

# GR551x开发者指南

版本: 2.0

发布日期: 2021-01-05

深圳市汇顶科技股份有限公司

#### 版权所有 © 2021 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

**G@DiX**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,不得暗中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: 深圳市福田保税区腾飞工业大厦B座2层、13层

电话: +86-755-33338828 传真: +86-755-33338099

网址: <u>www.goodix.com</u>

# 前言

#### 编写目的

本文档主要介绍了Goodix GR551x低功耗蓝牙片上系统(SoC)的软件开发工具包(SDK),以及使用Keil开 发和调试程序的方法,以帮助开发者开发低功耗蓝牙(Bluetooth Low Energy, BLE)应用。

#### 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
- • 
   文档工程师

#### 版本说明

本手册为第8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"2.5.3 RAM电源管理"、"4.6.4 输出调试Log"章节描述
1.5	2020-05-30	更新"1简介"中的芯片型号、"2.3存储器映射"、"2.6 GR551x SDK目录结
		构"、"4 使用SDK开发调试"等章节描述
1.6	2020.05.20	优化 "2.4.1 SCA" 章节布局图、更新 "2.5.3 RAM电源管理" 章节、更新 "4.6.4 输出调
1.6 20	2020-00-30	试Log"
1.7	2020-08-30	更新"1简介"中的芯片型号
	2020-09-25	更新"2.5.2 Mirror模式的RAM布局"章节Mirror模式的RAM布局图、新增"App Code
1.0		Execution Region"段的起始值设置方法
1.8		更新 "4.5 下载.hex文件到Flash"章节, GR551x_8MB_Flash.FLM文件在SDK包中的存放路
		径、增加下载.hex文件过程中出现的"No Cortex-M SW Device Found"报错说明
1.9	2020-11-25	优化 "2.4.2 NVDS" 章节关于NVDS的描述、更新idx的取值范围为0x0000 ~ 0x3FFF
		增加"4.3.2.1 配置custom_config.h"章节的配置参数,包括: CHIP_TYPE、
		CFG_MAX_LEG_EXT_ADVS、 CFG_MAX_PER_ADVS、 CFG_MAX_SCAN、
		CFG_MAX_PER_ADV_SYNC

## GODIX

版本	日期	修订内容
2.0	2021-01-05	调整"4.6.4 输出调试Log"章节为"4.6.4.1 配置参数"、"4.6.4.2 模块初始
		化"和"4.6.4.3使用方法"三个子章节

前言	I
1 简介	1
1.1 GR551x SDK	1
1.2 低功耗蓝牙协议栈	1
2 GR551x低功耗蓝牙软件平台	4
2.1 硬件架构	4
2.2 软件架构	5
2.3 存储器映射	6
2.4 Flash存储映射	7
2.4.1 SCA	7
2.4.2 NVDS	9
2.5 RAM存储映射	10
2.5.1 XIP模式的RAM布局	
2.5.2 Mirror模式的RAM布局	
2.5.3 RAM电源管理	
2.6 GR551x SDK目录结构	
3 启动流程(Bootloader)	
4 使用SDK开发调试	
4.1 安装Keil	
4.2 安装GR551x SDK	
4.3 创建BLE Application	
4.3.1 准备ble_app_example	
4.3.2 配置工程	24
4.3.2.1 配置custom_config.h	
4.3.2.2 配置存储器布局	27
4.3.2.3 配置After Build	
4.3.3 添加用户代码	29
4.3.3.1 修改主函数	
4.3.3.2 实现BLE业务逻辑	
4.3.3.3 BLE_Stack_IRQ、BLE_SDK_IRQ与Application的调度机制	
4.4 生成.hex文件	33
4.5 下载.hex文件到Flash	
4.6 调试	
4.6.1 配置调试器	
4.6.2 启动调试	
4.6.3 Mirror模式下的调试	40

## G@DiX

4.6.4 输出调试Log	
4.6.4.1 配置参数	
4.6.4.2 模块初始化	
4.6.4.3 使用方法	
4.6.5 使用GRToolbox调试	
5 术语和缩略语	

## GODiX

## 1 简介

GR551x系列芯片是Goodix推出的支持Bluetooth 5.1的单模低功耗蓝牙片上系统(SoC)芯片,可以配置为 广播者(Broadcaster)、观察者(Observer)、外围设备(Peripheral)和中央设备(Central),并支持上述各 种角色的组合应用,可广泛应用于物联网(IoT)和智能穿戴设备领域。

GR551x系列架构以ARM<sup>®</sup> Cortex<sup>®</sup>-M4F CPU为核心,集成了Bluetooth 5.1协议栈、2.4 GHz RF收发器、片上可 编程存储器Flash、RAM以及多种外设。

GR551x系列芯片已推出多款不同封装类型的芯片产品(表 1-1 ),开发者可根据项目需要选择合适的芯片。

GR551x Series	GR5515IGND	GR5515RGBD	GR5515GGBD	GR5513BEND	GR5515I0ND
CPU	Cortex <sup>®</sup> -M4F				
RAM	256 КВ	256 КВ	256 КВ	128 КВ	256 КВ
Flash	1 MB	1 MB	1 MB	512 КВ	N/A
Package	QFN56	BGA68	BGA55	QFN40	QFN56

#### 表 1-1 GR551x系列芯片封装类型

### 1.1 GR551x SDK

GR551x软件开发工具包(Software Development Kit, SDK)为GR551x系列SoC提供全面的软件开发支 持。该工具包中包含了BLE(Bluetooth Low Energy)Stack API、System API、外设驱动程序、hex文件生成和下载 工具、工程示例代码以及相关的用户文档等。

本文档支持的GR551x SDK版本,适用于GR551x系列的所有芯片。

## 1.2 低功耗蓝牙协议栈

低功耗蓝牙(BLE)协议栈的架构如图 1-1所示。

BLE Protocol Stack		
Host		
Generic Attribute Profile (GATT)	Generic Access Profile (GAP)	
Attribute Protocol (ATT)	Security Manager (SM)	
Logical Link Control and Adaption Protocol (L2CAP)		
Host Controller Interface (HCI)		
Controller		
Link Layer (LL)		
Physical Layer (PHY)		

#### 图 1-1 BLE协议栈架构

BLE协议栈由控制器(Controller)、主机控制接口(HCI)和主机(Host)组成。

#### 控制器(Controller)

- 物理层(Physical Layer, PHY)支持1 Mbps和2 Mbps的自适应跳频GFSK(高斯频移键控)射频(RF)操作。
- 链路层(Link Layer, LL),控制设备的射频状态,设备可以处于如下的五种状态,根据应用需求相 互切换: Standby, Advertising, Scanning, Initiating或者Connection。

#### 主机控制接口(HCI)

• 主机控制接口(Host-Controller Interface, HCI)提供了Host与Controller之间的通信。该接口层的实现可以是软件接口,也可以是标准硬件接口,比如UART,Secure Digital(SD)或USB。HCI commands和events通过这个接口层在Host与Controller之间传递。

#### 主机(Host)

- 逻辑链路控制和适配协议(Logical Link Control and Adaption Protocol, L2CAP)为上层提供了多路复用、数据分段与重组服务,并且支持逻辑端对端的数据通信。
- 安全管理层(Security Manager, SM)定义了配对和密钥分发的方法,为上层协议栈和应用程序提供 端到端的安全连接和数据交换的功能。
- 通用访问规范层(Generic Access Profile, GAP)为上层应用和Profiles提供和协议栈通信交互的接口,主要包括广播、扫描、连接发起、服务发现、连接参数更新、安全过程发起和响应的相关功能。
- 属性协议层(Attribute Protocol, ATT)定义了服务端和客户端之间的服务数据交互协议。
- 通用属性规范层(Generic Attribute Profile, GATT)基于ATT协议之上,定义了一系列用于GATT Client和GATT Server之间服务数据交互的通信过程,供上层应用、Profile和Service进行使用。

更多BLE技术及其协议的资料请访问Bluetooth SIG的官方网站www.bluetooth.com获取。

GAP、SM、L2CAP和GATT的规范包含在<u>Bluetooth Core Spec v5.1</u>中。其他BLE应用层Profiles/Services规范可以在<u>GATT Specs</u>页面下载。BLE应用可能会用到的Assigned Numbers,IDs和Codes都列在<u>Assigned Numbers</u>页面。

## 2 GR551x低功耗蓝牙软件平台

GR551x SDK是一套基于GR551x芯片定义的低功耗蓝牙应用开发的软件套件,包括BLE 5.1 API、System API和外设驱动API接口,并提供了丰富的蓝牙和外设应用示例工程和使用说明文档。应用开发者可以基于GR551x SDK的示例工程进行快速产品开发和迭代。

### **2.1** 硬件架构

GR551x的硬件框图如下。本节将简要介绍芯片内各模块,更多的详细资料请参考《GR551x Datasheet》。



图 2-1 GR551x硬件架构

- ARM<sup>®</sup> Cortex<sup>®</sup>-M4F: GR551x SoC芯片的系统核心处理器(CPU)。BLE协议栈和Application代码都运行 在该处理器上。
- RAM: 随机存取存储器,提供程序执行时需要的内存空间。
- ROM: 只读存储器,固化了Bootloader,BLE协议栈的软件部分。
- Security Cores:安全计算引擎单元,主要包括TRNG、AES、SHA和PKC等模块,提供了对加密的用户应用Firmware进行校验的功能。对加密Firmware的校验是通过ROM中的安全启动流程完成的(Bluetooth SPEC中与安全相关的计算单元是包含于Communication Core中的独立模块,与SecurityCore无关)。
- Peripherals: GPIO、DMA、I2C、SPI、UART、PWM、Timer等硬件。
- RF Transceiver: 2.4 GHz射频信号收发器。
- Communication Core: Bluetooth 5.1协议栈控制器的物理层。它也是软件协议栈与2.4 GHz射频硬件之间的接口。
- PMU(Power Management Unit): 电源管理单元,为各系统模块提供电源供应,根据配置参数和当前的运行状态,设定合理的DC/DC、IO-LDO、Dig-LDO、RF Subsystem等模块参数。



