GR55xx FreeRTOS示例手册

版本： **1.8**

发布日期： **2021-04-20**

深圳市汇顶科技股份有限公司

前言

编写目的

本文档介绍如何使用和修改GR55xx 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：

- GR55xx 用户
- GR55xx 开发人员
- GR55xx 测试人员
- 开发爱好者
- 文档工程师

版本说明

本文档为第6次发布，对应的产品系列为GR55xx 。

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	更新文档页脚版本时间
1.5	2020-05-30	更新“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
1.7	2020-12-15	更新GRToolbox软件界面截图
1.8	2021-04-20	优化“初次运行”和“应用详解”章节

目录

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GR55xx 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。

在进行操作前，可参考以下文档。

表 1-1 文档参考

名称	描述
对应GR55xx系列的开发者指南	GR55xx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用户当前所使用的GR55xx系列SDK的根目录。

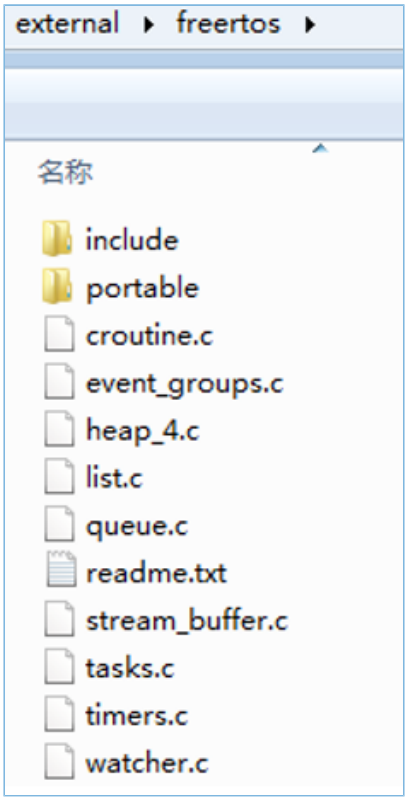

图 2-1 GR55xx SDK中的freertos文件夹

- include目录：FreeRTOS全部API，相关结构体和宏定义。
- portable目录：GR55xx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GR55xx SDK中的FreeRTOS示例。

3.1 支持平台

FreeRTOS示例工程支持下列开发平台。

表 3-1 支持开发平台

硬件平台	开发板型号
GR551x开发套件	GR5515-SK-BASIC

3.2 固件烧录

用户可使用GProgrammer将*ble_app_template_freertos_fw.bin*固件烧录至开发板。GProgrammer烧录固件的具体操作方法，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\`。
- GProgrammer位于`SDK_Folder\tools\GProgrammer`。

3.3 测试验证

测试FreeRTOS示例所需软硬件如表 3-2 所示。

表 3-2 测试所需软硬件

名称	描述
Android手机	Android 4.4（KitKat）及以上版本
iOS设备	支持BLE 4.0及以上的iOS设备，如iPhone 4s及其以上版本、iPad 3及其以上版本
GRToolbox（iOS）	BLE调试工具，可通过App Store下载
GRToolbox（Android）	BLE调试工具，位于`SDK_Folder\tools\GRToolbox`
GRUart（Windows）	串口调试工具，位于`SDK_Folder\tools\GRUart`

说明:

本文中GRToolbox的截图仅供用户了解操作步骤，实际界面请参考最新版本GRtoolbox。

当准备好GR55xx SK开发板和测试所需软件后即可进行FreeRTOS示例的测试验证，本示例的测试内容包括以下两方面：

- FreeRTOS功能

- 蓝牙功能

1. 验证FreeRTOS功能

   启动GRUart，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。

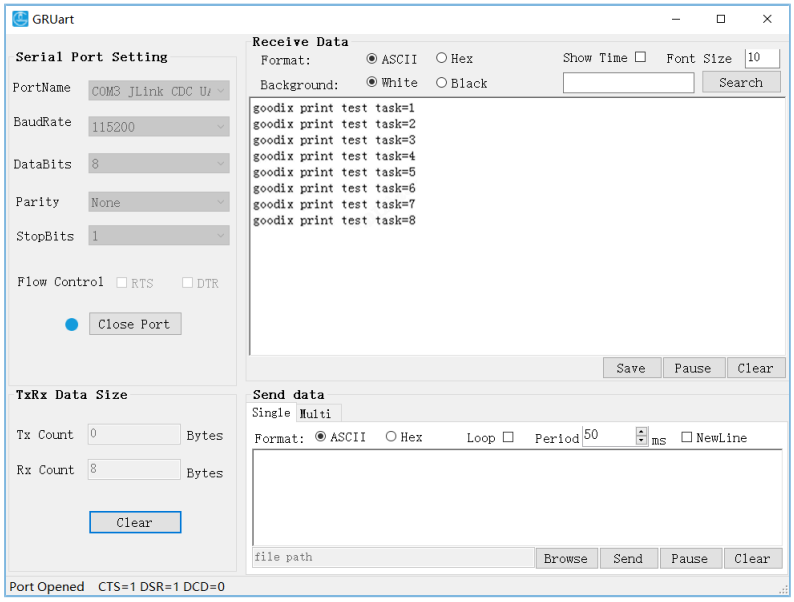

图 3-1 运行结果

2. 验证蓝牙功能

   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 3-2 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

