基于Visual Studio 2012开发RT-Thread项目

（一）Visual Studio中VisualGDB环境的搭建

Before we get start

确认你的电脑已经安装了：

- Visual Studio 2012
- VisualGDB 4.1 or later

在我们接下来的步骤，你还将安装：

- GCC compiler for ARM
- GDB debugger for ARM
- JTAG driver for ARM

当然，我们的工程项目系统平台是基于

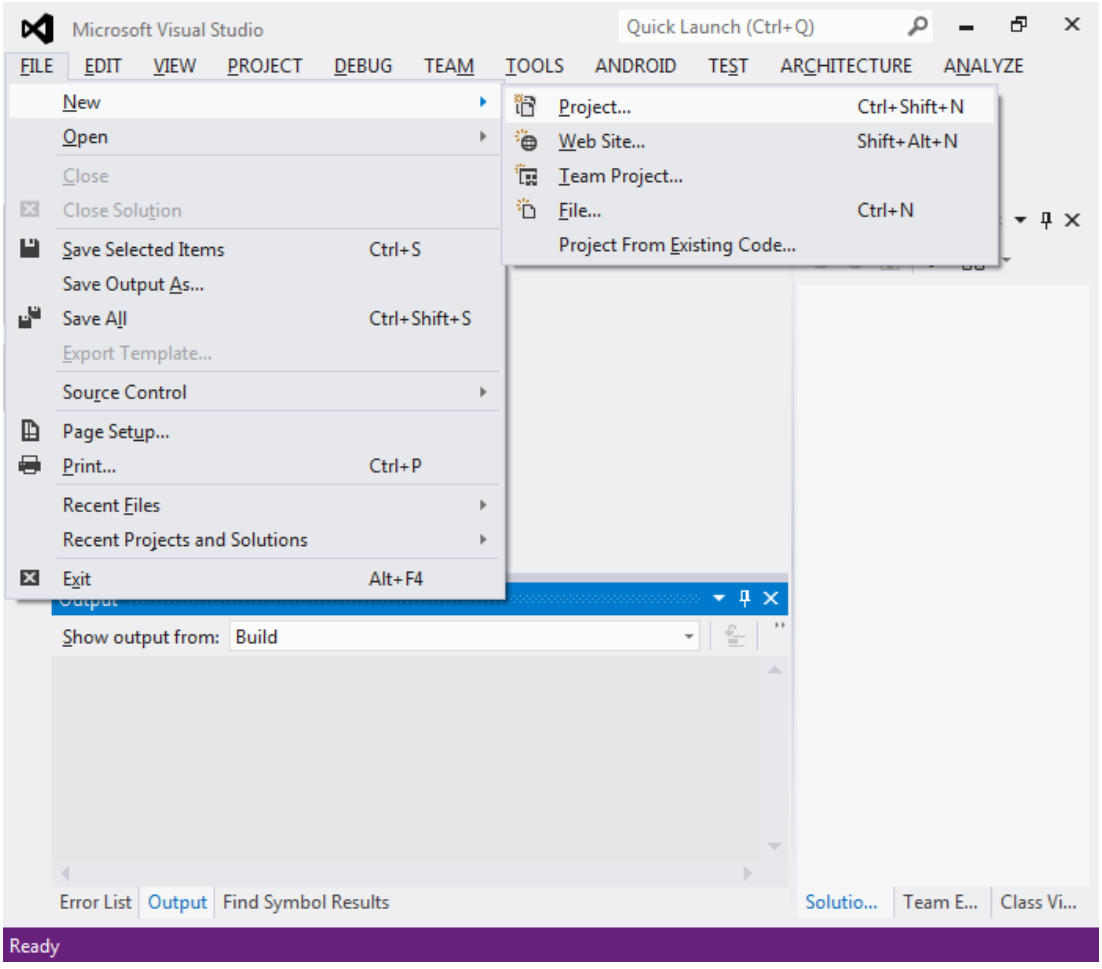
- RT-Thread 1.2.2

而硬件平台是基于

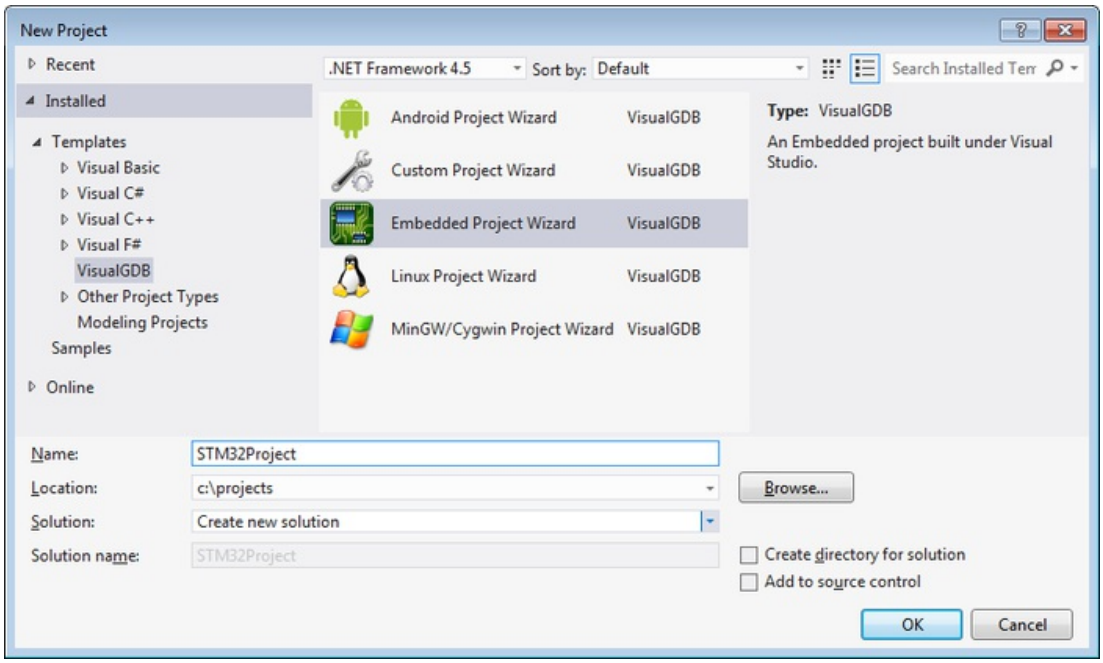
- STM32F407VE

What do we waiting for

