PAN1080 开发套件使用手册

发布 0.3.0

磐启微电子 PAN1080 项目组

2022 年 06 月 22 日

Table of contents

Chapter 1

快速入门

1.1 SDK 快速入门

1.1.1 1 概述

本文是 PAN1080 SoC 开发的快速入门指引，旨在帮助使用者快速入门 PAN1080 SDK 的开发，搭建软硬件环境，并编译、运行、调试一个例程。

1.1.2 2 PAN1080 EVB 硬件资源介绍

请参考：PAN1080 EVB 硬件资源介绍

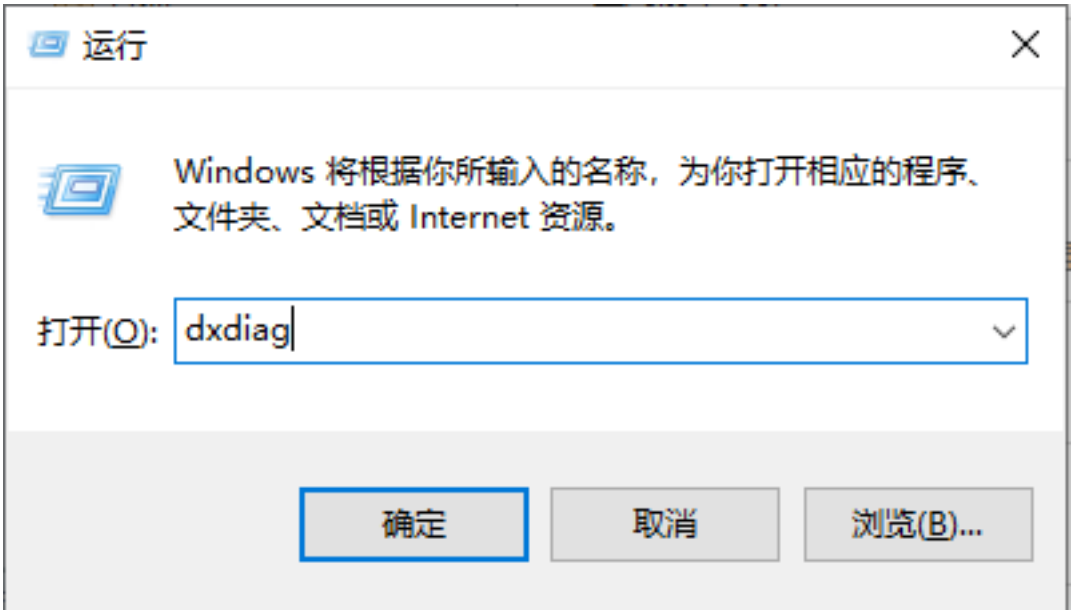
1.1.3 3 PAN1080 SDK 开发环境确认

3.1 PC 环境检查

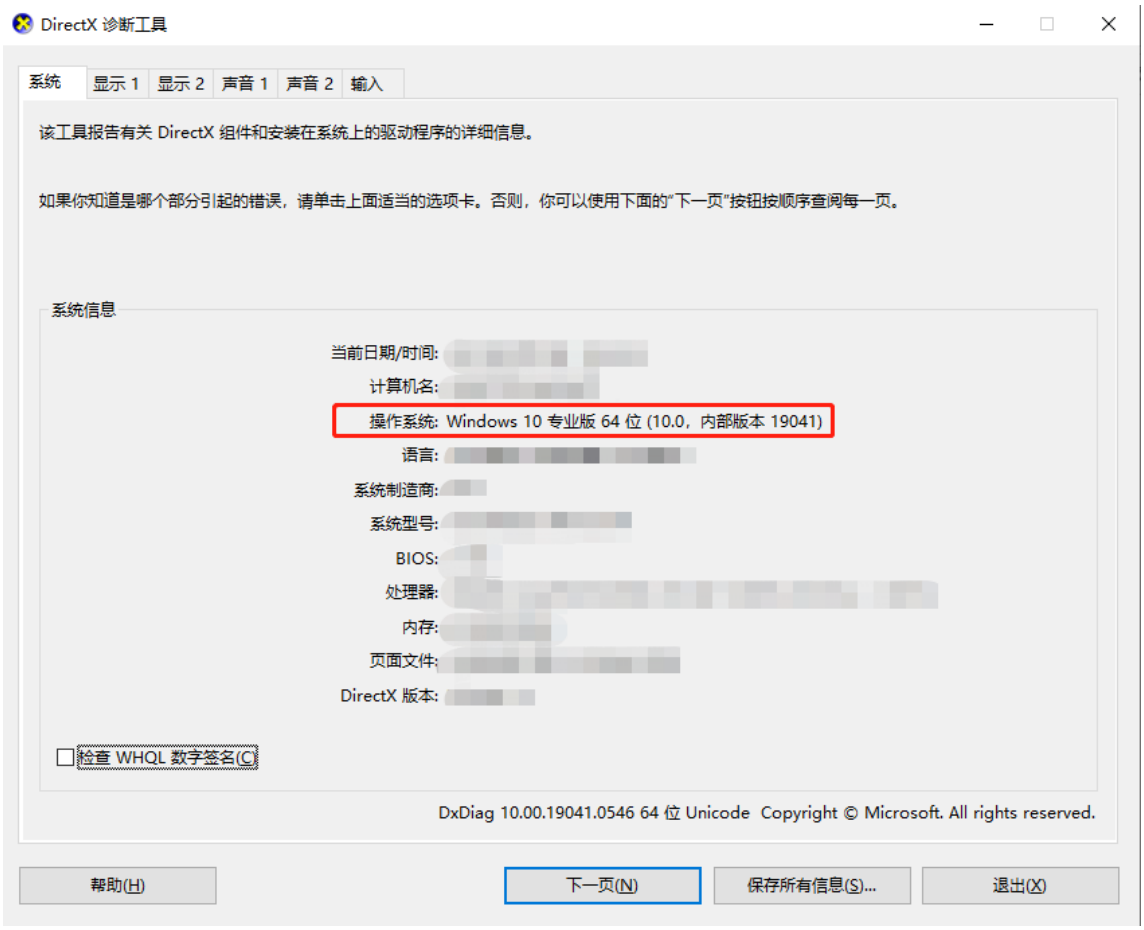
目前 PAN1080 提供的编译工具链只支持 Windows 7 及以上版本的 64 位操作系统，请确保您的开发环境满足此要求。

您可以按照如下操作快速确认您的 PC 是否满足要求：

1. 按快捷键 `Win + R`，在弹出的运行对话框中输入 `dxdiag` 并回车：

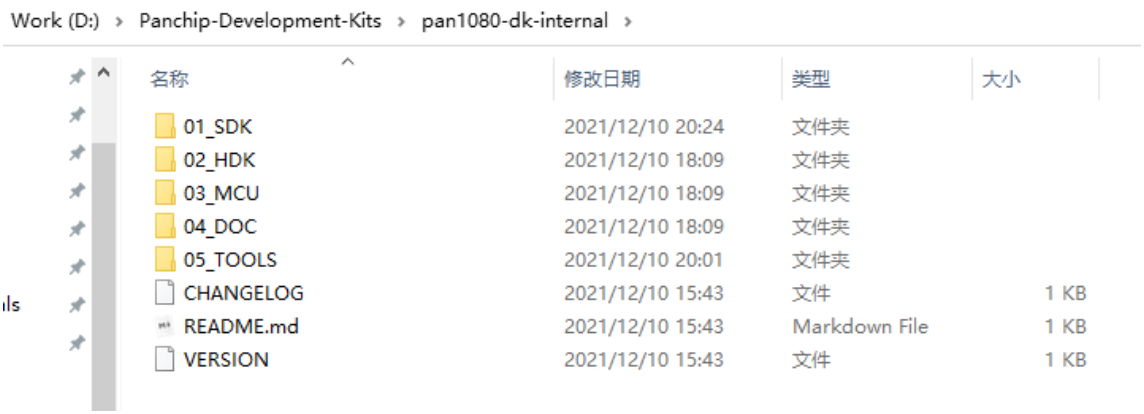


2. 稍等片刻，在弹出的 DirectX 诊断工具对话框中，可以看到当前 PC 的系统信息：



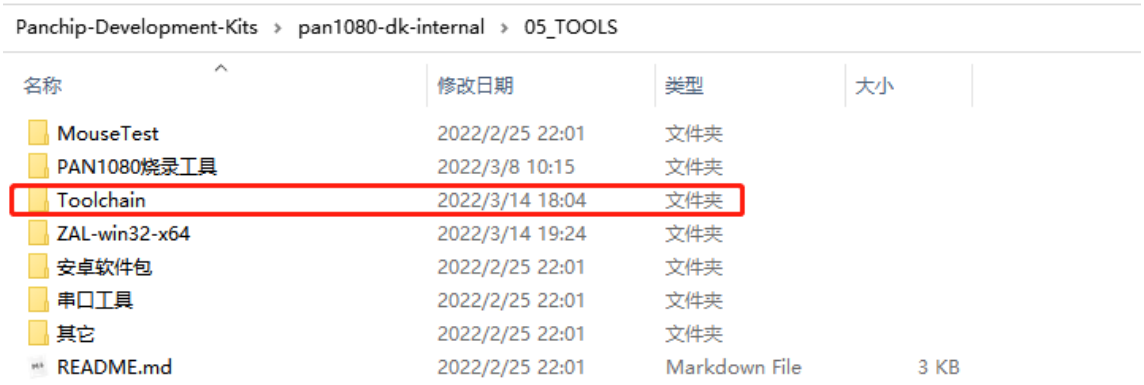
3.2 SDK 编译工具链环境确认

1. 请确保 PC 上已经正常解压缩了 `PAN1080 Development Kit` 开发包：



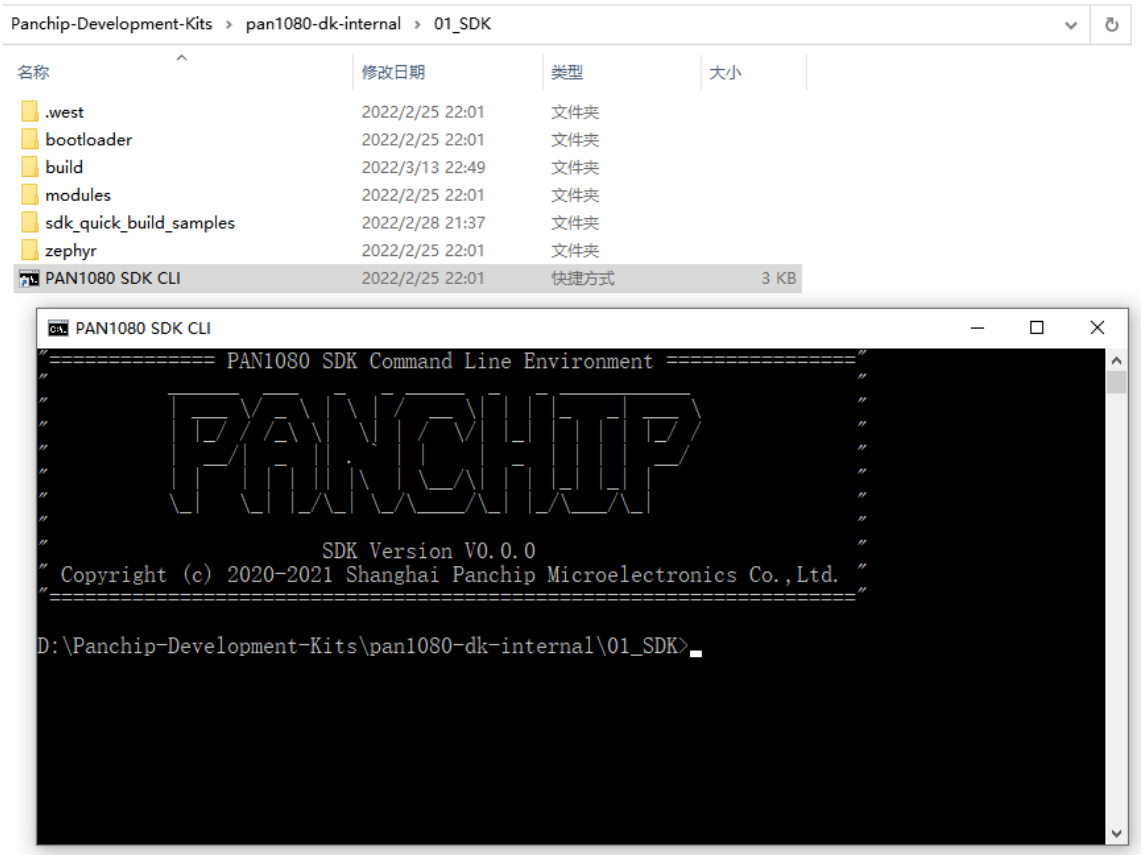
PAN1080 Development Kit 最新版本开发包可以从 Panchip Wiki 下载，地址：https://wiki.panchip.com/ble/pan1080_series/

2. 请确认 PAN1080 Development Kit 开发包中，05_TOOLS 目录下正常解压缩了编译工具链，即存在 Toolchain 目录：



编译工具链 Toolchain 可以从 Panchip 官方论坛下载，地址：http://bbs.panchip.com/forum.php?mod=viewthread&tid=7518&page=1&extra=#pid7827

3. 进入 01_SDK 目录，双击 PAN1080 SDK CLI 图标，如果弹出如下所示的信息，则说明环境配置成功：

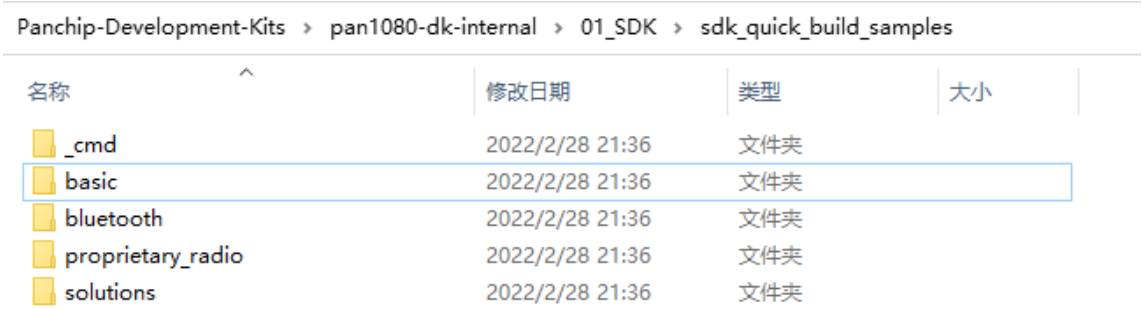


注意：开发环境搭建过程中，请一定确保以下 2 **个压缩包**都已经下载解压并按照上述步骤配置完成！其中，

1. `PAN1080 Development Kit` 压缩包中含有 PAN1080 SDK 的核心文件，包括 OS 内核、蓝牙协议、示例程序等等；
2. `Toolchain` 压缩包中含有编译调试 PAN1080 SDK 所需的所有工具，包括 GCC-ARM、CMake、Ninja、Python、JLink、VS Code 等等。

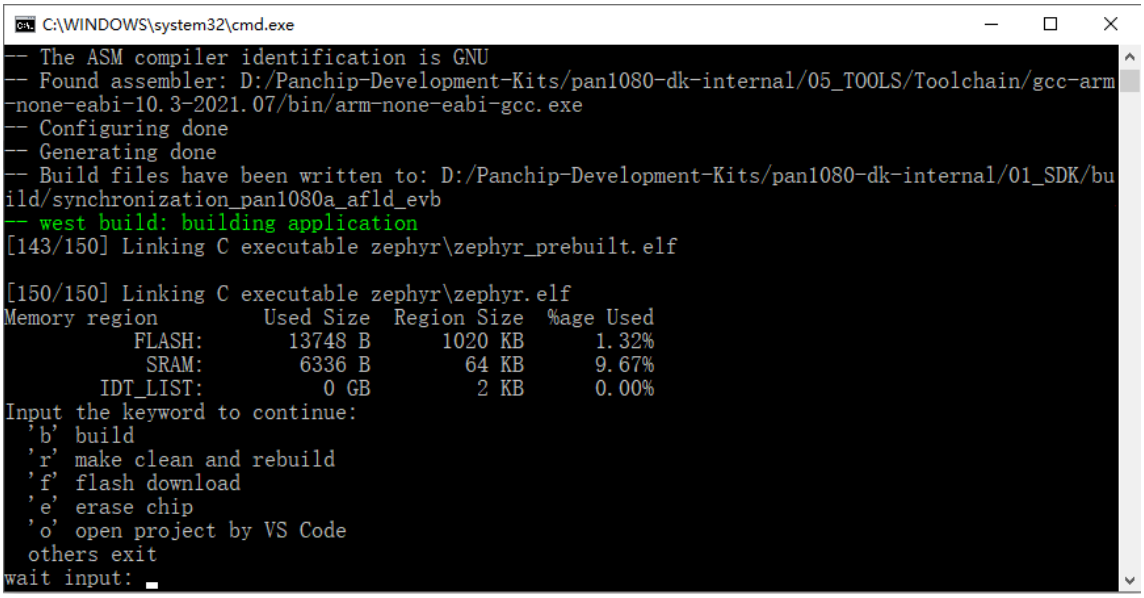
3.3 快速编译运行一个简单的例程

1. 硬件接线准备，请确认已经将 PAN1080 EVB 板的：
   1. SWD（P46：SWD_CLK，P47：SWD_DAT，GND：SWD_GND）接口通过 JLink 连接至 PC
   2. UART1（P06：Tx，P07：Rx）接口通过 USB 转串口模块连接至 PC
2. 进入 01_SDK/quick_build_samples 子目录，这里列出了 PAN1080 SDK 中提供的所有例程的快速编译运行脚本：



3. 这里以编译运行一个**多线程打印消息例程**为例，进入 `basic` 子目录，双击 `synchronization.bat` 脚本，开始 build：

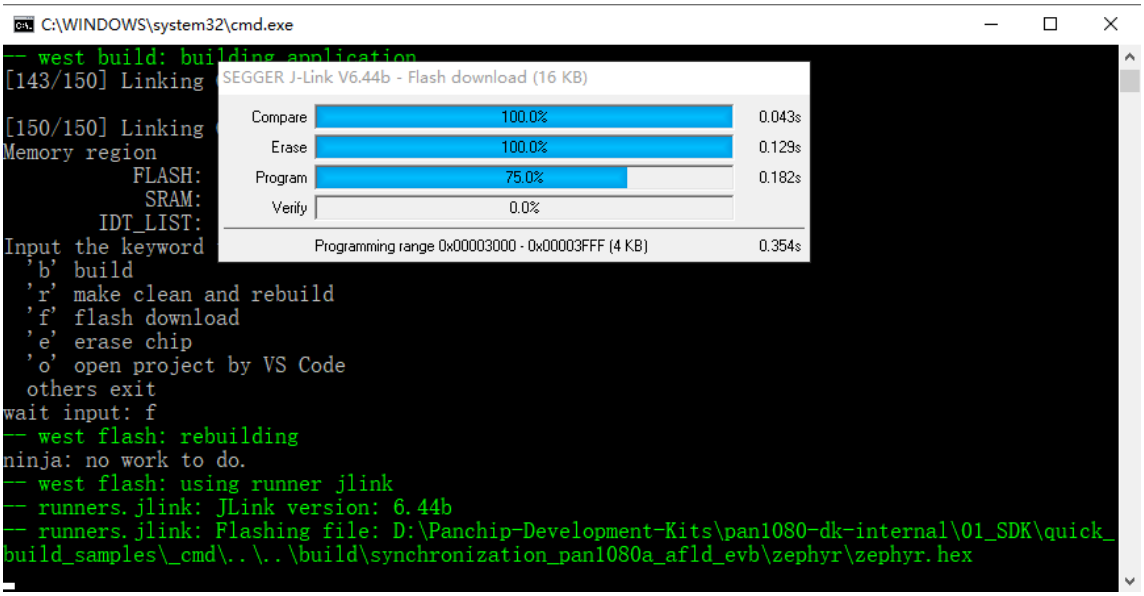
4. 稍等片刻，看到如下等待用户后续输入的界面，即表示 build 成功：



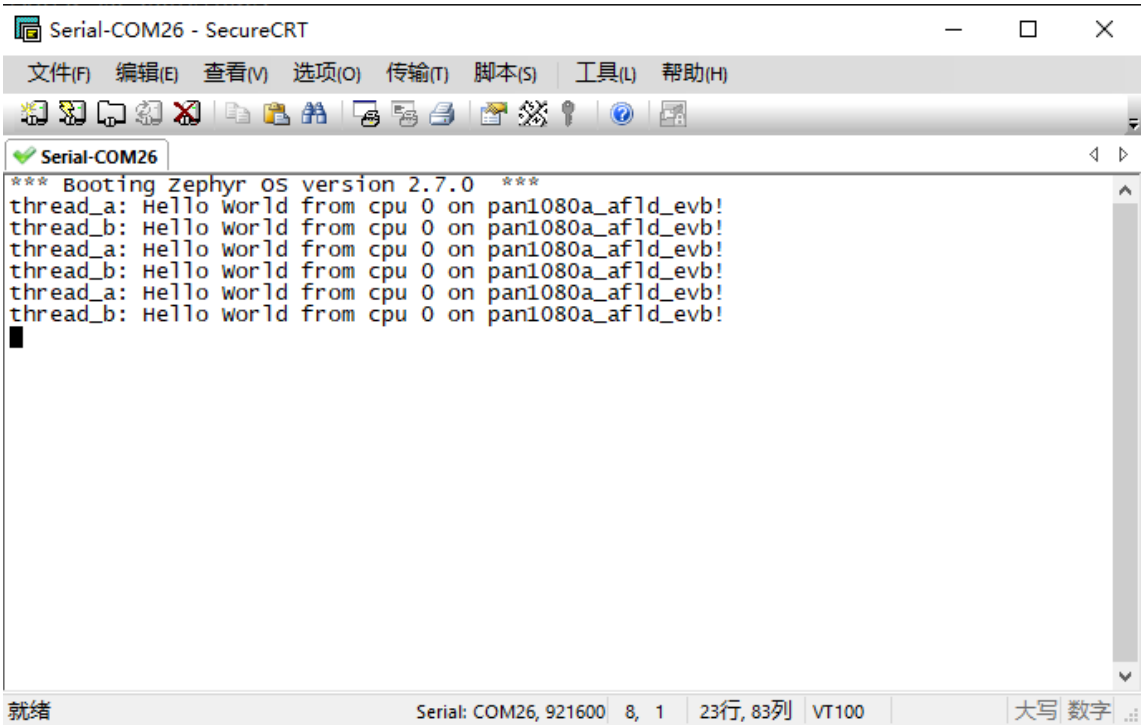
Build 成功后，可以看到 01_SDK 目录下生成了 build/synchronization_pan1080a_afld_evb 子目录，进入 build/synchronization_pan1080a_afld_evb/zephyr 子目录下，可以看到一些关键的编译输出文件：

- `zephyr.bin`：编译输出的二进制格式 Image
- `zephyr.dts`：DeviceTree 配置
- `zephyr.elf`：编译输出的带调试信息的 Image
- `zephyr.hex`：编译输出的十六进制文本格式 Image
- `zephyr.lst`：反汇编文件
- `zephyr.map`：Map 文件

5. 在上一步骤等待界面，输入 f 并回车，即开始执行 JLink 烧录过程：

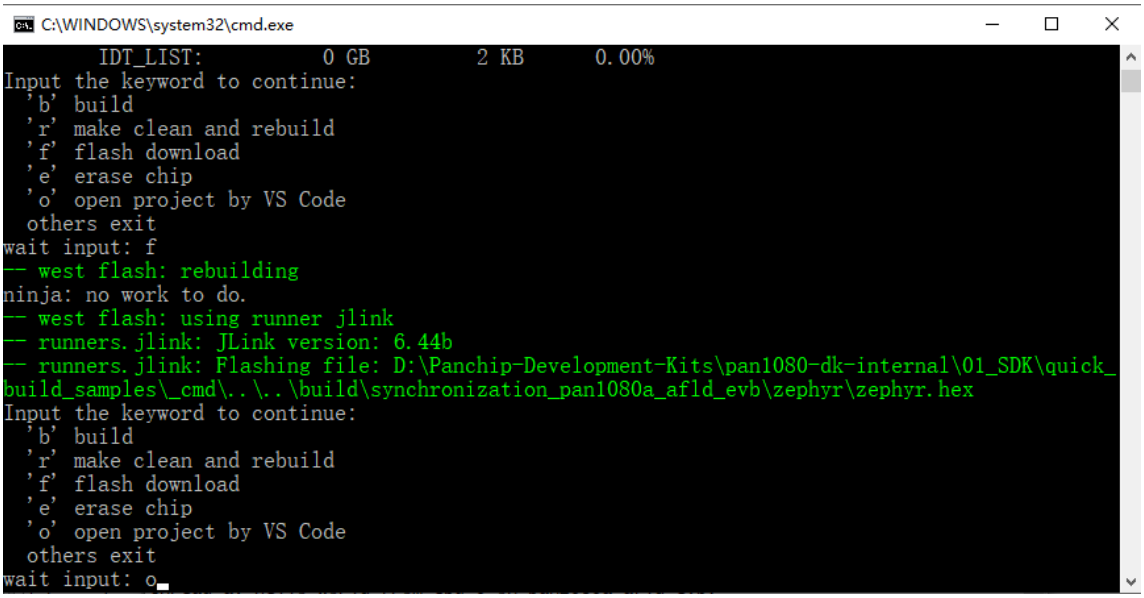


6. 烧录完成后，打开串口工具（如 SecureCRT），将波特率设置为 921600，看到两个线程交替打印的 Log，即表示程序烧录执行成功：

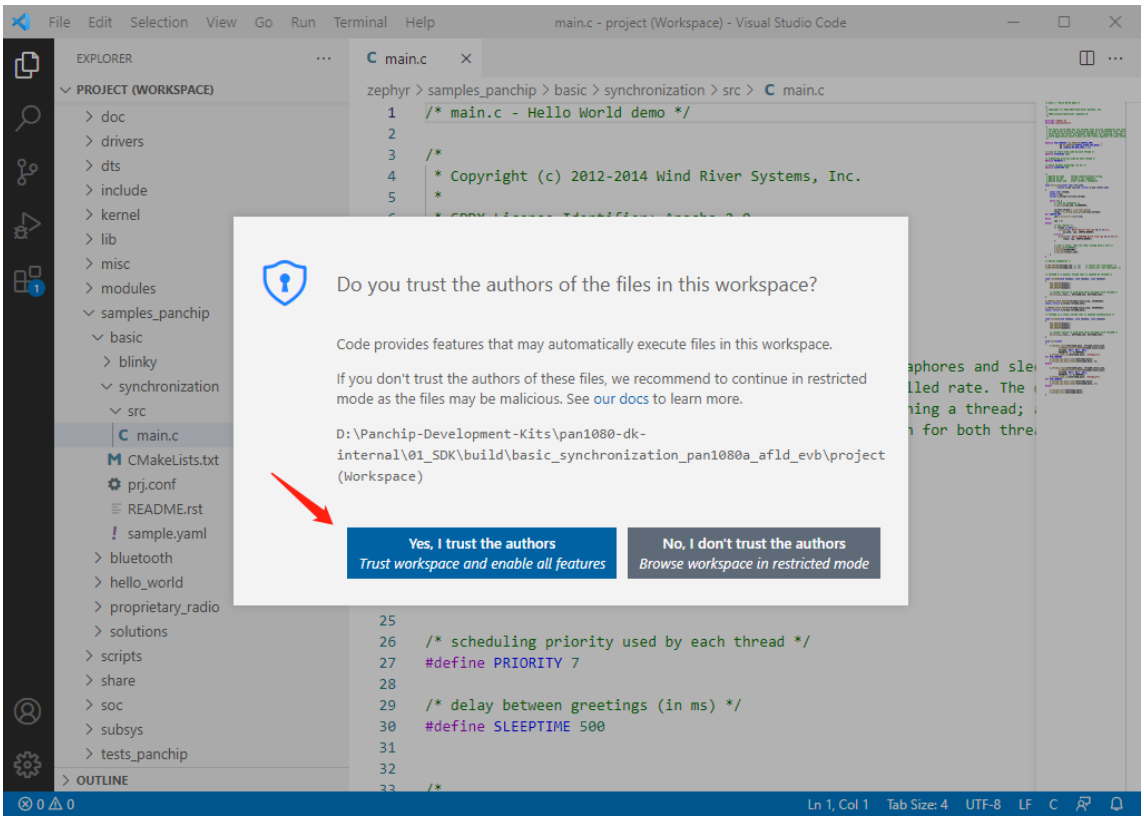


3.4 使用 VS Code 修改、编译、烧录程序

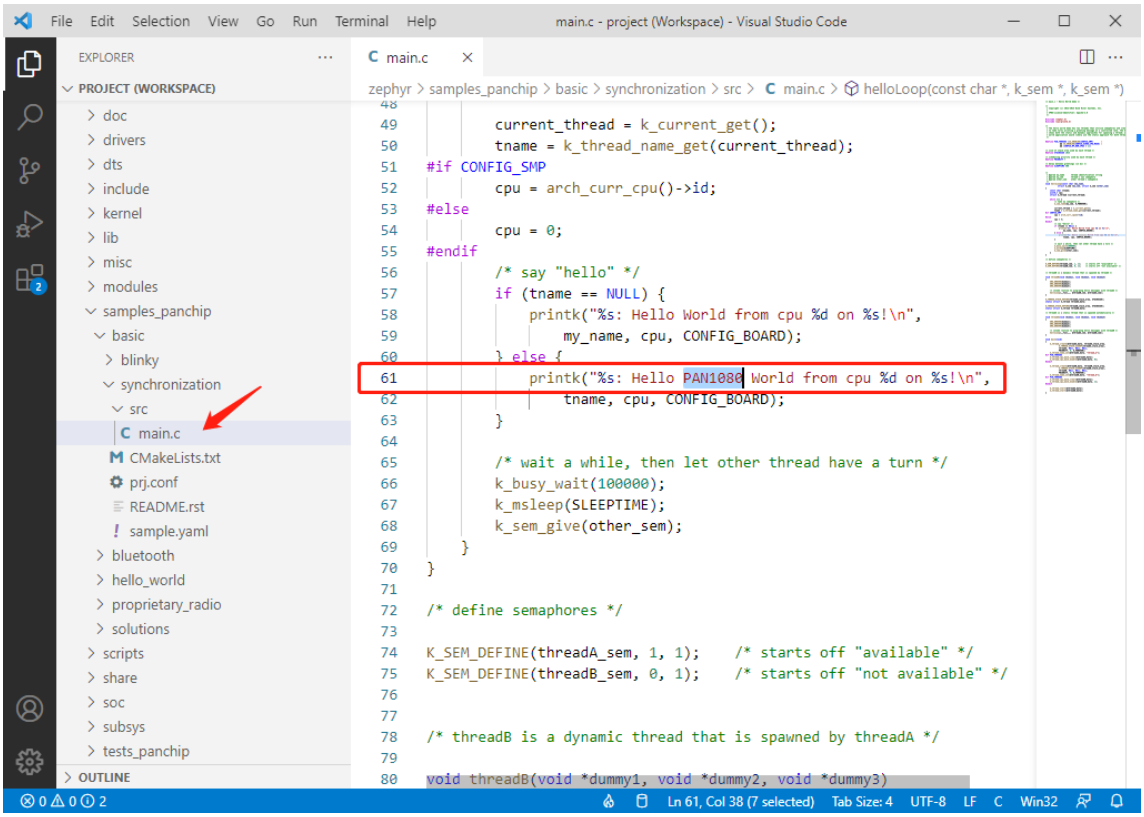
1. 前一阶段执行 `synchronization.bat` 脚本的 F (Flash) 命令并烧录成功后，会返回操作菜单，这时候再输入 o (Open) 命令，即可打开 IDE 环境 VS Code：



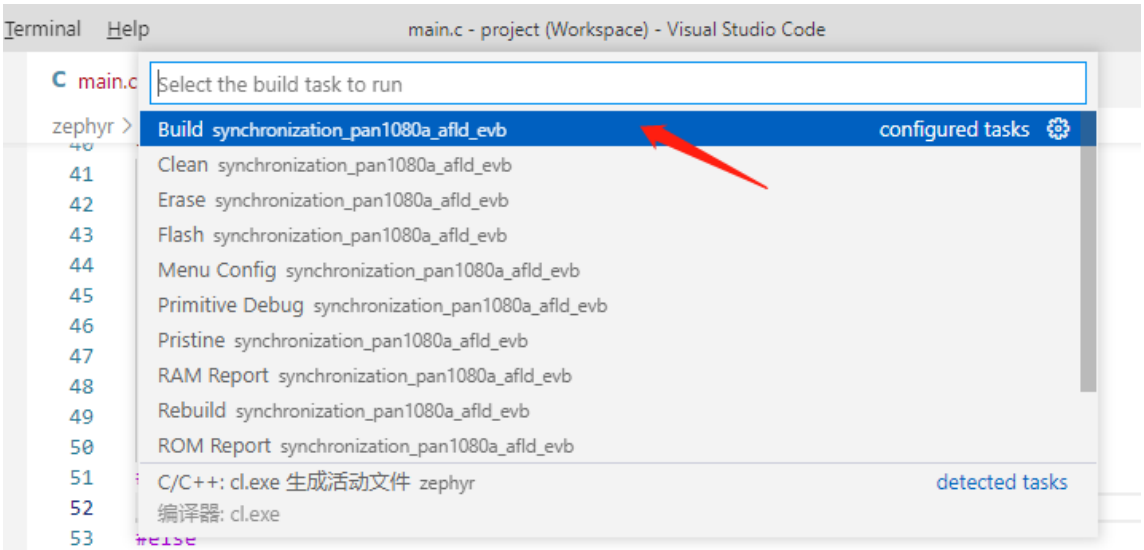
2. VS Code 在第一次打开时会自动做一些初始化配置工作，打开速度会慢一些；稍等片刻，即可看到 VS Code 界面；第一次打开某个工程文件，会弹出信任对话框，点击 “Yest, I trust the authors” 按钮，进入主界面：



3. 在左侧 PROJECT 侧边栏中，找到并打开 `zephyr/samples_panchip/basic/synchronization/src/main.c` 文件，修改第 61 行的打印消息（例如将 Hello World from…修改为 Hello PAN1080 World from…）：



4. 修改完成后，点击 `Terminal` 菜单，选择 `Run Build Task...`（或直接快捷键 `Ctrl + Shift + B`）：



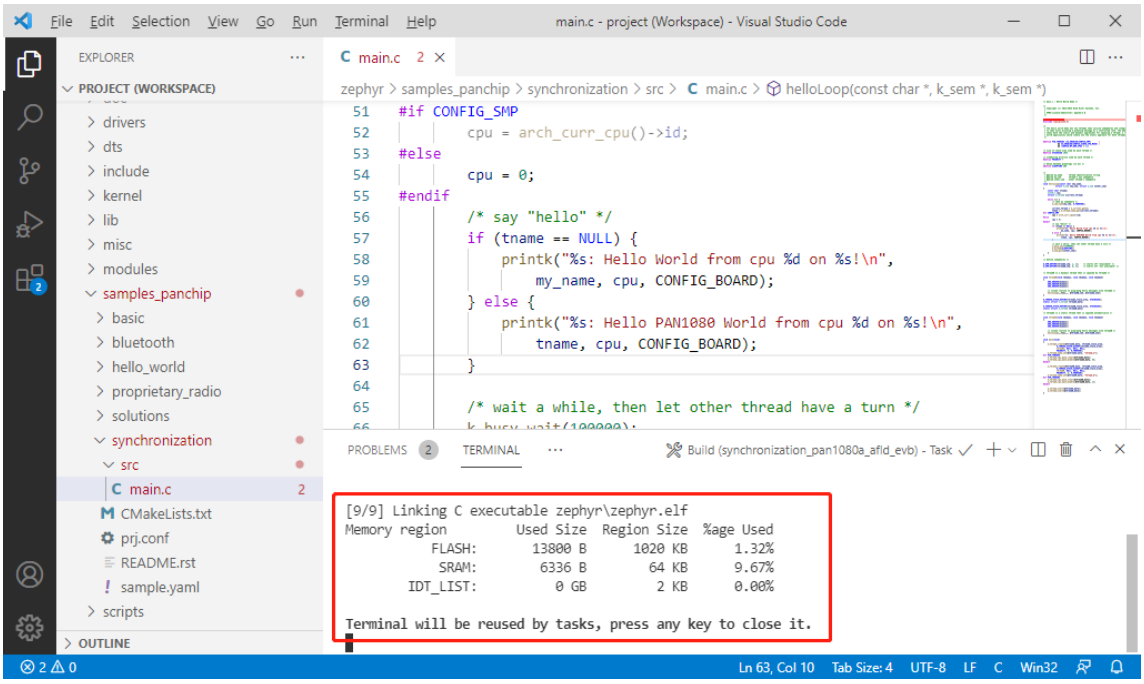
各指令含义如下：

1. `Build`：编译/增量编译
2. `Rebuild`：重新编译，效果相当于先执行 `Pristine`，再执行 `Build`
3. `Clean`：删除 Build Phase 下生成的文件（如.obj/.elf/.hex 等），但保留 Configuration Phase 下生成的文件（如.config 文件等）

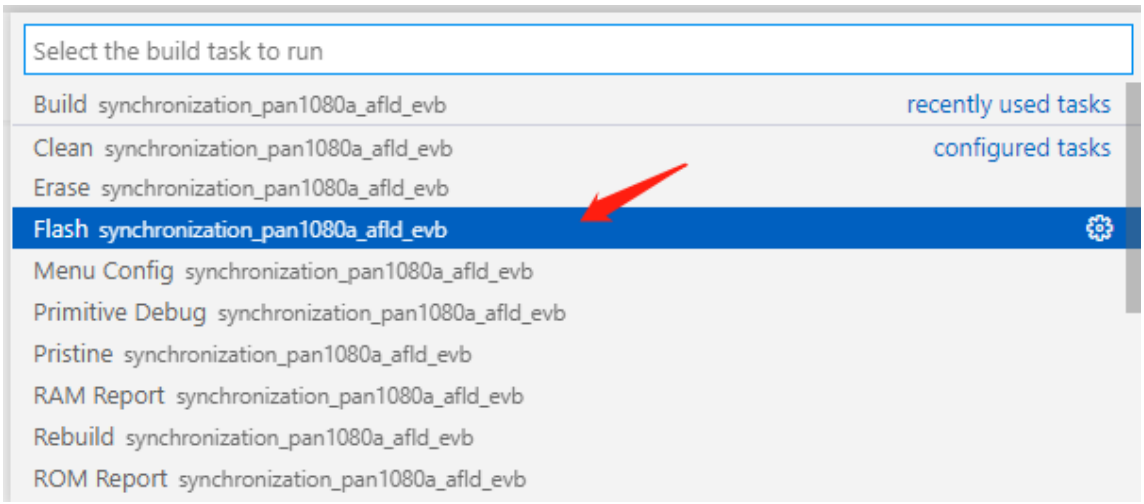
   注：关于 Configuration Phase 和 Build Phase 的概念，请参考 Zephyr 官方文档：Build Overview
4. `Pristine`：删除所有 Zephyr Build System 生成的文件
5. `Flash`：烧录 Image

6. `Erase`：擦除整块 Flash（Jlink 方式）
7. `Menu Config`：在新的 CMD 窗口下打开 Zephyr menuconfig 配置菜单
8. `ROM Report`：生成 ROM 使用情况报告
9. `RAM Report`：生成 RAM 使用情况报告
10. `Primitive Debug`：在新的 CMD 窗口下打开原始的 JLink-GDB 调试界面

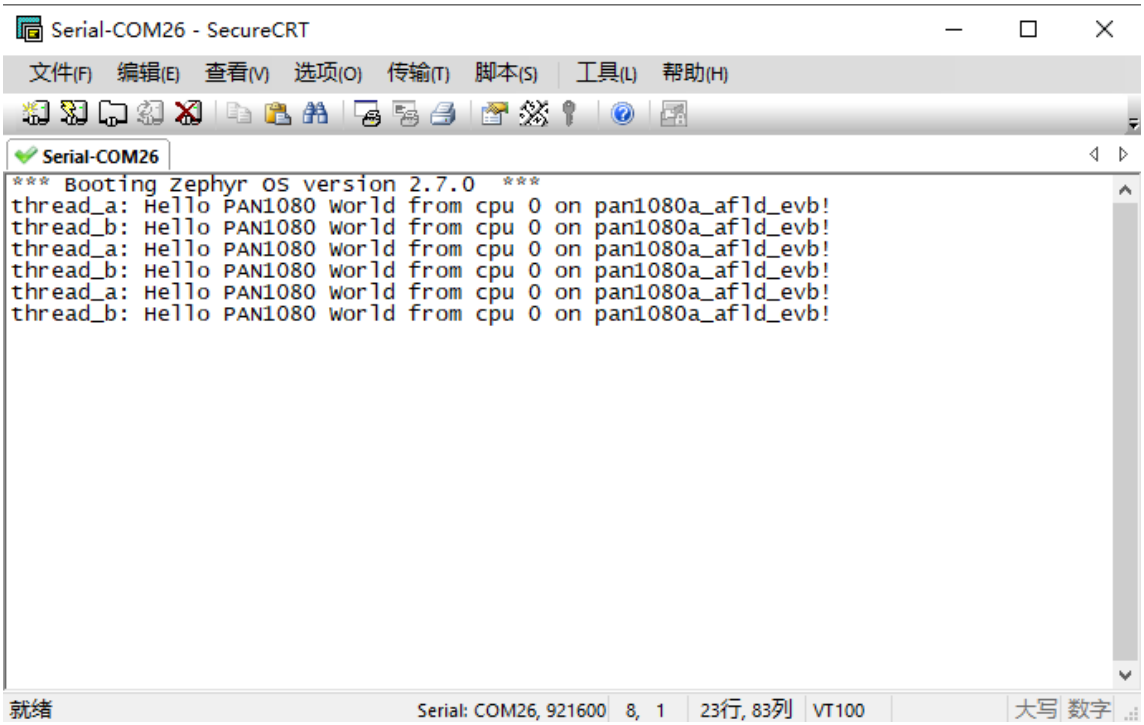
5. 在弹出的 `Build Task` 菜单中，执行 Build，增量编译修改后的程序；从 VS Code 的底部的 TERMINAL 界面中可以看到当前的编译状态：



6. 编译成功后，再次点击 `Terminal` 菜单，选择 `Run Build Task...`（或直接快捷键 `Ctrl + Shift + B`），在弹出的 `Build Task` 菜单中，执行 Flash，将修改后的程序烧录至开发板：

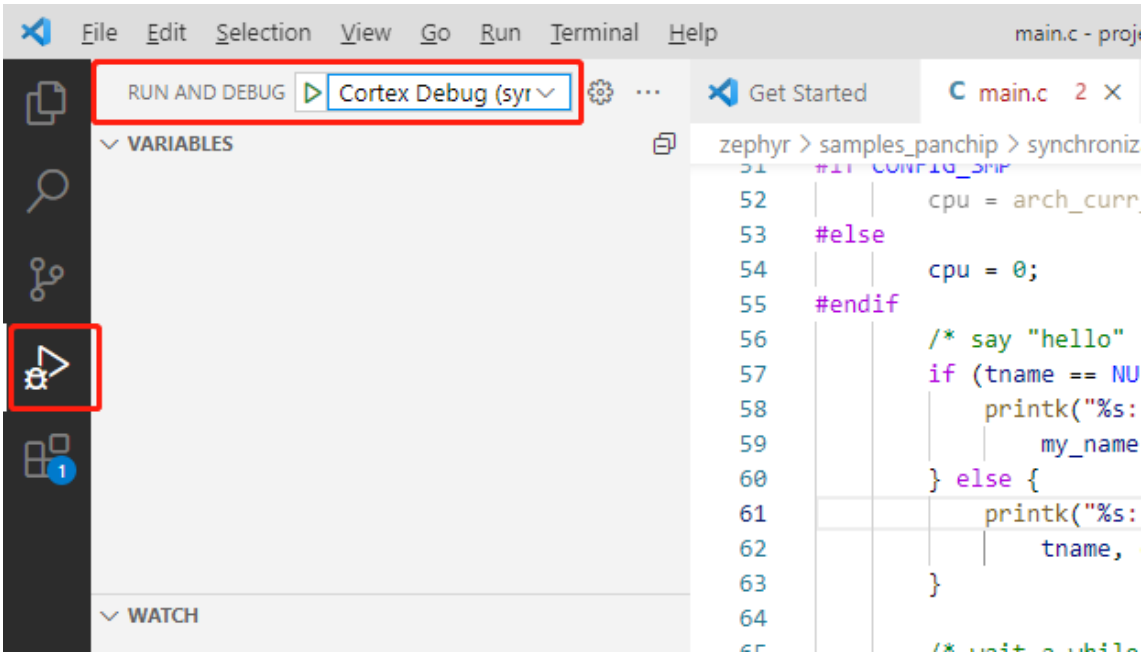


7. 烧录成功后，查看串口输出，可以看到刚才的修改已经生效：

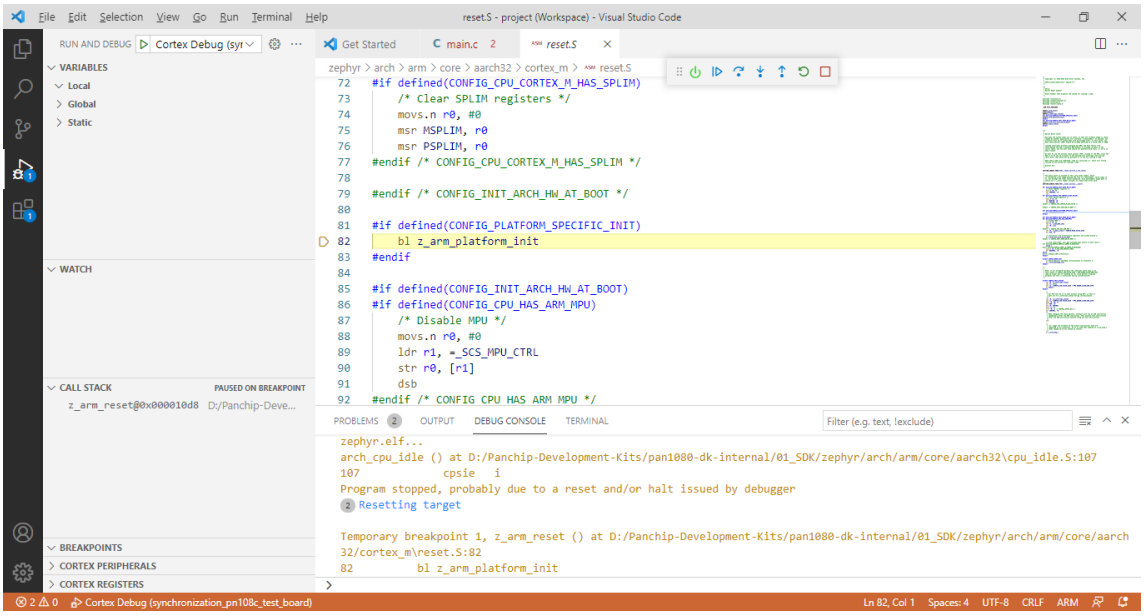


3.5 使用 VS Code 调试程序

1. PAN1080 SDK 支持直接在 VS Code 中调试程序
    1. 在 VS Code 中可以使用一个名为 `Cortex-Debug` 的插件调试基于 ARM Cortex-M 的 SoC，PAN1080 基于 ARM Cortex-M0 设计，因此我们可以很方便地使用 VS Code 调试 PAN1080 的程序；
    2. `Cortex-Debug` 插件已经默认集成到 PAN1080 SDK Toolchain 中，无需另外安装；
2. 打开 VS Code 左侧的 Debug 侧边栏，即可看到左上角已经有配置好的 Debug 选项：



3. 点击绿色箭头（或按快捷键 `F5`），即可进入 Debug 界面：

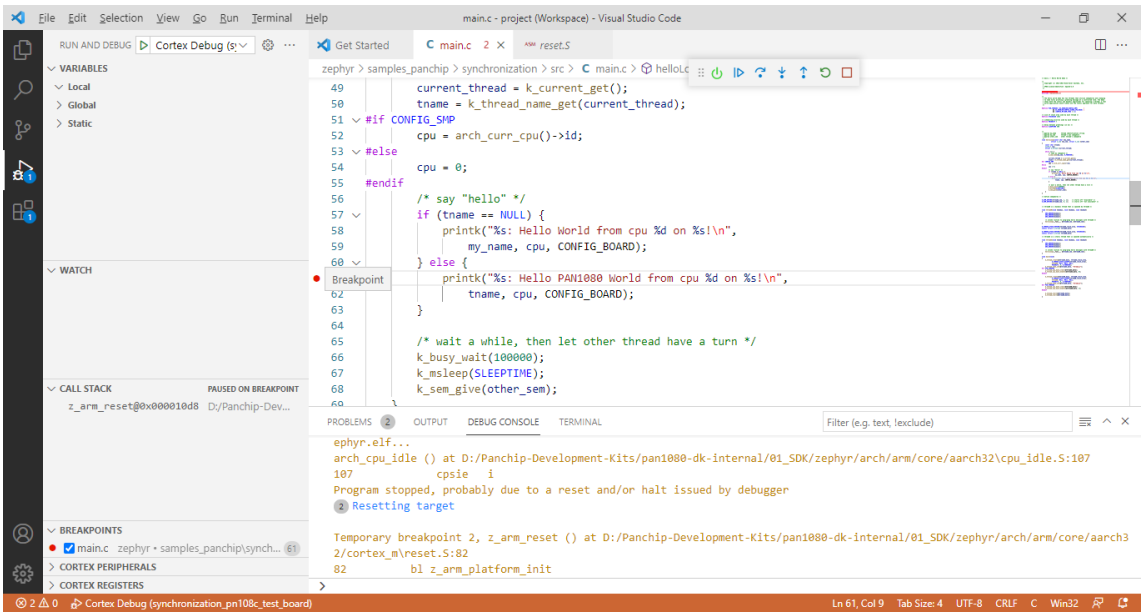


基于 Cortex-Debug 插件的 Debug 功能包括：

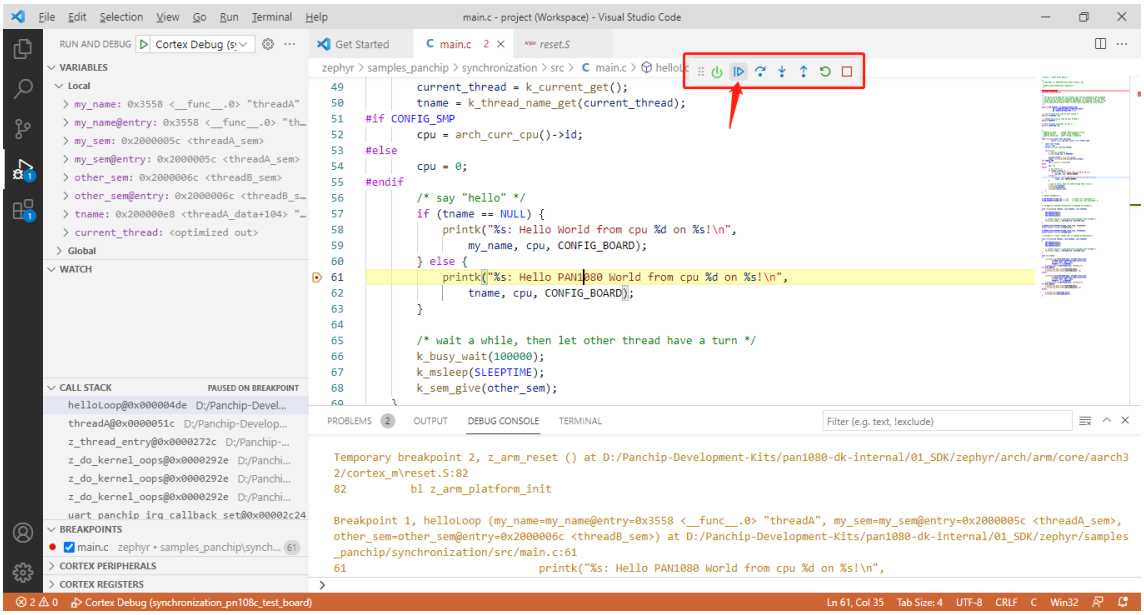
- 单步调试
- 添加断点
- 查看变量
- 查看函数调用关系
- 查看 Core Register
- 查看某函数的反汇编代码：先按快捷键 `Ctrl + Shift + P`，然后输入指令 `Cortex-Debug: View Disassembly (Function)`
- 查看某段 RAM 的内容：先按快捷键 `Ctrl + Shift + P`，然后输入指令 `Cortex-Debug: View Memory`

更多 Debug 功能介绍可以参考 VS Code Debug 官方文档，以及 Cortex Debug 插件文档。

4. 打开 `main.c`，单击第 61 行**行号左边的空白处**（鼠标指针变为手型），即可在此处加入断点（红色实心圆点）：



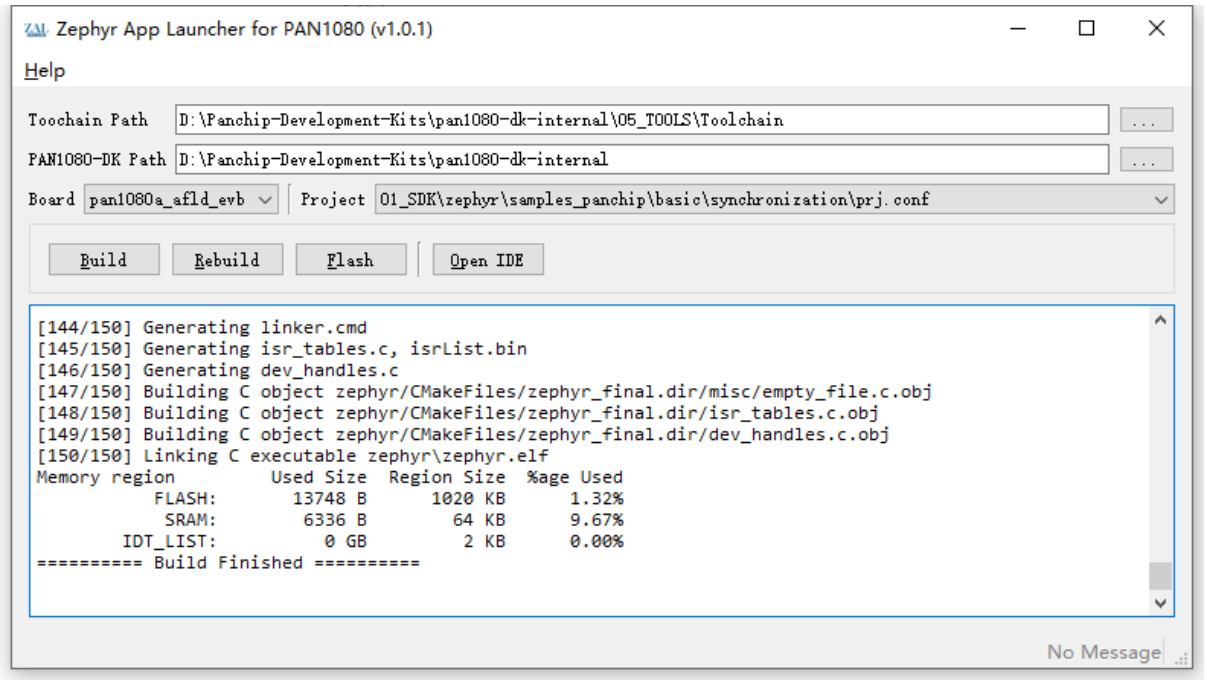
5. 点击 Debug 功能按钮区的 Continue 按钮（或按快捷键 F5），程序将执行到断点处并停止：



程序暂停后，可以在左侧侧边栏中，查看变量（VARIABLES 栏）、增加监视变量（WATCH 栏）、查看调用堆栈（CALL STACK 栏）、查看断点信息（BREAKPOINTS 栏）、查看外设寄存器信息（CORTEX PERIPHERALS 栏）或 CPU 寄存器信息（CORTEX REGISTERS 栏）等。

3.6 Zephyr App Luancher 工具

前面介绍了第一次编译 App 时，使用**快速编译脚本**进行编译、烧录、打开 VS Code 等操作；实际上，PAN1080-DK 还提供了一个名为 Zephyr App Launcher 的 Windows GUI 工具，可以代替上述这些操作：



使用方法请参考：Zephyr APP Launcher 工具介绍。

1.1.4 4 创建自己的 App 工程

请参考：

1. 快速上手 SoC App 开发
2. 快速上手 BLE App 开发

1.1.5 5 更多相关文档

下面这些文档有助于您进一步了解 PAN1080 SoC 开发的相关知识：

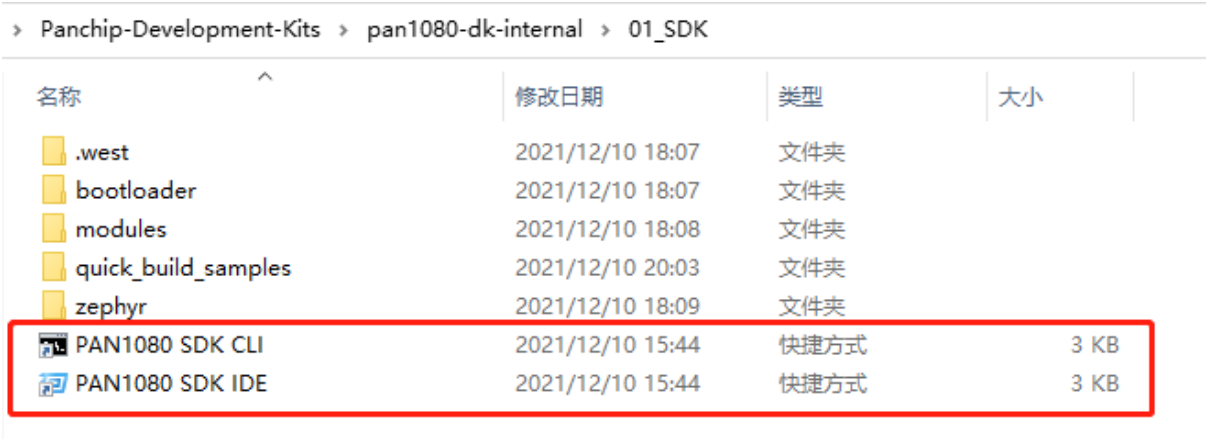
1. Zephyr 简介：简要介绍 Zephyr，以及基于 Zephyr 的 PAN1080 SDK 有哪些特性和优势
2. PAN1080 SDK 架构介绍：介绍 PAN1080 SDK 整体软件框架
3. SDK 开发环境介绍：介绍 PAN1080 SDK 的基本开发环境
4. 常见问题（FAQs）：介绍 PAN1080 SDK 使用过程中的常见问题及解答

1.2 SDK 开发环境介绍

PAN1080 SDK 为开发者提供了 2 种开发环境：

- 命令行方式（Command Line）
- 图形化界面方式（Zephyr App Launcher + VS Code）

打开 PAN1080 Development Kit 文件夹，进入 01_SDK 子目录，可以看到两种开发环境的快捷方式：



其中：

- PAN1080 SDK CLI 为命令行方式的开发环境入口
- PAN1080 SDK IDE 为图形化界面方式的开发环境入口

在 PAN1080 SDK 实际开发过程中，经常需要将两种方式结合起来使用，下面对 2 种开发环境分别做简单的介绍。

1.2.1 1 命令行方式（Command Line）

PAN1080 SDK 使用一个名为 `west` 的命令行工具来执行编译、烧录、调试等操作。

`west` 是一个 “Meta Tool”，在执行 `west` 支持的子命令过程中，不同的命令会调用不同的工具来执行具体的操作。例如：

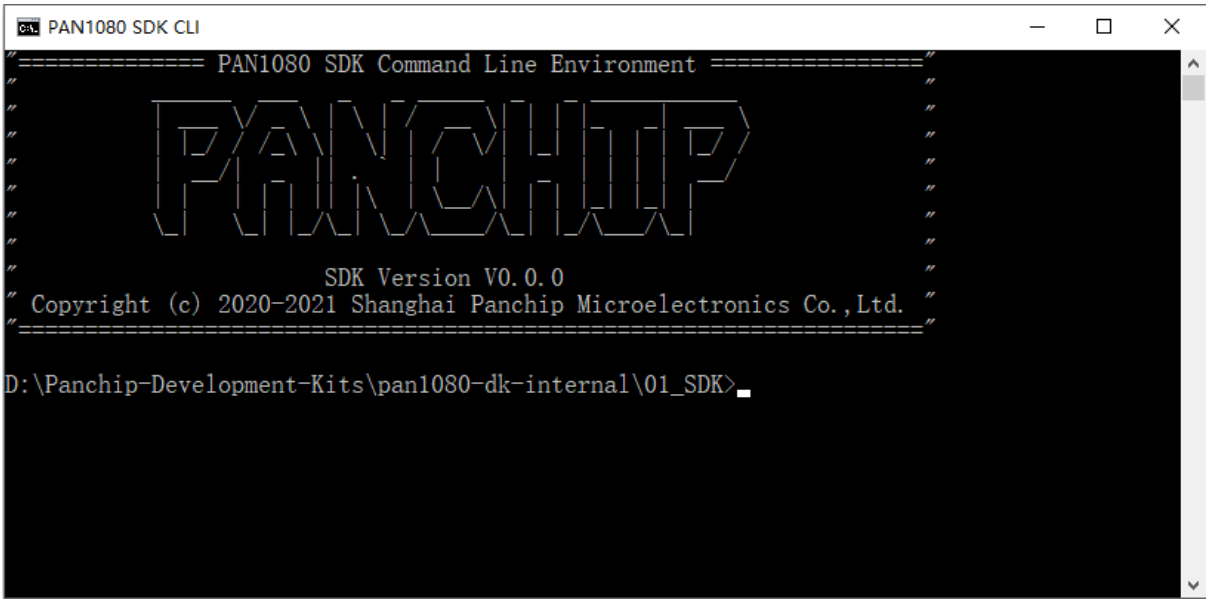
- 执行 `west build` 子命令后，`west` 会自动调用 CMake、ninja、dtc、Kconfig、gcc 等工具进行配置和编译；
- 执行 `west flash` 子命令后，`west` 会自动调用相关烧录工具（如 JLink）进行烧录；

- 执行 `west debug` 子命令后，`west` 会自动调用相关调试工具（如 JLink-GDB-Sever + ARM-Embedded-GDB）进行调试。

  实际上，PAN1080 SDK 基于开源实时操作系统 Zephyr Project 扩展而来，其基础的命令行工具 `west` 也是 Zephyr 提供的；关于此工具的更多信息，可以参考 zephyr 官方文档：West (Zephyr’s meta-tool)。

1.1 使用演示：编译、烧录、调试一个 Sample

1. 双击 PAN1080 SDK CLI 快捷方式，打开命令行界面：

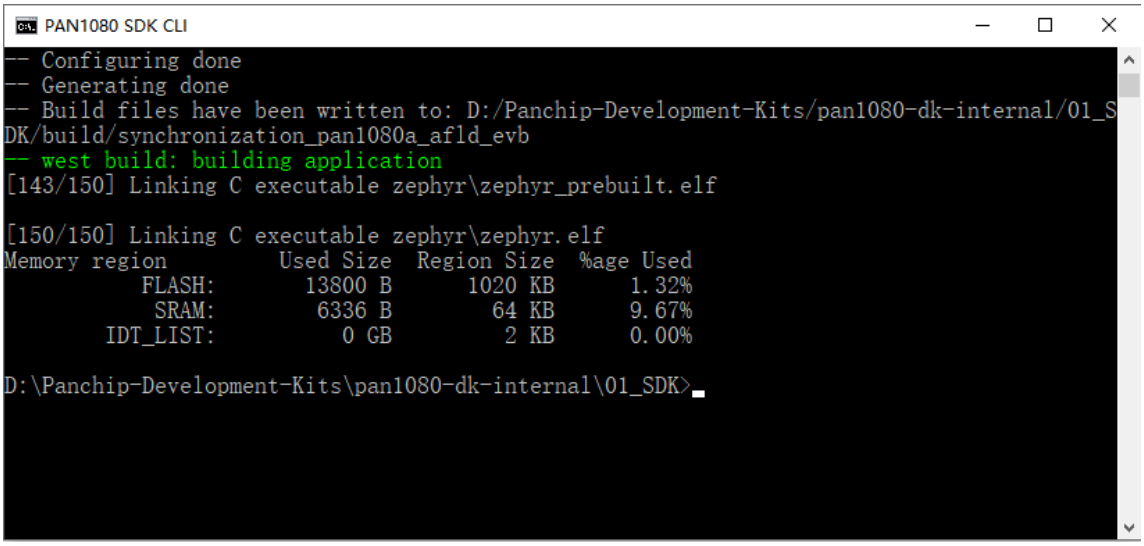


1. 使用 `west build` 命令，编译 Synchronization Sample：

```
west build -d build/synchronization_pan1080a_afld_evb -b pan1080a_afld_evb ./zephyr/
↪samples_panchip/basic/synchronization -p
```

参数解释：

- `build`：执行 build 子命令；
- `-d build/synchronization_pan1080a_afld_evb`：设置 build 输出文件目录为 build/synchronization_pan1080a_afld_evb，所有编译过程中生成的文件均存此目录中，目录名可以任意指定；另外，此参数可以省略，若省略则等价于`-d build`；
- `-b pan1080a_afld_evb`：将 board 配置为 PAN1080A-AFLD EVB，于是编译工具链会自动从 `zephyr/boards/arm/pan1080a_afld_evb` 目录下搜索需要的 board 配置文件；
- `./zephyr/samples_panchip/basic/synchronization`：程序源码目录，此处为演示多线程同步的例程 synchronization；
- `-p`：重新编译（Rebuild）；如果`-d` 参数指定的目录中存在之前编译的信息，则：
  - 若不指定`-p` 参数，则 `west` 执行默认的**增量编译**操作；
  - 若指定`-p` 参数，则 `west` 执行**重新编译**操作；



2. 使用 `west flash` 命令，烧录 Synchronization Sample：

```
west flash -d build/synchronization_pan1080a_afld_evb -r jlink --reset-after-load
```

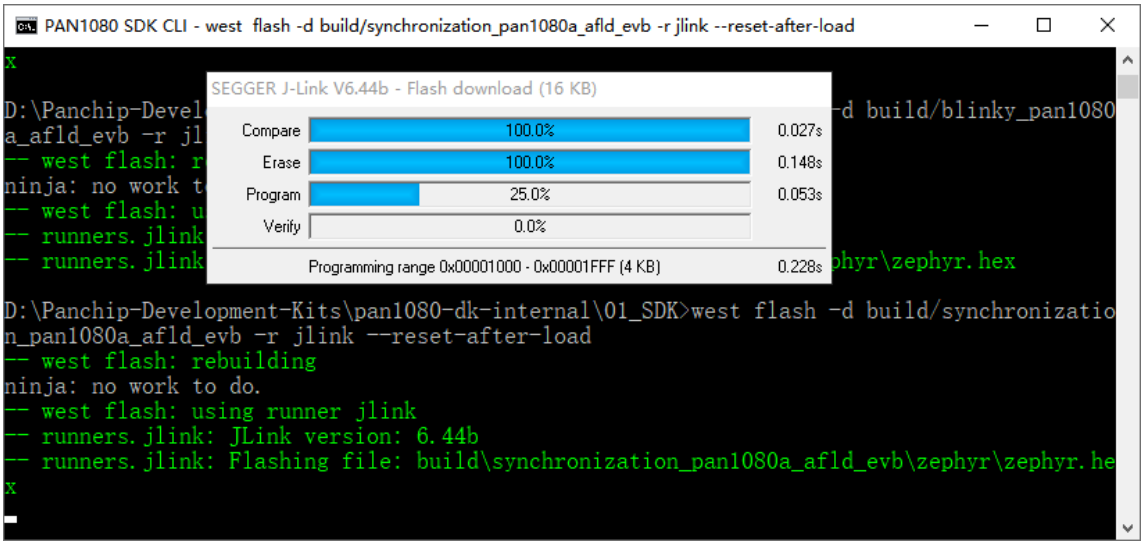
参数解释：

- `flash`：执行**烧录**子命令；
- `-d build/synchronization_pan1080a_afld_evb`：在目录 build/synchronization_pan1080a_afld_evb 内查找待烧录文件；
- `-r jlink`：将烧录工具指定为 JLink，于是 west 将会调用 `JLink.exe` 进行 image 烧录；
- `--reset-after-load`：指定此参数后，目标板将会在烧录完成后自动 Reset 并执行烧录后的程序；

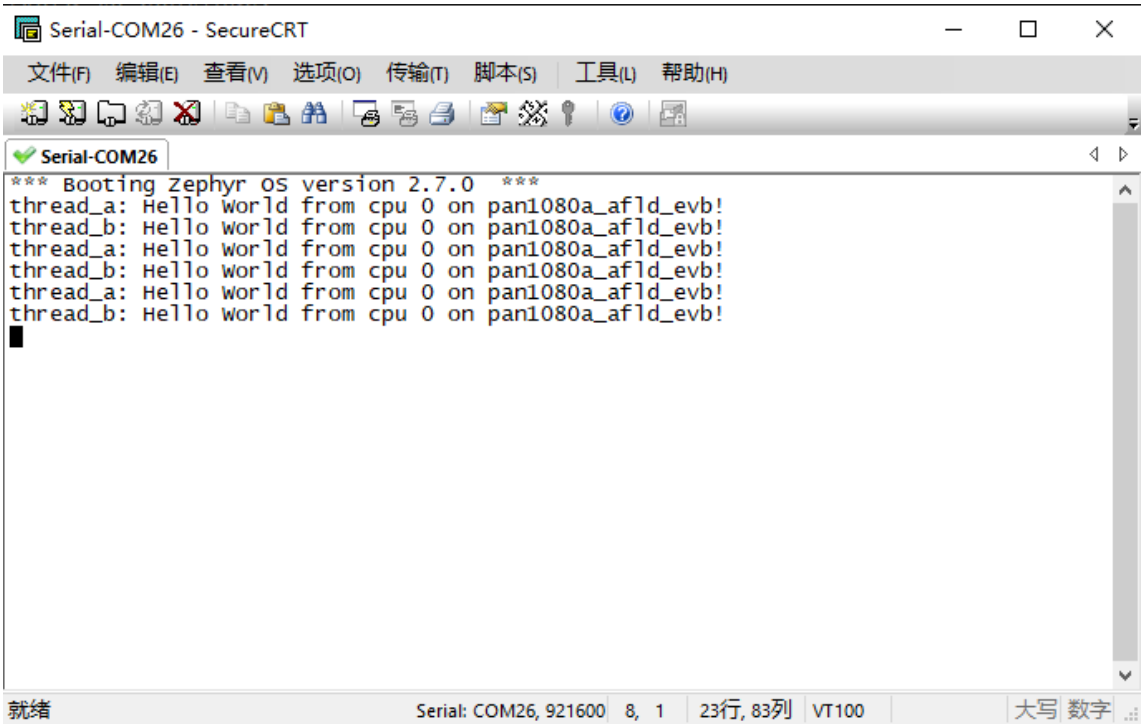
注：程序烧录执行前，请确认已经将 PN108C 测试板的：

- SWD（P46：SWD_CLK；P47：SWD_DAT）接口通过 JLink 连接至 PC
- UART（P30：Tx；P31：Rx；Baudrate：921600）接口通过 USB 转串口模块连接至 PC

1. 烧录过程：



1. 程序执行：



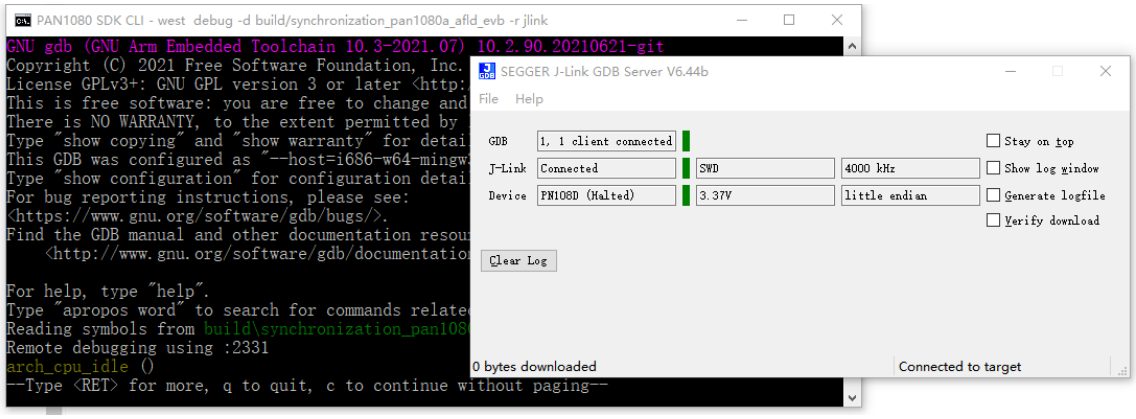
3. 使用 `west debug` 命令，调试 Synchronization Sample：

```
west debug -d build/synchronization_pan1080a_afld_evb -r jlink
```

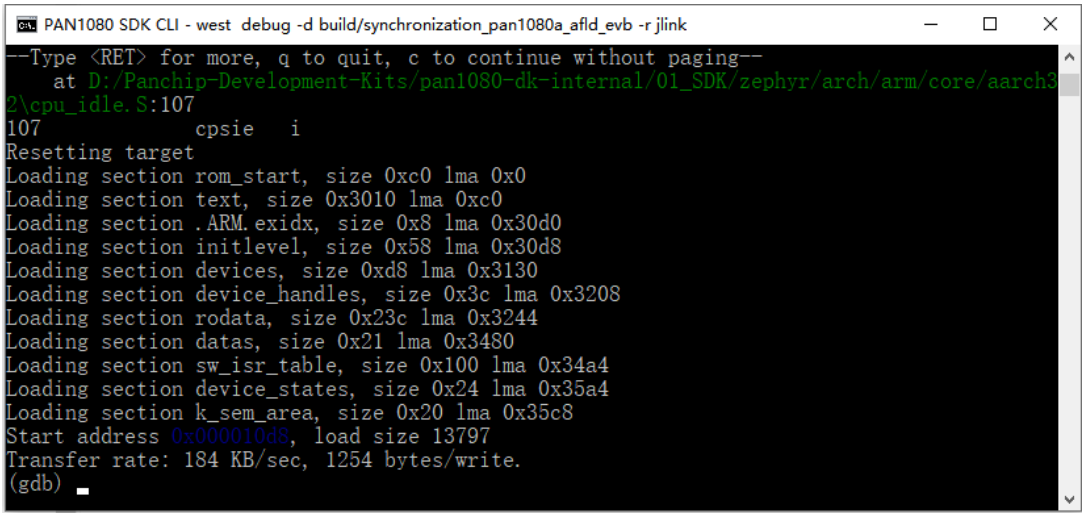
参数解释：

- `debug`：执行**调试**子命令；
- `-d build/synchronization_pan1080a_afld_evb`：在目录 build/synchronization_pan1080a_afld_evb 内查找待调试文件；
- `-r jlink`：将调试工具指定为 JLink + GDB，于是 west 将会调用 `JLinkGDBServer.exe` + `arm-none-eabi-gdb.exe` 进行调试；

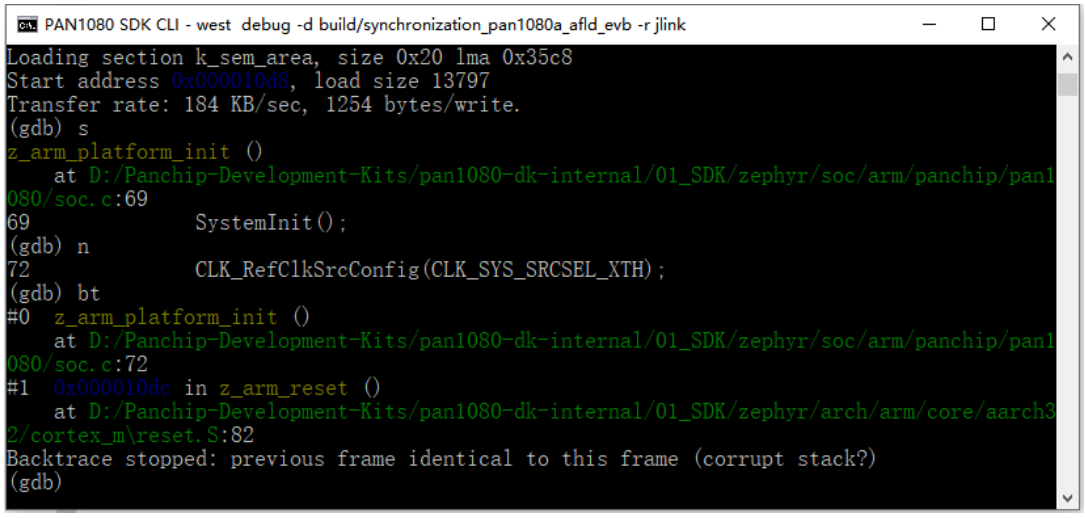
1. 启动 JLink GDB Server，并进入调试准备界面：



1. 将 GDB Server 界面最小化（请勿关闭），并在命令行界面按回车键，进入调试命令等待界面：



2. 输入 GDB 调试命令，开始调试：



调试支持通用的 GDB **调试指令**：

- `s`：单步，Step In
- `n`：单步，Step Over
- `b <func_name>`：设置断点，Breakpoint
- `d break <index>`：删除断点，Delete Breakpoint
- `c`：继续执行，Continue
- `<Ctrl+C>`：暂停执行，Halt
- `p <param_name>`：打印变量值，Print
- `bt`：查看调用堆栈，Backtrace

更多的 GDB 命令行调试介绍请查阅：

- GDB 官方文档：https://sourceware.org/gdb/current/onlinedocs/gdb/
- JLink GDB Server 官方文档：https://www.segger.com/downloads/jlink/UM08001, Chapter 4

1.2 其他常用命令

1. 某个曾经编译过的 Sample（如 Synchronization Sample）代码有改动，对其执行增量编译：

```
west build -d build/synchronization_pan1080a_afld_evb
```

> **注**：某些情况下，使用此方式可能无法自动识别到改动的文件；例如，若改动到了某个 Kconig 的默认配置（如某个 `Kconfg.defconfig` 文件），则执行增量编译是无法检测到此改动的，此时需要使用`-p` 参数对目标做 Rebuild。

2. 编译时，通过传入 `CONF_FILE` 参数，载入某个 Sample（如 BLE Peripheral HeartRate Sample）的特殊配置：

```
west build -d build/bluetooth_peripheral_hr_minimal_pan1080a_afld_evb -b pan1080a_afld_
↪evb ./zephyr/samples_panchip/bluetooth/peripheral_hr -- -DCONF_FILE="prj_minimal.conf"
```

> **注**：若不指定 `CONF_FILE` 参数，则 `west build` 过程中，会应用**默认的配置文件**`"prj.conf"`；若明确指定 `CONF_FILE` 参数，则 `west build` 过程中，会应用**新指定的配置文件**。

3. 编译时，通过传入 `OVERLAY_CONFIG` 参数，覆盖某个 Sample（如 Multimode Mouse Sample）的默认配置：

```
west build -d build/solutions_mouse_ble_usb_pan1080a_afld_evb -b pan1080a_afld_evb ./
↪zephyr/samples_panchip/solutions/model_mouse -- -DOVERLAY_CONFIG="ble_usb_flow.conf"
```

> **注**：若不指定 `OVERLAY_CONFIG` 参数，则 `west build` 过程中，会应用默认的配置文件`"prj.conf"`；若指定 `OVERLAY_CONFIG` 参数，则 `west build` 过程中，会先应用默认的配置文件`"prj.conf"`，然后再应用指定的 Overlay 配置文件，如果 Overlay 配置文件中存在与默认配置文件中相同的 Config 选项，则覆盖掉默认配置文件中的配置。

4. 部分清理（Clean），删除 Build Phase 下生成的文件（如.obj/.elf/.hex 等），但保留 Configuration Phase 下生成的文件（如.config 文件等）：

```
west build -d build/synchronization_pan1080a_afld_evb -t clean
```

> 关于 Build Phase 与 Configuration Phase 的含义，请参阅 Zephyr 官方文档：Build Overview。

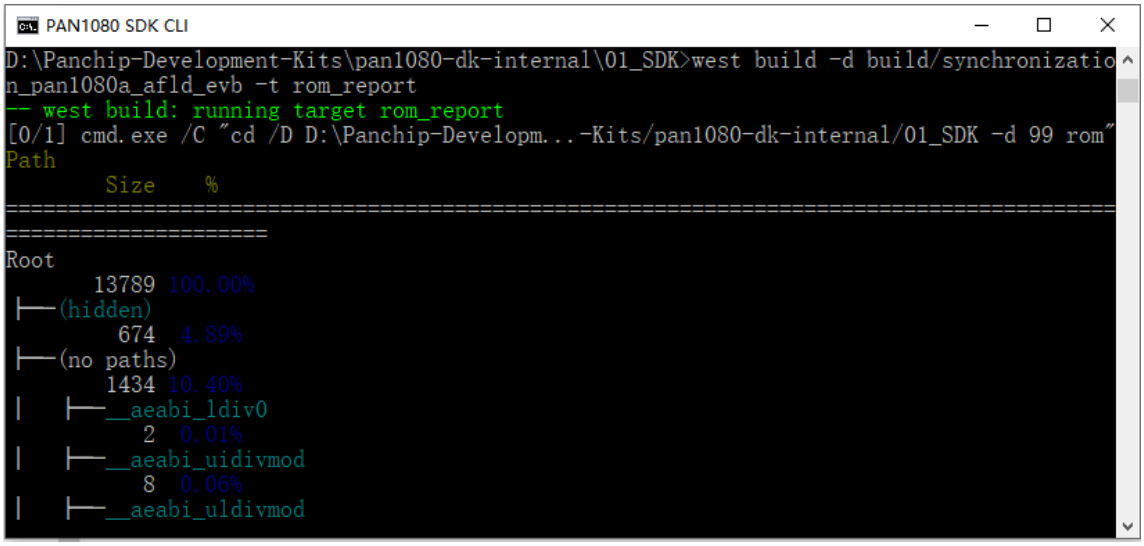
5. 完整清理（Pristine），同时删除 Build Phase 与 Configuration Phase 下生成的文件：

```
west build -d build/synchronization_pan1080a_afld_evb -t pristine
```

6. ROM 报告（ROM Report），列出 ROM 空间占用信息：

```
west build -d build/synchronization_pan1080a_afld_evb -t rom_report
```

执行结果：

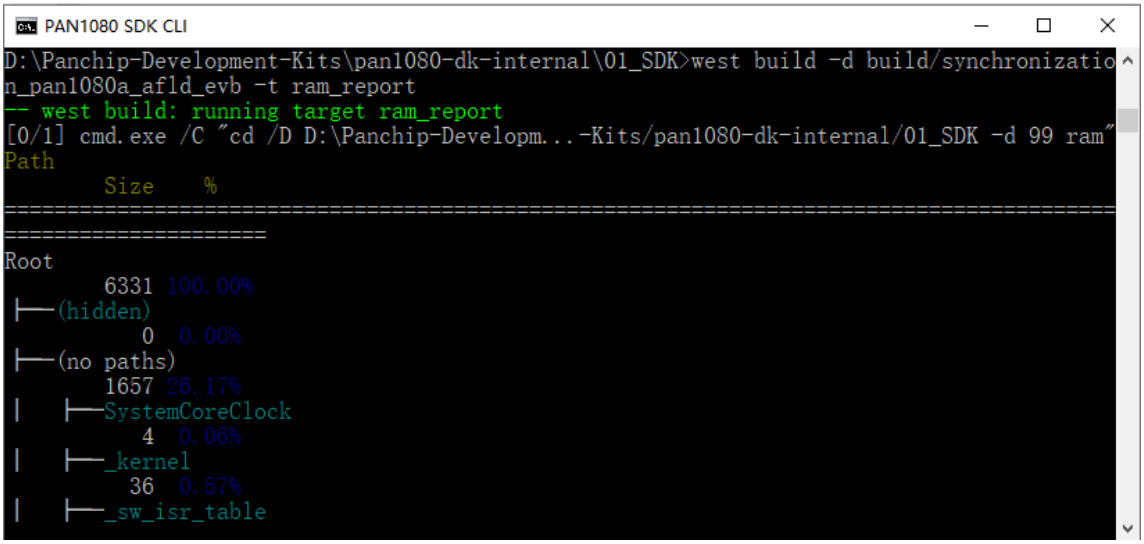


注：ROM Report 中统计结果与编译后的 Image Size 并不等价，如果需要精确了解编译后的 Image 大小，请至 Map 文件（zephyr.map）中查看。

7. RAM 报告（RAM Report），列出 RAM 空间占用信息：

```
west build -d build/synchronization_pan1080a_afld_evb -t ram_report
```

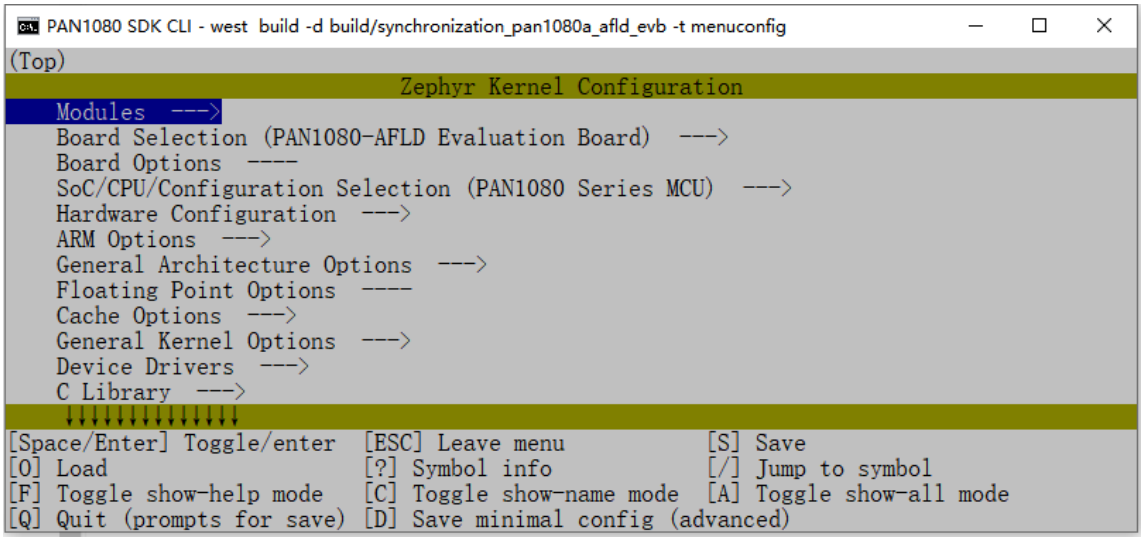
执行结果：



8. 打开 Menu Config 菜单：

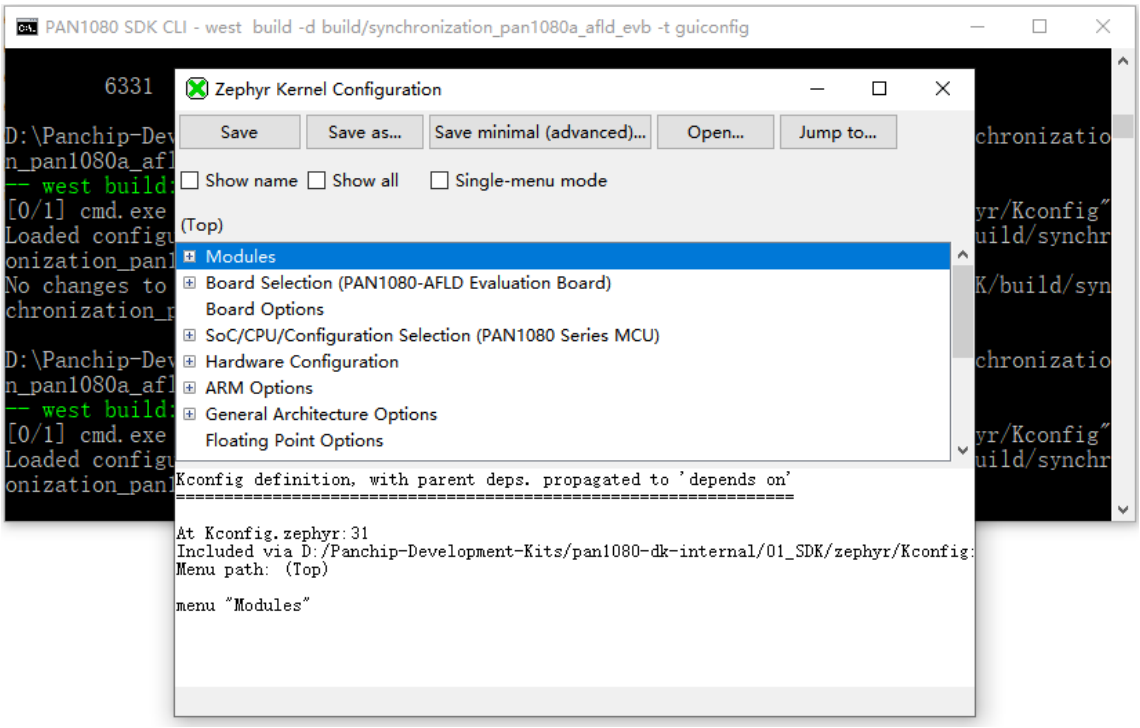
```
west build -d build/synchronization_pan1080a_afld_evb -t menuconfig
```

执行结果：



9. 打开 GUI Config 菜单：

```
west build -d build/synchronization_pan1080a_afld_evb -t guiconfig
```



1.2.2 2 图形化界面方式（VS Code）

PAN1080 SDK 使用 VS Code 作为图形化集成开发环境（IDE）。

VS Code 是一个轻量级的代码编辑器，其优秀的可扩展性使得 PAN1080 SDK 有机会将 zephyr 的 west 命令行编译环境迁移到图形界面环境中来：

- 利用 VS Code 的构建任务特性，PAN1080 SDK 将常用的 `west` 命令集成到 Build Tasks 中；
- 利用 VS Code 的插件特性，PAN1080 SDK 将 `Cortex-Debug` 插件集成进来并进行适配，使其可以调试 zephyr 编译生成的 Image；

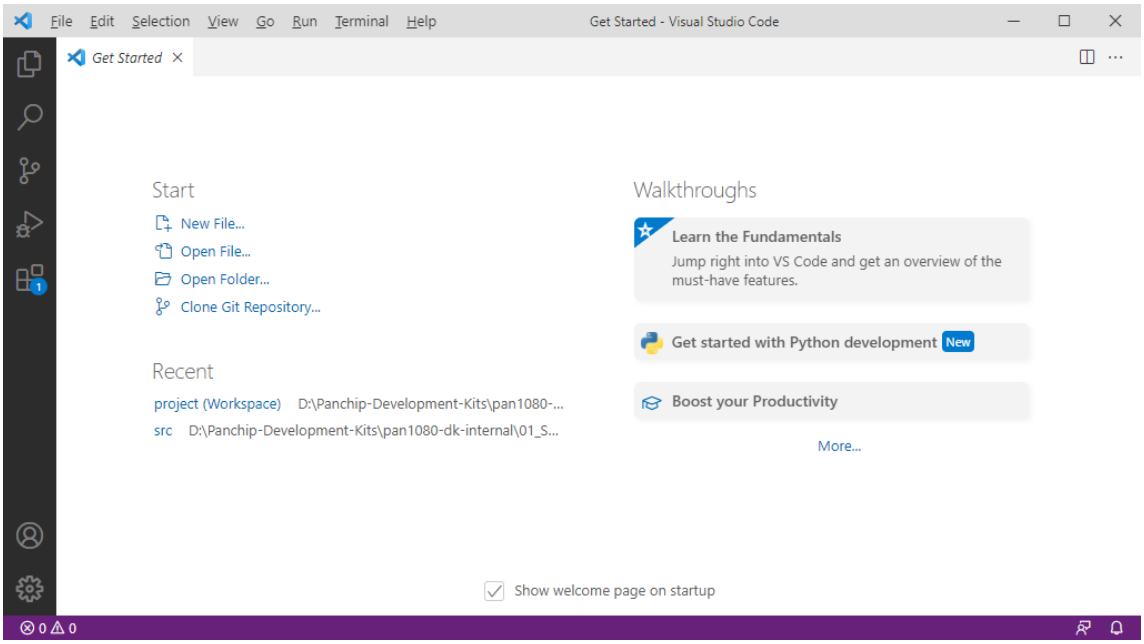
2.1 使用 VS Code 打开 Sample 工程

1. 先双击 PAN1080 SDK CLI 快捷方式，打开命令行界面，使用命令行方式编译一次例程（如编译 Synchronization Sample）：

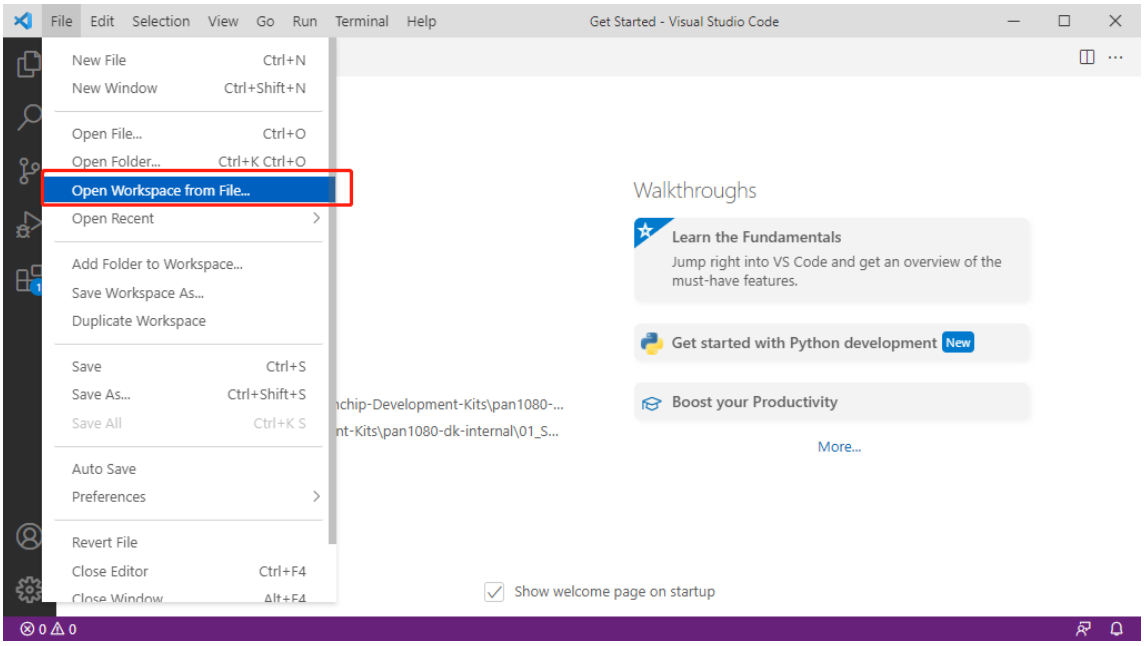
```
west build -d build/synchronization_pan1080a_afld_evb -b pan1080a_afld_evb ./zephyr/
↪samples_panchip/basic/synchronization
```

> **注**：为使 VS Code 可以正确编译和调试 PAN1080 程序，在使用 VS Code 打开某个 App 或 Sample 工程前，必须先用命令行方式编译一次工程，使其生成 VS Code 工程文件 `project.code-workspace`。

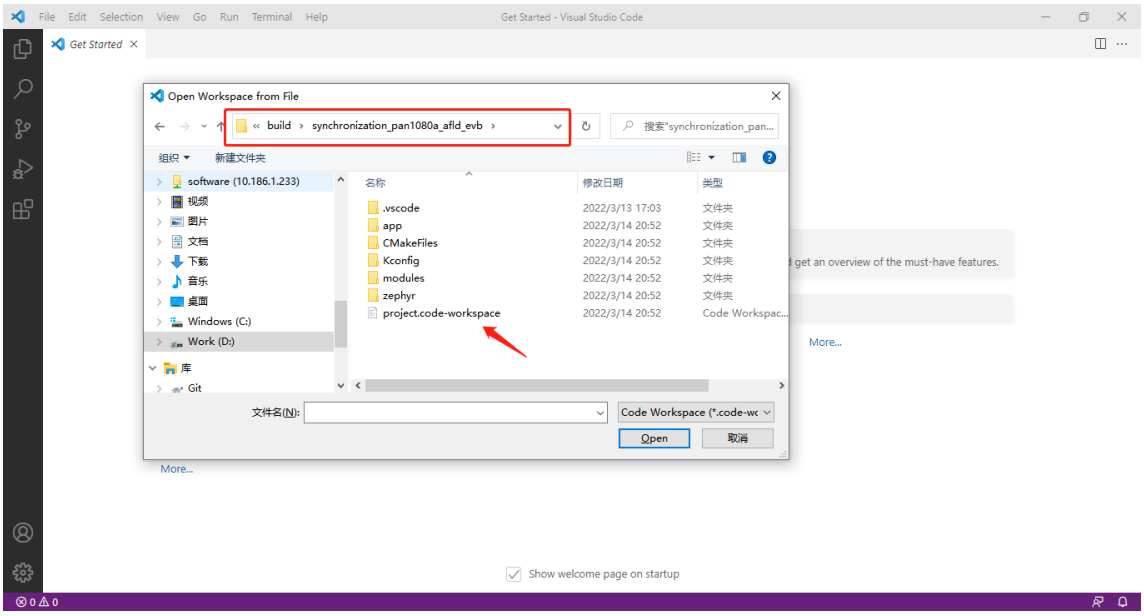
2. 再双击 PAN1080 SDK IDE 快捷方式，打开 VS Code 界面：



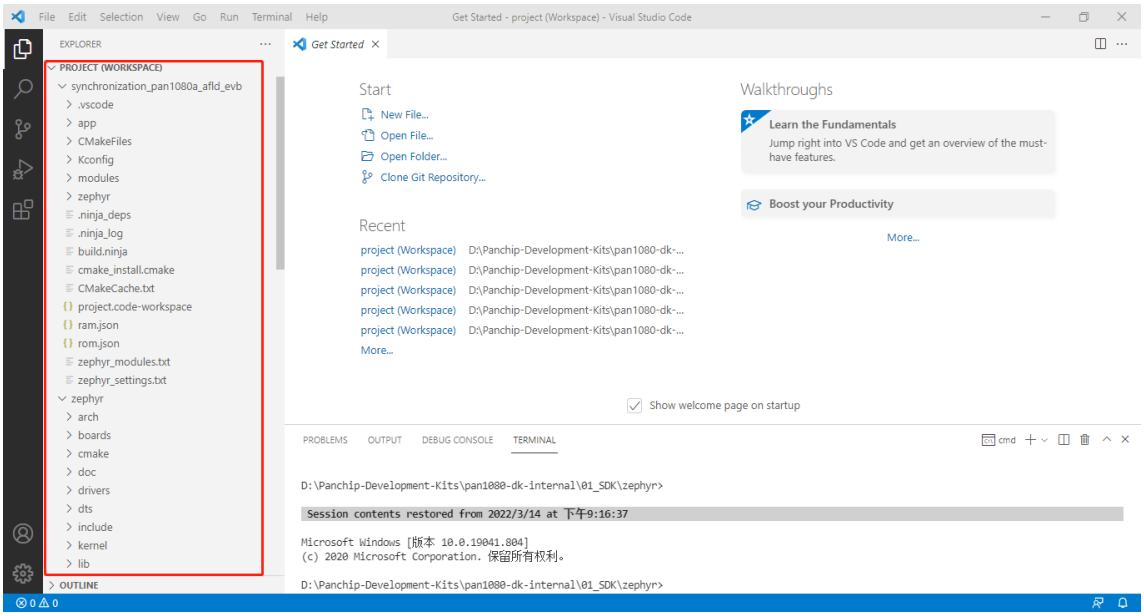
3. 点击菜单栏 `File` 菜单，选择 `Open Workspace from File...`：



4. 在弹出的对话框中，进入第一步 build 后生成的目录路径（`01_SDK/build/synchronization_pan1080a_afld_evb`），找到 project.code-workspace'文件，将其打开：



5. 选择 `project.code-workspace` 文件，点击 `Open` 按钮，打开工程：



6. 成功打开工程后，请参照文档PAN1080 SDK 快速入门中 VS Code 使用的相关介绍，编译、烧录、调试程序。

2.2 其他说明

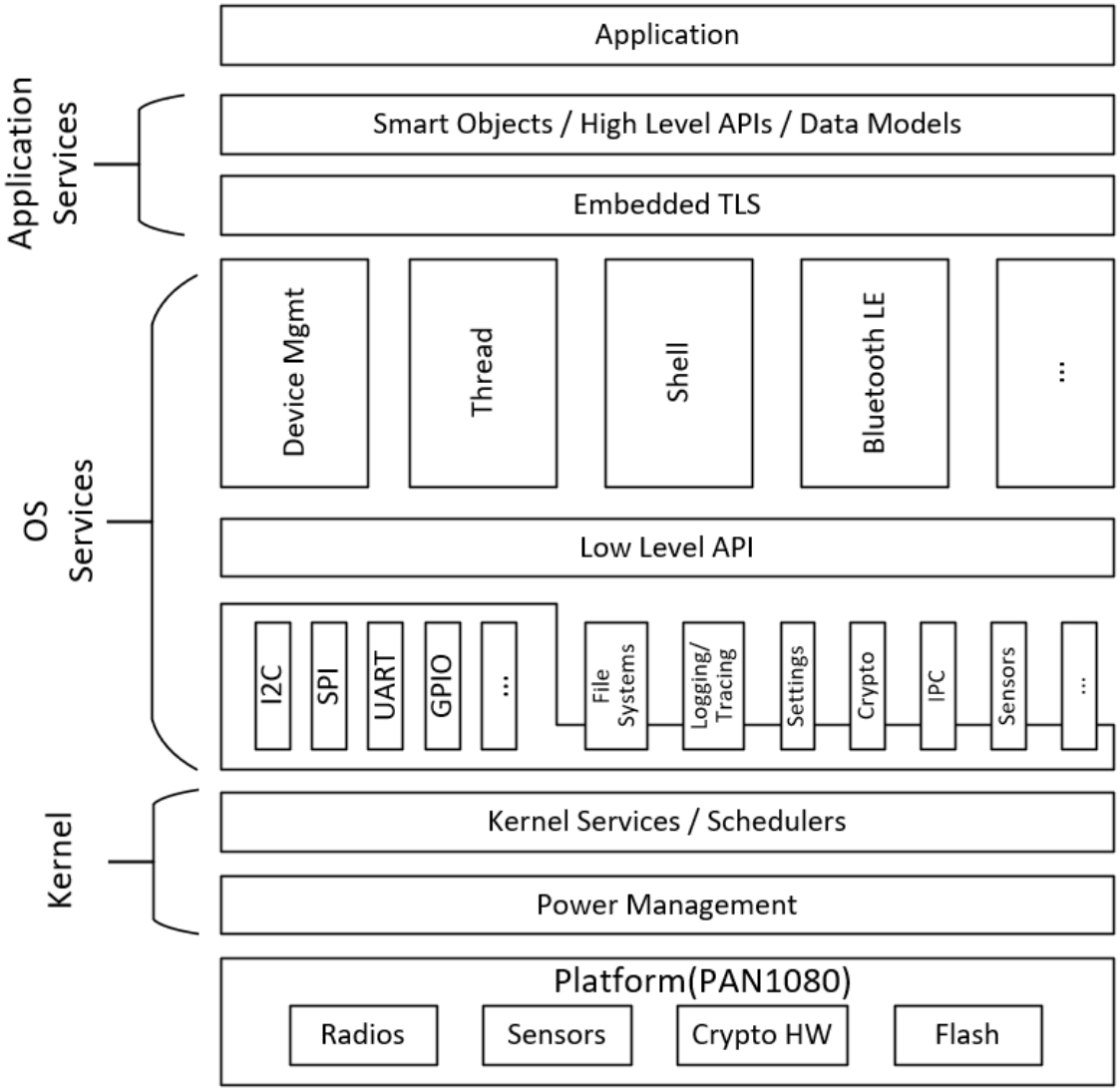
1. 对于 PAN1080 SDK 中自带的例程来说，也可以通过执行 `01_SDK/sdk_quick_build_samples` 目录中例程脚本的方式，打开 VS Code，具体请参考文档PAN1080 SDK 快速入门中的相关步骤介绍。
2. 请注意，目前 PAN1080 SDK 只支持使用 SDK Toolchain 中自带的 VS Code 进行编译和调试；如果您的 PC 上有自己的 VS Code，是无法直接用于编译调试 PAN1080 SDK 程序的。

1.3 SDK 整体框架介绍

1.3.1 1 简介

PAN1080 SDK 基于开源实时操作系统 Zephyr OS 修改而来（目前使用的 Zephyr 内核版本为 `v2.7.0-lts`）。

Zephyr OS 是一个内存占用极低的内核，它主要设计用于资源受限系统：从简单的嵌入式环境传感器、LED 可穿戴设备，到复杂的智能手表、IoT 无线网关。有关 Zephyr 的详细说明，请参考官方文档：Zephyr Project Documentation。



为了方便开发，我们对 Zephyr 进行了裁剪，只保留了 arm 架构，去掉了 PAN1080 不支持的子系统，添加了 PAN1080 BLE Controller 等，并适配了 VS Code。

PAN1080 SDK 源码树结构如下：

```
<home>/01_SDK
 .west
 bootloader
 build
 modules
 sdk_quick_build_samples
 zephyr
 PAN1080 SDK CLI
```

- `.west`：west 配置文件。

- `bootloader`：bootloader 程序源码，包括 `mcuboot` 和 `panboot`。
- `build`：编译时自动生成，包含 VS Code 的`.code-workspace` 文件及 zephyr 子目录下载烧录相关的`.bin`、`.elf`、`.hex` 文件。
- `modules`：必要的一些模组，包括：PAN1080 外设库，Controller 库等。
- `quick_build_samples`：编译脚本，可以快速编译和下载例程。
- `zephyr`：zephyr 主文件夹，包括：zephyr os，板级适配，例程。
- `PAN1080 SDK CLI`：快捷方式，用以启动 PAN1080 SDK 命令行终端。

1.3.2 2 sdk_quick_build_samples

PAN1080 SDK 快速编译脚本树结构如下：

```
<home>/01_SDK/quick_build_samples
 basic
 bluetooth
 drivers
 proprietary_radio
 solutions
 _cmd
```

- `basic`：zephyr 基础例程快速编译下载脚本。
- `bluetooth`：pan1080 bluetooth 快速编译下载脚本。
- `drivers`：pan1080 适配 zephyr 的 driver 快速编译下载脚本。
- `proprietary_radio`：pan1080 2.4G 例程快速编译下载脚本。
- `solutions`：pan1080 应用方案快速编译下载脚本。
- `_cmd`：脚本核心逻辑源文件。

例程有关源码的更多内容请参考 1.2.4 章节。

有关例程的更多内容请参考：PAN1080 例程介绍。

1.3.3 3 zephyr

PAN1080 SDK zephyr 源码树结构如下：

```
<home>/01_SDK/zephyr
 arch
 boards
 cmake
 doc
 drivers
 dts
 include
 kernel
 lib
 misc
 modules
 samples_panchip
 scripts
 share
 soc
 subsys
 tests_panchip
```

- `arch`：存放芯片 CPU 架构级信息，由于 PAN1080 是 ARM 架构，因此其中只保留了 ARM 架构信息。
- `boards`：存放板级信息，比如 PAN1080 各种 EVB 开发板。
- `cmake`：存放 CMake 构建 Zephyr App 所需的相关配置和脚本文件。
- `doc`：Zephyr 官方文档，其内容与 Zephyr 文档官网一致。
- `drivers`：存放设备驱动代码，例如 pan1080 spi 驱动文件。
- `dts`：存放 DeviceTree（芯片硬件初始化配置）信息。
- `include`：存放 Zephyr 公开的 API 头文件。
- `kernel`：存放 Zephyr OS 内核代码。
- `lib`：存放库文件，包含一个简单的标准 C 语言库等。
- `misc`：其他一些不便分类的文件。
- `modules`：存放关联第三方文件和目录的 modules 目录，提供第三方模块 Kconfig 相关定义途径。
- `samples_panchip`：存放 Panchip 提供的 PAN1080 官方例程。
- `scripts`：存放编译测试相关的脚本文件。
- `share`：存放一些额外的架构无关的数据，目前包含 Zephyr CMake Package。
- `soc`：存放芯片 SoC 级信息及一些默认配置。
- `subsys`：Zephyr 子系统，包括：USB Device Stack（USB 设备栈）、File System（文件系统）、Bluetooth Host and Controller（低功耗蓝牙 Host 端与 Controller 端）等实现代码。
- `tests_panchip`：存放 Panchip 提供的适配 zephyr 公开的 API 测试 case。

3.1 boards

pan1080 SDK 提供了两种不同板级信息，其结构树如下：

```
<home>/01_SDK/zephyr/boards
   arm
     pan1080a_afld_evb
     pan1080a_afx_evb
```

- `pan1080a_afld_evb`：存放 64 PIN 封装，Flash 大小为 1MB 的 PAN1080A SoC 板级信息。
- `pan1080a_afx_evb`：存放 32PIN 封装，Flash 大小为 1MB 的 PAN1080A SoC 板级信息。

有关 pan1080 板级信息配置的更多内容请参考：Zephyr Board 配置指南。

3.2 drivers

pan1080 SDK 适配的 zephyr 驱动结构树如下：

```
<home>/01_SDK/zephyr/drivers
 acc
 adc
 console
 counter
 flash
 gpio
 i2c
 pinmux
 pwm
 qdec
 serial
```

(下页继续)

(续上页)

```
spi
timer
```

3.3 dts

pan1080 SDK DeviceTree 初始化配置信息结构树如下：

```
<home>/01_SDK/zephyr/dts
 arm
   panchip
     pan1080
 bindings
```

- `arm/panchip/pan1080`：存放 pan1080 芯片硬件初始化配置信息。
- `bindings`：bindings 中包含一系列.yaml 格式的文件，其中描述了 dts 各个节点中各个属性的实际含义。

有关 DeviceTree 的更多内容请参考：Zephyr 配置系统指南的 Devicetree 配置部分介绍。

3.4 samples_panchip

pan1080 SDK 提供驱动及应用相关例程的官方源码，其结构树如下：

```
<home>/01_SDK/zephyr/samples_panchip
 basic
 bluetooth
 drivers
 proprietary_radio
 solutions
```

- `basic`：存放基础例程源码。
- `bluetooth`：存放 bluetooth 系列例程源码。
- `drivers`：存放 driver 系列例程源码。
- `proprietary_radio`：存放 2.4G 系列例程源码。
- `solutions`：存放应用方案系列例程源码。

以 `basic\blinky` 为例

```
blinky
   CMakeLists.txt
   prj.conf
   README.rst
   src
       main.c
```

- `src\main.c`： 工程源码。
- `prj.conf`： 工程配置文件。
- `README.rst`：工程说明文件。

有关例程的更多内容请参考：PAN1080 例程介绍。

3.5 subsys

3.5.1 bluetooth Controller

源码路径：<home>/01_SDK/zephyr/subsys/bluetooth/controller_panchip。

通过 HCI 控制 Controller，启动了两个线程：

- controller_thread_data：controller 中断处理函数。
- recv_thread_data：接收 controller 上报的 L2CAP、HCI Events 数据。

Host

源码路径：<home>/01_SDK/zephyr/subsys/bluetooth/host。

GATT Services

源码路径：<home>/01_SDK/zephyr/subsys/bluetooth/host。

GATT Service	File	Configuration
GAP Service	gatt.c	CONFIG_BT_GAP_SERVICE=y (default)
GATT Service Changed	gatt.c	CONFIG_BT_GATT_SERVICE=y (default)CONFIG_BT_GATT_SERVICE_CHANGE (default)

源码路径：<home>/01_SDK/zephyr/subsys/bluetooth/services。

GATT Service	File	Configuration
BAS (Battery Service)	bas.c	CONFIG_BT_BAS=y
TPS (TX Power Service)	tps.c	CONFIG_BT_TPS=y
HRS (Heart Rate Service)	hrs.c	CONFIG_BT_HRS=y
DIS (Device Information Service)	dis.c	CONFIG_BT_DIS=y

Mesh

源码路径：<home>/01_SDK/zephyr/subsys/bluetooth/mesh。

TBD

Mesh Models

源码路径：<home>/01_SDK/zephyr/subsys/bluetooth/mesh。

Mesh Model	File	Configuration
Configuration Client model	cfg_cli.c	CONFIG_BT_MESH_CFG_CLI=y
Configuration Server model	cfg_srv.c	-
Health Client model	health_cli.c	CONFIG_BT_MESH_HEALTH_CLI=y
Health Server model	health_srv.c	-

源码路径：<home>/01_SDK/zephyr/subsys/bluetooth/mesh_models/sig_models。

Mesh Model	File	Configuration
SIG OTA Server model	blob_srv.c	CONFIG_BT_MESH_SIG_OTA_SRV=y
PB Remote Server model	remote_srv.c	CONFIG_BT_MESH_PB_REMOTE_SRV=y

Vendor Models

源码路径：<home>/01_SDK/zephyr/subsys/bluetooth/mesh_models/vendor_models。

Mesh Model	File	Configuration
Packet Test Vendor Server model	ptv_srv.c	CONFIG_BT_PAN_MESH_MODELS=yCONFIG_BT_MESH_PTV_SRV=

3.6 tests_panchip

pan1080 SDK 提供驱动 API 测试例程的官方源码，其结构树如下：

```
<home>/01_SDK/zephyr/tests_panchip
 bluetooth
 drivers
 subsys
```

- `bluetooth`：存放 bluetooth API 测试代码。
- `drivers`：存放驱动 API 测试代码。
- `subsys`：存放 power manage、storage 等 API 测试代码。

1.4 Zephyr 简介

1.4.1 1 Zephyr Project 概述

Zephyr™ 项目是一个采用 Apache 2.0 协议许可，Linux 基金会托管的协作项目。为所有资源受限设备，构建了针对低功耗、小型内存微处理器设备而进行优化的物联网嵌入式小型、可扩展的实时操作系统（RTOS），支持多种硬件架构及多种开发板。由于 IoT 领域需要部署大量的联网设备，因此每个设备的成本必须得到控制。控制成本第一个有效方法是降低昂贵组件的标准，例如使用 RAM 更低、ROM 更低的芯片。Zephyr 就是专为这样的芯片而生的，它可运行在只有 8 Kb 内存的 MCU 之上，甚至能在只有 2 Kb 内存的 MCU 上演示 Hello World。

1.4.2 2 Zephyr 主要特性

- **高度可配置**

  降低成本的另一个方式是按需裁剪硬件。物联网设备一般都是专用设备，因此在面对某个特定市场时只需要特定的硬件。

- **多个架构的支持**

  目前支持种芯片架构：ARM，X86，MIPS，ARC，RISC-V 等等，便于后续产品端跨不同的平台适配和使用。

- **可移植性强**

  用户可以很方便的将自己的平台移植到 Zephyr 系统（目前已经有 350+ 个芯片和 board 已经完成了 Zephyr 平台的适配），用户只需专注于应用层的软件开发和使用。同时已有的产品的上层应用程序，可以无缝对接，而不需要重新的开发。目前我们 pan1080 已经全面适配到该系统了。

- **强大的社区支持**

  Zephyr 受到几大厂商/基金会的支持，包括 Linux 基金会、Linaro 组织（成员包括 ARM、飞思卡尔、IBM、三星、ST、TI、华为海思等）、Intel、NXP、新思科技等，开发者众多。

- Shell **支持**

  想要查看设备内部的运行情况？Zephyr 提供了一个 shell 接口，您可以将您的应用程序的部分接口暴露给 shell，这样您就能与设备进行人机交互了。当然，更高大上的做法是使用 APP 控制设备，直接在 APP 上与设备进行交互，不过这样的开发成本会增加很

多。此外，该功能也是可裁剪的，您可以在开发测试阶段使用该功能，正式投入产品后将其裁剪掉。

- **丰富的组件**
  - SEGGER IDE/Debug Tool：用户可以方便的基于现有的 Jlink 配套的工具进行烧录调试
  - GNU Compiler/Debug Tools：丰富的开源工具链
  - IDE **的支持**：eclipse/VS code
  - memfault：允许开发人员远程监控、调试和更新固件，对于复杂系统长时间异常稳定性分析非常有帮助
  - MCUboot：安全的 BootLoader
  - Mbed TLS：嵌入式平台的最小编码的加解密算法库
  - **丰富的应用例程**：完整的应用例程供开发者参考，可以快速搭建应用程序，不需要从 0 开始编写应用程序
  - mcumgr：支持 SMP（simple management protocol）作为 OTA 协议（手机 APP 端有标准的协议支持）
  - ...
- **电源管理**

  很多物联网设备都是用纽扣电池供电的，经常更换电池的成本是非常高的。当然，这里说的高并非电池本身昂贵，而是指更换电池时的人力成本。要想一颗小小的纽扣电池运行数年可不是一件简单的事儿，Zephyr 早已考虑到了这一点，它提供了一个电源管理子系统，管理外设的功耗，以达到省电的目的。此外，该功能也是可裁剪的，如果您的设备没有低功耗要求，可以将其裁剪掉。
- **非易失存储子系统**

  Zephyr 提供一套完善易用的 flash 管理机制，提供经过长期验证 NVS （Non-Volatile Storage）存储机制，可以最大可能的延迟 flash 的存储寿命。Zephyr 同时提供便捷的应用操作接口（settings），对应用层使用提供简单的键值对的使用接口。用户只需要专注应用数据的读取和写入使用，不需要关心其它的 flash 底层细节，进而提升应用开发的效率。
- **日志系统**

  Zephyr 包含了一个非常强大的日志系统。输出日志时的执行时间对程序的逻辑会产生影响，导致不可预期的后果。而 Zephyr 强大的 logging 系统，输出一行日志，因为缓冲区的存在，几乎不占用执行时间 (只占用数据拷贝的时间)，日志先缓存到缓冲区中，然后在系统空闲时，将缓冲区的数据输出到后端设备 (如串口，网络，rtt 等)。

1.4.3 3 关于 RTOS 的一些说明

嵌入式开发人员通常习惯于裸机编程，或者对使用实时操作系统 (RTOS) 有所保留。这里谈谈 RTOS 的作用，以及为什么应该考虑使用它们。

现今的产品开发周期变得越来越复杂。开发时间越来越短，但所需功能集日益扩展，使得繁忙的开发人员需要千方百计在更短的时间内完成更多工作。通常，使用 RTOS 可以提高任务管理和资源共享的效率。

- **什么是实时操作系统** (RTOS)

  简而言之，RTOS 是一款旨在有效管理中央处理器 (CPU) 时间的软件。当时间是应用的重点时，这对于嵌入式系统尤为重要。Windows 等操作系统与嵌入式系统中常用的 RTOS 的主要区别，在于对外部事件的响应时间。Windows 等普通 OS 提供对事件的不确定性响应，即使试图保持响应速度也无法保证何时处理事件。使得用户认为操作系统响应灵敏，比处理基础任务更为重要。另一方面，RTOS 则提供了实时响应和高度确定性的反应。习惯使用 Windows 或 Linux 等操作系统的开发人员将会非常熟悉嵌入式 RTOS 的特性。它们经设计在内存有限的系统中运行，并且可以无限期运行而无需重置。由于 RTOS 旨在快速响应事件并在重负载下执行，RTOS 在执行大型任务时可能会较其他 OS 慢。

- RTOS **调度**

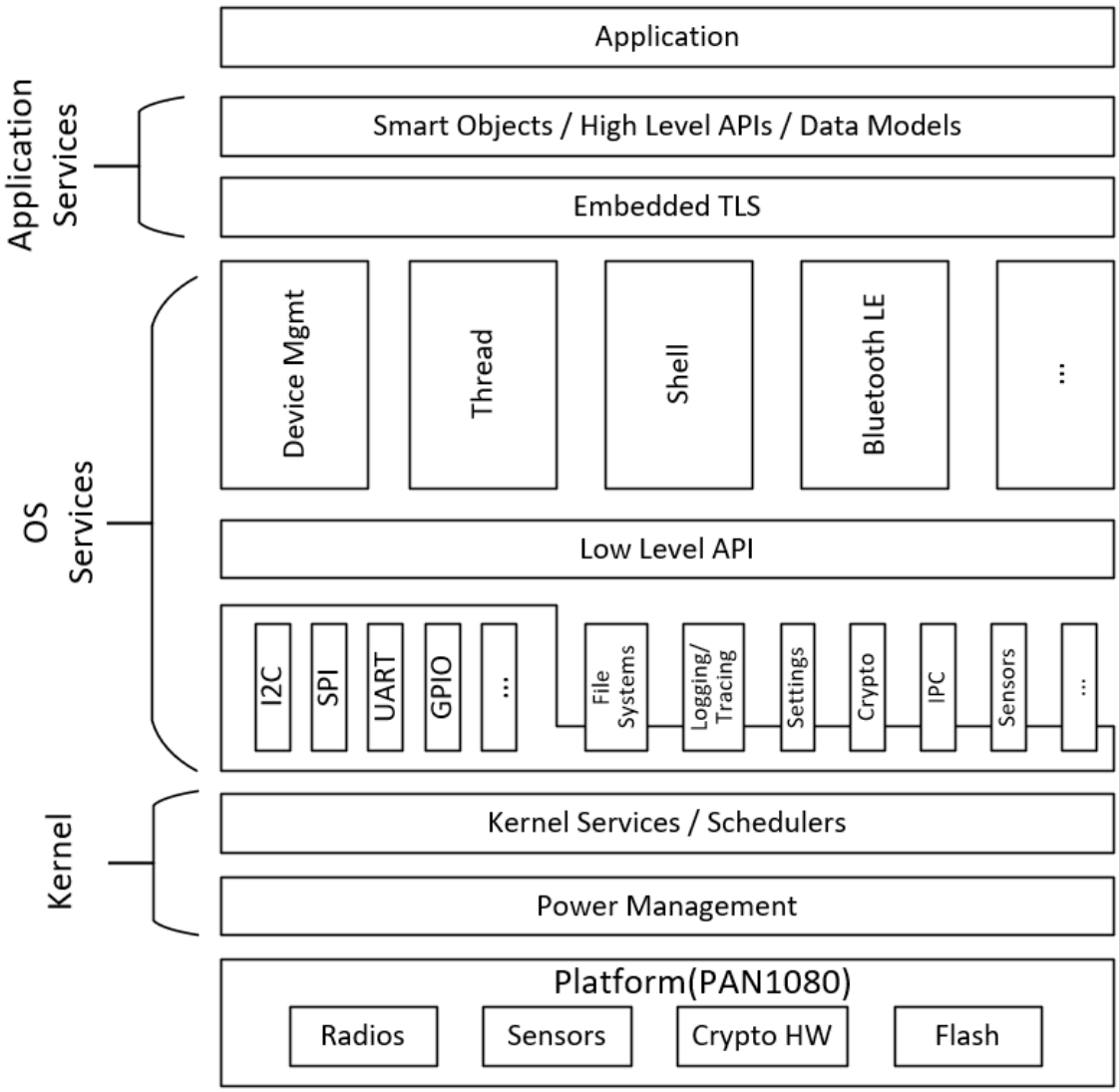
  RTOS 的价值在于很高的响应速度，而高级调度算法是其中的关键组件。嵌入式系统的时间要求各有不同，从软实时洗衣机控制系统到硬实时飞机安全系统等等。在后者的情况下，只有能够准确预测 OS 调度程序的行为，才能满足实时要求的基础需求。许多操作系统给人以一次执行多个程序的印象，但是这种多任务处理其实只是一种错觉。单个处理器内核在任一时间只能运行单个执行线程。操作系统的调度程序决定何时运行哪个程序或线程。通过在线程之间快速切换，它带来了同时执行多项任务的错觉。灵活的 RTOS 调度程序允许采用广泛的方法来处理优先级，尽管 RTOS 通常主要用于非常狭窄的应用程序集。RTOS 调度程序应提供最小的中断延迟和最小的线程切换开销。这是使得 RTOS 与重视时间的嵌入式系统如此密切相关的原因。

- **在嵌入式设计中使用** RTOS

  许多嵌入式设计编程人员避免使用 RTOS，因为他们怀疑 RTOS 给其应用程序增加了太多的复杂性，或者 RTOS 其实是一个未知的领域。RTOS 通常需要最多占用 5% 的 CPU 资源来执行其任务。尽管总会有一些资源上的损失，但 RTOS 可以在简化的确定性，通过硬件抽象的易用性，减少的开发时间以及更便利的调试等等方面弥补这一不足。使用 RTOS 意味着你可以同时运行多项任务，并在需要时引入基本的连接性、隐私、安全性等。RTOS 允许针对项目的特定需求创建优化的解决方案。

1.4.4 4 Zephyr For PAN1080 **的特点**

目前 PAN1080 已经全面适配了 Zephyr 2.7.0 LTS2，并进行了深度裁剪和优化，整体框架如下：



4.1 Zephyr 特性说明

- Kernel & OS Services
- Bluetooth LE Core v5.1
  - Bluetooth Low Energy Controller support (LE Link Layer)
    - Unlimited role and connection count, all roles supported
    - Concurrent multi-protocol support ready
    - Intelligent scheduling of roles to minimize overlap
    - Portable design to any open BLE radio, currently supports Nordic Semiconductor nRF51 and nRF52, as well as proprietary radios
    - Supports little and big endian architectures and abstracts the hard real-time specifics so that they can be encapsulated in a hardware-specific module
    - Support for Controller (HCI) builds over different physical transports
  - Bluetooth Host support
    - Generic Access Profile (GAP) with all possible LE roles

        - Peripheral & Central
        - Observer & Broadcaster
      - GATT (Generic Attribute Profile)
        - Server (to be a sensor)
        - Client (to connect to sensors)
      - Pairing support, including the Secure Connections feature from Bluetooth 4.2
      - Non-volatile storage support for permanent storage of Bluetooth-specific settings and data
    - Bluetooth mesh support
      - Relay, Friend Node, Low-Power Node (LPN) and GATT Proxy features
      - Both Provisioning bearers supported (PB-ADV & PB-GATT)
      - Highly configurable, fits as small as 16k RAM devices
    - Clean HCI driver abstraction
- OS Abstraction
- File System / Storage
- Debugging
- Power Management
- Device Firmware Upgrade / Device Management
- Cryptography
- USB Device Stack
- Test Framework

4.2 特色说明

- **集成独立的开发环境**

  只需要下载我们的开发包，一键解压即可使用，不需要繁杂的开发环境搭建和适配

- **可视化的开发环境**

  基于可视化的编辑工具 VS Code，构建图形化的编译，开发，调试和下载环境，用户不需要使用命令行也可以进行开发

- **源代码开源**

  除 Bluetooth LE Controller（这部分代码与硬件涉及强相关，由 PANCHIP 维护）以外，其余所有源代码都开源

- Bluetooth LE 5.1 **全功能运行支持**

  基于强大的资源，PAN1080 支持单芯片运行所有的功能（BLE 和 Mesh 以及一些外设都可以在一个程序上运行），可以适用于更多的应用场景

4.3 补充说明

此处补充介绍一下 PAN1080 Bluetooth LE 和 Proprietary Radio 2.4G 的相关特性。

4.3.1 Bluetooth Features

4.3.1.1 Bluetooth Low Energy Controller The PAN1080 Bluetooth Low Energy Controller supports all low-energy features required by Bluetooth specification version 5.0. it also supports Constant Tone Extension defined in Bluetooth specification version 5.1. The controller supports the following:

- Support 1M PHY, 2M PHY and Coded PHY (s2 and s8)
- Support Advertising, Scanning, Initiating and Connection (both of Central and Peripheral) role
- Up to 10 Link Layer state machines concurrently:
  - 1 * Passive Scanning
  - 1 * non-connectable advertising
  - 8 * any other combinations (Legacy/Extended/Periodic Adverstising, Scanning and Connection)
- Support LE Features:
  - LL Encryption
  - LE Data Packet Length Extension
  - LL Privacy
  - Extended Scanner Filter Policies
  - Multiple PHYs
  - LE Extended and Periodic Adverstising
  - Channel Selection Algorithm #2
  - Minimum Number of Used Channels
  - Constant Tone Extension
  - and etc
- Support Update Channel Statistics

4.3.1.2 Bluetooth Host

- Generic Access Profile (GAP) with all possible LE roles
  - Peripheral & Central
  - Observer & Broadcaster
- GATT (Generic Attribute Profile)
  - Server (to be a sensor)
  - Client (to connect to sensors)
- Pairing support, including the Secure Connections feature from Bluetooth 4.2
- Non-volatile storage support for permanent storage of Bluetooth-specific settings and data
- IPSP/6LoWPAN for IPv6 connectivity over Bluetooth LE
  - IPSP node sample application
- Clean HCI driver abstraction
  - 3-Wire (H:5) & 5-Wire (H:4) UART
  - SPI
  - Local controller support as a virtual HCI driver

4.3.1.3 Bluetooth Mesh

- Compatible with Bluetooth SIG Mesh Profile 1.0.1
- Support Mesh Provisioning
- Provisioner: PB-ADV
- Provisionee: PB-ADV, PB-GATT and PB-Remote
- Support Mesh Node Feature : Relay, Proxy, Friend, LPN
- Support Mesh Models
  - SIG Models: Config Model, Health Model and Generic Models (Onoff and Light Control Models)
  - SIG Developing Models: PB-Remote Model and SIG OTA Model
- Support multiple smart speakers control concurrently for BaiDu Xiaodu, Alibaba Aligenie and Amazon Echo
- Support network control: HeartBeat, Subnet, Secure Beacon and Group Control
- Support switch control for over 256 nodes without delay
- Mesh Security
  - Provisioning：FIPS P-256 Elliptic Curve
  - Message：AES-CCM Encrytion
  - Network：SEQ Control, IV Index and Key Fresh

4.3.2 Proprietary Radio 2.4G Features

- Support 1M and 2M PHY
- XN297L, nRF24L01, and nRF52832 2.4 GHz Transceiver protocol compliant
- Support No Acknowledge, Acknowledge and Acknowledge with payload
- Support CRC8, CRC16 and CRC24
- Support whitening

Chapter 2

硬件资料

2.1 PAN1080 EVB 介绍

2.1.1 1 概述

本文是 PAN1080A Evaluation Board V1.0 EVB 开发板介绍，包括相关板级硬件模块、各模块在 EVB 板的位置、以及对应电路原理图，旨在帮助开发者快速了解 PAN1080A EVB 开发板。

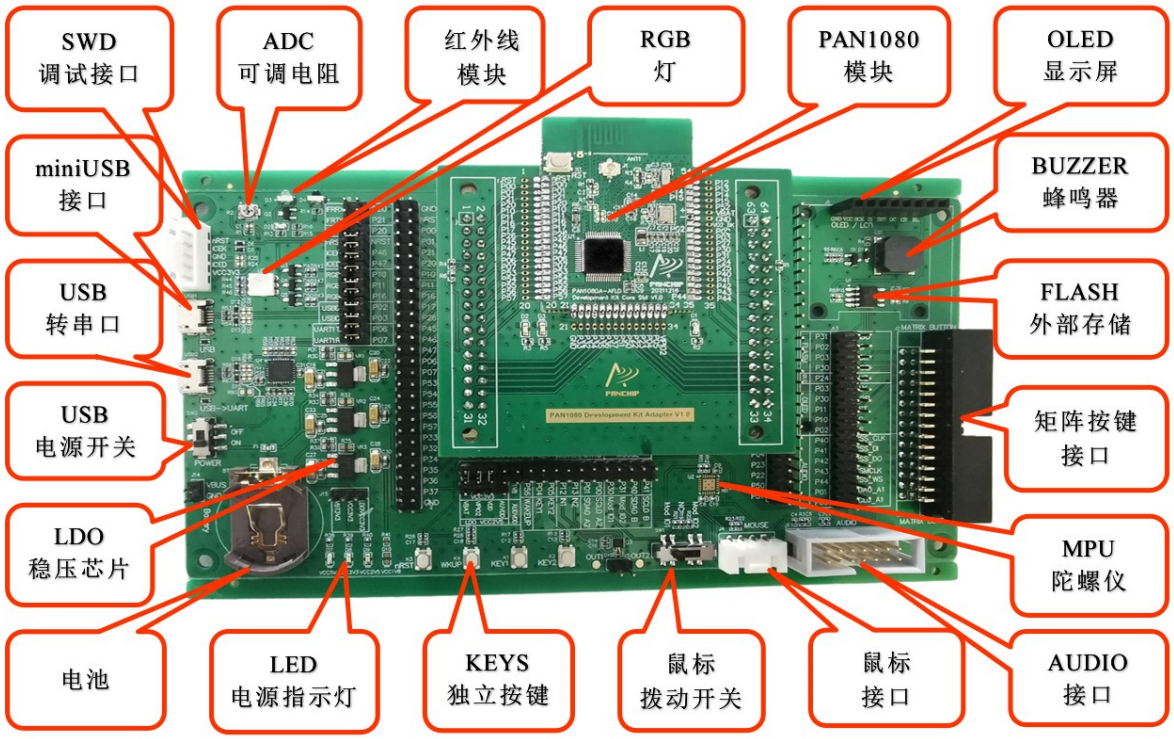
PAN1080A EVB 开发板由核心板、转接板以及底板三大部分组成，其中：

- 核心板提供了 PAN1080A SoC 的最小系统，主要包含有 PAN1080 SoC 芯片、32MHz 高速晶振、32768Hz 低速晶振、复位按钮、板载天线以及一些必要的无源器件；

  核心板目前有两个型号：

  - PAN1080A-AFLD Development Kit Core Std V1.0：其主控 SoC 型号为 PAN1080LB5（LQFP 封装、1MiB Flash、64 Pin）
  - PAN1080A-AFX Development Kit Core Std V1.0：其主控 SoC 型号为 PAN1080UB1（QFN 封装、1MiB Flash、32 Pin）

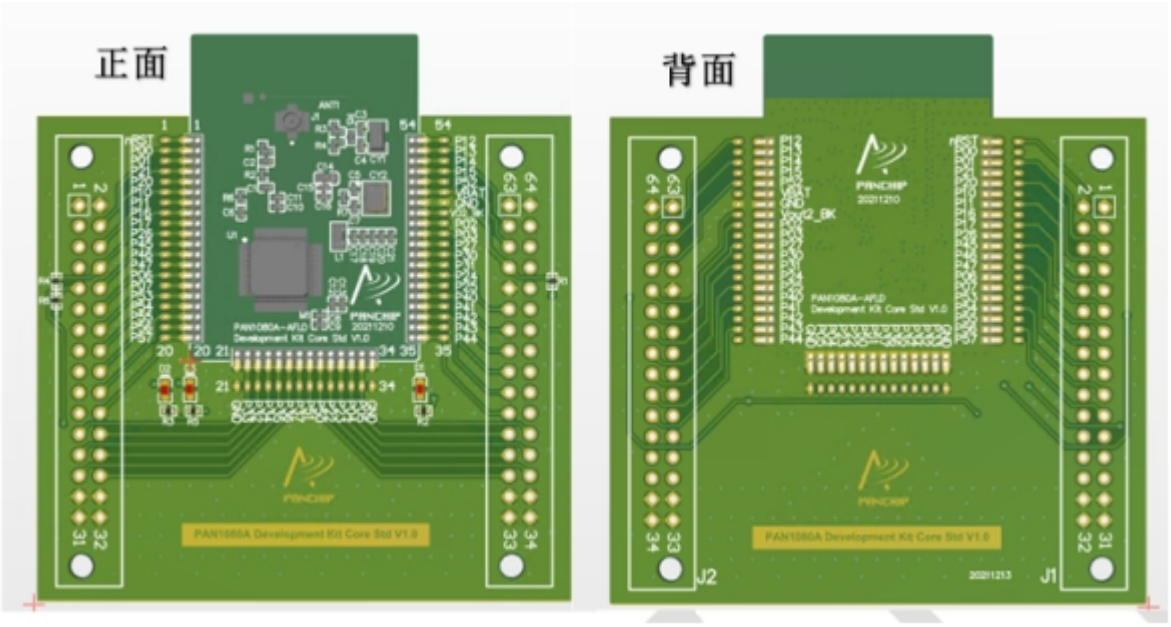
- 转接板用于连接核心板与底板，一般与核心板固定在一起，其主要包含 2 个防反接的插头，以及 3 个 LED 发光二极管；
- 底板上提供了诸多 PAN1080 SoC 支持的外设模块，其中包含有电源系统、USB 转串口模块、电机驱动模块、RGB 三色灯、MPU 六轴传感器、OLED 显示屏接口、外部 SPI FLASH、无源蜂鸣器、独立按键、可调电阻、鼠标接口，矩阵按键接口、音频接口、红外模块等等；



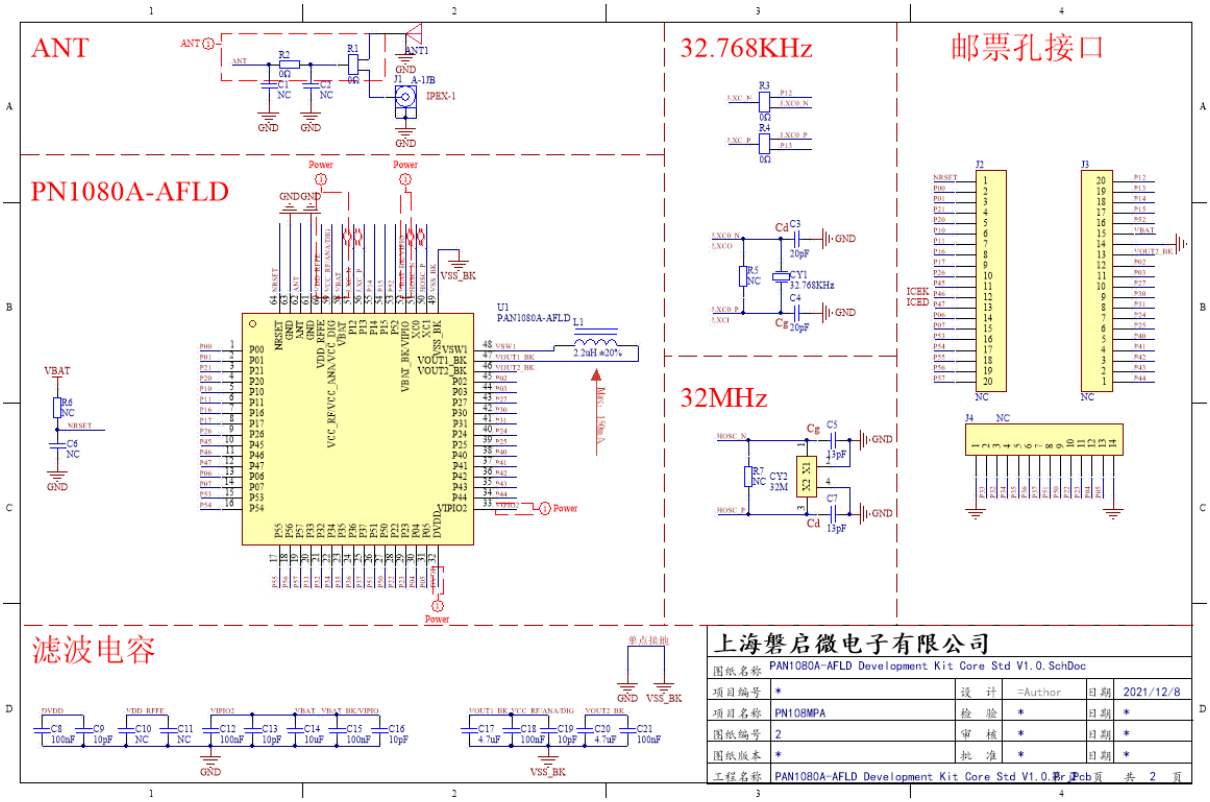
2.1.2 2 开发板硬件资源

2.1 PAN1080 最小系统

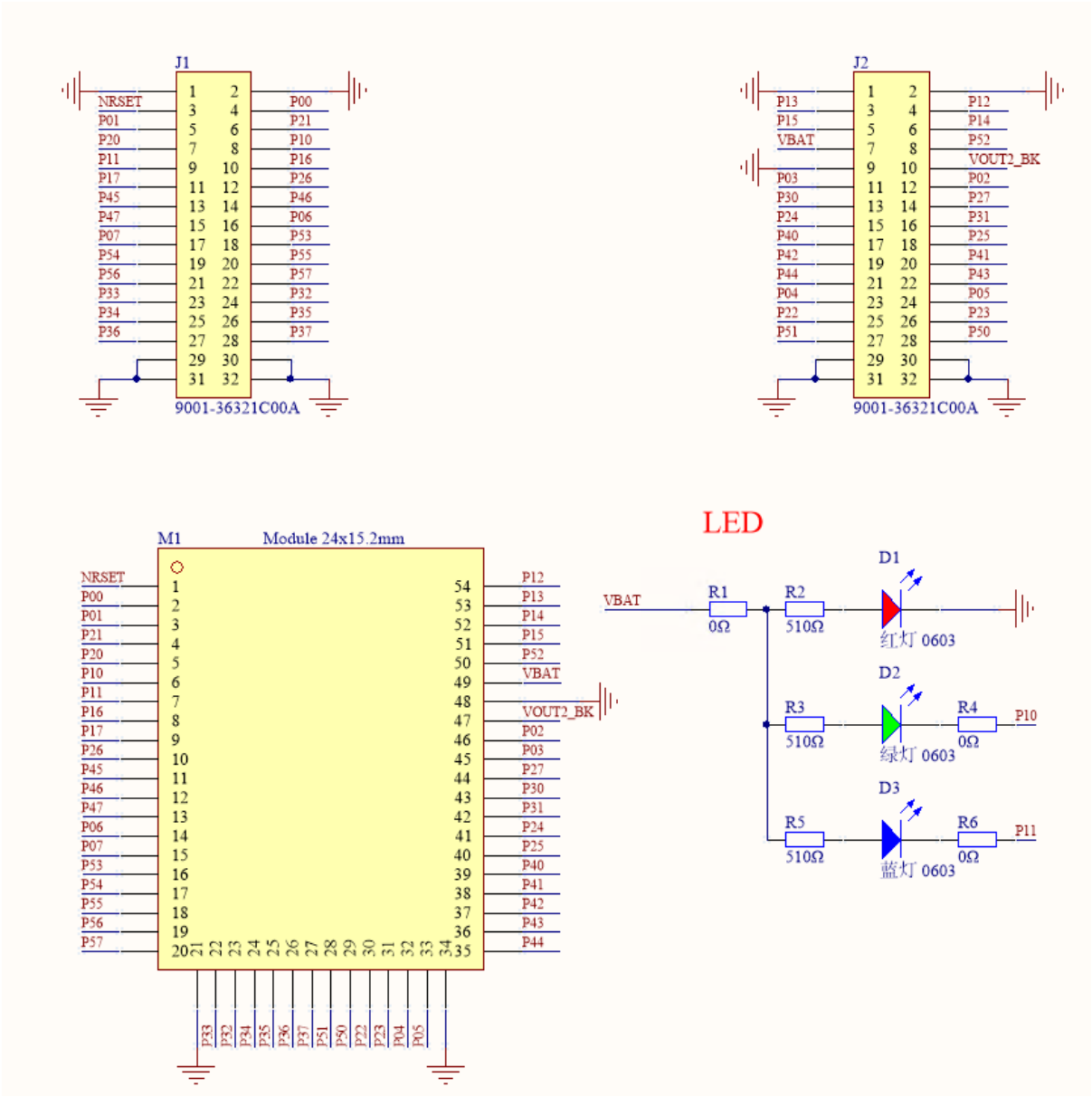
PAN1080 最小系统由核心板和转接板组成。芯片目前仅提供了 LQFP64 和 QFN32 两种封装，最小系统以模块形式嵌入开发板底板中，可分离式设计方便单独调试及应用于其它场景，如下图所示：



