GR5526 Developer Guide

Version: 1.2

Release Date: 2025-07-11

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 GR5526 Bluetooth Low Energy (Bluetooth LE) System-on-Chip (SoC) and Keil for program development and debugging, to help you quickly get started with secondary development of Bluetooth LE applications.

Audience

This document is intended for:

- Device user
- Developer
- Test engineer
- Technical writer

Release Notes

This document is the third release of *GR5526 Developer Guide*, corresponding to GR5526 SoC series.

Revision History

Version	Date	Description
1.0	2023-01-10	Initial release
1.1	2025-06-06	Updated the sections "GR5526 SDK Directory Structure", "Configuring custom_config.h", and "Debugging with GRToolbox".
1.2	2025-07-11	• Updated the RAM layout in XIP/mirror mode. • Added a note about the Bluetooth LE events supported by GR5526 SDK in **Implementing Bluetooth LE Service Logic**.

Contents

1 Introduction

The Goodix GR5526 family is a low-power System-on-Chip (SoC) that supports Bluetooth 5.3. It supports Bluetooth Low Energy (Bluetooth LE) direction finding: angle of arrival (AoA) and angle of departure (AoD), as well as LE isochronous channels (LE Audio), making it an ideal choice for Internet of Things (IoT), LE Audio, and smart wearable devices.

Based on ARM® Cortex®-M4F CPU core, the GR5526 series integrates Bluetooth 5.3 Protocol Stack, a 2.4 GHz RF transceiver, on-chip programmable Flash memory, RAM, multiple peripherals, enhanced I2C/UART port number for sensor applications, as well as expanded I/O functionality. GR5526 delivers a feature-rich display and graphics solution by providing the option of graphics acceleration (GPU + DC) and system-in-package (SiP) pseudostatic RAM (PSRAM) to accommodate display while still leaving plenty of resource for wearable schemes.

GR5526 series 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.

GR5526 series comes in two package choices: BGA83 and QFN68. The detailed configurations are listed below

Table 1-1 GR5526 series

GR5526 Series	GR5526VGBIP	GR5526VGBI	GR5526RGNIP	GR5526RGNI
CPU	Cortex®-M4F	Cortex®-M4F	Cortex®-M4F	Cortex®-M4F
RAM	512 KB	512 KB	512 KB	512 KB
SiP Flash	1 MB	1 MB	1 MB	1 MB
SiP PSRAM	8 MB	N/A	8 MB	N/A
GPU + DC	Yes	N/A	Yes	N/A
I/O Number	50	50	48	48
Package (mm)	BGA83 (4.3 x 4.3 x 0.96)	BGA83 (4.3 x 4.3 x 0.96)	QFN68 (7.0 x 7.0 x 0.85)	QFN68 (7.0 x 7.0 x 0.85)

1.1 GR5526 SDK

The GR5526 Software Development Kit (SDK) provides comprehensive software development support for GR5526 SoCs. The SDK contains Bluetooth LE APIs, System APIs, Real Time Location Service (RTLS) APIs, peripheral drivers, a tool for debugging and download, project example code, and related user documents.

The GR5526 SDK version mentioned in this document is applicable to all GR5526 SoCs.

1.2 Bluetooth LE Stack

The architecture of Bluetooth LE Stack is shown in Figure 1-1.

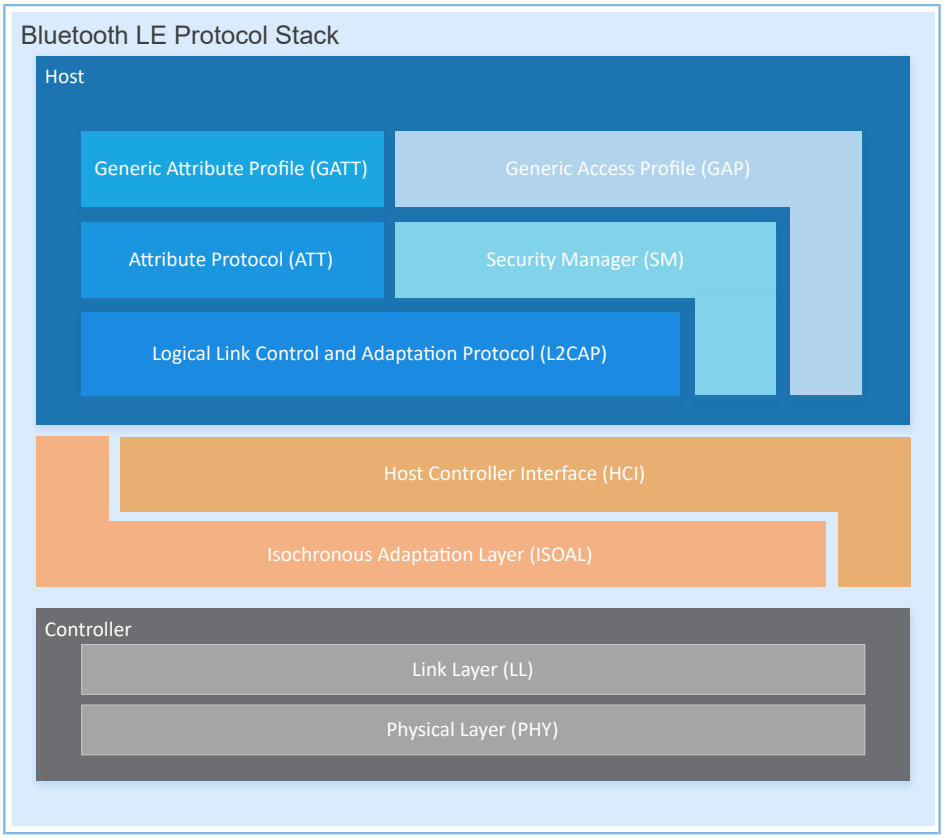


Figure 1-1 Bluetooth LE Stack architecture

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

Controller

- Physical Layer (PHY): Supports 1-Mbps and 2-Mbps 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 modes, and can be switched between the modes on demand: Standby, Advertising, Scanning, Initiating, or Connection.

ISOAL

- ISOAL: Enables adaptation of isochronous data between the Host and the Controller by assembling segmented data frames into data streams for the Application layer, or by segmenting data streams from the Application layer into data frames and transmitting the data frames through air interfaces.

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, 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, which fulfills 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 Bluetooth SIG official website.
- 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 GR5526 Bluetooth LE Software Platform

The GR5526 SDK is designed for GR5526 SoCs, to help users develop Bluetooth LE applications. It integrates Bluetooth 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 GR5526 SDK.

2.1 Hardware Architecture

The GR5526 hardware architecture is shown as follows.

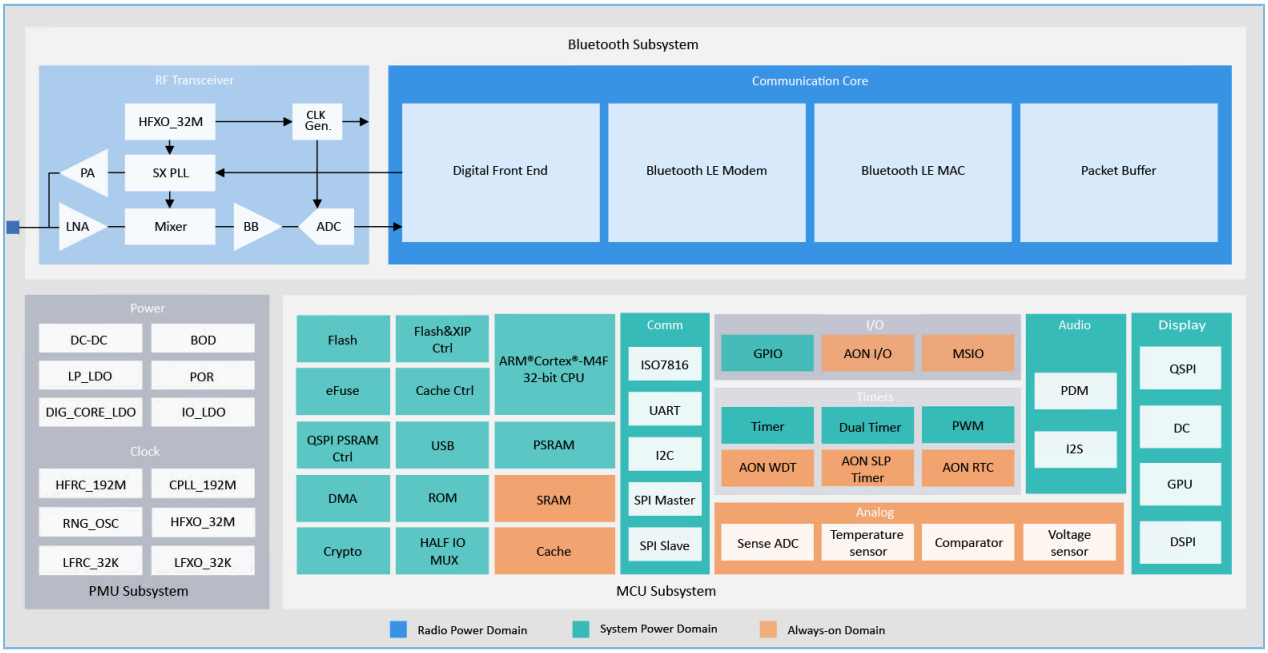


Figure 2-1 GR5526 hardware architecture

- ARM® Cortex®-M4F: GR5526 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.
- Security Cores: the secure computing engine unit, mainly including TRNG, AES, SHA, and PKC modules, which allows checking encrypted user application firmware. Check on encrypted firmware relies on the secure boot process launched in ROM. (In *Bluetooth Core Spec*, the secure computing unit is an independent module in Communication Core, and is irrelevant to Security Cores.)
- Peripherals: including GPIO, DMA, I2C, I2S, SPI, QSPI, DSPI, OSPI, UART, PWM, Timer, GPU, and DC
- RF Transceiver: 2.4 GHz RF signal 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, IO-LDO, Dig-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 information about the modules in a GR5526 SoC, see *GR5526 Datasheet*.

2.2 Software Architecture

The software architecture of the GR5526 SDK is shown as follows.

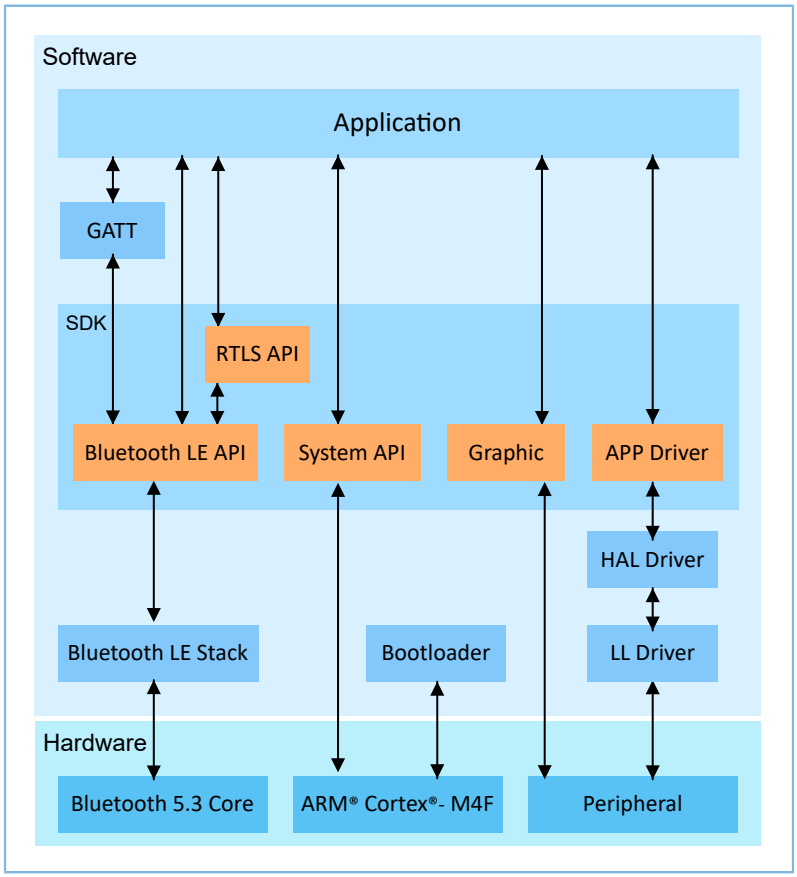


Figure 2-2 GR5526 software architecture

- Bootloader

  A boot program built in GR5526 SoCs, used for GR5526 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, ISOAL, HCI, and Host protocols (including ATT, L2CAP, GAP, SM, and GATT), and supports roles of Broadcaster, Observer, Peripheral, and Central.

