图文并茂教你在 keil 下搭建 RT-thread 最小系统工程

日期：2013-05-4　keilMDK 版本号：4.54

对于初次接触 RT-thread 的朋友来说，要想自己重新建立一个 keil 下的工程，可能会觉得不知所措，那么看到这篇文字，可能对你会有帮助。

我在这里演示了如何提取官方 bsp 包中 stm32 分支中的相关文件，重新组织文件结构，按照下图中的文件夹分配，重新生成 keil 下的工程，这个工程将会包括 RT-thread 的内核和 finsh 组件。

我愿意在开始前说明下分别建立这几个文件夹的作用：

project　——存放 MDK 工程文件；

RT-thread ——存放 rtt 源码包（放在最外层）；

apps　——存放我们自己（用户）写的一些应用代码；

drivers　——存放硬件外设驱动；

third_part ——存放第三方程序源码，比如 stm32 固件库、解码库等；

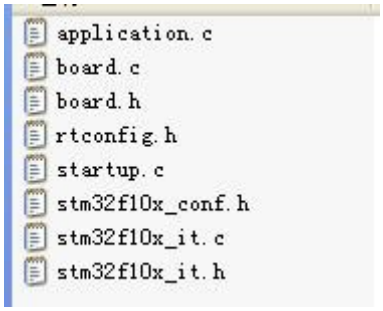
obj　——目标文件；

这么一来，各类代码分类一清二楚，好了，现在开始一步一步带大家走一遍生成这个工程的过程，当你明白后可按照自己的意愿去生成工程。

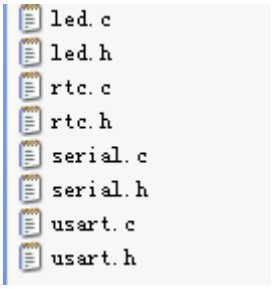
✧ 拷贝所需的文件

解压 RT-Thread 源码，将源码放在我们所建立的工程文件夹外面（这么放是方便以后的工程可以共用）

从源码 bsp\stm32f10x 目录 copy 下图所列出的文件，放入 篇 3-例程 1-重构 RTT 最小系统\apps 目录



