



GR551x FreeRTOS示例手册

版本： 1.6

发布日期： 2020-06-30

版权所有 © 2020 深圳市汇顶科技股份有限公司。保留一切权利。

非经本公司书面许可，任何单位和个人不得对本手册内的任何部分擅自摘抄、复制、修改、翻译、传播，或将其全部或部分用于商业用途。

商标声明

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

免责声明

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

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

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

深圳市汇顶科技股份有限公司

总部地址：深圳市福田区腾飞工业大厦B座2层、13层

电话：+86-755-33338828 传真：+86-755-33338830

网址：www.goodix.com

前言

编写目的

本文档介绍如何使用和修改GR551x SDK中的FreeRTOS示例，旨在帮助用户快速进行二次开发。

读者对象

本文适用于以下读者：

- GR551x用户
- GR551x开发人员
- GR551x测试人员
- 开发爱好者
- 文档工程师

版本说明

本文档为第4次发布，对应的产品系列为GR551x。

修订记录

版本	日期	修订内容
1.0	2019-12-08	首次发布
1.3	2020-03-16	更新了文档页脚版本时间
1.5	2020-05-30	更新了“4.1 工程目录”章节中工程目录图片
1.6	2020-06-30	基于SDK刷新版本

目录

前言.....	1
1 简介.....	1
2 FreeRTOS源码目录介绍.....	2
3 初次运行.....	3
3.1 准备工作.....	3
3.2 硬件连接.....	3
3.3 下载固件.....	4
3.4 测试验证.....	4
3.4.1 验证FreeRTOS功能.....	4
3.4.2 验证蓝牙功能.....	5
4 应用详解.....	6
4.1 工程目录.....	6
4.2 配置介绍.....	7
4.2.1 配置内存管理策略.....	7
4.2.2 配置内核.....	8
4.3 应用代码介绍.....	9
4.3.1 任务创建及初始化.....	9
4.3.2 BLE调度详解.....	10
5 常见问题.....	13
5.1 串口无打印.....	13
5.2 蓝牙无广播.....	13

1 简介

FreeRTOS是一个开源的（MIT License）、轻量级的嵌入式实时操作系统，占用较少的RAM/ROM资源，具有可移植、可裁减、调度策略灵活的特点。该系统包含任务管理、时间管理、信号量、消息队列、内存管理等功能。

本文档介绍GR551x SDK中的FreeRTOS移植示例，包括示例的使用方法以及关键源码的说明。

在使用和修改FreeRTOS示例前，建议参考以下文档和信息。

表 1-1 文档参考

名称	描述
GR551x开发者指南	GR551x软硬件介绍、快速使用及资源总览
Keil用户指南	Keil详细操作说明： www.keil.com/support/man/docs/uv4/
FreeRTOS Documentation	FreeRTOS的使用方法： www.freertos.org/Documentation/RTOS_book.html

2 FreeRTOS源码目录介绍

FreeRTOS源码位于目录SDK_Folder\external\freertos，包括include文件夹、portable文件夹和.c源文件。

 说明:

SDK_Folder是GR551x SDK的根目录。

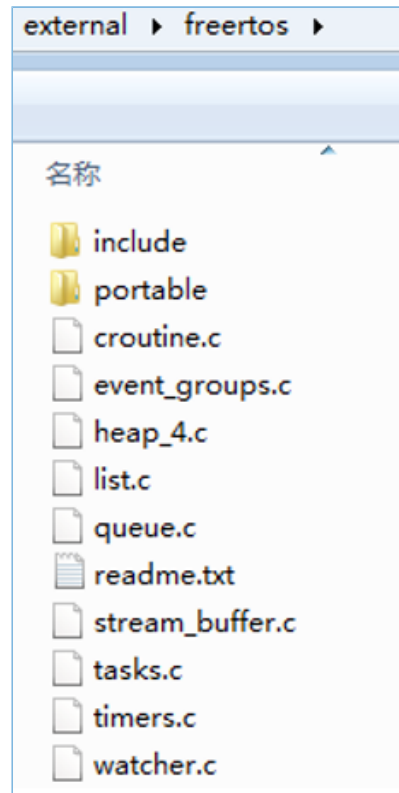


图 2-1 GR551x SDK中的freertos文件夹

- include目录：FreeRTOS全部API，相关结构体和宏定义。
- portable目录：GR551x平台移植代码。
- .c源文件：FreeRTOS的核心业务代码实现。