- 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.

- LL Driver

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

- Bluetooth LE SDK

  SDK that provides easy-to-use Bluetooth LE APIs, System APIs, and RTLS APIs

  - Bluetooth LE APIs: Include L2CAP, GAP, SM, GATT APIs, and LE Audio APIs.
  - System APIs: Provide Non-volatile Data Storage (NVDS), Device Firmware Update (DFU), system power management, and generic system-level access interfaces.
  - APP Driver APIs: Provide definitions for APIs of common peripherals such as UART, I2C, and ADC. APP Driver APIs call HAL/LL APIs to enable the corresponding functionalities.
  - RTLS APIs: The APIs are used for AoA and AoD functionalities. They are at the intermediate layer between the application layer and Bluetooth LE APIs. In their upward communication with the application layer, accepting configurations on locating parameters, receiving commands on locating, reporting raw data about locating, and reporting calculated results of locating are made possible. In the downward communication with Bluetooth LE APIs, the APIs obtain Iq data report events. In addition, the RTLS APIs are related to other algorithms including music algorithms and density-based spatial clustering of applications with noise (DBSCAN).

- Application

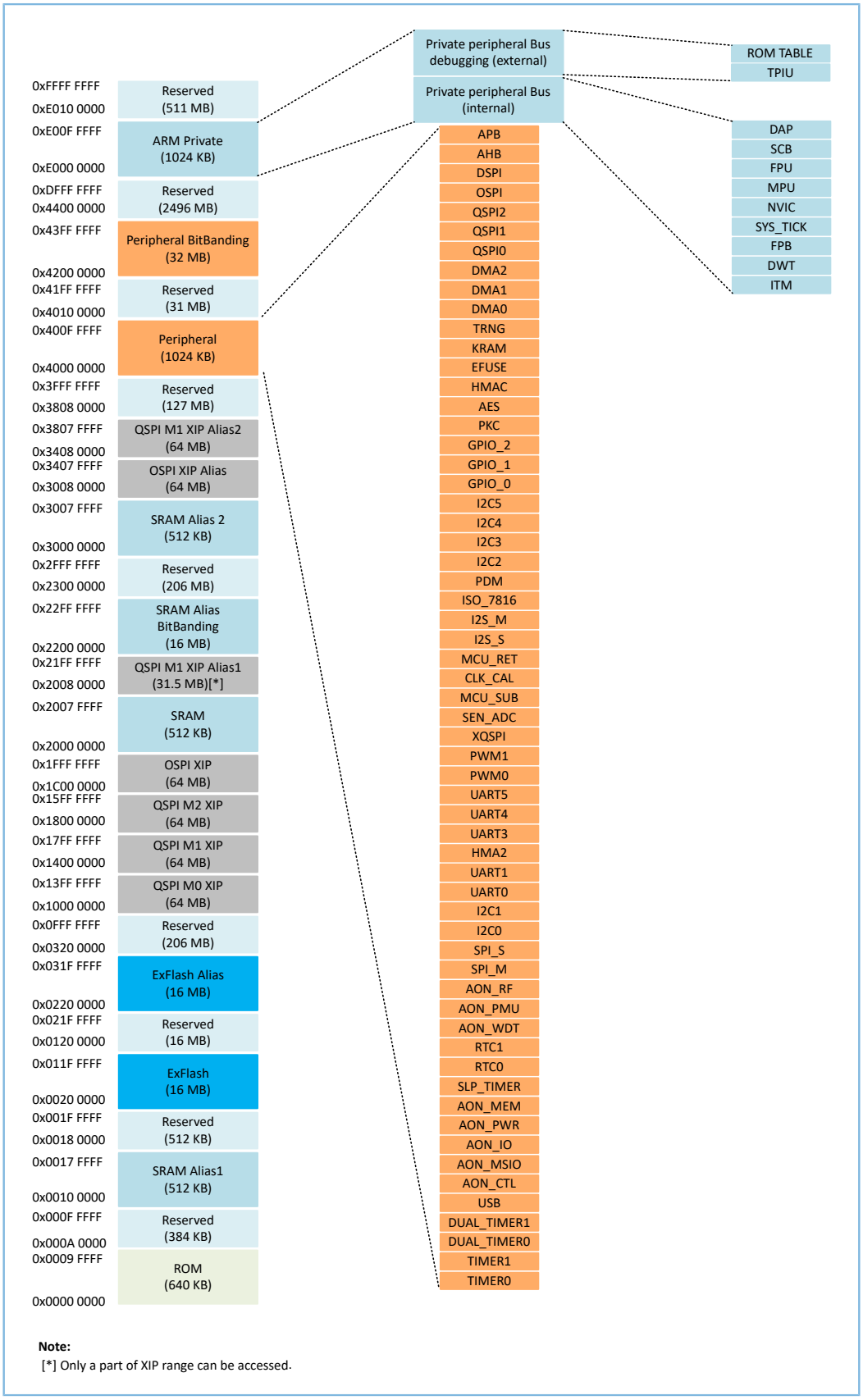
  The SDK provides abundant Bluetooth and peripheral example projects. Each project contains compiled binary files; you can download these files to GR5526 SoCs for operation and test. GRToolbox (Android) in the SDK provides rich functionalities to allow users to test most Bluetooth applications with ease.

- Graphic

  An SDK library for the graphic processing unit (GPU) display driver module

2.3 Memory Mapping

The memory mapping of a GR5526 SoC is shown as follows.



Note:
[*] Only a part of XIP range can be accessed.

Figure 2-3 GR5526 memory mapping

- RAM: 0x0010_0000 to 0x0017_FFFF, 0x2000_0000 to 0x2007_FFFF, or 0x3000_0000 to 0x3007_FFFF, 512 KB in total
  - 0x2000_0000 to 0x2007_FFFF: bit field operations supported, mapping to the region from 0x2200_0000 to 0x2207_FFFF, in which atomic operations are supported. Variables of the SDK including RW, ZI, HEAP, and STACK are in this range.
  - 0x0010_0000 to 0x0017_FFFF: Features higher access efficiency thanks to the Cortex-M4F architecture. Therefore, executable code RAM_CODE is in this region.

Note:

QSPI0, QSPI1, QSPI2, and OSPI support XIP mode, which allows mapping of address from Flash or PSRAM to memories, enabling direct operations on memory.

- When an external PSRAM is used, the PSRAM is mounted to QSPI1, forming contiguous SRAM space with the region from 0x2000 0000 to 0x2007 FFFF.
- When an internal PSRAM is used, the PSRAM is mounted to OSPI, forming contiguous SRAM space with the region from 0x3000 0000 to 0x3007 FFFF.

- Flash: 0x0020_0000 to 0x011F_FFFF or 0x0220_0000 to 0x031F_FFFF, 16 MB in total
  - 0x0020_0000 to 0x011F_FFFF: Stores code and unencrypted data.
  - 0x0220_0000 to 0x031F_FFFF: Stores encrypted data.

Note:

Internal Flash of GR5526 SoCs is 1 MB, from 0x0020_0000 to 0x002F_FFFF.

2.4 Flash Memory Mapping

GR5526 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 of typical GR5526 application scenarios is as shown in Figure 2-4.

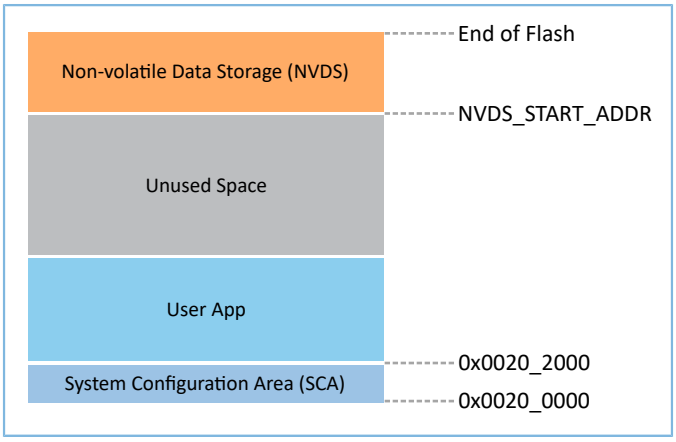


Figure 2-4 Flash memory layout

- System Configuration Area (SCA): an area to store information including system boot parameter configurations
- 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.
- NVDS: Non-volatile Data Storage area

Note:

By default, NVDS occupies the last two sectors of Flash memory. 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, see "Section 4.3.2.1 Configuring custom_config.h".

The start address of NVDS shall be aligned with that of the Flash sectors.

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 an SCA image file based on BUILD_IN_APP_INFO in the application firmware, and program the image info (stored in SCA) to Flash along with 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:

Note:

The BUILD_IN_APP_INFO structure is in SDK_Folder\platform\soc\common\gr_platform.c, and SDK_Folder is the root directory of GR5526 SDK.

```
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: 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 the value 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 as follows.

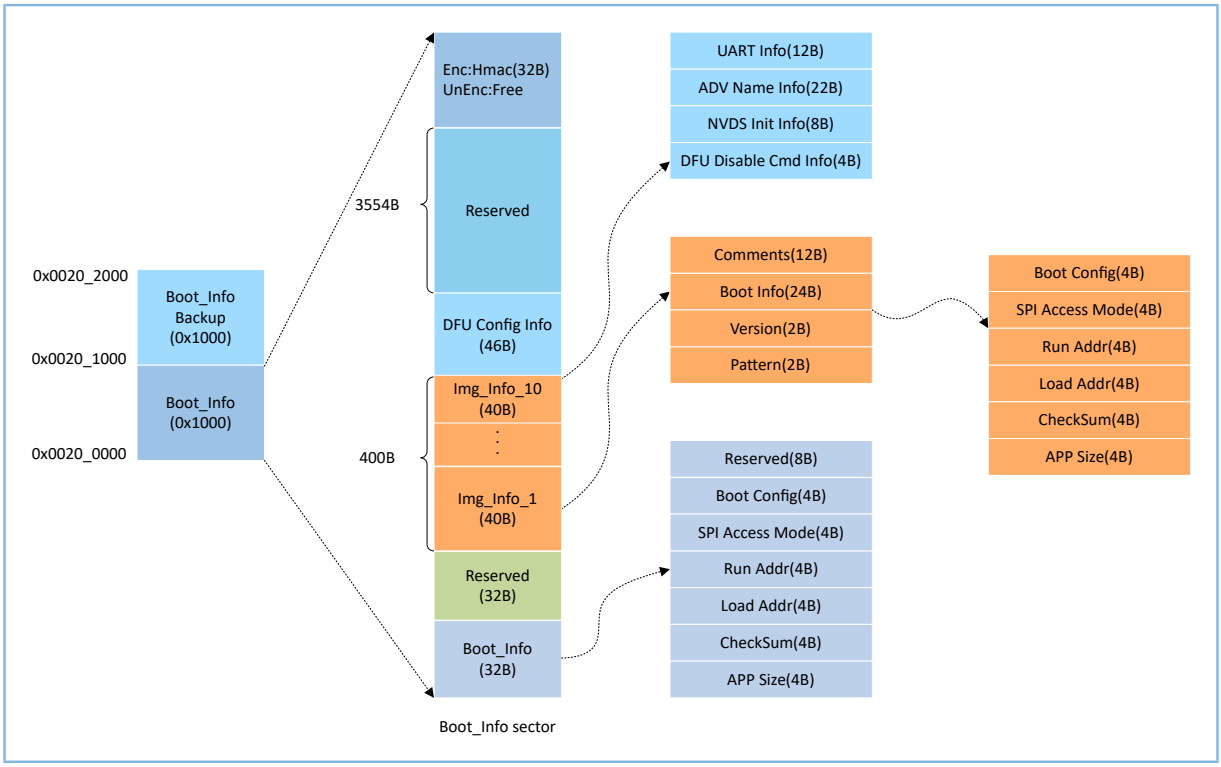


Figure 2-5 SCA layout

- Boot_Info and Boot_Info Backup store the same information. The latter is the backup of the Boot_Info.
  - In non-security mode, the Bootloader obtains boot information from Boot_Info by default.
  - In security mode, the Bootloader checks Boot_Info first; if the check fails, the Bootloader checks Boot_Info Backup and obtains boot information from it.
- 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 are 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 *custom_config.h*.
  - Pattern: This area stores a fixed value 0x4744.
- The DFU Config Info area stores configurations of DFU module in ROM.
  - UART Info: This area stores UART configurations of DFU module, including status bit, baud rate, and GPIO configurations.
  - ADV Name Info: This area stores advertising configurations of DFU module, including status bit, advertising name, and advertising length.
  - NVDS Init Info: This area stores initialization configurations of NVDS system in DFU module, including status bit, NVDS area size, and start address.
  - DFU Disable Cmd Info: This area stores DFU disable command configurations of DFU module, including status bit and Disable DFU Cmd (2 B, set as Bitmask). You can set the Disable DFU Cmd value to disable a DFU command.
- The HMAC area stores the HMAC check value. This area is valid only in security mode.

2.4.2 NVDS

NVDS is a lightweight logical data storage system based on Flash HAL. NVDS is located in the Flash memory and data in it will not be lost in power-off status. By default, NVDS uses the last two sectors of the Flash memory. You can also specify the number of Flash sectors to be occupied. In NVDS, the last sector is for defragmentation, and the other sector for data storage.

NVDS is an ideal choice to store small data blocks, for example, application configuration parameters, calibration data, states, and user information. Bluetooth LE Stack stores parameters such as device bonding parameters in NVDS.

NVDS features:

- Each storage item (TAG) has a unique TAG ID. User applications can read and change data according to TAG IDs, regardless of the physical addresses for data storage.
- It is optimized based on medium characteristics of Flash memory and supports data check, word alignment, defragmentation, and erase/write balance.
- The size and start address of NVDS are configurable. Compared with Flash memory which is made up of 4 KB sectors, NVDS can be in several sectors as configured. Make sure the start address of NVDS is 4 KB aligned.

Note:

- You can add NVDS_START_ADDR to and modify NVDS_NUM_SECTOR in *custom_config.h*, to configure the start address and the size of NVDS.
- Bluetooth LE Stack and applications share the same NVDS storage area. However, TAG ID namespace is divided into different categories. You can only use the TAG ID name category assigned to applications.
  - Applications have to use NV_TAG_APP(idx) to obtain the TAG ID of application data. The TAG ID is used as an NVDS API parameter.
  - Applications cannot use idx as the NVDS API parameter directly. The idx value ranges from 0x4000 to 0x7FFF.
- Before running an application for the first time, you can use GProgrammer to write the initial TAG ID value used by Bluetooth LE Stack and the application to NVDS.
- If you specify an NVDS area, instead of using the default NVDS area in the GR5526 SDK, make sure the start address configured in GProgrammer is 4 KB aligned.

Data stored in NVDS is in the format below.

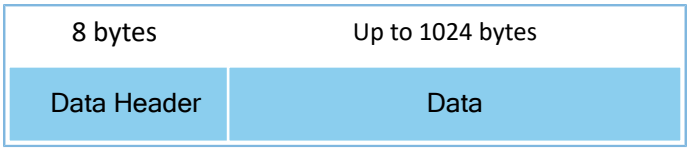


Figure 2-6 Data format in NVDS

Details of data header are described below.

Table 2-1 Data header format

Byte	Name	Description
0–1	tag	Data tag
2–3	len	Data length
4–4	checksum	Checksum of data header
5–5	value_cs	Checksum of data
6–7	reserved	Reserved bits

GR5526 SDK provides the following NVDS APIs, to facilitate developers to manipulate non-volatile data in Flash.

Table 2-2 NVDS APIs

Function Prototype	Description
uint8_t nvds_init(uint32_t start_addr, uint8_t sectors)	Initialize the Flash sectors used by NVDS.
uint8_t nvds_get(NvdsTag_t tag, uint16_t *p_len, uint8_t *p_buf)	Read data according to TAG IDs from NVDS.

Function Prototype	Description
uint8_t nvds_put(NvdsTag_t tag, uint16_t len, const uint8_t *p_buf)	Write data to NVDS and mark the data with TAG IDs. You need to create a TAG ID if you write data for the first time.
uint8_t nvds_del(NvdsTag_t tag)	Remove the corresponding data of a TAG ID in NVDS.
uint16_t nvds_tag_length(NvdsTag_t tag)	Obtain the data length of a specified TAG.
uint8_t nvds_drv_func_replace(nvds_drv_func_t *p_nvds_drv_func)	Replace the APIs that can directly control Flash.
uint8_t nvds_func_replace(nvds_func_t *p_nvds_func)	Replace the APIs that control NVDS.
void nvds_retention_size(uint8_t bond_dev_num)	Reserve space for device bonding. The space reserved depends on the number of devices to be bonded.

Note:

- For details about NVDS APIs, see the NVDS header file (in SDK_`Folder\components\sdk\gr55xx_nvds.h`).
- NVDS usage recommendation: It is recommended that NVDS be used only for static system configuration (not for dynamic configuration). For more robust data storage needs, you may develop your own solution or adopt an open-source file system such as LittleFS.

2.5 RAM Mapping

The RAM of a GR5526 SoC is 512 KB in size with the start address of 0x3000_0000. It consists of 11 RAM blocks. Each of the first two RAM blocks is 16 KB, followed by two 32-KB blocks, six 64-KB blocks, and a 32-KB block in sequence. Each RAM block can be powered on/off by software independently.

Note:

GR5526 provides RAM (start address: 0x3000_0000) with an aliasing memory with the start address being 0x0010_0000 and 0x2000_0000. For more information, see Figure 2-3.

- The region (start address: 0x2000_0000) supports bit field operations, mapping to the region starting from 0x2200_0000.
- The region starting from 0x0010_0000 features higher access efficiency thanks to the Cortex®-M4F architecture. Therefore, executing code in this region promotes running speed.
- In GR5526 SDK, RW, ZI, HEAP, and STACK use the RAM region starting from 0x2000_0000; RAM_CODE uses the region starting from 0x0010_0000.

The 512 KB RAM layout is shown in Figure 2-7:

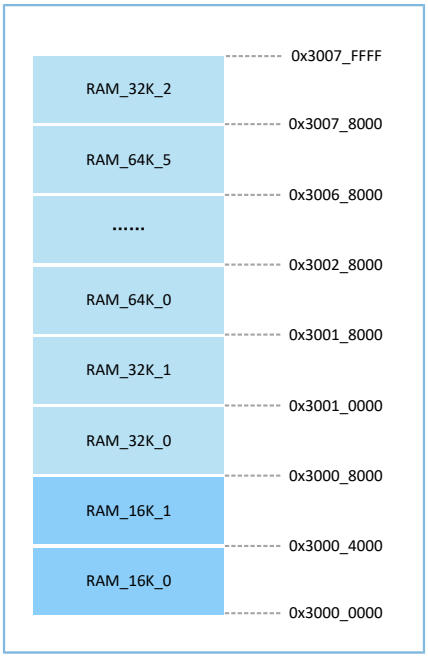