从源码 bsp\stm32f10x 目录 copy 下图列出的必要的驱动文件，放入 篇 3-例程 1-重构 RTT 最小系统\drivers 目录

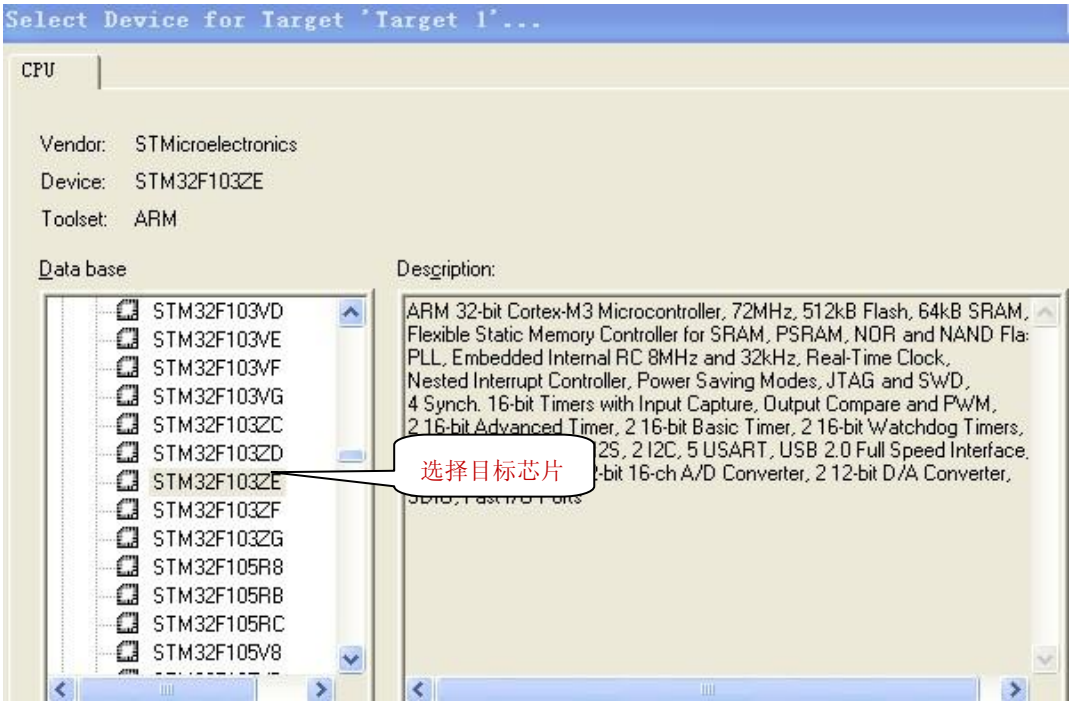


从源码 bsp\stm32f10x\Libraries 目录 copy CMSIS、STM32F10x_StdPeriph_Driver，两个文件夹放入 篇 3-例程 1-重构 RTT 最小系统\ third_part 目录

✧ 建立 MDK 工程

启动 MDK，新建一个工程，将工程保存路径设置到篇 3-例程 1-重构 RTT 最小系统\project 目录，并给工程命名。如下图所示：

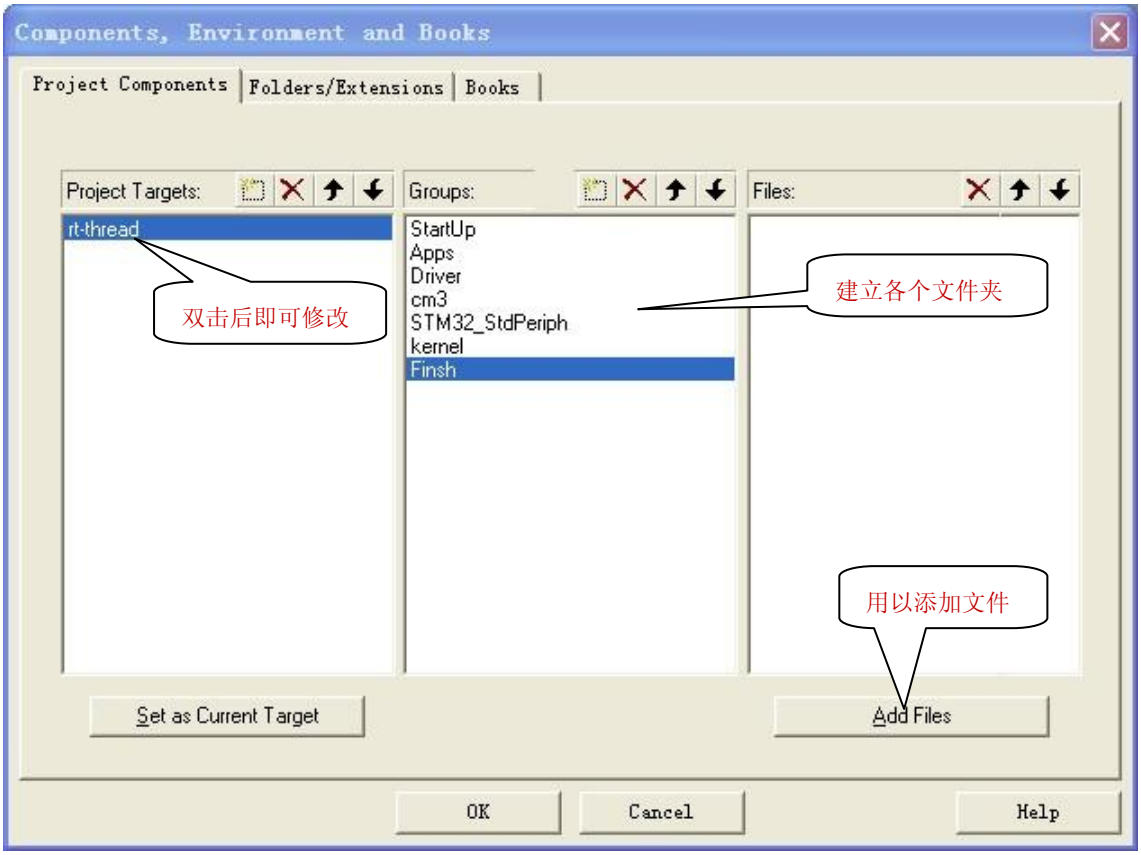
之后点击 保存 按钮，进入选择目标芯片界面，如下图：



选择以后，会跳出提示，询问我们是否需要加入 MDK 为我们准备好的启动文件，我们选择否，即不加入：

✧ 给工程加入所需文件

我们先修改下工程属性，并按模块建立几个工程文件夹，如下所示：



接下来，我们开始往各个文件夹中添加文件：

StartUp 中加入 apps、drivers 文件夹下所有.c 文件；

Apps、Driver 暂时不加入任何内容；

cm3 中加入 RT-Thread 源码下 libcpu\arm\common\ 下所有.c 文件，加入 libcpu\arm\ cortex-m3\下 cpuport.c 和 context_rvds.S 这两个文件；