核心板搭载 PAN1080 主控芯片、外部 32M 晶振、板载天线等，通过标准间距 2.54mm 双排针引出了所有 IO，PAN1080A-AFLD 核心板原理图如下所示：



转接板将邮票孔转为防反接的插头，其原理图如下所示：

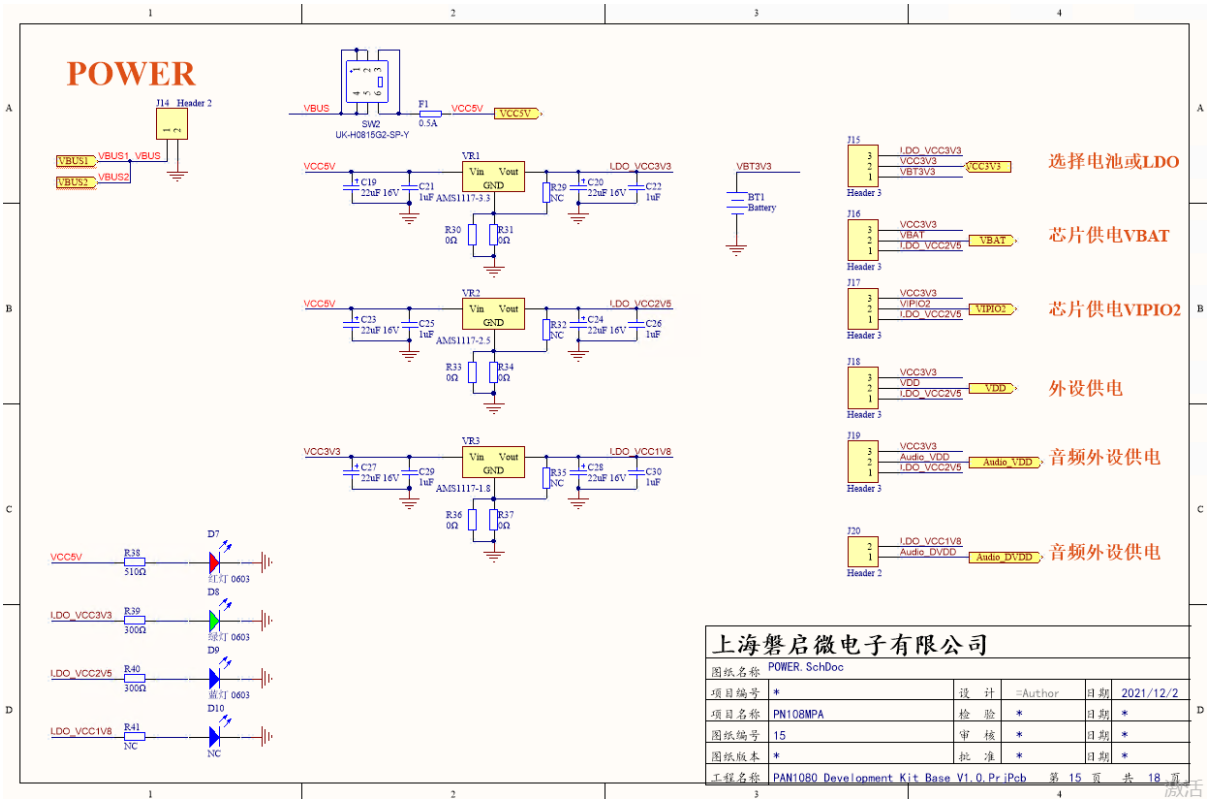


2.2 电源模块

PAN1080A EVB 开发板可选择使用以下两种供电方式之一：

- 5V 的 USB 供电；
- 3V 的 CR2032 纽扣电池供电；

EVB 开发板电源模块原理图如下：

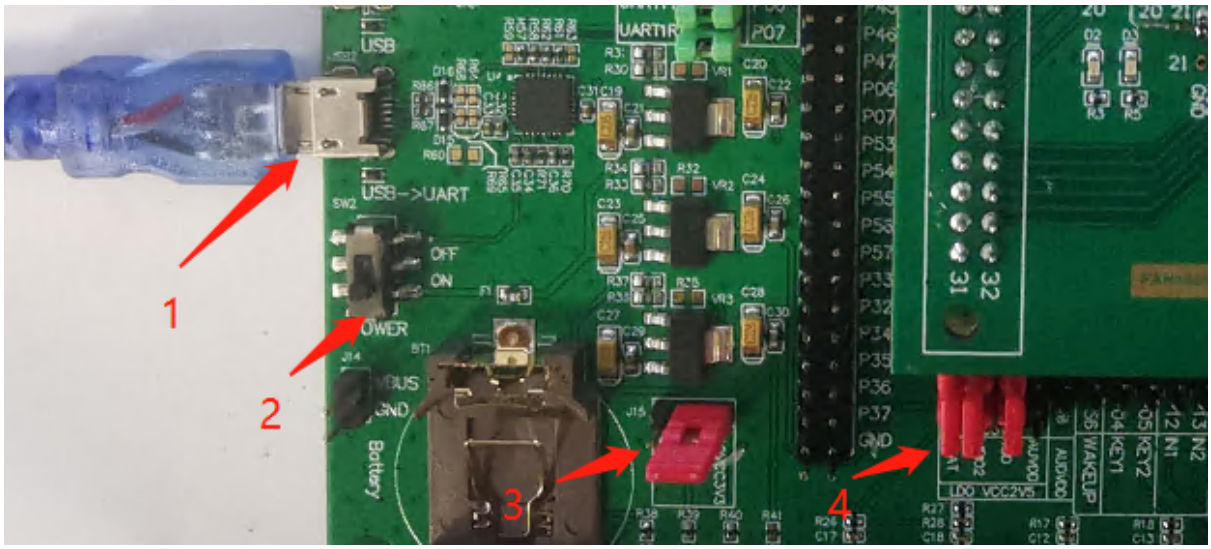


EVB 开发板左侧有两个 MicroUSB 接口 `USB1` 与 `USB2`，它们的电源引脚均与 EVB 的 5V 电源网络连在一起；除此之外，二者还有以下区别：

- `USB1` 为普通 USB 接口，使用跳线帽将 EVB 底板的 `USBDM/USBDP` 分别与 SoC 的 `P02/P03` 引脚相连，即可使用 PAN1080 SOC 的 USB 模块；
- `USB2` 为 USB 转 UART 模块的 USB 端接口，使用跳线帽将 EVB 底板的 `UART1 TX/UART1 RX` 分别与 SoC 的 `P06/P07` 引脚相连，即可通过 USB 转串口模块，实现 PAN1080 SOC 的 UART1 与 PC 进行通信；

  注：在实际使用过程中，选用任意一个 USB 口供电即可，但需注意，板载电源切换开关应拨动至 ON 档位，此时稳压芯片 VR1、VR2 工作，电源指示灯 D9、D10 常亮。

一种典型的供电方法如下图所示：



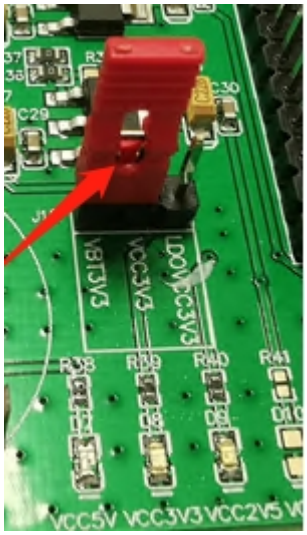
其中：

1. `USB2` 连接至 PC（**图注** 1）；

2. 拨动开关 SW2 往下拨到 ON 档位（**图注** 2），可以看到红灯亮（LED-D7，5V），绿灯亮（LED-D8，LDO3.3V），蓝灯亮（LED-D8，LDO2.5V 输出）；
3. 使用跳线帽短接左下方排针 J15，目的是将 LDO 电源模块的输出引脚 LDOVCC3V3 连接至 VCC3V3（**图注** 3）；
4. 使用跳线帽短接下方排针 J16、J17、J18，作用是 VCC3V3 分别连接至 VBAT、VIPIO2、VDD（**图注** 4）；

> 电源网络的三个 LDO 模块，分别输出 3.3V、2.5V、1.8V，其中 3.3V 或 2.5V 用于给 PAN1080 SoC 芯片供电，1.8V 用于给音频外设供电。

另外，若希望开发板由左下角纽扣电池供电，则使用跳线帽将 J15 与 BAT3V3 短接即可：

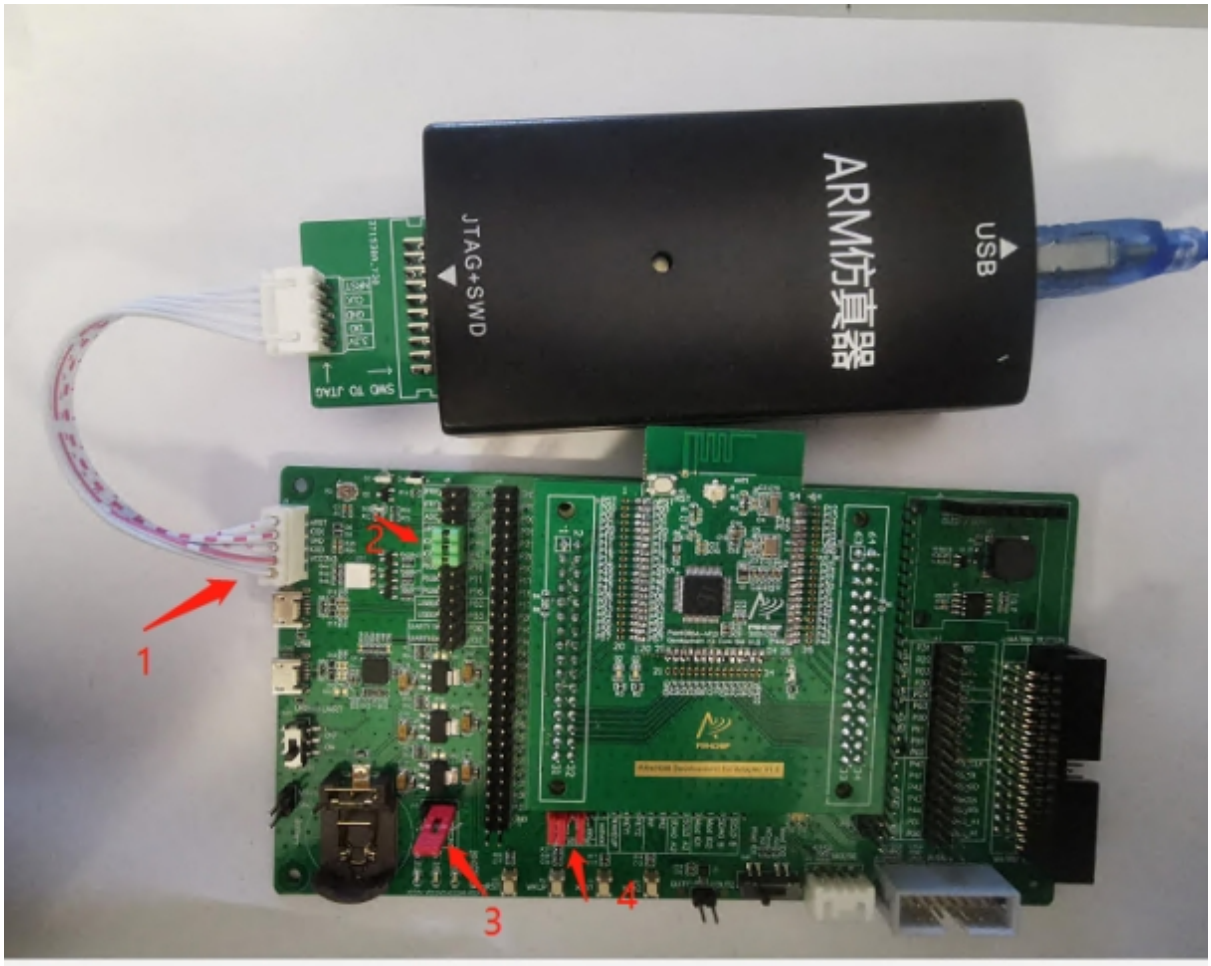


2.3 SWD **调试接口**

开发板提供了单排针接口用于连接 J-LINK 实现 SWD 调试和程序下载功能，该排针接口位于 USB 接口上方。

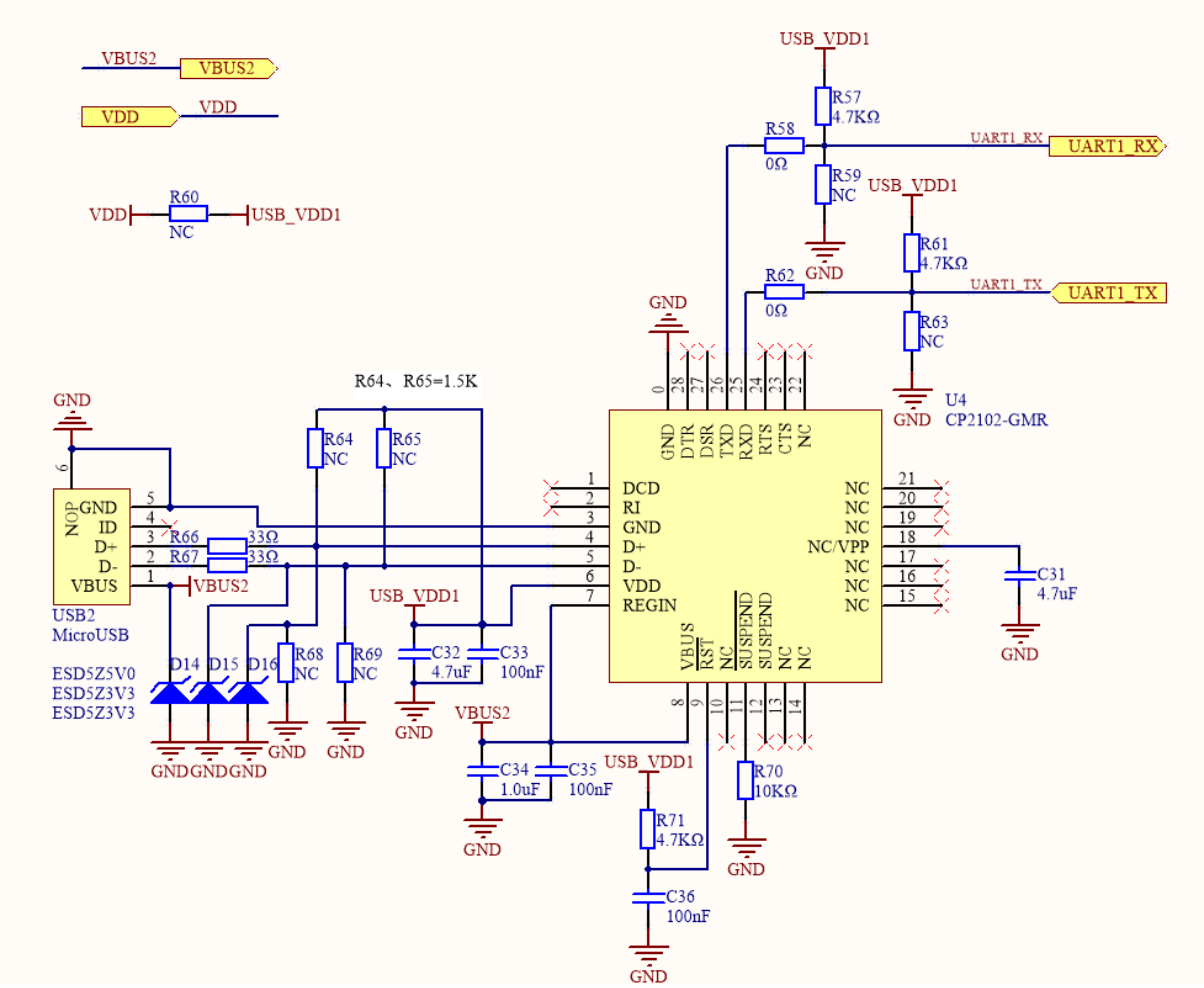
一种典型的使用方法如下图所示：

1. JLINK 下载器插到左上方 SWD 接口 J9（**图注** 1）；
2. 用跳线帽短接左上方排针 J8 的三对引脚：nRST 连接到 NRST，ICEK 连接到 P46，ICED 连接到 P47（**图注** 2）；
3. 使用跳线帽短接左下方排针 J15，目的是将 LDO 电源模块的输出引脚 LDOVCC3V3 连接至 VCC3V3（**图注** 3）；
4. 使用跳线帽短接下方排针 J16、J17、J18，作用是 VCC3V3 分别连接至 VBAT、VIPIO2、VDD（**图注** 4）；

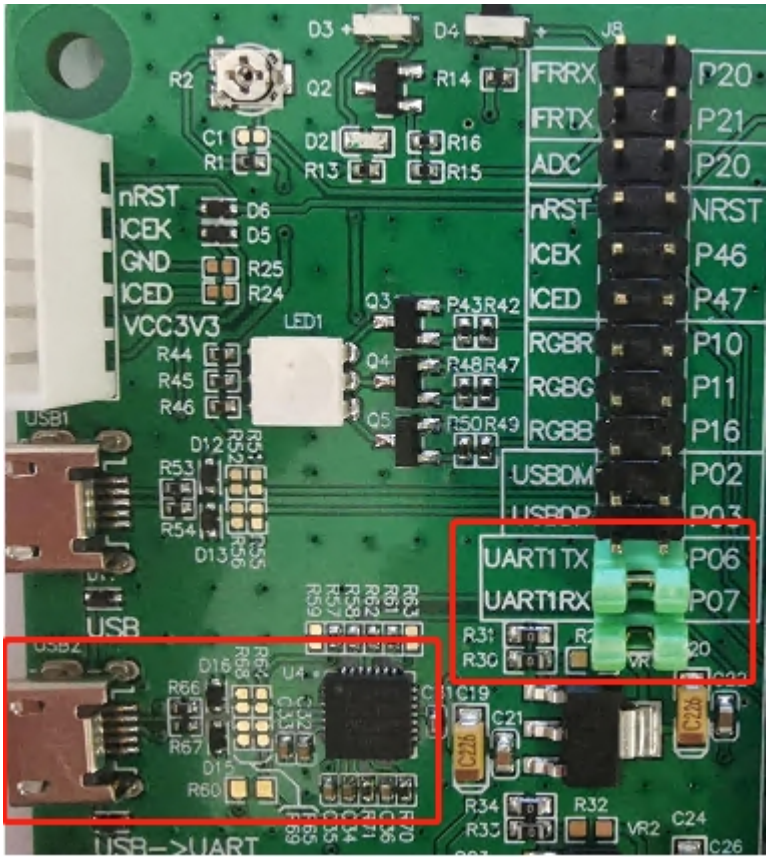


2.4 USB 转串口模块

PAN1080 SoC 的 P06、P07 引脚可通过软件配置成 UART 串口功能，然后通过 CP2102 模块转为 MicroUSB 接口。

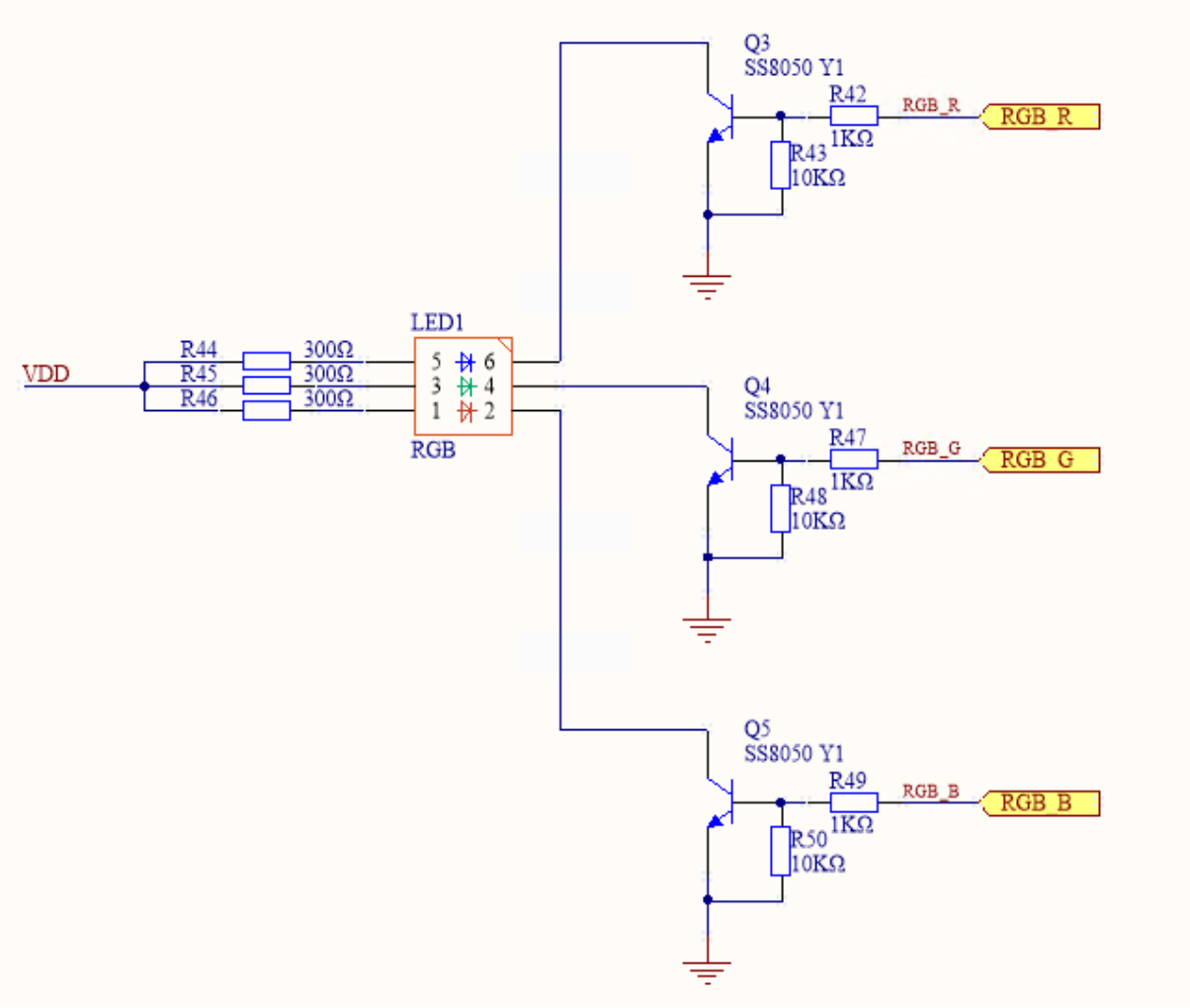


USB 转串口模块使用 USB2 接口，需要使用跳线帽短接排针 J8 的两对引脚：UART1 TX 连接到 P06、UART1 RX 连接到 P07，如下图所示：

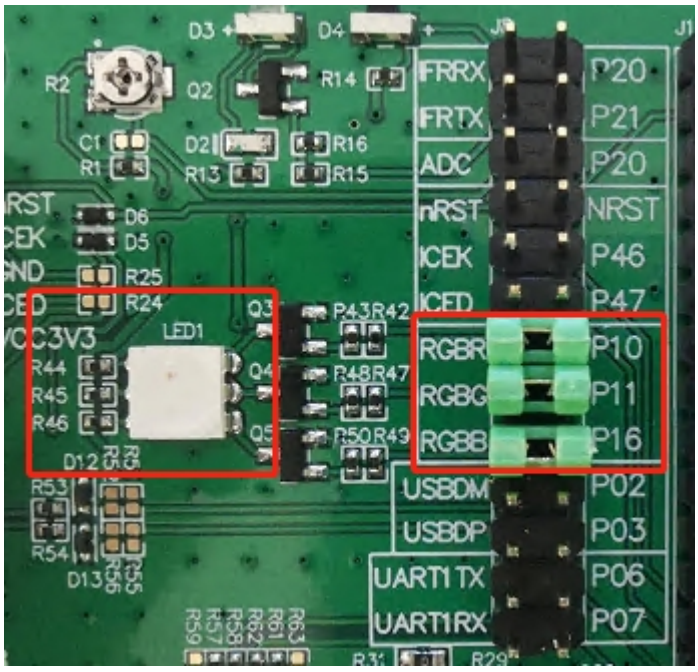


2.5 RGB 灯

开发板搭载单颗 5050 封装的 RGB 灯，可由芯片 P10/P11/P16 三个 IO 通过晶体管控制，实现亮灭或渐变等效果。

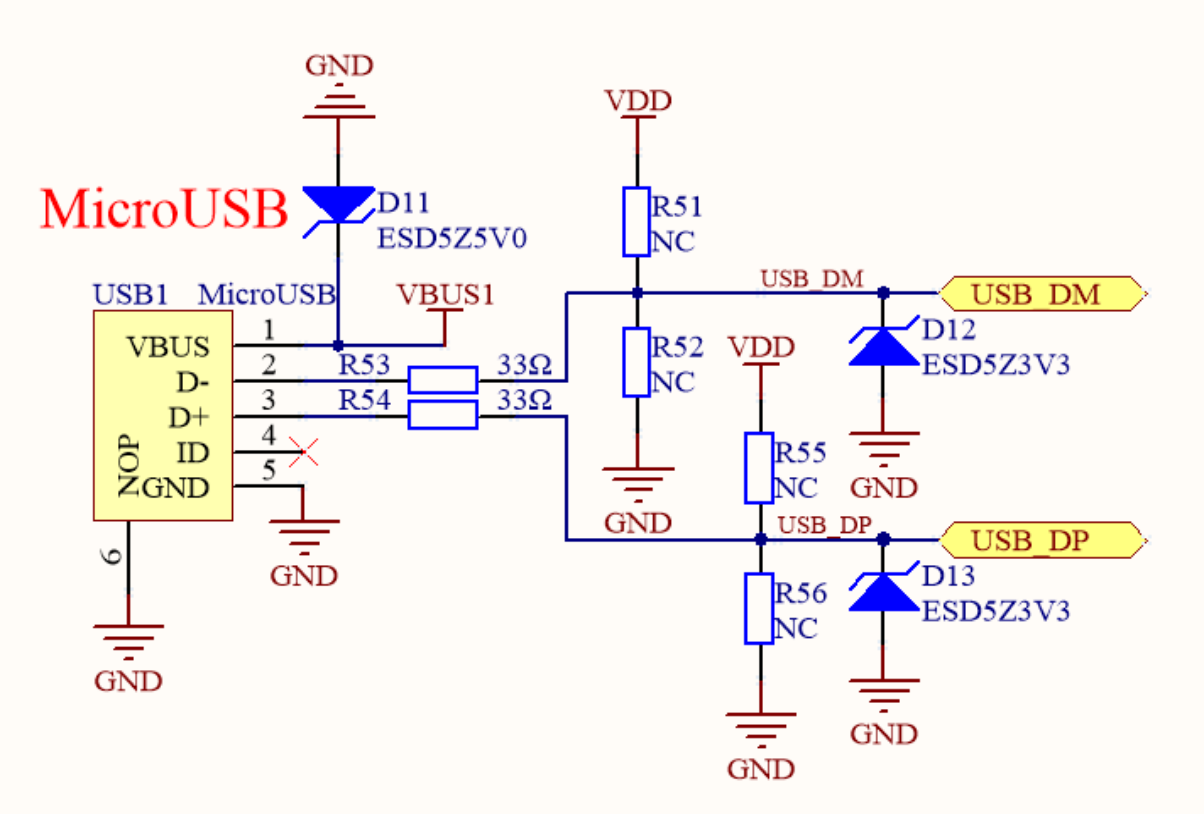


为使用此模块，需要使用跳线帽短接排针 J8 的三对引脚：RGBR 连接到 P10、RGBG 连接到 P11、RGBB 连接到 P16，如下图所示：

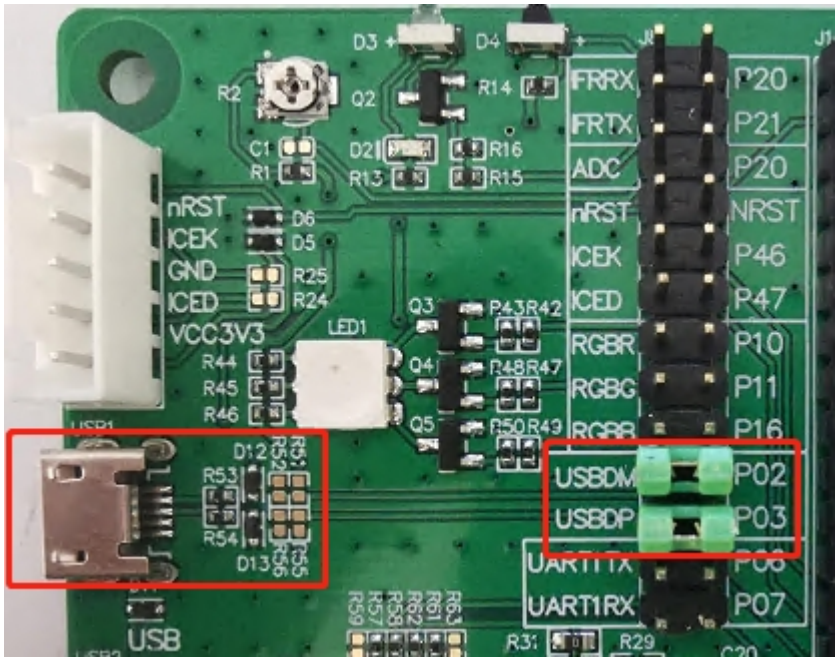


2.6 USB 模块

开发板提供一个 MicroUSB 接口，原理图如下:

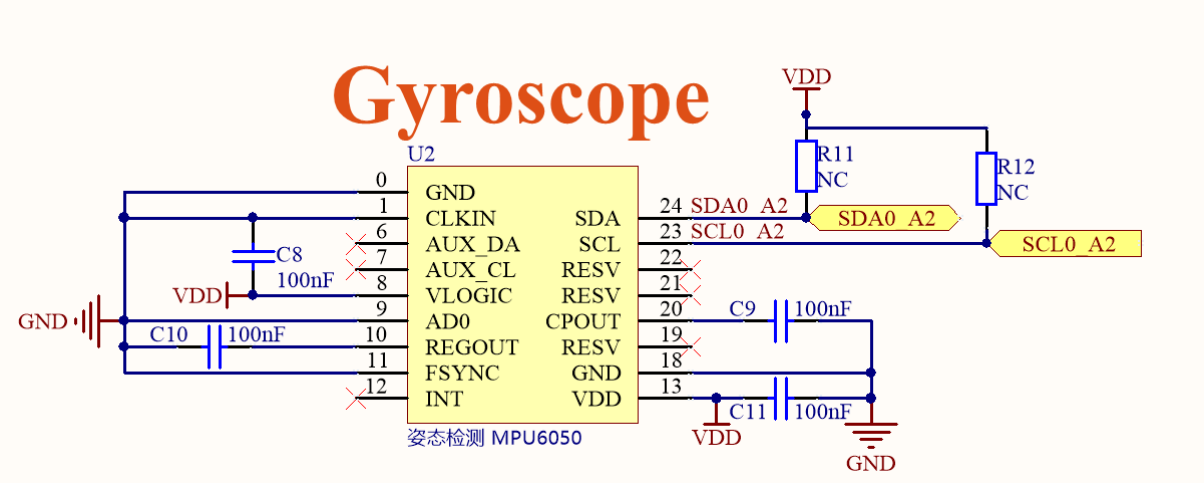


为使用此模块，需要使用跳线帽短接排针 J8 的两对引脚：USBDM 连接到 P02，USBDP 连接到 P03，如下图所示:

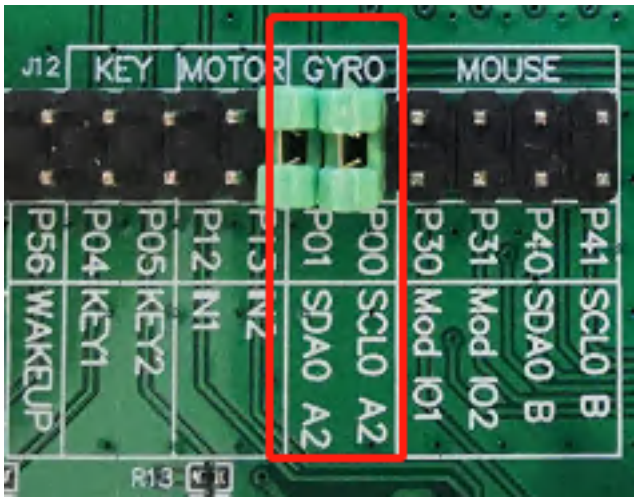


2.7 运动传感器

开发板搭载了六轴运动传感器 MPU6050，集成三轴 MEMS 陀螺仪和三轴 MEMS 加速度计，提供了 IIC 接口与主控芯片进行通信，原理图如下：

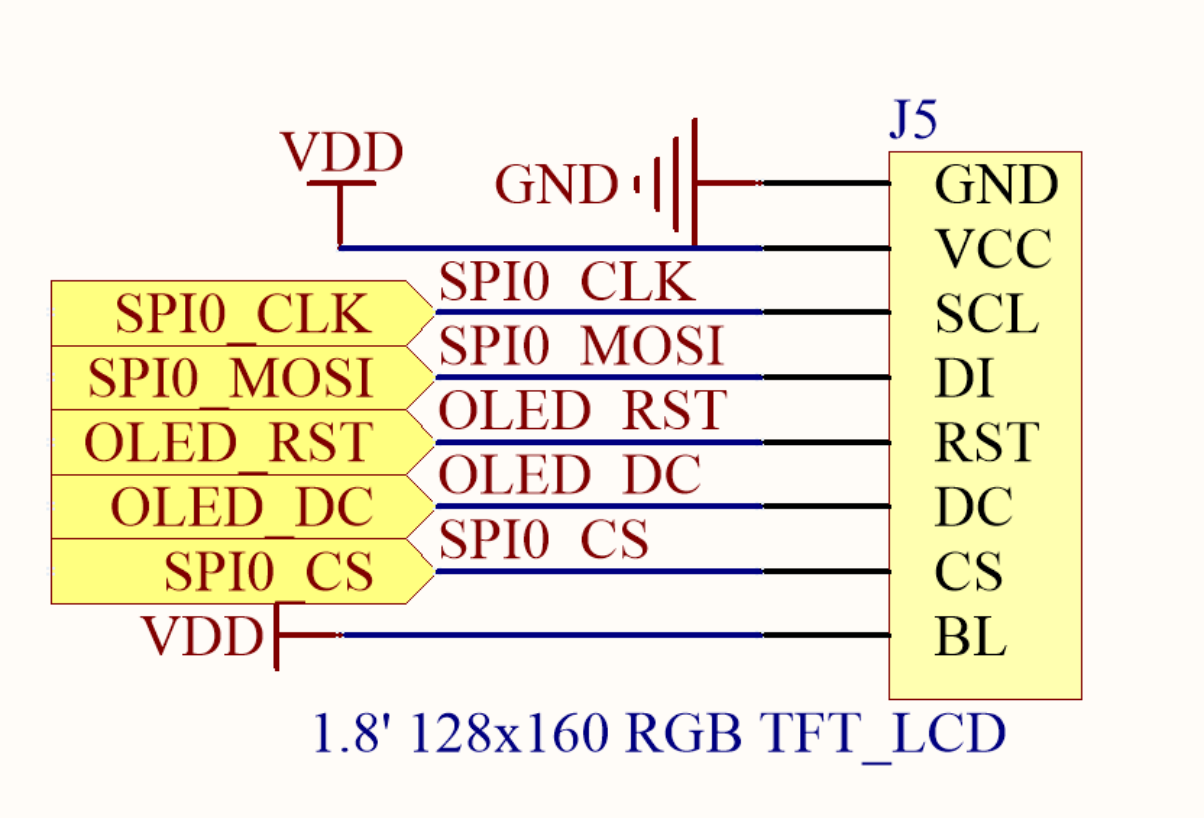


为使用此模块，需要使用跳线帽短接排针 J12 的两对引脚：P01 连接到 SDA0_A2、P00 连接到 SCL0_A2，如下图所示：

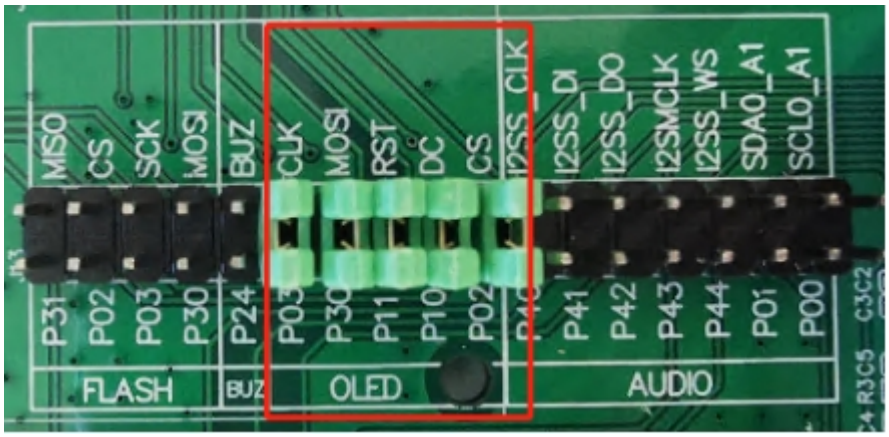


2.8 OLED 显示屏

开发板搭载了常见的 0.96 寸、七针接口、128*64 分辨率的 OLED 显示屏模块接口，OLED 模块使用 SSD1306 显示驱动芯片进行控制，具备内部升压，对外默认提供三线 SPI 接口与主控芯片进行通信，其原理图如下：



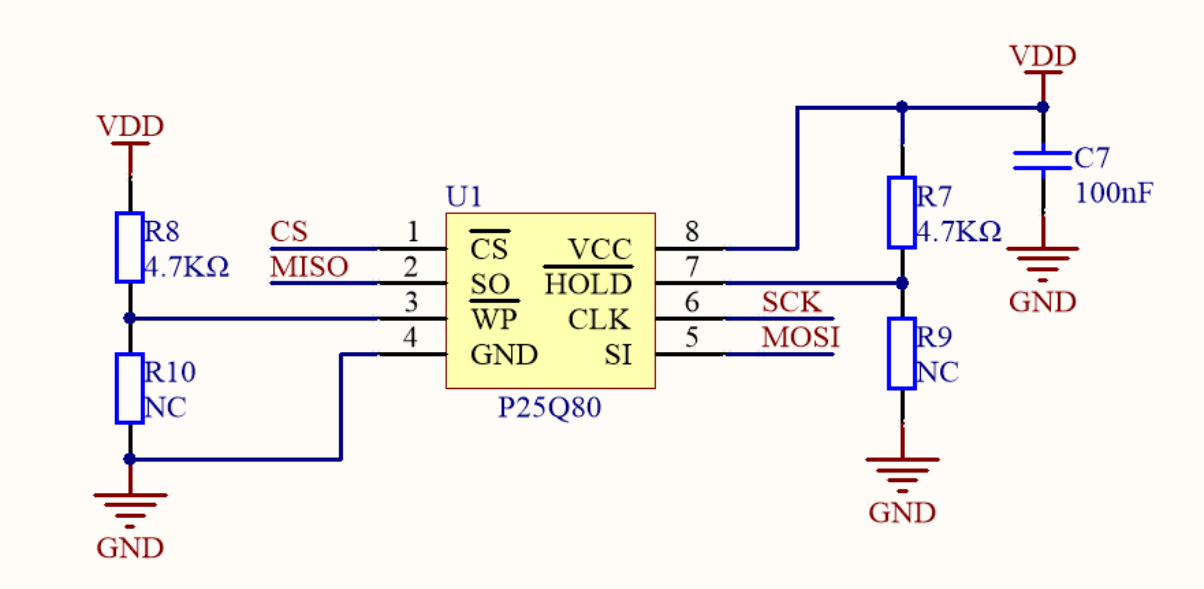
为使用此模块，需要使用跳线帽短接排针 J13 的 5 对引脚：P03 连接到 CLK、P30 连接到 MOSI、P11 连接到 RST、P10 连接到 DC、P02 连接到 CS，如下图所示：



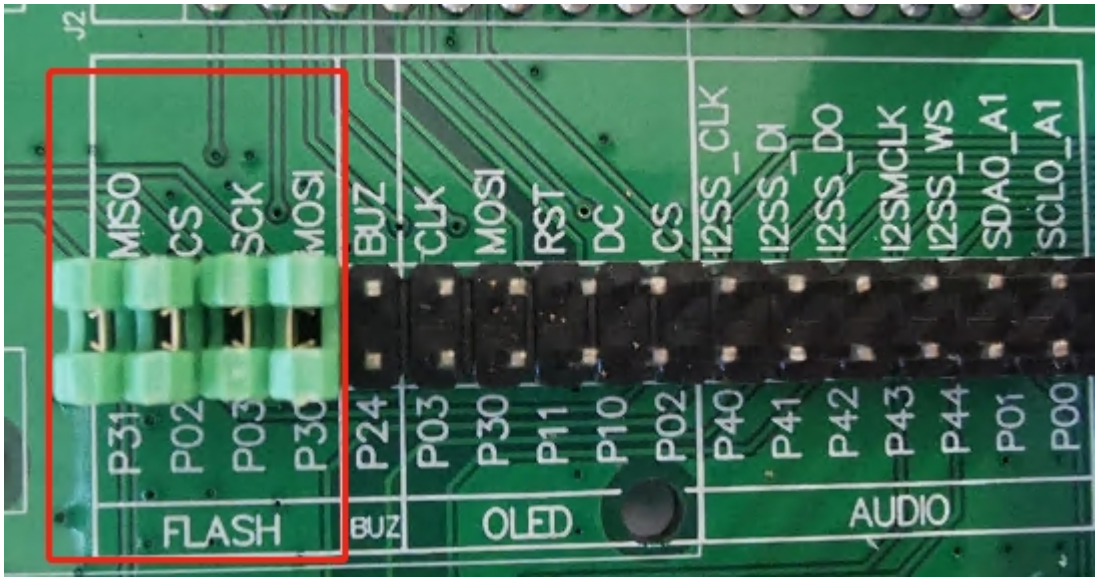
注：OLED 显示屏模块与板载 SPI FLASH 芯片共用主控芯片的 SPI 接口。

2.9 外部 SPI FLASH

开发板搭载了具备 8Mbit 存储空间的外部 FLASH 芯片 GD25WQ80，该芯片与板载显示屏模块共用主控芯片的 SPI 接口，其原理图如下：



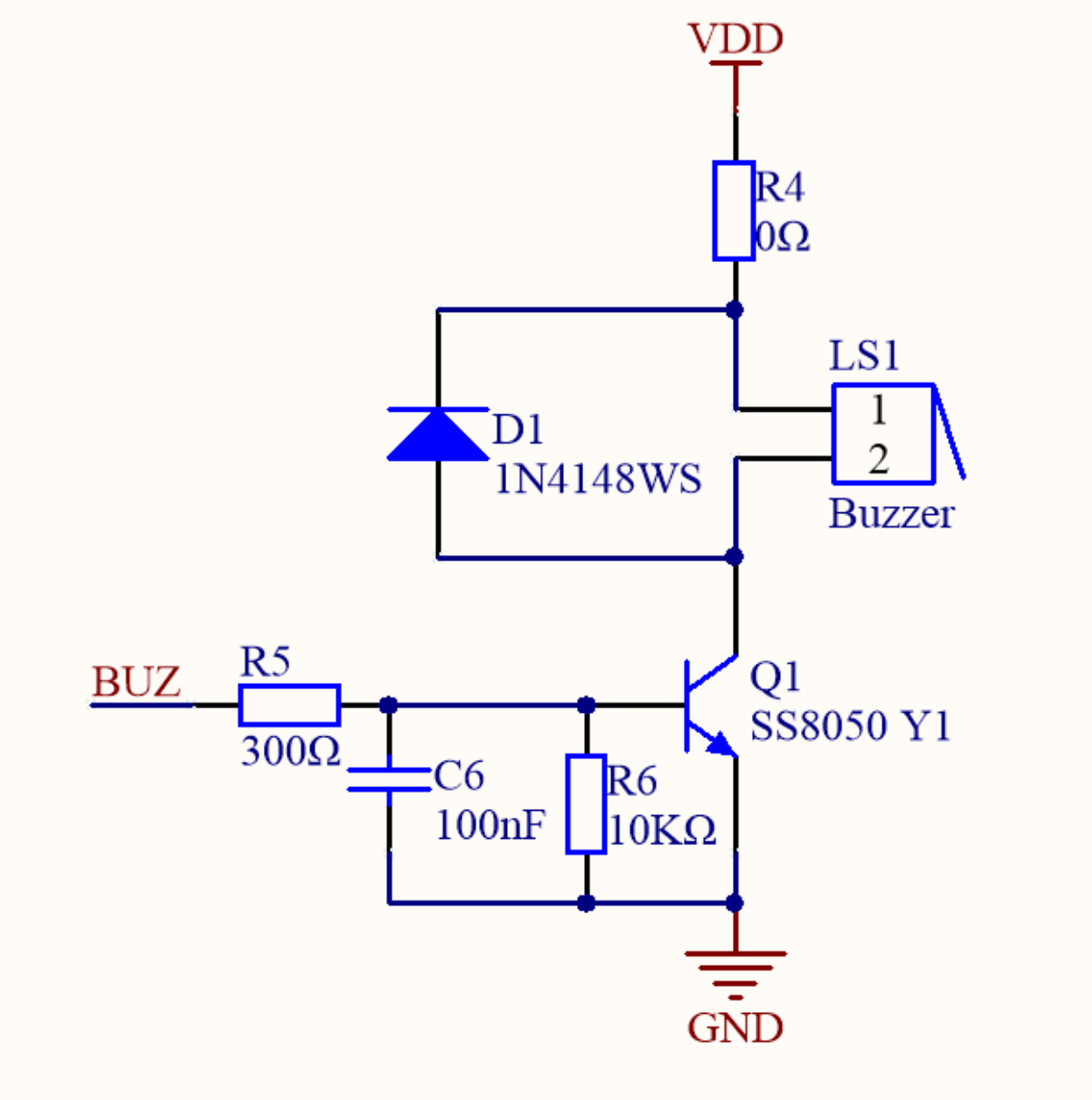
为使用时此模块，需要使用跳线帽短接排针 J13 的 4 对引脚：P31 连接到 MISO、P02 连接到 CS、P03 连接到 SCK、P30 连接到 MOSI，如下图所示：



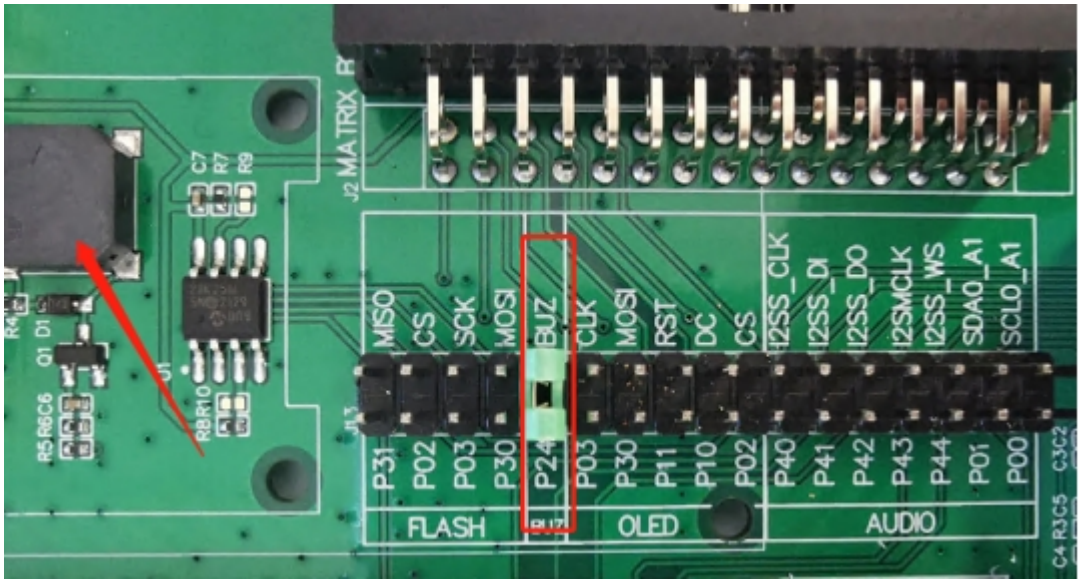
注意：某些 EVB 底板上实际焊接的可能是 SPI 接口的外部 SRAM 芯片，使用时请注意区分。

2.10 蜂鸣器

开发板搭载了贴片无源蜂鸣器电路用于声音提示、报警等功能，可由 PAN1080 SoC 通过 PWM 输出 2KHz~3KHz 频率的方波控制发声，其原理图如下：

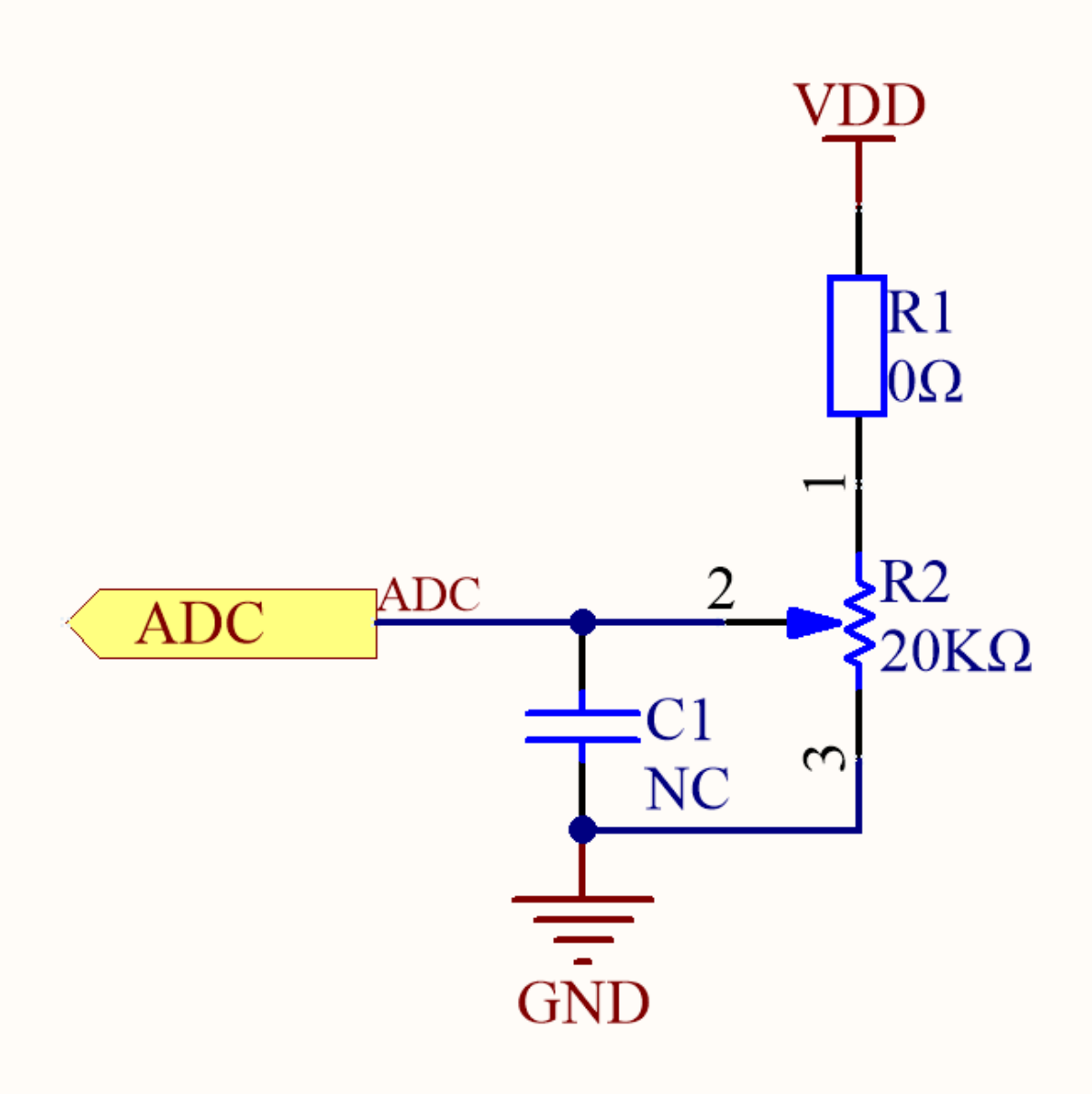


为使用此模块，需要使用跳线帽短接排针 J13 的一对引脚：P24 连接到 BUZ，如下图所示：

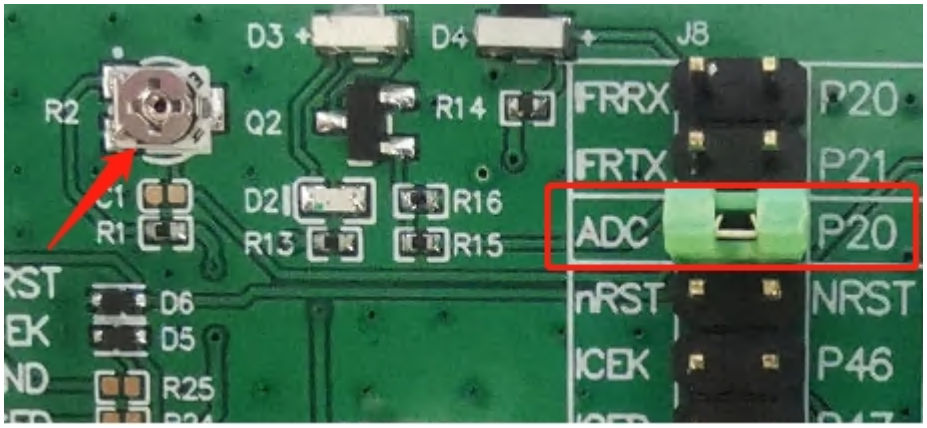


2.11 可调电阻

开发板搭载了一个最大阻值为 20K 的可调电阻与 0 欧姆精密电阻串联接入电源电路，在 LDO 提供 VDD 电源的系统中，可在 ADC 采样点产生 0V~ VDDV 的可调电压，用于测试 PAN1080 SoC 的 ADC（模数转换器）采样功能，其原理图如下：



为使用此模块，需要使用跳线帽短接排针 `J18` 的一对引脚：`P20` 连接到 `ADC`，如下图所示：

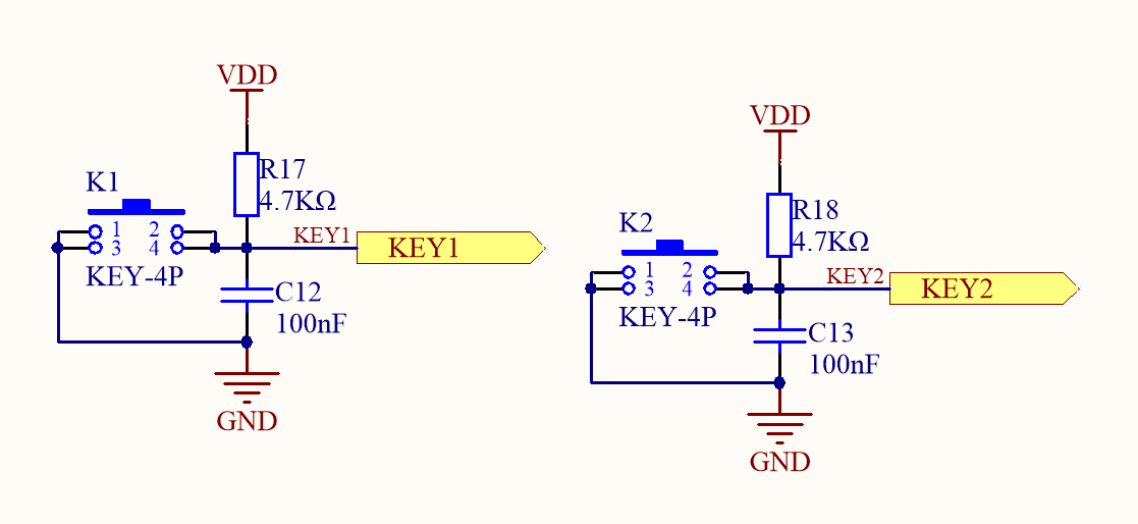


2.12 轻触按键

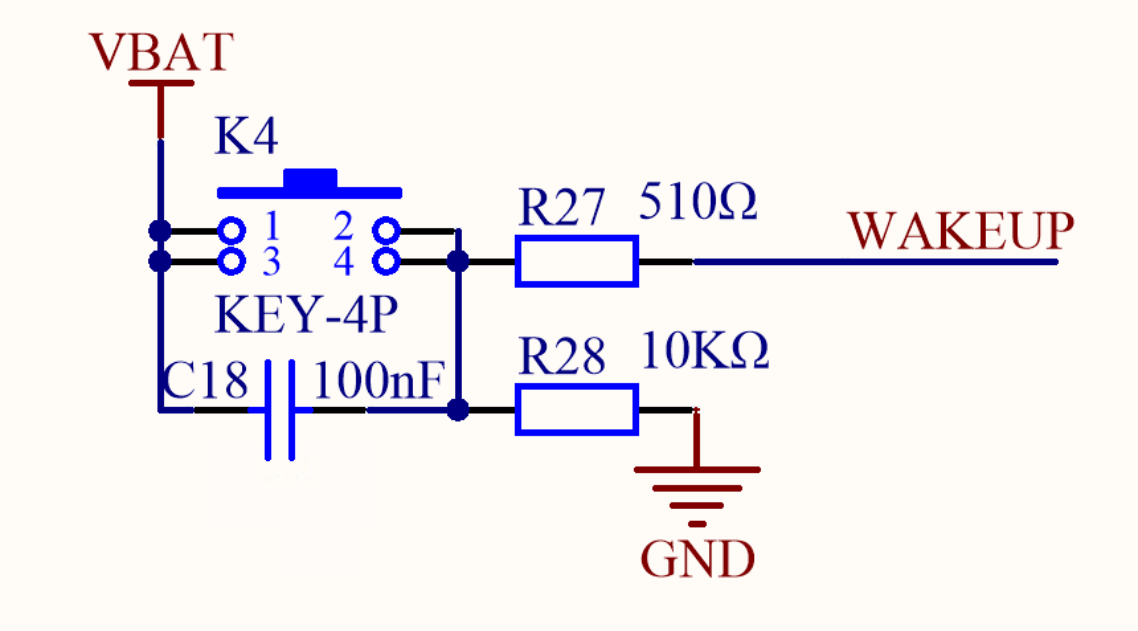
开发板底板配备了 4 个按键：2 个普通按键、1 个低功耗唤醒按键和 1 个复位按键。其中：

- 按键 `K1`、`K2` 可通过跳线帽连接至 PAN1080 SoC 的 GPIO 口 `P04`、`P05`，当做普通按键使用；
- 按键 `K3` 可通过跳线帽连接至 PAN1080 SoC 的 `RST` 引脚，用于控制芯片复位；
- 按键 `K4` 可通过跳线帽连接至 PAN1080 SoC 的 `P56` 引脚，在 PAN1080 SoC 处于待机（Standby）模式下时，P56 引脚可被配置为低功耗唤醒引脚。

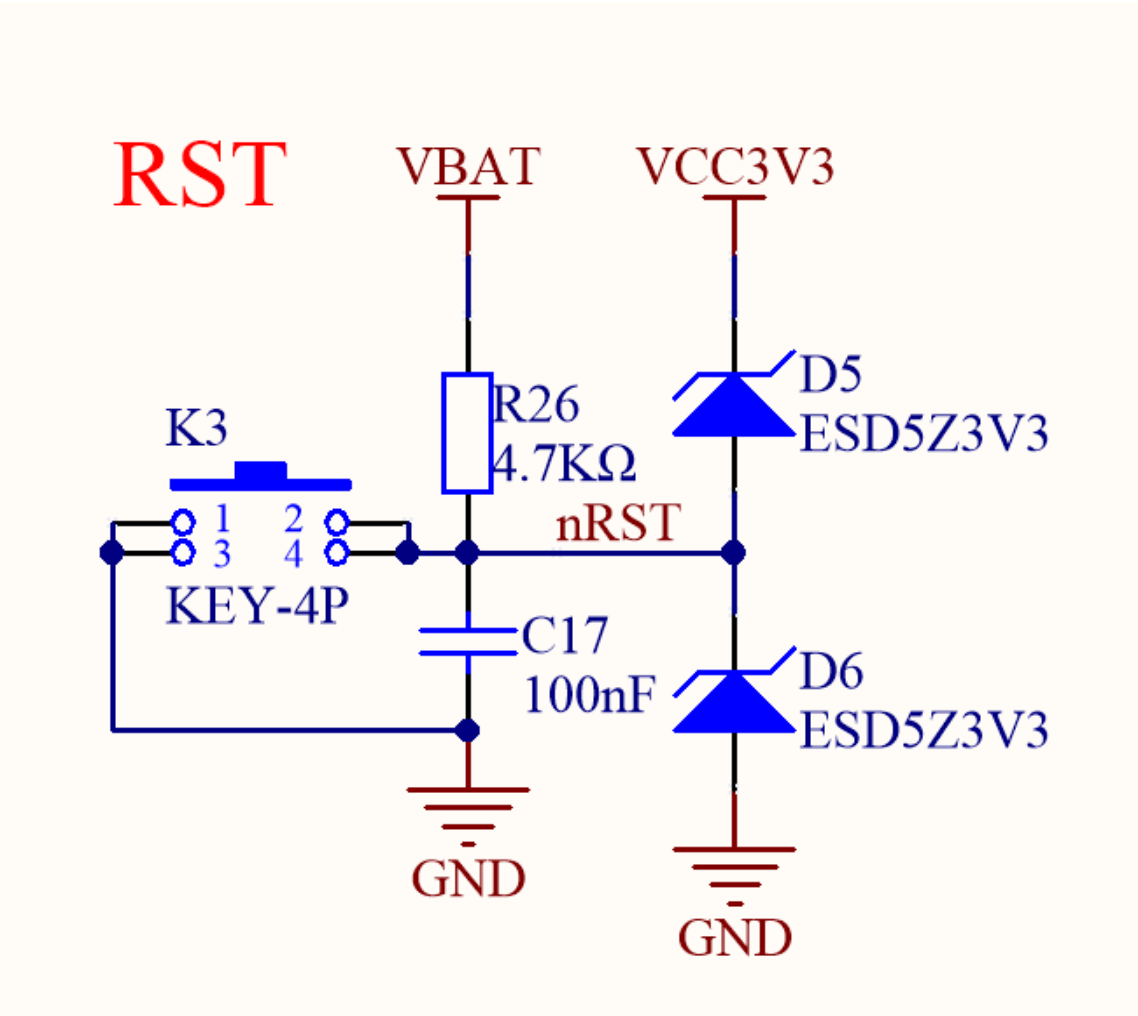
1. 普通按键，原理图如下：



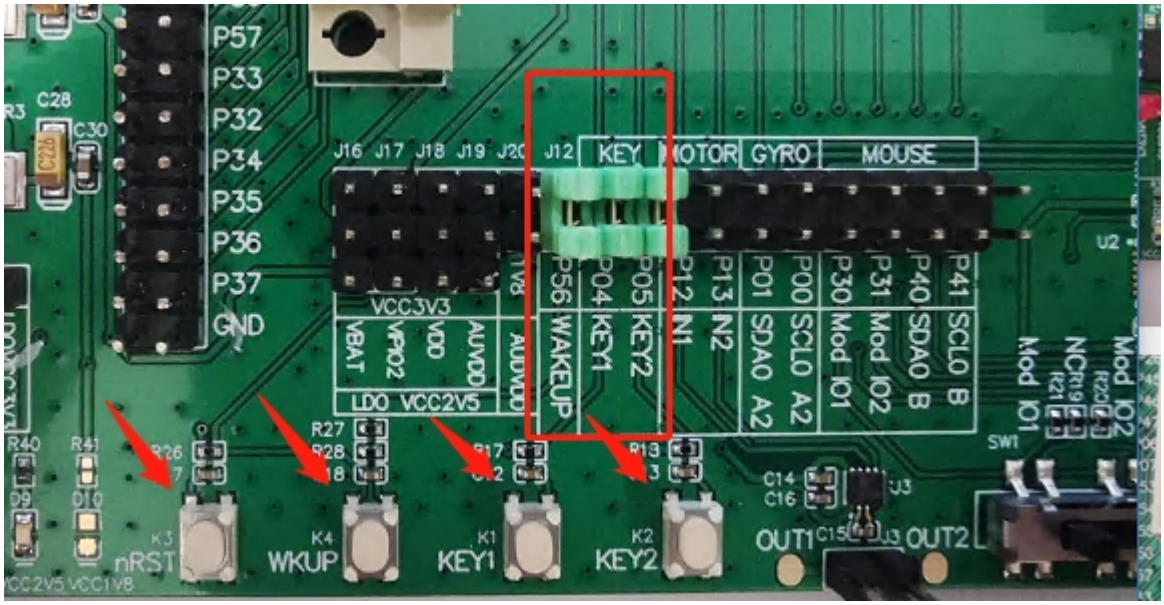
2. 低功耗环境按键，原理图如下：



3. 复位按键，原理图如下：

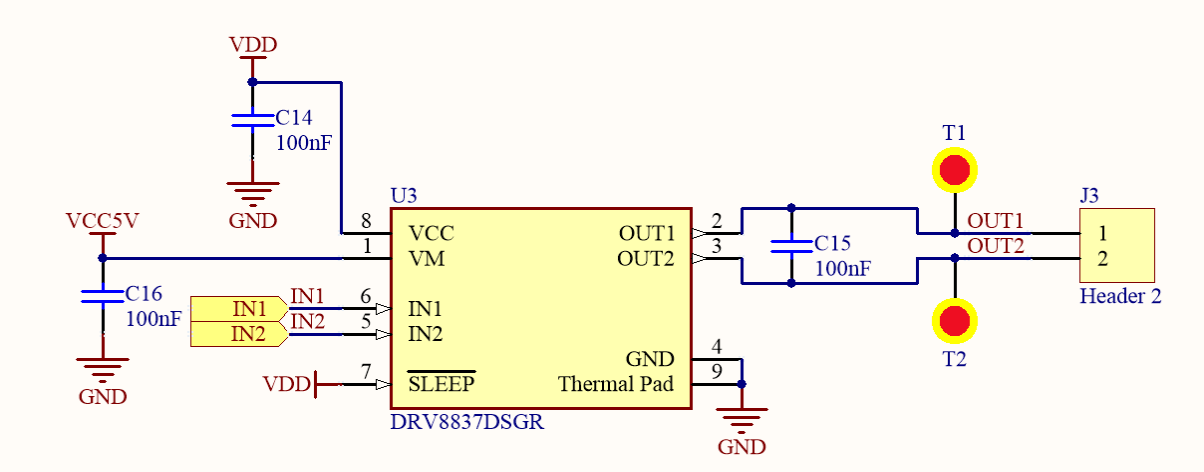


为正常使用所有按键，需要使用跳线帽短接排针 J12 的三对引脚：P56 连接到 WAKEUP、P04 连接到 KEY1、P05 连接到 KEY2，如下图所示：

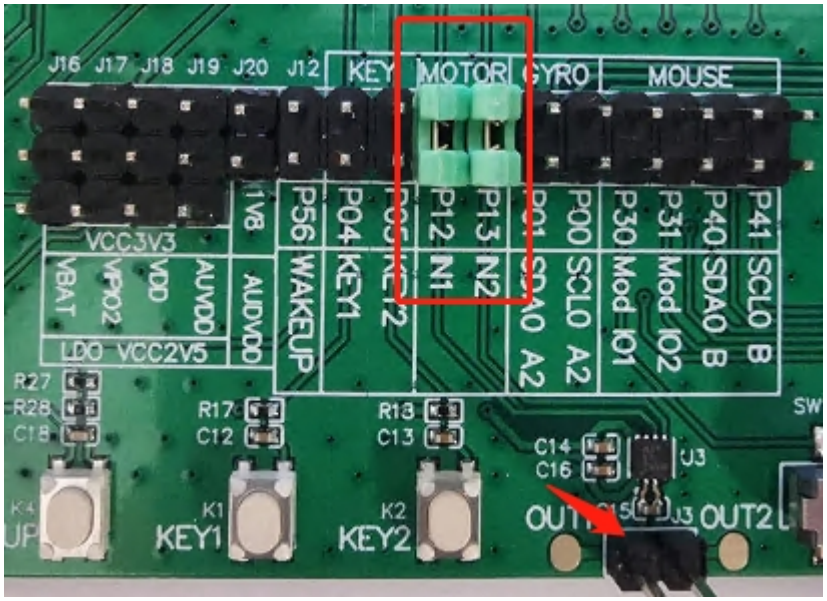


2.13 电机驱动

开发板搭载一颗 TI 超薄小外形尺寸 (2mm*2mm) 的低压 H 桥驱动芯片 DRV8837，可用于驱动一个直流电机或其他负载，其具备 PWM 输入接口及休眠控制引脚，能够提供高达 1.8A 的驱动电流，同时提供用于过流保护、短路保护、欠压闭锁和过热保护的内部关断功能，其原理图如下：



为使用此模块，需要使用跳线帽短接排针 J12 的两对引脚：P12 连接到 IN1，P13 连接到 IN2，并在 J3-OUT1/OUT2 连接诸如直流电机或螺线管等负载器件，然后在程序中使用 GPIO 或 PWM 等控制驱动器的输出，如下图所示：

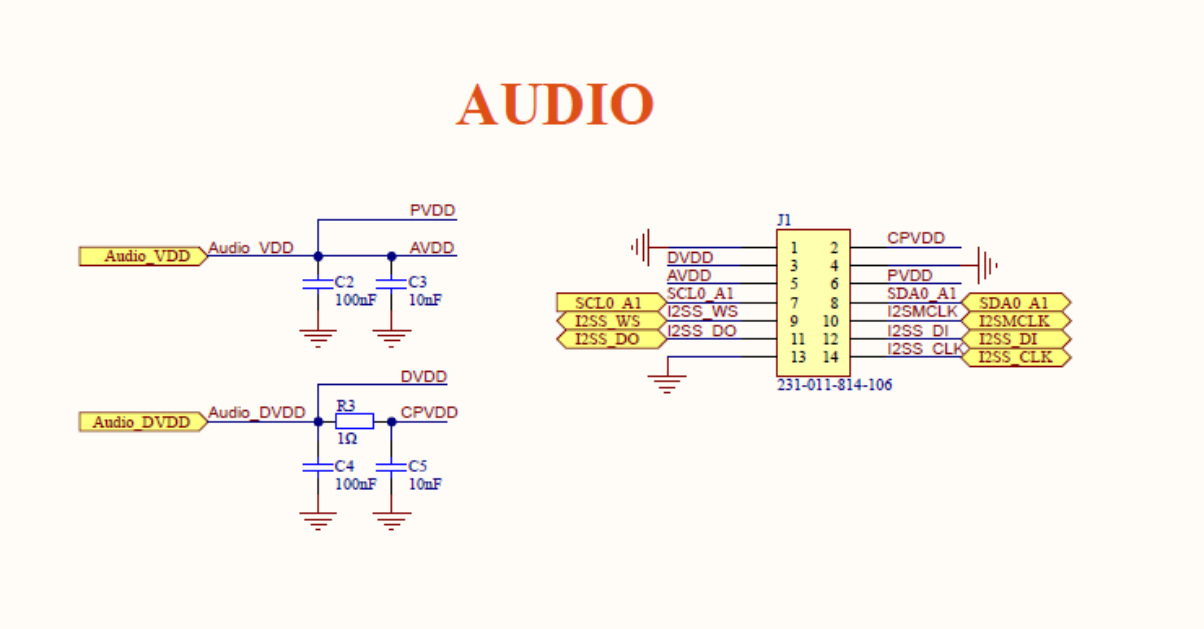


注意:

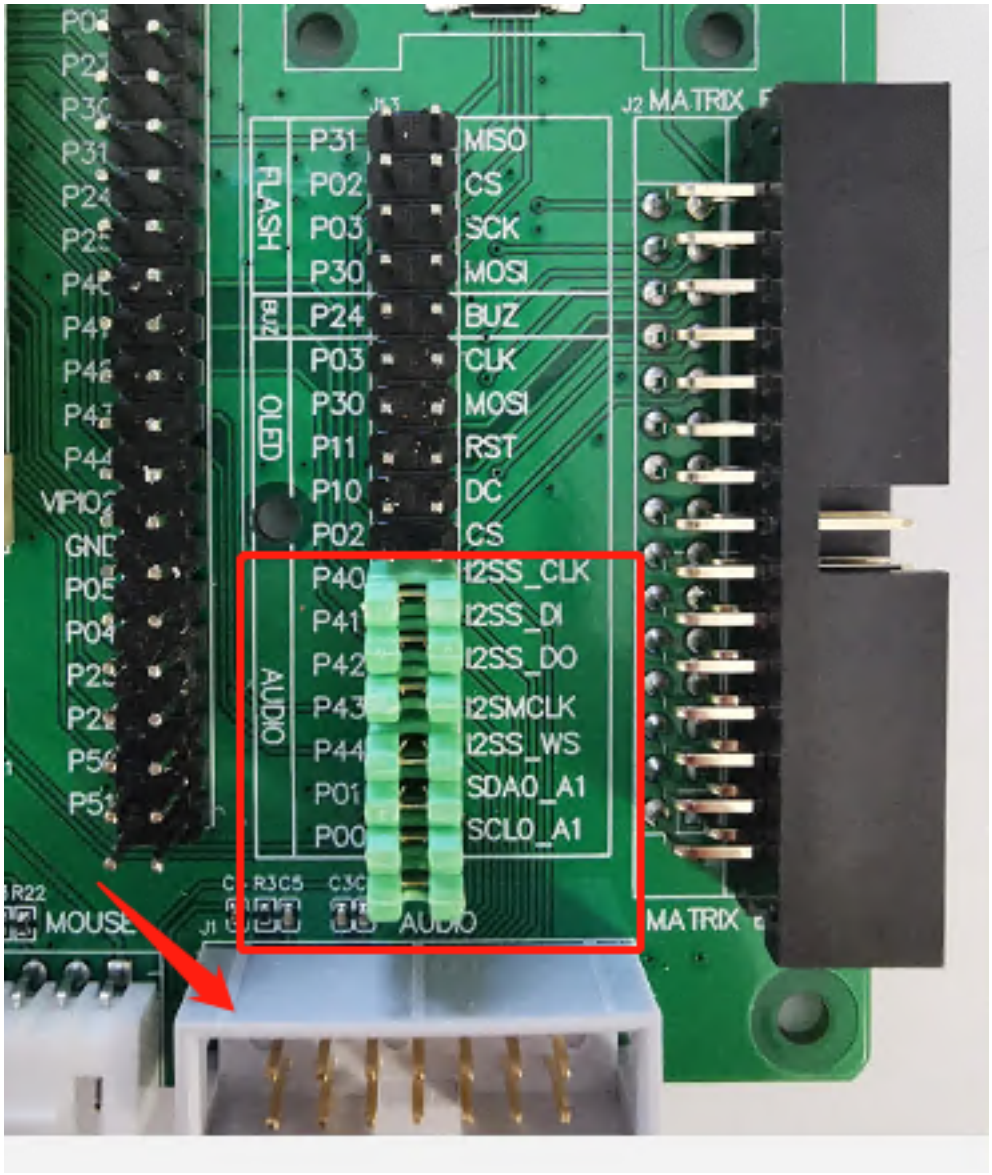
1. 该器件仅可在 USB 供电方式下使用;
2. 由于 PAN1080 SoC 的 P12、P13 两个引脚默认连接了 EVB 核心板上的低速晶振，且通过 0Ω 电阻与 SoC 连接；因此，底板上的元器件在使用这两个引脚之前，需要先将核心板上与 P12、P13 连接的两个 0Ω 电阻拆掉，并换个方向焊接。

2.14 音频接口

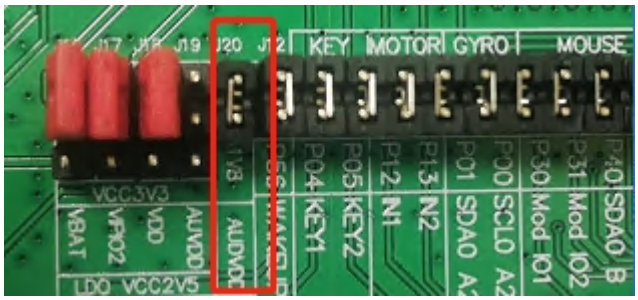
开发板预留 I2S Slave 和 I2C 接口，用于接入第三方 CODEC 芯片进行音频应用的开发，其原理图如下:



为使用此模块，需要使用跳线帽短接排针 J13 的 7 对引脚：P40 连接到 I2SS_CLK、P41 连接到 I2SS_DI、P42 连接到 I2SS_DO、P43 连接到 I2SS_MCLK、P44 连接到 I2SS_WS、P01 连接到 SDA0_A1、P00 连接到 SCL0_A1，如下图所示:

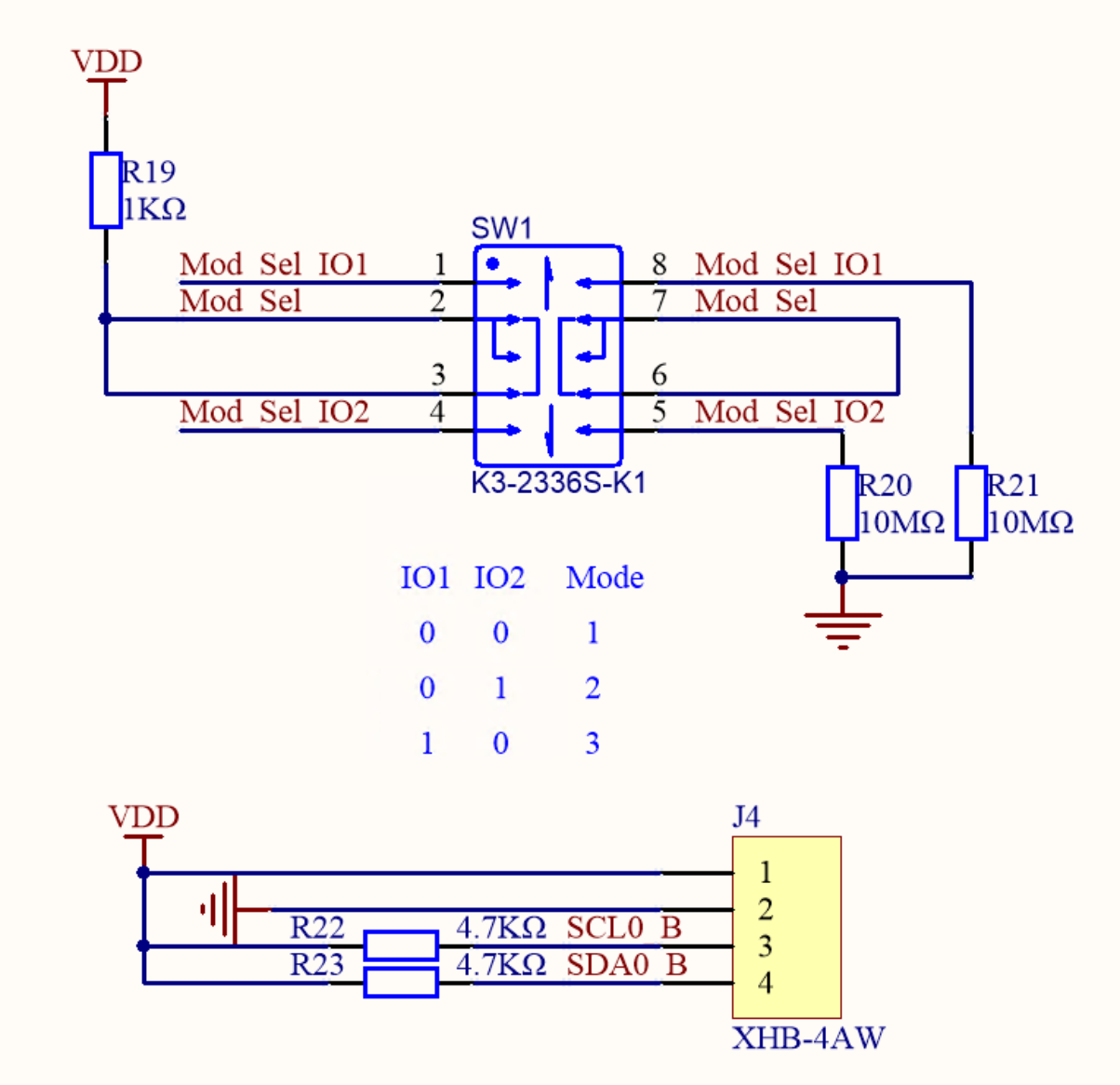


需要注意的是，CODEC 芯片一般需要 1.8V 供电，为此还需要使能 EVB 上的 1.8V 电源，方法是将 J20 使用跳线帽短接，将 1V8 电源灌入 AUDVDD，如下图所示：

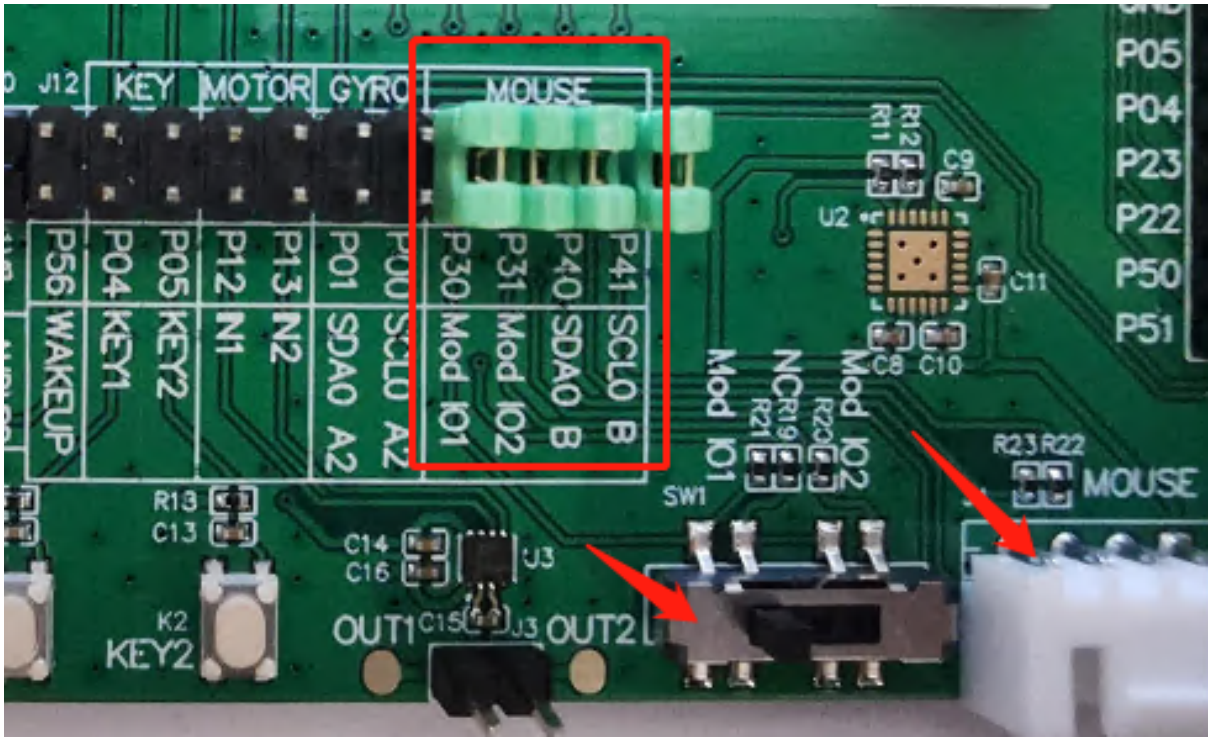


2.15 鼠标接口

开发板支持多模（USB/BLE/2.4G）鼠标开发，为此搭载有一个拨动开关和一个 I2C 接口，其中 P30、P31 可通过拨动开关可以切换 3 个状态，模拟鼠标切换三模模式，其原理图如下：

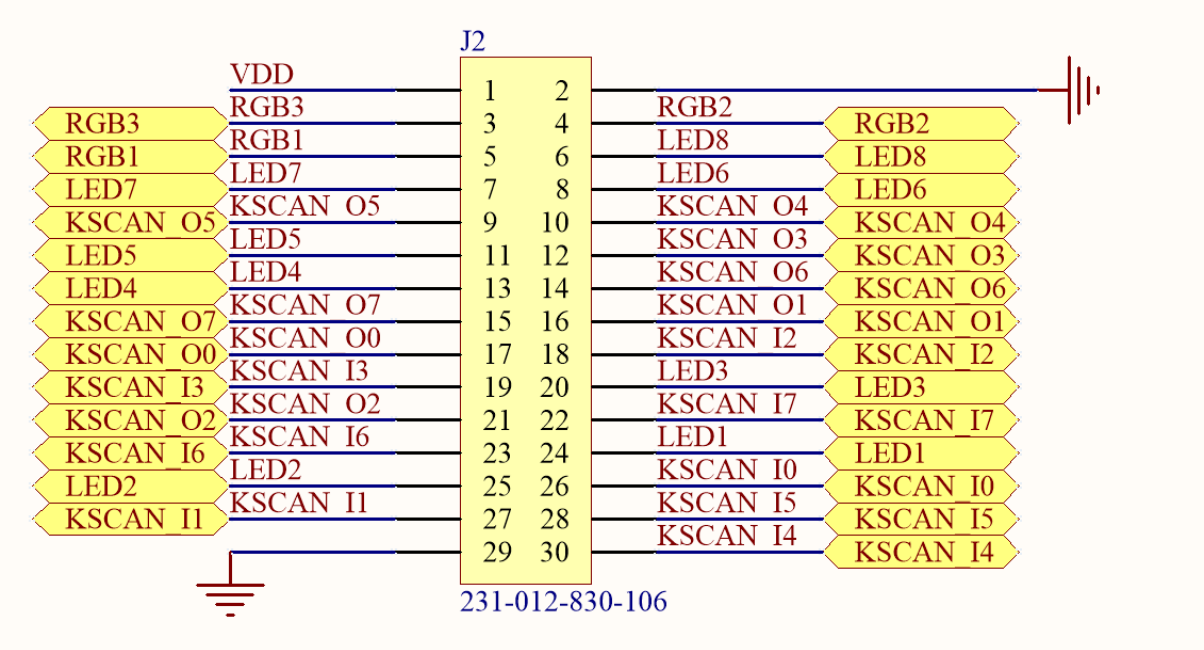


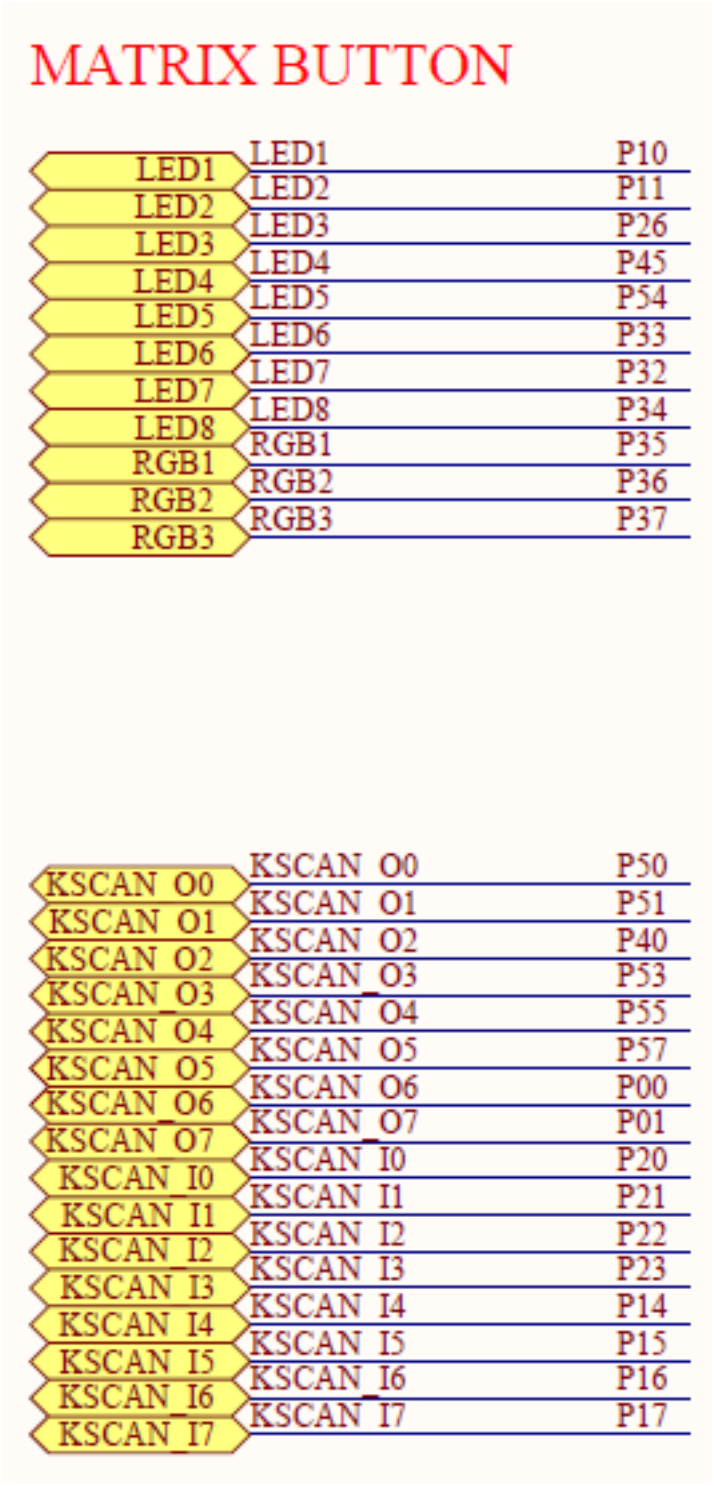
为使用此模块，需要使用跳线帽短接排针 J12 的 4 对引脚：P30 连接到 MOD IO1、P31 连接到 MOD IO2、P40 连接到 SDA0_B、P41 连接到 SCL0_B，如下图所示：



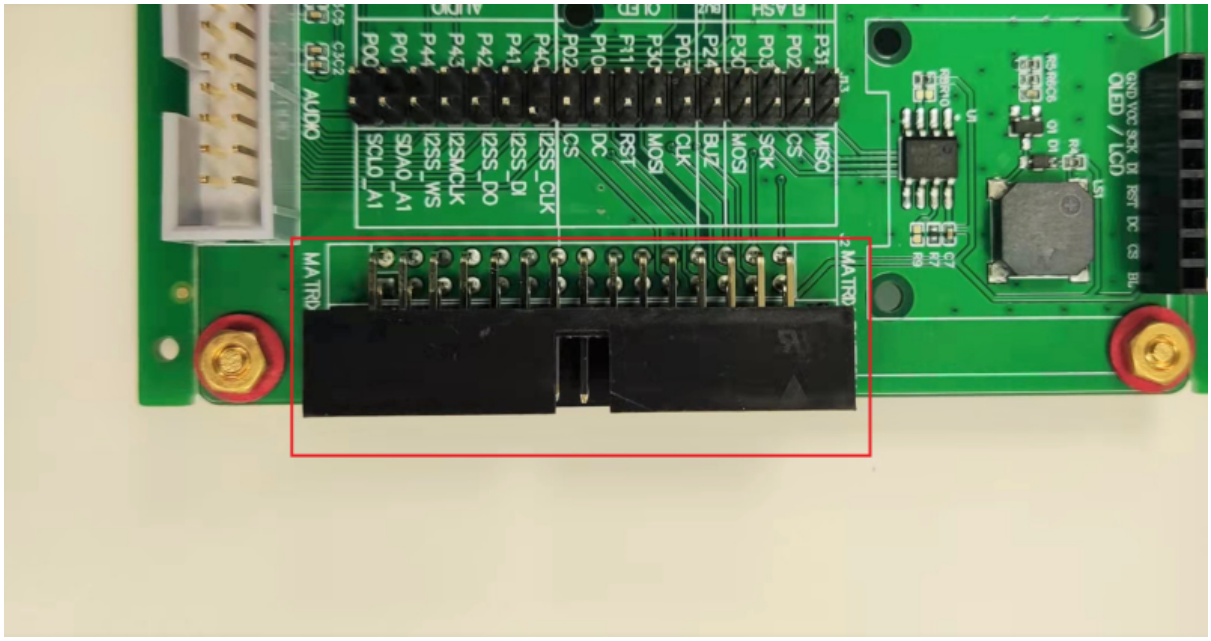
2.16 键盘接口

开发板支持多模（USB/BLE/2.4G）键盘开发，为此预留 1 个 8x8 LED 矩阵键盘接口，可通过 SoC 硬件 KeyScan 模块实现矩阵按键扫描，通过 SoC 硬件 PWM 模块实现 LED 呼吸灯灯效，其原理图如下：



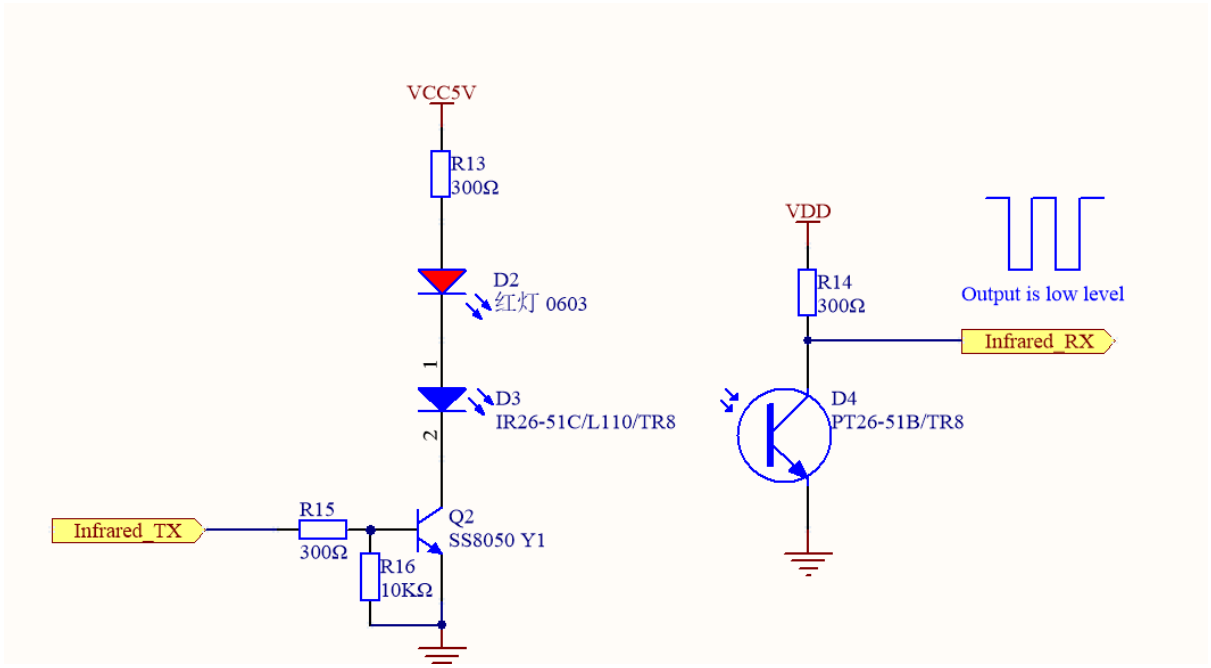


此模块接口直接将 PAN1080 SoC 通过 MATRIX BUTTON 插座与外部矩阵键盘模块连接，因此无需跳线，使用前只需保证无引脚冲突即可，如下图所示：



2.17 红外模块

开发板搭载了一个红外收发模块，包括一个发射电路和一个接收电路，其中发射电路还串联一个红色可见光 LED，可当做普通 LED 电路使用，其原理图如下：



为使用此模块，需要使用跳线帽短接排针 J8 的两对引脚：P20 连接 IFR RX，P21 连接 IFR TX，如下图所示：

2.1.3 3 更多信息

1. PAN1080 SoC 产品说明书，请查阅 PAN1080 Development Kit`04_DOC/05_soc_manual` 目录下的相关内容；
2. PAN1080 SoC 板级硬件参考设计，请查阅这里；
3. PAN1080A EVB 开发板的电路图，请查阅 PAN1080 Development Kit`02_HDK` 目录下的相关内容；

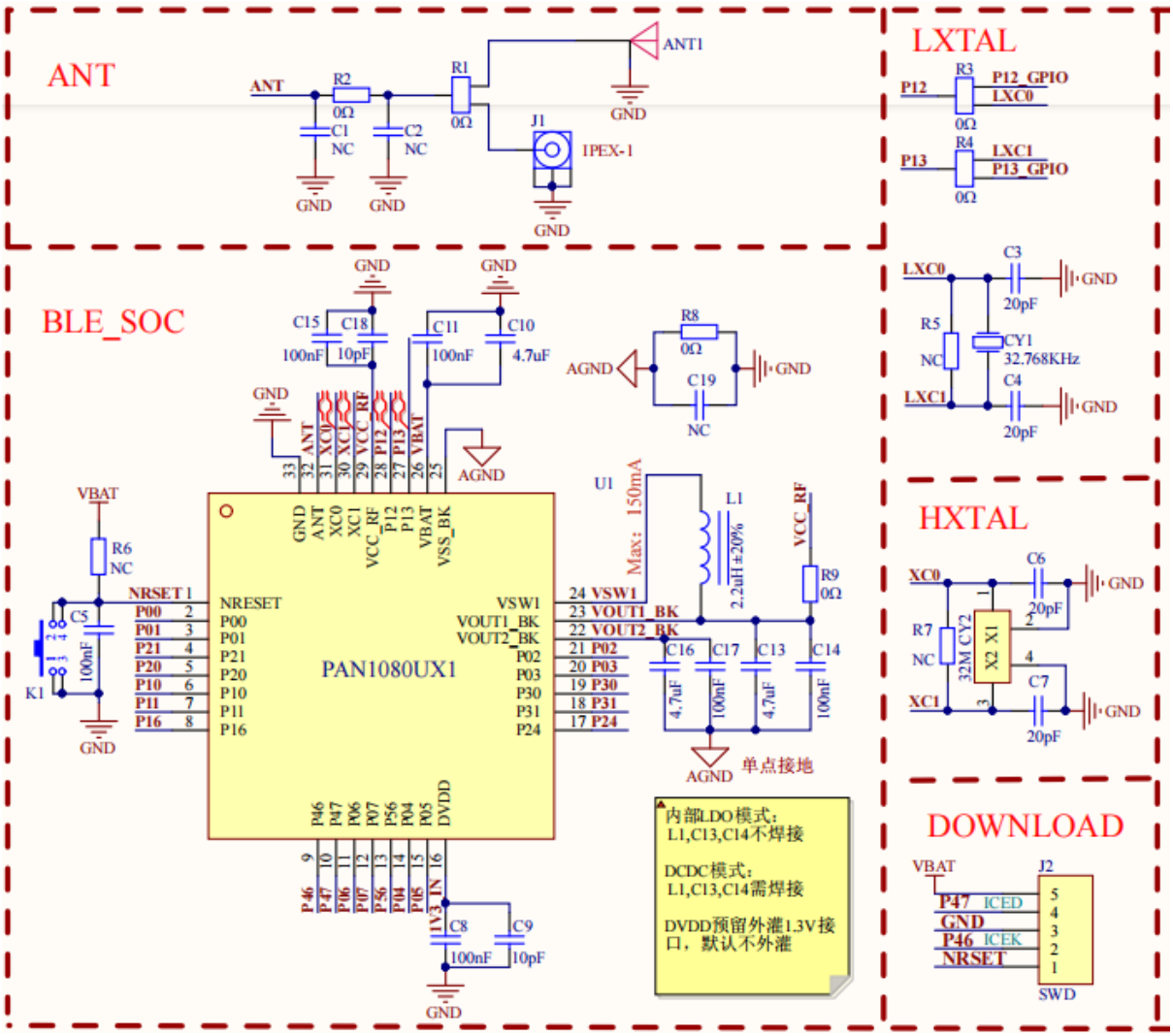
2.2 PAN1080 硬件参考设计

2.2.1 1 概述

本文档主要介绍 PAN1080UX1/PAN1080LX5 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 PAN1080UX1/PAN1080LX5 芯片的硬件设计方法。

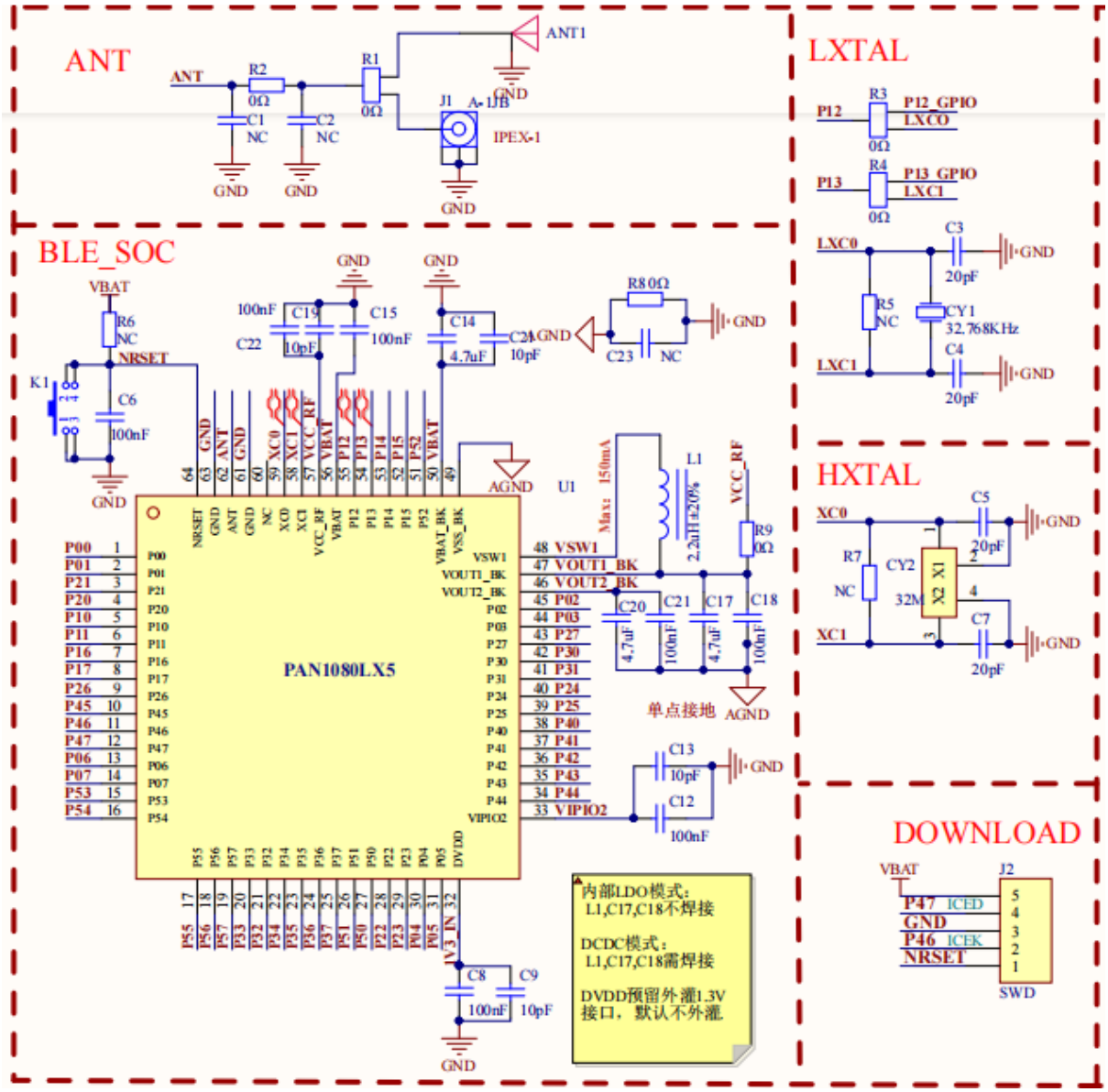
2.2.2 2 原理图设计建议

2.1 PAN1080UX1 参考设计原理图



如上图电路系统由电源去耦电容、DCDC 降压、晶振电路、天线匹配网络组成。

2.2 PAN1080LX5 参考设计原理图



2.3 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 。
- VCC_RF、VBAT、VOUT2_B、KVOUT2_BK 至少预留两个电容，靠近芯片管脚。

2.4 DCDC

DCDC 电路外围为 L1、C13、C14。要求 L1 DCR 小于 80mΩ，峰值电流至少为 150mA。芯片电源有两种工作模式，DCDC 模式和内部 LDO 模式。开启 DCDC 可以节省功耗。开启 LDO 模式后芯片内部将 VBAT 连接到 VOUT1_BK, 这时 VOUT1_BK 处的一个电感和两个电容可以不焊接。

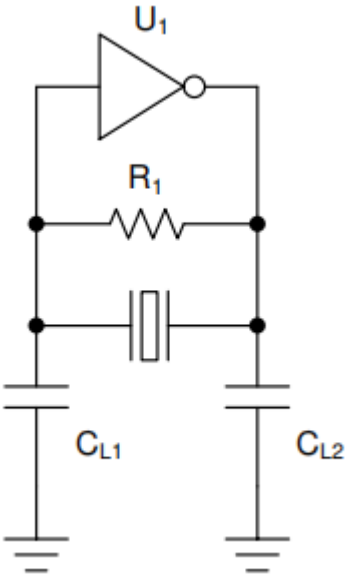
2.5 DVDD

DVDD 需要预留外灌 1.3V 接口，且外挂 100nF 和 10pF 电容。1.3V 外灌默认不焊接，是否外灌和软件应用运行功耗有关。如果外灌则要求供电压 1.3V，驱动电流大于 20mA，纹波小于 2%。

2.6 VIPIO2

- VIPIO2 电源脚在 LQFP64 封装上有独立的 PIN 脚。因此 PAN1080LX5 可配置两组不同的 IO 输出电平。P40、P41、P42、P43、,P44 的输出电平等于电源 VIPIO2 的供电电压。VIPIO 输入电压范围为 1.8V–VBAT。其余 IO 的输出电平等于 VBAT 供电电压。
- VIPIO2 电源脚在 QFN32 封装上和 VBAT 脚绑定在一起。因此 PAN1080UX1 IO 输出电平等于 VBAT。

2.6 32M **晶振**



- 上图这种振荡器，晶体和负载电容器构成 型滤波器，为内部放大器提供 180° 相移，同时使振荡器一直锁定在指定的频率。为了使该频率正确，必须根据晶体的容性负载 (CL) 参数正确地确定负载电容的尺寸。可以通过相对于晶体的所需负载电容 CL 正确确定负载电容器的尺寸来设置 32MHz 晶体振荡器的频率。从晶体的角度而言，两个电容器串联放置，这意味着必须使用用于计算最终总电容的“电阻器并联”方程。还要注意 PCB 迹线和焊盘会增加一些寄生电容。可以通以下公式来计算正确的负载电容值。

$$CL = \frac{C1 \times C2}{C1 + C2} + C_{parasitic} \approx \frac{\text{load capacitor value}}{2} + C_{parasitic}$$

  最后的简化要求 C1 和 C2 相等，Cparasitic =5pF 。

- 上图中 C1、C2 为高速晶振的负载电容。其参数将影响晶振频率，负载电容的选择请参考所选晶振的规格书。
- 晶振推荐如下：

  1）晶体频率 132MHz；

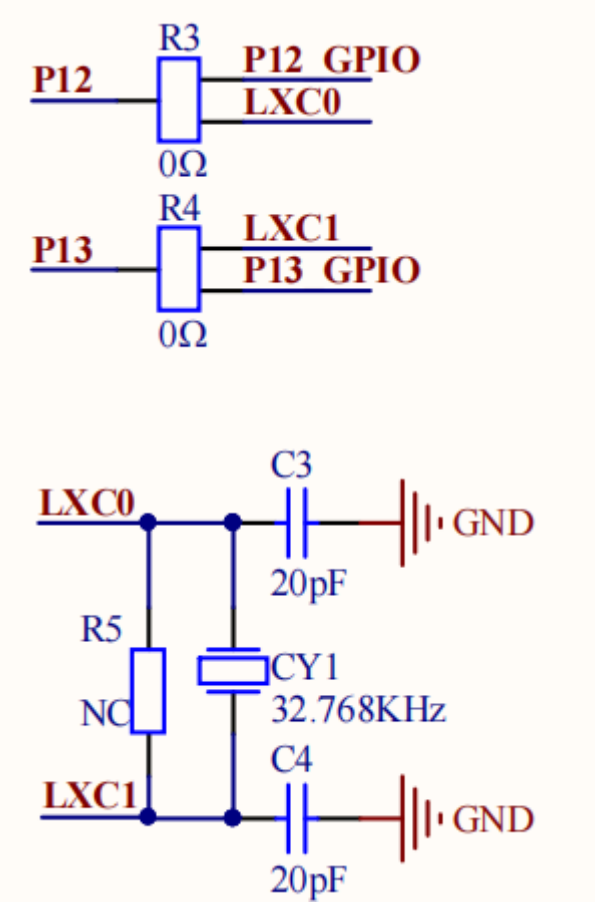
  2）ESR 小于等于 60ohm；

  3）晶体负载电容小于等于 20pF；

  4）晶体频率精度高于 ±20ppm；

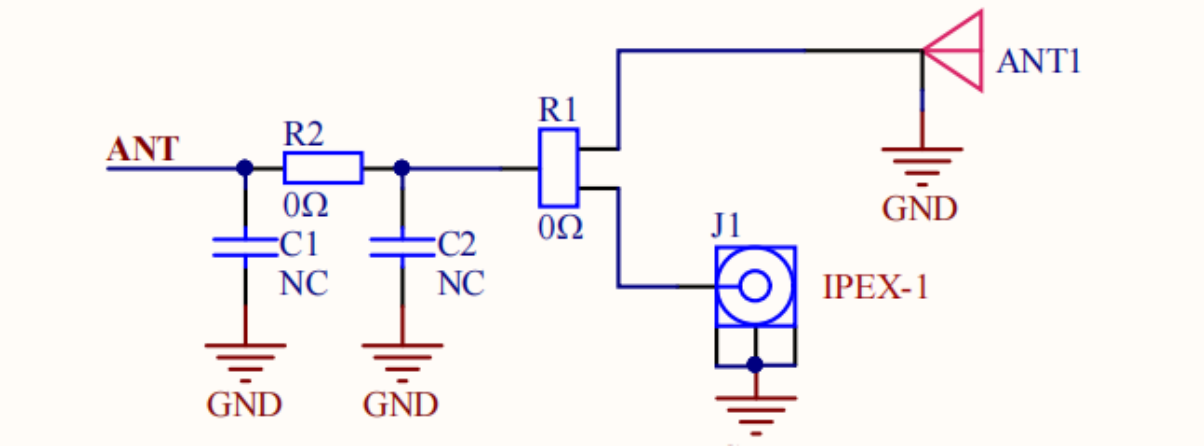
晶振封装形式	晶振负载电容/pF	焊接电容值/pF
3225	9	10
12	12	
20	30	
圆柱	9	10
12	12	
20	30	
49S	9	10
12	15	
20	30	

2.7 32K 晶振



- 低速晶振电路支持外部 32.768KHz 无源晶振。C3、C4 为低速晶振的负载电容；低速晶振推荐用户选择 ESR<80KΩ 的晶振。
- 由于部分用户需要使用外部 32K 的芯片管脚复用为 IO 使用，所以做以下电路进行预留接口。通过两个 0 欧姆电阻将 P1.2、P1.3 复用为晶振接口和普通 IO。

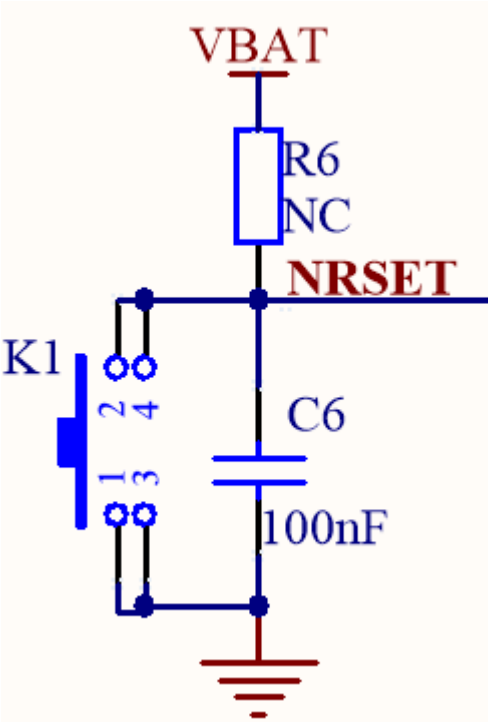
2.8 天线匹配



- 由于芯片内部已做射频前端匹配电路，所以此处预留射频前端匹配网络电路，其中 C1、C2、R2 预留元件位置，需焊接 R2 为 3.9pF 电容，R1 为 0Ω 电阻。如果客户设计 PCB 不合理导致射频前端失配，那么可以通过 C1、C2、R1、R2 组成匹配网络进行调整匹配。
- 通过电阻 R1 位置可选择 PCB 天线或 IPEX 座子。

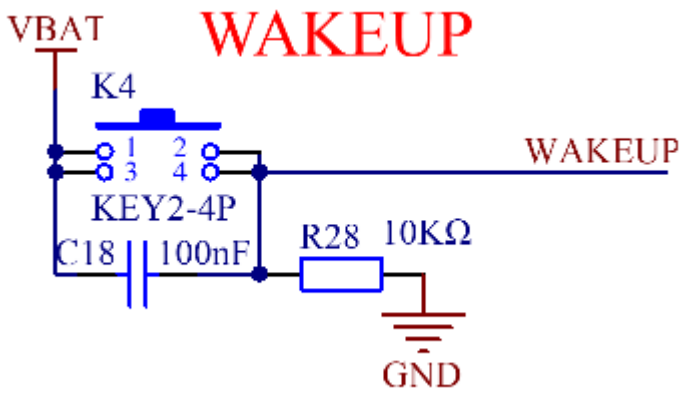
2.9 复位电路

复位电路如图图 1-4 所示，在应用中必须有电容，参数为 100nF。加电容的作用是在系统受到强干扰时，稳定复位脚的状态。



3.0 休眠 M0 按键唤醒方式

在低功耗 Standby M0 模式，仅有一个 IO P56 支持唤醒, IO 默认外部拉低。



2.2.3 3 PCB 设计建议

3.1 制版工艺

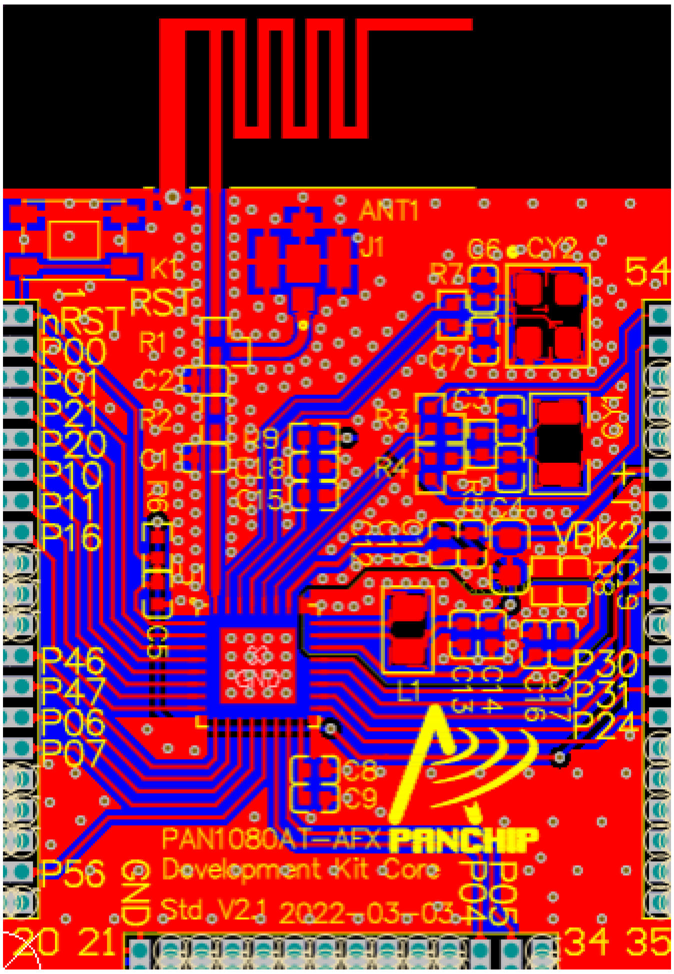
- 本 Guide 主要针对二层板并且单面贴设计，叠层如下图所示。PCB 具体厚度根据实际情况和阻抗要求适当调整。

层		厚度
TOP	================================	1.8(0.5oz+Plating)
	Core	44(mil)
BOT	================================	1.8(0.5+Plating)
完成板厚:1.2(+0.12/-0.12) MM		

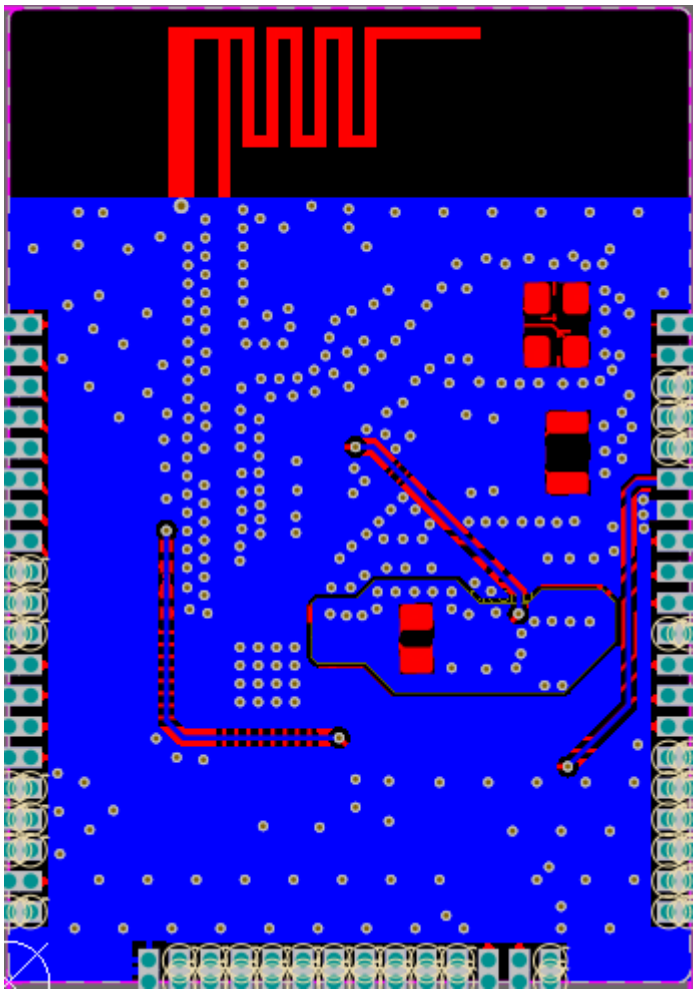
* 线宽推荐如下：

板材属性	参数
PCB 板材	FR4
PCB 板厚	1.2mm
50 欧姆 RF 线宽	20mil
接地铺铜与 RF 走线间距	5.1mil

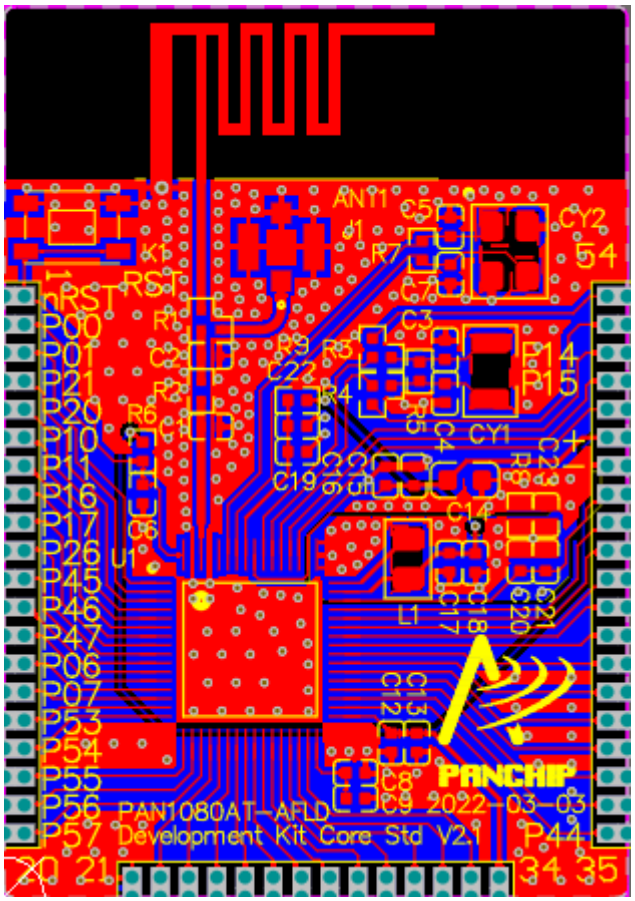
3.2 PAN1080UX1 PCB 顶层布局



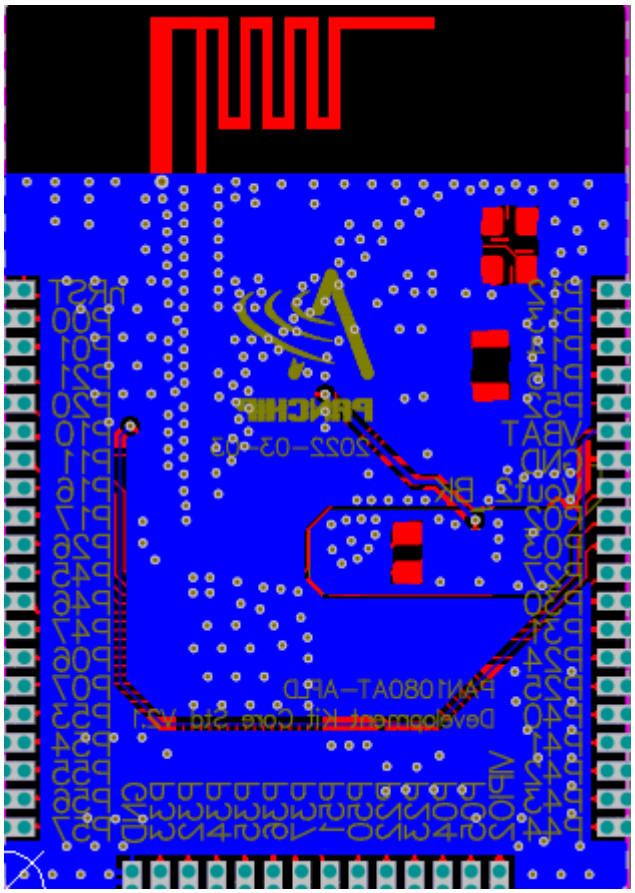
3.3 PAN1080UX1 PCB 底层布局



3.4 PAN1080LX5 PCB 顶层布局



3.5 PAN1080LX5 PCB 底层布局

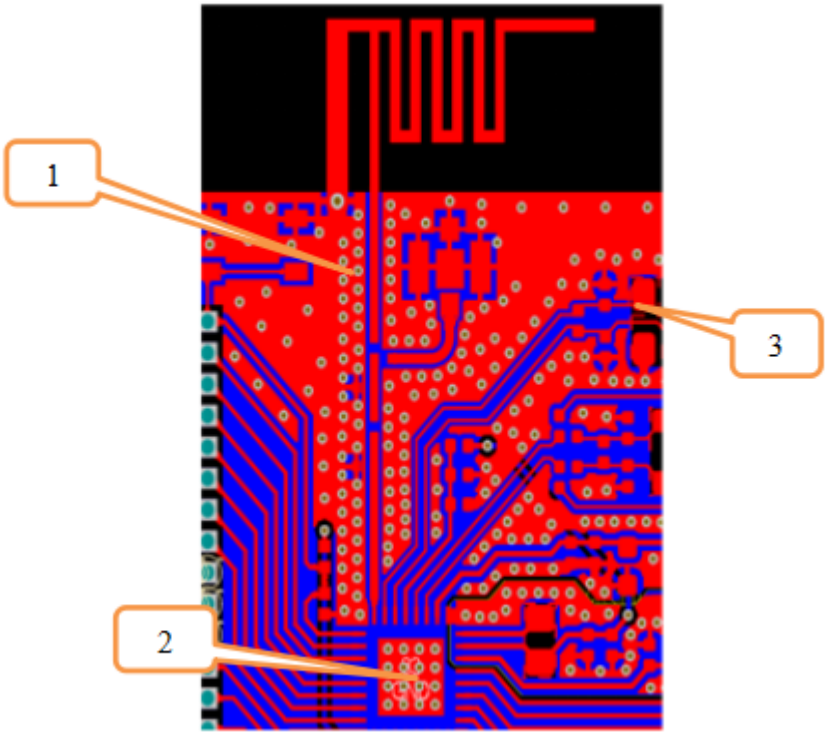


3.6 射频走线注意事项

- 射频匹配链路按照 50Ω 走线，可以参考 TOP 和 BOT 层的 GND 平面，RF 线与焊盘宽度一致，阻

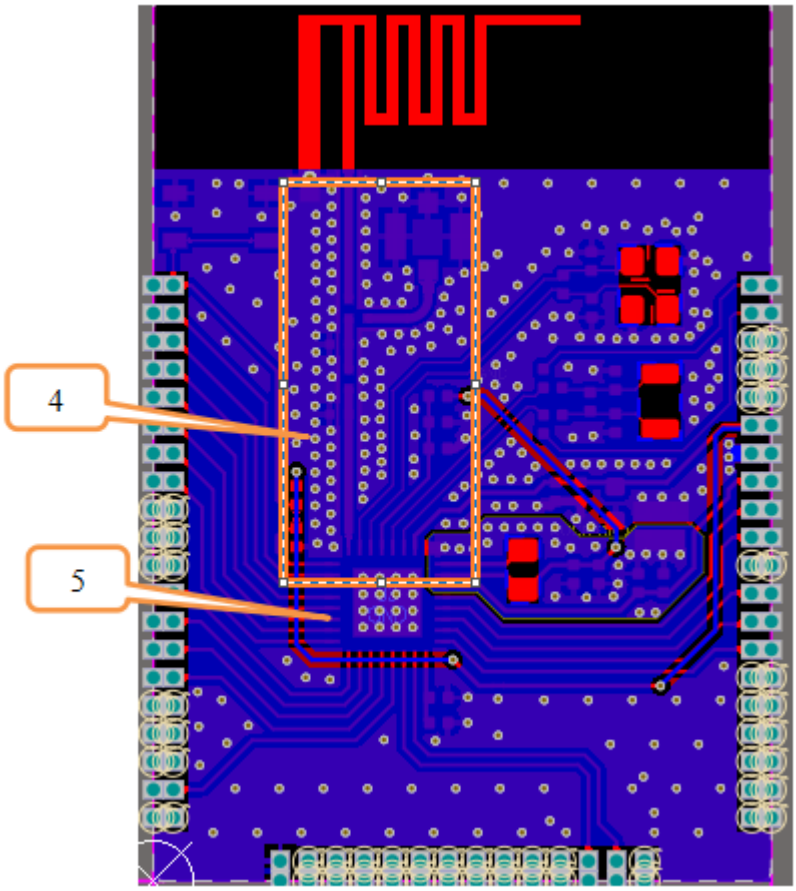
  抗没有突变。
- RF 线有完整的参考地，从 IC 端出来就进行包地处理，两边均匀的打 GND 过孔，底层到芯片底部地平面尽量宽，如标签 1；
- 芯片 EPAD 多打过孔，如标签 2；
- 晶振要远离天线和天线匹配链路，晶振走线和其他走线垂直布线，减少晶振对 RF 的干扰，如标签 3；
- 晶振线两边包地，以降低对电源和 RF 的干扰
- 天线辐射区域尽量保证没有金属器件。
- 天线的 Pi 型匹配电路要走顺，并联元件焊盘和走线重合为好。

  射频链路走线参考如下：



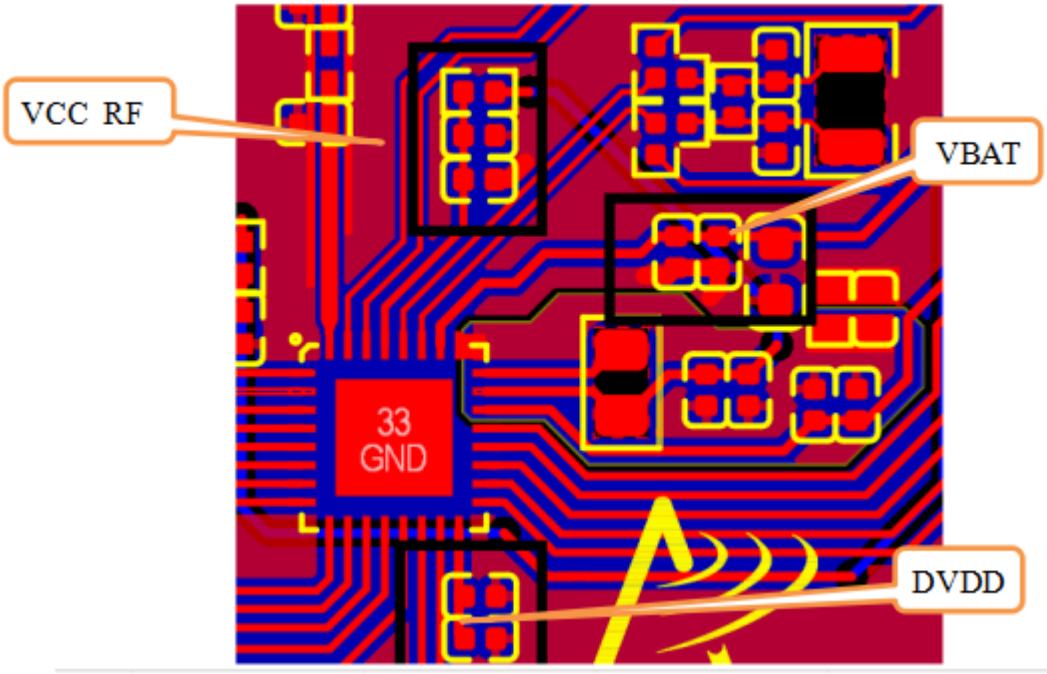
- 天线匹配链路底层不走线，天保证线地回路到芯片最短。天线匹配链路的地和芯片 EPAD 是一块完整平滑的地。如标签 4，橙色方框；
- 芯片底层不要走线，如标签 5；

  射频地线走线如下：

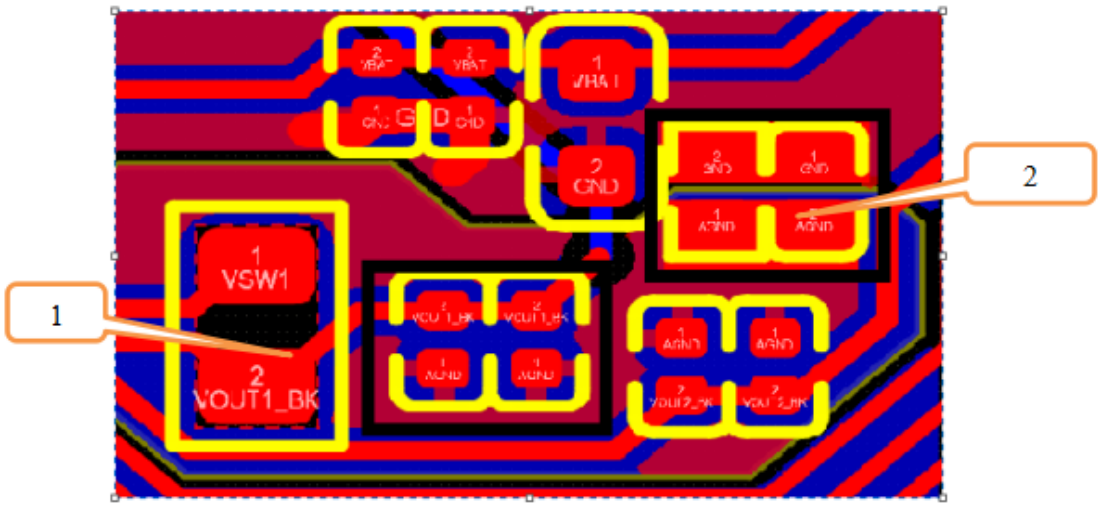


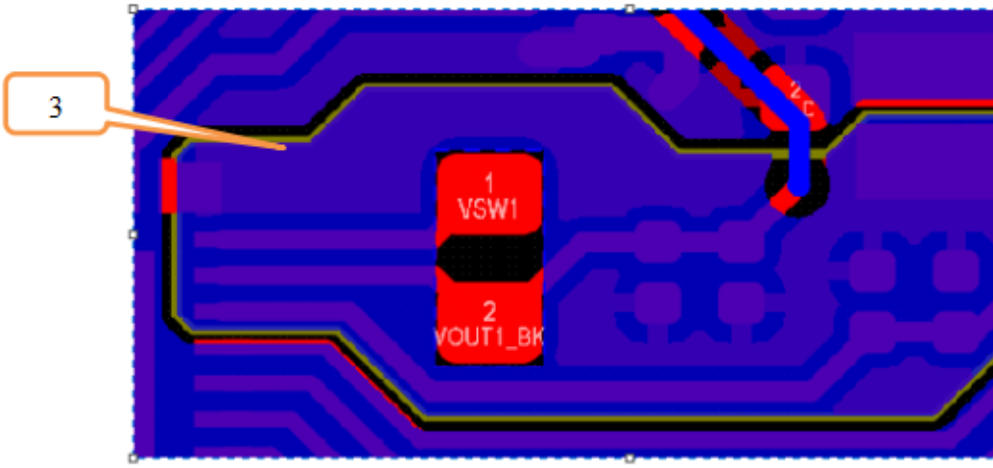
3.7 **电源部分注意事项如下。**

- VCC_RF,VBAT,DVDD 管脚就近放置电容，地回路越短越好。**电源去耦电容布局如下图：**



- DCDC 电感靠近 VSW1 放置，DCDC 相关电容靠近电感放置，如标签 1。DCDC 地和芯片地要分开，通过 0Ω 电阻单点接地，如标签 2。DCDC 地线和芯片地在底层进行分隔，如标签 3。DCDC **单点接地顶层布局如下：**

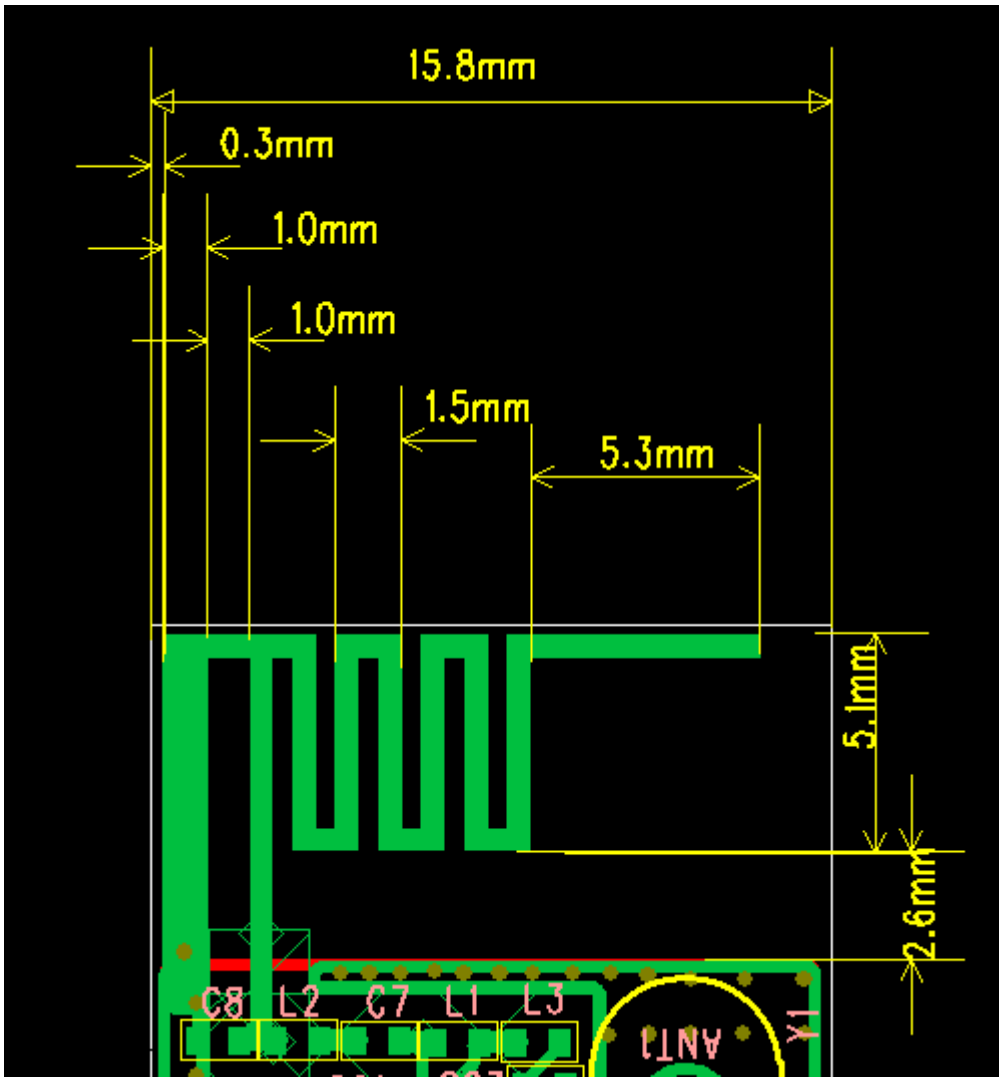




DCDC **单点接地底层布局如下：**

2.2.4 4 板载天线

PCB Layout 参考中 PIFA 天线尺寸如图所示。 **天线设计尺寸参考**


15.8mm
0.3mm
1.0mm
1.0mm
1.5mm
5.3mm
5.1mm
2.6mm
C8
L2
C7
L1
L3
ANT1
Y1

Chapter 3

演示例程

3.1 例程介绍

3.1.1 基础例程

源码路径: `zephyr\samples_panchip\basic`。

- Basic: Blinky: 演示 LED 闪灯，`led0` 以一秒为周期亮灭。
- Basic: Synchronization: 演示内核基本功能，系统启动两个线程交替向 `Console` 打印消息。
- Basic: Hello World: 演示打印功能，支持 `mcu boot`。

3.1.2 蓝牙例程

源码路径: `zephyr\samples_panchip\bluetooth`。

- Bluetooth: Audio Client: 演示蓝牙语音传输主机端。
- Bluetooth: Audio Server: 演示蓝牙语音传输从机端。
- Bluetooth: Beacon: 演示蓝牙 beacon。
- 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: Central / Heart-rate Monitor: 演示蓝牙主机功能，主要是 HR (heart-rate) 服务相关，包括：发现设备, 解析广播数据并与包含 HR 服务的设备建立连接；查找并订阅 HR 服务。
- Bluetooth: Central / Health Thermometer sensor: 演示蓝牙主机功能，主要是 HT (health thermometer) 服务相关，包括：发现设备, 解析广播数据并与包含 HT 服务的设备建立连接；查找并订阅 HT 服务。
- Bluetooth: Central / Multilink: 演示主机多连接功能，可以发现设备并与最多 8 个从机设备建立连接。
- Bluetooth: Eddystone: 演示 Google `Eddystone Configuration Service` 和 `Eddystone beacon` 功能。
- Bluetooth: HCI UART: 单 BLE Controller，可以通过串口发送 HCI 命令，用于 DTM，或者配合外部 Host 使用。
- Bluetooth: iBeacon: 演示 Apple iBeacon 功能，在支持 iBeacon 的应用上，可以粗略的显示距离信息。
- Bluetooth: IoT WeChat: 演示微信硬件开发平台的 `AirSync` 协议。
- Bluetooth: Mesh Demo: 演示 MESH 组网，friend-LPN，heartbeat publish 等。

- Bluetooth: Mesh Echo: 演示蓝牙 MESH 功能，可与 Google ECHO 音响进行绑定，并进行开关灯控制。
- Bluetooth: Mesh Provisioner: 演示蓝牙 MESH Provisioner 功能，先进行自我配置 netkey，存储，然后通过 pb adv 对其他待入网设备广播进行扫描，建立 link，入网的流程，并包括后续的配置流程。
- Bluetooth: Mesh Speaker: 演示蓝牙 MESH 功能，可与天猫精灵、百度小度音响进行绑定，并进行开关灯控制。
- Bluetooth: Multi-roles: 演示蓝牙多角色（主从一体）功能，可以通过 `shell` 进行广播、扫描和连接，支持最多 8 组连接。
- Bluetooth: Peripheral: 演示蓝牙从机功能，包含 GATT 服务: CTS/BAS/HRS。
- Bluetooth: Peripheral CSC: 演示蓝牙从机功能，包含 GATT 服务：CSC (Cycling Speed and Cadence)。
- Bluetooth: Peripheral DIS: 演示蓝牙从机功能，包含 GATT 服务: DIS (Device Information)。
- Bluetooth: Peripheral ESP: 演示蓝牙从机功能，包含 GATT 服务: ESP (Environmental Sensing Profile)。
- Bluetooth: Peripheral HIDs: 演示蓝牙从机功能，包含 GATT 服务：HID，通用鼠标。
- Bluetooth: Peripheral / Heart-rate Monitor: 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。
- Bluetooth: Peripheral / Health Thermometer sensor: 演示蓝牙从机功能，包含 GATT 服务：HT (Health Thermometer)，连接订阅服务后，会上报虚拟的温度数据。
- Bluetooth: Peripheral Identity: 演示从机多连接功能，可以与最多 8 个主机设备建立连接。
- Bluetooth: Peripheral OTA: 演示蓝牙从机 OTA (Over-The-Air) 无线升级功能。
- Bluetooth: Scan & Advertise: 演示蓝牙广播和扫描功能，将扫描到的设备个数，放在特定的广播数据中发出去。

3.1.3 外设驱动例程

源码路径: `zephyr\samples_panchip\drivers`。

- Driver: ACC: 演示硬件乘法器除法器功能。
- Driver: ADC: 演示 adc 单次转换与多次转换功能。
- Driver: Counter: 演示硬件计数器循环计数功能。
- Driver: Flash Shell: 演示 flash 读写擦功能。
- Driver: GPIO: 演示 gpio 输入输出及中断功能。
- Driver: I2C Master: 演示 i2c master 读写功能。
- Driver: I2C Slave: 演示 i2c slave 读写功能。
- Driver: Pinmux: 演示开启或关闭 SoC 内部上拉或下拉电阻及切换引脚功能。
- Driver: Power Management: 演示 Zephyr 低功耗流程。
- Driver: PWM & RGB: 演示用 PWM 控制 rgb 灯的功能。
- Driver: QDEC & PWM: 演示 qdec 计数功能。
- Driver: SPI Master: 演示 spi master 读写功能。
- Driver: UART FIFO: 演示 uart 收发功能。