本章将介绍ble_app_template_freertos示例的配置和关键代码。

4.1 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。

4.1.1 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 4-1 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	• 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	• 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	• 封装malloc()和free()函数，使得他们具备线程保护。 • 需在启动汇编文件*startup_gr55xx.s*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	• 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。 • 多次申请和释放内存资源会造成内存碎片。
Heap_5.c	• 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。

说明:

内存管理策略源文件位于`SDK_Folder\external\freertos\portable\MemMang`。

4.1.2 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 4-2 所示：

表 4-2 FreeRTOS常见宏定

宏定义	配置
configUSE_IDLE_HOOK	1：使能IDLE任务的HOOK函数。 0：禁用IDLE任务的HOOK函数。
configUSE_TICK_HOOK	1：使能TICK中断执行HOOK函数。 0：禁用TICK中断执行HOOK函数。
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	定义可供用户使用的最大优先级数。 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。
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	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。 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。
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	定义FreeRTOS管理的最高优先级中断。数字越小，优先级越高。 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。

说明:

- *FreeRTOSConfig.h*位于`SDK_Folder\app\projects\ble\ble_peripheral\ble_app_template_freertos\Src\user`。
- 更多宏配置，请参考https://www.freertos.org/a00110.html。

4.2 关键代码

本节主要介绍如何使用代码实现任务的创建以及BLE调度。

4.2.1 任务创建

- 创建任务

  在本示例工程中创建的任务是print_test_task，该任务主要负责打印信息。

  路径：ble_app_template_freertos\Src\user\main.c

  函数：vStartTasks();

```
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);
    }
}
```

4.2.2 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协议栈初始化后会使能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。