STM32_StdPeriph 中加入 third_part\ STM32F10x_StdPeriph_Driver\src 下所有.c 文件，加入 third_part\CMSIS\CM3\CoreSupport\core_cm3.c

加入 third_part\CMSIS\DeviceSupport\ST\STM32F10x\system_stm32f10x.c

加入 third_part\CMSIS\DeviceSupport\ST\STM32F10x\ startup\arm\ startup_stm32f10x_hd.s

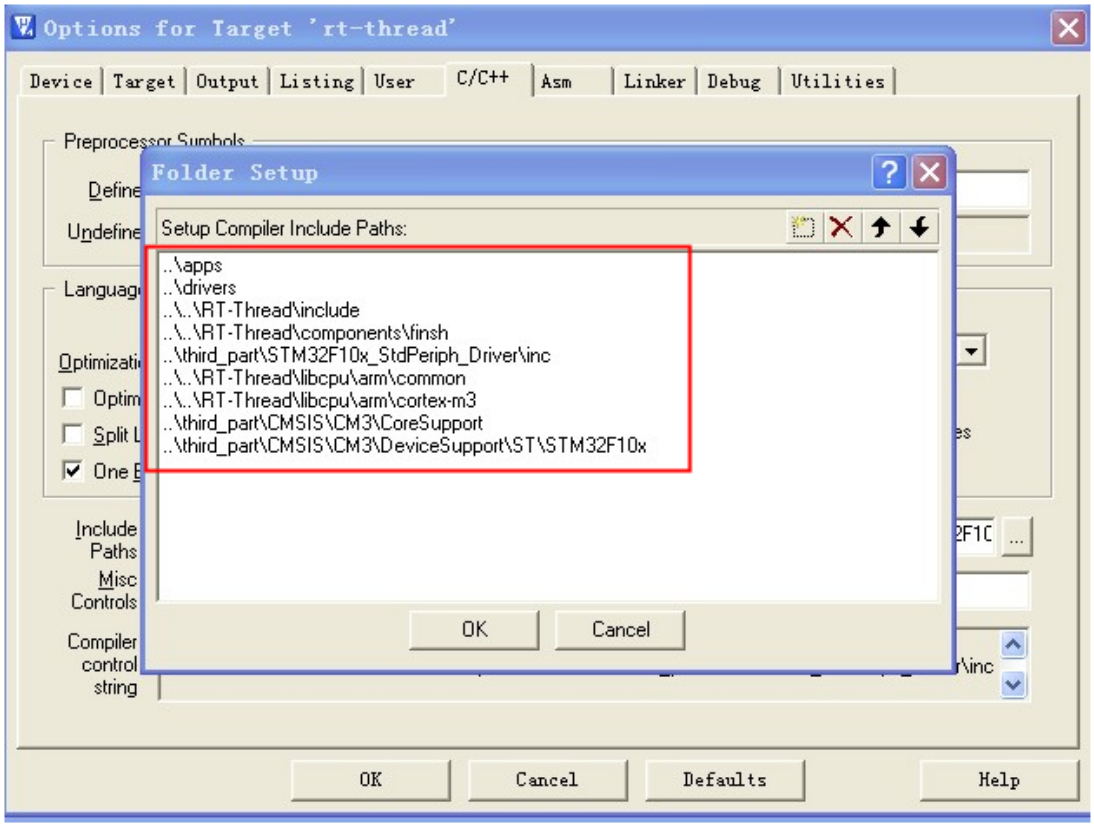
