代码解析

获取温度值

```
short DS18B20_Get_Temp(void)
```

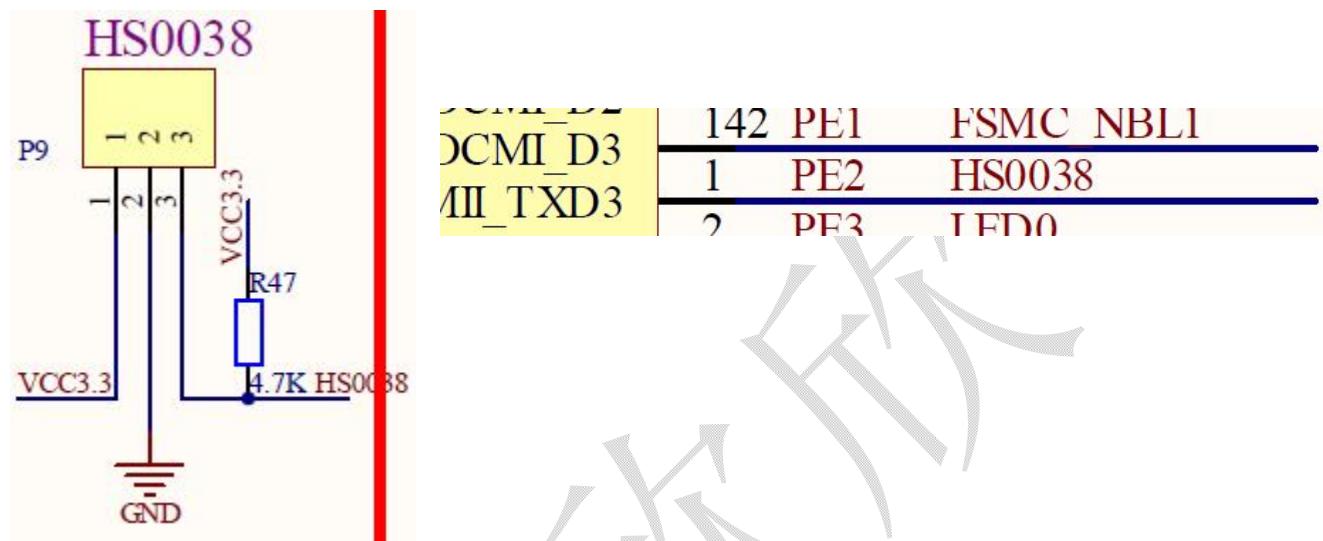
```
{  
    u8 temp;  
    u8 TL,TH;  
    short tem;  
    DS18B20_Start(); // ds1820 start convert  
    DS18B20_Rst();  
    DS18B20_Check();  
    DS18B20_Write_Byte(0xcc); // skip rom  
    DS18B20_Write_Byte(0xbe); // convert  
    TL=DS18B20_Read_Byte(); // LSB  
    TH=DS18B20_Read_Byte(); // MSB  
    if(TH>7)  
    {  
        TH=~TH;  
        TL=~TL;  
        temp=0; // 温度为负  
    }else temp=1; // 温度为正  
    tem=TH; // 获得高八位  
    tem<<=8;  
    tem+=TL; // 获得底八位  
    tem=(double)tem*0.625; // 转换 此时的转换已经把温度扩大 10 倍  
    if(temp) return tem; // 返回温度值  
    else return -tem;  
}
```

3.22.3 实验操作与现象

程序下进去后，每200ms读取温度值，并且在显示屏显示。LED0 闪烁显示系统正在运行。

第 23 章 红外传感器 HS0038

3.23.1 硬件设计



3.23.2 核心代码解析

说明：Ir_Record[4]; 处理后的红外码，分别是 客户码，客户码，数据码，数据码反码

```
*****
void Ir_Decode(void)
{
    u8 i, j, k;
    u8 Record,Value;
    IR_OK = 1;
    k = 1;

    for(i = 0;i < 4;i++)          //处理 4 个字节
    {
        delay_ms(1);
        for(j = 1;j <= 8;j++)    //处理 1 个字节 8 位
        {
            Record = Ir_TimeData[k];
            if(Record >7)      //大于某值为 1, 这个和晶振有绝对关系, 这里使用
12M 计算, 此值可以有一定误差
        {
    
```

```

        Value = Value | 0x80;
    }
    else
    {
        Value = Value;
    }
    if(j < 8)
    {
        Value = Value >>1;
    }
    k++;
}
Ir_Record[i] = Value;
Value = 0;
}
Decode_OK = 1;//处理完毕标志位置 1
}

```

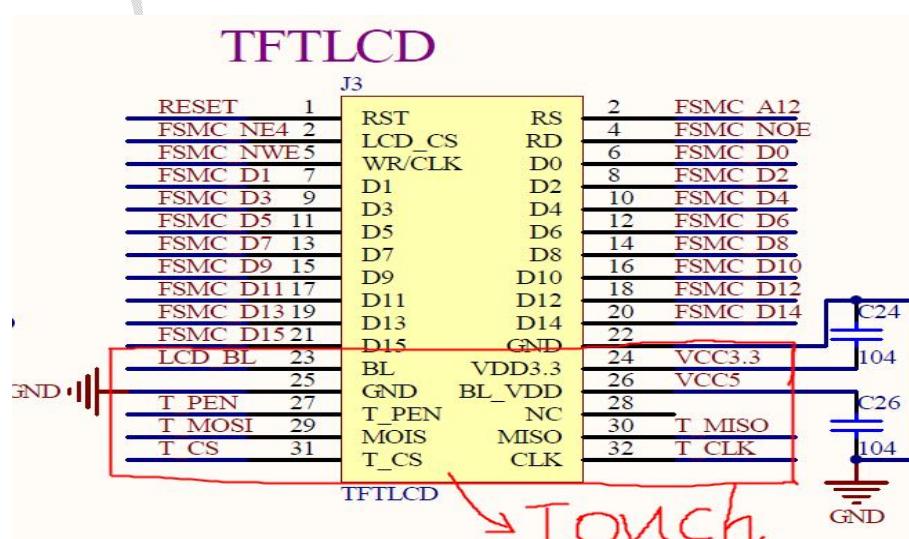
电路硬件设计红外的接口接于普通 IO 口，所以使用外部中断来检测红外数据信号。

3.23.3 实验操作与现象

程序下进去后，按下红外遥控，屏幕会显示相应的键值。

第 24 章 触摸屏使用

3.24.1 硬件设计



3.24.2 核心代码解析

触摸按键扫描， type:0,屏幕坐标;1,物理坐标(校准等特殊场合用)

```
void RTouch_Scan(u8 type)
{
    Xup=0xffff;
    Yup=0xffff;
    if(PEN==0)//有按键按下
    {
        if(type)RTouch_Read_XY2(&x,&y);//读取物理坐标
        else if(RTouch_Read_XY2(&x,&y))//读取屏幕坐标
        {
            x=xFactor*x+xOffset;      //将结果转换为屏幕坐标
            y=yFactor*y+yOffset;
        }
        Xdown=x;
        Ydown=y;
        time++;
    }else //键抬起
    {
        if(time>2)
        {
            Xup=x;
            Yup=y;
        }
        time=0;
        Xdown=0xffff;
        Ydown=0xffff;
    }
}
```