• Flash: 封装在芯片内部的Flash存储单元,用于用户代码和数据存储,支持用户代码片上运行模式 (Execute in Place, XIP)。

### 2.2 软件架构

图 2-2展示了GR551x SDK的软件架构。



图 2-2 GR551x软件架构

Bootloader

引导程序,负责初始化芯片的软硬件环境,校验并启动应用程序。

BLE Stack

低功耗蓝牙协议栈实现核心,由控制器(Controller)、主机控制接口(HCI)和主机(Host)协议组成(包括LL,HCI,L2CAP,GAP,SM,GATT),支持Broadcaster、Observer、Peripheral和Central角色。

BLE SDK

软件开发工具包,提供易于使用的SDK BLE API和SDK System API。

- 。 SDK BLE API包括L2CAP、GAP、SM和GATT API。
- <sup>2</sup> SDK System API提供了对非易失性数据存储系统(NVDS)、Firmware升级(Device Firmware Update, DFU)、系统电源管理以及通用系统级访问接口的API定义。
- Application

在SDK包中,提供了丰富的蓝牙及外设示例工程,每个示例工程中都包含编译后的二进制文件,用 户可以下载到芯片中运行和测试。对于大部分的蓝牙应用,SDK包中的安卓APP也提供了对应的功 能,方便用户测试。

## GODIX

#### • Drivers

外设驱动部分的API定义及说明。关于Drivers的详细介绍,请参考<u>《GR551x HAL及LL驱动用户手</u>册》。

### 2.3 存储器映射

图 2-3展示了GR551x系列SoC的存储器映射。



图 2-3 GR551x存储器映射

#### 🛄 说明:

• GR5515系列SoC:

RAM存储: 0x3000\_0000 ~ 0x3003\_FFFF或0x0080\_0000 ~ 0x0083\_FFFF,共256 KB。 Flash存储: 0x0100\_0000 ~ 0x010F\_FFFF或0x0300\_0000 ~ 0x030F\_FFFF,共1 MB。 GR5515I0ND使用外部QSPI Flash。

• GR5513 SoC:

RAM存储: 0x3000\_0000~0x3001\_FFFF或0x0080\_0000~0x0081\_FFFF,共128 KB。

Flash存储: 0x0100\_0000 ~ 0x0107\_FFFF或0x0300\_0000 ~ 0x0307\_FFFF,共512 KB。

## GODiX

## 2.4 Flash存储映射

GR551x封装了一个采用XQSPI总线接口的可擦除外部Flash存储器。该Flash物理上由若干个4 KB大小的Flash Sector组成:逻辑上可根据不同的应用场景,分成不同用途的存储区域。

图 2-4为GR5515典型应用场景的Flash存储布局。



#### 图 2-4 Flash存储布局

- System Configuration Area (SCA):系统配置区,主要用于存储系统启动参数配置信息。
- User App: Application Firmware存储区域。
- Unused Space: 空闲区域。开发者可以自行使用该区域。比如,在DFU升级过程中,用Unused Space临时存储新的Application Firmware。
- Non-volatile Data Storage (NVDS): 非易失性数据存储区域。

#### 🛄 说明:

NVDS缺省占用Flash的最后一个扇区(Sector)。开发者也可以根据产品的Flash使用布局,合理安排NVDS的起始地址以及所占用的扇区数量,具体配置方法参考4.3.2.1 配置custom\_config.h。

注意: NVDS起始地址需要和Flash扇区的起始地址对齐。

### 2.4.1 SCA

System Configuration Area (SCA) 位于Flash的前两个Sector (共8 KB, 0x0100\_0000 ~ 0x0100\_2000)。SCA中存储了系统启动过程使用的标志以及其他系统配置参数。SDK toolchain链接工具会根据用户配置文件*custom\_config.h* (路径: Project\_Folder\Src\config)生成SCA Image,并将其烧写到SCA,参考4.4 生成.hex文件。图 2-5为System Configuration Area布局。

#### 🛄 说明:

Project\_Folder为工程的根目录。

## G@DiX



#### 图 2-5 System Configuration Area布局

- Boot\_Info与Boot\_Info Backup存储了相同的信息,Boot\_Info Backup是Boot\_Info的备份。
  - <sup>°</sup> 在非安全模式下,Bootloader会默认从Boot\_Info中获取启动信息。
  - 在安全模式下,Bootloader会先校验Boot\_Info,如果Boot\_Info校验不过,则会校验Boot\_Info
     Backup,并从Boot\_Info Backup中获取启动信息。
- Boot\_Info(32B)区域中存储了Firmware启动信息,Bootloader会根据启动信息,校验和跳转 到Firmware的入口地址。
  - <sup>°</sup> Boot Config为系统启动配置信息。
  - 。 SPI Access Mode为SPI访问方式配置。为系统固定配置,用户无法修改。
  - <sup>°</sup> Run Addr为Firmware运行地址,与*custom\_config.h*中的APP\_CODE\_RUN\_ADDR对应。
  - <sup>°</sup> Load Addr为Firmware存储地址,与*custom\_config.h*中的APP\_CODE\_LOAD\_ADDR对应。
  - ° CheckSum为Firmware校验和,生成Firmware后,由SDK toolchain自动算出。
  - 。 APP Size为Firmware的Size信息,生成Firmware后,由SDK toolchain自动算出。
- Img\_Info区域存储了最多10个Firmware的信息。当使用GProgrammer下载Firmware或使用DFU升 级Firmware时,Firmware信息会被存储到Img\_Info区域。
  - <sup>°</sup> Comments为Firmware描述信息,支持最多12个字符。生成Firmware后,SDK toolchain使用Firmware文件名作为Comments信息。
  - <sup>°</sup> Boot Info(24B)为Firmware启动信息,与上述Boot\_Info(32B)的低24 Byte相同。
  - <sup>°</sup> Version为Firmware版本信息,与custom\_config.h中VERSION对应。

- <sup>?</sup> Pattern为固定值0x4744。
- DFU Config Info区域存储了ROM中DFU模块的配置信息,开发者可以通过调用相应API来更改该区域的数据,进而配置DFU模块。
  - <sup>°</sup> UART Info为DFU模块的UART串口相关配置,包括状态位、波特率、GPIO配置等。
  - <sup>°</sup> ADV Name Info为DFU模块的广播相关配置,包括状态位、广播名、广播长度。
  - <sup>°</sup> NVDS Init Info为DFU模块的NVDS系统的初始化配置,包括状态位、NVDS区域大小、起始地址。
  - <sup>°</sup> DFU Disable Cmd Info为DFU模块的DFU禁用命令配置,包括状态位和Disable DFU Cmd(2B,设置 格式为Bitmask),可通过设置Disable DFU Cmd值来禁用某些DFU命令。
- Hmac区域存储了Hmac校验值。该区域仅在安全模式下有效。关于安全模式,参考文档<u>《GR551x固</u> <u>件加密及应用介绍》</u>。

#### 2.4.2 NVDS

NVDS是一个轻量级逻辑数据存储系统,它依赖于Flash硬件抽象层(Flash HAL)。其存储于Flash中,掉电时数据不会丢失。NVDS默认使用Flash最后一个扇区(Sector)作为存储区域。

NVDS系统适合存储小块数据,例如应用程序的配置参数、校准数据、状态和用户信息等。BLE协议栈也会使用NVDS存储设备绑定等参数。

NVDS系统具有以下特性:

- 每个存储项(TAG)都有唯一的TAG ID用于标识,用户程序可以根据TAG ID对数据内容进行读取和更改操作,不需要关心存储的物理地址。
- 针对Flash存储介质的特性进行了优化,支持数据校验、Word对齐、碎片整理和擦写平衡。
- 存储区域的大小和起始地址可配置, Flash存储区以Sector为单位,一个Sector的大小为4 KB, NVDS存储区域可配置为若干个Sector. 配置的起始地址须按4 KB对齐。

NVDS提供了以下五个简单的API用于操作Flash中的非易失性数据。

函数原型	描述
uint8_t nvds_init(uint32_t start_addr, uint8_t sectors)	初始化NVDS使用的Flash Sectors。
uint8_t nvds_get(NvdsTag_t tag, uint16_t *p_len, uint8_t *p_buf)	从NVDS中读出tag标识对应的数据。
wint& though hut/NudeTag thag wint16 then construint& the huf)	将数据写入到NVDS并用tag标识。如果这个tag不存
	在,则创建一个新的。
uint8_t nvds_del(NvdsTag_t tag)	删除NVDS中tag标识的数据。
uint16_t nvds_tag_length(NvdsTag_t tag)	获取指定Tag的数据长度。

#### 表 2-1 NVDS API

**NVDS API**的详细说明请参考<u>《GR551x API Reference》</u>,或直接查看NVDS头文件(位于SDK\_Folder \components\sdk\gr55xx\_nvds.h)。

NVDS存储数据的格式如图 2-6所示:



#### 图 2-6 NVDS存储数据格式

数据头(Data Header)格式如表 2-2 所示:

表 2-2 Data Header格式

Byte	Name	Description
0-1	tag	数据的tag标识
2-3	len	数据的长度
4-4	checksum	数据头的校验和
5-5	value_cs	数据的校验和
6-7	reserved	保留字段

#### 🛄 说明:

BLE协议栈也会使用NVDS存储一些参数,因此必须为NVDS分配Flash存储区域。GR551x SDK会缺省 使用Flash Memory最后一个Sector作为NVDS使用的存储区域,可通过更改*custom\_config.h*文件中的 宏NVDS\_START\_ADDR和NVDS\_NUM\_SECTOR来配置NVDS区域的起始地址和占用大小。BLE协议栈和Application共 享相同的NVDS存储区域,但是TAG ID命名空间被划分为不同的类别,开发者只能使用分配给Application的TAG ID命名类别。

- Application必须使用NV\_TAG\_APP(idx)来获取应用程序数据的TAG ID。该TAG ID被用作NVDS API的参数。
- Application不能将idx直接作为NVDS API的参数。idx取值范围是0x0000 ~ 0x3FFF。