kernel 中加入 RT-thread 源码目录中 src 下所有.c 文件

Finsh 中加入 RT-thread 源码目录中 src\ components\ finsh 下所有.c 文件

至此文件添加完毕！

✧ 设置包含路径

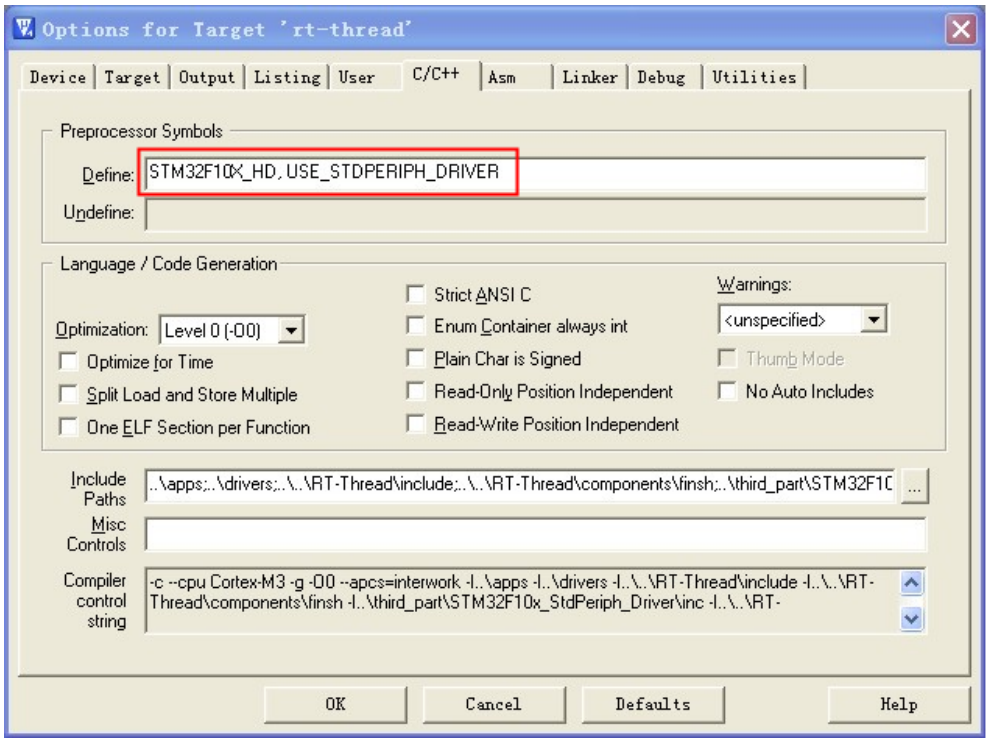
按照下图中所列的 include paths 添加文件包含列表



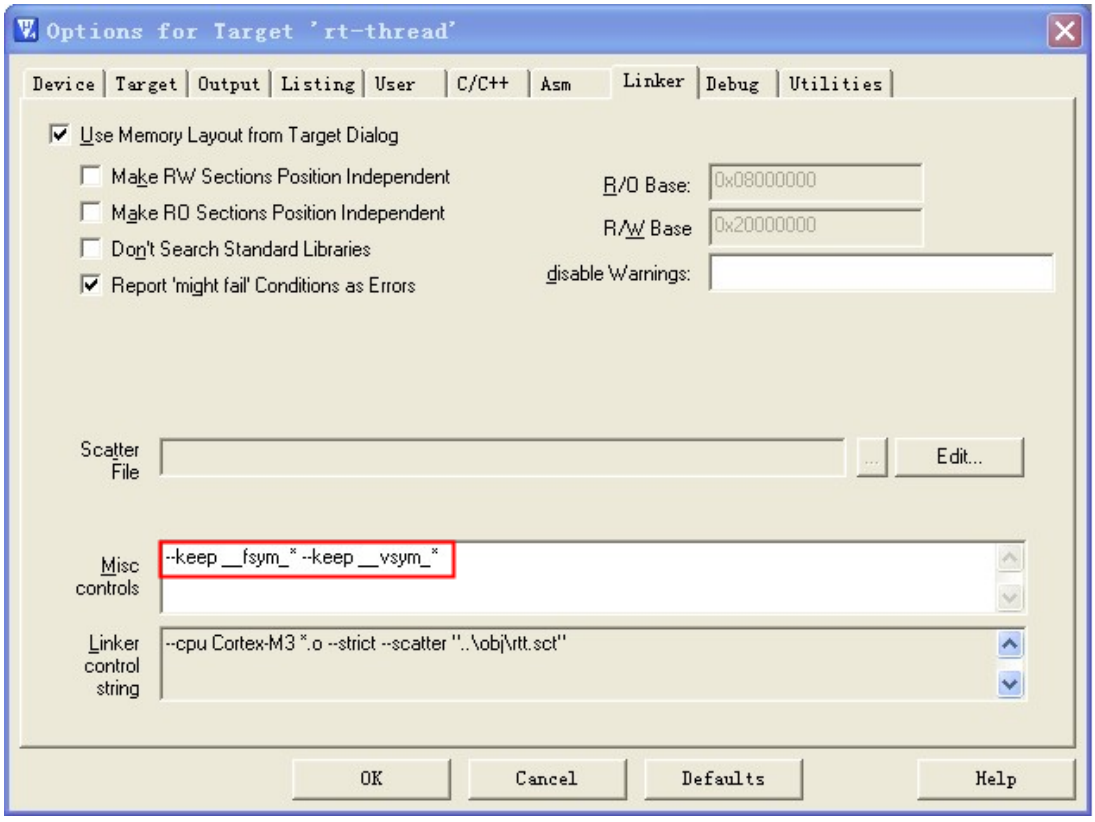
并将工程编译时的 output 路径和 listing 路径执行 obj 文件夹

