GR5405 Developer Guide

Version: 1.1

Release Date: 2025-04-22

Shenzhen Goodix Technology Co., Ltd.

Trademarks and Permissions

GOODiX and other Goodix trademarks are trademarks of Shenzhen Goodix Technology Co., Ltd. All other trademarks and trade names mentioned in this document are the property of their respective holders.

Disclaimer

Information contained in this document is intended for your convenience only and is subject to change without prior notice. It is your responsibility to ensure its application complies with technical specifications.

Shenzhen Goodix Technology Co., Ltd. (hereafter referred to as “Goodix”) makes no representation or guarantee for this information, express or implied, oral or written, statutory or otherwise, including but not limited to representation or guarantee for its application, quality, performance, merchantability or fitness for a particular purpose. Goodix shall assume no responsibility for this information and relevant consequences arising out of the use of such information.

Without written consent of Goodix, it is prohibited to use Goodix products as critical components in any life support system. Under the protection of Goodix intellectual property rights, no license may be transferred implicitly or by any other means.

Shenzhen Goodix Technology Co., Ltd.

Headquarters: 26F, Goodix Headquarters, No.1 Meikang Road, Futian District, Shenzhen, China

TEL: +86-755-33338828 Zip Code: 518000

Website: www.goodix.com

Preface

Purpose

This document introduces the Software Development Kit (SDK) of the Goodix GR5405 automotive Bluetooth Low Energy (Bluetooth LE) System-on-Chip (SoC) and Keil/IAR for program development and debugging, to help you quickly get started with secondary development of automotive applications.

Audience

This document is intended for:

- Device user
- Developer
- Test engineer
- Technical support engineer

Release Notes

This document is the second release of *GR5405 Developer Guide*, corresponding to GR5405 SoCs.

Revision History

Version	Date	Description
1.0	2024-08-30	Initial release
1.1	2025-04-22	• Added Mesh-related descriptions. • Updated "Tools". • Revised "Configuring custom_config.h". • Revised the code example for bsp_log_int() and the notes about outputting debug logs through UART.

Contents

Preface I

1 Introduction 1

1.1 GR5405 SDK 1

1.2 Bluetooth LE Protocol Stack 1

2 GR5405 Bluetooth LE Software Platform 3

2.1 Hardware Architecture 3

2.2 Software Architecture 4

2.3 Memory Mapping 5

2.4 Flash Memory Mapping 7

2.4.1 SCA 7

2.4.2 NVDS 10

2.5 RAM Mapping 10

2.5.1 Typical RAM Layout 11

2.5.2 RAM Power Management 12

2.6 SDK Directory Structure 13

2.7 Tools 15

3 Bootloader 17

4 Development and Debugging with SDK in Keil 18

4.1 Installing Keil MDK 18

4.2 Installing SDK 19

4.3 Building a Bluetooth LE Application 19

4.3.1 Preparing ble_app_example 19

4.3.2 Configuring a Project 21

4.3.2.1 Configuring *custom_config.h* 21

4.3.2.2 Configuring Memory Layout 28

4.3.3 Adding User Code 29

4.3.3.1 Modifying the main() Function 29

4.3.3.2 Implementing Bluetooth LE Service Logics 30

4.3.3.3 Scheduling BLE_Stack_IRQ, BLE_SDK_IRQ, and Applications 31

4.4 Generating Firmware 32

4.5 Downloading .hex Files to Flash 32

4.6 Debugging 35

4.6.1 Configuring the Debugger 35

4.6.2 Starting Debugging 37

4.6.3 Outputting Debug Logs 38

4.6.3.1 Module Initialization 38

4.6.3.2 Application 40

4.6.4 Debugging with GRToolbox.....41
4.7 Download and Debugging with IAR.....41

5 Glossary..... 43

1 Introduction

The Goodix GR5405 is an automotive Bluetooth LE 5.3 SoC designed to operate across a wide temperature range from –40°C to 105°C, and is AEC-Q100 Grade 2 certified. It is suitable for various automotive applications, including digital key and Tire Pressure Monitoring System (TPMS). Additionally, it supports Bluetooth Mesh networking protocols.

Based on Arm® Cortex®-M4F CPU core running at 64 MHz, the GR5405 integrates a 2.4 GHz RF transceiver, Bluetooth LE 5.3 protocol stack, 512 KB on-chip Flash memory, 96 KB system SRAM, and a rich set of peripherals. It provides outstanding RF performance, with a maximum TX power of +15 dBm, an RX sensitivity of -99 dBm in Bluetooth LE 1 Mbps mode, achieving an overall link budget of up to 114 dB.

The GR5405 supports connection between multiple centrals and multiple peripherals. It can be configured as a Broadcaster, an Observer, a Peripheral, or a Central, and supports the combination of all the above roles.

1.1 GR5405 SDK

The GR5405 Software Development Kit (SDK) provides comprehensive software development support for GR5405 SoCs. The SDK contains Bluetooth LE APIs, Mesh APIs, System APIs, peripheral drivers, a tool for debugging and download, project example code, and related user documents.

1.2 Bluetooth LE Protocol Stack

The Bluetooth LE Protocol Stack (Bluetooth LE Stack) architecture is as shown in the figure below.

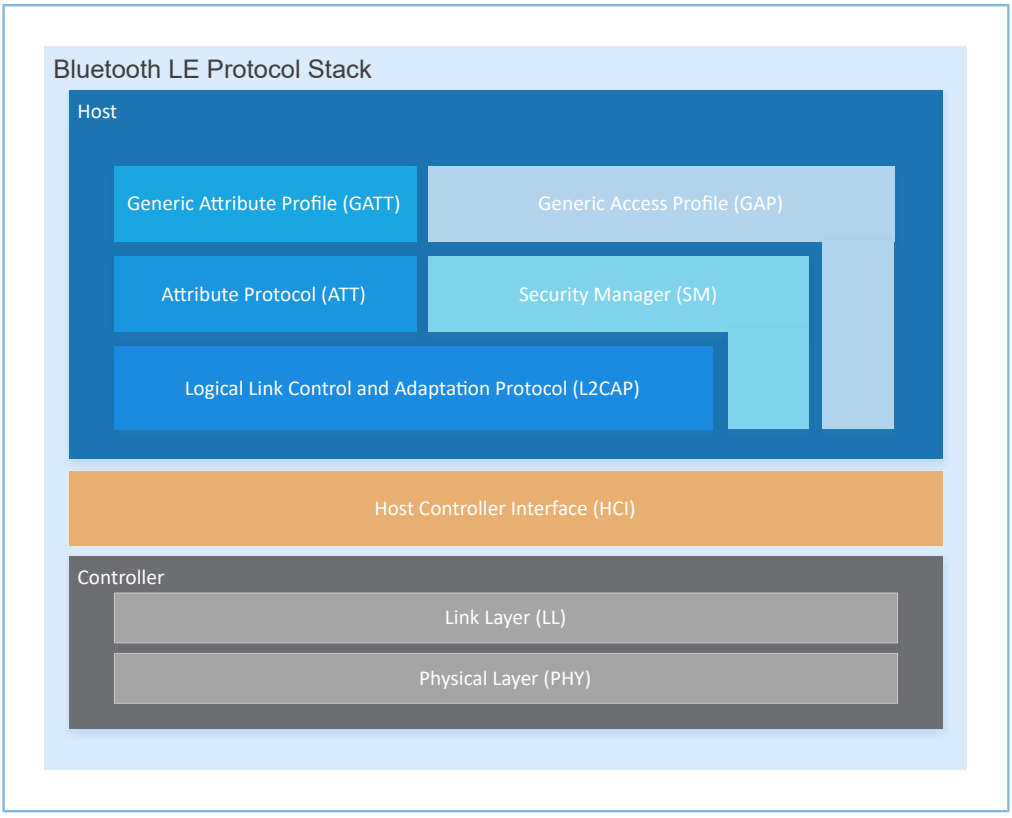

Figure 1-1 Bluetooth LE Stack architecture

The Bluetooth LE Stack consists of the Controller, the Host Controller Interface (HCI), and the Host.

Controller

- Physical Layer (PHY): Supports 1 Mbps, 2 Mbps, 500 kbps, and 125 kbps adaptive frequency hopping and Gaussian Frequency Shift Keying (GFSK).
- Link Layer (LL): Controls the RF state of devices. Devices are in one of the following five states, and can switch between the states on demand: Standby, Advertising, Scanning, Initiating, and Connection.

HCI

- HCI: Enables communication between Host and Controller, supported by software interfaces or standard hardware interfaces, for example, UART, Secure Digital (SD), or USB. HCI commands and events are transferred between Host and Controller through HCI.

Host

- Logical Link Control and Adaptation Protocol (L2CAP): Provides channel multiplexing and data segmentation and reassembly services for upper layers. It also supports logic end-to-end data communication.
- Security Manager (SM): Defines pairing and key distribution methods, providing upper-layer protocol stacks and applications with end-to-end secure connection and data exchange functionalities.
- Generic Access Profile (GAP): Provides upper-layer applications and profiles with interfaces to communicate and interact with protocol stacks, fulfilling functionalities such as advertising, scanning, connection initiation, service discovery, connection parameter update, secure process initiation, and response.
- Attribute Protocol (ATT): Defines service data interaction protocols between a server and a client.
- Generic Attribute Profile (GATT): Based on the top of ATT, it defines a series of communication procedures for upper-layer applications, profiles, and services to exchange service data between GATT Client and GATT Server.

Tip:

- For more information about Bluetooth LE technologies and protocols, visit the Bluetooth SIG official website: https://www.bluetooth.com.
- Specifications of GAP, SM, L2CAP, and GATT are provided in *Bluetooth Core Spec*. Specifications of other profiles/services at the Bluetooth LE application layer are available on the GATT Specs page. Assigned numbers, IDs, and code which may be used by Bluetooth LE applications are listed on the Assigned Numbers page.

---

2 GR5405 Bluetooth LE Software Platform

The GR5405 SDK is designed for GR5405 SoCs, to help users develop Bluetooth LE applications. It integrates Bluetooth LE 5.3 APIs, System APIs, and peripheral driver APIs, with various example projects and instruction documents for Bluetooth and peripheral applications. Application developers are able to quickly develop and iterate products based on example projects in the GR5405 SDK.

2.1 Hardware Architecture

The GR5405 hardware architecture is shown as follows.

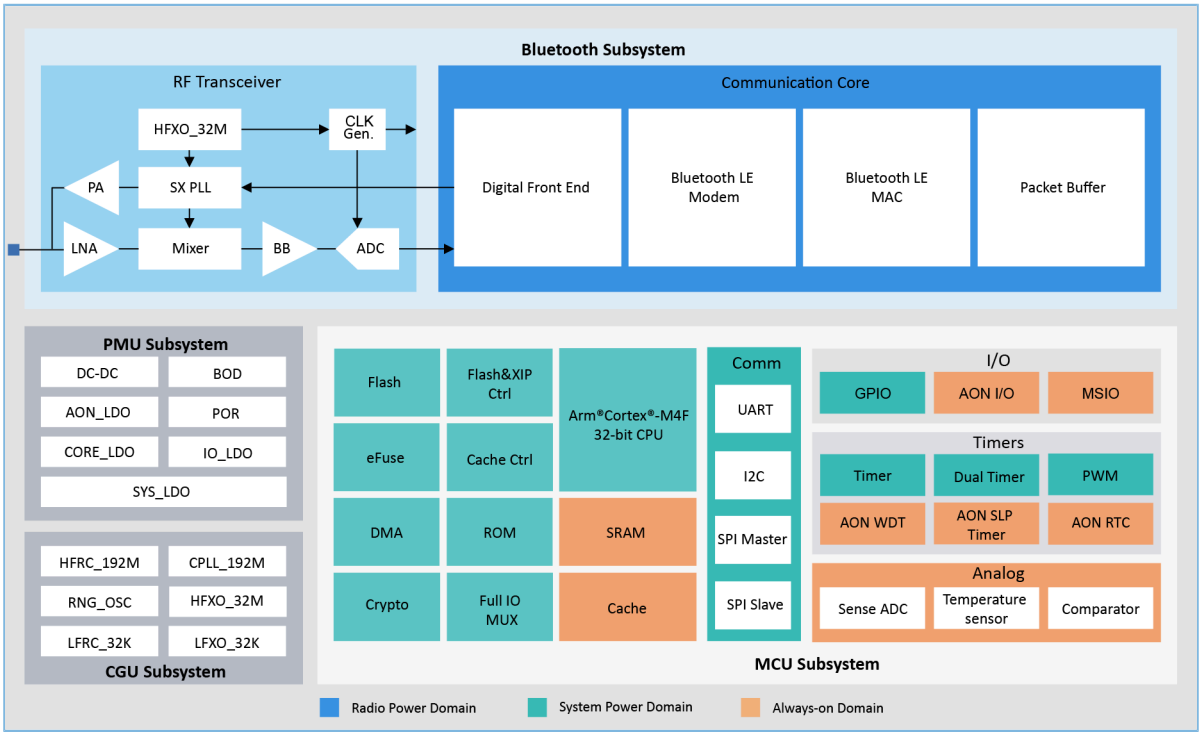

Figure 2-1 GR5405 hardware architecture

- Arm® Cortex®-M4F: GR5405 CPU. Bluetooth LE Stack and application code run on the CPU.
- SRAM: static random access memory that provides memory space for program execution
- ROM: read-only memory, containing the software code (cannot be modified after being programmed) for Bootloader and Bluetooth LE Stack
- Flash: Flash memory unit embedded in the SoC. It stores user code and data, and supports the Execute in Place (XIP) mode for user code.
- Peripherals: GPIO, DMA, I2C, SPI, UART, PWM, Timer, ADC, TRNG, and more
- RF Transceiver: 2.4 GHz RF transceiver
- Communication Core: PHY of Bluetooth 5.3 Protocol Stack Controller, enabling communication between the software protocol stack and 2.4 GHz RF hardware

- Power Management Unit (PMU): It supplies power for system modules, and sets reasonable parameters for modules, including DC-DC, SYS_LDO, IO-LDO, CORE_LDO, and RF Subsystem, based on configuration parameters and the current operating state of the system, so that the power can be managed automatically.