如需了解FreeRTOS的详细介绍，请访问FreeRTOS官网www.freertos.org。

3 初次运行

本章介绍如何快速验证GR551x SDK中的FreeRTOS示例。

3.1 准备工作

验证FreeRTOS示例之前，请完成以下准备工作。

- 硬件准备

表 3-1 硬件准备

名称	描述
J-Link工具	SEGGER公司推出的JTAG仿真器，如需更多了解，请访问 www.segger.com/products/debug-probes/j-link/
开发板	GR5515 Starter Kit开发板
连接线	Micro USB 2.0串口连接线

- 软件准备

表 3-2 软件准备

名称	描述
Windows	Windows 7/Windows 10操作系统
Keil MDK5	IDE工具，下载网址： www.keil.com/download/product/
Lightblue (iOS)	iOS BLE (Bluetooth Low Energy) 调试工具，可通过App Store下载
GRToolbox (Android)	GR551x BLE调试工具，位于SDK_Folder\tools\GRToolbox
GRUart (Windows)	GR551x串口调试工具，位于SDK_Folder\tools\GRUart
GProgrammer (Windows)	GR551x Programming工具，位于SDK_Folder\tools\GProgrammer

3.2 硬件连接

使用Micro USB 2.0数据线连接GR5515 Starter Kit开发板与计算机。

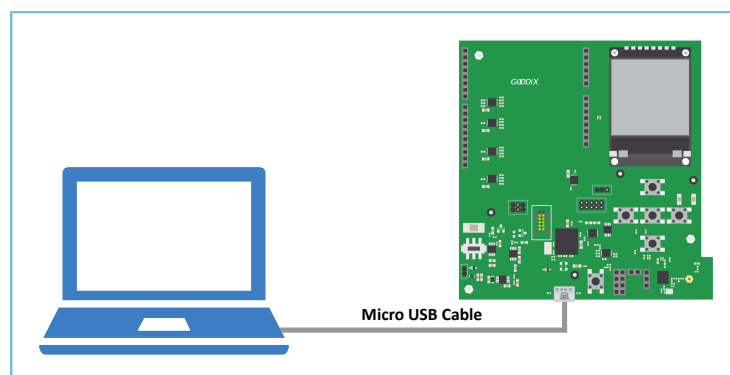



图 3-1 硬件连接示意图

3.3 下载固件

下载FreeRTOS示例的`ble_app_template_freertos_fw.bin`固件至开发板。具体操作方法，请参考《GProgrammer用户手册》。

 说明:

`ble_app_template_freertos_fw.bin`位于`SDK_Folder\projects\ble\ble_peripheral\ble_app_template_freertos\build\`。

3.4 测试验证

通过检查串口打印信息来验证FreeRTOS示例。

3.4.1 验证FreeRTOS功能

FreeRTOS任务调度验证步骤如下:

1. 启动GRUart，按照表 3-3 中的参数配置串口。

表 3-3 GRUart串口配置参数

PortName	BaudRate	DataBits	Parity	StopBits	Flow Control
需根据实际选择	115200	8	None	1	不勾选

