ls_sdk

linkedsemi

2022 年 02 月 22 日

Contents:

This document site is **deprecated**.

Please visit the new site Linkedsemi Documentation Center

CHAPTER 1

入门指南

1.1 软件开发环境搭建

我们支持如下开发环境：

1. Python 3 + MDK-KEIL
2. Python 3 + VS Code + GCC(ARM)

下载地址：

GCC(ARM) 9.2.1 20191025

VS Code (64bit)

Python 3.8.2 (64bit)

MDK_KEIL

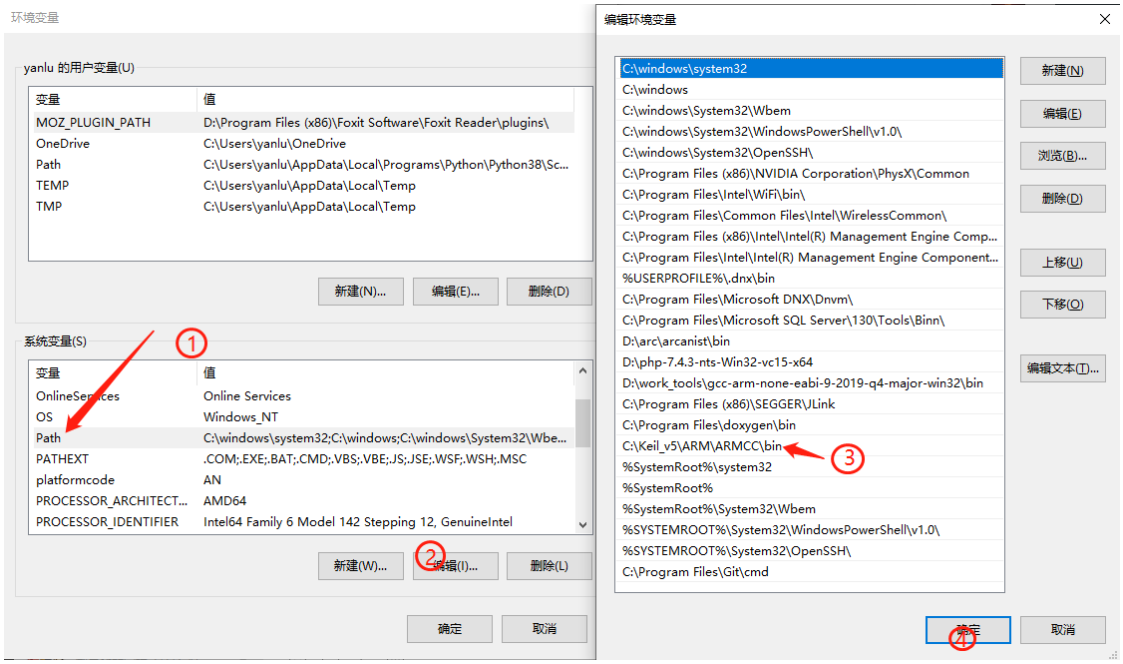
SDK 下载链接：

gitee

github

1.1.1 一、MDK-KEIL

1. keil 版本需要安装 5.25 以上, 或者直接使用我们链接提供的版本，**不建议使用 5.29 以上版本**
2. 打开下载的 ls_ble_sdk 目录，将当前目录下的 `tools\prog\LinkedSemi\le501x_flash_algo.elf` 文件复制一份并修改文件名为 le501x_flash_algo.FLM, 并将 le501x_flash_algo.FLM 文件拷贝到 keil 安装目录下的 `ARM\Flash` 路径中
3. 将 fromelf 执行文件的所在路径添加系统环境变量中，重启 keil 生效，该文件所在路径在 keil 的安装目录下面 `Keil_v5\ARM\ARMCC\bin` ，否则在使用 keil 编译时会报“fromelf 不是内部或外部命令，也不是可运行的程序或批处理文件”的警告。(如何设置添加环境变量)

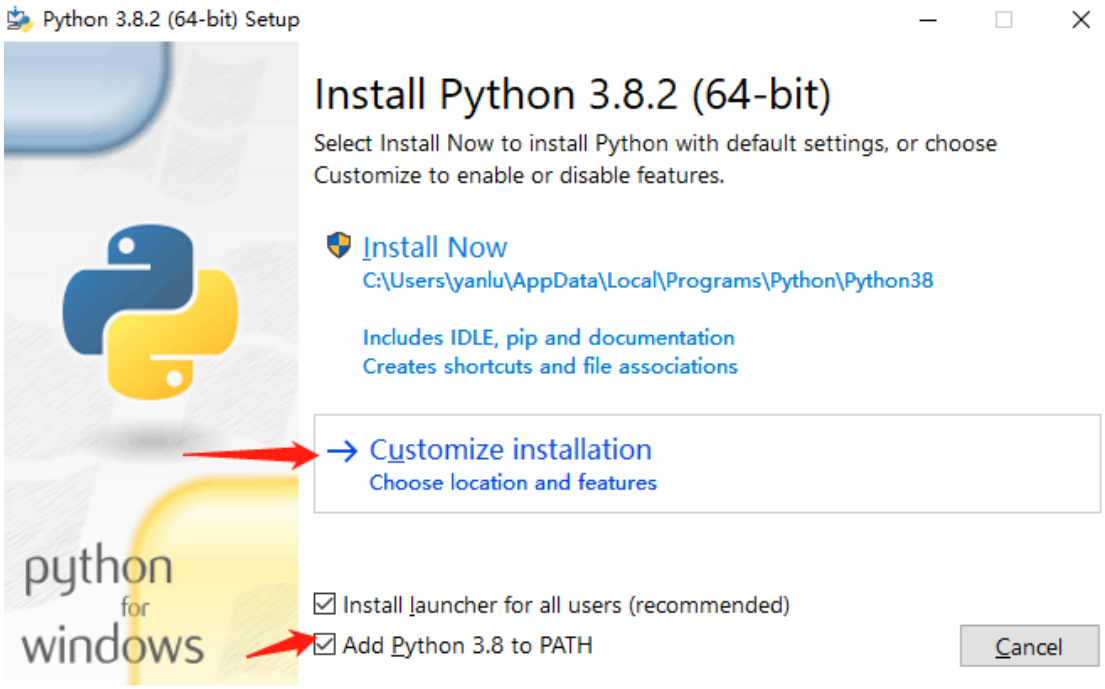


1.1.2 二、Python 3 + VS Code + GCC(ARM)

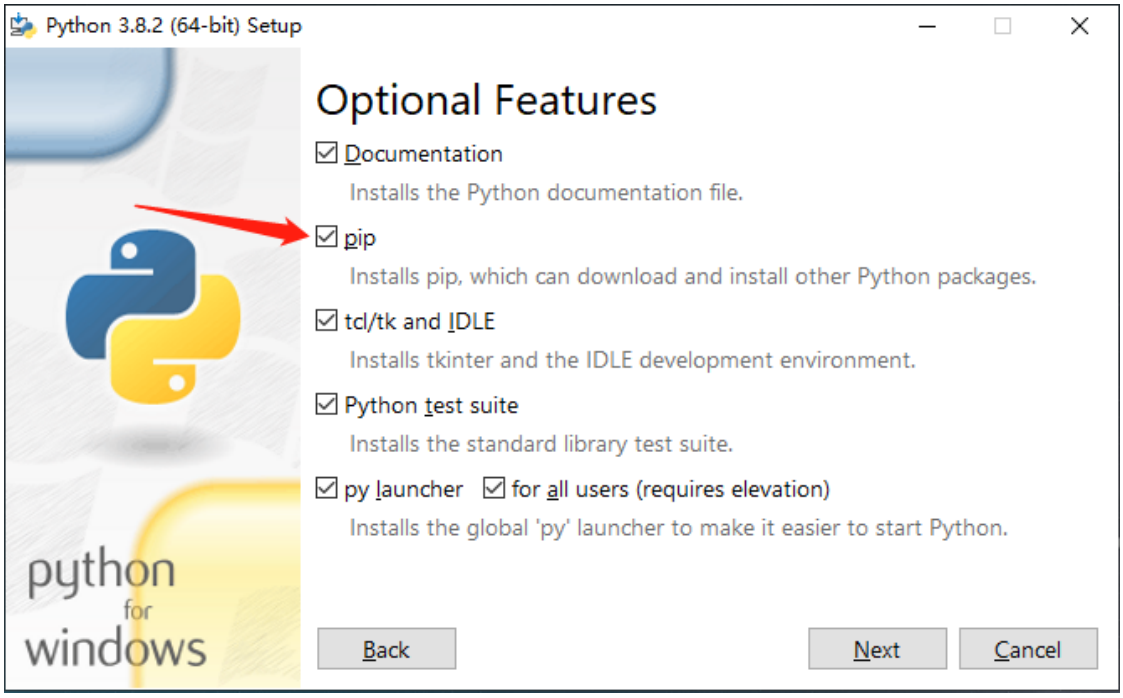
- 解压 GCC，将 `{GCC_SETUP_DIR}/bin` 路径添加到系统环境变量 PATH 中
- 安装 Python 3(勾选安装 pip 模块、添加 Python 到 PATH)、VS Code

安装 Python 相关详细说明

1. 勾选 Add Python 3.8 to PATH，然后选择 Customize install 开始安装

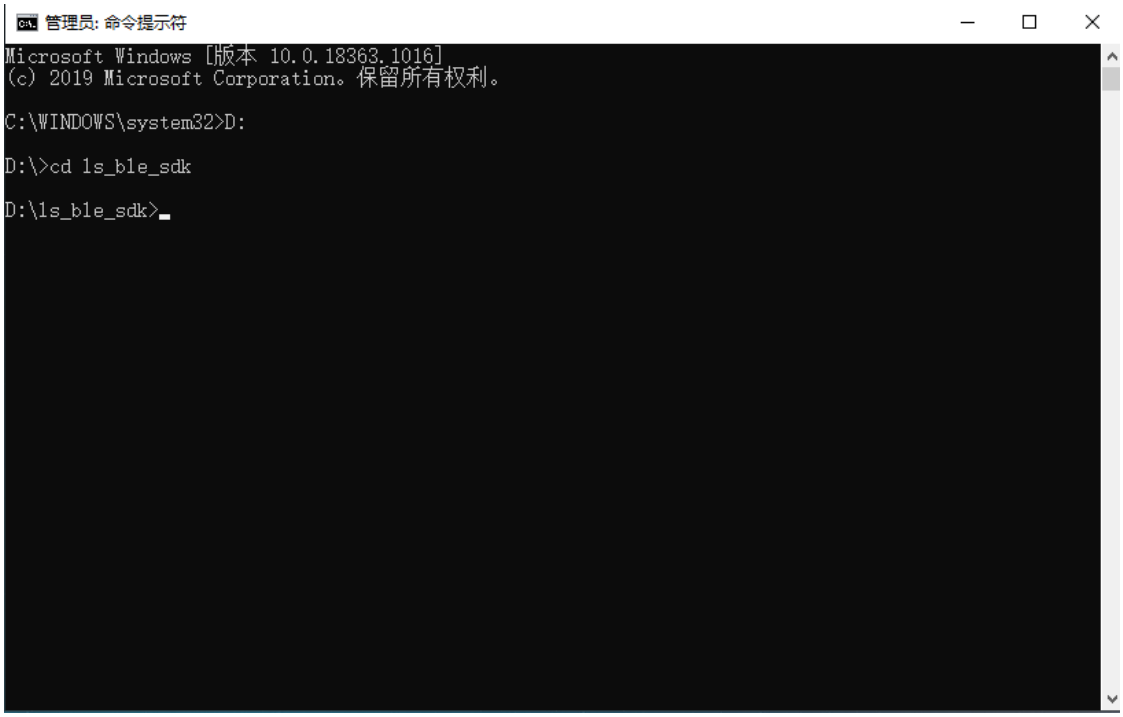


2. 确认 pip 模块已经被勾选，其它配置默认就行，然后点击 Next 进行下一步



3. 点击 Install 等待安装完成

4. 以系统管理员身份打开命令行，然后切换到 SDK 根下目录，例如 SDK 下载在 D 盘下 `D:\ls_ble_sdk`，首先在 CMD 界面内输入 `D:` 切换到 D 盘，然后再使用 `cd ls_ble_sdk` 进入到 SDK 根目录里



- 进入 SDK 根目录，以系统管理员身份打开命令行，执行下述命令，安装 Python 依赖库:

```
pip install -r requirements.txt
```

如果下载速度比较慢导致下载失败，请尝试更换国内 pip 源，或者使用以下命令:

```
pip install -r requirements.txt -i https://pypi.tuna.tsinghua.edu.cn/simple
```

- 用 VS Code 打开 SDK 目录，点击左侧 Extensions（快捷键 `Ctrl + Shift + X`），在搜索框输入 `@recommended`，安装所有 Workspace Recommendations 插件

1.1.3 三、离线安装 Python3+VS Code+GCC(ARM)

1.1.4 安装工具

python 相关的工具

- python3

  python-3.8.5.exe

- Scons

  SCons-4.1.0.post1-py3-none-any.whl

- ecdsa

  ecdsa-0.16.1-py2.py3-none-any.whl

- intelhex

  intelhex-2.3.0-py2.py3-none-any.whl

VSCODE 相关工具

- vscode 工具

  VSCodeUserSetup-x64-1.55.2.exe

- vscode 插件
  - arm 插件

    dan-c-underwood.arm-1.5.0.vsix

  - Cortex_Debug 插件

    marus25.cortex-debug-0.3.12.vsix

  - C/C++ 插件

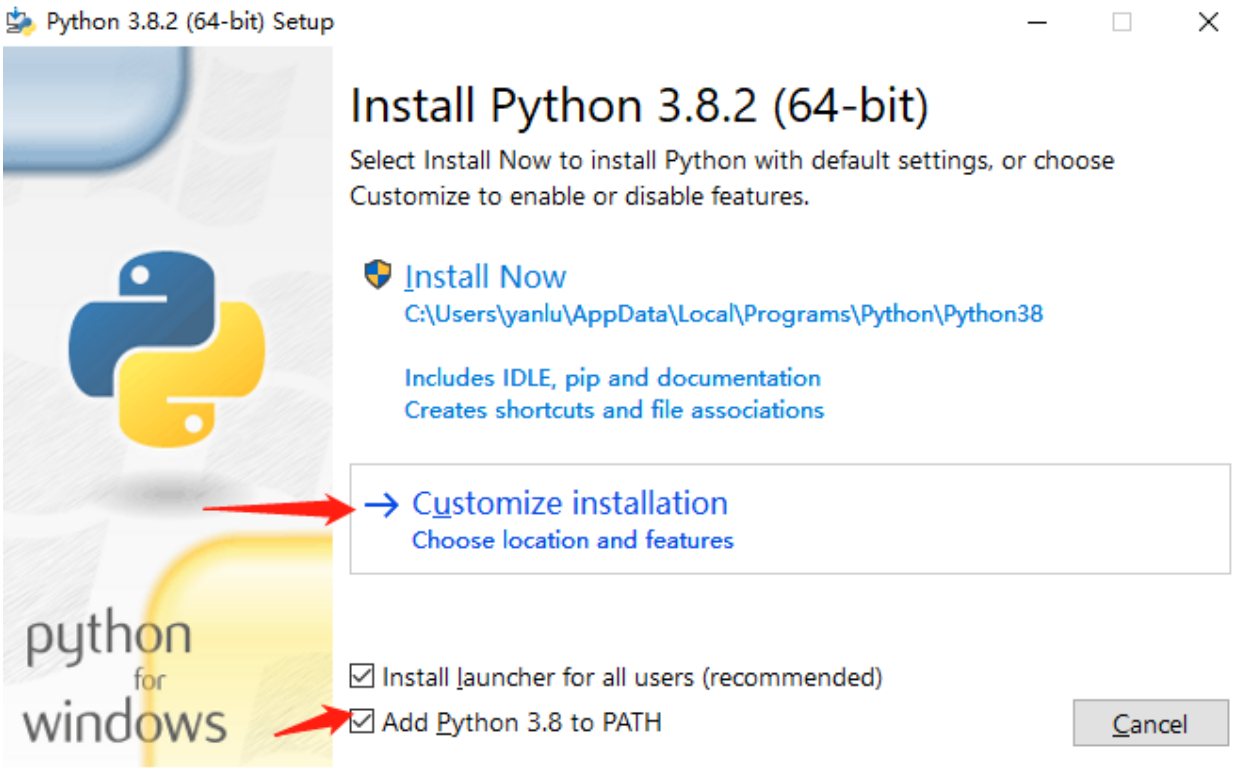
    ms-vscode.cpptools-1.3.1.vsix

  - 链接脚本插件

    ZixuanWang.linkerscript-1.0.1.vsix

1.1.5 安装步骤

1. 安装 python3



安装过程中建议勾选 Add Python 3.8 to PATH，省去后续系统环境的添加。后续一直 next，直到安装完成。

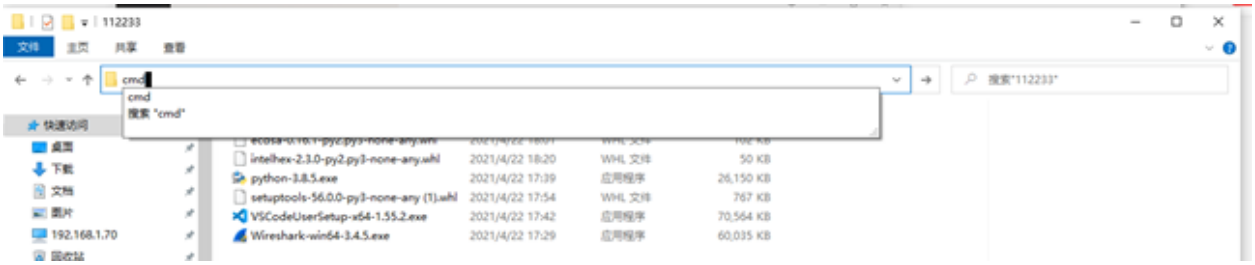
2. 检测 python 安装是否成功

上图表示，安装 Python3 已成功。

上图表示，安装 pip 已成功。

3. 安装 python3 插件

- 在插件所在的文件夹下，cmd 命令



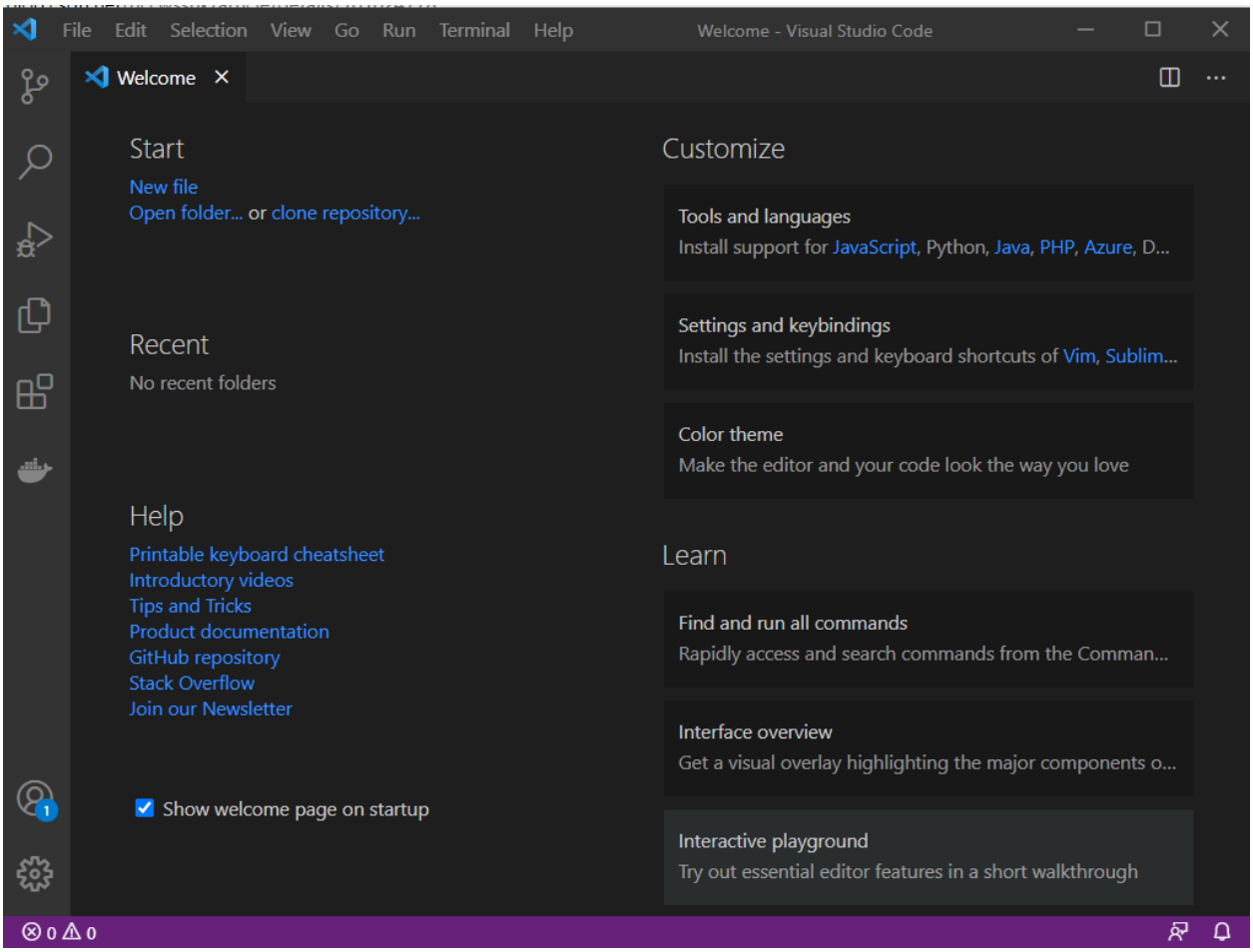
- pip 分别安装以下插件

```
12 ls
13 pip install .\ecdsa-0.16.1-py2.py3-none-any.whl
14 pip install .\intelhex-2.3.0-py2.py3-none-any.whl
15 pip install .\SCons-4.1.0.post1-py3-none-any.whl
16 pwd


PS C:\tools\112233>
```

4. vscode 安装

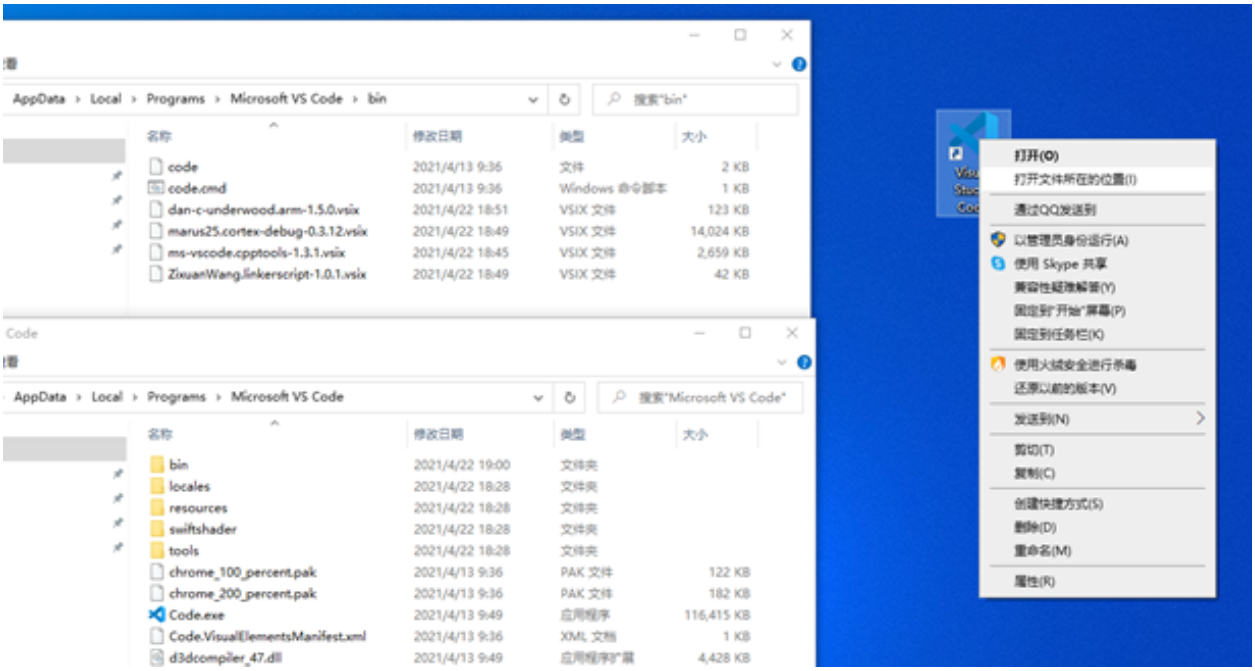
直接运行安装 VSCodeUserSetup-x64-1.55.2.exe

安装完成界面

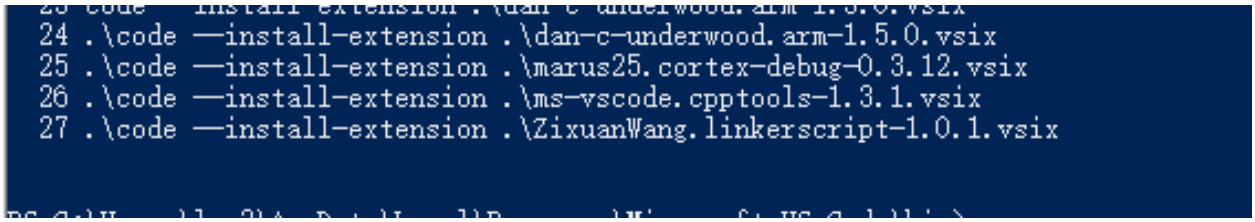


5. 安装 vscode 插件

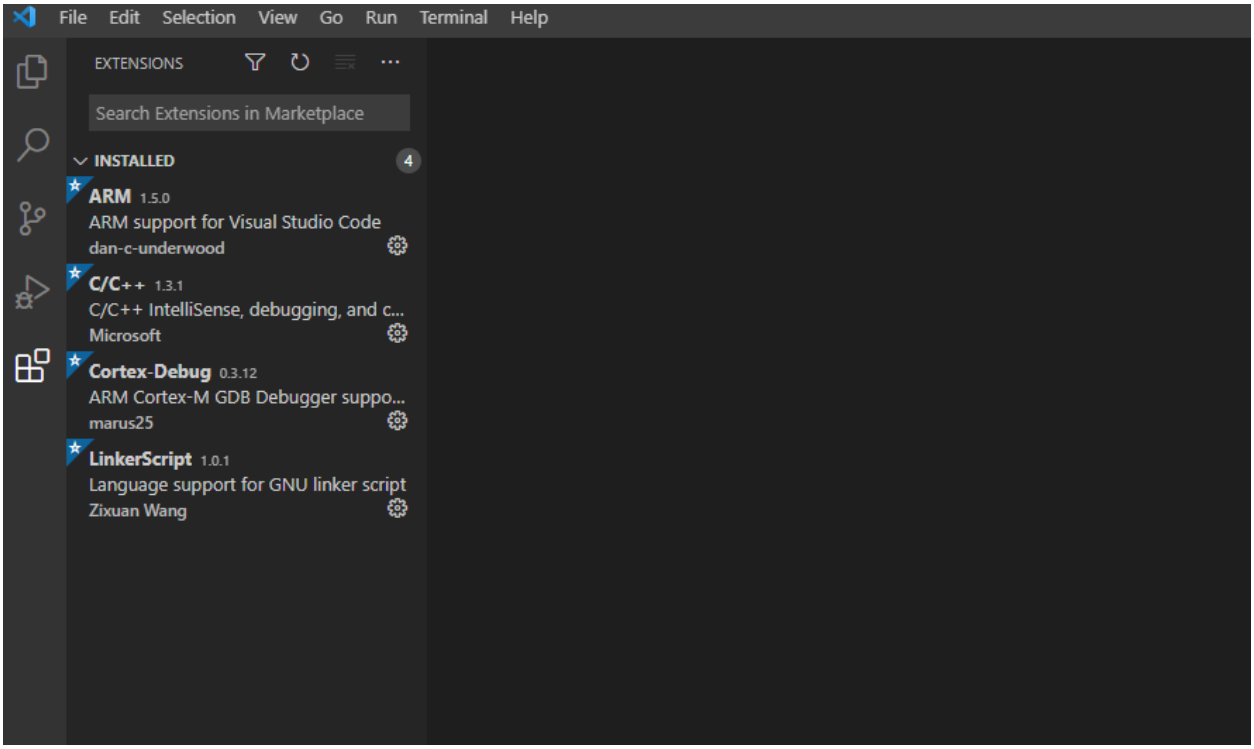
- 找到 Vscode 安装目录，把 vscode 插件全部复制到 vscode 安装目录下的 bin 文件夹下。