在Application第一次运行前,开发者可使用工具GProgrammar将BLE协议栈和Application所使用的TAG初始值写入到NVDS。如果开发者不使用GR551x SDK缺省的NVDS区域而要自行指定NVDS区域起始地址,请确保GProgrammar中的NVDS区域起始地址配置与*custom\_config.h*中定义的NVDS区域起始地址相同。关于*custom\_config.h*中NVDS区域起始地址的配置,参考4.3.2.1 配置custom\_config.h。

### 2.5 RAM存储映射

GR5515的RAM为256 KB,起始地址为0x3000\_0000,由11个内存块(RAM Block)组成(前4个内存块的大小均为8 KB,其余内存块的大小为32 KB)。每个RAM内存块都可由软件独立打开/关闭电源。

#### 🛄 说明:

GR5515为起始地址0x3000\_0000的RAM提供了一个起始地址为0x0080\_0000的Aliasing Memory,见图 2-3。如果代码的运行地址在Aliasing Memory地址范围,可以加快代码在RAM中的运行速度。GR551x SDK缺省使能了该Aliasing Memory。

256 KB RAM存储布局如图 2-7所示:



#### 图 2-7 256 KB RAM存储布局

程序运行模式可配置为Execute in Place(XIP)模式或Mirror模式。详细的配置方法参考4.3.2.1 配置custom\_config.h中的"APP\_CODE\_RUN\_ADDR"。这两种运行模式有不同的RAM布局。

#### 表 2-3 程序运行模式

运行模式	描述
VID档式	片上运行模式,用户应用程序存储在片上Flash空间,程序运行空间和加载空间相
NP侠式	同。系统完成上电配置后,通过Cache Controller直接从Flash空间取指运行。
	镜像运行模式,用户应用程序存储在片上Flash空间,程序的运行空间定义在RAM空
Mirror模式	间。在程序启动阶段,会在校验完成后,将程序从外部Flash空间加载到RAM空
	间,并跳转到RAM中进行运行。

🛄 说明:

由于XIP模式运行时需要持续访问Flash,因此该模式下的运行功耗会略高于Mirror模式。

### 2.5.1 XIP模式的RAM布局

图 2-8为XIP模式的典型RAM布局,开发者可以根据产品需要对其进行修改。



#### 图 2-8 XIP模式的RAM布局

XIP模式布局允许在代码加载处直接执行Application的代码,从而让Application能使用更多的RAM内存。在进行Flash数据更新时,需要关闭XIP模式,导致CPU无法从Flash中取指。因此,在编译链接时必须将Flash驱动代码运行地址定向至RAM空间。在GR551x SDK的示例工程所使用的Scatter文件中,定义了RAM\_CODE区域,用于运行那些运行地址在RAM中的代码。睡眠期间,RAM\_CODE区域占用的RAM Block应处于Retention状态。

🛄 说明:

开发者不能从Scatter文件中删除"RAM\_CODE"段。关于Scatter文件描述,请参考4.3.2.2 配置存储器布局。

### 2.5.2 Mirror模式的RAM布局

图 2-9为Mirror模式的典型RAM布局,开发者可以根据产品需求对其进行修改。



#### 图 2-9 Mirror模式的RAM布局

Mirror模式布局允许在RAM中执行Application的代码。芯片上电之后,会进入冷启动流程。Bootloader会将Application的代码从Flash中复制到名为 "App Code Execution Region"的RAM段。睡眠模式的芯片被唤醒后进入热启动流程。为减少热启动时间,Bootloader不会重新复制Application的代码到名为 "App Code Execution Region"的RAM段中。

"App Code Execution Region"段的起始位置由*custom\_config.h*中的宏APP\_CODE\_RUN\_ADDR决定。开发者 需要根据Application的.*data*和.*bss*实际使用情况,来确定代码运行地址APP\_CODE\_RUN\_ADDR的值,避免与低地 址处的.bss段或高地址处的Call Stack段地址重叠。开发者可根据.*map*文件来获得RAM各段的分布情况。

建议开发者使用RAM Aliasing Memory地址(0x0080\_0000 ~ 0x0083\_FFFF)来设置APP\_CODE\_RUN\_ADDR。若出现RAM段重叠,在工程构建时会出现error并提示重叠位置,帮助开发者确认并快速定位RAM段重叠情况。

#### 2.5.3 RAM电源管理

每一个RAM Block可以处于三种不同的电源状态: POWER OFF、RETENTION或FULL。

- FULL对应于系统Active状态,MCU可以进行RAM Block读写。
- RETENTION主要用于系统Sleep状态,处于该电源状态的RAM Block中的数据将不会丢失,供系统 从Sleep状态恢复到Active态使用。
- 处于POWER OFF状态的RAM Block会掉电,存储于其中的数据会丢失,需用户提前进行保存。

在GR551x中,电源管理单元(PMU)在系统启动时默认开启了全部RAM电源。GR551x SDK中也提供了完备的RAM电源管理API,开发者可以根据应用需要,合理配置RAM Block的电源。

系统启动时,默认启用了自动RAM功耗管理模式,系统会根据Application的RAM使用情况,自动进行 各RAM Block的电源控制。配置规则如下:



- 在系统Active状态,不使用的RAM Block会设置为POWER OFF状态,需要使用的RAM Block则设置为FULL状态。
- 当系统进入Sleep状态时,会将不使用的RAM Block保持为POWER OFF状态,而需要使用的RAM Block设置为RETENTION状态。

在实际应用中,应配置如下:

- 在BLE应用中,RAM\_8K\_0和RAM\_8K\_1是保留给Bootloader和BLE协议栈使用,Application不可 使用。在系统Active时,它们应处于FULL状态,在系统Sleep期间,它们应处于RETENTION状态。非BLE类MCU应用可以使用这两个RAM Block。
- RAM\_8K\_2及以后的RAM Block的用途可由Application进行规划定义。通常,将用户数据和需要 在RAM中执行的代码段定义在从RAM\_8K\_2开始的连续区间,将函数调用栈(Call Stack)的栈顶定义 在RAM的高端地址。这些RAM Block的电源状态可以全部开启,也可以由Application自行控制。

#### 🛄 说明:

- 1. 只有当RAM Block处于FULL状态时,MCU才能对其进行访问。
- 开发者如果要自己管理RAM电源,使用Application的Memory Layout信息中没有包含的RAM Block,则 需要在Application初始化阶段调用mem\_pwr\_mgmt\_mode\_set\_from(uint32\_t start\_addr, uint32\_t size)函数给RAM Block上电。
- 更多RAM电源管理API的详细说明在SDK\_Folder\components\sdk\platform\_sdk.h中。
   SDK\_Folder为GR551x SDK的根目录。

### 2.6 GR551x SDK目录结构

图 2-10为GR551x SDK的文件夹目录结构。



#### 图 2-10 GR551x SDK目录

#### 表 2-4 为GR551x SDK中各文件夹的详细描述。

#### 表 2-4 GR551x SDK文件夹

文件夹	描述
build\binaries	构建过程所需要使用的工具。
huild\config	工程配置目录,用于存放custom_config.h模板文件。该文件的内容用于配置工程和
build (coning	提供SDK toolchain所需的相关输入参数。
build\scripts	批处理文件和脚本文件,用于构建、下载、调试。
componets\app_drivers	易于Application开发者使用的驱动API源代码。
components\boards	开发板对应的头文件。
components\drivers_ext	开发板上第三方元器件的驱动。

#### GR551x低功耗蓝牙软件平台



文件夹	描述
components\libraries	GR551x SDK提供的libraries。
components\patch	ROM Patch相关的文件。为ROM中的BLE Stack提供增量更新。
components\profiles	GR551x SDK提供的GATT Services/Service Clients实现示例的源文件。
components\sdk	GR551x SDK提供的API头文件。
components\sdk\linker	GR551x SDK提供给链接器使用的符号表文件和库文件。
documentation	GR551x API Reference。
drivers\inc	GR551x外设驱动的HAL和LL的头文件。
drivers\src	GR551x外设驱动的HAL的源代码。
external\fat_fs	第三方程序,FatFS源代码。
external\freertos	第三方程序,FreeRTOS源代码。
external\segger_rtt	第三方程序,SEGGER RTT源代码。
external\wechat	第三方程序,Wechat源代码。
projects\ble	BLE Application工程示例,比如Heart Rate Sensor,Proximity Reporter。
projects\peripheral	芯片外设工程示例。
projects\peripheral_app	芯片外设APP工程示例。
toolchain\cmsis	CMSIS的Toolchain文件。
toolchain\gr551x	编译平台的工具链文件。
tools	GR551x的开发与调试工具软件。

## 3 启动流程(Bootloader)

GR551x支持两种代码运行模式: XIP模式和Mirror模式。系统上电后,启动引导代码从系统配置区(SCA) 读取系统启动配置信息,并据此进行应用固件完整性校验、系统初始化配置后,跳转到代码运行空间执行代码。不同的运行模式,启动流程略有差异。

- XIP模式下,启动代码在完成应用固件校验后,初始化Cache和XIP控制器,然后跳转到Flash空间的代码运行地址进行代码执行。
- Mirror模式下,启动代码在完成应用固件校验后,根据系统配置信息,将Flash空间的代码加载到对 应的RAM运行空间后,复位Flash等接口配置,跳转到RAM空间进行代码执行。

图 3-1为GR551x SDK上Application的启动流程。



图 3-1 GR551x SDK上的Application启动流程

- 1. 设备上电后,CPU将跳转到0x0000\_0000处,执行ROM中的reset\_handler,进入Bootloader。
- 2. Bootloader执行Flash初始化。
- 3. Bootloader从Flash中的SCA读取启动信息,并执行Application Firmware的完整性检查。

## GODIX

🛄 说明:

GR551x提供了安全功能对Application Firmware进行加密和签名。

- 安全模式:若使能安全模式,启动程序从SCA读取启动信息并进行HMAC校验;校验成功后,会解密SCA启动信息,后续执行安全启动流程的验证签名流程,用于保证Firmware完整性和防止篡改,防止伪装等;若验证签名成功,则会使能自动解密功能。详情请参考文档《GR551x固件加密及应用介绍》。
- 非安全模式:若未使能安全模式,启动程序使用SCA启动信息对Application Firmware进行CRC完整性校验。
  - 4. 如果完整性检查失败,则会启动BLE DFU Service。开发者可以通过该Service配合手机APP更新Flash中的Application Firmware。
  - 5. 如果通过了完整性检查, Bootloader判断运行模式。
    - 对于XIP模式,Bootloader会在完成XIP配置后跳转到Flash中的Application Firmware开始执行。
    - 对于Mirror模式,Bootloader会把Application Firmware从Flash中拷贝到RAM中的指定区段,然后执行RAM中的Application Firmware。

## 4 使用SDK开发调试

图 4-1为Keil启动后的主界面。

本章将以Keil为例,介绍如何使用SDK来完成BLE Application的创建、编译、下载和调试。

### 4.1 安装Keil

Keil MDK-ARM IDE(Keil)是ARM<sup>®</sup>公司提供的用于Cortex<sup>®</sup>和ARM设备的集成开发环境(IDE)。开发者可 从官方网站<u>https://www.keil.com/demo/eval/arm.htm</u>下载Keil安装包并进行安装。GR551x SDK必须为Keil V5.20及 以上的版本。

🛄 说明:

关于Keil MDK-ARM IDE的使用,可查看ARM提供的在线用户手册: http://www.keil.com/support/man\_arm.htm。

	- ø ×
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help	
1   2   2   2   2   2   2   2   2   2	
Build Output	a E
	, v
	N

图 4-1 Keil软件界面

表 4-1 为Keil的常用功能按钮。

表 4-1 Keil常用功能按钮

Keil Icon	Description
хх.	Options for Target
Q	Start/Stop debug session
	Download
<b>**</b>	Build

## G@DiX

## 4.2 安装GR551x SDK

开发者解压GR551x SDK zip软件包后即可使用,无需手动安装。

🛄 说明:

- SDK\_Folder为GR551x SDK的根目录。
- Keil\_Folder为Keil的根目录。

### 4.3 创建BLE Application

本节介绍如何创建一个BLE Application。

### 4.3.1 准备ble\_app\_example

进入SDK\_Folder\projects\ble\ble\_peripheral\,拷贝ble\_app\_template到当前目录,并 重命名为ble\_app\_example。将ble\_app\_example\Keil\_5中的.uvoptx和.uvprojx的主文件名修改 为ble\_app\_example。

<ul> <li>ble_app_example</li> <li>Keil_5</li> </ul>
名称
ble_app_example.uvoptx ble_app_example.uvprojx

#### 图 4-2 ble\_app\_example文件夹

双击*ble\_app\_example.uvprojx*, Keil打开该工程示例。点击 ≤, 在 "Options for Target 'GR551x\_SK'"中选择 "Output", 在 "Name of Executable" 输入 "ble\_app\_example"。



I Options for Target 'GR551x_SK'	×
Device   Target Output   Listing   User   C/C++   Asm   Linker   Debug   Utilities	
Select Folder for Objects Name of Executable: ble_app_example	
Image: Create Executable: .\Objects\ble_app_example         Image: Debug Information         Image: Create HEX File	
I     Browse Information       ○     Create Library: .\Objects\ble_app_example.lib	
OK Cancel Defaults Help	

图 4-3 修改Name of Executable

在Keil Project Explore中,可查看到ble\_app\_example工程下的所有groups。



图 4-4 ble\_app\_example项目

ble\_app\_example工程下的groups主要分为两类: SDK groups和User groups。

SDK groups

SDK groups包

括gr\_startup、gr\_arch、gr\_stack\_lib、gr\_drivers、gr\_app\_drivers、gr\_libraries、gr\_profiles和external。



#### 图 4-5 SDK groups

SDK groups下的源文件无需修改,各group的具体描述如下:

#### 表 4-2 SDK groups

SDK group名称	描述
gr_startup	系统启动文件。
gr_arch	System Core、PMU的初始化配置文件和系统中断的接口实现
gr_stack_lib	GR551x SDK lib文件。
gr_drivers	芯片外设的驱动源文件。开发者可自行添加项目所需的相关硬件驱动。
gr_app_drivers	易于Application开发者使用的驱动API源文件。开发者可自行添加项目所需要的相关app drivers。
gr_libraries	SDK提供的常用辅助软件模块、外设驱动的开源文件。
gr_profiles	GATT Services/Service Clients源文件。开发者可自行添加项目所需的相关GATT源文件。
external	第三方程序的源文件,例如freertos, segger rtt。开发者可自行添加项目所需第三方程序。

#### User groups

User groups包括user\_callback、user\_platform和user\_app。



#### 图 4-6 user\_groups

User groups下的源文件需要开发者来实现,各group的具体描述如下:

#### 表 4-3 User groups

User group名称	描述
user_callback	应用程序实现的BLE Stack Callback函数。Callbacks的更多介绍,请参考表 4-5。
user_platform	软硬件资源的设置和应用程序的初始化,开发者需要根据自己项目需要实现相应的接口。
user_app	主函数入口以及开发者创建的其他源文件,配置BLE协议栈运行时参数和实现GATT Service/
	Service Client的事件处理函数。

### 4.3.2 配置工程

开发者需要根据自己产品的特性配置相应的工程选项,包括NVDS,代码运行模式,存储器布局,After Build以及其他配置项。

### 4.3.2.1 配置custom\_config.h

*custom\_config.h*用于配置Application工程的参数。SDK\_Folder\build\config\提供了一个*custom\_config.h*的模板。各Application示例工程的*custom\_config.h*位于其工程目录下的Src\config.

宏	描述
CHIP_TYPE	对芯片类型进行选择。
	0: GR5515
	1: GR5513
	打印Callstack Trace Info的开关。打开该开关后,发生HardFault时,会将Callstack Trace Info通
SVS FALLET TRACE ENABLE	过串口打印出来。
	0: 关闭Trace打印
	1: 打开Trace打印
APP_DRIVER_USE_ENABLE	App Drivers模块的开关。
	0:不使用App Drivers模块
	1: 使用App Drivers模块
	APP LOG模块的开关。
APP_LOG_ENABLE	0: 关闭Application中的Logs
	1: 打开Application中的Logs
SK_GUI_ENABLE	Gr5515 Start Kit Board的GUI模块的开关。
	0: 不使用GUI模块
	1: 使用GUI模块
DTM_TEST_ENABLE	DTM Test功能的开关。
	0:关闭DTM Test功能
	1: 打开DTM Test功能

#### 表 4-4 custom\_config.h中的参数

## G@DiX

宏	描述	
	DFU功能的开关。	
DFU_ENABLE	0: 关闭DFU功能	
	1: 打开DFU功能	
	PMU CALIBRATION功能开关,打开后,系统会自动监控温度和电压并做自适应调整,高低温	
	场景必须开启,建议默认开启。	
	0: 关闭PMU CALIBRATION功能	
	1: 打开PMU CALIBRATION功能	
	Application在Flash擦写过程中,可阻止异常事件响应的优先级等级。	
FLASH PROTECT PRIORITY	当FLASH_PROTECT_PRIORITY配置为N时,中断优先级小于或等于N的中断请求会被挂	
	起,待Flash擦写完毕后,可响应挂起的中断请求。默认情况下,Flash擦写过程中不响应任	
	何中断请求,开发者可以根据实际需要设置其值。	
	NVDS占用区域的起始地址,默认状态下该宏在cutom_config.h中已被注释,开发者若需要更	
NVDS_START_ADDR	改NVDS占用区域的配置,可启用该宏并更改为所需的值(必须按4 KB对齐)。	
NVDS_NUM_SECTOR	NVDS使用的Flash Sector的数量。	
	Appplication所需要的Call Stack和Heap的大小,开发者可以根据Application的实际使用情	
	况,调整其值,但不能少于6 KB。当前默认大小为32 KB。	
CSTACK_HEAP_SIZE	ble_app_example编译后其目录下的Keil_5\Objects\ble_app_example.htm提供了可	
	供参考的Maximum Stack Usage。	
	Application中所有全局变量的占用的RAM内存大小。为支持在Sleep Mode下关闭/打开Non-	
	retention内存块,其值需保证该RAM段的边界对齐到RAM内存块的边界。	
	Application在Flash中占用的代码段的大小。开发者根据实际的Application的code size来设置该	
AFF_WAA_CODE_SIZE	值。	
APP_CODE_LOAD_ADDR *	程序存储空间的起始地址。该地址应在Flash地址范围内。	
	程序运行空间的起始地址。	
APP_CODE_RUN_ADDR *	如果该地址的值与APP_CODE_LOAD_ADDR相等,则Application会采用XIP模式运行。	
	如果该地址的值在RAM地址范围内,则Application会采用Mirror模式运行。	
	系统时钟频率。其取值如下:	
SYSTEM_CLOCK *	0: 64 MHz	
	1: 48 MHz	
	2: 16 MHz (XO)	
	3: 24 MHz	
	4: 16 MHz	
	5: 32 MHz (PLL)	
CFG_LF_ACCURARY_PPM	BLE低频睡眠时钟精度,其取值范围1~500,单位PPM。	

