PAN1070 开发套件使用手册

发布 0.2.0

磐启微电子 PAN1070 项目组

2023 年 11 月 22 日

Table of contents

Chapter 1

PAN1070 ZDK

TBD

Chapter 2

PAN1070 NDK

2.1 NDK 快速入门

2.1.1 NDK 快速入门指南

1 概述

本文是 PAN107x NDK 开发的快速入门指引，旨在帮助使用者快速入门 PAN1070 NDK 的相关开发。

2 PAN107x EVB 介绍

目前只有 PAN107B QFN40 测试板。

3 PAN1070 NDK 开发环境确认

3.1 PC 环境检查 请确认 KEIL(推荐 5.25 版本以上), PAN107x 下载依赖的 flm 文件，Jlink 设备等准备就绪。

3.2 快速编译运行一个简单的例程 硬件接线准备，请确认您已经将 PAN107x 测试板的：

1. SWD（P00：SWD_CLK，P01：SWD_DAT，GND：SWD_GND）接口通过 JLink 连接至 PC
2. SoC UART0 接口通过板上的 USB 转串口模块连接至 PC
   - UART0-Tx：`P16`，UART0-Rx：`P17`
3. 打开一个 sample 工程，例如 `01_SDK\nimble\samples\solutions\esl\Keil` 下的 keil 目录下的工程文件 `esl.uvprojx`
4. 点击 `Build` 编译按钮，然后点击 `Download` 按钮进行下载（若无法正常下载，请检查 FLM 文件是否正常载入）。
5. 下载完成可以通过串口观察 log 输出（**串口波特率：**921600）

2.1.2 NDK 开发环境介绍

开发 IDE

KEIL 官方下载连接：

https://www.keil.com/download/product/

当前 NDK 开发使用的工具为 KEIL, 开发中使用的版本为 5.25, 所以建议使用该版本及以上版本。

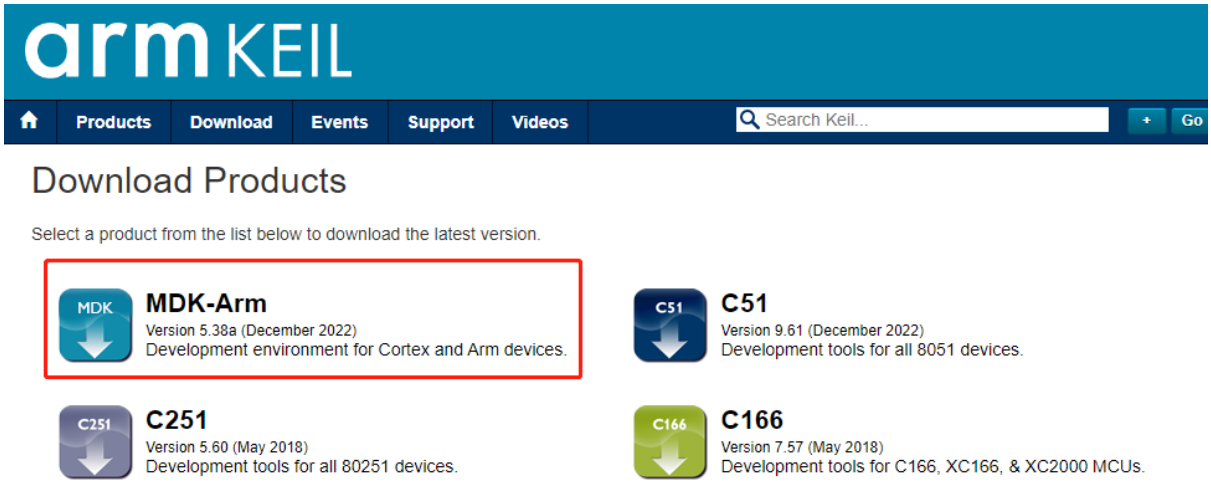


图 1: Keil 下载 MDK-Arm 版本

Keil 使用版本如下：

Keil Flash 下载程序

为使用 Keil + Jlink 烧录代码，请将此目录下的 FLM 文件，拷贝到 Keil MDK 安装目录下，如：

- C:\Keil_v5\ARM\Flash

相关 FLM 文件默认在 SDK 中路径为 `pan1070-ndk\03_MCU\mcu_misc`

- `PAN1080_508KB_FLASH_KEIL.FLM`
- `PAN1080_1020KB_FLASH_KEIL.FLM`

2.1.3 NDK 整体框架介绍

1 简介

PAN1070 NDK 是基于开源蓝牙协议栈 Nimble(Host) 以及开源系统 FreeRTOS，以及私有 BLE Controller 实现完成。Nimble 和 FreeRTOS 均开发源码，BLE Controller 通过标准化 HCI 接口实现。需要注意的是 NDK 依赖的 IDE 主要是 KEIL。

2 NDK 目录结构

PAN1070 NDK 源码树结构如下：

```
<home>/01_SDK
  modules
     hal
  nimble
      README.md
      controller
      host
      lib
      os
      samples
```

- `moudules/hal`：与 ZDK 同根同源，属于外设和 driver 相关的硬件抽象层
- `README.md`：nimble 基本介绍

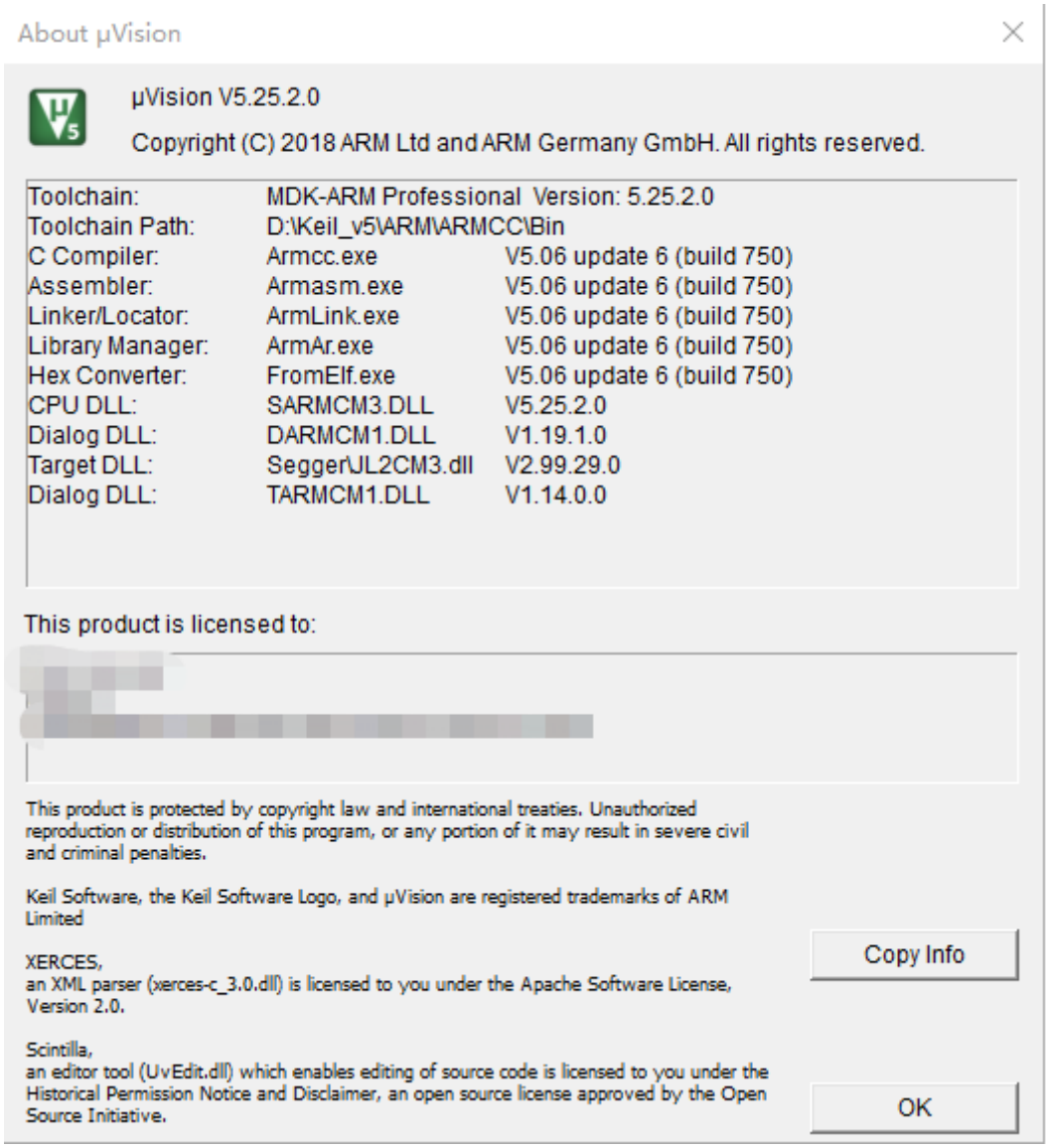

About µVision
µVision V5.25.2.0
Copyright (C) 2018 ARM Ltd and ARM Germany GmbH. All rights reserved.
Toolchain: MDK-ARM Professional Version: 5.25.2.0
Toolchain Path: D:\Keil_v5\ARM\ARMCC\Bin
C Compiler: Armcc.exe V5.06 update 6 (build 750)
Assembler: Armasm.exe V5.06 update 6 (build 750)
Linker/Locator: ArmLink.exe V5.06 update 6 (build 750)
Library Manager: ArmAr.exe V5.06 update 6 (build 750)
Hex Converter: FromElf.exe V5.06 update 6 (build 750)
CPU DLL: SARMCM3.DLL V5.25.2.0
Dialog DLL: DARMCM1.DLL V1.19.1.0
Target DLL: Segger\JL2CM3.dll V2.99.29.0
Dialog DLL: TARMCM1.DLL V1.14.0.0
This product is licensed to:
This product is protected by copyright law and international treaties. Unauthorized reproduction or distribution of this program, or any portion of it may result in severe civil and criminal penalties.
Keil Software, the Keil Software Logo, and µVision are registered trademarks of ARM Limited
XERCES,
an XML parser (xerces-c_3.0.dll) is licensed to you under the Apache Software License, Version 2.0.
Scintilla,
an editor tool (UvEdit.dll) which enables editing of source code is licensed to you under the Historical Permission Notice and Disclaimer, an open source license approved by the Open Source Initiative.
Copy Info
OK

图 2: Keil 版本

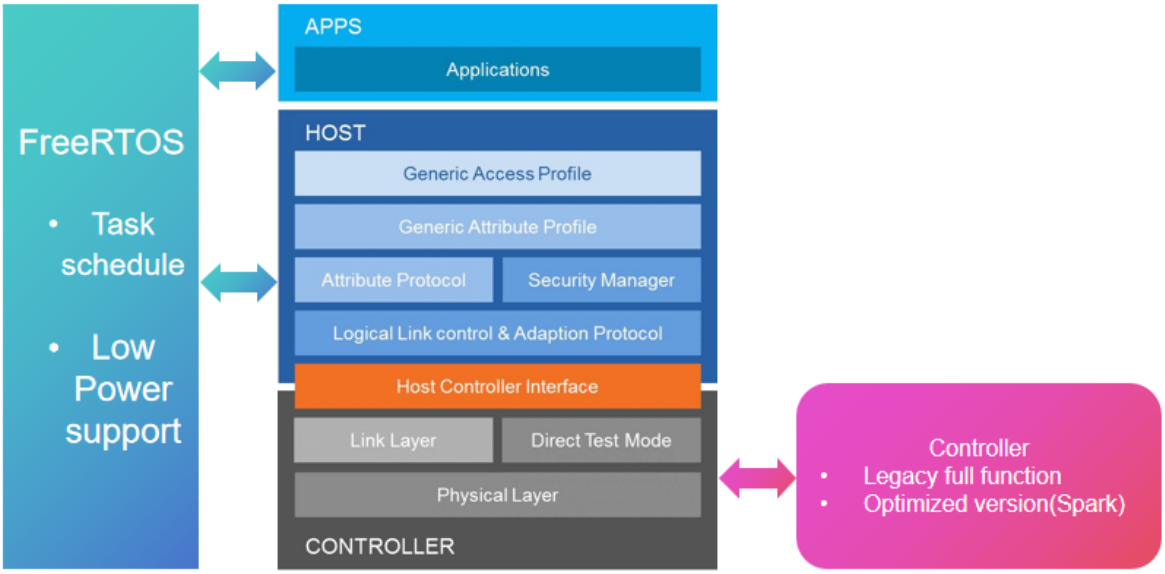


图 3: NDK 整体结构

- `controller`：nimble 工程所需要的 controller 相关头文件
- `host`：nimble host 协议栈所在的主要目录，同时包含 kv_store 组件，该组件主要用于 flash 数据库存储。
- `lib`：该文件包含两个版本 controller 的实现，具体实现以 lib 的形式添加到 keil 工程中。Controller 两个版本分别是 Origin 版本，该版本是全功能版本，但是具体实现优化比较少，执行速度较慢，代码相对较大；另外一个版本是优化版本 (Spark 版本)，该版本为精简和优化版本，速度更快功耗更低，但是目前主要实现 BLE 4.2+ 版本 feature，其他 5.0+feature 功能后续迭代升级。
- `os`:freertos 的相关代码
- `samples`: 基于 nimble 的相关 demo

2.1 modules　modules 在 NDK 中主要为外设以及 USB，2.4G 等依赖的库文件

```
<home>/01_SDK/modules
.
  hal
      panchip
          panplat
              pan1070
                  bsp
                      cmsis
                      device
                      peripheral
                      radio
                      usb
```

2.2 controller　controller 中主要包含了两个版本 controller 依赖的头文件

```
<home>/01_SDK/nimble/controller
.
  dummy.txt
  pan107x
      shrd_utils
  pan107x_spark
      include
```

2.3 host　host 中包含 nimble 的主体实现以及其他必须的组件，目前 nvs 数据库存储使用是的 kv_store 组件，主要用于存储 ble 的配对信息。当然开发者也可以用于自己的项目存储数据。

```
<home>/01_SDK/nimble/host
.
  kv_store
      mtb_init.c
      mtb_kvstore.c
      mtb_kvstore.h
      mtd_kv_port.h
  nimble
      CODING_STANDARDS.md
      LICENSE
      NOTICE
      README.md
      RELEASE_NOTES.md
      apps
      babblesim
      docs
      ext
      nimble
      porting
      repository.yml
      targets
      uncrustify.cfg
      version.yml
```

- `kv_store` 组件：
  - 支持任何可以建模为块设备的存储，包括内部闪存或外部闪存（例如通过 QSPI）。
  - 通过实例化库的多个实例来对存储进行分区。
  - 设计为抗电源故障。
  - 旨在促进存储的均匀磨损。
- `nimble`：

  参考 nimble 专页介绍，nimble 基于 V1.5.0 版本移植实现。

2.4 lib　lib 中主要包含了两个版本 controller 主体实现的库文件:

```
<home>/01_SDK/nimble/lib
.
  pan107x
      ble.lib
  pan107x_spark
      ble_spark.lib
```

Controller 两个版本分别是 `Origin` 版本，该版本是全功能版本，但是具体实现优化比较少，执行速度较慢，代码相对较大，所对应的库为 `pan107x/ble.lib`;

另外一个版本是优化版本，内部代号为 `Spark`, 所对应的库为 `pan107x_spark/ble_spark.lib`。该版本为精简和优化版本，速度更快功耗更低，但是目前主要实现 BLE 4.2+ 版本 feature，其他 5.0+feature 功能后续迭代升级。

2.5 OS　OS 目前只包含 freertos 的主体代码

2.6 samples　nimble 示例工程：

```
<home>/01_SDK/nimble/samples
.
  solutions
      esl
          keil
          src
```

目前包含一个工程：

- `solutions/esl`：电子货架标签方案 Demo 工程。

2.1.4 Nimble 简介

`NDK` 版本基于 `Nimble V1.5.0` 版本

Overview

Apache NimBLE 是一个开源的蓝牙 5.1 协议栈（包括主机和控制器），完全取代了 Nordic 芯片上的专有协议栈。它是 Apache Mynewt 项目的一部分。

特点:

- 支持 251 字节数据包长度。
- 支持 4 种角色并发工作：Broadcaster, Observer, Peripheral and Central。
- 支持 32 个连接并发工作。
- 支持 Legacy 和 SC (secure connections) SMP 配对和绑定。
- 支持扩展广播。
- 支持周期广播。
- 支持 Code phy 和 2M phy。
- 支持蓝牙 mesh[NDK 暂未测试对接]。

支持硬件

目前支持 PAN107x 系列芯片，同时 OS 使用 freertos。

概览

如果您在浏览源代码树，并且想要查看一些主要的功能块，这里有一些指引：

- nimble/controller: nrf 的部分 controller 实现，Pan108x 封装于相关 lib 中。
- nimble/drivers: 射频相关收发 (Nordic nRF51 and nRF52)，Pan108x 封装于相关 lib 中。
- nimble/host: 包含主机子系统的代码。这包括 L2CAP 和 ATT 等协议，支持 HCI 命令和事件，通用访问配置文件（GAP），通用属性配置文件（GATT）和安全管理器（SM）。
- nimble/host/mesh: 包含蓝牙 Mesh 子系统的代码。
- nimble/transport: 包含支持主机和控制器之间的传输协议的代码。这包括 UART、emSPI 和 RAM（在主机和控制器在同一 CPU 上运行时使用的组合构建）。
- porting: 包含支持的操作系统的 NimBLE 移植层（NPL）的实现。
- ext: 包含 NimBLE 使用的外部库。如果操作系统没有提供这些库，则会使用它们。

应用示例

还有一些示例应用程序，展示了如何使用 Apache Mynewt NimBLE 协议栈。如下:

- `ble_central`: 蓝牙主机示例，主要用于演示主机连接从机 ANS 报警通知服务. 该示例可以直接和 `bleprph_enc` 完成对测。
- `bleprph_hr`: 蓝牙从机示例，主要演示从机心跳服务.
- `bleprph_enc`: 蓝牙从机加密示例，主要演示从机特定特性读写时会进行加密配对服务. 可以和 `ble_central` 完成连接对测。但是 `ble_central` 并没有访问加密特性，所以不会触发加密配对流程。

API 接口

如果想在线了解 API 可以参考官方提供的 API 接口指南 https://mynewt.apache.org/v1_10_0/network/ble_hs/ble_hs.html 去了解相关接口功能。

当然也可以通过磐启官方 wiki 了解相关使用指南 https://docs.panchip.com/pan1080dk-doc/latest/index.html。

2.2 硬件资料

文档列表:

2.2.1 PAN107x 硬件参考设计

1 概述

本文档主要介绍 PAN107 系列芯片方案的硬件原理图设计、PCB 设计建议、天线设计。本文档提供 PAN107BUA2A、PAN1070UA1A 芯片的硬件设计方法。

2 原理图设计建议

2.1 PAN107 系列芯片硬件参考设计原理图 如上图电路系统由电源去耦电容、DC-DC 降压、晶振电路、天线匹配网络组成。

2.2 电源

- VBAT 为芯片电源脚，要求供电能力不小于 60mA，供电范围为 1.8V–3.6V 。
- VBAT、VCC_RF、VOUT_BK 电源相关引脚需要至少预留 1 个电容，预留一大一小 2 个电容更佳。电容推荐为 4.7uF 和 100nF。
- 电容靠近芯片引脚摆放，电容焊盘和芯片焊盘之间最大距离不超过 5mm。请遵循指导要求，否则易引起 DC-DC 带不起 RF 以及 EVM 异常。

2.2.1 DC-DC DC-DC 芯片外围电路

1. PAN107 系列芯片 DC-DC 外围电路组成为：2.2 H 电感、100nF 电容、4.7 F 电容。
2. L1 推荐型号：PIM252010-2R2MTS00。选择功率电感，2.2 H，**最小峰值电流为** 150mA，DCR **不超过** 80mΩ，未满足要求在 DC-DC 模式可能会造成 RF 功能异常。
3. DCR 过大会影响 BUCK 效率，能量会转化成热量损耗掉，DC-DC 输出的驱动电流是有限的，效率越低，能够供给到芯片的有效能量就越少。

- DC-DC 的两种工作模式:

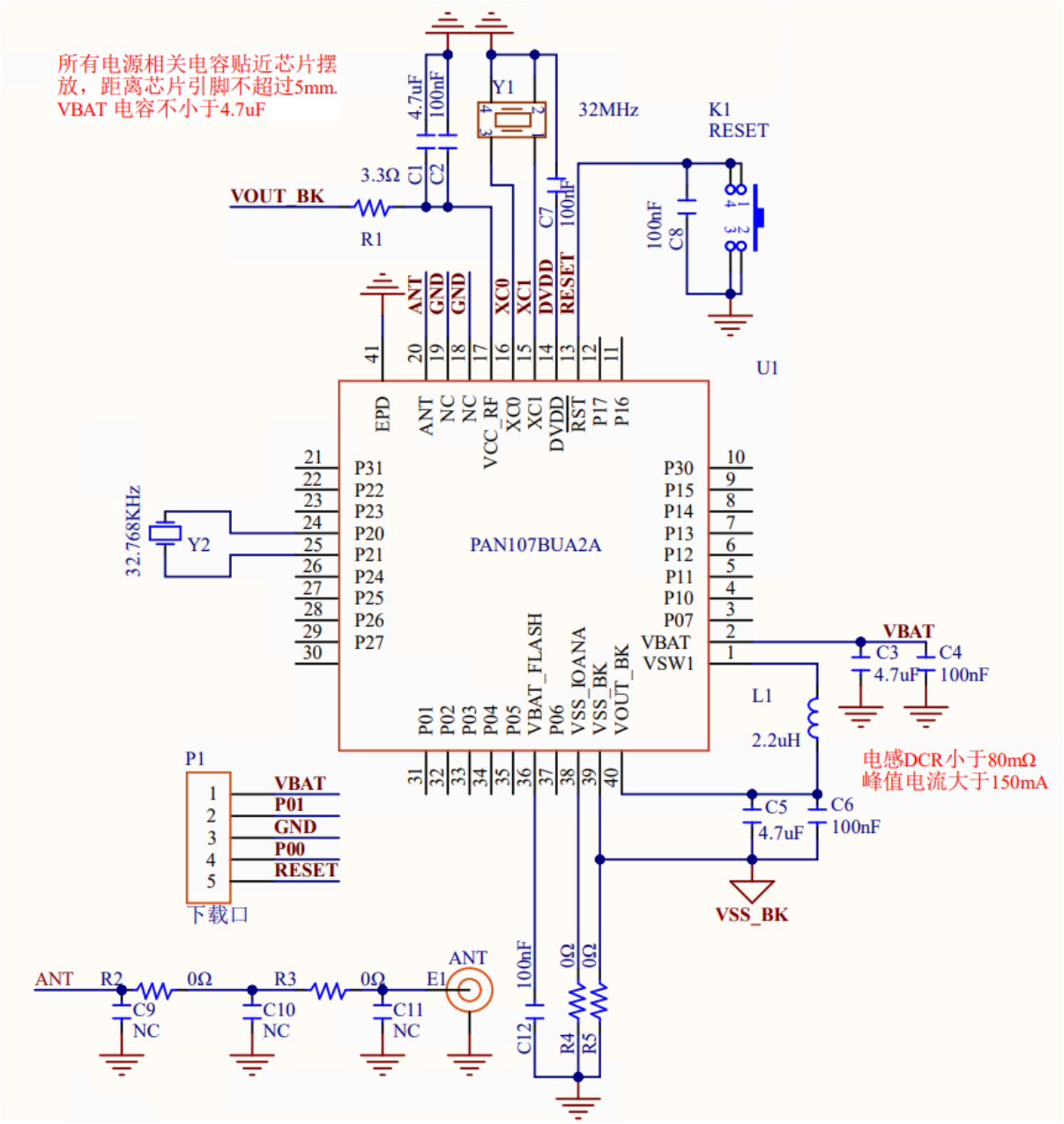


图 4: PAN107BUA2A 最小系统参考设计原理图

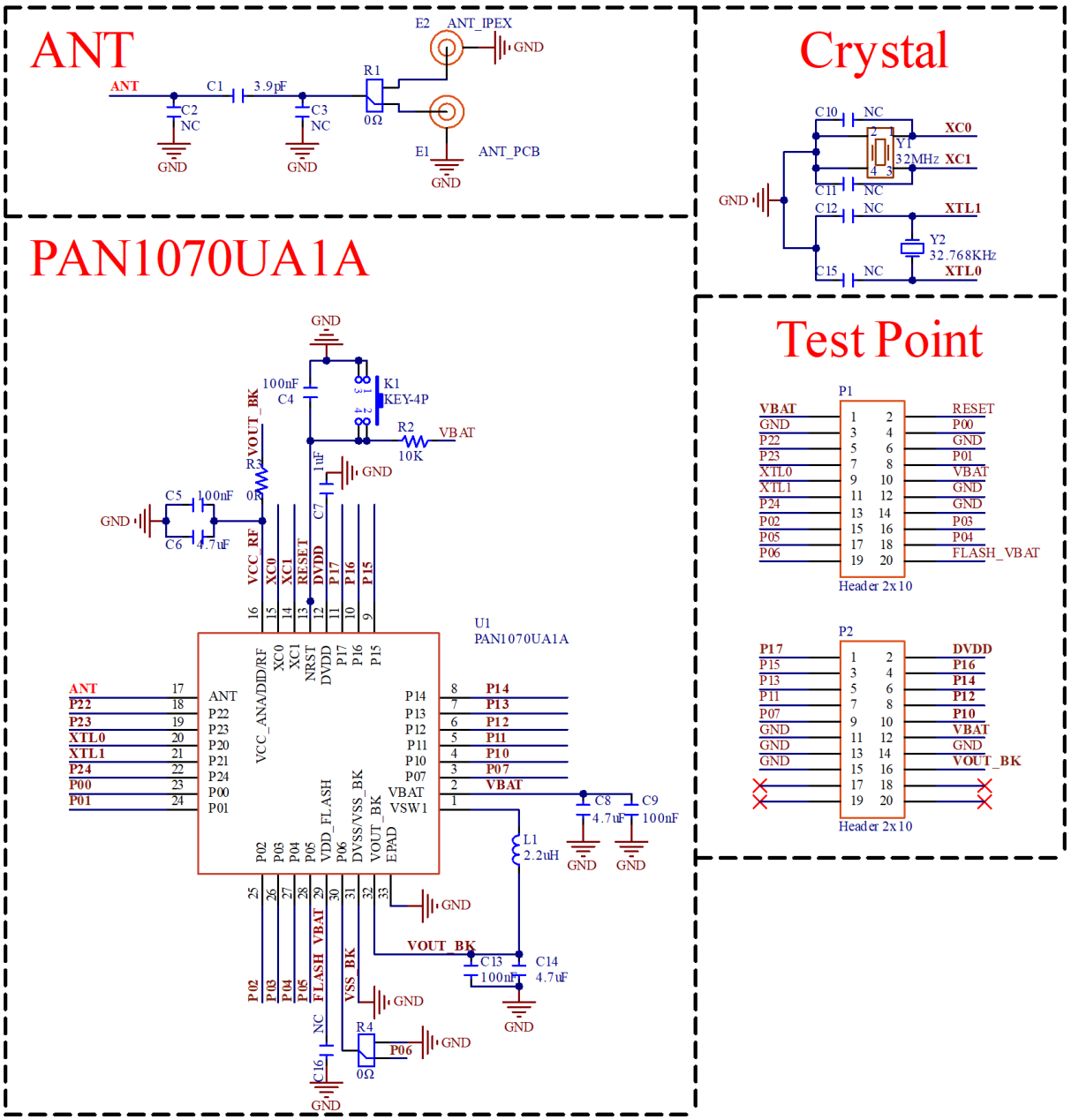


图 5: PAN107UA1A 最小系统参考设计原理图

1. 开启 DC-DC 模式可以降低系统功耗。
2. 开启 LDO（Bypass）模式后芯片内部将 VBAT 连接到 VSW1，这时 VSW1 处的 2.2uH 电感作用为一段导体，可以用 0Ω 电阻替换。
3. DC-DC、LDO 两种模式不能同时开启。
4. 在不考虑功耗的前提下，可将 VCC_RF 直接连接到 VBAT，此时应将电源模式设置为 LDO 模式。

