

# 技术路线文档

撰写人：陆熠熠

撰写时间：2020.01.13——2020.01.14

内容：1.0 基于 Keil5 的环境搭建

# 目录

第一章 有关综述和一些配置.....	3
1.1 有关 STM32F407ZGT6 的选择和概述.....	3
1.2 学习 STM32 的软件平台.....	3
1.3 STM32 程序开发的环境搭建.....	4
1.3.1 安装 Keil.....	4
1.3.2 建立工程.....	5
1.4 工程模板的建立（细致版）.....	8
1.4.1 STM32 的固件库.....	8
1.4.2 有关固件库的分析和笔记.....	9
1.5 程序的烧写.....	11
第二章 有关 GPIO 的输入和输出 1.....	12
2.1 输出方式.....	13
2.2 输入方式.....	17
2.2.1 单功能按键输入.....	17
2.2.2 复用功能按键输入.....	19
2.2.3 非按键类开关信号输入.....	23
2.3 GPIO 输入/输出小结.....	23
第三章 有关 PWM 波的原理和应用.....	25
3.1 有关 PWM 波的原理.....	25
3.2 有关 PWM 的基本应用场景.....	25
3.3 STM32 的 PWM 实现原理.....	26
3.3.1 STM32 的 PWM 原理.....	26
3.3.2 STM32 的 PWN 程序实现步骤.....	26
3.3.3 STM32PWM 程序实现的三个要点.....	27
3.4 基于 PWM 的呼吸灯的实现思路.....	28
3.4.1 实现思路.....	28
3.4.2 实现程序.....	28

# 第一章 有关综述和一些配置

## 1.1 有关 STM32F407ZGT6 的选择和概述

根据程序存储容量，ST 芯片分为三大类：“LD”<64KB，“MD”<256KB），“HD”>256KB。我们所选用的 STM32F407ZGT6 属于第三类。

以下是他的性能简介：**（红色为 RM 比赛需要重点了解的部分）**

（1）内核：带有 FPU 的 ARM® 32 位 Cortex®-M4CPU、在 Flash 存储器中实现零等待状态运行性能的自适应实时加速器（ART 加速器™）。

（2）**高达 1 MB Flash：地址从 0x8000000 开始，大小为：0x100000 即 1M。**高达 192+4 KB 的 SRAM，包括 64-KB 的 CCM（内核耦合存储器）数据 RAM。

I、普通内存：地址从 0x20000000 开始，大小为：0x20000 即 128k，这部分内存任何外设都可以访问。

II、**CCM 内存：地址从 0x10000000 开始，大小为：0x10000 即 64k，**这部分内存仅 CPU 可以访问，DMA 之类的不可以直接访问。

III、备份 SRAM：大小为：4 K

（3）LCD 并行接口，兼容 8080/6800 模式

（4）时钟、复位和电源管理：支持 1.8 V 到 3.6 V 供电和 I/O，支持 4 至 26 MHz 晶振，内置经工厂调校的 16 MHz RC 振荡器（1%精度）。

（5）低功耗，**支持睡眠、停机和待机模式。**VBAT 可为 RTC、20×32 位备份寄存器 + 可选的 4 KB 备份 SRAM 供电。

（6）3 个 12 位、2.4 MSPS ADC：多达 24 通道，三重交叉模式下的性能高达 7.2 MSPS

（7）2 个 12 位 D/A 转换器

（8）通用 DMA：具有 FIFO 和突发支持的 16 路 DMA 控制器

（9）**多达 17 个定时器，12 个 16 位定时器，和 2 个频率高达 168 MHz 的 32 位定时器，**每个定时器都带有 4 个输入捕获 / 输出比较 /PWM，或脉冲计数器与正交（增量）编码器输入。

（10）**多达 140 个具有中断功能的 I/O 端口，**高达 136 个快速 I/O，最高 84 MHz，高达 138 个可耐 5 V 的 I/O。

（11）**多达 15 个通信接口。2 个 CAN（2.0B 主动）**以及 SDIO 接口。

（12）高级连接功能：具有片上 PHY 的 USB 2.0 全速器件/主机/OTG 控制器。具有专用 DMA、片上全速 PHY 和 ULPI 的 USB 2.0 高速 /全速器件 /主机 /OTG 控制器。具有专用 DMA 的 10/100 以太网 MAC：支持 IEEE 1588v2 硬件，MII/RMII。

（13）**8~14 位并行照相机接口：**速度高达 54MB/s

（14）真随机数发生器

## 1.2 学习 STM32 的软件平台

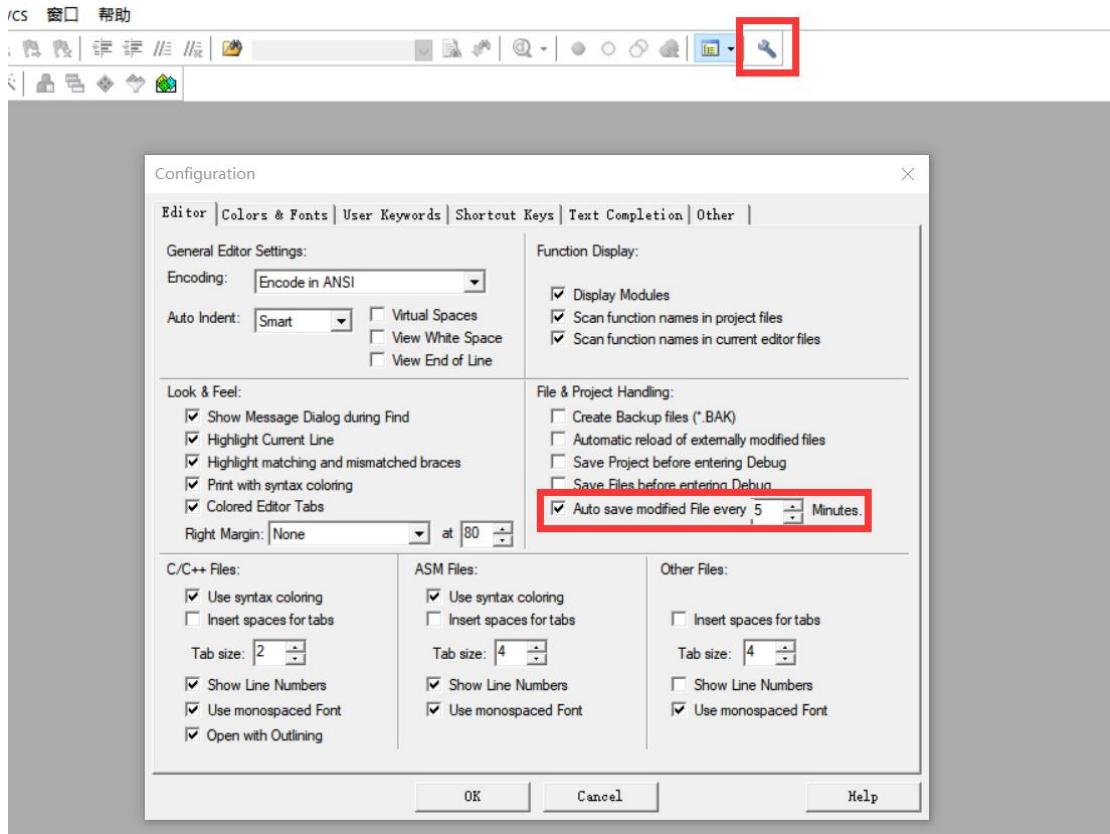
烧写 STM32 必须有一个开发平台：我这里因为手边具有教程和交材，所以使用 Keil。

Keil MDK 是 ARM 公司提供的编译环境，我使用的版本 Keil5 具有支持自动补全关键字的功能，非常方便，这个功能在程序设计时十分实用。要开启这样的功能我们首先需要进行

一个设置。

EDIT 菜单——Configuration——勾选其中的 Symbol after 3 characters。

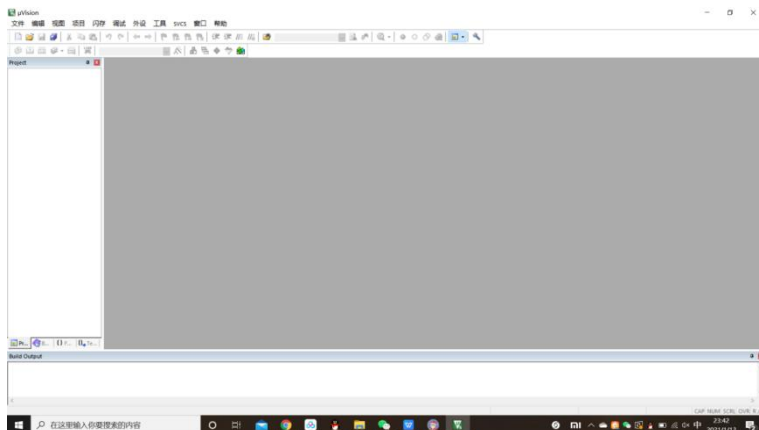
(好家伙, 由于我装了 Keil5 的汉化补丁, 我好像找不到这个东西...) 那我就推荐一下防止写代码不慎关机的自动保存功能。



