目录

第一章 实验环境安装

1.1 开发环境简介

RVMDK 源自德国的 KEIL 公司，是 RealView MDK 的简称，RealView MDK 集成了业内最领先的技术，包括 μ Vision5 集成开发环境与 RealView 编译器（俗称 Keil5）。支持 ARM7、ARM9 和最新的 Cortex-M3 核处理器，自动配置启动代码，集成 Flash 烧写模块，强大的 Simulation 设备模拟，性能分析等功能，与 ARM 之前的工具包 ADS 等相比，RealView 编译器的最新版本可将性能改善超过 20%。

1.2 开发环境安装

1.2.1 温馨提示

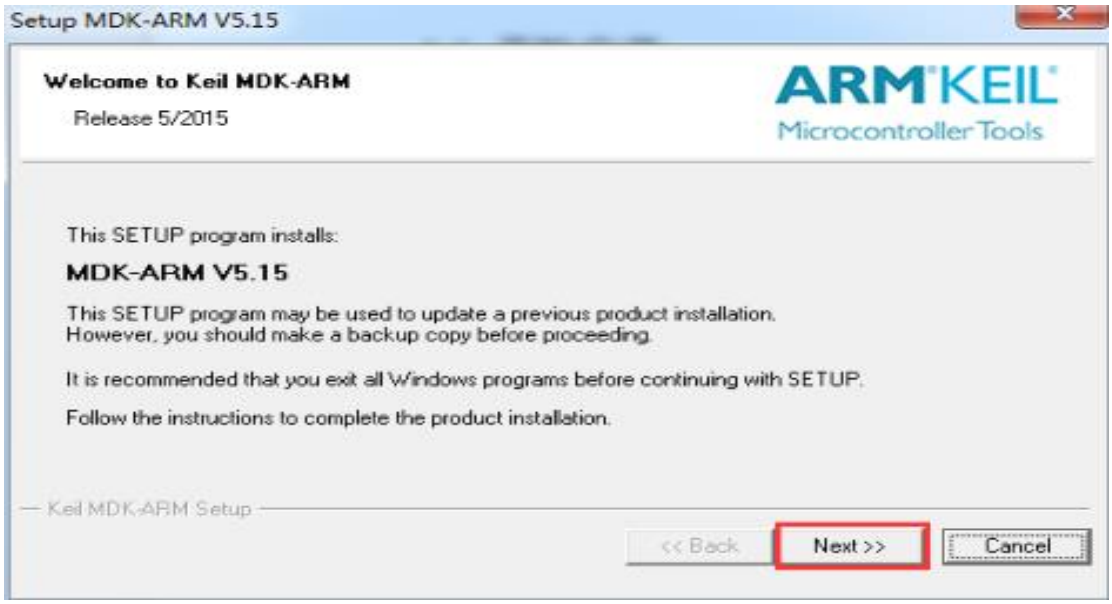
1、安装路径不能带中文，必须是英文路径
2、安装目录不能跟 51 的 KEIL 或者 KEIL4 冲突，三者目录必须分开
3、KEIL5 的安装比起 KEIL4 多了一个步骤，必须添加 MCU 库（安装芯片包），不然没法使用。

1.2.2 获取 KEIL5 的安装包

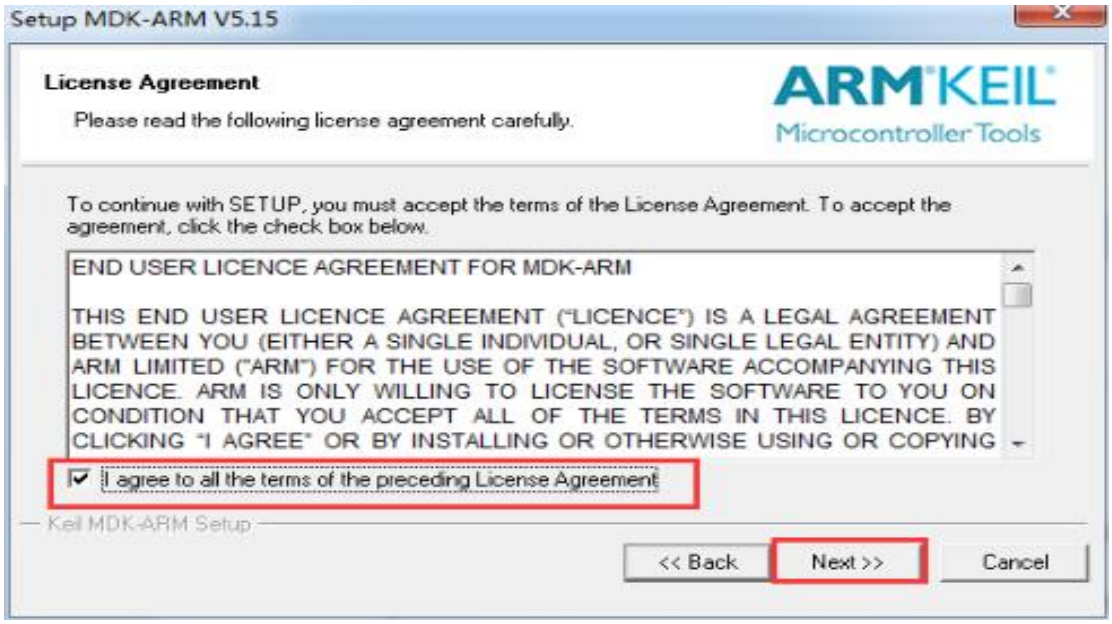
KEIL 的官网下载：https://www.keil.com/download/product/

1.2.3 安装 KEIL5

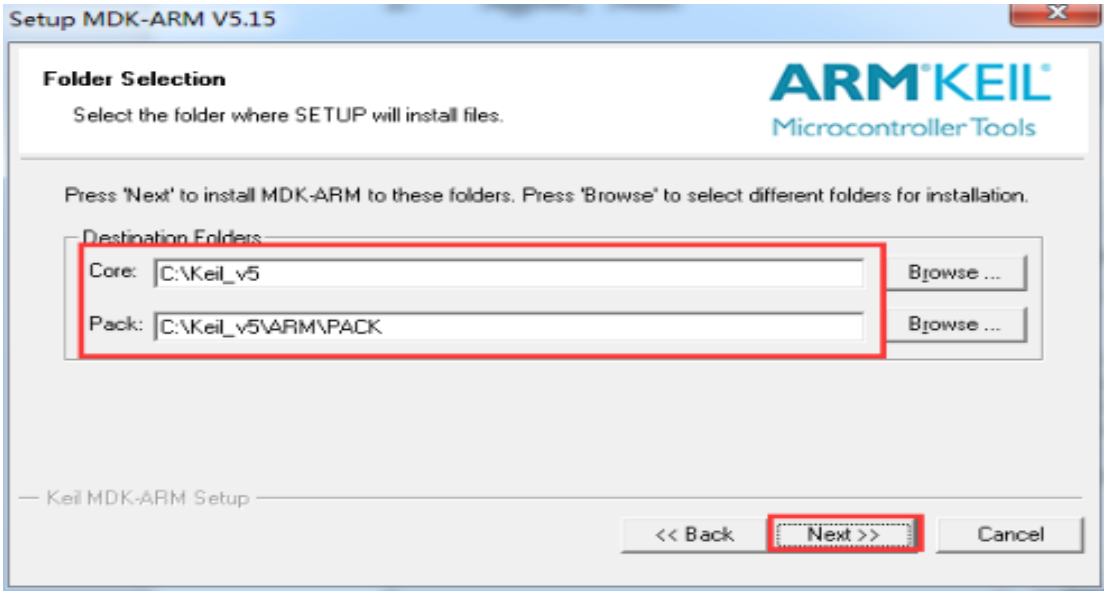
1 双击 KEIL5 的安装包，开始安装，点击“next”；



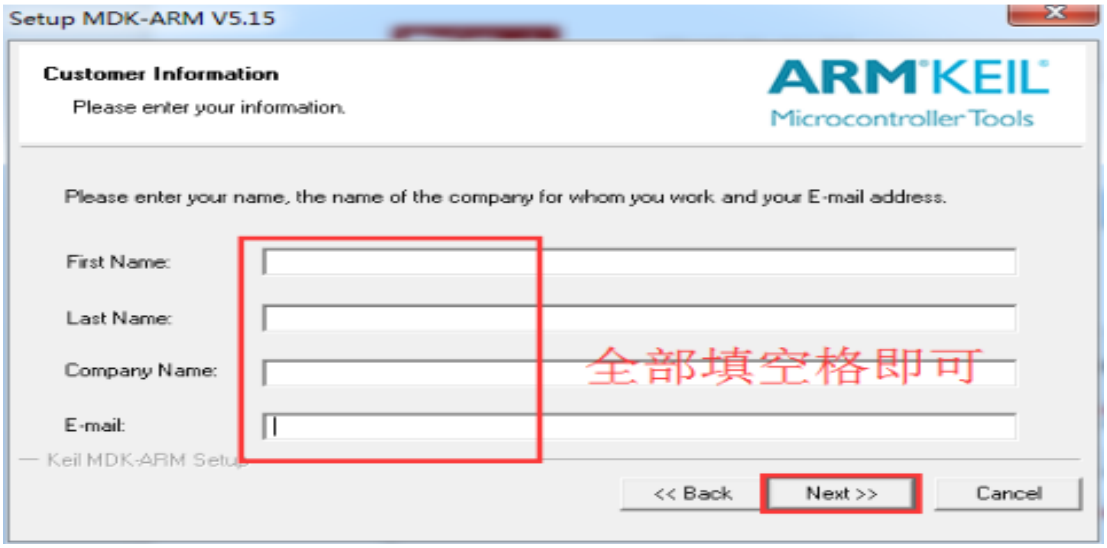
2 点击“Agree”，单击“next”；（红框中标识的）



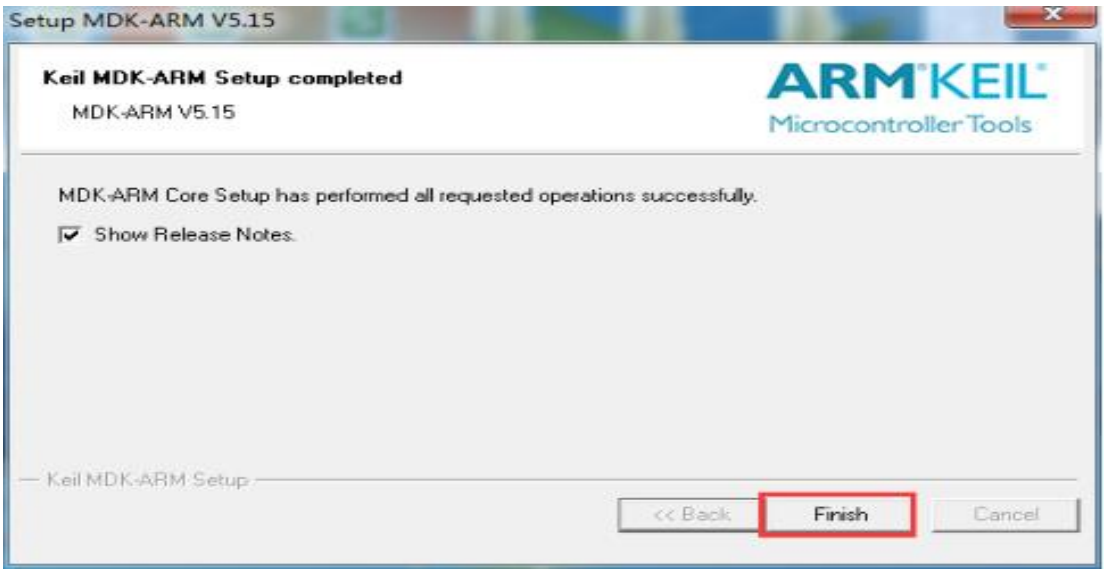
3 选择安装路径，不能带中文！切记，切记！单击”next”；



4 填写用户信息，任意字符，或全部填空格（键盘的 SPACE 键）即可

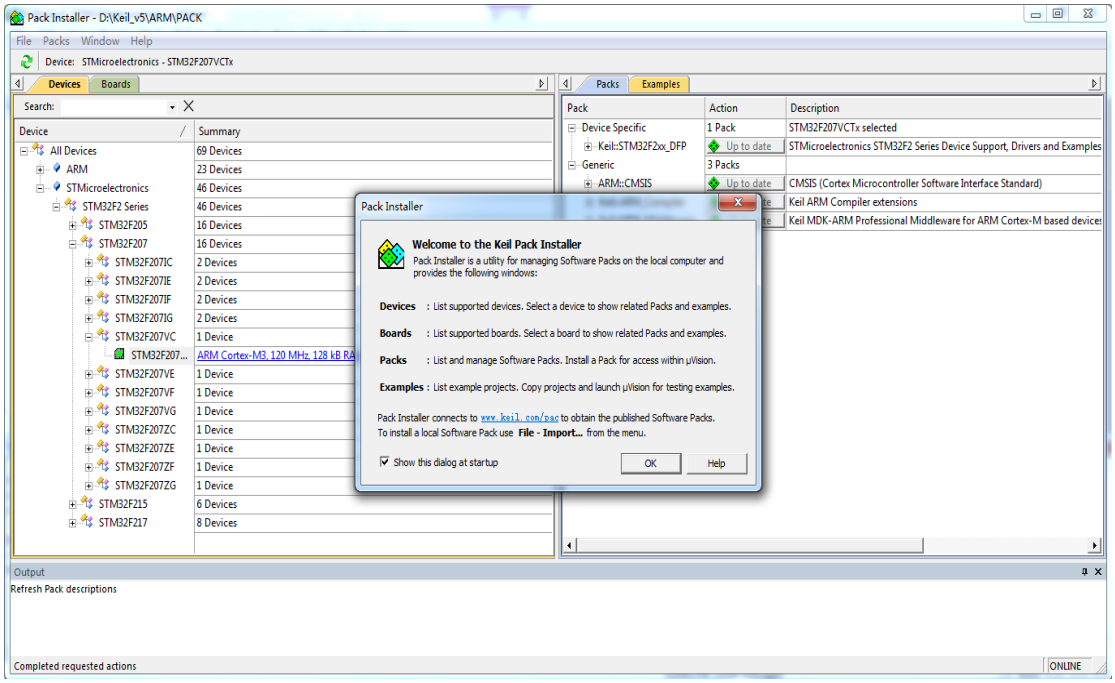


5 点击“Finish”，安装完毕。

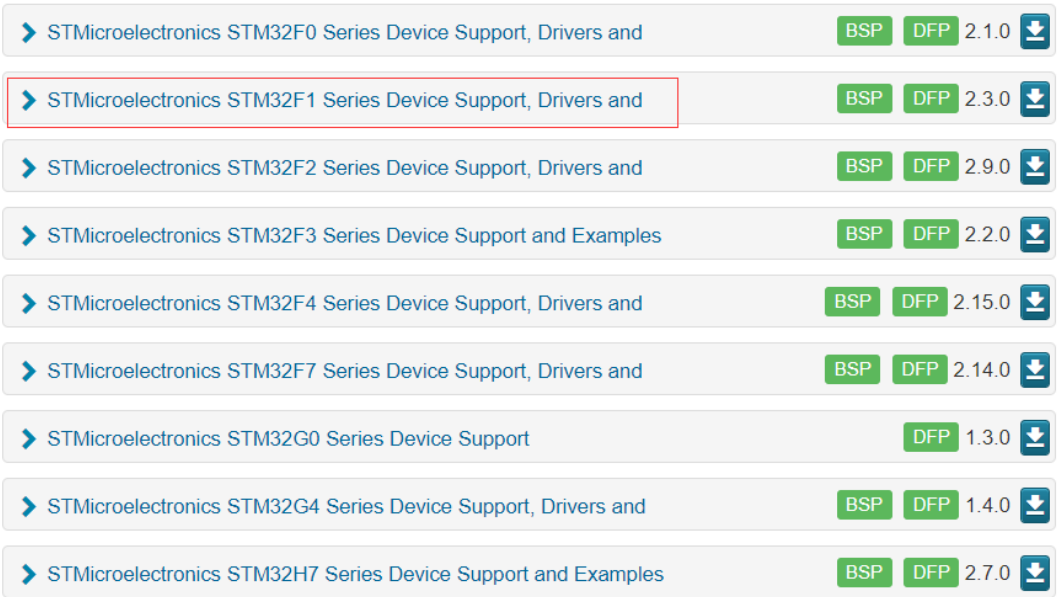


1.2.4 安装 STM32 芯片包

KEIL5 不像 KEIL4 那样自带了很多厂商的 MCU 型号，KEIL5 需要自己安装。把下面弹出的界面关掉，我们直接去 keil 的官网下载：http://www.keil.com/dd2/pack/。



在官网中找到 STM32F1 这个系列的包下载到本地电脑，具体下载哪个系列的根据使用的型号下载即可，这里只下载实验需要使用的 F1 或者 F2 这个系列的包。

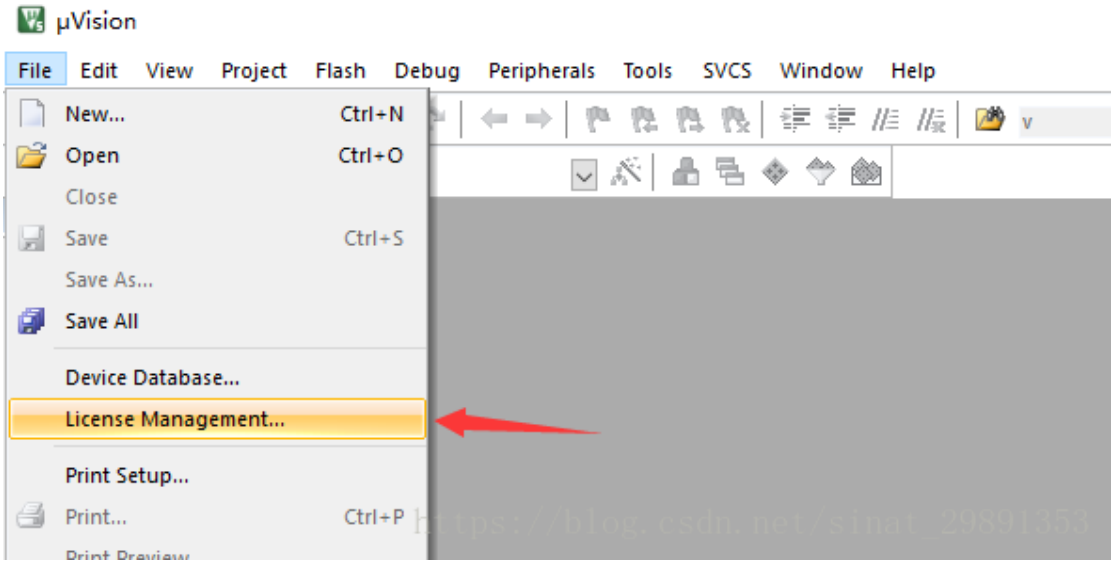


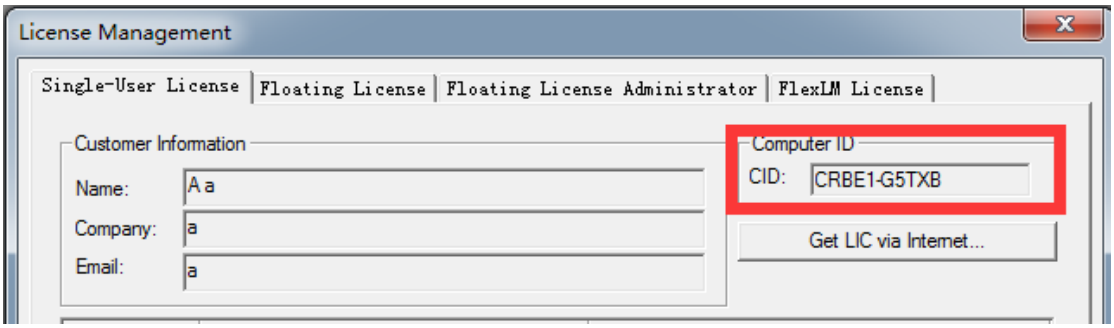
把下载好的包双击安装即可，安装路径选择跟 KEIL5 一样的安装路径，安装成功之后，在 KEIL5 的 Pack Installer 中就可以看到我们安装的包，以后我们新建工程的时候，就有单片机的型号可选。

1.3 安装环境破解

KEIL5 安装好后，可以直接使用。但是由于 Keil 是商业版软件，没有授权序列号的情况下只能试用，代码不得超过 32K。对于学习来说，可以通过网络找到用于学习目的序列号生成工具，下面讲解如何生成序列号。

