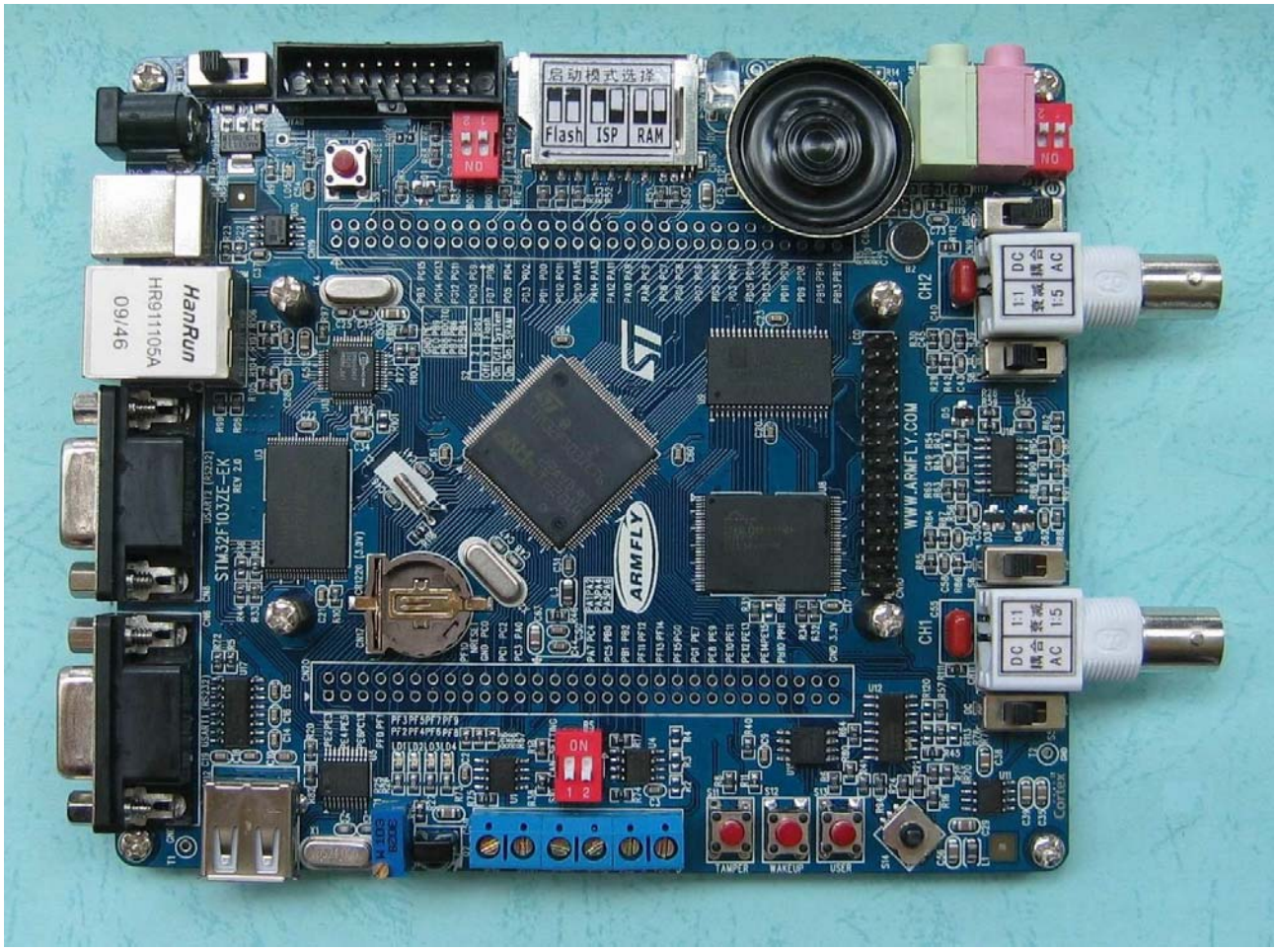
安富莱STM32开发板（V2标准版）

用户手册

版本：V2.5



安富莱电子开发网

WWW.ARMFLY.COM

声　　明

安富莱STM32F103ZE-EK开发板是由安富莱电子自主设计开发的一款产品。该产品不仅包括开发板硬件，也包括完整的软件开发环境、软件例程以及售后技术支持。

本文档为安富莱STM32F103ZE-EK开发板配套的用户手册，详细介绍了开发板的使用方法。

配套光盘中提供的原理图、软件例程、文档和开发软件仅用于学习和设计参考，我们不承担客户将相关资料用于产品时所带来的一切风险，这些风险包括产品质量问题、版权问题。

购买开发板之后，您可以加入售后 QQ 群（群内成员都拥有开发板）寻求技术支持。您也可以在我们的论坛（www.armfly.com）注册一个账号，我们会升级该账号为 VIP 权限，这样您就可以自由下载升级例程（终身有效）。申请加入 QQ 群和升级 BBS 用户权限时，必须提供开发板的 SN（及 CPU 的序号），否则会审核不过。1.1 章介绍了如何读取开发板 SN。

注意事项

- 外接电源必须是 5.0V 的直流电源，插头有极性为内正外负。
- 板子放置的桌面区域不要有金属物品，以免造成短路。
- 触摸屏属于易碎物品，请注意保护。
- 扬声器纸盆容易被硬物戳伤，请注意保护。
- 收到开发板之后，请首先检查有无硬件问题。

保修方法

- 开发板自出售之日起，提供一年免费保修服务。往返运费由客户承担。
- 超过 1 年后可提供免费维修服务，仅收取器件材料成本。往返运费全部由客户承担。
- 保修凭据是 CPU 序号。该 CPU 具有全球唯一的序号，我们出厂的每块板子均会登记此序号。因此保修手续简单，无需其它收据、发票等纸档凭证。
- 非保修范畴如下。超出保修范畴的仅收取器件更换成本，往返运费由客户承担。

a）由于用户使用不当（比如供电高压过高，短路等）造成板子损坏；

b）用户做试验或自行维修造成线路板焊盘脱落、铜线起皮的；

c）用户日常维护不当造成线路板腐蚀、基板出现裂纹的；

d）显示模块（含触摸屏）；

e）扬声器纸盆破裂；

目　　录

1.开发板使用方法

1.1. 检查硬件功能

我们设计了一个程序专门用于测试板子的硬件功能。出厂的板子均预先烧录了这个测试程序。如果你拿到的板子没有预先烧写这个测试程序，请按照“**烧写演示Demo程序**”步骤进行操作。

这个测试程序由几个目标镜像组成，存放光盘路径： \STM32F103ZE-EK光盘\05.预装的镜像

上电前的准备工作：

(1) 插上串口线（使用USB转串口的线也行），连接开发板的COM1口至PC机的串口。

(2) 插上网线（交叉网线和直连网线均可），连接开发板的网口至交换机端口或者电脑内置网卡。

(3) 打开串口工具或超级终端（波特率115200，1个起始位，1个停止位，8个数据位，无校验，无硬件流控）。

确保开发板周围没有可能造成板子短路的物品。插上USB电缆，打开开发板上的电源开关（S1）,电源指示灯LD5将点亮。

操作说明：

启动方式有6种：

操作说明：

（1）不按任何键启动，会进入STM32F103ZE-EK Demo程序。

（2）按住USER键启动，会进入DfuSe在线升级状态，此时可以使用PC软件进行程序升级操作。

（3）按住TAMPER键启动，会进入ucGUI Demo演示程序。

（4）按住WAKEUP键启动，会进入示波器演示程序。可以用示波器探头去测量DAC1输出的正弦波。

（5）按住摇杆OK键启动，会进入FM收音机演示程序。

（6）按住摇杆右键启动，会进入I2S录音和放音演示程序。

进入STM32F103ZE-EK Demo程序后，会出现提示界面要求你设置日期和时间，请不断地按摇杆OK键（中间）确认，之后开发板LCD上显示如下界面： （中间白色区域是320x240，两侧的紫色带是40像素宽）

PC机超级终端上显示如下内容（*----红色字体是笔者添加的注释*）：

```
************************************************
*  STM32F103ZE-EK Evaluate Kit
*  www.armfly.com Copyright 2009
*
*  Software Version : V1.0  2009-12-05
************************************************
CPU Reset
 0x40011400 = B8BB44BB
 0x40011404 = BBBBBBBB
 0x40011800 = BBBBB4BB
 0x40011804 = BBBBBBBB
 0x40011C00 = 44BBBBBB
 0x40011C04 = BBBB4444
 0x40012000 = 48BBBBBB
 0x40012004 = 444B4BB4
 0xA0000010 = 00001011
 0xA0000014 = 00000200
CPU : STM32F103ZET6, LQFP144, FFFFFF36 XXXXXXXX 43XXXXXX   ----CPU序号，每个板子均不一样
Testing the SRAM, Address = 0x68000000, Size = 1M Bytes    ----测试外部SRAM,1MB空间全扫描
The SRAM is OK
LCD Detected (SPFD5420A) 3.0" TFT LCD, WQVGA 400x240,      ----检测到LCD驱动芯片
DM9000AE Detect Ok, vid&pid = 0A469000                     ----检测到网卡芯片
CH374T Detect Ok                                           ----检测到USB Host驱动芯片
WM8978 Detect Ok                                           ----检测到CODEC芯片
24LC02 Detect Ok                                           ----检测到串行EEPROM芯片
FM TEA5767 Detect Ok                                       ----检测到FM收音机模块
Nor Flash ID = 0001,227E,2221,2201  Type = S29GL128P       ----检测到NOR Flash，显示ID
Nand Flash ID = AD,F1,80,1D  Type = HY27UF081G2A            ----检测到NAND Flash，显示ID
SPI Flash ID = BF2541                                      ----检测到串行Flash芯片
SPI Flash Type : SST25VF016B                               ----每批板子用的芯片型号可能不同
```

```
Testing CAN ......                                  ----检测CAN芯片
CAN is OK
Testing RS485 ......                                ----检测RS485芯片
RS485 is OK
*********************************************
Hardware Testing Ok                                 ----所有的硬件测试均通过
*********************************************
Display the STM32 introduction
```

测试项目列表：

序号	测试对象	说明
1	串口1	判据：观察串口1打印信息。 说明：很多测试信息将打印到串口1
2	串口2	判据：将PC机串口连接到开发板的COM2，然后按开发板复位按键。如果超级终端接收到“COM2 ....” 则表示COM2硬件正常。 说明：程序启动后会向COM2输出一串信息。
3	NAND Flash	判据：观察串口打印信息。 说明：读取芯片ID，如果正确则表示硬件无问题
4	NOR Flash	判据：观察串口打印信息。关注图标显示和音频播放。 说明：读取芯片ID，如果正确则表示硬件连接无问题。主界面显示的图片以及wav音频文件均存放在NOR Flash(约15M字节)
5	外部SRAM	判据：观察串口打印信息。 说明：对整个1M字节空间进行写读测试
6	串行Flash	判据：观察串口打印信息 说明：读取芯片ID，如果正确则表示硬件无问题
7	串行EEPROM	修改IP地址，IP地址将保存到EEPROM
8	RS485芯片	判据：观察串口打印信息 原理：同时使能RS485芯片的接收和发送，然后发送1个字节，如果接收到这个字节，表示硬件无问题。
9	CAN芯片	判据：观察串口打印信息 说明：通过自发自收检测芯片好坏。
10	SD卡测试	操作： 1. 在SD卡座插入一张SD卡

		2. 主菜单-USB Mass Storages – Start 3. 在windows文件浏览器中可以看到SD卡设备，并可以读取或修改SD卡内文件 说明： （1）目前仅支持小容量的SD卡，2G或2G以上的卡的支持还存在问题。 （2）NAND Flash的支持也存在问题。
11	网卡	操作： 1. 主菜单-Ethernet-Start WebServer 2. 在IE浏览器输入 http://192.168.1.10 3. 在windows CMD状态，执行ping 192.168.1.10 判据：IE浏览器显示web页面；观察ping命令执行结果。 说明：本程序移植了uIP协议栈
12	音频测试	操作： 1、主菜单-Wave Player 2、按摇杆上键或下键开始放音（上键是输出到扬声器，下键是输出到耳机） 3、开始放音后，按上键是增大音量，按下键是降低音量。 说明：播放一段WAV格式语音，检查耳机输出和扬声器输出
13	USB HOST测试	操作： 4、主菜单-USB Host – USB Disk 5、插入U盘，进行读写文件测试 6、通过串口打印的信息观察测试结果。 说明：这个例子支持FAT文件系统，插入U盘后，自动显示根目录下所有的文件名，然后将一个文本文件写到U盘根目录。
14	示波器测试	按住WAKEPU键启动进入示波器程序。 板子的DAC1接线端子输出频率10KHz，峰值2V的正弦波信号，可以用示波器探头去探测。
15	USB接口	按下USER 按键，然后复位开发板，windows发现一个USB硬件设备插入，这是DFU升级程序再运行。
16	JTAG接口	通过仿真器验证
17	触摸屏测试	按住TAMPER键启动进入ucGUI的演示程序，可以用触摸笔

		操作。

1.2. 跳线配置

为了便于初学者使用，本开发尽可能地减少了配置跳线，仅有2个拨码开关。

拨码开关S2选择CPU的启动方式

位1状态	位2状态	启动模式
OFF	无关	用户闪存存储器启动
ON	OFF	从系统存储器启动
ON	ON	从内嵌SRAM启动

拨码开关S3用于选择MIC插座连接的话筒类型 （仅在REV 2.0版本有效）

位1状态	位2状态	外接话筒类型
ON	ON	驻极体式话筒
OFF	OFF	动圈式话筒

拨码开关S10用于设置CAN接口

位1状态	控制120欧负载
OFF	断开CAN总线并接的120欧负载电阻
ON	连接120欧负载电阻

位2状态	控制CAN芯片的RS口线状态
OFF	CAN芯片工作于高速模式
ON	CAN芯片工作于待机模式

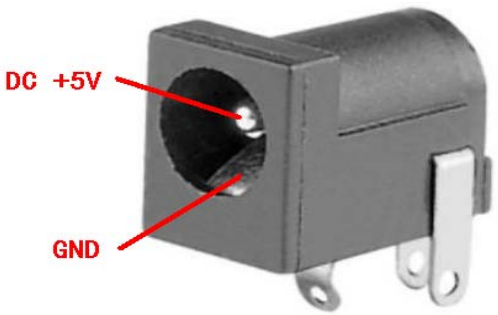
1.3. 电源供应

STM32F103ZE-EK开发板有三种供电方式，并且提供了电源开关S1进行切换。

1、通过外部5V直流电源供电（插座CN13）。

2、通过USB电缆供电（插座CN3）。

3、通过J-LINK调试器供电（插座CN16）。



对外接5V电源的要求：

（1） 主板的电源插座CN1为内正外负。

（2） 电源的电压不得高于5.5V。

J-LINK供电的方法：

（1） 将J-LINK的排线插入主板JTAG插座CN16，并将电源开关S1拨向板边。

（2） 启动windows开始菜单中的“J-Link Commander”，然后通过如下命令控制目标板的上电、下电和复位。

命令	功能
power on	目标板上电。J-LINK掉电后再上电时，将不会对目标板供电
power off	目标板下电
power on perm	目标板上电，并设置为缺省状态。以后每次连接上J-LINK时将自动给目标板上电
power off perm	目标板下电，并设置为缺省状态。
r0	使目标板处于复位状态
r1	使目标板退出复位状态，组合r0 r1 命令可实现目标板的复位

1.4. 启动模式

STM32F103ZE-EK开发板有以下三种启动方式：

- 从用户闪存存储器启动（正常运行时选择这种模式）
- 从系统存储器启动（做ISP下载时用）
- 从内嵌SRAM启动（调试用，一般很少使用）

通过设置子板上的拨码开关S2选择启动方式

位1状态	位2状态	启动模式
OFF	无关	用户闪存存储器启动
ON	OFF	从系统存储器启动
ON	ON	从内嵌SRAM启动

1.5. 纽扣电池

型号：CR1220
容量：40mAm
电压：3.0V
尺寸：Φ：12.5　H：　2.0（MM）

主板上的电池座可以安装CR1220型的纽扣电池。该电池提供给CPU内部RTC使用。当外部供电时，纽扣电池不对RTC电路供电，电流几乎为零。长期不使用时，请卸下电池。

注意：由于快递限制，本开发板标准配件不包括电池。

1.6. 恢复出厂程序

恢复出厂程序的步骤：

（1） 烧写 ” 512K整体烧写.hex” 文件到CPU内部Flash；
（2） 烧写 ” NOR_Flash.hex “ 文件到NOR Flash。

出厂程序存放位置：开发板光盘\05.出厂预装的程序

使用 J-Link 烧写 CPU 程序和 NOR Flash 数据文件的方法：

第1步：使用J-Link 烧写 ” 512K整体烧写.hex” 文件到CPU内部Flash。J-Link工程配置文件为：

STM32F103ZE-EK.jflash

第2步：使用J-Link 烧写 ” NOR_Flash.hex “ 文件到NOR Flash。J-Link工程配置文件为：

STM32F103ZE+S29GL128P.jflash

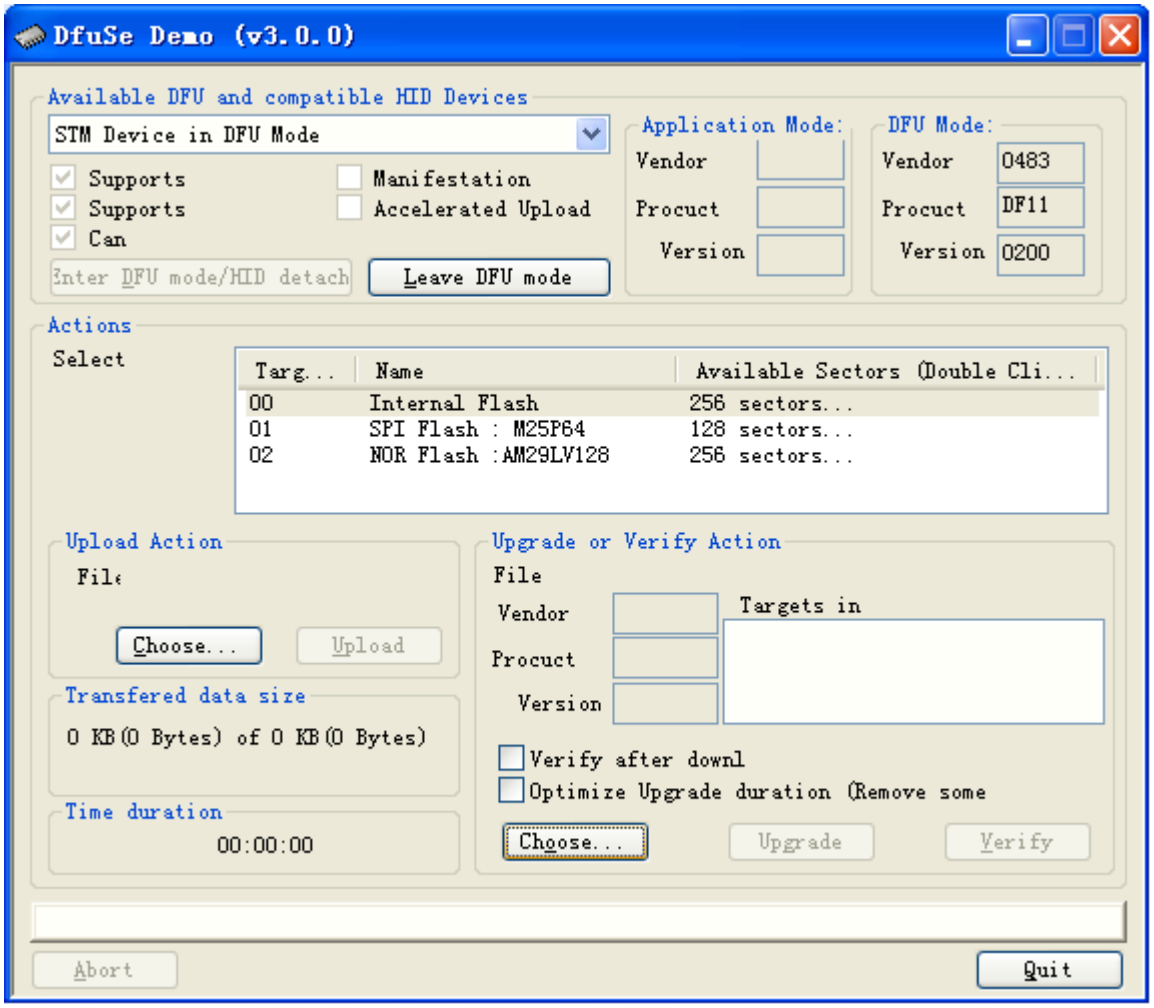
使用 DFU 软件，从 USB 接口烧写 NOR Flash 的方法：

烧写STM3210E-EVAL_Demo_NORFlash_V1.1.1.dfu到NOR Flash。起始地址：0x64006000

（1） 安装um0412中的 DfuSe_Demo_V3.0_Setup.exe

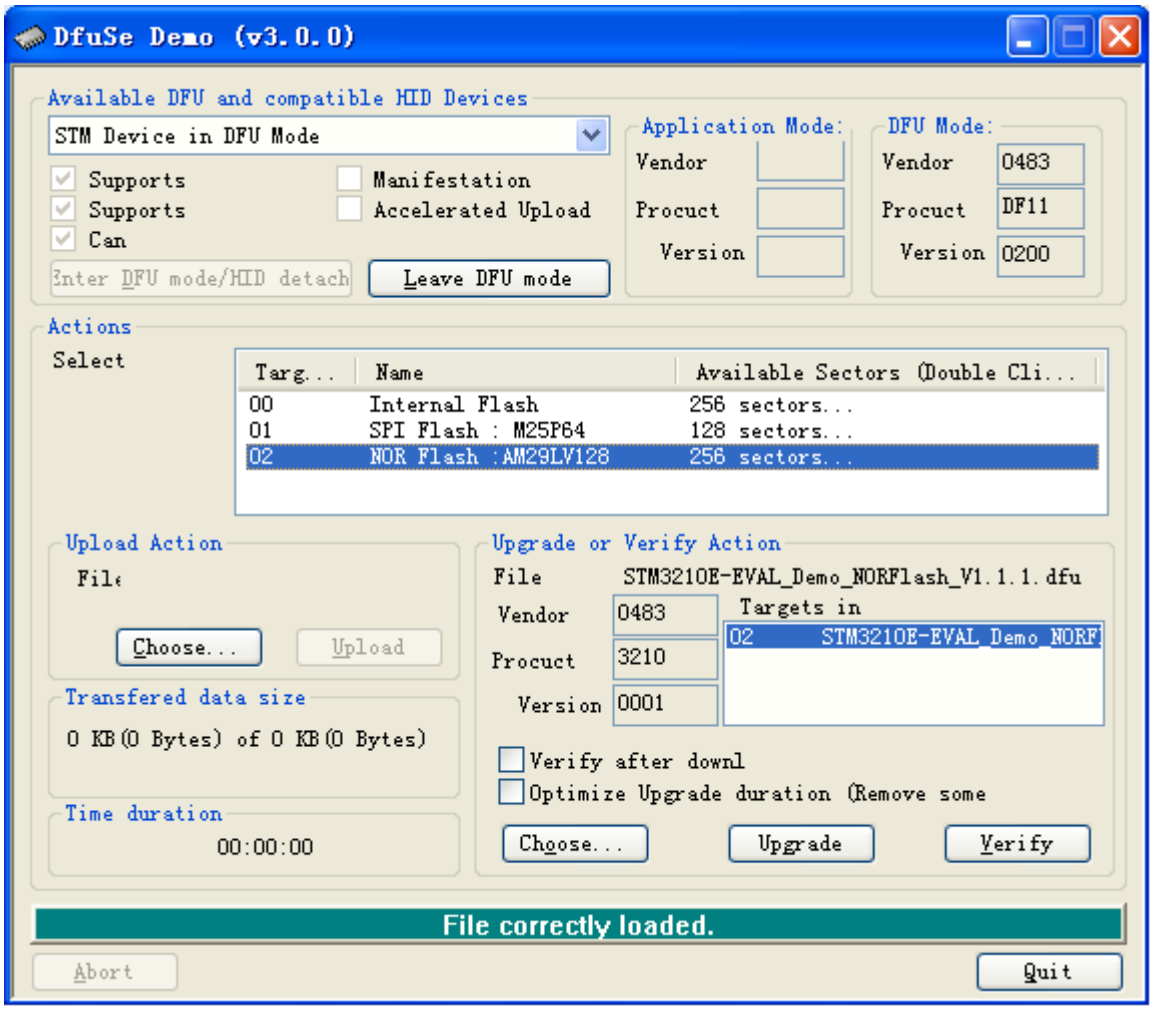
（2） 执行windows开始菜单 “STMicroelectronics” - “DfuSe” - “DfuSe Demonstration”

（3） 开发板上电，按住开发板上的” USER” 键不放，再按复位键，此时Windows应该发现USB设备插入，USB驱动安装成功后，即可在DfuSe Demonstration界面看到开发板上的资源。



（4） 点击按钮“Choose”，选择“STM3210E-EVAL_Demo_NORFlash_V1.1.1.dfu”文件

（5） 将“Select”区域的光标移动到 第3行，即选中NOR Flash。



（6） 点击按钮“Upgrade”，将开始下载文件到NOR Flash。此过程耗时很长，请耐心等候。

如果不加校验，总耗时约13分钟，加上校验则耗时23分钟。

注：我做过测试，单独写个程序完成NOR Flash整个16M空间擦写和重写只需要3分钟。看来大部分时间耗费在USB通信上了。这个Dfu Boot程序和PC机程序的通信机制还有待改进。我分析慢的原因：

a）USB通信时每包数据长度过小，只有512字节，造成通信量很大，需要32768包数据。

b）一问一答式的通信协议也是效率低下的一个原因。

1.7. 烧写程序

1.7.1. 使用J-LINK仿真器烧写CPU内部Flash

（1） 启动J-Link ARM软件。

（2） 点击主菜单 “Options” – “Project Settings”

（3） 选择“CPU”页签，然后在”Device“下拉列表框中选择”STM32F103ZE“

（4） 点击“确定”后完成设置。你也可以再点击 “File ” - “Save Project As”保存 *.prj文件，以便于下次进行重复设置。

（5） 然后点击主菜单“File “ – ” Open“装入待烧写的文件。该软件会根据后缀自动识别文件格式。（支持hex，bin，s19等）。

（6） 点击主菜单“Target “-” Connect “，测试一下目标板和J-Link之间的连接是否正常。

（7） 点击主菜单“Target ” -“Auto” ，将开始烧写Flash，之后会进行校验。也可以直接按“F7”键开始自动烧写。

1.7.2. 使用J-LINK仿真器烧写Nor Flash

（1） 启动J-Link ARM软件。

（2） 点击主菜单 “File” – “Open Project…” 。

（3） 找到“STM32F103ZE+S29GL128P.jflash”文件。
路径： 05.出厂预装的程序\J-Link配置文件\STM32F103ZE+S29GL128P.jflash

（4） 然后点击主菜单“File “ – ” Open“装入待烧写的文件。该软件会根据后缀自动识别文件格式。（支持hex，bin，s19等）。

（5） 点击主菜单“Target “-” Connect “，测试一下目标板和J-Link之间的连接是否正常。

（6） 点击主菜单“Target ” -“Auto” ，将开始烧写Flash，之后会进行校验。也可以直接按“F7”键开始自动烧写。

1.7.3. 使用串口ISP烧写CPU内部Flash

安装软件

安装文件路径：光盘\04.工具软件\ISP下载\ST原版\um0462安装

双击文件夹下的 Flash_Loader_Demonstrator_V2.0_Setup.exe 文件，根据安装程序界面的提示完成ISP软件安装。

硬件连接

使用串口线连接开发板的COM1口和计算机的串口（9针串口），如果计算机没有硬件串口，可以使用USB转串口线。

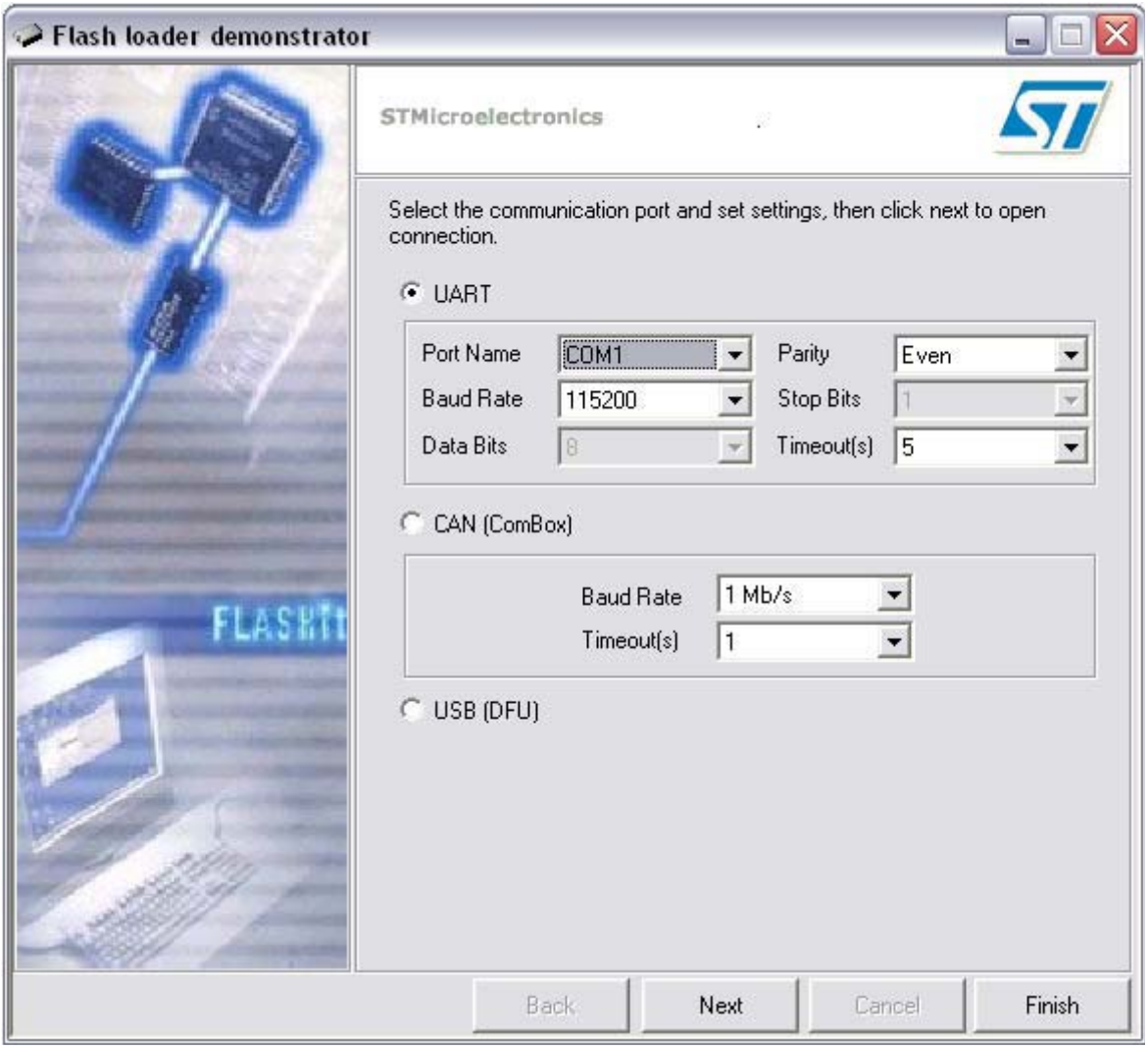
连接好电缆后，请给开发板上电

- 如果是外部5V电源，请拨动电源开关，给开发板上电。
- 如果是USB供电，请拨动电源开关（另外一个方向），给开发板上电。
- 开发板上电后，开发板上的红色电源指示灯（在电源插座附近）将点亮。

操作步骤 1：

运行Flash Load Demo程序。 程序位置：

开始菜单\所有程序\STMicroelectronics\Flash Load Demonstrator\Flash Load Demo



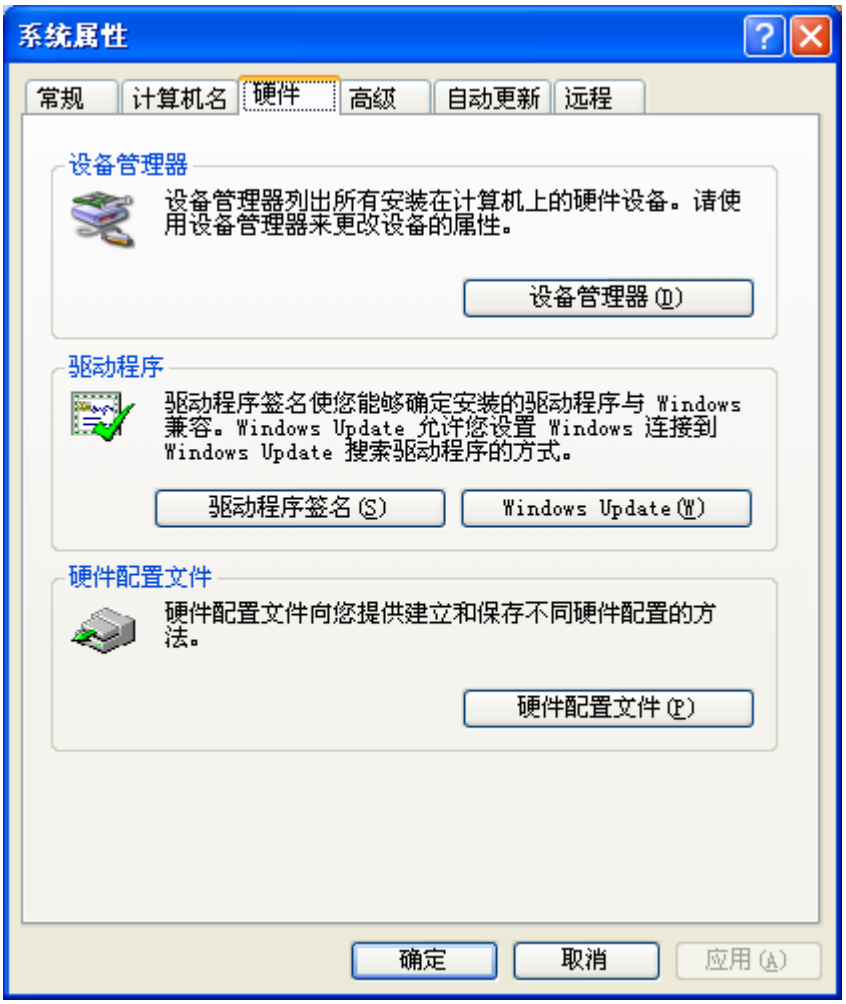
- Port Name 根据实际情况选择，如果计算机有硬件串口，那么一般是COM1；如果使用USB转

串口线，可能是COM2或以上。

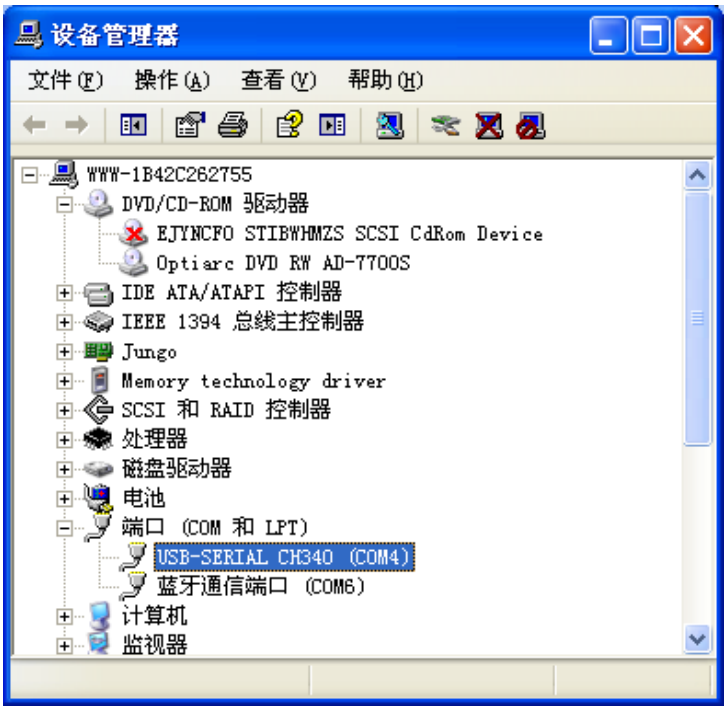
- Baud Rate 选择115200。
- 设置完毕后，请点击“Next”按钮。

如果不知道串口号，请按如下方法进行查询：

- 鼠标移到Windows桌面上的“我的电脑”图标，鼠标右键呼出“系统属性”窗口



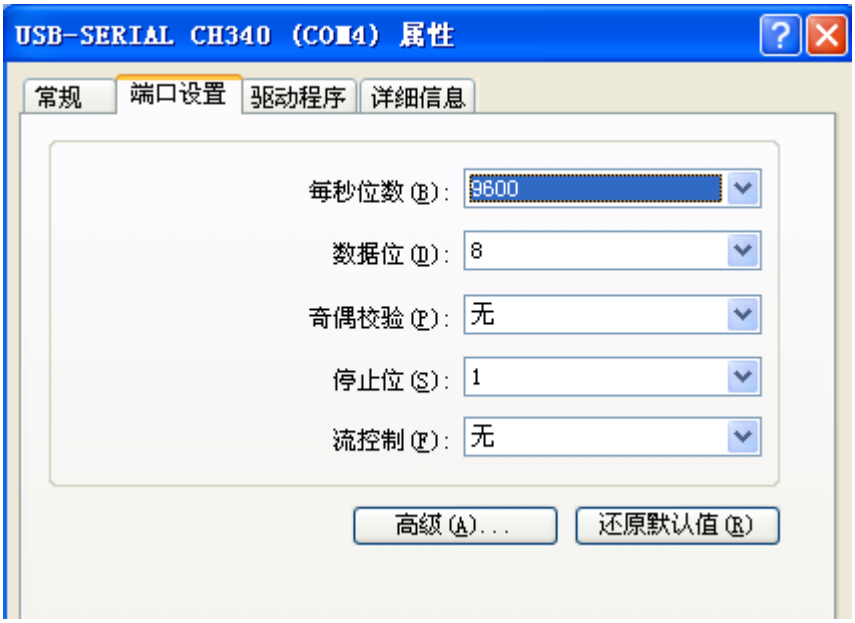
- 选择“硬件”选项卡，点击“设备管理”打开“设备管理器”窗口



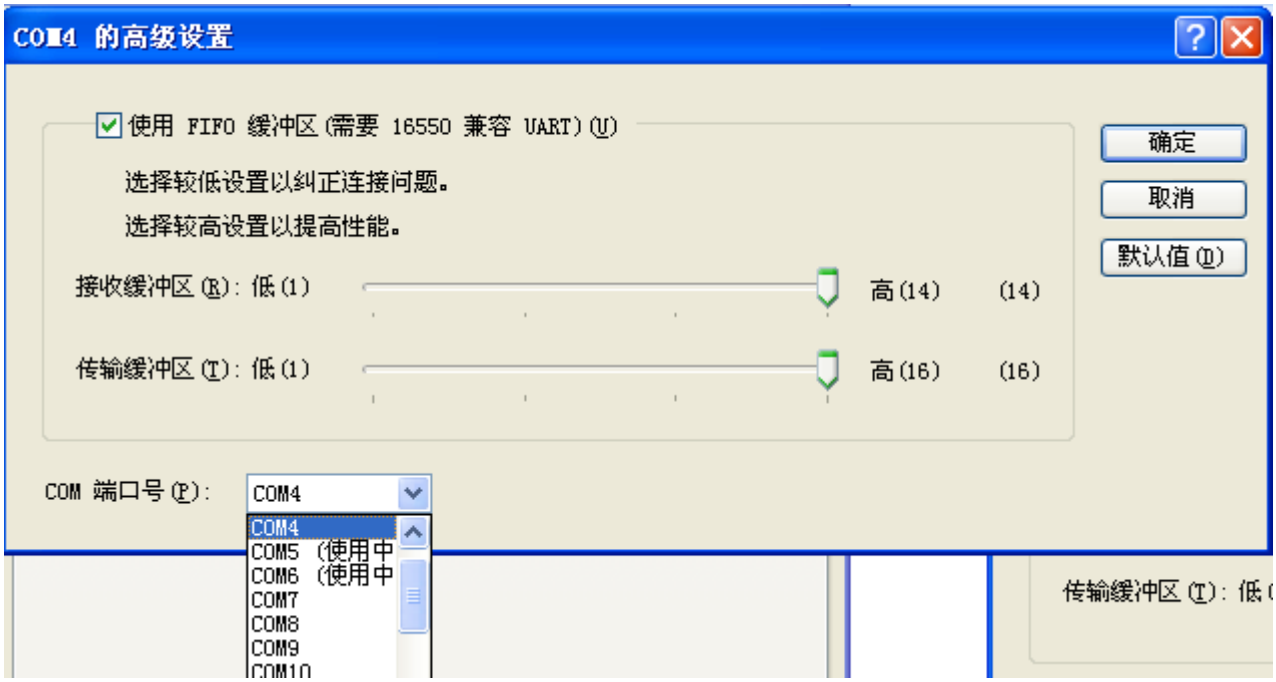
- 找到“端口（COM和LPT）”条目，点击“+”展开。按上图，USB转串口线的COM口为COM4。

如果想修改这个端口号，可以继续按如下方法操作：

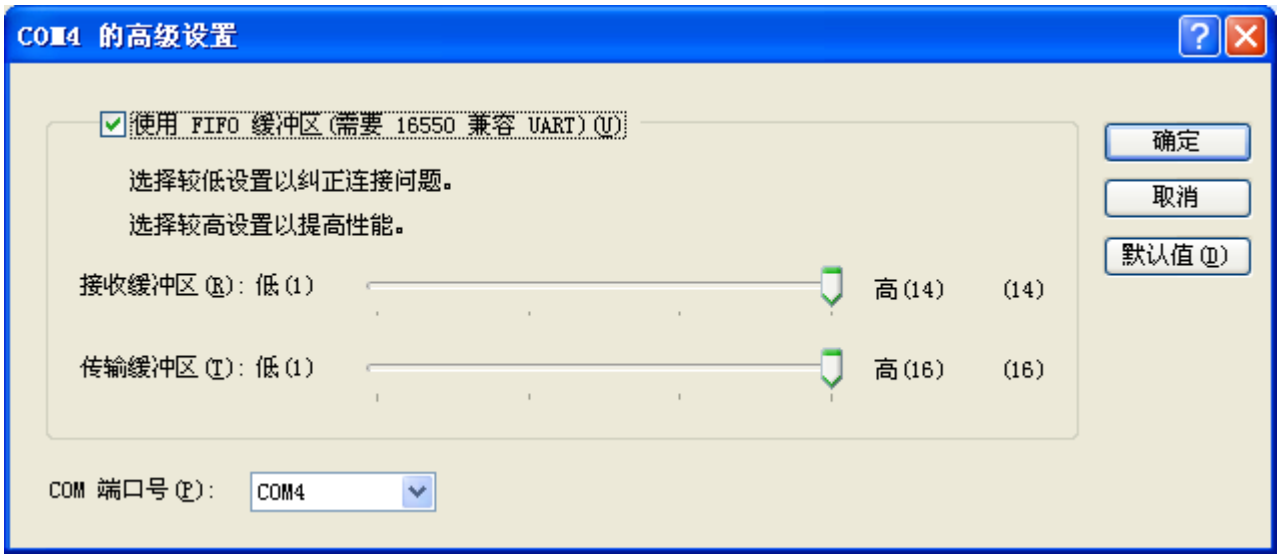
- 鼠标移动到“USB-SERIAL CH340 (COM4)”,然后点击鼠标右键，打开属性窗口



- 点击“高级”按钮，打开“COM4 高级设置“窗口



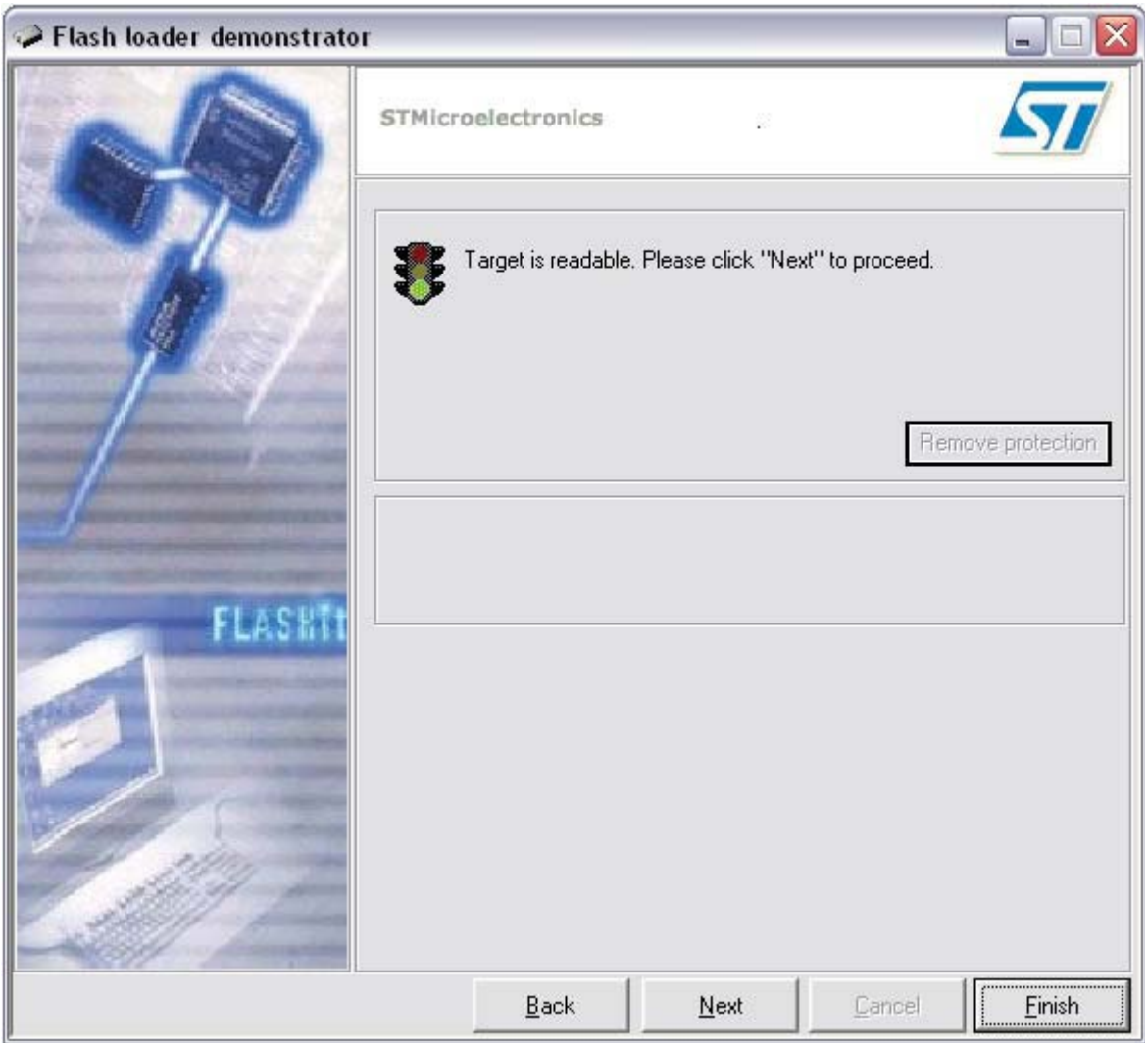
- 点击左下角“COM端口号“的下拉列表框，然后可以选择需要的端口号
- 选择完毕后，按“确认“按钮保存即可。



操作步骤 2：

到在这一步，已经连接成功。窗口会显示Flash存储器的状态，比如“可读“或”读保护“。如果是”读保护“，可以通过点击”Remove protection“按钮解除Flash的读保护。