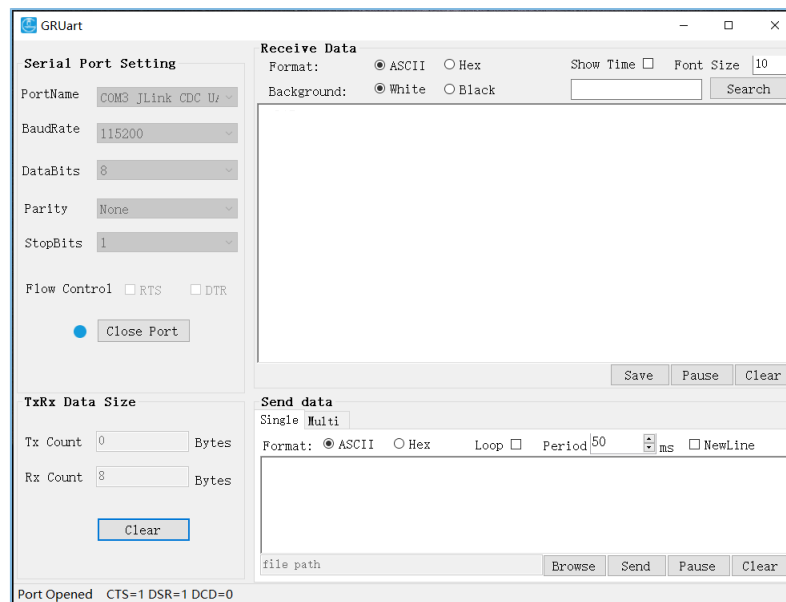


图 3-2 GRUart串口配置

2. 打开配置的串口，查看运行结果。在GRUart的Receive Data窗口中，如果每隔一秒打印一行“goodix print test task=累加的序号”，则表示FreeRTOS系统运行正常。

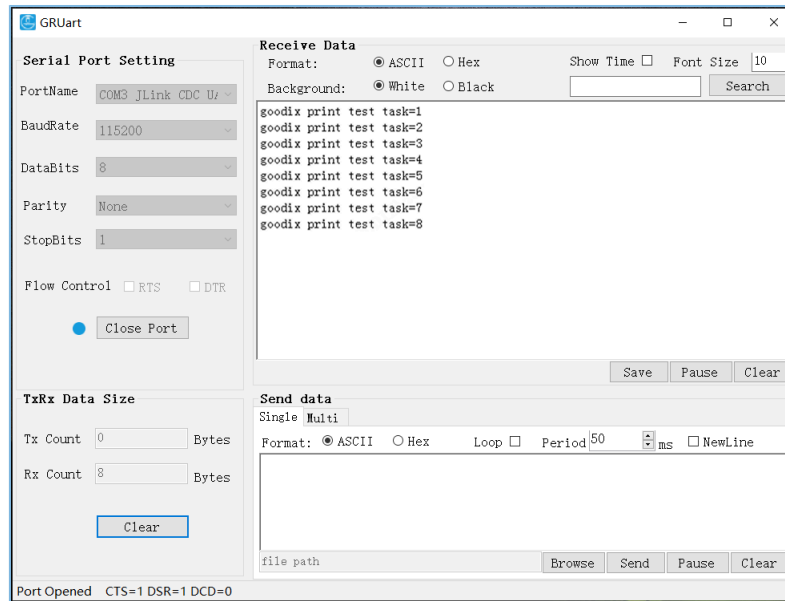



图 3-3 运行结果

3.4.2 验证蓝牙功能

利用GRToolbox（Android）可验证FreeRTOS工程示例的蓝牙功能。

 说明:

也可选择LightBlue（iOS）来测试验证。

启动GRToolbox，扫描周边蓝牙设备。如果在设备列表中发现名为“Goodix_Tem_OS”的广播设备，则表示FreeRTOS应用运行正常。

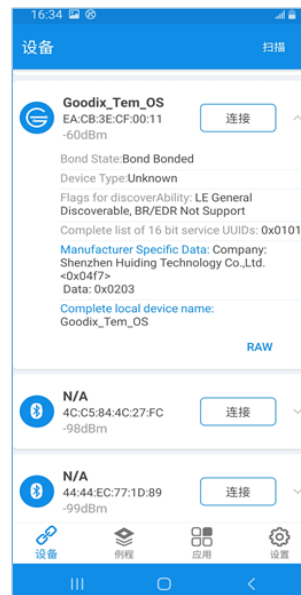


图 3-4 扫描到蓝牙设备Goodix_Tem_OS

4 应用详解

通过修改ble_app_template_freertos示例的相关配置，用户可自定义FreeRTOS应用，如：

- 修改FreeRTOS配置
- 修改示例程序配置

4.1 工程目录

FreeRTOS示例的源代码和工程文件位于：SDK_Folder\projects\ble\ble_peripheral\ble_app_template_freertos，其中工程文件位于Keil_5文件夹。