1 在 KEIL5 中打开注册管理窗口（File -> License Management），并复制 CID（备用）

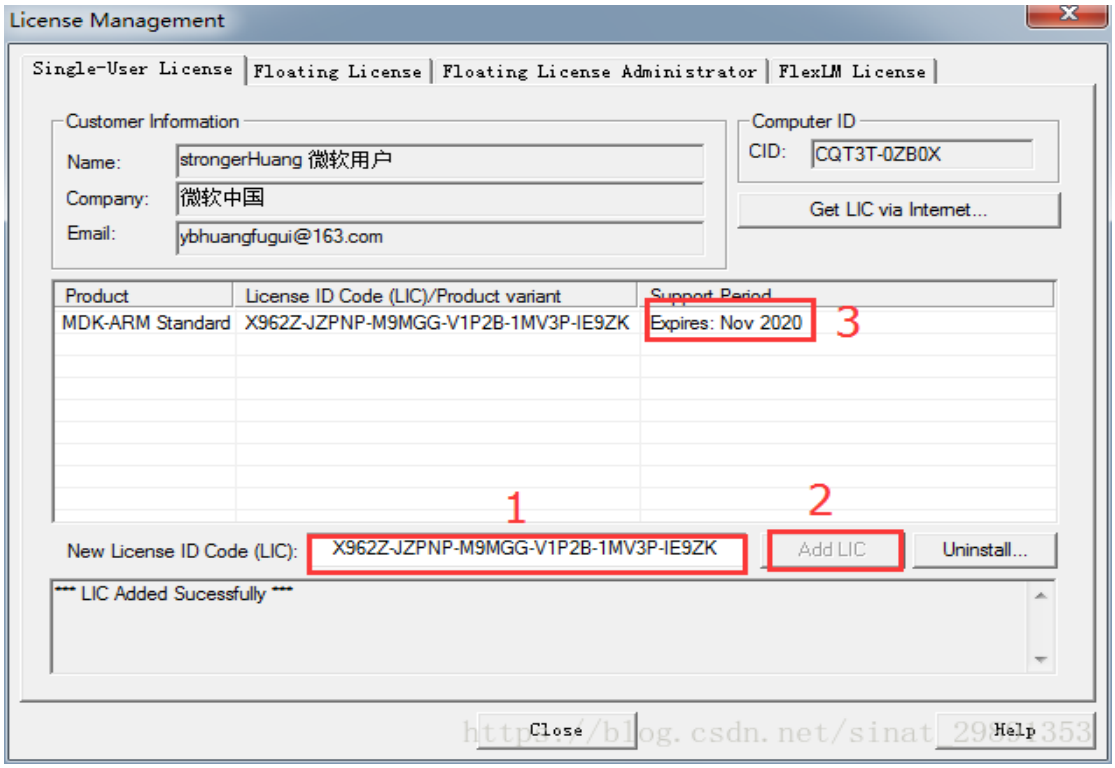




2. 打开注册机，(1)粘贴上面复制的 CID，(2)目标选择 ARM，(3)生成注册码，(4)复制注册码。

注意：注册机在解压出来的时候需要关闭所有的电脑管家、杀毒软件，包括 Windows 的病毒检测。

3 回到注册界面：(1)拷贝上面图中第四个框中生成的“注册码”，并粘贴到下图的第一个红框中(2)点击“Add LIC”，(3)看见显示信息说明，日期晚于当前日期，软件可正常使用，否则重复前述步骤，重新生成注册码。



第二章 实验例程讲解篇

2.1 开发环境实验

2.1.1 实验目的

通过本实验，掌握开发环境的使用方法，为后续实验打下基础。

2.1.2 实验要求

1、掌握工程新建的方法
2、掌握工程设置方法
3、掌握调试方法

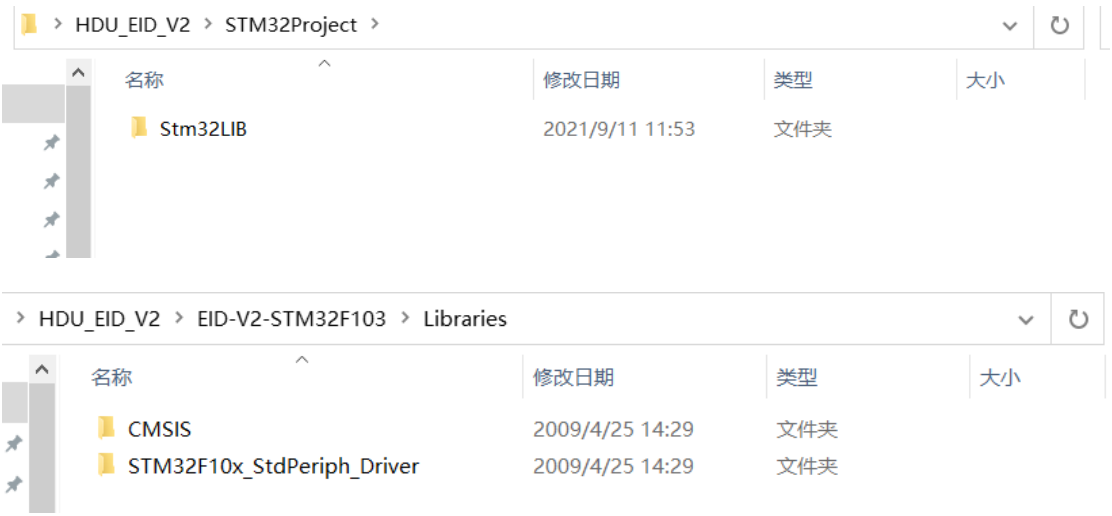
2.1.3 新建工程

1）. 在电脑中想要建立工程的地方先新建一个文件夹。这里我在 E 盘下新建了一个文件夹，命名为 STM32Project。

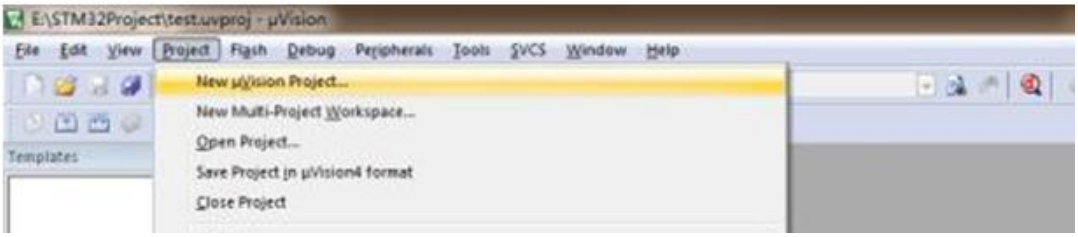
2）. 在新建的 STM32Project 文件夹下再新建一个文件夹，命名为 Stm32LIB，然后把 DRIVER

内的固件库放在文件夹中，方便查找进行复制。

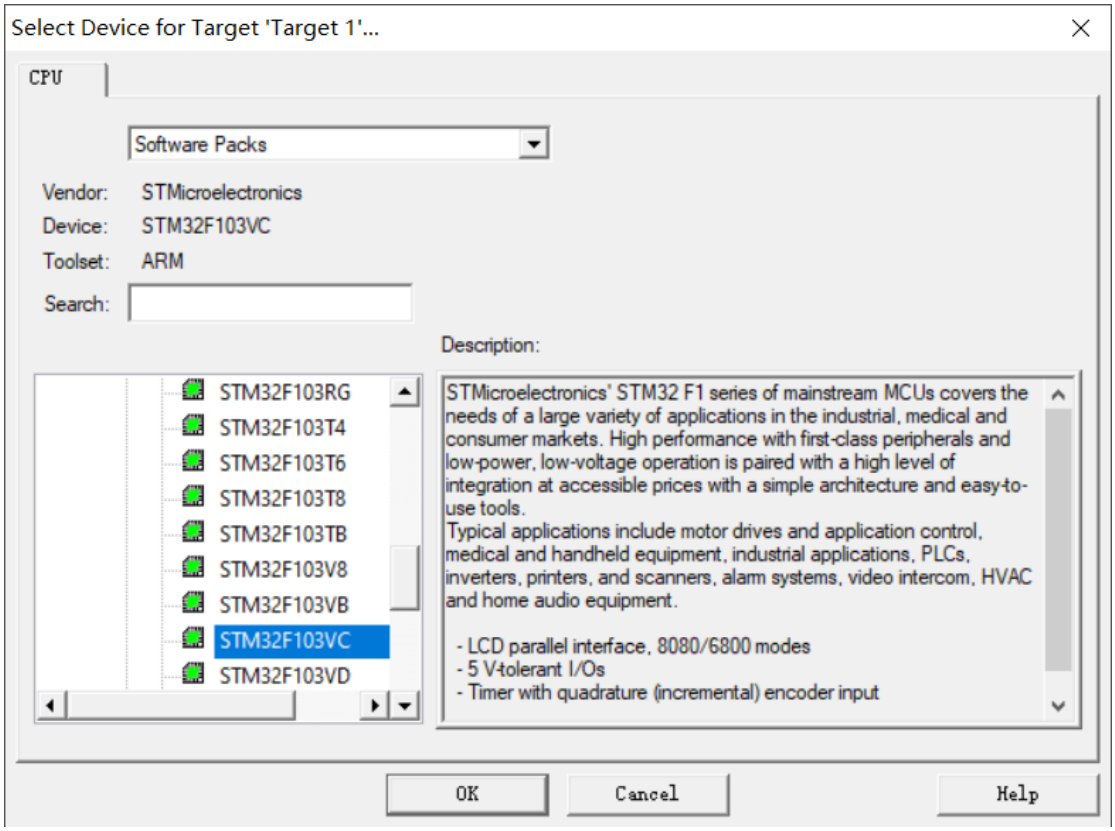
注：固件库可以从官网下载，也可以从例程中获取



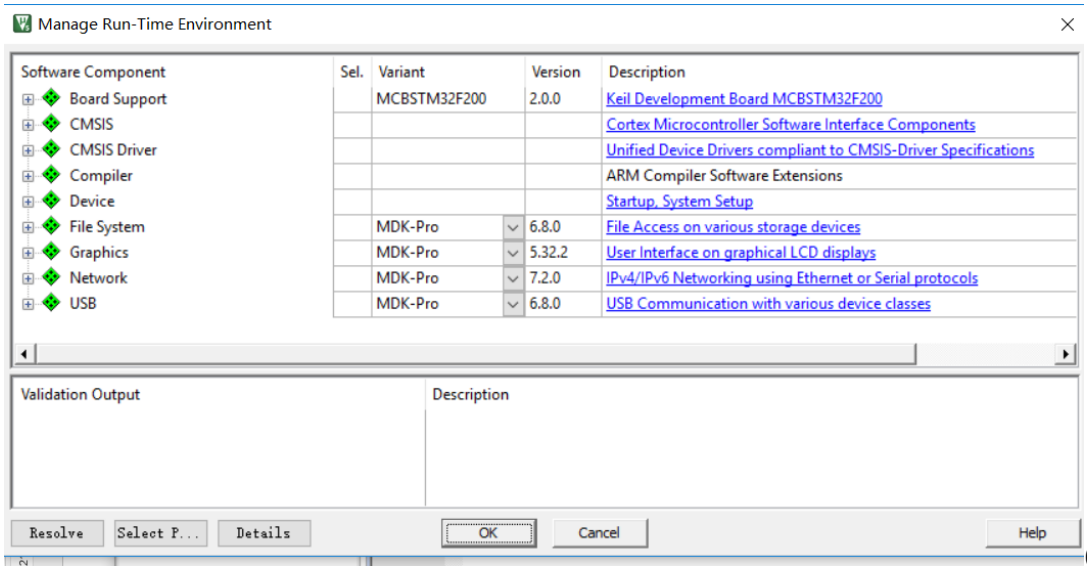
3）．打开Keil uVision5软件，在软件的菜单栏中找到Project——>New uVision Project...，单击，找到上述新建的文件夹STM32Project，对新建的工程进行命名，这里命名为test，点击保存。



4）．此时软件弹出一个选择芯片的窗口：“Select Device for Target ‘Target 1’...”，在左侧的芯片库中找到你要使用的STM32系列的芯片，这里选择的是STMicroelectronics——>STM32F1 Series——>STM32F103——>STM32F103VC，单击OK。



5）. 此时弹出一个窗口：“Manage Run-Time Environment”，这个窗口在第二个建工程的方法中会使用到，这里先直接点击OK，跳过。

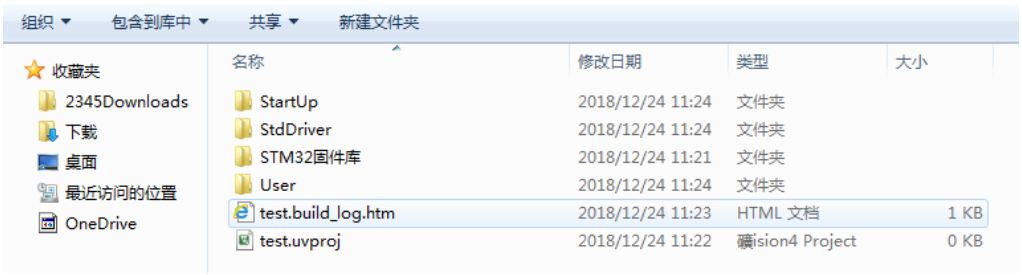


6）现在开始添加相关文件及文件夹（非必须，为方便日后查看，建立下述文件夹）。

user：放置用户自己编写的相关文件

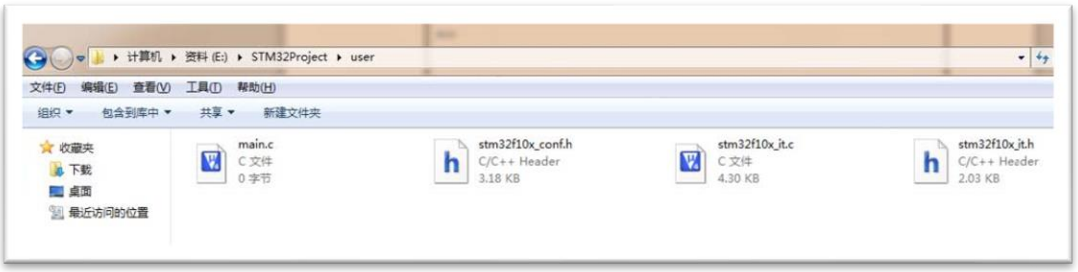
startup：放置启动单片机的汇编文件

StdDriver：放置外设操作的驱动文件



再向这四个文件夹中添加固件库中的相关文件：

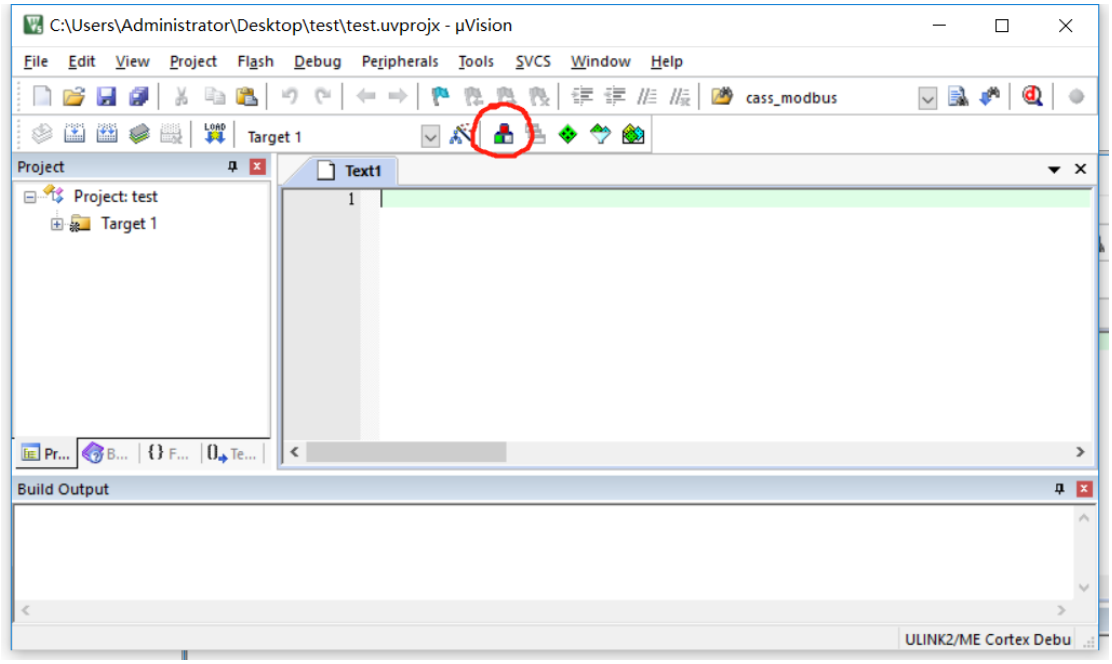
user目录：stm32f10x_conf.h、stm32f10x_it.c、stm32f10x_it.h，同时新建一个文本文档，重命名为main.c。



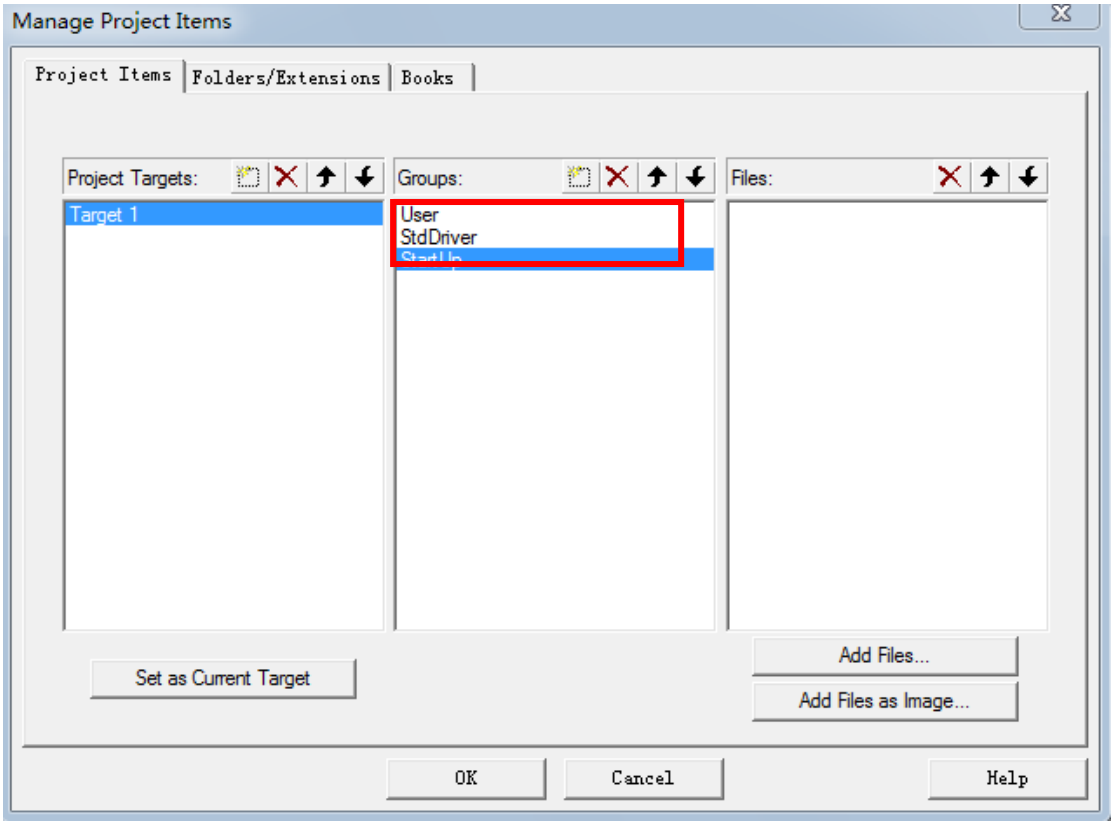
startup目录： 复制STM32LIB\CMSIS\Core\CM3到startup目录。

StdDriver目录： STM32固件库\STM32F1x_StdPeriph_Driver的src文件夹内的内容全部复制粘贴到driver文件夹中。

7）.文件复制结束后，现在回到Keil软件界面，对新建的工程进行管理：在软件上方的工具栏中找到如下图标识的图标，单击（工程右键出现菜单栏，选择Manage Project Items，图标和工具栏图标一致）