注意：解除读保护的同时，会擦除整个Flahsh。

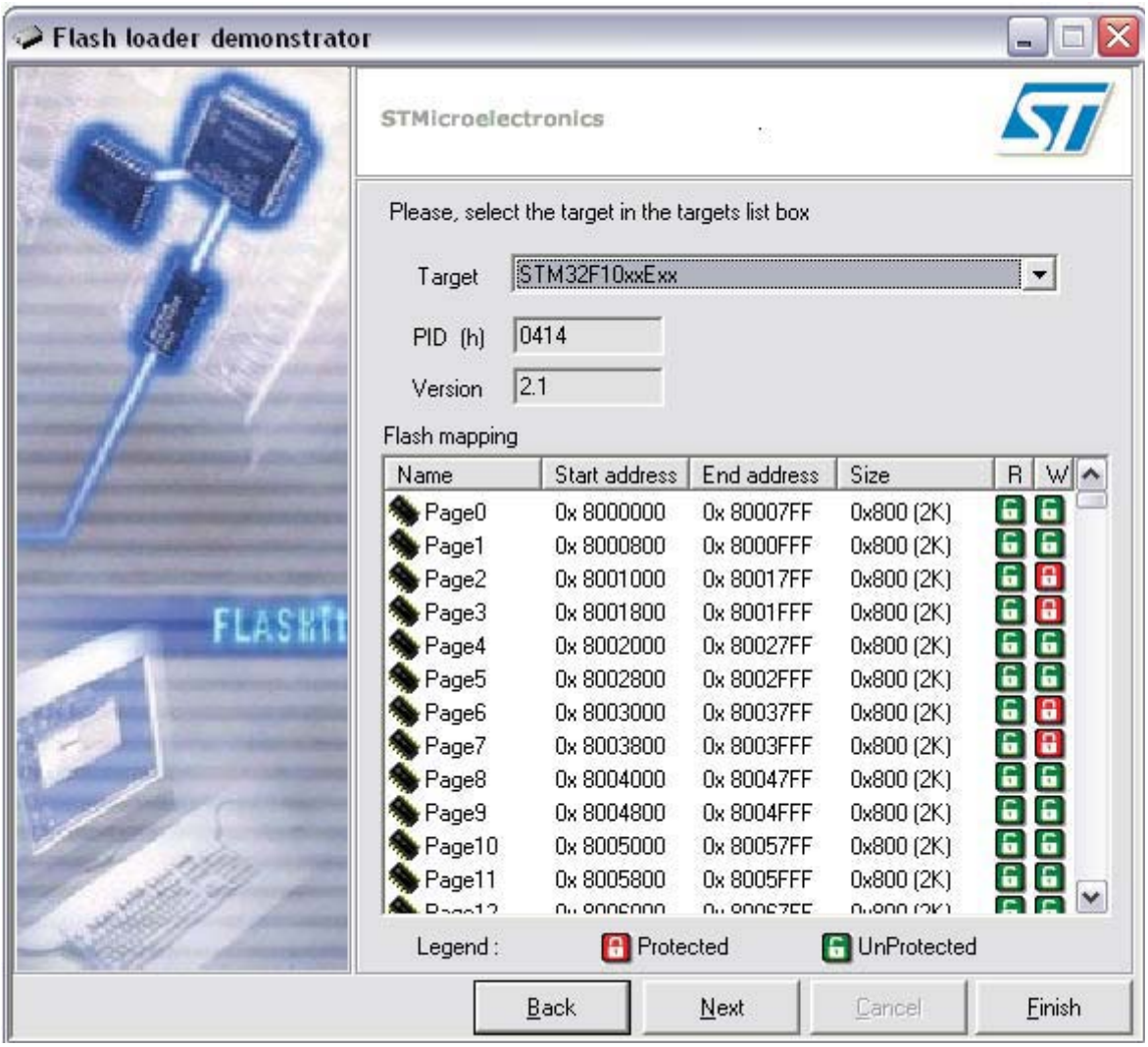


点击“Next“按钮，进入下个步骤。

操作步骤 3：

在这一步，界面将显示支持的芯片，如Target ID、固件版本、Flash存储器地址映射、Flash存储器保护状态。

选择正确的芯片型号，然后点击“Next“继续下一步。



操作步骤 4：

在这一步，用户可以进行擦除、下载、上传、使能或禁止Flash保护、编辑选项字。

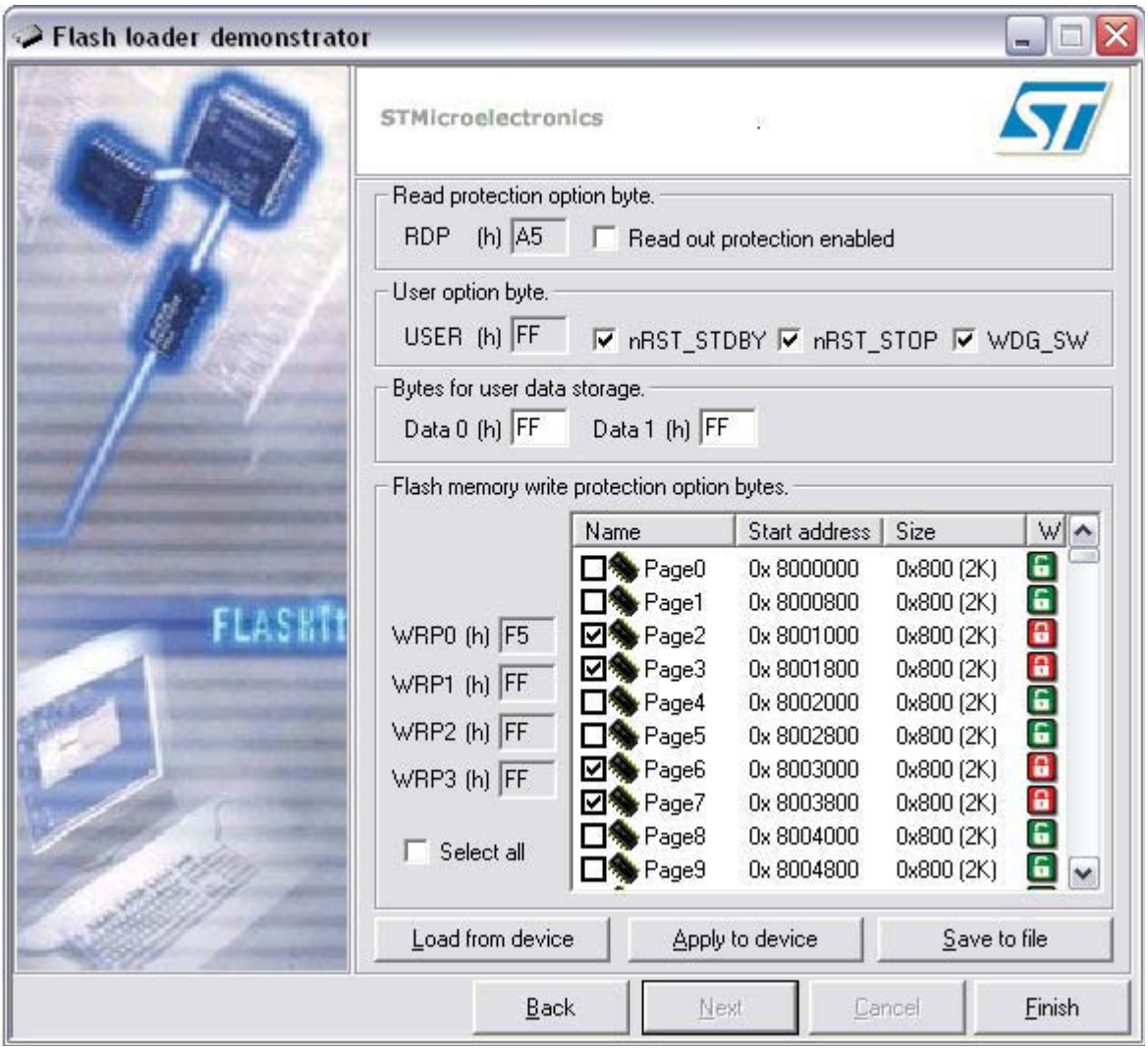
- **Erase（擦除）**
  - 选择“All“擦除整个存储器
  - 选择“Selection“定制擦除选项。单击”...“按钮将显示存储器映射对话框。然后检查被擦除的页面，点击”Ok“
- **Download（下载）**
  - 选择浏览按钮打开bin、hex或S19文件。如果是bin文件，需要修改“@（H）“编辑框的内容指定起始地址。如果是hex或S19文件，则这个编辑框是只读的，因为地址信息已经包含在hex或s19文件内部了
  - 选中“Verfiy “，当下载完成后，进行自动校验
  - 选中“Jump to the user program “，下载完毕后启动用户程序
  - 选中“Optimize “，编程时自动过滤0xFF包（以256字节为单位）

- 选中“Apply option bytes”，并且点击”...“按钮指派选项字文件，下载完毕后会自动将选项字写入芯片

- **Upload（上传）**
  - 点击“...“按钮，指定保存上传数据的文件格式和文件名。
- **Disable/Enable Flash protection （禁止/使能Flash保护）**
  - 点击2个下拉列表框选择操作命令（使能读保护、禁止读保护、使能写保护、禁止写保护）。
  - 除了“Enable Write Protect”（使能写保护）外，其它保护命令都是针对Flash所有的页。
  - 对于“Enable Write Protect”命令，点击“...“按钮可以选择需要写保护的页。
- **Edit option bytes**
  - 如果选中“Edit option bytes”，那么点击“Next”后，将进入选项编辑界面

操作步骤 5：

这步操作仅针对STM32，STM8没有这步操作。软件界面依赖于第4步的设置。



- **第4步为“Edit option bytes”操作**

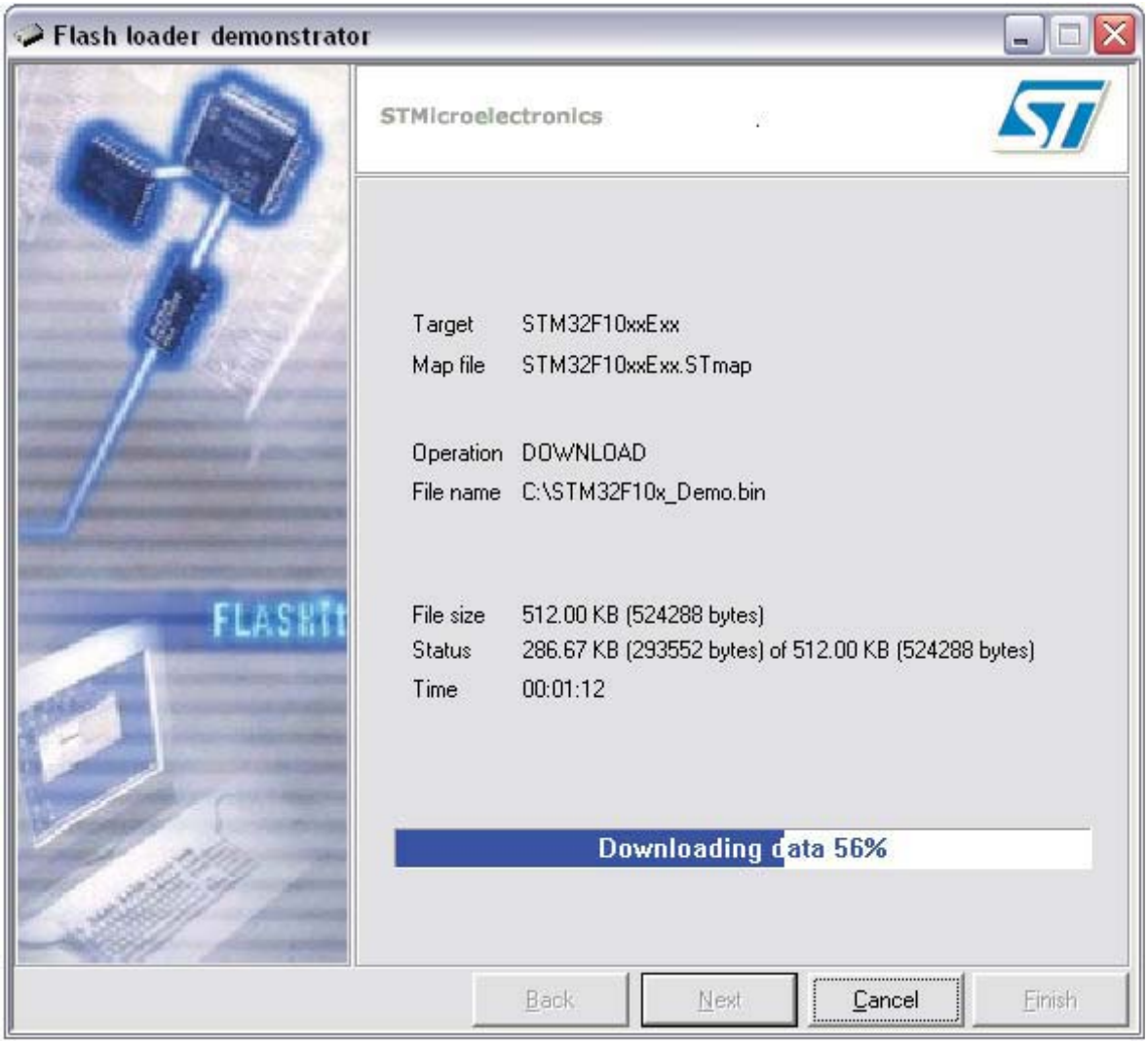
将出现选项编辑窗口。它包括从当前芯片读出的选项值：RDP、USER、Data0、Data1、WRP0、WRP1、WRP2和WRP3。更详细的信息，请参考 “ “STM32F10xxx Flash programming manual”（PM0042 www.st.com）

在这一步，可将选项值从芯片读出并保存为文件。

- **第4步为其它操作**

将出现如下界面。包括下载或上传的数据大小、完成的百分比和操作时间。

  - 如果操作成功，进度条显示绿色，如果有错误发生，进度条将变为红色，并显示错误内容
  - 点击“Cancel”按钮可以中止操作
  - 如果第4步选中了“Jump to the user program” ，操作执行完毕后，CPU将跳转到Flash区执行用户程序，Bootloader程序将丢失，在这种状态下，点击“Back”按钮，将返回到第1步。
    如果第4步没有选中“Jump to the user program” ，点击“Back”按钮，将返回到第4步。



1.8. 学习建议

学习STM32开发，必须安装集成开发环境软件。集成开发环境（IDE）是一套复杂的软件，提供了编辑器、编译器、连接器等工具，并提供编译设置、连接设置、调试器接口等界面以方便软件开发。对于STM32，国内常用的IDE有Keil公司的RealView MDK和IAR公司的EWARM。

开发板配套的光盘上已经收录两种软件的安装程序，请根据自己的需要进行选择安装。

特别说明：大家从网上下载的IDE版本可能是功能受限或者存在缺陷的版本。开发板光盘收录的版本都是经过我们验证无问题的。光盘上的例程也是经过编译、运行测试通过的。如果客户使用其它版本的IDE出现编译不过或者程序执行异常的现象，请换用我们提供的IDE版本。

关于开发软件的安装和选用，请见“软件开发环境”章节的详细介绍。

本产品的售后和技术支持方法：

（1） 到BBS上提问。http://www.armfly.com/bbs/index.php

（2） QQ邮件交流。armfly@qq.com

（3） 加入QQ群，群内每位成员都拥有安富莱STM32开发板，提供QQ群空间便于大家相互交流。

本开发板的官方资料下载地址：

http://www.armfly.com/bbs/forumdisplay.php?fid=2

ST（意法半导体）官方资料下载地址：

http://www.st.com/mcu/familiesdocs-110.html

如果对C语言不熟悉，可以参考光盘上收录的一本c语言书（受到广大网友推荐），路径在：

00. 初学者教程\C Primer Plus（第五版）中文版.pdf

如果没有做过ARM开发，不熟悉ARM开发环境，建议看看光盘上收录的视频教程（中文语音），这个教程是ST官方提供的，以STM3210E-LK学习板为例子进行讲解。安富莱STM32F103ZE-EK开发板和STM3210E-LK学习板用的CPU一样，4个LED指示灯的控制口线也一样。第3课例程中用到了LCD，这个和安富莱的板子不一样，仅供参考。

路径：光盘\00. 初学者教程\

STM32 New Comer Lesson One. pps -- STM32入门培训系列课程第一课

STM32 New Comer Lesson Two. -- STM32入门培训系列课程第二课

STM32 New Comer Lesson Three.pps -- STM32入门培训系列课程第三课

My First STM32 Project_for_Lesson2.zip -- 第二课中建立的用户跑马灯程序。

My First STM32 Project-improved_for_Lesson2.zip -- 第二课中修改延时程序后的跑马灯程序。

STM32_Calendar_for_Lesson3.zip -- 第三课中日历时程序。

对于有一定C语言基础和ARM开发基础的人员，首要目的是快速熟悉STM32 CPU硬件和软件库的用法，推荐的学习步骤：

（1） 快速浏览CPU参考手册中的Memory and bus architecture，PWR，RCC，GPIO、USART、interupts and events等章节。虽然看CPU的DataSheet很枯燥，但是这是最权威的信息，挑重点章节多看几遍比去网上搜索答案更有效，可以好不夸张地说“阅读DataSheet也是一种技能“。

ST的CPU手册分两个文档，一个叫数据手册（1.6M，描述不同封装CPU间的差异以及管脚定义），另外一个叫参考手册（8.5M，详细描述CPU内部结构、外设功能以及各个寄存器的用法）。

光盘上同时收录了中文版本和英文版本的手册，请主要以英文版为主，中文译本是从早期英文版翻译过来的。路径：\02.硬件资料\02.STM32 CPU数据手册

（2） 将光盘上提供的几个重要的基础例程跑一遍，主要是熟悉ST软件库中常用函数的用法。顺序可以按 GPIO，USART，RTC，PWR，NVIC开始，然后熟悉ADC，CAN，SPI，USB等。

（3） 阅读并跟踪调试Demo程序，Demo程序几乎使用了板子的全部资源。包括LCD、网卡、USB Host、DfuSe固件升级。

（4） 阅读并跟踪调试uCOS + ucGUI的Demo程序。（可选）

（5） 在学习的过程中，要逐步掌握开发工具的用法。主要是KEIL或IAR集成环境、J-Link仿真器。有一定基础后，可以自己设计或者在别人的基础上设计一个有使用价值的完整的程序，然后共享你的成果，和大家一起分享成功的喜悦。

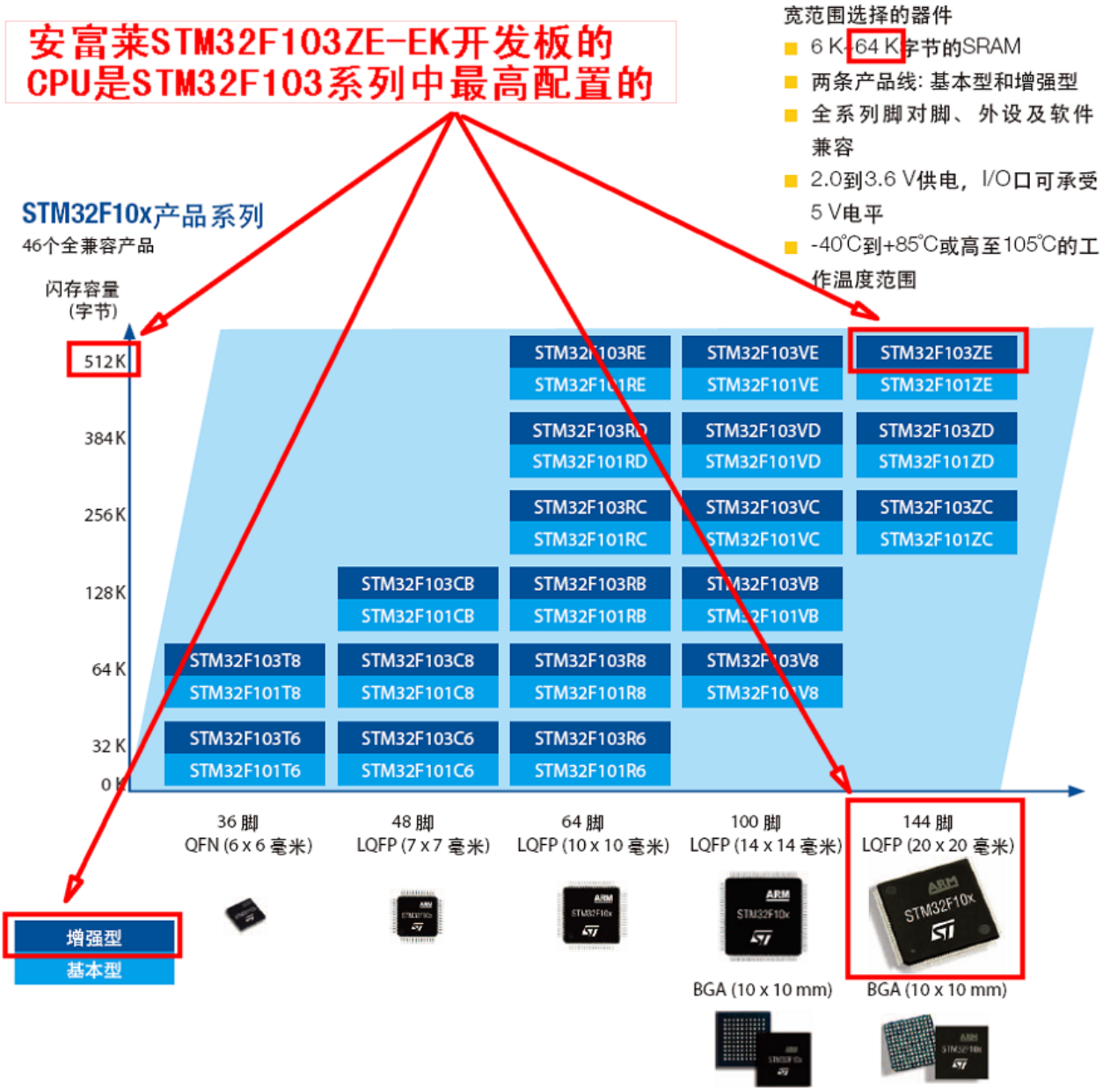
如果你对Cortex核心感兴趣，可以阅读

\01.文档\06.认识Cortex-M3核心\STM32权威指南（中文）.pdf

2.开发板硬件

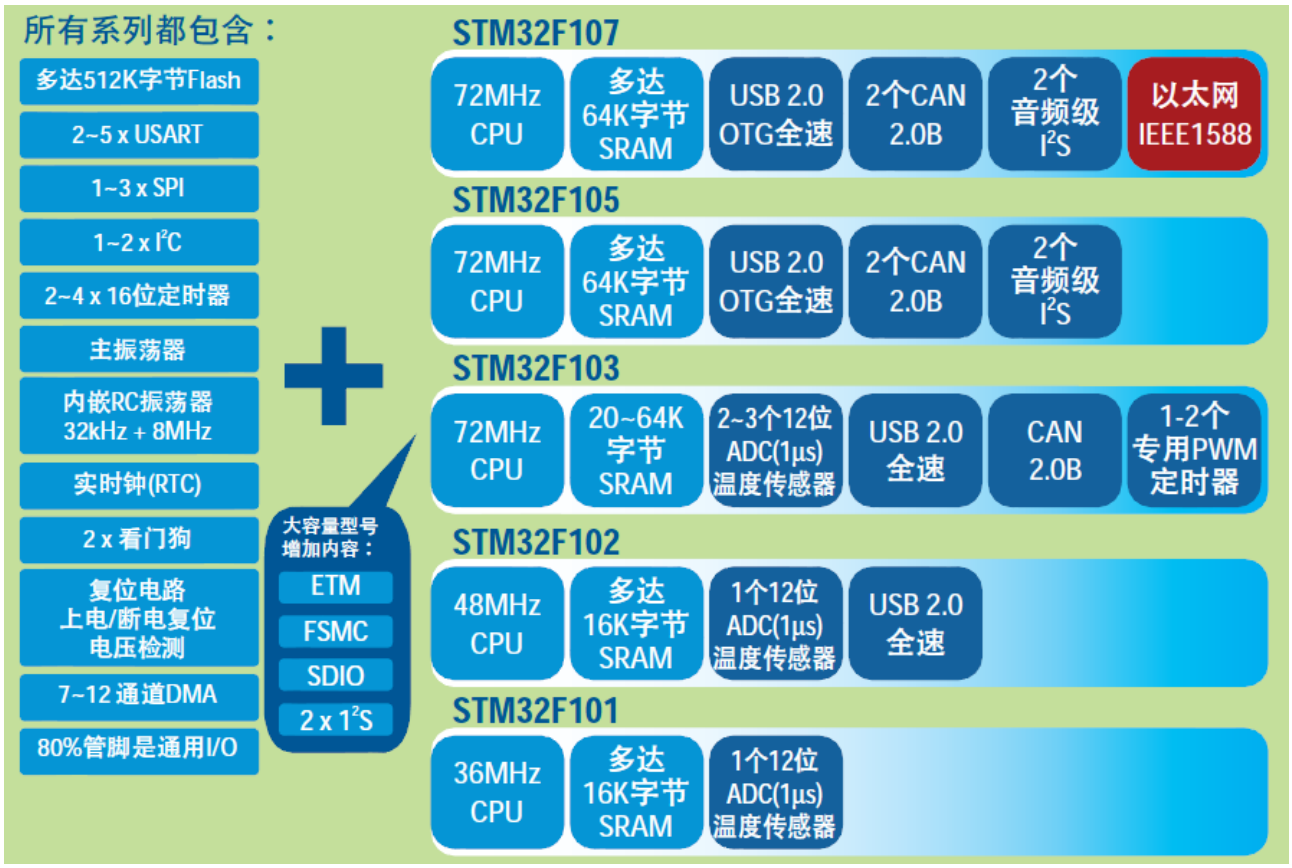
2.1. CPU介绍

安富莱STM32F103ZE-EK开发板的CPU为ST（意法半导体）公司的**STM32F103ZET6**。它是STM31F103XX产品线中功能最强大的一款CPU。



STM32F103ZET6 片内集成512kB Flash、64kB RAM、1个USB、1个CAN、8个定时器、5个USART、

3个ADC、2个DAC、3个SPI、2个I2C、2个I2S、1个SDIO、112个GPIO、FSMC总线（支持NOR、NAND、SRAM）。CPU主频为72MHz。



STM32的主要优点

- 使用ARM最新的、先进架构的Cortex-M3内核
- 优异的实时性能
- 杰出的功耗控制
- 出众及创新的外设
- 最大程度的集成整合
- 易于开发，可使产品
- 快速进入市场

2.2. 硬件规格

简介

STM32F103ZE-EK开发板以STM32F103ZET6（LQFP144）为核心。

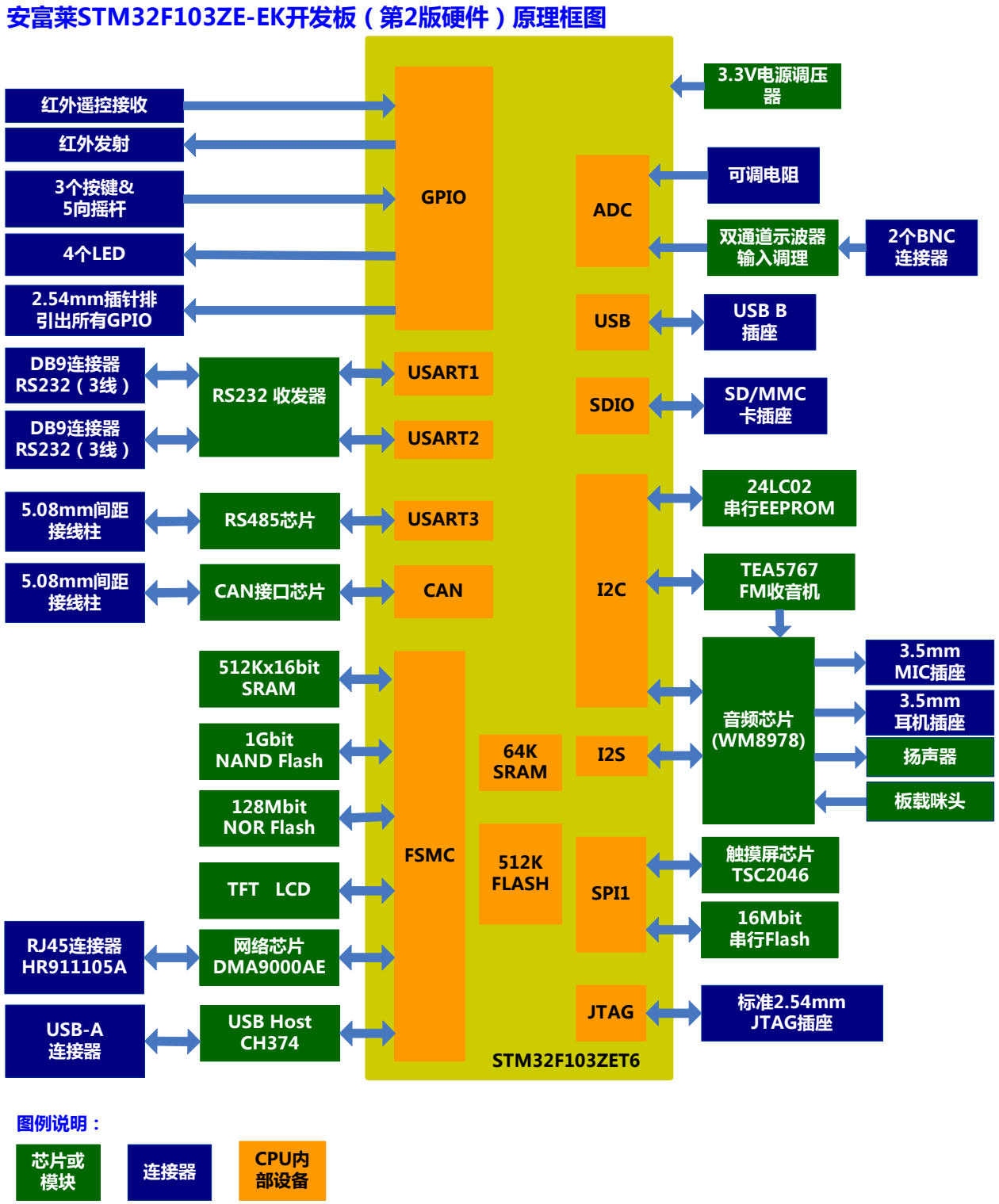
STM32F103ZE 是ST（意法半导体）公司推出的ARM Crotex-M3产品线中功能最强大的一款CPU。片内集成512kB Flash、64kB RAM、1个USB、1个CAN、 8个定时器、5个USART、3个ADC、2个DAC、3个SPI、2个I2C、2个I2S、1个SDIO、112个GPIO、FSMC总线（支持NOR,NAND,SRAM）。CPU主频72MHz,广泛适用于各种应用场合。

本开发板具备丰富的硬件资源，配套的试验例程均提供源代码，文档齐备，非常适合于学习和项目评估。

硬件资源

- 8M晶振作为MCU的时钟，32768晶振用于RTC
- 1M字节SRAM，16M字节NOR Flash，128M字节NADN Flash
- 2M字节串行Flash，256字节串行EEPROM
- 1个SD/MMC卡座
- 1个CAN2.0A/B接口
- 2个RS232串口
- 1个RS485接口
- 1个USB2.0全速DEVICE接口
- 1个USB2.0全速HOST接口
- 1个100M/10M以太网接口
- I2S音频CODEC（24bit，48kHz），1个立体声耳机插座，1个MIC插座，1个咪头，1个扬声器
- 3.0寸TFT真彩触摸LCD（WQVGA，400x240）
- 集成FM调频收音机模块
- 1个红外遥控接收模块，1个红外遥控发射器
- 1个5向摇杆，1个Reset按钮、1个wakeup按钮、1个自定义按钮
- 4个自定义LED，1个电源LED，1个音频LED
- 1个CR1220电池座
- 1个精密可调电阻连接到ADC输入
- 所有的GPIO引到2.54mm间距焊盘
- 1个DAC引出端子，1个PWM引出端子
- 标准2.54mm间距JTAG插座
- 2个BNC输入端子，集成双通道示波器电路，具备AC/DC切换、输入增益切换开关
- 3种供电方式：USB电缆、外接5V电源、JTAG调试接口（J-LINK仿真器）
- 1个电源开关，上下电时无需拔插电缆
- 3种启动方式：用户Flash、系统存储器、SRAM
- 用拨码开关取代跳线帽，避免跳线帽丢失
- 板子规格：14cm x 12cm

2.3. 硬件模块框图


安富莱STM32F103ZE-EK开发板（第2版硬件）原理框图
红外遥控接收
红外发射
3个按键&
5向摇杆
4个LED
2.54mm插针排
引出所有GPIO
GPIO
3.3V电源调压
器
可调电阻
ADC
双通道示波器
输入调理
2个BNC
连接器
USB
USB B
插座
DB9连接器
RS232（3线）
DB9连接器
RS232（3线）
RS232 收发器
USART1
USART2
SDIO
SD/MMC
卡插座
5.08mm间距
接线柱
5.08mm间距
接线柱
RS485芯片
CAN接口芯片
USART3
CAN
I2C
24LC02
串行EEPROM
TEA5767
FM收音机
512Kx16bit
SRAM
1Gbit
NAND Flash
128Mbit
NOR Flash
TFT LCD
64K
SRAM
I2S
音频芯片
(WM8978)
3.5mm
MIC插座
3.5mm
耳机插座
扬声器
板载咪头
FSMC
512K
FLASH
SPI1
触摸屏芯片
TSC2046
16Mbit
串行Flash
RJ45连接器
HR911105A
网络芯片
DMA9000AE
USB-A
连接器
USB Host
CH374
JTAG
标准2.54mm
JTAG插座
STM32F103ZET6
图例说明：
芯片或
模块
连接器
CPU内
部设备

2.4. GPIO资源分配

I = 输入，O = 输出，X = 空闲

管脚	方向	用途		管脚	方向	用途
PA0	I	WKUP		PB0	I	红外遥控信号输入
PA1	I	DM9000AE中断		PB1	O	LCD背光
PA2	O	串口2，USART2_TX		PB2	O	BOOT1&串行Flash片选
PA3	I	串口2，USART2_RX		PB3	O	JTAG, TDO
PA4	O	DAC_OUT1		PB4	I	JTAG, TRST
PA5	O	串行Flash，SPI1_SCK		PB5	I	触摸屏，TP_BUSY
PA6	I	串行Flash，SPI1_MISO		PB6	I	串行EEPROM，I2C1_SCL
PA7	O	串行Flash，SPI1_MOSI		PB7	I/O	串行EEPROM，I2C1_SDA
PA8	O	红外遥控信号发射		PB8	I	CANRX
PA9	O	串口1，USART1_TX		PB9	O	CANTX
PA10	I	串口1，USART1_RX		PB10	O	RS485，USART3_TX
PA11	-	USBDM (D-)		PB11	I	RS485，USART3_TX
PA12	-	USBDP (D+)		PB12	O	WM8978，I2S2_WS
PA13	I	JTAG, TMS		PB13	O	WM8978，I2S2_CK
PA14	I	JTAG, TCK		PB14	O	USB上拉使能
PA15	I	JTAG, TDI		PB15	O	WM8978，I2S2_SD
PC0	I	WKUP		PD0	I/O	FSMC_D2
PC1	I	DSO，ADC123_IN10		PD1	I/O	FSMC_D3
PC2	I	DSO，ADC123_IN11		PD2	O	SD卡，SDIO_CMD
PC3	I	DSO，ADC123_IN12		PD3	I	摇杆，下
PC4	I	DSO，ADC123_IN13		PD4	O	FSMC_NOE
PC5	I	触摸屏中断		PD5	O	FSMC_NWE
PC6	O	WM8978，I2S2_MCK		PD6	I	FSMC_NWAIT
PC7	O	SD卡插入检测		PD7	O	FSMC_NCE2
PC8	I/O	SD卡，SDIO_D0		PD8	I/O	FSMC_D13
PC9	I/O	SD卡，SDIO_D1		PD9	I/O	FSMC_D14
PC10	I/O	SD卡，SDIO_D2		PD10	I/O	FSMC_D15

PC11	I/O	SD卡，SDIO_D3		PD11	O	FSMC_A16
PC12	O	SD卡，SDIO_CK		PD12	O	FSMC_A17
PC13	I	按键，TAMPER		PD13	O	FSMC_A18
PC14	I	32K晶振，OS32_IN		PD14	I/O	FSMC_D0
PC15	O	32K晶振，OS32_OUT		PD15	I/O	FSMC_D1
PE0	O	FSMC_NBL0		PF0	O	FSMC_A0
PE1	I	FSMC_NBL1		PF1	O	FSMC_A1
PE2	I	CH374中断信号		PF2	O	FSMC_A2
PE3	O	FSMC_A19		PF3	O	FSMC_A3
PE4	O	FSMC_A20		PF4	O	FSMC_A4
PE5	O	FSMC_A21		PF5	O	FSMC_A5
PE6	O	FSMC_A22		PF6	O	LED1
PE7	I/O	FSMC_D4		PF7	O	LED2
PE8	I/O	FSMC_D5		PF8	O	LED3
PE9	I/O	FSMC_D6		PF9	O	LED4
PE10	I/O	FSMC_D7		PF10	O	RS485，发送使能（1有效）
PE11	I/O	FSMC_D8		PF11	O	RS485，接收使能（0有效）
PE12	I/O	FSMC_D9		PF12	O	FSMC_A6
PE13	I/O	FSMC_D10		PF13	O	FSMC_A7
PE14	I/O	FSMC_D11		PF14	O	FSMC_A8
PE15	I/O	FSMC_D12		PF15	O	FSMC_A9
PG0	O	FSMC_A10				
PG1	O	FSMC_A11				
PG2	O	FSMC_A12				
PG3	O	FSMC_A13				
PG4	O	FSMC_A14				
PG5	O	FSMC_A15				
PG6	O	FSMC_INT2				
PG7	I	摇杆，中间键				
PG8	I	按键， USER				
PG9	O	FSMC_NE2				
PG10	O	FSMC_NE3				

PG11	O	触摸屏SPI片选				
PG12	I/O	FSMC_NE4				
PG13	I/O	摇杆，右				
PG14	I/O	摇杆，左				
PG15	I/O	摇杆，上				

除了FSMC类的GPIO外，其它的输出型（方向=’O’）的GPIO均可直接外接其它外设，可作输出口线也可做输入口线用，用户不必担心存在板载器件和外扩设备的冲突问题。

如果用户需要将输入型(方向=’I’)的GPIO用于外设,那么需要将该GPIO上串联的0欧姆电阻去掉，否则可能会存在冲突问题。

2.5. FSMC资源分配

器件标号，芯片	地址范围	占用的FSMC管脚
U8，NOR Flash	0x6400 0000 ~ 0x64FF FFFF	A0-A22,D0-D15,NE2
U9，SRAM	0x6800 0000 ~ 0x680F FFFF	A0-A18,D0-D15,NE3
CN10，LCD	0x6C00 0000，0x6C00 0002	A0,D0-D15,NE4(外扩地址译码器)
U13，DM9000AE	0x6C10 0000，0x6C10 0008	A2,D0-D15,NE4(外扩地址译码器)
U5，CH374	0x6C20 0000，0x6C20 0008	A2,D0-D7,NE4(外扩地址译码器)

STM32的CPU同时只能使用4个片选，也就是只能外接4个总线型设备。NAND Flash已经占用一个，剩余的3个片选口线无法同时连接5个设备。

为了解决上述问题，本开发板外扩一个地址译码器电路，将 NE3 和 NE4 空间各划分出 4 等份。也就是将 2 根片选扩充为 8 根，本开发板占用其中的 4 根，剩余的 4 个片选也引到 GPIO 排针，用户还可以外接 4 个总线设备。

2.6. 时钟源

STM32F103ZE-EK开发板上有四种时钟源：

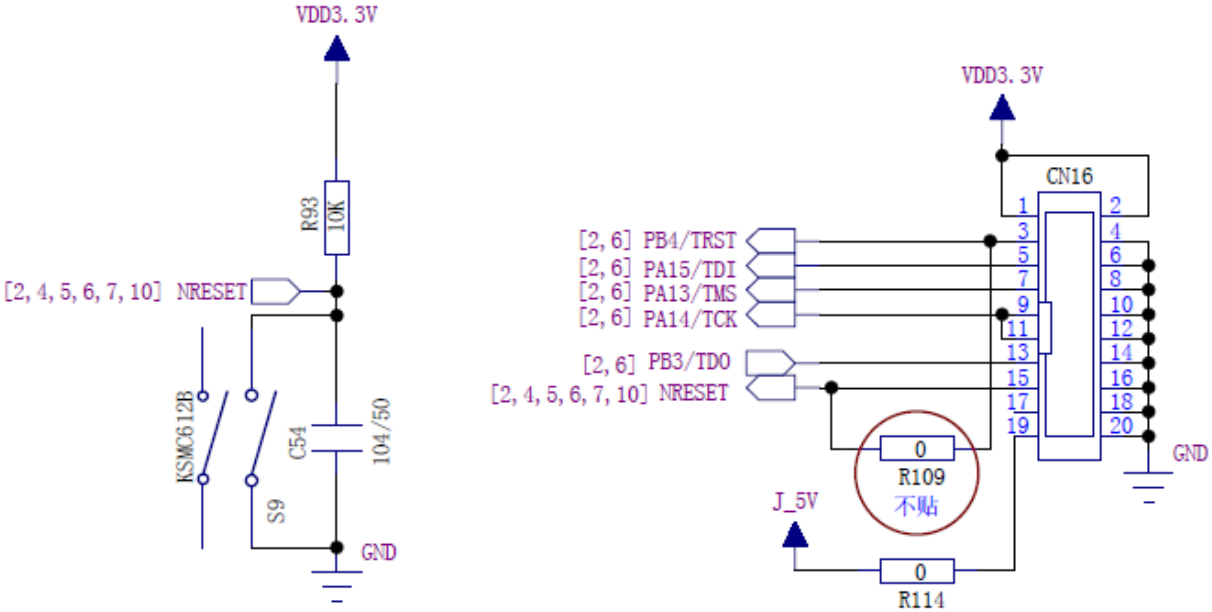
- 32KHz 晶振X3作为RTC的时钟源【注意：需要用负载电容6pF的晶振】
- 8MHz晶振X2作为MCU的时钟源
- 25MHz晶振X4作为网卡芯片的时钟源

- 24MHz晶振X1作为USB Host芯片的时钟源

2.7. 复位电路

STM32F103ZE-EK开发板有两种复位方式：

- 通过开发板上的复位按键S9复位
- 通过JTAG调试口输入复位信号



STM32F103ZE-EK开发板有两种复位方式：

- 通过开发板上的复位按键S9复位
- 通过JTAG调试口输入复位信号

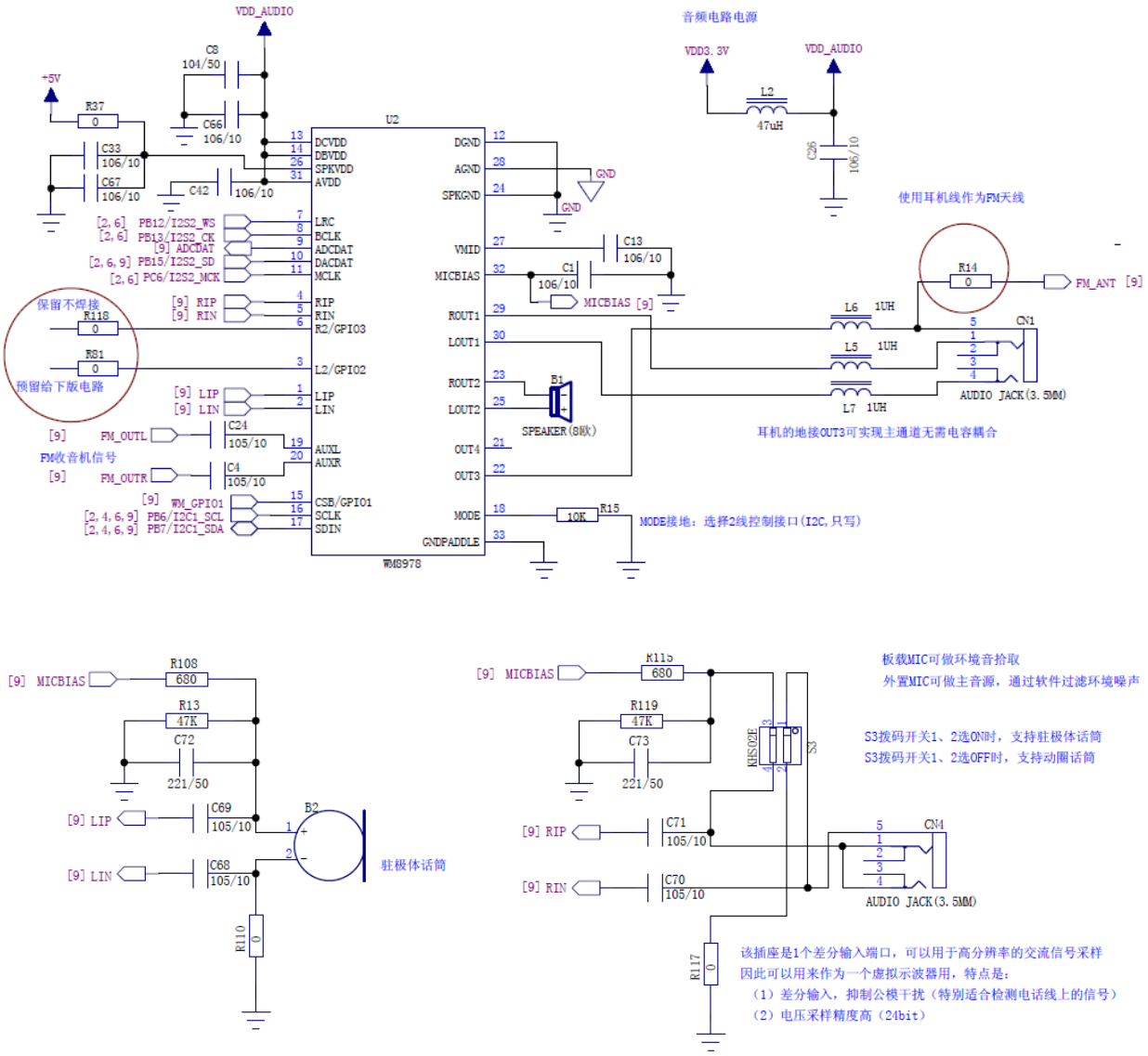
STM32这款CPU的复位引脚是低电平有效，即NRESET为低电平时，CPU处于复位状态。

R93和C54为最简单的RC复位电路。当系统上电时，C54电容两端电压为0，CPU处于复位状态。3.3V电源通过R93给C54充电，当C54的电压升到CPU的高电平门槛电压时，CPU退出复位状态转入运行状态。在设计电路时，需要选择适当的R值和C值，以保证NRESET低电平持续时间满足CPU复位最小脉宽的要求。

当按下S9轻触开关时，C54两端被短路接地，可实现手动复位CPU。

右边的电路是JTAG调试接口，用于连接调试器（比如J-Link仿真器）。CN16的15脚连接到CPU的复位引脚，因此可以通过J-Link调试器控制CPU的复位过程。

2.8. 音频电路【WM8978】



STM32F103ZE-EK开发板集成了一个音频DAC芯片WM8978，具有立体声耳机驱动输出和扬声器驱动输出。

CPU和WM8978之间通过I2C和I2S接口连接。

I2C接口（SCLK、SDIN）用于配置WM8978的工作状态（音量、外放和耳机切换、均衡控制等）。

I2S接口（LRC、BCLK、ADCDAT、MCLK）用于传输音频数据流。

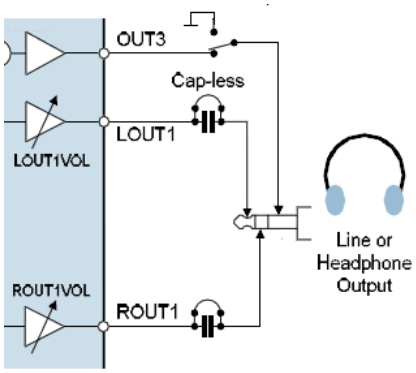
WM8978和CPU之间可以选择2线模式（I2C接口）和3线模式（SPI接口）进行连接。MODE接低电平时选择2线模式，接高电平时选择3线模式。本开发板将MODE引脚通过R15接地，因此选择的是2线接口。2线模式类似于I2C接口，但是它不是一个完整的I2C接口，因为WM8978只支持写操作，不支持读操作。读者在学习WM8978相关的例程时，必须了解这一点。在设计软件时，有时需要读回寄存器的数值修

改后再回写，我们可以通过将WM8978所有寄存器的值保存在一个内存镜像中（数组变量）中来实现读寄存器功能。每次修改WM8978的寄存器时，同时修改这个内存镜像。当读WM8978寄存器时，直接返回内存镜像中的值。

WM8978的功能很强大，具有2路立体声单端输入，1路立体声差分输入，2路立体声输出、2路单声道输出。

引脚	方向	用途
LIP，LIN	左声道，差分输入	连接一个驻极体话筒
RIP，RIN	右声道，差分输入	连接了一个音频插座，可以连接一个驻极体话筒或者动圈话筒。
L2	左声道，线路输入	保留未用
R2	右声道，线路输入	保留未用
AUXL	左声道，辅助输入	连接FM收音机芯片输出的左声道
AUXR	右声道，辅助输入	连接FM收音机芯片输出的右声道
LOUT1	左声道，第1路输出	连接了一个音频插座，可以插入立体声耳机。 为了实现无电容方式连接耳机，OUT3配置为中间电平，直接接立体声耳机的公共线。
ROUT1	右声道，第1路输出	
OUT3	第3路输出	
LOUT2	左声道，第2路输出	连接一个扬声器，芯片内部将ROUT2设置为LOUT2的反相信号
ROUT2	右声道，第2路输出	
OUT4	第4路输出	保留未用

插座CN1用于插入立体声耳机。耳机的连接方法有两种，一种是有隔直电容的（耳机公共线接地），一种是无电容的（耳机公共线接WM8978的OUT3，虚拟地）。本开发板选择的是第2种。



两种接法各有优缺点。

方案	优点	缺点
有电容	● 即可以接耳机，也可以连接功率放大器（如有源音箱） ● 避免外部设备的直流电平导	● 增加了成本，需要2个大容量隔直电容 ● 隔直电容会影响音质，造成低

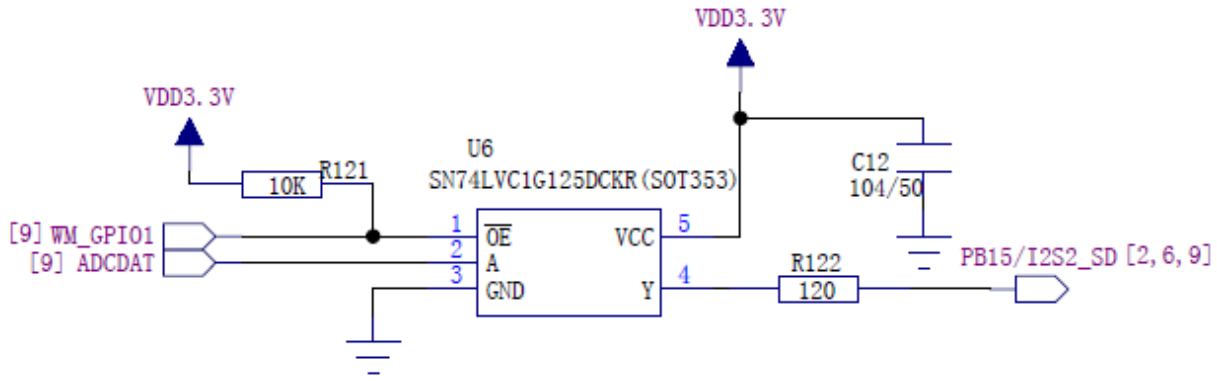
	致WM8978输出口线损坏	音成分损失
无电容	● 成本低 ● 无隔直电容，耳机音质最好	● 只能接耳机。由于有源音箱的输入插座的公共端是接地的，而开发板耳机输出插座的公共端是中间电平，会导致电平不匹配，声音严重失真。 ● 外部设备的直流电平容易导致WM8978输出口线损坏

本开发板选择无电容方案主要是从降低成本的角度出发的。用户在设计语音提示类的产品时，推荐选择有电容输出的方式。这样便于连接音箱进行功率放大。如果需要较好的音质效果，可以选择大容量的隔直电容（比如220uF以上的）。

本开发板带FM收音机功能，需要借助耳机线作为天线。耳机插座连线上串联的3个电感（L5~L7），用于隔离FM调频信号。

WM8978可以支持同时放音和录音，但是STM32的I2S接口是个“跛子”，同一时刻只能选择一个方向传输数据。标准的I2S接口的CODEC芯片（比如WM8978）具有2个独立的数据引脚，1个是ADCDAT，用于录音；1个是DACDAT用于放音。由于STM32的I2S接口只有一个数据引脚，因此在设计引脚的时候，就需要确定是录音还是放音。为了方便学习，开发板上增加了软件可控制的录音和放音切换电路。可以实现类似于对讲机的半双工模式（即不能同时录音和放音）

原理图如下：



SN74LVC1G125是一个单路数字切换开关。OE = 0时，Y = A；OE = 1时，Y = 高阻。

OE = 0 为录音状态。WM8978的ADCDAT引脚通过SN74LVC1G125连接到CPU的I2S2_SD引脚。

OE = 1 为放音状态。CPU的I2S2_SD引脚连接到WM8978的ADCDAT，WM8978的ADCDAT引脚和CPU的I2S2_SD引脚隔离，避免干扰I2S2_SD引脚上的音频数字信号。

为了节约CPU的GPIO资源，SN74LVC125的OE引脚由WM8978的WM_GPIO1控制。WM_GPIO1可以通过给WM8978发送命令设置为高或者低，从而实现录音和放音的切换。

WM8978复位之后，WM_GPIO1处于输入状态，R121上拉电阻的设计保证了SN74LVC1G125的Y输出脚在上电后处于高阻状态，从而避免和CPU的PB15引脚冲突。

R122限流电阻的设计保证了即使软件设计存在BUG(即SN74LVC1G125的Y脚输出使能 ,CPU的PB15输出使能，会造成2个输出信号短路）也不会导致硬件损坏。

我们选择小型化的SN74LVC1G125主要目的是降低PCB板占用面积。如果从成本方面考虑，大家可以选择成本低廉的通用封装的74HC125。

WM8978所有的电源引脚必须连接到合适的电源 ,所有的GND必须接地(包括芯片底部的散热焊盘)。

SPKVDD为WM8978内部扬声器功率放大器的供电电源，可以比3.3V略高（不能超过6V）以提高驱动功率。

注意： 为了降低成本，本开发板的SPKVDD直接由外部电源的+5V供电。因此本开发板的电源插座上不能连接电压大于6V的外置电源，否则会导致WM8978烧毁。用户在设计产品时，需要注意这一点。