Figure 2-7 512 KB RAM layout

Running modes for applications include XIP and mirror modes. For more information about configurations, see **APP_CODE_RUN_ADDR** in "Section 4.3.2.1 Configuring custom_config.h". RAM layouts of the two modes are different.

Table 2-3 Running modes for applications

Running Mode	Description
XIP mode	It refers to Execute in Place 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.
Mirror mode	In mirror mode, user applications are stored in on-chip Flash, and the running space of applications is defined in RAM. During application boot, applications are loaded into RAM from external Flash after check is completed, and the system jumps to RAM for operation.

Note:

Continuous access to Flash is required in XIP mode. Therefore, power consumption in this mode is a little higher than that in mirror mode.

2.5.1 Typical RAM Layout in XIP Mode

The typical RAM layout under XIP mode is as shown in Figure 2-8. Developers are able to modify the layout based on product needs.

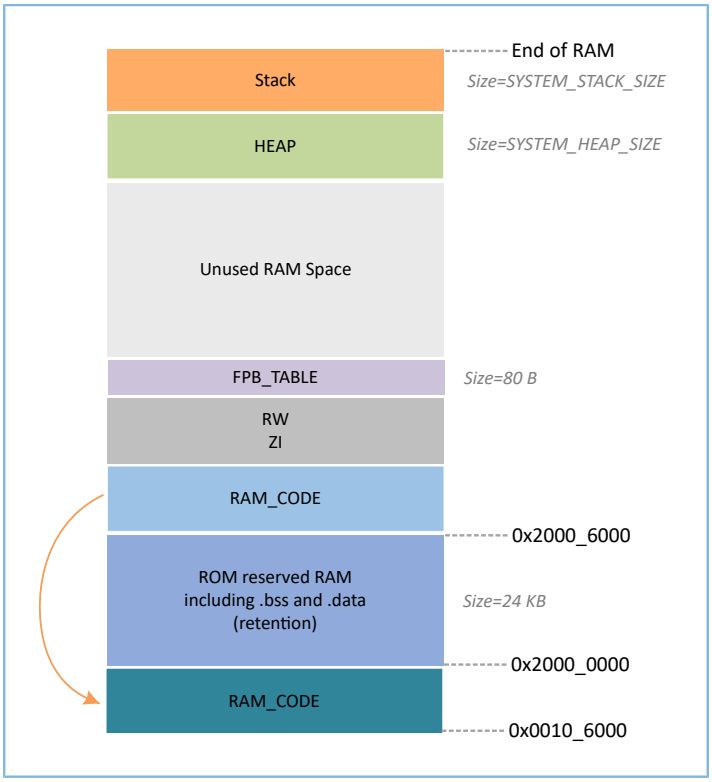


Figure 2-8 RAM layout in XIP mode

RAM_CODE saves code executed in RAM. To boost the efficiency in execution, it is recommended to define this region in the aliasing memory (at physical address 0x00100).

The layout in XIP mode allows application firmware to be run directly in the code loading area, so that more RAM space is available for applications. During update to contents in Flash memory, XIP mode is disabled; during erase of Flash memory, interrupts with priority lower than FLASH_PROTECT_PRIORITY cannot be generated.

Note:

- QSPI0, QSPI1, QSPI2, and OSPI support XIP mode. In this mode, users can map the address of Flash memory or PSRAM to memory, so that users can operate on memory directly.
  - When an external PSRAM is used, the PSRAM is mounted to QSPI1, forming continuous SRAM space with the region from 0x2000 0000 to 0x2007 FFFF.
  - When an internal PSRAM is used, the PSRAM is mounted to OSPI, forming continuous SRAM space with the region from 0x3000 0000 to 0x3007 FFFF.
- Users can add self-defined sections as needed. Avoid modifying the default scatter file of the SDK or deleting part of the scatter file (such as deleting **RAM_CODE** from the scatter file). For details about the scatter file, see "Section 4.3.2.2 Configuring Memory Layout".

2.5.2 Typical RAM Layout in Mirror Mode

The typical RAM layout in mirror mode is as shown in Figure 2-9. Users can modify the layout based on product needs.

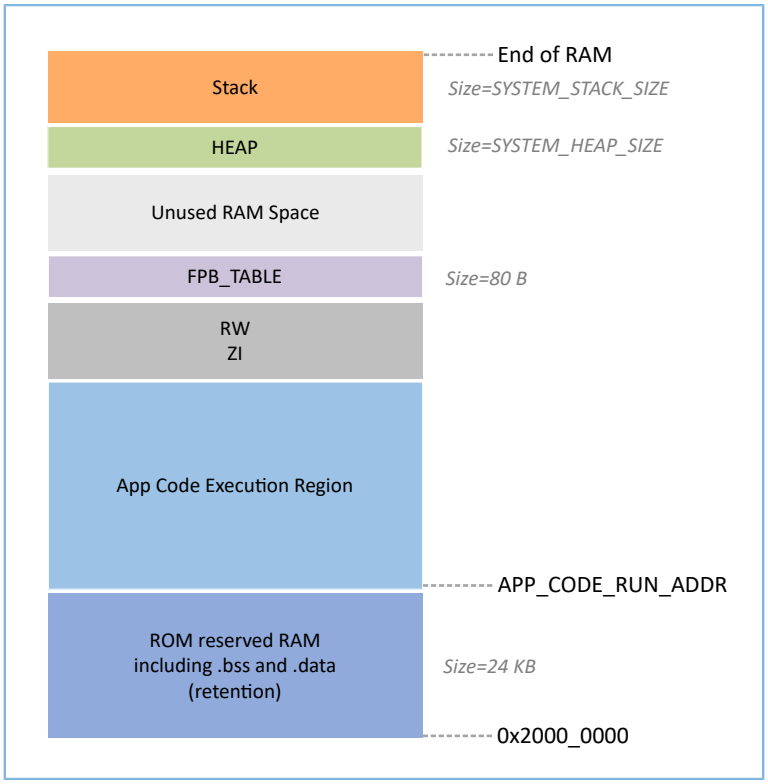


Figure 2-9 RAM layout in mirror mode

The layout in mirror mode allows application firmware to be run in RAM. The SoC enters cold boot process after power-on. The Bootloader copies application firmware from Flash to the RAM segment **App Code Execution Region**. After wake-up from sleep mode, GR5526 SoC enters warm boot process. To shorten the warm boot time, the Bootloader does not redo copy of application firmware to the RAM segment **App Code Execution Region**.

The start address of the **App Code Execution Region** segment depends on APP_CODE_RUN_ADDR in *custom_config.h*. Users need to decide the value of APP_CODE_RUN_ADDR based on the use of .data and .bss segments, to avoid address overlap between the Call Stack segment (higher address segment) and .bss segments (lower address segment). Users can view the layout of RAM segments from the *.map* file.

2.5.3 RAM Power Management

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

- Full Power: The system is in active mode; MCU is permitted to read from and write to RAM blocks.
- Retention Power: The system is in sleep mode; data in RAM blocks does not get lost and is ready for use by the system when it switches from sleep mode to active mode.
- Power Off: The system is in power-off mode; RAM blocks will be powered off and 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 GR5526 enables all RAM power sources when the system starts. The GR5526 SDK also provides a complete set of RAM power management APIs. You can configure the power mode 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 mode, unused RAM blocks are set to **Power Off** mode, and RAM blocks to be used are set to **Full Power** mode.
- When the system enters sleep mode, unused RAM blocks remain in **Power Off** mode, and RAM blocks to be used are set to **Retention Power** mode.

The recommended RAM configurations in practice are described below:

- In Bluetooth LE applications, the first 8 KB of RAM_16K_0 and RAM_16K_1 are reserved for Bootloader and Bluetooth LE Stack only, not available for applications. When the system is in active mode, RAM_16K_0 and RAM_16K_1 shall be in **Full Power** mode; when the system is in sleep mode, the two RAM blocks shall be in **Retention Power** mode. Non-Bluetooth LE applications can use these two RAM blocks.
- Purposes of RAM_32K_0 and other RAM blocks are defined by applications. Generally, user data and the code segments to be executed in RAM are defined in continuous segments starting from RAM_32K_0; the top of function call stacks is defined in upper address part of RAM. The power mode of these RAM blocks can be enabled, or be controlled by applications.

Note:

- An MCU is permitted only when a RAM block is in **Full Power** mode.
- For details about RAM power management APIs, refer to `SDK_Folder\components\sdk\platform_sdk.h`.

2.6 PSRAM

GR5526VGBIP SoC and GR5526RGNIP SoC have an 8-MB PSRAM with OSPI for data access. The PSRAM address is mapped to 0x30080000, forming a contiguous SRAM space together with the area from 0x3000 0000 to 0x3007 FFFF, providing larger SRAM space for users. The PSRAM features:

- Low power consumption
  - Partial array self-refresh (PASR)
  - Auto Temperature Compensated Self-Refresh (ATCSR) of built-in temperature sensor
  - User-configurable refresh rate
  - Ultra-low power consumption (ULP) in half sleep mode with data retained
- Software reset
- Output driver low voltage complementary metal oxide semiconductor (LVCMOS) with programmable drive strength
- Data mask for data write

- Data strobe enabled high-speed data read
- Register-configurable write and read initial latencies
- Write burst length: 2 bytes to 1024 bytes
- Wrap burst and hybrid burst at 16 B, 32 B, 64 B, and 1 KB
- Linear burst command
- Row boundary crossing (RBX)
  - Read can be enabled by mode register.
  - RBX write is not supported.

Note:

- GR5526 SoCs are embedded with PSRAM. By default, PSRAM is disabled. Users can enable PSRAM with OSPI controller before use.
- To improve power consumption performance, users can modify the impedance matching between the OSPI controller and PSRAM by adjusting PSRAM drive strength.
