GR5525开发者指南

版本： **1.0**

发布日期： **2023-08-30**

深圳市汇顶科技股份有限公司

商标声明

GOODiX 和其他汇顶商标均为深圳市汇顶科技股份有限公司的商标。本文档提及的其他所有商标或注册商标，由各自的所有人持有。

免责声明

本文档中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。

深圳市汇顶科技股份有限公司（以下简称“GOODIX”）对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。GOODIX对因这些信息及使用这些信息而引起的后果不承担任何责任。

未经GOODIX书面批准，不得将GOODIX的产品用作生命维持系统中的关键组件。在GOODIX知识产权保护下，不得暗中或以其他方式转让任何许可证。

深圳市汇顶科技股份有限公司

总部地址：深圳市福田保税区腾飞工业大厦B座12-13层

电话：+86-755-33338828　　邮编：518000

网址：www.goodix.com

前言

编写目的

本文档主要介绍Goodix GR5525低功耗蓝系统级芯片（SoC）的软件开发工具套件（SDK），以及使用Keil开发和调试程序的方法，以帮助开发者开发低功耗蓝牙（Bluetooth LE）应用。

读者对象

本文适用于以下读者：

- GR5525用户
- GR5525开发人员
- GR5525测试人员
- 文档工程师

版本说明

本手册为第1次发布，对应的产品系列为GR5525。

修订记录

版本	日期	修订内容
1.0	2023-08-30	首次发布

目录

1 简介

GR5525系列芯片是Goodix推出的一款支持Bluetooth 5.3的低功耗蓝牙（Bluetooth LE）系统级芯片（SoC），可以配置为广播者（Broadcaster）、观察者（Observer）、外围设备（Peripheral）和中央设备（Central），并支持这些角色的组合应用，可广泛应用于物联网（IoT）和智能穿戴设备领域。

GR5525系列芯片架构以ARM® Cortex®-M4F CPU为核心，集成Bluetooth 5.3协议栈、2.4 GHz RF收发器、片上可编程存储器Flash、RAM以及多种外设。

GR5525 系列芯片提供QFN56和QFN68两种封装，具体的芯片配置如下表所示。

表 1-1 GR5525系列芯片配置

产品型号	**GR5525RGNI**	**GR5525IGNI**	**GR5525IENI**	**GR5525I0NI**
内核	Cortex®-M4F	Cortex®-M4F	Cortex®-M4F	Cortex®-M4F
RAM	256 KB	256 KB	256 KB	256 KB
SiP Flash	1 MB	1 MB	512 KB	N/A
I/O数量	50	39	39	39
I/O电压	1.8 V ～ 3.6 V	1.8 V ～ 3.6 V	1.8 V ～ 3.6 V	跟随外部Flash
封装（mm）	QFN68 (7.0 x 7.0 x 0.85)	QFN56 (7.0 x 7.0 x 0.75)	QFN56 (7.0 x 7.0 x 0.75)	QFN56 (7.0 x 7.0 x 0.75)

1.1 GR5525 SDK

GR5525软件开发工具套件（Software Development Kit，SDK）为GR5525系列SoC提供全面的软件开发支持。该SDK包含Bluetooth LE API、System API、外设驱动程序、调试/下载工具、工程示例代码以及相关的用户文档等。

本文档描述的GR5525 SDK版本，适用于GR5525系列所有芯片。

1.2 低功耗蓝牙协议栈

低功耗蓝牙协议栈的架构如图 1-1所示。

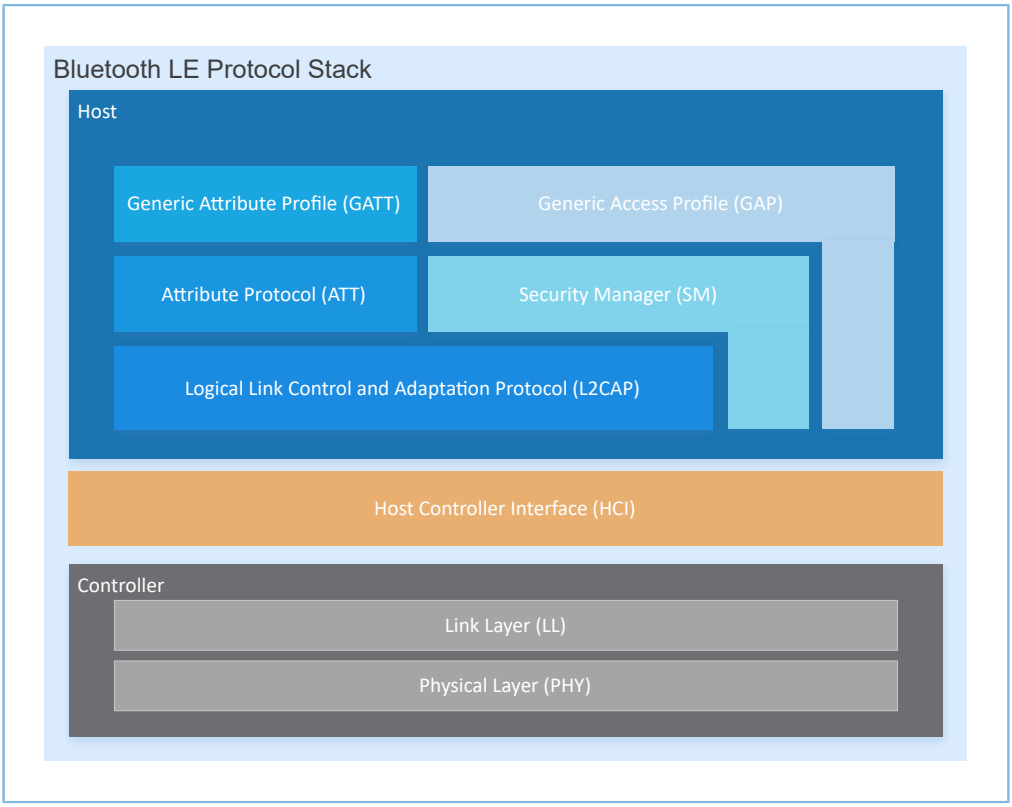

图 1-1 低功耗蓝牙协议栈架构

低功耗蓝牙协议栈由控制器（Controller）、主机控制接口（HCI）和主机（Host）组成。

控制器（Controller）

- 物理层（Physical Layer，PHY）：支持1 Mbps和2 Mbps的自适应跳频GFSK（高斯频移键控）射频（RF）操作。
- 链路层（Link Layer，LL）：控制设备的射频状态，支持五种设备状态（Standby、Advertising、Scanning、Initiating或Connection），并可根据实际需求切换状态。

主机控制接口（HCI）

- 主机控制接口（Host Controller Interface，HCI）：提供Host与Controller之间的通信接口。该接口层的实现可以是软件接口，也可以是标准硬件接口，例如UART、Secure Digital（SD）或USB。HCI commands和events通过该接口层在Host与Controller之间传递。

主机（Host）

- 逻辑链路控制和适配协议层（Logical Link Control and Adaptation Protocol，L2CAP）：为上层提供多路复用、数据分段与重组服务，并且支持逻辑端对端的数据通信。
- 安全管理层（Security Manager，SM）：定义配对和密钥分发的方法，为上层协议栈和应用程序提供端到端的安全连接和数据交换功能。
- 通用访问规范层（Generic Access Profile，GAP）：为上层应用和Profiles提供与协议栈通信交互的接口，主要包括广播、扫描、连接发起、服务发现、连接参数更新、安全过程发起与响应等功能。
- 属性协议层（Attribute Protocol，ATT）：定义了服务端和客户端之间的服务数据交互协议。

- 通用属性规范层（Generic Attribute Profile，GATT）：基于ATT协议之上，定义了一系列用于GATT Client和GATT Server之间服务数据交互的通信过程，供上层应用、Profile及Service使用。

提示:

更多Bluetooth LE技术及其协议的相关资料，请访问Bluetooth SIG的官方网站www.bluetooth.com。

GAP、SM、L2CAP及GATT规范包含在Bluetooth Core Spec中，其他Bluetooth LE应用层的Profiles/Services规范可以在GATT Specs页面下载。Bluetooth LE应用可能会用到的Assigned Numbers、IDs及Codes均列在Assigned Numbers页面。

---

2 GR5525低功耗蓝牙软件平台

GR5525 SDK是基于GR5525芯片定义的低功耗蓝牙应用开发的软件套件，包括Bluetooth LE 5.3 API、System API和外设驱动API，并提供丰富的蓝牙和外设应用示例工程和使用说明文档。应用开发者可以基于GR5525 SDK的示例工程进行快速产品开发和迭代。

2.1 硬件架构

GR5525的硬件框图如下所示。

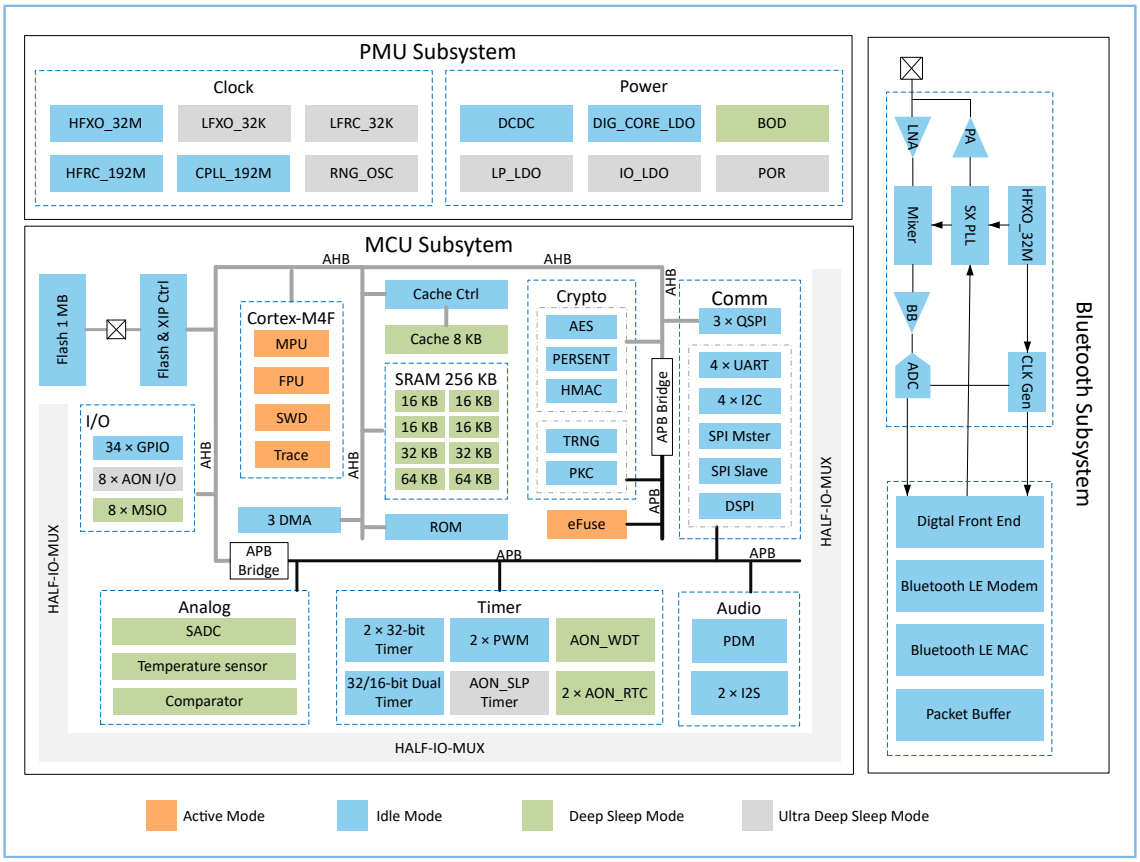

图 2-1 GR5525硬件框图

- Bluetooth subsystem：
  - 包含蓝牙5.3版本2.4 GHz射频信号收发器和数字通信控制器
- MCU subsystem：
  - 包含ARM® Cortex®-M4F内核以及存储与外设单元
  - 安全模组同时支持安全应用与安全启动执行
- Power Management Unit（PMU）subsystem：
  - 电源管理模组，用于给内部模组和外部外设模组提供高效的电源供应

- 在待机状态下支持极低功耗模式，采用HFRC_192M、RNG_OSC、LFRC_32K、Wakeup GPIOs（Wake up）、Low-power comparator（LP Comp.）和Power state controller（Power Sequencer）模块来控制系统或外设模组的供电状态。

提示:

关于GR5525系列芯片各模块的详细介绍，请参考《GR5525 Datasheet》。

2.2 软件架构

GR5525 SDK的软件架构如下图所示。

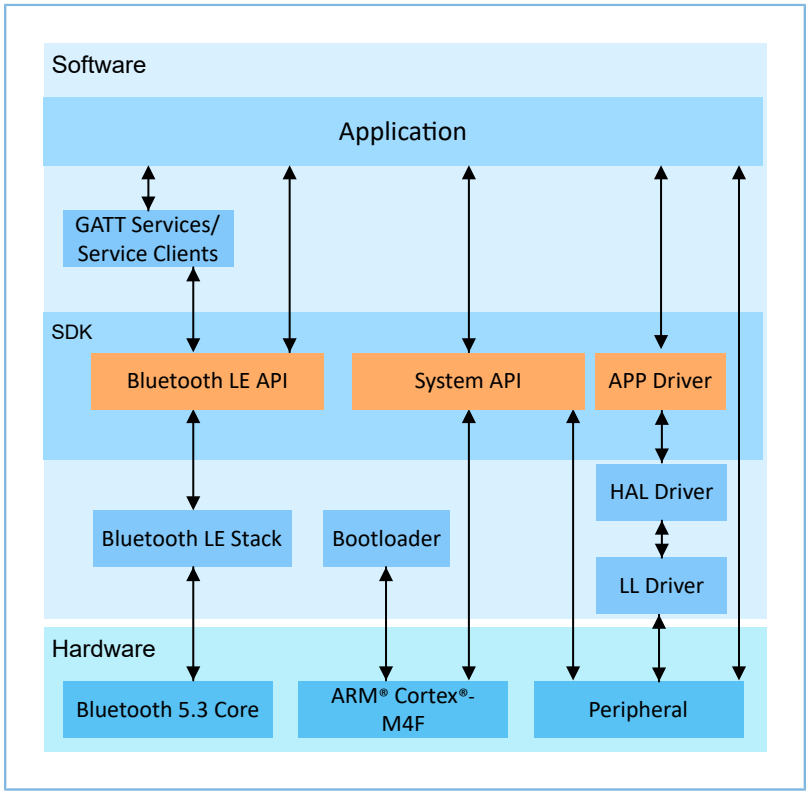

图 2-2 GR5525软件架构

- Bootloader

  固化在芯片中的引导程序，负责初始化芯片的软硬件环境，校验并启动应用程序。

- Bluetooth LE Stack

  低功耗蓝牙协议实现核心，由控制器（Controller）、主机控制接口（HCI）和主机（Host）协议组成（包括ATT、L2CAP、GAP、SM、GATT），支持Broadcaster、Observer、Peripheral和Central角色。

- HAL Driver

  硬件抽象驱动层，介于APP驱动层和LL驱动层之间的一个抽象层。其提供一组标准化的API接口，可方便APP驱动层通过调用HAL层API访问底层外设资源。

说明:

HAL层的API接口通常只适用于开发底层驱动和系统级服务，而不适用于普通应用程序开发。因此，不推荐开发者直接调用HAL层的API接口。

- LL Driver

  底层驱动层，直接利用寄存器操作驱动外设，包括对外设的控制和管理。

- Bluetooth LE SDK

  软件开发工具包，提供简单易用的Bluetooth LE API和System API。

  - Bluetooth LE API包括L2CAP、GAP、SM和GATT API。
  - System API提供对非易失性数据存储系统（NVDS）、固件升级（DFU）、系统电源管理以及通用系统级访问的接口。
  - APP Driver API提供UART、I2C以及ADC等通用外设的API定义。其调用HAL/LL层 API，实现应用功能。