WM8978芯片简介

WM8978是欧胜（Wolfson）推出的一款全功能音频处理器。

全球范围内，主攻音频DAC的专业厂家不多，位于英国爱丁堡的欧胜（Wolfson）微电子公司是其中一家专业公司，这家公司专门设计用于音频领域的数字模拟混合电路，包括音频编解码器、音频处理器器、音频DAC，应于与CD唱机、数字音响、数字电视、便携数字播放器、手机等领域。它的大客户很多，包括苹果IPOD、微软XBOX360、索尼PSP、健伍HI-FI CD唱机、SONY Ericsson手机等。Wolfson音频芯片的特点是种类繁多，新品迭出，无论是技术还是参数性能都处于行业领先地位。

WM8978是一款低功耗、高品质的立体声编解码芯片，主要应用于手持设备，比如数码相机、便携式摄像机。

该芯片集成了立体声麦克风差分前置放大器、扬声器和耳机驱动器、差分或者立体声线路输出驱动器。需要的外围器件很少，不需要额外的麦克风放大器和耳机放大器。

片内具备高级的数字信号处理功能，包括5段均衡、麦克风输入信号或线路输入信号自动增益控制，可以获得良好的录音和回放效果。ADC通道上可编程的滤波器适合于各种滤波应用，比如降低风噪声。

WM8978编解码器可工作在主模式或者从模式。内部PLL可以产生各种音频时钟。

WM8978的模拟部分供电电压范围为2.5V到3.3V，内核工作电压可以低到1.62V以节省功耗。扬声器输出和OUT3/4线路输出可以工作在5V电压以提高输出功率。可以通过软件关闭不需要的功能模块。

主要特性：

立体声编解码：

- DAC信噪比 ： 98dB，谐波失真 ： -84dB （48kHz采样率，A-weighted）
- ADC信噪比 ： 90dB，谐波失真 ： -80dB （48kHz采样率，A-weighted）
- 内置耳机放大器
  - 可以选择“无电容”方式

- 输出功率 ： 40mW（16欧/3.3V SPKVDD）

- 内置扬声器驱动器
  - 无电容单声道输出
  - 立体声驱动输出
  - 输出功率：0.9W （8欧 BTL方式 / 5V SPKVDD）
- 支持8、11.025、12、16、22.05、24、32、44.1及48kHz的采样速率

麦克风前置放大器：

- 立体声差分输入或者单声道麦克风接口
  - 放大器增益可编程
  - 抑制共模噪声的准差分输入
  - ADC通道自动增益控制和噪声门限控制（可编程）
- 低噪声的驻极体麦克风偏置电路

其它特性：

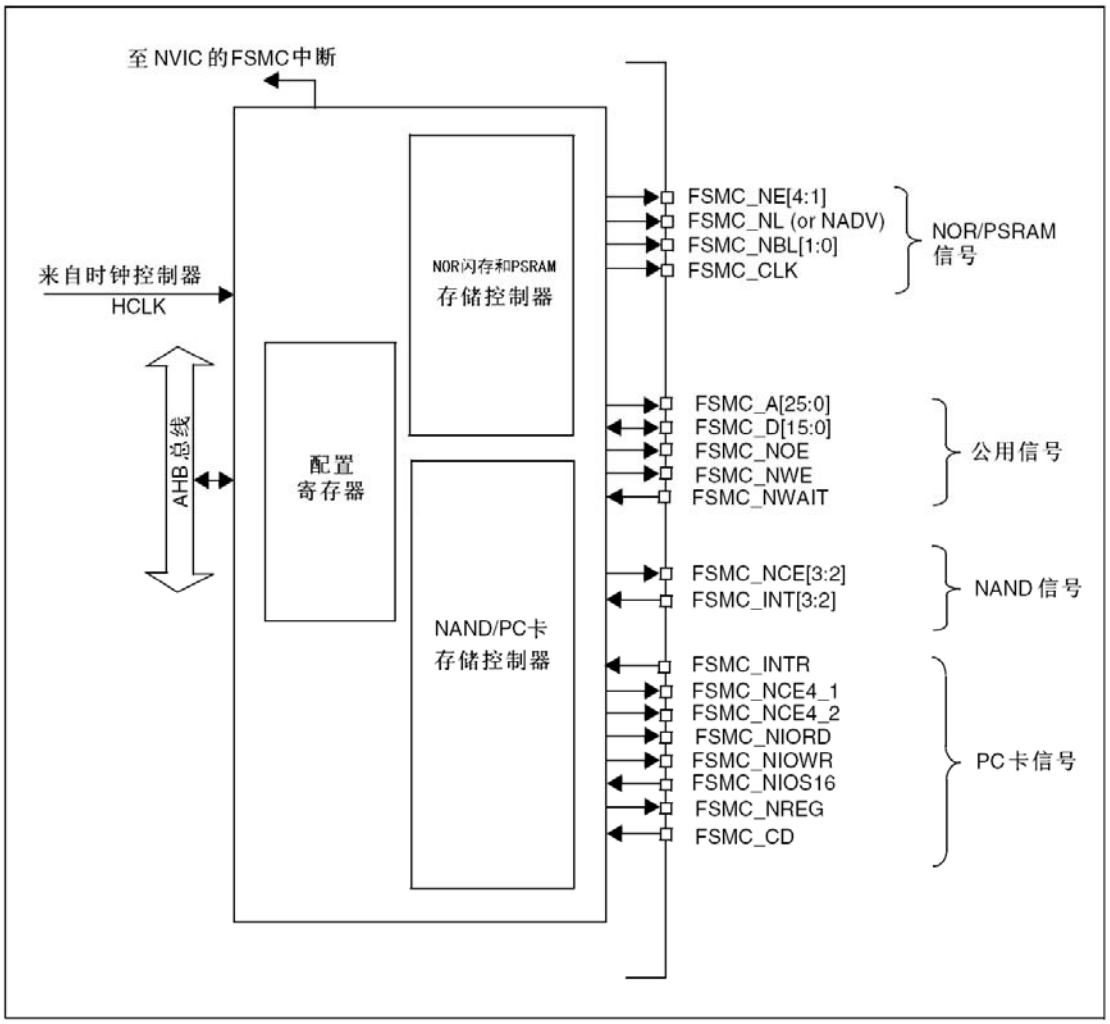
- 增强的3-D功能提高立体声分离度。
- 回放音量限制
- 5段均衡器（录音和回放）
- 可编程的ADC高通滤波器（降低风噪声）
- 可编程的ADC陷波器
- 辅助输入接口（（AUX）用于立体声模拟信号输入或者“蜂鸣提示音”输入
- 片内PLL支持12、13、19.2MHz和其它时钟
- 低功耗，低电压
  - 2.5V 到 3.6V （数字内核： 1.62V 到 3.6V）
  - 所有的功能全部打开时功耗 < 30mW （2.5V供电）
- 5x5mm 32脚QFN封装
- 工作温度范围： -25℃到+85℃

2.9. FSMC和地址译码器【74HC139】

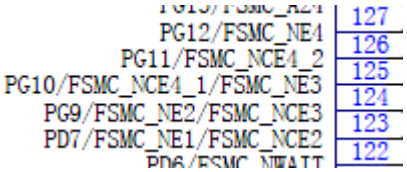
STM32103ZE这款CPU内置灵活的静态存储器控制器(FSMC),可以外扩总线型的存储器(如SRAM、NOR Flash)。FSMC的英文全称为“Flexible static memory controller”。

所有的外部存储器共享控制器输出的地址、数据和控制信号，每个外部设备可以通过一个唯一的片选信号加以区分。FSMC在任一时刻只访问一个外部设备。

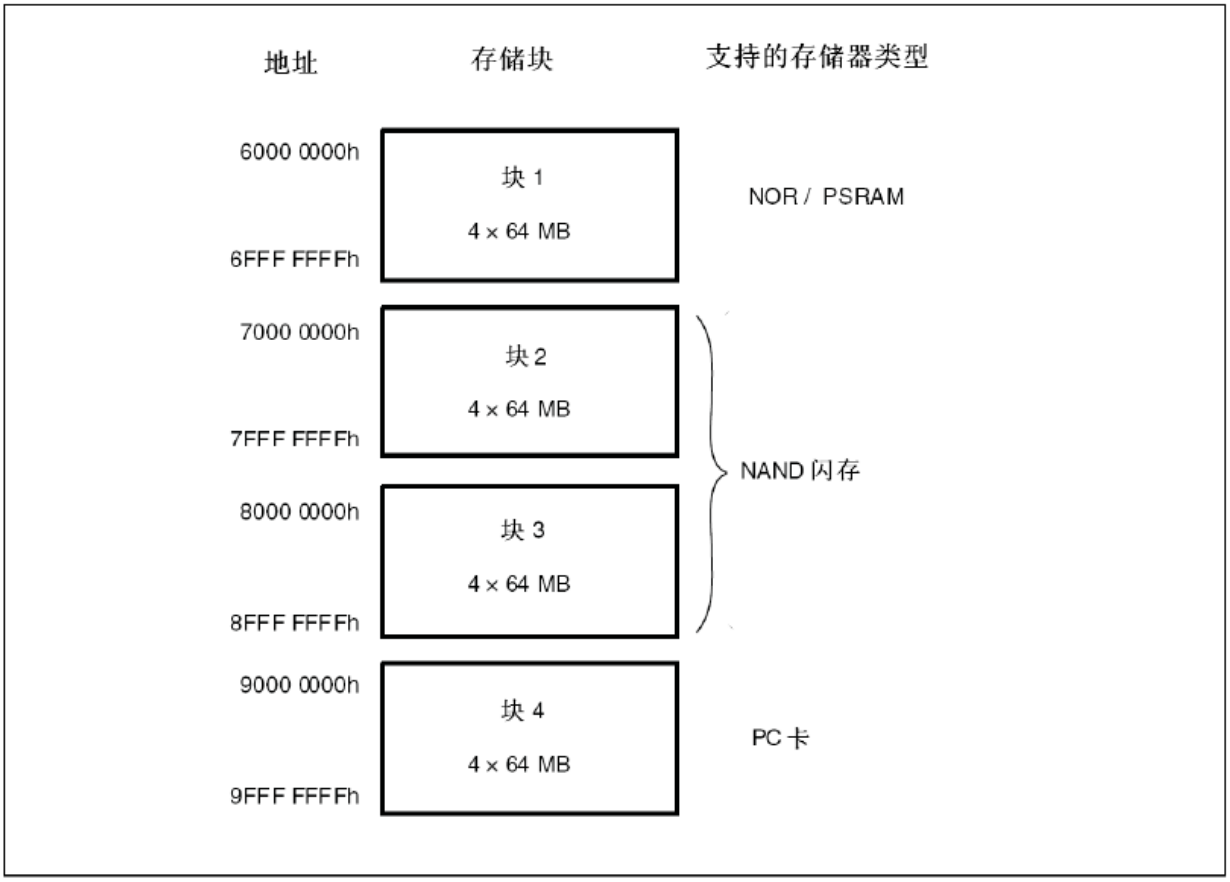
FSMC的框图如下：



注意： 上图中的有些信号是多个信号引到同一个 CPU 管脚。比如 FSMC_NCE4_1 和 FSMC_NE3 同时引到了 PG10 管脚。软件可配置 GPIO 和 FSMC 相关的寄存器进行选择。



从FSMC的角度看，可以把外部存储器划分为固定大小为256M字节的四个存储块（BLOCK），见下图。



- 存储块 1 用于访问最多 4 个 NOR 闪存或 SRAM 存储设备。这个存储区被划分为 4 个 BANK 并有 4 个专用的片选 FSMC_NE[4:1]。 CPU 的数据手册将这个区命名为 NOR/PSRAM 区，其中 NOR 指 NOR Flash；PSRAM 指伪 SRAM，即接口封装为 SRAM 形式的动态 RAM（DRAM）。其实，这个区也可以直接用于 SRAM。
- 存储块 2 和 3 用于访问 NAND 闪存设备，每个存储块连接一个 NAND 闪存。
- 存储块4用于访问PC卡设备 每一个存储块上的存储器类型是由用户在配置寄存器中定义的。

HADDR是需要转换到外部存储器的内部AHB地址线。HADDR[25:0]已连接到CPU口线，对应FSMC_A[25:0]。HADDR[27:26]仅在CPU内部使用，没有连接到CPU口线。

HADDR[27:26]位用于选择四个存储块之一：

HADDR[27:26]	选择的存储块
00	存储块1 NOR/PSRAM 1
01	存储块1 NOR/PSRAM 2
10	存储块1 NOR/PSRAM 3
11	NOR/PSRAM 4

HADDR是字节地址，而存储器访问不都是按字节访问，因此接到存储器的地址线依存储器的数据宽度有所不同，如下表：

数据宽度	连到存储器的地址线	最大访问存储器空间(位)
8位	HADDR[25:0]与FSMC_A[25:0]对应相连	64M字节 x 8 = 512 M位
16位	HADDR[25:1]与FSMC_A[24:0]对应相连，HADDR[0]未接	64M字节/2 x 16 = 512 M位

对于16位宽度的外部存储器，FSMC将在内部使用HADDR[25:1]产生外部存储器的地址FSMC_A[24:0]。不论外部存储器的宽度是多少(16位或8位)，FSMC_A[0]始终应该连到外部存储器的地址线A[0]。

请初学者仔细领会这一点，很多客户根据开发板原理图中FSMC地址线接法计算出来的地址和例程中定义的不一致，偏移了1个bit。原因就是这些存储器是按16bit模式配置的。另外在软件实现上，8bit的总线设备（比如CH374T）也是可以配置为16bit模式，访问的时候丢弃高8bit即可

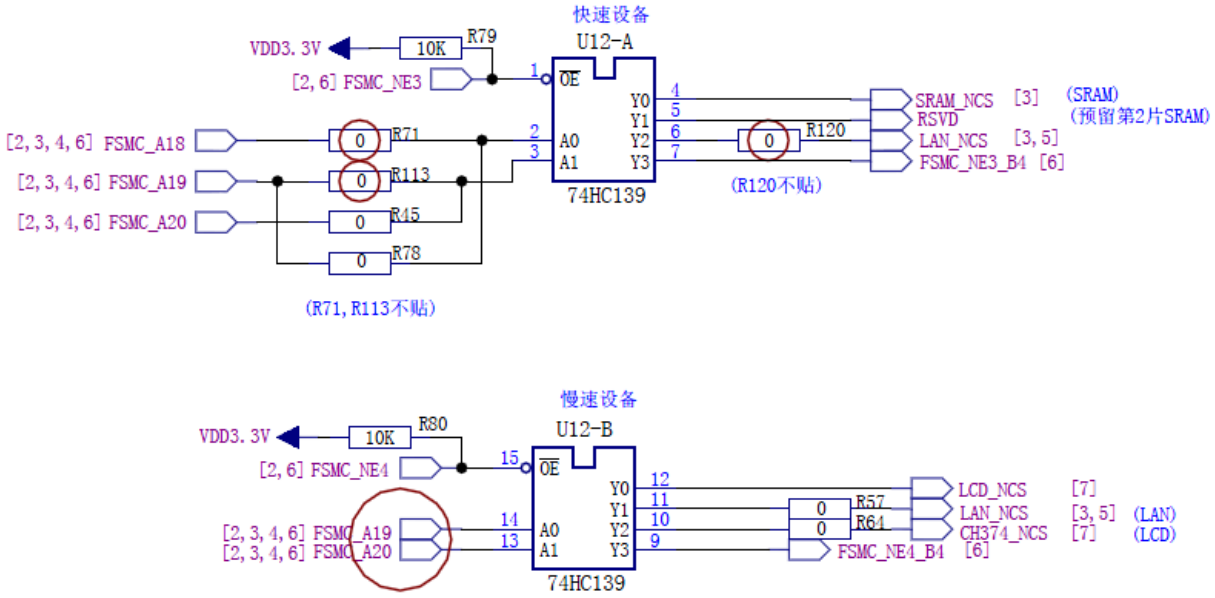
STM32的CPU仅引出了4个片选信号，也就是CPU硬件最多只能外接4个总线型设备。安富莱开发板具有6个FSMC外设。

器件标号，芯片	地址范围	占用的FSMC管脚
U8，NOR Flash	0x6400 0000 ~ 0x64FF FFFF	A0-A22,D0-D15,NE2,OE,WE
U9，SRAM	0x6800 0000 ~ 0x680F FFFF	A0-A18,D0-D15,NE3,OE,WE
CN10，LCD	0x6C00 0000，0x6C00 0002	A0,D0-D15,NE4,OE,WE
U13，DM9000AE	0x6C10 0000，0x6C10 0008	A2,D0-D15,NE4,OE,WE
U5，CH374T	0x6C20 0000，0x6C20 0008	A2,D0-D15,NE4,OE,WE
U3， NAND Flash	0x7000 0000 ~ 0x77FF FFFF	A16-17,D0-D15,NCE2,OE,WE,INT2

NAND Flash已经占用1个片选口线，剩余的3个片选口线无法同时连接5个设备。

为了解决上述问题，本开发板外扩一个地址译码器电路，将 NE4 空间分为 4 部分，第 1 份给 LCD,第

2 份给 LAN，第 3 份给 CH374T，第 4 份保留。



74HC139是双2-4线地址译码器。当74HC139的OE引脚为低电平时，Y0-Y3输出高电平；当OE引脚为高电平时，Y0-Y3中只有一个口线输出低电平（外部存储器的片选是低电平选通）。具体是哪个口线输出低电平由A0、A1的电平决定，也就是由FSMC_A19和FSMC_A20口线决定。

下面给出U12-A的真值表 （U12-B与此类同）

FSMC_NE3口线	FSMC_A[20:19]口线	74HC139输出口线状态
0	00	只有Y0输出0，其余输出1
0	01	只有Y1输出0，其余输出1
0	10	只有Y2输出0，其余输出1
0	11	只有Y3输出0，其余输出1
1	00	Y0-Y3全部输出1
1	01	Y0-Y3全部输出1
1	10	Y0-Y3全部输出1
1	11	Y0-Y3全部输出1

74HC139的OE口线连接的10K上拉电阻的主要作用是避免CPU启动运行前期发生总线冲突。CPU复位后，CPU的FSMC_NE3（PG10）、FSMC_NE4(PG12)、FSMC_A[20:19]口线缺省配置为GPIO高阻输入模式，电平状态随机的，这有可能导致多个外部存储器被选中，从而导致数据总线冲突。增加上拉电阻后，可以确保上电后外设的片选处于高电平。即使程序一开始就配置GPIO为FSMC功能，由于CPU执行这些代码需要时间，因此也会造成短时总线冲突问题。用户在设计产品硬件时，需要特别注意这一点。

安富莱开发板缺省选择FSMC_A[20:19]口线作为74HC139的地址线，主要考虑是实际存储器地址转换为16进制表示时，正好处于同一组4bit域。用户可以根据实际需要换用其它的FSMC地址线。

电路图中几个0欧姆电阻主要作用是方便用户通过飞线的方式修改硬件地址分配关系。用户在设计产品中，没有必要增加这些0欧姆电阻。

画圆框的0欧电阻是没有贴装的。

R71和R113两个0欧电阻（缺省未贴装）主要目的是可以选择FSMC_A18、FSMC_A19口线作为74HC139的地址线（开发板缺省不是选择这种接法）。按照这种电路接法，当用户再扩充1-3片1M字节的SRAM时，硬件上可以实现SRAM的整个地址空间是连续的。这对软件设计是非常有利的。

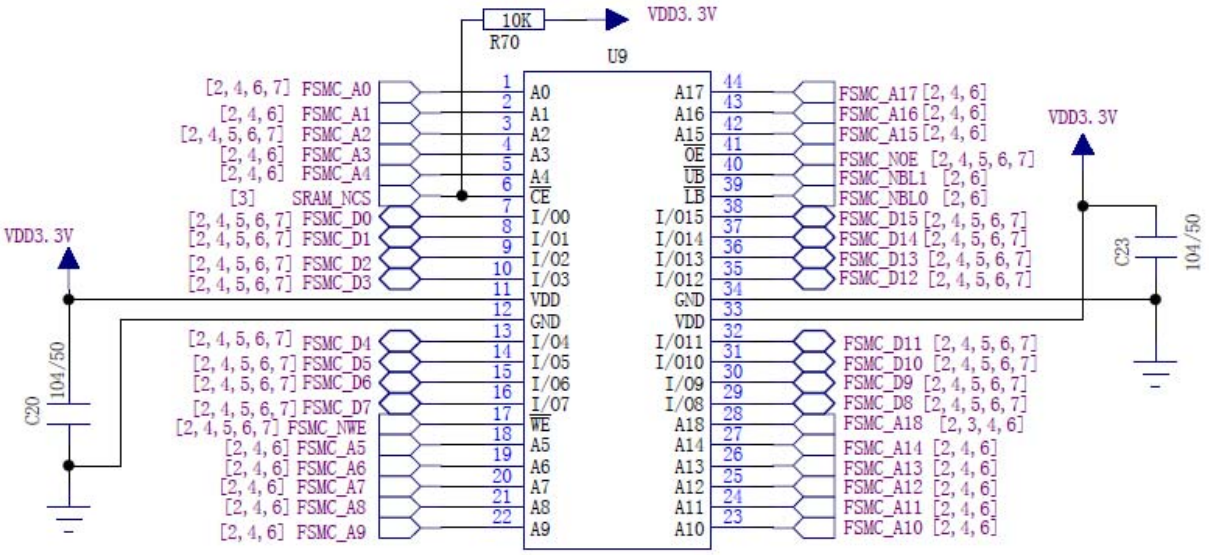
SRAM和DM9000AE速度最快， LCD写速度很快，但是读很慢，CH374T最慢。由于配置FSMC参数（包括访问速度）时，是以BANK为单位的，也就是以FSMC_NE3和FSMC_NE4所对应的空间为单位的。因此为了提高访问效率，我们需要合理的分配这些外设的地址空间。

设计R57和R120两个0欧电阻，主要是方便用户将网卡芯片DM9000AE切换到和SRAM一组，以获得较快的网络芯片访问速度。安富莱开发板没有选择这种模式，主要是为了便于开发板硬件升级，以支持4M字节的SRAM。

2.10. SRAM【EM681FV16BU-55LF】

SRAM是英文Static RAM的缩写，它是一种具有静止存取功能的内存，不需要刷新电路即能保存它内部存储的数据。SRAM掉电后，SRAM内的数据将丢失，这是和NOR Flash最大的区别。

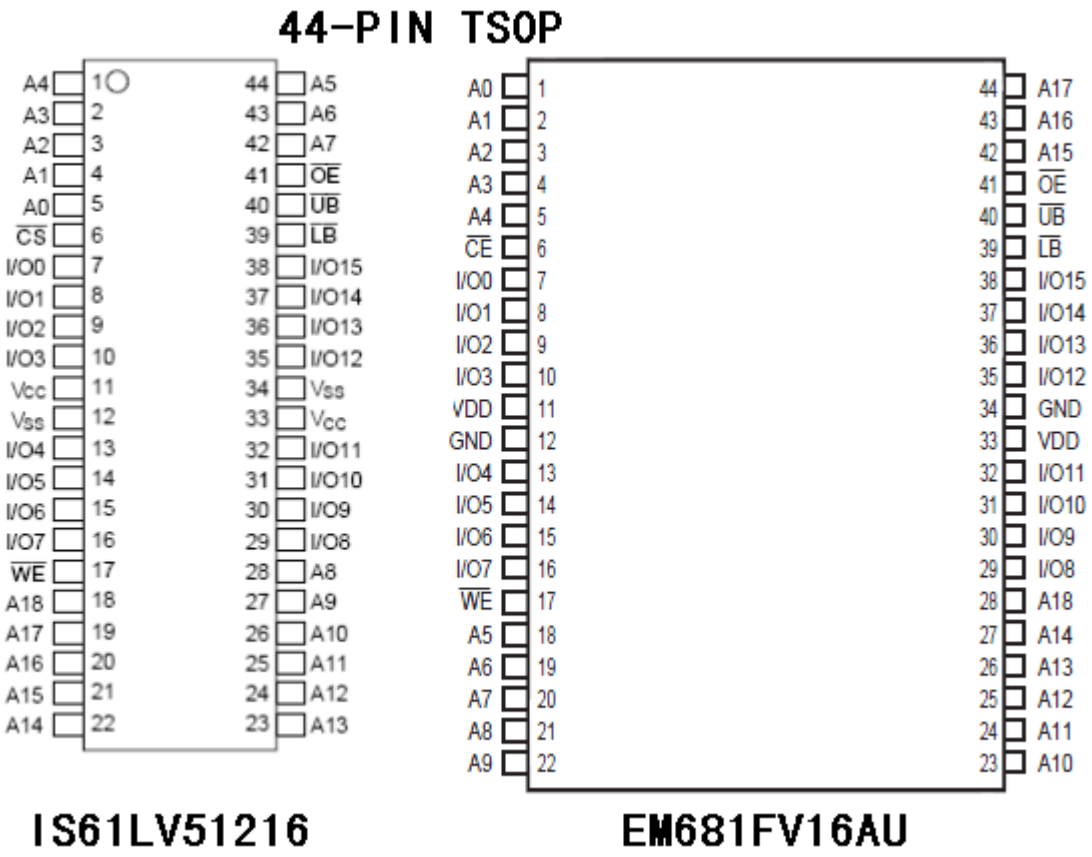
STM32F103ZE-EK开发板板载1片512X16的静态存储器(1M字节 SRAM),连接到FSMC的BANK3，支持8bit和16bit访问模式。



安富莱开发板早期硬件版本选用的SRAM为IS61LV51216-12T。这是ISSI公司生产的高速（12ns），供电电源3.3V±10%，1M字节容量，16bit数据宽度的SRAM。由于价格很高（60元），不易购买。IS61LV51216的高速性能用在STM32上有些浪费，因为STM32的FSMC速度较慢。

安富莱开发板后期版本（目前稳定供货的版本）选择的SRAM为EM681FV16AU-55LF。这是EMLSI公司生产的速度为55ns，1M字节容量，16bit数据宽度的SRAM。批量价格约20元。

这两种SRAM的管脚定义如下：



两种SRAM的封装都是TSOP-44。电源、地、控制总线、数据总线的管脚定义是一致的，但是地址线的定义顺序不一致。不过这没不影响直接替换使用，因为SRAM是一种随机访问器件。虽然内存芯片实际存放的单元地址不同，但是对于CPU软件而言是透明的，CPU软件并不关心给定的存储单元地址和实际的物理存储位置的关系。

SRAM的管脚定义如下：

引脚	功能	开发板连接方式
CS	片选，0表示选中	安富莱开发板通过外接的地址译码器输出此信号
OE	输出使能，0表示读数据	接CPU的FSMC_NOE
WE	写使能，0表示写数据	接CPU的FSMC_NWE
A0~A18	地址总线	接CPU的FSMC_A[18:0]
I/O0~I/O15	数据总线	接CPU的FSMC_D[15:0]
UB	高字节控制（I/O8~I/O15）	接CPU的FSMC_NBL1
LB	低字节控制（I/O0~I/O7）	接CPU的FSMC_NBL0
VCC	电源	接3.3V

GND	地	接地

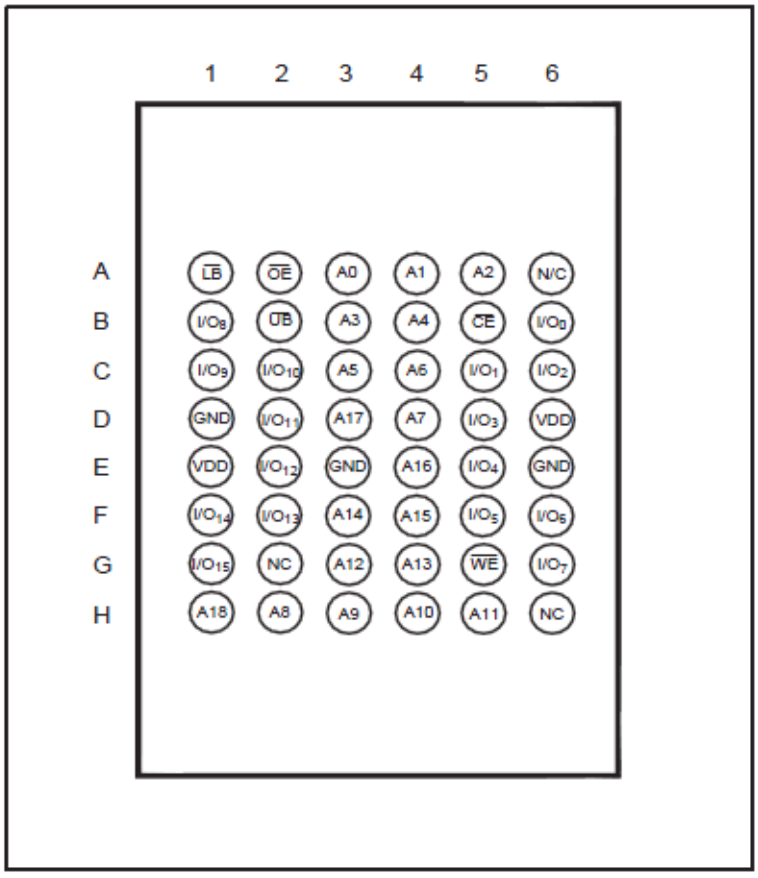
在同一时刻，CPU只能访问一个外部总线设备（SRAM、NOR Flash、NAND Flash、CH374T、LCD）。外部总线设备间的协调就考CS片选信号进行的，也就是同一时刻，只有一个外设的片选信号为低。除了片选信号外，其它的地址线、数据线、控制线都是共用的。

当SRAM的片选为高电平时，也就是不被选中时，SRAM的数据线I/O15~I/O0呈高阻状态，已免影响其它总线设备的通信。

对于TSOP-44封装的SRAM，最大容量是1M字节。如果用户需要扩大内存容量，可以采用如下两种方案：

（1） 更换为BGA封装的，BGA封装具有更多的管脚以便容纳高位地址线。

48-Pin mini BGA (9mmx11mm)



（2） 增加同样的SRAM芯片，通过不同的片选控制信号进行区分。

第1种方案的缺点是BGA芯片的焊接问题，不是所有的电子加工厂都能加工。

第2种方案的缺点是占用PCB面积大，成本略高些。但是TSSOP封装的IC很好焊接。

SRAM的PCB布线指南

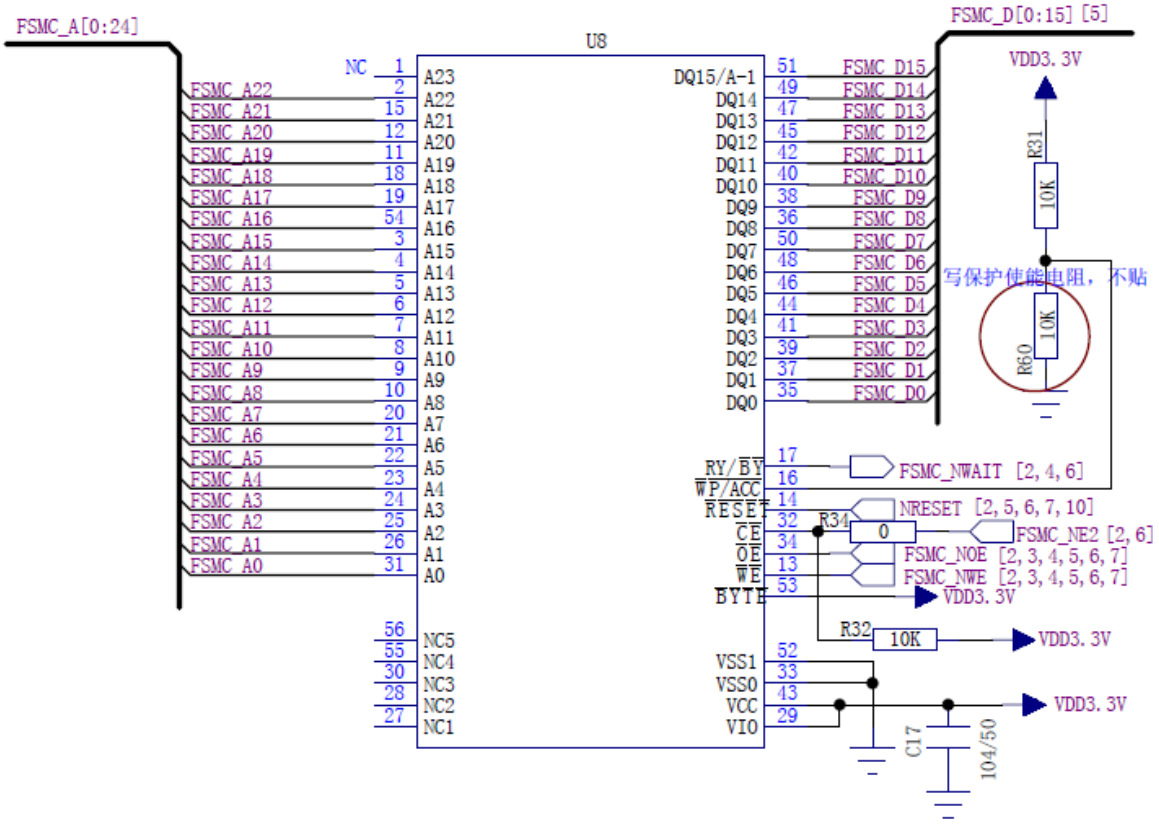
- 虽然STM32的FSMC总线时钟不高（小于100MHz），无需严格按照数据线、地址线等长规则布

线，但是走线过长，或者线长差距太大，对电路稳定性是不利的。因此请仅可能的缩短SRAM相关的走线。

- 为了布线方便，SRAM的地址线可以任意交换，不会对软件产生影响。

建议SRAM的片选口增加一个上拉电阻，保证CPU复位期间和CPU执行配置GPIO的指令之前，SRAM不被选中，已免导致总线设备的冲突问题。

2.11. NOR Flash【S29GL128P】



NOR FLASH是很常见的一种存储芯片，掉电不会丢失数据。NOR FLASH支持Execute On Chip，即程序可以直接在FLASH片内执行。这点和NAND FLASH不一样。因此，在嵌入式系统中，NOR FLASH很适合作为程序的存储介质。NOR FLASH的读取和RAM很类似，但不可以直接进行写操作。对NOR FLASH的写操作需要遵循特定的命令序列，最终由芯片内部的控制单元完成写操作。

FLASH一般都分为很多个SECTOR，每个SECTOR包括一定数量的存储单元。对有些大容量的FLASH，还分为不同的BANK，每个BANK包括一定数目的SECTOR。FLASH的擦除操作一般都是以SECTOR，BANK或是整片FLASH为单位的。

在对FLASH进行写操作的时候，每个BIT可以通过编程由1变为0，但不可以有0修改为1。为了保证写操作的正确性，在执行写操作前，都要执行擦除操作。擦除操作会把FLASH的一个SECTOR，一个BANK或是整片FLASH的值全修改为0xFF。这样，写操作就可以正确完成了。

STM32F103ZE-EK开发板板载1片16M字节的NOR Flash，连接到FSMC的NAND BANK1。

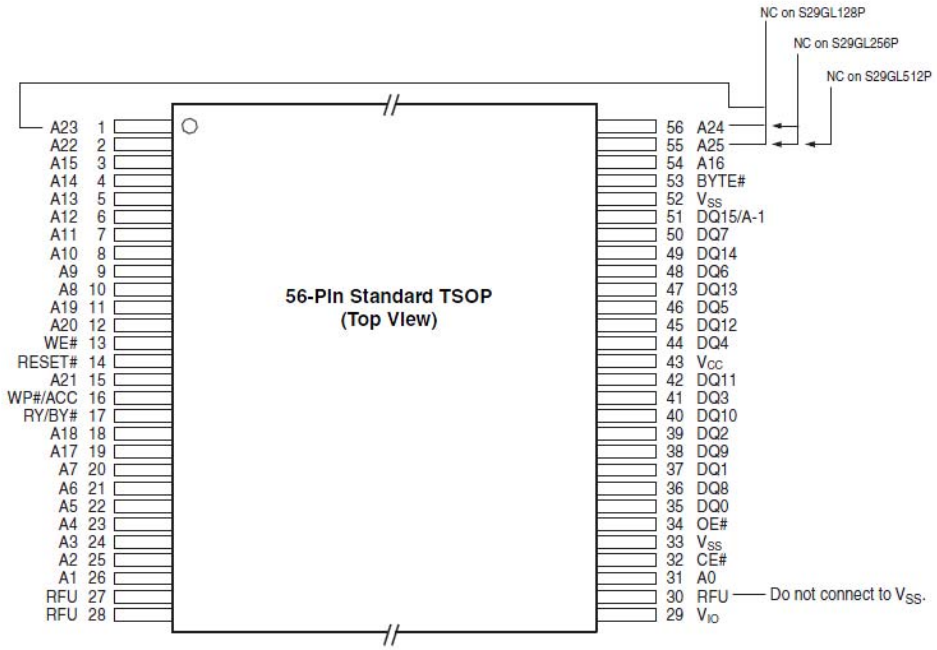
U8的16脚是写使能控制，0表示写保护，1表示允许写操作。安富莱开发板缺省未贴R60电阻，因此缺省状态下，是允许对NOR Flash进行写操作的。

R31上拉电阻（10kΩ）的主要作用是避免CPU启动运行前期发生总线冲突。

安富莱开发板早期硬件版本选用的NOR Flash为AMD公司的AM29LV128ML。

安富莱开发板后期版本（目前稳定供货的版本）选择的NOR Flash为SPANSION公司的S29GL128P10FAI01。

两种NOR Flash的封装都是TSOP-56， 容量为16M字节。管脚是兼容的，脚位图如下：



对于S28GL128P，A23、A24、A25是空脚。

管脚功能定义为：(#表示低电平有效)

引脚	功能	开发板连接方式
CE#	片选，0表示选中	接CPU的FSMC_NE2
OE#	输出使能控制，0表示读数据	接CPU的FSMC_NOE
WE#	写使能控制，低电平有效	接CPU的FSMC_NWE
WP#	写保护控制，0表示写保护使能	通过上拉电阻接3.3V电源，选择写保护禁止
BY#	忙信号，输出0表示内部操作未完	接CPU的FSMC_NWAIT
BYTE#	总线宽度选择，0表示8bit，1表示16bit	接3.3V电源，选择16bit模式
A0~A22	地址总线	接CPU的FSMC_A[22:0]
DQ0~DQ15	数据总线	接CPU的FSMC_D[15:0]
RESET#	复位信号，低电平是芯片退到读模式	接CPU的NRESET

VIO	IO电源	接3.3V电源
VCC	电源	接3.3V电源
VSS	地	接地
RFU	保留， Reserved for future use.	悬空

对于TSOP-56封装的SRAM，最大容量是64M字节。安富莱开发板将FSMC_A24、FSMC_A25所在的GPIO用作它用，因此无法再扩大NOR Flash容量。如果用户需要扩大内存容量，需要将NOR Flash的高位地址线A24、A25连接至CPU的FSMC_A24、FSMC_A25引脚。

在同一时刻，CPU只能访问一个外部总线设备（SRAM、NOR Flash、NAND Flash、CH374T、LCD）。外部总线设备间的协调就考CS片选信号进行的，也就是同一时刻，只有一个外设的片选信号为低。除了片选信号外，其它的地址线、数据线、控制线都是共用的。

当NOR Flash的片选为高电平时，也就是不被选中时，NOR Flash的数据线I/O15~I/O0呈高阻状态，已免影响其它总线设备的通信。

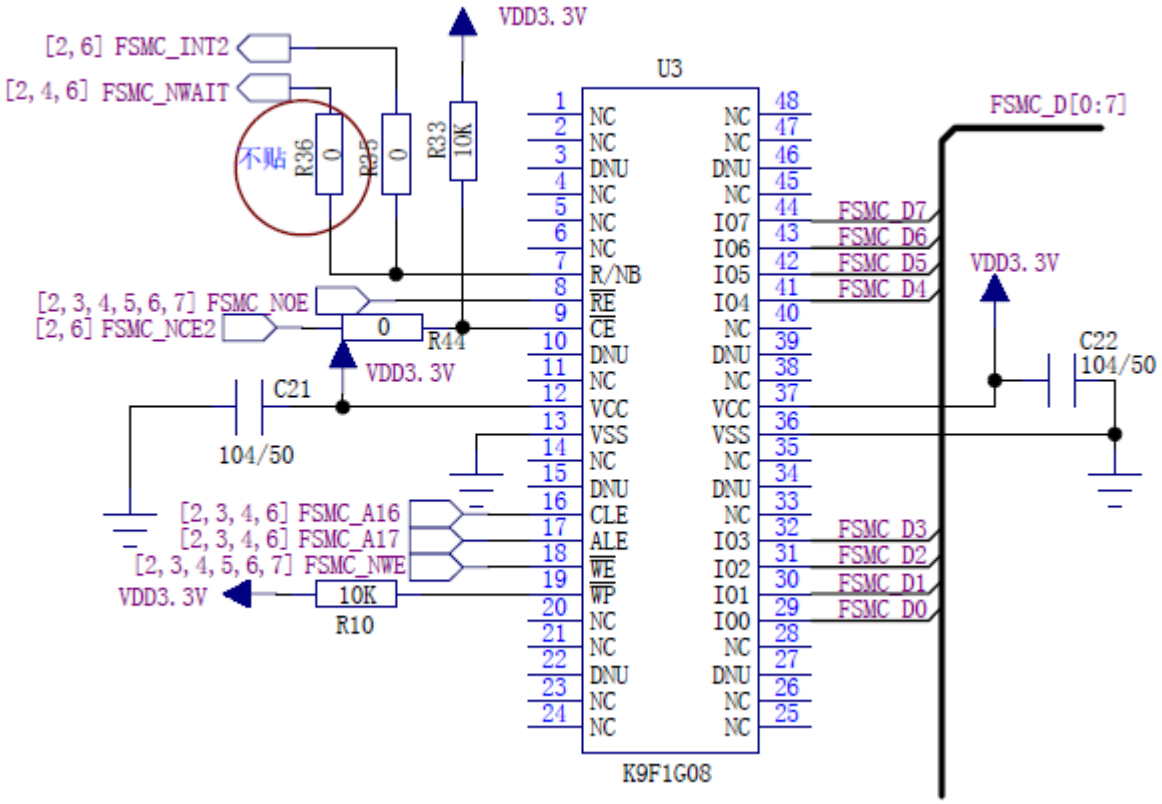
NOR Flash的PCB布线指南

- 虽然STM32的FSMC总线时钟不高（小于100MHz），无需严格按照数据线、地址线等长规则布线，但是走线过长，或者线长差距太大，对电路稳定性是不利的。因此请仅可能的缩短NOR Flash相关的走线。
- 建议片选口增加一个上拉电阻，保证CPU复位期间和CPU执行配置GPIO的指令之前，NOR Flash不被选中，已免导致总线设备的冲突问题。

2.12. NAND Flash【HY27UF081G2A】

Nand-flash内存是flash内存的一种，其内部采用非线性宏单元模式，为固态大容量内存的实现提供了廉价有效的解决方案。Nand-flash存储器具有容量较大，改写速度快等优点，适用于大量数据的存储，因而在业界得到了越来越广泛的应用，如嵌入式产品中包括数码相机、MP3随身听记忆卡、体积小巧的U盘等。

STM32F103ZE-EK开发板板载1片128M字节的NADN Flash。



U3的19脚WP是写使能控制，0表示写保护，1表示允许写操作。安富莱开发板缺省通过R10 （10K）电阻接到3.3V电源，因此缺省状态下，是允许对NAND Flash进行写操作的。

R33上拉电阻（10kΩ）的主要作用是避免CPU启动运行前期发生总线冲突。

U3的19脚CE是NAND Flash的片选信号，连接到CPU的FSMC_NCE2。

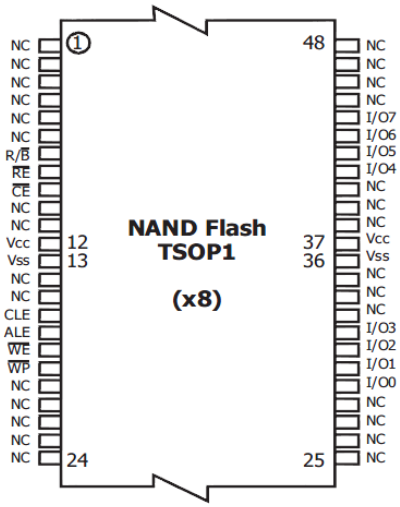
U3的7脚R/NB信号是NAND Flash忙信号，连接到FSMC_INT2引脚。当R/NB = 0时，表示NAND Flash正在执行内部操作。R36（0欧）电阻缺省未贴，针对不同的NAND Flash，可以选择将R/NB信号接到FSMC_NWAIT。一般情况下，CPU软件是通过GPIO口线检测R/NB口线的电平状态来判断NAND Flash

是否正忙。

安富莱开发板早期硬件版本选用的NAND Flash为三星(SAMSUNG)公司的K9F1G08U0A。

安富莱开发板后期版本（目前稳定供货的版本）选择的NAND Flash为海力士（HYNIX）公司的HY27UF081G2A。

两种NAND Flash的封装都是TSOP-48， 容量为128M字节。管脚是兼容的，脚位图如下：



NC表示空脚。

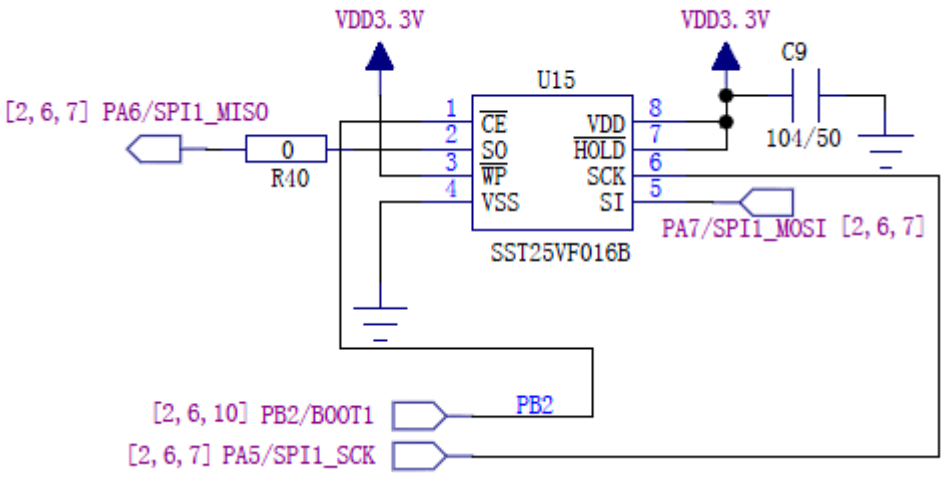
管脚功能定义为：(标号上的横线表示低电平有效)

引脚	功能	开发板连接方式
R/$\overline{B}$	忙信号，输出0表示内部操作未完	接CPU的FSMC_INT2
$\overline{RE}$	输出使能（Read Enable）0表示读数据	接CPU的FSMC_NOE
$\overline{CE}$	片选（Chip Enable）0表示选中	接CPU的FSMC_NCE2
$\overline{WE}$	写使能（Write Enable）低电平有效	接CPU的FSMC_NWE
$\overline{WP}$	写保护使能（Write Protect）0表示写保护使能	通过上拉电阻接3.3V电源，选择写保护禁止
CLE	命令锁存使能（Command Latch Enable）	接CPU的FSMC_A16
ALE	地址锁存使能（Address Latch Enable）	接CPU的FSMC_A16
I/O0~I/O7	数据总线	接CPU的FSMC_D[7:0]

VCC	电源	接3.3V电源
VSS	地	接地
NC	空脚 （No Connect）	悬空

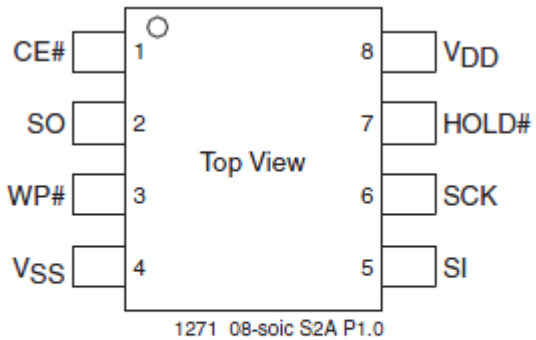
2.13. 串行Flash【SST25VF016B】

STM32F103ZE-EK开发板集成了一个串行Flash芯片SST25VF016B，容量为16Mbit（2M字节）。串行Flash连接到CPU的SPI1接口，其片选口线由CPU的PB2 (也就是BOOT1)脚控制。



串行Flash的片选虽然和BOOT1（控制引导模式）复用，但是这个地方不存在冲突问题。因为在大多数情况下，BOOT1切换到高电平（从内部Flash启动或者系统存储器启动），只有在从内部RAM启动时，BOOT1才通过1个10K电阻接地。如果你的代码需要从内部RAM启动，那么CPU初始化外设时，应该立即将PB2口线设置为输出，并且设置为高电平状态。

SST25VF016B管脚图如下：



8-LEAD SOIC

管脚功能定义为：(#表示低电平有效)

引脚	功能	开发板连接方式
CE#	片选（Chip Enable） CE由高变低时，芯片被使能。在命令传输器件，CE必须持续为低	接CPU的PB2/BOOT1 通过软件控制PB2片选的状态
SO	串行数据输出（Serial Data Output） 在SCK的下降沿，Flash的数据输出到SO口线。 如果配置了SO作为RY/BY#功能，那么在AAI自动地址递增编程期间，SO输出Flash的忙状态	接CPU的PA6/SPI1_MISO
WP#	写保护使能（Write Protect） 写保护管脚用于使能状态字中的BPL位	接3.3V电源 缺省是允许写入数据的
SI	串行数据输入（Serial Data Input） 在SCK的上升沿，CPU发送的命令、地址和数据输出到Flash	接CPU的PA7/SPI1_MOSI
HOLD#	保持使能 无需复位Flash，Hold为低时，可临时停止SPI通信	通过上拉电阻接3.3V电源，SPI通信被允许
VCC	电源	接3.3V电源
VSS	地	接地

安富莱开发板缺省安装的SST25VF016B是民用级的。如果用于工业设备，请选择工业级的型号。

级别	型号全称	温度范围
民用级	SST25VF016B-50-4C-S2AF	0~70℃
工业级	SST25VF016B-50-4I-S2AF	-40~85℃

SST25VF016B的封装的宽度比AT24CXX系列的串行EEPROM要款，大家在制作封装的时候需要特别注意。

SST25VF016B简介

SST25VF016B 是SST公司设计的一款串行Flash存储器。

其主要特点如下：

- 单电源读写操作
  - 2.7~3.6V
- 串行接口

  - SPI兼容性：支持模式0和模式3
- 高速
  - 50MHz
- 可靠性高
  - 擦写寿命：10万次（典型）
  - 数据保持时间：大于100年
- 低功耗
  - 读操作时工作电流：10mA（典型）
  - 待机电流： 5uA（典型）
- 灵活的擦除能力
  - 以4K字节的扇区为单位
  - 以32K字节块为单位
  - 以64字节块为单位
- 自动地址递增编程（AAI）
  - 字节编程操作，缩短总编程时间
- 写操作结束判断
  - 软件轮询状态寄存器的BUSY位
  - 在AAI模式，BUSY状态输出到SO口线
- 暂停控制口线（HOLD#）
  - 无需去除片选，即可挂起到设备的串行通信
- 硬件写保护（WP#）
  - 使能或除能状态寄存器的写保护锁
- 软件写保护（WP#）
  - 通过状态寄存器中的块保护位使能写保护
- 温度范围
  - 民用级：0°C 到 +70°C
  - 工业级 ：-40°C 到 +85°C
- 封装
  - SOIC-8 （200mil）
  - WSON-8 (6mm x 5mm)
- 所有的器件都是无铅的，符合RoHS规范

