

青风带你玩蓝牙 nRF52832 系列教程.....	2
-----作者: 青风.....	2
作者: 青风.....	3
出品论坛: www.qfv8.com	3
淘宝店: http://qfv5.taobao.com	3
QQ 技术群: 346518370.....	3
硬件平台: 青云 QY-nRF52832 开发板.....	3
一: 开发环境 keil 的使用及工程建立.....	3
1 开发环境 KEIL5 简介.....	3
2 工程项目的建立.....	7
3 工程项目的仿真与调试.....	18
3.1 仿真工具的选择与设置:	18
3.2 项目仿真错误定位.....	20

青风带你玩蓝牙 nRF52832 系列教程

-----作者: 青风

出品论坛: www.qfv8.com 青风电子社区



作者: 青风**出品论坛: www.qfv8.com****淘宝店: <http://qfv5.taobao.com>****QQ 技术群: 346518370****硬件平台: 青云 QY-nRF52832 开发板**

一: 开发环境 keil 的使用及工程建立

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 μ Vision5 集成开发环境与 RealView 编译器。支持 ARM 全系列的内核处理器, 自动配置启动代码, 集成 Flash 烧写模块, 强大的 Simulation 设备模拟, 性能分析等功能, 与 ARM 之前的工具包 ADS 等相比, RealView 编译器的最新版本可将性能改善超过 20%。

1 开发环境 KEIL5 简介

KEIL MDK 的更新速度非常快, 目前已经更新到最新的 RVMDK 5.24 版本。5.0 以上版本的 MDK keil 对 IDE 界面、MDK core 编译器等方面都有巨大的改变。笔者对 MDK4 和 MDK5 进行了详细的对比。

1: 首先 KEIL5 的 SWD 下载速度最大可以提升到 50M (KEIL4 最大速度为 10M, 速度提升 5 倍, 下载程序只用一瞬间, 不管是做实验, 还是量产, 有效提升开发进度)。

2: 从 KEIL3 到 KEIL4, 你有没有明显的感觉到 KEIL 的体积越来越大? 例如, KEIL3.8 才 122M, 到了 KEIL4.6 已经 487M 了。这是为什么了?

这是因为单片机的种类过一段时间就会增加, 为了能够编译支持开发最新的芯片, MDK 不得不变得越来越大, 但是我们并不是都能用到里面的芯片。而 KEIL4 就是这种情况, 为了能够解决此问题, KEIL5 应运而生了。KEIL MDK5 这一次重大改进, 更加方便我们使用。与之前的版本相比, 最大的区别在于器件(Software Packs)与编译器(MDK core)分离。也就是说, 我们安装好编译器(mdk_5xx.exe)以后, 编译器里面没有任何器件。如果我们要对 nrf51822 进行开发, 只需要再下载 nrf51822 的器件安装包(packs)即可。

3: KEIL5 工程中可以在线生成启动文件 startup.s 文件和系统文件 system.c 文件, 这样就不用像 keil4 一样需要在工程中添加对应的文件。同时还可以通过 packs 包在工程中生成一些驱动文件, 方便我们编程。

4: 由于第 2 点和第 3 点的原因, keil5 对之前版本的 keil 兼容性并不是很好, 不能打开并且运行比如 keil4 的工程。为了能够兼容 KEIL4 以及之前的工程文件, KEIL 另外提供了一个安装补丁程序(mdkcmxxx.exe), 安装好这个程序之后, 就可以直接打开原来用 KEIL4 做的工程文件, 编译下载等操作都不会出现问题。也就是说, 你安装好 KEIL5 之后, 再安装一个兼容文件, 你之前的 KEIL4

工程就可以正常编译下载。

关于 KEIL MDK 的安装, 这里就不多讲了, 教程软件篇里有详细说明。现在主要说明下 pack 包的安装。

方法一: 点击 KEIL5 编辑器里面点击“packs installer”图标, 如图所示, 会弹出来一个安装器件 (pack installer) 的界面, 也就是说, 你要用它来开发哪个芯片。

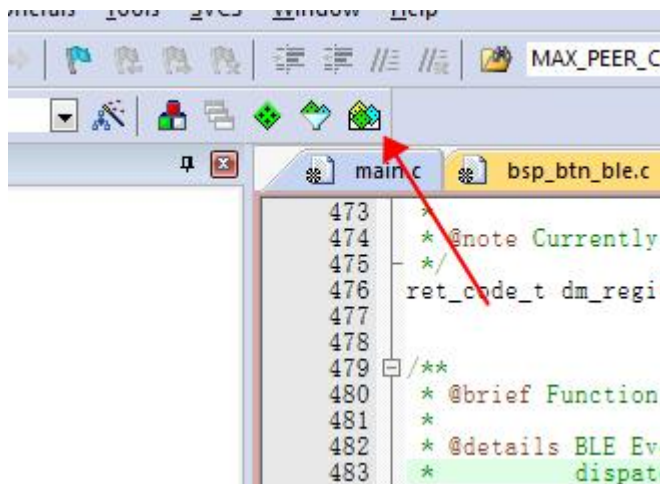


图 3.9 packs installer 图标

假如说我们要开发 nrf51822 芯片的话, 在右边的窗口中, 找到 nordic Semiconductor, 如下图所示: 点击 nRF52 Series 前面的+号, 就可以找到 nRF52 全系列的芯片。也就是我们要开发的芯片型号。

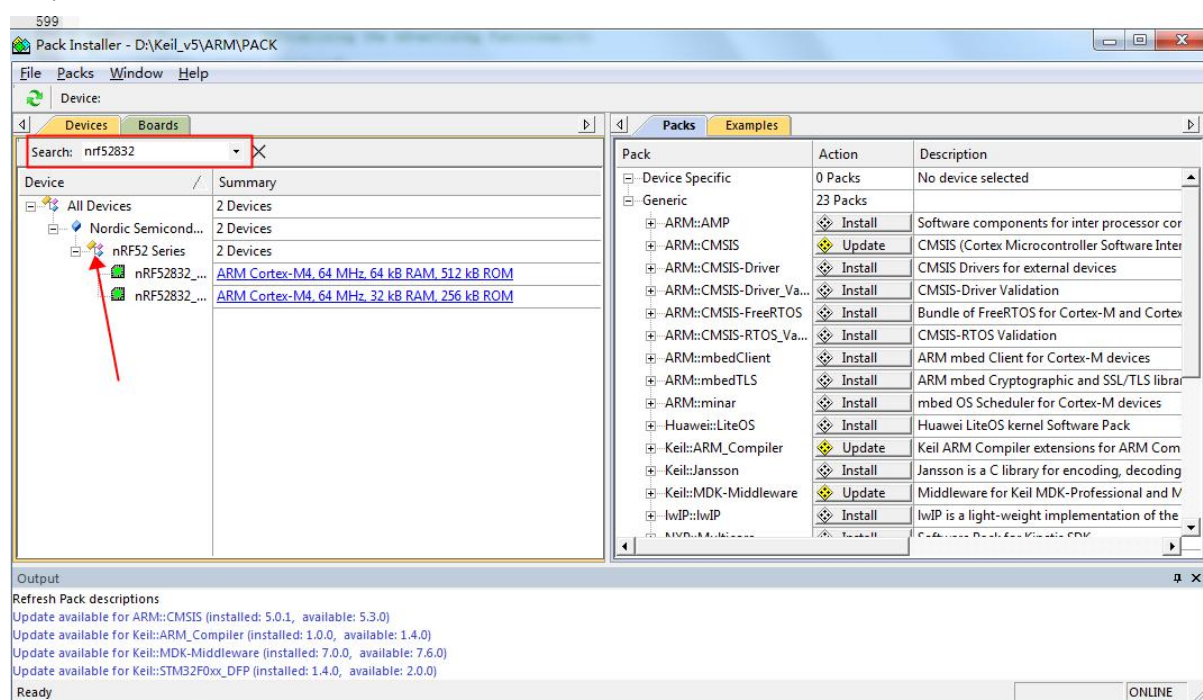


图 3.10 packs installer 界面

用鼠标点击选择芯片以后, 在窗口的左边, 会有提示, 如下图所示, 这个 Packs 就是 keil5 开发 nRF52 系列芯片需要的 packs:

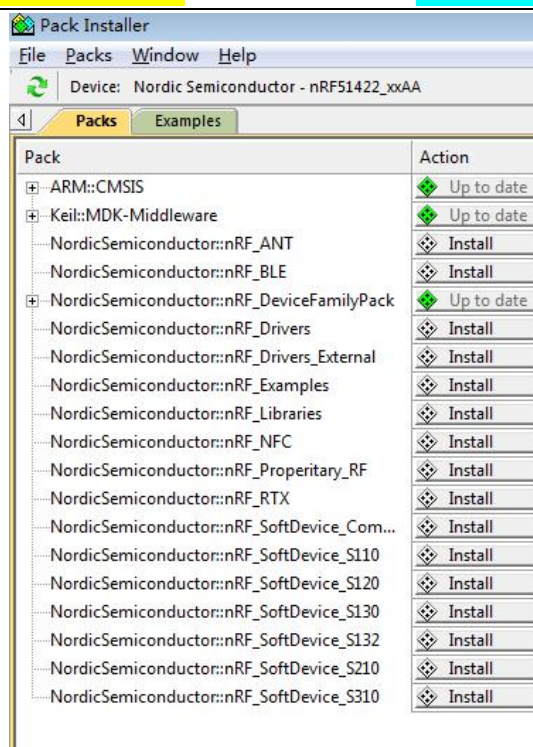


图 3.11 需要安装的 packs 列表

找到 nRF_DeviceFamilyPack 后面的 Install 按钮，点击安装，上图中，你看到 nRF_DeviceFamilyPack 后面的，不是 Install，而是 Up to data，那是表示已经安装好了，再点击就是升级了。点击 Install 以后，编译器会链接到一个网址，先下载后安装，在窗口的右下角有下载进度条，实际测试中，可是下载速度非常的慢，而且大部分时候，等一段时间会有下载失败的提示。如果出现这种情况，我们采用方法二进行安装。

方法二：直接到 MDK 官方网站上去下载 Pack 支持包，点击安装包的网站地址：<http://www.keil.com/dd2/Pack/>，找到 nordic Semiconductor 的支持包，如下图所示，nordic Semiconductor 提供很多开发板供读者使用，方便编程。