- Application

  SDK包提供丰富的蓝牙及外设示例工程，且每个示例工程都包含编译后的二进制文件，用户可以直接将其下载至芯片中运行和测试。对于大部分蓝牙应用，SDK包中的GRToolbox（Android）也提供了对应的功能，可方便用户测试。

2.3 存储器映射

GR5525系列SoC的存储器映射如下图所示。

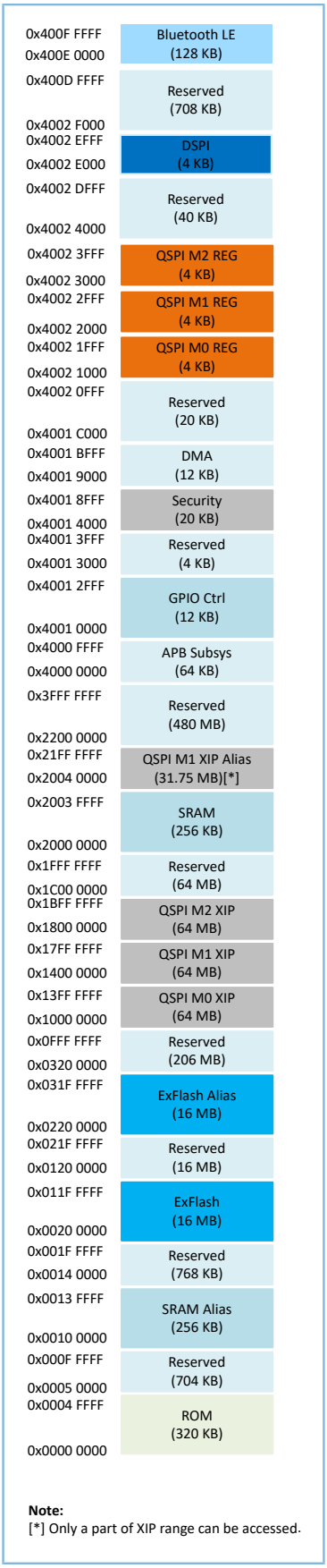

图 2-3 GR5525存储器映射

- RAM存储：0x0010_0000 ～ 0x0013_FFFF或0x2000_0000 ～ 0x2003_FFFF，共256 KB。
  - 0x2000_0000 ～ 0x2003_FFFF：该区域支持位段操作，它对应的位段地址为0x2200_0000 ～ 0x227F_FFFF，该区域可进行数据原子操作。SDK中RW、ZI、HEAP、STACK等变量位于该区域。
  - 0x0010_0000 ～ 0x0013_FFFF：由于 Cortex-M4F 总线架构的特点该区域的访问效率高于其他区域，故SDK中RAM_CODE可执行代码位于该区域。

说明:

QSPI0/QSPI1/QSPI2均支持XIP模式。在这种模式下，可以将QSPI Flash的数据空间映射到内存中，方便直接对内存地址进行操作。

- Flash存储：0x0020_0000 ～ 0x011F_FFFF或0x0220_0000 ～ 0x031F_FFFF，共16 MB。
  - 0x0020_0000 ～ 0x011F_FFFF区域存放代码以及非加密模式下的数据。
  - 0x0220_0000 ～ 0x031F_FFFF区域存放加密模式下的数据。

说明:

GR5525芯片内部Flash封装为1 MB，地址为0x0020_0000 ～ 0x002F_FFFF。

2.4 Flash存储映射

GR5525封装了一个采用XQSPI总线接口的可擦除外部Flash存储器。该Flash物理上由若干个4 KB大小的Flash扇区（Sector）组成；逻辑上可根据不同的应用场景，划分为不同用途的存储区域。

图 2-4为GR5525典型应用场景的Flash存储布局。

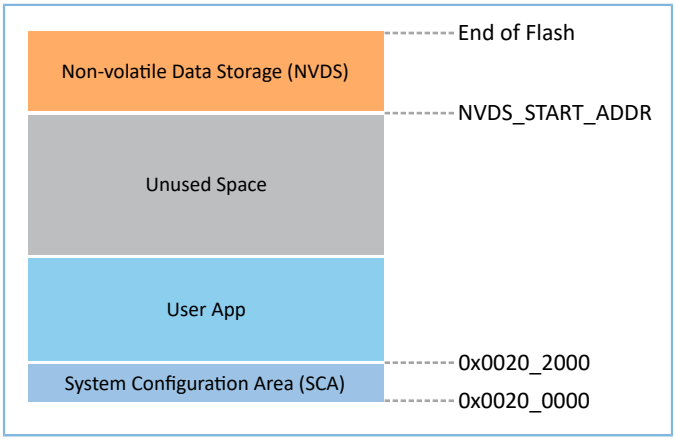

图 2-4 Flash存储布局

- System Configuration Area（SCA）：系统配置区，主要用于存储系统启动参数等配置信息。
- User App：Application Firmware存储区域，主要用于存储应用固件。
- Unused Space：空闲区域，开发者可以自行使用该区域。例如，在DFU升级过程中，使用Unused Space临时存储新的Application Firmware。

- Non-volatile Data Storage（NVDS）：非易失性数据存储区域。

说明:

NVDS缺省占用Flash的最后两个Sector。开发者也可以根据产品的Flash存储布局，合理配置NVDS的起始地址与所占用的Flash Sector数，具体配置方法参考4.3.2.1 配置custom_config.h。

NVDS起始地址需与Flash Sector的起始地址对齐。

2.4.1 SCA

系统配置区（SCA）占用Flash的前两个Sector（共8 KB，0x0020_0000 ～ 0x0020_2000），其主要存储系统启动过程使用的标志以及其他系统配置参数。

下载固件时，下载算法或GProgrammer会根据Application Firmware中的BUILD_IN_APP_INFO结构体生成SCA Image，并将其与应用固件一并烧写至Flash中（Image_Info被存放在SCA中）。系统启动时Bootloader根据SCA区域中的启动信息进行校验，校验通过后再跳转至固件的入口地址。

BUILD_IN_APP_INFO的定义和配置如下：

提示:

该结构体位于SDK_Folder\platform\soc\common\gr_platform.c，其中SDK_Folder为SDK的根目录。

```
const APP_INFO_t BUILD_IN_APP_INFO __attribute__((section(".app_info"))) =
#endif
{
    .app_pattern      = APP_INFO_PATTERN_VALUE,
    .app_info_version = APP_INFO_VERSION,
    .chip_ver         = CHIP_VER,
    .load_addr        = APP_CODE_LOAD_ADDR,
    .run_addr         = APP_CODE_RUN_ADDR,
    .app_info_sum     = CHECK_SUM,
    .check_img        = BOOT_CHECK_IMAGE,
    .boot_delay       = BOOT_LONG_TIME,
    .sec_cfg          = SECURITY_CFG_VAL,
#ifdef APP_INFO_COMMENTS
    .comments         = APP_INFO_COMMENTS,
#endif
};
```

- app_pattern：固定值0x47525858。
- app_info_version：固件信息版本，与APP_INFO_VERSION对应。
- chip_ver：固件对应的芯片版本，与*custom_config.h*中的CHIP_VER对应。
- load_addr：固件存储地址，与*custom_config.h*中的APP_CODE_LOAD_ADDR对应。
- run_addr：固件运行地址，与*custom_config.h*中的APP_CODE_RUN_ADDR对应。
- app_info_sum：固件信息的校验和，由CHECK_SUM宏自动计算。

- check_img：系统启动配置参数，与*custom_config.h*中的BOOT_CHECK_IMAGE对应。当此参数配置为“1”时，启动时Bootloader会对固件进行校验。
- boot_delay：启动配置参数，与*custom_config.h*中的BOOT_LONG_TIME对应。当此参数配置为“1”时，系统冷启动时将增加1 秒延时。
- sec_cfg：安全配置参数，保留值。
- comments：固件描述信息，最大长度为12 字节。

System Configuration Area布局如下图所示。

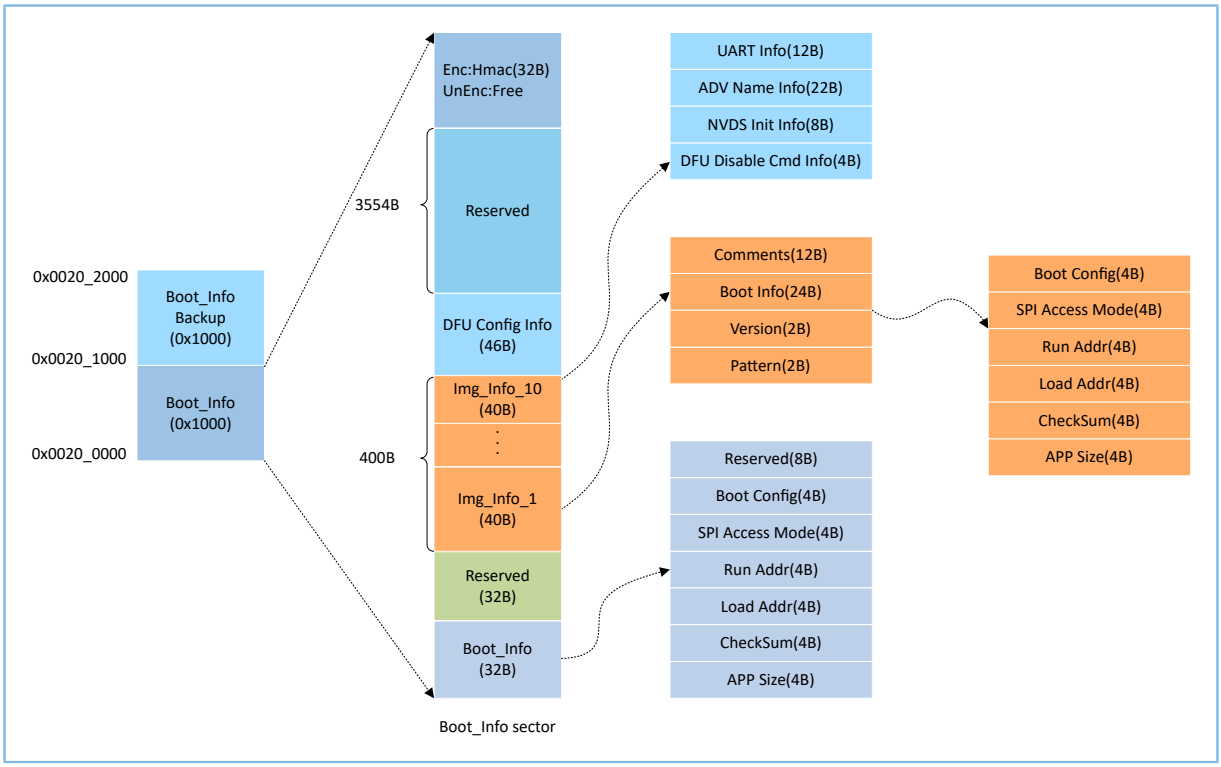

图 2-5 System Configuration Area布局

- Boot_Info与Boot_Info Backup存储相同信息，Boot_Info Backup为Boot_Info的备份。
  - 在非安全模式下，Bootloader会默认从Boot_Info中获取启动信息。
  - 在安全模式下，Bootloader会先校验Boot_Info，如果Boot_Info校验不过，则会校验Boot_Info Backup，并从Boot_Info Backup中获取启动信息。
- Boot_Info（32B）区域中存储固件启动信息。系统启动时，Bootloader会根据启动信息进行校验，校验通过后跳转至固件的入口地址。
  - Boot Config：系统启动配置信息。
  - SPI Access Mode：SPI访问方式配置。为系统固定配置，用户无法修改。
  - Run Addr：固件运行地址，与BUILD_IN_APP_INFO中的run_addr对应。
  - Load Addr：固件存储地址，与BUILD_IN_APP_INFO中的load_addr对应。
  - CheckSum：固件校验和，生成固件后，由下载算法自动计算。

  - APP Size：固件的Size信息，生成固件后，由下载算法自动计算。

- Img_Info区域可存储至多10个固件信息。当使用GProgrammer下载固件或使用DFU升级固件时，固件信息会被存储至Img_Info区域。
  - Comments：固件描述信息，最大长度为12个字符。生成固件后，下载算法使用固件文件名作为Comments信息。
  - Boot Info（24B）：固件启动信息，与Boot_Info（32B）的低24 Bytes数据相同。
  - Version：固件版本信息，与*custom_config.h*中VERSION对应。
  - Pattern：固定值0x4744。
- DFU Config Info区域存储ROM中DFU模块的配置信息。
  - UART Info：DFU模块的UART串口相关配置，包括状态位、波特率、GPIO配置等。
  - ADV Name Info：DFU模块的广播相关配置，包括状态位、广播名、广播长度。
  - NVDS Init Info：DFU模块的NVDS系统的初始化配置，包括状态位、NVDS区域大小、起始地址。
  - DFU Disable Cmd Info：DFU模块的DFU禁用命令配置，包括状态位和Disable DFU Cmd（2B，设置格式为Bitmask），可通过设置Disable DFU Cmd值来禁用某些DFU命令。
- HMAC区域存储HMAC校验值。该区域仅在安全模式下有效。

2.4.2 NVDS

NVDS是一个轻量级逻辑数据存储系统，它依赖于Flash硬件抽象层（Flash HAL）。其存储于Flash中，掉电时数据不会丢失。NVDS默认占用Flash最后两个Sector，也可由开发者自行设定占用的Flash Sector数。在NVDS区域中，最后一个扇区用于碎片整理，其余扇区用于数据存储。

NVDS系统适合存储小块数据，例如应用程序的配置参数、校准数据、状态和用户信息等。Bluetooth LE协议栈也会使用NVDS存储设备绑定等参数。

NVDS系统具有以下特性：

- 每个存储项（TAG）具有唯一的标识TAG ID。用户程序可以根据TAG ID对数据内容进行读取和更改操作，而无需关心数据存储的物理地址。
- 针对Flash存储介质的特性进行了优化，支持数据校验、Word对齐、碎片整理和擦写平衡。
- 存储区域的大小和起始地址可配置，Flash存储区以Sector为单位，一个Sector的大小为4 KB，NVDS存储区域可配置为若干个Sector；配置的起始地址需按4 KB对齐。

说明:

- 开发者可在*custom_config.h*文件中添加宏NVDS_START_ADDR和更改宏NVDS_NUM_SECTOR，配置NVDS区域的起始地址和占用大小。
- Bluetooth LE 协议栈与Application共享相同的NVDS存储区域，但TAG ID命名空间被划分为不同类别，开发者只能使用分配给Application的TAG ID命名类别。
  - Application必须使用NV_TAG_APP(idx)获取应用程序数据的TAG ID。该TAG ID被用作NVDS API的参数。
  - Application不能将idx直接作为NVDS API的参数。idx取值范围是0x4000 ～ 0x7FFF。
- 在Application第一次运行前，开发者可使用工具GProgrammar将 Bluetooth LE 协议栈和Application所使用的TAG ID初始值写入到NVDS。
- 若开发者不使用GR5525 SDK缺省的NVDS区域，而需自行指定NVDS区域，则必须确保GProgrammar中配置的NVDS起始地址配置按4 KB对齐。