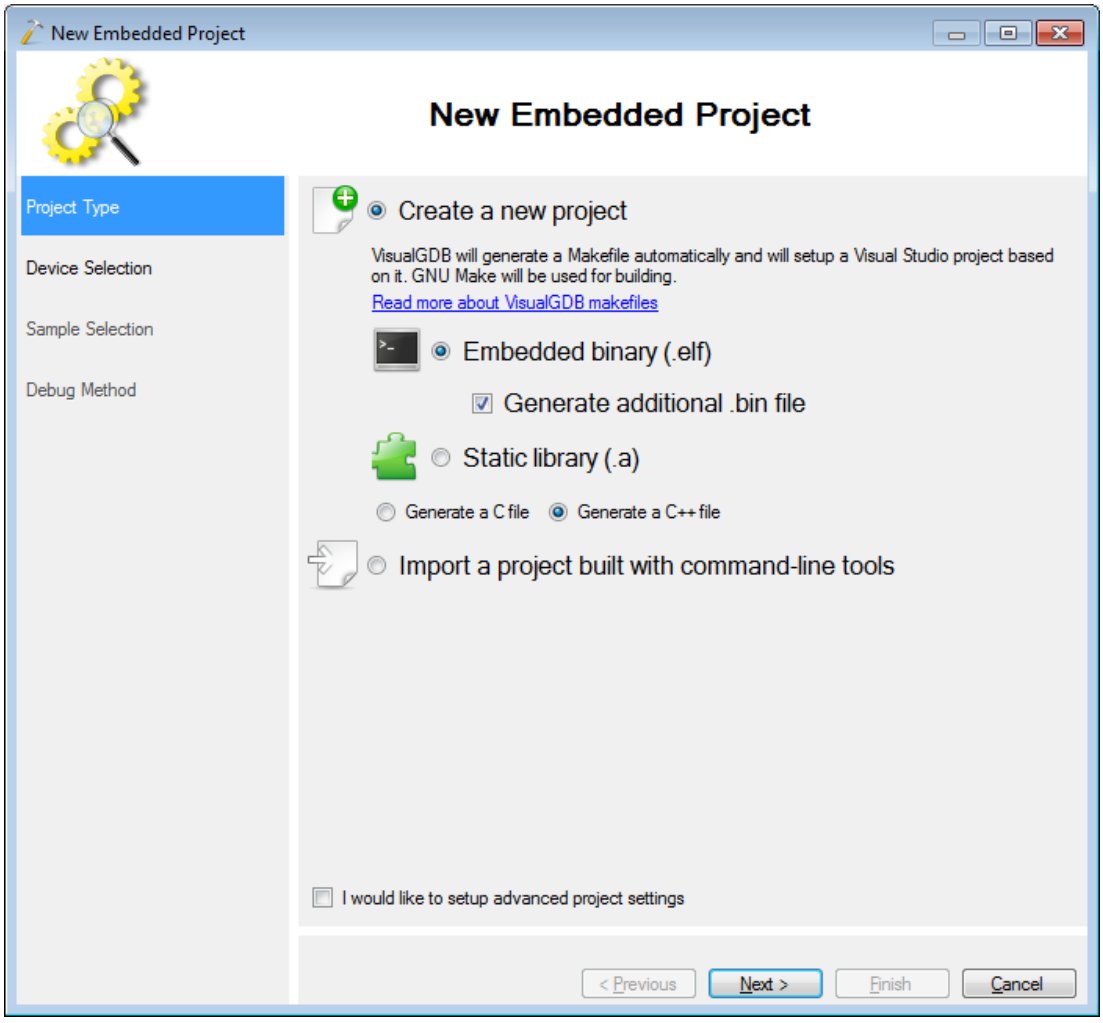
1. 下面开始新建工程：



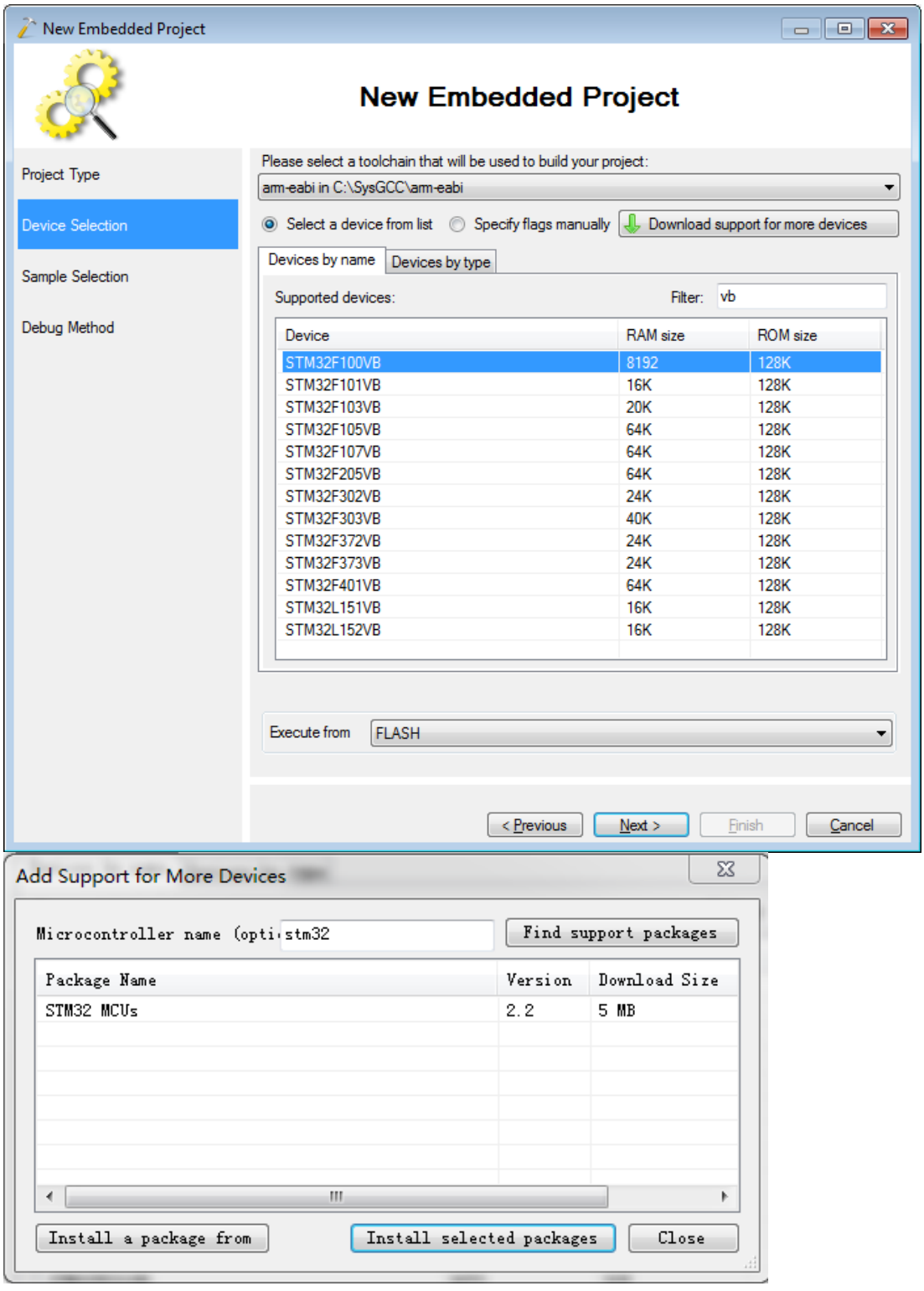
2. 然后选择“VisualGDB -> Embedded Project Wizard”，并给项目选择好文件夹和名字



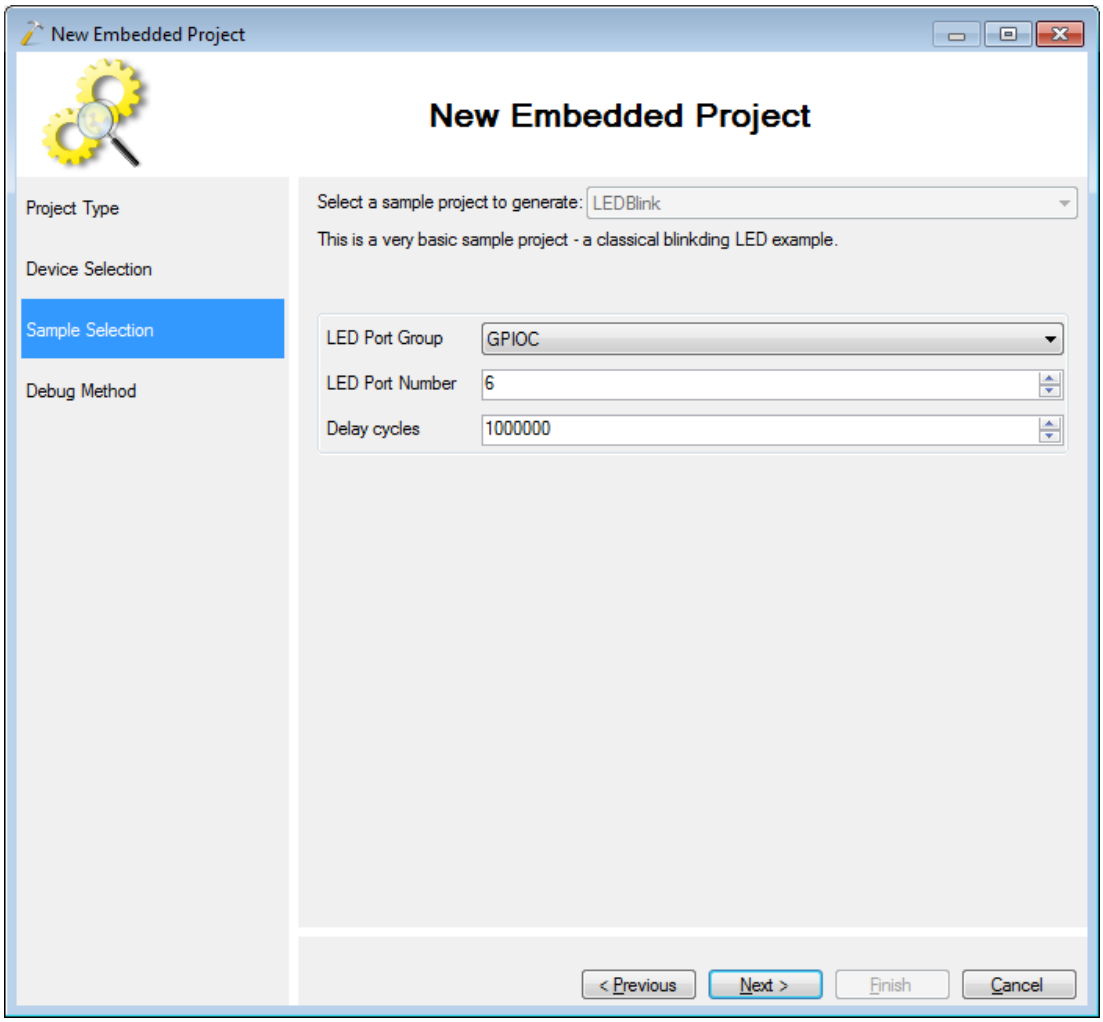
3. 选择新建工程，并选择C语言工程（图中的是**C++的工程**）



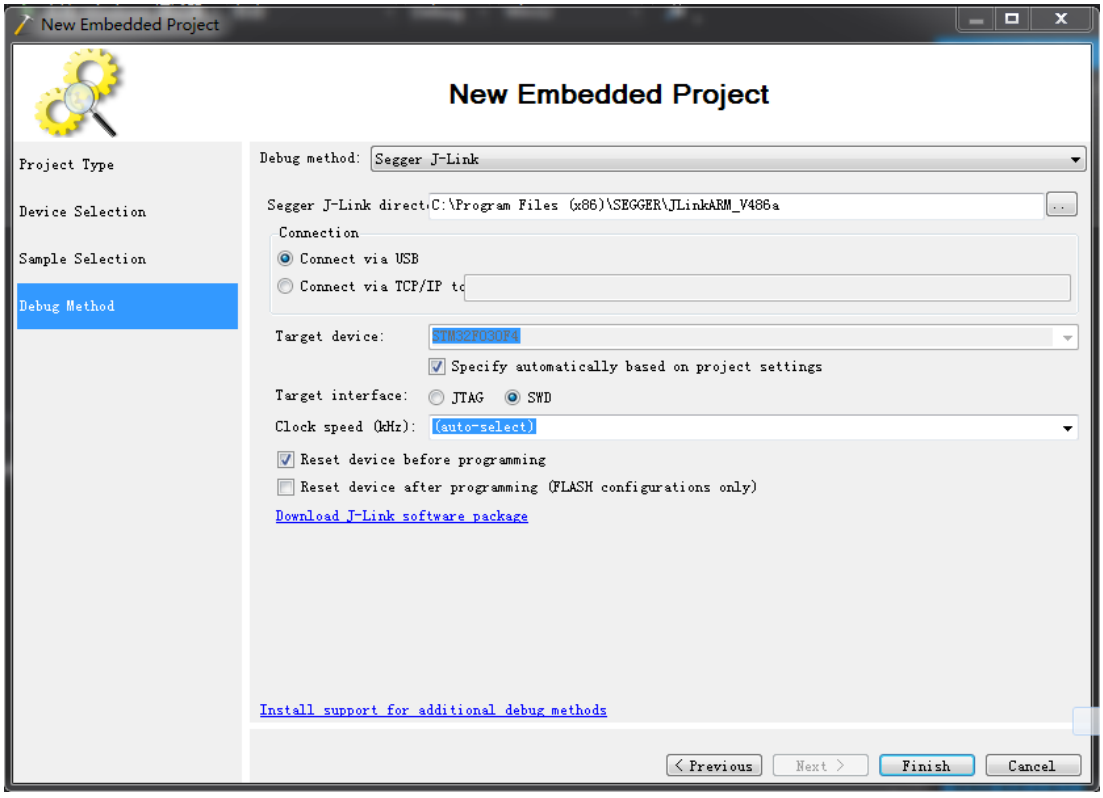
4. 下载交叉编译工具，选择"Download support for more devices"，并如图查找stm32的支持包，选择"STM32 MCUs"下载安装，之后，你就可以如下图选择CPU型号了