Tip:

For more details about device modules, refer to *GR5405 Datasheet*.

2.2 Software Architecture

The software architecture of GR5405 SDK is shown below.

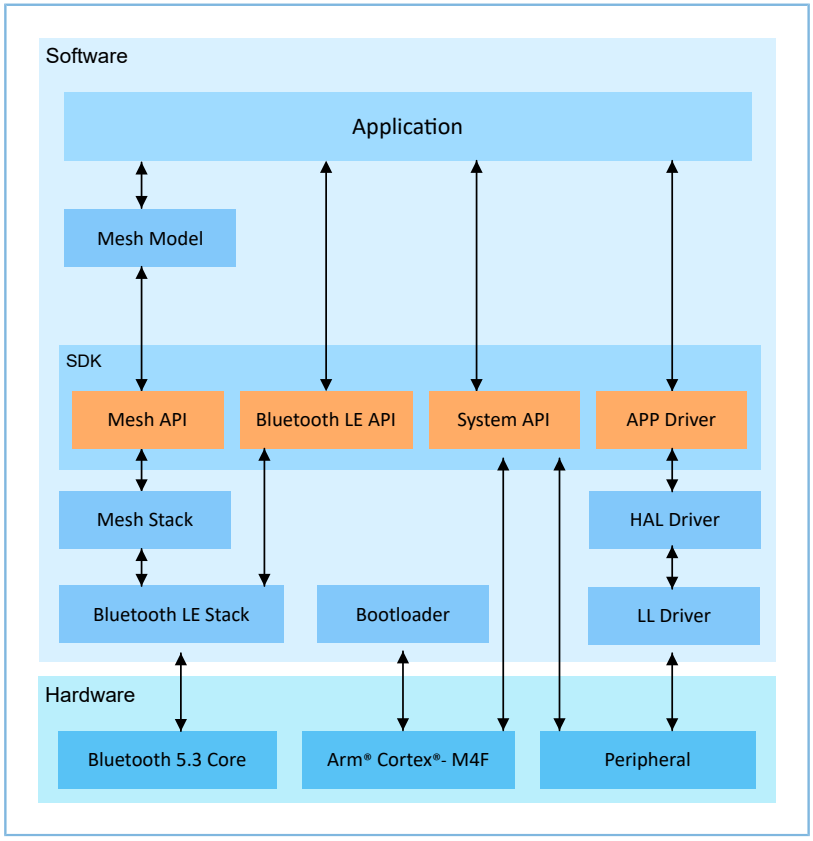

Figure 2-2 GR5405 software architecture

- Bootloader

  A boot program built in GR5405 SoCs, used for GR5405 software and hardware environment initialization, and to check and start applications

- Bluetooth LE Stack

  The core to implement Bluetooth LE protocols. It consists of Controller, HCI, and Host protocols (including LL, HCI, L2CAP, GAP, SM, and GATT), and supports roles of Broadcaster, Observer, Peripheral, and Central.

- Mesh Stack

  The core to implement Bluetooth LE Mesh protocols. It integrates Bearer Layer, Network Layer, Lower Transport Layer, Upper Transport Layer, Access Layer, and some functionalities of the Foundation Model Layer.

- LL Driver

  Low Layer (LL) drivers which control and manage peripherals by registers

- HAL Driver

  Hardware Abstraction Layer (HAL) drivers; the HAL Driver layer is between the APP Driver layer and the LL Driver layer. HAL drivers offer a set of standard APIs, to allow the APP driver layer to access the LL peripheral resources by calling HAL APIs.

Note:

Generally, HAL APIs are used for developing LL drivers and system services, not for developing common applications. Therefore, it is not recommended for developers to directly call HAL APIs.

- Bluetooth LE SDK

  SDK that provides easy-to-use Mesh APIs, Bluetooth LE APIs, system APIs, and APP Driver APIs

  - Mesh APIs: Include APIs required for developing Mesh applications.
  - Bluetooth LE APIs: Include L2CAP, GAP, SM, and GATT APIs.
  - System APIs: Provide APIs for Non-volatile Data Storage (NVDS), Device Firmware Update (DFU), system power management, and generic system-level access.
  - APP Driver: the application driver layer, which encapsulates common functionalities of various peripherals into APIs based on HAL drivers. These APIs not only retain the characteristics of HAL drivers, but also feature more stable, secure, and user-friendly. Generally, developers are recommended to use the APIs at the APP layer.

- Mesh Model

  It contains example implementation code for standard Mesh Model (such as Lightness Model) from Bluetooth SIG. You can refer to the example code to develop Mesh applications.

- Application

  The SDK provides abundant Bluetooth and peripheral example projects. Each project contains compiled binary files; you can download these files to GR5405 SoCs for operation and test. In addition, GRToolbox (Android) provides rich functionalities to allow users to test most Bluetooth applications with ease.

2.3 Memory Mapping

The memory mapping of a GR5405 SoC is shown below.

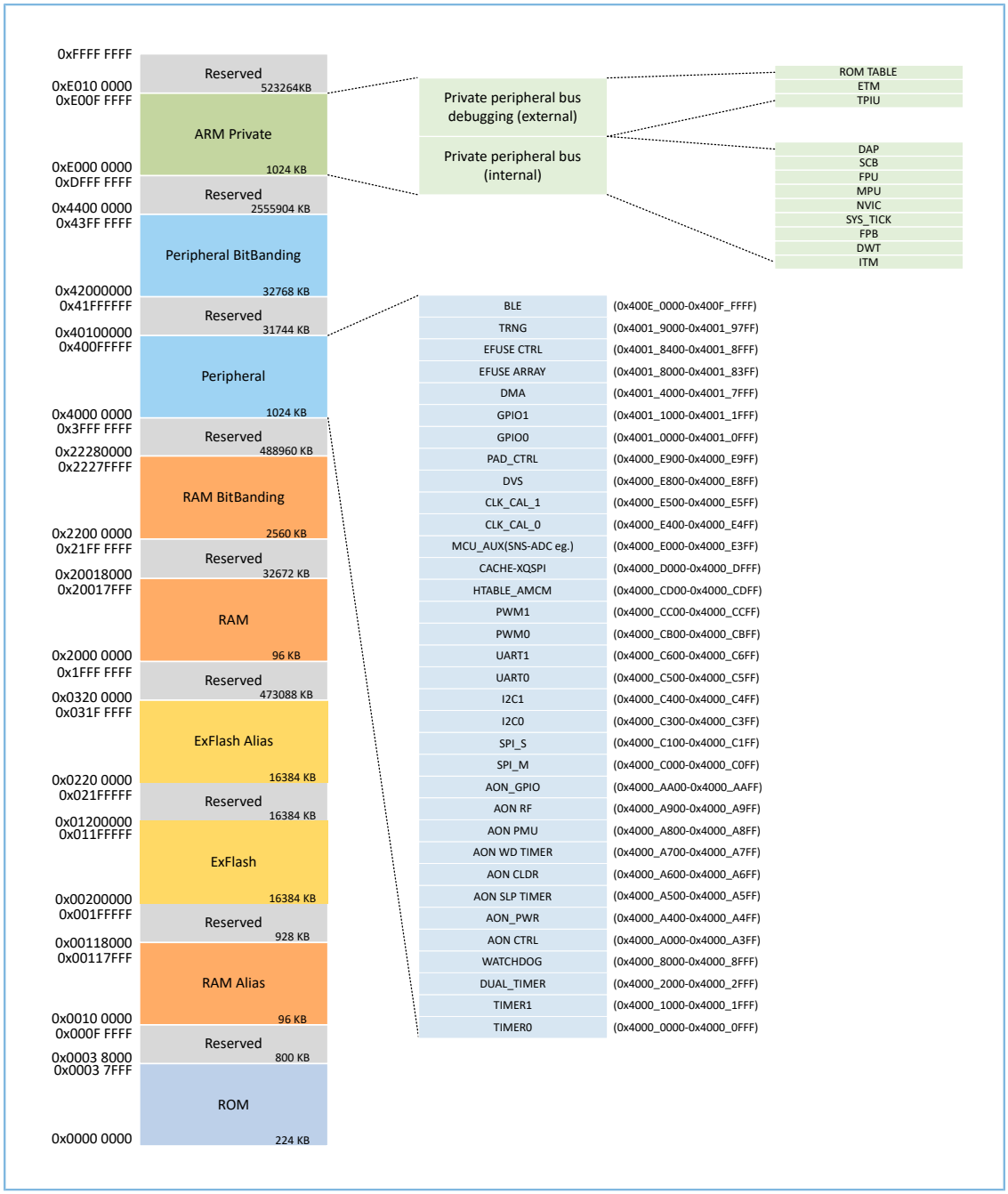

Figure 2-3 GR5405 memory mapping

- RAM: 96 KB in total; 0x0010_0000 to 0x0011_7FFF, or 0x2000_0000 to 0x2001_7FFF.
  - 0x2000_0000 to 0x2001_7FFF: Variables of the SDK including RW, ZI, HEAP, and STACK are in this range. The 16 KB storage area at the end of SRAM can be used as Exchange Memory (EM) for baseband when you configure Bluetooth LE projects. The actual area used as EM is determined by the maximum Bluetooth LE service volume configured in *custom_config.h*. The unused EM area will form a contiguous address space with other SRAM areas. In addition, bit field operations are supported in the region from 0x2000_0000 to 0x2001_3FFF, mapping to the region from 0x2200_0000 to 0x2227_FFFF, in which atomic operations are supported.

  - 0x0010_0000 to 0x0011_7FFF: This region features higher access efficiency thanks to the Cortex® -M4F architecture. Therefore, executable code RAM_CODE is in this area.

- Flash: Internal Flash of GR5405 SoCs is 512 KB, from 0x0020_0000 to 0x0027_FFFF.

2.4 Flash Memory Mapping

GR5405 packages an on-chip erasable Flash memory, which supports XQSPI bus interface. This Flash memory physically consists of several 4 KB Flash sectors; it can be logically divided into storage areas for different purposes based on application scenarios.

The Flash memory layout for typical GR5405 application scenarios is shown below.

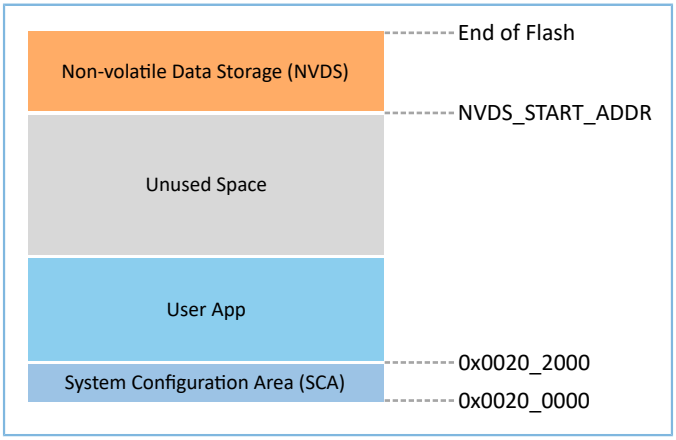

Figure 2-4 Flash memory layout

- System Configuration Area (SCA): an area to store configurations such as system boot parameters
- User App: an area to store application firmware
- Unused Space: a free area for developers. For example, developers can store new application firmware in the Unused Space temporarily during DFU.

Note:

- You can configure the start address of NVDS and the number of occupied sectors according to Flash memory layout of products. For more information about the configuration, refer to "Section 4.3.2.1 Configuring custom_config.h".
- The start address of NVDS shall be aligned with that of the Flash sectors.
- Developers can implement non-volatile data storage by porting open-source components such as LittleFS according to their actual needs.

2.4.1 SCA

SCA is in the first two sectors (8 KB in total; 0x0020_0000 to 0x0020_2000) of Flash memory. It mainly stores flags and other system configuration parameters used during system boot.

During firmware download, the download algorithm or GProgrammer will generate Image Info based on the BUILD_IN_APP_INFO structure in the application firmware, and program the Image Info (stored in SCA) to Flash along with the application firmware. During system boot, Bootloader will check the boot information in SCA, and then jump to the entry address of the firmware if the check passes.