NVDS存储数据的格式如图 2-6所示：

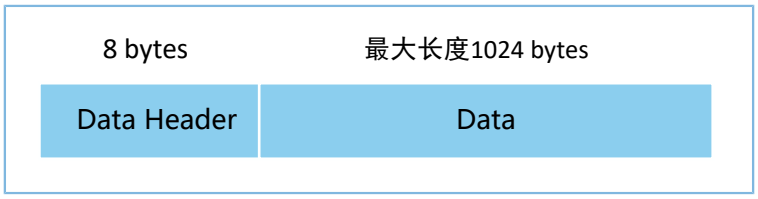

图 2-6 NVDS存储数据格式

数据头（Data Header）格式如表 2-1 所示：

表 2-1 Data Header格式

Byte	Name	Description
0-1	tag	数据的tag标识
2-3	len	数据的长度
4-4	checksum	数据头的校验和
5-5	value_cs	数据的校验和
6-7	reserved	保留字段

GR5525 SDK提供了下列NVDS API，可方便开发者操作Flash中的非易失性数据。

表 2-2 NVDS API

函数原型	描述
uint8_t nvds_init(uint32_t start_addr, uint8_t sectors)	初始化NVDS使用的Flash Sector。
uint8_t nvds_get(NvdsTag_t tag, uint16_t *p_len, uint8_t *p_buf)	从NVDS中读取tag标识对应的数据。
uint8_t nvds_put(NvdsTag_t tag, uint16_t len, const uint8_t *p_buf)	将数据写入到NVDS并使用tag标识。若为首次写数据，则需创建一个tag标识。
uint8_t nvds_del(NvdsTag_t tag)	删除NVDS中tag标识对应的数据。

函数原型	描述
uint16_t nvds_tag_length(NvdsTag_t tag)	获取指定Tag标识的数据长度。
uint8_t nvds_drv_func_replace(nvds_drv_func_t *p_nvds_drv_func)	替代直接操作Flash的相关API。
uint8_t nvds_func_replace(nvds_func_t *p_nvds_func)	替代NVDS操作相关API。
void nvds_retention_size(uint8_t bond_dev_num)	为设备绑定信息预留空间，且空间大小由设备绑定数决定。

说明:

关于NVDS API的详细说明，可参考NVDS头文件SDK_Folder\components\sdk\gr55xx_nvds.h。

2.5 RAM存储映射

GR5525的RAM为256 KB，起始地址为0x2000_0000，由8个内存块（RAM Block）组成（前4个内存块的大小均为16 KB，第5个和第6个内存块大小为32 KB，最后的2个内存块大小为64 KB）。每个RAM内存块均可由软件独立打开/关闭电源。

说明:

GR5525为起始地址0x2000_0000的RAM提供一个起始地址为0x0010_0000的Alias Memory，见图 2-3。

- 起始地址为0x2000_0000区域支持位段操作，它对应的位段起始地址是0x2200_0000。
- 由于 Cortex®-M4F总线架构的特点起始地址为0x0010_0000区域的访问效率高于其他区域，故代码的运行地址在0x0010_0000区域地址范围，可以加快运行速度。
- GR5525 SDK中RW、ZI、HEAP、STACK使用0x2000_0000区域的RAM，RAM_CODE可执行代码使用0x0010_0000区域的RAM。

256 KB RAM存储布局如图 2-7所示：

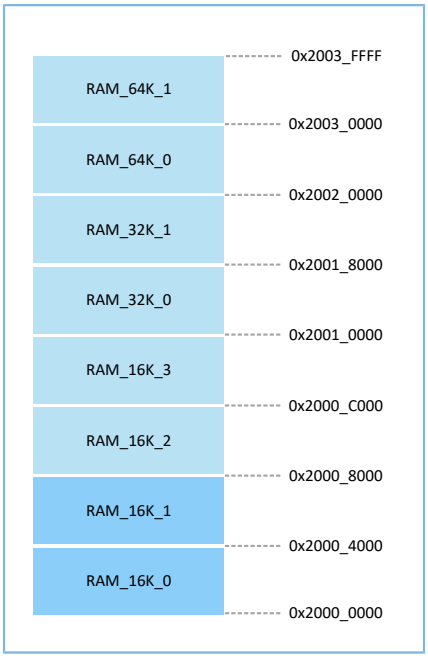

图 2-7 256 KB RAM存储布局

程序运行模式可配置为Execute in Place（XIP）模式或Mirror模式。详细的配置方法参考4.3.2.1 配置custom_config.h中的“APP_CODE_RUN_ADDR”。这两种运行模式有不同的RAM布局。

表 2-3 程序运行模式

运行模式	描述
XIP模式	片上运行模式，用户应用程序存储在片上Flash空间，程序运行空间和加载空间相同。系统完成上电配置后，通过Cache Controller直接从Flash空间取指运行。
Mirror模式	镜像运行模式，用户应用程序存储在片上Flash空间，程序的运行空间定义在RAM空间。在程序启动阶段，会在校验完成后，将程序从外部Flash空间加载到RAM空间，并跳转到RAM中进行运行。

说明:

由于XIP模式运行时需要持续访问Flash，因此该模式下的运行功耗会略高于Mirror模式。

2.5.1 XIP模式的典型RAM布局

图 2-8为XIP模式的典型RAM布局，开发者可以根据产品需要对其进行修改。

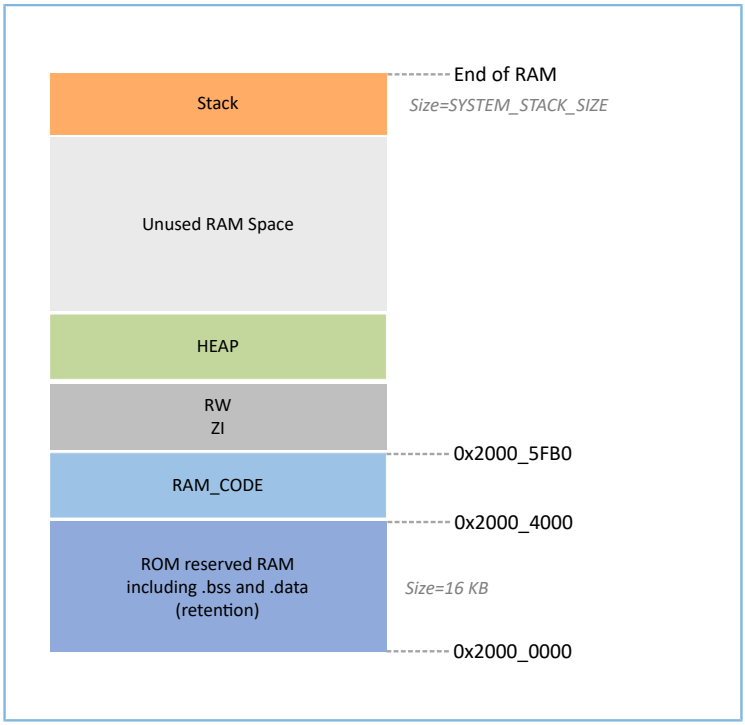

图 2-8 XIP模式的RAM布局

RAM CODE为在RAM中执行的代码，为提高执行效率，建议将其定位到同物理地址的0x0010_0000 Aliasing Memory区域 。

XIP模式布局允许在代码加载处直接执行Application的固件，从而让Application能使用更多RAM内存。在对Flash存储内容进行更新时，会关闭XIP模式，在最小颗粒度的擦写（写256 Bytes、 擦4 KB）期间所有中断无法生成。

说明:

- QSPI0/QSPI1/QSPI2均支持XIP模式。在这种模式下，可以将QSPI Flash的数据空间映射到内存中，方便直接对内存地址进行操作。
- 开发者可以根据实际需求合理地添加自定义section，但不要轻易删改SDK默认的Scatter文件布局，如从Scatter文件中删除“RAM_CODE”段。关于Scatter文件描述，请参考4.3.2.2 配置存储器布局。

2.5.2 Mirror模式的典型RAM布局

图 2-9为Mirror模式的典型RAM布局，开发者可以根据产品需求对其进行修改。

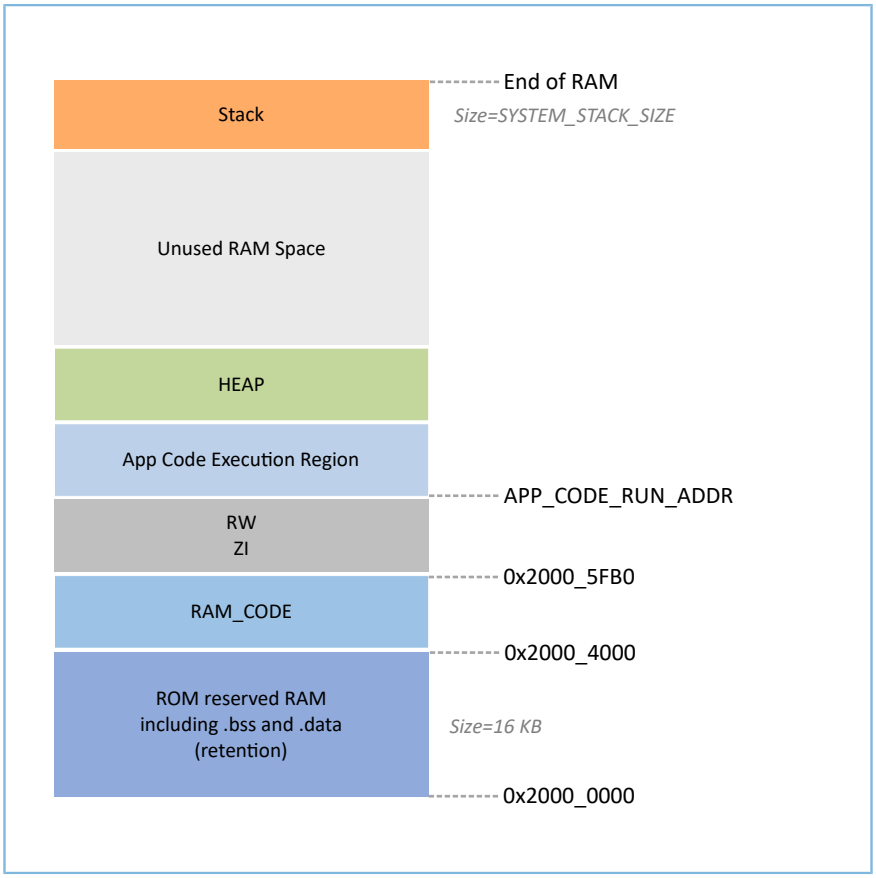

图 2-9 Mirror模式的RAM布局

Mirror模式布局允许在RAM中执行Application的固件。芯片上电之后，会进入冷启动流程。Bootloader会将Application的固件从Flash中复制到名为“App Code Execution Region”的RAM段。睡眠模式的芯片被唤醒后进入热启动流程。为减少热启动时间，Bootloader不会重新复制Application的固件到名为“App Code Execution Region”的RAM段中。

“App Code Execution Region”段的起始位置由*custom_config.h*中的宏APP_CODE_RUN_ADDR决定。开发者需要根据Application的.data和.bss实际使用情况，来确定代码运行地址APP_CODE_RUN_ADDR的值，避免与低地址处的.bss段或高地址处的Call Stack段地址重叠。开发者可根据*.map*文件来获得RAM各段的分布情况。

建议开发者使用RAM Aliasing Memory地址（0x0010_0000 ～ 0x0013_FFFF）来设置APP_CODE_RUN_ADDR。若出现RAM段重叠，在工程构建时会出现error并提示重叠位置，帮助开发者确认并快速定位RAM段重叠情况。

2.5.3 RAM电源管理

每一个RAM Block可以处于三种不同电源状态：Full Power、Retention Power或Power Off。

- Full Power：系统处于Active状态，MCU可以进行RAM Block读写。
- Retention Power：系统进入Sleep状态模式时，RAM Block中储存的数据不会丢失，可供系统从Sleep状态恢复到Active状态使用。
- Power Off：系统关机时，RAM Block会掉电，其存储的数据也会丢失。因此，开发者需提前保存数据。

GR5525的电源管理单元（PMU）在系统启动时默认开启全部RAM电源。GR5525 SDK中也提供了完备的RAM电源管理API，开发者可以根据应用需求，合理配置RAM Block电源状态。

系统启动时，默认启用自动RAM功耗管理模式：根据Application的RAM使用情况，自动进行RAM Block电源状态控制，具体配置规则如下：

- 在系统Active状态下，将未使用的RAM Block设置为“Power Off”状态，使用的RAM Block设置为“Full Power”状态。
- 当系统进入Sleep状态时，将未使用的RAM Block保持为“Power Off”状态，而使用的RAM Block设置为“Retention Power”状态。

在实际应用中，建议RAM配置如下：

- 在Bluetooth LE应用中，RAM_16K_0和RAM_16K_1的前 8 KB预留给Bootloader和Bluetooth LE协议栈使用，Application不可使用。在系统Active下，RAM_16K_0和RAM_16K_1应处于“Full Power”状态；在系统Sleep期间，它们应处于“Retention Power”状态。非Bluetooth LE类MCU应用可以使用这两个RAM Block。
- RAM_16K_2及其他的RAM Block的用途可由Application进行规划定义。通常，将用户数据和需要在RAM中执行的代码段定义在从RAM_16K_2开始的连续区间；将函数调用栈（Call Stack）的栈顶定义在RAM的高端地址。这些RAM Block的电源状态可以全部开启，也可以由Application自行控制。

说明:

- 仅当RAM Block处于“Full Power”状态时，MCU才能对其进行访问。
- 更多RAM电源管理API的详细说明，可参考`SDK_Folder\components\sdk\platform_sdk.h`。

2.6 GR5525 SDK目录结构

GR5525 SDK的文件夹目录结构如下图所示。

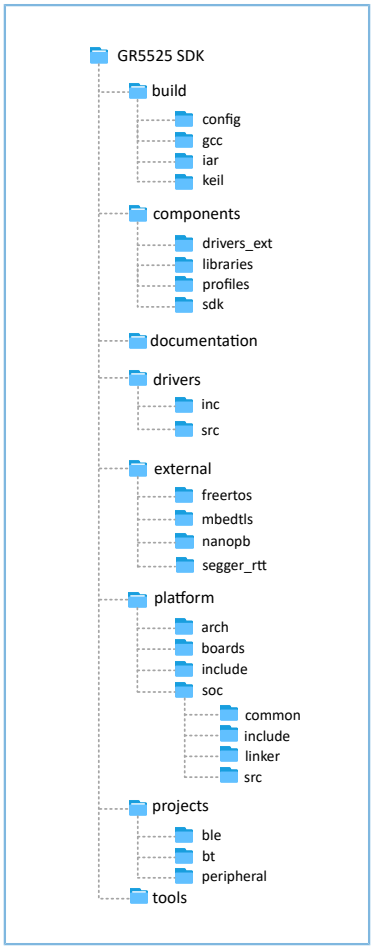

图 2-10 GR5525 SDK目录

GR5525 SDK中各文件夹的详细描述参见下表。

表 2-4 GR5525 SDK文件夹

文件夹	描述
build\config	工程配置目录，用于存放*custom_config.h*模板文件。该文件主要用于配置工程和参数。
build\gcc	运行GCC开发环境所需要使用的工具。
build\keil	运行Keil开发环境所需使用的工具。
build\iar	运行IAR开发环境所需要使用的工具。
components\drivers_ext	开发板上第三方元器件的驱动。
components\libraries	GR5525 SDK提供的libraries。
components\profiles	GR5525 SDK提供的GATT Services/Service Clients实现示例的源文件。
components\sdk	GR5525 SDK提供的API头文件。
documentation	GR5525 API Reference。
drivers\inc	易于Application开发者使用的驱动API头文件。
drivers\src	易于Application开发者使用的驱动API源代码。