SPI协议简介

SPI，是英语Serial Peripheral interface的缩写，顾名思义就是串行外围设备接口。是MOTOROLA

首先在其MC68HCXX系列处理器上定义的。SPI接口主要应用在 EEPROM，FLASH，实时时钟，AD转换器，还有数字信号处理器和数字信号解码器之间。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，同时为PCB的布局上节省空间，提供方便，正是出于这种简单易用的特性，现在越来越多的芯片集成了这种通信协议，比如STM32。

SPI总线系统是一种同步串行外设接口，它可以使MCU与各种外围设备以串行方式进行通信以交换信息。外围设置FLASHRAM、网络控制器、LCD显示驱动器、A/D转换器和MCU等。SPI总线系统可直接与各个厂家生产的多种标准外围器件直接接口，该接口一般使用4条线：串行时钟线（SCK）、主机输入/从机输出数据线MISO、主机输出/从机输入数据线MOSI和低电平有效的从机选择线SS(有的SPI接口芯片带有中断信号线INT或INT、有的SPI接口芯片没有主机输出/从机输入数据线MOSI)。

SPI的通信原理很简单，它以主从方式工作，这种模式通常有一个主设备和一个或多个从设备，需要至少4根线，事实上3根也可以（单向传输时）。也是所有基于SPI的设备共有的，它们是SDI（数据输入），SDO（数据输出），SCK（时钟），CS（片选）。

（1）SDO – 主设备数据输出，从设备数据输入

（2）SDI – 主设备数据输入，从设备数据输出

（3）SCLK – 时钟信号，由主设备产生

（4）CS – 从设备使能信号，由主设备控制

其中CS是控制芯片是否被选中的，也就是说只有片选信号为预先规定的使能信号时（高电位或低电位），对此芯片的操作才有效。这就允许在同一总线上连接多个SPI设备成为可能。

接下来就负责通讯的3根线了。通讯是通过数据交换完成的，这里先要知道SPI是串行通讯协议，也就是说数据是一位一位的传输的。这就是SCK时钟线存在的原因，由SCK提供时钟脉冲，SDI，SDO则基于此脉冲完成数据传输。数据输出通过 SDO线，数据在时钟上升沿或下降沿时改变，在紧接着的下降沿或上升沿被读取。完成一位数据传输，输入也使用同样原理。这样，在至少8次时钟信号的改变（上沿和下沿为一次），就可以完成8位数据的传输。

要注意的是，SCK信号线只由主设备控制，从设备不能控制信号线。同样，在一个基于SPI的设备中，至少有一个主控设备。这样传输的特点：这样的传输方式有一个优点，与普通的串行通讯不同，普通的串行通讯一次连续传送至少8位数据，而SPI允许数据一位一位的传送，甚至允许暂停，因为SCK时钟线由主控设备控制，当没有时钟跳变时，从设备不采集或传送数据。也就是说，主设备通过对SCK时钟线的控制可以完成对通讯的控制。SPI还是一个数据交换协议：因为SPI的数据输入和输出线独立，所以允许同时完成数据的输入和输出。不同的SPI设备的实现方式不尽相同，主要是数据改变和采集的时间不同，在时钟信号上沿或下沿采集有不同定义，具体请参考相关器件的文档。

在点对点的通信中，SPI接口不需要进行寻址操作，且为全双工通信，显得简单高效。在多个从设备的系统中，每个从设备需要独立的使能信号，硬件上比I2C系统要稍微复杂一些。

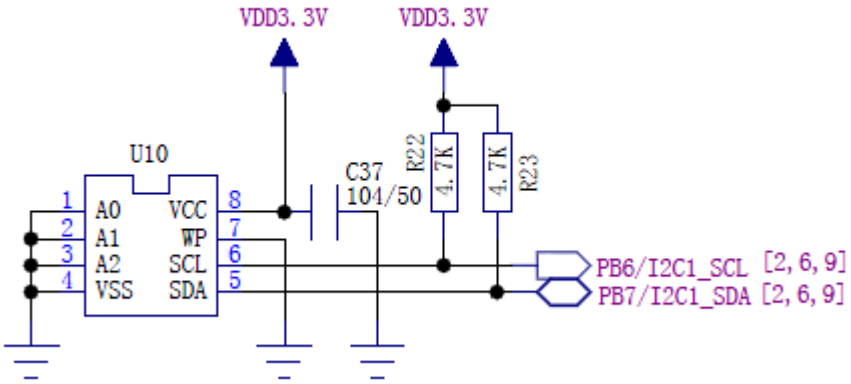
最后，SPI接口的一个缺点：没有指定的流控制，没有应答机制确认是否接收到数据。

STM32的SPI接口主要由4个引脚构成：SPICLK、MOSI、MISO及 /SS，其中SPICLK是整个SPI总线

的公用时钟，MOSI、MISO作为主机，从机的输入输出的标志，MOSI是主机的输出，从机的输入，MISO是主机的输入，从机的输出。/SS是从机的标志管脚，在互相通信的两个SPI总线的器件，/SS管脚的电平低的是从机，相反/SS管脚的电平高的是主机。在一个SPI通信系统中，必须有主机。SPI总线可以配置成单主单从，单主多从，互为主从。

2.14. 串行EEPROM【AT24C02N】

STM32F103ZE-EK开发板集成了一个2线串行EEPROM芯片AT24C02N，容量为256字节。串行EEPROM连接到CPU的I2C接口。



I2C接口上连接的设备还有音频芯片WM8978、FM收音机模块TEA5767。由于设备的从地址不同，因此不存在冲突问题。

串行EEPROM和CPU之间通过I2C总线进行通信。I2C总线是一种2线制总线。一个I2C总线上只允许一个I2C主设备（即CPU），可以允许并接很多I2C从设备，这些设备必须具备唯一的地址以便于区分。

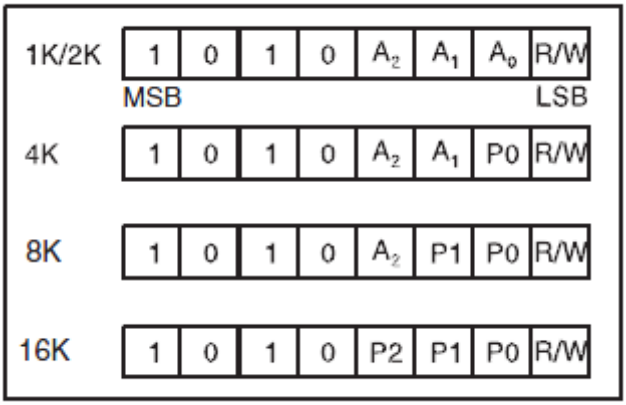
为了避免设备间硬件冲突，CPU输出的SCL和SDA均为开漏输出模式，因此必须外接上拉电阻才能输出高电平。上拉电阻越大越省电，但是信号的上升沿越平滑，这将限制I2C通信的速率。上拉电阻越小，上升沿越陡，通信速率可以更高，但是低电平输出时的功耗越大。因此需要选择合适的上拉电阻阻值以便在功耗和速度之间寻求一个平衡点。一般取1-10K欧均可。I2C总线的SCL和SDA线只需要分别放置一个上拉电阻即可，无需在每个设备上放置上拉电阻。

EERPOM的A0-A2是地址线，决定了EEPROM的I2C硬件地址，实际上一个完整的I2C硬件地址有7个bit，A0-A2只是硬件地址的一部分。

ATMEL公司的AT24Cxx系列串行EEPROM，具有如下型号：（封装兼容）

型号	容量（位/.字节）	总线扩展
AT24C01A	1K Bit （128字节）	最多允许连接8个同类设备
AT24C02	2K Bit （256字节）	最多允许连接8个同类设备
AT24C04	4K Bit （512字节）	最多允许连接4个同类设备
AT24C08A	8K Bit （1K字节）	最多允许连接2个同类设备
AT24C16A	16K Bit （2K字节）	不支持扩展

EEPROM的I2C设备地址由A0-A2引脚的电平决定。如下表：



R/W是读写控制位，R/W = 0 表示写， R/W = 1表示读操作。

对于AT24C02，当A2、A1、A0均接地时，其设备地址为 0xC0。

目前串行EERPOM的型号很多，可供选择的容量也很多，有4K字节、8K字节、16K字节、32K字节、64K字节的。

MicorChip的24LC515是一个64K字节的2线串行EERPOM，一个总线上最多允许4个器件并联，可以到达512K字节的最大容量。

由于大容量串行EERPOM的价格较高，因此在很多应用中，我们往往选择廉价的SPI接口的串行Flash来代替串行EEPROM。

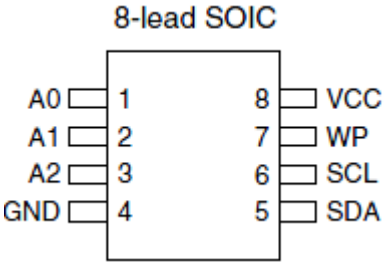
AT24C02N芯片介绍

AT24C02N是ATMEL公司设计的一款串行EEPROM，容量为256字节。

其主要特性如下：

- 低电压和标准电压两种版本
  - 2.7 （VCC = 2.7V 到 5.5V）
  - 1.8 （VCC = 1.8V 到 5.5V）
- 内部组织：128x8（1K）、256x8（2K）、512x8（4K）、1024x8（8K）、2048x8（16K）
- 2线串行接口
- 施密特触发器、滤波输入抑制噪声
- 双向数据传输协议
- 100KHz（1.8V）和400KHz（2.7V, 5V）兼容
- 高可靠性
  - 擦写寿命：100万次
  - 数据保持寿命：100年

- 汽车级应用
- 封装：8脚双列直插、8脚SOIC贴片、5脚SOT23、8脚TSSOP和8脚BGA



管脚功能定义为：(#表示低电平有效)

引脚	功能	开发板连接方式
A0~A2	地址输入	全部接地 通过配置A0-A2，最多允许接8个EEPROM设备，这些设备的I2C地址高位由A0-A2决定
SDA	串行数据（双向）	接CPU的PB7/I2C1_SDA
SCL	串行时钟	接CPU的PB6/I2C1_SCL
WP	写保护 高电平是写保护使能	接地 缺省是允许写入数据的
VCC	电源	接3.3V电源
VSS	地	接地

2.15. CAN

STM32F103ZE-EK开发板使用 SN65HVD230 (U10) 作为CAN 驱动器。CAN接口采用5.08mm间距的接线端子引出。

拨码开关S10用于设置CAN接口

位1状态	控制120欧负载
OFF	断开CAN总线并接的120欧负载电阻
ON	连接120欧负载电阻

位2状态	控制CAN芯片的RS口线状态
OFF	CAN芯片工作与高速模式
ON	CAN芯片工作与待机模式

如果需要将CANRX引脚用作它用，可以将R75电阻（0欧）去掉。

2.16. RS232 接口

STM32F103ZE-EK开发板提供2路RS232接口（DB9连接器），CPU的USART1连接到COM1端口，USART2连接到COM2端口。两个COM端口的信号均是标准的RS232电平。

DB9连接器脚位示意图如下：

COM1端口

DB9脚位	CPU引脚信号
2	USART1_TX (PA9)
3	USART1_RX (PA10)
5	GND
1、4、6、7、8、9	空脚

COM2端口

DB9脚位	CPU引脚信号
2	USART2_TX （PA2）
3	USART2_RX (PA3)
5	GND
1、4、6、7、8、9	空脚

STM32F103ZE-EK开发板和PC机连接使用直连的串口线（不是对传的串口线），市面出售的USB转串口线可以直接使用。

2.17. RS485 接口

STM32F103ZE-EK开发板提供1路RS485接口，RS485接口芯片为SP3485。485接口通过5.08mm间距的接线端子引出。

RS485芯片的数据端口连接到CPU的USART3接口，RS485芯片的接收使能和发送使能引脚通过两个独立的GPIO控制，因此可以做回环通信试验（自发自收模式）。

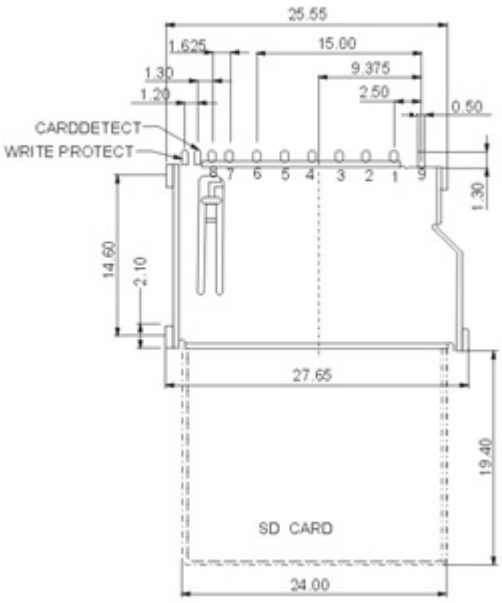
RS485总线上拉电阻、下拉电阻以及A-B间负载电阻由客户根据实际需要选贴。

如果用户需要将USRT3_RX用作它用，可以将R74电阻（0欧）去掉。

2.18. SD卡

脚位	信号名称	对应的SMT32引脚

STM32F103ZE-EK开发板具有SD卡接口，支持SD卡的读写。SD卡和STM32F103ZE-EK开发板接口的连接信号如下：



1	CD/DAT3	SD_DAT3 （PC11）
2	CMD	SD_CMD (PD2)
3	VSS1	GND
4	VDD	VDD3.3V
5	CLK	SD_CLK(PC12)
6	VSS2	GND
7	DAT0	SD_DAT0(PC8)
8	DAT1	SD_DAT1(PC9)
9	DAT2	SD_DAT2(PC10)
10	WP（写保护检测）	空
11	NCD（卡插入检测）	PC7

2.19. ADC输入

STM32F103ZE-EK开发板提供1个精密可调电阻，连接到引脚PC4/AIN12_14，可以做ADC采样试验。

2.20. 示波器电路

STM32F103ZE-EK开发板集成了1个双通道示波器前端调理电路，提供2个BNC接口，输入组阻抗为1M欧姆，支持负极信号输入，可直接使用标准的示波器探头。利用这个电路可以在开发板上进行USB虚拟示波器或者低端手持示波器项目。

通道1的模拟输出连接到CPU的AIN123_IN10、AIN123_IN12、AIN123_IN13，可以启用2个或者3个ADC对信号进行分时采样，从而到达更高的采样速率。

通道2的模拟输出连接到CPU的AIN123_IN11，可以启用两个ADC对通道1和通道2进行同步采样。

每个通道配置有2个小型拨动开关，分别控制AC/DC切换和增益切换。

拨动开关外观：

开关标号	功能	值
S5	CH1通道AC/DC切换	拨向CPU方向选择DC，反之是AC
S6	CH1通道增益切换	拨向CPU方向选择高增益(-2V~+2V) 反之是低增益（-10V~+10V）
S7	CH2通道AC/DC切换	拨向CPU方向选择DC，反之是AC
S8	CH2通道增益切换	拨向CPU方向选择高增益(-2V~+2V) 反之是低增益（-10V~+10V）

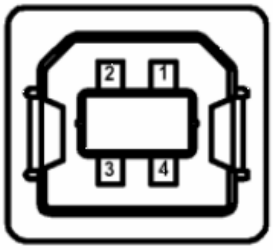
USB虚拟示波器性能指标：

序号	指标名称	值
1	模拟带宽	3MHz
2	CH1最高采样速率	3MHz
3	CH2最高采样速率	1MHz
4	采集深度	由内存分配决定
5	样本分辨率	12bit
6	AC/DC	双通道独立切换AC输入和DC输入
7	输入信号幅值范围	1档 ： -2V ~ +2V 2档 ： -10V ~ +10V 3档 ： -20V ~ +20V （探头X10） 4档 ： -100V ~ +100V （探头X10）

2.21. USB Device

STM32F103ZE-EK开发板提供了1个全速USB2.0 设备端口，通过标准的USB-B型连接器引出。通过该连接器，可以由PC机给目标供电（最大电流500mA限制）。

USB-D+信号线上的上拉电阻通过PB14引脚控制，高电平使能USB总线。



USB-B型连接器信号定义：

脚位	信号名称	对应的开发板信号
1	VBUS	USB_5V (5V电源)
2	D-	PA11/USBDM
3	D+	PA12/USBDP
4	GND	GND

和 USB-B 插座配套的 USB 电缆又称为打印机电缆，电脑市场很多柜台有售。

2.22. USB Host

STM32F103ZE处理器内部并不带USB HOST功能，因此本开发板外扩了一个常用的USB专用芯片CH374T。

CH374T 是一个USB总线的通用接口芯片，支持USB-HOST主机方式，支持低速和全速的控制传输、批量传输、中断传输以及同步/等时传输。CH374 T具有8 位数据总线和读、写、片选控制线以及中断输出，可以方便地挂接到单片机/DSP/MCU/MPU等控制器的系统总线上。

STM32F103ZE-EK开发板提供一个USB-A型连接器，可以连接U盘、USB鼠标、USB键盘等设备。USB-A型连接器对外接的设备提供5V电源。

需要注意的是，如果这个5V电源是通过USB-DEVICE接口获得的，那么需要考虑一下外接设备的功耗问题。

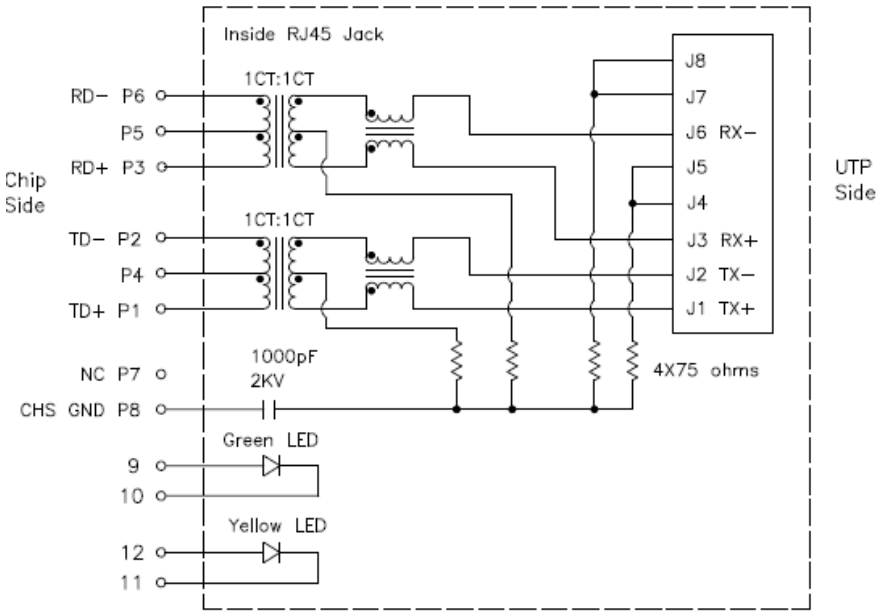
2.23. 10M/100M网卡

STM32F103ZE处理器内部没有集成 MAC电路，所以需要外接网络芯片。STM32F103ZE-EK开发板选择的网络芯片是DM9000A。

DM9000A是中国台湾DAVICOM公司推出的一款高速以太网接口芯片，其基本特征是：

- 48P LQFP封装
- 集成10/100M物理层接口
- 内部带有16K字节SRAM用作接收发送的FIFO缓存
- 遵循IEEE颁布的802.3以太网传输协议
- 支持8/16bit两种主机工作模式
- 支持唤醒帧、连接状态变化和magic包远程唤醒
- 通过HP认证的AUTO-Mdix(支持直接互连自动翻转)功能
- 支持TCP/IP加速（硬件实现IP/TCP/UDP包校验和的生成和校验）减轻CPU负担
- 支持从串行EEPROM自动装载制造商ID和产品ID(串行EEPROM是可选的)
- 支持唤醒帧、连接状态变化和magic包远程唤醒
- 具备接口速度和连接/激活状态指示灯控制信号
- 内建3.3V到2.5V的电压调节器
- 10ns I/O读写时间
- 具备待机和挂起睡眠模式

为了保证网络可靠运行，我们使用了集成网络变压器的 RJ45，HR911105A，带2个LED指示灯。



2.24. TFT显示屏和触摸屏接口

CPU没有内置触摸屏控制器，因此外扩一个触摸屏接口芯片ADS7843E（或TSC2046）。

ADS7843E和CPU间采用SPI接口通信，占用了CPU的SPI1接口，并且具有中断输出口线。当触摸屏被按压时PENIRQ口线输出低电平。

CPU没有内置LCD控制器，因此需要选用带控制器的LCD屏。STM32F103ZE-EK开发板标配的LCD屏为MP4、MP5中流行的3.0寸WQVGA（400x240）真彩屏，驱动芯片为SPFD5420A。该显示屏具有8bit和16bit两种总线模式，采用4个白色LED作为背光源，可以非常方便地和单片机接口。

触摸屏信号和LCD显示信号集通过一个2x15间距2.54mm的双排针引出。双排针的定义如下：

脚位	信号符号	中文含义	对应的CPU管脚
1	NCS	LCD控制器总线片选信号，低有效	CPU的PG12/FSMC_NE4经过地址译码器连接到NCS
2	RS	LCD控制器寄存器地址线,用于选择命令寄存器或数据寄存器	PF0/FSMC_A0
3	NWE	LCD控制器总线写使能信号，低有效	PD5/FSMC_NWE
4	NOE	LCD控制器总线写使能信号	PD4/FSMC_NOE
5	NRESET	LCD控制器总线复位信号，低有效	NRESET
6	DB0	LCD控制器总线数据线（16bit）	PD14/FSMC_D0
7	DB1	LCD控制器总线数据线（16bit）	PD15/FSMC_D1
8	DB2	LCD控制器总线数据线（16bit）	PD0/FSMC_D2
9	DB3	LCD控制器总线数据线（16bit）	PD1/FSMC_D3
10	DB4	LCD控制器总线数据线（16bit）	PE7/FSMC_D4
11	DB5	LCD控制器总线数据线（16bit）	PE8/FSMC_D5
12	DB6	LCD控制器总线数据线（16bit）	PE9/FSMC_D6
13	DB7	LCD控制器总线数据线（16bit）	PE10/FSMC_D7
14	DB8	LCD控制器总线数据线（16bit）	PE11/FSMC_D8
15	DB9	LCD控制器总线数据线（16bit）	PE12/FSMC_D9
16	DB10	LCD控制器总线数据线（16bit）	PE13/FSMC_D10
17	DB11	LCD控制器总线数据线（16bit）	PE14/FSMC_D11
18	DB12	LCD控制器总线数据线（16bit）	PE15/FSMC_D12
19	DB13	LCD控制器总线数据线（16bit）	PD8/FSMC_D13
20	DB14	LCD控制器总线数据线（16bit）	PD9/FSMC_D14

21	DB15	LCD控制器总线数据线（16bit）	PD10/FSMC_D15
22	TP_BUSY	触摸屏芯片输出的忙信号	PB5
23	TP_NCS	触摸屏芯片片选（SPI）	PG11
24	TP_SCK	触摸屏芯片SPI接口时钟	PA5/SPI1_SCK
25	TP_MISO	触摸屏芯片SPI接口数据线（触摸屏至CPU）	PA6/SPI1_MISO
26	TP_MOSI	触摸屏芯片SPI接口数据线（CPU至触摸屏）	PA7/SPI1_MOSI
27	TP_NIRQ	触摸屏芯片输出的中断信号	PC5
28	BACK_LIGHT	LCD背光控制	PB1
29	GND	地	GND
30	VCC	电源	VDD3.3V

2.25. 按键和LED指示灯

STM32F103ZE-EK开发板总共具有6个LED指示灯，包括1个电源（3.3V LDO）指示灯、4个GPIO控制的指示灯、1个WM8753音频芯片控制的指示灯。

器件标号	功能说明	对应的CPU管脚
LD1	软件自由控制	PF6
LD2	软件自由控制	PF7
LD3	软件自由控制	PF8
LD4	软件自由控制	PF9
LD5	3.3V电源指示灯	VDD3.3V
LD6	WM8753音频芯片控制的指示灯，CPU通过I2C总线向WM8753发送命令，可以控制这个LED指示灯	WM8753的GP1

STM32F103ZE-EK开发板总共具有5个按键，1个USER按键、1个Tamper按键、1个WAKEUP按键、1个5向摇杆（相当于5个独立的按键）、1个复位按键。

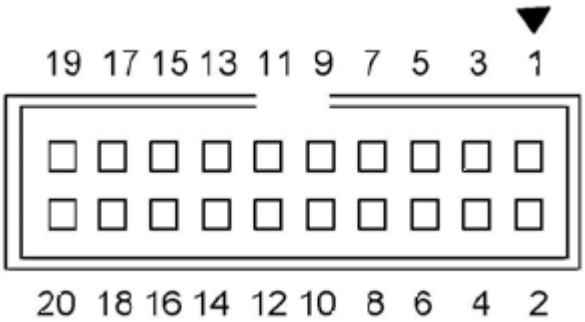
器件标号	功能说明	对应的CPU管脚
S9	复位按键	NRESET
CN4	用户自定义按键	PG8
CN5	TAMPER测试按键，也可以自定义功能	PC13/TAMPER

CN7	WAKEUO测试按键，也可以自定义功能	PA0/WKUP
CN2	5向摇杆，上键	PG15
CN2	5向摇杆，下键	PD3
CN2	5向摇杆，左键	PG14
CN2	5向摇杆，右键	PG13
CN2	5向摇杆，确认键	PG7

2.26. JTAG调试接口

STM32F103ZE-EK开发板提供了一个标准的2.54mm间距的20P插座，可以使用ARM仿真器烧写固件和调试程序。

JTAG接口定义如下：



脚位	功能描述		脚位	功能描述
1	3.3V电源（注1）		2	3.3V电源
3	TRST		4	GND
5	TDI		6	GND
7	TMS		8	GND
9	TCK		10	GND
11	RTCK （注2）		12	GND
13	TDO		14	GND
15	RESET		16	GND
17	空		18	GND
19	对目标板供电脚（注3）		20	GND

注 1：1 脚和 3.3V 网络接通。ARM 仿真器中的 JTAG 接口芯片需要通过 1 脚供电。

注 2： RTCK 功能一般用于 ARM 仿真器自动测试 CPU 的 JTAG 最高速度，但是 STM32 不支持该信号，因此在开发板上 RTCK 和 TCK 短接在一起，也就是说仿真器自动测试到 JTAG 速度不是真实速度。

注 3：19 脚和 5V 电源网络接通。J-Link 仿真器可以通过该引脚对目标板供电。

3.软件开发环境

3.1. STM32 软件开发预备知识（初学者必看）

在学习前，用户必须掌握c语言语法、变量空间、程序空间、堆栈、c编译器和调试器工作原理等相关知识，也必须了解STM32F103ZE这款CPU内部Flash和内部RAM的特性和地址空间。为了方便广大初学者学习，我们对这些概念进行一些必要的说明。

首先，我们必须明确设计程序的最终目的是生成一个可用于烧写的文件（常见的就是hex格式，本文档的附录章节有hex文件格式说明），这个文件可以通过烧写软件烧写到CPU内部Flash。在CPU上电后，CPU的硬件取指系统会自动的读取并执行Flash中的指令序列。

“源文件“是指由汇编源文件(*.s)和c源文件（*.c、*.h）构成的文件集合。我们通过编辑器（比如UltraEdit、SourceInsight或者IDE开发环境自带的编辑器）来书写源文件。源文件经过编译器（*.s由汇编编译器负责编译，*.c、*.h由c编译器负责编译）编译后生成中间目标文件(*.obj)。每个源文件会对应一个同名的obj文件，这些obj文件中的程序地址以及变量地址是未定位的。之后，由连接器负责将这些*.obj文件链接为一个完整的目标文件（如axf文件），这个axf文件包含CPU可执行的二进制代码和一些必要的调试信息。生成axf文件后，我们就可以通过集成开发环境（IDE）提供的调试接口下载程序到Flash，并通过仿真器进行跟踪调试。为了便于产品批量烧写程序，我们可以借助于IDE提供的工具，将axf文件转换为可直接烧写的文件（如hex文件）。

同一个源文件采用不同的编译设置（如预编译宏、优化级别）会生成不同的obj文件。同一组obj文件采用不同的连接设置（如程序空间地址、变量空间地址、堆栈空间）会生成不同的axf文件。在下载和调试程序，不同的硬件仿真器（如J-Link、U-Link2）需要使用不同的调试驱动程序。

集成开发环境（IDE）是一套复杂的软件，提供了编辑器、编译器、连接器等工具，并提供编译设置、连接设置、调试器接口等界面以方便软件开发。对于STM32，国内常用的IDE有Keil公司的RealView MDK和IAR公司的EWARM。

IDE通过“工程文件“来管理所有的源文件和各种设置参数。MDK的工程文件为*.uv2或*. .uvproj。EWARM的工程文件为*.eww。

所谓的“工程“是指包含源文件、工程文件甚至一些文档的文件集合。一般情况下，每个工程只包含一套源代码，但是可以包含多个target（可以翻译为目标），比如定位在内部Flash的target和定位在内部RAM的target。

为了便于工程中所有文件的查询、阅读和更改，有必要对这些文件进行分组管理。IDE具有分组管理功能，即用户可以在IDE中创建不同的group，然后将功能相关的文件放入不同的group。

STM32软件开发常用的工具有：

软件类别	可选的软件	备注
集成开发环境（IDE）	Keil公司的RealView MDK IAR公司的EWARM	KEIL公司的MDK的界面美观 ,易用性好，KEIL被ARM公司收购， 而STM32的Crotex内核也是ARM公司设计的； EWARM的开发商IAR公司是一个老牌IDE开发商，有很多其他单片机的IDE产品，这些IDE操作方法大体一致，适合于很多转型过来的用户。
源文件编辑器	UltraEdit SourceInsight IDE自带的编辑器	UltraEdit编辑源代码非常方便，原胜过IDE内置的编辑器； SourceInsight非常适合于源代码浏览
仿真器	J-Link U-Link	J-Link可用于MDK和EWARM，但是U-LINK2只能用于MDK； U-LINK2是KEIL公司开发
串口工具	Windows的超级终端 SecureCRT	Windows XP自带的超级终端是一个串口工具，可用来显示例程通过串口打印出来的信息，并将键盘输入通过串口发送到开发板； SecureCRT是一个专业的超级终端工具，支持很多接口（包括串口）和协议，推荐高级用户使用

下面我们简单介绍一下和软件设计相关的STM32和开发板的特性。

STM32F103ZE这款CPU，内部集成64K字节的RAM（首地址是 0x0800 0000）和512K字节的Flash（首地址是0x0800 0000）。安富莱STM32F103ZE-EK开发板外扩了一片1M字节的SRAM（首地址是0x6800 0000）和一片16M字节的NOR Flash（地址是0x6400 0000）。建议初学者牢记这些地址范围，这对于工程设置、调试问题排查非常有帮助。例如，程序调试异常时，可以关注一下PC指针是否在期望的地址空间。

“程序空间“指二进制可执行指令序列（包括一些常量）存放的空间，” 变量空间“指程序运行时各种变量（局部的和全局的）以及堆栈的存放空间。程序空间可以定位在CPU内部Flash、CPU内部RAM、外部SRAM和外部NOR Flash。变量空间只能定位在CPU内部SRAM和外部SRAM。

在使用STM32开发产品时，程序空间必须定位在CPU内部Flash，因为上电后，CPU都是从内部Flash开始取值运行的。在调试开发阶段，为了节省程序装载时间和延长Flash寿命，可以将代码空间定位到CPU

内部RAM或者外部SRAM。一个工程包含多个target的意义就在于此，可以方便地选择调试模式和最终成品模式两种连接配置。

在c程序中，栈（stack）用来保存函数返回地址、局部变量空间分配。堆（heap）用于通过malloc之类函数给变量分配空间。栈和堆的大小需要根据用户的具体程序进行合理设置，过小的栈和堆会引起程序执行异常。

3.2. Keil公司的RealView MDK开发软件

3.2.1. RealView MDK简介

RealView MDK开发套件源自德国Keil公司（后被ARM公司收购），是ARM公司目前最新推出的针对各种嵌入式处理器的软件开发工具。RealView MDK集成了业内最领先的技术，包括μVision3（目前已升级到μVision4）集成开发环境与 RealView编译器。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%。

3.2.2. RealView MDK的突出特性

- 启动代码生成向导，自动引导

启动代码和系统硬件结合紧密，必须用汇编语言编写，因而成为许多工程师难以跨越多门槛。RealView MDK的μVision3工具可以帮您自动生成完善的启动代码，并提供图形化的窗口，随您轻松修改。无论对于初学者还是有经验的开发工程师，都能大大节省时间，提高开发效率。

- 软件模拟器，完全脱离硬件的软件开发过程

RealView MDK的设备模拟器可以仿真整个目标硬件，包括快速指令集仿真、外部信号和I/O仿真、中断过程仿真、片内所有外围设备仿真等。开发工程师在无硬件的情况下即可开始软件开发和调试，使软硬件开发同步进行，大大缩短开发周期。而一般的ARM开发工具仅提供指令集模拟器，只能支持ARM内核模拟调试。

- 性能分析器，看得更远、看得更细、看得更清

RealView MDK的性能分析器好比哈雷望远镜，让您看得更远和更准，它辅助您查看代码覆盖情况，程序运行时间，函数调用次数等高端控制功能，指导您轻松的进行代码优化，成为嵌入式开发高手。通常

这些功能只有价值数千美元的昂贵的Trace工具才能提供。

- Cortex-M3支持

RealView MDK支持的Cortex-M3核是ARM公司最新推出的针对微控制器应用的内核，它提供业界领先的高性能和低成本的解决方案，未来几年将成为MCU应用的热点和主流。目前国内只有ARM公司的MDK和RVDS开发工具可以支持Cortex-M3芯片的应用开发。

- 业界最优秀的ARM编译器—RealView 编译器，代码更小，性能更高

RealView MDK的RealView编译器与ADS 1.2比较：

代码密度：比ADS 1.2编译的代码尺寸小10%；

代码性能：比ADS 1.2编译的代码性能高20%。

- 支持多种仿真器，自带Flash编程模块，轻松实现Flash烧写

RealView MDK支持ULINK2仿真器、J-Link仿真器，轻松实现CPU片内Flash、外扩Flash烧写，并支持用户自行添加Flash编程算法；而且能支持Flash整片删除、扇区删除、编程前自动删除以及编程后自动校验等功能，轻松方便。

3.2.3. RealView MDK安装

安富莱开发板例程使用MDK3.70和MDK 4.13a开发。光盘上提供MDK的安装软件。

光盘\04.工具软件\KEIL_MDK

名称	大小	类型
Keil uVision3(MDK3.70)破解版		文件夹
Keil uVision4(MDK4.12)破解版		文件夹
Keil uVision4(MDK4.13a)破解版		文件夹
MDK412破解版下载及说明.pdf	176 KB	Adobe Acrobat 文档
版本说明.txt	1 KB	文本文档

uVision3和uVision4的区别：

- uVision3工程文件：project.Uv2
- uVision4工程文件：project.uvproj
- uVision4版本的IDE界面好看多了，支持更多的器件。
- uVision3不能打开uVision4的工程； uVision4可以打开uVision3的工程；
- MDK4.12存在BUG，编译Ex001例程存在问题，打开MicroLib后无法编译成功，主要是由于scanf函数引起的。MDK4.13解决了scanf的问题。

安装和注册方法：

1、运行MDK413a.exe 安装MDK。建议安装路径放在C:\Keil或者D:\Keil。可以同时安装uVision3和uVision4版本，只要保证安装路径不同即可。

2、运行MDK，打开”File“下的”License Management...“，复制CID编号（11字符）。

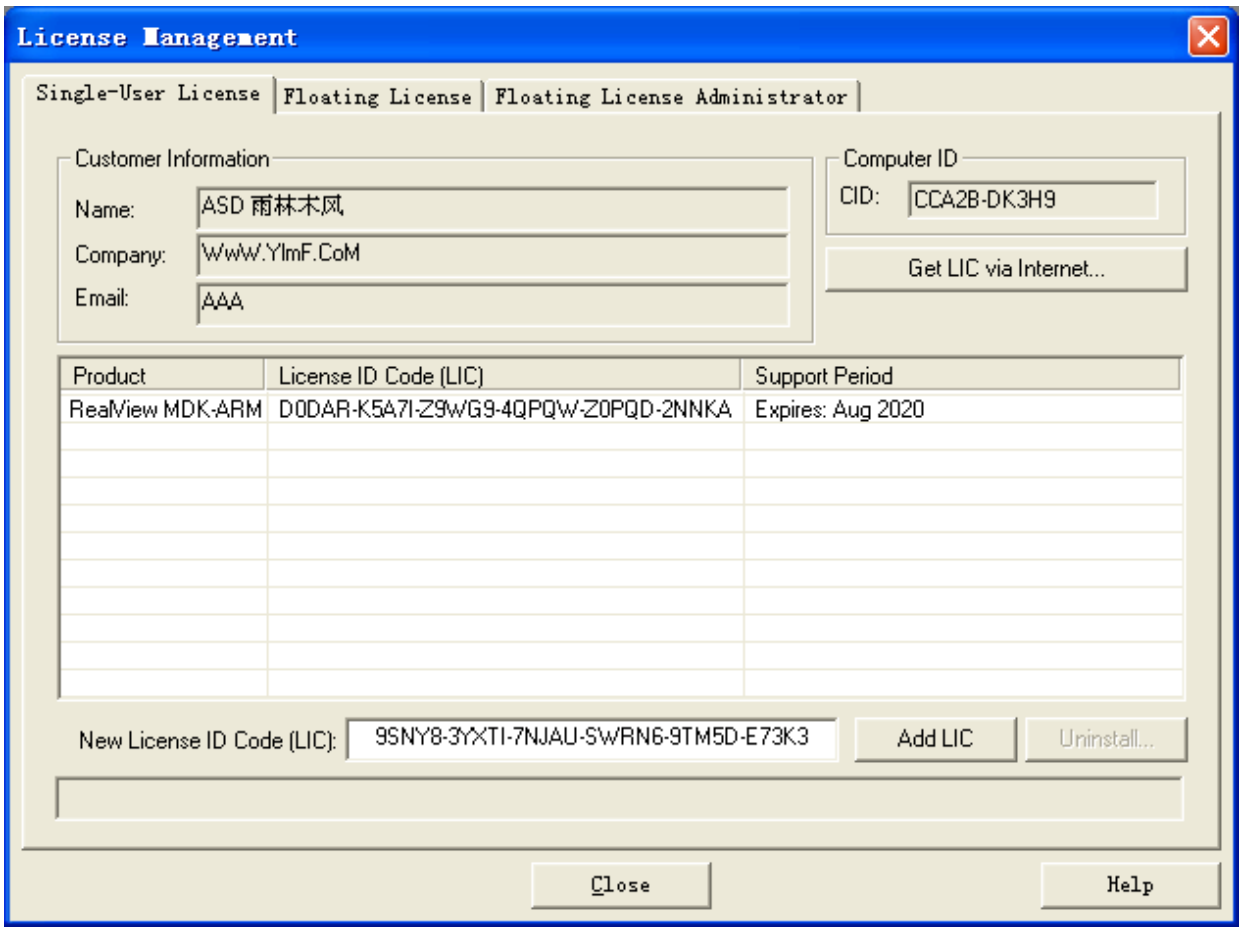
3、运行破解软件KEIL_Lic.exe。有些杀毒软件（如卡巴斯基）会误报为病毒程序，请临时屏蔽掉杀毒软件后再运行。

4、KEIL_Lic 的“target”下拉列表框选择“ARM”。

5、将MDK 的CID 编号粘贴到KEIL_lic中的CID编辑框内，然后点击“Generate”按钮。

6、把方框中生成的30位注册码，拷贝到MDK中的“New License ID Code”内。

7、点击“AddLIC”完成注册。如果截止日期太短的话，就继续用keil_Lic 生成注册码重新注册。



注意：如果J-Link仿真器内部固件的版本比2个驱动文件的版本高，可能会造成调试异常。

因此，建议手动升级MDK自带的J-Link驱动到最新版本。

具体方法为：

找到J-Link驱动程序的安装路径（比如：C:\Program Files\SEGGER\JLinkARM_V414d ）

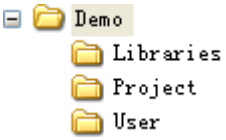
将目录下的 “JLink.exe “和” JLinkARM.dll “文件复制到MDK的安装路径(比如 :C:\KeilV4\ARM\Segger ）覆盖掉同名文件。

3.2.4. 创建MDK工程

本章节描述如何创建一个基于ST固件库V3.4.0，程序代码空间定位在CPU内部Flash、程序变量空间定位在CPU内部RAM的工程。IDE为MDK V4.13a。

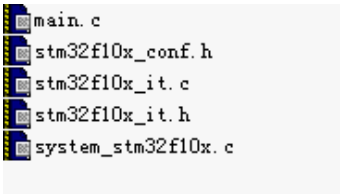
1) 解压stm32f10x_stdperiph_lib.zip。这个文件可以从ST官方网站免费下载。

2) 创建一个Demo文件夹；创建子文件夹User，用于存放用户源程序；创建子文件夹Project，用于存放工程文件。



3) 将stm32f10x_stdperiph_lib\Libraries整个文件夹复制到Demo文件夹下。这就是ST的标准库，是以源代码形式提供的。

4) 将STM32F10x_StdPeriph_Lib\Project\STM32F10x_StdPeriph_Template文件夹下的如下文件复制到Demo\User文件夹下。

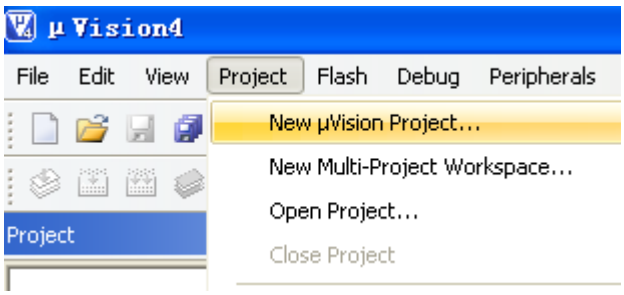


5) 将 main.c 文件内容用以下代码替换。这是一个跑马灯的程序。

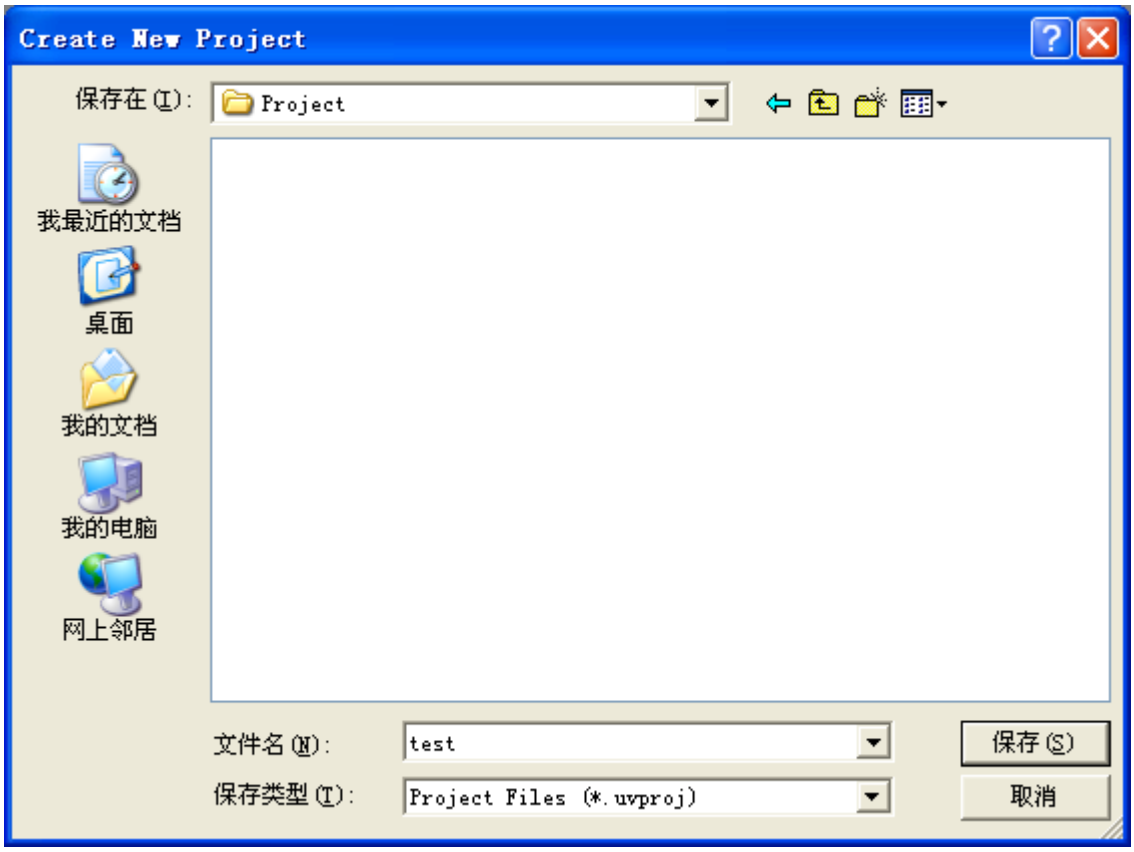
```
#include "stm32f10x.h"

/* 为了使用编程书写方便，我们定义几个控制LED开关的宏 */
#define LED1_ON()  {GPIO_ResetBits(GPIOF, GPIO_Pin_6);} /* PF6 = 0 点亮LED1 */
#define LED1_OFF() {GPIO_SetBits(GPIOF, GPIO_Pin_6);}       /* PF6 = 1 熄灭LED1 */

#define LED2_ON()  {GPIO_ResetBits(GPIOF, GPIO_Pin_7);} /* PF7 = 0 点亮LED2 */
#define LED2_OFF() {GPIO_SetBits(GPIOF, GPIO_Pin_7);}       /* PF7 = 1 点亮LED2 */

#define LED3_ON()  {GPIO_ResetBits(GPIOF, GPIO_Pin_8);} /* PF8 = 0 点亮LED3 */
#define LED3_OFF() {GPIO_SetBits(GPIOF, GPIO_Pin_8);}       /* PF8 = 1 点亮LED3 */
```

```
#define LED4_ON()  {GPIO_ResetBits(GPIOF, GPIO_Pin_9);} /* PF9 = 0 点亮LED4 */
#define LED4_OFF() {GPIO_SetBits(GPIOF, GPIO_Pin_9);}       /* PF9 = 1 点亮LED4 */

void Delay(__IO uint32_t nCount)
{
    for(; nCount != 0; nCount--);
}

int main(void)
{
    {
        GPIO_InitTypeDef GPIO_InitStructure;

        /* 打开GPIOF 时钟 */
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOF, ENABLE);

        /* 在将LED设置为输出前先设置输出1，避免一开始就点亮LED */
        GPIO_SetBits(GPIOF, GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9);

        /* 可以对PF6 - PF9 一起初始化 */
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7
            | GPIO_Pin_8 | GPIO_Pin_9;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;    /* 推挽输出模式 */
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIOF, &GPIO_InitStructure);    /* 调用库函数初始化GPIO */
    }

    /* 下面这个 while循环实现简单的跑马灯功能 */
    while (1)
    {
        LED1_ON();              /* 点亮LED1 */
        Delay(0xAFFFF);         /* 插入延时 */
        LED2_ON();              /* 点亮LED2 */
        LED3_ON();              /* 点亮LED3 */
        LED1_OFF();             /* 熄灭LED1 */
        Delay(0xAFFFF);         /* 插入延时 */
        LED4_ON();              /* 点亮LED4 */
        LED2_OFF();             /* 关闭LED2 */
        LED3_OFF();             /* 关闭LED3 */
        Delay(0xAFFFF);         /* 插入延时 */
        LED4_OFF();             /* 关闭LED4 */
    }
}
```

6) 启动MDK，点击菜单Project –> New uVision Project...

