GR551x FreeRTOS Example Application

Version: 1.7

Release Date: 2020-12-15

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: 2F. & 13F., Tower B, Tengfei Industrial Building, Futian Free Trade Zone, Shenzhen, China

TEL: +86-755-33338828 FAX: +86-755-33338099

Website: www.goodix.com

Preface

Purpose

This document introduces how to use and modify a FreeRTOS example in a GR551x SDK, to help users quickly get started with secondary development.

Audience

This document is intended for:

- GR551x user
- GR551x developer
- GR551x tester
- Hobbyist developer
- Technical writer

Release Notes

This document is the fifth release of *GR551x FreeRTOS Example Application*, corresponding to GR551x SoC series.

Revision History

Version	Date	Description
1.0	2019-12-08	Initial release
1.3	2020-03-16	Updated the release time in the footers.
1.5	2020-05-30	Updated the project directory figure in “Section 4.1 Project Directory”.
1.6	2020-06-30	Updated the document version based on SDK changes.
1.7	2020-12-15	Updated GRToolbox UI figure based on software update.

Contents

1 Introduction

FreeRTOS is an excellent embedded real-time operating system for microcontrollers. Being light-weighted, distributed freely under MIT open-source License, and built with an emphasis on portability, tailorability, and flexible scheduling policy, it requires low RAM/ROM consumption and supports management of task, time, semaphore, message queue, and memory.

This document introduces the FreeRTOS porting example in the GR551x SDK, including usage of the example and descriptions of key source code.

Before you use and modify a FreeRTOS example, it is recommended to refer to the following documents and information.

Table 1-1 Reference documents

Name	Description
GR551x Developer Guide	Introduces the software/hardware and quick start guide of GR551x SoCs.
Keil User Guide	Offers detailed Keil operational instructions. Available at www.keil.com/support/man/docs/uv4/.
FreeRTOS Documentation	Provides guidance on using FreeRTOS. Available at www.freertos.org/Documentation/RTOS_book.html.

2 Introduction to FreeRTOS Source Directory

FreeRTOS source code is in SDK_Folder\external\freertos, which contains the include folder, the portable folder, and the .c source files.

Note:

SDK_Folder is the root directory of GR551x SDK.

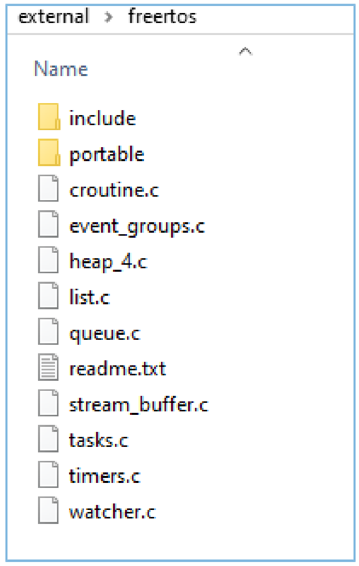

Figure 2-1 The freertos folder in GR551x SDK

- The include folder: It contains all FreeRTOS APIs, related structures, and macro definitions.
- The portable folder: It contains FreeRTOS code to be ported to GR551x SoCs with modifications.
- The .c source files: Implement core service code of FreeRTOS.

For more information about FreeRTOS, visit the FreeRTOS official website: www.freertos.org.

3 Initial Operation

This chapter introduces how to rapidly verify the FreeRTOS example in the GR551x SDK.

3.1 Preparation

Perform the following tasks before verifying a FreeRTOS example.

- **Hardware preparation**

Table 3-1 Hardware preparation

Name	Description
J-Link debug probe	JTAG emulator launched by SEGGER. For more information, visit www.segger.com/products/debug-probes/j-link/.
Development board	GR5515 Starter Kit Board (GR5515 SK Board)
Connection cable	Micro USB 2.0 serial cable

- **Software preparation**

Table 3-2 Software preparation

Name	Description
Windows	Windows 7/Windows 10
Keil MDK5	An integrated development environment (IDE). Available at www.keil.com/download/product/.
LightBlue (iOS)	An iOS Bluetooth Low Energy (Bluetooth LE) debugging tool. Available at the App Store.
GRToolbox (Android)	A Bluetooth LE debugging tool for GR551x. Available in `SDK_Folder\tools\GRToolbox`.
GRUart (Windows)	A GR551x serial port debugging tool. Available in `SDK_Folder\tools\GRUart`.
GProgrammer (Windows)	A GR551x programming tool. Available in `SDK_Folder\tools\GProgrammer`.