双击打开ble_app_template_freertos.uvprojx工程文件，在Keil中查看FreeRTOS示例ble_app_template_freertos工程目录结构，如图4-1所示。

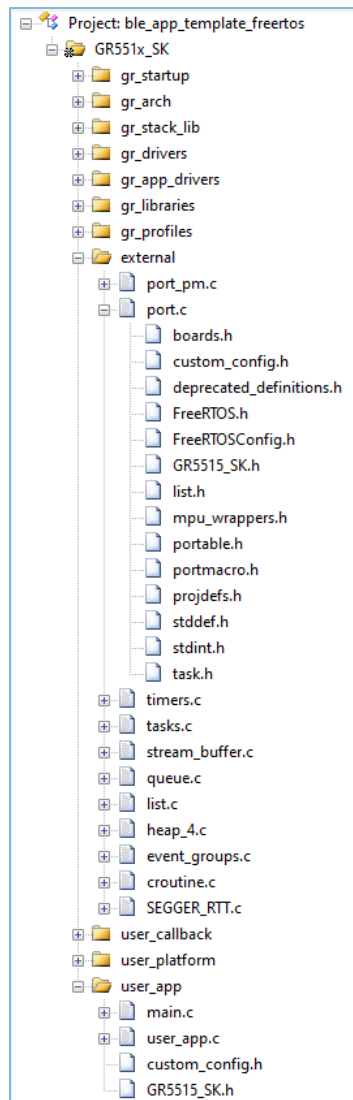


图 4-1 ble_app_template_freertos工程目录

相关文件说明如表 4-1 所示。

表 4-1 ble_app_template_freertos文件说明

Group	文件	描述
external	port_pm.c	FreeRTOS电源管理接口
	port.c	GR551x平台移植代码
	heap_4.c	FreeRTOS内存管理策略
user_app	main.c	包括FreeRTOS测试的核心代码
	user_app.c	定义蓝牙广播

说明:

如果`port.c`不能展开, 请使用Keil打开`ble_app_template_freertos`示例工程, 按F7进行编译, 使.c文件能显示引用的头文件。

`FreeRTOSConfig.h`: FreeRTOS的源码引用的.h头文件, 用于配置FreeRTOS内核。

4.2 配置介绍

用户可根据产品需求自定义FreeRTOS的内存管理策略和FreeRTOS内核。

4.2.1 配置内存管理策略

本工程的内存管理策略采用`heap_4.c`。用户可将`heap_4.c`替换为自己所需的内存管理策略源文件。

FreeRTOS支持5个内存管理策略, 分别由`heap_1.c`, `heap_2.c`, `heap_3.c`, `heap_4.c`和`heap_5.c`来实现。各内存管理策略源文件的描述如下:

表 4-2 FreeRTOS内存管理策略

内存管理策略源文件	内存管理特点
heap_1.c	<ul style="list-style-type: none"> 实现相对简单, 代码量比较小。 只能申请内存, 申请成功后就不能被释放。
Heap_2.c	<ul style="list-style-type: none"> 使用最佳匹配算法。 允许释放已分配的内存块。 不合并相邻空闲块, 可能造成内存碎片。 多次申请和释放内存资源会造成内存碎片。
Heap_3.c	<ul style="list-style-type: none"> 封装<code>malloc()</code>和<code>free()</code>函数, 使得他们具备线程保护。 需在启动汇编文件<code>startup_gr55xx.s</code>中配置heap大小。 Keil集成环境Options for Target窗口的“Target”选项, 需勾选“Use MicroLIB”, 否则无法正常使用。
Heap_4.c	<ul style="list-style-type: none"> 使用最佳匹配算法。 允许释放已分配的内存块。 合并相邻的空闲内存块。

内存管理策略源文件	内存管理特点
Heap_5.c	<ul style="list-style-type: none"> • 多次申请和释放内存资源会造成内存碎片。 • 使用最佳匹配算法。 • 允许释放已分配的内存块。 • 合并相邻的空闲内存块。 • 允许内存堆跨越多个非连续的内存区。 • 需要依次分别初始化内存堆。