文件夹	描述
external\freertos	第三方程序，FreeRTOS源代码。
external\mbedtls	第三方程序，Mbedtls源代码。
external\nanopb	第三方程序，nanopb源代码。
external\segger_rtt	第三方程序，SEGGER RTT源代码。
platform\arch	CMSIS的Toolchain文件。
platform\boards	存放GR5525 Starter Kit开发板的板级初始化源文件，主要实现对板级基础外设的初始化。
platform\include	存放与平台相关公共头文件。
platform\soc\common	存放Goodix全系Bluetooth LE SoC兼容的公共源文件，如*gr_interrupt.c*、*gr_platform.c*和*gr_system.c*。
platform\soc\linker	链接器使用的符号表文件和库文件。
platform\soc\include	存放SoC寄存器、时钟配置等与底层驱动强相关的公共头文件。
platform\soc\src	存放*gr_soc.c*，主要实现SoC芯片强相关的一些初始化流程实现，如Flash、NVDS初始化，晶振配置和PMU校准等。
projects\ble	Bluetooth LE Application工程示例，比如Heart Rate Sensor，Proximity Reporter。
projects\bt	BT示例工程
projects\peripheral	芯片外设工程示例。
tools	GR5525的开发与调试工具软件。

3 启动流程（Bootloader）

GR5525支持两种固件运行模式：XIP模式和Mirror模式。系统上电后，启动引导程序（Bootloader）从系统配置区（SCA）读取系统启动配置信息，并据此进行应用固件完整性校验、初始化配置后，跳转到代码运行空间执行固件。不同的运行模式，启动流程略有差异。

- XIP模式下，启动引导程序在完成应用固件校验后，初始化Cache和XIP控制器，然后跳转至Flash空间的代码运行地址执行代码。
- Mirror模式下，启动引导程序在完成应用固件校验后，根据系统配置信息，将Flash空间的固件加载到对应的RAM运行空间后，跳转至RAM空间进行代码执行。

GR5525 SDK上Application的启动流程如下图所示。

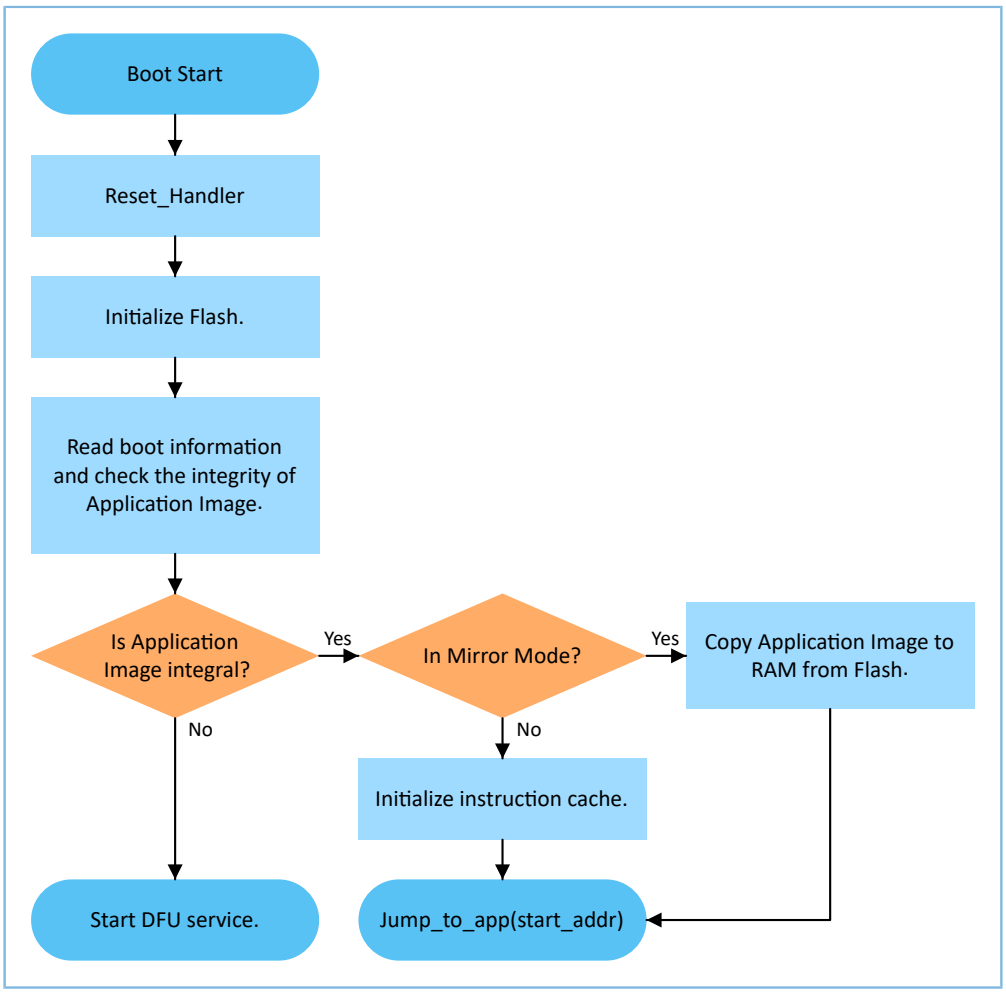

图 3-1 Application启动流程

1. 设备上电后，CPU将跳转至0x0000_0000处，从该地址处取出C-STACK栈顶指针并赋值给MSP，然后PC指向0x0000_004处，执行ROM中的Reset_Handler，进入Bootloader。
2. Bootloader执行Flash初始化。
3. Bootloader从Flash中的SCA读取启动信息，并执行Application Firmware的完整性检查。

说明:

GR5525支持通过安全模式对Application Firmware进行加密和签名。

- 安全模式：若使能安全模式，启动引导程序从SCA读取启动信息并进行HMAC校验；校验成功后，会解密SCA启动信息，后续执行安全启动流程的验证签名流程，用于保证固件完整性和防止篡改，防止伪装等；若验证签名成功，则会使能自动解密功能。
- 非安全模式：若未使能安全模式，启动引导程序使用SCA启动信息对Application Firmware进行CRC完整性校验。

4. 若固件完整性检查失败，则启动Bluetooth LE DFU Service。
5. 若固件完整性检查通过，则Bootloader将判断运行模式。
    - 对于XIP模式，Bootloader会在完成XIP配置后跳转至Flash中的Application Firmware开始执行。
    - 对于Mirror模式，Bootloader会将Application Firmware从Flash中拷贝至RAM中的指定区段，然后执行RAM中的Application Firmware。

4 使用SDK开发调试

本章将以Keil为例，介绍如何基于SDK完成Bluetooth LE Application的创建、编译、下载以及调试。

4.1 安装Keil

Keil MDK-ARM IDE（Keil）是ARM®公司提供的用于Cortex®和ARM设备的集成开发环境（IDE）。开发者可从官方网站https://www.keil.com/demo/eval/arm.htm下载Keil安装包并进行安装。GR5525 SDK必须运行在Keil V5.20及以上的版本。

说明:

关于Keil MDK-ARM IDE的使用，可查看ARM提供的在线用户手册：https://www.keil.com/support/man_arm.htm。

Keil启动后的主界面如下图所示。

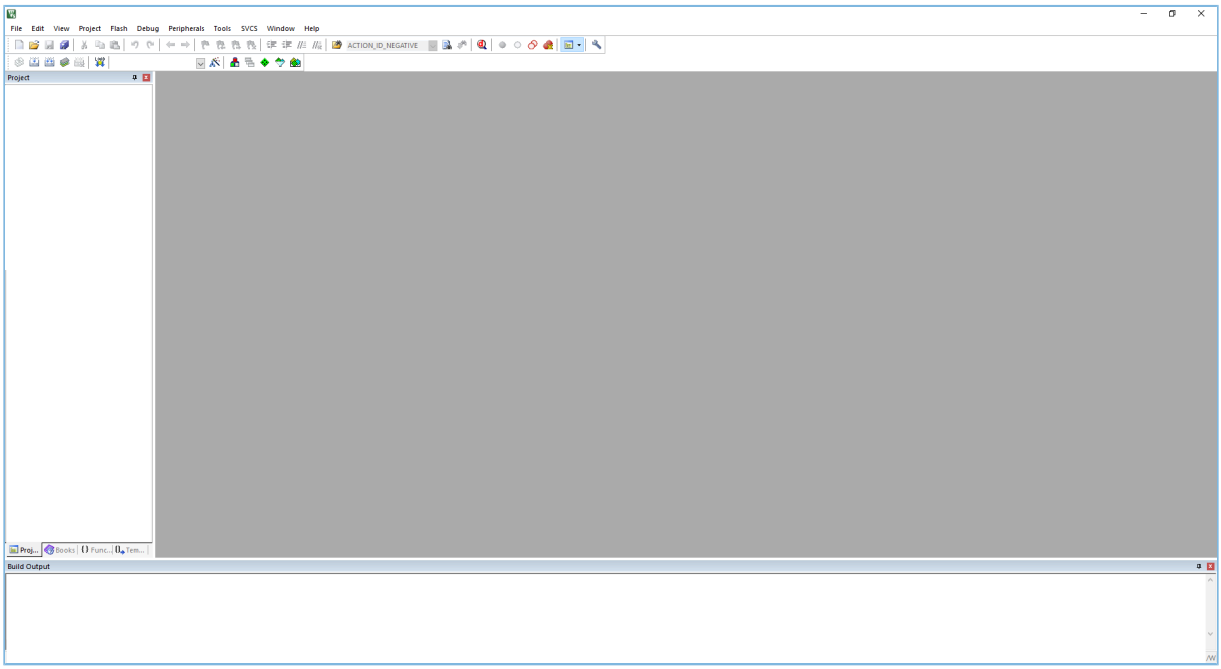

图 4-1 Keil软件界面

表 4-1 为Keil的常用功能按钮，包括：

表 4-1 Keil常用功能按钮

Keil按钮	功能描述
	Options for Target
	Start/Stop debug session
	Download
	Build

4.2 安装SDK

GR5525 SDK包为.zip文件，直接解压即可使用。

说明:

- SDK_Folder为GR5525 SDK的根目录。
- Keil_Folder为Keil的根目录。

4.3 创建Bluetooth LE Application

本节介绍如何基于Keil开发环境，利用GR5525 SDK快速创建用户自定义的Bluetooth LE应用。

4.3.1 准备ble_app_example

基于GR5525 SDK提供的模板工程，创建一个新工程。

进入`SDK_Folder\projects\ble\ble_peripheral\`，拷贝ble_app_template到当前目录，并将其重命名为“ble_app_example”。将`ble_app_example\Keil_5`中的*.uvoptx*和*.uvprojx*的主文件名修改为“ble_app_example”。

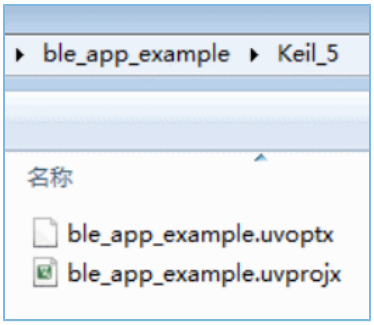

图 4-2 ble_app_example文件夹

双击*ble_app_example.uvprojx*，在Keil中打开工程示例。在，打开“Options for Target 'GRxx_Soc'”窗口，选择“Output”标签页，在“Name of Executable”栏中输入“ble_app_example”，将生成的目标文件名称设置为“ble_app_example”。

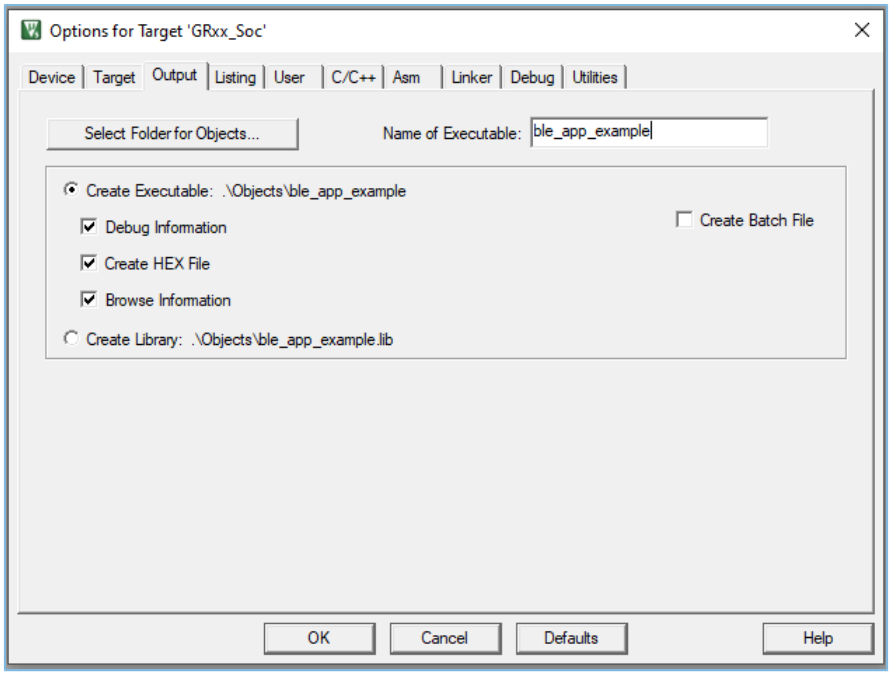

图 4-3 修改Name of Executable

在Keil Project Explore中，可查看到ble_app_example工程下的所有groups。

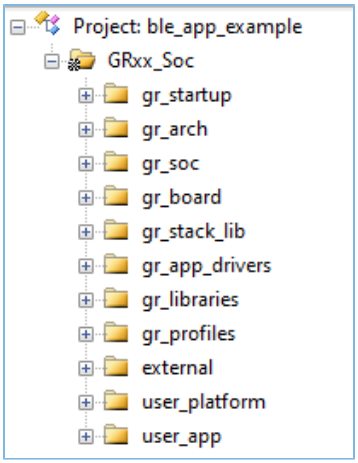

图 4-4 ble_app_example groups

ble_app_example工程下的groups主要分为两类：SDK groups和User groups。

- SDK groups

  包括gr_startup、gr_arch、gr_soc、gr_board、gr_stack_lib、gr_app_drivers、gr_libraries、gr_profiles和external。

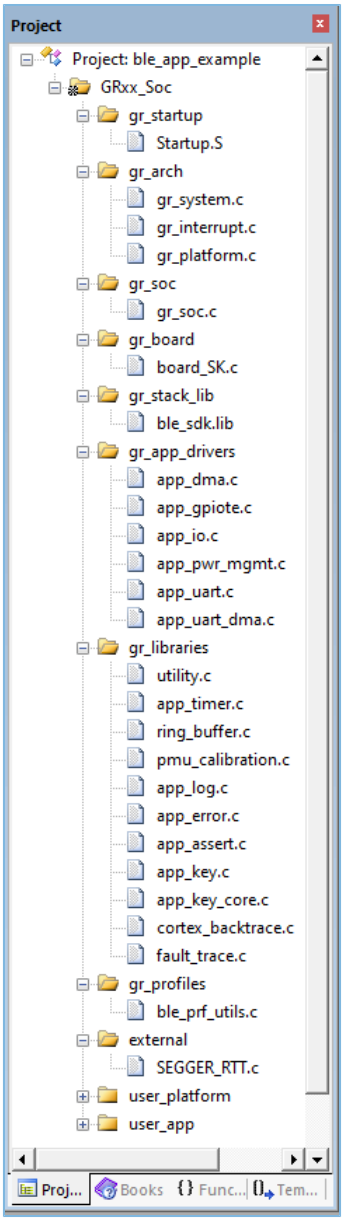

图 4-5 SDK groups

SDK groups下的源文件无需修改，各group的具体描述如下：

表 4-2 SDK groups

SDK group名称	描述
gr_startup	系统启动文件。
gr_arch	System Core、PMU的初始化配置文件和系统中断的接口实现。
gr_soc	与SoC相关的处理文件，主要是在进入main函数之前，对Clock、PMU、Vector等模块进行初始化与校准。
gr_board	板级描述文件，主要实现Log、Key、Led等相关组件。
gr_stack_lib	GR5525 SDK lib文件。