✧ 其他设置

在下图所示区域设置器件类型和制定使用 stm32 固件库：



最后加入一些编译控制，防止一些没被调用的函数被连接器移除（使用 finsh 时很重要）



✧ 编译、下载验证

如果上面步骤都没问题，工程将会顺利编译通过，如果有错误提示，请检查各个步骤的操作。

```
linking...
Program Size: Code=63506 RO-data=5058 RW-data=456 ZI-data=6584
"..\obj\rtt.axf" - 0 Error(s), 0 Warning(s).
```

0 error 0 warning

编译 ok 后，说明我们的设置都 ok，接下来，我们修改下硬件配置使之与我们的目标板对应：

led.c 中我们用 GPIOF8、GPIOF9 来替换原来的 GPIOE2、GPIOE3：

```
#define led1_rcc                    RCC_APB2Periph_GPIOF
#define led1_gpio                   GPIOF
#define led1_pin                    GPIO_Pin_8

#define led2_rcc                    RCC_APB2Periph_GPIOF
#define led2_gpio                   GPIOF
#define led2_pin                    GPIO_Pin_9
```

程序中默认使用串口 1 作为终端，针对我的目标板就无需改了，如果你的板子串口不是串口 1，则改动下面两处：

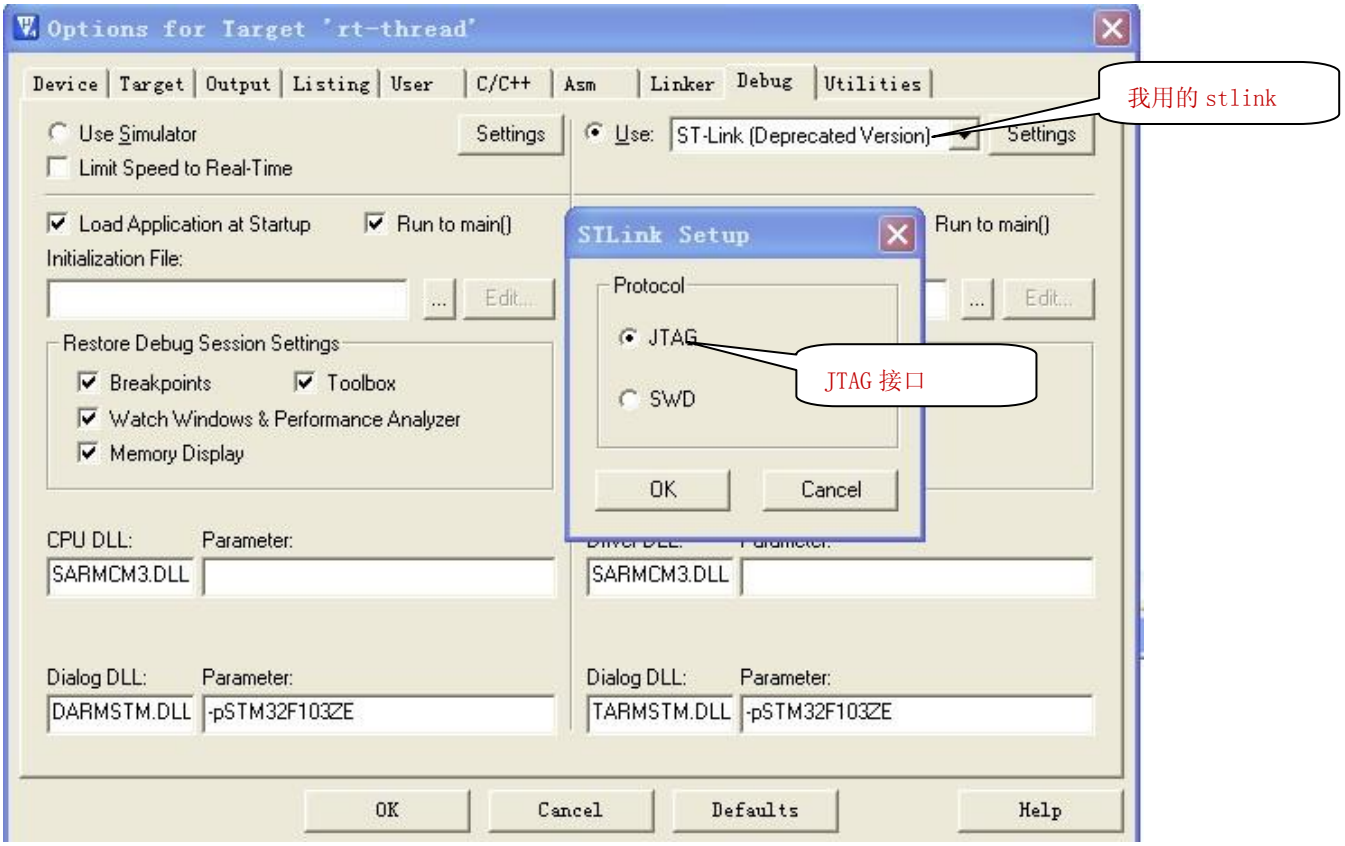
board.h 中：

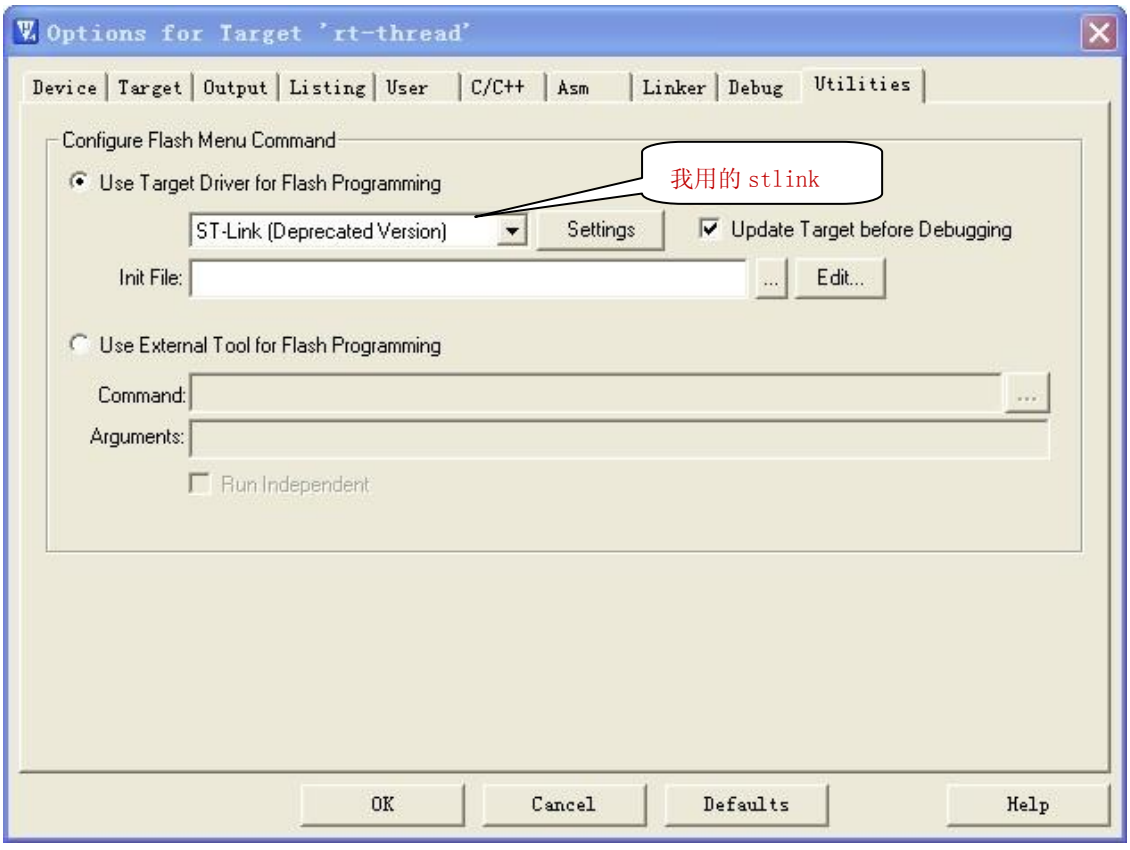
```
#define STM32_CONSOLE_USART     1 //根据目标板自己实际修改
```