注意参数： **Xup Xdown Yup Ydown**， **up** 为触摸按下后手抬起后返回的按键值， **down** 为触摸按下就返回的按键值。

3.24.3 实验操作与现象

程序下进去后，会检测触摸屏是否校准过，如没校准执行校准程序。校准后，进入写字板功能，可在屏幕书写任意笔画。

第 25 章 232_485_can 数据转换通信

3.25.1 硬件设计

使用 232、485 和 CAN 相关电路。

3.25.2 核心代码解析

数据转换函数

```
void Change_232TO485(u8 numcnt)
{
    RS485_Send_Data(receive_str_232,numcnt);
}

void Change_485TO232(u8 numcnt)
{
    uart1SendChars(receive_str_485,numcnt);
}

void Change_232TOcan1(u8 numcnt)
{
    CAN1_Send_Msg(receive_str_232,numcnt);
}

void Change_can1TO232(u8 numcnt)
{
    uart1SendChars(receive_str_can1,numcnt);
}

void Change_485TOcan1(u8 numcnt)
{
    CAN1_Send_Msg(receive_str_485,numcnt);
}

void Change_can1TO485(u8 numcnt)
{
    RS485_Send_Data(receive_str_can1,numcnt);
}
```

由于 232 和 485 采用的是中断模式，故在中断接收函数中处理数

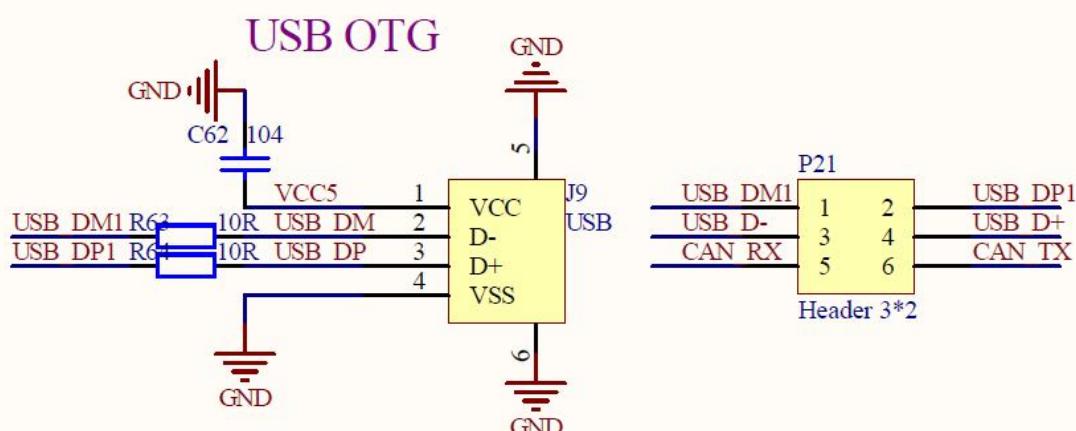
据转换当接收到数据时，原样从 485 或 232 和 can1 发出去，而 can1 则采用扫描方式在主函数中每隔 10ms 检测接收函数。

3.25.3 实验操作与现象

程序下进去后，当串口助手发送数据给232，232的串口1接收到数据后会从485和can1原样发出去刚才从232那里接收到的数据。485和can1同理。

第 26 章 USB U 盘 (Host)

3.26.1 硬件设计



由于 USB 与 CAN 公用 IO 口所以在使用 USB 的时候需要将短路帽连接在 USB 端。就是 P21 的 USB_DM1 接 USB_D-, USB_DP1 接 USB_D+。

3.26.2 核心代码解析

读 U 盘函数

- * 名 称: u8 USBH_UDISK_Read(u8* buf,u32 sector,u32 cnt)
- * 功 能: 读 U 盘
- * 入口参数: buf:读数据缓存区
sector:扇区地址
cnt:扇区个数
- * 返回参数: 错误状态;0,正常;其他,错误代码;

```
u8 USBH_UDISK_Read(u8* buf,u32 sector,u32 cnt)
{
    u8 res=1;
    if(HCD_IsDeviceConnected(&USB_OTG_Core)&&AppState==USH_USR_FS_TEST)/
    /连接还存在,且是 APP 测试状态
    {
        do
        {
            res=USBH_MSC_Read10(&USB_OTG_Core,buf,sector,512*cnt);
            USBH_MSC_HandleBOTXfer(&USB_OTG_Core,&USB_Host);
            if(!HCD_IsDeviceConnected(&USB_OTG_Core))
            {
                res=1;//读写错误
                break;
            };
        }while(res==USBH_MSC_BUSY);
    }else res=1;
    if(res==USBH_MSC_OK)res=0;
    return res;
}
```

写 U 盘函数

```
* 名 称: u8 USBH_UDISK_Write(u8* buf,u32 sector,u32 cnt)
* 功 能: 写 U 盘
* 入口参数: buf:写数据缓存区
             sector:扇区地址
             cnt:扇区个数
* 返回参数: 错误状态;0,正常;其他,错误代码;
u8 USBH_UDISK_Write(u8* buf,u32 sector,u32 cnt)
{
    u8 res=1;
    if(HCD_IsDeviceConnected(&USB_OTG_Core)&&AppState==USH_USR_FS_TEST)/
    /连接还存在,且是 APP 测试状态
    {
        do
        {
            res=USBH_MSC_Write10(&USB_OTG_Core,buf,sector,512*cnt);
            USBH_MSC_HandleBOTXfer(&USB_OTG_Core,&USB_Host);
            if(!HCD_IsDeviceConnected(&USB_OTG_Core))
            {
                res=1;//读写错误
                break;
            };
        }while(res==USBH_MSC_BUSY);
```