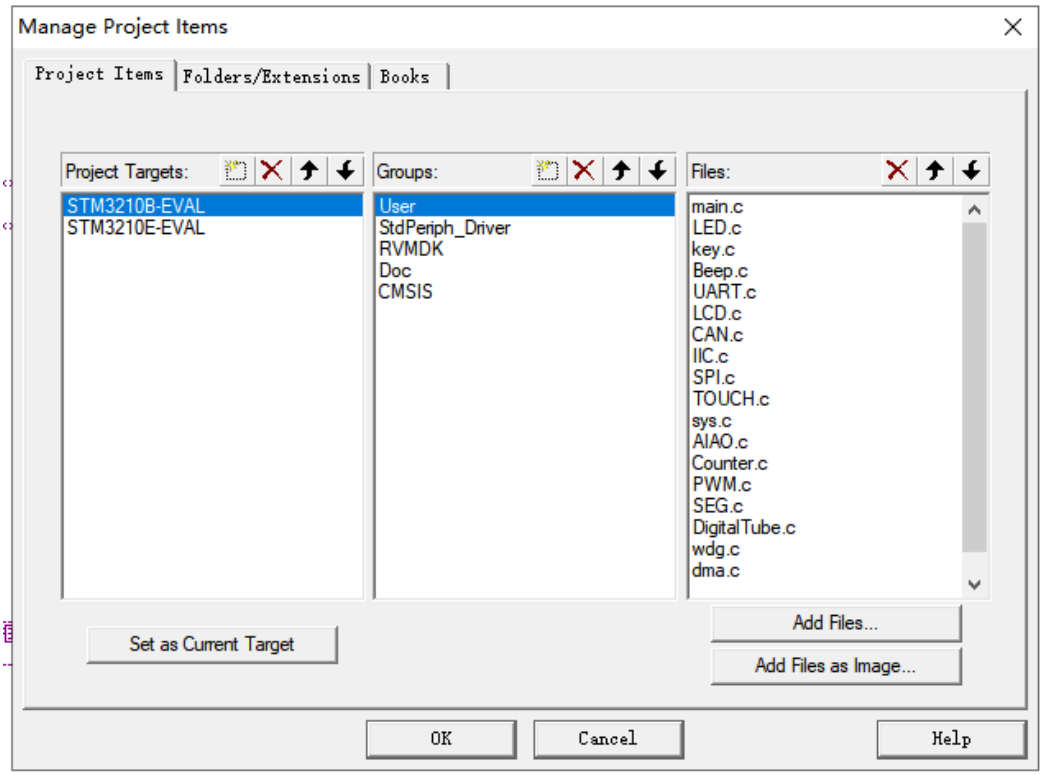


8）. 进行完第7步操作后进入到Manage Project Items界面。在这个界面里，我们可以对该工程进行管理，通过Project Target的删除和新建，可以对该工程进行重命名，这里命名为test。删除Groups中原有的一个文件，新建第6步操作中在工程文件夹中添加的对应的3个文件名，分别为user、startup和StdDriver。

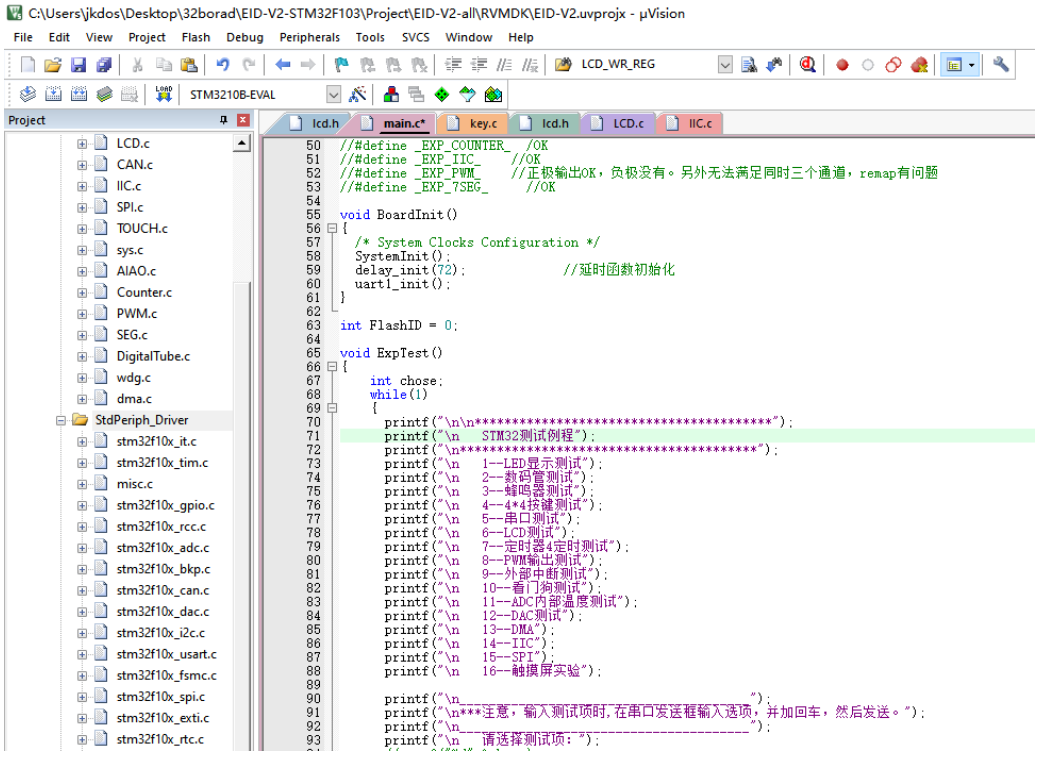


9）. 分别对上述的user、startup和StdDriver 3个文件进行Add Files操作。所要添加的文件分别为第6步操作中对应文件夹中的相关对应文件，点击OK。

注意：添加文件时只添加**.c**后缀的文件，**.h**后缀的文件不需要添加，只有**startup**中要添加Stm32LIB\CMSIS\Core\CM3\startup\arm\startup_stm32f10x_md.s后缀文件，StdDriver中添加的文件在对应文件夹StdDriver中。

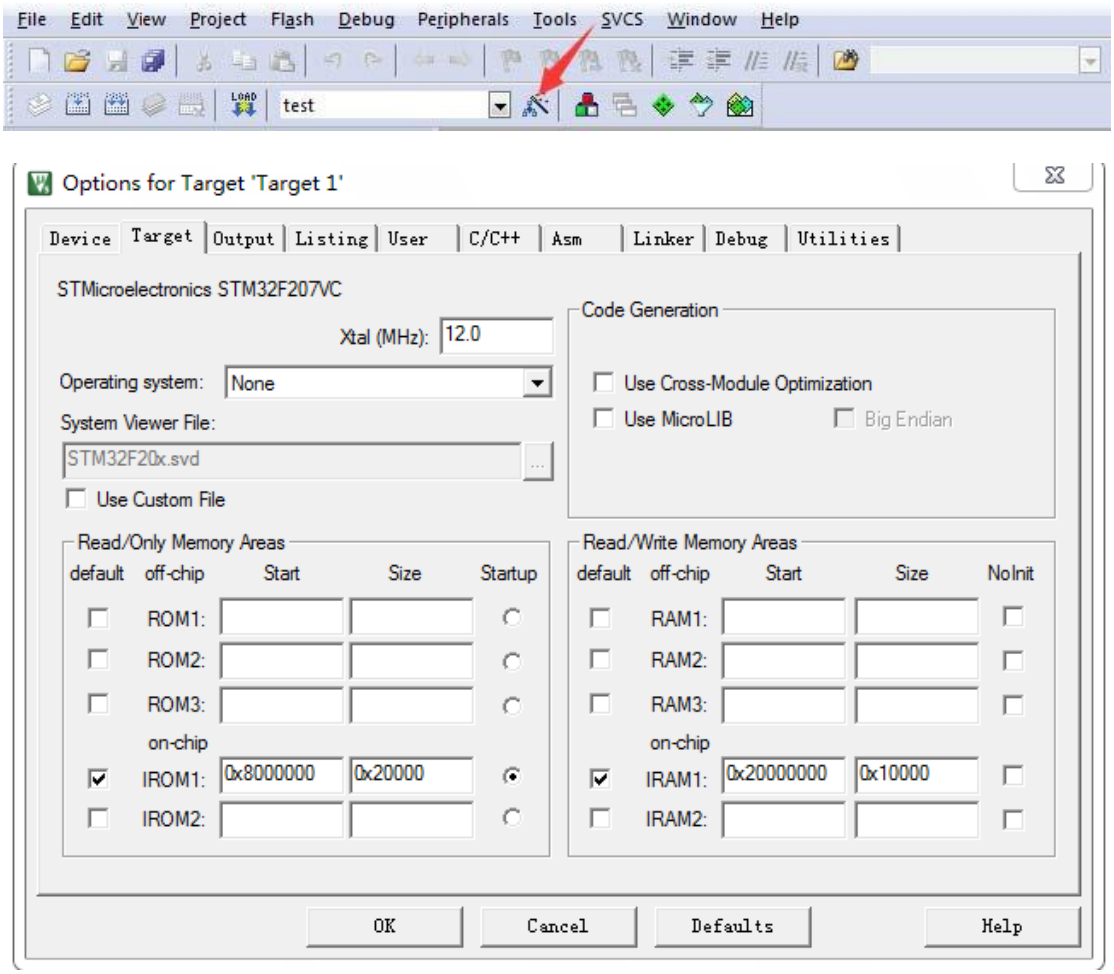


完成后，可以看到刚刚添加的文件已经进入到工程文件列表中。

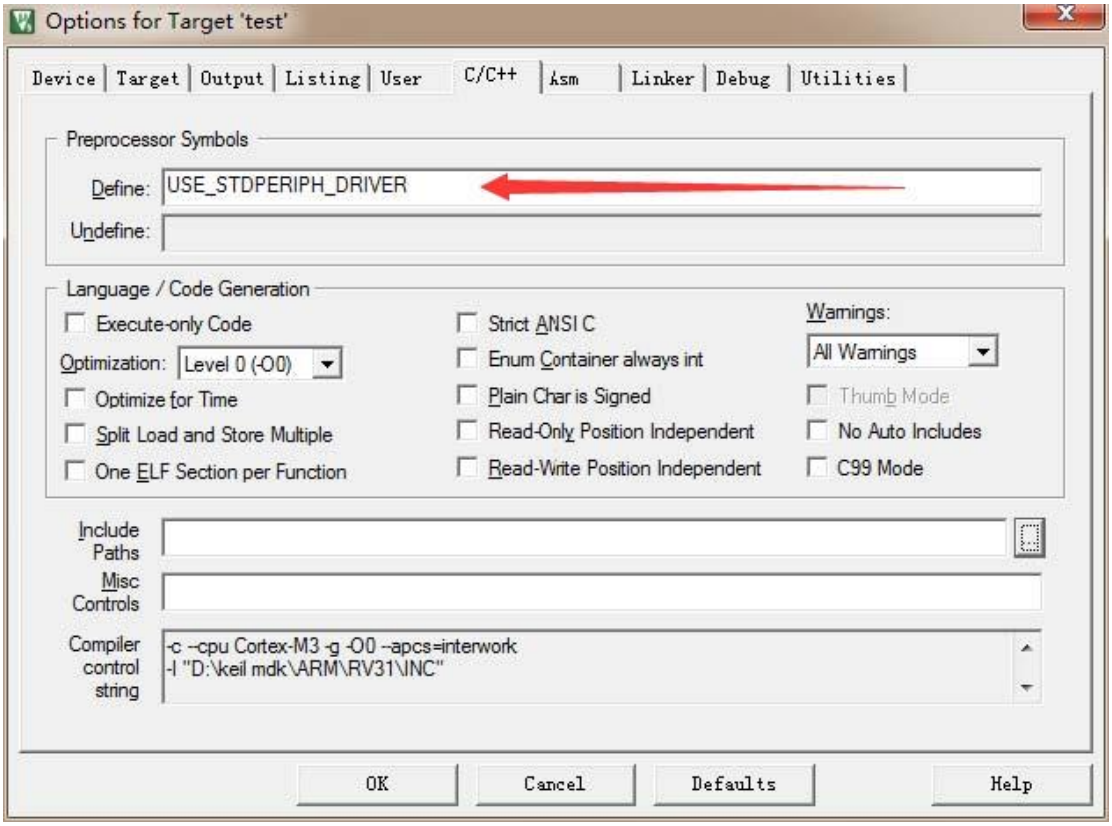


10）. 配置编译相关的设置选项。

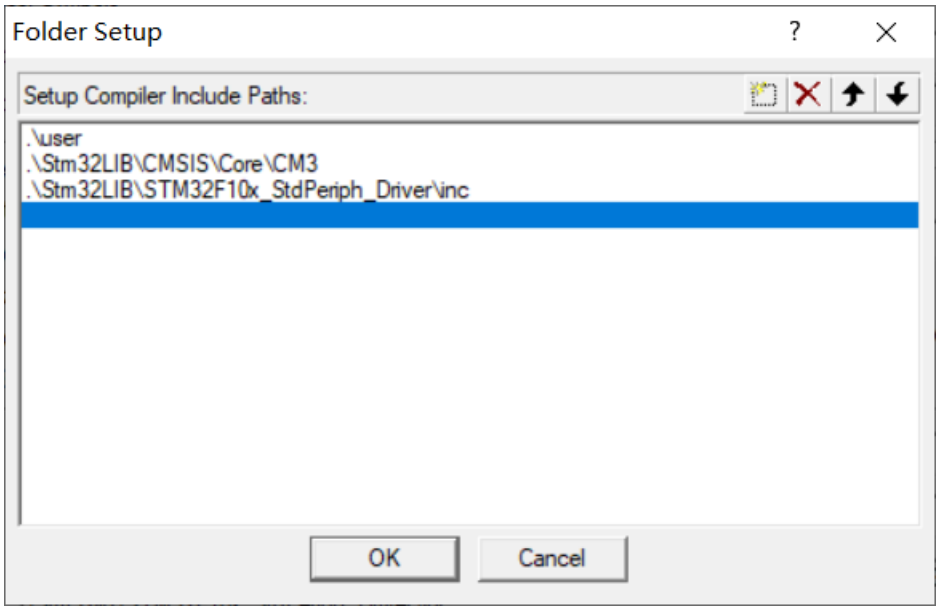
(1) 单击魔法棒形图标，出现Options for Target ‘test’界面，先将Target标签下的晶振频率设置成和你电路板上一样的值，这里是12MHz。



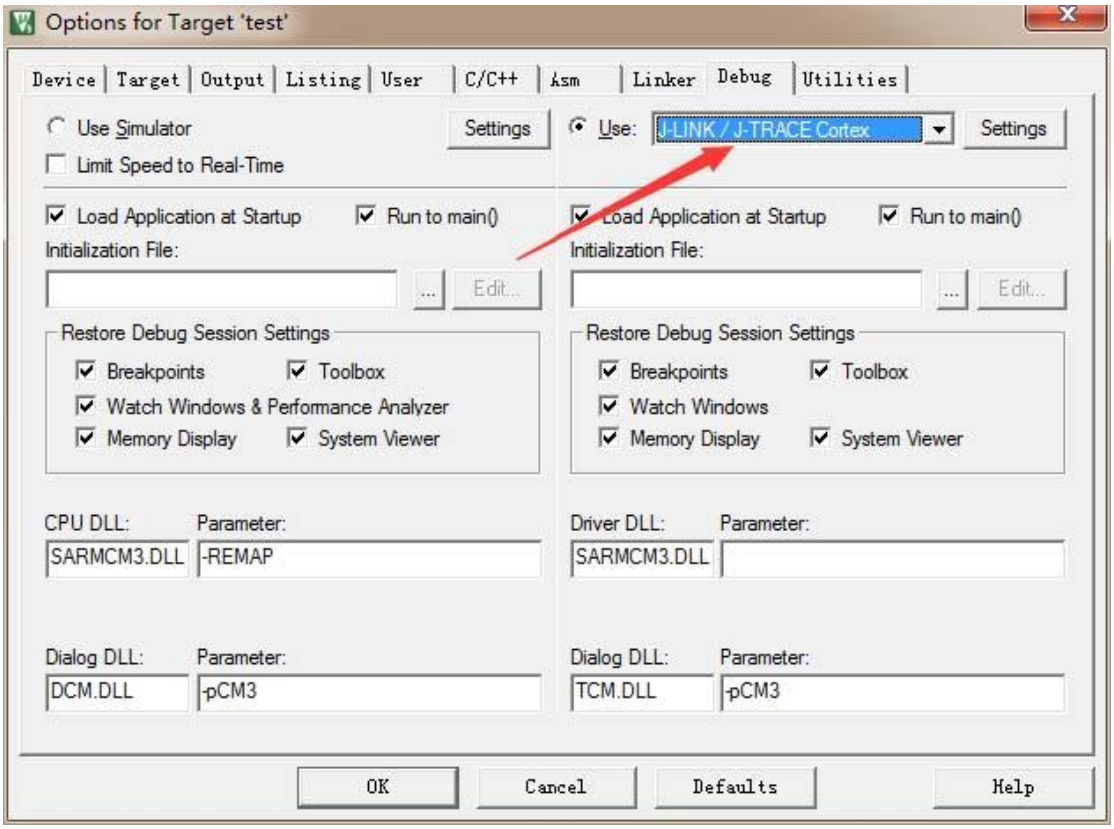
(2) 选择C/C++标签，在Define中输入：USE_STDPERIPH_DRIVER



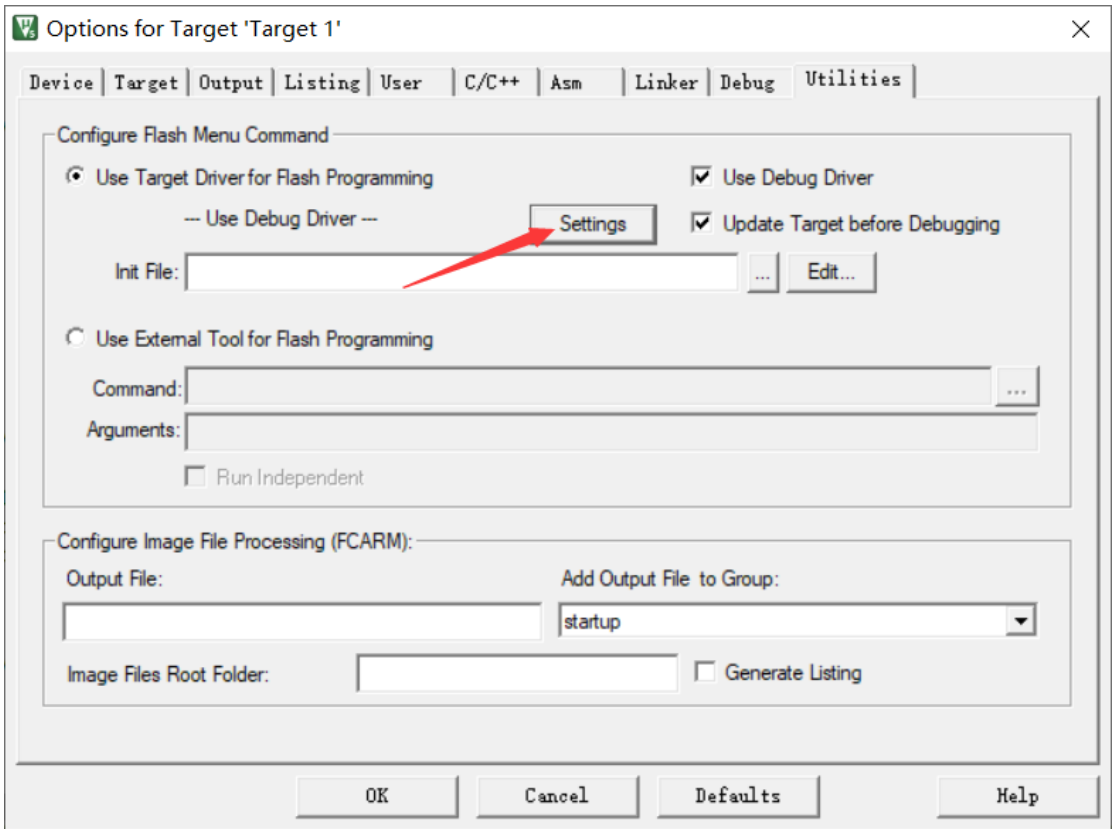
（3）在Include Paths中添加所有.h文件的根目录。



(4) Keil具有仿真方式和在线调试方式。仿真方式在Keil中内建芯片运行环境，可仿真算法类程序，与外围硬件端口状态无关。如果需要结合外围端口状态，则需进入在线调试模式，在Debug标签中选择所使用的的调试器，可以是ST-Link、Ulink、J-LINK/J-TRACE Cortex等不同的调试器，具体使用的调试器类型见硬件标识。图例使用J-Link。其它参数按照图中的配置默认即可。