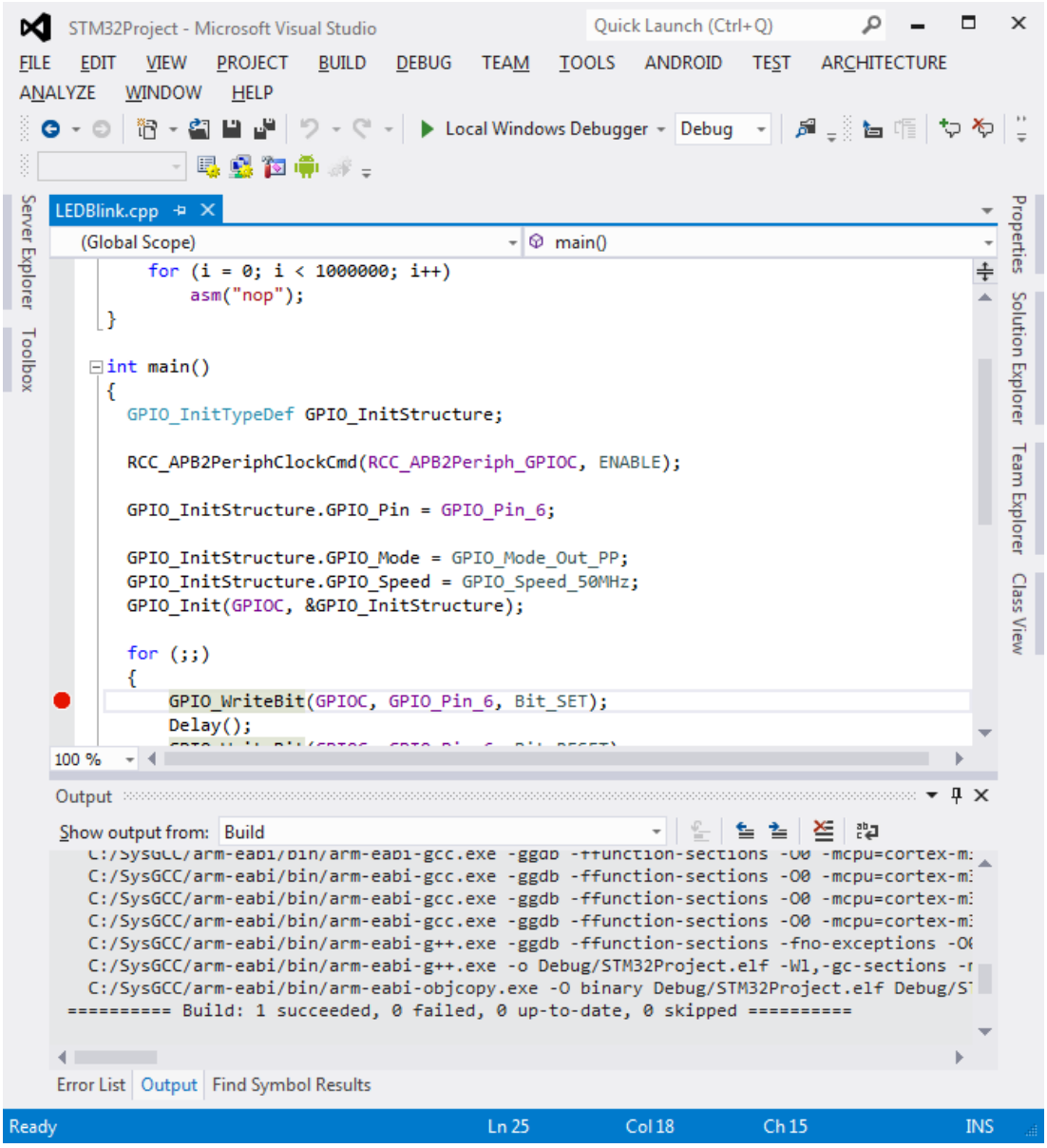
5. 然后下一步里面，VisualGDB会自动添加一个点灯的程序，这里可以和你的硬件对应起来，如果不想修改直接点下一步



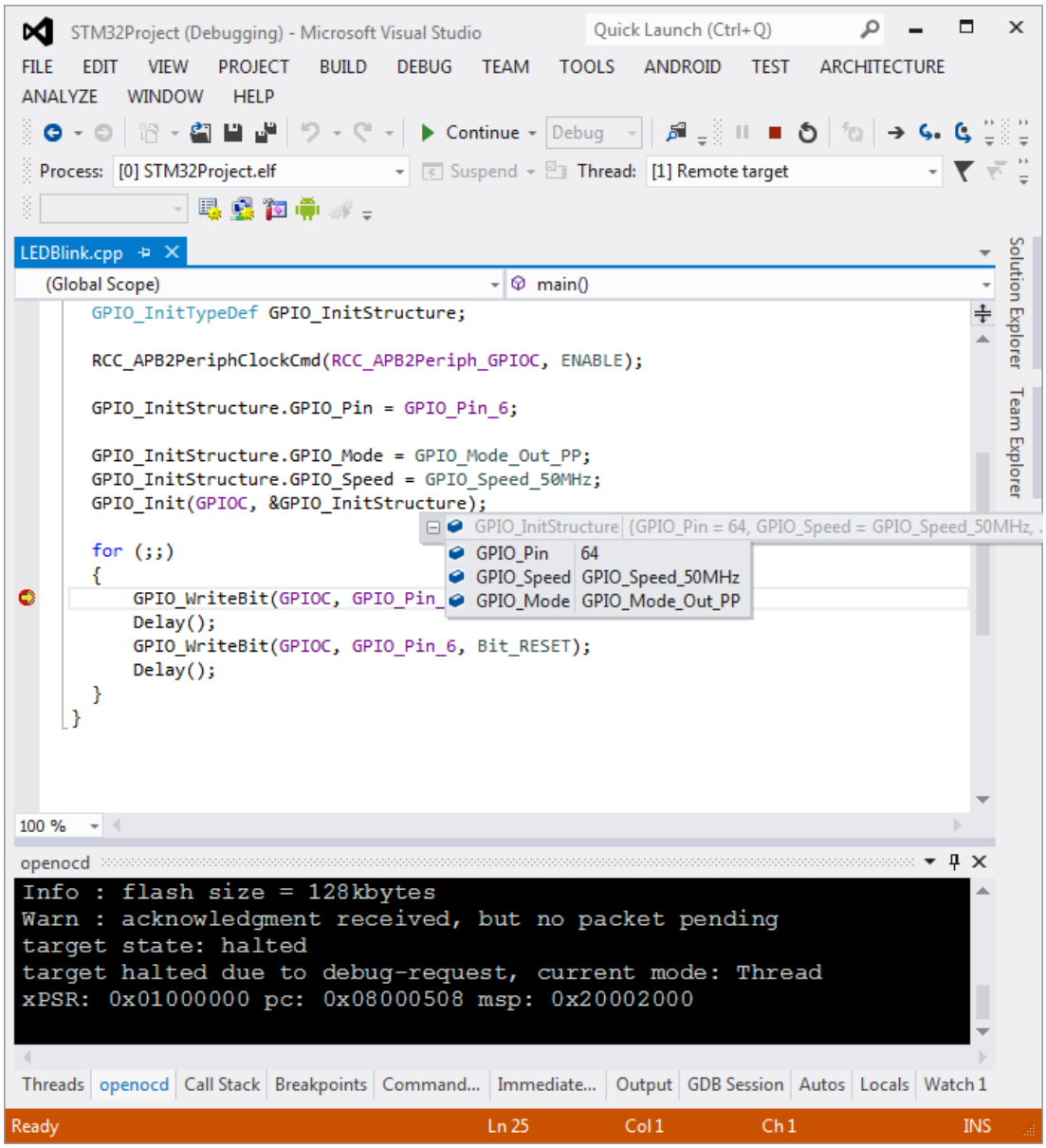
6. 选择编译器，这里一开始没有JTAG驱动，没关系，我们下载一个，下面三个按钮的上面有一个链接"Install support for additional debug methods"，点这个选择Segger J-Link下载，然后按下图配置即可（用的SWD）