The BUILD_IN_APP_INFO structure is defined and configured as follows:

Tip:

The BUILD_IN_APP_INFO structure is in SDK_Folder\platform\soc\common\gr_platform.c, and SDK_Folder is the root directory of GR5405 SDK.

```
const APP_INFO_t BUILD_IN_APP_INFO __attribute__((at(APP_INFO_ADDR))) = {
    .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
.reserved1        = {APP_INFO_RESERVED}
};
```

- app_pattern: a fixed value 0x47525858
- app_info_version: firmware version information, corresponding to APP_INFO_VERSION
- chip_ver: version of the SoC that the firmware runs on, corresponding to CHIP_VER in *custom_config.h*
- load_addr: firmware load address, corresponding to APP_CODE_LOAD_ADDR in *custom_config.h*
- run_addr: firmware run address, corresponding to APP_CODE_RUN_ADDR in *custom_config.h*
- app_info_sum: checksum of firmware information, which is automatically calculated by CHECK_SUM
- check_img: system boot configuration parameter, corresponding to BOOT_CHECK_IMAGE in *custom_config.h*. When check_img is set to **1**, Bootloader will check the firmware at booting.
- boot_delay: boot configuration parameter, corresponding to BOOT_LONG_TIME in *custom_config.h*. When boot_delay is set to **1**, the system cold boot will be launched after a one-second delay.
- sec_cfg: security configuration parameter, reserved
- comments: firmware information, up to 12 bytes

The SCA layout is shown below.

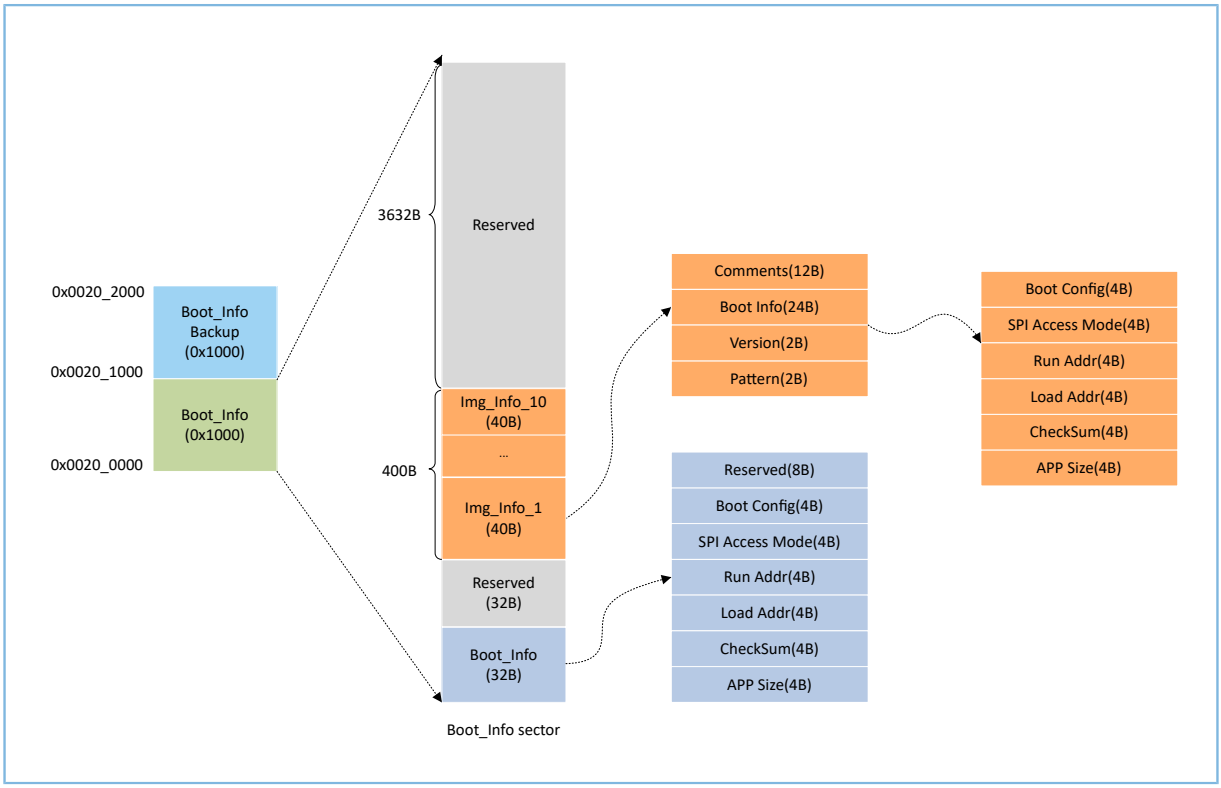

Figure 2-5 SCA layout

- Boot_Info and Boot_Info Backup store the same information. The latter is the backup of the Boot_Info.
- The firmware boot information is stored in the Boot_Info (32 B) area. During system boot, Bootloader will check the boot information, and then jump to the entry address of the firmware if the check passes.
  - Boot Config: This area stores the system boot configuration information.
  - SPI Access Mode: This area stores the SPI access mode configuration. It is a fixed configuration of the system and cannot be modified.
  - Run Addr: Indicates the firmware run address, corresponding to run_addr of BUILD_IN_APP_INFO.
  - Load Addr: Indicates the firmware load address, corresponding to load_addr of BUILD_IN_APP_INFO.
  - CheckSum: This area stores the firmware checksum which is calculated automatically by the download algorithm after firmware is generated.
  - APP Size: This area stores the firmware size which is calculated automatically by the download algorithm after firmware is generated.
- Up to 10 pieces of firmware information can be stored in Img_Info areas. Firmware information is stored in Img_Info areas when you use GProgrammer to download firmware or update firmware in DFU mode.
  - Comments: This area stores the descriptive information (up to 12 characters) about firmware. Every time a firmware file is generated, the file name will be saved in the Comments area by the download algorithm.

- Boot Info (24 B): This area stores the firmware boot information which is the same as the low 24-byte information in the Boot_Info (32 B) area mentioned above.
- Version: This area stores the firmware version, corresponding to VERSION in the *custom_config.h*.
- Pattern: This area stores a fixed value 0x4744.

2.4.2 NVDS

Non-volatile Data Storage (NVDS) is a lightweight key-value pair data storage system that uses the Flash read/write APIs provided by the Flash Hardware Abstraction Layer (Flash HAL) to store data in Flash memory, ensuring that data will not get lost even if the system is powered off.

NVDS is an ideal choice for storing small data blocks, for example, application configuration parameters, calibration data, state information, and user information. Bluetooth LE Stack stores parameters such as device binding parameters in NVDS.

NVDS features:

- Each storage item (tag) has a unique tag ID for identification. User applications can read and change data according to tag IDs, regardless of physical storage addresses.
- It is optimized based on media characteristics of Flash memory and supports data check, garbage collection, and wear-leveling.
- The size and start address of NVDS area are configurable. Flash is divided into sectors, with each sector being 4 KB in size. NVDS area can be configured to occupy several sectors, and the start address shall be 4 KB aligned.

By default, GR5405 SDK uses the last several sectors in Flash for NVDS, with the start address being the Flash end address minus the NVDS area size. You can specify the start address and the number of sectors by configuring NVDS_START_ADDR and NVDS_NUM_SECTOR in *custom_config.h*. Note that NVDS_NUM_SECTOR excludes the NVDS garbage collection area. The total number of sectors occupied by NVDS is **NVDS_NUM_SECTOR + 1**.

Note:

For details and instructions on NVDS, refer to *GR54xx NVDS User Manual*.

2.5 RAM Mapping

The RAM start address is 0x2000_0000, and it comprises six RAM blocks, each with a size of 16 KB, totaling 96 KB. Each RAM block can be independently powered on or off by software.

The 96 KB RAM layout is shown below.

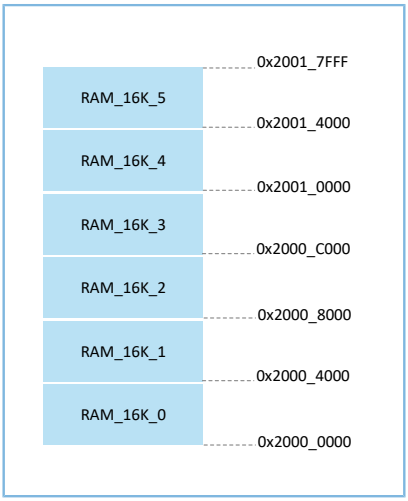

Figure 2-6 96 KB RAM layout

Applications run in Execute in Place (XIP) mode. User applications are stored in on-chip Flash, and applications use the same space for running and loading. When the system is powered on, it fetches and executes commands from Flash directly through the Cache Controller.

2.5.1 Typical RAM Layout

The typical RAM layout with Bluetooth LE projects in running is shown below. Developers are able to modify the RAM layout based on product needs.

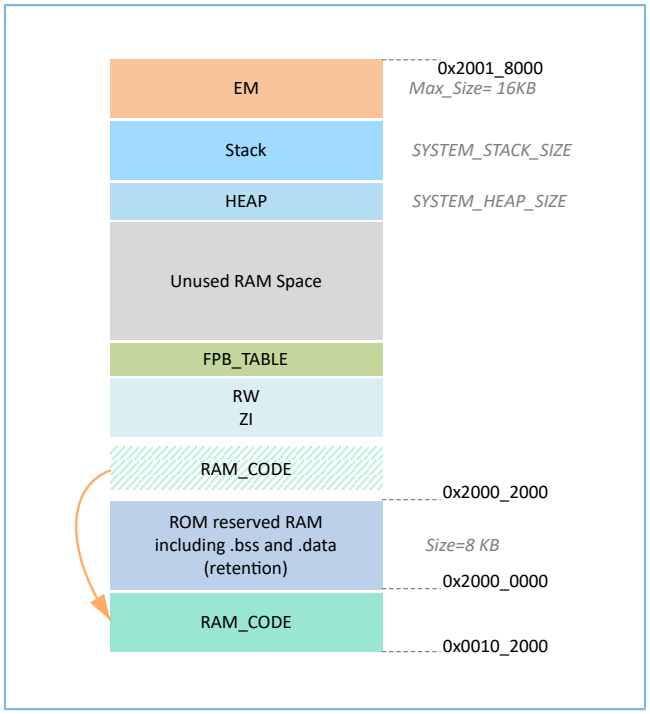

Figure 2-7 RAM layout in XIP mode (with Bluetooth LE projects)

- RAM_CODE saves code that is executed in RAM. To boost the efficiency in execution, it is recommended to define this region in the Aliasing memory (at physical address 0x00100000).
- EM is used by the Bluetooth LE core. It is managed together with SRAM used by the MCU, located at the highest address space of SRAM. EM size is determined by the Bluetooth service volume configured in *custom_config.h*. If no Bluetooth LE service is included in the project, the value of the BLE_SUPPORT macro in *custom_config.h* can be set to 0.
- Stack stores the task call stack. In peripheral projects without Bluetooth LE services, Stack is defined at the highest address of RAM. In projects with Bluetooth LE services, Stack is defined after the address of EM. The Stack size is defined by the SYSTEM_STACK_SIZE macro. You need to determine the size according to the function call depth and the consumption of the call stack in the project.

2.5.2 RAM Power Management

Each RAM block has three power modes: Full Power, Retention Power, and Power Off.

- Full Power: The system is in active state; MCU is permitted to read from and write to RAM blocks.
- Retention Power: The system is in sleep state; data in RAM blocks does not get lost and is ready for use by the system when it switches from sleep state to active state.
- Power off: The system is in power-off state; RAM blocks will be powered off and the data in the blocks will get lost. Therefore, you need to save the data before the system is powered off.

By default, the PMU in the GR5405 enables all RAM power sources when the system starts. The GR5405 SDK also provides a complete set of RAM power management APIs. You can configure the power state of RAM blocks based on application needs.

By default, the system enables automatic RAM power management mode during boot: It automatically implements power mode control of RAM blocks according to RAM usage of applications. The configuration rules are provided as follows:

- When the system is in active state, set the unused RAM blocks to **Power off** mode, and RAM blocks to be used to **Full Power** mode.
- When the system is in sleep state, set the unused RAM blocks to **Power off** mode, and RAM blocks to be used to **Retention Power** mode.

Recommended RAM configurations in practice are described below:

- In Bluetooth LE applications, the first 8 KB of RAM_16K_0 are reserved for Bootloader and Bluetooth LE Stack only, not available for applications. When the system is in active state, RAM_16K_0 shall be in **Full Power** mode; when the system is in sleep state, RAM_16K_0 shall be in **Retention Power** mode. Non-Bluetooth LE MCU applications can use this RAM block.
- Purposes of RAM_16K_1 and subsequent RAM blocks are defined by applications. The GR5405 RAM has been reasonably arranged according to execution efficiency and SRAM utilization. You can also re-configure it according to actual application requirements. The power mode of these RAM blocks can be enabled, or be controlled by applications.

Note:

- An MCU access is permitted only when a RAM block is in **Full Power** mode.
- Details about RAM power management APIs are in SDK_`Folder\components\sdk\platform_sdk.h`.

2.6 SDK Directory Structure

The folder directory structure of GR5405 SDK is shown as follows.