3.1.4 私有 2.4G 例程

源码路径: `zephyr\samples_panchip\proprietary_radio`。

- PRF: IO Pulse Receiver: 演示脉冲传输功能，接收发送端的 2.4G 信号，并恢复出波形，通过 IO 口输出。
- PRF: IO Pulse Transmitter: 演示脉冲传输功能，通过 IO 口接收将外部的 PWM 波形，并通过 2.4G 传输给接收端设备。
- PRF: 2.4G Receiver: 演示 2.4G 接收端功能。
- PRF: 2.4G Transmitter: 演示 2.4G 发送端功能。

3.1.5 解决方案

源码路径: `zephyr\samples_panchip\solutions`。

- Solution: BLE Google Light: 谷歌灯解决方案，通过谷歌音响控制灯。
- Solution: BLE HID Selfie: 自拍解决方案，通过蓝牙 HID 控制手机拍照。
- Solution: BLE Panchip-CTE Beacon: 磐启定位标签解决方案，通过广播发送特定的定位数据。
- Solution: BLE RGB Light: 蓝牙 RGB 三色灯解决方案，可以用小程序连接并进行控制。
- Solution: Mesh Panchip: 蓝牙 MESH 解决方案，支持远程入网，OTA 灯。
- Solution: Multi-mode Mouse: 鼠标解决方案，支持 BLE、2.4G、USB 三中模式。
- Solution: USB Dongle: dongle 解决方案，支持 BLE、2.4G。

注解: SDK 中的所有例程均已适配了 PAN1080 EVB 开发板，我们可在 `zephyr\boards\arm\` 路径下找到当前支持的开发板配置文件。

开发版的配置信息存储在各自目录下的 `*.dts` 中。

- pan1080a_afld_evb
  - console, shell-uart, bt-c2h-uart
    - uart1: TX: P06, RX: P07, 8N1, baudrate: 921600
  - LEDs
    - led_red (led0): P21, active low
  - PWM LEDs
    - pwm_led_red: pwm0_ch4, P10
    - pwm_led_green: pwm0_ch5, P11
    - pwm_led_blue: pwm0_ch6, P16
  - Buttons
    - key1: P04, active low
    - key2: P05, active low

3.2 例程列表

3.2.1 基础例程

Basic: Blinky

1 功能概述 Blinky 是一个简单的应用程序，源代码展示了如何配置 GPIO 引脚作为输出，然后打开和关闭它们。需要确保 board.dts 对应连接 led 作为 GPIO 的输出。

2 环境要求

- board: pan1080a_afld_evb
- led: P21 连接的 led red 作为演示输出灯

3 编译和烧录 项目位置: `zephyr\samples_panchip\basic\blinky`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置: `quick_build_samples\blinky.bat`

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`, 可查看源码, 执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明 连接 P21 与 RGBR（红色 LED 灯），下载代码，观察到 led 灯以 1s 间隔闪烁。

5 开发说明 通过 board devicetree 获取或者定义 led 对应的 gpio 及管脚

通过 `gpio_pin_configure` 配置 GPIO 输出模式

```
gpio_pin_configure(dev, PIN, GPIO_OUTPUT_ACTIVE | FLAGS);
```

通过 `gpio_pin_set` 配置管脚拉高拉低

```
while (1) {
                gpio_pin_set(dev, PIN, (int)led_is_on);
                led_is_on = !led_is_on;
                k_msleep(SLEEP_TIME_MS);
        }
```

Basic: Hello World

1 功能概述 此 sample 为基础 hello_world 的演示 sample，其中可编译基础 hello_world 程序和供 mcuboot 使用参考的 hello_world_with_boot(不能单独使用)。

2 环境要求

- board: pan1080a_afld_evb
- uart (option): 显示串口 log

3 编译和烧录 项目位置: zephyr\samples_panchip\hello_world

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置 (脚本分为四种，分别配置了节点的不同模式):

1. quick_build_samples\hello_world.bat
2. quick_build_samples\hello_world_with_boot.bat

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明

4.1 测试 hello_world 流程 运行脚本 quick_build_samples\hello_world.bat，编译后进行烧录至 pan1080a_afld_evb

复位后通过串口 921600 波特率打印 log:

```
*** Booting Zephyr OS build zephyr-v2.7.0-300-g2b224e899a36  ***
Hello World from Zephyr on pan1080a_afld_evb without boot!
```

4.2 测试 hello_world_with_boot 流程 运行脚本 quick_build_samples\hello_world_with_boot.bat, 获得编译后的工程源文件，供后续带 boot 的操作使用

之后配合\04_dev_guides\Zephyr Mcuboot Guidance.md 内的介绍测试，使用。

Basic: Synchronization

1 功能概述 一个简单的应用程序，演示了内核的基本功能。两个线程 (A 和 B) 轮流向控制台打印问候消息，并使用睡眠请求和信号量来控制消息的生成速率。这说明了内核调度，通信，而且时间安排是正确的。

2 环境要求

- board: pan1080a_afld_evb
- uart: 显示串口 log

3 编译和烧录 项目位置: zephyr\samples_panchip\basic\synchronization

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置: quick_build_samples\synchronization.bat

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
```

(下页继续)

(续上页)

```
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`, 可查看源码, 执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明 下载后，串口工具中看到程序成功运行，两个线程 threadA 与 threadB 交替打印 Log 信息

```
   threadA: Hello World!
   threadB: Hello World!
   threadA: Hello World!
   threadB: Hello World!
   threadA: Hello World!
   threadB: Hello World!
   threadA: Hello World!
   threadB: Hello World!
   threadA: Hello World!
   threadB: Hello World!
```

5 开发说明 建立两个线程

```
        k_thread_create(&threadA_data, threadA_stack_area,
                        K_THREAD_STACK_SIZEOF(threadA_stack_area),
                        threadA, NULL, NULL, NULL,
                        PRIORITY, 0, K_FOREVER);
        k_thread_name_set(&threadA_data, "thread_a");

        k_thread_create(&threadB_data, threadB_stack_area,
                        K_THREAD_STACK_SIZEOF(threadB_stack_area),
                        threadB, NULL, NULL, NULL,
                        PRIORITY, 0, K_FOREVER);
        k_thread_name_set(&threadB_data, "thread_b");
```

轮询调度线程信号量

```
/*
 * @param my_name      thread identification string
 * @param my_sem       thread's own semaphore
 * @param other_sem    other thread's semaphore
 */
void helloLoop(const char *my_name,
               struct k_sem *my_sem, struct k_sem *other_sem)
{
        const char *tname;
        uint8_t cpu;
        struct k_thread *current_thread;

        while (1) {
                /* take my semaphore */
                k_sem_take(my_sem, K_FOREVER);

                current_thread = k_current_get();
                tname = k_thread_name_get(current_thread);
                /* say "hello" */
                if (tname == NULL) {
                        printk("%s: Hello World from cpu %d on %s!\n",
                                my_name, cpu, CONFIG_BOARD);
                } else {
                        printk("%s: Hello World from cpu %d on %s!\n",
```

(下页继续)

(续上页)

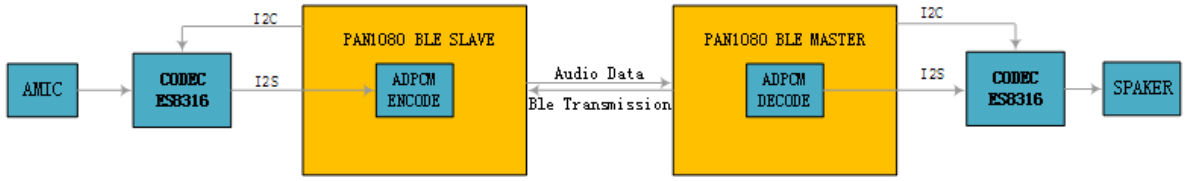
```
                                tname, cpu, CONFIG_BOARD);
                }

                /* wait a while, then let other thread have a turn */
                k_busy_wait(100000);
                k_msleep(SLEEPTIME);
                k_sem_give(other_sem);
        }
}
```

3.2.2 蓝牙例程

Bluetooth: Audio Client

1 功能概述 本文主要介绍 PAN1080 BLE 音频传输-主机例程，负责通过蓝牙接收从机端的音频数据，并在音频设备播放出来。

系统框图如图所示：



2 环境要求

- board: pan1080a_afld_evb
- uart (option): 显示串口 log
- ES8316 模块：带喇叭或耳机

  需要搭配一个运行audio_server 的板子一起使用。

3 编译和烧录 项目位置：`zephyr\samples_panchip\bluetooth\audio_client`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\bluetooth\audio_client.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明

1. 端口连接

   IIC 端口连接：

PAN1080 EVB 板端口	ES8316 模块端口
P0_0（I2C0_SCL）	IIC:CLK
P0_1（I2C0_SDA）	IIC:DAT

IIS 端口连接：

PAN1080 EVB 板端口	ES8316 模块端口
P4_3（I2SMCLK）	MCLK
P4_4（I2SM_WS）	DLRCLK
P4_0（I2SM_CLK）	SCLK
P4_1（I2SM_DI）	Useless
P4_2（I2SM_DO）	DSDIN

2. 配置语音发送端（参考audio_server）。
3. 上电，等待蓝牙连接。
4. 蓝牙连接成功后，喇叭或耳机就能听到对端的声音。

5 开发说明

5.1 音频数据采集 从机端的 CODEC 芯片将 MIC 的数据通过 I2S 传输给 PAN1080，此时 PAN1080 作 I2S 的 SLAVE，CODEC 芯片作 I2S 的 MASTER，PAN1080 接收从 CODEC 芯片采集的数据。

CODEC 芯片初始化的接口：

```
void I2S_HW_Init(void)
{
        uint32_t data_len,cycle_cnt = 0;
    printk("\nCPU @ %dHz,\n", SystemCoreClock);

        I2S_GPIO_Init();
        I2C_InitRole(I2C0,I2C_MODE_MASTER,ADR10BIT_DISABLE);
    I2C_SetTxTirggerLevel(I2C0, I2C_TX_TL_1);

        data_len = sizeof(codec8316_startup);
        while(cycle_cnt < data_len)
    {
                I2C_Write(I2C0,codec8316_startup+cycle_cnt,2);
                cycle_cnt += 2;
                if((codec8316_startup[cycle_cnt-2] == 0x0) &&
           (codec8316_startup[cycle_cnt-1] == 0xc0))
        {
                        k_sleep(50);
                }
                if(cycle_cnt == 2)
        {
                        k_sleep(5);
                }
                if(cycle_cnt == 6)
        {
                        k_sleep(30);
                }
        }
        I2S_DMA_Init();
}
```

PAN1080 DMA 方式接收音频数据初始化接口：

```
void I2S_DMA_Init(void)
{
        DMAC_ChannelConfigTypeDef RxConfigTmp = {0,};

        CLK_APB1PeriphClockCmd(CLK_APB1Periph_I2SS,ENABLE);

        I2S_ModduleInit();
        I2S_SysInit();
        DMAC_Init(DMA);
        IRQ_DIRECT_CONNECT(DMA_IRQn, 0, DMA_IRQHandler, 0);
        irq_enable(DMA_IRQn);
        I2S_EnableI2s(TGT_IIS);
        I2S_EnableRecBlock(TGT_IIS);

    RxConfigTmp.AddrChangeDst    = DMAC_AddrChange_Increment;
    RxConfigTmp.AddrChangeSrc    = DMAC_AddrChange_NoChange;
    RxConfigTmp.BurstLenDst      = DMAC_BurstLen_4;
    RxConfigTmp.BurstLenSrc      = DMAC_BurstLen_4;
    RxConfigTmp.DataWidthDst     = DMAC_DataWidth_16;
    RxConfigTmp.DataWidthSrc     = DMAC_DataWidth_16;
    RxConfigTmp.TransferType     = DMAC_TransferType_Per2Mem;
    RxConfigTmp.FlowControl      = DMAC_FlowControl_DMA;
    RxConfigTmp.HandshakeDst     = DMAC_Handshake_Hardware;
    RxConfigTmp.HandshakeSrc     = DMAC_Handshake_Hardware;
    RxConfigTmp.PeripheralSrc    = DMAC_Peripheral_IIS_Rx;
    RxConfigTmp.IntEnable        = ENABLE;

        /*get free dma channel;*/
        gDMARxChNum = DMAC_AcquireChannel(DMA);
        /*enable dma transfer interrupt*/
        DMAC_ClrIntFlagMsk(DMA,gDMARxChNum,DMAC_FLAG_INDEX_TFR);
        DMAC_SetChannelConfig(DMA,gDMARxChNum,&RxConfigTmp);
        I2S_Recv_Enable();
        printk("dma init\n");
}
```

CODEC 采样频率可设置成 8khz 或 16khz，采样频率设置接口：

```
void I2S_SetI2sClkDivider(uint16_t u32Div)
{
        CLK->APB1_CLK_CTRL1 &=  ~0xFFFF;
        //CLK->APB1_CLK_CTRL1 |= u32Div;
    CLK->APB1_CLK_CTRL1 |= (3 << 16) | (CODEC_FREQ << 18);       //lrclk
}
```

注：DMA 中断优先级设成 0，BLE 中断优先级设成 1。DMA 中断优先级比 BLE 中断优先级高是为了防止 BLE 影响音频数据的采集从而导致音频数据采集不连续。

5.2 **音频数据编解码** 本系统音频编解码采用的是 ADPCM 编解码方式。

ADPCM (ADPCM Adaptive Differential Pulse Code Modulation)，是一种针对 16bit (或者更高) 声音波形数据的一种有损压缩算法，它将声音流中每次采样的 16bit 数据以 4bit 存储，所以压缩比 1:4。而且编解码算法简单，所以是一种低空间消耗，高质量声音获得的好途径。

音频数据编码接口：

```
int adpcm_encoder(short *indata, unsigned char *outdata, int len)
{
    short *inp;         /* Input buffer pointer */
    unsigned char *outp;/* output buffer pointer */
    int val;            /* Current input sample value */
```

(下页继续)

(续上页)

```
    int sign;           /* Current adpcm sign bit */
    unsigned int delta; /* Current adpcm output value */
    int diff;           /* Difference between val and valprev */
    unsigned int udiff; /* unsigned value of diff */
    unsigned int step;  /* Stepsize */
    int valpred;        /* Predicted output value */
    unsigned int vpdiff;/* Current change to valpred */
    int index;          /* Current step change index */
    unsigned int outputbuffer = 0;/* place to keep previous 4-bit value */
    int bufferstep;     /* toggle between outputbuffer/output */
    int count = 0;      /* the number of bytes encoded */

    outp = outdata;
    inp = indata;

    valpred = s_valprev;
    index = s_index;

    step = stepsizeTable[index];

    bufferstep = 1;

    while (len-- > 0 ) {
        val = *inp++;

        /* Step 1 - compute difference with previous value */
        diff = val - valpred;
        if(diff < 0)
        {
            sign = 8;
            diff = (-diff);
        }
        else
        {
            sign = 0;
        }
        /* diff will be positive at this point */
        udiff = (unsigned int)diff;

        /* Step 2 - Divide and clamp */
        /* Note:
        ** This code *approximately* computes:
        **    delta = diff*4/step;
        **    vpdiff = (delta+0.5)*step/4;
        ** but in shift step bits are dropped. The net result of this is
        ** that even if you have fast mul/div hardware you cannot put it to
        ** good use since the fixup would be too expensive.
        */
        delta = 0;
        vpdiff = (step >> 3);

        if ( udiff >= step ) {
            delta = 4;
            udiff -= step;
            vpdiff += step;
        }
        step >>= 1;
        if ( udiff >= step  ) {
            delta |= 2;
            udiff -= step;
            vpdiff += step;
```

(下页继续)

(续上页)

```
        }
        step >>= 1;
        if ( udiff >= step ) {
            delta |= 1;
            vpdiff += step;
        }

        /* Phil Frisbie combined steps 3 and 4 */
        /* Step 3 - Update previous value */
        /* Step 4 - Clamp previous value to 16 bits */
        if ( sign != 0 )
        {
            valpred -= vpdiff;
            if ( valpred < -32768 )
                valpred = -32768;
        }
        else
        {
            valpred += vpdiff;
            if ( valpred > 32767 )
                valpred = 32767;
        }

        /* Step 5 - Assemble value, update index and step values */
        delta |= sign;

        index += indexTable[delta];
        if ( index < 0 ) index = 0;
        if ( index > 88 ) index = 88;
        step = stepsizeTable[index];

        /* Step 6 - Output value */
        if ( bufferstep != 0 ) {
            outputbuffer = (delta << 4);
        } else {
            *outp++ = (char)(delta | outputbuffer);
            count++;
        }
        bufferstep = !bufferstep;
    }

    /* Output last step, if needed */
    if ( bufferstep == 0 )
    {
        *outp++ = (char)outputbuffer;
        count++;
    }

    *outp++=s_valprev & 0xff;         //save value prev
        *outp++=(s_valprev >> 8) & 0xff;
        *outp=s_index;                          //save index
        count = count + 3;

    s_valprev = (short)valpred;
    s_index = (char)index;

    return count;
}
```

音频数据解码接口：

```
int adpcm_decoder(unsigned char *indata, short *outdata, int len)
{
    unsigned char *inp; /* Input buffer pointer */
    short *outp;        /* output buffer pointer */
    unsigned int sign;  /* Current adpcm sign bit */
    unsigned int delta; /* Current adpcm output value */
    unsigned int step;  /* Stepsize */
    int valpred;        /* Predicted value */
    unsigned int vpdiff;/* Current change to valpred */
    int index;          /* Current step change index */
    unsigned int inputbuffer = 0;/* place to keep next 4-bit value */
    int bufferstep;     /* toggle between inputbuffer/input */
    int count = 0;

    short value_tmp;
    char index_tmp;

    outp = outdata;
    inp = indata;

    value_tmp = inp[len-3] | (inp[len-2] << 8);        //recover value
    index_tmp = inp[len - 1];          //recover index
    len = len - 3;

    valpred = value_tmp;
    index = index_tmp;
    step = stepsizeTable[index];

    bufferstep = 0;

    len *= 2;

    while ( len-- > 0 ) {

        /* Step 1 - get the delta value */
        if ( bufferstep != 0 ) {
            delta = inputbuffer & 0xf;
        } else {
            inputbuffer = (unsigned int)*inp++;
            delta = (inputbuffer >> 4);
        }
        bufferstep = !bufferstep;

        /* Step 2 - Find new index value (for later) */
        index += indexTable[delta];
        if ( index < 0 ) index = 0;
        if ( index > 88 ) index = 88;

        /* Step 3 - Separate sign and magnitude */
        sign = delta & 8;
        delta = delta & 7;

        /* Phil Frisbie combined steps 4 and 5 */
        /* Step 4 - Compute difference and new predicted value */
        /* Step 5 - clamp output value */
        /*
        ** Computes 'vpdiff = (delta+0.5)*step/4', but see comment
        ** in adpcm_coder.
        */
        vpdiff = step >> 3;
        if ( (delta & 4) != 0 ) vpdiff += step;
        if ( (delta & 2) != 0 ) vpdiff += step>>1;
```

(下页继续)

(续上页)

```
        if ( (delta & 1) != 0 ) vpdiff += step>>2;

        if ( sign != 0 )
        {
            valpred -= vpdiff;
            if ( valpred < -32768 )
                valpred = -32768;
        }
        else
        {
            valpred += vpdiff;
            if ( valpred > 32767 )
                valpred = 32767;
        }

        /* Step 6 - Update step value */
        step = stepsizeTable[index];

        /* Step 7 - Output value */
        *outp++ = (short)valpred;
        count++;
    }

    return count;
}
```

5.3 **音频数据传输** PAN1080 从机将采集的音频数据编码后通过 BLE 传输给 PAN1080 主机。

BLE 从机上传数据的接口：

```
int proj_template_s2c_notify(const u8_t *data,u16_t len)
{
        return bt_gatt_notify(NULL, &proj_temp.attrs[1], data, len);
}
```

BLE 主机接收数据的接口：

```
static u8_t notify_func(struct bt_conn *conn,
                        struct bt_gatt_subscribe_params *params,
                        const void *data, u16_t length)
{
        if (!data) {
                printk("[UNSUBSCRIBED]\n");
                params->value_handle = 0U;
                return BT_GATT_ITER_STOP;
        }

        P20 = 1;
        uint8_t *p_buf = (uint8_t *)decode_data;

        decode_data_len = adpcm_decoder(data,decode_data,length);
        ring_buf_put(&ringbuf_raw, p_buf, (decode_data_len * 2));
        if((I2S_data_flag == 0) && (buf_count >= 2))
        {
                I2S_data_flag = 1;
                I2S_EnableClk(TGT_IIS);
        }
        else
        {
```

(下页继续)

(续上页)

```
                buf_count++;
        }
        P20 = 0;

        return BT_GATT_ITER_CONTINUE;
}
```

5.4 音频数据播放　主机端 PAN1080 将解码后的音频数据通过 I2S 传输给 CODEC 芯片，此时 PAN1080 作 I2S 的 MASTER，CODEC 芯片作 I2S 的 SLAVE，PAN1080 将音频数据发送给 CODEC 芯片播放。

CODEC 芯片初始化接口：

```
void I2S_HW_Init(void)
{
        uint32_t data_len,cycle_cnt = 0;
    printk("\nCPU @ %dHz,\n", SystemCoreClock);

        I2S_GPIO_Init();
        I2C_InitRole(I2C0,I2C_MODE_MASTER,ADR10BIT_DISABLE);
    I2C_SetTxTirggerLevel(I2C0, I2C_TX_TL_1);

        data_len = sizeof(codec8316_startup);
        while(cycle_cnt < data_len)
    {
                I2C_Write(I2C0,codec8316_startup+cycle_cnt,2);
                cycle_cnt += 2;
                if((codec8316_startup[cycle_cnt-2] == 0x0)
           && (codec8316_startup[cycle_cnt-1] == 0xc0))
        {
                        k_sleep(50);
                }
                if(cycle_cnt == 2)
        {
                        k_sleep(5);
                }
                if(cycle_cnt == 6)
        {
                        k_sleep(30);
                }
        }
        I2S_DMA_Init();
}
```

主机端 CODEC 芯片初始化流程和从机端 CODEC 芯片初始化流程一样，不同的地方是 CODEC 芯片的配置，从机端配置成音频数据采集，主机端配置成音频数据播放。

PAN1080 DMA 方式发送音频数据初始化接口：

```
void I2S_DMA_Init(void)
{
        DMAC_ChannelConfigTypeDef TxConfigTmp = {0,};

        CLK_APB1PeriphClockCmd(CLK_APB1Periph_I2SM,ENABLE);
        CLK_APB1PeriphClockCmd(CLK_APB1Periph_I2SS,DISABLE);

        I2S_ModduleInit();
        I2S_SysInit();
        DMAC_Init(DMA);
        IRQ_DIRECT_CONNECT(DMA_IRQn, 0, DMA_IRQHandler, 0);
        irq_enable(DMA_IRQn);
```

(下页继续)

(续上页)

```
        I2S_EnableI2s(TGT_IIS);

        I2S_SetTxTrigLevel(TGT_IIS,0,4);
        I2S_EnableTransmitCh(TGT_IIS,0);

        I2S_EnableTransmitBlock(TGT_IIS);

    /*set tx dma channel control&config register*/
    TxConfigTmp.AddrChangeDst    = DMAC_AddrChange_NoChange;
    TxConfigTmp.AddrChangeSrc    = DMAC_AddrChange_Increment;
    TxConfigTmp.BurstLenDst      = DMAC_BurstLen_8;
    TxConfigTmp.BurstLenSrc      = DMAC_BurstLen_8;
    TxConfigTmp.DataWidthDst     = DMAC_DataWidth_16;
    TxConfigTmp.DataWidthSrc     = DMAC_DataWidth_16;
    TxConfigTmp.TransferType     = DMAC_TransferType_Mem2Per;
    TxConfigTmp.FlowControl      = DMAC_FlowControl_DMA;
    TxConfigTmp.HandshakeDst     = DMAC_Handshake_Hardware;
    TxConfigTmp.HandshakeSrc     = DMAC_Handshake_Hardware;
    TxConfigTmp.PeripheralDst    = DMAC_Peripheral_IIS_Tx;
    TxConfigTmp.IntEnable        = ENABLE;

        /*get free dma channel;*/
        gDMATxChNum = DMAC_AcquireChannel(DMA);
        /*enable dma transfer interrupt*/
        DMAC_ClrIntFlagMsk(DMA,gDMATxChNum,DMAC_FLAG_INDEX_TFR);
        DMAC_SetChannelConfig(DMA,gDMATxChNum,&TxConfigTmp);

        //I2S_EnableClk(TGT_IIS);
        I2S_Send_Enable();
}
```

CODEC 播放频率可设置成 8khz 或 16khz，播放频率设置接口：

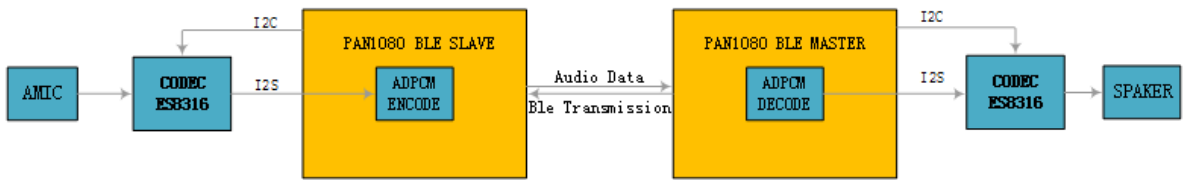
```
void I2S_SetI2sClkDivider(uint16_t codec_freq)
{
        CLK->APB1_CLK_CTRL1 &=  ~0xFFFF;
        CLK->APB1_CLK_CTRL1 |= codec_freq;
    CLK->APB1_CLK_CTRL1 |= (3 << 16) | (3 << 18);
}
```

注：DMA 中断优先级设成 0，BLE 中断优先级设成 1。DMA 中断优先级比 BLE 中断优先级高是为了防止 BLE 影响音频数据的播放从而导致音频数据播放不连续从而产生噪音。

Bluetooth: Audio Server

1 功能概述 本文主要介绍 PAN1080 BLE 音频传输-从机例程，负责通过 MIC 采集音频数据，并通过蓝牙发送给主机端设备。

系统框图如图所示：



2 环境要求

- board: pan1080a_afld_evb

- uart (option): 显示串口 log
- ES8316 模块：带喇叭或耳机

  需要搭配一个运行audio_client 的板子一起使用。

3 编译和烧录 项目位置：`zephyr\samples_panchip\bluetooth\audio_server`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\bluetooth\audio_server.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`，可查看源码，执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明

1. 端口连接

   IIC 端口连接：

PAN1080 EVB 板端口	ES8316 模块端口
P0_0（I2C0_SCL）	IIC:CLK
P0_1（I2C0_SDA）	IIC:DAT

   IIS 端口连接：

PAN1080 EVB 板端口	ES8316 模块端口
P4_3（I2SMCLK）	MCLK
P4_4（I2SS_WS）	ALRCLK
P4_0（I2SS_CLK）	SCLK
P4_1（I2SS_DI）	ASDOUT
P4_2（I2SS_DO）	Useless

2. 上电。
3. 配置语音接收端（参考audio_client），并等待蓝牙连接。
4. 蓝牙连接成功后，可以通过 MIC 将语音发送给对端。

5 开发说明

5.1 音频数据采集 从机端的 CODEC 芯片将 MIC 的数据通过 I2S 传输给 PAN1080，此时 PAN1080 作 I2S 的 SLAVE，CODEC 芯片作 I2S 的 MASTER，PAN1080 接收从 CODEC 芯片采集的数据。

CODEC 芯片初始化的接口：

```
void I2S_HW_Init(void)
{
        uint32_t data_len,cycle_cnt = 0;
    printk("\nCPU @ %dHz,\n", SystemCoreClock);
```

(下页继续)

(续上页)

```
        I2S_GPIO_Init();
        I2C_InitRole(I2C0,I2C_MODE_MASTER,ADR10BIT_DISABLE);
    I2C_SetTxTirggerLevel(I2C0, I2C_TX_TL_1);

        data_len = sizeof(codec8316_startup);
        while(cycle_cnt < data_len)
    {
                I2C_Write(I2C0,codec8316_startup+cycle_cnt,2);
                cycle_cnt += 2;
                if((codec8316_startup[cycle_cnt-2] == 0x0) &&
           (codec8316_startup[cycle_cnt-1] == 0xc0))
        {
                        k_sleep(50);
                }
                if(cycle_cnt == 2)
        {
                        k_sleep(5);
                }
                if(cycle_cnt == 6)
        {
                        k_sleep(30);
                }
        }
        I2S_DMA_Init();
}
```

PAN1080 DMA 方式接收音频数据初始化接口：

```
void I2S_DMA_Init(void)
{
        DMAC_ChannelConfigTypeDef RxConfigTmp = {0,};

        CLK_APB1PeriphClockCmd(CLK_APB1Periph_I2SS,ENABLE);

        I2S_ModduleInit();
        I2S_SysInit();
        DMAC_Init(DMA);
        IRQ_DIRECT_CONNECT(DMA_IRQn, 0, DMA_IRQHandler, 0);
        irq_enable(DMA_IRQn);
        I2S_EnableI2s(TGT_IIS);
        I2S_EnableRecBlock(TGT_IIS);

    RxConfigTmp.AddrChangeDst   = DMAC_AddrChange_Increment;
    RxConfigTmp.AddrChangeSrc   = DMAC_AddrChange_NoChange;
    RxConfigTmp.BurstLenDst     = DMAC_BurstLen_4;
    RxConfigTmp.BurstLenSrc     = DMAC_BurstLen_4;
    RxConfigTmp.DataWidthDst    = DMAC_DataWidth_16;
    RxConfigTmp.DataWidthSrc    = DMAC_DataWidth_16;
    RxConfigTmp.TransferType    = DMAC_TransferType_Per2Mem;
    RxConfigTmp.FlowControl     = DMAC_FlowControl_DMA;
    RxConfigTmp.HandshakeDst    = DMAC_Handshake_Hardware;
    RxConfigTmp.HandshakeSrc    = DMAC_Handshake_Hardware;
    RxConfigTmp.PeripheralSrc   = DMAC_Peripheral_IIS_Rx;
    RxConfigTmp.IntEnable       = ENABLE;

        /*get free dma channel;*/
        gDMARxChNum = DMAC_AcquireChannel(DMA);
        /*enable dma transfer interrupt*/
        DMAC_ClrIntFlagMsk(DMA,gDMARxChNum,DMAC_FLAG_INDEX_TFR);
        DMAC_SetChannelConfig(DMA,gDMARxChNum,&RxConfigTmp);
```

(下页继续)

(续上页)

```
        I2S_Recv_Enable();
        printk("dma init\n");
}
```

CODEC 采样频率可设置成 8khz 或 16khz，采样频率设置接口：

```
void I2S_SetI2sClkDivider(uint16_t u32Div)
{
        CLK->APB1_CLK_CTRL1 &=  ~0xFFFF;
        //CLK->APB1_CLK_CTRL1 |= u32Div;
    CLK->APB1_CLK_CTRL1 |= (3 << 16) | (CODEC_FREQ << 18);       //lrclk
}
```

注：DMA 中断优先级设成 0，BLE 中断优先级设成 1。DMA 中断优先级比 BLE 中断优先级高是为了防止 BLE 影响音频数据的采集从而导致音频数据采集不连续。

5.2 音频数据编解码　本系统音频编解码采用的是 ADPCM 编解码方式。

ADPCM (ADPCM Adaptive Differential Pulse Code Modulation)，是一种针对 16bit (或者更高) 声音波形数据的一种有损压缩算法，它将声音流中每次采样的 16bit 数据以 4bit 存储，所以压缩比 1:4。而且编解码算法简单，所以是一种低空间消耗，高质量声音获得的好途径。

音频数据编码接口：

```
int adpcm_encoder(short *indata, unsigned char *outdata, int len)
{
    short *inp;         /* Input buffer pointer */
    unsigned char *outp;/* output buffer pointer */
    int val;            /* Current input sample value */
    int sign;           /* Current adpcm sign bit */
    unsigned int delta; /* Current adpcm output value */
    int diff;           /* Difference between val and valprev */
    unsigned int udiff; /* unsigned value of diff */
    unsigned int step;  /* Stepsize */
    int valpred;        /* Predicted output value */
    unsigned int vpdiff;/* Current change to valpred */
    int index;          /* Current step change index */
    unsigned int outputbuffer = 0;/* place to keep previous 4-bit value */
    int bufferstep;     /* toggle between outputbuffer/output */
    int count = 0;      /* the number of bytes encoded */

    outp = outdata;
    inp = indata;

    valpred = s_valprev;
    index = s_index;

    step = stepsizeTable[index];

    bufferstep = 1;

    while (len-- > 0 ) {
        val = *inp++;

        /* Step 1 - compute difference with previous value */
        diff = val - valpred;
        if(diff < 0)
        {
            sign = 8;
            diff = (-diff);
        }
```

(下页继续)

(续上页)

```
        else
        {
            sign = 0;
        }
        /* diff will be positive at this point */
        udiff = (unsigned int)diff;

        /* Step 2 - Divide and clamp */
        /* Note:
        ** This code *approximately* computes:
        **    delta = diff*4/step;
        **    vpdiff = (delta+0.5)*step/4;
        ** but in shift step bits are dropped. The net result of this is
        ** that even if you have fast mul/div hardware you cannot put it to
        ** good use since the fixup would be too expensive.
        */
        delta = 0;
        vpdiff = (step >> 3);

        if ( udiff >= step ) {
            delta = 4;
            udiff -= step;
            vpdiff += step;
        }
        step >>= 1;
        if ( udiff >= step  ) {
            delta |= 2;
            udiff -= step;
            vpdiff += step;
        }
        step >>= 1;
        if ( udiff >= step ) {
            delta |= 1;
            vpdiff += step;
        }

        /* Phil Frisbie combined steps 3 and 4 */
        /* Step 3 - Update previous value */
        /* Step 4 - Clamp previous value to 16 bits */
        if ( sign != 0 )
        {
            valpred -= vpdiff;
            if ( valpred < -32768 )
                valpred = -32768;
        }
        else
        {
            valpred += vpdiff;
            if ( valpred > 32767 )
                valpred = 32767;
        }

        /* Step 5 - Assemble value, update index and step values */
        delta |= sign;

        index += indexTable[delta];
        if ( index < 0 ) index = 0;
        if ( index > 88 ) index = 88;
        step = stepsizeTable[index];

        /* Step 6 - Output value */
```

(下页继续)

(续上页)

```
        if ( bufferstep != 0 ) {
            outputbuffer = (delta << 4);
        } else {
            *outp++ = (char)(delta | outputbuffer);
            count++;
        }
        bufferstep = !bufferstep;
    }

    /* Output last step, if needed */
    if ( bufferstep == 0 )
    {
        *outp++ = (char)outputbuffer;
        count++;
    }

    *outp++=s_valprev & 0xff;        //save value prev
        *outp++=(s_valprev >> 8) & 0xff;
        *outp=s_index;                          //save index
        count = count + 3;

    s_valprev = (short)valpred;
    s_index = (char)index;

    return count;
}
```

音频数据解码接口：

```
int adpcm_decoder(unsigned char *indata, short *outdata, int len)
{
    unsigned char *inp; /* Input buffer pointer */
    short *outp;        /* output buffer pointer */
    unsigned int sign;  /* Current adpcm sign bit */
    unsigned int delta; /* Current adpcm output value */
    unsigned int step;  /* Stepsize */
    int valpred;        /* Predicted value */
    unsigned int vpdiff;/* Current change to valpred */
    int index;          /* Current step change index */
    unsigned int inputbuffer = 0;/* place to keep next 4-bit value */
    int bufferstep;     /* toggle between inputbuffer/input */
    int count = 0;

    short value_tmp;
    char index_tmp;

    outp = outdata;
    inp = indata;

    value_tmp = inp[len-3] | (inp[len-2] << 8);        //recover value
    index_tmp = inp[len - 1];          //recover index
    len = len - 3;

    valpred = value_tmp;
    index = index_tmp;
    step = stepsizeTable[index];

    bufferstep = 0;

    len *= 2;
```

(下页继续)

(续上页)

```
    while ( len-- > 0 ) {

        /* Step 1 - get the delta value */
        if ( bufferstep != 0 ) {
            delta = inputbuffer & 0xf;
        } else {
            inputbuffer = (unsigned int)*inp++;
            delta = (inputbuffer >> 4);
        }
        bufferstep = !bufferstep;

        /* Step 2 - Find new index value (for later) */
        index += indexTable[delta];
        if ( index < 0 ) index = 0;
        if ( index > 88 ) index = 88;

        /* Step 3 - Separate sign and magnitude */
        sign = delta & 8;
        delta = delta & 7;

        /* Phil Frisbie combined steps 4 and 5 */
        /* Step 4 - Compute difference and new predicted value */
        /* Step 5 - clamp output value */
        /*
        ** Computes 'vpdiff = (delta+0.5)*step/4', but see comment
        ** in adpcm_coder.
        */
        vpdiff = step >> 3;
        if ( (delta & 4) != 0 ) vpdiff += step;
        if ( (delta & 2) != 0 ) vpdiff += step>>1;
        if ( (delta & 1) != 0 ) vpdiff += step>>2;

        if ( sign != 0 )
        {
            valpred -= vpdiff;
            if ( valpred < -32768 )
                valpred = -32768;
        }
        else
        {
            valpred += vpdiff;
            if ( valpred > 32767 )
                valpred = 32767;
        }

        /* Step 6 - Update step value */
        step = stepsizeTable[index];

        /* Step 7 - Output value */
        *outp++ = (short)valpred;
        count++;
    }

    return count;
}
```

5.3 **音频数据传输** PAN1080 从机将采集的音频数据编码后通过 BLE 传输给 PAN1080 主机。BLE 从机上传数据的接口：

```
int proj_template_s2c_notify(const u8_t *data,u16_t len)
{
        return bt_gatt_notify(NULL, &proj_temp.attrs[1], data, len);
}
```

BLE 主机接收数据的接口：

```
static u8_t notify_func(struct bt_conn *conn,
                        struct bt_gatt_subscribe_params *params,
                        const void *data, u16_t length)
{
        if (!data) {
                printk("[UNSUBSCRIBED]\n");
                params->value_handle = 0U;
                return BT_GATT_ITER_STOP;
        }

        P20 = 1;
        uint8_t *p_buf = (uint8_t *)decode_data;

        decode_data_len = adpcm_decoder(data,decode_data,length);
        ring_buf_put(&ringbuf_raw, p_buf, (decode_data_len * 2));
        if((I2S_data_flag == 0) && (buf_count >= 2))
        {
                I2S_data_flag = 1;
                I2S_EnableClk(TGT_IIS);
        }
        else
        {
                buf_count++;
        }
        P20 = 0;

        return BT_GATT_ITER_CONTINUE;
}
```

5.4 音频数据播放　主机端 PAN1080 将解码后的音频数据通过 I2S 传输给 CODEC 芯片，此时 PAN1080 作 I2S 的 MASTER，CODEC 芯片作 I2S 的 SLAVE，PAN1080 将音频数据发送给 CODEC 芯片播放。

CODEC 芯片初始化接口：

```
void I2S_HW_Init(void)
{
        uint32_t data_len,cycle_cnt = 0;
    printk("\nCPU @ %dHz,\n", SystemCoreClock);

        I2S_GPIO_Init();
        I2C_InitRole(I2C0,I2C_MODE_MASTER,ADR10BIT_DISABLE);
    I2C_SetTxTirggerLevel(I2C0, I2C_TX_TL_1);

        data_len = sizeof(codec8316_startup);
        while(cycle_cnt < data_len)
    {
                I2C_Write(I2C0,codec8316_startup+cycle_cnt,2);
                cycle_cnt += 2;
                if((codec8316_startup[cycle_cnt-2] == 0x0)
           && (codec8316_startup[cycle_cnt-1] == 0xc0))
        {
                        k_sleep(50);
                }
                if(cycle_cnt == 2)
```

(下页继续)

(续上页)

```
        {
                        k_sleep(5);
                }
                if(cycle_cnt == 6)
        {
                        k_sleep(30);
                }
        }
        I2S_DMA_Init();
}
```

主机端 CODEC 芯片初始化流程和从机端 CODEC 芯片初始化流程一样，不同的地方是 CODEC 芯片的配置，从机端配置成音频数据采集，主机端配置成音频数据播放。

PAN1080 DMA 方式发送音频数据初始化接口：

```
void I2S_DMA_Init(void)
{
        DMAC_ChannelConfigTypeDef TxConfigTmp = {0,};

        CLK_APB1PeriphClockCmd(CLK_APB1Periph_I2SM,ENABLE);
        CLK_APB1PeriphClockCmd(CLK_APB1Periph_I2SS,DISABLE);

        I2S_ModduleInit();
        I2S_SysInit();
        DMAC_Init(DMA);
        IRQ_DIRECT_CONNECT(DMA_IRQn, 0, DMA_IRQHandler, 0);
        irq_enable(DMA_IRQn);

        I2S_EnableI2s(TGT_IIS);

        I2S_SetTxTrigLevel(TGT_IIS,0,4);
        I2S_EnableTransmitCh(TGT_IIS,0);

        I2S_EnableTransmitBlock(TGT_IIS);

    /*set tx dma channel control&config register*/
    TxConfigTmp.AddrChangeDst    = DMAC_AddrChange_NoChange;
    TxConfigTmp.AddrChangeSrc    = DMAC_AddrChange_Increment;
    TxConfigTmp.BurstLenDst      = DMAC_BurstLen_8;
    TxConfigTmp.BurstLenSrc      = DMAC_BurstLen_8;
    TxConfigTmp.DataWidthDst     = DMAC_DataWidth_16;
    TxConfigTmp.DataWidthSrc     = DMAC_DataWidth_16;
    TxConfigTmp.TransferType     = DMAC_TransferType_Mem2Per;
    TxConfigTmp.FlowControl      = DMAC_FlowControl_DMA;
    TxConfigTmp.HandshakeDst     = DMAC_Handshake_Hardware;
    TxConfigTmp.HandshakeSrc     = DMAC_Handshake_Hardware;
    TxConfigTmp.PeripheralDst    = DMAC_Peripheral_IIS_Tx;
    TxConfigTmp.IntEnable        = ENABLE;

        /*get free dma channel;*/
        gDMATxChNum = DMAC_AcquireChannel(DMA);
        /*enable dma transfer interrupt*/
        DMAC_ClrIntFlagMsk(DMA,gDMATxChNum,DMAC_FLAG_INDEX_TFR);
        DMAC_SetChannelConfig(DMA,gDMATxChNum,&TxConfigTmp);

        //I2S_EnableClk(TGT_IIS);
        I2S_Send_Enable();
}
```

CODEC 播放频率可设置成 8khz 或 16khz，播放频率设置接口：

```
void I2S_SetI2sClkDivider(uint16_t codec_freq)
{
        CLK->APB1_CLK_CTRL1 &=  ~0xFFFF;
        CLK->APB1_CLK_CTRL1 |= codec_freq;
    CLK->APB1_CLK_CTRL1 |= (3 << 16) | (3 << 18);
}
```

注：DMA 中断优先级设成 0，BLE 中断优先级设成 1。DMA 中断优先级比 BLE 中断优先级高是为了防止 BLE 影响音频数据的播放从而导致音频数据播放不连续从而产生噪音。

Bluetooth: Beacon

1 功能概述 此项目演示了蓝牙广播功能。

2 环境要求

- board: pan1080a_afld_evb
- uart (option): 显示串口 log
- 测试软件 (option): nRF Connect

3 编译和烧录 项目位置：`zephyr\samples_panchip\bluetooth\beacon`。

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\bluetooth\beacon.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

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

- Advertising Type: ADV_NONCONN_IND
- Advertising Interval Time: 100ms - 150ms
- Advertising data: Eddystone URL (the Zephyr website)
- Device Name: Test Beacon

Bluetooth: 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，并通过足够强的信号建立到第一个设备的连接，演示了非常基本的 BLE central 角色功能。

2 环境要求

- board: 支持 BLE 的蓝牙设备
- uart(option): 用来显示串口 log
- 测试软件: nRF Connect

3 编译和烧录 项目位置：zephyr\samples_panchip\bluetooth\central

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：quick_build_samples\bluetooth\central.bat。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`，可查看源码，执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明

1. 烧录完成后，设备可以显示 Scanning successfully started，并且显示查找到的设备，并显示设备的 RSSI 大小

   ```
   Scanning successfully started
   Device found: 78:5A:A2:78:1C:2B (random) (RSSI -84)
   Device found: 57:92:22:A1:0D:5F (random) (RSSI -93)
   Device found: 60:9D:E2:44:03:C7 (random) (RSSI -80)
   Device found: F4:7A:F6:6A:22:D7 (random) (RSSI -82)
   Device found: D0:90:D0:CF:42:C7 (random) (RSSI -83)
   Device found: 68:73:48:04:2F:6E (random) (RSSI -96)
   ```

2. 当时 central 扫描到一个 RSSI 小于-70 且 adv type 类型为 ADV_IND 或者 ADV_DIRECT_IND 的设备时，此时 central 设备马上建立连接

   ```
   Connected: CD:ED:61:AE:DB:1D (random)
   ```

3. 连接成功后，central 设备会立即断开连接，同时继续扫描满足 2 中所述条件的广播，然后再次连接，如此反复。

   ```
   Disconnected: CD:ED:61:AE:DB:1D (random) (reason 0x13)
   ```

Bluetooth: Central / Heart-rate Monitor

1 功能概述 此项目演示蓝牙主机功能，不同的是这个应用程序专门查找心率监视器，并在连接后报告心率读数。

2 环境要求

- board: 支持 BLE 的蓝牙设备
- uart(option): 用来显示串口 log
- 测试软件: nRF Connect

3 编译和烧录 项目位置：zephyr\samples_panchip\bluetooth\central_hr

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：quick_build_samples\bluetooth\central_hr.bat。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`，可查看源码，执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明

1. 准备 2 个设备，分别烧录 peripheral_hr（见 peripheral_hr 例程），烧录 centrl_hr 例程
2. 开启 central 设备，此时 central 设备可以看到扫描数据

```
Scanning successfully started
[DEVICE]: 1D:0C:35:49:40:DC (random), AD evt type 3, AD data len 31, RSSI -43
[DEVICE]: 1D:0C:35:49:40:DC (random), AD evt type 3, AD data len 31, RSSI -42
[DEVICE]: 59:70:28:64:72:07 (random), AD evt type 3, AD data len 31, RSSI -71
[DEVICE]: 59:70:28:64:72:07 (random), AD evt type 3, AD data len 31, RSSI -72
[DEVICE]: DF:BC:88:D2:0C:B9 (random), AD evt type 0, AD data len 11, RSSI -42
[AD]: 1 data_len 1
[AD]: 3 data_len 6
```

3. 打开 peripheral_hr，显示被 central_hr 设备扫描，串口数据打印 connected

```
Connected

<inf> hrs: HRS notifications enabled
```

4. 在 central_hr 串口上会显示 connect，收到上报的心率信息

```
Connected: DF:BC:88:D2:0C:B9 (random)
[ATTRIBUTE] handle 25
[ATTRIBUTE] handle 28
[SUBSCRIBED]
[NOTIFICATION] data 0x20009f6c length 2
[NOTIFICATION] data 0x20009f6c length 2
[NOTIFICATION] data 0x20009f6c length 2
[NOTIFICATION] data 0x20009f6c length 2
```

Bluetooth: Central / Health Thermometer sensor

1 功能概述 此项目演示蓝牙主机功能，不同的是这个应用程序专门寻找健康温度计传感器，并在连接建立后上报模具温度读数。

2 环境要求

- board: 支持 BLE 的蓝牙设备
- uart(option): 用来显示串口 log
- 测试软件: nRF Connect

3 编译和烧录 项目位置：`zephyr\samples_panchip\bluetooth\central_ht`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\bluetooth\central_ht.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`，可查看源码，执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明

1. 准备 2 块板卡，1 块烧录 peripheral_ht（见 peripheral_ht 例程），一块烧录 central_ht
2. 开启 central 设备，此时 central 设备可以看到扫描数据

```
Scanning successfully started
[DEVICE]: 35:C2:56:72:32:6C (random), AD evt type 3, AD data len 31, RSSI -43
[DEVICE]: EB:46:FA:EA:98:2F (random), AD evt type 0, AD data len 11, RSSI -43
[AD]: 1 data_len 1
[AD]: 3 data_len 6
[DEVICE]: EB:46:FA:EA:98:2F (random), AD evt type 4, AD data len 25, RSSI -43
[DEVICE]: 35:C2:56:72:32:6C (random), AD evt type 3, AD data len 31, RSSI -43
[DEVICE]: 5C:7B:03:66:6F:39 (random), AD evt type 0, AD data len 17, RSSI -42
[AD]: 1 data_len 1
[AD]: 10 data_len 1
[AD]: 255 data_len 9
[DEVICE]: 5C:7B:03:66:6F:39 (random), AD evt type 4, AD data len 0, RSSI -42
[DEVICE]: EB:46:FA:EA:98:2F (random), AD evt type 0, AD data len 11, RSSI -43
```

3. 打开 peripheral_ht，显示被 central_ht 设备扫描，串口数据打印 connected，并显示上报的温度信息

   Connected

   temperature is 21C

   Indication success Indication complete

   temperature is 22C

   Indication success Indication complete

4. 在 central_ht 串口上会显示 connect，查看上报的温度信息

```
Connected: C0:69:E0:DD:2B:1B (random)
[ATTRIBUTE] handle 25
[ATTRIBUTE] handle 26
[ATTRIBUTE] handle 28
[SUBSCRIBED]
Temperature 21C.
Temperature 22C.
```

Bluetooth: Central / Multilink

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，并使用足够强的信号与多达 8 个外设建立连接，演示 BLE Central 功能。

2 环境要求

- board: 支持 BLE 的蓝牙设备
- uart(option): 用来显示串口 log

- 测试软件: nRF Connect

3 编译和烧录 项目位置: `zephyr\samples_panchip\bluetooth\central_multilink`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置: `quick_build_samples\bluetooth\central_multilink.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`, 可查看源码, 执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明

1. 烧录完成后，设备可以正常扫描，串口打印 log

```
Scanning successfully started
Device found: DF:BC:88:D2:0C:B9 (random) (RSSI -43)
Device found: 6F:5D:36:7B:0E:78 (random) (RSSI -81)
Device found: DF:BC:88:D2:0C:B9 (random) (RSSI -42)
Device found: A0:E6:F8:AB:0F:6C (public) (RSSI -50)
Device found: 54:A5:72:32:E5:77 (random) (RSSI -43)
Device found: A0:E6:F8:AB:0F:6C (public) (RSSI -46)
```

2. 扫描到设备的 RSSI< -35 时，就会 connect;
3. 准备 8 个 csc 设备，主设备可以正常连接 8 个设备

```
配置连接个数:
CONFIG_BT_MAX_CONN=8
CONFIG_BT_CTLR_MAX_NUM_OF_STATES=8
```

Bluetooth: Eddystone

1 功能概述 此项目演示 “Eddystone 配置服务”

Eddystone 配置服务作为 GATT 服务在广播上运行，但它是可连接的，并允许配置发布的数据、广播功率级别和发包间隔。

它还构成了了 Eddystone-EID 广播如何配置和向可信解析器注册的定义的一部分。

2 环境要求

- board: 支持 BLE 的蓝牙设备
- uart(option): 用来显示串口 log
- 测试软件: nRF Connect

3 编译和烧录 项目位置: `zephyr\samples_panchip\bluetooth\eddystone`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置: `quick_build_samples\bluetooth\eddystone.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

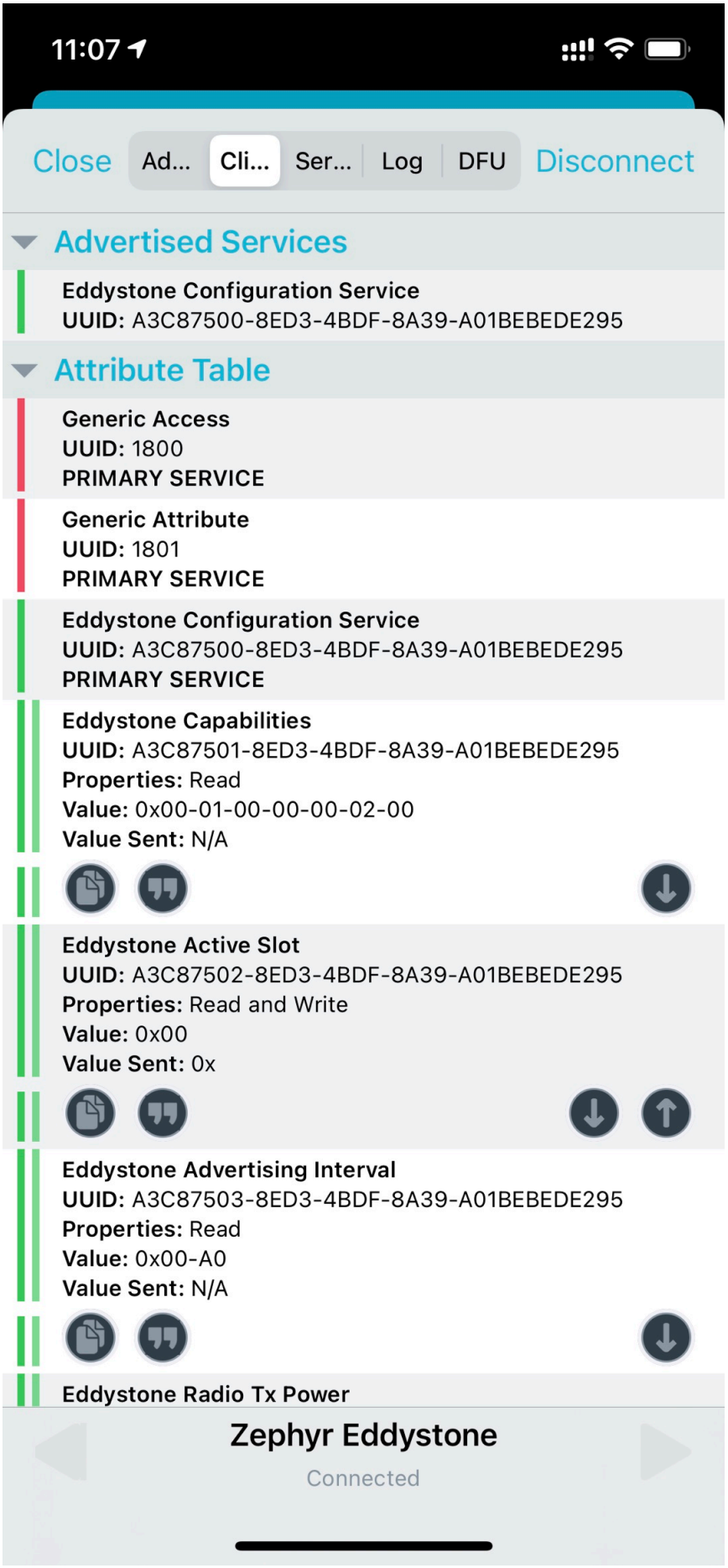
```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`，可查看源码，执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明

1. 烧录完成后，设备自动启动蓝牙广播，可以在手机或抓包工具上获取如下信息：
   - Service UUID: A3C87500-8ED3-4BDF-8A39-A01BEBEDE295
   - Device Name: Zephyr Eddystone
2. 当手机或其它主设备与其建立连接后，串口 log 会显示连接信息，如下：

   ```
   Connected
   ```

3. 连接后通过 Eddystone Configuration Service 进行读写操作，设备 log 及 app 显示如下


11:07
Close
Ad...
Cli...
Ser...
Log
DFU
Disconnect
Advertised Services
Eddystone Configuration Service
UUID: A3C87500-8ED3-4BDF-8A39-A01BEBEDE295
Attribute Table
Generic Access
UUID: 1800
PRIMARY SERVICE
Generic Attribute
UUID: 1801
PRIMARY SERVICE
Eddystone Configuration Service
UUID: A3C87500-8ED3-4BDF-8A39-A01BEBEDE295
PRIMARY SERVICE
Eddystone Capabilities
UUID: A3C87501-8ED3-4BDF-8A39-A01BEBEDE295
Properties: Read
Value: 0x00-01-00-00-00-02-00
Value Sent: N/A
Eddystone Active Slot
UUID: A3C87502-8ED3-4BDF-8A39-A01BEBEDE295
Properties: Read and Write
Value: 0x00
Value Sent: 0x
Eddystone Advertising Interval
UUID: A3C87503-8ED3-4BDF-8A39-A01BEBEDE295
Properties: Read
Value: 0x00-A0
Value Sent: N/A
Eddystone Radio Tx Power
Zephyr Eddystone
Connected

Bluetooth: HCI UART

1 功能概述 此项目演示单 BLE Controller 的功能，支持 HCI UART，可以通过串口发送 HCI 命令，用于 DTM，或者配合外部 Host 使用。

2 环境要求

- board: pan1080a_afld_evb
- uart: hci uart (8n1, 115200)
- 测试软件: Panchip Serial Assistant (或其他串口软件)

3 编译和烧录 项目位置：`zephyr\samples_panchip\bluetooth\hci_uart`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\bluetooth\hci_uart.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明

1. HCI_Reset

   串口发送 (0x) 01 03 0C 00

   串口返回 (0x) 04 0E 04 01 03 0C 00

2. LE Transmitter Test

   串口发送 (0x) 01 1E 20 03 00 25 00

   串口返回 (0x) 04 0E 04 01 1E 20 00

3. LE Test End

   串口发送 (0x) 01 1F 20 00

   串口返回 (0x) 04 0E 06 01 1F 20 00 00 00

   有关 DTM 测试流程请参考Core Specification v5.3 Volume 6 Part F.

5 Host Controller Interface 主机控制器接口 (HCI) 提供了访问蓝牙控制器功能的统一接口方法。有关接口的细节可以参考 Core Specification v5.3 Volume 4.

5.1 HCI Command packet The HCI Command packet is used to send commands to the Controller from the Host.

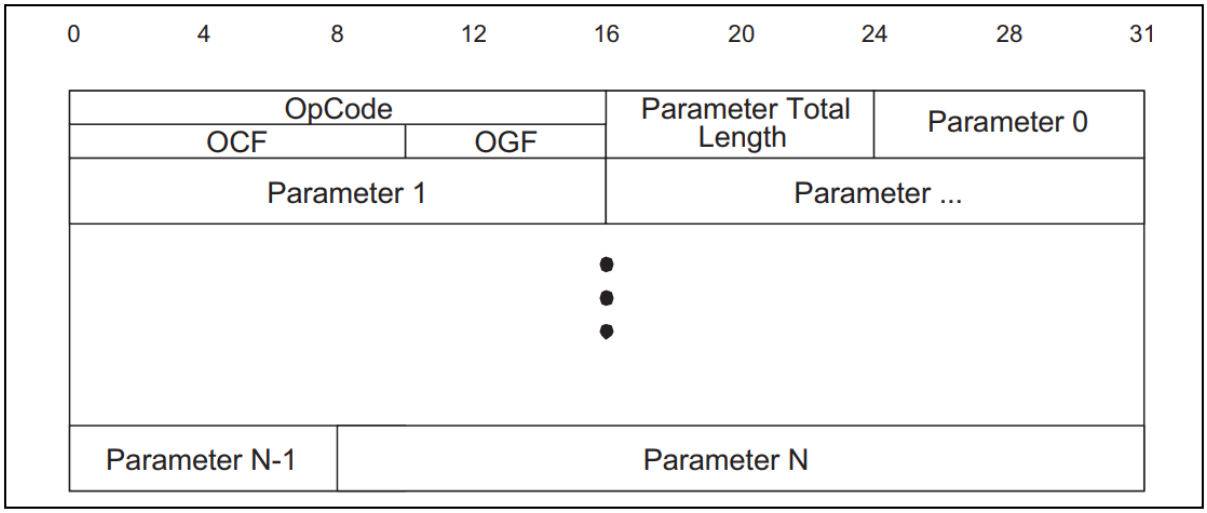


Figure 1 HCI Command packet

OpCode: (Size: 2 octets)

Value	Parameter Description
0xXXXX	OGF Range (6 bits): 0x00 to 0x3F (0x3F reserved for vendor-specific debug commands) OCF Range (10 bits): 0x0000 to 0x03FF

HCI OpCode Group Field (OGF)	Value	Description
BT_OGF_LINK_CTRL	0x01	Link Control commands
BT_OGF_BASEBAND	0x03	Controller & Baseband commands
BT_OGF_INFO	0x04	Informational parameters
BT_OGF_STATUS	0x05	Status parameters
BT_OGF_LE	0x08	LE Controller commands

Parameter_Total_Length: (Size: 1 octet)

Value	Parameter Description
0xXX	Lengths of all of the parameters contained in this packet measured in octets. (N.B.: total length of parameters, not number of parameters)

Parameter 0 - N: (Size: Parameter Total Length)

Value	Parameter Description
0xXX	Lengths of all of the parameters contained in this packet measured in octets. (N.B.: total length of parameters, not number of parameters)

5.2 HCI Event packet The HCI Event packet is used by the Controller to notify the Host when events occur.

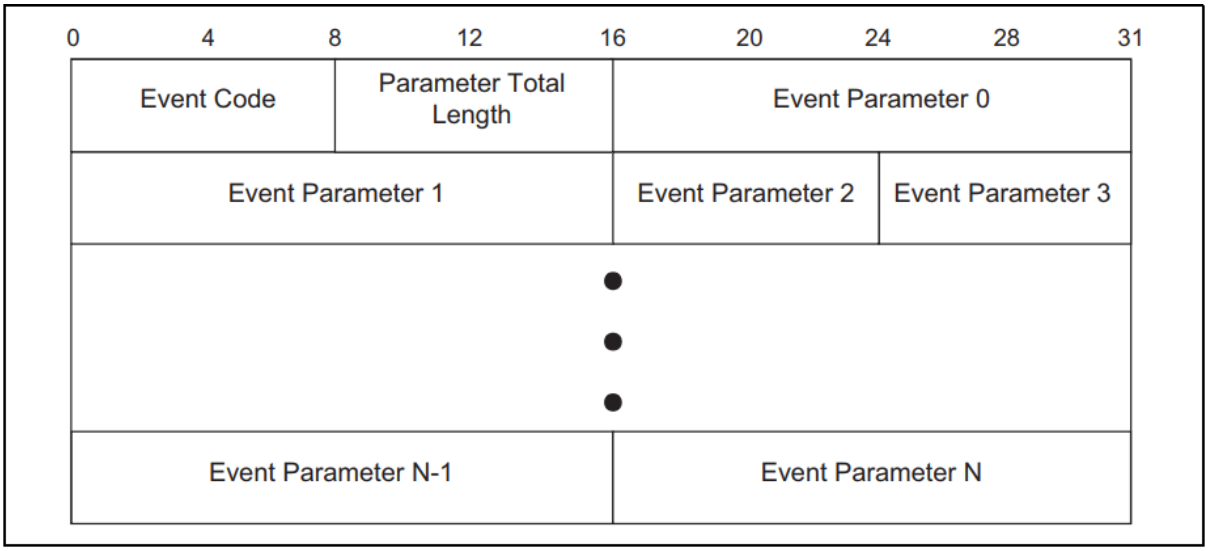


Figure 2 HCI Event packet

Event Code: (Size: 1 octet)

Value	Parameter Description
0xXX	Each event is assigned a 1-Octet event code used to uniquely identify different types of events.Range: 0x00 to 0xFF (The event code 0xFF is reserved for the event code used for vendor-specific debug events.)

Parameter_Total_Length: (Size: 1 octet)

Value	Parameter Description
0xXX	Length of all of the parameters contained in this packet, measured in octets.

Event_Parameter 0 - N: (Size: Parameter_Total_Length)

Value	Parameter Description
0xXX	Each event has a specific number of parameters associated with it. These parameters and the size of each of the parameters are defined for each event. Each parameter is an integer number of octets in size.

Bluetooth: iBeacon

1 功能概述 此项目通过广播 Apple iBeacon 演示了 BLE broadcast 角色功能。

校准的 RSSI 在 1 米距离可以使用 IBEACON_RSSI 构建变量设置 (例如，IBEACON_RSSI=0xc8 在 1 米位置为-56 dBm RSSI)，或手动编辑’‘main.c ‘‘文件中的默认值。

由于 iBeacon UUID、major 和 minor 的值是硬编码的，因此该应用程序不适合生产使用，但对于快速演示 iBeacon 功能非常方便。

2 环境要求

- board: 支持 BLE 的蓝牙设备
- uart(option): 用来显示串口 log
- 测试软件: nRF Connect

3 编译和烧录 项目位置：zephyr\samples_panchip\bluetooth\ibeacon

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：quick_build_samples\bluetooth\ibeacon.bat。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`, 可查看源码, 执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明

1. 烧录完成后，通过 nRF Connect 查看不可连接广播：
   - UUID: 18ee1516-016b-4bec-ad96-bcb96d166e97
   - Device Name: N/A(iBeacon)
   - Major：0
   - Minor：0
   - RSSI at 1m: -56 dBm

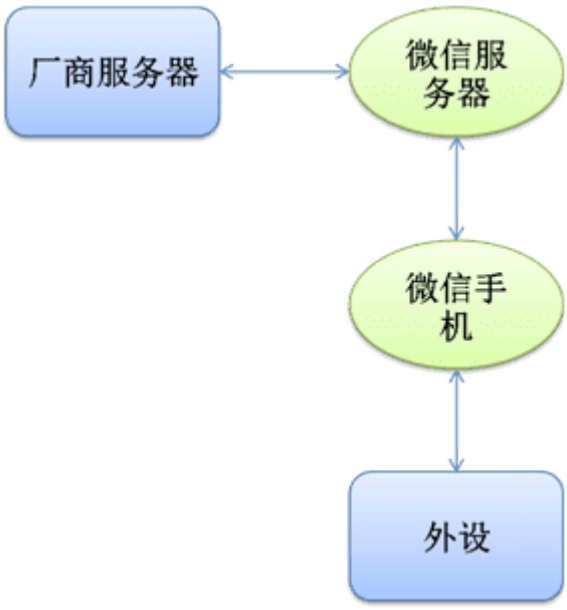
Bluetooth: IoT WeChat

1 功能概述 AirSync 是微信硬件平台提供的一种微信客户端与蓝牙设备间通讯的技术协议，它允许蓝牙设备与微信客户端之间收发数据，并支持通过微信客户端透传到远程服务器。该技术在支持微信互联的蓝牙手环、血压计、智能秤、血糖仪等设备上有比较多的应用。

AirSync 主要在如下场景中使用：

- 蓝牙设备与微信客户端间数据收发，如手环、血压计发送运动健康数据到微信客户端。
- 蓝牙设备与远端服务器收发数据，如获取微信运动排行榜名次，接收设备消息等。

协议架构图如下，其中厂商服务器和外设，由厂商开发完成。微信会提供服务器的接口以对接厂商的服务器，会提供手机的接口（规定的蓝牙协议）以对接厂商的外设，而实际上微信只是数据的中转站，用于厂商服务器与设备之间的数据传递。:



本例程包含 AirSync 协议的全部功能，通过微信硬件平台的官网提供的 AirSyncDebugger APP 测试该例程的功能是否符合协议规范。

2 环境要求

- board: pan1080a_afld_evb
- uart (option): 显示串口 log
- 测试软件: AirSyncDebugger

3 编译和烧录

项目位置：`zephyr\samples_panchip\bluetooth\iot_wechat`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\bluetooth\iot_wechat.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明

1. PAN1080 EVB 板上电后 BLE 的广播名字是 `ble_wechat`，用 AirSyncDebugger APP 可以扫描到对应的设备。
2. 在 APP 上点击设备，点击 `AirSync` 协议就可以进入测试模式，测试分为手动测试和自动测试。
3. 手动测试过程中第四步需要按下 PAN1080 EVB 上的 K1 按键方可完成测试，自动测试可以自己完成测试无需在测试过程中手动参与。
4. 测试都通过后就可以和微信公众号进行连接了。

Note: AirSyncDebugger APP 的使用可参考“微信蓝牙协议调试工具 AirSyncDebugger 说明文档 v2.0”。

5 补充说明

5.1 广播包格式 蓝牙外设广播的目的是让手机微信能够识别到。微信 AirSync 协议是这样规定蓝牙外设广播数据格式的：

- WeChat primary service UUID：0xFEE7。

  Service UUID：0x03+0x03+0xFEE7(WeChat primary service UUID)

- 在自定义厂商数据 manufature specific data 中需以 MAC 地址（身份注册时的 MAC 地址，也是该设备真实的 MAC 地址）结尾，并且要求 manufature specific data 的长度大于等于 8。

  Customer specification：0x09+0xFF+0x6101(BLE core vendor)+ 设备 MAC 地址

广播包对应的数组如下：

```
static const struct bt_data ad[] = {
        BT_DATA_BYTES(BT_DATA_FLAGS, (BT_LE_AD_GENERAL | BT_LE_AD_NO_BREDR)),
        BT_DATA_BYTES(BT_DATA_MANUFACTURER_DATA, 0x01,0x61,0xbb,0xaa,0x11,0x20,0x00,0x01),
        BT_DATA_BYTES(BT_DATA_UUID16_ALL, 0xE7, 0xFE),
};

static const struct bt_data sd[] = {
        BT_DATA(BT_DATA_NAME_COMPLETE, DEVICE_NAME, DEVICE_NAME_LEN),
};
```

5.2 GATT Service 微信蓝牙服务使用自定义的 GATT Service，如下：

Service Attribute	UUID
Primary service	0xFEE7（经蓝牙官方授权）
Write characteristic declaration	0xFEC7
Read characteristic declaration	0xFEC9
Indication characteristic declaration	0xFEC8
Indication characteristic configuration	0x2902

源码如下：

```
BT_GATT_SERVICE_DEFINE(proj_wechat,
BT_GATT_PRIMARY_SERVICE(BT_UUID_DECLARE_16(BLE_UUID_WECHAT_SERVICE)),
                       BT_GATT_CHARACTERISTIC(BT_UUID_DECLARE_16(BLE_UUID_WECHAT_WRITE_
↪CHARACTERISTICS),
                                              BT_GATT_CHRC_WRITE,
                                              BT_GATT_PERM_READ | BT_GATT_PERM_WRITE,
                                              NULL, write_value_cb, cts_data),
                       BT_GATT_CHARACTERISTIC(BT_UUID_DECLARE_16(BLE_UUID_WECHAT_READ_
↪CHARACTERISTICS),
                                              BT_GATT_CHRC_READ,
                                              BT_GATT_PERM_READ | BT_GATT_PERM_WRITE,
                                              read_value_cb, NULL, stc_data),
                       BT_GATT_CHARACTERISTIC(BT_UUID_DECLARE_16(BLE_UUID_WECHAT_INDICATE_
↪CHARACTERISTICS),
                                              BT_GATT_CHRC_INDICATE,
                                              BT_GATT_PERM_READ | BT_GATT_PERM_WRITE,
                                              NULL, NULL, temp_data),
                       BT_GATT_CCC(ccc_cfg_changed, BT_GATT_PERM_READ | BT_GATT_PERM_WRITE),
                                          );
```

满足广播包和 Profile 两个条件代表已经能够跟手机微信进行数据通信了。但是交互过程是一个协作的过程，就像我们访问业务系统一样也要先登陆，再初始化后，才能进行正常的业务通信。那么登陆和初始化我们可以理解为应用控制信令，而后续的数据通信也是应用数据通信。

5.3 Airsync 协议　Demo 例程中是通过嵌入的 data_handler 结构体实现对于不同事件的处理。

事件主要分成三大类：

- 上电初始化
- 微信 BLE 设备与微信公众号的连接
- 微信 BLE 设备与微信公众号的交互

data_handler 结构体主要包括以下内容：

```
data_handler mpbledemo2_data_handler =
{
    .m_product_type              = PRODUCT_TYPE_MPBLEDEMO2,                          //产
品类型码
    .m_data_produce_func         = &mpbledemo2_data_produce_func,                 //打包要发送的
数据
    .m_data_free_func            = &mpbledemo2_data_free_func,                       //释放申请
的内存
    .m_data_consume_func         = &mpbledemo2_data_consume_func,                 //处理收到的数
据包
    .m_data_error_func           = &mpbledemo2_data_error_func,                       //错误处
理函数
    .m_data_init_peripheral_func = &mpbledemo2_init_peripheral_func,          //外设初始化
    .m_data_init_func            = &mpbledemo2_init,                                 //
↪demo 初始化
    .m_data_main_process_func    = &mpbledemo2_main_process,                    //连接微信公众号
    .m_data_button_handler_func  = &mpbledemo2_button_handler,                       //按键事件
处理
    .m_data_produce_args         = &m_info,
↪                                                          //发送数据的 context
    .next                        = NULL
};
```

1. 连接微信

   连接微信的 Auth 和 Init 请求处理都是在 mpbledemo2_main_process 接口函数中进行的，接口如下：

   ```
   void mpbledemo2_main_process()
   {
           int error_code = 0;
           if((mpbledemo2Sta.indication_state) && (!mpbledemo2Sta.auth_state) && (!
   ↪mpbledemo2Sta.auth_send) )
           {
                   error_code = device_auth();
           if (0 == error_code)
           {
               mpbledemo2Sta.auth_send = true;
           }
           }
           if((mpbledemo2Sta.auth_state) && (!mpbledemo2Sta.init_state) && (!mpbledemo2Sta.
   ↪init_send))
           {
                   error_code = device_init();
           if (0 == error_code)
           {
               mpbledemo2Sta.init_send = true;
           }
           }
   }
   ```

   先进行 device_auth 然后再进行 device_init。

2. 与微信数据交互

- 微信发数据到设备

  设备端接收数据的接口如下：

```
int mpbledemo2_data_consume_func(uint8_t *data, uint32_t len)
{
                BpFixHead *fix_head = (BpFixHead *)data;
                uint8_t fix_head_len = sizeof(BpFixHead);
                #ifdef CATCH_LOG
                arch_printf("\r\n##Received data: ");
                uint8_t *d = data;
                for(uint8_t i=0;i<len;++i){
                arch_printf(" %x",d[i]);}
                arch_printf("\r\n CMDID: %d", ntohs(fix_head->nCmdId));
                arch_printf("\r\n len: %d", ntohs(fix_head->nLength));
                arch_printf("\r\n Seq: %d",ntohs(fix_head->nSeq));
                #endif
                switch(ntohs(fix_head->nCmdId))
                {
                        case ECI_none:
                                {
                                }
                                break;
                        .
                        .
                        .
                }
                return 0;
}
```

  设备端判断不同的”cmdid”做相应的处理。

- 设备发送数据到微信

  按下开发板的 K1 按键，设备端将数据发送到微信上。

  发送数据的接口如下：

```
// send data to wechat server
int32_t mpbledemo2_sendData(uint8_t* ptrData, uint32_t lengthInByte)
{
    uint8_t *data = NULL;
        uint32_t len = 0;
        ARGS_ITEM_SET(mpbledemo2_info, m_mpbledemo2_handler->m_data_produce_args,␣
↪cmd, CMD_SENDDAT);
        ARGS_ITEM_SET(mpbledemo2_info, m_mpbledemo2_handler->m_data_produce_args,␣
↪send_msg.len, lengthInByte);
        ARGS_ITEM_SET(mpbledemo2_info, m_mpbledemo2_handler->m_data_produce_args,␣
↪send_msg.str, (const char *)ptrData);

        m_mpbledemo2_handler->m_data_produce_func(m_mpbledemo2_handler->m_data_
↪produce_args, &data, &len);
        if(data == NULL)
        {
                return errorCodeProduce;
        }
        ble_wechat_indicate_data(data, len);
    return 0;
}
```

  Demo 是将”Hello, WeChat!”发送到公众号上。微信公众号的 ID 和 BLE 设备 ID 都是在”mpbledemo2.h”文件中修改。

Bluetooth: Mesh Demo

1 功能概述 此 sample 为针对 mesh 组网的演示项目，可以演示测试的 Mesh Feature 如下

- Friend-LPN
- Heartbeat Publish

2 环境要求

- board: pan1080a_afld_evb * 2（A&B），pan1020 Mesh Demo 板（用于测试，其他 1080 板入网 PanMesh 也可以，记为 C 板）
- uart (option): 显示串口 log
- 测试软件: PanMesh

3 编译和烧录 项目位置：`zephyr\samples_panchip\bluetooth\mesh_demo`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置 (脚本分为四种，分别配置了节点的不同模式)：

1. `quick_build_samples\bluetooth\mesh_demo.bat`(默认配置普通节点)
2. `quick_build_samples\bluetooth\mesh_demo_friend.bat`（配置为 friend 节点）
3. `quick_build_samples\bluetooth\mesh_demo_low_power_node.bat`（配置为低功耗节点）
4. `quick_build_samples\bluetooth\mesh_demo_heartbeat_publisher.bat`（配置为 heartbeat 发布者）

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明

4.1 测试 Friend-LPN 流程

1. 运行脚本 `quick_build_samples\bluetooth\mesh_demo_firend.bat`，编译后进行烧录至 pan1080a_afld_evb A
2. 运行脚本 `quick_build_samples\bluetooth\mesh_demo_low_power_node.bat`，编译后进行烧录至 pan1080a_afld_evb B
3. 观察 Friend-LPN 建立及轮询过程
4. 将 code 内的 netkey&appkey 提取到 PanMesh（或者将 PanMesh 中的 key 更新到代码内），入网其他 Mesh 板，通过 Mesh 网络对 LPN 进行开关操作，验证 Friend-LPN 交互流程

   friend 连接建立 log

   ```
   bt_mesh_friend.bt_mesh_friend_poll: Friendship established with 0x0002
   ```

   friend 开始轮询 log

```
bt_mesh_friend.buf_send_end: Waiting 30000 ms for next poll
```

lpn 连接已建立 log

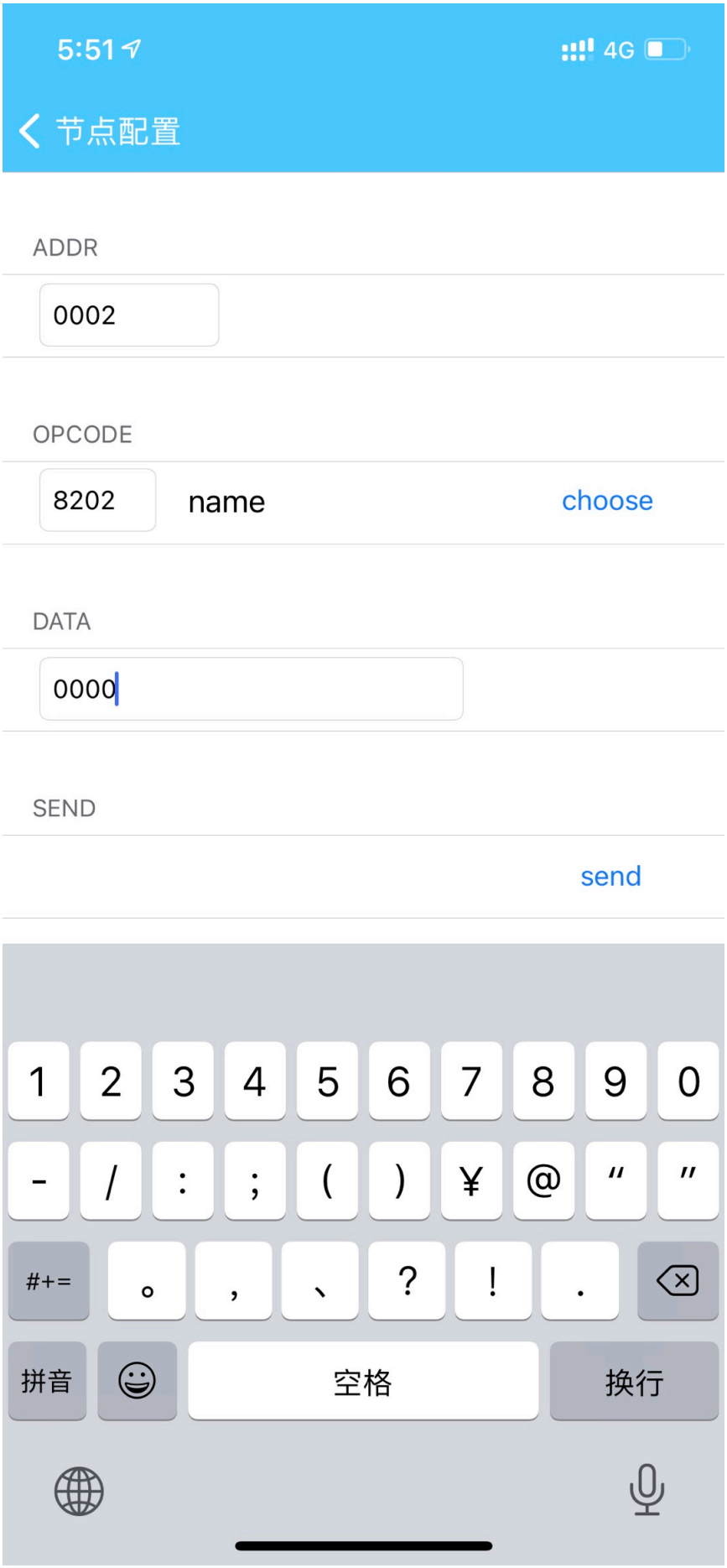
```
bt_mesh_lpn.lpn_timeout: state: established
```

lpn 轮询周期 log

```
bt_mesh_lpn.poll_timeout: Poll Timeout is 26790ms
```

5. 建立连接后，通过 PanMesh 处理好 key 信息入网 C 板，连接 C 板作为 proxy，通过 Vendor Message 界面发送给 LPN 一条控制消息如图

   0000 代表关灯，0100 代表开灯


5:51
4G
节点配置
ADDR
0002
OPCODE
8202
name
choose
DATA
0000
SEND
send
1 2 3 4 5 6 7 8 9 0
- / : ; ( ) ¥ @ " "
#+= 。 , 、 ? ! .
拼音
空格
换行

6. friend 为 lpn 存储 message log

```
bt_mesh_friend.friend_lpn_enqueue_rx: Queued message for LPN 0x0002, queue_size 1
```

7. 30s 的 friend poll 到达时，friend 为 lpn 节点发送存储消息，成功开/关灯。

4.2 测试 Heartbeat publish 流程

1. 运行脚本 quick_build_samples\bluetooth\mesh_demo.bat，编译后进行烧录至 pan1080a_afld_evb A
2. 运行脚本 quick_build_samples\bluetooth\mesh_demo_heartbeat_publisher.bat，编译后进行烧录至 pan1080a_afld_evb B
3. 观察打印 log 如下

   板 A:

   Log 打印 Subscribing to heartbeat messages

   不丢包情况下，每 4s 收取一次 heartbeat publish 消息

```
[16:51:20.480]收←*** Booting Zephyr OS build zephyr-v2.7.0-220-g27db6b8446e7  ***
Initializing...mesh_adhoc
board_init
Unicast address: 0x0003

[16:51:20.523]收←RCL_CTRL = 0x00000000

[16:51:20.648]收←Bluetooth initialized
gen_onoff_init
Mesh initialized
Provisioning completed
Configuring...
Configuration complete
Subscribing to heartbeat messages
[00:00:00.001,000] <wrn> bt_hci_core: Failed to set device name (-12)
[00:00:00.176,000] <inf> bt_hci_core: Identity: D4:4B:52:E4:BF:CD (random)
[00:00:00.176,000] <inf> bt_hci_core: HCI: version 5.1 (0x0a) revision 0x0003,␣
↪manufacturer 0x07d1
[00:00:00.176,000] <inf> bt_hci_core: LMP: version 5.1 (0x0a) subver 0x0000
[00:00:00.177,000] <inf> bt_mesh_main: Primary Element: 0x0003
[00:00:00.177,000] <dbg> bt_mesh_main.bt_mesh_provision: net_idx 0x0000 flags 0x00 iv_
↪index 0x0000

[16:51:28.517]收←heartbeat
board_heartbeat,hops = 1, feat = 1

[16:51:32.653]收←heartbeat
board_heartbeat,hops = 1, feat = 1

[16:51:40.930]收←heartbeat
board_heartbeat,hops = 1, feat = 1

[16:51:45.153]收←heartbeat
board_heartbeat,hops = 1, feat = 1
```

   板 B:

   Log 打印 Publishing heartbeat messages

```
*** Booting Zephyr OS build zephyr-v2.7.0-220-g27db6b8446e7  ***
Initializing...mesh_adhoc
board_init
```

(下页继续)

(续上页)

```
Unicast address: 0x000f
RCL_CTRL = 0x00000000
Bluetooth initialized
gen_onoff_init
Mesh initialized
Provisioning completed
Configuring...
Publishing heartbeat messages
Configuration complete
[00:00:00.001,000] <wrn> bt_hci_core: Failed to set device name (-12)
[00:00:00.175,000] <inf> bt_hci_core: Identity: FD:2D:D9:E3:14:04 (random)
[00:00:00.175,000] <inf> bt_hci_core: HCI: version 5.1 (0x0a) revision 0x0003,␣
↪manufacturer 0x07d1
[00:00:00.175,000] <inf> bt_hci_core: LMP: version 5.1 (0x0a) subver 0x0000
[00:00:00.176,000] <inf> bt_mesh_main: Primary Element: 0x000f
[00:00:00.176,000] <dbg> bt_mesh_main.bt_mesh_provision: net_idx 0x0000 flags 0x00 iv_
↪index 0x0000
```

5 开发说明

5.1 KCONFIG 宏说明

1. flash 相关，不需要存储 flash 相关功能

```
# CONFIG_BT_SETTINGS=y
# CONFIG_FLASH=y
# CONFIG_FLASH_PAGE_LAYOUT=y
# CONFIG_FLASH_MAP=y
# CONFIG_NVS=y
# CONFIG_SETTINGS=y
# CONFIG_BT_MESH_RPL_STORE_TIMEOUT=600
```

1. 调试 LOG 宏，基础宏打开，由 CONFIG_BT_PAN_MESH_NODE_TYPE 选择 friend/lpn 相关打印

```
CONFIG_BT_DEBUG_LOG=y
CONFIG_BT_MESH_DEBUG=y
```

1. 由于此 demo 可以展示配置不同 node 功能，由 zephyr\samples_panchip\bluetooth\mesh_demo\proj.conj 内的宏进行修改选择, 以下四个宏四选一进行编译

```
CONFIG_BT_PAN_MESH_NODE_TYPE_NORMAL=y
CONFIG_BT_PAN_MESH_NODE_TYPE_FRIEND=y
CONFIG_BT_PAN_MESH_NODE_TYPE_LPN=y
CONFIG_BT_PAN_MESH_NODE_TYPE_HB_PUB=y
```

5.2 MAIN 函数说明 main 函数包括 3 部分

1. flash 相关的配置擦除, 包括函数 ps_settings_init,short_time_multireset_bt_mesh_unprovisioning
2. bt_enable() 初始化蓝牙
3. bt_ready() 初始化 MESH
4. 蓝牙回调接口

```
static void connected(struct bt_conn *conn, uint8_t err)
{
        if (err) {
                printk("Connection failed (err 0x%02x)\n", err);
```

(下页继续)

(续上页)

```
        } else {
                printk("Connected\n");
        }
}

static void disconnected(struct bt_conn *conn, uint8_t reason)
{
        printk("Disconnected (reason 0x%02x)\n", reason);
}

BT_CONN_CB_DEFINE(conn_callbacks) = {
        .connected = connected,
        .disconnected = disconnected,
};
```

5.3 ble_mesh **说明** ble/mesh 相关操作:

1. 此 `mesh_demo`sample 没有进行 provisioner 入网，通过 `err = bt_mesh_provision(net_key, net_idx, flags, iv_index, addr,dev_key);` 进行自入网，通过 `static void configure(void)` 绑定相关 config 信息，为了减少 gatt 对扫描的干扰
2. `struct bt_mesh_model root_models[]` 配置 mesh model
3. 实现 heartbeat 的 publish 绑定和 onoff model 的灯控演示

Bluetooth: Mesh Echo

1 **功能概述** 此 sample 为针对智能音箱 Amazon Echo 的演示项目，可以完成入网及控制开关。

2 **环境要求**

- board: pan1080a_afld_evb
- uart (option): 显示串口 log
- Amazon Echo: 测试音箱
- 测试软件: PanMesh 或 nRF Mesh

3 **编译和烧录** 项目位置: `zephyr\samples_panchip\bluetooth\mesh_echo`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置: `quick_build_samples\bluetooth\mesh_echo.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 **演示说明**

1. 根据 Zephyr 编译环境搭建 VSCode 开发环境
2. 搭建 Echo 测试环境，不同板子需要更新不同的 DEV UUID

3. 下载即可验证 Echo 入网的功能，在 app 端显示为 Plug

5 开发说明

5.1 KCONFIG 宏说明

1. flash 相关，应用在快速重启擦除入网信息

```
CONFIG_BT_SETTINGS=y
CONFIG_FLASH=y
CONFIG_FLASH_MAP=y
CONFIG_NVS=y
CONFIG_SETTINGS=y
```

1. 调试 LOG 宏，不建议全开，log 较多占一定 ram

```
CONFIG_BT_DEBUG_LOG=y

CONFIG_BT_MESH_DEBUG=y
CONFIG_BT_DEBUG_CONN=y
CONFIG_BT_MESH_DEBUG_BEACON=y
CONFIG_BT_MESH_DEBUG_PROV=y
CONFIG_BT_MESH_DEBUG_ADV=y
CONFIG_BT_MESH_DEBUG_PROXY=y
CONFIG_BT_MESH_DEBUG_MODEL=y
CONFIG_BT_MESH_DEBUG_ACCESS=y
CONFIG_BT_MESH_DEBUG_TRANS=y
CONFIG_BT_MESH_DEBUG_KEYS=y
CONFIG_BT_MESH_DEBUG_NET=y
```

1. 广播相关参数

```
CONFIG_BT_DEVICE_NAME_DYNAMIC=y      # dynamic dev name
CONFIG_BT_DEVICE_NAME="echo"         # dev name
CONFIG_BT_MESH_UNPROV_BEACON_INT=2   # pb adv interval (s)
```

5.2 MAIN 函数说明

main 函数包括 3 部分

1. flash 相关的配置擦除, 包括函数 `ps_settings_init`,`short_time_multireset_bt_mesh_unprovisioning`
2. `bt_enable()` 初始化蓝牙
3. `bt_ready()` 初始化 MESH
4. 蓝牙回调接口

```
static void connected(struct bt_conn *conn, uint8_t err)
{
        if (err) {
                printk("Connection failed (err 0x%02x)\n", err);
        } else {
                printk("Connected\n");
        }
}

static void disconnected(struct bt_conn *conn, uint8_t reason)
{
        printk("Disconnected (reason 0x%02x)\n", reason);
}

BT_CONN_CB_DEFINE(conn_callbacks) = {
```

(下页继续)

(续上页)

```
        .connected = connected,
        .disconnected = disconnected,
};
```

5.3 ble_mesh 说明 ble/mesh 相关操作:

1. `static const struct bt_mesh_prov prov` 配置入网参数
2. `struct bt_mesh_model root_models[]` 配置 mesh model
3. UUID 生成方式：静态数组 `static uint8_t dev_uuid_echo[16]`
4. 入网认证方式：`no oob`（Echo 选择的，因此不需要类似天猫小度的三元组静态 oob）
5. 入网 beaer 选择：`bt_mesh_prov_enable(BT_MESH_PROV_ADV | BT_MESH_PROV_GATT);`

5.4 其他说明

1. `light model` 和 `config model`，`health model` 请参考 [Mesh 开发指南](../../dev_guides/Developing Bluetooth Mesh Applications.md)。
2. 目前只添加了 `Generic Onoff Model`，其它灯控相关的 `CTL`，`HSL` 等将在后续版本中添加。
3. 入网方式为 `Pb gatt`，通过连接方式进行入网，入网后每次连接会进行 `proxy configuration` 和 `filter add` 等操作。

Bluetooth: Mesh Provisioner

1 **功能概述** 此 sample 为 pan1080a_afld_evb 作为 mesh_provisioner 的实例演示项目，可以演示作为 provisioner，先进行自我配置 netkey，存储，然后通过 pb adv 对其他待入网设备广播进行扫描，建立 link，入网的流程，并包括后续的配置流程。

2 环境要求

- board: pan1080a_afld_evb
- uart (option): 显示串口 log

  需要烧录有 `mesh_echo` 或支持 Mesh 入网功能的设备。

3 **编译和烧录** 项目位置：`zephyr\samples_panchip\bluetooth\mesh_provisioner`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\bluetooth\mesh_provisioner.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明

1. 根据 Zephyr 编译环境搭建 VSCode 开发环境, 准备至少 1 块其他待入网设备，发送 beacon 如 1020.
2. 下载前需要 erase chip（CONFIG_BT_SETTINGS=y 情况下需要，因为需要 stored cdb，防止 flash 冲突），CONFIG_BT_SETTINGS=n 情况下不需要
3. 烧录完成后观察入网，当附近没有待入网设备时，provisioner 持续扫描，当附近存在待入网设备时，provisioner 会入网第一个扫描的设备，并尝试配置
4. 配置由于 adv 扫描效率问题，可能会出现失败，这时可以采用复位已入网的设备继续流程（不是擦除入网信息，是保留入网状态条件的复位）

5 开发说明

5.1 KCONFIG 宏说明

1. flash 相关

   CONFIG_BT_SETTINGS 建议关闭, 此宏控制入网信息是否保存至 provisioner 的 CDB (configuration database)

```
# Store Bluetooth state and configuration persistently
CONFIG_BT_SETTINGS=n
CONFIG_BT_MESH_RPL_STORE_TIMEOUT=600

# menuconfig Enable support for the flash hardware.
CONFIG_FLASH=y
CONFIG_FLASH_PAGE_LAYOUT=y #Enables API for retrieving the layout of flash memory pages.

# menuconfig Enable support of flash map abstraction.
CONFIG_FLASH_MAP=y

# Enable support of Non-volatile Storage.
CONFIG_NVS=y

# menuconfig users settings store
CONFIG_SETTINGS=y
```

2. Provisioner 相关
   1. 选择 provisioner
   2. CDB 使能
   3. 网络 node 最大 2：除去 provisioner 可以再入 1 个

```
# need config as this to be noticed as provisioner rather than a provisionee
CONFIG_BT_MESH_PROVISIONER=y
CONFIG_BT_MESH_PROV_DEVICE=n
# creat a configuration database not use stored
CONFIG_BT_MESH_CDB=y
# count of net config,can only provision other 1 node
CONFIG_BT_MESH_CDB_NODE_COUNT=2
CONFIG_BT_MESH_CDB_SUBNET_COUNT=3
CONFIG_BT_MESH_CDB_APP_KEY_COUNT=3
```

3. 调试 LOG 宏，不建议全开，log 较多占一定 ram

```
CONFIG_BT_DEBUG_LOG=y

CONFIG_BT_MESH_DEBUG=y
CONFIG_BT_DEBUG_CONN=y
```

（下页继续）

(续上页)

```
CONFIG_BT_MESH_DEBUG_BEACON=y
CONFIG_BT_MESH_DEBUG_PROV=y
CONFIG_BT_MESH_DEBUG_ADV=y
CONFIG_BT_MESH_DEBUG_PROXY=y
CONFIG_BT_MESH_DEBUG_MODEL=y
CONFIG_BT_MESH_DEBUG_ACCESS=y
CONFIG_BT_MESH_DEBUG_TRANS=y
CONFIG_BT_MESH_DEBUG_KEYS=y
CONFIG_BT_MESH_DEBUG_NET=y
```

4. mesh 相关参数

```
CONFIG_BT_MESH=y
CONFIG_BT_MESH_SUBNET_COUNT=1
CONFIG_BT_MESH_APP_KEY_COUNT=1
CONFIG_BT_MESH_ADV_BUF_COUNT=10
CONFIG_BT_MESH_TX_SEG_MSG_COUNT=3
CONFIG_BT_MESH_RX_SEG_MAX=32
CONFIG_BT_MESH_MODEL_GROUP_COUNT=2
CONFIG_BT_MESH_LABEL_COUNT=0
CONFIG_BT_MESH_CFG_CLI=y
CONFIG_BT_MESH_HEALTH_CLI=y
CONFIG_BT_MESH_BEACON_ENABLED=n
CONFIG_BT_MESH_RELAY=y
CONFIG_BT_MESH_RELAY_RETRANSMIT_COUNT=3
```

5. ble 相关参数

```
CONFIG_BT=y
CONFIG_BT_TINYCRYPT_ECC=y

# Select this for LE Observer role support
CONFIG_BT_OBSERVER=y
# Select this for LE Broadcaster role support.
CONFIG_BT_BROADCASTER=y
```

5.2 MAIN 函数说明

main 函数包括 3 部分：

- `bt_enable()` 初始化蓝牙
- `bt_ready()` 初始化 MESH
- 信号量操作 `k_sem_reset`

```
k_sem_reset(&sem_unprov_beacon);
k_sem_reset(&sem_node_added);
```

先 reset 信号量标志，再通过 sem_unprov_beacon 或者 sem_node_added 的回调 give 信号量，while (1) 持续获取 take 以上信号量完成循环入网/配置流程：

1. 先获取配置信息，首先配置自己，由于 `CONFIG_BT_SETTINGS=n`, 因此不会记录入网信息，重启后复原，不会采用 stored cdb
2. 搜索广播包，由于配置最大网络个数 2，`CONFIG_BT_MESH_CDB_NODE_COUNT=2`, 还可以入网 1 个设备
3. 入网后会对该设备进行配置

   ```
   static void configure_node(struct bt_mesh_cdb_node *node)
   ```

   配置包括 get composition data，add appkey，bind appkey to each model as usual.

5.3 其他说明

1. 入网演示走的流程是 pb adv 广播方式入网，包含的 zephyr 应用库 subsys\bluetooth\mesh 下的 `pb_adv.c prov.c prov_bearer.c provisioner.c` 等源码，运行 sample 可以参考以上。
2. 对比原始 sample，provisioner 调整过 CONFIG_MAIN_STACK_SIZE=2048 栈大小
3. 对比原始 sample，由于入网 adv 扫描在 pan1080a_afld_evb 上还待优化，修改过入网超时时间

   ```
   err = k_sem_take(&sem_node_added, K_SECONDS(30));
   ```

   入网不会失败，但会经过较长时间。

Bluetooth: Mesh Speaker

1 功能概述 此 sample 为针对智能音箱（小度音箱/天猫精灵）的演示项目，可以完成入网及灯控过程。可以通过 CONFIG 控制不同设备类型/同时支持两种音箱。

2 环境要求

- board: pan1080a_afld_evb
- uart (option): 显示串口 log
- 音响：小度音箱，天猫精灵；同时需要准备相对应的三元组
- 测试软件：PanMesh 或 nRF Mesh

3 编译和烧录 项目位置：`zephyr\samples_panchip\bluetooth\mesh_speaker`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\bluetooth\mesh_speaker.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明

1. 根据 Zephyr 编译环境搭建 VSCode 开发环境
2. 天猫小度三元组在 `ble_mesh.c` 对应填充, 多块板子需要填充不同的三元组
3. 下载即可验证小度/天猫同时入网的功能。入网前循环打印小度/天猫的 UUID unpro beacon

   ```
   [00:45:23.777,000] <dbg> bt_mesh_beacon.beacon_send:
   [00:45:23.777,000] <dbg> bt_mesh_beacon.unprovisioned_beacon_send:
   [00:45:23.777,000] <dbg> bt_mesh_beacon.unprovisioned_beacon_send: beacon_flag tmall
   [00:45:25.779,000] <dbg> bt_mesh_beacon.beacon_send:
   [00:45:25.779,000] <dbg> bt_mesh_beacon.unprovisioned_beacon_send:
   [00:45:25.779,000] <dbg> bt_mesh_beacon.unprovisioned_beacon_send: beacon_flag xiaodu
   ```

4. 可以尝试更新 CONFIG_BT_MESH_UNPROV_BEACON_INT 来加快或延长 unprovision 广播发送时间。

5 开发说明

5.1 KCONFIG 宏说明

1. flash 相关，应用在快速重启擦除入网信息

```
CONFIG_BT_SETTINGS=y
CONFIG_FLASH=y
CONFIG_FLASH_MAP=y
CONFIG_NVS=y
CONFIG_SETTINGS=y
```

1. 调试 LOG 宏，不建议全开，log 较多占一定 ram

```
CONFIG_BT_DEBUG_LOG=y

CONFIG_BT_MESH_DEBUG=y
CONFIG_BT_DEBUG_CONN=y
CONFIG_BT_MESH_DEBUG_BEACON=y
CONFIG_BT_MESH_DEBUG_PROV=y
CONFIG_BT_MESH_DEBUG_ADV=y
CONFIG_BT_MESH_DEBUG_PROXY=y
CONFIG_BT_MESH_DEBUG_MODEL=y
CONFIG_BT_MESH_DEBUG_ACCESS=y
CONFIG_BT_MESH_DEBUG_TRANS=y
CONFIG_BT_MESH_DEBUG_KEYS=y
CONFIG_BT_MESH_DEBUG_NET=y
```

1. 广播相关参数

```
CONFIG_BT_DEVICE_NAME_DYNAMIC=y       # dynamic dev name
CONFIG_BT_DEVICE_NAME="speaker"       # dev name
CONFIG_BT_MESH_UNPROV_BEACON_INT=2    # pb adv interval (s)
CONFIG_BT_MESH_MULTIPLE_BEACON=y      # y means can provision both speakers
```

5.2 MAIN 函数说明

main 函数包括 3 部分

1. flash 相关的配置擦除, 包括函数 `ps_settings_init`,`short_time_multireset_bt_mesh_unprovisioning`
2. `bt_enable()` 初始化蓝牙
3. `bt_ready()` 初始化 MESH

5.3 ble_mesh 说明

ble/mesh 相关操作:

1. `static const struct bt_mesh_prov prov` 配置入网参数
2. `struct bt_mesh_model root_models[]` 配置 mesh model
3. 天猫小度切换入网流程

5.4 其他说明

1. speaker_operate 为三元组生成 beacon dev uuid 和 auth_static 相关操作，通用于小度，天猫，小米等智能网关，根据三元组（pid，mac，sec）生成 UUID 和静态 oob 认证 data。
2. light model 和 config model，health model 相关参考后续介绍 model 文档，参考 spec 文档。
3. 小度天猫 vendor 暂未加入，后续需要添加至 `subsys/bluetooth/mesh_models` 中
4. 小度入网后，由 heartbeat 监测设备在线状态，**目前暂未完整验证**，所以可能会上报设备不在线状态。

Bluetooth: Multi-roles

1 **功能概述** 此项目演示了蓝牙多角色的功能。可以通过 zephyr shell 控制设备执行广播、扫描、连接（主和从）等。

2 环境要求

- board: pan1080a_afld_evb
- uart: 显示串口 log 和 shell 交互
- 测试软件：SecureCRT，用于 shell 交互
- 测试软件 (option): nRF Connect

  执行多连接时，需要多个支持蓝牙的设备。

3 **编译和烧录** 项目位置：zephyr\samples_panchip\bluetooth\mult_roles

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：quick_build_samples\bluetooth\mult_roles.bat。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明

4.1 启动广播

```
uart:~$ ble adv conn_ind_start
Advertising successfully started
```

4.2 启动扫描

```
uart:~$ ble scan passive_start
Scanning successfully started
uart:~$ 00 > 37:09:A7:33:53:B6 (random) (RSSI -98) (ADV_NONCONN_IND)
01 > 39:E9:F4:88:25:21 (random) (RSSI -95) (ADV_NONCONN_IND)
02 > 3B:2C:F8:A6:0D:85 (random) (RSSI -95) (ADV_NONCONN_IND)
03 > 25:A0:8B:A9:CE:CB (random) (RSSI -98) (ADV_NONCONN_IND)
04 > 1A:E1:B2:2D:BC:1A (random) (RSSI -94) (ADV_NONCONN_IND)
05 > 74:AC:D5:CF:97:3A (random) (RSSI -92) (ADV_IND)
06 > 0A:6F:3B:F4:FC:77 (random) (RSSI -93) (ADV_NONCONN_IND)
07 > 00:20:21:05:27:FF (public) (RSSI -92) (ADV_IND)
08 > BB:CA:11:20:00:06 (public) (RSSI -91) (ADV_IND)
09 > 01:95:C8:38:2E:C1 (random) (RSSI -81) (ADV_NONCONN_IND)
10 > 7B:7B:2B:BD:A7:BA (random) (RSSI -81) (ADV_NONCONN_IND)
11 > 3E:18:64:B8:8A:B4 (random) (RSSI -92) (ADV_NONCONN_IND)
12 > 26:D1:B3:49:78:A5 (random) (RSSI -92) (ADV_NONCONN_IND)
13 > 12:1D:7C:92:53:B1 (random) (RSSI -85) (ADV_NONCONN_IND)
14 > 64:D6:45:C8:ED:3C (random) (RSSI -86) (ADV_IND)
15 > FF:FF:FF:FF:FF:FF (public) (RSSI -75) (ADV_IND)
```

(下页继续)

(续上页)

```
16 > 21:88:3B:73:56:62 (random) (RSSI -94) (ADV_NONCONN_IND)
17 > 2B:46:04:9D:A9:08 (random) (RSSI -88) (ADV_NONCONN_IND)
18 > 1A:A0:5E:42:34:6E (random) (RSSI -88) (ADV_NONCONN_IND)
19 > 4E:E5:97:61:EB:3B (random) (RSSI -88) (ADV_IND)
```

需要注意的是:

- 当前可以存储 20 条设备信息，当扫描到的设备信息存储满时，将**自动终止扫描**。
- 发起连接（`ble conn create`）时如果当前处于扫描状态，则会终止扫描后启动连接

4.3 显示设备信息

显示扫描、连接设备列表信息。

```
uart:~$ ble devs show
Detected List:
00 > 37:09:A7:33:53:B6 (random) (RSSI -98) (ADV_NONCONN_IND)
01 > 39:E9:F4:88:25:21 (random) (RSSI -95) (ADV_NONCONN_IND)
02 > 3B:2C:F8:A6:0D:85 (random) (RSSI -95) (ADV_NONCONN_IND)
03 > 25:A0:8B:A9:CE:CB (random) (RSSI -98) (ADV_NONCONN_IND)
04 > 1A:E1:B2:2D:BC:1A (random) (RSSI -94) (ADV_NONCONN_IND)
05 > 74:AC:D5:CF:97:3A (random) (RSSI -92) (ADV_IND)
06 > 0A:6F:3B:F4:FC:77 (random) (RSSI -93) (ADV_NONCONN_IND)
07 > 00:20:21:05:27:FF (public) (RSSI -92) (ADV_IND)
08 > BB:CA:11:20:00:06 (public) (RSSI -91) (ADV_IND)
09 > 01:95:C8:38:2E:C1 (random) (RSSI -81) (ADV_NONCONN_IND)
10 > 7B:7B:2B:BD:A7:BA (random) (RSSI -81) (ADV_NONCONN_IND)
11 > 3E:18:64:B8:8A:B4 (random) (RSSI -92) (ADV_NONCONN_IND)
12 > 26:D1:B3:49:78:A5 (random) (RSSI -92) (ADV_NONCONN_IND)
13 > 12:1D:7C:92:53:B1 (random) (RSSI -85) (ADV_NONCONN_IND)
14 > 64:D6:45:C8:ED:3C (random) (RSSI -86) (ADV_IND)
15 > FF:FF:FF:FF:FF:FF (public) (RSSI -75) (ADV_IND)
16 > 21:88:3B:73:56:62 (random) (RSSI -94) (ADV_NONCONN_IND)
17 > 2B:46:04:9D:A9:08 (random) (RSSI -88) (ADV_NONCONN_IND)
18 > 1A:A0:5E:42:34:6E (random) (RSSI -88) (ADV_NONCONN_IND)
19 > 4E:E5:97:61:EB:3B (random) (RSSI -88) (ADV_IND)
Connected List:
- Empty
```

4.4 与设备建立连接

在与设备建立连接时，需要先获取待连接设备的编号，可以通过 `ble devs show` 获取。

如下命令，表示将与 `Detected List` 列表中编号为 `05` 的设备建立连接:

```
uart:~$ ble conn create 5
Connecting ...
uart:~$ Connect to 74:AC:D5:CF:97:3A (random) successful
```

5 其它说明

5.1 配置

项目配置文件: `prj.conf`，其中包含了 `Bluetooth`、`shell`、`debug` 等的一些配置。

其中支持的最大连接数量由如下参数配置:

```
CONFIG_BT_MAX_CONN=8
```

5.2 shell 命令

`shell` 支持的 `ble` 命令

```
ble - BLE commands
Subcommands:
  adv    :LE Advertising commands
  Subcommands:
    conn_ind_start     :LE Start Advertising with ADV_IND command
    nonconn_ind_start  :LE Start Advertising with ADV_NONCONN_IND command
    stop               :LE Stop Advertising command
  scan   :LE Scanning commands
  Subcommands:
    passive_start  :LE Start Passive Scanning command
    active_start   :LE Start Active Scanning command
    stop           :LE Stop Scanning command
  conn   :LE Connection commands
  Subcommands:
    create   :LE Create Connection command
    disconn  :Disconnect command
  devs   :LE Device Status commands
  Subcommands:
    show   :Display BLE Devices command
    clear  :Clear Detected List command
  reset  :Reset commands
```

Bluetooth: Peripheral

1 功能概述 此项目演示了蓝牙从机的功能，该工程包含了 Heart Rate 服务，可以上报心率数据给主机。

在不支持心率的设备上，这些心率数据是虚拟的。

2 环境要求

- board: 支持 BLE 的蓝牙设备
- uart(option): 用来显示串口 log
- 测试软件: nRF Connect

3 编译和烧录 项目位置：'zephyr\samples_panchip\bluetooth\peripheral

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\bluetooth\peripheral.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                       编译项目
  'r' make clean and rebuild      重新编译项目
  'f' flash download              下载
  'e' erase chip                  擦除芯片
  'o' open project by VS Code     打开 `VS Code`，可查看源码，执行编译下载等
  others exit                     退出
wait input:
```

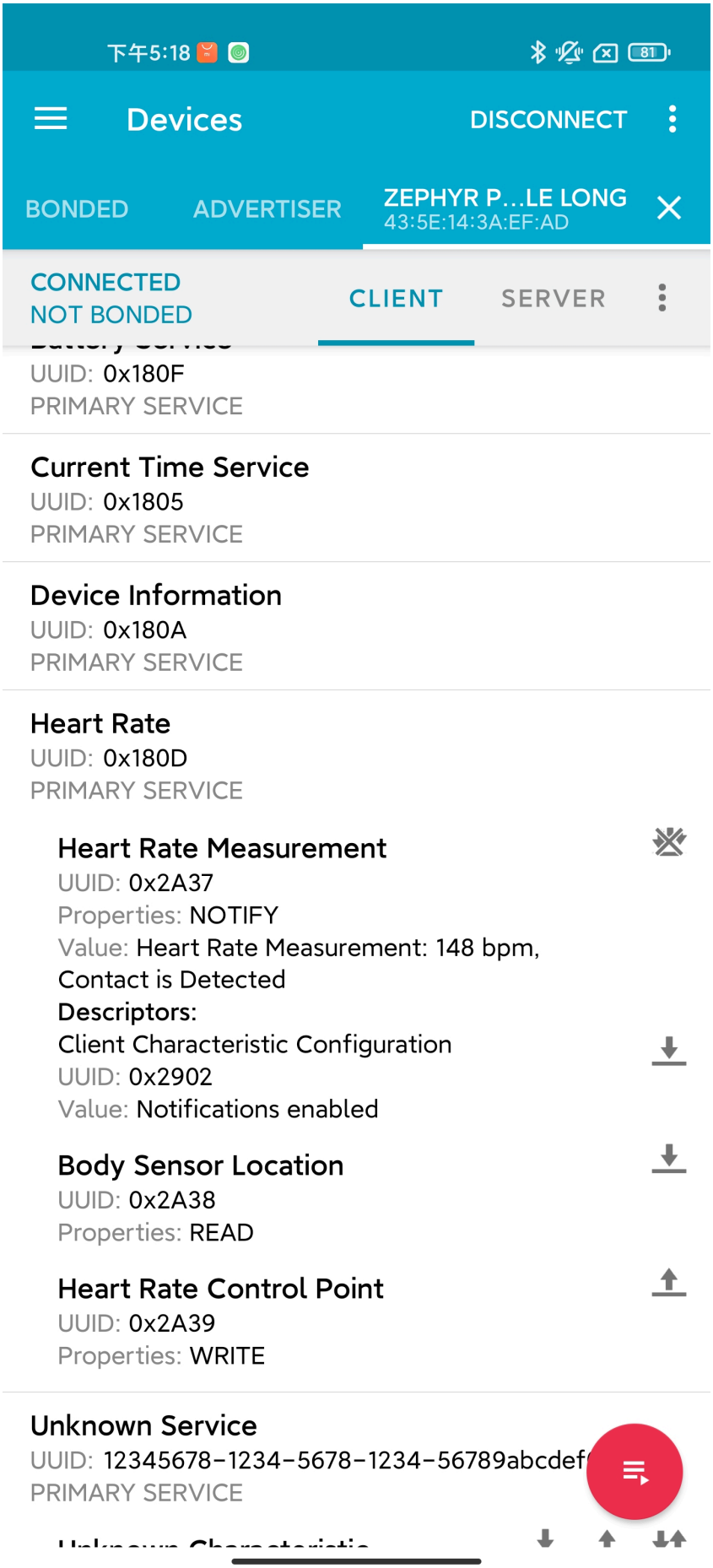
4 演示说明

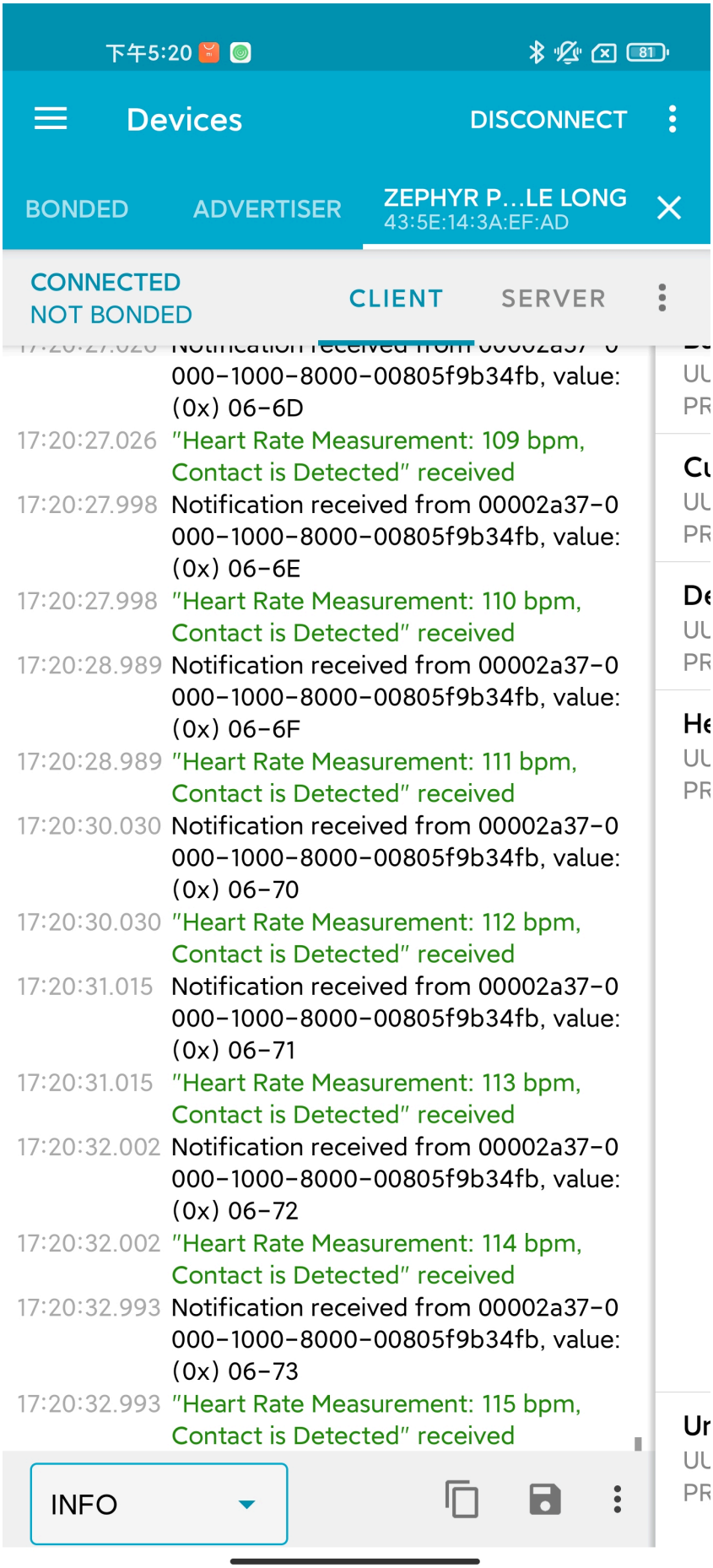
1. 烧录完成后，设备自动启动蓝牙广播，可以在手机或抓包工具上获取如下信息：
   - Service UUID: 0x1805, 0x180D, 0x180F
   - Device Name: Zephyr Peripheral Sample Long

2. 当手机或其它主设备与其建立连接后，串口 log 会显示连接信息，如下：

```
Connected
```

3. 此时，如果使能了 `Heart Rate Measurement`，主机将收到心率数据，如下：


下午5:18
Devices
DISCONNECT
BONDED
ADVERTISER
ZEPHYR P...LE LONG
43:5E:14:3A:EF:AD
CONNECTED
NOT BONDED
CLIENT
SERVER
UUID: 0x180F
PRIMARY SERVICE
Current Time Service
UUID: 0x1805
PRIMARY SERVICE
Device Information
UUID: 0x180A
PRIMARY SERVICE
Heart Rate
UUID: 0x180D
PRIMARY SERVICE
Heart Rate Measurement
UUID: 0x2A37
Properties: NOTIFY
Value: Heart Rate Measurement: 148 bpm,
Contact is Detected
Descriptors:
Client Characteristic Configuration
UUID: 0x2902
Value: Notifications enabled
Body Sensor Location
UUID: 0x2A38
Properties: READ
Heart Rate Control Point
UUID: 0x2A39
Properties: WRITE
Unknown Service
UUID: 12345678-1234-5678-1234-56789abcdef
PRIMARY SERVICE


下午5:20
Devices
DISCONNECT
BONDED
ADVERTISER
ZEPHYR P...LE LONG
43:5E:14:3A:EF:AD
CONNECTED
NOT BONDED
CLIENT
SERVER
000-1000-8000-00805f9b34fb, value: (0x) 06-6D
17:20:27.026 "Heart Rate Measurement: 109 bpm, Contact is Detected" received
17:20:27.998 Notification received from 00002a37-0000-1000-8000-00805f9b34fb, value: (0x) 06-6E
17:20:27.998 "Heart Rate Measurement: 110 bpm, Contact is Detected" received
17:20:28.989 Notification received from 00002a37-0000-1000-8000-00805f9b34fb, value: (0x) 06-6F
17:20:28.989 "Heart Rate Measurement: 111 bpm, Contact is Detected" received
17:20:30.030 Notification received from 00002a37-0000-1000-8000-00805f9b34fb, value: (0x) 06-70
17:20:30.030 "Heart Rate Measurement: 112 bpm, Contact is Detected" received
17:20:31.015 Notification received from 00002a37-0000-1000-8000-00805f9b34fb, value: (0x) 06-71
17:20:31.015 "Heart Rate Measurement: 113 bpm, Contact is Detected" received
17:20:32.002 Notification received from 00002a37-0000-1000-8000-00805f9b34fb, value: (0x) 06-72
17:20:32.002 "Heart Rate Measurement: 114 bpm, Contact is Detected" received
17:20:32.993 Notification received from 00002a37-0000-1000-8000-00805f9b34fb, value: (0x) 06-73
17:20:32.993 "Heart Rate Measurement: 115 bpm, Contact is Detected" received
INFO

Bluetooth: Peripheral CSC

1 功能概述 此项目演示了蓝牙从机的功能，该工程专门演示 CSC(Cycling Speed and Cadence)GATT 服务。

2 环境要求

- board: 支持 BLE 的蓝牙设备
- uart(option): 用来显示串口 log
- 测试软件: nRF Connect,nRF Toolbox

3 编译和烧录 项目位置：zephyr\samples_panchip\bluetooth\peripheral_csc

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：quick_build_samples\bluetooth\peripheral_csc.bat。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明

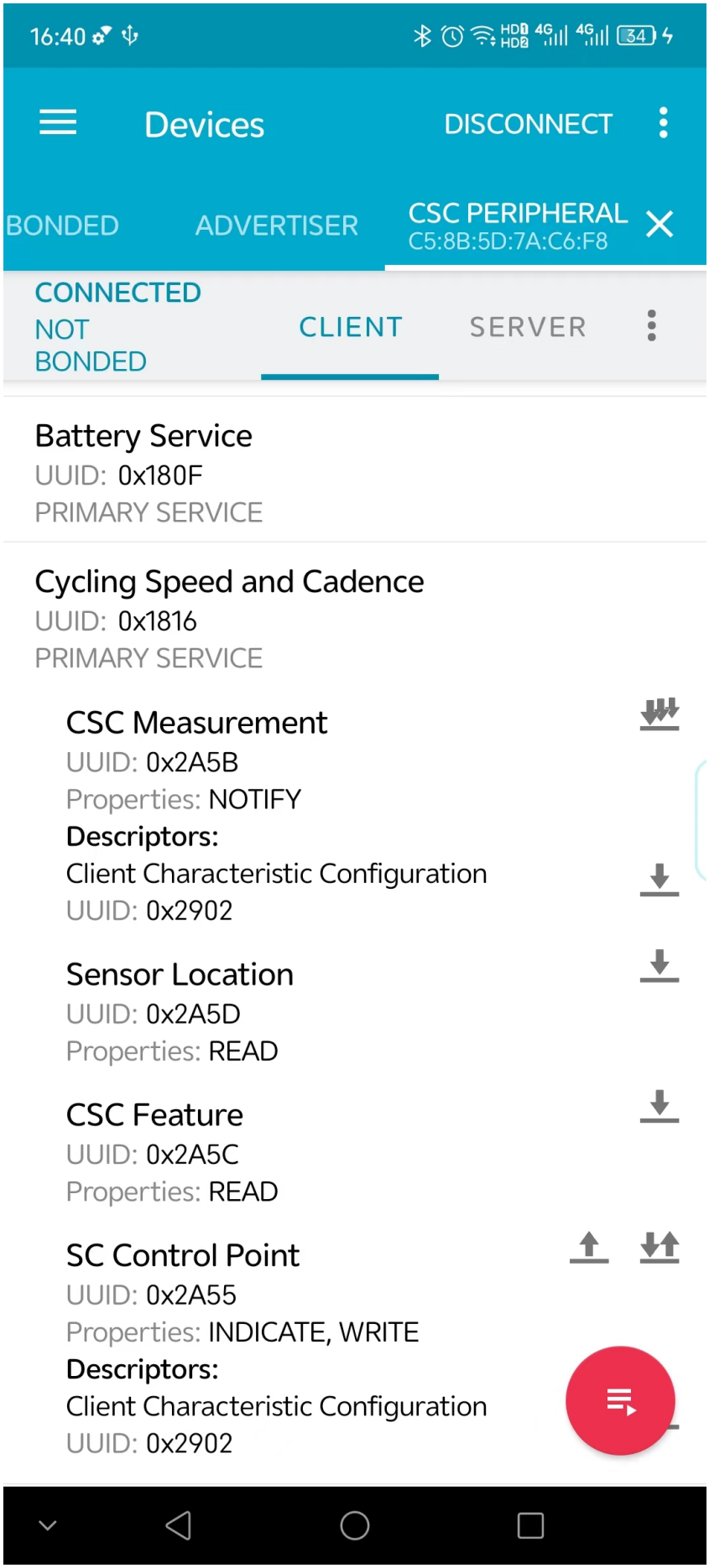
1. 烧录完成后，设备自动启动蓝牙广播，可以在手机或抓包工具上获取如下信息:

- Service UUID: 0x1809, 0x1816, 0x180F
- Device Name: CSC_PERIPHERAL

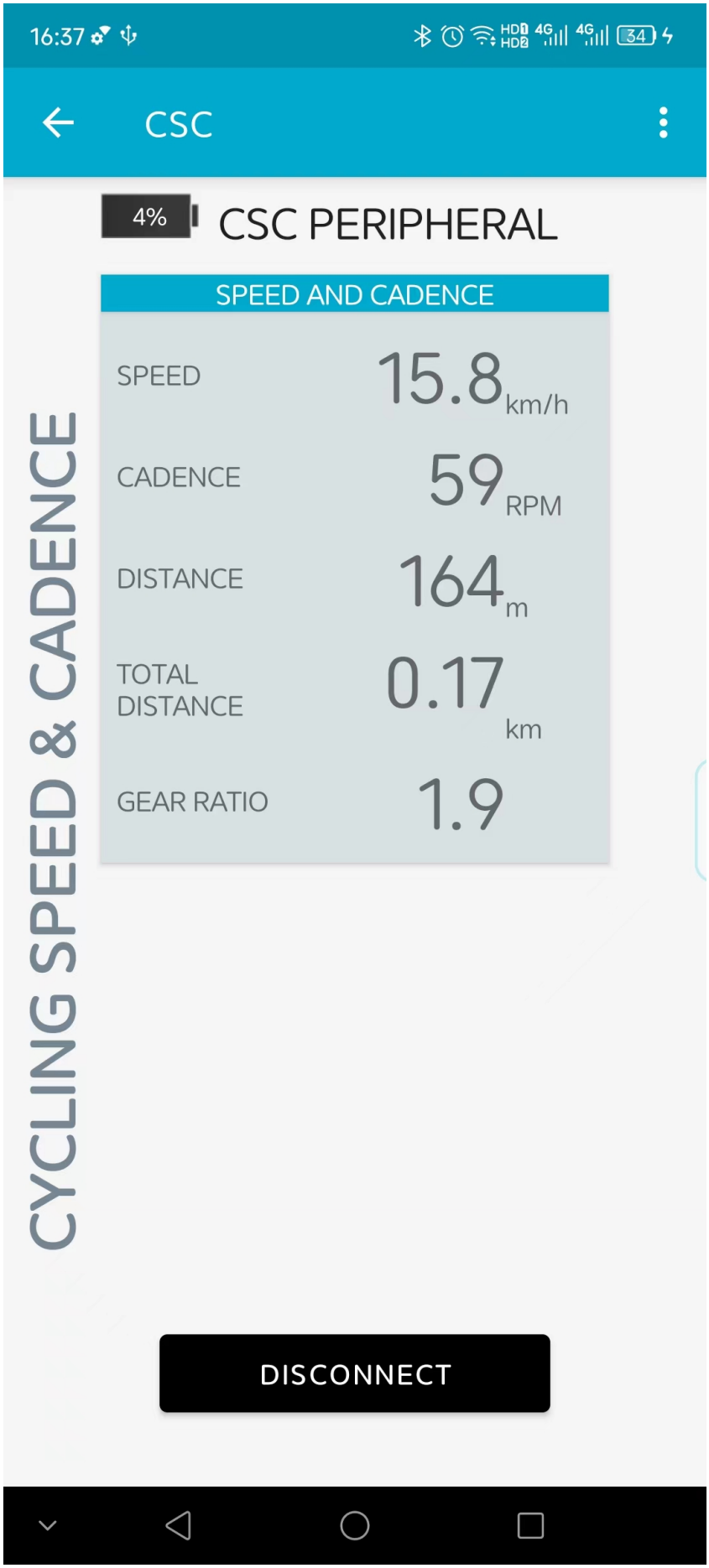
1. 当手机或其它主设备与其建立连接后，串口 log 会显示连接信息，如下:

```
Connected
```

1. 通过 nRF Connect App 连接 CSC_PERIPHERAL 设备显示如下服务:


16:40
Devices
DISCONNECT
BONDED
ADVERTISER
CSC PERIPHERAL
C5:8B:5D:7A:C6:F8
CONNECTED
NOT
BONDED
CLIENT
SERVER
Battery Service
UUID: 0x180F
PRIMARY SERVICE
Cycling Speed and Cadence
UUID: 0x1816
PRIMARY SERVICE
CSC Measurement
UUID: 0x2A5B
Properties: NOTIFY
Descriptors:
Client Characteristic Configuration
UUID: 0x2902
Sensor Location
UUID: 0x2A5D
Properties: READ
CSC Feature
UUID: 0x2A5C
Properties: READ
SC Control Point
UUID: 0x2A55
Properties: INDICATE, WRITE
Descriptors:
Client Characteristic Configuration
UUID: 0x2902

1. 通过 `nRF Toolbox` 连接 CSC_PERIPHERAL 设备后动态刷新 SPEED,CADENCE 相关数据：


16:37
CSC
4%
CSC PERIPHERAL
SPEED AND CADENCE
SPEED
15.8 km/h
CADENCE
59 RPM
DISTANCE
164 m
TOTAL DISTANCE
0.17 km
GEAR RATIO
1.9
CYCLING SPEED & CADENCE
DISCONNECT

Bluetooth: Peripheral DIS

1 功能概述 此项目演示了蓝牙从机的功能，该工程专门演示 DIS(Device Information)GATT 服务。

2 环境要求

- board: 支持 BLE 的蓝牙设备
- uart(option): 用来显示串口 log
- 测试软件: nRF Connect

3 编译和烧录 项目位置：zephyr\samples_panchip\bluetooth\peripheral_dis

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：quick_build_samples\bluetooth\peripheral_dis.bat。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

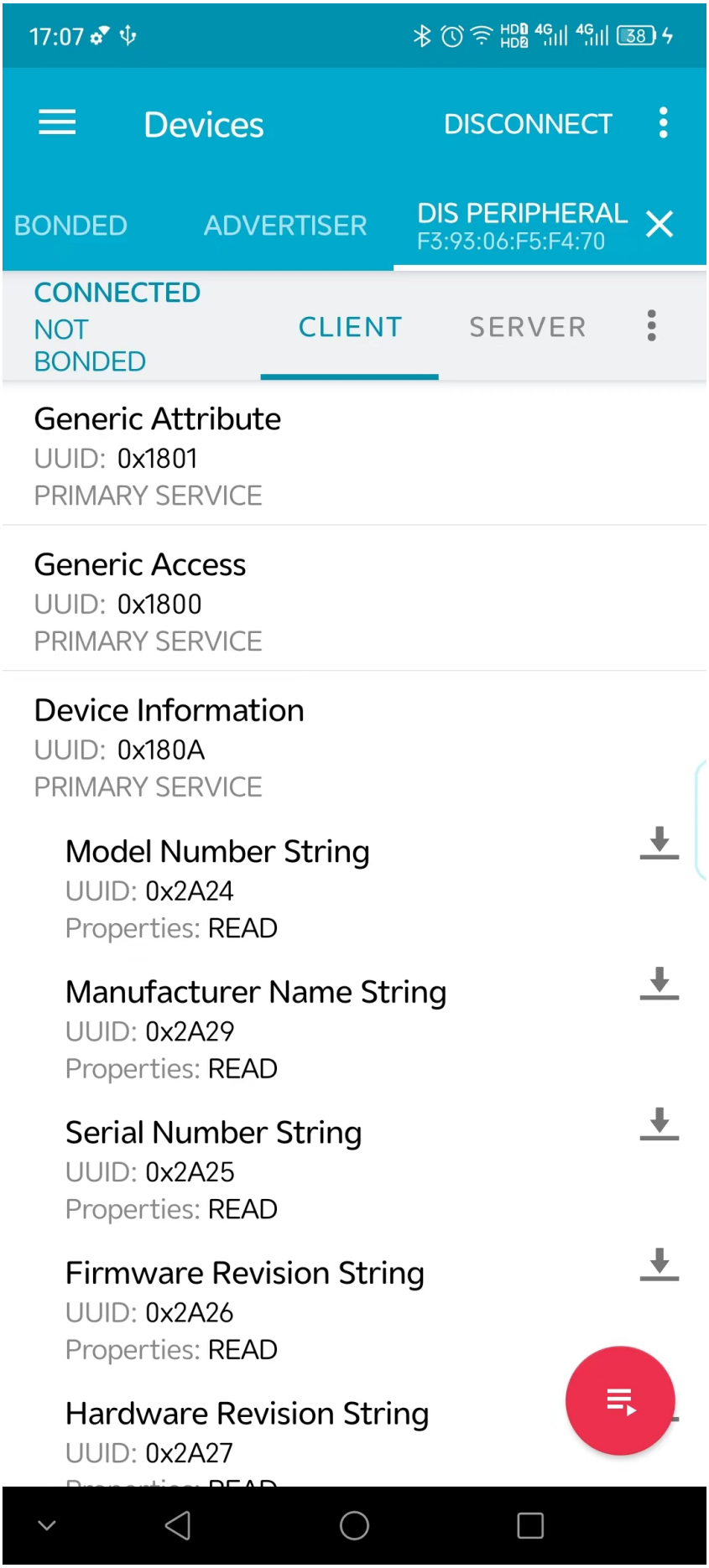
```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`，可查看源码，执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明

1. 烧录完成后，设备自动启动蓝牙广播，可以在手机或抓包工具上获取如下信息:
   - Service UUID: 0x180A
   - Device Name: DIS peripheral
2. 当手机或其它主设备与其建立连接后，串口 log 会显示连接信息，如下:

   ```
   Connected
   ```

3. 使用 nRF Connect 连接上 DIS peripheral 设备后，app 显示如下:


17:07
Devices
DISCONNECT
BONDED
ADVERTISER
DIS PERIPHERAL
F3:93:06:F5:F4:70
CONNECTED
NOT
BONDED
CLIENT
SERVER
Generic Attribute
UUID: 0x1801
PRIMARY SERVICE
Generic Access
UUID: 0x1800
PRIMARY SERVICE
Device Information
UUID: 0x180A
PRIMARY SERVICE
Model Number String
UUID: 0x2A24
Properties: READ
Manufacturer Name String
UUID: 0x2A29
Properties: READ
Serial Number String
UUID: 0x2A25
Properties: READ
Firmware Revision String
UUID: 0x2A26
Properties: READ
Hardware Revision String
UUID: 0x2A27

Bluetooth: Peripheral ESP

1 功能概述 此项目演示了蓝牙从机的功能，该工程专门演示 ESP (Environmental Sensing Profile)GATT 服务。

2 环境要求

- board: 支持 BLE 的蓝牙设备
- uart(option): 用来显示串口 log
- 测试软件: nRF Connect

3 编译和烧录 项目位置：`zephyr\samples_panchip\bluetooth\peripheral_esp`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\bluetooth\peripheral_esp.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

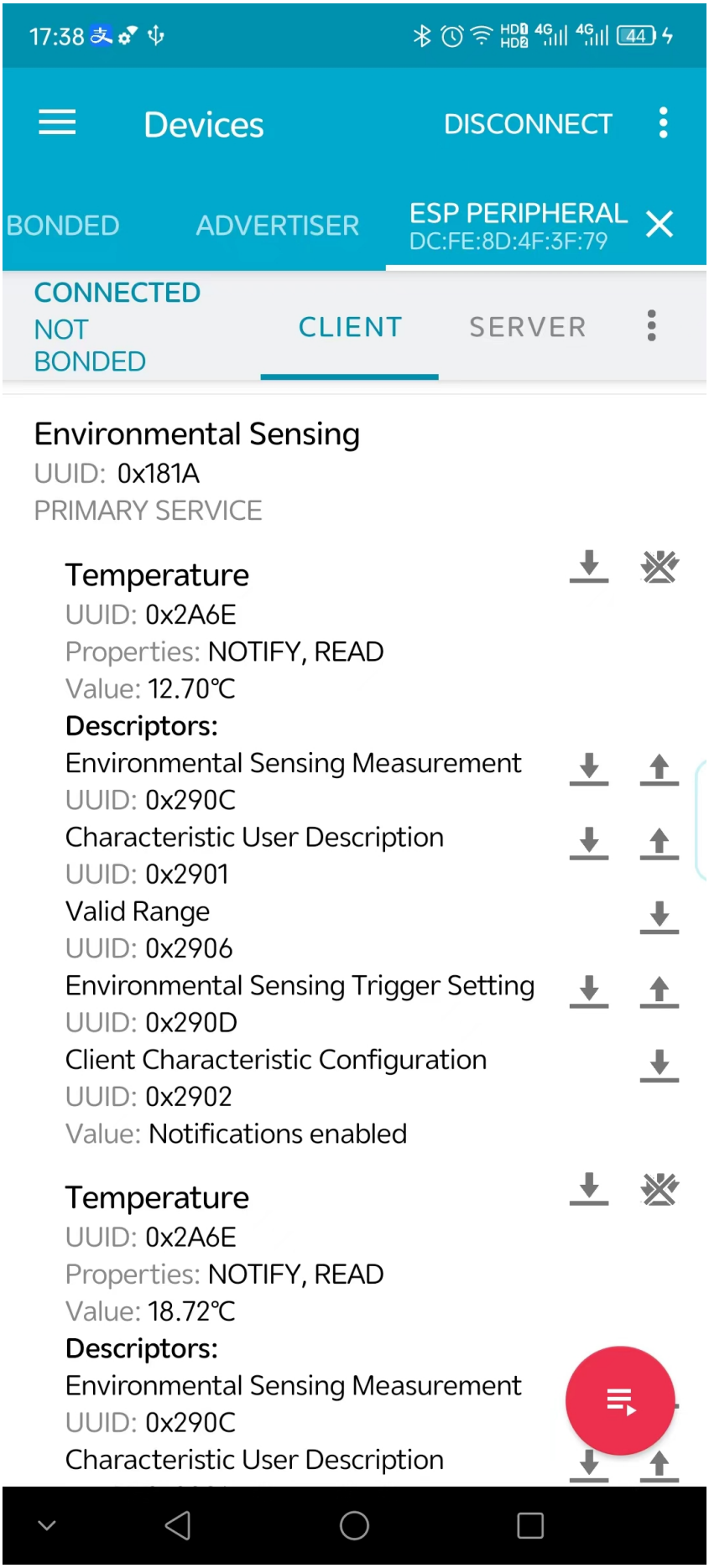
```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

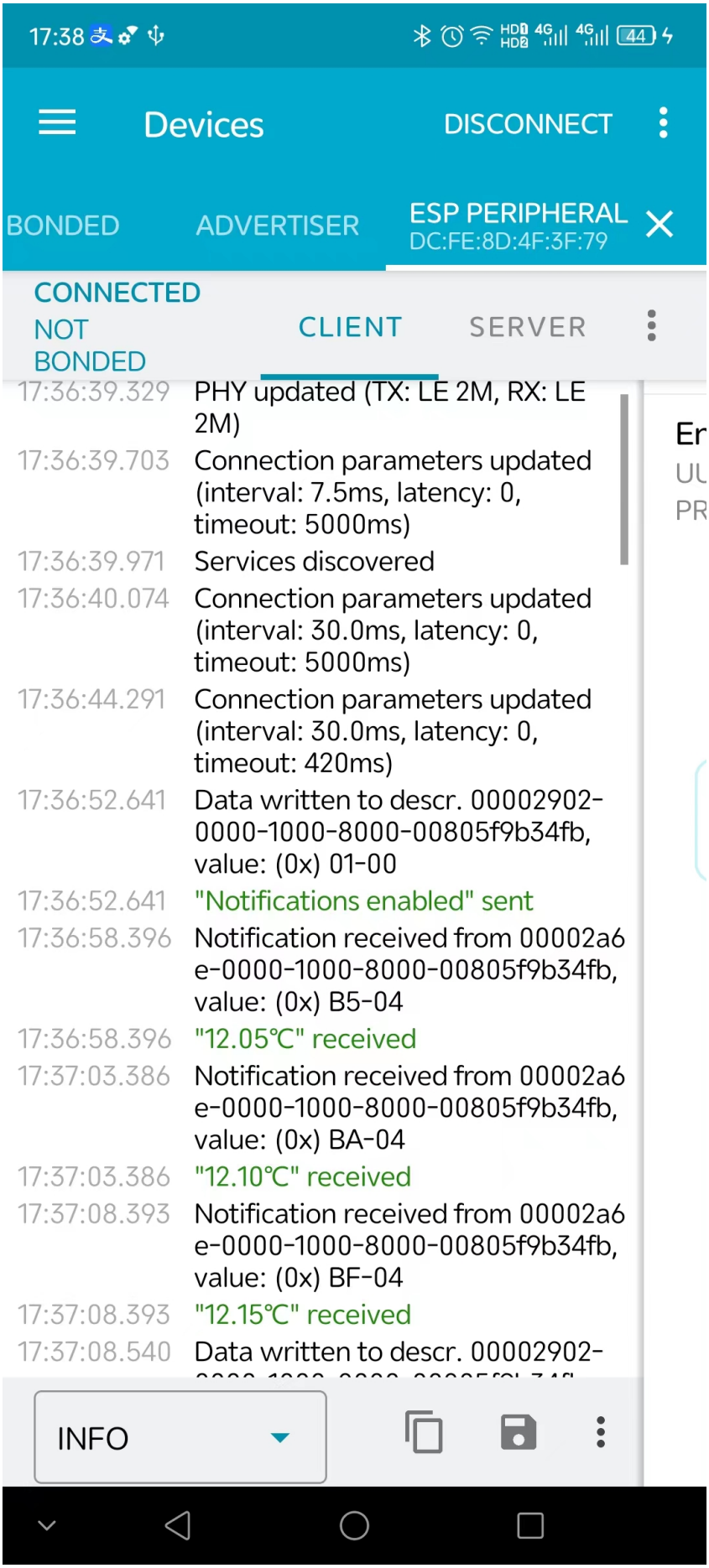
4 演示说明

1. 烧录完成后，设备自动启动蓝牙广播，可以在手机或抓包工具上获取如下信息：
   - Service UUID: 0x180A, 0x181A, 0x180F
   - Device Name: ESP PERIPHERAL
2. 当手机或其它主设备与其建立连接后，串口 log 会显示连接信息，如下：

   ```
   Connected
   ```

3. 此时，如果使能了 Environmental Sensing Measurement，主机将收到温度数据，如下：


17:38
Devices
DISCONNECT
BONDED
ADVERTISER
ESP PERIPHERAL
DC:FE:8D:4F:3F:79
CONNECTED
NOT
BONDED
CLIENT
SERVER
Environmental Sensing
UUID: 0x181A
PRIMARY SERVICE
Temperature
UUID: 0x2A6E
Properties: NOTIFY, READ
Value: 12.70℃
Descriptors:
Environmental Sensing Measurement
UUID: 0x290C
Characteristic User Description
UUID: 0x2901
Valid Range
UUID: 0x2906
Environmental Sensing Trigger Setting
UUID: 0x290D
Client Characteristic Configuration
UUID: 0x2902
Value: Notifications enabled
Temperature
UUID: 0x2A6E
Properties: NOTIFY, READ
Value: 18.72℃
Descriptors:
Environmental Sensing Measurement
UUID: 0x290C
Characteristic User Description


17:38
Devices
DISCONNECT
BONDED
ADVERTISER
ESP PERIPHERAL
DC:FE:8D:4F:3F:79
CONNECTED
NOT BONDED
CLIENT
SERVER
17:36:39.329 PHY updated (TX: LE 2M, RX: LE 2M)
17:36:39.703 Connection parameters updated (interval: 7.5ms, latency: 0, timeout: 5000ms)
17:36:39.971 Services discovered
17:36:40.074 Connection parameters updated (interval: 30.0ms, latency: 0, timeout: 5000ms)
17:36:44.291 Connection parameters updated (interval: 30.0ms, latency: 0, timeout: 420ms)
17:36:52.641 Data written to descr. 00002902-0000-1000-8000-00805f9b34fb, value: (0x) 01-00
17:36:52.641 "Notifications enabled" sent
17:36:58.396 Notification received from 00002a6e-0000-1000-8000-00805f9b34fb, value: (0x) B5-04
17:36:58.396 "12.05℃" received
17:37:03.386 Notification received from 00002a6e-0000-1000-8000-00805f9b34fb, value: (0x) BA-04
17:37:03.386 "12.10℃" received
17:37:08.393 Notification received from 00002a6e-0000-1000-8000-00805f9b34fb, value: (0x) BF-04
17:37:08.393 "12.15℃" received
17:37:08.540 Data written to descr. 00002902-
INFO