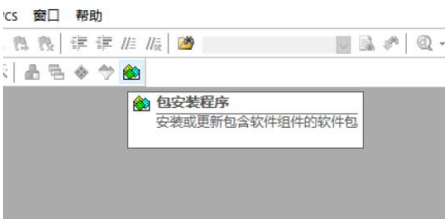
## 1.3 STM32 程序开发的环境搭建

### 1.3.1 安装 Keil

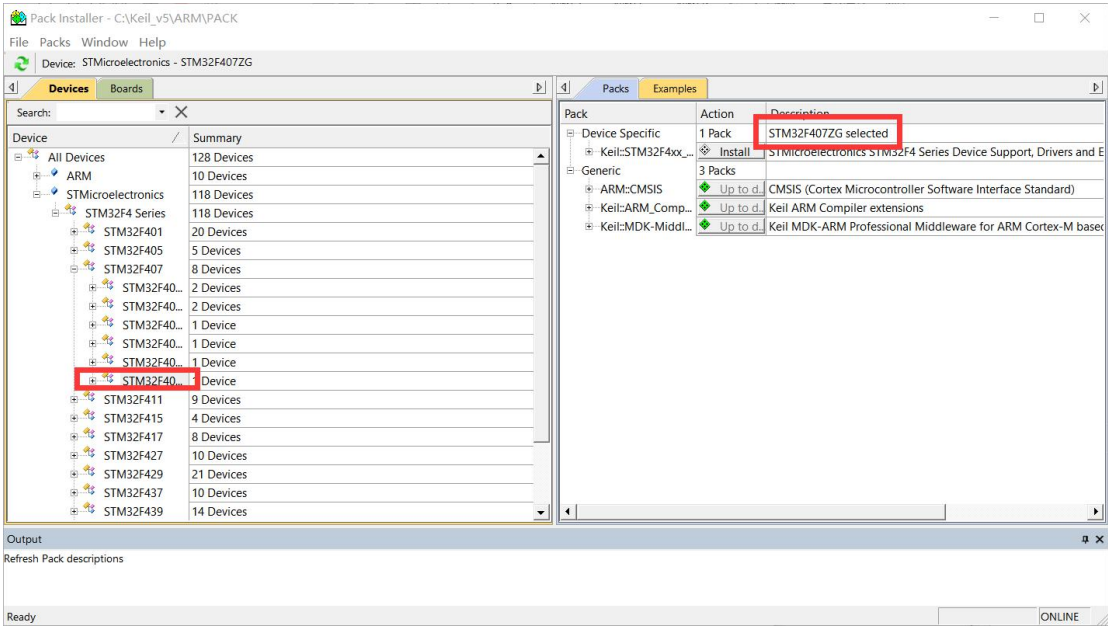
我的 Keil 是不知道谁给我的, 所以这是一个好问题。如果有需要建议去官网 <http://www.keil.com/download/product/> 下载并安装, 注意可以更改安装路径, 但是不能安在需要管理员权限的文件夹, 例如不能在 Program Files, 否则会出现一些问题。



打开 Pack Installer。



左边选择 STMicroelectronics，右边选择 STM32F4ZG。



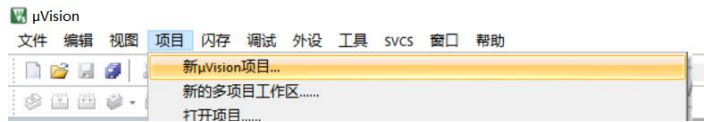
在 Action 里面会出现很多按钮，他们有自己独特的含义：

- 1) install: 需要安装
- 2) update: 需要升级
- 3) up to date: 已安装到最新版

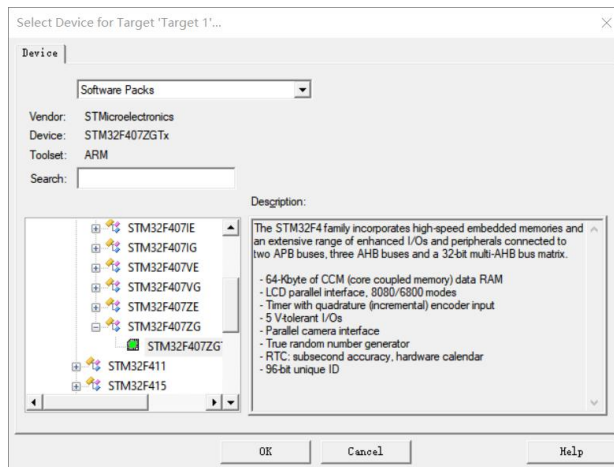
Examples		
	Action	Description
pecific	1 Pack	STM32F407
STM32F4xx_...	Install	STMicroele
	3 Packs	
:CMSIS	Up to d..	CMSIS (Cor
ARM_Comp...	Up to d..	Keil ARM C
MDK-Middl...	Up to d..	Keil MDK-A

### 1.3.2 建立工程

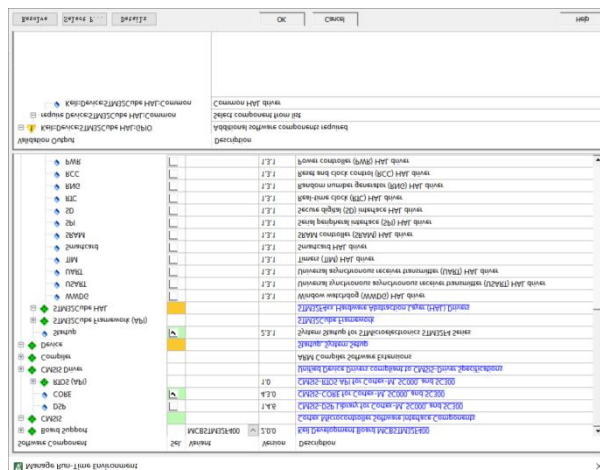
一般选择在一个空文件夹建立工程，会建立很多文件。



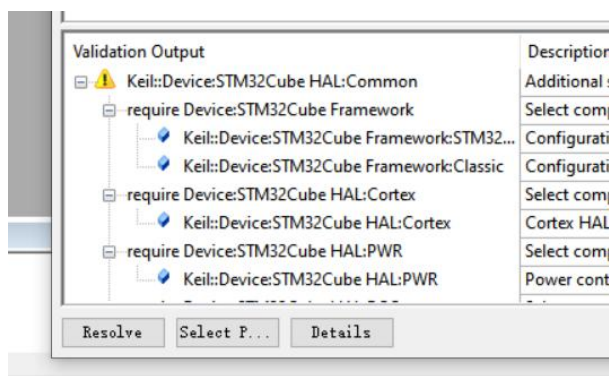
选择 STM32F407ZG 的芯片型号。



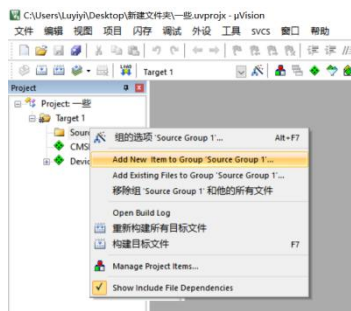
这是运行环境管理窗口，我们选择 CMSIS——> CORE 和 Device——>Startup 和 STM32Cube HAL ——> GPIO。



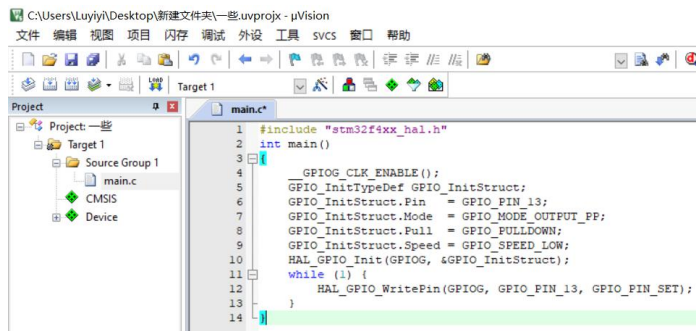
下方会提示需要哪些库，都需要进行选择。（单击可以直接找到）



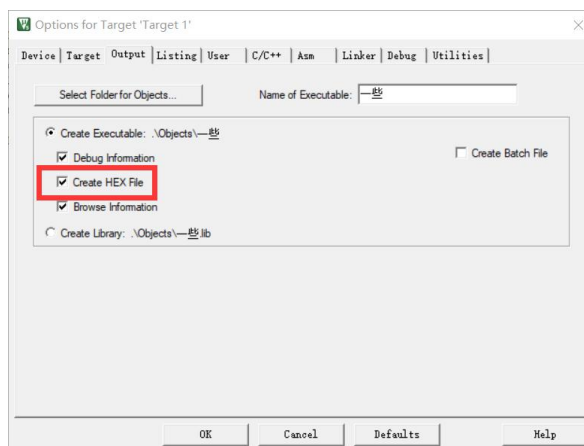
搭建主函数 main 文件。在 Source Group1 右键，添加一个命名为 main 的文件。