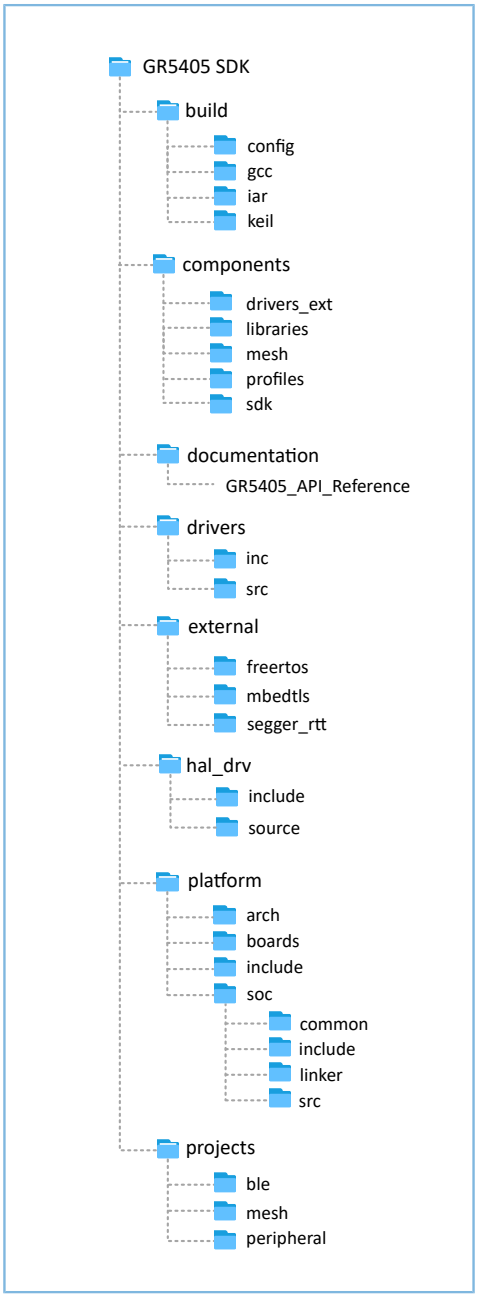

Figure 2-8 GR5405 SDK directory structure

Detailed description of folders in GR5405 SDK is shown below.

Table 2-1 GR5405 SDK folders

Folder	Description
build\config	Project configuration directory that stores the *custom_config.h* template file. This file is used to configure project parameters.
build\gcc	GCC tools
build\keil	Keil MDK tools

Folder	Description
build\iar	IAR tools
components\drivers_ext	Drivers of third-party components on the development board
components\libraries	Libraries provided in GR5405 SDK
components\profiles	Source files of GATT Services/Service Clients implementation examples
components\mesh	Mesh API header files, library files, and source files to implement Mesh model applications
components\sdk	API header files
documentation	*GR5405 API Reference Manual*
drivers\inc	Driver API header files which are easy to use for application developers
drivers\src	Driver API source code which is easy to use for application developers
external\freertos	Source code of FreeRTOS (a third-party program)
external\mbedtls	Source code of Mbed TLS (a third-party program)
external\segger_rtt	Source code of SEGGER RTT (a third-party program)
hal_drv\include	Header files of HAL driver APIs
hal_drv\source	Source files of HAL driver APIs
platform\arch	Toolchain files of CMSIS
platform\boards	Source files for initializing GR5405 Starter Kit Board. The files are used for initializing basic peripherals at board level.
platform\include	Common header files related to platform
platform\soc\common	Public source files compatible to GR5405 SoCs. The files include *gr_interrupt.c*, *gr_platform.c*, and *gr_system.c*.
platform\soc\linker	Symbol table files and library files for the linker
platform\soc\include	Common header files closely related to underlying driver configurations such as registers and clock configurations
platform\soc\src	*gr_soc.c* which is about initialization processes closely related to SoC implementation. The processes include initializing Flash and NVDS, configuring crystal, and calibrating PMU.
projects\ble	Bluetooth LE application project examples, such as ble_app_template
projects\mesh	Mesh demo project example
projects\peripheral	Peripheral project examples of an SoC

2.7 Tools

Developers can use the following tools to develop and debug GR5405 applications.

Table 2-2 Development/Debugging tools

Name	Description	Recommended Version
GProgrammer	A firmware programming tool that supports functionalities such as firmware download, Flash read/write, and eFuse download. Available on both Windows and Linux platforms.	V2.0.2 and later
GRUart	A serial port debugging tool. Available on Windows platform only.	V2.1 and later
GRDirect Test Mode Tool	An RF test tool that controls the Device Under Test (DUT) to perform Direct Test Mode (DTM) tests by delivering HCI commands. Available on Windows platform only.	V1.5.5 and later
GRPLT	A mass production configuration tool for the online mass production programming board (PLT) that supports batch firmware download, Flash data programming, crystal calibration, and functionality testing. Available on Windows platform only.	V1.6.0.0.03 and later
GRCalibration	A GR5xx Bluetooth crystal oscillator calibration tool that is designed for calibrating the frequency offset of the 32 MHz crystal oscillator on the GR551x SoC, GR5405 SoC, and the PLT production programming board. Available on Windows platform only.	V1.1.0 and later
GRToolbox	A mobile APP that enables users to scan for Bluetooth devices, set connection parameters, demonstrate standard profiles, and debug profiles/services from Goodix Bluetooth LE platform. Both Android and iOS versions are supported.	V2.21 and later
GRMesh	A mobile APP for configuring, managing, and controlling Goodix Bluetooth Mesh network. Only Android version is supported.	V1.07 and later

3 Bootloader

The GR5405 code runs in XIP mode. When the system is powered on, the Bootloader first reads the system boot configuration information from SCA, then performs application firmware integrity check and initialize Cache and XIP controller accordingly, and finally jumps to the code running space to run firmware.

The application boot procedures of the GR5405 SDK are shown as follows.

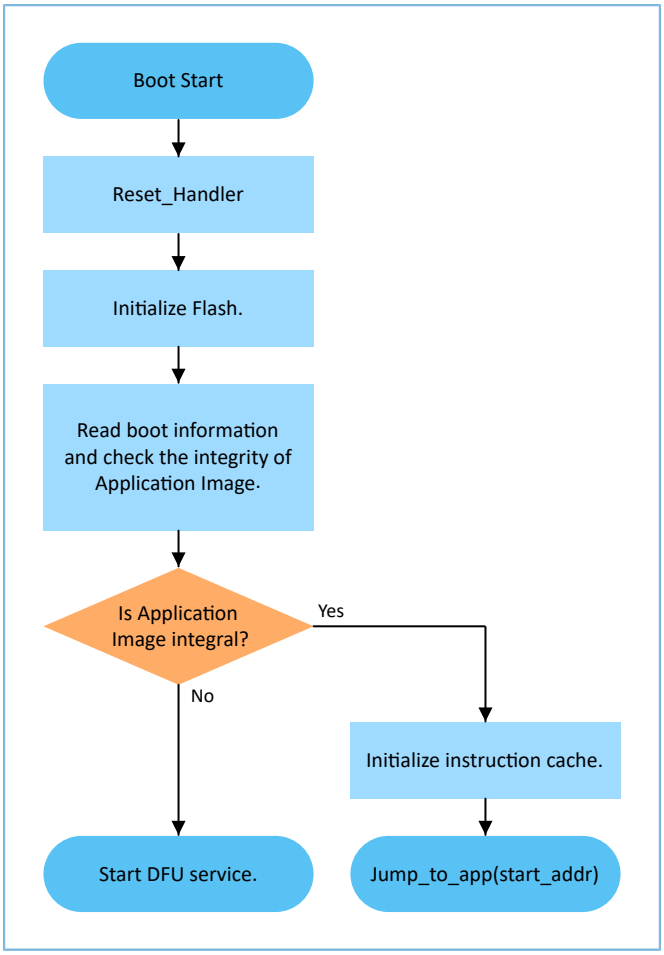

Figure 3-1 Application boot procedures of the GR5405 SDK

1. When the device is powered on, CPU jumps to 0x0000_0000 to extract the extended stack pointer (ESP) of C-Stack and assigns the value to the main stack pointer (MSP). Then, the program counter (PC) jumps to 0x0000_004, and executes Reset_Handler in ROM to enter the Bootloader.
2. Bootloader initializes Flash.
3. Bootloader reads boot information from SCA in Flash and checks application firmware integrity.
4. If the integrity check fails, the Bootloader enters J-Link DFU mode. You can update application firmware in Flash with GProgrammer and J-Link.
5. If the integrity check passes, the Bootloader jumps to the run address of the application firmware in Flash to execute the code after completing the XIP configuration.

4 Development and Debugging with SDK in Keil

This chapter introduces how to build, compile, download, and debug Bluetooth LE applications with the SDK in Keil.

4.1 Installing Keil MDK

Keil MDK-ARM IDE (Keil) is an Integrated Development Environment (IDE) provided by ARM® for Cortex® and ARM devices. You can download and install the Keil installation package from the Keil official website https://www.keil.com/demo/eval/arm.htm. For the GR5405 SDK, Keil V5.20 or a later version shall be installed.

Note:

For more information about how to use Keil MDK-ARM IDE, refer to online manuals provided by ARM: https://www.keil.com/support/man_arm.htm.

The main interface of Keil is as shown below.

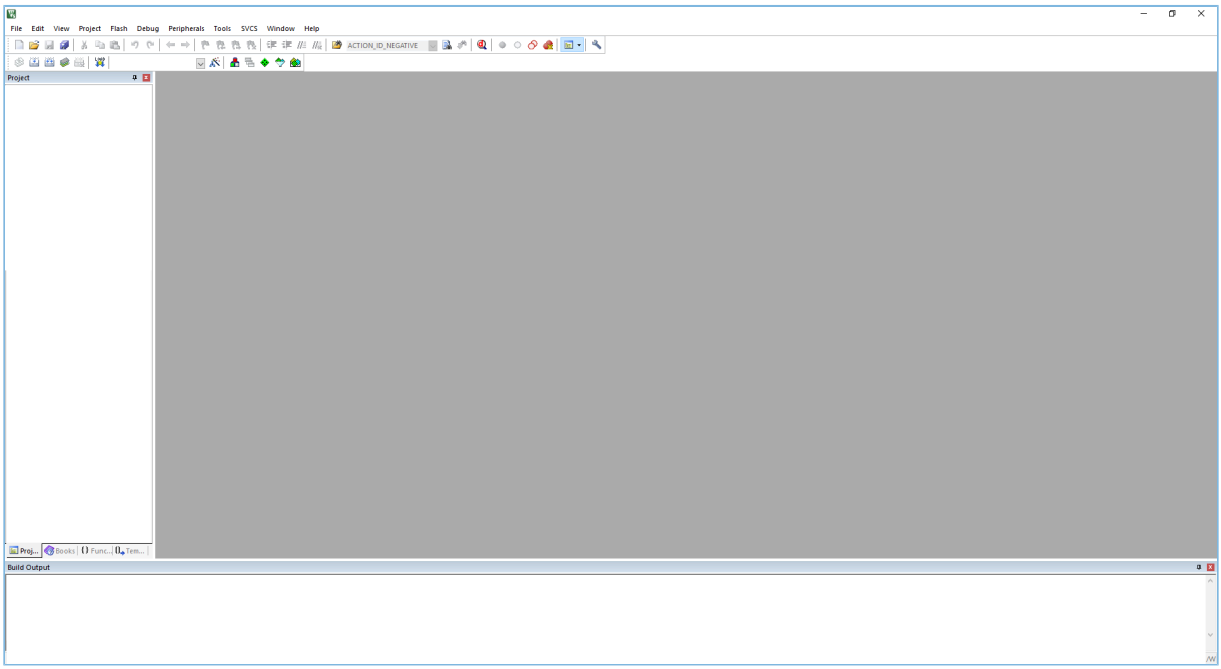

Figure 4-1 Keil interface

Frequently used function buttons of Keil are listed below:

Table 4-1 Frequently used function buttons of Keil

Button	Description
	Options for Target
	Start/Stop Debug Session
	Download
	Build

4.2 Installing SDK

GR5405 SDK is in a .zip file. You can access the details after extracting the file.

Note:

- SDK_Folder is the root directory of GR5405 SDK.
- Keil_Folder is the root directory of Keil.

4.3 Building a Bluetooth LE Application

This section introduces how to quickly build a custom Bluetooth LE application with Keil and GR5405 SDK.

4.3.1 Preparing ble_app_example

This section elaborates on how to create a project based on the template project provided in GR5405 SDK.

Open SDK_`Folder\projects\ble\ble_peripheral\`, copy ble_app_template to the current directory, and rename it as **ble_app_example**. Change the base name of *.uvoptx* and *.uvprojx* files in `ble_app_example\Keil_` 5 to **ble_app_example**.

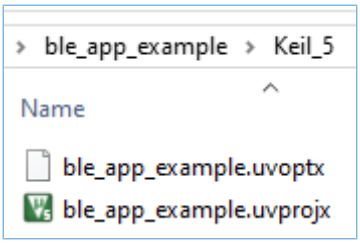

Figure 4-2 ble_app_example folder

Double-click *ble_app_example.uvprojx* to open the project example in Keil. Click , and the **Options for Target 'GRxx_Soc'** window opens. Choose the **Output** tab, and type **ble_app_example** in the **Name of Executable** field, to name the output file as **ble_app_example**.