NordicSemiconductor

➤ ANT services and data modelling support modules.	BSP	2.0.1-2.alpha	↓
➤ Bluetooth Low Energy (Bluetooth Smart) services and software modules for	BSP	4.0.0-2.alpha	↓
➤ Common components for Nordic Semiconductor nRF family SoftDevices.	BSP	2.0.0-2.alpha	↓
➤ Components for ANT/ANT+ S210 SoftDevice for Nordic Semiconductor nRF	BSP	5.0.2	↓
➤ Components for Bluetooth Low Energy (Bluetooth Smart) and ANT/ANT+	BSP	3.0.1	↓
➤ Components for Bluetooth Low Energy (Bluetooth Smart) S110 SoftDevice	BSP	8.0.3	↓
➤ Components for Bluetooth Low Energy (Bluetooth Smart) S120 SoftDevice	BSP	2.1.1	↓
➤ Components for Bluetooth Low Energy (Bluetooth Smart) S130 SoftDevice	BSP	2.0.0-7.alpha	↓
➤ Components for Bluetooth Low Energy (Bluetooth Smart) S132 SoftDevice	BSP	2.0.0-7.alpha	↓
➤ Components for Bluetooth Low Energy (Bluetooth Smart) S1xx_iot	BSP	0.0.1-prototype2	↓
➤ Drivers for external hardware used by Nordic Semiconductor nRF family	BSP	1.2.1-2.alpha	↓
➤ Drivers for Nordic Semiconductor nRF family.	BSP	4.0.0-2.alpha	↓
➤ Examples and BSP for Nordic Semiconductor IoT SDK.	BSP	0.8.0	↓
➤ Examples and BSP for Nordic Semiconductor nRF family.	BSP	11.0.0-2.alpha	↓
➤ NFC services and data modelling support modules.	BSP	1.0.0-2.alpha	↓
➤ Nordic Semiconductor nRF ARM devices Device Family Pack.	DFP	8.12.0	↓

图 3.12 nordic Semiconductor 的支持包

找到 nRF ARM device Family Pack 选项, 点开后, 下面出现不同版本下的 PACKS 的下载地址, 需要下载哪个直接点击 Download:

▼ Nordic Semiconductor nRF ARM devices Device Family Pack. DFP 8.12.0

Version: 8.12.0 (2017-02-22) *NordicSemiconductor.nRF_DeviceFamilyPack.8.12.0.pack*
Fixed capabilities of device nRF52840_xxAA.
Corrected operating conditions and package information of different devices.
Added file startup_config.h and software component StartupConfig.

[Download](#)

Version: 8.11.1 (2016-11-18) *NordicSemiconductor.nRF_DeviceFamilyPack.8.11.1.pack*
Updated support for nRF52840_xxAA.
Other minor changes.

[Download](#)

Version: 8.11.0 (2016-11-09) *NordicSemiconductor.nRF_DeviceFamilyPack.8.11.0.pack*
Added support for nRF51824 device.
Added support for nRF52840 device.

[Download](#)

Version: 8.9.0 (2016-09-30) *NordicSemiconductor.nRF_DeviceFamilyPack.8.9.0.pack*
Added workaround for Errata 108.
Updated nrf52xxx.flm.
Updated nrf52xxx_sde.flm.
Updated nrf51 and nrf52 header files.

[Download](#)

Version: 8.7.1 (2016-07-15) *NordicSemiconductor.nRF_DeviceFamilyPack.8.7.1.pack*
Added GCC linker files to pack.

[Download](#)

图 3.13 nRF ARM 芯片支持包

下载 pack 后直接点击安装, 安装好后 Pack Installer 显示如下图所示, 我们选择安装的是 8.9.0 版本:

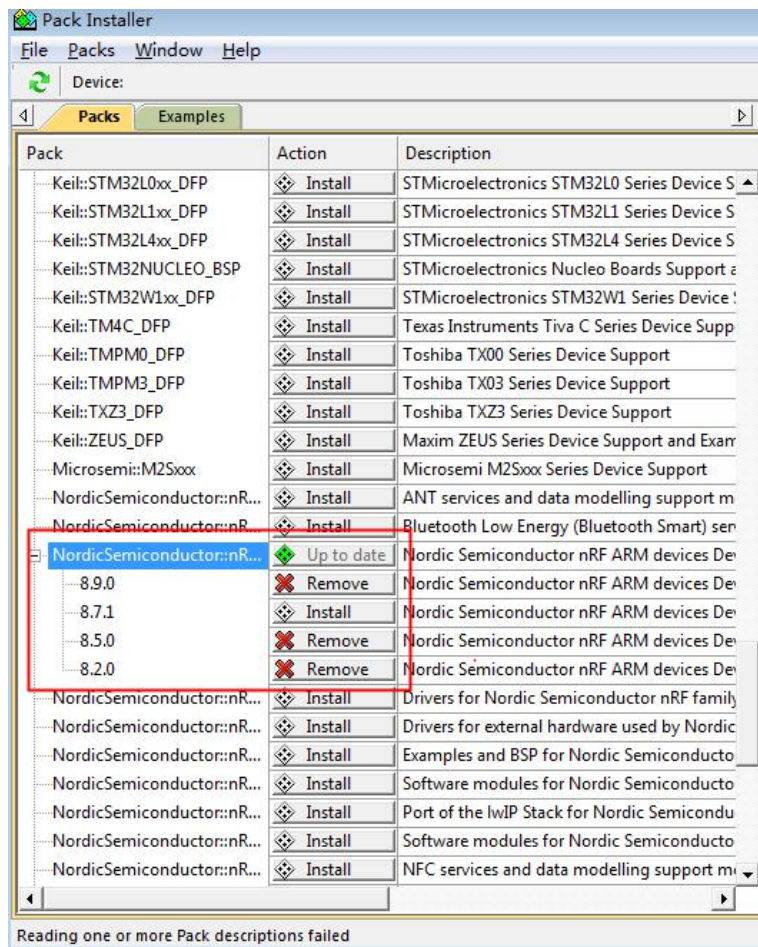


图 3.14 安装后显示 pack

2 工程项目的建立

前面两节详细介绍了如何配置 keil 开发环境以应用于 nrf52832 的开发, 并且说明了为了能够快速编程, 官方提供的 SDK 具体细节。这些都为本节建立自己的 nrf52832 工程打下了基础。本节将通过图片演示如何建立自己的一个外设工程项目。

1. 工程目录数规划: 在建立工程文件之前, 应该完成这个给目录树的规划。也就是说我们建立的工程放在什么位置? 如何调用官方给的驱动函数和库函数?

我们建立工程时候, 真正需要的官方的驱动文件库都包含在了上一节介绍的官方 components 文件夹内和 integration, modules 文件夹, 这部分作为我们程序的基本库:

名称	修改日期	类型	大小
components	2018-04-01 下午...	文件夹	
config	2018-04-01 下午...	文件夹	
documentation	2018-04-01 下午...	文件夹	
examples	2018-04-01 下午...	文件夹	
external	2018-04-01 下午...	文件夹	
external_tools	2018-04-01 下午...	文件夹	
integration	2018-04-01 下午...	文件夹	
modules	2018-04-01 下午...	文件夹	
license	2018-03-22 下午...	文本文档	1 KB
nRF5x_MDK_8_16_0_IAR_NordicLicense	2018-03-22 下午...	Windows Install...	1,640 KB
nRF5x_MDK_8_16_0_Keil4_NordicLice...	2018-03-22 下午...	Windows Install...	2,180 KB

图 3.20 建立工程所需文件包

建立新建文件夹命名为: 1: 工程的建立, 把 components 文件夹和 integration,modules 文件夹整体从官方 SDK 中拷贝过来, 同时新建两个文件夹。为了统一管理, 我们定义两个文件夹为 drive 和 examples。

drive 文件夹: 规划为我们自己所编写的一些设备驱动的存放文件夹, 比如演示的 led 灯的驱动, 这样后面程序移植。

examples 文件夹里包含两个文件夹: 一个为 bps 文件包定义为板级设备端口定义, 仿照官方的定义进行设置。一个为我们的工程包 ble_peripheral, 存放工程 and 主函数 main。整个目录结构如图所示:

名称	修改日期	类型
components	2018-06-11 上午...	文件夹
drive	2018-06-11 上午...	文件夹
examples	2018-06-14 下午...	文件夹
integration	2018-06-11 上午...	文件夹
modules	2018-06-11 上午...	文件夹

图中红色框标注了 components, drive, examples, integration, modules 文件夹。红色箭头从 components, integration, modules 指向“拷贝过来的”，从 drive, examples 指向“新建的文件夹”。

图 3.21 新的工程文件夹

2. 安装好 MDK keil 后, 打开 keil, 点开 keil 图标后在 porject 选项上打开 New uVision Project, 新建一个工程, 如下图所示:

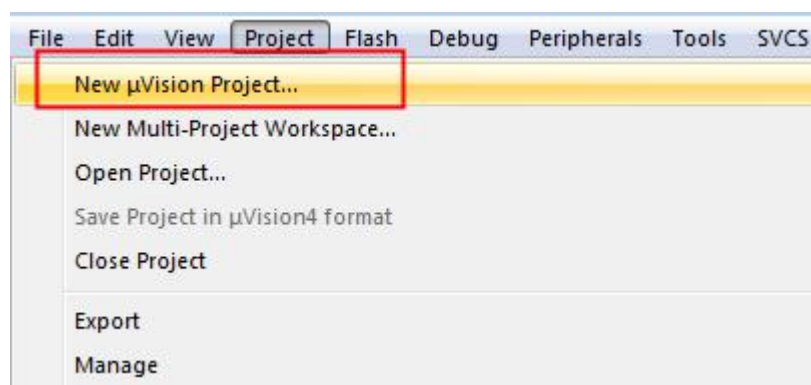


图 3.22 project 选项打开 New uVision Project

3. 点击后会弹出一个保存路径框，新建一个以main 命名的工程，保存到第一步规划的工程文件包 ble_peripheral 内，然后点击保存。如下图所示：

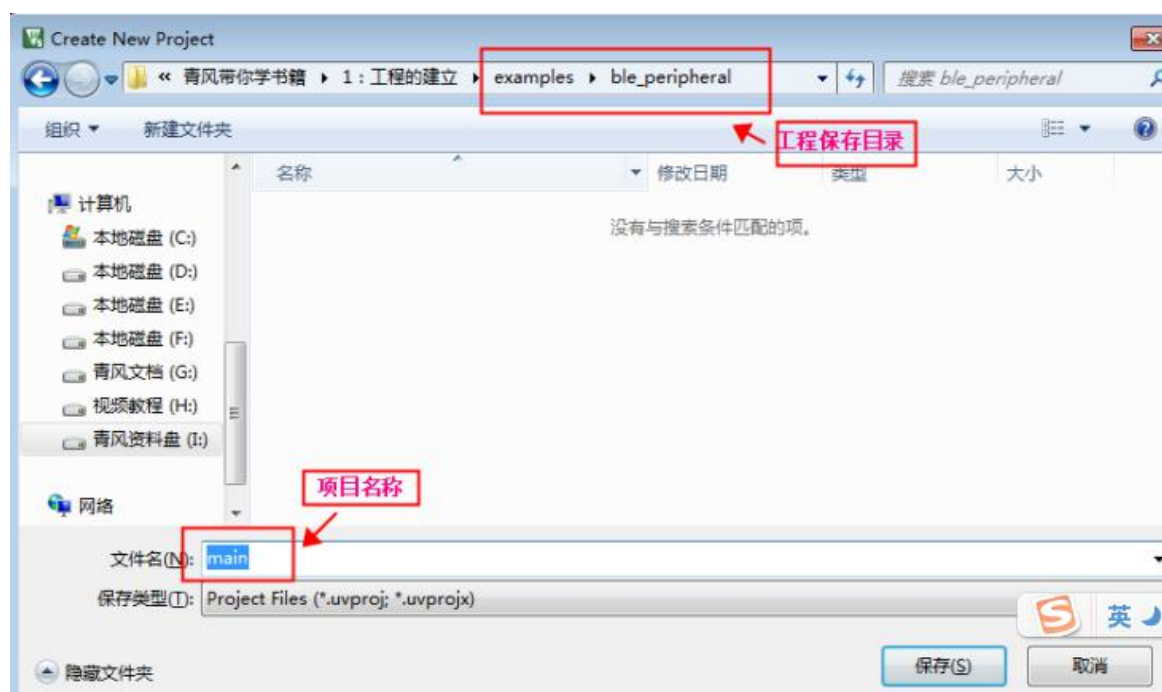


图 3.23 工程项目名称设置

4. 保存你选定的文件夹后，出现选择 CPU 的选择，找的我们所使用的芯片类型，这里需要注意一下，你需要找到Nordic nRF5x Series Devices 设备包含另个型号，我们选择 nrf52Series 点开展开后，选择芯片具体型号，选择好后点击 OK，如图所示：

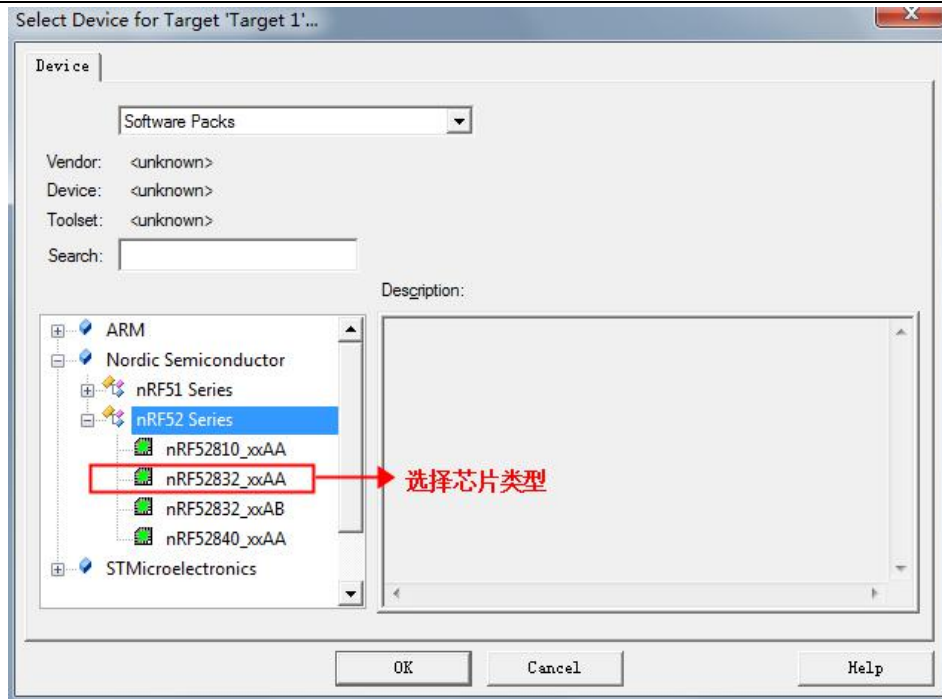


图 3.24 芯片选择

5. 此时会弹出一个设备 Manage Run-Time Environment, 这个管理器可以在线自动生成各类文件, 比如基本的启动文件、系统文件或者加入操作系统的文件等等。在建立一个基本工程项目的时候, 两个是必须的。一个是针对 arm 的内核的 CMSIS 的 CORE, 生成系统文件, 一个是芯片设备的启动文件 Device 中的 Startup, 生成启动文件。如图所示, 勾选两个选项, 如果正确, 图标中会显示绿色, 如果错误更具错误级别会显示橙色或者红色。勾选后, 点击 OK。

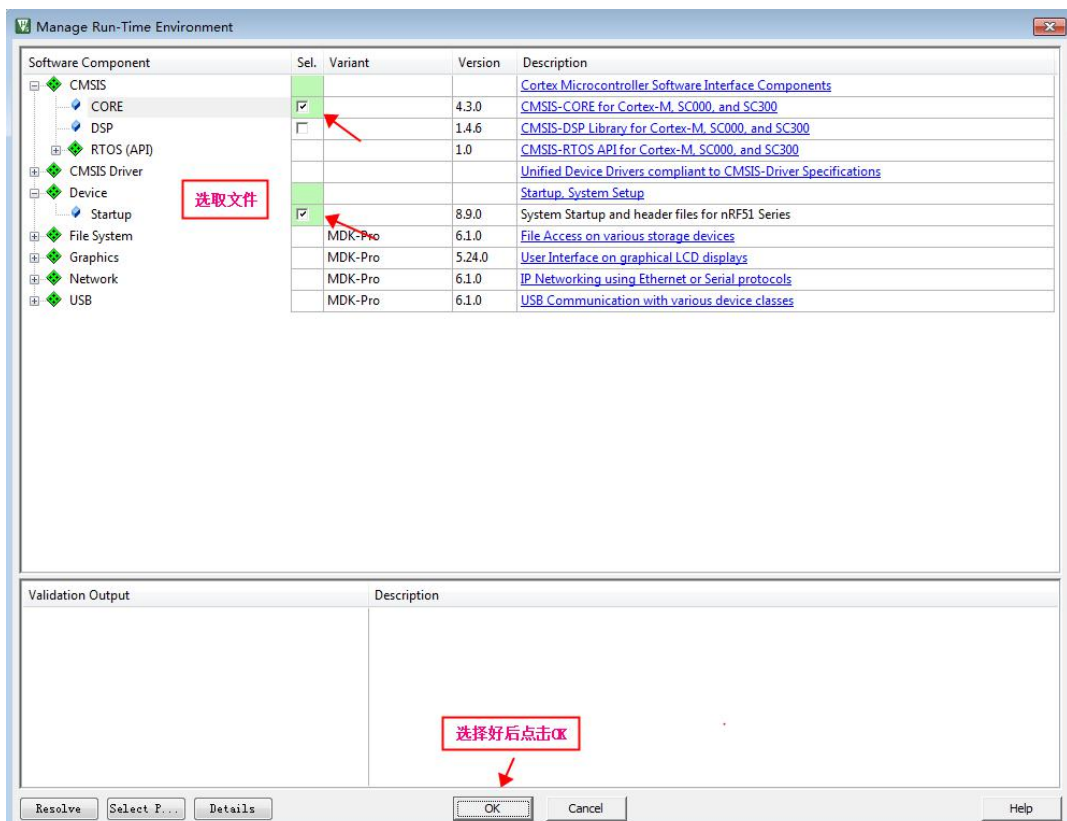


图 3.25 启动文件和系统文件生成

点击 OK 后, 就在工程目录中生成了系统文件和启动文件。启动文件的名称称为 `arm_startup_nrf52.s (startup)`, 这个文件是用汇编编写, 所以后缀为 S, 这是因为 C 语言生成的代码是不能上电后立即运行的, 因为此时还不具备运行条件, 比如全局变量还没有初始化, 系统堆栈还没有设置等。因此从系统上电, 到正式运行 `main` 主函数之前, 盱眙运行一段代码, 这段代码被称为启动代码。

启动代码主要包含了实现向量表的定义、堆栈初始化、系统变量初始化、中断系统初始化、地址重映射等操作。

`System_nrf52.c` 文件为芯片系统文件, 该文件主要定义系统时钟, 系统初始化等配置。一般情况下不需要用户修改。因此也可以通过系统自动生成。如下图所示:

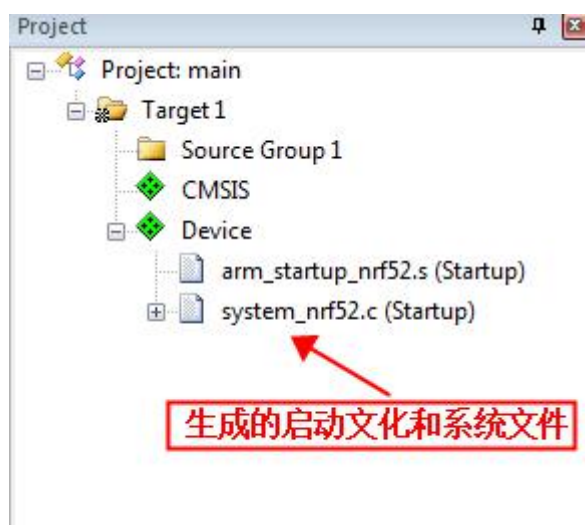


图 3.26 工程目录树添加文件

6. 工程项目建立成功后, 可以在工程项目 Target 里面添加工程组, 大家可以根据之前的规划名称直接来定义组的名称, 鼠标放 Target 上点击右键, 在下拉框里选择 Manage Project Items 工程管理选项后点击开:



图 3.27 工程管理 Manage Project Items

在工程管理添加你定义的分组，我们按照之前规划，定义两个分组：main 和 drive，如下图所示，添加好后点击 ok:



图 3.28 添加工程组

然后我们就需要在工作组中添加自己的 C 文件驱动了。先在 examples 的 ble_peripheral 中新建一个 main.c 文件，如下图所示:

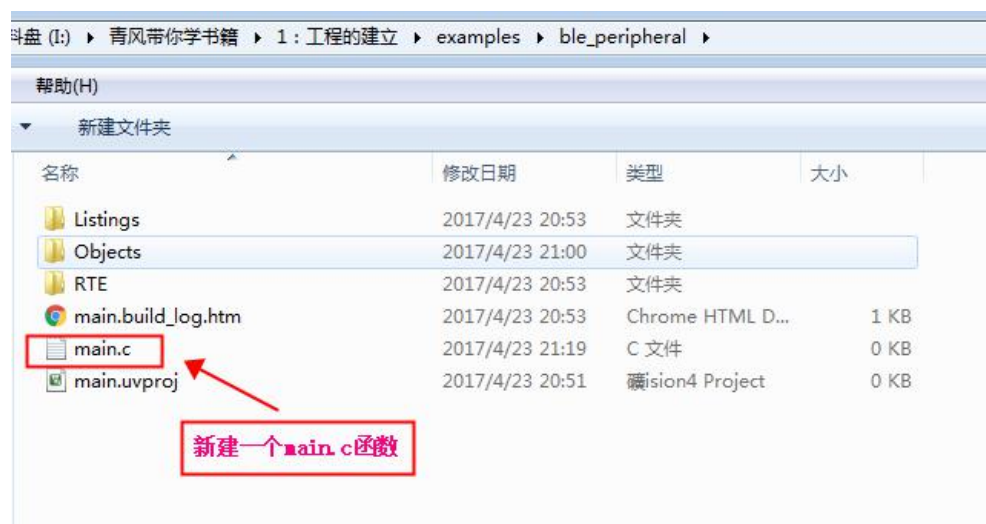


图 3.29 新建 main 主函数文件

然后可以向不同的组内添加驱动文件了，鼠标放到对应需要添加的文件夹上，点击右键在下拉选项框中选择 Add Existing Files to Group ‘main’:

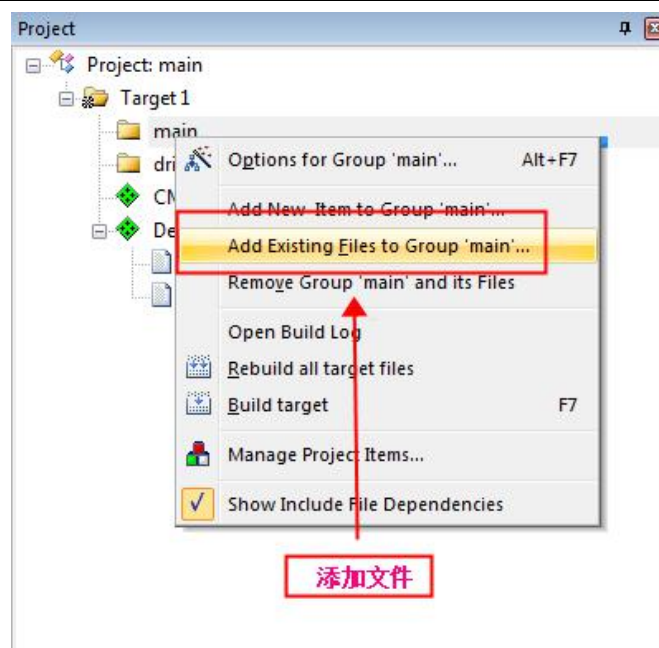


图 3.30 组中添加文件

同样的方式添加 led.c 驱动和延迟函数 nrf_delay.c 驱动, 添加后工程目录如下图所示:

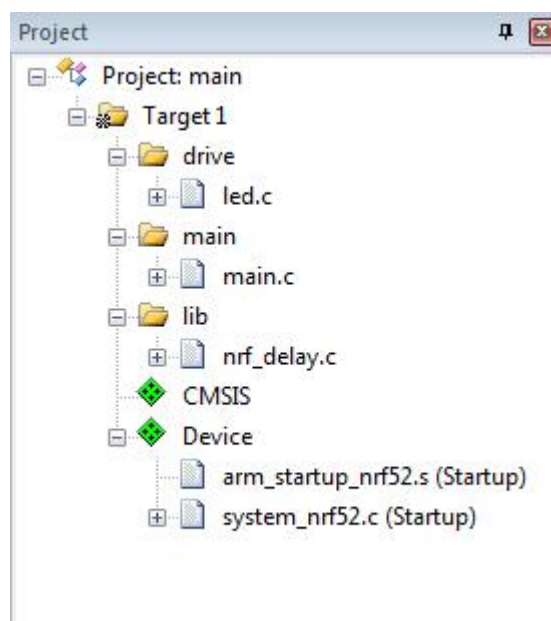


图 3.31 工程目录

7: 工程项目配置。在编译工程之前, 需要按照不同的需求进行工程配置。下面详细说明下工程项目中哪些位置需要设置的。鼠标放到 Target1 上点击右键, 在下拉选项框中点击 Options for Target 'target1' 打开工程设置选项框, 如下图所示:

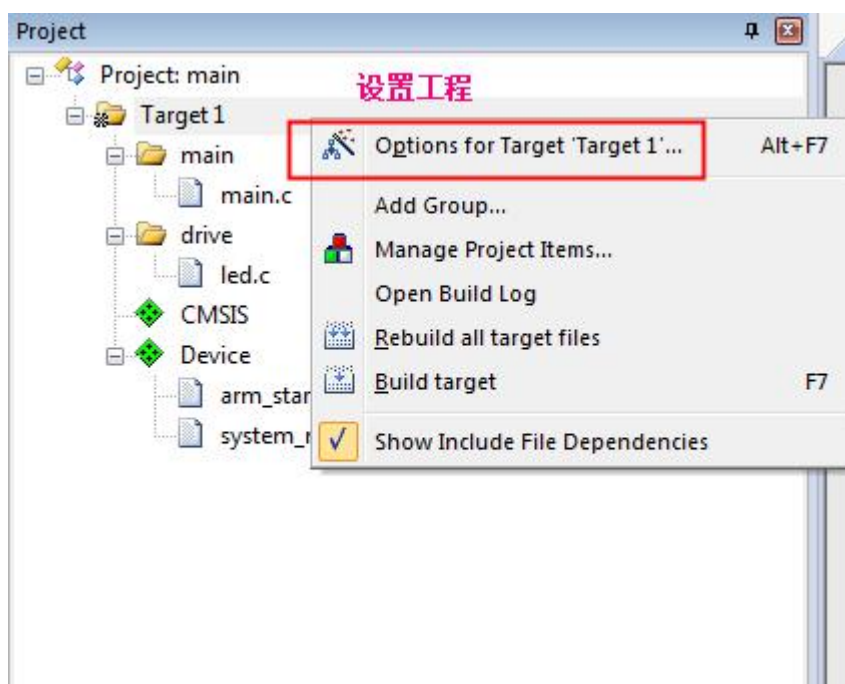


图 3.32 工程设置选项

1) 弹出一个工程设置框, 我们选择选项中的第二项 ‘Target’ 选项, 如下图所示, 该选项需要设置下面几个位置:

- Xtal (MHz) 填写硬件实际连接的高速晶振的频率, 我们开发板使用的是 16MHz。
- 如果需要使用串口 printf 等函数, 需要把 Use MicroLIB 勾选。

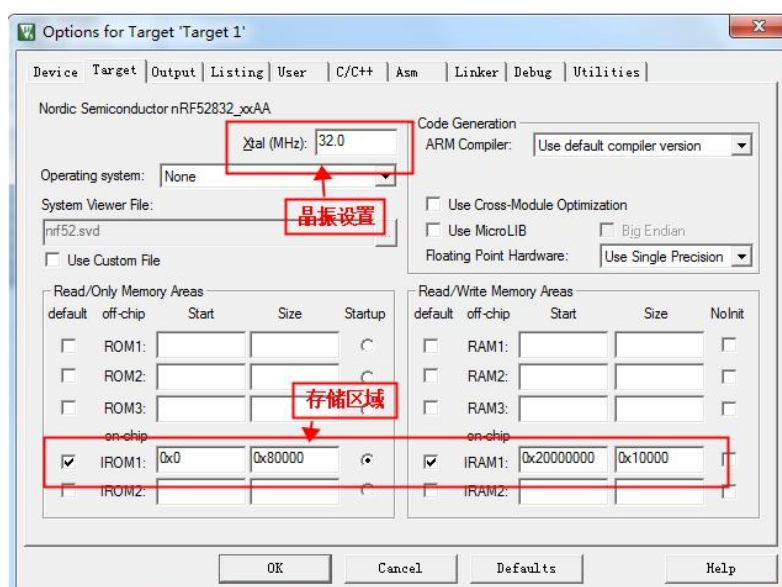


图 3.33 Target 选项卡

● 存储区的存储空间设置, 包括 FLASH ROM 和内存 RAM 的大小和起始地址。当然地址设置不能随便设置, 必须遵循芯片硬件以及协议栈需求设置。

2) 我们选择选项中的第二项 ‘Output’ 选项, 如下图所示, 该选项需要设置下面几个位置:

● 一般情况下 Debug Information 和 Browse Information 默认会被勾选, 以便编译的时候有提示信息。

● 如果需要生成 hex 文件, 首先勾选 create HEX Flie , 然后点击 Select Folder for Objects 选择生成 hex 的存放路径, 最后在 Name of Executable 中填写需要生成的 hex 的名称。