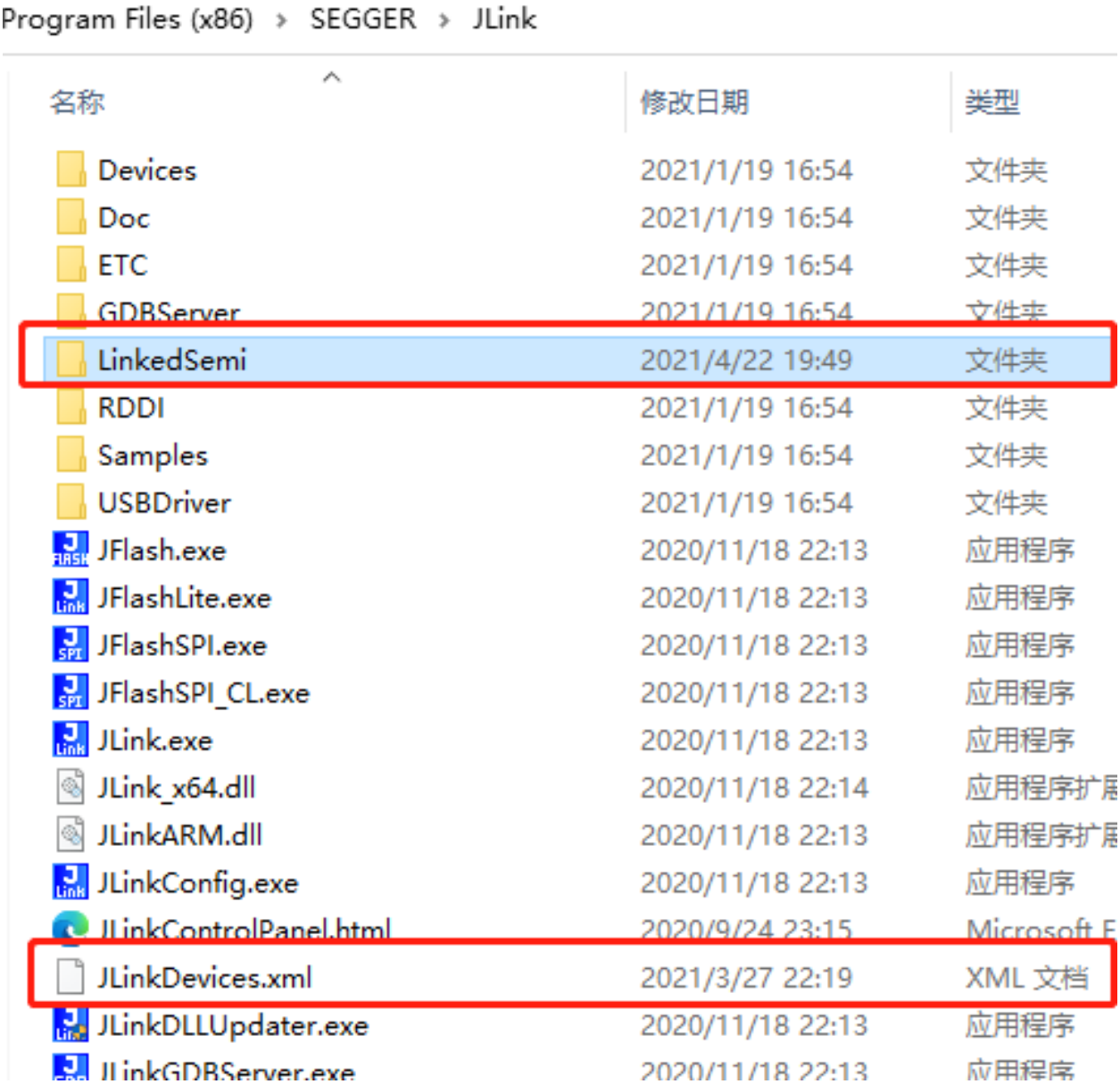
- 在插件所在的文件夹下，cmd 命令



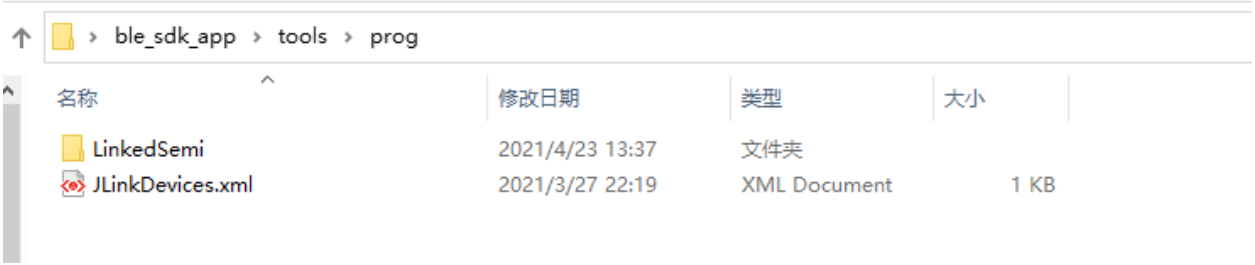
- 显示插件安装成功


File Edit Selection View Go Run Terminal Help
EXTENSIONS
Search Extensions in Marketplace
INSTALLED 4
ARM 1.5.0
ARM support for Visual Studio Code
dan-c-underwood
C/C++ 1.3.1
C/C++ IntelliSense, debugging, and c...
Microsoft
Cortex-Debug 0.3.12
ARM Cortex-M GDB Debugger suppo...
marus25
LinkerScript 1.0.1
Language support for GNU linker script
Zixuan Wang