3.2 Hardware Connection

Connect a GR5515 SK Board to a PC with a Micro USB 2.0 cable.

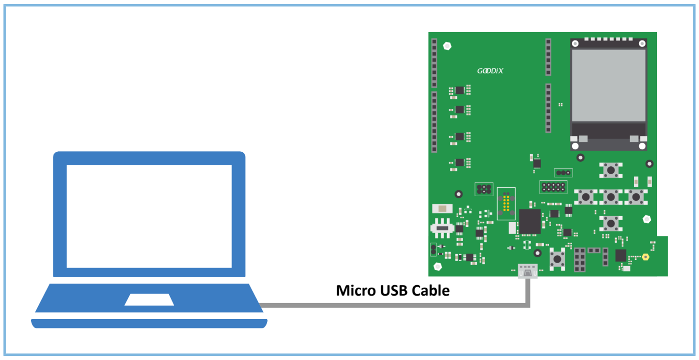

Figure 3-1 Hardware connection

3.3 Firmware Download

Download the firmware file *ble_app_template_freertos_fw.bin* of the FreeRTOS example to the GR5515 SK Board. For details, see *GProgrammer User Manual*.

Note:

The *ble_app_template_freertos_fw.bin* file is in `SDK_Folder\projects\ble\ble_peripheral\ble_app_template_freertos\build\`.

3.4 Test and Verification

Verify the FreeRTOS example by checking output information from GRUart.

3.4.1 Verification of FreeRTOS Features

To verify FreeRTOS task scheduling, follow the steps below:

1. Start GRUart, and configure the serial ports according to parameters in the table below.

Table 3-3 Configuring serial port parameters on GRUart

PortName	BaudRate	DataBits	Parity	StopBits	Flow Control
Select on demand	115200	8	None	1	Uncheck

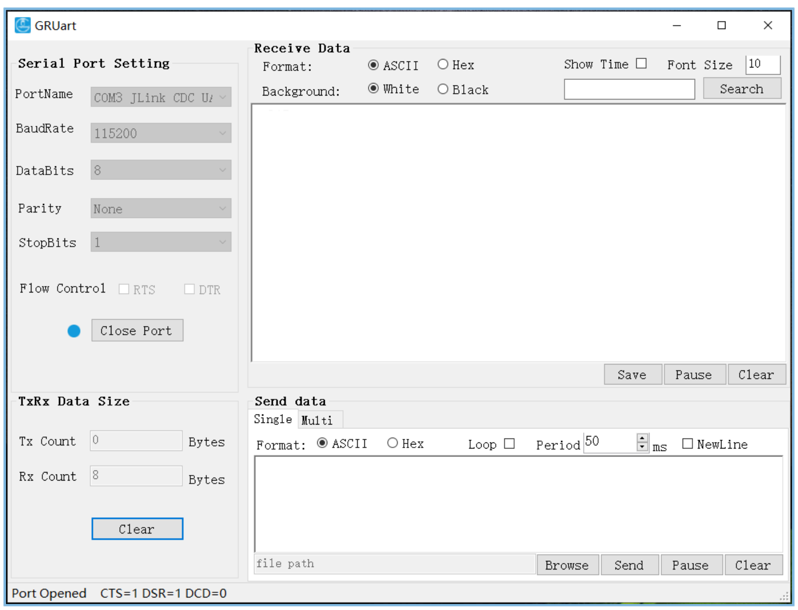

Figure 3-2 GRUart serial port configuration interface

2. Open the configured serial port, and check the trace results. If GRUart prints log information like **goodix print test task = ${N}** every other second in the **Receive Data** pane, the FreeRTOS system runs successfully.