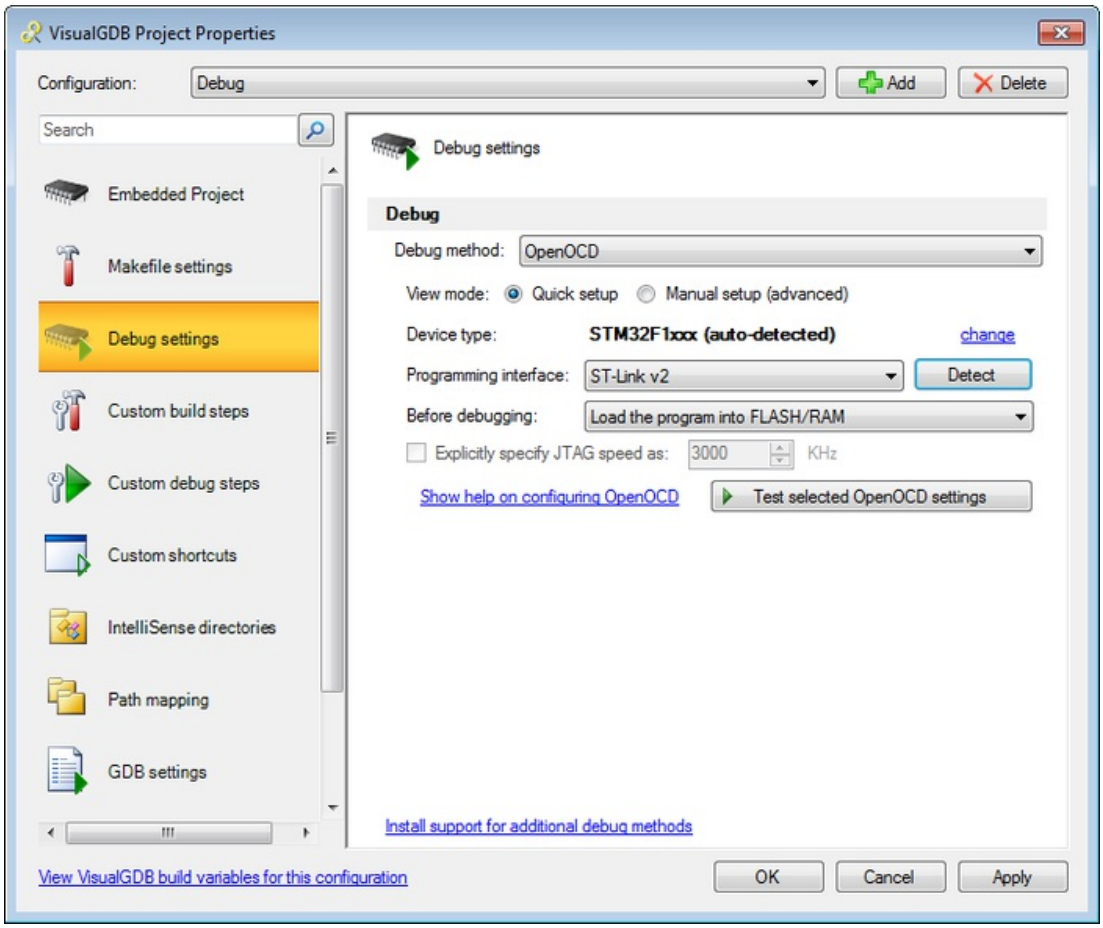
7. 点Finish结束设置，开始工程



8. 此时，如果你设置的点灯引脚和你的硬件对应，那么你可以点F7编译，F5调试，设置断点后可以按F10单步调试，并可以看到硬件led闪动



9. 你还可以右键工程文件名，查看并设置VisualGDB项目的各项属性



（二）替换其他版本的固件库文件进Visual Studio 2012中

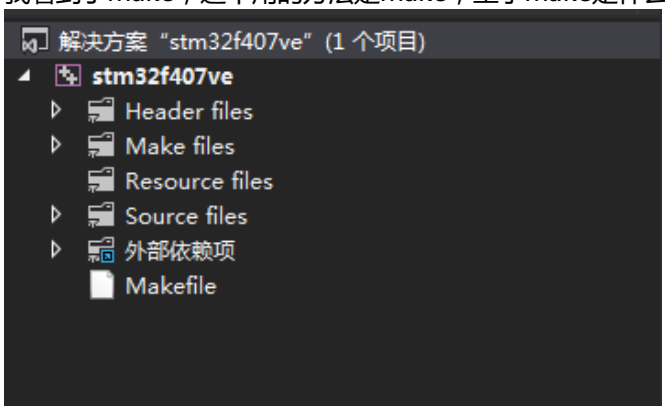
如果你裸奔

如果你不要操作系统，那么此刻你已经可以开始工作了，只是ST的固件库是最新的了，都是基于硬件抽象层做的（HAL），你调用的函数名可能有改变，如果你不想改变，我们就来改造一下工程

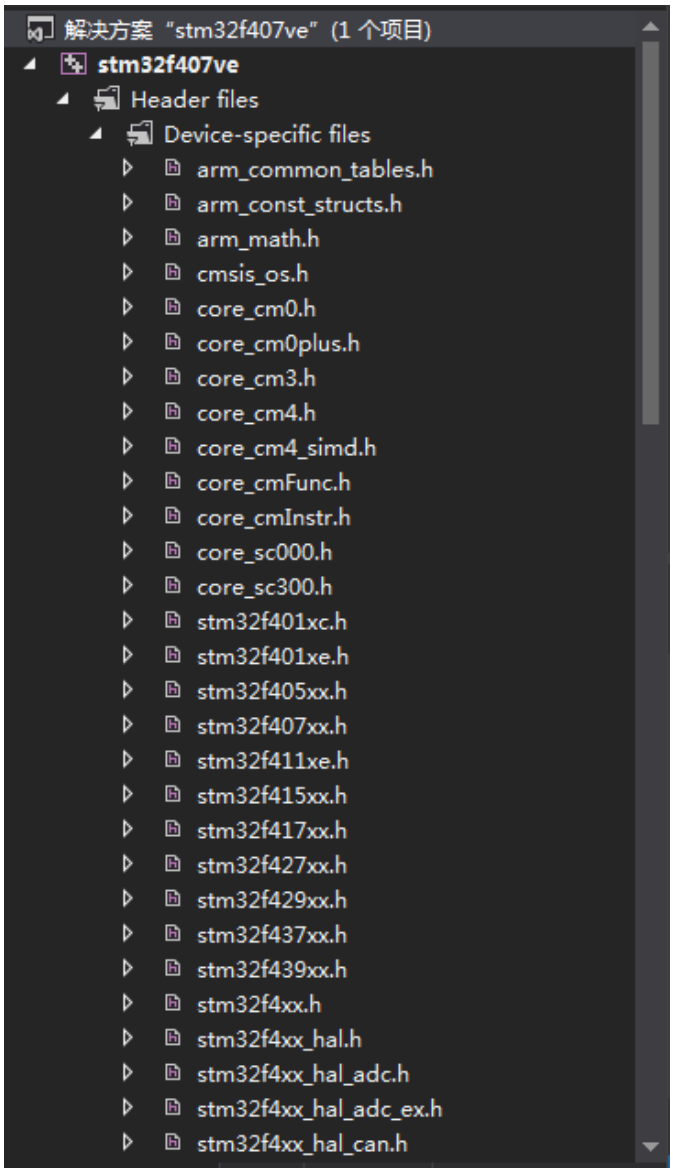
There is nothing about RTOS

不管你要不要操作系统，你可能都需要替换固件库文件
所谓"***知己知彼，百战不殆***"，我们先来熟悉下VisualGDB嵌入式项目的工程结构

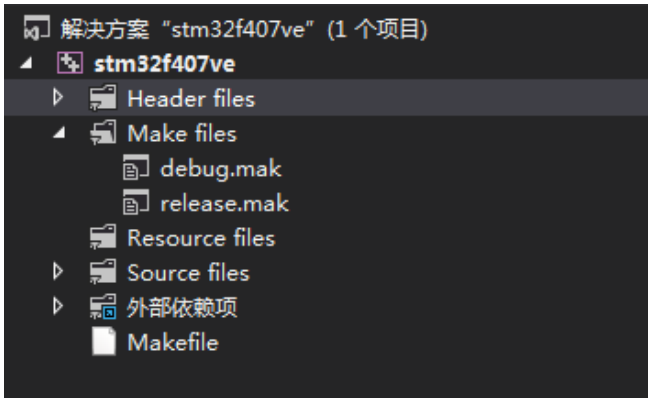
1. 我看到了make，这个用的方法是make，至于make是什么，请自行百度



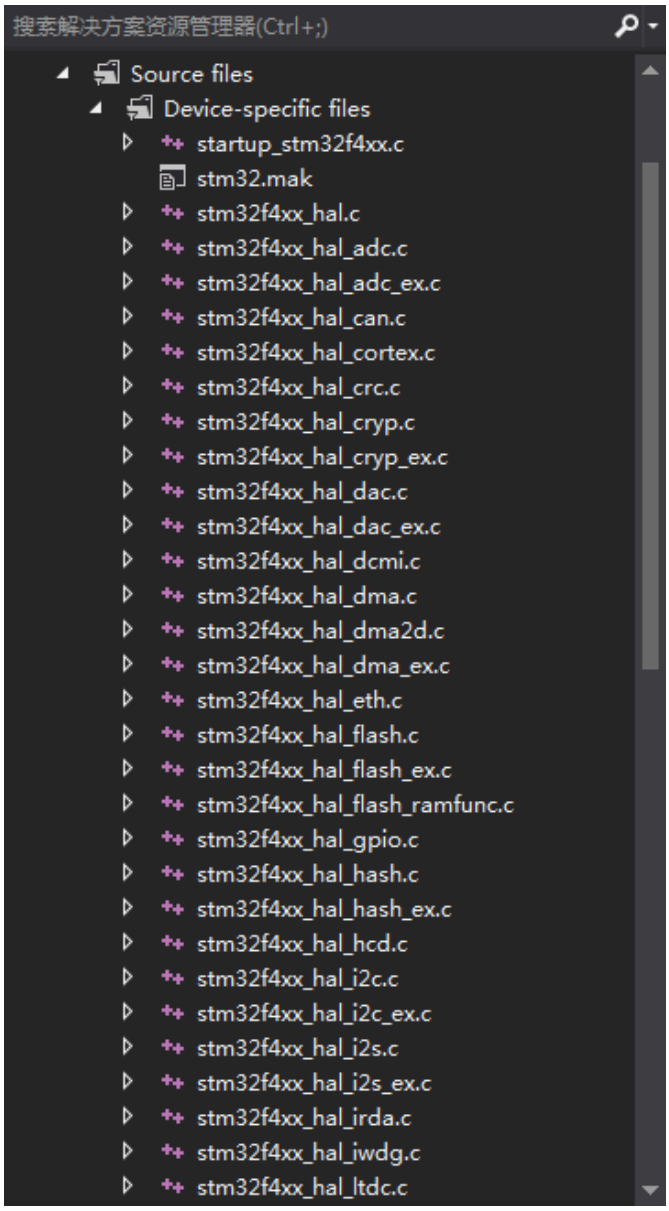
2. 挨个看之Header files——>Device-specific files，头文件，里面有好多



