

SES 的使用以及和 MDK keil 类比

刘权

Email:450547566@qq.com

2017/12/29

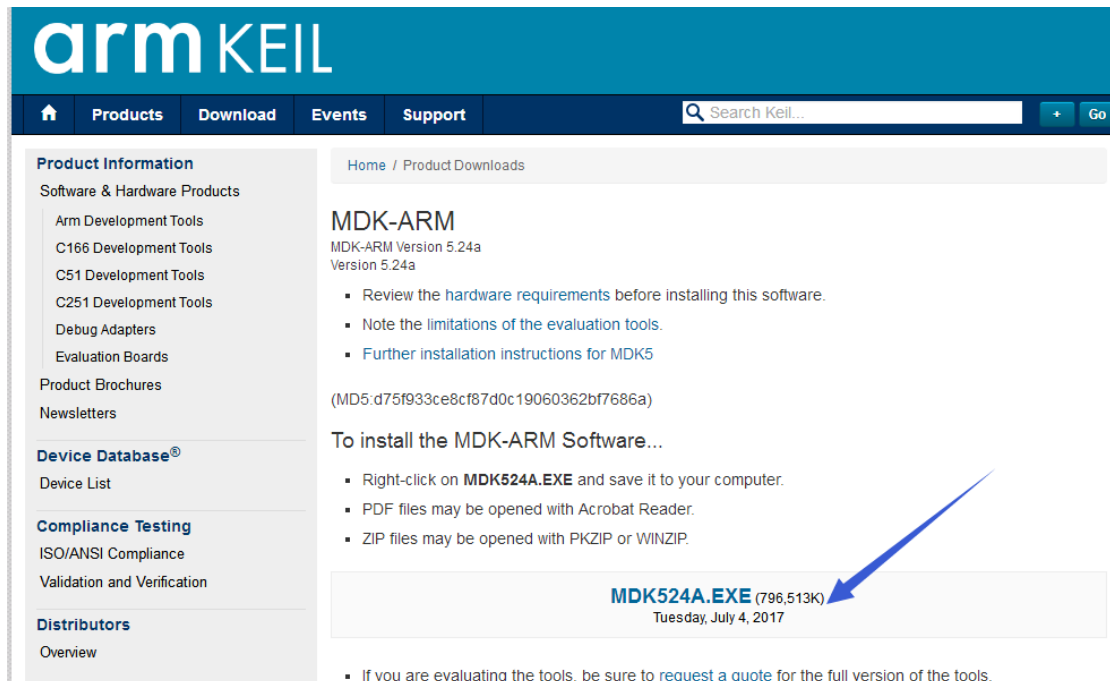
目录

1 安装包下载.....	1
2 整体照	3
3 工程配置.....	4
3.1 SES 的公共和私有工程配置	4
3.2 工程设置入口.....	4
3.3 芯片型号选择.....	6
3.3.1 keil 选择	6
3.3.2 SES 选择	6
3.4 ROM 和 RAM 范围设置.....	8
3.4.1 keil 设置	8
3.4.2 SES 设置	8
3.5 目标输出路径和生成目标文件格式.....	9
3.5.1 keil 设置	9
3.5.2 SES 的输出路径设置	9
3.5.3 目标格式选择.....	11
3.6 预编译宏、工程头文件和编译优化等级.....	11
3.6.1 keil 设置	11
3.6.2 SES 预编译和工程头文件	12
3.6.3 SES 编译优化等级	12
3.7 支持 C99	13
3.7.1 keil 设置	13
3.7.2 SES 设置	13
3.8 调试设置.....	14
3.8.1 keil 设置	14
3.8.2 SES 设置调试模式	14
3.8.3 SES 设置 jlink 通信接口模式.....	15
4 工程编译.....	16
4.1 keil 编译	16
4.2 SES 编译	16
5 工程调试.....	17
5.1 keil 调试	17
5.2 SES 调试	19
6 SES 新建工程.....	20
6.1 SES 新建最小工程	20
6.2 添加协议栈.....	25
6.3 修改资源配置文件 flash_placement.xml	26
6.3.1 官方新建时的 flash_placement.xml	26
6.3.2 nordic 提供的 flash_placement.xml.....	27
6.4 计算应用程序的 ROM 和 RAM.....	29
6.5 向工程添加 flash_placement.xml	30
6.6 工程创建文件夹、新建.c 或.s、添加资源文件.....	31

6.7 添加头文件.....	31
6.8 其他设置.....	31
7 导入 keil 工程时使用 arm 编译器支持 C99 编译.....	31
8 支持标准库.....	33
8.1 keil 设置	33
8.2 SES 设置	33
9 支持 CMSIS-DSP 库.....	34
9.1 CMSIS 包的下载路径.....	34
9.1.1 github 下载	35
9.1.2 arm 官网下载	35
9.2 解压 CMSIS pack.....	35
9.3 SES 使用浮点运算	36
9.3.1 使用硬浮点算法库.....	36
10 常用功能.....	38
10.1 查找	38
10.2 代码查看.....	39
10.3 工具窗口.....	40
11 SES 调试深究.....	41
11.1 SES 断点	41
12 程序中的段.....	42
13 XML 文件详解(Section Placement file)	43
13.1 ProgramSection 属性	43
13.1.1 name 名字属性	43
13.1.2 start 起始地址属性	43
13.1.3 size 段具体大小属性.....	43
13.1.4 address_symbol 地址开始符号属性.....	43
13.1.5 end_symbol 地址结束符号属性	43
13.1.6 size_symbol 段大小符号属性	43
13.1.7 alignment 访问对齐属性	44
13.1.8 fill 填充属性.....	44
13.1.9 inputsections 输入哪些文件到 name 段.....	44
13.1.10 keep 保持属性.....	44
13.1.11 load 加载属性.....	45
13.1.12 place_from_segment_end 段末尾开始放数据属性	45
13.1.13 runin 属性	45
13.1.14 runoffset 属性.....	45
14 连接文件浅析.....	46

1 安装包下载

Keil 下载地址: <https://www.keil.com/demo/eval/arm.htm#/DOWNLOAD>



armKEIL

Home / Product Downloads

MDK-ARM

MDK-ARM Version 5.24a
Version 5.24a

- Review the [hardware requirements](#) before installing this software.
- Note the limitations of the evaluation tools.
- [Further installation instructions for MDK5](#)

(MD5:d75f933ce8cf87d0c19060362bf7686a)

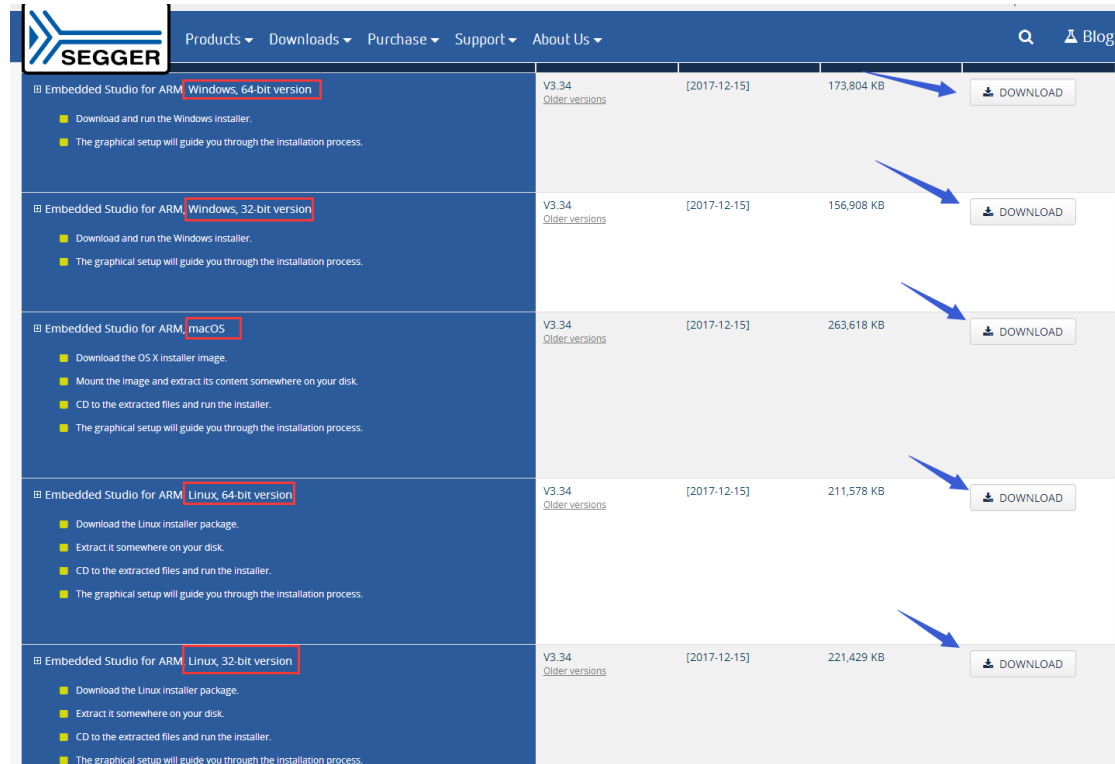
To install the MDK-ARM Software...

- Right-click on **MDK524A.EXE** and save it to your computer.
- PDF files may be opened with Acrobat Reader.
- ZIP files may be opened with PKZIP or WINZIP.

MDK524A.EXE (796,513K)
Tuesday, July 4, 2017

- If you are evaluating the tools, be sure to [request a quote](#) for the full version of the tools.

SES 下载地址: <https://www.segger.com/downloads/embedded-studio>




SEGGER

Products Downloads Purchase Support About Us

Embedded Studio for ARM Windows, 64-bit version <ul style="list-style-type: none">Download and run the Windows installer.The graphical setup will guide you through the installation process.	V3.34 Older versions	[2017-12-15]	173,804 KB	DOWNLOAD
Embedded Studio for ARM Windows, 32-bit version <ul style="list-style-type: none">Download and run the Windows installer.The graphical setup will guide you through the installation process.	V3.34 Older versions	[2017-12-15]	156,908 KB	DOWNLOAD
Embedded Studio for ARM macOS <ul style="list-style-type: none">Download the OS X installer image.Mount the image and extract its content somewhere on your disk.CD to the extracted files and run the installer.The graphical setup will guide you through the installation process.	V3.34 Older versions	[2017-12-15]	263,618 KB	DOWNLOAD
Embedded Studio for ARM Linux, 64-bit version <ul style="list-style-type: none">Download the Linux installer package.Extract it somewhere on your disk.CD to the extracted files and run the installer.The graphical setup will guide you through the installation process.	V3.34 Older versions	[2017-12-15]	211,578 KB	DOWNLOAD
Embedded Studio for ARM Linux, 32-bit version <ul style="list-style-type: none">Download the Linux installer package.Extract it somewhere on your disk.CD to the extracted files and run the installer.The graphical setup will guide you through the installation process.	V3.34 Older versions	[2017-12-15]	221,429 KB	DOWNLOAD

Jlink 下载地址: <https://www.segger.com/downloads/jlink#>



[Products](#)
[Downloads](#)
[Purchase](#)
[Support](#)
[About Us](#)

[Contact Us](#)
[Web Shop](#)
[Newsletter](#)

J-Link Software and Documentation Pack

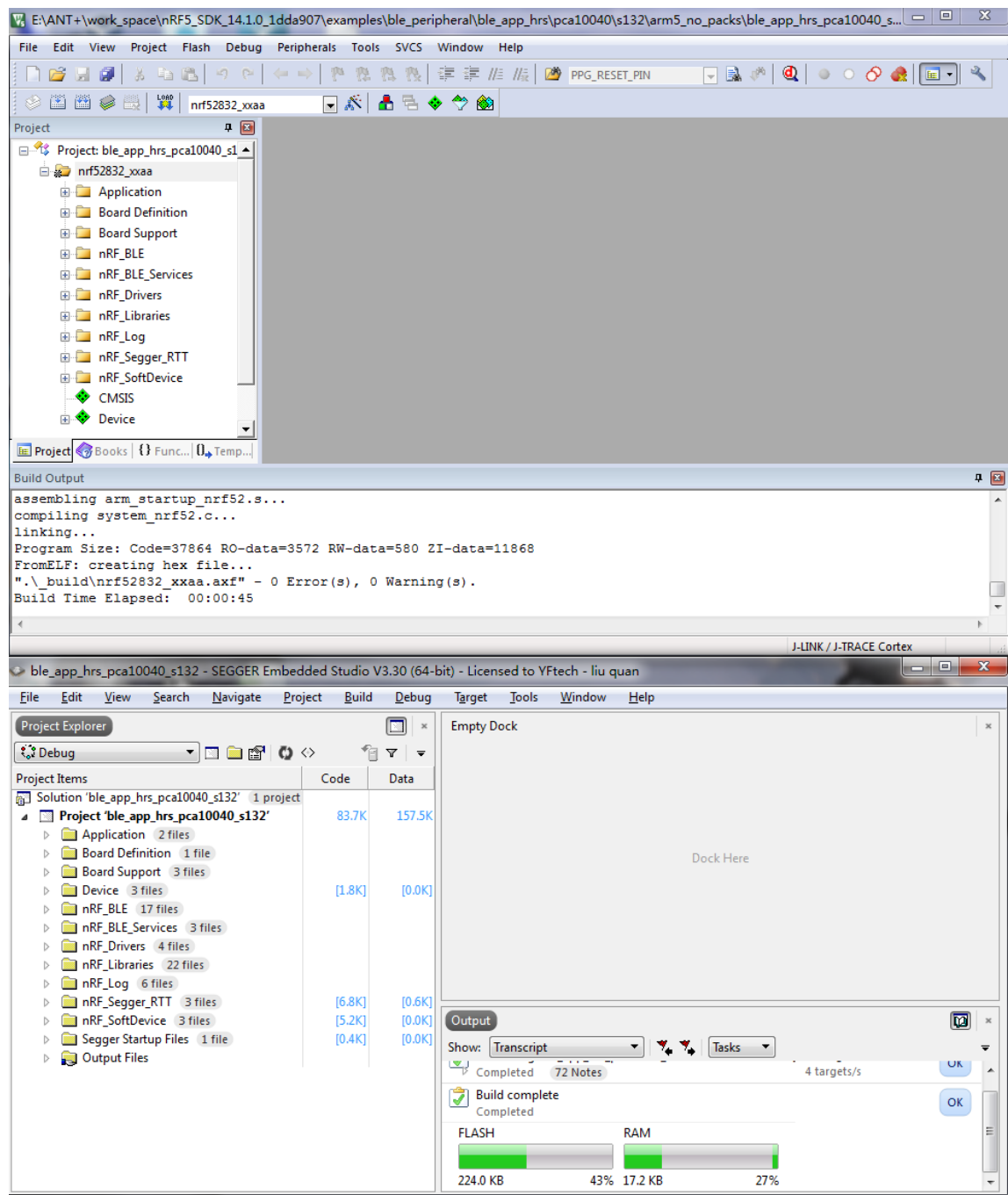
- All-in-one debugging solution
- Can be downloaded and used free of charge by any owner of a SEGGER J-Link, J-Trace or Flasher model.
Not all features of it may be available on all J-Link / J-Trace / Flasher models.
- Updated frequently
- [Release Notes](#)
- [More information](#)

[Click for downloads](#)

	Version	Date	File size	
J-Link Software and Documentation pack for Windows <small>Installing the software will automatically install the J-Link USB drivers and offers to update applications which use the J-Link DLL. Multiple versions of the J-Link software can be installed on the same PC without problems; they will co-exist in different directories.</small>	V6.22d Older versions	[2017-12-14]	26,690 KB	DOWNLOAD
J-Link Software and Documentation pack for MacOSX	V6.22d Older versions	[2017-12-14]	47,326 KB	DOWNLOAD
J-Link Software and Documentation pack for Linux, DEB Installer, 32-bit	V6.22d Older versions	[2017-12-14]	13,690 KB	DOWNLOAD
J-Link Software and Documentation pack for Linux, DEB Installer, 64-bit	V6.22d Older versions	[2017-12-14]	22,315 KB	DOWNLOAD
J-Link Software and Documentation pack for Linux, RPM Installer, 32-bit	V6.22d Older versions	[2017-12-14]	13,687 KB	DOWNLOAD
J-Link Software and Documentation pack for Linux, RPM Installer, 64-bit	V6.22d Older versions	[2017-12-14]	18,320 KB	DOWNLOAD
J-Link Software and Documentation pack for Linux, TGZ archive, 32-bit	V6.22d Older versions	[2017-12-14]	13,702 KB	DOWNLOAD
J-Link Software and Documentation pack for Linux, TGZ archive, 64-bit	V6.22d Older versions	[2017-12-14]	22,335 KB	DOWNLOAD
J-Link Software and Documentation pack for Linux ARM systems <small>Note: This package is only needed in the special case of J-Link being controlled by a Linux system which is running on an ARM hardware (so J-Link is connected via USB/Ethernet to this ARM system that is running Linux). In case J-Link is just connected to a PC running Linux, the "J-Link software & documentation pack for Linux" is the correct package.</small>	V6.22d Older versions	[2017-12-14]	13,541 KB	DOWNLOAD

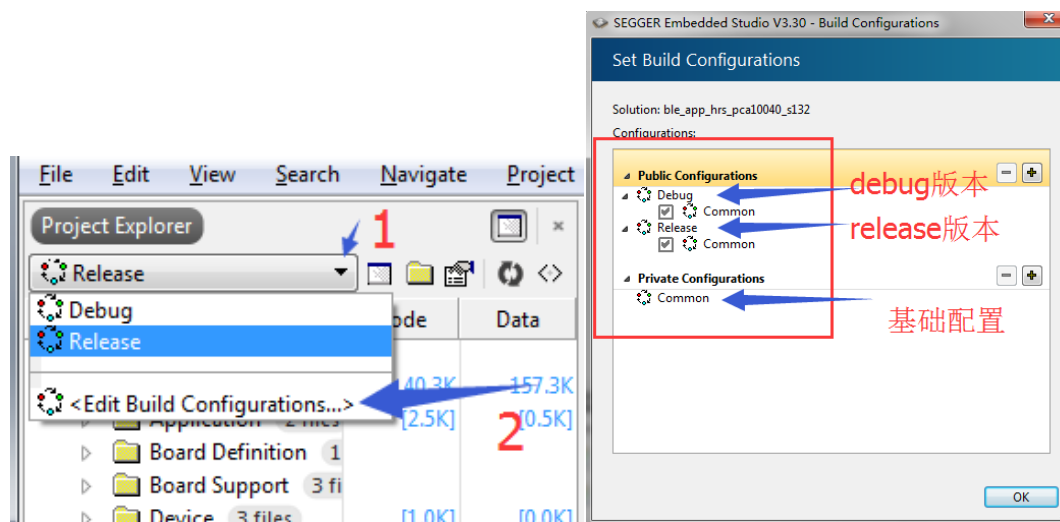
Note: This package comes without any support.