- DC-DC 相关引脚说明：

1. VBAT 为 DC-DC 的供电引脚。
2. VSW1 为 DC-DC 的功率开关（P-MOS）漏极输出引脚，功率电感应靠近该引脚放置。
3. VOUT_BK 为 DC-DC 的反馈引脚，电容应靠近该引脚放置。
4. VSS_BK 为 DC-DC 电源的 GND 引脚。

2.2.2 DVDD　芯片 DVDD 管脚**推荐放置** 100nF **电容**。电容最大不超过 1uF，否则会影响芯片正常启动，电容应靠近该引脚放置。

注意：为避免电路异常，该电容容值请不要随意更改！

2.2.3 VCC_RF　VCC_RF 外部需要接一级 RC 滤波器并尽量靠近该引脚。R1 一般为 3.3Ω、C1 为 4.7uF，截止频率约为 10KHz。请遵循先 R 后 C，电容摆放位置距离芯片引脚不超过 5mm。

2.2.4 VBAT_FLASH　VABT_FLASH 处预留一个 100nF 的电容，默认焊接。

2.3 晶振

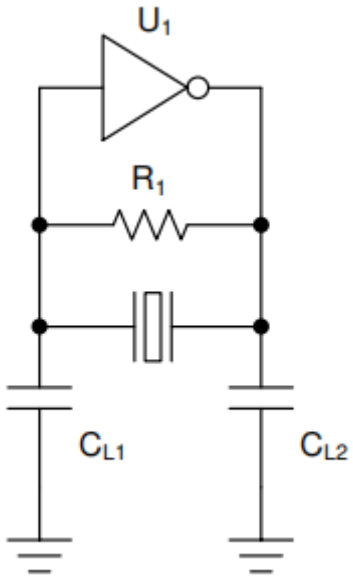


图 6: 32MHz 晶振外围电路示意图

2.3.1 晶振 32Mhz

- 上图振荡器由晶振、反馈电阻 R1、负载电容 CL、放大器构成。可以通过晶体所需负载电容 CL 来调整晶体振荡器频率。
- PAN107x 系列芯片内部集成了电阻 R1、负载电容 CL1、CL2。**只需焊接外部晶体即可工作**。推荐型号为：X322532MOB4SI 32MHz 12pF ±10ppm 。该晶体下的 RF EVM 参数实测效果较好。

- **芯片内部有可调负载电容，不需要外部焊接电容**。需要软件调整负载电容（值 00-0x2f）。该值在出厂前已经校准好，需要软件导入最佳配置值。2.4G 最大允许频偏为 50khz，频偏越小越好。

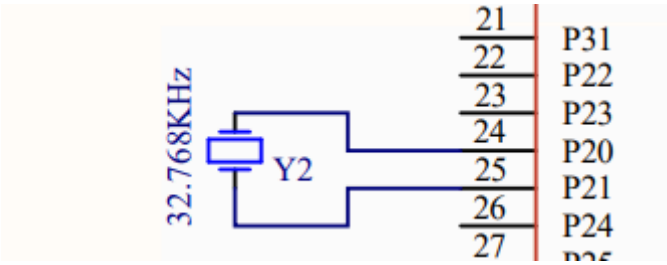


图 7: 32KHz 晶振外围电路原理图

2.3.2 晶振 32.768Khz

- 低速晶振电路支持外部 32.768KHz 无源晶振。**内部有可调负载电容**；低速晶振推荐用户选择 ESR<80KΩ 的晶振。

2.4 复位电路

复位引脚可以悬空，或增加外部按键。在外部按键应用中必须有电容，参数为 100nF。加电容的作用是在系统受到强干扰时，稳定复位脚的电平状态。

> **注意**：该引脚内部有一个 50KΩ 左右的上拉电阻，低电平会使复位生效，为避免电路异常，该电容容值请不要随意更改！

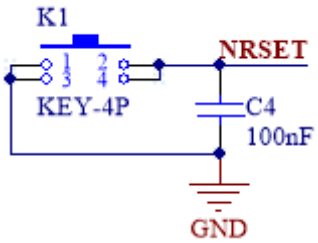


图 8: 复位电路

2.5 静电防护

2.5.1 IO 端静电防护

使用的 IO 要预留串联电阻和 ESD 静电防护元件焊盘位置，便于过认证前的调试整改。串电阻的作用主要是减少 IO 信号的反射、降低外部毛刺信号干扰以及削弱静电对 IO 的影响。**频繁与外部进行数据交换的 IO 例如使用 USB 功能脚等，必须使用 TVS 管进行保护**。建议使用的 TVS 管类型如单向的 ESD5Z3V3 或双向的 CESD923NC5VB，**靠近外设接口位置摆放，**ESD **静电防护元件附近建议保留完整、连续的地，周围打尽量多过孔，有利电荷泄放**。

2.5.2 电源端静电防护

电源输入端 VBAT 建议加上静电防护元件，若有静电进入可快速将电荷泄放到地，尽可能避免损坏芯片。ESD **静电防护元件靠近电源输入端接口摆放**。可选择 ESD5Z3V3。

2.5.3 天线端静电防护

无论是板载天线还是其他天线，本质上都是一段长导体，必然有概率吸引到静电电荷，为预防静电打坏芯片 RF，天线端建议加上静电防护元件，**必须使用低容值 (小于** 3pF) **的** TVS **元器件，尽可能不影响** RF **阻抗，如** BTRD04A035。ESD **静电防护元件靠近芯片摆放，若** RF **阻抗受到影响还可通过 匹配调整**。在天线的位置放置一个 ESD 管。推荐使用有馈地点的天线，板载天线推荐 PIFA。

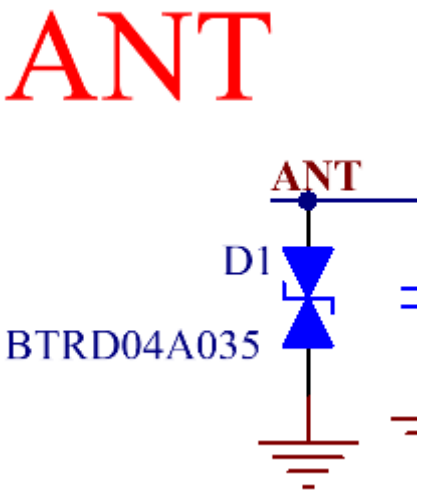


图 9: 天线端静电防护示意图

3 PCB 设计建议

3.1 制版工艺

- 本 Guide 主要针对二层板并且单面贴设计，叠层如下图所示。PCB 具体厚度根据实际情况和阻抗要求适当调整。

	Layer Name	Type	Material	Thickness (mil)	Dielectric Material	Dielectric Constant
	Top Overlay	Overlay				
	Top Solder	Solder Mask/Coverlay	Surface Material	0.4	Solder Resist	3.5
1	Top Layer	Signal	Copper	1.8		
	Dielectric1	Dielectric	None	59.4	FR-4	4.6
2	Bottom Layer	Signal	Copper	1.8		
	Bottom Solder	Solder Mask/Coverlay	Surface Material	0.4	Solder Resist	3.5
	Bottom Overlay	Overlay				

图 10: 制版工艺说明

* 线宽推荐如下：

板材属性	参数
PCB 板材	FR4
PCB 板厚	1.6mm
50 欧姆 RF 线宽	20mil
接地铺铜与 RF 走线间距	5mil

3.2 电源部分注意事项

3.2.1 电源去耦电容布局

- VBAT，VCC_RF, VOUT_BK,DVDD 管脚就近放置电容，走线尽量短粗, 如下图绿色框图部分。

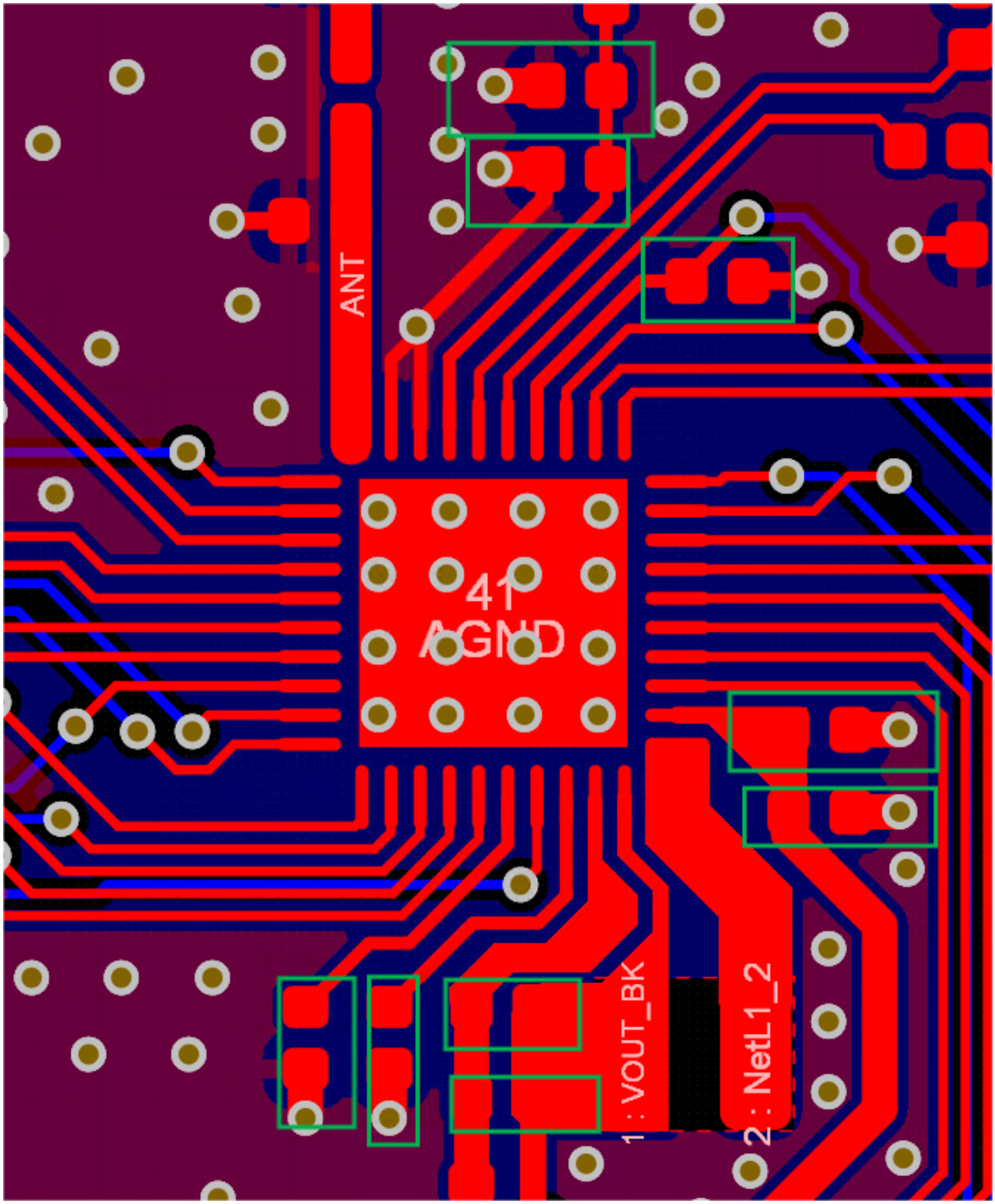


图 11: 电源去耦电容布局

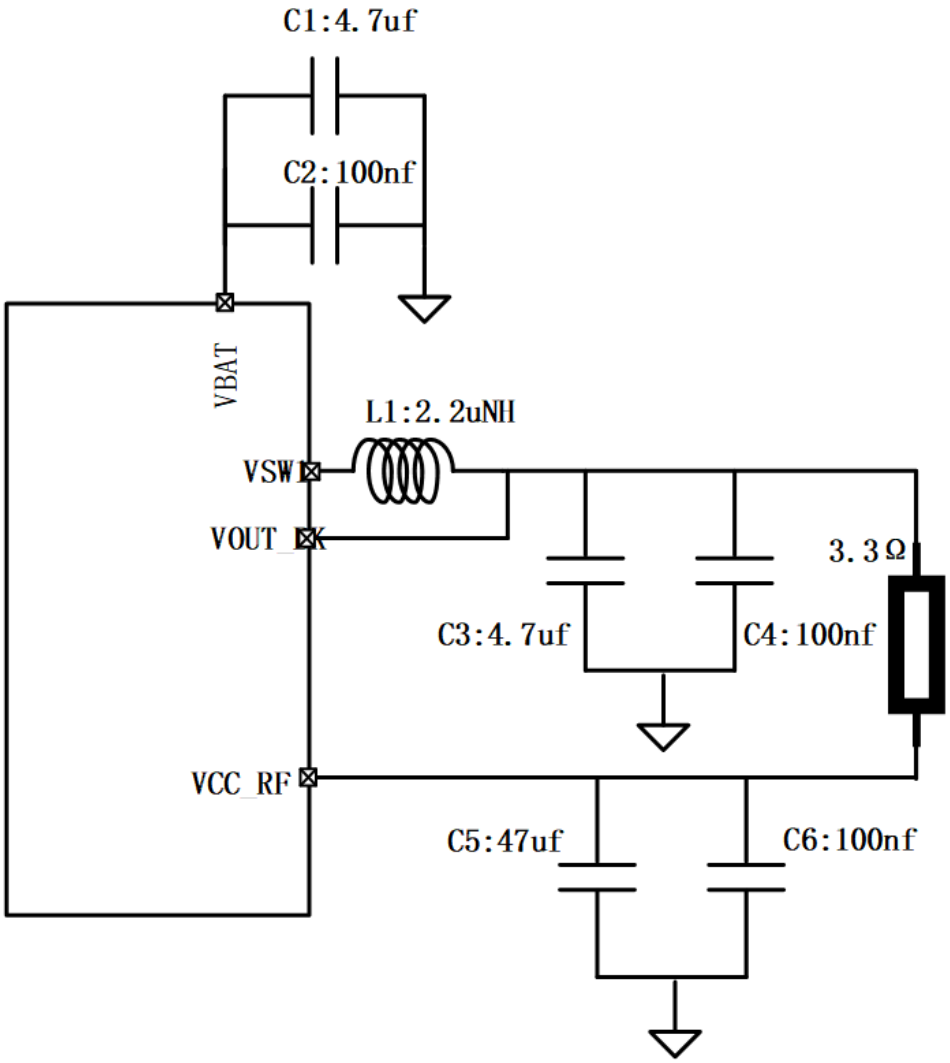


图 12: DC-DC 外围电路

3.2.2 DC-DC PCB 布局参考设计

- VSW1 管脚与电感 L1 距离尽量的短，且附铜面积尽量大。
- L1 与 C3&C4 之间的走线尽量短，且附铜面积尽量大。
- C1&C2 靠近 VBAT_BK 管脚位置摆放，走线尽量短
- VOUT 与 VCC_RF 之间增加磁珠（或小电阻 3.3Ω 左右），磁珠（或小电阻）靠近 C5&C6 摆放
- 电感 L1 尽量远离晶振 XTH
- 电感四周用 GND 隔离
- DCDC 的地不能与 XTH 共用，DCDC 的地通过 0Ω 电阻和其他地连接。DCDC 的地不能直连到 EPAD。

  总体原则为 DCDC **电流回路路径尽可能短；DCDC 地属于干扰源尽量隔离，避免干扰晶振**。

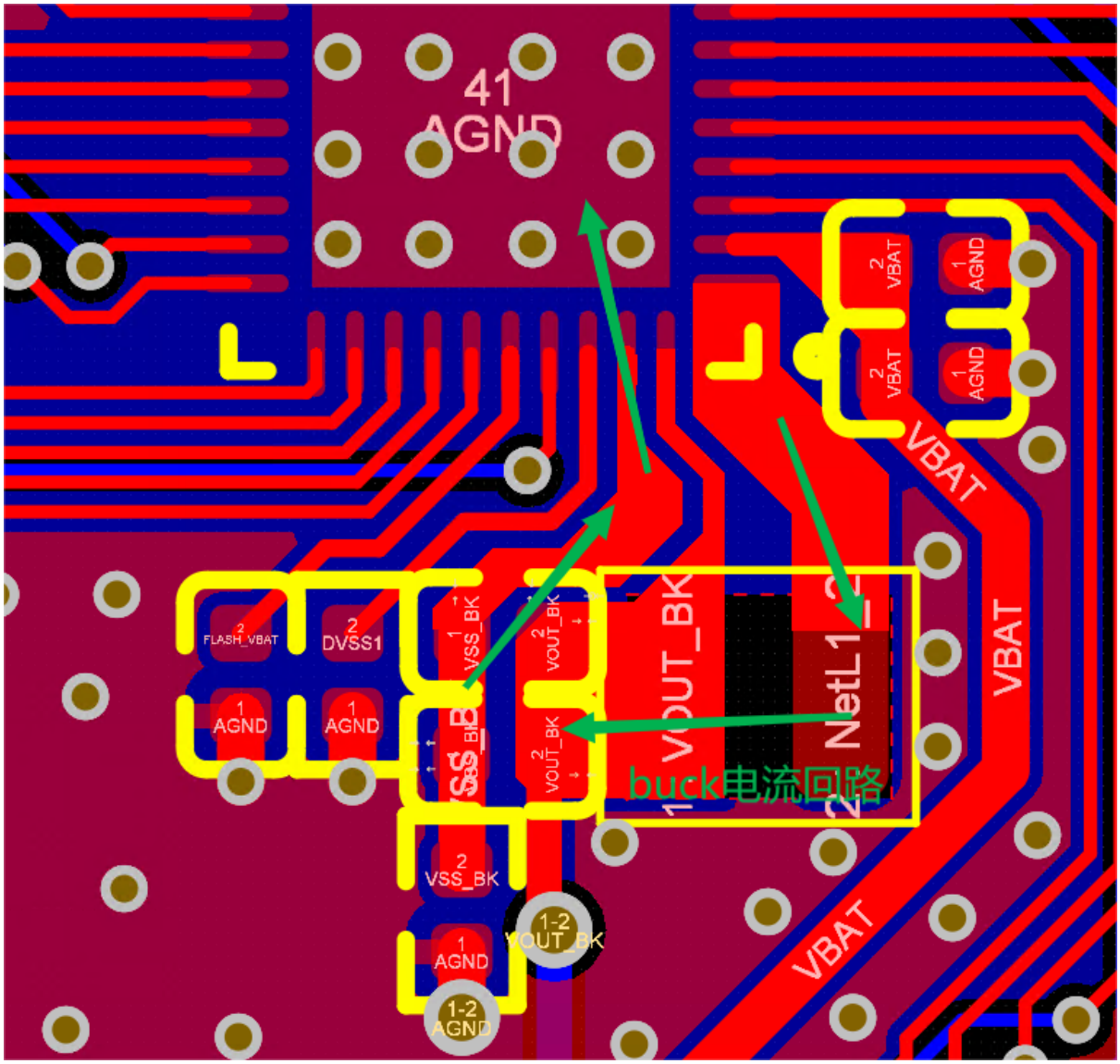


图 13: DC-DC Layout 示意图

3.3 射频走线注意事项

- 晶振尽量靠近芯片引脚摆放。

- 射频匹配链路按照 50Ω 走线，可以参考 TOP 和 BOTTOM 层的 GND 平面，RF 走线尽可能短，RF 线与焊盘宽度一致，天线的 型匹配并联元件焊盘和走线重合为佳。
- RF 线有完整的参考地，从 IC 端出来就进行包地处理，两边打 GND 过孔，底层地平面尽量宽，如**标签** 1 所指信号走线。
- IPEX-1 代天线端子信号引脚挖空，周围包地，尽量减小寄生电容导致阻抗突变，如**标签** 3。
- 芯片底部多打过孔，QFN 封装则打在 E-PAD 上，如**标签** 2。
- 晶振应远离天线，TOP 层挖空，周围包地，降低对电源和 RF 的干扰，需要挖空的部分如**标签** 3。
- 天线辐射区域不要摆放金属器件，净空区挖空处理。

射频链路走线参考如下：

- 天线匹配链路底层不要走线，保证地回路到芯片最短。天线匹配链路的地和芯片 EPAD 是一块完整连续的地。如**标签 "射频地"**。
- 芯片底层不要走线。

射频地线走线如下：

3.4 板载天线

PCB Layout 参考中 MIFA 天线尺寸如图所示。

天线设计尺寸参考

3.5 RF 网络匹配调整

- 如果客户的样板有足够的空间和成本预算来放置匹配的网络组件以及拥有天线调谐能力，我们还建议使用匹配网络。

1. 要进行 RF 匹配调试，建议在芯片引脚附近、靠近天线辐射端都各加入一个 形匹配，并且在两个 匹配之间串上一个 0Ω 电阻便于调试。如下图 RF **网络匹配原理示意图**。
2. 建议先调远端，先进行天线端匹配调试。断开两个 形匹配之间的串联电阻（如 RF **网络匹配原理示意图**中的**标记** 2），使用如矢量网络分析仪可观测天线阻抗、S11 参数的仪器接入到天线端 形匹配网络（如 RF **网络匹配原理示意图**中的**标记** 3 位置），调试天线端阻抗和 S11 驻波。
3. 之后进行芯片输出功率调试，断开两个 形匹配之间的串联电阻，使用如矢量网络分析仪等可观测天线阻抗、S11 参数的仪器接入到芯片端的 形匹配网络（如 RF **网络匹配原理示意图**中的**标记** 1 位置），调整芯片输出阻抗以及功率。
4. 若没有仪器观测阻抗参数，亦可断开两个 形匹配之间的串联电阻，将频谱仪或其他可以检测 RF 输出功率的仪器接入到芯片端的 形匹配网络，通过检测 2402、2440、2480 三个频点的功率值，来调整芯片的输出功率。
5. 最后还需要测试芯片灵敏度情况，因为发射机和接收机内部匹配不完全相等，所以当发射功率调好以后，最后需要确认接收灵敏度情况，如果发射功率调整好后发生接收机灵敏度下降，联系我司工程师修改内部匹配。

   注意：量产前一定要进行距离测试!