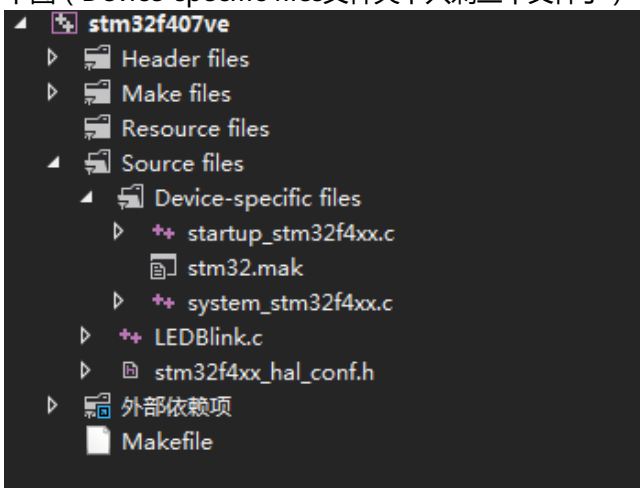
3. 挨个看之Make files：发现两个.mak文件，这个就是make程序的配置文件



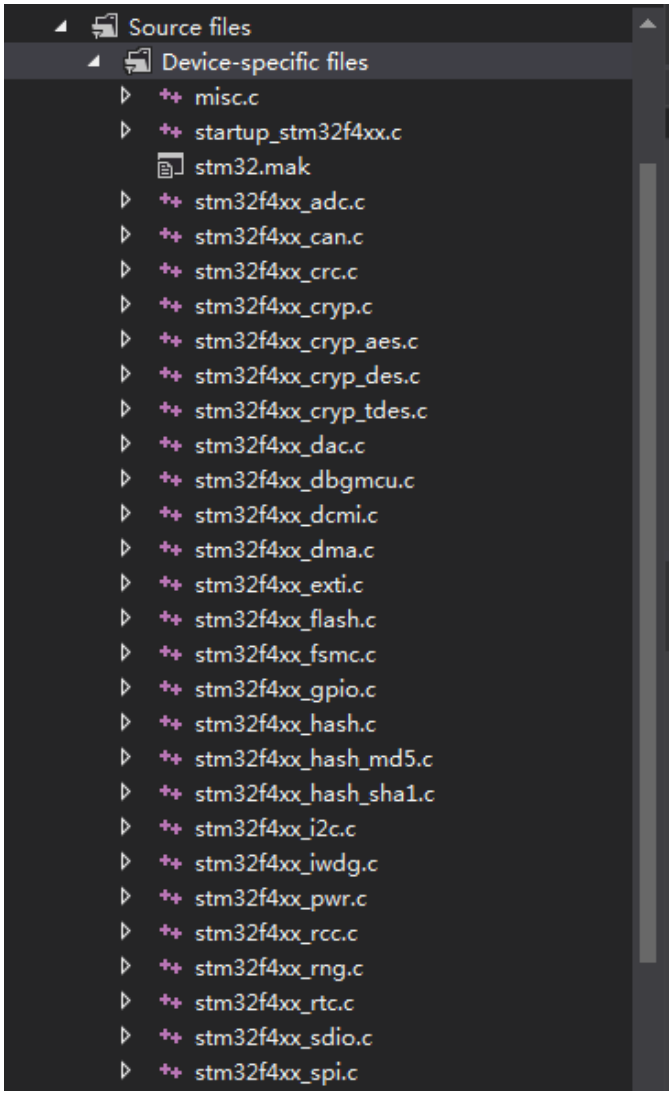
4. 挨个看之Source files——>Device-specific files，发现库函数都带hal了，里面的函数也改变了好多，这就是为什么我们要替换，因为新的固件库是基于硬件抽象层概念建立的，效率不如原先，而且改动较大，从软件转过来的人可能更喜欢这个，而对于做硬件的，可能更习惯原来的固件库，甚至不需要固件库（膜拜ing...Orz），当然，这里还有一个stm32.mak，这个文件很重要



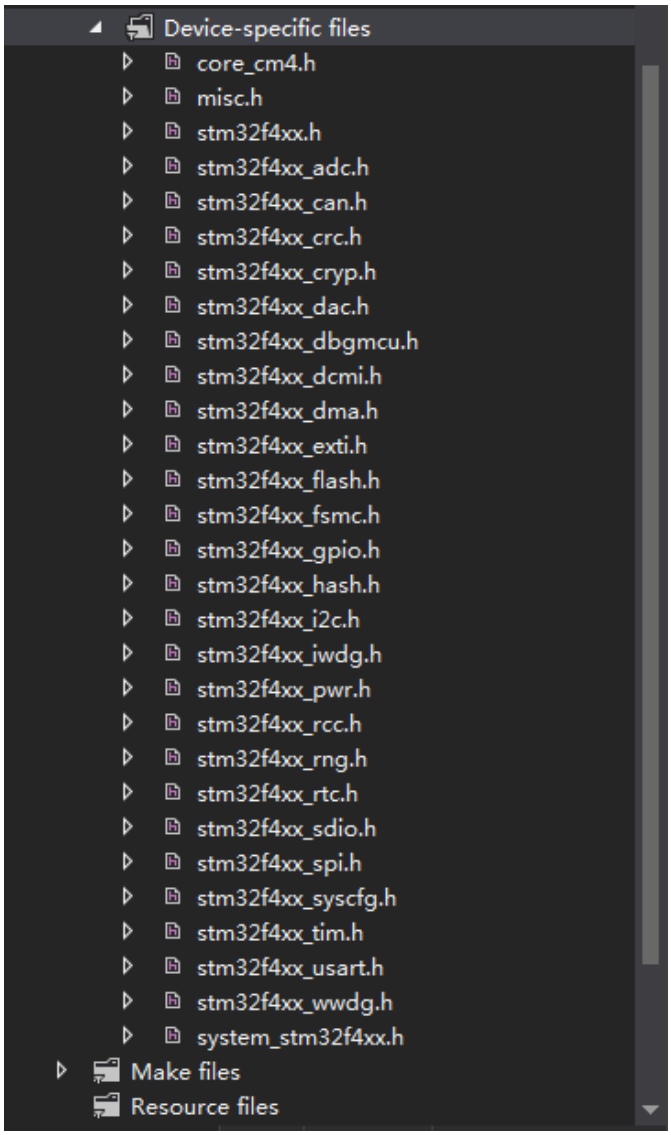
5. 其他的诸如：外部依赖项、makefile什么的都是自动生成的，可以不用管，我们开始替换固件库文件
   宗旨就是：***该删的删，不该删的别删***
   - 先删源程序里面的固件库文件，第一步先删除外设固件库文件（adc、gpio那些），删后如下图（Device-specific files文件夹下只剩三个文件了）



- 然后把刚才删除的文件对应的旧固件库文件添加进来，先都添加进来吧，加完了如下图



- startup_stm32f4xx.c压根没找到，直接删了，还有就是把system_stm32f4xx.c替换成老固件库版本，这步没有图
- 接着删除LEDBlink.c这个自带的C程序
- 替换stm32f4xx_hal_conf.h为stm32f4xx_conf.h
- 对，你说对了，stm32.mak别动~~~
- 下面如法炮制，删除Header files，替换后的如下图，所有文件都替换哦！



6. 弄完了，但是一编译，就说找不到好多文件，这个是因为include目录默认在C盘里，需要我们更改一下include目录，在哪改？stm32.mak还记得吗？

> #This file is generated by VisualGDB.
>
> #It contains GCC settings automatically derived from the board support package (BSP).
>
> #DO NOT EDIT MANUALLY. THE FILE WILL BE OVERWRITTEN.
>
> #Use VisualGDB Project Properties dialog or modify Makefile or per-configuration .mak files instead.
>
> #VisualGDB provides BSP_ROOT via environment when running Make. The line below will only be active if GNU Make is started manually.
>
> BSP_ROOT ?= C:/Users/lujun/AppData/Local/VisualGDB/EmbeddedBSPs/arm-eabi/com.sysprogs.arm.stm32
>
> TOOLCHAIN_ROOT := C:/SysGCC/arm-eabi
>
> #Embedded toolchain
>
> CC:=$(TOOLCHAIN_ROOT)/bin/arm-eabi-gcc.exe
>
> CXX := $(TOOLCHAIN_ROOT)/bin/arm-eabi-g++.exe

LD := $(CXX)

AR := $(TOOLCHAIN_ROOT)/bin/arm-eabi-ar.exe

OBJCOPY := $(TOOLCHAIN_ROOT)/bin/arm-eabi-objcopy.exe

#Additional flags

PREPROCESSOR_MACROS += ARM_MATH_CM4 stm32_flash_layout STM32F407VE STM32F407xx

INCLUDE_DIRS += . $(BSP_ROOT)/STM32F4xxxx-HAL/CMSIS/Device/ST/STM32F4xx/Include

$(BSP_ROOT)/STM32F4xxxx-HAL/CMSIS/Include

$(BSP_ROOT)/STM32F4xxxx-HAL/CMSIS/RTOS

$(BSP_ROOT)/STM32F4xxxx-HAL/STM32F4xx_HAL_Driver/Inc

LIBRARY_DIRS +=

LIBRARY_NAMES +=

ADDITIONAL_LINKER_INPUTS +=

MACOS_FRAMEWORKS +=

CFLAGS += -mcpu=cortex-m4 -mthumb

CXXFLAGS += -mcpu=cortex-m4 -mthumb

ASFLAGS += -mcpu=cortex-m4 -mthumb

LDFLAGS += -mcpu=cortex-m4 -mthumb -T$(BSP_ROOT)/STM32F4xxxx-HAL/LinkerScripts/STM32F407xE_flash.lds

COMMONFLAGS += -mfloat-abi=soft

改为：

#This file is generated by VisualGDB.