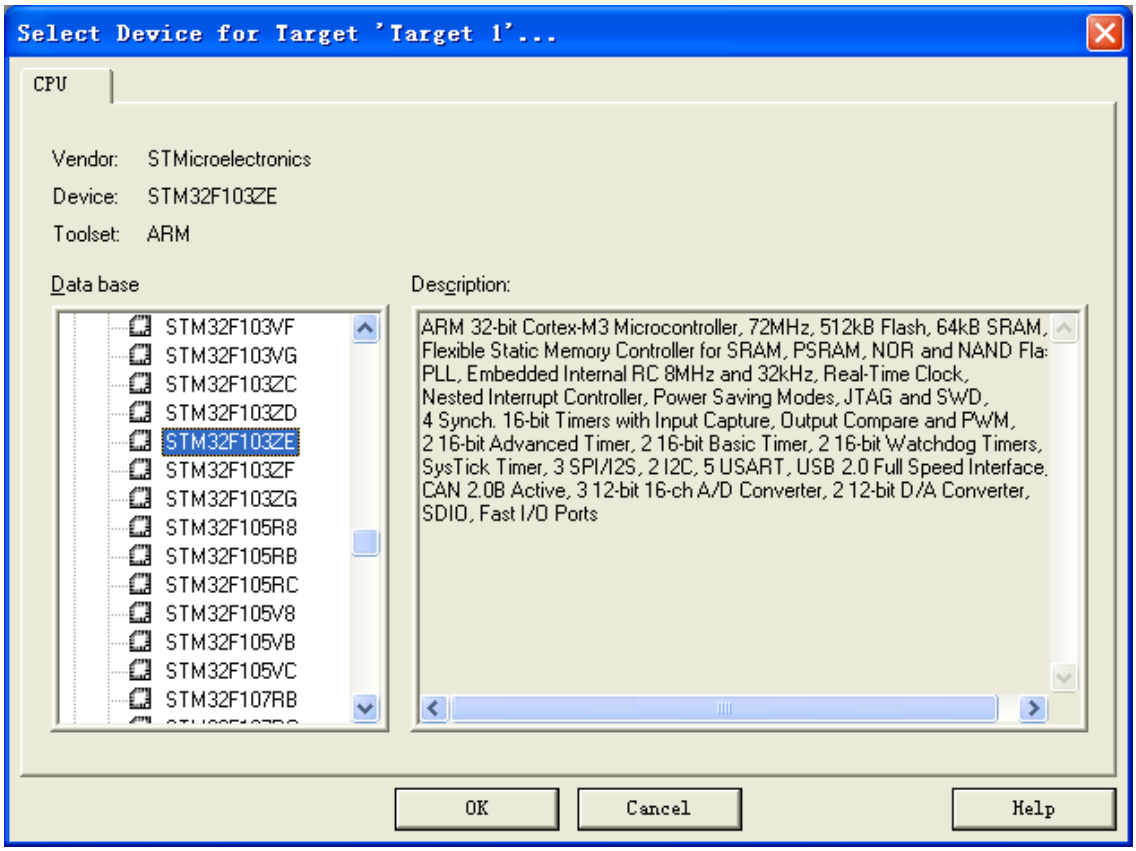


7) 在“Create New Project”对话框，选择工程文件存放路径。请指向Demo\Project，并输入工程文件名“test”，然后点击“保存”按钮。



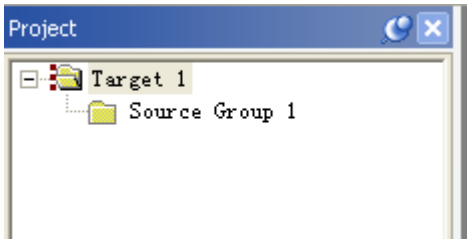
8) 选择CPU型号。请指向STM32F103ZE，然后点击“OK”按钮。

9)



10) 当提示是否复制启动代码时，请选择否。（我们用最新的ST固件库中的启动代码，不用Keil软件自带的旧版本启动文件）

11) 此时MDK会自动创建一个Target，名字为“Taeget 1”，将鼠标光标移到“Target 1”上单击一下鼠标左键，此时可以根据需要修改Target名字。我们将其修改为“Test”。



12) 为了便于代码管理，在这个Project下创建几个Group (名字可以任意)

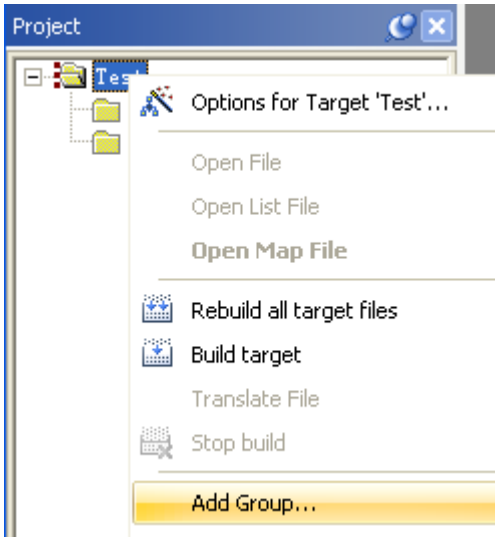
User ：存放用户自己写的源代码

RVMDK：存放MDK适用的启动文件（汇编文件）

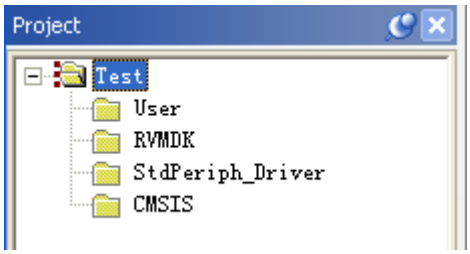
StdPeriph_Driver：存放ST标准库文件

CMSIS：存放CMSIS接口文件（这也是库的一部分）

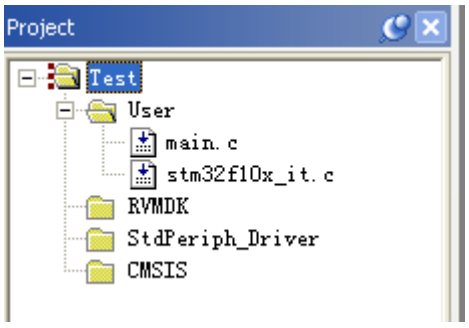
将鼠标移到Test上，然后点击鼠标右键，执行“Add Group...“ 命令，可以创建一个分组并修改分组的名称。



13) 依次创建其它几个Group，最终结果如下：

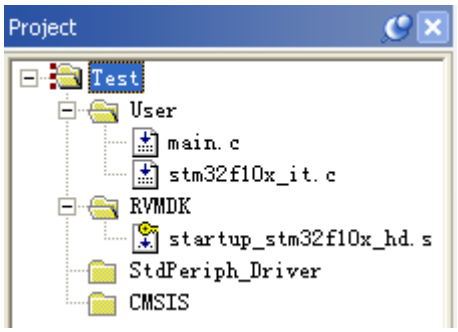


14) 双击“User”添加添加2个源代码文件，结果如下：



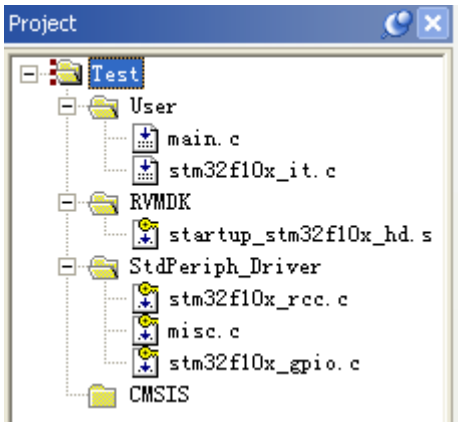
15) 双击“RVMDK”添加1个启动文件startup_stm32f10x_hd.s，

文件路径：Demo\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x\startup\arm，结果如下：



16) 双击“StdPeriph_Driver”添加本例程用到ST固件库文件，

文件路径：Demo\Libraries\STM32F10x_StdPeriph_Driver\src，结果如下：

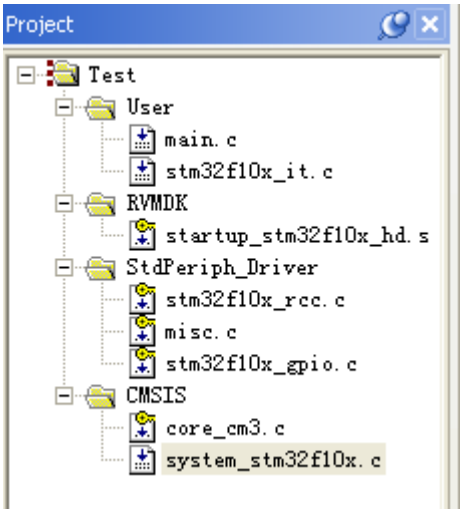


17) 双击“CMSIS”添加2个文件：

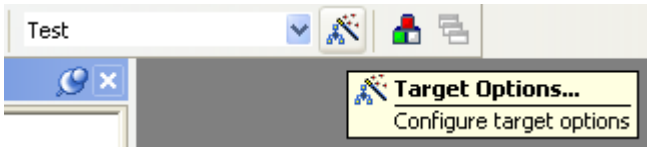
Demo\Libraries\CMSIS\CM3\CoreSupport \core_cm3.c

Demo\User\system_stm32f10x.c

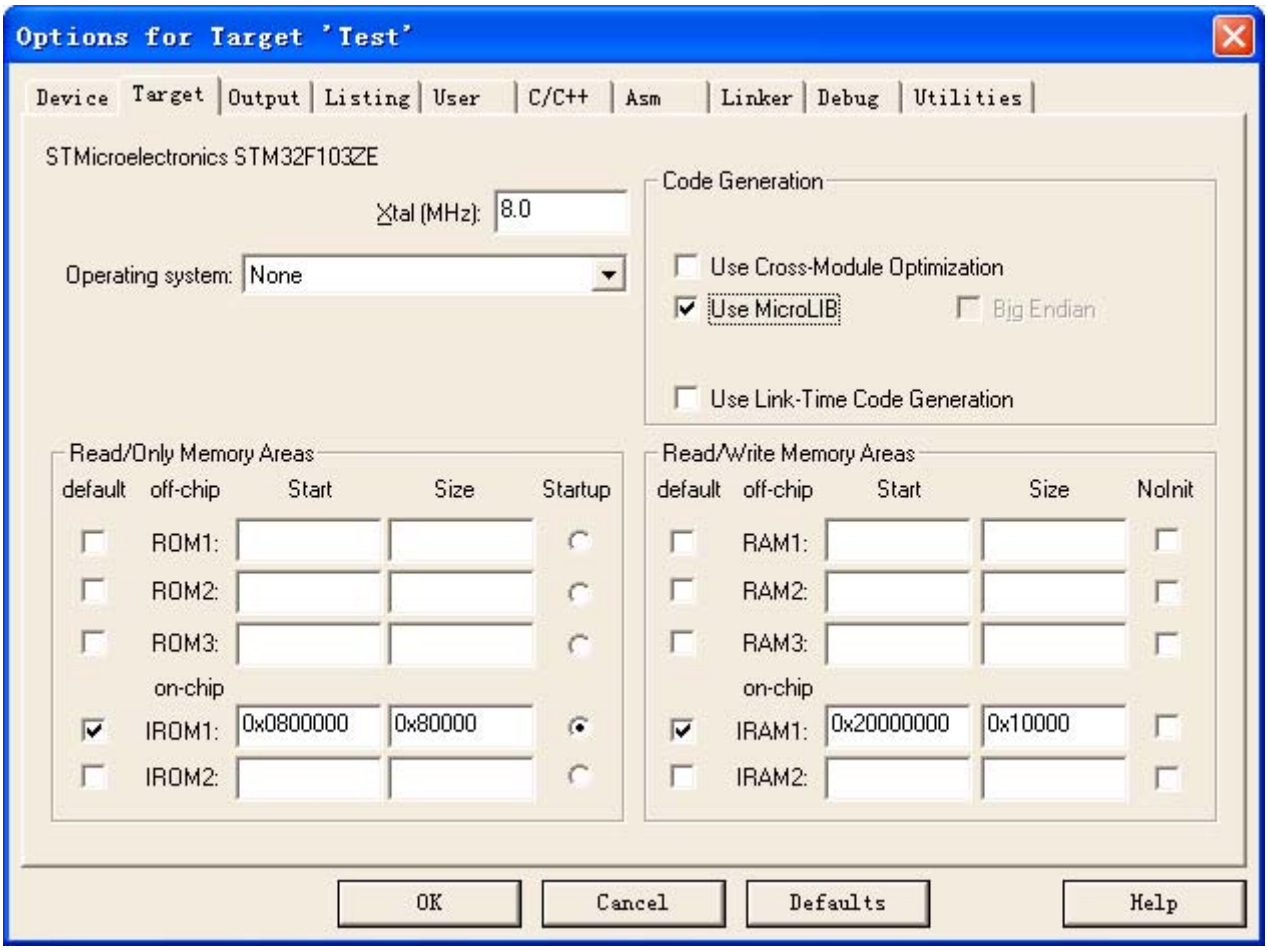
至此所有的文件添加完毕，结果如下：



18) 点击工具栏上的“Target Options...”按钮。

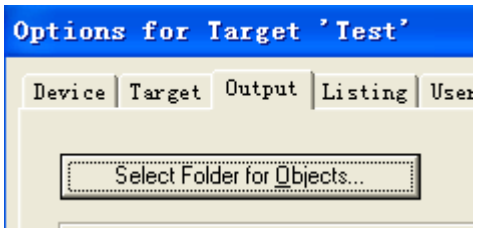


19) 点击工具栏上的“Target Options...”按钮，出现如下界面。在“Use MicroLIB”前打钩。如果不打钩，那么通过USART1执行printf函数将无输出（不过本例程未使用printf）



请确认IROM1的起始地址为0x0800 0000，IRAM1的起始地址为0x2000 0000。这2个地址决定了程序空间定位CPU内部Flash，变量空间定位在CPU内部RAM。

20) 切换到“Output”页签，点击 “Select Folder for Objects...”按钮，指定编译时生成的中间目标文件*.obj存放文件夹。这个不是必须的。不过为了便于管理，我们可以在Project下新建一个Obj文件夹，然后将obj存放文件夹指向Protect\Obj

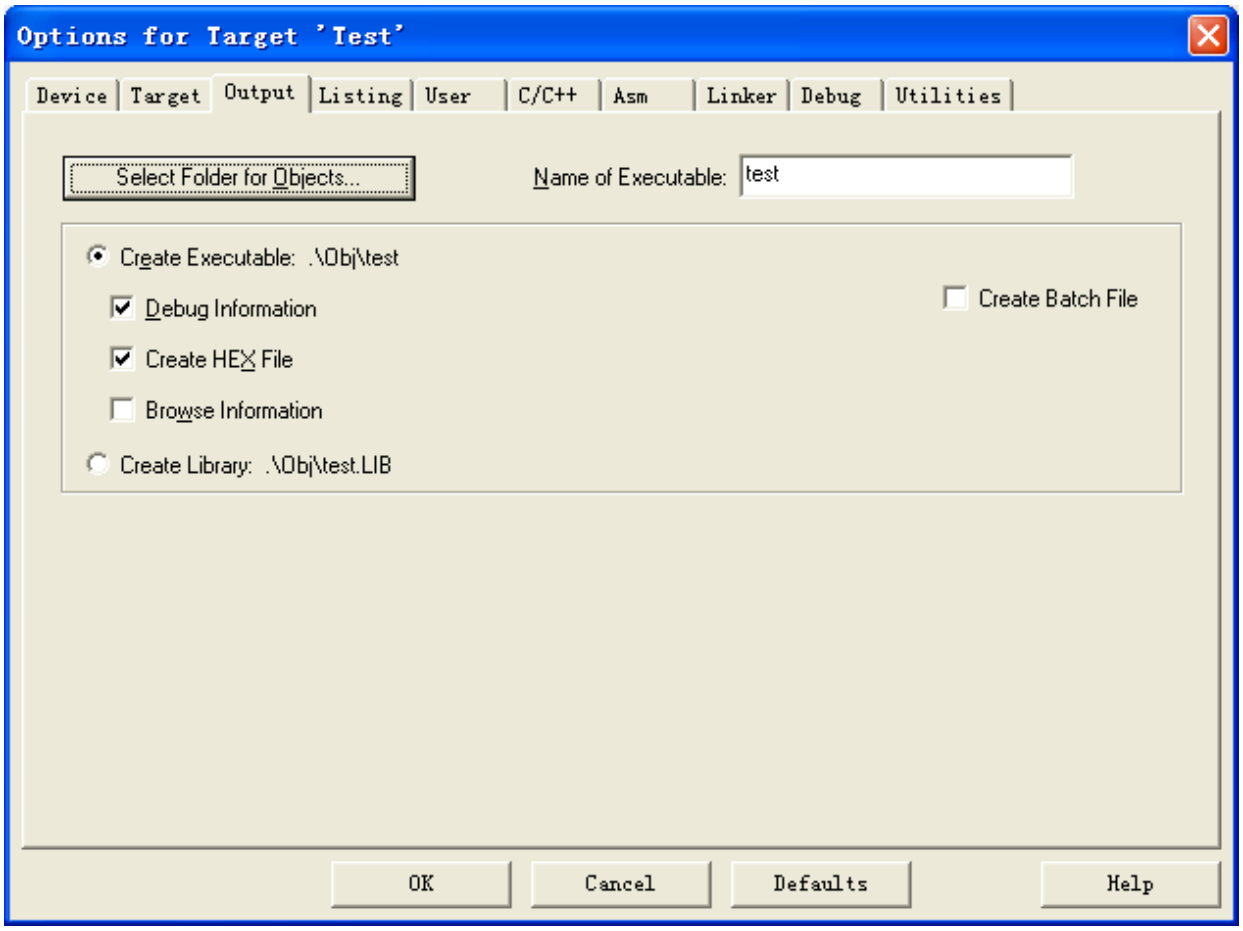


21) 在“Output”页签，选择“Create Executable .Obj\Test”，在Name of Executable编辑框中输入目标文件名，在此我们输入“test”，即生成的test.axf文件和test.hex文件。

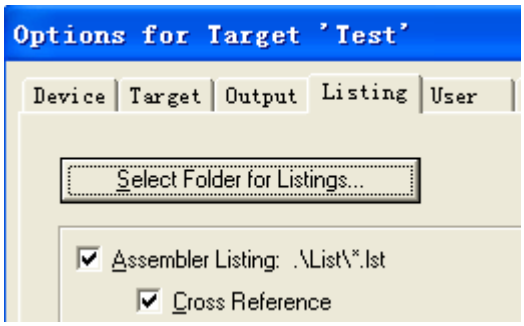
a）取消“Browse Infomation”前面的钩，不是必须的。因为勾选“Browse Infomation”后，会导致编译速度很慢。取消这个钩，会导致无法使用F12快捷键快速定位到函数原型。

b）请在“Create HEX File”前面打钩，不是必须的。这个表示自动生成hex文件，这个hex文件可以用来脱离IDE环境进行烧录程序。

最终的“Output”页签设置如下：



22) 切换到“List”页签，点击 “Select Folder for Objects...”按钮，指定编译时生成的中间列表文件*.lst存放的文件夹。这个不是必须的。不过为了便于管理，我们可以在Project下新建一个List文件夹，然后将存放文件夹指向Protect\List。

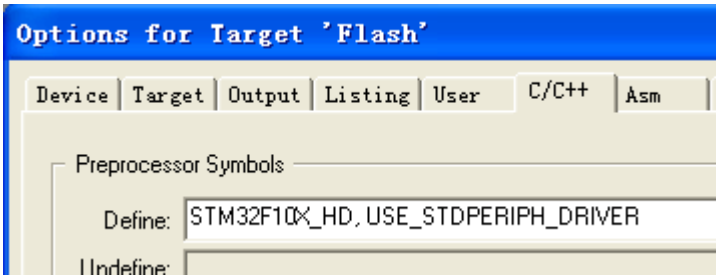


23) 切换到“C/C++”页签，增加2个预定义宏：

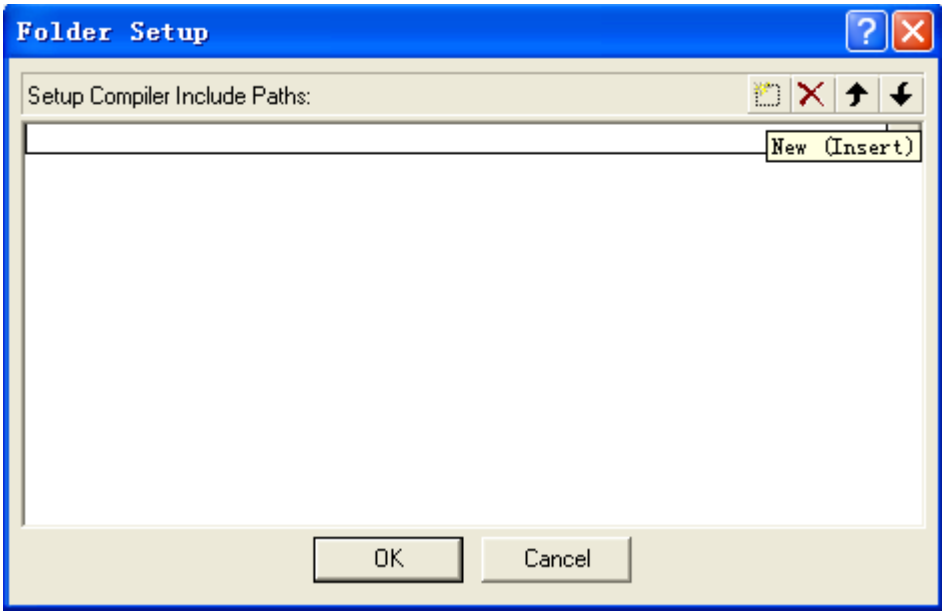
STM32F10X_HD 用于ST固件库，表示CPU类型为HD高密度型。

USE_STDPERIPH_DRIVER 用于ST固件库，表示使能ST固件库。

两个宏之间用逗号分隔，设置后的结果如下：



24) 继续设置“Include Paths”，增加头文件搜索路径。点击Include Paths后面的  按钮出现“Folder Setup”窗口



请依次添加如下路径：

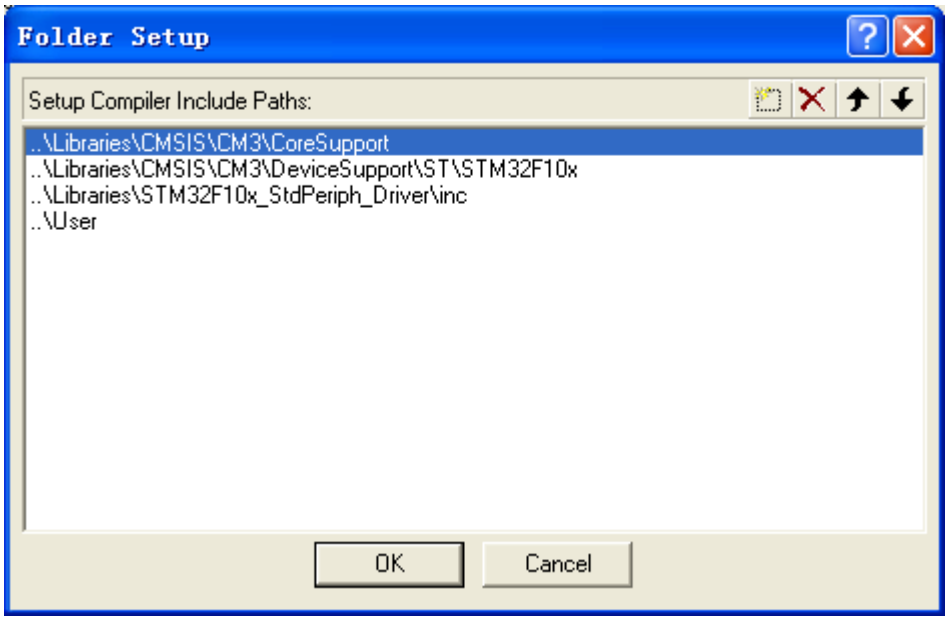
..\Libraries\CMSIS\CM3\CoreSupport

..\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x

..\Libraries\STM32F10x_StdPeriph_Driver\inc

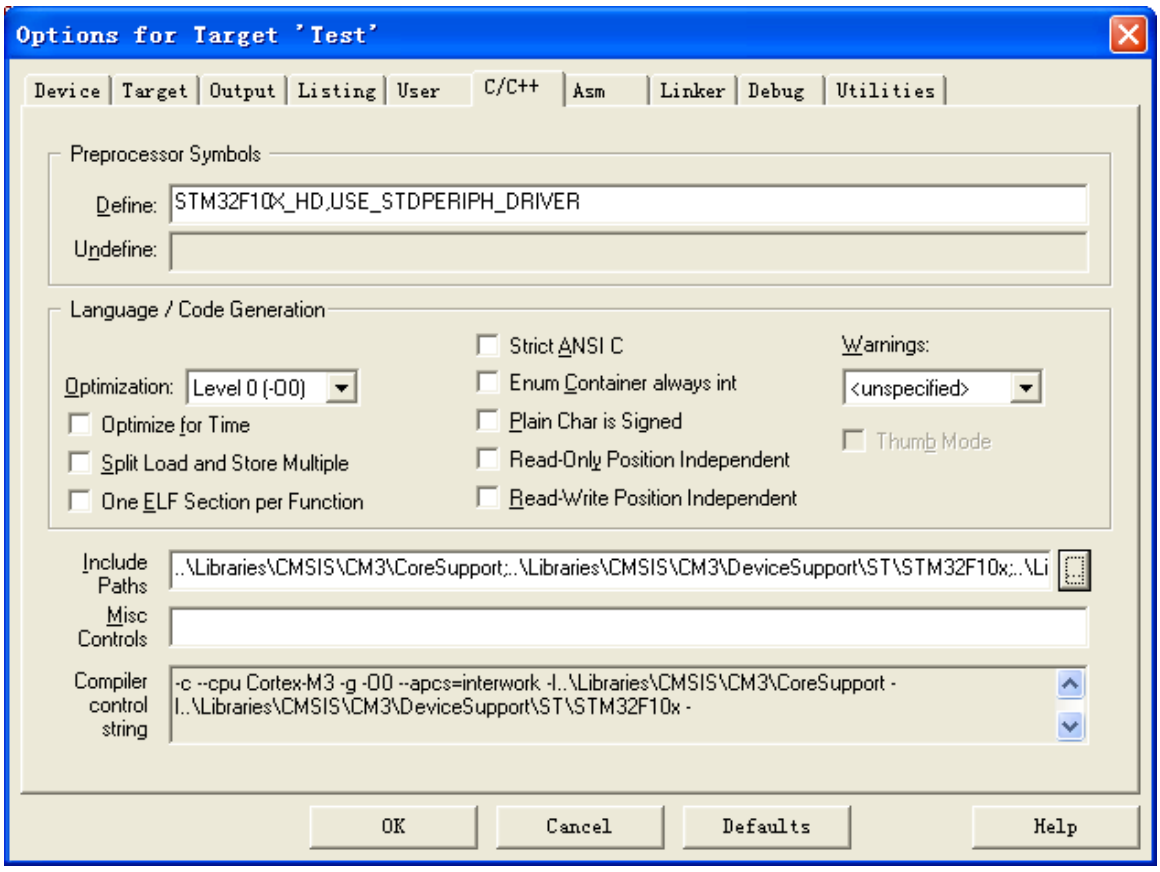
..\User

结果如下：



注意这些路径都是路径，指相对Demo\Project\ test.uvproj文件的路径。..表示上层目录。如果不指定相对路径，那么整个Demo文件夹复制到其他位置后，会导致编译失败。

25) 点击“OK”后，即完成C/C++编译器的设置，完整的结果如下：



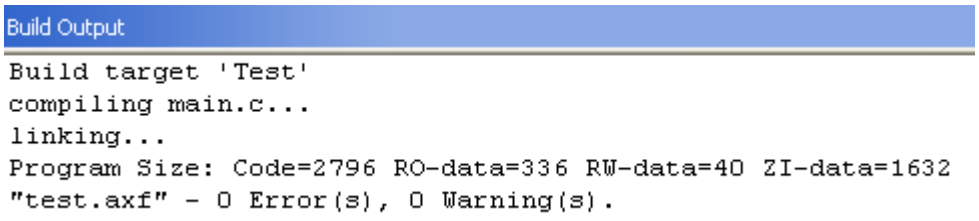
其中“Compiler contraol string”的完整内容如下：

-c --cpu Cortex-M3 -g -O0 --apcs=interwork -I..\Libraries\CMSIS\CM3\CoreSupport -I..\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x -I..\Libraries\STM32F10x_StdPeriph_Driver\inc -I..\User -I "d:\KeilV4\ARM\INC" -I "d:\KeilV4\ARM\INC\ST\STM32F10x" -DSTM32F10X_HD -DUSE_STDPERIPH_DRIVER -o "*.o" --omf_browse "*.crf" --depend "*.d"

如果你需要使用中文字符串常量，请在Misc Controls中增加一个编译控制选项：--diag_suppress=870

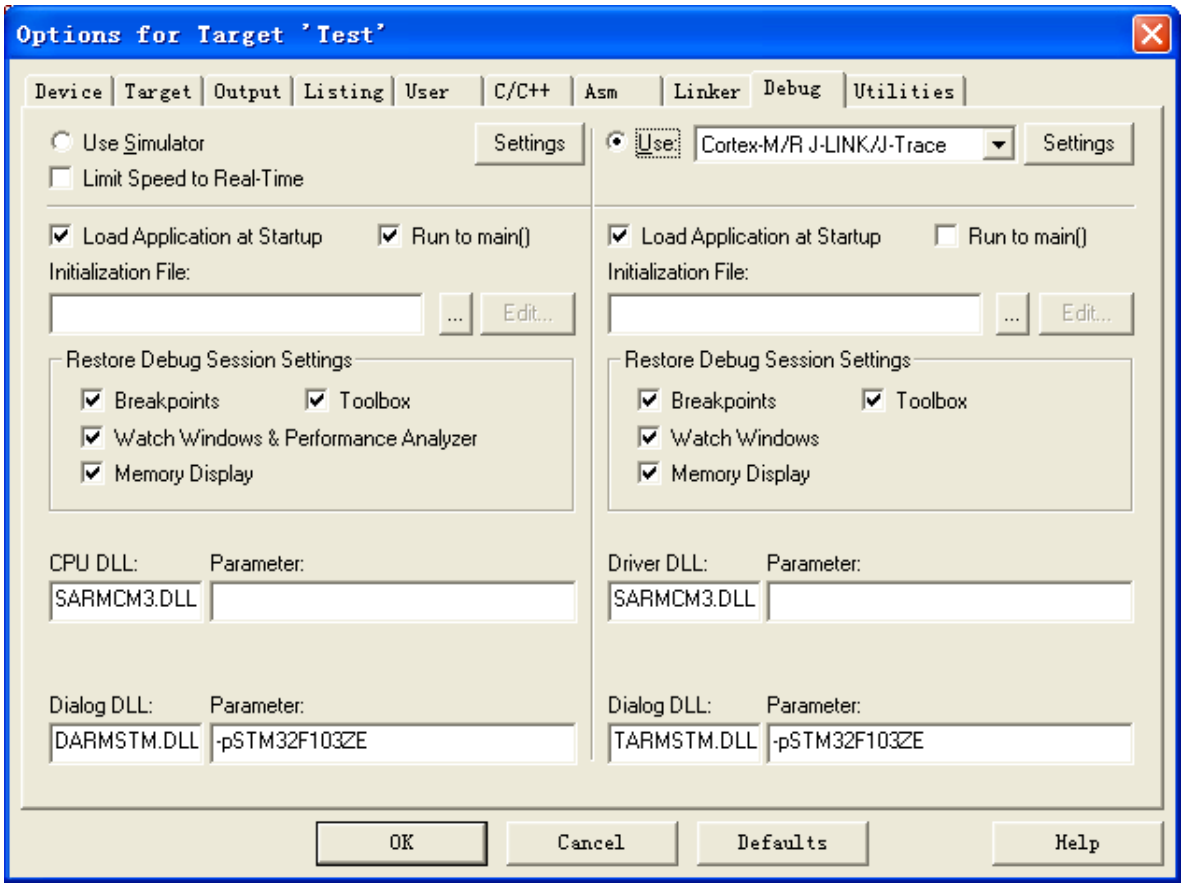
这个选项表示忽略非ASCII字符串警告。

到这一步，应该就可以编译、连接成功了。请点击  按钮测试一下。输出窗口应该显示如下：

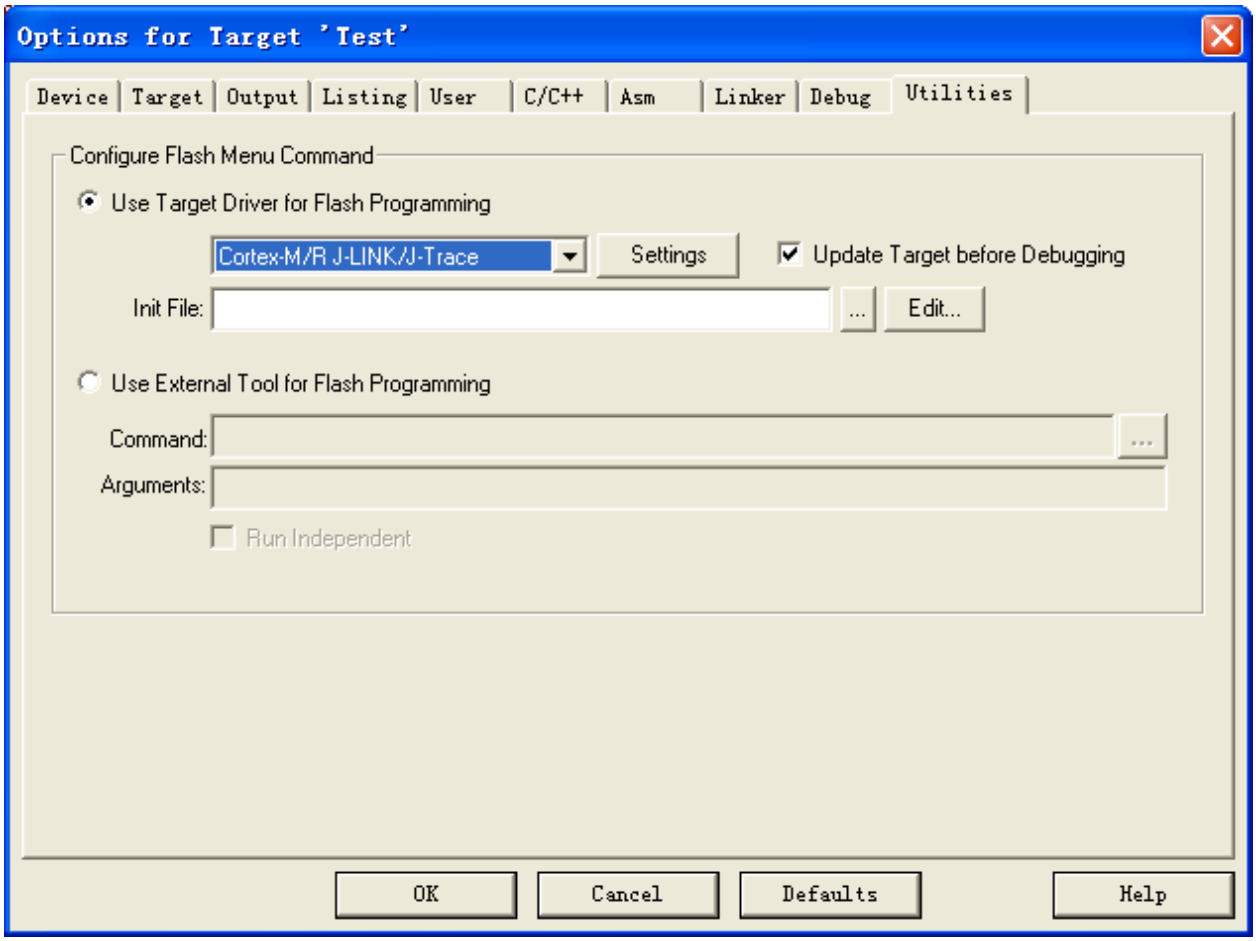


下面，我们再来设置调试器接口和选择Flash下载算法。

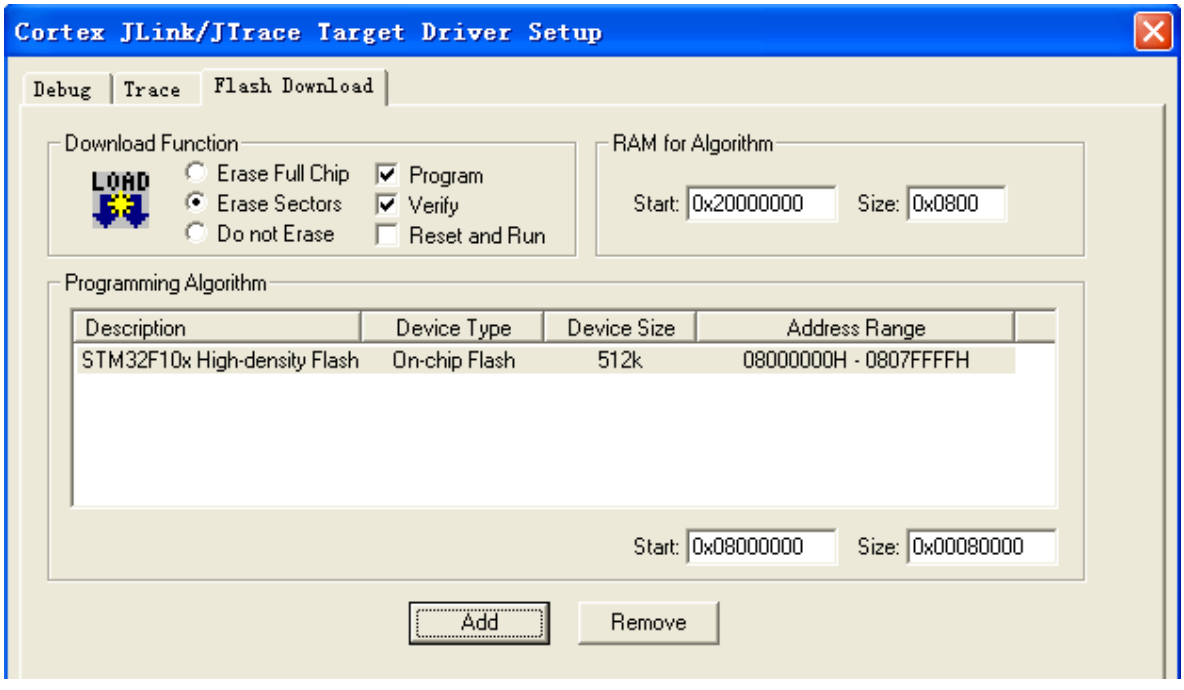
26) 切换到“Debug”页签设置调试器接口，选择“Use Cortex-M/R J-LINK/J-Trace”，界面如下：



27) 切换到“Debug”页签，选择“Use Target Driver for Flash Programming”，并在“Update Target before Debuggin”前面打钩。界面如下：



点击“Settings”按钮，出现如下“Flash Download”窗口。请再点击“Add”按钮添加编程算法，最终结果如下：



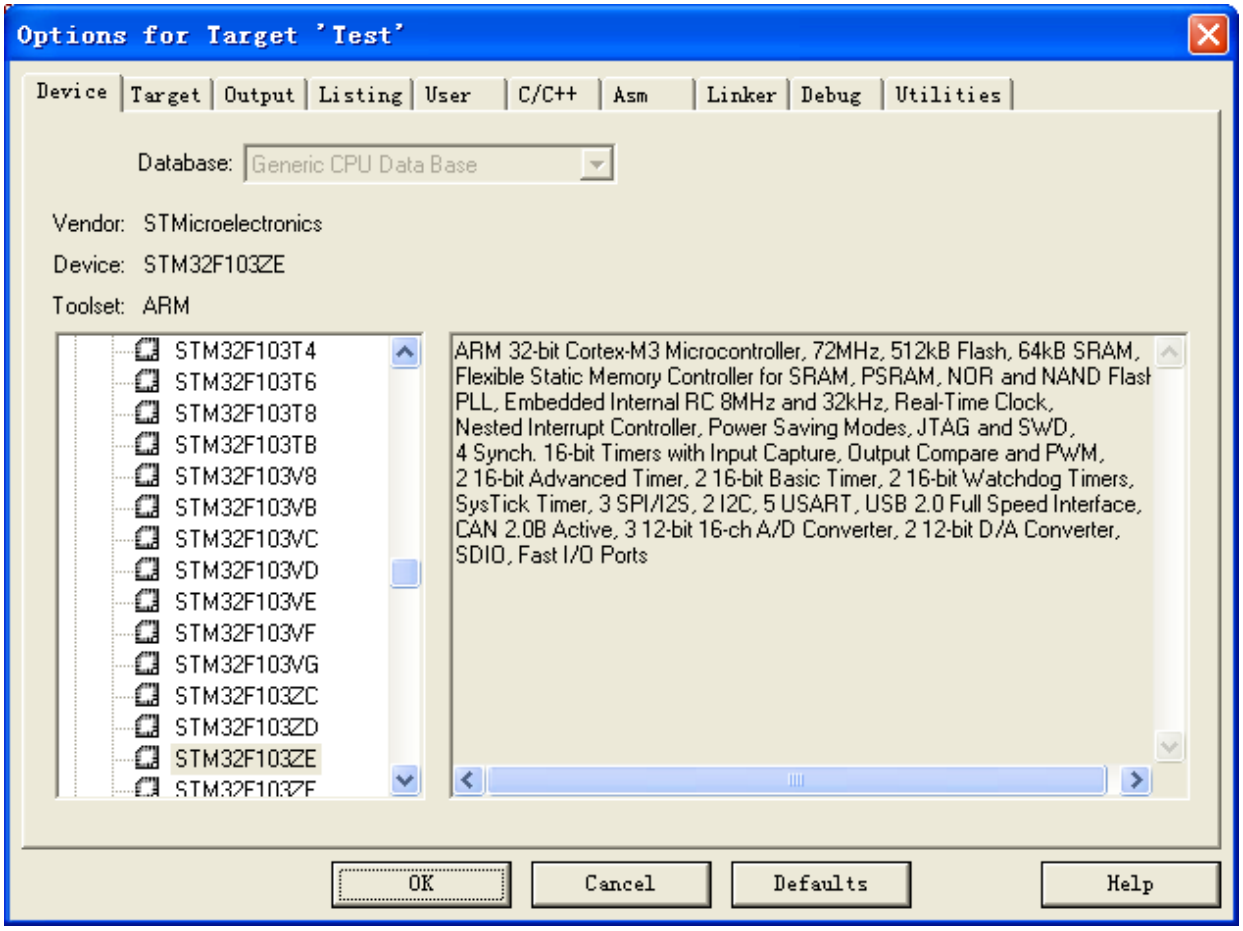
28) 至此，所有的设置完毕，其它未设置的页签保持缺省设置即可。请点击“Ok”退出设置界面，然后可以进行如下操作：

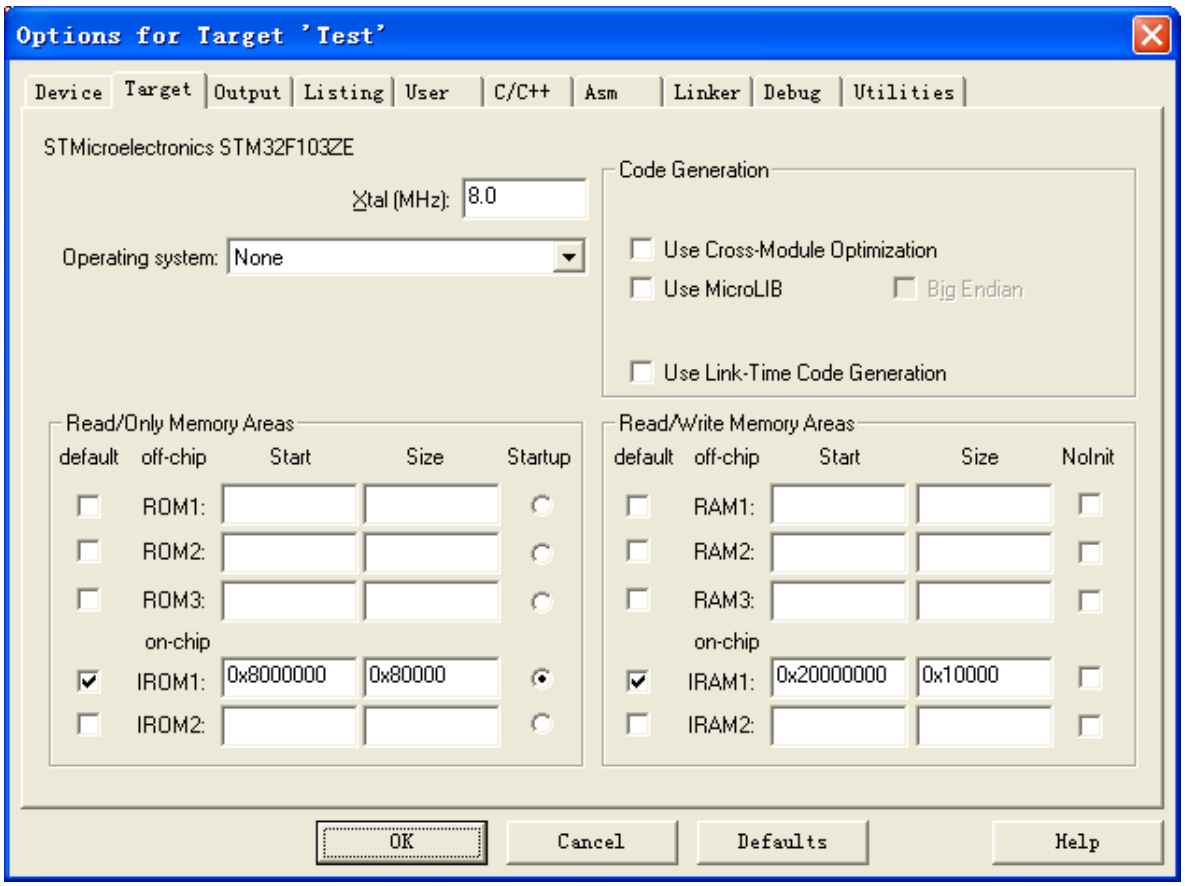
点击按钮  编译和连接整个工程的源代码，以后可以只用点击按钮  编译修改过的源文件。

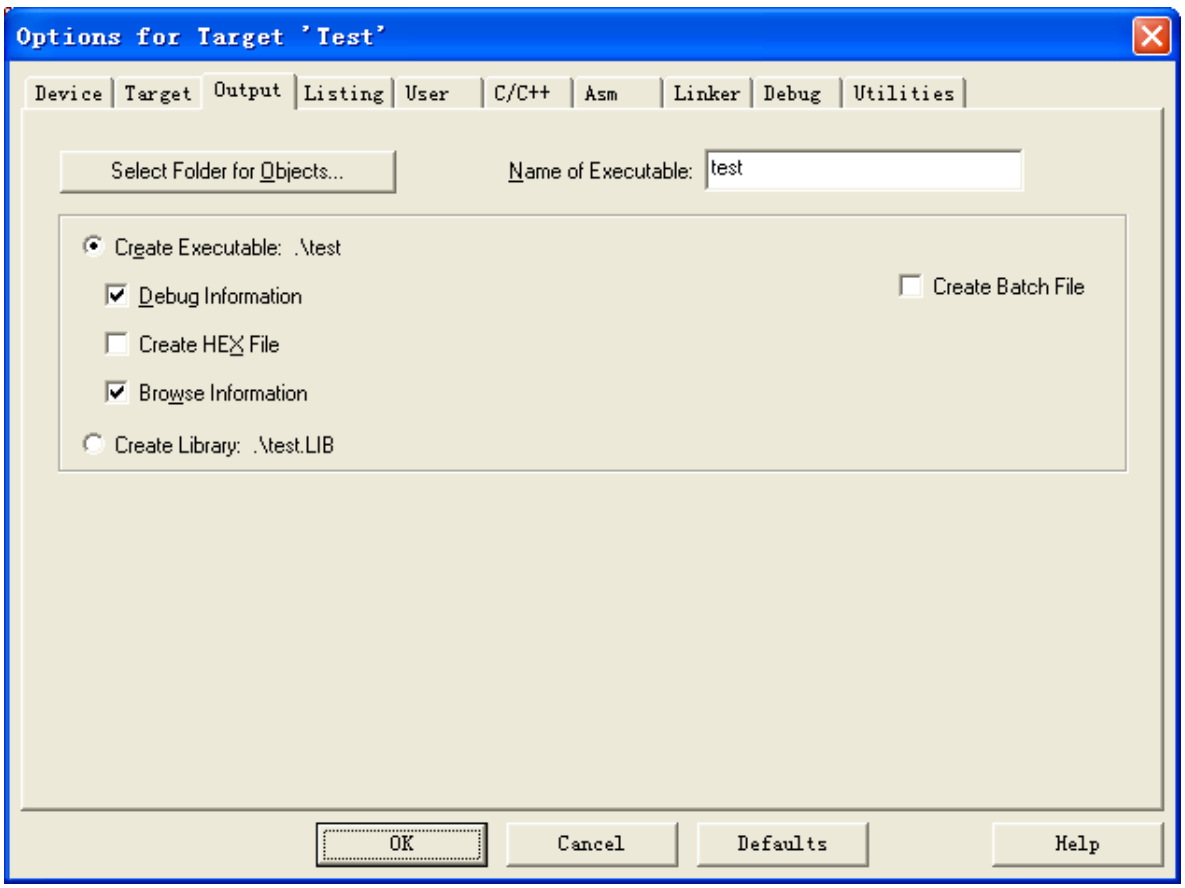
编译和连接成功后，就可以点击  仅下载程序到Flash，也点击按钮  直接进入Debug状态。