4 BOM

最小系统 BOM **参考下表，所有** PAN107x **系列通用：**

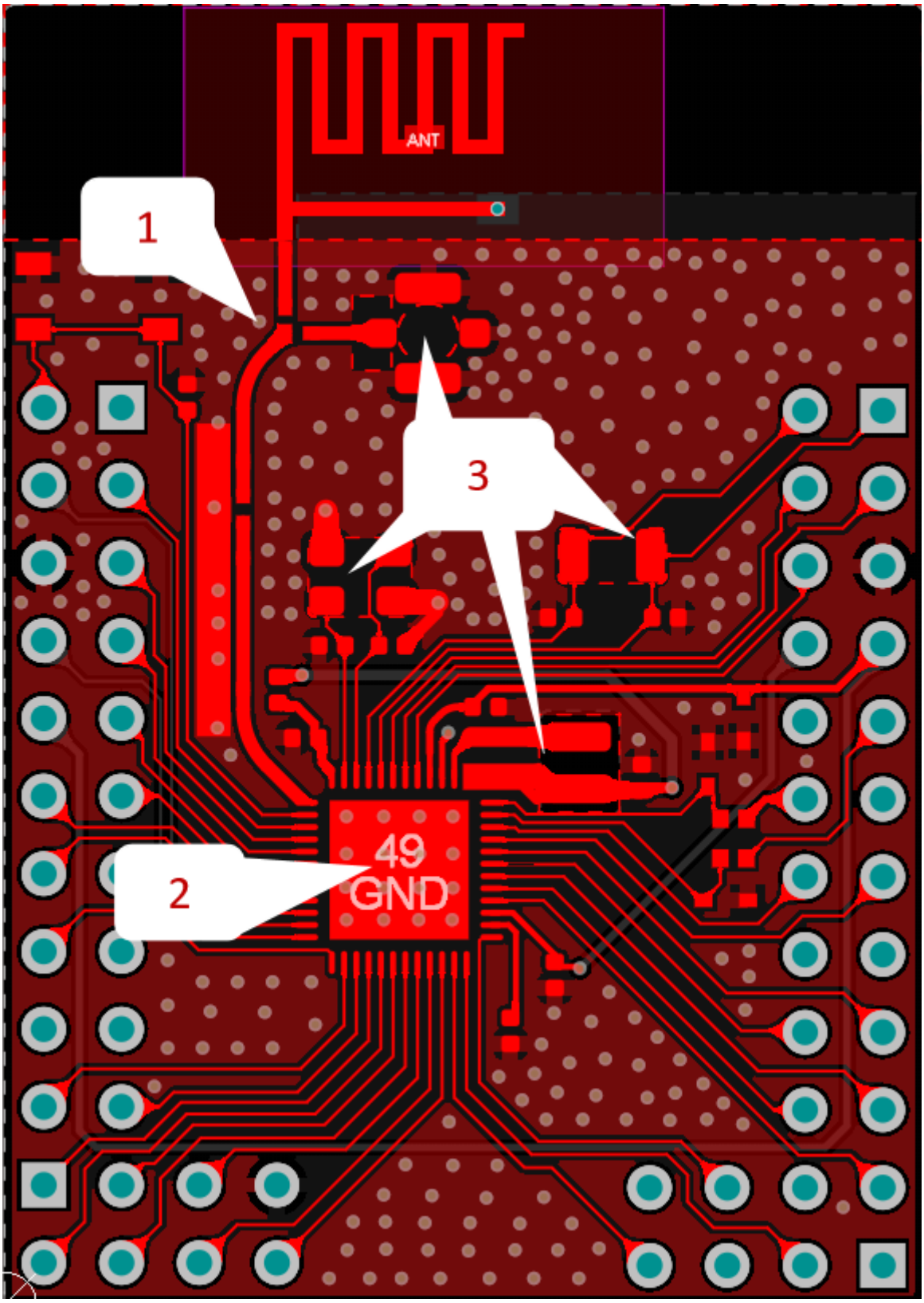


图 14: 射频链路走线示意图

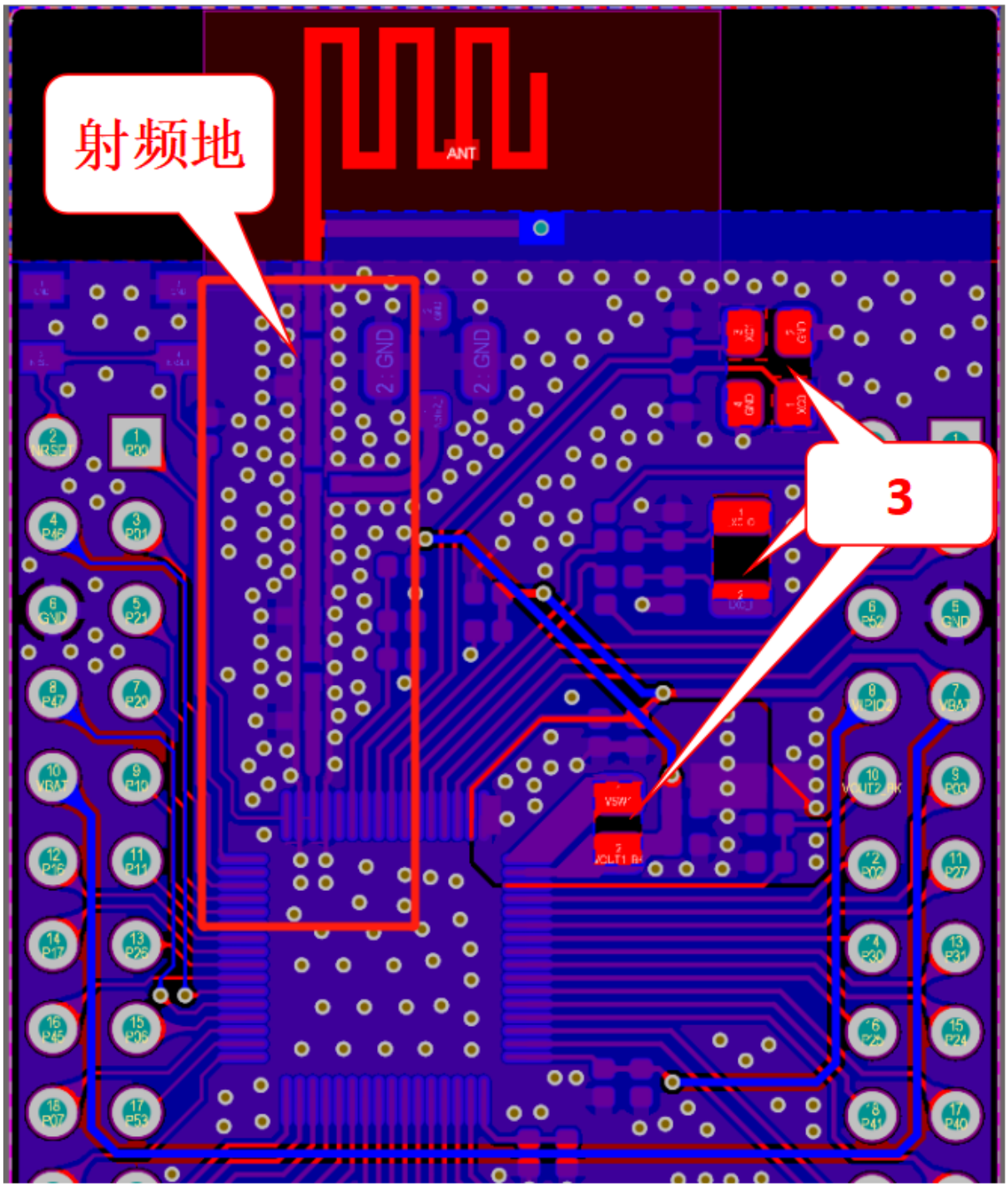


图 15: 射频地线走线示意图

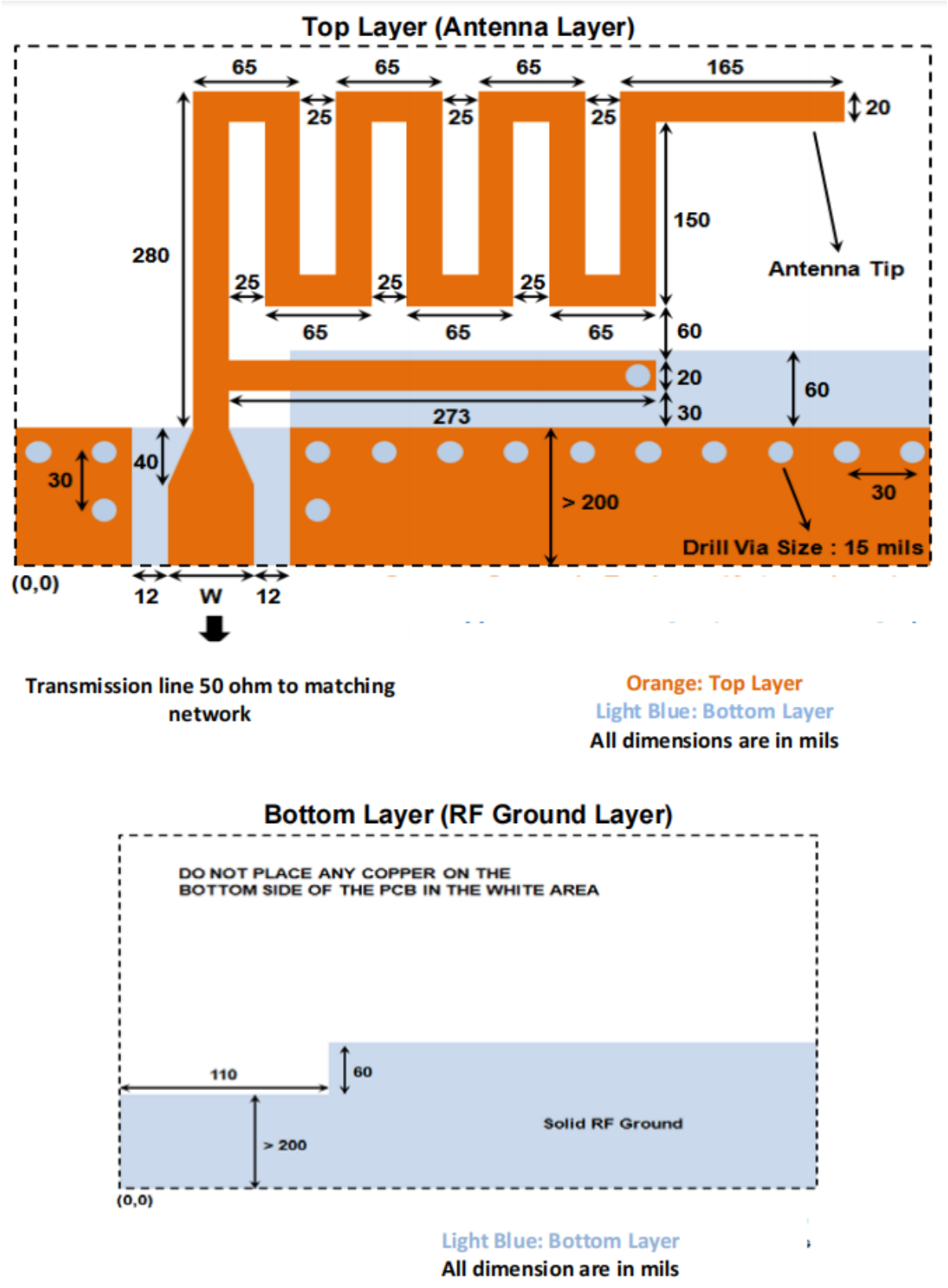
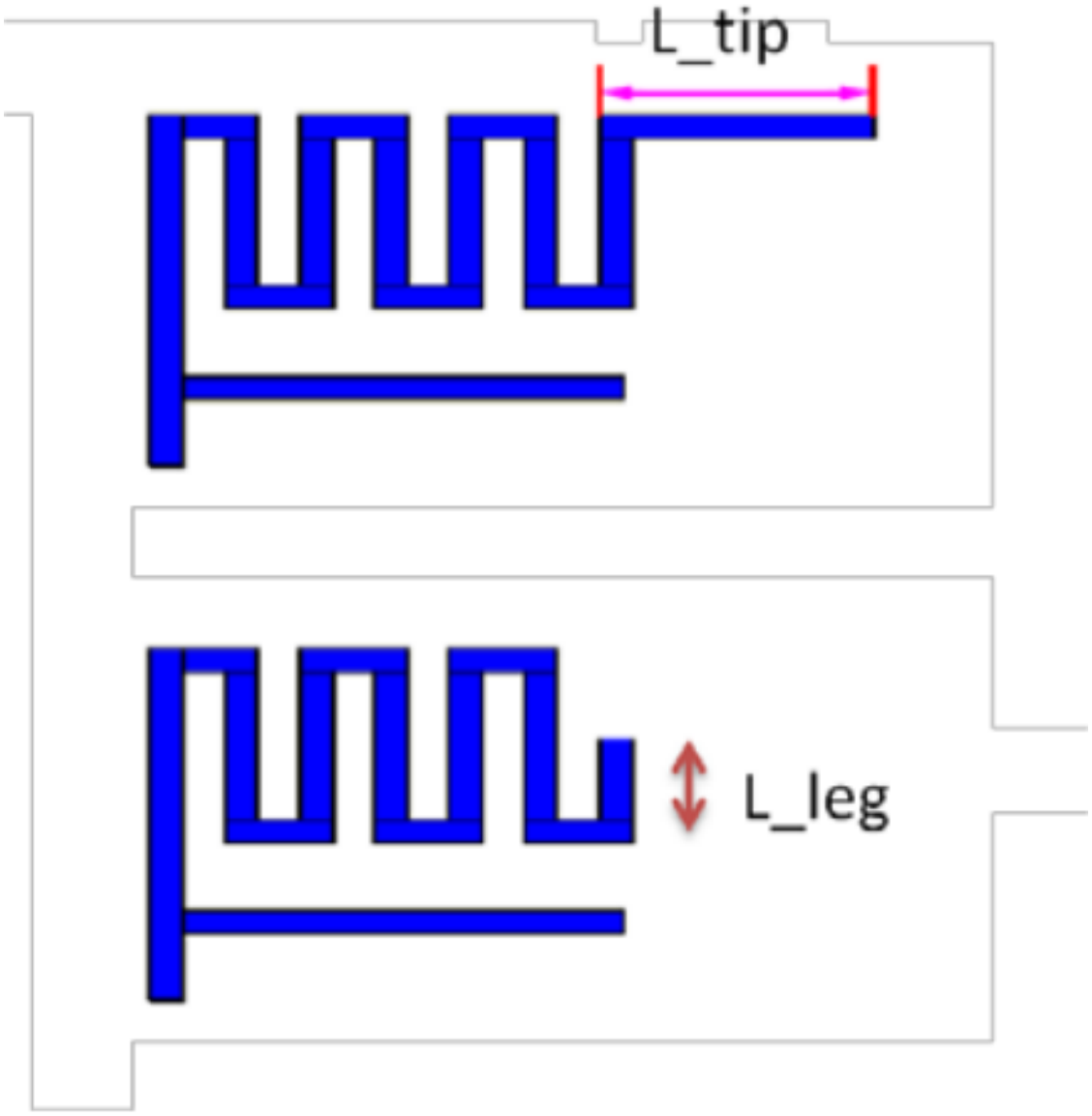


图 16: 天线设计尺寸参考示意图



PCB Thickness	Antenna L_Tip / L_leg
16 mils	L_tip= 353 Mils
31 mils	L_tip= 165 Mils
47 mils	L_tip= 125 Mils
62 mils	L_leg= 115 Mils

图 17: 天线设计匹配参考示意图

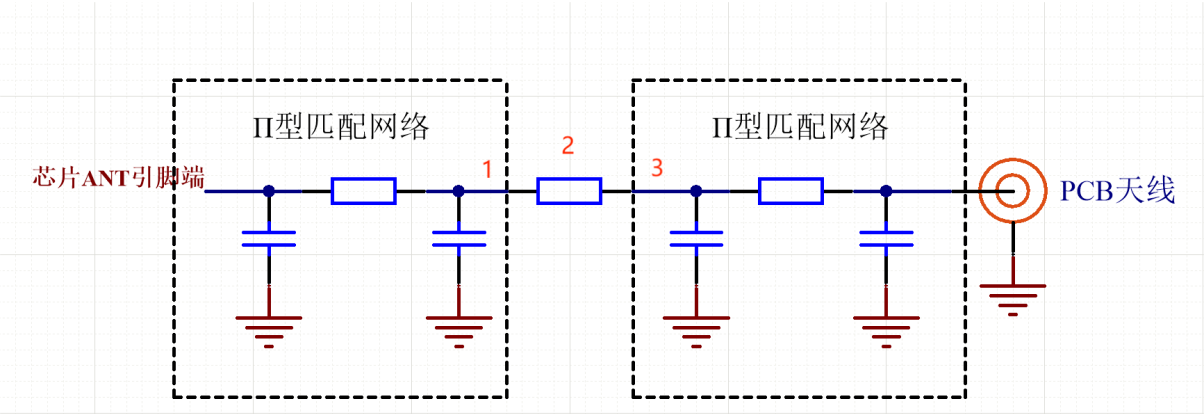


图 18: RF 网络匹配原理示意图

品种	参数	型号	品牌	立创编号	位号	封装	数量
贴片陶瓷电容	4.7uF	0402X475M6R3NT	广东风华高新科技股份有限公司	C168172	C1, C3, C5	0402_C	4
贴片陶瓷电容	100nF	0402B104K160NT	广东风华高新科技股份有限公司	C41851	C2,C4,C6,C7,C8	0402_C	1
贴片按键	4P-4.2mm x 3.25mm	K2-1808SN-A4SW-01	韩荣电子有限公司	C92589	K1	SW-SMD(4.2x3.25x2.5)	1
贴片功率电感	2.2uH	PIM252010-2R2MTS00	广东风华高新科技股份有限公司	C298679	L1	SMD-2520-1.0	1
贴片电阻	3.3Ω 1%	RC-02U3R30FT	广东风华高新科技股份有限公司	C321181	R1	0402_R	1
贴片电阻	0Ω 1%	RC-02000FT	广东风华高新科技股份有限公司	C140225	R2,R3,R4,R5	0402_R	2
贴片 4 脚晶振	32MHz 10ppm 12pF	X322532MOB4SI	深圳扬兴科技有限公司	C9009	Y1	SMD-3225_4P	1
贴片 2 脚晶振	32.768KHz 12.5pF	X321532768KGB2SI	深圳扬兴科技有限公司	C620155	Y2	SMD-3215_2P FC - 135	1
贴片 IC	PAN107BUA2A	\	上海磐启微电子有限公司	\	U1	QFN40	1

HDK **内容**:

硬件开发资料（HDK）位于: `<PAN107X-ZDK>\02_HDK`，其包含如下内容:

02_HDK 子目录	包含内容
PAN107MP_QFN40	PAN107B 芯片测试板硬件设计资料（原理图、PCB 文件等）和生产资料（BOM、gerber、坐标等文件）

2.3 NDK 演示例程

2.3.1 蓝牙例程

BLE Central and Peripheral

1 **功能概述** 此项目演示蓝牙主从一体功能，其实就是整合了 ble_central 以及 ble_peripheral_hr 功能。

- 作为主机：可以直接扫描和连接 ble_peripheral_enc 示例，可以直接下载 ble_peripheral_enc 到另外一块 EVB 板上。
- 作为从机：其实就是一个 ble_peripheral_hr 例程，可以使用手机 nrf_connect app 与其相连。

作为主机和从机的功能可以同时使用。

2 环境要求

- board: pan107x evb
- uart(option): 用来显示串口 log（波特率 921600，选项 8n1）

3 编译和烧录

例程位置：<home>\nimble\samples\bluetooth\ble_cent_prph\keil

使用 keil 进行打开项目选择需要的 controller 进行烧录 ble_cent_prph_spark 或者 ble_cent_prph_spark 进行编译烧录。

4 演示说明

1. 烧录完成后，如果空中有 ble_peripheral_enc 存在，则会主动连接上。同时使用 nrf connect 扫描发发现 cent_prph 设备，连接上后会出现 heartrate service 服务。
   - 主机 log 如下：

```
[15:32:17.967]Try to load HW calibration data.. DONE.
- Chip Type         : 0x80
- Chip CP Version   : None
- Chip FT Version   : 8
- Chip MAC Address  : D0000C0CBBF5
- Chip Flash UID    : 32313334320EAC834330FFFFFFFFFFFF
- Chip Flash Size   : 1024 KB
LL Spark Controller Version:b0e99c4

[15:32:18.011]ble_store_config_num_our_secs:0
ble_store_config_num_peer_secs:0
ble_store_config_num_cccds:0
blehr_advertise

[15:32:19.216]Connection established handle=0 our_ota_addr_type=0 our_ota_
↪addr=06:05:04:03:06:06 our_id_addr_type=0 our_id_addr=06:05:04:03:06:06 peer_ota_addr_
↪type=0 peer_ota_addr=06:05:04:03:02:01 peer_id_addr_type=0 peer_id_
↪addr=06:05:04:03:02:01 conn_itvl=40 conn_latency=0 supervision_timeout=256 encrypted=0␣
↪authenticated=0 bonded=0
/*作为主机连接 slave 设备*/
[15:32:21.923]Service discovery complete; status=0 conn_handle=0

[15:32:22.021]Read complete; status=0 conn_handle=0 attr_handle=12 value=0x00
Write complete; status=270 conn_handle=0 attr_handle=22

[15:32:22.073]Subscribe complete; status=0 conn_handle=0 attr_handle=20

[15:32:27.516]connection established; status=0
/*作为从机被手机主机连接*/
[15:32:30.908]subscribe event; cur_notify=1
 value handle; val_handle=3
```

   - 从机 ble_peripheral_enclog 如下：

```
[15:32:15.447] Try to load HW calibration data.. DONE.
- Chip Type         : 0x80
- Chip CP Version   : None
- Chip FT Version   : 7
- Chip MAC Address  : D0000C06FB6E
- Chip Flash UID    : 31373237304A23094330FFFFFFFFFFFF
- Chip Flash Size   : 1024 KB
LL Spark Controller Version:b0e99c4

[15:32:15.491] ble_store_config_num_our_secs:0
ble_store_config_num_peer_secs:0
ble_store_config_num_cccds:0
registered service 0x1800 with handle=1
registering characteristic 0x2a00 with def_handle=2 val_handle=3
registering characteristic 0x2a01 with def_handle=4 val_handle=5
registered service 0x1801 with handle=6
registering characteristic 0x2a05 with def_handle=7 val_handle=8
registered service 0x1811 with handle=10
registering characteristic 0x2a47 with def_handle=11 val_handle=12
registering characteristic 0x2a46 with def_handle=13 val_handle=14
registering characteristic 0x2a48 with def_handle=16 val_handle=17
registering characteristic 0x2a45 with def_handle=18 val_handle=19
registering characteristic 0x2a44 with def_handle=21 val_handle=22
registered service 59462f12-9543-9999-12c8-58b459a2712d with handle=23
registering characteristic 33333333-2222-2222-1111-111100000000 with def_handle=24 val_
↪handle=25
registering descriptor 34343434-2323-2323-1212-121201010101 with handle=27
Device Address: 01 02 03 04 05 06


[15:32:19.216] connection established; status=0 handle=1 our_ota_addr_type=0 our_ota_addr=01␣
↪02 03 04 05 06
 our_id_addr_type=0 our_id_addr=01 02 03 04 05 06
 peer_ota_addr_type=0 peer_ota_addr=06 06 03 04 05 06
 peer_id_addr_type=0 peer_id_addr=06 06 03 04 05 06
 conn_itvl=40 conn_latency=0 supervision_timeout=256 encrypted=0 authenticated=0 bonded=0


[15:32:22.021] subscribe event; conn_handle=1 attr_handle=19 reason=1 prevn=0 curn=1 previ=0␣
↪curi=0
```

1. 具体 app 功能实现可以参考 `BLE Central` 和 `BLE Peripheral ENC` 的文档。
2. 多连接相关：
   - 对于 controller 层,：指示 origin controller 多连接资源的宏 `CONFIG_BT_CTLR_MAX_NUM_OF_STATES`；指示 spark controller 多连接资源的宏 `CONFIG_BT_CTLR_MAX_MST_CONN` 和 `CONFIG_BT_CTLR_MAX_SLV_CONN`.
   - 对于 host 层：`MYNEWT_VAL_BLE_MAX_CONNECTIONS` 会影响 host 层的连接资源数量。

5 RAM/Flash 资源使用情况

Origin Controller:

```
FLASH:      147.35 KB
SRAM:       37.19 KB
```

Spark Controller:

```
FLASH:      109.10 KB
SRAM:       39.04 KB
```

BLE Central

1 功能概述 此项目演示蓝牙主机功能，通过扫描其他 BLE 设备，并通过特定的服务 UUID 进行识别，此处是 ANS 服务 0x1811。作为对端，可以直接使用 bleprph_enc 进行编译下载另外一个 EVB 上和主机 sample 完成测试。