#It contains GCC settings automatically derived from the board support package (BSP).

#DO NOT EDIT MANUALLY. THE FILE WILL BE OVERWRITTEN.

#Use VisualGDB Project Properties dialog or modify Makefile or per-configuration .mak files instead.

#VisualGDB provides BSP_ROOT via environment when running Make. The line below will only be active if GNU Make is started manually.

#BSP_ROOT ?= C:/Users/lujun/AppData/Local/VisualGDB/EmbeddedBSPs/arm-eabi/com.sysprogs.arm.stm32

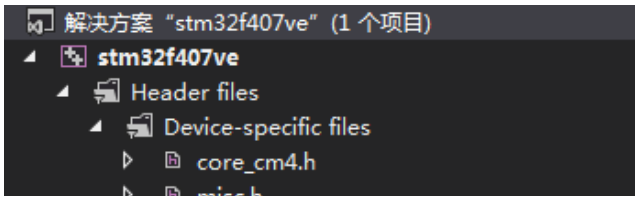
TOOLCHAIN_ROOT := C:/SysGCC/arm-eabi

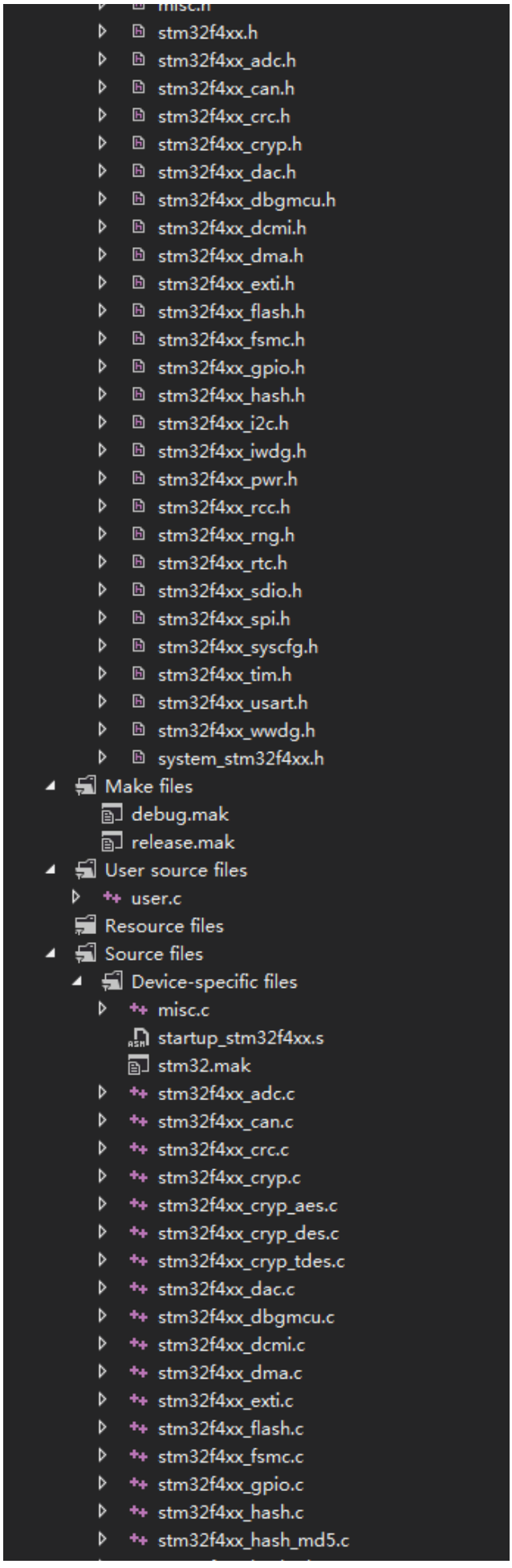
PROJ_ROOT:=$(ProjectDir)rt-thread-1.2.2

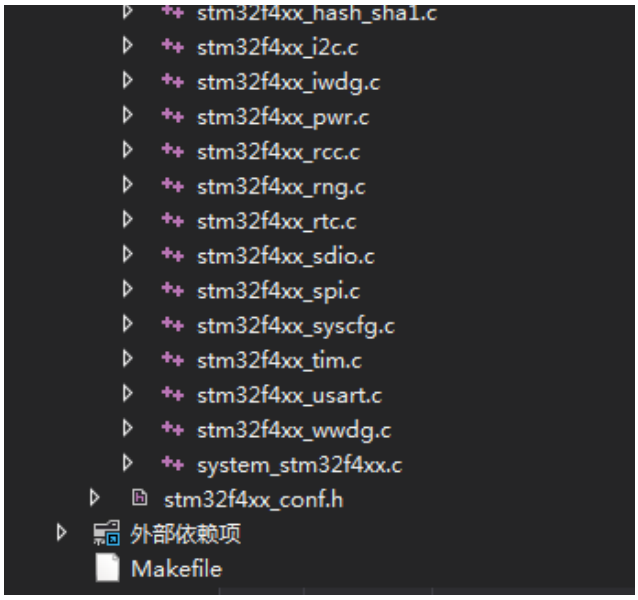
#Embedded toolchain

> CC:=$(TOOLCHAIN_ROOT)/bin/arm-eabi-gcc.exe
>
> CXX := $(TOOLCHAIN_ROOT)/bin/arm-eabi-g++.exe
>
> LD := $(CXX)
>
> AR := $(TOOLCHAIN_ROOT)/bin/arm-eabi-ar.exe
>
> OBJCOPY := $(TOOLCHAIN_ROOT)/bin/arm-eabi-objcopy.exe
>
> #Additional flags
>
> PREPROCESSOR_MACROS += ARM_MATH_CM4 stm32_flash_layout STM32F407VE STM32F407xx
>
> **INCLUDE_DIRS += . $(PROJ_ROOT)/bsp/stm32f40x**
>
> **$(PROJ_ROOT)/bsp/stm32f40x/Libraries/CMSIS/Include**
>
> **$(PROJ_ROOT)/bsp/stm32f40x/Libraries/CMSIS/ST/STM32F4xx/Include**
>
> **$(PROJ_ROOT)/bsp/stm32f40x/Libraries/STM32F4xx_StdPeriph_Driver/inc**
>
> **$(PROJ_ROOT)/include**
>
> **$(PROJ_ROOT)/bsp/stm32f40x/drivers**
>
> **$(PROJ_ROOT)/components/finsh**
>
> LIBRARY_DIRS +=
>
> LIBRARY_NAMES +=
>
> ADDITIONAL_LINKER_INPUTS +=
>
> MACOS_FRAMEWORKS +=
>
> CFLAGS += -mcpu=cortex-m4 -mthumb
>
> CXXFLAGS += -mcpu=cortex-m4 -mthumb
>
> ASFLAGS += -mcpu=cortex-m4 -mthumb
>
> **LDFLAGS += -mcpu=cortex-m4 -mthumb -T$(PROJ_ROOT)/bsp/stm32f40x/stm32_rom.ld**
>
> COMMONFLAGS += -mfloat-abi=soft

7. 然后编译竟然给通过，其实这里还少一个启动文件，你会发现输出里面提示没找到0x80000000，我们添加startup_stm32f4xx.s文件进来，一定要gcc_ride7文件夹下面的哦！
8. 这样添加后，编译会产生一个错误，因为项目没有main函数，添加一个C文件，然后随便弄个main()出来就OK了
9. 最后的工程list如下图，user.c里面含有main函数