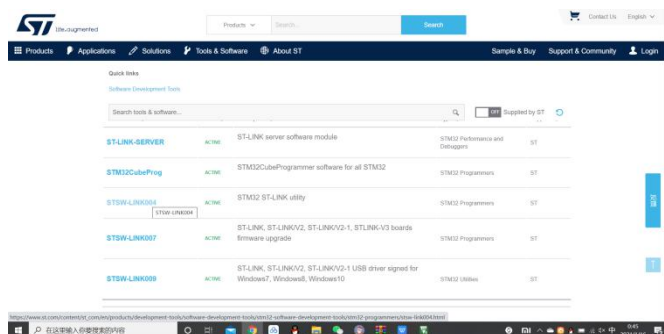
在这个页面导入我们需要的主函数，以点亮 LED 小灯为例。



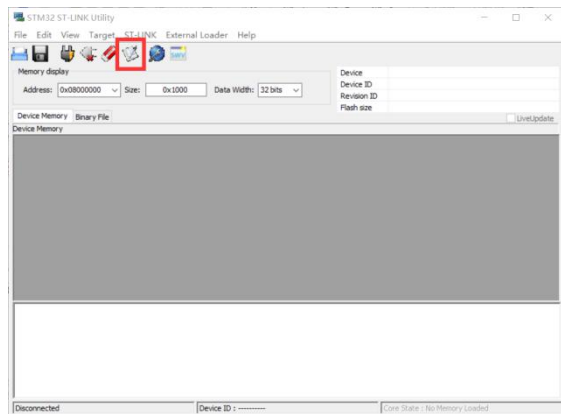
打开“目标选项”，勾选配置 HEX 文件。



接下来，我们要上官网获取我们所需要的驱动。



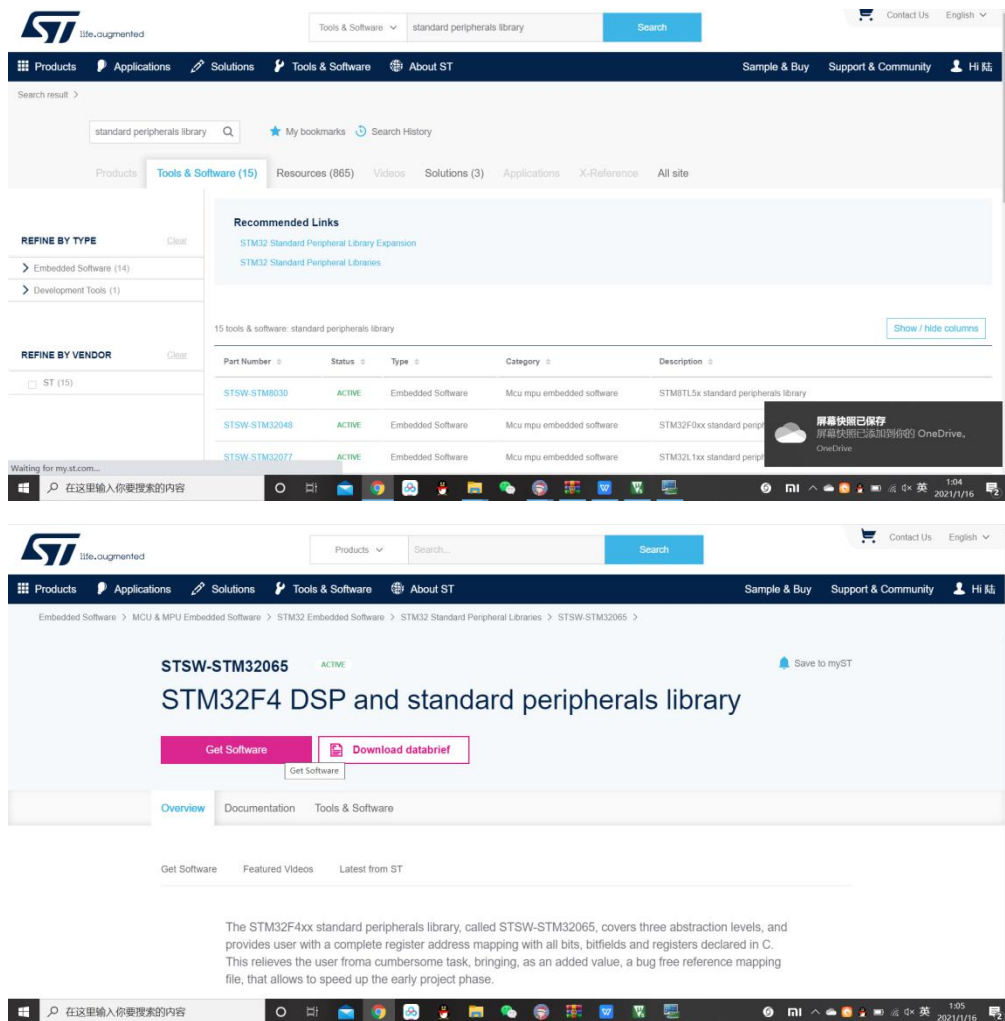
这是烧录程序，点击后点 start 就会烧录程序。重新编译后也要重新选择一次 HEX 文件。



## 1.4 工程模板的建立（细致版）

### 1.4.1 STM32 的固件库

固件库是一个包含所有的标准外设的设备驱动程序。  
我们可以从官网下载固件库并且加以使用。

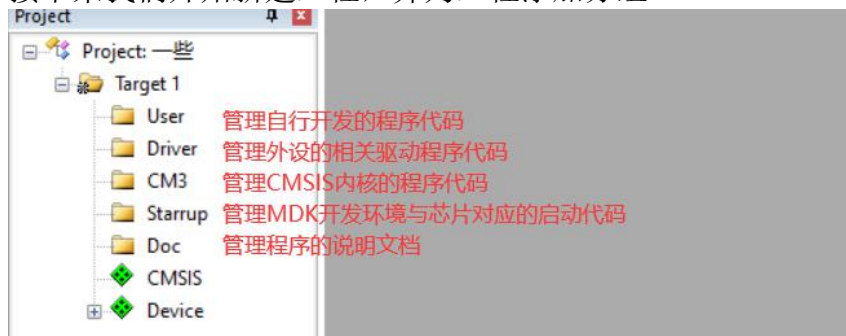




## 1.4.2 有关固件库的分析和笔记

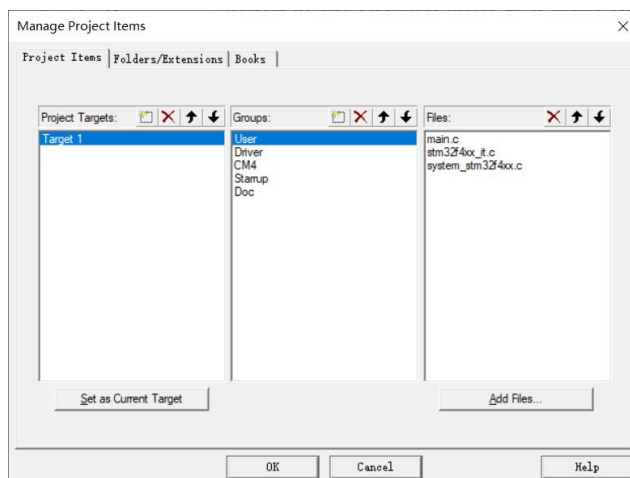
名称	修改日期	类型
_htmresc	2016/11/10 18:32	文件夹
Libraries 下载驱动库的源代码与启动文件	2016/11/10 18:33	文件夹
Project 驱动库的例子和每一个工程模板	2016/11/10 18:34	文件夹
Utilities	2016/11/10 18:34	文件夹
MCD-ST Liberty SW License Agree...	2016/11/5 1:32	WPS PDF 文档
Release_Notes	2016/11/9 2:34	URL:HyperText ...
stm32f4xx_dsp_stdperiph_lib_um	2016/11/8 23:33	编译的 HTML 帮...

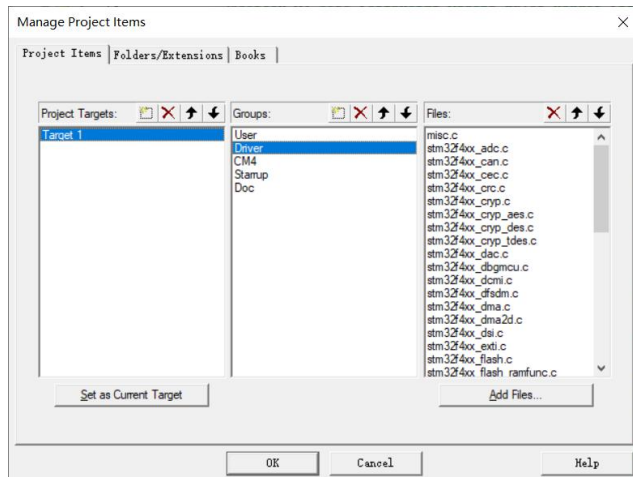
接下来我们开始新建工程，并为工程添加分组。



从我们的固件库里选择并且复制需要的文件。

大概像下面这样。（一些操作）

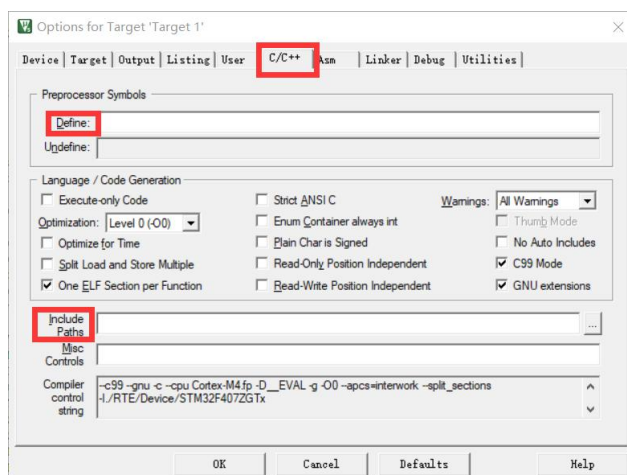
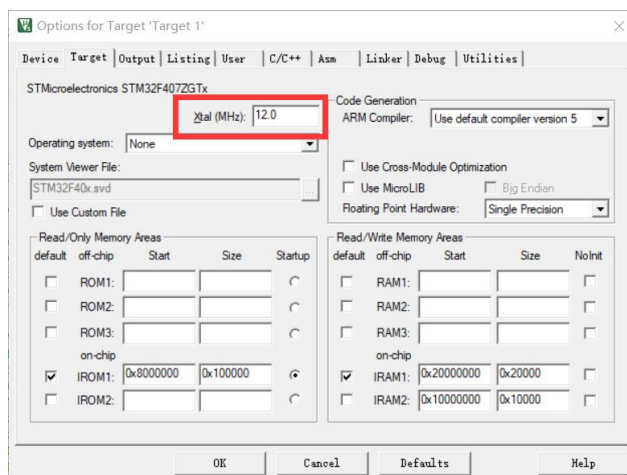