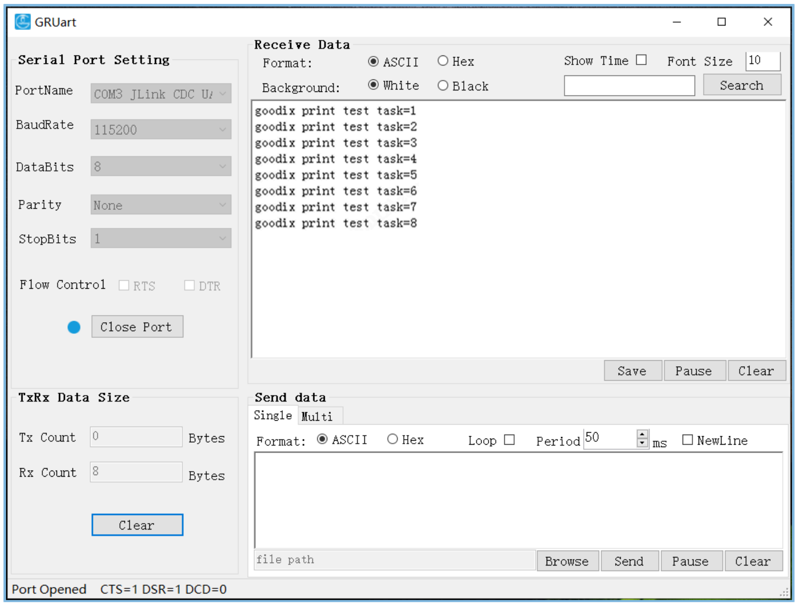

Figure 3-3 Operating results

3.4.2 Verification of Bluetooth Function

Verify the Bluetooth function of the FreeRTOS example with GRToolbox (Android).

Note:

For iOS devices, choose LightBlue for the verification.

Run GRToolbox and scan Bluetooth devices nearby. If **Goodix_Tem_OS** is in the device list, the FreeRTOS application runs normally.

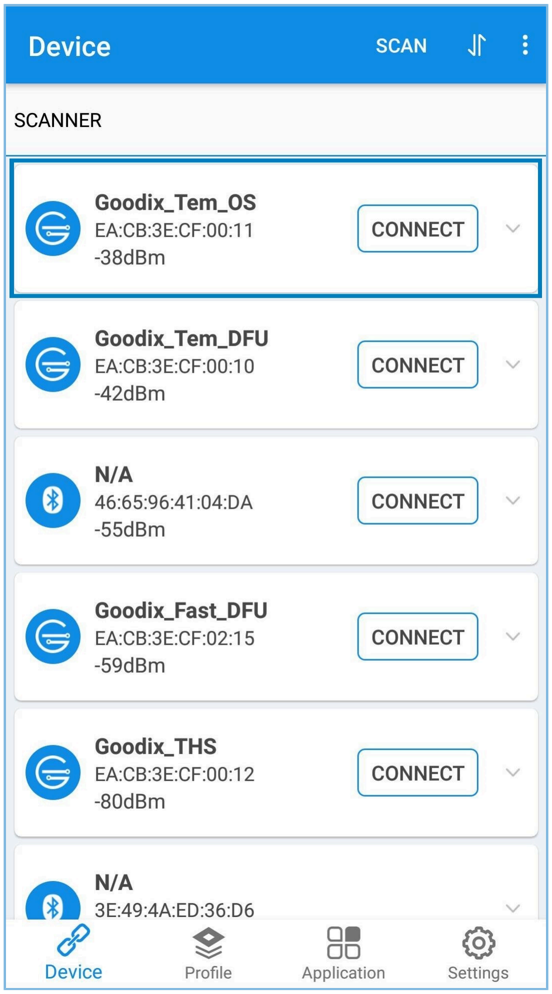

Figure 3-4 Discovering **Goodix_Tem_OS**

4 Application Details

Users can customize the FreeRTOS application by modifying configurations of the ble_app_template_freertos example. For example:

- Modify the FreeRTOS configurations.
- Modify the example program configurations.

4.1 Project Directory

The source code and project file of the FreeRTOS example are in `SDK_Folder\projects\ble\ble_peripheral\ble_app_template_freertos`, and project file is in the Keil_5 folder.

Double-click the project file *ble_app_template_freertos.uvprojx*, to view the ble_app_template_freertos project directory structure of the FreeRTOS example in Keil, as shown in the figure below.