## GODIX

宏	描述		
	是否使用芯片内部的OSC时钟作为BLE低频睡眠时钟。若使用,则CFG_LF_ACCURARY_PPM强		
CFG_LPCLK_INTERNAL_EN	制设置为500。		
	0: 不使用		
	<b>1</b> : 使用		
	设置芯片启动时是否需要延迟1s再执行后半段启动代码。		
BOOT_LONG_TIME *	0: 不延迟		
	1: 延迟1秒		
	在XIP模式中,冷启动时是否要对image进行校验。		
BOOT_CHECK_IMAGE	<b>0</b> :不进行校验		
	1: 进行校验		
VERSION *	Application Firmware版本号,长度2 Bytes,采用十六进制存储。		
	DAP Boot模式的开关。		
DAP_BOOT_ENABLE	0: 关闭DAP Boot模式		
	1: 打开DAP Boot模式		
	Application支持的最大可绑定设备数量。开发者根据实际需求设置该值,其值越大占用		
CFG_MAX_BOND_DEV_NUM	的RAM空间就越大。		
	Application所能够包含的GATT Profile/Service的最大数量。开发者根据实际需求设置该值,其		
CFM_MAX_PRF_NB	值越大占用的RAM空间就越大。		
	Application支持的最大连接数量,最大值为10。开发者可以根据实际需要设置其值。		
	该值越大,BLE Stack Heaps需要占用的RAM空间就越大。BLE Stack Heaps具体大小		
	由flash_scatter_config.h中的以下四个宏来定义,开发者不可修改这四个宏:		
CFG_MAX_CONNECTIONS	ENV_HEAP_SIZE,		
	ATT_DB_HEAP_SIZE,		
	KE_MSG_HEAP_SIZE,		
	NON_RET_HEAP_SIZE。		
CFG_MAX_LEG_EXT_ADVS	Application支持的BLE传统广播和扩展广播总量的最大值。		
	Application支持的最大BLE周期性广播数量。		
	说明:		
Crg_IVIAX_PER_ADV3	配置的传统广播和扩展广播总量值(CFG_MAX_LEG_EXT_ADVS)与周期性广播数量值		
	(CFG_MAX_PER_ADVS)的总和不能超过5。		
CFG_MAX_SCAN	Application支持的最大BLE扫描数量,最大值为1。		
	周期性广播同步数量,用于预留协议栈所需的RAM资源。开发者可根据实际使用的周期性广		
CFG_IVIAX_PER_ADV_SYNC	播同步数量设置该值,最大值为5。		
	是否支持MESH功能。		
CFG_MESH_SUPPORT	0: 不支持		

## GODIX

宏	描述
	1: 支持

\*: SDK toolchain中的*ble\_tools.exe*会读取该宏的值,用于生成系统配置区(SCA)镜像文件。Bootloader程序会从系统配置区读取该值,作为启动参数。

*custom\_config.h*的注释符合Keil的Configuration Wizard Annotations规范,因此开发者可以使用图形化的Keil Configuration Wizard来配置Application工程参数。强烈推荐开发者使用Wizard以避免出现非法参数值。

	<b>▼</b> ×
Expand All Collapse All Help Show Grid	
Option	Value
-Basic configuration	
Enable system fault trace module	ENABLE
Enable APP driver module	ENABLE
Eanble APP log module	ENABLE
Enable SK GUI module	ENABLE
Enable DTM test support	DISABLE
Enable BLE DFU support	ENABLE
Enable PMU calibration	ENABLE
Protection priority level	0
The Number of sectors for NVDS	1
Call stack size	0x0000 8000
RAM size of application	0x0003 0000
Code size of application	0x0080 0000
Boot info configuration	
Code load address	0x0100 2000
Code run address	0x0100 2000
System clock	64MHZ
Low power clock accuracy	500
Enable internal osc as low power clock	Default: Disable internal osc as low power clock
Delay time for boot startup	Delay 1s
Enable image check during cold boot startup in xip mode	ENABLE
Code version.16bits	1
Enable DAP boot mode	ENABLE
-BLE resource configuration	
Maximum number of BLE profiles	10
Maximum number of bonded devices	4
Maximum number of BLE links	10
-Mesh support configuration	
Mesh support	NOT SUPPORT
Text Editor Configuration Wizard	

图 4-7 Configuration Wizard for custom\_config.h

#### 4.3.2.2 配置存储器布局

Keil用.sct文件定义链接器用到的存储段。GR551x SDK为Application开发者提供了一个示例*flash\_scatter\_common.sct*。该.sct文件使用的宏被定义在*flash\_scatter\_config.h*中。

#### 🛄 说明:

在Keil中, \_\_attribute\_\_((section("name")))可用来将一个函数或变量放在单独的内存段中, 其中 "name" 取决于开发者的选择。Scatter(*.sct*)文件可用来将被命名的段放在特定的位置。例如,将应用程序的ZI(零初始化)数据放在名为\_\_attribute\_\_((section(".bss.app")))段中。

开发者可按照以下步骤配置存储器布局:

- 点击Keil Toolbar的"Options for Target"按钮▲,打开"Options for Target 'GR551x\_SK'"对话框。选中"Linker"标签页。
- 在 "Scatter File" 栏,点击按钮 "…"浏览选择SDK\_Folder\toolchain\gr551x\source \arm下的*flash\_scatter\_common.sct*文件,或者将scatter文件(*.sct*)及其包含的*.h*文件复制 到ble\_app\_example工程目录再选择。

#### 🛄 说明:

*flash\_scatter\_common.sct*中的#! armcc -E -I ..\Src\config\ --cpu Cortex-M4指定了一 个Include路径,其指向Application工程的*custom\_config.h*所在的目录。如果该路径错误,则会产生Linker Error。

3. 点击"Edit..."按钮,打开.sct文件,然后根据自己产品的存储器布局修改相应的代码。

Options for Target 'GR551x SK'	×
Device Target [Autout] Listing Meer [C/C++ ] Asm Linker Debug   Milities]	
Use Memory Layout from Target Dialog       X/O Base:         Make RW Sections Position Independent       R/O Base:         Make RO Sections Position Independent       R/O Base:         Dont Search Standard Libraries       0x00000000         Report 'might fail' Conditions as Errors       disable Warnings:	
Scatter <u>Edit.</u>	
Misc	~
Linker control stringstrictscatter "\\\.toolchain\gr551x\source\am\flash_scatter_common.sct"	< >
OK Cancel Defaults	Help

#### 图 4-8 配置Scatter File

4. 点击"OK"按钮,保存设置。

#### 4.3.2.3 配置After Build

Keil的"After Build"可以指定一个可执行程序或批处理文件在成功build之后运

行。ble\_app\_template工程默认已添加after\_build.bat文件在"After Build"中,开发者无需再给基

于ble\_app\_template的ble\_app\_example工程手动配置"After Build"。SDK提供的*after\_build.bat*的功能参考4.4 生成.hex文件。

如果开发者新建一个工程,则需要按照以下步骤配置After Build:

点击Keil Toolbar的"Options for Target"按钮▲,打开"Options for Target 'GR551x\_SK'"对话框。选中"User"标签页。

## G@DiX

 在 "After Build/Rebuild"展开的选项中勾选 "Run #1",然后点击 ☑ 浏览选择SDK\_Folder\build \scripts下的after\_build.bat文件,或者先将SDK\_Folder\build\scripts下的after\_build.bat文件复制到ble\_app\_example工程目录再选择它。

#### 🛄 说明:

*after\_build.bat*中含有一些相对路径,比如SDK tools目录的路径。如果该批处理文件不再位于SDK\_Folder \build\scripts中了,开发者应该修改这些相对路径,以避免After Build Error。

3. 点击"OK",保存设置。

🛚 Options for Target 'GR551x_	<u>S</u> K'		×	
Device   Target   Output   Listing User C/C++   Asm   Linker   Debug   Vtilities				
0				
Command Items	User Command	Stop on Exi S		
Before Compile C/C++ File				
		对 Not Specified		
Run #2		对 Not Specified		
Before Build/Rebuild				
Run #1		对 Not Specified		
Run #2		对 Not Specified		
After Build/Rebuild				
	\\\\build\scripts\after_build.bat @L @K	对 Not Specified		
2 Run #2		对 Not Specified		
Run 'After-Build' Conditionally				
✓ Beep When Complete	☐ Start Debugging			
3	OK Cancel Defaults	Help		

图 4-9 配置After Build

### 4.3.3 添加用户代码

开发者需要根据自己的需求,修改ble\_app\_example中相应的代码。

#### 4.3.3.1 修改主函数

以下为典型的main.c文件的内容。

```
/**@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) 定义了四个全局数组,供BLE协议栈作为Heap使用。开 发者不可修改此定义,否则BLE协议栈无法正常运行。关于Heap的大小,参考4.3.2.1 配 置custom\_config.h的CFG\_MAX\_CONNECTIONS。
- 开发者可以在app\_periph\_init()中初始化外设。在开发调试阶段,该函数中的SYS\_SET\_BD\_ADDR可用于设置临时的Public Address。该函数所在的*user\_periph\_setup.c*包含以下主要代码:

```
/**@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_uart_init();
    app_log_assert_init();
    pwr_mgmt_mode_set(PMR_MGMT_SLEEP_MODE);
}
```

- 开发者应该在while(1) { }中添加Application的Main Loop代码,比如处理外部输入、更新GUI。
- 如果开发者使用了App Log模块,则需要在Main Loop中调用app\_log\_flush()。这可以保证在系统 进入Sleep状态之前,Logs被完整输出。关于App Log模块使用,参考4.6.4 输出调试Log。
- 需调用pwr mgmt shcedule()以实现自动功耗管理,降低系统功耗。

#### 4.3.3.2 实现BLE业务逻辑

Application的BLE相关业务逻辑是通过GR551x SDK中定义的若干BLE SDK Callback来驱动的。Application需要 注册这些Callback到SDK中,以获得BLE Stack的运行结果或者状态改变通知。由于BLE SDK Callback是在BLE SDK IRQ的中断上下文(Interrupt Context)中被调用的,因此开发者不能在Callback中执行比较耗时的操作,比如阻 塞式函数调用、无限循环等;否则,将阻塞整个系统运行,导致BLE Stack和SDK BLE模块无法按照正常的时序 运行。

BLE SDK Callback按照GAP、Security Manager、L2CAP、GATT Common和GATT Client来分类。表 4-5 中罗列 了GR551x SDK支持的所有Callback Functions。



#### 表 4-5 BLE SDK Callback Function

File Name	Callbacks Struct	Callback Function
user_app.c	app_callback_t	app_ble_init_cmp_callback
		app_gap_param_set_cb
		app_gap_psm_manager_cb
		app_gap_phy_update_cb
		app_gap_dev_info_get_cb
		app_gap_adv_start_cb
		app_gap_adv_stop_cb
		app_gap_scan_req_ind_cb
		app_gap_adv_data_update_cb
		app_gap_scan_start_cb
		app_gap_scan_stop_cb
		app_gap_adv_report_ind_cb
		app_gap_sync_establish_cb
user_gap_callback.c	gap_cb_fun_t	app_gap_stop_sync_cb
		app_gap_sync_lost_cb
		app_gap_connect_cb
		app_gap_disconnect_cb
		app_gap_connect_cancel_cb
		app_gap_auto_connection_timeout_cb
		app_gap_peer_name_ind_cb
		app_gap_connection_update_cb
		app_gap_connection_update_req_cb
		app_gap_connection_info_get_cb
		app_gap_peer_info_get_cb
		app_gap_le_pkt_size_info_cb
		app_rslv_addr_read_cb
		app_l2cap_lecb_conn_req_cb
		app_l2cap_lecb_conn_cb
		app_l2cap_lecb_add_credits_ind_cb
user_l2cap_callback.c	l2cap_lecb_cb_fun_t	app_l2cap_lecb_disconn_cb
		app_l2cap_lecb_sdu_recv_cb
		app_l2cap_lecb_sdu_send_cb
		app_l2cap_lecb_credit_add_cmp_cb
user_sm_callback.c	sec_cb_fun_t	app_sec_enc_req_cb



File Name	Callbacks Struct	Callback Function
		app_sec_enc_ind_cb
		app_sec_keypress_notify_cb
user gett common cellback c	Ilback c gatt common ch func t	app_gatt_mtu_exchange_cb
	gatt_common_cb_rune_t	app_gatt_prf_register_cb
		app_gattc_srvc_disc_cb
		app_gattc_inc_srvc_disc_cb
		app_gattc_char_disc_cb
		app_gattc_char_desc_disc_cb
user_gattc_callback.c	gattc_cb_fun_t	app_gattc_read_cb
		app_gattc_write_cb
		app_gattc_ntf_ind_cb
		app_gattc_srvc_browse_cb
		app_gattc_cache_update_cb

开发者需要根据产品的功能需求实现所需的BLE SDK Callback。例如,如果产品不支持Security Manager,则可以不实现对应的Callback,如果产品只支持GATT Server而不支持GATT Client,则可以不实现GATT Client对应的Callback。每一类Callback里的Callback Function也不一定需要全部实现,开发者只需要实现产品所 必须的那些Callback Function。

关于BLE API和Callback API的用法,请参考<u>《GR551x BLE Stack用户指南》</u>、SDK\_Folder \documentation\GR551x\_API\_Reference以及SDK\_Folder\projects\ble中的BLE示例源代码。

#### 4.3.3.3 BLE\_Stack\_IRQ、BLE\_SDK\_IRQ与Application的调度机制

BLE Stack是低功耗蓝牙协议栈实现核心,它直接操作Bluetooth 5.1 Core硬件(参考2.2 软件架构)。因此,BLE\_Stack\_IRQ具有整个系统中次高的优先级(SVCall IRQ具有最高优先级),以保证BLE Stack严格按照Bluetooth Core Spec规定的时序运行。

#### 🛄 说明:

SDK\_Folder\toolchain\gr551x\source\system\_gr55xx.c中的system\_priority\_init()设置了系统各模 块默认的中断优先级。

BLE Stack的状态改变会触发优先级较低的BLE\_SDK\_IRQ中断。在该中断处理函数中会调用Application实现的BLE SDK Callback,将BLE Stack的状态改变通知以及相关的业务数据发送到Application。开发者在这些Callback中应避免操作耗时的业务,应当将耗时业务转移到Main Loop或者用户级线程中处理。开发者可使用SDK\_Folder\components\libraries\app\_queue模块从BLE SDK Callback向Main Loop传递事件,或者使用自己的Application Framework。关于在用户线程中处理的方法,参考《GR551x FreeRTOS示例手册》。





图 4-10 non-OS system schedule

### 4.4 生成.hex文件

*after\_build.bat*在Keil完成building后会被Keil自动执行。它会调用*ble\_tools.exe*读取用户配置文件*custom\_config.h*中的配置参数,生成SCA Image,然后与Keil输出的Application Firmware合并成一个.*hex*文件。

BLE Application创建完成以后,开发者可直接点击Keil Toolbar中的"Build"按钮构建工程。工程构建完成之后,在工程目录下的Keil\_5\build文件夹中就会有*ble\_app\_example\_fw.bin,ble\_app\_example\_fw.hex和load\_app.hex*。

#### 🛄 说明:

- *load\_app.hex与ble\_app\_example\_fw.hex*文件内容相同,只是文件名不同。
- *load\_app.hex*是为了方便使用同一个Download Script下载Firmware,参考4.5下载.hex文件到Flash中提到的*download.ini*文件。

## 4.5 下载.hex文件到Flash

构建成功以后,开发者需将构建生成的.hex文件下载到Flash中,具体操作步骤如下:

**1**. 配置Keil flash编程算法。

 复制SDK\_Folder\build\binaries\xflash\_flm\_tools\Keil \GR551x\_8MB\_Flash.FLM文件到Keil\_Folder\ARM\Flash目录。

GODIX

(2) 点击Keil Toolbar中的"Options for Target"按钮 ▲,打开"Options for Target'GR551x\_SK'"对 话框,选择"Debug"标签页。点击"Use: J-LINK/J-TRACE Cortex"右侧的"Settings"按钮。

evice   larget	Output   Listing   Vser   C/C++	Asm Linker Debug Vtilities	
Use Simulator Limit Speed to	with restrictions Settings Real-Time	Use: J-LINK / J-TRACE Cortex     Settings	
Load Application	ion at Startup 🔽 Run to main()	Load Application at Startup II Run to main()	
Initialization File:		Initialization File:	
	Edit	\\\\build\scripts\sram.ini Edit	
-Restore Debug	Session Settings	Restore Debug Session Settings	
IV preakpoint		Vatch Windows	
Watch Wi	ndows & Performance Analyzer	Watch Windows	
Watch Wi Memory Di	ndows & Performance Analyzer isplay	Image: Watch Windows	
Watch Wi Memory Di CPU DLL:	ndows & Performance Analyzer isplay IV System Viewer Parameter:	Watch Windows     Memory Display     Very System Viewer	
Watch Wi     Memory Di     CPU DLL:     SARMCM3.DLL	ndows & Performance Analyzer isplay V System Viewer Parameter:	Image: Second points     Image: Second points       Image: Second points     Ima	
✓ Breakpoint ✓ Watch Wi ✓ Memory Di CPU DLL: SARMCM3.DLL Dialog DLL:	Arameter:	Image: Disearchowing     Image: Provide Action of the second	
V Breakpoint Watch Wi Memory Di CPU DLL: SARMCM3.DLL Dialog DLL: DCM.DLL	Is point to the point of the p	Image: Second points     Image: Product All and a second points       Image: Second points     Image: Product All and	

#### 图 4-11 Debug标签页

(3) 在打开的"Cortex JLink/JTrace Target Driver Setup"窗口中,选中"Flash
 Download"项。在"Download Function"区域,开发者可以设置Erase方式、选择是
 否"Program"、"Verify"、"Reset and Run"。Keil默认配置如下:

Cortex JLink/JTrace Target Driver Setup	×
Debug Trace Flash Download	
C Download Function C Erase Full Chip I⊄ Program C Erase Sectors I⊄ Verify C Do not Erase I⊄ Reset and Run	gorithm 30000000 Size: 0xF000
Programming Algorithm	
Description Device Size Device Type	Address Range
Start:	Size:
Add	]
	<b>确定 取消</b> 应用(A)

图 4-12 选择 "Download Function"

(4) 点击"Add"按钮,在"Programming Algorithm"中添加GR551x 8MB Flash.FLM。

ebug Trace Flash Download Download Function LOAD Crase Full Chip © Erase Sectors © Do not Erase	Image: Program     RAM for Algorithm       Image: Program     Start: 0x30000000       Image: Program     Start: 0x30000000       Image: Program     Start: 0x30000000	200
Description GR551x 8MB Flash	Device Size Device Type Address Range 8M Ext. Flash SPI 01000000H - 017FFFFFH	
	Start: Size: Size: Add Remove	

#### 图 4-13 添加GR551x 8MB Flash编程算法



(5) 配置 "RAM for Algorithm",它定义了加载和执行编程算法的地址空间。"Start"输入框中的 值必须为GR551x中RAM的起始地址"0x30000000","Size"输入框中的值为"0xF000"。

RAM for Algorithm	
Start: 0x30000000	Size: OxF000

图 4-14 "RAM for Algorithm" 设置

- (6) 点击"确定",保存设置。
- 2. 配置"Configure Flash Menu Command"。
  - (1) 点击Keil Toolbar中的"Options for Target"按钮 ▲,打开"Options for Target 'GR551x\_SK'"对 话框,选择"Utilities"标签页。
  - (2) 在 "Configure Flash Menu Command"区域的 "Init File" 栏,点击 "…" 浏览选择SDK\_Folder \build\scripts下的download.ini文件,或者先将download.ini文件复制到自己的工程目录然 后再选择。

	Menu Command		
<ul> <li>Use Targe</li> </ul>	t Driver for Flash Programming	Vse Debu	g Driver
	Use Debug Driver	Settings 🗌 Update Ta	arget before Debugging
Init File:	\\\\build\scripts\download.in		Edit
O Use Extern	nal Tool for Flash Programming		
Command:			
Arguments:			
	🗖 Run Independent		
Configure Imag Output File:	e File Processing (FCARM):	Add Output File to Group:	
		gr_startup	•

图 4-15 编程的初始化文件

#### 🛄 说明:

"Use Target Driver for Flash Programming"和"Use Debug Driver"是默认选中的。

不要选择"Update Target before Debugging"。

## G@DiX

3. 下载.hex文件。

配置完成以后,点击Keil Toolbar上的 "Downlod" 按钮 III 将.hex文件下载到Flash中。如果下载成功,Keil的 "Build Output" 窗口将显示如下结果。

🛄 说明:

下载过程中,若界面提示"No Cortex-M SW Device Found",说明芯片当前可能处于睡眠状态(即开启了睡眠 模式的工程正在运行),无法直接下载.hex文件到Flash中。开发者需要先按下GR5515 SK板的"RESET"键,并 间隔1秒左右点击"Downlod"按钮 II,重新下载文件。

Build Output	Ļ	x
/**		~
* * * * * * * * * * * * * * * * * * * *		
*GR551x object loading script through download interface (e.g.Jlink, etc).		
*The goal of this script is to load the Keils's object file to the GR551x Flash		
* * * * * * * * * * * * * * * * * * * *		
*/		
//Load object file		
LOAD .\build\load_app.hex		
Erase Done.		
Programming Done.		
Verify OK.		
* JLink Info: Reset: Halt core after reset via DEMCR.VC_CORERESET.		
* JLink Info: Reset: Reset device via AIRCR.SYSRESETREQ.		
Application running		
Flash Load finished at 17:23:59		$\checkmark$
<	>	

#### 图 4-16 下载结果

### 4.6 调试

Keil提供了调试器,支持代码在线调试。该调试器支持设置6个硬件断点和多个软件断点。开发者还可以用多种方式设置调试命令。

### 4.6.1 配置调试器

在启动调试之前,需要配置调试器。点击Keil Toolbar中的"Options for Target"按钮 ≤,打开"Options for Target 'GR551x\_SK'"对话框,选择"Debug"标签页。窗口左边为使用软件仿真调试,右边为使用硬件在线调试。BLE Examples工程使用硬件在线调试,相关的调试器默认配置如下:



📱 Options for Target 'GR551x_SK'	×
Device   Target   Output   Listing   Vser   C/C++   A	Asm Linker Debug Utilities
C Use Simulator <u>with restrictions</u> Settings ☐ Limit Speed to Real-Time	Use: J-LINK / J-TRACE Cortex      Settings
✓ Load Application at Startup ✓ Run to main()	Load Application at Startup 🔽 Run to main()
Initialization File:	Initialization File:
Edit	
Restore Debug Session Settings	Restore Debug Session Settings
E Provinciata E Taulhau	🔽 Perstanista 🔤 Taslhau
I♥ Breakpoints I♥ Toolbox	
✓ Watch Windows & Performance Analyzer	I✓ Watch Windows
Memory Display System Viewer	Memory Display System Viewer
CPU DLL: Parameter:	Driver DLL: Parameter:
SARMCM3.DLL -MPU	SARMCM3.DLL -MPU
Dialog DLL: Parameter:	Dialog DLL: Parameter:
DCM.DLL pCM4	TCM.DLL pCM4
Warn if outdated Executable is loaded	Warn if outdated Executable is loaded
Manage Component Vie	ewer Description Files
0K Can	cel Defaults Help

#### 图 4-17 配置调试器

默认使用的Initialization File: *sram.ini*在SDK\_Folder\build\scripts目录中。开发者可以直接使用该 文件,也可以将该文件复制到自己的工程目录下使用。

#### 🛄 说明:

SDK\_Folder为GR551x SDK的根目录。

初始化文件*sram.ini*中包含一组调试命令,调试过程将执行这些命令。在"Initialization File"栏,点击右侧的"Edit..."按钮打开*sram.ini*文件。*sram.ini*的代码示例如下:

```
/**
* GR551x 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
* GR551x RAM
* assuring that the GR551x 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(0x0000000)
// Load program counter
= RDWORD(0x0000004)
```

```
// Write 0 to vector table register, remap vector
WDWORD(0xE000ED08, 0x0000000)
```

#### 🛄 说明:

Keil支持按照以下顺序执行开发者设置的调试器命令:

- 当 "Options for Target 'GR551x\_SK' > Debug > Load Application at Startup"被使能,调试器会首先载 入 "Options for Target 'GR551x\_SK' > Output > Name of Executable"中的文件。
- 2. 执行 "Options for Target 'GR551x\_SK' > Debug > Initialization File"所指定文件中的命令。
- 当 "Options for Target 'GR551x\_SK' > Debug > Restore Debug Session Settings"包含的选项被选 中,恢复相应的Breakpoints, Watch Windows, Memory Display等。
- 4. 当 "Options for Target 'GR551x\_SK' > Debug > Run to main()" 被选中或者命令g, main被发现位 于Initialization File中,调试器就会自动开始执行CPU指令,直到遇到main()才会停下来。

### 4.6.2 启动调试

完成调试器配置后,点击Keil Toolbar的"Start/Stop Debug Session"按钮 @即可开始调试。

#### 🛄 说明:

需要确保"Connect & Reset Options"均为"Normal",如图 4-18所示。这样可以保证在Start Debug Sessioin之后,点击了Keil Toolbar的"Reset"按钮,程序还能正常运行。

001 402061107		.e		
SN: 483001187		IDCODE	Device Name	Move
Device: J-Link OB-SAM3U128	SWD	⊙ 0x2BA01477	ARM CoreSight SW-DP	Up
HW : V3.00 dll : V6.44a	-			Dava
EW : U-Link OB-SAM3U128 V3 co				Down
Port Max	C Auto	matic Detection	ID CODE:	
SW V SMH7 V	C Man	ual Configuration	Davies Names	
			Device Marrie:	
Auto Clk	Add	Delete Up	odate IR len:	
Connect & Reset Options Connect: Normal 🗸 Reset:	Normal		Downlo	ad Options fy Code Download roload to Elash
Connect & Reset Options Connect: Normal 💽 Reset: 🗍 📝 Reset after Connect	Normal	Cach	Downlo Cache Code Cache Memory Downlo	ad Options fy Code Download rnload to Flash
Connect & Reset Options Connect: Normal  Reset: Reset after Connect	Normal	Cacl	Downlo Cache Code	ad Options fy Code Download mload to Flash Misc
Connect & Reset Options Connect: Normal  Reset: Reset after Connect Interface C USB C TCP/IP Networ	Normal k Settings —		Downlo Cache Code Cache Memory	ad Options fy Code Download nload to Flash Misc
Connect & Reset Options Connect: Normal  Reset: Reset after Connect Interface CUSB C TCP/IP Networ IP-Add	Normal k Settings	Cacler (A	Auto:	ad Options fy Code Download nload to Flash Misc JLink Info
Connect & Reset Options Connect: Normal  Reset: Reset after Connect Interface CUSB C TCP/IP Scan Connect Normal  Networ	Normal k Settings res: . 0 . 0	- Cacl	Auto: De Options Cache Code Cache Memory Auto: De Ping	ad Options fy Code Download mload to Flash Misc JLink Info JLink Cmd

图 4-18 配置 "Connect & Reset Options" 均为 "Normal"

对于XIP模式下的调试,没有额外的注意事项,但是对于Mirror模式还有一些需要额外注意的事项,请参考4.6.3 Mirror模式下的调试。

### 4.6.3 Mirror模式下的调试

在Mirror模式下,开发者必须在Application Firmware被拷贝到RAM后设置Breakpoint。

#### 🛄 说明:

如果Breakpoint被设置在RAM地址范围,Keil会使用Software Breakpoint以节省硬件资源(用BKPT指令替换原 有指令)。开发者设置了Breakpoint之后,Bootloader拷贝Application Firmware到设置了Breakpoint的地址,该 地址上的BKPT指令将会被Application Firmware覆盖,Application就无法停在该地址上。参考ARM Keil官方文 档Breakpoints are not hit when debugging in RAM。

开发者需要在Application的main()开始执行之前设置Breakpoint,具体设置方法如下:

1. 在Application的main()的第一行加入 BKPT (X)。示例代码如下:

```
int main(void)
{
    __BKPT(0);
    app_periph_init();
    /*<init user periph .*/
...</pre>
```

- 2. 点击Keil Toolbar的"Build"按钮完成代码编译链接。
- 点击Keil Toolbar的 "Start/Stop Debug Session" 按钮 
   ④,开始调试。程序开始执行后,会停在 BKPT(0)。

   BKPT(0)。
- 4. 在Application中设置新的断点。
- 5. 按"F10" (注意不是按"F5") 单步执行到下一条代码, 然后开发者可以正常方式继续调试代码。

🛄 说明:

按"<u>F10</u>"是只执行下一条代码行,按"<u>F5</u>"是执行所有剩余代码行。在遇到\_\_BKPT的时候,Keil只能响应"<u>F10</u>"。

### 4.6.4 输出调试Log

GR551x SDK支持自定义输出方式从硬件端口(UART、J-Link RTT或ARM ITM(Instrumentation Trace Macrocell))输出Application的调试Log。为方便开发者使用,GR551x SDK提供了APP LOG模块。

#### 4.6.4.1 配置参数

开发者可以通过以下的宏来配置APP LOG模块输出的模式和硬件端口。

#### 表 4-6 APP LOG模块配置参数

宏	描述	位置
	LOG输出的硬件端口选择。	
	1: 使用UART方式输出Log	SDK_Folder\components\libraries
AFF_LOG_DEVICE_TTPE	2: 使用J-Link RTT方式输出Log	\app_log\app_log.h
	3:使用ARM ITM方式输出Log	
	IOC输出立大差鱼枚式的开关	SDK_Folder\components\libraries
AFF_LOG_COLOK_LINABLE	LOG抽山文平有已俗式的万大。	\app_log\app_log.h
APP LOG TAG ENABLE	LOG	SDK_Folder\components\libraries
	输出文本添加TAG的开关。	\app_log\app_log.h

#### 🛄 说明:

- 阻塞模式会阻塞BLE Stack任务调度。如果Application一次性输出过多的调试Log,可能会破坏执行BLE
   Stack任务的时序,导致一些问题,比如响应不及时或连接断开等。但该模式可以保证所有调试Log都能被完整输出。
- 中断模式是先将需要输出的调试Log缓存,再利用UART中断发送模式将缓存中的调试Log分多次输出。UART硬件在以中断发送模式输出调试Log期间,程序可以继续执行,不会干扰执行BLE Stack任务的时序,但是如果需要输出的调试Log超出了缓存的大小限制,会出现部分调试Log丢失。

若选择使用J-Link RTT方式输出Log时,推荐在SEGGER\_RTT.c中做如下修改:

SEGGER_RTT.C					
	242	-*/			
	243	//			
	244	<pre>// RTT Control Block and allocate buffers for channel 0</pre>			
	245				
	246	attribute((section (".ARMat_0x00805000"))) SEGGER_RTT_CB _SEGGER_RTT;			
	247	<pre>//SEGGER_RTT_PUT_CB_SECTION(SEGGER_RTT_CB_ALIGN(SEGGER_RTT_CB_SEGGER_RTT));</pre>			
	240				

图 4-19 创建RTT Control block并置于地址0x00805000处

#### 4.6.4.2 模块初始化

完成配置后,开发者还需要在外设初始化阶段,通过调用app\_log\_init()设置Log参数,注册Log输出接口、注册Flush接口完成APP LOG模块的初始化。APP LOG模块支持使用标准C库函数printf()和APP LOG API输出调试Log,若使用APP LOG API,还可以利用Log级别、格式、过滤方式等参数设置来优化Log。

APP LOG模块为开发者提供了一些接口,可以方便设置Log级别和Log格式(若只是使用printf(),则可以将Log级别和Log格式的参数值设置为NULL)。

*user\_periph\_setup.c*中的函数app\_log\_assert\_init()展示了初始化APP LOG模块的代码。

• 选择UART方式输出Log

在调用该接口之前,需先调用void bsp\_uart\_init(void)接口初始化硬件UART,该接口默认使用中断方式输出Log,接口代码如下:

## G@DiX