Bluetooth: Peripheral HIDs

1 功能概述 此项目演示了蓝牙从机的功能，该工程演示了 HID over GATT 服务，模拟一个通用鼠标设备。

2 环境要求

- board: 支持蓝牙的设备
- uart: 用来显示配对码，日志等
- 测试设备: 手机、平板等

3 编译和烧录 项目位置：zephyr\samples_panchip\bluetooth\peripheral_hids。

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：quick_build_samples\bluetooth\peripheral_hids.bat。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

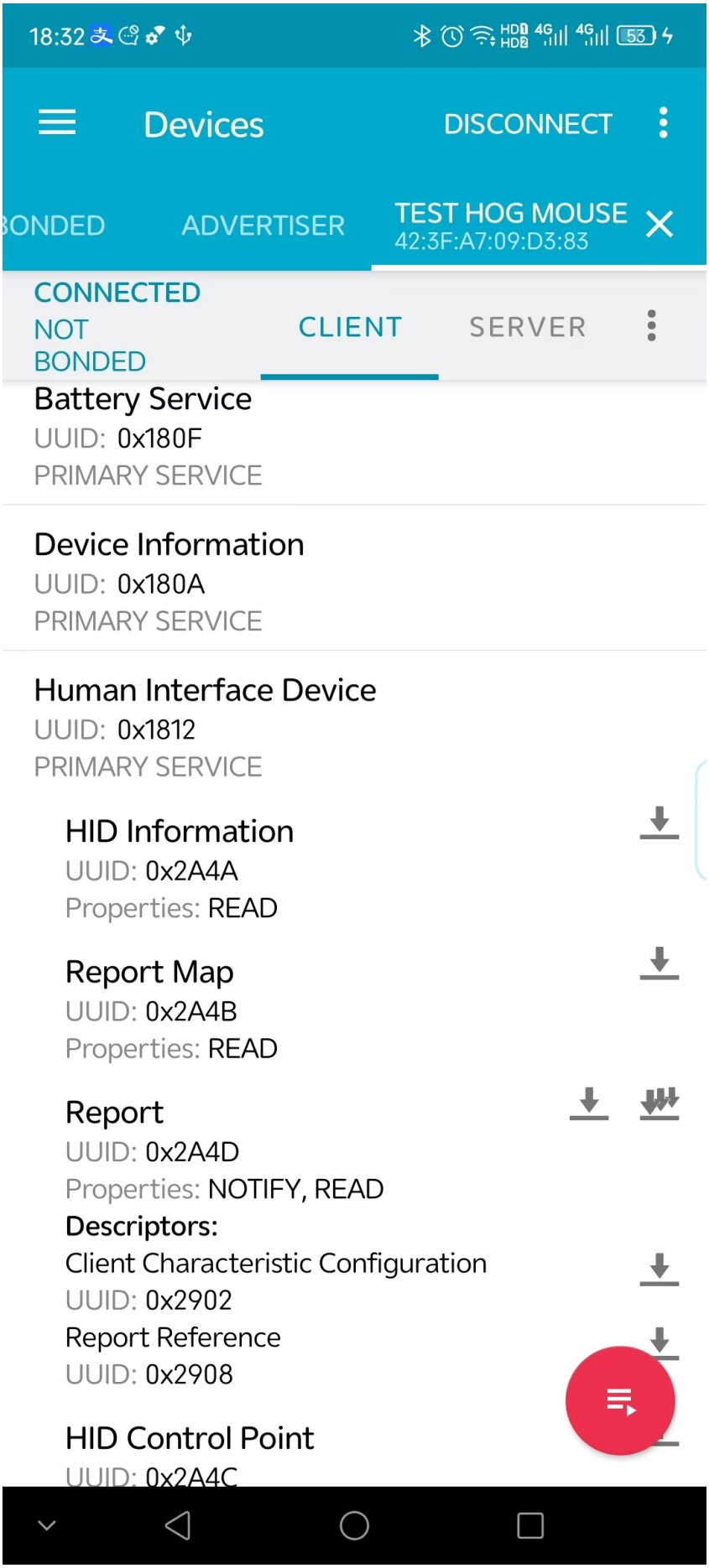
```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明

1. 烧录完成后，设备自动启动蓝牙广播，可以在手机或抓包工具上获取如下信息：
   - Service UUID: 0x1812, 0x180A, 0x180F
   - Device Name: Test HoG mouse
2. 当手机或其它主设备与其建立连接后，串口 log 会显示连接信息，如下：

   ```
   Connected
   ```

3. 该示例主要演示 HID over GATT 服务：


18:32
Devices
DISCONNECT
ONDED
ADVERTISER
TEST HOG MOUSE
42:3F:A7:09:D3:83
CONNECTED
NOT
BONDED
CLIENT
SERVER
Battery Service
UUID: 0x180F
PRIMARY SERVICE
Device Information
UUID: 0x180A
PRIMARY SERVICE
Human Interface Device
UUID: 0x1812
PRIMARY SERVICE
HID Information
UUID: 0x2A4A
Properties: READ
Report Map
UUID: 0x2A4B
Properties: READ
Report
UUID: 0x2A4D
Properties: NOTIFY, READ
Descriptors:
Client Characteristic Configuration
UUID: 0x2902
Report Reference
UUID: 0x2908
HID Control Point
UUID: 0x2A4C

5 其他说明　修改 CONFIG_BT_MAX_PAIRED 可以调整保存的配对信息个数，如下修改将支持保存两组配对信息。

```
CONFIG_BT_MAX_PAIRED=2
```

当需要与多个设备配对时，可以修改 CONFIG_BT_KEYS_OVERWRITE_OLDEST 来支持覆盖旧设备的配对信息。

否则配对信息存储达到上限时，将无法与新设备进行配对。

```
CONFIG_BT_KEYS_OVERWRITE_OLDEST=y
```

Bluetooth: Peripheral / Heart-rate Monitor

1 功能概述　此项目演示了蓝牙从机的功能，该工程专门暴露了 HR (Heart Rate)GATT 服务。

一旦设备连接上，它就会产生虚拟的心率值。

2 环境要求

- board: 支持 BLE 的蓝牙设备
- uart(option): 用来显示串口 log
- 测试软件: nRF Connect

3 编译和烧录　项目位置：zephyr\samples_panchip\bluetooth\peripheral_hr

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：quick_build_samples\bluetooth\peripheral_hr.bat。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

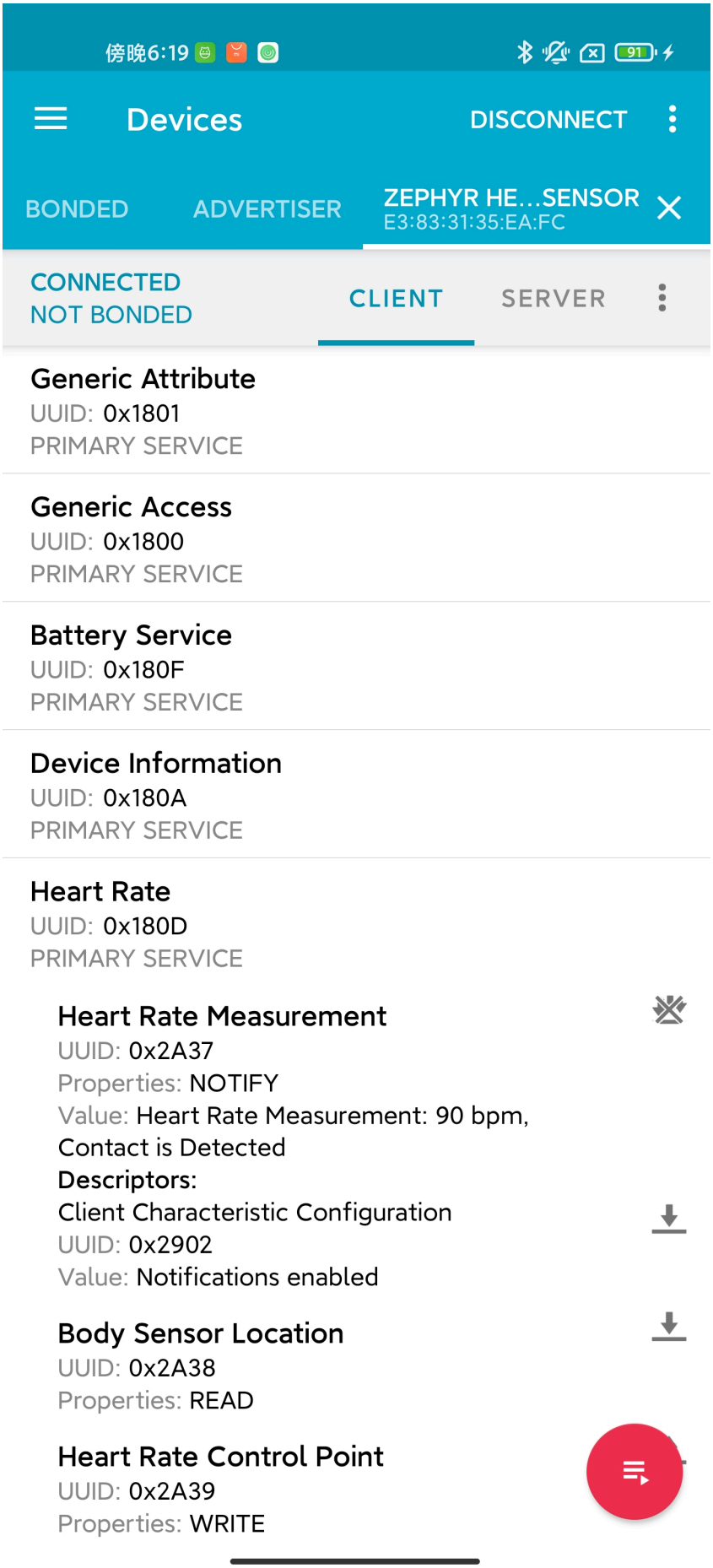
```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

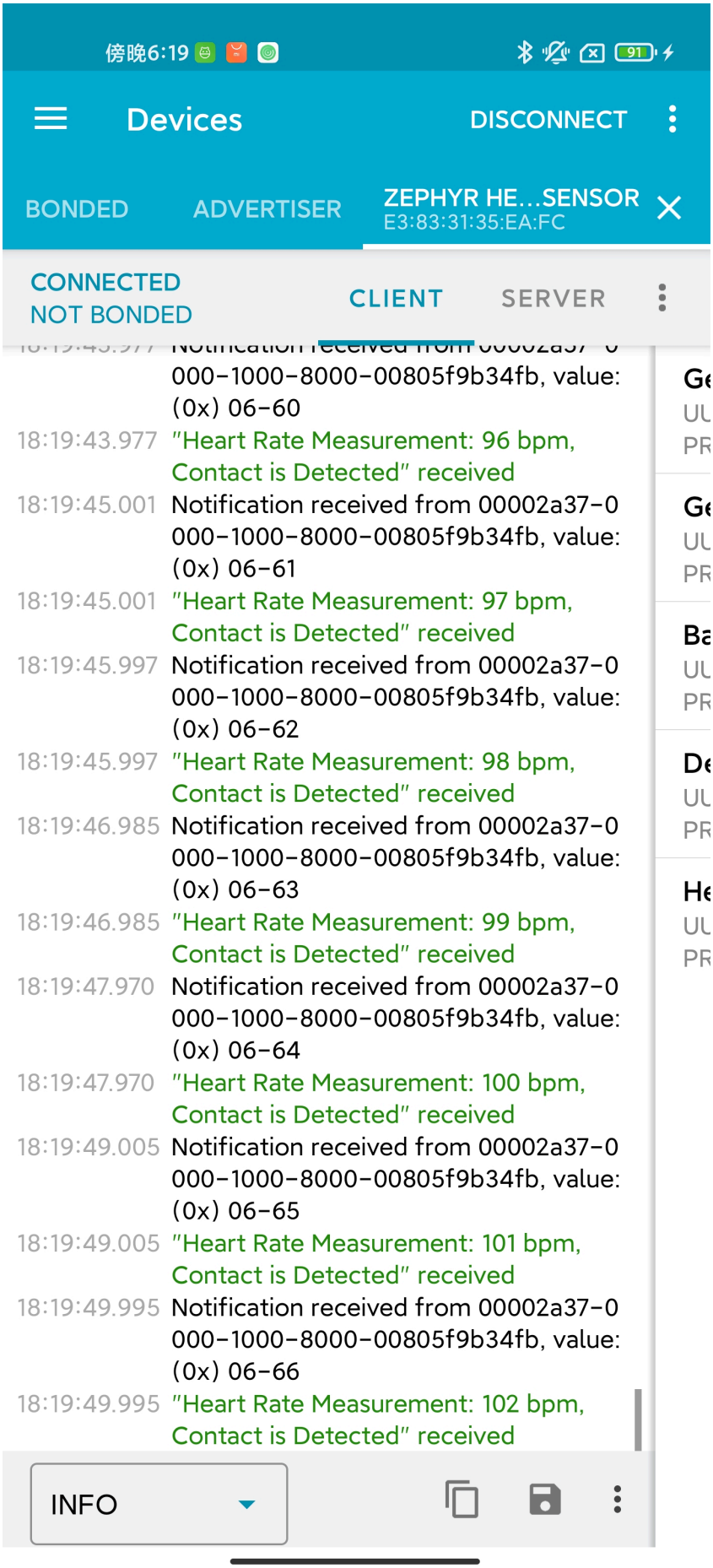
4 演示说明

1. 烧录完成后，设备自动启动蓝牙广播，可以在手机或抓包工具上获取如下信息:
   - Service UUID: 0x180D, 0x180A, 0x180F
   - Device Name: Zephyr Heartrate Sensor
2. 当手机或其它主设备与其建立连接后，串口 log 会显示连接信息，如下:

   ```
   Connected
   ```

3. 此时，如果使能了 Heart Rate Measurement，主机将收到温度数据，如下:


傍晚6:19
Devices
DISCONNECT
BONDED
ADVERTISER
ZEPHYR HE...SENSOR
E3:83:31:35:EA:FC
CONNECTED
NOT BONDED
CLIENT
SERVER
Generic Attribute
UUID: 0x1801
PRIMARY SERVICE
Generic Access
UUID: 0x1800
PRIMARY SERVICE
Battery Service
UUID: 0x180F
PRIMARY SERVICE
Device Information
UUID: 0x180A
PRIMARY SERVICE
Heart Rate
UUID: 0x180D
PRIMARY SERVICE
Heart Rate Measurement
UUID: 0x2A37
Properties: NOTIFY
Value: Heart Rate Measurement: 90 bpm,
Contact is Detected
Descriptors:
Client Characteristic Configuration
UUID: 0x2902
Value: Notifications enabled
Body Sensor Location
UUID: 0x2A38
Properties: READ
Heart Rate Control Point
UUID: 0x2A39
Properties: WRITE


傍晚6:19
Devices
DISCONNECT
BONDED
ADVERTISER
ZEPHYR HE...SENSOR
E3:83:31:35:EA:FC
CONNECTED
NOT BONDED
CLIENT
SERVER
000-1000-8000-00805f9b34fb, value: (0x) 06-60
18:19:43.977 "Heart Rate Measurement: 96 bpm, Contact is Detected" received
18:19:45.001 Notification received from 00002a37-0000-1000-8000-00805f9b34fb, value: (0x) 06-61
18:19:45.001 "Heart Rate Measurement: 97 bpm, Contact is Detected" received
18:19:45.997 Notification received from 00002a37-0000-1000-8000-00805f9b34fb, value: (0x) 06-62
18:19:45.997 "Heart Rate Measurement: 98 bpm, Contact is Detected" received
18:19:46.985 Notification received from 00002a37-0000-1000-8000-00805f9b34fb, value: (0x) 06-63
18:19:46.985 "Heart Rate Measurement: 99 bpm, Contact is Detected" received
18:19:47.970 Notification received from 00002a37-0000-1000-8000-00805f9b34fb, value: (0x) 06-64
18:19:47.970 "Heart Rate Measurement: 100 bpm, Contact is Detected" received
18:19:49.005 Notification received from 00002a37-0000-1000-8000-00805f9b34fb, value: (0x) 06-65
18:19:49.005 "Heart Rate Measurement: 101 bpm, Contact is Detected" received
18:19:49.995 Notification received from 00002a37-0000-1000-8000-00805f9b34fb, value: (0x) 06-66
18:19:49.995 "Heart Rate Measurement: 102 bpm, Contact is Detected" received
INFO

5 低功耗电流测试演示说明

1. 如果需要测试最低电流的话，我们需要使能 DCDC 模式，首先打开 `samples_panchip\bluetooth\peripheral_hr` 目录下的 `prj_lp_xtl.conf` 文件，使能 `CONFIG_SOC_DCDC_PAN1080=y`，默认是处于注释状态的。
2. 打开 `samples_panchip\bluetooth\peripheral_hr\src` 中的 `main.c`，将宏 `TEST_LOW_POWER_CURRENT` 设置为 1。使能后会使得 peripheral_hr 广播时为 1S 周期间隔，同时连接上以后也会更新为 1 秒连接间隔。
3. 运行脚本 `quick_build_samples\bluetooth\peripheral_hr_low_power_xtl.bat`，然后选择下载。
4. 然后使用电流测试工具观测电流，下图为 DCDC 模式 48M 主频广播电流：

1. 在 SDK V0.3 版本中低功耗模式下，使用 RCL/XTL32000 和 XTL32768 低功耗电流优化程度暂时是不一样的：RCL/XTL32000 支持动态发射功率，但是 TX/RX 电流偏大，所以低功耗工作电流偏大；XTL32768 固定 0dBm 发射功率，同时 TX/RX 电流较小，所以低功耗电流偏小。

Bluetooth: Peripheral / Health Thermometer sensor

1 功能概述 此项目演示了蓝牙从机的功能，该工程包含了 `HT (Health Thermometer)` 服务，可以采集温度并上报给主机。

在不支持温度采集的设备上，这些温度数据是虚拟的。

2 环境要求

- board: 支持 BLE 的蓝牙设备
- uart(option): 用来显示串口 log
- 测试软件: nRF Connect

3 编译和烧录 项目位置：`zephyr\samples_panchip\bluetooth\peripheral_ht`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\bluetooth\peripheral_ht.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

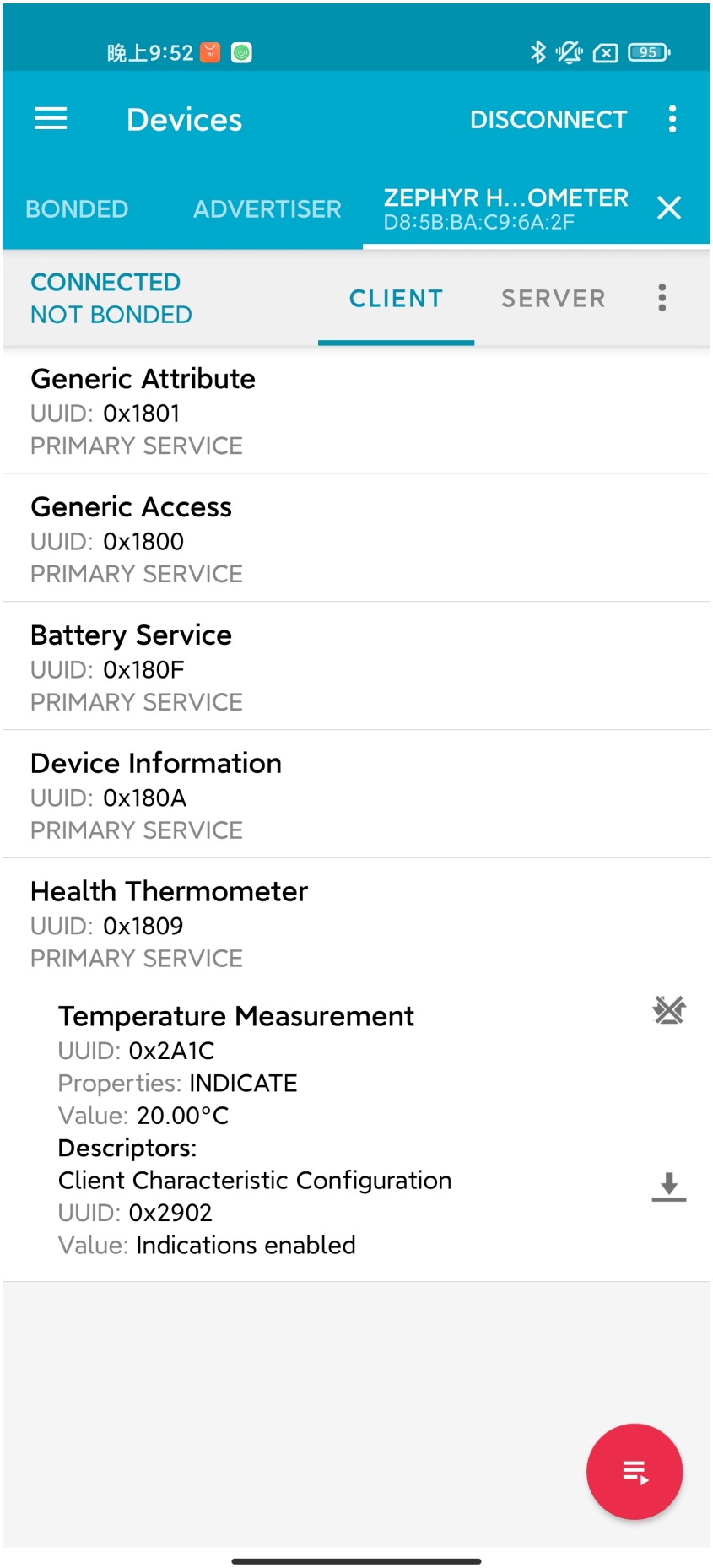
```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`，可查看源码，执行编译下载等
  others exit                        退出
wait input:
```

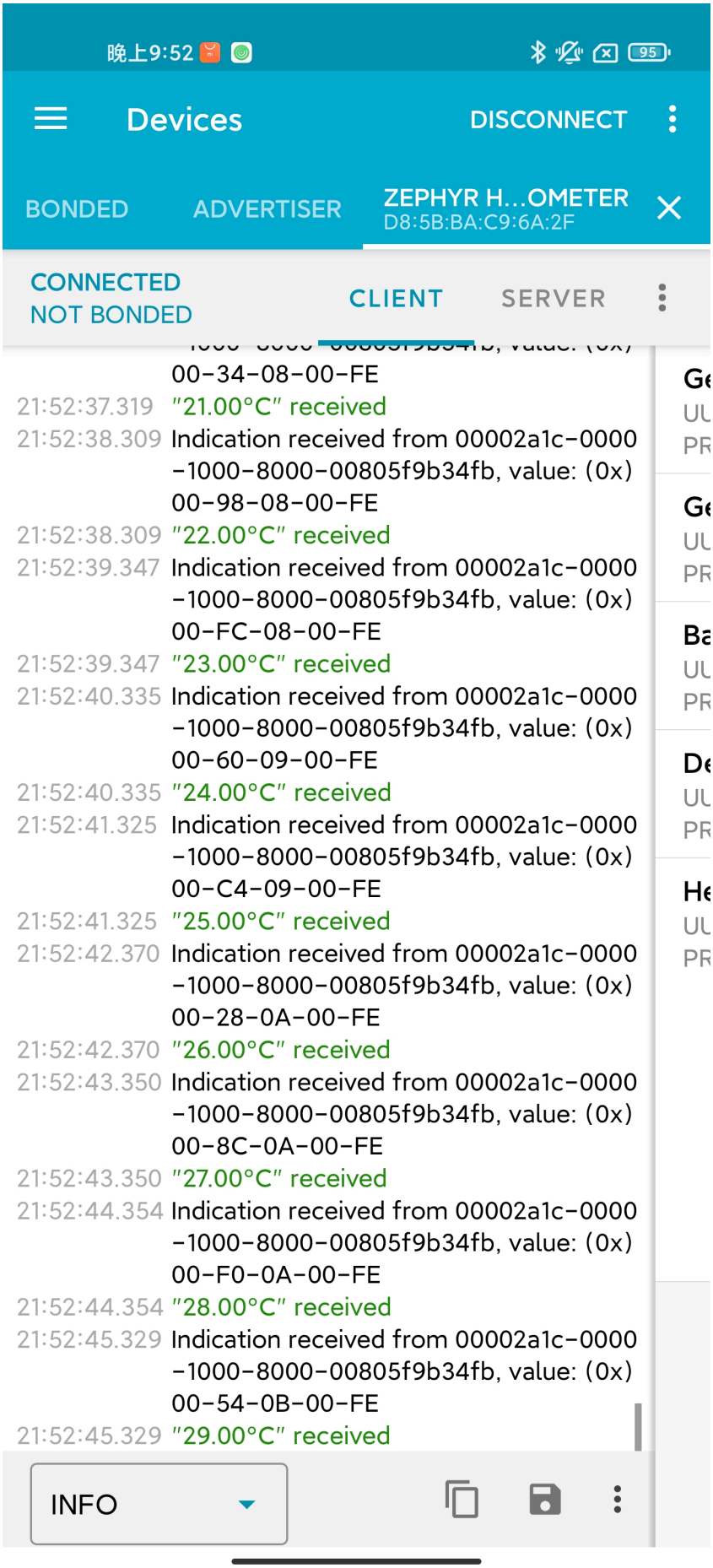
4 演示说明

1. 烧录完成后，设备自动启动蓝牙广播，可以在手机或抓包工具上获取如下信息：
   - Service UUID: 0x1809, 0x180A, 0x180F
   - Device Name: Zephyr Health Thermometer
2. 当手机或其它主设备与其建立连接后，串口 log 会显示连接信息，如下：

   ```
   Connected
   ```

3. 此时，如果使能了 `Temperature Measurement`，主机将收到温度数据，如下：


晚上9:52
Devices
DISCONNECT
BONDED
ADVERTISER
ZEPHYR H...OMETER
D8:5B:BA:C9:6A:2F
CONNECTED
NOT BONDED
CLIENT
SERVER
Generic Attribute
UUID: 0x1801
PRIMARY SERVICE
Generic Access
UUID: 0x1800
PRIMARY SERVICE
Battery Service
UUID: 0x180F
PRIMARY SERVICE
Device Information
UUID: 0x180A
PRIMARY SERVICE
Health Thermometer
UUID: 0x1809
PRIMARY SERVICE
Temperature Measurement
UUID: 0x2A1C
Properties: INDICATE
Value: 20.00°C
Descriptors:
Client Characteristic Configuration
UUID: 0x2902
Value: Indications enabled


晚上9:52
Devices
DISCONNECT
BONDED
ADVERTISER
ZEPHYR H...OMETER
D8:5B:BA:C9:6A:2F
CONNECTED
NOT BONDED
CLIENT
SERVER
00-34-08-00-FE
21:52:37.319 "21.00°C" received
21:52:38.309 Indication received from 00002a1c-0000-1000-8000-00805f9b34fb, value: (0x) 00-98-08-00-FE
21:52:38.309 "22.00°C" received
21:52:39.347 Indication received from 00002a1c-0000-1000-8000-00805f9b34fb, value: (0x) 00-FC-08-00-FE
21:52:39.347 "23.00°C" received
21:52:40.335 Indication received from 00002a1c-0000-1000-8000-00805f9b34fb, value: (0x) 00-60-09-00-FE
21:52:40.335 "24.00°C" received
21:52:41.325 Indication received from 00002a1c-0000-1000-8000-00805f9b34fb, value: (0x) 00-C4-09-00-FE
21:52:41.325 "25.00°C" received
21:52:42.370 Indication received from 00002a1c-0000-1000-8000-00805f9b34fb, value: (0x) 00-28-0A-00-FE
21:52:42.370 "26.00°C" received
21:52:43.350 Indication received from 00002a1c-0000-1000-8000-00805f9b34fb, value: (0x) 00-8C-0A-00-FE
21:52:43.350 "27.00°C" received
21:52:44.354 Indication received from 00002a1c-0000-1000-8000-00805f9b34fb, value: (0x) 00-F0-0A-00-FE
21:52:44.354 "28.00°C" received
21:52:45.329 Indication received from 00002a1c-0000-1000-8000-00805f9b34fb, value: (0x) 00-54-0B-00-FE
21:52:45.329 "29.00°C" received
INFO

串口 log 也会显示当前温度和数据传输的信息：

```
temperature is 20C
Indication success
Indication complete
temperature is 21C
Indication success
Indication complete
temperature is 22C
Indication success
Indication complete
temperature is 23C
Indication success
Indication complete
temperature is 24C
Indication success
Indication complete
temperature is 25C
Indication success
Indication complete
temperature is 26C
Indication success
Indication complete
temperature is 27C
Indication success
Indication complete
```

Bluetooth: Peripheral Identity

1 功能概述 此项目演示了蓝牙从机，与多个主机建立连接的功能。

2 环境要求

- board: 支持 BLE 的蓝牙设备
- uart(option): 用来显示串口 log
- 测试软件: nRF Connect

  验证多连接功能时，需要多个支持蓝牙的设备。

3 编译和烧录 项目位置：`zephyr\samples_panchip\bluetooth\peripheral_identity`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\bluetooth\peripheral_identity.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明

1. 烧录完成后，设备自动启动蓝牙广播，可以在手机或抓包工具上获取如下信息：

- Device Name: Zephyr Peripheral

2. 当手机或其它主设备与其建立连接后，串口 log 会显示连接信息，如下：

```
Connected (1): 52:A7:8D:29:A4:5B (random)
```

3. 并且，设备会使用新的一组地址进行广播，如下：

```
New id: 1
Using current id: 1
Advertising successfully started
```

广播数据的格式与第一阶段一致。

重复步骤 1-2，可以实现建立多个连接。

5 其它说明

5.1 配置

prj.conf 中的如下两个宏，会影响最大支持的连接个数。

```
CONFIG_BT_MAX_CONN=8
CONFIG_BT_ID_MAX=8
```

Bluetooth: Peripheral OTA

1 功能概述 此项目演示了蓝牙从机 OTA 的功能，该工程在” peripheral” 从机例程的基础上增加了 OTA 的 SMP 服务，可以通过 “nrf connect” APP 对设备进行软件升级。

2 环境要求

- board: 支持 BLE 的蓝牙设备
- uart(option): 用来显示串口 log
- 测试软件: nRF Connect V4.24.3
- 烧录工具：j-link 或 panlink

3 编译和烧录 项目位置：zephyr\samples_panchip\bluetooth\peripheral_ota

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：quick_build_samples\bluetooth\peripheral_ota.bat。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

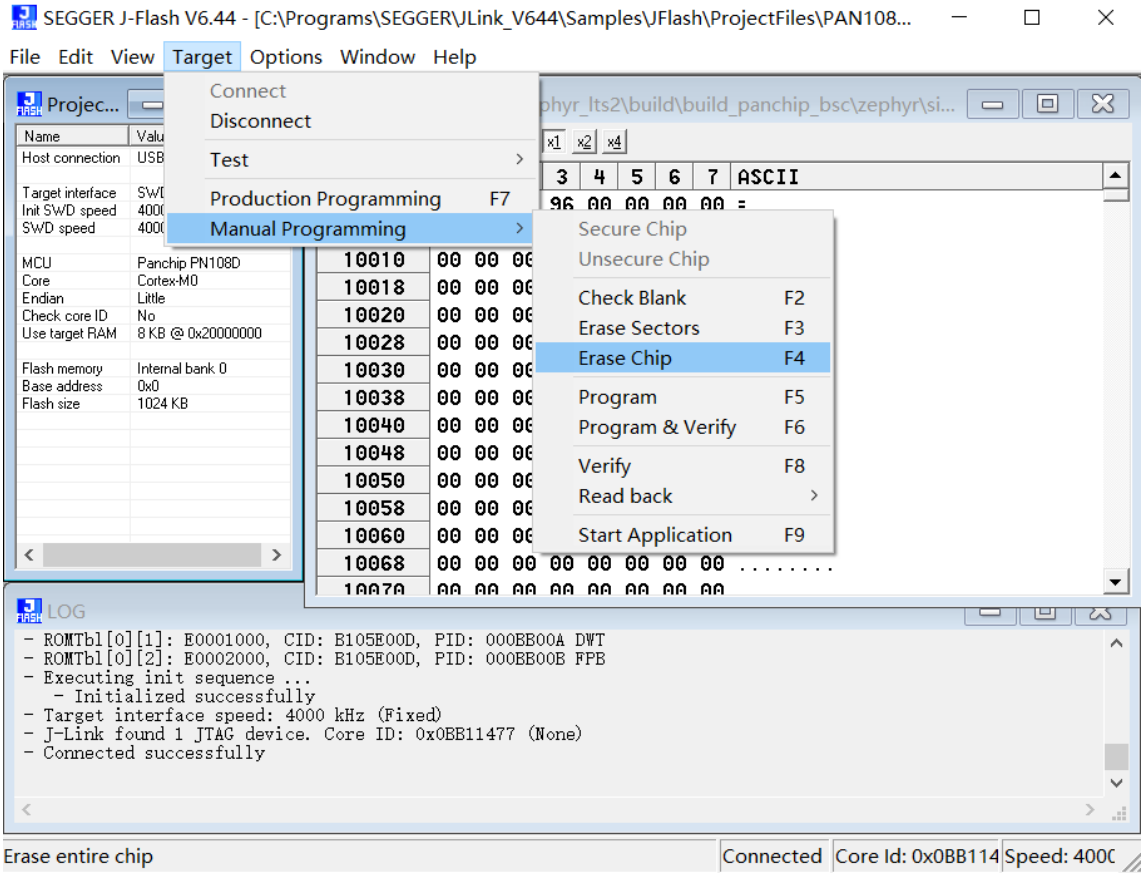
```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

烧录：

本文档使用的是 segger j-flash 烧录，烧录流程可参考 j-flash 烧录使用文档。以下流程的前提是 j-flash 中已经添加了 PAN1080 的芯片配置，而且能通过 j-link 正常连接 PAN1080 EVB 板。

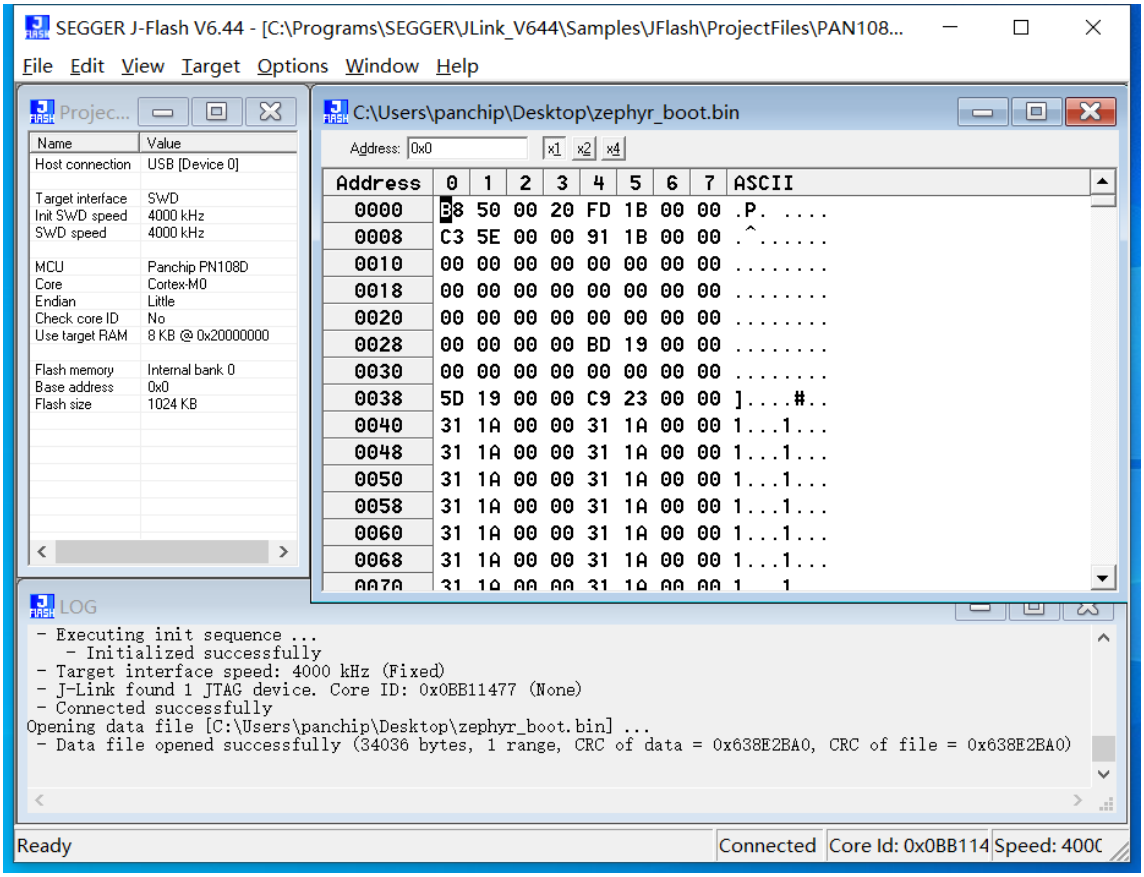
1. 擦除 flash

   这里芯片的 flash 全部擦除



2. 烧录 boot

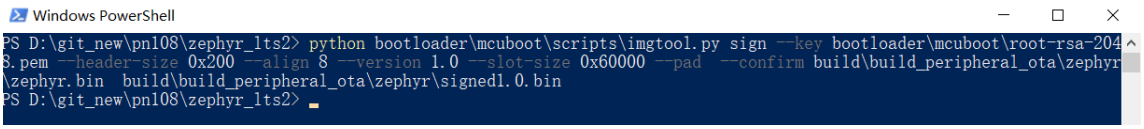
   boot 程序目录“zephyr\samples_panchip\bluetooth\peripheral_ota\boot\zephyr_boot_V1.0.bin”。boot 程序从 0x0 地址开始烧录。如下图所示：



3. 应用程序签名

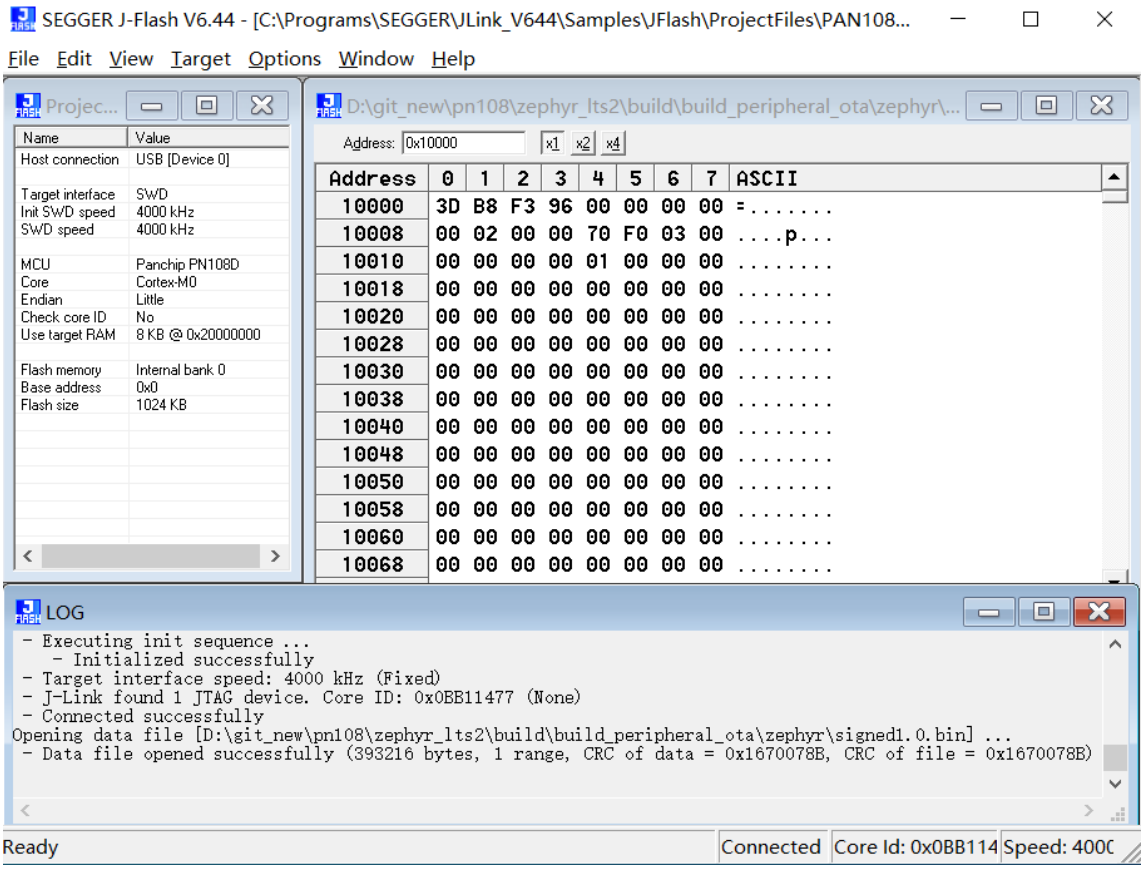
   将编译后的 zephyr.bin 文件进行签名，打开 powershell 执行如下命令

```
python bootloader\mcuboot\scripts\imgtool.py sign --key bootloader\mcuboot\root-rsa-2048.
↪pem --header-size 0x200 --align 8 --version 1.0 --slot-size 0x60000 --pad  --confirm␣
↪build\build_peripheral_ota\zephyr\zephyr.bin  build\build_peripheral_ota\zephyr\signed1.
↪0.bin
```



4. 烧录应用程序

   将签名后的应用程序 “signed1.0.bin”。应用程序从 0x10000 地址开始烧录。如下图所示：



烧录完重新上电，PAN1080 EVB 板的程序就可以正常运行起来了。

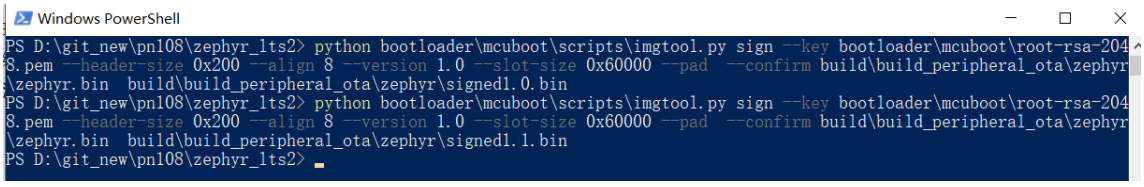
5. 制作 OTA 程序

   在“prj.cof”文件中将蓝牙设备的名字改成”Zephyr Peripheral Ota Test”，然后编译。

```
CONFIG_BT_DEVICE_NAME="Zephyr Peripheral Ota Test"
```

   将编译后的 zephyr.bin 文件进行签名，打开 powershell 执行如下命令

```
python bootloader\mcuboot\scripts\imgtool.py sign --key bootloader\mcuboot\root-rsa-2048.
↪pem --header-size 0x200 --align 8 --version 1.0 --slot-size 0x60000 --pad  --confirm␣
↪build\build_peripheral_ota\zephyr\zephyr.bin  build\build_peripheral_ota\zephyr\signed1.
↪1.bin
```

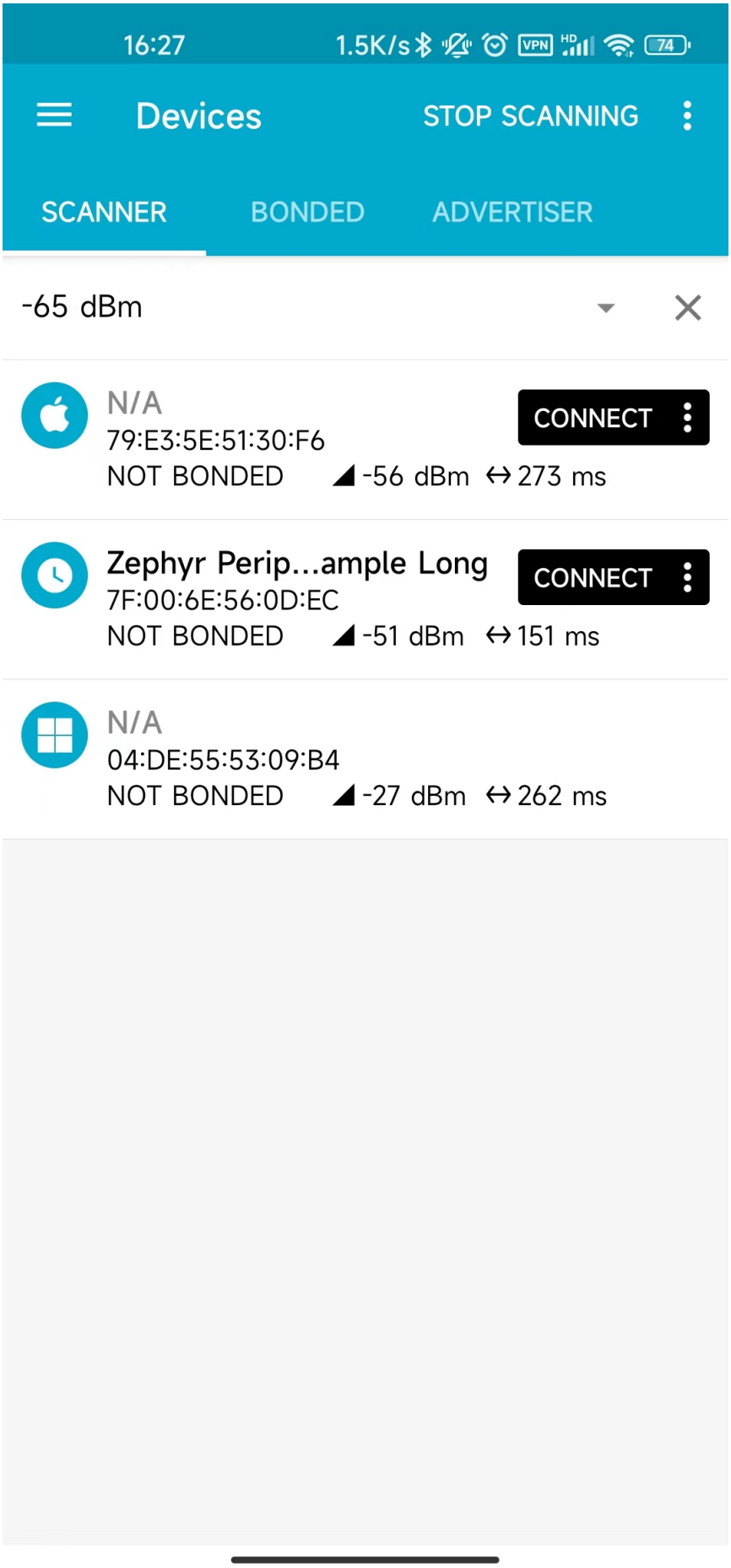


   将签名后的 OTA 程序拷贝到手机上。

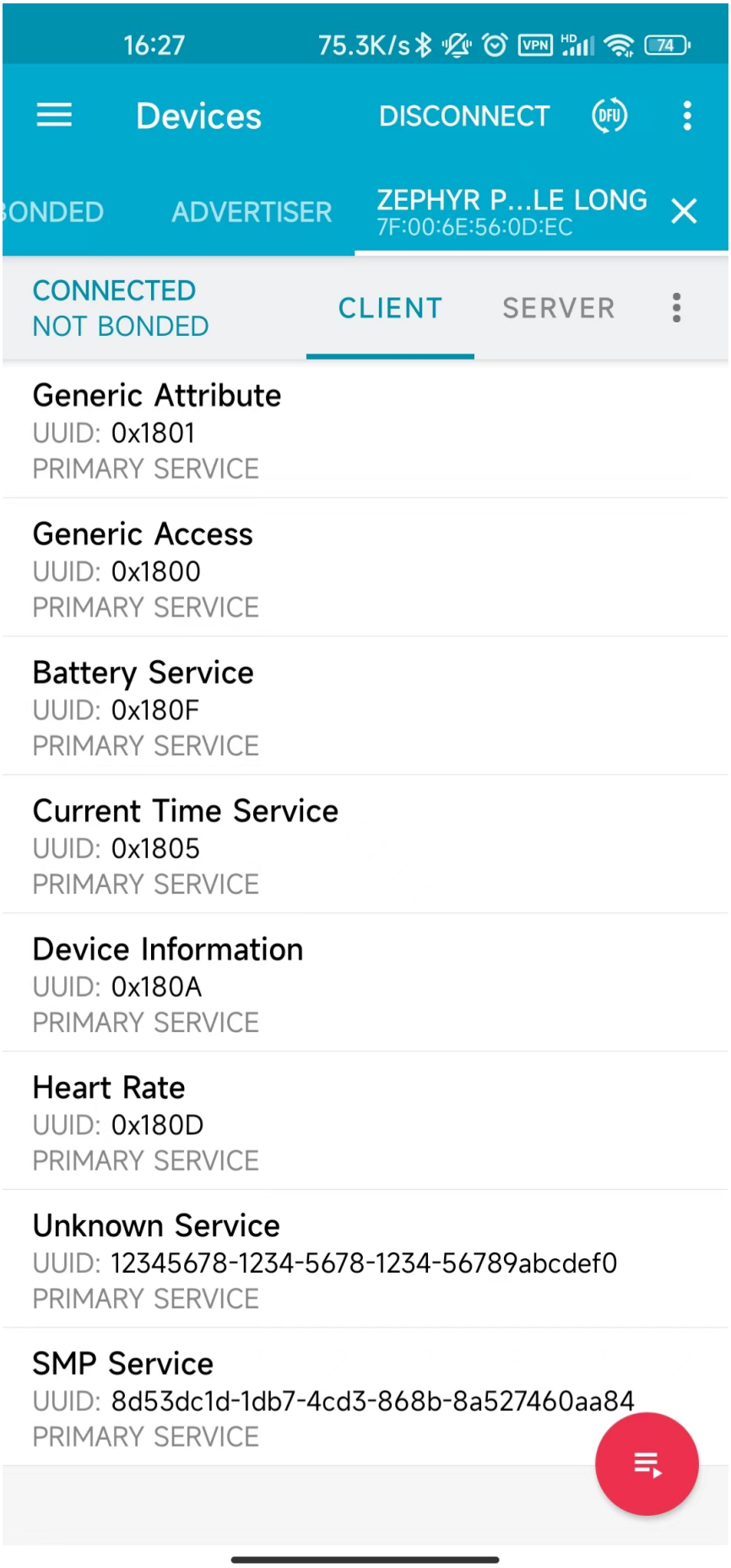
4 演示说明

1. 烧录完成后，设备自动启动蓝牙广播，可以在手机 APP nRF Connect 或抓包工具上获取如下信息：
   - Service UUID: 0x1805, 0x180D, 0x180F
   - Device Name: Zephyr Peripheral Sample Long Name

   如下图所示：


16:27
1.5K/s
Devices
STOP SCANNING
SCANNER
BONDED
ADVERTISER
-65 dBm
N/A
79:E3:5E:51:30:F6
CONNECT
NOT BONDED
-56 dBm
273 ms
Zephyr Perip...ample Long
7F:00:6E:56:0D:EC
CONNECT
NOT BONDED
-51 dBm
151 ms
N/A
04:DE:55:53:09:B4
NOT BONDED
-27 dBm
262 ms

2. 当手机 APP nRF Connect 连接成功后，APP 显示如下：


16:27
75.3K/s
Devices
DISCONNECT
DFU
ONDED
ADVERTISER
ZEPHYR P...LE LONG
7F:00:6E:56:0D:EC
CONNECTED
NOT BONDED
CLIENT
SERVER
Generic Attribute
UUID: 0x1801
PRIMARY SERVICE
Generic Access
UUID: 0x1800
PRIMARY SERVICE
Battery Service
UUID: 0x180F
PRIMARY SERVICE
Current Time Service
UUID: 0x1805
PRIMARY SERVICE
Device Information
UUID: 0x180A
PRIMARY SERVICE
Heart Rate
UUID: 0x180D
PRIMARY SERVICE
Unknown Service
UUID: 12345678-1234-5678-1234-56789abcdef0
PRIMARY SERVICE
SMP Service
UUID: 8d53dc1d-1db7-4cd3-868b-8a527460aa84
PRIMARY SERVICE

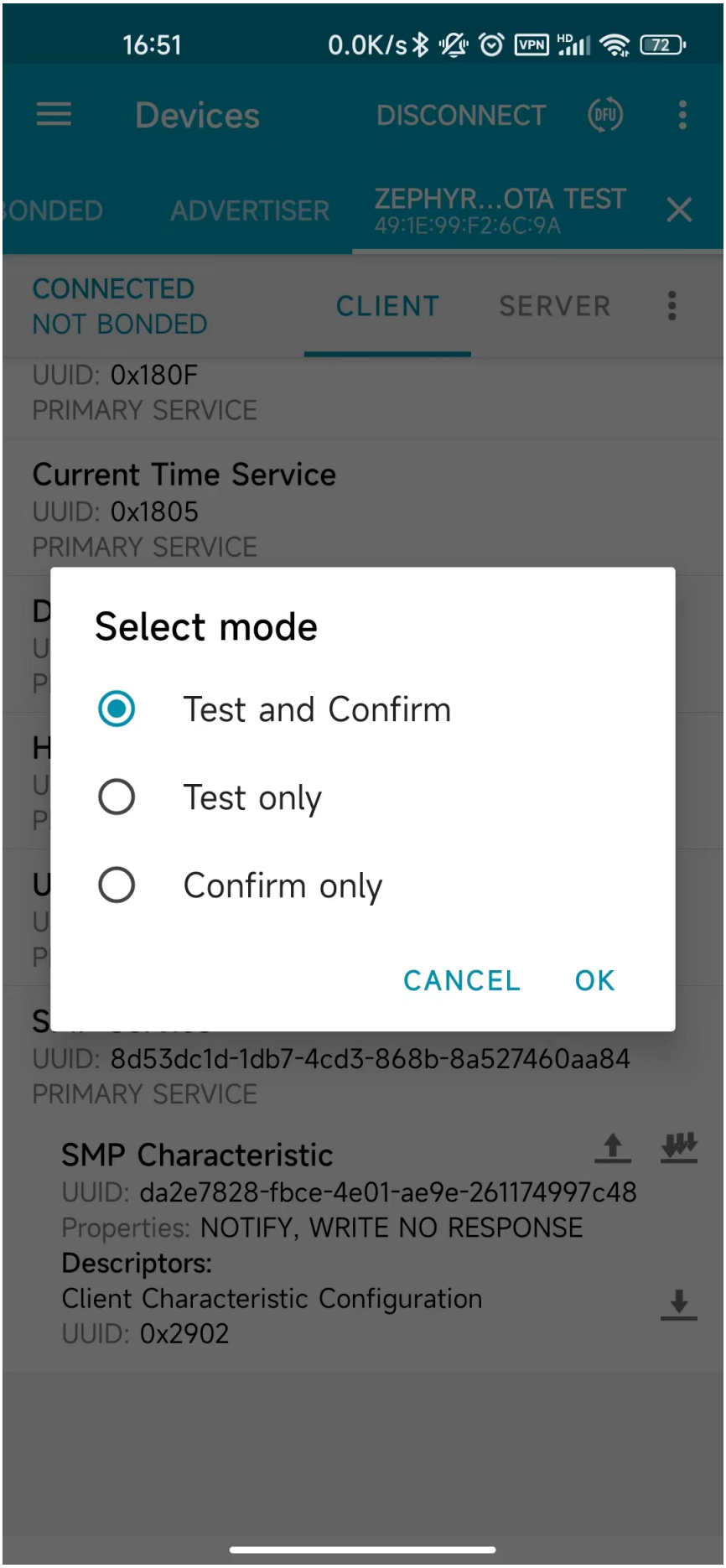
SMP 服务就是 OTA 的服务。

3. 点击手机右上角的“DFU”按钮，进入 OTA 文件选择界面，如下图所示：


16:29
2.8K/s
WeiXin
下载内容
WeiXin
音频
文档
大型文件
本周
“下载”中的文件
signed1.1.bin
393 kB 16:29

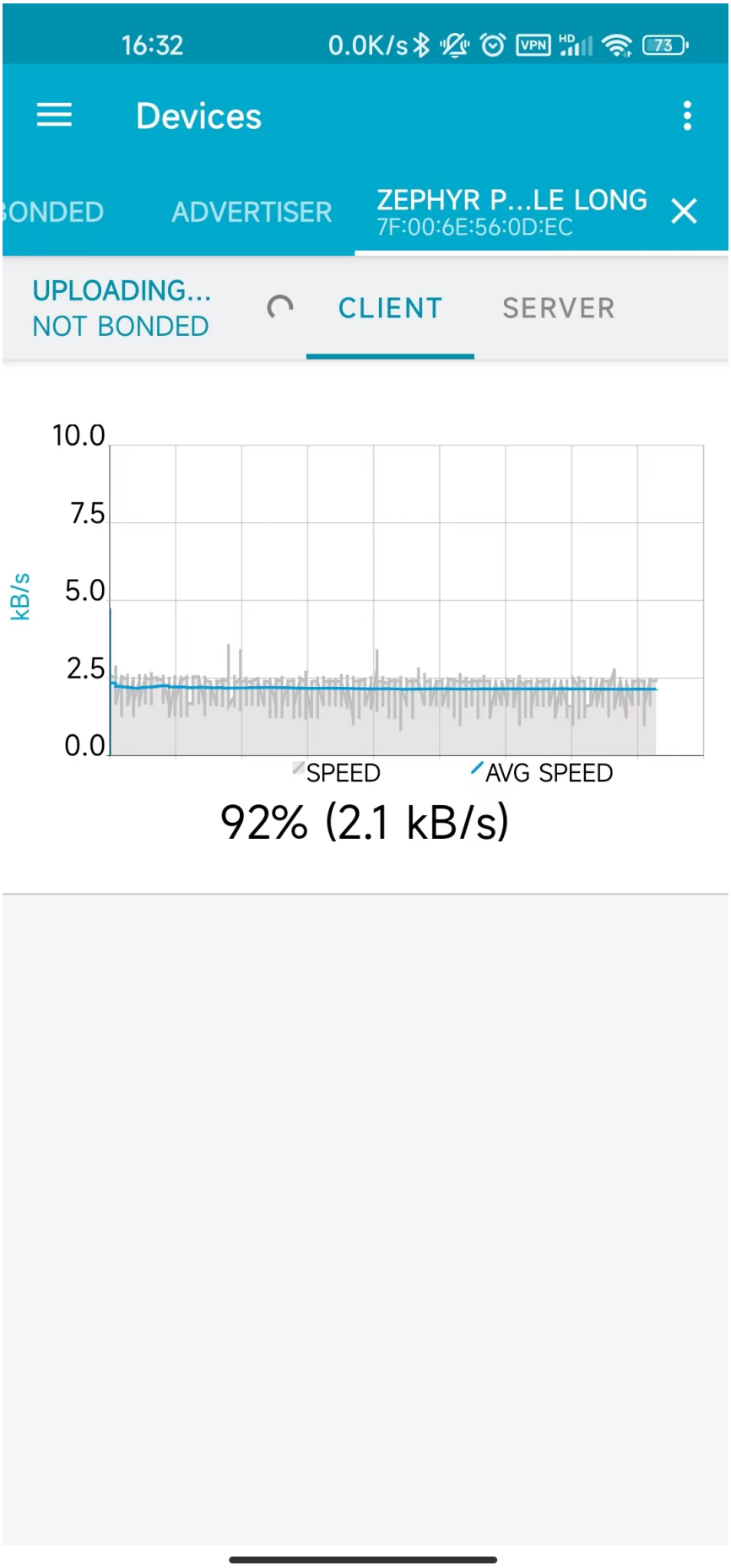
选择刚刚生成的 OTA 文件 “signed1.1.bin”。

选择 OTA 的模式:


16:51
0.0K/s
Devices
DISCONNECT
ONDED
ADVERTISER
ZEPHYR...OTA TEST
49:1E:99:F2:6C:9A
CONNECTED
NOT BONDED
CLIENT
SERVER
UUID: 0x180F
PRIMARY SERVICE
Current Time Service
UUID: 0x1805
PRIMARY SERVICE
Select mode
Test and Confirm
Test only
Confirm only
CANCEL
OK
UUID: 8d53dc1d-1db7-4cd3-868b-8a527460aa84
PRIMARY SERVICE
SMP Characteristic
UUID: da2e7828-fbce-4e01-ae9e-261174997c48
Properties: NOTIFY, WRITE NO RESPONSE
Descriptors:
Client Characteristic Configuration
UUID: 0x2902

4. 开始 OTA

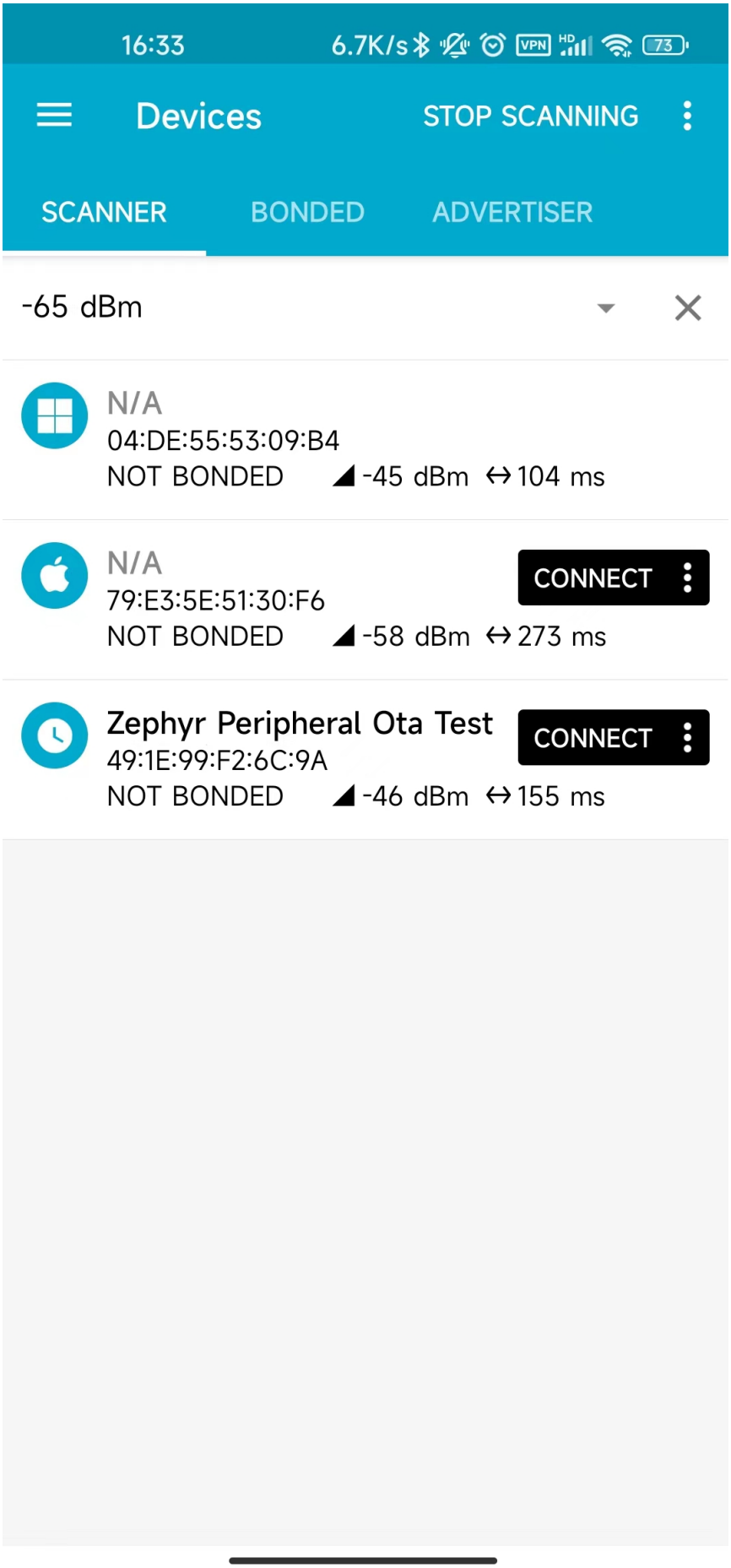
   选好文件后就开始 OTA 了，OTA 支持断点续传功能。


16:32
0.0K/s
Devices
ONDED
ADVERTISER
ZEPHYR P...LE LONG
7F:00:6E:56:0D:EC
UPLOADING...
NOT BONDED
CLIENT
SERVER
10.0
7.5
5.0
2.5
0.0
kB/s
SPEED
AVG SPEED
92% (2.1 kB/s)

5. OTA 结束

   OTA 文件传输完成后，芯片自动复位重启。使用手机 APP nRF Connect 扫描设备的广播信息，此时设备的名字已经改变。

   如下图所示：


16:33
6.7K/s
Devices
STOP SCANNING
SCANNER
BONDED
ADVERTISER
-65 dBm
N/A
04:DE:55:53:09:B4
NOT BONDED
-45 dBm
104 ms
N/A
CONNECT
79:E3:5E:51:30:F6
NOT BONDED
-58 dBm
273 ms
Zephyr Peripheral Ota Test
CONNECT
49:1E:99:F2:6C:9A
NOT BONDED
-46 dBm
155 ms

OTA 升级固件成功。

5 OTA **功能移植** 如果其他工程想要使用 OTA 功能，需要执行以下三个步骤：

1. 在 “prj.conf” 文件中添加 CONFIG 配置

```
#######################  OTA CONFIG  ############################
# Allow for large Bluetooth data packets.
CONFIG_BT_L2CAP_TX_MTU=252
CONFIG_BT_BUF_ACL_RX_SIZE=256

# Enable the Bluetooth (unauthenticated) and shell mcumgr transports.
CONFIG_MCUMGR_SMP_BT=y
CONFIG_MCUMGR_SMP_BT_AUTHEN=n

# Enable the LittleFS file system.
CONFIG_FILE_SYSTEM=y
CONFIG_FILE_SYSTEM_LITTLEFS=y

# Add 256 bytes to accommodate upload command (lfs_stat overflows)
CONFIG_SYSTEM_WORKQUEUE_STACK_SIZE=2304

# Enable mcumgr.
CONFIG_MCUMGR=y

# Ensure an MCUboot-compatible binary is generated.
CONFIG_BOOTLOADER_MCUBOOT=y

# Required by the `taskstat` command.
CONFIG_THREAD_MONITOR=y

# Enable statistics and statistic names.
CONFIG_STATS=y
CONFIG_STATS_NAMES=y

# Enable most core commands.
CONFIG_MCUMGR_CMD_IMG_MGMT=y
CONFIG_MCUMGR_CMD_OS_MGMT=y
CONFIG_MCUMGR_CMD_STAT_MGMT=y
#############################################################################
```

2. 在 “main.c” 文件中包含 OTA 相关的头文件

```
/******************  OTA INCLUDE ******************/
#ifdef CONFIG_MCUMGR_CMD_OS_MGMT
#include "os_mgmt/os_mgmt.h"
#endif
#ifdef CONFIG_MCUMGR_CMD_IMG_MGMT
#include "img_mgmt/img_mgmt.h"
#endif
#ifdef CONFIG_MCUMGR_CMD_STAT_MGMT
#include "stat_mgmt/stat_mgmt.h"
#endif
#ifdef CONFIG_MCUMGR_SMP_BT
#include <mgmt/mcumgr/smp_bt.h>
#endif
/**************************************************/
```

3. 在 “main()” 函数中添加 OTA 相关初始化代码

```
        err = bt_enable(NULL);
        if (err) {
```

(下页继续)

(续上页)

```
                printk("Bluetooth init failed (err %d)\n", err);
                return;
        }

        bt_ready();

/******************  OTA SERVICE  ******************/
#ifdef CONFIG_MCUMGR_CMD_OS_MGMT
        os_mgmt_register_group();
#endif
#ifdef CONFIG_MCUMGR_CMD_IMG_MGMT
        img_mgmt_register_group();
#endif
#ifdef CONFIG_MCUMGR_CMD_STAT_MGMT
        stat_mgmt_register_group();
#endif
#ifdef CONFIG_MCUMGR_SMP_BT
        smp_bt_register();
#endif
/***************************************************/
```

在新的工程添加以上三部分代码后，新的工程也支持 OTA 功能了。

Bluetooth: Scan & Advertise

1 功能概述 此项目演示了蓝牙广播和扫描功能。

2 环境要求

- board: 支持 BLE 的蓝牙设备
- uart(option): 用来显示串口 log
- 测试软件 (option):: nRF Connect

3 编译和烧录 项目位置：`zephyr\samples_panchip\bluetooth\scan_adv`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\bluetooth\scan_adv.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明 烧录完成后，设备会定期启动蓝牙广播和扫描：

- 广播数据中包含厂商自定义字段，该字段包含一个字节的数据，用于显示扫描态收到的数据包个数 (0 到 255 循环)。

3.2.3 外设驱动例程

Driver: ACC

1 功能概述 外设模块 ACC 模块主要用于计算两个 32 位无符号数相乘，并将其值累加存储起来，用于加速向量点积操作，同时也可当作硬件除法器。该 sample 演示了外设模块 ACC 的点乘运算和除法运算。

2 环境要求

- PAN1080 EVB 一块
- Micro USB 线一条（用于供电和查看串口打印 Log）
- 硬件接线：
  - 使用 USB 线，将 PC USB 与 EVB MicroUSB（USB->UART）相连
  - 使用**杜邦线**将 EVB 上的:
    - UART1 TX 与 P06 相连
    - UART1 RX 与 P07 相连
- PC 软件: 串口调试助手（UartAssist）或终端工具（SecureCRT），波特率 921600

3 编译和烧录 项目位置：zephyr\samples_panchip\drivers\acc

目前可使用 ZAL 工具或 quick build **脚本**进行编译和下载。

脚本位置：`quick_build_samples\drivers\acc.bat`

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明

4.1 点乘运算

1. 设置了两个包含 8 个固定元素的向量。

```
static uint32_t mult_left[MULT_BUFFER_SIZE] = { /* 向量 1 元素 */
        0x1122, 0x2233, 0x3344, 0x3,
        0x4, 0x5, 0x6, 0x1234
};
static uint32_t mult_right[MULT_BUFFER_SIZE] = { /* 向量 2 元素 */
        0, 1, 2, 3, 4, 5, 6, 7
};
```

2. 启动 ACC 乘法运算，并获取结果。

```
ret = acc_start_multiplier(acc_dev, &acc_mult_cfg); /* 启动 acc 模块点乘运算 */
if (ret != 0) {
        LOG_INF("acc start multiplier failed with code %d", ret);
}

ret = acc_get_value(acc_dev, &acc_mult_cfg); /* 获取运算结果 */
if (ret != 0) {
        LOG_INF("acc get value failed with code %d", ret);
}

for (uint8_t i = 0; i < RESULT_BUFFER_SIZE; i++) { /* 缓存运算结果 */
        result_buffer[i] = acc_mult_cfg.buffer[i];
}
```

3. 对比 ACC 模块计算值与事先已验证过的正确值，判断 ACC 模块点乘运算功能是否正常，数值一致打印 success，否则打印 “[数值 _sw] should equal to [数值 _hw] ”。

4.2 除法运算

1. 从向量中选取一组数值作为除数与被除数，启动 ACC 除法运算，并获取结果。

```
ret = acc_start_divider(acc_dev, &acc_div_cfg); /* 启动 acc 模块除法运算 */
if (ret != 0) {
        LOG_INF("acc start divider failed with code %d", ret);
}

result = acc_div_cfg.buffer[0];         /* 获取运算结果 */
```

2. 使用除法器中运算使用的除数与被除数通过软件计算方式计算结果，对比软件结果与除法器结果，数值一致打印 divider success，否则打印 “[数值 _sw] != [数值 _hw]”。

```
result_sw = acc_div_cfg.divisor / acc_div_cfg.dividend; /* 软件除法运算 */

if (result != result_sw) {         /* 对比结果 */
        LOG_INF("[%08x] != [%08x]", result, result_sw);
}
```

5 开发说明

5.1 启用 ACC 模块

在 prj.conf 文件中添加 “CONFIG_ACC=y” 启用 ACC 模块。

```
CONFIG_ACC=y
```

5.2 初始化 ACC

```
int ret;
const struct device *acc_dev;

acc_dev = device_get_binding(DT_LABEL(DT_INST(0, panchip_pan_acc))); /* 获取 acc 设备 */
if (!acc_dev) {
    LOG_INF("Cannot get ACC device\n");
}

ret = acc_set_up(acc_dev, acc_cfg);         /* 初始化 acc 模块 */
if (ret != 0) {
    LOG_INF("Setting up acc config failed with code %d", ret);
}
```

其中 acc_cfg 数据结构及参数含义如下

```
/** @brief Structure with generic accumulator features.
 *
 * @param  operate_mode operate mode select(multiplier or divider)
 * @param  first_multiplier first multiplier data
 * @param  sec_multiplier second multiplier data
 * @param  divisor divisor value
 * @param  dividend dividend value
 * @param  number_words The multiplier calculates the number of polls
 * @param  cycle: The multiplier reserves the computation period
 * @param  buffer: result buffer
 */
struct acc_config_info {
        uint8_t operate_mode;
        uint32_t *first_multiplier;
        uint32_t *sec_multiplier;
        uint32_t divisor;
        uint32_t dividend;
        uint8_t number_words;
        uint8_t calc_cycle;
        uint32_t buffer[16];
};
```

5.3 ACC API 接口

- 接口总览

```
__subsystem struct acc_driver_api {
        acc_api_set_up set_up;
        acc_api_set_int enable_int;
        acc_api_start_multiplier start_multiplier;
        acc_api_get_multiplier_value get_value;
        acc_api_start_divider start_divider;
};
```

- acc 初始化模块配置

```
/**
 * @brief  This function is used to setup acc device
 * @param  dev Pointer to the device structure for the driver instance.
 * @param  acc_cfg Pointer to acc configuration.
 * @retval 0       On success.
 * @retval -EINVAL If a parameter with an invalid value has been provided.
 */
__syscall int acc_set_up(const struct device *dev,
                         const struct acc_config_info *acc_cfg);
```

- acc 中断初始化

```
/**
 * @brief  This function is used to enable acc interrupt
 * @param  dev Pointer to the device structure for the driver instance.
 * @param  enable_state interrupt enable or not.
 * @retval 0       On success.
 * @retval -EINVAL If a parameter with an invalid value has been provided.
 */
__syscall void acc_enable_int(const struct device *dev, bool enable_state);
```

- acc 启动乘法运算

```
/**
 * @brief  This function is used to start multiplier
```

(下页继续)

(续上页)

```
 * @param  dev Pointer to the device structure for the driver instance.
 * @param  acc_cfg Pointer to acc configuration.
 * @retval 0       On success.
 * @retval -EINVAL If a parameter with an invalid value has been provided.
 */
__syscall int acc_start_multiplier(const struct device *dev,
                                   const struct acc_config_info *acc_cfg);
```

- acc 获取运算结果

```
/**
 * @brief  This function is used to get multiplier result
 * @param  dev Pointer to the device structure for the driver instance.
 * @param  acc_cfg Pointer to acc configuration.
 * @retval 0       On success.
 * @retval -EINVAL If a parameter with an invalid value has been provided.
 */
__syscall int acc_get_value(const struct device *dev,
                            const struct acc_config_info *acc_cfg);
```

- acc 启动除法运算

```
/**
 * @brief  This function is used to start divider
 * @param  dev Pointer to the device structure for the driver instance.
 * @param  acc_cfg Pointer to acc configuration.
 * @retval 0       On success.
 * @retval -EINVAL If a parameter with an invalid value has been provided.
 */
__syscall int acc_start_divider(const struct device *dev,
                                const struct acc_config_info *acc_cfg);
```

Driver: ADC

1 功能概述 该 sample 演示了外设模块 ADC 单次转换及多次转换功能。

2 环境要求

- PAN1080 EVB 一块
- Micro USB 线一条（用于供电和查看串口打印 Log）
- 硬件接线：
  - 使用 USB 线，将 PC USB 与 EVB MicroUSB（USB->UART）相连
  - 使用**杜邦线**将 EVB 上的:
    - UART1 TX 与 P06 相连
    - UART1 RX 与 P07 相连
- PC 软件: 串口调试助手（UartAssist）或终端工具（SecureCRT），波特率 921600

3 编译和烧录 项目位置：zephyr\samples_panchip\drivers\adc

目前可使用 ZAL 工具或 quick build **脚本**进行编译和下载。

脚本位置：`quick_build_samples\drivers\adc.bat`

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`，可查看源码，执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明

4.1 环境说明

- 采集通道：ADC_CHANNEL1 (P30)，像采集通道输入电压
- ADC 的换算方法（理想状态）

低压档（0~1.2V）：N=V / 2 * 4096

高压档（0~VDD）：N=V / VDD * 4096

其中 N 为 ADC 输出 Code，V 为 ADC 采样电压。

- 每 5s 执行一次例程

4.2 ADC 单次转换功能

1. 配置 adc 序列信息并 adc 初始化。

```
const struct adc_sequence sequence = {
    .channels = BIT(ADC_CHANNEL_ID),                                    /* 通道选择 */
    .buffer = m_sample_buffer,                                                  /* 结果缓存
buffer*/
    .buffer_size = sizeof(m_sample_buffer),                            /*buffer 大小，不能小于
次数 */
    .resolution = ADC_RESOLUTION,                                           /* 我们固定为
12bit adc*/
};

const struct device *adc_dev = init_adc();        /* 初始化 adc 模块 */

if (!adc_dev) {
    return -ENODEV;
}
```

2. 读取 adc 转换值。

```
ret = adc_read(adc_dev, &sequence);        /* 读取 adc 转换的 code*/
if (ret != 0) {
    printk("adc_read() failed with code %d\n", ret);
}
printk("adc sampling data: 0x%x\n", m_sample_buffer[0]); /* 打印获取 code*/
```

3. 软件通过公式计算 adc 采样电压，电压值放大 100 倍后打印

```
v_adc = ADC_REF_VOLTAGE * adc_value / 4096.0f;
printk("The voltage value after amplified by 100 times is %d\n", (uint32_t)(v_adc*100)); /
↪* 打印采样电压 */
```

4. 观测输入电压是否大致与采样电压一致

   note：采样电压是放大了 100 倍后的值，单位是 v，例如 252，那么就是 2.52v

4.3 ADC 多次转换功能

1. 配置 adc 序列信息并 adc 初始化。

```
const struct adc_sequence_options options = {
    .callback        = repeated_samplings_callback,        /* 中断回调函数 */
    .extra_samplings = ADC_BUFFER_SIZE / 2,                /* 转换次数 */
    .interval_us     = 0,                                  /* 间隔时
间 */
    .user_data       = user_data,                          /* 用户数据 */
};

const struct adc_sequence sequence = {
    .options     = &options,                               /* 增加附加序列
信息 */
    .channels = BIT(ADC_CHANNEL_ID),                       /* 通道选择 */
    .buffer = m_sample_buffer,                             /* 结果缓存
buffer*/
    .buffer_size = sizeof(m_sample_buffer),                /*buffer 大小，不能小于
次数 */
    .resolution = ADC_RESOLUTION,                          /* 我们固定为
12bit adc*/
};

const struct device *adc_dev = init_adc();        /* 初始化 adc 模块 */

if (!adc_dev) {
    return -ENODEV;
}
```

2. 设置回调函数

```
static enum adc_action repeated_samplings_callback(const struct device *dev,
                                                   const struct adc_sequence *sequence,
                                                   uint16_t sampling_index)
{
        ++m_samplings_done;
        if (m_samplings_done == 1U) {
                /* After first sampling continue normally. */
                return ADC_ACTION_CONTINUE;
        } else {
                /*
                 * The second sampling is repeated 9 times (the samples are
                 * written in the same place), then the sequence is finished
                 * prematurely.
                 */
                if (m_samplings_done < ADC_BUFFER_SIZE / 2) {
            /* 这里选择 continue，code 会依次存放在 buffer 中 */
            /* 如果选择 repeat，那么只会存放在 buffer 的固定第二个数据中 */
                        return ADC_ACTION_CONTINUE;
                } else {
                        convert_finish = true;
                        return ADC_ACTION_FINISH;
                }
        }
}
```

1. 读取 adc 转换值并计算平均值。

```
ret = adc_read(adc_dev, &sequence);        /* 读取 adc 转换的 code*/
if (ret != 0) {
    printk("adc_read() failed with code %d\n", ret);
}
```

(下页继续)

(续上页)

```
for (uint8_t i = 0; i < ADC_BUFFER_SIZE / 2; i++) {
    avg_code += m_sample_buffer[i];
    printk("adc sampling data: 0x%x\n", m_sample_buffer[i]);
}
```

2. 软件通过公式计算 adc 采样电压，电压值放大 100 倍后打印

```
v_adc = ADC_REF_VOLTAGE * adc_value / 4096.0f;
printk("The voltage value after amplified by 100 times is %d\n", (uint32_t)(v_adc*100)); /
↪* 打印采样电压 */
```

3. 观测输入电压是否大致与采样电压一致

   note：采样电压是放大了 100 倍后的值，单位是 v，例如 252，那么就是 2.52v

5 开发说明

5.1 启用 ADC 模块

在 prj.conf 文件中添加 “CONFIG_ADC=y” 启用 ADC 模块。

```
CONFIG_ADC=y
```

5.2 初始化 ADC

```
int ret;
const struct device *adc_dev = device_get_binding(DT_LABEL(DT_INST(0, panchip_pan_adc))); /* 获
取 adc 设备 */

if (!adc_dev) {
    printk("Cannot get ADC device\n");
}

ret = adc_channel_setup(adc_dev, &m_channel_cfg); /* 初始化 adc 模块 */
if (ret != 0) {
    printk("Set up ADC channel failed with code %d\n", ret);
}
```

5.3 ADC 数据结构及参数含义

- 通道配置

```
struct adc_channel_cfg {
        /** Gain selection. */
        enum adc_gain gain;

        /** Reference selection. */
        enum adc_reference reference;

        /**
         * Acquisition time.
         * Use the ADC_ACQ_TIME macro to compose the value for this field or
         * pass ADC_ACQ_TIME_DEFAULT to use the default setting for a given
         * hardware (e.g. when the hardware does not allow to configure the
         * acquisition time).
         * Particular drivers do not necessarily support all the possible units.
         * Value range is 0-16383 for a given unit.
         */
        uint16_t acquisition_time;
```

(下页继续)

(续上页)

```
        /**
         * Channel identifier.
         * This value primarily identifies the channel within the ADC API - when
         * a read request is done, the corresponding bit in the "channels" field
         * of the "adc_sequence" structure must be set to include this channel
         * in the sampling.
         * For hardware that does not allow selection of analog inputs for given
         * channels, but rather have dedicated ones, this value also selects the
         * physical ADC input to be used in the sampling. Otherwise, when it is
         * needed to explicitly select an analog input for the channel, or two
         * inputs when the channel is a differential one, the selection is done
         * in "input_positive" and "input_negative" fields.
         * Particular drivers indicate which one of the above two cases they
         * support by selecting or not a special hidden Kconfig option named
         * ADC_CONFIGURABLE_INPUTS. If this option is not selected, the macro
         * CONFIG_ADC_CONFIGURABLE_INPUTS is not defined and consequently the
         * mentioned two fields are not present in this structure.
         * While this API allows identifiers from range 0-31, particular drivers
         * may support only a limited number of channel identifiers (dependent
         * on the underlying hardware capabilities or configured via a dedicated
         * Kconfig option).
         */
        uint8_t channel_id   : 5;
        /** Channel type: single-ended or differential. */
        uint8_t differential : 1;

#ifdef CONFIG_ADC_CONFIGURABLE_INPUTS
        /**
         * Positive ADC input.
         * This is a driver dependent value that identifies an ADC input to be
         * associated with the channel.
         */
        uint8_t input_positive;

        /**
         * Negative ADC input (used only for differential channels).
         * This is a driver dependent value that identifies an ADC input to be
         * associated with the channel.
         */
        uint8_t input_negative;
#endif /* CONFIG_ADC_CONFIGURABLE_INPUTS */
};
```

- adc 采样序列参数

```
struct adc_sequence {
        /**
         * Pointer to a structure defining additional options for the sequence.
         * If NULL, the sequence consists of a single sampling.
         */
        const struct adc_sequence_options *options;

        /**
         * Bit-mask indicating the channels to be included in each sampling
         * of this sequence.
         * All selected channels must be configured with adc_channel_setup()
         * before they are used in a sequence.
         */
        uint32_t channels;
```

(下页继续)

(续上页)

```
        /**
         * Pointer to a buffer where the samples are to be written. Samples
         * from subsequent samplings are written sequentially in the buffer.
         * The number of samples written for each sampling is determined by
         * the number of channels selected in the "channels" field.
         * The buffer must be of an appropriate size, taking into account
         * the number of selected channels and the ADC resolution used,
         * as well as the number of samplings contained in the sequence.
         */
        void *buffer;

        /**
         * Specifies the actual size of the buffer pointed by the "buffer"
         * field (in bytes). The driver must ensure that samples are not
         * written beyond the limit and it must return an error if the buffer
         * turns out to be not large enough to hold all the requested samples.
         */
        size_t buffer_size;

        /**
         * ADC resolution.
         * For single-ended channels the sample values are from range:
         *   0 .. 2^resolution - 1,
         * for differential ones:
         *   - 2^(resolution-1) .. 2^(resolution-1) - 1.
         */
        uint8_t resolution;

        /**
         * Oversampling setting.
         * Each sample is averaged from 2^oversampling conversion results.
         * This feature may be unsupported by a given ADC hardware, or in
         * a specific mode (e.g. when sampling multiple channels).
         */
        uint8_t oversampling;

        /**
         * Perform calibration before the reading is taken if requested.
         *
         * The impact of channel configuration on the calibration
         * process is specific to the underlying hardware.  ADC
         * implementations that do not support calibration should
         * ignore this flag.
         */
        bool calibrate;
};
```

- adc 采样附加选项参数

```
struct adc_sequence_options {
        /**
         * Interval between consecutive samplings (in microseconds), 0 means
         * sample as fast as possible, without involving any timer.
         * The accuracy of this interval is dependent on the implementation of
         * a given driver. The default routine that handles the intervals uses
         * a kernel timer for this purpose, thus, it has the accuracy of the
         * kernel's system clock. Particular drivers may use some dedicated
         * hardware timers and achieve a better precision.
         */
        uint32_t interval_us;
```

(下页继续)

(续上页)

```
        /**
         * Callback function to be called after each sampling is done.
         * Optional - set to NULL if it is not needed.
         */
        adc_sequence_callback callback;

        /**
         * Pointer to user data. It can be used to associate the sequence
         * with any other data that is needed in the callback function.
         */
        void *user_data;

        /**
         * Number of extra samplings to perform (the total number of samplings
         * is 1 + extra_samplings).
         */
        uint16_t extra_samplings;
};
```

5.4 ADC API 接口

- 接口总览

```
__subsystem struct adc_driver_api {
        adc_api_channel_setup channel_setup;
        adc_api_read          read;
#ifdef CONFIG_ADC_ASYNC
        adc_api_read_async    read_async;
#endif
        uint16_t ref_internal;        /* mV */
};
```

- adc 初始化模块配置

```
/**
 * @brief Configure an ADC channel.
 *
 * It is required to call this function and configure each channel before it is
 * selected for a read request.
 *
 * @param dev          Pointer to the device structure for the driver instance.
 * @param channel_cfg  Channel configuration.
 *
 * @retval 0       On success.
 * @retval -EINVAL If a parameter with an invalid value has been provided.
 */
__syscall int adc_channel_setup(const struct device *dev,
                                const struct adc_channel_cfg *channel_cfg);
```

- adc 读操作

```
/**
 * @brief Set a read request.
 *
 * @param dev       Pointer to the device structure for the driver instance.
 * @param sequence  Structure specifying requested sequence of samplings.
 *
 * If invoked from user mode, any sequence struct options for callback must
 * be NULL.
 *
```

(下页继续)

(续上页)

```
 * @retval 0         On success.
 * @retval -EINVAL  If a parameter with an invalid value has been provided.
 * @retval -ENOMEM  If the provided buffer is to small to hold the results
 *                  of all requested samplings.
 * @retval -ENOTSUP If the requested mode of operation is not supported.
 * @retval -EBUSY   If another sampling was triggered while the previous one
 *                  was still in progress. This may occur only when samplings
 *                  are done with intervals, and it indicates that the selected
 *                  interval was too small. All requested samples are written
 *                  in the buffer, but at least some of them were taken with
 *                  an extra delay compared to what was scheduled.
 */
__syscall int adc_read(const struct device *dev,
                       const struct adc_sequence *sequence);
```

- adc 异步读（一般用于多个 ADC 操作）

```
/**
 * @brief Set an asynchronous read request.
 *
 * @note This function is available only if @kconfig{CONFIG_ADC_ASYNC}
 * is selected.
 *
 * If invoked from user mode, any sequence struct options for callback must
 * be NULL.
 *
 * @param dev       Pointer to the device structure for the driver instance.
 * @param sequence  Structure specifying requested sequence of samplings.
 * @param async     Pointer to a valid and ready to be signaled struct
 *                  k_poll_signal. (Note: if NULL this function will not notify
 *                  the end of the transaction, and whether it went successfully
 *                  or not).
 *
 * @returns 0 on success, negative error code otherwise.
 *          See adc_read() for a list of possible error codes.
 *
 */
__syscall int adc_read_async(const struct device *dev,
                             const struct adc_sequence *sequence,
                             struct k_poll_signal *async);
```

- adc 获取参考电压（这里我们固定是 1.2V）

```
/**
 * @brief Get the internal reference voltage.
 *
 * Returns the voltage corresponding to @ref ADC_REF_INTERNAL,
 * measured in millivolts.
 *
 * @return a positive value is the reference voltage value.  Returns
 * zero if reference voltage information is not available.
 */
static inline uint16_t adc_ref_internal(const struct device *dev)
{
        const struct adc_driver_api *api =
                                (const struct adc_driver_api *)dev->api;

        return api->ref_internal;
}
```

Driver: Counter

1 **功能概述** 该 sample 演示了外设模块 COUNTER 的连续计数功能，间隔 20ms 触发一次中断。

2 环境要求

- PAN1080 EVB 一块
- Micro USB 线一条（用于供电和查看串口打印 Log）
- 硬件接线：
  - 使用 USB 线，将 PC USB 与 EVB MicroUSB（USB->UART）相连
  - 使用**杜邦线**将 EVB 上的:
    - UART1 TX 与 P06 相连
    - UART1 RX 与 P07 相连
- PC 软件: 串口调试助手（UartAssist）或终端工具（SecureCRT），波特率 921600

3 编译和烧录

例程位置：`zephyr\samples_panchip\drivers\counter`

目前可使用 ZAL 工具或 quick build **脚本**进行编译和下载。

脚本位置：`quick_build_samples\drivers\counter.bat`

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`，可查看源码，执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明

4.1 连续计数模式

1. 设置回调函数

```
static void top_handler(const struct device *dev, void *user_data)
{
        uint32_t cnt;

        counter_get_value(dev, &cnt);                          /* 获取中断产生时的计数器值 */
        if (last_cnt > cnt) {
                cnt = cnt + 0xffffff - last_cnt;
        }
        LOG_INF("isr,expect counter is 320000, meas count is %d", cnt - last_cnt);
        last_cnt = cnt;                                                    /* 保存上一
次中断计数器值 */
        if (user_data != exp_user_data) {
                LOG_INF("Unexpected callback\n");
        }
        if (IS_ENABLED(CONFIG_ZERO_LATENCY_IRQS)) {
                top_cnt++;

                return;
```

(下页继续)

(续上页)

```
        }

        k_sem_give(&top_cnt_sem);
}
```

2. 获取设备并配置定时时间

```
dev = device_get_binding(DT_LABEL(DT_INST(0, panchip_pan_timer)));
if (!dev) {
    LOG_INF("Cannot get timer0 device\n");
} else {
    /* if more counter drivers exist other than RTC,
                * the test value set to 20000 by default
                */
    counter_period_us = 20000;                /*20ms*/
}
top_cfg.ticks = counter_us_to_ticks(dev, counter_period_us);        /* 转换为 tick 值 */
```

3. 启动 COUNTER。

```
err = counter_start(dev);
if (err != 0) {
    LOG_INF("Counter failed to start\n");
}
```

4. 设置 COUNTER 比较值。

```
err = counter_set_top_value(dev, &top_cfg);
if (err != 0) {
    LOG_INF("Counter failed to set top value (err: %d)\n", err);
}
```

5. 观测打印信息，查看获取的比较值与期望值是否符合，打印信息如下：

```
[10:47:58.410]收←◆[00:05:11.189,000] □[0m<inf> counter_sample: isr,expect counter is 320000, meas count is 320498□[0m
[00:05:11.210,000] □[0m<inf> counter_sample: isr,expect counter is 320000, meas count is 320222□[0m
[00:05:11.230,000] □[0m<inf> counter_sample: isr,expect counter is 320000, meas count is 320053□[0m
[00:05:11.250,000] □[0m<inf> counter_sample: isr,expect counter is 320000, meas count is 320053□[0m
[00:05:11.270,000] □[0m<inf> counter_sample: isr,expect counter is 320000, meas count is 320053□[0m
[00:05:11.290,000] □[0m<inf> counter_sample: isr,expect counter is 320000, meas count is 320053□[0m
[00:05:11.310,000] □[0m<inf> counter_sample: isr,expect counter is 320000, meas count is 320053□[0m
[00:05:11.330,000] □[0m<inf> counter_sample: isr,expect counter is 320000, meas count is 320053□[0m
[00:05:11.350,000] □[0m<inf> counter_sample: isr,expect counter is 320000, meas count is 320053□[0m
[00:05:11.370,000] □[0m<inf> counter_sample: isr,expect counter is 320000, meas count is 320053□[0m
[00:05:11.391,000] □[0m<inf> counter_sample: isr,expect counter is 320000, meas count is 320513□[0m
```

5 开发说明

5.1 启用 COUNTER 模块

在 prj.conf 文件中添加 “CONFIG_COUNTER=y” 启用 COUNTER 模块。

```
CONFIG_COUNTER=y
CONFIG_COUNTER_PANCHIP_TIMER0=y
```

5.2 COUNTER 数据结构及参数含义

计数器配置

```
/** @brief Top value configuration structure.
 *
 * @param ticks               Top value.
 * @param callback        Callback function. Can be NULL.
 * @param user_data        User data passed to callback function. Not valid if
```

(下页继续)

(续上页)

```
 *                        callback is NULL.
 * @param flags                 Flags. See @ref COUNTER_TOP_FLAGS.
 */
struct counter_top_cfg {
        uint32_t ticks;
        counter_top_callback_t callback;
        void *user_data;
        uint32_t flags;
};
```

到点警报配置

```
/** @brief Alarm callback structure.
 *
 * @param callback Callback called on alarm (cannot be NULL).
 * @param ticks Number of ticks that triggers the alarm. It can be relative (to
 *                now) or absolute value (see @ref COUNTER_ALARM_CFG_ABSOLUTE).
 *                Absolute alarm cannot be set further in future than top_value
 *                decremented by the guard period. Relative alarm ticks cannot
 *                exceed current top value (see @ref counter_get_top_value).
 *                If counter is clock driven then ticks can be converted to
 *                microseconds (see @ref counter_ticks_to_us). Alternatively,
 *                counter implementation may count asynchronous events.
 * @param user_data User data returned in callback.
 * @param flags        Alarm flags. See @ref COUNTER_ALARM_FLAGS.
 */
struct counter_alarm_cfg {
        counter_alarm_callback_t callback;
        uint32_t ticks;
        void *user_data;
        uint32_t flags;
};
```

5.3 COUNTER API 接口

- 接口总览

```
__subsystem struct counter_driver_api {
        counter_api_start start;
        counter_api_stop stop;
        counter_api_get_value get_value;
        counter_api_set_alarm set_alarm;
        counter_api_cancel_alarm cancel_alarm;
        counter_api_set_top_value set_top_value;
        counter_api_get_pending_int get_pending_int;
        counter_api_get_top_value get_top_value;
        counter_api_get_guard_period get_guard_period;        /* 暂不支持 */
        counter_api_set_guard_period set_guard_period;        /* 暂不支持 */
};
```

- 启动 counter，开始计数

```
/**
 * @brief Start counter device in free running mode.
 *
 * @param dev Pointer to the device structure for the driver instance.
 *
 * @retval 0 If successful.
 * @retval Negative errno code if failure.
 */
__syscall int counter_start(const struct device *dev);
```

- 停止 counter，暂停计数

```
/**
 * @brief Stop counter device.
 *
 * @param dev Pointer to the device structure for the driver instance.
 *
 * @retval 0 If successful.
 * @retval -ENOTSUP if the device doesn't support stopping the
 *                  counter.
 */
__syscall int counter_stop(const struct device *dev);
```

- 获取 counter 计数器值

```
/**
 * @brief Get current counter value.
 * @param dev Pointer to the device structure for the driver instance.
 * @param ticks Pointer to where to store the current counter value
 *
 * @retval 0 If successful.
 * @retval Negative error code on failure getting the counter value
 */
__syscall int counter_get_value(const struct device *dev, uint32_t *ticks);
```

- 设置到点警报

```
/**
 * @brief Set a single shot alarm on a channel.
 *
 * After expiration alarm can be set again, disabling is not needed. When alarm
 * expiration handler is called, channel is considered available and can be
 * set again in that context.
 *
 * @note API is not thread safe.
 *
 * @param dev               Pointer to the device structure for the driver instance.
 * @param chan_id        Channel ID.
 * @param alarm_cfg        Alarm configuration.
 *
 * @retval 0 If successful.
 * @retval -ENOTSUP if request is not supported (device does not support
 *                  interrupts or requested channel).
 * @retval -EINVAL if alarm settings are invalid.
 * @retval -ETIME  if absolute alarm was set too late.
 */
__syscall int counter_set_channel_alarm(const struct device *dev,
                                        uint8_t chan_id,
                                        const struct counter_alarm_cfg *alarm_cfg);
```

- 取消到点警报

```
/**
 * @brief Cancel an alarm on a channel.
 *
 * @note API is not thread safe.
 *
 * @param dev               Pointer to the device structure for the driver instance.
 * @param chan_id        Channel ID.
 *
 * @retval 0 If successful.
 * @retval -ENOTSUP if request is not supported or the counter was not started
 *                  yet.
```

(下页继续)

(续上页)

```
 */
__syscall int counter_cancel_channel_alarm(const struct device *dev,
                                           uint8_t chan_id);
```

- 设置比较值

```
/**
 * @brief Set counter top value.
 *
 * Function sets top value and optionally resets the counter to 0 or top value
 * depending on counter direction. On turnaround, counter can be reset and
 * optional callback is periodically called. Top value can only be changed when
 * there is no active channel alarm.
 *
 * @ref COUNTER_TOP_CFG_DONT_RESET prevents counter reset. When counter is
 * running while top value is updated, it is possible that counter progresses
 * outside the new top value. In that case, error is returned and optionally
 * driver can reset the counter (see @ref COUNTER_TOP_CFG_RESET_WHEN_LATE).
 *
 * @param dev                Pointer to the device structure for the driver instance.
 * @param cfg                Configuration. Cannot be NULL.
 *
 * @retval 0 If successful.
 * @retval -ENOTSUP if request is not supported (e.g. top value cannot be
 *                  changed or counter cannot/must be reset during top value
 *                  update).
 * @retval -EBUSY if any alarm is active.
 * @retval -ETIME if @ref COUNTER_TOP_CFG_DONT_RESET was set and new top value
 *                  is smaller than current counter value (counter counting up).
 */
__syscall int counter_set_top_value(const struct device *dev,
                                    const struct counter_top_cfg *cfg);
```

- 获取中断标志

```
/**
 * @brief Function to get pending interrupts
 *
 * The purpose of this function is to return the interrupt
 * status register for the device.
 * This is especially useful when waking up from
 * low power states to check the wake up source.
 *
 * @param dev Pointer to the device structure for the driver instance.
 *
 * @retval 1 if any counter interrupt is pending.
 * @retval 0 if no counter interrupt is pending.
 */
__syscall int counter_get_pending_int(const struct device *dev);
```

- 获取设置的比较值

```
/**
 * @brief Function to retrieve current top value.
 *
 * @param[in]  dev    Pointer to the device structure for the driver instance.
 *
 * @return Top value.
 */
__syscall uint32_t counter_get_top_value(const struct device *dev);
```

- 获取设置的比较值

```
/**
 * @brief Function to retrieve current top value.
 *
 * @param[in]  dev    Pointer to the device structure for the driver instance.
 *
 * @return Top value.
 */
__syscall uint32_t counter_get_top_value(const struct device *dev);
```

Driver: Flash Shell

1 **功能概述** 本文主要介绍 PAN1080 EVB 板 flash shell 演示，可以通过类 shell 命令来实现 flash api 的相关操作。

2 **环境要求**

- board: pan1080a_afld_evb
- uart(波特率 921600)：显示串口 shell

3 **编译和烧录** 项目位置：`zephyr\samples_panchip\drivers\flash_shell`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\drivers\flash_shell.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 **演示说明** 通过串口命令输入 flash_cmd 命令来实现 flash api 操作，通过连续两个 `tab` 会自动补全命令，例如输入 `flash_cmd` 后，则 shell 会输出 flash_cmd 相关细指令。

详情参考 `5.1 flash_shell 支持的命令`，`5.2 flash_cmd 使用以及命令解析`.

5 **开发说明**

5.1 flash_shell **支持的命令**

```
flash_cmd - Flash related commands.
Subcommands:
  erase             :<off> <len>
                     Erase <len> bytes from device offset <off>, subject to
                     hardware page limitations.
  page_count        :Print the number of pages on the flash device.
  page_erase        :<page> [num]
                     Erase [num] pages (default 1), starting at page <page>.
  page_layout       :[start_page] [end_page]
                     Print layout of flash pages in the range [start_page,
                     end_page], which is inclusive. By default, all pages are
                     printed.
```

(下页继续)

(续上页)

```
  page_read          :<page> <len> OR <page> <off> <len>
                      Read <len> bytes from given page, starting at page offset
                      <off>,or offset 0 if not given. No checks are made that
                      bytes read are all within the page.
  page_write         :<page> <off> <byte1> [... byteN]
                      Write given bytes to given page, starting at page offset
                      <off>. No checks are made that the bytes all fall within
                      the page. Pages must be erased before they can be written.
  read               :<off> <len>
                      Read <len> bytes from device offset <off>.
  set_device         :<device_name>
                      Set flash device by name. If a flash device was not found,
                      this command must be run first to bind a device to this
                      module.
  write              :<off> <byte1> [... byteN]
                      Write given bytes, starting at device offset <off>.
                      Pages must be erased before they can be written.
  write_block_size   :Print the device's write block size. This is the smallest
                      amount of data which may be written to the device, in
                      bytes.
  write_unaligned    :<off> <byte1> [... byteN]
                      Write given bytes, starting at device offset <off>.
                      Being unaligned, affected memory areas are backed up,
                      erased, protected and then overwritten.
                      This command is designed to test writing to large flash
                      pages.
  write_pattern      :<off> <len>
                      Writes a pattern of (0x00 0x01 .. 0xFF 0x00 ..) of
                      length<len> at the offset <off>.
                      Unaligned writing is used, i.e. protection and erasing are
                      automated.This command is designed to test writing to large
                      flash pages.
```

5.2 flash_cmd 使用以及命令解析

5.2.1 使用 flash_cmd read 读取地址 0x1000 处 16 个字节

```
uart:~$ flash_cmd read 0x1000 0x10
53 46 1d 68 | af 68 fb 68 |00 2b 04 d0 | 00 21 28 00
```

5.2.2 使用 flash_cmd write 写入地址 0xc8000 处 4 个字节

```
uart:~$ flash_cmd read 0xc8000 0x4                                    //从 800K 地址处读取 4
个字节，因为该地址正常没写入过，默认为全 ff
ff ff ff ff
uart:~$ flash_cmd write 0xc8000 0x1 0x2 0x3 0x4                //从 800K 地址处写入 4 个字节:
0x1 0x2 0x3 0x4
Reading back written bytes:
01 02 03 04
uart:~$ flash_cmd read 0xc8000 0x4                                    //从 800K 地址处读取 4
个字节: 0x1 0x2 0x3 0x4
01 02 03 04
```

5.2.3 使用 flash_cmd erase 擦除地址 0xc8000 处 4K 字节

```
uart:~$ flash_cmd erase 0xc8000 0x1000                              //从 800K 地址处擦除 4K 字
节，默认擦写必须 4K 对齐，这个可以使用 flash_cmd write_block_size
                                                                                              /
↪/获得，该值默认在 dts 中定义。
```

(下页继续)

(续上页)

```
uart:~$ flash_cmd read 0xc8000 0x4
ff ff ff ff
```

5.2.4 **flash_cmd read 命令代码解析** flash_cmd 所有的命令均在下面，从 SHELL_CMD_ARG(read, NULL, READ_HELP, cmd_read, 3, 0) 我们可以知道 read 命令对应的实现函数，在 cmd_read 中。

```
SHELL_STATIC_SUBCMD_SET_CREATE(sub_flash,
        /* Alphabetically sorted to ensure correct Tab autocompletion. */
        SHELL_CMD_ARG(erase,        NULL,        ERASE_HELP,        cmd_erase, 3, 0),
#ifdef CONFIG_FLASH_PAGE_LAYOUT
        SHELL_CMD_ARG(page_count,  NULL, PAGE_COUNT_HELP, cmd_page_count, 1, 0),
        SHELL_CMD_ARG(page_erase, NULL, PAGE_ERASE_HELP, cmd_page_erase, 2, 1),
        SHELL_CMD_ARG(page_layout, NULL, PAGE_LAYOUT_HELP,
                      cmd_page_layout, 1, 2),
        SHELL_CMD_ARG(page_read,   NULL, PAGE_READ_HELP,  cmd_page_read, 3, 1),
        SHELL_CMD_ARG(page_write,  NULL, PAGE_WRITE_HELP,
                      cmd_page_write, 3, 255),
#endif
        SHELL_CMD_ARG(read,                NULL,        READ_HELP,        cmd_read, 3, 0),
        SHELL_CMD_ARG(set_device, NULL, SET_DEV_HELP, cmd_set_dev, 2, 0),
        SHELL_CMD_ARG(write,          NULL,        WRITE_HELP,        cmd_write, 3, 255),
        SHELL_CMD_ARG(write_block_size,        NULL,        WRITE_BLOCK_SIZE_HELP,
                                                    cmd_write_block_size, 1, 0),
        SHELL_CMD_ARG(write_unaligned,  NULL,        WRITE_UNALIGNED_HELP,
                                                       cmd_write_unaligned, 3, 255),
        SHELL_CMD_ARG(write_pattern,        NULL,        WRITE_PATTERN_HELP,
                                                       cmd_write_pattern, 3, 255),
        SHELL_SUBCMD_SET_END /* Array terminated. */
);

SHELL_CMD_REGISTER(flash_cmd, &sub_flash, "Flash related commands.", NULL);
```

5.3 Flash API 使用参考

5.3.1 flash_read API **参考** cmd_read 也是调用 do_read 函数，对于 flash_read 函数我们可以参考 do_read 函数：

```
/* Read bytes, dumping contents to console and printing on error. */
static int do_read(const struct shell *shell, off_t offset, size_t len)
{
        uint8_t buf[64];
        int ret;

        while (len > sizeof(buf)) {
                ret = flash_read(flash_device, offset, buf, sizeof(buf));
                if (ret) {
                        goto err_read;
                }
                dump_buffer(shell, buf, sizeof(buf));
                len -= sizeof(buf);
                offset += sizeof(buf);
        }
        ret = flash_read(flash_device, offset, buf, len);
        if (ret) {
                goto err_read;
        }
        dump_buffer(shell, buf, len);
```

(下页继续)

(续上页)

```
        return 0;

 err_read:
        PR_ERROR(shell, "flash_read error: %d\n", ret);
        return ret;
}
```

5.3.2 flash_write API 参考

```
/* Write bytes and printing on error. */
static int do_write(const struct shell *shell, off_t offset, uint8_t *buf,
                    size_t len, bool read_back)
{
        int ret;

        ret = flash_write(flash_device, offset, buf, len);
        if (ret) {
                PR_ERROR(shell, "flash_write failed (err:%d).\n", ret);
                return ret;
        }

        if (read_back) {
                PR_SHELL(shell, "Reading back written bytes:\n");
                ret = do_read(shell, offset, len);
        }
        return ret;
}
```

5.3.3 flash_erase API 参考

```
/* Erase area and printing on error. */
static int do_erase(const struct shell *shell, off_t offset, size_t size)
{
        int ret;

        ret = flash_erase(flash_device, offset, size);
        if (ret) {
                PR_ERROR(shell, "flash_erase failed (err:%d).\n", ret);
                return ret;
        }

        return ret;
}
```

5.3.4 do_write_unaligned 非对齐地址写参考实现

```
static int do_write_unaligned(const struct shell *shell, off_t offset, uint8_t *buf,
                    size_t len, bool read_back)
{
        int ret = 0;
        size_t page_size = flash_get_write_block_size(flash_device);
        size_t size_before = offset % page_size;
        size_t size_after = page_size - ((size_before + len) % page_size);
        size_t aligned_size = size_before + len + size_after;
        off_t  start_page = offset - size_before;
        off_t  last_page = start_page + aligned_size - page_size;
        bool single_page_write = (size_before + len < page_size);

        char *before_data;
```

(下页继续)

(续上页)

```
        char *after_data;

        if (0 == size_before && 0 == size_after) {
                /* Aligned write */
                flash_erase(flash_device, offset, len);
                flash_write(flash_device, offset, buf, len);

                return 0;
        }

        before_data = k_malloc(page_size);
        after_data = k_malloc(page_size);

        if (!before_data || !after_data) {
                PR_ERROR(shell, "No heap memory for flash manipulation\n");
                ret = -ENOMEM;
                goto free_buffers;
        }

        /* Stash useful data from the pages that will be affected. */
        if (single_page_write) {
                /* Read old data before new data is written. */
                if (size_before) {
                        flash_read(flash_device, start_page, before_data, size_before);
                }
                /* Fill the with new data. */
                memcpy(before_data + size_before, buf, len);
                /* Fill the last part of old data. */
                if (size_after) {
                        flash_read(flash_device, offset + len,
                                        before_data + size_before + len,
                                        size_after);
                }
        } else {
                /* Multipage write, different start and end pages. */
                if (size_before) {
                        flash_read(flash_device, start_page, before_data,
                                        size_before);
                        /* Fill the rest with new data. */
                        memcpy(before_data + size_before, buf,
                               page_size - size_before);
                }
                if (size_after) {
                        /* Copy ending part of new data. */
                        memcpy((void *)after_data,
                               (void *)(buf + len -
                                   ((len + size_before) % page_size)),
                               page_size - size_after);
                        /* Copy ending part of flash page. */
                        flash_read(flash_device, offset + len,
                                        after_data + (page_size - size_after),
                                        size_after);
                }
        }

        /* Erase all the pages that overlap with new data. */
        flash_erase(flash_device, start_page, aligned_size);

        /* Write stashed and new data. */
        if (single_page_write || size_before > 0) {
                /* Write first page if available. */
```

(下页继续)

(续上页)

```
                flash_write(flash_device, start_page, before_data,
                            page_size);
        }
        if (!single_page_write) {
                size_t middle_data_len = aligned_size;
                off_t middle_page_start = start_page;
                off_t data_offset = (off_t)buf;

                /* Write the middle bit if available */
                if (size_before > 0) {
                        middle_page_start += page_size;
                        middle_data_len -= page_size;
                        data_offset += (page_size - size_before);
                }
                if (size_after > 0) {
                        middle_data_len -= page_size;
                }
                if (middle_data_len > 0) {
                        flash_write(flash_device, middle_page_start,
                                    (const void *)data_offset,
                                    middle_data_len);
                }

                /* Write the last page if needed. */
                if (size_after > 0) {
                        flash_write(flash_device, last_page, after_data,
                                    page_size);
                }
        }

        if (read_back) {
                PR_SHELL(shell, "Reading back written bytes:\n");
                ret = do_read(shell, offset, len);
        }

free_buffers:
        k_free(before_data);
        k_free(after_data);

        return ret;
}
```

Driver: GPIO

1 功能概述 本文主要介绍 PAN1080 EVB 板 GPIO 输入，输出，中断模式的使用方法。

2 环境要求

- board: pan1080a_afld_evb
- uart (波特率 921600): 显示串口 log

3 编译和烧录 项目位置: `zephyr\samples_panchip\drivers\gpio`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置: `quick_build_samples\drivers\gpio.bat`。

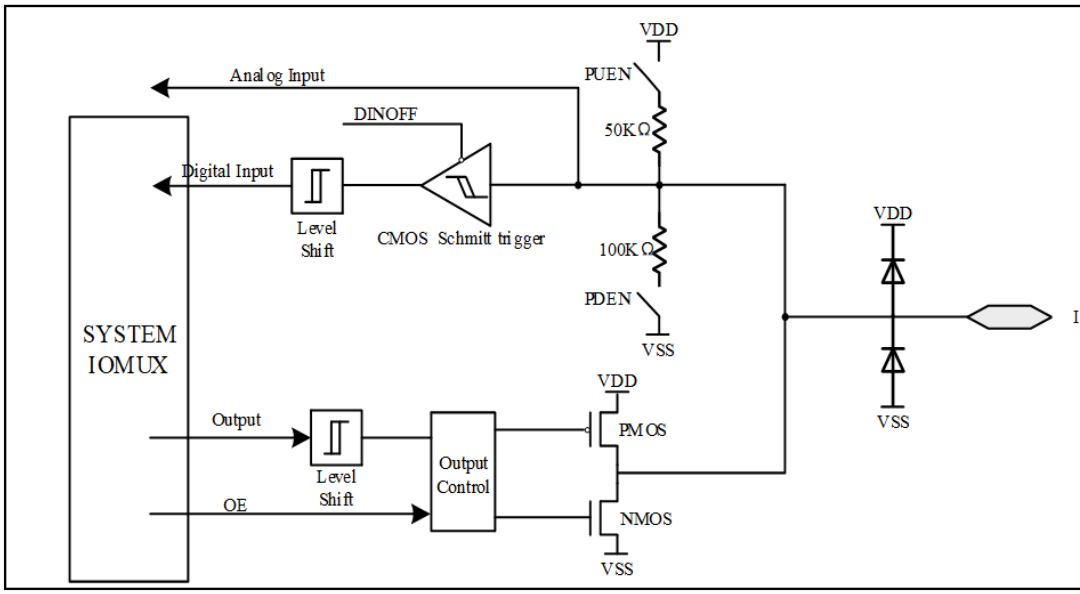
打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`，可查看源码，执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明

1. GPIO 输入和 GPIO 中断可以通过串口观察打印。
2. GPIO 输出需要通过逻辑分析仪器观察输出电平。
3. 详情可以参考 5.5 测试现象说明。

5 开发者说明



5.1 GPIO Block Diagram

5.2 Zephyr 中提供的 GPIO 工作模式

```
/** Output: 典型的输出模式 */
#define OUT_PUSH_PULL                   1           //推挽输出，不使能数字输入
#define OUT_PUSH_PULL_IN_DIGTAL         0           //推挽输出，使能数字输入
#define OUT_OPEN_DRAIN                  0           //开漏输出，不使能内部的上拉或者下拉电
路，1 的状态由外部电路决定
#define OUT_OPEN_DRAIN_EXTRA_PULL_UP    0           //开漏输出，使能内部的上拉电路
#define OUT_QUASI_BI_EXTRA_PULL_UP      0           //准双向输出，使能内部的上拉电路；0 读
时可能需要写 1，即确保内部为开漏断开状态

/** Input: 典型的输入模式 */
#define IN_DIGTAL                       0           //数字输入使能，处于悬空 (Floating)
#define IN_DIGTAL_EXTRA_PULL_UP         0           //数字输入使能，使能内部的上拉电路
#define IN_DIGTAL_EXTRA_PULL_DOWN       0           //数字输入使能，使能内部的下拉电路
#define IN_ANALOG                       0           //模拟输入，即数字输入关闭

/** Interruption: 典型的中断模式 */
#define INT_EDGE_RISING                 0           //上升沿中断触发模式
```

(下页继续)

(续上页)

```
#define INT_EDGE_FALLING                0               //下降沿中断触发模式
#define INT_EDGE_BOTH                   0               //边沿触发模式
#define INT_LEVEL_HIGH                  0               //高电平中断触发模式
#define INT_LEVEL_LOW                   0               //低电平中断触发模式
```

5.3 测试 Case 使用说明　使能相应的宏为 1 即打开相应的测试模式 case，例如 OUT_PUSH_PULL 设置为 1，即使能推挽输出测试，详情参考 5.2 Zephyr 中提供的 GPIO 工作模式。注意不支持同时测试多个 case，否则会编译失败。

5.4 Sample 中 GPIO API 说明

输入输出模式配置

```
        /**step1: 获取 GPIO 端口的 port*/
        const struct device *port;
        port = device_get_binding(PORT);

        /**step2: 配置 GPIO 模式，详细模式参考输入输出模式 */
        gpio_pin_configure(port, PIN, GPIO_OUTPUT | GPIO_PUSH_PULL);

        /**step3: GPIO 读写接口 */
        gpio_pin_set(port, PIN,  1);                        //GPIO 设置接口
        pin_val = gpio_pin_get(port, PIN);                 //GPIO 读接口
```

中断模式配置

```
        /**step1: 获取 GPIO 端口的 port*/
        const struct device *port;
        port = device_get_binding(PORT);

        /**step2: 配置 GPIO 模式，此处配置成输入模式 */
        gpio_pin_configure(port, PIN, GPIO_OUTPUT);

        /**step3: 配置中断模式以及回调函数 */
        gpio_init_callback(&gpio_cb, callback_edge, BIT(PIN));        //初始化中断回调函数，添加
到 gpio_cb
        ret = gpio_add_callback(port, &gpio_cb);                                    //将 gpio_cb 添
加到 port 中
        ret = gpio_pin_interrupt_configure(port, PIN, GPIO_INT_EDGE_RISING);        //此处中断模
式设置为上升沿

        //中断回调函数，在触发 10 次后通过`gpio_pin_interrupt_configure`关闭中断
    static void callback_edge(const struct device *port, struct gpio_callback *cb,
                  gpio_port_pins_t pins)
    {
        cb_count++;
        printk("Detected interrupt on edge:%d\n", cb_count);

        if (cb_count >= MAX_INT_CB_COUNT) {
            //关闭 GPIO 中断
            gpio_pin_interrupt_configure(port, PIN, GPIO_INT_DISABLE);
            printk("Disable GPIO interruption\n");
        }
    }
```

关于如何选择 GPIO　请参考 Device Tree 和 PINMUX 功能相应文档，该 sample 选择的是 dts 中已经设置的 led 引脚（P21）。

5.5 **测试现象说明** 在测试条件有限时，两个 EVB 通过该 Sample 进行互相测试时 (箭头左右为不同 EVB)：

```
OUT_PUSH_PULL <===> IN_DIGTAL (IN_DIGTAL 间隔 1 秒输出 0, 1)

OUT_PUSH_PULL_IN_DIGTAL <===> IN_DIGTAL (OUT_PUSH_PULL_IN_DIGTAL, IN_DIGTA 均能间隔 1 秒输出 0, 1)

OUT_OPEN_DRAIN <===> IN_DIGTAL_EXTRA_PULL_UP (IN_DIGTAL_EXTRA_PULL_UP 间隔 1 秒输出 0, 1)

OUT_OPEN_DRAIN_EXTRA_PULL_UP <===> IN_DIGTAL (IN_DIGTAL 间隔 1 秒输出 0, 1)

OUT_QUASI_BI_EXTRA_PULL_UP <===> IN_DIGTAL (IN_DIGTAL 间隔 1 秒输出 0, 1)
```

```
IN_DIGTAL (OUT_PUSH_PULL 中已测)

IN_DIGTAL_EXTRA_PULL_UP (上电时读取默认值为 1; 然后将 P21 连接到 GND, 读取值为 0)

IN_DIGTAL_EXTRA_PULL_DOWN (上电时读取默认值为 0; 然后将 P21 连接到 VCC, 读取值为 1)

IN_ANALOG (未做功能测试, 根据相关模拟 case 自行测试, 例如 ADC)
```

```
INT_EDGE_RISING <===> OUT_PUSH_PULL (INT_EDGE_RISING 间隔 2 秒输出中断打印)

INT_EDGE_FALLING <===> OUT_PUSH_PULL (INT_EDGE_FALLING 间隔 2 秒输出中断打印)

INT_EDGE_BOTH <===> OUT_PUSH_PULL (INT_EDGE_BOTH 间隔 1 秒输出中断打印)

INT_LEVEL_HIGH <===> OUT_PUSH_PULL (INT_LEVEL_HIGH 在 OUT_PUSH_PULL 初始化一直输出中断打印, 直到关闭中断)

INT_LEVEL_LOW <===> OUT_PUSH_PULL (INT_LEVEL_LOW 在 OUT_PUSH_PULL 初始化后 1 秒一直输出中断打印, 直到关闭中断)
```

Driver: I2C Master

1 **功能概述** `i2c_master sample` 演示了 Zephyr I2C Driver 在 PAN1080 SoC 上的使用方法，主要包括：

- I2C Master 配置
- 按照地址 I2C Slave 地址写入 I2C 数据
- 按照地址 I2C Slave 地址读取 I2C 数据
- (可选) 通过逻辑分析仪抓取 I2C 数据观测

2 **环境要求**

- PAN1080 EVB 一块
- Micro USB 线一条（用于供电和查看串口打印 Log）
- （可选）Saleae16 Logic Analysis 与配套软件 `Logic 2.3.50`
- 硬件接线：
  - 使用 USB 线，将 PC USB 与 EVB MicroUSB（USB->UART）相连
  - 使用**杜邦线**将 EVB 上的:
    - Master P40 接口与 Slave P40 相连
    - Master P41 接口与 Slave P41 相连

  * （可选）Master P40 与逻辑分析仪 SDA channel 相连，Master P41 与逻辑分析仪 SCL channel 相连，并且 EVB 与 Logic Analysis GND 相连

- PC 软件: 串口调试助手（UartAssist）或终端工具（SecureCRT），波特率 921600

3 编译和烧录

例程位置：`zephyr\samples_panchip\driver\i2c\i2c_master`

目前可使用 ZAL 工具或 quick build **脚本**进行编译和下载。

脚本位置：`quick_build_samples\drivers\i2c_master.bat`

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`，可查看源码，执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明

- 连线

运行脚本 `quick_build_samples\i2c_master.bat`，编译后进行烧录至 pan1080a_afld_evb

之后将 master P40（I2C SDA）与 slave 端 P40 进行连接，master P41（I2C SCL）与 slave 端 P41 进行连接

有额外条件还可将 P40，P41 与逻辑分析仪 2 个 channel 进行连接，GND 与逻辑分析仪 GND 连接后，开始抓数据

I2C Master	I2C Slave	Logic Analyzer
P40（I2C SDA）	P40（I2C SDA）	CH0 SDA
P41（I2C SCL）	P41（I2C SCL）	CH1 SCL
GND		GND

- Sample 执行流程:
  - 先进行 I2C Slave 的下载并复位
  - 配置 I2C 为 Master 模式，并配置标准速率模式
  - 生成 16B 数据（Ex：data[i] = i），通过 I2C Master 指定 Slave I2C 通信地址（0x50）后，发送全部 16B 数据
  - 通过 I2C Master 指定 Slave I2C 通信地址（0x50）后，读取 16B 数据
  - 对比写入与读出的数据是否一致

  Slave 端收转发，Master 端发转收，完成 I2C Master 对 I2C Slave 的读写操作

- 示例打印出 master 读出的写入 slave 数据

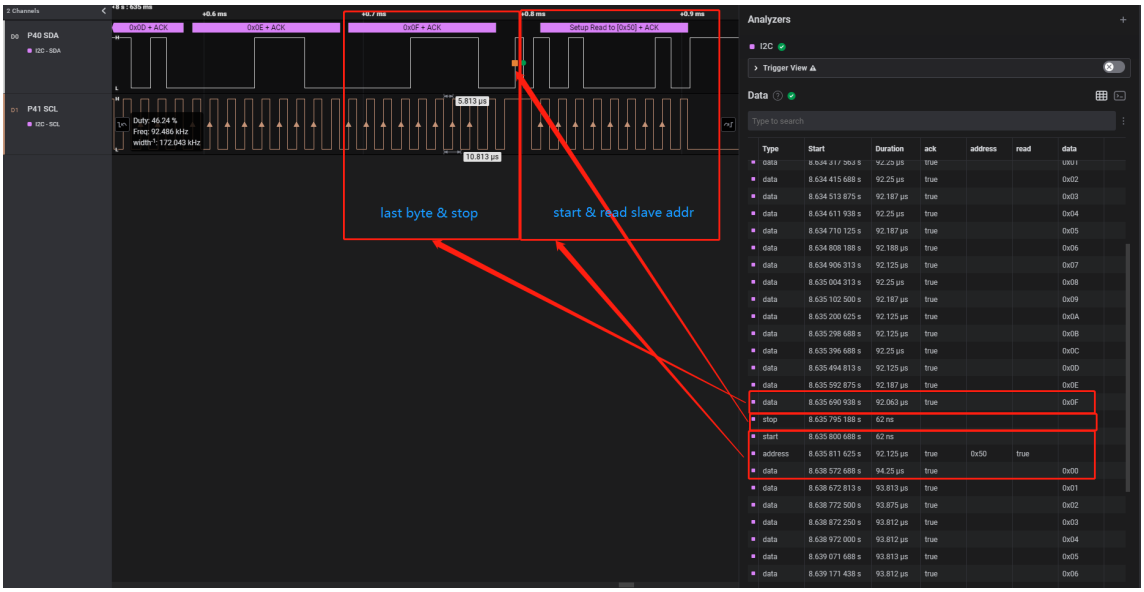
```
  cmp_data[0] = 0
  cmp_data[1] = 1
  cmp_data[2] = 2
  cmp_data[3] = 3
  cmp_data[4] = 4
  cmp_data[5] = 5
  cmp_data[6] = 6
  cmp_data[7] = 7
```

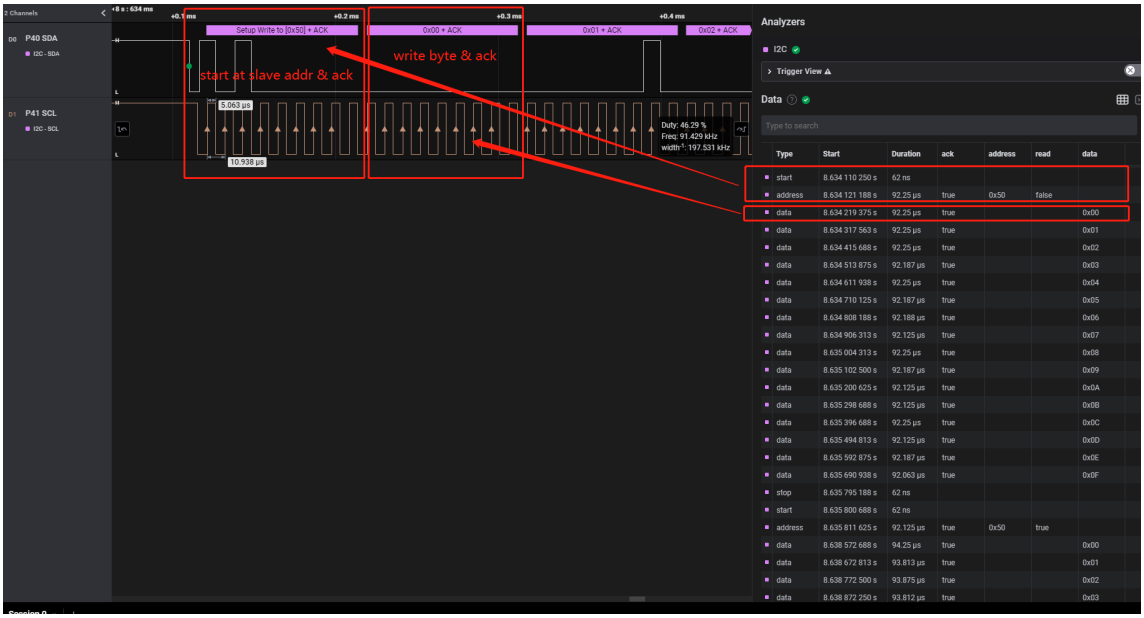
（下页继续）

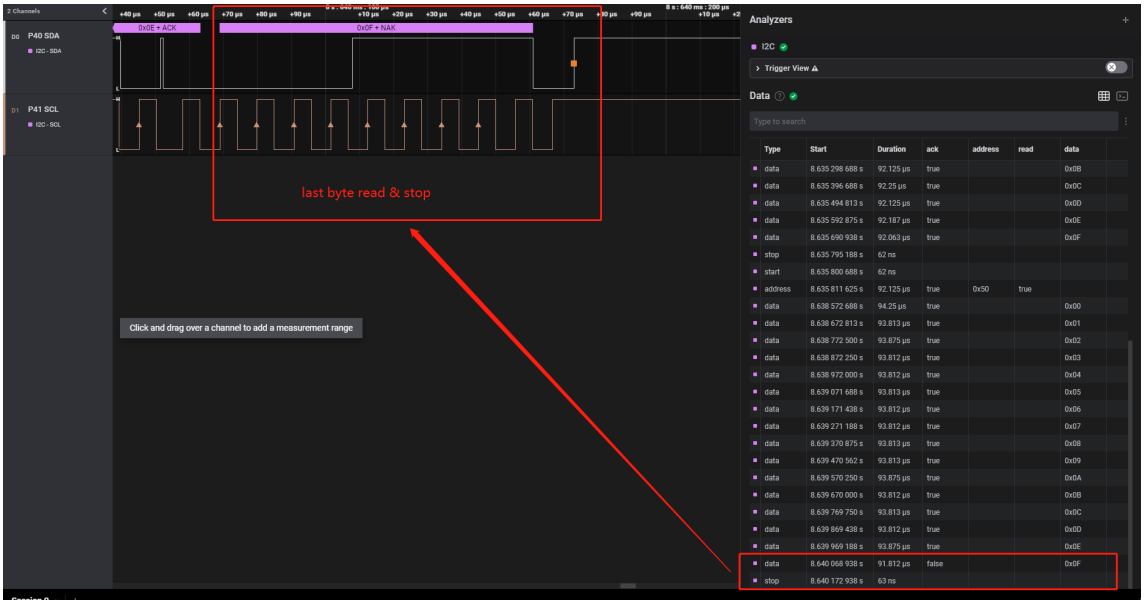
(续上页)

```
cmp_data[8] = 8
cmp_data[9] = 9
cmp_data[10] = 10
cmp_data[11] = 11
cmp_data[12] = 12
cmp_data[13] = 13
cmp_data[14] = 14
cmp_data[15] = 15
```

- 逻辑分析仪示例数据：抓取了从写开始到写到终止，到读开始到读终止







5 开发说明

5.1 I2C Kconfig　使用 I2C 时，需要使能 I2C 指定宏

```
CONFIG_I2C=y
```

5.2 I2C API　I2C 使用包括：获取 I2C 设备；配置 I2C 基础模式；配置读/写进行通信操作

相关底层实现存在于 `zephyr\drivers\i2c\i2c_panchip.c`

5.2.1 通过 label 获取 DTS I2C

```
const struct device *i2c_dev = DEVICE_DT_GET(DT_NODELABEL(i2c0));
```

5.2.2 配置 I2C 速率与 Master 模式

```
uint32_t i2c_cfg = I2C_SPEED_SET(I2C_SPEED_STANDARD) | I2C_MODE_MASTER
i2c_configure(i2c_dev, i2c_cfg)
```

5.2.3 I2C Master Write to I2C Slave Addr

```
i2c_write(i2c_dev, datas, 16, TAR_I2C_ADDR);
```

5.2.4 I2C Master Read to I2C Slave Addr

```
i2c_read(i2c_dev, cmp_data, 16, TAR_I2C_ADDR);
```

Driver: I2C Slave

1 **功能概述**　`i2c_slave sample` 演示了 Zephyr I2C Driver 在 PAN1080 SoC 上的使用方法，主要包括：

- I2C Slave 配置
- 配置 I2C Slave 地址，读取 I2C 数据并保存
- 配置 I2C Slave 地址，写入读到的 I2C 数据

- (可选) 通过逻辑分析仪抓取 I2C 数据观测

2 环境要求

- PAN1080 EVB 一块
- Micro USB 线一条（用于供电和查看串口打印 Log）
- （可选）Saleae16 Logic Analysis 与配套软件 `Logic 2.3.50`
- 硬件接线：
  - 使用 USB 线，将 PC USB 与 EVB MicroUSB（USB->UART）相连
  - 使用**杜邦线**将 EVB 上的:
    - Master P40 接口与 Slave P40 相连
    - Master P41 接口与 Slave P41 相连
    - （可选）Master P40 与逻辑分析仪 SDA channel 相连，Master P41 与逻辑分析仪 SCL channel 相连，并且 EVB 与 Logic Analysis GND 相连
- PC 软件: 串口调试助手（UartAssist）或终端工具（SecureCRT），波特率 921600

3 编译和烧录

例程位置：`zephyr\samples_panchip\driver\i2c\i2c_slave`

目前可使用 ZAL 工具或 quick build **脚本**进行编译和下载。

脚本位置：`quick_build_samples\drivers\i2c_slave.bat`

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明

- 连线

运行脚本 `quick_build_samples\i2c_slave.bat`，编译后进行烧录至 pan1080a_afld_evb

之后将 master P40（I2C SDA）与 slave 端 P40 进行连接，master P41（I2C SCL）与 slave 端 P41 进行连接

有额外条件还可将 P40，P41 与逻辑分析仪 2 个 channel 进行连接，GND 与逻辑分析仪 GND 连接后，开始抓数据

I2C Master	I2C Slave	Logic Analyzer
P40（I2C SDA）	P40（I2C SDA）	CH0 SDA
P41（I2C SCL）	P41（I2C SCL）	CH1 SCL
GND		GND

- Sample 执行流程：
  - 配置 I2C 为 Slave 模式，并配置标准速率模式
  - 指定 I2C Slave 地址并读取 16B 数据

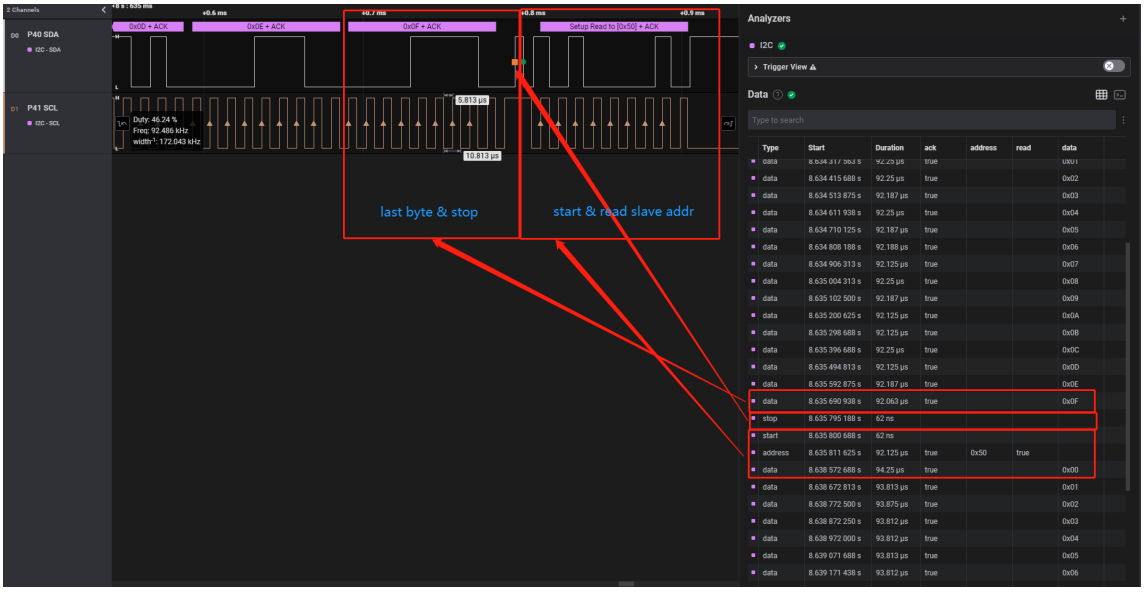
  - 将读取到的 16B 数据保存打印，并通过 I2C Write 发送全部 16B 数据
  - 进行 I2C Slave 的下载并复位，打开串口，之后进行 I2C master 的下载
  - 观察读出的数据是否与 master 发送的数据一致

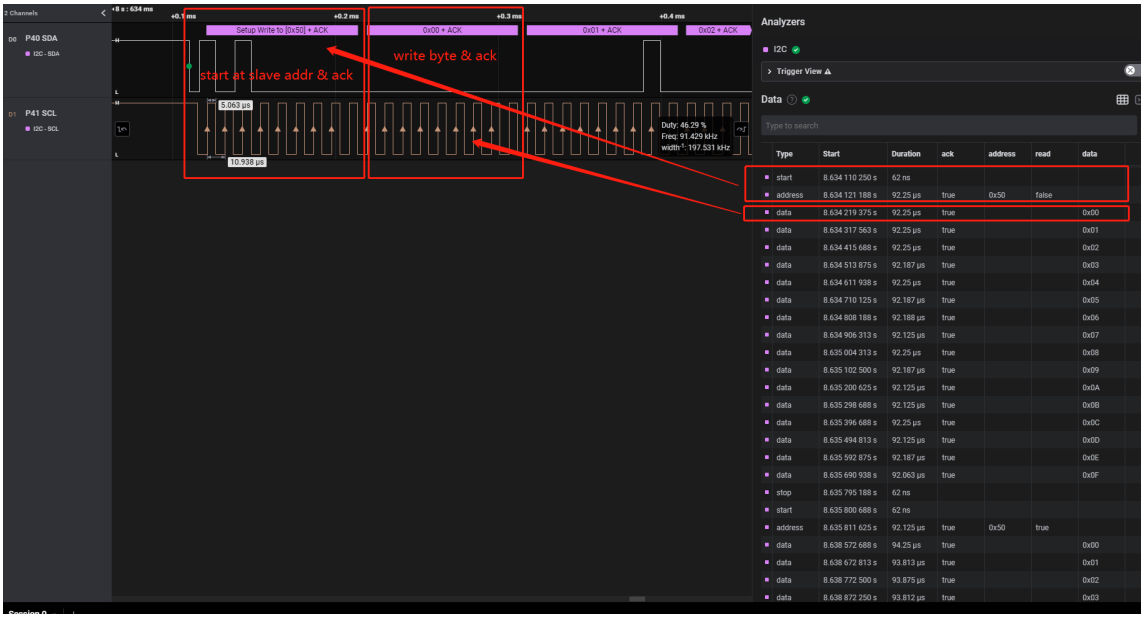
  Slave 端收转发，Master 端发转收，完成 I2C Master 对 I2C Slave 的读写操作

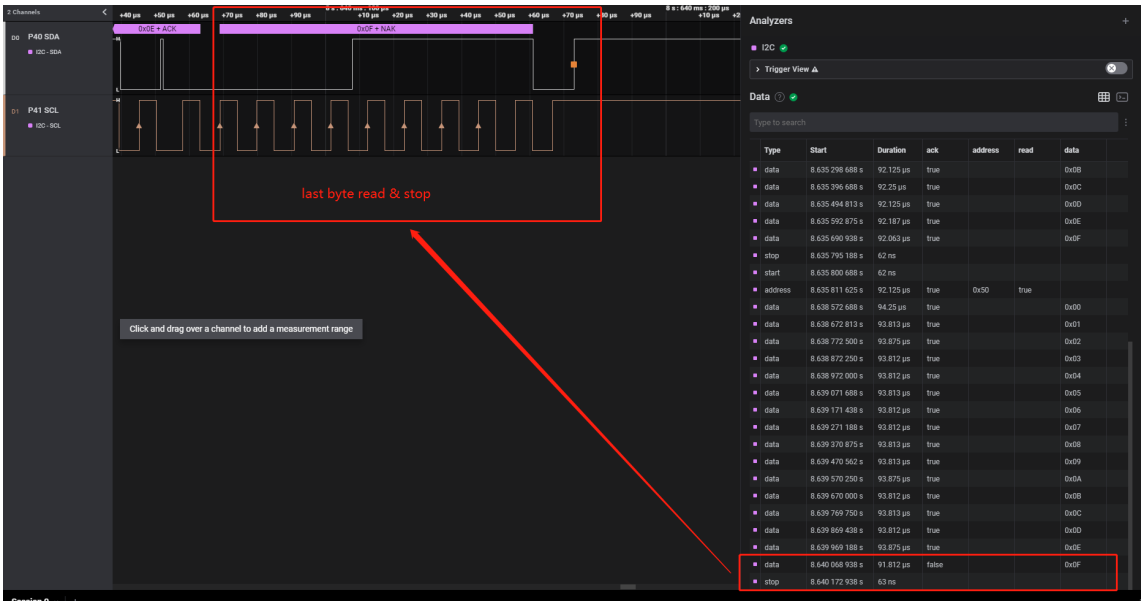
- 打印 slave 收到的 master 写入数据

```
datas[0] = 0
datas[1] = 1
datas[2] = 2
datas[3] = 3
datas[4] = 4
datas[5] = 5
datas[6] = 6
datas[7] = 7
datas[8] = 8
datas[9] = 9
datas[10] = 10
datas[11] = 11
datas[12] = 12
datas[13] = 13
datas[14] = 14
datas[15] = 15
```

- 逻辑分析仪示例数据：抓取了从写开始到写到终止，到读开始到读终止







5 开发说明

5.1 I2C Kconfig　使用 I2C 时，需要使能 I2C 指定宏

```
CONFIG_I2C=y
```

5.2 I2C API　I2C 使用包括：获取 I2C 设备；配置 I2C 基础模式；配置读/写进行通信操作

相关底层实现存在于 `zephyr\drivers\i2c\i2c_panchip.c`

5.2.1 通过 label 获取 DTS I2C

```
const struct device *i2c_dev = DEVICE_DT_GET(DT_NODELABEL(i2c0));
```

5.2.2 配置 I2C 速率与 Slave 模式

```
uint32_t i2c_cfg = I2C_SPEED_SET(I2C_SPEED_STANDARD)
i2c_configure(i2c_dev, i2c_cfg)
```

5.2.3 I2C Slave Addr Receive

```
i2c_read(i2c_dev, datas, 16, SLAVE_I2C_ADDR);
```

5.2.4 I2C Slave Addr Write