然后对 KEIL 的开发环境进行设置。

我们选择 Options for Target1。

这里外部时钟，可以根据晶振需求来调整。

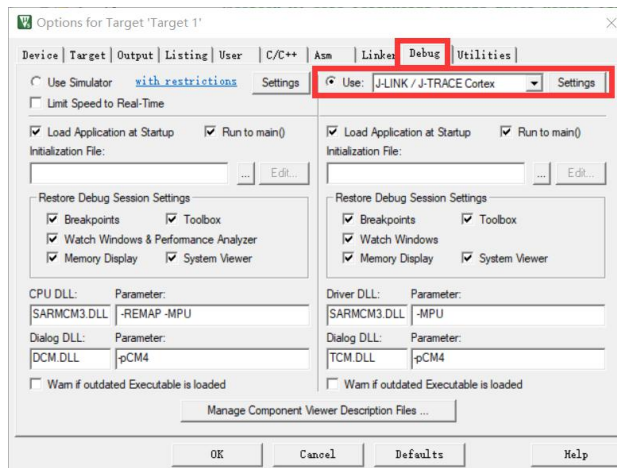


在 C/C++ 页面中主要设置这两个。

Define: 设职位编译过程中的预处理宏定义符号。

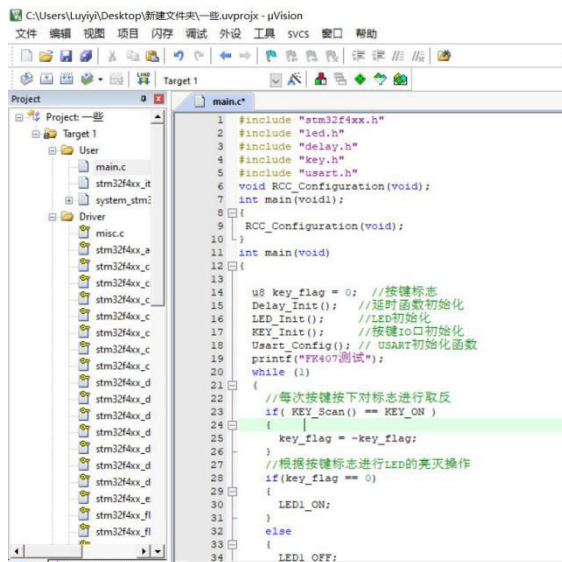
Inuclude Path: 设置为编译过程中文件包要查找的路径。

在 Debug 里面选择使用的仿真器。



之后，我们欸指以下工程模板中的相关文件。

对 main.c 的文件加以调整。只保留部分。



在这里我保留了系统时钟配置函数，保留了默认值，并且添加了我需要的点亮 led 的函数。

## 1.5 程序的烧写

每个 STM32 上都有两个引脚 BOOT0 和 BOOT1，他们在芯片复位时候的电平状态决定了他们从哪个区域开始执行。

当 BOOT0=x 和 BOOT1=0，从用户闪存启动，这是正常的工作模式。

当 BOOT0=0 和 BOOT1=1，从系统存储器启动。他是一个 ROM 区。

当 BOOT0=1 和 BOOT1=1，从内置 SRAM 中启动，可以用来调试。（掉电后数据会消失。）

一般当 BOOT0 和 BOOT1 都接 GND，在 ISP 下载的时候当 BOOT0=1、BOOT1=0，下载完成后都接 0，就会正常工作。

## 第二章 有关 GPIO 的输入和输出 1

这里补充三种设置延时的方式：

### 1) 基于延时函数的延时

```
//有关延时函数的延时代码
void delay_nums(u16 time)
{
    u16 i=0;
    while(time--)
    {
        i=12000;
        while(i--)
        {
        }
    }
}
```

### 2) SysTick 中断延时

```
//有关 SysTick 中断延时
//stm3210x_it.h
extern _IO uint32_t TimingDeelay;
//stm32f40x_it.c 定义中断函数
void SysTick_Handler(void)
{
    if(TimingDeelay !=0x00)
    {
        TimingDeelay--;
    }
}
//mian.c 定义函数和全局变量
void Init_SysTick(void)
{
    if(SysTick_Config(SystemCoreClock/1000))//1ms 的时间基准
    while(1)
}
//延时函数
_IO uint32_t TimingDeelay;
void delay_ms(_IO uint32_t nTime)
{
    TimingDeelay = nTime;
    while(TimingDeelay!=0);
}
```

### 3) 定时器中断延时

```
//有关定时器中断延时
//前期操作：1) 由于使用了 TIMx 所以将#include stm32f40x_tim.h 使用
//stm32f40x_it.c 加入中断函数
void TIM3_IROHandler(void)
```

```

{
    if(TIM_GetITStatus(TIM3,TIM_IT_Update)!=RESET)
        //检查 TIM 中断是否发生
    {
        TIM_ClearITPendingBit(TIM3,TIM_IT_Update);
        //清除中断待处理位: TIM 中断源

        i++;
        //外部全局变量 1s 点亮 1s 关闭
        if(i==1000)
        {
            LED0_ON;
        }
        if(i=2000)
        {
            LED0_OFF;
            i=0;
        }
    }
}
//stm32f40x_it.h 加入
extern u16 i;//外部变量说明, 在主函数致中定义, 在中断函数中使用
void TIM3_IROHandler(void);//TIM3 中断函数说明

```

## 2.2 GPIO 输出——GPIO 口的各种输出方式及其应用

### 2.1 输出方式

1) 推免输出：可以输出高、低电平，连接数字器件。

2) 开漏输出：输出端相当于三极管的继电器。如果要得到高电平状态需要外界上拉电阻。可以做电流型的驱动。

该项目的接线方式：

D2——PC8（推免输出）

D4——PC9（推免输出）

D6——PA8（开漏输出）

D7——PA11（开漏输出）

D8——PA12（开漏输出）

1、在 conf.h 中选择我们需要的内容，把不需要的注销掉

```

1
2 #ifndef __STM32F4xx_CONF_H
3 #define __STM32F4xx_CONF_H
4 /*#include "stm32f4xx_adc.h"
5 #include "stm32f4xx_crc.h"
6 #include "stm32f4xx_dbgmcu.h"
7 #include "stm32f4xx_dma.h"
8 #include "stm32f4xx_exti.h"
9 #include "stm32f4xx_flash.h"*/
10 #include "stm32f4xx_gpio.h"
11 /*#include "stm32f4xx_i2c.h"
12 #include "stm32f4xx_iwdg.h"
13 #include "stm32f4xx_pwr.h"*/
14 #include "stm32f4xx_rcc.h"
15 /*#include "stm32f4xx_rtc.h"
16 #include "stm32f4xx_sdio.h"
17 #include "stm32f4xx_spi.h"
18 #include "stm32f4xx_syscfg.h"
19 #include "stm32f4xx_tim.h"
20 #include "stm32f4xx_usart.h"
21 #include "stm32f4xx_wdg.h"*/
22 #include "misc.h" /* High level functions for NVIC and SysTick

```

## 2、在 main.c 中编写我们的核心代码

### 1) 配置用到的 I/O 接口

```

1 #include <stm32f4xx.h>
2 //宏定义
3 #define D2_ON GPIO_ResetBits(GPIOC,GPIO_Pin_8)
4 #define D2_OFF GPIO_SetBits(GPIOC,GPIO_Pin_8)
5 #define D4_ON GPIO_ResetBits(GPIOC,GPIO_Pin_9)
6 #define D4_OFF GPIO_SetBits(GPIOC,GPIO_Pin_9)
7 #define D6_ON GPIO_ResetBits(GPIOA,GPIO_Pin_8)
8 #define D6_OFF GPIO_SetBits(GPIOA,GPIO_Pin_8)
9 #define D7_ON GPIO_ResetBits(GPIOA,GPIO_Pin_11)
10 #define D7_OFF GPIO_SetBits(GPIOA,GPIO_Pin_11)
11 #define D8_ON GPIO_ResetBits(GPIOA,GPIO_Pin_12)
12 #define D8_OFF GPIO_SetBits(GPIOA,GPIO_Pin_12)
13 //配置用到的I/O接口

```

### 2) 对于 I/O 口的输出方式进行一些配（输出的方式和端口）

```

14 void LED_GPIO_Config(void)
15 {
16     GPIO_InitTypeDef GPIO_InitStructure;
17     PCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC,ENABLE); //使C端时钟开始
18     GPIO_InitStructure.GPIO_Pin=GPIO_Pin_8|GPIO_Pin_9; //选择引脚
19     GPIO_InitStructure.GPIO_Mode=GPIO_Mode_OUT_PP;
20     GPIO_InitStructure.GPIO_Speed=GPIO_Speed_10Mhz; //初始速度
21     GPIO_Init(GPIOC,&GPIO_InitStructure); //初始化C端口
22
23     PCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE); //使A端时钟开始
24     GPIO_InitStructure.GPIO_Pin=GPIO_Pin_8|GPIO_Pin_11|GPIO_Pin_12; //选择引脚
25     GPIO_InitStructure.GPIO_Mode=GPIO_Mode_OUT_OD; //开漏输出
26     GPIO_InitStructure.GPIO_Speed=GPIO_Speed_10Mhz; //初始速度
27     GPIO_Init(GPIOA,&GPIO_InitStructure); //初始化C端口
28 }

```

### 3) 延时函数

```

29 //有关延时函数的延时代码
30 void delay_nums(ul6 time)
31 {
32     ul6 i=0;
33     while(time-->0)
34     {
35         i=12000;
36         while(i-->0)
37         {
38
39
40
41
42
43
44
45
46

```

### 4) LED 的初始状态

```

38 //LED的初始状态
39 void TurnOffALLled(void)
40 {
41     D2_OFF;
42     D4_OFF;
43     D6_OFF;
44     D7_OFF;
45     D8_OFF;
46 }

```

### 5) 主函数

```

47 //主函数
48 int main(void)
49 {
50     SystemInit();
51     LED_GPIO_Config();
52     TurnOffALLled();
53     while(1)
54     {
55         D2_ON;
56         delay_nums(1000);
57         D2_OFF;
58         delay_nums(1000);
59         D4_ON;
60         delay_nums(1000);
61         D4_OFF;
62         delay_nums(1000);
63         D6_ON;
64         delay_nums(1000);
65         D6_OFF;
66         delay_nums(1000);
67         D7_ON;
68         delay_nums(1000);
69         D7_OFF;
70         delay_nums(1000);
71         D8_ON;
72         delay_nums(1000);
73         D8_OFF;
74         delay_nums(1000);
75     }
76 }

```

(这是完整的有关输出方式 GPIO 的定义和程序 VSC 版)

```

//接哪
//D2—PC8 (推免输出)
//D4—PC9 (推免输出)
//D6—PA8 (开漏输出)
//D7—PA11 (开漏输出)
//D8—PA12 (开漏输出)
//输出的方式和端口
void LED_GPIO_Config(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    PCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE); //使 C 端时钟开始
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9; //选择引脚
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10Mhz; //初始速度
    GPIO_Init(GPIOC, &GPIO_InitStructure); //初始化 C 端口

    PCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE); //使 A 端时钟开始
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_11 | GPIO_Pin_12; //选择引脚
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT_OD; //开漏输出
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10Mhz; //初始速度
    GPIO_Init(GPIOA, &GPIO_InitStructure); //初始化 C 端口
}

//有关延时函数的延时代码
void delay_nums(u16 time)
{
    u16 i = 0;
    while(time--)
    {
        i = 12000;
        while(i--)
        {
        }
    }
}