4.2.2 配置内核

FreeRTOS内核由*FreeRTOSConfig.h*头文件中的宏定义进行配置，包括配置主时钟频率，最大任务优先级等。用户可修改这些宏定义来制定新的内核。FreeRTOS的常见宏定义如表 4-3 所示：

表 4-3 FreeRTOS常见宏定

宏定义	配置
configUSE_IDLE_HOOK	<p>1: 使能IDLE任务的HOOK函数。</p> <p>0: 禁用IDLE任务的HOOK函数。</p>
configUSE_TICK_HOOK	<p>1: 使能TICK中断执行HOOK函数。</p> <p>0: 禁用TICK中断执行HOOK函数。</p>
configCPU_CLOCK_HZ	定义CPU的主频，单位Hz，当前平台默认为64000000。
configTICK_RATE_HZ	定义系统时钟节拍数，单位Hz，当前平台默认为1000。
configMAX_PRIORITIES	<p>定义可供用户使用的最大优先级数。</p> <p>如果这个定义的是5，那么用户可以使用的优先级号是0、1、2、3、4，不包含5。</p>
configMINIMAL_STACK_SIZE	定义系统任务默认最小使用的堆栈大小，单位word，当前平台默认为512 words（共2048 bytes）。
configTOTAL_HEAP_SIZE	<p>该宏表示内存管理所使用的内存池容量大小，单位KB，当前平台默认为32 KB。</p> <p>如果使用动态API，所有FreeRTOS内核中将会从该内存池中申请内存。需要根据实际情况分配总量，避免引起系统运行时异常。</p>
configPRIO_BITS	表示当前平台优先级设定占用比特位数，当前平台默认为4。
configLIBRARY_LOWEST_INTERRUPT_PRIORITY	表示当前平台支持的最小优先级，当前平台默认为15。
configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY	<p>定义FreeRTOS管理的最高优先级中断。数字越小，优先级越高。</p> <p>如果设置为5，那么小于5的优先级是不受FreeRTOS管控。在屏蔽中断时，小于5的优先级不被屏蔽。</p>

更多宏配置，请参考<https://www.freertos.org/a00110.html>。

4.3 应用代码介绍

本节主要介绍如何使用代码实现任务的创建和初始化。

4.3.1 任务创建及初始化

路径: ble_app_template_freertos\Src\user\main.c

函数: int main(void);

该函数为应用程序主入口, 实现外设初始化、BLE协议栈初始化、FreeRTOS任务创建以及FreeRTOS调度启动。

```
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 (;;)
}
```

路径: ble_app_template_freertos\Src\user\main.c

函数: vStartTasks();

该函数中创建了“Print_test_task”。该任务主要负责打印信息。

```
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);
}
```

路径: ble_app_template_freertos\Src\user\main.c

函数: print_test_task();