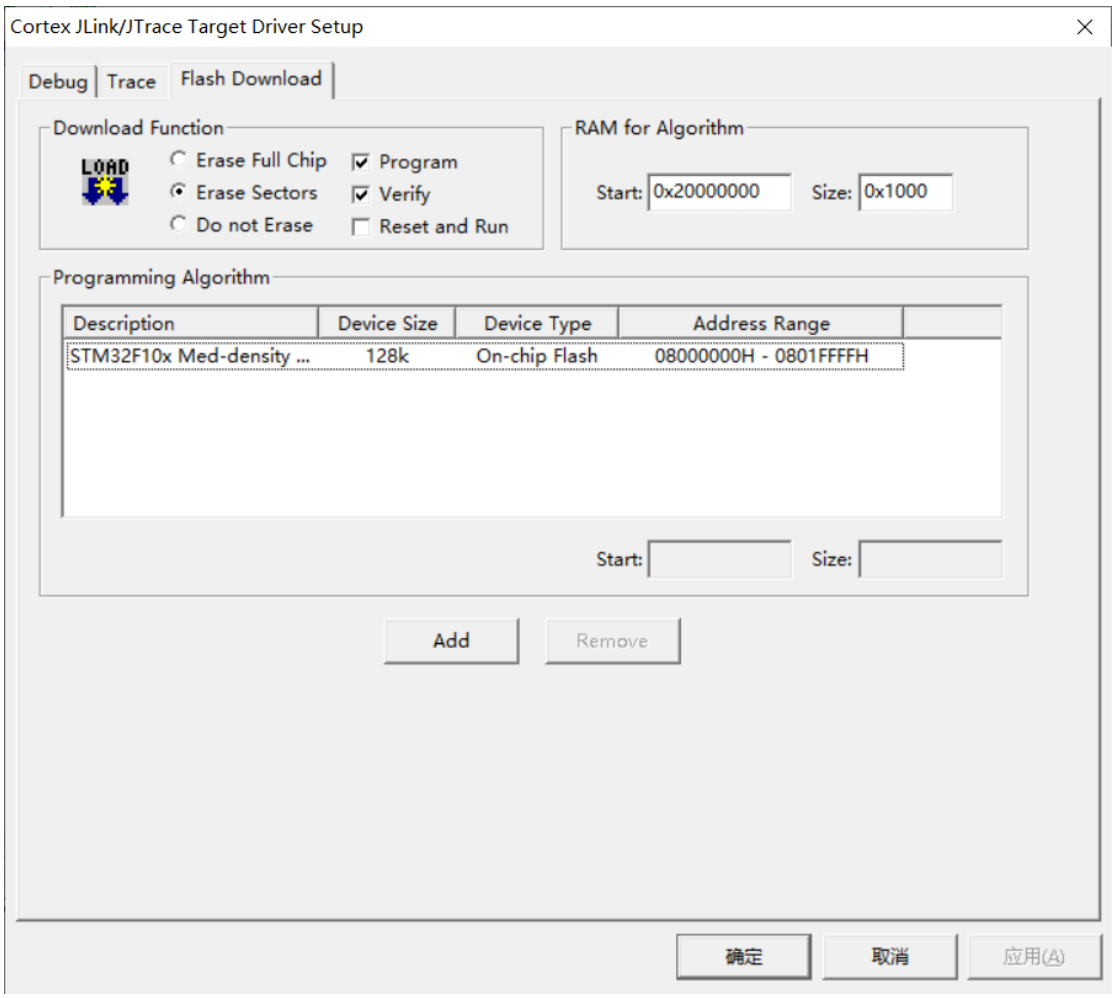


(5) 在Utilities标签中， Setting按钮右侧两个复选框选中，表示与Debug页的调试器使用同一个接口驱动程序。然后选择Settings按钮，点击。



进入如下的烧写配置页面，DownLoad Function功能下的单选框可以选择整

片擦除（Erase full Chip，就是把整个flash都擦除一遍）、扇区擦除（Erase Sector，只擦除用到的扇区）或者不擦除（Do not Erase，只是进入调试状态，不破坏原来的程序）。复选框的Program表示把程序写入flash，Verify表示写完后校验是否正确，Reset and Run表示复位后运行。RAM for Algorithm默认即可。Programming Algorithm选择编程算法，点击Add按钮选择目标芯片对应的型号即可。



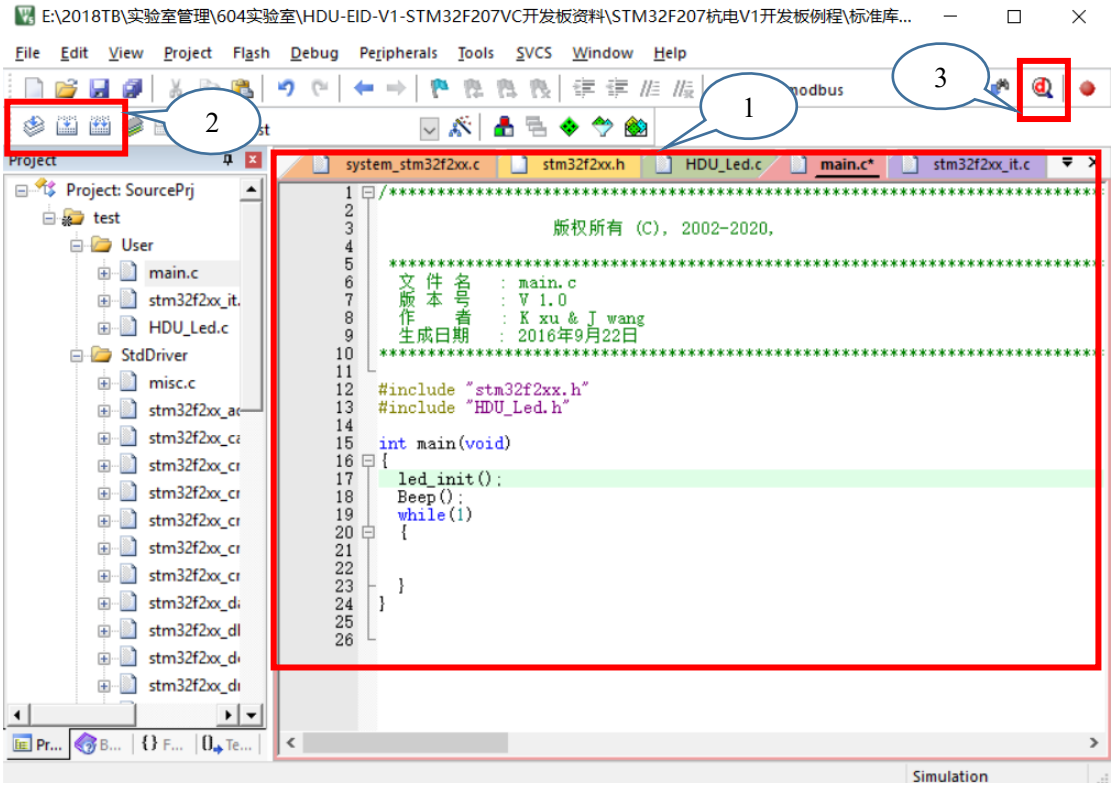
点击OK，完成工程新建步骤。

2.1.4 程序开发与调试

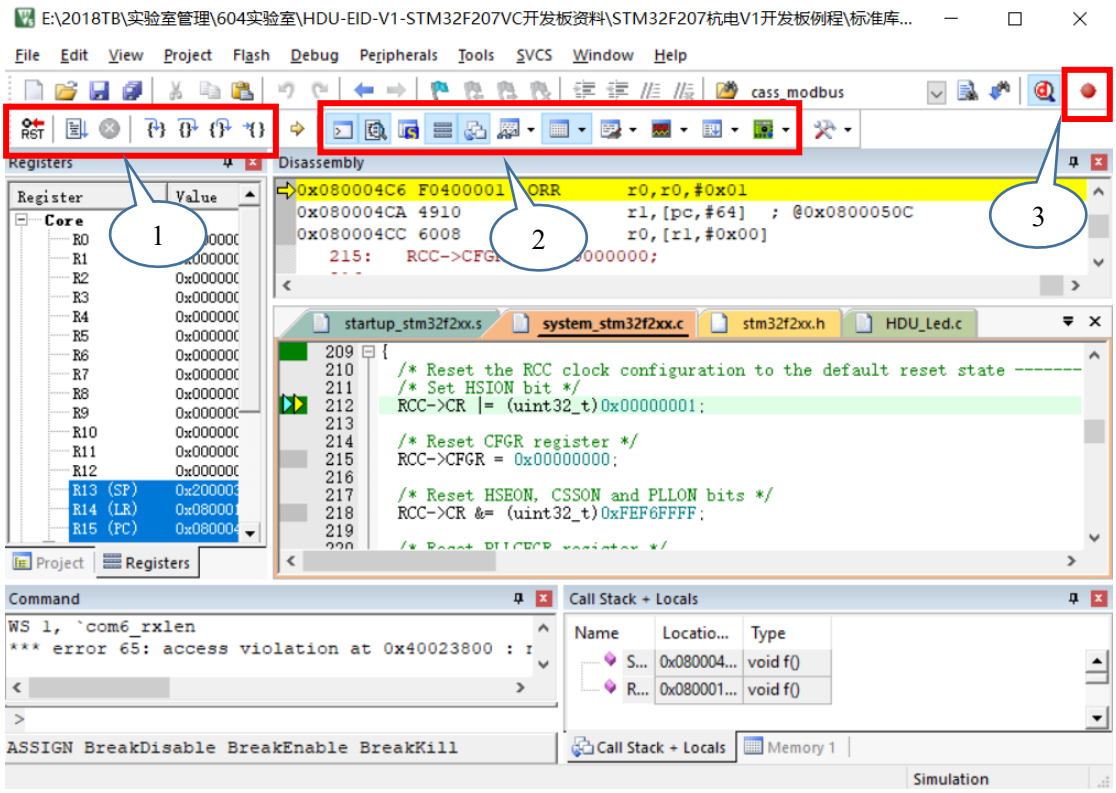
如下图所示，标号 1 为程序开发窗口，通过在左侧的工程目录中双击需要编辑的文件，在窗口标号 1 处完成代码编写。

标号 2 处为程序检查/构建按钮。从左到右依次为编译按钮，只检查语法功能；第二个按钮为构建按钮，在使用过程中，对文件执行编译和链接工作，生成可执行目标文件（.axf 文件），默认只对修改过的文件执行编译，对所有目标文件执行链接；第三个按钮为重新编译所有文件并执行性链接动作，耗时较长。

标号 3 执行调试功能，包含程序的下载和进入调试界面。



点击上图标号 3 的调试按钮后，进入如下界面，常用的功能按钮如下图红色框中的按钮。标号 1 所示按钮从左只有依次为程序复位、连续运行、停止运行、单步跟进（例如函数调用，会跟进函数里面）、单步跳过、跳出当前函数、运行到光标处，标号 1 按钮用于控制程序的执行。标号 2 按钮用于显示或隐藏调试过程中的窗口，包括状态窗口、内存窗口、寄存器窗口、变量窗口等，在程序调试过程中非常有用，鼠标移到按钮上会显示窗口提示信息，通过单击打开或关闭相应窗口。标号 3 按钮为断点按钮，点击后在光标所在行添加或删除断点。



2.1.5 考核方式

要求学生熟练掌握调试功能，能自由打断点、运行到断点处、运行到光标处、单步执行（指令单步、程序单步）、看变量、看内存等，熟练掌握程序调试方法。

2.2 跑马灯实验

2.2.1 实验目的

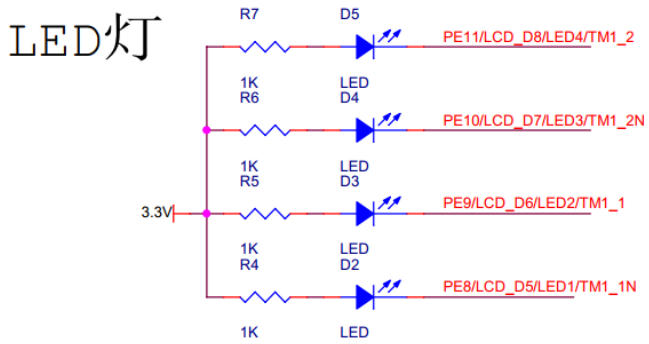
本实验为基础入门实验，希望学生对嵌入式编程有初步的了解，掌握 RCC 与 GPIO 的操作和使用。

2.2.2 实验要求

通过控制 PE8，PE9，PE10，PE11 的电平变化实现核心板上 D2、D3、D4、D5 四个 LED 灯的亮灭。

2.2.3 实验原理

如下图所示，四个 LED 正极通过电阻连接到电源+3.3V，LED 负端连接到 CPU 的 PEx 引脚。由此可知，只需要通过 CPU 控制 PEx 引脚为低电平，对应的指示灯即可点亮，如果控制 PEx 引脚为高电平，这对应的 LED 熄灭。通过程序循环扫描，即可实现跑马灯控制。




```
RCC->AHB1ENR |=(uint32_t)0x00000040;          //端口时钟使能
GPIOE->CRH &= 0XFFFFFFF0;                     // PE8 设置清 0
GPIOE->CRH |= 0X00000003;                     //设为通用推挽输出，最大速度 50MHZ
GPIOE->ODR |= 1<<8;                           //PE8 置为高电平
```

上述实例为针对 PE8 引脚配置，PE9、PE10、PE11 配置方式相同.

```
GPIOE->BRR |= (uint16_t)0x0100;     //引脚输出低电平，控制指示灯亮
GPIOE->BSRR |= (uint16_t)0x0100;    //引脚输出高电平，控制指示灯灭
```

2.1.4 实验步骤

1）连接仿真器 USB 线，注意此时 USB 线不要和 PC 连接
2）连接仿真和开发板之间的 20 芯排线
3）仿真器 USB 线插入电脑 USB 接口
4）下载程序并调试，分析并检验实验结果。

2.1.5 考核方式

考核四个 LED 按顺序熄灭点亮功能是否正确，是否掌握 IO 相关寄存器的编程控制方法。

2.3 数码管显示实验

2.3.1 实验目的

加深对 gpio 口的了解，同时结合工程实际，掌握常用的扫描法控制多个 LED 数码管。通过查阅 datasheet 与原理图，在确定驱动电路的基础上，实现对数码管的控制。本实验涉及 RCC，GPIO 的配置。

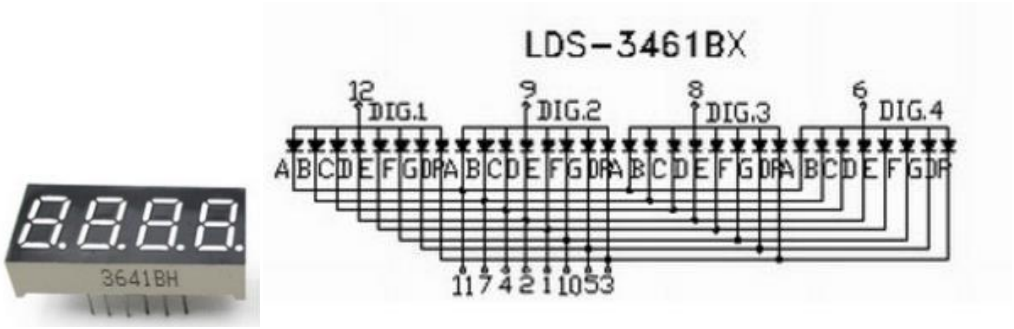
2.3.2 实验要求

通过控制数码管各段对应的 CPU 引脚的电平变化实现每一段的亮灭控制。同时实现每个数码管的总电源开关的通断控制，4 个数码管分时循环扫描控制。

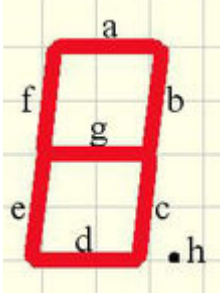
设计字段码表，实现 0-9，A-F 的数值显示。

2.3.3 实验原理

如下图所示，四个数码管共同分装在一个整体外壳中，其内部每个数码管的 8 段为 8 个 LED 灯，LED 灯采用共阳极接法设计。由于四个数码管数据线采用共用方式，四个数码管在某一时刻只能控制其中某一个显示，或者四个数码管同时显示相同内容。显然，在大部分场合时要求显示不同内容，所以只能通过对 4 给数码管进行分时控制控制，循环显示不同内容，在保证一定的扫描速度的情况下，人的眼睛是看不出来只亮了其中一个数码管(其原理类似显示器的刷新功能)。

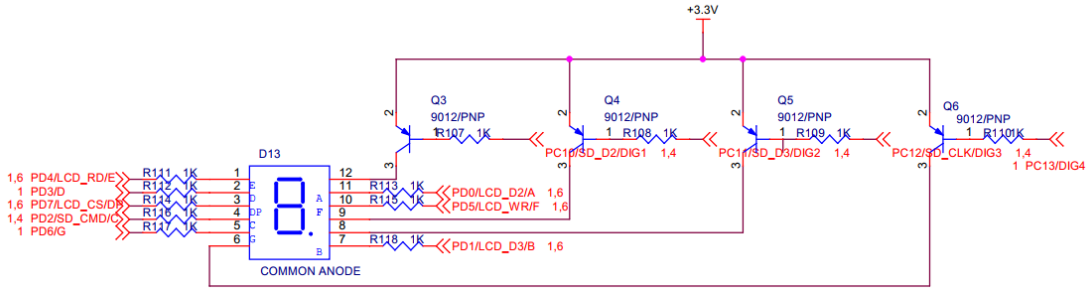


下图是数码管不同段对应的编码，只需控制每一段的亮灭即可在数码管上显示数值 0-9 和字母 AbCdEF 及其它有别于前述的这些数字或字母。



下图是开发板数码管模块原理图，abcdefgh 八段分别连接到 CPU 的 PD0、PD1、PD2、

PD3、PD4、PD5、PD6、PD7 。DIG1-4 分别接至 CPU 的 PC110、PC11、PC12、PC13。




```
    RCC->APB2ENR|=3<<4;        //使能 PORTC 和 PORTD 时钟
    GPIOC->CRH&=0xFF0000FF;
    GPIOC->CRH|=0x00333300;     //PC.10 11 12 13  推挽输出
//  GPIOC->ODR|=15<<10;         //PC.10 11 12 13  输出高

    GPIOD->CRL&=0X00000000;
    GPIOD->CRL|=0X33333333;//PD.0 1 2 3 4 5 6 7  推挽输出
//  GPIOD->ODR |= 0xFF;         //PD.0 1 2 3 4 5 6 7  输出高

    GPIOD->BSRR = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3;          //输出高
电平
    GPIOD->BSRR = GPIO_Pin_4 | GPIO_Pin_5 | GPIO_Pin_6 | GPIO_Pin_7;
    GPIOC->BSRR = GPIO_Pin_10 | GPIO_Pin_11 | GPIO_Pin_12 | GPIO_Pin_13;
```

2.3.4 实验步骤

1）连接仿真器 USB 线，注意此时 USB 线不要和 PC 连接
2）连接仿真和开发板之间的 20 芯排线
3）仿真器 USB 线插入电脑 USB 接口
4）下载程序并调试，分析并检验实验结果。

2.3.5 考核方式

要求数码管同时显示四个不同的数字，例如 1234。

2.4 串口通讯实验

2.4.1 实验目的

串口作为 MCU 的重要外部接口，同时也是软件开发重要的调试手段。本实验要求学生掌握对 usart 的操作，掌握串口通讯的收发规则，能熟练使用串口收发数据，同时涉及内部中断向量知识的掌握，本实验涉及 USART，GPIO，NVIC，RCC 的配置。

2.4.2 实验要求

PC 端串口调试助手软件（第三方工具，网上下载），向板子发送特定的十六进制数据串，CPU 收到数据后如果数据判断正确则返回应答数据。要求学生在程序运行中通过程序断点，理解数据收发过程。

2.4.3 实验原理

1）、串行通信基础知识：

串口通信是指外设和计算机间，通过数据信号线 、地线、控制线等，按位进行传输数据的一种通讯方式。这种通信方式使用的数据线少，在远距离通信中可以节约通信成本，但其传输速度比并行传输低。

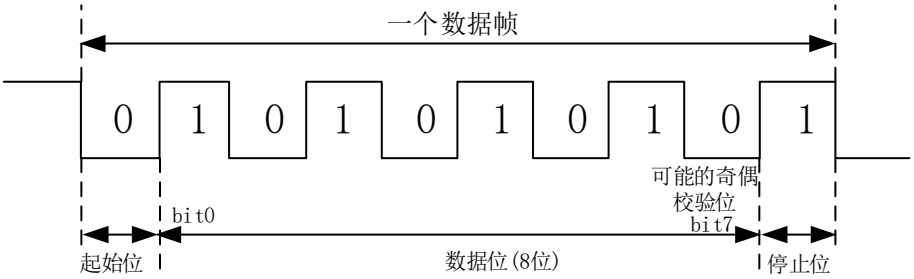
串行通讯的特点是：数据位传送，传按位顺序进行，最少只需一根传输线即可完成，成本低但送速度慢。串行通讯的距离可以从几米到几千米。

RS-232（串口的英文代名词）是串行通信方式的一种，采取不平衡传输方式，即所谓单端通讯，其共模抑制能力差，再加上双绞线上的分布电容，其传送距离最长为约 15 米，最高速率为 20kb/s。RS-232 是为点对点（即只有一对收、发设备）通讯而设计的，其驱动器负载为 3～7kΩ，适合本地设备之间的通信。

串行通信又分为两种方式：异步通信与同步通信。

2）、串行异步通信及其协议

异步通信以一个字符为传输单位，通信中两个字符间的时间间隔不固定可以是任意长的，然而在同一个字符中的相邻两个位的时间间隔是固定的，接收时钟和发送时钟只要相近就可以。 通信双方必须使用约定的相同的一些规则（也叫通信协议）。RS-232 的传送一个字符的信息格式规定有起始位、数据位、奇偶校验位（可选）、停止位，其中各位的意义如下：



