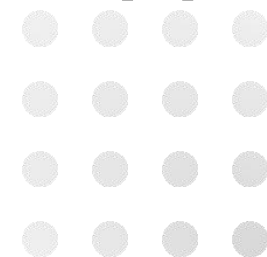


FREOCHIP 富芮坤

FR800X上手 使用介绍



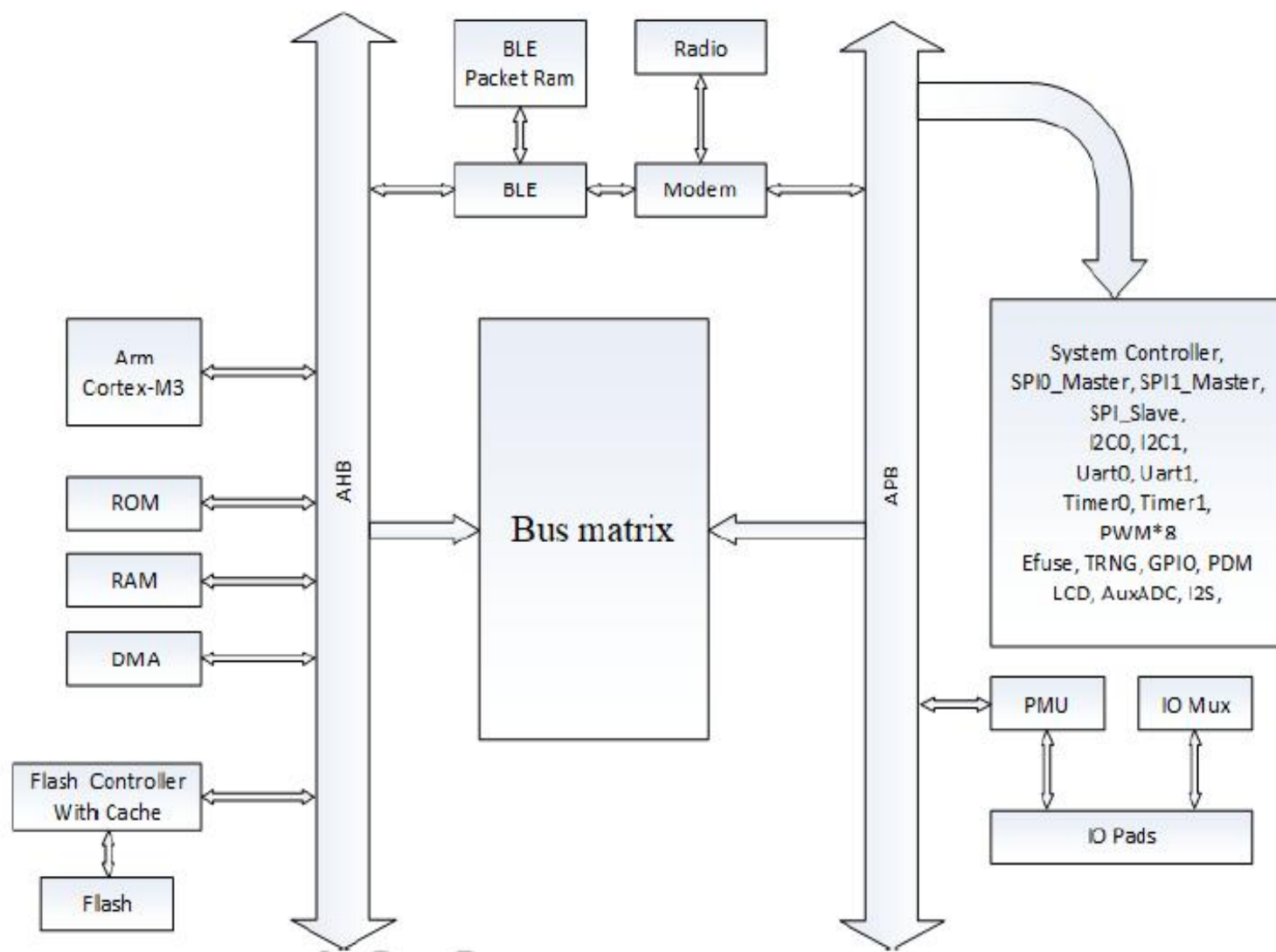
目录

- 
- 1、FR800X硬件结构
 - 2、FR800X基带和调制解调器
 - 3、FR800X 存储结构
 - 4、FR800XSDK介绍
 - 5、FR800X开发环境
 - 6、FR800X烧录
 - 7、FR800X程序流程

FR800X硬件结构

MCU :

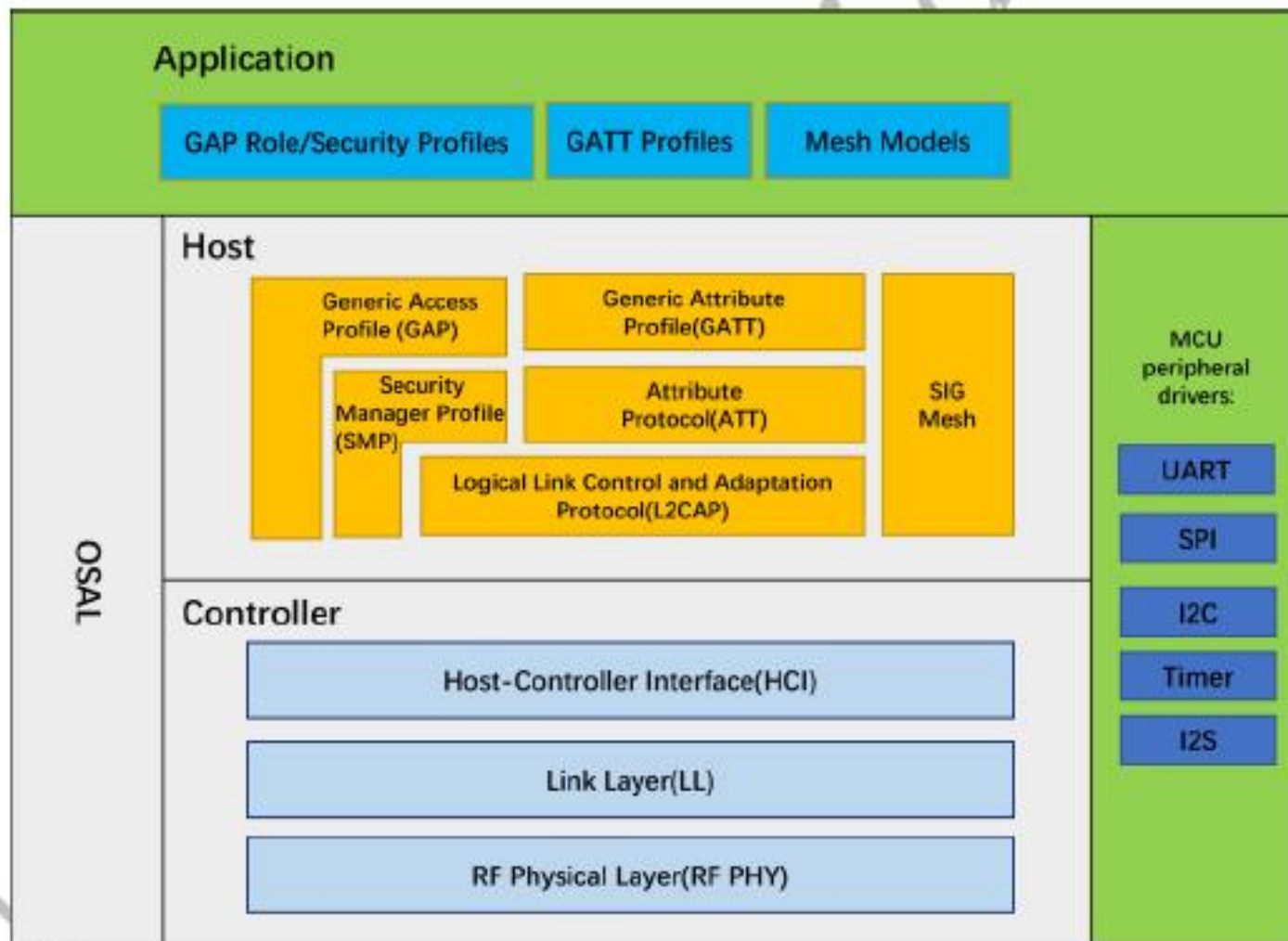
- Cortex-M3 内核
- 48MHz, 96MHz可配置主频
- 64KB RAM
- 128K ROM
- SIP QSPI 512KB Flash
- UART*2
- SPI*2 (支持主从, 1,2,4线制, 4-16位数据宽度, 比特率48Mbps, DMA)
- QSPI*1 (**SIP pSRAM**)
- I2C*2 (支持主从, 位宽8bit)
- TX RX fifo 32)
- USB (usb2.0全速设备)
- Timer*2 (32位, 向下计数)
- PWM*8
- ADC*8(深度64 10位分辨率)
- DMA
- LCD (8080 6800时序)