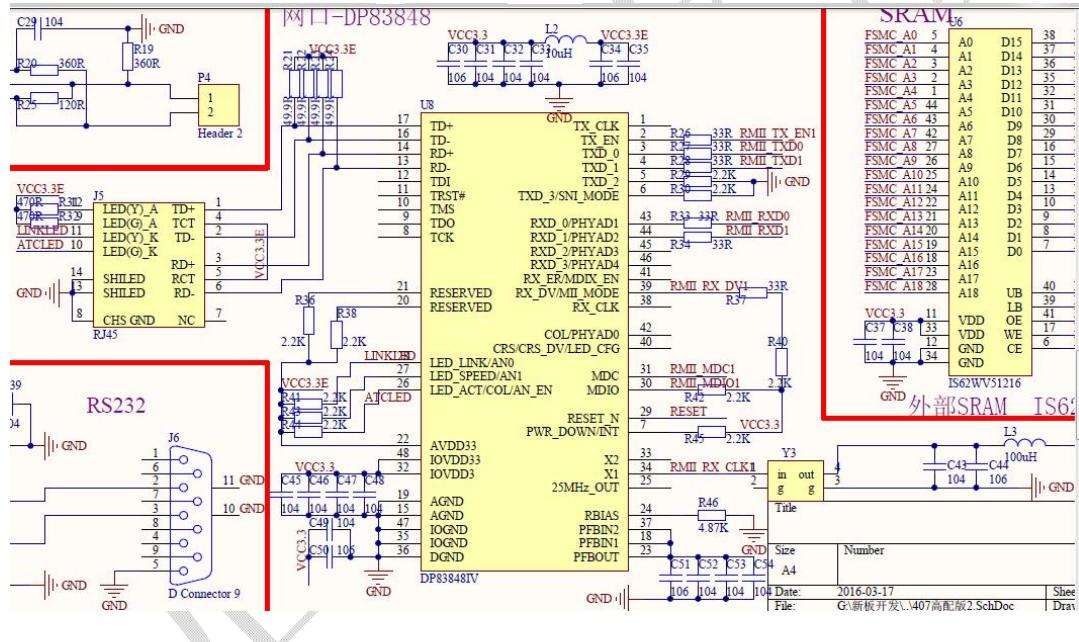
```
    }else res=1;  
    if(res==USBH_MSC_OK)res=0;  
    return res;  
}
```

3.26.3 实验操作与现象

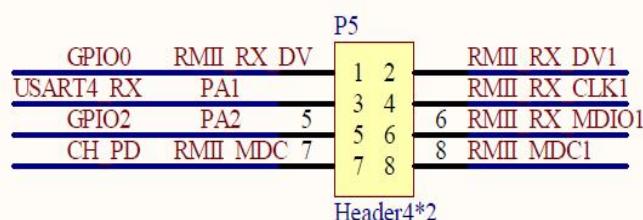
程序下进去后，屏幕会显示U盘的总容量和剩余容量。

第 27 章 TCP 服务器数据收发实验

3.27.1 硬件设计



网口与wifi公用IO口选择



由于网络与 WIFI 接口部分 IO 口公用，所以在使用网络需要把 P5 的短路帽 **全部接上**。

注意：网线要接入开发板后，LWIP 才能初始化成功。

3.27.2 核心代码解析

请仔细看程序代码。

3.27.3 实验操作与现象

两种接网线方法：

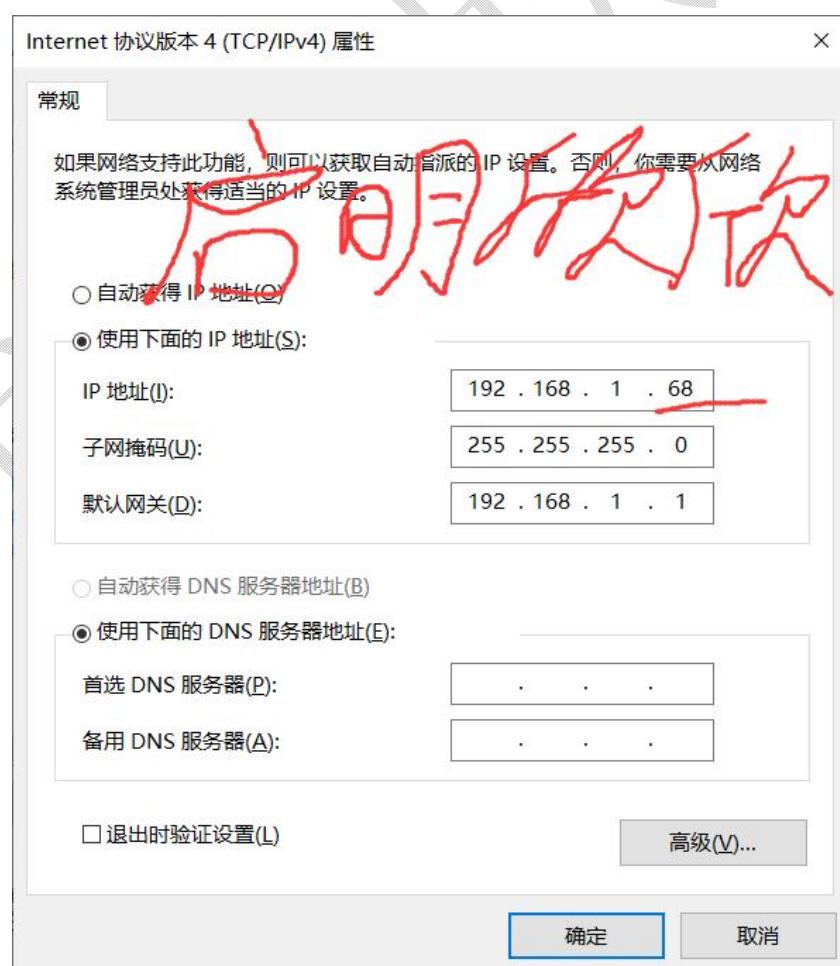
(1)直接通过网线连接开发板和电脑

(2)开发板和电脑都连接路由器

采用第(1)方法：

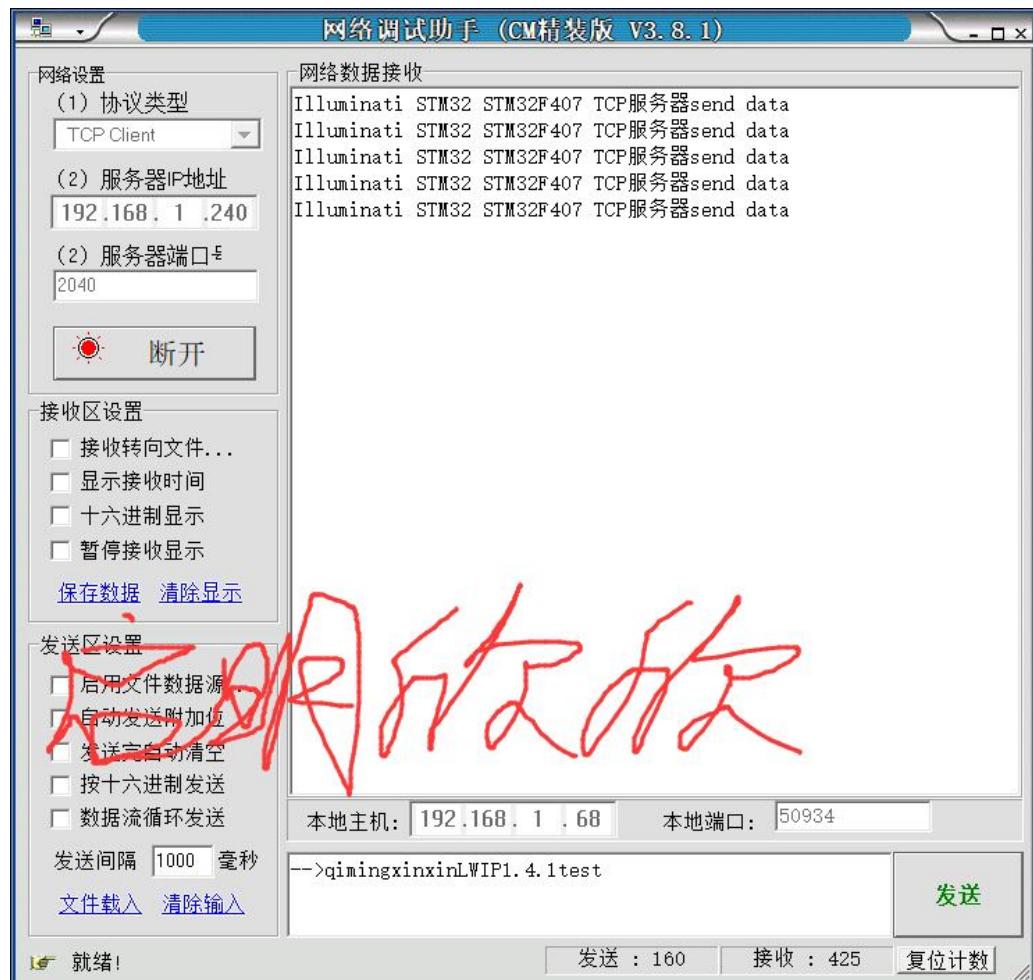
由于电脑没有DHCP分配IP的功能，所以开发板在自动获取IP超时后，会使用程序设置的静态IP 192.168.1.240，作为开发板的IP地址。