串行异步传输时的数据格式：

⑴ 起始位：起始位必须是持续一个比特时间的逻辑“0”电平，标志传送一个字符的开始。

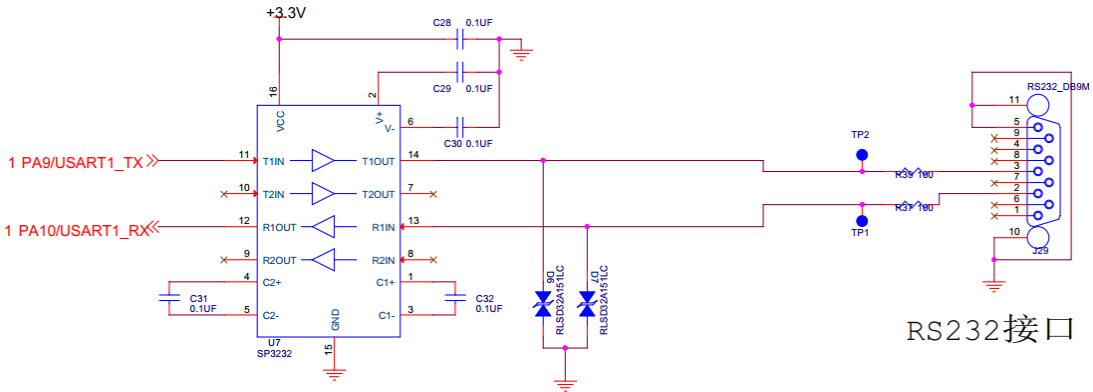
⑵ 数据位：数据位为 5-8 位，它紧跟在起始位之后，是被传送字符的有效数据位。传送时先传送字符的低位，后传送字符的高位。数据位究竟是几位，可由硬件或软件来设定。

⑶ 奇偶位：奇偶校验位仅占一位，用于进行奇校验或偶校验，也可以不设奇偶位。

⑷ 停止位：停止位为 1 位、1.5 位或 2 位，可有软件设定。它一定是逻辑“1”电平，标志着传送一个字符的结束。

⑸ 空闲位：空闲位表示线路处于空闲状态，此时线路上为逻辑“1”电平。空闲位可以没有，此时异步传送的效率为最高。

实验板电路原理图如下图所示，左侧 UART1_TX、UART1_TX 连接到 CPU 的串口 1，为 TTL 电平，低电平表示数据 0，高电平表示数据 1。右侧 RS232_Tout、RS232_Rin 连接至板子对外接口 J29，为 EIA-RS-232C 定义的逻辑电平：数据（信息码）：逻辑“1”电平为-3~-15V，逻辑“0”（空号）的电平高于+3~+15V；介于-3～+3V 之间的电压无意义，低于-15V 或高于+15V 的电压也认为无意义。



接口定义：开发板串口采用 3 线制串口，即 RS-232 接口端（DB9 公头/针型）只用了其中的 3 根脚，引脚定义为：2（RXD），3（TXD），5（GND），该接口定义符合 DB9 串口的

引脚哦规范，可使用 9 芯的串口交叉线与计算机的串口连接。

配置程序：

```
RCC->APB2ENR|=1<<2;          //使能 PORTA 口时钟
RCC->APB2ENR|=1<<14;         //使能串口时钟
GPIOA->CRH&=0XFFFFF00F;  //PA9、PA10 状态设置
GPIOA->CRH|=0X000000B0;     //PA9 复用推挽输出，最大速度 50MHZ
GPIOA->CRH|=0X00000400;     //PA10 浮空输入，最大速度 50MHZ
RCC->APB2RSTR|=1<<14;        //复位串口 1
RCC->APB2RSTR&=~(1<<14); //停止复位
USART1->BRR = 0x1d4c;        //波特率设置为 9600
USART1->CR1 |= 0X200C;       //1 位停止,无校验位, 接收、发送使能位置 1
USART1->CR1|=1<<5;           //接收缓冲区非空中断使能
```

读取数据：

```
if(USART1->SR&(1<<5)){                  // 判断接收缓冲区是否非空
    RS232InData = USART1->DR;           // 将数据从寄存器中读出
}
```

发送数据：

```
USART1->DR = RS232InData ;          //发送一个字节数据
```

2.4.4 实验步骤

1）连接仿真器 USB 线，注意此时 USB 线不要和 PC 连接
2）连接仿真和开发板之间的 20 芯排线
3）仿真器 USB 线插入电脑 USB 接口
4）使用串口线，连接左边串口至电脑
5）下载程序并调试，分析并检验实验结果。

2.4.5 考核方式

正确实现一个自己定义的通讯协议，与计算机上的串口调试工具软件实现数据交互。

2.5 按键扫描实验

2.5.1 实验目的

在按键数量多的场合，矩阵键盘与独立式按键键盘相比可以节省很多的 I/O 口线。通过本实验理解矩阵键盘扫描原理，掌握程序开发方法。

2.5.2 实验要求

按相应的键，从串口输出相应的键值，键值可以自己定义。

2.5.3 实验原理

矩阵键盘是单片机外部设备中所使用的排布类似于矩阵的键盘组。矩阵式结构的键盘显然比直接法要复杂一些，识别也要复杂一些。主要应用在按键较多，但是处理器引脚受限的场合。通过将按键排列成矩阵形式，减少 I/O 口的占用。在矩阵式键盘中，每条水平线和垂直线在交叉处不直接连通，而是通过一个按键加以连接。这样，8 个 IO 接口就可以构成 4*4=16 个按键，比之直接将端口线用于键盘多出了一倍，而且线数越多，区别越明显，比如再多加一条线就可以构成 20 键的键盘，而直接用端口线则只能多出一键（9 键）。由此可见，在需要的键数比较多时，采用矩阵法来做键盘是合理的。

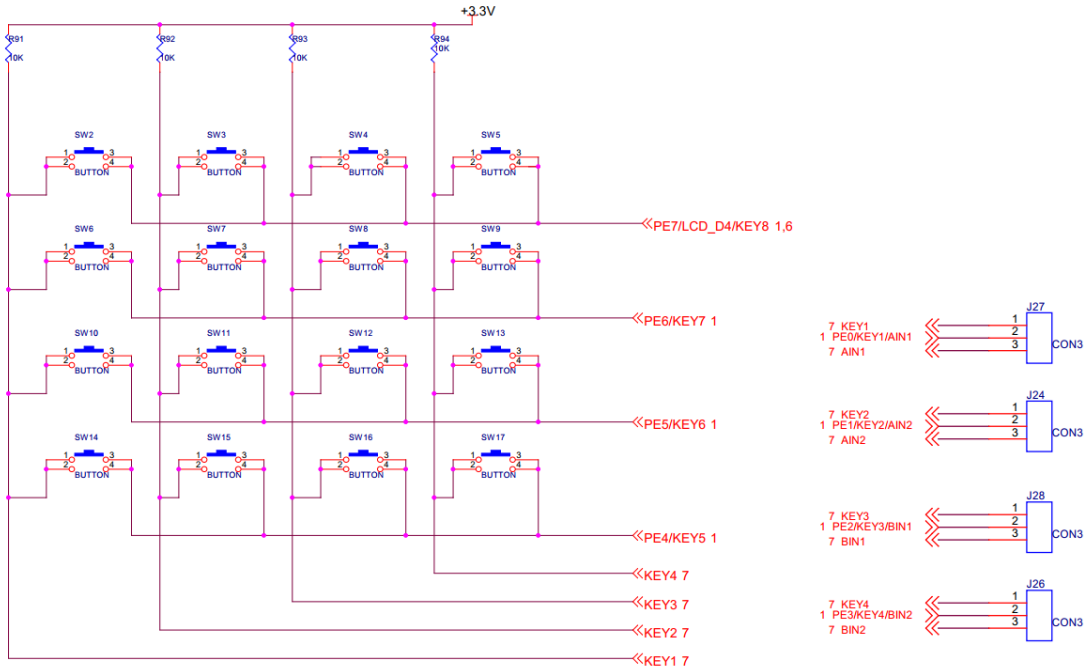
如何确定矩阵式键盘上何键被按下的方法有多种，本文介绍一种“行扫描法”。

行扫描法又称为逐行（或列）扫描查询法，是一种最常用的按键识别方法，介绍过程如下。

1、判断键盘中有无键按下：将全部行线置低电平，然后检测列线的状态。只要有一列的电平为低，则表示键盘中有键被按下，而且闭合的键位于低电平线与 4 根行线相交叉的 4 个按键之中。若所有列线均为高电平，则键盘中无键按下。

2、判断闭合键所在的位置：在确认有键按下后，即可进入确定具体闭合键的过程。其方法是：依次将行线置为低电平，即在置某根行线为低电平时，其它线为高电平。在确定某根行线位置为低电平后，再逐行检测各列线的电平状态。若某列为低，则该列线与置为低电平的行线交叉处的按键就是闭合的按键。

实验用 STM32 开发板总共有 4*4 个功能按键，由用户自定义其功能。按键分别由行和列构成，行线分别连接 CPU 的 PE4、PE5、PE6、PE7。列线分别连接到 CPU 的 PE0、PE1、PE2、PE3。注意行线对应的 CPU 引脚和其他功能接口引脚复用，通过跳线选择（见 3.2 说明，J 24、J26、J27、J28 短路到指定位置）。



引脚配置：

RCC->APB2ENR|=1<<6;　　//使能 PORTE 时钟

GPIOE->CRL=0X33338888;　　//PE.0 1 2 3　上拉输入　4 5 6 7　推挽输出

特别注意：key1-4 是输入模式，key5-8 为输出模式

2.4.4 实验步骤

1）连接仿真器 USB 线，注意此时 USB 线不要和 PC 连接

2）连接仿真和开发板之间的 20 芯排线

3）仿真器 USB 线插入电脑 USB 接口

4）使用串口线，连接左边串口至电脑

5）下载程序并调试，分析并检验实验结果。

2.4.5 考核方式

对按键进行编号，正确显示按下的按键编号值。

2.6 定时器实验

2.6.1 实验目的

掌握定时器的使用，及定时器与中断之间的联系。本实验涉及 TIM，GPIO，NVIC，RCC 等部件应用。

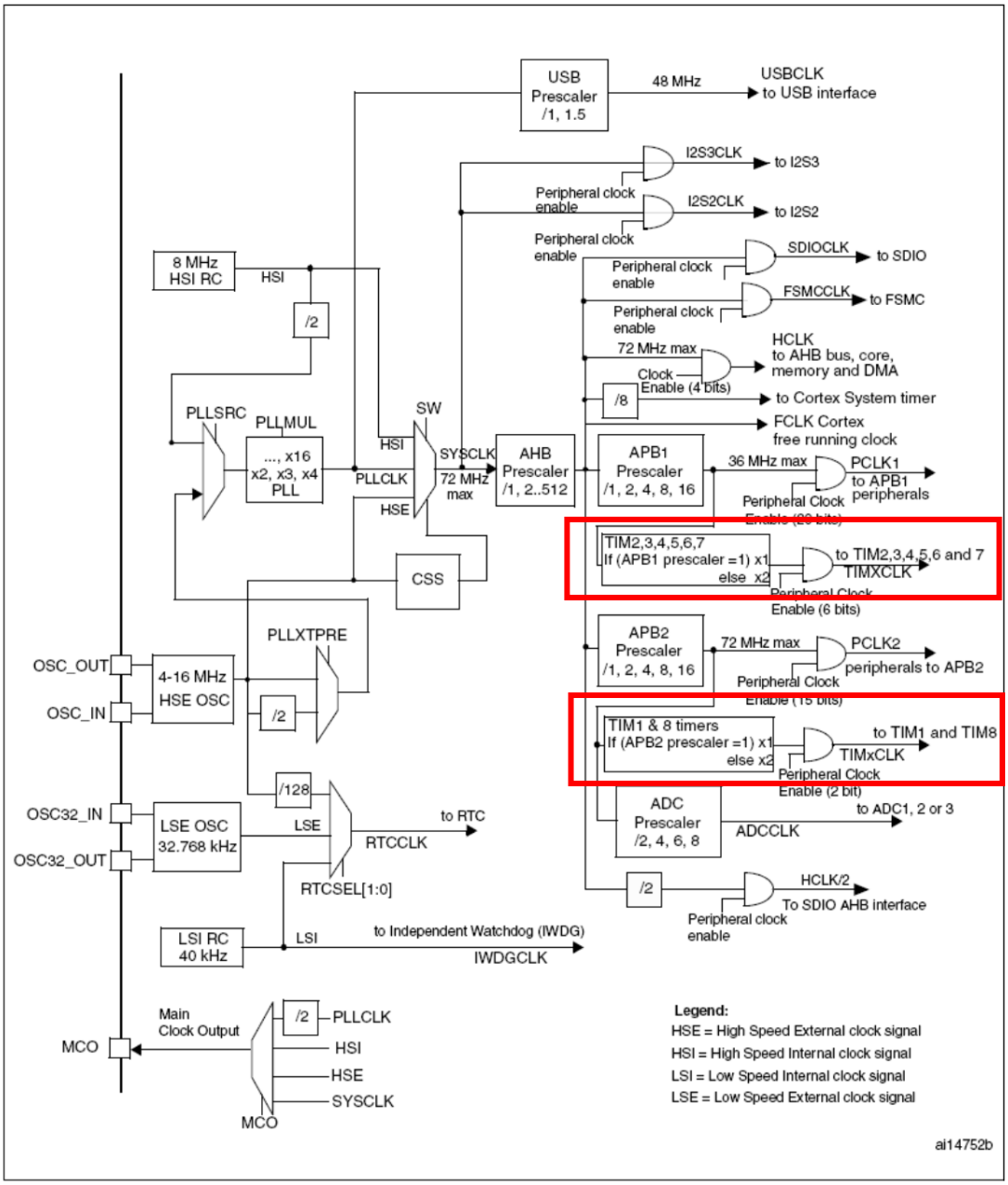
2.6.2 实验要求

使用定时器中断实现四个 LED 周期性闪灭，要求学生掌握定时器的配置和控制，实现不同时长的定时。

2.6.3 实验原理

为了讲明定时器原理，首先讲计数器。计数器是对输入信号高低电平变化次数的记录，当信号一次反转，计数单元数值加一。输入信号可以是周期信号，也可以是偶发性的脉冲信号。当时输入信号是周期信号时，可以根据记录的信号变化次数，换算出这些信号所经历的确切时间。定时器，就是根据特定的周期信号，由芯片自身提供的计数器，对脉冲进行计数。如果需要定时特定的时间，只要设定计数器所需记录的脉冲个数，当记录的脉冲个数到达这个设定值时，输出时间到的信号，用于执行定时任务，这就是定时器的基本原理。对于定时器来说，输入的周期脉冲信号通常为 CPU 的各种工作时钟。

STM32F103 芯片定时器的时钟输入源是 APB1（TIM2-7）或 APB2（TIM1，TIM8），可以根据选择是否 2 倍频，见图中的红色框部分。下面以通用定时器 2 的时钟说明这个倍频器的作用：当 APB1 的预分频系数为 1 时，这个倍频器不起作用，定时器的时钟频率等于 APB1 的频率；当 APB1 的预分频系数为其它数值(即预分频系数为 2、4、8 或 16)时，这个倍频器起作用，定时器的时钟频率等于 APB1 的频率两倍。具体时钟频率参看工程 SystemInit（）函数中的 SetSysClock（）配置，还跟系统晶振频率有关。




```
RCC->APB1ENR|=1<<2;     //TIM4 时钟使能
TIM4->ARR=4999;         //设定计数器自动重装值//刚好 1ms
TIM4->PSC=7199;         //预分频器 7200,得到 10Khz 的计数时钟
TIM4->DIER|=1<<0;     //允许更新中断
TIM4->CR1|=0x01;      //使能定时器 4

NVIC_InitTypeDef     NVIC_InitStructure;                    // 中断优先级配置
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

中断函数

```
void TIM2_IRQHandler(void)
{
  if((TIM4->SR&0X0001))     //溢出中断
  {
      // 中断处理部分
  }
  TIM4->SR&=~(1<<0);        //清除中断标志位
}
```

2.6.4 实验步骤

1）连接仿真器 USB 线，注意此时 USB 线不要和 PC 连接
2）连接仿真和开发板之间的 20 芯排线
3）仿真器 USB 线插入电脑 USB 接口
4）下载程序并调试，分析并检验实验结果。

2.6.5 考核方式

将定时器与已经学过的外设相关联。可以选择数码管定时变化数字，自增长；或者流水灯的熄灭点亮时长或者串口定时输出。

2.7 外部中断实验

2.7.1 实验目的

本实验要求学生掌握外部中断的使用和编写，能够和其他外设相关联。本实验涉及SYSCFG，EXTI，NVIC，GPIO，RCC 的配置。

2.7.2 实验要求

按下 SW18 按键，改变灯的状态，要求学生掌握外部中断与内部向量中断之间的对应，能在调试过程中，准确找到中断发生情况时程序的应对。

2.7.3 实验原理

外部中断是单片机实时地处理外部事件的一种内部机制。外部中断/事件控制器由 20 个产生事件/中断请求的边沿检测器组成，对于其它产品，则有 19 个能产生事件/中断请求的边沿检测器。每个输入线可以独立地配置输入类型(脉冲或挂起)和对应的触发事件(上升沿或下降沿或者双边沿都触发)。每个输入线都可以独立地被屏蔽。挂起寄存器保持着状态线的中断请求。

```
//设置 NVIC 分组
//NVIC_Group:NVIC 分组  0~4  总共 5 组
void MY_NVIC_PriorityGroupConfig(u8 NVIC_Group)
{
    u32 temp,temp1;
    temp1=(~NVIC_Group)&0x07;//取后三位
    temp1<<=8;
    temp=SCB->AIRCR;   //读取先前的设置
    temp&=0X0000F8FF; //清空先前分组
    temp|=0X05FA0000; //写入钥匙
    temp|=temp1;
    SCB->AIRCR=temp;   //设置分组
}
//设置 NVIC
//NVIC_PreemptionPriority:抢占优先级
//NVIC_SubPriority          :响应优先级
//NVIC_Channel                :中断编号
//NVIC_Group                  :中断分组  0~4
//注意优先级不能超过设定的组的范围!否则会有意想不到的错误
//组划分:
//组 0:0 位抢占优先级,4 位响应优先级
//组 1:1 位抢占优先级,3 位响应优先级
```

```
//组 2:2 位抢占优先级,2 位响应优先级
//组 3:3 位抢占优先级,1 位响应优先级
//组 4:4 位抢占优先级,0 位响应优先级
//NVIC_SubPriority 和 NVIC_PreemptionPriority 的原则是,数值越小,越优先
void MY_NVIC_Init(u8 NVIC_PreemptionPriority,u8 NVIC_SubPriority,u8 NVIC_Channel,u8 NVIC_Group)
{
    u32 temp;
    MY_NVIC_PriorityGroupConfig(NVIC_Group);//设置分组
    temp=NVIC_PreemptionPriority<<(4-NVIC_Group);
    temp|=NVIC_SubPriority&(0x0f>>NVIC_Group);
    temp&=0xf;                                  //取低四位
    NVIC->ISER[NVIC_Channel/32]|=(1<<NVIC_Channel%32);//使能中断位(要清除的话,相反操作就 OK)
    NVIC->IP[NVIC_Channel]|=temp<<4;        //设置响应优先级和抢断优先级

}
//外部中断配置函数
//只针对 GPIOA~G;不包括 PVD,RTC 和 USB 唤醒这三个
//参数:
//GPIOx:0~6,代表 GPIOA~G
//BITx:需要使能的位;
//TRIM:触发模式,1,下升沿;2,上降沿;3，任意电平触发
//该函数一次只能配置 1 个 IO 口,多个 IO 口,需多次调用
//该函数会自动开启对应中断,以及屏蔽线
void Ex_NVIC_Config(u8 GPIOx,u8 BITx,u8 TRIM)
{
    u8 EXTADDR;
    u8 EXTOFFSET;
    EXTADDR=BITx/4;//得到中断寄存器组的编号
    EXTOFFSET=(BITx%4)*4;
    RCC->APB2ENR|=0x01;//使能 io 复用时钟
    AFIO->EXTICR[EXTADDR]&=~(0x000F<<EXTOFFSET);//清除原来设置!!!
    AFIO->EXTICR[EXTADDR]|=GPIOx<<EXTOFFSET;//EXTI.BITx 映 射 到 GPIOx.BITx
    //自动设置
    EXTI->IMR|=1<<BITx;//  开启 line BITx 上的中断
    //EXTI->EMR|=1<<BITx;//不屏蔽 line BITx 上的事件 (如果不屏蔽这句,在硬件上是可以的,但是在软件仿真的时候无法进入中断!)
    if(TRIM&0x01)EXTI->FTSR|=1<<BITx;//line BITx 上事件下降沿触发
    if(TRIM&0x02)EXTI->RTSR|=1<<BITx;//line BITx 上事件上升降沿触发
}
```