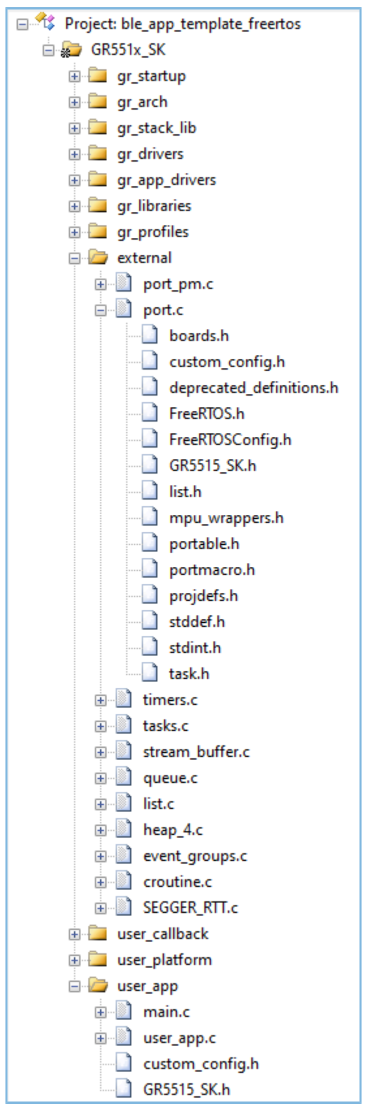

Figure 4-1 ble_app_template_freertos project directory

For related files, see the table below.

Table 4-1 File description of ble_app_template_freertos

Group	File	Description
external	port_pm.c	This file contains the FreeRTOS power management interface.
	port.c	This file contains FreeRTOS code to be ported to GR551x SoCs with modifications.
	heap_4.c	This file contains the FreeRTOS memory management policy.
user_app	main.c	This file contains core code of the FreeRTOS tests.
	user_app.c	This file defines Bluetooth advertising.

Note:

If you cannot expand *port.c*, open the ble_app_template_freertos example project in Keil, and then press F7 to compile the project, so that the *.c* file can display the referenced header files.

FreeRTOSConfig.h: It is a header file referenced by FreeRTOS source code, to configure FreeRTOS kernel.

4.2 Configuration

Users can customize the FreeRTOS memory management policy and the FreeRTOS kernel based on product requirements.

4.2.1 Memory Management Policy Configuration

The project adopts *heap_4.c* as the memory management policy. Users can replace the *heap_4.c* with other ones on demand.

FreeRTOS supports five memory management policies, which are implemented through *heap_1.c*, *heap_2.c*, *heap_3.c*, *heap_4.c*, and *heap_5.c* respectively. Information about each file is provided as follows:

Table 4-2 FreeRTOS memory management policy

Memory Management Policy Source File	Memory Management Characteristics
heap_1.c	• It is easy to be implemented with less code. • It supports memory application only, and does not permit memory to be released once the memory has been allocated.
heap_2.c	• Apply the optimum matching algorithm. • Allow releasing allocated memory blocks. • Do not merge adjacent free blocks, which may cause memory fragmentation. • Repeated applications and releases of memory cause memory fragmentation.

Memory Management Policy Source File	Memory Management Characteristics
heap_3.c	• Wrap malloc() and free() functions for thread safety. • Need to configure the heap size in the startup assembling file *startup_gr55xx.s*. • It requires checking the **Use MicroLIB** in the **Options for Target 'GR5515_SK'** pane of Keil; otherwise, this policy cannot work.
heap_4.c	• Apply the optimum matching algorithm. • Allow releasing allocated memory blocks. • Merge adjacent free memory blocks. • Repeated applications and releases of memory cause memory fragmentation.
heap_5.c	• Apply the optimum matching algorithm. • Allow releasing allocated memory blocks. • Merge adjacent free memory blocks. • Allow spanning memory heaps across multiple non-adjacent memory blocks. • Need to initialize memory heaps successively.

4.2.2 Kernel Configuration

FreeRTOS kernel is configured by the macro definitions in the *FreeRTOSConfig.h*, including configuration of the main clock frequency and the highest priority level of a task. Users can modify these macro definitions to customize a new kernel. Common macro definitions of FreeRTOS are shown in the table below:

Table 4-3 FreeRTOS common macro definitions

Macro Definition	Configuration
configUSE_IDLE_HOOK	1: Enable the HOOK function of idle tasks. 0: Disable the HOOK function of idle tasks.
configUSE_TICK_HOOK	1: Enable the Hook function of the TICK interrupt. 0: Disable the Hook function of the TICK interrupt.
configCPU_CLOCK_HZ	Define the main frequency of CPU (unit: Hz); the default value of the current platform is 64000000.
configTICK_RATE_HZ	Define the clock tick count of the system (unit: Hz); the default value of the current platform is 1000.
configMAX_PRIORITIES	Define the maximum priorities for users. If the maximum number is defined to 5, the priority levels available for users are 0, 1, 2, 3 and 4, excluding 5.
configMINIMAL_STACK_SIZE	Define the default minimum stack size for system tasks (unit: word); the default value of the current platform is 512 words (2,048 bytes in total).