1、设置电脑本地IP地址



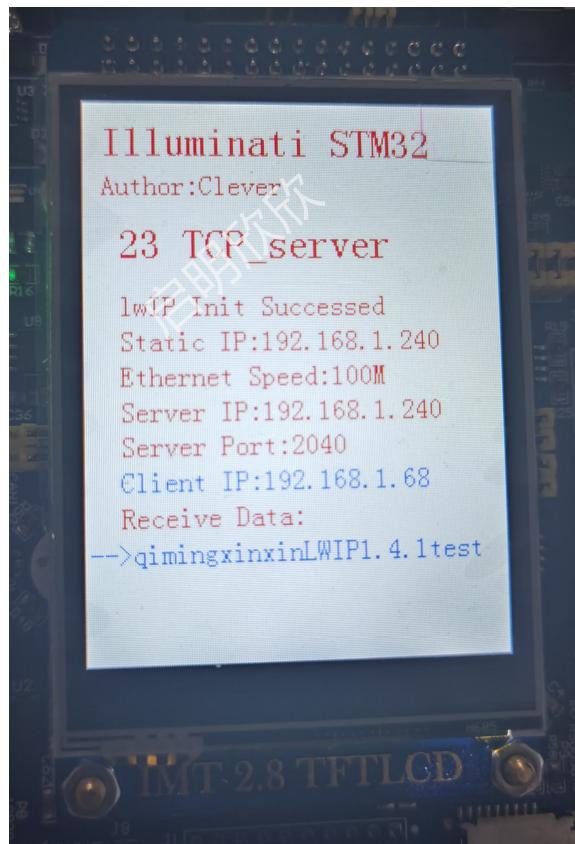
由于是开发板作为服务器，电脑网络助手做客户端，所以网络调试助手选择用TCP Client，要连接到的开发板作为的服务器，网络助手的服务器IP地址设置为192.168.1.240，端口号跟程序设置的一样：2040。

2、打开网络助手设置



3、实验现象

网络助手点击连接后，屏幕显示客户端IP地址（电脑网络助手作为客户端），就是前面设备的电脑本地IP地址192.168.1.68。



前面都设置好后，网络助手点击连接，网络助手就连接上开发板的服务器。之后网络助手点击发送。此时就是客户端向服务器发送数据，服务器（开发板）接到数据后，将接收到的数据显示到屏幕上，按KEY0，开发板把程序里的测试数据发给网络助手。