```

```

//LED 的初始状态
void TurnOffALLled(void)
{
    D2_OFF;
    D4_OFF;
    D6_OFF;
    D7_OFF;
    D8_OFF;
}

//主函数
int main(void)
{
    SystemInit(); //篇日志系统时钟为 72Mhz, 这个步骤是可以省略的
    LED_GPIO_Config(); //GPIO 端口初始化
    TurnOffALLled(); //关灯
    while(1)
    {
        D2_ON;
        delay_nums(1000);
        D2_OFF;
        delay_nums(1000);
        D4_ON;
        delay_nums(1000);
        D4_OFF;
        delay_nums(1000);
        D6_ON;
        delay_nums(1000);
        D6_OFF;
        delay_nums(1000);
        D7_ON;
        delay_nums(1000);
        D7_OFF;
        delay_nums(1000);
        D8_ON;
        delay_nums(1000);
        D8_OFF;
        delay_nums(1000);
        //开关开关开关开关 (有输出方式的区别)
    }
}

```

运行问题：（固件库报错、待处理）

```

systemcoreclock = HSE_VALUE;
..\STM32F4xx_DSP_StdPeriph_Lib_V1.8.0\Project\STM32F4xx_StdPeriph_Templates\system_stm32f4xx.c(562): warning: #550-D: variable "pllsource" was set but never used
uint32_t tmp = 0, pllvc0 = 0, pllp = 2, pllsource = 0, pllm = 2;
..\STM32F4xx_DSP_StdPeriph_Lib_V1.8.0\Project\STM32F4xx_StdPeriph_Templates\system_stm32f4xx.c(562): warning: #550-D: variable "pllm" was set but never used
uint32_t tmp = 0, pllvc0 = 0, pllp = 2, pllsource = 0, pllm = 2;
..\STM32F4xx_DSP_StdPeriph_Lib_V1.8.0\Project\STM32F4xx_StdPeriph_Templates\system_stm32f4xx.c: 2 warnings, 2 errors
".\Objects\template.axf" - 3 Error(s), 2 Warning(s).

```



## 2.2 输入方式

### 2.2.1 单功能按键输入

主要算法与原理:

【1】GPIO 初始化为上拉输出。按键一个引脚 PB15，一个引脚接地。没有按下的时候 PB15 高电平，按下的时候 PB15 低电平。

【2】按键捕捉：不断判断 PB15 是否为高电平。

【3】按键奇数次、偶数次——二极管的点亮与熄灭。

步骤 1: stm32f4xx\_conf.h 选择需要的外设模块的头文件。（GPIO, RCC, MISC）

```

1
2 #ifndef __STM32F4xx_CONF_H
3 #define __STM32F4xx_CONF_H
4 /*#include "stm32f4xx_adc.h"
5 #include "stm32f4xx_crc.h"
6 #include "stm32f4xx_dbgmcu.h"
7 #include "stm32f4xx_dma.h"
8 #include "stm32f4xx_exti.h"
9 #include "stm32f4xx_flash.h"*/
10 #include "stm32f4xx_gpio.h"
11 /*#include "stm32f4xx_i2c.h"
12 #include "stm32f4xx_iwdg.h"
13 #include "stm32f4xx_pwr.h"*/
14 #include "stm32f4xx_rcc.h"
15 /*#include "stm32f4xx_rtc.h"
16 #include "stm32f4xx_sdio.h"
17 #include "stm32f4xx_spi.h"
18 #include "stm32f4xx_syscfg.h"
19 #include "stm32f4xx_tim.h"
20 #include "stm32f4xx_usart.h"
21 #include "stm32f4xx_wwdg.h"*/
22 #include "misc.h" /* High level functions for NVIC and SysTick (add-on to CMSIS functions) */
23

```

步骤 2: 撰写 mian.c

配置用到的 I/O 口和情况

```

#include <stm32f4xx.h>
//配置用到的I/o口和情况
#define D2_ON GPIO_ResetBits(GPIOC,GPIO_Pin_8)
#define D2_OFF GPIO_SetBits(GPIOC,GPIO_Pin_8)
#define S1_DOWN GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_15)=0
#define S1_UP GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_15)=1

```

设置

```

void LEDKEY_GPIO_Config(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOB,ENABLE); //使C,B端时钟开始
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_8;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_OUT_PP;
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50Mhz;
    GPIO_Init(GPIOC,&GPIO_InitStructure); //初始化C端口
    GPIO_SetBits(GPIOC,GPIO_Pin_8); //关闭LED

    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_15; //选择引脚
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_OUT_IPU; //上拉输入（复位成为高电平）
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50Mhz;
    GPIO_Init(GPIOB,&GPIO_InitStructure); //初始化B端口
}

```

下面使我们的可以通过判断和执行完成的主函数

```

34 //主函数
35 int main(void)
36 {
37     u8 kcnt=0;
38     SystemInit();
39     LEDKEY_GPIO_Config();
40     while(1)
41     {
42         if(S1_DOWN)
43         {
44             delay_nums(10);
45             if(S1_DOWN)
46                 kcnt++;
47             if(kcnt%2)
48                 D2_ON;
49             else
50                 D2_OFF;
51             while(S1_DOWN);
52         }
53     }
54 }

```

（以下是该部分的完整代码）

```

#include <stm32f44x.h>
//配置用到的 I/O 口和情况
#define D2_ON GPIO_ResetBits(GPIOC,GPIO_Pin_8)//控制发光二极管点亮和熄灭的宏语句
#define D2_OFF GPIO_SetBits(GPIOC,GPIO_Pin_8)//控制发光二极管点亮和熄灭的宏语句
#define S1_DOWN GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_15)=0//关于按键的定义
#define S1_UP GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_15)=1//关于按键的定义
void LEDKEY_GPIO_Config(void)
{
    GPIO_InitTypeDefGPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOB,ENABLE);//使 C,B 端时钟开始
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_8;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_OUT_PP;
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50Mhz;
    GPIO_Init(GPIOC,&GPIO_InitStructure);//初始化 C 端口
    GPIO_SetBits(GPIOC,GPIO_Pin_8);//关闭 LED

    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_15;//选择引脚
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_OUT_IPU;//上拉输入（复位成为高电平）
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50Mhz;
    GPIO_Init(GPIOB,&GPIO_InitStructure);//初始化 B 端口
}
//有关延时函数的延时代码
void delay_nums(u16 time)
{
    u16 i=0;
    while(time--)
    {
        i=12000;
        while(i--)
        {
        }
    }
}
//主函数
int main(void)
{

```

```

u8 kcnt=0;

SystemInit();
LEDKEY_GPIO_Config();
while(1)
{
    if(S1_DOWN)
    {
        delay_nums(10);

        if(S1_DOWN)
            kcnt++;

        if(kcnt%2)
            D2_ON;
        else
            D2_OFF;

        while(S1_DOWN);
    }
}
}

```

PS: GPIO 有两种常用的开关量输入方式：上拉输入和下拉输入。

上拉输入：GPIO\_Mode\_IPU，利用芯片的上拉电阻使得复位后的默认状态变为高电平，因此引脚无需外界上拉电阻。

下拉输入：GPIO\_Mode\_IDU，利用芯片的上拉电阻使得复位后的默认状态变为低电平，因此引脚无需外界下拉电阻。

## 2.2.2 复用功能按键输入

按键复用：一按键多用的形式。

1——场景切换。（根据上下文执行的不同切换模式）

2——时间切换。（根据按键时间长短决定模式）

3——组合按键。（根据按键数量的多少决定模式）

### 【1】功能

按键 S1 短按：控制灯 D2 的亮灭。

按键 S1 长按：系统自动重启。

### 【2】实现原理

按键 S1 按下——系统开始计时——判断是否为三秒以内——执行操作

### 【3】实现过程

选取程序需要的外设模块文件（同上例，不详细描述）

写取宏定义，包括按键和小灯。

```

#include <stm32f4xx.h>
//配置用到的I/o口和情况
#define D2_ON GPIO_ResetBits(GPIOC,GPIO_Pin_8)
#define D2_OFF GPIO_SetBits(GPIOC,GPIO_Pin_8)
#define S1_DOWN GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_15)=0
#define S1_UP GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_15)=1

```

对 I/O 口的配置进行定义和撰写

```

void LEDKEY_GPIO_Config(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOB,ENABLE); //使C,B端时钟开始
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_8;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_OUT_PP;
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_Init(GPIOC,&GPIO_InitStructure); //初始化C端口
    GPIO_SetBits(GPIOC,GPIO_Pin_8); //关闭LED

    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_15; //选择引脚
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_OUT_IPU; //上拉输入（复位成为高电平）
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_Init(GPIOB,&GPIO_InitStructure); //初始化B端口
}

```

### 延时函数的撰写

```

25 //有关延时函数的延时代码
26 void delay_nums(u16 time)
27 {
28     u16 i=0;
29     while(time-->0)
30     {
31         i=12000;
32         while(i-->0)
33     }
34 }

```

### 判断按键时间和场景

```

36 u8 KeyPressed(u16 time)
37 {
38     u16 cnt=0;
39     if(S1_DOWN)
40     {
41         delay_nums(10);
42         if(S1_DOWN)
43         {
44             while(S1_DOWN)
45             {
46                 delay_nums(10);
47                 cnt++;
48             }
49         }
50     }
51     if(cnt==0)
52     {
53         return 0;
54     }
55     else
56     {
57         if(cnt<time/10){
58             return 1;
59         }
60         else
61         {
62             return 2;
63         }
64     }
65 }

```

还要写一个系统软件复位函数，用来执行系统的复位要求。

```

66 //系统软件复位函数
67 void sysRST(void)
68 {
69     _set_FAULTMASK(1) //关闭中断
70     NVIC_SystemReset(); //系统软件复位
71 }

```

### 主函数

```

73 //主函数
74 int main(void)
75 {
76     u8 kcnt=0;
77     SystemInit();
78     LEDKEY_GPIO_Config();
79     while(1)
80     {
81         switch (KeyPressed(3000))
82         {
83             case 0:
84                 break;
85             case 1: kcnt++;
86                     if(kcnt%2){D2_ON;}
87                     else{D2_OFF;}
88                 break;
89             case 2:D2_ON;
90                 delay_nums(10);
91                 sysRST();
92                 break;
93         }
94     }
95 }