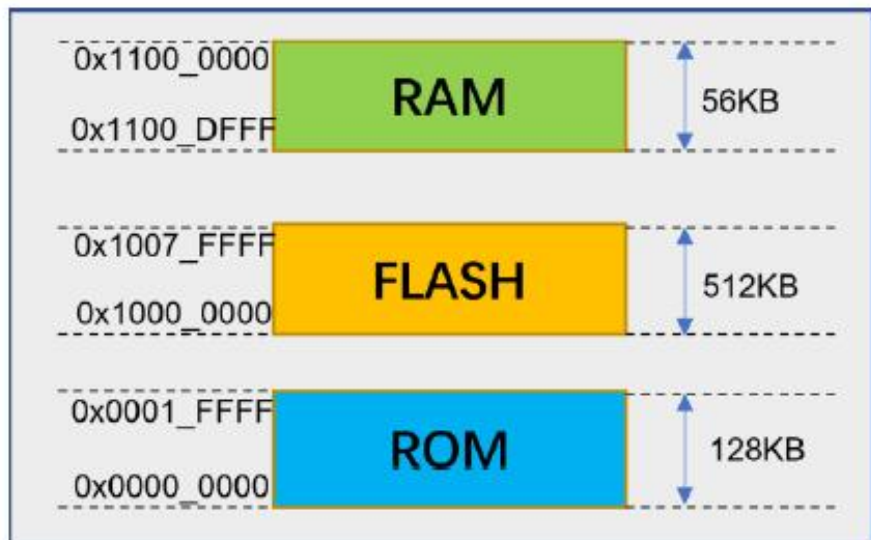
FR800X 基带和调制解调器

基带和调制解调器:

- BLE 5.2
- 8K BLE Packet RAM
- 支持外部 PA
- 发送功率-20dBm~10dBm
- 接收灵敏度最高-101dBm
- PHY 125K/500K/1M/2M
- RSSI 精度1dB
- 支持多协议 SIG mesh, Beacon
- 支持多主多从



FR800X 存储结构



1、运行内存(RAM) 56K + 8K (BLE Packet RAM) :

用于变量，堆栈，重新映射后的中断向量地址，支持低功耗保留功能；

2、Flash :

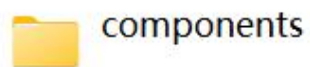
最开始的8K用于存放OTA区域固件信息，最后12K用于存放对端服务，配对信息，本地KEY信息；

3、ROM 128K :