采用第(2)方法：

开发板和电脑都连接路由器，路由器会为电脑和开发板分配IP地址。

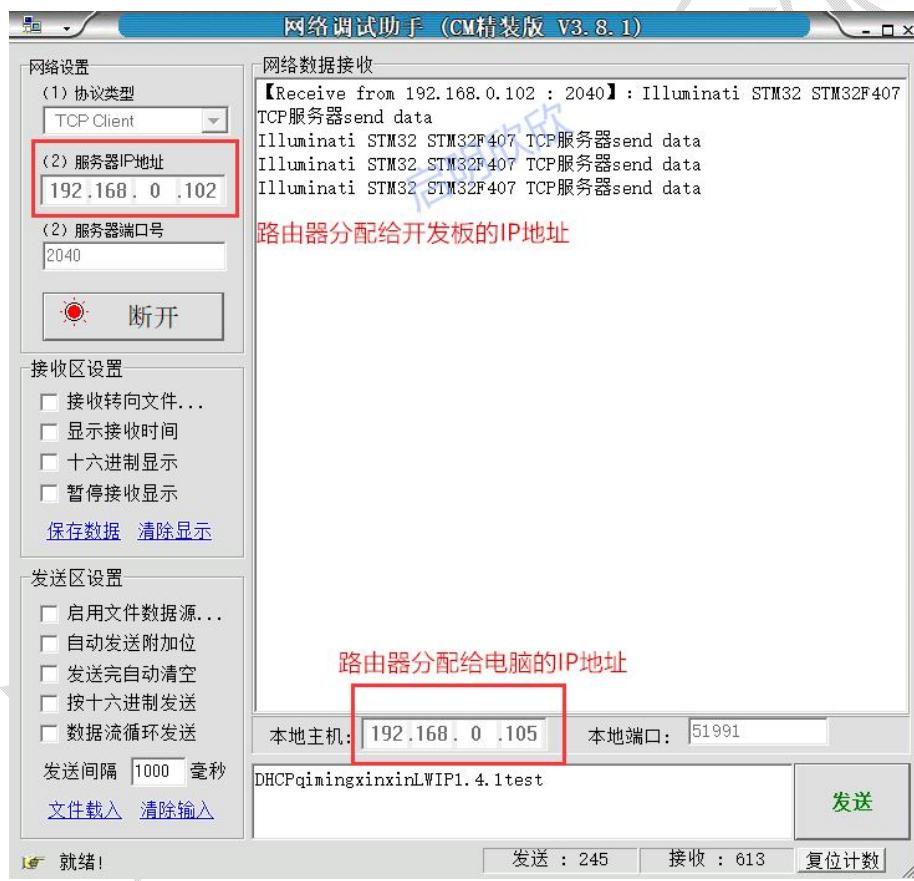
1、电脑本地IP设置为自动获取IP



2、打开网络助手设置

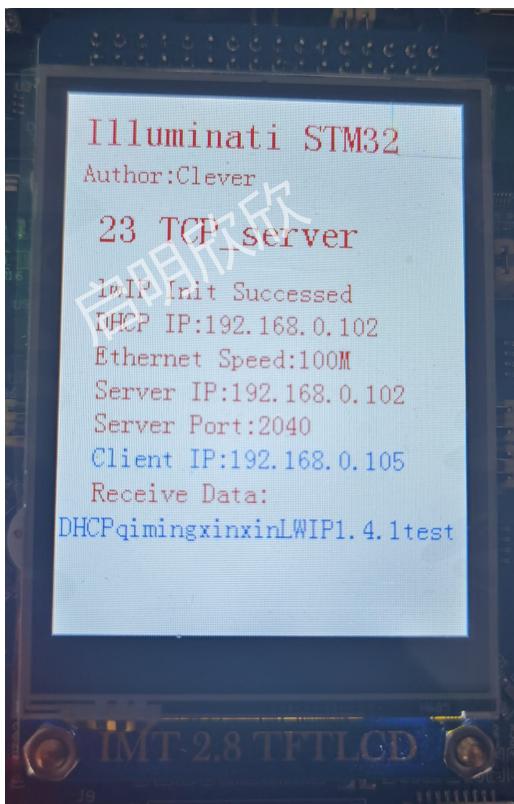
程序运行后，开发板进行DHCP，路由器为开发板分配到IP：192.168.0.102（不同路由器分配到的IP不一样用户按自己实际情况定），因为开发板是作为服务器的，所以这个是服务器IP。

由于是开发板作为服务器，电脑网络助手做客户端，所以网络调试助手选择用TCP Client，要连接到的开发板作为的服务器，网络助手的服务器IP地址设置为192.168.0.102（这个IP地址用户根据自己实际路由器分配到的IP进行填写），端口号跟程序设置的一样：2040。



3、实验现象

前面都设置好后，网络助手点击连接，网络助手就连接上开发板的服务器。之后网络助手点击发送，此时就是客户端向服务器发送数据，服务器接到数据后，将接收到的数据显示到屏幕上，按KEY0，开发板把程序里的测试数据发给网络助手。



从屏幕中我们可以看出路由器为开发板分配到的IP。

第 28 章 TCP 客户端数据收发实验

3.28.1 硬件设计

同第 27 章。

3.28.2 核心代码解析

请仔细看程序代码。

3.28.3 实验操作与现象

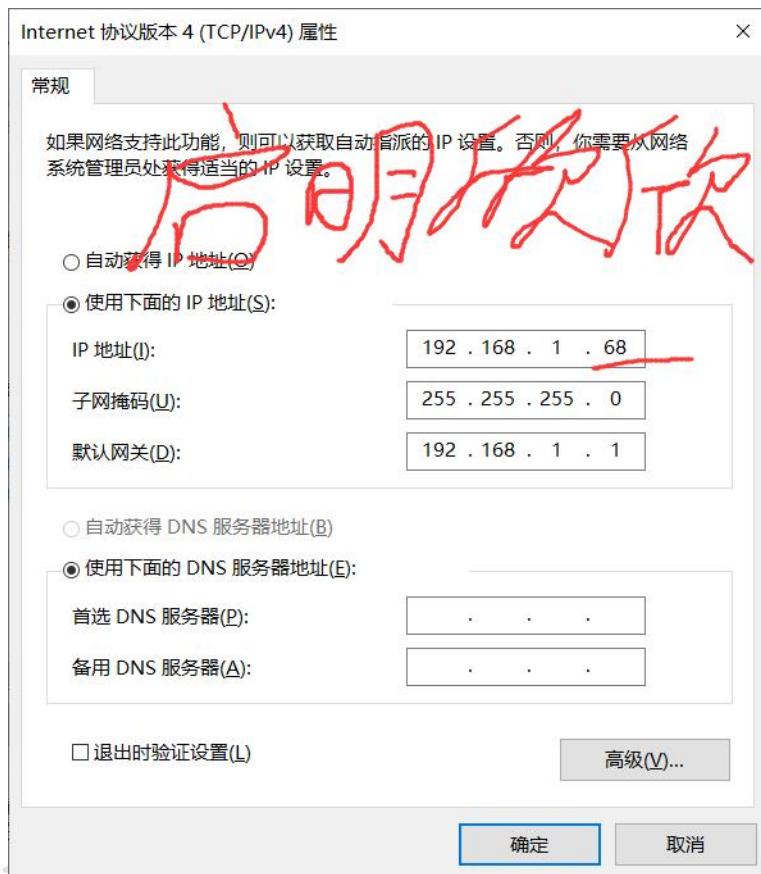
跟上一章一样，两种接网线方法：

- (1)直接通过网线连接开发板和电脑
- (2)开发板和电脑都连接路由器

采用第(1)方法：

由于电脑没有DHCP分配IP的功能，所以开发板在自动获取IP超时后，会使用程序设置的静态IP192.168.1.240，作为开发板的IP地址。

1、设置电脑本地IP地址

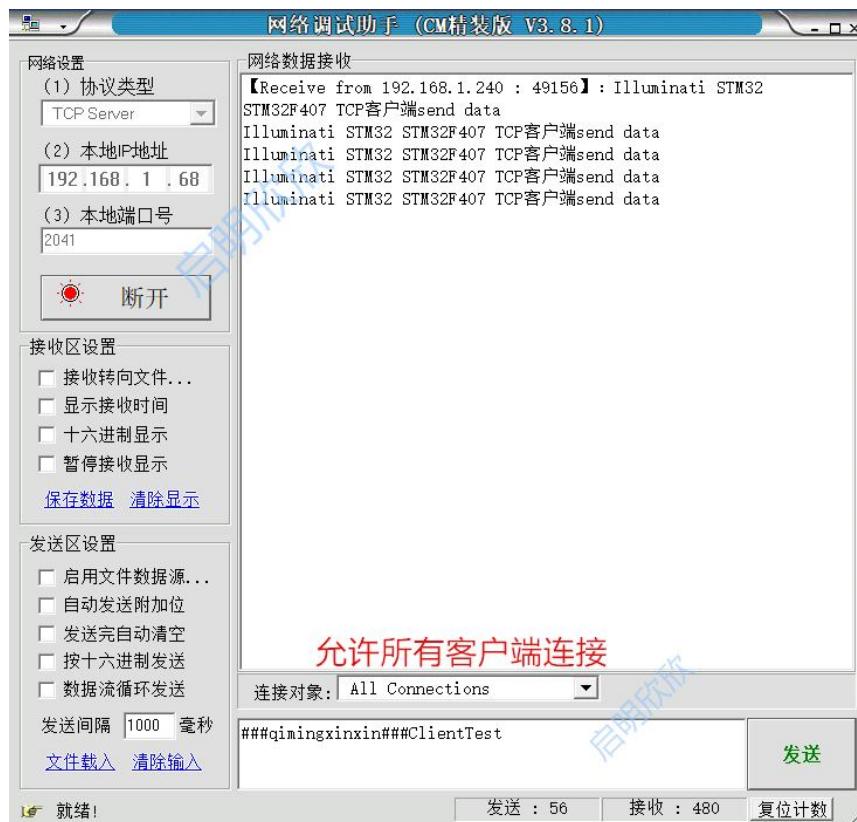


由于是开发板作为客户端，电脑网络助手做服务器，所以网络调试助手选择用TCP Server，网络助手的本地IP(服务器IP)设置为地址设置为192.168.1.68，端口号跟程序设置的一样：2041。