rt_config.h 中：

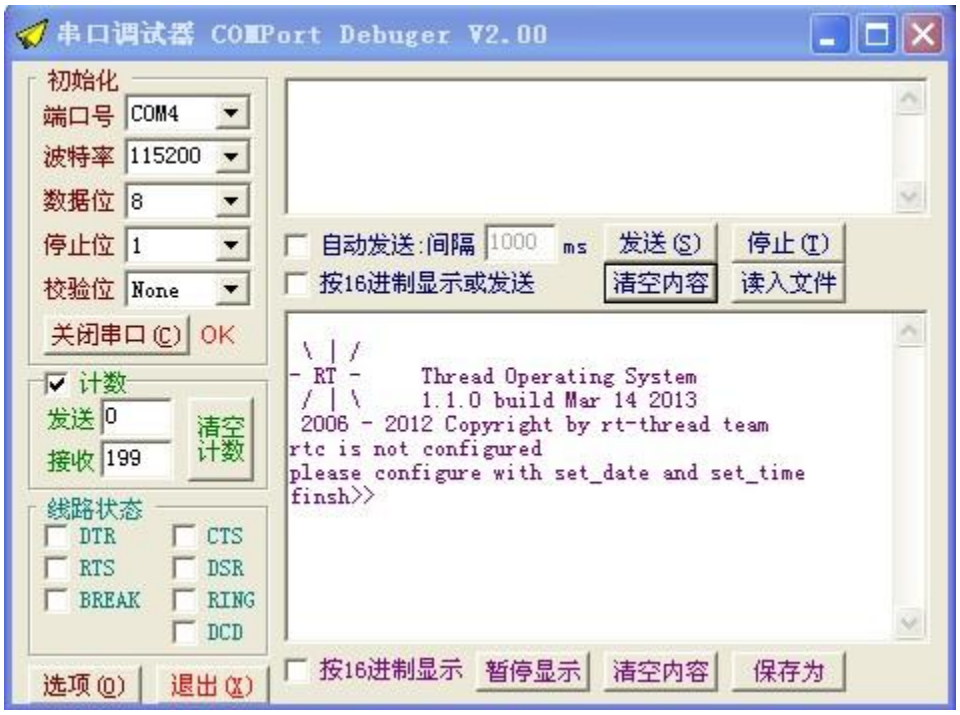
```
#define RT_USING_UART1          //根据目标板实际情况修改
```

最后进行下载前的最后设置：


Options for Target 'rt-thread'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
我用的 stlink
Use Simulator
Settings
Use:
ST-Link (Deprecated Version)
Settings
Limit Speed to Real-Time
Load Application at Startup
Run to main()
Initialization File:
Edit...
Restore Debug Session Settings
Breakpoints
Toolbox
Watch Windows & Performance Analyzer
Memory Display
STLink Setup
Protocol
JTAG
SWD
JTAG 接口
OK
Cancel
CPU DLL:
Parameter:
SARMCM3.DLL
Dialog DLL:
DARMSTM.DLL
-pSTM32F103ZE
TARMSTM.DLL
Defaults
Help


Options for Target 'rt-thread'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
Configure Flash Menu Command
Use Target Driver for Flash Programming
我用的 stlink
ST-Link (Deprecated Version)
Settings
Update Target before Debugging
Init File:
Edit...
Use External Tool for Flash Programming
Command:
Arguments:
Run Independent
OK
Cancel
Defaults
Help

以上都 ok 后，我们点击 MDK 下 debug->start/stop Debug session 即可下载程序，完成后后点击 debug->run 即可观察到串口的输出和 led 灯闪烁，至此工程建立完毕！！！



。