2 整体照



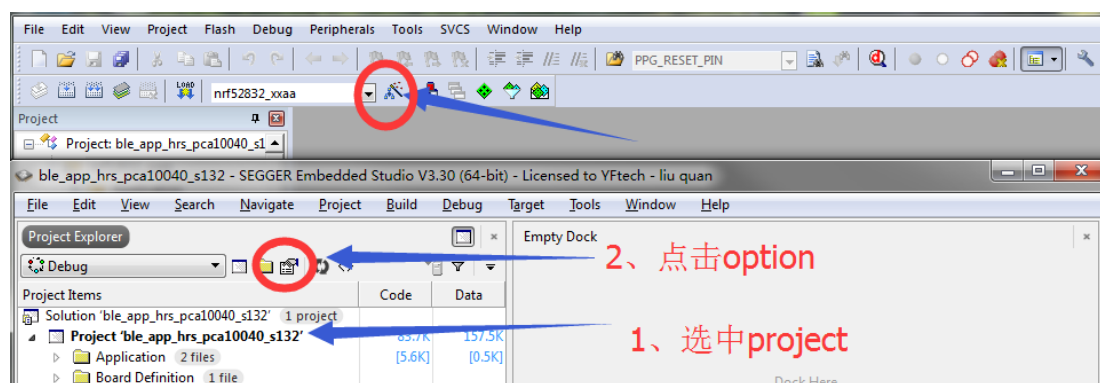
3 工程配置

3.1 SES 的公共和私有工程配置

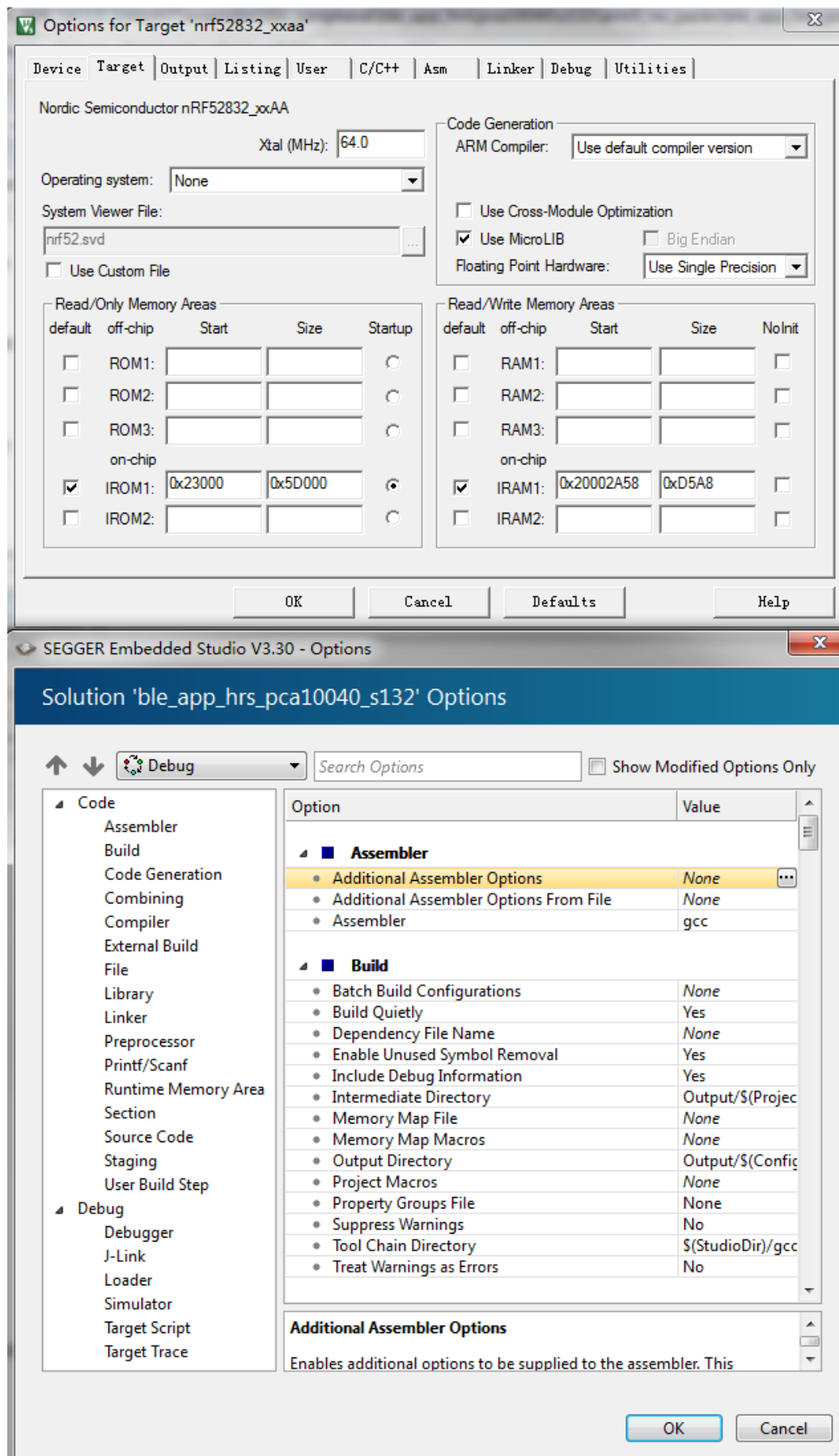


因为有些配置是在 common 里面才能修改，下面会用到这些选项。

3.2 工程设置入口

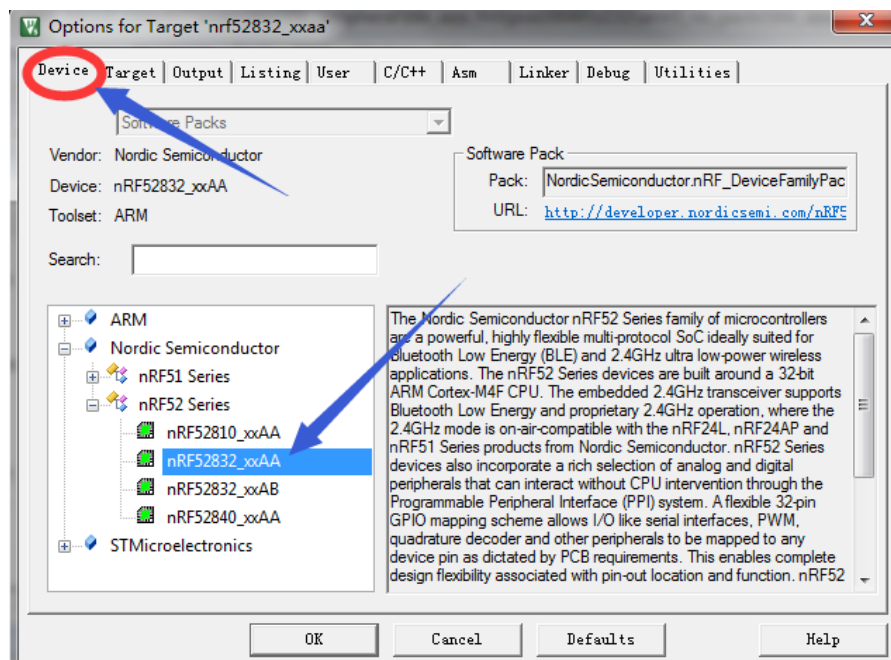


点击之后显示如下：

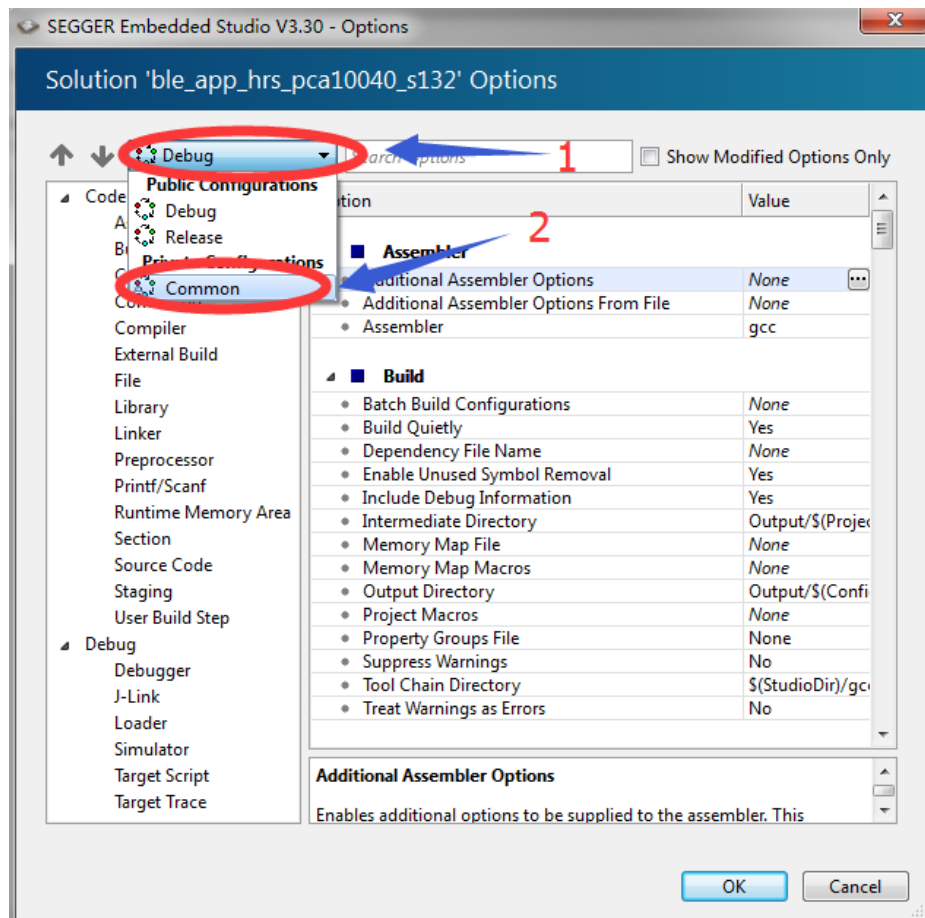


3.3 芯片型号选择

3.3.1 keil 选择

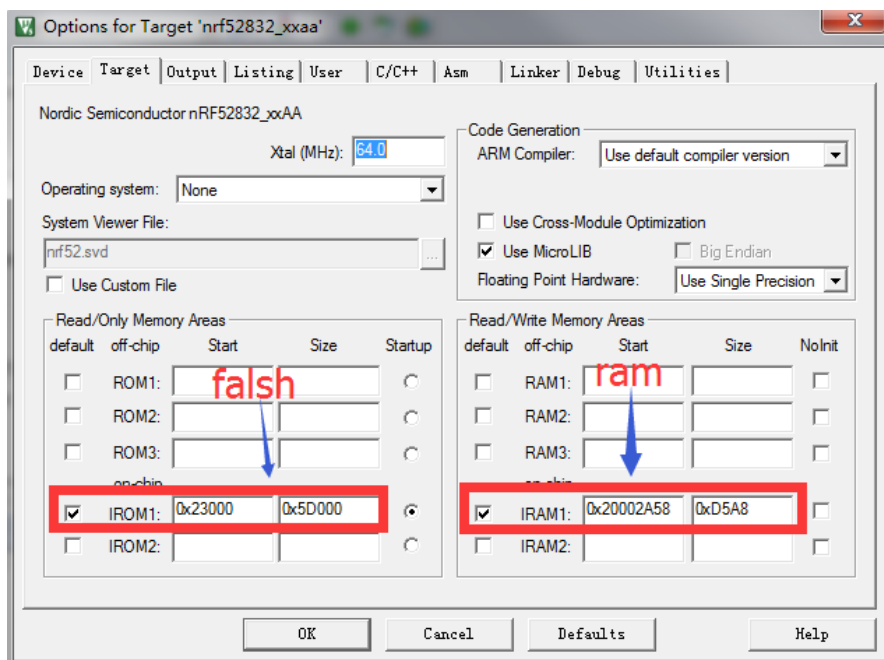


3.3.2 SES 选择

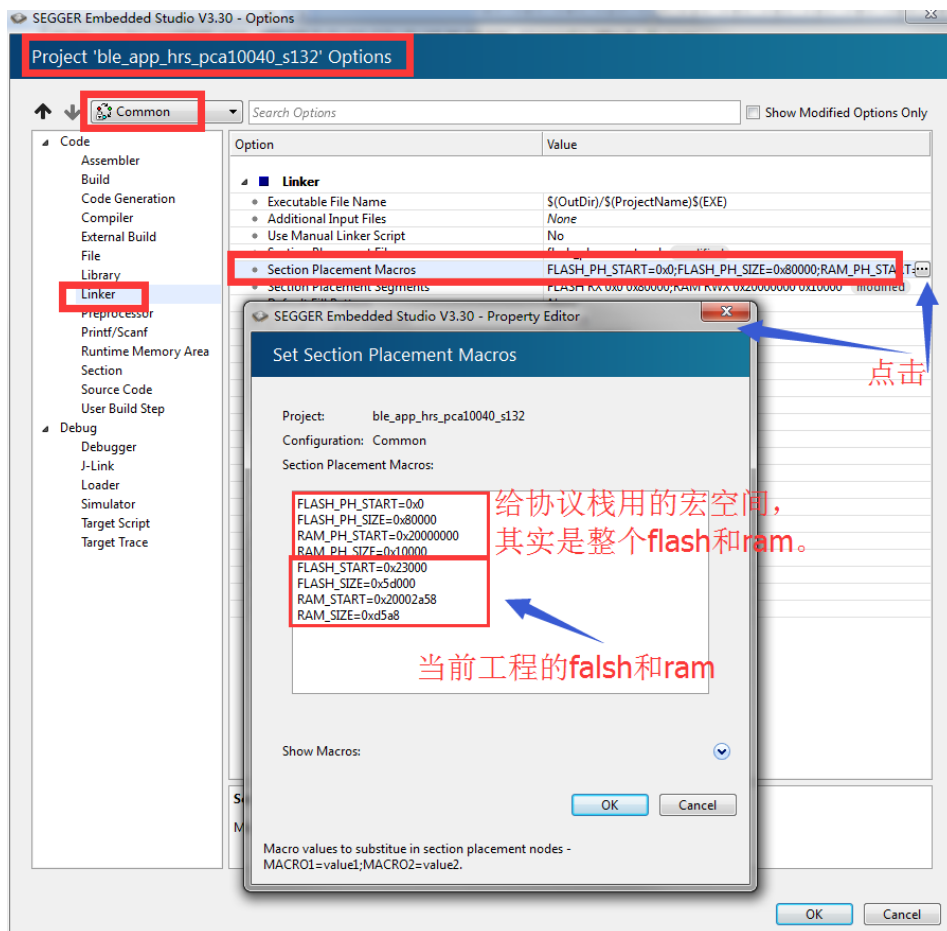


3.4 ROM 和 RAM 范围设置

3.4.1 keil 设置

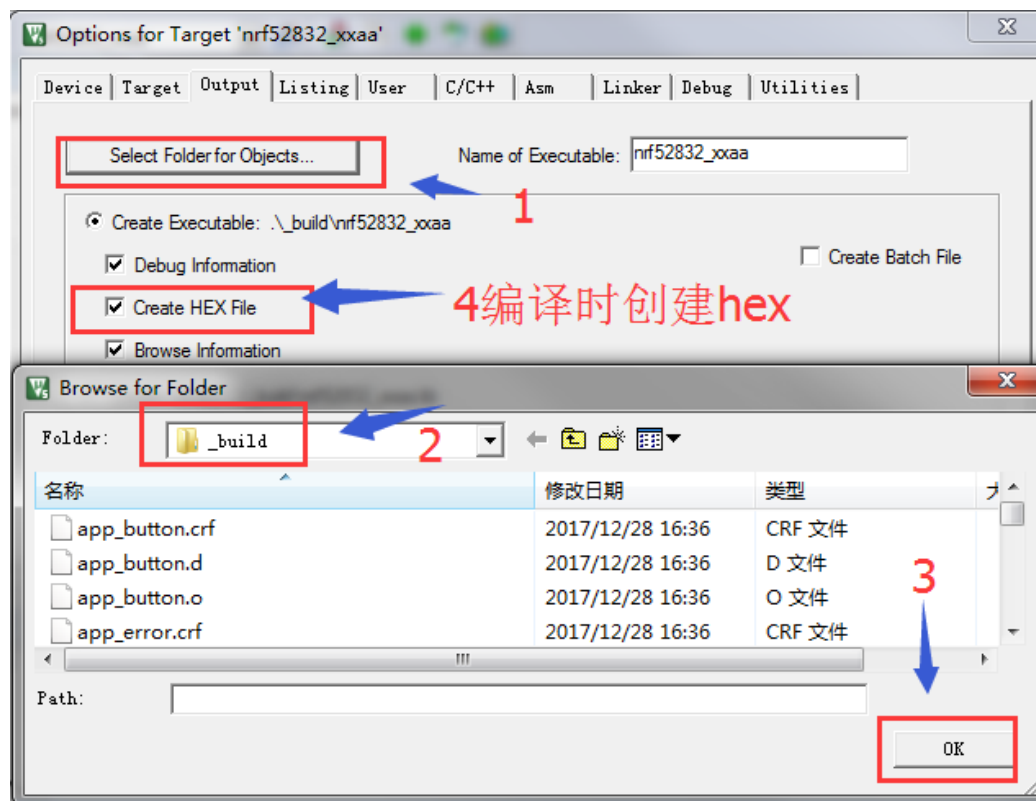


3.4.2 SES 设置

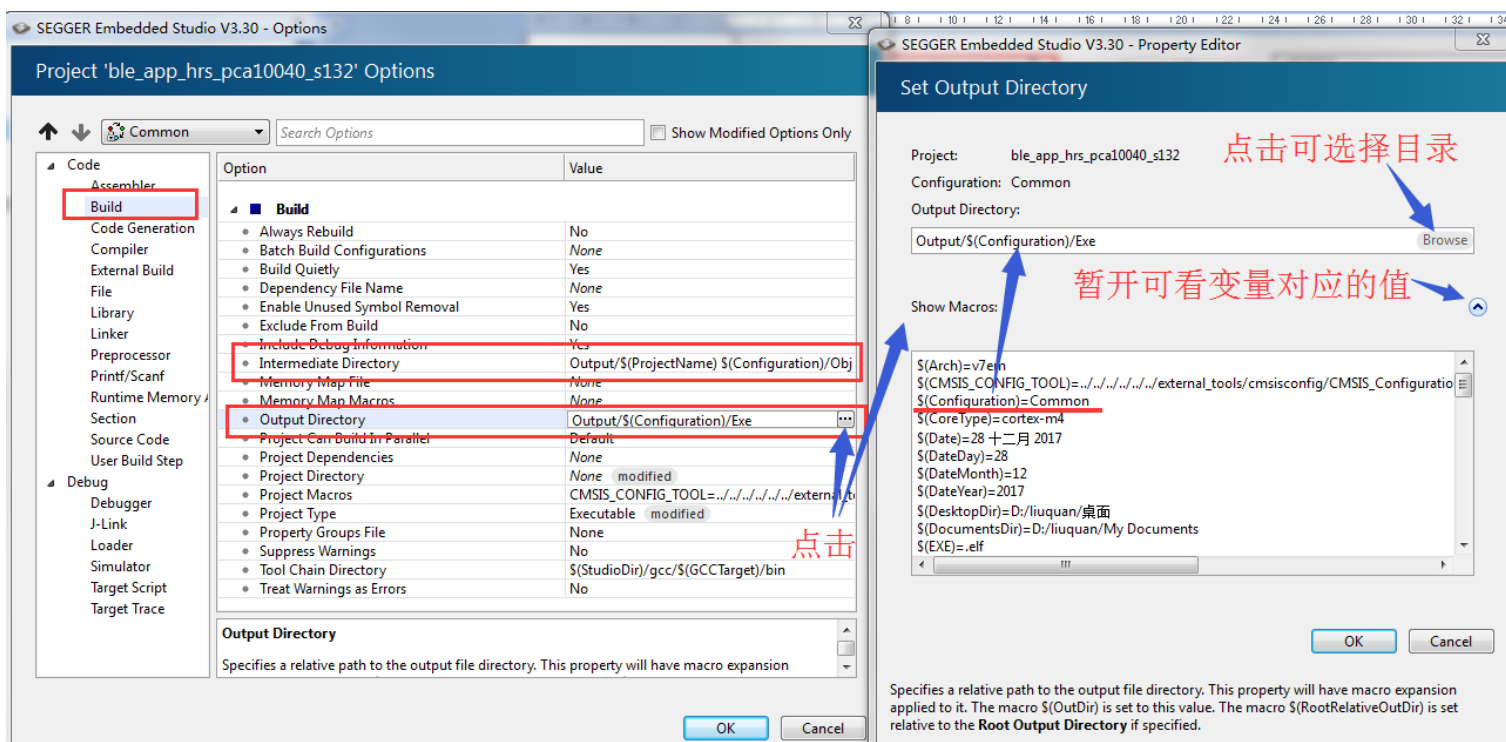


3.5 目标输出路径和生成目标文件格式

3.5.1 keil 设置



3.5.2 SES 的输出路径设置



这里注意：编译时可以选择 debug 和 release 两个选项进行编译，所以在上图中的 \$(Configuration) = Common 会变为 Debug 或者 Release。

上图中的 Intermediate Directory 为中间生产项的文件夹路径，而 Output Directory 则是目标文件输出的文件路径。例如这个工程的路径如下：

中间生成项输出路径设置为：

Output/\$(ProjectName) \$(Configuration)/Obj

其中\$(ProjectName)为

`$(ProjectNodeName)=ble_app_hrs_pca10040_s132`

\$(Configuration) = Debug 或者 Release。所以中间项产生的路径是：

Output/\$(ProjectName) \$(Configuration)/Obj=Output\ble_app_hrs_pca10040_s132 Debug\Obj

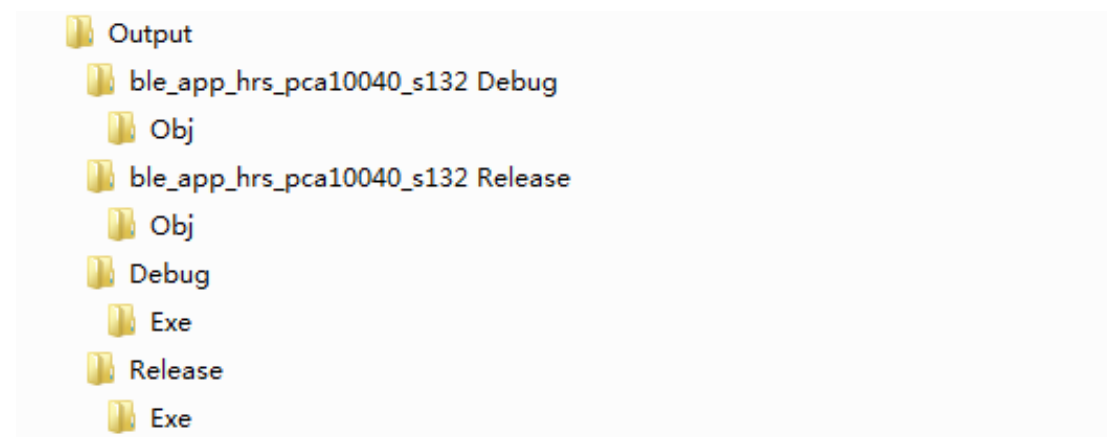
Output/\$(ProjectName) \$(Configuration)/Obj =Output\ble_app_hrs_pca10040_s132 Release\Obj