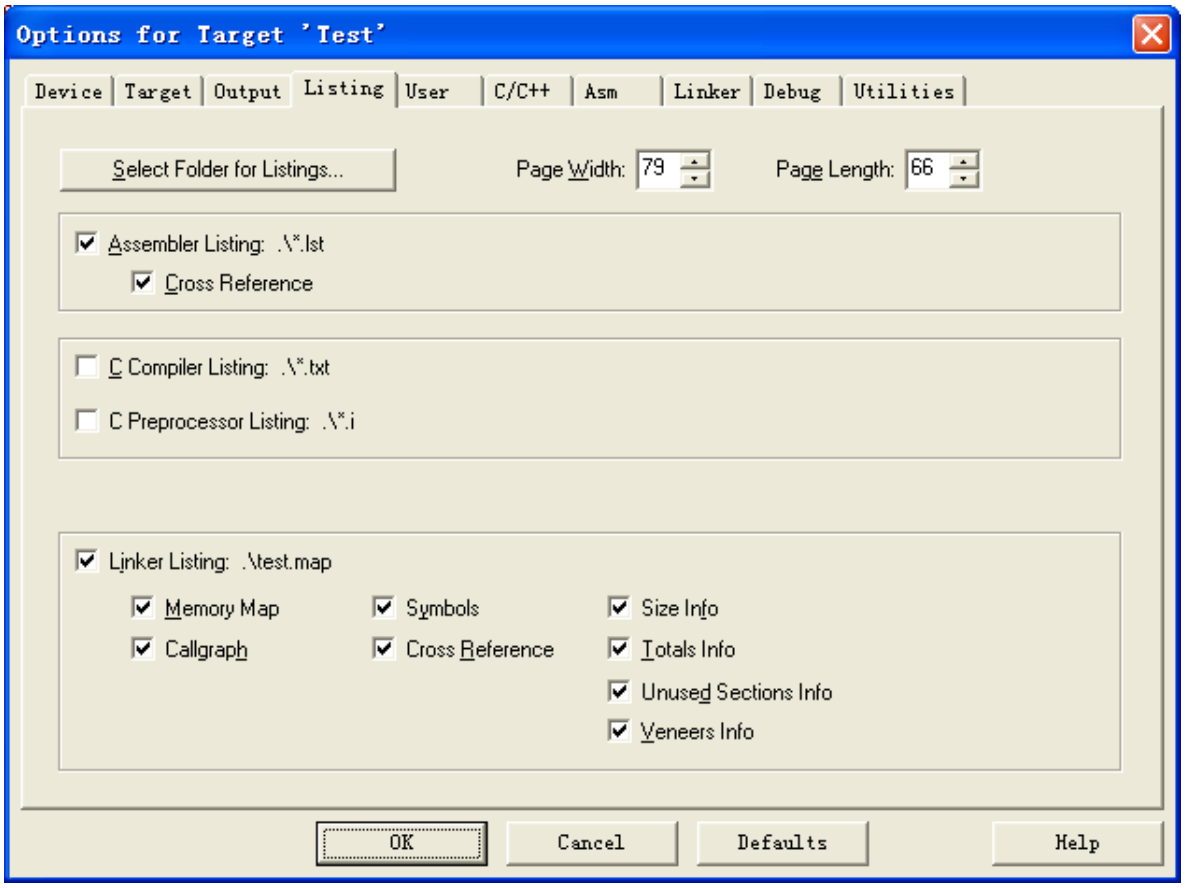
进入Debug界面后，可以进行全速、单步、跟踪、暂停等操作。

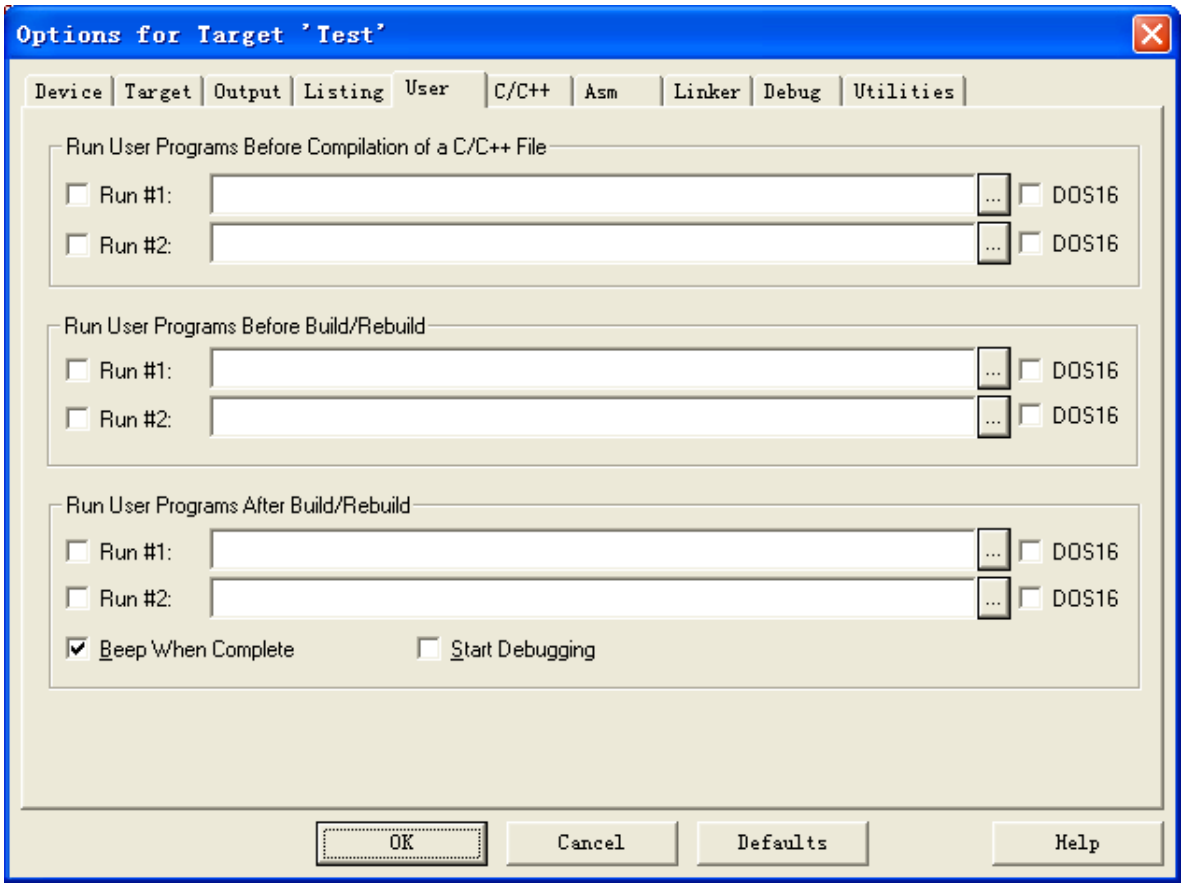
附：汇总所有的设置页签，包括其它无需改变缺省值的页签。

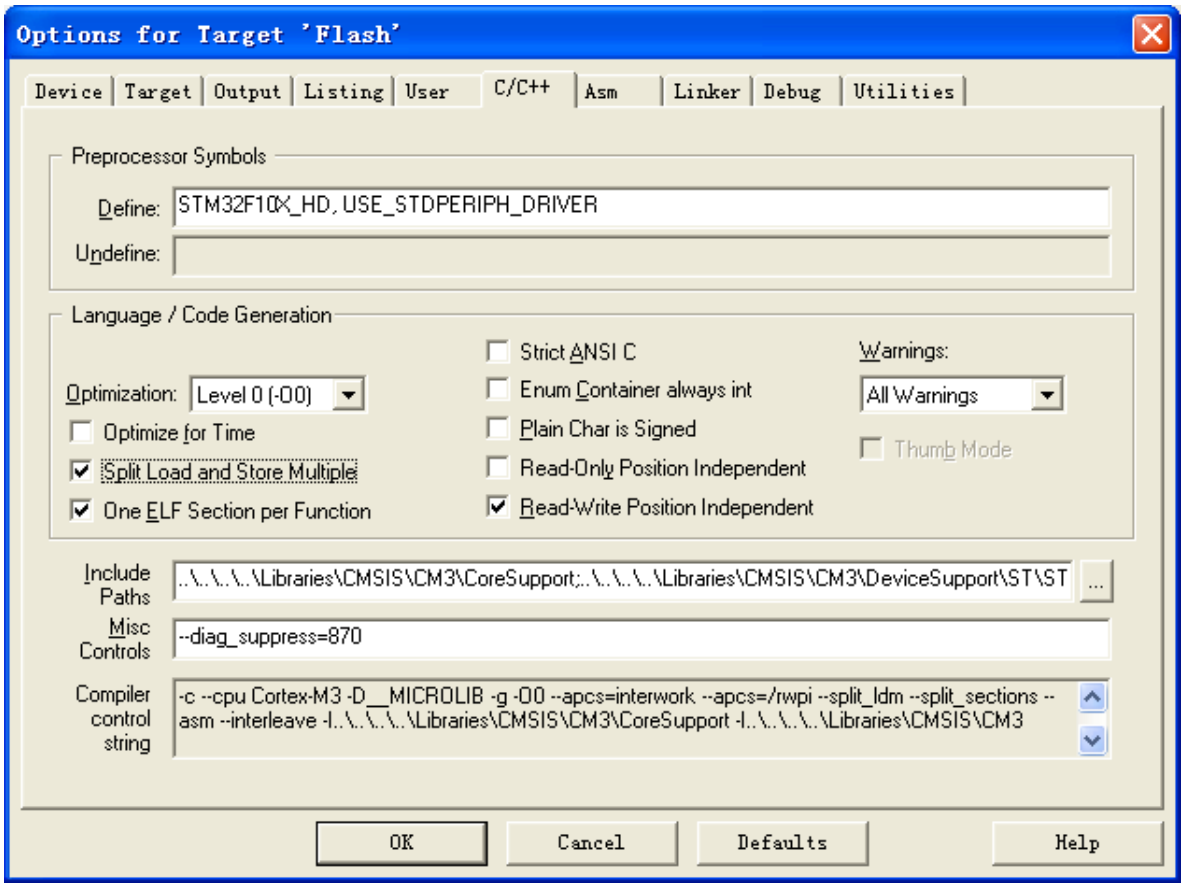


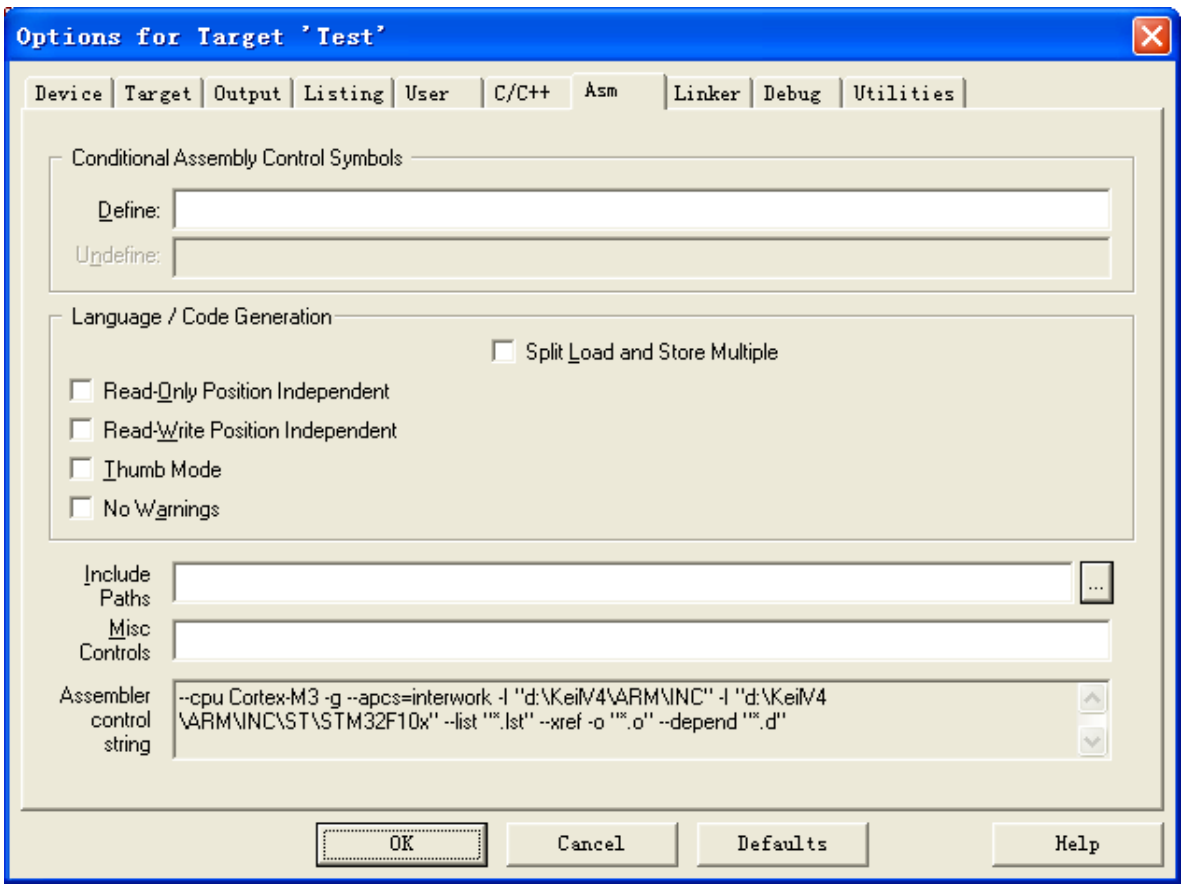

Options for Target 'Test'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
STMicroelectronics STM32F103ZE
Xtal (MHz): 8.0
Operating system: None
Code Generation
Use Cross-Module Optimization
Use MicroLIB
Big Endian
Use Link-Time Code Generation
Read/Only Memory Areas
default
off-chip
Start
Size
Startup
ROM1:
ROM2:
ROM3:
on-chip
IROM1: 0x8000000 0x80000
IROM2:
Read/Write Memory Areas
default
off-chip
Start
Size
NoInit
RAM1:
RAM2:
RAM3:
on-chip
IRAM1: 0x20000000 0x10000
IRAM2:
OK
Cancel
Defaults
Help

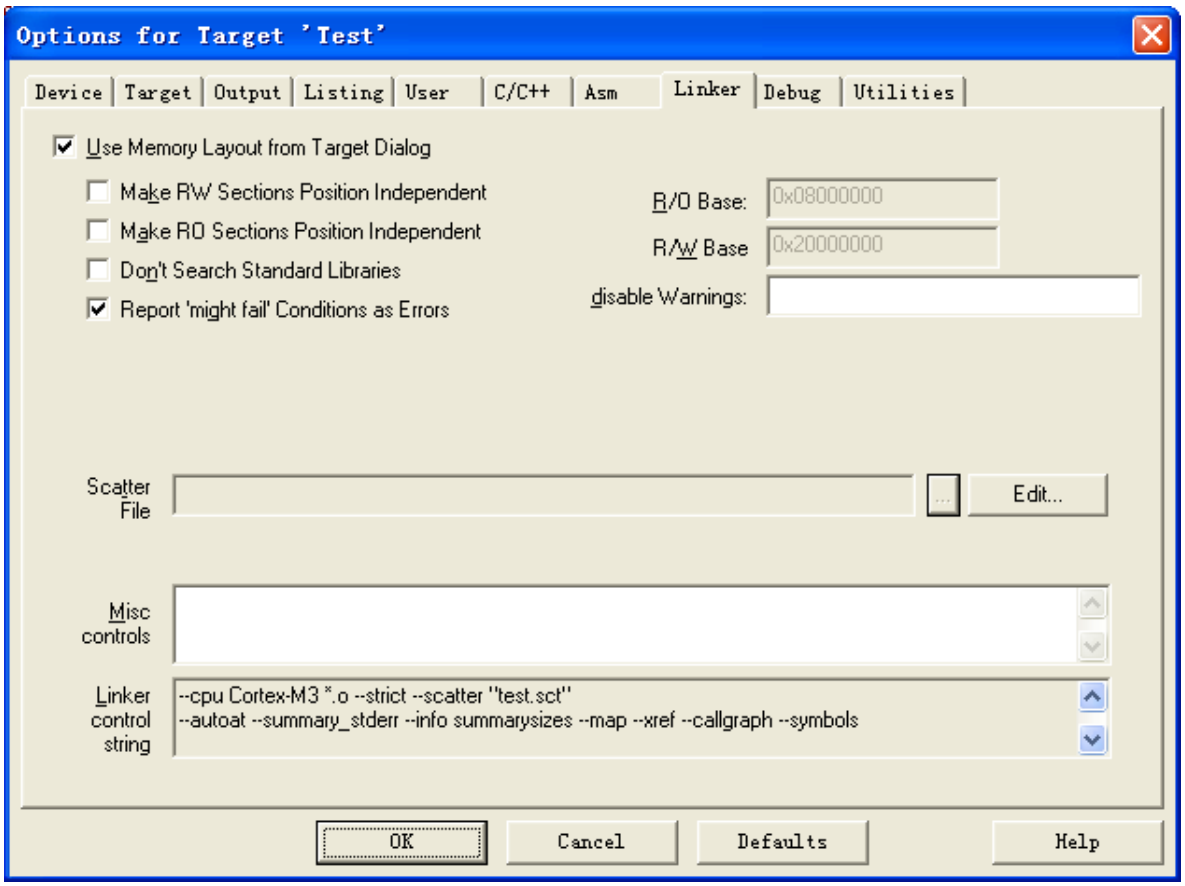

Options for Target 'Test'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
Select Folder for Objects...
Name of Executable: test
Create Executable: .\test
Debug Information
Create HEX File
Browse Information
Create Library: .\test.LIB
Create Batch File
OK
Cancel
Defaults
Help

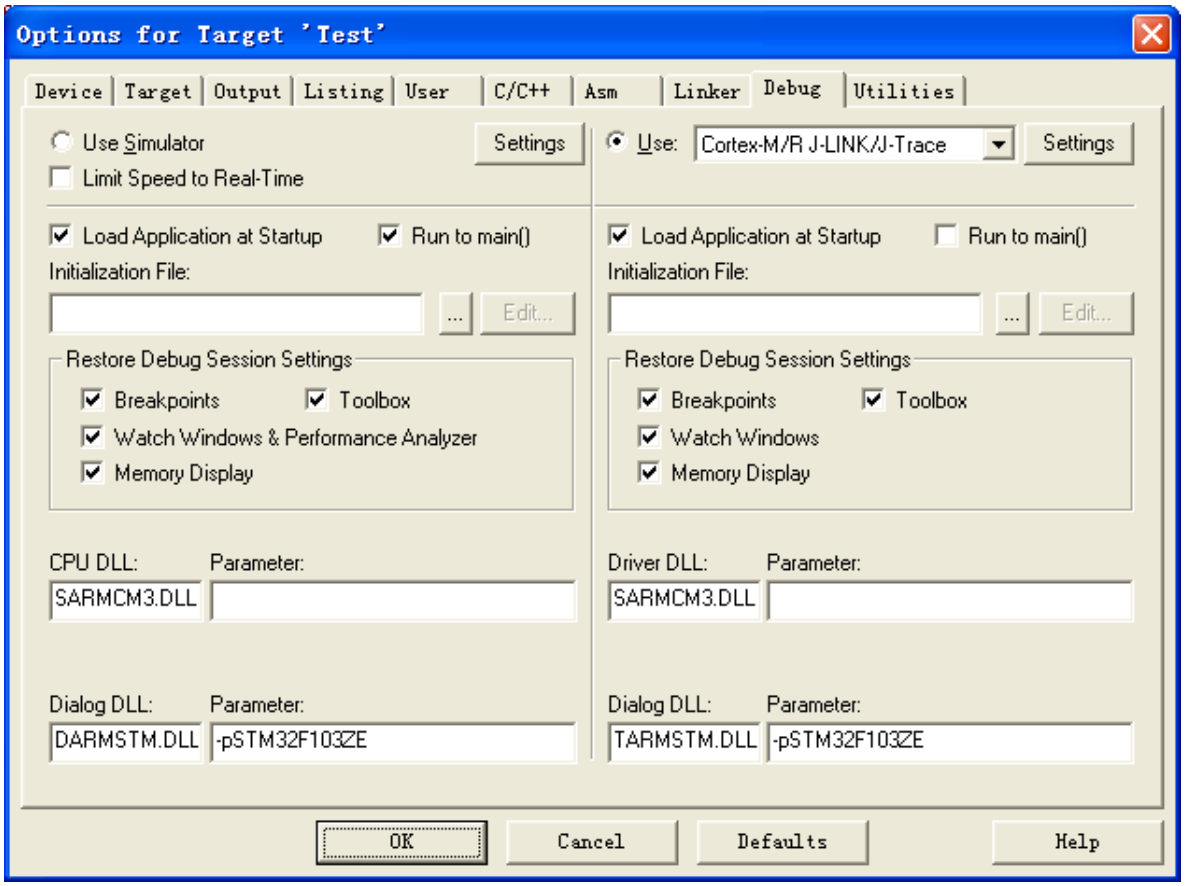

Options for Target 'Test'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
Select Folder for Listings...
Page Width: 79
Page Length: 66
Assembler Listing: .*.lst
Cross Reference
C Compiler Listing: .*.txt
C Preprocessor Listing: .*.i
Linker Listing: .\test.map
Memory Map
Symbols
Size Info
Callgraph
Cross Reference
Totals Info
Unused Sections Info
Veneers Info
OK
Cancel
Defaults
Help

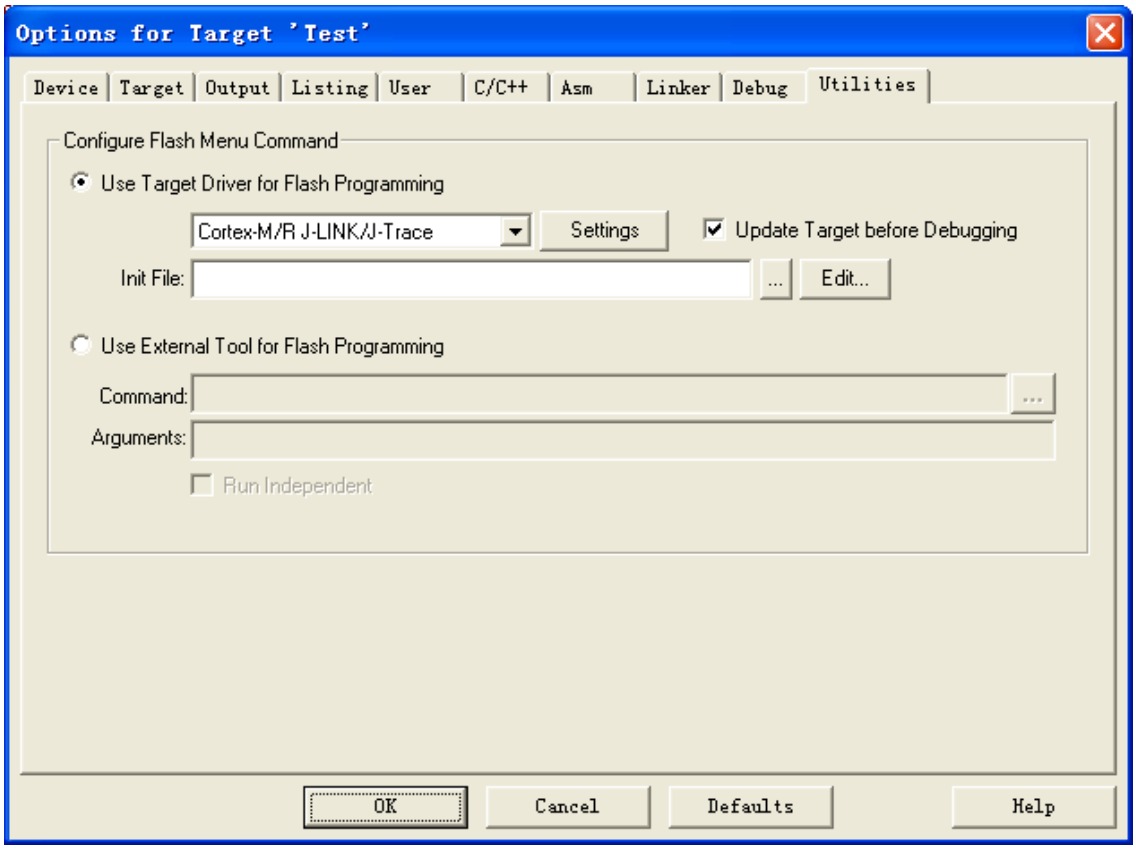

Options for Target 'Test'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
Run User Programs Before Compilation of a C/C++ File
Run #1:
Run #2:
DOS16
Run User Programs Before Build/Rebuild
Run #1:
Run #2:
DOS16
Run User Programs After Build/Rebuild
Run #1:
Run #2:
DOS16
Beep When Complete
Start Debugging
OK
Cancel
Defaults
Help


Options for Target 'Flash'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
Preprocessor Symbols
Define:
STM32F10X_HD, USE_STDPERIPH_DRIVER
Undefine:
Language / Code Generation
Optimization:
Level 0 (-O0)
Optimize for Time
Split Load and Store Multiple
One ELF Section per Function
Strict ANSI C
Enum Container always int
Plain Char is Signed
Read-Only Position Independent
Read-Write Position Independent
Warnings:
All Warnings
Thumb Mode
Include Paths
..\..\..\..\Libraries\CMSIS\CM3\CoreSupport;..\..\..\..\Libraries\CMSIS\CM3\DeviceSupport\ST\ST
Misc Controls
--diag_suppress=870
Compiler control string
-c --cpu Cortex-M3 -D__MICROLIB -g -O0 --apcs=interwork --apcs=/rwpi --split_ldm --split_sections --
asm --interleave -I..\..\..\..\Libraries\CMSIS\CM3\CoreSupport -I..\..\..\..\Libraries\CMSIS\CM3
OK
Cancel
Defaults
Help

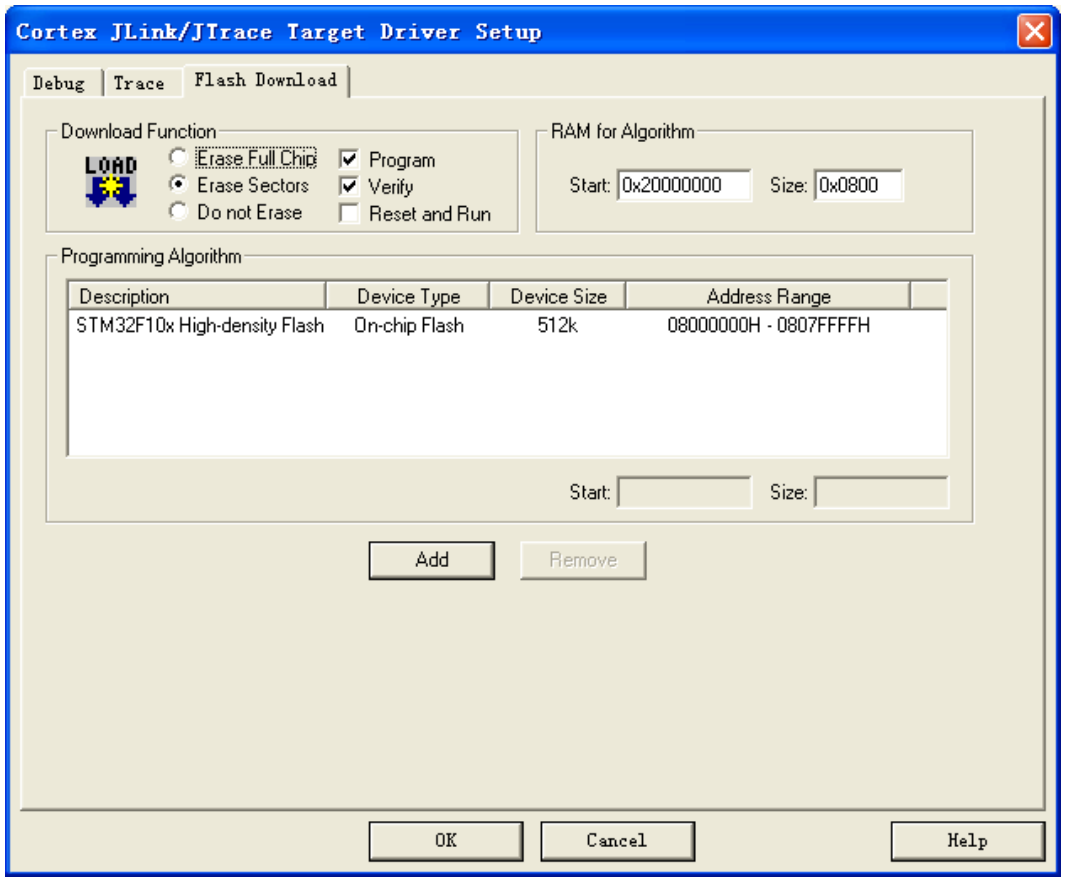

Options for Target 'Test'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
Conditional Assembly Control Symbols
Define:
Undefine:
Language / Code Generation
Split Load and Store Multiple
Read-Only Position Independent
Read-Write Position Independent
Thumb Mode
No Warnings
Include Paths
Misc Controls
Assembler control string
--cpu Cortex-M3 -g --apcs=interwork -I "d:\KeilV4\ARM\INC" -I "d:\KeilV4
\ARM\INC\ST\STM32F10x" --list "*.lst" --xref -o "*.o" --depend "*.d"
OK
Cancel
Defaults
Help


Options for Target 'Test'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
Use Memory Layout from Target Dialog
Make RW Sections Position Independent
Make RO Sections Position Independent
Don't Search Standard Libraries
Report 'might fail' Conditions as Errors
R/O Base:
0x08000000
R/W Base
0x20000000
disable Warnings:
Scatter File
Edit...
Misc controls
Linker control string
--cpu Cortex-M3 *.o --strict --scatter "test.sct"
--autoat --summary_stderr --info summarysizes --map --xref --callgraph --symbols
OK
Cancel
Defaults
Help


Options for Target 'Test'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
Use Simulator
Settings
Limit Speed to Real-Time
Use:
Cortex-M/R J-LINK/J-Trace
Settings
Load Application at Startup
Run to main()
Initialization File:
Edit...
Restore Debug Session Settings
Breakpoints
Toolbox
Watch Windows & Performance Analyzer
Watch Windows
Memory Display
CPU DLL:
Parameter:
SARMCM3.DLL
Driver DLL:
Dialog DLL:
DARMSTM.DLL
-pSTM32F103ZE
TARMSTM.DLL
OK
Cancel
Defaults
Help



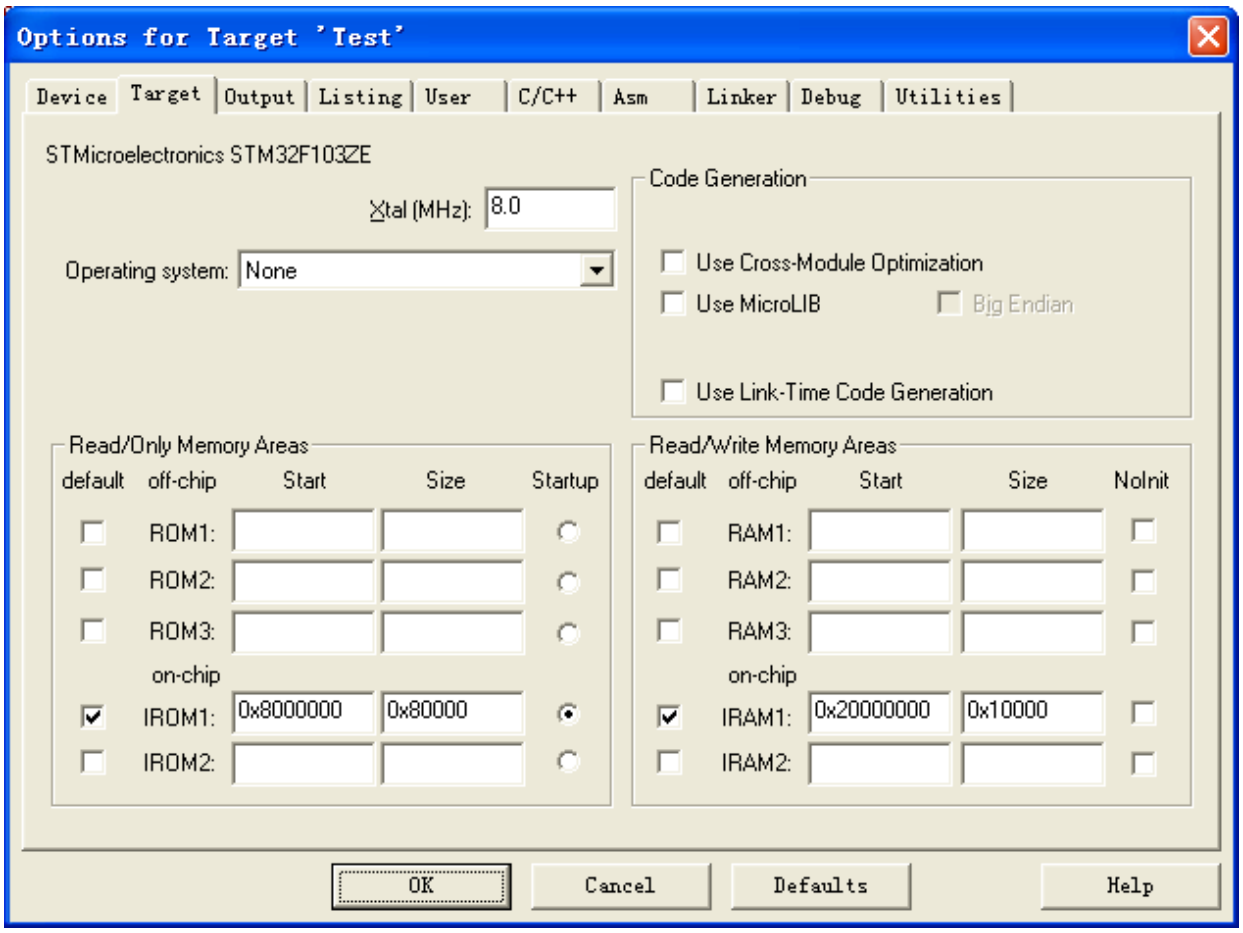
点击“Settings”，界面如下：



3.2.5. 程序空间在CPU内部Flash，变量空间在CPU内部RAM

将程序空间定位在CPU内部Flash，变量空间定位在CPU内部RAM。这是最通用的设置，一般最终发布的软件就是按照这种设置。

关键设置界面为：



IROM1 设置为 0x80000000，IRAM1设置为0x20000000。.

该设置的程序可以用于下载到Flash，脱离仿真器运行。也可以直接使用仿真器器进行调试跟踪。

3.2.6. 程序空间和变量空间都在CPU内部RAM

该设置的程序只能使用仿真器进行调试。

优点：

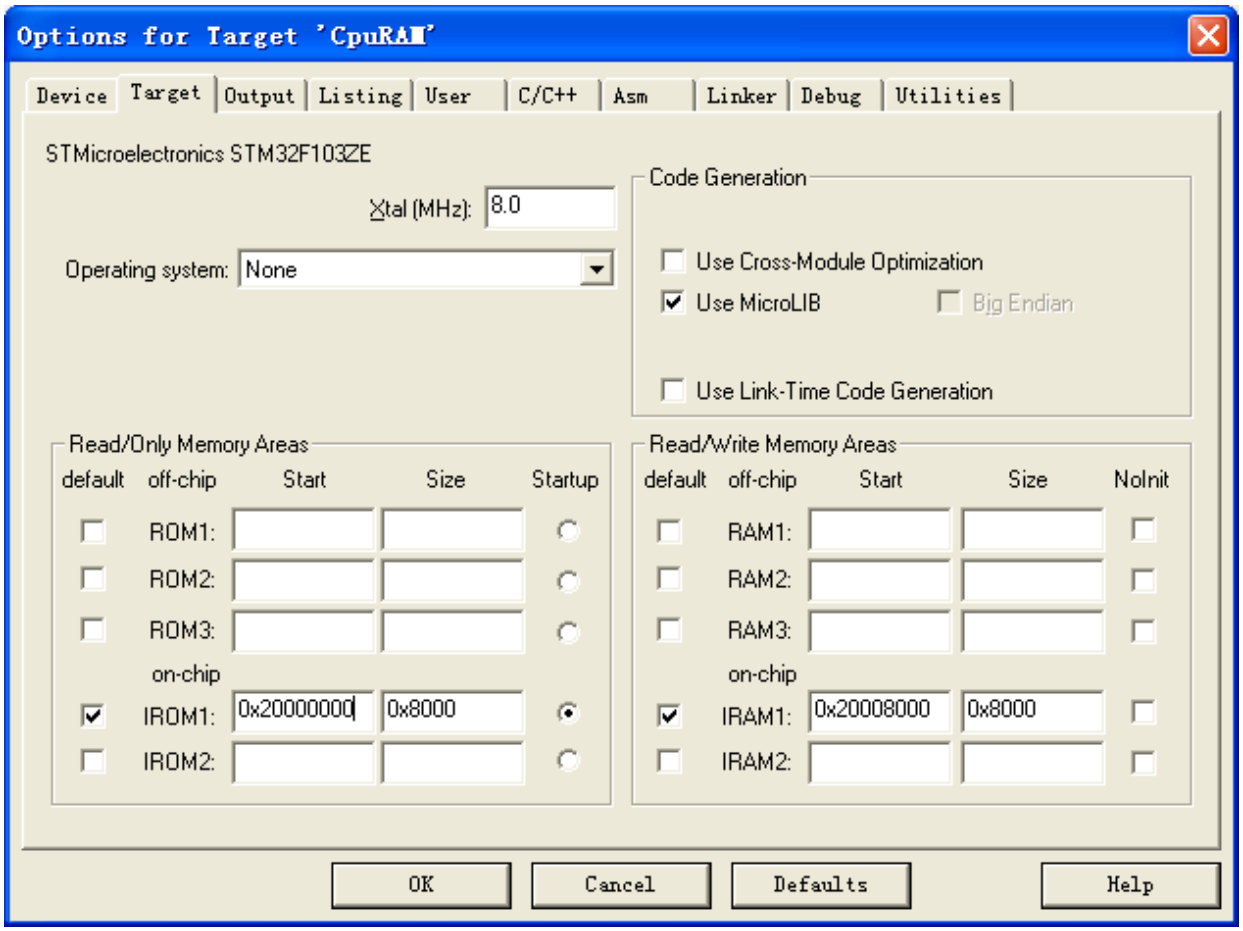
- 下载速度快；
- 不改写CPU内部Flash已有的程序；
- 无需拨动启动模式选择开关（即拨打CPU 内部Flash启动也可以下载到RAM进行调试；
- 程序执行速度和在CPU内部Flash一样快。

缺点：

- 开发板掉电会丢失程序；
- 暂时无法使用调试界面的复位按钮进行复位。
- 程序空间最大32K字节，变量空间最大限制在32K字节。（用户也可以自行调整）

重要说明： 编译、连接完毕后，不要使用  按钮下载程序。因为这个按钮只针对下载程序到CPU内部Flash有效。请直接点击  按钮启动调试即可。IDE会自动将程序装入CPU内部RAM。

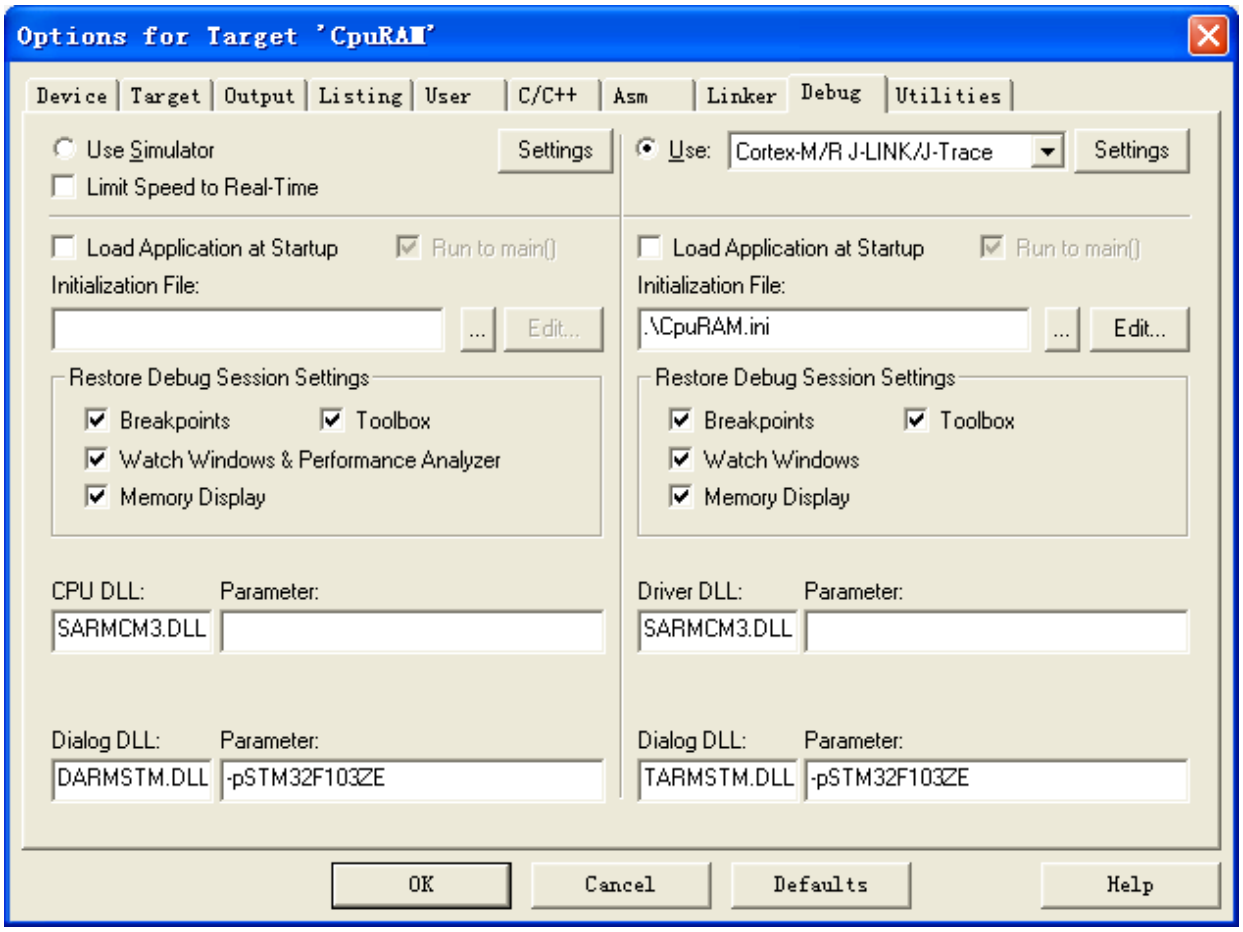
首先设置Target定位地址，设置界面如下：



IROM1 = 0x20000000，Size = 0x8000，这是CPU内部RAM区的前32K字节空间；

IRAM1 = 0x20008000，Size = 0x8000，这是CPU内部RAM区的后32K字节空间。

然后设置Debug调试接口，界面如下：



a) 取消“Load Application at Startup”前面的钩。在CpuRAM.ini初始化脚本中自动装入程序。

b) 在“Initialization Files：“编辑框指定”.\CpuRAM.ini“。 .\ 表示工程文件所在的当前目录。

c) 请将CpuRAM.ini文件和工程文件放在同一个文件夹下。CpuRAM.ini文件是一个文本格式的初始化脚本文件。当启动调试时，IDE会执行这个脚本中的命令。

初始化脚本文件说明:

a) 新建一个CpuRAM.ini的空文件，然后使用记事本将如下内容复制到这文件保存即可。

b) CpuRAM.ini的内容如下：

```
/*********************************************************************************
    Copyright (C), 2010 安富莱电子 www.armfly.com

  【本例程在安富莱 STM32F103ZE-EK 开发板上调试通过                    】
  【QQ: 1295744630, 旺旺：armfly, Email: armfly@qq.com                 】

    文件名：  CpuRAM.ini

    这是 CPU 内部 RAM 调试脚本。开始 Load 程序时，由 IDE 控制仿真器执行这段脚本程序。

    本脚本完成的功能是
```

```
    (1) 装载目标程序到 CPU 内部 RAM
    (2) 设置堆栈指针 SP
    (3) 修改 PC 指针

    脚本的语法：
    参加 MDK 的 HELP, 搜索关键字 “uv3 Library Routines” 可以看到 uv3 支持的脚本命令
*/

FUNC void Setup (void) {
  SP = _RDWORD(0x20000000) + 4;           // 设置堆栈指针
  PC = _RDWORD(0x20000004);           // 设置 PC 指针
}

LOAD obj\output.axf INCREMENTAL    // 先装载代码到 CPU 内部 RAM
Setup();                                // 再调用 Setup 函数修改堆栈和 PC 指针
g, main                                     // 运行到 main()函数
```

IDE控制调试器将程序装入CPU内部RAM后，0x20000000单元存储了程序的堆栈初值（堆栈区域的最大值+4，向下增长）。0x20000004单元存储了复位向量地址。

3.2.7. 程序空间和变量空间都在外部SRAM

该设置的程序只能使用仿真器进行调试。

优点：

- 下载速度快；
- 不改写CPU内部Flash已有的程序；
- 无需拨动启动模式选择开关（即拨打CPU 内部Flash启动也可以下载到外部RAM进行调试。

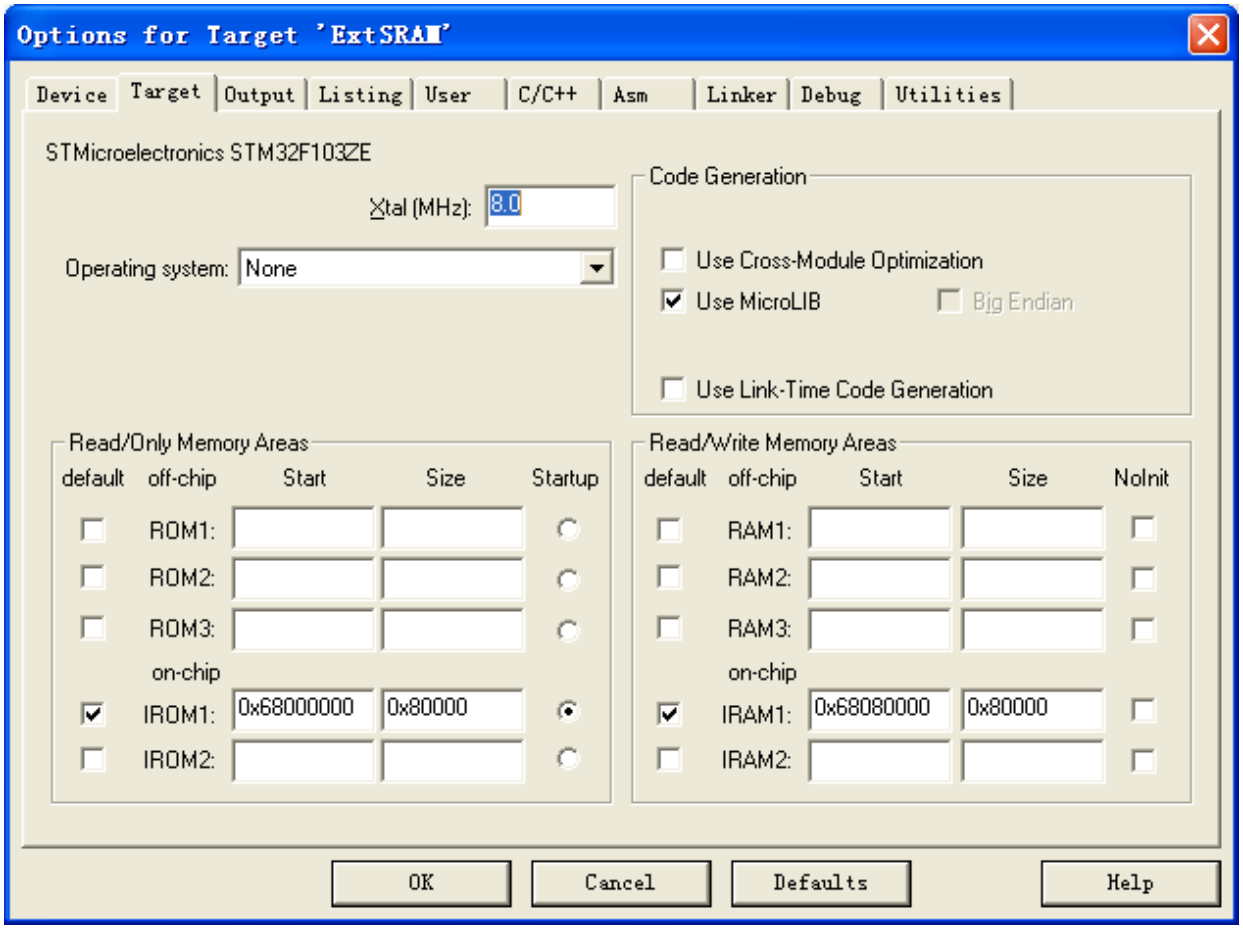
缺点：

- 开发板掉电会丢失程序；
- 暂时无法使用调试界面的复位按钮进行复位。
- 程序空间最大512K字节，变量空间最大512K字节（用户也可以自行调整）；
- 程序执行速度比CPU内部Flash慢很多。

重要说明： 编译、连接完毕后，不要使用  按钮下载程序。因为这个按钮只针对下载程序到CPU内

部Flash有效。请直接点击 按钮启动调试即可。IDE会自动将程序装入CPU外部RAM。

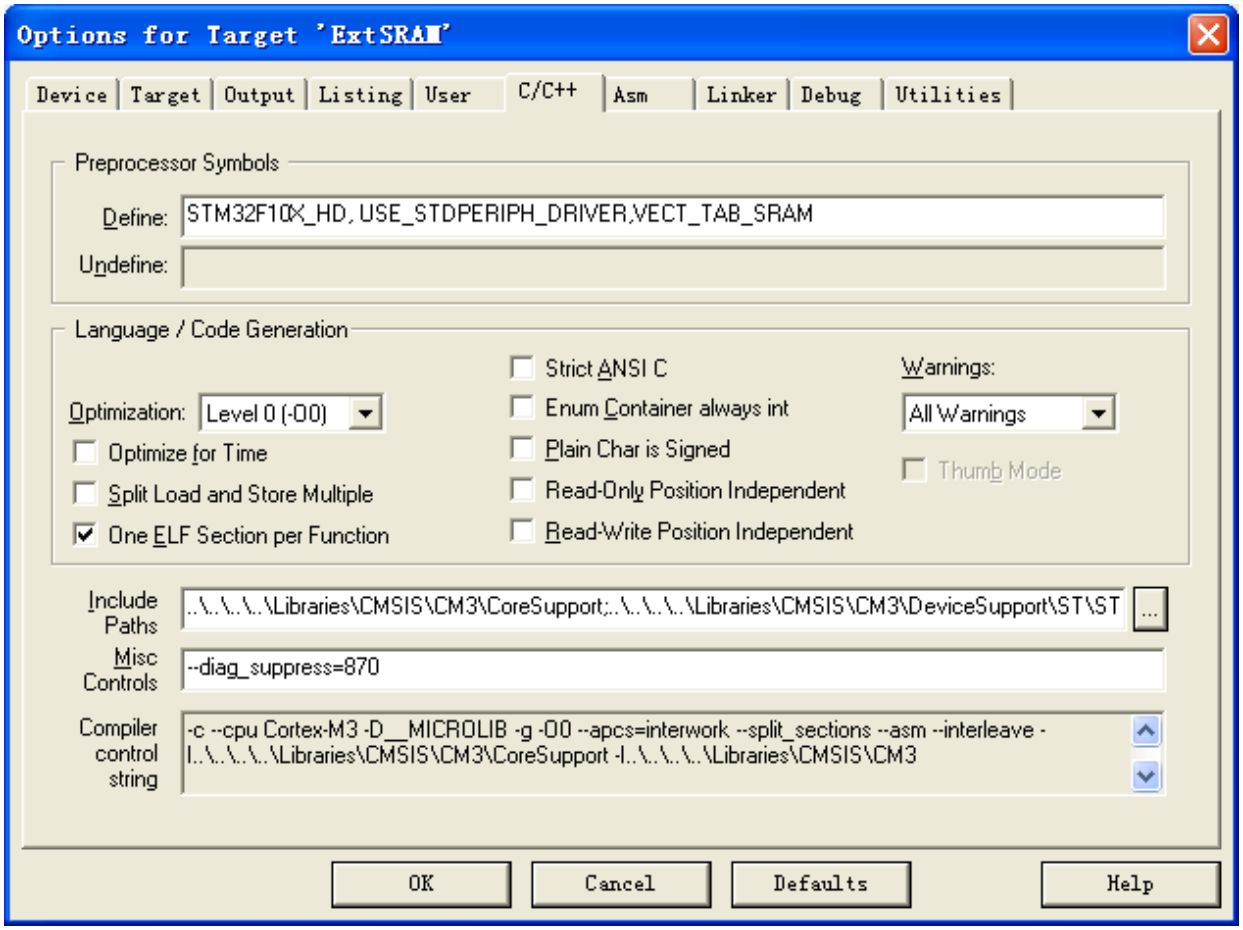
首先设置 Target 定位地址，设置界面如下：



IROM1 = 0x68000000，Size = 0x80000，这是CPU外部RAM区的前512K字节空间；

IRAM1 = 0x68080000，Size = 0x80000，这是CPU外部RAM区的后512K字节空间。

然后设置C/C++编译器编译选项，界面如下：



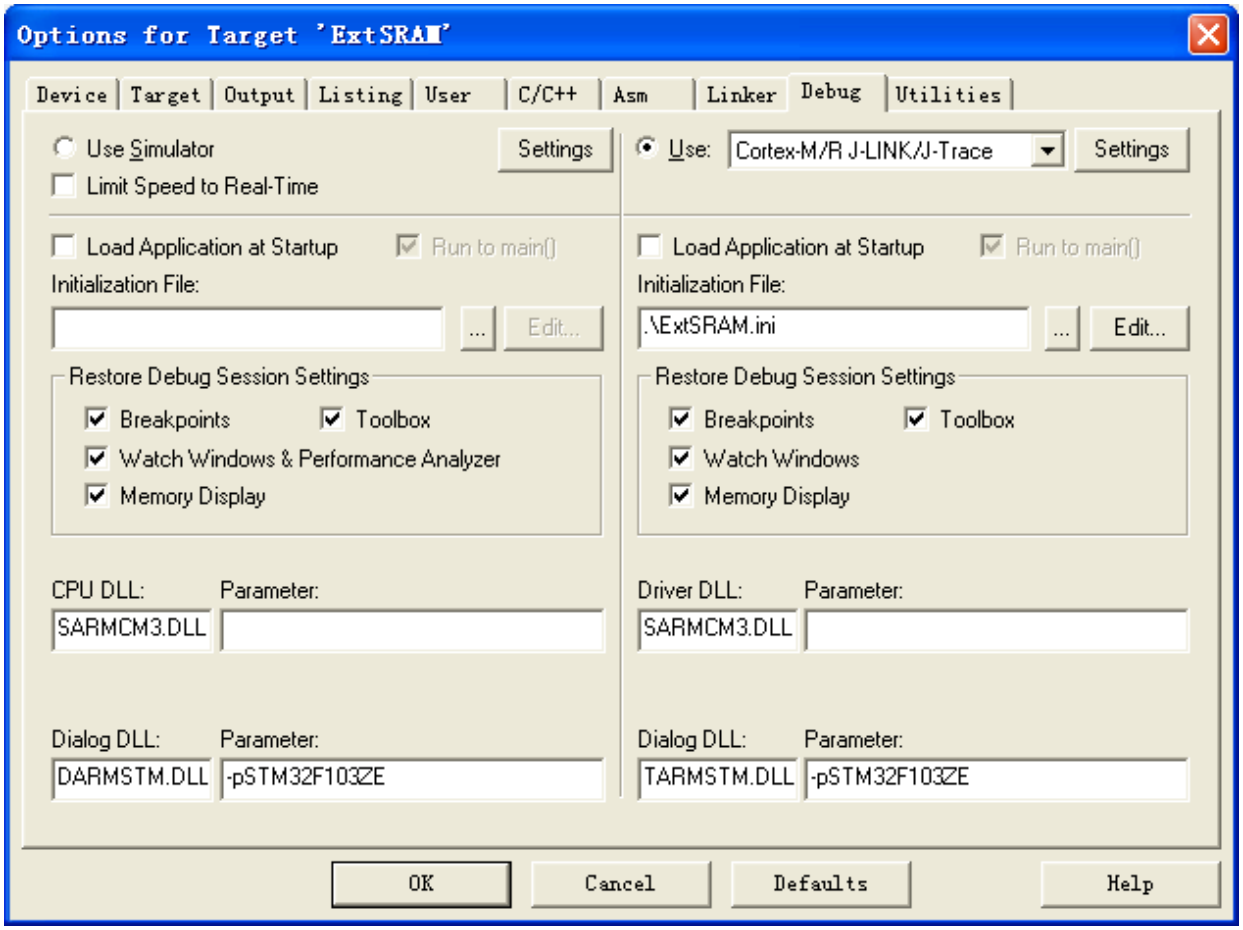
由于STM32这款CPU只能将中断向量表定位在CPU内部Flash或者CPU内部RAM，不支持将中断向量表定位在CPU外部存储器（外部SRAM或外部NOR Flash）。因此，我们需要通过调试脚本将外部RAM的中断向量表复制到CPU内部RAM。