● 如果需要对代码保密, 生成封装好的 lib 文件的。则可以选择勾选最后一个选项: Creaata Library:. Objects\\main.lib

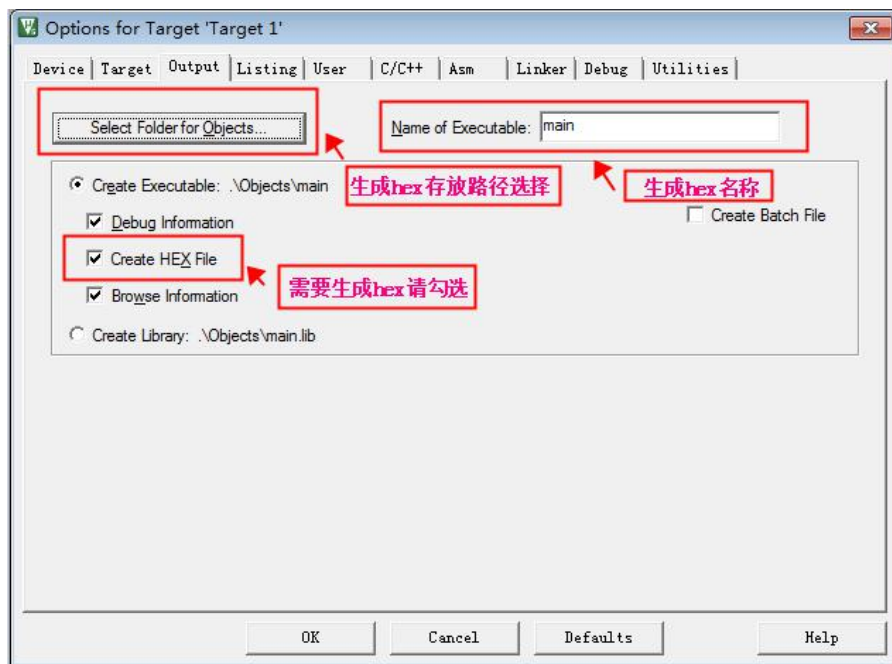


图 3.34 Output 选项卡

3) 最后需要设置的, 在C/c++内首先在Define 内添加库函数的使用定义, 然后在include 内点击添加编译源文件的路径, 设置如下图红色框图所示, 配置时需要注意以下几点:

● preprocessor symbols:C 语言编程中用于预处理标号, 预定义中。可以在 Define 中设置整个工程全局宏定义, 以提高编译效率。

● Optimization:keil 优化等级设置。Keil5 中提供四级优化:

Level 0:应用最小优化 (applies minimum optimizations)

Level 1:应用受限优化 (applies restricted optimization)

Level 2:应用高优化 (applies high optimization)

Level 3:应用最激进优化 (applies the most aggressive optimization)

详细解释请参考 MDK 官方说明。选择优化水平时请慎重, 优化水平过高可能影响代码执行逻辑。但是优化级别越高, 其编译的效率也是最高的, 可以有效的减小代码容量, 官方 BLE 蓝牙程序默认选项最高优化级别。

● One ELF Section per Function: 选项的主要功能是对冗余函数的优化。通过这个选项, 可以在最后生成的二进制文件中将冗余函数排除掉(虽然其所在的文件已经参与了编译链接), 以便最大程度地优化最后生成的二进制代码。而该选项实现的机

制是将每一个函数作为一个优化的单元，而并非整个文件作为参与优化的单元

选项 One ELF Section per Function 所具有的这种优化功能特别重要，尤其是在对于生成的二进制文件大小有严格要求的场合。特别是在带协议栈的 BLE 程序中将一系列接口函数放在一个文件里，然后将其整个包含在工程中，即使这个文件将只有一个函数被用到。这样，最后生成的二进制文件中就有可能包含众多的冗余函数，造成了宝贵存储空间浪费。

选项 One ELF Section per Function 对于一个大工程的优化效果尤其突出，有时候甚至可以达到减半的效果。当然，对于小工程或是少有冗余函数的工程来说，其优化效果就没有那么明显了。所以在建立带协议栈的蓝牙 BLE 程序时，请勾选此选项。

● C99 Mode: C 语言的最新的一种标准规范，比传统的 C98 增强了预处理功能，增加了对编译器的限制，初始化结构的时候允许对特定的元素赋值、浮点数的内部数据描述支持了新标准，可以使用 `#pragma` 编译器指令指定，除了已有的 `__line__`、`__file__` 以外，增加了 `__func__` 得到当前的函数名等等功能，如果需要用此种标准规范编程时请注意勾选。

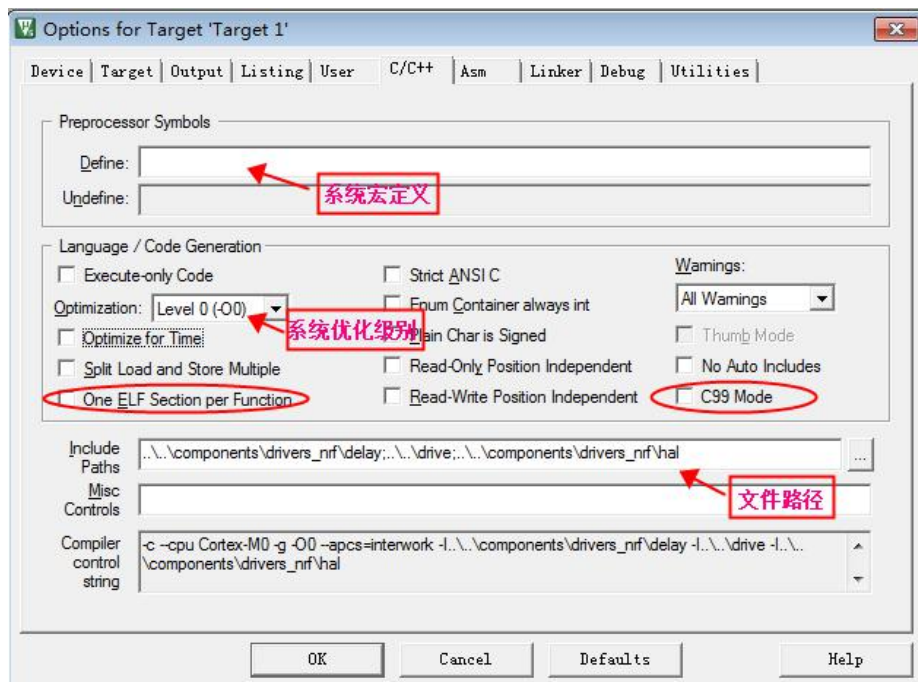


图 3.35 c/c++选项

● 选择添加文件路径操作如下图所示步骤实现，选择工程中调用文件驱动的文件夹路径添加：

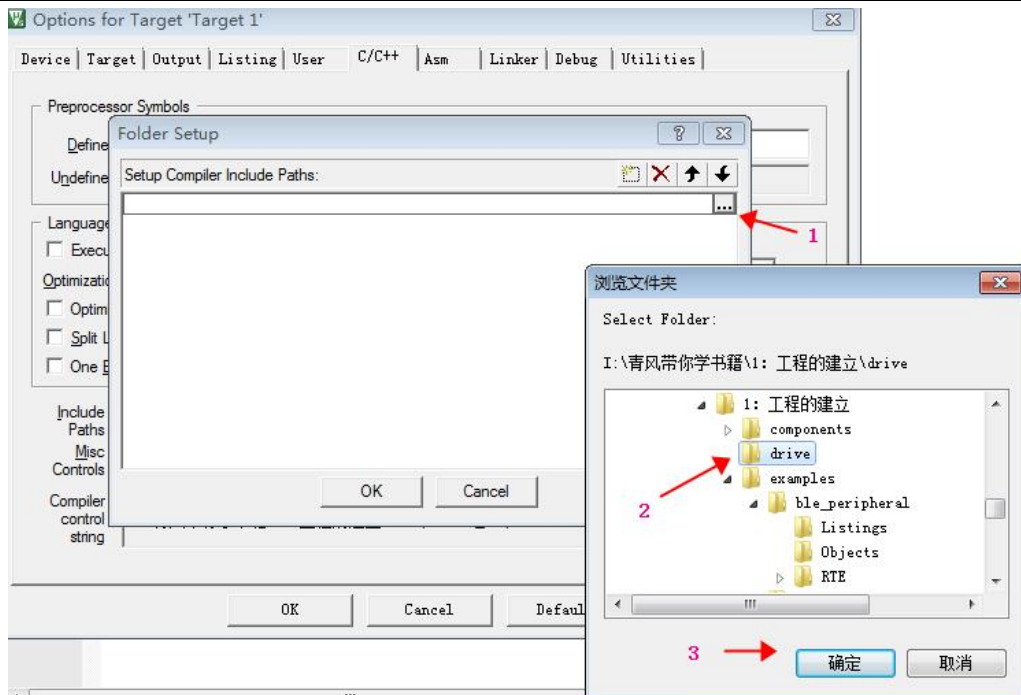


图 3.36 添加路径

完成了以上的步骤后,一个工程项目就建立成功了。此时就可以点击工具框上的整体编译按钮,如下图所示:



图 3.37 编译按钮

如果一切设置成功,则会出现如下图所示编译提示信息。

比如: Program Size: Code=1032 RO-data=224 RW-data=4 ZI-data=4196, Code 是代码占用的空间,RO-data 是 Read Only 只读常量的大小,如 const 型,RW-data 是 (Read Write) 初始化了的可读写变量的大小,ZI-data 是 (Zero Initialize) 没有初始化的可读写变量的大小。ZI-data 不会被算做代码里因为不会被初始化。

简单的说就是在烧写的时候是 FLASH 中的被占用的空间为: Code+RO-data+RW-data。程序运行的时候,芯片内部 RAM 使用的空间为: RW-data+ZI-data。初始化时 RW-data 从 flash 拷贝到 RAM。通过编译提示能够估算硬件设备的空间大小是否满足我们代码容量的要求。

最后一句: 提示为 0 错误, 0 警告

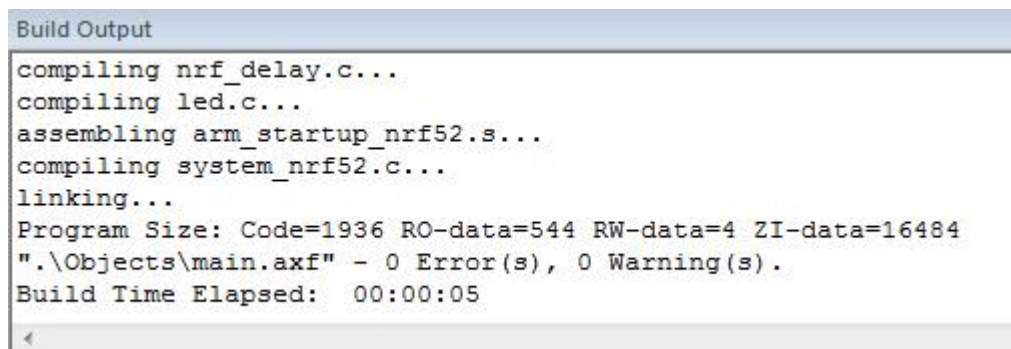


图 3.38 编译信息

3 工程项目的仿真与调试

3.1 仿真工具的选择与设置:

由于目前在 WIN 系统中 NRFgo Studio 下载协议栈仅仅支持 JLINK 仿真器, 因此我们调试中也选取这款仿真器。

在 MDK 的 Options for Target‘target1’ 中打开工程设置选项框, 点击 Debug 选择框。设置仿真器的选项。点击选择硬件仿真: jlink/J-Trace Cortex, 同时勾选 Run to main()。如下图所示, 然后点击 settings 设置:



图 3.39 Debug 选项卡

如果仿真器和设备连接好后, 点开 settings 后会弹出一个选项框, 选择在 Debug 框左边显示仿真器的类型, 驱动版本等信息。由于 nRF5x 系列处理器只提供 SWD 仿真接口, 因此 Port 选择 SW 模式, 如下图所示。如果设备和仿真器连接成功提示发现硬件内核的 IDCODE 和 Device Name。

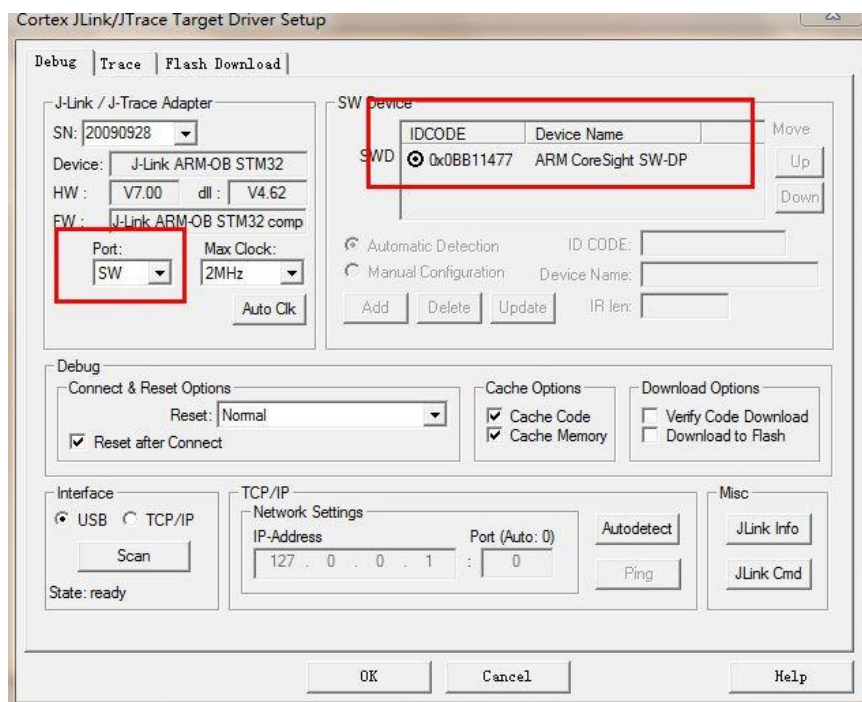


图 3.40 Debug 选项卡设置

切换到 FLASH Download 选项，选择相应的器件的 FLASH 大小，我们选择 2M FLASH：

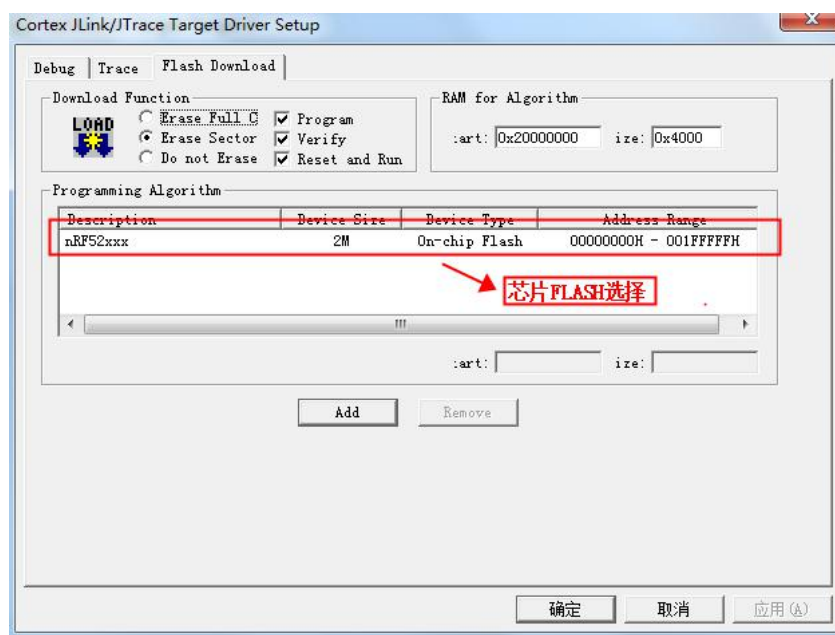


图 3.41 FLASH Download 选项卡

以上设置好后就可以下载并且仿真了。



图 3.42 debug 选项卡

3.2 项目仿真错误定位

为了检测我们编写的程序是否正常,在 MDK keil 上通过 jlink 可以实现硬件的在线仿真。但是很多编程者编写的程序在仿真时出现跑飞或者出错状态,却不知道如何找出错误的原因。为了解决这个问题, nRF5x 系列处理器在软件包中提供了错误检查机制。这个机制的核心就是 APP_ERROR_CHECK(err_code)函数,也就是错误码检测方式。本节将详细讲述如何使用这种方式找出自己程序中出现错误原因。

其实 APP_ERROR_CHECK(err_code)这句话就是 Nordic 为了方便使用者快速定位到错误的地方。APP_ERROR_CHECK(err_code)函数适用于目前所有 SDK 开发包。在这里以蓝牙工程样例为例讲解仿真调试过程。

找到该函数内部定义,当设置了 DEBUG 时:执行 app_error_handler((ERR_CODE), __LINE__, (uint8_t*) __FILE__);否则执行 app_error_handler((ERR_CODE), 0, 0);

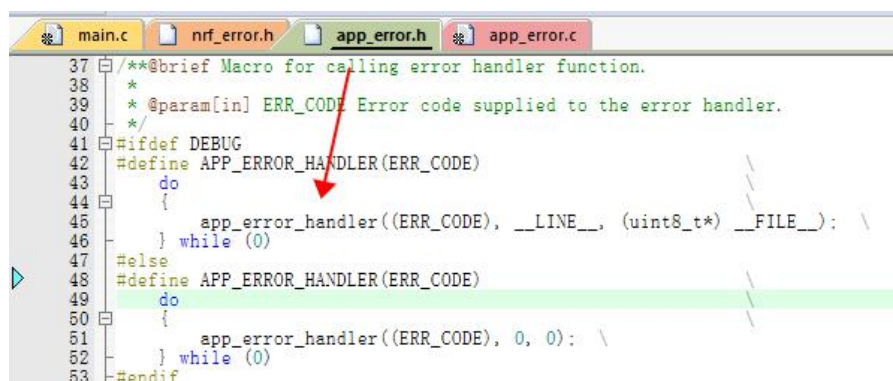
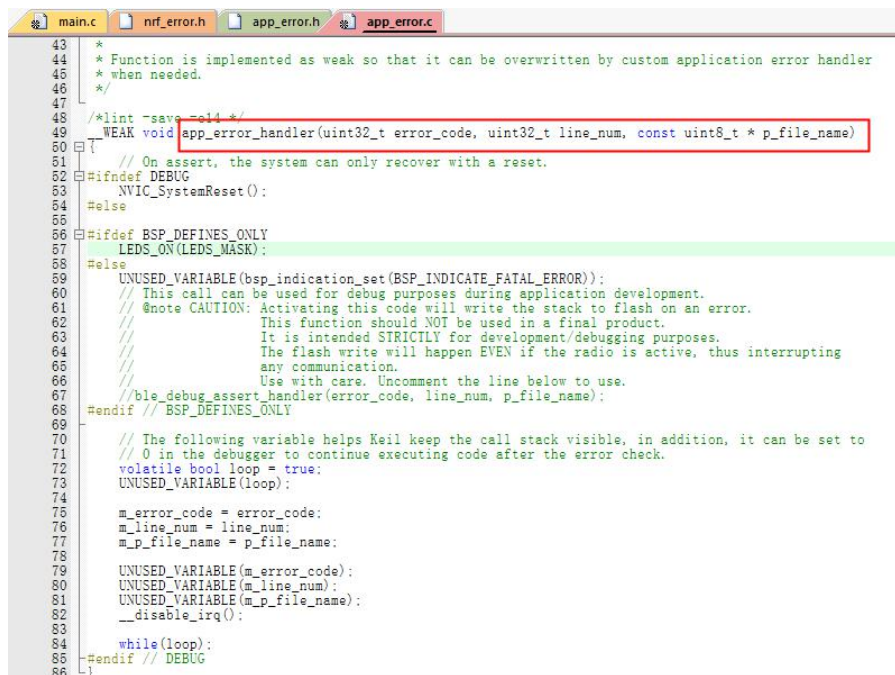


图 3.43 app_error 错误执行函数

程序最终会运行进入 app_error.c 中的 app_error_handler 函数。关注两个形参: uint32_t line_num 和 const uint8_t * p_file_name , 两个形参一个是定位错误的行数, 一个是定位错误的文件的名称。如果我们没有宏定义 DEBUG, 那这两个参数全部设置为 0, 也就是不能定位。否则设置为 __LINE__ 和 (uint8_t*) __FILE__, 那么就可以进行错误定位了。app_error_handler 函数定义如下图所示:



```
43 *
44 * Function is implemented as weak so that it can be overwritten by custom application error handler
45 * when needed.
46 */
47
48 /*lint -save -e14 */
49 WEAK void app_error_handler(uint32_t error_code, uint32_t line_num, const uint8_t * p_file_name)
50 {
51     // On assert, the system can only recover with a reset.
52     #ifndef DEBUG
53         NVIC_SystemReset();
54     #else
55     #endif
56     #ifndef BSP_DEFINES_ONLY
57         Leds_On(Leds_Mask);
58     #else
59         UNUSED_VARIABLE(bsp_indication_set(BSP_INDICATE_FATAL_ERROR));
60         // This call can be used for debug purposes during application development.
61         // @note CAUTION: Activating this code will write the stack to flash on an error.
62         // This function should NOT be used in a final product.
63         // It is intended STRICTLY for development/debugging purposes.
64         // The flash write will happen EVEN if the radio is active, thus interrupting
65         // any communication.
66         // Use with care. Uncomment the line below to use.
67         // ble_debug_assert_handler(error_code, line_num, p_file_name);
68     #endif // BSP_DEFINES_ONLY
69
70     // The following variable helps Keil keep the call stack visible, in addition, it can be set to
71     // 0 in the debugger to continue executing code after the error check.
72     volatile bool loop = true;
73     UNUSED_VARIABLE(loop);
74
75     m_error_code = error_code;
76     m_line_num = line_num;
77     m_p_file_name = p_file_name;
78
79     UNUSED_VARIABLE(m_error_code);
80     UNUSED_VARIABLE(m_line_num);
81     UNUSED_VARIABLE(m_p_file_name);
82     __disable_irq();
83
84     while(loop);
85 #endif // DEBUG
86 }
```