目标生成路径设置为：

Output/\$(Configuration)/Exe = Output\Debug\ Exe

Output/\$(Configuration)/Exe = Output\Release\ Exe

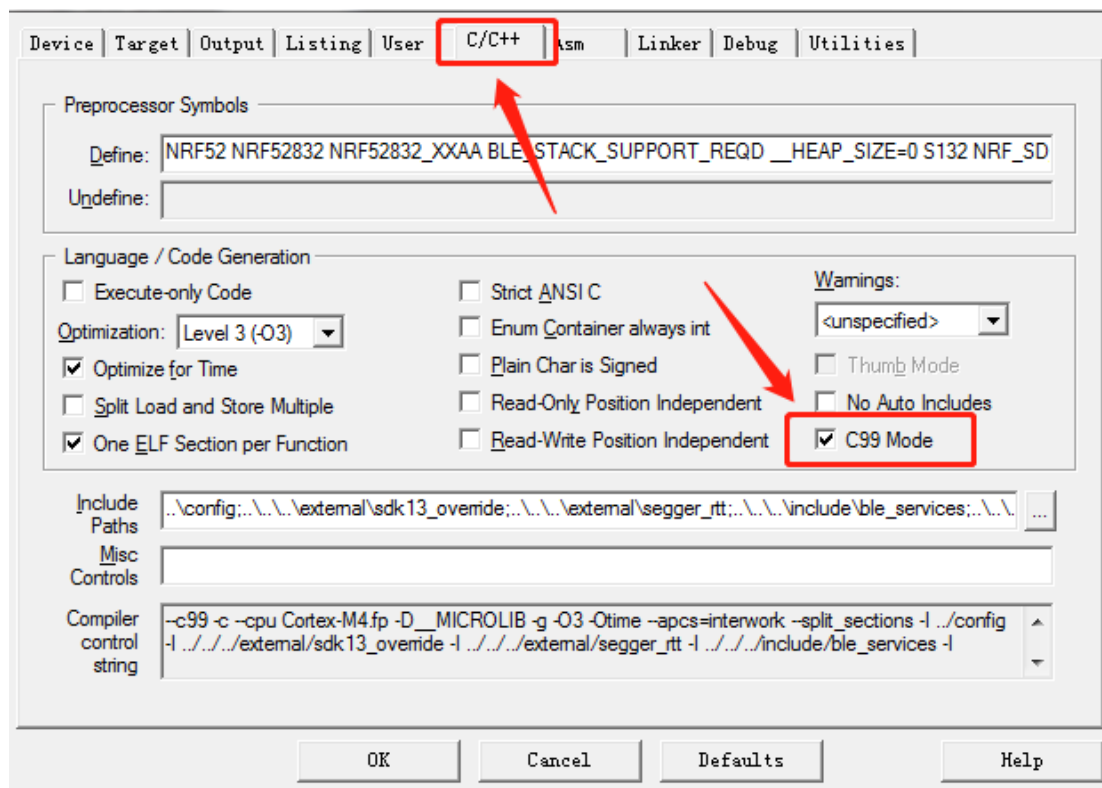
实际生成如下图：



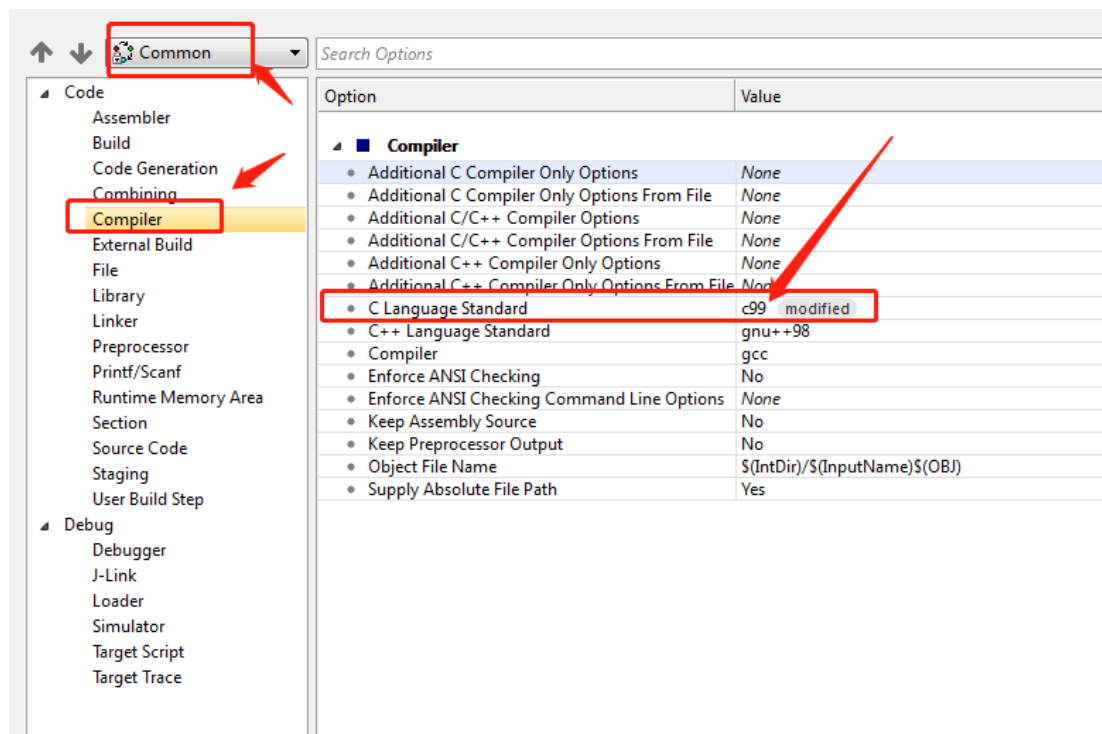


3.7 支持 C99

3.7.1 keil 设置

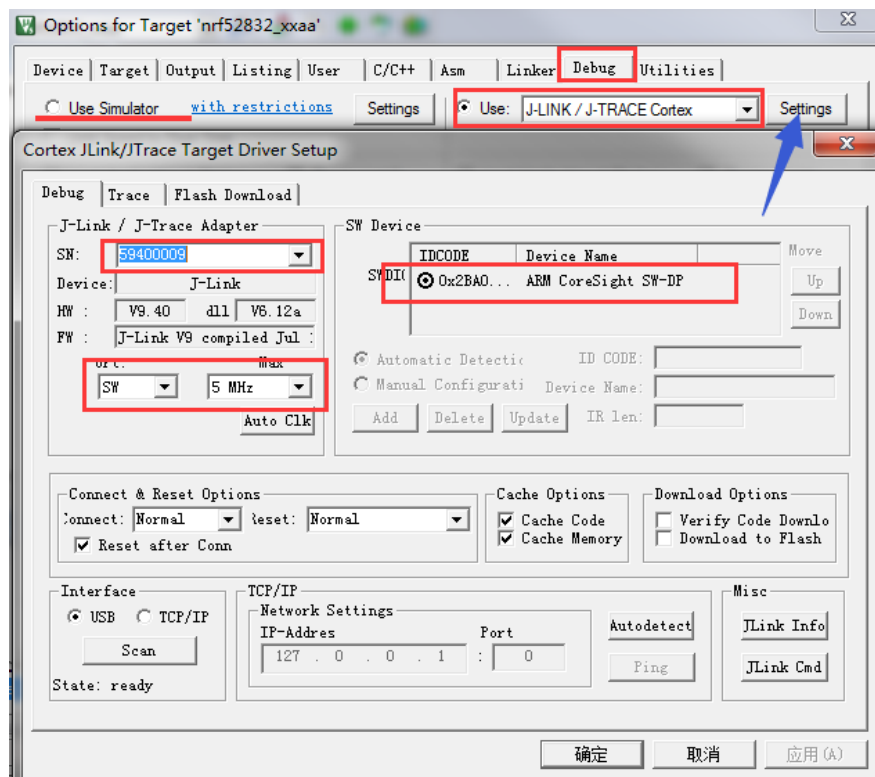


3.7.2 SES 设置

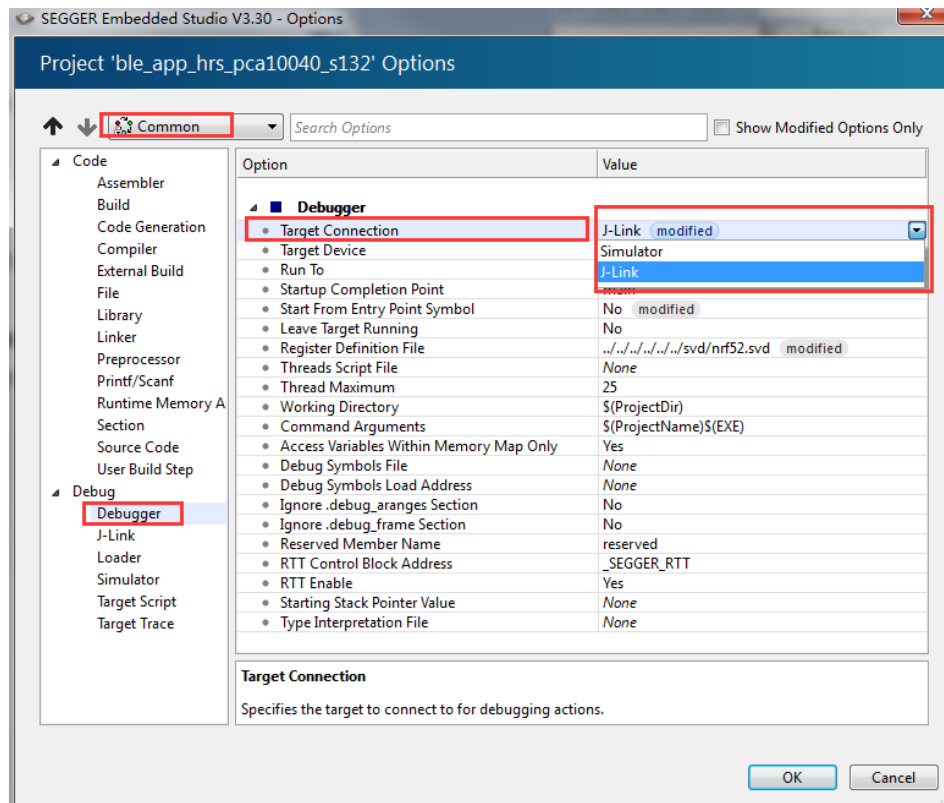


3.8 调试设置

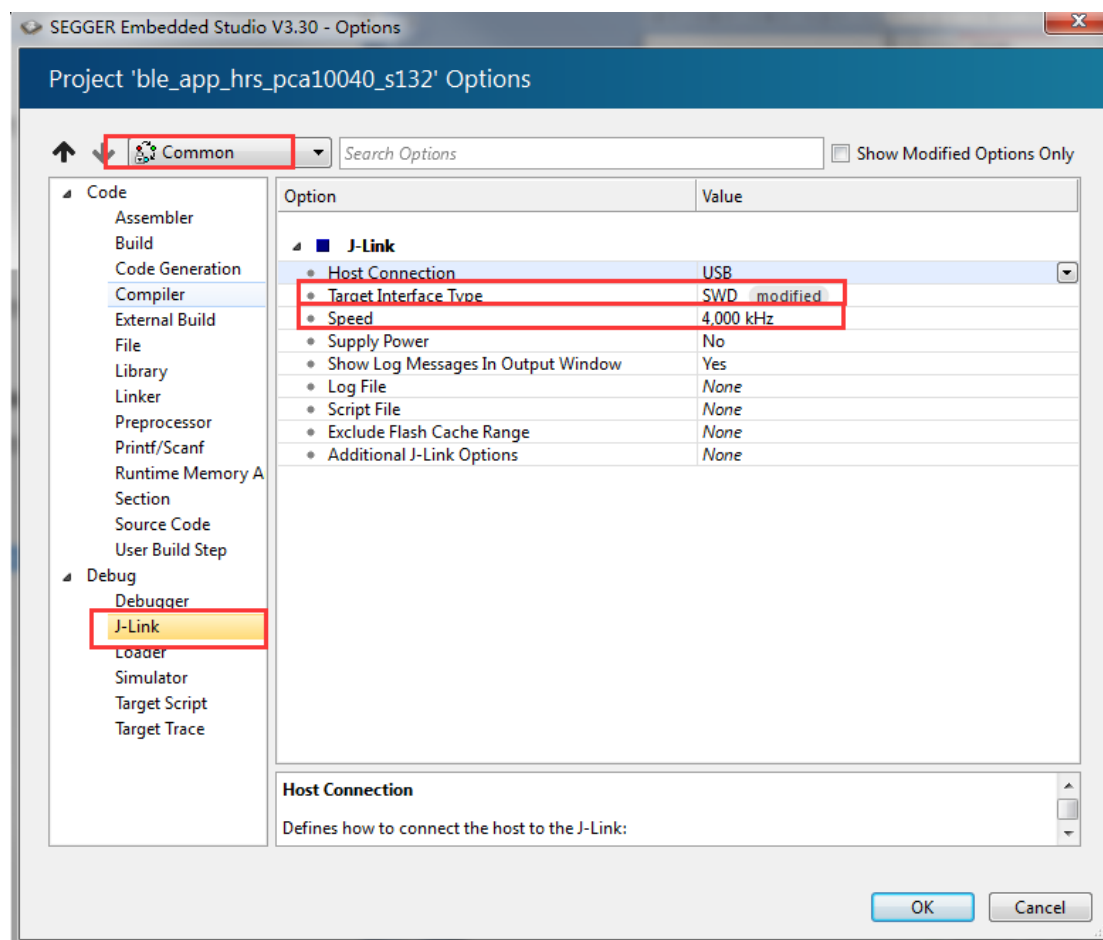
3.8.1 keil 设置



3.8.2 SES 设置调试模式

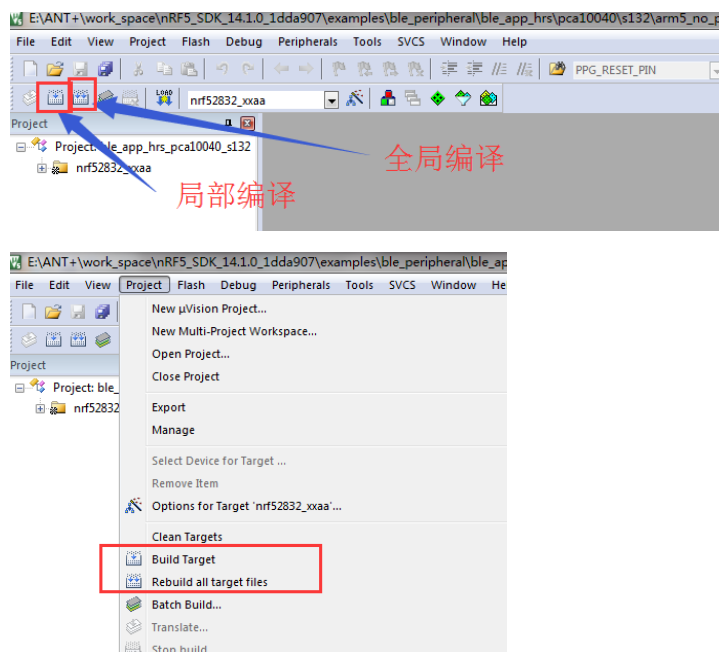


3.8.3 SES 设置 jlink 通信接口模式

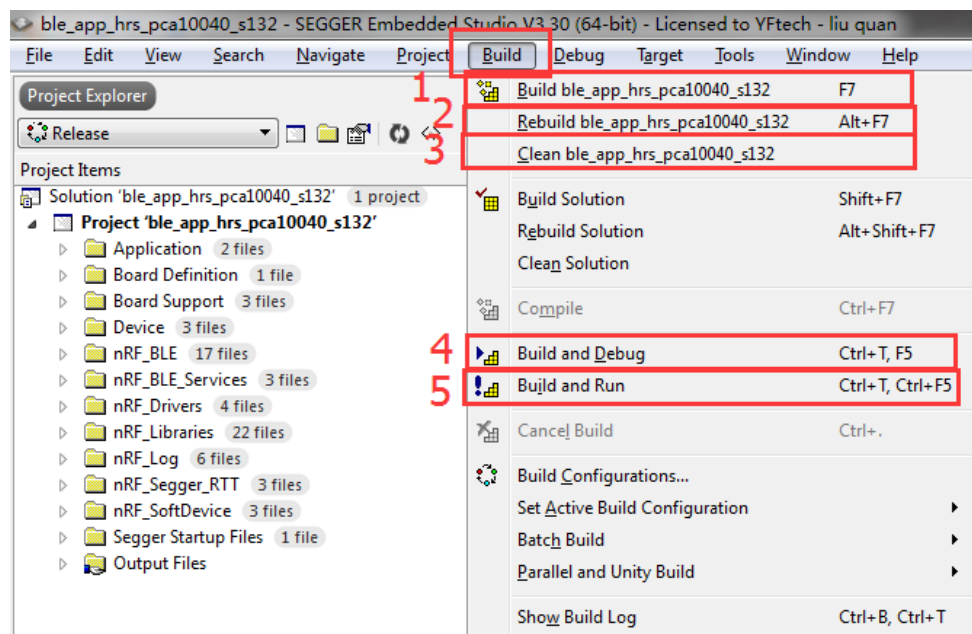


4 工程编译