SDK group名称	描述
gr_app_drivers	易于Application开发者使用的驱动API源文件。开发者可根据实际需求，添加项目所需APP驱动。
gr_libraries	SDK提供的常用辅助软件模块、外设驱动的开源文件。
gr_profiles	GATT Services/Service Clients源文件。开发者可根据实际需求，添加项目所需GATT源文件。
external	第三方程序的源文件，例如FreeRTOS、SEGGER RTT。开发者可根据实际需求，添加项目所需第三方程序。

- User groups

  User groups包括user_platform和user_app。

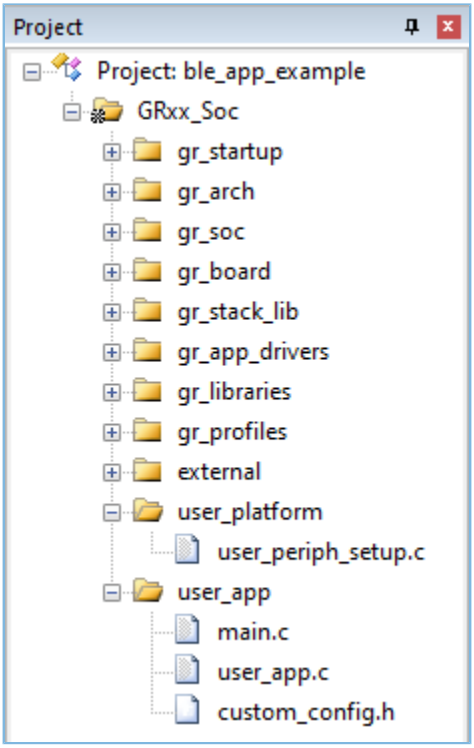

图 4-6 user_groups

User groups下的源文件需要开发者实现，各group的具体描述如下：

表 4-3 User groups

User group名称	描述
user_platform	软硬件资源的配置和应用程序的初始化，开发者需要根据实际项目需求，实现相应接口。
user_app	主函数入口、开发者创建的其他源文件，配置Bluetooth LE协议栈运行时参数和实现GATT Service/Service Client的事件处理函数。

4.3.2 配置工程

开发者需根据产品特性，配置相应的工程选项，包括NVDS、代码运行模式、存储器布局、After Build以及其他配置项等。

4.3.2.1 配置custom_config.h

*custom_config.h*用于配置Application工程参数。开发者可选择直接修改文件或者使用Wizard界面进行配置。

说明:

各Application示例工程的*custom_config.h*位于其工程目录下的`Src\config`。

- 修改文件

  GR5525 SDK提供了一个Application工程配置模板文件（`SDK_Folder\build\config\custom_config.h`），开发者可直接修改该模板文件，配置Application工程参数。

表 4-4 custom_config.h中的参数

宏	描述
SOC_GR5525	定义芯片版本号。
SYS_FAULT_TRACE_ENABLE	使能/禁用Trace信息打印。 使能该功能后，发生HardFault时，可打印Call Stack 中的Trace信息。 ◦ 0：禁用Trace打印 ◦ 1：使能Trace打印
ENABLE_BACKTRACE_FEA	使能/禁用栈回溯功能。 ◦ 0：禁用栈回溯功能 ◦ 1：使能栈回溯功能
APP_DRIVER_USE_ENABLE	App Drivers模块开关。 ◦ 0：禁用App Drivers模块 ◦ 1：使能App Drivers模块
APP_LOG_ENABLE	APP LOG模块开关。 ◦ 0：禁用Application中的Logs ◦ 1：使能Application中的Logs
APP_LOG_STORE_ENABLE	APP LOG STORE模块开关。 ◦ 0：禁用APP LOG STORE模块 ◦ 1：使能APP LOG STORE模块
DTM_TEST_ENABLE	DTM Test功能开关。 ◦ 0：禁用DTM Test功能 ◦ 1：使能DTM Test功能
PMU_CALIBRATION_ENABLE	使能/禁用PMU校准功能，使能该功能后，系统将自动监控温度与电压，并自适应调整。 ◦ 0：禁用PMU校准功能 ◦ 1：使能PMU校准功能

宏	描述
	说明： 在高/低温使用场景下，必须使能PMU校准功能。
NVDS_START_ADDR	NVDS占用Flash区域的起始地址。 默认状态下该宏在*cutom_config.h*中已被注释，开发者若需更改NVDS占用区域的配置，可启用该宏，再设置自定义值（必须按4 KB对齐）。 **说明：** 该起始地址不能设置在存储器的SCA或User App等已使用的区域内。
NVDS_NUM_SECTOR	NVDS占用的Flash Sector数。
SYSTEM_STACK_SIZE	Application所需的Call Stack的大小，默认为32 KB。 开发者可根据实际使用情况，调整该值，但不能少于6 KB。 **说明：** ble_app_example示例工程编译后，可在其目录下的`Keil_5\Objects\ble_app_example.htm`中查看参考的Maximum Stack Usage。
SYSTEM_HEAP_SIZE	Application所需要的Heap大小，默认为16 KB。 开发者可根据实际使用情况，调整该值。
APP_CODE_LOAD_ADDR *	程序存储空间的起始地址。 **说明：** 该地址应在Flash地址范围内。
APP_CODE_RUN_ADDR *	程序运行空间的起始地址。 若该地址的值与APP_CODE_LOAD_ADDR相等，则Application采用XIP模式运行。 若该地址的值在RAM地址范围内，则Application采用Mirror模式运行。
SYSTEM_CLOCK *	系统时钟频率。 ◦ 0：96 MHz ◦ 1：64 MHz ◦ 2：16 MHz（XO） ◦ 3：48 MHz ◦ 4：24 MHz ◦ 5：16 MHz ◦ 6：32 MHz（PLL）
CFG_LPCLK_INTERNAL_EN	是否使用芯片内部的OSC时钟作为Bluetooth LE低频睡眠时钟。若使用，则CFG_LF_ACCURACY_PPM被强制设置为500 PPM。 ◦ 0：不使用 ◦ 1：使用
CFG_LF_ACCURACY_PPM	Bluetooth LE低频睡眠时钟精度，其取值范围1 ～ 500，单位PPM。
BOOT_LONG_TIME *	设置芯片启动时是否需要延迟1s再执行后半段启动代码。 ◦ 0：不延迟

宏	描述
	◦ 1：延迟1秒
BOOT_CHECK_IMAGE	在XIP模式中，冷启动时是否对Image进行校验。 ◦ 0：不校验 ◦ 1：校验
IO_LDO_USE_3P3_V	配置是否开启LDO 3.3 V。 ◦ 0：不开启 ◦ 1：开启
SECURITY_CFG_VAL	配置算法安全等级。 ◦ 0：安全等级1 ◦ 1：安全等级2
CHIP_VER	固件使用的芯片版本。
CFG_CONTROLLER_ONLY	是否仅使用Bluetooth LE Controller。 ◦ 0：使用Bluetooth LE Controller和Host ◦ 1：仅使用Bluetooth Controller
CFG_MAX_PRFS	Application所能够包含的GATT Profile/Service的最大数量。开发者根据实际需求设置该值，其值越大占用的RAM空间就越大。
CFG_MAX_BOND_DEVS	Application支持的最大绑定设备数，最大值为4。
CFG_MAX_CONNECTIONS	Application支持的最大连接数量，最大值为10。开发者可以根据实际需要设置其值。该值越大，Bluetooth LE Stack Heaps需要占用的RAM空间就越大。Bluetooth LE Stack Heaps具体大小由*flash_scatter_config.h*中的以下四个宏定义，开发者不可修改这四个宏： ◦ ENV_HEAP_SIZE ◦ ATT_DB_HEAP_SIZE ◦ KE_MSG_HEAP_SIZE ◦ NON_RET_HEAP_SIZE
CFG_MAX_ADVS	Application支持的Bluetooth LE传统广播和扩展广播总量的最大值。
CFG_MAX_SCAN	是否支持扫描功能。 ◦ 0：不支持 ◦ 1：支持
CFG_MUL_LINK_WITH_SAME_DEV	是否支持同一设备的多个链接。 ◦ 0：不支持 ◦ 1：支持
CFG_BT_BREDR	是否支持LE链路生成Bluetooth Classic Link Key。 ◦ 0：不支持 ◦ 1：支持
CFG_CAR_KEY_SUPPORT	是否支持车钥匙功能。

宏	描述
	◦ 0：不支持 ◦ 1：支持
CFG_MASTER_SUPPORT	是否支Master功能。 ◦ 0：不支持 ◦ 1：支持
CFG_SLAVE_SUPPORT	是否支持Slave功能。 ◦ 0：不支持 ◦ 1：支持
CFG_LEGACY_PAIR_SUPPORT	是否支持Legacy配对功能。 ◦ 0：不支持 ◦ 1：支持
CFG_SC_PAIR_SUPPORT	是否支持SC配对功能。 ◦ 0：不支持 ◦ 1：支持
CFG_COC_SUPPORT	是否支持COC功能。 ◦ 0：不支持 ◦ 1：支持
CFG_GATTS_SUPPORT	是否支持GATT Service模块。 ◦ 0：不支持 ◦ 1：支持
CFG_GATTC_SUPPORT	是否支持GATT Client模块。 ◦ 0：不支持 ◦ 1：支持
CFG_CONN_AOA_AOD_SUPPORT	是否支持连接方式的AoA/AoD功能。 ◦ 0：不支持 ◦ 1：支持
CFG_CONNLESS_AOA_AOD_SUPPORT	是否支持无连接的AoA/AoD功能。 ◦ 0：不支持 ◦ 1：支持
CFG_RANGING_SUPPORT	是否支持Ranging功能。 ◦ 0：不支持 ◦ 1：支持

*：上表中带*的宏可用于初始化BUILD_IN_APP_INFO结构体，该结构体被定义在固件的0x200地址处，使用*custom_config.h*中的宏进行初始化。系统启动时，Bootloader程序会从该地址读取固件的配置信息，作为启动参数。

- 使用Wizard配置参数

  *custom_config.h*文件中的注释符合Keil的Configuration Wizard Annotations规范。因此，开发者可在Keil中打开*custom_config.h*文件，利用图形化的Keil “Configuration Wizard”界面配置Application工程参数。

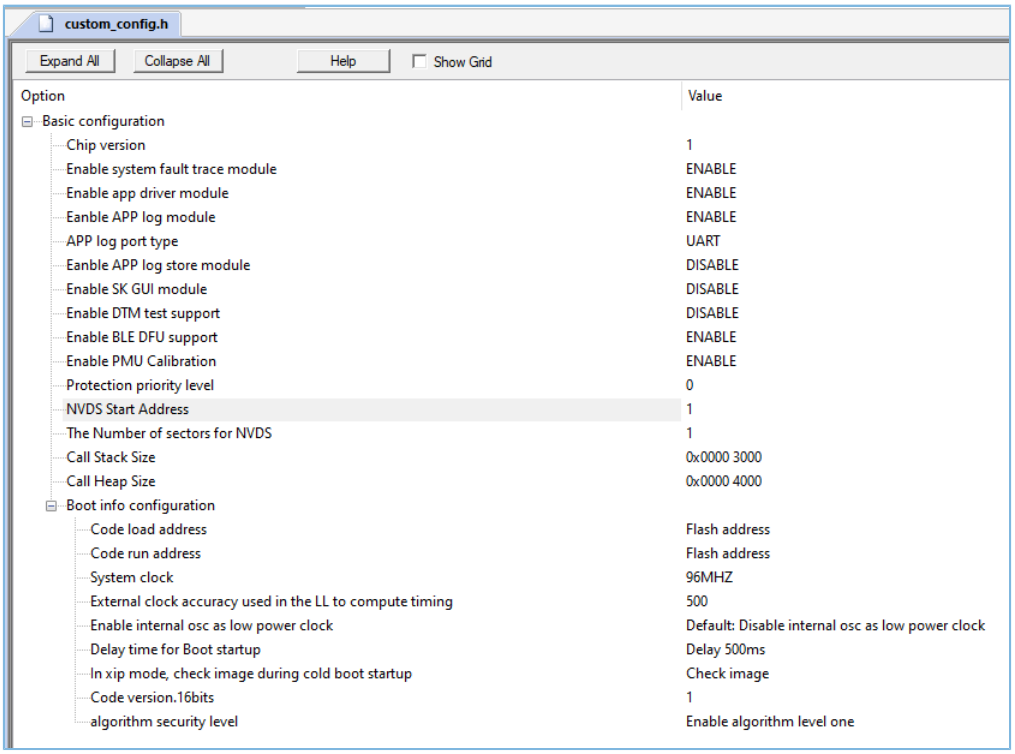

图 4-7 Configuration Wizard for custom_config.h

4.3.2.2 配置存储器布局

在Keil工程中，Scatter（*.sct*）文件描述链接器使用的存储区域。GR5525 SDK提供了一个Scatter示例文件（`SDK_Folder\platform\soc\linker\keil\flash_scatter_common.sct`），可帮助开发者快速完成存储器布局配置。另外，*flash_scatter_common.sct*使用的宏定义在*flash_scatter_config.h*中。

说明:

在Keil中，`__attribute__((section("name")))`可用于将一个函数或变量定义在特定的内存段中，其中“name”由开发者自定义。Scatter(*.sct*)文件可用于将自定义字段定义在特定位置。例如，将应用程序的ZI（零初始化）数据定义在名称为“.bss.app”的内存段中，则可设置“attribute”为“`__attribute__((section(".bss.app")))`”。

开发者可按照以下步骤配置存储器布局：

1. 点击Keil工具栏中的“Options for Target”按钮，打开“Options for Target ‘GRxx_Soc’ ”对话框，再选中“Linker”标签页。

2. 在“Linker”页面中的“Scatter File”栏，点击按钮“...”，浏览选择`SDK_Folder\platform\soc\linker\keil`下的*flash_scatter_common.sct*文件。开发者还可先将Scatter（*.sct*）和配置文件（*.h*）拷贝至ble_app_example工程的对应目录，再浏览选择文件。

说明:

*flash_scatter_common.sct*中的`#! armcc -E -I`语句指定了*flash_scatter_common.sct*依赖的头文件的目录。若该路径错误，则会产生Linker Error。