```
i2c_write(i2c_dev, datas, 16, SLAVE_I2C_ADDR);
```

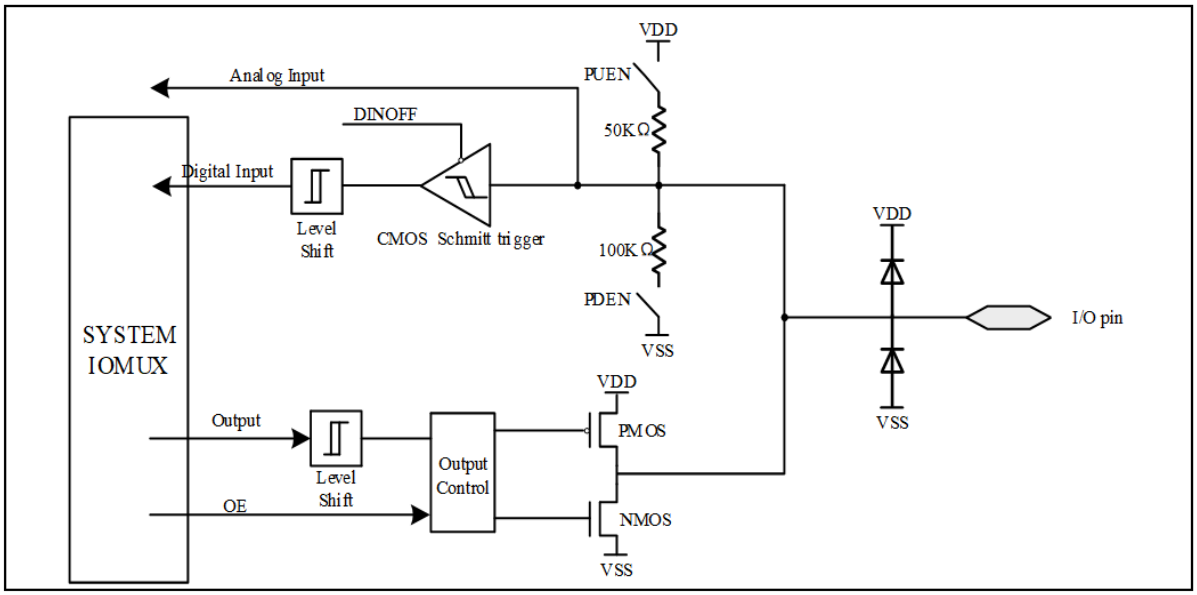
5.3 **补充说明** 目前 i2c 可以演示 i2c 通信功能，但实际的 i2c slave 功能完整实现需要借助真实 i2c slave 设备，完善 panchip 作为 i2c controller 对 i2c slave 的控制。

Driver: Pinmux

1 **功能概述** `pinmux sample` 演示了 Zephyr PINMUX Driver 在 PAN1080 SoC 上的使用方法，主要包括：

- 开启/关闭 SoC 内部上拉/下拉电阻通路开关
- 切换某个 IO 引脚的多功能引脚 MFP (Multi-Function Pin) 功能配置

硬件上，PAN1080 PINMUX 与 GPIO 是密切相关的，其框图如下所示：



其中与 PINMUX 相关的有：

- `PUEN` 对应 PINMUX 的上拉电阻通路开关
- `PDEN` 对应 PINMUX 的下拉电阻通路开关
- `DINOFF` 对应 PINMUX 的数字输入通路开关
- `SYSTEM IOMUX` 对应 PINMUX 的多功能引脚 MFP 功能配置

2 **环境准备**

- PAN1080 EVB 一块
- Micro USB 线一条（用于供电和查看串口打印 Log）

- 硬件接线：
  - 使用 USB 线，将 PC USB 与 EVB MicroUSB（USB->UART）相连
  - 使用**杜邦线**将 EVB 上的:
    * UART1 TX 与 P06 相连
    * UART1 RX 与 P07 相连
- PC 软件: 串口调试助手（UartAssist）或终端工具（SecureCRT），波特率 921600

3 编译和烧录 例程位置：`zephyr\samples_panchip\drivers\pinmux`

目前可使用 ZAL 工具或 quick build **脚本**进行编译和下载。

脚本位置：`quick_build_samples\drivers\pinmux.bat`

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明 本例程包含 2 个示例，其中：

- `pinmux_pullup_pulldown_io_level()` 使用 PINMUX API 将一个 GPIO 的内部上拉电阻和下拉电阻分别使能，并通过 GPIO Input 功能检测线上的电平变化来确认上拉、下拉电阻是否使能成功；其涉及如下 2 个 PINMUX API:
  - pinmux_pin_pullup()
  - pinmux_pin_pulldown()
- `switch_uart_console_pinmux()` 使用 PINMUX API 将当前串口输出引脚从 `P06` 切换至 `P24`；其涉及如下 2 个 PINMUX API:
  - pinmux_pin_get()
  - pinmux_pin_set()

4.1 示例 1：使能 SoC 内部上拉与下拉电阻

1. 确认 GPIO P57 引脚为悬空状态
2. 当程序执行 `pinmux_pullup_pulldown_io_level()` 函数时，会每隔 500ms 切换一次 GPIO P57 引脚的内部上下拉电阻，每次切换后均立刻使用 GPIO 检测 IO 线上的电平，然后向串口打印当前检测结果：

```
Start to pull up and pull down GPIO P57..
Pull up,   detected IO level: 1
Pull down, detected IO level: 0
Pull up,   detected IO level: 1
Pull down, detected IO level: 0
Pull up,   detected IO level: 1
Pull down, detected IO level: 0
Pull up,   detected IO level: 1
Pull down, detected IO level: 0
Pull up,   detected IO level: 1
Pull down, detected IO level: 0
Stopped.
```

4.2 示例 2：切换 SoC 多功能引脚 MFP 功能

1. 当程序执行 `switch_uart_console_pinmux()` 函数时，其首先会先获取 P06 和 P24 引脚的当前的 MFP (Multi-Function Pin) 配置：

```
Before switch pinmux:
 - Pin function of P06 is: 0x00004000
 - Pin function of P24 is: 0x00000000
```

   此处打印的是引脚 MFP 配置的数字数字标识，其代表的含义可以从以下头文件中找到对应的含义：

   - `include\drivers\pinmux\pinmux_pan1080.h`

   例如，当前 `P06` MFP 配置的数字标识为 0x00004000，我们查阅 `pinmux_pan1080.h` 中的 P06 引脚相关的 MFP 定义，在 78 行可看到其代表的含义是 UART1_TX 功能：

```
#define PAN1080_PIN_FUNC_P06_UART1_TX            0x00004000UL
```

   同理可查得当前 `P24` 的 MFP 配置（数字标识 0x00000000）对应的 MFP 功能为 GPIO：

```
#define PAN1080_PIN_FUNC_P24_GPIO                0x00000000UL
```

2. 随后打印切换 pinmux 引脚的提示：

```
switching pinmux..
```

3. 此时在 20s 内将 EVB 板上接在 P06 引脚上的杜邦线拔掉，将其再接到 P24 引脚上，即可看到串口继续打印如下消息：

```
After switch pinmux (loop_cnt = 6):
 - Pin function of P06 is: 0x00000000
 - Pin function of P24 is: 0x00000010
After switch pinmux (loop_cnt = 5):
 - Pin function of P06 is: 0x00000000
 - Pin function of P24 is: 0x00000010
After switch pinmux (loop_cnt = 4):
 - Pin function of P06 is: 0x00000000
 - Pin function of P24 is: 0x00000010
After switch pinmux (loop_cnt = 3):
 - Pin function of P06 is: 0x00000000
 - Pin function of P24 is: 0x00000010
After switch pinmux (loop_cnt = 2):
 - Pin function of P06 is: 0x00000000
 - Pin function of P24 is: 0x00000010
After switch pinmux (loop_cnt = 1):
 - Pin function of P06 is: 0x00000000
 - Pin function of P24 is: 0x00000010
Stopped.
```

   此处可根据前面介绍的方法查询得知：当前 `P06` 的 MFP 配置（数字标识 0x00000000）对应的 MFP 功能为 GPIO：

```
#define PAN1080_PIN_FUNC_P06_GPIO                0x00000000UL
```

   当前 `P24` 的 MFP 配置（数字标识 0x00000010）对应的 MFP 功能为 UART1_TX：

```
#define PAN1080_PIN_FUNC_P24_UART1_TX            0x00000010UL
```

4. 以上现象说明已经成功完成两个 IO 的 MFP 功能切换：
   - 将 `P06` 引脚由 UART1 Tx 功能切换为 GPIO 功能
   - 将 `P24` 引脚由 GPIO 功能切换为 UART1 Tx 功能

5 开发者说明

5.1 使能 PINMUX Driver

确保当前工程开启了如下 config 以使能 Zephyr PINMUX Driver：

```
CONFIG_PINMUX=y
```

> **注**：实际上，我们在 EVB Board Configuration 中已经默认开启了 PINMUX 功能，因此在 App 工程中将其使能已经是非必要操作。

5.2 API 相关头文件位置

1. `zephyr\include\drivers\pinmux.h`：PINMUX API 接口函数声明
2. `zephyr\include\drivers\pinmux\pinmux_panchip.h`：Panchip SoC 通用 PINMUX 宏定义，包括引脚编号、上拉使能/除能、下拉使能/除能、数字输入使能/除能等
3. `zephyr\include\drivers\pinmux\pinmux_pan1080.h`：PAN1080 SoC 所有引脚的 MFP 配置定义

5.3 其他配置 SoC PINMUX 参数的方法

除使用本例程介绍的 PINMUX Driver 以外，PAN1080 还支持如下的方式修改 SoC PINMUX 的相关配置：

- 使用 GPIO Driver API 进行配置（可在运行时动态修改）
- 使用 Devicetree 进行配置（仅在系统初始化时配置一次）

5.3.1 使用 GPIO Driver API 配置 PINMUX 参数

GPIO Driver 的 `gpio_pin_configure()` 接口支持配置 SoC PINMUX 的**上拉电阻通路**、**下拉电阻通路**以及**数字输入通路**。

具体可参考 GPIO Sample 中的介绍与演示。

5.3.2 使用 Devicetree 配置 PINMUX 参数

通过修改 Zephyr Devicetree (`.dts` 文件)，可配置**上拉电阻通路**、**下拉电阻通路**、**数字输入通路**以及 MFP **引脚配置**等 PINMUX 支持的参数。

此方式用于在**系统初始化**的时候配置某个外设的 MFP 引脚功能，如果需要，还可同时配置上拉、下拉或数字输入使能等功能。

例如，对于 PAN1080 EVB 来说，系统初始化的时候会默认将 `P06` 配置为 `UART1_TX` 功能，将 `P07` 配置为 `UART1_RX` 功能，就是通过配置 Devicetree 实现的。此处以 `pan1080a_afld_evb` board 为例说明如何配置：

1. 在 `zephyr\boards\arm\pan1080a_afld_evb\pan1080a_afld_evb.dts` 文件中，配置 UART1 模块的必要参数，并将 PINMUX（即 `pinctrl-0` 属性）配置为 `p0_6_uart1_tx`、 `p0_7_uart1_rx`：

```
...
&uart1 {
        current-speed = <921600>;
        pinctrl-0 = <&p0_6_uart1_tx &p0_7_uart1_rx>;
        status = "okay";
};
...
```

> **注**：关于 `pan1080a_afld` SoC 支持的所有 `pinctrl-0` 属性引脚定义，可以从以下 dts 头文件中查阅：
>
> - `zephyr\dts\arm\panchip\pan1080\pan1080a_afld_pinctrl.dtsi`

2. 配置 MFP 引脚功能后，还需考虑是否要同时配置对应引脚的上拉、下拉或数字输入使能等功能。对于 UART 来说，Rx (`P07`) 是输入引脚，因此需要开启数字输入功能，但无需开启上拉、下拉使能，方法是在 `soc/pin-controller@40030000` 节点中，使用 `DT_PAN_PINS()` API 接口重新配置此引脚的 PINMUX，传入标识（Flag）参数 `input-enable`，以开启 `P07` 引脚的数字输入功能：

```
        soc {
                pin-controller@40030000 {
                        /* port, pin, pinmux_name, pinmux_sel [, flag1, ... ] */
                        ...
                        DT_PAN_PINS(p0, 7, uart1_rx, PAN1080_PIN_FUNC_P07_UART1_RX, input-
↪enable);
                        ...
                };
        };
```

注 1：之所以说是**重新配置** soc/pin-controller@40030000 节点的引脚定义，原因是所有引脚均有其原始的定义（通常是不带 Flag 参数的）；对于 pan1080a_afld SoC 来说，相关定义位于前面已经提到的 dts 头文件中：

- zephyr\dts\arm\panchip\pan1080\pan1080a_afld_pinctrl.dtsi

注 2：pinmux dts 中支持的所有标识（Flag）参数，均可从以下 dts-bindings 文件中查阅：

- zephyr\dts\bindings\pinctrl\panchip,pan-pinctrl.yaml

```
description: |
  Panchip SoC Pinctrl node

  The Panchip SoC pins implements following pin configuration option:

    * bias-pull-up
    * bias-pull-down
    * input-enable

  These options define devicetree flags that are converted to SoC flags at
  PAN_PIN_FLAGS().
```

其中：

- bias-pull-up 表示使能上拉电阻通路；
- bias-pull-down 表示使能下拉电阻通路；
- input-enable 表示使能数字输入通路；

注 3：pinmux dts 中支持的标识（Flag）参数，可以同时配置，且没有顺序限制；例如，使用如下的方式可以同时使能 P07 引脚的数字输入与上拉电阻功能：

```
DT_PAN_PINS(p0, 7, uart1_rx, PAN1080_PIN_FUNC_P07_UART1_RX, input-enable, bias-
↪pull-up);
```

注 4：DT_PAN_PINS() API 接口必须与外设模块中的 pinctrl-0 属性一起使用才会真正生效；如果只使用此接口配置了某个引脚的上拉/下拉/数字输入使能，但没有任何外设的 pinctrl-0 属性中配置了此引脚功能，则当前的 DT_PAN_PINS() 配置不会生效。

5.4 PAN1080 引脚使用注意事项

1. 切换引脚的复用功能时，需要特别注意此引脚是否为**数字输入**引脚，如果是，则一定要显式开启此引脚的数字输入功能，具体可以采样上述介绍的任意一种可行的方式；

   例如：我们需要将某个 IO 配置为 UART Rx 功能，由于 Rx 引脚用于接收外部的数字信号，因此将 IO 的 MFP 功能切换为 UART_RX 后，还应将此 IO 的数字输入功能打开，否则 UART 将无法正确接收对端发来的数据。

2. IO 引脚的**数字输入**功能一般需遵守**非必要不打开**的原则，这样可以减少可能的电流漏入，有助于降低芯片整体功耗；
3. PINMUX Driver 操作的上拉电阻、下拉电阻、数字输入等功能，需要确保已经开启了 GPIO 模块时钟方可正常配置（SDK 中已默认开启）；

Driver: Power Management

1 **功能概述** 本文主要介绍 PAN1080 EVB 板 PM 中进入低功耗前后的回调函数使用说明。

2 环境要求

- board: pan1080a_afld_evb
- uart (option): 显示串口 log

3 编译和烧录 项目位置: zephyr\samples_panchip\drivers\pm

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置: quick_build_samples\drivers\pm.bat。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`，可查看源码，执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明

```
*** Booting Zephyr OS build zephyr-v2.7.0-413-g0af245658bdf  ***
pm entry
pm exit
Wake up from runtime-idle
pm entry
pm exit
Wake up from suspend-to-idle
pm entry
pm exit
Wake up from runtime-idle
pm entry
pm exit
Wake up from suspend-to-idle
pm entry
pm exit
Wake up from runtime-idle
pm entry
```

上电后打印如上数据，该 sample 未和蓝牙 deepsleep 功能绑定，暂时只用于 pm_notifier_register 演示，只会进入 MCU Sleep 模式。

5 开发者说明

进入低功耗的时机

```
&cpu0 {
        cpu-power-states = <&state1>;
};

&state1 {
```

(下页继续)

(续上页)

```
        compatible = "zephyr,power-state";
        power-state-name = "suspend-to-idle";
        min-residency-us = <6000>;
        exit-latency-us = <32>;
};
```

以该参数来说明:

cpu0 只有一种低功耗状态，为 state1:suspend-to-idle

如果 idle 时间满足以下条件:

```
if (time_to_next_scheduled_event >= (state.min_residency_us + state.exit_latency))) {
return state
}
```

则进入相应的 state。

进入低功耗前后的回调函数

```
static void notify_pm_state_entry(enum pm_state state)
{
        printk("pm entry\n");
}

static void notify_pm_state_exit(enum pm_state state)
{
        printk("pm exit\n");
}

static struct pm_notifier notifier = {
        .state_entry = notify_pm_state_entry,        //进入低功耗前的回调函数
        .state_exit = notify_pm_state_exit,                //从低功耗出来后的回调函数
};

/** 注册低功耗回调函数*/
pm_notifier_register(&notifier);

/** 撤销注册低功耗回调函数*/
pm_notifier_unregister(&notifier);
```

Driver: PWM & RGB

1 功能概述 本文主要介绍 PAN1080 EVB 板 RGB 灯控制，定时改变 RGB 灯的颜色，演示外设 PWM 的使用。

2 环境要求

- board: pan1080a_afld_evb
- uart (option): 显示串口 log

3 编译和烧录 项目位置: `zephyr\samples_panchip\drivers\pwm\pwm_rgb`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置: `quick_build_samples\drivers\pwm_rgb.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`, 可查看源码, 执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明

1. PAN1080 EVB 板 GPIO P10、P11、P16 与 RGB 电路用跳线帽连接。
2. EVB 板上电灯的颜色默认是白色，然后灯在红、绿、蓝三种颜色 1s 间隔切换一次。

5 开发说明

5.1 启用 PWM 模块

在 prj.conf 文件中添加 “CONFIG_PWM=y” 启用 pwm 模块。

```
CONFIG_PWM=y
```

5.2 初始 PWM

```
if (!device_is_ready(led_red)) {
    printk("PWM_0 device is not ready\n");
}

if (!device_is_ready(led_green)) {
    printk("PWM_1 device is not ready\n");
}

if (!device_is_ready(led_blue)) {
    printk("PWM_2 device is not ready\n");
}
```

5.3 配置 PWM

在 “pan1080a_afld_evb.overlay” 文件中添加 PWM 参数配置:

```
pwmleds {
    compatible = "pwm-leds";
    pwm_led_red: pwm_led_r {
        pwms = <&pwm0 4 PWM_POLARITY_NORMAL>;
        label = "RGB_LED_RED";
    };
    pwm_led_green: pwm_led_g {
        pwms = <&pwm0 5 PWM_POLARITY_NORMAL>;
        label = "RGB_LED_GREEN";
    };
    pwm_led_blue: pwm_led_b {
        pwms = <&pwm0 6 PWM_POLARITY_NORMAL>;
        label = "RGB_LED_BLUE";
    };
};
```

使用 PWM0 的 4、5、6 通道, 正向极性, 高电平 1 为有效状态 *(active), 低电平 0 为无效状态 (inactive)*。

PWM PIN 脚配置:

```
&pwm0 {
        pinctrl-0 = <&p1_0_pwm0_ch4 &p1_1_pwm0_ch5 &p1_6_pwm0_ch6>;
        status = "okay";
};
```

配置 P10、P11、P16 三个引脚为 PWM 功能。

5.4 PWM 输出 PWM 输出接口

```
/**
 * @brief Set the period and pulse width for a single PWM output.
 *
 * @param dev Pointer to the device structure for the driver instance.
 * @param pwm PWM pin.
 * @param period Period (in microseconds) set to the PWM.
 * @param pulse Pulse width (in microseconds) set to the PWM.
 * @param flags Flags for pin configuration (polarity).
 *
 * @retval 0 If successful.
 * @retval Negative errno code if failure.
 */
static inline int pwm_pin_set_usec(const struct device *dev, uint32_t pwm,
                                   uint32_t period, uint32_t pulse,
                                   pwm_flags_t flags)
{
        uint64_t period_cycles, pulse_cycles, cycles_per_sec;

        if (pwm_get_cycles_per_sec(dev, pwm, &cycles_per_sec) != 0) {
                return -EIO;
        }

        period_cycles = (period * cycles_per_sec) / USEC_PER_SEC;
        if (period_cycles >= ((uint64_t)1 << 32)) {
                return -ENOTSUP;
        }

        pulse_cycles = (pulse * cycles_per_sec) / USEC_PER_SEC;
        if (pulse_cycles >= ((uint64_t)1 << 32)) {
                return -ENOTSUP;
        }

        return pwm_pin_set_cycles(dev, pwm, (uint32_t)period_cycles,
                                  (uint32_t)pulse_cycles, flags);
}
```

配置 PWM 设备（PWM0、PWM1、PWM2）、通道（0~7）、频率、占空比、极性模式。

调用这个接口后，对应的 pin 脚就能输出 PWM 了。

Driver: QDEC & PWM

1 功能概述 本文主要介绍 PAN1080 EVB 板 QDEC 外设模块的使用，检测外部 PWM 信号，触发 QDEC 计数。

2 环境要求

- board: pan1080a_afld_evb
- uart (option): 显示串口 log
- Panchip Serial Assistant V0.0.006.exe

3 编译和烧录

项目位置：zephyr\samples_panchip\drivers\qdec\qdec_pwm

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：quick_build_samples\drivers\qdec_pwm.bat。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`，可查看源码，执行编译下载等
  others exit                        退出
wait input:
```

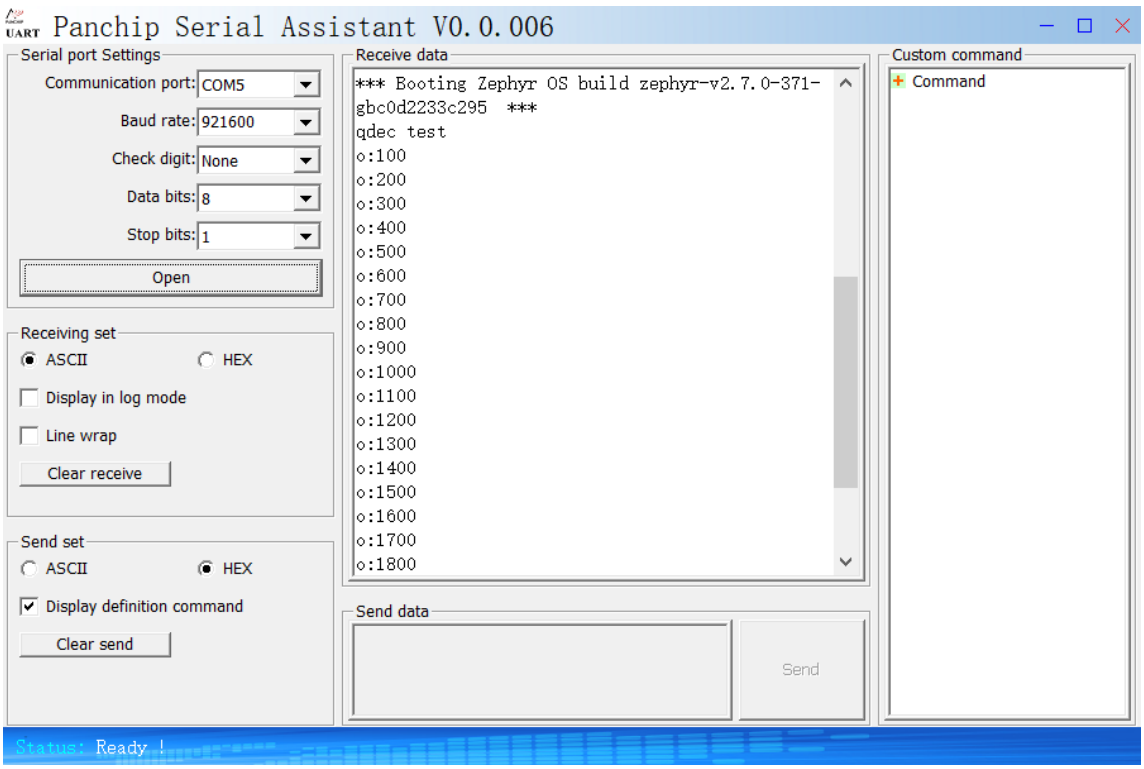
4 演示说明

1. PAN1080 EVB 板 PIN 脚接线说明:

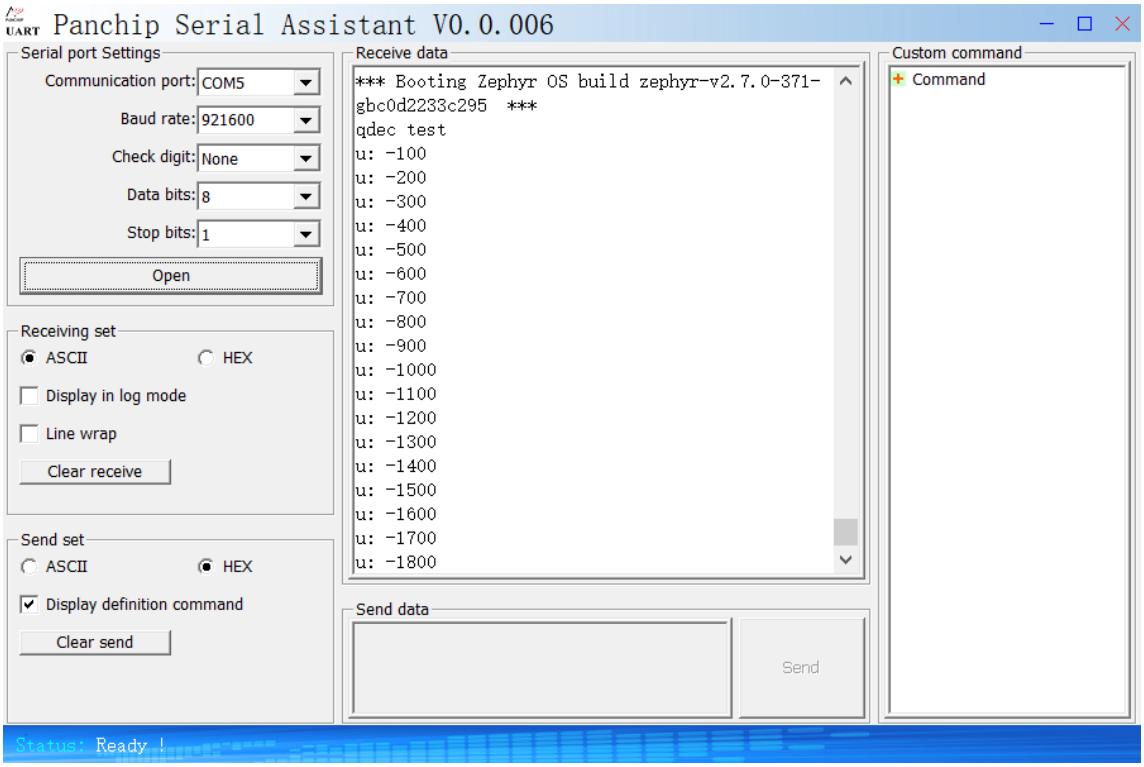
通道	PIN 脚	PWM
通道 X0	P04	P02 PWM CH2
通道 X1	P05	P03 PWM CH3
通道 Y0	P06	P02 PWM CH2
通道 Y1	P07	P03 PWM CH3
通道 Z0	P20	P02 PWM CH2
通道 Z1	P21	P03 PWM CH3

2. 例程中默认使用的是通道 X0、和 X1，向上计数和向下计数的结果如下图所示:

   向上计数:



   向下计数:



5 开发说明

5.1 启用 QDEC 模块

在 prj.conf 文件中添加 “CONFIG_QDEC=y” 启用 qdec 模块。

```
CONFIG_QDEC=y
```

5.2 初始 QDEC

```
const struct device *qdec_dev = DEVICE_DT_GET(DT_NODELABEL(qdec));
```

5.3 配置 QDEC

在 “pan1080a_afld_evb.overlay” 文件中添加 QDEC 参数配置:

```
&qdec {
        event-threshold = <100>;
        polarity = "polarity-low";
        resolution = "cnt-resolution-1x";
        filter-threshold = "filter-threshold-3";
        pinctrl-0 = <&p0_4_qdec_x0 &p0_5_qdec_x1>;
        status = "okay";
};
```

Config Type	Description
event-threshold	设置 qdec 事件触发阈值（此处是 100）
polarity	设置 qdec 电平极性，可设置为高或者低电平有效（此处是低电平有效）
resolution	设置 qdec 分辨率，可设置为 1x、2x、4x（此处是 1x）
filter-threshold	设置 qdec 短脉冲过滤，设置范围为 0~7（此处是 3）

QDEC PIN 脚配置:

```
pinctrl-0 = <&p0_4_qdec_x0 &p0_5_qdec_x1>;
```

配置 X0、X1 两个通道。

同理配置 Y0、Y1 和 Z0、Z1 两个通道

```
pinctrl-0 = <&p0_4_qdec_x0 &p0_5_qdec_x1 &p0_6_qdec_y0 &p0_7_qdec_y1 &p2_0_qdec_z0 &p2_1_qdec_
↪z1>;
```

5.4 QDEC 中断配置使能

通道配置:

```
uint8_t channel = QDEC_CNT_IDX_X;

qdec_clear_event_cnt(qdec_dev, channel);
```

代码默认配置通道 X，也可配置成 QDEC_CNT_IDX_Y、QDEC_CNT_IDX_Z。

```
qdec_irq_add_cb(qdec_dev, qdec_irq_cb);

qdec_clear_int_msk(qdec_dev, ENABLE, QDEC_INT_CNT_OVERFLOW_Msk | QDEC_INT_CNT_UNDERFLOW_Msk);
qdec_clear_event_cnt(qdec_dev, channel);
qdec_clear_int_flag(qdec_dev, QDEC_FUNC_ALL);
qdec_clear_raw_int_flag(qdec_dev, QDEC_FUNC_ALL);

qdec_set_enable(qdec_dev, ENABLE, QDEC_EN_Msk | QDEC_FILTER_EN_Msk |  (QDEC_CHANNEL_X_EN_Msk <
↪< channel) | QDEC_EVENT_EN_Msk);
```

注册中断回调函数 “qdec_irq_cb”，配置通道 X、向上计数和向下计数中断，清中断标志位。

Driver: SPI Master

1 功能概述 该 sample 演示了 SPI 作为 Master 对开发板上的 SRAM 进行读写操作。

2 环境要求

- PAN1080 EVB 一块
- Micro USB 线一条（用于供电和查看串口打印 Log）
- 硬件接线:
  - 使用 USB 线，将 PC USB 与 EVB MicroUSB（USB->UART）相连
  - 使用**杜邦线**或**跳线帽**将 EVB 上的:
    - UART1 TX 与 P06 相连
    - UART1 RX 与 P07 相连
- PC 软件: 串口调试助手（UartAssist）或终端工具（SecureCRT），波特率 921600

3 编译和烧录 例程位置: `zephyr\samples_panchip\drivers\spi_master`

目前可使用 ZAL 工具或 quick build **脚本**进行编译和下载。

脚本位置: `quick_build_samples\drivers\spi_master.bat`

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`，可查看源码，执行编译下载等
```

(下页继续)

(续上页)

```
  others exit                          退出
wait input:
```

4 演示说明

- 连接 SPI 引脚到 SRAM 模块（开发板上进行跳线）。

  == **注：请注意开发板上引脚共用问题，不要将下面四个引脚跳线至其他模块** ==

引脚	功能
P02	CS
P03	SCK
P30	MOSI
P31	MISO

- Sample 执行流程：
  1. 写一字节数据【0xAE】到 0x00 地址
  2. 从 0x00 地址读一字节数据
  3. 写 32 字节数据到 0x0004 地址
  4. 从 0x0004 地址读 32 字节数据并和写入数据进行比较，验证读取是否成功
- 程序正确执行时，日志界面如下（日志中包含 Read 字样的行表示从 Slave 读取到的数据）：

```
*** Booting Zephyr OS build zephyr-v2.7.0-386-gff31e1c4b5e6  ***
SRAM example application
Wrote 0xAE to address 0x00.
Read 0xAE from address 0x00.
Wrote 32 bytes to address 0x0004.
Read 32 bytes from address 0x0004.
Data comparison successful.
```

5 开发说明

5.1 启用 SPI 模块

在 prj.conf 文件中添加 “CONFIG_SPI=y” 启用 SPI 模块。

```
CONFIG_SPI=y
```

5.2 初始化 SPI

```
const struct device *spi = DEVICE_DT_GET(DT_NODELABEL(spi0));
if (!device_is_ready(spi)) {
    printk("SPI device %s is not ready\n", spi->name);
    return;
}
```

5.3 SPI 配置

可根据实际情况配置即可，需要注意的是参数 frequency 指的是时钟的分频系数，并非实际的时钟数。例如当前系统外设时钟为 16MHz，frequency 配置为 4，则 SPI 的时钟速率为 16/4=4MHz。

5.4 写操作

实现向 slave 写数据

```
struct spi_buf bufs[] = {
    {
        .buf = data,
        .len = len
    },
};
struct spi_buf_set tx = {
    .buffers = bufs,
    .count = 1
};

/**
 * @brief Write the specified amount of data from the SPI driver.
 *
 * @note This function is synchronous.
 *
 * @note This function is an helper function calling spi_transceive.
 *
 * @param dev Pointer to the device structure for the driver instance
 * @param config Pointer to a valid spi_config structure instance.
 *        Pointer-comparison may be used to detect changes from
 *        previous operations.
 * @param tx_bufs Buffer array where data to be sent originates from.
 *
 * @retval 0 If successful.
 * @retval -errno Negative errno code on failure.
 */
spi_write(spi, spi_cfg, &tx);
```

5.5 读操作　实现从 slave 读取数据

```
struct spi_buf bufs[] = {
    {
        .buf = data,
        .len = len
    },
};
struct spi_buf_set rx = {
    .buffers = bufs,
    .count = 1
};

/**
 * @brief Read the specified amount of data from the SPI driver.
 *
 * @note This function is synchronous.
 *
 * @note This function is an helper function calling spi_transceive.
 *
 * @param dev Pointer to the device structure for the driver instance
 * @param config Pointer to a valid spi_config structure instance.
 *        Pointer-comparison may be used to detect changes from
 *        previous operations.
 * @param rx_bufs Buffer array where data to be read will be written to.
 *
 * @retval 0 If successful.
 * @retval -errno Negative errno code on failure.
 */
spi_read(spi, spi_cfg, &rx);
```

5.6 **读写操作** 实现向 slave 发送数据的同时接收来自 slave 的数据

```
struct spi_buf bufs_tx[] = {
                {
                                .buf = data,
                                .len = len
                },
};

struct spi_buf bufs_rx[] = {
                {
                                .buf = data,
                                .len = len
                },
};

struct spi_buf_set tx = {
                .buffers = bufs_tx,
                .count = 1
};

struct spi_buf_set rx = {
                .buffers = bufs_rx,
                .count = 1
};

/**
 * @brief Read/write the specified amount of data from the SPI driver.
 *
 * @note This function is synchronous.
 *
 * @param dev Pointer to the device structure for the driver instance
 * @param config Pointer to a valid spi_config structure instance.
 *        Pointer-comparison may be used to detect changes from
 *        previous operations.
 * @param tx_bufs Buffer array where data to be sent originates from,
 *        or NULL if none.
 * @param rx_bufs Buffer array where data to be read will be written to,
 *        or NULL if none.
 *
 * @retval frames Positive number of frames received in slave mode.
 * @retval 0 If successful in master mode.
 * @retval -errno Negative errno code on failure.
 */
spi_transceive(spi, spi_cfg, &tx, &rx);
```

Driver: UART FIFO

1 **功能概述** 本文主要介绍 PAN1080 EVB 板外设 UART 的使用。

2 **环境要求**

- board: pan1080a_afld_evb
- uart (option): 显示串口 log
- uart:USB 转串口工具
- Panchip Serial Assistant V0.0.006.exe

3 **编译和烧录** 项目位置: `zephyr\samples_panchip\drivers\uart\uart_fifo`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\drivers\uart_fifo.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

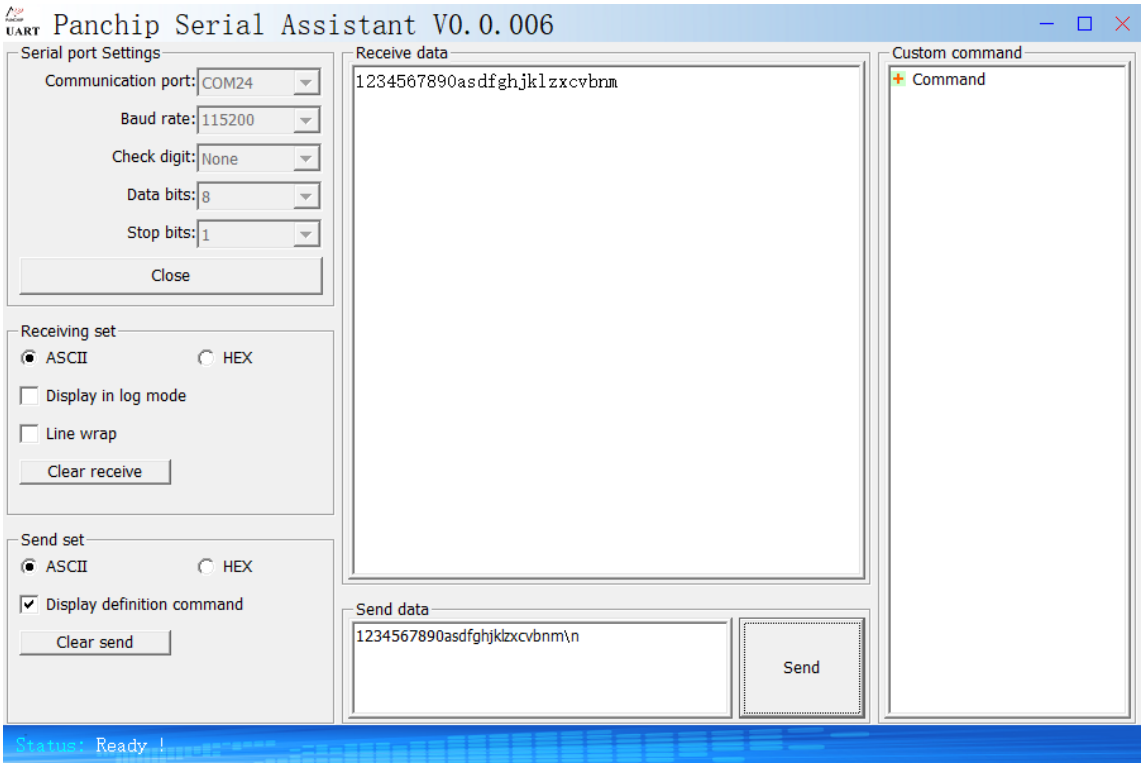
```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明

1. PAN1080 EVB 板 PIN 脚接线说明:

PAN1080 EVB 板 GPIO	USB 转串口工具
P0_0_uart0_tx	UART_RX
P0_1_uart0_rx	UART_TX

2. 串口上位机发送”1234567890asdfghjklzxcvbnm\n”，可以看到接收界面也显示”1234567890asdfghjklzxcvbnm”。本例程主要演示 PAN1080 EVB 板串口的收发功能。



5 开发说明

5.1 启用 UART 模块

在 prj.conf 文件中添加 “CONFIG_UART=y” 启用 uart 模块。

```
CONFIG_UART=y
```

5.2 初始 UART

```
const struct device *uart0 = DEVICE_DT_GET(DT_NODELABEL(uart0));

if (!device_is_ready(uart0)) {
    printk("uart devices not ready\n");
    return;
}
```

5.3 配置 UART

```
#define BAUDRATE                        115200

const struct uart_config uart_cfg = {
        .baudrate = BAUDRATE,
        .parity = UART_CFG_PARITY_NONE,
        .stop_bits = UART_CFG_STOP_BITS_1,
        .data_bits = UART_CFG_DATA_BITS_8,
        .flow_ctrl = UART_CFG_FLOW_CTRL_NONE
};
```

配置串口的波特率、校验位、停止位、数据位和流控。

- 串口配置读取与设置:

```
uart_config_get(uart0, &uart_cfg_check);
if(uart_cfg_check.baudrate != BAUDRATE) {
    uart_configure(uart0, &uart_cfg);
}
```

- UART PIN 脚配置:

```
&uart0 {
        current-speed = <115200>;
        pinctrl-0 = <&p4_4_uart0_tx &p4_3_uart0_rx>;
        status = "okay";
};
```

配置 P43 和 P44 两个引脚为 UART0 功能。

5.4 UART 中断回调和使能

- 中断回调注册:

```
uart_irq_callback_set(uart0, uart_fifo_callback);
```

- 使能 UART TRX:

```
uart_irq_tx_enable(uart0);
uart_irq_rx_enable(uart0);
```

- TX 的逻辑处理:

```
static void tx_isr(const struct device *dev)
{
        static uint8_t tx_index = 0;
        int len;

        if(tx_index >= data_len) {
                uart_irq_tx_disable(dev);
                uart_irq_rx_enable(dev);
                tx_index = 0;
```

(下页继续)

(续上页)

```
        } else {
                len = uart_fifo_fill(dev, fifo_data + tx_index, (data_len - tx_index));
                tx_index += len;
                if(tx_index >= BUFFER_SIze) {
                        tx_index = 0;
                }
        }
}
```

将 tx 的数据写到 fifo 中，直到所有的数据都写完，然后关闭 tx，使能 rx，等待数据接收。fifo 的最大深度为 16。

- RX 的逻辑处理：

```
static void rx_isr(const struct device *dev)
{
        static uint8_t rx_index = 0;

        uart_fifo_read(dev, fifo_data + rx_index, 1);
        if ((fifo_data[rx_index] == '\n') || (fifo_data[rx_index] == '\r')) {
                uart_irq_rx_disable(dev);
                data_len = rx_index + 1;
                rx_index = 0;
                uart_irq_tx_enable(dev);
        } else {
                rx_index ++;
                if(rx_index >= BUFFER_SIze) {
                        rx_index = 0;
                }
        }
}
```

将串口工具发过来的数据保存到 “fifo_data”，每次接收一个数据，直到出现换行符，停止接收。然后将接收的数据原封不动的发送给串口工具。

3.2.4 私有 2.4G 例程

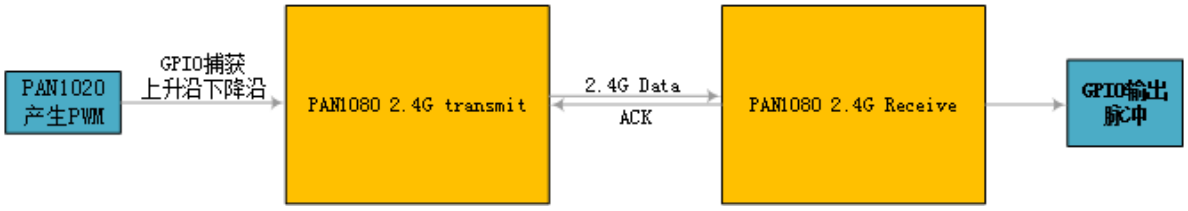
PRF: IO Pulse Receiver

重要: 此例程仅存在于特殊版本的 SDK 中，如有需要请联系 Panchip。

1 功能概述 此项目演示了无线脉冲透传-接收端功能：接收发送端的 2.4G 信号，恢复出波形并通过 IO 口输出。

发射端（参考prf_io_pulse_tx）捕获上升沿和下降沿，在上升沿来到时，发个脉冲包，在下降沿来到时，再发个脉冲包。接收端收到脉冲包后控制 GPIO 输出恢复原始脉冲。

系统框图：



2 环境要求

- board: pan1080a_afld_evb
- uart (option): 显示串口 log
- 波形分析工具：逻辑分析仪或示波器。

  需要搭配一个运行 `prf_io_pulse_tx` 的板子一起使用。

3 编译和烧录

项目位置：`zephyr\samples_panchip\proprietary_radio\prf_io_pulse_rx`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\proprietary_radio\prf_io_pulse_rx.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`，可查看源码，执行编译下载等
  others exit                        退出
wait input:
```

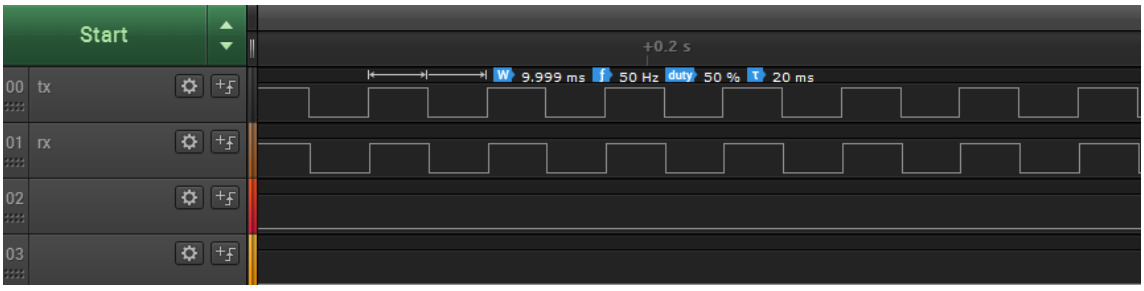
4 演示说明

1. 将 P2_1 接到逻辑分析仪或示波器上。
2. 配置发送端（参考prf_io_pulse_tx）。
3. 在逻辑分析仪上检查输出波形。

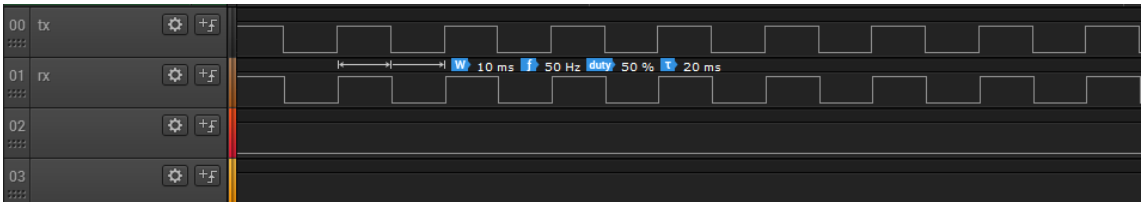
4.1 演示结果

1. PAN1020 PWM 为 50hz，占空比为 50%，输出脉冲的宽度为 10ms。发射端和接收端的波形如下:

   发射端:



   接收端:

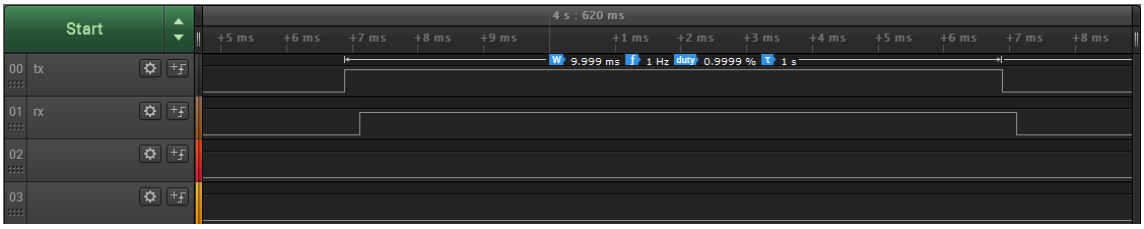


   发射端和接收端的脉冲宽度相差 1us。

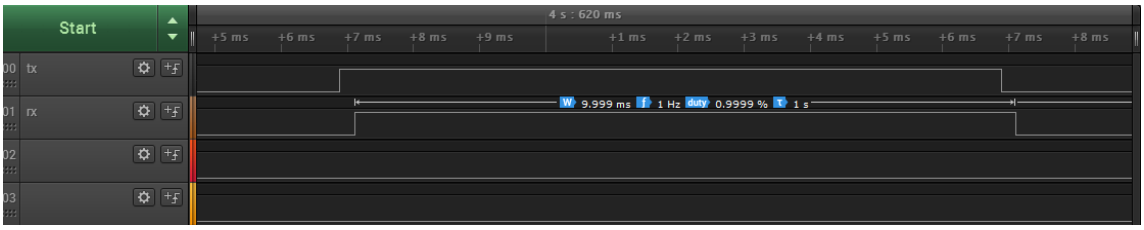
发射端和接收端的脉冲延迟 220us。

2. PAN1020 PWM 为 1hz，占空比为 1%，输出脉冲的宽度为 10ms。发射端和接收端的波形如下：

   发射端：



   接收端：



   发射端和接收端的脉冲宽度无误差。

   发射端和接收端的脉冲延迟 220us。

5 性能指标

- 支持带外传输：脉冲包通过私有 2.4G 传输，传输频点为 2360~2510Mhz 中的非标频点。
- 一个脉冲的长度，约 10ms - 80ms，脉冲的频率 1hz~50hz。还原脉冲的精度误差小于 1us。从发射端捕获脉冲到接收端还原脉冲的延迟小于 10ms。
- 为了确保传输过程中不丢包必须要用 2.4G 增强型模式，收发转换满足 150us。

6 开发说明

6.1 GPIO 初始化

```
void GPIO_Init(void)
{
        port = device_get_binding(DT_LABEL(DT_NODELABEL(p2)));

        gpio_pin_configure(port, 1, GPIO_OUTPUT);
        gpio_pin_configure(port, 3, GPIO_OUTPUT);
}
```

6.2 2.4G 接收初始化　接收频点为带外频点 2566Mhz

```
    pan_prf_config_t rx_config = {
        .work_mode                          = PRF_MODE_NORMAL,
        .chip_mode                          = PRF_CHIP_MODE_SEL_XN297,
        .trx_mode                          = PRF_RX_MODE,
        .phy                                  = PRF_PHY_1M,
        .crc                                  = PRF_CRC_SEL_CRC8,
        .src                                  = PRF_SRC_SEL_NOSRC,
        .rx_timeout                          = DISABLE,
        .rf_channel                          = 2566,
        .tx_no_ack                          = DISABLE,
        .nrf52_mode                          = DISABLE,
        .rx_length                          = 4,
        .addr_length                 = 5,
        .addr                                     = { 0x11, 0x22, 0x33, 0x44, 0x55 },
};
```

2.4G 初始化配置说明如下：

初始化配置的结构体“pan_prf_config_t”

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，no crc
prf_scramble_sel_t	src	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，配置范围 2402-2480Mhz
uint8_t	tx_no_ack	配置增强型模式下 tx 是否需要 ack
uint8_t	nrf52_mode	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	addr_length	接入地址长度配置，可配置为 3、4、5 字节
uint8_t	addr[5]	接入地址的内容

prf_mode_t：

Type	Value	Description
PRF_MODE_NORMAL	0	普通型
PRF_MODE_ENHANCE	1	增强型

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_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_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

6.3 2.4G 接收中断处理

```
void event_rx_fun(void)
{
        static uint8_t flag;
        panchip_prf_payload_t rx_payload;

        rx_payload.data_length = panchip_prf_data_rec(&rx_payload);
        printk("rx data:");
        data_printk(rx_payload.data, rx_payload.data_length);

        flag ^= 1;
        gpio_pin_set(port, 1, (int)flag);                                   /* P2_1 */

        if (rx_config.work_mode == PRF_MODE_ENHANCE) {
                static panchip_prf_payload_t tx_payload = {
                        .data_length = 10,
                        .data = { 0x10, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa },
                };

                tx_payload.data[0]++;
                panchip_prf_set_ack_data(&tx_payload);
        } else {
                panchip_prf_trx_start();
        }
}
```

注：此例程对时间要求较高，因此在 GPIO 中断和 LL 中断中不宜添加打印或者执行时间长的代码。

PRF: IO Pulse Transmitter

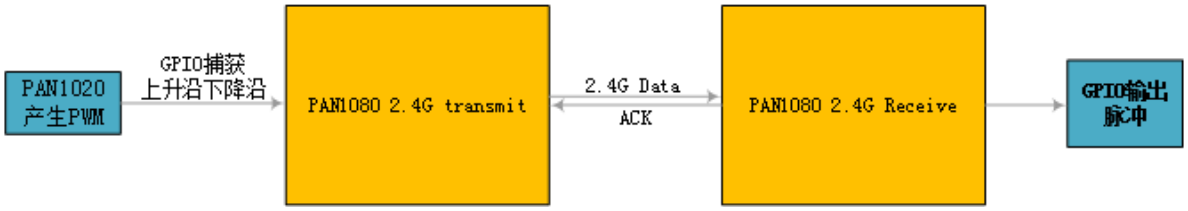
重要: 此例程仅存在于特殊版本的 SDK 中，如有需要请联系 Panchip。

---

1 **功能概述** 此项目演示了无线脉冲透传-发送端功能：通过 IO 口接收将外部的 PWM 波形，并通过 2.4G 传输给接收端设备。

发射端捕获上升沿和下降沿，在上升沿来到时，发个脉冲包，在下降沿来到时，再发个脉冲包。接收端（参考prf_io_pulse_rx）收到脉冲包后控制 GPIO 输出恢复原始脉冲。

系统框图：



2 环境要求

- board: pan1080a_afld_evb
- uart (option): 显示串口 log
- 脉冲输出: PAN1020 EVB 或者手动调整 IO 输入。
- 波形分析工具：逻辑分析仪或示波器。

  需要搭配一个运行 `prf_io_pulse_rx` 的板子一起使用。

3 编译和烧录

项目位置：`zephyr\samples_panchip\proprietary_radio\prf_io_pulse_tx`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\proprietary_radio\prf_io_pulse_tx.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`，可查看源码，执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明

1. 将” samples_panchip\proprietary_radio\prf_io_pulse_tx\PAN1020_PWM.hex “通过 j-link 烧录到 PAN1020 EVB 板。

   如果使用 PAN1020 EVB 作为脉冲产生器，需要一块 PAN1020 开发板，烧录 `PAN1020_PWM.hex`。这个例程会有两个输出：

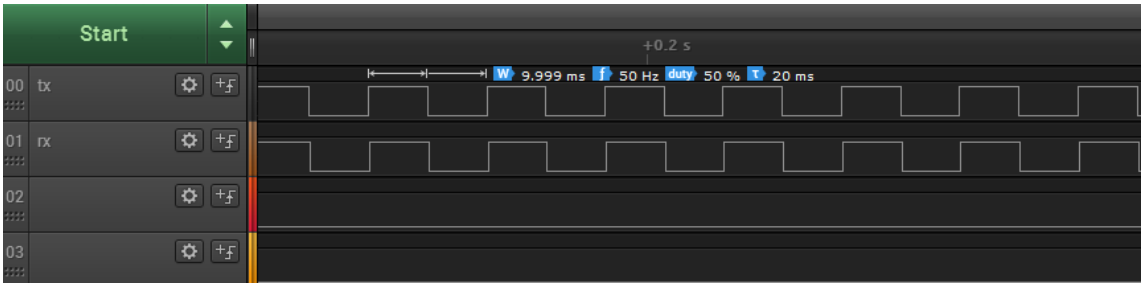
   - PWM FREQ 50Hz: P2_3
   - PWM FREQ 1Hz: P2_4

2. PAN1080 EVB 板发射端的 GPIO 捕获端口 P2_3 和 PAN1020 EVB 板输出端口连接 P2_3 或 P2_4，同时将捕获端口接到逻辑分析仪或示波器上。
3. 配置接收端（参考prf_io_pulse_rx）。
4. 在逻辑分析仪上检查输出波形。

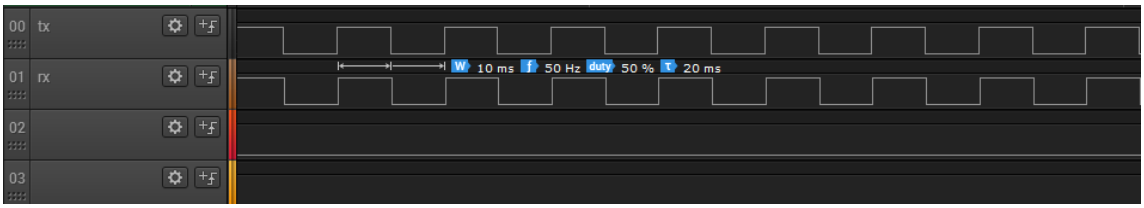
4.1 演示结果

1. PAN1020 PWM 为 50hz，占空比为 50%，输出脉冲的宽度为 10ms。发射端和接收端的波形如下：

   发射端：



接收端:

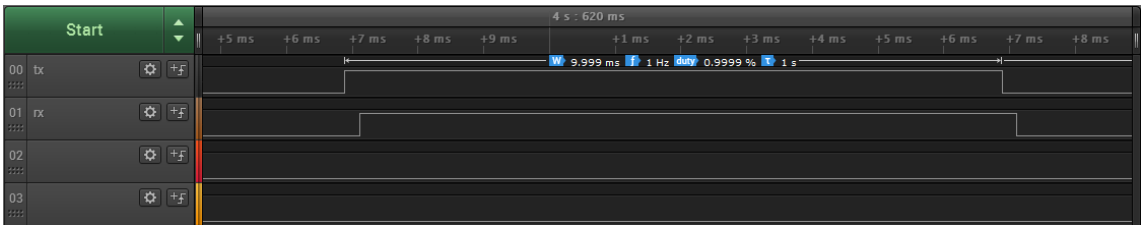


发射端和接收端的脉冲宽度相差 1us。

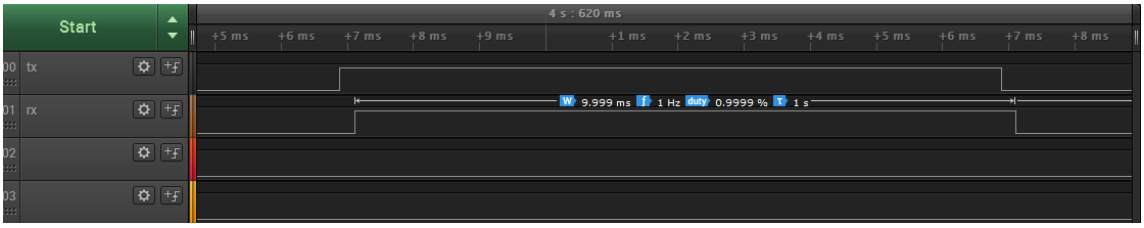
发射端和接收端的脉冲延迟 220us。

2. PAN1020 PWM 为 1hz，占空比为 1%，输出脉冲的宽度为 10ms。发射端和接收端的波形如下:

发射端:



接收端:



发射端和接收端的脉冲宽度无误差。

发射端和接收端的脉冲延迟 220us。

5 性能指标

- 支持带外传输：脉冲包通过私有 2.4G 传输，传输频点为 2360~2510Mhz 中的非标频点。
- 一个脉冲的长度，约 10ms - 80ms，脉冲的频率 1hz~50hz。还原脉冲的精度误差小于 1us。从发射端捕获脉冲到接收端还原脉冲的延迟小于 10ms。
- 为了确保传输过程中不丢包必须要用 2.4G 增强型模式，收发转换满足 150us。

6 开发说明 发射端 GPIO 捕获上升沿和下降沿，上升沿和下降沿都会产生中断，每来一次中断发送一包 2.4G 数据。

6.1 GPIO 初始化

```
static void callback_edge(const struct device *port, struct gpio_callback *cb,
                          gpio_port_pins_t pins)
{
        panchip_prf_payload_t tx_payload = {
                .data_length                = 4,
                .data                                = { 0x01, 0x02, 0x03, 0x04, },
        };

        if (pins == BIT(2)) {
        }

        if (pins == BIT(3)) {
                panchip_prf_set_data(&tx_payload);
                panchip_prf_trx_start();
        }
}

void GPIO_Init(void)
{
        const struct device *port;

        port = device_get_binding(DT_LABEL(DT_NODELABEL(p2)));

        gpio_pin_configure(port, 2, GPIO_INPUT);
        gpio_pin_configure(port, 3, GPIO_INPUT);

        gpio_init_callback(&gpio_cb, callback_edge, BIT(2) | BIT(3));
        gpio_add_callback(port, &gpio_cb);

        gpio_pin_interrupt_configure(port, 2, GPIO_INT_EDGE_BOTH);
        gpio_pin_interrupt_configure(port, 3, GPIO_INT_EDGE_BOTH);
}
```

6.2 2.4G 发射初始化

发射频点为带外频点 2566Mhz：

```
pan_prf_config_t tx_config = {
                .work_mode                        = PRF_MODE_NORMAL,
                .chip_mode                        = PRF_CHIP_MODE_SEL_XN297,
                .trx_mode                        = PRF_TX_MODE,
                .phy                                = PRF_PHY_1M,
                .crc                                = PRF_CRC_SEL_CRC8,
                .src                                = PRF_SRC_SEL_NOSRC,
                .rx_timeout                        = 50000,
                .rf_channel                        = 2566,
                .tx_no_ack                        = DISABLE,
                .nrf52_mode                        = DISABLE,
                .rx_length                        = 0,
```

(下页继续)

(续上页)

```
        .addr_length                = 5,
        .addr                               = { 0x11, 0x22, 0x33, 0x44, 0x55 },
};
```

2.4G 初始化配置说明如下：

初始化配置的结构体“pan_prf_config_t”

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，no crc
prf_scramble_sel_t	src	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，配置范围 2402-2480Mhz
uint8_t	tx_no_ack	配置增强型模式下 tx 是否需要 ack
uint8_t	nrf52_mode	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	addr_length	接入地址长度配置，可配置为 3、4、5 字节
uint8_t	addr[5]	接入地址的内容

prf_mode_t：

Type	Value	Description
PRF_MODE_NORMAL	0	普通型
PRF_MODE_ENHANCE	1	增强型

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_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_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

6.3 2.4G 各中断处理

```
void event_tx_fun(void)
{
        printk("tx done\n");
}

void event_rx_fun(void)
{
        panchip_prf_payload_t rx_payload;

        rx_payload.data_length = panchip_prf_data_rec(&rx_payload);
        printk("rx data:");
        data_printk(rx_payload.data, rx_payload.data_length);
}

void event_rx_timeout_fun(void)
{
        printk("rx timeout\n");
}

void event_crc_err_fun(void)
{
        printk("rx data crc err\n");
}

void event_pid_err_fun(void)
{

}
```

注：此例程对时间要求较高，因此在 GPIO 中断和 LL 中断中不宜添加打印或者执行时间长的代码。

PRF: 2.4G Receiver

1 **功能概述** 此项目演示了 2.4G 接收端功能：接收发送端的 2.4G 信号，并将接收到的数据通过串口打印出来。

发射端（参考prf_sample_tx）每隔 1s 发送一次 2.4G 数据包，长度 5 个字节。

2 **环境要求**

- board: pan1080a_afld_evb
- uart: 显示串口输出 log
- PC 串口工具：Panchip Serial Assistant V0.0.006.exe

  需要搭配一个运行 `prf_sample_tx` 的板子一起使用。

3 **编译和烧录** 项目位置：`zephyr\samples_panchip\proprietary_radio\prf_sample_rx`。

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：

- xn297 mode: `quick_build_samples\proprietary_radio\prf_sample_rx_xn297.bat`
- 24l01 mode: `quick_build_samples\proprietary_radio\prf_sample_rx_24l01.bat`
- nrf52 mode: `quick_build_samples\proprietary_radio\prf_sample_rx_nrf52.bat`

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

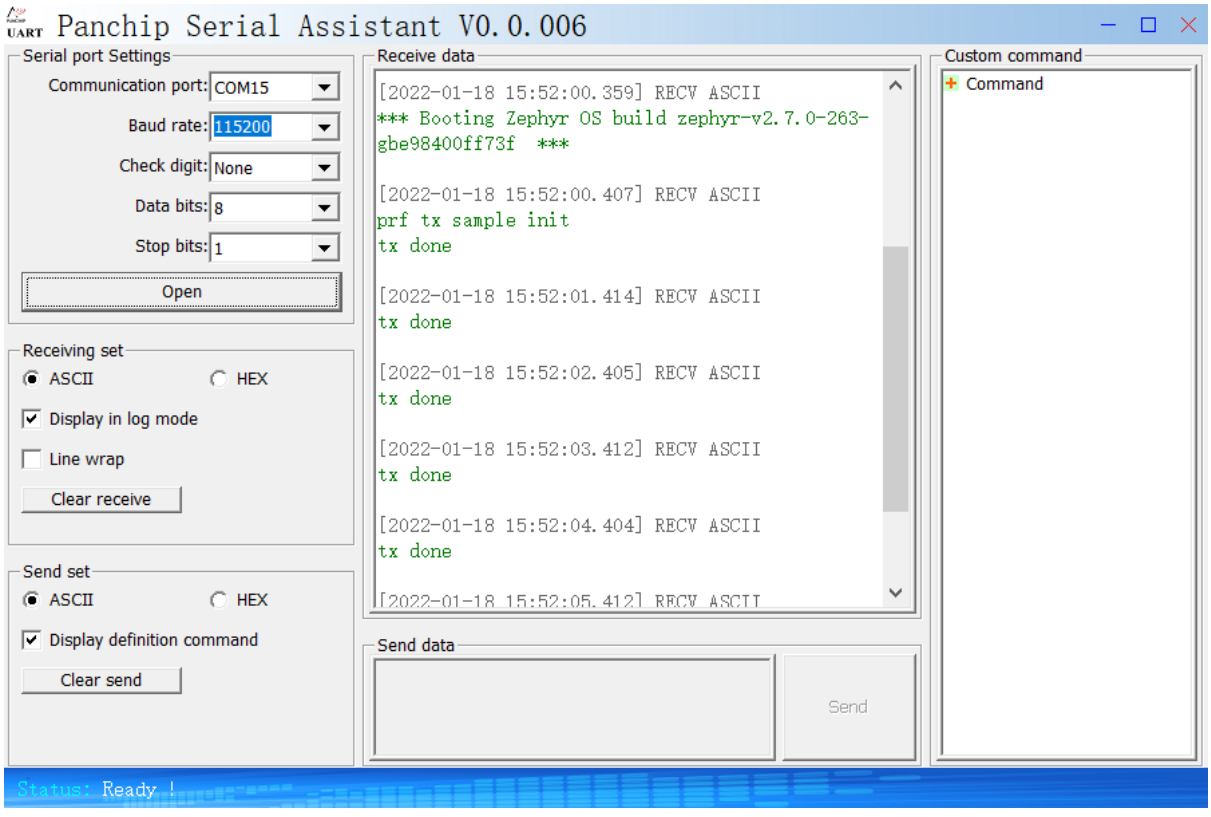
```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`，可查看源码，执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明

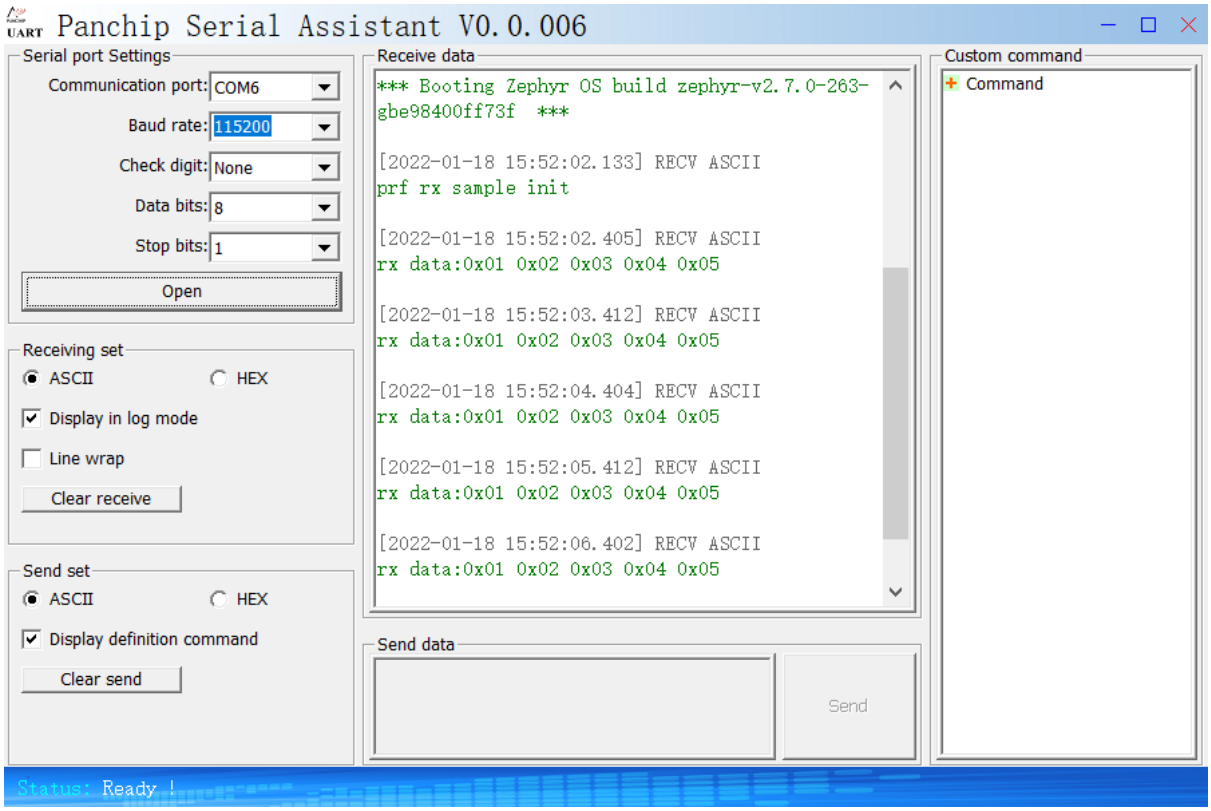
1. 将接收端串口和发射端串口分别接到 PC 的 USB 端口上。
2. 配置接收端和发送端（参考prf_sample_tx）。
3. 观察 PC 串口工具的输出结果。

4.1 演示结果

1. 发射端输出结果:



1. 接收端输出结果:



5 开发说明

5.1 2.4G 接收初始化　设置接收频点 2450Mhz

```
pan_prf_config_t rx_config = {
        .work_mode                      = CONFIG_PRF_WORK_MODE,
        .chip_mode                      = CONFIG_PRF_CHIP_MODE,
        .trx_mode                      = PRF_RX_MODE,
        .phy                              = PRF_PHY_1M,
        .crc                              = CONFIG_PRF_CRC_MODE,
        .src                              = CONFIG_PRF_SRC_MODE,
        .rx_timeout                     = DISABLE,
        .rf_channel                     = 2450,
        .tx_no_ack                     = DISABLE,
        .nrf52_mode                     = CONFIG_PRF_NRF52_MODE,
        .rx_length                     = 5,
        .addr_length                = 4,
        .addr                                 = { 0x71, 0x76, 0x41, 0x76 },
};
```

2.4G 初始化配置说明如下：

初始化配置的结构体 “pan_prf_config_t”

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，no crc
prf_scramble_sel_t	src	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，配置范围 2402-2480Mhz
uint8_t	tx_no_ack	配置增强型模式下 tx 是否需要 ack
uint8_t	nrf52_mode	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	addr_length	接入地址长度配置，可配置为 3、4、5 字节
uint8_t	addr[5]	接入地址的内容

prf_mode_t:

Type	Value	Description
PRF_MODE_NORMAL	0	普通型
PRF_MODE_ENHANCE	1	增强型

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_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_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

5.2 中断处理

```
void event_rx_fun(void)
{
        panchip_prf_payload_t rx_payload;

        rx_payload.data_length = panchip_prf_data_rec(&rx_payload);
        printk("rx data:");
        data_printk(rx_payload.data, rx_payload.data_length);

        if (rx_config.work_mode == PRF_MODE_ENHANCE) {
                static panchip_prf_payload_t tx_payload = {
                        .data_length = 10,
                        .data = { 0x10, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa },
                };

                tx_payload.data[0]++;
                panchip_prf_set_ack_data(&tx_payload);
        } else {
                panchip_prf_trx_start();
        }
}
```

- 普通型模式下将接收的数据打印出来并启动下一次接收。
- 增强型模式下将接收的数据打印出来并发送 ack 数据。

Note：2.4G **接收只支持一个通道，也就是点对点通信。如果要支持一对多通信需要遍历初始化接收端的接入的地址**。

PRF: 2.4G Transmitter

1 功能概述 此项目演示了 2.4G 发射端功能：每隔 1s 发送一次 2.4G 数据包，长度 5 个字节。

接收端（参考prf_sample_rx）接收发送端的 2.4G 信号，并将接收到的数据通过串口打印出来。

2 环境要求

- board: pan1080a_afld_evb
- uart: 显示串口输出 log
- PC 串口工具：Panchip Serial Assistant V0.0.006.exe

  需要搭配一个运行 `prf_sample_rx` 的板子一起使用。

3 编译和烧录 项目位置：`zephyr\samples_panchip\proprietary_radio\prf_sample_tx`。

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：

- xn297 mode: `quick_build_samples\proprietary_radio\prf_sample_tx_xn297.bat`
- 24l01 mode: `quick_build_samples\proprietary_radio\prf_sample_tx_24l01.bat`
- nrf52 mode: `quick_build_samples\proprietary_radio\prf_sample_tx_nrf52.bat`

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
```

(下页继续)

(续上页)

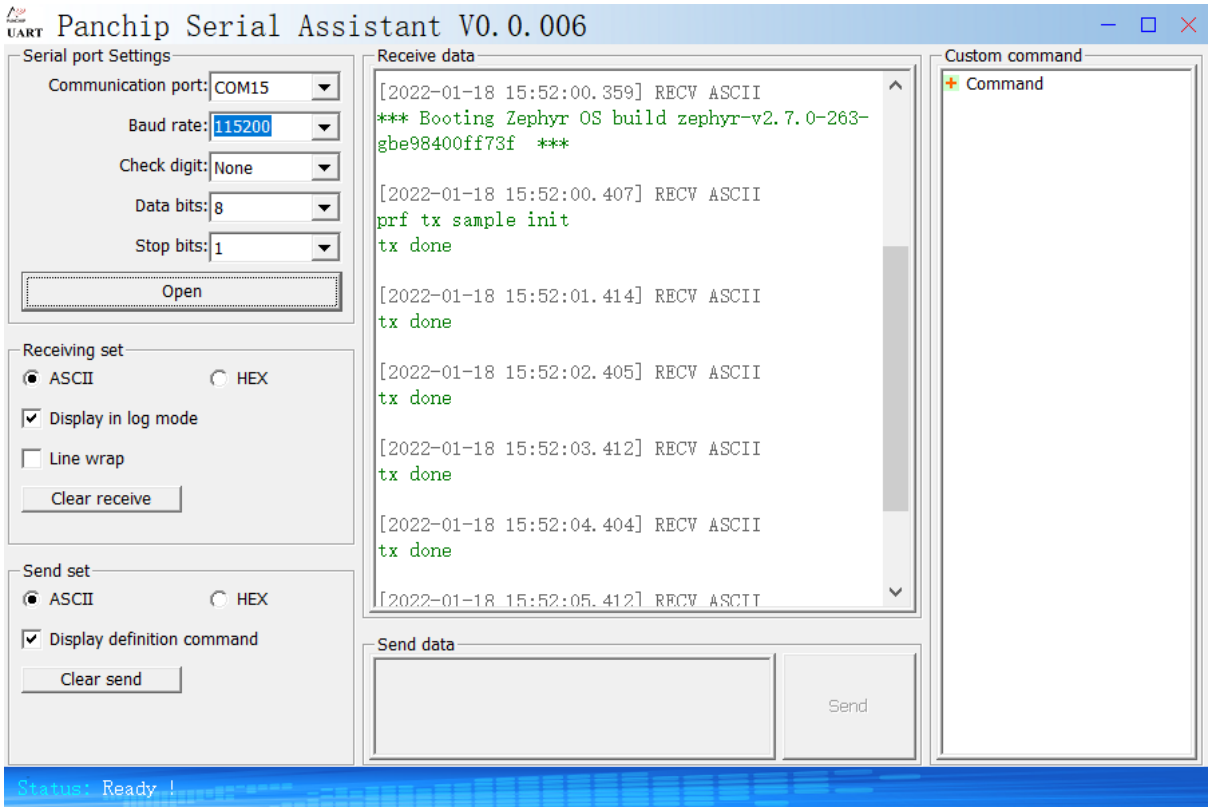
```
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`, 可查看源码, 执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明

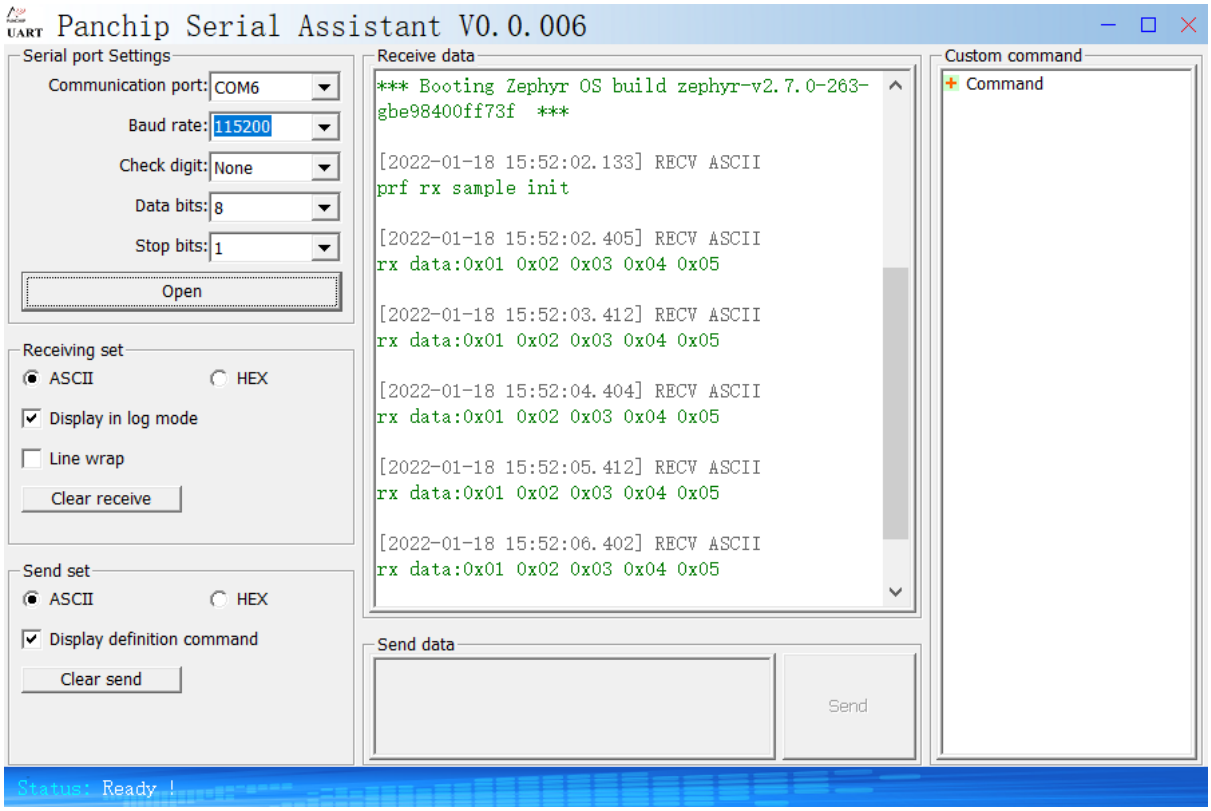
1. 将接收端串口和发射端串口分别接到 PC 的 USB 端口上。
2. 配置发送端和接收端（参考prf_sample_rx）。
3. 观察 PC 串口工具的输出结果。

4.1 演示结果

1. 发射端输出结果：



1. 接收端输出结果：



5 **开发说明** 发射端启动了一个 1s 的定时器，每隔 1s 发送一包 2.4G 数据。发送成功串口输出 log" tx done"。

5.1 GPIO 初始化

```
static void callback_edge(const struct device *port, struct gpio_callback *cb,
                          gpio_port_pins_t pins)
{
        panchip_prf_payload_t tx_payload = {
                .data_length                = 4,
                .data                                   = { 0x01, 0x02, 0x03, 0x04, },
        };

        if (pins == BIT(2)) {
        }

        if (pins == BIT(3)) {
                panchip_prf_set_data(&tx_payload);
                panchip_prf_trx_start();
        }
}

void GPIO_Init(void)
{
        const struct device *port;

        port = device_get_binding(DT_LABEL(DT_NODELABEL(p2)));

        gpio_pin_configure(port, 2, GPIO_INPUT);
        gpio_pin_configure(port, 3, GPIO_INPUT);

        gpio_init_callback(&gpio_cb, callback_edge, BIT(2) | BIT(3));
        gpio_add_callback(port, &gpio_cb);
```

(下页继续)

(续上页)

```
        gpio_pin_interrupt_configure(port, 2, GPIO_INT_EDGE_BOTH);
        gpio_pin_interrupt_configure(port, 3, GPIO_INT_EDGE_BOTH);
}
```

5.2 2.4G 发射初始化　设置发射频点 2450Mhz

```
        pan_prf_config_t tx_config = {
                .work_mode                          = CONFIG_PRF_WORK_MODE,
                .chip_mode                          = CONFIG_PRF_CHIP_MODE,
                .trx_mode                          = PRF_TX_MODE,
                .phy                                  = PRF_PHY_1M,
                .crc                                  = CONFIG_PRF_CRC_MODE,
                .src                                  = CONFIG_PRF_SRC_MODE,
                .rx_timeout                         = 50000,
                .rf_channel                         = 2450,
                .tx_no_ack                         = DISABLE,
                .nrf52_mode                         = CONFIG_PRF_NRF52_MODE,
                .rx_length                         = 0,
                .addr_length                = 4,
                .addr                                   = { 0x71, 0x76, 0x41, 0x76 },
        };
```

2.4G 初始化配置说明如下：

初始化配置的结构体 “pan_prf_config_t”

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，no crc
prf_scramble_sel_t	src	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，配置范围 2402-2480Mhz
uint8_t	tx_no_ack	配置增强型模式下 tx 是否需要 ack
uint8_t	nrf52_mode	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	addr_length	接入地址长度配置，可配置为 3、4、5 字节
uint8_t	addr[5]	接入地址的内容

prf_mode_t：

Type	Value	Description
PRF_MODE_NORMAL	0	普通型
PRF_MODE_ENHANCE	1	增强型

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_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_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

5.3 中断处理

```
void event_tx_fun(void)
{
        printk("tx done\n");
}

void event_rx_fun(void)
{
        panchip_prf_payload_t rx_payload;

        rx_payload.data_length = panchip_prf_data_rec(&rx_payload);
        printk("rx data:");
        data_printk(rx_payload.data, rx_payload.data_length);
}

void event_rx_timeout_fun(void)
{
        printk("rx timeout\n");
}

void event_crc_err_fun(void)
{
        printk("rx data crc err\n");
}

void event_pid_err_fun(void)
{

}
```

- 增强型模式下发送完成后还会接收接收端发送的 ack 数据。

Note：2.4G **发射只支持一个通道，也就是点对点通信。如果要支持一对多通信需要遍历初始化发射端的接入的地址。**

3.2.5 解决方案

Solution: BLE Google Light

1 功能概述 本文主要介绍 PAN1080 BLE 灯控接入 google home 音箱，通过谷歌音箱控制 RGB 灯的亮度与颜色。

2 环境要求

- board: pan1080a_afld_evb
- uart (option): 显示串口 log
- google home 音箱
- google home app

3 编译和烧录 项目位置：`zephyr\samples_panchip\solutions\ble_google_light`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\solutions\ble_google_light.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`，可查看源码，执行编译下载等
  others exit                       退出
wait input:
```

4 演示说明

1. PAN1080 EVB 板 GPIO P10、P11、P16 与 RGB 电路用跳线帽连接。
2. EVB 板上电灯的颜色默认是白色，BLE 广播设备的名字是 “EnergeticLighting”。
3. 打开安卓手机” google home “app 与 google home 音箱连接，然后在 app 上添加灯泡设备，app 上启动搜索灯泡。
4. 搜索到后连接并配网，配网的过程中灯会闪烁。
5. 配网成功后 app 上会显示一个灯泡的图案，然后就可以控制灯的开关、亮度和颜色。
6. 灯和音箱断开连接后的 BLE 广播设备的名字是 “EnergeticLightingPRO”，此时与音箱连接无需重新配网可以直接连接。
7. app 上设置移除灯泡后此时灯进入退网状态，BLE 广播设备的名字是 “EnergeticLighting”，与音箱连接需要重新配网。

5 设备配网和控制

5.1 广播数据

Data Type	Description	Length	Detail
0xff	Device id	6byte	mac addr
0xfe	Deviceinfo: Manufacturer model	7byte	“panchip”
0xfd	HW Version	1byte	0x01
0xfc	SW Version	1byte	0x01
0x09(未配网)	Device name	n byte	“EnergeticLighting”
0x09(已配网)	Device name	n byte	“EnergeticLightingPRO”

配网前和配网后的广播包主要是名字不一样，其他的都是一样的。

5.2 GATT 服务

Function	Service Attribute	UUID(128bit)
Useless	Primary service	0x1800（Generic GAP Service）
读灯的状态	Read characteristic declaration	0x35, 0xb1, 0xc3, 0xd8, 0x7e, 0xd4, 0x21, 0x81, 0x13, 0x46, 0xb7, 0x9c, 0x43, 0xaf, 0x83, 0x76
读灯的名字	Read characteristic declaration	0x36, 0xb1, 0xc3, 0xd8, 0x7e, 0xd4, 0x21, 0x81, 0x13, 0x46, 0xb7, 0x9c, 0x43, 0xaf, 0x83, 0x76
控制灯的配网和灯状态	Write characteristic declaration	0x37, 0xb1, 0xc3, 0xd8, 0x7e, 0xd4, 0x21, 0x81, 0x13, 0x46, 0xb7, 0x9c, 0x43, 0xaf, 0x83, 0x76
读灯的亮度和开关状态	Read characteristic declaration	0x38, 0xb1, 0xc3, 0xd8, 0x7e, 0xd4, 0x21, 0x81, 0x13, 0x46, 0xb7, 0x9c, 0x43, 0xaf, 0x83, 0x76
读灯的软件版本信息	Read characteristic declaration	0x39, 0xb1, 0xc3, 0xd8, 0x7e, 0xd4, 0x21, 0x81, 0x13, 0x46, 0xb7, 0x9c, 0x43, 0xaf, 0x83, 0x76
主动上报灯的亮度和开关状态	Notify characteristic declaration	0x40, 0xb1, 0xc3, 0xd8, 0x7e, 0xd4, 0x21, 0x81, 0x13, 0x46, 0xb7, 0x9c, 0x43, 0xaf, 0x83, 0x76

5.3 通信协议

5.3.1 Read Light Status　UUID = {0x35,0xb1,0xc3,0xd8,0x7e,0xd4,0x21,0x81,0x13,0x46,0xb7,0x9c,0x43,0xaf,0x83,0x76}

Function	Length	Detail
Light OnOff	1byte	0x0: 关闭，0x1: 使能
Light Brightness	1byte	0x0: 关闭，0x1: 使能
Light Color Temperature	1byte	0x0: 关闭，0x1: 使能
Light Color Setting	1byte	0x0: 关闭，0x1: 使能
Light Min Temperature	2byte	eg：2000
Light Max Temperature	2byte	eg：7500

灯发送数据如下：

data[8]={0x01 ,0x01 ,0x01 ,0x01 ,0x07 ,0xd0, 0x1d ,0x4c}

5.3.2 Read Light Name　UUID = {0x36,0xb1,0xc3,0xd8,0x7e,0xd4,0x21,0x81,0x13,0x46,0xb7,0x9c,0x43,0xaf,0x83,0x76}

Function	Length	Detail
Name	14byte	“A19 Full Color”

灯的名字：

name_data[] = “A19 Full Color”

5.3.3 Light Control　UUID = {0x37,0xb1,0xc3,0xd8,0x7e,0xd4,0x21,0x81,0x13,0x46,0xb7,0x9c,0x43,0xaf,0x83,0x76}

Function	Length	Type（1byte）	Detail
Onoff	2byte	0x00	on：0x01；off：0x00
Bright	2byte	0x01	bright：0~100
Color Temperature	3byte	0x02	temp：2000~7500
Blink	2byte	0x03	on：0x01；off：0x00
Provision	2byte	0x04	succeed：0x01；failed：0x00
Unprovision	2byte	0x05	succeed：0x01；failed：0x00
Color	4byte	0x06	red：0~255；green：0~255；blue：0~255

控制灯的开关、亮度、色温、颜色、闪烁、配网和退网。

5.3.4 Read Light Version　UUID = {0x39,0xb1,0xc3,0xd8,0x7e,0xd4,0x21,0x81,0x13,0x46,0xb7,0x9c,0x43,0xaf,0x83,0x76}

Function	Length	Detail
Hardware and software version	2byte	hw：0x01；sw：0x01

5.3.5 Read Light Bright and Color temp　UUID = {0x38,0xb1,0xc3,0xd8,0x7e,0xd4,0x21,0x81,0x13,0x46,0xb7,0x9c,0x43,

Function	Length	Detail
Onoff	1byte	on：0x01；off：0x00
Bright	1byte	bright：0~100
ColorTemperature	2byte	temp：2000~7500

5.3.6 Notify Light Status　UUID = {0x40,0xb1,0xc3,0xd8,0x7e,0xd4,0x21,0x81,0x13,0x46,0xb7,0x9c,0x43,0xaf,0x83,0x76}

每次收到控制命令后将灯的状态通知给音箱。

Solution: BLE HID Selfie

1 功能概述　此项目演示模拟蓝牙自拍杆功能。

2 环境要求

- board: pan1080a_afld_evb
- uart (option): 显示串口 shell
- 测试硬件: 智能手机
- PC 工具: 软件 shell 工具（支持串口，波特率 921600）

3 编译和烧录　项目位置：`zephyr\samples_panchip\solutions\ble_hid_selfie`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\solutions\ble_hid_selfie.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
```

（下页继续）

(续上页)

```
  'o' open project by VS Code       打开 `VS Code`, 可查看源码, 执行编译下载等
  others exit                       退出
wait input:
```

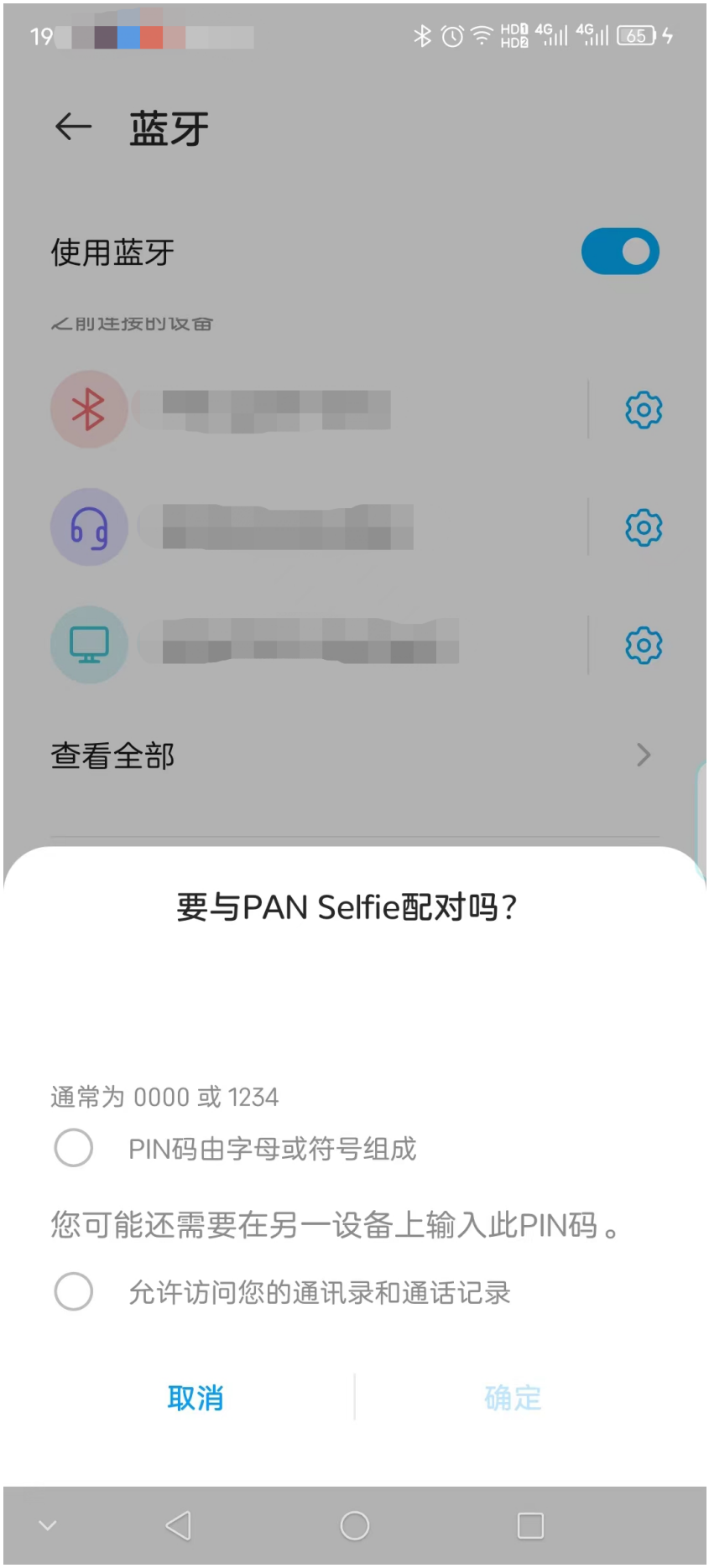
4 演示说明

1. 将 ble_hid_selfie 烧录至 EVB 板后，使用 shell 工具连接上 EVB 开发板（波特率 921600），会显示如下 log。通过连续双击 tab 可以显示出当前可用的命令。

```
Bluetooth initialized
Advertising successfully started


[00:00:00.029,000] <inf> fs_nvs: 8 Sectors of 4096 bytes
[00:00:00.029,000] <inf> fs_nvs: alloc wra: 0, fe8
[00:00:00.029,000] <inf> fs_nvs: data wra: 0, 0
[00:00:00.129,000] <inf> bt_hci_core: No ID address. App must call settings_load()
[00:00:00.139,000] <inf> bt_hci_core: Identity: EC:5F:48:F6:62:BD (random)
[00:00:00.139,000] <inf> bt_hci_core: HCI: version 5.1 (0x0a) revision 0x0003, manufacturer␣
↪0x07d1
[00:00:00.139,000] <inf> bt_hci_core: LMP: version 5.1 (0x0a) subver 0x0000
[00:00:00.154,000] <inf> bt_settings: Saving ID
uart:~$
  clear    date     device   devmem   flash    help     history  kernel
  log      resize   sensor   shell    test
uart:~$ test
  photos    ac_home   vol_up    vol_down  raw_hex
```

1. 智能手机蓝牙设置中找到 Pan Selfie 设备，选中连接并输入 shell 产生的随机数 Passkey 进行配对。

```
uart:~$ Connected 75:19:99:D2:BE:BA (random)
Passkey for 75:19:99:D2:BE:BA (random): 905570
Security changed: 75:19:99:D2:BE:BA (random) level 4
ccc 0001
ccc consumer 0001
```


蓝牙
使用蓝牙
查看全部
要与PAN Selfie配对吗?
通常为 0000 或 1234
PIN码由字母或符号组成
您可能还需要在另一设备上输入此PIN码。
允许访问您的通讯录和通话记录
取消
确定

1. 然后我们就可以通过支持的 test 命令来模拟蓝牙自拍杆的相关动作

5 支持的模拟测试命令

测试命令	功能描述
test photos	模拟拍照按键
test ac_home	模拟 Home 键
test vol_up	模拟音量 + 键
test vol_down	模拟音量-键

相应命令支持的函数可以参考 shell 命令实现:

```
SHELL_STATIC_SUBCMD_SET_CREATE(sub_test,
                               SHELL_CMD(photos,       NULL,   "Take Photos command",  cmd_
↪test_photos),
                               SHELL_CMD(ac_home,      NULL,   "AC Home command",      cmd_
↪test_ac_home),
                               SHELL_CMD(vol_up,       NULL,   "Volume up command",    cmd_
↪test_vol_up),
                               SHELL_CMD(vol_down,     NULL,   "Volume down command",  cmd_
↪test_vol_down),
                               SHELL_CMD(raw_hex,      NULL,   "RAW Hexgroup command", cmd_
↪test_raw_hex),
                               SHELL_SUBCMD_SET_END /* Array terminated. */
                               );
SHELL_CMD_REGISTER(test, &sub_test, "HID Over GATT Test commands", NULL);
```

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: pan1080a_afld_evb
- uart (option): 显示串口 log

3 编译和烧录 项目位置: `zephyr\samples_panchip\solutions\ble_pcte_beacon`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置: `quick_build_samples\solutions\ble_pcte_beacon.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

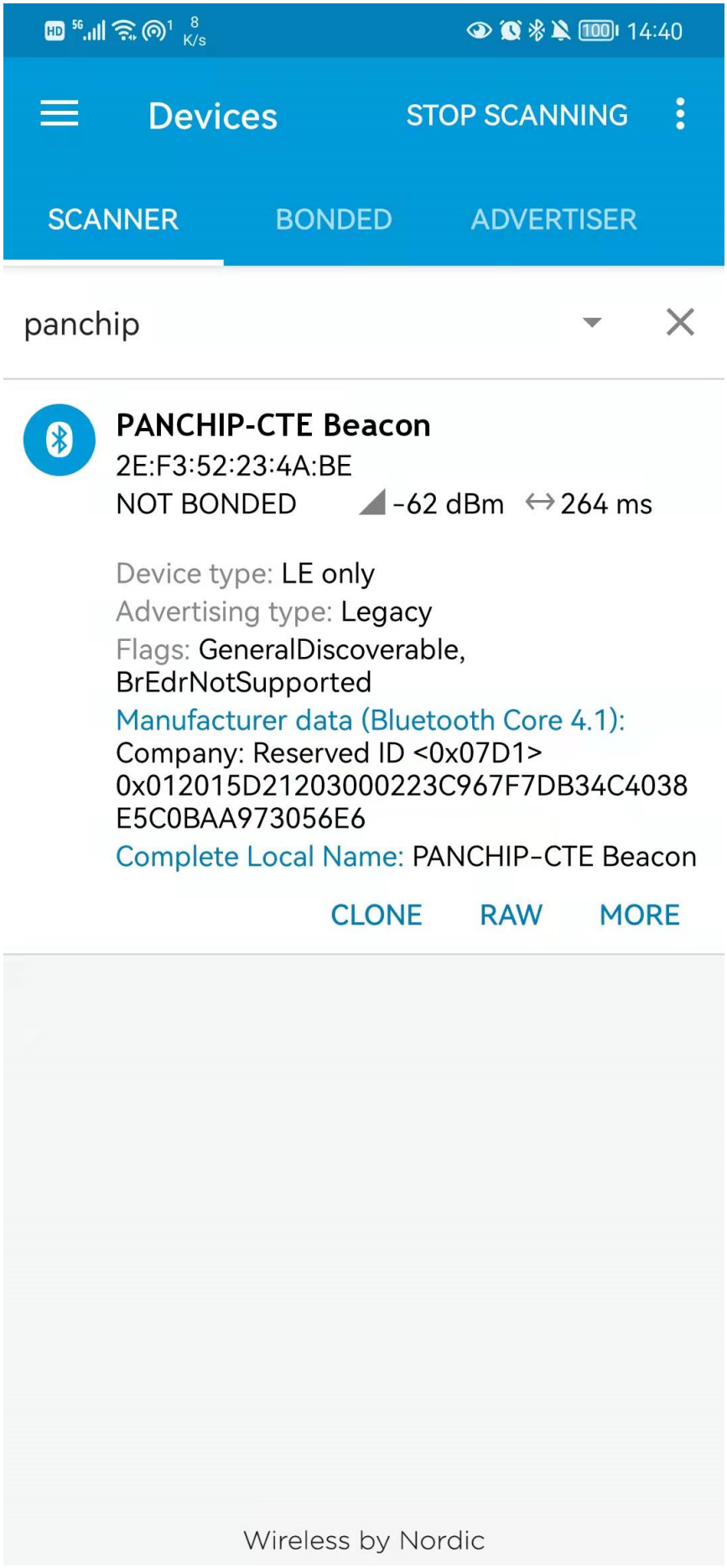
```
Input the keyword to continue:
  'b' build                         编译项目
  'r' make clean and rebuild        重新编译项目
  'f' flash download                下载
  'e' erase chip                    擦除芯片
  'o' open project by VS Code       打开 `VS Code`, 可查看源码, 执行编译下载等
  others exit                       退出
wait input:
```

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。


14:40
Devices
STOP SCANNING
SCANNER
BONDED
ADVERTISER
panchip
PANCHIP-CTE Beacon
2E:F3:52:23:4A:BE
NOT BONDED
-62 dBm
264 ms
Device type: LE only
Advertising type: Legacy
Flags: GeneralDiscoverable,
BrEdrNotSupported
Manufacturer data (Bluetooth Core 4.1):
Company: Reserved ID <0x07D1>
0x012015D21203000223C967F7DB34C4038
E5C0BAA973056E6
Complete Local Name: PANCHIP-CTE Beacon
CLONE
RAW
MORE
Wireless by Nordic

5 广播数据 广播数据包含两个 AD Element，如下表。

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	Check-sum	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。

Solution: BLE RGB Light

1 功能概述 本文主要介绍 PAN1080 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: pan1080a_afld_evb
- 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 项目位置：`zephyr\samples_panchip\solutions\ble_rgb_light`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\solutions\ble_rgb_light.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
```

(下页继续)

(续上页)

```
  'o' open project by VS Code        打开 `VS Code`, 可查看源码, 执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明

1. PAN1080 EVB 板 GPIO P10、P11、P16 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。

Solution: Mesh Panchip

1 功能概述 此 sample 为 pan1080a_afld_evb 在 solutin 应用中，多种 model 的应用，多种 function 的示例，目的为同时实现多种 model 的演示，方案级的代码构成。

具体支持的 feature 如下：1. Boot 应用 2. 多次擦除验证 BT Setting 操作 flash 3. 读取 DTS ADC 温度传感器 4. Light model DTS GPIO 灯控功能 5. Pb remote 入网功能 6. Panchip Test Vendor Model 测试收包率 7. output string oob 方式入网 8. BLE 更新连接参数

2 环境要求

- board: pan1080a_afld_evb
- uart (option): 显示串口 log
- App: PanMesh(IOS) 或 nRF Mesh(IOS/Android)
- mesh:

1. 测试 PB Remote：需要额外准备一块开发板。
2. Mesh OTA 只能运行在 1K flash 的板子上。
3. 测试 Boot OTA: Boot 源码 keil 烧录，准备串口 dongle，升级及 boot log 通过 P30，P31 打印，固件生成需要拷贝 `ota.bin` 至 build 目录。

3 编译和烧录 项目位置：`zephyr\samples_panchip\solutions\mesh_panchip`

统一的配置、编译、下载工具正在开发中，当前可以使用脚本进行编译和下载。

脚本位置：`quick_build_samples\solutions\mesh_panchip.bat`。

打开脚本后默认会编译项目，编译完成时，可输入字符进行后续下载等操作:

```
Input the keyword to continue:
  'b' build                          编译项目
  'r' make clean and rebuild         重新编译项目
  'f' flash download                 下载
  'e' erase chip                     擦除芯片
  'o' open project by VS Code        打开 `VS Code`，可查看源码，执行编译下载等
  others exit                        退出
wait input:
```

4 演示说明

1. 根据 Zephyr 编译环境搭建 VSCode 开发环境, 准备至少 1 块其他待入网设备，发送 beacon 如 1020.
2. 下载前需要 erase chip（`CONFIG_BT_SETTINGS=y` 情况下需要，因为需要 stored cdb，防止 flash 冲突），`CONFIG_BT_SETTINGS=n` 情况下不需要
3. 观察 ADC 打印
4. 通过 Nrf Mesh 进行入网，入网时选择入网方式 output oob
5. 连接后可以通过抓包确认连接间隔为 100ms
6. 入网后按键 4 次擦除入网信息
7. 通过 PanMesh 入网，验证灯控
8. 通过 PanMesh 绑定 appkey 后验证 packet test vendor
9. 通过 PanMesh 验证 Pbremote 入网流程

5 开发说明

5.1 KCONFIG

1. 栈大小分配

```
CONFIG_SYSTEM_WORKQUEUE_STACK_SIZE=2048
CONFIG_ISR_STACK_SIZE=2048
CONFIG_BT_RX_STACK_SIZE=3072
```

2. flash 区域划分，boot 应用相关，带 boot 的程序需要如下配置

```
CONFIG_IS_BOOTLOADER=n
CONFIG_USE_DT_CODE_PARTITION=y
```

3. driver 相关，需要 adc driver 相关的宏

```
CONFIG_ADC=y
```

4. flash 相关，demo 开启了 flash 存储功能，入网信息可以保存擦除等操作，并且需要 flash driver 完成 ota

```
CONFIG_BT_SETTINGS=y
CONFIG_FLASH=y
CONFIG_FLASH_MAP=y
CONFIG_NVS=y
CONFIG_SETTINGS=y
```

5. BLE 相关，demo 涉及 DLE，CONFIG_BT_DEVICE_NAME 显示等

```
CONFIG_BT=y
CONFIG_BT_OBSERVER=y
CONFIG_BT_DATA_LEN_UPDATE=n
CONFIG_BT_AUTO_DATA_LEN_UPDATE=y
CONFIG_BT_PHY_UPDATE=n
CONFIG_BT_PERIPHERAL=y
CONFIG_BT_TINYCRYPT_ECC=y
CONFIG_BT_L2CAP_TX_BUF_COUNT=5
CONFIG_BT_DEVICE_NAME_DYNAMIC=y
CONFIG_BT_DEVICE_NAME="panchip"
```

6. mesh 相关参数
    1. CONFIG_BT_PAN_MESH_MODELS=y 开发 model 层总开关
    2. 可以有 3 个可选择 model
    3. 其他 mesh 参数的分配和使能

```
CONFIG_BT_MESH=y

CONFIG_BT_PAN_MESH_MODELS=y

CONFIG_BT_MESH_PB_REMOTE_SRV=y
CONFIG_BT_MESH_SIG_OTA_SRV=y
CONFIG_BT_MESH_PTV_SRV=y

CONFIG_BT_MESH_SUBNET_COUNT=1
CONFIG_BT_MESH_APP_KEY_COUNT=2
CONFIG_BT_MESH_MODEL_GROUP_COUNT=2
CONFIG_BT_MESH_LABEL_COUNT=1

CONFIG_BT_MESH_TX_SEG_MSG_COUNT=6
CONFIG_BT_MESH_RX_SEG_MSG_COUNT=6
CONFIG_BT_MESH_ADV_BUF_COUNT=30
CONFIG_BT_MESH_TX_SEG_MAX=6
```

(下页继续)

(续上页)

```
CONFIG_BT_MESH_RX_SEG_MAX=6

CONFIG_BT_MESH_UNPROV_BEACON_INT=5

CONFIG_BT_MESH_MODEL_EXTENSIONS=y
CONFIG_BT_MESH_RELAY=y
CONFIG_BT_MESH_FRIEND=n
CONFIG_BT_MESH_PB_GATT=y
CONFIG_BT_MESH_PB_ADV=y
CONFIG_BT_MESH_GATT_PROXY=y
```

7. 调试 LOG 宏，不建议全开，log 较多占一定 ram

```
CONFIG_BT_DEBUG_LOG=y

CONFIG_BT_MESH_DEBUG=y
CONFIG_BT_DEBUG_CONN=y
CONFIG_BT_MESH_DEBUG_BEACON=y
CONFIG_BT_MESH_DEBUG_PROV=y
CONFIG_BT_MESH_DEBUG_ADV=y
CONFIG_BT_MESH_DEBUG_PROXY=y
CONFIG_BT_MESH_DEBUG_MODEL=y
CONFIG_BT_MESH_DEBUG_ACCESS=y
CONFIG_BT_MESH_DEBUG_TRANS=y
CONFIG_BT_MESH_DEBUG_KEYS=y
CONFIG_BT_MESH_DEBUG_NET=y
```

5.2 MAIN 函数

main 函数包括 3 部分

1. flash 相关的配置擦除, 包括函数 `ps_settings_init`,`short_time_multireset_bt_mesh_unprovisioning`
2. `bt_enable()` 初始化蓝牙
3. `bt_ready()` 初始化 MESH
4. 蓝牙回调接口

```
static void connected(struct bt_conn *conn, uint8_t err)
{
        if (err) {
                printk("Connection failed (err 0x%02x)\n", err);
        } else {
                printk("Connected\n");
        }
}

static void disconnected(struct bt_conn *conn, uint8_t reason)
{
        printk("Disconnected (reason 0x%02x)\n", reason);
}

BT_CONN_CB_DEFINE(conn_callbacks) = {
        .connected = connected,
        .disconnected = disconnected,
};
```

5.3 ble_mesh

ble/mesh 相关操作:

1. `static const struct bt_mesh_prov prov` 配置入网参数
2. `struct bt_mesh_model root_models[]` 配置 mesh model

通过 `pro.conj` 配置使能相应 model

3. UUID 生成方式：静态数组 `static uint8_t dev_uuid[16]`
4. 入网认证方式：output string
5. 入网 beaer 选择：`bt_mesh_prov_enable(BT_MESH_PROV_ADV | BT_MESH_PROV_GATT);`

5.4 其他说明

1. light model 和 config model，health model 相关参考后续介绍 model 文档，参考 spec 文档。
2. Pb remote model 实现在 zephyr 内 `subsys\bluetooth\mesh_models\sig_models\remote_srv.c`
3. Sig OTA 实现在 zephyr 内 `subsys\bluetooth\mesh_models\sig_models\blob_srv.c`
4. Packet Test Vendor Model 实现在 zephyr 内 `subsys\bluetooth\mesh_models\vendor_models\ptv_srv.c`

6 补充说明

1. mesh_model 源码在 `subsys\bluetooth\mesh_models\`内统一 coding
2. 由于 Ram 剩余不足，Mesh Log 分情况打开。

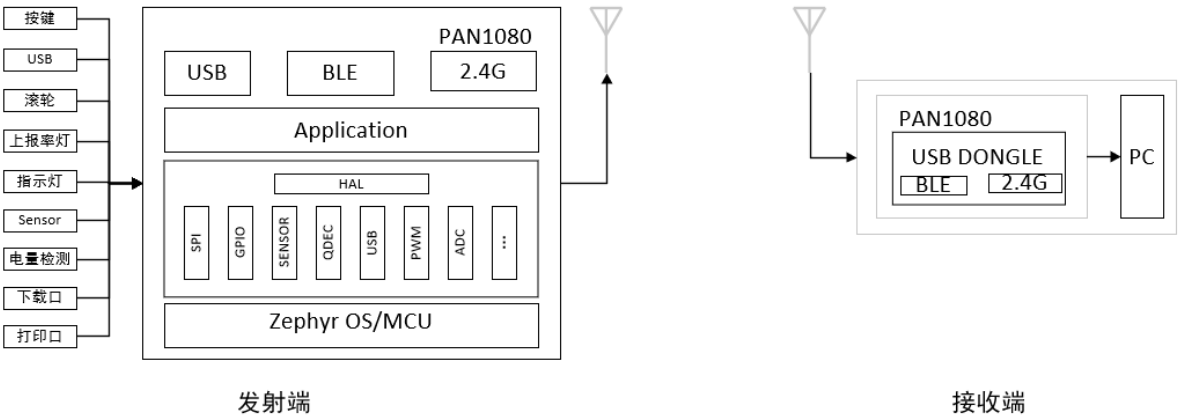
Solution: Multi-mode Mouse

重要: 此例程仅存在于特殊版本的 SDK 中，如有需要请联系 Panchip。

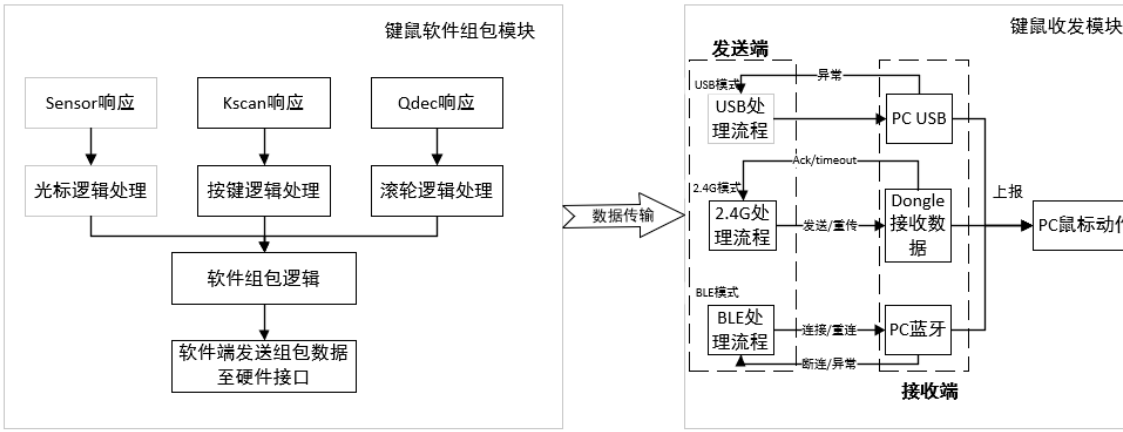
1.1 概述

本文主要介绍 PAN1080 鼠标解决方案的设计，方案主体包含发送端与接收端两部分，其中接收端仅为 rf 模式服务，发送端可以根据实际使用选择 BLE 模式/2.4G 私有模式/USB 模式进行交互，其中：

- BLE 模式下仅蓝牙工作，交互对象可以是 PC 或其他支持蓝牙的设备。
- 2.4G 私有模式仅 private rf 工作，接收端为 usb dongle，接收端需设置为相同工作频点，rf 接收数据并上报给 usb 发送。
- USB 模式仅 SOC 自带 usb 模块工作。



1.2 系统框图



1.2.1 软件工作流程：

- 其中组包模块包含一个 1ms 的定时器，用于定时采集 sensor/qdec/kscan 的实时数据并完成鼠标数据包组包操作，最后将数据发送给收发模块的发送端。
- 其中收发模块中 BLE 模式的发送端收到软件给出的数据，判别 BLE 连接状态，在连接时将数据发送给接收端，否则进行其他异常处理，BLE 模式下接收端为 pc 或者支持蓝牙的设备。
- 其中收发模块中 2.4G 私有模式的发送端收到软件给出的数据并发送给 dongle 端，dongle 端收到数据并回复 ack 数据包给发送端，dongle 端跳频，发送端收到数据后跳频。若未回复，发送端进行重传，其中 2.4G 私有模式下接收端仅为 usb dongle。
- 其中收发模块中 USB 模式的发送端收到软件给出的数据并在下一个 usb sof 开始发送，其中 USB 模式下接收端为 pc 或者其他 host 设备。

1.3 硬件环境

1. PAN1080 EVB 板子两个，其中一个作为发送端，一个接收端（dongle）
   - P30_UART0_TX、P31_UART0_RX （uart0 log 打印）
   - P46_SWD_CLK、P47_SWD_DAT （swd 下载口）
2. USB：P02_USB_DM、P03_USB_DP，其中 usb 线对应关系如下
   - 红线：电源正极（接线上的标识为：+5V 或 VCC）
   - 白线：DM 数据线（标识为：Data-或 USB Port -）
   - 绿线：DP 数据线（标识为：Data+ 或 USB Port +）
   - 黑线：接地（标识为：GROUND 或 GND）
3. Secure CRT（串口打印窗口）
4. 带 BLE 功能电脑
5. usb 抓包工具，bushound 或者 usbmonitor
6. 上报率测试工具，MouseTest.exe（仅部分电脑可以达到上报率 1KHz）
7. 逻辑分析仪

1.4 软件环境

1. 测试源文件目录
   - 鼠标发送端:..\BLE-Group\PAN1080\pan1080-dk-internal\01_SDK\zephyr\samples_panchip\solutions\moc
   - 接收端:..\BLE-Group\PAN1080\pan1080-dk-internal\01_SDK\zephyr\samples_panchip\solutions\usb_don

1.5 软件功能点

功能点描述	工作模式	备注
BLE-USB 模式鼠标常规功能	ble-usb mode	此模式下 BLE 模块和 USB 模块均打开，2.4G 模块不工作，可切换模式，需要鼠标外围如 sensor 等支持
2.4G-USB 模式鼠标常规功能	rf-usb mode	此模式下 2.4G 模块和 USB 模块均打开，BLE 模块不工作，可切换模式，需要鼠标外围如 sensor 等支持
2.4G-ONLY 上报率	rf-only in band mode	此模式下仅 2.4G 模块打开，BLE 模块和 USB 模块不工作，鼠标上报固定画圈数据
USB-ONLY 上报率	usb-only mode	此模式下仅 USB 模块打开，BLE 模块和 2.4G 模块不工作，鼠标上报固定画圈数据
BLE-ONLY 上报率	ble-only mode	此模式下仅 BLE 模块打开，USB 模块和 2.4G 模块不工作，鼠标上报固定画圈数据
2.4G 私有模式重传功能	rf-enhance in band mode	需要鼠标外围如 sensor 等支持，测试模式已支持
2.4G 私有模式跳频功能	rf-freq-hop in band mode	需要鼠标外围如 sensor 等支持，测试模式已支持
电池电量检测	ble-usb mode 或者 rf-usb mode	需要外围电路支持
模式切换	ble-usb mode 或者 rf-usb mode	需要鼠标外围如 sensor 等支持
RGB 呼吸灯（扩展）	暂不支持	暂不支持

1.6 性能指标

1. 支持有线最大 1KHz 上报率，2.4G 1KHz，蓝牙 130Hz。
2. 通信距离（待定）

1.7 测试用例

1.7.1 编译命令　mouse 端编译下载命令:

1. mouse ble-usb mode:

```
脚本位置:
.\sdk_quick_build_samples\solutions\model_mouse_ble_usb.bat
```

> **说明**: 此 case 下 ble lib 使能，case 在拔出 usb 后通过 ble 发送数据并在 PC 上显示鼠标操作，插入 usb 时关闭 ble 扫描并通过 usb 发送数据。此 case 需要鼠标 sensor 等器件支持。

2. mouse rf-usb mode:

```
脚本位置:
.\sdk_quick_build_samples\solutions\model_mouse_prf_usb.bat
```

> **说明**: 此 case 下 rf lib 使能，case 在拔出 usb 时通过 rf 发送数据并在 PC 上显示鼠标操作，插入 usb 时关闭 rf 发送流程并通过 usb 发送数据。此 case 需要鼠标 sensor 等器件支持。

3. mouse rf-only in band mode:

```
脚本位置:
.\sdk_quick_build_samples\solutions\model_mouse_prf_only.bat
```

> **说明**: 此 case 下 rf lib 使能，仅 rf 工作发送数据，rf 工作在带内，发送固定画圈数据。此 case 作为测试 case 固定画圈。

4. mouse usb-only mode:

```
脚本位置:
.\sdk_quick_build_samples\solutions\model_mouse_usb_only.bat
```

> **说明**: 此 case 下无 lib 使能，发送固定画圈数据。此 case 作为测试 case 固定画圈。

5. mouse ble-only mode:

```
脚本位置:
.\sdk_quick_build_samples\solutions\model_mouse_ble_only.bat
```

> **说明**: 此 case 下 ble lib 使能，发送固定画圈数据。此 case 作为测试 case 固定画圈。

6. mouse rf-enhance in band mode:

```
脚本位置:
.\sdk_quick_build_samples\solutions\model_mouse_prf_enhance.bat
```

> **说明**: 此 case 下，rf 使用 enhance 模式并在 1ms 内出现无应答时重传。此 case 作为测试 case 固定画圈。

7. mouse rf-freq-hop in band mode:

```
脚本位置:
.\sdk_quick_build_samples\solutions\model_mouse_prf_freq_hop.bat
```

> **说明**: 此 case 下，rf 使用 enhance 模式并在 1ms 内出现无应答时重传，1ms 跳频一次。此 case 作为测试 case 固定画圈。

dongle **端编译命令：**

1. mouse rf-only normal mode:

```
脚本位置:
.\sdk_quick_build_samples\solutions\usb_dongle.bat
```

> **说明**: 此 case 下仅 rf 工作接收数据，rf 工作在带内，rf 接收数据发送给 usb。

2. mouse rf-only in band mode:

```
脚本位置:
.\sdk_quick_build_samples\solutions\usb_dongle_prf_retransmission.bat
```

> **说明**: 此 case 下仅测试模式专用，rf 工作接收数据，rf 工作在带内，rf 判断接收数据得第一 byte 是否于前一包数据的 id 一致，一致则不发送给 usb。

3. mouse rf-enhance in band mode:

```
脚本位置:
.\sdk_quick_build_samples\solutions\usb_dongle_prf_enhance.bat
```

> **说明**: 此 case 下 rf 收到数据后上报给 usb 端并给 tx 端发送 ack 包，不切换频率。

4. mouse rf-freq-hop in band mode:

```
脚本位置:
.\sdk_quick_build_samples\solutions\usb_dongle_prf_freq_hop.bat
```

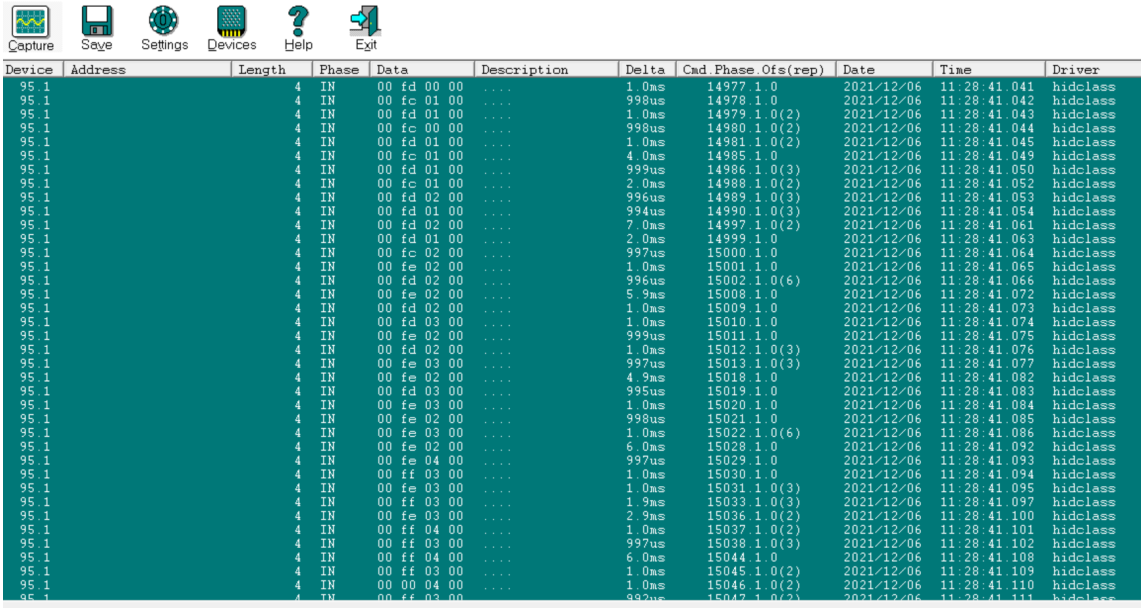
> **说明**: 此 case 下 rf 收到数据后上报给 usb 端并给 tx 端发送 ack 包，同时切换频率。

1.7.2 **测试** case

1.7.3 ble-usb 模式鼠标基础功能及上报率　测试流程

1. 执行鼠标端 mouse ble-usb mode 编译命令，下载程序。
2. 鼠标端断开 usb，进入 ble 模式，开始广播，同时观察是否触发 usb plug out 中断。
3. 打开电脑 “开始-> 设置-> 设备-> 蓝牙和其他设备-> 添加蓝牙和其他设备”，找寻设备名称为 pan ble mouse 的蓝牙设备，点击连接。或者打开 nrfConnect app 搜寻 pan ble mouse 设备，连接此设备，至配对成功。
4. 打开 MouseTest.exe 工具，移动鼠标，测试 ble 模式下上报率
5. 鼠标端插入 usb，进入 usb 模式，可以观察是否触发 usb plug in 中断，同时搜寻 ble 广播设备是否已经没有名称为 pan ble mouse 的蓝牙设备。
6. 打开 MouseTest.exe 工具，移动鼠标，测试 usb 模式下上报率。
7. 多次循环执行 usb 插拔以测试能否正确识别是何种模式生效。

测试结果

1. 断开 usb，ble 模式使用 Mouse Test 工具观测上报率，最大上报率约为 130Hz
2. 插入 usb，上报率为 1KHz
3. 

Device	Address	Length	Phase	Data	Description	Delta	Cmd.Phase.Ofs(rep)	Date	Time	Driver
95.1		4	IN	00 fd 00 00	....	1.0ms	14977.1.0	2021/12/06	11:28:41.041	hidclass
95.1		4	IN	00 fc 01 00	....	998us	14978.1.0	2021/12/06	11:28:41.042	hidclass
95.1		4	IN	00 fd 01 00	....	1.0ms	14979.1.0(2)	2021/12/06	11:28:41.043	hidclass
95.1		4	IN	00 fc 00 00	....	998us	14980.1.0(2)	2021/12/06	11:28:41.044	hidclass
95.1		4	IN	00 fd 01 00	....	1.0ms	14981.1.0(2)	2021/12/06	11:28:41.045	hidclass
95.1		4	IN	00 fc 01 00	....	4.0ms	14985.1.0	2021/12/06	11:28:41.049	hidclass
95.1		4	IN	00 fd 01 00	....	999us	14986.1.0(3)	2021/12/06	11:28:41.050	hidclass
95.1		4	IN	00 fc 01 00	....	2.0ms	14988.1.0(2)	2021/12/06	11:28:41.052	hidclass
95.1		4	IN	00 fd 02 00	....	996us	14989.1.0(3)	2021/12/06	11:28:41.053	hidclass
95.1		4	IN	00 fd 01 00	....	994us	14990.1.0(3)	2021/12/06	11:28:41.054	hidclass
95.1		4	IN	00 fd 02 00	....	7.0ms	14997.1.0(2)	2021/12/06	11:28:41.061	hidclass
95.1		4	IN	00 fd 01 00	....	2.0ms	14999.1.0	2021/12/06	11:28:41.063	hidclass
95.1		4	IN	00 fc 02 00	....	997us	15000.1.0	2021/12/06	11:28:41.064	hidclass
95.1		4	IN	00 fe 02 00	....	1.0ms	15001.1.0	2021/12/06	11:28:41.065	hidclass
95.1		4	IN	00 fd 02 00	....	996us	15002.1.0(6)	2021/12/06	11:28:41.066	hidclass
95.1		4	IN	00 fe 02 00	....	5.9ms	15008.1.0	2021/12/06	11:28:41.072	hidclass
95.1		4	IN	00 fd 02 00	....	1.0ms	15009.1.0	2021/12/06	11:28:41.073	hidclass
95.1		4	IN	00 fd 03 00	....	1.0ms	15010.1.0	2021/12/06	11:28:41.074	hidclass
95.1		4	IN	00 fe 02 00	....	999us	15011.1.0	2021/12/06	11:28:41.075	hidclass
95.1		4	IN	00 fd 02 00	....	1.0ms	15012.1.0(3)	2021/12/06	11:28:41.076	hidclass
95.1		4	IN	00 fe 03 00	....	997us	15013.1.0(3)	2021/12/06	11:28:41.077	hidclass
95.1		4	IN	00 fe 02 00	....	4.9ms	15018.1.0	2021/12/06	11:28:41.082	hidclass
95.1		4	IN	00 fd 03 00	....	995us	15019.1.0	2021/12/06	11:28:41.083	hidclass
95.1		4	IN	00 fe 03 00	....	1.0ms	15020.1.0	2021/12/06	11:28:41.084	hidclass
95.1		4	IN	00 fe 02 00	....	998us	15021.1.0	2021/12/06	11:28:41.085	hidclass
95.1		4	IN	00 fe 03 00	....	1.0ms	15022.1.0(6)	2021/12/06	11:28:41.086	hidclass
95.1		4	IN	00 fe 02 00	....	6.0ms	15028.1.0	2021/12/06	11:28:41.092	hidclass
95.1		4	IN	00 fe 04 00	....	997us	15029.1.0	2021/12/06	11:28:41.093	hidclass
95.1		4	IN	00 ff 03 00	....	1.0ms	15030.1.0	2021/12/06	11:28:41.094	hidclass
95.1		4	IN	00 fe 03 00	....	1.0ms	15031.1.0(3)	2021/12/06	11:28:41.095	hidclass
95.1		4	IN	00 ff 03 00	....	1.9ms	15033.1.0(3)	2021/12/06	11:28:41.097	hidclass
95.1		4	IN	00 fe 03 00	....	2.9ms	15036.1.0(2)	2021/12/06	11:28:41.100	hidclass
95.1		4	IN	00 ff 04 00	....	1.0ms	15037.1.0(2)	2021/12/06	11:28:41.101	hidclass
95.1		4	IN	00 ff 03 00	....	997us	15038.1.0(3)	2021/12/06	11:28:41.102	hidclass
95.1		4	IN	00 ff 04 00	....	6.0ms	15044.1.0	2021/12/06	11:28:41.108	hidclass
95.1		4	IN	00 ff 03 00	....	1.0ms	15045.1.0(2)	2021/12/06	11:28:41.109	hidclass
95.1		4	IN	00 00 04 00	....	1.0ms	15046.1.0(2)	2021/12/06	11:28:41.110	hidclass
95.1		4	IN	00 fe 03 00	....	998us	15047.1.0(2)	2021/12/06	11:28:41.111	hidclass

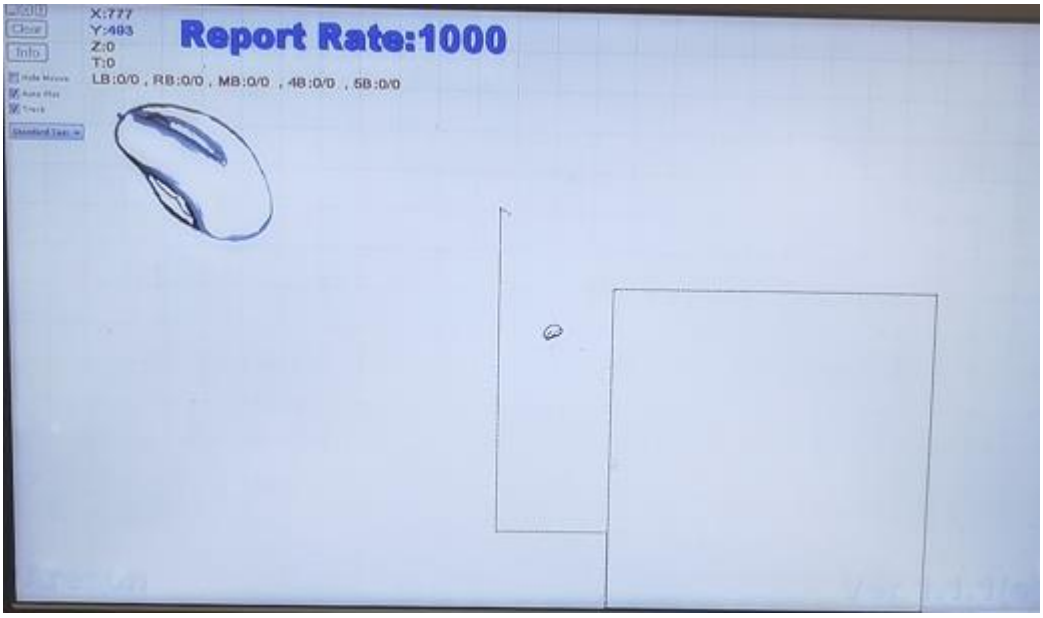
4. 循环插拔 usb，可切换模式

1.7.4 rf-usb 模式鼠标基础功能及上报率　测试流程

1. 执行鼠标端 mouse rf-only in band mode 编译命令，下载程序。
2. 执行 dongle 端 mouse rf-only normal mode 编译命令，下载程序。
3. 鼠标端断开 usb，进入 2.4G 私有模式，同时观察是否触发 usb plug out 中断。
4. 打开 MouseTest.exe 工具，移动鼠标，测试 2.4G 私有模式下上报率。
5. 鼠标端插入 usb，进入 usb 模式，可以观察是否触发 usb plug in 中断。
6. 打开 MouseTest.exe 工具，移动鼠标，测试 usb 模式下上报率。
7. 多次循环执行 usb 插拔以测试能否正确识别是何种模式生效。

测试结果

1. 断开 usb，rf 模式使用特定电脑上（实验室台式机）的 Mouse Test 工具观测上报率，最大上报率约为 998Hz

2. 

3. 插入 usb，上报率为 1KHz
4. 循环插拔 usb，可切换模式

1.7.5 rf-only 带内测试模式上报率　测试流程

1. 执行鼠标端 mouse rf-only in band mode 编译命令，下载程序。
2. 执行 dongle 端 mouse rf-only in band mode 编译命令，下载程序。
3. 发送端按住复位不释放，dongle 端 USB 插入等待设备在 pc 设备管理器上显示。
4. dongle 端复位，释放发送端 reset，这里主要是需要等待 dongle 端的 usb 完成初始化，避免被 rf 中断打断。
5. 打开 bushound 或者 Mouse Test 工具，测试上报率情况。
6. 测试 tx 发送两次数据需要的时间（扩展，需要修改代码，用逻辑分析仪抓取时间间隔）。

扩展测试代码添加

1. 在..\modules\hal\panchip\panplat\pan1080\prf\src\comm_prf.c 中，panchip_prf_init 函数中**打开**注释代码

```
#if 1
// o_pp_phy_drv_ll_rx_phy_en
PRI_RF_WRITE_REG_VALUE(PRI_RF, TEST_MUX02, TST_MUX_SELECT_09, 0x32);
SYS->P2_MFP |= SYS_MFP_P21_LL_DBG09;

// o_pp_phy_drv_ll_tx_phy_en
PRI_RF_WRITE_REG_VALUE(PRI_RF, TEST_MUX02, TST_MUX_SELECT_10, 0x31);
SYS->P0_MFP |= SYS_MFP_P04_LL_DBG10;

// pp_acc_addr_match
PRI_RF_WRITE_REG_VALUE(PRI_RF, TEST_MUX02, TST_MUX_SELECT_08, 0x33);
SYS->P2_MFP |= SYS_MFP_P20_LL_DBG08;

// i_phy_drv_ll_rx_clk
PRI_RF_WRITE_REG_VALUE(PRI_RF, TEST_MUX03, TST_MUX_SELECT_12, 0x35);
SYS->P2_MFP |= SYS_MFP_P26_LL_DBG12;

// i_phy_drv_ll_rx_data
PRI_RF_WRITE_REG_VALUE(PRI_RF, TEST_MUX03, TST_MUX_SELECT_13, 0x34);
```

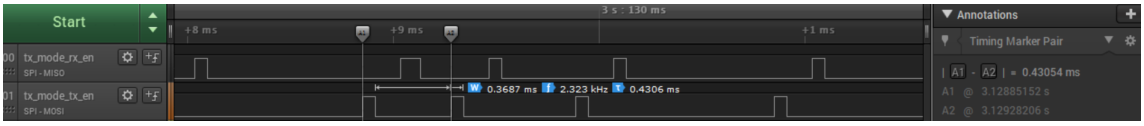
(下页继续)

(续上页)

```
SYS->P1_MFP |= SYS_MFP_P16_LL_DBG13;
#endif
```

2. 编译后逻辑分析仪抓取时间间隔

测试结果

1. rf 模式使用特定电脑上（实验室台式机）的 Mouse Test 工具观测上报率，最大上报率约为 1KHz
2. 两次发送的时间总耗时约为 400us
3. 

$$ 单次发送时间 = pre_tx + tx + post_tx $$

$$ 两次发送时间 = 单次发送时间 * 2 + software_delay（上图中两次高电平中的低电平时间，软件设置为 160us，可缩短）$$

1.7.6 usb-only 模式鼠标上报率 测试流程

1. 执行鼠标端 mouse usb-only mode 编译命令，下载程序。
2. 鼠标端断开 usb，观察是否触发 usb plug out 中断。
3. 鼠标端插入 usb，观察是否触发 usb plug in 中断。
4. 将 GPIO P23 杜邦线接地，先不让 usb 上报数据。
5. GPIO P23 杜邦线接高电平，打开 MouseTest.exe 工具，测试 usb 模式下上报率。
6. 多次循环执行 usb 插拔以测试能否正确识别 USB。
7. 将 GPIO P23 杜邦线接地，等待电脑灭屏。
8. 在串口显示 USB isr in: Suspend evt 信息后，拉高 P23，观察电脑是否亮屏并自动画圈。
9. 多次执行步骤 7，8，观察是否每次都能亮屏画圈。

测试结果

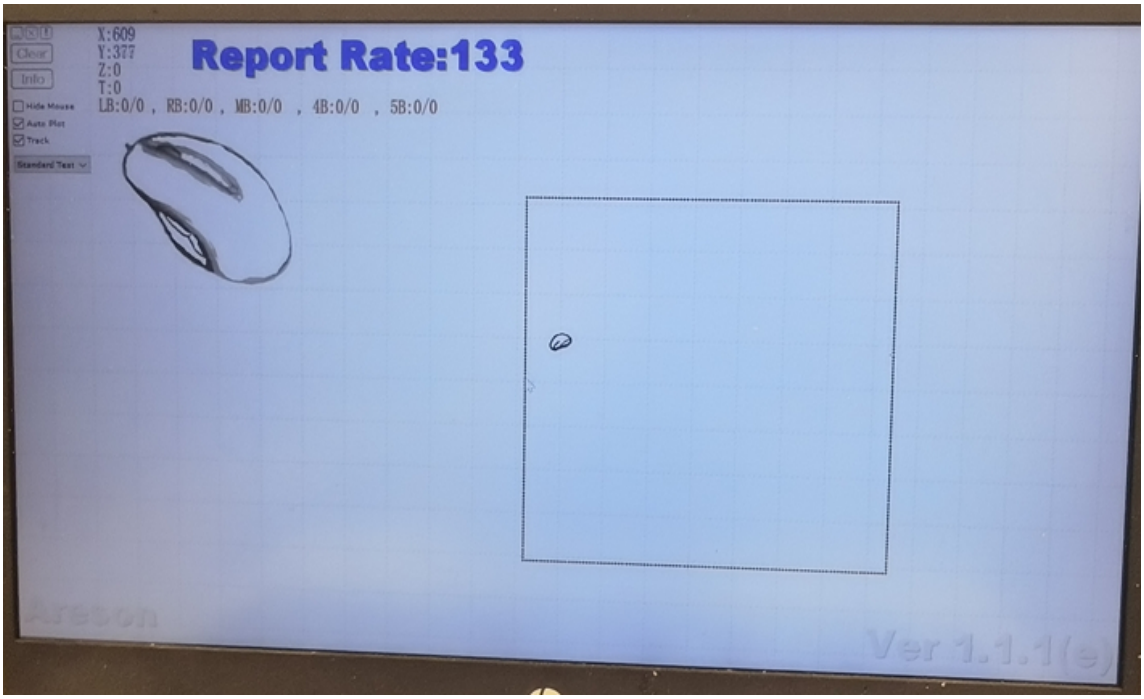
1. 断开 usb，鼠标不动作，触发 plug out 中断。
2. 插入 usb，触发 plug in 中断，最大上报率为 1KHz。
3. 循环插拔 usb，可正确识别 usb 设备。

1.7.7 ble-only 模式鼠标上报率 测试流程

1. 执行鼠标端 mouse ble-only mode 编译命令，下载程序。
2. 打开电脑 “开始-> 设置-> 设备-> 蓝牙和其他设备-> 添加蓝牙和其他设备”，找寻设备名称为 pan ble mouse 的蓝牙设备，点击连接。或者打开 nrfConnect app 搜寻 pan ble mouse 设备，连接此设备，至配对成功。
3. 打开 MouseTest.exe 工具，测试 ble 模式下上报率。

测试结果

1. 断开 usb，鼠标不动作，触发 plug out 中断。
2. 插入 usb，触发 plug in 中断，最大上报率为 135Hz

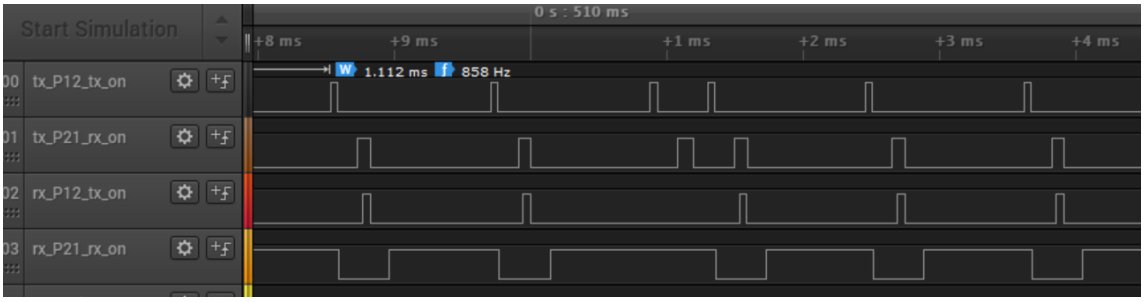
3. 

4. 循环插拔 usb，可正确识别 usb 设备。

1.7.8 rf-enhance 模式带重传跳频的上报率　测试流程

1. 执行鼠标端 mouse rf-freq-hop in band mode 编译命令，下载程序。
2. 执行 dongle 端 mouse rf-freq-hop in band mode 编译命令，下载程序。
3. 发送端按住复位不释放，dongle 端复位，再释放发送端 reset，这里主要是需要等待 dongle 端的 usb 完成初始化，避免被 rf 中断打断。
4. 打开 bushound 或者 Mouse Test 工具测试上报率情况。
5. 测试是否能在 1ms 内重传完成。
6. 测试跳频机制是否正常。测试 case 中 tx 端每间隔 1ms 改变一次频点，如果在 1ms 内发送一次数据后未正常收到 ack，那么此时频点不改变，同时再次发送与第一次发送的数据相同的包，收到 ack 后跳频。rx 端在接收数据并发送完成数据后改变频率。

测试结果

1. 此模式为带重传跳频机制的上报，在 mouse test 工具上最大上报率约为 1KHz
2. 跳频重传功能正常
3. 