```
//外部中断初始化程序
//初始化 PB6 中断输入.
void EXTIX_Init(void)
{
    Ex_NVIC_Config(GPIO_B,6,FTIR);          //下降沿触发
    MY_NVIC_Init(2,3,EXTI9_5_IRQn,2);      //抢占 2，子优先级 3，组 2
}

中断服务函数：
int temps=0;
void EXTI9_5_IRQHandler(void)
{
    int num=-1;
    num=KEY_Scan();
    delay_ms(10); //消抖
    if(num==1)
    {
        temps++;
    }
    if(temps%2) One_LED_ON(1);
    else One_LED_OFF(1);
    EXTI->PR=1<<6;   //清除 LINE6 上的中断标志位
}
```

2.7.4 实验步骤

1）连接仿真器 USB 线，注意此时 USB 线不要和 PC 连接
2）连接仿真和开发板之间的 20 芯排线
3）仿真器 USB 线插入电脑 USB 接口
4）下载程序并调试，分析并检验实验结果。

2.7.5 考核方式

考察中断程序的编写能力。可以将外部中断与其他外设相关联，例如按键使得数码管数字变化，或者按键控制串行通讯的数据发送。

2.8 计数器实验

2.8.1 实验目的

本实验要求学生掌握定时器计数的使用和编写，能够和其他外设相关联。本实验涉及SYSCFG，TIM，GPIO，RCC 的配置。

2.8.2 实验要求

在 debug 模式下，按下 SW18 按键，观察计数器寄存器 TIM4->CNT 数值变化，要求学生理解掌握通用定时器的计数器模式。

2.8.3 实验原理

通用定时器可以向上计数、向下计数、向上向下双向计数模式。
①向上计数模式：计数器从 0 计数到自动加载值(TIMx_ARR)，然后重新从 0 开始计数并且产生一个计数器溢出事件。
②向下计数模式：计数器从自动装入的值(TIMx_ARR)开始向下计数到 0，然后从自动装入的值重新开始，并产生一个计数器向下溢出事件。
③中央对齐模式（向上/向下计数）：计数器从 0 开始计数到自动装入的值-1，产生一个计数器溢出事件，然后向下计数到 1 并且产生一个计数器溢出事件；然后再从 0 开始重新计数。

```
RCC->APB2ENR|=1<<3;        //使能 PORTB 时钟
GPIOB->CRL|=0x04000000;    //PB.6 浮空输入

RCC->APB1ENR|=1<<2;       //使能 TIM4 时钟
TIM4->ARR=0x1000;         //设定计数器自动重装值
TIM4->PSC=0;              //不预分频

TIM4->CCMR1|=1<<0;        //CC1S=01     选择输入端  IC1 映射到 TI1 上
TIM4->CCMR1|=0<<4;              //IC1F=0000  配置输入滤波器  不滤波
TIM4->CCMR1|=0<<10;       //IC2PS=00     配置输入分频,不分频

TIM4->CCER|=0<<1;         //CC1P=0       上升沿捕获
TIM4->CCER|=1<<0;         //CC1E=1       允许捕获计数器的值到捕获寄存器
中

TIM4->SMCR = 0x50 ;     //选择同步计数器的触发模式为定时器输入 1
TIM4->SMCR |= 0x07;    //外部触发模式
TIM4->CNT=0;               //初始化计数器
TIM4->CR1|=0x01;       //使能定时器 4
```

2.8.4 实验步骤

1）连接仿真器 USB 线，注意此时 USB 线不要和 PC 连接
2）连接仿真和开发板之间的 20 芯排线
3）仿真器 USB 线插入电脑 USB 接口
4）下载程序并调试，分析并检验实验结果。

2.8.5 考核方式

考察计数器程序的编写能力。可以将定时器与其他外设相关联，例如按键使得灯亮灭。

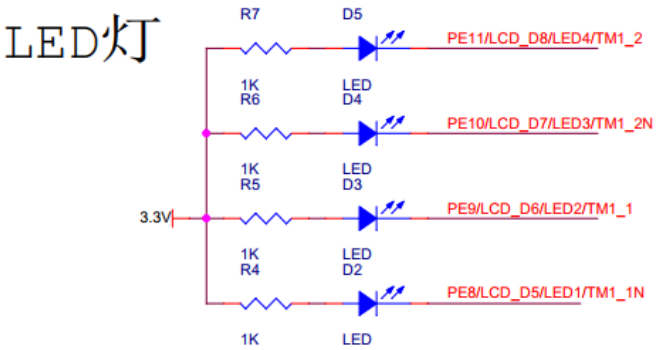
2.9 PWM 实验

2.9.1 实验目的

单片机有部分 I/O 口具有输出 PWM 信号的功能，是一种模拟控制。本实验要求学生掌握 pwm 的使用，了解 pwm 占空比等概念。本实验涉及 GPIO，TIM，RCC 的配置

2.9.2 实验要求

实验板上 LED 灯 D5 亮度变化，要求通过改变占空比，改变灯的亮暗程度和时间长短。



2.9.3 实验原理：

脉冲宽度调制(PWM)，是英文“Pulse Width Modulation”的缩写，简称脉宽调制，是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的技术。简单一点，就是对脉冲宽度的控制。

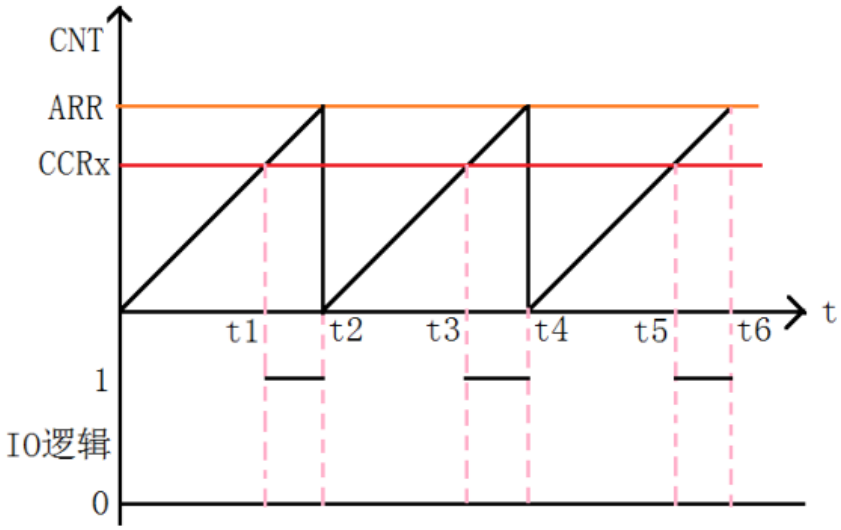


图 14.1.1 PWM 原理示意图

STM32 的定时器除了 TIM6 和 TIM7。其他的定时器都可以用来产生 PWM 输出。其中高级定时器 TIM1 和 TIM8 可以同时产生多达 7 路的 PWM 输出。而通用定时器也能同时产生多达 4 路的 PWM 输出，这样，STM32 最多可以同时产生 30 路 PWM 输出.

根据原理图，D5 和 D3 对应 TIM1 的 ch2 和 ch1，引脚配置如下：

```
RCC->APB2ENR|=1<<11;                //TIM1 时钟使能
RCC->APB2ENR|=1<<6;                 //使能 PORTE 时钟

GPIOE->CRH&=0XFFFF0F0F;             //PB5 输出
GPIOE->CRH|=0X0000B0B0;             //复用功能输出
GPIOE->ODR |= 0x1001 << 8;          // 上拉

TIM1->ARR=899;                      //设定计数器自动重装值
TIM1->PSC=0;                        //预分频器不分频
TIM4->ARR=899;                      //设定计数器自动重装值
TIM4->PSC=0;                        //预分频器不分频

TIM1->CCMR1|=7<<4;                  //CH1 PWM1 模式
TIM1->CCMR1|=1<<3;                  //CH1 预装载使能
TIM1->CCMR1|=7<<12;                 //CH2 PWM1 模式
TIM1->CCMR1|=1<<11;                 //CH2 预装载使能
TIM1->CCER|=1;                      //OC2 输出使能
TIM1->CCER|=1 << 12;                //OC2 输出使能

TIM1->BDTR |= 1<<15;
TIM1->CR1=0x0080;                   //ARPE 使能
TIM1->CR1|=0x01;                    //使能定时器 1
```

2.9.4 实验步骤

1）连接仿真器 USB 线，注意此时 USB 线不要和 PC 连接

2）连接仿真和开发板之间的 20 芯排线
3）仿真器 USB 线插入电脑 USB 接口
4）下载程序并调试，分析并检验实验结果。

2.9.5 考核方式

考核定时器的 PWM 工作方式的理解，通道配置，程序控制能力。

2.10 独立看门狗实验

2.10.1 实验目的

独立看门狗可用于监测程序是否正常运行，防止程序异常引起的死机（死循环）。本实验要求学生掌握独立看门狗的使用，并能用于保证自己程序的正确性，方便日后工程开发。本实验涉及 IWDG，GPIO，RCC 的配置。

2.10.2 实验要求

程序上电后，关闭指示灯，延时 2 秒后，开启看门狗，并通过程序控制 LED 灯周期性闪烁，程序循环中保持喂狗。如果按住 SW18 键，停止喂狗，查看指示灯情况，记录分析结果。

2.10.3 实验原理

独立看门狗(IWDG)由专用的低速时钟(LSI)驱动，即使主时钟发生故障它也仍然有效。IWDG 最适合应用于那些需要看门狗作为一个在主程序之外，能够完全独立工作，并且对程序进行运行状态进行监测的场合，时间精度较低。

对于 STM32F103 芯片，在寄存器(IWDG_KR)中写入 0xCCCC，开始启用独立看门狗；此时计数器开始从其复位值 0xFFF 递减计数。当计数器计数到末尾 0x000 时，会产生一个复位信号(IWDG_RESET)。无论何时，只要在键寄存器 IWDG_KR 中写入 0xAAAA，IWDG_RLR 中的值就会被重新加载到计数器，从而避免产生看门狗复位。

看门狗配置如下：

```
IWDG->KR = (uint16_t)0x5555;  //使能对寄存器 IWDG_PR 和 IWDG_RLR 的写操作
IWDG->PR = 4;          //设置 IWDG 预分频值
IWDG->RLR = 625;     //设置 IWDG 重装载值
IWDG->KR = (uint16_t)0xAAAA;     //按照 IWDG 重装载寄存器的值重装载 IWDG 计数器
 IWDG->KR = (uint16_t)0xCCCC;     //使能 IWDG
喂狗操作
IWDG->KR = (uint16_t)0xAAAA;     //按照 IWDG 重装载寄存器的值重装载 IWDG 计数器
```

2.10.4 实验步骤

1）连接仿真器 USB 线，注意此时 USB 线不要和 PC 连接
2）连接仿真和开发板之间的 20 芯排线
3）仿真器 USB 线插入电脑 USB 接口
4）下载程序并调试，分析并检验实验结果。

2.10.5 考核方式

理解看门狗原理，掌握看门狗使用方法，并在自己的其他工程中能加入看门狗程序，验证工程正确性。

2.11 窗口看门狗实验

2.11.1 实验目的

窗口看门狗同样用于防止程序死循环，但他含有中断。本实验要求学生掌握窗口看门狗的使用，并能用于保证自己程序的正确性，方便日后工程开发。本实验涉及 WWDG，GPIO，NVIC，RCC 的配置。

2.11.2 实验要求

程序实现 LED 快速的闪灭，表示在窗口看门狗中断中正在重新装载窗口看门狗的计数，表明喂狗成功，若不成功，LED 闪灭速度缓慢。要求学生明白喂狗过程及其原理，及喂狗失败的程序结果，了解窗口看门狗和中断的联系。

2.11.3 实验原理

窗口看门狗通常被用来监测由外部干扰或不可预见的逻辑条件造成的应用程序背离正常的运行序列而产生的软件故障。除非递减计数器的值在 T6 位变成 0 前被刷新，看门狗电路在达到预置的时间周期时，会产生一个 MCU 复位。在递减计数器达到窗口寄存器数值之前，如果 7 位的递减计数器数值(在控制寄存器中)被刷新， 那么也将产生一个 MCU 复位。这表明递减计数器需要在一个有限的时间窗口中被刷新。

拥有以下特性

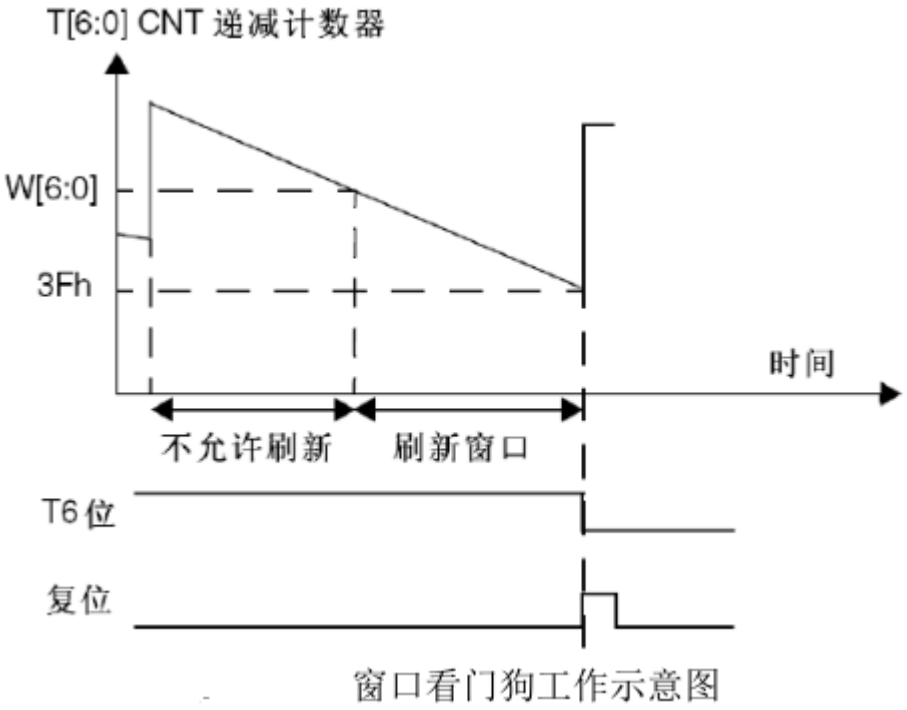
1.可编程的自由运行递减计数器

2.条件复位

3.当递减计数器的值小于 0x40，(若看门狗被启动)则产生复位。

4.当递减计数器在窗口外被重新装载，(若看门狗被启动)则产生复位。

5.如果启动了看门狗并且允许中断，当递减计数器等于 0x40 时产生早期唤醒中断(EWI)，它可以被用于重装载计数器以避免 WWDG 复位



窗口看门狗工作示意图

窗口看门狗参考代码：

```
RCC->APB1ENR|=1<<11;            //使能 wwdg 时钟
WWDG_CNT=tr&WWDG_CNT;           //初始化 WWDG_CNT.
WWDG->CFR|=fprer<<7;             //PCLK1/4096 再除 2^fprer
WWDG->CFR&=0XFF80;
WWDG->CFR|=wr;                  //设定窗口值
WWDG->CR|=WWDG_CNT;             //设定计数器值
WWDG->CR|=1<<7;                 //开启看门狗
NVIC->IP[0]=0x00;                //设置相应的中断
NVIC->ISER[0]=0x00000001;
WWDG->SR=0X00;                  //清除提前唤醒中断标志位
WWDG->CFR|=1<<9;                //使能提前唤醒中断
```

2.11.4 实验步骤

1）连接仿真器 USB 线，注意此时 USB 线不要和 PC 连接
2）连接仿真和开发板之间的 20 芯排线
3）仿真器 USB 线插入电脑 USB 接口
4）下载程序并调试，分析并检验实验结果。

2.11.5 考核方式

掌握窗口看门狗的原理及编程方法，并在自己的其他工程中能加入看门狗程序，证明程序正确性。

2.12 ADC 内部温度实验

2.12.1 实验目的

模数转换是指将连续变化的模拟信号转换为离散的数字信号。本实验要求学生掌握 adc 的转换使用，转换过程及其规则，掌握模拟到数字的转换方法及配置。本实验涉及 ADC，GPIO，RCC 等的配置。

2.12.2 实验要求

读取芯片内部温度传感器的温度并在软件环境中显示，或者通过串口、数码管等方式显示。需要学生掌握白 AD 转换的方法和过程，及转换规则和通道的输入使用方式。

2.12.3 实验原理

STM32 有一个内部的温度传感器，可以用来测量 CPU 及周围的温度(TA)。该温度传感器在内部和 ADCx_IN16 输入通道相连接，此通道把传感器输出的电压转换成数字值。温度传感器模拟输入推荐采样时间是 17.1μs。 STM32 的内部温度传感器支持的温度范围为：-40~125 度，精度为±1.5℃左右。

STM32 的 ADC 是 12 位逐次逼近型的模拟数字转换器。它有 18 个通道，可测量 16 个外部和 2 个内部信号源。各通道的 A/D 转换可以单次、连续、扫描或间断模式执行。ADC 的结果可以左对齐或右对齐方式存储在 16 位数据寄存器中。

STM32 将 ADC 的转换分为 2 个通道组：规则通道组和注入通道组。规则通道相当于你正常运行的程序，而注入通道呢，就相当于中断。在你程序正常执行的时候，中断是可以打断你的执行的。同这个类似，注入通道的转换可以打断规则通道的转换， 在注入通道被转换完成之后，规则通道才得以继续转换

内部温度传感器固定的连接在 ADC1 的通道 16 上，所以，我们在设置好 ADC1 之后只要读取通道 16 的值，就是温度传感器返回来的电压值了。

根据这个值，我们就可以计算出当前温度。

配置代码如下：

```
RCC->APB2ENR|=1<<9;       //ADC1 时钟使能
RCC->APB2RSTR|=1<<9;      //ADC1 复位
RCC->APB2RSTR&=~(1<<9);//复位结束
RCC->CFGR&=~(3<<14);      //分频因子清零
//SYSCLK/DIV2=12M ADC 时钟设置为 12M,ADC 最大时钟不能超过 14M!
//否则将导致 ADC 准确度下降!
RCC->CFGR|=2<<14;
ADC1->CR1&=0XF0FFFF;      //工作模式清零
ADC1->CR1|=0<<16;         //独立工作模式
ADC1->CR1&=~(1<<8);       //非扫描模式
ADC1->CR2&=~(1<<1);       //单次转换模式
```

```
ADC1->CR2&=~(7<<17);
ADC1->CR2|=7<<17;          //软件控制转换
ADC1->CR2|=1<<20;          //使用用外部触发(SWSTART)!!! 必须使用一个事件来触发
ADC1->CR2&=~(1<<11);     //右对齐

ADC1->CR2|=1<<23;          //使能温度传感器

ADC1->SQR1&=~(0XF<<20);
ADC1->SQR1|=0<<20;         //1 个转换在规则序列中  也就是只转换规则序列 1

ADC1->SMPR1&=~(7<<3*6);//清除通道 16 原来的设置
ADC1->SMPR1|=7<<(3*6); //通道 16    239.5 周期,提高采样时间可以提高精确度

ADC1->CR2|=1<<0;           //开启 AD 转换器
ADC1->CR2|=1<<3;           //使能复位校准
while(ADC1->CR2&1<<3); //等待校准结束
   //该位由软件设置并由硬件清除。在校准寄存器被初始化后该位将被清除。