```

（以下是整个函数的代码）

```

//第二个程序，复位按键
#include <stm32f44x.h>
//配置用到的 I/O 口和情况
#define D2_ON GPIO_ResetBits(GPIOC,GPIO_Pin_8)//控制发光二极管点亮和熄灭的宏语句
#define D2_OFF GPIO_SetBits(GPIOC,GPIO_Pin_8)//控制发光二极管点亮和熄灭的宏语句
#define S1_DOWN GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_15)=0//关于按键的定义
#define S1_UP GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_15)=1//关于按键的定义
void LEDKEY_GPIO_Config(void)
{
    GPIO_InitTypeDefGPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOB,ENABLE);//使 C,B 端时钟开始
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_8;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_OUT_PP;
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_Init(GPIOC,&GPIO_InitStructure);//初始化 C 端口
    GPIO_SetBits(GPIOC,GPIO_Pin_8);//关闭 LED

    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_15;//选择引脚
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_OUT_IPU;//上拉输入（复位成为高电平）
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_Init(GPIOB,&GPIO_InitStructure);//初始化 B 端口
}
//有关延时函数的延时代码
void delay_nums(u16 time)
{
    u16 i=0;
    while(time--)
    {
        i=12000;
        while(i--)
        {
        }
    }
}
//参数 time 为判断时间设置，3s 就是 3000
//返回值：0 表示没有按键，1 表示短按，2 表示长按

```

```
u8 KeyPressed(u16 time)
{
    u16 cnt=0;
    if(S1_DOWN)
    {
        delay_nums(10);
        if(S1_DOWN)
        {
            while(S1_DOWN)
            {
                delay_nums(10);
                cnt++;
            }
        }
    }
    if(cnt=0)
    {
        return 0;
    }
    else
    {
        if(cnt<time/10){
            return 1;
        }
        else
        {
            return 2;
        }
    }
}

//系统软件复位函数
void sysRST(void)
{
    _set_FAULTMASK(1)//关闭中断
    NVIC_SystemReset();//系统软件复位
}

//主函数
int main(void)
{
    u8 kcnt=0;
    SystemInit();
    LEDKEY_GPIO_Config();
    while(1)
    {
```

```

switch (KeyPressed(3000))//扫描按键状态
{
case 0://没有按键
break;
case 1:kcnt++;//按键短暂
    if(kcnt%2){D2_ON;}
    else{D2_OFF;}
break;
case 2:D2_ON;//按键长按
delay_nums(10);
sysRST();//系统重启
break;
}
}
}

```

### 2.2.3 非按键类开关信号输入

GPIO 的引脚输入方式主要有四种：

**【1】上拉输入 (GPIO\_Mode\_IPU)**

信号进入芯片后被内部的一个上拉电阻上拉，再经过施密特触发器转换成为 0、1 信号。因此复位后引脚电平为高电平。

**【2】下拉输入 (GPIO\_Mode\_IPD)**

信号进入芯片后被内部的一个下拉电阻下拉，再经过施密特触发器转换成为 0、1 信号。因此复位后引脚电平为低电平。

**【3】模拟输入 (GPIO\_Mode\_AIN)**

信号不经过上拉电阻或者下拉电阻，也不经过施密特触发器，通过另外一个线路把电压信号传送到片商相应的外设模块，通常为 ADC 模块。模拟输入的信号事没有被处理过的信号。

**【4】浮空输入 (GPIO\_Mode\_IN\_FLOATING)**

信号不经过上拉电阻或者下拉电阻，只经过施密特触发器。把这种模式用于标准的通信协议，如 I2C，USART 等。

## 2.3 GPIO 输入/输出小结

● STM32 的 GPIO 输入/输出方式一共有八种。

**【1】上拉输入 (GPIO\_Mode\_IPU)**

**【2】下拉输入 (GPIO\_Mode\_IPD)**

**【3】模拟输入 (GPIO\_Mode\_AIN)**

**【4】浮空输入 (GPIO\_Mode\_IN\_FLOATING)**

**【5】开漏输出 (GPIO\_Mode\_Out\_OD)**

**【6】推免输出 (GPIO\_Mode\_Out\_PP)**

**【7】复用开漏输出 (GPIO\_Mode\_AF\_OD)——片内外设功能：TX1;MOSI;SCK;SS**

## 【8】复用推免输出(GPIO\_Mode\_AF\_PP)——片内外设功能：I2C(SCL\SDA)

## ● GPIO 设置。

读取引脚状态的调用语句

```
GPIO_ResetBits(GPIOC,GPIO_Pin_8)//控制发光二极管点亮和熄灭的宏语句
```

引脚拉高的调用语句

```
GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_15)=0//关于按键的定义
```

延时函数语句

```
//有关延时函数的延时代码
```

```
void delay_nums(u16 time)
```

```
{
    u16 i=0;
    while(time--)
    {
        i=12000;
        while(i--)
        {
        }
    }
}
```

系统复位语句

```
void sysRST(void)
```

```
{
    _set_FAULTMASK(1)//关闭中断
    NVIC_SystemReset();//系统软件复位
}
```

复位按键场景切换语句（时间）

```
//参数 time 为判断时间设置，3s 就是 3000
```

```
//返回值：0 表示没有按键，1 表示短按，2 表示长按
```

```
u8 KeyPressed(u16 time)
```

```
{
    u16 cnt=0;
    if(S1_DOWN)
    {
        delay_nums(10);
        if(S1_DOWN)
        {
            while(S1_DOWN)
            {
                delay_nums(10);
                cnt++;
            }
        }
    }
    if(cnt=0)
    {
        return 0;
    }
}
```



```

else
{
    if(cnt<time/10){
        return 1;
    }
    else
    {
        return 2;
    }
}
}
}

```

问题：（固件库报错、待处理）

```

systemcorelookup = HSE_VALUE;
..\STM32F4xx_DSP_StdPeriph_Lib_V1.8.0\Project\STM32F4xx_StdPeriph_Templates\system_stm32f4xx.c(562): warning: #550-D: variable "pllsource" was set but never used
uint32_t tmp = 0, pllvc = 0, pllp = 2, pllsource = 0, pllm = 2;
..\STM32F4xx_DSP_StdPeriph_Lib_V1.8.0\Project\STM32F4xx_StdPeriph_Templates\system_stm32f4xx.c(562): warning: #550-D: variable "pllm" was set but never used
uint32_t tmp = 0, pllvc = 0, pllp = 2, pllsource = 0, pllm = 2;
..\STM32F4xx_DSP_StdPeriph_Lib_V1.8.0\Project\STM32F4xx_StdPeriph_Templates\system_stm32f4xx.c: 2 warnings, 2 errors
".\Objects\template.axf" - 3 Error(s), 2 Warning(s).

```

## 第三章 有关 PWM 波的原理和应用

### 3.1 有关 PWM 波的原理

脉冲宽度调制的简称：脉宽调制。（Pulse Width Modulation）

主要有以下几种技术：

- 1) 相电压控制 PWM
- 2) 脉宽 PWM——镍氢电池，改变脉冲列的周期调频，改变脉冲的宽度和占比调压。[通过调整改变脉冲的周期和占空比达到控制充电电流的目的。](#)（占空比：在遗传理想的脉冲序列（方波）中，正脉冲的持续时间与脉冲总周期的比值。）
- 3) 随机 PWM
- 4) SPWM
- 5) 线电压控制 PWM

### 3.2 有关 PWM 的基本应用场景

- 1) 直流电机调速：占空比——直流电机两端有效电压。
- 2) LED 发光二极管的亮度调节，占空比——有效电流。
- 3) 变频调速：PWM 频率——电机转速——节能——xxx

## 3.3 STM32 的 PWM 实现原理

### 3.3.1 STM32 的 PWM 原理

STM32 单片机定时器中的计数单元对一定频率的时钟进行计数, 达到设定值改变引脚的输出状态 (高电平——低电平 e. g. ), 计数单元溢出再次改变。周期波的频率取决于时钟的频率。

相关概念:

- 1) 输出通道: PWM 波形的输出引脚, (单片机的每个定时器通常有对应的 4 个输出通道) TIMx\_CHx, x-定时器, X-定时器对应的通道。
- 2) 互补输出: 提供给交替驱动的信号。E. G. 无刷直流电机, 马达一圈——相位驱动电流改变两次——相位驱动电压改变——MOSFET or IGBT 驱动器。一个接通的时候另外一个关闭。

### 3.3.2 STM32 的 PWM 程序实现步骤

#### 【1】配置输出通道

GPIO 必须进行引脚和输出方式配置。

```
//PWM 的输出口必须是复用推挽输出
GPIO_Mode_AF_PP;
//如果连接多个接口需要打开重定向功能 Remap, 因此需要打开 AFIO 时钟 (复用时钟使能)
```

#### 【2】配置定时器 (以 TIMx 的相关寄存器)

ARR——自动重装溢出值 (控制周期的)

PSC——预分频值 (控制频率的)

例: 假设要产生 500Hz 的 PWM 波形。

```
TIM_TimeBaseStructure_Prescaler=72-1; //系统默认时钟 72MHz, 预分频 71+1 次, 得到 TIM 计数始终为 1MHz,
计数长度为 1999+1=2000
TIM_TimeBaseStructure_Period=2000-1; //必须-1, 在内部计算的时候自动加 1
//配置定时器的主要设置
TIM_DeInit(TIM2); //利用 TIM_DeInit 函数将 Timer 设置成为默认值
TIM_InternationalClockConfig (TIM2); //选择 TIM2 设置内时钟源
TIM_TimeBaseStructure.TIM_Prescaler=72; //设置预分频系数为 72
```

定义 1: 时钟分割: 在定时器钟频率 (CK\_INT) 和数字滤波器 (ETR, T1x) 使用的采样频率之间的分频比例

```
TIM_TimeBaseStructure.TIM_ClockDivision=TIM_CKD_D1V1; //在固件库中的设置, 参数表见书本 P90
```

设置计数模式 TIM2 TIM5 可以向上计数、向下计数、向上向下双向计数。

向上计数: 计数器 0——>自动加载值 (TIMx\_RR) 的周期

向下计数: 值 (TIMx\_ARR) ——>0 (并产生计数器向下溢出事件) 的周期

向上向下计数 (中央对齐模式): 0——>自动装入的值-1 (产生计数器溢出时间) ——>向下计数到 1 (产生计数器溢出事件)

```
TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up; //向上计数模式
```

设置计数溢出大小 (设置 PWM 波形的周期)

```
TIM_TimeBaseStructure.TIM_Period=1000-1;//设置计数溢出大小
```