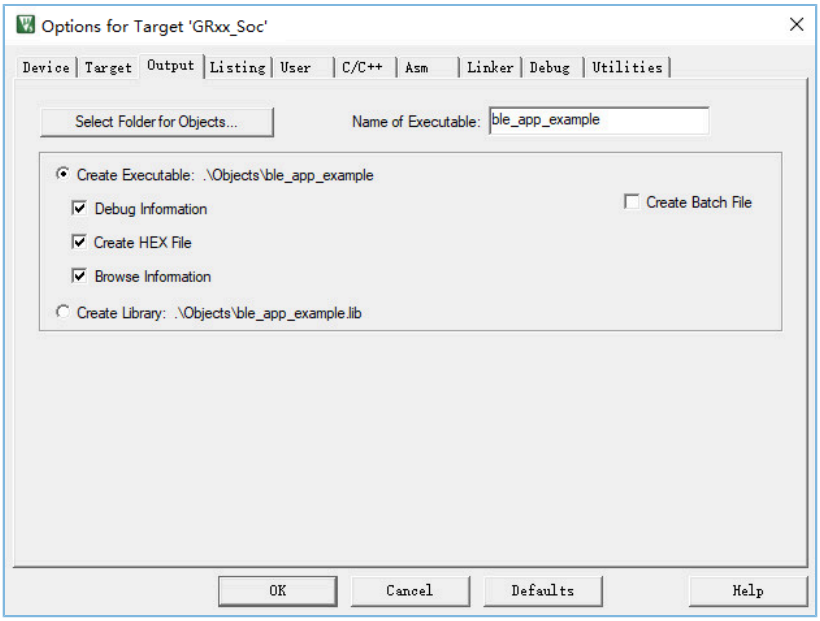

Figure 4-3 Modifications to **Name of Executable**

All groups of the ble_app_example project are available in the **Project** window of Keil.

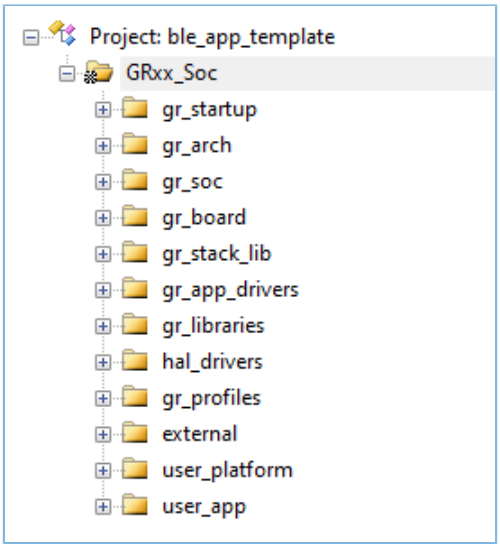

Figure 4-4 ble_app_example groups

Groups of the ble_app_example project are mainly in two categories: SDK groups and User groups.

- SDK groups

  The SDK groups are detailed as follows:

Table 4-2 SDK groups

SDK Group Name	Description
gr_startup	System boot file
gr_arch	Initialization configuration files and system interrupt API implementation files for System Core and PMU

SDK Group Name	Description
gr_soc	*gr_soc.c* which is used for initializing and calibrating modules such as Clock, PMU, and Vector before entering the main() function
gr_board	Board-level description file which is used for implementing components such as log, key, and LED
gr_stack_lib	A SDK .lib file
gr_app_drivers	Driver API source files which are easy to use for application developers. You can add related application drivers on demand.
gr_libraries	Open source files of common assistant software modules and peripheral drivers provided in the SDK
hal_drivers	Source files for HAL driver APIs. You can add necessary HAL drivers for projects on demand.
gr_profiles	Source files of GATT Services/Service Clients. You can add necessary GATT source files for projects on demand.
external	Source files for third-party programs, such as FreeRTOS and SEGGER RTT. You can add third-party programs on demand.

- User groups

  The User groups are detailed as follows:

Table 4-3 User groups

User Group Name	Description
user_platform	Software and hardware resource setting and application initialization; you need to execute corresponding APIs on demand.
user_app	main() function entries and other source files created by developers, which are used to configure runtime parameters of Bluetooth LE Stack and execute event handlers of GATT Services/Service Clients

4.3.2 Configuring a Project

You should configure corresponding project options according to product characteristics, including NVDS, code running mode, memory layout, and other configuration items.

4.3.2.1 Configuring *custom_config.h*

custom_config.h is used to configure parameters of application projects. Developers can directly modify the configurations in the file or configure parameters in the **Configuration Wizard** interface of Keil.

Tip:

custom_config.h of each application example project is in `Src\config` under project directory.

- Modify the configurations in *custom_config.h*.

  GR5405 SDK provides a template configuration file *custom_config.h* (in `SDK_Folder\build\config\custom_config.h`). You can directly modify the template file to configure parameters for application projects.

Table 4-4 Parameters in *custom_config.h*

Macro	Description
SOC_GR5405	Define the SoC version number.
CHIP_TYPE	Specify the SoC model. ◦ 0: GR5405 **Note**: During project compilation, configure this macro according to the SoC model in use.
CFG_APP_DRIVER_SUPPORT	Use the peripheral APP driver or not. ◦ 0: No ◦ 1: Yes
ENABLE_BACKTRACE_FEA	Enable/Disable the stack backtrace functionality. ◦ 0: Disable ◦ 1: Enable
APP_LOG_ENABLE	Enable/Disable the APP LOG module. ◦ 0: Disable ◦ 1: Enable
APP_LOG_STORE_ENABLE	Enable/Disable the APP LOG STORE module. ◦ 0: Disable ◦ 1: Enable
APP_LOG_PORT	Set the output mode of APP LOG module. ◦ 0: UART ◦ 1: J-Link RTT ◦ 2: ARM ITM
PLATFORM_SDK_INIT_ENABLE	Enable/Disable platform initialization. ◦ 0: Disable ◦ 1: Disable
PMU_CALIBRATION_ENABLE	Enable/Disable PMU calibration. When PMU calibration is enabled, the system monitors temperature and voltage automatically with adaptive adjustment. It is recommended to enable macro by default.

Macro	Description
	◦ 0: Disable ◦ 1: Enable **Note:** PMU calibration shall be enabled in high/low temperature scenarios.
NVDS_START_ADDR	Start address of NVDS in Flash; default: 0 If no address is specified, the end area in Flash will be used by default.
NVDS_NUM_SECTOR	Number of Flash sectors for NVDS; default: 3
SYSTEM_STACK_SIZE	Size of Call Stack required by applications. The default value is 8 KB. You can set the value as needed. **Note:** After compilation of ble_app_example, the Maximum Stack Usage is provided in K`eil_5\Objects\ble_app_example.htm` for reference.
SYSTEM_HEAP_SIZE	Size of Heap required by applications. The default value is 0 KB. You can set the value as needed.
CHIP_VER	Version of the SoC that the firmware runs on; default: 0x5405
APP_CODE_LOAD_ADDR*	Start address of the application storage area **Note:** This address shall be within the Flash address range.
APP_CODE_RUN_ADDR*	Start address of the application running space **Note:** The value shall be the same as APP_CODE_LOAD_ADDR, and applications run in XIP mode.
SYSTEM_CLOCK*	Set the system clock frequency. ◦ 0: 64 MHz ◦ 1: 32 MHz ◦ 2: 16 MHz (XO) ◦ 3: 16 MHz ◦ 4: 8 MHz ◦ 5: 2 MHz
RF_TX_PA_SELECT	Select an RF power amplifier. ◦ 1: SPA (supported TX power: –20 dBm to 5 dBm) ◦ 2: HPA (supported TX power: –10 dBm to 15 dBm)
SYSTEM_POWER_MODE	Set the system power supply mode. ◦ 0: Supplied by DC-DC

Macro	Description
	◦ 1: Supplied by SYS_LDO The system is supplied by SYS_LDO when an HPA is selected.
CFG_LF_ACCURACY_PPM	Bluetooth LE low-frequency sleep clock accuracy. The value shall range from 1 to 500 (unit: ppm).
CFG_LPCLK_INTERNAL_EN	Enable/Disable the OSC inside an SoC as the Bluetooth LE low-frequency sleep clock. If the OSC clock is enabled, CFG_LF_ACCURACY_PPM will be set to 500 ppm by force. ◦ 0: Disable ◦ 1: Enable
BOOT_LONG_TIME*	Set 1-second delay (during SoC boot before implementing the second half Bootloader). ◦ 0: No delay ◦ 1: Delay for 1 second.
BOOT_CHECK_IMAGE	Determine whether to check the image during cold boot in XIP mode. ◦ 0: Do not check. ◦ 1: Check.
BLE_SUPPORT	Support Bluetooth LE or not. ◦ 0: MCU only, no Bluetooth LE supported ◦ 1: Support Bluetooth LE.
CFG_CONTROLLER_ONLY	Support Bluetooth LE controller (for external host or HCI UART transmission) only or not. ◦ 0: Support Bluetooth LE controller and host. ◦ 1: Support Bluetooth LE controller only.
CFG_BT_BREDR	Support generating Bluetooth Classic link keys through the LE link or not. ◦ 0: No ◦ 1: Yes
DTM_TEST_V1_CMD_ENABLE	Enable/Disable DTM test command V1. ◦ 0: Disable ◦ 1: Enable
DTM_TEST_V2_CMD_ENABLE	Enable/Disable DTM test command V2. ◦ 0: Disable ◦ 1: Enable

Macro	Description
DTM_TEST_V3_CMD_ENABLE	Enable/Disable DTM test command V3. ◦ 0: Disable ◦ 1: Enable
DTM_TEST_V4_CMD_ENABLE	Enable/Disable DTM test command V4. ◦ 0: Disable ◦ 1: Enable
DTM_TEST_PRIVATE_CMD_ENABLE	Enable/Disable DTM test private commands. ◦ 0: Disable ◦ 1: Enable
CFG_MAX_PRFS	Maximum number of supported GATT Profiles/Services. You can set the value on demand. A larger value means more RAM space will be occupied. Range: 1–64
CFG_MAX_BOND_DEVS	Maximum number of devices that can be bonded; default: 4, minimum value: 1
CFG_MAX_CONNECTIONS	Maximum number of devices that can be connected; the number shall be no greater than 8. You can set the value on demand. A larger value means more RAM space will be occupied by Bluetooth LE Stack Heaps. The size of Bluetooth LE Stack Heaps is defined by the following four macros in *flash_scatter_config.h*: ◦ ENV_HEAP_SIZE ◦ ATT_DB_HEAP_SIZE ◦ KE_MSG_HEAP_SIZE ◦ NON_RET_HEAP_SIZE **Note:** The above four macros cannot be changed by developers.
CFG_MAX_ADVS	Maximum number of supported Bluetooth LE legacy advertising and extended advertising Range: 0–4 **Note:** The maximum number of supported Bluetooth LE legacy advertising and extended advertising shall be no greater than 4.
CFG_MAX_SCAN	Maximum number of supported Bluetooth LE device used for scanning Range: 0–1

Macro	Description
CFG_MUL_LINK_WITH_SAME_DEV	Support a device connecting to multiple slave devices or not. ◦ 0: No ◦ 1: Yes
CFG_CCC_SC_OOB_PAIR_SUPPORT	Support CCC 3.0 Bluetooth LE OOB secure pairing or not. ◦ 0: No ◦ 1: Yes
CFG_MASTER_SUPPORT	Support master role or not. ◦ 0: No ◦ 1: Yes
CFG_SLAVE_SUPPORT	Support slave role or not. ◦ 0: No ◦ 1: Yes
CFG_SUPPER_ADV_SUPPORT	Support a minimum interval of 5 ms for low-duty-cycle advertisement or not. ◦ 0: No; the minimum interval shall be 20 ms according to specifications. ◦ 1: Yes
CFG_LEGACY_PAIR_SUPPORT	Support legacy pairing or not. ◦ 0: No ◦ 1: Yes
CFG_SC_PAIR_SUPPORT	Support secure pairing or not. ◦ 0: No ◦ 1: Yes
CFG_COC_SUPPORT	Support Connection-oriented Channel (COC) or not. ◦ 0: No ◦ 1: Yes
CFG_GATTS_SUPPORT	Support GATT Server or not. ◦ 0: No ◦ 1: Yes
CFG_GATTC_SUPPORT	Support GATT Client or not. ◦ 0: No ◦ 1: Yes
CFG_CONN_AOA_AOD_SUPPORT	Support connection-based AoA/AoD or not. ◦ 0: No (default) ◦ 1: Yes

Macro	Description
	Note: The macro is configured to a fixed value of '0'.
CFG_CONNLESS_AOA_AOD_SUPPORT	Support connectionless AoA/AoD or not. ◦ 0: No ◦ 1: Yes **Note**: The macro is configured to a fixed value of '0'.
CFG_MESH_SUPPORT	Support Mesh or not. ◦ 0: No ◦ 1: Yes
SECURITY_CFG_VAL	Configure the algorithm security level. ◦ 0: Enable Level 1 algorithm. ◦ 1: Enable Level 2 algorithm.
WDT_RUN_ENABLE	Enable/Disable background running of WDT. ◦ 0: Disable ◦ 1: Enable

Note:

*: Macros marked with an asterisk (*) in the table above are used to initialize the BUILD_IN_APP_INFO structure which is defined at 0x200 in the firmware and is initialized with the macros in *custom_config.h*. When system boots, the Bootloader reads value from 0x200 and uses it as a boot parameter.

- Configure parameters in the **Configuration Wizard** interface.

  Comments in *custom_config.h* are compliant with Configuration Wizard Annotations of Keil, making it possible for developers to configure macros in *custom_config.h* in the **Configuration Wizard** interface of Keil.

Tip:

It is recommended to configure parameters in the **Configuration Wizard** interface, to prevent inputting invalid parameters.