4.1 keil 编译



4.2 SES 编译



标号 1: 编译工程

标号 2: 重新编译工程

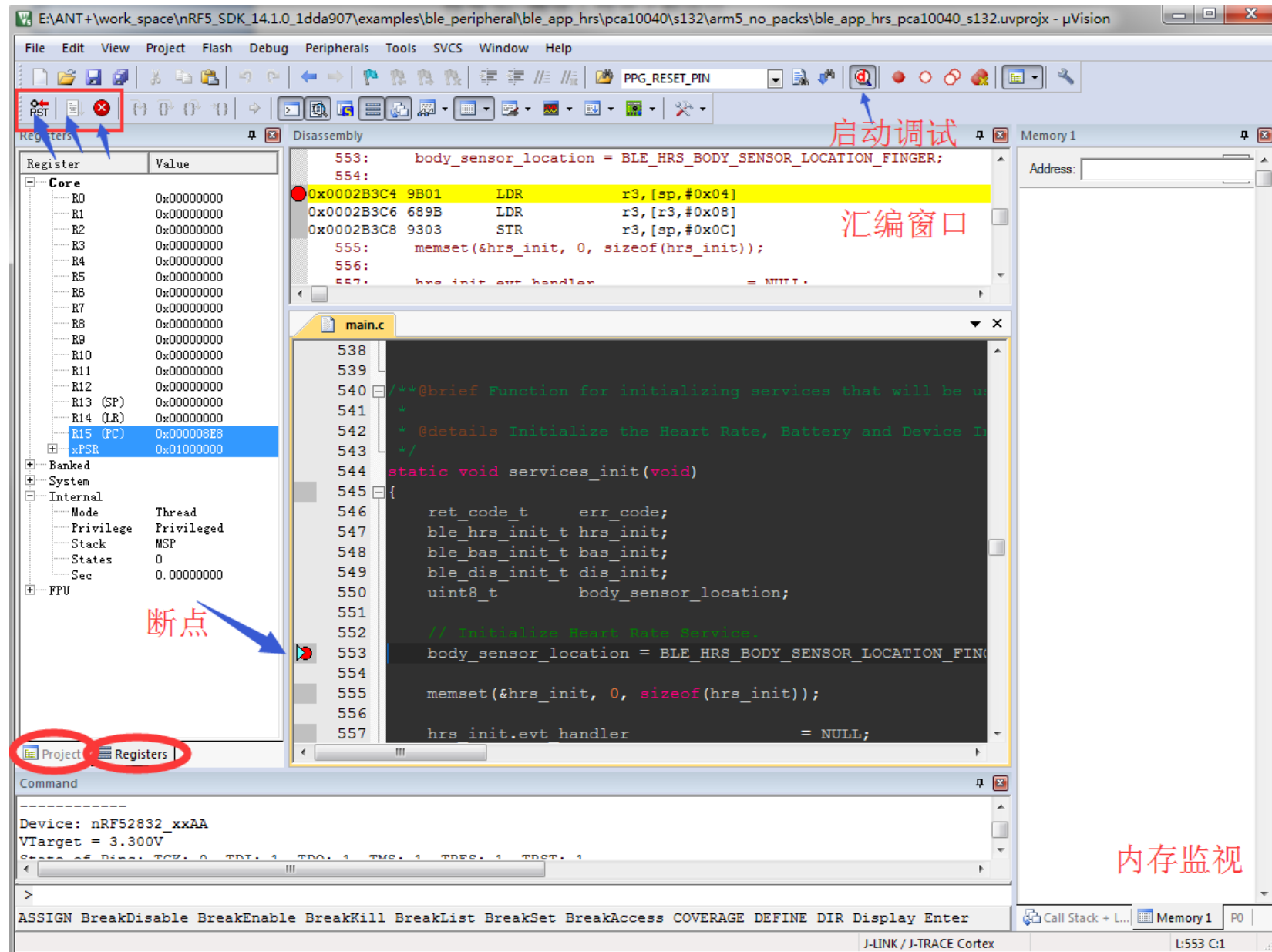
标号 3: 清除编译了工程

标号 4: 编译工程下载并开始调试

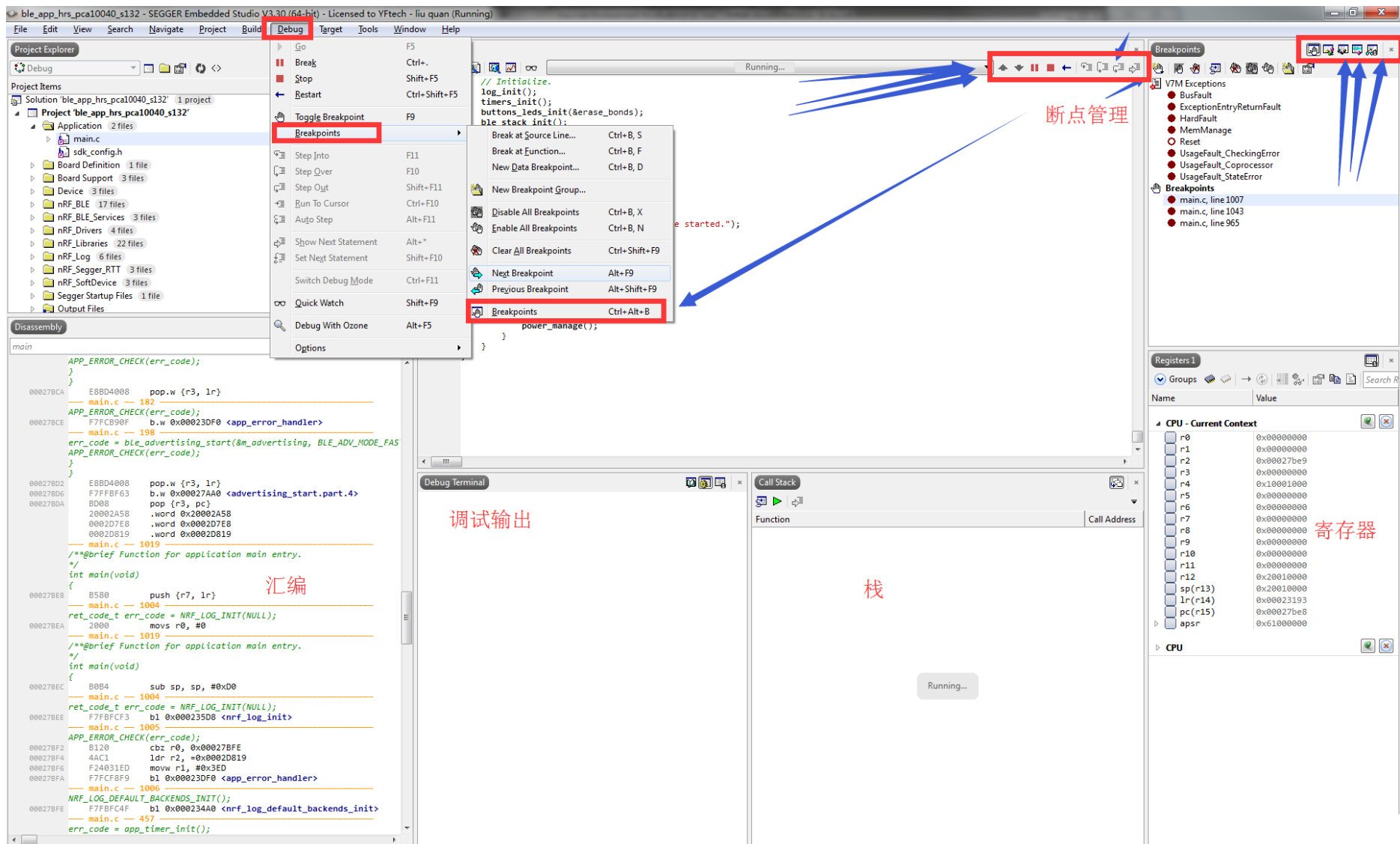
标号 5: 编译工程并下载运行

5 工程调试

5.1 keil 调试



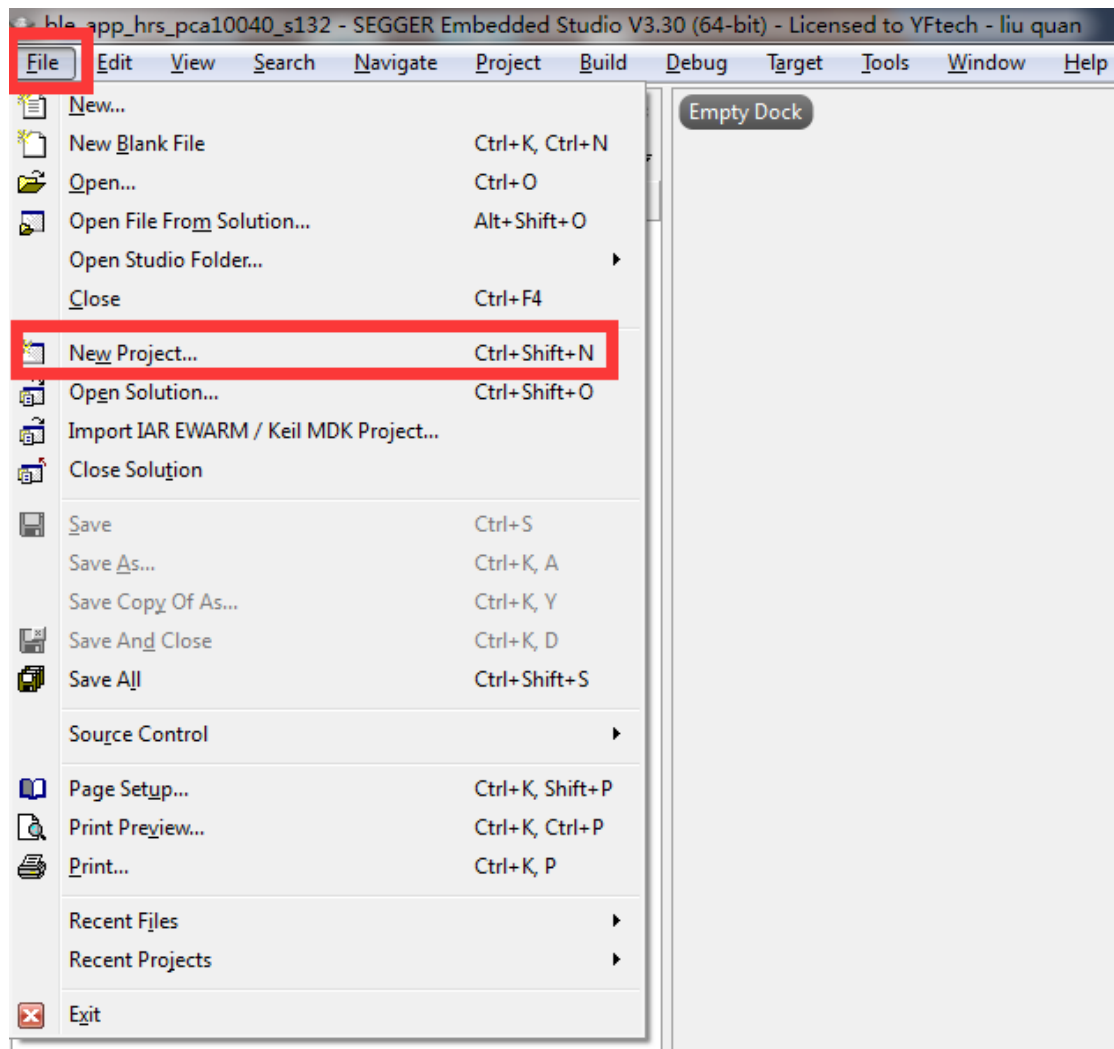
5.2 SES 调试

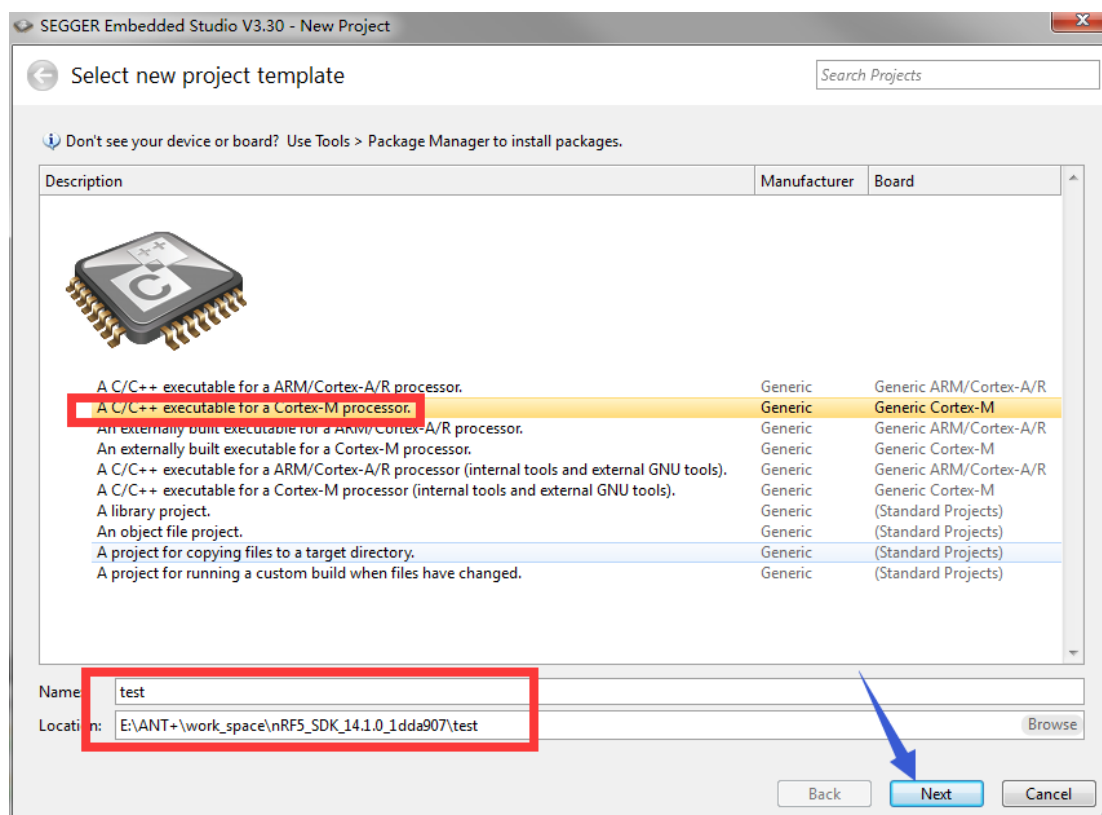
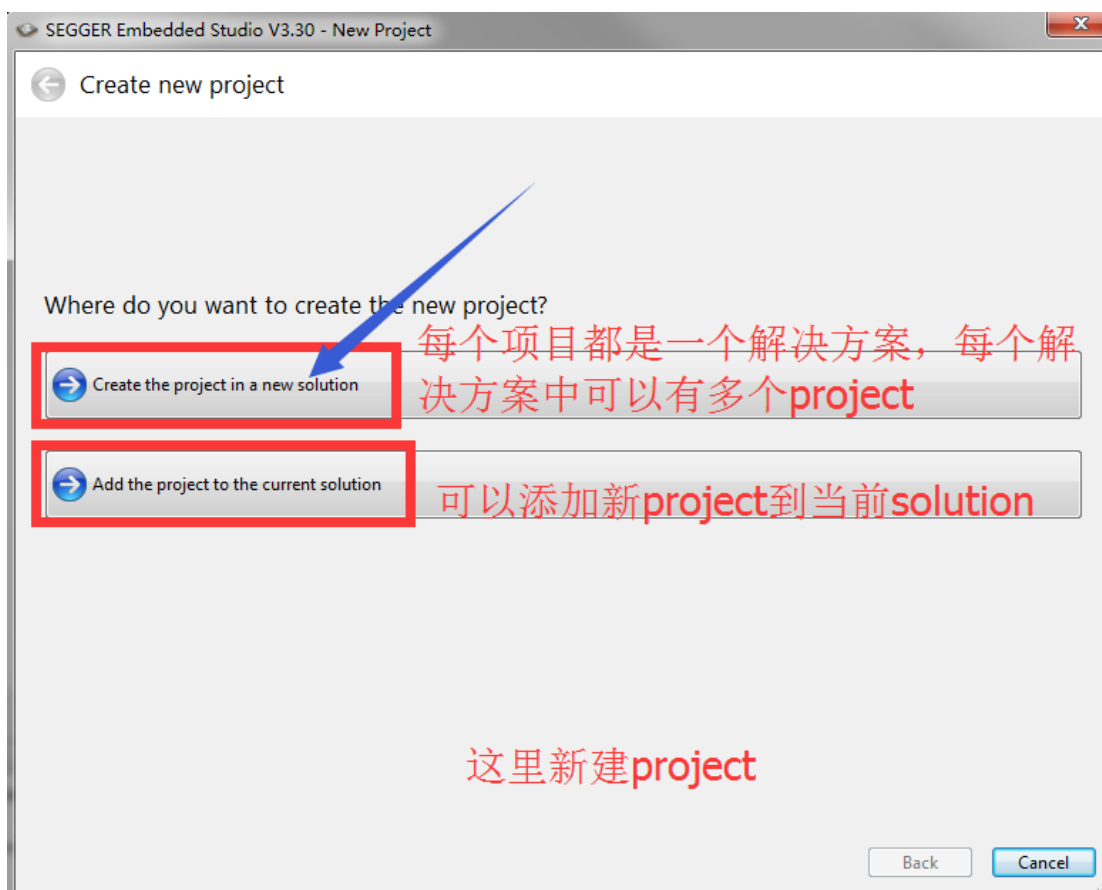


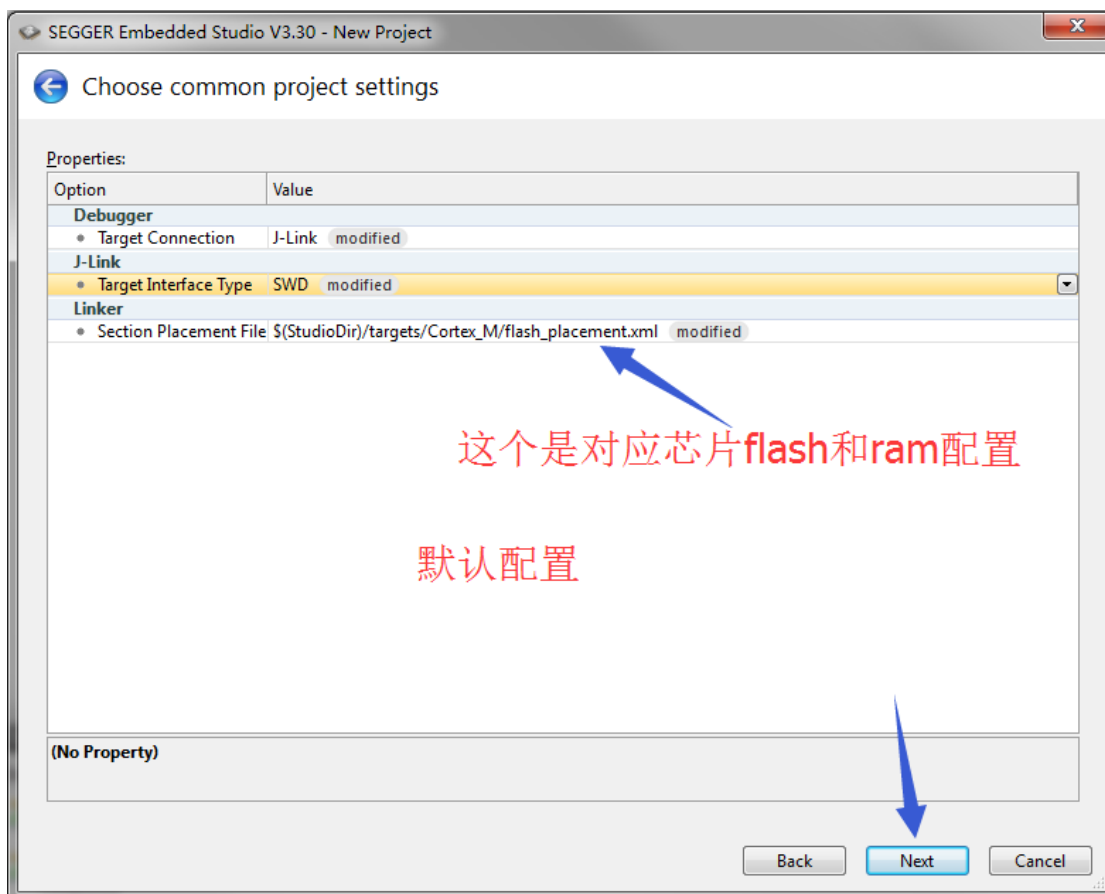
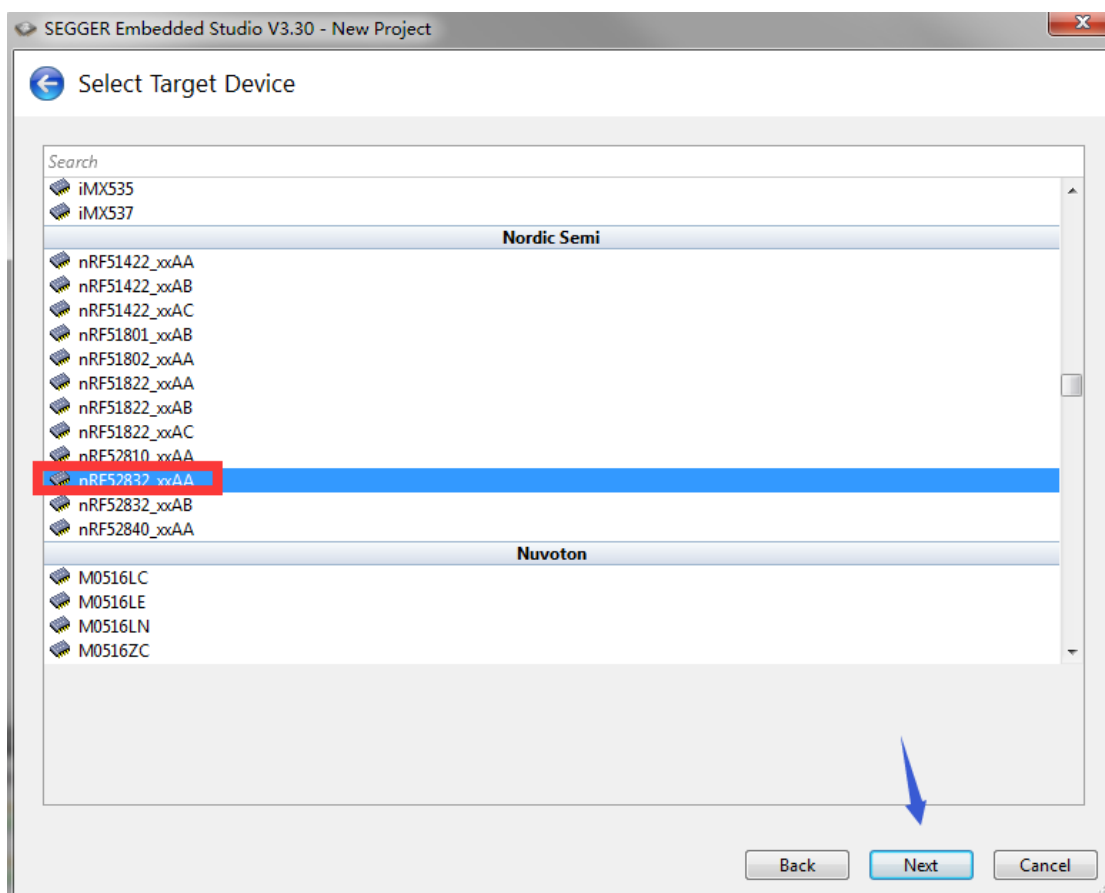
注意只有有左边有小箭头的地方才能打断点

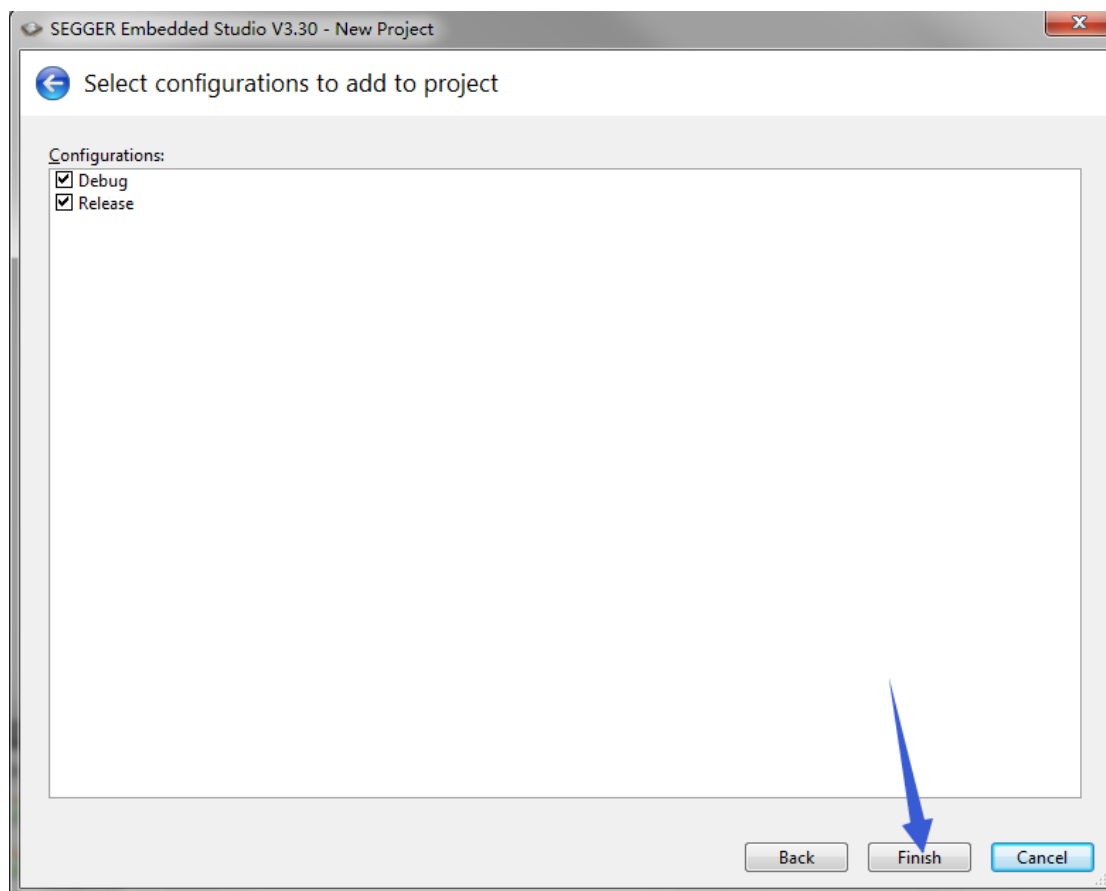
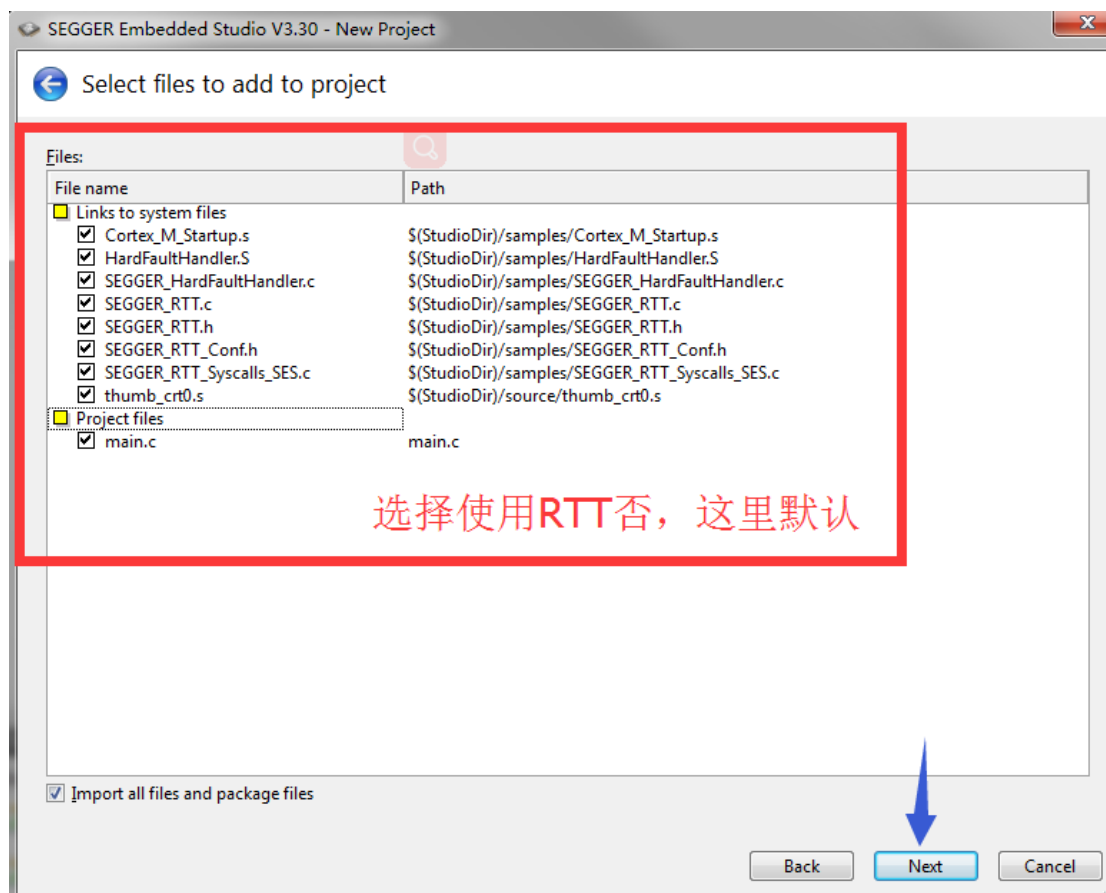
6 SES 新建工程