Macro Definition	Configuration
configTOTAL_HEAP_SIZE	Refer to the memory pool capacity for memory management (unit: KB); the default value of the current platform is 32 KB. If dynamic APIs are used, the FreeRTOS kernel requests memory from the memory pool. The total memory shall be allocated on demand to avoid abnormal system operation.
configPRIO_BITS	Refer to bits occupied by the priority level set for the current platform (default value: 4).
configLIBRARY_LOWEST_INTERRUPT_PRIORITY	Refer to the lowest priority level supported by the current platform (default value: 15).
configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY	Define the highest priority level of interrupts under the FreeRTOS management. A smaller number indicates a higher priority level. If the number is set to 5, tasks at a priority level below 5 are beyond the control of FreeRTOS. In interrupt masking, interrupts at priority levels below 5 are not masked.

For more information about macro configurations, visit https://www.freertos.org/a00110.html.

4.3 Application Code

This section describes how to use code to create and initialize tasks.

4.3.1 Task Creation and Initialization

Path: `ble_app_template_freertos\Src\user\main.c`

Function: int main(void);

This is the main entry of applications. It enables peripheral initialization, BLE Protocol Stack initialization, FreeRTOS task creation, and FreeRTOS scheduling and startup.

```
int main(void)
{
    /*< Initialize user peripherals. */
    app_periph_init();

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

    /*< Create some demo tasks via freertos. */
    xTaskCreate(vStartTasks, "create_task", 512, NULL, 0, NULL);

    /*< FreeRTOS runs all tasks. */
    vTaskStartScheduler();

    /*< Never perform here */
    for (;;);
}
```

Path: `ble_app_template_freertos\Src\user\main.c`

Function: vStartTasks();

`Print_test_task` is created in this function. This task is responsible for printing information.

```
static void vStartTasks(void *arg)
{
    xTaskCreate(print_test_task, "print_task", APP_TASK_STACK_SIZE,
                NULL, configMAX_PRIORITIES - 1, NULL);
    xTaskCreate(dfu_schedule_task, "dfu_schedule_task", DFU_TASK_STACK_SIZE,
                NULL, configMAX_PRIORITIES - 2, NULL);
    vTaskDelete(NULL);
}
```

Path: `ble_app_template_freertos\Src\user\main.c`

Function: print_test_task();

This function implements cyclic printing at a 1-second latency. The vTaskDelay function is in units of millisecond.

```
static void print_test_task(void *arg)
{
    uint8_t index = 0;
    while (1)
    {
        APP_LOG_INFO("goodix print test task=%d\r\n", index++);
        app_log_flush();
        vTaskDelay(1000);
    }
}
```

4.3.2 Bluetooth LE Scheduling

This section introduces how BLE Protocol Stack and Bluetooth LE applications schedule tasks in FreeRTOS.

After entering the main() function, complete the following steps before performing FreeRTOS task scheduling:

1. Initialize hardware peripherals.
2. Implement required BLE_SDK_Callback interfaces for Bluetooth LE applications, and use these interfaces to initialize corresponding member variables in app_callback_t.
3. Apply for the memory block (heaps_table) required to run the BLE Protocol Stack.
4. Initialize BLE Protocol Stack.

After initialization, BLE Protocol Stack enables two interrupts: BLE_IRQ and BLE_SDK_IRQ.

- Notify the Bluetooth LE Event of BLE Protocol Stack to Bluetooth LE applications.