上图中 00 为发送端的 tx，01 为发送端 rx，02 为接收端 tx，03 为接收端 rx，从图中可见发送端的第三个波形在第一次发送时没有 ack 发出，rx 在异常后，tx 发送端在很短时间内紧接着又发送了一次数据（数据相同，因为未对发送数据作重新载入）并成功接收了数据，第二包数据发送时 tx 和 rx 的频点均未发生改变，及至下一次发送时频点才发生改变。

Solution: USB Dongle

重要: 此例程仅存在于特殊版本的 SDK 中，如有需要请联系 Panchip。

请参考Solution: Multi-mode Mouse中有关 dongle 的说明。

Chapter 4

开发指南

4.1 快速上手 SoC App 开发

本文演示如何新建一个简单的 App 工程，并举例说明如何在 App 工程中操作 PAN1080 EVB 开发板的外设模块。

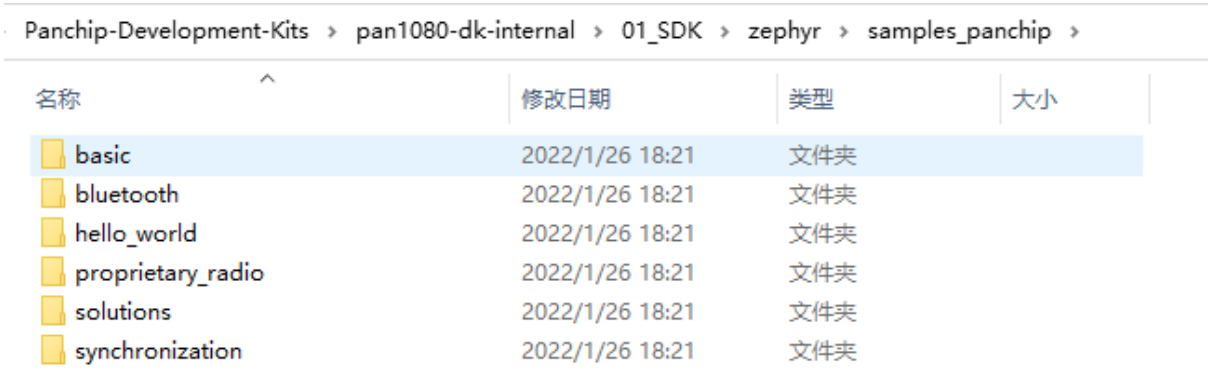
4.1.1 1 确认开发环境

参考SDK 快速入门，确认软硬件开发环境，可以正常编译、下载和调试程序。

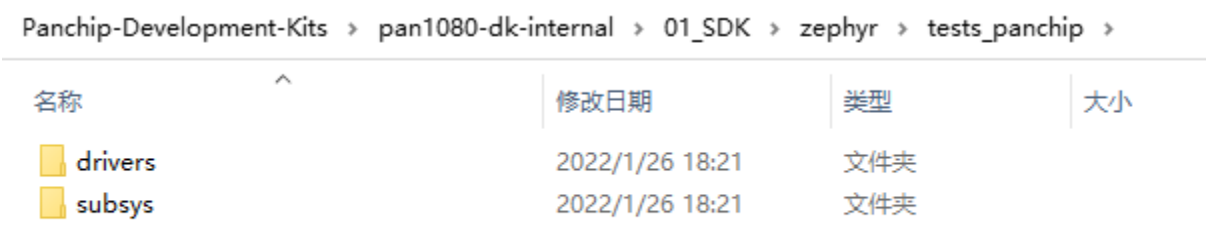
4.1.2 2 参考相关例程

SDK 中提供了一些例程（位于 `01_SDK/zephyr/samples_panchip` 目录），可以直接编译下载到 EVB 开发板上执行，包括多线程打印消息、LED 闪灯、以及蓝牙相关例程等等。

在进行开发之前，建议先看一下相关的例程和文档，熟悉 Zephyr OS 的基本框架；然后实际将这些例程编译烧录至 EVB 开发板中查看运行效果，同时进一步熟悉开发环境的使用。



另外，SDK 中也提供了一些外设模块的测试用例（位于 `01_SDK/zephyr/tests_panchip` 目录），用于测试外设 Driver 功能是否正常。阅读这些测试用例代码，也有助于熟悉 Zephyr 外设 Driver 的使用方法。



4.1.3 3 新建一个 App 工程

假设我们希望创建一个名为 `my_led_blink` 的 App 工程，使用 PWM 的方式将 EVB 开发板上的红色 LED 灯点亮并令其闪烁，要求：

1. 首先，LED 灯以 5Hz 的频率闪烁，持续 5s
2. 重复上述闪烁规则
3. 每次闪烁频率切换的时候，均向 UART 串口打印闪烁频率信息

下面我们详细讲解如何新建一个工程来实现上述需求。

3.1 从例程中拷贝一份相似的工程

例程 `blinky`（位置：`01_SDK/zephyr/samples_panchip/basic/blinky` 目录）演示了如何使用 GPIO 的方式点亮 LED 并令其每隔 1s 闪烁一次。

我们以此例程为基础进行修改：

1. 首先，拷贝一份 `blinky` 并将其重命名为 `my_led_blink`：

打开 `CMakeList.txt`，将第 5 行 project 参数修改为新的项目名称 `my_led_blink`：

```
# SPDX-License-Identifier: Apache-2.0

cmake_minimum_required(VERSION 3.20.0)
find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
project(my_led_blink)

target_sources(app PRIVATE src/main.c)
```

注：本 App 工程目录中，必要的文件为：

- `src\main.c`: 主程序代码
- `CMakeList.txt`: cmake 文件
- `prj.conf`: 项目配置文件

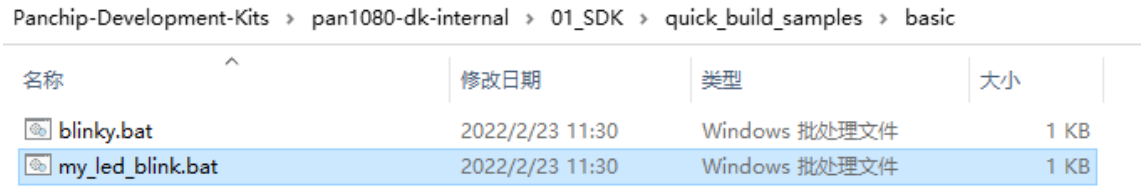
其它两个文件不是必须的（在本示例中我们可以直接将其删除）：

- `README.rst`: 例程说明文件

- `sample.yaml`: 用于 Zephyr 单元测试的信息描述文件

2. 然后，拷贝一份名为 `blinky.bat` 的 Quick Build 脚本（位于 `01_SDK/quick_build_samples/basic` 目录），用于执行编译烧录；我们同样将其重命名为与 App 工程相同，即 `my_led_blink.bat`：

   注：脚本内容无需修改，其会根据脚本文件名自动关联同名的 App 工程。

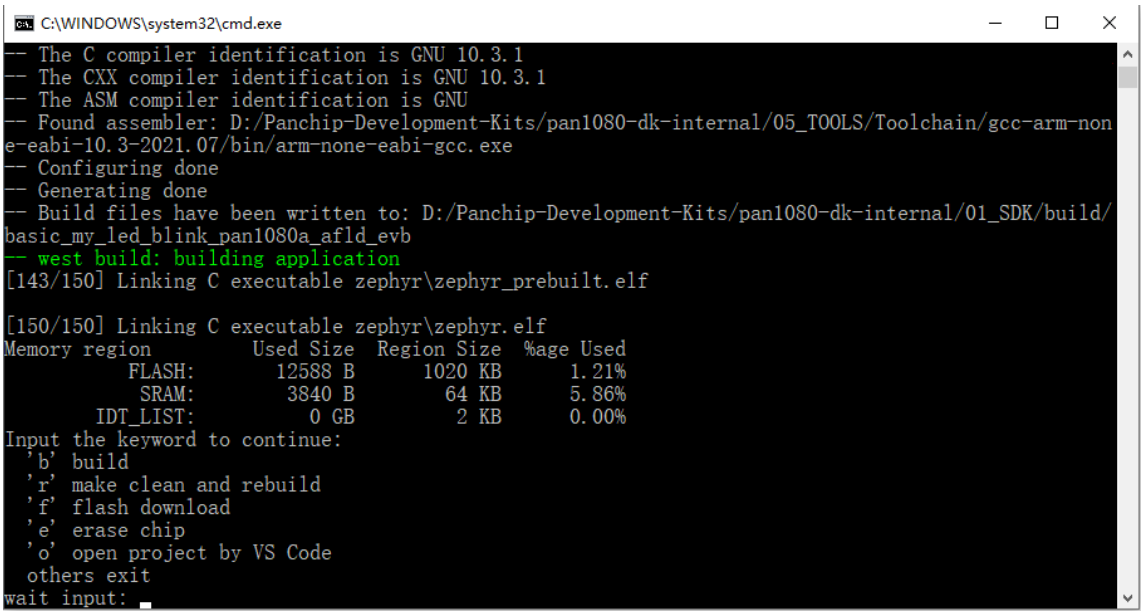


3. 最后，确认新增的 `my_led_blink.bat` 可以正常编译、烧录、运行新增的 `my_led_blink` 工程：

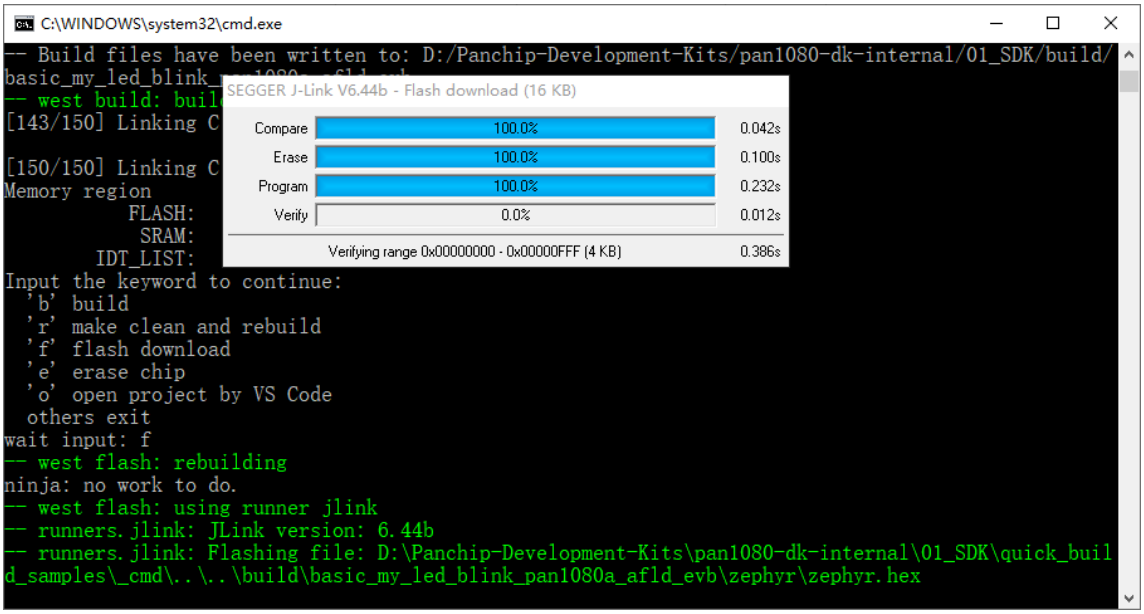
   1. 将 EVB 开发板通过 USB2（`USB to UART`）接口供电，并确认：

      1. JLink SWD 正确连接至 EVB 板（`P46: ICEK`，`P47: ICED`）
      2. 使用跳线帽将 EVB 底板（左上侧）的 LED/红外发射共用电路的排针连接至 PAN1080 SoC（`P21: FRTX`）

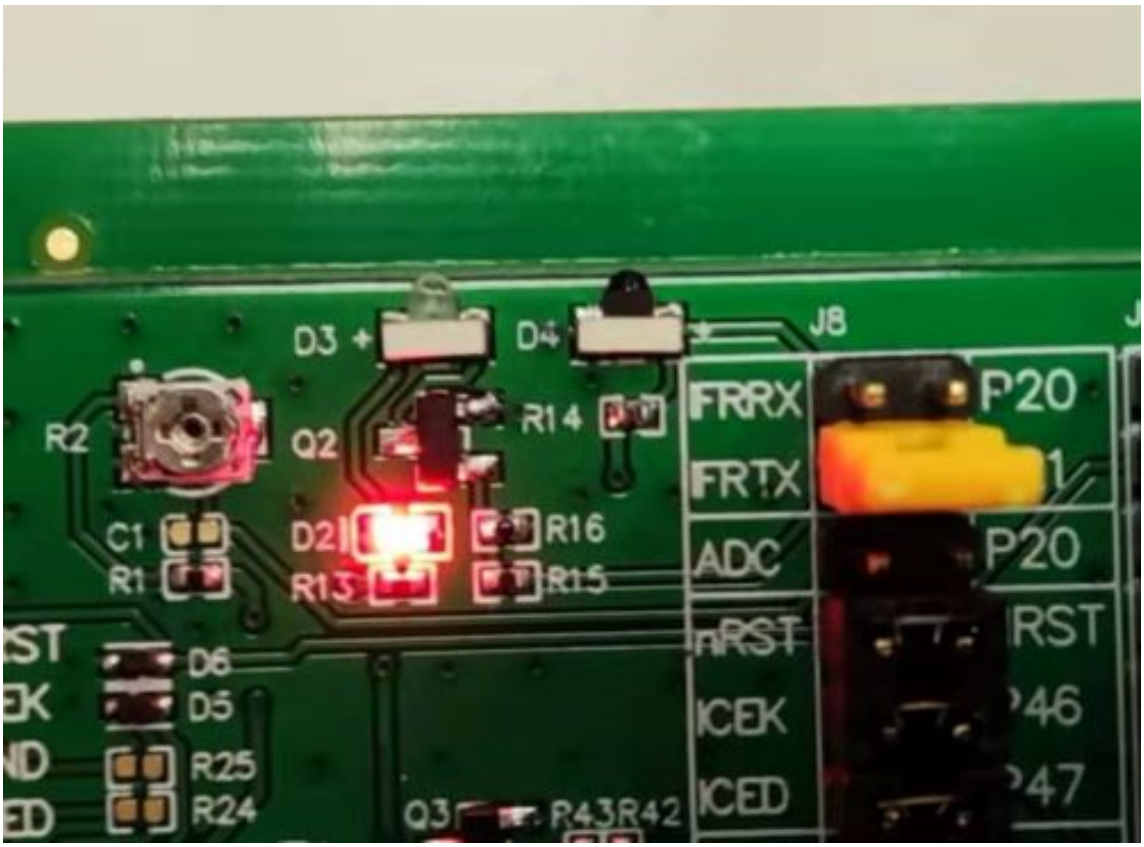
4. 双击 `my_led_blink.bat` 脚本，执行第一次编译，成功后，会显示成功生成名为 `zephyr.elf` 的输出文件，同时显示当前工程的 FLASH 和 SRAM 占用信息：



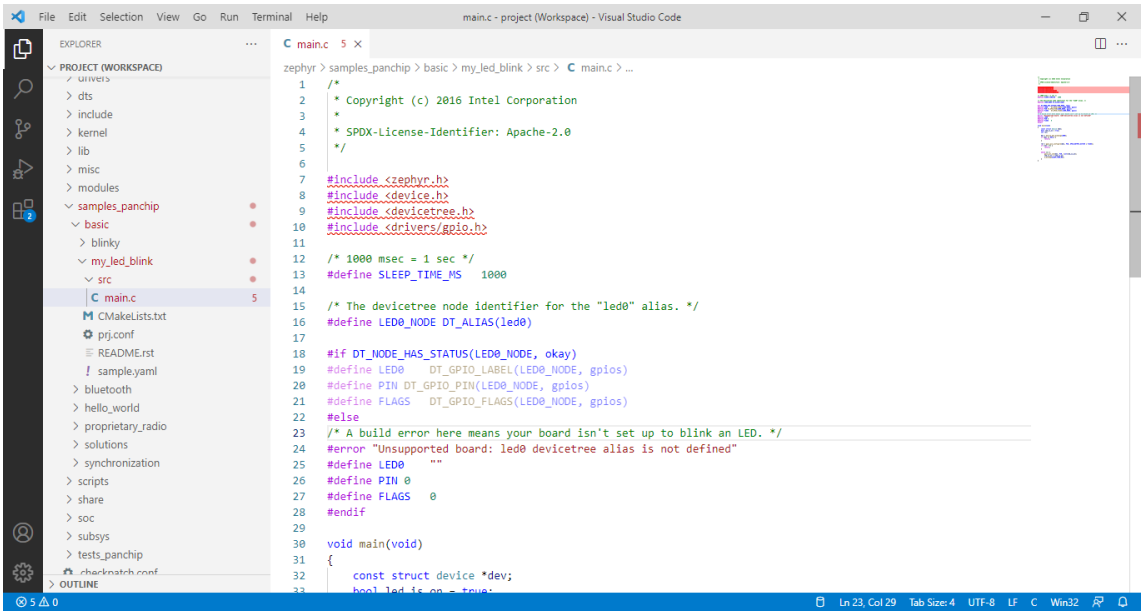
5. 输入指令 `f` 并回车，将程序烧录至芯片 Flash 中：



6. 烧录成功后，即可观察到 EVB 底板左上角的红色 LED 灯以 1Hz 的频率不断闪烁：



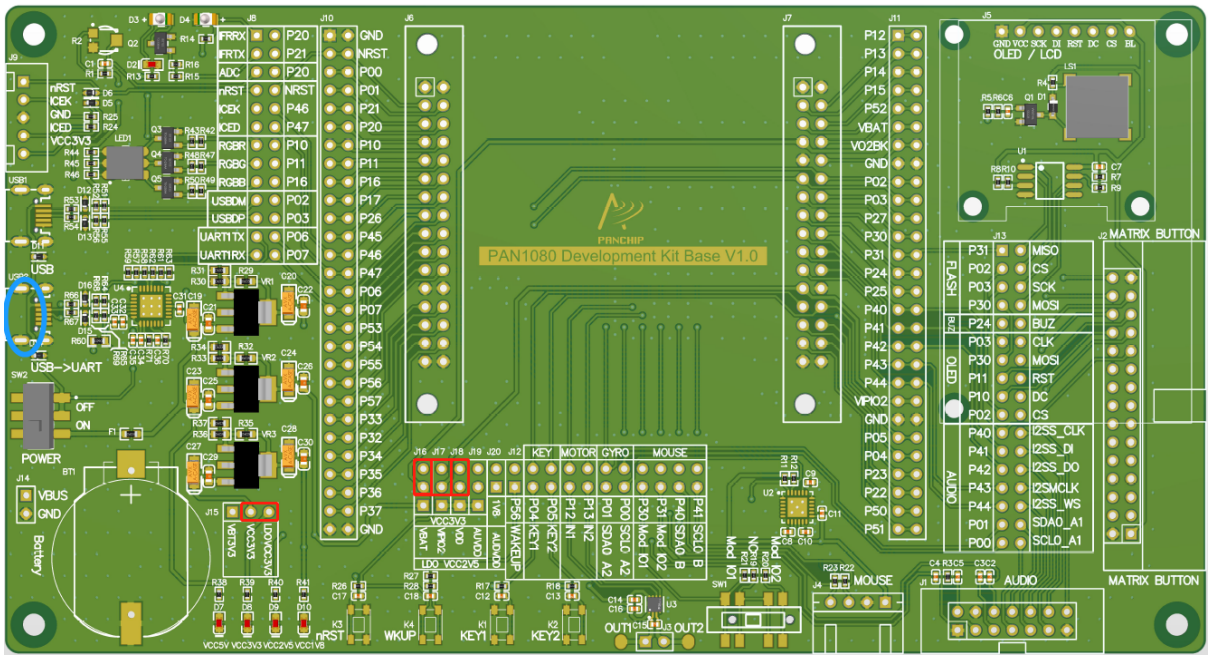
7. 最后，我们输入指令 `o`，启动 `VS Code` ，后续代码修改与编译烧录调试均可在 VS Code 环境下进行：



3.2 硬件资源规划

3.2.1 时钟与电源配置 在实际项目中我们需要确定 SoC 时钟源、内部各模块的时钟配置、电源选择等事项。但这里我们直接使用 EVB 板开发，因此暂不关注这些内容，使用 EVB 板的默认配置参数即可。

我们通过 PC 使用 MicroUSB 数据线连接至 EVB 开发板的 `USB to UART` 接口（如图**蓝色圆框**所示），为开发板提供 5V 电源；同时确认开发板相关电源跳线（如图**红色方框**所示）连接正确：



3.2.2 PWM 模块配置 我们使用 PAN1080 SoC 的 PWM 模块控制 EVB 底板上的 LED 灯，为此我们需要规划：

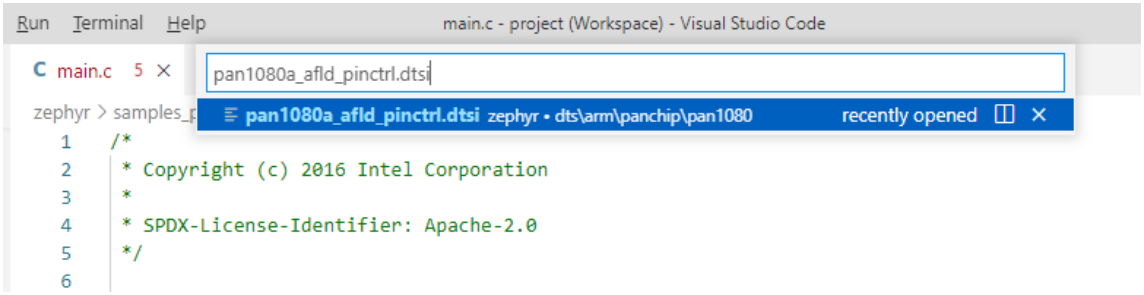
- 使用哪个 PWM 模块的哪个输出通道（PAN1080 SoC 中内置了 3 个硬件 PWM 模块，分别为 `PWM0`、`PWM1`、`PWM2`，每个模块均有 8 个通道）
- 将 PWM 波形输出到 SoC 的哪个引脚

需要注意的是，由于 PAN1080 SoC 的每个引脚最多只能配置成预设的 8 个功能之一，因此我们在规划引脚资源的时候，要查阅各个引脚的 PINMUX 定义，从中找到合适的引脚使用。

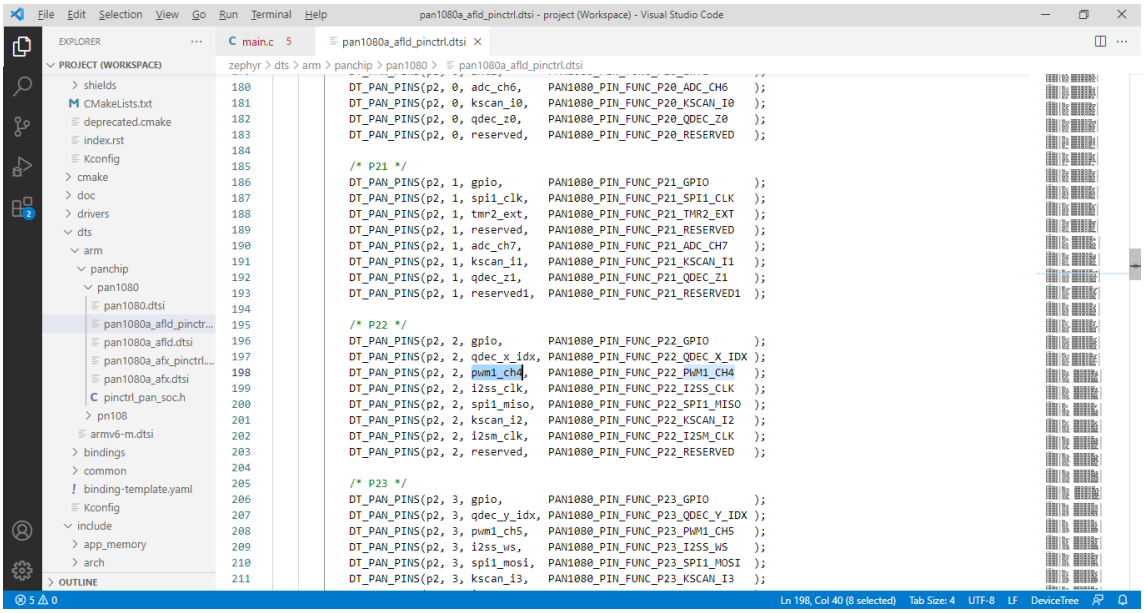
1. 我们可以从 PAN1080 Development Kit 中的如下几个文件的任意一个中找到 PAN1080 SoC 的 PINMUX 引脚定义：

   - `01_SDK/zephyr/dts/arm/panchip/pan1080/pan1080a_afld_pinctrl.dtsi`
   - `01_SDK/zephyr/include/drivers/pinmux/pinmux_pan1080.h`
   - `04_DOC/05_soc_manual/PAN1080 Datasheet.pdf` 或 `04_DOC/05_soc_manual/PAN1080 产品说明书.pdf`

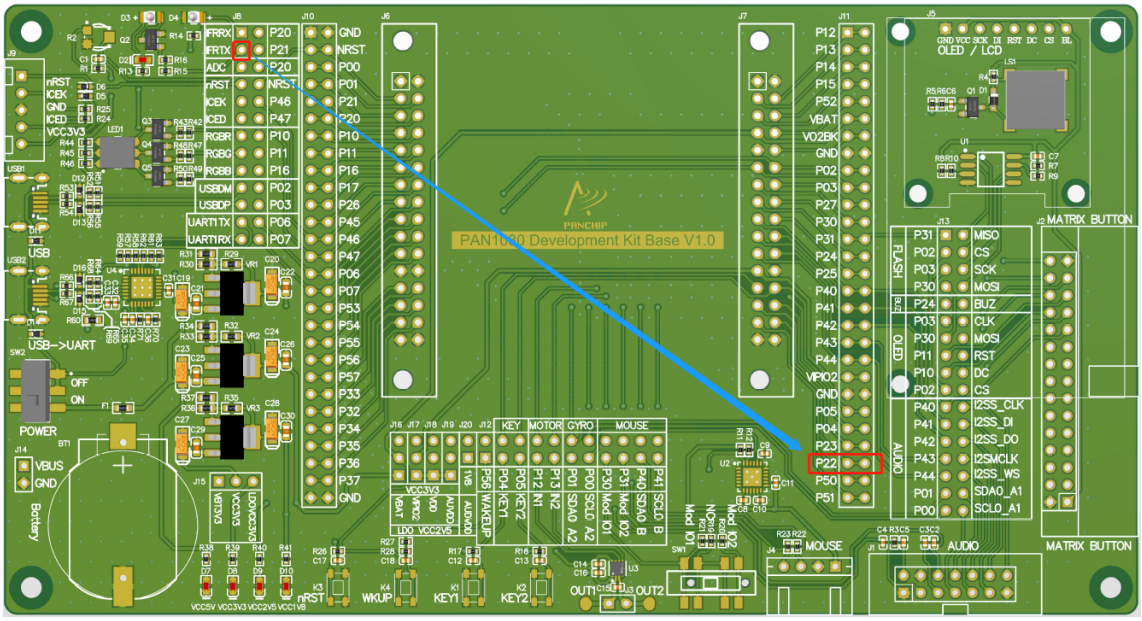
2. 这里，我们直接从 VS Code 中打开 `pan1080a_afld_pinctrl.dtsi` 文件（按快捷键 `Ctrl+P`，输入文件名即可）：



3. 由于当前 EVB 底板上的 LED 灯已经与 `P21` 引脚相连，我们先查找 `P21` 引脚的 PINMUX 定义，看是否有 PWM 功能；查阅后发现其并没有 PWM 功能，但相邻的 `P22` 引脚，有一个 `pwm1_ch4 (PWM1 Channel 4)` 的定义，可以供我们使用：

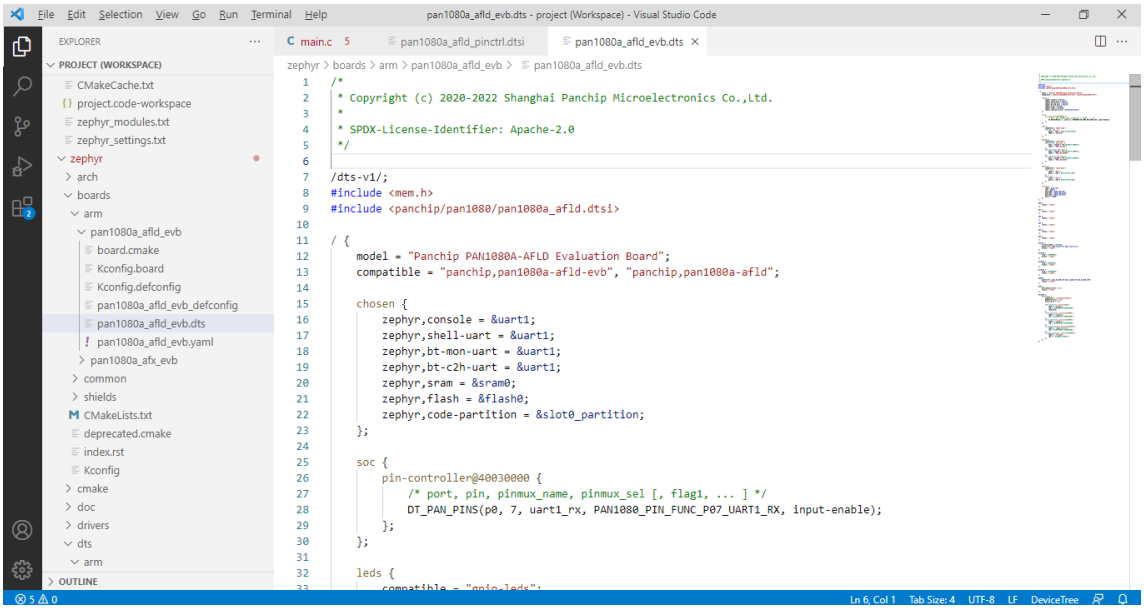


4. 在 EVB 底板上，将 `P21` 排针与 `FRTX` 排针连接的跳线帽拔出，另使用一根杜邦线将 `P22` 排针与 `FRTX` 排针相连：



5. 为使我们规划的硬件资源生效，我们需要修改 SDK 中的板级 DeviceTree 文件。

   从 VS Code 中打开名为 `pan1080a_afld_evb.dts` 的文件（按快捷键 `Ctrl+P`，输入文件名即可）：



   此文件中描述了关于 PAN1080A-AFLD EVB 的所有板级硬件配置信息，这些配置保证了我们可以使用 EVB 开发板成功运行 SDK 中的所有例程（`samples_panchip`）和测试用例（`tests_panchip`）。

   > 关于 Zephyr DeviceTree（`.dts`/`.dtsi`）的更多细节，请参考 Zephyr 官方文档：Introduction to devicetree

6. 我们暂不关心除 PWM 以外的模块配置，可以看到，`pan1080a_afld_evb.dts` 文件第 123~126 行描述了一个 `PWM0` 模块的配置：

```
...

&pwm0 {
        pinctrl-0 = <&p1_0_pwm0_ch4 &p1_1_pwm0_ch5 &p1_6_pwm0_ch6>;
        status = "okay";
};

...
```

其含义如下：

- 第一行 `&pwm0` 表示，当前配置文件中对 `pwm0` 中属性值的修改会更新默认配置文件（名为 `pan1080.dtsi`）中的值，亦即，若本次更新的属性不在默认配置文件中，则为此属性赋予一个值；若本次更新的属性已经在默认配置文件中已经有一个值，则覆盖此属性的默认值；
- 第二行 `pinctrl-0` 表示，修改当前 `pwm` 模块的 PINMUX 配置，其中尖括号中：
  - `&p1_0_pwm0_ch4` 表示将芯片 `P10` 引脚的 PINMUX 切换为 PWM0 Channel 4 功能；
  - `&p1_1_pwm0_ch5` 表示将芯片 `P11` 引脚的 PINMUX 切换为 PWM0 Channel 5 功能；
  - `&p1_6_pwm0_ch6` 表示将芯片 `P16` 引脚的 PINMUX 切换为 PWM0 Channel 6 功能；
- 第三行 `status = "okay"` 表示，在系统初始化过程中，使能 DeviceTree 中对当前 `PWM` 模块的各项参数配置；

7. 由于我们的规划是使用 `PWM1 Channel 4` 从芯片 `P22` 引脚输出方波，因此我们仿照上述对 `PWM0` 模块的描述，添加一个 `PWM1` 模块配置：

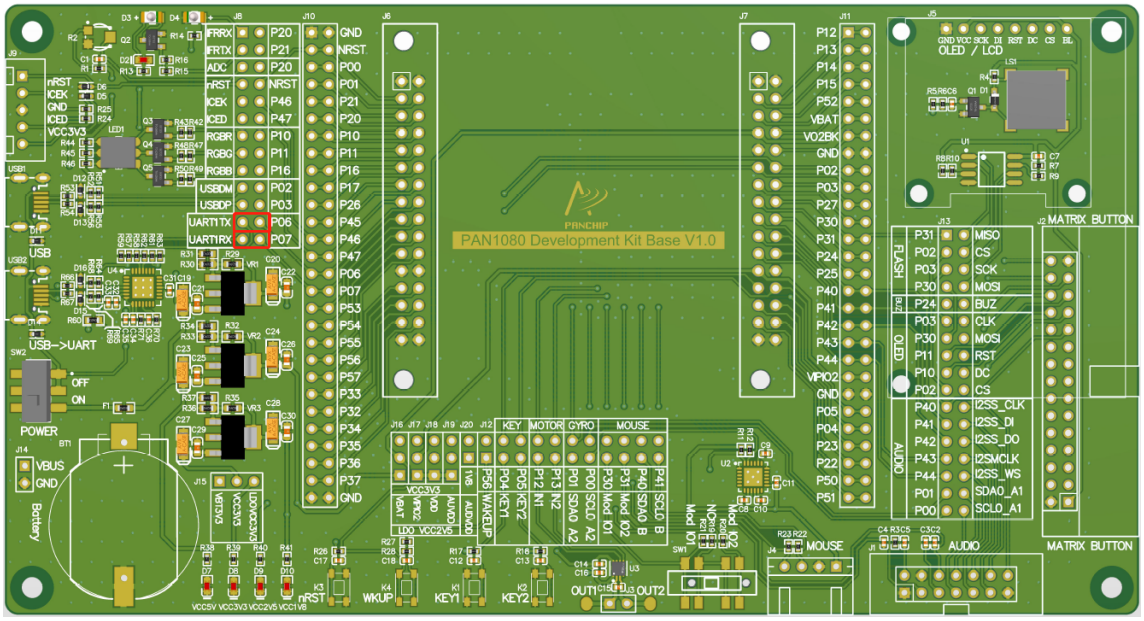
```
...

&pwm0 {
        pinctrl-0 = <&p1_0_pwm0_ch4 &p1_1_pwm0_ch5 &p1_6_pwm0_ch6>;
        status = "okay";
};

&pwm1 {
        pinctrl-0 = <&p2_2_pwm1_ch4>;
        status = "okay";
};

...
```

3.2.3 UART **模块配置** 我们希望使用 PAN1080 SoC 的 `UART` 模块输出 Log 日志到 PC，为此我们需要明确：

- 使用哪个 UART 模块（PAN1080 SoC 中内置了 2 个 UART 模块，分别为 `UART0`、`UART1`）
- UART Tx/Rx 分别映射到 SoC 的哪个引脚
- 如何与 PC 通信

PAN1080 EVB 底板中提供了一个 USB 转 UART 的模块电路，其可以很方便地通过跳线帽与 PAN1080 SoC 的 `P06`（`UART1 TX`）、`P07`（`UART1 RX`）相连，这里就采用这种方式与 PC 通信。

1. 使用跳线帽将 EVB 底板（左上侧）的 USB 转 UART 模块的 2 个通信排针连接至 PAN1080 SoC：



2. 再次在 VS Code 中打开板级 DeviceTree 文件 `pan1080a_afld_evb.dts`，我们可以看到其中已经有了一些与 UART 有关的配置：

```
...

        chosen {
                zephyr,console = &uart1;
                zephyr,shell-uart = &uart1;
                zephyr,bt-mon-uart = &uart1;
                zephyr,bt-c2h-uart = &uart1;
        ...
        };

        soc {
                pin-controller@40030000 {
                        /* port, pin, pinmux_name, pinmux_sel [, flag1, ... ] */
                        DT_PAN_PINS(p0, 7, uart1_rx, PAN1080_PIN_FUNC_P07_UART1_RX, input-
↪enable);
                };
        };

...

&uart1 {
        current-speed = <921600>;
        pinctrl-0 = <&p0_6_uart1_tx &p0_7_uart1_rx>;
        status = "okay";
};

...
```

其含义如下：

- `chosen` 节点描述了 Zephyr OS 用到的硬件模块与实际硬件的映射关系：
  - `zephyr,console = &uart1` 表示将 Zephyr Console 模块配置为使用 `UART1` 通信，我们此次向 UART 打印 LED 状态消息即使用此方式；
  - `zephyr,shell-uart = &uart1`、`zephyr,bt-mon-uart = &uart1`、`zephyr,bt-c2h-uart = &uart1` 均与本文内容无关，这里不做详细介绍，只需了解我们将 Zephyr 所有用到的串口均默认映射到了 PAN1080 SoC 的 `UART1` 模块即可；
- `soc` 节点描述了 SoC 级的硬件配置信息：

  - `pin-controller@40030000` 是一个子节点，代表 PAN1080 SoC 的 PINMUX（MFP）模块；
  - `DT_PAN_PINS(p0, 7, uart1_rx, PAN1080_PIN_FUNC_P07_UART1_RX, input-enable)` 用来辅助配置 `UART1` 的 PINMUX 参数；其中，此配置的前 4 个参数表示我们将芯片 `P07` 引脚的 PINMUX 切换为 UART1 RX 功能，最后一个参数表示辅助配置参数，这里 `input-enable` 表示打开此引脚的数字信号输入功能；

    注 1：除 `input-enable` 参数外，此处还可以配置 `bias-pull-up` 以使用当前引脚的**内部上拉**电阻，或配置 `bias-pull-down` 以使用当前引脚的**内部下拉**电阻；

    注 2：此处的 PINMUX 配置不是必须的，只有在需要打开某个引脚**数字信号输入**功能或**内部上拉/下拉电阻**功能的时候，才应该在此处进行配置，并且配置的引脚功能应当与对应外设模块中的 `pinxtrl-0` 中的配置一致，例如：

    * 我们在 DeviceTree 的 `uart1` 节点中将 Tx 功能配置到了 `P06` 引脚，将 Rx 功能配置到了 `P07` 引脚，因此我们需要在 `soc/pin-controller` 节点中将 `P07` 引脚的数字信号输入功能打开，但无需配置数字信号输出引脚 `P06`；
    * 如果我们使用 `I2C` 模块，则由于 I2C 协议要求有上拉电阻，因此我们可以配置 `bias-pull-up` 参数以使用 SoC 内部上拉电阻；如果我们将 `I2C` 模块配置为 Slave，则由于 `SCL` 和 `SDA` 两根线均为输入信号，因此还需要配置 `input-enable` 参数以打开这两个引脚的数字信号输入功能；

- `&uart1` 表示，修改 `uart1` 节点中的属性，且其中的修改会更新默认配置文件（名为 `pan1080.dtsi`）中的值：
  - `current-speed = <921600>` 表示，波特率配置为 921600；
  - `pinctrl-0 = <&p0_6_uart1_tx &p0_7_uart1_rx>` 表示：
    * 将芯片 `P06` 引脚的 PINMUX 切换为 UART1 TX 功能；
    * 将芯片 `P07` 引脚的 PINMUX 切换为 UART1 RX 功能（与 `soc/pin-controller` 节点中的配置一致）；
  - `status = "okay"` 表示，在系统初始化过程中，使能 DeviceTree 中对当前 `UART` 模块的各项参数配置；

3. 从 dts 文件中可知，SDK 中对于 UART 的配置已经与我们在 EVB 板中的接线一致，因此无需再做修改。

3.3 修改代码

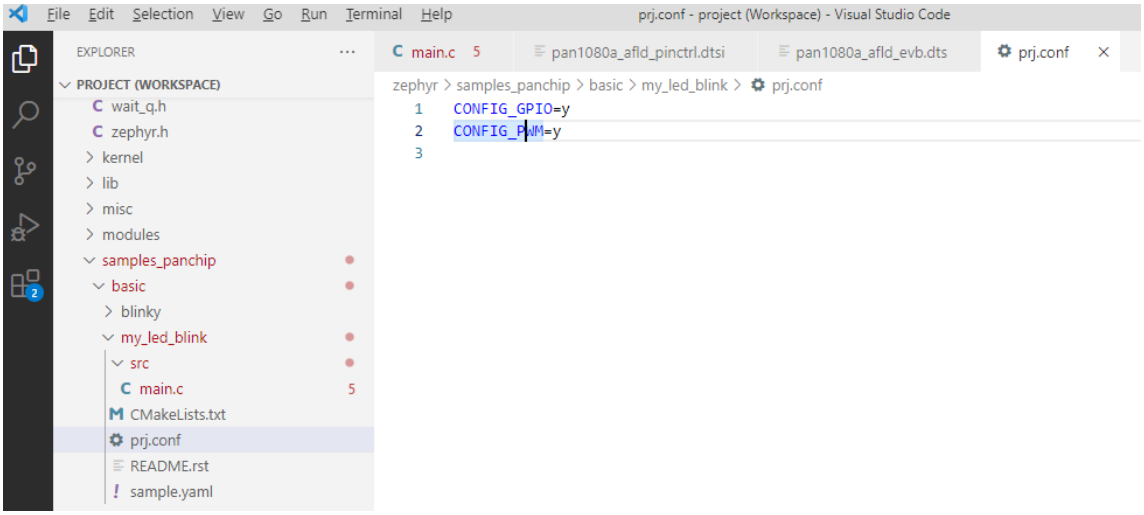
下面我们演示如何在 VS Code 环境下写 App 代码，实现预期功能。

3.3.1 使用 PWM

1. Zephyr 提供了标准的 PWM API 接口（头文件位置：`01_SDK/zephyr/include/drivers/pwm.h`），为了使用这些接口，我们需要先使能 Zephyr PWM Driver，方法是在 App 目录的 `prj.conf`，增加一个 Config 项：

   ```
   CONFIG_PWM=y
   ```

   增加后文件内容如图所示：



2. 接着，我们在 `main.c` 中，删除一些无关的代码，并增加 PWM 相关操作代码：

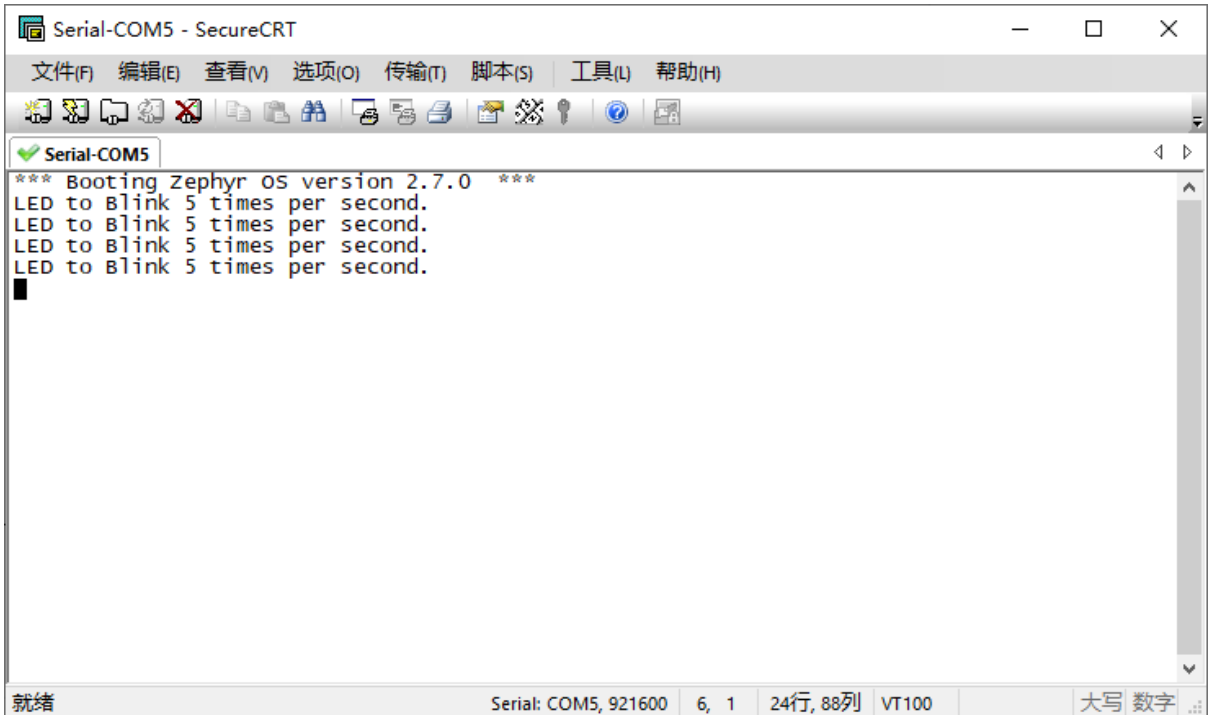
```
/*
 * Copyright (c) 2021-2022 Shanghai Panchip Microelectronics Co.,Ltd.
 *
 * SPDX-License-Identifier: Apache-2.0
 */

#include <zephyr.h>
#include <device.h>
#include <devicetree.h>
#include <drivers/gpio.h>
#include <drivers/pwm.h>

void main(void)
{
        const struct device *dev;

        dev = DEVICE_DT_GET(DT_NODELABEL(pwm1));
        if (dev == NULL) {
                return;
        }

        while (1) {
                /* LED Blink 5Hz */
                pwm_pin_set_usec(dev, 4, 200000, 100000, PWM_POLARITY_NORMAL);
                k_msleep(5000);
        }
}
```

3. 点击 VS Code 的 `Terminal` 菜单，选择 `Run Build Task...`（或直接快捷键 `Ctrl + Shift + B`），依次执行 Build 和 Flash 命令，将程序烧录至芯片中，即可观察到 LED 灯以 5Hz 的频率闪烁。

3.3.2 **使用 UART** 我们可以直接使用 zephyr 提供的 printk 函数，将 Log 信息输出至串口，修改 `while(1)` 循环内的代码如下：

```
        while (1) {
                printk("LED to Blink 5 times per second\n");
                pwm_pin_set_usec(dev, 4, 200000, 100000, PWM_POLARITY_NORMAL);
                k_msleep(5000);
        }
```

重新编译烧录后，在 PC 中使用串口工具（如 SecureCRT）即可看到每 5s 打印一次消息：



4.1.4 4 更多相关文档

1. Zephyr Kconfig 配置指南：介绍 Zephyr Kconfig 配置的相关技巧
2. Zephyr DeviceTree APIs：Zephyr 官方对于 DeviceTree 接口的详细介绍
3. Zephyr Peripheral APIs：Zephyr 官方提供的外设 Driver API 接口详细介绍

4.2 快速上手 BLE App 开发

本文将演示蓝牙的基础功能开发，以及如何添加新的 GATT 服务。

4.2.1 1 确认开发环境

参考SDK 快速入门，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。

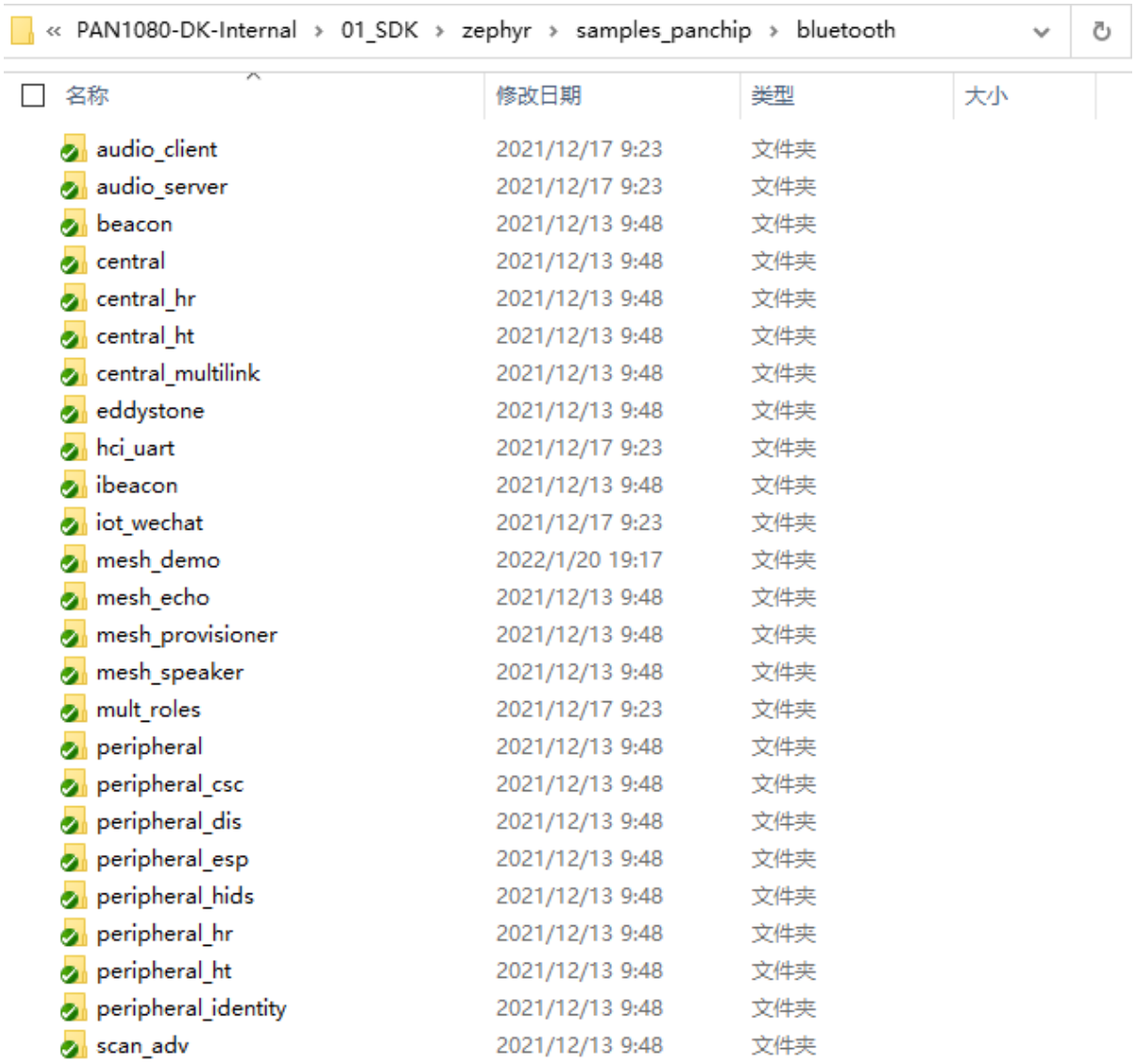
建议连接板载的 micro USB，通过串口工具监测 Log。

4.2.2 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 beacon、central、peripheral、hid、mesh 等。

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



4.2.3 3 新建一个蓝牙例程

在对蓝牙协议以及相关例程有一些了解之后，就可以尝试进行开发了，这里我们演示一个蓝牙收、发服务例程，具体服务描述如下：

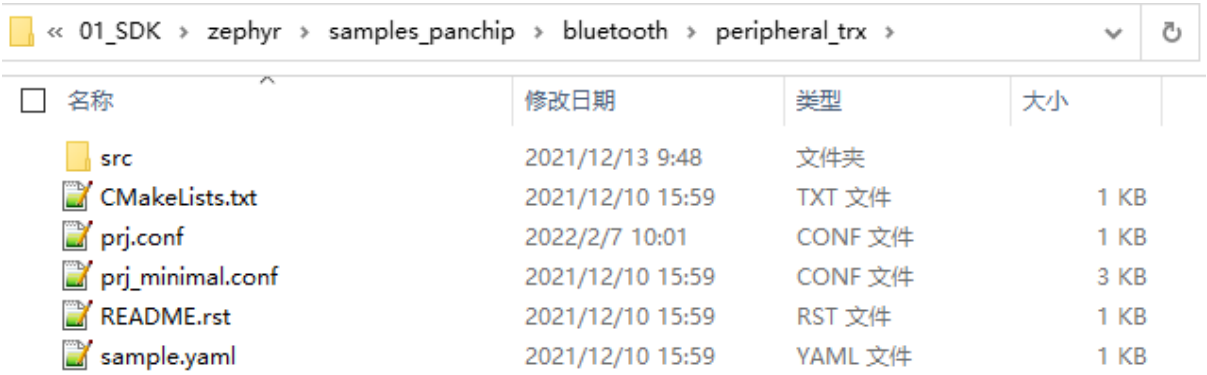
Service or Characteristic	UUID	Properties
T/Rx Service	0x12345678-1234-5678-1234-56789abcdef0	
Data From Client to Server Characteristic	0x12345678-1234-5678-1234-56789abcdef1	Write
Data From Server to Client Characteristic	0x12345678-1234-5678-1234-56789abcdef2	Read, Notify

3.1 拷贝一份相似的工程

(1) 拷贝一份与自己项目需求相近的例程

当然你也可以在原有例程上进行修改，初期并不建议开发者从头开始进行开发。

例程peripheral_hr 演示了简单的健康温度计 (Health Thermometer) 服务，可以在建立连接后将采集温度并上报给主机，我们将以这个例程为基础，来进行修改和补充，首先拷贝一份 `peripheral_hr` 并重命名为 `peripheral_trx`。



其中的必要的文件为:

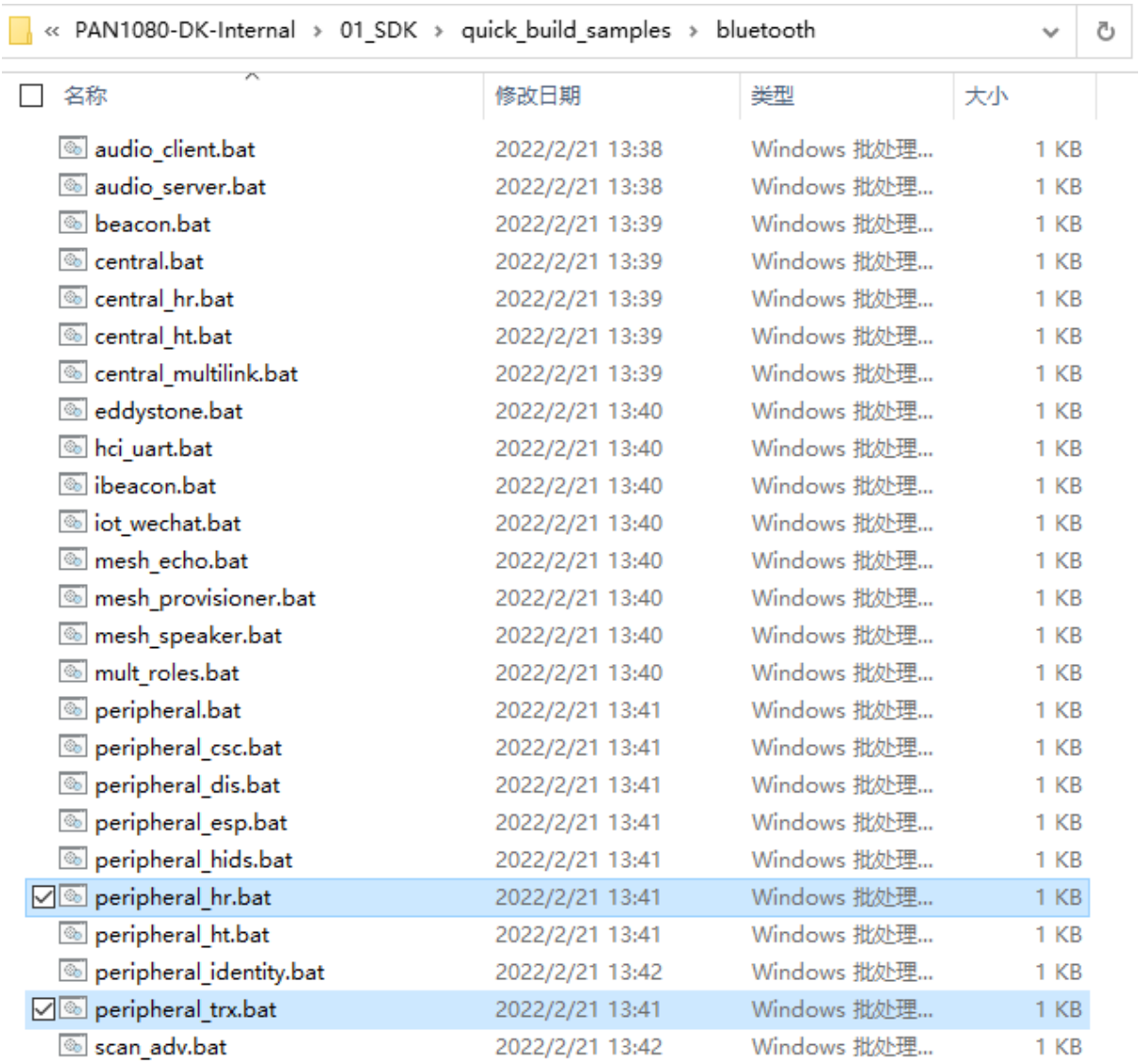
- `src\main.c`: 主程序代码
- `CMakeList.txt`: cmake 文件
- `prj.conf`: 项目配置文件

其它几个文件都不是必须的:

- `prj_minimal.conf`: 编译阶段可选的项目配置文件
- `README.rst`: 项目说明文件
- `sample.yaml`: 用于 zephyr tests 的文件

(2) 拷贝 `quick_build` 脚本，并重命名

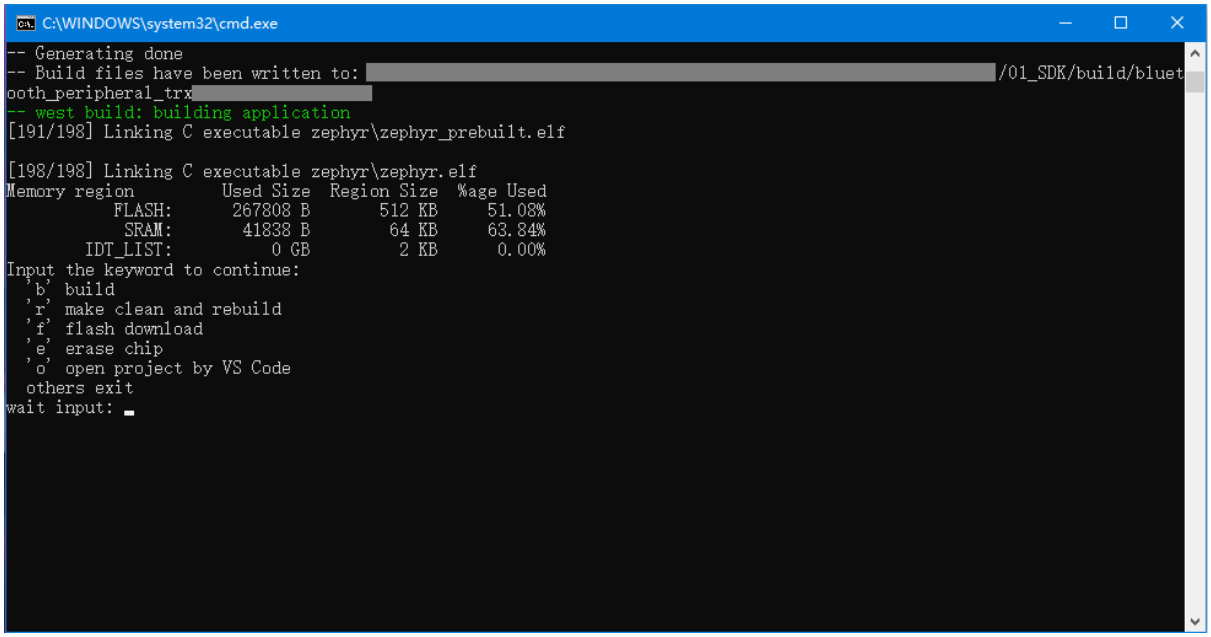
名称需要与项目名称保持一致。



(3) 运行修改后的编译脚本

这个步骤不是必须的，这里只是为了再次确认环境。

编译成功时，会有如下显示。

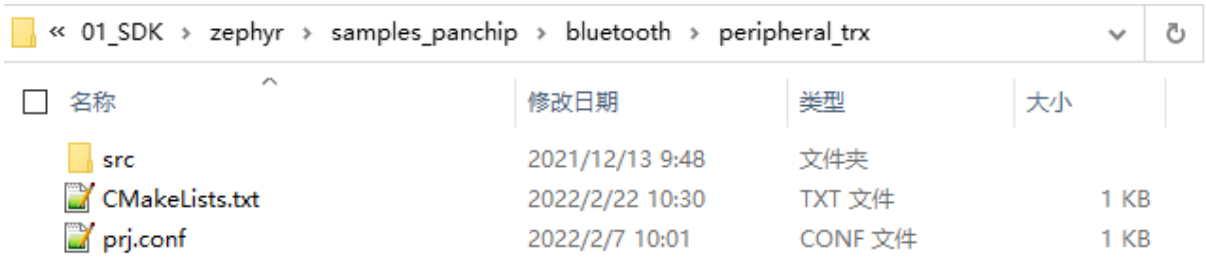


此时可以输入 o, 来启动 VS Code ，在其中修改代码和配置文件，编译和下载；也使用其它编辑工具开发，使用当前脚本进行编译和下载等。

3.2 进行必要的调整

(1) 为了简单起见，我们删掉不必要的文件。

删掉 prj_minimal.conf，README.rst，sample.yaml。



(2) 在 CMakeList.txt 文件中修改项目名称为 peripheral_trx。

```
## SPDX-License-Identifier: Apache-2.0

cmake_minimum_required(VERSION 3.20.0)

find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
project(peripheral_trx)

FILE(GLOB app_sources src/*.c)
target_sources(app PRIVATE
  ${app_sources}
  )

zephyr_library_include_directories(${ZEPHYR_BASE}/samples/bluetooth)
```

(3) 在 prj.conf 中修改项目的配置，关闭不必要的配置。

```
CONFIG_BT=y
CONFIG_BT_DEBUG_LOG=y
CONFIG_BT_PERIPHERAL=y
```

(下页继续)

(续上页)

```
CONFIG_BT_DEVICE_NAME="Peripheral TRx"
CONFIG_BT_DEVICE_NAME_DYNAMIC=y
```

- CONFIG_BT=y: 使能蓝牙
- CONFIG_BT_DEBUG_LOG=y: 使能蓝牙 log
- CONFIG_BT_PERIPHERAL=y: 使能 peripheral
- CONFIG_BT_DEVICE_NAME="Peripheral TRx": 修改蓝牙设备名称

(4) 在 main.c 中删掉不必要的代码，如原来代码中关于 BAS、HRS、AUTH 相关的内容。

```
#include <zephyr/types.h>
#include <stddef.h>
#include <string.h>
#include <errno.h>
#include <sys/printk.h>
#include <sys/byteorder.h>
#include <zephyr.h>

#include <bluetooth/bluetooth.h>
#include <bluetooth/hci.h>
#include <bluetooth/conn.h>
#include <bluetooth/uuid.h>
#include <bluetooth/gatt.h>

static const struct bt_data ad[] = {
        BT_DATA_BYTES(BT_DATA_FLAGS, (BT_LE_AD_GENERAL | BT_LE_AD_NO_BREDR)),
};

static void connected(struct bt_conn *conn, uint8_t err)
{
        if (err) {
                printk("Connection failed (err 0x%02x)\n", err);
        } else {
                printk("Connected\n");
        }
}

static void disconnected(struct bt_conn *conn, uint8_t reason)
{
        printk("Disconnected (reason 0x%02x)\n", reason);
}

BT_CONN_CB_DEFINE(conn_callbacks) = {
        .connected = connected,
        .disconnected = disconnected,
};

static void bt_ready(void)
{
        int err;

        printk("Bluetooth initialized\n");

        err = bt_le_adv_start(BT_LE_ADV_CONN_NAME, ad, ARRAY_SIZE(ad), NULL, 0);
        if (err) {
                printk("Advertising failed to start (err %d)\n", err);
                return;
        }

        printk("Advertising successfully started\n");
```

(下页继续)

(续上页)

```
}

void main(void)
{
        int err;

        err = bt_enable(NULL);
        if (err) {
                printk("Bluetooth init failed (err %d)\n", err);
                return;
        }

        bt_ready();
}
```

上述代码比较简单，但有个点需要说明一下：

- 你可以使用 BT_CONN_CB_DEFINE 来为连接事件注册回调函数，但却不需要对它进行额外的初始化；系统会将用这种方法定义的回调函数放到特定代码空间进行统一处理。
- 初次接触的开发者可能会感到困惑，但它会很方便，特别是多个文件都需要处理这个回调函数时，只需要按照同样的方式处理即可。

(5) 重新编译和下载例程

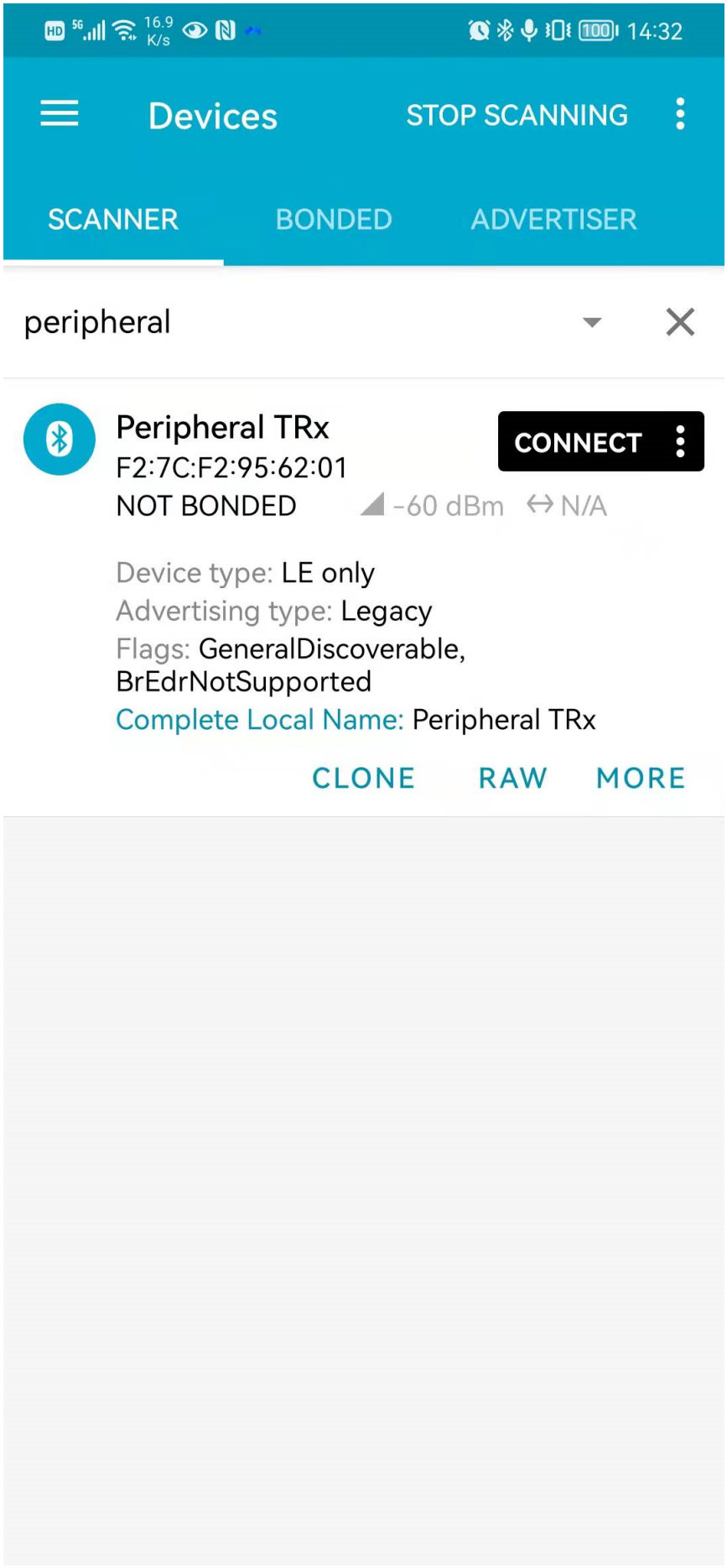
此时我们重新编译 (rebuild)，并下载。

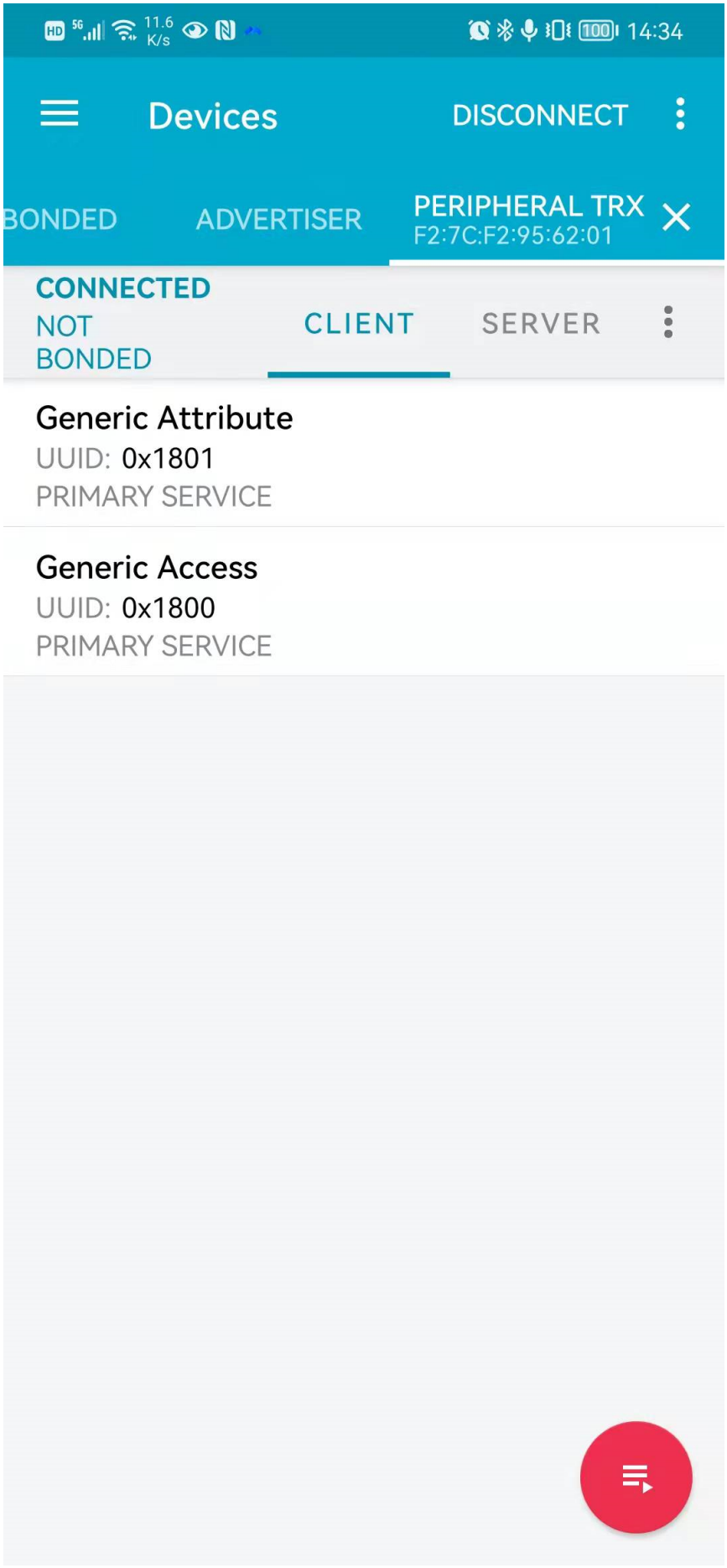
下载成功，系统执行起来以后，串口工具将显示如下 Log：

```
*** Booting Zephyr OS version 2.7.0  ***
Bluetooth initialized
Advertising successfully started
[00:00:00.186,000] <inf> bt_hci_core: Identity: C4:3E:A1:88:FD:9A (random)
[00:00:00.186,000] <inf> bt_hci_core: HCI: version 5.1 (0x0a) revision 0x0003, manufacturer␣
↪0x07d1
[00:00:00.187,000] <inf> bt_hci_core: LMP: version 5.1 (0x0a) subver 0x0000
```

手机端可以通过 nRF Connect APP 进行搜索、连接蓝牙设备

> 注意，非 HID 相关的例程，不建议通过手机设置界面查看和连接蓝牙设备，因为这可能涉及到一些手机系统的设备过滤和功能要求。


16.9
K/s
14:32
Devices
STOP SCANNING
SCANNER
BONDED
ADVERTISER
peripheral
Peripheral TRx
F2:7C:F2:95:62:01
NOT BONDED
-60 dBm
N/A
CONNECT
Device type: LE only
Advertising type: Legacy
Flags: GeneralDiscoverable,
BrEdrNotSupported
Complete Local Name: Peripheral TRx
CLONE
RAW
MORE


11.6
K/s
14:34
Devices
DISCONNECT
BONDED
ADVERTISER
PERIPHERAL TRX
F2:7C:F2:95:62:01
CONNECTED
NOT
BONDED
CLIENT
SERVER
Generic Attribute
UUID: 0x1801
PRIMARY SERVICE
Generic Access
UUID: 0x1800
PRIMARY SERVICE

3.3 添加 GATT 服务

(1) 定义 UUID

```
/* TRx Service Variables */
#define BT_UUID_TRX_SERVICE_VAL \
        BT_UUID_128_ENCODE(0x12345678, 0x1234, 0x5678, 0x1234, 0x56789abcdef0)

static struct bt_uuid_128 trx_svc_uuid = BT_UUID_INIT_128(
        BT_UUID_TRX_SERVICE_VAL);

static struct bt_uuid_128 trx_c2s_uuid = BT_UUID_INIT_128(
        BT_UUID_128_ENCODE(0x12345678, 0x1234, 0x5678, 0x1234, 0x56789abcdef1));

static struct bt_uuid_128 trx_s2c_uuid = BT_UUID_INIT_128(
        BT_UUID_128_ENCODE(0x12345678, 0x1234, 0x5678, 0x1234, 0x56789abcdef2));
```

这里把 BT_UUID_TRX_SERVICE_VAL 单独列出来，是为了后面加到广播数据中，以暴露此服务。

(2) 定义 GATT 服务

可以使用 BT_GATT_SERVICE_DEFINE 来定义服务，它和前面介绍的 BT_CONN_CB_DEFINE 一样，将被系统放到特定代码空间，不需要额外初始化。

定义了三个 buffer 来存放待收发的数据：trx_c2s_value，trx_s2c_value，trx_s2c_ntf_value。

定义了三个接口，处理数据收发：write_trx_c2s，read_trx_s2c，trx_s2c_ccc_cfg_changed。

GATT 服务一般要包含：BT_GATT_PRIMARY_SERVICE, BT_GATT_CHARACTERISTIC，包含 notify / indicate 时，还需要用到 BT_GATT_CCC 来进行管理是否使能。

更多的示例，请参考例程 bluetooth\peripheral。

```
#define TRX_MAX_LEN 20

static uint8_t trx_c2s_value[TRX_MAX_LEN + 1] = {
        'T', 'R', 'X', '_', 'C', '2', 'S'
};
static uint8_t trx_s2c_value[TRX_MAX_LEN + 1] = {
        'T', 'R', 'X', '_', 'S', '2', 'C'
};
static uint32_t trx_s2c_ntf_value;

static ssize_t write_trx_c2s(struct bt_conn *conn, const struct bt_gatt_attr *attr,
                         const void *buf, uint16_t len, uint16_t offset,
                         uint8_t flags)
{
        uint8_t *value = attr->user_data;

        if (offset + len > TRX_MAX_LEN) {
                return BT_GATT_ERR(BT_ATT_ERR_INVALID_OFFSET);
        }

        memcpy(value + offset, buf, len);
        value[offset + len] = 0;

        printk("write (%d): ", len);
        for (uint8_t i = 0; i < len; i++)
        {
                printk("0x%02X ", value[i]);
        }
        printk("\n");

        return len;
```

(下页继续)

(续上页)

```
}

static ssize_t read_trx_s2c(struct bt_conn *conn, const struct bt_gatt_attr *attr,
                        void *buf, uint16_t len, uint16_t offset)
{
        const char *value = attr->user_data;

        return bt_gatt_attr_read(conn, attr, buf, len, offset, value,
                                 strlen(value));
}

static uint8_t simulate;

static void trx_s2c_ccc_cfg_changed(const struct bt_gatt_attr *attr, uint16_t value)
{
        simulate = (value == BT_GATT_CCC_NOTIFY) ? 1 : 0;
}

/* TRx Primary Service Declaration */
BT_GATT_SERVICE_DEFINE(trx_svc,
        BT_GATT_PRIMARY_SERVICE(&trx_svc_uuid),
        BT_GATT_CHARACTERISTIC(&trx_c2s_uuid.uuid,
                               BT_GATT_CHRC_WRITE, BT_GATT_PERM_WRITE,
                               NULL, write_trx_c2s, trx_c2s_value),
        BT_GATT_CHARACTERISTIC(&trx_s2c_uuid.uuid,
                               BT_GATT_CHRC_READ | BT_GATT_CHRC_NOTIFY,
                               BT_GATT_PERM_READ,
                               read_trx_s2c, NULL, trx_s2c_value),
        BT_GATT_CCC(trx_s2c_ccc_cfg_changed,
                    BT_GATT_PERM_READ | BT_GATT_PERM_WRITE),
);
```

(3) 定义接口

这里定义了两个接口：

- `trx_init`：初始化 notify attribute
- `trx_notify`：上报数据给 client

```
static struct bt_gatt_attr *trx_ntf_attr;

static void trx_init(void)
{
        trx_ntf_attr = bt_gatt_find_by_uuid(trx_svc.attrs, trx_svc.attr_count, &trx_s2c_uuid.
↪uuid);
}

static void trx_notify(void)
{
        if (simulate) {
                bt_gatt_notify(NULL, trx_ntf_attr, &trx_s2c_ntf_value, sizeof(trx_s2c_ntf_
↪value));
                trx_s2c_ntf_value++;
        }
}
```

(4) 修改广播数据

添加了 TRx Service UUID，以便在广播数据中发现它。

```
static const struct bt_data ad[] = {
        BT_DATA_BYTES(BT_DATA_FLAGS, (BT_LE_AD_GENERAL | BT_LE_AD_NO_BREDR)),
```

(下页继续)

(续上页)

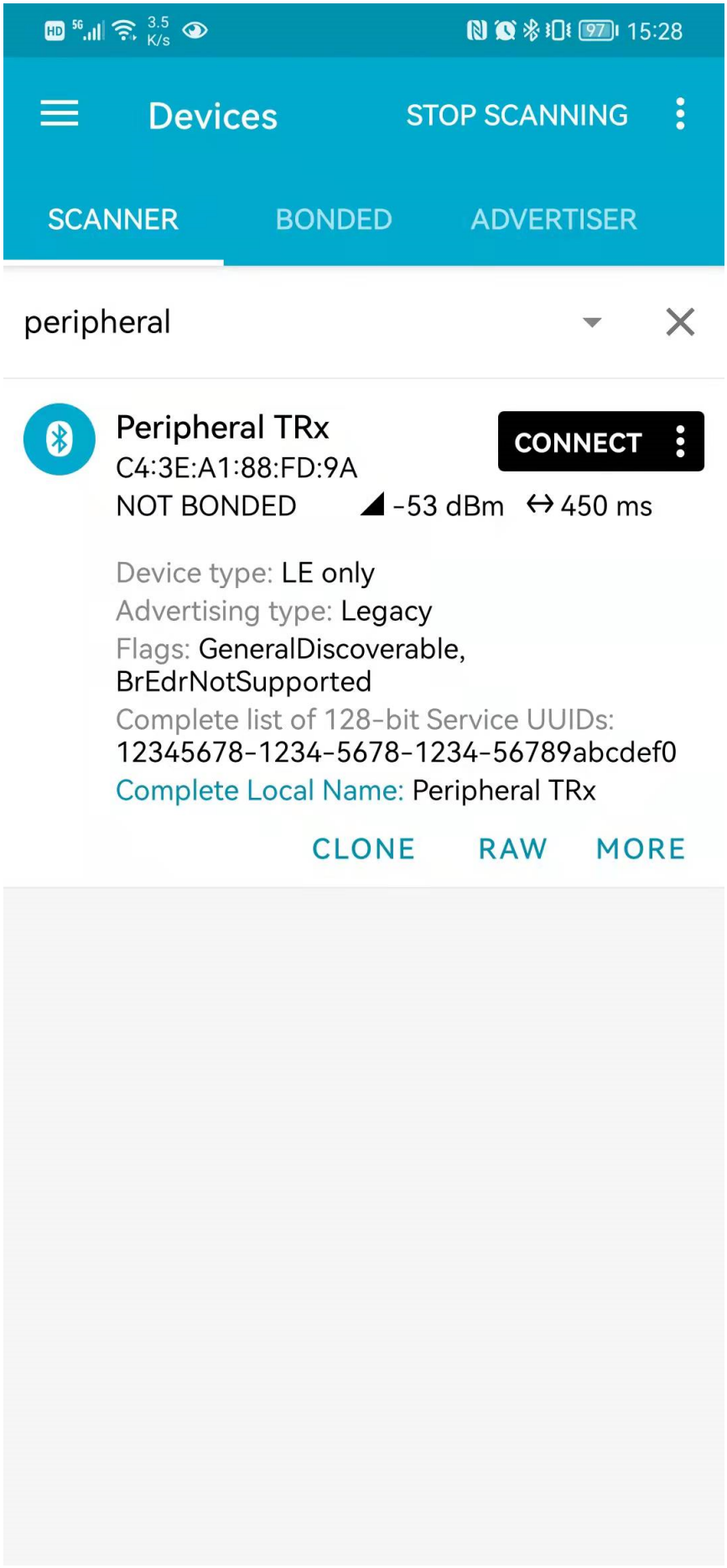
```
        BT_DATA_BYTES(BT_DATA_UUID128_ALL, BT_UUID_TRX_SERVICE_VAL),
};
```

(5) 初始化 trx service 并在连接建立且使能 notify 后上报数据

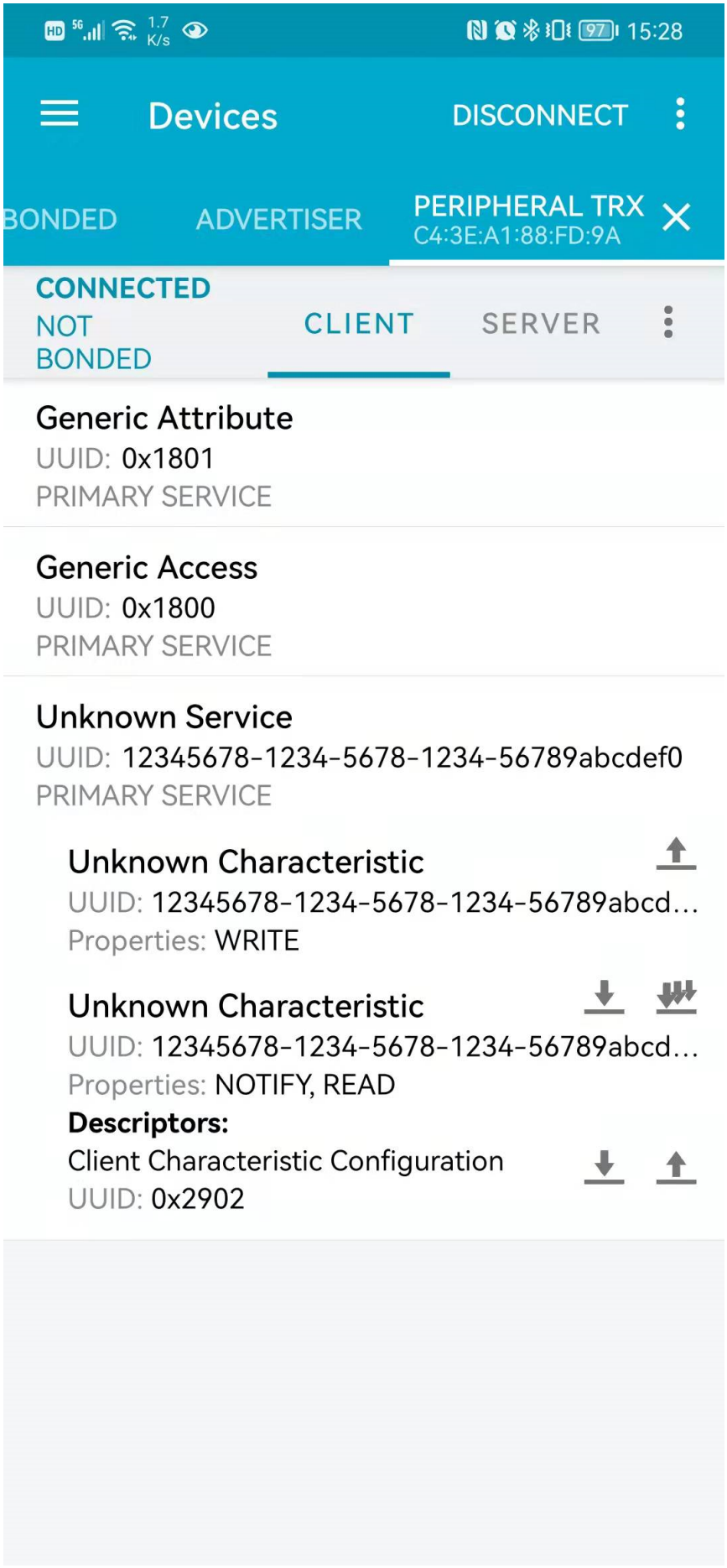
```
void main(void)
{
        int err;

        err = bt_enable(NULL);

        trx_init();

        if (err) {
                printk("Bluetooth init failed (err %d)\n", err);
                return;
        }

        bt_ready();

        while (1) {
                k_sleep(K_SECONDS(1));

                trx_notify();
        }
}
```

3.4 功能演示

(1) 扫描设备


3.5
K/s
15:28
Devices
STOP SCANNING
SCANNER
BONDED
ADVERTISER
peripheral
Peripheral TRx
C4:3E:A1:88:FD:9A
CONNECT
NOT BONDED
-53 dBm
450 ms
Device type: LE only
Advertising type: Legacy
Flags: GeneralDiscoverable,
BrEdrNotSupported
Complete list of 128-bit Service UUIDs:
12345678-1234-5678-1234-56789abcdef0
Complete Local Name: Peripheral TRx
CLONE
RAW
MORE

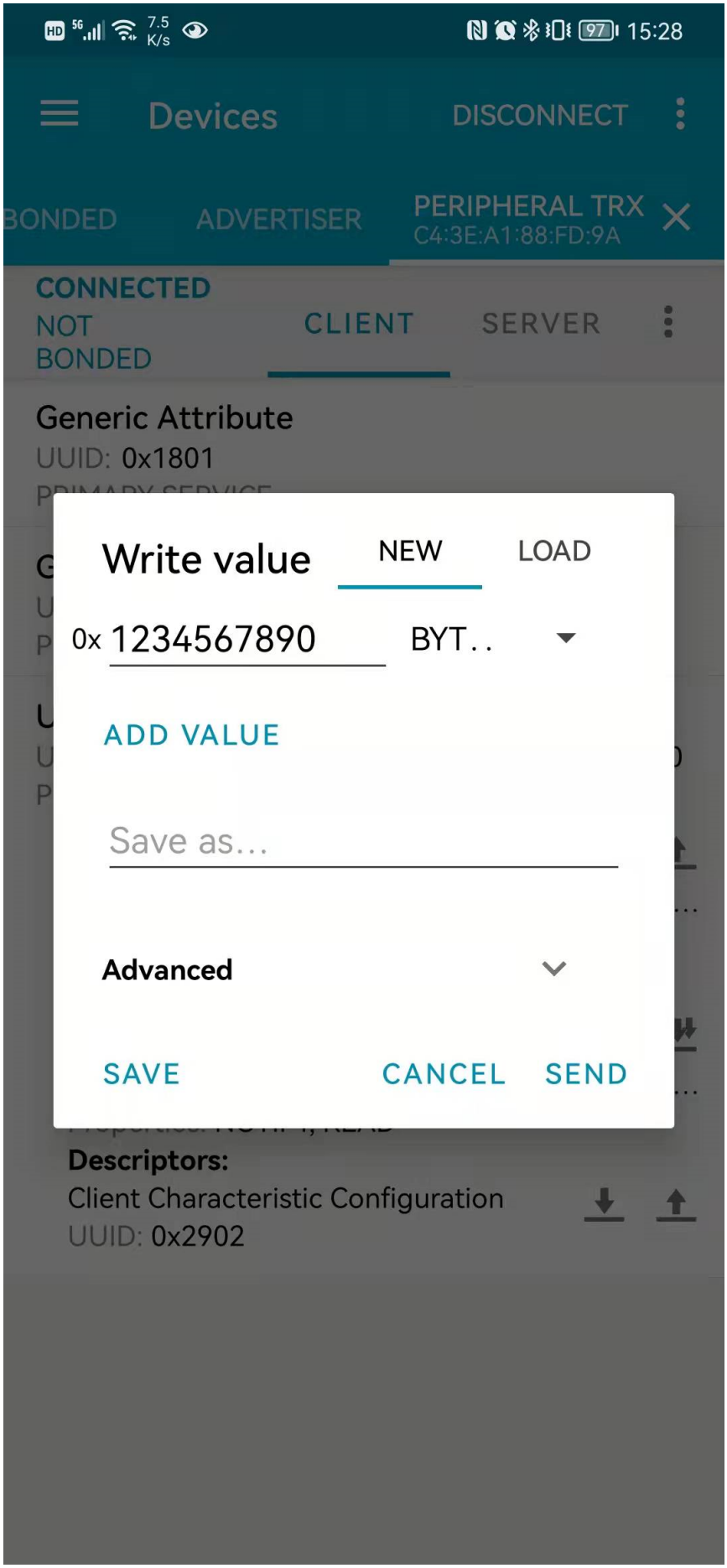
(2) 建立连接


1.7 K/s
15:28
Devices
DISCONNECT
BONDED
ADVERTISER
PERIPHERAL TRX
C4:3E:A1:88:FD:9A
CONNECTED
NOT BONDED
CLIENT
SERVER
Generic Attribute
UUID: 0x1801
PRIMARY SERVICE
Generic Access
UUID: 0x1800
PRIMARY SERVICE
Unknown Service
UUID: 12345678-1234-5678-1234-56789abcdef0
PRIMARY SERVICE
Unknown Characteristic
UUID: 12345678-1234-5678-1234-56789abcd...
Properties: WRITE
Unknown Characteristic
UUID: 12345678-1234-5678-1234-56789abcd...
Properties: NOTIFY, READ
Descriptors:
Client Characteristic Configuration
UUID: 0x2902

串口工具将打印连接状态变化的信息：

```
Connected
```

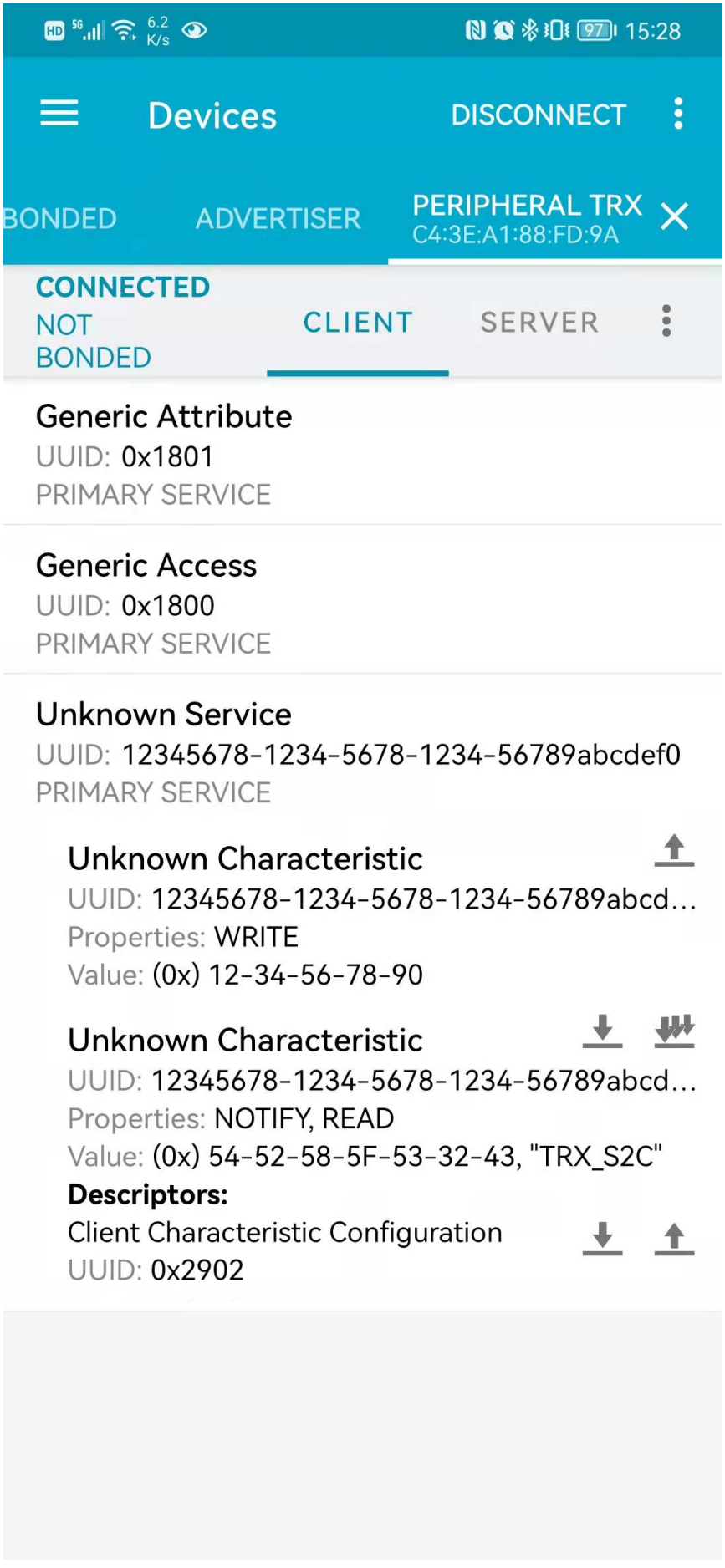
(3) 写数据


7.5 K/s
15:28
Devices
DISCONNECT
BONDED
ADVERTISER
PERIPHERAL TRX
C4:3E:A1:88:FD:9A
CONNECTED
NOT BONDED
CLIENT
SERVER
Generic Attribute
UUID: 0x1801
Write value
NEW
LOAD
0x 1234567890
BYT..
ADD VALUE
Save as...
Advanced
SAVE
CANCEL
SEND
Descriptors:
Client Characteristic Configuration
UUID: 0x2902

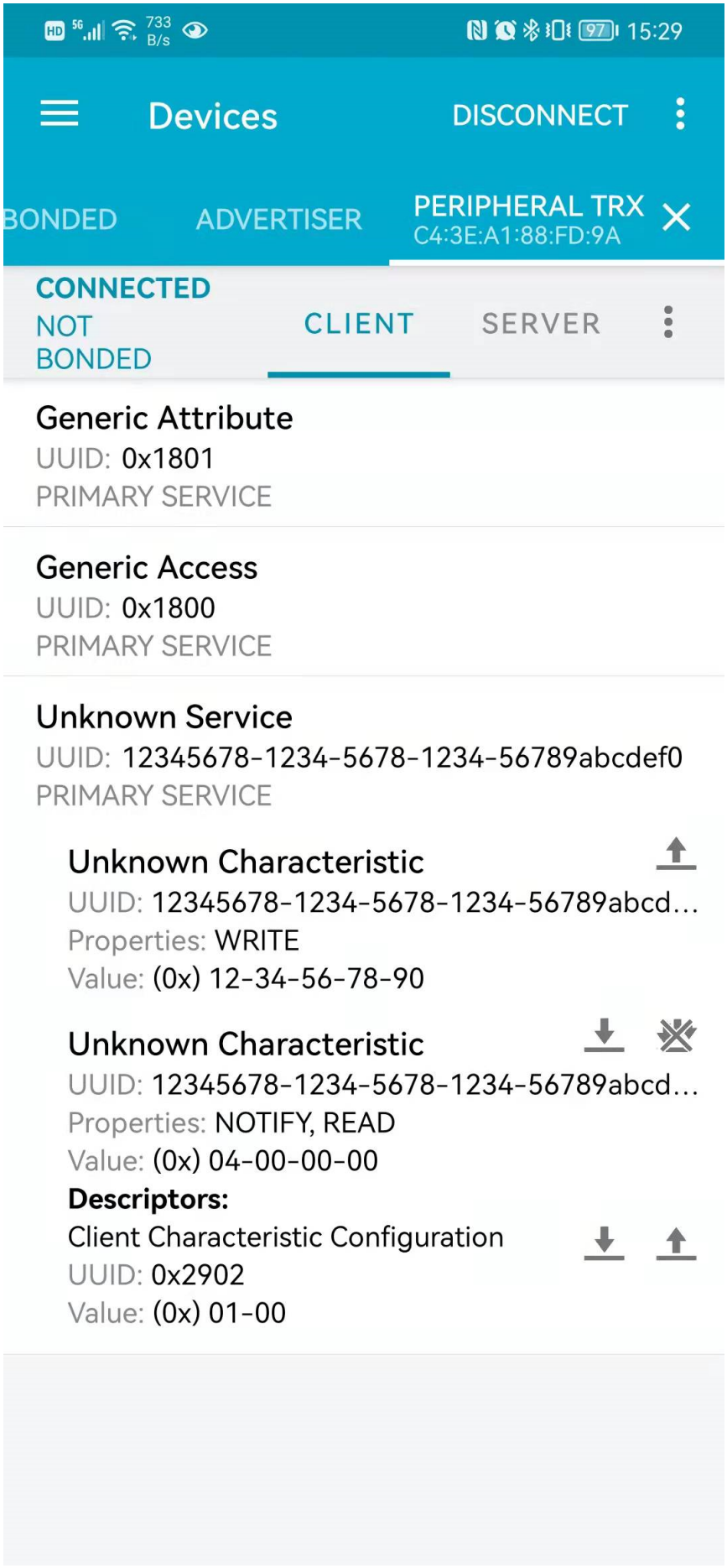
串口工具将打印收到的数据：

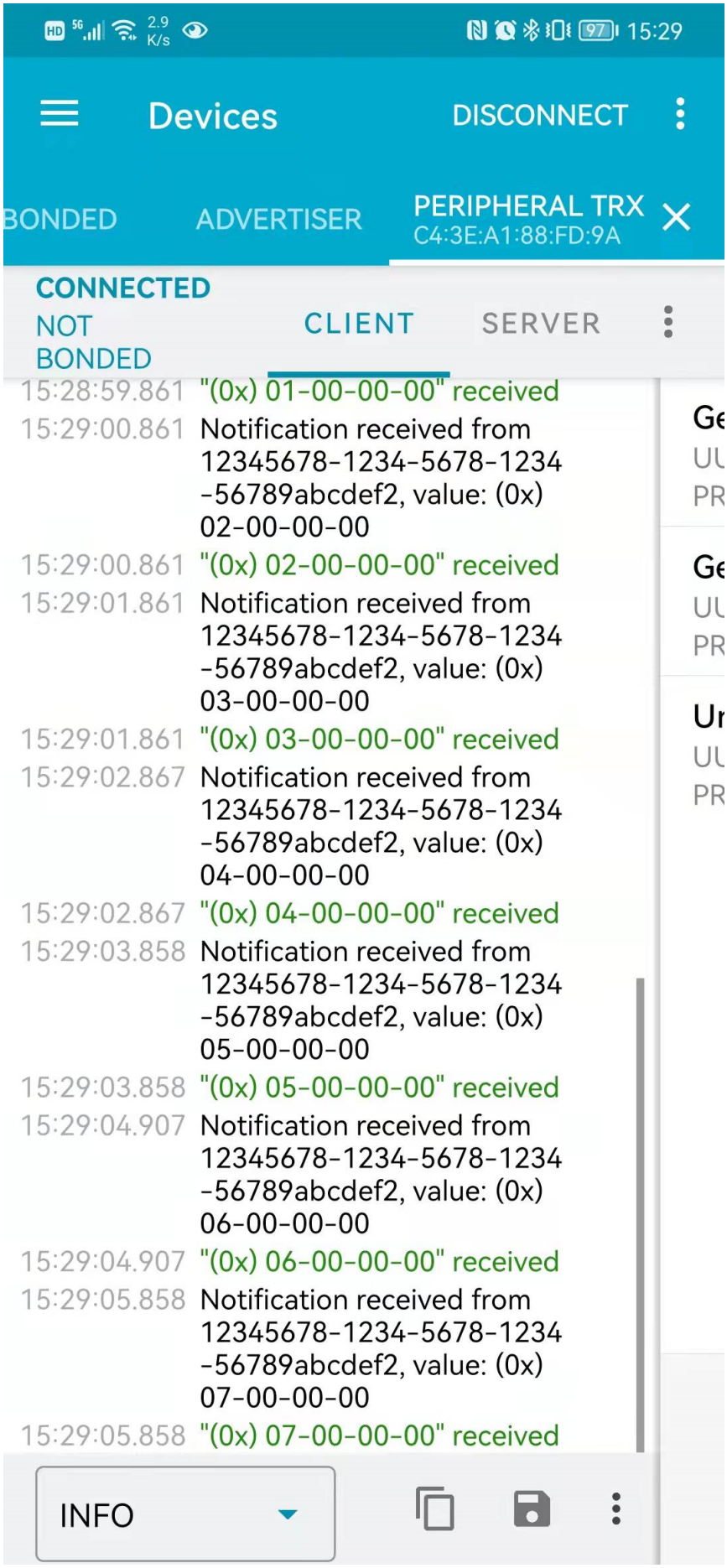
```
write (5): 0x12 0x34 0x56 0x78 0x90
```

(4) 读数据


6.2 K/s
15:28
Devices
DISCONNECT
BONDED
ADVERTISER
PERIPHERAL TRX
C4:3E:A1:88:FD:9A
CONNECTED
NOT BONDED
CLIENT
SERVER
Generic Attribute
UUID: 0x1801
PRIMARY SERVICE
Generic Access
UUID: 0x1800
PRIMARY SERVICE
Unknown Service
UUID: 12345678-1234-5678-1234-56789abcdef0
PRIMARY SERVICE
Unknown Characteristic
UUID: 12345678-1234-5678-1234-56789abcd...
Properties: WRITE
Value: (0x) 12-34-56-78-90
Unknown Characteristic
UUID: 12345678-1234-5678-1234-56789abcd...
Properties: NOTIFY, READ
Value: (0x) 54-52-58-5F-53-32-43, "TRX_S2C"
Descriptors:
Client Characteristic Configuration
UUID: 0x2902

(5) Notify


733 B/s
15:29
Devices
DISCONNECT
BONDED
ADVERTISER
PERIPHERAL TRX
C4:3E:A1:88:FD:9A
CONNECTED
NOT BONDED
CLIENT
SERVER
Generic Attribute
UUID: 0x1801
PRIMARY SERVICE
Generic Access
UUID: 0x1800
PRIMARY SERVICE
Unknown Service
UUID: 12345678-1234-5678-1234-56789abcdef0
PRIMARY SERVICE
Unknown Characteristic
UUID: 12345678-1234-5678-1234-56789abcd...
Properties: WRITE
Value: (0x) 12-34-56-78-90
Unknown Characteristic
UUID: 12345678-1234-5678-1234-56789abcd...
Properties: NOTIFY, READ
Value: (0x) 04-00-00-00
Descriptors:
Client Characteristic Configuration
UUID: 0x2902
Value: (0x) 01-00


15:29
Devices
DISCONNECT
BONDED
ADVERTISER
PERIPHERAL TRX
C4:3E:A1:88:FD:9A
CONNECTED
NOT BONDED
CLIENT
SERVER
15:28:59.861 "(0x) 01-00-00-00" received
15:29:00.861 Notification received from 12345678-1234-5678-1234-56789abcdef2, value: (0x) 02-00-00-00
15:29:00.861 "(0x) 02-00-00-00" received
15:29:01.861 Notification received from 12345678-1234-5678-1234-56789abcdef2, value: (0x) 03-00-00-00
15:29:01.861 "(0x) 03-00-00-00" received
15:29:02.867 Notification received from 12345678-1234-5678-1234-56789abcdef2, value: (0x) 04-00-00-00
15:29:02.867 "(0x) 04-00-00-00" received
15:29:03.858 Notification received from 12345678-1234-5678-1234-56789abcdef2, value: (0x) 05-00-00-00
15:29:03.858 "(0x) 05-00-00-00" received
15:29:04.907 Notification received from 12345678-1234-5678-1234-56789abcdef2, value: (0x) 06-00-00-00
15:29:04.907 "(0x) 06-00-00-00" received
15:29:05.858 Notification received from 12345678-1234-5678-1234-56789abcdef2, value: (0x) 07-00-00-00
15:29:05.858 "(0x) 07-00-00-00" received
INFO

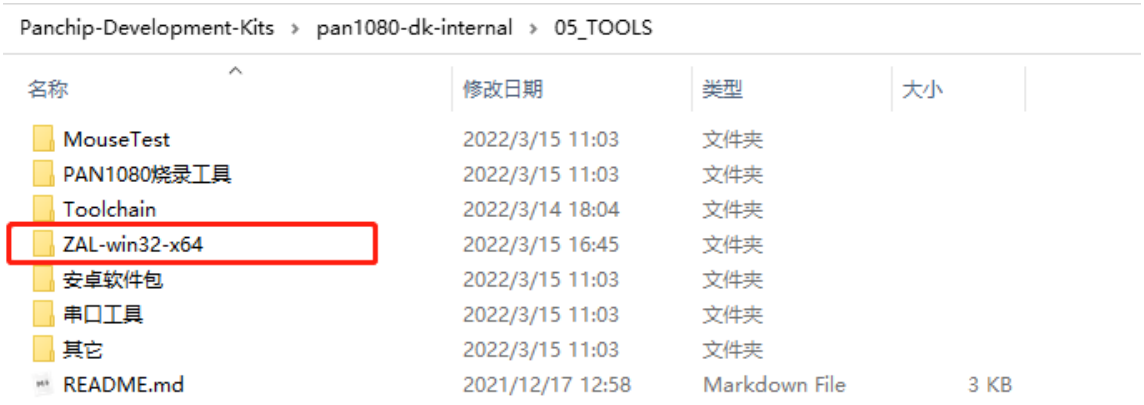
4.3 App Launcher for PAN1080 工具介绍

4.3.1 1 概述

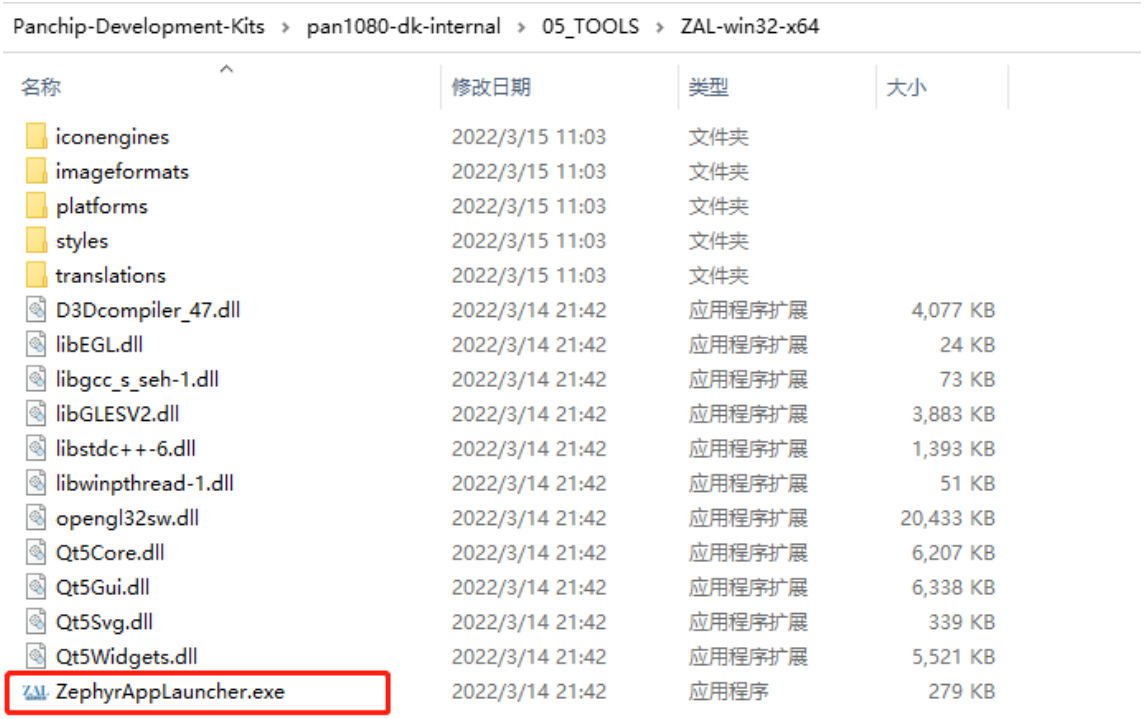
Zephyr App Launcher (以下简称 ZAL) 是 *Shanghai Panchip Microelectronics Co.,Ltd.* 为 PAN1080 SDK 提供的入口界面软件，其设计目标是以 Windows GUI 的方式，简化 PAN1080 SDK 中 App（例程与测试用例等）的编译烧录过程，同时提供了通向 IDE 集成开发环境（VS Code）的入口。

4.3.2 2 获取并打开软件

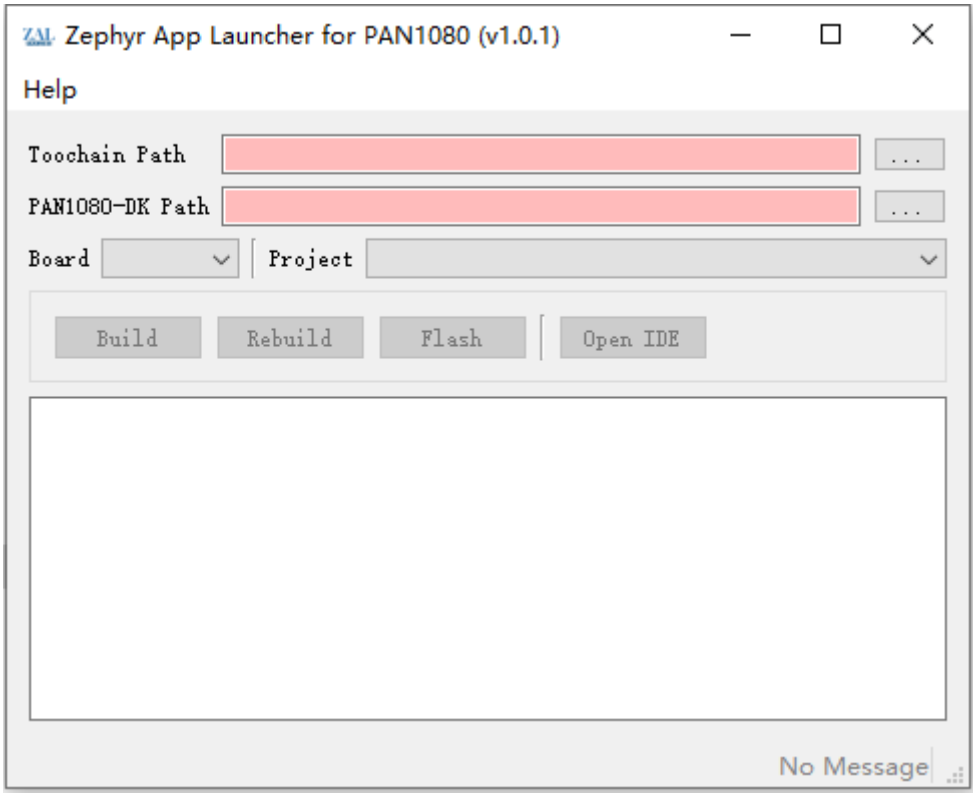
1. ZAL 是绿色软件，无需安装，默认集成到 PAN1080 Development Kit 中，位于<PAN1080-DK>\05_TOOLS 目录下：



2. 进入 ZAL-win32-x64 子目录，找到 ZephyrAppLauncher.exe：



3. 双击 ZephyrAppLauncher.exe，打开后界面如下：



注:

1. ZAL 目前只提供 Windows x64 版本;
2. 为后续使用方便，可以采用如下方式为 ZAL 创建快捷入口（以 Win10 系统为例）:
   - 固定到任务栏：直接将 ZephyrAppLauncher.exe 文件拖拽到 Window 任务栏;
   - 固定到开始菜单：在 ZephyrAppLauncher.exe 图标上点击右键，选择**固定到“开始”屏幕**;
   - 创建桌面快捷方式：在 ZephyrAppLauncher.exe 图标上点击右键，选择**发送到** -> **桌面快捷方式**;

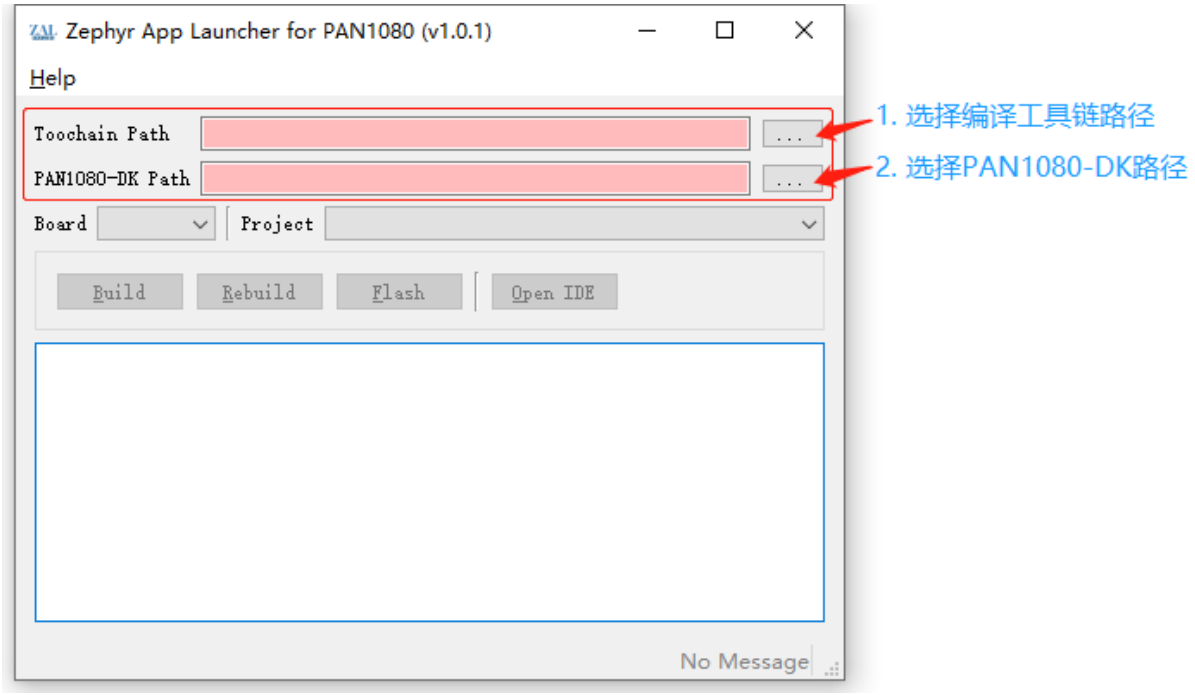
4.3.3 3 功能介绍

下面详细介绍 Zephyr App Launcher for PAN1080（ZAL）的具体功能及使用方法。

3.1 工具配置

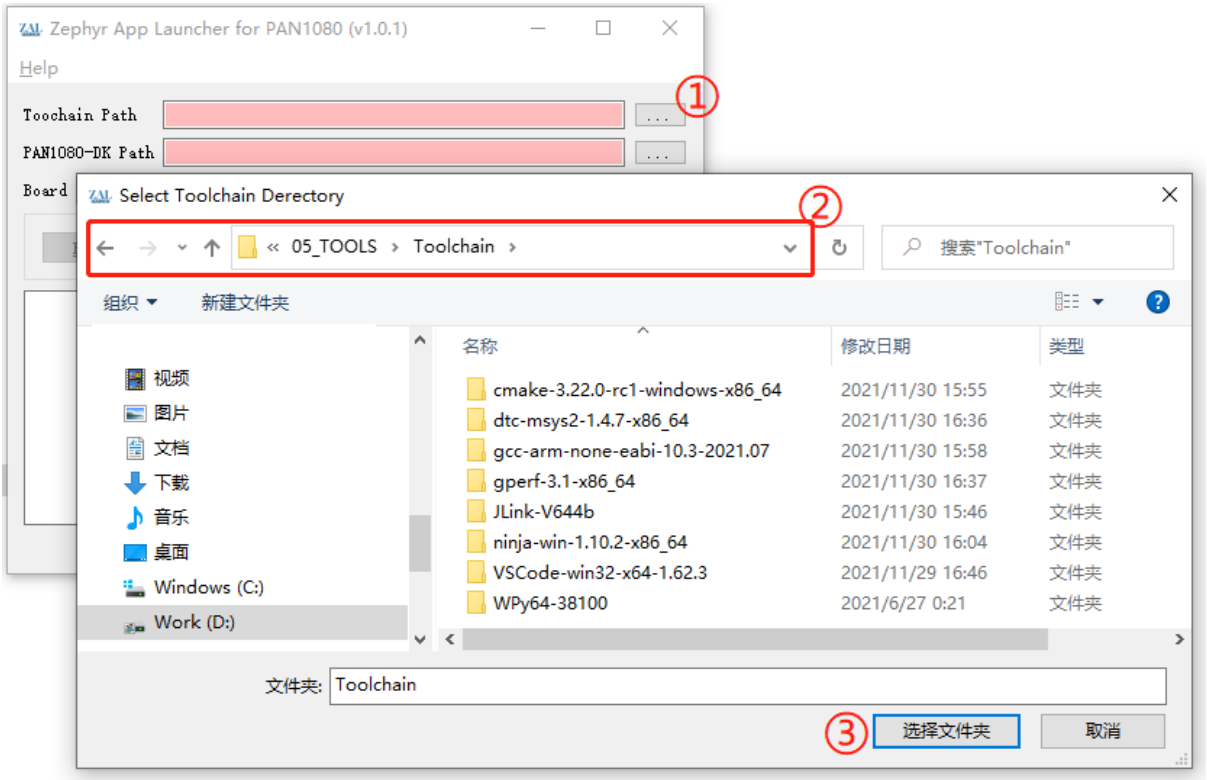
第一次使用，需要先指定如下两个路径信息:

1. 配置 PAN1080 编译工具链 Toolchain 的路径;
2. 配置 PAN1080 DK 开发包的路径;

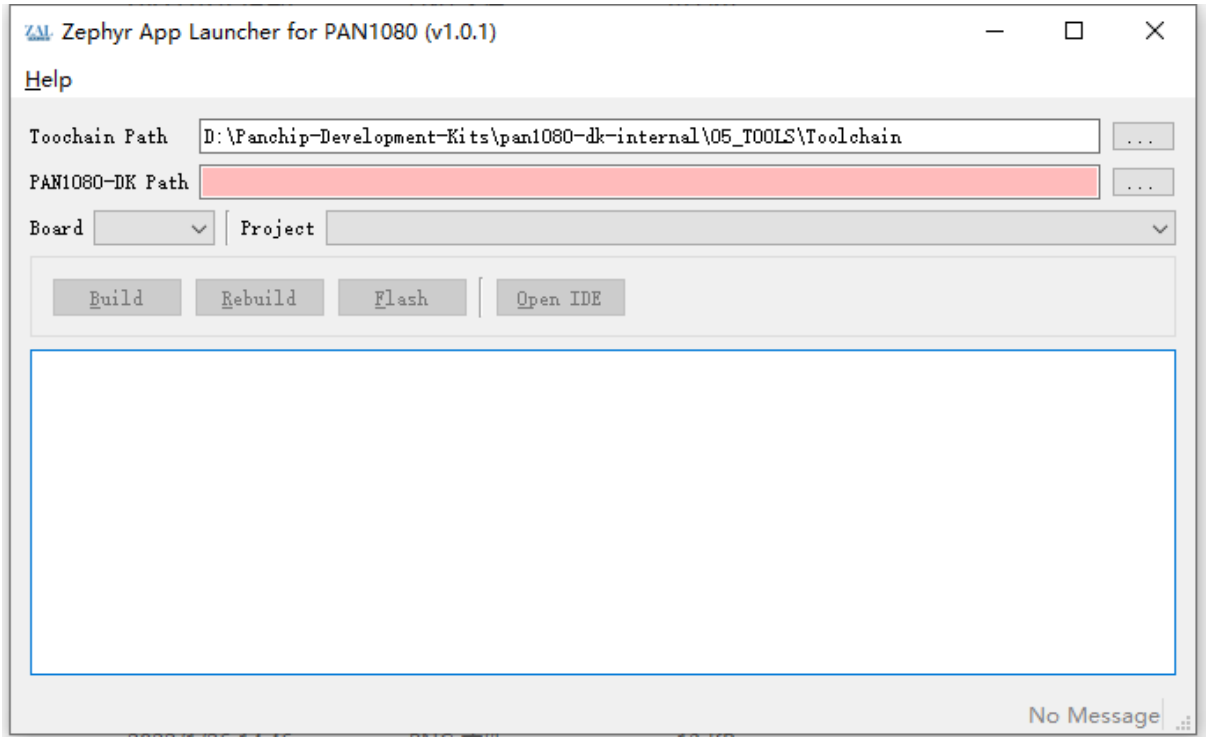


3.1.1 编译工具链选择

1. 点击 Toolchain Path 行末的... 按钮，打开目录选择对话框；
2. 点击路径一栏，进入 05_TOOLS 目录下的 Toolchain 子目录；
3. 点击右下角选择文件夹按钮；

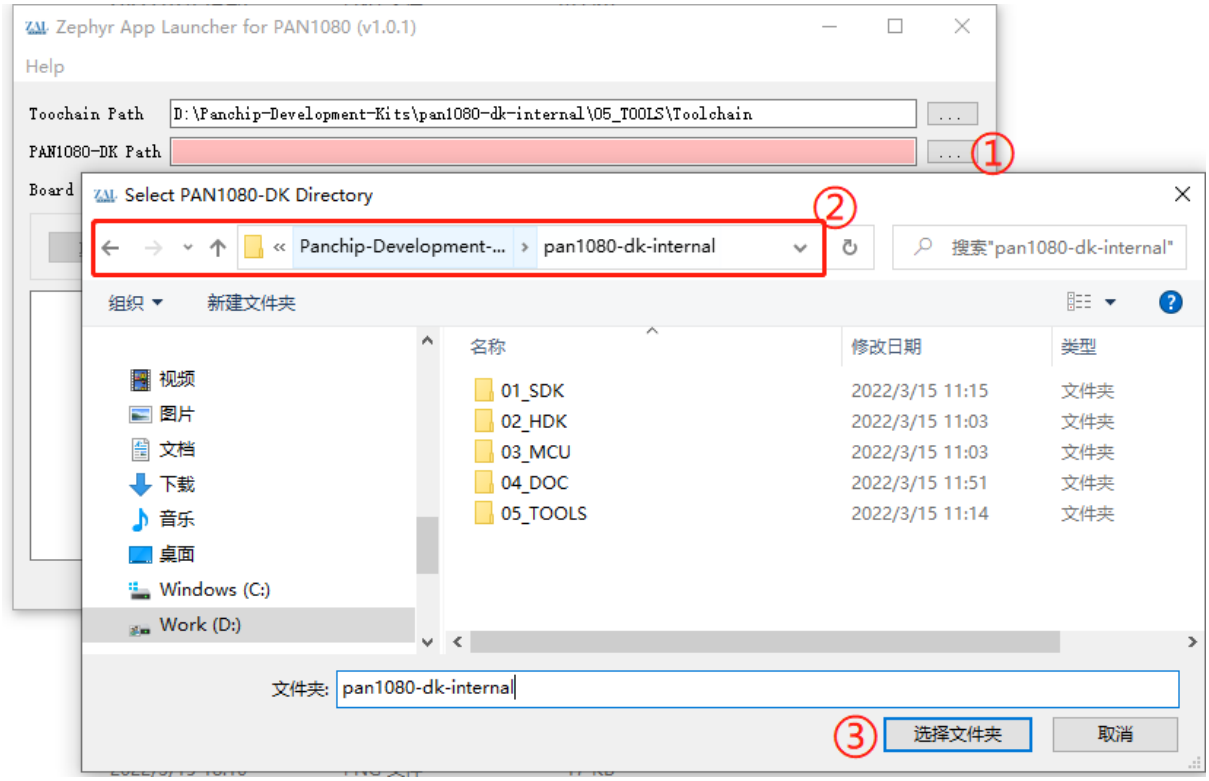


如果配置成功，则可以看到 Toolchain Path 的路径框背景颜色变为白色：

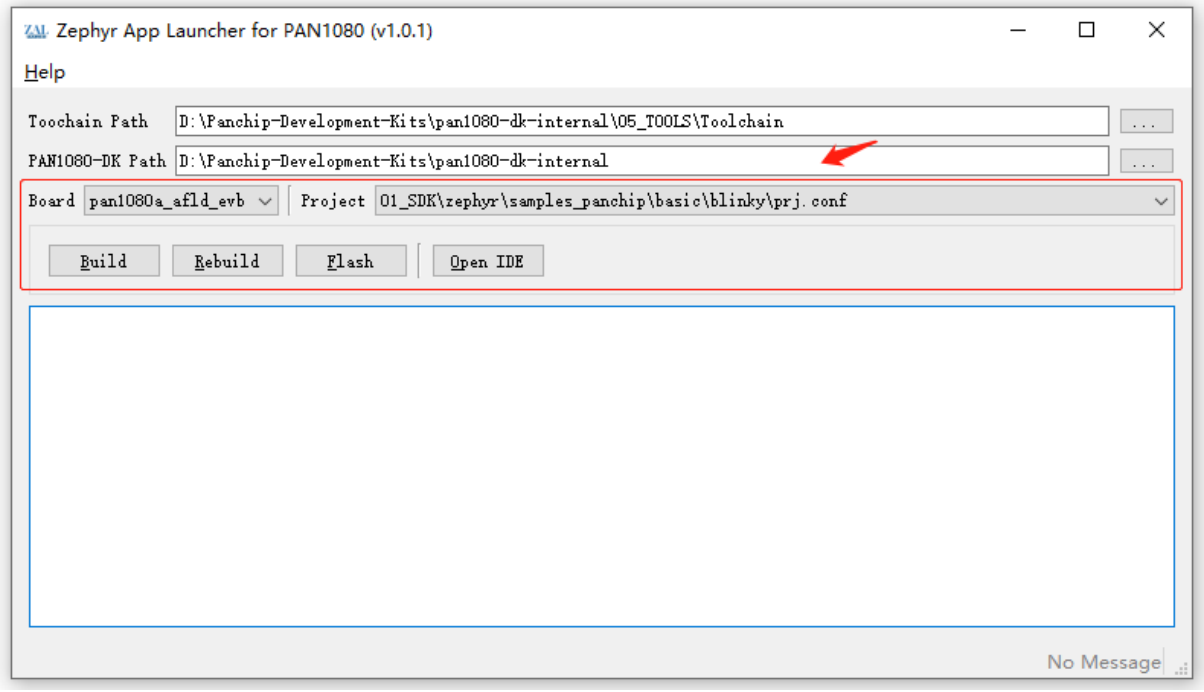


3.1.2 PAN1080 DK 选择

1. 点击 PAN1080-DK Path 行末的... 按钮，打开目录选择对话框；
2. 点击路径一栏，进入 PAN1080-DK 根目录；
3. 点击右下角选择文件夹按钮；

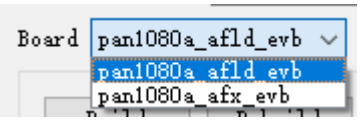


如果配置成功，则可以看到 PAN1080-DK Path 的路径框背景颜色变为白色，同时下方的 Board 选择、Project 选择、Build、Rebuild、Flash、Open IDE 等按钮变为可操作状态：



3.2 Board 选择

Board 下拉列表中显示了<PAN1080-DK>\01_SDK\zephyr\boards\arm 目录下的所有开发板：



以上图为例，可以看到当前选择的 PAN1080-DK 开发包中，默认支持的 2 个 EVB 开发板：

- pan1080a_afld_evb：表示主控是封装为 LQFP64、Flash 大小为 1MB 的 PAN1080A EVB
- pan1080a_afx_evb：表示主控是封装为 QFN32、Flash 大小为 1MB 的 PAN1080A EVB

3.3 Project 选择

Project 下拉列表中显示了<PAN1080-DK>\01_SDK\zephyr\samples_panchip 及<PAN1080-DK>\01_SDK\zephyr\tests_panchip 目录下的所有的 App Project 及对应的 Config 文件：

3.4 输出目录

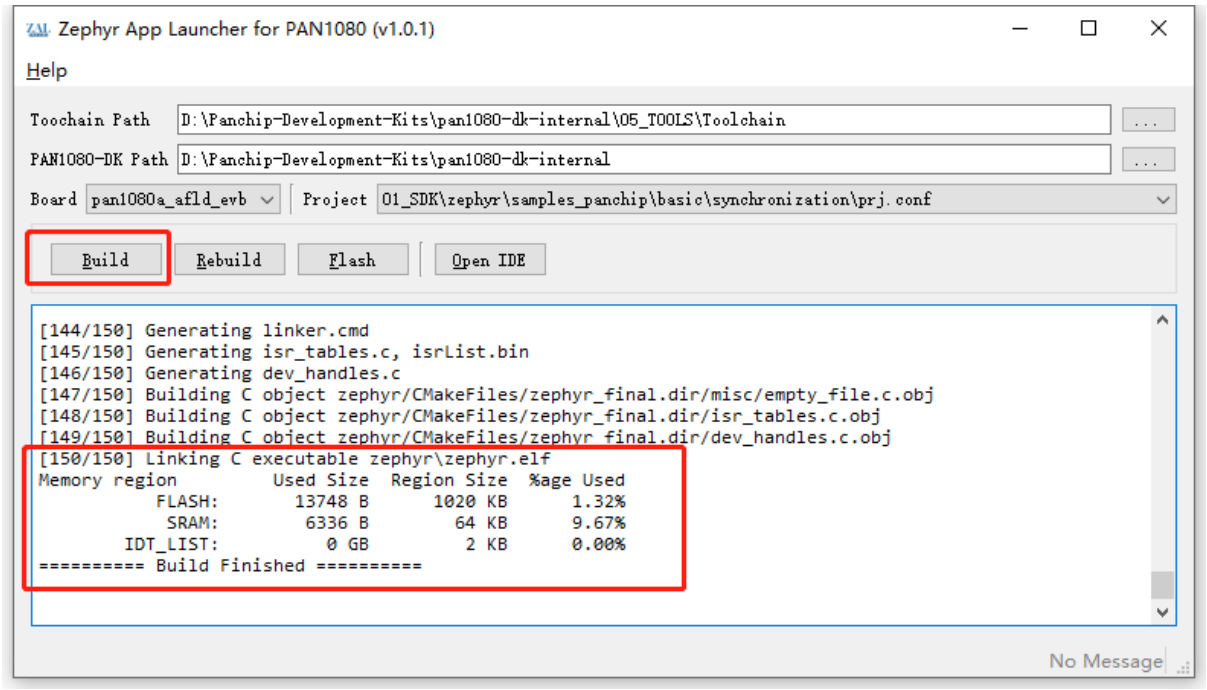
在执行 Build/Rebuild 操作时，ZAL 会**自动创建输出目录**；在执行 Flash 操作时，ZAL 会从自动创建的**输出目录**中查找编译生成的 Image 文件进行烧录。

ZAL 根据不同的 Board 和 Project 选择生成不同名称的**输出目录**，规则为 output_folder = <PAN1080-DK>\01_SDK\build\<app_name>_<board_name>，其中：

1. <PAN1080-DK> 表示前面**工具配置**阶段设置的 PAN1080-DK Path 目录；
2. <app_name> 表示前面 Project **选择**阶段设置的 App Config（*.conf）文件所在目录，例如当我们选择 Project 为<01_SDK\zephyr\samples_panchip\basic\synchronization\prj.conf> 时，<app_name> 表示 synchronization；
3. <board_name> 表示前面 Board **选择**阶段设置的 Board 名称；

3.5 基本功能：Build

Board 和 Project 选择完成后，点击 Build 按钮，即可开始**编译（增量编译）**，软件底部 Log 窗口会显示编译过程，完成后如下图所示：

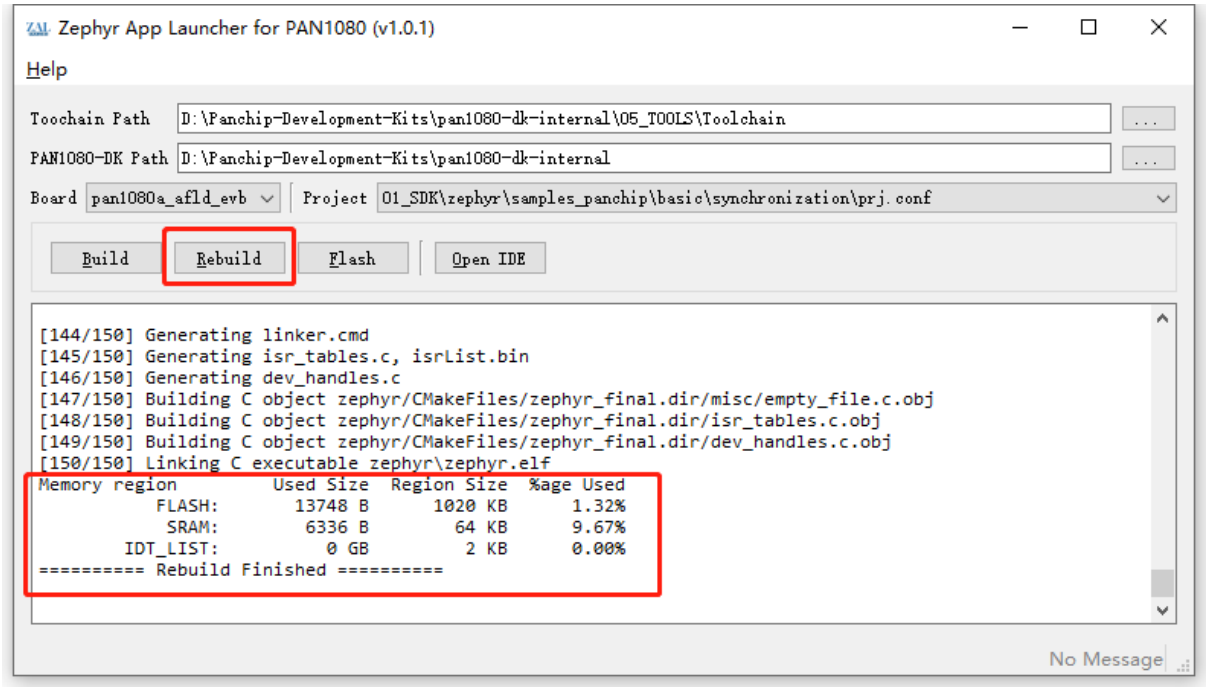


注：

1. Build 按钮含有**增量编译**的功能，即如果当前选择的 Board+Project：
   1. 之前没有编译过，则会执行**重新编译**命令；
   2. 如果之前编译过，则会执行**增量编译**命令；
2. Build 输出目录

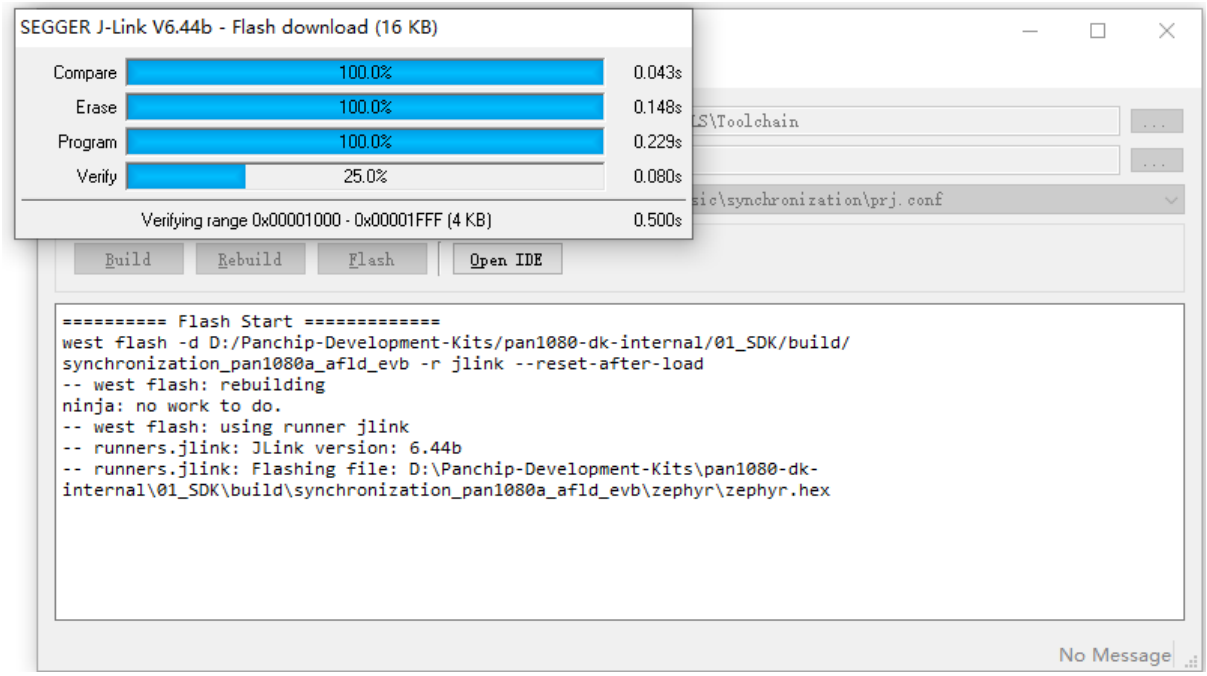
3.6 基本功能：Rebuild

点击 Rebuild 按钮，即可开始**重新编译**，软件底部 Log 窗口会显示编译过程，完成后如下图所示：



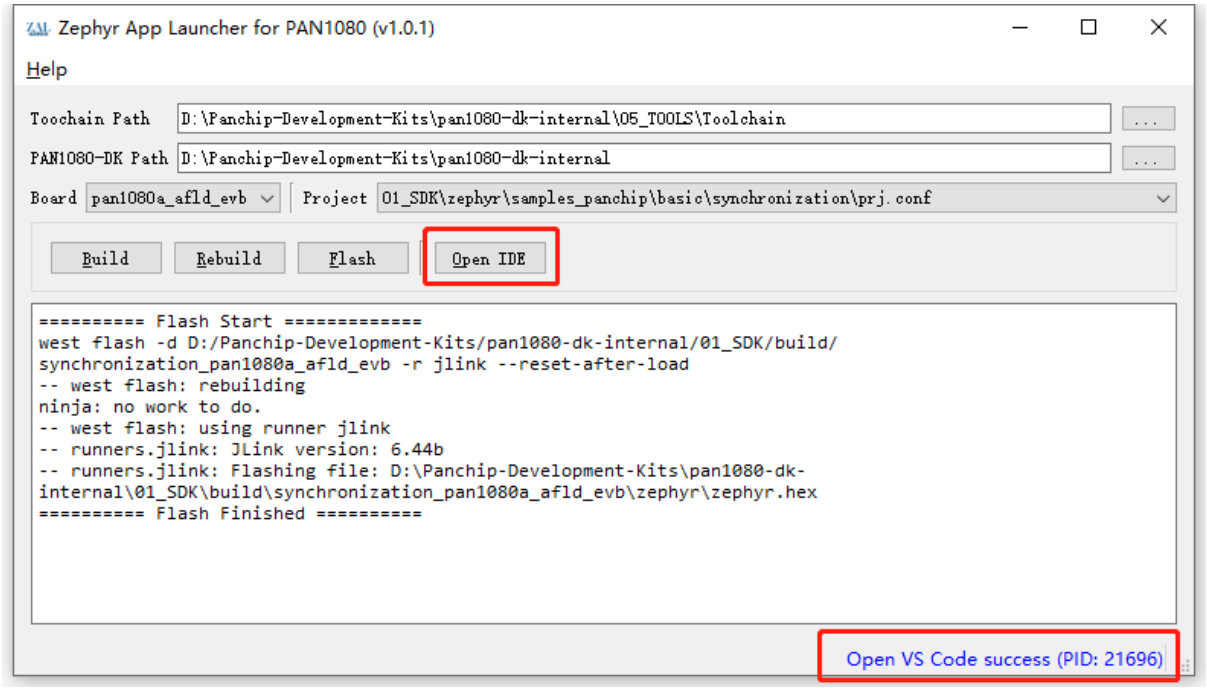
3.7 基本功能：Flash

点击 Flash 按钮，即可开始**烧录**，软件底部 Log 窗口会显示烧录指令，稍后会弹出 JLink 烧录过程对话框，如下图所示：

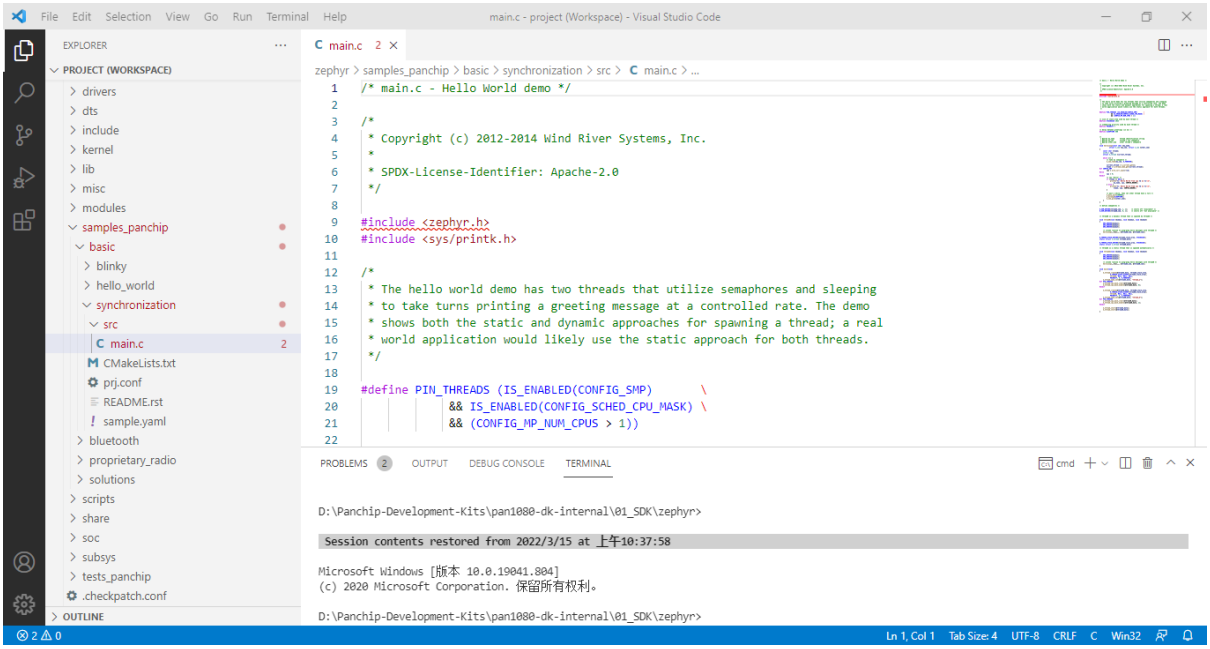


3.8 基本功能：Open IDE

点击 Open IDE 按钮，即可使用 Toolchain 中自带的 VS Code 打开当前选择的 App 工程，如果打开成功，则软件状态栏右下角会提示 “`Open VS Code success`”，如下图所示：



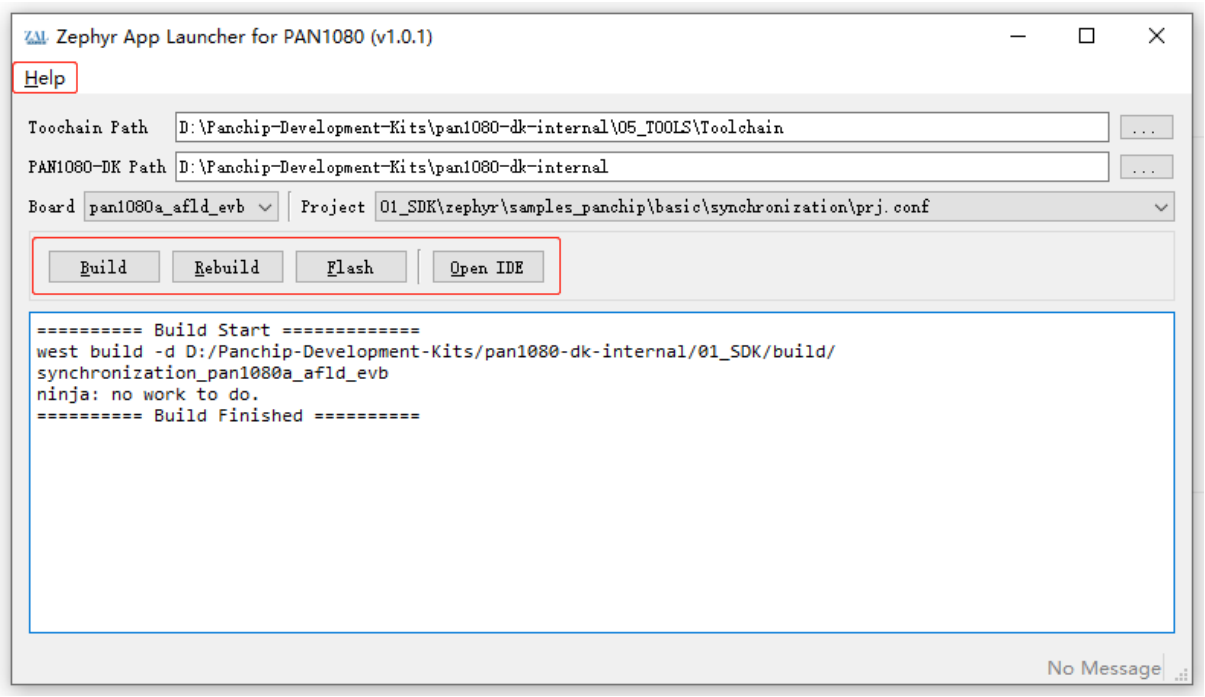
VS Code 打开成功：



注：第一次打开 VS Code 的速度可能会比较慢，只要状态栏提示打开成功，就说明 VS Code 程序已被成功唤起，只需耐心等待片刻即可。

3.9 快捷键

ZAL 支持常用操作的快捷键功能。在软件主界面中，**按住键盘 Alt 键**，可以看到**菜单栏**和**功能按钮**名称中的某个字符下方出现下划线，即表示此**菜单**或**功能按钮**支持快捷键：