6.1 SES 新建最小工程

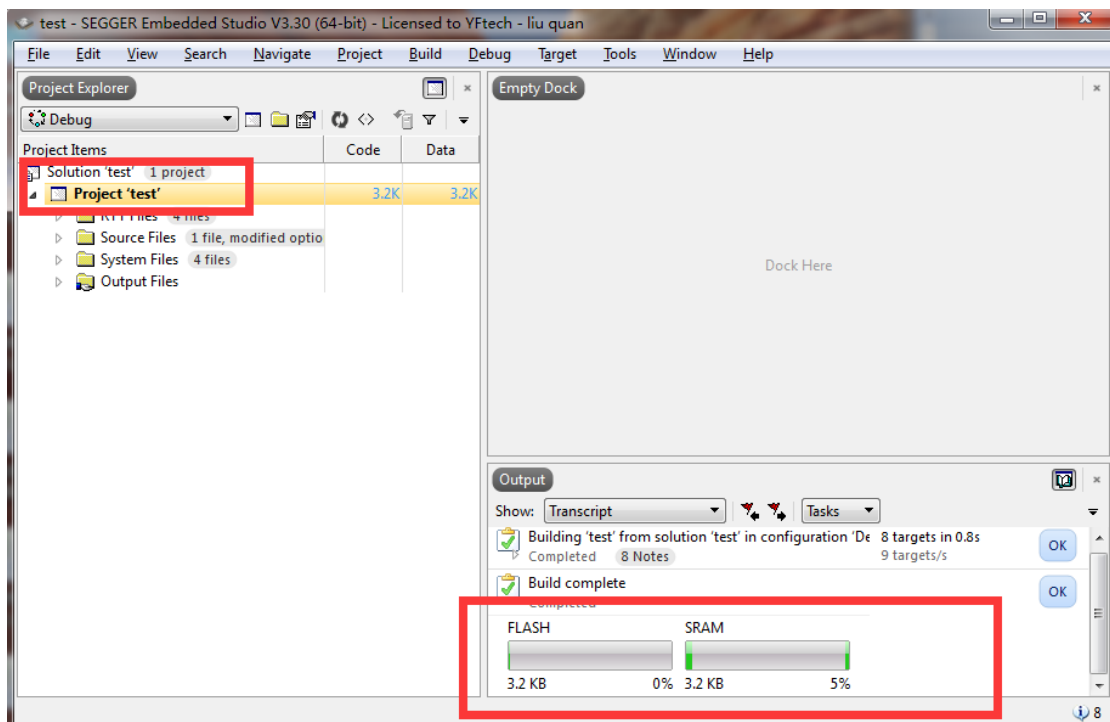






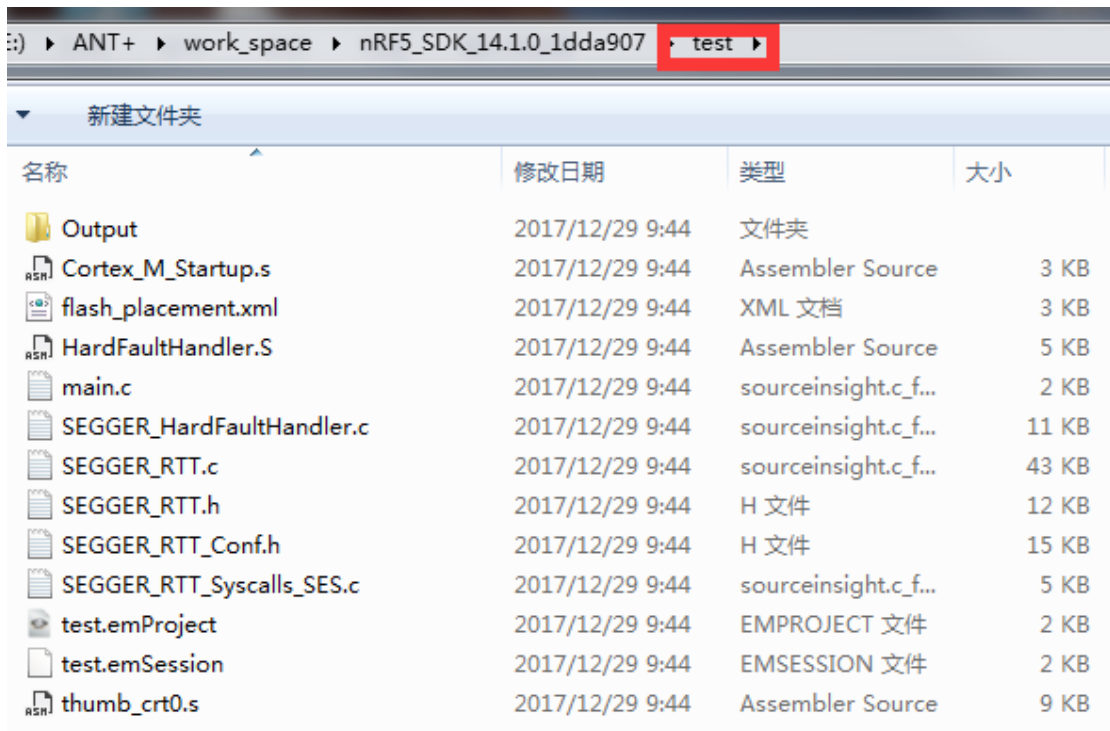


这样就建立完成了，可以直接编译

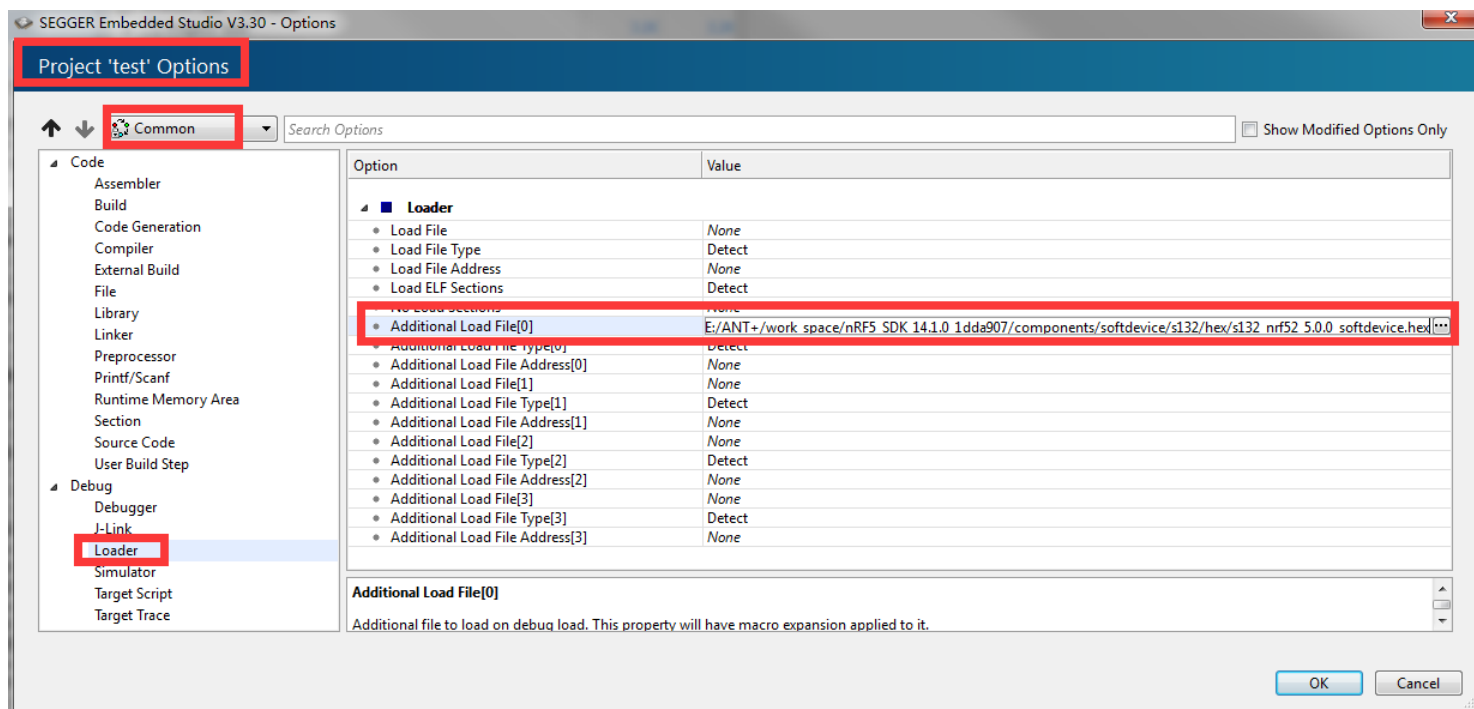


再对应工程配置修改一些配置就可以了。

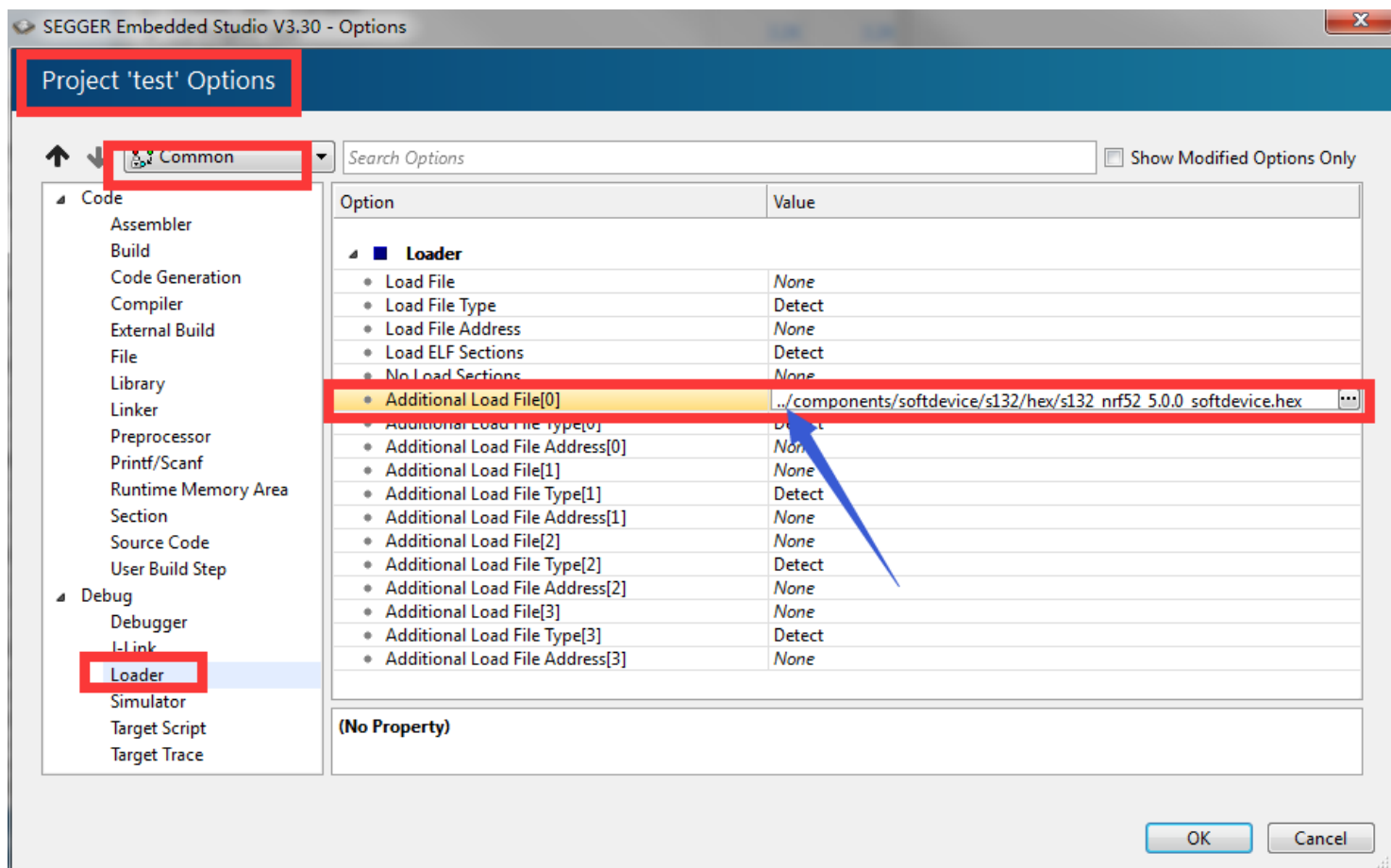
工程文件夹：



6.2 添加协议栈



上面用的是绝对路径，这样工程移动就会出问题，所以可以改为相对于工程目录的相对路径：



6.3 修改资源配置文件 flash_placement.xml

这个文件可以直接使用 nordic 提供的，因为是对协议栈的资源配置，也不知道它里面干了啥！直接用吧！下面贴出官方新建时的 flash_placement.xml 和 nordic 提供的 flash_placement.xml

6.3.1 官方新建时的 flash_placement.xml

```
<!DOCTYPE Linker_Placement_File>
<Root name="Flash Section Placement">
  <MemorySegment name="$(FLASH_NAME:FLASH)">
    <ProgramSection alignment="0x100" load="Yes" name=".vectors" start="$(FLASH_START:)" />
    <ProgramSection alignment="4" load="Yes" name=".init" />
    <ProgramSection alignment="4" load="Yes" name=".init_rodata" />
    <ProgramSection alignment="4" load="Yes" name=".text" />
    <ProgramSection alignment="4" load="Yes" name=".ctors" />
    <ProgramSection alignment="4" load="Yes" name=".dtors" />
    <ProgramSection alignment="4" load="Yes" name=".rodata" />
    <ProgramSection alignment="4" load="Yes" name=".ARM.exidx" address_symbol="__exidx_start" end_symbol="__exidx_end" />
    <ProgramSection alignment="4" load="Yes" runin=".fast_run" name=".fast" />
    <ProgramSection alignment="4" load="Yes" runin=".data_run" name=".data" />
    <ProgramSection alignment="4" load="Yes" runin=".tdata_run" name=".tdata" />
  </MemorySegment>
  <MemorySegment name="$(RAM_NAME:RAM);SRAM">
    <ProgramSection alignment="0x100" load="No" name=".vectors_ram" start="$(RAM_START:$(SRAM_START:))" />
    <ProgramSection alignment="4" load="No" name=".fast_run" />
    <ProgramSection alignment="4" load="No" name=".data_run" />
    <ProgramSection alignment="4" load="No" name=".tdata_run" />
    <ProgramSection alignment="4" load="No" name=".bss" />
    <ProgramSection alignment="4" load="No" name=".tbss" />
    <ProgramSection alignment="4" load="No" name=".non_init" />
    <ProgramSection alignment="4" size="__HEAPSIZE__" load="No" name=".heap" />
    <ProgramSection alignment="8" size="__STACKSIZE__" load="No" place_from_segment_end="Yes" name=".stack" />
    <ProgramSection alignment="8" size="__STACKSIZE_PROCESS__" load="No" name=".stack_process" />
  </MemorySegment>
  <MemorySegment name="$(FLASH2_NAME:FLASH2)">
    <ProgramSection alignment="4" load="Yes" name=".text2" />
    <ProgramSection alignment="4" load="Yes" name=".rodata2" />
    <ProgramSection alignment="4" load="Yes" runin=".data2_run" name=".data2" />
  </MemorySegment>
  <MemorySegment name="$(RAM2_NAME:RAM2)">
    <ProgramSection alignment="4" load="No" name=".data2_run" />
    <ProgramSection alignment="4" load="No" name=".bss2" />
  </MemorySegment>
</Root>
```

6.3.2 nordic 提供的 flash_placement.xml

```
<!DOCTYPE Linker_Placement_File>
<Root name="Flash Section Placement">
  <MemorySegment name="FLASH" start="$ (FLASH_PH_START)" size="$ (FLASH_PH_SIZE)">
    <ProgramSection load="no" name=".reserved_flash" start="$ (FLASH_PH_START)" size="$ (FLASH_START)-$ (FLASH_PH_START)" />
    <ProgramSection alignment="0x100" load="Yes" name=".vectors" start="$ (FLASH_START)" />
    <ProgramSection alignment="4" load="Yes" name=".init" />
    <ProgramSection alignment="4" load="Yes" name=".init_rodata" />
    <ProgramSection alignment="4" load="Yes" name=".text" />
    <ProgramSection alignment="4" keep="Yes" load="Yes" name=".sdh_soc_observers" inputsections="*(SORT(.sdh_soc_observers*))" address_symbol="__start_sdh_soc_observers"
end_symbol="__stop_sdh_soc_observers" />
    <ProgramSection alignment="4" keep="Yes" load="Yes" name=".pwr_mgmt_data" inputsections="*(SORT(.pwr_mgmt_data*))" address_symbol="__start_pwr_mgmt_data"
end_symbol="__stop_pwr_mgmt_data" />
    <ProgramSection alignment="4" keep="Yes" load="Yes" name=".sdh_ble_observers" inputsections="*(SORT(.sdh_ble_observers*))" address_symbol="__start_sdh_ble_observers"
end_symbol="__stop_sdh_ble_observers" />
    <ProgramSection alignment="4" keep="Yes" load="Yes" name=".log_const_data" inputsections="*(.log_const_data*)" address_symbol="__start_log_const_data" end_symbol="__stop_log_const_data" />
    <ProgramSection alignment="4" keep="Yes" load="Yes" name=".sdh_state_observers" inputsections="*(SORT(.sdh_state_observers*))" address_symbol="__start_sdh_state_observers"
end_symbol="__stop_sdh_state_observers" />
    <ProgramSection alignment="4" keep="Yes" load="Yes" name=".sdh_stack_observers" inputsections="*(SORT(.sdh_stack_observers*))" address_symbol="__start_sdh_stack_observers"
end_symbol="__stop_sdh_stack_observers" />
    <ProgramSection alignment="4" keep="Yes" load="Yes" name=".sdh_req_observers" inputsections="*(SORT(.sdh_req_observers*))" address_symbol="__start_sdh_req_observers"
end_symbol="__stop_sdh_req_observers" />
    <ProgramSection alignment="4" keep="Yes" load="Yes" name=".cli_command" inputsections="*(.cli_command*)" address_symbol="__start_cli_command" end_symbol="__stop_cli_command" />
    <ProgramSection alignment="4" keep="Yes" load="No" name=".nrf_sections" address_symbol="__start_nrf_sections" />
    <ProgramSection alignment="4" keep="Yes" load="Yes" name=".log_dynamic_data" inputsections="*(.log_dynamic_data*)" runin=".log_dynamic_data_run"/>
    <ProgramSection alignment="4" keep="Yes" load="Yes" name=".fs_data" inputsections="*(.fs_data*)" runin=".fs_data_run"/>
    <ProgramSection alignment="4" keep="Yes" load="Yes" name=".cli_sorted_cmd_ptrs" inputsections="*(.cli_sorted_cmd_ptrs*)" runin=".cli_sorted_cmd_ptrs_run"/>
```

```

<ProgramSection alignment="4" load="Yes" name=".dtors" />
<ProgramSection alignment="4" load="Yes" name=".ctors" />
<ProgramSection alignment="4" load="Yes" name=".rodata" />
<ProgramSection alignment="4" load="Yes" name=".ARM.exidx" address_symbol="__exidx_start" end_symbol="__exidx_end" />
<ProgramSection alignment="4" load="Yes" runin=".fast_run" name=".fast" />
<ProgramSection alignment="4" load="Yes" runin=".data_run" name=".data" />
<ProgramSection alignment="4" load="Yes" runin=".tdata_run" name=".tdata" />
</MemorySegment>
<MemorySegment name="RAM" start="$({RAM_PH_START})" size="$({RAM_PH_SIZE})">
  <ProgramSection load="no" name=".reserved_ram" start="$({RAM_PH_START})" size="$({RAM_START})-${({RAM_PH_START})}" />
  <ProgramSection alignment="0x100" load="No" name=".vectors_ram" start="$({RAM_START})" address_symbol="__app_ram_start__" />
  <ProgramSection alignment="4" keep="Yes" load="No" name=".nrf_sections_run" address_symbol="__start_nrf_sections_run" />
  <ProgramSection alignment="4" keep="Yes" load="No" name=".log_dynamic_data_run" address_symbol="__start_log_dynamic_data" end_symbol="__stop_log_dynamic_data" />
  <ProgramSection alignment="4" keep="Yes" load="No" name=".fs_data_run" address_symbol="__start_fs_data" end_symbol="__stop_fs_data" />
  <ProgramSection alignment="4" keep="Yes" load="No" name=".cli_sorted_cmd_ptrs_run" address_symbol="__start_cli_sorted_cmd_ptrs" end_symbol="__stop_cli_sorted_cmd_ptrs" />
  <ProgramSection alignment="4" keep="Yes" load="No" name=".nrf_sections_run_end" address_symbol="__end_nrf_sections_run" />
  <ProgramSection alignment="4" load="No" name=".fast_run" />
  <ProgramSection alignment="4" load="No" name=".data_run" />
  <ProgramSection alignment="4" load="No" name=".tdata_run" />
  <ProgramSection alignment="4" load="No" name=".bss" />
  <ProgramSection alignment="4" load="No" name=".tbss" />
  <ProgramSection alignment="4" load="No" name=".non_init" />
  <ProgramSection alignment="4" size="__HEAPSIZE__" load="No" name=".heap" />
  <ProgramSection alignment="8" size="__STACKSIZE__" load="No" place_from_segment_end="Yes" name=".stack" address_symbol="__StackLimit" end_symbol="__StackTop" />
  <ProgramSection alignment="8" size="__STACKSIZE_PROCESS__" load="No" name=".stack_process" />
</MemorySegment>
</Root>