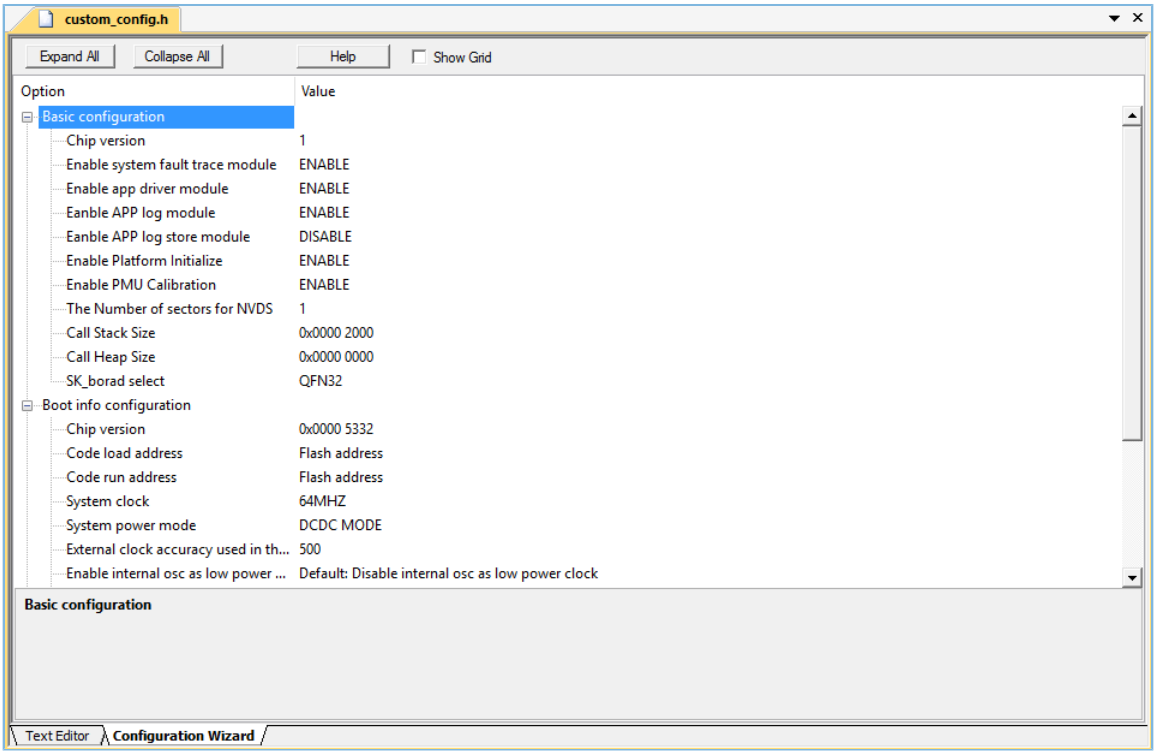

Figure 4-5 *custom_config.h* in the **Configuration Wizard** interface

4.3.2.2 Configuring Memory Layout

In a Keil project, the memory area for the linker is defined in Scatter (.sct) files. The GR5405 SDK provides an example Scatter file (SDK_`Folder\platform\soc\linker\keil\flash_scatter_common.sct`) to help developers quickly configure memory layout. The macros used by *flash_scatter_common.sct* are defined in *flash_scatter_config.h*.

Note:

In Keil, `__attribute__((section("name")))` can be used to define a function or a variable in a specific memory segment, in which **name** can be customized by developers. The scatter (*.sct*) file defines the location for customized fields. For example, to define the Zero-Initialized (ZI) data of applications in the segment named as **.bss.app**, you can set **attribute** to **attribute ((section(".bss.app ")))**.

You can follow the steps below to configure the memory layout:

1. Click (**Options for Target**) on the Keil toolbar and open the **Options for Target 'GRxx_Soc'** dialog box. Select the **Linker** tab.
2. On the **Scatter File** bar of the **Linker** tab, click **...** to browse and select the *flash_scatter_common.sct* file in SDK_`Folder\platform\soc\linker\keil`. You can also copy the scatter (.sct) file and the configuration (.h) file to the ble_app_example project directory and then select the scatter file.

Note:

- `#! armcc -E –I` in *flash_scatter_common.sct* specifies the directory of the header file on which *flash_scatter_common.sct* depends. A wrong path results in a linker error.
- In *flash_scatter_common.sct* of the GR5405 SDK, you can use the macro definition BLE_SUPPORT in *custom_config.h* to determine whether it is necessary to configure EM for Bluetooth LE in the end area of SRAM. You need to define BLE_SUPPORT based on whether the project includes Bluetooth LE services.

3. Click **Edit...** to open the .sct file, and modify corresponding code based on practical product memory layout.

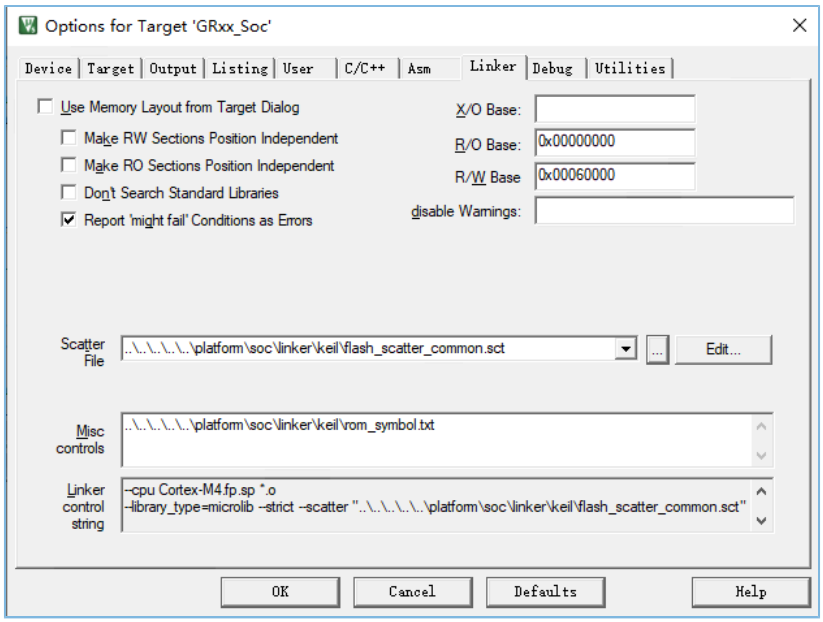

Figure 4-6 Configuration of scatter file

4. Click **OK** to save the settings.

4.3.3 Adding User Code

You can modify corresponding code in ble_app_example on demand.

4.3.3.1 Modifying the main() Function

Code of a typical *main.c* file is provided as follows:

```
/**@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(ble_evt_handler, &heaps_table);
```

```
    // loop
    while (1)
    {
        app_log_flush();
        pwr_mgmt_schedule();
    }
}
```

- `STACK_HEAP_INIT(heaps_table)` defines seven global arrays as Heaps for Bluetooth LE Stack. Do not modify the definition; otherwise, Bluetooth LE Stack may not work properly. The Heap size is determined by the Bluetooth LE service volume in "Section 4.3.2.1 Configuring custom_config.h".
- `app_periph_init()` is used to initialize peripherals. In development and debugging phases, SYS_SET_BD_ADDR in this function can be used to set a temporary Public Address; pwr_mgmt_mode_set() sets the MCU operation mode (SLEEP/IDLE/ACTIVE) during automatic power management; app_periph_init() is implemented in *user_periph_setup.c*, and the example code is as follows.

```
/**@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);
    board_init();
    pwr_mgmt_mode_set(PMR_MGMT_SLEEP_MODE);
}
```

- Add main loop code of applications to `while(1) { }`, for example, code to handle external input and update GUI.
- To use the APP LOG module, call `app_log_flush()` in the main loop, to ensure logs are output completely before the system enters sleep state. For more information about the APP LOG module, refer to "Section 4.6.3 Outputting Debug Logs".
- Call `pwr_mgmt_shcedule()` to implement automatic power management to reduce system power consumption.

4.3.3.2 Implementing Bluetooth LE Service Logics

Bluetooth LE service logics of applications are driven by a number of Bluetooth LE events which are defined in GR5405 SDK. Therefore, applications need to implement the corresponding event handlers in GR5405 SDK to obtain operation results or state change notifications of Bluetooth LE Stack. The event handlers are called in the interrupt context of Bluetooth LE SDK IRQ. Therefore, do not perform long-running operations in handlers, for example, blocking function call and infinite loop; otherwise, the system is blocked, causing Bluetooth LE Stack and the SDK Bluetooth LE module unable to run in a normal timing.

Bluetooth LE events fall into eight categories: Common, GAP Management, GAP Connection Control, Security Manager, L2CAP, GATT Common, GATT Server, and GATT Client.

Note:

The Bluetooth LE events supported by GR5405 SDK can be found in SDK_Folder\components\sdk\ble_event.h.

You need to implement necessary Bluetooth LE event handlers according to functional requirements of your products. For example, if a product does not support Security Manager, you do not need to implement corresponding events; if the product supports GATT Server only, you do not need to implement the events corresponding to GATT Client. Only those event handlers required for products are to be implemented.

Tip:

For details about the usage of Bluetooth LE APIs and event APIs, refer to the source code of Bluetooth LE examples in SDK_Folder\documentation\GR5405_API_Reference and SDK_Folder\projects\ble.

4.3.3.3 Scheduling BLE_Stack_IRQ, BLE_SDK_IRQ, and Applications

Bluetooth LE Stack is the core to implement Bluetooth LE protocols. It can directly operate the Bluetooth 5.3 Core (refer to "Section 2.2 Software Architecture"). Therefore, BLE_Stack_IRQ has the second-highest priority after SVCall IRQ, which ensures that Bluetooth LE Stack runs strictly in a timing specified in *Bluetooth Core Spec*.

A state change of Bluetooth LE Stack triggers the BLE_SDK_IRQ interrupt with lower priority. In this interrupt handler, the Bluetooth LE event handlers (to be executed in applications) are called to send state change notifications of Bluetooth LE Stack and related service data to applications. Avoid time-consuming operations when using these event handlers. Perform such operations in the main loop or in user-level threads instead. You can use the module in SDK_Folder\components\libraries\app_queue, or your own application framework, to transfer events from Bluetooth LE event handlers to the main loop.

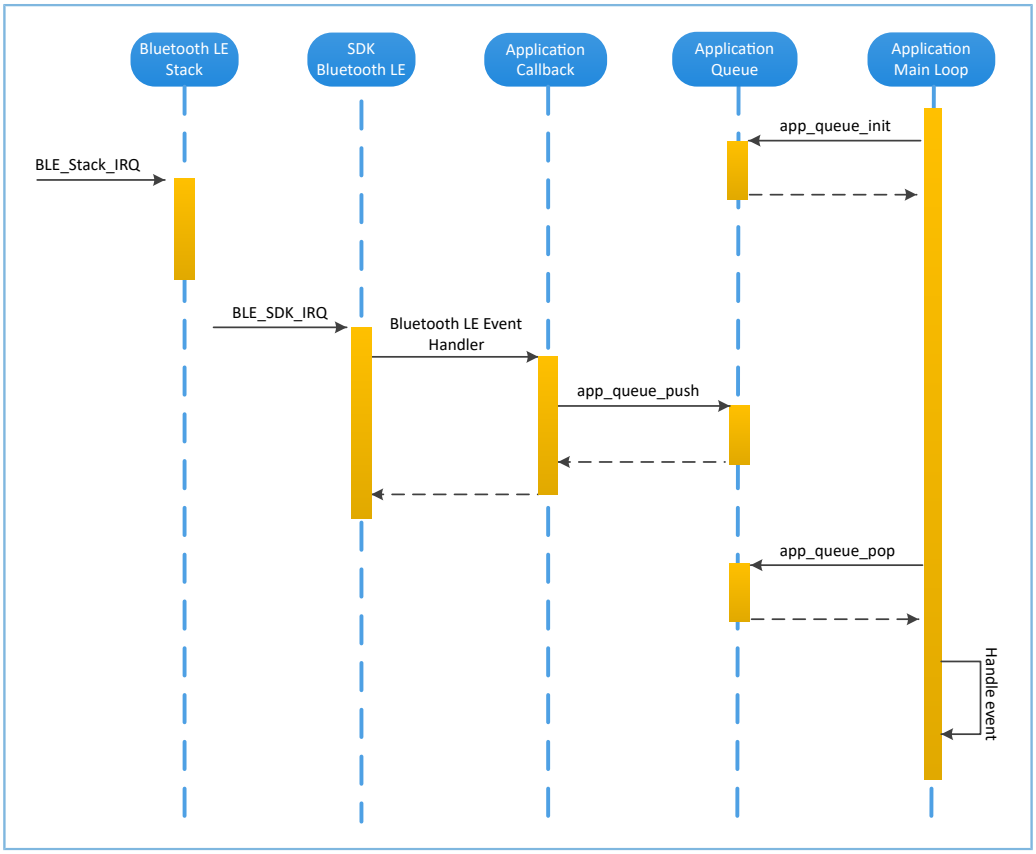

Figure 4-7 System schedule (without OS)

4.4 Generating Firmware

After building a Bluetooth LE application, you can directly click (**Build**) on the Keil toolbar to build a project.

After project compilation is completed, two firmware files (in .bin and .hex formats) are created in `Listings` and `Objects` respectively in the project directory.

Table 4-5 Firmware files generated

Name	Description
ble_app_example.bin	Binary application firmware, can be downloaded to Flash through GProgrammer for running
ble_app_example.hex	Binary application firmware, can be downloaded to Flash through Keil or GProgrammer for running

Tip:

Both the two types of firmware can be downloaded to Flash through GProgrammer for running. Refer to *GProgrammer User Manual* for details.