ADC1->CR2|=1<<2;           //开启 AD 校准
while(ADC1->CR2&1<<2);   //等待校准结束
//该位由软件设置以开始校准，并在校准结束时由硬件清除
```

2.12.4 实验步骤

1）连接仿真器 USB 线，注意此时 USB 线不要和 PC 连接
2）连接仿真和开发板之间的 20 芯排线
3）仿真器 USB 线插入电脑 USB 接口
4）下载程序并调试，分析并检验实验结果。

2.12.5 考核方式

学生应熟练掌握 AD 转换的配置及规则，以便能应用于其他 demo 板上测量电压等功能。

2.13 DAC 实验

4.13.1 实验目的

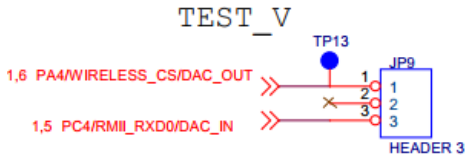
数字模拟转换是一种将数字信号转换为模拟信号。本实验要求学生掌握 DAC 的使用及其转换过程与规则。本实验涉及 RCC，GPIO，DAC，RCC 的配置

2.13.2 实验要求

掌握 DAC 原理及控制方法，通过万用表测试输出电压的正确性，或者通过 AD 通道读取 DAC 输出的电压值，验证 DA 是否正确。

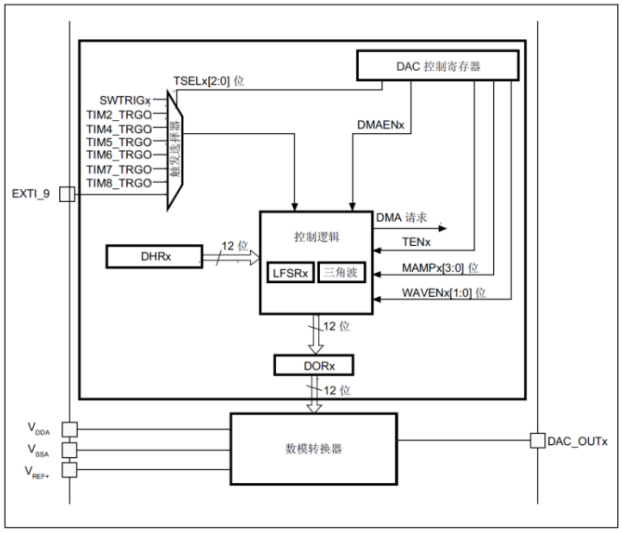
2.13.3 实验原理：

STM32 的 DAC 模块(数字/模拟转换模块)是 12 位数字输入，电压输出型的 DAC。DAC 可以配置为 8 位或 12 位模式，也可以与 DMA 控制器配合使用。DAC 工作在 12 位模式时，数据可以设置成左对齐或右对齐。DAC 模块有 2 个输出通道，每个通道都有单独的转换器。在双 DAC 模式下，2 个通道可以独立地进行转换，也可以同时进行转换并同步地更新 2 个通道的输出。实验板电路如图所示，DA 输出引脚为 PA4。输出电压可以通过 TP13 使用万用表进行测试，也可连接到 JP9 的 3 脚，利用 CPU 的 AD 输入进行检测。



STM32 的 DAC 模块主要特点有：

① 2 个 DAC 转换器：每个转换器对应 1 个输出通道
② 8 位或者 12 位单调输出
③ 12 位模式下数据左对齐或者右对齐
④ 同步更新功能
⑤ 噪声波形生成
⑥ 三角波形生成
⑦ 双 DAC 通道同时或者分别转换
⑧ 每个通道都有 DMA 功能



DAC 通道模块框图

配置代码如下：

```
RCC->APB2ENR|=1<<2;          //使能 PORTA 时钟
RCC->APB1ENR|=1<<29;         //使能 DAC 时钟

GPIOA->CRL&=0XFFF0FFFF;
GPIOA->CRL|=0X00000000;      //PA4  模拟输入
DAC->CR|=1<<0;               //使能 DAC1
DAC->CR|=1<<1;               //DAC1 输出缓存不使能  BOFF1=1
DAC->CR|=0<<2;               //不使用触发功能  TEN1=0
DAC->CR|=0<<3;               //DAC TIM6 TRGO,不过要 TEN1=1 才行
DAC->CR|=0<<6;               //不使用波形发生
DAC->CR|=0<<8;               //屏蔽、幅值设置
DAC->CR|=0<<12;              //DAC1 DMA 不使能
DAC->DHR12R1=0;              //数据保持寄存器置 0
```

2.13.4 实验步骤

1）连接仿真器 USB 线，注意此时 USB 线不要和 PC 连接
2）连接仿真和开发板之间的 20 芯排线
3）仿真器 USB 线插入电脑 USB 接口
4）下载程序并调试，分析并检验实验结果。

2.13.5 考核方式

学生应熟练掌握 DAC，了解 DAC 的具体配置及工作过程，能在实验板上实现输出想要的输出电压。

2.14 DMA 实验

2.14.1 实验目的

DMA 允许不同速度的硬件装置来沟通，而不需要依赖于 CPU 的大量中断负载。本实验要求学生学会使用 DMA 来发送数据。本实验涉及 DMA，USART，GPIO，RCC 的配置。

2.14.2 实验要求

打开串口助手，按下 SW17 键，串口输出数据。要求学生了解 DMA 的工作方式，体会 DMA 传输的好处。

2.14.3 实验原理

DMA，全称为：Direct Memory Access，即直接存储器访问。DMA 传输方式无需 CPU 直接控制传输，也没有中断处理方式那样保留现场和恢复现场的过程，通过硬件为 RAM 与 I/O 设备开辟一条直接传送数据的通路，能使 CPU 的效率大为提高。

STM32 最多有 2 个 DMA 控制器（DMA2 仅存在大容量产品中），DMA1 有 7 个通道。DMA2 有 5 个通道。每个通道专门用来管理来自于一个或多个外设对存储器访问的请求。还有一个仲裁起来协调各个 DMA 请求的优先权。

STM32 的 DMA 有以下一些特性：

●每个通道都直接连接专用的硬件 DMA 请求，每个通道都同样支持软件触发。这些功能通过软件来配置。

●在七个请求间的优先权可以通过软件编程设置(共有四级：很高、高、中等和低)，假如在相等优先权时由硬件决定(请求 0 优先于请求 1，依此类推) 。

●独立的源和目标数据区的传输宽度(字节、半字、全字)，模拟打包和拆包的过程。源和目标地址必须按数据传输宽度对齐。

●支持循环的缓冲器管理

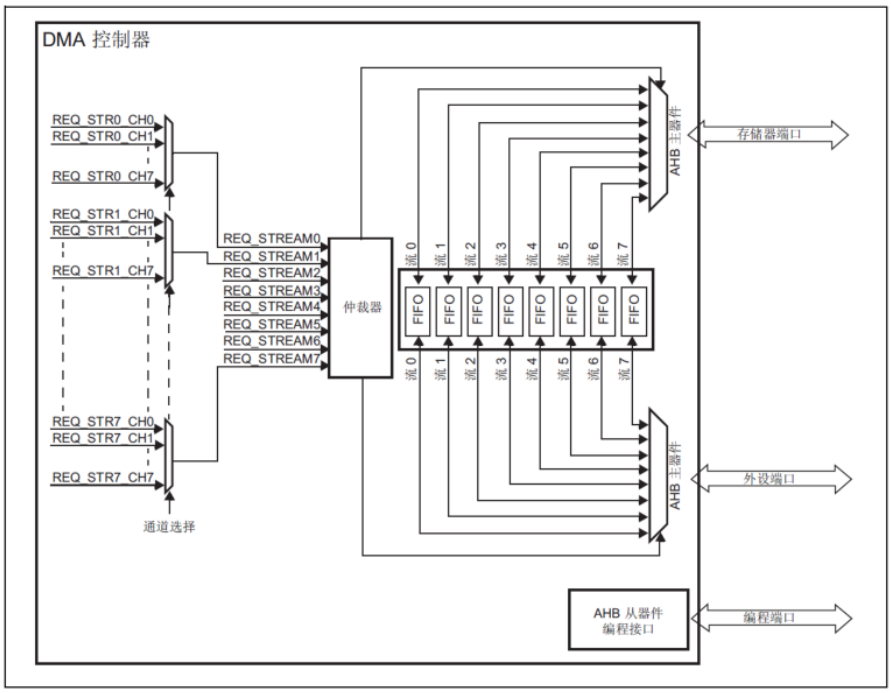
●每个通道都有 3 个事件标志(DMA 半传输，DMA 传输完成和 DMA 传输出错)，这 3 个事件标志逻辑或成为一个单独的中断请求。

●存储器和存储器间的传输

●外设和存储器，存储器和外设的传输

●闪存、SRAM、外设的 SRAM、APB1 APB2 和 AHB 外设均可作为访问的源和目标。

●可编程的数据传输数目：最大为 65536



MYDMA_Config(DMA1_Channel4,(u32)&USART1->DR,(u32)SendBuff,SEND_BUF_SIZE);//DMA1 通道 4,外设为串口 1,存储器为 SendBuff,长度 SEND_BUF_SIZE.

```
//DMA1 的各通道配置
//这里的传输形式是固定的,这点要根据不同的情况来修改
//从存储器->外设模式/8 位数据宽度/存储器增量模式
//DMA_CHx:DMA 通道 CHx
//cpar:外设地址
//cmar:存储器地址
//cndtr:数据传输量
void MYDMA_Config(DMA_Channel_TypeDef*DMA_CHx,u32 cpar,u32 cmar,u16 cndtr)
{
 RCC->AHBENR|=1<<0;                    //开启 DMA1 时钟
 delay_ms(5);                          //等待 DMA 时钟稳定
 DMA_CHx->CPAR=cpar;                   //DMA1  外设地址
 DMA_CHx->CMAR=(u32)cmar;              //DMA1,存储器地址
 DMA1_MEM_LEN=cndtr;                   //保存 DMA 传输数据量
 DMA_CHx->CNDTR=cndtr;                 //DMA1,传输数据量
 DMA_CHx->CCR=0X00000000;              //复位
 DMA_CHx->CCR|=1<<4;                   //从存储器读
 DMA_CHx->CCR|=0<<5;                   //普通模式
 DMA_CHx->CCR|=0<<6;                   //外设地址非增量模式
 DMA_CHx->CCR|=1<<7;                   //存储器增量模式
 DMA_CHx->CCR|=0<<8;                   //外设数据宽度为 8 位
 DMA_CHx->CCR|=0<<10;                  //存储器数据宽度 8 位
```

```
    DMA_CHx->CCR|=1<<12;          //中等优先级
    DMA_CHx->CCR|=0<<14;          //非存储器到存储器模式
}
```

2.14.4 实验步骤

1）连接仿真器 USB 线，注意此时 USB 线不要和 PC 连接
2）连接仿真和开发板之间的 20 芯排线
3）仿真器 USB 线插入电脑 USB 接口
4）使用串口线，连接左边串口至电脑
5）下载程序并调试，分析并检验实验结果。

2.14.5 考核方式

学生可以改变 DMA 传输的数据，或者使用 DMA 的其他工作模式。比如 DMA 和另一个串口相连，或者 DMA 连接串口与串口。

2.15 SPI 实验

2.15.1 实验目的

要求学生学会使用 SPI 的通讯方式，使用 SPI 来传输数据。本实验涉及 SPI，GPIO，RCC 的配置。

2.15.2 实验要求

往 w25q80 芯片的特定位置写入数据，并读出数据，验证读写功能是否正确。实验中可先读取 w25q80 芯片 ID 号，确保读功能正确，然后在调试写功能。

2.15.3 实验原理

SPI，是一种高速的，全双工，同步的通信总线，并且在芯片的管脚上只占用四根线，节约了芯片的管脚。SPI 接口一般使用 4 条线通信：

MISO 主设备数据输入，从设备数据输出。

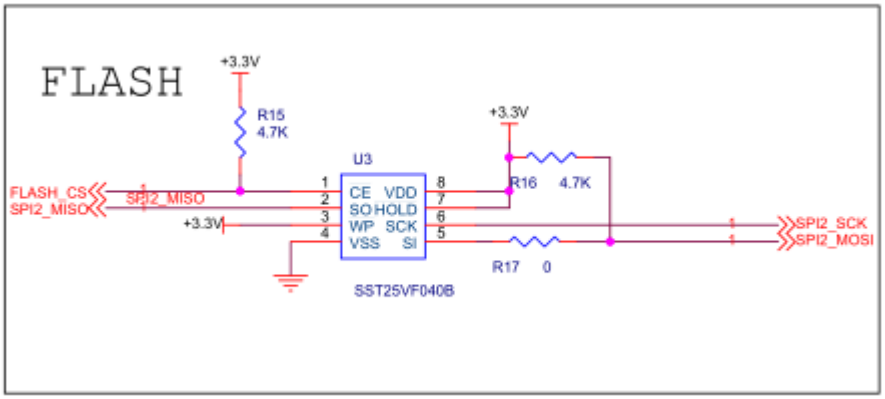
MOSI 主设备数据输出，从设备数据输入。

SCLK 时钟信号，由主设备产生。

CS 从设备片选信号，由主设备控制。

SPI 主要特点有：可以同时发出和接收串行数据；可以当作主机或从机工作；提供频率可编程时钟；发送结束中断标志；写冲突保护；总线竞争保护等。

SPI 总线四种工作方式 SPI 模块为了和外设进行数据交换，根据外设工作要求，其输出串行同步时钟极性和相位可以进行配置，时钟极性（CPOL）对传输协议没有重大的影响。如果 CPOL=0，串行同步时钟的空闲状态为低电平；如果 CPOL=1，串行同步时钟的空闲状态为高电平。时钟相位（CPHA）能够配置用于选择两种不同的传输协议之一进行数据传输。如果 CPHA=0，在串行同步时钟的第一个跳变沿（上升或下降）数据被采样；如果 CPHA=1，在串行同步时钟的第二个跳变沿（上升或下降）数据被采样。SPI 主模块和与之通信的外设备时钟相位和极性应该一致。



```
RCC->APB2ENR |= 0x11<<3;          // PORTB、PORTC 时钟使能
RCC->APB1ENR |= 0x01<<14;         // SPI2 时钟使能
GPIOB->CRH &= 0X000FFFFF;
GPIOB->CRH |= 0XBBB00000;         // PB13/14/15 复用
GPIOB->ODR |= 0X7<<13;            // PB13/14/15 上拉
GPIOC->CRL &= 0XFFFF0FFF;
GPIOC->CRL |= 0X00003000;         // PC3  通用推挽输出
SPI2->CR1 |= 0<<10;               //  全双工模式
SPI2->CR1 |= 1<<9;                //  软件 nss 管理
SPI2->CR1 |= 1<<8;                // NSS 引脚高电平
SPI2->CR1 |= 1<<2;                // SPI 主机
SPI2->CR1 |= 0<<11;               // 8bit 数据格式
SPI2->CR1 |= 1<<1;                //  空闲模式下 SCK 为 1 CPOL=1
SPI2->CR1 |= 1<<0;                //  数据采样从第二个时间边沿开始,CPHA=1
    //对 SPI2 属于 APB1 的外设.时钟频率最大为 36M.
SPI2->CR1 |= 3<<3;                // Fsck=Fpclk1/256
SPI2->CR1 |= 0<<7;                // MSBfirst
SPI2->CR1 |= 1<<6;                // SPI 设备使能
```

2.15.4 实验步骤

1）连接仿真器 USB 线，注意此时 USB 线不要和 PC 连接
2）连接仿真和开发板之间的 20 芯排线
3）仿真器 USB 线插入电脑 USB 接口
4）下载程序并调试，分析并检验实验结果。

2.15.5 考核方式

需要学生掌握 SPI 的 通讯方式，并能够按照数据手册，编写出该芯片的发送数据与读数据的通讯过程。

2.16 I²C 实验

2.16.1 实验目的

I²C 总线是双向、两线(SCL、SDA)、串行、多主控(multi-master)接口标准，具有总线仲裁机制，非常适合在器件之间进行近距离、非经常性的数据通信。本实验要求学生学会使用 I2C 来通讯和传输数据。本实验涉及 I2C，GPIO，RCC 的配置

2.16.2 实验要求

掌握 I²C 通讯的时序，了解 I²C 数据发送接受的流程。明白 I²C 主从模式的区别。根据 AT34C02 的 datasheet 编写 I²C 芯片读写控制。

2.16.3 实验原理

I²C 总线接口是一种常见的串行总线接口，它提供多主机功能，控制所有 I²C 总线特定的时序、协议、仲裁和定时。支持标准和快速两种模式，同时与 SMBus 2.0 兼容。

I²C 模块有多种用途，包括 CRC 码的生成和校验、SMBus(系统管理总线—System ManagementBus)和 PMBus(电源管理总线—Power Management Bus)。

根据特定设备的需要，可以使用 DMA 以减轻 CPU 的负担。

I²C (Inter－Integrated Circuit)总线是一种由 PHILIPS 公司开发的两线式串行总线，用于连接微控制器及其外围设备。它是由数据线 SDA 和时钟 SCL 构成的串行总线，可发送和接收数据。

在 CPU 与被控 IC 之间、IC 与 IC 之间进行双向传送，高速 IIC 总线一般可达 400kbps 以上。

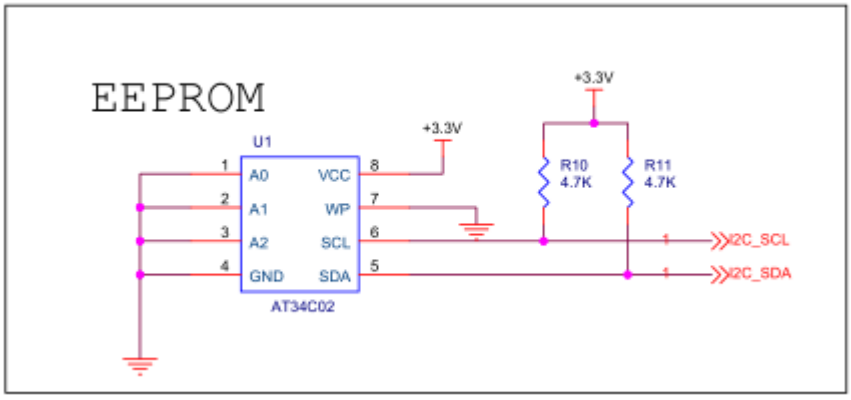
I²C 总线在传送数据过程中共有三种类型信号， 它们分别是：开始信号、结束信号和应答信号。

开始信号：SCL 为高电平时，SDA 由高电平向低电平跳变，开始传送数据。

结束信号：SCL 为高电平时，SDA 由低电平向高电平跳变，结束传送数据。

应答信号：接收数据的 IC 在接收到 8bit 数据后，向发送数据的 IC 发出特定的低电平脉冲，表示已收到数据。CPU 向受控单元发出一个信号后，等待受控单元发出一个应答信号，CPU 接收到应答信号后，根据实际情况作出是否继续传递信号的判断。若未收到应答信号，由判断为受控单元出现故障。

这些信号中，起始信号是必需的，结束信号和应答信号，都可以不要。



I²C 总线接口配置代码如下:

```
RCC->APB2ENR|=1<<3;          //先使能外设 IO PORTB 时钟
GPIOB->CRL&=0X00FFFFFF;   //PB6,7  推挽输出
GPIOB->CRL|=0X33000000;
GPIOB->ODR|=3<<6;          //PB6,7  输出高
```

2.16.4 实验步骤

1）连接仿真器 USB 线，注意此时 USB 线不要和 PC 连接
2）连接仿真和开发板之间的 20 芯排线
3）仿真器 USB 线插入电脑 USB 接口
4）下载程序并调试，分析并检验实验结果。

2.16.5 考核方式

考查学生是否掌握 I²C 的通讯过程，并根据 I²C 存储器芯片的手册编写数据写入和读出（注意需要考虑是否组要擦除）。

2.17 触摸屏实验

2.17.1 实验目的

本实验要求学生学会使用触摸屏，了解触摸屏的工作原理。本实验涉及 SPI（IO 接口模拟），GPIO，SYSCFG，EXTI，NVIC，RCC 的配置

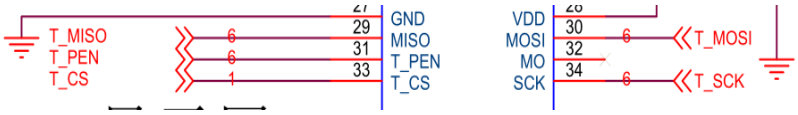
2.17.2 实验要求

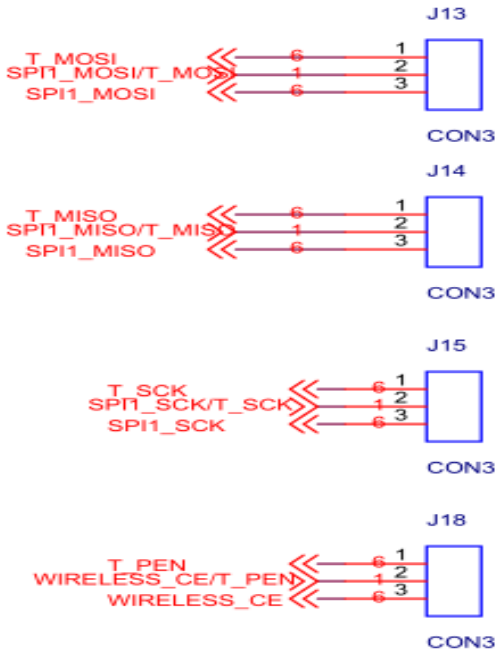
正确检测触摸屏的触摸事件，并能根据触摸的位置，获取其 AD 值，转换成对应的位置信息。

2.17.3 实验原理：

电阻式触摸屏利用压力感应进行触点检测控制，需要直接应力接触，通过检测电阻来定位触摸位置。ALIENTEK 2.4/2.8/3.5 寸 TFTLCD 模块自带的触摸屏都属于电阻式触摸屏，下面简单介绍下电阻式触摸屏的原理。

电阻触摸屏的主要部分是一块与显示器表面非常配合的电阻薄膜屏，这是一种多层的复合薄膜，它以一层玻璃或硬塑料平板作为基层，表面涂有一层透明氧化金属（透明的导电电阻）导电层，上面再盖有一层外表面硬化处理、光滑防擦的塑料层、它的内表面也涂有一层涂层、在他们之间有许多细小的（小于 1/1000 英寸）的透明隔离点把两层导电层隔开绝缘。当手指触摸屏幕时，两层导电层在触摸点位置就有了接触，电阻发生变化，在 X 和 Y 两个方向上产生信号，然后送触摸屏控制器。控制器侦测到这一接触并计算出（X，Y）的位置，再根据获得的位置模拟鼠标的方式运作。这就是电阻技术触摸屏的最基本的原理。电阻触摸屏的优点：精度高、价格便宜、抗干扰能力强、稳定性好。电阻触摸屏的缺点：容易被划伤、透光性不太好、不支持多点触摸。





触摸屏幕按下是，产生外部中断，来表明产生了触摸。外部中断原理可看外部中断实验。