```

6.4 计算应用程序的 ROM 和 RAM

在上节中可以看到，使用到了一些宏：

FLASH_PH_START

FLASH_PH_SIZE

RAM_PH_START

RAM_PH_SIZE

FLASH_START

FLASH_SIZE

RAM_START

RAM_SIZE

在 3.4.2 节中也提到了资源 ROM 和 RAM 的配置。

那么该如何配置协议栈呢？可以直接使用 nordic 官方工程的配置，也可以找到配置依据，这里使用的 SDK 是 14.1，对应的协议栈是 132 V5.0.0。可以看它的说明文档，路径是

\nRF5_SDK_14.1.0_1dda907\components\softdevice\s132\doc\

s132_nrf51822_5.0.0_release-notes.pdf:

SoftDevice properties

- An updated SoftDevice Specification document will be available at <http://infocenter.nordicsemi.com/>.
- This version of the SoftDevice contains the Master Boot Record (MBR) version 2.2.0 (DRGN-8852).
 - This version of the MBR is compatible with the previous versions.
- The combined MBR and SoftDevice memory requirements for this version are as follows:
 - Flash: **140 kB** (0x23000 bytes).
 - RAM: **5.18 kB** (0x14b8 bytes). This is the minimum required memory with the BLE stack enabled. The actual requirements depend on the configuration chosen at `sd_ble_enable()` time.

上面的文档是在 SDK 中的

所以 flash 就是 140kB(0x230000)

最小 RAM 是 5.18kB(0x14b8)，这个是最小的 RAM 大小，根据应用代码可以放大。

所以该如何设置呢！

保留 flash : `start="$(FLASH_PH_START)" size="$(FLASH_START)-$(FLASH_PH_START)"`

flash 的起始地址是 0x00000000，即 **FLASH_PH_START = 0x00000000**;

而 `size="$(FLASH_START)-$(FLASH_PH_START) = $(FLASH_START) - 0 = 0x23000`;所以 **FLASH_START = 0x23000**;

这个其实就是应用代码的 flash 的起始 flash 地址。

保留 ram: `start="$(RAM_PH_START)" size="$(RAM_START)-$(RAM_PH_START)"`

ram 的起始地址是 0x20000000，即 **RAM_PH_START = 0x20000000**;

而 `size="$(RAM_START)-$(RAM_PH_START) = $(RAM_START) - 0x20000000 = 0x14b0`;所以 **RAM_START**

=0x200014b0; 这个 ram 只能扩大，不能在缩小了，需要根据应用程序控制。

综上：

FLASH_PH_START	=0x0	//芯片 flash 的起始地址
FLASH_PH_SIZE	=0x80000	//芯片 flash 的总大小
RAM_PH_START	=0x20000000	//芯片 RAM 的起始地址
RAM_PH_SIZE	=0x10000	//芯片 RAM 的起始地址
FLASH_START	=0x23000	//应用代码 flash 的起始地址
FLASH_SIZE	=0x5d000	//应用代码 flash 的大小 0x80000-0x23000
RAM_START	=0x200014b0	//应用代码 RAM 的起始地址
RAM_SIZE	=0xdf20	//应用代码 RAM 的大小 0x80000-0x23000

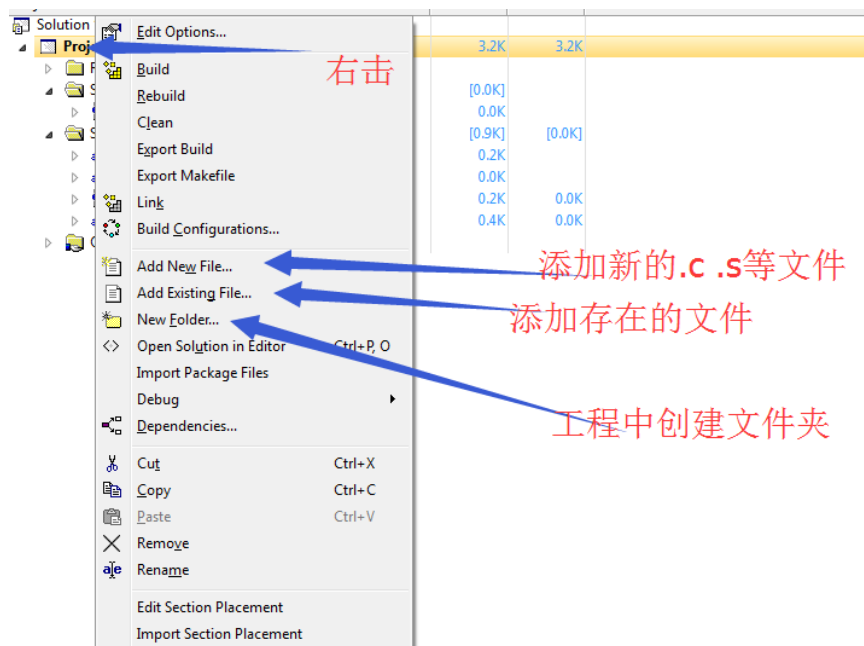
所以设置如下：

• Section Placement File	\$ProjectDir/flash_placement.xml	modified
• Section Placement Macros	FLASH_PH_START=0x0; FLASH_PH_SIZE=0x80000; RAM_PH_START=0x20000000; RAM_PH_SIZE=0x10000; FLASH...	

The screenshot shows the 'Common' tab in the Keil uVision IDE. The 'Linker' option is selected in the left-hand menu. The 'Section Placement File' is highlighted with a red box, and a red arrow points to its value, which is '\$(ProjectDir)/flash_placement.xml modified'.

Option	Value
Executable File Name	\$(OutDir)/\$(ProjectName)\$(EXE)
Additional Input Files	None
Use Manual Linker Script	No
Section Placement File	\$(ProjectDir)/flash_placement.xml modified
Section Placement Macros	FLASH_PH_START=0x0;FLASH_PH_SIZE=0x80000;RAM_PH_START=0x2000000
Section Placement Segments	None modified
Default Fill Pattern	None
DebugIO Implementation	Default
Additional Output Format	hex modified
Additional Output File Gap Fill Value	0x00
Generate Map File	Yes
Entry Point	reset_handler
Linker Symbol Definitions	None
Keep Symbols	None
Strip Debug Information	No
Strip Symbols	No
Allow Multiple Symbol Definition	No
Use Indirect File	Yes
No Enum Size Warning	No
No Wide Char Size Warning	No
Suppress Warning on Mismatch	No
Treat Linker Warnings as Errors	Yes
Additional Linker Options	None
Additional Linker Options From File	None
Symbols File	None

6.6 工程创建文件夹、新建.c 或.s、添加资源文件



6.7 添加头文件

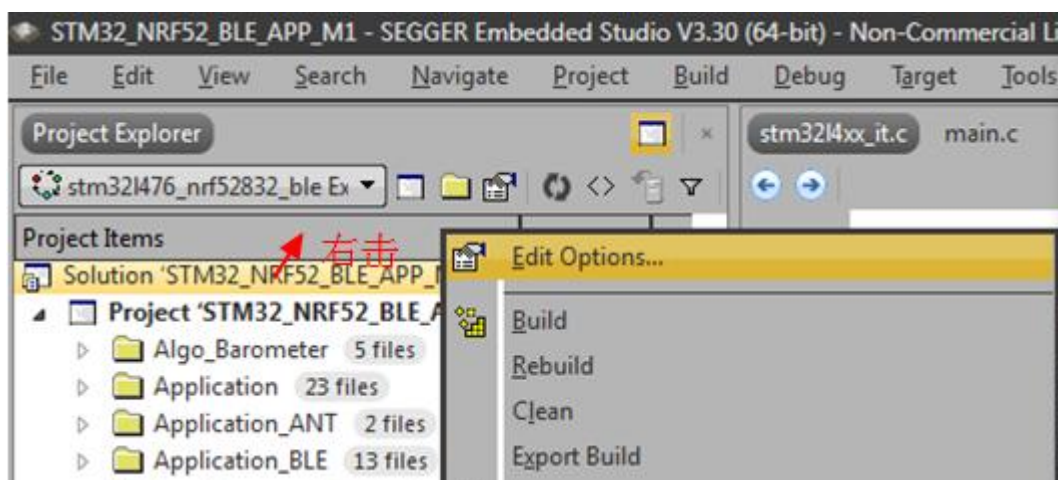
参考 [3.6.2 节](#)

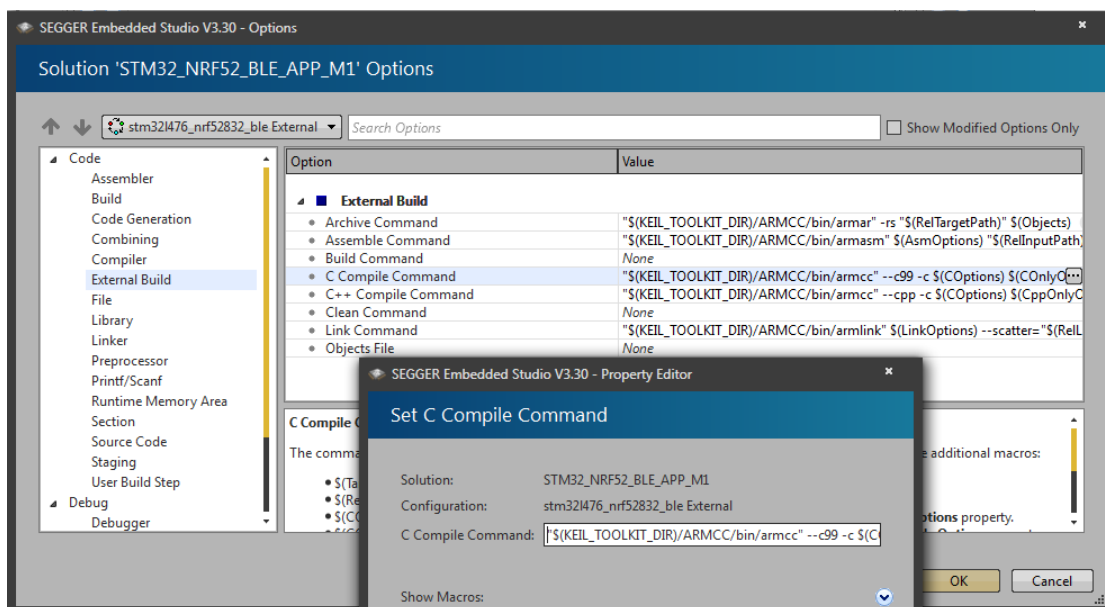
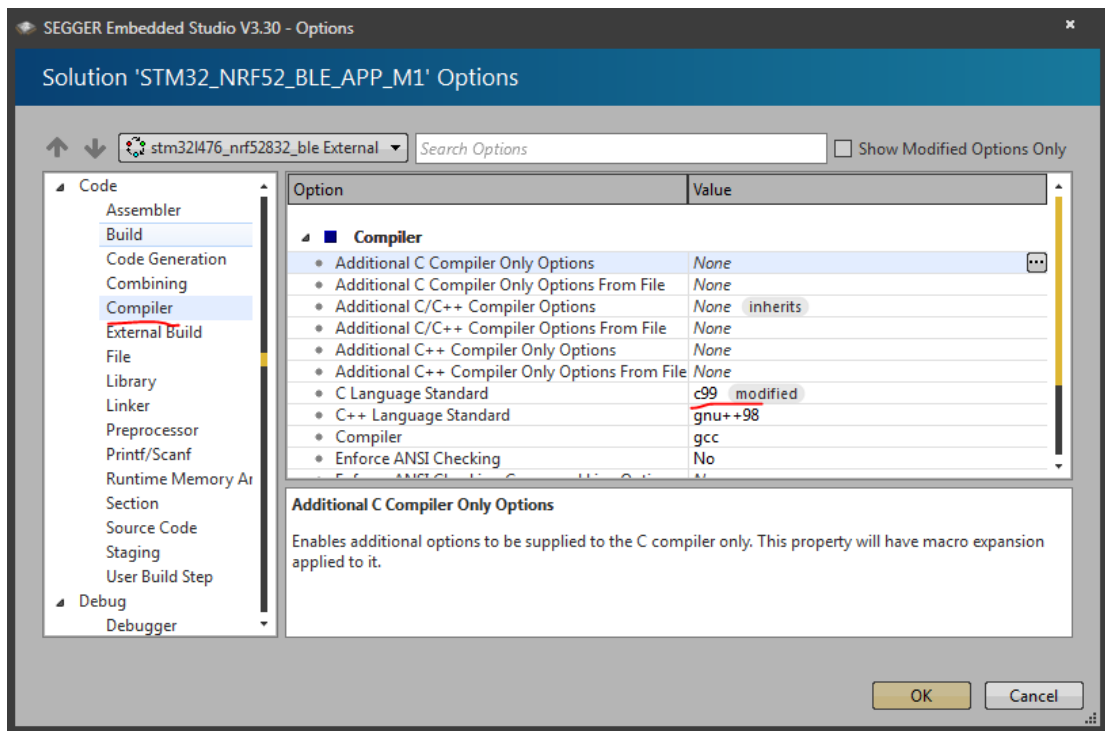
6.8 其他设置

参考 [第 3 节工程配置](#)

7 导入 keil 工程时使用 arm 编译器支持 C99 编译

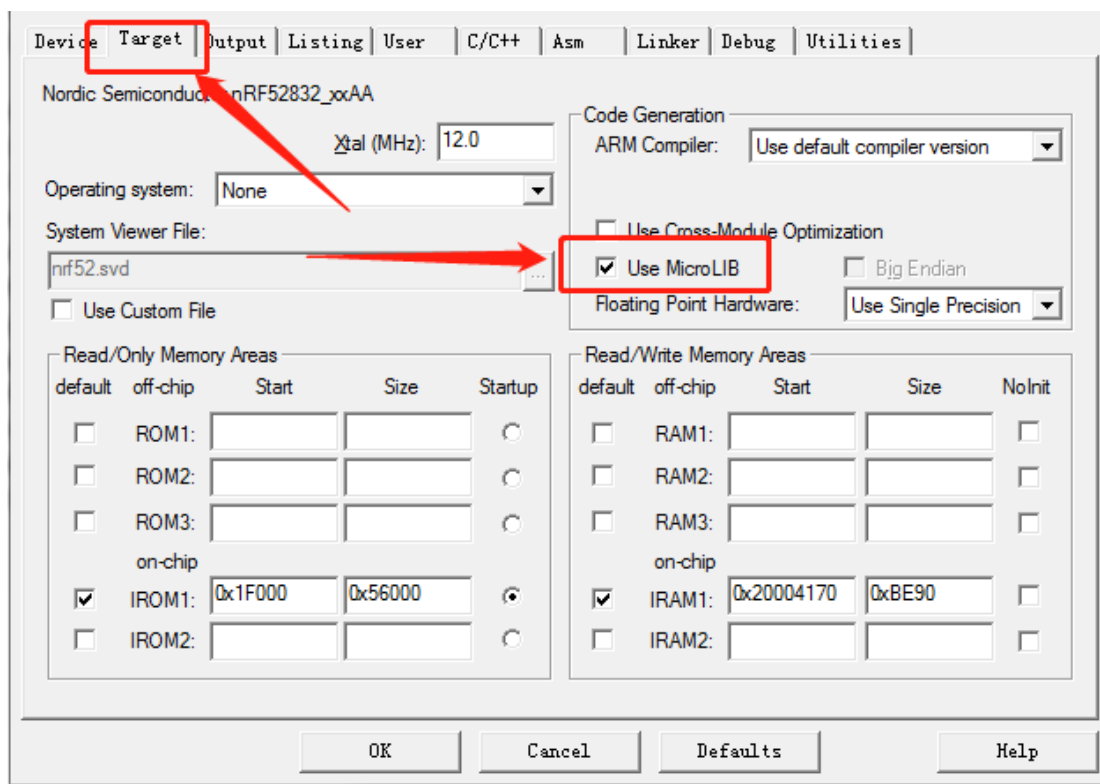
导入工程时会 SES 会询问使用外部编译器还是内部编译器，例如下面使用的外部编译器。



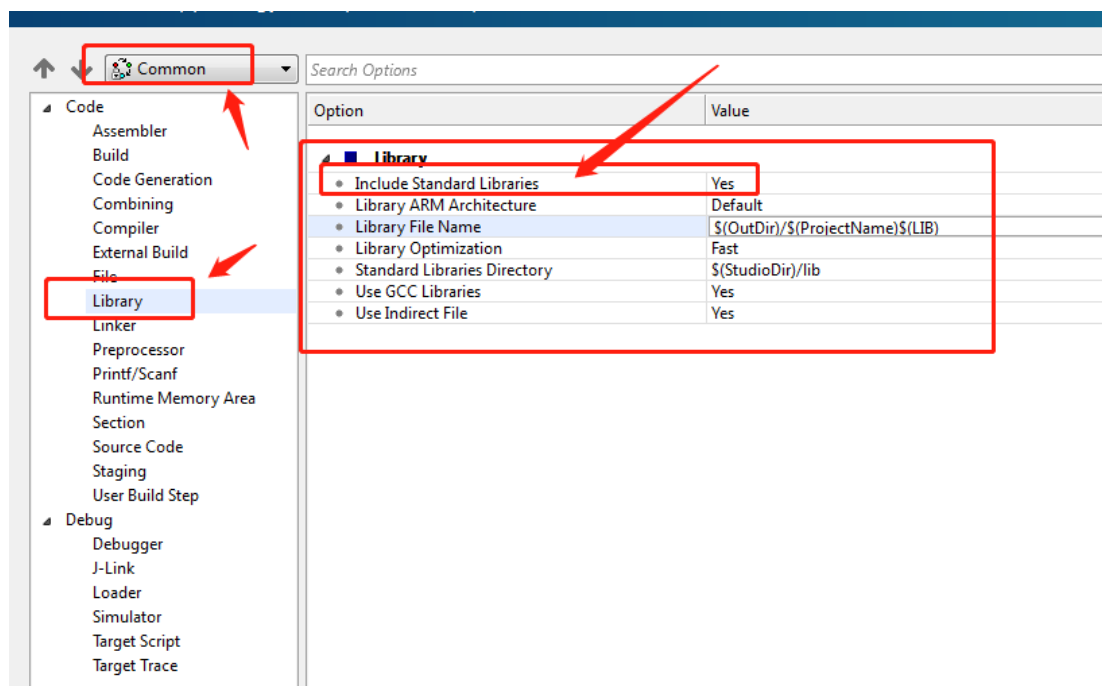


8 支持标准库

8.1 keil 设置

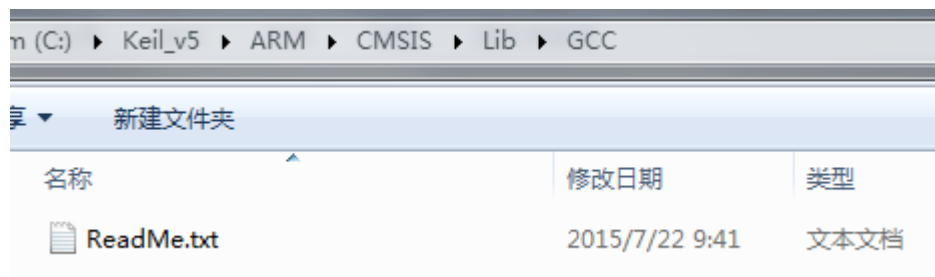
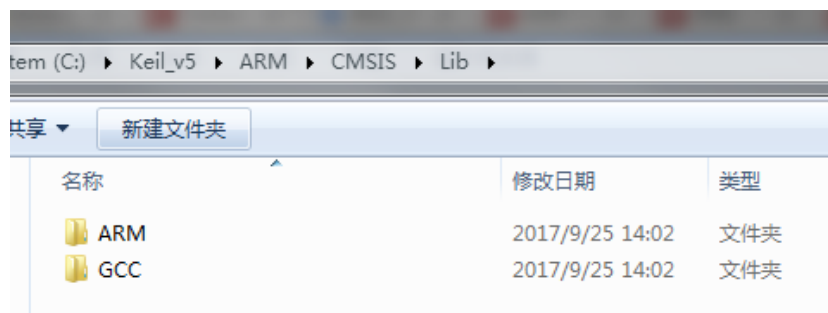
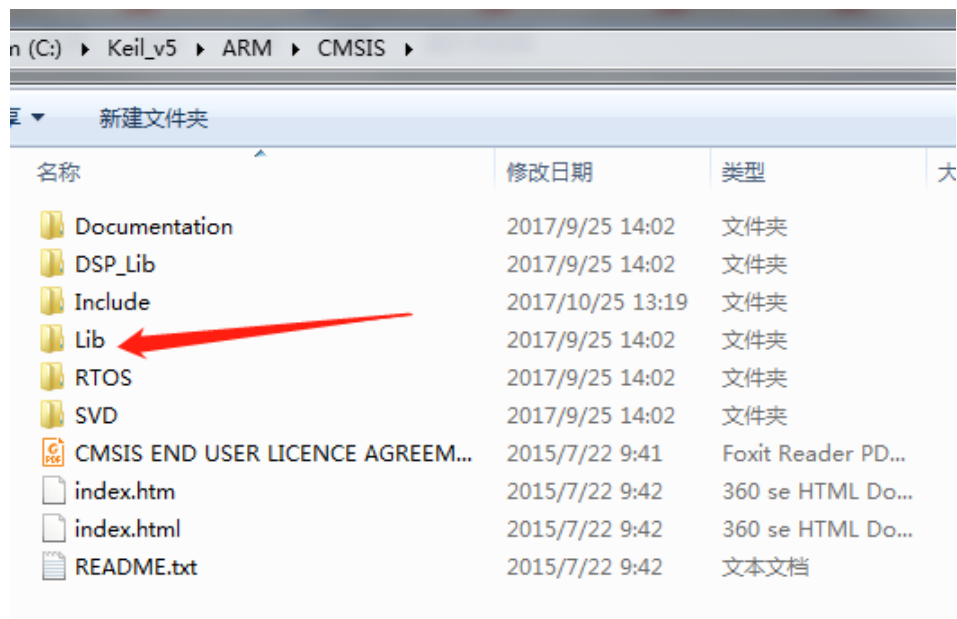