该函数实现以1秒为延时的循环打印。其中vTaskDelay函数的单位为毫秒。

```
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 BLE调度详解

本节介绍在FreeRTOS下，BLE协议栈与BLE应用如何进行调度。

进入main()函数后，在执行FreeRTOS的任务调度之前，需要完成：

1. 实现各种硬件外设的初始化。
2. 实现BLE应用需要的若干BLE_SDK_Callback接口，并用这些接口函数初始化结构体app_callback_t中对应的成员变量。
3. 声明运行BLE协议栈需使用的内存（heaps_table）。
4. 完成BLE协议栈的初始化。

BLE协议栈初始化后会启用BLE_IRQ以及BLE_SDK_IRQ两个中断。

- 将BLE协议栈的BLE Event通知给BLE应用。

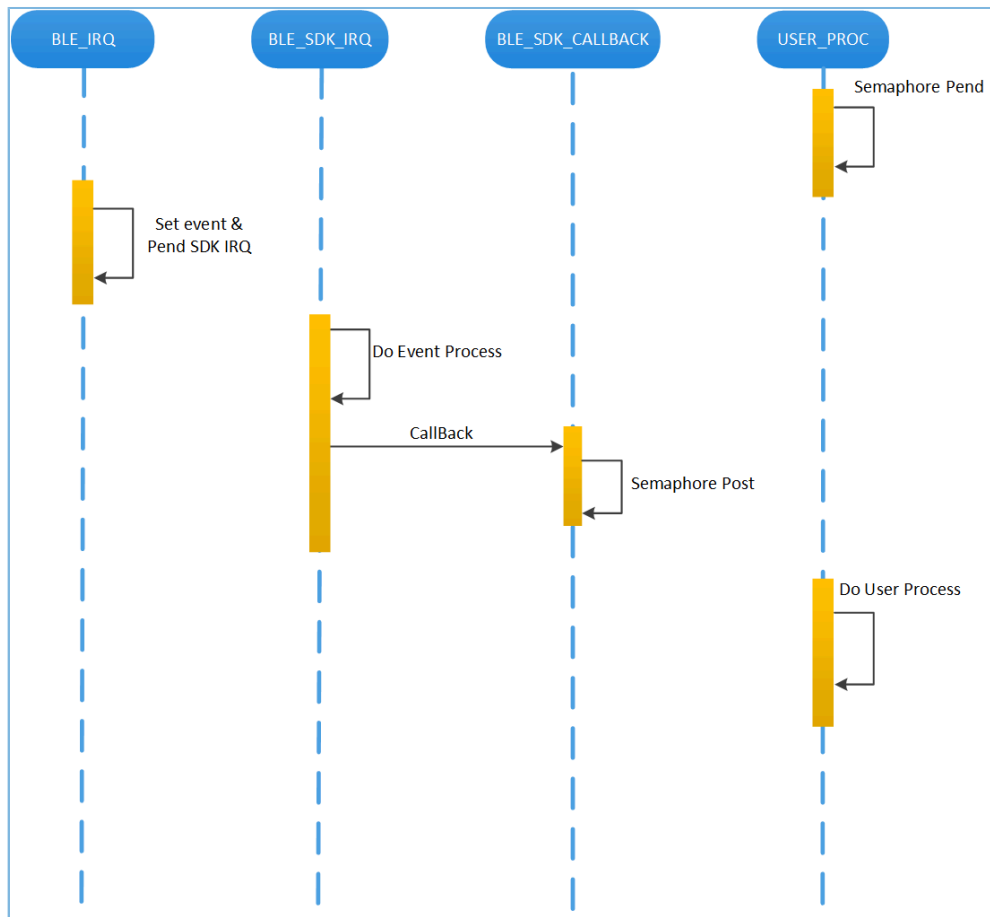


图 4-2 BLE协议栈将BLE Event通知给BLE应用

在图 4-2中，当BLE Baseband收到数据包后，会触发BLE_IRQ中断，BLE_IRQ_Handler产生BLE Event并将BLE_SDK_IRQ中断置为Pending态；在BLE_SDK_IRQ_Handler执行期间，BLE Event会被处理，并通过BLE_SDK_Callback函数将部分BLE Event通知到BLE应用。

关于BLE_SDK_Callback函数实现的建议：

1. Callback函数是在中断处理函数（BLE_SDK_IRQ_Handler）中被调用的，因此建议不要在Callback函数进行耗时操作，否则可能会延迟用户任务的执行。
 2. 如果Callback函数中有需要BLE应用及时处理的数据或状态信息，建议使用信号量机制在用户任务中完成业务逻辑的处理，即在用户任务等待信号量（Pend），在Callback函数中释放（Post）信号量。
 3. 如果Callback函数中的数据较多、处理较耗时、需有序处理，建议开发者采用消息队列将数据缓存后转由其他任务处理。
 4. 在BLE_SDK_Callback函数中，如果需要使用系统API，则请调用FreeRTOS中FromISR结尾的API，且禁止在BLE_SDK_Callback函数中执行等待信号量的操作。