主要为启动代码，BLE Controller 部分协议栈

FR800X SDK介绍

获取SDK方式 <https://www.freqchip.com/fr800x>



components

应用层的依赖库, ble_lib、Profile、deiver



docs

相关说明文档, 包括用户手册



examples

应用层代码, 包含参考例程, gui_demo, nome_evm



tools

包含烧录上位机, J-Link配置文件, Flash描述文件

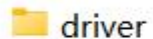


components



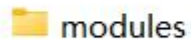
ble

BLE协议栈、Profile文件



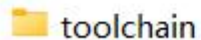
driver

外设驱动文件



modules

可用的软件代码库



toolchain

gcc工具编译相关文件



examples

evm

gui_demo

LVGL 例程

none_evm

ble_AT

AT透传例程

ble_freertos_demo

Free RTOS例程

ble_hid_kbd_mice

HID例程

ble_RT-thread_demo

RT-Thread例程

ble_simple_central

主机例程

ble_simple_peripheral

从机例程

peripheral_demo

驱动外设例程

FR800X开发环境

- 1、Windows 7以上系统的电脑
- 2、准备软件
 - 2·1、Keil V5推荐5.26版本或者更高
 - 2·2、需要安装支持Cortex-M3核的软件包 - MDKCM525.EXE
 - 2·3、J-Link安装包
- 3、添加Flash描述文件
 - 3·1、将Fr800x SDK/Tools/FR8010H.FLM 文件拷贝到如下目录：
C:\Keil_v5\ARM\Flash
- 5、打开fr8000-0.4.10\examples\none_evm\ble_simple_peripheral\keil下的工程文件进行编译下载
- 6、连接线：PC6<-> SWCLK ， PC7<->SWDIO, GND, VCC
- 7、Keil->options for target->Debug->use->选择Jlink
- 8、在flash download选项卡中添加“FR800x 512kB Flash”
- 9、修改RAM for Algorithm的Start: 0x11000000 Size: 0x4000



FR800X烧录

1、打开PC端串口烧录工具，选择正确的串口号，加载烧录文件（选择要烧录的bin文件），然后打开串口，打开图5的区域，进入等待连接状态

2、将串口工具的TX连接到芯片PA0（芯片端的RX），RX连接到芯片的PA1（芯片端的TX）

3、将串口工具的VCC与芯片VBAT连接，最后将串口工具的GND和芯片的GND连接，这时芯片与PC工具握手成功后在工具端会打印 **已连接 自动开始烧录**（若未显示已连接，重新拔插GND，或者按下复位键