8.2 SES 设置



9 支持 CMSIS-DSP 库

首先在安装的 SES 根目录里有 LIB 文件夹，但是这个支持标准库使用的。并没有对 CMSIS-DSP 的支持，在 keil 的中可以找到目录



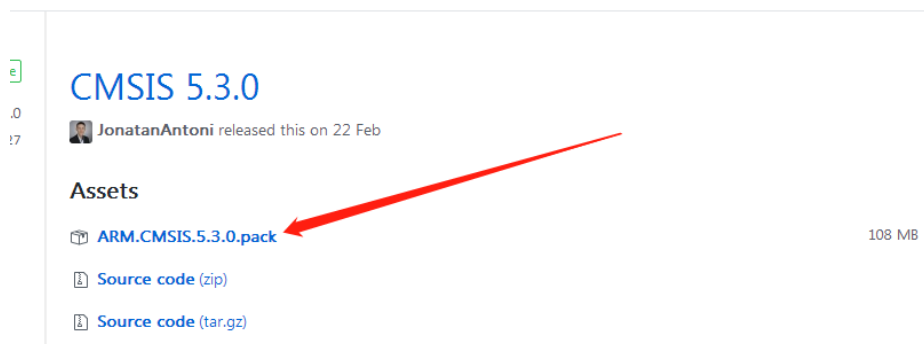
发现 GCC 里面是空的，所以需要下载 CMSIS 包。

9.1 CMSIS 包的下载路径

有两个路径：

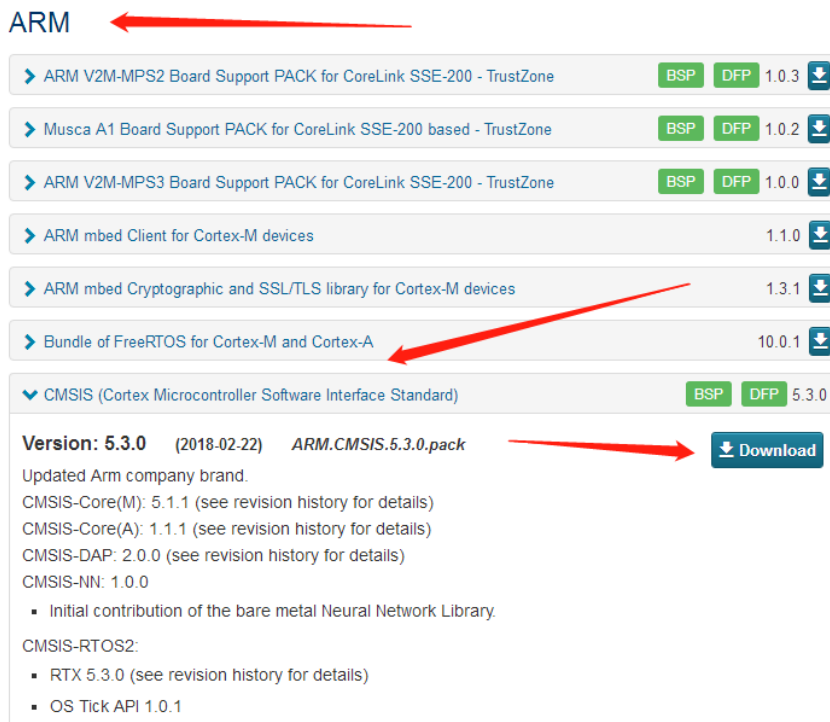
- arm github
https://github.com/ARM-software/CMSIS_5/releases
- arm 官网
<http://www.keil.com/dd2/Pack/>

9.1.1 github 下载



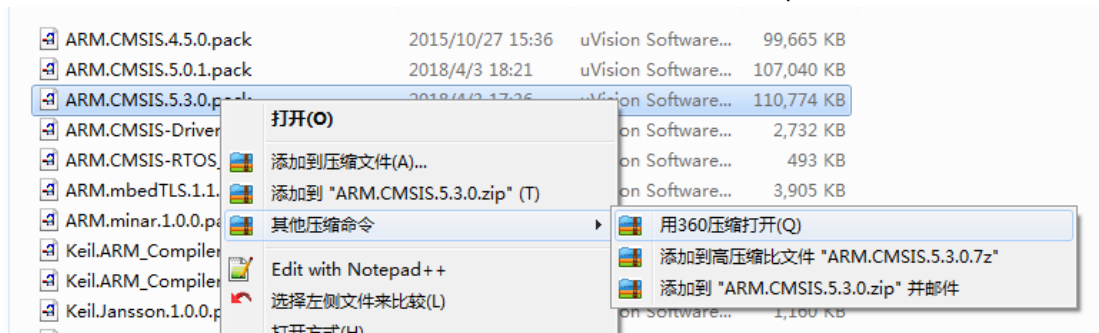
不要下载 ZIP，因为头文件目录没有，直接下载 pack

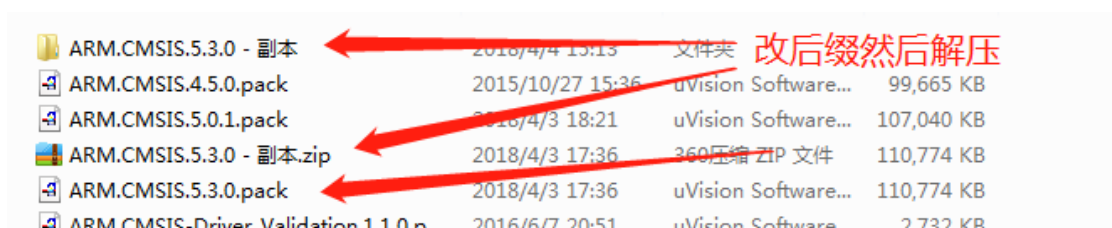
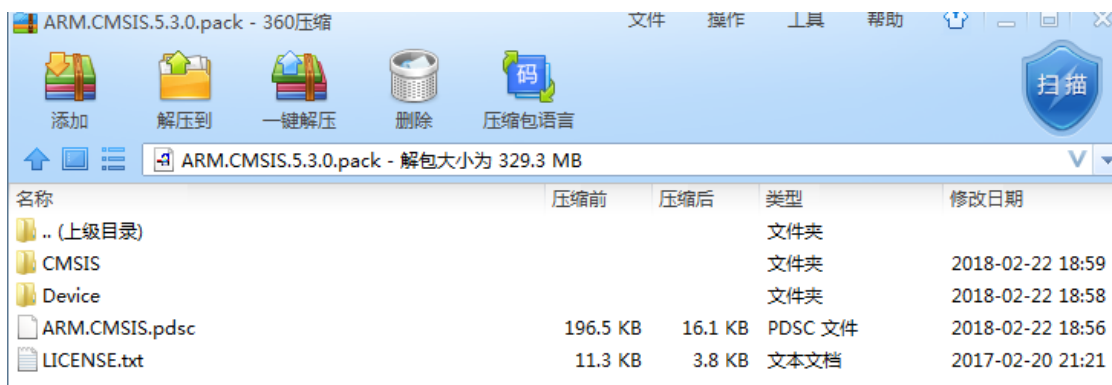
9.1.2 arm 官网下载



9.2 解压 CMSIS pack

这个 pack 其实就是一个压缩包，可以直接将其解压为 zip 文件，或者直接将.pack 后缀改为.ZIP 然后进行解压。也可以直接通过 360 压缩工具打开.pack 文件。

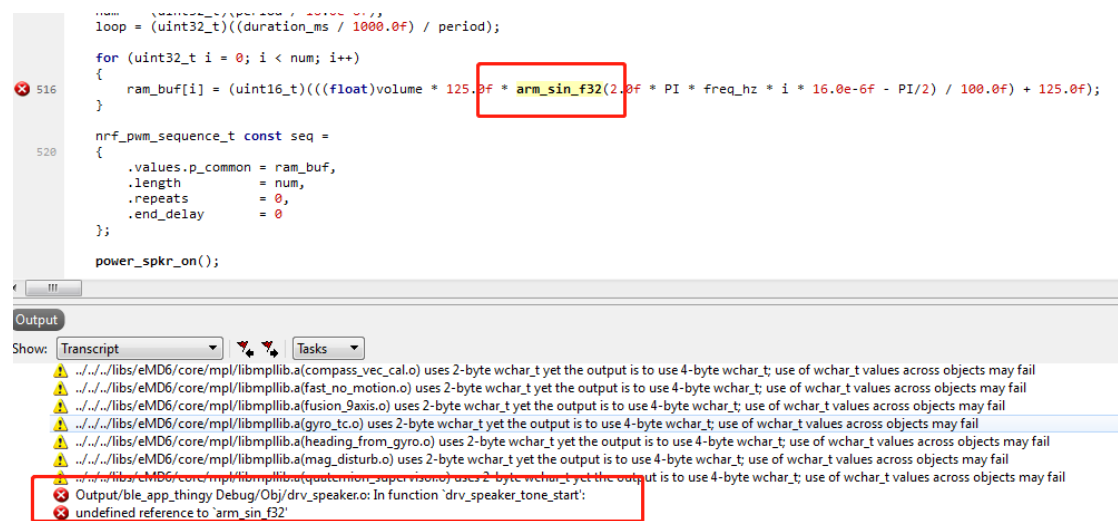




9.3 SES 使用浮点运算

这里将解压的 CMSIS 文件夹放到你的工程目录中, 需要包含一些头文件以及一些.a 的库文件。

在编译 nordic 的 thingy 工程时, 因为官方 demo 没有提供 SES 的工程, 所以按照 keil 新建工程后编译出现如下错误:

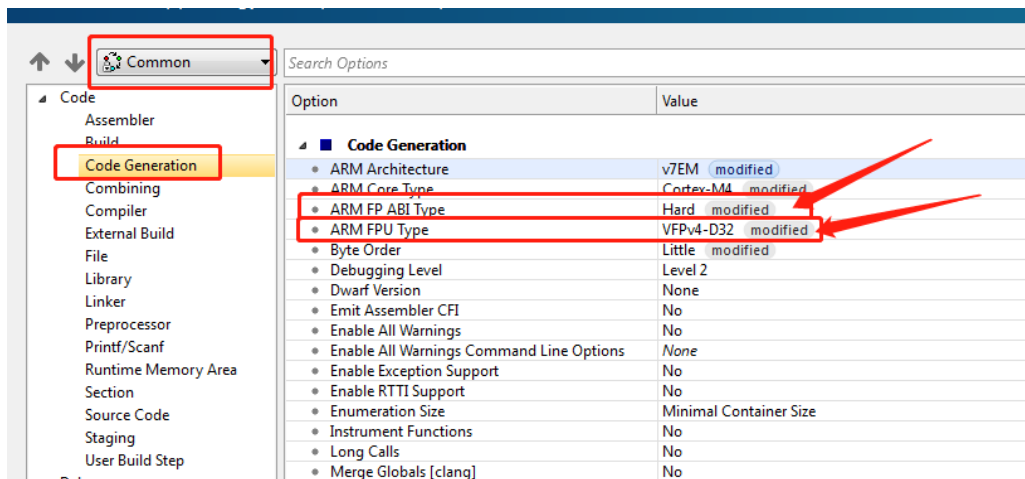


也就是一个 sin 函数没有定义, 在 keil 中是直接调用库文件: arm_cortexM4lf_math.lib 这里没有这个文件。所以需要添加这个文件以及进行一些相关配置。

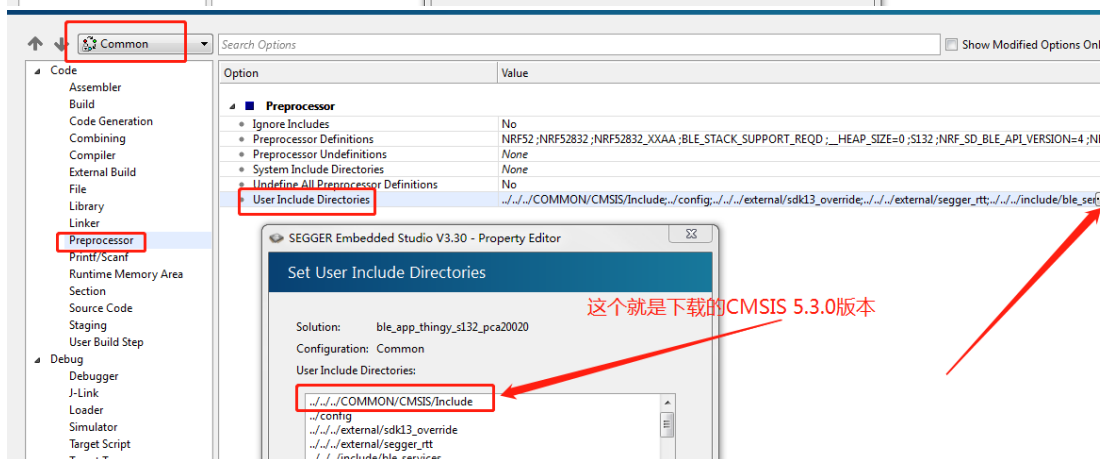
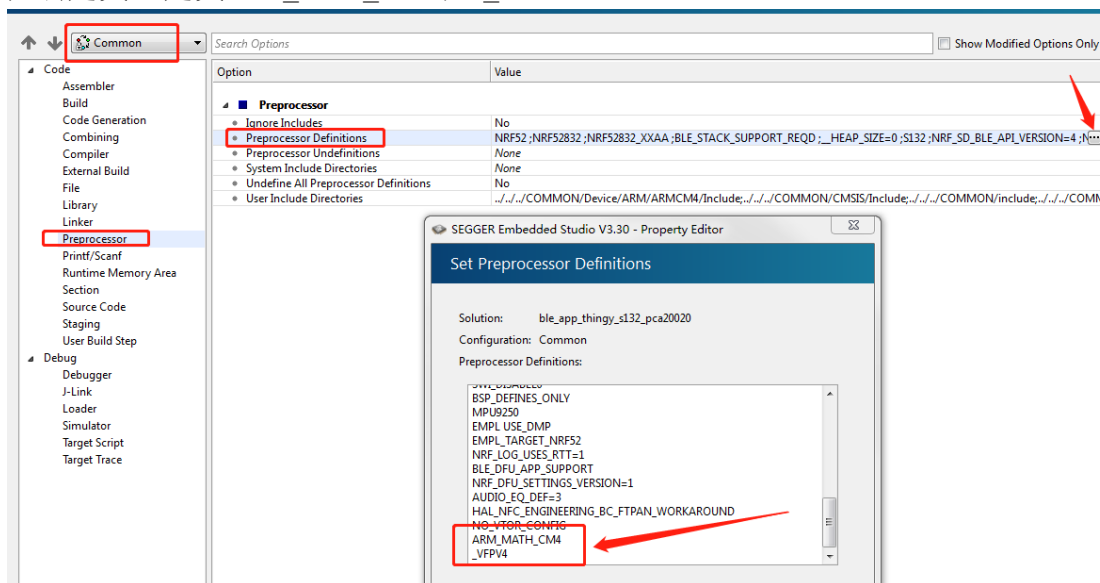
注意, SES 的 CMSIS 的 DSP 库需要添加到工程中, 他貌似不会自动添加这些.a。即使将这些.a 文件放到 SES 的安装目录中的 LIB 中也不会自动添加, 所以需要手动添加到工程, 以及包含相应的头文件。

9.3.1 使用硬浮点算法库

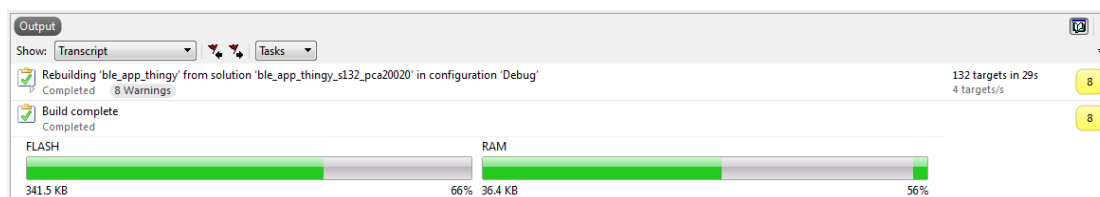
设置 SES 相关信息



在宏定义，定义 ARM_MATH_CM4 和 _VFPV4

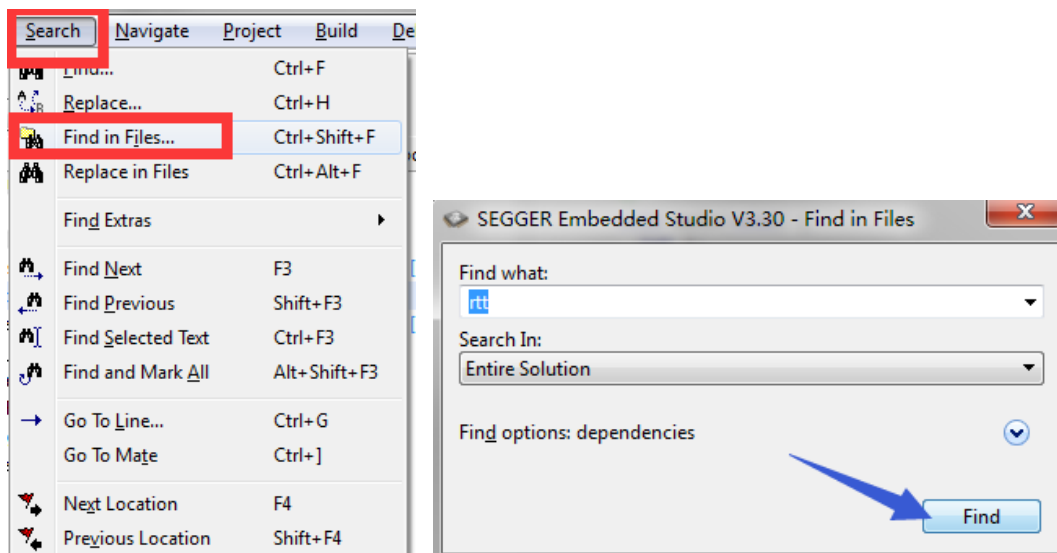


编译:

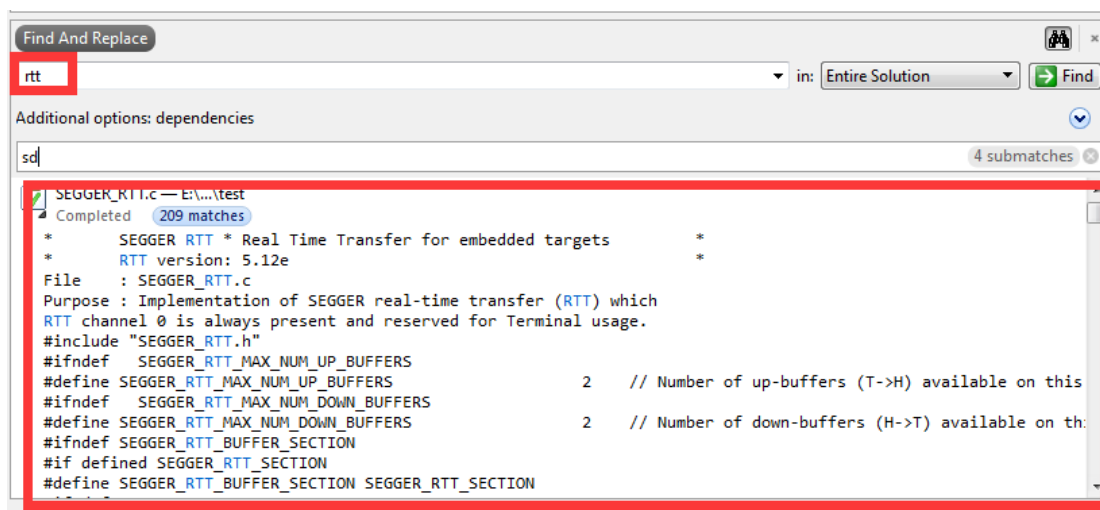


10 常用功能

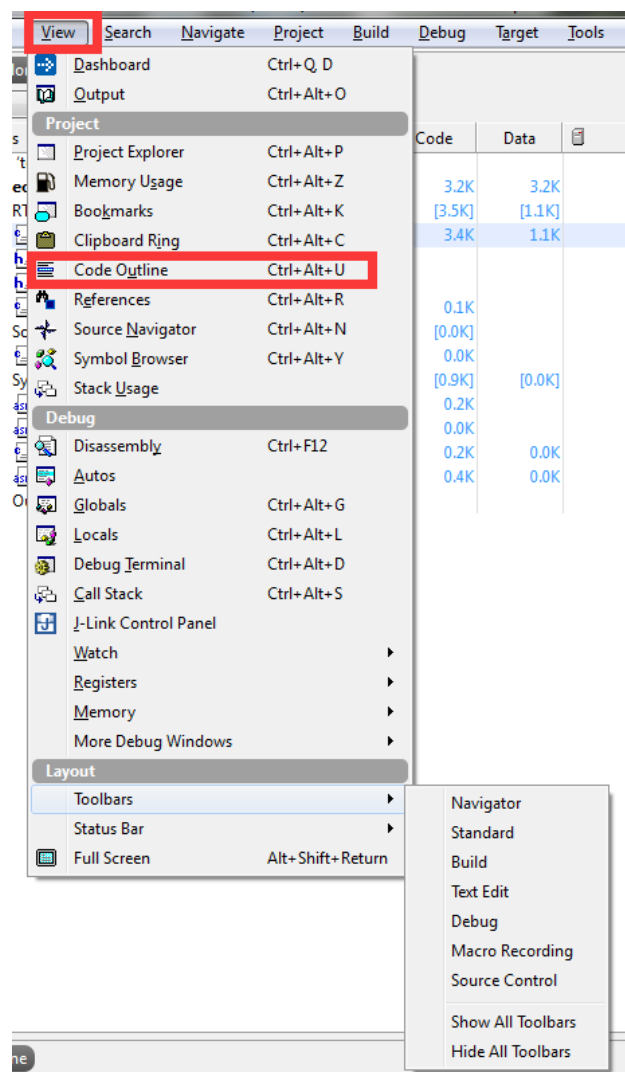
10.1 查找



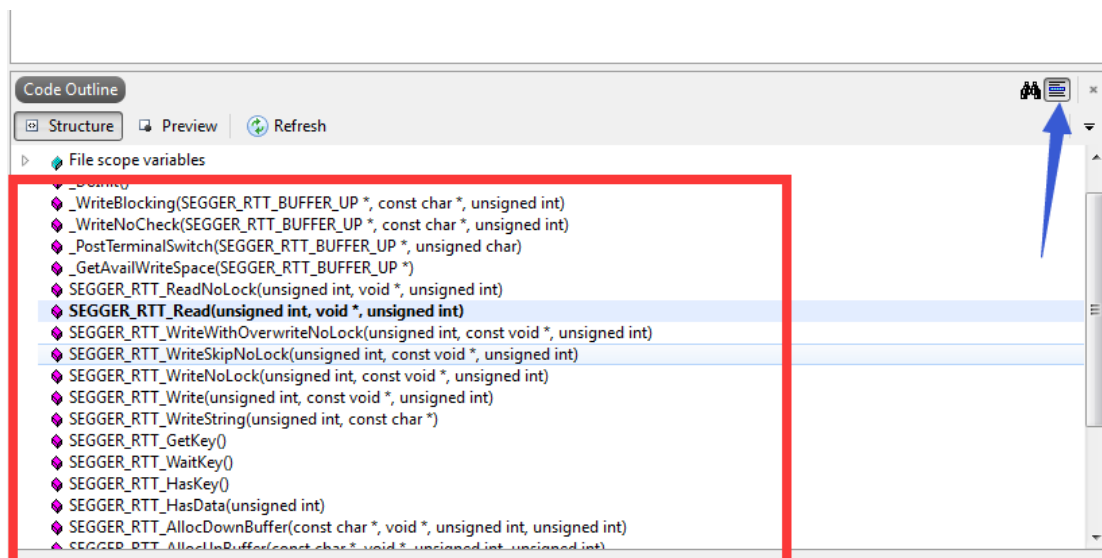
会在工程左下方中出现如下窗口：查找非常方便



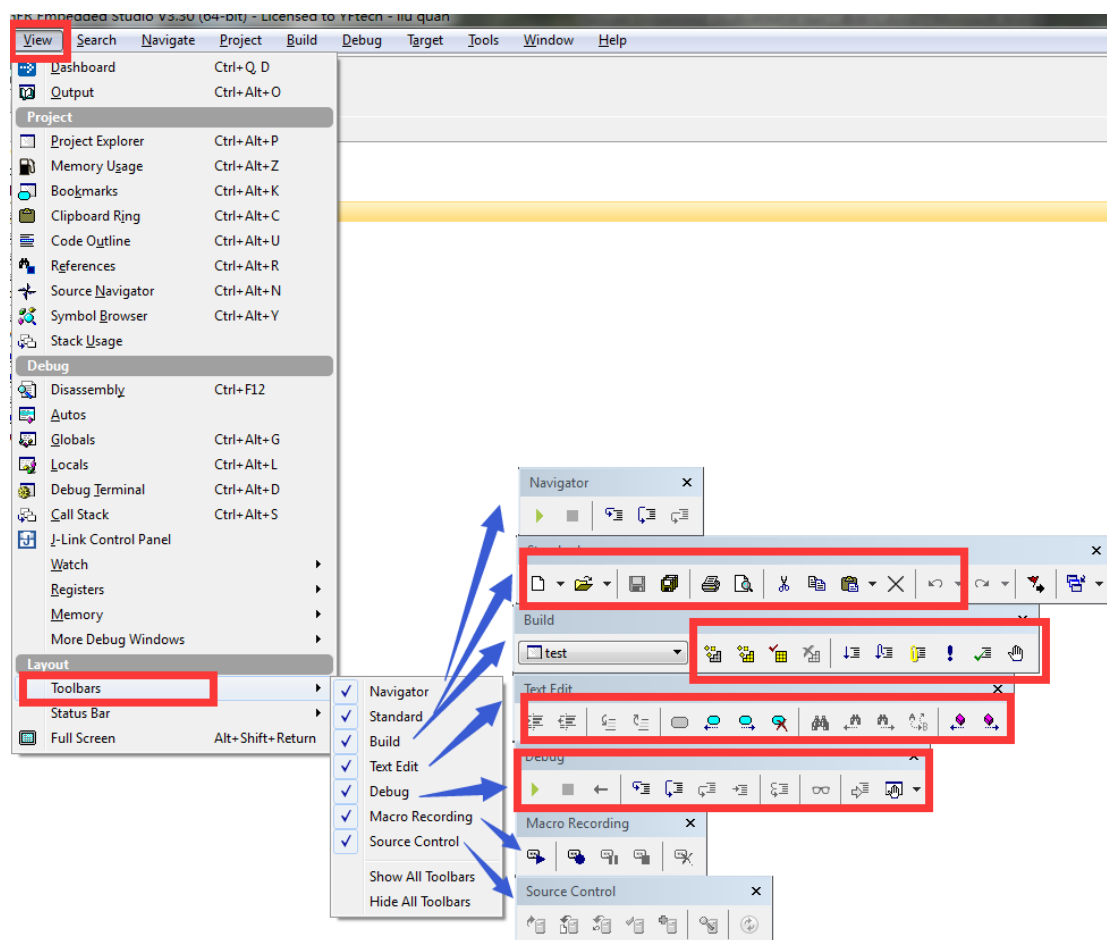
10.2 代码查看



上面 View 的所有的都可以进行点击看看，点击 code Outline，就会在左下方出现如下窗口



10.3 工具窗口



11 SES 调试深究

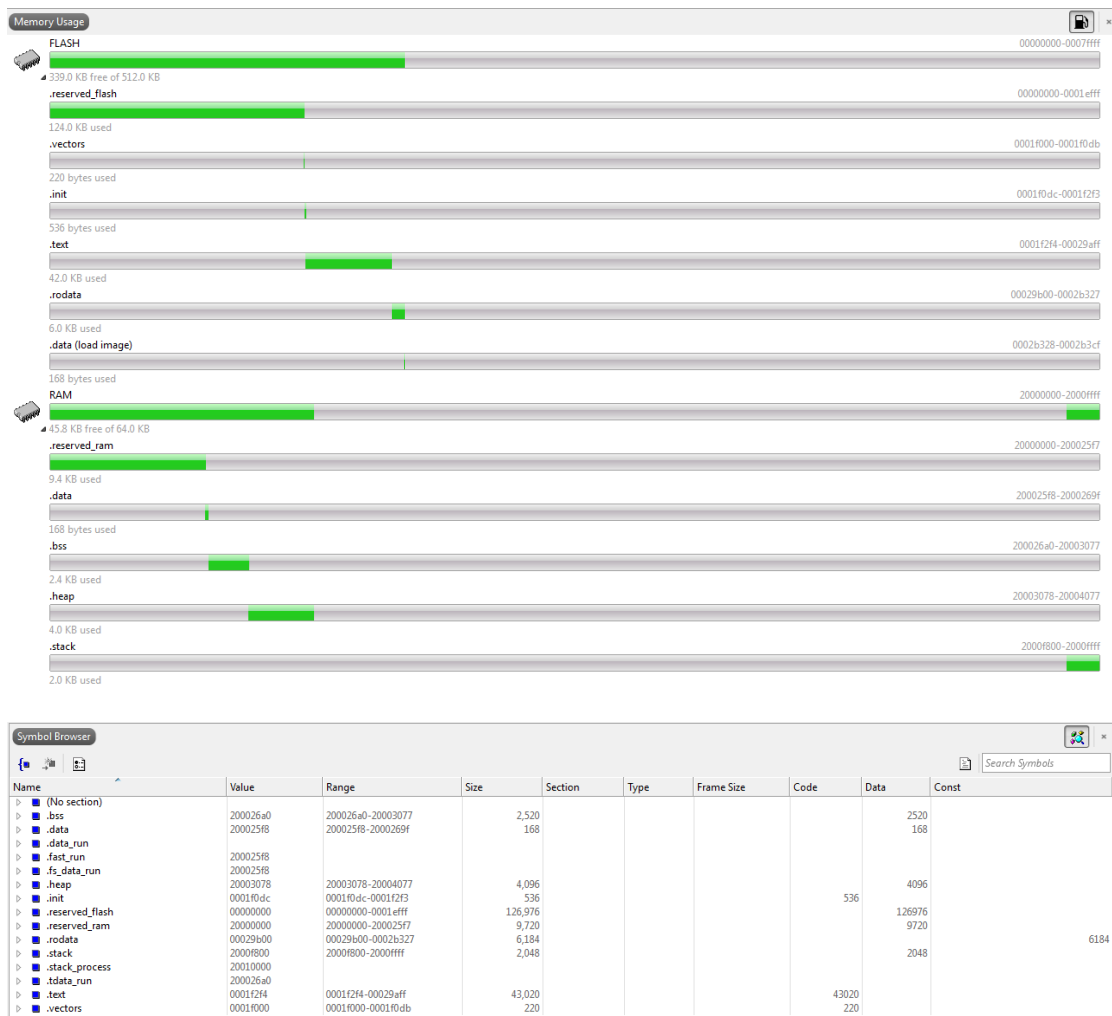
这里讨论 SES 的断点和寄存器查看

11.1 SES 断点

SES 的调试会根据设置的调试水平

12 程序中的段

先看一下 SES 编译出了存储和内存的使用情况的布局图



13 XML 文件详解(Section Placement file)

SES 使用的 XML 语法作为段定位文件。

文件的第一行为文档类型: <!DOCTYPE Linker_Placement_File>

第二行为文件根, 一个段定位文件只能有一个根: <Root name="Flash Section Placement">, Root 有一个名字属性。

在这个根下可以有多个内存段(MemorySegment), 每个内存段下可有多个程序段(ProgramSection), ProgramSection 的有一些属性, 都有一个名字属性。

13.1 ProgramSection 属性

ProgramSection 共有 *name*、*start*、*address_symbol*、*end_symbol*、*alignment*、*fill*、*inputsections*、*keep*、*load*、*place_from_segment_end*、*runin*、*runoffset*、*size*、*size_symbol* 和 *start* 等属性, 当然这些属性有些是可选的, 有些是必须的。

13.1.1 name 名字属性

名字属性在 Root、MemorySegment 以及 ProgramSection 都有的一个元素, 且这个属性是必须的, 使用方法是 *name*="xxx", 这个名字需要唯一, 在一个段定位文件中。双引号中不能以数组开头, 一般以字母或者.xxx。例如:

```
<Root name="Flash Section Placement">
  <MemorySegment name="FLASH" start="$ (FLASH_PH_START)" size="$ (FLASH_PH_SIZE)">
    <ProgramSection alignment="0x100" load="Yes" name=".vectors" start="$ (FLASH_START)" />
```

13.1.2 start 起始地址属性

start 属性是一个 MemorySegment 以及 ProgramSection 的属性, 后面双引号中的值必须是以 0x 开头的 16 进制数据。表示这个 *name* 属性段的起始地址是多少。例如 *start*="0x20010000", 或者 *start*="\$ (FLASH_PH_START)", 而 FLASH_PH_START 是一个 16 进制的宏。

13.1.3 size 段具体大小属性

这个用来表明 *name* 段具体的大小, 以字节为单位。它的值必须是以 0x 开头的 16 进制数据。

13.1.4 address_symbol 地址开始符号属性

address_symbol 为地址开始符号, 注意这里是符号, 和上面 *start* 的属性的不同在于, *start* 给是固定的地址, 而这里的 *address_symbol* 是表示 *name* 属性段的起始地址的符号, 这个符号是在编译链接的过程中用到, 并确定这个符号的值。例如: *address_symbol*="__StackLimit"

13.1.5 end_symbol 地址结束符号属性

end_symbol 和上面的 *address_symbol* 相对应, 也就是是 *name* 属性段的接收地址的符号。例如: *end_symbol*="__StackTop"

13.1.6 size_symbol 段大小符号属性

这个用来表示一个段大小的符号, 也就是 *end_symbol* - *address_symbol* 的值的符号。

13.1.7 alignment 访问对齐属性

alignment 对齐方式，以字节对齐，强制规定 name 属性段的以多数个字节对齐，使用方法：alignment="0x100"，双引号中的值必须是以 0x 开头的 16 进制数据。

13.1.8 fill 填充属性

这是一个可选的属性，用于填充内存中没有指定范围的值，使用方法：fill="0xff"，双引号中必须是以 0x 开头的 16 进制数据。

13.1.9 inputsections 输入哪些文件到 name 段

这个属性一般是不用声明的，当 SES 编译时会用到这个 XML 文件中的符号，而在链接时会生成连接文件，链接文件其实就是将各个文件按照编译出来的东西(代码、常量/赋值了的全局变量、为初始化的全局变量)，把这些文件进行组合放到指定的地方。而 inputsections 的作用是告诉链接文件，name 段里存放的只有特定的文件后缀或特定的文件名称，也就是通过 inputsections 来缩小存放到 name 段的文件。

一般情况先.text, .dtors, .ctors, .data, .rodata, 或 bss 是不允许使用 inputsections 进行属性限制的，这些段可放的文件名一般是*(.name.name.*)。例如：

```
__init_load_start__ = ALIGN(__vectors_end__, 4);
__init ALIGN(__vectors_end__, 4) : AT(ALIGN(__vectors_end__, 4))
{
    __init_start__ = .;
    *(.init .init.*)
}
```

(.init .init.)的*表示所有的.O 文件，括号里面的表示这个段中存放的是所有.O 文件中 section 名是.init 或者代码中带有.init.* (*也为通配符)的符号都放到这个段中。

这里的*(.init .init.*)就是编译器默认存放.init 段名的代码或者数据。假设我只想存放一个特定的段名或者特定的符号到相应的段中呢？？这个时候 inputsections 就起作用了，例如：

没有使用 inputsections 说明属性段生成的连接件。

```
<ProgramSection alignment="4" keep="Yes" load="Yes" name=".fs_data" runin=".fs_data_run"/>

__fs_data_load_start__ = ALIGN(__nrf_sections_end__, 4);
__fs_data ALIGN(__nrf_sections_run_end__, 4) : AT(ALIGN(__nrf_sections_end__, 4))
{
    __fs_data_start__ = .;
    KEEP(*(.fs_data .fs_data*))
}
```

使用 inputsections 说明属性段生成的连接件。

```
<ProgramSection alignment="4" keep="Yes" load="Yes" name=".fs_data" inputsections="*(.fs_data*)" runin=".fs_data_run"/>

__fs_data_load_start__ = ALIGN(__nrf_sections_end__, 4);
__fs_data ALIGN(__nrf_sections_run_end__, 4) : AT(ALIGN(__nrf_sections_end__, 4))
{
    __fs_data_start__ = .;
    KEEP(*(.fs_data*))
}
```

也就是指定了输入到这个段的段文件名为*(.fs_data*)。

13.1.10 keep 保持属性

keep 如果等于"YES"的作用是 name 属性段中的符号即使没有别程序代码调用，也将这

个 ProgramSection 属性保留。否则 NO 就是没有使用就丢弃，不在连接文件中体现。

13.1.11 load 加载属性

如果 load="YES", 这个段将会被加载到 RAM, 所以这个属性只能是 flash 可以设置为 YES, 如果是 RAM 内的程序段必须将这个属性设置为 NO。

13.1.12 place_from_segment_end 段末尾开始放数据属性

这个其实就是为了栈 stack 准备的属性, ARM 一般都是满减栈, 也就是栈地址向下生长。值是 YES 或者 NO, 默认其实就 NO。例如:

```
<ProgramSection alignment="8" size="__STACKSIZE__" load="No" place_from_segment_end="Yes"
name=".stack" address_symbol="__StackLimit" end_symbol="__StackTop"/>
```

13.1.13 runin 属性

这个用来将 flash 中 runin 定义的段名字拷贝到对应的 RAM 中的同名字的名字段。例如:

```
<!DOCTYPE Linker_Placement_File>
<Root name="Flash Section Placement">
  <MemorySegment name="FLASH" start="$(FLASH_PH_START)" size="$(FLASH_PH_SIZE)">
    .....
    <ProgramSection alignment="4" load="Yes" runin=".fast_run" name=".fast" />
    <ProgramSection alignment="4" load="Yes" runin=".data_run" name=".data" />
    <ProgramSection alignment="4" load="Yes" runin=".tdata_run" name=".tdata" />
    .....
  </MemorySegment>
  <MemorySegment name="RAM" start="$(RAM_PH_START)" size="$(RAM_PH_SIZE)">
    .....
    <ProgramSection alignment="4" load="No" name=".fast_run" />
    <ProgramSection alignment="4" load="No" name=".data_run" />
    <ProgramSection alignment="4" load="No" name=".tdata_run" />
    .....
  </MemorySegment>
</Root>
```

13.1.14 runoffset 属性

这个属性的作用是 name 段加载到 ram 中时, 加载的地址不是从 name 段的起始地址进行加载, 而是从起始地址的偏移量 runoffset 设置值进行偏移加载。这个值必须是以 0x 开头的 16 进制数据。

14 连接文件浅析

GNU 的链接文件的使用可以百度《ld 中文使用手册完全版》，这里只是说明链接文件中的符号的来源。

```
MEMORY
{
    UNPLACED_SECTIONS (wx) : ORIGIN = 0x100000000, LENGTH = 0
    RAM (wx) : ORIGIN = 0x20000000, LENGTH = 0x00010000
    FLASH (wx) : ORIGIN = 0x00000000, LENGTH = 0x00080000
}

SECTIONS
{
    __RAM_segment_start__ = 0x20000000;
    __RAM_segment_end__ = 0x20010000;
    __RAM_segment_size__ = 0x00010000;
    __FLASH_segment_start__ = 0x00000000;
    __FLASH_segment_end__ = 0x00080000;
    __FLASH_segment_size__ = 0x00080000;

    __HEAPSIZE__ = 4096;
    __STACKSIZE_PROCESS__ = 0;
    __STACKSIZE__ = 2048;

    __vectors_load_start__ = 0x0;
    .vectors 0x0 : AT(0x0)
    {
        __vectors_start__ = .;
        *(.vectors.vectors.*)
    }
    __vectors_end__ = __vectors_start__ + SIZEOF(.vectors);
    __vectors_size__ = SIZEOF(.vectors);
    __vectors_load_end__ = __vectors_end__;

    . = ASSERT(__vectors_start__ == __vectors_end__ || (__vectors_end__ >= __FLASH_segment_start__ &&
    __vectors_end__ <= __FLASH_segment_end__), "error: .vectors is too large to fit in FLASH memory segment");}
```