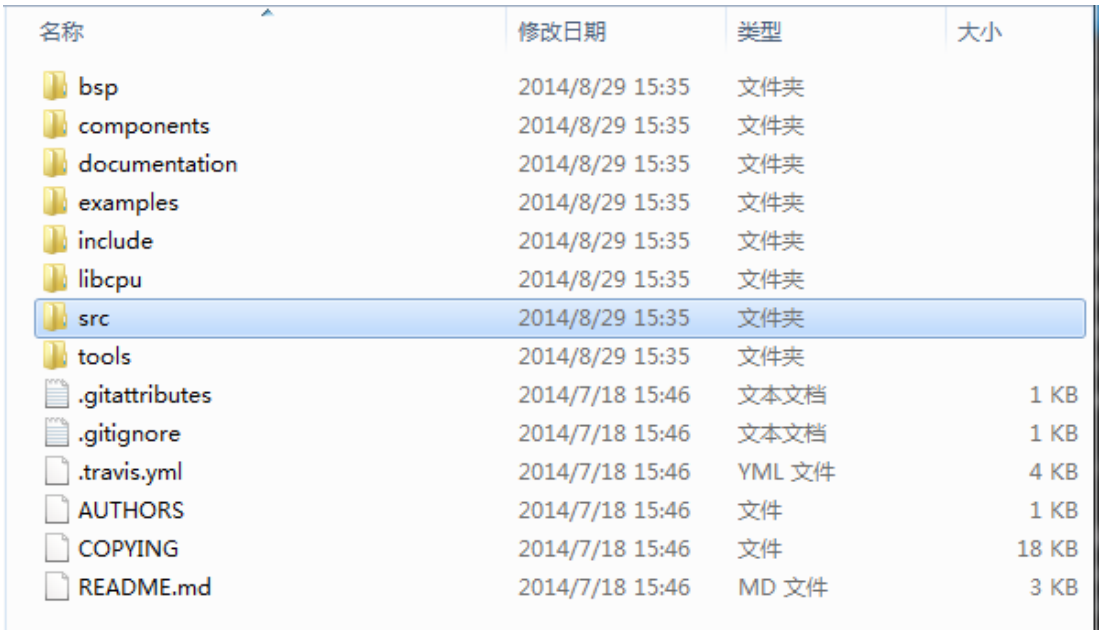

misc.h
stm32f4xx.h
stm32f4xx_adc.h
stm32f4xx_can.h
stm32f4xx_crc.h
stm32f4xx_cryp.h
stm32f4xx_dac.h
stm32f4xx_dbgmcu.h
stm32f4xx_dcmi.h
stm32f4xx_dma.h
stm32f4xx_exti.h
stm32f4xx_flash.h
stm32f4xx_fsmc.h
stm32f4xx_gpio.h
stm32f4xx_hash.h
stm32f4xx_i2c.h
stm32f4xx_iwdg.h
stm32f4xx_pwr.h
stm32f4xx_rcc.h
stm32f4xx_rng.h
stm32f4xx_rtc.h
stm32f4xx_sdio.h
stm32f4xx_spi.h
stm32f4xx_syscfg.h
stm32f4xx_tim.h
stm32f4xx_usart.h
stm32f4xx_wwdg.h
system_stm32f4xx.h
Make files
debug.mak
release.mak
User source files
user.c
Resource files
Source files
Device-specific files
misc.c
startup_stm32f4xx.s
stm32.mak
stm32f4xx_adc.c
stm32f4xx_can.c
stm32f4xx_crc.c
stm32f4xx_cryp.c
stm32f4xx_cryp_aes.c
stm32f4xx_cryp_des.c
stm32f4xx_cryp_tdes.c
stm32f4xx_dac.c
stm32f4xx_dbgmcu.c
stm32f4xx_dcmi.c
stm32f4xx_dma.c
stm32f4xx_exti.c
stm32f4xx_flash.c
stm32f4xx_fsmc.c
stm32f4xx_gpio.c
stm32f4xx_hash.c
stm32f4xx_hash_md5.c



（三）把RT-Thread系统放进Visual Studio 2012工程中

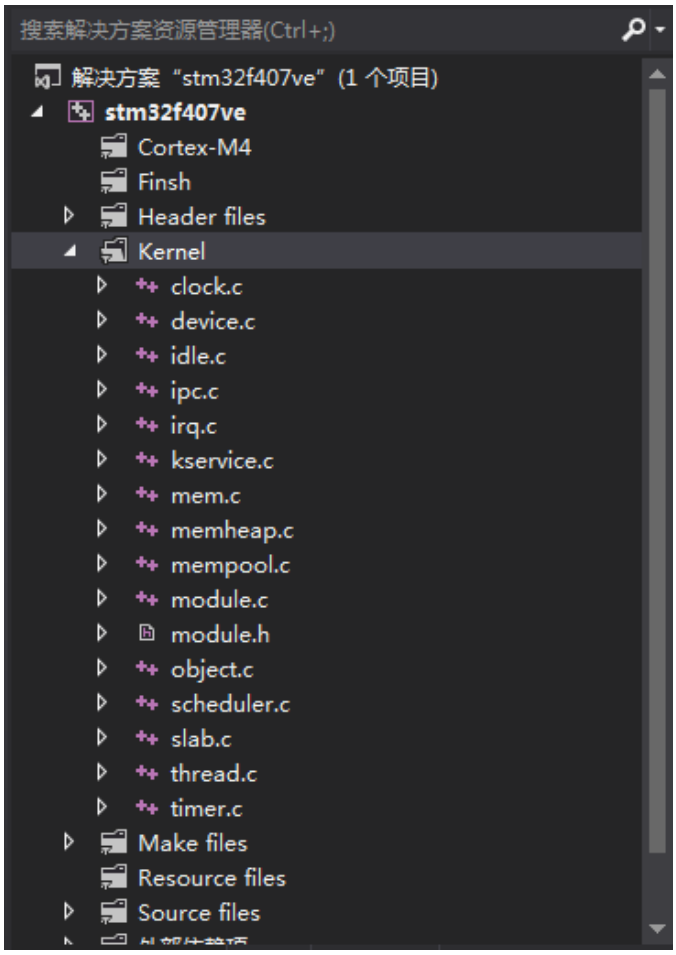
依据Keil的工程文件List，我们逐次添加RT-Thread系统文件进项目中

1. RT-Thread系统内核文件，在src文件夹里面

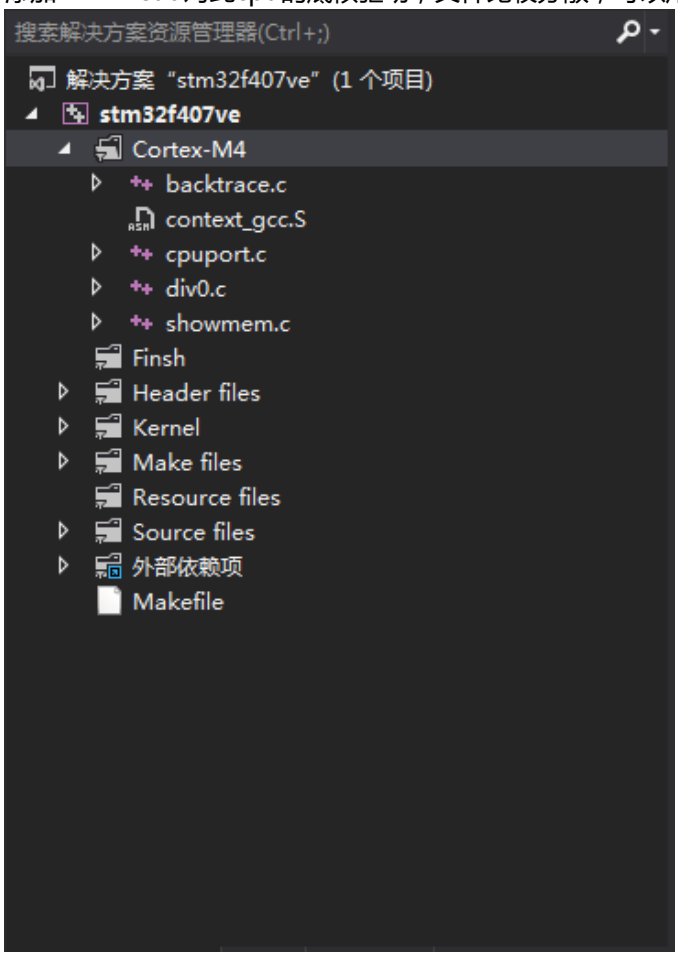


名称	修改日期	类型	大小
bsp	2014/8/29 15:35	文件夹	
components	2014/8/29 15:35	文件夹	
documentation	2014/8/29 15:35	文件夹	
examples	2014/8/29 15:35	文件夹	
include	2014/8/29 15:35	文件夹	
libcpu	2014/8/29 15:35	文件夹	
src	2014/8/29 15:35	文件夹	
tools	2014/8/29 15:35	文件夹	
.gitattributes	2014/7/18 15:46	文本文档	1 KB
.gitignore	2014/7/18 15:46	文本文档	1 KB
.travis.yml	2014/7/18 15:46	YML 文件	4 KB
AUTHORS	2014/7/18 15:46	文件	1 KB
COPYING	2014/7/18 15:46	文件	18 KB
README.md	2014/7/18 15:46	MD 文件	3 KB