  - Lower drive strength means lower power consumption, and the waveform tends to be triangle wave, which is of lower quality.
  - Greater drive strength means higher power consumption, and the waveform tends to be square wave, which features higher quality.
  - Set the drive strength appropriate to the application scenario to avoid system crash caused by excessively high drive strength.
- The efficiency of MCU reading through OSPI is low. Therefore, it is recommended to access OSPI based on DMA alignment.

2.7 GR5526 SDK Directory Structure

The folder directory structure of GR5526 SDK is as shown below.

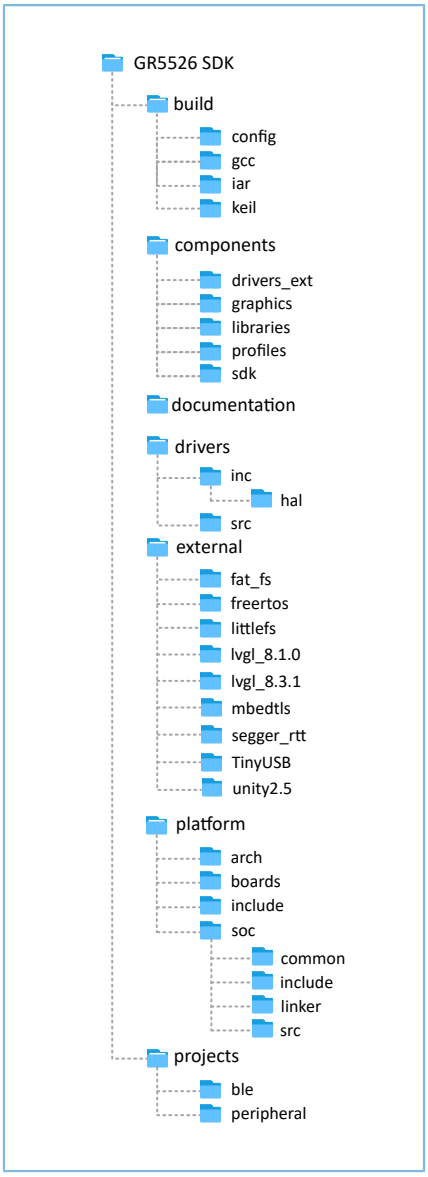


Figure 2-10 GR5526 SDK directory structure

Detailed description of folders in GR5526 SDK is as shown below.

Table 2-4 GR5526 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
build\iar	IAR tools
components\drivers_ext	Drivers of third-party components on the development board
components\graphics	Contents about GPU display

Folder	Description
components\libraries	Libraries provided in the GR5526 SDK
components\profiles	Source files of GATT Services/Service Clients implementation examples provided in the GR5526 SDK
components\sdk	API header files provided in the GR5526 SDK
documentation	*GR5526 API Reference Manual*
drivers\inc\hal	HAL and LL header files of the GR5526 peripheral drivers
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\fat_fs	Source code from FatFs (a third-party program)
external\freertos	Source code from FreeRTOS (a third-party program)
external\littlefs	Source code from LittleFS (a third-party program)
external\lvgl_8.1.0	Source code from LVGL V8.1.0 (a third-party program)
external\lvgl_8.3.1	Source code from LVGL V8.3.1 (a third-party program)
external\mbedtls	Source code from mbed TLS (a third-party program)
external\segger_rtt	Source code from SEGGER RTT (a third-party program)
external\TinyUSB	Source code from TinyUSB (a third-party program)
external\unity2.5	Source code from Unity 2.5 (a third-party program)
platform\arch	Toolchain files of CMSIS
platform\boards	Source files for initializing GR5526 Starter Kit Board (GR5526 SK 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 GR5526 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 Heart Rate Sensor and Proximity Reporter
projects\peripheral	Peripheral project examples of a GR5526 SoC

3 Bootloader

The GR5526 supports two firmware running modes: XIP and mirror. When the system is powered on, the Bootloader first reads the system boot configuration information from SCA, then performs application firmware integrity check and initialization configuration accordingly, and finally jumps to the code running space to run firmware. The boot procedures may vary in different running modes.

- In XIP mode, the Bootloader first initializes Cache and XIP controllers after finishing application firmware check, and then jumps to the code run address in Flash to run code.
- In mirror mode, after finishing application firmware check, the Bootloader loads the firmware in Flash to corresponding RAM running space based on system configurations, and jumps to and runs the firmware in RAM.

The application boot procedures of the GR5526 SDK are shown as follows.

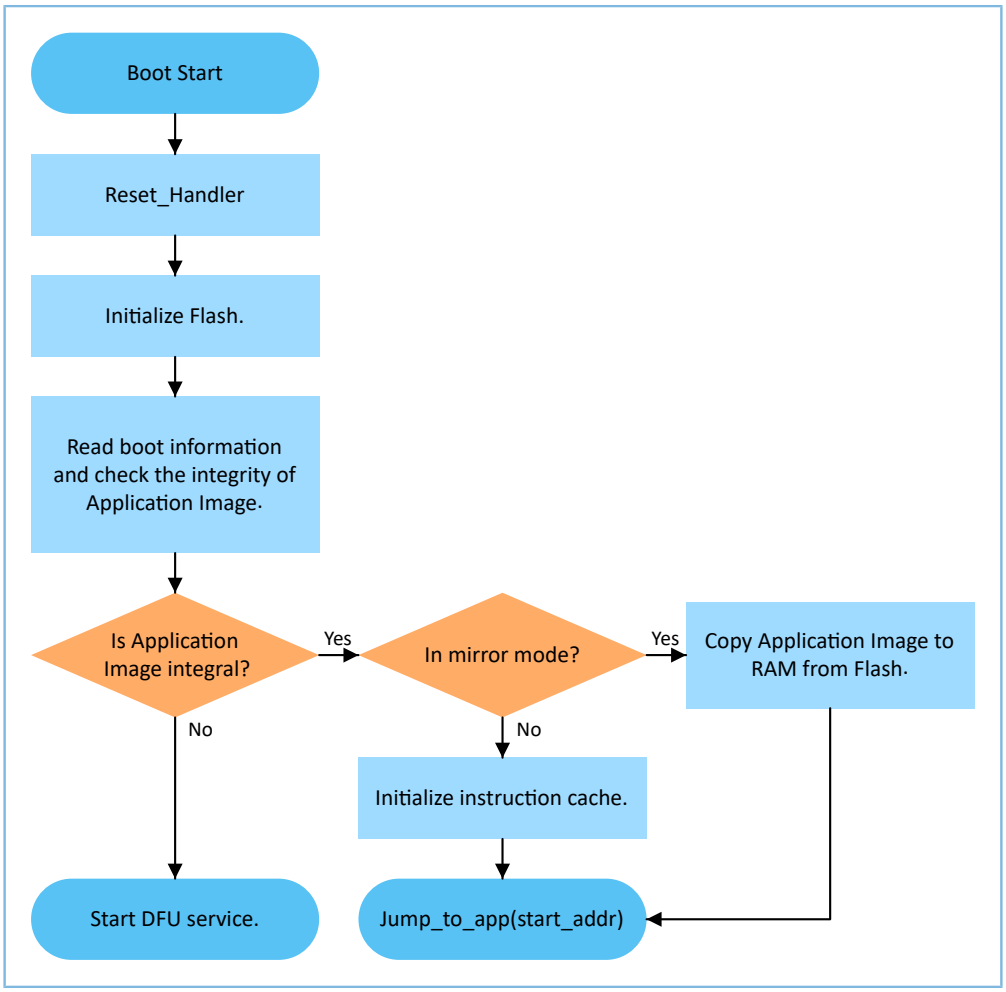


Figure 3-1 Application boot procedures of the GR5526 SDK

1. When the device is powered on, CPU jumps to 0x0000_0000, from which extracts 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.

Note:

GR5526 supports encrypting and signing application firmware in security mode.

- Security mode: If the security mode is enabled, the Bootloader reads boot information from SCA and performs HMAC check; after the check succeeds, the Bootloader decrypts SCA boot information and then implements the signature verification process in the secure boot process, to guarantee firmware integrity and prevent tampering or disguise; if signature verification succeeds, the automatic decryption functionality is enabled.
- Non-security mode: If the security mode is not enabled, the Bootloader performs cyclic redundancy check (CRC) on application firmware based on SCA boot information.

---

4. If CRC fails, the Bootloader enters J-Link DFU mode. You can update application firmware in Flash with GProgrammer and J-Link.
5. If CRC passes, Bootloader checks the running mode.
    - In XIP mode, the Bootloader jumps to the application firmware in Flash to start implementation after XIP configuration is completed.
    - In mirror mode, the Bootloader copies the application firmware in Flash to a specified segment in RAM, and then runs the application firmware in RAM.

4 Development and Debugging with GR5526 SDK in Keil

This chapter introduces how to build, compile, download, and debug Bluetooth LE applications with the GR5526 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. For the GR5526 SDK, Keil V5.20 or a later version shall be installed.

Note:

For more information about how to use Keil MDK-ARM IDE, see ARM online manuals.

The main interface of Keil is as shown below.

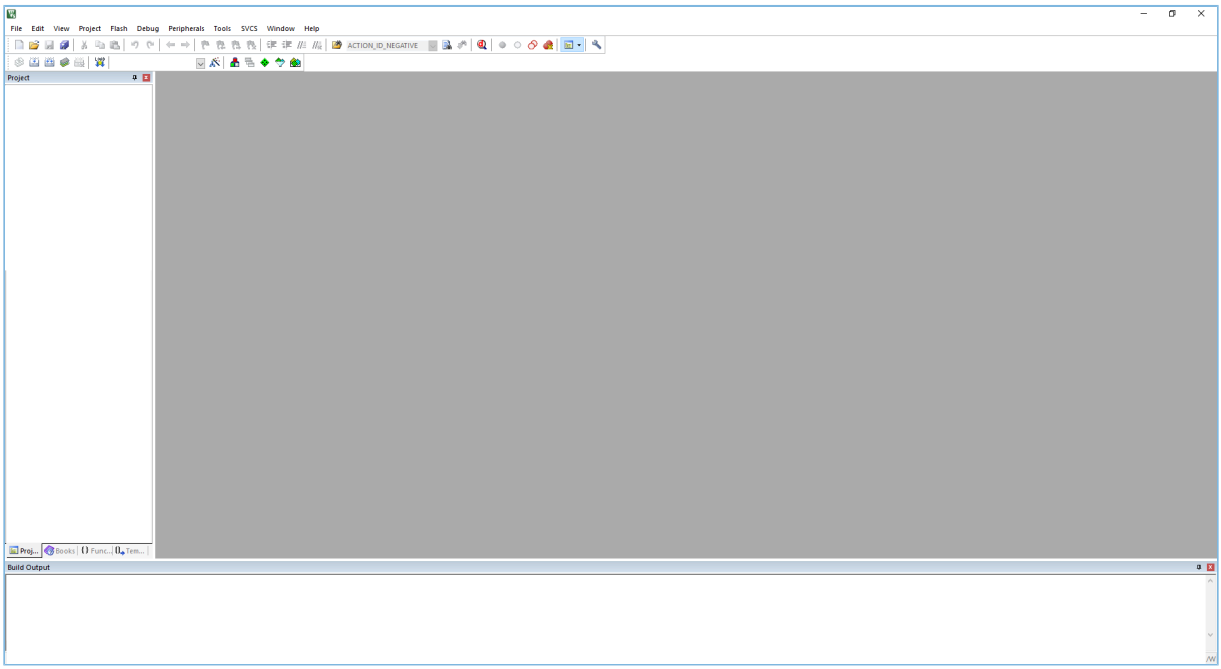


Figure 4-1 Keil interface

Frequently used function buttons of Keil are as shown below.

Table 4-1 Frequently used function buttons of Keil

Keil Icon	Description
	Options for Target
	Start/Stop Debug Session
	Download
	Build

4.2 Installing GR5526 SDK

The GR5526 SDK is in a .zip file. You can access the details after extracting the file.

Note:

- SDK_Folder is the root directory of GR5526 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 GR5526 SDK.

4.3.1 Preparing ble_app_example

This section elaborates on how to create a project based on the template project provided in GR5526 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**.

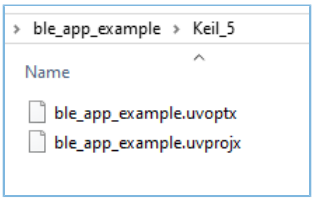


Figure 4-2 ble_app_example folder

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**.

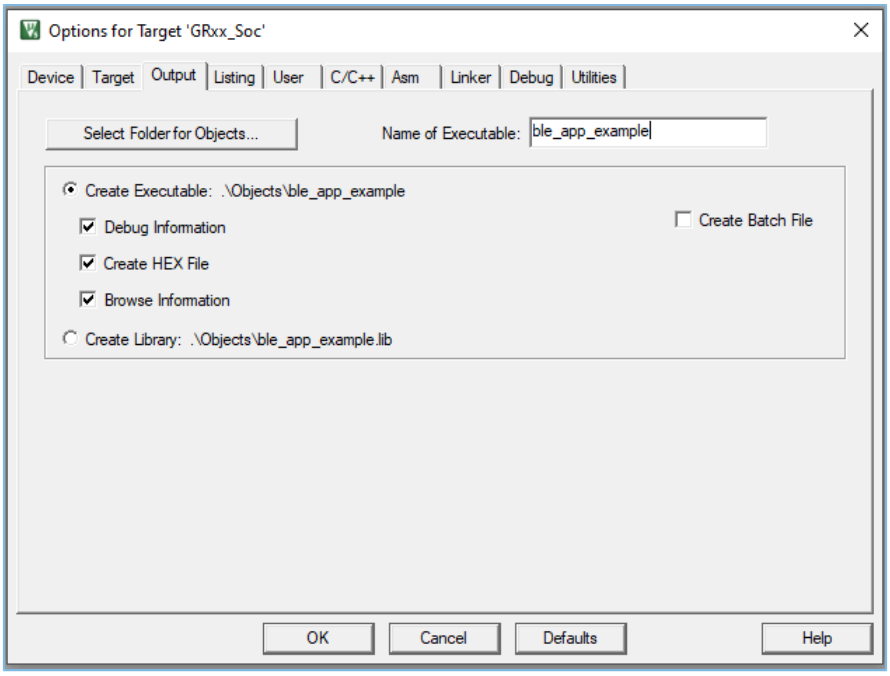


Figure 4-3 Modifications to **Name of Executable**

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

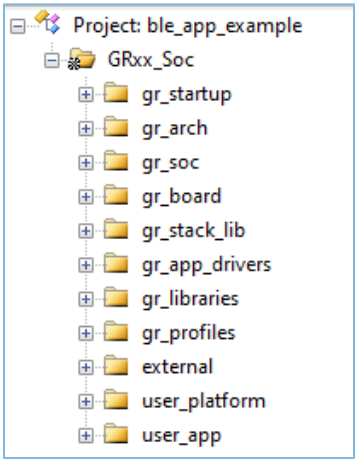


Figure 4-4 ble_app_example groups

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

- SDK groups

  The SDK groups include gr_startup, gr_arch, gr_soc, gr_board, gr_stack_lib, gr_app_drivers, gr_libraries, gr_profiles, and external.

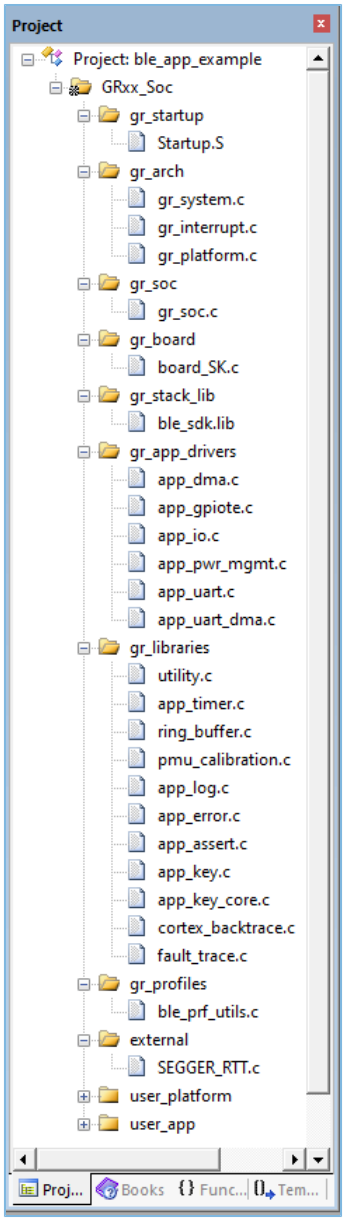


Figure 4-5 SDK groups

Source files in the SDK groups are not required to be modified. Group descriptions are provided below:

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
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

SDK Group Name	Description