配置应用到定时器中

```
TIM_TimeBaseInit(TIM2,&TIM_TimeBaseStructure);//配置应用到具体 TIM2 定时器中
```

配置 ARR 预装载寄存器

预装载寄存器：可写入或者读出的寄存器。——软件更新预装载寄存器的时候可以保证不更新真正操作的寄存器

影子寄存器：看不见的、无法对其进行读写操作的。——多通道输出的同步

```
TIM_ARRPreloadConfig(TIM2,DISABLE);//不需要同步输出的时候禁止预装载寄存器
```

使能定时器

```
TIM_Cmd(TIM2,ENABLE);//使能 TIMx 外设
```

### 【3】配置 PWM 模式

为了设置 TIM3\_CH3 & TIM4\_CH4 为 PWM 模式（默认是冻结的），故而通过函数 TIM\_OC1Init()——TIM\_OC4Init() 实现，原型为

```
void TIM_OC3Init(TIM_TypeDef*TIMx,TIM_OCInitTypeDef*TIM_OCInitStruct
```

以下是详细步骤

```
TIM_OCStructInit(&TimOCInitStruct); //设置默认值
TimOCInitStruct.TIM_OCMode=TIM_OCMode_PWM1; //设置输出模式为 PWM1
TimOCInitStruct.TIM_Pulse=400-1; //设置占空比 (CCRx/ARR) *100% or (TIM_Pulse/TIM_Period)*100%
TimOCInitStruct.TIM_OCPolarity=TIM_OCPolarity_HIGH; //TIM 输出可比性：高
TimOCInitStruct.TIM_OutputState=TIM_OutputState_Enable; //使能输出状态
TIM_OC3Init(TIM3,&TimOCInitStruct); //TIM3 的 CH3 输出
TIM_CtrlPWMOutput(TIM3,ENABLE); //使能
```

## 3.3.3 STM32PWM 程序实现的三个要点

### 1) PWM 的两种输出模式

模式 1 (PWM1) 和模式 2 (PWM2)，由 TIMx\_CCMRx 寄存器中的 OCxM 位确定的（110 为模式 1，111 为模式 2）

【1】110-PWM1：TIMx\_CNT=TIMx\_CCR1 通道 1 为无效电平（OC1REF=0），否则为有效电平。

【2】111-PWM2：TIMx\_CNT=TIMx\_CCR1 通道 1 为有效电平（OC1REF=0），否则为无效电平

### 2) 动态调整占空比

修改 TIM3\_CCRx (x=1, 2, 3, 4)

```
//修改占空比的函数是
void TIM_SerCompare2(TIM_TypeDef*TIUMx,uint16_t Compare2);
//对于其他函数
TIM_SetComparex(x=1,2,3,4)
```

### 3) 归纳 STM32 定时器要实现 PWM 输出的基本步骤

设置 RCC 时钟——设置 GPIO——设置 TIMx 定时器的相关寄存器——设置 TIMx 定时器的 PWM 相关寄存器——动态调整 PWM 的占空比

## 3.4 基于 PWM 的呼吸灯的实现思路

### 3.4.1 实现思路

1s 之内全灭然后按照 10 级（或）逐渐点亮，然后再完全熄灭。

首先根据不同的 PWM 周期，决定全亮的占空比值。（以 500hz 为例子，则 2000 占空比 100%，1000 占空比 50%... 同 1）

### 3.4.2 实现程序

#### 定义引脚

```

1 #include "stm32f4xx.h"
2 #include "usart.h"
3 #include "delay.h"
4 #include "stm32f4xx_gpio.h"
5 //define Premap
6 #define Premap
7 //根据宏定义控制PWM的引脚
8 #ifdef Premap//部分重定位
9     #define Ch1 GPIO_Pin_4//GPIOB
10    #define Ch2 GPIO_Pin_5//GPIOB
11    #define Ch3 GPIO_Pin_0//GPIOB
12    #define Ch4 GPIO_Pin_1//GPIOB
13 #else
14 #ifdef Premap//完全重定位
15     #define Ch1 GPIO_Pin_6//GPIOC
16     #define Ch2 GPIO_Pin_7
17     #define Ch3 GPIO_Pin_8
18     #define Ch4 GPIO_Pin_9
19 #else
20     #define Ch1 GPIO_Pin_6//GPIOA
21     #define Ch2 GPIO_Pin_7//GPIOA
22     #define Ch3 GPIO_Pin_0//GPIOB
23     #define Ch4 GPIO_Pin_1//GPIOB
24 #endif
25 #endif

```

根据 PWM 波的输出通道进行串口设置

```

27 void GPIO_Tim3PWM(u8 chx)
28 {
29     GPIO_InitTypeDef GPIO_InitStructure;
30     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA|RCC_AHB1Periph_GPIOB|RCC_AHB1Periph_GPIOC,ENABLE); //GPIO时钟
31     RCC_APB1PeriphClockCmd(GPIO_AF_TIM3,ENABLE); //使能定时器三时钟
32     switch(chx) //chx是要选择的通道号
33     {
34     case 1:
35         GPIO_InitStructure.GPIO_Pin=Ch1;//TIM1_CH1
36         break;
37     case 2:
38         GPIO_InitStructure.GPIO_Pin=Ch2;//TIM1_CH2
39         break;
40     case 3:
41         GPIO_InitStructure.GPIO_Pin=Ch3;//TIM1_CH3
42         break;
43     case 4:
44         GPIO_InitStructure.GPIO_Pin=Ch4;//TIM1_CH4
45         break;
46     }
47     GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF; //复用模式
48     GPIO_InitStructure.GPIO_OType=GPIO_OType_PP; //推挽输出
49     GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
50     //根据Pwm通道自适应定义引脚

```

根据 PWM 通道自适应定义引脚

```

50 //根据PWM通道自适应定义引脚
51 #ifndef Premap
52     GPIO_Init(GPIOB,&GPIO_InitStructure);
53 #else
54 #ifndef Premap
55     GPIO_Init(GPIOC,&GPIO_InitStructure);
56 #else
57     switch (chx)
58     {
59     case 1:
60         GPIO_Init(GPIOA,&GPIO_InitStructure)
61         break;
62     case 2:
63         GPIO_Init(GPIOA,&GPIO_InitStructure)
64         break;
65     case 3:
66         GPIO_Init(GPIOB,&GPIO_InitStructure)
67         break;
68     case 4:
69         GPIO_Init(GPIOB,&GPIO_InitStructure)
70         break;
71     }
72 #endif
73 #endif
74 }

```

输出引脚重定位，remap 为重定位函数，0 为无重定位，1 为部分重定位，2 为全部重定位。

```

75 //输出引脚重定位，remap为重定位函数，0为无重定位，1为部分重定位，2为全部重定位。
76 void TIM3PinReMap(u8 remap)
77 {
78     switch (remap)
79     {
80     case 0:
81         break;
82     case 1:
83         //TIM2输出引脚部分重定位
84         break;
85     case 2:
86         //TIM3输出引脚部分重定位
87         break;
88     }
89 }

```

初始化 TIM3，设置 TIM3 的 ARR（自动溢出装置，控制周期的）和 PSC（预分频值，定时器的周期）。

```

90 void TIM_Init(TIM_TypeDef*TIMx,u16 arr,u16 psc)
91 {
92     TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
93     TIM_TimeBaseStructure.TIM_Period=arr-1;//设置自动重装载值
94     TIM_TimeBaseStructure.TIM_Prescaler=psc-1;//设置预分频值
95     TIM_TimeBaseStructure.TIM_ClockDivision=TIM_CKD_DIV1;//设置时钟分割:TIM_CKD_DIV1=0的时候PWM不延时
96     TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up;//向上计数模式
97     TIM_TimeBaseInit(TIMx,&TIM_TimeBaseStructure);//根据制定的参数初始化TIMx
98     //TIM_ARRPreloadConfig(TIMx,DISABLE);
99     //禁止ARR预装载缓冲器
100     TIM_ARRPreloadConfig(TIMx,ENABLE);//使能ARR预装载缓冲器
101 }

```

设置 TIM 的 PWM 模式，使能 TIM 的输出

参数：

TIMx=定时器 x，chx=选定的通道，H2L=高电平还是低电平（1 高 0 低），pulse=脉冲宽度  
TIMx 的输出通道必须用相应的 TIM\_OCxInit（）函数加以设定和使能  
对 2，3，4 加以同样的设置

```

102 void TIM_FWMMode(TIM_TypeDef*TIMx,u8 chx,u8 H2L,u16 pulse)
103 {
104     TIM_OCInitTypeDef TIM_OCInitStructure;
105     switch (chx)
106     {
107     case 1:
108         if(H2L)
109             TIM_OCInitStructure.TIM_OCMode=TIM_OCMode_PWM1;
110         //选择定时器模式,TIM脉冲宽度调制模式1,相等时转换为低
111         else
112             TIM_OCInitStructure.TIM_OCMode=TIM_OCMode_PWM2;
113         //选择定时器模式,TIM脉冲宽度调制模式2,相等时转换为高
114         TIM_OCInitStructure.TIM_OutputState=TIM_OutputState_Enable;//比较输出使能
115         TIM_OCInitStructure.TIM_Pulse=pulse-1;
116         TIM_OCInitStructure.TIM_OCPolarity=TIM_OCPolarity_High;//输出极性:TIM输出比较极性高
117         TIM_OC1Init(TIMx,&TIM_OCInitStructure);//根据TIM_OCInitStruct中指定的参数初始化外设TIMx
118         TIM_OC1PreloadConfig(TIMx,TIM_OCPreload_Enable);//使能TIMx在CCR1上的预装载寄存器
119         TIM_CtrlPWMOutputs(TIMx,ENABLE);//设置TIMx的PWM输出为使能
120         TIM_ARRPreloadConfig(TIMx,ENABLE);//使能TIMx在ARR上的预装载寄存器
121         TIM_Cmd(TIMx,ENABLE);//使能TIMx外设
122         break;
123     //后文同理
124     case 2:
125         if(H2L)
126             TIM_OCInitStructure.TIM_OCMode=TIM_OCMode_PWM1;
127         else

```

有关延时函数的延时代码

```

170 void delay_nums(u16 time)
171 {
172     u16 i=0;
173     while(time--)
174     {
175         i=12000;
176         while(i--);
177     }