“Define”编辑框需要添加宏“VECT_TAB_SRAM “，这个宏表示中断向量表定位在CPU内部RAM。

在ST固件库提供的system_stm32f10x.c 文件中，void SystemInit (void) 函数会改写中断向量表地址寄存器SCB->VTOR。函数中的关键代码如下：

```
#ifdef VECT_TAB_SRAM
  SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal SRAM. */
#else
  SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal FLASH. */
#endif
```

最后设置“Debug“选项。



a) 取消“Load Application at Startup”前面的钩。在CpuRAM.ini初始化脚本中自动装入程序。

b) 在“Initialization Files：“编辑框指定”.\ExtSRAM.ini“。 .\ 表示工程文件所在的当前目录。

c) 请将ExtSRAM.ini文件和工程文件放在同一个文件夹下。ExtSRAM.ini文件是一个文本格式的初始化脚本文件。当启动调试时，IDE会执行这个脚本中的命令。

初始化脚本文件说明

a) 新建一个ExtSRAM.ini的空文件，然后使用记事本将如下内容复制到这文件保存即可。

b) ExtSRAM.ini的内容如下：

```
/*********************************************************************************
    Copyright (C), 2010 安富莱电子 www.armfly.com

  【本例程在安富莱 STM32F103ZE-EK 开发板上调试通过                 】
  【QQ: 1295744630, 旺旺：armfly, Email: armfly@qq.com          】

    文件名：  ExtSRAM.ini

    这是外部 SRAM 调试脚本。开始 Load 程序时，由 IDE 控制仿真器执行这段脚本程序。

    本脚本完成的功能是
```

```
    (1) 配置 CPU 的 FSMC 总线，使 CPU 能够访问外部 SRAM，已便于后面装入程序
    (2) 复制外部 SRAM 的中断向量表（0x130 字节）到 CPU 内部 RAM
    (3) 设置堆栈指针 SP
    (4) 修改 PC 指针

    注意：工程选项 IRAM1 的起始地址必须是 0x20000200 开始

    脚本的语法：
    参见 MDK 的 HELP，搜索关键字 “uv3 Library Routines” 可以看到 uv3 支持的脚本命令
*/

FUNC void Setup (void) {
  SP = _RDWORD(0x68000000);           // 设置堆栈指针
  PC = _RDWORD(0x68000004);           // 设置 PC 指针（程序计数器）
  _WDWORD(0xE000ED08, 0x20000000);    // 设置中断向量表地址寄存器 = 0x20000000
}

// 初始化 FSMC，用于外部 SRAM
FUNC void InitSRAM (void) {
  //InitRCCC
  _WDWORD(0x40021000, 0x00005083);
  _WDWORD(0x40021004, 0x00000000);
  _WDWORD(0x40021004, 0x00000000);
  _WDWORD(0x40021000, 0x00005083);
  _WDWORD(0x40021004, 0x00000000);
  _WDWORD(0x40021008, 0x009F0000);

  _WDWORD(0x40021000, 0x00015083);
  _sleep_ (100);                      // Wait for PLL lock

  _WDWORD(0x40022000, 0x00000030);
  _WDWORD(0x40022000, 0x00000030);
  _WDWORD(0x40022000, 0x00000032);
  _WDWORD(0x40021004, 0x00000000);
  _WDWORD(0x40021004, 0x00000000);
  _WDWORD(0x40021004, 0x00000400);
  _WDWORD(0x40021004, 0x00000400);
  _WDWORD(0x40021004, 0x001D0400);
  _WDWORD(0x40021000, 0x01035083);
  _sleep_ (100);
  _WDWORD(0x40021004, 0x001D0400);
  _WDWORD(0x40021004, 0x001D0402);
  _sleep_ (100);
```

```
  _WDWORD(0x40021014, 0x00000114);  /* Enable AHBPeriphClock */
  _WDWORD(0x40021018, 0x000001E0);  /* Enable APB2PeriphClock */

  /* GPIO Configuration for FSMC */
  _WDWORD(0x40011400, 0xB8BB44BB);
  _WDWORD(0x40011404, 0xBBBBBBBB);
  _WDWORD(0x40011800, 0xBBBBB4BB);   /* NBL0, NBL1 & adress configuration */
  _WDWORD(0x40011804, 0xBBBBBBBB);
  _WDWORD(0x40011C00, 0x33BBBBBB);
  _WDWORD(0x40011C04, 0xBBBB3333);
  _WDWORD(0x40012000, 0x48BBBBBB);
  _WDWORD(0x40012004, 0x444B4BB4);   /* NE3 configuration */

  _WDWORD(0xA0000010, 0x00001010);   /* FSMC Configuration */
  _WDWORD(0xA0000014, 0x00000200);   /* FSMC_DataSetupTime = 2; */
  _WDWORD(0xA0000010, 0x00001011);   /* Enable FSMC Bank1_SRAM Bank */

  _sleep_ (200);
}

/*
     复制中断向量表
     中断向量表地址必须是 512 字节的整数倍。
     中断向量表实际大小 ： 0x00000130
*/
FUNC void CopyVectTable(void) {
     _WDWORD(0x20000000, _RDWORD(0x68000000));
     _WDWORD(0x20000004, _RDWORD(0x68000004));
     _WDWORD(0x20000008, _RDWORD(0x68000008));
     _WDWORD(0x2000000C, _RDWORD(0x6800000C));

     _WDWORD(0x20000010, _RDWORD(0x68000010));
     _WDWORD(0x20000014, _RDWORD(0x68000014));
     _WDWORD(0x20000018, _RDWORD(0x68000018));
     _WDWORD(0x2000001C, _RDWORD(0x6800001C));

     _WDWORD(0x20000020, _RDWORD(0x68000020));
     _WDWORD(0x20000024, _RDWORD(0x68000024));
     _WDWORD(0x20000028, _RDWORD(0x68000028));
     _WDWORD(0x2000002C, _RDWORD(0x6800002C));

     _WDWORD(0x20000030, _RDWORD(0x68000030));
     _WDWORD(0x20000034, _RDWORD(0x68000034));
     _WDWORD(0x20000038, _RDWORD(0x68000038));
```

```
_WDWORD(0x2000003C, _RDWORD(0x6800003C));

_WDWORD(0x20000040, _RDWORD(0x68000040));
_WDWORD(0x20000044, _RDWORD(0x68000044));
_WDWORD(0x20000048, _RDWORD(0x68000048));
_WDWORD(0x2000004C, _RDWORD(0x6800004C));

_WDWORD(0x20000050, _RDWORD(0x68000050));
_WDWORD(0x20000054, _RDWORD(0x68000054));
_WDWORD(0x20000058, _RDWORD(0x68000058));
_WDWORD(0x2000005C, _RDWORD(0x6800005C));

_WDWORD(0x20000060, _RDWORD(0x68000060));
_WDWORD(0x20000064, _RDWORD(0x68000064));
_WDWORD(0x20000068, _RDWORD(0x68000068));
_WDWORD(0x2000006C, _RDWORD(0x6800006C));

_WDWORD(0x20000070, _RDWORD(0x68000070));
_WDWORD(0x20000074, _RDWORD(0x68000074));
_WDWORD(0x20000078, _RDWORD(0x68000078));
_WDWORD(0x2000007C, _RDWORD(0x6800007C));

_WDWORD(0x20000080, _RDWORD(0x68000080));
_WDWORD(0x20000084, _RDWORD(0x68000084));
_WDWORD(0x20000088, _RDWORD(0x68000088));
_WDWORD(0x2000008C, _RDWORD(0x6800008C));

_WDWORD(0x20000090, _RDWORD(0x68000090));
_WDWORD(0x20000094, _RDWORD(0x68000094));
_WDWORD(0x20000098, _RDWORD(0x68000098));
_WDWORD(0x2000009C, _RDWORD(0x6800009C));

_WDWORD(0x200000A0, _RDWORD(0x680000A0));
_WDWORD(0x200000A4, _RDWORD(0x680000A4));
_WDWORD(0x200000A8, _RDWORD(0x680000A8));
_WDWORD(0x200000AC, _RDWORD(0x680000AC));

_WDWORD(0x200000B0, _RDWORD(0x680000B0));
_WDWORD(0x200000B4, _RDWORD(0x680000B4));
_WDWORD(0x200000B8, _RDWORD(0x680000B8));
_WDWORD(0x200000BC, _RDWORD(0x680000BC));

_WDWORD(0x200000C0, _RDWORD(0x680000C0));
_WDWORD(0x200000C4, _RDWORD(0x680000C4));
```

```
    _WDWORD(0x200000C8, _RDWORD(0x680000C8));
    _WDWORD(0x200000CC, _RDWORD(0x680000CC));

    _WDWORD(0x200000D0, _RDWORD(0x680000D0));
    _WDWORD(0x200000D4, _RDWORD(0x680000D4));
    _WDWORD(0x200000D8, _RDWORD(0x680000D8));
    _WDWORD(0x200000DC, _RDWORD(0x680000DC));

    _WDWORD(0x200000E0, _RDWORD(0x680000E0));
    _WDWORD(0x200000E4, _RDWORD(0x680000E4));
    _WDWORD(0x200000E8, _RDWORD(0x680000E8));
    _WDWORD(0x200000EC, _RDWORD(0x680000EC));

    _WDWORD(0x200000F0, _RDWORD(0x680000F0));
    _WDWORD(0x200000F4, _RDWORD(0x680000F4));
    _WDWORD(0x200000F8, _RDWORD(0x680000F8));
    _WDWORD(0x200000FC, _RDWORD(0x680000FC));

    _WDWORD(0x20000100, _RDWORD(0x68000100));
    _WDWORD(0x20000104, _RDWORD(0x68000104));
    _WDWORD(0x20000108, _RDWORD(0x68000108));
    _WDWORD(0x2000010C, _RDWORD(0x6800010C));

    _WDWORD(0x20000110, _RDWORD(0x68000110));
    _WDWORD(0x20000114, _RDWORD(0x68000114));
    _WDWORD(0x20000118, _RDWORD(0x68000118));
    _WDWORD(0x2000011C, _RDWORD(0x6800011C));

    _WDWORD(0x20000120, _RDWORD(0x68000120));
    _WDWORD(0x20000124, _RDWORD(0x68000124));
    _WDWORD(0x20000128, _RDWORD(0x68000128));
    _WDWORD(0x2000012C, _RDWORD(0x6800012C));
}

// 从这里开始执行代码，之前的都是函数定义
InitSRAM();                          // 配置 FSMC 用于 SRAM
LOAD obj\output.axf INCREMENTAL      // 下载程序到外部 SRAM
CopyVectTable();                     // 将外部 SRAM 的中断向量表复制到 CPU 内部 RAM
Setup();                             // 配置堆栈和 PC 指针
g, main                              // 运行到 main()函数后暂停
```

由于STM32的FSMC总线缺省是不使能的，外部SRAM无法由仿真器直接读写，因此我们需要先配置FSMC总线以保证SRAM可以通过CPU直接访问。

IDE控制调试器将程序装入CPU外部SRAM后，0x68000000单元存储了程序的堆栈初值（堆栈区域的最大值+4，向下增长）。0x68000004单元存储了复位向量地址。

3.3. IAR EWARM开发软件

3.3.1. IAR EWARM简介

IAR Embedded Workbench for ARM version 是一个针对ARM 处理器的集成开发环境，包含项目管理器、编辑器、编译连接工具和支持RTOS 的调试工具，在该环境下可以使用C/C++和汇编语言方便地开发嵌入式应用程序。

IAR EWARM 的主要优势：

- 不同结构，一种方案

对于30余种不同架构的处理器，IAR Embedded Workbench提供统一的使用界面，是一个真正的集成开发环境；方便了从8位/16位处理器转向ARM的用户。

- 高度集成的工具链

EWARM无缝集成了C/C++ Compiler, Assembler, Linker, Librarian, Text Editor, Project Manager以及C-SPY(tm) Debugger等工具，并能够方便地与第三方的Text Editor，Source Control System以及其它工具进行整合，使得用户能够在一个熟悉的IDE下，不中断地完成整个嵌入式软件的开发。

- ARM 内核和芯片

支持市面上所有的ARM内核，包括最新的Cortex-M3。此外对于绝大多数种类的ARM处理器还提供芯片级的支持，包括头文件/寄存器描述文件/Flash烧写/初始化例程等，使得用户很快就能进行开发和调试。

- C/C++ 优化编译器

提供先进的ARM C/C++ Compiler，其优化性能居业界领先地位，能够最大程度地生成体积小，速度快的优质可执行代码。每个源文件/每个函数的优化方式和优化级别均可方便地调节，以适应各种不同的应用。IAR的ARM C/C++ Compiler既能够严格遵循ANSI C/Embedded C++国际标准，也提供针对ARM架构的语言扩展，还支持对MISRA C编程规则进行自动校验。

- 强大的C-SPY 调试器

灵活的断点设置/精确的运行控制/丰富的资源监控/强大的模拟仿真

- 硬件调试更容易

支持所有主流的JTAG ARM仿真器，从成本低廉的Macraigor Wiggler等JTAG接口，到性价比极高的IAR J-Link，以及所有符合ARM公司RDI调试标准的JTAG仿真器。对于含有ETM模块的ARM处理器，还提供对IAR J-Trace等Trace仿真器的支持。

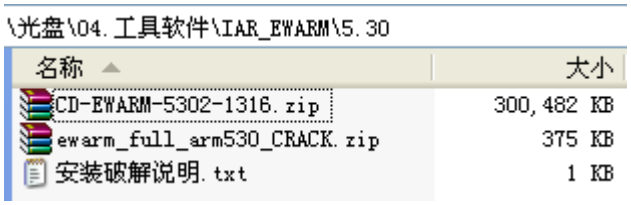
- 实时多任务操作系统调试

对于IAR PowerPAC, Micrim μC/OS-II，Segger embOS，OSE Epsilon，OSEK (ORTI)，ThreadX，CMX-RTX，CMX-Tiny+，RTXC Quadros，Fusion，MiSPO NORTi，SMX等操作系统，EWARM均提供Kernel-Awareness调试插件 ,以帮助在调试时察看任务/队列/信号量/定时器等与操作系统相关的信息。

3.3.2. IAR EWARM安装

安装程序：

安富莱开发配套光盘提供了EWARM的安装软件。



- CD-EWARM-5302-1316.zip ：安装文件压缩包
- ewarm_full_arm530_CRACK.zip ：软件注册机

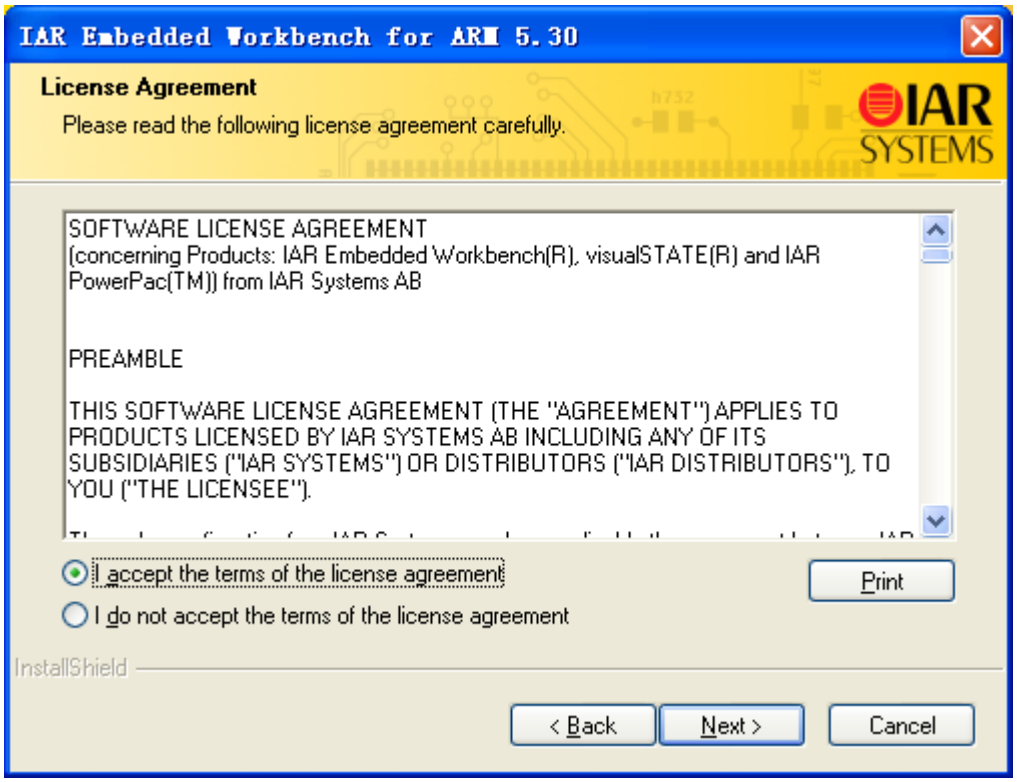
安装步骤：

1、将光盘上的 CD-EWARM-5302-1316.zip 文件解压到硬盘任意文件夹。

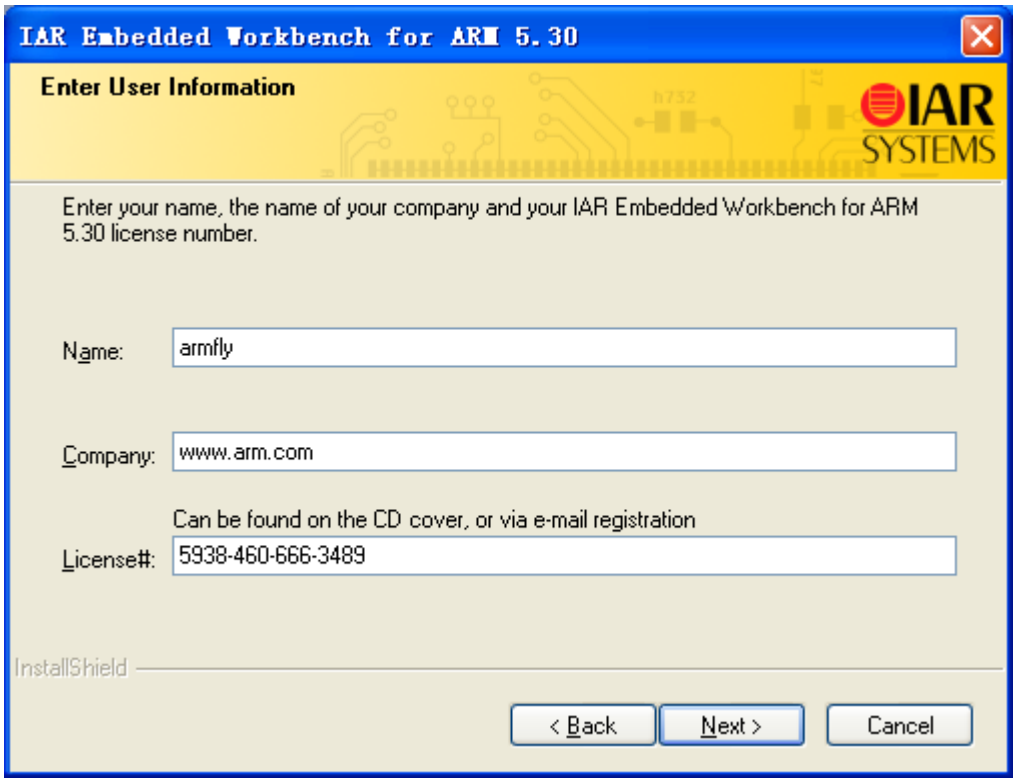
2、运行autorun.exe。出现如下界面：

3、点击“Install IAR Embedded Workbench”开始安装。等待若干秒后，将欢迎界面：

4、点击“Next”按钮继续安装。出现软件授权声明界面：



5、选择“I accept the terms of the license agreement ” 同意，然后点击“Next”按钮继续安装。出现输入用户信息的界面：

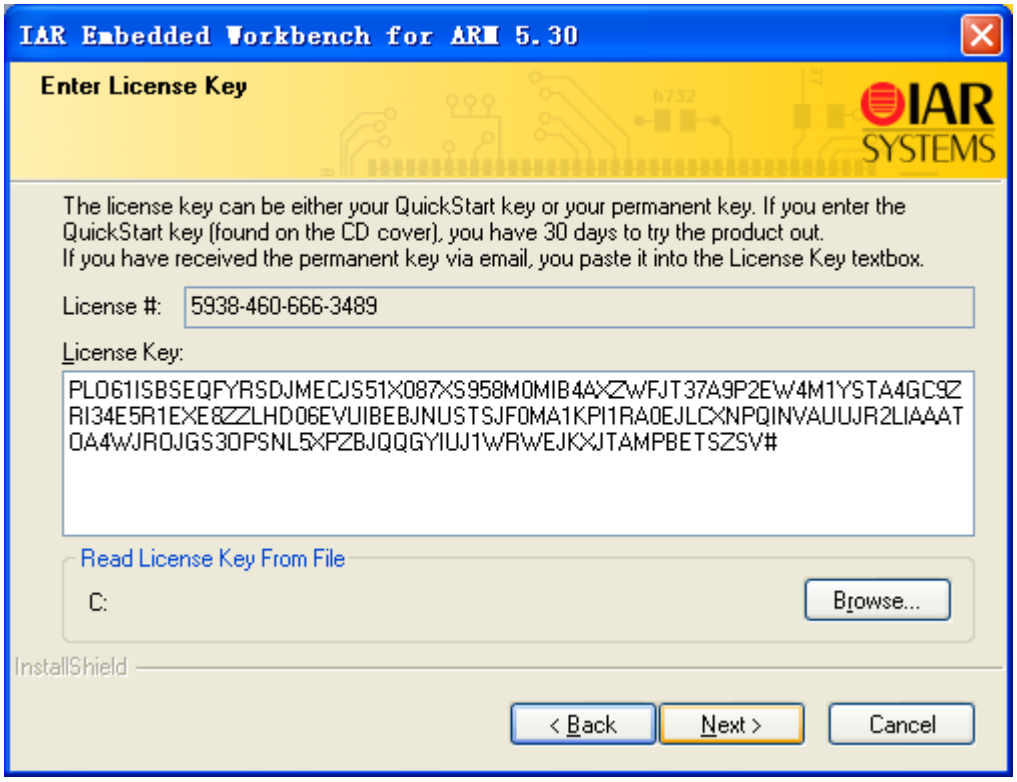


如果不能自动填充License，请将光盘上的ewarm_full_arm530_CRACK.zip文件解压到硬盘的任意文件夹。然后运行EWARM_FULL_ARM530.exe，界面如下：

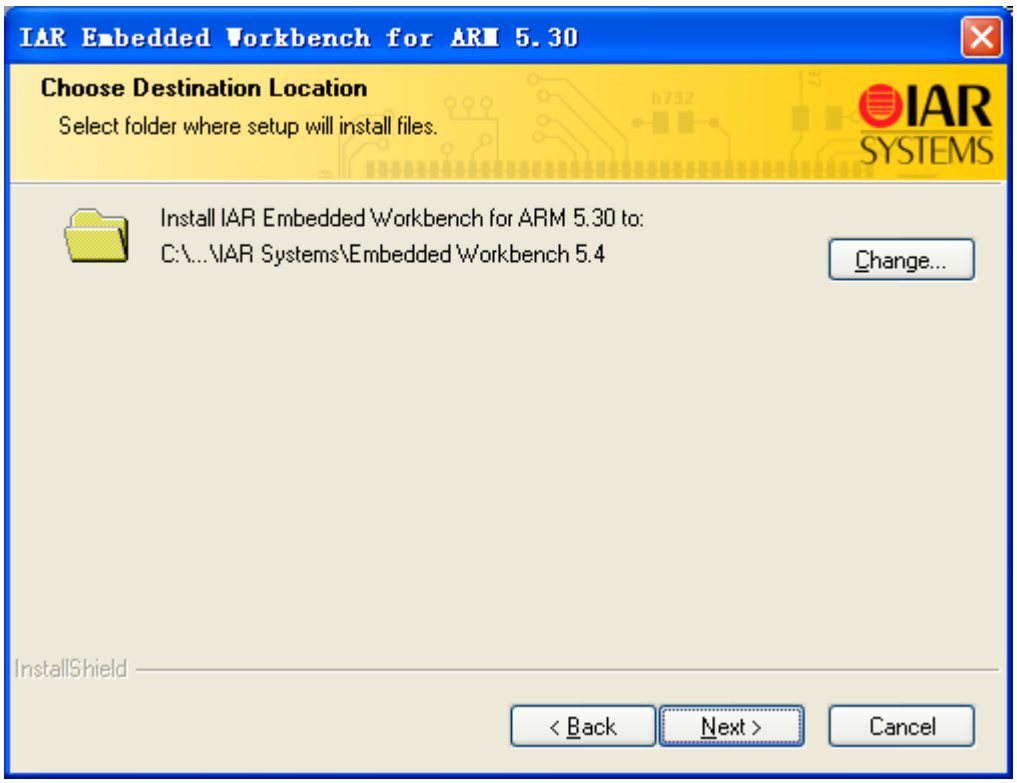
“Product”栏选择“Enbedded Workbench For ARM v5.30”。然后将“License number”下面的文本复制到IAR安装界面的“License#”文本框中。

下一步安装界面需要“License key”，如果安装软件不能自动填充，请采取相同的处理办法。

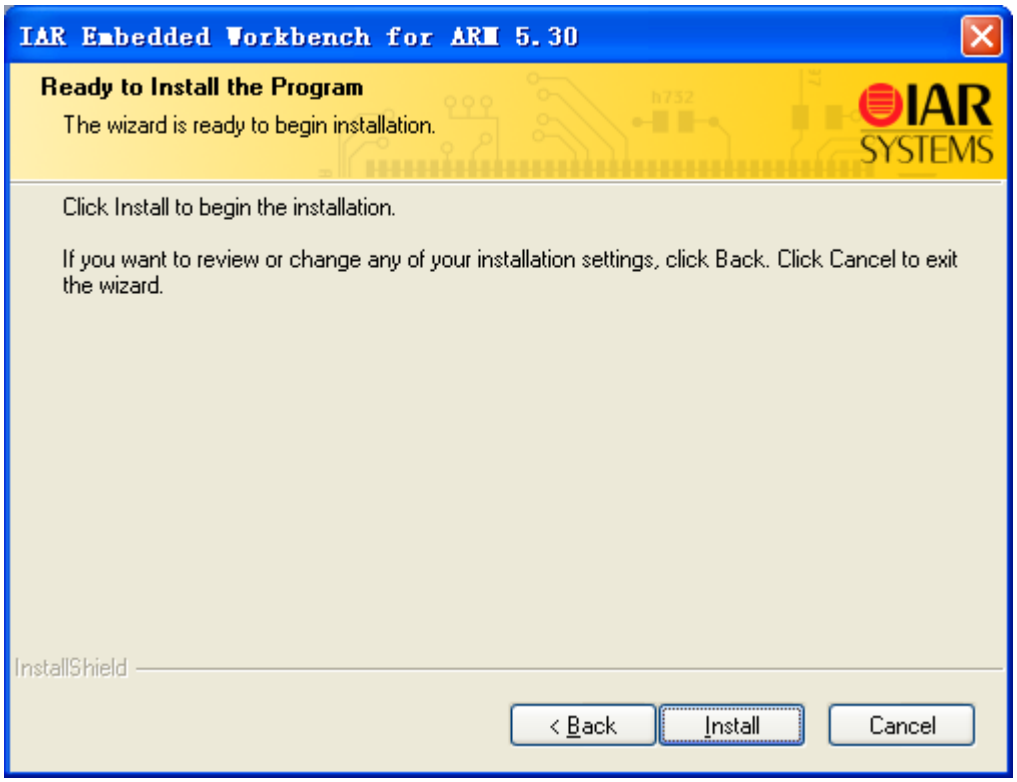
6、安装软件会自动填充每个字段，我们只需要点击“Next”按钮继续安装。出现请求输入License的界面：



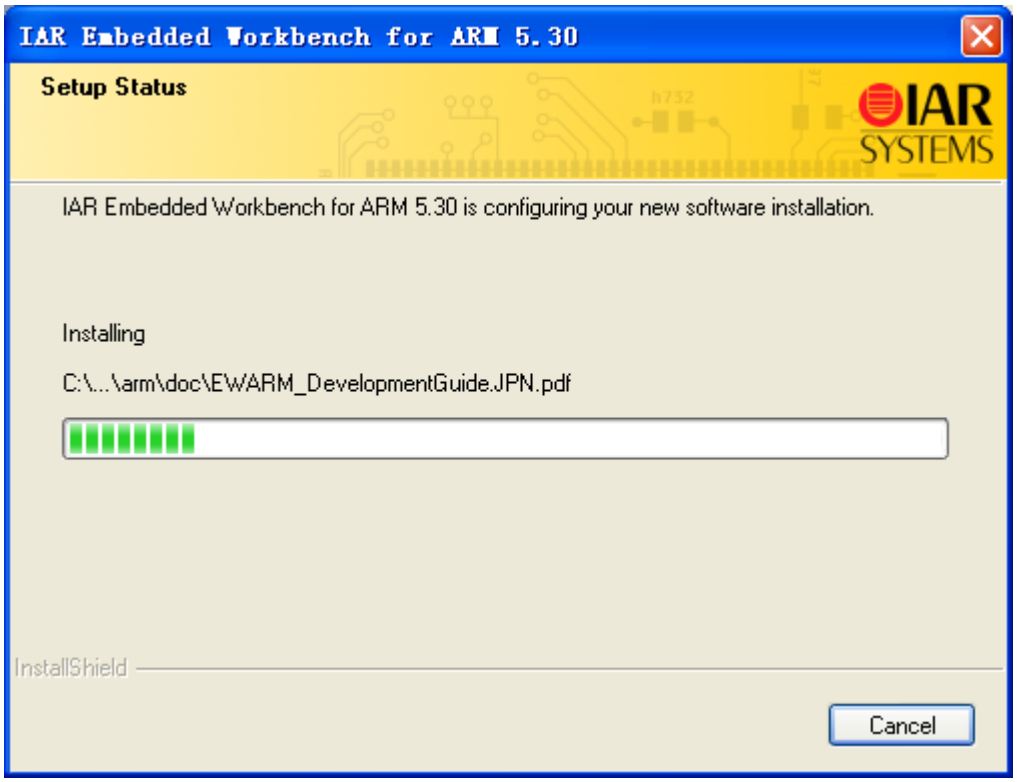
7、安装软件会自动填充License Key字段。我们只需要点击“Next”按钮继续安装。出现选择安装目标文件夹的界面：



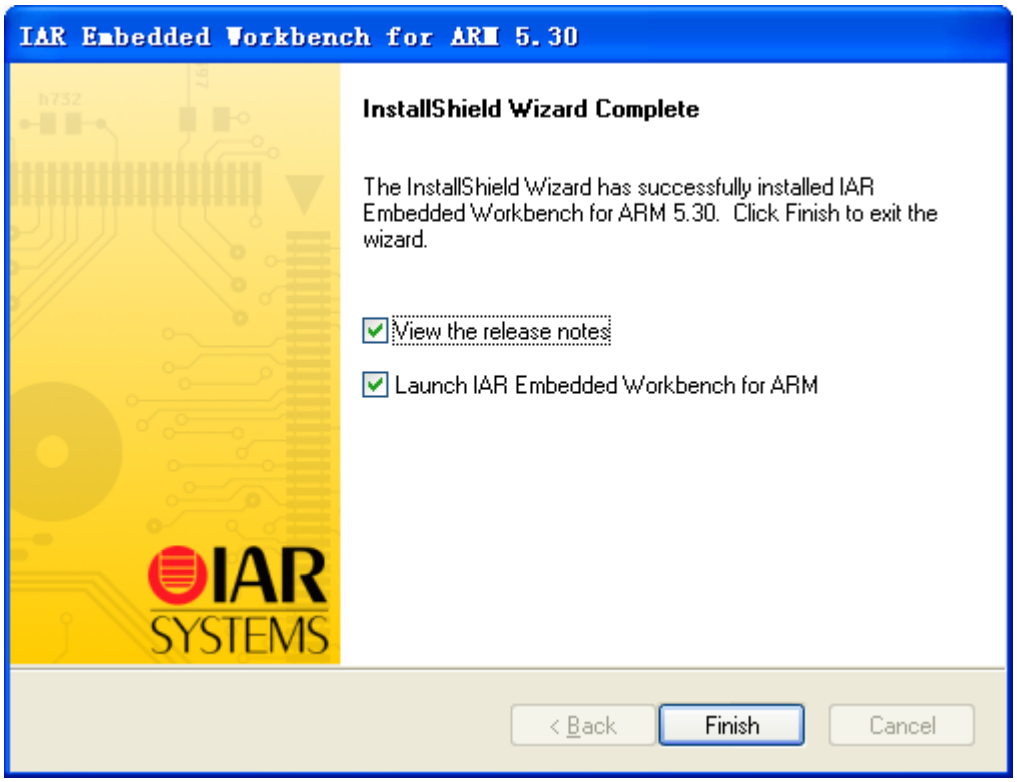
8、 一般情况下，我们就选择默认的安装路径。如果需要更改安装路径，请点击“Change...”按钮。我们现在点击“Next”按钮继续安装。出现确认开始安装的界面：



9、 一般情况下，我们就选择默认的安装路径。如果需要更改安装路径，请点击“Change...”按钮。我们现在点击“Next”按钮正式开始安装。下面是正在安装的界面：



10、 等待几分钟后，出现安装完成的界面



11、 点击“Finish”按钮。安装成功结束。

3.4. Windows超级终端工具

超级终端是Windows操作系统自带的一个通讯工具，可以通过这个工具对路由器交换机等进行配置。使用调制解调器或一条零调制解调电缆以太网连接，再调用此程序能够连接到其他计算机、Telnet 站点、公告板系统（BBS)、联机服务和主机。我们可以用它来调试电路是否可行。

使用：开始→程序→附件→通讯→超级终端（可新建或者使用现有的连接对设备进行配置）

启动命令是：hypertrm.exe

超级终端是一个通用的串行交互软件，很多嵌入式应用的系统有与之交换的相应程序，通过这些程序，可以通过超级终端与嵌入式系统交互，使超级终端成为嵌入式系统的“显示器”。

超级终端的原理

超级终端的原理并不复杂，它是将用户输入随时发向串口（采用TCP协议时是发往网口，这里只说串口的情况），但并不显示输入。它显示的是从串口接收到的字符。所以，嵌入式系统的相应程序应该完成的任务便是：

1、将自己的启动信息、过程信息主动发到运行有超级终端的主机；

2、将接收到的字符返回到主机，同时发送需要，也可以远程管理服务器。

超级终端常用操作

超级终端应用比较简单，和一般的串口软件差不多。

1、发送0x0C（12）：清屏；

2、发送0x08（8）：将光标退格（注意这并不删除字符）；

3、发送0x09（9）；将光标右移一个制表符（相当于TAB键）；

4、发送0x0D（13）；将光标移动到行首；

5、发送0x0A（10）或0x0B（11）；将光标移动到同一列的下一行；

6、发送0x0D跟0x0A，换行功能。

疑难问题解答

1、把超级终端最大化时，那个实际屏幕还是没有变化。

原因："超级终端"的终端屏幕大小由所使用的字体大小决定。它将自行显示为 24 行，每行为 80 或 132 个字符，字体为所选字体。

解决方案：在超级终端的"查看"菜单上，选择"字体"。如果想要较大的终端屏幕，就选择较大的字体。如果想要较小的终端屏幕，就选择较小的字体。

2、键入的信息没有显示在超级终端上。

原因：终端屏幕显示的信息是来自远程计算机所发送的，而不是已输入到本地计算机上的信息。为了查看所键入的信息，远程计算机必须可反馈输入信息。这可能会在输入信息与终端屏幕显示信息之间存在时间滞后的问题。

解决方案：请确保与远程计算机正确连接，并且远程计算机可以反馈用户输入信息。

3、ANSI字符不能够正确显示。

原因：未使用终端字体。

解决方案：在超级终端的"查看"菜单上，选择"字体"。单击"终端"，然后选择"确定"。

4、连接到远程计算机后，终端屏幕显示无意义信息。

原因：未选择正确的终端仿真类型。

解决方案：在超级终端的"文件"菜单上，选择"属性"。选中"设置"选项卡。在"仿真"下拉框中，选择远程计算机的终端类型。如果远程计算机类型没有在下拉框中列出，则超级终端不支持该类型。

5、不能从终端删除字符。

原因：所连接的远程计算机已经控制了显示在终端屏幕上的字符。远程计算机期望光标能根据已发送到屏幕上的数据而定位到屏幕中的特定位置。如果在本地上改变该屏幕，那么就有可能以主机所不能预料或控制的方式，潜在地中断了您与远程计算机之间交互操作。因此，超级终端不允许从屏幕上删除字符。

解决方案：不能。

6、用CTRL+V不能将数据粘贴到终端屏幕。

原因：如果在该连接属性的"终端键"进行了设置，按 CTRL+V 将会给模拟器发送转义序列。许多主机使用 CTRL+V 来导航它们的系统。

解决方案：可以将该设置更改到"Windows 键"中，然后 CTRL+V 就会正常运作。如要更改，请单击超级终端"文件"菜单中的"属性"。单击"设置"选项卡，然后单击" Windows 键"单选按钮。

3.5. J-Link仿真器

产品介绍

J-Link是SEGGER公司为支持仿真ARM内核芯片推出的JTAG仿真器。配合IAR EWARM，ADS，KEIL，WINARM，RealView等集成开发环境支持所有ARM7/ARM9内核芯片的仿真，通过RDI接口和各集成开发环境无缝连接，操作方便、连接方便、简单易学，是学习开发ARM最好最实用的开发工具。

SEGGER公司更新固件频率很快，大家可以到SEGGER网站下载最新版本。

下载地址：http://www.segger.com/download_jlink.html

J-Link ARM主要特点

- IAR EWARM集成开发环境无缝连接的JTAG仿真器
- 支持所有ARM7/ARM9/ARM11内核的芯片，以及cortex M3，包括Thumb模式
- 支持ADS、IAR、KEIL、WINARM等几乎所有的开发环境
- 下载速度高达ARM7:600kB/s，ARM9:550kB/s，通过DCC最高可达800 kB/s
- 最高JTAG速度12 MHz
- 目标板电压范围1.2V –3.3V，5V兼容
- 自动速度识别功能
- 监测所有JTAG信号和目标板电压
- 完全即插即用
- 使用USB电源
- 带USB连接线和20芯扁平电缆
- 支持多JTAG器件串行连接

- 标准20芯JTAG仿真插头
- 选配14芯JTAG仿真插头
- 选配用于5V目标板的适配器
- 带J-Link TCP/IP server，允许通过TCP/ IP网络使用J-Link
- 带J-Flash软件，允许脱离IDE开发环境烧写目标板程序

J-Link支持ARM内核

- ARM7TDMI（Rev 1）
- ARM7TDMI（Rev 3）
- ARM7TDMI-S（Rev 4）
- ARM720T * ARM920T
- ARM926EJ-S
- ARM946E-S
- ARM966E-S
- ARM11
- Cortex-M3

J-Link硬件版本之间的区别

J-Link硬件版本有V1、V2、V3、V4、V5、V6、V7、V8。J-Link进入国内市场时，是从V6开始的，因此国内用的较多的版本就是V6、V7、V8。目前最新是V8版本，以前的版本已经停产了。

V6 和V7版本的J-Link 在硬件电路和软件方面都做了加强。

硬件电路方面增加了USB 保护，降低了仿真器功耗，拓展了接口电平支持范围。

软件方面更有重大改进，主要是支持了最新的SWD 接口及SWV (Serial Wire Viewer)，SWD 是ARM公司新推出的一种调试接口，它仅需要2 条线即可进行调试，与传统的4 线JTAG 相比可以有效减少调试占用的口线资源，有效提高少引脚芯片的口线利用率。目前SWD 接口主要存在Cortex-M3 内核的芯片上，如ST 公司的STM32 系列、Luminary 公司的LM3S 系列。

注意，只有V6 及以后版本的J-Link 才支持SWD！

V7.0版本J-LINK，除拥有上一版本V6.0的全部功能外，对于Cortex-M3的Serial Wire Viewer(SWV)速度是V6.0的12倍。

V8.0版本相对V7.0版本的改进：

- SWD硬件接口支持1.2-5.0V的目标板，V7.0的SWD只能支持3.3V的目标板。
- V8.0使用双色LED可以指示更多的工作状态，V7.0只有1个LED指示灯。
- V8.0将JTAG接口限流电阻由220欧姆修改为110欧姆，增强了JTAG驱动能力，提高了目标板的兼容性。

- 优化了固件结构，将固件升级功能移到bootloader区，使应用程序区扩大一倍，便于增加新的功能。

J-Link配套的软件

- **J-Mem：**可查询可修改内存；
- **J-Link Server：**可通过TCP/IP连接到J-Link；
- **J-Flash：**支持独立的Flash编程，可以作为量产解决方案；
- **RDI插件：**使J-Link适合任何RDI兼容的调试器如IAR、ADS、Relview和Keil等；
  - **RDI Flash BP：**可以实现在RDI下，在Flash中设置无限断点；
  - **RDI Flash DLL：**可以实现在RDI下的对Flash的独立编程；
- **GDB server：**可以实现在GDB环境下的调试。

4.基础例程(基于ST官方例程)

4.1. 基础例程介绍

基础例程是针对CPU某一个硬件功能的测试程序，这些测试程序功能单一，源代码代码比较简单，主要目的是帮助用户了解CPU的接口以及ST标准固件库的用法。

本章节介绍的基础例程全部来自于ST公司官方网站发布的源码，我们对每个例程添加了工程文件，并根据板子硬件的特点做了少量修改。

初学者通过运行基础例程，可以快速掌握开发环境和调试工具的基本用法，以及了解STM32系列CPU的基本特征。

STM32F103ZE-EK开发板配套的试验例程均基于ST公司的固件库。我们精心收集并整理了每个例程的源码，确保每个例程可在STM32F103ZE-EK开发板上调试并运行。

所有的基础例程均提供单独的KEIL和IAR的工程文件。每个KEIL的工程中包含3套目标代码设置：

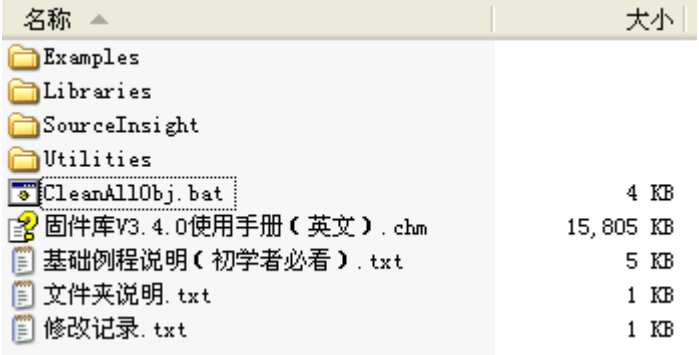
- 在CPU内部Flash运行，Target名字：flash
- 在CPU内部RAM运行，Target名字：ram
- 在外部SRAM运行，Target名字：extram

我们采用源代码方式将固件源码嵌入工程，没有采用库文件形式。这样做的目的是方便开发者步单步跟踪调试库中的每个函数。

每个工程的编译优化选项设置为最低，目的也是为了便于单步调试。

我们修改过的源代码均进行了注释，注释中均包含有“armfly ： ”标识。

4.2. 源码目录结构



Example ：存放每个例程的源代码和工程文件。

Libraries ：存放 ST公司提供的固件库源代码。

SourceInsight ： SourceInsight 工程文件（这是安富莱电子增加，目的是为了方便代码浏览）

Utilities ：存放跟开发板外围硬件相关的源代码文件。

4.3. 基础例程列表

基础例程有 80 多个，按照功能分别存放在如下几个文件夹下：

基础例程较多，初学者根据自己的需要选择性的了解即可。
其中 GPIO\IOToggle e 是跑马灯例程。

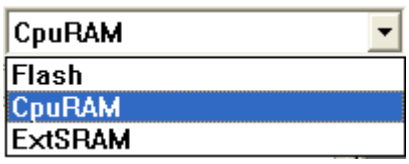
4.4. 基础例程调试方法（KEIL）

（1） 安装好KEIL MDK集成环境，以及segger仿真器驱动。

（2） 接好J-Link仿真器，给开发板上电。

（3） 进入 STM32基础例程（基于固件库V3.4.0）\Examples\GPIO\IOToggle\MDK-ARM(uV3) 目录，双击Project.Uv2，启动KEIL开发环境。

点击下来列表框，可以选择目标类型，推荐用ram。



点击按钮  编译和连接整个工程的源代码，以后可以只用点击按钮  编译修改过的源文件。
编译和连接成功后，就可以点击按钮  进入Debug状态。

进入Debug界面后，可以进行全速、单步、跟踪、暂停等操作。

4.5. 基础例程调试方法（IAR）

目前，IAR的工程都是在Flash中运行中。

1. 安装好IAR集成环境，以及segger仿真器驱动。
2. 接好J-Link仿真器，给开发板上电。
3. 进入 \Example\GPIO_IOToggle\EWARMv5目录，双击Project.eww，启动IAR EWARM开发环境。

点击按钮  编译和连接整个工程的源代码。
编译和连接成功后，就可以点击按钮  开始下载程序并且进入Debug状态。

进入Debug界面后，可以进行全速、单步、跟踪、暂停等操作。

5.开发板配套例程(安富莱原创)

5.1. 原创例程介绍

原创例程是安富莱电子根据实际项目经验整理的一些实用的综合应用的例子。程序结构和编码相对比较规范，每个例子都可以非常容易的扩充功能。您可以直接将其用于自己的产品或者项目。

原创例程全部采用ST标准固件库开发。目前最新的固件版本为3.5.0.