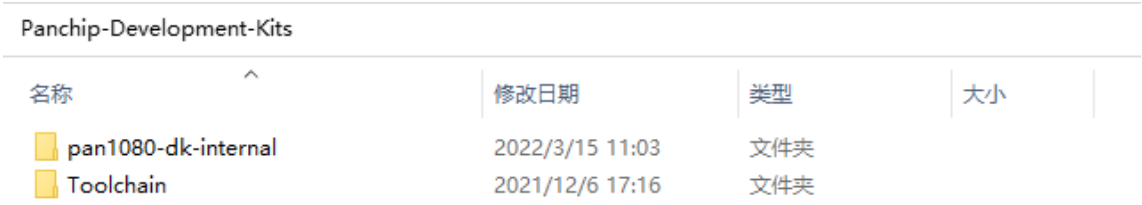


目前软件支持的快捷键功能及其对应键位如下：

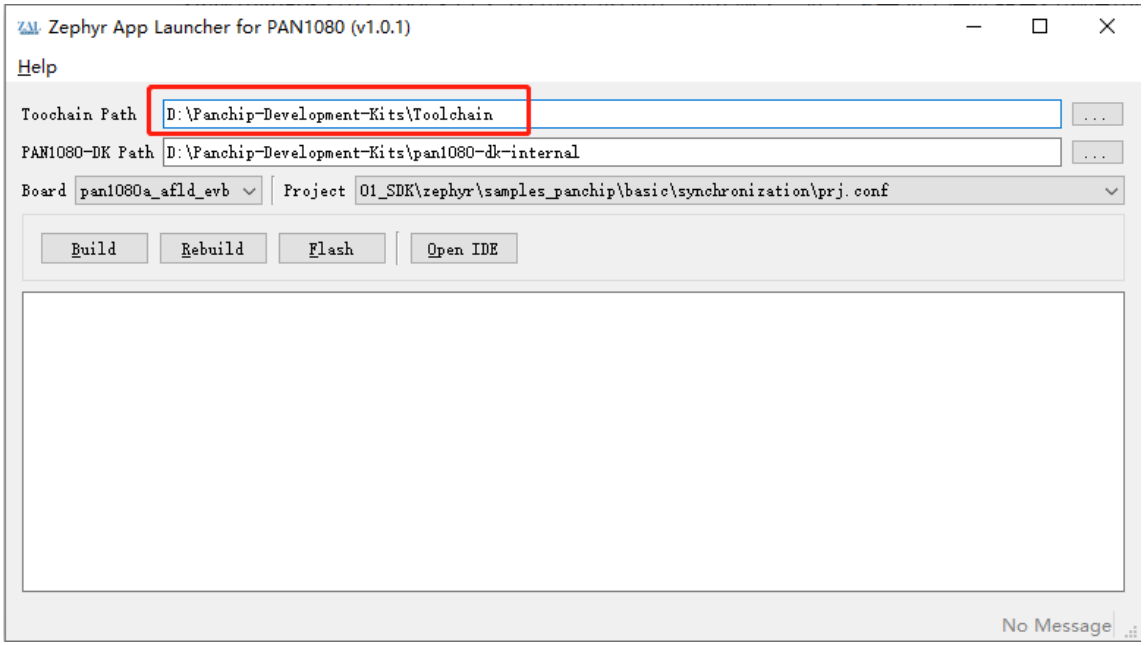
功能	快捷键
Build	Alt + B
Rebuild	Alt + R
Flash	Alt + F
Open IDE	Alt + O
Help	Alt + H

4.3.4 4 Tips

1. ZAL 第一次打开后，会在自身目录中创建一个配置文件，用来记录软件的一些设置，比如配置的路径信息、当前选择的 Board 和 Project 信息、甚至软件的窗口大小和位置等，以方便软件后续打开使用。
2. ZAL 是绿色软件，其本身可以被拷贝到任何目录中，只需注意把 ZAL 所在的目录（名为 ZAL-win32-x64）整体拷贝即可。
3. 在SDK 快速入门文档中**编译工具链配置**部分我们提到，编译工具链 Toolchain 目录，需要解压缩到<PAN1080-DK>\05_TOOLS 目录下方能正常使用；而实际上，此要求是为了保证命令行编译功能（<PAN1080-DK>\01_SDK\PAN1080 SDK CLI）和快速编译脚本（<PAN1080-DK>\01_SDK\quick_build_samples）能够正常使用而提出的。而对于 ZAL 工具来说，Toolchain 目录可以存放在 PC 中的任意目录中（需要注意路径中不能有中文或空格），例如，我们可以把 Toolchain 目录解压到与<PAN1080-DK> 同级的目录中：



然后在 ZAL 中配置修改后的 Toolchain 目录即可：



注：将 `Toolchain` 移到`<PAN1080-DK>` 外面，带来的好处是不需要每次更新 PAN1080-DK 的时候，都重新解压一份 `Toolchain` 放到`<PAN1080-DK>\05_TOOLS` 目录下，而是可以直接使用 `ZAL`，只需配置正确的路径即可。但**需要注意**，这样做将使得我们后面只能使用 `ZAL` 来编译 App 程序，无法再使用命令行编译或快速编译脚本进行编译。

4. 当点击 `Open IDE` 按钮，打开 VS Code 时，若当前选择的 Project 是第一次打开，则会默认打开当前 App 的 main.c 文件；若当前选择的 Project 不是第一次打开，则会恢复上一次 Project 关闭时的文件打开状态。

4.4 Zephyr Devicetree 与 Kconfig 配置指南

4.4.1 1 概述

Zephyr Configuration System 是 PAN1080 SDK 配置各种 Feature 功能的核心框架。

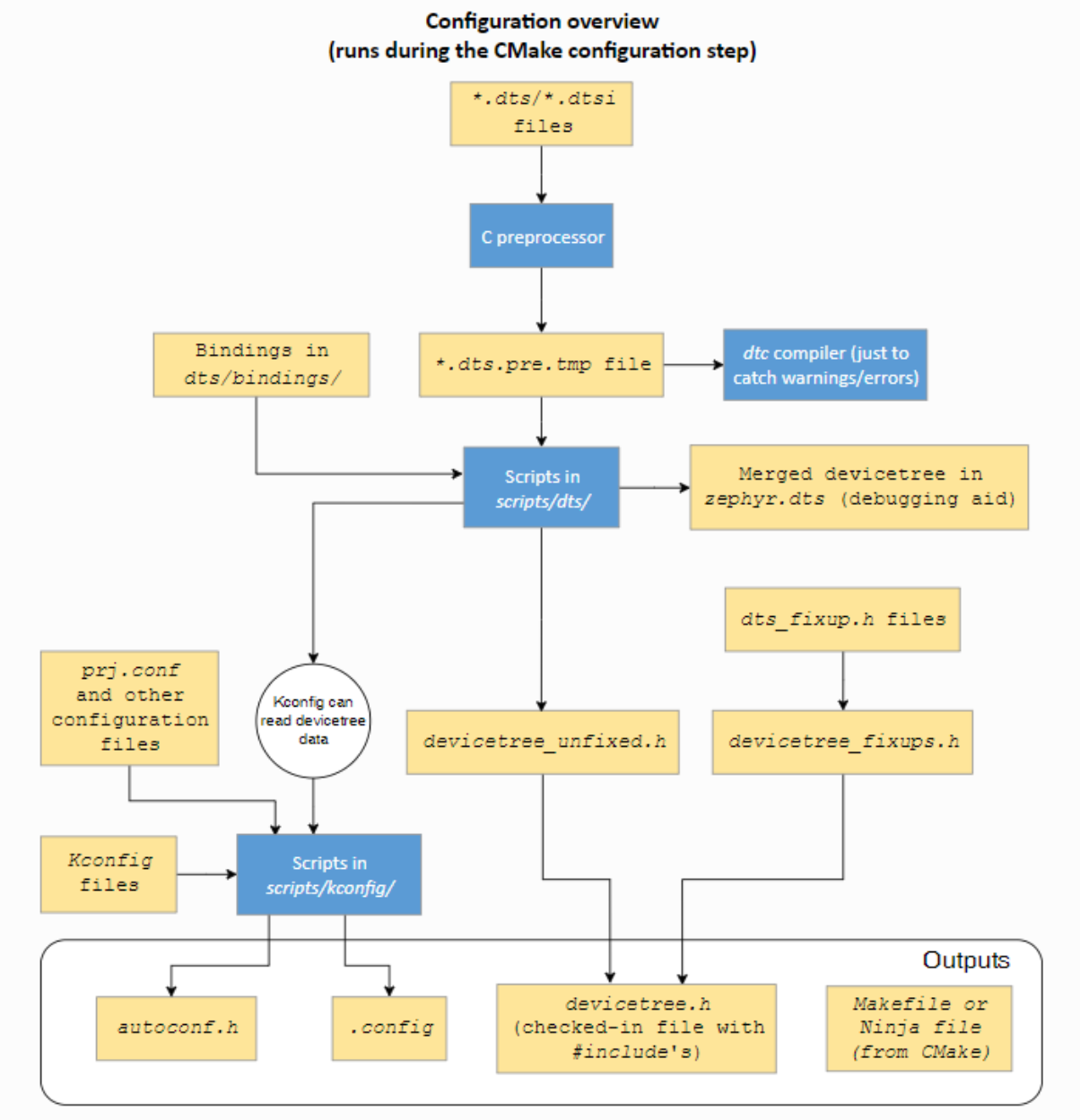
实际上，zephyr 使用 CMake 进行构建，在生成 Application Image 的过程中，又可分为两大阶段：

1. 配置阶段（Configuration Phase）：此阶段下，通过执行相关目录下的 `CMakeLists.txt` 脚本，对当前的 Kconfig 配置、dts 配置等进行解析，最终产生一些配置输出文件（`autoconf.h`、`devicetree.h`、`make/ninja file` 等），供下一阶段使用；
2. 构建阶段（Build Phase）：此阶段下，在前一阶段的输出文件的基础上，调用 `GCC-ARM` 编译工具链进行编译链接，最终生成可执行文件供后续烧录调试使用；

本文将介绍 zephyr 配置阶段（Configuration Phase）的基本概念和相关知识，包括 2 大部分：

- Devicetree（设备树）配置；
- Kconfig 选项配置；

Zephyr Configuration System 整体框架如下图所示：



Zephyr 使用 CMake 实现了以上所有流程，CMake 从 App 工程目录下的 `CMakeLists.txt` 脚本开始执行，在执行过程中，此文件又会引用 zephyr 根目录下的 `CMakeLists.txt` 脚本，在其执行过程中又会转而引用分散在各个待编译目录下的 `CMakeLists.txt` 脚本，最终生成一组 Ninja（类似于 Makefile）文件供后续编译使用。

而在 CMake 执行过程中，除了生成 Ninja 文件外，还会调用相关脚本工具，解析 Devicetree 和 Kconfig 文件，并生成对应的`.h` 文件供后续使用：

- Devicetree：
  - `*.dts` (DeviceTree Source) 和 `*.dtsi` (DeviceTree Source Include) 文件由构建系统从 architecture、soc、board、app 等目录中收集；
  - `*.dtsi` 文件由 `#include` 的方式包含在 `*.dts` 文件中，并由 C 预处理器进行解析；另一方面，C 预处理器同时还会解析 `*.overlay` 文件；在解析的过程中，其中的宏定义也将被展开；
  - 经过预处理后的 Devicetree 文件将会合并保存至一个名为 `*.dts.pre.tmp` 的文件中，此文件一方面会被一个名为 DeviceTree Compiler (dtc) 的工具检查，确保其中没有语法错误；另一方面会作为源文件，输入给 dts **工具**进行后续处理；
  - 上述 dts **工具**在处理 `*.dts.pre.tmp` 文件的同时，还会从 `zephyr/dts/bindings` 目录下拉

取相关的 *.yaml 文件，对 dts 中的各个 property 值进行检查，检查通过后会生成一个名为 zephyr.dts 的文件，供后续参考；

- 除 zephyr.dts 文件以外，dts **工具**还会生成一个名为 devicetree_unfixed.h 的头文件，此文件会与另一个名为 devicetree_fixups.h 的头文件一起，被包含在 devicetree.h 中，作为后续构建阶段的输入；

- Devicetree fixups：
  - 在 architecture、soc、board、app 等目录中可能会存在一些名为 dts_fixup.h 的头文件，这些文件用于重命名 devicetree_unfixed.h 中的一些宏，以满足的某些 app 的需求；

    注：PAN1080 SDK 中所有的 app 代码均直接使用 zephyr.dts 对应的 devicetree_unfixed.h 文件，因此无需关注 dts_fixup.h。

- Kconfig：
  - Kconfig 文件分布在 architecture、soc、board、app 等目录中，其定义了各个目录模块中当前的配置选项以及各选项之间的依赖关系；
  - 构建系统调用 Kconfig **工具**，将 app 级的配置文件（*.conf）、board 级的配置文件（*Kconfig.defconfig/Kconfig.board/*_defconfig）、soc 级的配置文件（Kconifg、Kconfig.defconfig、Kconfig.soc）、以及其他分散在各个目录中的相关 Kconfig 文件进行解析和合并处理，最终生成 2 个文件：autoconf.h 和.config；其中：autoconf.h 文件用于后续构建阶段的输入，而.config 文件则用于后续通过 menuconfig/guiconfig 的方式修改配置；

4.4.2 2 Devicetree 配置

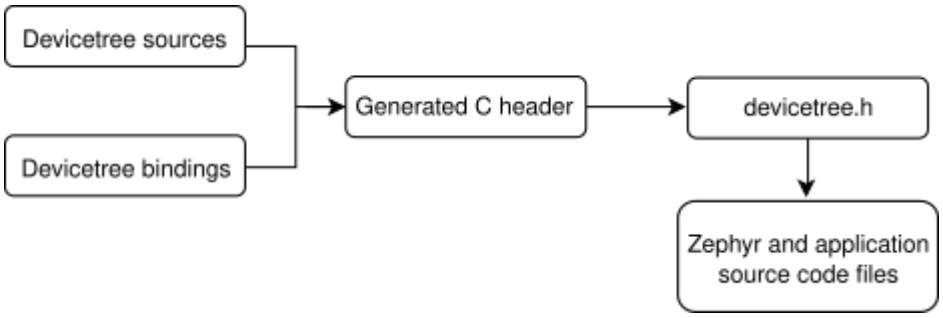
2.1 基本概念

Devicetree 是从 Linux Kernel 借取的概念，其本质是一系列用于描述硬件的分层数据结构。Zephyr 使用 Devicetree 来描述其支持的 soc 和 board 级硬件，以及各个硬件模块的初始配置。

前面的框图中也提到，Devicetree 有两种输入源文件：devicetree sources 与 devicetree bindings，其中 devicetree sources 是后缀为.dts 和.dtsi 的文件，用于描述 Devicetree 自身，而 devicetree bindings 则是后缀为.yaml 的文件，其用于描述 devicetree sources 文件中各个属性的定义和数据类型。

Devicetree 的最终输出为一个名为 devicetree_unfixed.h 的头文件，而 zephyr 提供了一个名为 devicetree.h 的文件，其在包含 devicetree_unfixed.h 的同时，还提供了可以访问 devicetree 的 API 接口（函数式宏），它们都以 DT_ 开头，zephyr 在编译时会默认包含此文件，保证 devicetree 中的内容可以在各个 C 文件中访问到。

一个简化的 devicetree 相关文件处理流程如下：



下面举一个简单的例子说明 devicetree 的基本语法。

正如其名，devicetree 本质上是一个树（tree）状的数据结构，其对应一种名为 DTS (DeviceTree Source) 的易读文本格式（关于此格式的 SPEC 细节可参考 Devicetree specification）。

一个简单而完整的 DTS 文件内容如下：

```
/dts-v1/;

/ {
        a-node {
                subnode_label: a-sub-node {
                        foo = <3>;
                };
        };
};
```

其中：

- 第 1 行 `/dts-v1/;` 表示当前 DTS 格式的版本号为 `version 1`（用于区分已经废弃的 `version 0` 版本）；
- 其后的内容表示此树（tree）包含 3 个节点（nodes）：
  1. 根节点: `/`
  2. 根节点的直接子节点 `a-node`
  3. 节点 `a-node` 的直接子节点 `a-sub-node`,

节点可以有一个 ** 唯一（不重名）** 的标签（labels），在 dts 文件的其他位置中可以通过标签名的方式快速引用其他节点。例如，上述 dts 文件中，`a-sub-node` 的即有一个名为 `subnode_label` 的标签。一个节点可以有 0 个、1 个或多个标签。

除标签以外，还可以使用类似 Unix 文件系统中的完整路径来描述节点，上述 dts 文件中，`a-sub-node` 的完整路径为`/a-node/a-sub-node`。

Devicetree 节点中可以定义属性（`properties`），属性实际上是键值对（name/value pairs）。属性的值可以是任意字节串（属性值的具体含义需要在当前节点对应的 dts bindings 文件中定义）。

上述 dts 文件中 `a-sub-node` 节点中包含了一个名为 `foo` 的属性，其数据是一个 int 类型，值为 `3`。

> **注**：更多 Devicetree 基本概念介绍请参考 Zephyr 官方文档：Introduction to devicetree。

2.2 Devicetree Bindings

Devicetree 设备树本身只是 Zephyr 硬件描述机制的半壁江山，因为其是一种非结构化（unstructured）的格式；换句话说，仅用`.dts` 文件是无法完整准确描述硬件的，原因是其虽然包含了完整的节点（Node）和属性（Property）描述，但其自身是无法确保这些描述准确无误的。于是 Zephyr 硬件描述机制的另外半壁江山：Devicetree Bindings 就应运而生了。

Devicetree Bindings 文件（`*.yaml`）用于描述 Devicetree 文件（`*.dts`）中各个节点（Node）及其中各个属性（Property）的准确含义，为它们提供必要的语义信息（semantic information）。

需要注意的是，Zephyr devicetree bindings 文件是一组自定义内容的 YAML 格式文件，并不与 Linux kernel 共用的 `dt-schema` 工具。

下面举一个简单的例子阐述构建系统是如何将 Devicetree 与 Devicetree Bingdings 关联起来的。

一个简单的 Devicetree 文件中（`example.dts`）含有如下所示的节点：

```
/* Node in a DTS file */
bar-device {
    compatible = "foo-company,bar-device";
    num-foos = <3>;
};
```

而其对应的 bindings 文件（`foo-company,bar-device.yaml`）内容如下：

```
# A YAML binding matching the node

compatible: "foo-company,bar-device"
```

（下页继续）

(续上页)

```
properties:
  num-foos:
    type: int
    required: true
```

1. Devicetree 文件（example.dts）中，名为 bar-device 的节点内，有一个 compatible 属性，值为"foo-company,bar-device"；而上述 Binding 文件，名称为 foo-company,bar-device.yaml，其中包含一个 compatible: "foo-company,bar-device" 的键值对；这样就成功将 Devicetree 某个节点与对应的 Binding 文件关联了起来；
2. Devicetree 文件中的 bar-device 的节点还描述了一个名为 num-foos 的节点，其值为 3；同时我们看到对应的 Binding 文件中，描述了一个名为 num-foos 的属性，其类型（type）为 int，表示整型数据，而需求（required）为 true，表示此属性为必需：当构建系统在当前 dts 节点中发现此属性未赋值时会直接报错；

   注：更详细的 Devicetree Bindings 介绍请参考官方文档：Devicetree bindings。

2.3 Devicetree Overlays

普通的 Devicetree 文件（.dts/.dtsi），一般用于描述 SoC **级**和**板级**硬件参数，但有时候我们会遇到这种场景：

板级 Devicetree 文件中配置的硬件参数，可适用于大多数 App 工程；但某个特殊的 App 工程，需要做额外的修改才能正常工作。

我们不希望专门为某个特殊的 App 需求直接修改板级 Devicetree 文件，因为这可能会影响其他 App 的工作，这时候，我们可以在此特殊的 App 工程下，加入一个 Devicetree Overlay 文件，用来覆盖板级 Devicetree 的配置。

达成以上目的的方法是（这里以 pan1080a_afld_evb board 为例）：

1. 在当前 App 工程的目录下，新建一个 boards 目录；
2. 在此 boards 目录下，新建一个名为 pan1080a_afld_evb.overlay 的文件；
3. 假设当前 App 需要修改串口 Log 打印的波特率：
   - 在板级 Devicetree 配置文件\zephyr\boards\arm\pan1080a_afld_evb\pan1080a_afld_evb.dts 中可以看到，默认的串口 Log 打印波特率为 921600：

     ```
     &uart1 {
             current-speed = <921600>;
             pinctrl-0 = <&p0_6_uart1_tx &p0_7_uart1_rx>;
             status = "okay";
     };
     ```

   - 我们在 App 工程新建的 pan1080a_afld_evb.overlay 文件中，直接加入如下的语句，即可将波特率改为 115200：

     ```
     &uart1 {
             current-speed = <115200>;
     };
     ```

   更多 Devicetree Overlay 细节请参考 Zephyr 官方文档：

   1. Set Devicetree Overlays；
   2. Use Devicetree Overlays；

2.4 如何从 C 代码中获取 Driver 的 device 结构体指针

在开发 App 的过程中，经常会需要调用 Zephyr Driver API 操作 SoC 的外设，而所有的 Zephyr Driver API 接口均需要传入一个类型为 `device` 的结构体指针，而此指针只能通过调用 Devicetree 相关 API 进行间接获取。

这里我们介绍获取 `device` 的结构体指针的方法。

假设我们有如下的 Devicetree 文件片段，我们需要获取 UART 节点 `serial@40002000` 对应的 device 结构体指针：

```
 {
        soc {
                serial0: serial@40002000 {
                        status = "okay";
                        current-speed = <115200>;
                        /* ... */
                };
        };

        aliases {
                my-serial = &serial0;
        };

        chosen {
                zephyr,console = &serial0;
        };
};
```

我们可以先在 C 源文件中使用如下方式之一，定义一个名为 `MY_SERIAL` 的宏，获取 UART0 节点的标识符（Identifier）：

```
/* Option 1: by node label */
#define MY_SERIAL DT_NODELABEL(serial0)

/* Option 2: by alias */
#define MY_SERIAL DT_ALIAS(my_serial)

/* Option 3: by chosen node */
#define MY_SERIAL DT_CHOSEN(zephyr_console)

/* Option 4: by path */
#define MY_SERIAL DT_PATH(soc, serial_40002000)
```

接着，我们可是有如下 2 种方式之一获取 UART0 对应的 device 结构体指针：

```
/* Option 1: use device_get_binding() */
#if DT_NODE_HAS_STATUS(MY_SERIAL, okay)
const struct device *uart_dev = device_get_binding(DT_LABEL(MY_SERIAL));
#else
#error "Node is disabled"
#endif
```

或

```
/* Option 2: use DEVICE_DT_GET() */
const struct device *uart_dev = DEVICE_DT_GET(MY_SERIAL);

if (!device_is_ready(uart_dev)) {
        /* Not ready, do not use */
        return -ENODEV;
}
```

UART0 对应的 device 结构体指针获取成功后，即可调用 UART Driver API 接口进行后续操作。

注：更多 C 代码中访问 Devicetree 的介绍请参考 Zephyr 官方文档：

1. Get a struct device from a devicetree node
2. Devicetree access from C/C++
3. Devicetree API

2.5 PAN1080 SDK 相关 Devicetree 配置文件

1. soc 级 dts 配置：
   - `zephyr\dts\arm\panchip\pan1080\pan1080.dtsi`：PAN1080 硬件描述；
   - `zephyr\dts\arm\panchip\pan1080\pan1080xb1.dtsi`：PAN1080XB1 SoC 硬件描述；
   - `zephyr\dts\arm\panchip\pan1080\pan1080xb5.dtsi`：PAN1080XB5 SoC 硬件描述；
   - `zephyr\dts\arm\panchip\pan1080\pan1080xx1_pinctrl.dtsi` ：PAN1080XX1 SoC 对应的 PINMUX 硬件描述（32 Pin）；
   - `zephyr\dts\arm\panchip\pan1080\pan1080xx5_pinctrl.dtsi`：PAN1080XX5 SoC 对应的 PINMUX 硬件描述（64 Pin）；
2. board 级 dts 配置：
   - `boards\arm\pan1080a_afld_evb\pan1080a_afld_evb.dts`：PAN1080A_AFLD_EVB 板级硬件描述；
   - `boards\arm\pan1080a_afx_evb\pan1080a_afx_evb.dts`：PAN1080A_AFX_EVB 板级硬件描述；
3. dts-bindings：
   - `dts\bindings\<driver>\*.yaml`：各个硬件模块的 dts 属性对应数据定义；

2.6 如何调试 Devicetree 编译错误

有时候我们在修改 Devicetree（`*.dts`）文件的过程中会出现错误，zephyr 提供了一些方法，用于帮助我们找到错误的原因。

前面介绍 Zephyr Configuration System 整体框架的时候提到，在生成 `zephyr.dts` 的过程中，会先产生一个临时的用于调试的文件 `*.dts.pre.tmp`，如果出现 dts 编译错误，可能无法成功生成 `zephyr.dts`，但 `*.dts.pre.tmp` 文件仍然会生成，这时候只需阅读编译错误的 Log，即可在 `*.dts.pre.tmp` 文件中找到错误位置。

例如，我们尝试修改板级 dts 文件 `pan1080a_afld_evb.dts`，但是在随后的编译过程中发现如下错误：

```
Error: pan1080a_afld_evb.dts.pre.tmp:864.1-8 syntax error
FATAL ERROR: Unable to parse input tree
CMake Error at D:/PDK/pan1080-dk-internal/01_SDK/zephyr/cmake/dts.cmake:225 (message):
  command failed with return code: 1
Call Stack (most recent call first):
  D:/PDK/pan1080-dk-internal/01_SDK/zephyr/cmake/app/boilerplate.cmake:541 (include)
  D:/PDK/pan1080-dk-internal/01_SDK/zephyr/share/zephyr-package/cmake/ZephyrConfig.cmake:24␣
↪(include)
  D:/PDK/pan1080-dk-internal/01_SDK/zephyr/share/zephyr-package/cmake/ZephyrConfig.cmake:40␣
↪(include_boilerplate)
  CMakeLists.txt:4 (find_package)
```

从第一行提示可知，dts 中出现了句法错误；我们找到输出目录中名为 `pan1080a_afld_evb.dts.pre.tmp` 的文件，定位到 864 行：

```
859  &uart1 {
860   current-speed = <921600>;
861   pinctrl-0 = <&p0_6_uart1_tx &p0_7_uart1_rx>;
862   status = "okay";
863  }
864  &timer0 {
865   freq = <16000000>;
866   status = "okay";
867  };
868  &timer1 {
869   freq = <8000000>;
870   status = "okay";
871  };
```

观察上下文，可以发现 863 行少了一个分号;，在 `pan1080a_afld_evb.dts` 中修复后，重新编译 app 工程，即可发现问题解决。

注：更多 Devicetree 调试技巧请参考 Zephyr 官方文档：Troubleshooting devicetree。

4.4.3 3 Kconfig 配置

3.1 基本概念

Zephyr 的内核（Kernel）和子系统（Subsystem）支持的 Features 均可以根据实际应用需求，在构建阶段进行配置，而配置的方法是通过一个名为 Kconfig 的机制进行的。正如其名，Zephyr Kconfig 机制在概念上与 Linux Kernel Kconfig 是基本相同的。

配置选项（通常也称为 symbols）在一系列 `Kconfig` 文件中定义，其中还描述了各个配置选项之间的依赖关系和层次关系等信息。

Kconfig 机制的最终有效输出是一个名为 `autoconf.h` 的头文件，此文件会在编译阶段作为最优先的头文件包含在各个 C 源文件中，供各个文件编译使用。`autoconf.h` 中名为 `CONFIG_XXX` 的宏定义，其在 Kconfig 文件中的对应定义是 `config XXX`。

Kconfig Symbol 名称与 `autoconf.h` 中宏定义的关系为：

一个典型的 Kconfig 文件具有如下的形式 (`zephyr/Kconfig.zephyr` 文件片段)：

```
...
source "boards/Kconfig"
source "soc/Kconfig"
source "arch/Kconfig"
source "kernel/Kconfig"
source "dts/Kconfig"
source "drivers/Kconfig"
source "lib/Kconfig"
source "subsys/Kconfig"

osource "$(TOOLCHAIN_KCONFIG_DIR)/Kconfig"

...

menu "Linker Options"

choice
        prompt "Optimization level"
        default NO_OPTIMIZATIONS    if COVERAGE
        default DEBUG_OPTIMIZATIONS if DEBUG
        default SPEED_OPTIMIZATIONS
```

(下页继续)

(续上页)

```
        help
          Note that these flags shall only control the compiler
          optimization level, and that no extra debug code shall be
          conditionally compiled based on them.

config SIZE_OPTIMIZATIONS
        bool "Optimize for size"
        help
          Compiler optimizations will be set to -Os independently of other
          options.

config SPEED_OPTIMIZATIONS
        bool "Optimize for speed"
        help
          Compiler optimizations will be set to -O2 independently of other
          options.

config DEBUG_OPTIMIZATIONS
        bool "Optimize debugging experience"
        help
          Compiler optimizations will be set to -Og independently of other
          options.

config NO_OPTIMIZATIONS
        bool "Optimize nothing"
        help
          Compiler optimizations will be set to -O0 independently of other
          options.

endchoice
...
```

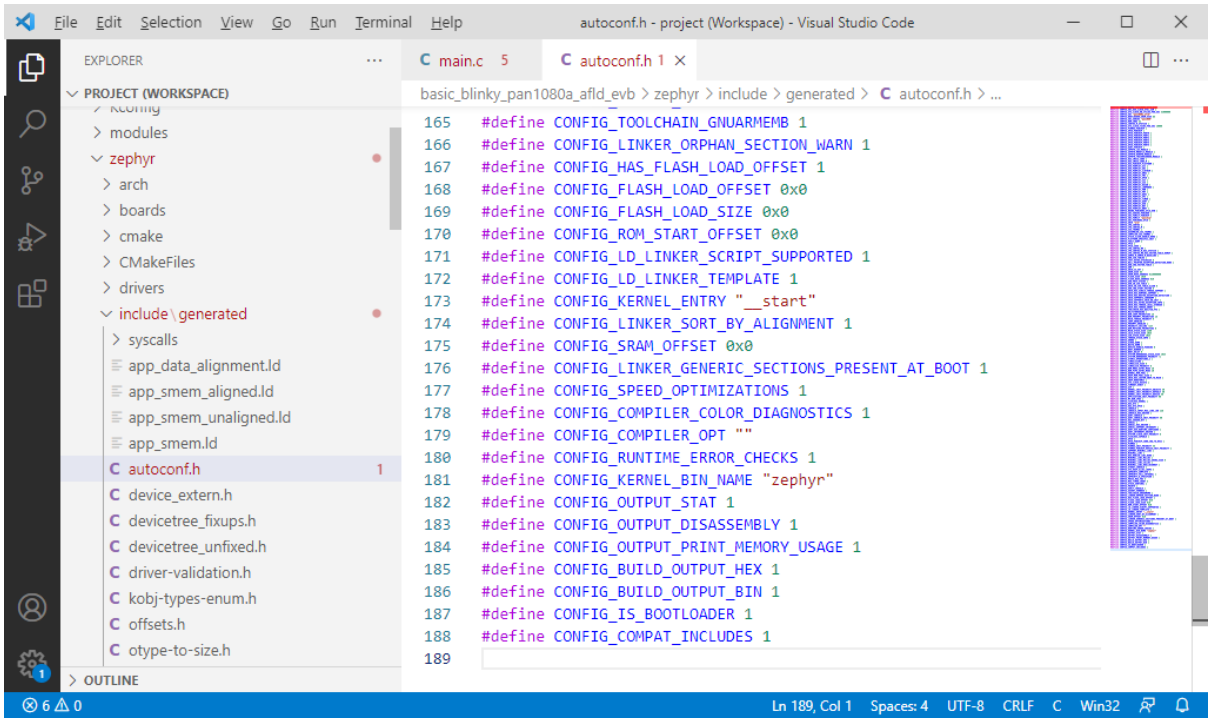
注：更详细的 Kconfig 基本概念请参考 Zephyr 官方文档：Zephyr Kconfig System。

3.2 如何确认当前 App 开启了哪些 Kconfig

根据前面介绍，所有使能的 Kconfig 选项，最终会合并输出到一个名为 `autoconf.h` 的头文件中。因此在成功编译 App 后，可以从生成目录中找到 autoconf.h 文件，即可了解当前哪些 Kconfig 选项被使能了。

这里以编译 `blinky` 例程为例，假设输出目录为 `01_SDK\build\basic_blinky_pan1080a_afld_evb`，则编译成功后，可以在以下目录中找到 `autoconf.h` 文件：

- `01_SDK\build\basic_blinky_pan1080a_afld_evb`\zephyr\include\generated‘



3.3 如何查找 Kconfig 定义

在 Zephyr App 开发过程中，我们经常会遇到以下问题：

1. 我们在 `autoconf.h` 中发现有些可疑的 Kconfig 被使能，需要追踪，但却不知道其在哪里；
2. 我们预期的某个 Kconfig 宏应该打开，但在 `autoconf.h` 中却找不到；

这时候，我们可以利用 Menuconfig（或 Gui Config）工具，追踪特定 Kconfig 选项的定义即调用关系。

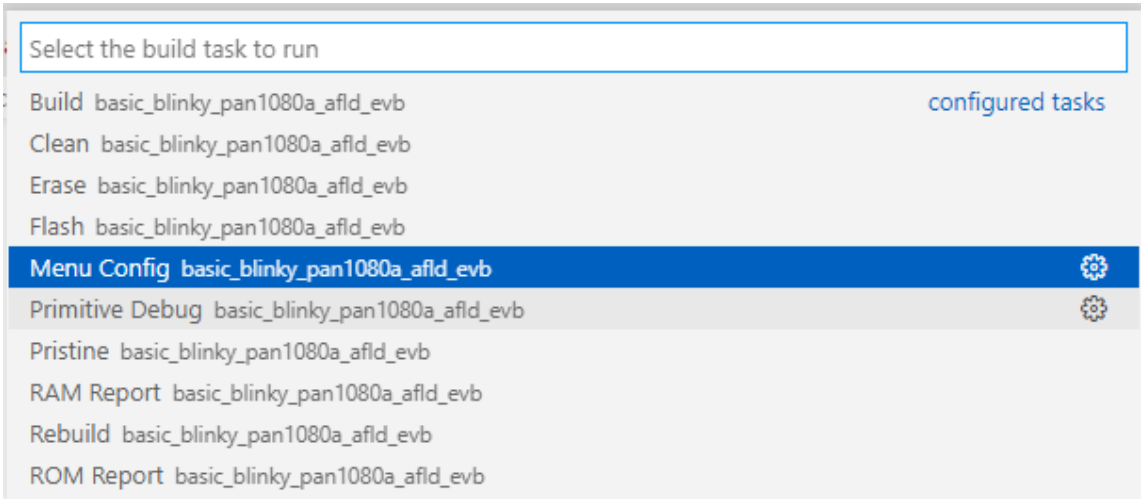
这里以 Menuconfig 为例演示一种场景：

1. 我们在 autoconf.h 中发现一处感兴趣的配置：

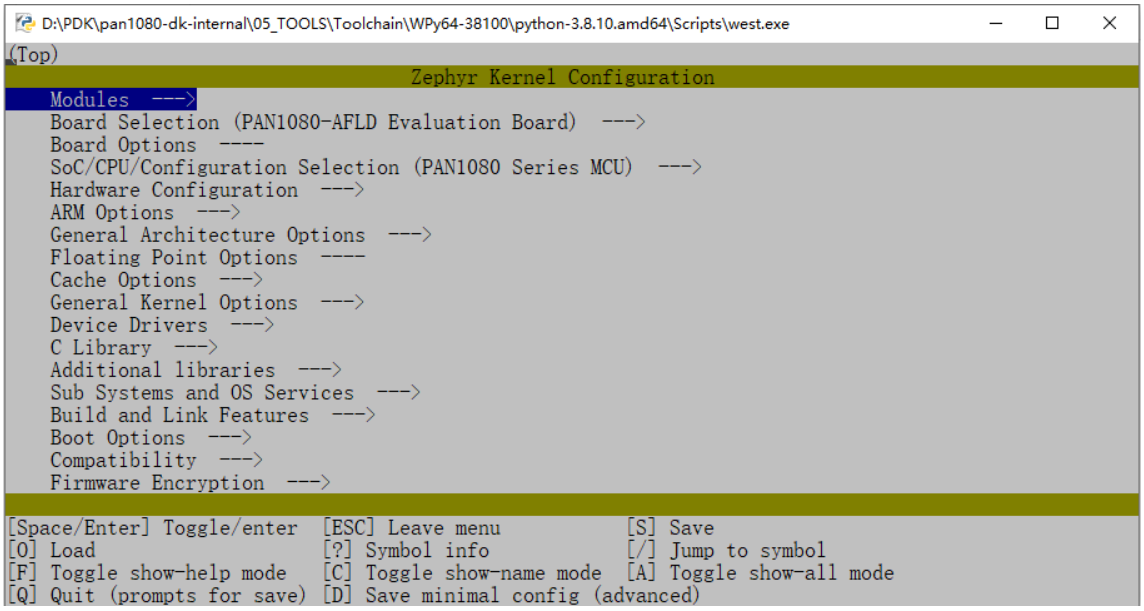
   ```
   #define CONFIG_SPEED_OPTIMIZATIONS 1
   ```

   看名字，似乎是将编译优化选项配置为速度优先的方式。

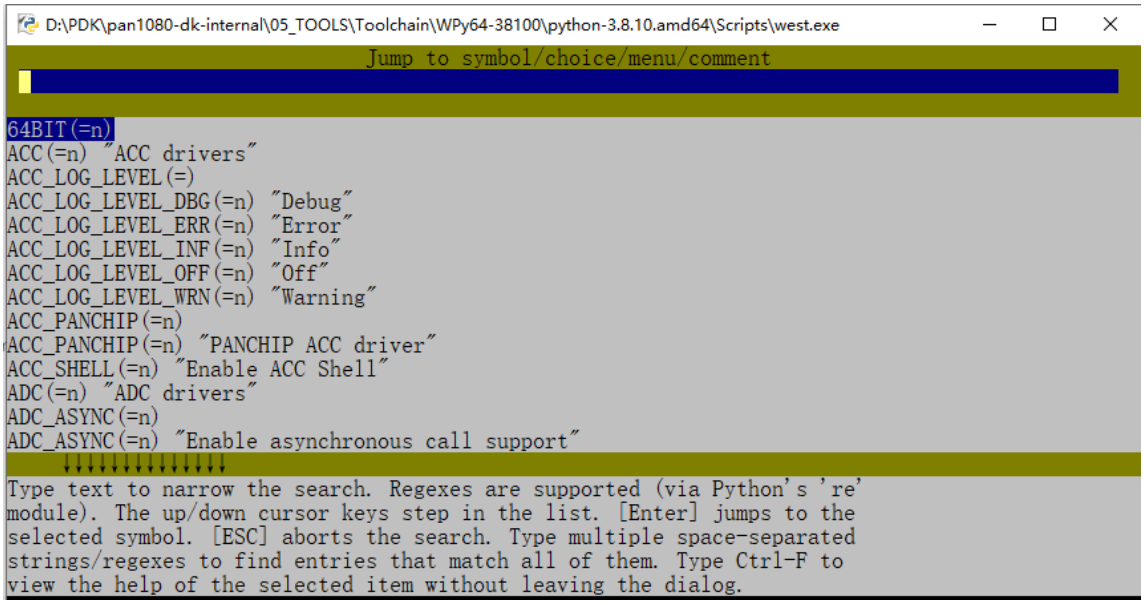
2. 为确认此配置的真实含义，在当前工程的 VS Code 界面中按快捷键 `Ctrl + Shift + B`，打开 Build 菜单，然后执行 `Menu Config` 命令：



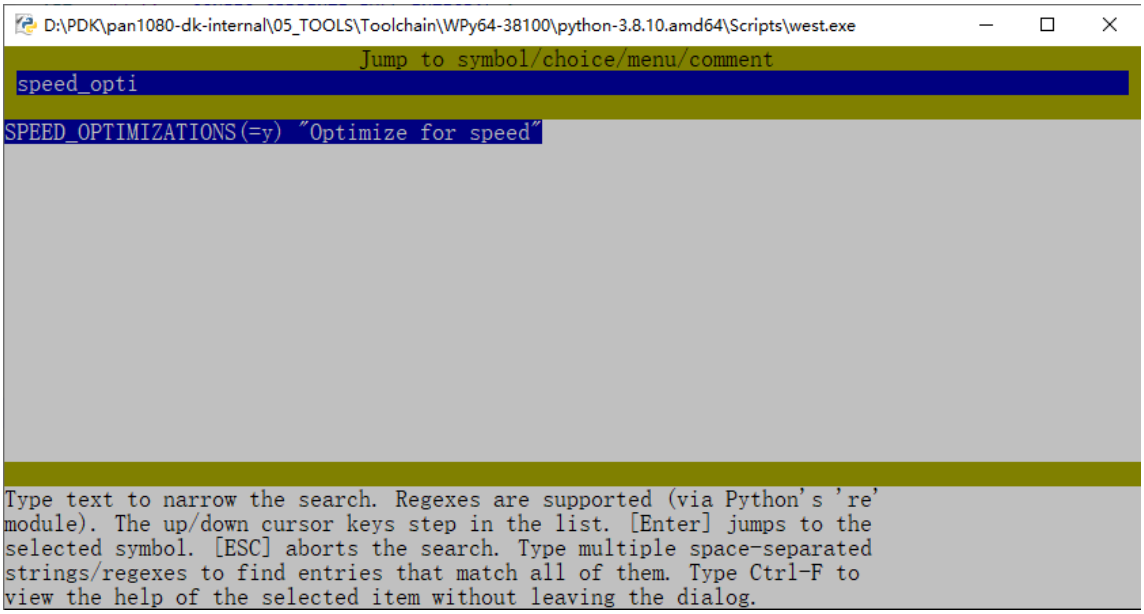
3. 打开的 Menuconfig 主界面如下：



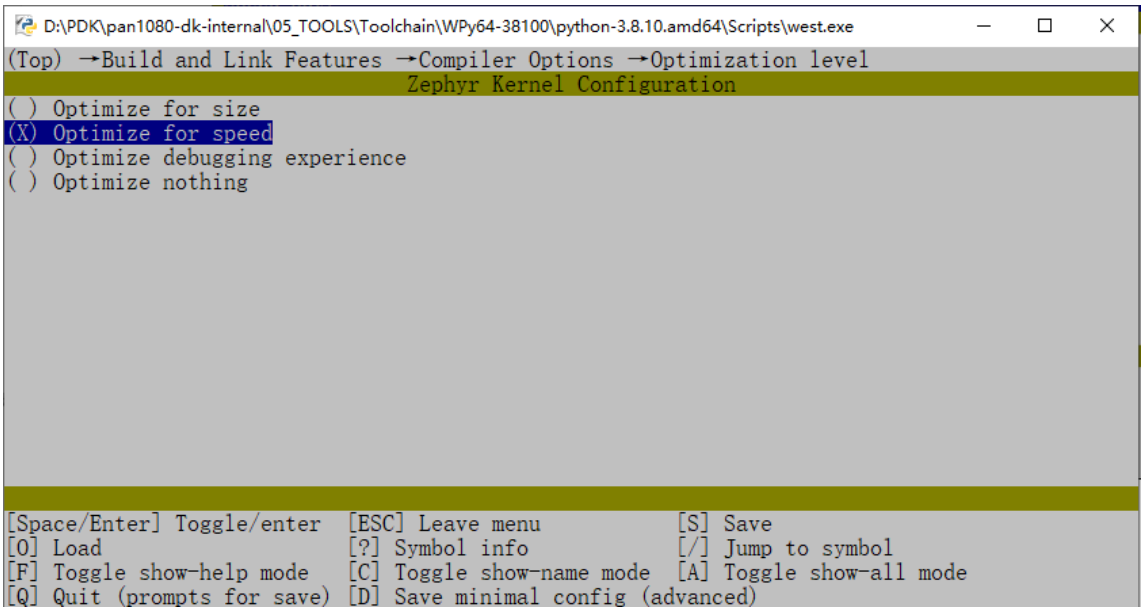
4. 按键盘上的/键进入 Symbol 搜索模式：



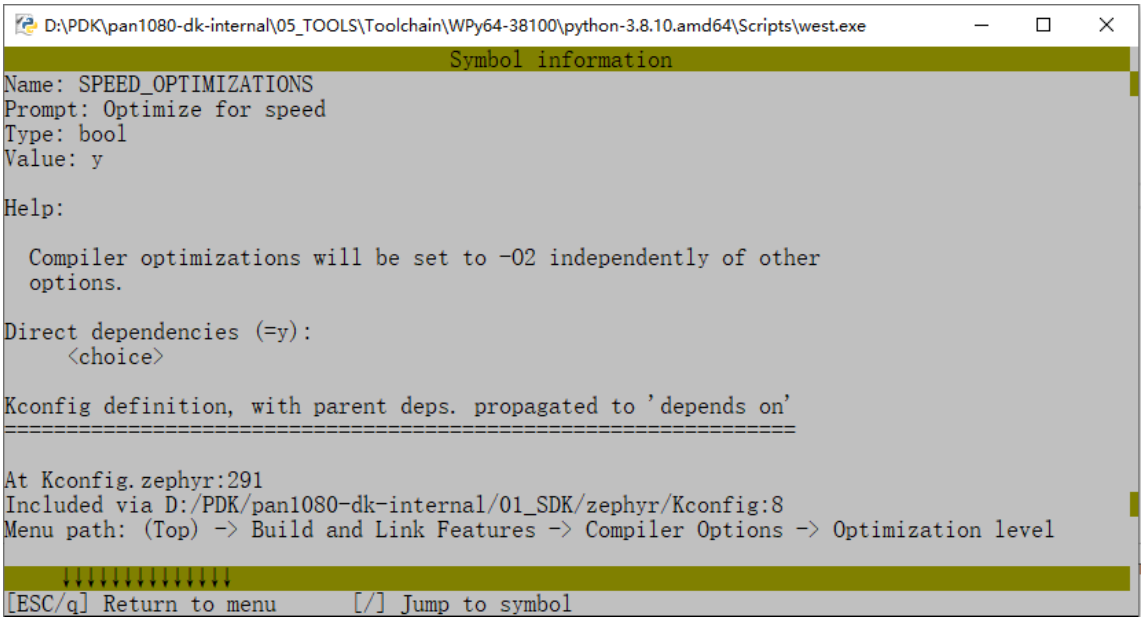
5. 在顶部搜索框中输入想要搜索的 Symbol（SPEED_OPTIMIZATIONS，不区分大小写）：



6. 找到后，按回车键，即可跳转到对应的选项：



7. 向 Menuconfig 工具中输入`?`字符（实际上是按键盘上的 `Shift + /`）键，跳转至当前 Symbol 的定义：



上述界面各字段含义如下：

1. `Name` 字段表示当前 Symbol 名称为：`SPEED_OPTIMIZATIONS`；
2. `Prompt` 字段表示当前 Symbol 在 Menuconfig 菜单中的显示名为：`Optimize for speed`；
3. `Type` 字段表示当前 Symbol 类型为：`bool`；
4. `Name` 字段表示当前 Kconfig Symbol 名称为：`SPEED_OPTIMIZATIONS`；
5. `Value` 字段表示当前 Symbol 的使能情况为：`y`，表示 yes；
6. `Help` 字段是当前 Symbol 的帮助信息（字符串），一般用于描述此 Symbol 的功能或用法；
7. `Direct dependencies` 字段列出了当前 Symbol 的依赖关系，对于 `SPEED_OPTIMIZATIONS` 来说，其直接依赖于`<choice>` 菜单选项，并不依赖其他任何 Kconfig 选项；
8. `At Kconfig.zephyr:291` 及之后的 2 行描述，提供了 2 个信息：
9. 此 Symbol 在 Kconfig 文件中的定义位置为 zephyr 目录顶层 `Kconfig.zephyr` 文件的第 291 行；
10. 此 Symbol 在 Menuconfig 中的菜单层次为 `Menu path: (Top) -> Build and Link Features -> Compiler Options -> Optimization level`；
11. 有些 Symbol Info 比较多，一页无法显示完全，此时可以按键盘的上下键和翻页键进行翻页查看；翻页后可以看到再后面是当前 Symbol 在 Kconfig 源文件中的完整定义信息：

```
config SPEED_OPTIMIZATIONS
        bool "Optimize for speed"
        help
          Compiler optimizations will be set to -O2 independently of other
          options.
```

注：更多 Menuconfig 使用技巧请参考 Zephyr 官方文档：Interactive Kconfig interfaces。

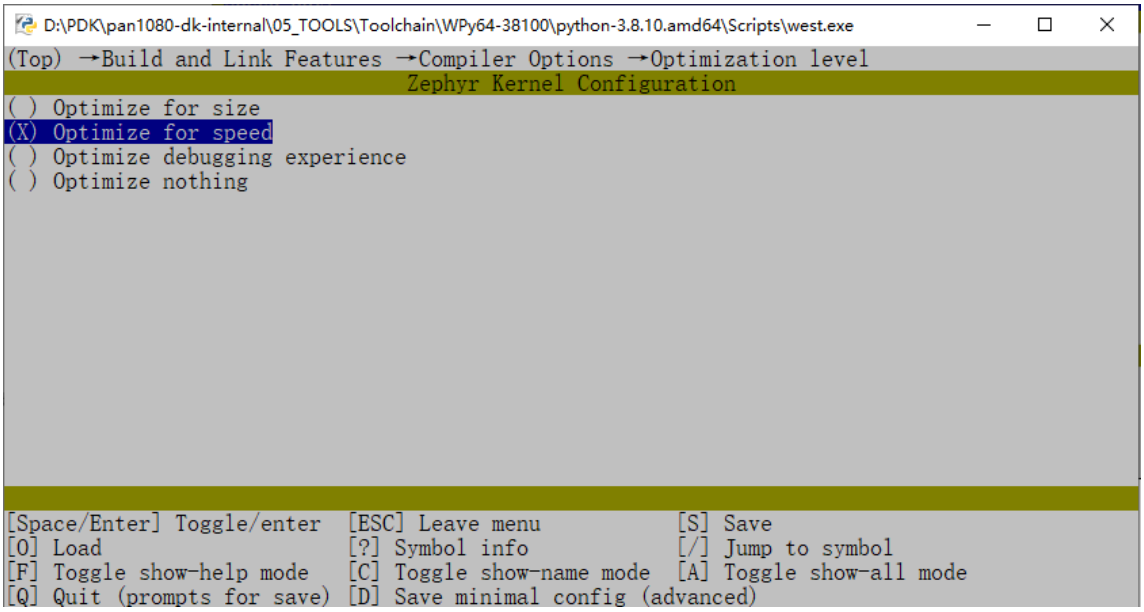
3.4 如何修改 Kconfig

在实际开发过程中，我们经常需要修改 Kconfig 配置，以满足各种不同的需求。

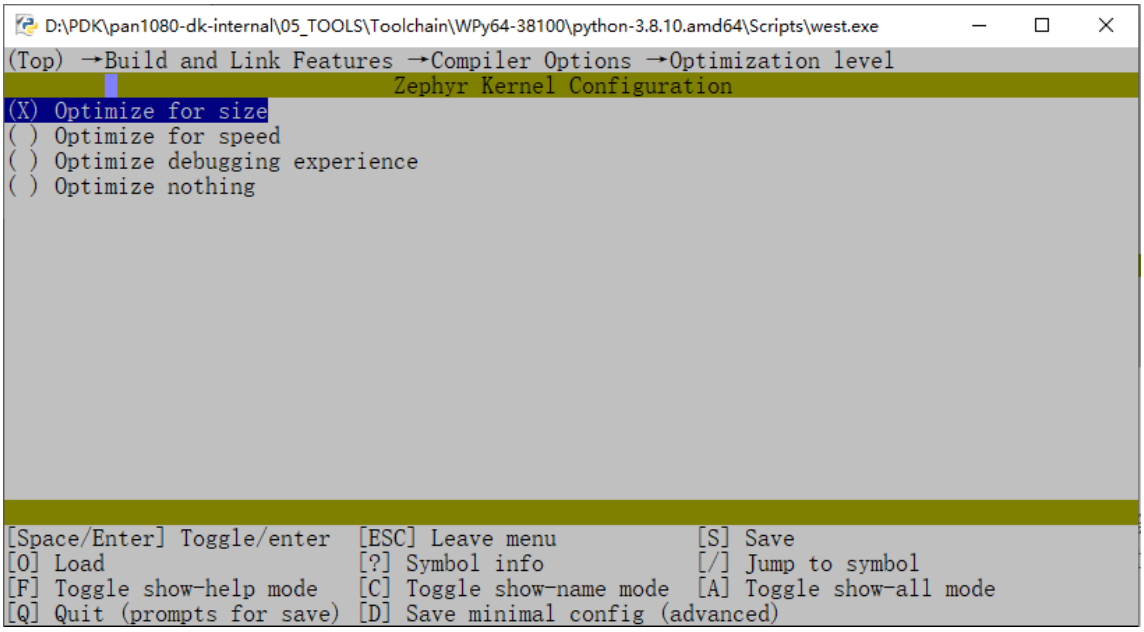
这里仍然以前面编译的 `blinky` 例程为例，分别介绍 2 种比较常用的修改 Kconfig 配置的方法，将此 App 的编译优化选项由 `SPEED_OPTIMIZATIONS` 修改为 `SIZE_OPTIMIZATIONS`。

3.4.1 使用 Menuconfig 进行修改（临时方式）

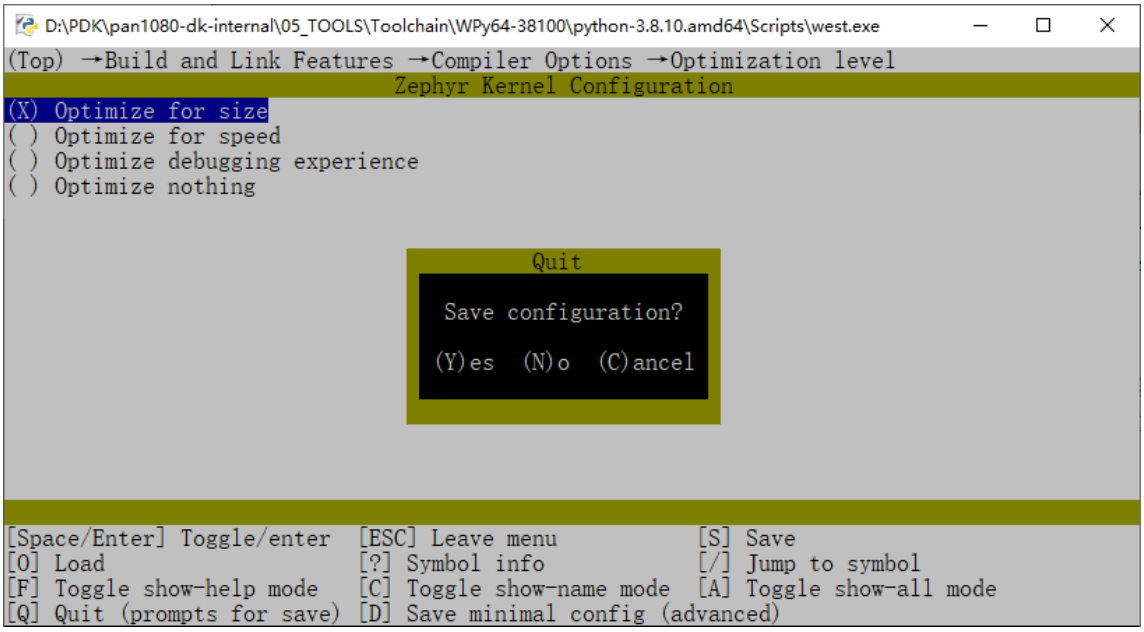
1. 在 3.3 节的步骤 6 中我们看到，Menuconfig 的 `Optimization level` 子界面中，`Optimize for speed` 选项是被勾选上的：



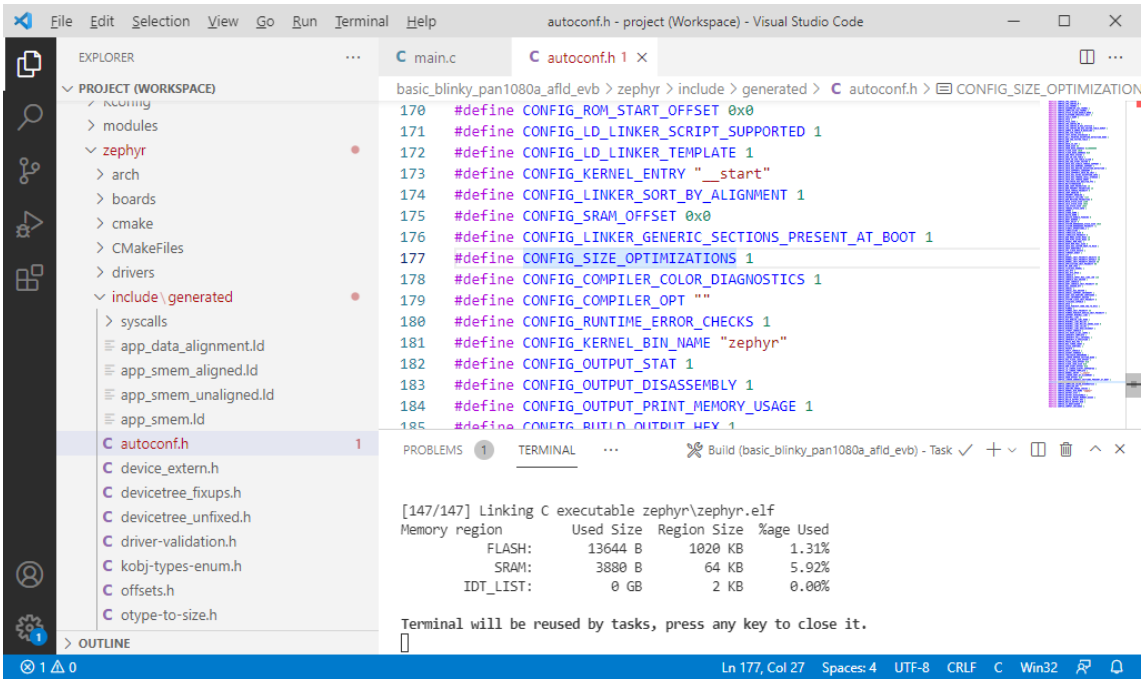
2. 我们此时只需按键盘的上下键，选择 `Optimzie for size` 选项，然后按回车键确定即可：



3. 接着直接按 `Q` 键退出 Menuconfig，在退出的过程中会提示是否保存配置，输入 `Y` 即可实现保存并退出：



4. 再次执行 `Build` 指令，可以看到，`autoconf.h` 中 `#define CONFIG_SPEED_OPTIMIZATIONS 1` 宏定义被替换为 `#define CONFIG_SIZE_OPTIMIZATIONS 1`，说明编译优化选项已经被我们成功修改了：



注意：使用 Menuconfig 修改 Kconfig 配置是**临时的**；之所以这样说，是因为此方法修改后的配置会先更新至 build 输出目录中一个名为`.config` 的文件中（本文开始的框图中有介绍）；而这个文件在执行重新编译（`Rebuild` 或 `Pristine`）操作之后会被删除；之后如果还要修改，则需重新进入 Menuconfig 菜单中进行配置。

3.4.2 **直接编辑** config **相关源文件进行修改（长久方式）** 我们可以使用另一种更直接的方式修改 Kconfig 配置：直接修改 config 相关的源文件。

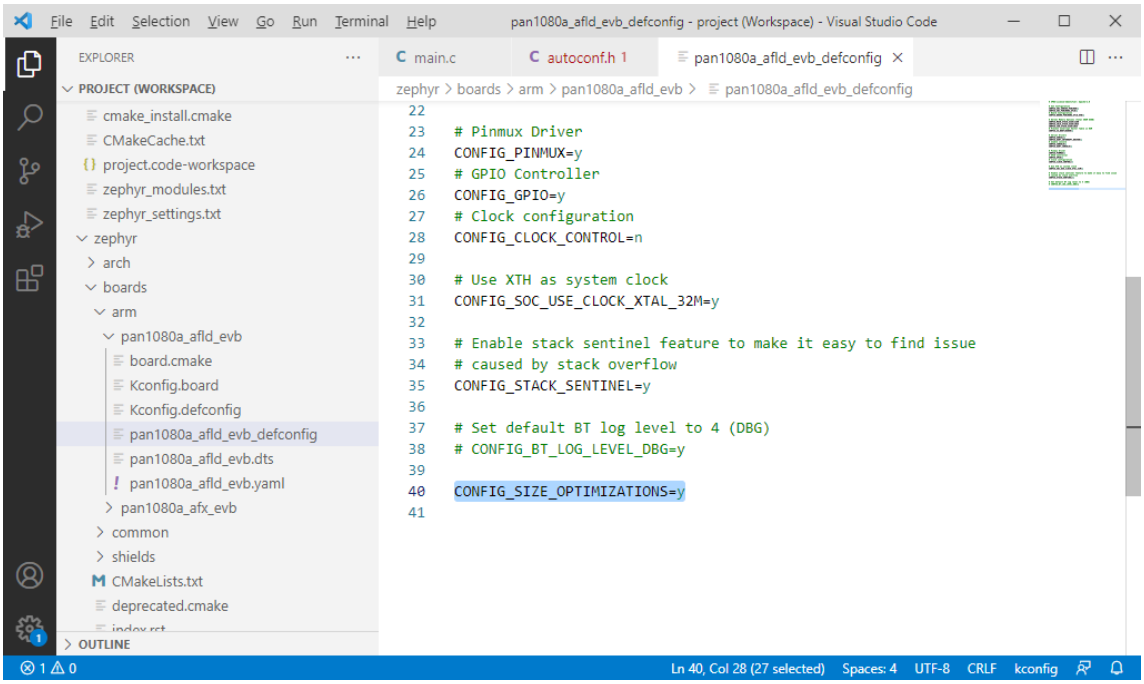
从 config 源文件中修改配置，也有多种途径可供：

1. 直接修改 Kconfig 定义中的 `default` 值，例如，直接将 `Kconfig.zephyr` 中的 `choice` 改为 `default SIZE_OPTIMIZATIONS`：

```
choice
        prompt "Optimization level"
        default NO_OPTIMIZATIONS    if COVERAGE
        default DEBUG_OPTIMIZATIONS if DEBUG
        default SIZE_OPTIMIZATIONS
        help
          Note that these flags shall only control the compiler
          optimization level, and that no extra debug code shall be
          conditionally compiled based on them.
```

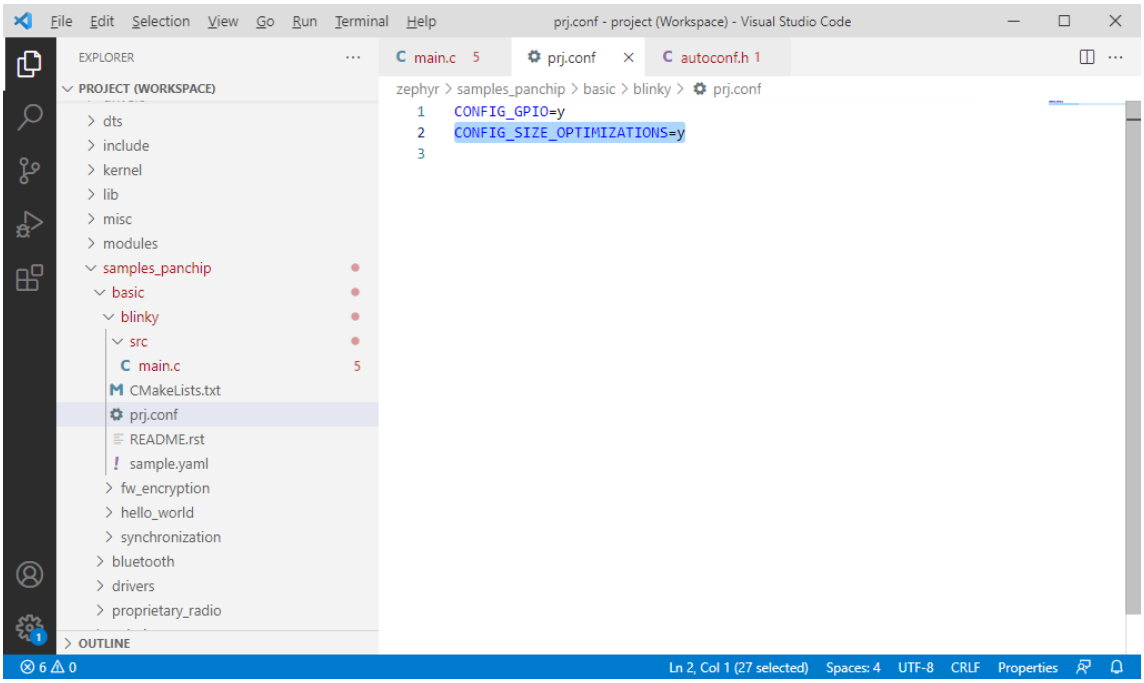
注：此方法会影响到所有 board 编译的 App 工程，一般不推荐使用。

2. 在当前 board 目录中的 `pan1080a_afld_evb_defconfig` 文件中，新增 `CONFIG_SIZE_OPTIMIZATIONS=y` 语句，将当前 board 的编译优化配置强制修改为新的配置：



注：此方法会影响到某个特定 board 编译的所有 App 工程，仅当确实这种需求的时候才推荐这么修改。

3. 在当前 App 工程目录中的 `prj.conf` 文件中，新增 `CONFIG_SIZE_OPTIMIZATIONS=y` 语句，将当前 App 工程的编译优化配置强制修改为新的配置：



注：此方法仅会影响当前 App 工程，在没有特殊需求的场景下，推荐使用这种方式。

注：更多 Kconfig 配置方法请参考 Zephyr 官方文档：Setting Kconfig configuration values。

4.4.4 5 更多相关文档

1. Zephyr Build and Configuration Systems：Zephyr 官方对于 Build 和 Configuration 的整体流程介绍
2. Zephyr Kconfig System：Zephyr 官方 Kconfig 介绍
3. Devicetree Guide：Zephyr 官方 Devicetree 指南
4. Zephyr Kconfig Options：Zephyr 官方支持的所有 Kconfig 选项列表

4.5 Zephyr Board 配置指南

4.5.1 1 概述

Zephyr 使用 Board 的概念将各个板级硬件区分开，于是我们可以使用不同的 board 来编译不同的 App 工程；这种特性为实际项目提供了某种灵活性：如果 App 代码架构规划合理，将有办法使其很容易在不同 board 甚至 SoC 之间进行移植，缩短后续的开发周期。

Zephyr 默认的 board 存放目录为 `zephyr\boards`，而对于 PAN1080 来说，其支持的 boards 目录为 `zephyr\boards\arm`。目前 SDK 中支持 2 种 PAN1080 EVB Board：

1. `pan1080a_afx_evb`：子板主控型号为 `PAN1080UB1`（QFN 封装、1MiB Flash Size、32 Pin）
2. `pan1080a_afld_evb`：子板主控型号为 `PAN1080LB5`（LQFP 封装、1MiB Flash Size、64 Pin）

4.5.2 2 PAN1080 Evaluation Board

下面以 `pan1080a_afld_evb` board 为例，介绍相关文件含义。

`pan1080a_afld_evb` 目录树如下：

```
<home>/01_SDK/zephyr/boards/arm/pan1080a_afld_evb
 board.cmake
 Kconfig.board
 Kconfig.defconfig
 pan1080a_afld_evb.dts
 pan1080a_afld_evb.yaml
 pan1080a_afld_evb_defconfig
```

其中：

- `board.cmake`：板级 CMake 配置；
- `Kconfig.board`：板级 Kconfig 定义；
- `Kconfig.defconfig`：板级 Kconfig 默认值，配置后在 Menuconfig 中可修改；
- `pan1080a_afld_evb.dts`：板级 Devicetree 定义；
- `pan1080a_afld_evb.yaml`：板级测试相关信息（用于 zephyr 自动化单元测试系统）；
- `pan1080a_afld_evb_defconfig`：板级 Kconfig 默认值，配置后在 Menuconfig 中不可修改；

2.1 board.cmake

`board.cmake` 文件用于描述当前板级的 cmake 信息，目前主要是配置了 jlink 相关参数，供 JLink 烧录和调试的时候调用：

```
# SPDX-License-Identifier: Apache-2.0

board_runner_args(jlink "--device=PN108D" "--speed=2000")

include(${ZEPHYR_BASE}/boards/common/openocd.board.cmake)
include(${ZEPHYR_BASE}/boards/common/jlink.board.cmake)
```

> **注**：此文件非必要请不要修改，唯一可考虑修改的参数为`"--speed=2000"`，可以根据实际情况修改 JLink 通信速率。

2.2 Kconfig.board

`Kconfig.board` 文件用于描述当前板级的 Kconfig 定义，目前主要定义了当前 EVB 板的名称为 `BOARD_PAN1080A_AFLD_EVB`，并通过一系列 `select` 关键字，使能了 PAN1080 HAL 层的各个底层 Driver：

```
# PAN1080A-AFLD EVB MCU configuration

# Copyright (c) 2020-2022 Shanghai Panchip Microelectronics Co.,Ltd.
# SPDX-License-Identifier: Apache-2.0

config BOARD_PAN1080A_AFLD_EVB
        bool "PAN1080-AFLD Evaluation Board"
        depends on SOC_PAN1080A_AFLD
        select USE_PANPLAT_ACC
        select USE_PANPLAT_ADC
        select USE_PANPLAT_CLKTRIM
        select USE_PANPLAT_DMAC
        select USE_PANPLAT_FMC
        select USE_PANPLAT_GPIO
        select USE_PANPLAT_I2C
        select USE_PANPLAT_I2S
        select USE_PANPLAT_KSCAN
        select USE_PANPLAT_LOWPOWER
```

（下页继续）

(续上页)

```
        select USE_PANPLAT_PWM
        select USE_PANPLAT_PRF
        select USE_PANPLAT_QDEC
        select USE_PANPLAT_SPI
        select USE_PANPLAT_TIMER
        select USE_PANPLAT_UART
        select USE_PANPLAT_USB
        select USE_PANPLAT_WDT
        select USE_PANPLAT_WWDT
```

注：此文件非必要请不要修改，但如果希望新增一些板级 Kconfig 定义，则直接在文件结尾新增定义即可。

2.3 Kconfig.defconfig

`Kconfig.defconfig` 文件用于描述当前板级的默认值，并且此处配置的默认值是非强制的；换句话说，即使在此处配置了默认值，后续仍然可以在 Menuconfig 中或者 App 工程目录的 `prj.conf` 文件中进行强制修改：

```
# PAN1080A-AFLD EVB MCU configuration

# Copyright (c) 2020-2022 Shanghai Panchip Microelectronics Co.,Ltd.
# SPDX-License-Identifier: Apache-2.0

if BOARD_PAN1080A_AFLD_EVB

config BOARD
        default "pan1080a_afld_evb"

# BT_CTLR depends on BT. When BT is enabled we should default to also
# enabling the controller.
config BT_CTLR
        default y if BT

# Use Panchip BLE Link Layer instead of Zephyr's
choice BT_LL_CHOICE
        default BT_LL_PANCHIP if BT_CTLR
endchoice

if BT_LL_PANCHIP

config SOC_USE_CLOCK_XTAL_32M
        default y
config SOC_USE_DPLL_64M_CLK_REF_XTH
        default n
config SYS_CLOCK_HW_CYCLES_PER_SEC
        default 48000000
config SOC_USE_DPLL_48M_CLK_REF_XTH
        default y
config SOC_DCDC_PAN1080
        default n
config SOC_USE_MINIMAL_PERIPHERAL
        default n
config BT_HCI_TX_STACK_SIZE
        default 2048

endif # BT_LL_PANCHIP

# Change default Logging configurations
if LOG
```

(下页继续)

(续上页)

```
config LOG_BUFFER_SIZE
        default 5120

config LOG_STRDUP_BUF_COUNT
        default 8

config LOG_DEFAULT_LEVEL
        default 3

endif # LOG

if USB_DEVICE_STACK

config USB_DEVICE_VID
        default 0x2FE4

config USB_DEVICE_PID
        default 0x0001

config USB_DEVICE_MANUFACTURER
        default "Panchip"

config USB_DEVICE_REMOTE_WAKEUP
        default n

endif # USB_DEVICE_STACK

# Increase ztest_thread_stack[] from default 1024 to 2048, since
# the default size is not enough, such as when test fcb_raw cases.
config ZTEST_STACKSIZE
        default 2048
        depends on ZTEST

endif # BOARD_PAN1080A_AFLD_EVB
```

注：此文件可以根据实际需要进行修改。

2.4 pan1080a_afld_evb.dts

pan1080a_afld_evb.dts 文件用于描述当前板级的 Devicetree 定义：

```
/*
 * Copyright (c) 2020-2022 Shanghai Panchip Microelectronics Co.,Ltd.
 *
 * SPDX-License-Identifier: Apache-2.0
 */

/dts-v1/;
#include <mem.h>
#include <panchip/pan1080/pan1080a_afld.dtsi>

/ {
        model = "Panchip PAN1080A-AFLD Evaluation Board";
        compatible = "panchip,pan1080a-afld-evb", "panchip,pan1080a-afld";

        chosen {
                zephyr,console = &uart1;
                zephyr,shell-uart = &uart1;
                zephyr,bt-mon-uart = &uart1;
```

(下页继续)

(续上页)

```
                zephyr,bt-c2h-uart = &uart1;
                zephyr,sram = &sram0;
                zephyr,flash = &flash0;
                zephyr,code-partition = &slot0_partition;
        };

        soc {
                pin-controller@40030000 {
                        /* port, pin, pinmux_name, pinmux_sel [, flag1, ... ] */
                        DT_PAN_PINS(p4, 3, uart0_rx, PAN1080_PIN_FUNC_P43_UART0_RX, input-
↪enable);
                        DT_PAN_PINS(p0, 7, uart1_rx, PAN1080_PIN_FUNC_P07_UART1_RX, input-
↪enable);
                        DT_PAN_PINS(p0, 4, qdec_x0, PAN1080_PIN_FUNC_P04_QDEC_X0, bias-pull-up,
↪ input-enable);
                        DT_PAN_PINS(p0, 5, qdec_x1, PAN1080_PIN_FUNC_P05_QDEC_X1, bias-pull-up,
↪ input-enable);
                        DT_PAN_PINS(p3, 1, spi0_miso, PAN1080_PIN_FUNC_P31_SPI0_MISO, input-
↪enable);
                        DT_PAN_PINS(p4, 0, i2c0_sda, PAN1080_PIN_FUNC_P40_I2C0_SDA, bias-pull-
↪up, input-enable);
                        DT_PAN_PINS(p4, 1, i2c0_scl, PAN1080_PIN_FUNC_P41_I2C0_SCL, bias-pull-
↪up, input-enable);
                };
        };

        leds {
                compatible = "gpio-leds";
                led_red: led_r {
                        gpios = <&p2 1 GPIO_ACTIVE_HIGH>;
                        label = "LED_RED";
                };
        };

        pwmleds {
                compatible = "pwm-leds";
                pwm_led_red: pwm_led_r {
                        pwms = <&pwm0 4 PWM_POLARITY_NORMAL>;
                        label = "RGB_LED_RED";
                };
                pwm_led_green: pwm_led_g {
                        pwms = <&pwm0 5 PWM_POLARITY_NORMAL>;
                        label = "RGB_LED_GREEN";
                };
                pwm_led_blue: pwm_led_b {
                        pwms = <&pwm0 6 PWM_POLARITY_NORMAL>;
                        label = "RGB_LED_BLUE";
                };
        };

        buttons {
                compatible = "gpio-keys";
                key1: k1 {
                        label = "KEY_1";
                        gpios = <&p0 4 GPIO_ACTIVE_LOW>;
                };
                key2: k2 {
                        label = "KEY_2";
                        gpios = <&p0 5 GPIO_ACTIVE_LOW>;
                };
        };
```

(下页继续)

(续上页)

```
        aliases {
                led0 = &led_red;
                sw0 = &key1;
                pwm-led0 = &pwm_led_red;
                pwm-led1 = &pwm_led_green;
                pwm-led2 = &pwm_led_blue;
                pwm-0 = &pwm0;
        };
};

&acc {
        status = "okay";
};

&p0 {
        status = "okay";
};

&p1 {
        status = "okay";
};

&p2 {
        status = "okay";
};

&p3 {
        status = "okay";
};

&p4 {
        status = "okay";
};

&p5 {
        status = "okay";
};

&uart0 {
        current-speed = <115200>;
        pinctrl-0 = <&p4_4_uart0_tx &p4_3_uart0_rx>;
        status = "okay";
};

&uart1 {
        current-speed = <921600>;
        pinctrl-0 = <&p0_6_uart1_tx &p0_7_uart1_rx>;
        status = "okay";
};

&timer0 {
        freq = <16000000>;
        status = "okay";
};

&timer1 {
        freq = <8000000>;
        status = "okay";
};
```

(下页继续)

(续上页)

```
&timer2 {
        freq = <16000000>;
        status = "okay";
};

&pwm0 {
        pinctrl-0 = <&p1_0_pwm0_ch4 &p1_1_pwm0_ch5 &p1_6_pwm0_ch6>;
        status = "okay";
};

&adc {
        #io-channel-cells = <1>;
        status = "okay";
};

&qdec {
        event-threshold = <100>;
        polarity = "polarity-low";
        resolution = "cnt-resolution-1x";
        filter-threshold = "filter-threshold-3";
        pinctrl-0 = <&p0_4_qdec_x0 &p0_5_qdec_x1>;
        status = "okay";
};

&i2c0 {
        pinctrl-0 = <&p4_0_i2c0_sda &p4_1_i2c0_scl>;
        status = "okay";
};

&spi0 {
        pinctrl-0 = <&p3_0_spi0_mosi &p3_1_spi0_miso &p0_2_spi0_cs &p0_3_spi0_clk>;
        #address-cells = <1>;
        #size-cells = <0>;
        status = "okay";
};

&flash0 {
        partitions {
                compatible = "fixed-partitions";
                #address-cells = <1>;
                #size-cells = <1>;

                boot_partition: partition@0 {
                        label = "mcuboot";
                        reg = <0x00000000 0x00010000>;
                        read-only;
                };
                slot0_partition: partition@10000 {
                        label = "image-0";
                        reg = <0x00010000 0x00060000>;
                };
                slot1_partition: partition@70000 {
                        label = "image-1";
                        reg = <0x00070000 0x00060000>;
                };
                scratch_partition: partition@D0000 {
                        label = "image-scratch";
                        reg = <0x000D0000 0x00020000>;
                };
                storage_partition: partition@f0000 {
                        label = "storage";
```

(下页继续)

(续上页)

```
                        reg = < 0xf0000 0xF000 >;
                };
        };
};
```

注：此文件请谨慎修改，因为修改后可能会导致某些例程执行条件发生变化。如果某个 App 确实需要修改 dts 文件，则可以优先考虑通过 DTS Overlay 的方式进行修改，相关说明请参考Zephyr 配置系统指南文档中的介绍。

2.5 pan1080a_afld_evb.yaml

pan1080a_afld_evb.yaml 文件用于描述当前板级的测试环境相关信息，供 zephyr 自动化单元测试系统解析，以了解当前的板级硬件资源情况：

```
identifier: pan1080a_afld_evb
name: PAN1080A-AFLD Evaluation Board
type: mcu
arch: arm
toolchain:
  - zephyr
  - gnuarmemb
  - xtools
ram: 64
flash: 1020
supported:
  - adc
  - counter
  - gpio
  - pwm
  - serial
```

注：仅供 Zephyr 自动化单元测试系统使用，一般实际项目中可以不用关心此文件。

2.6 pan1080a_afld_evb_defconfig

pan1080a_afld_evb_defconfig 文件用于配置当前板级的默认值，并且此处配置的默认值是强制的，注意这点与 Kconfig.defconfig 不同；换句话说，如果在此处配置了默认值，那么后续将无法在 Menuconfig 中或者 App 工程目录的 prj.conf 文件中再进行修改：

```
# SPDX-License-Identifier: Apache-2.0

# SoC Configuration
CONFIG_SOC_SERIES_PAN1080=y
CONFIG_SOC_PAN1080A_AFLD=y
# Board Configuration
CONFIG_BOARD_PAN1080A_AFLD_EVB=y

# Kernel Memory Options (Total SRAM 64KB)
CONFIG_MAIN_STACK_SIZE=1280
CONFIG_IDLE_STACK_SIZE=1024
CONFIG_ISR_STACK_SIZE=1024
# Prevent Interrupt Vector Table in RAM
CONFIG_IS_BOOTLOADER=y

# Serial Drivers
CONFIG_SERIAL=y
CONFIG_UART_INTERRUPT_DRIVEN=y
# enable console
```

(下页继续)

(续上页)

```
CONFIG_CONSOLE=y
CONFIG_UART_CONSOLE=y

# Pinmux Driver
CONFIG_PINMUX=y
# GPIO Controller
CONFIG_GPIO=y
# Clock configuration
CONFIG_CLOCK_CONTROL=n

# Use XTH as system clock
CONFIG_SOC_USE_CLOCK_XTAL_32M=y

# Enable stack sentinel feature to make it easy to find issue
# caused by stack overflow
CONFIG_STACK_SENTINEL=y

# Set default BT log level to 4 (DBG)
# CONFIG_BT_LOG_LEVEL_DBG=y

CONFIG_SIZE_OPTIMIZATIONS=y
```

注：此文件可以根据实际需要进行修改。

4.5.3 3 创建一个自己的 Board

3.1 为什么要创建自己的 Board

由前面的介绍我们知道，SDK 中已经存在 2 个对应 PAN1080 EVB 的 board，其中的 DTS 文件描述了 EVB 的板级硬件资源以及各板级硬件模块之间的连接关系，Kconfig 文件中则配置了能保证各个例程均能正确运行所需要开启的 Zephyr 功能。

而然，对于一个实际项目来说，一定有单独的板级硬件，其中的硬件资源可能与 EVB 上的完全不同了，并且使用过程中所需的板级 Kconfig 配置也大概率会与 EVB 不同，因此，为自己的项目重新创建一个 zephyr board 是一个必要且合理的选择。

3.2 如何创建 Board

创建 Board 最简单的方法是从当前已有的模板中修改，方法如下（假设新的 board 硬件主控 SoC 型号与 pan1080a_afld_evb 相同，也为 `PAN1080LB5`，并将新的 board 命名为 `my_custom_board`）：

1. 将 `zephyr\boards\arm` 路径下的 `pan1080a_afld_evb` 目录整体拷贝一份，并重新命名为 `my_custom_board`；
2. 进入 `my_custom_board` 子目录：
    1. 将 `pan1080a_afld_evb.yaml` 文件删除；
    2. 分别将 `pan1080a_afld_evb.dts` 和 `pan1080a_afld_evb_defconfig` 文件重命名为 `my_custom_board.dts` 和 `my_custom_board_defconfig`；
    3. 打开 `Kconfig.board` 文件，将其中的 Kconfig 定义修改为：

    ```
    config BOARD_MY_CUSTOM_BOARD
            bool "My Custom Board"
    ```

    4. 打开 `Kconfig.defconfig` 文件，将其开头的 if 判断条件，以及 BOARD 默认值改为：

```
if BOARD_MY_CUSTOM_BOARD

config BOARD
        default "my_custom_board"

...

endif # BOARD_MY_CUSTOM_BOARD
```

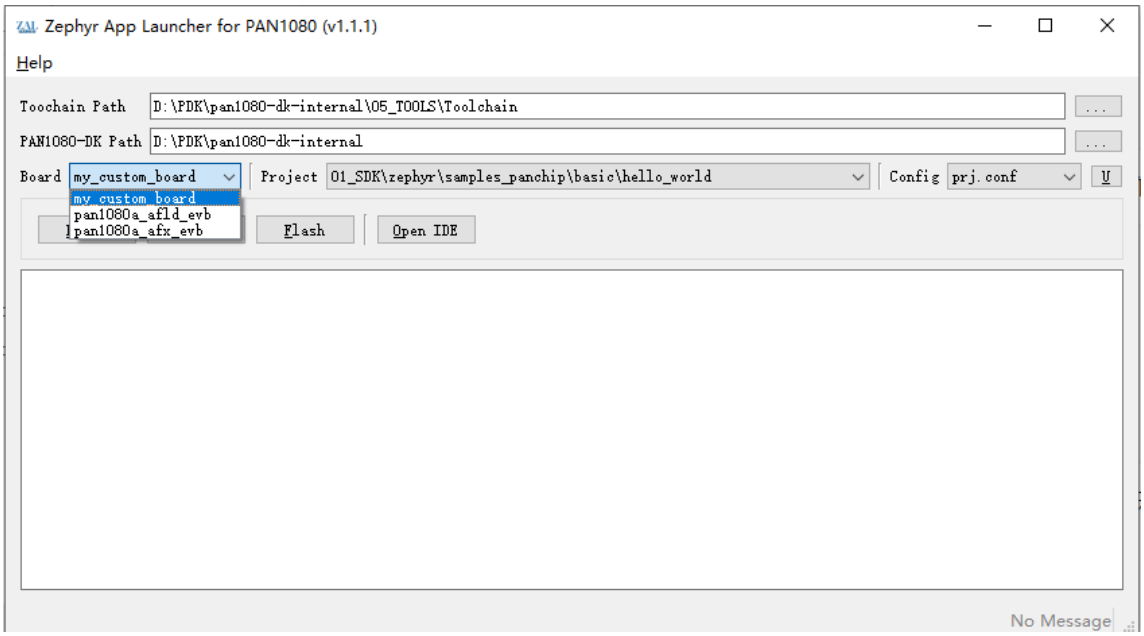
5. 打开 `my_custom_board_defconfig` 文件，将当前的 Board Configuration 修改为新的 board 名称：

```
...

# Board Configuration
CONFIG_BOARD_MY_CUSTOM_BOARD=y

...
```

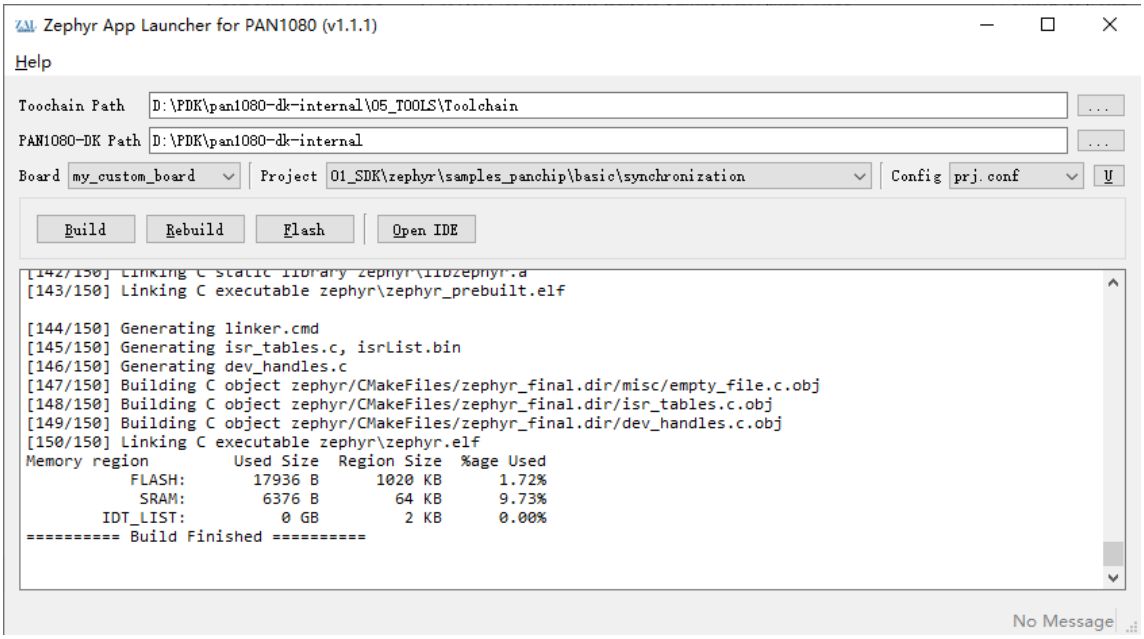
6. 打开 `my_custom_board.dts` 文件，修改根节点下的 `model` 与 `compatible` 值，以与 EVB 作区分：

```
...

/ {
        model = "My custom Board";
        compatible = "vendor,my-custom-board", "panchip,pan1080a-afld";

...
```

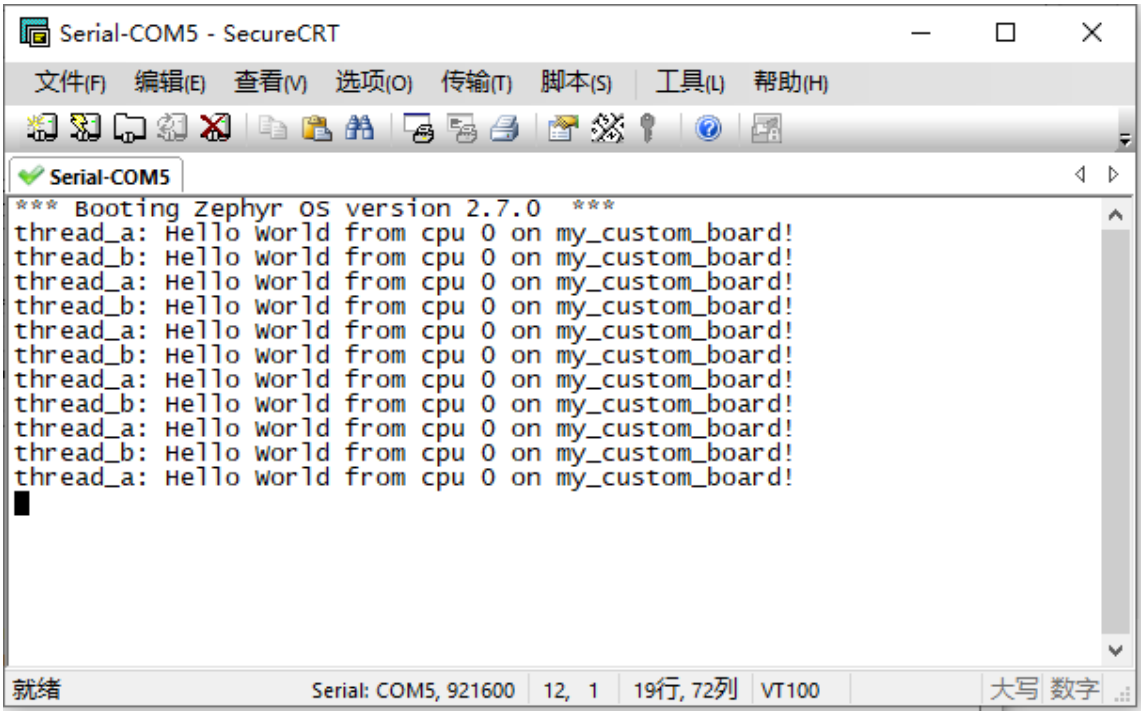
3. 至此，一个除名称不同外，各种配置完全与 EVB 相同的可用 board 就创建好了；这时候我们打开 `ZAL` 工具，在 Board 下拉菜单上可以看到新创建的 board 已经被成功识别：



4. 我们选择这个新增的 `my_custom_board`，编译 synchronization 例程：



5. 直接点击 `Flash` 按钮进行烧录，查看 UART1 串口 Log，可以看到例程成功运行，并且 board 名称也已经与我们新修改的相同：



注意：在实际项目中，仅仅将 Board 重命名是不够的，后续还应根据实际需要：

1. 修改 DTS 文件中的其他部分，以符合实际的板级硬件情况；
2. 修改 Kconfig 文件中的其他部分，以符合当前项目的实际需求；

4.5.4 4 更多相关文档

1. Zephyr Board Porting Guide：Zephyr 官方板级目录移植指南

4.6 SoC App 开发指南

本文介绍 PAN1080 SDK APP 开发的基础知识。

4.6.1 1 概述

PAN1080 SDK 基于 Zephyr OS 开发，其构建系统（Build System）使用 CMake。

`01_SDK/zephyr` 目录下存放有 Zephyr OS 的核心代码、内核配置信息、SoC 定义、开发板定义以及 Panchip 提供的例程等。

从文件结构上来说，构建系统允许我们将自己的 APP 工程创建在任意目录下；只要配置正确，构建系统可以在编译 APP 工程的时候，自动识别到 SDK 中 Zephyr 的路径。

一个最简单的 APP 工程文件结构如下所示：

```
<work_dir>/my_app
   CMakeLists.txt
   prj.conf
   src
       main.c
```

其中：

- CMakeLists.txt: 构建系统从此文件中查找待编译 APP 源码，并将 APP 目录与 Zephyr 链接起来，使得 APP 可以编译、配置、使用 Zephyr 提供的各种功能。
- **配置文件**: APP 需要提供一个 Kconfig 配置文件（通常称为 `prj.conf`），其中包含一个或多个 Zephyr 内核配置信息，这些配置与特定的芯片（SoC）配置、特定的开发板（Board）配置等配置信息合并，生成最终的配置文件（`.config` 文件与 `autoconf.h` 文件）。
- APP **源码文件**: APP 需要包含一个或多个 C 语言或汇编源码文件，这些文件通常位于名为 `src` 的目录下。

APP 工程创建后，我们即可以使用 `west build` 命令触发编译操作（其内部是调用 CMake ），编译过程中会自动生成单独的 Build **（构建）目录**，其中存放编译输出的所有文件。

下面介绍如何创建、构建和运行一个自定义的 APP。

4.6.2 2 Zephyr 目录结构

- Zephyr 的核心代码位于 `01_SDK/zephyr` 目录下，其中包含一些重要的文件：
  - `CMakeLists.txt`：CMake 构建系统的顶层文件，包含构建 Zephyr 核心代码所需的 CMake 信息。
  - `Kconfig`：Kconfig 顶层文件，其直接引用另一个顶层文件 `Kconfig.zephyr`。
  - `west.yml`：west manifest 文件，其中列出了 west 工具能够识别和管理的外部模块目录。
- Zephyr 的核心代码还包括如下这些顶层目录（每个顶层目录下还包括一个或多个子目录）：
  - `arch`：存放芯片 CPU 架构级信息，由于 PAN1080 是 ARM 架构，因此其中只保留了 ARM 架构信息。
  - `soc`：存放芯片 SoC 级信息及一些默认配置。
  - `boards`：存放板级信息，比如 PAN1080 各种 EVB 开发板。
  - `doc`：Zephyr 官方文档，其内容与 Zephyr 文档官网一致。
  - `drivers`：存放设备驱动代码。
  - `dts`：存放 DeviceTree（芯片硬件初始化配置）信息。
  - `include`：存放 Zephyr 公开的 API 头文件。

- `kernel`：存放 Zephyr OS 内核代码。
- `lib`：存放库文件，包含一个简单的标准 C 语言库等。
- `misc`：其他一些不便分类的文件。
- `samples_panchip`：存放 Panchip 提供的 PAN1080 官方例程。
- `scripts`：存放编译测试相关的脚本文件。
- `cmake`：存放 CMake 构建 Zephyr App 所需的相关配置和脚本文件。
- `subsys`：Zephyr 子系统，包括：USB Device Stack（USB 设备栈）、File System（文件系统）、Bluetooth Host and Controller（低功耗蓝牙 Host 端与 Controller 端）等实现代码。
- `share`：存放一些额外的架构无关的数据，目前包含 Zephyr CMake Package。

4.6.3 3 创建一个 APP 工程

创建一个 APP 最简单的方式，是从 `01_SDK/zephyr/samples_panchip` 目录下，Copy 一个例程，并将其中的非必要文件（如 `README.rst` 和 `sample.yaml`）删除即可。

下面介绍如何重新创建一个 APP 工程。

1. 在合适的位置创建一个空目录，作为 APP 工程目录（此处以存放在 PAN1080 Development Kit 同级目录下为例）：

   虽然 SDK 中的所有例程 APP 均存放在 `zephpyr/sample_panchip` 子目录下，但实际上，zephyr 的构建系统支持将 APP 工程创建在任意不包含中文和空格的路径下。

2. 将 APP 程序源码，存放在其中的 `src` 子目录下，此处假设我们创建了一个名为 `main.c` 的源文件。
3. 在 `my_app` 目录中创建一个 `CMakeLists.txt` 文件，其内容如下：

```
# Find Zephyr. This also loads Zephyr's build system.
cmake_minimum_required(VERSION 3.20.0)
find_package(Zephyr)
project(my_app)

# Add your source file to the "app" target. This must come after
# find_package(Zephyr) which defines the target.
target_sources(app PRIVATE src/main.c)
```

   其中：

   - `cmake_minimum_required` 指定了 CMake 的最低版本需求；
   - `find_package(Zephyr)` 表示 APP 会与 Zephyr 一起编译；
   - `project(my_app)` 指定了当前 APP 工程的名称；
   - `target_sources(app PRIVATE src/main.c)` 表示将 APP 工程 `src` 子目录下的 `main.c` 文件加入构建系统参与编译；

4. 增加一个名为 `prj.conf` 的文件，用于配置 Kconfig 选项；

   例如，我们想在 APP 中使用 Zephyr 的 GPIO Driver，则在其中填入：

   ```
   CONFIG_GPIO=y
   ```

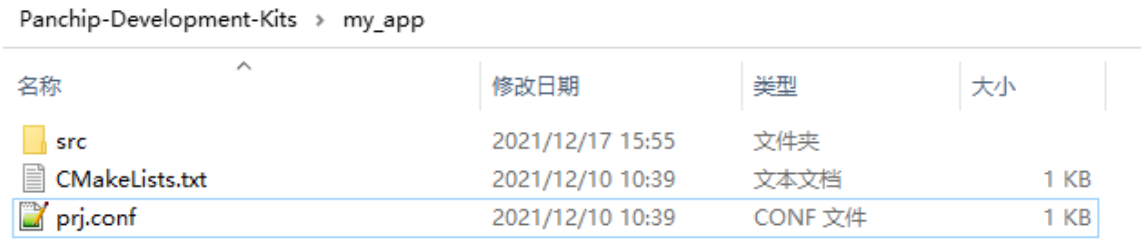
   又如，我们想开启 C++ 支持，则在其中填入：

   ```
   CONFIG_CPLUSPLUS=y
   ```

5. （可选）如果需要配置额外的 DeviceTree 信息，还可以在 APP 目录下增加 Overlay 文件，以覆盖默认的配置。

   关于 Devicetree Overlay 的更多信息，请参考 Zephyr 官方文档：如何配置 DeviceTree Overlay。

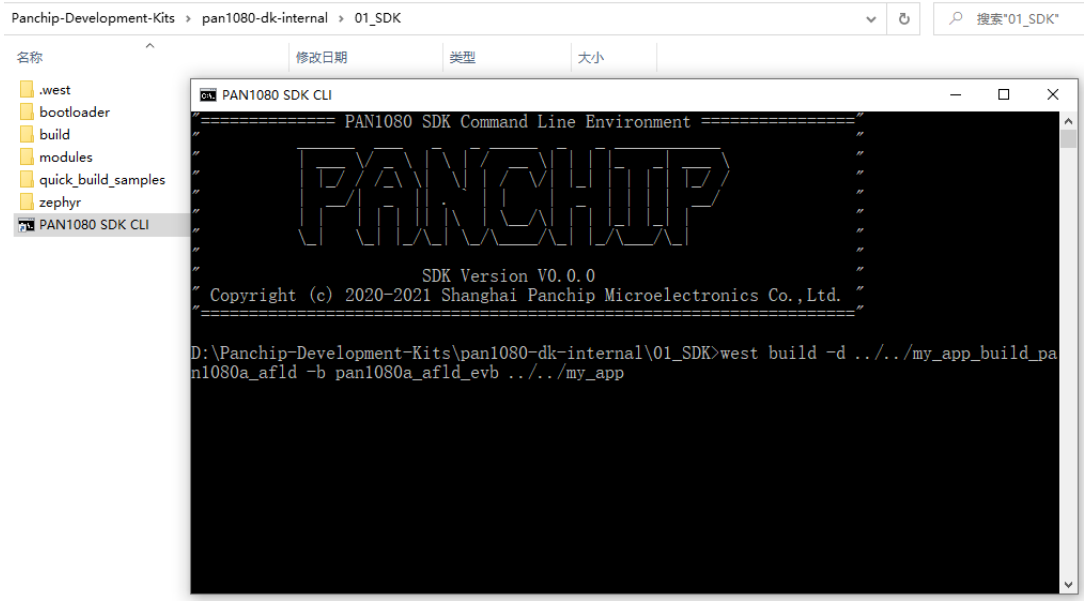
6. 创建完成后的目录结构如图所示：



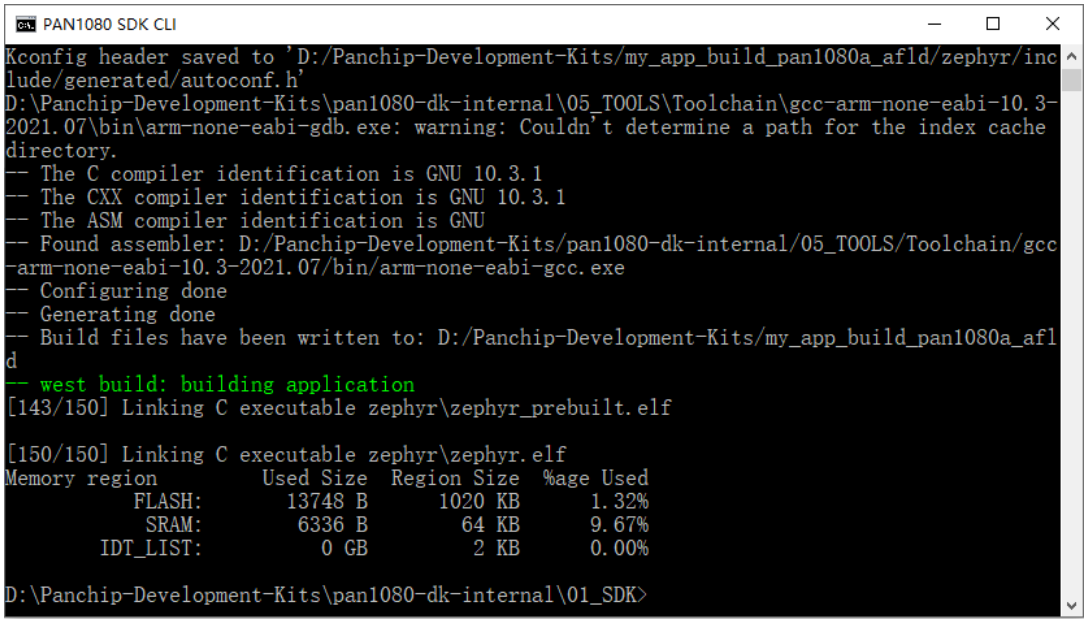
7. APP 基础框架创建完成后，即可使用 west 工具进行编译：

   - 打开 PAN1080 SDK CLI（位于 `01_SDK` 目录下），输入如下命令：

     ```
     west build -d ../../my_app_build_pan1080a_afld -b pan1080a_afld_evb ../../my_app
     ```



   - 编译成功：



8. 编译完成，可以继续执行烧录和调试，方法请参考SDK 开发环境介绍

4.7 BLE 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。

4.7.1 1 GAP

1.1 使用静态地址

开发过程中，我们有时想使用静态（随机）地址，可以在 bt_enable 之前，先设置一下设备地址，如下：

```
#include <bluetooth/addr.h>
#include <bluetooth/hci_ble.h>

void main(void)
{
        bt_addr_t addr = {
                .val = { 0x55, 0x44, 0x33, 0x22, 0x11, 0xC0 }
        };

        bt_static_address_init(&addr);

        ...
        bt_enable(NULL);
        ...
}
```

需要注意的是，48-bit 静态地址的最高两位会被强制写为 1。

4.7.2 2 GATT

2.1 Service

16-bit 或 128-bit 的 UUID 唯一决定了 Service/Characteristic.

- 16-bit UUID 用于标准 Bluetooth Service/Characteristic
- 128-bit UUID 用于厂商自定义的一些服务

2.1.1 Bluetooth Service

2.1.1.1 GAP Service

这是一个基础服务，默认开启，不建议用户修改。

如果确实需要修改，建议使用 `CONFIG_BT_GAP_SERVICE=n` 关掉这个服务，然后在应用层重新实现，这样可以避免代码更新导致的冲突。

源码路径：`zephyr\subsys\bluetooth\host\gatt.c`。

Service or Characteristics	Kconfig	Description	ROM	SRAM
GAP Service	`CONFIG_BT_GAP_SERVICE`	Enable GAP service (Enabled by default)		
Device Name Characteristic	`N/A`	Device name characteristic (Always enable)	-	-
Appearance Characteristic	`N/A`	Appearance characteristic (Always enable)	-	-
Central Address Resolution Characteristic	`CONFIG_BT_CENTRAL` `CONFIG_BT_PRIVACY`	Depends on Central Role & Privacy Feature	-	-
Service Changed Characteristic	`CONFIG_BT_GAP_PERIPHERAL_PREF_PARAMS`	Configure peripheral preferred connection parameters		

2.1.1.2 GATT Service

这是一个基础服务，默认开启，不建议用户修改。

如果确实需要修改，建议使用 `CONFIG_BT_GATT_SERVICE=n` 关掉这个服务，然后在应用层重新实现，这样可以避免代码更新导致的冲突。

源码路径：`zephyr\subsys\bluetooth\host\gatt.c`。

Service or Characteristics	Kconfig	Description	ROM	SRAM
GATT Service	`CONFIG_BT_GATT_SERVICE`	Enable GATT service (Enabled by default)		
Service Changed Characteristic	`CONFIG_BT_GATT_SERVICE_CHANGED`	GATT Service Changed support		
Client Supported Features & Database Hash Characteristic	`CONFIG_BT_GATT_CACHING`	GATT Caching support		
Server Supported Features Characteristic	`CONFIG_BT_EATT`	Enhanced ATT Bearers support [EXPERIMENTAL]		

2.1.1.3 Battery Service

源码路径：`zephyr\subsys\bluetooth\services\bas.c`。

Service or Characteristics	Kconfig	Description	ROM	SRAM
Battery Service	`CONFIG_BT_BAS`	Enable GATT Battery service		
Battery Level Characteristic	`N/A`	BAS Battery level characteristic (Always enable)	-	-

APIs

- bt_bas_get_battery_level
- bt_bas_set_battery_level

  需要包含头文件 `#include <bluetooth/services/bas.h>`

2.1.1.4 Device Information Service 源码路径：zephyr\subsys\bluetooth\services\dis.c。

Service or Characteristics	Kconfig	Description	ROM	SRAM
Device Information Service	CONFIG_BT_DIS	Enable GATT Device Information service		
	CONFIG_BT_DIS_SETTINGS	Enable Settings usage in Device Information Service		
	CONFIG_BT_DIS_STR_MAX	Maximum size in bytes for DIS strings	-	-
Model Number String Characteristic	N/A	DIS Model number string characteristic (Always enable)	-	-
	CONFIG_BT_DIS_MODEL	Model name	-	-
Manufacturer Name String Characteristic	N/A	DIS Manufacturer name string characteristic (Always enable)	-	-
	CONFIG_BT_DIS_MANUF	Manufacturer name	-	-
Firmware Revision String Characteristic	CONFIG_BT_DIS_FW_REV	Enable DIS Firmware Revision characteristic		
	CONFIG_BT_DIS_FW_REV_STR	Firmware revision	-	-
Hardware Revision String Characteristic	CONFIG_BT_DIS_HW_REV	Enable DIS Hardware Revision characteristic		
	CONFIG_BT_DIS_HW_REV_STR	Hardware revision	-	-
PnP ID Characteristic	CONFIG_BT_DIS_PNP	Enable PnP_ID characteristic		
	CONFIG_BT_DIS_PNP_PID	Product ID	-	-
	CONFIG_BT_DIS_PNP_VER	Product Version	-	-
	CONFIG_BT_DIS_PNP_VID	Vendor ID	-	-
	CONFIG_BT_DIS_PNP_VID_SRC	Vendor ID source	-	-
Serial Number String Characteristic	CONFIG_BT_DIS_SERIAL_NUMBER	Enable DIS Serial number characteristic		
	CONFIG_BT_DIS_SERIAL_NUMBER_STR	Serial Number	-	-
Software Revision String Characteristic	CONFIG_BT_DIS_SW_REV	Enable DIS Software Revision characteristic		
	CONFIG_BT_DIS_SW_REV_STR	Software revision	-	-

2.1.1.5 Heart Rate Service 源码路径：zephyr\subsys\bluetooth\services\hrs.c。

Service or Characteristics	Kconfig	Description	ROM	SRAM
Heart Rate Service	CONFIG_BT_HRS	Enable GATT Heart Rate service		
	CONFIG_BT_HRS_DEFAULT_PERM_RW	Read and write allowed		
	CONFIG_BT_HRS_DEFAULT_PERM_RW_AUTHEN	Require encryption and authentication for access		
	CONFIG_BT_HRS_DEFAULT_PERM_RW_ENCRYPT	Require encryption for access		
Measurement Interval Characteristic	N/A	HRS Measurement interval characteristic (Always enable)	-	-
Body Sensor Location Characteristic	N/A	HRS Body sensor location characteristic (Always enable)	-	-
Control Point Characteristic	N/A	HRS Control Point characteristic (Always enable)	-	-

APIs

- bt_hrs_notify

  需要包含头文件 `#include <bluetooth/services/hrs.h>`

2.1.1.6 Object Transfer Service

源码路径：`zephyr\subsys\bluetooth\services\ots\*`。

Service or Characteristics	Kconfig	Description	ROM	SRAM
Object Transfer Service	CONFIG_BT_OTS	Object Transfer Service (OTS) [EXPERIMENTAL]		
	CONFIG_BT_OTS_DIR_LIST_OBJ	Enables the Directory Listing Object		
	CONFIG_BT_OTS_DIR_LIST_OBJ_NAME	The object name of the Directory Listing Object		
	CONFIG_BT_OTS_L2CAP_CHAN_RX_MTU	Size of RX MTU for Object Transfer Channel		
	CONFIG_BT_OTS_L2CAP_CHAN_TX_MTU	Size of TX MTU for Object Transfer Channel		
	CONFIG_BT_OTS_MAX_INSTANCE	Maximum number of available OTS instances		
	CONFIG_BT_OTS_MAX_OBJ_CNT	Maximum number of objects that each service instance can store		
	CONFIG_BT_OTS_OACP_CREATE_SUPPORT	Support OACP Create Operation		
	CONFIG_BT_OTS_OACP_DELETE_SUPPORT	Support OACP Delete Operation		
	CONFIG_BT_OTS_OACP_PATCH_SUPPORT	Support patching of objects		
	CONFIG_BT_OTS_OACP_READ_SUPPORT	Support OACP Read Operation		
	CONFIG_BT_OTS_OACP_WRITE_SUPPORT	Support OACP Write Operation		
	CONFIG_BT_OTS_OBJ_MAX_NAME_LEN	Maximum object name length		
	CONFIG_BT_OTS_OBJ_NAME_WRITE_SUPPORT	Support object name write		
	CONFIG_BT_OTS_OLCP_GOTO_SUPPORT	Support OLCP Go To Operation		
	CONFIG_BT_OTS_SECONDARY_SVC	Register OTS as Secondary Service		
OTS Feature Characteristic	N/A	OTS Feature characteristic (Always enable)	-	-
OTS Object Name Characteristic	N/A	OTS Object name characteristic (Always enable)	-	-
OTS Object Type Characteristic	N/A	OTS Object type characteristic (Always enable)	-	-
OTS Object Size Characteristic	N/A	OTS Object size characteristic (Always enable)	-	-
OTS Object ID Characteristic	N/A	OTS Object ID characteristic (Always enable)	-	-
OTS Object Properties Characteristic	N/A	OTS Object properties characteristic (Always enable)	-	-
OTS Object Action Control Point Characteristic	N/A	OTS Object action control point characteristic (Always enable)	-	-
OTS Object List Control Point Characteristic	N/A	OTS Object list control point characteristic (Always enable)	-	-

APIs

- bt_ots_init
- bt_ots_obj_add
- bt_ots_obj_delete

- bt_ots_svc_decl_get
- bt_ots_free_instance_get
- bt_ots_obj_id_to_str

  需要包含头文件 #include <bluetooth/services/ots.h>

2.1.2 Vendor Service

2.2 Notify

参考 bt_gatt_indicate 参数计算。

```
int notify_send(const void *data, uint16_t len)
{
    return bt_gatt_notify(NULL, &cvs.attrs[1], data, len);
}
```

2.3 Indicate

```
static void indicate_cb(struct bt_conn *conn,
                                                struct bt_gatt_indicate_params *params, uint8_
↪t err)
{
        printk("Indication %s\n", err != 0U ? "fail" : "success");
}

static void indicate_destroy(struct bt_gatt_indicate_params *params)
{
        printk("Indication complete\n");
}

int indicate_send(const void *data, uint16_t len)
{
    static struct bt_gatt_attr *ind_attr;
    static struct bt_gatt_indicate_params ind_params;

    vnd_ind_attr = bt_gatt_find_by_uuid(svc.attrs, svc.attr_count,
                                                    &chr_uuid.uuid);

    ind_params.attr = ind_attr;
        ind_params.func = indicate_cb;
        ind_params.destroy = indicate_destroy;
        ind_params.data = data;
        ind_params.len  = len;

    return bt_gatt_indicate(NULL, &ind_params);
}
```

2.4 Attribute Index

我们在使用 bt_gatt_indicate，bt_gatt_notify 时，需要传入一个参数 attr，这个参数与 GATT Service 的定义有关。下面介绍一下这个参数如何填写。

Service 是多个 Attribute 的集合，为了方便定义，我们使用宏进行组织服务，每个宏包含 1 或 2 个 Attribute，如下：

宏	描述	数量
BT_GATT_PRIMARY_SERVICE	Primary Service Declaration Macro.	1
BT_GATT_SECONDARY_SERVICE	Secondary Service Declaration Macro.	1
BT_GATT_INCLUDE_SERVICE	Include Service Declaration Macro.	1
BT_GATT_CHARACTERISTIC	Characteristic and Value Declaration Macro.	2
BT_GATT_CCC	Client Characteristic Configuration Declaration Macro.	1
BT_GATT_CEP	Characteristic Extended Properties Declaration Macro.	1
BT_GATT_CUD	Characteristic User Format Descriptor Declaration Macro.	1
BT_GATT_CPF	Characteristic Presentation Format Descriptor Declaration Macro.	1
BT_GATT_DESCRIPTOR	Descriptor Declaration Macro.	1

如下是 Battery Service 的定义，源码位置：`\subsys\bluetooth\services\bas.c`

```
BT_GATT_SERVICE_DEFINE(bas,
        BT_GATT_PRIMARY_SERVICE(BT_UUID_BAS),
        BT_GATT_CHARACTERISTIC(BT_UUID_BAS_BATTERY_LEVEL,
                               BT_GATT_CHRC_READ | BT_GATT_CHRC_NOTIFY,
                               BT_GATT_PERM_READ, read_blvl, NULL,
                               &battery_level),
        BT_GATT_CCC(blvl_ccc_cfg_changed,
                    BT_GATT_PERM_READ | BT_GATT_PERM_WRITE),
);
```

其 Notify Characteristic Attribute index 为 1 (`BT_GATT_PRIMARY_SERVICE`)。

同样的，如下 `hog_svc`：

```
BT_GATT_SERVICE_DEFINE(hog_svc,
        BT_GATT_PRIMARY_SERVICE(BT_UUID_HIDS),
        BT_GATT_CHARACTERISTIC(BT_UUID_HIDS_INFO, BT_GATT_CHRC_READ,
                               BT_GATT_PERM_READ, read_info, NULL, &info),
        BT_GATT_CHARACTERISTIC(BT_UUID_HIDS_REPORT_MAP, BT_GATT_CHRC_READ,
                               BT_GATT_PERM_READ, read_report_map, NULL, NULL),
        BT_GATT_CHARACTERISTIC(BT_UUID_HIDS_REPORT,
                               BT_GATT_CHRC_READ | BT_GATT_CHRC_NOTIFY,
                               BT_GATT_PERM_READ_AUTHEN,
                               read_input_report, NULL, NULL),
        BT_GATT_CCC(input_ccc_changed,
                    BT_GATT_PERM_READ | BT_GATT_PERM_WRITE),
        BT_GATT_DESCRIPTOR(BT_UUID_HIDS_REPORT_REF, BT_GATT_PERM_READ,
                           read_report, NULL, &input),
        BT_GATT_CHARACTERISTIC(BT_UUID_HIDS_REPORT,
                               BT_GATT_CHRC_READ | BT_GATT_CHRC_NOTIFY,
                               BT_GATT_PERM_READ_AUTHEN,
                               read_input_report_consumer, NULL, NULL),
        BT_GATT_CCC(input_ccc_changed_consumer,
                    BT_GATT_PERM_READ | BT_GATT_PERM_WRITE),
        BT_GATT_DESCRIPTOR(BT_UUID_HIDS_REPORT_REF, BT_GATT_PERM_READ,
                           read_report_consumer, NULL, &input_consumer),
        BT_GATT_CHARACTERISTIC(BT_UUID_HIDS_CTRL_POINT,
                               BT_GATT_CHRC_WRITE_WITHOUT_RESP,
                               BT_GATT_PERM_WRITE,
                               NULL, write_ctrl_point, &ctrl_point),
);
```

- 第一个 Notify Characteristic Attribute index 为 5

  `BT_GATT_PRIMARY_SERVICE`+`BT_GATT_CHARACTERISTIC`+`BT_GATT_CHARACTERISTIC` = 1+2+2

- 第二个 Notify Characteristic Attribute index 为 9

  BT_GATT_PRIMARY_SERVICE+BT_GATT_CHARACTERISTIC+BT_GATT_CHARACTERISTIC + BT_GATT_CHARACTERISTIC + BT_GATT_CCC + BT_GATT_DESCRIPTOR = 1+2+2+2+1+1

4.8 BLE Mesh **开发指南**

4.8.1 1 Mesh Introduce

PAN1080 基于 zephyr，验证并开发了 Mesh 大部分 feature 和 model, 并兼容了 PAN1080 方案内的部分功能

1.1 **区域划分**

Mesh 可以选择带 Boot 程序的烧录方式，支持 OTA 功能，config 宏需要配置

```
CONFIG_IS_BOOTLOADER=y
CONFIG_USE_DT_CODE_PARTITION=n
```

也可以选择不带 Boot 程序的烧录方式，config 宏需要配置

```
CONFIG_IS_BOOTLOADER=n
CONFIG_USE_DT_CODE_PARTITION=y
```

当使能**带** Boot **程序的烧录方式**时，zephyr 区域划分如下

Chip Data 区：

- 4K 受保护的出厂 info 区；
- 存放不同项目定制的 OTA 区域划分，关键参数
  - OTA_header_addr , OTA_header_size
    - OTA header 文件存储位置和大小
  - OTA_image_addr ， image_max_size
    - OTA image 文件起始位置，image 最大大小
  - zephyr_data_start_addr,zephyr_data_max_size
    - Zephyr 系统存储数据区起始位置和大小

Bootloader 区：

- 28K 出厂烧录，通常不会进行更新的 boot 程序，包含搬运程序（最大 4k）及兼容后续开发上位机通信升级程序（最大 24K）

Main Program header 区：

- 大小 512B
- 存放备用信息，便于扩展 B 方案

Main Program 区：

- 大小根据项目不同 = image_max_size；
- bootloader 跳转的实际程序区

OTA temporary 区 header：

- 大小：512B
- 存放固件对应信息，搬运时使用 crc 及 size 确认固件完整性，搬运后标志相应 flag 标识结束

OTA temporary 区：

分为 OTA header 及 OTA image，合起来在固件传输时可以称为 firmware

- 临时存放升级程序区，大小 = image_max_size
- OTA image 内容同 Main Program；编译生成固件方式同 Main Program（不需要特殊映射编译地址）

Reserved Code 区

- 大小：zephyr_data_start_addr - (OTA_image_addr + OTA_image_max_size) ;
- 剩余空闲区域

Reserved Data 区

- 存放用户数据区，预留大小为剩余部分

Zephyr OS Data 区

- 存放系统数据区（CONFIG_FLASH 区），使用 zephyr config flash 方式存储项目中的信息（如入网信息）
- 用户可以选择升级后擦除/保留该数据区内容

1.2 Provisioning

Mesh provisioning 为 Mesh 最主要的组网功能，未入网设备可以通过以下 3 种方式进行入网

入网通过 Nrf Mesh App 时，可以选择 OOB 入网方式进行

广义入网通常还包括入网后的 Config 过程，狭义正式的入网通常只包括到 Provision data 分配完成

1.2.1 Key 说明

入网涉及到多种 key，在抓包分析时起到比较重要的作用，以下对其进行介绍

1.2.1.1 Session Key

狭义入网过程并不需要 key 进行解析，但在入网过程中，涉及到 ECDH 中交换 session key 的过程，用来完成 ECC 加解密完成入网，可以参考 `Mesh Spec` 进行简单了解

1.2.1.2 Network Key

在网络层加密认证消息。有了网络层的密钥就可以成为 Mesh 网络的一员，可以对数据进行加密、解密和转发。如果 Mesh 网络中有多个子网，每个子网都会用自己的 Network Key，不同子网的设备是没有办法进行数据转发的

1.2.1.3 Device Key

Device Key 是一种特殊的 App Key，每个节点的 Device Key 都是独一无二的，它用在配网阶段，用于实现数据的安全传输。只有 Provisioner 和入网的设备知道这个 Device Key

Foundation Model 需要用 dev key 加密进行传输

1.2.1.4 App Key　在 access 层加密认证消息。虽然有 Network Key 可以实现数据的加解密和转发，但是网络层不知道应用层的数据具体是什么。要解决这个问题，就需要用到 Application Key，简称 App Key。在一个 Mesh 网络里，可能有多个 App Key，每个 App Key 对应一组应用和相关的节点

1.2.1.5 Key 通过 Code 获取　Key 通过 code 开启相应的宏，可以打印出来供解析使用

1. netkey&devkey 通过入网是 prov_data 获取，通过开启

```
CONFIG_BT_DEBUG_LOG=y
CONFIG_BT_MESH_DEBUG=y
CONFIG_BT_MESH_DEBUG_PROV=y
```

在 subsys\bluetooth\mesh\prov_device.c 中打印

```
static void prov_data(const uint8_t *data)
{
        ...

        /* Log of netkey & devkey */
        uint8_t netkey[16];

        memcpy(netkey, &pdu, 16);
        BT_DBG("NetKey: %s", bt_hex(netkey, 16));
        BT_DBG("DevKey: %s", bt_hex(dev_key, 16));

        ...
}
```

2. appkey 在入网后 config 进行获取，通过开启 config 宏

```
CONFIG_BT_DEBUG_LOG=y
CONFIG_BT_MESH_DEBUG=y
CONFIG_BT_MESH_DEBUG_KEYS=y
```

在 app_keys.c 中打印

```
uint8_t bt_mesh_app_key_add(uint16_t app_idx, uint16_t net_idx,
                            const uint8_t key[16])
{
        struct app_key *app;

        BT_DBG("net_idx 0x%04x app_idx %04x val %s", net_idx, app_idx,
               bt_hex(key, 16));

    ...

}
```

1.2.2 OOB 方式入网说明　入网时，需要了解入网的验证方式，入网验证方式在 provisioner 发送 invite 后，由设备将支持的 oob 方式在 capabilities 内体现，由 provisioner 发送 Start 消息时继进行选择，有如下几种

oob 配置方式为通常在 mesh init prov 结构体时选择支持方式如下

```
static const struct bt_mesh_prov prov = {

        ...

        .output_size = 4,
        .output_actions = BT_MESH_DISPLAY_STRING,
        .output_string = prov_output_string,
```

(下页继续)

(续上页)

```
        ...
};
```

1.2.2.1 No OOB　PanMesh/Echo 入网时，start 内选择 No OOB，不需要额外进行交互即可进行入网

1.2.2.2 Static OOB　天猫精灵/小度音箱通过三元组生成的 Static OOB 在云端请求验证方式进行入网，因此三元组在入网流程中是必要的

1.2.2.3 Input OOB　由 provisioner show，由芯片端进行输入的方式进行入网 oob 交互验证

1.2.2.4 Output OOB　由芯片端 show，由 provisioner 进行输入的方式进行入网 oob 交互验证

1.2.3 pb_adv **入网**　pb_adv 在项目内通过使能相应宏

```
CONFIG_BT_MESH_PB_ADV=y
```

并且 code 内开启 pb_adv 入网方式, 通常 code 如下即可开启两种 bearer 的应用，而之上的 Config 决定最终生效的 bearer

```
bt_mesh_prov_enable(BT_MESH_PROV_ADV | BT_MESH_PROV_GATT);
```

1.2.3.1 **功能说明**　设备入网是指将一个未入网的设备加入到 Mesh 网络的过程，这整个过程是由 Provisioner 管理的。Provisioner 提供给未入网的设备一些入网数据（包括 Network Key，当前的 IV Index，元素的单播地址等）来使其成为一个入网的 Mesh 节点。Provisioner 主要是智能手机或者一些智能设备，我们这里介绍的 Provisioner 主要指的是天猫精灵。

（1）建立入网 bearer

要入网一个设备，Provisioner 和设备之间要建立入网的 bearer，Provisioner 主要通过设备的 Device UUID 和其他信息来对设备进行识别。

（2）建立共享的密钥

Provisioner 和设备使用 ECDH 协议建立一个共享的密钥。

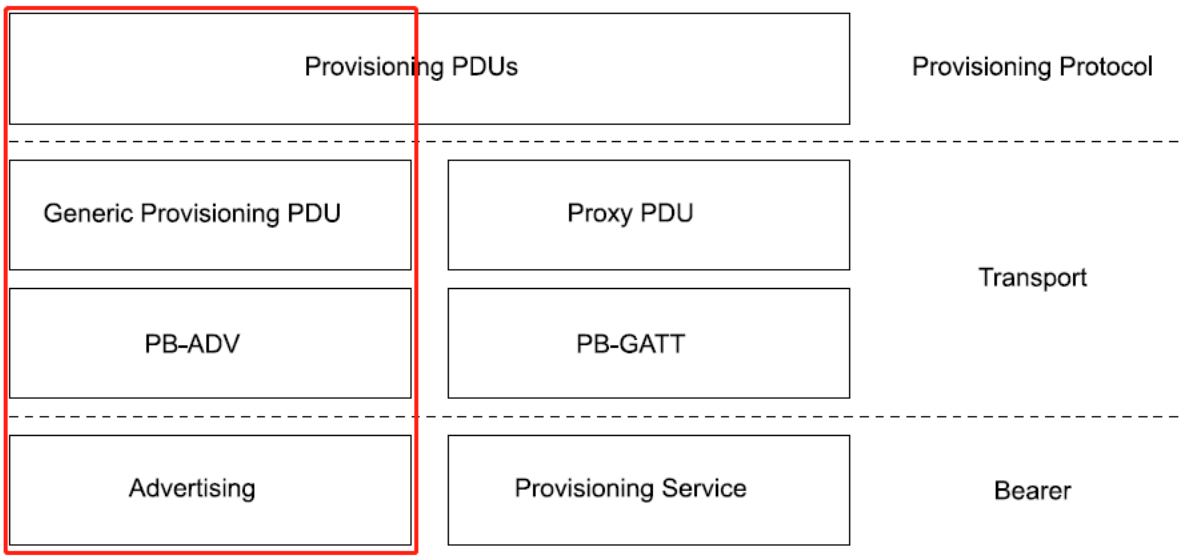
（3）认证设备

然后使用指定的 OOB 信息（设备的 public key 等等）认证设备。

（4）传输加密的数据

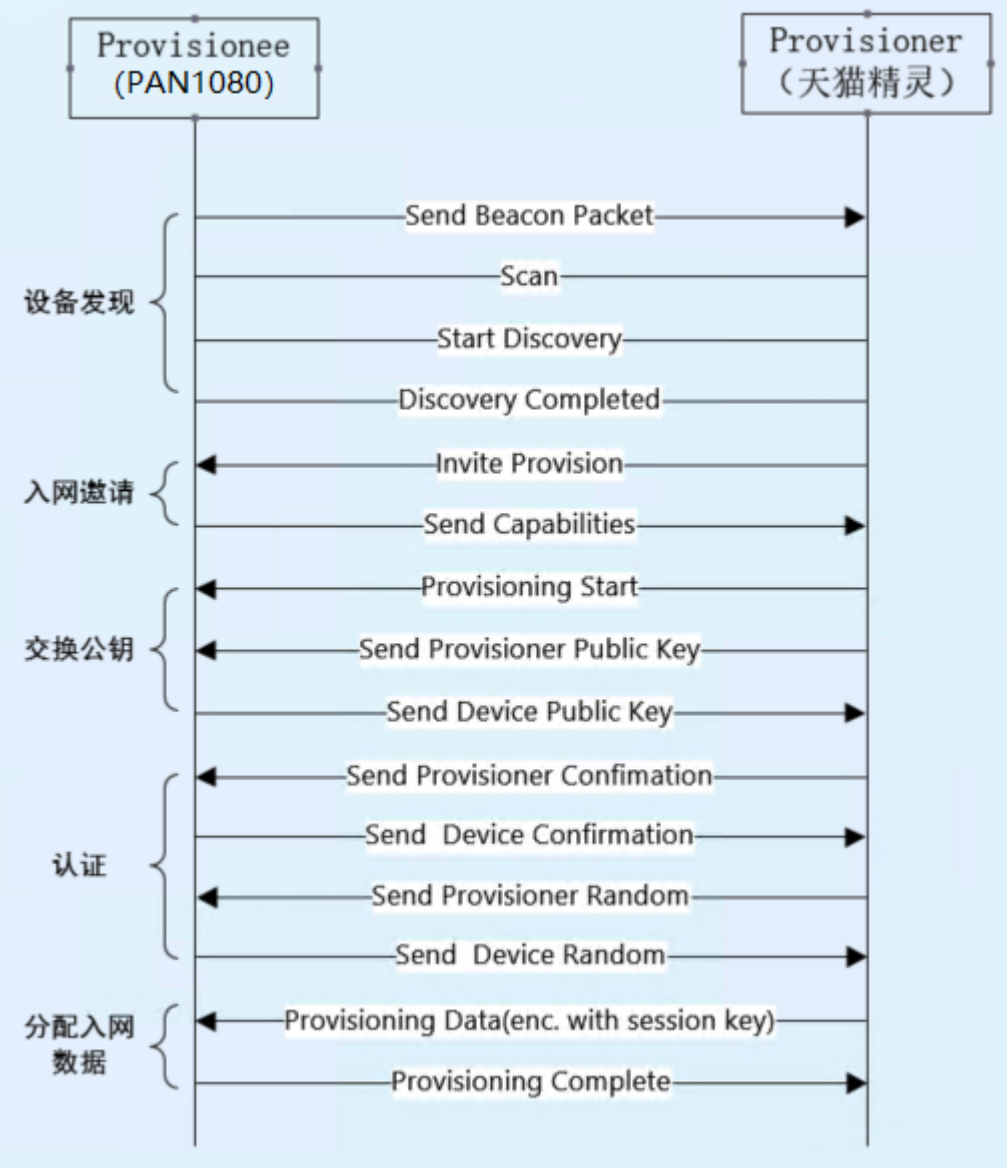
传输加密（使用共享的密钥）的数据。

如图所示，展示的是我们 Mesh 入网的协议栈，GAP 入网主要走的是红框标注的流程。



Provisioner 使用 PB-ADV，通过广播的方式向未入网的设备发送 Generic Provisioning PDU 对设备进行入网。Provisioner 一次可以对多个设备入网，但 Provisionee 一次只能被一个 Provisioner 入网。PB-ADV bearer 被用来传输 Generic Provisioning PDU，PB-ADV bearer MTU（最大传输单元）长度是 24 字节。

1.2.3.2 **入网流程** GAP 入网通常需要 5 个步骤来完成，分别是设备发现，入网邀请，交换公钥，认证和分配入网数据，关于 GAP 的详细入网流程可以参考下图。



（1）设备发现

设备（如 PAN1080）在未入网且没有处在入网流程时，应该向外界广播未入网的 Beacon 数据包。Provisioner（如天猫精灵）在准备好对未入网设备入网时，一直处于扫描状态，扫描窗口接近 100%，当扫描到未入网设备发的 Beacon 包之后，进行一些验证，完成设备发现操作。

（2）入网邀请

在建立了一个入网 bearer 之后，Provisioner 应该向未入网的设备发送 Provisioning Invite PDU，未入网的设备收到邀请后向 Provisioner 回复一个 Provisioning Capabilities PDU。

（3）交换公钥

在收到设备的入网能力之后，一旦 Provisioner 决定对设备进行入网，它应该发送一个 Provisioning Start PDU。紧接着 Provisioner 和设备之间交换公钥。

（4）认证

Provisioner 和设备接着发送 Provisioning Confirmation PDU 给对方，来确认到目前为止接收到的值。然后 Provisioner 和设备互相发送 Provisioning Random PDU 来使对方的确认生效。

（5）分配入网数据

一旦设备被认证了，Provisioner 和设备应该使用密钥加密入网的数据。Provisioner 然后向设备发送 Provisioning Data PDU（包含加密和认证的入网数据）给设备。发送完且设备接收配置完成之后，入网

操作完成。

1.2.3.3 **多音箱支持同时入网** Panchip 为适配同一代码实现不同音箱根据不同三元组开发了兼容入网的功能

当客户有需求实现一个设备同时可以通过天猫精灵和小度入网时，使能兼容入网功能，

```
CONFIG_BT_MESH_MULTIPLE_BEACON=y
```

交替广播天猫精灵和小度广播使天猫精灵和小度都可以发现设备并入网

需要注意入网过程中会通过 link ID 选择不同的 static OOB（三元组生成）

```
static void link_open(struct prov_rx *rx, struct net_buf_simple *buf)
{
        ...
#if defined(CONFIG_BT_MESH_MULTIPLE_BEACON)

        if (!memcmp(buf->data, bt_mesh_prov_get()->uuid, 16)) {
                BT_DBG("Bearer open message for tmall");
                speaker_dev_type = SPEAKER_DEV_TMALL;
        } else if(!memcmp(buf->data, bt_mesh_prov_get()->uuid2, 16)) {
                BT_DBG("Bearer open message for xiaodu");
                speaker_dev_type = SPEAKER_DEV_XIAODU;
        } else {
                BT_DBG("Bearer open message not for us");
                return;
        }
#else
        if (memcmp(buf->data, bt_mesh_prov_get()->uuid, 16)) {
                BT_DBG("Bearer open message not for us");
                return;
        }
#endif
        ...
}
```

1.2.3.4 pb_remote_adv **入网** pb remote model server 对 unprovisionee node 进行入网时，实际上是通过 pb adv 进行最后的入网消息通信，对于 unprovisionee 来说，支持 pb adv 入网的功能即可验证 pb remote 的功能

具体请参考 Remote Model 的介绍

1.2.4 pb_gatt **入网** pb_gatt 在项目内通过使能相应宏

```
CONFIG_BT_MESH_PB_GATT=y
```

并且 code 内开启 pb_gatt 入网方式, 通常 code 如下即可开启两种 bearer 的应用，而之上的 Config 决定最终生效的 bearer

```
bt_mesh_prov_enable(BT_MESH_PROV_ADV | BT_MESH_PROV_GATT);
```

pb-gatt 入网主要应用与手机入网及 Echo 入网，较 pb adv 入网，由于 pb gatt 是基于连接，入网会更稳定更快速

pb gatt 入网连接建立需要参考 Mesh Profile，Mesh GATT Provisioning Services 介绍

连接建立后消息 network 层消息同 pb adv 一致

入网后通常通过 Network ID/Node ID 进行重连 config

入网后 config 阶段有些情况下需要 proxy protocol 内的 configuration 对 proxy 进行细节 filtrting 配置（如 echo，每次重新连接 gatt 时需要增加白名单过滤）

1.2.5 **开发说明** 以兼容入网为例，ble_mesh.c 中可以追溯 prov 结构体的 APIs

```
static const struct bt_mesh_prov prov = {
        .uuid = dev_uuid_tmall,
        .output_size = 4,
        .output_actions = BT_MESH_DISPLAY_STRING,
        .static_val = m_auth_data_tmall,
        .static_val_len = 16,
        .complete = prov_complete,
        .reset = prov_reset,
        .output_string = prov_output_string,
#if defined(CONFIG_BT_MESH_MULTIPLE_BEACON)
        .uuid2 = dev_uuid_xiaodu,
        .static_val2 = m_auth_data_xiaodu,
#endif
};
```

pb bearer 使能：

```
bt_mesh_prov_enable(BT_MESH_PROV_ADV | BT_MESH_PROV_GATT);
```

1.3 Features

Mesh Spec 支持 4 种大的 Feature，分别为 proxy，relay，friend，low power

1.3.1 Proxy

1.3.1.1 功能说明

1.3.1.1.1 **广播行为** 入网以后，通过广播 gatt 广播包，内容为 node id 广播包，Provisioner 可以通过扫描进行重连配置

配置之后，node id 广播包存在周期 60s 后，切换为 net id 广播包

此时入网设备（如 PanMesh）可以通过扫描 netid 广播包对 proxy node 进行重连

1.3.1.2 参数

1.3.1.2.1 **使能** Proxy 为 mesh 一大特色，通过使能节点的 proxy 功能

```
CONFIG_BT_MESH_GATT_PROXY=y
```

gatt client 可以连接 gatt server，对整个 mesh 网络进行控制

1.3.1.2.2 **配置连接参数** 通常，节点作为 proxy 被连接时，遵守 proxy client 的连接参数

某些情况下，需要 proxy server 端主动更新连接参数以适应部分场景

作为 gatt 连接 proxy 节点时，需要代码内进行主动更新连接参数，code 参考如下，此段逻辑可以完成 proxy 连接时，更新连接参数 gatt interval 到 100ms，并在 connected 与 disconneted 时进行 log 打印。

```
static void connected(struct bt_conn *conn, uint8_t err)
{
        /* gatt interval adjust to 100 ms to improve scan */
        const struct bt_le_conn_param *param = BT_LE_CONN_PARAM(80, 80, 0, 500);

        bt_conn_le_param_update(conn, param);
```

(下页继续)

(续上页)

```
        if (err) {
                printk("Connection failed (err 0x%02x)\n", err);
        } else {
                printk("Connected\n");
        }
}

static void disconnected(struct bt_conn *conn, uint8_t reason)
{
        printk("Disconnected (reason 0x%02x)\n", reason);
}

BT_CONN_CB_DEFINE(conn_callbacks) = {
        .connected = connected,
        .disconnected = disconnected,
};
```

1.3.1.2.3 Proxy 配置参数

```
if BT_CONN

# Virtual option enabled whenever any Proxy protocol is needed
config BT_MESH_PROXY
        bool

config BT_MESH_GATT
        bool

config BT_MESH_GATT_SERVER
        bool
        select BT_MESH_GATT
        select BT_GATT_DYNAMIC_DB

config BT_MESH_PB_GATT
        bool "Provisioning support using GATT (PB-GATT)"
        select BT_MESH_GATT_SERVER
        select BT_MESH_PROV
        help
          Enable this option to allow the device to be provisioned over
          GATT.

config BT_MESH_GATT_PROXY
        bool "GATT Proxy Service support"
        select BT_MESH_GATT_SERVER
        select BT_MESH_PROXY
        help
          This option enables support for the Mesh GATT Proxy Service,
          i.e. the ability to act as a proxy between a Mesh GATT Client
          and a Mesh network.

config BT_MESH_GATT_PROXY_ENABLED
        bool "GATT Proxy enabled"
        depends on BT_MESH_GATT_PROXY
        default y
        help
          Controls whether the GATT Proxy feature is enabled by default.
          Can be changed through runtime configuration.

config BT_MESH_NODE_ID_TIMEOUT
        int "Node Identity advertising timeout"
```

(下页继续)

(续上页)

```
        depends on BT_MESH_GATT_PROXY
        range 1 60
        default 60
        help
          This option determines for how long the local node advertises
          using Node Identity. The given value is in seconds. The
          specification limits this to 60 seconds, and implies that to
          be the appropriate value as well, so just leaving this as the
          default is the safest option.

config BT_MESH_PROXY_USE_DEVICE_NAME
        bool "Include Bluetooth device name in scan response"
        depends on BT_MESH_GATT_PROXY
        help
          This option includes GAP device name in scan response when
          the GATT Proxy feature is enabled.

config BT_MESH_PROXY_FILTER_SIZE
        int "Maximum number of filter entries per Proxy Client"
        default 3 if BT_MESH_GATT_PROXY
        default 1
        range 1 32767
        depends on BT_MESH_GATT_SERVER
        help
          This option specifies how many Proxy Filter entries the local
          node supports.

endif # BT_CONN
```

1.3.2 Relay

1.3.2.1 **功能说明**　Relay 为 network 层的一种消息转发机制

1.3.2.2 **参数**

1.3.2.2.1 **使能**　通过使能 sample 内 relay 功能使能

```
CONFIG_BT_MESH_RELAY=y
```

1.3.2.2.2 **配置参数**

```
menuconfig BT_MESH_RELAY
        bool "Relay support"
        help
          Support for acting as a Mesh Relay Node.

if BT_MESH_RELAY

config BT_MESH_RELAY_ENABLED
        bool "Relay enabled"
        default y
        help
          Controls whether the Mesh Relay feature is enabled by default. Can be
          changed through runtime configuration.

config BT_MESH_RELAY_RETRANSMIT_COUNT
```

(下页继续)

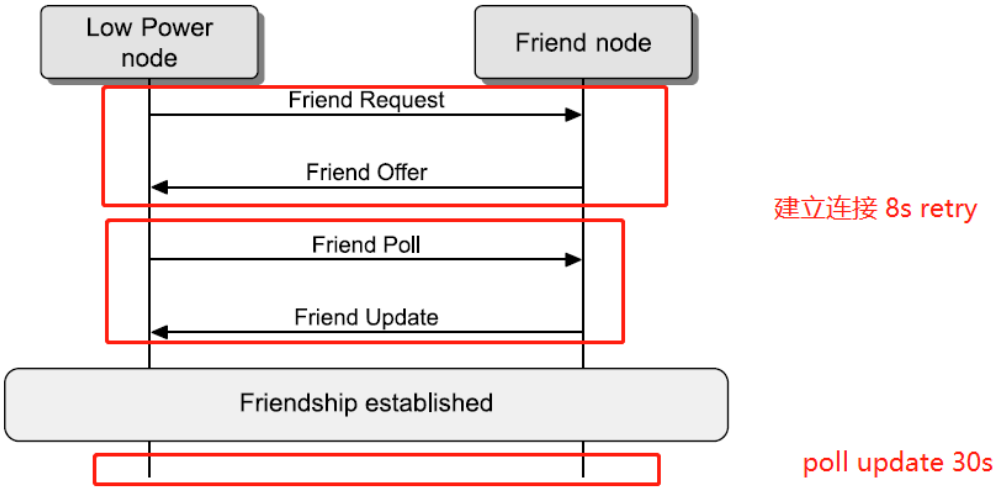
(续上页)

```
        int "Relay Retransmit Count"
        default 2
        range 0 7
        help
          Controls the initial number of retransmissions of relayed messages, in
          addition to the first transmission. Can be changed through runtime
          configuration.

config BT_MESH_RELAY_RETRANSMIT_INTERVAL
        int "Relay Retransmit Interval"
        default 20
        range 10 330
        help
          Controls the initial interval between retransmissions of relayed
          messages, in milliseconds. Can be changed through runtime
          configuration.

endif
```