gr_stack_lib	GR5526 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
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

  User groups include user_platform and user_app.

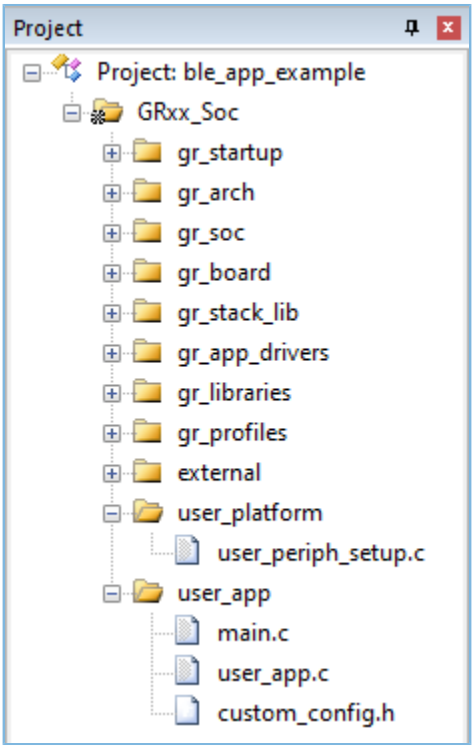


Figure 4-6 User groups

Functionalities for source files in User groups need to be implemented by developers. Group descriptions are provided below:

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 Group Name	Description
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, After Build, and other configuration items.

4.3.2.1 Configuring custom_config.h

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

Note:

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

- Modify the configurations in *custom_config.h*.

  GR5526 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_GR5526	Define the SoC version number.
SYS_FAULT_TRACE_ENABLE	Enable/Disable Callstack Trace Info printing. If printing is enabled, the Callstack Trace Info is printed when a HardFault occurs. ◦ 0: Disable ◦ 1: Enable
ENABLE_BACKTRACE_FEA	Enable/Disable stack backtrace functionality. ◦ 0: Disable ◦ 1: Enable
APP_DRIVER_USE_ENABLE	Enable/Disable the App Drivers module. ◦ 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.

Macro	Description
	◦ 0: Disable ◦ 1: Enable
APP_LOG_PORT	Set the output mode of APP LOG module. ◦ 0: UART ◦ 1: J-Link RTT ◦ 2: ARM ITM **Note:** By default, this macro is commented out in *custom_config.h*, and the default value is 0. It can be redefined by developers on demand.
SK_GUI_ENABLE	Enable/Disable the GUI module on GR5526 SK Board. ◦ 0: Disable ◦ 1: Enable
DTM_TEST_ENABLE	Enable/Disable DTM Test. ◦ 0: Disable ◦ 1: Enable
PMU_CALIBRATION_ENABLE	Enable/Disable PMU calibration. When PMU calibration is enabled, the system monitors temperature and voltage automatically with adaptive adjustment. ◦ 0: Disable ◦ 1: Enable **Note:** PMU calibration shall be enabled in high/low temperature scenarios.
FLASH_PROTECT_PRIORITY	Set the priority level to respond to an exception during Flash write or erase operation. When FLASH_PROTECT_PRIORITY is set to N, interrupt requests with a priority level not higher than N are suspended. After the Flash write or erase operation is completed, the system responds to the suspended interrupt requests. By default, the system does not respond to any interrupt request during Flash write or erase operation. Developers can set a value on demand.
NVDS_START_ADDR	Start address of NVDS in Flash By default, the macro is commented out in *custom_config.h*. If you need to reconfigure the NVDS address, enable the macro and set the address as needed (4-KB alignment is compulsory). **Note:** The start address cannot be set in used areas in the memory (such as SCA and user App).
NVDS_NUM_SECTOR	Number of Flash sectors for NVDS
SYSTEM_STACK_SIZE	Size of Call Stack required by applications. The default value is 12 KB.

Macro	Description
	You can set the value as needed. Please note that the value shall not be less than 6 KB. **Note:** After compilation of ble_app_example, the Maximum Stack Usage is provided in `Keil_5\Objects\ble_app_example.htm` for reference.
SYSTEM_HEAP_SIZE	Size of Heap required by applications. The default value is 16 KB. You can set the value as needed.
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 If the value is the same as APP_CODE_LOAD_ADDR, applications run in XIP mode. If the value is within the RAM address range, applications run in mirror mode.
EXT_RAM1_STRAT_ADDR	Start address of PSRAM (OSPI)
EXT_RAM2_STRAT_ADDR	Start address of PSRAM (QSPI M1)
SYSTEM_CLOCK*	Set the system clock frequency. ◦ 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	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
CFG_LF_ACCURACY_PPM	Bluetooth LE low-frequency sleep clock accuracy. The value shall range from 1 to 500 (unit: ppm).
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.
VERSION*	Version number of application firmware; length: 2 bytes; it is stored in hexadecimal format.

Macro	Description
CHIP_VER	Version of the SoC that the firmware runs on, currently set to 0x5526
CFG_CONTROLLER_ONLY	Use Bluetooth LE Controller only or not. ◦ 0: Use Bluetooth LE Controller and Host. ◦ 1: Use Bluetooth LE Controller only.
CFG_MAX_PRFS	Maximum number of GATT Profiles/Services supported by applications You can set the value on demand. A larger value means occupying more RAM space.
CFG_MAX_BOND_DEVS	Maximum number of devices that can be bonded to applications; Max.: 4
CFG_MAX_CONNECTIONS	Maximum number of devices that can be connected to applications; the number shall be no greater than 10. You can set the value based on needs. A larger value means more RAM space to 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 Bluetooth LE legacy advertising and extended advertising supported by applications
CFG_MAX_PER_ADVS	Maximum number of Bluetooth LE periodic advertising supported by applications **Note:** The total number of legacy advertising and extended advertising (CFG_MAX_ADVS) plus the number of periodic advertising (CFG_MAX_PER_ADVS) shall be no greater than 5.
CFG_MAX_SYNCS	Number of synchronized periodic advertising; used for reserving RAM for Bluetooth LE Stack. You can set the value according to the number of synchronized periodic advertising in use. Max.: 5.
CFG_MAX_SCAN	Maximum number of supported Bluetooth LE device used for scanning in applications. Max.: 1.
CFG_MUL_LINK_WITH_SAME_DEV	Support multi-link functionality for a single device or not, typically used for Find My applications. ◦ 0: No ◦ 1: Yes
CFG_EATT_SUPPORT	Support the Bluetooth LE EATT functionality or not. ◦ 0: No ◦ 1: Yes

Macro	Description
CFG_MAX_EATT_CHANNELS	Maximum number of Bluetooth LE EATT channels supported in Application. Max.: 10.
CFG_ISO_SUPPORT	Indicate whether the ISO functionality is supported. ◦ 0: No ◦ 1: Yes
CFG_BT_BREDR	Support Bluetooth BR/EDR or not. ◦ 0: No ◦ 1: Yes
CFG_CAR_KEY_SUPPORT	Support digital car key applications or not. ◦ 0: No ◦ 1: Yes
CFG_LCP_SUPPORT	Support the LCP module or not. ◦ 0: No ◦ 1: Yes
SECURITY_CFG_VAL	Configure the algorithm security level. ◦ 0: Level 1 ◦ 1: Level 2

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

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

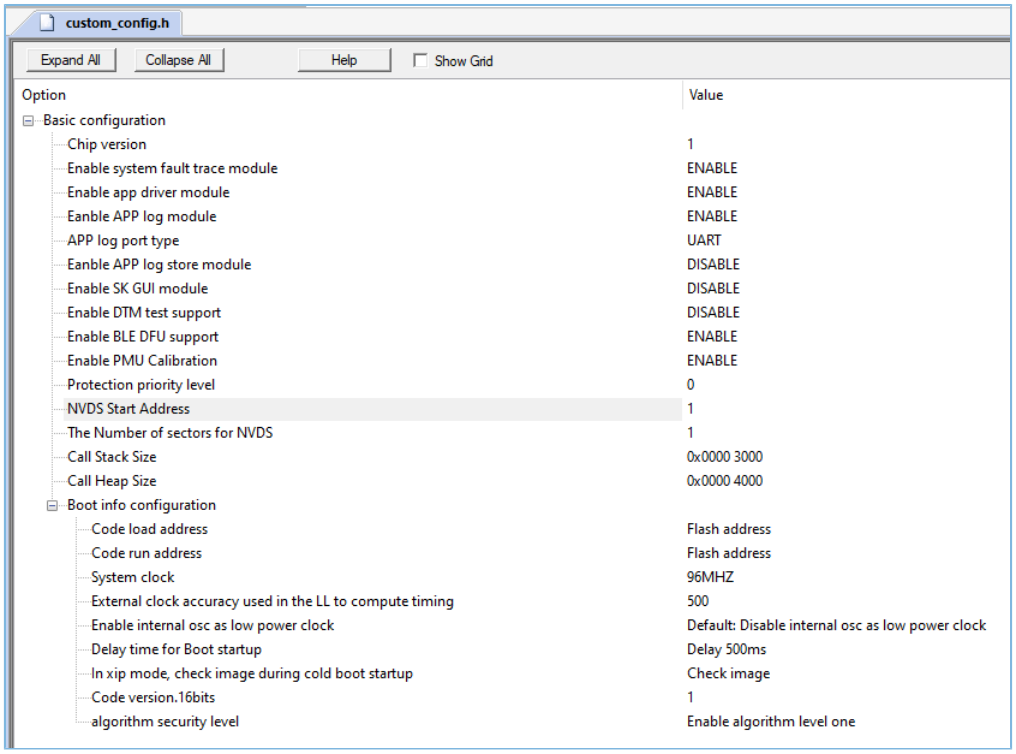


Figure 4-7 custom_config.h in the **Configuration Wizard** interface

Comments in *custom_config.h* are compliant with Configuration Wizard Annotations of Keil, making it possible for users to open *custom_config.h* in Keil and configure application project parameters in the **Configuration Wizard** interface of Keil.

Tip:

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

4.3.2.2 Configuring Memory Layout

Keil defines memory segments for the linker in *.sct* files. The GR5526 SDK provides an example *flash_scatter_common.sct* (in SDK_`Folder\platform\soc\linker\keil\flash_scatter_common.sct`) to help developers quickly configure memory layout. The macros used by this *.sct* file are defined in *flash_scatter_config.h*.

Note:

In Keil, `__attribute__((section("name")))` can be used to define a function or a variable at a specific memory segment, in which **name** depends on your choice. A scatter (*.sct*) file specifies the location for a customized segment. For example, to define Zero-Initialized (ZI) data of applications at 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 file (*.h*) 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 required for *flash_scatter_common.sct*. A wrong path results in a linker error.

---

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

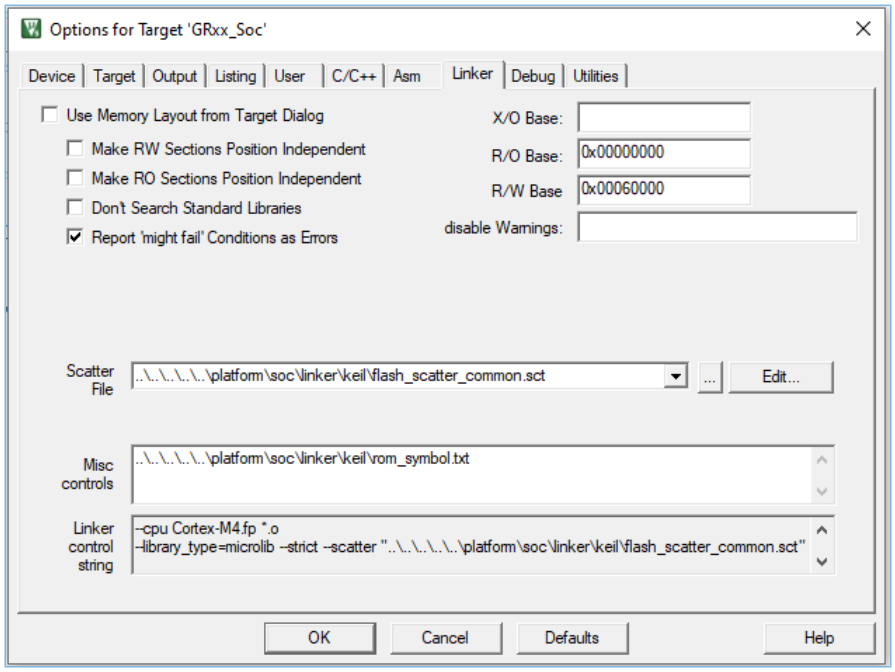


Figure 4-8 Configuration of scatter file

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

4.3.2.3 Configuring After Build

After Build in Keil can specify the command to be executed after a project is built.

By default, the after build command will be executed for ble_app_template. ble_app_example, which is based on ble_app_template, does not require manual configuration of **After Build**.

If you build a project in Keil, follow the steps below to configure **After Build**:

1. Click (**Options for Target**) on the Keil toolbar and open the **Options for Target 'GRxx_Soc'** dialog box. Select the **User** tab.

2. From the options expanded from **After Build/Rebuild**, select **Run #1**, and type **fromelf.exe --text -c --output Listings\@L.s Objects\@L.axf** in the corresponding **User Command** field. This step helps you utilize Keil fromelf to generate a compiling file based on the selected .axf file.
3. From the options expanded from **After Build/Rebuild**, select **Run #2**, and type **fromelf.exe --bin --output Listings \@L.bin Objects\@L.axf** in the corresponding **User Command** field. This step helps you utilize Keil fromelf to generate a compiling file based on the selected .axf file.
4. Click **OK** to save the settings.

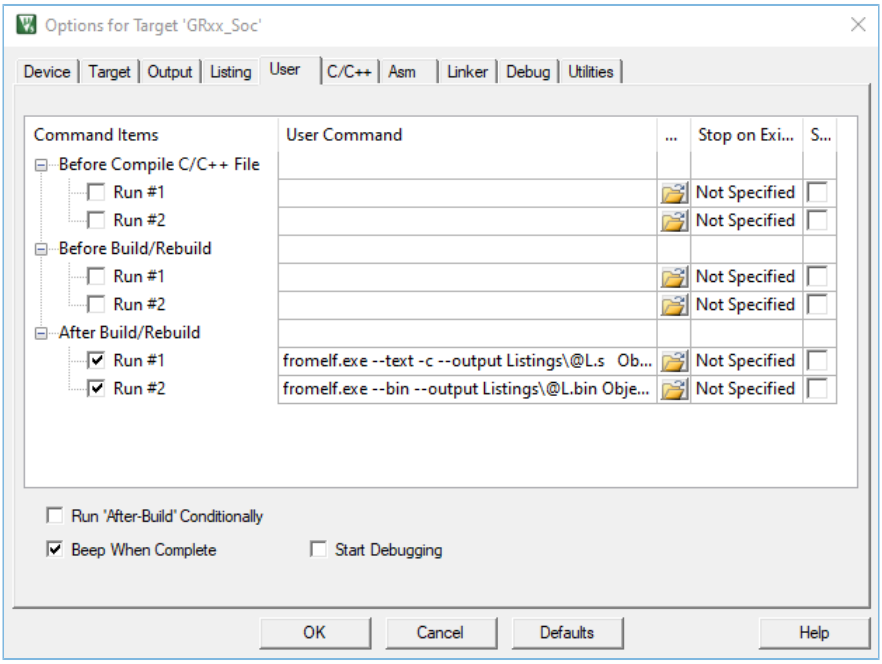


Figure 4-9 Configuration of After Build

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(&&m_app_ble_callback, &heaps_table);

    // Main Loop
    while (1)
    {
```



```
        /*
         * Add Application code here, e.g. GUI Update.
         */
         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 might not work normally. The Heap size is related to the Bluetooth LE service volume configured in "Section 4.3.2.1 Configuring custom_config.h".
- app_periph_init() is used for initializing peripherals. In development and debugging phases, the 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);
   bsp_log_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 enable the APP LOG module, call the app_log_flush() in the main loop. This is to ensure logs are output completely before the SoC enters sleep mode. For more information about the APP LOG module, see "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 Logic

Related Bluetooth LE service logic of applications are driven by a number of Bluetooth LE events which are defined in the GR5526 SDK. Therefore, applications need to implement the corresponding event handlers in GR5526 SDK to obtain operation results or state change notifications of Bluetooth LE Stack. 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 might be 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:

You can view the Bluetooth LE events supported by GR5526 SDK in SDK_Folder\components\sdk\ble_event.h.

You need to implement necessary 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\GR5526_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 (see "Section 2.2 Software Architecture"). Therefore, BLE_Stack_IRQ has the second-highest priority after SVCall IRQ, ensuring that Bluetooth LE Stack runs strictly in a timing specified in *Bluetooth Core Spec*.

A state change of Bluetooth LE Stack triggers 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.

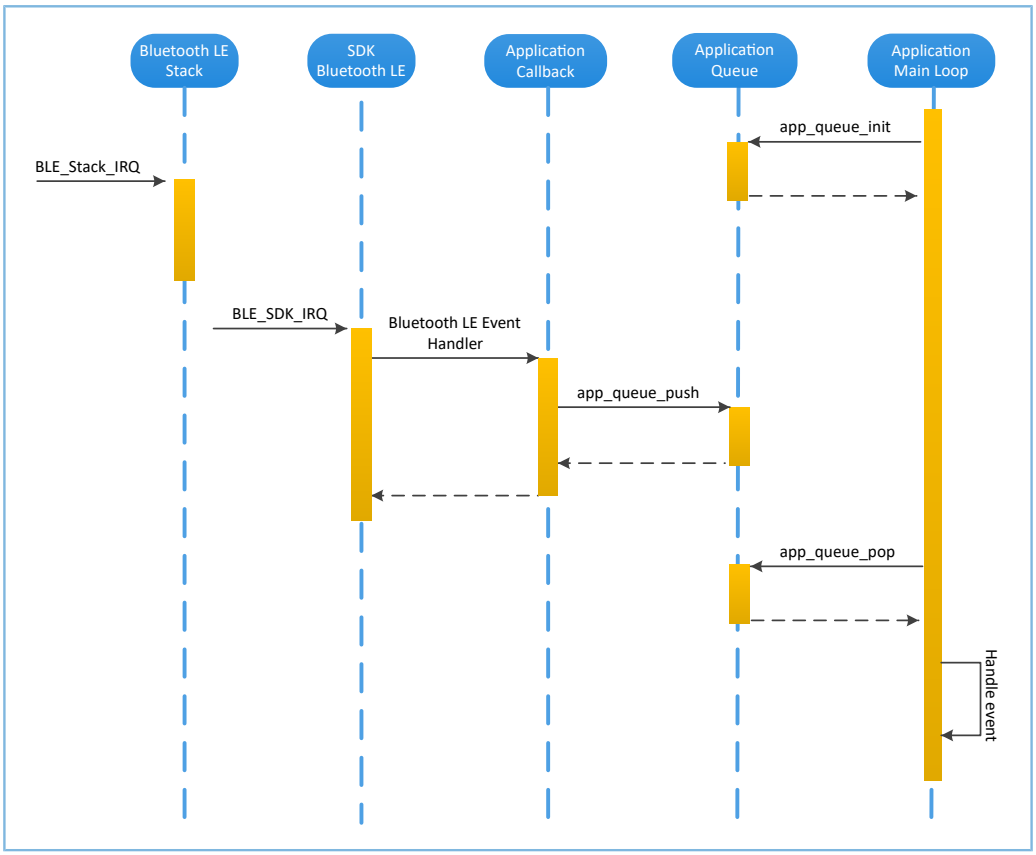


Figure 4-10 System schedule (without OS)

4.4 Generating Firmware

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

After the project compilation is completed, two firmware files are created in `Keil_5\Listings` and `Keil_5\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. See *GProgrammer User Manual* for detailed operations.

4.5 Downloading .hex Files to Flash

After *.hex* files are generated, you can download these files to Flash following the steps 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**.

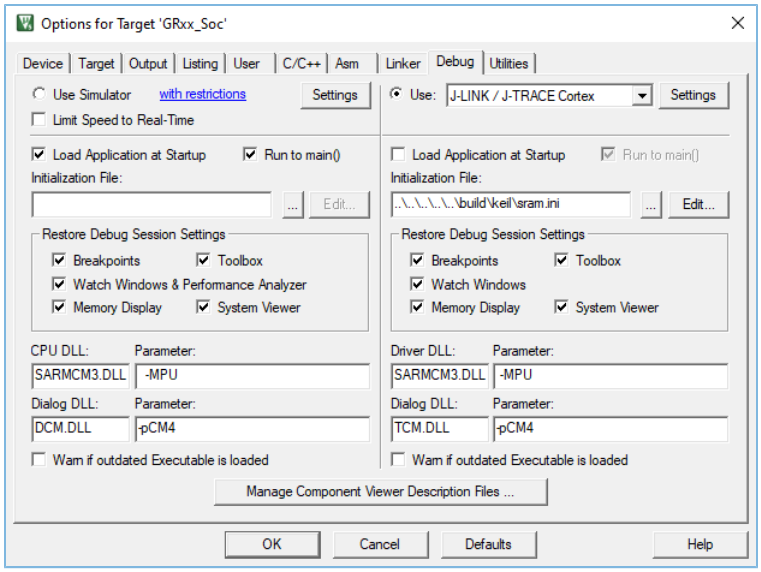


Figure 4-11 Debug tab

(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:

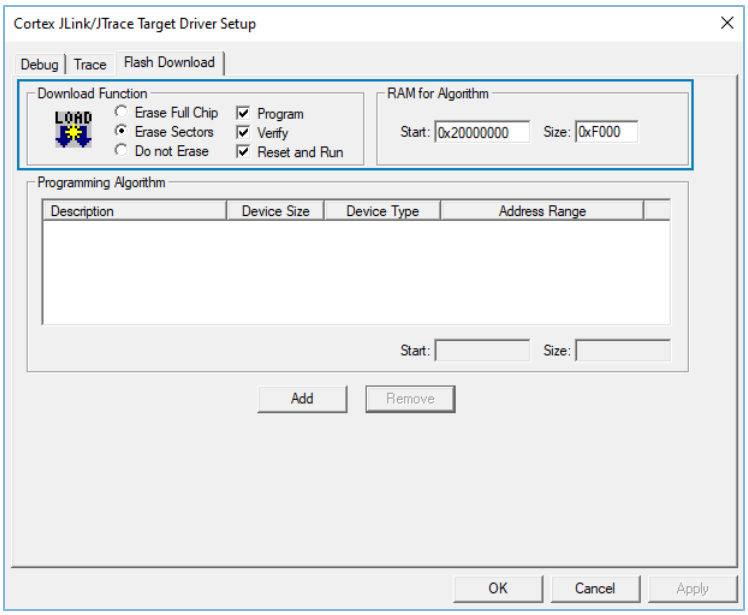


Figure 4-12 Default configurations in the Download Function pane

(4) Click **Add** to add *GR5xxx_16MB_Flash.FLM* (in SDK_`Folder\build\keil\`) to the **Programming Algorithm**.

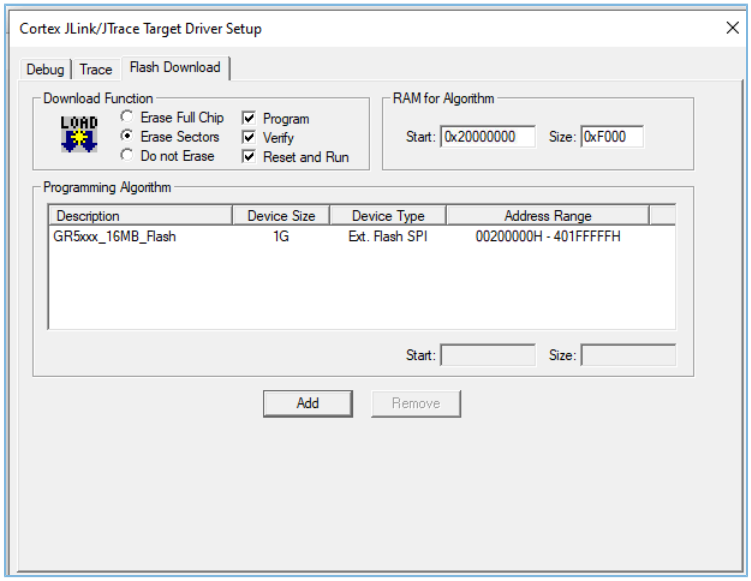


Figure 4-13 Adding GR5xxx_16MB_Flash.FLM to Programming Algorithm

(5) Configure **RAM for Algorithm** to define the address space to load and implement the programming algorithm. Enter the start address of RAM in GR5526 in the **Start** input field: **0x20000000**. Enter **0xF000** in the **Size** input field.

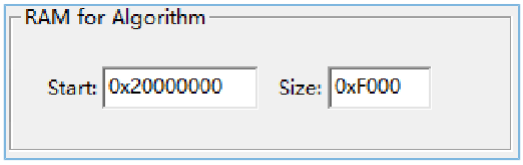


Figure 4-14 Settings of **RAM for Algorithm**

(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.

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 GR5526 SK Board and wait for about 1 second; then click (**Download**) to download the file again.

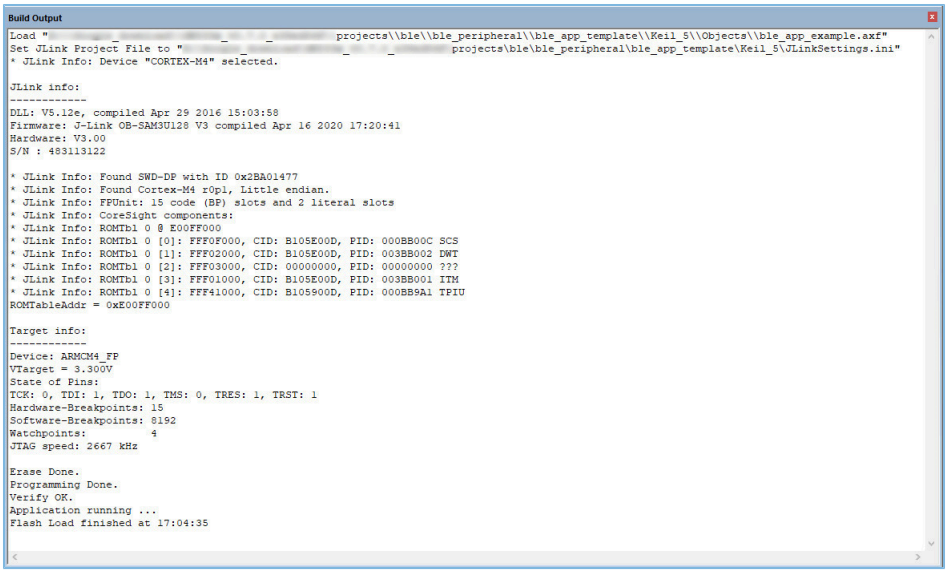


Figure 4-15 Download results

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 **Debug** tab. In the window, software simulation debugging configurations

display on the left, and online hardware debugging configurations display on the right. Bluetooth LE example projects adopt the online hardware debugging. Related default configurations of the debugger are shown as follows:

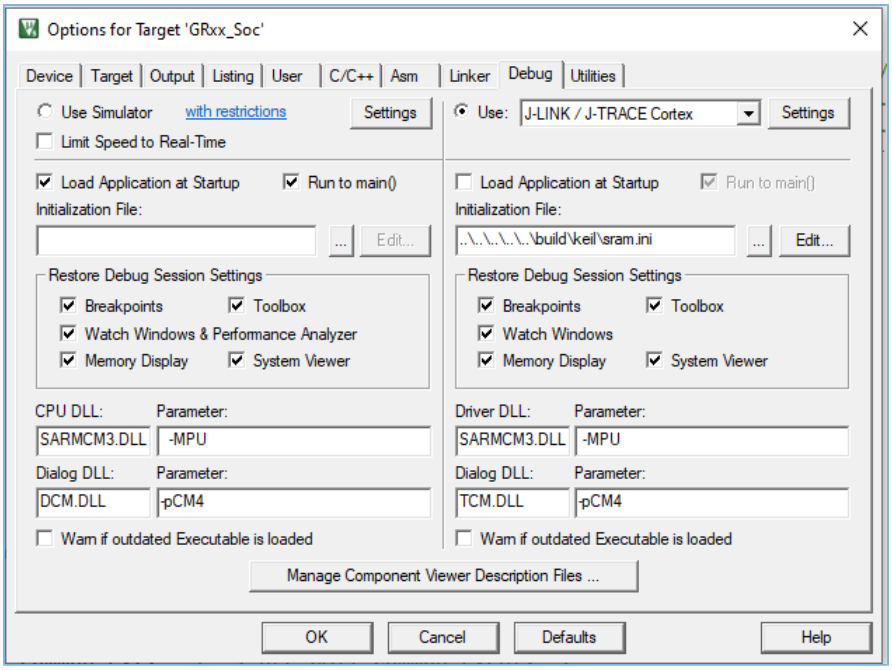


Figure 4-16 Configuring the Debugger

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.

sram.ini contains a set of debug commands, which are executed during debugging. Click **Edit...** on the right side of the **Initialization File** bar, 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 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)
```