图 3.44 app_error_handler 函数定义

为了在仿真调试时，能够快速定位错误出现的位置，我们就需要定义一下宏 DEBUG，打开工程的 Options for Target 选项框，在 Define 中输入宏定义 DEBUG，注意和之前的定义通过空格或者逗号隔开。输入如下图所示：

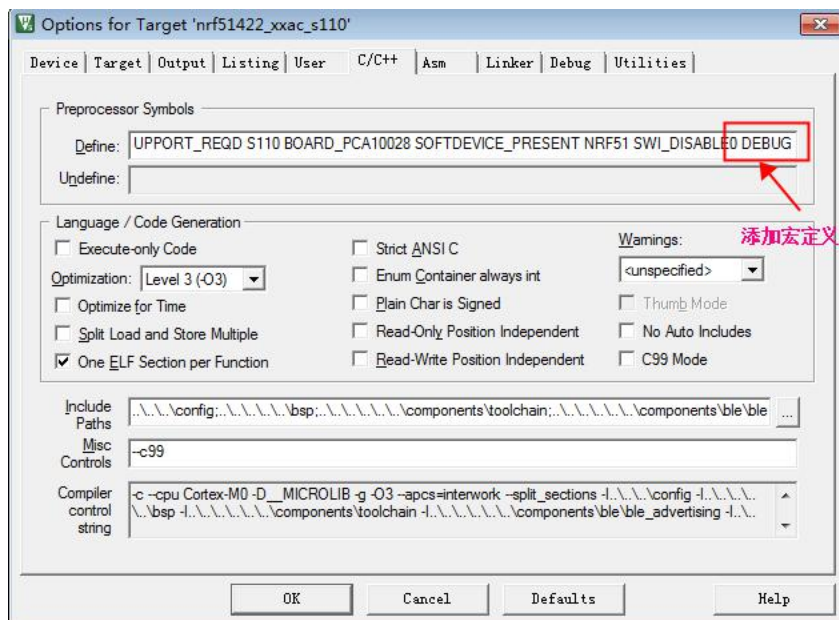


图 3.45 添加 DEBUG

为了验证我们的仿真器调试方法，我们设置一个错误名称定义，蓝牙数据包最多为 31 个字节，程序中如果广播超过了 31 个字节，比如蓝牙名称过长太长或者 UUID 过多，则会造成调用了 APP_ERROR_CHECK(err_code)就会出现复位的情况，进入错误提示中。这里我们改成一个较长的蓝牙名称，如图所示，修改完这两个地方了以后编译烧写程序进入芯片。

```
48
49 #define DEVICE_NAME "Nordic_Templatexxxxxxxxxxxxxxxxxxxxxxxxxxxx"
50 #define MANUFACTURER_NAME "NordicSemiconductor"
51 #define APP_ADV_INTERVAL 300
52 #define APP_ADV_TIMEOUT_IN_SECONDS 180
```

点击仿真 debug 选项，如下图所示，进入仿真状态：



图 3.46 debug 图标

仿真中设置多个断点。如何断点前的程序无错误，那么直接点击运行至断点按钮，程序会运行到断点停下来，在断点处出现黄色三角号，表明之前的运行正常。如下图所示：

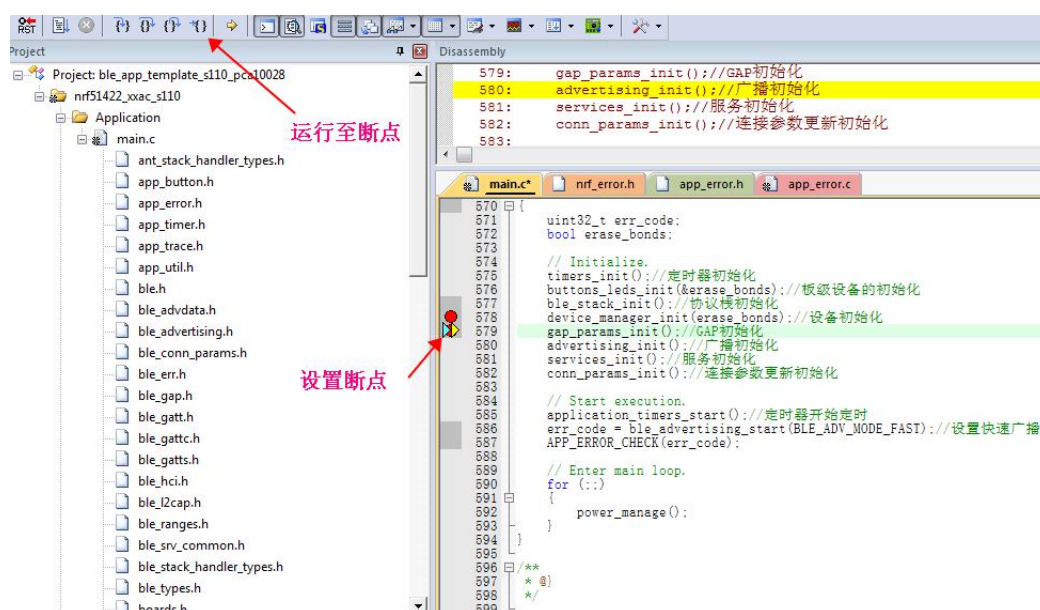


图 3.47 断点设置

如果点击运行至断点按钮，程序一直无法再断点处停下来，不出现黄色三角标号。此时表示程序已经出错跑飞。我们点击红色 x 按钮，停止运行。程序会在错误处停下来。

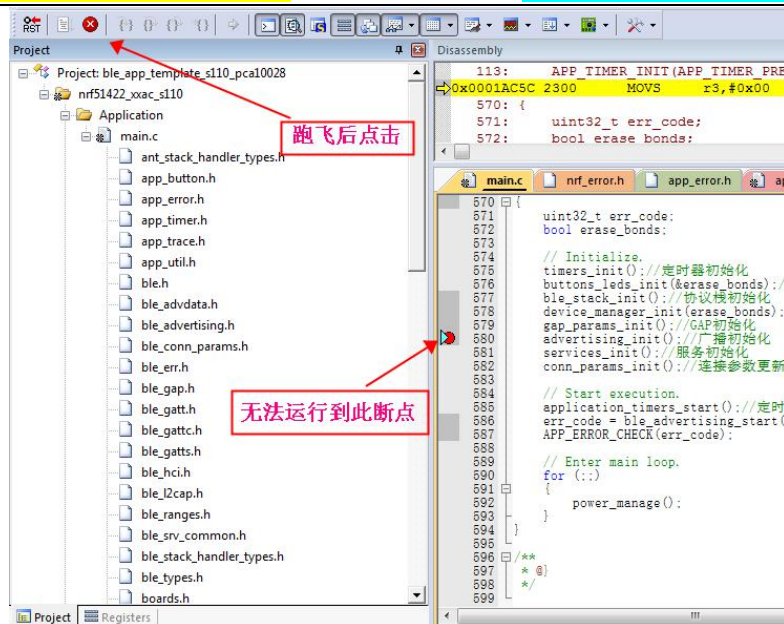


图 3.48 模拟跑飞

程序果然已经进入到了 `app_error_handler` 函数的死循环中, 此时可以通过形参进行定位错误点了:

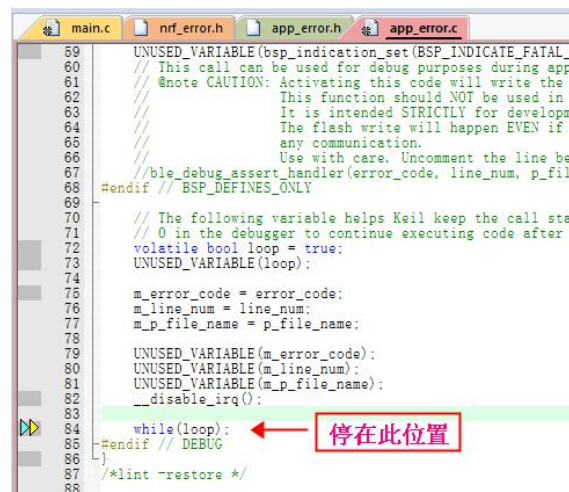


图 3.49 错误后运行的位置

把鼠标放到形参上, 点击鼠标右键, 在下拉框中点击添加到窗口, 使用 keil 的 watch1 窗口观察一下 `m_error_code`, `m_line_num` 和 `m_p_file_name` 这三个形参变量

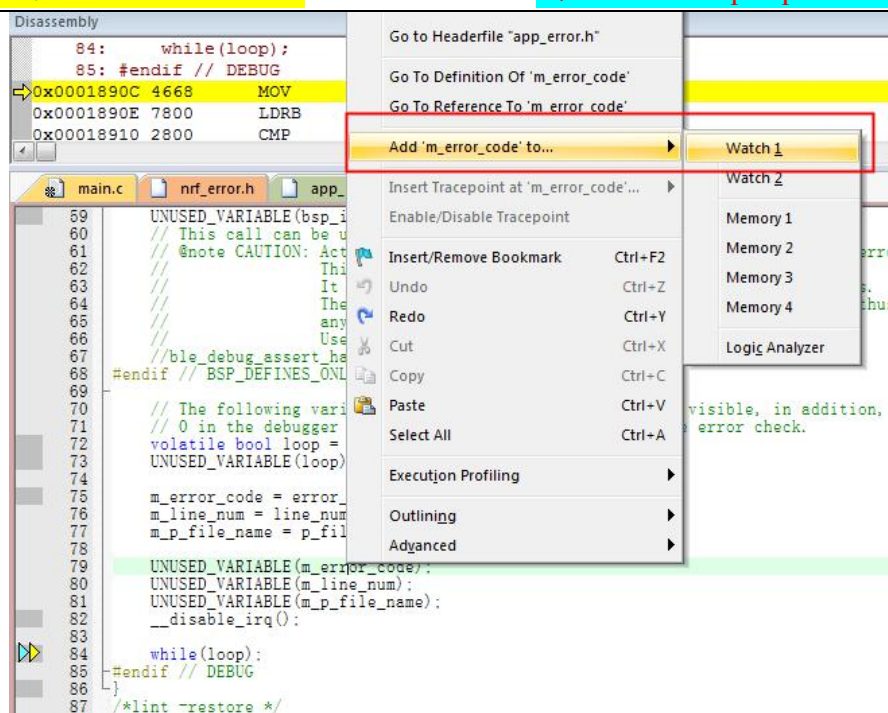


图 3.50 开窗口观察错误码

程序中 `m_error_code` 表示定位错误的指示码, `m_line_num` 表示定位错误的行数, `m_p_file_name` 表示定位错误的指示文件名称。