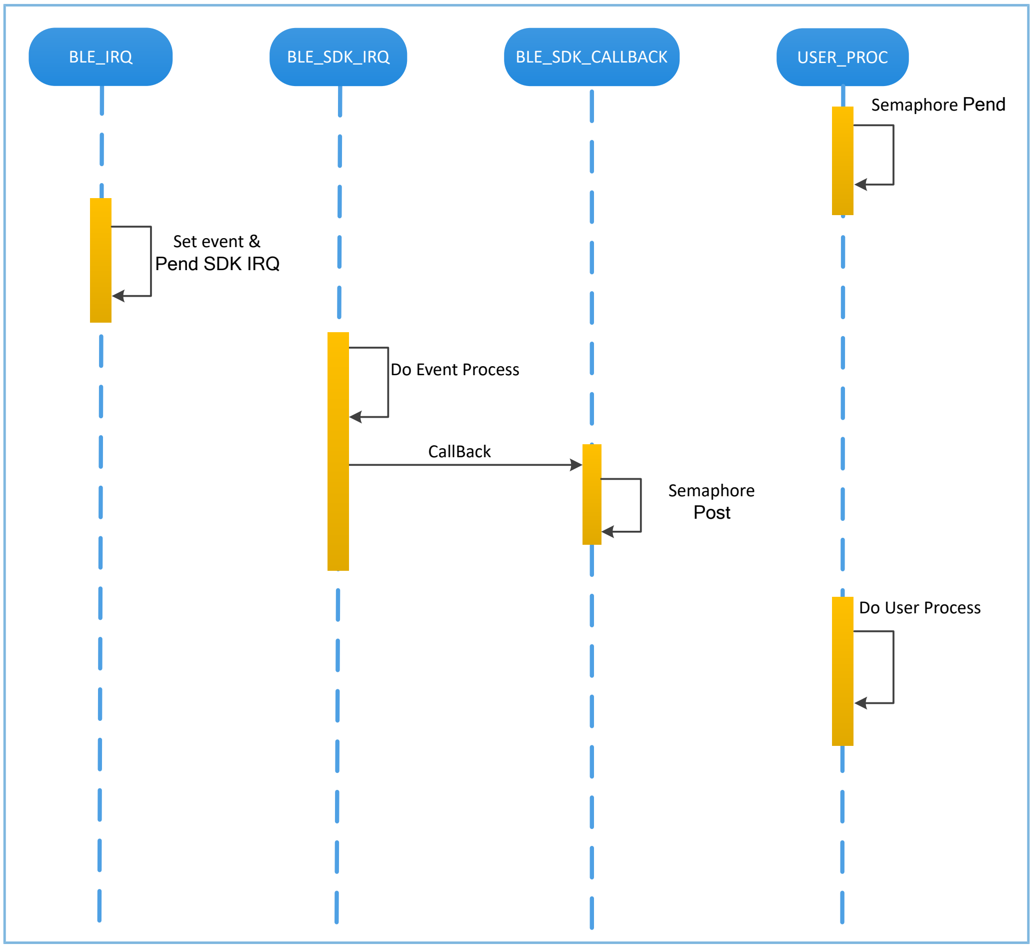

Figure 4-2 BLE Protocol Stack notifying Bluetooth LE applications of a Bluetooth LE Event

As shown in the figure above, when Bluetooth LE Baseband receives a data package, it triggers BLE_IRQ interrupt. BLE_IRQ_Handler generates a Bluetooth LE Event and sets the BLE_SDK_IRQ interrupt to Pending state. During BLE_SDK_IRQ_Handler execution, the Bluetooth LE Event is processed and Bluetooth LE applications are notified of part of the Bluetooth LE Event through the BLE_SDK_Callback function.

Recommendations for implementing BLE_SDK_Callback function:

1. The BLE_SDK_Callback function is called in the interrupt handling function (BLE_SDK_IRQ_Handler). Thus it is recommended not to perform long-running operation in the callback function; otherwise, implementation of user tasks may be delayed.

2. If any data or state information in the callback function requires timely processing by Bluetooth LE applications, it is recommended to use the semaphore mechanism to complete service logic processing in user tasks. This means you should wait for the semaphore (Pend) in user tasks, and release the semaphore (Post) in the callback function.

3. If the callback function contains a large amount of data, and requires long-time processing or ordered processing, developers are recommended to use the message queue to cache data and then transfer the data to other tasks for processing.

4. In the BLE_SDK_Callback function, call FreeRTOS APIs that end in “FromISR” if required, and forbid waiting for semaphore in the BLE_SDK_Callback function.

- Send requests from Bluetooth LE application layer to BLE Protocol Stack.