6. JLink 安装目录添加 Linkedsemi 相关文件

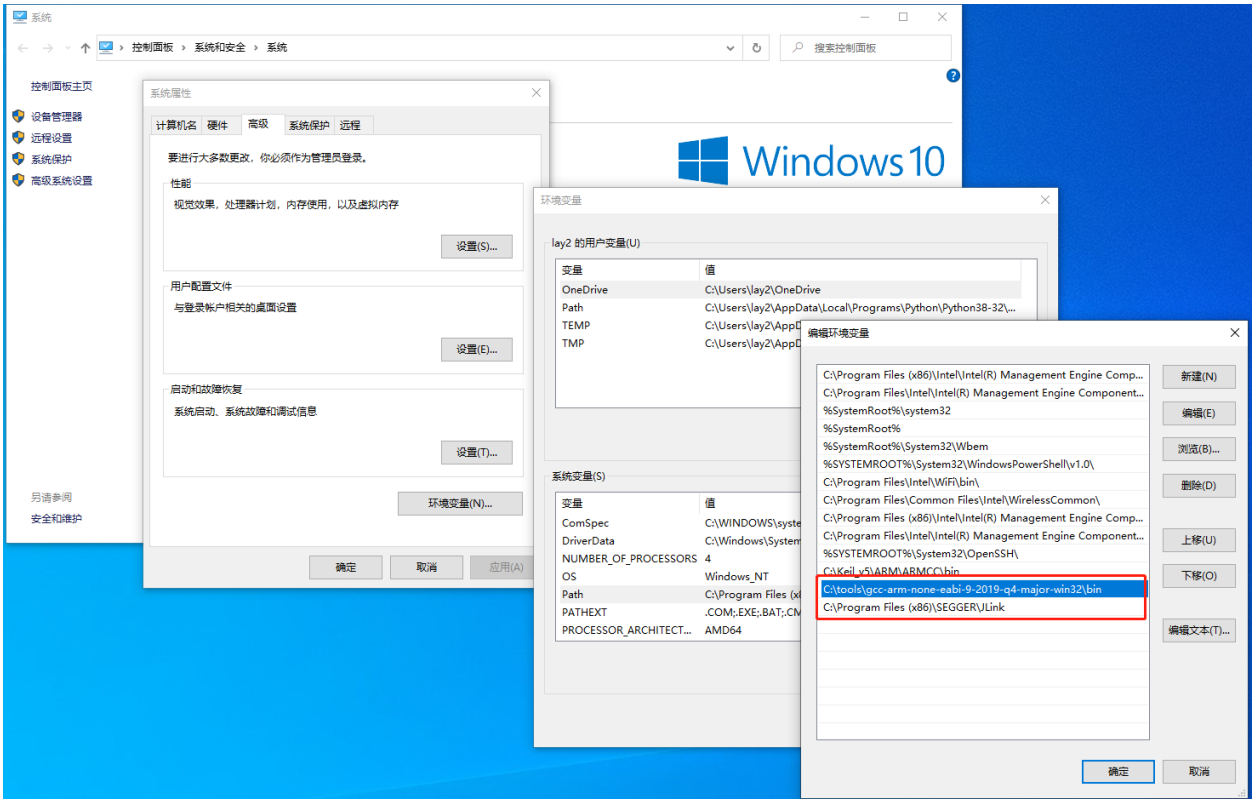


其中 LinkedSemi 文件夹和 JLinkDevices.xml 在 SDK 目录，可以复制添加到 JLink 目录下。



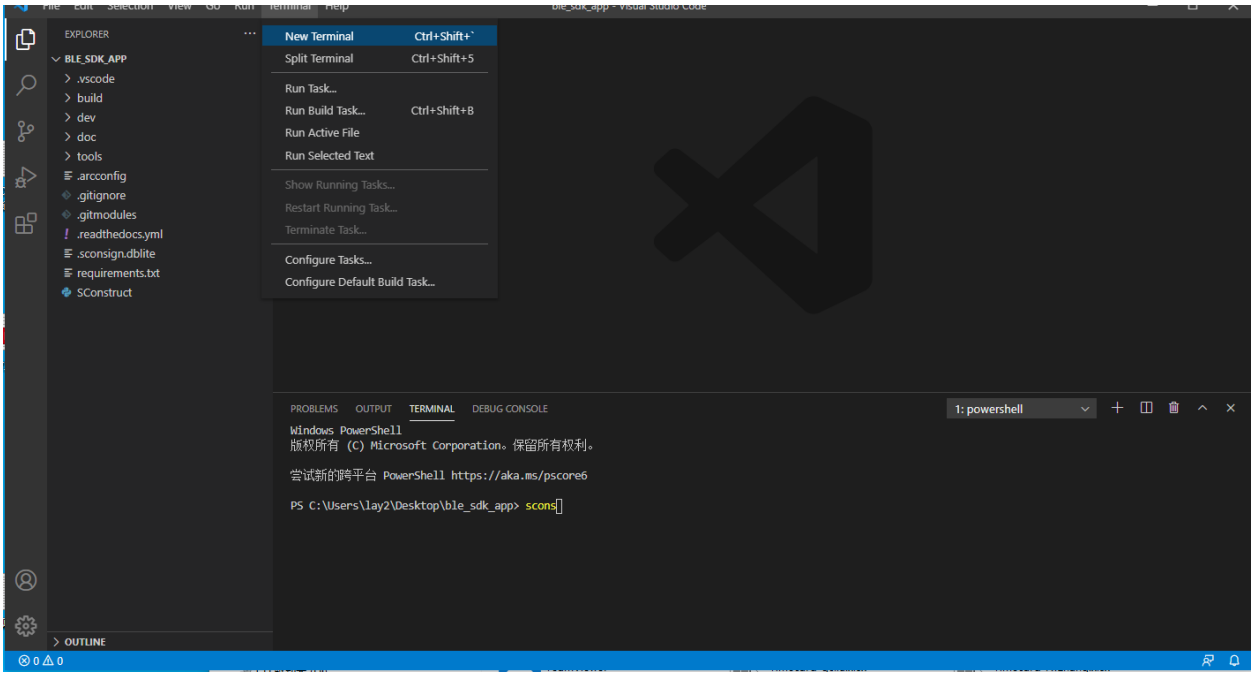
7. 添加 gcc-arm 环境变量和 Jlink 环境变量

解压 gcc-arm-none-eabi-9-2019-q4-major-win32.zip，并把解压后的文件夹目录下的 bin 添加到 PC 系统环境变量中。

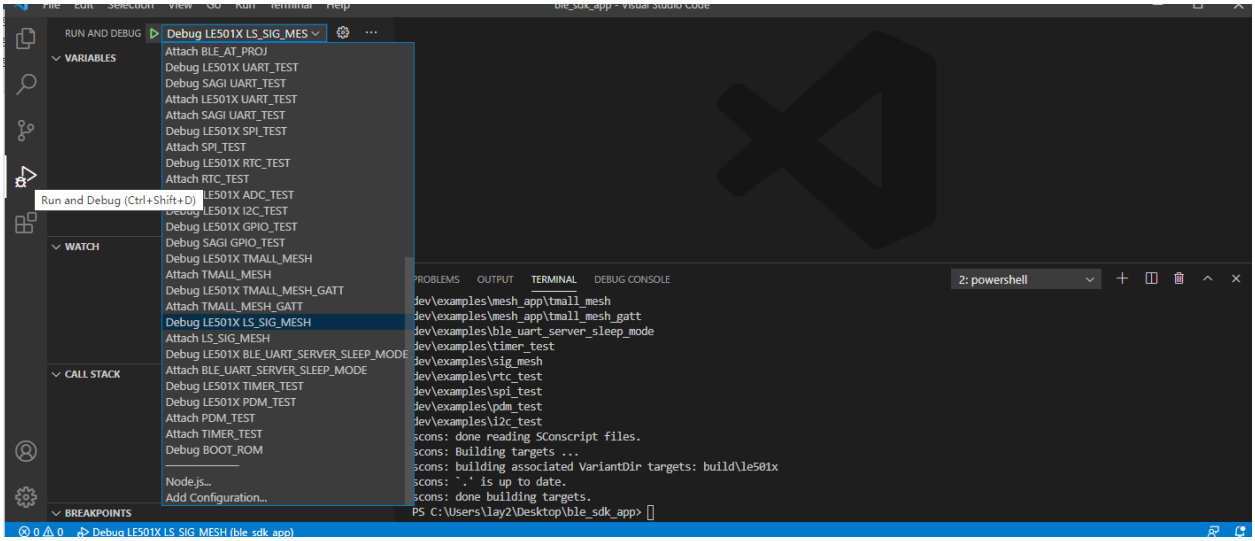


1.1.6 安装完成

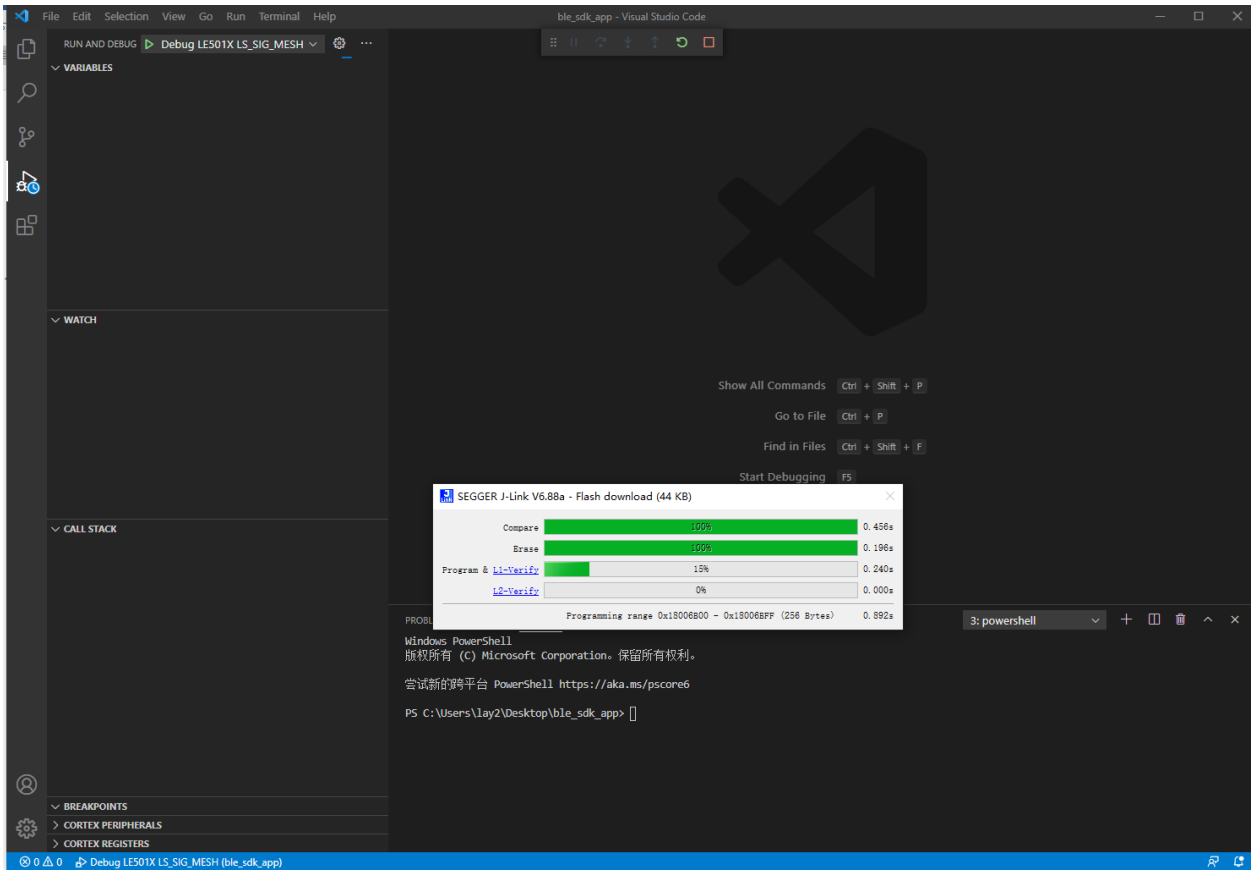
1. scons 编译 SDK 工程



2. 选择 debug 的相关示例



3. 启动 debug 模式



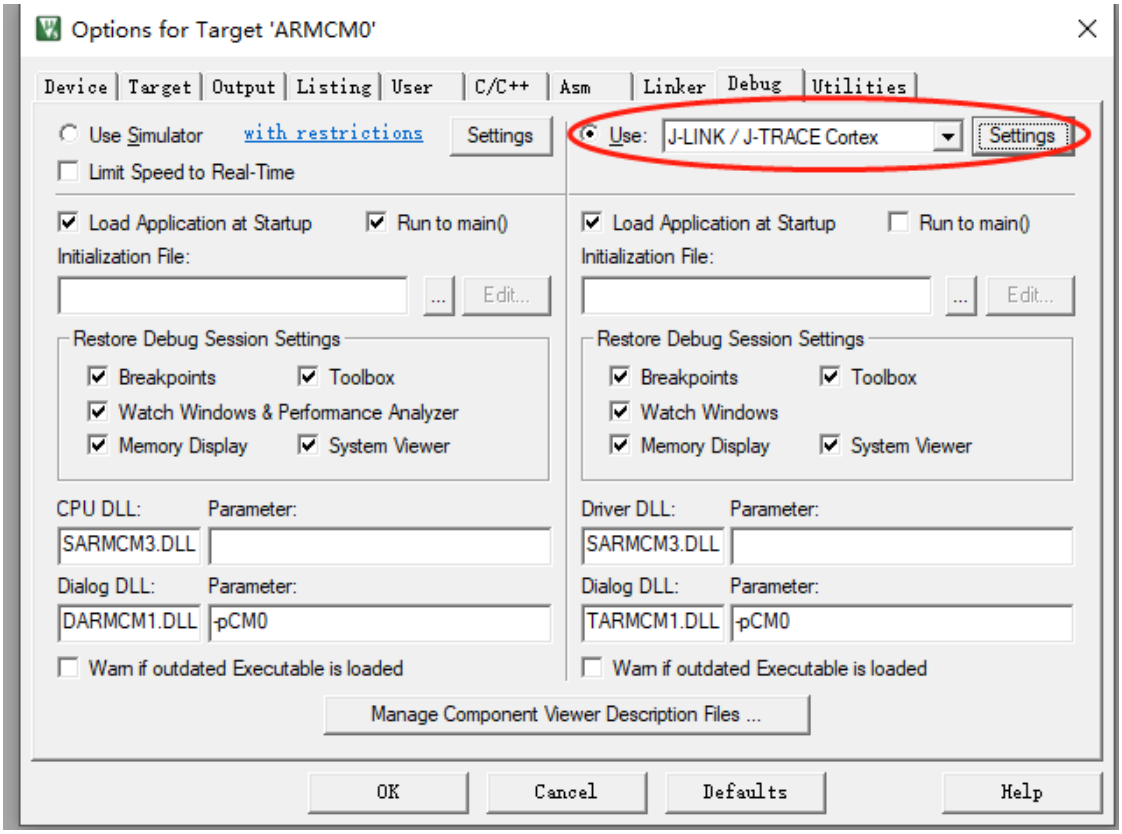
注：离线安装包请联系我们

1.2 Keil+JLink 构建、烧录、调试

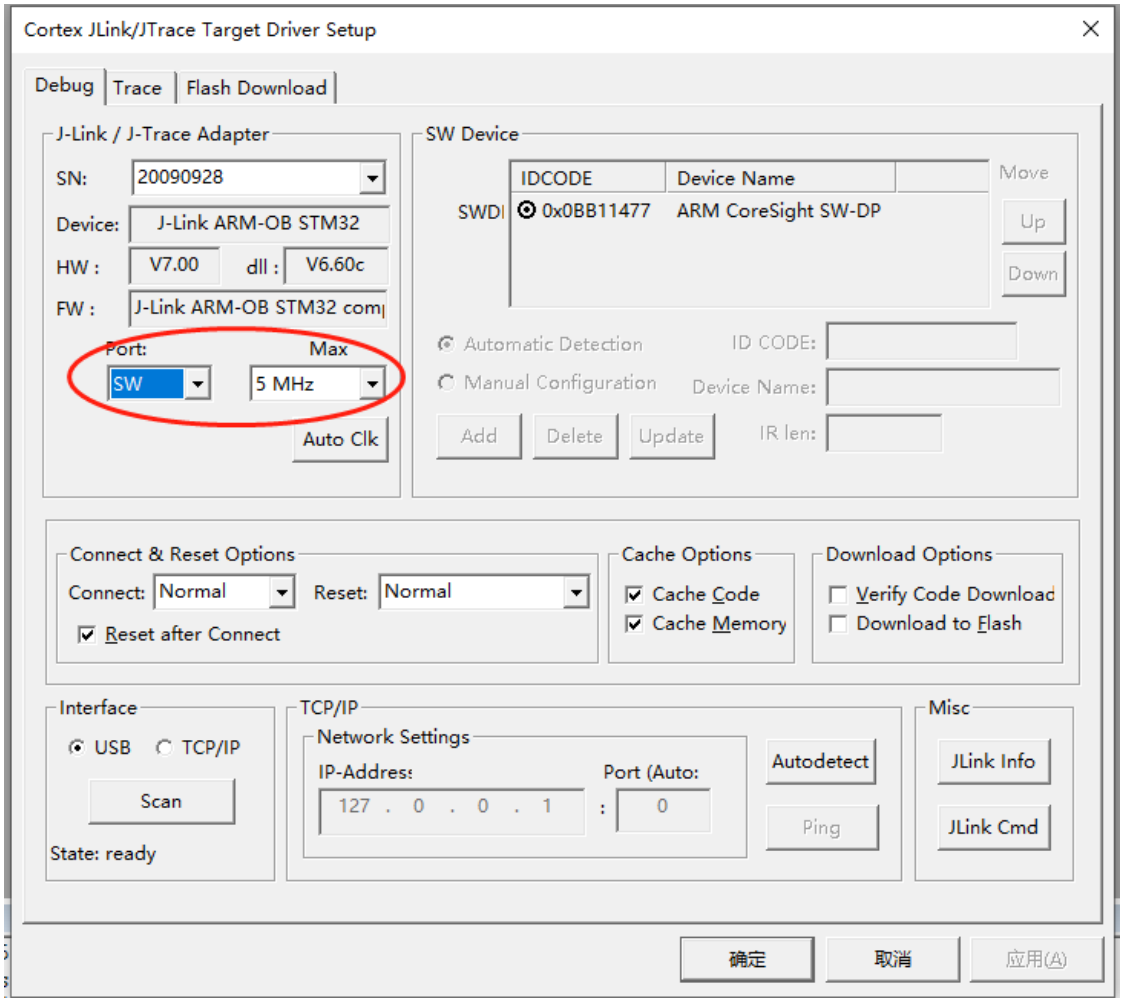
1.2.1 构建

完成 keil 环境安装后，以 ble_dis 工程为例，打开 ls_ble_sdk\dev\examples\ble_dis\mdk 路径下的工程文件

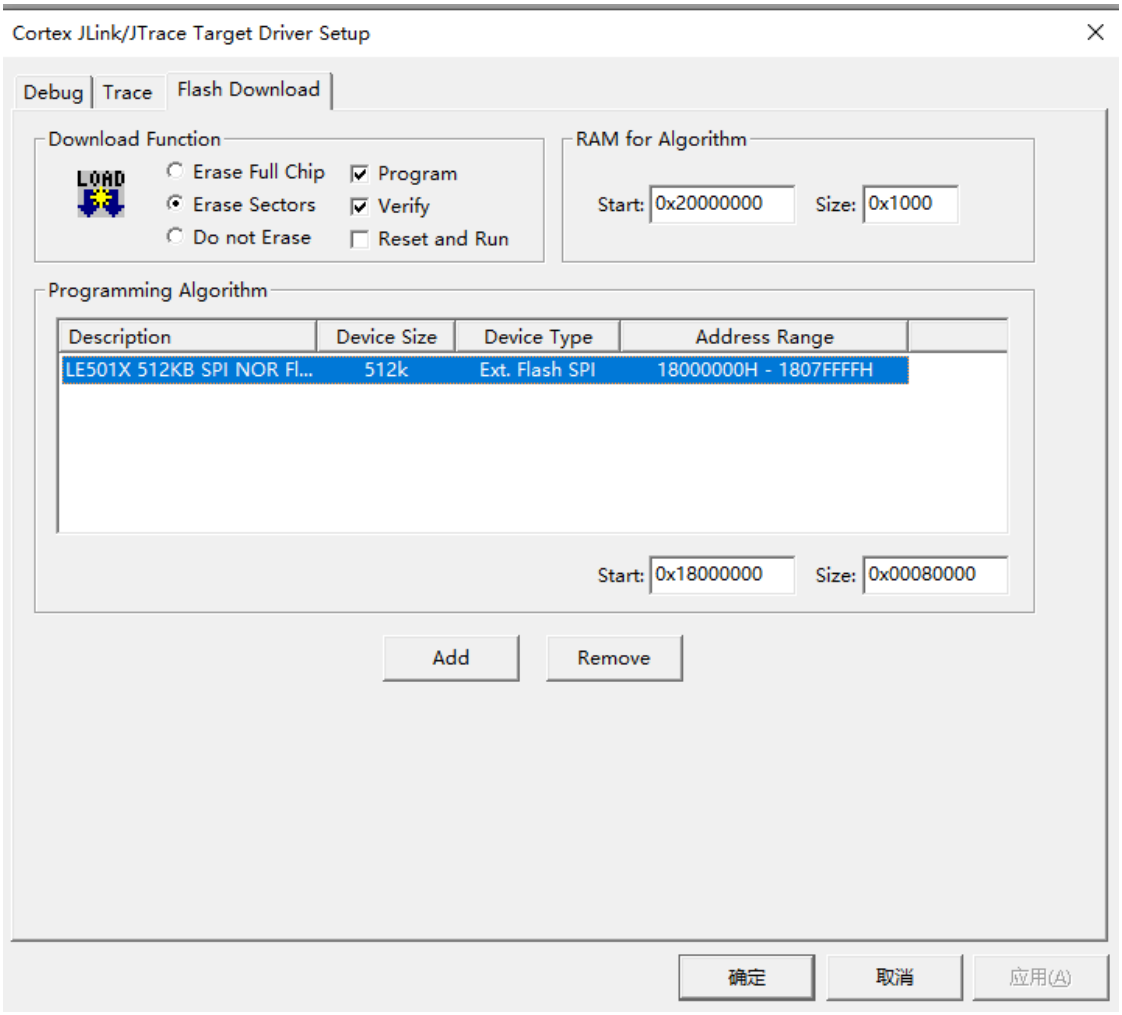
1. 选择 J-Link 作为调试工具


Options for Target 'ARMCM0'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
Use Simulator
with restrictions
Settings
Limit Speed to Real-Time
Use:
J-LINK / J-TRACE Cortex
Settings
Load Application at Startup
Run to main()
Initialization File:
Edit...
Restore Debug Session Settings
Breakpoints
Toolbox
Watch Windows & Performance Analyzer
Watch Windows
Memory Display
System Viewer
CPU DLL:
Parameter:
SARMCM3.DLL
Dialog DLL:
DARMCM1.DLL
-pCM0
Driver DLL:
TARMCM1.DLL
Warn if outdated Executable is loaded
Manage Component Viewer Description Files ...
OK
Cancel
Defaults
Help

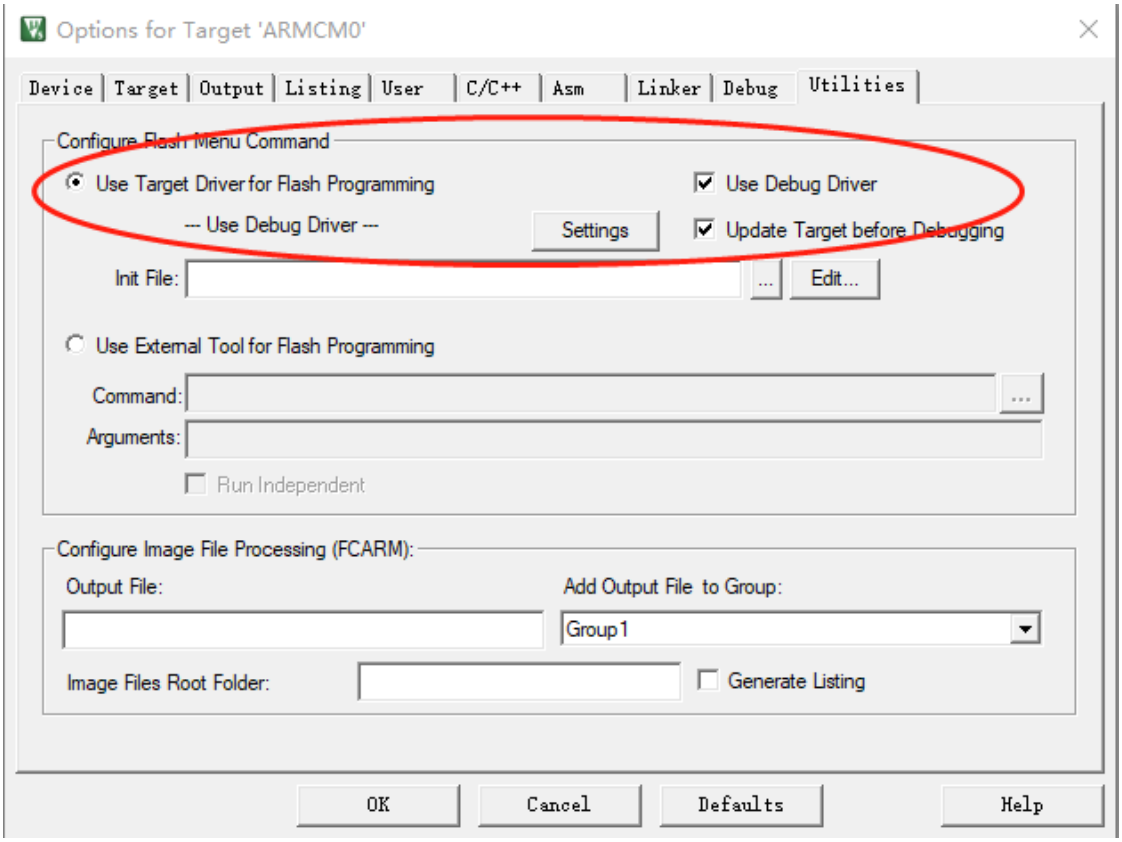
2. 选择 SW 作为调试



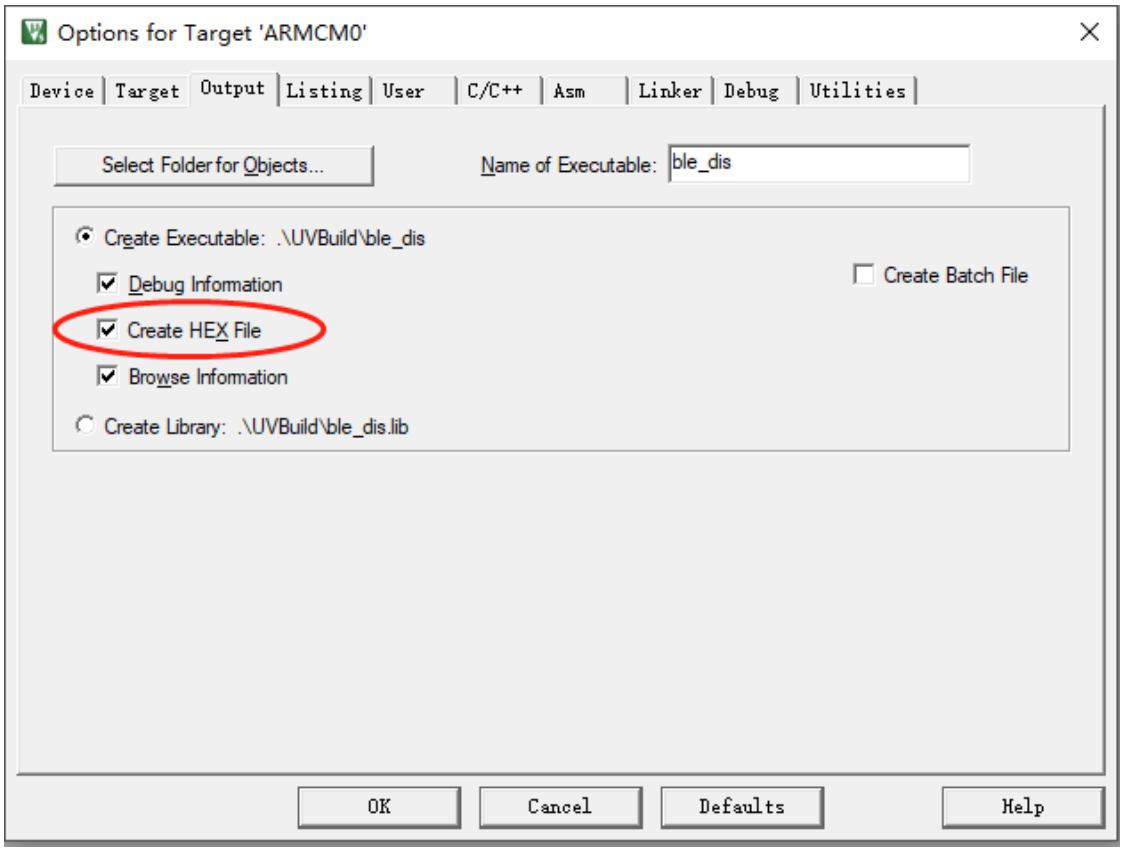
3. 在 flash download 选项卡中配置下载选项，不能选择 `Erase Full Chip` 选项，如果在编程算法选项内没有找到下图的选项，请检查软件开发环境搭建章节内的 keil 环境配置的第二条内容是否执行正确。



4. 设置使用 Debug Drive 进行 flash 的烧录



5. 选择使用创建 Hex 文件

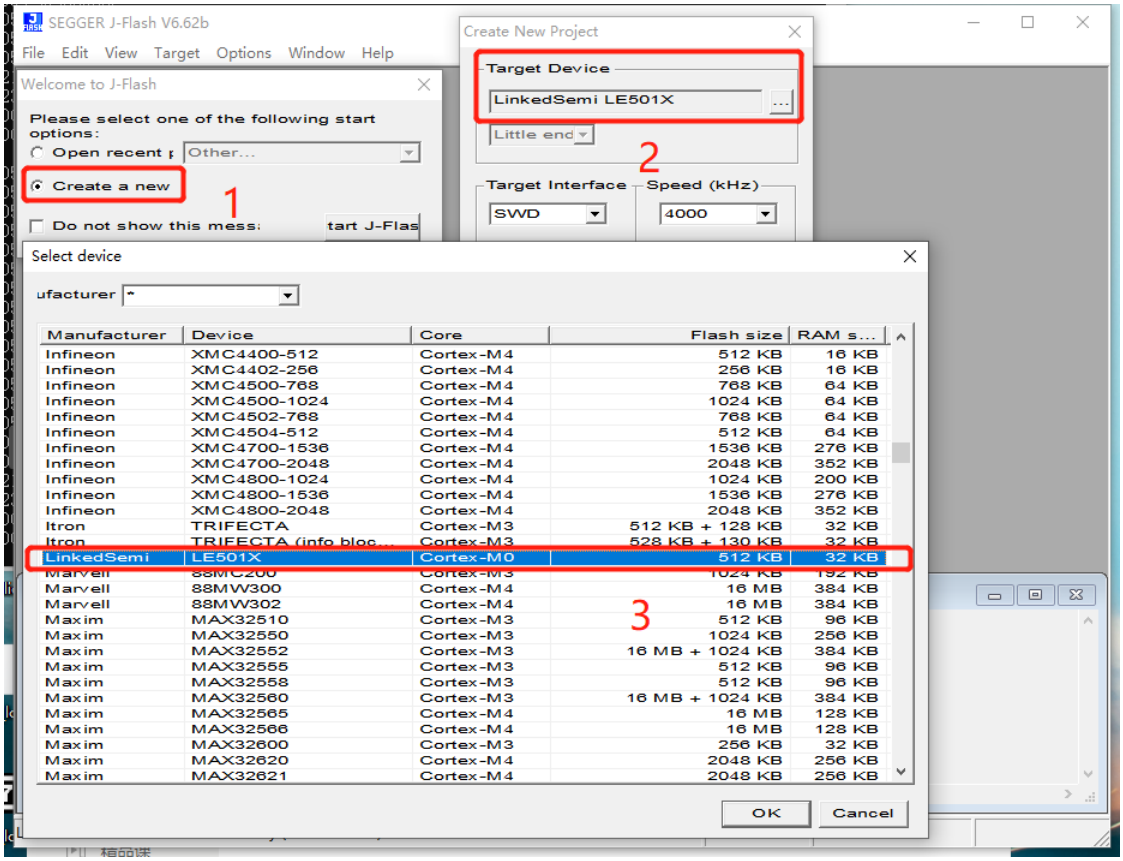


1.2.2 JLINK 烧录、调试的准备工作

1. 将 JLink 安装路径（JFlash.exe 、JLinkGDBServerCL.exe 所在目录）`{JLINK_SETUP_DIR}` 添加到系统环境变量 PATH。
2. 将 SDK `tools/prog/` 目录下所有内容拷贝到 JLink 安装路径 `{JLINK_SETUP_DIR}` 覆盖原有文件。

1.2.3 JFlash 使用

1. 打开 J-FLash, 配置 target device 为 `LinkedSemi LE501X`



2、点击 File->Open data file... 选择要烧录的 hex 文件, 或者直接将文件拖入 JFLash 中；

3、选择烧录文件之后，点击 Target->Connect, 如果能够连接成功会在 LOG 窗口最后一行显示 “Connected successfully”，否则请检查硬件接线是否正确；

4、点击 Target->Manual Programming->Erase 执行芯片全擦；

5、点击 Target->Production Programming 开始烧录选中的 hex 文件。

1.2.4 运行单独工程

1. **务必先完成 JLink 烧录、调试的准备工作**；
2. 打开 `ble_sdk_app\dev\examples\ble_uart_server\mdk` 路径下的 ble_uart_server.uvprojx 工程文件, 开始编译, 编译成功会在 `ble_sdk_app\dev\examples\ble_uart_server\mdk\UVBuild` 路径下面生成 ble_uart_server.hex、info_sbl.hex 和 ble_uart_server_production.hex 以及其他编译产生的文件

注解： XXX_production.hex 是将 second bootloader(info_sbl.hex)、协议栈 (fw.hex) 以及应用代码合并之后的文件

---

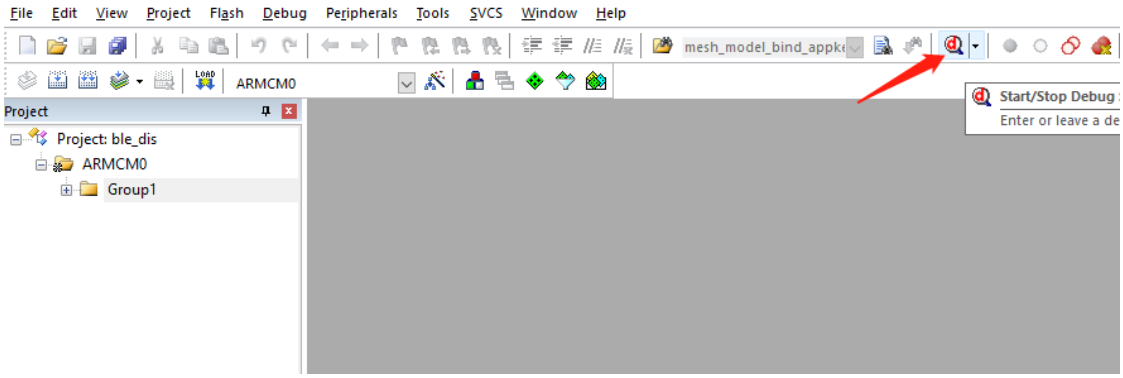
1. 芯片在第一次使用或者执行过全擦指令后，需要提前烧录 second bootloader 和协议栈内容

注解: 使用 JFlash 工具将 XXX_production.hex 烧录到芯片内，或者分别将 info_sbl.hex 、ble_sdk_appdevsocarm_cmle501xbinfw.hex 和 ble_uart_server.hex 烧录到芯片内，烧录顺序没有限制

1. 在使用 keil 的 download 或者 Debug 功能时，需要确认芯片内已经烧录了 second bootloader 和协议栈，否则程序不能跑起来，如果已经烧录过则不需要重复烧录。如果需要重新烧录 second bootloader 和协议栈，先执行芯片全擦然后再开始烧录。

1.2.5 调试

1. 使用 keil 图形界面的 Start/Stop Debug Session 选项进入调试模式



1.3 VS Code 构建、烧录、调试

1.3.1 构建

完成软件开发环境搭建 操作后，在 VS Code 中打开 SDK **根目录**，通过快捷键 `Ctrl + ` 打开 Terminal，执行:

```
scons
```

会编译所有示例程序，生成文件保存在 SDK `build/examples/` 目录下

生成文件

一般情况，每个示例工程会生成.asm，.elf，.hex，.map 四个文件。

.elf 包含调试信息的工程编译链接输出

.asm 由.elf 文件导出的反汇编

.map 编译链接生成的符号和交叉引用信息

.hex 由.elf 文件导出的 Intel Hex 格式程序镜像，用于烧录

1.3.2 JLINK 烧录、调试的准备工作

1. 将 JLink 安装路径（JFlash.exe 、JLinkGDBServerCL.exe 所在目录）`{JLINK_SETUP_DIR}` 添加到系统环境变量 PATH。
2. 将 SDK `tools/prog/` 目录下所有内容拷贝到 JLink 安装路径 `{JLINK_SETUP_DIR}` 覆盖原有文件。

1.3.3 烧录

根据*存储系统*一节的介绍，Flash 被划分为 Info、Second Boot、Image and OTA Image、Persistent Data、Protocol Stack 五个区域。除了 Persistent Data 区域是运行时写入的数据，其他四个区域都需要预先写入 Flash，程序才能正确运行。

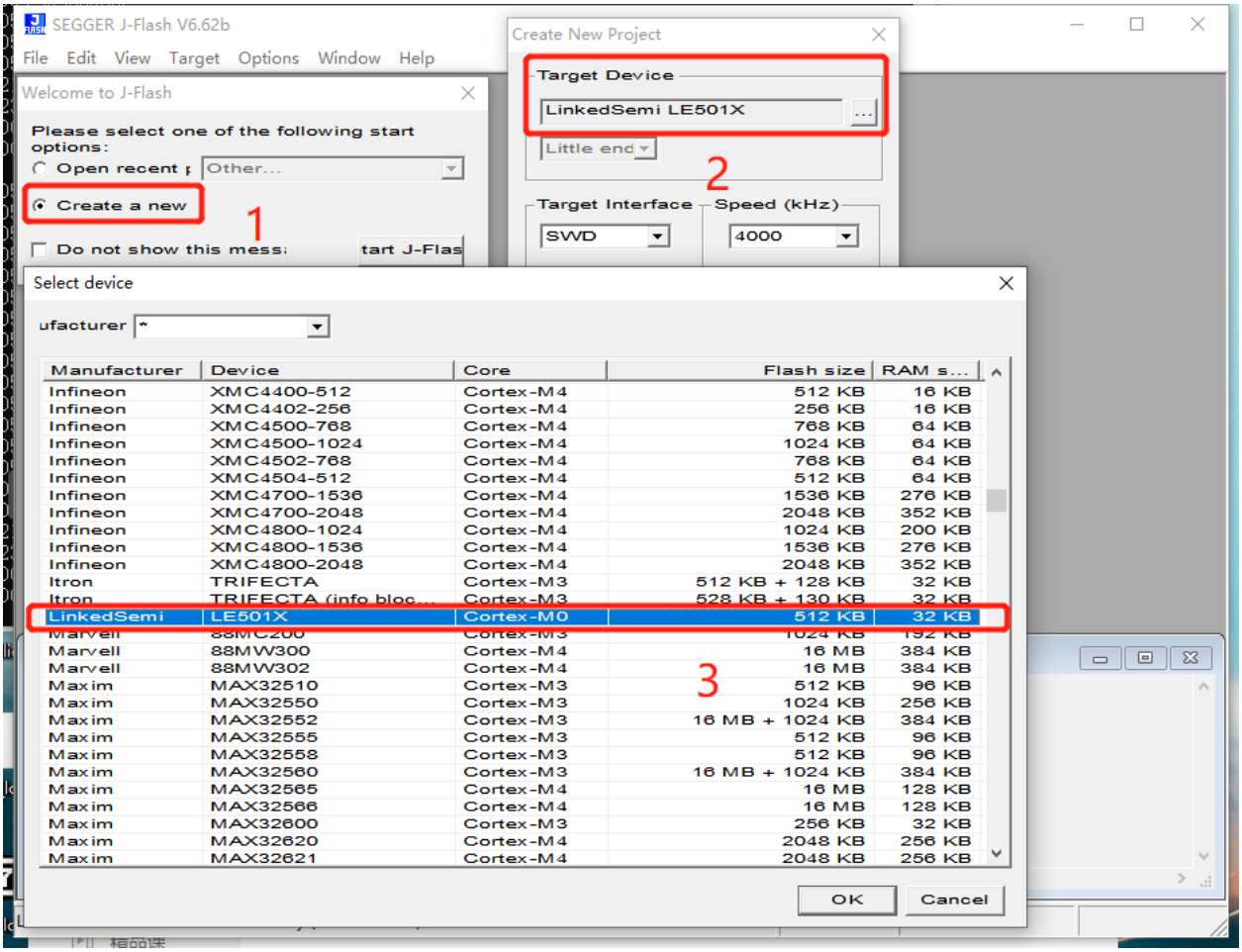
Info + Second Boot build/examples/le501x/info_sbl.hex

Protocol Stack dev/soc/arm_cm/le501x/bin/fw.hex

开发调试阶段，需要预先将上述两个 hex 文件烧录 Flash。

量产时，可以使用对应工程的 XXX_production.hex，该文件是由 info_sbl.hex、fw.hex 以及应用 hex 文件合并之后的固件。

JFlash 使用



1.3.4 调试

打开 VS Code 调试视图（快捷键 `Ctrl + Shift + D`），可以选择需要调试的程序。

Debug {example 1} 表示复位系统，自动烧录 `{example 1}` 应用镜像到 Flash 中，并从头运行

Attach {example 1} 表示以 `{example 1}` 为程序，调试器直接连接到当前运行现场

1.4 存储系统

1.4.1 Memory Map

Runtime Memory	Range
Flash	0x18000000 - 0x18080000
SRAM	0x0000 - 0xC000

1.4.2 Flash

No.	Section Name	Start Offset	End Offset	Size
8	OTA SETTINGS	0x7f000	0x80000	4KB
7	BLE PROTOCOL STACK	__stack_lma__	0x7f000	About 205KB
6	**BLE MESH STACK** (optional)	__mesh_stack_lma__	__stack_lma__	About 120KB
5	SINGLE BANK FOTA UTILITY (optional)	0x3d000		About 40KB
4	IMAGE and OTA IMAGE	0x5000		
3	PERSISTENT DATA	0x2000	0x5000	SECTION_NUM * SECTION_SIZE (Default 12KB)
2	SECOND BOOT-LOADER	0x1000	0x2000	4KB
1	INFORMATION	0x0	0x1000	4KB