`m_error_code` 为 0x0c, 十进制为: 12

`m_line_num` 为 0x8F 十进制为: 143

`m_p_file_name` 为 main 名称

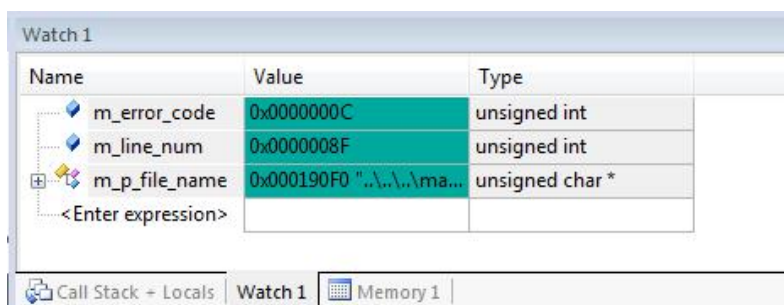
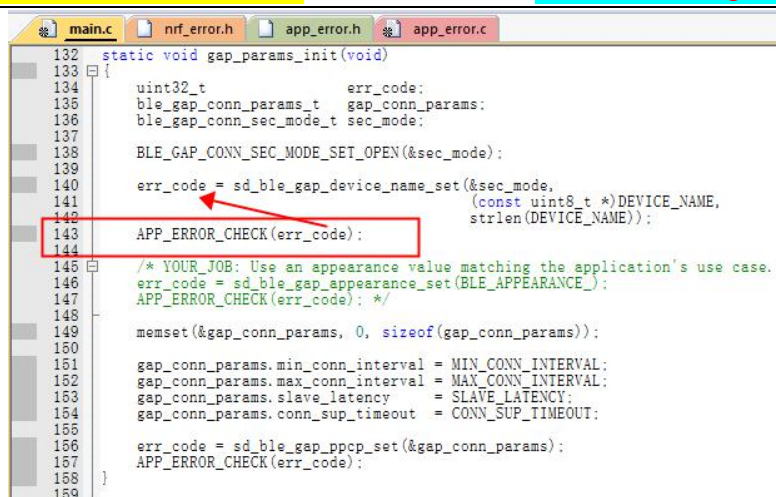


图 3.51 错误码

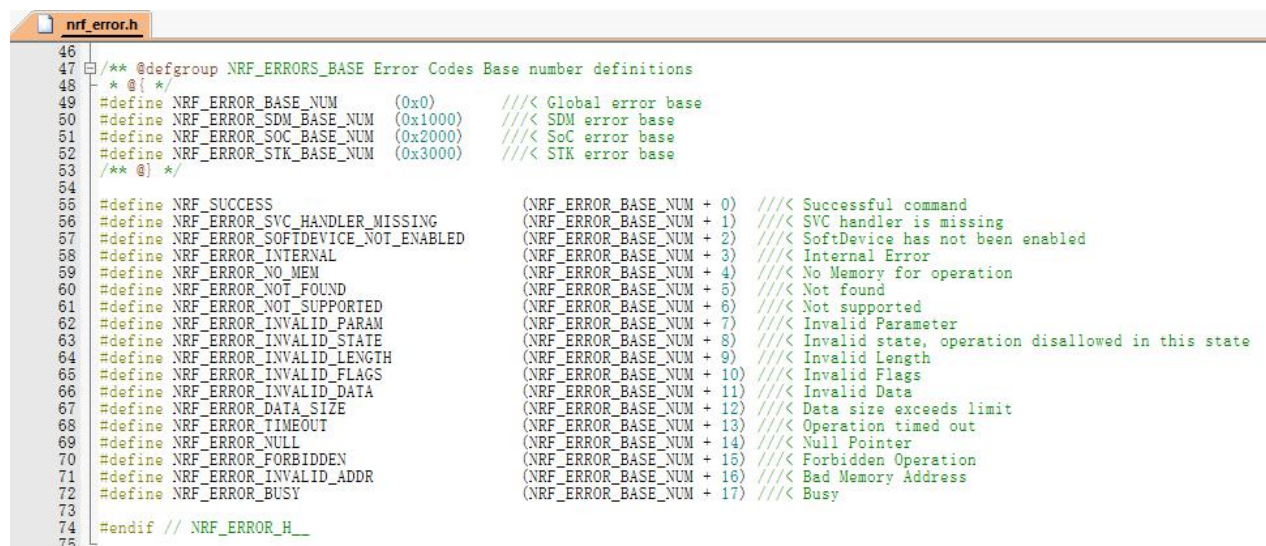
那么错误就出现在文件名称 main 的第 143 行, 找到这个行, 这个行的错误码赋值为设备名称的设置, 如下图所示:



```
132 static void gap_params_init(void)
133 {
134     uint32_t err_code;
135     ble_gap_conn_params_t gap_conn_params;
136     ble_gap_conn_sec_mode_t sec_mode;
137
138     BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);
139
140     err_code = sd_ble_gap_device_name_set(&sec_mode,
141                                           (const uint8_t *)DEVICE_NAME,
142                                           strlen(DEVICE_NAME));
143     APP_ERROR_CHECK(err_code);
144
145     /* YOUR_JOB: Use an appearance value matching the application's use case.
146     err_code = sd_ble_gap_appearance_set(BLE_APPEARANCE);
147     APP_ERROR_CHECK(err_code); */
148
149     memset(&gap_conn_params, 0, sizeof(gap_conn_params));
150
151     gap_conn_params.min_conn_interval = MIN_CONN_INTERVAL;
152     gap_conn_params.max_conn_interval = MAX_CONN_INTERVAL;
153     gap_conn_params.slave_latency = SLAVE_LATENCY;
154     gap_conn_params.conn_sup_timeout = CONN_SUP_TIMEOUT;
155
156     err_code = sd_ble_gap_ppcp_set(&gap_conn_params);
157     APP_ERROR_CHECK(err_code);
158 }
159
```

图 3.52 出错程序位置

错误代码可以在 `nrf_error.h` 中查询，这里可以看出 `0x0c` 即为 12 号错误 `NRF_ERROR_DATA_SIZE`，意思为数据长度超出限制了，这里就可以分析出是广播中的内容太长了，导致发生了错误。



```
46
47 /** @defgroup NRF_ERRORS_BASE Error Codes Base number definitions
48 * @{ */
49 #define NRF_ERROR_BASE_NUM (0x0) ///< Global error base
50 #define NRF_ERROR_SDM_BASE_NUM (0x1000) ///< SDM error base
51 #define NRF_ERROR_SOC_BASE_NUM (0x2000) ///< SoC error base
52 #define NRF_ERROR_STK_BASE_NUM (0x3000) ///< STK error base
53 /** @} */
54
55 #define NRF_SUCCESS (NRF_ERROR_BASE_NUM + 0) ///< Successful command
56 #define NRF_ERROR_SVC_HANDLER_MISSING (NRF_ERROR_BASE_NUM + 1) ///< SVC handler is missing
57 #define NRF_ERROR_SOFTDEVICE_NOT_ENABLED (NRF_ERROR_BASE_NUM + 2) ///< SoftDevice has not been enabled
58 #define NRF_ERROR_INTERNAL (NRF_ERROR_BASE_NUM + 3) ///< Internal Error
59 #define NRF_ERROR_NO_MEM (NRF_ERROR_BASE_NUM + 4) ///< No Memory for operation
60 #define NRF_ERROR_NOT_FOUND (NRF_ERROR_BASE_NUM + 5) ///< Not found
61 #define NRF_ERROR_NOT_SUPPORTED (NRF_ERROR_BASE_NUM + 6) ///< Not supported
62 #define NRF_ERROR_INVALID_PARAM (NRF_ERROR_BASE_NUM + 7) ///< Invalid Parameter
63 #define NRF_ERROR_INVALID_STATE (NRF_ERROR_BASE_NUM + 8) ///< Invalid state, operation disallowed in this state
64 #define NRF_ERROR_INVALID_LENGTH (NRF_ERROR_BASE_NUM + 9) ///< Invalid Length
65 #define NRF_ERROR_INVALID_FLAGS (NRF_ERROR_BASE_NUM + 10) ///< Invalid Flags
66 #define NRF_ERROR_INVALID_DATA (NRF_ERROR_BASE_NUM + 11) ///< Invalid Data
67 #define NRF_ERROR_DATA_SIZE (NRF_ERROR_BASE_NUM + 12) ///< Data size exceeds limit
68 #define NRF_ERROR_TIMEOUT (NRF_ERROR_BASE_NUM + 13) ///< Operation timed out
69 #define NRF_ERROR_NULL (NRF_ERROR_BASE_NUM + 14) ///< Null Pointer
70 #define NRF_ERROR_FORBIDDEN (NRF_ERROR_BASE_NUM + 15) ///< Forbidden Operation
71 #define NRF_ERROR_INVALID_ADDR (NRF_ERROR_BASE_NUM + 16) ///< Bad Memory Address
72 #define NRF_ERROR_BUSY (NRF_ERROR_BASE_NUM + 17) ///< Busy
73
74 #endif // NRF_ERROR_H_
75
```

图 3.52 nrf_error 程序指示错误类型

通过上面的仿真分析，我们可以快速查找定位错误出现的位置和出现错误的原因。在协议栈没有开源的情况下，为我们的编程提供了极大的方便。