4.5 Downloading .hex Files to Flash

After a firmware file is are generated, you need to download the file to Flash. Specific steps are provided below:

1. Configure Keil Flash programming algorithm.

(1) Copy `SDK_Folder\build\Keil\GR5xxx_16MB_Flash.FLM` to `Keil_Folder\ARM\Flash`.

(2) Click (**Options for Target**) on the Keil toolbar, open the **Options for Target 'GRxx_Soc'** dialog box, and select the **Debug** tab. Click **Settings** on the right side of **Use: J-LINK/J-TRACE Cortex**.

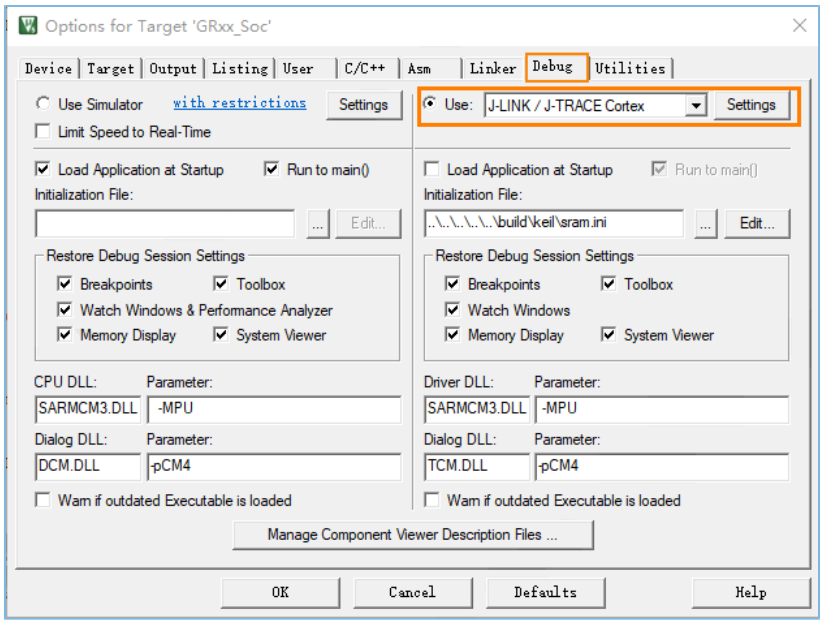

Figure 4-8 **Debug** tab

(3) In the **Cortex JLink/JTrace Target Driver Setup** window, select **Flash Download**. In the **Download Function** pane, you can set the erase type and check optional items: **Program**, **Verify**, and **Reset and Run**. Default configurations of Keil are shown below:

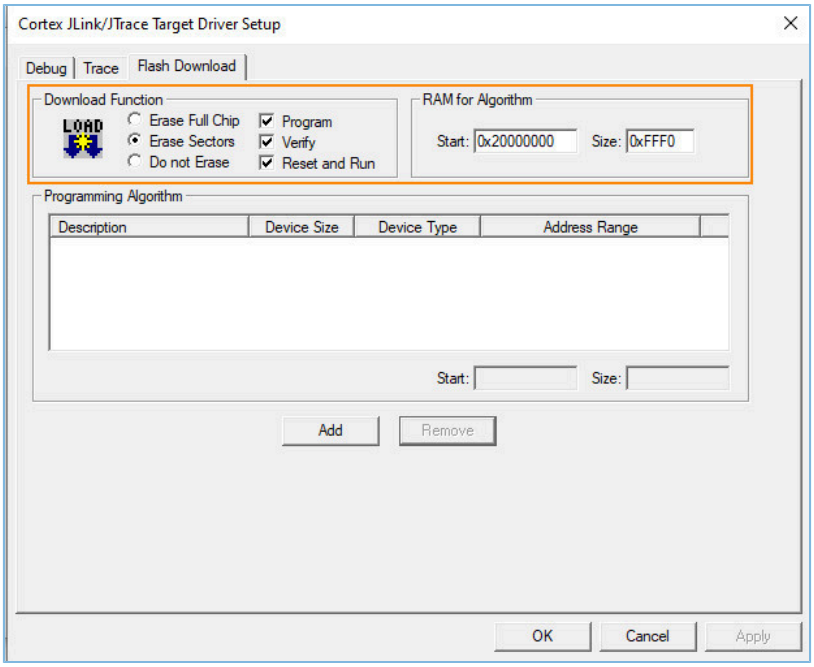

Figure 4-9 Default configurations in the **Download Function** pane

(4) Click **Add** to add SDK_Folder\build\keil\GR5xxx_16MB_Flash.FLM to **Programming Algorithm**.

Note:

To facilitate multi-chip inheritance development for users, *GR5xxx_16MB_Flash.FLM* is used for all the Goodix GR5xx Bluetooth LE SoC series which share the same download algorithm.

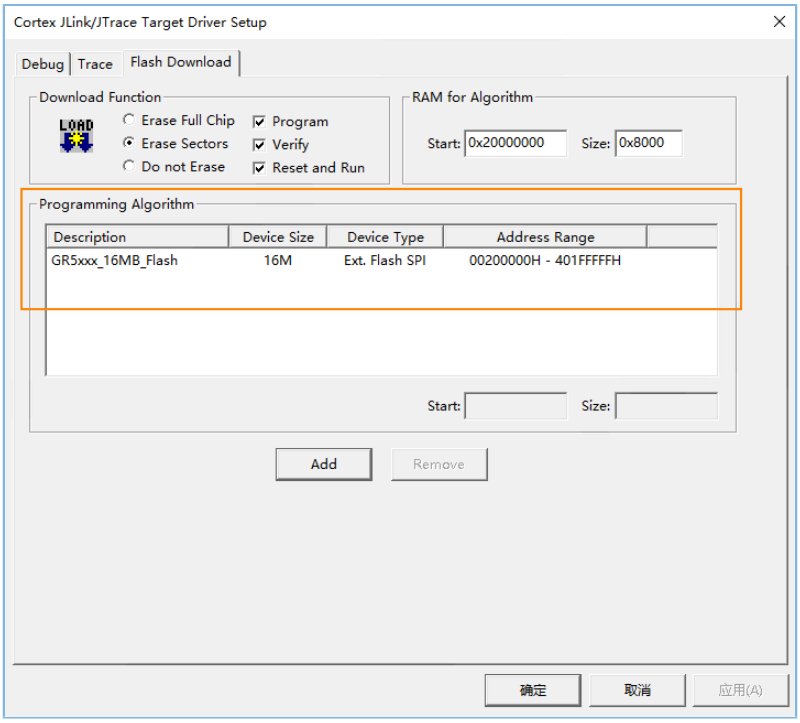

Figure 4-10 Adding *GR5xxx_16MB_Flash.FLM* to **Programming Algorithm**

(5) Configure **RAM for Algorithm**, which defines address space to load and implement the programming algorithm. Enter the start address of RAM in GR5405 in the **Start** input field: **0x20000000**. Enter **0x8000** in the **Size** input field.

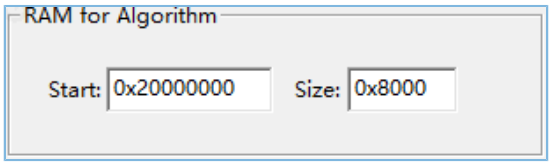

Figure 4-11 Settings of **RAM for Algorithm**

(6) Click **OK** to save the settings.

2. Download firmware.

After completing configuration, click (**Download**) on the Keil toolbar to download *ble_app_example.axf* to Flash. After download is completed, the following results are displayed in the **Build Output** window of Keil.

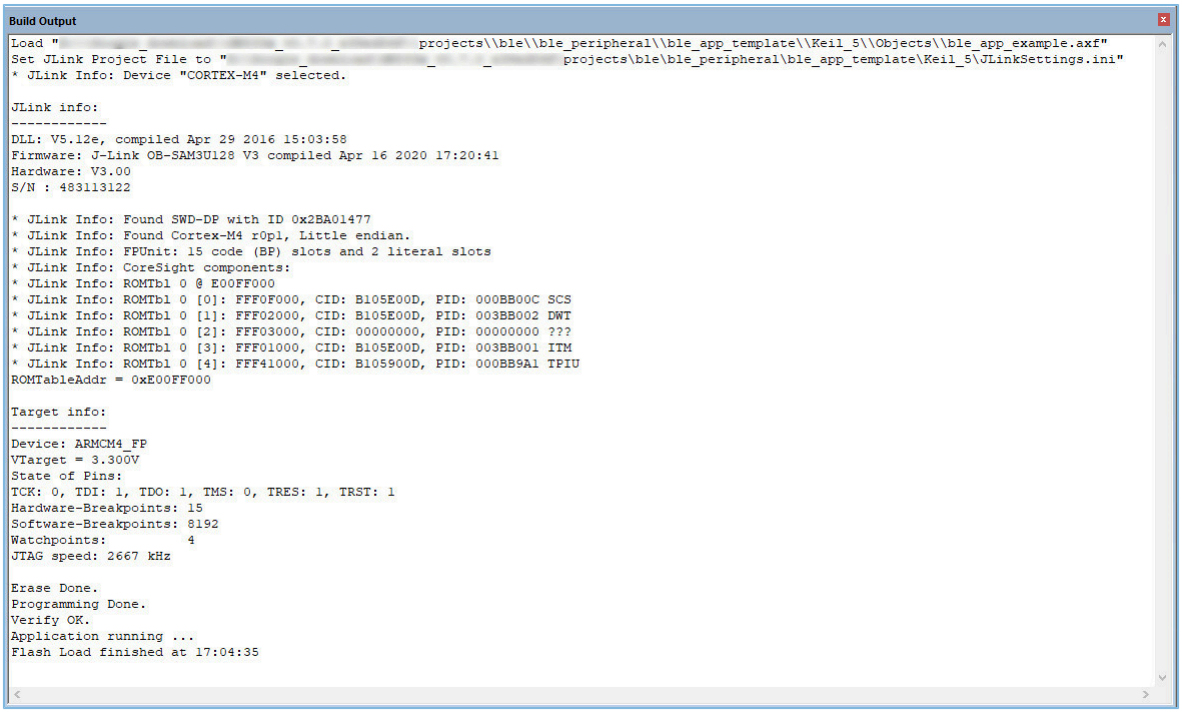

Figure 4-12 Download results

Note:

During file download, if **No Cortex-M SW Device Found** pops up, it indicates the SoC may be in sleep state at that moment (the firmware with sleep mode enabled is running), so the .hex file cannot be downloaded to Flash. In this case, developers need to press **RESET** on the GR5405 SK Board and wait for about 1 second; then click (**Download**) to download the file again.

4.6 Debugging

Keil provides a debugger for online code debugging. The debugger supports setting six hardware breakpoints and multiple software breakpoints. It also provides developers with multiple methods to set debug commands.

4.6.1 Configuring the Debugger

Configure the debugger before debugging. Click (**Options for Target**) on the Keil toolbar, open the **Options for Target 'GRxx_Soc'** dialog box, and select the **Debug** tab. In the window, software simulation debugging configurations display on the left side, and online hardware debugging configurations display on the right side.

Bluetooth LE example projects adopt the online hardware debugging. Related default configurations of the debugger are shown as follows:

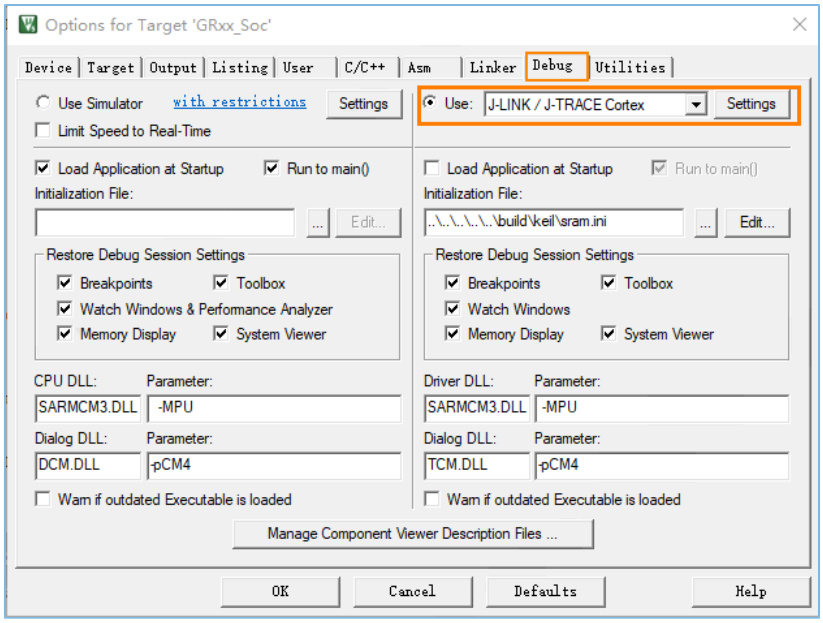

Figure 4-13 Configuring the debugger

The default initialization file *sram.ini* is in SDK_`Folder\build\keil`. You can use this file directly, or copy it to the project directory.

The initialization file *sram.ini* contains a set of debug commands, which are executed during debugging. On the **Initialization File** bar, click **Edit...** on the right side, to open *sram.ini*.