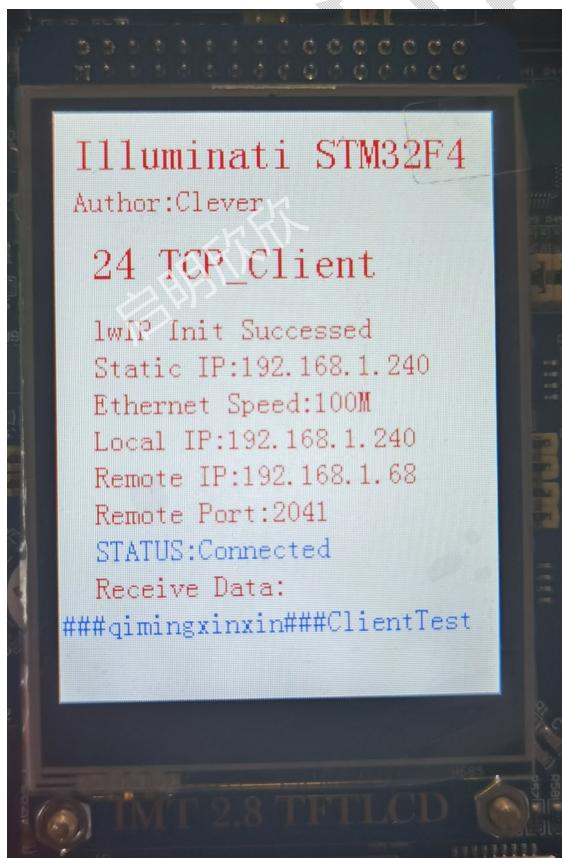
知道服务器IP，程序就要写一样的IP，如下图设置“68”

```
//设置远端服务器IP 前三个IP保持和DHCP得到的IP一致
lwipdev.remoteip[0]=lwipdev.ip[0];
lwipdev.remoteip[1]=lwipdev.ip[1];
lwipdev.remoteip[2]=lwipdev.ip[2];
lwipdev.remoteip[3]=68;
//程序默认68，实际用户可以根据服务器的ip填写在编译程序下载到开发板
//就是要知道远端服务器的IP地址，开发板作为客户端才可以连接
```

2、打开网络助手设置



3、实验现象



网络助手在点击连接的时候，WIN10系统的需要关闭防火墙（实测只在通过网线连接开发板和电脑这种情况下才需要**关闭电脑防火墙**才能连接成功，如果是接路由器DHCP无需关闭防火墙）。

当连接成功后屏幕会显示：
STATUS:Connected
此时网络助手点击发送，就是服务器向客户端(开发板)发送数据并显示在屏幕上，开发板按KEY0，向服务器发送测试数据。

采用第(2)方法:

开发板和电脑都连接路由器，路由器会为电脑和开发板分配IP地址。

1、电脑本地IP设置为自动获取IP



2、打开网络助手设置

程序运行后，开发板进行DHCP，路由器为开发板分配到IP:
192.168.0.102(不同路由器分配到的IP不一样按用户自己的**实际**情况定)，开发板是作为客户端故此IP为客户端IP，开发板要连接到服务器就要知道电脑端的服务器IP，此时可以在电脑上查看网络属性，找到路由器给电脑分配到的IP。电脑查看IP方法：

I、WIN+R, 调出，命令窗口

II、输入命令：ipconfig

III、找到IPV4地址

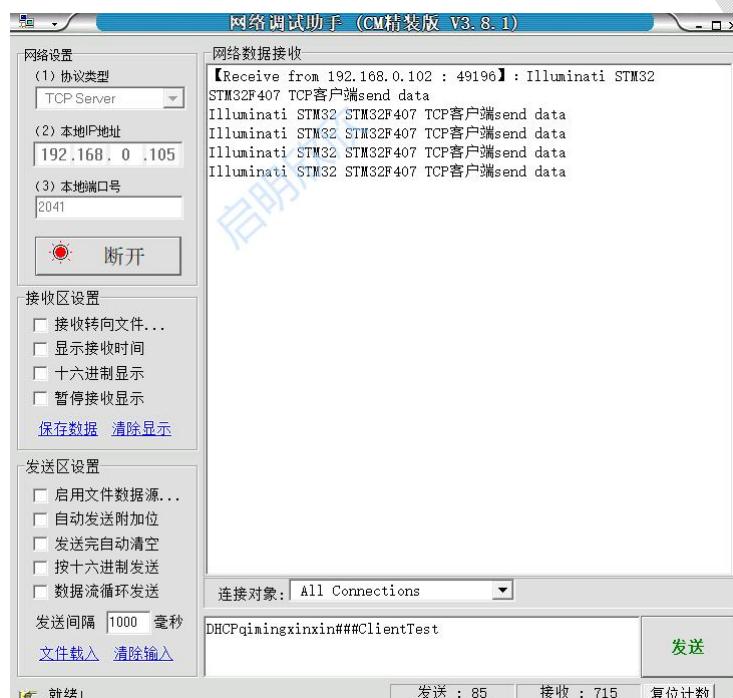
```
选择C:\WINDOWS\system32\cmd.exe
媒体状态 连接特定的 DNS 后缀 . . . . . : 媒体已断开连接
无线局域网适配器 本地连接* 12:
    媒体状态 连接特定的 DNS 后缀 . . . . . : 媒体已断开连接
    以太网适配器 以太网:
        连接特定的 DNS 后缀 . . . . . : fe80::a19a:59d:1b7e:6750%17
        本地链接 IPv6 地址 . . . . . : fe80::a19a:59d:1b7e:6750%17
        IPv4 地址 . . . . . : 192.168.1.68
        子网掩码 . . . . . : 255.255.255.0
        默认网关 . . . . . : 192.168.1.1
    无线局域网适配器 WLAN:
        连接特定的 DNS 后缀 . . . . . : fe80::8595:c98:7c75:e72b%18
        本地链接 IPv6 地址 . . . . . : fe80::8595:c98:7c75:e72b%18
        IPv4 地址 . . . . . : 192.168.0.105
        子网掩码 . . . . . : 255.255.255.0
        默认网关 . . . . . : 192.168.0.1
以太网适配器 蓝牙网络连接:
    媒体状态 连接特定的 DNS 后缀 . . . . . : 媒体已断开连接
C:\Users\clever>
```