所有的例程均提供单独的KEIL和IAR的工程文件。每个KEIL的工程中包含3套目标代码设置：

- 在CPU内部Flash运行，Target名字：flash
- 在CPU内部RAM运行，Target名字：ram
- 在外部SRAM运行，Target名字：extram　[注意：部分例子不支持]

我们采用源代码方式将固件源码嵌入工程，没有采用库文件形式。这样做的目的是方便开发者步单步跟踪调试库中的每个函数。

每个工程的编译优化选项设置为最低，目的也是为了便于单步调试。

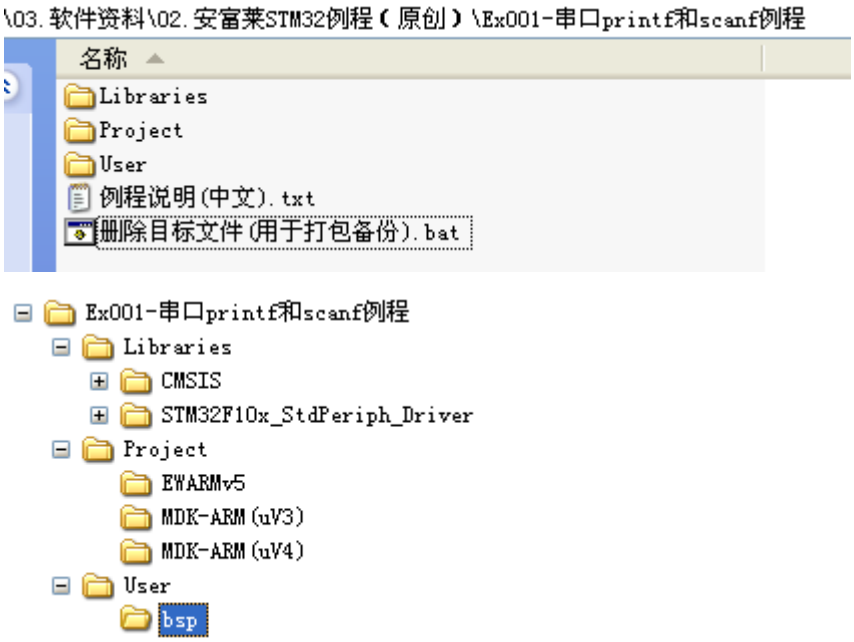
5.2. 光盘路径

下面是例程文件夹的截图（升级版本的例程也会放在这个文件夹）

每个压缩包都是一个独立的工程，包含用到的库文件源代码。每个工程包含KEIL和IAR两个工程配置。

支持在CPU内部Flash、CPU内部RAM、CPU外部SRAM调试程序。

5.3. 例程文件夹说明



Libraries ： ST 官方固件库。我们未作任何修改。大家可以从st 官网免费下载。ST 官方原始文件为stm32f10x_stdperiph_lib.zip，展开后即可看到Libraries 文件夹。我们将其独立出来，是为了方便用户自己升级库文件。

User ： 用户源代码。这是最宝贵的文件夹。

bsp ：存放板级支持包（Board Support Package），也就是硬件底层驱动程序。

Project ： 工程文件夹。存放开发环境的配置文件。下面有2 个子目录

MDK-ARM(Uv3) ：RealView MDK，就是我们常说的KEIL，也可以简称为 MDK。

MDK-ARM(Uv4) ：RealView MDK，就是我们常说的KEIL，也可以简称为 MDK。

EWARMv5 ：Embedded WorkBench for ARM，就是我们常说的IAR。

在编译和调试时，Project 文件夹内会产生很多中间临时文件（*.obj，*.bak，*.list 等），小文件较多。为了便于同时维护IAR 和KEIL 工程，因此单独将Project 列出来。

在例程根目录下，有一个bat 文件：**删除目标文件(用于打包备份).bat**

双击后，可以删除Project 下的临时文件，这样对代码打包后的Size 将大大减小。

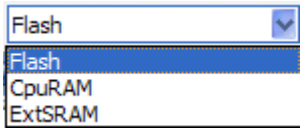
5.4. 配套例程调试方法（KEIL）

（1） 安装好KEIL MDK集成环境，以及segger仿真器驱动。

（2） 接好J-Link仿真器，给开发板上电。

（3） 进入\Project\MDK-ARM(uV3)目录，双击Project.Uv2，启动KEIL开发环境。

点击下来列表框，可以选择目标类型。推荐用Flash 或者CpuRam。



点击按钮  编译和连接整个工程的源代码，以后可以只用点击按钮  编译修改过的源文件。
编译和连接成功后，就可以点击按钮  进入Debug状态。

进入 Debug 界面后，可以进行全速、单步、跟踪、暂停等操作。

5.5. 配套例程调试方法（IAR）

目前，IAR的工程都是在Flash中运行中。

1. 安装好IAR集成环境，以及segger仿真器驱动。
2. 接好J-Link仿真器，给开发板上电。
3. 进入 \Project\EWARMv5目录，双击Project.eww，启动IAR EWARM开发环境。

点击按钮  编译和连接整个工程的源代码。
编译和连接成功后，就可以点击按钮  开始下载程序并且进入Debug状态。

进入Debug界面后，可以进行全速、单步、跟踪、暂停等操作。

6.附录：STM32 硬件设计指南

6.1. 注意事项

STM32的基本系统主要涉及下面几个部分：

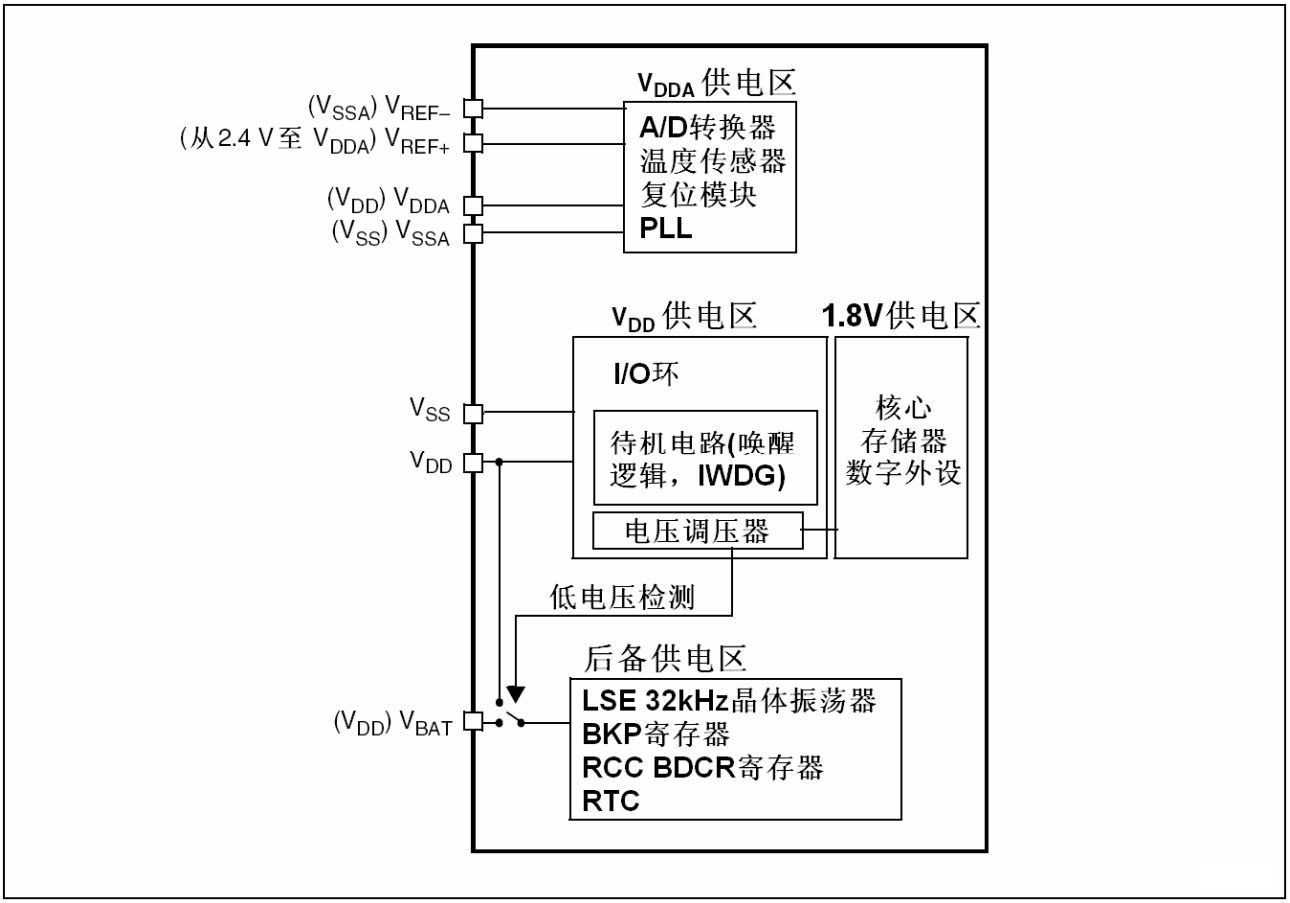
- **电源**
  - 无论是否使用模拟部分和AD部分，MCU外围出去的VCC和GND，VDDA、VSSA、Vref(如果封装有该引脚)都必需要连接，不可悬空；
  - 对于每组对应的VDD和GND都应至少放置一个104的陶瓷电容用于滤波，并将该电容尽量靠近MCU。
- **复位、启动选择**
  - Boot引脚与JTAG无关。其仅用于MCU启动后，判断执行代码的起始地址。在设计产品时，一般Boot引脚应配置为从CPU内部Flash启动；
  - 在电路设计上可能Boot引脚不会使用，但要求一定要外部连接电阻到地或电源，切不可悬空。
- **ADC**
  - ADC是有工作电压的，且与MCU的工作电压不完全相同。MCU工作电压可以到2.0V～3.6V，但ADC模块工作的电压在2.4V～3.6V。设计电路时需要注意。
- **时钟**
  - STM32上电默认是使用内部高速RC时钟(HSI)启动运行，如果做外部时钟(HSE)切换，外部时钟是不会运行的。因此，判断最小系统是否工作时，用示波器检查OSC是否有时钟信号是错误的方法；
  - RTC时钟要求使用的32.768kHz振荡器的负载电容是6pF ,这个电容区别于振荡器外部接的负载电容。
- **GPIO**
  - IO推动电流较大的负载（如LED）时，建议使用灌电流的方式，即输出低电平点亮；
  - 在Stop等低功耗模式下，为了更省电，建议GPIO配置为带上拉的输出模式，输出电平由外部电路决定。
- **FSMC**
  - 只有100pin和144pin的STM32才有FSMC。FSMC的功能与I2C存在硬件冲突，这是CPU自身的硬件缺陷。如果FSMC时钟打开，I2C1的硬件模式无法工作。这在STM32F10xxx的勘误表中是有描述的。

6.2. 供电

6.2.1. 简介

该芯片要求2.0~3.6V的操作电压(VDD)，并采用嵌入式的调压器提供内部1.8V的数字电源。

当主电源VDD关闭时，实时时钟(RTC)和备用寄存器可以从VBAT供电。



注意： *VDDA*和*VSSA*必须分别连到*VDD*和*VSS*。

6.2.2. 独立A/D转换器供电以及参考电压

为提高转换精度，ADC有一个独立的电源供应，它可以被单独滤波和屏蔽以不受PCB噪音的干扰。

- 一个独立的VDDA引脚给ADC供电
- VSSA引脚提供一个隔离的接地输入

若有VREF–(取决于封装)时，它必须连到VSSA。

100引脚和144引脚的封装

为保证低电压输入时能得到更好的精度，用户可以连接一个独立的外部参考电压到VREF+,它的电压范围为2.4V到VDDA。

64引脚及更小的封装

没有VREF+和VREF-，它们在内部分别被连接到ADC的供电电源(VDDA)和ADC的地(VSSA)。

6.2.3. 备用电池

为了在VDD关闭时仍能保持备份寄存器的内容 ,VBAT引脚可以有选择地连接到一个由电池或其它电源提供的备用电压。

VBAT引脚也给RTC单元供电，使得RTC在主数字电源(VDD)关闭时仍能正常运行。VBAT的开关由复位模块内的掉电复位(PDR)电路控制。

如果应用中没有外部电池，VBAT必须在外部被连接到VDD。

6.2.4. 电压调压器

复位后调压器始终开启。根据应用模式的不同，它也有三种不同的工作模式。

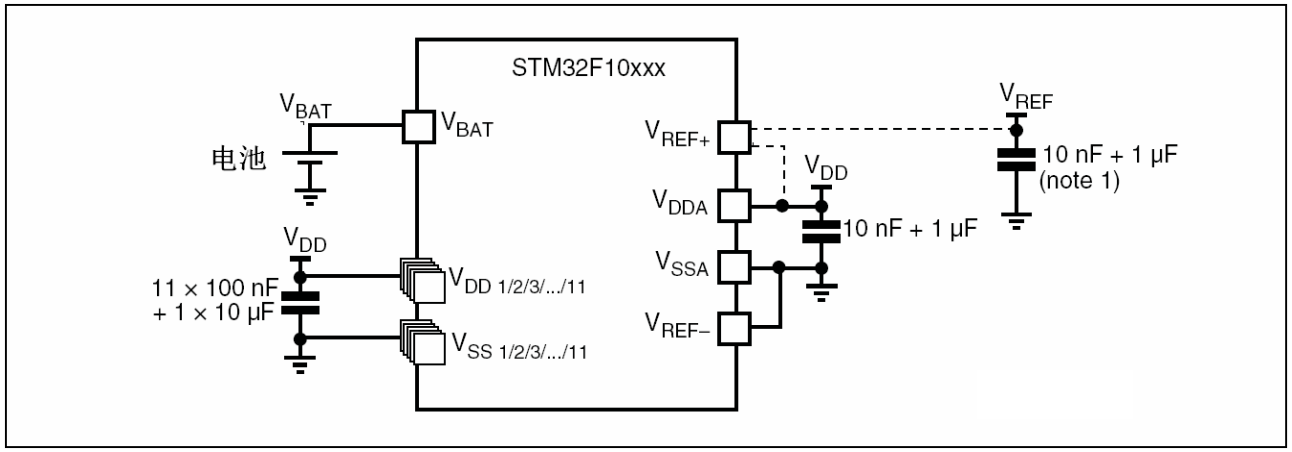
- 在运行模式下，调压器以正常功耗模式提供1.8V电源(内核、内存和数字外设)
- 在停止模式下，调压器以低功耗模式提供1.8V电源，以保存寄存器和SRAM的内容
- 在待机模式下，调压器停止供电。除了备用电路和备份域外，寄存器和SRAM的内容全部丢失

6.2.5. 供电方案

电路由稳定的电源VDD供电。

- 注意：
- 如果使用ADC，VDD的范围必须在2.4V到3.6V之间
- 如果没有使用ADC，VDD的范围为2V到3.6V
- VDD引脚必须连接到带外部稳定电容(11个100nF的陶瓷电容和一个钽电容(最小值4.7μF，典型值10μF)的VDD电源。
- VBAT引脚必须被连接到外部电池(1.8V < VBAT < 3.6V)。如果没有外部电池，这个引脚必须和100nF的陶瓷电容一起连接到VDD电源上
- VDDA引脚必须连接到两个外部稳定电容(10nF陶瓷电容+1μF钽电容)。
- VREF+引脚可以连接到VDDA外部电源。如果在VREF+上使用单独的外部参考电压，必须在这个引脚上连接一个10nF和一个1μF的电容。在所有情况下，VREF+必须在2.4V和VDDA之间。

供电方案：

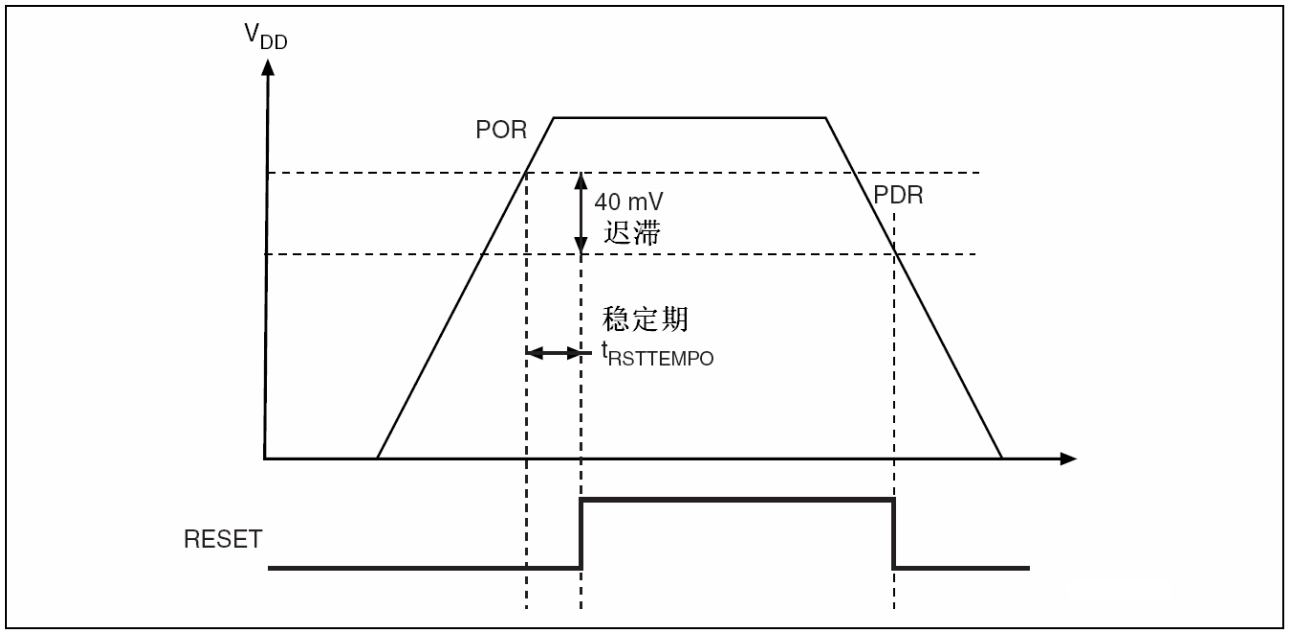


- note1 可选。如果在VREF+上使用单独的外部参考电压，必须连接两个电容(10nF和1uF)。
- VREF+连接到VDDA或VREF+。

6.2.6. 上电复位(POR)/掉电复位(PDR)

STM32集成了一个上电复位(POR)和掉电复位(PDR)电路，当供电电压达到2V时系统就能正常工作。只要VDD低于特定的阈值—VPOR/PDR，不需要外部复位电路，STM32就一直处于复位模式。更多有关上电/掉电复位阈值的细节，请参考STM32F101xx和STM32F103xx数据手册的电气性能部分。

上电复位/掉电复位波形：



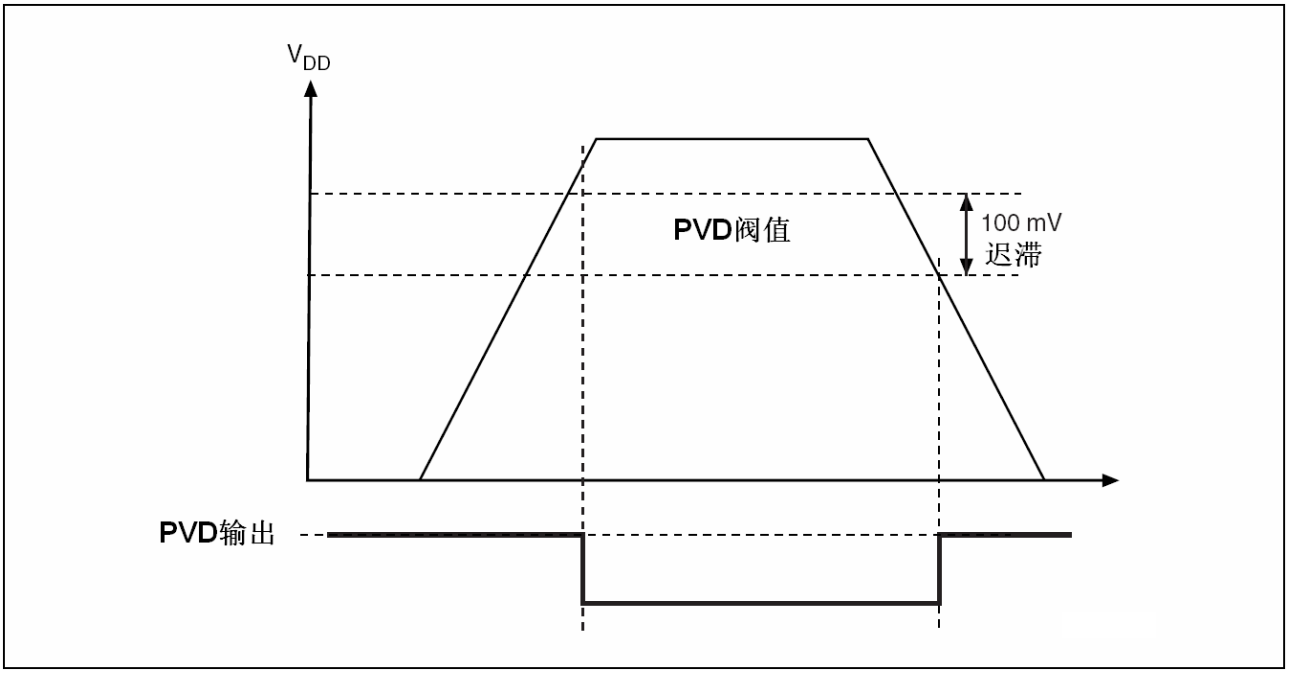
6.2.7. 可编程电压监测器(PVD)

用户可以利用PVD对VDD电压与电源控制寄存器(PWR_CR)中的PLS[2:0]位进行比较来监控电源 ,这几位选择监控电压的阀值。

通过设置PVDE位来使能PVD。

电源控制/状态寄存器(PWR_CSR)中的PVDO标志用来表明VDD是高于还是低于PVD的电压阈值。该事件在内部连接到外部中断的第16线 ,若该中断在外部中断寄存器中被使能 ,该事件还将产生一个中断。当VDD低于PVD阈值且/或当VDD高于PVD阈值(根据外部中断第16线的上升/下降沿触发配置)时产生PVD中断。例如，这一特性在实际中可用作执行紧急关闭的任务。

PVD阀值：



6.2.8. 系统复位

系统复位将复位除了时钟控制器CSR中的复位标志和备用域的寄存器以外的所有寄存器。

当下列事件有一个发生都将产生系统复位：

（1） NRST引脚上出现低电平(外部复位)

（2） 窗口看门狗计数终止(WWDG复位)

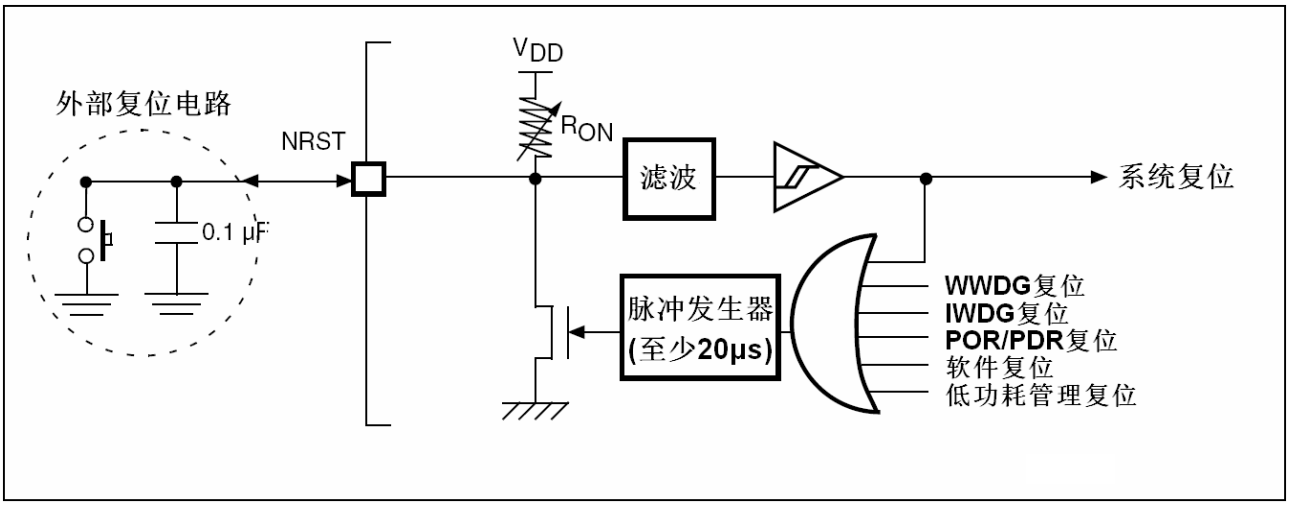
（3） 独立看门狗计数终止(IWDG复位)

（4） 软件复位(SW复位)

（5） 低功耗管理复位

可通过查看控制/状态寄存器(RCC_CSR)中的复位标志来识别复位源。

复位电路：



6.3. 时钟

三个不同的时钟源可以用来驱动系统时钟(SYSCLK)：

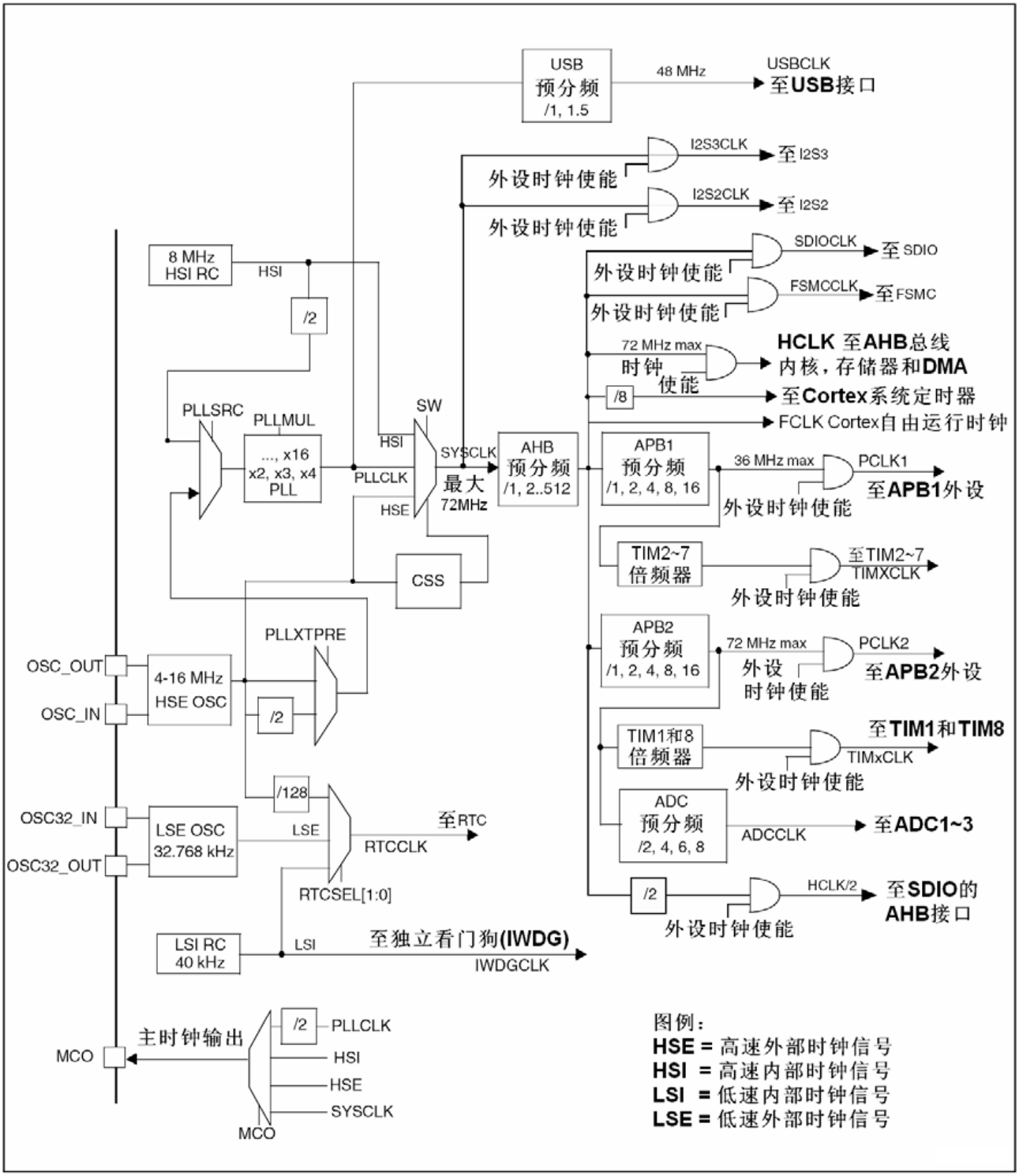
- HSI晶振时钟(高速内部时钟信号)
- HSE晶振时钟(高速外部时钟信号)
- PLL时钟

STM32有两个二级时钟源：

- 40kHz的低速内部RC，它可以驱动独立看门狗，还可选择地通过程序选择驱动RTC。RTC用于从停机/待机模式下自动唤醒系统。
- 32.768kHz的低速外部晶振，可选择它用来驱动RTC(RTCCLK)。

每个时钟源在不使用时都可以单独被打开或关闭，这样就可以优化系统功耗

时钟树：



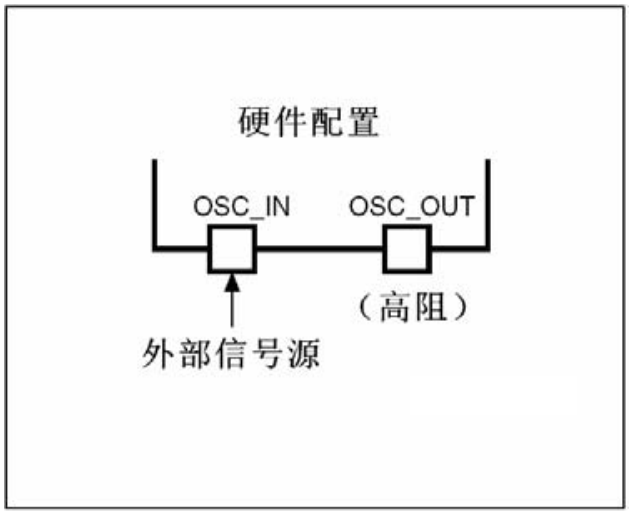
当使用HSI作为PLL时钟的输入时，所能达到的最大系统时钟为64MHz。

6.3.1. HSE时钟

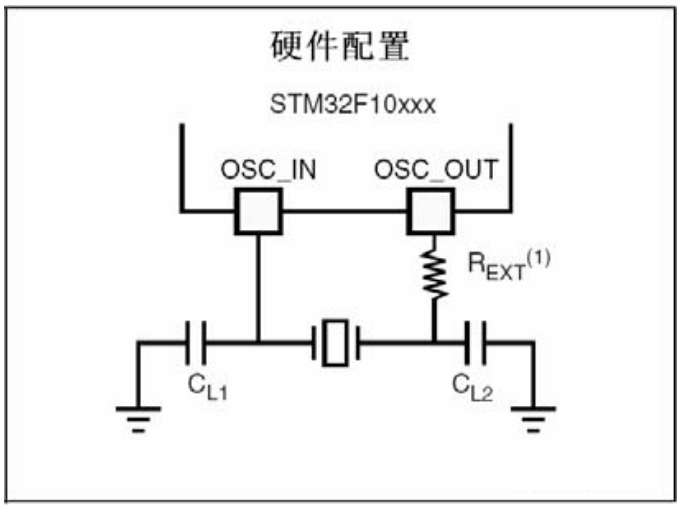
高速外部时钟信号(HSE)由以下两种时钟源产生：

- HSE外部晶体 / 陶瓷 谐振器
- HSE用户外部时钟

外部时钟：



晶体：



（1） REXT的值由晶体特性决定。典型值的范围在5至6倍的RS(谐振器串行阻抗)

（2） 负载电容CL遵循以下公式：CL = CL1 x CL2 / (CL1 + CL2) + Cstray。这里，Cstray是引脚电容以及PCB相关的电容。典型值在2pF到7pF 之间。

外部时钟源(HSE旁路) ：

在这种模式下，必须提供一个外部时钟源。它的频率可高达25MHz。外部时钟信号(占空比为50%的方波、 正弦波或三角波)必须连到OSC_IN引脚，同时保证OSC_OUT引脚悬空

外部晶体 / 陶瓷谐振器(HSE晶体) ：

这个4~16MHz的外部晶振的优点在于能产生非常精确的主时钟。

谐振器和负载电容需要尽可能近地靠近振荡器的引脚，以减小输出失真和启动稳定时间。负载电容值必须根据选定的晶振进行调节。

对于CL1和CL2，我们推荐使用高质量的典型值在5pF到25pF的陶瓷电容，这种电容是设计用于需要高频率的场合，并且可以满足晶体或谐振器的需求。CL1和CL2通常具有相同的值。晶体制造商通常指定一个负载电容值，该值为CL1和CL2的串联电容值。当选择CL1和CL2时，PCB和MCU引脚的电容值也必须被计算进去(10pF可作为引脚和板电容的粗略估计)。

更多细节请参考STM32F101xx和STM32F103xx数据手册的电气特性部分。

6.3.2. LSE时钟

低速外部时钟源(LSE)可以由两个可能的时钟源来产生：

- LSE外部晶体 / 陶瓷谐振器
- LSE用户外部时钟

外部源(LSE 旁路)：

在这种模式下，必须提供一个外部时钟源。它的频率必须为32.768kHz。外部信号(占空比为50%的方波、 正弦波或三角波)必须连到OSC32_IN引脚，同时保证OSC_OUT引脚悬空。

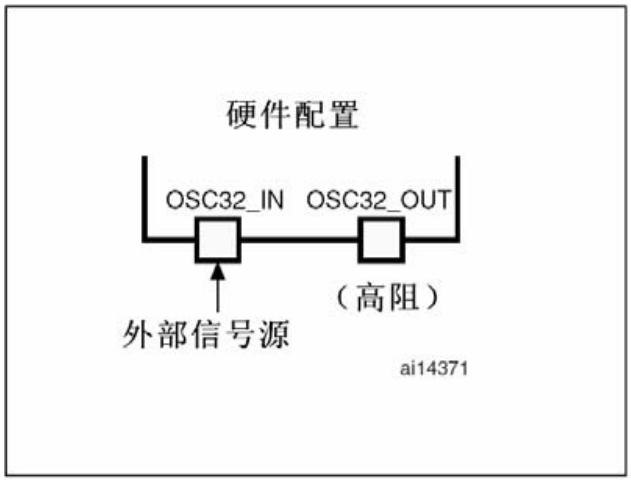
在这种模式下，必须提供一个外部时钟源。它的频率必须为32.768kHz。外部信号(占空比为50%的方波、 正弦波或三角波)必须连到OSC32_IN引脚，同时保证OSC_OUT引脚悬空。(见图10和2.2.1)。

外部晶体 / 陶瓷谐振器(LSE晶体) ：

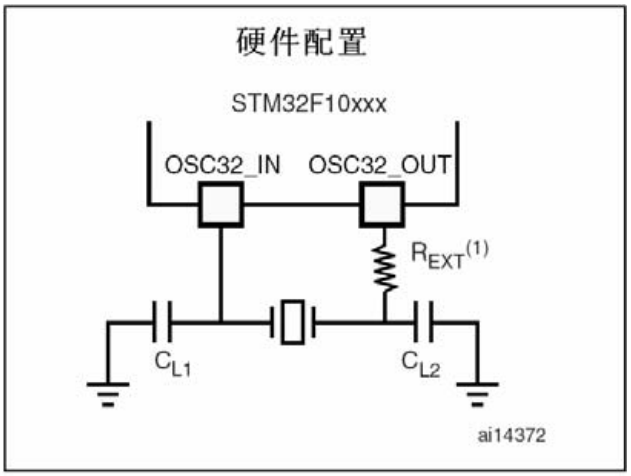
这个LSE晶体是一个32.768kHz的低速外部晶体或陶瓷谐振器。它的优点在于能为实时时钟部件(RTC)提供一个低速的，但高精确的时钟源。RTC可以用于时钟/日历或其它需要计时的场合。

谐振器和加载电容需要尽可能近地靠近晶振引脚，这样能使输出失真和启动稳定时间减到最小。负载电容值必须根据选定的晶振进行调节。

外部时钟



晶体/陶瓷谐振器



1．REXT的值由晶体特性决定。

2．OSC32_IN和OSC_OUT引脚可以用作GPIO，但是建议不要在同一个应用中同时使用该引脚的RTC和GPIO功能。

6.3.3. 时钟输出能力

微控制器的时钟输出(MCO)功能允许在外部MCO引脚上输出时钟信号。相应的GPIO端口必须设置为复用功能模式。下面的四个信号中的任何一个都可以选作MCO时钟：

- SYSCLK
- HSI
- HSE
- PLL时钟除以2

6.3.4. 时钟安全系统(CSS)

时钟安全系统可以通过软件来激活。一旦其被激活，时钟监测器将在HSE振荡器启动延迟后被使能，在HSE时钟关闭后被关闭。

- 如果HSE时钟发生故障，HSE振荡器被自动关闭，时钟失效事件被送到TIM1高级控制定时器的刹车输入端，并将产生时钟安全中断CSSI，从而允许MCU完成营救操作。此CSSI中断连到了Cortex™-M3的NMI(非屏蔽中断)异常向量。
- 如果HSE被直接或间接用作系统时钟(间接是指它被用作PLL输入时钟，而PLL时钟被用作系统时钟)，时钟故障将导致系统时钟被切换到HSI振荡器，并且禁止外部HSE晶振。时钟失效时，如果HSE晶振时钟是用作系统时钟的PLL的时钟输入(无论是否被分频)，那么PLL也被禁止。

6.4. 启动配置

6.4.1. 启动模式选择

在STM32F10xxx中，由BOOT[1:0]引脚决定了三种不同的启动模式，见下表：

BOOT1	BOOT0	启动模式	说 明
X	0	用户闪存存储器启动	主闪存存储器被选作启动区
0	1	从系统存储器启动	系统存储器被选作启动区
1	1	从内嵌SRAM启动	内嵌SRAM被选作启动区

这些选择将每个启动模式下的物理存储区域映射到存储块0(启动存储区)。BOOT引脚的值在复位后SYSCLK的第四个上升沿时被锁定。由用户来设置BOOT1和BOOT0引脚，选择在复位后需要的启动模式。

当退出待机模式时，也需要检测BOOT引脚的值。因此在待机模式下BOOT引脚也应保持需要的启动模式配置。

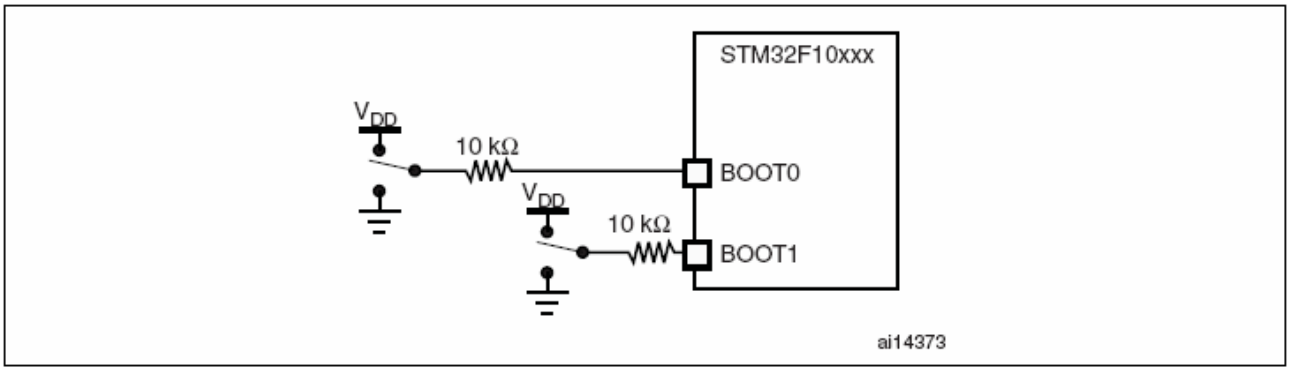
即使被重映射到启动存储空间，仍可以在相关存储区(闪存或SRAM)的原来地址访问它们。

启动延迟过后，CPU从启动存储器的0x0000_0004指示的地址开始执行代码。

6.4.2. 启动引脚连接

下图显示了STM32F10xxx选择启动存储器时所需的外部连接

启动模式实现实例



图中电阻值只作为典型值给出。

6.4.3. 内嵌自举模式

内嵌的自举程序用于通过串行接口(通常是UART1)对闪存进行重新编程。该程序位于系统存储器内，由ST在生产线上写入。

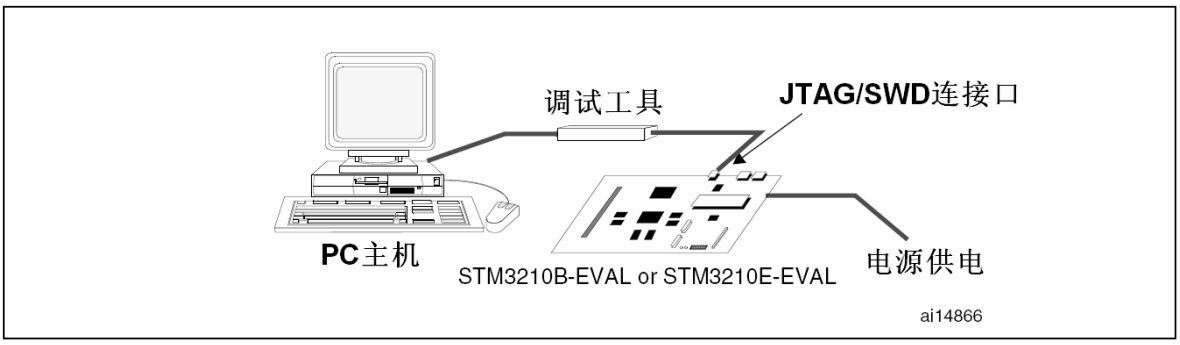
更多细节请参考www.st.com网站上的应用笔记AN2606。

6.5. 调试管理

6.5.1. 简介

主机/目标接口是用于将主机连接到目标板。这个接口由三个部分组成：一个硬件调试工具，一个JTAG或SWD连接器和一根连接主机和调试工具的连线。

主机与板的连接



6.5.2. SWJ调试端口(serial wire和JTAG)

STM32F10xxx内核集成了串行线/JTAG调试接口(SWJ – DP)。这是标准的ARM® CoreSight调试接口，包括JTAG – DP接口(5引脚)和SW – DP接口(2引脚)。

JTAG调试接口(JTAG – DP)为AHP – AP模块提供5针标准JTAG接口。

串行线调试接口(SW – DP)为AHP – AP模块提供2针(时钟+数据)接口。

在SWJ – DP接口中，SW – DP的2个引脚与JTAG 接口的5个引脚中的一些是复用的。

6.5.3. 引脚分布和调试端口脚

STM32F10xxx 微控制器的不同封装有不同的引脚数目。因此，某些与引脚相关的功能可能随封装而不同。

6.5.4. SWJ调试端口引脚

作为通用I/O口的复用功能，STM32F10xxx的5个管脚可用作SWJ-DP接口引脚。如下表所示，这些引脚在所有的封装里都存在。

调试端口引脚分配表：

SWJ-DP端口名称	JTAG调试接口		SWD调试接口		管脚分配
	方向	说明	方向	说明	
JTMS/SWDIO	输入	JTAG模式选择	入/出	串行数据输入/输出	PA13
JTCK/SWCLK	输入	JTAG时钟	输入	串行时钟	PA14
JTDI	输入	JTAG数据输入	-	——	PA15
JTDO/TRACESWO	输出	JTAG数据输出	-	跟踪时为TRACESWO信号	PB3
JNTRST	输入	JTAG模块复位	-	——	PB4

6.5.5. 灵活的SWJ-DP引脚分配

复位(SYSRESETn或PORESETn)后，属于SWJ – DP的5个引脚都被初始化为可被调试器使用的专用引脚(注意，跟踪输出脚并没有被初始化，除非调试器对其进行了定义)。

然而，STM32F10xxx 微控制器可通过一个寄存器来禁止SWJ – DP接口的部分或所有引脚的功能，这样就能释放这些专用引脚用于普通I/O。这个寄存器被映射到和Cortex™–M3系统总线相连接的APB桥上。这个寄存器由用户进行设置而不是由调试器完成。

SWJ I/O引脚可用性：

调试端口	SWJ IO引脚分配				
	PA13/ JTMS/ SWDIO	PA14/ JTCK/ SWCLK	PA15/ JTDI	PB3/ JTDO	PB4/ JNTRST
所有的SWJ引脚SWJ (JTAG-DP + SW-DP) – 复位状态	专用	专用	专用	专用	专用
所有的SWJ引脚SWJ (JTAG-DP + SW-DP) – 除了JNTRST引脚	专用	专用	专用	专用	
禁止JTAG-DP接口，允许SW-DP接口	专用	专用	可作为普通IO		
禁止JTAG-DP接口和SW-DP接口					

显示了不同情况下，释放的专用引脚

更多细节请参考www.st.com网站上的STM32F10xxx参考手册。

6.5.6. JTAG引脚的内部上拉和下拉电阻

由于JTAG的输入引脚直接连接到内部触发器来控制调试模式功能 ,所以JTAG的输入引脚一定不能是悬空。必须特别注意SWCLK/TCK引脚，因为它们直接连接到一些触发器的时钟端。

为了避免出现任何不受控制的I/O电平，STM32F10xxx在JTAG输入引脚内部嵌入了上拉和下拉电阻：

- JNTRST：内部上拉
- JTDI：内部上拉
- JTMS/SWDIO：内部上拉
- TCK/SWCLK：内部下拉

一旦JTAG的I/O被用户代码释放，GPIO控制器就再次取得了控制权。复位时这些I/O口的状态被设置到相应的状态：

- JNTRST：带上拉的输入

- JTDI：带上拉的输入
- JTMS/SWDIO：带上拉的输入
- JTCK/SWCLK：带下拉的输入
- JTDO：浮空输入

软件可以把这些I/O引脚用作普通的I/O。

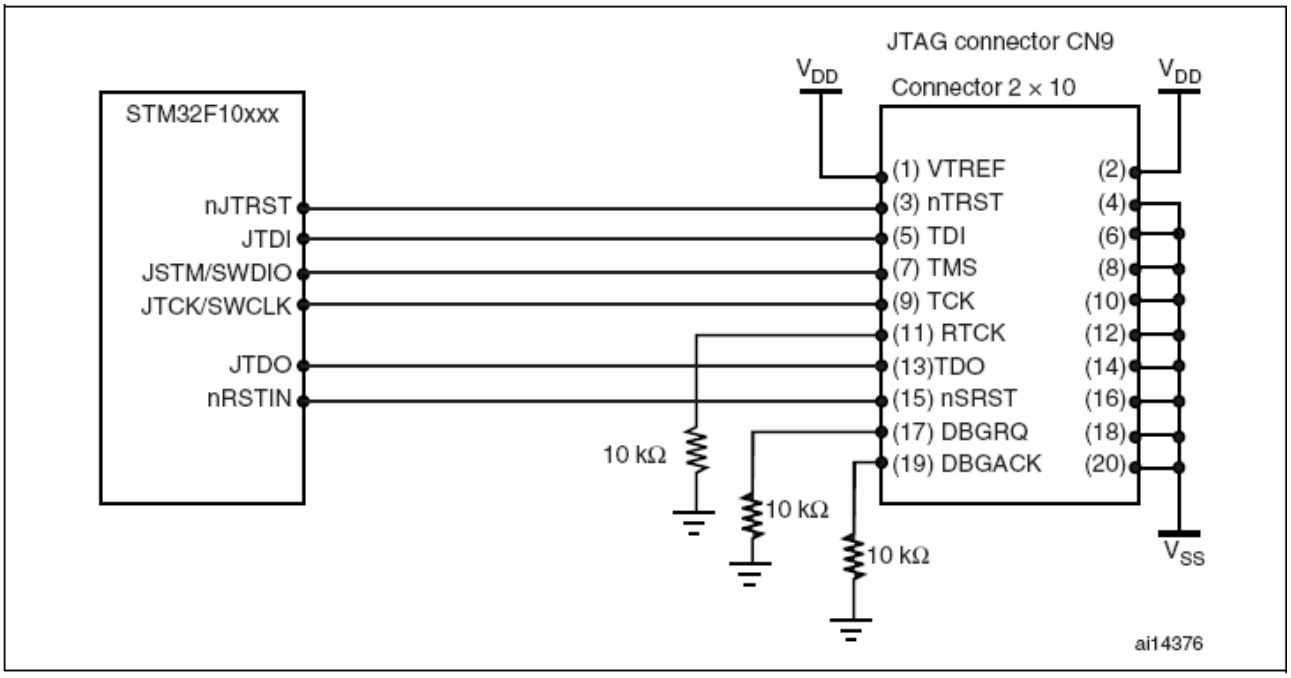
注意： JTAG的IEEE标准推荐对TDI，TMS和nTRST上拉，但是对TCK没有特别建议。然而，在STM32F10xxx中JTCK引脚有下拉电阻。

有了嵌入的上拉和下拉电阻，就不需要加外部电阻了。

6.5.7. 与标准JTAG连接器相连的SWJ调试端口

下图显示了STM32F10xxx和一个标准JTAG连接器的连接。

JTAG连接器的实现



6.6. PCB布线建议

6.6.1. 印制电路板

出于技术的考虑，最好使用有专门独立的接地层(VSS)和专门独立的供电层(VDD)的多层印制电路板，

这样能提供好的耦合性能和屏蔽效果。很多应用中，受经济条件限制不能使用这样的印制电路板，那么就需要保证一个好的接地和供电的结构。

6.6.2. 器件位置

为了减少PCB上的交叉耦合，设计版图时就需要根据各自对EMI影响的不同，而把不同的电路分开。比如，大电流电路、低电压电路以及数字器件等。

6.6.3. 接地和供电(VSS, VDD)

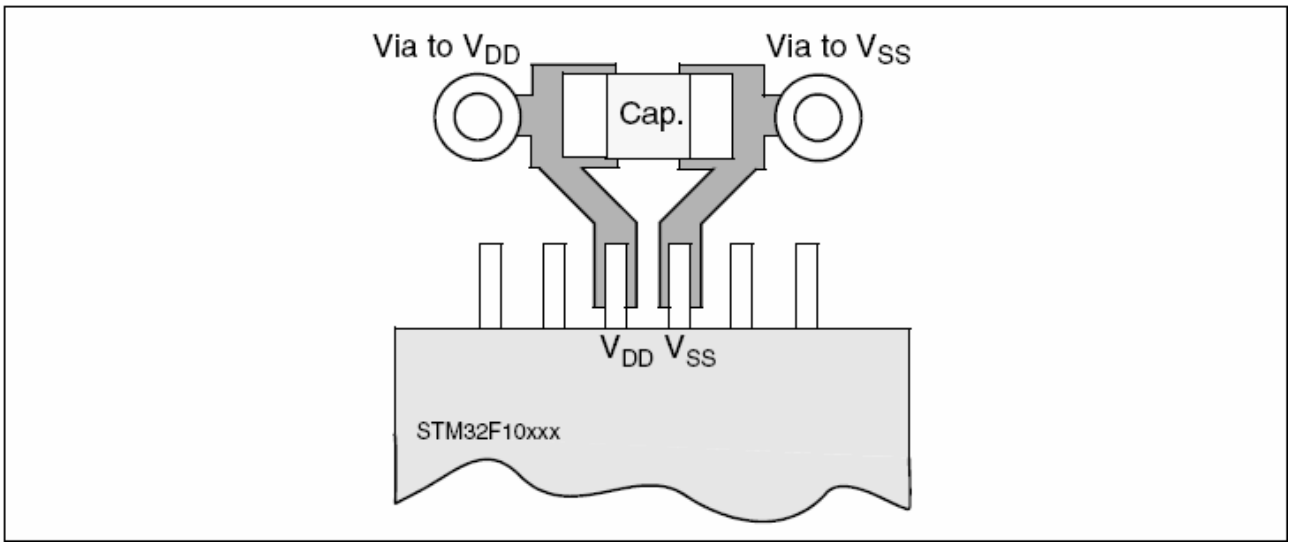
每个模块(噪声电路、敏感度低的电路、数字电路)都应该单独接地，所有的地最终都应在一个点上连到一起。尽量避免或者减小回路的区域。为了减少供电回路的区域，电源应该尽量靠近地线，这是因为，供电回路就像个天线，成为EMI的发射器和接收器。PCB上没有器件的区域，需要填充为地，以提供好的屏蔽效果(特别是对单层PCB，尤其如此)。

6.6.4. 去耦合

所有的引脚都需要适当地连接到电源。这些连接，包括焊盘、连线和过孔应该具备尽量小的阻抗。通常采用增加连线宽度的办法，包括在多层PCB中使用单独的供电层。

同时，STM3210xxx上每个电源引脚应该并联去耦合的滤波陶瓷电容C(100nF)和化学电容C(10μF)。这些电容应该尽可能的靠近电源/地引脚；或者在PCB另一层，处于电源/地引脚之下。典型值一般从10nF到100nF，具体的容值取决于实际应用的需要。图14显示了这样的电源/地引脚的典型布局。

VDD/VSS引脚的典型布局：



6.6.5. 其它信号

实际应用中，关注以下几点可以提高EMC性能：

- 那些受暂时的干扰会影响运行结果的信号(比如中断或者握手抖动信号，而不是LED命令之类的信号)。 对于这些信号，信号线周围铺地，缩短走线距离，消除邻近的噪声和敏感的连线都可以提高EMC性能。 对于数字信号，为有效地区别2种逻辑状态，必须能够达到最佳可能的信号特性余量(译注： 尽可能抬高逻辑’1’的高电平，拉低逻辑’0’的低电平)。推荐使用慢速施密特触发器来消除寄生状态。
- 噪声信号(时钟等)。
- 敏感信号(高阻等)。

6.6.6. 未用到的I/O及其特性

所有微控制器都为各种应用而设计，而通常的应用都不会用到所有的微控制器资源。

为了提高EMC性能，不用的时钟、计数器或者I/O管脚，需要做相应处理，比如，I/O端口应该被设置为’0’或’1’(对不用到的I/O引脚上拉或者下拉)；没有用到的模块应该禁止或者“冻结”。

7.更新记录

版本	更改说明	作者	发布日期
1.0	初版	armfly	2009-10-01
1.1	1、更正原理框图中的错误 2、修改“烧写演示Demo程序”一节	armfly	2009-10-16
2.0	1、更正JTAG调试接口描述。 2、硬件升级到REV 2.0，相关内容进行调整。	armfly	2009-11-07
2.1	1、更新基础例程列表。 2、更新预装Demo程序打印信息。 3、完善学习建议（添加视频教程） 4、更正8.2，第1版和第2板LED指示灯的控制极性是相同的，和ST官方是相反的。	armfly	2009-12-16
2.2	1、更新预装程序的测试方法。	armfly	2010-04-06
2.3	1、修正TFT显示接口信号定义的笔误（3.21节）	armfly	2010-07-18
2.4	1、修改手册封面 2、修改“检查硬件功能”章节	armfly	2010-08-18
2.5	1、文档更名，目录结构调整 2、增加很多硬件相关的内容	armfly	2012-05-02