3. 点击“Edit...”按钮，打开*.sct*文件，然后根据实际的产品存储器布局，修改相应代码。

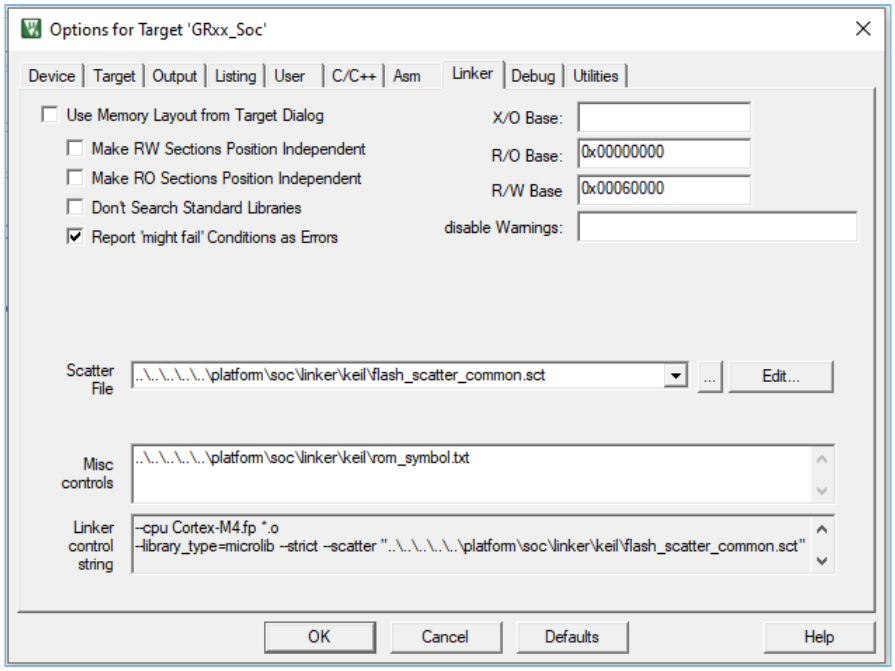

图 4-8 配置Scatter File

4. 点击“OK”按钮，保存设置。

4.3.2.3 配置After Build

在Keil中，“After Build”可用于指定工程Build完成后执行的命令行语句。

ble_app_template工程已默认配置After Build。因此，基于Template工程创建的新工程（ble_app_example）可无需手动配置。

若开发者直接通过Keil的“新建工程”创建一个工程，则需要按照以下步骤配置“After Build”：

1. 点击Keil Toolbar的“Options for Target”按钮，打开“Options for Target ‘GRxx_Soc’”对话框，选择“User”标签页。

2. 在“After Build/Rebuild”展开的选项中勾选“Run #1”，并在对应的“User Command”栏输入“fromelf.exe --text -c --output Listings\@L.s Objects\@L.axf”，以便调用Keil fromelf工具，基于axf文件生成汇编文件。

3. 在“After Build/Rebuild”展开的选项中勾选“Run #2”，并在对应的“User Command”栏输入“fromelf.exe --bin --output Listings\@L.bin Objects\@L.axf”，以便调用Keil fromelf工具，基于axf文件生成汇编文件。

4. 点击“OK”，保存设置。

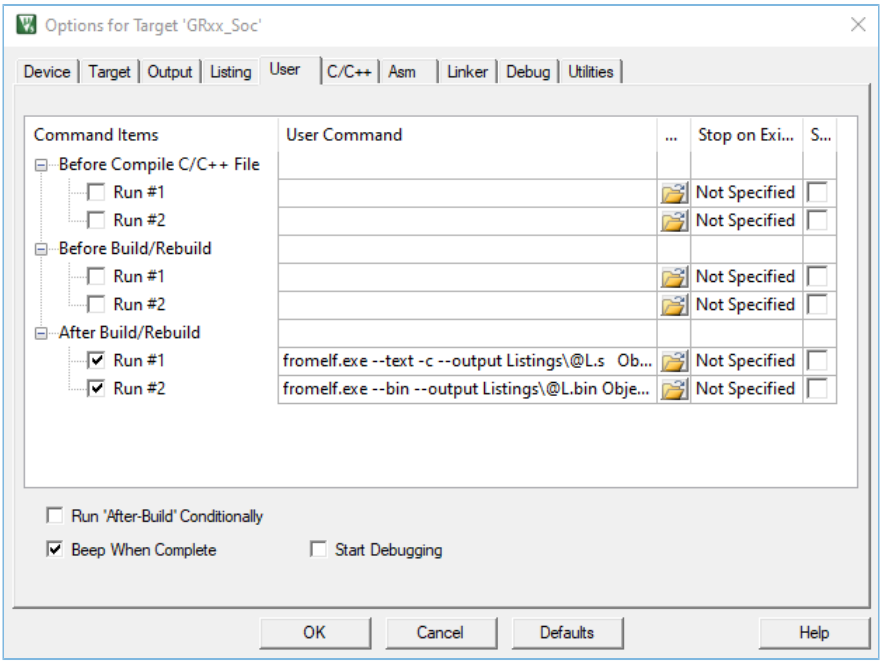

图 4-9 配置After Build

4.3.3 添加用户代码

开发者可根据实际应用需求，修改ble_app_example中相应代码。

4.3.3.1 修改主函数

以下为典型的*main.c*文件的内容。

```
/**@brief Stack global variables for Bluetooth protocol stack. */
STACK_HEAP_INIT(heaps_table);
...
int main (void)
{
    /** Initialize user peripherals. */
    app_periph_init();

    /** Initialize BLE Stack. */
    ble_stack_init(&&m_app_ble_callback, &heaps_table);

    // Main Loop
    while (1)
    {
        app_log_flush();
        pwr_mgmt_schedule();
    }
}
```

- STACK_HEAP_INIT(heaps_table)定义了7个全局数组，供Bluetooth LE协议栈作为Heap使用。开发者不可修改此定义，否则Bluetooth LE协议栈可能无法正常运行。Heap的大小与4.3.2.1 配置custom_config.h中的Bluetooth LE业务量有关。
- app_periph_init()用于初始化外设。在开发调试阶段，该函数中的SYS_SET_BD_ADDR可用于设置临时的Public Address，pwr_mgmt_mode_set()中可设置在自动功耗管理时MCU的工作模式（SLEEP/ IDLE/ ACTIVE）。app_periph_init()函数实现位于*user_periph_setup.c*文件，其示例代码如下所示：

```
/**@brief Bluetooth device address. */
static const uint8_t s_bd_addr[SYS_BD_ADDR_LEN] = {0x11, 0x11, 0x11, 0x11,0x11, 0x11};
...
void app_periph_init(void)
{
    SYS_SET_BD_ADDR(s_bd_addr);
    bsp_log_init();
    pwr_mgmt_mode_set(PMR_MGMT_IDLE_MODE);
}
```

- 在while(1) { }中添加Application的Main Loop代码，比如处理外部输入、更新GUI。
- 若需打开APP LOG模块，则应在Main Loop中调用`app_log_flush()`，以保证系统进入Sleep状态之前，完整输出所有Log。关于 APP LOG模块使用，参考4.6.3 输出调试Log。
- 调用pwr_mgmt_shcedule()实现自动功耗管理，以降低系统功耗。

4.3.3.2 实现Bluetooth LE业务逻辑

Application的Bluetooth LE业务逻辑由GR5525 SDK中定义的若干Bluetooth LE Events进行驱动。因此，Application需要实现相应的Event Handler，以获取Bluetooth LE Stack的运行结果或状态变更通知。由于Event Handler在Bluetooth LE SDK IRQ的中断上下文（Interrupt Context）中被调用，因此开发者不能在Handler中执行比较耗时的操作，例如阻塞式函数调用、无限循环等。否则，可能阻塞整个系统运行，导致Bluetooth LE Stack与SDK Bluetooth LE模块无法按照正常时序运行。

Bluetooth LE Events按照Common、GAP Management 、GAP Connection Control、Security Manager、L2CAP、GATT Common、GATT Server和GATT Client分类。GR5525 SDK支持的Bluetooth LE Events如下表所示。

表 4-5 Bluetooth LE Events

Event类别	Event名称	描述
Common	BLE_COMMON_EVT_STACK_INIT	Bluetooth LE Stack init complete event.
GAP Management	BLE_GAPM_EVT_CH_MAP_SET	Channel Map Set complete event.
	BLE_GAPM_EVT_WHITELIST_SET	Whitelist Set complete event.
	BLE_GAPM_EVT_PER_ADV_LIST_SET	Periodic Advertising List Set complete event.

Event类别	Event名称	描述
	BLE_GAPM_EVT_PRIVACY_MODE_SET	Privacy Mode for Peer Device Set complete event.
	BLE_GAPM_EVT_LEPSM_REGISTER	LEPSM Register complete event.
	BLE_GAPM_EVT_LEPSM_UNREGISTER	LEPSM Unregister complete event.
	BLE_GAPM_EVT_DEV_INFO_GOT	Device Info Get event.
	BLE_GAPM_EVT_ADV_START	Advertising Start complete event.
	BLE_GAPM_EVT_ADV_STOP	Advertising Stop complete event.
	BLE_GAPM_EVT_SCAN_REQUEST	Scan Request event.
	BLE_GAPM_EVT_ADV_DATA_UPDATE	Advertising Data update event.
	BLE_GAPM_EVT_SCAN_START	Scan Start complete event.
	BLE_GAPM_EVT_SCAN_STOP	Scan Stop complete event.
	BLE_GAPM_EVT_ADV_REPORT	Advertising Report event.
	BLE_GAPM_EVT_SYNC_ESTABLISH	Periodic Advertising Synchronization Establish event.
	BLE_GAPM_EVT_SYNC_STOP	Periodic Advertising Synchronization Stop event.
	BLE_GAPM_EVT_SYNC_LOST	Periodic Advertising Synchronization Lost event.
	BLE_GAPM_EVT_READ_RSLV_ADDR	Read Resolvable Address event.
GAP Connection Control	BLE_GAPC_EVT_PHY_UPDATED	PHY Update event.
	BLE_GAPC_EVT_CONNECTED	Connected event.
	BLE_GAPC_EVT_DISCONNECTED	Disconnected event.
	BLE_GAPC_EVT_CONNECT_CANCEL	Connect Cancel event.
	BLE_GAPC_EVT_AUTO_CONN_TIMEOUT	Auto Connect Timeout event.
	BLE_GAPC_EVT_CONN_PARAM_UPDATED	Connect Parameter Updated event.
	BLE_GAPC_EVT_CONN_PARAM_UPDATE_REQ	Connect Parameter Request event.
	BLE_GAPC_EVT_PEER_NAME_GOT	Peer Name Get event.
	BLE_GAPC_EVT_CONN_INFO_GOT	Connect Info Get event.
	BLE_GAPC_EVT_PEER_INFO_GOT	Peer Info Get event.
	BLE_GAPC_EVT_DATA_LENGTH_UPDATED	Data Length Updated event.
	BLE_GAPC_EVT_DEV_INFO_SET	Device Info Set event.
	BLE_GAPC_EVT_CONNECT_IQ_REPORT	Connection IQ Report info event.

Event类别	Event名称	描述
	BLE_GAPC_EVT_CONNECTLESS_IQ_REPORT	Connectionless IQ Report info event.
	BLE_GAPC_EVT_LOCAL_TX_POWER_READ	Local transmit power read indication info event.
	BLE_GAPC_EVT_REMOTE_TX_POWER_READ	Remote transmit power read indication info event.
	BLE_GAPC_EVT_TX_POWER_CHANGE_REPORT	Transmit power change reporting info event.
	BLE_GAPC_EVT_PATH_LOSS_THRESHOLD_REPORT	Path loss threshold reporting info event.
	BLE_GAPC_EVT_RANGING_IND	Ranging indication event.
	BLE_GAPC_EVT_RANGING_SAMPLE_REPORT	Ranging sample report event.
	BLE_GAPC_EVT_RANGING_CMP_IND	Ranging complete indication event.
	BLE_GAPC_EVT_DFT_SUBRATE_SET	Default subrate param set complete event.
	BLE_GAPC_EVT_SUBRATE_CHANGE_IND	Subrate change indication event.
GATT Common	BLE_GATT_COMMON_EVT_MTU_EXCHANGE	MTU Exchange event.
	BLE_GATT_COMMON_EVT_PRF_REGISTER	Service Register event.
GATT Server	BLE_GATTS_EVT_READ_REQUEST	GATTS Read Request event.
	BLE_GATTS_EVT_WRITE_REQUEST	GATTS Write Request event.
	BLE_GATTS_EVT_PREP_WRITE_REQUEST	GATTS Prepare Write Request event.
	BLE_GATTS_EVT_NTF_IND	GATTS Notify or Indicate Complete event.
	BLE_GATTS_EVT_CCCD_RECOVERY	GATTS CCCD Recovery event.
	BLE_GATTS_EVT_MULT_NTF	GATTS Multiple Notifications event.
	BLE_GATTS_EVT_ENH_READ_REQUEST	GATTS Enhanced Read Request event.
	BLE_GATTS_EVT_ENH_WRITE_REQUEST	GATTS Enhanced Write Request event.
	BLE_GATTS_EVT_ENH_PREP_WRITE_REQUEST	GATTS Enhanced Prepare Write Request event.
	BLE_GATTS_EVT_ENH_NTF_IND	GATTS Enhanced Notify or Indicate Complete event.

Event类别	Event名称	描述
	BLE_GATTS_EVT_ENH_CCCD_RECOVERY	GATTS Enhanced CCCD Recovery event.
	BLE_GATTS_EVT_ENH_MULT_NTF	GATTS Enhanced Multiple Notifications event.
GATT Client	BLE_GATTC_EVT_SRVC_BROWSE	GATTC Service Browse event.
	BLE_GATTC_EVT_PRIMARY_SRVC_DISC	GATTC Primary Service Discovery event.
	BLE_GATTC_EVT_INCLUDE_SRVC_DISC	GATTC Include Service Discovery event.
	BLE_GATTC_EVT_CHAR_DISC	GATTC Characteristic Discovery event.
	BLE_GATTC_EVT_CHAR_DESC_DISC	GATTC Characteristic Descriptor Discovery event.
	BLE_GATTC_EVT_READ_RSP	GATTC Read Response event.
	BLE_GATTC_EVT_WRITE_RSP	GATTC Write Response event.
	BLE_GATTC_EVT_NTF_IND	GATTC Notify or Indicate Receive event.
	BLE_GATTC_EVT_CACHE_UPDATE	GATTC Cache Update event.
	BLE_GATTC_EVT_ENH_SRVC_BROWSE	GATTC Enhanced Service Browse event.
	BLE_GATTC_EVT_ENH_PRIMARY_SRVC_DISC	GATTC Enhanced Primary Service Discovery event.
	BLE_GATTC_EVT_ENH_INCLUDE_SRVC_DISC	GATTC Enhanced Include Service Discovery event.
	BLE_GATTC_EVT_ENH_CHAR_DISC	GATTC Enhanced Characteristic Discovery event.
	BLE_GATTC_EVT_ENH_CHAR_DESC_DISC	GATTC Enhanced Characteristic Descriptor Discovery event.
	BLE_GATTC_EVT_ENH_READ_RSP	GATTC Enhanced Read Response event.
	BLE_GATTC_EVT_ENH_WRITE_RSP	GATTC Enhanced Write Response event.
	BLE_GATTC_EVT_ENH_NTF_IND	GATTC Enhanced Notify or Indicate Receive event.
Security Manager	BLE_SEC_EVT_LINK_ENC_REQUEST	Link Encrypted Request event.
	BLE_SEC_EVT_LINK_ENCRYPTED	Link Encrypted event.

Event类别	Event名称	描述
	BLE_SEC_EVT_KEY_PRESS_NTF	Key Press event.
	BLE_SEC_EVT_KEY_MISSING	Key Missing event.
L2CAP	BLE_L2CAP_EVT_CONN_REQ	L2cap Connect Request event.
	BLE_L2CAP_EVT_CONN_IND	L2cap Connected Indicate event.
	BLE_L2CAP_EVT_ADD_CREDITS_IND	L2cap Credits Add Indicate event.
	BLE_L2CAP_EVT_DISCONNECTED	L2cap Disconnected event.
	BLE_L2CAP_EVT_SDU_RECV	L2cap SDU Receive event.
	BLE_L2CAP_EVT_SDU_SEND	L2cap SDU Send event.
	BLE_L2CAP_EVT_ADD_CREDITS_CPLT	L2cap Credits Add Completed event.
	BLE_L2CAP_EVT_ENH_CONN_REQ	L2cap Enhanced Connect Request event.
	BLE_L2CAP_EVT_ENH_CONN_IND	L2cap Enhanced Connected Indicate event.
	BLE_L2CAP_EVT_ENH_RECONFIG_CPLT	L2cap Enhanced Reconfig Completed event.
	BLE_L2CAP_EVT_ENH_RECONFIG_IND	L2cap Enhanced Reconfig Indicate event.