启明欣欣 STM32F407 开发板(高配版) V5.1 使用手册

按上面的方面找到的IP就是开发板作为客户端要连接的服务器的IP地址，这个用户自己按实际找到的IP地址，把此IP地址写进去程序再编译后下载到开发板。我们找到的IP地址是192.168.0.105。

```
//设置远端服务器IP 前三个IP保持和DHCP得到的IP一致
lwipdev.remoteip[0]=lwipdev.ip[0];
lwipdev.remoteip[1]=lwipdev.ip[1];
lwipdev.remoteip[2]=lwipdev.ip[2];
lwipdev.remoteip[3]=105;
//程序默认68，实际用户可以根据服务器的ip填写在编译程序下载到开发板
//就是要知道远端服务器的IP地址，开发板作为客户端才可以连接
```

网络助手设置如下：



3、实验现象



网络助手点击连接后，当连接成功后屏幕会显示： STATUS:Connected

此时网络助手点击发送，就是服务器向客户端（开发板）发送数据并显示在屏幕上，开发板按KEY0，向服务器发送测试数据。

第 29 章 UDP 客户端数据收发实验

3.29.1 硬件设计

同第二十三章。

3.29.2 核心代码解析

请仔细看程序代码。

3.29.3 实验操作与现象

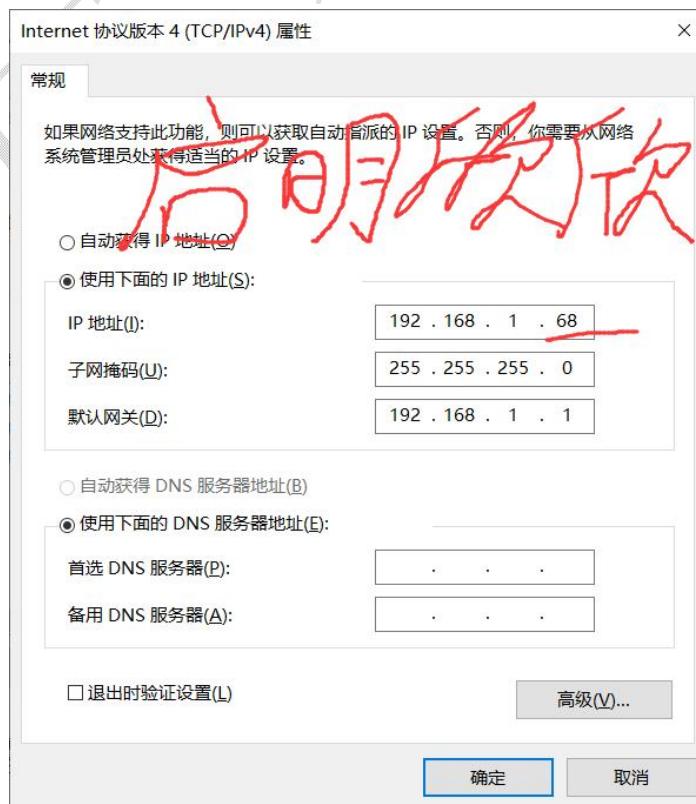
跟上一章一样，两种接网线方法，UDP操作跟TCP客户端基本一样：

- (1)直接通过网线连接开发板和电脑
- (2)开发板和电脑都连接路由器

采用第(1)方法：

由于电脑没有DHCP分配IP的功能，所以开发板在自动获取IP超时后，会使用程序设置的静态IP192.168.1.240，作为开发板的IP地址。

1、设置电脑本地IP地址



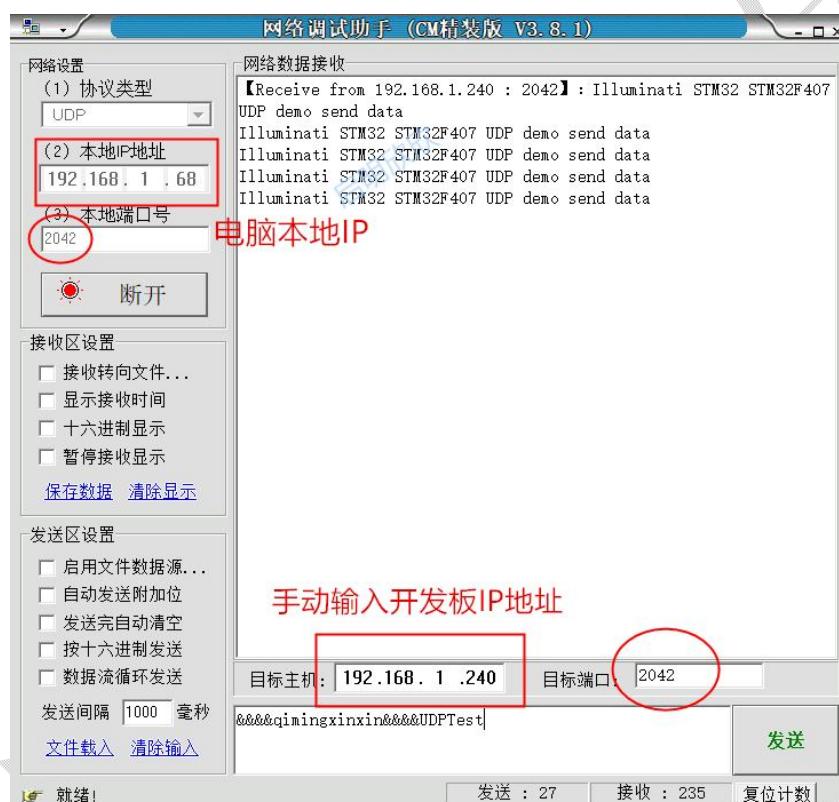
启明欣欣 STM32F407 开发板(高配版) V5.1 使用手册

开发板作为UDP客户端，网络调试助手选择用UDP，网络助手的本地IP设置为地址设置为192.168.1.68，端口号跟程序设置的一样：2042。点击连接后设置目标主机地址，也就是开发板的IP地址，端口号也是2042。

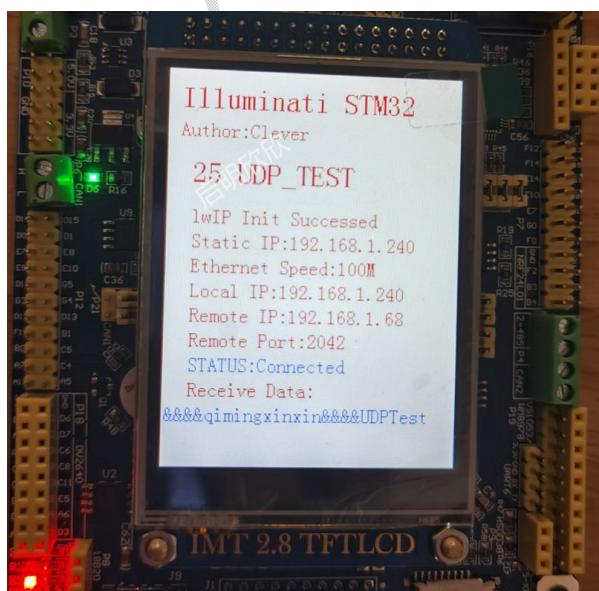
知道UDP服务器IP，程序就要写一样的IP，如下图设置“68”

```
//设置远端服务器IP 前三个IP保持和DHCP得到的IP一致  
lwipdev.remoteip[0]=lwipdev.ip[0];  
lwipdev.remoteip[1]=lwipdev.ip[1];  
lwipdev.remoteip[2]=lwipdev.ip[2];  
lwipdev.remoteip[3]=68;  
//程序默认68，实际用户可以根据服务器的ip填写在编译程序下载到开发板  
//就是要知道远端服务器的IP地址，开发板作为客户端才可以连接
```