FR800X程序流程

`Void user_main(void)`

会在这里初始化时钟 串口 配置协议栈 注册ble事件 广播事件，我们的初始化的代码需要放初始化 SMP之后

`main_loop();`

就是死循环，还有休眠的处理

FR800X MCU用户自定义任务及其他系统接口

user_task.c

创建任务

向任务发送消息

任务回调中处理消息

创建任务	<code>os_task_create</code>
向任务中推送消息	<code>os_msg_post</code>

注意事项：

1. 最多创建20个task
2. 协议栈的调度也是基于这个task
3. 对于一个事件的处理时间不能太长

os_timer

`os_timer_init`

`os_timer_start`

`os_timer_stop`

`os_timer_destroy`

注意事项：

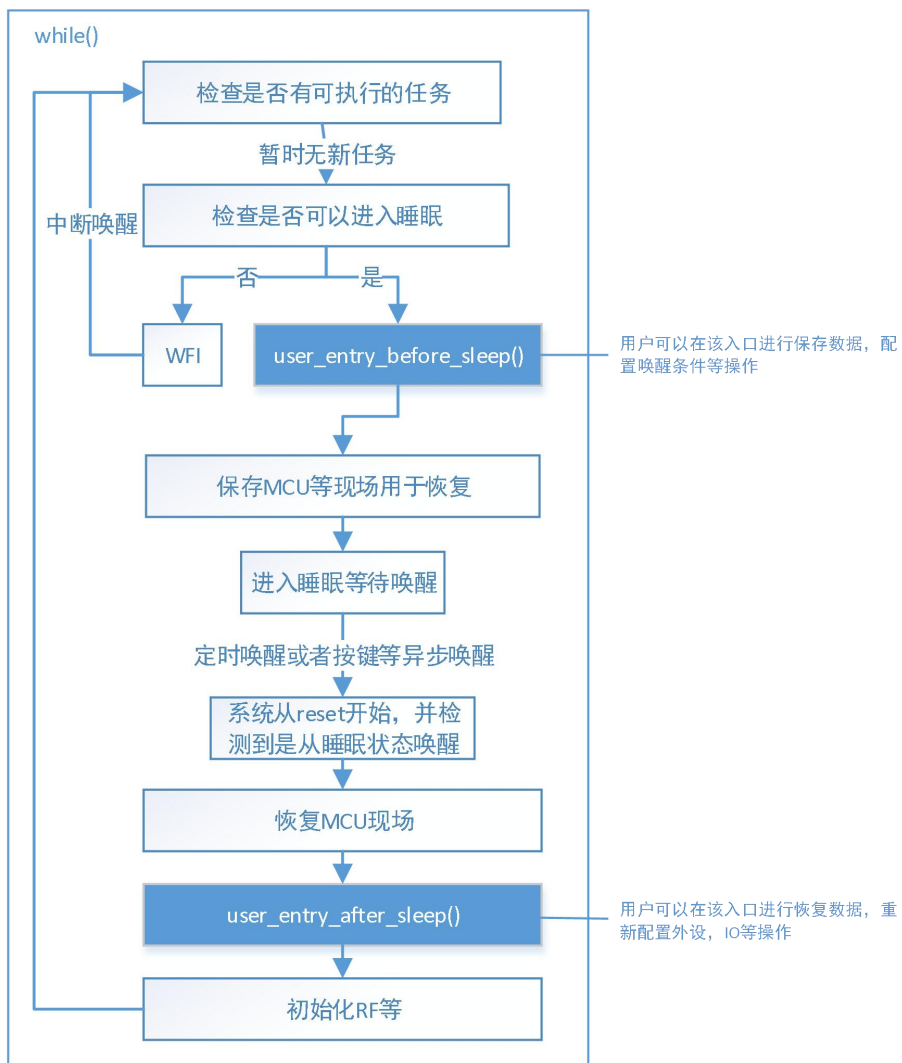
1. 定时精度在10ms，最多创建40个
2. 基于baseband时钟的一个软件timer
3. 对于一个timer事件的处理时间不能太长

OS_MEM

`os_malloc`

`os_free`

FR800x 低功耗机制



入口函数

- 1.user_entry_before_sleep: 该函数在进入睡眠前被调用，用户可在里面实现控制GPIO的状态保持（针对GPIO在系统工作和睡眠状态下的控制参见外设驱动章节），配置睡眠唤醒条件等行为。
- 2.user_entry_after_sleep: 在系统唤醒后，用户可以在该函数中重新进行外设的初始化（进入睡眠后外设的状态因为掉电都会丢失）等操作。

唤醒条件

- 1.同步唤醒：同步唤醒来自一个硬件timer，这个timer的设置由协议栈中代码完成，主要取决于BT pagescan和inquiry sacn间隔、BT sniff间隔、BLE广播间隔、BLE连接间隔等参数，在应用层代码中无需关注。
- 2.异步唤醒：异步主要来自于PMU（电源管理单元）的中断信号，PMU的中断源有：充电器插入拔出、RTC、GPIO状态监测模块等，这些中断源可以在系统初始化时进行设置。