2 环境要求

- board: pan107x evb
- uart(option): 用来显示串口 log（波特率 921600，选项 8n1）

3 编译和烧录 例程位置: <home>\nimble\samples\bluetooth\ble_central\keil

使用 keil 进行打开项目选择需要的 controller 进行烧录 ble_central 或者 ble_central_spark 进行编译烧录。

4 演示说明

1. 烧录完成后，设备会一直打印收到的广播信息，连接上会显示 Connection established, 服务发现完成后输出 Service discovery complete。

```
[10:48:46.565]LL Controller Version:bd5923c

[10:48:46.603]ble_store_config_num_our_secs:0
ble_store_config_num_peer_secs:0
ble_store_config_num_cccds:0

[10:48:46.673]flags=0x06
    uuids16(complete)=0xe0ff
    mfg_data=0x53:0x50:0x04:0x11:0x23:0x00:0x00:0x00:0x00:0x60:0x13
    flags=0x06
    name(complete)=QHM-C109
    mfg_
↪data=0x06:0x00:0x01:0x09:0x20:0x02:0x3f:0x6b:0x8e:0x3d:0x22:0x86:0x72:0xf0:0x67:0x3e:0x52:0x1f:0x94:0xe1
    flags=0x1a
    tx_pwr_lvl=12
    mfg_data=0x4c:0x00:0x10:0x05:0x1c:0x18:0x03:0x6b:0xee
    flags=0x06
    uuids16(complete)=0xe0ff
    mfg_data=0x53:0x50:0x04:0x11:0x23:0x00:0x00:0x00:0x00:0x60:0x13
    flags=0x06
    name(complete)=QHM-C109
    flags=0x06
    uuids16(complete)=0xe0ff
    mfg_data=0x53:0x50:0x04:0x11:0x23:0x00:0x00:0x00:0x00:0x60:0x13
    flags=0x06
    name(complete)=QHM-C109
.....................................

[10:48:48.154]flags=0x06
    name(complete)=QHM-C109
    flags=0x06
    uuids16(complete)=0xe0ff
    mfg_data=0x53:0x50:0x04:0x11:0x23:0x00:0x00:0x00:0x00:0x60:0x13
    flags=0x06
    uuids16(complete)=0x1811
    name(complete)=nimble-bleprph
    tx_pwr_lvl=0

[10:48:48.242]Connection established handle=0 our_ota_addr_type=0 our_ota_
↪addr=06:05:04:03:06:06 our_id_addr_type=0 our_id_addr=06:05:04:03:06:06 peer_ota_addr
↪type=0 peer_ota_addr=06:05:04:03:02:01 peer_id_addr_type=0 peer_id_
↪addr=06:05:04:03:02:01 conn_itvl=32 conn_latency=0 supervision_timeout=256 encrypted=0
↪authenticated=0 bonded=0
```

(下页继续)

(续上页)

```
[10:48:50.397]Service discovery complete; status=0 conn_handle=0

[10:48:50.515]Read complete; status=0 conn_handle=0 attr_handle=12 value=0x00
Write complete; status=270 conn_handle=0 attr_handle=22
Subscribe complete; status=0 conn_handle=0 attr_handle=20
```

2. 断链时会有如下输出。

```
disconnect; reason=0x08
handle=0 our_ota_addr_type=0 our_ota_addr=06:05:04:03:06:06 our_id_addr_type=0 our_id_
↪addr=06:05:04:03:06:06 peer_ota_addr_type=0 peer_ota_addr=06:05:04:03:02:01 peer_id_
↪addr_type=0 peer_id_addr=06:05:04:03:02:01 conn_itvl=32 conn_latency=0 supervision_
↪timeout=256 encrypted=0 authenticated=0 bonded=0
```

3. **广播显示和连接过滤调整**

如果广播打印太过频繁，影响查看相关 log 可以手动关掉：

```
blecent_gap_event(struct ble_gap_event *event, void *arg)
{
    struct ble_gap_conn_desc desc;
    struct ble_hs_adv_fields fields;
    int rc;

    switch (event->type) {
    case BLE_GAP_EVENT_DISC:
        rc = ble_hs_adv_parse_fields(&fields, event->disc.data,
                                     event->disc.length_data);
        if (rc != 0) {
            return 0;
        }

        /* An advertisment report was received during GAP discovery. */
        print_adv_fields(&fields); /* 此函数用于输出广播数据，可以屏蔽盖函数关掉 */

        /* Try to connect to the advertiser if it looks interesting. */
        blecent_connect_if_interesting(&event->disc); /* 连接时的过滤函数 */
        return 0;
```

连接时的过滤条件 `blecent_connect_if_interesting`：

```
/**
 * Connects to the sender of the specified advertisement of it looks
 * interesting.  A device is "interesting" if it advertises connectability and
 * support for the Alert Notification service.
 */
static void
blecent_connect_if_interesting(const struct ble_gap_disc_desc *disc)
{
    uint8_t own_addr_type;
    int rc;

    /*Filter adv by rssi*/
    if (disc->rssi < -70) /* 此处可以调整 RSSI 进行过滤 */
    {
        return;
    }

    /* Don't do anything if we don't care about this advertiser. */
    if (!blecent_should_connect(disc)) {
```

(下页继续)

(续上页)

```
        return;
    }

    /* Scanning must be stopped before a connection can be initiated. */
    rc = ble_gap_disc_cancel();
    if (rc != 0) {
        printf("Failed to cancel scan; rc=%d\n", rc);
        return;
    }

    /* Figure out address to use for connect (no privacy for now) */
    rc = ble_hs_id_infer_auto(0, &own_addr_type);
    if (rc != 0) {
        printf("error determining address type; rc=%d\n", rc);
        return;
    }

    /* Try to connect the the advertiser.  Allow 30 seconds (30000 ms) for
     * timeout.
     */
    rc = ble_gap_connect(own_addr_type, &disc->addr, 30000, NULL,
                         blecent_gap_event, NULL);
    if (rc != 0) {
        printf("Error: Failed to connect to device; addr_type=%d "
                           "addr=%s\n; rc=%d",
                    disc->addr.type, addr_str(disc->addr.val), rc);
        return;
    }
}
```

5 RAM/Flash **资源使用情况** Origin Controller:

```
FLASH:      143.79 KB
SRAM:       36.75 KB
```

Spark Controller:

```
FLASH:      105.03 KB
SRAM:       38.43 KB
```

BLE Peripheral ENC

1 **功能概述** 此项目演示从机 ANS 服务以及自定义加密特性演示功能，可以配合主机 sample`ble central` 演示主从连接，同时也可以配合手机 `nrf connect` 演示配对加密功能。

2 **环境要求**

- board: `pan107x evb`
- uart(option): 用来显示串口 log（波特率 921600，选项 `8n1`）
- 手机 app `nrf connect`

3 **编译和烧录** 例程位置：`<home>\nimble\samples\bluetooth\bleprph_enc\keil`

使用 `keil` 进行打开项目选择需要的 controller 进行烧录 `bleprph_enc` 或者 `bleprph_enc_spark` 进行编译烧录。

4 演示说明

1. 烧录完成后，设备会显示上电 log，连接上会显示 `Connection established`, 主机订阅完成后输出 `subscribe event;` 。

```
[11:46:57.699]LL Controller Version:bd5923c

[11:46:57.736]ble_store_config_num_our_secs:0
ble_store_config_num_peer_secs:0
ble_store_config_num_cccds:0
registered service 0x1800 with handle=1
registering characteristic 0x2a00 with def_handle=2 val_handle=3
registering characteristic 0x2a01 with def_handle=4 val_handle=5
registered service 0x1801 with handle=6
registering characteristic 0x2a05 with def_handle=7 val_handle=8
registered service 0x1811 with handle=10
registering characteristic 0x2a47 with def_handle=11 val_handle=12
registering characteristic 0x2a46 with def_handle=13 val_handle=14
registering characteristic 0x2a48 with def_handle=16 val_handle=17
registering characteristic 0x2a45 with def_handle=18 val_handle=19
registering characteristic 0x2a44 with def_handle=21 val_handle=22
registered service 59462f12-9543-9999-12c8-58b459a2712d with handle=23
registering characteristic 33333333-2222-2222-1111-111100000000 with def_handle=24 val_
↪handle=25
registering descriptor 34343434-2323-2323-1212-121201010101 with handle=27

[11:46:57.796]Device Address: 01 02 03 04 05 06

[11:47:00.271]connection established; status=0 handle=0 our_ota_addr_type=0 our_ota_
↪addr=01 02 03 04 05 06
 our_id_addr_type=0 our_id_addr=01 02 03 04 05 06
 peer_ota_addr_type=0 peer_ota_addr=06 06 03 04 05 06
 peer_id_addr_type=0 peer_id_addr=06 06 03 04 05 06
 conn_itvl=32 conn_latency=0 supervision_timeout=256 encrypted=0 authenticated=0 bonded=0

[11:47:02.454]subscribe event; conn_handle=0 attr_handle=19 reason=1 prevn=0 curn=1␣
↪previ=0 curi=0
```

2. 使用手机 `nrf connect` 扫描蓝牙设备名称 `nimble-bleprph` 并且连接

5 RAM/Flash 资源使用情况

Origin Controller:

```
FLASH:      151.99 KB
SRAM:       35.33 KB
```

Spark Controller:

```
FLASH:      125.59 KB
SRAM:       37.16 KB
```

BLE Peripheral HR

1 功能概述

此项目演示从机 heartrate 服务, 可以配合手机 `nrf connect` 进行演示。

2 环境要求

- board: `pan107x evb`
- uart(option): 用来显示串口 log（波特率 921600，选项 `8n1`）
- 手机 app `nrf connect`

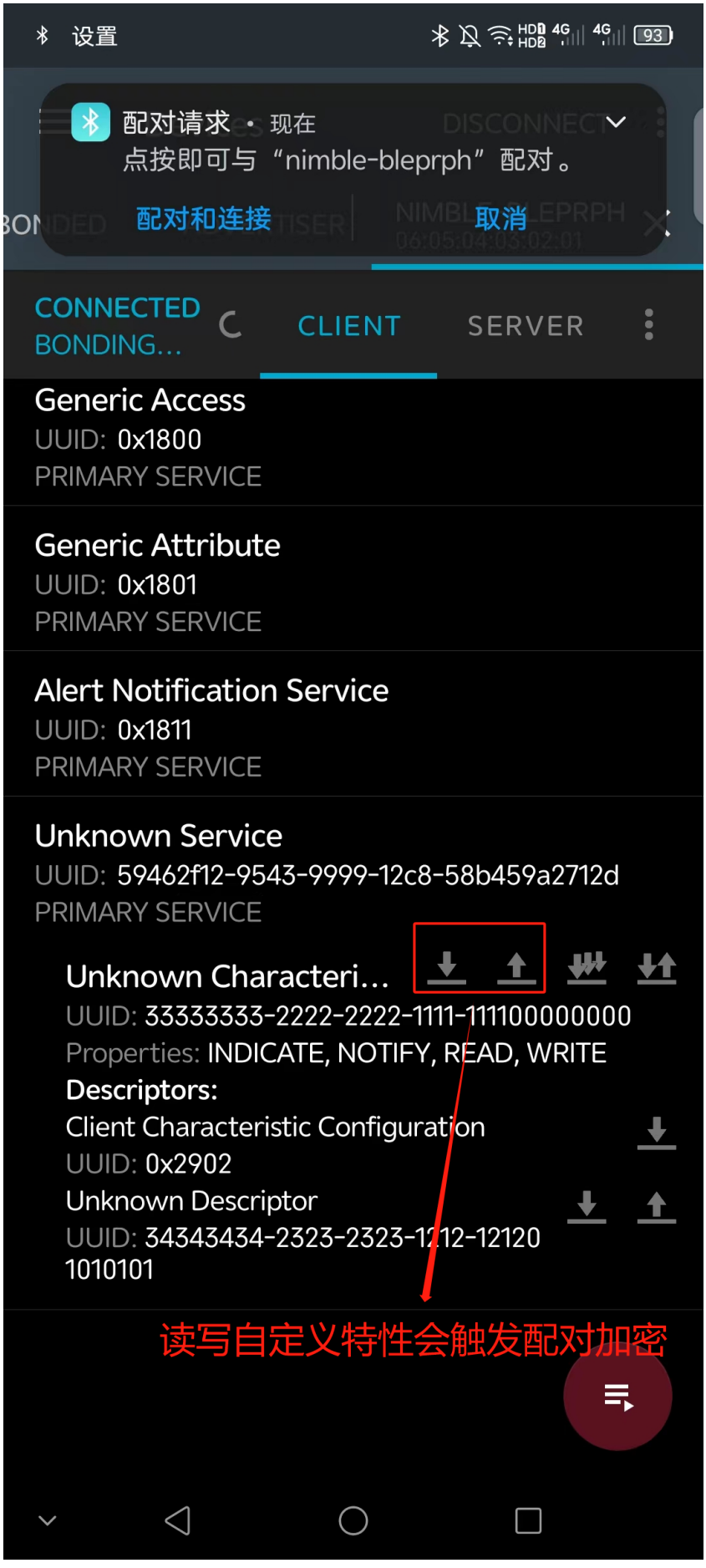


图 19: nrf connect 连接 nimble-bleprph

3 **编译和烧录** 例程位置: `<home>\nimble\samples\bluetooth\bleprph_hr\keil`

使用 `keil` 进行打开项目选择需要的 controller 进行烧录 `bleprph_hr` 或者 `bleprph_hr_spark` 进行编译烧录。

4 **演示说明**

1. 烧录完成后，设备会显示上电 log，连接上会显示 `Connection established`, 主机订阅完成后输出 `subscribe event;` 。

```
[13:17:18.158]LL Controller Version:bd5923c

[13:17:18.197]app started

[13:18:20.460]connection established; status=0

[13:18:26.943]subscribe event; cur_notify=1
 value handle; val_handle=3
```

2. 使用手机 `nrf connect` 扫描蓝牙设备名称 `ble_hr` 并且连接

5 RAM/Flash **资源使用情况** Origin Controller:

```
FLASH:      113.95 KB
SRAM:       33.79 KB
```

Spark Controller:

```
FLASH:      87.75 KB
SRAM:       35.62 KB
```

2.3.2 解决方案

BLE HID Selfie

1 **功能概述** 此项目演示基于 HID 服务的自拍服务，用 K1 和 K2 模拟的音量键，我们可以在相机模式下可以实现拍照功能。

2 **环境要求**

- board: `pan107x evb`
- uart(option): 用来显示串口 log（波特率 921600，选项 `8n1`）
- 手机 app `nrf connect`

3 **编译和烧录** 例程位置: `<home>\nimble\samples\solutions\ble_hid_selfie\keil`

使用 `keil` 进行打开项目选择需要的 controller 进行烧录 `ble_hid_selfie` 或者 `ble_hid_selfie_spark` 进行编译烧录。

4 **演示说明**

1. 在设置蓝牙界面连接 `nimble_hid` 进行配对，通过按键 KEY1 和 KEY2 模拟音量增大和减小，同时利用音量增大键和减小键实现拍照快捷键功能。

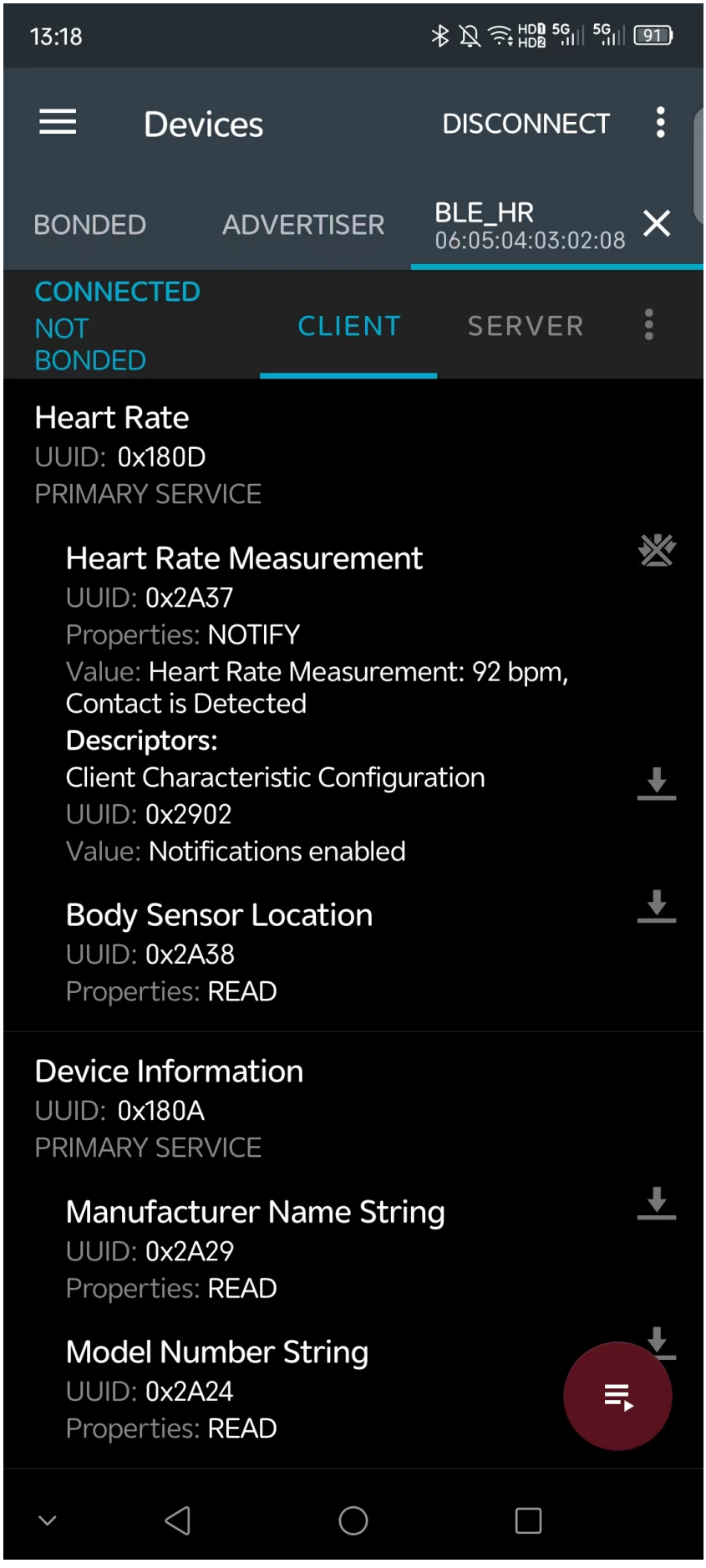


图 20: nrf connect 连接 ble_hr

```
[19:23:23.691]  Try to load HW calibration data.. DONE.
- Chip Type         : 0x80
- Chip CP Version   : None
- Chip FT Version   : 8
- Chip MAC Address  : D0000C0CBBF5
- Chip Flash UID    : 32313334320EAC834330FFFFFFFFFFF
- Chip Flash Size   : 1024 KB
LL Spark Controller Version:b0e99c4

[19:23:23.735]  ble_store_config_num_our_secs:0
ble_store_config_num_peer_secs:0
ble_store_config_num_cccds:0
registered service 0x1812 with handle=1
registering characteristic 0x2a4a with def_handle=2 val_handle=3
registering characteristic 0x2a4b with def_handle=4 val_handle=5
registering characteristic 0x2a4d with def_handle=6 val_handle=7
registering descriptor 0x2908 with handle=9
registering characteristic 0x2a4d with def_handle=10 val_handle=11
registering descriptor 0x2908 with handle=13
registering characteristic 0x2a4c with def_handle=14 val_handle=15
Device Address: 01 02 03 04 05 06


[19:23:42.214]  connection established; status=0 handle=1 our_ota_addr_type=0 our_ota_
↪addr=01 02 03 04 05 06
 our_id_addr_type=0 our_id_addr=01 02 03 04 05 06
 peer_ota_addr_type=1 peer_ota_addr=1e ee c1 b6 69 57
 peer_id_addr_type=1 peer_id_addr=1e ee c1 b6 69 57
 conn_itvl=24 conn_latency=0 supervision_timeout=500 encrypted=0 authenticated=0 bonded=0


[19:23:45.043]  encryption change event; status=0 handle=1 our_ota_addr_type=0 our_ota_
↪addr=01 02 03 04 05 06
 our_id_addr_type=0 our_id_addr=01 02 03 04 05 06
 peer_ota_addr_type=1 peer_ota_addr=1e ee c1 b6 69 57
 peer_id_addr_type=1 peer_id_addr=1e ee c1 b6 69 57
 conn_itvl=24 conn_latency=0 supervision_timeout=500 encrypted=1 authenticated=0 bonded=1

ediv=0 rand=0 authenticated=0 ltk= cddfa0325abc4490ff77622c3075800a irk=␣
↪00000000000000000000000000000000
ediv=0 rand=0 authenticated=0 ltk= cddfa0325abc4490ff77622c3075800a irk=␣
↪4a3711a0c1b7cd2c92d64048e813fe34

[19:23:45.432]  connection updated; status=0 handle=1 our_ota_addr_type=0 our_ota_addr=01␣
↪02 03 04 05 06
 our_id_addr_type=0 our_id_addr=01 02 03 04 05 06
 peer_ota_addr_type=1 peer_ota_addr=1e ee c1 b6 69 57
 peer_id_addr_type=0 peer_id_addr=07 49 34 4d af 50
 conn_itvl=6 conn_latency=0 supervision_timeout=500 encrypted=1 authenticated=0 bonded=1


[19:23:45.610]  connection updated; status=0 handle=1 our_ota_addr_type=0 our_ota_addr=01␣
↪02 03 04 05 06
 our_id_addr_type=0 our_id_addr=01 02 03 04 05 06
 peer_ota_addr_type=1 peer_ota_addr=1e ee c1 b6 69 57
 peer_id_addr_type=0 peer_id_addr=07 49 34 4d af 50
 conn_itvl=24 conn_latency=0 supervision_timeout=500 encrypted=1 authenticated=0 bonded=1


[19:23:46.352]  subscribe event; conn_handle=1 attr_handle=7 reason=1 prevn=0 curn=1␣
↪previ=0 curi=0
```

(下页继续)

(续上页)

```
[19:23:46.412]  subscribe event; conn_handle=1 attr_handle=11 reason=1 prevn=0 curn=1␣
↪previ=0 curi=0

[19:23:48.997]  GPIO ISR in..
P04 occurred  (KEY1 音量增大)
notify_tx event; conn_handle=1 attr_handle=11 status=0 is_indication=0notify_tx event;␣
↪conn_handle=1 attr_handle=11 status=0 is_indication=0key_vol_up pressed

[19:23:49.544]  GPIO ISR in..
P04 occurred
notify_tx event; conn_handle=1 attr_handle=11 status=0 is_indication=0notify_tx event;␣
↪conn_handle=1 attr_handle=11 status=0 is_indication=0key_vol_up pressed

[19:23:50.843]  GPIO ISR in..
P05 occurred  (KEY2 音量减小)
notify_tx event; conn_handle=1 attr_handle=11 status=0 is_indication=0key_vol_down pressed
notify_tx event; conn_handle=1 attr_handle=11 status=0 is_indication=0
[19:23:51.443]  GPIO ISR in..
P05 occurred
notify_tx event; conn_handle=1 attr_handle=11 status=0 is_indication=0key_vol_down pressed
notify_tx event; conn_handle=1 attr_handle=11 status=0 is_indication=0
```

1. hog 相关 GATT Service 的初始化在 gatt_svr.c 中:

```
static const struct ble_gatt_svc_def gatt_svr_svcs[] = {
    {
        /* Service: Heart-rate */
        .type = BLE_GATT_SVC_TYPE_PRIMARY,
        .uuid = BLE_UUID16_DECLARE(BT_UUID_HIDS),
        .characteristics = (struct ble_gatt_chr_def[]) { {
            /* Characteristic: hids information */
            .uuid = BLE_UUID16_DECLARE(BT_UUID_HIDS_INFO),
            .access_cb = gatt_svr_chr_access_hid,
            .flags = BLE_GATT_CHR_F_READ,
        }, {
            /* Characteristic: hids report map */
            .uuid = BLE_UUID16_DECLARE(BT_UUID_HIDS_REPORT_MAP),
            .access_cb = gatt_svr_chr_access_hid,
            .flags = BLE_GATT_CHR_F_READ | BLE_GATT_CHR_F_READ_ENC,
        }, {
            /* Characteristic: hids inout report */
            .uuid = BLE_UUID16_DECLARE(BT_UUID_HIDS_REPORT),
            .access_cb = gatt_svr_chr_access_hid_input_report,
            .flags = BLE_GATT_CHR_F_READ | BLE_GATT_CHR_F_NOTIFY,
            .val_handle = &hid_input_handle,
            .descriptors = (struct ble_gatt_dsc_def[]) { {
                .uuid = BLE_UUID16_DECLARE(BT_UUID_HIDS_REPORT_REF),
                .access_cb = gatt_svr_chr_access_hid_input_report,
                .att_flags = BLE_ATT_F_READ,
            }, {
                0
            }, }
        }, {
             /* Characteristic: hids consumer report */
            .uuid = BLE_UUID16_DECLARE(BT_UUID_HIDS_REPORT),
            .access_cb = gatt_svr_chr_access_hid_consumer_report,
            .flags = BLE_GATT_CHR_F_READ | BLE_GATT_CHR_F_NOTIFY,
            .val_handle = &hid_consumer_input_handle,
            .descriptors = (struct ble_gatt_dsc_def[]) { {
```