2、打开网络助手设置



3、实验现象



网络助手点击连接后，当连接成功后屏幕会显示：
STATUS:Connected
此时网络助手点击发送，发送数据给开发板并显示在屏幕上，开发板按KEY0，向网络助手发送测试数据。

采用第(2)方法:

开发板和电脑都连接路由器，路由器会为电脑和开发板分配IP地址。

1、电脑本地IP设置为自动获取IP



2、打开网络助手设置

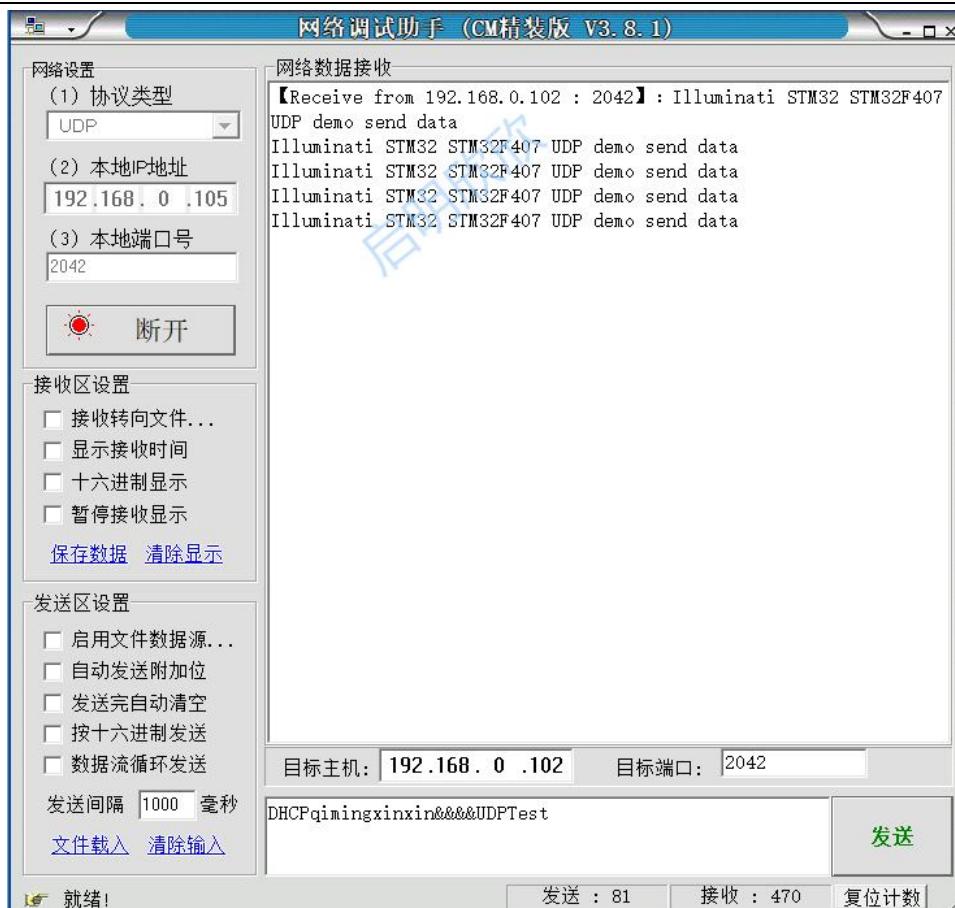
程序运行后，开发板进行DHCP，从上一章我们知道路由器为开发板分配到IP: 192.168.0.102(不同路由器分配到的IP不一样按用户自己的实际情况定)，为电脑分配到的IP是192.168.0.105。

把IP地址192.168.0.105写进去程序再编译后下载到开发板。

```
//设置远端服务器IP 前三个IP保持和DHCP得到的IP一致
lwipdev.remoteip[0]=lwipdev.ip[0];
lwipdev.remoteip[1]=lwipdev.ip[1];
lwipdev.remoteip[2]=lwipdev.ip[2];
lwipdev.remoteip[3]=105;
//程序默认68，实际用户可以根据服务器的ip填写在编译程序下载到开发板
//就是要知道远端服务器的IP地址，开发板作为客户端才可以连接
```

网络助手设置如下：

启明欣欣 STM32F407 开发板(高配版) V5.1 使用手册



3、实验现象



网络助手点击连接后，当连接成功后屏幕会显示： STATUS:Connected 此时网络助手点击发送，就是服务器向客户端（开发板）发送数据并显示在屏幕上，开发板按KEY0，向服务器发送测试数据。

第 30 章 串口 1-232 与 TCP 服务器双向通信

3.30.1 硬件设计

用到串口 1 和网口相关硬件电路，参考前面两者电路

3.30.2 核心代码解析

232 到 TCP 的代码：

```
else if(rec_data=='E')           //如果 E, 表示是命令信息传送的结束位
{
    //接收到的一帧数据发送到与开发板连接的客户端
    for(cpcb = tcp_active_pcbs;cpcb != NULL; cpcb = cpcb->next)
    {
        cpcb -> flags = TF_NODELAY;
        tcp_write(cpcb,&receive_str,uart_byte_count,TCP_WRITE_FLAG_COPY);
        tcp_output(cpcb);
    }
}
```

TCP 到 232 的代码：

```
if(p_temp != NULL)
{
    tcp_recved(pcb, p_temp->tot_len);      //获取数据长度 tot_len: tcp 数据块
                                                //的长度
    while(p_temp != NULL)
    {
        uart1SendChars(p_temp->payload,p_temp->len); //接收到的数据从
                                                //串口 1-232 发送出去 payload 为 TCP 数据块的起始位置
        p_temp = p_temp->next;
    }
}
```

3.30.3 实验操作与现象

网口部分操作跟TCP服务器一样，串口1-232跟前面232例程操作

一样。如果下图接上232串口线和网线。



3、实验现象

前面都设置好后，网络助手和串口助手，全部设置好后点击连接，网络助手就连接上开发板的服务器，串口助手也连接上串口线之后，网络助手发送的数据会传到串口助手，同样串口助手发送的数据也会传到网络助手助手。

启明欣欣 STM32F407 开发板(高配版) V5.1 使用手册