- 将BLE应用层的请求发给BLE协议栈。

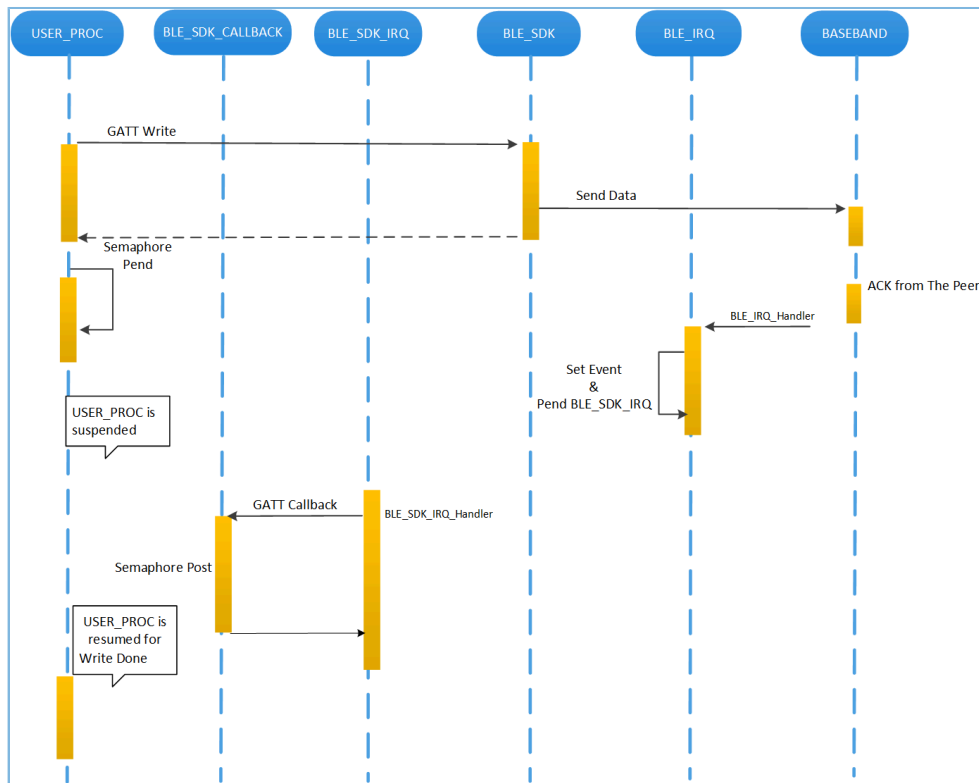


图 4-3 处理BLE应用向BLE协议栈发起的请求

在图 4-3 中，BLE 应用使用 GATT API 完成 WRITE 数据到对端设备。该操作涉及到与对端设备交互，且并不能立刻获得操作结果。BLE 应用需要等待 BLE 协议栈的处理结果。开发者可根据 BLE 应用的业务逻辑需求，使用信号量将异步函数调用转换为同步函数调用：

1. 用户任务调用 GATT API 后，使用信号量（Pend）接口将该任务挂起。
2. BLE 协议栈将来自 BLE 应用的数据发送完成后，等待对端的 Ack。
3. BLE Baseband 收到来自对端的 Ack 后，会触发 BLE_IRQ 中断。
4. BLE_IRQ_Handler 产生 BLE Event 并将 BLE_SDK_IRQ 中断置为 Pending 态。
5. 在 BLE_SDK_IRQ_Handler 执行期间，该 BLE Event 会被处理，并调用对应的 BLE_SDK_Callback 函数。
6. 在 BLE_SDK_Callback 函数中执行（Post）信号量操作来释放阻塞的信号量。

此时用户任务恢复运行，获得 WRITE 数据的操作结果。

一般情况下，开发者只需要关心应用层功能以及与用户进行交互的 Callback 函数接口的业务逻辑实现，BLE 协议栈对开发者而言是透明的。GR551x SDK 编程模型，请参考《GR551x 开发者指南》。

5 常见问题

本章描述了在验证及应用FreeRTOS示例时，可能出现的问题、原因及处理方法。

5.1 串口无打印

- 问题描述

程序运行后，串口终端无任何打印信息。

- 问题分析

串口设置存在问题，如波特率不正确，则串口工具将不能正确显示收到的数据。

- 处理方法

请检查串口线连接是否正常，COM端口号是否选择正确，波特率等是否按照表 3-3 配置，建议先使用SDK默认固件进行开发环境检测。

5.2 蓝牙无广播

- 问题描述

程序运行后，手机无法搜索到广播。

- 问题分析

因固件没有正常运行，必定导致蓝牙没有广播。

- 处理方法

可以尝试复位或重新下载默认固件，同时检查天线端情况。