INFORMATION 运行时，为只读区域。需应用程序确保不写此区域。

该区域 0x30-0x35 6 个字节默认用于存储设备默认蓝牙地址 (Default MAC Address)。

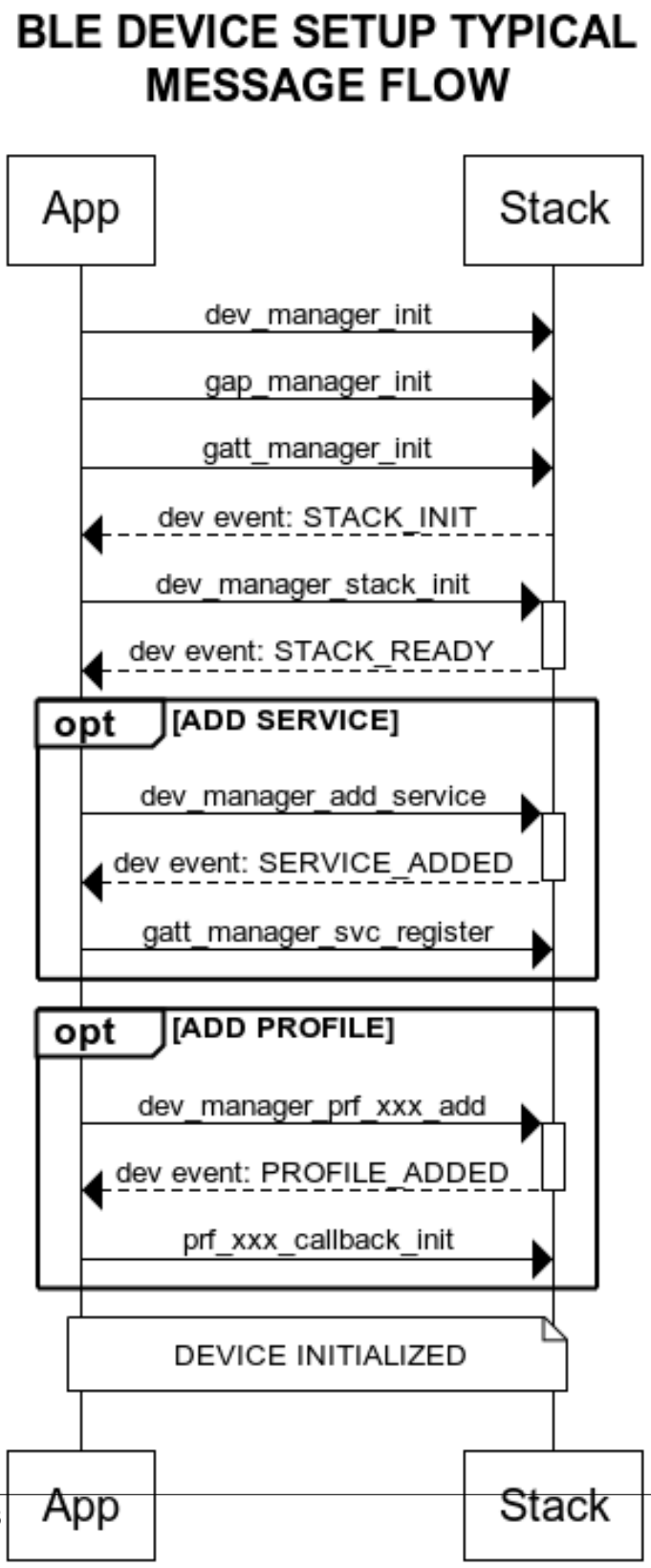
在天猫 MESH 应用里，该区域 0x200 - 0x219 26 个字节默认用于存储三元组。

1.4.3 SRAM

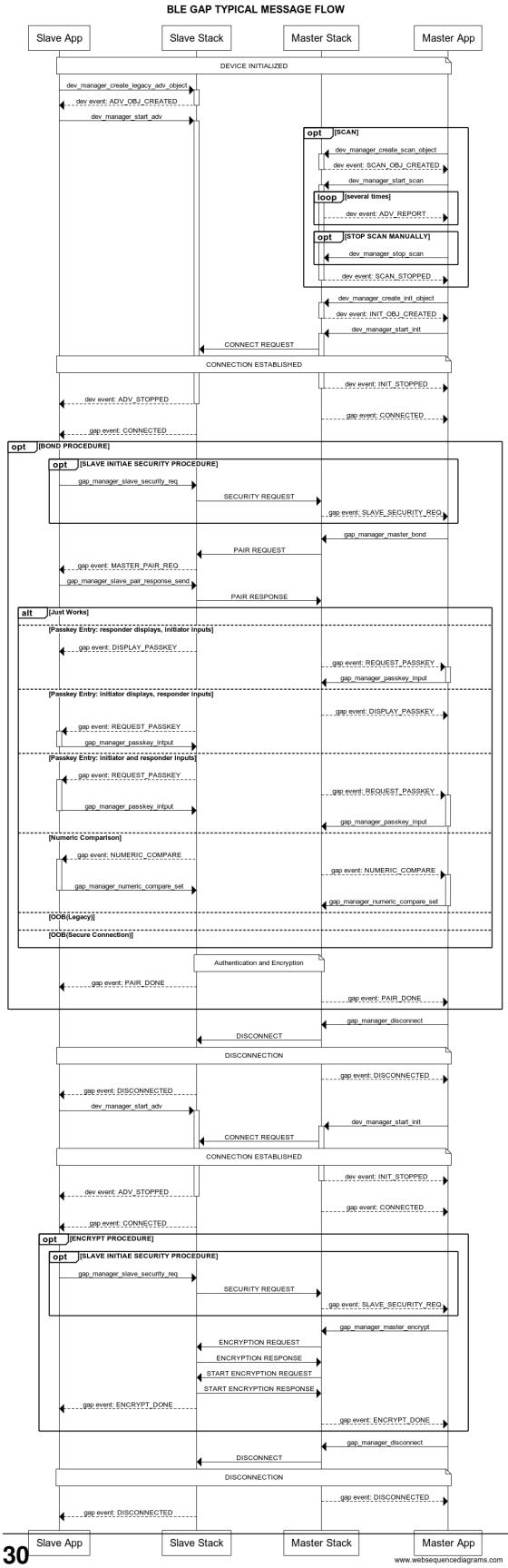
Runtime Section	Start Address	End Address	Size
BLE PROTOCOL DATA	Stack_Top	0xC000	
APP STACK		Stack_Top	
APP HEAP			
APP BSS			
APP DATA		__data_end__	
APP XIP_BANNED			
RESET_RETAIN	0xC0		
APP ISR_VECTOR	0x0	0xC0	

1.5 BLE 工作流程

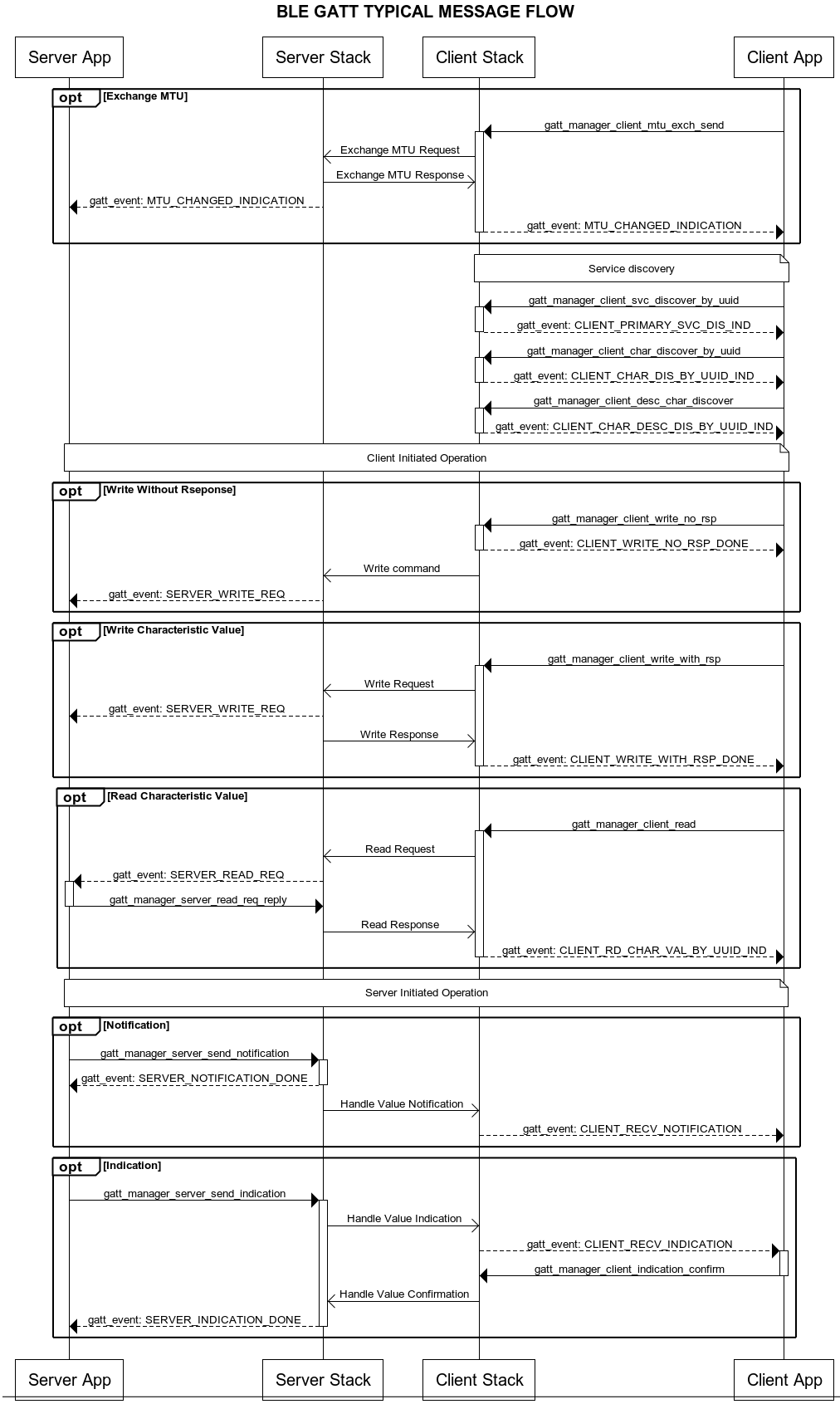
1.5.1 设备初始化



1.5.2 GAP


BLE GAP TYPICAL MESSAGE FLOW
Slave App
Slave Stack
Master Stack
Master App
DEVICE INITIALIZED
dev_manager_create_legacy_adv_object
dev event: ADV_OBJ_CREATED
dev_manager_start_adv
opt [SCAN]
dev_manager_create_scan_object
dev event: SCAN_OBJ_CREATED
dev_manager_start_scan
loop [several times]
dev event: ADV_REPORT
opt [STOP SCAN MANUALLY]
dev_manager_stop_scan
dev event: SCAN_STOPPED
dev_manager_create_init_object
dev event: INIT_OBJ_CREATED
dev_manager_start_init
CONNECT REQUEST
CONNECTION ESTABLISHED
dev event: INIT_STOPPED
dev event: ADV_STOPPED
gap event: CONNECTED
gap event: CONNECTED
opt [BOND PROCEDURE]
opt [SLAVE INITIAE SECURITY PROCEDURE]
gap_manager_slave_security_req
SECURITY REQUEST
gap event: SLAVE_SECURITY_REQ
gap_manager_master_bond
PAIR REQUEST
gap event: MASTER_PAIR_REQ
gap_manager_slave_pair_response_send
PAIR RESPONSE
alt [Just Works]
[Passkey Entry: responder displays, initiator inputs]
gap event: DISPLAY_PASSKEY
gap event: REQUEST_PASSKEY
gap_manager_passkey_input
[Passkey Entry: initiator displays, responder inputs]
gap event: DISPLAY_PASSKEY
gap event: REQUEST_PASSKEY
gap_manager_passkey_intput
[Passkey Entry: initiator and responder inputs]
gap event: REQUEST_PASSKEY
gap event: REQUEST_PASSKEY
gap_manager_passkey_intput
gap_manager_passkey_input
[Numeric Comparison]
gap event: NUMERIC_COMPARE
gap event: NUMERIC_COMPARE
gap_manager_numeric_compare_set
gap_manager_numeric_compare_set
[OOB(Legacy)]
[OOB(Secure Connection)]
Authentication and Encryption
gap event: PAIR_DONE
gap event: PAIR_DONE
gap_manager_disconnect
DISCONNECT
DISCONNECTION
gap event: DISCONNECTED
gap event: DISCONNECTED
dev_manager_start_adv
dev_manager_start_init
CONNECT REQUEST
CONNECTION ESTABLISHED
dev event: INIT_STOPPED
dev event: ADV_STOPPED
gap event: CONNECTED
gap event: CONNECTED
opt [ENCRYPT PROCEDURE]
opt [SLAVE INITIAE SECURITY PROCEDURE]
gap_manager_slave_security_req
SECURITY REQUEST
gap event: SLAVE_SECURITY_REQ
gap_manager_master_encrypt
ENCRYPTION REQUEST
ENCRYPTION RESPONSE
START ENCRYPTION REQUEST
START ENCRYPTION RESPONSE
gap event: ENCRYPT_DONE
gap event: ENCRYPT_DONE
gap_manager_disconnect
DISCONNECT
DISCONNECTION
gap event: DISCONNECTED
gap event: DISCONNECTED
Slave App
Slave Stack
Master Stack
Master App
www.websequencediagrams.com

1.5.3 GATT


BLE GATT TYPICAL MESSAGE FLOW
Server App
Server Stack
Client Stack
Client App
opt
[Exchange MTU]
gatt_manager_client_mtu_exch_send
Exchange MTU Request
Exchange MTU Response
gatt_event: MTU_CHANGED_INDICATION
gatt_event: MTU_CHANGED_INDICATION
Service discovery
gatt_manager_client_svc_discover_by_uuid
gatt_event: CLIENT_PRIMARY_SVC_DIS_IND
gatt_manager_client_char_discover_by_uuid
gatt_event: CLIENT_CHAR_DIS_BY_UUID_IND
gatt_manager_client_desc_char_discover
gatt_event: CLIENT_CHAR_DESC_DIS_BY_UUID_IND
Client Initiated Operation
opt
[Write Without Rseponse]
gatt_manager_client_write_no_rsp
gatt_event: CLIENT_WRITE_NO_RSP_DONE
Write command
gatt_event: SERVER_WRITE_REQ
opt
[Write Characteristic Value]
gatt_manager_client_write_with_rsp
Write Request
gatt_event: SERVER_WRITE_REQ
Write Response
gatt_event: CLIENT_WRITE_WITH_RSP_DONE
opt
[Read Characteristic Value]
gatt_manager_client_read
Read Request
gatt_event: SERVER_READ_REQ
gatt_manager_server_read_req_reply
Read Response
gatt_event: CLIENT_RD_CHAR_VAL_BY_UUID_IND
Server Initiated Operation
opt
[Notification]
gatt_manager_server_send_notification
gatt_event: SERVER_NOTIFICATION_DONE
Handle Value Notification
gatt_event: CLIENT_RECV_NOTIFICATION
opt
[Indication]
gatt_manager_server_send_indication
Handle Value Indication
gatt_event: CLIENT_RECV_INDICATION
gatt_manager_client_indication_confirm
Handle Value Confirmation
gatt_event: SERVER_INDICATION_DONE
Server App
Server Stack
Client Stack
Client App

1.6 SIG MESH 工作流程

1.6.1 1. sig mesh 初始化

1.6.2 2. Register Model

1.6.3 3. Provisoning

1.7 外设

1.7.1 GPIO

本节介绍 GPIO 输入输出和中断配置。GPIO 操作 API 详见 `io_config.h` 。

每一根 GPIO 都可以在 **普通输入输出**和 **外设功能**两种工作模式之间切换。

> **注意:** 在不同组的 IO，只有一个对应的 PIN 中断有效，例如：先使能 PA01 外部中断，再使能 PB01 外部中断，最后使能 PC01 外部中断，只有 PC01 有效，PA01 和 PB01 无法正常上中断。

普通输入输出模式

输出高低电平

```
io_cfg_output(PB00); //设置 PB00 为输出模式
io_write_pin(PB00,1); //设置 PB00 输出高电平
io_write_pin(PB00,0); //设置 PB00 输出低电平
io_set_pin(PB00); //设置 PB00 输出高电平
io_clr_pin(PB00); //设置 PB00 输出低电平
io_toggle_pin(PB00); //翻转 PB00 输出电平
```

读取输入电平

```
io_cfg_input(PB01); //设置 PB01 为输入模式
uint8_t val = io_read_pin(PB01); //读取 PB01 电平，保存到 val 变量
```

配置上升、下降沿中断

```
io_cfg_input(PA00); //设置 PA00 为输入模式
io_exti_config(PA00,INT_EDGE_RISING); //配置 PA00 中断，上升沿触发
io_exti_enable(PA00,true); //使能 PA00 中断
io_cfg_input(PB11); //设置 PB11 为输入模式
```

(下页继续)

(续上页)

```
io_exti_config(PB11,INT_EDGE_FALLING); //配置 PB11 中断，下降沿触发
io_exti_enable(PB11,true); //使能 PB11 中断

void io_exti_callback(uint8_t pin) // override io_exti_callback
{
    switch(pin)
    {
    case PA00:
        // do something
    break;
    case PB11:
        // do something
    break;
    default:
    break;
    }
}
```

注解： 在 LE501X 系统中，只有 PA00、PA07、PB11、PB15 四个 IO 可以在任意状态下触发中断，其余 IO 只能在 CPU 工作状态下触发中断，不能在 BLE 休眠状态下触发中断。

BLE 处于广播或者连接时，系统根据广播、连接的间隔定期自动休眠、唤醒，所以这种应用场景，建议使用上述四根 IO 作为中断。

配置内部上、下拉：

```
io_pull_write(PA04,IO_PULL_UP); //设置 PA04 内部上拉
io_pull_write(PA04,IO_PULL_DOWN); //设置 PA04 内部下拉
io_pull_type_t pull = io_pull_read(PA04); //读取 PA04 内部上下拉状态，保存到 pull 变量
io_pull_write(PA04,IO_PULL_DISABLE); //禁用 PA04 内部上下拉
```

外设功能模式

具体配置参考各外设文档

UART

I2C

TIMER

PDM

1.7.2 UART

UART（Universal Asynchronous Receiver/Transmitter）通用异步收发传输器，UART 作为异步串口通信协议的一种，工作原理是将传输数据的每个字符一位接一位地传输。

UART 是在应用程序开发过程中使用次数最多的数据总线。

一、初始化

1.1 设置 UART 模块的 IO

调用 IO 的初始化接口，可以将任意 IO 配置 UART 的 TX 或者 RX，与其他设备进行通信。

```
void uart1_io_init(uint8_t txd,uint8_t rxd);
void uart2_io_init(uint8_t txd,uint8_t rxd);
void uart3_io_init(uint8_t txd,uint8_t rxd);
```

注解： 芯片的 IO 一共有 34 个，具体情况需根据封装图来定义。为了避免不必要的 bug，在使用 UART 通信的时候，请先初始化 IO，再进行其余参数的配置。

1.2 设置 UART 模块参数变量

设置 UART 模块的参数变量，其结构体的参数原型如下：

```
typedef struct
{
    app_uart_baudrate_t BaudRate;                    /*!< This member configures the␣
↪UART communication baud rate.*/

    uint8_t      WordLength:2,                 /*!< Specifies the number of data bits␣
↪transmitted or received in a frame.
                                                    This parameter can be a value of␣
↪@ref UART_Word_Length */

                 StopBits:1,                   /*!< Specifies the number of stop bits␣
↪transmitted.
                                              This parameter can be a value of @ref UART_
↪Stop_Bits */

                 Parity:2,                     /*!< Specifies the parity mode.
                                              This parameter can be a value of @ref UART_
↪Parity
```

(下页继续)

(续上页)

```
                                        @note When parity is enabled, the computed␣
↪parity is inserted
                                              at the MSB position of the␣
↪transmitted data (9th bit when
                                              the word length is set to 9 data␣
↪bits; 8th bit when the
                                              word length is set to 8 data bits).␣
↪*/
                MSBEN:1,
                HwFlowCtl:1;                  /*!< Specifies whether the hardware flow␣
↪control mode is enabled or disabled.
                                              This parameter can be a value of␣
↪@ref UART_Hardware_Flow_Control */
} UART_InitTypeDef;
```

提供的配置参数可取值的为如下宏定义:

```
//UART communication baud rate.
typedef enum
{
    UART_BAUDRATE_1200   = UART_BUADRATE_ENUM_GEN(1200),
    UART_BAUDRATE_2400   = UART_BUADRATE_ENUM_GEN(2400),
    UART_BAUDRATE_4800   = UART_BUADRATE_ENUM_GEN(4800),
    UART_BAUDRATE_9600   = UART_BUADRATE_ENUM_GEN(9600),
    UART_BAUDRATE_14400 = UART_BUADRATE_ENUM_GEN(14400),
    UART_BAUDRATE_19200 = UART_BUADRATE_ENUM_GEN(19200),
    UART_BAUDRATE_28800 = UART_BUADRATE_ENUM_GEN(28800),
    UART_BAUDRATE_38400  = UART_BUADRATE_ENUM_GEN(38400),
    UART_BAUDRATE_57600 = UART_BUADRATE_ENUM_GEN(57600),
    UART_BAUDRATE_76800  = UART_BUADRATE_ENUM_GEN(76800),
    UART_BAUDRATE_115200 = UART_BUADRATE_ENUM_GEN(115200),
    UART_BAUDRATE_230400 = UART_BUADRATE_ENUM_GEN(230400),
    UART_BAUDRATE_250000 = UART_BUADRATE_ENUM_GEN(250000),
    UART_BAUDRATE_500000 = UART_BUADRATE_ENUM_GEN(500000),
    UART_BAUDRATE_460800 = UART_BUADRATE_ENUM_GEN(460800),
    UART_BAUDRATE_750000=  UART_BUADRATE_ENUM_GEN(750000),
    UART_BAUDRATE_921600 = UART_BUADRATE_ENUM_GEN(921600),
    UART_BAUDRATE_1000000= UART_BUADRATE_ENUM_GEN(1000000),
    UART_BAUDRATE_2000000= UART_BUADRATE_ENUM_GEN(2000000),
}app_uart_baudrate_t;
//Parity
#define UART_NOPARITY       0x0     // Parity diable
#define UART_ODDPARITY      0x1     // Parity Odd
#define UART_EVENPARITY     0x3     // Parity Even
```

(下页继续)

(续上页)

```
//UART_BYTESIZE
#define UART_BYTESIZE5      0X0     // Byte size 5 bits
#define UART_BYTESIZE6      0X1     // Byte size 6 bits
#define UART_BYTESIZE7      0X2     // Byte size 7 bits
#define UART_BYTESIZE8      0X3     // Byte size 8 bits
//UART_STOPBITS
#define UART_STOPBITS1      0x0     // Stop 1 bits
#define UART_STOPBITS2      0x1     // Stop 2 bits
//UART_BIT_ORDER
#define UART_LSB      0x0     // LSBEN
#define UART_MSB      0x1     // MSBEN
```

1.3 初始化 UART 模块

通过初始化接口，应用程序可以对串口设备进行参数配置。

```
HAL_StatusTypeDef HAL_UART_Init(UART_HandleTypeDef *huart);
```

二、反初始化

2.1 反初始化 UART 模块

通过反初始化接口，应用程序可以关闭 UART 外设，从而在运行 BLE 的程序的时候，降低系统的功耗。

```
HAL_StatusTypeDef HAL_UART_DeInit(UART_HandleTypeDef *huart);
```

2.2 反初始化 UART IO

反初始化 IO 接口的主要目的是为了避免在进入低功耗模式时，IO 上产生漏电，或者给对接设备发送不必要的数据。调用此接口后, 会默认的将 UART 的 TX IO 配置成高电平，RX 配置成无上下拉的输入模式。

```
void uart1_io_deinit(void);
void uart2_io_deinit(void);
void uart3_io_deinit(void);
```

注解： UART 初始化动作会向系统注册 UART 进入工作状态，当系统检测到有任一外设处于工作状态时，都不会进入低功耗休眠。因此，UART 使用完毕，需要进入低功耗状态之前，必须反初始化 UART。

三、UART 设备数据的收发

串口数据接收和发送数据的模式分为 3 种：非阻塞（中断）模式、阻塞模式、DMA 模式。在使用的时候，这 3 种模式只能选其一。

3.1 数据收发——阻塞方式

以阻塞方式接收发送模式使用串口设备的接口如下所示：

```
HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_
↪t Size, uint32_t Timeout);
HAL_StatusTypeDef HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint16_
↪t Size,uint32_t Timeout);
```

注解： Timeout 以 ms 为单位，当 Timeout = 0xffffffff 时，超时时间为无限长。

3.2 数据收发——非阻塞（中断）方式

以非阻塞（中断）方式接收发送模式使用串口设备的接口如下所示：

```
HAL_StatusTypeDef HAL_UART_Transmit_IT(UART_HandleTypeDef *huart, uint8_t *pData,␣
↪uint16_t Size);
HAL_StatusTypeDef HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData,␣
↪uint16_t Size);
```

3.3 数据收发——DMA 方式

以 DMA 方式接收发送模式使用串口设备的接口如下所示：

```
HAL_StatusTypeDef HAL_UART_Transmit_DMA(UART_HandleTypeDef *huart, uint8_t *pData,␣
↪uint16_t Size);
HAL_StatusTypeDef HAL_UART_Receive_DMA(UART_HandleTypeDef *huart, uint8_t *pData,␣
↪uint16_t Size);
```

四、代码示例

初始化以及非阻塞（中断）模式收发的示例如下：

```
#include "io_config.h"
#include "lsuart.h"
#define TEST_ZONE_SIZE 512
uint8_t uart_rx_buf[TEST_ZONE_SIZE] ;
uint8_t uart_tx_buf[TEST_ZONE_SIZE] ;

UART_HandleTypeDef UART_Config;

// UART Transmit complete callback
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
    /*note:When entering this function, it means that UART TX is complete*/
}
//UART Receive Complete Callback
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    /*note:When entering this function, it means that UART RX is complete*/
}

void uart1_init()
{
    uart1_io_init(PB00,PB01);                                    // init step 1
    UART_Config.UARTX = UART1;
    UART_Config.Init.BaudRate = UART_BAUDRATE_115200;
    UART_Config.Init.MSBEN = 0;
    UART_Config.Init.Parity = UART_NOPARITY;
    UART_Config.Init.StopBits = UART_STOPBITS1;
    UART_Config.Init.WordLength = UART_BYTESIZE8;                // init step 2
    HAL_UART_Init(&UART_Config);                                 // init step 3
}

void uart1_deinit()
{
    HAL_UART_DeInit(&UART_Config);                               // deinit step 1
    uart1_io_deinit();                                           // deinit step 2
}

static void uart_test()
{
    HAL_UART_Transmit_IT(&UART_Config,uart_tx_buf,1);
    HAL_UART_Receive_IT(&UART_Config,uart_rx_buf,1);
```

(下页继续)

(续上页)