```
//-----------------配置触发 IO 口-----------------//
    RCC->APB2ENR|=1<<4;
    GPIOC->CRL&=0XFFFFF0FF;                    //PC2 输入（触摸屏画笔引脚）
    GPIOC->CRL|=0X00000800;                    //上拉输入
//-----------------配置外部中断-----------------//
    RCC->APB2ENR|=1<<0;                        //开启辅助时钟
    AFIO->EXTICR[(uint8_t)0x02 >> 0x02] &= ~(((uint32_t)0x0F) << (0x04 *
((uint8_t)0x02 & (uint8_t)0x03)));          // 设置外部中断为 EXIT2
    AFIO->EXTICR[(uint8_t)0x02 >> 0x02] |= (((uint32_t)(uint8_t)0x02) << (0x04 *
((uint8_t)0x02 & (uint8_t)0x03)));
    EXTI->IMR &= ~((uint32_t)0x00004);
    EXTI->EMR &= ~((uint32_t)0x00004);
    *(__IO uint32_t *)(EXTI_BASE + (uint32_t)0x00)|= ((uint32_t)0x00004);
    EXTI->RTSR &= ~((uint32_t)0x00004);        // 清除边沿触发设置
    EXTI->FTSR &= ~((uint32_t)0x00004);
    EXTI->RTSR |= ((uint32_t)0x00004);         // 设置为下降沿触发
    EXTI->FTSR |= ((uint32_t)0x00004);
//-----------------配置中断优先级-----------------//
    uint32_t tmppriority = 0x00, tmppre = 0x00, tmpsub = 0x0F;

    SCB->AIRCR = ((uint32_t)0x05FA0000) | ((uint32_t)0x500);        // 设置组别
    tmppriority = (0x700 - ((SCB->AIRCR) & (uint32_t)0x700))>> 0x08;
    tmppre = (0x4 - tmppriority);
    tmpsub = tmpsub >> tmppriority;

    tmppriority = (uint32_t)0 << tmppre;
    tmppriority |=  0 & tmpsub;
```

```
tmppriority = tmppriority << 0x04;
NVIC->IP[8] = tmppriority;
NVIC->ISER[8 >> 0x05] = (uint32_t)0x01 << (8 & (uint8_t)0x1F);  // 使能通道
```

本实验触摸屏采用 SPI 接口与 CPU 进行通讯，本实验要求 CPU 侧根据触摸屏时序通过 IO 接口控制方式完成 SPI 通讯。

2.17.4 实验步骤

1）连接仿真器 USB 线，注意此时 USB 线不要和 PC 连接
2）连接仿真和开发板之间的 20 芯排线
3）仿真器 USB 线插入电脑 USB 接口
4）下载程序并调试，分析并检验实验结果。

2.17.5 考核方式

考核掌握触摸屏的触摸检测和位置信息获取。

2.18 LCD 显示实验

2.18.1 实验目的

TFT-LCD 即薄膜晶体管液晶显示器。其英文全称为：Thin Film Transistor-Liquid Crystal Display。TFT-LCD 与无源 TN-LCD、STN-LCD 的简单矩阵不同，它在液晶显示屏的每一个象素上都设置有一个薄膜晶体管（TFT），可有效地克服非选通时的串扰，使显示液晶屏的静态特性与扫描线数无关，因此大大提高了图像质量。本实验需要学生学会使用液晶屏。本实验涉及 FSMC，GPIO，RCC 的配置。

2.18.2 实验要求

屏幕全白，右下角有一红点，要求学生看懂液晶屏的 datasheet，掌握部分简单的液晶屏命令，同时对 FSMC 的存储方式有了解。

2.18.3 实验原理

1、液晶彩屏原理简介

LCD，即液晶显示器，因为其功耗低、体积小，承载的信息量大，因而被广泛用于信息输出、与用户进行交互，目前仍是各种电子显示设备的主流。 因为 STM32 内部没有集成专用的液晶屏和触摸屏的控制接口，所以在显示面板中应自带含有这些驱动芯片的驱动电路(液晶屏和触摸屏的驱动电路是独立的)，STM32 芯片通过驱动芯片来控制液晶屏和触摸屏。以 STM32F 207 开发板 2.8 寸液晶屏(240*320)为例，它使用 ILI9320 芯片控制液晶屏。

液晶屏的控制芯片内部结构相对比较复杂，液晶屏的彩屏驱动电路最主要的是位于中间 GRAM(Graphics RAM)，可以理解为显存。GRAM 就好比是一个彩屏数据缓冲 buffer，我们可以把大批的显示内容以显示矩阵的形式写到 buffer 里，让彩屏 LCD 来读取 buffer 里的数据再由彩屏驱动芯片显示到显示屏上，随着 GRAM 逐渐丰富和完善，除了显示矩阵以外还放很多的命令，GRAM 中每个存储单元都对应着液晶面板的一个像素点。它右侧的各种模块共同作用把 GRAM 存储单元的数据转化成液晶面板的控制信号，使像素点呈现特定的颜色，而像素点组合起来则成为一幅完整的图像。到现在，这些驱动/控制电路以及 buffer 都合起来放在一片芯片中，统一被称为 driver IC，这个 driver IC 就是上一段我们所说的 ILI9320 的 driver IC 芯片。

ILI9320 采用的是 16 位控制模式，以 16 位描述的像素点。ILI9320 最高能够控制 18 位的 LCD，但为了数据传输简便，我们采用它的 16 位控制模式，以 16 位描述的像素点。按照标准格式，16 位的像素点的三原色描述的位数为 R：G：B =5：6：5，描述绿色的位数较多是因为人眼对绿色更为敏感。

具体操作方式可参考 ILI9320 数据手册。

2、FSMC 介绍

STM32 系列中内部集成 256 KB 以上 Flash，后缀为 xC、xD 和 xE 的高存储密度微控制器特有的存储控制机制，一般芯片引脚数在 100 脚以上的 STM32F103 芯片都带有 FSMC 接口，FSMC 能够根据不同的外部存储器类型，发出相应的数据/地址/控制信号类型以匹配信

号的速度，从而使得 STM32 系列微控制器不仅能够应用各种不同类型、不同速度的外部静态存储器，而且能够在不增加外部器件的情况下同时扩展多种不同类型的静态存储器，满足系统设计对存储容量、产品体积以及成本的综合要求。

从上图我们可以看出，STM32 的 FSMC 将外部设备分为 3 类：NOR/PSRAM 设备、NAND 设备、PC 卡设备。他们共用地址数据总线等信号，他们具有不同的 CS 以区分不同的设备。

FSMC 主要的优势有以下几点：

1）支持多种静态存储器类型。STM32 通过 FSMC 可以与 SRAM、ROM、PSRAM、NOR Flash 和 NANDFlash 存储器的引脚直接相连。

2）支持丰富的存储操作方法。FSMC 不仅支持多种数据宽度的异步读/写操作，而且支持对 NOR/PSRAM/NAND 存储器的同步突发访问方式。

3）支持同时扩展多种存储器。FSMC 的映射地址空间中，不同的 BANK 是独立的，可用于扩展不同类型的存储器。当系统中扩展和使用多个外部存储器时，FSMC 会通过总线悬空延迟时间参数的设置，防止各存储器对总线的访问冲突。

4）支持更为广泛的存储器型号。通过对 FSMC 的时间参数设置，扩大了系统中可用存储器的速度范围，为用户提供了灵活的存储芯片选择空间。

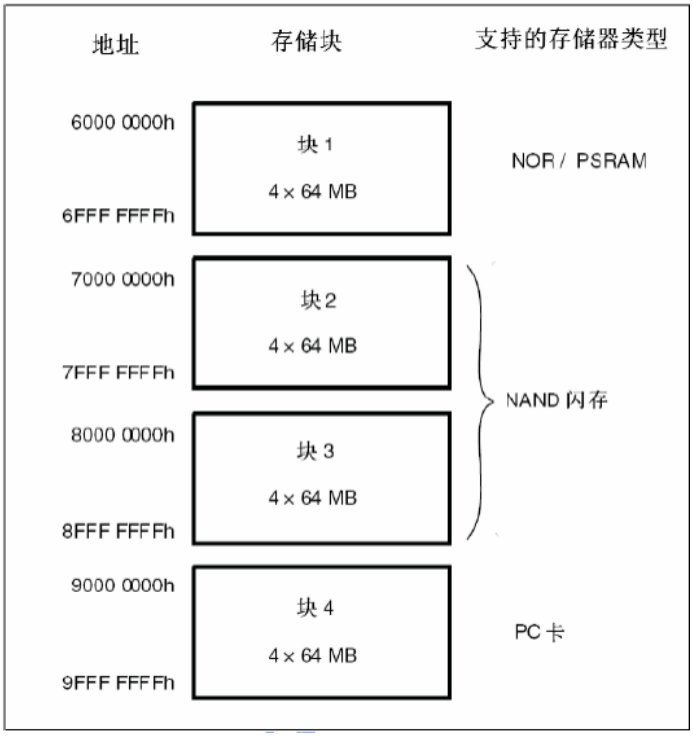
5）支持代码从 FSMC 扩展的外部存储器中直接运行，而不需要首先调入内部 SRAM。

这里我们介绍下为什么可以把 TFTLCD 当成 SRAM 设备用：首先我们了解下外部 SRAM 的连接，外部 SRAM 的控制一般有：地址线（如 A0~A18）、数据线（如 D0~D15）、写信号（WR）、读信号（OE）、片选信号（CS），如果 SRAM 支持字节控制，那么还有 UB/LB 信号。而 TFTLCD 的信号包括：RS、D0~D15、WR、RD、CS、RST 和 BL 等，其中真正在操作 LCD 的时候需要用到的就只有：RS、D0~D15、WR、RD 和 CS；

这里比较关键点就是 TFT LCD 并不需要用到 FSMC 的地址线（如 A0~A18），它只需要用到 16 根数据线以及一些其他的控制信号线就可以了，并且操作时序和 SRAM 的控制完全类似，下面是对比表：

资源	SRAM	TFT LCD
写信号（WE）	有	有
读信号（OE）	有	有
片选信号（CS）	有	有
数据线（D0~D15）	有	有
地址线（如A0~A18）	有	无
RS（TFT中决定是数据还是命令）	无	有
RST	无	有
BL	无	有

TFT LCD 通过 RS 信号来决定传送的数据是数据还是命令，本质上可以理解为一个地址信号，比如我们把 RS 接在 A0 上面，那么当 FSMC 控制器写的地址刚好让 A0 这根信号线为 0（A1~A18 因为并没有连接到 TFT 上，所以可以忽略不计）的时候，会使得连在 A0 上的 TFT 信号 RS 也为 0，对 TFT LCD 来说，就是写命令；同样，当 FSMC 写地址的地址让 A0 这个地址变成 1 的时候，A0 上的 TFT 信号 RS 也会变成 1，对 TFTLCD 来说，就是写数据了。这样，就把数据和命令区分开了，当然 RS 也可以接在其他地址线上，STM32 神舟开发板是把 RS 连接在 FSMC_A0 上面的。 STM32 的 FSMC 支持 8/16/32 位数据宽度，我们这里用到的 LCD 是 16 位宽度的，所以在配置 FMSC 控制器属性的时候，选择 16 位宽就 OK 了。我们再来看看 FSMC 的外部设备地址映像，STM32 的 FSMC 将外部存储器划分为固定大小为 256M 字节的四个存储块，如下图：

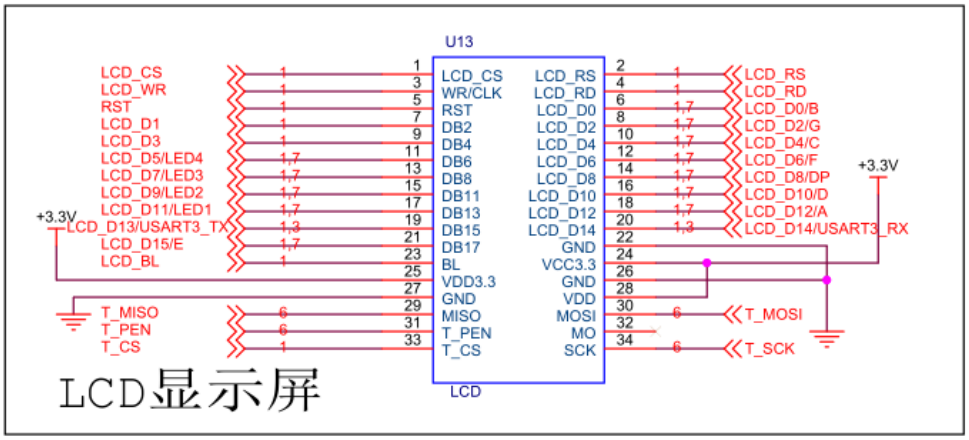


可以看到，我们需要操作的 TFT 彩屏与操作 SRAM 是相同的控制器，都是负责存储块的 Bank1 的内部控制器 NOR FLASH 控制器，对于 NOR FLASH 控制器，主要是通过 FSMC_BCRx（x=1、2、3、4）、FSMC_BTRx（x=1、2、3、4）和 FSMC_BWTRx（x=1、2、3、4）寄存器设置（其中 x=1~4，对应 Bank 中的 4 个区），下面介绍一下这几个配置寄存器各负责的功能：

FSMC_BCRx（x=1，2，3，4）：SRAM/NOR 闪存片选控制寄存器（FSMC_BCR1，FSMC_BCR2， FSMC_BCR3， FSMC_BCR4）每个 Bank 对应 1 个寄存器，这个寄存器包含了每个存储器块的控制信息，可以用于 SRAM、ROM、异步或成组传输的 NOR 闪存存储器（TFT LCD 也可以），主要包括设置数据总线宽度，比如 8 位、16 位；存储类型的设置，比如 SRAM 或者 PSRAM 或者 NOR 闪存；地址/数据是否复用；存储块使能设置。

FSMC_BTRx（x=1，2，3，4）：闪存片写时序寄存器（FSMC_BTR1，FSMC_BTR2，FSMC_BTR3，FSMC_BTR4）；可以通过这个寄存器设置模式，总线恢复时间，时钟分频，数据保持时间等时序相关的设置。

FSMC_BWTRx （x=1，2，3，4） ： 闪存写时序寄存器 （FSMC_BWTR1，FSMC_BWTR2，FSMC_BWTR3，FSMC_BWTR4） ；采用哪种访问模式，数据保持时间，地址保持时间，底子好建立时间（单位是 HCLK 时钟周期）




```
Void LCD_Configuration()
{
    RCC->APB2ENR|=7<<4;          //使能 PORTC,D,E 时钟
    GPIOD->CRL&=0X0F00FFFF;
    GPIOD->CRL|=0X30330000;
    GPIOD->CRH&=0XFFF00FFF;
    GPIOD->CRH|=0X00033000;   //PD4 5 7 11 12 推挽输出
    GPIOD->ODR|=0X1000;          //PD12 置高

    GPIOC->CRL&=0XFFFFFFF0;
    GPIOC->CRL|=0X00000003;              //PC0 推挽输出
}

void LCD_Config_DIN(void)
{
    GPIOD->CRL=0x34334488;
    GPIOD->CRH=0x88433888;
    GPIOD->BRR=(uint16_t)0xC703;

    GPIOE->CRL=0X84444444;
    GPIOE->CRH=0X88888888;
    GPIOE->BRR=(uint16_t)0xff80;
}

//   以下为液晶屏初始化过程，可按照数据手册对应查找命令含义
LCD_IO_WriteReg(0xCF);
LCD_IO_WriteData(0x00);
LCD_IO_WriteData(0xC1);
LCD_IO_WriteData(0X30);
LCD_IO_WriteReg(0xED);
LCD_IO_WriteData(0x64);
LCD_IO_WriteData(0x03);
```

```
LCD_IO_WriteData(0X12);
LCD_IO_WriteData(0X81);
LCD_IO_WriteReg(0xE8);
LCD_IO_WriteData(0x85);
LCD_IO_WriteData(0x10);
LCD_IO_WriteData(0x7A);
LCD_IO_WriteReg(0xCB);
LCD_IO_WriteData(0x39);
LCD_IO_WriteData(0x2C);
LCD_IO_WriteData(0x00);
LCD_IO_WriteData(0x34);
LCD_IO_WriteData(0x02);
LCD_IO_WriteReg(0xF7);
LCD_IO_WriteData(0x20);
LCD_IO_WriteReg(0xEA);
LCD_IO_WriteData(0x00);
LCD_IO_WriteData(0x00);
LCD_IO_WriteReg(0xC0);       //Power control
LCD_IO_WriteData(0x1B);      //VRH[5:0]
LCD_IO_WriteReg(0xC1);       //Power control
LCD_IO_WriteData(0x01);      //SAP[2:0];BT[3:0]
LCD_IO_WriteReg(0xC5);       //VCM control
LCD_IO_WriteData(0x30);     //3F
LCD_IO_WriteData(0x30);     //3C
LCD_IO_WriteReg(0xC7);       //VCM control2
LCD_IO_WriteData(0XB7);
LCD_IO_WriteReg(0x36);       // Memory Access Control
LCD_IO_WriteData(0x48);
LCD_IO_WriteReg(0x3A);
LCD_IO_WriteData(0x55);
LCD_IO_WriteReg(0xB1);
LCD_IO_WriteData(0x00);
LCD_IO_WriteData(0x1A);
LCD_IO_WriteReg(0xB6);       // Display Function Control
LCD_IO_WriteData(0x0A);
LCD_IO_WriteData(0xA2);
LCD_IO_WriteReg(0xF2);       // 3Gamma Function Disable
LCD_IO_WriteData(0x00);
LCD_IO_WriteReg(0x26);       //Gamma curve selected
LCD_IO_WriteData(0x01);
LCD_IO_WriteReg(0xE0);       //Set Gamma
LCD_IO_WriteData(0x0F);
LCD_IO_WriteData(0x2A);
LCD_IO_WriteData(0x28);
```

```
LCD_IO_WriteData(0x08);
LCD_IO_WriteData(0x0E);
LCD_IO_WriteData(0x08);
LCD_IO_WriteData(0x54);
LCD_IO_WriteData(0XA9);
LCD_IO_WriteData(0x43);
LCD_IO_WriteData(0x0A);
LCD_IO_WriteData(0x0F);
LCD_IO_WriteData(0x00);
LCD_IO_WriteData(0x00);
LCD_IO_WriteData(0x00);
LCD_IO_WriteData(0x00);
LCD_IO_WriteReg(0XE1);       //Set Gamma
LCD_IO_WriteData(0x00);
LCD_IO_WriteData(0x15);
LCD_IO_WriteData(0x17);
LCD_IO_WriteData(0x07);
LCD_IO_WriteData(0x11);
LCD_IO_WriteData(0x06);
LCD_IO_WriteData(0x2B);
LCD_IO_WriteData(0x56);
LCD_IO_WriteData(0x3C);
LCD_IO_WriteData(0x05);
LCD_IO_WriteData(0x10);
LCD_IO_WriteData(0x0F);
LCD_IO_WriteData(0x3F);
LCD_IO_WriteData(0x3F);
LCD_IO_WriteData(0x0F);
LCD_IO_WriteReg(0x2B);
LCD_IO_WriteData(0x00);
LCD_IO_WriteData(0x00);
LCD_IO_WriteData(0x01);
LCD_IO_WriteData(0x3f);
LCD_IO_WriteReg(0x2A);
LCD_IO_WriteData(0x00);
LCD_IO_WriteData(0x00);
LCD_IO_WriteData(0x00);
LCD_IO_WriteData(0xef);
LCD_IO_WriteReg(0x11); //Exit Sleep
Delay_ms(120);
LCD_IO_WriteReg(0x29); //display on;
```

2.18.4 实验步骤

1）连接仿真器 USB 线，注意此时 USB 线不要和 PC 连接
2）连接仿真和开发板之间的 20 芯排线
3）仿真器 USB 线插入电脑 USB 接口
4）下载程序并调试，分析并检验实验结果。

2.18.5 考核方式

学生需要实现画点画线画圆，对 fsmc 要有一定理解，并结合触摸屏做实验。