OTA 固件大小和版本

```
/*固件版本号，假设AB区域都有固件，重启的时候底层的boot也会根据版本号运行最高版本好的代码 */  
const struct jump_table_version_t _jump_table_version __attribute__((section("jump_table_3"))) =  
{  
    .stack_top_address = &system_stack[SYSTEM_STACK_SIZE/sizeof(uint32_t)],  
    .firmware_version = 0x00000000,  
};  
  
/*OTA 固件大小，假设需要OTA的固件大小为150K，那么可以修改为0x32000也就是空间为200K */  
const struct jump_table_image_t _jump_table_image __attribute__((section("jump_table_1"))) =  
{  
    .image_type = IMAGE_TYPE_APP,  
    .image_size = 0x32000,  
};
```

可以自定义OTA空间，如果需要把一些用户数据储存到flash，根据需求把需要储存到flash中的数据，储存到代码区之外，也就是 $32000 \times 2 = 64000$ 之后

添加OTA服务

服务的文件统一存放在
components\ble\profiles
目录下

```
Project: ble_simple_peripheral
├── ble_simple_peripheral
│   ├── application
│   │   ├── proj_main.c
│   │   ├── ble_simple_peripheral.c
│   │   └── ota.c
│   ├── profiles
│   │   ├── simple_gatt_service.c
│   │   ├── simple_gatt_service.h
│   │   └── ota_service.c
│   ├── driver
│   │   ├── driver_pmu.c
│   │   └── driver_rtc.c
│   ├── platform
│   │   └── boot_vectors.s
│   ├── lib
│   │   └── fr8000_stack.lib
│   └── modules
│       └── co_log.c
└── components\ble\profiles
    ├── ota.c
    └── ota_service.c
```

```
223  */
224  void simple_peripheral_init(void)
225  {
226      // set local device name
227      uint8_t local_name[] = "FR8000_0000"; //TE; // 修改蓝牙名称
228      mac_addr_t addr;
229      enum ble_addr_type addr_type;
230      gap_address_get(&addr, &addr_type);
231      LOG_INFO(app_tag, "Local BDADDR: 0x%2X%2X%2X%2X%2X%2X\r\n", a
232
233
234      byte2hexchar(addr.addr[1], &local_name[7]);
235      byte2hexchar(addr.addr[0], &local_name[9]);
236
237      byte2hexchar(addr.addr[1], &scan_rsp_data[9]);
238      byte2hexchar(addr.addr[0], &scan_rsp_data[11]);
239      void byte2hexchar(uint8_t byte, uint8_t *buf)
240      gap_dev_name_set(local_name, sizeof(local_name)); // 设置mac
241
242      // Initialize security related settings.
243      gap_security_param_t param =
244      {
245          .mitm = false,
246          .ble_secure_conn = false,
247          .io_cap = GAP_IO_CAP_NO_INPUT_NO_OUTPUT,
248          .pair_init_mode = GAP_PAIRING_MODE_WAIT_FOR_REQ,
249          .bond_auth = true,
250          .password = 0,
251      };
252
253      gap_security_param_init(&param);
254      gap_set_cb_func(app_gap_evt_cb);
255
256
257      // Adding services to database
258      sp_gatt_add_service();
259      ota_gatt_add_service(); // 添加OTA服务, 实现OTA
260
```

添加OTA服务文件

添加OTA服务, 实现OTA

相同外设资源区别

Uart	Uart0: FIFO深度32, 支持流控, 支持IrDA SIR1.0, 时钟可选择48M或96M
	Uart1: FIFO深度16, 不支持流控, 不支持IrDA SIR1.0, 时钟与APB时钟相同。
SPI	SPIM0、SPIM1: FIFO深度32, 支持一线、二线、四线模式
	SPIS: FIFO深度16, 仅支持一线
DMA	通道0: FIFO深度支持64 Byte
	通道1、通道2: FIFO深度支持32Byte

联系我们



上海 张江高科技园区碧波路912弄华依创新园8号楼5楼

青岛 青岛市崂山区科苑纬一路1号国际创新园D2座16层

深圳 宝安区西乡街道共和工业路华丰互联网创意园A座530室



www.freqchip.com



sales@freqchip.com



021-50270080



微信公众号