(下页继续)

(续上页)

```
                .uuid = BLE_UUID16_DECLARE(BT_UUID_HIDS_REPORT_REF),
                .access_cb = gatt_svr_chr_access_hid_consumer_report,
                .att_flags = BLE_ATT_F_READ,
            }, {
                0
            }, }
        }, {
            /* Characteristic: Body sensor location */
            .uuid = BLE_UUID16_DECLARE(BT_UUID_HIDS_CTRL_POINT),
            .access_cb = gatt_svr_chr_access_hid,
            .flags = BLE_GATT_CHR_F_WRITE_NO_RSP,
        }, {
            0, /* No more characteristics in this service */
        }, }

    }, {
         0, /* No more services. */
    },

};
```

1. 相关 hog 硬件初始化和处理以及发送消息的函数在 `hog.c` 中

5 RAM/Flash **资源使用情况** Origin Controller:

```
FLASH:      155535 B
SRAM:       38216 B
```

Spark Controller:

```
FLASH:      129335 B
SRAM:       40248 B
```

Solution: BLE HID Uart Mult Roles

1 **功能概述** 此 sample 为 `pan107` 上演示蓝牙 HID 串口设备的透传功能，支持 1 主 1 从

2 **环境要求**

- board: `pan107`（芯片型号）开发板 * 3
- uart0: overlay 中设置 P16，P17 作为默认的 LOG 输出端口
- uart1: overlay 中默认 P24 作为 Uart Tx 端-连接开发板 TX0，P10 作为 Uart Rx 端-连接开发板 RX0
- 蓝牙主机设备如手机

3 **编译和烧录** 例程位置：`<home>\nimble\samples\solutions\ble_hid_uart_mult_roles`

使用 `keil` 进行打开项目选择需要的 controller 进行编译烧录。

4 **演示说明**

4.1 AT 指令说明

1. 所有 AT 指令必须以\r\n 字符结尾。广播状态为 AT 指令模式。连接状态为数据透传。AT 指令模式以字符串格式发送。数据透传串口以 hex 格式发送。
2. 存储参数:

```
typedef struct {
        uint32_t baudrate;
        uint8_t own_mac[6];
        uint8_t device_name[28];
        uint8_t name_length;
        uint8_t bond_mac[6];
        uint32_t passkey;
        uint32_t rst_flag;
} fmc_data;
```

全擦除后上电第一次打印默认初始化参数，用户可以后续通过 AT 命令进行修改

```
default_data_init
Baudrate : 115200
Own_mac : 11 22 33 44 55 66
Bond_mac : 11 22 33 44 66 88
Name_length : 9
Device_name : mult_uart
Passkey : 123456
```

3. AT 指令表

AT 指令序号	AT 指令	回复	说明
1	AT	AT+OK	测试串口通讯是否正常
2	AT+RESET	OK+RESET	复位芯片指令
3	AT+DEFAULT	OK+DEFAULT	恢复出厂设置
4	AT+BAUD?	BAUD+ 波特率	10 进制值（1200-115200）
5	AT+BAUD+115200 示例: AT+BAUD+115200	OK+BAUDNO CHANGE BAUDOVER 115200 REJECT	设置波特率 1200-115200 任意值超出配置限制会被拒绝，相等配置同样不会进行设置切换波特率后需要更换波特率通信
6	AT+MAC?	MAC+ 地址	查询 MAC 地址
7	AT+SETMAC+ 地址 示例: AT+SETMAC+112233445566	OK+SETMAC	设置 MAC 地址成功
8	AT+NAME?	NAME+ 广播名字	查询蓝牙广播名字
9	AT+SETNAME+ 名字 示例: AT+SETNAME+HELLO_PAN	OK+SETNAME	设置广播名字成功，最长 28 字节，超过将会截断
10	AT+BONDMAC?	BONDMAC+ 地址	查询扫描过滤条件的绑定地址
11	AT+BONDMAC+ 地址 示例: AT+BONDMAC+112233446688	OK+BONDMAC	设置扫描过滤条件的绑定地址成功
12	AT+PIN?	PIN+ 设置配对密码	作为从机配对时需要在手机端输入密码
13	AT+PIN+ 配对密码 示例: AT+PIN+234567	OK+PIN	设置密码成功
14	AT+ADV START	OK+ADV START	默认开启了广播
15	AT+ADV STOP	OK+ADV STOP	在广播开启条件下可以停止广播
16	AT+SCAN START	OK+SCAN STARTSCAN DONE	依次输出 2 条第一条代表消息通信成功第二条代表扫描到设备后停止扫描
17	AT+SCAN STOP	OK+SCAN STOP	在扫描开启条件下可以停止扫描
18	AT+CONN 00	OK+CONNCONN DONEDISCOVERY DONECCC DONECONN WHOLE DONE	依次输出 5 条第一条代表消息通信成功后四条代表连接后消息交互
19	AT+DISCONN 00	OK+DISCONNDISCONN DONE	依次输出 2 条第一条代表消息通信成功第二条代表成功断连
20	AT+DEV SHOW	OK+DEV SHOW	显示连接列表目前默认在 log 端口显示列表有需要可以移植到透传端口显示
21	其他	AT+ERROR	未定义

注：当前 NDK 烧录代码后，默认开启广播，未开启扫描，命令 14-20 中仅 16 可用，并且扫描到指定 UUID 进行自动连接

1. 连接状态下，发送透传消息以 hex 格式发送，消息格式如下

Pattern(1B)	Send Connect Index(1B)	data(0~200B)
0x5A	发送给 index 的参数为置位 index bit 位 bit0 为发送给从机，bit1 为发送给主机	

4.2 演示流程

4.2.1AT 配置测试

命令 1-13 可以灵活配置和读取默认配置参数，先进行测试后，建议发送 `AT+DEFAULT` 恢复默认配置

注意：

1. 目前指令 5 设置 921600 会返回信息 OVER 115200 REJECT，设置相等值会返回 NO CHANGE BAUD，设置 115200 以下的串口波特率会返回 OK+BAUD RESET 并 reset 芯片，切换波特率可以继续通信测试
2. 部分设置命令会答应 `RESETING...` 进行芯片重启

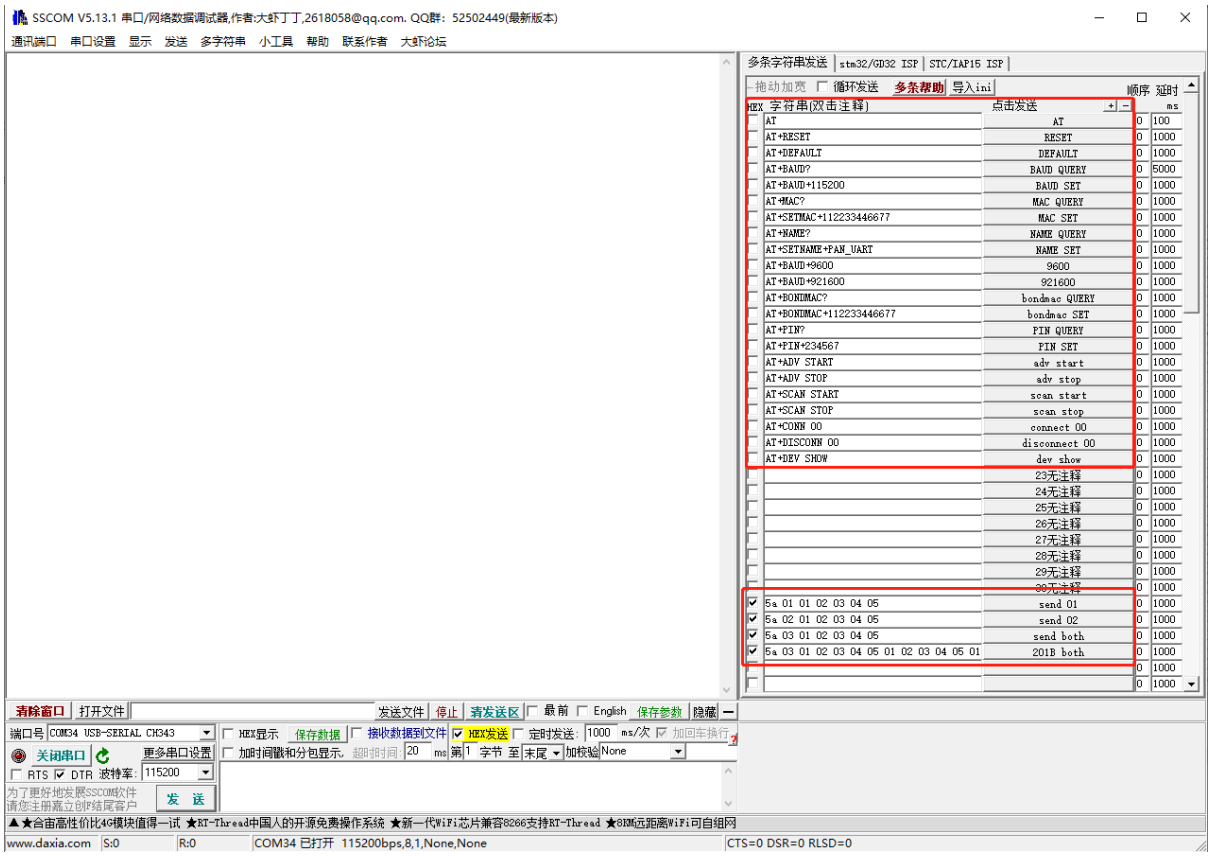


图 21: AT 指令测试窗口

4.2.2 蓝牙状态测试

主机和从机可以同时支持，默认上电开启了广播，最多支持 1 主 1 从

准备 2 块板子 A 和 B，分别烧录程序后，A 板子只作为从机，可以通过 AT 指令修改设备 mac 地址和名字，防止手机扫描到两个设备信息完全一样

然后对 B 板子以以下流程进行测试

4.2.2.1 从机模式

作为从机默认上电开启了蓝牙广播，测试流程可以按照以下顺序

1. 手机端蓝牙连接设备
2. 手机端对属性进行 `notify enable`
3. 芯片端发送消息上报
4. 手机端发送消息下发

4.2.2.2 主机模式

作为主机，需要主动开始扫描，扫描到设备后，主动进行连接已经开启的另一块从机设备，连接从机时默认会进行服务发现、`notify enable`，之后可以进行消息透传测试

1. AT+SCAN START，成功连接后返回 SCAN DONE
2. 芯片主机端进行数据传输测试
3. 可以进行第二块从机的准备和完成扫描连接流程
4. 可以对主机从机同时进行传输

5 RAM/Flash **资源使用情况** Spark Controller:

```
FLASH:      0x1c020 = 114720 B
SRAM:       0x9de0  = 40416 B
```

Solution: BLE Panchip-CTE Beacon

1 **功能概述** 此项目演示磐启蓝牙定位标签的功能，通过发送特定的广播数据，实现蓝牙定位功能。这是磐启蓝牙定位方案中的一部分，有关定位方案的更多信息请参考 **[待补充]**。

2 **环境要求**

- board: pan107
- uart (option): 显示串口 log

3 **编译和烧录** 例程位置: <home>\nimble\samples\solutions\ble_panchip_cte_beacon\keil

使用 keil 进行打开项目选择需要的 controller 进行烧录 ble_panchip_cte_beacon_origin 或者 ble_panchip_cte_beacon_spark 进行编译烧录，推荐选择 ble_panchip_cte_beacon_spark

4 **演示说明** 烧录完成后，设备自动启动蓝牙广播，可以在手机 nRF Connect 或抓包工具上获取如下信息:

- Advertising Type: ADV_SCAN_IND
- Advertising Interval Time: 250ms
- Company ID: (Shanghai Panchip Microelectronics Co., Ltd (0x07D1)
- Device Name: PANCHIP-CTE Beacon

下图是 nRF Connect(Android) 扫描到设备后显示的信息。

5 **广播数据** 广播数据包含两个 AD Element，如下表。

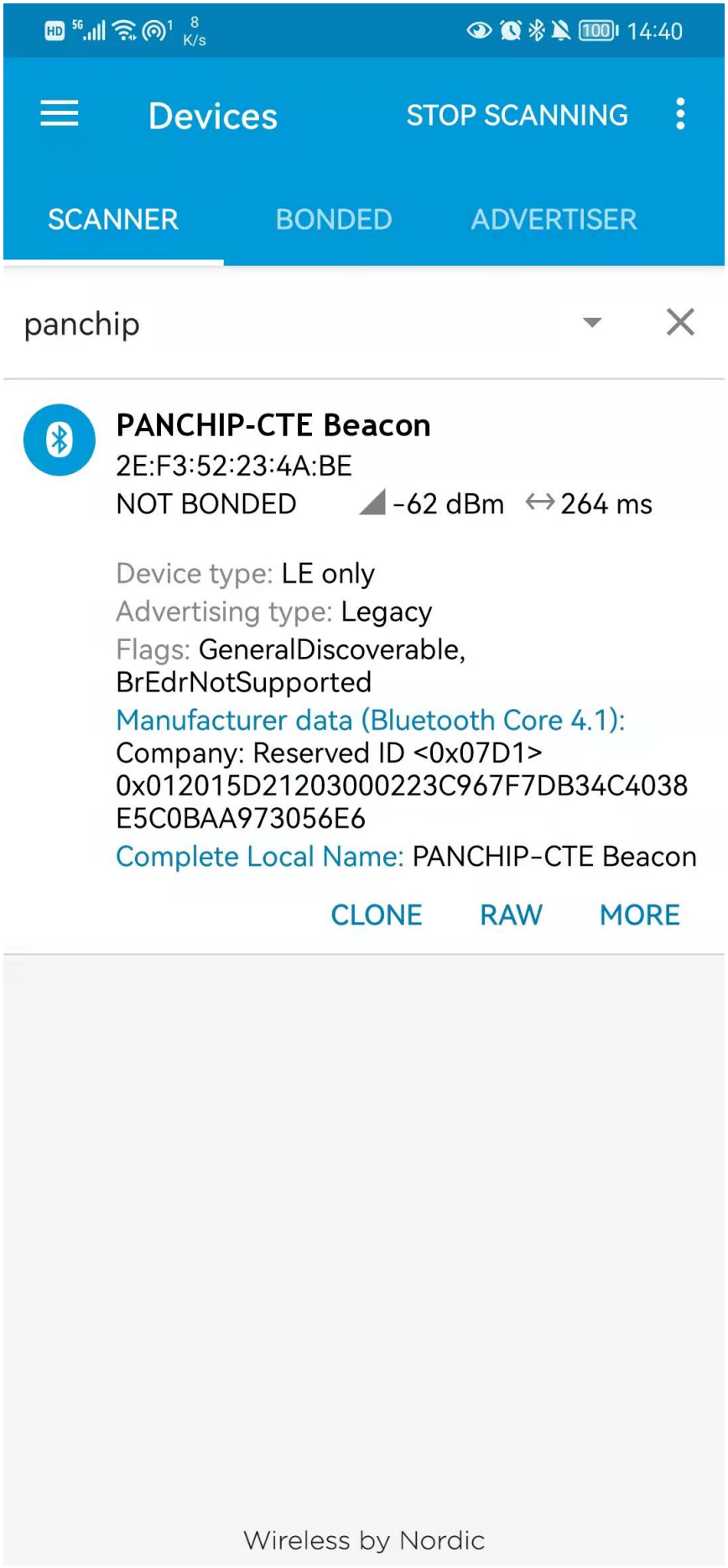


图 22: PANCHIP-CTE Beacon 手机端信息显示界面

Index (Byte)	Data	Name	Description
0	0x02	Length	Length of this AD Element 1
1	0x01	AD Type	Flags
2	0x06	Data	BT_LE_AD_GENERAL, BT_LE_AD_GENERAL
3	0x1B	Length	Length of this AD Element 2
4	0xFF	AD Type	Manufacturer Specific Data
5:6	0x07D1	Company ID	Shanghai Panchip Microelectronics Co., Ltd 厂商 ID 可由用户自定义用于区分设备厂家，标签和基站需要保持一致。
7	0x01	Packet ID	定位包 ID，用于区分同一厂家的不同设备，如标签、手环、IOS 微信小程序和安卓微信小程序等，标签使用 0x01。该部分可由用户自定义，标签和基站需保持一致。
8	0x20	Device Type	设备类型
9	0x15	Header	Carries information of Tag’ s TX rate, TX power and ID type
10:16	0xD2, 0x12, 0x03, 0x00, 0x02, 0x23,	Tag ID	用于区分不同的标签
17	0xC9	Checksum	CRC-8 [Device Type, Header, Tag ID]
18:31	0x67, 0xF7, 0xDB, 0x34, 0xC4, 0x03, 0x8E, 0x5C, 0x0B, 0xAA, 0x97, 0x30, 0x56, 0xE6	DF Field	该字段为辅助定位使用的固定字节。该段内容需保证空中抓取到的是固定频率的电磁波。根据 2402MHz 广播通道的白化算法规则和蓝牙先发送低字节的低比特的特性。修改信道时，需要对此进行调整。

6 RAM/Flash 资源使用情况

```
Total RO  Size (Code + RO Data)                 132644 ( 129.54kB)
Total RW  Size (RW Data + ZI Data)               22532 (  22.00kB)
Total ROM Size (Code + RO Data + RW Data)       132928 ( 129.81kB)
```

Solution: BLE RGB Light

1 功能概述 本文主要介绍 PAN107x BLE RGB 灯和手机 APP 进行连接，通过 APP 控制 RGB 灯的亮度与颜色。

2 环境要求

- board: `pan107`
- uart (option): 显示串口 log
- 安卓亿觅精灵灯 app V1.5.5，或微信小程序 (**待补充**)

3 编译和烧录 例程位置: `<home>\nimble\samples\solutions\ble_rgb_light\keil`

使用 `keil` 进行打开项目选择需要的 controller 进行烧录 `ble_rgb_light_origin` 或者 `ble_rgb_light_spark` 进行编译烧录，推荐选择 `ble_rgb_light_spark`

4 演示说明

1. PAN107 EVB 板 GPIO P11、P12、P14 与 RGB 电路用跳线帽连接。
2. EVB 板上电灯的颜色默认是蓝色，BLE 广播设备的名字是” b+EMIE Elfy”。
3. 打开安卓手机” 亿觅精灵灯 “app，在 app 上启动搜索设备。
4. 搜索到后点击连接，连接成功后就可以控制灯的开关和颜色了。

5 设备连接和控制

5.1 广播数据

Adv Data Type	Description	Length	Detail
0xff	Device id	6byte	0x11, 0x00, 0xc9, 0x7a, 0xbb, 0x8f, 0xdd, 0x4b, 0x00, 0x11
0x07	128-bit UUID	16byte	0x9e, 0xca, 0xdc, 0x24, 0x0e, 0xe5, 0xa9, 0xe0,0x93, 0xf3, 0xa3, 0xb5, 0x01, 0x20, 0x40, 0x6e
0x09	Device name	n byte	“b+EMIE Elfy”

5.2 GATT 服务

Function	Service Attribute	UUID(128bit)
Useless	Primary service	0x9e, 0xca, 0xdc, 0x24, 0x0e, 0xe5, 0xa9, 0xe0,0x93, 0xf3, 0xa3, 0xb5, 0x01, 0x20, 0x40, 0x6e
控制灯的状态	Write characteristic declaration	0x9e, 0xca, 0xdc, 0x24, 0x0e, 0xe5, 0xa9, 0xe0, 0x93, 0xf3, 0xa3, 0xb5, 0x02, 0x20, 0x40, 0x6e
Notify 灯的状态	notify characteristic declaration	0x9e, 0xca, 0xdc, 0x24, 0x0e, 0xe5, 0xa9, 0xe0,0x93, 0xf3, 0xa3, 0xb5, 0x03, 0x20, 0x40, 0x6e

5.3 通信协议

5.3.1 Light Control

UUID = {0x9e, 0xca, 0xdc, 0x24, 0x0e, 0xe5, 0xa9, 0xe0, 0x93, 0xf3, 0xa3, 0xb5, 0x03, 0x20, 0x40, 0x6e}

Function	Length	Detail
off	2byte	off：0xaa, 0x03
Color	5byte	0xaa,0x16,red：0~255,green：0~255,blue：0~255

控制灯的开关、颜色。

5.3.2 Notify Light Status

UUID = {0x9e, 0xca, 0xdc, 0x24, 0x0e, 0xe5, 0xa9, 0xe0, 0x93, 0xf3, 0xa3, 0xb5, 0x02, 0x20, 0x40, 0x6e}

每次收到控制命令后将灯的状态通知给手机 app。

6 RAM/Flash 资源使用情况

```
Total RO  Size (Code + RO Data)                 130900 ( 127.83kB)
Total RW  Size (RW Data + ZI Data)               22852 (  22.32kB)
Total ROM Size (Code + RO Data + RW Data)       131220 ( 128.14kB)
```

BLE Vehicles Key

1 功能概述 此项目演示基于 HID 服务的自动连接服务，通过 RSSI 值的大小模拟实现电动二轮车的利用距离自动开关功能。

2 环境要求

- board: `pan107x evb`
- uart(option): 用来显示串口 log（波特率 921600，选项 `8n1`）s_key

3 编译和烧录 例程位置：`<home>\nimble\samples\solutions\ble_vehicles_key\keil`

使用 `keil` 进行打开项目选择需要的 controller 进行烧录 `ble_vehicles_key` 或者 `ble_vehicles_key_spark` 进行编译烧录。

4 演示说明

1. 在设置蓝牙界面连接 `vehicles key` 进行配对后会自动跟踪 rssi 值的大小，模拟电动二轮车钥匙实现近距离自动开关。

```
[19:57:21.854]  Try to load HW calibration data.. DONE.
- Chip Type         : 0x80
- Chip CP Version   : None
- Chip FT Version   : 8
- Chip MAC Address  : D0000C0CBBF5
- Chip Flash UID    : 32313334320EAC834330FFFFFFFFFFFF
- Chip Flash Size   : 1024 KB
LL Spark Controller Version:b0e99c4

[19:57:21.919]  ble_store_config_num_our_secs:0
ble_store_config_num_peer_secs:0
ble_store_config_num_cccds:0
registered service 0x1812 with handle=1
registering characteristic 0x2a4a with def_handle=2 val_handle=3
registering characteristic 0x2a4b with def_handle=4 val_handle=5
registering characteristic 0x2a4d with def_handle=6 val_handle=7
registering descriptor 0x2908 with handle=9
registering characteristic 0x2a4d with def_handle=10 val_handle=11
registering descriptor 0x2908 with handle=13
registering characteristic 0x2a4c with def_handle=14 val_handle=15
Device Address: 01 02 03 04 05 06


[19:57:28.164]  connection established; status=0 handle=1 our_ota_addr_type=0 our_ota_
↪addr=01 02 03 04 05 06
 our_id_addr_type=0 our_id_addr=01 02 03 04 05 06
 peer_ota_addr_type=1 peer_ota_addr=f4 be 2e 4e 35 50
 peer_id_addr_type=1 peer_id_addr=f4 be 2e 4e 35 50
 conn_itvl=24 conn_latency=0 supervision_timeout=500 encrypted=0 authenticated=0 bonded=0


[19:57:30.923]  encryption change event; status=0 handle=1 our_ota_addr_type=0 our_ota_
↪addr=01 02 03 04 05 06
 our_id_addr_type=0 our_id_addr=01 02 03 04 05 06
 peer_ota_addr_type=1 peer_ota_addr=f4 be 2e 4e 35 50
 peer_id_addr_type=1 peer_id_addr=f4 be 2e 4e 35 50
 conn_itvl=24 conn_latency=0 supervision_timeout=500 encrypted=1 authenticated=0 bonded=1

ediv=0 rand=0 authenticated=0 ltk= f5f99f23fb223b4ff0f5f7f21c0146d1 irk=␣
↪00000000000000000000000000000000
```

(下页继续)