开发者需根据产品的功能需求，实现所需的Bluetooth LE Event Handler。例如，若产品不支持Security Manager，则可无需实现对应的Event；若产品只支持GATT Server而不支持GATT Client，则可无需实现GATT Client对应的Event。并且，对于每一类Event的Event Handler也并非需全部实现，仅需实现产品所必须的Event Handler即可。

提示:

关于Bluetooth LE API和Event API的使用方法，请参考`SDK_Folder\documentation\GR5525_API_Reference`以及`SDK_Folder\projects\ble`中的Bluetooth LE示例源代码。

4.3.3.3 BLE_Stack_IRQ、BLE_SDK_IRQ与Application的调度机制

Bluetooth LE Stack是低功耗蓝牙协议实现核心，可直接操作Bluetooth 5.3 Core硬件（参考2.2 软件架构）。因此，BLE_Stack_IRQ具有整个系统中次高的优先级（SVCall IRQ具有最高优先级），以保证Bluetooth LE Stack严格按照Bluetooth Core Spec规定的时序运行。

Bluetooth LE Stack的状态改变会触发优先级较低的BLE_SDK_IRQ中断。在该中断处理函数，可通过调用Application实现的Bluetooth LE Event Handler，将Bluetooth LE Stack的状态变更通知以及相关的业务数据发送至Application。在这些Event Handler中，应避免操作耗时的业务，而应将耗时业务转移至Main Loop或用户级线程中处理。开发者可使用`SDK_Folder\components\libraries\app_queue`模块（或自定义的Application Framework）从Bluetooth LE Event Handler向Main Loop传递事件。

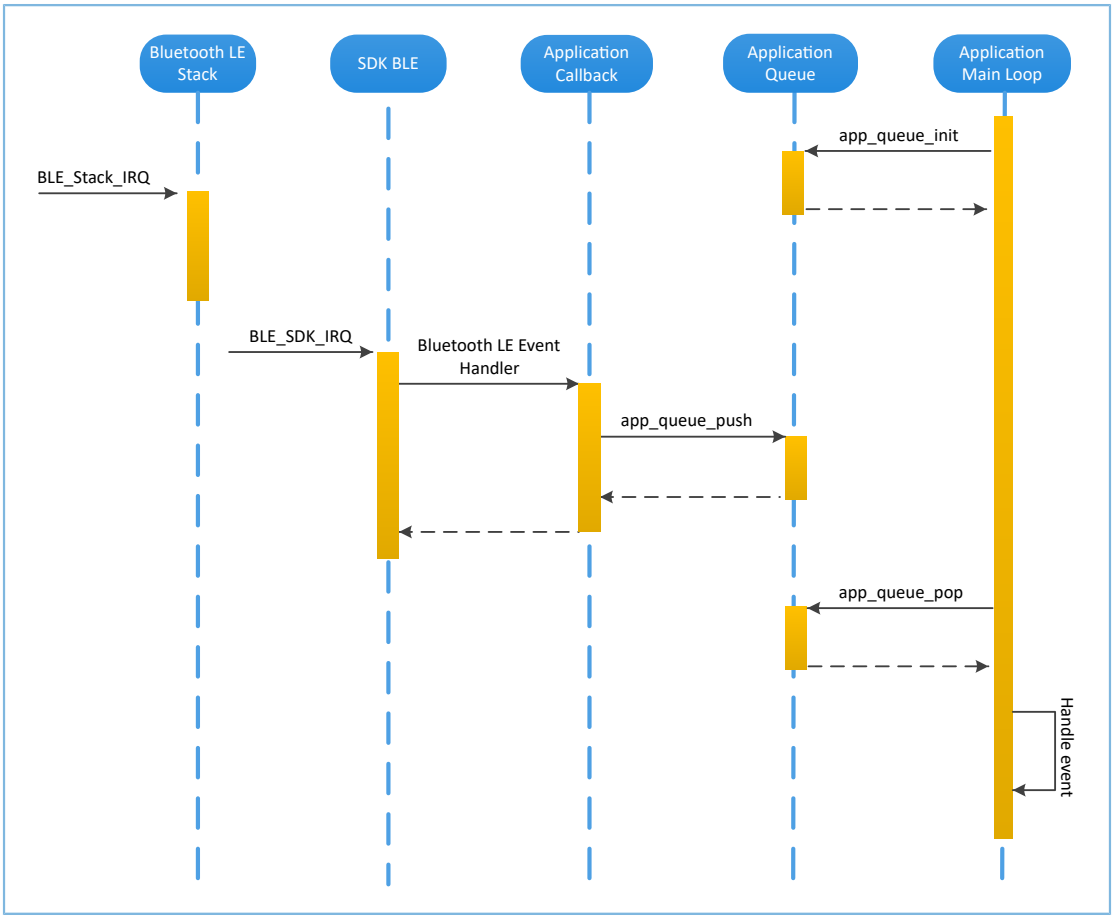

图 4-10 Non-OS system schedule

4.4 生成固件

Bluetooth LE Application创建完成以后，可直接点击Keil工具栏中的“Build” 按钮构建工程。工程编译成功后，将在工程目录下的`Keil_5\Listings`和`Keil_5\Objects`文件夹下分别生成.bin和.hex格式的固件文件。

表 4-6 生成的固件

名称	描述
ble_app_example.bin	二进制应用固件，可通过GProgrammer下载至芯片Flash中运行。
ble_app_example.hex	二进制应用固件，可通过Keil或GProgrammer下载至芯片Flash中运行。

提示:

上述两种格式的固件，均可使用GProgrammer下载至芯片Flash中运行。具体操作，可参考《GProgrammer用户手册》。

4.5 下载.hex文件至Flash

固件生成后，可按照以下步骤将固件下载至Flash中：

1. 配置Keil Flash编程算法。

(1) 拷贝SDK_Folder\build\binaries\download_algorithm\Keil\GR5xxx_16MB_Flash.FLM文件至Keil_Folder\ARM\Flash目录。

(2) 点击Keil工具栏中的“Options for Target”按钮，打开“Options for Target ‘GRxx_Soc’ ”对话框，选择“Debug”标签页；点击“Use: J-LINK/J-TRACE Cortex”右侧的“Settings”按钮。

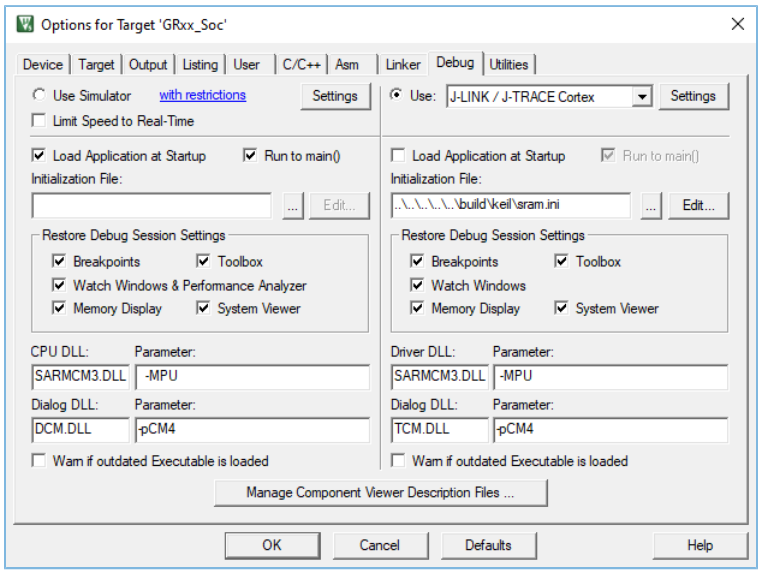

图 4-11 Debug标签页

(3) 在打开的“Cortex JLink/JTrace Target Driver Setup”窗口中，选中“Flash Download”项。在“Download Function”区域，开发者可以设置Erase方式、选择是否“Program”、“Verify”、“Reset and Run”。Keil默认配置如下：

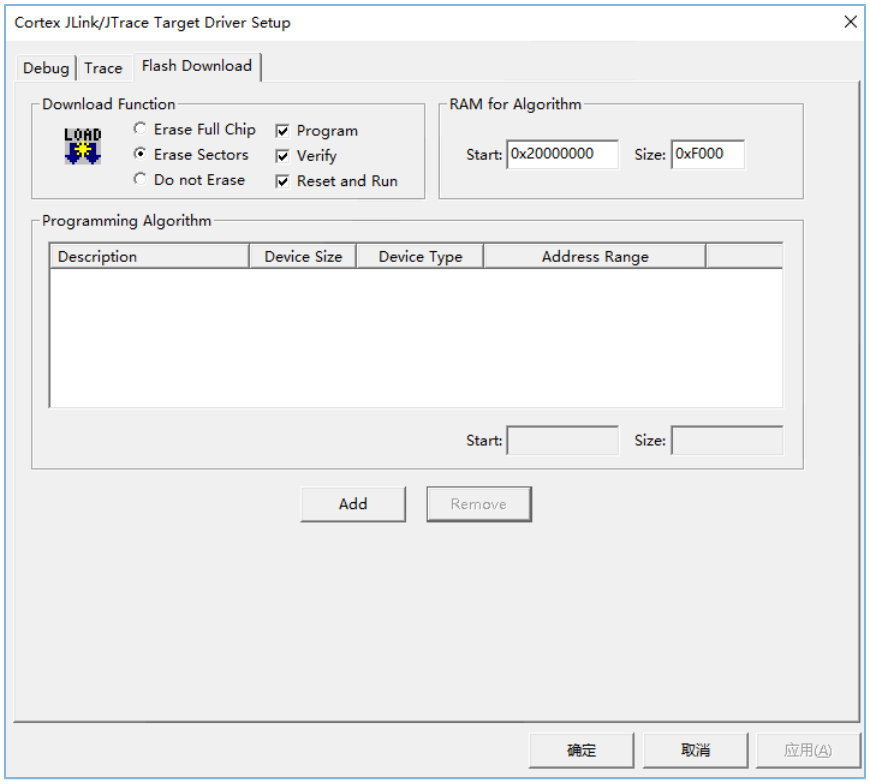

图 4-12 选择“Download Function”

(4) 点击“Add”按钮，在“Programming Algorithm”中添加*GR5xxx_ 16MB_Flash.FLM*（位于SDK_Folder\build\keil\）。

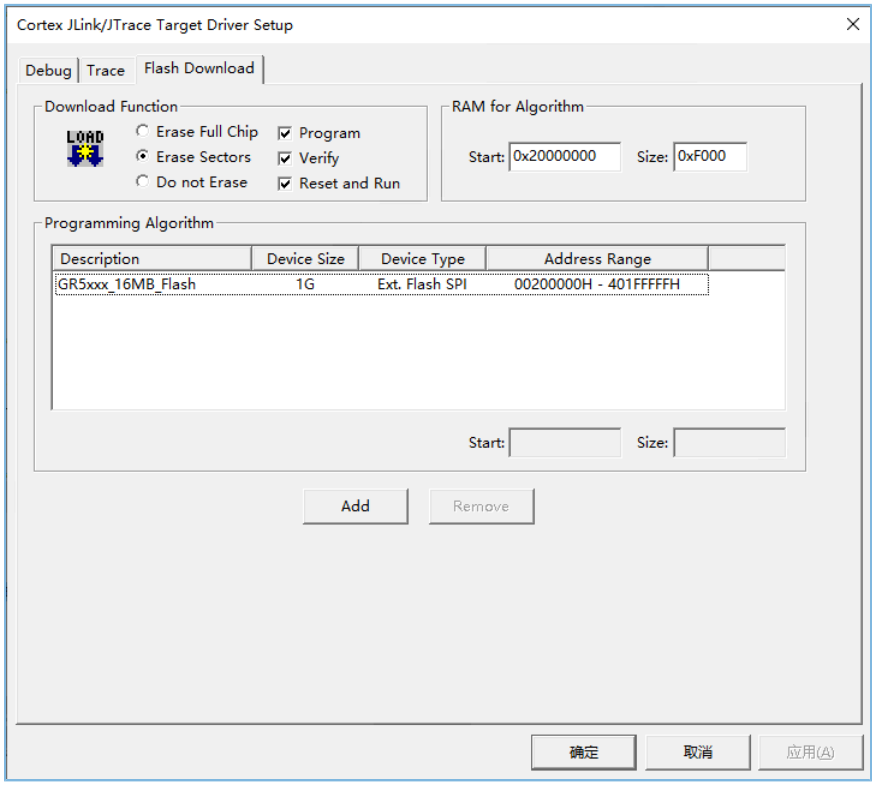

图 4-13 添加*GR5xxx_16MB_Flash.FLM*编程算法

(5) 配置“RAM for Algorithm”，以定义加载和执行编程算法的地址空间。“Start”的值应为GR5525中RAM的起始地址“0x20000000”，“Size”的值为“0xF000”。

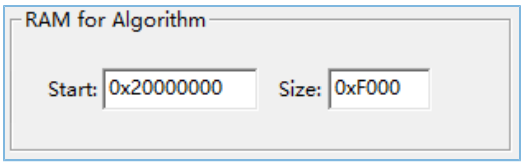

图 4-14 RAM for Algorithm设置

(6) 点击“OK”，保存设置。

2. 下载固件。

配置完成以后，点击Keil工具栏中的“Download”按钮 将*ble_app_example.axf*文件下载至芯片Flash中。如果固件下载成功，Keil的“Build Output”窗口将显示如下结果。

说明:

下载过程中，若界面提示“No Cortex-M SW Device Found”，则表示芯片当前可能处于睡眠状态（即开启睡眠模式的工程正在运行），无法直接下载.hex文件到Flash中。开发者需先按下GR5525 SK板的“RESET”键，间隔1秒左右，再点击“Download”按钮，重新下载文件。

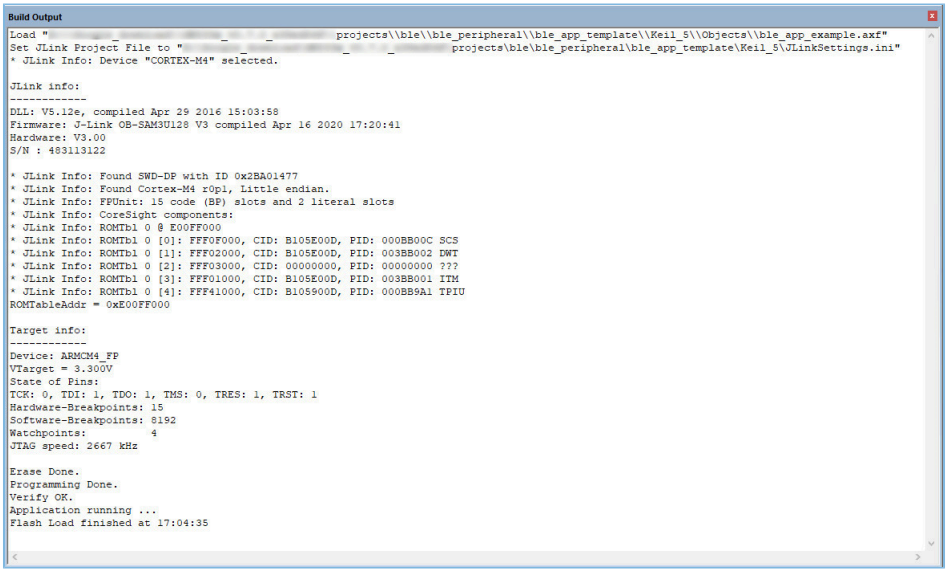

图 4-15 下载结果

4.6 调试

Keil提供了调试器，支持代码在线调试。该调试器支持设置6个硬件断点和多个软件断点。开发者还可以使用多种方式设置调试命令。

4.6.1 配置调试器

启动调试之前，需要配置调试器。点击Keil工具栏中的“Options for Target”按钮，打开“Options for Target ‘GRxx_Soc’”对话框，选择“Debug”标签页。窗口左侧为软件仿真调试配置，右侧为硬件在线调试配置。Bluetooth LE Examples工程使用硬件在线调试，相关的调试器默认配置如下：