```
static void app log assert 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);
   fault trace db init();
}
void bsp uart init (void)
#if APP DRIVER USE ENABLE
   app uart tx buf t uart buffer;
   app uart params t uart param;
   uart_buffer.tx_buf = s_uart_tx_buffer;
   uart buffer.tx buf size = UART TX BUFF SIZE;
   uart param.id
                                  = APP UART ID;
   uart param.init.baud rate
                                 = APP UART BAUDRATE;
                                 = UART DATABITS 8;
   uart param.init.data bits
   uart_param.init.stop_bits
                                 = UART STOPBITS 1;
   uart_param.init.parity
                                 = UART_PARITY_NONE;
   uart param.init.hw flow ctrl = UART HWCONTROL NONE;
   uart param.init.rx timeout mode = UART RECEIVER TIMEOUT ENABLE;
   uart param.pin cfg.rx.type
                                = APP UART RX IO TYPE;
   uart_param.pin_cfg.rx.pin
                                 = APP UART RX PIN;
   uart param.pin cfg.rx.mux
                                 = APP UART RX PINMUX;
   uart param.pin cfg.rx.pull
                                 = APP UART RX PULL;
                                 = APP UART TX IO TYPE;
   uart param.pin cfg.tx.type
   uart param.pin cfg.tx.pin
                                 = APP UART TX PIN;
                                 = APP UART TX PINMUX;
   uart param.pin cfg.tx.mux
   uart_param.pin_cfg.tx.pull
                                 = APP_UART_TX_PULL;
   uart_param.use_mode.type
                                 = APP UART TYPE INTERRUPT;
   app_uart_init(&uart_param, app_uart_evt_handler, &uart_buffer);
```

#### #else

gpio\_init\_t gpio\_config = GPIO\_DEFAULT\_CONFIG;

## G@DiX

```
gpio config.mode = GPIO MODE MUX;
   gpio config.pin = SERIAL PORT TX PIN;
   gpio_config.mux = SERIAL_PORT_TX_PINMUX;
   hal_gpio_init(SERIAL_PORT_PORT, &gpio config);
   gpio_config.pin = SERIAL_PORT_RX_PIN;
   gpio config.mux = SERIAL PORT RX PINMUX;
   hal gpio init (SERIAL PORT PORT, &gpio config);
   s uart handle.p instance
                                     = SERIAL PORT GRP;
   s uart handle.init.baud rate
                                    = SERIAL_PORT_BAUDRATE;
   s uart handle.init.data_bits
                                   = UART_DATABITS_8;
   s_uart_handle.init.stop_bits
                                    = UART STOPBITS 1;
   s_uart_handle.init.parity
                                    = UART PARITY NONE;
   s_uart_handle.init.hw_flow_ctrl = UART_HWCONTROL_NONE;
   s uart handle.init.rx timeout mode = UART RECEIVER TIMEOUT ENABLE;
   hal uart deinit(&s uart handle);
   hal_uart_init(&s_uart_handle);
#endif
}
void bsp uart send(uint8 t *p data, uint16 t length)
{
#if APP DRIVER USE ENABLE
   app_uart_transmit_async(APP_UART_ID, p_data, length);
#else
   hal uart transmit(&s uart handle, p data, length, 5000);
#endif
}
void bsp uart flush (void)
{
#if APP_DRIVER_USE_ENABLE
  app uart flush (APP UART ID);
#endif
}
```

## GODiX

#### 🛄 说明:

- <sup>°</sup> bsp\_uart\_init接口实现了app\_uart和hal\_uart的硬件端口初始化。其中app\_uart\_init接口使用中断模式初始 化UART硬件端口, hal\_uart\_init接口在bsp\_uart\_init接口中只实现了UART硬件端口初始化,使用中断模式 或者阻塞模式调用API接口的具体方式可参考<u>《GR551x HAL及LL驱动用户手册》</u>"2.23 HAL UART通用驱 动"章节。该接口中的内容用户可重写。
- 。 bsp\_uart\_send接口实现了app\_uart异步(app\_uart\_transmit\_async接口)和hal uart同步 (hal\_uart\_transmit接口)两种方式的输出接口,用户可根据具体的应用选择合适的Log输出方式。该接口 中的内容用户可重写。
- <sup>°</sup> bsp\_uart\_flush接口为uart flush接口,用于中断模式下输出缓存在内存中的未发完的数据。该接口中的内容用户可重写。
  - 选择J-Link RTT方式输出Log

在该方式下调用app\_log\_init()时无需注册Log输出接口和Flush接口,接口代码如下:

```
static void app_log_assert_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, NULL, NULL);
}
```

#### 4.6.4.3 使用方法

APP LOG模块初始化完成之后,开发者可以使用以下四个API输出调试Log:

- APP\_LOG\_ERROR()
- APP\_LOG\_WARNING()
- APP\_LOG\_INFO()
- APP\_LOG\_DEBUG()

如果使用了中断输出模式,可调用app\_log\_flush()函数将缓存中的全部调试Log输出,从而保证在芯片复位或系统睡眠前将调试Log全部输出。

当选择J-Link RTT方式输出Log时,需对J-Link RTT Viewer进行配置,配置方式如图 4-20所示。



🔝 J-Link RTT Viewer V6.51a (beta)   Configuration	?	×
Connection to J-Link		
USB Serial No		
⊖ TCP/IP		
O Existing Session		
Specify Target Device		
CORTEX-M4	~	
Script file (optional)		
Target Interface & Speed		
SWD	◄ 4000 kH;	z 🔻
RTT Control Block		
Address     Search Range	2	
Enter the address of the RTT Control block. Example: 0x20000000		
0x00805000		
ОК	Can	cel

#### 图 4-20 配置J-Link RTT Viewer

其中"RTT Control Block"的位置通过"Address"方式指定,输入值可通过查询编译工程生成的.map映射 文件中"\_SEGGER\_RTT"结构体的地址来获取,如下图所示。若在4.6.4.1 配置参数章节按图 4-19推荐方式创建 了RTT Control block并置于地址0x00805000处,则上图中可直接填入地址"0x00805000"。

store_rssi_addr	0x00804538	Data	4	<pre>lld_test_patch.o(.data)</pre>
preamble_arr	0x0080453c	Data	8	<pre>lld_lcp.o(.data)</pre>
tx_power_cs_tbl	0x00804544	Data	28	<pre>lld_lcp.o(.data)</pre>
CRC_TABLE	0x00804560	Data	1024	<pre>lld_lcp.o(.data)</pre>
CRC_TABLE_7	0x00804960	Data	512	lld_lcp.o(.data)
SEGGER_RTT	0x00805000	Data	120	<pre>segger_rtt.o(.ARMat_0x00805000)</pre>
s_pwr_env	0x0080559c	Data	84	app_pwr_mgmt.o(.bss)
s_uart_env	0x008055f0	Data	616	app_uart.o(.bss)

#### 图 4-21 获取RTT Control Block地址

### 4.6.5 使用GRToolbox调试

**GR551x SDK**提供了用于调试**GR551x BLE**应用的**Android App: GRToolbox**,位于SDK\_folder\tools\GRToolbox\GRToolbox-Version.apk。该**App**提供了如下功能:

- 通用的BLE扫描和连接,对characteristics的读写。
- 标准Profile的Demo展示,包括Heart Rate,Blood Pressure等。
- Goodix自定义应用程序。

# 5 术语和缩略语

#### 表 5-1 术语和缩略语

名称	描述
ATT	Attribute Protocol,属性协议层
BLE	Bluetooth Low Energy,低功耗蓝牙
DAP	Debug Access Port,调试访问端口
DFU	Device Firmware Update,设备固件更新
GAP	Generic Access Profile,通用访问规范
GATT	Generic Attribute Profile,通用属性规范
GFSK	Gauss Frequency Shift Keying,高斯频移键控
HAL	Hardware Abstract Layer,硬件抽象层
НСІ	Host-Controller Interface,主机-控制器接口
ют	Internet of Things,物联网
L2CAP	Logical Link Control and Adaption Protocol,逻辑链路控制与适配协议
LL	Link Layer,链路层
NVDS	Non-volatile Data Storage,非易失性数据存储
ΟΤΑ	Over The Air,用无线传输
PMU	Power Management Unit,电源管理单元
РНҮ	Physical Layer,物理层
RF	Radio Frequency,射频
SCA	System Configuration Area,系统配置区
SDK	Software Development Kit,软件开发工具包
SM	Security Manager,安全管理器
SoC	System on Chip,片上系统
XIP	Execute in Place,片上执行