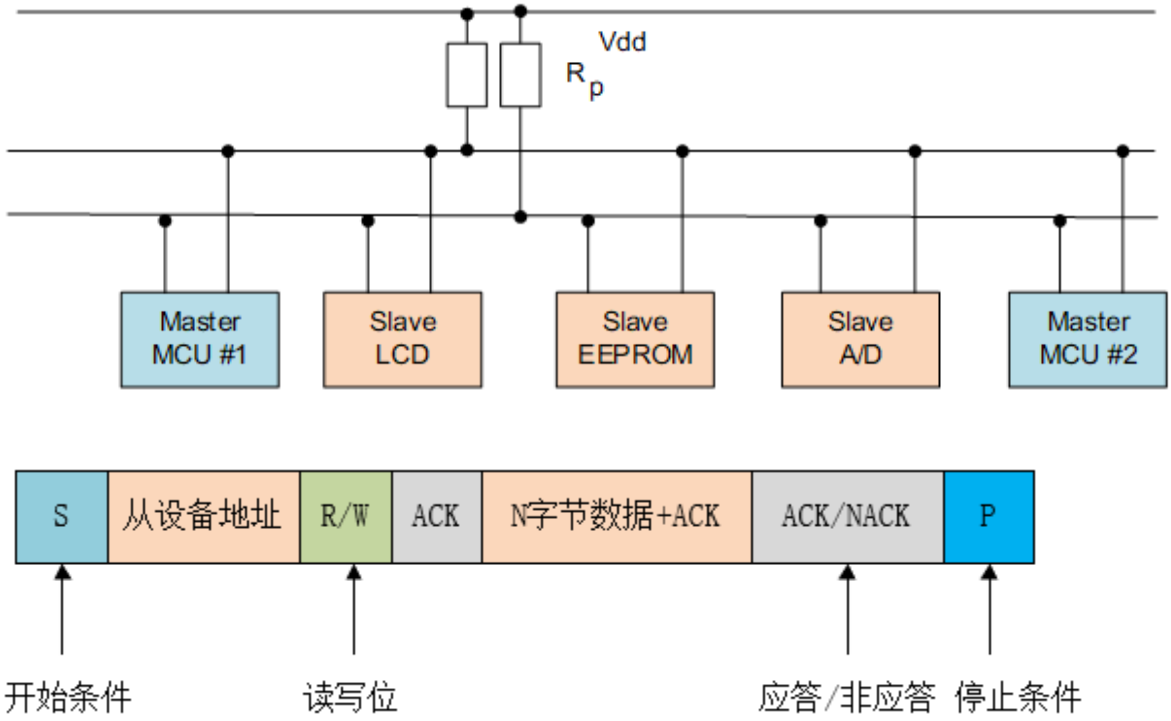
```
}

int main()
{
    uart1_init();
    uart1_test();
    while(1)
    {
    }
}
```

1.7.3 I2C

一、I2C 简介

I2C（Inter Integrated Circuit）总线是 PHILIPS 公司开发的一种半双工、双向二线制同步串行总线。I2C 总线传输数据时只需两根信号线，一根是双向数据线 SDA（serial data），另一根是双向时钟线 SCL（serial clock）。SPI 总线有两根线分别用于主从设备之间接收数据和发送数据，而 I2C 总线只使用一根线进行数据收发。

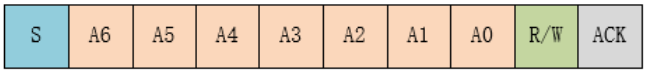
I2C 允许同时有多个主设备存在，每个连接到总线上的器件都有唯一的地址，主设备启动数据传输并产生时钟信号，从设备被主设备寻址，同一时刻只允许有一个主设备。

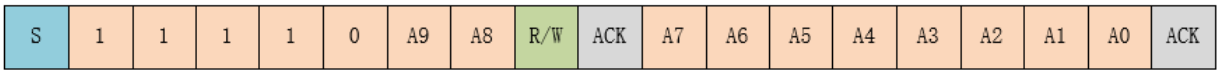


当总线空闲时，SDA 和 SCL 都处于高电平状态，当主机要和某个从机通讯时，会先发送一个开始条件，然后

发送从机地址和读写控制位，接下来传输数据（主机发送或者接收数据），数据传输结束时主机会发送停止条件。传输的每个字节为 8 位，高位在前，低位在后。数据传输过程中的不同名词详解如下所示：

开始条件：SCL 为高电平时，主机将 SDA 拉低，表示数据传输即将开始。从机地址：主机发送的第一个字节为从机地址，高 7 位为地址，最低位为 R/W 读写控制位，1 表示读操作，0 表示写操作。一般从机地址有 7 位地址模式和 10 位地址模式两种，如果是 10 位地址模式，第一个字节的头 7 位是 11110XX 的组合，其中最后两位（XX）是 10 位地址的两个最高位，第二个字节为 10 位从机地址的剩下 8 位，如下图所示：





应答信号：每传输完成一个字节的数据，接收方就需要回复一个 ACK（acknowledge）。写数据时由从机发送 ACK，读数据时由主机发送 ACK。当主机读到最后一个字节数据时，可发送 NACK（Not acknowledge）然后跟停止条件。数据：从机地址发送完后可能会发送一些指令，依从机而定，然后开始传输数据，由主机或者从机发送，每个数据为 8 位，数据的字节数没有限制。重复开始条件：在一次通信过程中，主机可能需要和不同的从机传输数据或者需要切换读写操作时，主机可以再发送一个开始条件。停止条件：在 SDA 为低电平时，主机将 SCL 拉高并保持高电平，然后在将 SDA 拉高，表示传输结束。

二、I2c 接口介绍

2.1 初始化：

一般情况下 MCU 的 I2C 器件都是作为主机和从机通讯。I2c 初始化，首先需要定义好 I2c 的 SDA 与 SCL 的 io 口。

```
void iic1_io_init(uint8_t scl,uint8_t sda);
void iic2_io_init(uint8_t scl,uint8_t sda);
```

然后开始初始化 I2c 设备。在初始化时需要先把设备一些配置信息先填写完成，如下代码所示：

```
HAL_I2C_Init(I2C_HandleTypeDef *hi2c);
```

如果初始化成功后便可以返回值为 HAL_OK，否则为 HAL_ERROR。

```
typedef struct __I2C_HandleTypeDef
{
reg_i2c_t                   *Instance;      /*!< I2C registers base address           ␣
↪    */

I2C_InitTypeDef             Init;           /*!< I2C communication parameters         ␣
↪    */
```

(下页继续)

(续上页)

```
uint8_t                    *pBuffPtr;      /*!< Pointer to I2C transfer buffer       ␣
↪    */

uint16_t                   XferSize;       /*!< I2C transfer size                    ␣
↪    */

uint16_t                   XferCount;      /*!< I2C transfer counter                 ␣
↪    */

uint32_t                   XferOptions;    /*!< I2C transfer options                 ␣
↪    */

uint32_t                   PreviousState;  /*!< I2C communication Previous state and␣
↪mode
                                                context for internal usage          ␣
↪    */
void                           *DMAC_Instance;
union{
        struct I2cDMAEnv DMA;
        struct I2cInterruptEnv Interrupt;
}Tx_Env,Rx_Env;                            /*!< I2C Tx/Rx DMA handle parameters      ␣
↪     */

HAL_LockTypeDef            Lock;           /*!< I2C locking object                   ␣
↪    */

HAL_I2C_StateTypeDef  State;          /*!< I2C communication state                */

HAL_I2C_ModeTypeDef   Mode;           /*!< I2C communication mode                 */

uint32_t              ErrorCode;      /*!< I2C Error code                         */

uint32_t              Devaddress;     /*!< I2C Target device address              */

uint32_t              EventCount;     /*!< I2C Event counter                      */
} I2C_HandleTypeDef;
```

注解： 注：具有三种传输模式：标准模式传输速率为 100kbit/s ，快速模式为 400kbit/s ，高速模式下可达 3.4Mbit/s，但目前大多 I2C 设备尚不支持高速模式。

2.2 i2c 收发：

参数描述

注解:

1. hi2c：配置 i2c 参数 (Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.)
2. DevAddress：从设备地址 (Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface)
3. pData：保存数据位置指针 (Pointer to data buffer)
4. Size：发送或接收的数据量
5. Timeout: 设置超时时间

2.2.1 轮询模式

```
HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData,␣
↪uint16_t Size, uint32_t Timeout)
/* 主设备以轮询模式向从设备发送数据 */
HAL_I2C_Master_Receive(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData,␣
↪uint16_t Size, uint32_t Timeout)
/* 主设备以轮询模式读取从设备的数据 */
```

2.2.2 中断模式

```
HAL_I2C_Master_Transmit_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t␣
↪*pData, uint16_t Size)
/* 主设备以中断的模式向从设备发送数据 */
HAL_I2C_Master_Receive_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t␣
↪*pData, uint16_t Size)
/* 主设备以中断的模式读取从设备的数据 */
```

2.2.3 回调函数

```
void HAL_I2C_MasterTxCpltCallback(I2C_HandleTypeDef *hi2c)
{

}
void HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef *hi2c)
{

}
```

注解:

在使用中断模式的时候，在执行发送与接受动作完成时会进相应的回调函数，这时候我们表示我们的数据已经发送或接收完成，现在可以进行下一步操作。

这个函数属于弱定义，用户可以自行定义，并完成相应的逻辑处理。

2.3 反初始化

2.3.1 反初始化 I2C 模块

通过反初始化接口，应用程序可以关闭 i2c 外设，从而在运行 BLE 的程序的时候，降低系统的功耗。

```
HAL_StatusTypeDef HAL_I2C_DeInit(I2C_HandleTypeDef *hi2c);
/* 如果初始化成功后便可以返回值为 HAL_OK，否则为 HAL_ERROR*/
```

2.3.2 反初始化 I2C IO

反初始化 IO 接口的主要目的是为了避免在进入低功耗模式时，IO 上产生漏电，或者给对接设备发送不必要的数据。调用此接口后，会默认的将 i2c 的 SDA 与 SCL 配置成输入模式（IO 浮空）。

```
void iic1_io_deinit(void);
void iic2_io_deinit(void);
```

注解: I2C 初始化动作会向系统注册 I2C 进入工作状态，当系统检测到有任一外设处于工作状态时，都不会进入低功耗休眠。因此，I2C 使用完毕，需要进入低功耗状态之前，必须反初始化 I2C。

1.7.4 TIMER

我们有高级控制定时器 (ADTIM)、通用定时器 (GPTIMA,GPTIMB,GPTIMC) 与基础 (BSTIM) 定时器，他们是完全独立的，不共享任何资源，可以同步操作。下图主要从预装载寄存器（ARR）位数、预分频寄存器（PSC）、计数模式、是否支持重复计数、具有几路输出比较/输入捕获以及互补输出通道来区分不同 timer 之间的区别：

timer	ARR reg	PSC reg	Counter mode	RCR reg	Compare/Capture	Complementary output
BSTIM	16bit	16bit	UP	N	N	N
GPTIMA	32bit	16bit	UP/DOWN	N	4	N
GPTIMB	16bit	16bit	UP/DOWN	N	4	N
GPTIMC	16bit	16bit	UP	Y	2	ch1
ADTIM	16bit	16bit	UP/DOWN	Y	4	ch1/ch2/ch3

时基介绍

时钟源

定时器时钟来源于系统时钟，软件中可以通过宏 `SDK_HCLK_MHZ` 获取

计数器时钟

定时器时钟经过 PSC 预分频器之后，用来驱动计数器计数。PSC 是一个 16 位的预分频器，可以对定时器时钟进行 1~65536 之间的任何一个数进行分频。具体计算方式为：SDK_HCLK_MHZ/(PSC+1)

计数器

除了 BSTIM 和 GPTIMC 只有递增计数模式外，其它定时器计数器都有三种计数模式，分别为递增计数模式、递减计数模式和递增/递减 (中心对齐) 计数模式。

1. 递增计数模式下，计数器从 0 开始计数，每来一个脉冲计数器就增加 1，直到计数器的值与自动重载寄存器 ARR 值相等，然后计数器又从 0 开始计数并生成计数器上溢事件，计数器总是如此循环计数。如果禁用重复计数器，在计数器生成上溢事件就马上生成更新事件；如果使能重复计数器，每生成一次上溢事件重复计数器内容就减 1，直到重复计数器内容为 0 时才会生成更新事件。
2. 递减计数模式下，计数器从自动重载寄存器 ARR 值开始计数，每来一个脉冲计数器就减 1，直到计数器值为 0，然后计数器又从自动重载寄存器 ARR 值开始递减计数并生成计数器下溢事件，计数器总是如此循环计数。如果禁用重复计数器，在计数器生成下溢事件就马上生成更新事件；如果使能重复计数器，每生成一次下溢事件重复计数器内容就减 1，直到重复计数器内容为 0 时才会生成更新事件。

3. 中心对齐模式下，计数器从 0 开始递增计数，直到计数值等于 (ARR-1) 值生成计数器上溢事件，然后从 ARR 值开始递减计数直到 1 生成计数器下溢事件。然后又从 0 开始计数，如此循环。每次发生计数器上溢和下溢事件都会生成更新事件。

重复计数器

对于 BSTIM、GPTIMA、GPTIMB 发生上/下溢事件时直接就生成更新事件，而 GPTIMC 和 ADTIM 在硬件结构上多出了重复计数器，在定时器发生上溢或下溢事件是递减重复计数器的值，只有当重复计数器为 0 时才会生成更新事件。在发生 N+1 个上溢或下溢事件 (N 为 RCR 的值) 时产生更新事件。

自动重装载寄存器

自动重装载寄存器里面装着计数器能计数的最大数值，最大值取决于寄存器的位数，除了 GPTIMA 是 32 位的外其余都是 16 位。当计数到这个值的时候，如果使能了中断的话，定时器就产生溢出中断。

定时时间计算

定时器的定时时间等于计数器的周期乘以计数次数。计一个数的时间则是计数器时钟的倒数，等于：1/（SDK_HCLK_MHZ*1000000/(PSC+1)） ，可以计算出我们需要的定时时间就等于：1/（SDK_HCLK_MHZ*1000000/(PSC+1)）*(ARR+1)。

PWM 输出频率计算

假设需要配置频率为 X 的脉冲，通过公式 ARR =（SDK_HCLK_MHZ*1000000/(PSC+1)）/ X - 1；可以确定需要配置的预分频寄存器（PSC）和自动重装载值寄存器 (ARR) 的值。

定时功能

初始化

基本的定时器功能只需要对结构体 `TIM_HandleTypeDef` 进行初始化即可

1. 初始化一个全局的 `TIM_HandleTypeDef` 结构体变量:

```
TIM_HandleTypeDef TimHandle;
```

2. 选择需要使用的硬件 Timer，例如需要使用 LSGPTIMA:

```
TimHandle.Instance = LSGPTIMA;
```

3. 根据应用需求填写相应的预分频系数、自动重装载值的大小、选择计数模式:

```
TimHandle.Init.Prescaler     = TIM_PRESCALER;
TimHandle.Init.Period        = TIM_PERIOD;
TimHandle.Init.CounterMode   = TIM_COUNTERMODE_UP;
```

4. 将初始化号的结构变量值配置到相应寄存器中:

```
HAL_TIM_Init(&TimHandle);
```

打开定时器

1. 需要产生中断的话，执行下述接口，定时器开始工作:

```
HAL_TIM_Base_Start_IT(&TimHandle);
```

2. 只要计数功能调用以下接口，定时器开始工作:

```
HAL_TIM_Base_Start(&TimHandle);
```

事件回调

1. 如果使能了中断,定时事件到了之后,会进入回调函数`void HAL_TIM_PeriodElapsedCallback(TIM_HandleTyp *htim);`用户可以在回调函数内处理应用逻辑
2. 如果单纯使用计数功能，用户可以使用宏 `__HAL_TIM_GET_COUNTER(&TimHandle)` 来获取当前计数值

关闭定时器

1. 如果使能了中断，需要关闭定时器时，调用函数:

```
HAL_StatusTypeDef HAL_TIM_Base_Stop_IT(TIM_HandleTypeDef *htim)
```

2. 如果只使用计数功能，需要关闭定时器时，调用函数:

```
HAL_StatusTypeDef HAL_TIM_Base_Start(TIM_HandleTypeDef *htim)
```

反初始化

反初始化 Timer 功能:

```
HAL_StatusTypeDef HAL_TIM_DeInit(TIM_HandleTypeDef *htim);
```

PWM 输出

初始化

1. 与定时功能里初始化步骤一样，需要先对定时器的时基部分进行配置
2. 根据选择使用的硬件 Timer，调用相应的接口函数初始化 PWM 使用到的相关 GPIO，比如需要使用 LSGPTIMB 的四个通道同时输出:

```
gptimb1_ch1_io_init(PA00, true, 0);
gptimb1_ch2_io_init(PA01, true, 0);
gptimb1_ch3_io_init(PB14, true, 0);
gptimb1_ch4_io_init(PB15, true, 0);
```

3. 初始化输出比较结构体 TIM_OC_InitTypeDef, 对指定定时器输出通道进行初始化配置
4. 调用以下接口完成输出通道的初始化配置:

```
HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel(TIM_HandleTypeDef *htim,TIM_OC_
↪InitTypeDef *sConfig,uint32_t Channel);
```

开始产生 PWM 脉冲

初始化配置完成之后，需要执行下述函数才会开始输出 PWM 波形:

```
HAL_StatusTypeDef HAL_TIM_PWM_Start(TIM_HandleTypeDef *htim, uint32_t Channel);
```

停止 PWM 输出

需要停止 PWM 时，调用以下函数接口:

```
HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT(TIM_HandleTypeDef *htim, uint32_t Channel);
```

反初始化

1. 反初始化 Timer 功能:

```
HAL_StatusTypeDef HAL_TIM_DeInit(TIM_HandleTypeDef *htim);
```

2. 调用相应的接口，对配置过的 IO 进行反初始化，比如对配置过的 LSGPTIMB 的四个通道的 IO 进行反初始化:

```
gptimb1_ch1_io_deinit();
gptimb1_ch2_io_deinit();
gptimb1_ch3_io_deinit();
gptimb1_ch4_io_deinit();
```

1.7.5 RTC

本节主要介绍 RTC 的常用功能和配置。RTC 的操作 API 详见 `lsrtc.h` 。RTC 的主要功能包括万年历功能和唤醒功能（LP0 & LP2 模式）。

一、初始化

使用 RTC 的功能，首先需要调用 RTC 的初始化函数:

```
void HAL_RTC_Init(uint8_t cksel);
```

其中参数 cksel 表示时钟源。如果使用外部 32K 晶振输入，该参数设置为 RTC_CKSEL_LSE，否则需要设置成 RTC_CKSEL_LSI。

二、功能配置

2.1 万年历功能

使用万年历功能之前，首先需要配置当前的年月日时分秒，参考代码如下:

```
static calendar_time_t calendar_time;
static calendar_cal_t calendar_cal;
static void rtc_calendar_set()
{
    calendar_cal.year = 24;
    calendar_cal.mon = 2;
    calendar_cal.date = 28;
    calendar_time.hour = 23;
```

(下页继续)

(续上页)

```
    calendar_time.min = 59;
    calendar_time.sec = 55;
    calendar_time.week = 7;
    RTC_CalendarSet(&calendar_cal,&calendar_time);
}
```

万年历能配置的最早年份为 2020 年，即 year 为 0。如上所示，初始化的时间点为 2024 年 2 月 28 日 23 点 59 分 55 秒周日，然后调用 RTC_CalendarSet 函数进行设置。之后万年历便开始运行，在休眠模式（LP0 & LP2）下也是如此。获取万年历当前数值的接口为：

```
HAL_StatusTypeDef RTC_CalendarGet(calendar_cal_t *calendar_cal, calendar_time_t␣
↪*calendar_time);
```

调用返回值为 HAL_OK 时，当前的万年历信息会从参数里返回出来。如果返回其他值，则表示返回值异常，需要重新调用此接口获取。

2.2 唤醒功能（LP0 模式）

在 LP0 模式下，可以通过配置 RTC wakeup 中断对系统进行定时唤醒。配置系统时间的接口如下：

```
void RTC_wkuptime_set(uint32_t second);
```

参数 second 为唤醒时间，单位为秒。当系统唤醒后，会调用 rtc_wkup_callback 函数：

```
void rtc_wkup_callback(void);
```

注意：该函数为全局函数，需要严格按照如上述形式实现，不可以加 static

2.3 唤醒模式（LP2 模式）

在 LP2 模式下，RTC 唤醒系统的接口和 LP0 模式一样：

```
void RTC_wkuptime_set(uint32_t second);
```

另外，进入 LP2 模式需要调用 enter_deep_sleep_mode_lvl2_lvl3()，如下：

```
struct deep_sleep_wakeup wakeup;
memset(&wakeup,0,sizeof(wakeup));
wakeup.rtc = 1;
enter_deep_sleep_mode_lvl2_lvl3(&wakeup);
```

wakeup.rtc = 1 是配置 RTC 为唤醒源。唤醒源还可以是某些 GPIO，NRST 或者 WDT 等。类似配置，此处不再赘述。与 LP0 模式不同的是，LP2 模式下 RTC 唤醒系统后不会调用 rtc_wkup_callback 函数。应用里，需要

通过调用 get_wakeup_source() 获取唤醒源信息：

```
uint8_t wkup_source = get_wakeup_source();
if ((RTC_WKUP & wkup_source) != 0)
{
    // do something for rtc wakeup
}
```

三、注意事项

1. RTC 定时唤醒系统不会很准，尤其是 LP2 模式。LP0 模式下 RTC 上中断到进入 rtc_wkup_callback 会有通常不超过 10ms 的误差，而 LP2 模式下由于系统会重启，因此误差往往在 100ms 以上

1.7.6 PDM

PDM（Pulse Density Modulation）是一种用数字信号表示模拟信号的调制方法。PDM 只有 1 位数据输出，要么为 0，要么为 1。

一、初始化

1. PDM 模块的 IO 配置

调用 IO 的初始化接口，可以将任意 IO 复用为 pdm 的 colck data0 或 data1 引脚。

```
void pdm_clk_io_init(uint8_t pin);
void pdm_data0_io_init(uint8_t pin);
void pdm_data1_io_init(uint8_t pin);
```

注解:

1. 芯片的 IO 一共有 34 个，具体情况需根据封装图来定义。

2. 为了避免不必要的 bug，在使用 pdm 通信的时候，请先初始化 IO，再进行下列参数的配置。

---

2. 初始化 PDM 模块

2.1 PDM 结构体参数原型如下：

```
typedef struct __PDM_Init_TypeDef
{
    const struct pdm_fir *fir;  /*!< pdm filter controller configure */
    PDM_CFG_TypeDef cfg;        /*!< pdm clock rate, capture delay, sampling rate,␣
↪and data gain configure */
    PDM_MODE_TypeDef mode;      /*!< pdm channel mode configure */
}PDM_Init_TypeDef;
```

2.2 调用初始化 PDM 模块函数接口

通过初始化接口，应用程序可以对 PDM 进行参数配置。

```
HAL_StatusTypeDef HAL_PDM_Init(PDM_HandleTypeDef *hpdm,PDM_Init_TypeDef *Init);
```

3.PDM 初始化参考代码如下：

```
#define PDM_CLK_KHZ 1024
#define PDM_SAMPLE_RATE_HZ 16000
void pdm_init()
{
    pdm_clk_io_init(PB10);  /*!< PB10 复用为 pdm clk 引脚 */
    pdm_data0_io_init(PB09);    /*!< PB09 复用为 pdm data0 引脚 */
    pdm.Instance = LSPDM;   /*!< PDM 外设的基址 */
    PDM_Init_TypeDef Init =
    {
        .fir = PDM_FIR_COEF_16KHZ,  /*!< 配置 PDM 的滤波控制器 */
        .cfg = {
            .clk_ratio = PDM_CLK_RATIO(PDM_CLK_KHZ),    /*!< 配置 PDM 的时钟频率为 1.
↪024MHZ */
            .sample_rate = PDM_SAMPLE_RATE(PDM_CLK_KHZ,PDM_SAMPLE_RATE_HZ), /*!< 配置
PDM 采样频率为 16KHZ */
            .cap_delay = 30,    /*!< 配置捕获延迟为 30 */
            .data_gain = 5,     /*!< 配置数据增益为 5 */
        },
        .mode = PDM_MODE_Mono,  /*!< 配置 PDM 为单通道模式 */
    };
    HAL_PDM_Init(&pdm,&Init);   /*!< 调用 PDM 初始化函数 */
}
```

二、反初始化

1. 反初始化 PDM 模块

通过反初始化接口，应用程序可以关闭 PDM 外设，在运行 BLE 程序时降低系统的功耗。

```
HAL_StatusTypeDef HAL_PDM_DeInit(PDM_HandleTypeDef *hpdm);
```

2. 反初始化 PDM IO

反初始化 IO 接口的主要目的是为了避免在进入低功耗模式时，IO 上产生漏电。

```
void pdm_clk_io_deinit(void);
void pdm_data0_io_deinit(void);
void pdm_data1_io_deinit(void);
```

注解： PDM 初始化动作会向系统注册 PDM 进入工作状态，当系统检测到有任一外设处于工作状态时，都不会进入低功耗休眠。因此，PDM 使用完毕，需要进入低功耗状态之前，必须反初始化 PDM。

三、PDM 相关函数接口

注解： 收 PDM 数据的模式分为 2 种：中断模式和 DMA 模式。

3.1 收 PDM 数据——中断方式

```
HAL_StatusTypeDef HAL_PDM_Transfer_Config_IT(PDM_HandleTypeDef *hpdm,uint16_t␣
↪*pFrameBuffer0,uint16_t *pFrameBuffer1,uint16_t FrameNum);
```

3.2 收 PDM 数据——DMA 方式

以 DMA 方式 (基本模式和乒乓模式) 收 PDM 数据如下所示：

```
HAL_StatusTypeDef HAL_PDM_Transfer_Config_DMA(PDM_HandleTypeDef *hpdm,uint16_t␣
↪*pFrameBuffer0,uint16_t *pFrameBuffer1,uint16_t FrameNum);
HAL_StatusTypeDef HAL_PDM_PingPong_Transfer_Config_DMA(PDM_HandleTypeDef *hpdm,struct␣
↪PDM_PingPong_Bufptr *CH0_Buf,struct PDM_PingPong_Bufptr *CH1_Buf,uint16_t FrameNum);
```

3.3 使能 PDM

```
HAL_StatusTypeDef HAL_PDM_Start(PDM_HandleTypeDef *hpdm);
```

3.4 失能 PDM

```
HAL_StatusTypeDef HAL_PDM_Stop(PDM_HandleTypeDef *hpdm);
```

3.5 PDM 中断处理函数

```
void HAL_PDM_IRQHandler(PDM_HandleTypeDef *hpdm);
```

3.6 在 PDM 中断处理函数中接收完 FrameNum 大小数据的回调函数

```
void HAL_PDM_CpltCallback(PDM_HandleTypeDef *hpdm);
```

3.7 在 DMA 模式下接收完 FrameNum 大小 pdm 数据的回调函数

```
void HAL_PDM_DMA_CpltCallback(PDM_HandleTypeDef *hpdm,uint8_t buf_idx);
```

1.7.7 ADC

ADC 是 Analog-to-Digital Converter 的缩写。指模/数转换器或者模拟/数字转换器。是指将连续变量的模拟信号转换为离散的数字信号的器件。

ADC 特性:

1. 可配置的转换精度（6/8/10/12 位）
2. 在规则转换、注入转换结束后以及发生模拟看门狗或溢出事件时产生中断
3. 支持单次或连续转换模式
4. 用于自动将通道 0 转换为通道“n”的扫描模式
5. 可配置的数据对齐方式
6. 可独立设置各通道采样时间
7. 支持不连续采样模式
8. 可配置的参考源选择
9. 可配置的转换时钟分频
10. 规则通道转换期间可产生 DMA 请求

ADC 系统框图如下：

一、初始化

1.1 设置 ADC 模块的 IO

调用 IO 的初始化接口，可以配置具体的 IO 的 ADC 模拟功能。

```
void adc12b_in0_io_init(void);
void adc12b_in1_io_init(void);
void adc12b_in2_io_init(void);
void adc12b_in3_io_init(void);
void adc12b_in4_io_init(void);
void adc12b_in5_io_init(void);
void adc12b_in6_io_init(void);
void adc12b_in7_io_init(void);
void adc12b_in8_io_init(void);
```

注解： 每一个 ADC 的通道都有其对应的 IO，不可随意映射。具体的对应关系如下：

GPIO	ANA_FUNC1
GPIO_PB12	ADC12B_AIN[0]
GPIO_PB13	ADC12B_AIN[1]
GPIO_PC00	ADC12B_AIN[2]
GPIO_PC01	ADC12B_AIN[3]
GPIO_PA00	ADC12B_AIN[4]
GPIO_PA01	ADC12B_AIN[5]
GPIO_PA02	ADC12B_AIN[6]
GPIO_PA03	ADC12B_AIN[7]
GPIO_PA04	ADC12B_AIN[8]

另外 ADC 模块还有对应的内部通道，分布如下：

内部输入信号	内部输入通道
芯片工作温度	ADC12B_AIN[9]
芯片工作电压 Vbat 信号	ADC12B_AIN[10]
芯片内部 ADC 参考电压 (标准 1.4v)	ADC12B_AIN[11]

1.2 设置 ADC 模块的参数变量

设置 ADC 模块的参数变量，其结构体的参数原型如下：

```
/**
  * @struct __ADC_HandleTypeDef
  * @brief  ADC handle type Structure definition
  */
typedef struct __ADC_HandleTypeDef
{
    reg_adc_t       *Instance; /*!< Register base address */
    ADC_InitTypeDef Init;
    void            *DMAC_Instance;
    union{
        struct AdcDMAEnv DMA;
        struct AdcInterruptEnv Interrupt;
    }Env;
    HAL_LockTypeDef Lock;
    volatile uint32_t State;
    volatile uint32_t ErrorCode;

} ADC_HandleTypeDef;
```

参数说明

（1）ADC 寄存器结构化处理 (Instance)

- 目前 LE5010/LE5110 仅支持一个 ADC
- ADC 基地址 0x40012400

（2）ADC 初始化处理 (Init)

- 请参见 ADC 初始化 1.2.1

（3）DMA 寄存器结构化处理 (DMAC_Instance)

- 请参见 DMA 初始化 1.2.2

（4）ADC 转换的触发方式 (Env)

- ADC-DMA 触发方式

  设置 DMA 通道和 Callback 函数 (规则组转换)

```
/**
  * @struct AdcDMAEnv
  * @brief  ADC DMA Structure definition
  */
 struct AdcDMAEnv
```

(下页继续)

(续上页)

```
{
    void                            (*Callback)();
    uint8_t                         DMA_Channel;
};
```

- ADC 中断单次触发方式

  设置读取 ADC 数据的变量 (单次转换)

```
/**
  * @struct AdcInterruptEnv
  * @brief  ADC Interrupt Structure definition
  */
struct AdcInterruptEnv
{
    uint8_t                         *pBuffPtr;      /*!< Pointer to ADC data Buffer␣
↪*/
    uint16_t                        XferCount;      /*!< UART ADC data Counter */
};
```

1.2.1ADC 初始化

```
/**
  * @struct  ADC_InitTypeDef
  * @brief  Structure definition of ADC and regular group␣
↪initialization
  * @note   Parameters of this structure are shared within 2 scopes:
  *          - Scope entire ADC (affects regular and injected␣
↪groups): DataAlign, ScanConvMode.
  *          - Scope regular group: ContinuousConvMode,␣
↪NbrOfConversion, DiscontinuousConvMode, NbrOfDiscConversion,
↪ExternalTrigConv.
  * @note   The setting of these parameters with function HAL_ADC_
↪Init() is conditioned to ADC state.
  *         ADC can be either disabled or enabled without conversion␣
↪on going on regular group.
  */
typedef struct
{
    uint32_t DataAlign;
    uint32_t ScanConvMode;
    FunctionalState ContinuousConvMode;
    uint32_t NbrOfConversion;
```

(下页继续)

(续上页)

```
    FunctionalState DiscontinuousConvMode;
    uint32_t NbrOfDiscConversion;
    uint32_t TrigType;
    uint32_t Vref;
    uint32_t AdcDriveType;
    uint32_t AdcCkDiv;
} ADC_InitTypeDef;
```

- 参数说明：

（1）数据对齐 (DataAlign)

- 默认情况，ADC 转换后的数据采用右对齐方式 (bit11:0)。
- 设置左对齐方式，ADC 转换后的数据：
- 规则组转换数据寄存器 (ADC_RDR)
- 注入组转换数据寄存器 ((ADC_JDRx)： *Raw Converted Data* + *InjectOffset*

（2）扫描模式 (ScanConvMode)

- 禁止

  单通道单次转换

  参数：NbrOfConversion 无效

  参数：NbrOfDiscConversion 无效

- 使能

  会扫描所有规则通道。

  与 ContinuousConvMode 的联动：

  使能 ContinuousConvMode，会连续采集所有通道，从 rank1 开始扫描，到最后一个 rank。

  禁止 ContinuousConvMode，只会扫描一轮，从 rank1 开始扫描，到最后一个 rank。

（3）连续转换模式 (ContinuousConvMode)

- 禁止

  单通道单次转换

- 使能

  连续多通道转换，与 NbrOfConversion 相对应。

（4）连续转换的次数 (NbrOfConversion)

- 规则组序列转换
- 转换次数范围：1 ~ 12

- 参数：ScanConvMode 必须使能

（5）间断转换模式 (DiscontinuousConvMode)

- 规则组子序列转换
- 参数：ScanConvMode 必须禁止
- 参数：ContinuousConvMode 必须禁止
- 具体每次采集的个数与 (NbrOfDiscConversion) 相对应。

（6）间断转换的次数 (NbrOfDiscConversion)

- 规则组转换子序列数
- 转换次数范围：≤8
- 参数：DiscontinuousConvMode 必须使能

（7）触发转换的方式 (TrigType)

触发转换的方式	TrigType
PIS	ADC_PIS_TRIG
软件规则组触发	ADC_REGULAR_SOFTWARE_TRIGT
软件注入组触发	ADC_INJECTED_SOFTWARE_TRIGT

（8）选择参考电压 (Vref)

选择参考电压	Vref
默认芯片内部 1.4V 为参考电压	ADC_VREF_INSIDE
PA05 输入参考电压	ADC_VREF_EXPOWER
芯片工作电压 AVDD 为参考电压	ADC_VREF_VCC

注解： 当选择外部 IO 为参考电压的时候，需要特殊配置 PA05。

（9）ADC 通道的驱动方式 (AdcDriveType)

ADC 通道的驱动方式	AdcDriveType
输入信号经过输入 buf 运放驱动 ADC	EINBUF_DRIVE_ADC
输入信号 1/3 分压，并经过输入 buf 运放驱动 ADC	INRES_ONETHIRD_EINBUF_DRIVE_ADC
默认关闭输入 buf 运放，输入信号直接驱动 ADC	BINBUF_DIRECT_DRIVE_ADC

（10）ADC 时钟分频系数 (AdcCkDiv)

- 系统时钟按 AdcCkDiv 分频获得 ADC 运行时钟，默认 ADC 时钟为 APBCLK 的 32 分频，可以选择。

```
#define ADC_CLOCK_DIV2          0x00000001U
#define ADC_CLOCK_DIV4          0x00000002U
#define ADC_CLOCK_DIV8          0x00000003U
#define ADC_CLOCK_DIV16         0x00000004U
#define ADC_CLOCK_DIV32         0x00000005U
#define ADC_CLOCK_DIV64         0x00000006U
#define ADC_CLOCK_DIV128        0x00000007U
```

1.2.2 ADC 采集通道的初始化

规则组转换参数配置

```
/**
* @struct ADC_ChannelConfTypeDef
* @brief  Structure definition of ADC channel for regular group
* @note   The setting of these parameters with function HAL_ADC_
↪ConfigChannel() is conditioned to ADC state.
*         ADC can be either disabled or enabled without conversion␣
↪on going on regular group.
*/
typedef struct
{
  uint32_t     Channel;
  uint32_t     Rank;
  uint32_t     SamplingTime;
} ADC_ChannelConfTypeDef;
```

- 参数说明

1. 规则通道 (Channel)

   - 采样通道说明:

```
#define ADC_CHANNEL_0               0x00000000U
#define ADC_CHANNEL_1               0x00000001U
#define ADC_CHANNEL_2               0x00000002U
#define ADC_CHANNEL_3               0x00000003U
#define ADC_CHANNEL_4               0x00000004U
#define ADC_CHANNEL_5               0x00000005U
#define ADC_CHANNEL_6               0x00000006U
#define ADC_CHANNEL_7               0x00000007U
```

(下页继续)

(续上页)

```
#define ADC_CHANNEL_8                0x00000008U
#define ADC_CHANNEL_TEMPSENSOR       0x00000009U      /* ADC internal␣
↪channel (no connection on device pin) */
#define ADC_CHANNEL_VBAT             0x0000000AU      /* ADC internal␣
↪channel (no connection on device pin) */
#define ADC_CHANNEL_VREFINT          0x0000000BU      /* ADC internal␣
↪channel (no connection on device pin) */
```

2. 规则转换序列 (Rank)

   - 规则组序列说明:

```
#define ADC_REGULAR_RANK_1           0x00000001U
#define ADC_REGULAR_RANK_2           0x00000002U
#define ADC_REGULAR_RANK_3           0x00000003U
#define ADC_REGULAR_RANK_4           0x00000004U
#define ADC_REGULAR_RANK_5           0x00000005U
#define ADC_REGULAR_RANK_6           0x00000006U
#define ADC_REGULAR_RANK_7           0x00000007U
#define ADC_REGULAR_RANK_8           0x00000008U
#define ADC_REGULAR_RANK_9           0x00000009U
#define ADC_REGULAR_RANK_10          0x0000000AU
#define ADC_REGULAR_RANK_11          0x0000000BU
#define ADC_REGULAR_RANK_12          0x0000000CU
```

3. 规则转换采样周期 (SamplingTime)

   - 采样周期说明:

```
#define ADC_SAMPLETIME_1CYCLE        0x00000000U                /*!<␣
↪Sampling time 1 ADC clock cycle */
#define ADC_SAMPLETIME_2CYCLES       0x00000001U                /*!<␣
↪Sampling time 2 ADC clock cycles */
#define ADC_SAMPLETIME_4CYCLES       0x00000002U                /*!<␣
↪Sampling time 4 ADC clock cycles */
#define ADC_SAMPLETIME_15CYCLES      0x00000003U                /*!<␣
↪Sampling time 15 ADC clock cycles */
```

- ADC 规则转换 API 函数

```
HAL_StatusTypeDef HAL_ADC_ConfigChannel(ADC_HandleTypeDef *hadc, ADC_
↪ChannelConfTypeDef *sConfig);
```

注入组转换参数配置

```
/**
  * @struct ADC_InjectionConfTypeDef
  * @brief  ADC Configuration injected Channel structure definition
  * @note   Parameters of this structure are shared within 2 scopes:
  *          - Scope channel: InjectedChannel, InjectedRank,␣
↪InjectedSamplingTime, InjectedOffset
  *          - Scope injected group (affects all channels of␣
↪injected group): InjectedNbrOfConversion,␣
↪InjectedDiscontinuousConvMode,
  *            AutoInjectedConv, ExternalTrigInjecConv.
  */
typedef struct
{
  uint32_t InjectedChannel;
  uint32_t InjectedRank;
  uint32_t InjectedSamplingTime;
  uint32_t InjectedOffset;
  uint32_t InjectedNbrOfConversion;
  FunctionalState InjectedDiscontinuousConvMode;
  FunctionalState AutoInjectedConv;
}ADC_InjectionConfTypeDef;
```

- 参数说明：

1. 注入通道 (InjectedChannel)

   与规则通道一致，请参考规则通道

2. 注入转换序列 (Rank)

   注入组序列说明：

```
#define ADC_INJECTED_RANK_1                0x00000001U
#define ADC_INJECTED_RANK_2                0x00000002U
#define ADC_INJECTED_RANK_3                0x00000003U
#define ADC_INJECTED_RANK_4                0x00000004U
```

3. 注入转换采样周期 (SamplingTime)

   与规则转换采样周期一致请参考规则转换采样周期

4. 注入转换数据偏移量 (InjectedOffset)

   - 该偏移量为有符号数，其中 bit11 表示符号位
   - 注入组转换数据寄存器 ((ADC_JDRx)：*Raw Converted Data + JnjectOffset*

5. 注入转换的次数 (InjectedNbrOfConversion)
    - 注入转换序列子序列数
    - 范围：1 ~ 4
    - 参数：ScanConvMode 必须使能。
6. 注入序列间断转换模式 (InjectedDiscontinuousConvMode)
    - 参数：ScanConvMode 必须禁止
    - 参数：ContinuousConvMode 必须禁止
7. 自动注入转换模式（AutoInjectedConv）
    - 参数：DiscontinuousConvMode 必须禁止
    - 参数：InjectedDiscontinuousConvMode 必须禁止

- ADC 注入转换 API 函数

```
HAL_StatusTypeDef  HAL_ADCEx_InjectedConfigChannel(ADC_HandleTypeDef* hadc,
↪ADC_InjectionConfTypeDef* sConfigInjected);
```

1.3 初始化 ADC 模块

```
HAL_StatusTypeDef HAL_ADC_Init(ADC_HandleTypeDef *hadc);
```

二、反初始化

2.1 反初始化 ADC 模块

通过反初始化函数，根据场景需求可以关闭 ADC 模块，可以降低系统的功耗。

```
HAL_StatusTypeDef HAL_ADC_DeInit(ADC_HandleTypeDef *hadc);
```

2.2 反初始化 ADC IO

根据场景需求通过反初始化函数，可以关闭 ADC 模块，对应的模拟 IO 反初始为普通 GPIO。

```
void adc12b_in0_io_deinit(void);
void adc12b_in1_io_deinit(void);
void adc12b_in2_io_deinit(void);
void adc12b_in3_io_deinit(void);
void adc12b_in4_io_deinit(void);
```

(下页继续)

(续上页)

```
void adc12b_in5_io_deinit(void);
void adc12b_in6_io_deinit(void);
void adc12b_in7_io_deinit(void);
void adc12b_in8_io_deinit(void);
```

注解： 由于 ADC 外部输入电压的不确定性，不好配置内部 IO 的状态，所以在使用 ADC 功能，在进入休眠之后，IO 内部电平状态与外部输入电压易产生压差，导致出现部分漏电。

三、ADC 模块采集数据

ADC 模块采集数据我们一共提供了三种接口，规则通道采集，注入通道采集和 DMA 采集。在配置完初始化相关信息之后，需要调用相应的 API 接口，让 ADC 模块开始工作。

3.1 数据采集——规则通道

```
HAL_StatusTypeDef HAL_ADC_Start_IT(ADC_HandleTypeDef *hadc);
```

3.2 数据采集——注入通道

```
HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT(ADC_HandleTypeDef* hadc);
```

3.3 数据采集——DMA 模式

```
HAL_StatusTypeDef HAL_ADC_Start_DMA(ADC_HandleTypeDef* hadc, uint16_t* pData, uint32_
↪t Length,void (*Callback)());
```

示例代码：

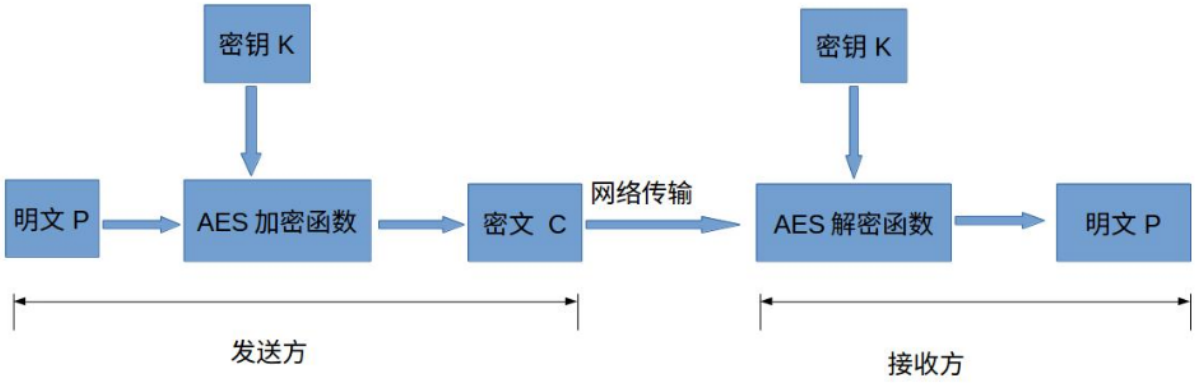
参考：<install_file>/dev/examples/adc_test/adc_single_channel

1.7.8 CRYPT

一、CRYPT 简介

硬件加密模块主要用于由硬件对数据进行加密或解密操作，支持的标准有 AES、DES。AES（Advanced Encryption Standard）是最新的分组对称密码算法，兼容联邦信息处理标准出版物（FIPSPUB197，2001 年 11 月 26 日）规定的高级加密标准（AES）。DES（Data Encryption Standard）是应用非常广泛的对称密码算法，兼

容联邦信息处理标准出版物（FIPS PUB 46-3，1999 年 10 月 25 日）规定的数据加密标准（DES）和三重 DES（TDES），遵循美国国家标准协会（ANSI）X9.52 标准。



AES 支持 128、192 和 256 位的密钥支持 ECB、CBC 模式支持 1、8、16 或 32 位数据交换

DES/TDES 支持 ECB、CBC 模式支持 64、128 和 192 位密钥支持在 CBC 模式下使用的 2×32 位初始化向量（IV）支持 1、8、16 或 32 位数据交换

AES-ECB 模式加密: 在 AES-ECB 加密中，执行位/字节/半字交换后，128 位明文数据块将用作输入块。输入块通过 AEA 在加密状态下使用 128 位、192 位或 256 位密钥进行处理。执行位/字节/半字交换后，生成的 128 位输出块将用作密文。该密文随后给到结果寄存器。

AES-ECB 模式解密: 要在 ECB 模式下执行 AES 解密，需要准备密钥（需要针对加密执行完整的密钥计划），方法为: 收集最后一个轮密钥并将其用作解密密文的第一个轮密钥。该准备过程通过 AES 内核计算完成。在 AES-ECB 解密中，执行位/字节/半字交换后，128 位密文块将用作输入块。该密钥序列与加密处理中的密钥序列相反。执行位/字节/半字交换，生成的 128 位输出块将产生明文。

AES-CBC 模式加密: 在 AES-CBC 加密中，执行位/字节/半字交换后所获得的第一个输入块是通过一个 128 位初始化向量 IV 与第一个明文数据块进行异或运算形成的。输入块通过 AEA 在加密状态下使用 128 位、192 位或 256 位密钥进行处理。生成的 128 位输出块将直接用作密文。然后，第一个密文块与第二个明文数据块进行异或运算，从而生成第二个输入块。第二个输入块通过 AEA 处理而生成第二个密文块。此加密处理会不断将后续密文块和明文块链接到一起，直到消息中最后一个明文块得到加密为止。如果消息中包含的数据块数不是整数，则应按照应用程序指定的方式对最后的不完整数据块进行加密。在 CBC 模式下（如 ECB 模式），必须准备密钥才可以执行 AES 解密。

AES-CBC 模式解密: 在 AES-CBC 解密中，第一个 128 位密文块将直接用作输入块。输入块通过 AEA 在解密状态下使用 128 位、192 位或 256 位密钥进行处理。生成的输出块与 128 位初始化向量 IV（必须与加密期间使用的相同）进行异或运算，从而生成第一个明文块。然后，第二个密文块将用作下一个输入块，并由 AEA 进行处理。生成的输出块与第一个密文块进行异或运算，从而生成第二个明文数据块。AES-CBC 解密过程将以此方式继续进行，直到最后一个完整密文块得到解密为止。必须按照应用程序指定的方式对不完整数据块密文进行解密。

二、CRYPT 接口介绍

2.1 初始化:

首先需要进行 CRYPT 模块初始化。

```
HAL_LSCRYPT_Init(void);
```

由于在进行加密解密的过程中需要使用到一个相同的 key，这个 key 用来加密明文的密码，在对称加密算法中，加密与解密的密钥是相同的。密钥为接收方与发送方协商产生，但不可以直接在网络上传输，否则会导致密钥泄漏，最终导致明文被还原。所以还需要输入一个 key。如下代码所示：

```
HAL_LSCRYPT_AES_Key_Config(const uint32_t *key, enum aes_key_type key_size);
```

在加密中我们可以有三种 key 的类型，如下代码所示：

```
enum aes_key_type
{
    AES_KEY_128 = 0,
    AES_KEY_192,
    AES_KEY_256,
};
```

2.2 CRYPT 加解密:

参数描述

注解:

1. plaintext: 明文数据，没有经过加密过后的数据。
2. ciphertext: 密文数据，经过加密过后的数据
3. iv:CBC 模式下使用的 2×32 位初始化向量（IV）

2.2.1 轮询模式

```
HAL_LSCRYPT_AES_ECB_Encrypt(const uint8_t *plaintext,uint32_t length,uint8_t␣
↪*ciphertext);
HAL_LSCRYPT_AES_ECB_Decrypt(const uint8_t *ciphertext,uint32_t length,uint8_t␣
↪*plaintext);
/*ECB 模式下以轮询模式对数据进行加密与解密 */
HAL_LSCRYPT_AES_CBC_Encrypt(const uint32_t iv[4],const uint8_t *plaintext,uint32_t␣
↪length,uint8_t *ciphertext);
```

(下页继续)

(续上页)

```
HAL_LSCRYPT_AES_CBC_Decrypt(const uint32_t iv[4],const uint8_t *ciphertext,uint32_t␣
↪length,uint8_t *plaintext);
/*CBC 模式下以轮询模式对数据进行加密与解密 */
```

2.2.2 中断模式

```
HAL_LSCRYPT_AES_ECB_Encrypt_IT(const uint8_t *plaintext,uint32_t length,uint8_t␣
↪*ciphertext);
HAL_LSCRYPT_AES_ECB_Decrypt_IT(const uint8_t *ciphertext,uint32_t length,uint8_t␣
↪*plaintext);
/*ECB 模式下以中断模式对数据进行加密与解密 */
HAL_LSCRYPT_AES_CBC_Encrypt_IT(const uint32_t iv[4],const uint8_t *plaintext,uint32_t␣
↪length,uint8_t *ciphertext);
HAL_LSCRYPT_AES_CBC_Decrypt_IT(const uint32_t iv[4],const uint8_t *ciphertext,uint32_
↪t length,uint8_t *plaintext);
/*CBC 模式下以中断模式对数据进行加密与解密 */
```

2.2.3 回调函数

```
HAL_LSCRYPT_AES_Complete_Callback(bool Encrypt,bool CBC);
{

}
```

注解:

函数中的“Encrypt”为 true 时选择的是加密模式（Encrypt），当为 false 时选择的是解密模式（Decrypt）。
函数中的“CBC”为 true 时选择的是 CBC 模式，当为 false 时选择的是 ECB 模式。
这个函数属于弱定义，用户可以自行定义，并完成相应的逻辑处理。

2.3 反初始化

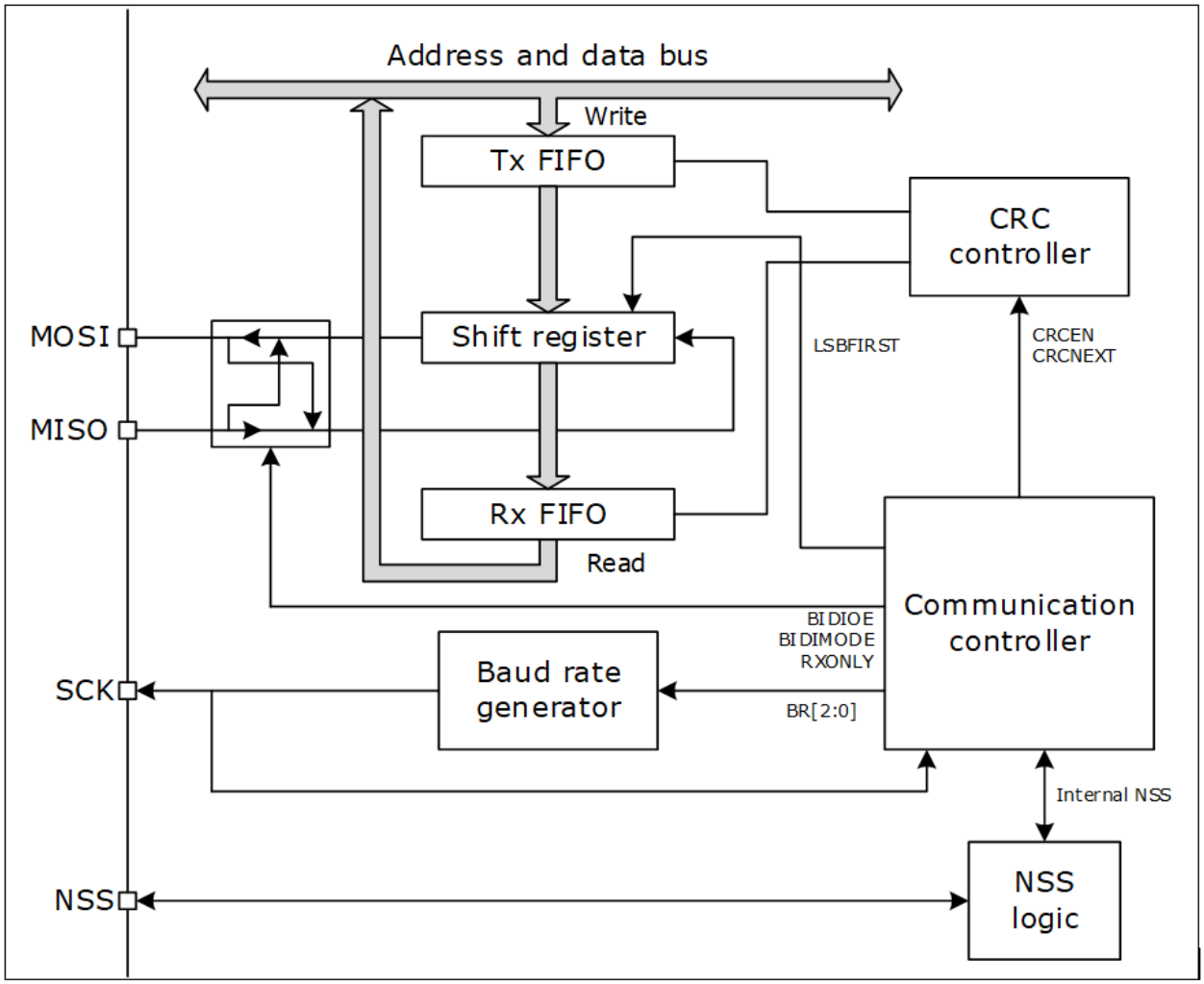
反初始化 CRYPT 模块

通过反初始化接口，应用程序可以关闭 CRYPT 外设，从而在运行 BLE 的程序的时候，降低系统的功耗。

```
HAL_LSCRYPT_DeInit(void);
```