(续上页)

```
ediv=0 rand=0 authenticated=0 ltk= f5f99f23fb223b4ff0f5f7f21c0146d1 irk=␣
↪4a3711a0c1b7cd2c92d64048e813fe34

[19:57:31.223]  connection updated; status=0 handle=1 our_ota_addr_type=0 our_ota_addr=01␣
↪02 03 04 05 06
 our_id_addr_type=0 our_id_addr=01 02 03 04 05 06
 peer_ota_addr_type=1 peer_ota_addr=f4 be 2e 4e 35 50
 peer_id_addr_type=0 peer_id_addr=07 49 34 4d af 50
 conn_itvl=6 conn_latency=0 supervision_timeout=500 encrypted=1 authenticated=0 bonded=1

connection updated; status=0 handle=1 our_ota_addr_type=0 our_ota_addr=01 02 03 04 05 06
 our_id_addr_type=0 our_id_addr=01 02 03 04 05 06
 peer_ota_addr_type=1 peer_ota_addr=f4 be 2e 4e 35 50
 peer_id_addr_type=0 peer_id_addr=07 49 34 4d af 50
 conn_itvl=24 conn_latency=0 supervision_timeout=500 encrypted=1 authenticated=0 bonded=1

[19:57:32.135]
subscribe event; conn_handle=1 attr_handle=7 reason=1 prevn=0 curn=1 previ=0 curi=0

[19:57:32.189]
subscribe event; conn_handle=1 attr_handle=11 reason=1 prevn=0 curn=1 previ=0 curi=0

[19:57:32.238]  open led
```

2. 如果使用 EVB 板的话，请将 RGB-G 的跳线帽连接，如果 RSSI 距离达到接近阈值时会触发开的动作，足够远时会触发关的距离动作。
3. 关于 rssi 的滤波算法使用的是去除最大和最小值各 RSSI_REMOVE_BORDER_NUM 个，默认 RSSI_REMOVE_BORDER_NUM 为 2 个，排序使用冒泡排序。
4. RSSI 的整体处理在 hog.c 中。

5 RAM/Flash **资源使用情况** Origin Controller:

```
FLASH:      155759 B
SRAM:       38248 B
```

Spark Controller:

```
FLASH:      129555 B
SRAM:       40280 B
```

Solution: Electronic Shelf Label

1 **功能概述** 此 sample 为 pan107x(40pin 芯片) 在电子价签板下的应用。

具体支持的 feature 如下：

1. 外挂 spi flash：外挂 flash 存储价签图案数据，每隔 45s 通过 dma 方式从 flash 读取一个图案
2. epd 墨水屏：外挂 flash 读取的数据通过 3 线 spi 传输给墨水屏，启动墨水屏刷屏
3. 低功耗模式：刷图完成后芯片及 epd 均进入休眠模式，模块进入超低功耗模式（standby），定时 15s 唤醒
4. RF 发送：芯片唤醒进入 rf 发送流程
5. 每 3 次发送完成后，启动刷屏流程，1~4 步骤重复

2 环境要求

- board: ‘pan107x 40pin esl 价签板
- 外挂 flash、墨水屏
- 电流监测工具 nrf ppk

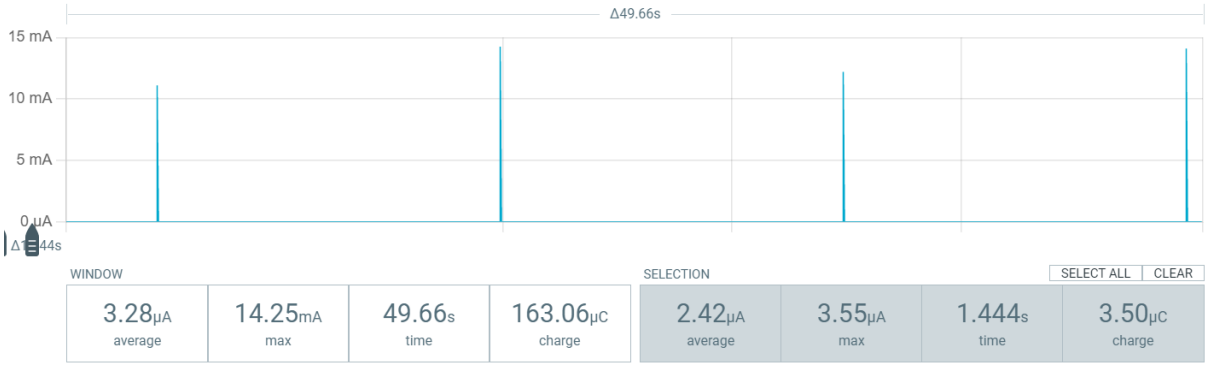
3 编译和烧录

例程位置：`nimble\pan107x_samples\solutions\esl`

使用 keil 工具可以对其进行编译、烧录、调试等操作。

4 演示说明

1. 准备 esl 价签板，107/FM/EPD 跳线帽短接
2. 插入 epd2266 墨水屏（SE2266JS0C5）
3. 打开 PPK 并使用其供电 3.3v
4. 观测 PPK 电流变化及墨水屏刷屏过程（45s 刷屏一次），电流每 15s 进入低功耗



5 主要数据结构说明

配置的结构体“pan_prf_config_t”，各成员介绍如下：

Type	name	Description
prf_mode_t	work_mode	工作模式配置，包括普通型和增强型
prf_chip_mode_sel_t	chip_mode	xn297 通信协议和 nordic 通信协议配置
prf_trx_mode_t	trx_mode	收发模式配置
prf_phy_t	phy	通信速率配置，可配置为 1M 和 2M
prf_crc_sel_t	crc	数据包 CRC 配置，可配置为 crc 16bit，crc 8bit，crc 24bit ，no crc
prf_scramble_sel_t	scr	数据包扰码的配置，可配置为使用扰码和不使用扰码
uint16_t	rx_timeout	接收超时时间配置，最大 50000us
uint16_t	rf_channel	2.4g 频点配置，任意频点可设 (2402Mhz~2480Mhz)
uint8_t	tx_no_ack	配置增强型模式下 tx 是否需要 ack
prf_trf_t	trf_type	nordic 的长包模式配置，最大 payload 的长度为 255
uint8_t	rx_length	rx 接收数据包长度配置，增强型模式下可不配置
uint8_t	sync_length	接入地址长度配置，可配置为 3、4、5 字节
uint8_t	sync[5]	接入地址的内容 (xn297 模式下可白化地址，防止出现长 0 和长 1 的地址)
prf_dev_sel_t	dev	设置 deviation，可以选择 BLE 模式（1M 250k；2M 500k），NRF 模式（1M 160K；2M 320）
int8_t	tx_power	设置发射功率，范围 (-45dbm~7dbm)
uint8_t	pid_manual_flag	pid 手动配置的标志，使能后可以自定义 pid
uint8_t	crc_include_sync	crc 计算包含地址
uint8_t	src_include_sync	白化包含地址
uint16_t	tx_trans_time	发送传输时间设置
uint16_t	rx_trans_time	接收传输时间设置
prf_pipe_t	pipe	管道配置，可配置为 0~7

prf_mode_t:

Type	Value	Description
PRF_MODE_NORMAL	0	普通型
PRF_MODE_ENHANCE	1	增强型
PRF_MODE_NORMAL_M1	2	普通型 M1 模式

prf_chip_mode_sel_t:

Type	Value	Description
PRF_CHIP_MODE_SEL_BLE	1	蓝牙模式
PRF_CHIP_MODE_SEL_XN297	2	XN297 模式
PRF_CHIP_MODE_SEL_NORDIC	3	NORCDIC 模式

prf_trx_mode_t:

Type	Value	Description
PRF_TX_MODE	0	2.4G 发射
PRF_RX_MODE	1	2.4G 接收

prf_phy_t:

Type	Value	Description
PRF_PHY_1M	1	1M 通信速率
PRF_PHY_2M	2	2M 通信速率
PRF_PHY_CODED_S8	3	S8 模式
PRF_PHY_CODED	4	S2 模式
PRF_PHY_250K	5	250K 模式

prf_crc_sel_t:

Type	Value	Description
PRF_CRC_SEL_NOCRC	0	no crc
PRF_CRC_SEL_CRC8	1	crc 8bit
PRF_CRC_SEL_CRC16	2	crc 16bit
PRF_CRC_SEL_CRC24	3	crc 24bit

prf_scramble_sel_t:

Type	Value	Description
PRF_SRC_SEL_NOSRC	0	不使能扰码
PRF_SRC_SEL_EN	1	使能扰码

prf_dev_sel_t:

Type	Value	Description
PRF_DEV_NRF	1	NRF 模式 deviation 配置，1M 170k，2M 340K
PRF_DEV_BLE	2	NRF 模式 deviation 配置，1M 250k，2M 500K

prf_addr_length_sel_t:

Type	Value	Description
PRF_ADDR_LENGTH_SEL_3	3	3 BYTE 地址长度
PRF_ADDR_LENGTH_SEL_4	4	4 BYTE 地址长度
PRF_ADDR_LENGTH_SEL_5	5	5 BYTE 地址长度

prf_pipe_t:

Type	Value	Description
PRF_PIPE0	1«0	管道 0
PRF_PIPE1	1«1	管道 1
PRF_PIPE2	1«2	管道 2
PRF_PIPE3	1«3	管道 3
PRF_PIPE4	1«4	管道 4
PRF_PIPE5	1«5	管道 5
PRF_PIPE6	1«6	管道 6
PRF_PIPE7	1«7	管道 7

prf_trf_t:

Type	Value	Description
PRF_TRF_NORMAL	0	普通模式传输
PRF_TRF_NRF52	1	NRF 模式传输
PRF_TRF_BOE	2	250k 模式传输

6 补充说明　补充说明当前功耗测试情况，支持中遇到的问题（供参考）及已知仍可能存在的问题

6.1 功耗说明　蓝牙例程

源码路径: `<PAN1070-NDK>\01_SDK\nimble\samples\bluetooth`

例程	说明
Bluetooth: Central and Peripheral	演示蓝牙主从一体功能
Bluetooth: Central	演示蓝牙主机功能，发现设备并与设备建立连接和断连
Bluetooth: BLE Peripheral ENC	演示外设以及加密配对功能，可以和主机示例进行对测
Bluetooth: Peripheral HR	演示蓝牙从机功能，包含 GATT 服务：HR (Heart Rate)，连接订阅服务后，会上报虚拟的心率值

解决方案

源码路径: `<PAN1070-NDK>\01_SDK\nimble\samples\solutions`

例程	说明
Solution: BLE HID Selfie	自拍解决方案，通过蓝牙 HID 控制手机拍照
Solution: BLE HID Uart Mult Roles	蓝牙串口透传解决方案，演示蓝牙 hid 串口透传功能，支持 1 主 1 从
Solution: BLE Panchip-CTE Beacon	Panchip 蓝牙定位标签方案，通过发送特定的广播数据，实现蓝牙定位功能
Solution: BLE RGB Light	蓝牙 RGB 灯控方案，演示 BLE RGB 灯与手机 APP 进行连接，通过 APP 控制 RGB 灯的亮度与颜色
Solution: BLE Vehicles Key	蓝牙车钥匙解决方案，演示基于 HID 服务的自动连接服务
Solution: Electronic Shelf Label	电子货架标签方案演示例程，支持外部 SPI Flash 存储、EPD 墨水屏、低功耗模式、RF 通信等功能

2.3.3 MCU Keil 例程

例程源码路径: `<PAN1070-NDK>\03_MCU\mcu_samples`

MCU 底层驱动（Low Level Driver）Keil 例程：

例程	说明
MCU Low Level ADC Driver Sample	MCU 底层 ADC 驱动例程演示说明
MCU Low Level CLKTRIM Driver Sample	MCU 底层 Clock Trim 驱动例程演示说明
MCU Low Level CLK Driver Sample	MCU 底层 CLK 驱动例程演示说明
MCU Low Level DMA Driver Sample	MCU 底层 DMA 驱动例程演示说明
MCU Low Level eFuse Driver Sample	MCU 底层 eFuse 驱动例程演示说明
MCU Low Level FMC Driver Sample	MCU 底层 FMC 驱动例程演示说明
MCU Low Level GPIO Driver Sample	MCU 底层 GPIO 驱动例程演示说明
MCU Low Level I2C Driver Sample	MCU 底层 I2C 驱动例程演示说明
MCU Low Level PWM Sample	MCU 底层 PWM 驱动例程演示说明
MCU Low Level SPI Sample	MCU 底层 SPI 驱动例程演示说明
MCU Low Level TIMER Sample	MCU 底层 TIMER 驱动例程演示说明
MCU Low Level UART Sample	MCU 底层 UART 驱动例程演示说明
MCU Low Level WDT Sample	MCU 底层 WDT 驱动例程演示说明
MCU Low Level WWDT Sample	MCU 底层 WWDT 驱动例程演示说明
MCU DebugProtect Sample	MCU Debug Protect 调试接口保护例程演示说明
MCU PRF TRX Sample	MCU 私有 2.4G 通信开发指南

2.4 NDK 开发指南

2.4.1 NDK App 开发指南

本文主要通过一些示例，介绍**蓝牙应用开发**过程中常用的方法以及可能遇到的问题。

1 基础指标

1.1 **功耗** 蓝牙在不同的工作模式下功耗如下表所示：

测试条件：

- 基于例程 `nimble\samples\bleprph_hr`

测试配置:`CONFIG_SOC_DCDC_PAN1070,CONFIG_PM_ENABLE,CONFIG_LOW_SPEED_CLOCK_SRC` 测试选项：`LOW_POWER_TESET_CI_100MS` 和 `LOW_POWER_TESET_CI_1000MS`

工作模式	tx power (dbm)	模式	休眠时钟	峰值电流 (mA)	休眠电流(uA)	平均电流 (uA, 100ms)	平均电流 (uA, 1000ms)
蓝牙广播	0	DCDC	XTL	19.15	3.97	/	13.91
			RCL	21.42	3.99	/	14.52
		LDO	XTL	19.61	3.94	/	21.54
			RCL	19.61	3.97	/	23.17
蓝牙连接		DCDC	XTL	19.6	3.89	/	10.61
			RCL	19.6	3.79	/	21.29
		LDO	XTL	19.61	3.84	/	14.34
			RCL	19.61	3.79	/	25.87
蓝牙广播	7	DCDC	XTL	19.6	3.99	/	18.75
			RCL	19.35	3.95	/	18.99
		LDO	XTL	27.52	3.96	/	31.18
			RCL	27.52	4.09	/	33.77
蓝牙连接		DCDC	XTL	19.6	4.04	/	11.26
			RCL	20.38	3.95	/	20.36
		LDO	XTL	27.37	3.78	/	15.75
			RCL	26.97	3.97	/	27.47

图 23: PAN1070 EVB 核心板功耗测试数据

工作模式	tx power (dbm)	模式	休眠时钟	峰值电流 (mA)	休眠电流(uA)	平均电流 (uA, 100ms)	平均电流 (uA, 1000ms)
蓝牙广播	0	DCDC	XTL	19.6	4.13	/	14.24
			RCL	19.84	4.01	/	14.61
		LDO	XTL	19.61	4.1	/	21.79
			RCL	19.61	4.02	/	23.3
蓝牙连接		DCDC	XTL	19.31	3.98	/	11.94
			RCL	19.6	4.01	/	25.99
		LDO	XTL	19.61	4.03	/	16.04
			RCL	19.61	4.03	/	30.32

图 24: PAN1070 EVB 核心板配置 latency 功耗测试数据

2 开发流程

2.1 确认开发环境 参考 NDK 快速入门指南，确认软硬件开发环境，可以正常的编译、下载和调试 SDK 提供的基础例程。

建议连接板载的 micro USB，通过串口工具监测 Log。

2.2 参考相关例程 蓝牙开发需要了解一些蓝牙协议相关的知识，可以参考蓝牙协议规范，网上也有很多协议的介绍，此处不作为重点。

当前 SDK 中提供了一些蓝牙相关的例程，涵盖了 central、peripheral 等。

在进行蓝牙开发之前，建议先看一下相关的文档，磨刀不误砍柴工，相信这些例程会对你的开发有所帮助。

2.3 了解蓝牙 app 代码的基本框架

2.3.1 app 和 host 初始化 我们以 `bleprph_hr` 为例，app 和 host 的初始化默认都是在 `app_main` 函数中：

```
void app_main(void)
{
    int rc;

    printf("app started\n");

    /** set public address*/
    uint8_t pub_mac[6]={8,2,3,4,5,6};
    db_set_bd_address(pub_mac);

    /* Initialize the NimBLE host configuration */
    ble_hs_cfg.sync_cb = blehr_on_sync;

    ble_npl_callout_init(&blehr_tx_timer, (struct ble_npl_eventq *)nimble_port_get_dflt_
↪eventq(),
                         blehr_tx_hrate, NULL);

    rc = gatt_svr_init();
    assert(rc == 0);

    /* Set the default device name */
    rc = ble_svc_gap_device_name_set(device_name);
    assert(rc == 0);

        hs_thread_init();
}
```

从这个初始化我们可以将真正需要的初始化切割出来：

- `ble_hs_cfg.sync_cb = blehr_on_sync;` 这个是 host 初始化完成后的回调函数，一般是将需要的**自定义广播函数**的回调添加此处。
- `gatt_svr_init()` GATT **服务**初始化。
- `hs_thread_init` host **协议栈**的初始化。
- `blehr_gap_event` 蓝牙状态事件的处理，比如广播，连接，断连等，可以关注下 `blehr_gap_event` 是在广播启动函数中 `ble_gap_adv_start` 注册的。注意：对于**主机**是在扫描启动函数中 `ble_gap_disc` 注册的。

其实一般来说，一个蓝牙工程有广播，GATT 服务，蓝牙事件处理，基本就搭起一个蓝牙应用的框架了，我们再由此进行展开。

同时 `ble_hs_cfg` 是一个全局变量，很多关键回调函数和状态依赖它：

```
/** @brief Bluetooth Host main configuration structure
 *
 * Those can be used by application to configure stack.
 *
 * The only reason Security Manager (sm_ members) is configurable at runtime is
 * to simplify security testing. Defaults for those are configured by selecting
 * proper options in application's syscfg.
 */
struct ble_hs_cfg {
    /**
     * An optional callback that gets executed upon registration of each GATT
     * resource (service, characteristic, or descriptor).
     */
    ble_gatt_register_fn *gatts_register_cb;

    /**
     * An optional argument that gets passed to the GATT registration
     * callback.
     */
    void *gatts_register_arg;

    /** Security Manager Local Input Output Capabilities */
    uint8_t sm_io_cap;

    /** @brief Security Manager OOB flag
     *
     * If set proper flag in Pairing Request/Response will be set.
     */
    unsigned sm_oob_data_flag:1;

    /** @brief Security Manager Bond flag
     *
     * If set proper flag in Pairing Request/Response will be set. This results
     * in storing keys distributed during bonding.
     */
    unsigned sm_bonding:1;

    /** @brief Security Manager MITM flag
     *
     * If set proper flag in Pairing Request/Response will be set. This results
     * in requiring Man-In-The-Middle protection when pairing.
     */
    unsigned sm_mitm:1;

    /** @brief Security Manager Secure Connections flag
     *
```

(下页继续)

(续上页)

```
     * If set proper flag in Pairing Request/Response will be set. This results
     * in using LE Secure Connections for pairing if also supported by remote
     * device. Fallback to legacy pairing if not supported by remote.
     */
    unsigned sm_sc:1;

    /** @brief Security Manager Key Press Notification flag
     *
     * Currently unsupported and should not be set.
     */
    unsigned sm_keypress:1;

    /** @brief Security Manager Local Key Distribution Mask */
    uint8_t sm_our_key_dist;

    /** @brief Security Manager Remote Key Distribution Mask */
    uint8_t sm_their_key_dist;

    /** @brief Stack reset callback
     *
     * This callback is executed when the host resets itself and the controller
     * due to fatal error.
     */
    ble_hs_reset_fn *reset_cb;

    /** @brief Stack sync callback
     *
     * This callback is executed when the host and controller become synced.
     * This happens at startup and after a reset.
     */
    ble_hs_sync_fn *sync_cb;

    /* XXX: These need to go away. Instead, the nimble host package should
     * require the host-store API (not yet implemented)..
     */
    /** Storage Read callback handles read of security material */
    ble_store_read_fn *store_read_cb;

    /** Storage Write callback handles write of security material */
    ble_store_write_fn *store_write_cb;

    /** Storage Delete callback handles deletion of security material */
    ble_store_delete_fn *store_delete_cb;

    /** @brief Storage Status callback.
     *
     * This callback gets executed when a persistence operation cannot be
     * performed or a persistence failure is imminent. For example, if is
     * insufficient storage capacity for a record to be persisted, this
     * function gets called to give the application the opportunity to make
     * room.
     */
    ble_store_status_fn *store_status_cb;

    /** An optional argument that gets passed to the storage status callback. */
    void *store_status_arg;
};
```

2.3.2 蓝牙广播或者扫描

广播函数:

```
static void
blehr_advertise(void)
{
    struct ble_gap_adv_params adv_params;
    struct ble_hs_adv_fields fields;
    int rc;

    /*
     *  Set the advertisement data included in our advertisements:
     *     o Flags (indicates advertisement type and other general info)
     *     o Advertising tx power
     *     o Device name
     */
    memset(&fields, 0, sizeof(fields));

    /*
     * Advertise two flags:
     *      o Discoverability in forthcoming advertisement (general)
     *      o BLE-only (BR/EDR unsupported)
     */
    fields.flags = BLE_HS_ADV_F_DISC_GEN |
                   BLE_HS_ADV_F_BREDR_UNSUP;


    fields.name = (uint8_t *)device_name;
    fields.name_len = strlen(device_name);
    fields.name_is_complete = 1;

    rc = ble_gap_adv_set_fields(&fields);
    if (rc != 0) {

        printf("error setting advertisement data; rc=%d\n", rc);
        return;
    }

    /* Begin advertising */
    memset(&adv_params, 0, sizeof(adv_params));
    adv_params.conn_mode = BLE_GAP_CONN_MODE_UND;
    adv_params.disc_mode = BLE_GAP_DISC_MODE_GEN;

    #if LOW_POWER_TESET_CI_100MS || LOW_POWER_TESET_LATENCY_100MS
    adv_params.itvl_min = BLE_GAP_ADV_ITVL_MS(100);
    adv_params.itvl_max = BLE_GAP_ADV_ITVL_MS(100);
    #endif

    #if LOW_POWER_TESET_CI_1000MS || LOW_POWER_TESET_LATENCY_1000MS
    adv_params.itvl_min = BLE_GAP_ADV_ITVL_MS(1000);
    adv_params.itvl_max = BLE_GAP_ADV_ITVL_MS(1000);
    #endif

    rc = ble_gap_adv_start(blehr_addr_type, NULL, BLE_HS_FOREVER,
                           &adv_params, blehr_gap_event, NULL);
    if (rc != 0) {
        printf("error enabling advertisement; rc=%d\n", rc);
        return;
    }
}
```

扫描函数:

(下页继续)

(续上页)

```
/**
 * Initiates the GAP general discovery procedure.
 */
static void
blecent_scan(void)
{
    uint8_t own_addr_type;
    struct ble_gap_disc_params disc_params;
    int rc;

    /* Figure out address to use while advertising (no privacy for now) */
    rc = ble_hs_id_infer_auto(0, &own_addr_type);
    if (rc != 0) {
        printf("error determining address type; rc=%d\n", rc);
        return;
    }

    /* Tell the controller to filter duplicates; we don't want to process
     * repeated advertisements from the same device.
     */
    disc_params.filter_duplicates = 0;

    /**
     * Perform a passive scan.  I.e., don't send follow-up scan requests to
     * each advertiser.
     */
    disc_params.passive = 1;

    /* Use defaults for the rest of the parameters. */
    disc_params.itvl = 60;
    disc_params.window = 50;
    disc_params.filter_policy = 0;
    disc_params.limited = 0;

    rc = ble_gap_disc(own_addr_type, BLE_HS_FOREVER, &disc_params,
                      blecent_gap_event, NULL);
    if (rc != 0) {
        printf("Error initiating GAP discovery procedure; rc=%d\n",
                    rc);
    }
}
```

在初始化状态这两个函数最终会注册到 `ble_hs_cfg.sync_cb`。

2.3.3 GATT 服务初始化

对于 GATT 服务初始化，我们看以下一段代码：

```
int gatt_svr_init(void)
{
    int rc;

    rc = ble_gatts_count_cfg(gatt_svr_svcs);
    if (rc != 0) {
        return rc;
    }

    rc = ble_gatts_add_svcs(gatt_svr_svcs);
    if (rc != 0) {
        return rc;
    }
```

(下页继续)