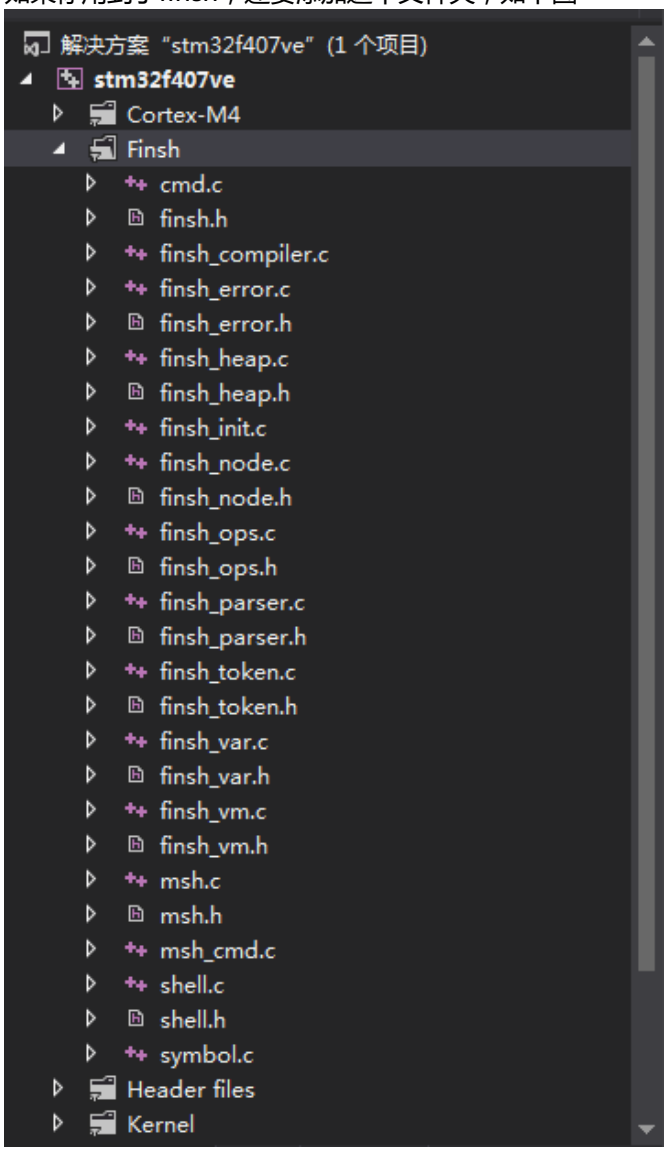
添加后如下图中的Kernel文件夹



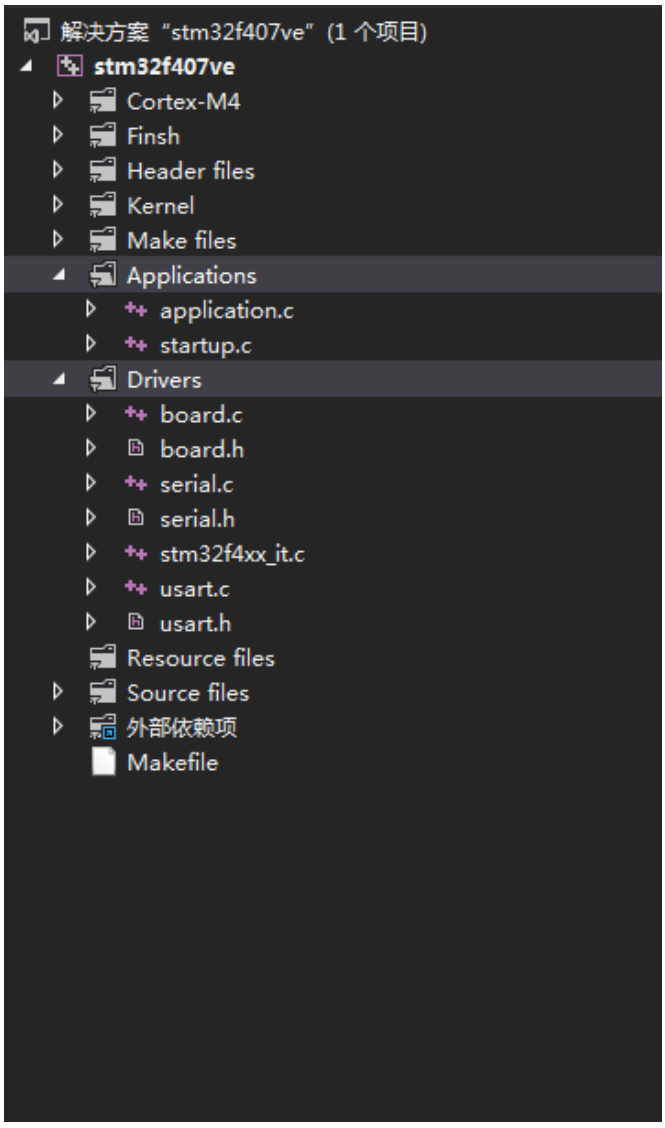
2. 添加RT-Thread对此cpu的底核驱动，文件比较分散，可以用查找功能添加，完成后如下图



3. 如果你用到了finsh，还要添加这个文件夹，如下图



4. 下面就是添加你自己写的程序文件了，比如默认的有applications和drivers两个目录，我们就按照这个来

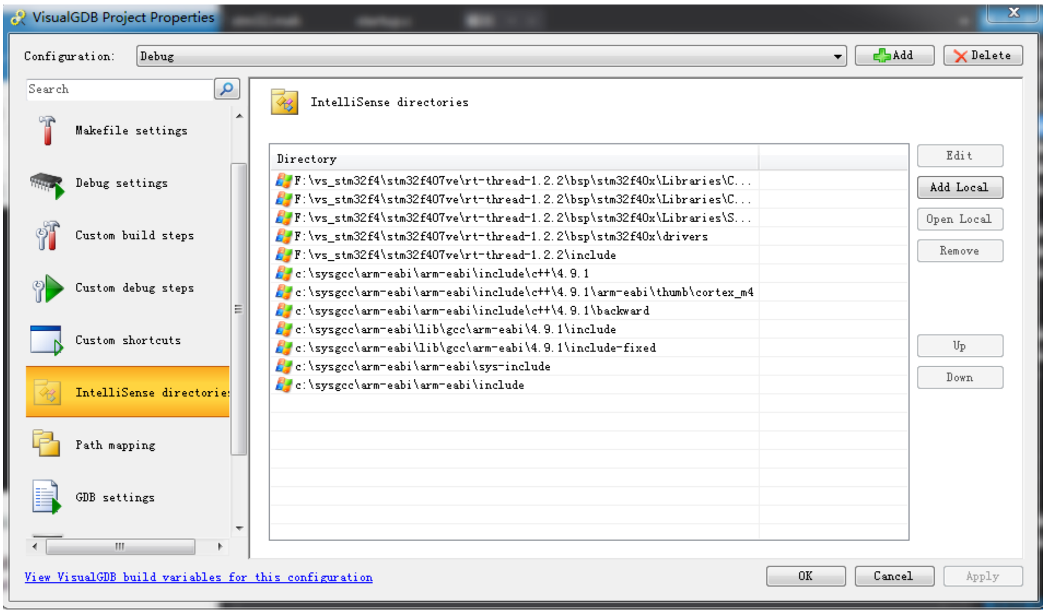


5. 路漫漫，其修远兮，此时认为已经完成了？还没！你一编译就发现好多库函数和结构体定义，甚至头文件都显示没找到？

   这是为什么呢？

   我们刚才添加的那些include路径，都是给make程序配置的，Visual Studio本身并不知道包含路径，因此，我们添加便是！

   - 右键工程名，然后选VisualGDB Project Properties
   - 在IntelliSense directories里面，把项目相关的.h文件所在的目录都添加进去，可以参考下图

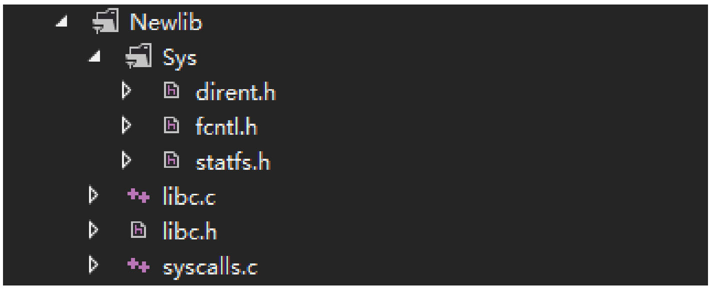


- 然后你就会发现#include下面的Error提示没有了

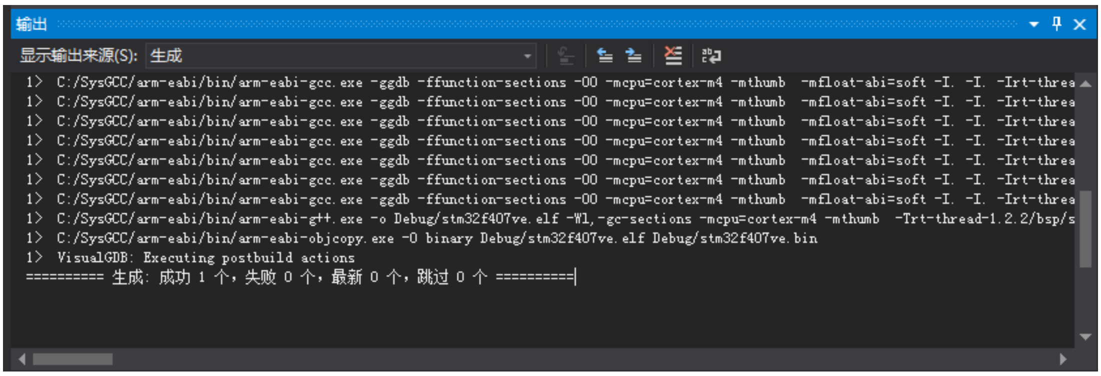
6. 接下来，你会发现好多库函数和结构体下面还是有Error，这一步我不知道是否有更好的解决办法，就是头文件的包含，工程结构和Keil不一样，所以，这里面好像没找到Keil的那种方法，只能给官方固件库文件里面依次添加stm32f4xx_conf.h头文件了（"那么多库函数都要加啊。。。给大爷跪了。。。Orz"，"没用的你现在可以删了，哦也~"），比如给stm32f4xx_gpio.c添加后，如下图

```
/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_conf.h"
```

7. 再编译，就还有一个错误，这个错误是RT-Thread1.2.x后独有的（我在RT-Thread1.1.0试的时候没有这个错误提示），就是gcc编译时，交叉编译工具链没有提供newlib库，好在RT-Thread提供了这个，包含进工程即可



8. 至此，编译应该没有问题了



9. 下面就是烧录和Debug了，这个请参考本文第一部分

> ## It is TIME

这样的话，就可以不用Keil了，虽然4.72a之后的版本有了代码补全功能，但是，跟本星球可能是最强的IDE相比，!!!Orz...

> ## 题外话
>
> 对于笔者来讲，折腾过Vim、Eclipse，甚至一度想在Xcode里面弄，其实Keil跟他们相比，最省心，最无痛地就可以开始工作了，工作本应如此
>
> 不应该太在意工具，不要流于表面功夫，更深层次的代码、业务逻辑、数据结构、算法、优化程度和资源分配等等，才真正地需要一个工程师的热情与坚持去完善！
>
> 与各位共勉！

本文作者：Sam.Lu@Tianjin*，有问题不想找到真正的答案请自行百度，想切实解决问题请上www.aol.com

*注：本文部分配图引自VisualGDB官网，部分文字翻译自VisualGDB官网：http://visualgdb.com/（Orz。。。非和谐版年费86美刀，年费。。。年。。。86x6。。。）