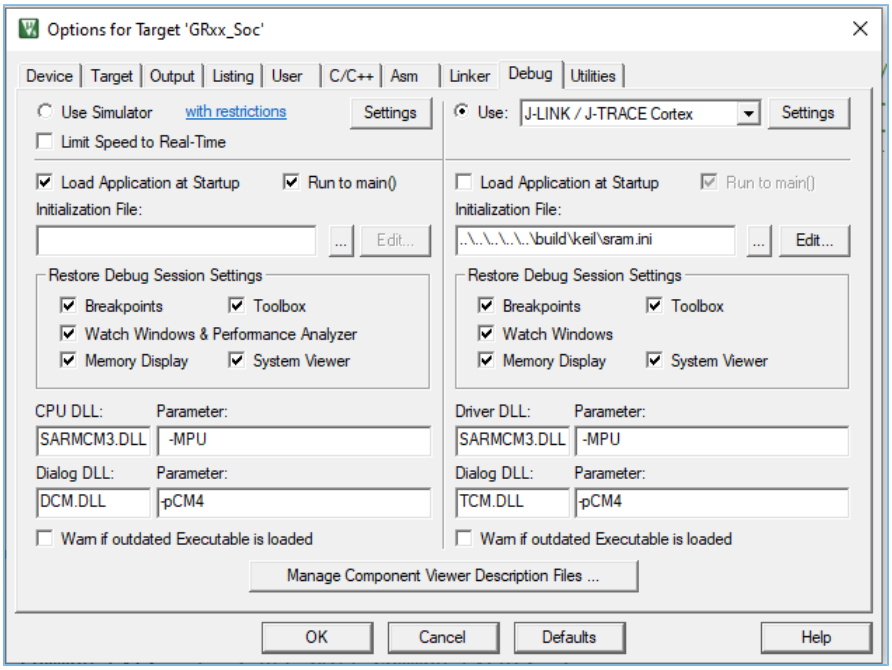

图 4-16 配置调试器

默认使用的Initialization File：*sram.ini*位于SDK_Folder\build\keil目录。开发者可以直接使用该文件，也可以将该文件复制到自己的工程目录下使用。

初始化文件*sram.ini*中包含一组调试命令，调试过程将执行这些命令。点击“Initialization File”栏右侧的“Edit...”按钮，可打开*sram.ini*文件。*sram.ini*的代码示例如下：

```
/**
*************************************************************
* GR55xx object loading script through debugger interface
* (e.g.Jlink, *etc).
* The goal of this script is to load the Keils's object file to the
* GR55xx RAM
* assuring that the GR55xx has been previously cleaned up.
*************************************************************
*/
// Debugger reset(check Keil debugger settings)
// Preselected reset type(found in Options->Debug->Settings)is
// Normal(0);
// -Normal:Reset core & peripherals via SYSRESETREQ & VECTRESET bit
// RESET
// Load object file
LOAD %L
// Load stack pointer
SP = _RDWORD(0x00000000)
```

```
// Load program counter
$ = _RDWORD(0x00000004)
// Write 0 to vector table register,  remap vector
_WDWORD(0xE000ED08,  0x00000000)
```

说明:

Keil支持按照以下顺序执行开发者设置的调试器命令：

1. 当“Options for Target ‘GRxx_Soc’ > Debug > Load Application at Startup”被使能，调试器会首先载入“Options for Target ‘GRxx_Soc’ > Output > Name of Executable”中的文件。
2. 执行“Options for Target ‘GRxx_Soc’ > Debug > Initialization File”所指定文件中的命令。
3. 当“Options for Target ‘GRxx_Soc’ > Debug > Restore Debug Session Settings”包含的选项被选中，恢复相应的Breakpoints，Watch Windows，Memory Display等。
4. 当“Options for Target ‘GRxx_Soc’ > Debug > Run to main()”被选中或者命令`g,main`被发现位于Initialization File中，调试器就开始自动执行CPU指令，直至遇到main()才会停下来。

4.6.2 启动调试

完成调试器配置后，点击Keil工具栏的“Start/Stop Debug Session”按钮 即可开始调试。

说明:

“Connect & Reset Options”均需设置为“Normal”，如图 4-17所示，以保证启动“Debug Session”之后，点击Keil工具栏的“Reset”按钮，程序仍能正常运行。

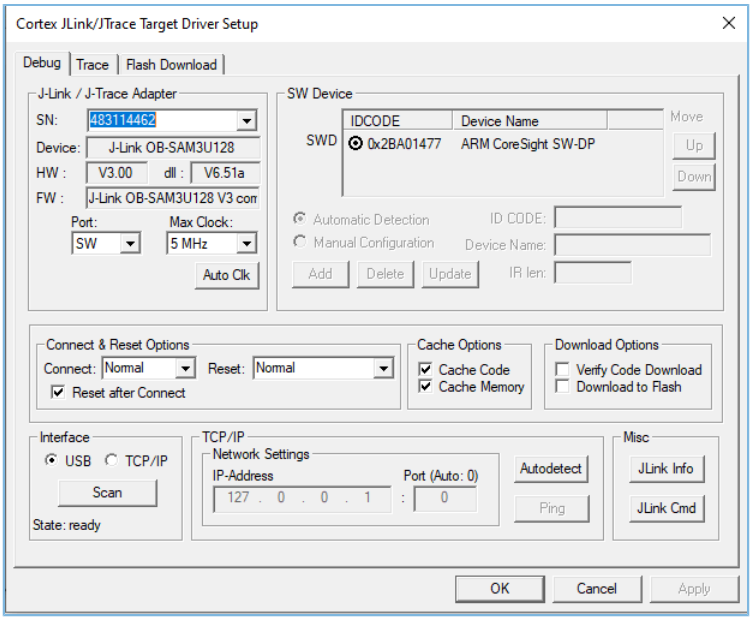

图 4-17 配置“Connect & Reset Options”均为“Normal”

4.6.3 输出调试Log

GR5525 SDK提供APP LOG模块，可支持硬件端口输出Application调试Log，并支持开发者自定义输出方式：UART、J-Link RTT或ARM ITM（Instrumentation Trace Macrocell）。若使用APP LOG模块，则需在*custom_config.h*中打开宏APP_LOG_ENABLE，并根据所需输出方式配置宏APP_LOG_PORT。

4.6.3.1 模块初始化

完成配置后，开发者还需在外设初始化阶段，调用app_log_init()完成APP LOG模块的初始化，包括配置Log参数、注册Log输出接口和Flush接口。

APP LOG模块支持使用标准C库函数printf()和APP LOG API输出调试Log。若使用APP LOG API，则可通过设置Log级别、格式、过滤方式等参数优化Log输出；若使用printf()，则可将Log参数设置为NULL。

根据开发者所设置的输出方式，调用相应模块的初始化函数（具体可参考SDK_Folder\platform\boards\board_SK.h），并注册相应的发送和Flush函数，可参考函数bsp_log_init()。以UART输出方式为例，bsp_log_init()实现如下：

```
void bsp_log_init(void)
{
#if APP_LOG_ENABLE

#if (APP_LOG_PORT == 0)
    bsp_uart_init();
#elif (APP_LOG_PORT == 1)
    SEGGER_RTT_ConfigUpBuffer(0, NULL, NULL, 0, SEGGER_RTT_MODE_BLOCK_IF_FIFO_FULL);
#endif

#if (APP_LOG_PORT <= 2)
    app_log_init_t   log_init;

    log_init.filter.level                  = APP_LOG_LVL_DEBUG;
    log_init.fmt_set[APP_LOG_LVL_ERROR]   = APP_LOG_FMT_ALL & (~APP_LOG_FMT_TAG);
    log_init.fmt_set[APP_LOG_LVL_WARNING] = APP_LOG_FMT_LVL;
    log_init.fmt_set[APP_LOG_LVL_INFO]    = APP_LOG_FMT_LVL;
    log_init.fmt_set[APP_LOG_LVL_DEBUG]   = APP_LOG_FMT_LVL;

#if (APP_LOG_PORT == 0)
    app_log_init(&log_init, bsp_uart_send, bsp_uart_flush);
#elif (APP_LOG_PORT == 1)
    app_log_init(&log_init, bsp_segger_rtt_send, NULL);
#elif (APP_LOG_PORT == 2)
    app_log_init(&log_init, bsp_itm_send, NULL);
#endif

#endif
    app_assert_init();
#endif
}
```

说明:

- app_log_init()接口的入参包括Log初始化参数、Log输出接口和Flush接口。其中，Flush接口可以选择不注册。
- GR5525 SDK提供了APP LOG STORE模块，该模块支持将调试Log存入Flash中以及从Flash中导出。使用该模块时需在*custom_config.h*中添加宏APP_LOG_STORE_ENABLE。`SDK_Folder\projects\ble\ble_peripheral\ble_app_rscs`工程中配置了该功能，开发者可参考该示例工程配置，使用APP LOG STORE模块。
- 使用printf()输出的Application Log无法使用APP LOG STORE模块进行存储。

当使用UART输出调试Log时，已实现的Log输出接口和Flush接口分别为bsp_uart_send()和bsp_uart_flush()。

- bsp_uart_send()实现了app_uart异步（app_uart_transmit_async接口）和hal_uart同步（hal_uart_transmit接口）两种方式的输出接口，开发者可根据实际需求，选择合适的Log输出方式。
- bsp_uart_flush()为uart_flush接口，用于中断模式下输出缓存在内存中缓存的Log数据。

说明:

开发者可重写上述两个接口函数。

当使用J-Link RTT或ARM ITM输出调试Log时，已实现的Log输出接口分别为bsp_segger_rtt_send() 和bsp_itm_send()。在这两种模式下，没有实现Flush接口。

4.6.3.2 使用方法

APP LOG模块初始化完成后，开发者可以使用以下四个API输出调试Log：

- APP_LOG_ERROR()
- APP_LOG_WARNING()
- APP_LOG_INFO()
- APP_LOG_DEBUG()

如果使用了中断输出模式，可调用app_log_flush()函数将缓存中的全部调试Log输出，从而保证芯片复位或系统睡眠之前，输出全部调试Log。

当选择ARMCC编译并使用J-Link RTT方式输出Log时，推荐在*SEGGER_RTT.c*中做如下修改：

SEGGER_RTT.c

```
238  *
239  *       Static data
240  *
241  **********************************************************************
242  */
243  //
244  // RTT Control Block and allocate buffers for channel 0
245  //
246  __attribute__((section (".ARM.__at_0x20005000"))) SEGGER_RTT_CB _SEGGER_RTT;
247  //SEGGER_RTT_PUT_CB_SECTION(SEGGER_RTT_CB_ALIGN(SEGGER_RTT_CB _SEGGER_RTT));
```

图 4-18 创建RTT Control Block并置于地址0x20005000处

若需对J-Link RTT Viewer进行配置，则可参考图 4-19中的参数配置。

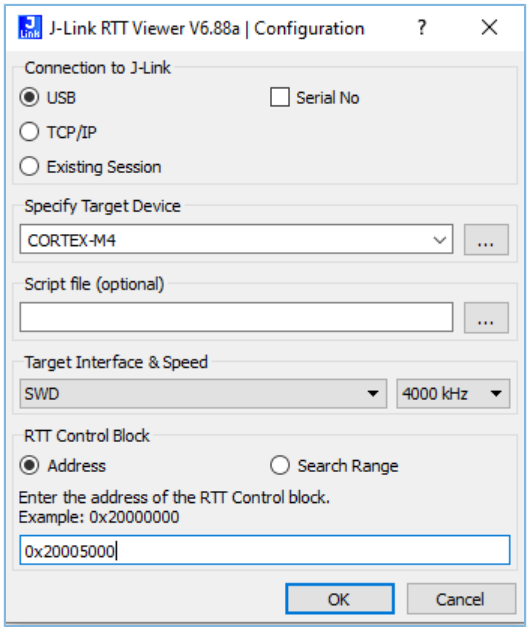

图 4-19 配置J-Link RTT Viewer

其中“RTT Control Block”的地址通过“Address”方式指定，输入值可设置为编译工程生成的.map映射文件中“_SEGGER_RTT”结构体的地址，如下图所示。若按图 4-18推荐方式创建RTT Control Block并将其置于地址0x20005000处，则“Address”应设置为“0x20005000”。

```
ultra_wfi_or_wfe          0x200037ec   Data          0  rom_symbol.txt ABSOLUTE
sdk_gap_env               0x200038ec   Data          0  rom_symbol.txt ABSOLUTE
_SEGGER_RTT               0x20005000   Data        120  segger_rtt.o(.ARM.__at_0x20005000)
jlink_opt_info            0x20006000   Data          0  rom_symbol.txt ABSOLUTE
SystemCoreClock           0x2000b000   Data          4  system_gr55xx.o(.data)
__stdout                  0x2000b044   Data          4  app_log.o(.data)
```

图 4-20 获取RTT Control Block地址

说明:

当使用GCC编译时，无需按照图 4-18修改，J-Link RTT Viewer中RTT Control block地址为编译工程生成的.map映射文件中“_SEGGER_RTT”结构体的地址。

4.6.4 使用GRToolbox调试

GR5525 SDK提供GRToolbox App （Android版），可用于GR5525 Bluetooth LE应用调试。该App主要提供以下功能：

- 通用的Bluetooth LE扫描和连接，对Characteristics的读写。
- 标准Profile的Demo展示，包括Heart Rate、Blood Pressure等。
- Goodix自定义应用程序。

提示:

GRToolbox安装文件为`SDK_Folder\tools\GRToolbox\GRToolbox-Version.apk`。

---

5 术语与缩略语

表 5-1 术语与缩略语

名称	描述
AoA/AoD	Angle of Arrival/Angle of Departure，到达角/出发角
API	Application Programming Interface，应用程序编程接口
ATT	Attribute Protocol，属性协议层
Bluetooth LE	Bluetooth Low Energy，低功耗蓝牙
DAP	Debug Access Port，调试访问端口
DFU	Device Firmware Update，设备固件更新
GAP	Generic Access Profile，通用访问规范
GATT	Generic Attribute Profile，通用属性规范
GFSK	Gaussian Frequency Shift Keying，高斯频移键控
HAL	Hardware Abstract Layer，硬件抽象层
HCI	Host Controller Interface，主机控制器接口
IoT	Internet of Things，物联网
L2CAP	Logical Link Control and Adaptation Protocol，逻辑链路控制与适配协议
LL	Link Layer，链路层
NVDS	Non-volatile Data Storage，非易失性数据存储
OTA	Over The Air，用无线传输
PMU	Power Management Unit，电源管理单元
PHY	Physical Layer，物理层
RF	Radio Frequency，射频
SCA	System Configuration Area，系统配置区
SDK	Software Development Kit，软件开发工具套件
SM	Security Manager，安全管理器
SoC	System-on-Chip，系统级芯片
UART	Universal Asynchronous Receiver/Transmitter，异步收发传输器
XIP	Execute in Place，片上执行