```

主函数部分

```

179 int main(void)
180 {
181     short int kcnt=2000;
182     SystemInit();
183     GPIO_Tim3PWM(3);
184     TIM3PinReMap(2);
185     TIM_Init(TIM3,2000,72);
186     GPIO_Tim3PWM(4);
187     TIM_FWMMode(TIM3,4,0,kcnt);
188     TIM_FWMMode(TIM3,4,0,kcnt);
189
190     while (1)
191     {
192         for(kcnt=2001;kcnt>0;kcnt=kcnt-200)
193         {
194             TIM_SetCompare3(TIM3,kcnt);//改变占空比
195             TIM_SetCompare3(TIM3,kcnt);
196             delay_nums(100);
197         }
198         for(kcnt=1;kcnt<=2001;kcnt=kcnt+200)
199         {
200             TIM_SetCompare3(TIM3,kcnt);
201             TIM_SetCompare3(TIM3,kcnt);
202             delay_nums(100);
203         }
204     }

```

以下是完整代码

```

#include "stm32f4xx.h"
#include "usart.h"
#include "delay.h"
#include "stm32f4xx_gpio.h"

//define Premap
#define Fremap
//根据宏定义控制 PWM 的引脚

#ifdef Premap//部分重定位
#define Ch1 GPIO_Pin_4//GPIOB
#define Ch2 GPIO_Pin_5//GPIOB
#define Ch3 GPIO_Pin_0//GPIOB
#define Ch4 GPIO_Pin_1//GPIOB
#else

```

```
#ifdef Fremap//完全重定位
#define Ch1 GPIO_Pin_6//GPIOC
#define Ch2 GPIO_Pin_7
#define Ch3 GPIO_Pin_8
#define Ch4 GPIO_Pin_9
#else
#define Ch1 GPIO_Pin_6//GPIOA
#define Ch2 GPIO_Pin_7//GPIOA
#define Ch3 GPIO_Pin_0//GPIOB
#define Ch4 GPIO_Pin_1//GPIOB
#endif
#endif

void GPIO_Tim3PWM(u8 chx)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA|RCC_AHB1Periph_GPIOB|RCC_AHB1Periph_GPIOC,ENABLE); //GPIO 时钟
    RCC_APB1PeriphClockCmd(GPIO_AF_TIM2,ENABLE);
    RCC_APB1PeriphClockCmd(GPIO_AF_TIM3,ENABLE); //使能定时器三时钟
    switch(chx) //chx 是要选择的通道号
    {
        case 1:
            GPIO_InitStructure.GPIO_Pin=Ch1; //TIM1_CH1
            break;
        case 2:
            GPIO_InitStructure.GPIO_Pin=Ch2; //TIM1_CH2
            break;
        case 3:
            GPIO_InitStructure.GPIO_Pin=Ch3; //TIM1_CH3
            break;
        case 4:
            GPIO_InitStructure.GPIO_Pin=Ch4; //TIM1_CH4
            break;
    }
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF; //复用模式
    GPIO_InitStructure.GPIO_OType=GPIO_OType_PP; //推免输出
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    //根据 PWM 通道自适应定义引脚
#ifdef Premap
    GPIO_Init(GPIOB,&GPIO_InitStructure);
#else
#ifdef Fremap
    GPIO_Init(GPIOC,&GPIO_InitStructure);
#else
    switch (chx)
```



```

{
    case 1:
        GPIO_Init(GPIOA,&GPIO_InitStructure)
        break;
    case 2:
        GPIO_Init(GPIOA,&GPIO_InitStructure)
        break;
    case 3:
        GPIO_Init(GPIOB,&GPIO_InitStructure)
        break;
    case 4:
        GPIO_Init(GPIOB,&GPIO_InitStructure)
        break;
}
#endif
#endif
}
//输出引脚重定位, remap 为重定位函数, 0 为无重定位, 1 为部分重定位, 2 为全部重定位。
void TIM3PinReMap(u8 remap)
{
    switch (remap)
    {
        case 0:
            break;
        case 1:
            //TIM2 输出引脚部分重定位
            break;
        case 2:
            //TIM3 输出引脚部分重定位
            break;
    }
}
void TIM_Init(TIM_TypeDef*TIMx,u16 arr,u16 psc)
{
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    TIM_TimeBaseStructure.TIM_Period=arr-1;//设置自动重装载值
    TIM_TimeBaseStructure.TIM_Prescaler=psc-1;//设置预分频值
    TIM_TimeBaseStructure.TIM_ClockDivision=TIM_CKD_DIV1;//设置时钟分割:TIM_CKD_DIV1=0 的时候 PWM
不延时
    TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up;//向上计数模式
    TIM_TimeBaseInit(TIMx,&TIM_TimeBaseStructure);//根据制定的参数初始化 TIMx
    //TIM_ARRPreloadConfig(TIMx,DISABLE);
    //禁止 ARR 预装载缓冲器
    TIM_ARRPreloadConfig(TIMx,ENABLE);//使能 ARR 预装载缓冲器

```



```

}
void TIM_PWMMode(TIM_TypeDef*TIMx,u8 chx,u8 H2L,u16 pulse)
{
    TIM_OCInitTypeDef TIM_OCInitStructure;
    switch (chx)
    {
        case 1:
            if(H2L)
                TIM_OCInitStructure.TIM_OCMode=TIM_OCMode_PWM1;
            //选择定时器模式,TIM 脉冲宽度调制模式 1,相等时转换为低
            else
                TIM_OCInitStructure.TIM_OCMode=TIM_OCMode_PWM2;
            //选择定时器模式,TIM 脉冲宽度调制模式 2,相等时转换为高
            TIM_OCInitStructure.TIM_OutputState=TIM_OutputState_Enable;//比较输出使能
            TIM_OCInitStructure.TIM_Pulse=pulse-1;
            TIM_OCInitStructure.TIM_OCPolarity=TIM_OCPolarity_High;//输出极性:TIM 输出比较极性高
            TIM_OC1Init(TIMx,&TIM_OCInitStructure);//根据 TIM_OCInitStruct 中指定的参数初始化外设 TIMx
            TIM_OC1PreloadConfig(TIM1,TIM_OCPreload_Enable);//使能 TIMx 在 CCR1 上的预装载寄存器
            TIM_CtrlPWMOutputs(TIM1,ENABLE);//设置 TIMx 的 PWM 输出为使能
            TIM_ARRPreloadConfig(TIM1,ENABLE);//使能 TIMx 在 ARR 上的预装载寄存器
            TIM_Cmd(TIMx,ENABLE);//使能 TIMx 外设
            break;
        //后文同理
        case 2:
            if(H2L)
                TIM_OCInitStructure.TIM_OCMode=TIM_OCMode_PWM1;
            else
                TIM_OCInitStructure.TIM_OCMode=TIM_OCMode_PWM2;
            TIM_OCInitStructure.TIM_OutputState=TIM_OutputState_Enable;
            TIM_OCInitStructure.TIM_Pulse=pulse-1;
            TIM_OCInitStructure.TIM_OCPolarity=TIM_OCPolarity_High;
            TIM_OC2Init(TIMx,&TIM_OCInitStructure);
            TIM_OC2PreloadConfig(TIM2,TIM_OCPreload_Enable);
            TIM_CtrlPWMOutputs(TIM2,ENABLE);
            TIM_ARRPreloadConfig(TIM2,ENABLE);
            TIM_Cmd(TIMx,ENABLE);
            break;
        case 3:
            if(H2L)
                TIM_OCInitStructure.TIM_OCMode=TIM_OCMode_PWM1;
            else
                TIM_OCInitStructure.TIM_OCMode=TIM_OCMode_PWM2;
            TIM_OCInitStructure.TIM_OutputState=TIM_OutputState_Enable;
            TIM_OCInitStructure.TIM_Pulse=pulse-1;

```

```

        TIM_OCInitStructure.TIM_OCPolarity=TIM_OCPolarity_High;
        TIM_OC3Init(TIMx,&TIM_OCInitStructure);
        TIM_OC3PreloadConfig(TIM3,TIM_OCPreload_Enable);
        TIM_CtrlPWMOutputs(TIM3,ENABLE);
        TIM_ARRPreloadConfig(TIM3,ENABLE);
        TIM_Cmd(TIMx,ENABLE);
        break;
    case 4:
        if(H2L)
            TIM_OCInitStructure.TIM_OCMode=TIM_OCMode_PWM1;
        else
            TIM_OCInitStructure.TIM_OCMode=TIM_OCMode_PWM2;
        TIM_OCInitStructure.TIM_OutputState=TIM_OutputState_Enable;
        TIM_OCInitStructure.TIM_Pulse=pulse-1;
        TIM_OCInitStructure.TIM_OCPolarity=TIM_OCPolarity_High;
        TIM_OC4Init(TIMx,&TIM_OCInitStructure);
        TIM_OC4PreloadConfig(TIM4,TIM_OCPreload_Enable);
        TIM_CtrlPWMOutputs(TIM4,ENABLE);
        TIM_ARRPreloadConfig(TIM4,ENABLE);
        TIM_Cmd(TIMx,ENABLE);
        break;
    }
}
//有关延时函数的延时代码
void delay_nums(u16 time)
{
    u16 i=0;
    while(time--)
    {
        i=12000;
        while(i--);
    }
}
//主函数
int main(void)
{
    short int kcnt=2000;
    SystemInit();
    GPIO_Tim3PWM(3);
    TIM3PinReMap(2);
    TIM_Init(TIM3,2000,72);
    GPIO_Tim3PWM(4);
    TIM_PWMMode(TIM3,4,0,kcnt);
    TIM_PWMMode(TIM3,4,0,kcnt);
    while (1)
    {

```

```
    for(kcnt=2001;kcnt>0;kcnt=kcnt-200)
    {
        TIM_SetCompare3(TIM3,kcnt);//改变占空比
        TIM_SetCompare3(TIM3,kcnt);
        delay_nums(100);
    }
    for(kcnt=1;kcnt<=2001;kcnt=kcnt+200)
    {
        TIM_SetCompare3(TIM3,kcnt);
        TIM_SetCompare3(TIM3,kcnt);
        delay_nums(100);
    }
}
```