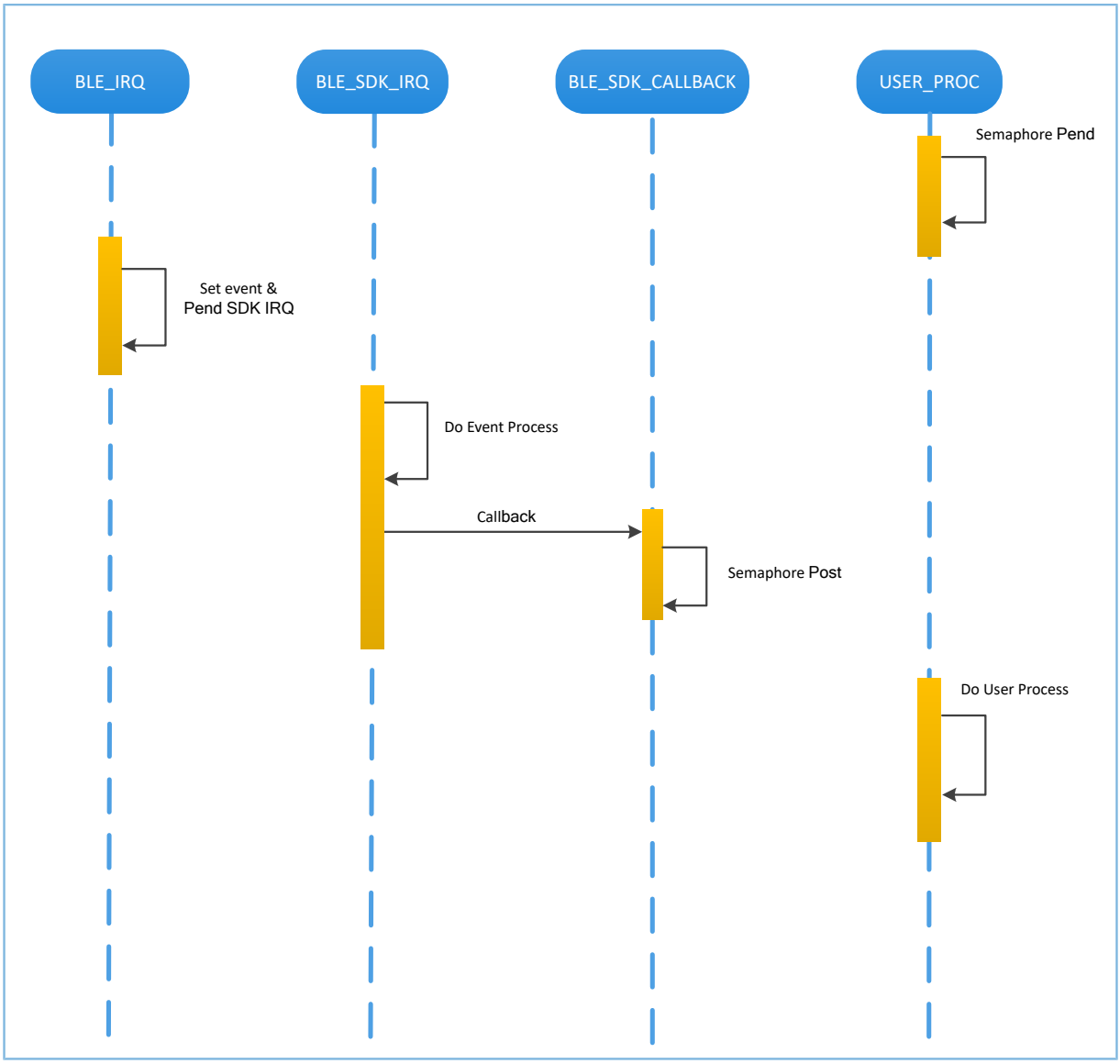

图 4-1 BLE协议栈将BLE Event通知给BLE应用

在图 4-1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。

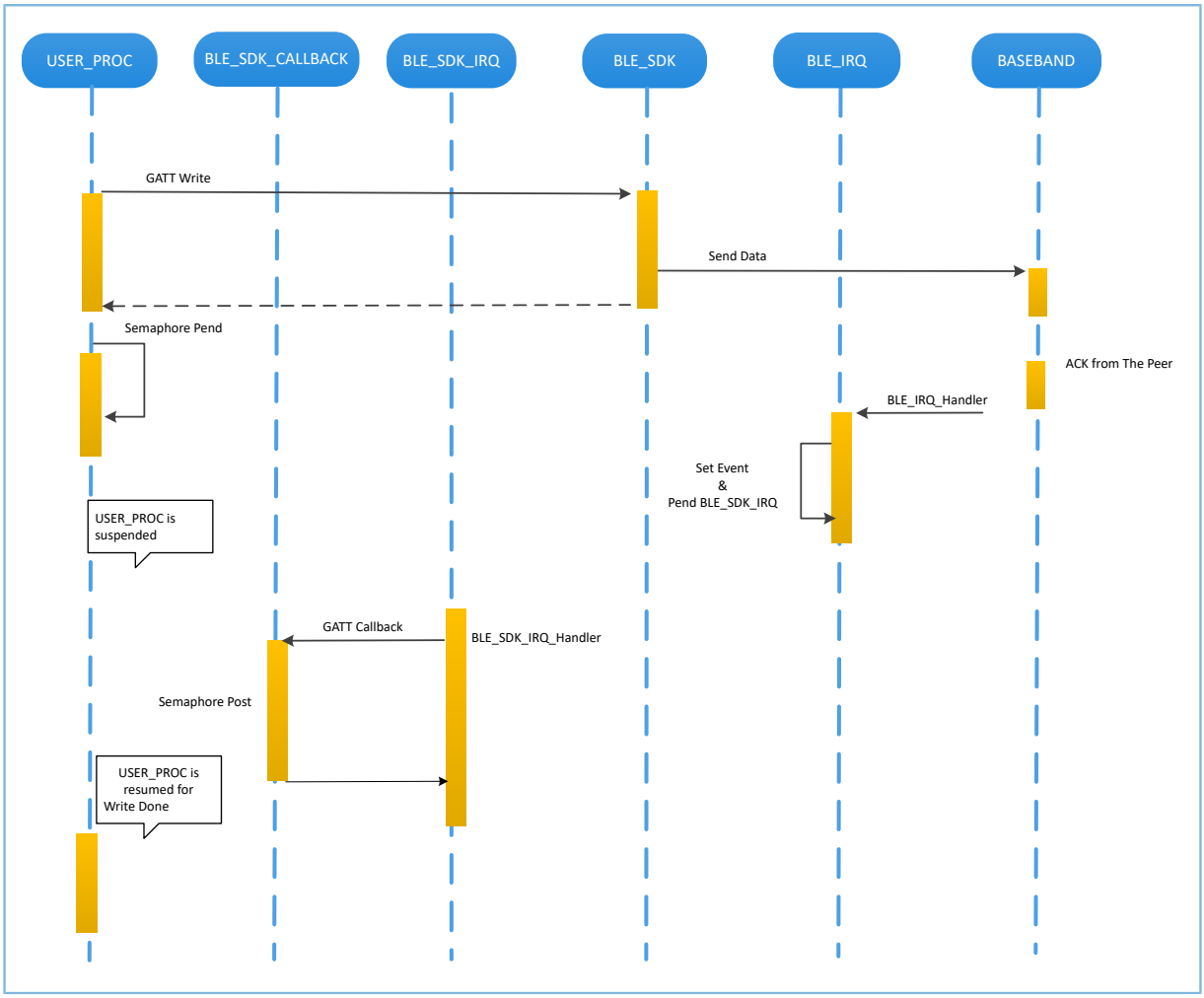

图 4-2 处理BLE应用向BLE协议栈发起的请求

在图 4-2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。GR55xx SDK编程模型，请参考用户当前所使用硬件平台对应的开发者指南。

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，波特率等是否正确配置，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。