1.3.3 Friend

1.3.3.1 功能说明 Mesh friend 功能主要是为了低功耗设备准备的一个解决方案，这让那些低功耗的设备一方面保持低电流消耗的睡眠模式，另一方面又能收到和自己相关的 Mesh 数据包。为了完成上述的目的，SIG Mesh 规定了 2 种角色分别是 Low Power node 和 Friend node。

Zephyr Friend LPN 基础 Feature 分为两部分



第一部分建立连接

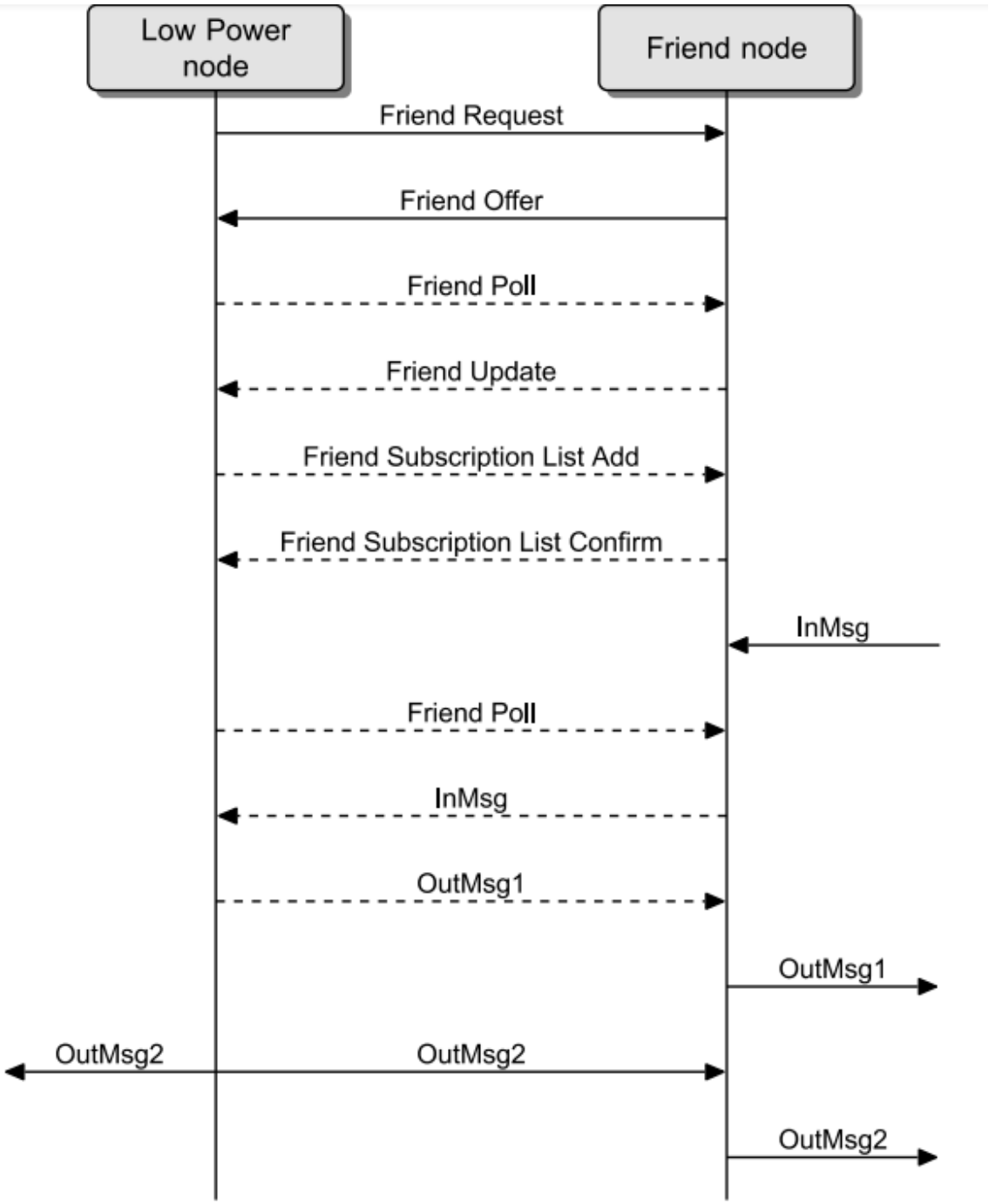
1）LPN 发送 request，广播范围内寻找 Friend，

2）收到消息的 Friend 会在 delay 后开始发送 friend offer

3）LPN 收到 offer 会进行 friend poll，等待 friend update

4）Friend 收到 Poll 消息后回复 Friend update，确认建立连接

5）LPN 收到消息，确认建立连接，进入第二部分



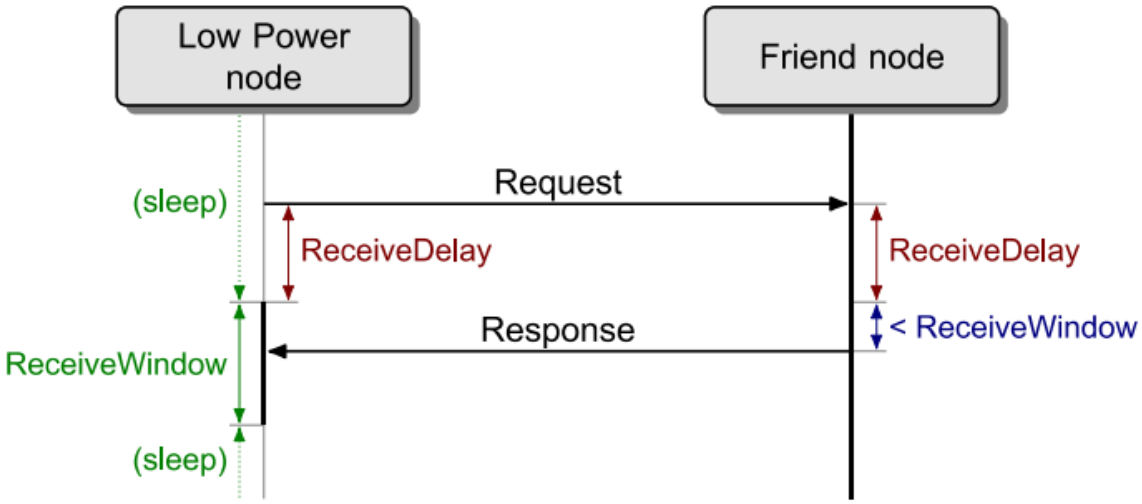
第二部分 Friend 保持

1）LPN 周期 30s，发送 Friend Poll，并定时开窗等待 Friend Update

2）Friend 收到 Friend Poll，check friend 是否有存储 LPN 的消息，若有：通过 Friend update 进行传递，若无：通过 Friend Update 进行保持

3）重复以上过程

1.3.3.2 **参数** Friend 关键参数如下图所示，receive window 和 receive delay 对 friend 的建立和维持起到关键作用



1.3.3.2.1 **使能** Friend 为友好节点，可以通过消息于 LPN 建立 Friend ship，周期性为 LPN 进行消息传递，通过使能 sample 内 friend 功能使能

```
CONFIG_BT_MESH_FRIEND=y
```

1.3.3.2.2 **配置参数**

```
config BT_MESH_FRIEND
        bool "Support for acting as a Friend Node"
        help
          Enable this option to be able to act as a Friend Node.

if BT_MESH_FRIEND

config BT_MESH_FRIEND_ENABLED
        bool "Friend feature enabled"
        default y
        help
          Controls whether the Friend feature is enabled by default. Can be
          changed through runtime configuration.

config BT_MESH_FRIEND_RECV_WIN
        int "Friend Receive Window"
        range 1 255
        default 255
        help
          Receive Window in milliseconds supported by the Friend node.

config BT_MESH_FRIEND_QUEUE_SIZE
        int "Minimum number of buffers supported per Friend Queue"
        range 2 65536
        default 16
        help
          Minimum number of buffers available to be stored for each
          local Friend Queue.

config BT_MESH_FRIEND_SUB_LIST_SIZE
        int "Friend Subscription List Size"
        range 0 1023
        default 3
        help
```

(下页继续)

(续上页)

```
          Size of the Subscription List that can be supported by a
          Friend node for a Low Power node.

config BT_MESH_FRIEND_LPN_COUNT
        int "Number of supported LPN nodes"
        range 1 1000
        default 2
        help
          Number of Low Power Nodes the Friend can have a Friendship
          with simultaneously.

config BT_MESH_FRIEND_SEG_RX
        int "Number of incomplete segment lists per LPN"
        range 1 1000
        default 1
        help
          Number of incomplete segment lists that we track for each LPN
          that we are Friends for. In other words, this determines how
          many elements we can simultaneously be receiving segmented
          messages from when the messages are going into the Friend queue.

endif # BT_MESH_FRIEND
```

1.3.4 LPN

1.3.4.1 **功能说明**

Low Power Node 为低功耗节点，与 Friend 建立 Friendship，周期性唤醒获取消息更新

1.3.4.2 **参数**

1.3.4.2.1 **使能**

通过使能 sample 内 low power 功能使能

```
CONFIG_BT_MESH_LOW_POWER=y
```

1.3.4.2.2 **配置**

```
config BT_MESH_LOW_POWER
        bool "Support for Low Power features"
        help
          Enable this option to be able to act as a Low Power Node.

if BT_MESH_LOW_POWER

config BT_MESH_LPN_ESTABLISHMENT
        bool "Perform Friendship establishment using low power"
        default y
        help
          Perform the Friendship establishment using low power, with
          the help of a reduced scan duty cycle. The downside of this
          is that the node may miss out on messages intended for it
          until it has successfully set up Friendship with a Friend
          node.

config BT_MESH_LPN_AUTO
        bool "Automatically start looking for Friend nodes once provisioned"
        default y
```

(下页继续)

(续上页)

```
        help
          Automatically enable LPN functionality once provisioned and start
          looking for Friend nodes. If this option is disabled LPN mode
          needs to be manually enabled by calling bt_mesh_lpn_set(true).

config BT_MESH_LPN_AUTO_TIMEOUT
        int "Time from last received message before going to LPN mode"
        default 15
        range 0 3600
        depends on BT_MESH_LPN_AUTO
        help
          Time in seconds from the last received message, that the node
          will wait before starting to look for Friend nodes.

config BT_MESH_LPN_RETRY_TIMEOUT
        int "Retry timeout for Friend requests"
        default 8
        range 1 3600
        help
          Time in seconds between Friend Requests, if a previous Friend
          Request did not receive any acceptable Friend Offers.

config BT_MESH_LPN_RSSI_FACTOR
        int "RSSIFactor, used in the Friend Offer Delay calculation"
        range 0 3
        default 0
        help
          The contribution of the RSSI measured by the Friend node used
          in Friend Offer Delay calculations. 0 = 1, 1 = 1.5, 2 = 2, 3 = 2.5.

config BT_MESH_LPN_RECV_WIN_FACTOR
        int "ReceiveWindowFactor, used in the Friend Offer Delay calculation"
        range 0 3
        default 0
        help
          The contribution of the supported Receive Window used in
          Friend Offer Delay calculations. 0 = 1, 1 = 1.5, 2 = 2, 3 = 2.5.

config BT_MESH_LPN_MIN_QUEUE_SIZE
        int "Minimum size of acceptable friend queue (MinQueueSizeLog)"
        range 1 7
        default 1
        help
          The MinQueueSizeLog field is defined as log_2(N), where N is
          the minimum number of maximum size Lower Transport PDUs that
          the Friend node can store in its Friend Queue. As an example,
          MinQueueSizeLog value 1 gives N = 2, and value 7 gives N = 128.

config BT_MESH_LPN_RECV_DELAY
        int "Receive delay requested by the local node"
        range 10 255
        default 100
        help
          The ReceiveDelay is the time between the Low Power node
          sending a request and listening for a response. This delay
          allows the Friend node time to prepare the response. The value
          is in units of milliseconds.

config BT_MESH_LPN_POLL_TIMEOUT
        int "The value of the PollTimeout timer"
        range 10 244735
```

(下页继续)

(续上页)

```
        default 300
        help
          PollTimeout timer is used to measure time between two
          consecutive requests sent by the Low Power node. If no
          requests are received by the Friend node before the
          PollTimeout timer expires, then the friendship is considered
          terminated. The value is in units of 100 milliseconds, so e.g.
          a value of 300 means 30 seconds.

config BT_MESH_LPN_INIT_POLL_TIMEOUT
        int "The starting value of the PollTimeout timer"
        range 10 BT_MESH_LPN_POLL_TIMEOUT
        default BT_MESH_LPN_POLL_TIMEOUT
        help
          The initial value of the PollTimeout timer when Friendship
          gets established for the first time. After this the timeout
          will gradually grow toward the actual PollTimeout, doubling
          in value for each iteration. The value is in units of 100
          milliseconds, so e.g. a value of 300 means 30 seconds.

config BT_MESH_LPN_SCAN_LATENCY
        int "Latency for enabling scanning"
        range 0 50
        default 10
        help
          Latency in milliseconds that it takes to enable scanning. This
          is in practice how much time in advance before the Receive Window
          that scanning is requested to be enabled.

config BT_MESH_LPN_GROUPS
        int "Number of groups the LPN can subscribe to"
        range 0 16384
        default 8
        help
          Maximum number of groups that the LPN can subscribe to.

config BT_MESH_LPN_SUB_ALL_NODES_ADDR
        bool "Automatically subscribe all nodes address"
        help
          Automatically subscribe all nodes address when friendship
          established.

endif # BT_MESH_LOW_POWER
```

1.4 Models

1.4.1 Light Mdels

Light Models 为 Panchip 为多种灯控整理的控制 models，都是基于 SIG Models 的实现

1.4.2 Sig OTA Model

基于 Sig Mesh V1.1，Mesh OTA 通过

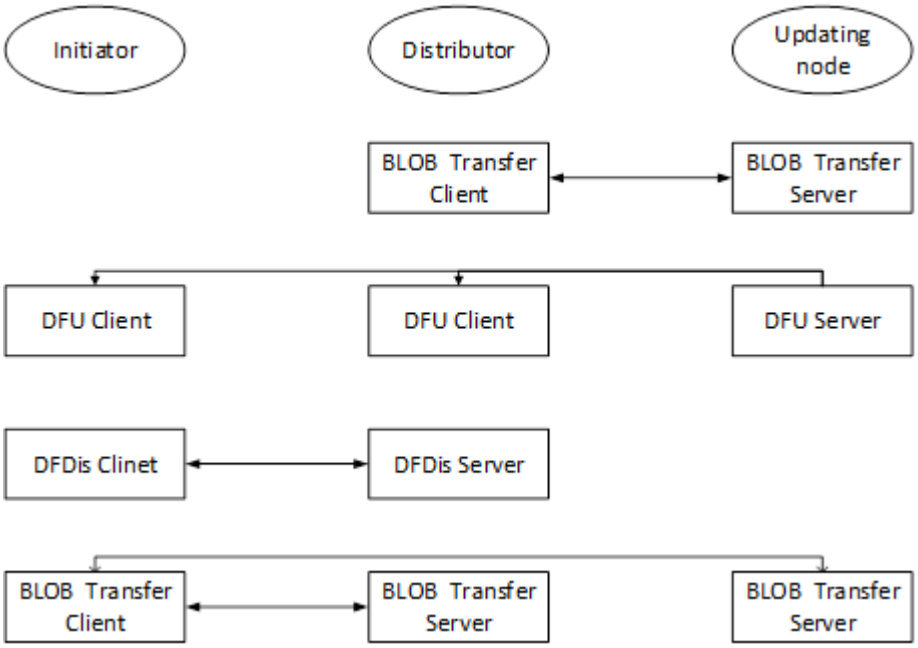
（1）BLOB transfer Model

（2）Device Firmware Update Model

（2）Distribute Model

实现升级数据的传送和整体升级流程的控制，从而达到为某些特定节点实现无线升级的功能。

Mesh OTA 身份简介如下图，具体细节请参考 SIG Mesh 官方文档。

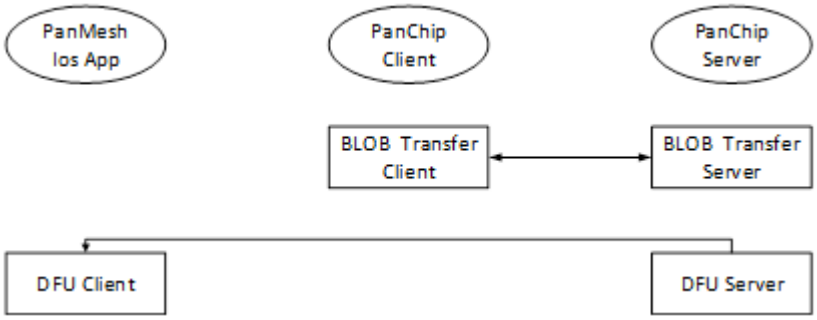


1.4.2.1 Panchip Mesh OTA **介绍** 基于如上考虑，Panchip Mesh 需要实现 Updating Node 身份，如下图 PanChip Server 所示，对应的 Model 为 BLOB Transfer Server 及 DFU Server。

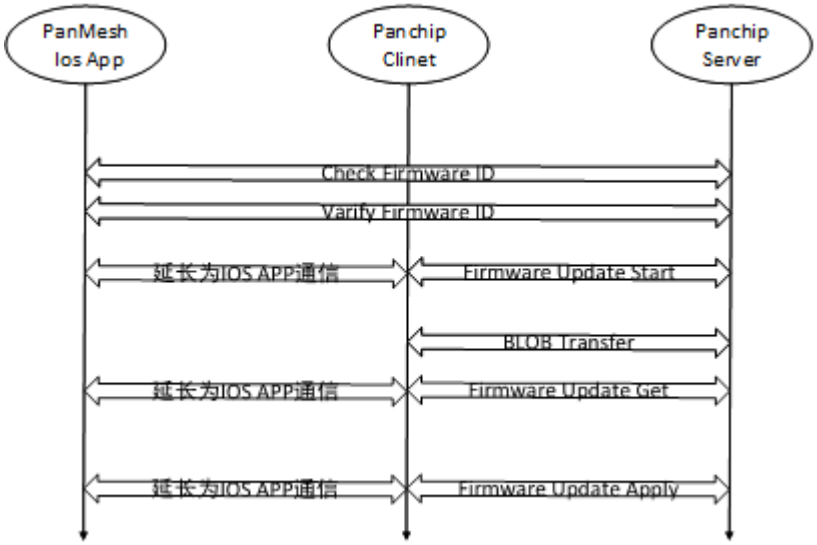
Distributor 为固件存储端及分发控制端，一般由对应厂商实现（如亚马逊），为方便调试，Panchip Client 可以简要实现其 BLOB Transfer Clinet 所实现的数据分发过程。因而固件需要提前准备好在 PanChip Clinet 端，根据 PAN1080 Mesh 区域划分规则，目前策略为将 80k-160k 区域烧录待升级固件。

Initiator 为升级控制端，一般由对应厂商实现（如亚马逊），为方便调试，Panchip 芯片可以简要实现其 DFU Client 对应的控制流程（检查升级状态，固件版本，流程开始、结束，新固件应用），PanMesh Ios App 需要从 PanMesh 下载链接进行下载。

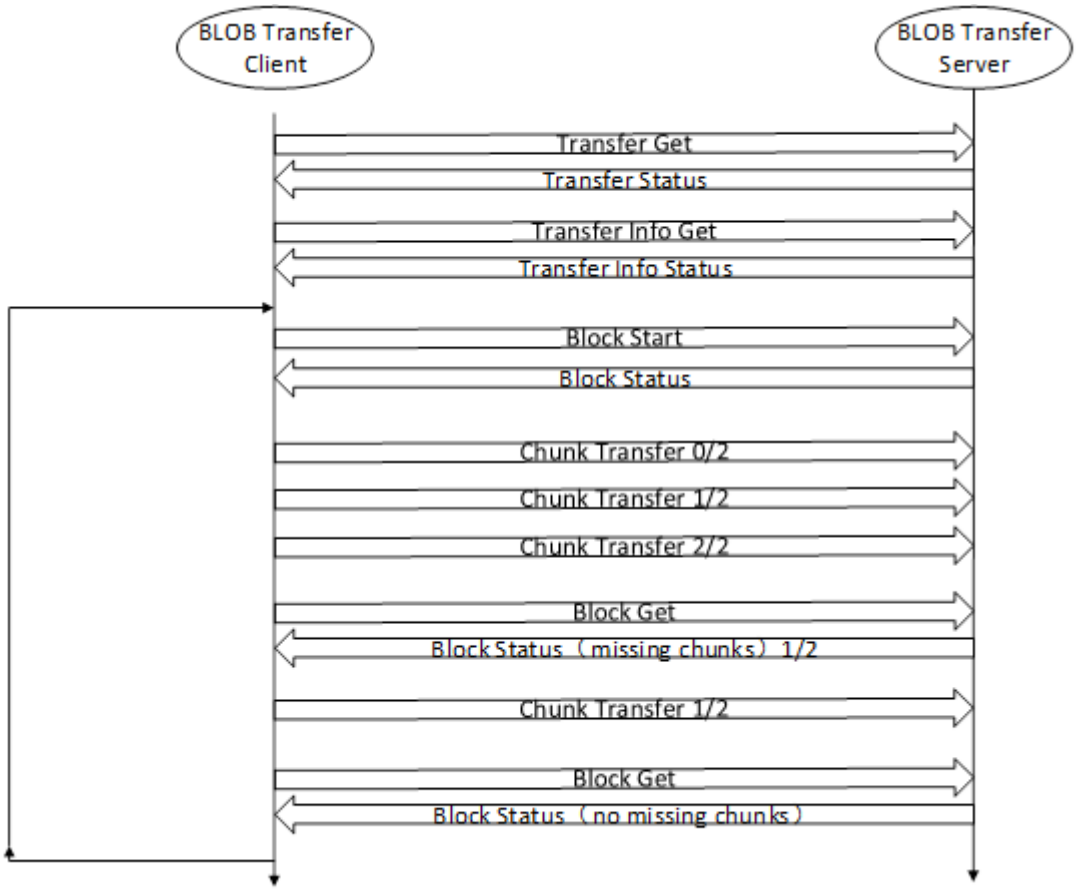
综上考虑，Panchip 提供完整的 Updating Node 角色实现及可供调试的对端身份如下图：



DFU 实现消息概述如下图：



1.4.2.2 BLOB Tranfer **流程介绍** BLOB Transfer Model 为用于数据传输的 Model，具体流程如下，实现细节参考 SIG 官方文档。



1.4.2.3 GATT SIG OTA 操作流程

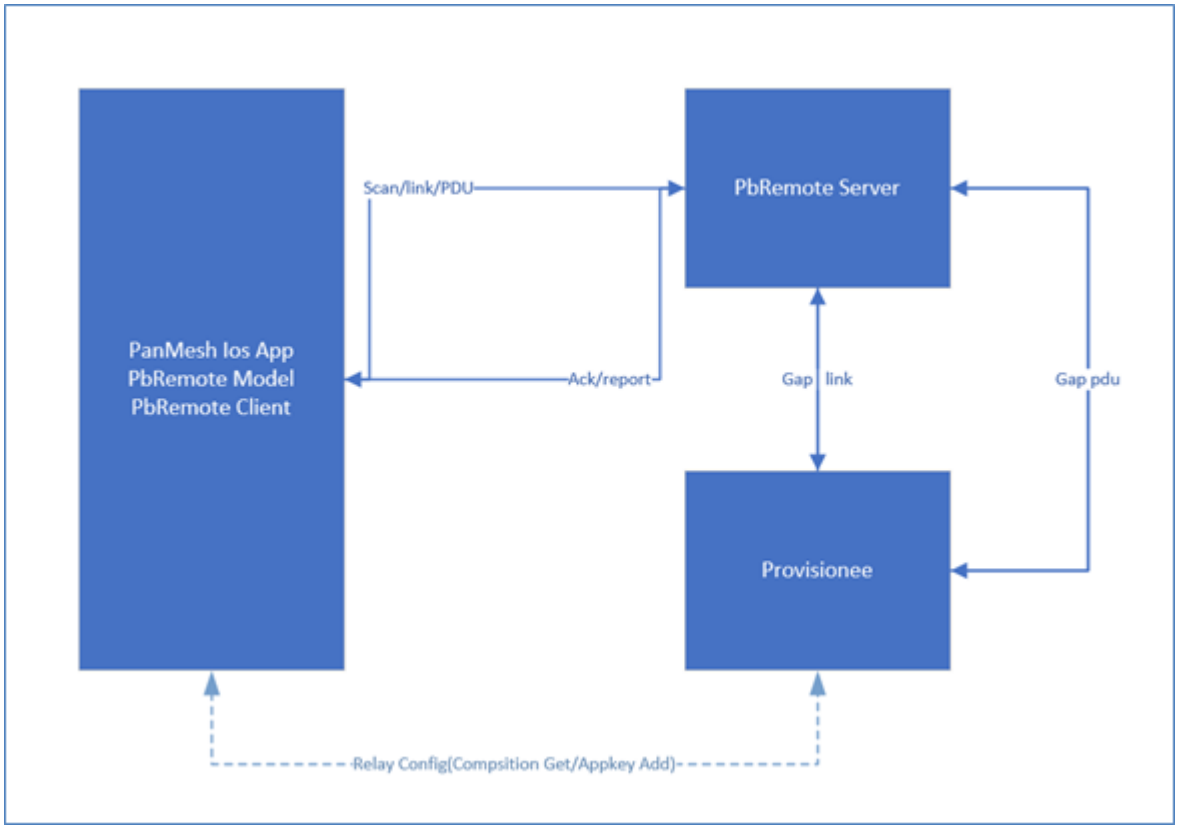
1. 在工程目录下，将 `ota.py` 拷贝至编译路径下，并运行，会由 `zephyer.bin` 生成 `zephyer_ota.bin`，参考 OTA Firmware 章节描述，生成了 header
2. 将 `zephyer_ota.bin` 拷贝至 IOS 手机端
3. PanMesh 待带升级节点入网，并绑定 appkey 到 SIG OTA Model

4. 进入 `Vendor Message` 界面，点击 `choose` 选择待升级固件
5. 点击 `OTA start` 开始 GATT SIG OTA 升级（当前 gatt 为分段包升级，chunk 64B，Block Size 512B）
6. 升级完成

1.4.3 PB Remote Model

1.4.3.1 PB Remote Model 功能 PbRemote Model 为 sig model 的一种，目的为实现通过 PbRemote Model 实现远程对 provisionee 进行入网。

目前手机端开发了简易的 PbRemote Client，与远端芯片端的 PbRemote Server 建立 Gatt 连接后，实现 scan/link/pdu 通信，PbRemote Server 收到 scan 消息进行 report 并开始进行扫描，扫描后通过 scan report 消息上报扫描到待入网设备 UUID，手机端通过选择 UUID 进行建立连接，与指定设备进行连接后，手机端开始分发入网 PDU 执行入网，执行入网后，实现配置消息从而在手机端显示入网设备。



测试环境：由上图，需要准备

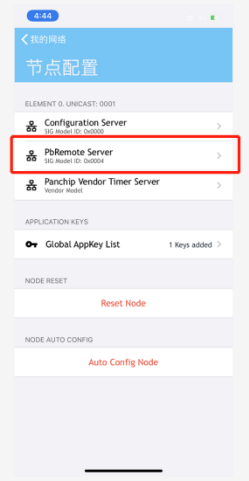
PanMesh Ios App：角色：PbRemote Client 以及 Provisioner

Chip#1：角色：PbRemote Server。烧录 PbRemote Server 代码完成与 PbRemote Client 通信并处于入网消息转发的角色。

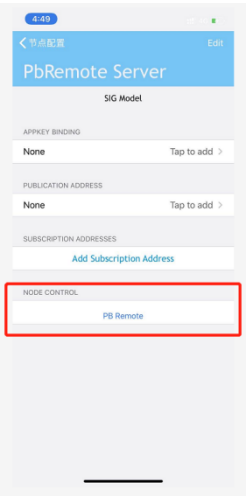
Chip#2：角色：Provisionee。待入网设备，烧录普通 mesh 待入网节点代码，支持 Gap 即可。

1.4.3.2 PB Remote 手机端操作流程

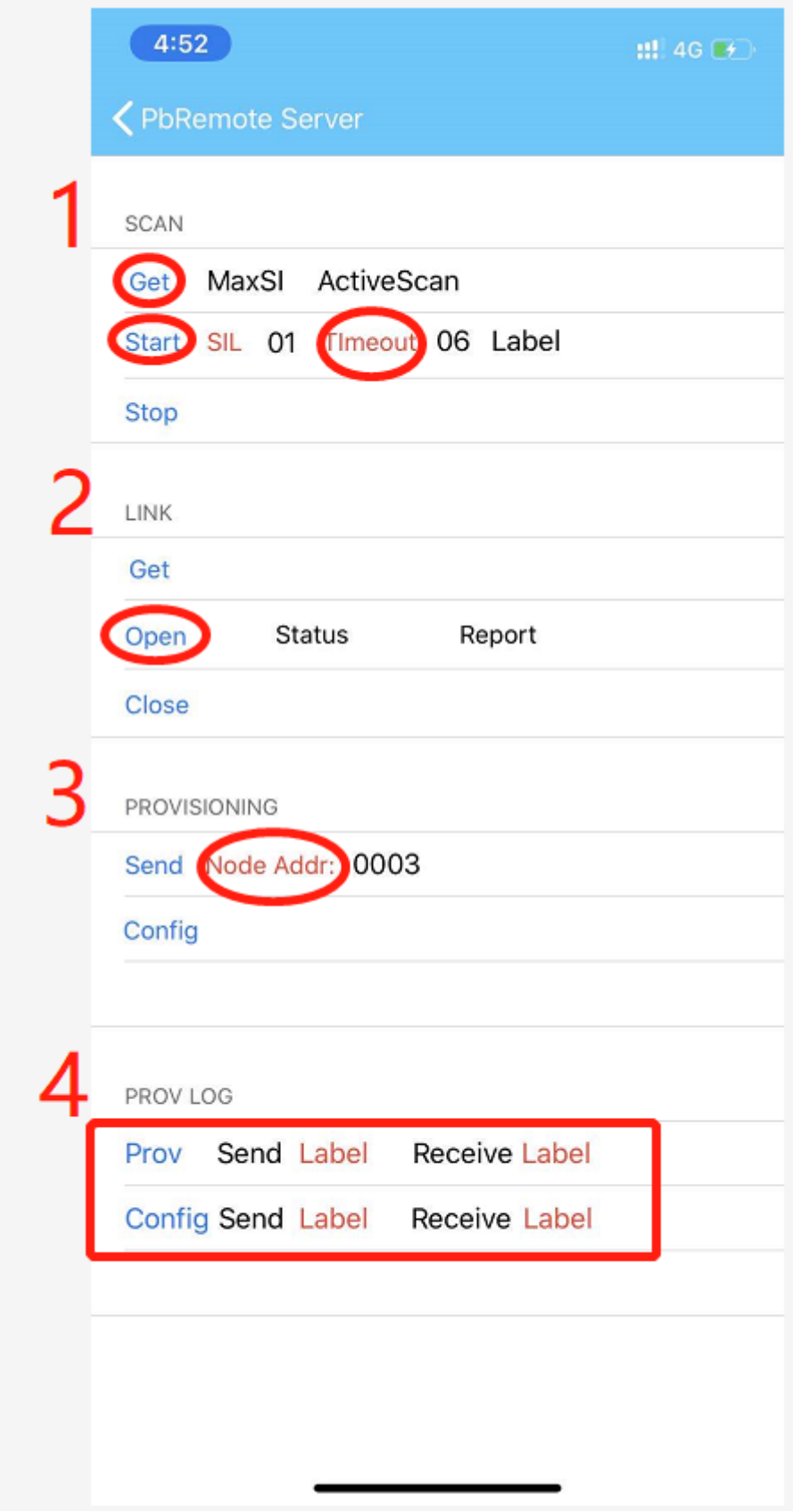
1. 对包含 PbRemote Server 芯片入网设备后，界面显示 PbRemote Server（Model ID 0x0004）（入网的时候可以不绑定 Appkey，流程待优化）



1. 从 Node Control 界面进入 Pb Remote 界面



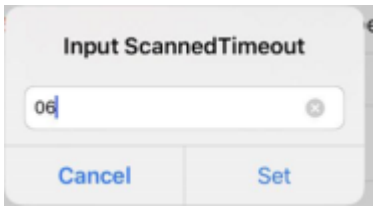
1. 进入 Server 消息操作界面，消息包含 Scan/Link/Provisioning PDU 以及入网配置 log 显示。



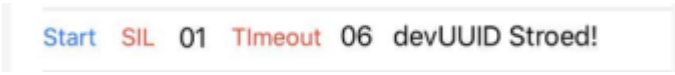
（1）点击 Scan 部分 Get 按键，获取 Server 的 MaxSI（最大扫描设备个数）及 ActiveScan（活跃扫描），获取结果如下。

显示支持最多扫描 4 个设备并且不支持 Active Scan

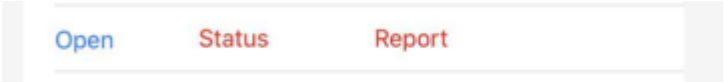
（2）点击 SIL 配置扫描待入网后上报节点 UUID 的最大个数（目前单个上报，此处默认 01），以及扫描时间（默认 06 及扫描六秒）



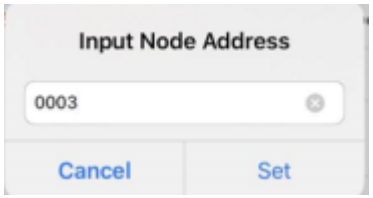
配置完成后，点击 Start 按键，Remote Server 收到消息后开始扫描，时间结束进行上报，上报后在后面 Label 处显示 devUUID Stored! 如图。



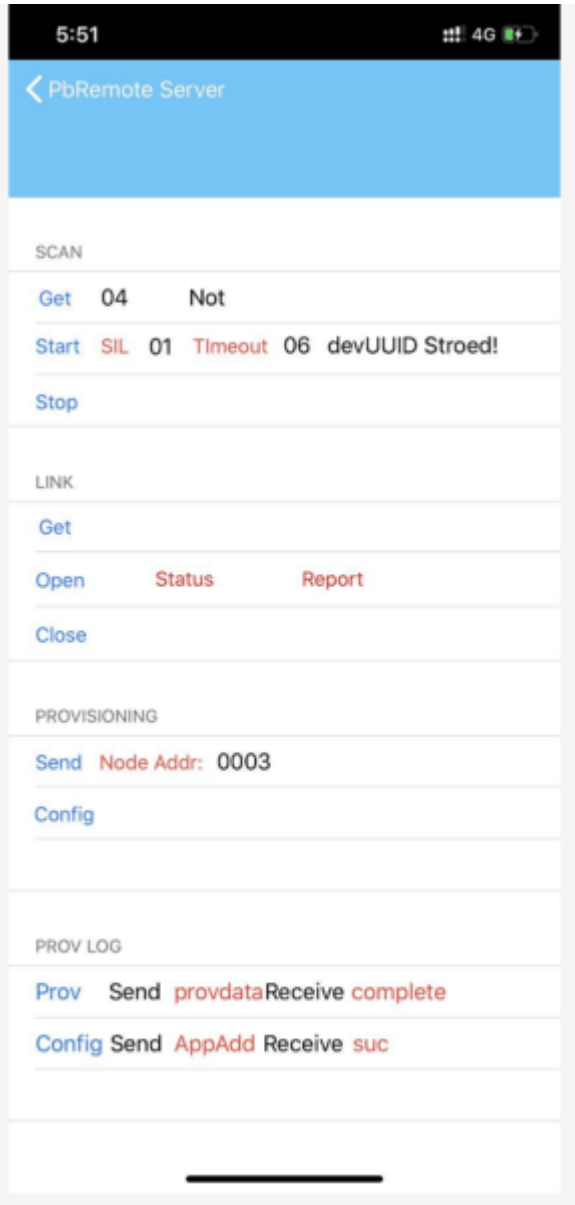
（3）点击 Link 部分 Open 按键，分别收到 Link Status 和 Link Report 消息后，后面的 Label 会变成红色表示建立连接成功。



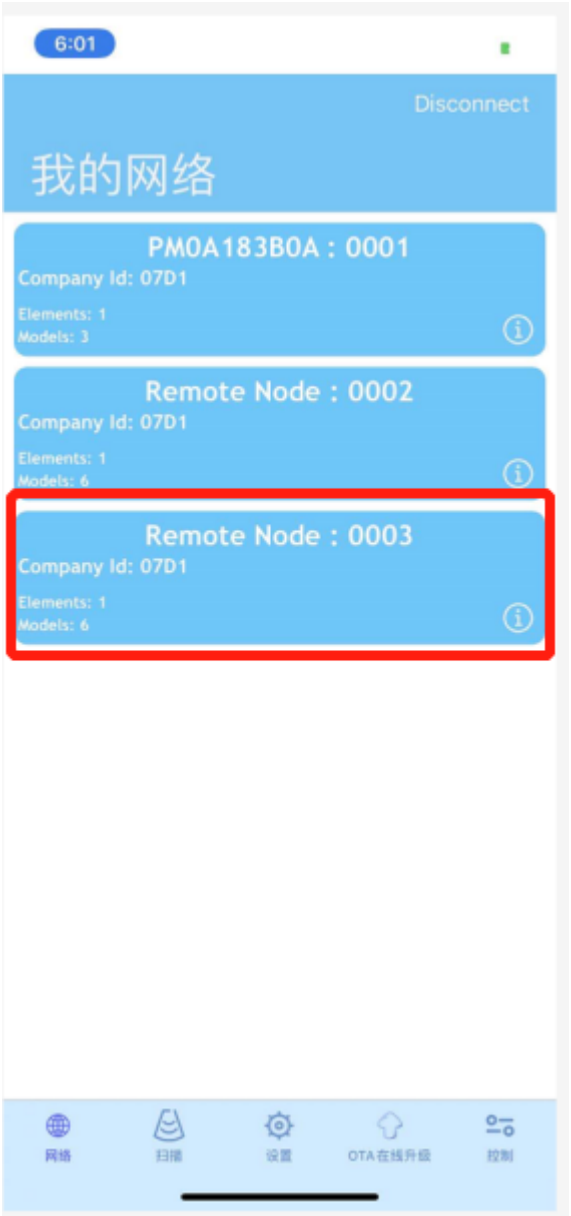
（4）Provisioning 部分，可点击 Node Addr 按键进行节点地址配置（默认地址即可），点击 Send 进行入网消息发送。



（5）PROV LOG 部分可以显示入网进度，Prov Log Send 依次为 Invite，Start，PublicKey，Confirmation，Random，ProvData，Prov Log Receive 依次为 Capabilities，PublicKey，Confirmation，Random，Complete。入网完成进行配置，Config Log Send 部分依次显示 Composition Get，Appkey Add，Config Log Receive 部分依次显示 Composition，Success 最终结果如图。



1. 入网结束后返回到主界面可以看到 PbRemote 入网的节点成功显示在网络界面，如下图。



1. Reset PbRemote Server 后可以重新对新的节点进行入网。

1.4.4 Panchip Test Vendor Model

1.4.4.1 **概述** Panchip Test Vendor 是 Panchip 开发的对端 Client Server。使用该 Model 时，需要在芯片端勾选 Panchip Test Vendor Enable 如下，Panchip Test Vendor ID：0x07D1（panchip company id）0x0004（PTV Server），主要的操作功能有：

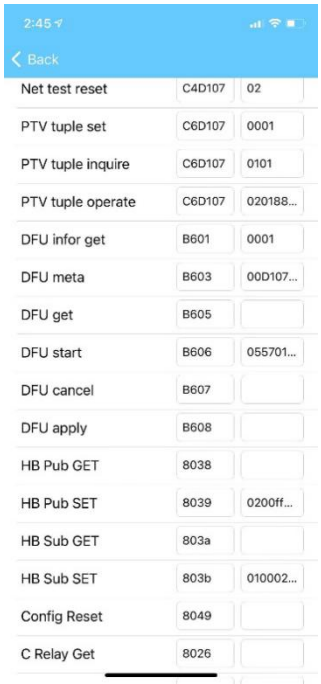
1. Proxy 自动发包测试
2. 三元组查改选
3. 广播参数配置

并在 Vendor 操作界面，增加了 DFU，Config HeartBeat，Config Relay Set 等消息构造。

PTV 定义的消息有如下：

Message	OPCODE
adv para set	C0
adv para set status	C1
packet count	C2
packet count staus	C3
net test set	C4
net test set status	C5
ptv_three_element_handle	C6
ptv_three_element_reply	C7

构造消息界面截图：



1.4.4.2 操作方式介绍

1.4.4.2.1 Proxy **回包测试** 通过 Panchip Test Vendor 可以测试 Proxy 收发包情况。

Step1：准备两块 Server，代码配置如下：

Step2：入网 chip#1（addr 0x0001），chip#2（addr 0x0002），gatt 连接 chip#1 作为 proxy。

Step3：进入 vendor 界面，配置 proxy 发包间隔 300ms 点击 Proxy auto，观察现象。

通过配置参数，默认参数 300ms 可以测试 proxy 节点的通信效率。

1.4.4.2.2 PTV **配置三元组** 参考天猫精灵小度章节，PTV Model 可以替代三元组手动写入方式，三元组数据及三元组查询，操作步骤如下

Step1：点击 choose，选择三元组 set 发送，三元组以 char 形式写入到芯片端。观察芯片段 log 及手机端回复消息。

Step2：选择三元组查询。

Step3：选择三元组读取方式。

1.4.4.2.3 Vendor **界面配置** Config Heartbeat　通过 Vendor 界面，可以选择 Config Heartbeat 消息的处理，操作步骤如下：

Step1：HB Sub Get 发送到 chip#2，HB Sub Set 发送到 chip#2；观察手机及芯片 Log。

Step2：HB Pub Get 发送到 chip#1，HB Pub Set 发送到 chip#1；观察手机及芯片 Log。

注：Heartbeat 配置默认配置了 chip#1（0x0001）每 4s 持续发布到 0x0002 的 heartbeat 消息，配置了 chip#2（0x0002）持续收取源地址为 0x0001 目的地址为 0x0002 的广播包 32S。

1.4.4.2.4 Vendor **界面配置** Config Relay　通过 Vendor 界面，可以选择 Config Relay 消息的处理，操作步骤如下：

Step1：Relay 消息 Set，Get 消息发送，观察手机端显示。

1.4.4.2.5 Vendor **界面配置** Config Reset　通过 Vendor 界面，可以选择 Config Reset 消息的处理，操作步骤如下：

Step1：Relay 消息 Set，Get 消息发送，观察手机端显示。

Step2：Reset 消息发送，并观察 reset node 是否成功退网，手机端是否有 Reset Status。

1.5 Control

1.5.1 DTS **控制**　Mesh 主要应用于灯控，灯控需要配置相应的 PWM，IO 灯外设，外设实现通过 Zephyer DTS 操作方式

1.5.2 **退网**　退网在 Amazon Echo 应用中为重点测试内容

退网是一条 config 消息，并调用相应 `mesh_reset()` 对 flash 存储的入网信息进行删除

1.5.3 heartbeat　Heartbeat tranport 层的消息通信，经过配置可以完成以下两个作用：

1）监测网络

2）监测网络拓扑结构

通过心跳包配置 publication 完成设备监控可以在小度使用中体现。

通过心跳包配置 subscription 可以配置 source 和 destination 可以完成指定源及目的地址配置。

Publication 配置参考小度实现。

Subscription 配置通过手机 config 消息进行配置，后续应用待拓展说明，配置方式见 vendor 配置消息如下：

Heatbeat 示例参考 Panchip Test Vendor Model 相关介绍说明。

1.5.4 subnet　subnet 为 network 层概念，通过绑定不同的 netkey，可以完成子网相关的构建控制

secure beacon 为不同的 subnet 周期发送的广播同步消息

1.5.5 subscribe&publication　在蓝牙 Mesh 里面发消息的动作我们叫做发布（publish）。我想告诉别人什么事情发生或者做什么事情就叫做发布。谁对某些消息感兴趣就可以订阅这些内容。就像是在知乎里面，如果是你对哪个专栏或者内容感兴趣就可以进行订阅。所有发布到这个专栏的文章你都会收到。节点发布消息到单播地址，组播地址或者虚拟地址。节点有兴趣接收这些数据的可以订阅这些地址。

1.6 Function

Function 为为适配芯片特性的一些功能以及支持的一些小的功能模块

1.6.1 **快速重启退网** 快速重启入网通过 Zephyr DTS 操作 flash，需要配置相关的宏使能 zephyr flash data 区存储功能

```
CONFIG_BT_SETTINGS=y
CONFIG_FLASH=y
CONFIG_FLASH_MAP=y
CONFIG_NVS=y
CONFIG_SETTINGS=y
```

1.6.2 **校准** 校准功能为 RF 校准功能，由于芯片随温度变化，RF 性能可能出现波动

需要 ADC 模块检测温度，RF 模块在合适的时机进行校准

如果需要使用某一校准配置作为存储表生成在 Flash 中，需要 Flash 模块温度校准生成温度校准表，后续温度变化至某一范围不需要重新校准，而可以通过读取写入 RF 相关配置进行快速校准。

1.6.3 BOD BOD 为芯片支持的上电低电压检测功能，使能以后低电压启动会进行复位

4.8.2 2 Reference

Mesh 需要额外的参考工具及参考文档，由下述内容说明

2.1 Reference Tools

Mesh 对端测试可以借助多种外界设备进行功能验证/做成相对应的产品或者方案

以下列举了多种测试软件的功能及基础说明。

2.1.1 Nrf Mesh App nrf mesh app 包括 IOS/Android 两种操作系统的 App，需要在相应的应用商店进行下载，运行时需要打开蓝牙许可权限。

可以验证其作为 pb gatt provisioner 对芯片端进行入网，并作为相应的 model client 对芯片端进行 gatt 控制。

Nrf Mesh App 可以验证 Output String oob 入网方式。

2.1.2 Silicon lab Mesh App 包括 IOS/Android 两种操作系统的 App，需要在相应的应用商店进行下载，运行时需要打开蓝牙许可权限。

可以验证其作为 pb gatt provisioner 对芯片端进行入网，并作为相应的 model client 对芯片端进行 gatt 控制

2.1.3 PanMesh Ios App PanMesh Ios App 为 Panchip 内部开发的 IOS 测试 Mesh 软件

安装方式：

Iphone 安装 testflight 并通过 https://testflight.apple.com/join/rS9OwagI 安装 PanMesh

测试演示功能主要包括如下：

1. pb gatt 入网
2. 灯控（组控，proxy，relay）
3. heartbeat
4. sig ota
5. pb remote
6. proxy packet test

7. etc

2.1.4 天猫精灵 Mesh　天猫精灵智能音箱可以作为 Pb adv provisioner 对芯片端进行入网，并通过 adv 方式进行灯控演示

天猫精灵 App 可以作为 Pb gatt provisioner 对芯片端进行入网，并通过 gatt 连接方式进行灯控演示

天猫精灵需要对应芯片端实现定时相关的 vendor model

天猫精灵需要三元组（pid4，mac6，secret16）共 26Bytes 的三元组信息，来生成对应的 Dev UUID，并通过 ECC 方式（sha256）由三元组生成 authvalue，在入网时进行 authvalue 云端验证，保证入网流程的正常进行

mesh 设备在 Provisioning Capabilities 阶段提供 OOB 方式，要求唯一支持 Static OOB 方式，其中的 AuthValue 计算过程如下：

AuthValue = SHA256(Product ID,MAC,Secret)。

即：将 ProductID，MAC，Secret 三元组通过字符串用**英文逗号**连接，然后进行 SHA256 摘要计算，取前 16 字节。注：这里用于计算 SHA256 的**英文字母全部为小写**。

三元组在开发者平台注册产品时会生成调试用的三元组，量产三元组请走开发者平台量产流程。

SHA256 计算示例

数据字段	数据格式与示例	计算使用的输入字符串
Product ID	十进制数值：168930，对应十六进制数值：0x293e2	“000293e2”
Mac Address	“AB:CD:F0:F1:F2:F3”（扫描到的蓝牙设备 MAC 地址）	“abcdf0f1f2f3”
Secret	“53daed805bc534a4a93c825ed20a7063”	“53daed805bc534a4a93c825ed20a7063”
连接后字符串	“000293e2,abcdf0f1f2f3,53daed805bc534a4a93c825ed20a7063”	
SHA256 结果输出 (HEX)	c1 c7 67 41 55 32 36 fb 7d a0 a5 86 e6 22 98 c2 31 da c2 88 5e 73 5f eb a6 b8 b441 7c 7d 9e 72	
Auth-Value(HEX)	c1 c7 67 41 55 32 36 fb 7d a0 a5 86 e6 22 98 c2	

2.1.4.1 三元组生成　三元组数据的详细定义可以参考如下链接：

https://doc-bot.tmall.com/docs/doc.htm?spm=0.7386797.0.0.6e071780ZzBI8F&source=search&treeId=578&articleId=1

通过登录网站 https://living.aliyun.com 添加相关新产品，同时生成三元组，添加完成之后，有图所示的结果

三元组生成后，留作芯片端写入接口进行使用

2.1.4.2 天猫精灵操作介绍　（1）在 mesh_application 工程里打开天猫精灵功能，填入新申请的天猫精灵三元组信息后，编译，生成应用层 bin 文件，通过 Panchip PAN1080 MESH Programmer 将应用层 bin 文件和 Profile 层 bin 文件烧录进 PAN1080 蓝牙模组中。

（2）在芯片运行起来之后，可以先按擦除按键，然后重启，确保芯片处于未配网状态。

（3）按照天猫精灵使用手册，通过天猫精灵手机 APP（IOS 或 Android 均可）绑定天猫精灵设备。

（4）对天猫精灵说“天猫精灵，找队友”，在天猫精灵发现智能设备之后对天猫精灵说“连接”，即可对天猫精灵进行配网。

（5）配网成功之后，可以通过语音指令对天猫精灵进行控制。目前支持的语音指令包括： “天猫精灵，打开灯”，“天猫精灵，关闭灯”；

“天猫精灵，将灯的亮度调整为 XX%（调整范围为 1%-100%）”；

“天猫精灵，将灯的色温调整为 XX%（调整范围为 0%-100%）”；

“天猫精灵，多少时间打开/关闭灯（目前天猫官方只支持定时打开/或者关闭灯，暂不支持定时调整亮度和色温，而且最小定时时间单位为分钟，如果定时不是在整分钟定时，定时可能会有误差）”。

在天猫精灵手机 APP 中对智能设备进行配置之后，相应的语音指令也会发生变化，用户可以自行摸索。

注： （1）天猫精灵对 PAN1080 蓝牙模组入网之后，再次进行入网时，需要擦除 PAN1080 蓝牙模组的配网信息，而天猫精灵手机 APP 上已显示的设备可以不用解绑。

（2）芯片刚烧录好天猫天猫精灵代码后，模组上的灯会呈现绿色。后续通过天猫精灵操作灯时，灯会呈现白色。

（3）由于我们模组上没有色温灯，目前我们是通过三色灯模拟色温的方式来展现给用户色温调节的效果。色温设置百分比偏低的时候，偏冷色调的蓝色比重会比较大；色温设置百分比偏高的时候，偏暖色调的红色比重会比较大。

2.1.5 **小度音箱** Mesh

小度智能音箱可以作为 Pb adv provisioner 对芯片端进行入网，并通过 adv 方式进行灯控演示

小度音箱监测设备在线状态可以演示 heartbeat 的使用

小度音箱需要三元组（pid4，mac6，secret16）共 26Bytes 的三元组信息，来生成对应的 Dev UUID，并通过 ECC 方式（sha256）由三元组生成 authvalue，在入网时进行 authvalue 云端验证，保证入网流程的正常进行

2.1.5.1 三元组生成

2.1.5.2 小度音箱操作介绍

2.1.6 Amazon Echo Mesh

Amazon Echo 智能音箱可以作为 Pb gatt provisioner 对芯片端进行入网

Amazon Echo 智能音箱可以通过 gatt 连接方式进行控制，也可以通过 adv 的方式进行控制

Amazon Echo 智能音箱需要适配 Amazon pb remote 来进行远程入网

Amazon Echo 智能音箱（厂商）正在确认和适配 Amazon sig ota 来进行节点升级

2.2 Reference Documents

2.3 Reference Function

2.3.1 Menu Config **使用**

当查看 PROXY 功能使能时，追溯 BT_CONN 开启方式

```
config BT_CONN
        bool
```

由于 BT_CONN bool 后无注释, 不能通过 `CONFIG_BT_CONN=y` 直接使能

通过 menuconfig 功能追溯可得

```
Name: BT_CONN
Type: bool
Value: y
Direct dependencies (=y):                                                    ␣
↪                               BT_HCI(=y)
&& BT(=y)
Symbols currently y-selecting this symbol:                                   ␣
↪                             - BT_PERIPHERAL
Symbols currently n-selecting this symbol (no effect):                       ␣
↪                             - BT_CENTRAL
Kconfig definition, with parent deps. propagated to 'depends on'             ␣
↪                           ============================================================
At subsys/bluetooth/Kconfig:124
Included via E:/zephyr_panchip_2.7.0/zephyr/Kconfig:8 -> Kconfig.zephyr:44 -> subsys/Kconfig:9
Menu path: (Top) -> Sub Systems and OS Services -> Bluetooth
config BT_CONN                                                               ␣
↪                                bool
depends on BT_HCI(=y) && BT(=y)
```

4.9 Zephyr Bootloader 开发指南

本文介绍 zephyr bootloader 使用方法，及相关 demo 展示的操作步骤配置。

相关工程位置：

- Bootloader：`bootloader/mcuboot/boot/zephyr`
- App：`zephyr/samples_panchip/hello_world`

文章分为三部分

- 功能说明：对 boot 的功能，区域划分做整体描述
- 工程编译烧录：包括 mcuboot 工程及 hello world（test sample）
  - mcuboot：编译时包括 5 种编译方式，对应 4 种升级方式 (其中第四种 BOOT_DIRECT_XIP 方式，可以选择是否可以 revert)
  - hello_world：测试 app 程序，包括 2 种编译方式（以 code-partition 作为区分），编译后的固件通过 imgtool 处理有 2 种方式，区分升级程序是否为 confirmed 程序
- 演示说明：对相对应得 5 种升级方式做演示说明
- 总结说明：补充一些说明

4.9.1 1 功能说明

mcuboot 为官方 bootloader 程序，支持的平台包括 zephyr/mynewt 等，我们基于 zephyr 平台，需要编译生成可与 zephyr 主程序交互的 boot 程序，来完成固件的解签，升级，标志固件信息等操作

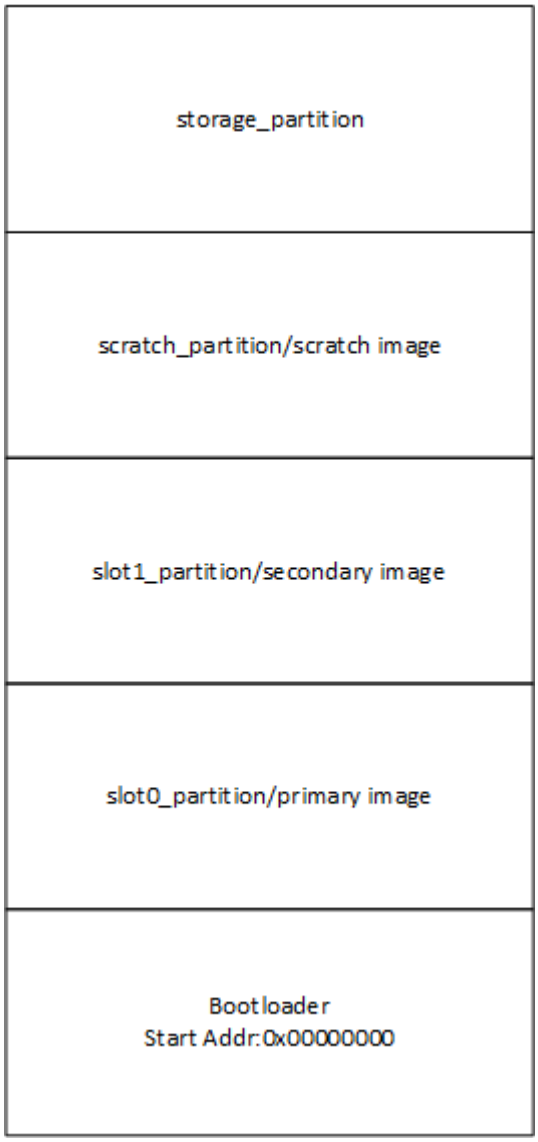
参考官方连接：

- zephyr 中关于 mcuboot 升级功能介绍
  - https://docs.zephyrproject.org/latest/guides/device_mgmt/dfu.html?highlight=mcuboot
- mcuboot 官方文档介绍
  - https://docs.mcuboot.com/
  - https://docs.mcuboot.com/design.html 描述固件升级细节
- mcuboot 关于 zephyr 应用介绍
  - https://www.mcuboot.com/documentation/readme-zephyr/

- 其他参考文档
  - https://blog.csdn.net/u010018991/article/details/79475166
  - https://blog.csdn.net/lt6210925/article/details/119519261

1.1 程序划分

带 bootloader 的程序，支持升级时，通常会按照以下区域进行划分



- bootloader：0x0 地址程序，上电运行 bootloader 主程序
- slot0_partition：app 主固件程序地址，作为 image 时，通常具有 header 和 tailer，并在 image 尾部有签名信息用来确认固件正确性可靠性，tailer 用来存放 magic pad 和升级固件信息（swap_type,copy_done,image_ok）
- slot1_partition：app 待升级固件存放区域，形式如 slot1_partition，但升级后的 tailer 会被擦除
- scratch_partition：可选区域，临时存放固件交换区域，相较于不带 scratch 的升级方式，此种类型固件搬运有一定保障性
- storage_partition：存放存储内容区域，固件升级时通常可以选择不处理

1.2 image 格式

初始编译的 app 程序，对头部进行了 500Bytes 的偏移用作 image header，主要存放固件版本固件大小等信息。

image 程序大小结束位置存放 hash 及签名信息，用来确认固件的完整正确性

image 总大小结束位置存放 image 的状态信息，用来管理升级交换方式

1.2.1 image header

- 大小：500Bytes=0x200
- 位置：存放于 image 头部区域
- 存放信息：

```
struct image_version {
    uint8_t iv_major;
    uint8_t iv_minor;
    uint16_t iv_revision;
    uint32_t iv_build_num;
};

/** Image header.  All fields are in little endian byte order. */
struct                                      image_header {
    uint32_t ih_magic;
    uint32_t ih_load_addr;
    uint16_t ih_hdr_size;           /* Size of image header (bytes). */
    uint16_t ih_protect_tlv_size;   /* Size of protected TLV area (bytes). */
    uint32_t ih_img_size;           /* Does not include header. */
    uint32_t ih_flags;              /* IMAGE_F_[...]. */
    struct image_version ih_ver;
    uint32_t _pad1;
};
```

- 作用：存放固件版本固件大小等信息

1.2.2 image sign infor

- 位置：image 尾部
- 存放信息：
  - hash 值
  - 根据固定 private+hash 值生成的签名信息
- 作用：存放 hash 值及签名信息

1.2.3 image trailer

- 位置：image 总大小的末尾
- 存放信息：

```
/* This is not actually used by mcuboot's code but can be used by apps
 * when attempting to read/write a trailer.
 */
struct image_trailer {
    uint8_t swap_type;
    uint8_t pad1[BOOT_MAX_ALIGN - 1];
    uint8_t copy_done;
    uint8_t pad2[BOOT_MAX_ALIGN - 1];
```

(下页继续)

(续上页)

```
    uint8_t image_ok;
    uint8_t pad3[BOOT_MAX_ALIGN - 1];
    uint8_t magic[16];
};
```

- 作用：用来判断 swap 类型，处理搬运信息，判断固件正确性以及固定 magic pattern 标识

1.2.4 imgtool 位于 `bootloader\mcuboot\scripts` 的 imgtool 为处理原始 image 的 python 脚本，目的是生成待烧录到对应分区的 2 个 bin 文件（分为 Not Confirmed 和 Confirmed，此 2 个文件根据 pad 构建生成）

操作方式如下：

根路径下打开 powershell，执行如下指令

```
python bootloader\mcuboot\scripts\imgtool.py sign --key bootloader\mcuboot\root-rsa-2048.pem --
↪header-size 0x200 --align 8 --version 1.x --slot-size 0x10000 --pad --confirm  build\hello_
↪world_test_board\zephyr\zephyr.bin  build\hello_world_test_board\zephyr\signed1.x.bin
```

- –key ： 跟签名 key 信息，需要与 Bootloader config 宏所对应的解签方式对应 CONFIG_BOOT_SIGNATURE_TYPE_RSA
- –header-size ： header 大小固定 0x200Bytes
- –align ： 8
- –version ： version 版本 1.x
- –slot-size 0x10000：image trailer 存放末尾位置
- –pad：为 image 添加 trailer，升级固件所必须的 pad magic 的添加命令
- –confirm：为 image trailer 添加 confirm 信息，存在则升级方式为 permanent，否则为 test 方式
- 后接原始文件路径及生成固件路径

1.3 升级方式

固件升级分为 4 种方式，通过 `prj.conf` 文件中进行四选一

```
if !SINGLE_APPLICATION_SLOT
choice
        prompt "Image upgrade modes"
        default BOOT_SWAP_USING_MOVE if SOC_FAMILY_NRF
        default BOOT_SWAP_USING_SCRATCH

config BOOT_SWAP_USING_SCRATCH
        bool "Swap mode that run with the scratch partition"
        help
          This is the most conservative swap mode but it can work even on
          devices with heterogeneous flash page layout.

config BOOT_UPGRADE_ONLY
        bool "Overwrite image updates instead of swapping"
        help
          If y, overwrite the primary slot with the upgrade image instead
          of swapping them. This prevents the fallback recovery, but
          uses a much simpler code path.

config BOOT_SWAP_USING_MOVE
        bool "Swap mode that can run without a scratch partition"
        help
```

(下页继续)

(续上页)

```
          If y, the swap upgrade is done in two steps, where first every
          sector of the primary slot is moved up one sector, then for
          each sector X in the secondary slot, it is moved to index X in
          the primary slot, then the sector at X+1 in the primary is
          moved to index X in the secondary.
          This allows a swap upgrade without using a scratch partition,
          but is currently limited to all sectors in both slots being of
          the same size.

config BOOT_DIRECT_XIP
        bool "Run the latest image directly from its slot"
        help
          If y, mcuboot selects the newest valid image based on the image version
          numbers, thereafter the selected image can run directly from its slot
          without having to move/copy it into the primary slot. For this reason the
          images must be linked to be executed from the given image slot. Using this
          mode results in a simpler code path and smaller code size.

endchoice
```

简单介绍四种方式的升级功能：

- 带 scratch 区的升级：BOOT_SWAP_USING_SCRATCH

  此种方式对 flash 占用相对较大，升级时，分为三步

  slot2_partition 搬运到 scratch_partition

  slot1_partition 搬运到 slot2_partition

  scratch_partition 搬运到 slot1_partition

- 不带 scratch 区的搬运升级：BOOT_UPGRADE_ONLY

  直接将 slot2_partition 搬运到 slot1_partition

- 不带 scratch 区的交换固件：BOOT_SWAP_USING_MOVE

  交换 slot1_partition 和 slot2_partition

- 不带 scratch 区的，不进行搬运的升级：BOOT_DIRECT_XIP

  此种方式通过选择宏 `CONFIG_BOOT_DIRECT_XIP_REVERT` 决定是否 revert，分为 2 种子方式

  slot1_partition 和 slot2_partition 不需要搬运

  但需要注意不同 partition 的 code 内，code_partion 对应的 slot_partion 需要在编译时对应，主要用于编译生成不同的 bin 文件时使用

  例如，编译 slot2_partion 时，需要在 `board.dtsi` 文件内，进行相关对应

  ```
  chosen {
                  zephyr,console = &uart0;
                  zephyr,shell-uart = &uart0;
                  zephyr,bt-mon-uart = &uart0;
                  zephyr,bt-c2h-uart = &uart0;
                  zephyr,sram = &sram0;
                  zephyr,flash = &flash0;
                  zephyr,code-partition = &slot0_partition;
          };
  ```

4.9.2 2 工程编译烧录

工程编译烧录介绍 mcuboot（bootloader project），以及 hello world 的编译，烧录方式

2.1 MCUBOOT

mcuboot 为 bootloader 的 project，包括五种编译方式，对应四种升级方式

2.1.1 功能概述 目前 Boot 程序可以完成以下功能

- 解密签名
- 判断固件信息进行 upgrade，有以下五种方式
  - 带 scratch 区的升级
  - 不带 scratch 区的搬运升级
  - 不带 scratch 区的交换固件
  - 不带 scratch 区的，不进行搬运的升级
  - 不带 scratch 区的，不进行搬运的升级，并支持非 confirm 回退
- 程序跳转

2.1.2 环境要求

- board: pan1080a_afld_evb
- uart (option): 显示串口 log

2.1.3 编译和烧录 项目位置：`bootloader/mcuboot/boot/zephyr`

mcuboot 包括 5 种编译方式，需要在 `boot/zephyr/prj.conf` 中关注 5 个 config 宏，根据选择升级方式，选择使用的宏

```
# chioce one upgrade type
CONFIG_BOOT_SWAP_USING_SCRATCH=y
CONFIG_BOOT_UPGRADE_ONLY=y
CONFIG_BOOT_SWAP_USING_MOVE=y
CONFIG_BOOT_DIRECT_XIP=y
CONFIG_BOOT_DIRECT_XIP_REVERT=y
```

方式一（默认）：

```
# chioce one upgrade type
# CONFIG_BOOT_SWAP_USING_SCRATCH=y
# CONFIG_BOOT_UPGRADE_ONLY=y
CONFIG_BOOT_SWAP_USING_MOVE=y
# CONFIG_BOOT_DIRECT_XIP=y
# CONFIG_BOOT_DIRECT_XIP_REVERT=y
```

方式二：

```
# chioce one upgrade type
CONFIG_BOOT_SWAP_USING_SCRATCH=y
# CONFIG_BOOT_UPGRADE_ONLY=y
# CONFIG_BOOT_SWAP_USING_MOVE=y
# CONFIG_BOOT_DIRECT_XIP=y
# CONFIG_BOOT_DIRECT_XIP_REVERT=y
```

方式三：

```
# chioce one upgrade type
# CONFIG_BOOT_SWAP_USING_SCRATCH=y
CONFIG_BOOT_UPGRADE_ONLY=y
# CONFIG_BOOT_SWAP_USING_MOVE=y
# CONFIG_BOOT_DIRECT_XIP=y
# CONFIG_BOOT_DIRECT_XIP_REVERT=y
```

方式四：

```
# chioce one upgrade type
# CONFIG_BOOT_SWAP_USING_SCRATCH=y
# CONFIG_BOOT_UPGRADE_ONLY=y
# CONFIG_BOOT_SWAP_USING_MOVE=y
CONFIG_BOOT_DIRECT_XIP=y
# CONFIG_BOOT_DIRECT_XIP_REVERT=y
```

方式五：

```
# chioce one upgrade type
# CONFIG_BOOT_SWAP_USING_SCRATCH=y
# CONFIG_BOOT_UPGRADE_ONLY=y
# CONFIG_BOOT_SWAP_USING_MOVE=y
CONFIG_BOOT_DIRECT_XIP=y
CONFIG_BOOT_DIRECT_XIP_REVERT=y
```

示例： 以 `pan1080a_afld_evb` 为例，在 SDK 目录打开 PAN1080 SDK CLI 后，可以用如下命令执行编译和下载:

1. 编译

   ```
   west build -d build/mcuboot_test_board -b pan1080a_afld_evb bootloader/mcuboot/boot/zephyr
   ```

2. 下载

   ```
   west flash -d build/mcuboot_test_board -r jlink
   ```

2.2 Hello World

hello_world 为 app 层对应演示 sample，可以区分编译链接 code 地址（通过 flash_map 中的 code_partition 映射），并在后续的工具处理中，可以选择是否为 confirmed 固件

2.2.1 功能概述

当需要配合 mcuboot 进行升级跳转展示时，需要 hello world 完成以下功能

- 打印基础信息 `hello_world`
- 打印 image header 信息，包括 version 版本

2.2.2 环境要求

- board: pan1080a_afld_evb
- uart (option): 显示串口 log

2.2.3 编译链接

项目位置：`zephyr/samples_panchip/hello_world`

hello world 包括 2 中编译方式，对应 code_partition 位置不同，其中不需要搬运的升级方式，需要根据对应需要下载位置，修改 `01_SDK/zephyr/boards/arm/pan1080a_afld_evb/pan1080a_afld_evb.dts` 中的 flash map 的 code_partition 进行修改

```
chosen {
                zephyr,console = &uart0;
                zephyr,shell-uart = &uart0;
                zephyr,bt-mon-uart = &uart0;
                zephyr,bt-c2h-uart = &uart0;
                zephyr,sram = &sram0;
                zephyr,flash = &flash0;
                zephyr,code-partition = &slot0_partition;
        };
```

示例: 以 pan1080a_afld_evb 为例，在 SDK 目录打开 PAN1080 SDK CLI 后, 可以用如下命令执行编译，在对应工程 build 目录下生成 zephyr.bin:

```
west build -d build/hello_world_test_board -b pan1080a_afld_evb zephyr/samples_panchip/hello_
↪world
```

2.2.4 **工具处理** imgtool 处理已编译的 zephyr.bin, 包括对 image 固件添加 header（版本大小），对 image 结尾位置后进行签名，为升级时添加需要的 pad 及 confirm 信息

版本号生成

通过--version 1.x 命令为固件添加具体固件版本信息，例如--version 1.0 即为固件添加版本为 1.0.0.0 的 image header 信息。

是否为 confirmed **固件**，分为以下 2 类示例命令:

不含 confirm 的烧录固件示例如下:

- west 根路径下打开 powershell 运行命令

```
python bootloader\mcuboot\scripts\imgtool.py sign --key bootloader\mcuboot\root-rsa-2048.
↪pem --header-size 0x200 --align 8 --version 1.x --slot-size 0x10000 --pad  build\hello_
↪world_test_board\zephyr\zephyr.bin  build\hello_world_test_board\zephyr\signed1.x.bin
```

  分别生成两个版本的 hello_world 程序，需要对应 signed.bin 文件的目录，例如分别生成 signed1.0.bin，signed1.1.bin

包含 confirm 的烧录固件示例如下:

- west 根路径下打开 powershell 运行命令

```
python bootloader\mcuboot\scripts\imgtool.py sign --key bootloader\mcuboot\root-rsa-2048.
↪pem --header-size 0x200 --align 8 --version 1.x --slot-size 0x10000 --pad ——confirm ␣
↪build\hello_world_test_board\zephyr\zephyr.bin  build\hello_world_test_board\zephyr\
↪signed_confirm1.x.bin
```

  分别生成两个版本的 hello_world 程序，pad 构建为 confirmed，需要对应 signed.bin 文件的目录，例如分别生成 signed_confirm1.0.bin，signed_confirm1.1.bin

2.2.5 **烧录** 烧录时不是烧录原始的 zephyr.bin 文件，需要将工具处理后的文件，通过 jflash 烧录至对应区域，分别为 0x40000 和 0x50000 区域，具体烧录地址在演示说明中对不同升级方式区分描述

4.9.3 3 演示说明

以下对 mcuboot 编译的五种升级方式，hello_world 编译并且通过工具处理的两种固件形式（是否经过 confirm 标识），进行测试现象描述说明:

三种搬运跳转升级方式（对应下文 Method1,2,3）说明

- 升级过程的验证进行时，主要方式是搬运原有程序存在于 slot0，待升级固件（程序链接起始地址 0x40000/slot0 地址）当通过某种方式（flash 烧录/OTA 传输）存在于 slot1 时，重启会对程序进

行搬运处理操作，三种方式处理细节有区别（下文会详细展示），最终将存放于 slot1 的程序搬运至 slot0，搬运结束后会进行一次跳转。

- 根据 slot1 存放的固件是否进行过 confirm 的 trailer pad 添加，区分为 confirmed 固件或者 not confirmed 固件。
  - confirmed 固件——第一次 reset 搬运后，固件之后会标记为 perm（permanent/永久的），之后再 reset 便不会发生变化
  - not confirmed 固件–第一次 reset 时，固件搬运结束后跳转 slot0 运行，由于非 confirm 的固件，只能作为 1 次测试，此时 log 显示 test，第二次复位时，会将之前存在的固件进行 revert 操作，还原为之前 slot 存在的固件并跳转运行，输出 log revert，而之后再进行的复位，由于 revert 标准位的存在，便不会再次进行升级操作。
- 前三种方式验证时可以通过固件的版本来判断运行的是哪个固件，而固件版本本身不会影响升级是否进行。

两种跳转升级方式（对应下文的 Method 4,5**）说明**

- 升级过程未进行搬运程序编译链接时，需要按存放区域构建对应的链接地址（存放于 slot0 的固件需要 code-partition 链接 slot0 区域，而存放于 slot1 的固件需要 code-partition 链接 slot1 区域）当 slot0 或 slot1 区域只存在一个固件时，首次上电便会跳转存在固件的起始地址，而当 slot0，slot1 同时存在固件时，首次上电会对固件进行版本比对，跳转运行较高版本的固件。
- Method 4 和 Method 5 升级方式区别在于，Method 5 支持对固件的 confirm 信息进行判断，版本比对在首次上电逻辑相同，而第二次上电时，上一次跳转的程序若为 not confirmed 固件，则会对上一固件进行擦除，并且跳转至另一区域固件，同样的，再次上电若仍为 not confirmed 固件，则会继续擦除，并最终进行 3 次复位后，在 flash code 区域不存在可运行的 image。
- 后两种方式的固件版本信息不光可以用来区分跳转后的程序内容，还为跳转逻辑提供比对依据。

以下对 5 种模式构建了验证方式，参照例图，对运行模式的说明和演示。

mcuboot 编译方式	hello world 编译方式	hello world 工具处理	对应升级演示
CONFIG_BOOT_SWAP_USING_SCRATCH=y	code_partion = slot0_partition	–pad	Method1 (Not confirmed)
CONFIG_BOOT_SWAP_USING_SCRATCH=y	code_partion = slot0_partition	–pad –confirm	Method1 (confirmed)
CONFIG_BOOT_UPGRADE_ONLY=y	code_partion = slot0_partition	–pad	Method2 (Not confirmed)
CONFIG_BOOT_UPGRADE_ONLY=y	code_partion = slot0_partition	–pad –confirm	Method2 (confirmed)
CONFIG_BOOT_SWAP_USING_MOVE=y	code_partion = slot0_partition	–pad	Method3 (Not confirmed)
CONFIG_BOOT_SWAP_USING_MOVE=y	code_partion = slot0_partition	–pad –confirm	Method3 (confirmed)
CONFIG_BOOT_DIRECT_XIP=y	code_partion = slotx_partition(x 根据烧录位置作区分，分别为 0/1)	–pad	Method4 (Not confirmed)
CONFIG_BOOT_DIRECT_XIP=y	code_partion = slotx_partition(x 根据烧录位置作区分，分别为 0/1)	–pad –confirm	Method4 (confirmed)
CONFIG_BOOT_DIRECT_XIP=y CONFIG_BOOT_DIRECT_XIP_REVERT=y	code_partion = slotx_partition(x 根据烧录位置作区分，分别为 0/1)	–pad	Method5 (Not confirmed)
CONFIG_BOOT_DIRECT_XIP=y CONFIG_BOOT_DIRECT_XIP_REVERT=y	code_partion = slotx_partition(x 根据烧录位置作区分，分别为 0/1)	–pad –confirm	Method5 (confirmed)

下述示例演示过程中，首先需要准备 2 种固件版本，烧录至 slot0（0x40000）的 `hello_world` 程序通过 imgtool 处理为 1.0.0.0，作为**原始固件**，烧录至 slot1（0x50000）的 `hello_world` 程序通过 imgtool 处理为 1.1.0.0，作为**待升级固件**。

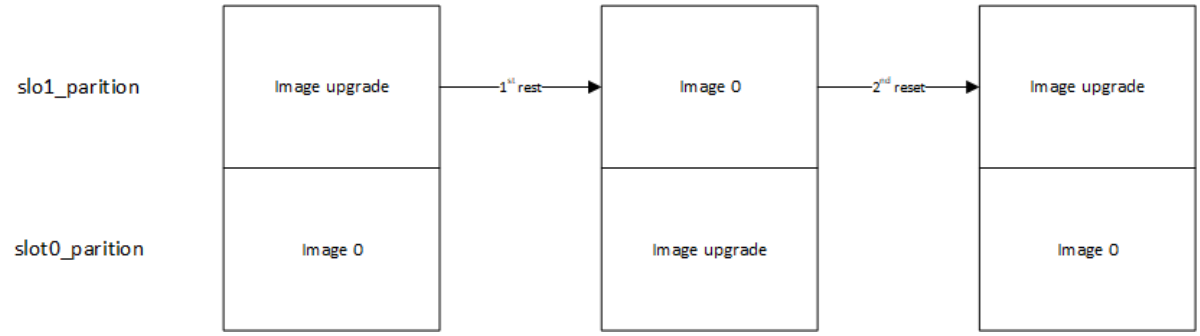
烧录结束后，通过观察 reset 后的 log，可以验证不同方法下升级跳转后具体的运行版本是否符合预期。

3.1 Method 1

默认不带 `scratch` 区的交换固件，`mcuboot sample` 的 `prj.conf` 配置如下

```
# chioce one upgrade type
# CONFIG_BOOT_SWAP_USING_SCRATCH=y
# CONFIG_BOOT_UPGRADE_ONLY=y
CONFIG_BOOT_SWAP_USING_MOVE=y
# CONFIG_BOOT_DIRECT_XIP=y
# CONFIG_BOOT_DIRECT_XIP_REVERT=y
```

boot 程序默认使用第三种方式不带 `scratch` 区的交换固件进行跳转升级，升级流程如下图所示，confirmed 固件进行 1 步 reset，而不带 confirmed 固件需要 2 步 reset，分别运行



3.1.1 Not Confirmed 当烧录至 slot 1（0x50000/secondary image）的程序为不含 confirm 程序时

复位第一次后，程序将打印 test 信息，运行搬运跳转后，运行版本为 1.1.0.0

复位第二次后，程序将打印 revert 信息，运行搬运跳转后，运行版本为 1.0.0.0

复位第三次后，程序将打印 none 信息，不进行搬运，运行跳转后，运行版本为 1.0.0.0

3.1.2 Confirmed 当烧录至 slot 1（0x50000/secondary image）的程序为 confirm 程序时

复位第一次后，程序将打印 perm 信息，运行搬运跳转后，运行版本为 1.1.0.0

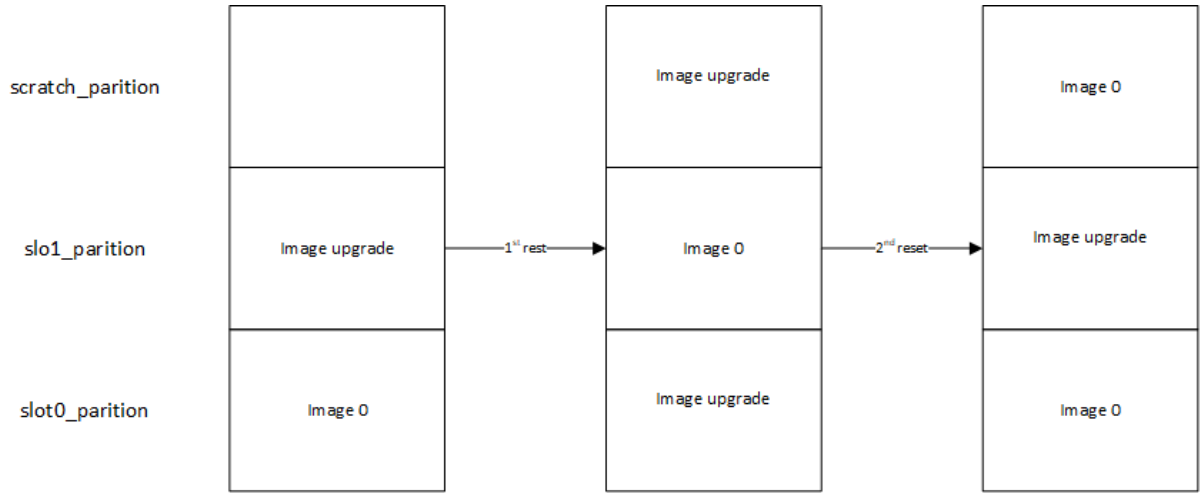
复位第二次后，程序将打印 none 信息，不进行搬运，运行跳转后，运行版本为 1.1.0.0

3.2 Method 2

带 scratch 区的升级，mcuboot sample 的 prj.conf 配置如下

```
# chioce one upgrade type
CONFIG_BOOT_SWAP_USING_SCRATCH=y
# CONFIG_BOOT_UPGRADE_ONLY=y
# CONFIG_BOOT_SWAP_USING_MOVE=y
# CONFIG_BOOT_DIRECT_XIP=y
# CONFIG_BOOT_DIRECT_XIP_REVERT=y
```

boot 程序默认使用第三种方式不带 scratch 区的交换固件进行跳转升级，升级流程如下图所示，confirmed 固件进行 1 步 reset，而不带 confirmed 固件需要 2 步 reset，分别运行



3.2.1 Not Confirmed 当烧录至 slot 1（0x50000/secondary image）的程序为不含 confirm 程序时

复位第一次后，程序将打印 test 信息，运行搬运跳转后，运行版本为 1.1.0.0

复位第二次后，程序将打印 revert 信息，运行搬运跳转后，运行版本为 1.0.0.0

复位第三次后，程序将打印 none 信息，不进行搬运，运行跳转后，运行版本为 1.0.0.0

3.2.2 Confirmed 当烧录至 slot 1（0x50000/secondary image）的程序为 confirm 程序时

复位第一次后，程序将打印 perm 信息，运行搬运跳转后，运行版本为 1.1.0.0

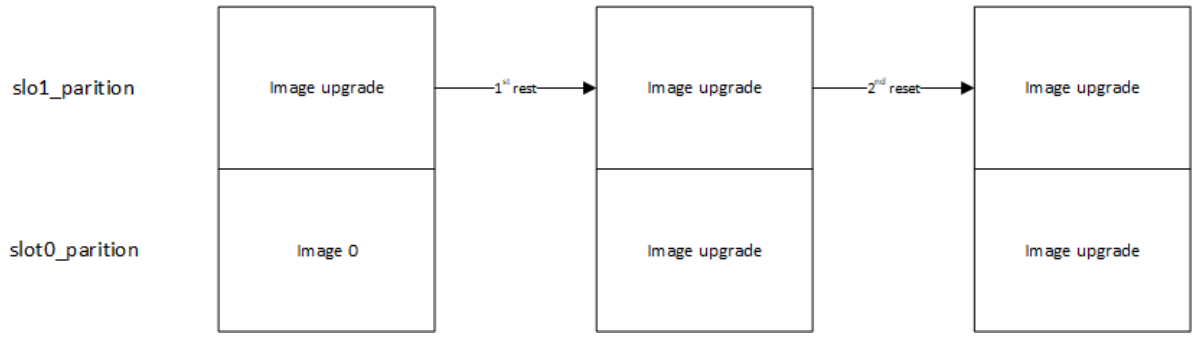
复位第二次后，程序将打印 none 信息，不进行搬运，运行跳转后，运行版本为 1.1.0.0

3.3 Method 3

不带 `scratch` 区的搬运升级，mcuboot sample 的 prj.conf 配置如下

```
# chioce one upgrade type
# CONFIG_BOOT_SWAP_USING_SCRATCH=y
CONFIG_BOOT_UPGRADE_ONLY=y
# CONFIG_BOOT_SWAP_USING_MOVE=y
# CONFIG_BOOT_DIRECT_XIP=y
# CONFIG_BOOT_DIRECT_XIP_REVERT=y
```

boot 程序默认使用第三种方式不带 `scratch` 区的搬运固件进行跳转升级，升级流程如下图所示，confirmed 固件进行 1 步 reset，而不带 confirmed 固件需要 2 步 reset，分别运行



3.3.1 Not Confirmed 当烧录至 slot 1（0x50000/secondary image）的程序为不含 confirm 程序时

复位第一次后，程序将打印 test 信息，运行搬运跳转后，运行版本为 1.1.0.0

复位第二次后，程序将打印 none 信息，不进行搬运，运行跳转后，运行版本为 1.1.0.0

3.3.2 Confirmed 当烧录至 slot 1（0x50000/secondary image）的程序为 confirm 程序时

复位第一次后，程序将打印 perm 信息，运行搬运跳转后，运行版本为 1.1.0.0

复位第二次后，程序将打印 none 信息，不进行搬运，运行跳转后，运行版本为 1.1.0.0

3.4 Method 4

默认不带 `scratch` 区的，不进行搬运的升级，`mcuboot sample` 的 `prj.conf` 配置如下

```
# chioce one upgrade type
# CONFIG_BOOT_SWAP_USING_SCRATCH=y
# CONFIG_BOOT_UPGRADE_ONLY=y
# CONFIG_BOOT_SWAP_USING_MOVE=y
CONFIG_BOOT_DIRECT_XIP=y
# CONFIG_BOOT_DIRECT_XIP_REVERT=y
```

固件准备流程发生变化

hello_world 准备

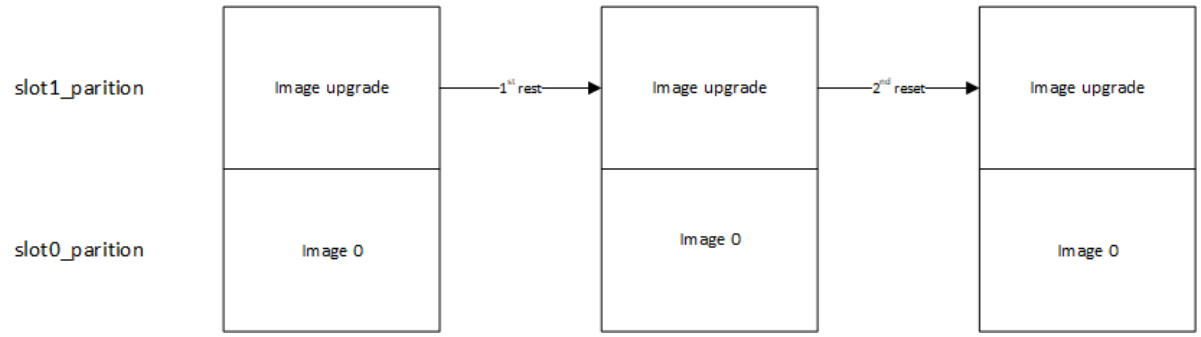
准备 primary image 时, 01_SDK\zephyr\boards\arm\pan1080a_afld_evb\pan1080a_afld_evb.dts 中确认 code-partition 为 slot0_partition

```
zephyr,code-partition = &slot0_partition;
```

准备 secondary image 时, 01_SDK\zephyr\boards\arm\pan1080a_afld_evb\pan1080a_afld_evb.dts 中确认 code-partition 为 slot1_partition

```
zephyr,code-partition = &slot1_partition;
```

分别通过 imgtool 生成不同版本的固件通过 jflash 进行烧录，升级流程如下图所示，confirmed 固件进行 1 步 reset，而不带 confirmed 固件需要 2 步 reset，分别运行



3.4.1 Not Confirmed 当烧录至 slot 1（0x50000/secondary image）的程序为不含 confirm 程序时

最终 reset 通过版本判断，会通过 bootloader 程序跳转到不同的 partition，运行较高版本固件

3.4.2 Confirmed 当烧录至 slot 1（0x50000/secondary image）的程序为 confirm 程序时

最终 reset 通过版本判断，会通过 bootloader 程序跳转到不同的 partition，运行较高版本固件

3.5 Method 5

默认不带 scratch 区的，不进行搬运的升级，支持回退版本的升级，mcuboot sample 的 prj.conf 配置如下

```
# chioce one upgrade type
# CONFIG_BOOT_SWAP_USING_SCRATCH=y
# CONFIG_BOOT_UPGRADE_ONLY=y
# CONFIG_BOOT_SWAP_USING_MOVE=y
CONFIG_BOOT_DIRECT_XIP=y
CONFIG_BOOT_DIRECT_XIP_REVERT=y
```

固件准备流程发生变化

hello_world 准备

准备 primary image 时，zephyr\boards\arm\pan1080a_afld_evb\pan1080a_afld_evb.dts 中确认 code-partition 为 slot0_partition

```
zephyr,code-partition = &slot0_partition;
```

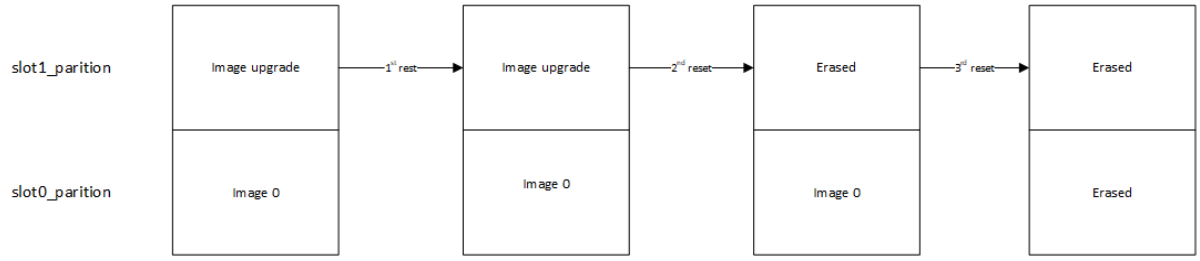
并通过 imgtool 进行待升级固件生成带或者不带 confirmed 固件，留作后续使用

准备 secondary image 时，zephyr\boards\arm\pan1080a_afld_evb\pan1080a_afld_evb.dts 中确认 code-partition 为 slot1_partition

```
zephyr,code-partition = &slot1_partition;
```

并通过 imgtool 进行待升级固件生成带或者不带 confirmed 固件，留作后续使用

通过 jflash 进行烧录不同的固件到对应区域（如 slot0 需要对应 0x40000，slot1 需要对应 0x50000），升级流程如下图所示，confirmed 固件进行 1 步 reset，而不带 confirmed 固件需要 3 步 reset，分别运行



3.5.1 Not Confirmed 当烧录至 slot 0（0x40000/secondary image）的程序为不含 confirm 程序，烧录至 slot 1（0x50000/secondary image）的程序为不含 confirm 程序时

复位第一次后，运行 compare 后，跳转到 slot1_partition，运行版本为 1.1.0.0

复位第二次后，擦除 slot1 后，跳转到 slot0_partition，运行版本为 1.0.0.0

复位第三次后，擦除 slot0 后，无运行版本

3.5.2 Confirmed 当烧录至 slot 0（0x40000/secondary image）的程序为 confirm 程序，烧录至 slot 1（0x50000/secondary image）的程序为 confirm 程序时

最终 reset 通过版本判断，会通过 bootloader 程序跳转到不同的 partition，运行较高版本固件

4.9.4 4 总结说明

以上为目前所有可以支持的升级方式说明，通过 hello_world sample 进行演示：

带 scratch 区的有助于理解搬运过程，但后续由于空间限制不一定采用。

不带 scratch 区的升级主要先以目前 method1 带搬运的升级作为主要验证方式，后续在 1024KB 上只需要替换适配 board.dts 文件，即可对其他 sample（flash 占用较大的程序）进行支持。

不搬运的方式同样可以在项目中考虑，如 method4，method5，不搬运的方式对 flash 使用具有一定好处，但对编译烧录方式有一定限制如上文述。

4.10 Zephyr RAM 使用情况分析指南

此文档记录 Bluetooth 项目中，通用的资源调试分析步骤

如 `mesh_panchip` 类似的 sample，不同需求产生不同配置需求，在资源有限的前提下，需要根据实际占用资源对资源分配情况进行分析和规划

主要包括以下两部分

- 分析编译占用 RAM 资源
  - 通过分析工具，观察宏观占用的资源情况，其中比较大的是线程栈的分配
  - 通过分析工具，观察细节资源占用情况，主要是不同 Feature 开启时，会占用零星的一些资源
- 优化 RAM 资源
  - 通过代码内开启栈实时调用情况打印的接口，运行多种 case，保证不溢出情况下的最小栈分配

 - 通过分析不同 feature 占用的资源，根据需求，裁剪 feature 或者在了解功能需求情况下，缩小 feature 内的变量大小

4.10.1 1 分析编译占用 RAM 资源

使用 VSCode 打开工程，ctrl+shift+B 运行 build 命令后后，TERMINAL 内显示资源总体占用情况，Memory region 可以看到整体占用的 Flash，SRAM，IDT_LIST，例如 mesh_panchip

```
Memory region         Used Size  Region Size  %age Used
         FLASH:      348352 B       1020 KB     33.35%
          SRAM:       49585 B         64 KB     75.66%
      IDT_LIST:           0 GB         2 KB      0.00%
```

其中 SRAM 包含

- Controller RAM Code
- RAM report 工具所分析的 SRAM 资源

1.1 Controller RAM Code

可以通过 bulid 工程目录下 zephyr/zephyr.map 文件确认具体 RAM 占用情况

搜索 ramfunc 即可，例如 mesh_panchip 内，Controller RAM Code 占用资源在 zephyr.map 文件中显示如下：

```
*(SORT_BY_ALIGNMENT(.ramfunc.*))
                0x200017c0                . = ALIGN (_region_min_align)
                0x200017c0                __ramfunc_end = .
                0x000017c0                __ramfunc_size = (__ramfunc_end - __ramfunc_start)
                0x00052aac                __ramfunc_load_start = LOADADDR (.ramfunc)
```

1.2 RAM report 工具所分析的 SRAM 资源

使用 VSCode 打开工程，ctrl+shift+B 运行 RAM Report 命令后后，TERMINAL 内显示资源具体占用情况，通常包含以下几部分，根资源包含以下几部分：

其中数据内容满足以下公式，字母小写同理 $$ X = A + B + C + D $$

$$ D = D1+ D2 + D3 +D4 + D5 $$

$$ D5 = D51+ D52 + D53 + D54 + D55 $$

```
Path                                                                                                    ␣
↪  Size    %
========================================================================================================================
Root                                                                                                    ␣
↪ X     100.00%
   (hidden)                                                                                            ␣
↪A          a%
   (no paths)                                                                                          ␣
↪B          b%
   WORKSPACE                                                                                           ␣
↪C          c%
   ZEPHYR_BASE                                                                                         ␣
↪D          d%
     driver                                                                                            ␣
↪D1        d1%
     kernel                                                                                            ␣
↪D2        d2%
     lib                                                                                               ␣
↪D3        d3%
```

(下页继续)

(续上页)

```
    sample                                                                                                 ␣
↪D4       d4%
    subsys                                                                                                 ␣
↪D5       d5%
      controller_panchip                                                                                   ␣
↪D52     d51%
      host                                                                                                 ␣
↪D53     d52%
      mesh                                                                                                 ␣
↪D54     d53%
      logging                                                                                              ␣
↪D55     d54%
      setting                                                                                              ␣
↪D56     d55%
      storage                                                                                              ␣
↪D57     d56%

==============================================================================================================
                                                                                                           ␣
↪ X
```

分析 RAM report 的内容如下

- ROOT：SRAM 总大小对应占比 100%
- (hidden) ： 隐藏 RAM 大小，通常不大
- (no paths)：包含 zephyr 未追溯到的 RAM 资源大小
- WORKSPACE：ble controller 占用资源大小
- ZEPHYR_BASE：zephyr 资源占用大小，通常包括以下及部分
  - driver：driver 占用的资源，通常与外设选择有关
  - kernel：主要是栈分配，主要包含 z_main_stack
  - lib: 外部调用库，通常包含 std.lib 等
  - sample：sample 应用占用 RAM，自己开发项目时管理
  - subsys
    - controller_panchip: controller 层级占用的 SRAM，主要包含 hci tx/rx stack
    - host: host 层级占用的 SRAM，主要包含 tx_thread_stack , ecc_thread_stack etc.
    - mesh: mesh 层级占用的 SRAM，主要包含 adv_thread_stack
    - logging: logging 占用的资源，不需要 zephyr 内的 log 系统可以关闭
    - setting：setting 占用的资源，通常很小，管理 nvs 机制等的接口
    - storage：storage 占用的资源，通常很小，管理存储接口

4.10.2 2 优化 RAM 资源

优化 RAM 资源分为两个方向：

- 通过代码内开启栈实时调用情况打印的接口，运行多种 case，保证不溢出情况下的最小栈分配
- 通过分析不同 feature 占用的资源，根据需求，裁剪 feature 或者在了解功能需求情况下，缩小 feature 内的变量大小

2.1 栈分配大小优化

栈分配大小优化时，需要以下步骤

- 确认使用栈的个数和大小
- 根据调试栈的代码使能，挂起测试多功能运行时，使用到的最大栈资源，并以此为据，在保证栈不溢出的情况下缩减栈资源

2.1.1 确认使用栈的个数和大小

由上述 RAM Report 工具，可以看到 stack 分配的个数和大小，并且可以通过工程编译目录下 `zephyr\include\generated\autoconf.h` 双重确认

2.1.2 调试栈打印宏

```
CONFIG_THREAD_NAME=y
CONFIG_THREAD_ANALYZER=y
CONFIG_THREAD_ANALYZER_AUTO=y
CONFIG_THREAD_ANALYZER_RUN_UNLOCKED=y
CONFIG_THREAD_ANALYZER_USE_PRINTK=y
CONFIG_THREAD_ANALYZER_AUTO_INTERVAL=5
CONFIG_CONSOLE=y
CONFIG_UART_CONSOLE=y
CONFIG_SERIAL=y
CONFIG_PRINTK=y
```

在 sample `prj.conf` 内增加开启以上宏后，可以每 `CONFIG_THREAD_ANALYZER_AUTO_INTERVAL` 的时间，打印出 STACK 占用累计的最大值

测试时，运行尽量多的功能，保持尽量长的时间，来确认线程中最大栈资源分配

并且还可以打印 CPU 使用率，来观测线程运行的频繁程度

注意：栈打印功能本身也会占用一定资源，栈分配时需要 8B 对齐

打印示例（最大栈运行没问题后，通常保持在分配最大栈的 80~90% 比较合理）：

```
BT Mesh adv          : STACK: unused 164 usage 860 / 1024 (83 %); CPU: 0 %
BT RX                : STACK: unused 1308 usage 1764 / 3072 (57 %); CPU: 1 %
BT CTLR              : STACK: unused 1420 usage 628 / 2048 (30 %); CPU: 7 %
BT ECC               : STACK: unused 212 usage 932 / 1144 (81 %); CPU: 0 %
BT TX                : STACK: unused 1036 usage 1012 / 2048 (49 %); CPU: 0 %
thread_analyzer      : STACK: unused 556 usage 468 / 1024 (45 %); CPU: 0 %
sysworkq             : STACK: unused 1200 usage 848 / 2048 (41 %); CPU: 0 %
logging              : STACK: unused 236 usage 532 / 768 (69 %); CPU: 0 %
idle 00              : STACK: unused 972 usage 52 / 1024 (5 %); CPU: 89 %
```

栈大小具有底层默认值，配置修改时，需要在 `prj.conf` 内修改，如 `mesh_pacnhip` sample 中示例

```
#           main stack
CONFIG_MAIN_STACK_SIZE=1024
#           isr stack
CONFIG_ISR_STACK_SIZE=2048
#           idle 00
CONFIG_IDLE_STACK_SIZE=128
#           sysworkq
CONFIG_SYSTEM_WORKQUEUE_STACK_SIZE=1024
#           BT ECC
CONFIG_BT_HCI_ECC_STACK_SIZE=1024
#           BT TX
CONFIG_BT_HCI_TX_STACK_SIZE_WITH_PROMPT=y
CONFIG_BT_HCI_TX_STACK_SIZE=1024
#           BT RX
CONFIG_BT_RX_STACK_SIZE=2048
```

(下页继续)

(续上页)

```
#           BT CTLR
CONFIG_BT_CTLR_RX_PRIO_STACK_SIZE=1024
#           BT Mesh adv
CONFIG_BT_MESH_ADV_STACK_SIZE=1024
#           logging
CONFIG_LOG_PROCESS_THREAD_STACK_SIZE=768
#           thread_analyzer
CONFIG_THREAD_ANALYZER_AUTO_STACK_SIZE=512
```

需要注意：修改 tx stack 需要按以下方式，需要开启 `CONFIG_BT_HCI_TX_STACK_SIZE_WITH_PROMPT`

```
CONFIG_BT_HCI_TX_STACK_SIZE_WITH_PROMPT=y
CONFIG_BT_HCI_TX_STACK_SIZE=1024
```

2.2 裁剪或减少 Feature 资源

通过上述 RAM report 分析后，可以观察到部分模块，也就是对应的 feature，会零星得占用 RAM 资源，这些资源是我们可以按照需求进行调整的，以减小资源占用

- 当某些模块在某些需求内不需要时，可以进行关闭，如 Mesh Friend Feature 在目前音箱中不需要则可以进行关闭
- 当某些模块在某些需求内，通过较小的数组就可以满足需求时，可以缩小数组变量的大小，如 Mesh Remote Feature/Provisioner 扫描到得设备数量，减少一个便可以减少 16 字节的 UUID 数组存储

2.2.1 分析工具 使用 VSCode 打开工程，`ctrl+shift+B` 运行 `Menu Config` 命令后后，弹出 Kconfig 分析文件

关于 Kconfig 文件的原理，可以参考官方文档 https://docs.zephyrproject.org/latest/guides/build/index.html?highlight=k

Kconfig 分析界面如下

2.2.2 裁剪规则 通过上述分析工具 `Menu Config`，可以观察到，通常情况下，不同模块根据功能来区分，裁剪/缩小资源占用可以按照以下步骤

- 找到对应 Feature 所处的总控制宏，可以对 Feature 进行关闭，对比开关造成的 RAM 资源不同关系
- 某一 feature 下可以通过 feature 内的配置某些数组的大小来统计 RAM 资源细节调整

4.10.3 3 总结说明

经过上述内容细节观察调试，我们知道，当资源比较紧张时

首先可以根据分析工具，观察整体和各部分资源占用情况

宏观上可以对较大的栈进行内存分配管理，细节上可以根据需求来裁剪/缩小变量数组

最后，当我们编码时，同样需要宏观上为线程合理分配栈大小，并且尽量使用 Kconfig 来规划 Feature 的整体和细节，方便后续调优/规整代码

4.11 常见问题（FAQs）

4.11.1 Q1：为什么我的 PC 编译 PAN1080 App 程序的速度特别慢？

一般来说，App 的编译速度与您的 PC 硬件配置有关，但如果相同 PC 配置条件下，您发现自己的 PC 编译速度仍然比其他 PC 慢许多（如 10 倍以上），那么可能是杀毒软件或带有主动防御功能的安全软件影响（如 360 安全卫士、360 杀毒等），**请关闭杀软后再尝试编译**。

4.11.2 Q2：除了 JLink 以外，PAN1080 SDK 是否支持其他调试工具？

Zephyr Project 支持包括 JLink、OpenOCD、PyOCD 等多个调试工具和软件，但是，Panchip 提供的 PAN1080 Toolchain 默认只包含了 JLink，且 SDK 中提供的所有 Board 均被默认配置为使用 JLink，因此，**目前您只能使用 JLink 进行调试**。

当然，PAN1080 SDK 是开源的，如果您了解相关知识，也可以参考 Zephyr 官方文档，根据自己的需要将 SDK 配置为使用其他工具进行调试。

4.11.3 Q3：为什么我尝试复制编译工具链 Toolchain 目录到其他位置会报错？

这是已知问题，原因是编译工具链 `Toolchain` 目录中一些文件路径比较深，导致 Windows 文件管理器无法正常复制这些文件。

目前有以下几个 workaround 方法供您参考：

1. 重新从压缩包中解压 `Toolchain` 目录到您期待的位置；
2. 对 `Toolchain` 目录执行剪切操作，而不是复制操作；
3. 尝试使用命令行进行复制操作；

4.11.4 Q4：为什么我编译 SDK 中的某些默认例程会出现错误？

PAN1080 SDK 中的例程本身都是经过测试后发布的，如果您在一个干净的 SDK 环境中编译某个例程，仍然出现编译错误，则可能是 SDK 目录名称长度过长导致的。

由于 Windows 的限制，一个文件完整路径的长度默认不可超过 260 字节；实际上，此规则会在编译时，由 Zephyr CMake Build System 检查：如果路径长度超过了 250 字节，则编译时会出现如下形式的 CMake 警告：

```
-- Configuring done
CMake Warning in D:/Panchip-Development-Kits/pan1080-dk-internal/01_SDK/zephyr/drivers/
↪interrupt_controller/CMakeLists.txt:
  The object file directory

    D:/Panchip-Development-Kits/pan1080-dk-internal/01_SDK/build/bluetooth_peripheral_hr_low_
↪power_xtl32768_pan1080a_afld_evb/zephyr/drivers/interrupt_controller/CMakeFiles/drivers__
↪interrupt_controller.dir/./

  has 205 characters.  The maximum full path to an object file is 250
  characters (see CMAKE_OBJECT_PATH_MAX).  Object file

    D_/Panchip-Development-Kits/pan1080-dk-internal/01_SDK/zephyr/misc/empty_file.c.obj

  cannot be safely placed under this directory.  The build may not work
  correctly.
```

并**大概率**随后会伴随编译错误：

```
D:/Panchip-Development-Kits/pan1080-dk-internal/01_SDK/modules/hal/panchip/panplat/pan1080/bsp/
↪device/Source/system_PANSeries.c:106:1: fatal error: opening dependency file modules\panchip\
↪panplat\pan1080\bsp\CMakeFiles\..__modules__hal__panchip__panplat__pan1080__bsp.dir\device\
↪Source\system_PANSeries.c.obj.d: No such file or directory
  106 | }
      | ^
compilation terminated.
[98/149] Building C object zephyr/drivers/serial/CMakeFiles/drivers__serial.dir/uart_panchip.c.
↪obj
ninja: build stopped: subcommand failed.
FATAL ERROR: command exited with status 1: 'D:\Panchip-Development-Kits\pan1080-dk-internal\05_
↪TOOLS\Toolchain\cmake-3.22.0-rc1-windows-x86_64\bin\cmake.EXE' --build 'D:\Panchip-
↪Development-Kits\pan1080-dk-internal\01_SDK\build\bluetooth_peripheral_hr_low_power_xtl32768_
↪pan1080a_afld_evb'
```

（下页继续）

(续上页)

如果出现这种 CMake 警告，请**务必**缩短 SDK 路径长度，方法可以是缩短目录本身的名称长度，也可以是减小 SDK 所在的目录深度，直到此警告消除。

例如，修改前 SDK 的目录绝对路径为 `D:/Panchip-Development-Kits/pan1080-dk-v0.3.0/01_SDK/`，那么我们可以尝试将其缩短成这样：`D:/PDK/pan1080dk-v0.3.0/01_SDK/`。

> **注意**：有时候，上述的 CMake 警告出现了，但后续编译仍然可以成功；这给我们造成一种假象：似乎当前编译过程没有问题。但实际上，此次即使编译成功，其编译出来的 Image 也是存在隐患的，不一定可以正常运行。因此，我们在编译时应当时刻关注编译过程 Log，确保无任何 CMake 警告后，再进行后续的烧录调试。

Chapter 5

更新日志

5.1 PAN1080 DK v0.3.0

PAN1080 Development Kit `v0.3.0` (2022-06-03) 已发布：

5.1.1 1. SDK

Zephyr 平台

- 适配 Zephyr ACC Driver （新增非标准接口）
- 修复 Zephyr ADC Driver 适配问题
- 修复 Zephyr GPIO Driver 适配问题
- 适配 Zephyr I2C Driver (支持 master 和部分 slave 特性)
- 调整 Zephyr PINMUX Driver 适配接口
- 适配 Zephyr QDEC Driver （新增非标准接口）
- 适配 Zephyr SPI Driver (目前只支持 master 模式)
- 调整 Zephyr 默认编译优化选项，由 Size Optimize 改为 Speed Optimize
- 调整 `CONFIG_MAIN_STACK_SIZE`，从 1024 增加至 1280
- 关闭 `CONFIG_DYNAMIC_INTERRUPTS` 使能
- 修改 PAN1080 SoC 的命名，并新增一些 SoC 型号
- 新增 SoC FW Encryption 的底层框架支持
- 优化 Zephyr 低功耗流程
- 优化 PAN1080 HAL 层 Driver，并修复一些问题

协议栈

- 更新 libble_controller，支持动态修改发射功率 (目前仅支持 -45dBm, 0dBm to 7dBm)
- 修复 Bluetooth Central 无法正确建立连接的问题
- 修复长时间连接异常断开连接的问题
- 支持 Bluetooth & Proprietary Radio 2.4G 双模同时工作
- 优化功耗

- 恢复 net_buf alloc 超时配置, 避免申请失败导致的异常
- 调整堆栈大小
- 修改 `BT_CTLR_SLEEP_CLOCK_SOURCE` 的描述

例程

- 新增例程:
  - `drivers/acc`: ACC 功能演示例程
  - `drivers/adc`: ADC 功能演示例程
  - `drivers/counter`: Counter (HW Timer) 功能演示例程
  - `drivers/flash_shell`: Flash 功能演示例程
  - `drivers/gpio`: GPIO 功能演示例程
  - `drivers/i2c_master`: I2C master 功能演示例程
  - `drivers/i2c_slave`: I2C slave 功能演示例程
  - `drivers/pinmux`: PINMUX 功能演示例程
  - `drivers/pm`: Zephyr Power Management（低功耗）功能演示例程
  - `drivers/pwm_rgb`: PWM 功能演示例程
  - `drivers/qdec_pwm`: QDEC 功能演示例程
  - `drivers/spi_master`: SPI master 通信演示例程
  - `drivers/uart_fifo`: UART 功能演示例程
  - `bluetooth/peripheral_ota`: BLE OTA 功能演示例程
- 问题修复或优化:
  - `bluetooth/mult_roles`: 解决 central 只能连接一次的问题
  - `bluetooth/peripheral_identity`: 解决运行时的堆栈溢出
  - `bluetooth/peripheral_hids`: 修改默认配置，支持与多个设备配对，支持保留 2 组配对信息
  - `bluetooth/peripheral_ota`: 新增例程，用于演示通过蓝牙连接方式进行 OTA
  - `solutions/model_mouse`: 增加同步流程，增加软件 buffer 缓存机制，修改 payload 为动态 payload 方式
  - `solutions/usb_dongle`: 增加同步流程，增加软件 buffer 缓存机制，修改 payload 为动态 payload 方式
  - 更新一些例程的 config 文件名
- 已知问题:
  - `bluetooth/audio_client (audio_server)`: 收发数据时可能因为环境等问题导致阻塞

测试用例

- 新增用例:
  - `drivers/acc/acc_api`: 测试 Zephyr ACC API
  - `drivers/acc/i2c_api`: 测试 Zephyr I2C API
  - `drivers/spi/spi_loopback`: 测试 Zephyr SPI API
- 问题修复或优化:

  - drivers/gpio/gpio_api_1pin: 优化测试流程
  - subsys/settings/nvs/raw: 修复编译失败的问题

5.1.2 2. HDK

无

5.1.3 3. DOC

- 新增文档:
  - 02_hardware/hardware_reference_design/pan1080_hw_reference_design.md: 新增硬件设计参考文档
  - 03_samples/basic/blinky.md: 新增 blinky 例程说明文档，演示 GPIO 控制 LED 灯闪烁
  - 03_samples/basic/synchronization.md: 新增 synchronization 例程说明文档，演示 zephyr 多线程调度
  - 03_samples/bluetooth/central.md: 新增 bluetooth central 例程说明文档
  - 03_samples/bluetooth/central_hr.md: 新增 bluetooth central_hr 例程说明文档
  - 03_samples/bluetooth/central_ht.md: 新增 bluetooth central_ht 例程说明文档
  - 03_samples/bluetooth/central_multilink.md: 新增 bluetooth central_multilink 例程说明文档
  - 03_samples/bluetooth/eddystone.md: 新增 bluetooth eddystone 例程说明文档
  - 03_samples/bluetooth/ibeacon.md: 新增 bluetooth ibeacon 例程说明文档
  - 03_samples/bluetooth/peripheral.md: 新增 bluetooth peripheral 例程说明文档
  - 03_samples/bluetooth/peripheral_csc.md: 新增 bluetooth peripheral_csc 例程说明文档
  - 03_samples/bluetooth/peripheral_dis.md: 新增 bluetooth peripheral_dis 例程说明文档
  - 03_samples/bluetooth/peripheral_esp.md: 新增 bluetooth peripheral_esp 例程说明文档
  - 03_samples/bluetooth/peripheral_hids.md: 新增 bluetooth peripheral_hids 例程说明文档
  - 03_samples/bluetooth/peripheral_hr.md: 新增 bluetooth peripheral_hr 例程说明文档
  - 03_samples/bluetooth/peripheral_ota.md: 新增 bluetooth peripheral_ota 例程说明文档
  - 03_samples/drivers/acc.md: 新增 acc 例程说明文档
  - 03_samples/drivers/adc.md: 新增 adc 例程说明文档
  - 03_samples/drivers/counter.md: 新增 counter 例程说明文档
  - 03_samples/drivers/flash_shell.md: 新增 flash_shell 例程说明文档
  - 03_samples/drivers/gpio.md: 新增 gpio 例程说明文档
  - 03_samples/drivers/i2c_master.md: 新增 i2c master 例程说明文档
  - 03_samples/drivers/i2c_slave.md: 新增 i2c slave 例程说明文档
  - 03_samples/drivers/pinmux.md: 新增 pinmux 例程说明文档
  - 03_samples/drivers/pm.md: 新增 pm (power management) 例程说明文档
  - 03_samples/drivers/pwm_rgb.md: 新增 pwm_rgb 例程说明文档
  - 03_samples/drivers/qdec_pwm.md: 新增 qdec_pwm 例程说明文档
  - 03_samples/drivers/spi_master.md: 新增 spi master 例程说明文档

  - 03_samples/drivers/uart_fifo.md：新增 uart_fifo 例程说明文档
  - 03_samples/solutions/ble_hid_selfie.md：新增 bluetooth hid selfie 方案说明文档
  - 03_samples/solutions/usb_dongle.md: 新增 usb dongle 方案说明文档
  - 04_dev_guides/zephyr_board_guidance.md: 新增 zephyr board 配置指南文档
  - 04_dev_guides/zephyr_configuration_guidance.md: 新增 zephyr 配置系统指南文档
- 更新文档:
  - 01_quick_start/quick_start_pan1080_sdk.md: 修复一些描述错误
  - 01_quick_start/sdk_framework.md: 补充 sdk 目录框架说明
  - 02_hardware/evaluation_board_introduction/pan1080_evb_intro.md: 修改文档路径
  - 03_samples/bluetooth/mult_roles.md: 更新注意事项描述
  - 03_samples/bluetooth/peripheral_ht.md: 更新图片
  - 03_samples/solutions/model_mouse.md: 更新图片，修改文档名称
  - 03_samples/README-internal.md: 更新例程文档总目录，新增 driver 例程说明
  - 04_dev_guides/Developing Bluetooth Mesh Applications.md: 更新部分描述
  - 04_dev_guides/faqs.md: 新增一些常见问题的解答
  - 04_dev_guides/how_to_develop_soc_app.md: 更新部分参考文档
  - 05_soc_manual/PAN1080 产品说明书.pdf: 更新 PAN1080 产品说明书
  - README-internal.md: 更新文档总目录

5.1.4 4. TOOLS

- 更新 Zephyr App Launcher 工具至 v1.1.1:
  - 修复 Build / OpenIDE 等按钮功能无效的问题
  - 新增更新 board 和 project 列表的按钮
  - 优化列表显示，将 project 与 config 拆分到两个列表中显示
  - 修复无法正确识别 Overlay Config 文件的问题
  - 优化界面显示，为按钮增加 Tooltip 快捷帮助信息

5.1.5 5. ISSUES

已解决问题

- ISSUE #41: Zephyr App Launcher for PAN1080 问题
- BUG #288: central 测试—Central 设备主动扫描功能，扫描到对端设备连接之后立即断链
- BUG #289: 蓝牙多角色功能实际支持的连接个数比配置的个数少一个
- BUG #290: peripheral_identity 测试—测试设备支持的最大连接个数时，出现设备发不出广播的情况
- BUG #295: mult_roles 测试—配置 0 主 8 从-链路超时连接失败
- BUG #299: peripheral_hids 测试—多次进行连接、测试、重连操作、会出现断链后但是设备仍然处于连接状态
- BUG #311: central_ht 测试—主设备 central_ht 无法与 peripheral_ht 建立连接，也无温度数据上报

- BUG #312: mult_roles 测试—配置 3 主 0 从-设备作为 Master，验证设备可以支持的最大链路数失败，建立一条链路，建立第二条时，会强制把第一条链路断开
- BUG #314: peripheral 测试—连接、断链、重连测试中，出现连接一次成功后，第 2 次连接异常，无法再次连接现象
- BUG #315: prf-2.4g 测试—点对点通信，过一段时间会偶现接收端 rx data crc err
- BUG #317: peripheral_hids 测试—苹果手机 iphone6s 和华为 nova 手机均不能与设备进行正常连接、断链、重连
- BUG #321: testcase 中 counter 和 gpio 测试失败
- BUG #322: rf-only 带内模式，鼠标烧录异常
- BUG #325: rf-only 带内模式下，复位 dongle 后鼠标不发起画圈
- BUG #354: rf-enhance 带重传跳频模式下，RF 上报率最大 976hz，不太稳定，偶现速率掉 0
- BUG #355: rf-only 带内模式下，RF 模式下上报率最大为 960hz
- BUG #328: central_hr 和 central_ht 运行时间超过 2 分钟时，就会出现 halting system，且复位不可恢复，只能重烧
- BUG #336: peripheral 测试—软件编译正常，打印信息异常 bt_settings 未设置成功 ID,IRK,Database Hash
- BUG #337: central_ht 测试—扫描设备 peripheral_ht 后，无相应广播数据
- BUG #338: central_hr 测试—扫描广播数据异常，且与 heartrate-server app 无法连接，无心率数据上报
- BUG #339: peripheral_identity 测试—无蓝牙广播包
- BUG #349: mult_roles 测试—设备连接功能异常
- BUG #354: rf-enhance 带重传跳频模式下，RF 上报率最大 976hz，不太稳定，偶现速率掉 0（目前掉 0 问题已解）
- BUG #356: peripheral_hr 测试—低功耗 XTL32768 软件由于 build_name 过长导致编译失败
- BUG #358: ZephyrAppLauncher.exe 工具在 build、flash 后不能 open IDE
- BUG #359: ZephyrAppLauncher.exe 工具编译 usb_only_mode_test.conf 失败
- BUG #360: qdec_pwm 测试—qdec_pwm 功能异常，log 没有打印
- BUG #367: peripheral_ota 可以正常进行升级，升级异常
- BUG #368: peripheral_hr 低功耗测试——XTL32768 有广播无法连接和 RCL32000 无广播

遗留问题

- ISSUE #22: sample: mesh_speaker 在小度音响上工作异常
- BUG #301: Amazon Echo 音箱测试-Echo Plus2-2 个音箱入网之后，不做任何操作，出现 Halting system 且灯控不可操作
- BUG #309: 2.4G 发送端信号 tx 和接收端信号 rx，设置不同的频点和不同的 CRC 搭配时，成功率偏低，丢包偏多
- BUG #310: 蓝牙作为 central 时，与其它设备建立连接时，会异常断开连接
- BUG #313: mult_roles 测试—设备作为 Slave 与 3 个手机保持长时间连接出现异常，bt_hci_core，断链后无法再次重连
- BUG #316: central_multilink 测试—Central 设备可以连接从设备，但是连接不稳定，连接上会发生断链
- BUG #318: 手机 NRF Mesh 测试-烧录 mesh_panhip 的 sample，在 nRF Mesh 界面上搜索设备进行连接时，有很多名称为 panchip 的设备显示且无法连接成功

- BUG #319: Friend-LPN，多个 Friend，一个 LPN，查看 Friend 切换流程，切换到新 Friend 时，不稳定，发送开关灯命令不生效
- BUG #320: Heartbeat Publish—查看 Heart beat 的打印间隔为 4s，丢包的概率 13/39 为 33%
- BUG #323: 手机 NRF Mesh 测试-对 mesh 设备 Models 进行入网绑定 App Key 验证开关灯功能出现几点问题
- BUG #324: ble-only 模式下，压力测试蓝牙配对、连接操作，鼠标在断链后无法实现自动重连
- BUG #329: mesh_panchip 会烧录后复位板卡会出现异常，且复位后信息不会刷新，经过较久时间仍然记得上次复位次数
- BUG #337: central_ht 测试—扫描设备 peripheral_ht 后，无相应广播数据
- BUG #342: eddystone 测试—不可连接广播包测试–偶现复位板卡 30s 之后，仍然可以连接板卡现象
- BUG #344: iot_wechat 测试—Android 手机与设备连接测试 AirSync 微信蓝牙协议第二次测试时 AUTH 失败
- BUG #348: central_multilink 测试—Central 设备无法连接从设备

新增问题

- BUG #371: mult_roles 测试—配置 3 主 0 从，实际上 central 只能连接 2 个外设，第 3 个外设无法连接
- BUG #362: 跳频模式下，Mouse 和 usb dongle 均连接 usb，运行大概 4 分钟左右就会出现掉 0
- BUG #373: central_ht 测试—central 和 peripheral_ht 连接后，多次复位 peripheral_ht 会出现连接 central 和外设均异常卡着，并且不上报数据

5.2 PAN1080 DK v0.2.0

PAN1080 Development Kit v0.2.0 (2022-3-17) 已发布：

5.2.1 1. SDK

Zephyr 平台

- 移除了对 PN108C/PN108MP 两个板子的支持，新增了对 PAN1080 EVB 板的支持，请使用新的 EVB 板进行测试验证
- 增加 DCDC 控制流程，使用 DCDC 模式会降低运行功耗
- 修改 flash 工作模式为 4 线 enhance 模式，加速运行速度
- 增加 private radio 通用库

协议栈

- 更新了 libble_controller，更新射频配置, 提高接收性能
- 适配低功耗流程

例程

- 问题修复：
  - samples: mult_roles: 解决 central 只能连接一次的问题
- 优化：
  - quick build 脚本已更新，且 sample 文档中的编译不再使用 west build 命令，而是以 quick build 脚本为基础进行说明
  - samples_panchip/bluetooth: Use public addr for Mesh samples
  - solutions/mouse: prf-only mode 下新增 private radio 增强型工作流程，并默认关闭 Log
  - solutions/mouse: prf-only mode 下新增 private radio 调频工作流程，并默认关闭 Log

测试用例

无

工具

- 新增 Zephyr App Launcher 可视化开发工具

5.2.2 2. DOC

- 新增文档:
  - documentation/01_quick_start: 新增 zephyr_introduction 文档
  - documentation/04_dev_guides: 新增 app launcher 工具介绍及 faq
  - documentation/02_hardware: 新增硬件介绍文档
  - 新增 EVB 底板、核心板设计图及相关设计文档
- 更新文档:
  - `solutions/mouse_solution`: 更新测试流程，使用 quick build 脚本代替编译命令
  - samples_panchip/bluetooth: 更新文档格式及图片，新增了蓝牙例程的 quick build 脚本
  - samples_panchip/basic: 更新文档格式及图片，新增了基础例程的 quick build 脚本
  - samples_panchip/proprietary_radio: 更新测试流程，新增了 private radio 例程的 quick build 脚本
  - 其他: 修正文档中的图片显示、描述及格式

5.2.3 3. ISSUES

已解决问题

- `ISSUE #152`: 解决 settings 功能异常
- `BUG #151`: 系统 64M 下 clktrim 校准失败
- `BUG #343`: hci_uart 在一些场景下工作异常
- `BUG #287`: central_ht 测试—查看设备串口显示的从机温湿度信息

遗留问题

- ISSUE #22: sample: mesh_speaker 在小度音响上工作异常
- ISSUE #27: sample: mouse_solution 例程在一些设备上工作异常
- BUG #288: central 测试—Central 设备主动扫描功能，扫描到对端设备连接之后立即断链
- BUG #289: 蓝牙多角色功能实际支持的连接个数比配置的个数少一个
- BUG #290: peripheral_identity 测试—测试设备支持的最大连接个数时，出现设备发不出广播的情况
- BUG #299: peripheral_hids 测试—多次进行连接、测试、重连操作、会出现断链后但是设备仍然处于连接状态
- BUG #309: 2.4G 发送端信号 tx 和接收端信号 rx，设置不同的频点和不同的 CRC 搭配时，成功率偏低，丢包偏多
- BUG #310: 蓝牙作为 central 时，与其它设备建立连接时，会异常断开连接
- mesh dev UUID 多设备显示为同一个，多设备烧录需要代码内修改
- BUG #337: central_ht 测试—扫描设备 peripheral_ht 后，无相应广播数据
- BUG #312: mult_roles 测试—配置 3 主 0 从-设备作为 Master，验证设备可以支持的最大链路数失败，建立一条链路，建立第二条时，会强制把第一条链路断开
- BUG #313: mult_roles 测试—设备作为 Slave 与 3 个手机保持长时间连接出现异常，bt_hci_core，断链后无法再次重连
- BUG #314: peripheral 测试—连接、断链、重连测试中，出现连接一次成功后，第 2 次连接异常，无法再次连接现象
- BUG #315: prf-2.4g 测试—点对点通信，过一段时间会偶现接收端 rx data crc err
- BUG #316: central_multilink 测试—Central 设备可以连接从设备，但是连接不稳定，连接上会发生断链
- BUG #317: peripheral_hids 测试—苹果手机 iphone6s 和华为 nova 手机均不能与设备进行正常连接、断链、重连
- BUG #318: 手机 NRF Mesh 测试-烧录 mesh_panhip 的 sample，在 nRF Mesh 界面上搜索设备进行连接时，有很多名称为 panchip 的设备显示且无法连接成功
- BUG #319: Friend-LPN，多个 Friend，一个 LPN，查看 Friend 切换流程，切换到新 Friend 时，不稳定，发送开关灯命令不生效
- BUG #320: Heartbeat Publish—查看 Heart beat 的打印间隔为 4s，丢包的概率 13/39 为 33%
- BUG #321: testcase 中 counter 和 gpio 测试失败
- BUG #323: 手机 NRF Mesh 测试-对 mesh 设备 Models 进行入网绑定 App Key 验证开关灯功能出现几点问题
- BUG #324: ble-only 模式下，压力测试蓝牙配对、连接操作，鼠标在断链后无法实现自动重连
- BUG #328: central_hr 和 central_ht 运行时间超过 2 分钟时，就会出现 halting system，且复位不可恢复，只能重烧
- BUG #329: mesh_panchip 会烧录后复位板卡会出现异常，且复位后信息不会刷新，经过较久时间仍然记得上次复位次数
- BUG #338 central_hr 测试—扫描广播数据异常，且与 heartrate-server app 无法连接，无心率数据上报
- BUG #339 peripheral_identity 测试—无蓝牙广播包
- BUG #342 eddystone 测试—不可连接广播包测试–偶现复位板卡 30s 之后，仍然可以连接板卡现象
- BUG #344 iot_wechat 测试—Android 手机与设备连接测试 AirSync 微信蓝牙协议第二次测试时 AUTH 失败

- `BUG #348` central_multilink 测试—Central 设备无法连接从设备
- `BUG #349` mult_roles 测试—设备连接功能异常

新增问题

- `BUG #356` peripheral_hr 测试—低功耗 XTL32768 软件由于 build_name 过长导致编译失败
- `BUG #358` ZephyrAppLauncher.exe 工具在 build、flash 后不能 open IDE
- `BUG #359` ZephyrAppLauncher.exe 工具编译 usb_only_mode_test.conf 失败
- `BUG #354` rf-enhance 带重传跳频模式下，RF 上报率最大 976hz，不太稳定，偶现速率掉 0（目前掉 0 问题已解）

5.3 PAN1080 DK v0.1.0

PAN1080 Development Kit `v0.1.0` (2022-1-20) 已发布：

5.3.1 1. SDK

Zephyr 平台

- 新增低功耗流程 (HW Deep Sleep)
- 修改 `mcuboot`，支持通过 PAN1080 完成跳转升级流程，验证签名升级流程
- 删除一些无用的文件

协议栈

- 修改 LL ISR 初始化方式，支持 BLE、PR 2.4G 动态切换（双模切换的 sample 还没有）
- 修改 BT_COMPANY_ID
- 修改一些默认 Stack 大小
- 支持通过 Kconfig 配置 Sleep Clock，统一 libble
- 支持通过 Kconfig 配置 libble 支持的并发状态机个数，优化 SRAM
- 修改 API: bt_static_address_init: Bluetooth Device Address 高 2 bits 强制配为 b00
- 修复某些安卓 BLE4.0 不支持 DLE 功能，当不开 DLE 时，l2cap len 异常导致入网时失败的问题
- 修复 mesh pb remote 指定 uuid 设备连接的问题（sdk V0.0 为方便测试做的限制）

例程

- 问题修复：
  - samples: mult_roles: 解决 central 只能连接一次的问题
  - samples: ble_hid_selfie/peripheral_hids: 解决配对绑定后重连的问题
  - samples: ble_hid_selfie: iphone 手机 hid 无响应问题
  - samples: audio: 修改噪声问题
- 优化：
  - mesh_echo: 修正打印 log 与注释
  - mesh_provisioner: `prj.conf` 文件增加描述

  - mesh_speaker: `prj.conf` 内默认关闭 LOG，关闭 friend feature
  - mesh_panchip: `prj.conf` 内：
    * 默认开启 LOG，默认关闭 SIG OTA，增加描述，增加 Stack 调试宏
    * IO DTS 灯控修正为可读
    * 优化 stack 大小，为后续开发提供足够 RAM
  - solutions/mouse: usb-only mode 下新增 usb remote wakeup flow，并默认开启 Log
- 新增例程:
  - proprietary_radio/prf_sample_rx: 2.4G 收基础例程
  - proprietary_radio/prf_sample_tx: 2.4G 发基础例程
  - bluetooth/mesh_demo: 演示自组网例程, adv 情况下的 Feature 展示（Friend-LPN, heartbeat publish）
  - hello_world: 简单串口打印例程，可以通过 with 或者 without mcuboot 的方式进行启动，展示版本信息

测试用例

- 新增用例：
  - drivers:
    * adc/adc_api: 测试 Zephyr ADC API
    * counter/counter_basic_api: 测试 Zephyr Counter API
    * flash: 测试 Zephyr Flash 基本读写擦 API
    * gpio/gpio_api_1pin: 测试 Zephyr GPIO API，只使用 1 根 pin 脚
    * gpio/gpio_basic_api: 测试 Zephyr GPIO API，使用 2 根 pin 脚对测
    * pwm/pwm_api: 测试 Zephyr PWM API
    * uart/uart_basic_api: 测试 Zephyr UART API
  - subsys/pm:
    * power_mgmt_soc: 测试 Zephyr Power Management（电源管理/低功耗）功能
    * power_states_api: 测试 Zephyr Power States API
  - subsys/settings:
    * settings/functional/nvs: 测试 Zephyr NVS 基本功能
    * settings/nvs/raw: 测试 Zephyr Settings 功能（使用 NVS 作为 Backend）
  - subsys/storage:
    * flash_map: 测试 Zephyr Flash Map API
    * stream/stream_flash: 测试 Zephyr Flash Stream API

5.3.2 2. DOC

- 新增文档:
  - `proprietary_radio/prf_sample_rx`: 新增, 2.4G 收发例程
  - `proprietary_radio/prf_sample_tx`: 新增, 2.4G 收发例程
  - `dev_guides/Zephyr Mcuboot Guidance.md`: 新增作为 mcuboot 的应用操作指南

  - dev_guides/Zephyr RAM Analysis Guidance.md: 新增作为 RAM 相关分析操作指南文档
  - samples/bluetooth/mesh_demo.md: 新增作为测试自组网，验证 adv feature（Friend-LPN,heartbeat）的文档
- 更新文档:
  - proprietary_radio/prf_io_pulse_rx: 修改了代码说明和配置说明等
  - proprietary_radio/prf_io_pulse_tx: 修改了代码说明和配置说明等
  - solutions/mouse_solution: 更新测试流程
  - dev_guides/Developing Bluetooth Mesh Applications.md: 更新增加补充了配置原理方面的细节，完善作为 Mesh 用户指南
  - samples/bluetooth/mesh_speaker.md: 更新文档描述，补充 log 截图
  - samples/bluetooth/mesh_echo.md: 更正文档描述
  - samples/solution/mesh_panchip.md: 更新文档描述，修正 DK0.0 反馈的文档错误
  - Developing Bluetooth Applications: 修改了静态地址相关的说明和参考代码，补充了 GATT Service 的配置和说明

5.3.3 3. ISSUES

已解决问题

- ISSUE #19: 静态地址设置失败
- ISSUE #23: sample: mesh_echo 在某些手机上工作异常
- ISSUE #24: sample: ble_hid_selfie 例程重连和控制异常
- ISSUE #25: proprietary_radio 与 xn24l01 通信时丢包
- ISSUE #26: sample: iot_wechat 例程测试不通
- BUG #294: sample: mult_roles 作为 central 与其它设备建立连接时，只能连接一次

遗留问题

- ISSUE #22: sample: mesh_speaker 在小度音响上工作异常
- ISSUE #27: sample: mouse_solution 例程在一些设备上工作异常
- BUG #286: **settings 可能存在问题，烧录不同的例程时，建议先对芯片进行擦除**
- BUG #287: central_ht 测试—查看设备串口显示的从机温湿度信息
- BUG #288: central 测试—Central 设备主动扫描功能，扫描到对端设备连接之后立即断链
- BUG #289: 蓝牙多角色功能实际支持的连接个数比配置的个数少一个
- BUG #290: peripheral_identity 测试—测试设备支持的最大连接个数时，出现设备发不出广播的情况
- BUG #299: peripheral_hids 测试—多次进行连接、测试、重连操作、会出现断链后但是设备仍然处于连接状态
- BUG #309: 2.4G 发送端信号 tx 和接收端信号 rx，设置不同的频点和不同的 CRC 搭配时，成功率偏低，丢包偏多
- BUG #310: 蓝牙作为 central 时，与其它设备建立连接时，会异常断开连接
- BUG #324: ble-only 模式下，压力测试蓝牙配对、连接操作，鼠标在断链后无法实现自动重连
- mesh dev UUID 多设备显示为同一个，多设备烧录需要代码内修改

新增问题

- BUG #312: mult_roles 测试—配置 3 主 0 从-设备作为 Master，验证设备可以支持的最大链路数失败，建立一条链路，建立第二条时，会强制把第一条链路断开
- BUG #313: mult_roles 测试—设备作为 Slave 与 3 个手机保持长时间连接出现异常，bt_hci_core，断链后无法再次重连
- BUG #314: peripheral 测试—连接、断链、重连测试中，出现连接一次成功后，第 2 次连接异常，无法再次连接现象
- BUG #315: prf-2.4g 测试—点对点通信，过一段时间会偶现接收端 rx data crc err
- BUG #316: central_multilink 测试—Central 设备可以连接从设备，但是连接不稳定，连接上会发生断链
- BUG #317: peripheral_hids 测试—苹果手机 iphone6s 和华为 nova 手机均不能与设备进行正常连接、断链、重连
- BUG #318: 手机 NRF Mesh 测试-烧录 mesh_panhip 的 sample，在 nRF Mesh 界面上搜索设备进行连接时，有很多名称为 panchip 的设备显示且无法连接成功
- BUG #319: Friend-LPN，多个 Friend，一个 LPN，查看 Friend 切换流程，切换到新 Friend 时，不稳定，发送开关灯命令不生效
- BUG #320: Heartbeat Publish—查看 Heart beat 的打印间隔为 4s，丢包的概率 13/39 为 33%
- BUG #321: testcase 中 counter 和 gpio 测试失败
- BUG #323: 手机 NRF Mesh 测试-对 mesh 设备 Models 进行入网绑定 App Key 验证开关灯功能出现几点问题
- BUG #328: central_hr 和 central_ht 运行时间超过 2 分钟时，就会出现 halting system，且复位不可恢复，只能重烧
- BUG #329: mesh_panchip 会烧录后复位板卡会出现异常，且复位后信息不会刷新，经过较久时间仍然记得上次复位次数

5.4 PAN1080 DK v0.0.0

PAN1080 Development Kit v0.0.0 (2021-12-17) 已发布：

5.4.1 1. SDK

- 基于开源操作系统 Zephyr 的软件集成开发环境（SDK CLI + IDE）
- Panchip 为 PAN1080 定制的交叉编译工具链（Toolchain）
- 支持 PN108C QFN40 测试板
- Panchip 为 PAN1080 提供了多个演示功能的 APP 例程：

基础例程

- blinky: 演示 LED 闪灯，led0 以一秒为周期亮灭。

蓝牙例程

- audio_client: 演示蓝牙语音传输主机端。
- audio_server: 演示蓝牙语音传输从机端。
- beacon: 演示蓝牙 beacon。

- central: 演示蓝牙主机功能，发现设备并与设备建立连接和断连。
- central_hr: 演示蓝牙主机功能，主要是 HR(heart-rate) 服务相关，包括：发现设备, 解析广播数据并与包含 HR 服务的设备建立连接；查找并订阅 HR 服务。
- central_ht: 演示蓝牙主机功能，主要是 HT(health thermometer) 服务相关，包括：发现设备, 解析广播数据并与包含 HT 服务的设备建立连接；查找并订阅 HT 服务。
- central_multilink: 演示主机多连接功能，可以发现设备并与最多 8 个从机设备建立连接。
- eddystone: 演示 Google Eddystone Configuration Service 和 Eddystone beacon 功能。
- hci_uart: 单 BLE Controller，可以通过串口发送 HCI 命令，用于 DTM，或者配合外部 Host 使用。
- ibeacon: 演示 Apple iBeacon 功能，在支持 iBeacon 的应用上，可以粗略的显示距离信息。
- iot_wechat: 演示微信硬件开发平台的 AirSync 协议。
- mesh_echo: 演示蓝牙 MESH 功能，可与 Google ECHO 音响进行绑定，并进行开关灯控制。
- mesh_provisioner: 演示蓝牙 MESH Provisioner 功能，先进行自我配置 netkey，存储，然后通过 pb adv 对其他待入网设备广播进行扫描，建立 link，入网的流程，并包括后续的配置流程。
- mesh_speaker: 演示蓝牙 MESH 功能，可与天猫精灵、百度小度音响进行绑定，并进行开关灯控制。
- multi_roles: 演示蓝牙多角色（主从一体）功能，可以通过 shell 进行广播、扫描和连接，支持最多 8 组连接。
- peripheral: 演示蓝牙从机功能，包含 GATT 服务: CTS/BAS/HRS。
- peripheral_csc: 演示蓝牙从机功能，包含 GATT 服务：CSC (Cycling Speed and Cadence)。
- peripheral_dis: 演示蓝牙从机功能，包含 GATT 服务: DIS (Device Information) 。
- peripheral_esp: 演示蓝牙从机功能，包含 GATT 服务: ESP (Environmental Sensing Profile) 。
- peripheral_hids: 演示蓝牙从机功能，包含 GATT 服务：HID，通用鼠标。
- 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。
- peripheral_ht: 演示蓝牙从机功能，包含 GATT 服务：HT (Health Thermometer)，连接订阅服务后，会上报虚拟的温度数据。
- peripheral_identity: 演示从机多连接功能，可以与最多 8 个主机设备建立连接。
- scan_adv: 演示蓝牙广播和扫描功能，将扫描到的设备个数，放在特定的广播数据中发出去。

私有 2.4G 例程

- prf_io_pulse_rx: 演示脉冲传输功能，接收发送端的 2.4G 信号，并恢复出波形，通过 IO 口输出。
- prf_io_pulse_tx: 演示脉冲传输功能，通过 IO 口接收将外部的 PWM 波形，并通过 2.4G 传输给接收端设备。

解决方案

- ble_google_light: 谷歌灯解决方案，通过谷歌音响控制灯。
- ble_hid_selfie: 自拍解决方案，通过蓝牙 HID 控制手机拍照。
- ble_pcte_beacon: 磐启定位标签解决方案，通过广播发送特定的定位数据。
- ble_rgb_light: 蓝牙 RGB 三色灯解决方案，可以用小程序连接并进行控制。
- mesh_panchip: 蓝牙 MESH 解决方案，支持远程入网，OTA 灯。
- mouse_solution: 鼠标解决方案，支持 BLE、2.4G、USB 三中模式。

- `usb_dongle`: dongle 解决方案，支持 BLE、2.4G。

其它

- `synchronization`: 演示内核基本功能，系统启动两个线程交替向 `Console` 打印消息。

5.4.2 2. HDK

目前版本提供了如下硬件相关资料：

- PAN1080 QFN32 开发参考原理图
- PN108C QFN40 测试板原理图

5.4.3 3. DOC

目前版本提供了如下开发文档：

- SDK 快速入门
- SDK 开发环境介绍
- SDK 架构介绍
- PN108C QFN40 测试板介绍
- 例程介绍
- Zephyr APP 开发指南
- BLE 开发指南
- Mesh 开发指南
- PAN1080 产品说明书（中文）
- PAN1080 Datasheet（EN）

5.4.4 4. TOOLS

目前版本提供了如下工具：

- 量产烧录工具（PC 工具）
- 鼠标上报率测试工具（PC 工具）
- 串口工具（PC 工具）
- nRF Connect（手机测试软件安卓 APK）
- Google Home（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）

5.4.5 5. 已知问题

- `ble_rgb_light` 与 `iot_wechat` 两个例程中，广播数据中的地址字段未匹配，可能会影响扫描
- `Beacon`：抓包工具查看广播间隔为 100ms-150ms，但是实际抓包间隔小于 100 且不断变化
- `central_hr`：主设备可以与从设备正常进行连接，但是没有心率数据上报
- `central_ht`：查看设备串口显示的从机温湿度信息

- `central`：Central 设备主动扫描功能，扫描到对端设备连接之后立即断链
- `peripheral_identity`：测试设备支持的最大连接个数时，测试过程中链路非正常断链
- `peripheral_identity`：测试设备支持的最大连接个数时，出现设备发不出广播的情况
- `mult_roles`：设备连接功能正常，ble conn create 某个设备后会立即断链，然后剩下的可连接设备均无法连接成功
- `mult_roles`：配置 0 主 8 从-设备作为 Slave，验证设备可以支持的最大链路数 8 个
- `peripheral_hids`：多次进行连接、测试、重连操作、会出现断链后但是设备仍然处于连接状态
- `mesh_panchip` 直接编译无法通过（原因是此 Sample 只支持 1MB 大小 Flash 的测试板，不支持在 512KB 的 QFN40 测试板上运行）
- 其他：芯片 RF 参数未经调优，可能会有性能问题