Note:

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

1. When **Load Application at Startup** (**Options for Target 'GRxx_Soc'** > **Debug** > **Load Application at Startup**) is enabled, the debugger first loads the file under **Name of Executable** (**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 both options under **Connect & Reset Options** are set to **Normal**, as shown in Figure 4-17. This is to ensure when you click **Reset** on the Keil toolbar after enabling **Debug Session**, the program can run normally.

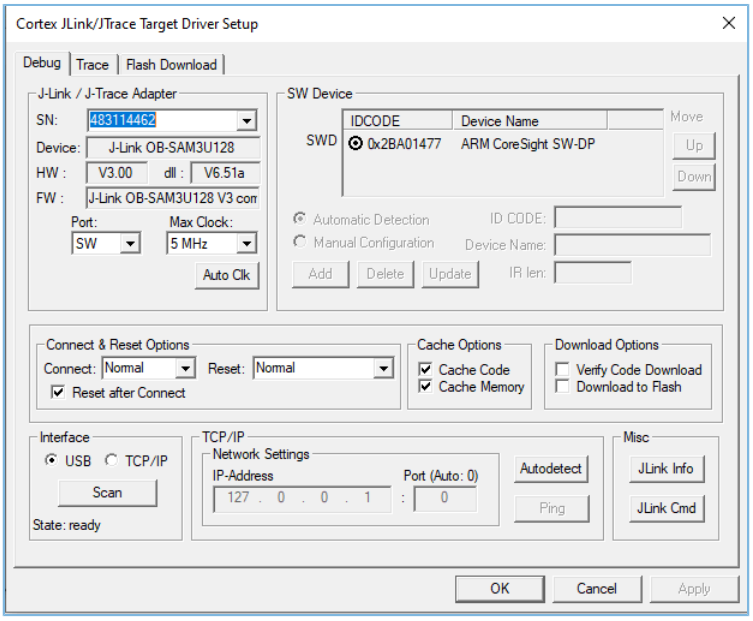


Figure 4-17 Setting **Connect** and **Reset** to **Normal** in **Connect & Reset Options**

4.6.3 Outputting Debug Logs

GR5526 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 configuring log parameters, and registering log output APIs and Flush APIs.

The APP LOG module supports using the `printf()` (a C standard library function) and APP LOG APIs to output debug logs. If you choose APP LOG APIs, you can optimize log output 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 (see SDK_`Folder\components\libraries\bsp\bsp.h` for details) and register corresponding transmission and flush functions (see user_log_debug_init() for reference) according to the configured output port.

If UART is the output port, user_log_debug_init() is implemented as follows.

```
static void user_log_debug_init(void)
{
    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;

    app_log_init(&log_init, bsp_uart_send, bsp_uart_flush);

#if APP_LOG_STORE_ENABLE
    app_log_store_info_t store_info;
    app_log_store_op_t   op_func;

    store_info.nv_tag   = APP_LOG_NVDS_TAG;
    store_info.db_addr  = APP_LOG_DB_START_ADDR;
    store_info.db_size  = APP_LOG_DB_SIZE;
    store_info.blk_size = APP_LOG_ERASE_BLK_SIZE;

    op_func.flash_init  = hal_flash_init;
    op_func.flash_erase = hal_flash_erase;
    op_func.flash_write = hal_flash_write;
    op_func.flash_read  = hal_flash_read;
    op_func.time_get    = NULL;

    app_log_store_init(&store_info, &op_func);
#endif

}
```

Note:

- The input parameters of app_log_init() include the log initialization parameter, log output API, and flush API (optional for registration).
- GR5526 SDK provides an APP LOG STORE module, which supports storing the debug logs in Flash and outputting the logs from Flash. To use the APP LOG STORE module, users need to enable APP_LOG_STORE_ENABLE in *custom_config.h*. This module is configured in the ble_app_rscs project (in `SDK_Folder\projects\ble\ble_peripheral\ble_app_rscs`). This configuration can be a reference when the APP LOG STORE module is used.
- Application logs output by using `printf()` cannot be stored by the APP LOG STORE module.

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 asynchronization (app_transmit_async) and hal_uart synchronization (hal_uart_transmit). You can choose the output methods as needed.
- bsp_uart_flush() is used to output the remaining data that is cached in memory in interrupt mode.

Note:

You can rewrite the above two 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.

If you choose armcc for compilation and output logs through J-Link RTT, it is recommended to make the following modifications in *SEGGER_RTT.c*:

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));
```

Figure 4-18 Creating RTT Control Block and placing it at 0x20005000

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

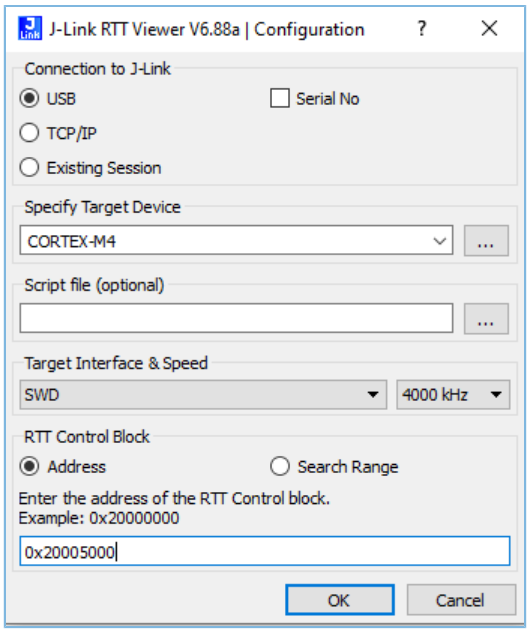


Figure 4-19 Configurations in J-Link RTT Viewer

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 4-18 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 4-20 Obtaining RTT Control Block address

Note:

If you choose GCC for compilation, modifications shown in Figure 4-18 are not required. The address to be entered for RTT Control Block in J-Link RTT Viewer should be the address of **_SEGGER_RTT** in the .map file generated by the compilation project.

4.6.4 Debugging with GRToolbox

GR5526 SDK provides an Android App, GRToolbox, to debug GR5526 Bluetooth LE applications. GRToolbox features the following:

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

Tip:

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

5 Glossary

Table 5-1 Glossary

Acronym	Description
AoA/AoD	Angle of Arrival/Angle of Departure
API	Application Programming Interface
ATT	Attribute Protocol
Bluetooth LE	Bluetooth Low Energy
DFU	Device Firmware Update
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GFSK	Gaussian Frequency Shift Keying
HAL	Hardware Abstract Layer
HCI	Host Controller Interface
IoT	Internet of Things
ISOAL	Isochronous Adaptation Layer
L2CAP	Logical Link Control and Adaptation Protocol
LL	Link Layer
NVDS	Non-volatile Data Storage
OTA	Over The Air
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
UART	Universal Asynchronous Receiver/Transmitter
XIP	Execute in Place