1.7.9 SPI

串行外设接口（SPI）协议，支持半双工，全双工和简单同步等方式与外部设备的串行通信。



SPI 主要特性

1. 主设备或从设备模式
2. 三线全双工同步传输
3. 两条线的半双工同步传输（双向数据线）
4. 两条线的简单同步传输（单向数据线）
5. 8 位到 16 位数据的大小选择
6. 8 个主模式波特率分频器。
7. 主机和从机模式下都可以由硬件或软件管理 NSS：主/从模式操作的动态变化
8. 可编程时钟极性和相位

9. 高位在前或低位在前可设置
10. 专用的发送和接收状态标志，全部支持中断触发
11. SPI 总线忙状态标志
12. 支持 Rx 和 Tx FIFO，深度是 16
13. 支持 DMA 传输
14. 支持 SPI TI 模式

一、SPI 模块初始化

1.1 选择 SPI 模块的复用 IO

芯片支持该模块复用 IO 全映射，所以用户可以通过调用 IO 的初始化接口，将任意 IO 配置 CLK、MOSI、MISO、及 NSS 功能，与外围设备进行通信。

```
void spi2_clk_io_init(uint8_t clk);
void spi2_nss_io_init(uint8_t nss);
void spi2_mosi_io_init(uint8_t mosi);
void spi2_miso_io_init(uint8_t miso);
```

注解： SNN 是 SPI 片选脚，当 SPI 为主机时，也可以用普通 IO 替代，如果工程师需要用 IO 功能替代时，可以不初始化该复用 IO。

1.2 设置 SPI 模块参数

在使用 SPI 模块前，需要设置 SPI 模块的参数，其结构体的参数原型如下：

```
typedef struct __SPI_HandleTypeDef
{
    reg_spi_t                   *Instance;                  /*!< SPI registers base␣
↪address                */
    SPI_InitTypeDef             Init;                       /*!< SPI communication␣
↪parameters             */
    uint8_t                     *pTxBuffPtr;                /*!< Pointer to SPI Tx␣
↪transfer Buffer        */
    uint16_t                    TxXferSize;                 /*!< SPI Tx Transfer size ␣
↪                   */
    uint16_t                    TxXferCount;                /*!< SPI Tx Transfer␣
↪Counter                  */
```

(下页继续)

(续上页)

```
    uint8_t                       *pRxBuffPtr;              /*!< Pointer to SPI Rx␣
↪transfer Buffer        */
    uint16_t                      RxXferSize;               /*!< SPI Rx Transfer size ␣
↪                    */
    uint16_t                      RxXferCount;              /*!< SPI Rx Transfer␣
↪Counter                   */
    void (*RxISR)(struct __SPI_HandleTypeDef *hspi);        /*!< function pointer on␣
↪Rx ISR                */
    void (*TxISR)(struct __SPI_HandleTypeDef *hspi);        /*!< function pointer on␣
↪Tx ISR                */
    void                         *DMAC_Instance;
    union{
            struct SPI_DMA_Env DMA;
    }Tx_Env,Rx_Env;
    HAL_LockTypeDef               Lock;                     /*!< Locking object       ␣
↪                    */
    HAL_SPI_StateTypeDef          State;                    /*!< SPI communication␣
↪state                   */
    uint32_t                      ErrorCode;                /*!< SPI Error code       ␣
↪                    */

} SPI_HandleTypeDef;
```

SPI 模块具体参数设置可参阅该模块 lsspi.h 文件。

1.3 初始化 SPI 模块

通过初始化接口，实现用户对 SPI 模块进行参数配置。

```
HAL_StatusTypeDef HAL_SPI_Init(SPI_HandleTypeDef *hspi);
```

如果初始化成功后便可以返回值为 HAL_OK，否则为 HAL_ERROR。

二、反初始化

2.1 反初始化 SPI 模块

通过反初始化接口，应用程序可以关闭 SPI 模块，从而在运行 BLE 的程序的时候，降低系统的功耗。

```
HAL_StatusTypeDef HAL_SPI_DeInit(SPI_HandleTypeDef *hspi);
```

2.2 反初始化复用 IO

反初始化 IO 接口的主要目的是为了避免在进入低功耗模式时，IO 上产生漏电，或者给对接设备发送不必要的数据。调用 SPI IO 反初始化接口后, 会将 SPI 复用 IO 恢复回 GPIO 功能。

```
void spi2_clk_io_deinit(void);
void spi2_nss_io_deinit(void);
void spi2_mosi_io_deinit(void);
void spi2_miso_io_deinit(void);
```

注解： SPI 初始化动作会向系统注册 UART 进入工作状态，当系统检测到有任一外设处于工作状态时，都不会进入低功耗休眠。因此，应用中如 SPI 使用完毕，需要进入低功耗状态之前，必须反初始化 SPI。

三、SPI 模块通信

1. 支持主机和从机两种模式；
2. 每种模式均支持 3 种通信方法：阻塞方式、非阻塞（中断）方式、DMA 方式。在使用的时候，这 3 种方式只能选其一。
3. 每种通信方式均支持单收、单发、收发同步接口。

3.1 阻塞方式

以阻塞方式使用 SPI 设备的 API 接口如下所示：

```
HAL_StatusTypeDef HAL_SPI_Transmit(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t␣
↪Size, uint32_t Timeout);
HAL_StatusTypeDef HAL_SPI_Receive(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t␣
↪Size, uint32_t Timeout);
HAL_StatusTypeDef HAL_SPI_TransmitReceive(SPI_HandleTypeDef *hspi, uint8_t *pTxData,␣
↪uint8_t *pRxData, uint16_t Size, uint32_t Timeout);
```

顾名思义，阻塞方式是指应用软件调用该接口后，CPU 需要等待本次通信完成后才退出，会一定程度上降低 CPU 的利用率。

3.2 非阻塞（中断）方式

以中断方式使用 SPI 设备的 API 接口如下所示：

```
HAL_StatusTypeDef HAL_SPI_Transmit_IT(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_
↪t Size);
HAL_StatusTypeDef HAL_SPI_Receive_IT(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_
↪t Size);
HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT(SPI_HandleTypeDef *hspi, uint8_t␣
↪*pTxData, uint8_t *pRxData, uint16_t Size);
```

3.3 DMA 方式

以 DMA 方式使用 SPI 设备的 API 接口如下所示：

```
HAL_StatusTypeDef HAL_SPI_Transmit_DMA(SPI_HandleTypeDef *hspi,void *Data,uint16_t␣
↪Count);
HAL_StatusTypeDef HAL_SPI_Receive_DMA(SPI_HandleTypeDef *hspi,void *Data,uint16_t␣
↪Count);
HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA(SPI_HandleTypeDef *hspi,void *TX_Data,
↪void *RX_Data,uint16_t Count);
```

注解： DMA 方式在使用时需要注意：DMA 只支持访问特定的 8Kram（0xa000~0xbFFF），所以，DMA 使用的变量或数组必须指定在这 8K 内，建议用户在申请变量时加 DMA_RAM_ATTR 限定。例如：DMA_RAM_ATTR uint8_t aTxBuffer[BUFFERSIZE];

四、代码示例

示例代码请参阅下面章节：应用说明->SPI 设备使用示例。

1.8 Firmware OTA

1.8.1 OTA 分类

前台 OTA OTA 功能以独立的程序映像烧录在芯片 Flash 特定地址，独立于应用程序映像。在实施 OTA 升级前，需要跳出应用，重启执行 OTA 功能。在 OTA 过程中，不能完成原有应用功能，只有 OTA 状态呈现，因此称作 *前台 OTA* 。

后台 OTA OTA 功能与应用编译在同一程序映像中。实施 OTA 升级时，原有应用功能依然可以运行，因此称作 *后台 OTA* 。

双区 OTA 可用于应用程序存储的 Flash 空间一分为二。一部分存储当前运行的应用固件，另一部分存新固件。当新固件全部收到之后，重启，将老固件擦除，新固件搬移到老固件所在的位置。这种方式固件存储区一份为二，因此称作 *双区 OTA* 。这种方式的好处在于，若新固件接收过程中出现异常，老固件可以重新运行。

单区 OTA 当应用程序固件较大，可用于应用程序存储的 Flash 空间无法容纳两份应用程序固件的时候。固件升级只能通过 *单区 OTA* 的方式进行。在固件升级前，先进入 OTA 模式，擦除原有固件，接收新固件，若接收失败，则继续维持 OTA 状态，必须重新接收新固件。

综合上述 4 个概念，可行的 OTA 方式有：`前台单区 OTA`，`前台双区 OTA`，`后台双区 OTA`。这三种方式我们均支持。

1.8.2 关键概念

Flash 可用空间 Flash 会存放信息页，Bootloader，协议栈，持久数据和 OTA 功能映像等内容，在这些区域之外，Flash 剩下的大片连续区域称作 *Flash 可用空间*

固件大小最大值 开发者需要对应用程序当前版本和后续一切新版本的程序映像大小最大值有一个估计。双区 OTA 的条件是 `Flash 可用空间大小 > 2 * 固件大小最大值`

应用程序映像地址、新固件下载地址 在单区 OTA 中：`新固件下载地址 == 应用程序映像地址`

在双区 OTA 中：`新固件下载地址 > 应用程序映像地址 + 应用程序大小`并且`新固件不能超出 Flash 可用空间范围`

OTA 功能程序映像地址（仅针对前台 OTA） 前台 OTA 的 OTA 功能程序存放地址和执行地址均不同于用户应用程序。OTA 功能程序既不能覆盖 Flash 其他区域信息，又要确保尽可能留出更大的 Flash 可用空间，因此在编写和链接的时候，要特别注意程序大小及链接地址。

1.8.3 固件签名

对用于升级的固件有安全性要求的场景下，可以使用固件签名。启用固件签名功能后，SBL 会检查签名是否合法，如果不合法，则拒绝升级，以此保障新固件的来源可靠。

固件签名采用 ECDSA 算法。

1. 利用 `tools/signing/key_gen.py` 生成一对密钥，包含公钥、私钥。
2. SBL 中，定义宏 `FW_ECC_VERIFY` 为 1，并公钥拷贝到 `sbl/pub_key.c` 文件中，编译生成带有验证签名功能的 SBL。
3. 新固件生成后，利用 `tools/signing/signing.py` 生成固件签名文件。手机 OTA 升级时，除了选择固件文件之外，还要选择此签名文件。

CHAPTER 2

应用说明

2.1 AT 串口指令

AT 指令是指在命令模式下用户通过 UART 与模块进行命令传递，以达到查询或者设置模块的某些配置。

2.1.1 一、串口配置

默认使用 PB00、PB01 作为 AT 指令的通讯串口，其中 PB00 为模块的 TX，PB01 为模块的 RX，默认的 uart 口的配置参数为：波特率 115200、无校验、8 位数据位、1 位停止位。

2.1.2 二、AT 指令格式

AT+ 指令可以直接通过常用的串口的调试助手程序进行输入，AT+ 指令采用基于 ASCII 码的命令。

2.1 格式说明

< >: 表示必须包含的部分

[ ]: 表示可选部分

2.2 命令消息

AT+<CMD>[op][para-1,para-2,para-3,para-4…]<CR><LF>

AT+: 命令消息前缀

[op]: 指令操作符，指定是参数设置或查询；= 表示参数设置，？表示查询

[para-n]: 参数设置时的输入，如果查询则不需要

<CR>: 结束符，回车，ASCII 码 0x0D

<LF>: 结束符，回车，ASCII 码 0x0A

2.3 响应消息

<CR><LF>+<RSP>[op] [para-1,para-2,para-3,para-4…]<CR><LF>

+: 响应消息前缀

RSP: 响应字符串，包括：`OK` 表示成功, `ERR` 表示失败

[para-n]: 查询时返回参数或出错时错误码

<CR>: 结束符，回车，ASCII 码 0x0D

<LF>: 结束符，换行，ASCII 码 0x0A

2.1.3 三、指令描述

3.1 AT+NAME

功能：查询/设置模块的名称

格式：

查询当前参数值：AT+NAME?{CR}{LF}

回应：{CR}{LF}+NAME:name{CR}{LF}OK{CR}{LF}

设置：AT+NAME=name{CR}{LF}

回应：{CR}{LF}+NAME:name{CR}{LF}OK{CR}{LF}

参数：

name：模块的名称

设置举例：设置模块名称为 AT_TEST，则需设置如下 `AT+NAME=AT_TEST{CR}{LF}`

3.2 AT+MAC

功能：查询/设置模块的 MAC 地址

格式：

查询当前参数值：AT+MAC?{CR}{LF}

回应：{CR}{LF}+MAC:mac{CR}{LF}OK{CR}{LF}

设置：AT+MAC=mac{CR}{LF}

回应：{CR}{LF}+MAC:mac{CR}{LF}OK{CR}{LF}

参数：

mac：模块的 MAC 地址，

设置举例：设置模块地址在手机上显示效果为 C00000000001, 则需设置如下

```
AT+MAC=0100000000C0{CR}{LF}
```

3.3 AT+ADVINT

功能：查询/设置广播间隔

格式：

查询当前参数值：AT+ADVINT?{CR}{LF}

回应：{CR}{LF}+ADVINT:set{CR}{LF}OK{CR}{LF}

设置：AT+ADVINT=set{CR}{LF}

回应：{CR}{LF}+ADVINT:set{CR}{LF}OK{CR}{LF}

参数：

set：模块的广播间隔

- 0：50ms
- 1：100ms
- 2：200ms
- 3：500ms
- 4：1000ms
- 5：2000ms

3.4 AT+ADV

功能：查询/设置广播工作状态

格式：

查询当前参数值：AT+ADV?{CR}{LF}

回应：{CR}{LF}+ADV:set{CR}{LF}OK{CR}{LF}

设置：AT+ADV=set{CR}{LF}

回应：{CR}{LF}+ADV:set{CR}{LF}OK{CR}{LF}

参数：

set：模块的广播状态

- B: 广播开启
- I: 广播空闲

3.5 AT+RESET

功能：控制模块重启

格式：

设置：AT+RESET?{CR}{LF}

回应：{CR}{LF}+RESET{CR}{LF}OK{CR}{LF}

3.6 AT+LINK

功能：查询模块的已连接的链路

格式：

查询当前参数值：AT+LINK?{CR}{LF}

回应：{CR}{LF}+LINK{CR}{LF}OK{CR}{LF}

Link_ID：{SPACE}ID{SPACE}LinkMode:MODE{SPACE}PeerAddr:MMAC{CR}{LF}

参数：

- ID：连接号
- LinkMode：在链接中的角色，M 表示做为 Master，S 表示做为 Slaver
- MAC：已连接设备的地址

3.7 AT+SCAN

功能：搜索周围的从机

格式：

设置扫描时间和执行一次扫描操作：AT+SCAN{CR}{LF} 或 AT+SCAN=time{CR}{LF}

回应：{CR}{LF}+SCAN:{CR}{LF}OK{CR}{LF}

No: {SPACE}num{SPACE}Addr:mac{SPACE}Rssi:sizedBm{LF}{LF}{CR}{LF}

参数：

- time：设置扫描的时间，单位：秒。
- num：搜索到从设备的索引号（最多显示周围 10 个设备）
- mac：搜索到从设备的 MAC 地址
- size：搜索到从设备的信号强度

3.8 AT+CONN

功能：通过搜索到索引号快速建立连接

格式：

设置当前参数值：AT+CONN=num{CR}{LF}

回应：{CR}{LF}+CONN:mac{CR}{LF}OK{CR}{LF}

参数：

- num：通过搜索之后的索引号
- mac：要连接的设备 MAC 值

3.9 AT+DISCONN

功能：设置断开当前连接

格式：

设置：AT+DISCONN=con_idx{CR}{LF}

回应：{CR}{LF}+DISCONN: con_idx {CR}{LF}OK{CR}{LF}

参数：

- con_idx：断开连接的连接号或字符'A', A 表示断开当前所有连接

3.10 AT+FLASH

功能：保存通过控制指令设置的参数到 FLASH 中

格式：

设置：AT+FLASH{CR}{LF}

回应：{CR}{LF}+FLASH{CR}{LF}OK{CR}{LF}

3.11 AT+SEND

功能：通过某个连接发送数据到对端

格式：

设置：AT+SEND=con_idx,len{CR}{LF}

回应：{CR}{LF}>{CR}{LF}

参数：

- con_idx: 要发送数据的链接号，从 AT+LINK? 的回复中得知
- len：本次要发送数据的长度

本条命令发送完毕，设备回复 >，表示设备进入单次透传模式，在设备发送完 *len* 指定的数据长度之前，不解析命令。发送的数据达到 *len* 指定长度时，退出单次透传模式

3.12 AT++++

功能：控制模块进入透传模式，仅在单连接时有用，此时不会解析 AT 指令

格式：

设置：AT++++{CR}{LF}

回应：{CR}{LF}+++{CR}{LF}ret{CR}{LF}

参数：

- ret: 模块进入透传的结果, `OK` 成功，`ERR` 失败

3.13 +++

- 在透传模式下，发送三个字符 +++，+++ 前面没有字符，在 500ms 之内后面也没有其他字符，即可退出透传模式进入命令模式
- 在单连接时，如果有第二个连接建立。设备会自动退出透传模式，进入命令模式

3.14 AT+AUTO+++

功能：查询/设置模块在连接上后是否自动进入透传模式

格式：

查询当前参数值：AT+AUTO+++?{CR}{LF}

回应：{CR}{LF}+AUTO+++:set{CR}{LF}OK{CR}{LF}

设置：AT+AUTO+++=set{CR}{LF}

回应：{CR}{LF}+AUTO+++:set {CR}{LF}OK{CR}{LF}

参数：

- set：Y 模块连接后自动进入透传，N 不会自动进入透传

3.15 AT+POWER

功能：查询/设置模块的射频功率

格式：

查询当前参数值：AT+POWER?{CR}{LF}

回应：{CR}{LF}+POWER:set{CR}{LF}OK{CR}{LF}

设置：AT+POWER=set{CR}{LF}

回应：{CR}{LF}+POWER:set{CR}{LF}OK{CR}{LF}

参数：

set：设置模块的发射功率

- 0：-39dBm/±1dB
- 1：-31dBm/±1dB
- 2：-18dBm/±1dB
- 3：-11dBm/±1dB
- 4：-5dBm/±1dB
- 5：-2dBm/±1dB
- 6：0dBm/±1dB
- 7：2dBm/±1dB
- 8：4dBm/±1dB
- 9：5dBm/±1dB
- 10: 6dBm/±1dB
- 11: 6.5dBm/±0.5dB
- 12: 6.8dBm/±0.5dB

- 13: 7dBm/±0.5dB
- 14: 7dBm/±0.5dB
- 15: 7.3dBm/±0.5dB
- 16: 12.3dBm/±0.5dB

3.16 AT+SLEEP

功能：控制模块进入睡眠模式

格式：

设置：AT+SLEEP=num{CR}{LF}

回应：{CR}{LF}+SLEEP{CR}{LF}ret{CR}{LF}

参数：

- num = 0: 模块进入 LP0 模式
- num = 1: 模块进入 LP2 模式
- num = 2: 模块进入 LP3 模式
- ret: 模块进入透传的结果, OK 成功，ERR 失败

注解： 进入 LP0 模式之前建议将广播间隔修改成 1S，再去测试系统功耗，通过给 PB15 IO 上升沿信号可以退出睡眠；进入 LP2 后，RAM 数据丢失，5 秒之后唤醒，唤醒之后程序会重新 REBOOT; 进入 LP3 后，RAM 数据丢失，通过给 PB15 IO 上升沿信号可以唤醒，唤醒之后程序会重新 REBOOT.

2.2 UART 设备使用示例

根据目前 SDK 所支持的三种通信模式，提供了三种对应的 UART 示例的路径如下：

阻塞模式：<install_file>/dev/examples/uart_test/uart_polling

非阻塞模式：<install_file>/dev/examples/uart_test/uart_it

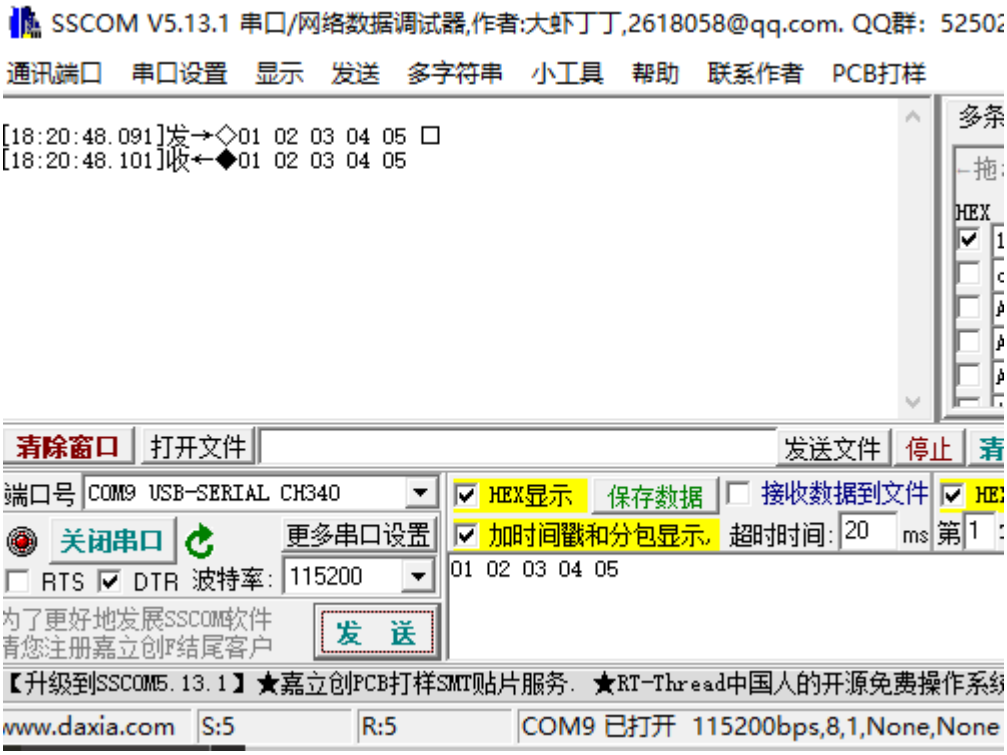
DMA 模式：<install_file>/dev/examples/uart_test/uart_dma

2.2.1 一、操作步骤:

三种示例的功能几乎都差不多，所以操作步骤基本上一样。

1. 先观察对应工程的代码，了解对应参数所代表的含义，其中阻塞模式下，设置了超时机制，超时时间默认是 10s，可以自己更改和配置，然后进行编译。
2. 将编译好的程序下载到测试的模块中。
3. 将芯片的 uart 接口（程序中设置的 IO 是 PB00PB01）接到串口转接板的 RX/TX 上。同时两者的 GND 要接到一起。
4. 打开电脑端的串口调试工具，设置波特率为：115200 数据位：8 停止位：1 奇偶校验位：null。
5. 发送数据。

2.2.2 二、预期结果:



2.3 IIC 设备使用示例

例程路径：<install_file>/dev/examples/peripheral/i2c

2.3.1 操作步骤:

1. iic 测试例程是使用我们的蓝牙芯片和 EEPROM（AT24CXX）进行通信，整个流程为：首先将数组"aTxBuffer" 的数据写入 EEPROM 中，然后再从 EEPROM 中读出来并写入数组"aRxBuffer"。
2. 将编译好的程序下载到测试的蓝牙芯片中，注意需要下载 fw.hex, info_sbl.hex, i2c_test.hex 这三个文件。
3. 将蓝牙芯片的 iic 接口（程序中设置的 IO 是 PB12(SDA)PB13(SCL)）接到 EEPROM 模块的 SDA/SCL 上。给 EEPROM 模块供电，同时两者的地线要接到一起。
4. 使用编译器仿真可以比较两个数组的值或者使用"LOG_I" 函数在"J-Link RTT Viewer" 软件中打印出来。
5. 比较结果。

2.3.2 预期结果:

1. aTxBuffer[12] = {0x00,0x10,0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,0x99,0xAA,} 注:0x00 与 0x10 为 iic 写入数据的地址
2. aRxBuffer[10] = {0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,0x99,0xAA,} 经比较数据部分完全吻合。

2.4 TIMER 使用实例

2.4.1 测试例程说明:

SDK 提供了 Timer 常用到的 4 种场景：

多通道输出 PWM：<install_file>/dev/examples/peripheral/timer/Basic_PWM

基本定时功能：<install_file>/dev/examples/peripheral/timer/Basic_TIM

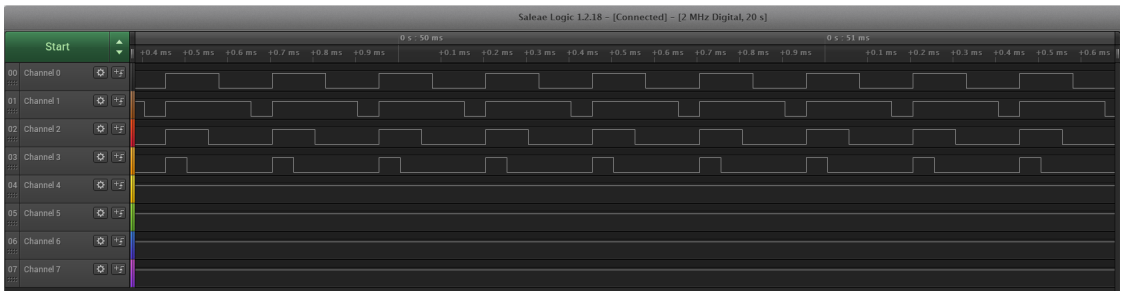
带死区嵌入的互补 PWM 输出：<install_file>/dev/examples/peripheral/timer/DTC_PWM