(续上页)

```
    return 0;
}
```

ble_gatts_count_cfg 获取 GATT config 描述个数和 ble_gatts_add_svcs``注册 GATT 服务这两个函数都会涉及到 gatt_svr_svcs 这个变量，跳转过去我们发现 gatt_svr_svcs 真正的定义 GATT 服务的数据库，我们可以在这个变量中自定义实现 GATT 服务。

```
static const struct ble_gatt_svc_def gatt_svr_svcs[] = {
    {
        /* Service: Heart-rate */
        .type = BLE_GATT_SVC_TYPE_PRIMARY,
        .uuid = BLE_UUID16_DECLARE(GATT_HRS_UUID),
        .characteristics = (struct ble_gatt_chr_def[]) { {
            /* Characteristic: Heart-rate measurement */
            .uuid = BLE_UUID16_DECLARE(GATT_HRS_MEASUREMENT_UUID),/* 声明蓝牙 GATT 服务 */
            .access_cb = gatt_svr_chr_access_heart_rate,
            .val_handle = &hrs_hrm_handle,
            .flags = BLE_GATT_CHR_F_NOTIFY,
        }, {
            /* Characteristic: Body sensor location */
            .uuid = BLE_UUID16_DECLARE(GATT_HRS_BODY_SENSOR_LOC_UUID),
            .access_cb = gatt_svr_chr_access_heart_rate,
            .flags = BLE_GATT_CHR_F_READ,
        }, {
            0, /* No more characteristics in this service */
        }, }
    },

    {
        /* Service: Device Information */
        .type = BLE_GATT_SVC_TYPE_PRIMARY,
        .uuid = BLE_UUID16_DECLARE(GATT_DEVICE_INFO_UUID),
        .characteristics = (struct ble_gatt_chr_def[]) { {
            /* Characteristic: * Manufacturer name */
            .uuid = BLE_UUID16_DECLARE(GATT_MANUFACTURER_NAME_UUID),
            .access_cb = gatt_svr_chr_access_device_info,
            .flags = BLE_GATT_CHR_F_READ,
        }, {
            /* Characteristic: Model number string */
            .uuid = BLE_UUID16_DECLARE(GATT_MODEL_NUMBER_UUID),
            .access_cb = gatt_svr_chr_access_device_info,
            .flags = BLE_GATT_CHR_F_READ,
        }, {
            0, /* No more characteristics in this service */
        }, }
    },

        {
            0, /* No more services */
        },
};
```

对于心跳服务访问 gatt_svr_chr_access_heart_rate 这个函数，实现相对简单。我们可以参考 bleprph_enc 例程中的 gatt_svc_access 函数,，它对**特性的读，写，读描述符，写描述符**等等做了不同的分类和实现：

```
static int
gatt_svc_access(uint16_t conn_handle, uint16_t attr_handle,
                struct ble_gatt_access_ctxt *ctxt, void *arg)
{
    const ble_uuid_t *uuid;
```

(下页继续)

(续上页)

```
    int rc;

    switch (ctxt->op) {
    case BLE_GATT_ACCESS_OP_READ_CHR:
        if (conn_handle != BLE_HS_CONN_HANDLE_NONE) {
            MODLOG_DFLT("Characteristic read; conn_handle=%d attr_handle=%d\n",
                        conn_handle, attr_handle);
        } else {
            MODLOG_DFLT("Characteristic read by NimBLE stack; attr_handle=%d\n",
                        attr_handle);
        }
        uuid = ctxt->chr->uuid;
        if (attr_handle == gatt_svr_chr_val_handle) {
            rc = os_mbuf_append(ctxt->om,
                                &gatt_svr_chr_val,
                                sizeof(gatt_svr_chr_val));
            return rc == 0 ? 0 : BLE_ATT_ERR_INSUFFICIENT_RES;
        }
        goto unknown;

    case BLE_GATT_ACCESS_OP_WRITE_CHR:
        if (conn_handle != BLE_HS_CONN_HANDLE_NONE) {
            MODLOG_DFLT("Characteristic write; conn_handle=%d attr_handle=%d",
                        conn_handle, attr_handle);
        } else {
            MODLOG_DFLT("Characteristic write by NimBLE stack; attr_handle=%d",
                        attr_handle);
        }
        uuid = ctxt->chr->uuid;
        if (attr_handle == gatt_svr_chr_val_handle) {
            rc = gatt_svr_write(ctxt->om,
                                sizeof(gatt_svr_chr_val),
                                sizeof(gatt_svr_chr_val),
                                &gatt_svr_chr_val, NULL);
            ble_gatts_chr_updated(attr_handle);
            MODLOG_DFLT("Notification/Indication scheduled for "
                        "all subscribed peers.\n");
            return rc;
        }
        goto unknown;

    case BLE_GATT_ACCESS_OP_READ_DSC:
        if (conn_handle != BLE_HS_CONN_HANDLE_NONE) {
            MODLOG_DFLT("Descriptor read; conn_handle=%d attr_handle=%d\n",
                        conn_handle, attr_handle);
        } else {
            MODLOG_DFLT("Descriptor read by NimBLE stack; attr_handle=%d\n",
                        attr_handle);
        }
        uuid = ctxt->dsc->uuid;
        if (ble_uuid_cmp(uuid, &gatt_svr_dsc_uuid.u) == 0) {
            rc = os_mbuf_append(ctxt->om,
                                &gatt_svr_dsc_val,
                                sizeof(gatt_svr_chr_val));
            return rc == 0 ? 0 : BLE_ATT_ERR_INSUFFICIENT_RES;
        }
        goto unknown;

    case BLE_GATT_ACCESS_OP_WRITE_DSC:
        goto unknown;
```

(下页继续)

(续上页)

```
    default:
        goto unknown;
    }

unknown:
    /* Unknown characteristic/descriptor;
     * The NimBLE host should not have called this function;
     */
    assert(0);
    return BLE_ATT_ERR_UNLIKELY;
}
```

我们可以参考这个函数对特性的读写做一些通用的实现。

2.3.4 GAP 事件处理　从上文我们可以知道，GAP 事件函数我们可以注册到广播或者扫描函数中，我们也可以对不同的事件进行分类处理，例如连接，断连，配对事件，订阅事件，扫描收到的广播包等等：

支持的事件如下：

```
#define BLE_GAP_EVENT_CONNECT               0
#define BLE_GAP_EVENT_DISCONNECT            1
/* Reserved                                 2 */
#define BLE_GAP_EVENT_CONN_UPDATE           3
#define BLE_GAP_EVENT_CONN_UPDATE_REQ       4
#define BLE_GAP_EVENT_L2CAP_UPDATE_REQ      5
#define BLE_GAP_EVENT_TERM_FAILURE          6
#define BLE_GAP_EVENT_DISC                  7
#define BLE_GAP_EVENT_DISC_COMPLETE         8
#define BLE_GAP_EVENT_ADV_COMPLETE          9
#define BLE_GAP_EVENT_ENC_CHANGE            10
#define BLE_GAP_EVENT_PASSKEY_ACTION        11
#define BLE_GAP_EVENT_NOTIFY_RX             12
#define BLE_GAP_EVENT_NOTIFY_TX             13
#define BLE_GAP_EVENT_SUBSCRIBE             14
#define BLE_GAP_EVENT_MTU                   15
#define BLE_GAP_EVENT_IDENTITY_RESOLVED     16
#define BLE_GAP_EVENT_REPEAT_PAIRING        17
#define BLE_GAP_EVENT_PHY_UPDATE_COMPLETE   18
#define BLE_GAP_EVENT_EXT_DISC              19
#define BLE_GAP_EVENT_PERIODIC_SYNC         20
#define BLE_GAP_EVENT_PERIODIC_REPORT       21
#define BLE_GAP_EVENT_PERIODIC_SYNC_LOST    22
#define BLE_GAP_EVENT_SCAN_REQ_RCVD         23
#define BLE_GAP_EVENT_PERIODIC_TRANSFER     24
#define BLE_GAP_EVENT_PATHLOSS_THRESHOLD    25
#define BLE_GAP_EVENT_TRANSMIT_POWER        26
```

```
static int blehr_gap_event(struct ble_gap_event *event, void *arg)
{
    switch (event->type) {
    case BLE_GAP_EVENT_CONNECT:
        /* A new connection was established or a connection attempt failed */
        printf("connection %s; status=%d\n",
                    event->connect.status == 0 ? "established" : "failed",
                    event->connect.status);

        if (event->connect.status != 0) {
```

(下页继续)

(续上页)

```
            /* Connection failed; resume advertising */
            blehr_advertise();
            conn_handle = 0;
        }
        else {
          conn_handle = event->connect.conn_handle;
          #if LOW_POWER_TESET_CI_100MS || LOW_POWER_TESET_CI_1000MS || LOW_POWER_TESET_LATENCY_
↪100MS || LOW_POWER_TESET_LATENCY_1000MS
            LowPower_Test_Timer();
          #endif
        }

        break;

    case BLE_GAP_EVENT_DISCONNECT:
        printf("disconnect; reason=0x%02x\n",  (uint8_t)event->disconnect.reason);
        conn_handle = BLE_HS_CONN_HANDLE_NONE; /* reset conn_handle */

        /* Connection terminated; resume advertising */
        blehr_advertise();
        break;

    case BLE_GAP_EVENT_ADV_COMPLETE:
        printf("adv complete\n");
        blehr_advertise();
        break;

    case BLE_GAP_EVENT_SUBSCRIBE:
        printf("subscribe event; cur_notify=%d\n value handle; "
                        "val_handle=%d\n",
                    event->subscribe.cur_notify, hrs_hrm_handle);
        if (event->subscribe.attr_handle == hrs_hrm_handle) {
            notify_state = event->subscribe.cur_notify;
            blehr_tx_hrate_reset();
        } else if (event->subscribe.attr_handle != hrs_hrm_handle) {
            notify_state = event->subscribe.cur_notify;
            blehr_tx_hrate_stop();
        }
        break;

    case BLE_GAP_EVENT_MTU:
        printf("mtu update event; conn_handle=%d mtu=%d\n",
                    event->mtu.conn_handle,
                    event->mtu.value);

        break;

    }

    return 0;
}
```

以下为主机中的 gap 事件处理：

```
static int blecent_gap_event(struct ble_gap_event *event, void *arg)
{
    struct ble_gap_conn_desc desc;
    struct ble_hs_adv_fields fields;
    int rc;

    switch (event->type) {
```

(下页继续)

(续上页)

```
    case BLE_GAP_EVENT_DISC:
        rc = ble_hs_adv_parse_fields(&fields, event->disc.data,
                                     event->disc.length_data);
        if (rc != 0) {
            return 0;
        }

        /* An advertisment report was received during GAP discovery. */
        print_adv_fields(&fields);

        /* Try to connect to the advertiser if it looks interesting. */
        blecent_connect_if_interesting(&event->disc);
        return 0;

    case BLE_GAP_EVENT_CONNECT:
        /* A new connection was established or a connection attempt failed. */
        if (event->connect.status == 0) {
            /* Connection successfully established. */
            printf("Connection established ");

            rc = ble_gap_conn_find(event->connect.conn_handle, &desc);
            assert(rc == 0);
            print_conn_desc(&desc);
            printf("\n");

            /* Remember peer. */
            rc = peer_add(event->connect.conn_handle);
            if (rc != 0) {
                printf("Failed to add peer; rc=%d\n", rc);
                return 0;
            }

            /* Perform service discovery. */
            rc = peer_disc_all(event->connect.conn_handle,
                               blecent_on_disc_complete, NULL);
            if (rc != 0) {
                printf("Failed to discover services; rc=%d\n", rc);
                return 0;
            }
        } else {
            /* Connection attempt failed; resume scanning. */
            printf("Error: Connection failed; status=%d\n",
                        event->connect.status);
            blecent_scan();
        }

        return 0;

    case BLE_GAP_EVENT_DISCONNECT:
        /* Connection terminated. */
        printf("disconnect; reason=0x%02x\n",  (uint8_t)event->disconnect.reason);
        print_conn_desc(&event->disconnect.conn);
        printf("\n");

        /* Forget about peer. */
        peer_delete(event->disconnect.conn.conn_handle);

        /* Resume scanning. */
        blecent_scan();
        return 0;
```

(下页继续)

(续上页)

```
    case BLE_GAP_EVENT_DISC_COMPLETE:
        printf("discovery complete; reason=%d\n",
                    event->disc_complete.reason);
        return 0;

    case BLE_GAP_EVENT_ENC_CHANGE:
        /* Encryption has been enabled or disabled for this connection. */
        printf("encryption change event; status=%d ",
                    event->enc_change.status);
        rc = ble_gap_conn_find(event->enc_change.conn_handle, &desc);
        assert(rc == 0);
        print_conn_desc(&desc);
        return 0;

    case BLE_GAP_EVENT_NOTIFY_RX:
        /* Peer sent us a notification or indication. */
        printf("received %s; conn_handle=%d attr_handle=%d "
                        "attr_len=%d\n",
                    event->notify_rx.indication ?
                        "indication" :
                        "notification",
                    event->notify_rx.conn_handle,
                    event->notify_rx.attr_handle,
                    OS_MBUF_PKTLEN(event->notify_rx.om));

        /* Attribute data is contained in event->notify_rx.attr_data. */
        return 0;

    case BLE_GAP_EVENT_MTU:
        printf("mtu update event; conn_handle=%d cid=%d mtu=%d\n",
                    event->mtu.conn_handle,
                    event->mtu.channel_id,
                    event->mtu.value);
        return 0;

    case BLE_GAP_EVENT_REPEAT_PAIRING:
        /* We already have a bond with the peer, but it is attempting to
         * establish a new secure link.  This app sacrifices security for
         * convenience: just throw away the old bond and accept the new link.
         */

        /* Delete the old bond. */
        rc = ble_gap_conn_find(event->repeat_pairing.conn_handle, &desc);
        assert(rc == 0);
        ble_store_util_delete_peer(&desc.peer_id_addr);

        /* Return BLE_GAP_REPEAT_PAIRING_RETRY to indicate that the host should
         * continue with the pairing operation.
         */
        return BLE_GAP_REPEAT_PAIRING_RETRY;

    default:
        return 0;
    }
}
```

2.4.2 NDK 低功耗开发指南

本文主要通过一些示例，介绍**低功耗**各个模式、使用的方法以及可能遇到的问题。

1 低功耗模式

低功耗模式介绍如下表所示：

模式名称	进入	唤醒	1.2V 区时钟	时钟	1.2V 供电
STANDBY_M0	Sleep_mode = 3, Buck_en_ctrl=0, Buck_bp_ctrl=0, Flashldo_bp_en_ctrl =0, Flashldo_lp_en_en_ctrl =0, WFI	P00, P01, P02, BOD/LVR (可选，需保证慢时钟开启)，PIN RESET	全部关闭	全部关闭	断电
STANDBY_M1	Sleep_mode = 2, ldo_pwr_ctrl = 0, ldol_ pwr ctrl = 0/1, cpu pwr ctrl = 0/1, sram0/1 pwr ctrl = 0/1, ll_ram pwr ctrl=0/1, phy_ram pwr _ctrl=0/1, Buck_en_ctrl=0, Buck_bp_ctrl=0, Flashldo_bp_en_ctrl =0, Flashldo_lp_en_en_ctrl =0, WFI	所有 GPIO (边沿去抖)，SLPTMR, WDT, BOD/LVR (可选)，PIN RESET	CLK32K, 其他全部关闭	慢时钟	LPLDOL/H : LL_RAM (可选)，PHY_RAM/PHY_REGS (可选)，SRAM0/1 (可选)，decrypt_ram(可选，cpu 模块不保电，没办法做到只保少部分寄存器) LPLDOL/H: asnactrl_l (rcc 的寄存器) GPIOWDTBOD，LVR
DEEPSLEEP	Sleep_mode = 1, ldol_power_ctrl = 0/1, Buck_en_ctrl=0, Buck_bp_ctrl=0, Flashldo_bp_en_ctrl =0/1, Flashldo_lp_en_en_ctrl =1/0, WFI	所有 GPIO, SLPTMR, WDT, TIMER0/1/2, BOD/LVR (可选)，PIN RESET	CLK32K, 其他全部关闭	慢时钟	LPLDOL/H: 其他数字模块, LPLDOL/H: WakeupGPIO, WDT, Timer0/1/2, (需要测试，确认如何安全使用)
SLEEP	Sleep_mode = 0，WFI	所有外设中断, BOD/LVR (可选)，PIN RESET	CLK32K, CPU_CLK 关闭, RCH、XTH、DPLL 根据软件配置选择打开	慢时钟快时钟	HP_LDO 供电

2 开发流程

2.1 低功耗使用流程

2.1.1 Sleep 模式

- **进入流程：**

1. 配置 sleep_mode 为 sleep 模式
2. 唤醒源配置
3. 设置 flash dp_en 为 0

4. 设置 CPU SLEEPDEEP 寄存器为 0；地址：0xE000ED10
5. 考虑安全，建议进行一次手动 3V 同步操作
6. ___WFI()；

- **退出流程：**

1. 唤醒源产生中断；
2. 处理中断，清除中断源；

- **备注：**

1. 支持 m0 调试模式，不支持 riscv 调试模式

2.1.2 Deepsleep 模式 进入流程：

1. 配置 sleep_mode 为 deepsleep 模式，配置各电压域的 power switch，配置 rcl_en_ctrl，xtl_pwr_ctrl，Buck 和 flashldo 的配置建议使用 driver 默认

2.

Power Switch	模式 1（推荐）	模式 2	模式 3
lpldoh_en	1	1	0
lpldol_en	1	0	1
Ldo_pwr_ctrl	1	1	1
Ldol_pwr_ctrl	0	1	1
Peri_pwr_ctrl(cpu/ll_sram/phy_sram/sram0/sram1)	1（ram 可配）	1（ram 可配）	1（ram 可配）
Lpldoh_iso_en	1（配置 0 待测试）	0	0

3. 唤醒源配置：所有 GPIO，SLPTMR，WDT，TIMER0/1/2，BOD/LVR（可选），PIN RESET。对于模式 1，如果 Lpldoh_iso_en 配置为 1，则不支持 TIMER0/1/2 唤醒；如果 Lpldoh_iso_en 配置为 0，则支持 TIMER0/1/2 唤醒和 PWM 输出（需要测试是否有漏电）。对于模式 2 或者模式 3，上述唤醒源都可以唤醒系统。BOD，LVR 唤醒需要开启 32K 时钟。
4. 对于模式 1，如果 Lpldoh_iso_en 配置为 1，不支持 PWM 输出；如果 Lpldoh_iso_en 配置为 0，则支持 PWM 输出（需要测试是否有漏电）。对于模式 2 或者模式 3，PWM 可以输出
5. Flash dp 设置为 1，并退出 enhance 模式。配置合适的 dp，和 rdp 时间
6. 建议 cpu 地址重映射功能开启，映射地址为 ram 保电区域，可加快唤醒后，程序执行速度
7. Dly_time2 需要根据测试结果重新配置，默认值比较大
8. 设置 CPU SLEEPDEEP 寄存器为 1；地址：0xE000ED10
9. Flash 控制器退出增强型模式；
10. 考虑安全，建议进行一次手动 3V 同步操作
11. _WFI()；

退出流程：

1. 唤醒源唤醒，产生 lp 中断或者唤醒源相对应的中断
2. 清除相应 flag

备注：

1. 支持 m0 调试模式（低功耗期间会丢失），不支持 riscv 调试模式
2. gpio 唤醒电平，至少需要保持一个完整的 32K 时钟周期。如果需要读取的 gpio 的中断，需要 7 个 32K 时钟周期（需要 dly2 的延时决定）；如果 32K 时钟关闭，则唤醒需要的时间更久，和 32K 时钟的启动时间相关

2.1.3 Standby_m1 模式 进入流程：

1. 配置 sleep_mode 为 standby_m1 模式，配置各电压域的 power switch，配置 rcl_en_ctrl，xtl_pwr_ctrl，Buck 和 flashldo 的配置建议使用 driver 默认

2.

	模式 1（推荐，需测试）	模式 2	模式 3
lpldoh_en	1	1	0
lpldol_en	1	0	1
Ldo_pwr_ctrl	0	0	0
Ldol_pwr_ctrl	0	1	1
Peri_pwr_ctrl(cpu/ll_sram/phy_sram/sram0/sram1)	1(可配)	1(可配)	1(可配)
Lpldoh_iso_en	1	1	1

3. 唤醒源配置：所有 GPIO，SLPTMR，WDT，BOD/LVR（可选），PIN RESET。BOD，LVR 唤醒需要开启 32K 时钟。
4. flash 如果使用 4 线模式，建议开启 dp 模式，flash 两线切换四线的时间特别长，一般会有 8ms；如果 flash 使用 2 线模式，不建议开启 dp 模式，flash 直接掉电处理。
5. 建议 cpu 地址重映射功能开启，映射地址为 ram 保电区域，可加快唤醒后，程序执行速度
6. 根据需求，决定是否开启 cpu 保电功能，寄存器 LP_FL_CTRL[4]。可硬件恢复现场，代码实现有特定需求，参见说明部分
7. Dly_time2 需要根据测试结果重新配置，默认值比较大
8. 设置 CPU SLEEPDEEP 寄存器为 1；地址：0xE000ED10
9. 考虑安全，建议进行一次手动 3V 同步操作
10. __WFI();

退出流程：

1. 唤醒源唤醒，产生 lp 中断以及唤醒源相对应的中断
2. 清除相应 flag

备注：

1. 不支持 m0 和 riscv 调试模式
2. 支持 m0 的 cpu retention 功能（现场恢复），不支持 riscv 的 cpu retention 功能
3. gpio 唤醒电平，至少需要保持一个完整的 32K 时钟周期。如果需要读取的 gpio 的中断，需要 7 个 32K 时钟周期（需要 dly2 的延时决定）；如果 32K 时钟关闭，则唤醒需要的时间更久，和 32K 时钟的启动时间相关

2.1.4 Standby_m0 模式 进入流程：

1. 配置 sleep_mode 为 standby_m0 模式，配置 rcl_en_ctrl，xtl_pwr_ctrl
2. 唤醒源配置：所有 P00，P01，P02，BOD/LVR（可选），PIN RESET。
3. Flash dp 设置为 0
4. Dly_time1 根据需要决定是否配置，如果不在意 m0 的唤醒时间不建议去修改。此处的时间测试遍历会比较多，设置的值不好控制
5. 设置 CPU SLEEPDEEP 寄存器为 1；地址：0xE000ED10
6. 考虑安全，建议进行一次手动 3V 同步操作
7. __WFI();

退出流程：

1. 唤醒源唤醒，产生 lp 中断以及唤醒源相对应的中断
2. 清除相应 flag

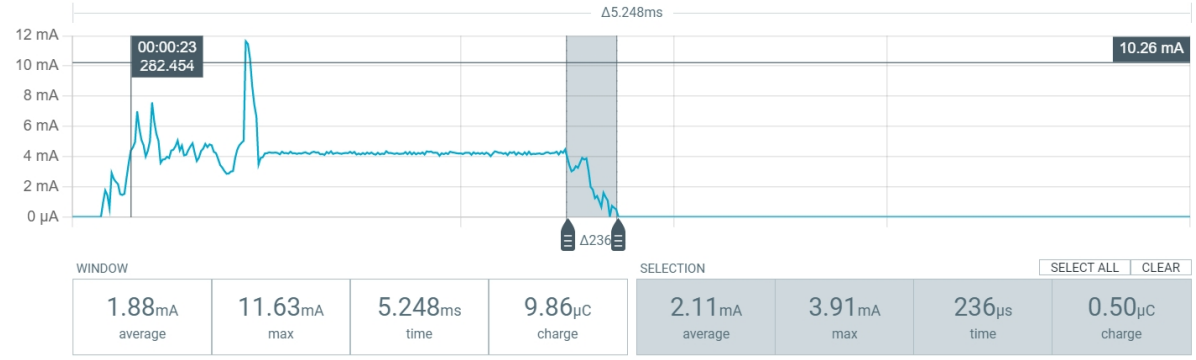
2.2 参考相关例程 当前 SDK 中提供了一些低功耗相关的例程，涵盖了章节 1 所述的所有低功耗模式等。

例程位置：03_MCU\mcu_samples\LP。

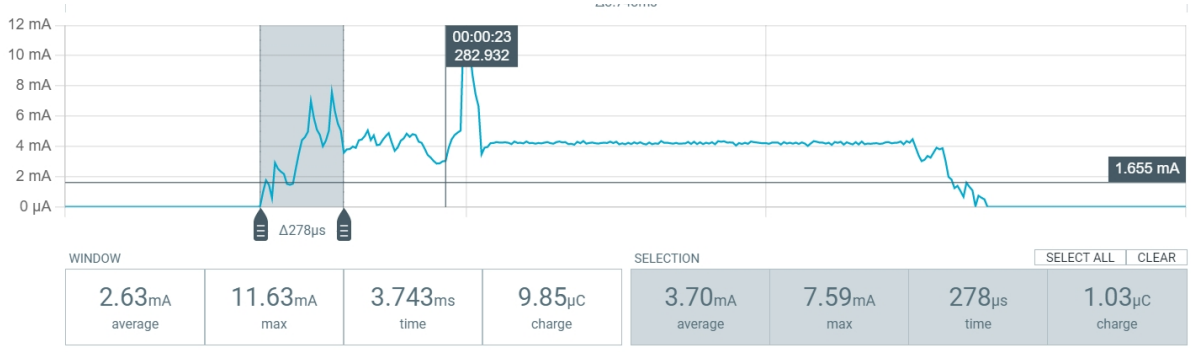
2.3 Standby_m1 休眠唤醒 以 standby m1 模式，cpu retention and cpu continue run 模式为基础介绍各个时间阶段 mcu 的动作。

阶段	说明	时 间 (us)	备注
进入休眠	从软件发送休眠指令至硬件完全休眠的时间	236	
硬件唤醒启动	唤醒源触发后硬件完全启动的时间	278	
Standby M1 retention 模式	continue run，唤醒至 rx ready 时间	465	此模式下**软件初始化和** RF **初始化步骤**可以省略
TX/RX(max 59B)	收发最大 payload 的时间，根据传输速率与字节数计算得出	1888	此处按最大数据计算
总计	休眠唤醒至 tx 完成/rx 完成的时间	2867	