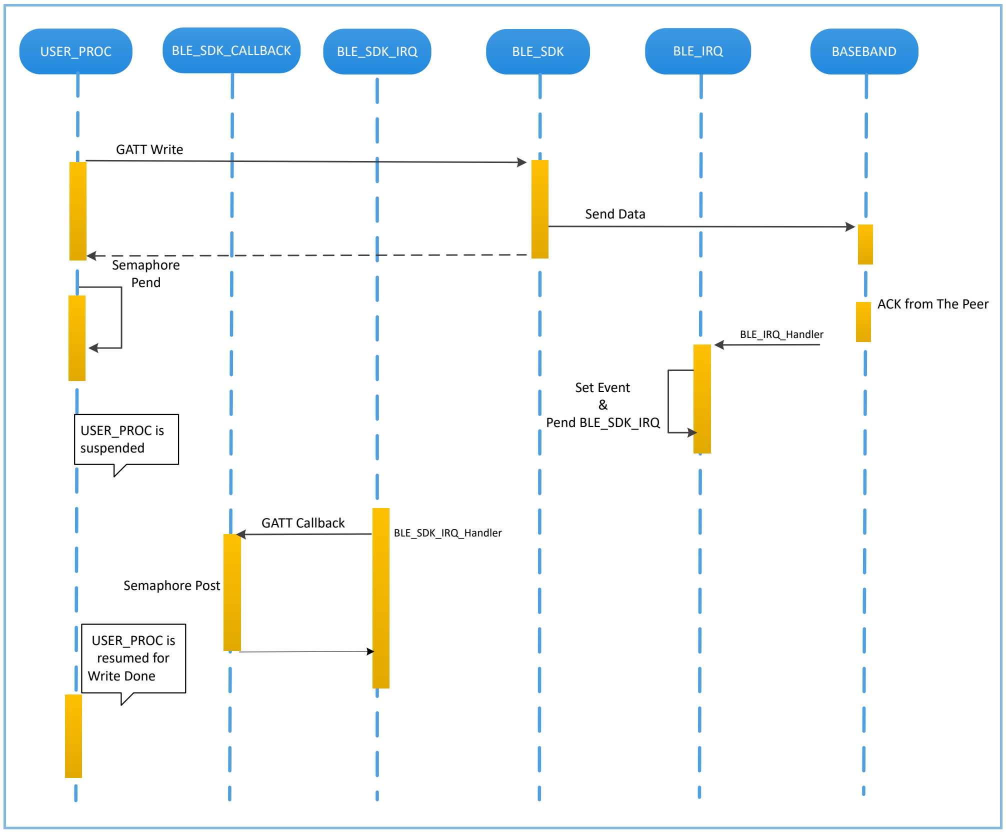

Figure 4-3 Processing of requests from Bluetooth LE applications to BLE Protocol Stack

As shown in the figure above, Bluetooth LE applications uses GATT APIs to write data to the peer device. This action requires interactions with the peer device, and the operating results cannot be obtained immediately. Bluetooth LE applications need to wait for the processing results from BLE Protocol Stack. Developers can use semaphore to convert an asynchronous function call to a synchronous function call according to service logic demands from Bluetooth LE applications:

1. Suspend the task by using the semaphore (Pend) interface after GATT APIs are called by user tasks.
2. BLE Protocol Stack waits for ACK from the peer device after sending the data from Bluetooth LE applications.
3. Bluetooth LE Baseband triggers the BLE_IRQ interrupt after receiving ACK from the peer device.
4. BLE_IRQ_Handler generates a Bluetooth LE Event and sets the BLE_SDK_IRQ interrupt to Pending state.
5. The Bluetooth LE Event is processed, and the BLE_SDK_Callback function is called during BLE_SDK_IRQ_Handler execution.
6. Implement the semaphore (Post) interface in the BLE_SDK_Callback function to release the blocked semaphore.

By then, implementation of user tasks resumes and data writes are done.

Generally, developers only need to focus on functions at the application layer, and how to implement callback functions to enable interaction with users. BLE Protocol Stack is transparent to developers. For GR551x SDK programming model, see *GR551x Developer Guide*.

5 FAQs

This chapter describes possible problems, reasons, and solutions during verification and application of the FreeRTOS example.

5.1 Why Is There No Output Information from GRUart?

- Description

  There is no output information from GRUart when the on-board program is running.

- Analysis

  Serial ports are set incorrectly. For example, if the baud rate is wrong, the serial port tool cannot correctly display the data received.

- Solution

  Check whether the serial cable is connected correctly, whether the COM port number is correct, and whether the baud rate is set correctly according to Table 3-3. It is recommended to first use the SDK default firmware to detect the development environment.

5.2 Why does the Mobile Phone Discover No Bluetooth Advertising?

- Description

  A mobile phone cannot discover advertising when the on-board program is running.

- Analysis

  The firmware cannot run normally, resulting in no Bluetooth advertising.

- Solution

  Try to reset or re-download the default firmware, and check the antennas.