Example code of *sram.ini* is provided as follows:

```
/**
*************************************************************************
*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 current 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)

//_WDWORD(0xE000E180, 0xFFFFFFFF)

//Write run address to 0xA000C578 register,For the debug mode;
```

```
//boot code will check the value of 0xA000C578 firstly,if the value of 0xA000C578 is
 valid,gr551x will jump to run

//_WDWORD(0xA000C578, 0x00810000)
```

Note:

Keil supports executing debugger commands set by developers in the following order:

1. When **Options for Target 'GRxx_Soc'** > **Debug** > **Load Application at Startup** is enabled, the debugger first loads the file under **Options for Target 'GRxx_Soc'** > **Output** > **Name of Executable**.
2. Execute the command in the file specified in **Options for Target 'GRxx_Soc'** > **Debug** > **Initialization File**.
3. When options under **Options for Target 'GRxx_Soc'** > **Debug** > **Restore Debug Session Settings** are checked, restore corresponding Breakpoints, Watch Windows, Memory Display, and other settings.
4. When **Options for Target 'GRxx_Soc'** > **Debug** > **Run to main()** is checked, or the command `g,main` is discovered in **Initialization File**, the debugger automatically starts executing CPU commands, until running to the main() function.

4.6.2 Starting Debugging

After completing debugger configuration, click (**Start/Stop Debug Session**) on the Keil toolbar, to start debugging.

Note:

Make sure that **Connect** is set to **Normal** and **Reset Options** is set to **Core**, as shown below. This is to ensure when you click **Reset** on the Keil toolbar after enabling **Debug Session**, the program can run normally.

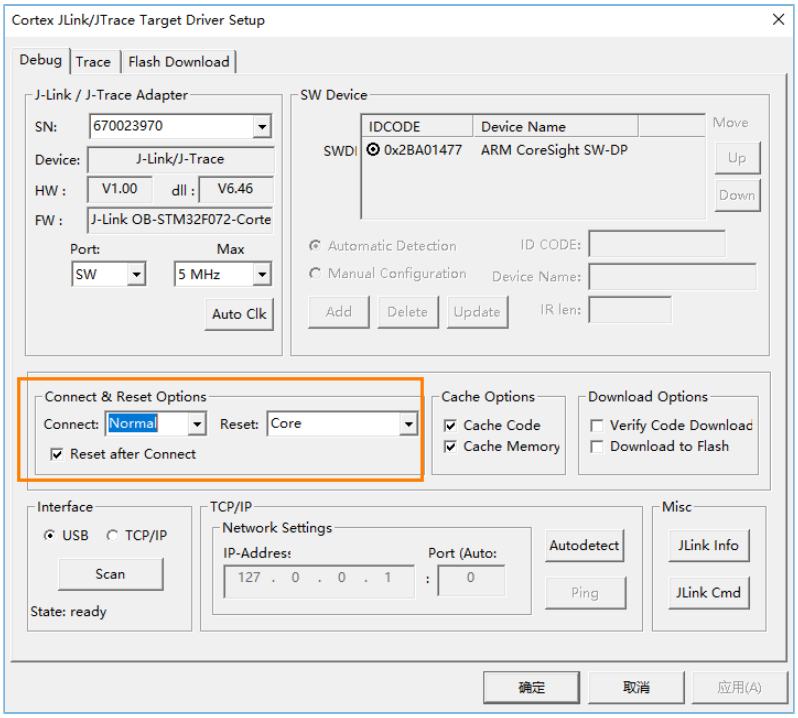

Figure 4-14 Setting **Connect** and **Reset**

4.6.3 Outputting Debug Logs

GR5405 SDK provides an APP LOG module and supports outputting debug logs of applications from hardware ports based on customization. Hardware ports include UART, J-Link RTT, and ARM Instrumentation Trace Macrocell (ARM ITM).

To use the APP LOG module, enable APP_LOG_ENABLE in *custom_config.h*, and configure APP_LOG_PORT based on the output method as needed.

4.6.3.1 Module Initialization

After configuration, you need to call app_log_init() during peripheral initialization to initialize the APP LOG module, including setting log parameters, and registering log output APIs and flush APIs.

The APP LOG module supports using `printf()` (a C standard library function) and APP LOG APIs to output debug logs. If you choose APP LOG APIs, you can optimize logs by setting log level, log format, filter type, or other parameters; if you choose `printf()`, set log parameters as NULL.

Call the initialization function of corresponding module (refer to SDK_`Folder\platform\boards\board_SK.c` for details) and register corresponding log output and flush APIs (see bsp_log_init() for reference) according to the configured output port.

If UART is the output port, bsp_log_init() is implemented as follows:

```
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_NO_BLOCK_TRIM);
#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;
#ifdef APP_LOG_NO_PFX
    log_init.fmt_set[APP_LOG_LVL_INFO]     = APP_LOG_FMT_NULL;
    log_init.fmt_set[APP_LOG_LVL_DEBUG]    = APP_LOG_FMT_NULL;
#else
    log_init.fmt_set[APP_LOG_LVL_INFO]     = APP_LOG_FMT_LVL;
    log_init.fmt_set[APP_LOG_LVL_DEBUG]    = APP_LOG_FMT_LVL;
#endif

#if (APP_LOG_PORT == 0)
    app_log_init(&log_init, bsp_uart_send, bsp_uart_flush);
    app_log_assert_flush_init(bsp_uart_assert_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 /* APP_LOG_PORT */

#endif /* APP_LOG_PORT <= 2 */

#endif /* APP_LOG_ENABLE == 1 */

    app_assert_init();
}
```

Note:

- The input parameters of app_log_init() include the log initialization parameter, log output API, and flush API.
- The input parameter of app_log_assert_flush_init() is the flush API designed for the APP ASSERT module. The implementation of the APP ASSERT module is in SDK_`Folder\components\libraries\app_assert`.

When debug logs are output through UART, the implemented log output API and flush API are bsp_uart_send() and bsp_uart_flush() respectively.

- bsp_uart_send() is the basis for two log output APIs: app_uart synchronization (app_uart_transmit_sync) and app_uart asynchronization (app_uart_transmit_async). Users can control the APIs by adding or removing the macro APP_LOG_ASYNC. Users can choose the output methods as needed.
- bsp_uart_flush() is used to output the log data that is cached in memory in app_uart asynchronization method.
- bsp_uart_assert_flush() is used to output the log data that is cached in memory when an assertion occurs.

Note:

You can rewrite the above three APIs.

When debug logs are output through J-Link RTT or ARM ITM, the implemented log output API is bsp_segger_rtt_send() or bsp_itm_send(). No flush API is to be implemented in the two modes.

4.6.3.2 Application

After completing initialization of the APP LOG module, you can use any of the following four APIs to output debug logs:

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

In interrupt output mode, call app_log_flush() to output all the debug logs cached, to ensure that all debug logs are output before the SoC is reset or the system enters the sleep mode.

To output logs through J-Link RTT, it is recommended to make the following modifications in *SEGGER_RTT.c*:

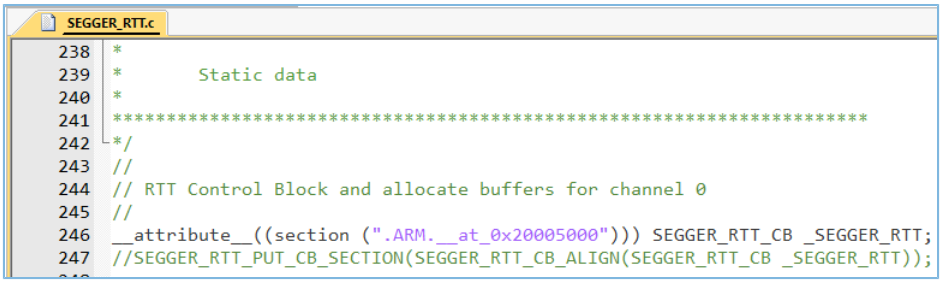

Figure 4-15 Creating RTT Control Block and placing it at 0x20005000

The figure below shows the reference configurations for J-Link RTT Viewer.

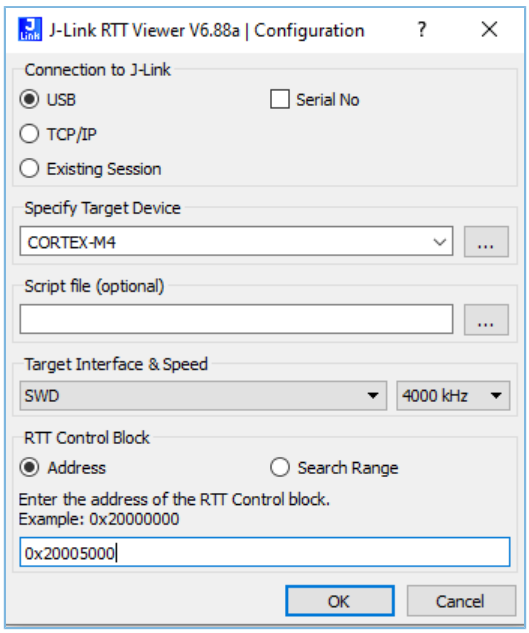

Figure 4-16 Configurations in J-Link RTT Viewer

The address of **RTT Control Block** can be specified by clicking **Address** and then entering a custom value, and the input value can be set to the address of the **_SEGGER_RTT** structure in the .map file generated by the compiled project, as shown in the figure below. If creating RTT Control Block through the method recommended in Figure 4-15 and placing it at 0x20005000, you need to set **Address** to **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)
```

Figure 4-17 Obtaining RTT Control Block address

4.6.4 Debugging with GRToolbox

GR5405 SDK provides an Android App, GRToolbox, to debug GR5405 Bluetooth LE applications. GRToolbox features the following:

- General Bluetooth LE scanning and connecting; characteristics read/write
- Demos for standard profiles, including Heart Rate
- Goodix-customized applications

Tip:

You can obtain the GRToolbox installation file from Goodix official website or download it from the application store.

4.7 Download and Debugging with IAR

GR5405 SDK provides not only Keil projects, but also IAR projects, allowing users to quickly compile an example project with IAR to generate application firmware. For example, the IAR project of ble_app_template is in `SDK_Folder\projects\ble\ble_peripheral\ble_app_template\IAR`; compile *ble_app_template.eww* to generate *ble_app_template.bin*.

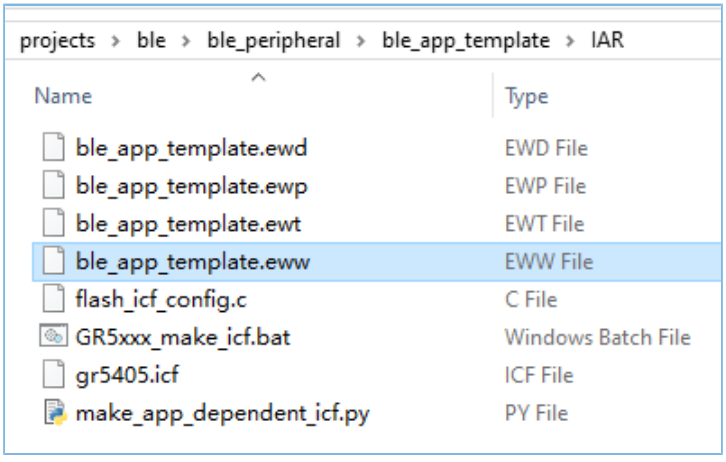

Figure 4-18 IAR project directory

Steps for firmware download and debugging with IAR are as follows:

1. On the menu bar of IAR IDE, select **Project** > **Options** to open the configuration window; select **Debugger** from the **Category** list on the left, and then select **J-Link/J-Trace** in the **Driver** pane of the **Setup** tab. Check **Use macro file(s)** and enter **$PROJ_DIR$\..\..\..\..\..\build\iar\GR5xxx.mac**.
2. Copy the download algorithm files *GR5xxx_IAR_16M.board*, *GR5xxx_IAR_16M.flash*, and *GR5xxx_IAR_flashloader_16M.out* in SDK_`Folder/build/iar` to `IAR_Install/arm/config/flashloader/Goodix` ("IAR_Install" is the IAR software installation directory, and "Goodix" is a new folder).
3. In IAR IDE, select **Project** > **Options** > **Debugger** > **Download**, to set *Override default .board file* as *GR5xxx_IAR_16M.board* (in `IAR_Install/arm/config/flashloader/Goodix`).
4. In IAR IDE, select **Project** > **Options** > **Debugger** > **J-Link/J-Trace** > **Setup**, and then select **Core** in the **Reset** pane.
5. On the menu bar of IAR IDE, select **Project** > **Download** > **Download Active Application**, and then click **OK** in the pop-up dialog box to start firmware download, and then enter debug mode.

5 Glossary

Table 5-1 Glossary

Name	Description
API	Application Programming Interface
ATT	Attribute Protocol
Bluetooth LE	Bluetooth Low Energy
DFU	Device Firmware Update
DTM	Direct Test Mode
DUT	Device Under Test
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GFSK	Gaussian Frequency Shift Keying
HAL	Hardware Abstract Layer
HCI	Host Controller Interface
HPA	High Power Amplifier
L2CAP	Logical Link Control and Adaptation Protocol
LL	Link Layer
NVDS	Non-volatile Data Storage
PMU	Power Management Unit
PHY	Physical Layer
RF	Radio Frequency
SCA	System Configuration Area
SDK	Software Development Kit
SM	Security Manager
SoC	System-on-Chip
SPA	Small Power Amplifier
UART	Universal Asynchronous Receiver/Transmitter
XIP	Execute in Place