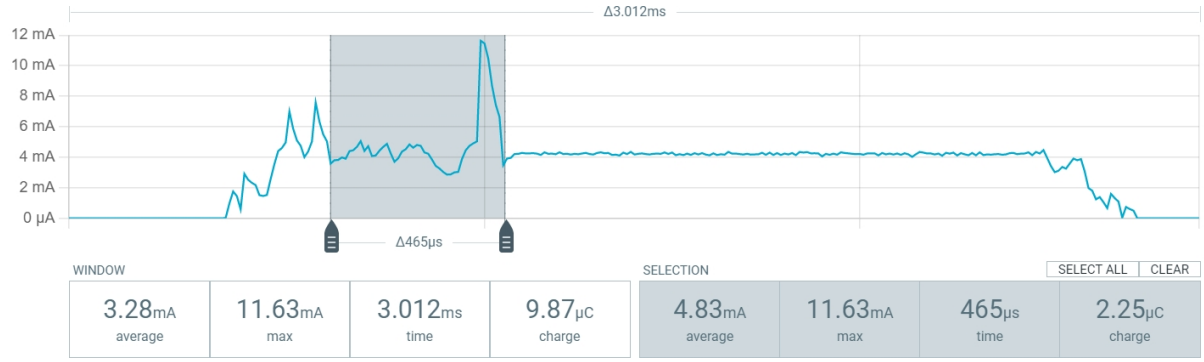
休眠时间



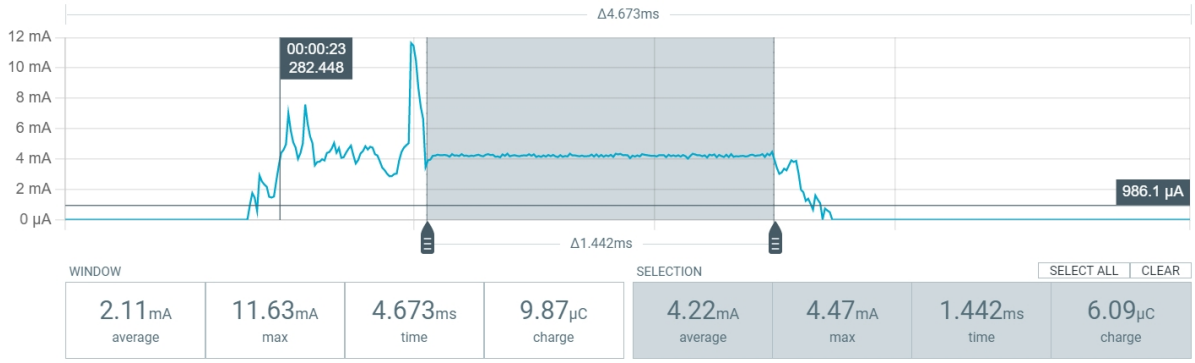
唤醒时间



软件运行至 RF ready 时间



RF 接收 32Byte 时间



3 低功耗注意事项

1. flash 运行在 4 线 enhance 模式，进入功耗前需要退出 enhance 模式。
2. 低功耗模式下供电分两部分 LPLDOL 和 LPLDOH，其中 LPLDOL 给 sram 供电，电压范围从 0.4~0.9v（未校准芯片有差异），LPLDOH 给 always on 区域部分供电，供电范围 0.5~1.2v（未校准芯片有差异），通常在进入低功耗在保证唤醒正常情况下尽量降低两个电压，常温下 LPLDOL/H 可设置未 0/1。
3. 为防止 LPLDOH 在电源抖动时出现不能唤醒的情况，增加了一个 LPLDOH_VREF_TRIM_AON（LP_LPLDO[23:21]）控制位，设置值的作用是弥补电源抖动，稳定电压，同时设置完成此值（例如设置为 2，LPLDOH 电压 0.7v）后再想拉低 LPLDOH 至 0.6v 是不能完成的，此属于正常现象。
4. DeepSleep 模式下外设 timer0/1/2 作为唤醒以及 PWM 在低功耗下输出需要将 deepsleep 低功耗模式设置为模式 2，即 dp_mode= LP_DEEPSLEEP_MODE2.
5. Standby M1 cpu retention 模式唤醒后外设部分及 RF MAC 层寄存器需要重新初始化，PHY、保电的 sram、时钟等不需要重新初始化

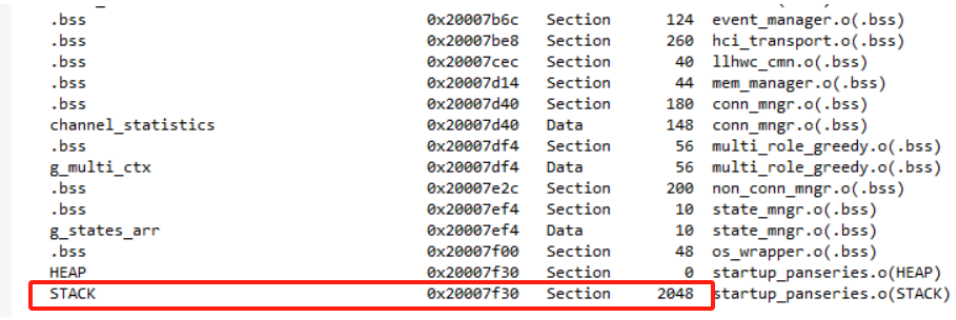
2.4.3 NDK RAM 使用情况分析以及优化指南

1 如何查看 KEIL 的 RAM 和 Flash 使用情况

因为当前工程中很多和 BLE 时间处理相关的函数为了提升处理速度使用了 RAM CODE，所以导致部分 CODE 会占用 RAM 空间，所以查看工程的实际占用空间要从 RAM 和 Flash 空间进行查看。

首先双击 keil 项目打开 map 文件

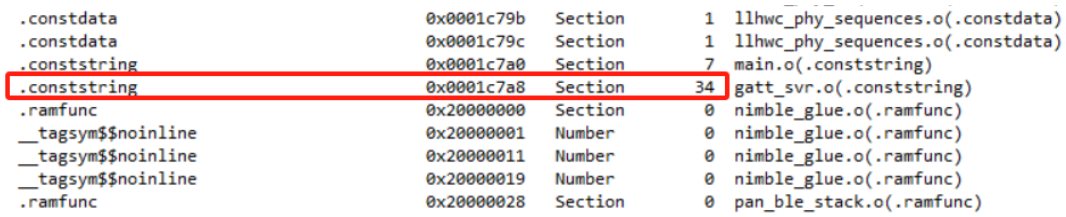
1.1 **如何查看 RAM 空间：** 因为 STACK 占用 RAM 边界位置，我们可以通过 STACK 占用可以知道 RAM 最多占用 `0x20007f30+0x800(2048) = 0x20008730` 的 RAM，因为 RAM 地址默认是 `0x20000000` 起始的，所以我们知道 RAM 占用了 `0x8730`（34608）字节的 RAM。



.bss	0x20007b6c	Section	124	event_manager.o(.bss)
.bss	0x20007be8	Section	260	hci_transport.o(.bss)
.bss	0x20007cec	Section	40	llhwc_cmn.o(.bss)
.bss	0x20007d14	Section	44	mem_manager.o(.bss)
.bss	0x20007d40	Section	180	conn_mngr.o(.bss)
channel_statistics	0x20007d40	Data	148	conn_mngr.o(.bss)
.bss	0x20007df4	Section	56	multi_role_greedy.o(.bss)
g_multi_ctx	0x20007df4	Data	56	multi_role_greedy.o(.bss)
.bss	0x20007e2c	Section	200	non_conn_mngr.o(.bss)
.bss	0x20007ef4	Section	10	state_mngr.o(.bss)
g_states_arr	0x20007ef4	Data	10	state_mngr.o(.bss)
.bss	0x20007f00	Section	48	os_wrapper.o(.bss)
HEAP	0x20007f30	Section	0	startup_panseries.o(HEAP)
STACK	0x20007f30	Section	2048	startup_panseries.o(STACK)

图 25: 通过 STACK 查看 RAM 占用

1.2 **如何查看 Flash 空间** 从 RAM 空间往上查找 flash 边界位置，0x20000000 前面的即是 flash 最大地址



.constdata	0x0001c79b	Section	1	llhwc_phy_sequences.o(.constdata)
.constdata	0x0001c79c	Section	1	llhwc_phy_sequences.o(.constdata)
.conststring	0x0001c7a0	Section	7	main.o(.conststring)
.conststring	0x0001c7a8	Section	34	gatt_svr.o(.conststring)
.ramfunc	0x20000000	Section	0	nimble_glue.o(.ramfunc)
__tagsym$$noinline	0x20000001	Number	0	nimble_glue.o(.ramfunc)
__tagsym$$noinline	0x20000011	Number	0	nimble_glue.o(.ramfunc)
__tagsym$$noinline	0x20000019	Number	0	nimble_glue.o(.ramfunc)
.ramfunc	0x20000028	Section	0	pan_ble_stack.o(.ramfunc)

图 26: 查看 flash 边界

通过 map 文件找到边界地址 0x1c7a8+0x22(34) = 0x1c7ca(116682) bytes。

2 关于堆空间的使用说明

2.1 **蓝牙 controller 的堆** 蓝牙 controller 的堆默认是用于 controller 的广播，连接等各种数据包动态分配使用的。而且是从 host 定义来进行分配的，但是不同的参数可能导致堆的需求空间不一致。

我们可以通过打开 `app_config.h` 或者 `app_config_spark.h` 中的 `BT controller Memory Pool usage print` 选项（对应的宏 `CONFIG_CNTRL_MEM_POOL_PRINT`）显示的输出底层 controller 所需要的内存。

正常分配时如下：

```
[19:32:53.628]收 ←LL Controller Version:bd5923c

[19:32:53.665]收 ←BT controller memory pool used: 400 bytes, remain bytes: 8496, total:8896
BT controller memory pool used: 764 bytes, remain bytes: 8132, total:8896
BT controller memory pool used: 3784 bytes, remain bytes: 5112, total:8896
BT controller memory pool used: 4840 bytes, remain bytes: 4056, total:8896
BT controller memory pool used: 5164 bytes, remain bytes: 3732, total:8896
BT controller memory pool used: 6124 bytes, remain bytes: 2772, total:8896
BT controller memory pool used: 8896 bytes, remain bytes: 0, total:8896
app started
```

上面的 log 显示正常分配的对内存为 8896bytes，所以我们可以打开 `nimble_glue.c` 或者 `nimble_glue_spark.c` 找到堆分配的宏 `PAN_BLE_CTLR_BUFFER_ALLOC` 将其的值修改为 `8896`。

```
#define PAN_BLE_CTLR_BUFFER_ALLOC        (8896)
#define PAN_BLE_CTLR_BUFFER_SIZE        (((PAN_BLE_CTLR_BUFFER_ALLOC) + 3)& (~((uint32_
↪t)0x03))) /*4 bytes aligned*/

static uint32_t mem_buffer[PAN_BLE_CTLR_BUFFER_SIZE/4];
static uint32_t mem_pos;
```

我们也可以将堆修改为异常很小的值，比如此处设置为 4000，看下实际输出：

```
[19:38:01.115] 收 ← LL Controller Version:bd5923c

[19:38:01.151] 收 ← BT controller memory pool used: 400 bytes, remain bytes: 3600, total:4000
BT controller memory pool used: 764 bytes, remain bytes: 3236, total:4000
BT controller memory pool used: 3784 bytes, remain bytes: 216, total:4000
BT controller allocating 1056 bytes failed
```

此时分配失败后会触发断言在初始化的地方卡住。

我们可以一开始设置比较大的堆值，然后再调整为合适的值即可。

2.2 App 以及 host 使用的堆（应用层堆全局使用 freertos 的堆） 为了方便资源管理，我们 app 和 host 全局使用 freertos 的堆，相应堆的分配函数 `pvPortMalloc`。

当前 SDK 哪些资源默认使用了 freertos 的堆呢?

- app 中显式使用 `pvPortMalloc` 的地方
- freertos task 的栈
- 创建 freertos 定时器时的栈（定时器也是 task)、
- freertos 创建信号量等

2.3 如何查看 freertos heap 的使用 我们可以通过打开 `app_config.h` 或者 `app_config_spark.h` 中的 `FreeRTOS Heap Usage Print` 选项（对应的宏 `CONFIG_FREERTOS_HEAP_PRINT`）显示的输出底层 controller 所需要的内存。

freertos heap 的宏是 `FreeRTOSConfig.h` 中的 `configTOTAL_HEAP_SIZE`，我们可以通过修改 `configTOTAL_HEAP_SIZE` 的值来改变全局堆的大小。

启动时会输出如下:

```
[19:47:57.875] 收 ← total allocated bytes:216,remain:5920
total allocated bytes:304,remain:5832
total allocated bytes:392,remain:5744
total allocated bytes:480,remain:5656
total allocated bytes:536,remain:5600
total allocated bytes:704,remain:5432
total allocated bytes:792,remain:5344
LL Controller Version:bd5923c

[19:47:57.918] 收 ← app started
total allocated bytes:848,remain:5288
total allocated bytes:2864,remain:3272
total allocated bytes:2960,remain:3176
total allocated bytes:3232,remain:2904
total allocated bytes:3328,remain:2808
total allocated bytes:4368,remain:1768
total allocated bytes:4464,remain:1672
total allocated bytes:4520,remain:1616
total allocated bytes:4752,remain:1384
total allocated bytes:4776,remain:1360
total allocated bytes:4808,remain:1328
```

我们故意将 `configTOTAL_HEAP_SIZE` 设置为一个很小的值，分配失败是会有如下显示:

```
[19:50:28.736] 收 ← total allocated bytes:216,remain:2848
total allocated bytes:304,remain:2760
```

(下页继续)

(续上页)

```
total allocated bytes:392,remain:2672
total allocated bytes:480,remain:2584
total allocated bytes:536,remain:2528
total allocated bytes:704,remain:2360
total allocated bytes:792,remain:2272
LL Controller Version:bd5923c

[19:50:28.778] 收 ← app started
total allocated bytes:848,remain:2216
total allocated bytes:2864,remain:200
total allocated bytes:2960,remain:104
pvPortMalloc failed
allocate 272 bytes failed,remain:104
```

另外我们也要注意一点，堆定义的空间可以适当多分配一点，有些堆的分配是在运行时才会去调用。

2.4.4 NDK 常见问题（FAQs）

TBD

2.5 其他文档

PAN1070 SoC 相关的其他文档请参考：

- PAN107x BQB Test Report
- PAN1070 功耗测试报告

2.6 量产测试

2.6.1 量产烧录

1. 芯片硬件系统说明

如果芯片按照我司提供的 PAN107x **硬件参考设计**设计的模块，烧录连接如表 1-1、表 1-2 所示。

J-Flash	连接	PAN107x 芯片模块
VTref 3.3V	<—>	VBAT
GND	<—>	GND
SWDIO	<—>	P01
SWDCLK	<—>	P00

PAN-LINK2.0	连接	PAN107x 芯片模块
VDD	<—>	VBAT
GND	<—>	GND
A2	<—>	RST
A3	<—>	P01
A4	<—>	P00

如果需要烧录**裸芯片**，没有任何外围器件的 PAN107x 芯片烧录连接如表 1-3、表 1-4 所示。

J-Flash	连接	PAN107x 裸芯片
VTref 3.3V	<—>	VCC_RF
GND	<—>	GND（注：LQFP64:GND QFN48:49 脚 (ePAD) QFN32:33 脚 (ePAD)）
SWDIO	<—>	P01
SWDCLK	<—>	P00

PAN-LINK2.0	连接	PAN107x 裸芯片
VDD	<—>	VCC_RF
GND	<—>	GND（注：LQFP64:GND QFN48:49 脚 (ePAD) QFN32:33 脚 (ePAD)）
A2	<—>	RST
A3	<—>	P01
A4	<—>	P00

2. 量产烧录工具

为配合 PAN-LINK2.0 烧录 PAN107x 芯片程序工具。

下载

2.1. **硬件准备**　预先将 PAN-LINK2.0 通过 MiniUSB 线连接到 PC 电脑。

图 27: 图 2-1-1 PAN-LINK2.0 烧录器

如果 PAN-LINK2.0 固件程序不支持 PAN107x 芯片烧录，则需要根据提示自动更新升级。
或按照帮助文档方法更新 PAN-LINK2.0 固件程序。

2.1.1.PAN107x **芯片烧录接线**　注：PAN-LINK2.0 接口的 VCC 与 VIO 通过跳线帽短接。

PAN-LINK2.0 接口脚	连接	PAN107x 芯片脚
VDD	<—>	VDD
GND	<—>	GND
A1	<—>	RST
A3	<—>	P01
A4	<—>	P00

图 28: 图 2-1-2 MiniUSB 连接线

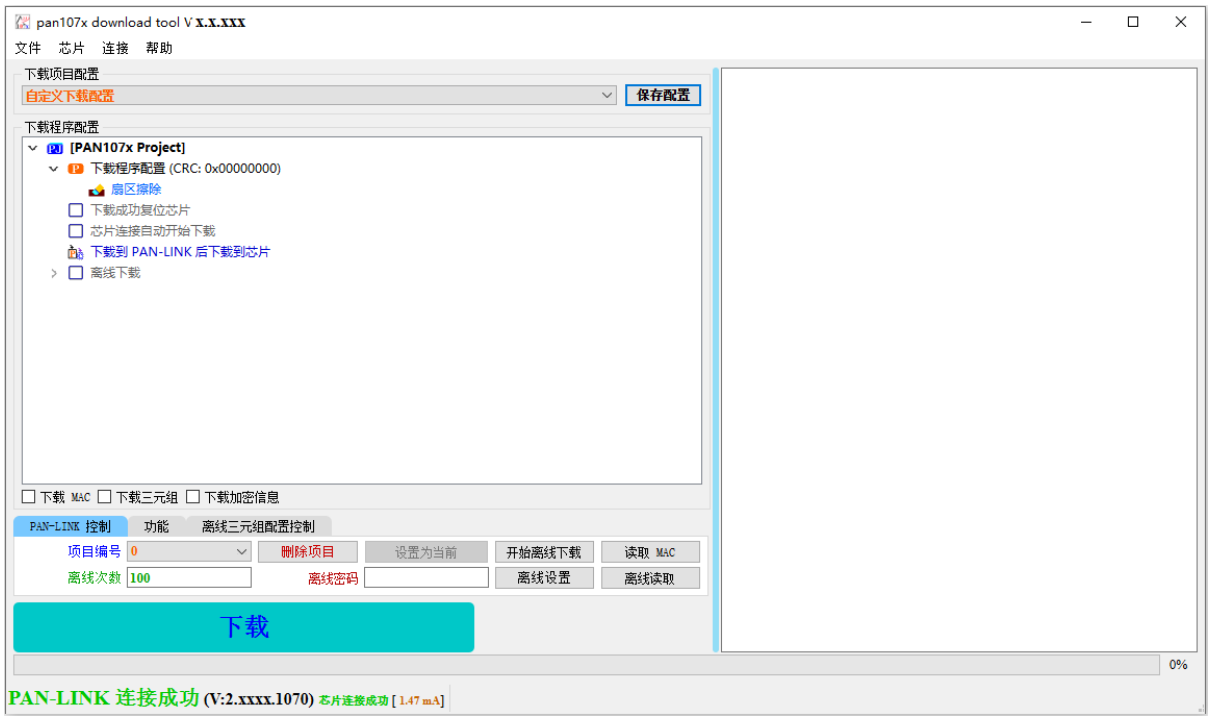


图 29: 图 2-2-1 烧录工具界面

2.2. **工具界面**　如上图 2-2-1 所示为烧录工具界面。

1、在下载程序配置中的下载程序配置项右键点击加载程序，实现加载烧录程序功能。

2、通过点击擦除模式前面图标或右键选择更改烧录擦除模式。

3、根据需求选择设置其他下载配置。

4、选择下载模式，或直接默认下载到 PAN-LINK 后下载到芯片模式。

5、点击下载开始下载程序到芯片。

2.3. **查看帮助文档**　通过烧录工具的**帮助**-> **查看帮助文档**或直接通过快捷键 F1, 打开查看帮助文档。PAN-LINK2.0 程序更新方法、以及烧录工具的详细使用说明都在帮助文档中有详述。

图 30: 图 2-3-1 查看帮助文档

2.7 NDK 更新日志

2.7.1 PAN1070 NDK v0.2.0

PAN1070 Nimble DK `v0.2.0` (2023-11-19) 已发布:

1. SDK

nimble

- 更新 BLE Controller，优化一些内部流程并修复一些问题
- 新增获取 MAC 地址的接口

Panchip HAL

- 新增载入 Hardware Calibration 校准参数的接口
- 优化 WDT 接口，扩大 WDT Reset 的复位范围
- 更新 RF Lib，优化 2.4G 通信流程

演示例程

- `ble_cent_prph`（新增）：演示蓝牙主从一体功能
- `ble_central`（新增）：演示蓝牙主机功能
- `bleprph_hr`（新增）：演示蓝牙从机功能，包含 GATT 服务：HR (Heart Rate)，连接订阅服务后，会上报虚拟的心率值
- `bleprph_enc`（新增）：演示外设以及加密配对功能，可以和主机示例进行对测
- `ble_hid_selfie`（新增）：自拍解决方案，通过蓝牙 HID 控制手机拍照
- `ble_panchip_cte_beacon`（新增）：Panchip 蓝牙定位标签方案，通过发送特定的广播数据，实现蓝牙定位功能
- `ble_rgb_light`（新增）：蓝牙 RGB 灯控方案，演示 BLE RGB 灯与手机 APP 进行连接，通过 APP 控制 RGB 灯的亮度与颜色
- `ble_hid_uart_mult_roles`（新增）：蓝牙串口透传解决方案，演示蓝牙 hid 串口透传功能，支持 1 主 1 从
- `ble_vehicles_key>`（新增）：蓝牙车钥匙解决方案，演示基于 HID 服务的自动连接服务

2. HDK

- 新增 PAN1070 UA1A EVB 图纸、设计源文件、生产文件

3. MCU

- 更新 LP 低功耗例程，优化 CPU Retention and Remap 流程
- 更新 2.4G 例程及对应文档，演示更多的通信模式
- 更新各个底层 Driver 例程，增加初始化阶段载入芯片校准信息的流程

4. DOC

- 新增 `ble_cent_prph` 例程文档
- 新增 `ble_central` 例程文档
- 新增 `bleprph_enc` 例程文档
- 新增 `bleprph_hr` 例程文档
- 新增 `ble_hid_selfie` 例程文档
- 新增 `ble_hid_uart_mult_roles` 例程文档
- 新增 `ble_pcte_beacon` 例程文档
- 新增 `ble_rgb_light` 例程文档
- 新增 `ble_vehicles_key` 例程文档
- 新增 `NDK App` 开发指南文档
- 新增 `PAN107x` 硬件参考设计文档
- 新增 量产烧录说明文档

5. TOOLS

- 新增量产烧录工具 `PAN107x Download Tool`
- 新增 Testbox RF 测试固件

2.7.2 PAN1070 NDK v0.1.0

PAN1070 Nimble DK `v0.1.0` (2023-10-24) 已发布：

1. SDK

NDK 软件开发框架基于 Keil + FreeRTOS + NimBLE，其中：

- Keil 是 SDK 支持的软件开发环境
- FreeRTOS 是一个开源实时操作系统（RTOS），用于配合 NimBLE 实现蓝牙应用
- NimBLE 是一个开源低功耗蓝牙（BLE）5.1 协议栈，其实际上是 Apache Mynewt 项目的一部分

解决方案

- `esl`: ESL 价签方案演示例程，支持外部 SPI Flash 存储、EPD 墨水屏、低功耗模式、RF 通信等功能。

2. HDK

目前版本提供了如下硬件相关资料：

- PAN107B QFN40 测试板图纸、设计源文件、生产文件

3. MCU

目前版本提供了如下 MCU 裸机 Keil 例程及相关文档：

- ADC
- CLK
- CLKTRIM
- DebugProtect
- DMA
- EFUSE
- FMC
- GPIO
- I2C
- LP
- PRF_B250K_RX
- PRF_B250K_TX
- PWM
- SPI
- TIMER
- UART
- USB_HID
- WDT
- WWDT

4. DOC

目前版本提供了如下文档：

- NDK 快速入门指南
- NDK 开发环境介绍
- NDK 整体框架介绍
- Nimble 简介
- PAN107x 硬件参考设计指南
- ESL 电子货架标签方案例程说明
- MCU 底层外设驱动例程说明
- 低功耗开发指南
- NDK RAM 使用情况分析以及优化指南

5. TOOLS

目前版本提供了如下工具：

- 串口工具（PC 工具）
- Air Sync Debugger（手机测试软件安卓 APK）
- Google Home（手机测试软件安卓 APK）
- nRF Connect（手机测试软件安卓 APK）
- nRF Mesh（手机测试软件安卓 APK）
- Siliconlabs Bluetooth Mesh（手机测试软件安卓 APK）

6. 已知问题

- MCU USB_HID 例程暂未通过测试