输入捕获功能：<install_file>/dev/examples/peripheral/timer/Input_Capture

多通道输出 PWM

该例程演示了 LSGPTIMB 的四个通道输出 PWM 波形, 频率 4KHz，占空比不一样

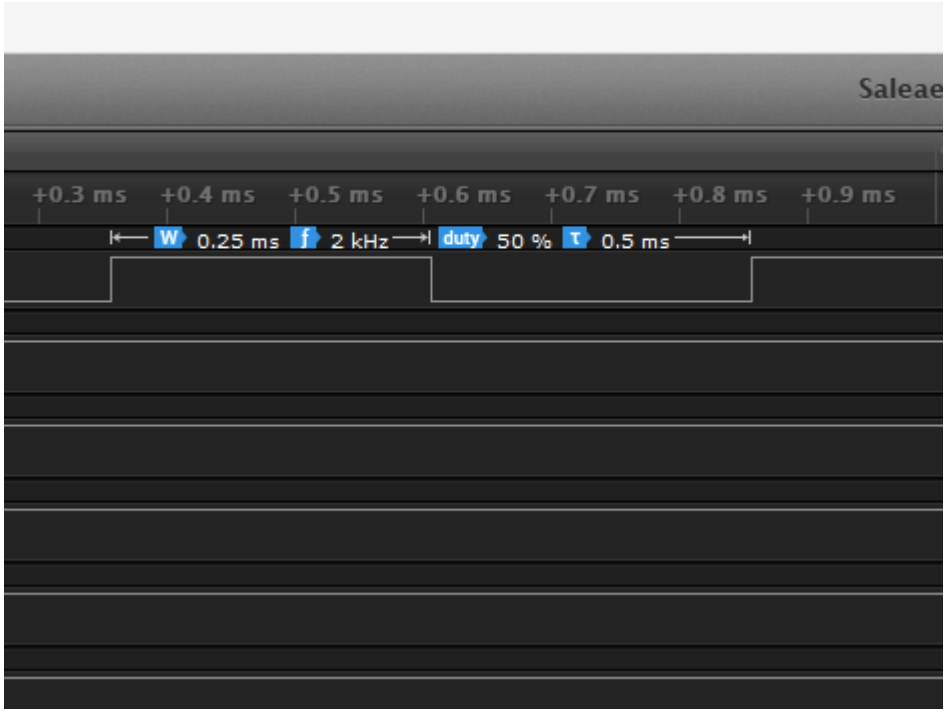
1. 将编译出来的程序烧录到芯片中，运行程序
2. 将芯片的 PA00、PA01、PB14、PB15 四个 IO 接到逻辑分析仪或者示波器
3. 四个 IO 输出的波形，如下图所示



基本定时

该例程演示了基本定时器的功能，每 250us 会上一次定时器中断，我们在中断里面翻转了 PA00 IO

1. 将编译出来的程序烧录到芯片中，运行程序
2. 将芯片的 PA00 IO 接到逻辑分析仪或者示波器
3. PA00 电平翻转波形，如下图所示



带死区嵌入的互补输出

1. 将编译出来的程序烧录到芯片中，运行程序
2. 将芯片的 PA00、PA01 两个 IO 接到逻辑分析仪或者示波器
3. 输出的波形，如下图所示

脉宽测量输入捕获

该例程中我们选用的是 LSGPTIMC 的 CH1, 使用 PA00 这个 GPIO 来测量信号的脉宽，测试模块上 PA00 接一个按键，接到 VDD，当按键按下的时候 IO 口会被拉高，这个时候我们可以利用定时器的输入捕获功能来测量按键按下的这段高电平的时间。另外按键按下和松开的时候也会 Toggle PA01 GPIO.

1. 将编译出来的程序烧录到芯片中，运行程序
2. 通过 LOG 信息可以查看捕获到的高电平时间

2.5 RTC（万年历）使用示例

例程路径：<install_file>/dev/examples/rtc_test

2.5.1 一、程序基本配置及说明:

rtc 示例程序包括三个功能配置：万年历测试，LP0 模式下 RTC 唤醒测试和 LP2 模式下 RTC 唤醒测试。根据 RC 时钟源的配置不同，又包括 LSI 和 LSE 配置下的上述测试

#define SDK_LSI_USED 1 -> 表示使用 LSI 做 RC 时钟源

#define SDK_LSI_USED 0 -> 表示使用 LSE 做 RC 时钟源，**此时板子上需要外挂 32768 晶振**

当 LSI 作为 RC 时钟源时，由于 RC 实际频率在 20KHz 左右，而 wakeuptime 计时的最细粒度也是在 1/16 秒，所以在此情况下实际误差可能会达到几十 ms 级别。而应用定时去获取万年历信息也会出现连续两秒的 second 不连续（没有变化，或者前后差 2）的情况。

当 LSE 作为 RC 时钟源时，定时是准确的，不会出现 LSI 的上述现象，但休眠功耗（LP0）会相对偏高。**（LP2 模式不支持 LSE）**RTC wakeup time 被配置为 RTC enable 的时候就开始生效，因此无论系统睡醒，RTC wakeup time 触发的中断会周期性的触发。

1.1 万年历测试

配置 #define RTC_TEST_CASE 1 该测试是针对 RTC 内部硬件万年历功能。测试流程，是将万年历初始化为一个具体的日期/时间，之后定时获取万年历运行结果。

1.2 RTC 唤醒测试（LP0 模式）

配置 #define RTC_TEST_CASE 2 在 LP0 模式下，RTC 会通过 RTC IRQ（系统处于 wakeup 时）或 LPWakeup IRQ（系统处于 sleep 时）触发中断。在系统初始化之后，程序会配置 RTC wakeup 中断，配置唤醒时间为 1 秒。

stage1：sleep_flag 初始值为 false，所以在第一次 rtc_wkup_callback 被调用（RTC 中断）之前，系统都不会休眠。

stage2：在 rtc_wkup_callback 被第一次调用后，sleep_flag 被置为 true，此时系统会立刻进入 LP0 模式

stage3：休眠模式下 RTC 中断唤醒系统，rtc_wkup_callback 被调用（LP Wakeup 中断），sleep_flag 被再次置成 false，系统不再休眠，重复 stage1

1.3 RTC 唤醒测试（LP2 模式）

配置 #define RTC_TEST_CASE 3，同时 SDK_LSI_USED 必须为 1

该测试是针对 LP2 模式下 RTC 唤醒系统功能，以及万年历功能。

与 LP0 模式的休眠唤醒不同，进入 LP2 后，每次唤醒实际上就是重启，休眠前用户 SRAM 中数据不被保存，因此在通过宏 SLEEP_CNT_MAX 配置 LP2 RTC 测试的休眠唤醒次数后，通过 LP2 模式下不会掉电的 BKD 寄存器记录已经睡醒的次数，每次唤醒都会自增。当 BKD 中计数达到 SLEEP_CNT_MAX 之后，在打印一条 log 后 CPU 会进入 while(1)，此时可以通过 RTT 的方式查看 log 输出。系统唤醒间隔为 1 秒，唤醒之后在 rtc_lp2_test 里会检查当前系统唤醒的唤醒源，如果是 RTC 唤醒则会点亮测试板的 LED（PA01）。

2.5.2 二、操作步骤及结果:

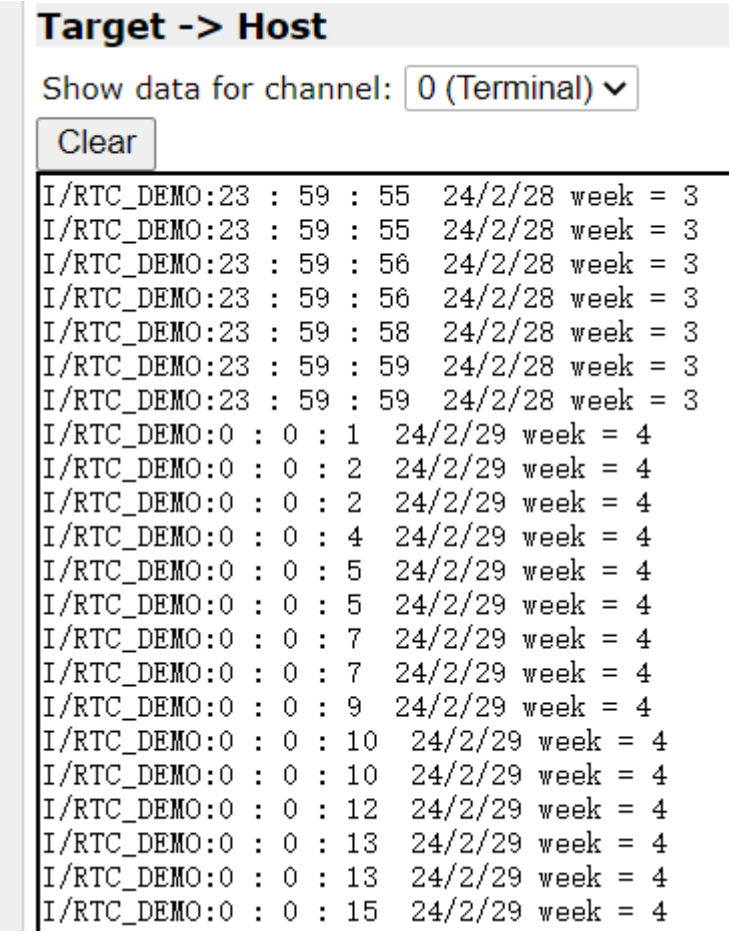
将编译好的程序下载到测试的模块中。

注意：烧录时如果系统处于运行休眠唤醒流程里，需要将 PB14 拉高后重新上电再烧录

2.1 万年历测试

#define SDK_LSI_USED 1

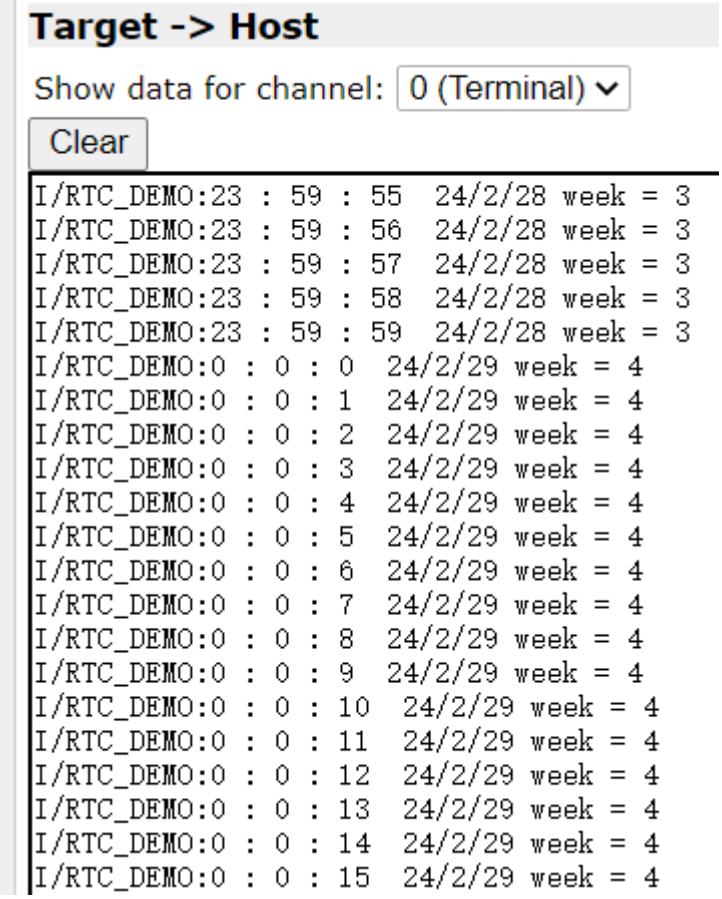
连接 RTT，可以看到程序跑起来之后每隔大概 1 秒钟左右会有一条打印输出，显示万年历信息。预期结果如下：



LSI 下的万年历输出不连续，但不会有累计误差。

#define SDK_LSI_USED 0

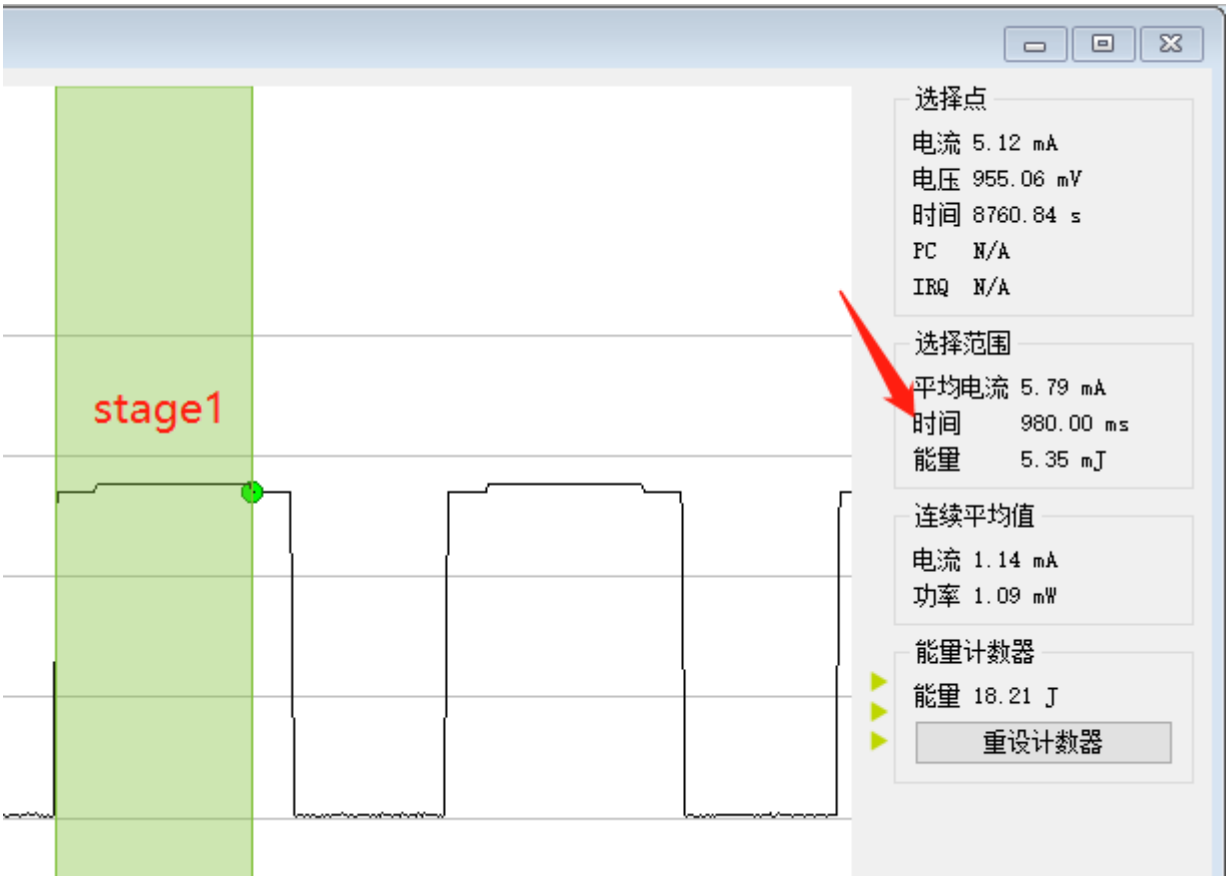
使用外部晶振作为 RC 时钟源，定时获取的万年历时间是连续的。如下图所示：



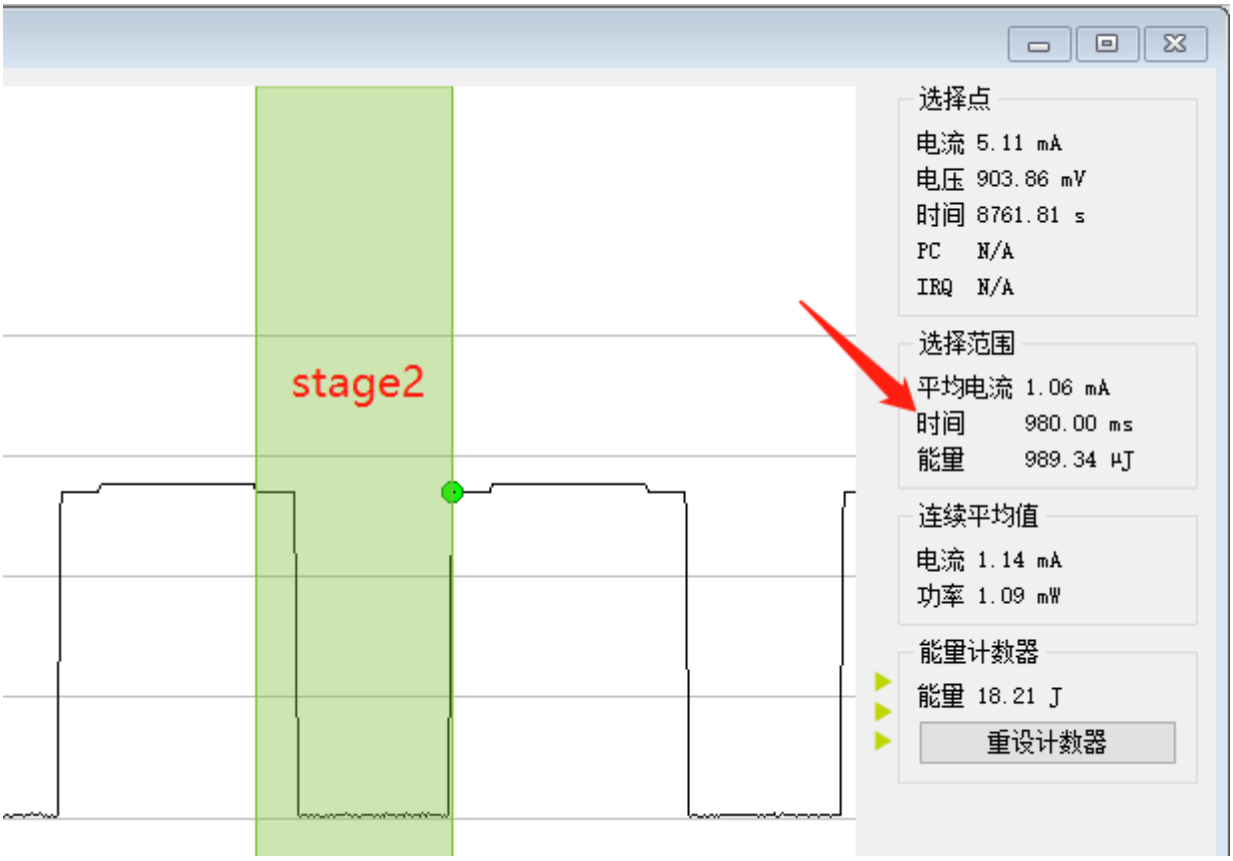
2.2 RTC 唤醒测试（LP0 模式）

#define SDK_LSI_USED 1

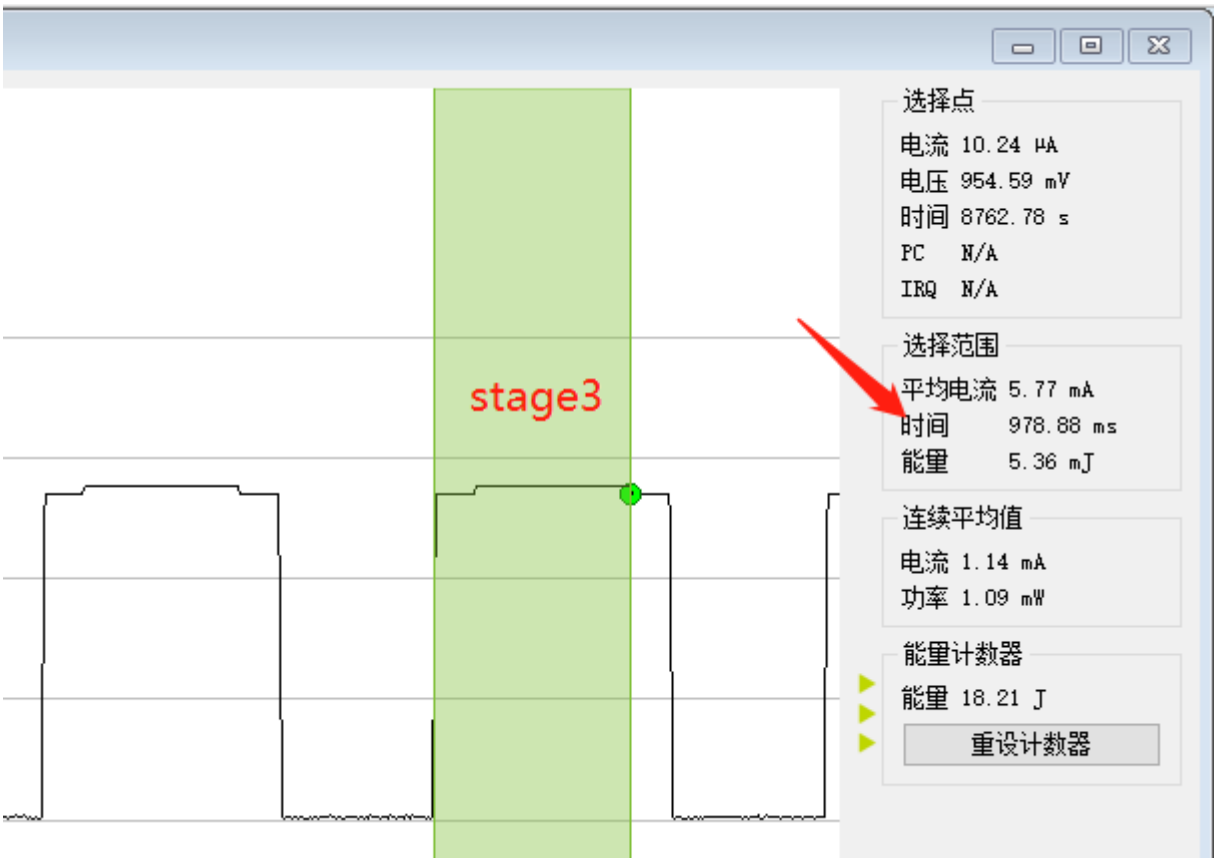
LP0 测试的 stage1 部分如下：


stage1
选择点
电流 5.12 mA
电压 955.06 mV
时间 8760.84 s
PC N/A
IRQ N/A
选择范围
平均电流 5.79 mA
时间 980.00 ms
能量 5.35 mJ
连续平均值
电流 1.14 mA
功率 1.09 mW
能量计数器
能量 18.21 J
重设计数器

stage2 部分如下：

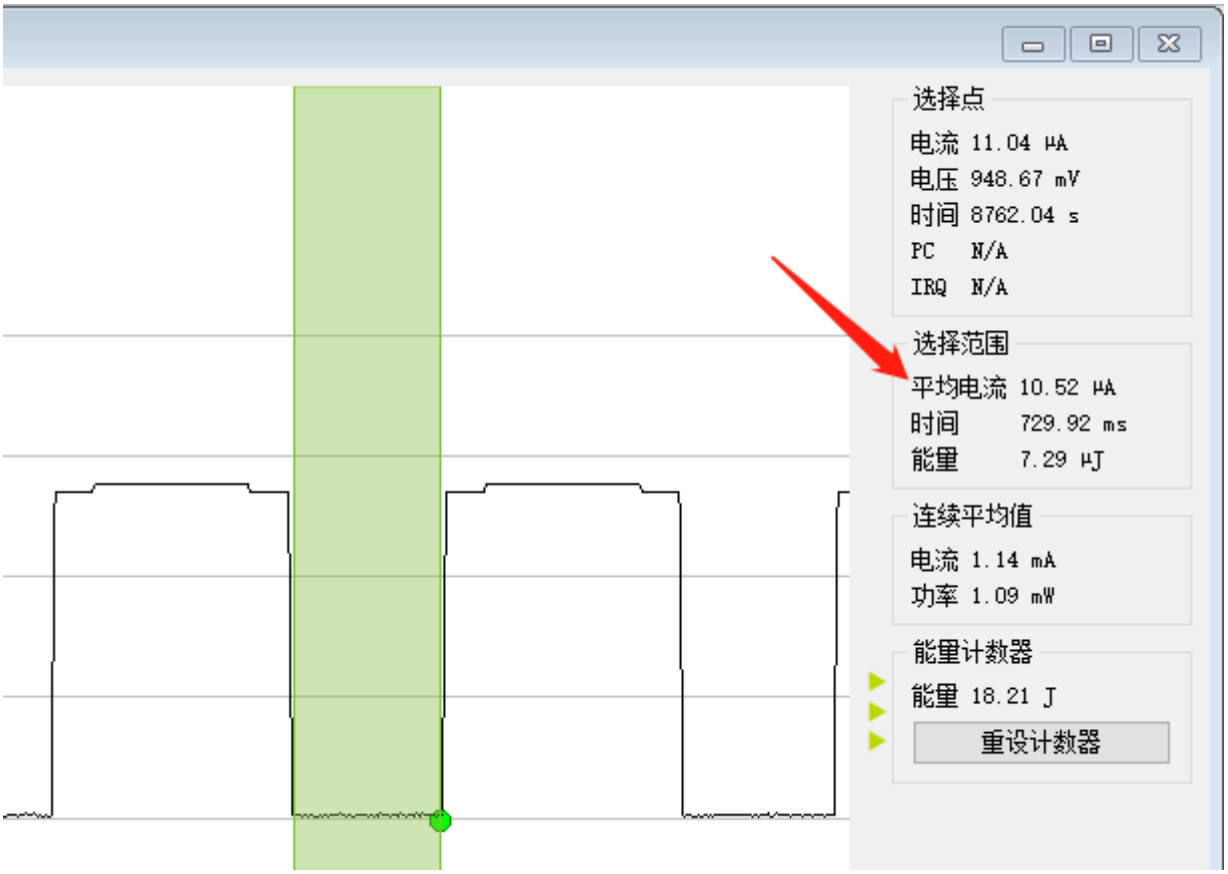


stage3 部分如下（stage3 实际是重复 stage1）：



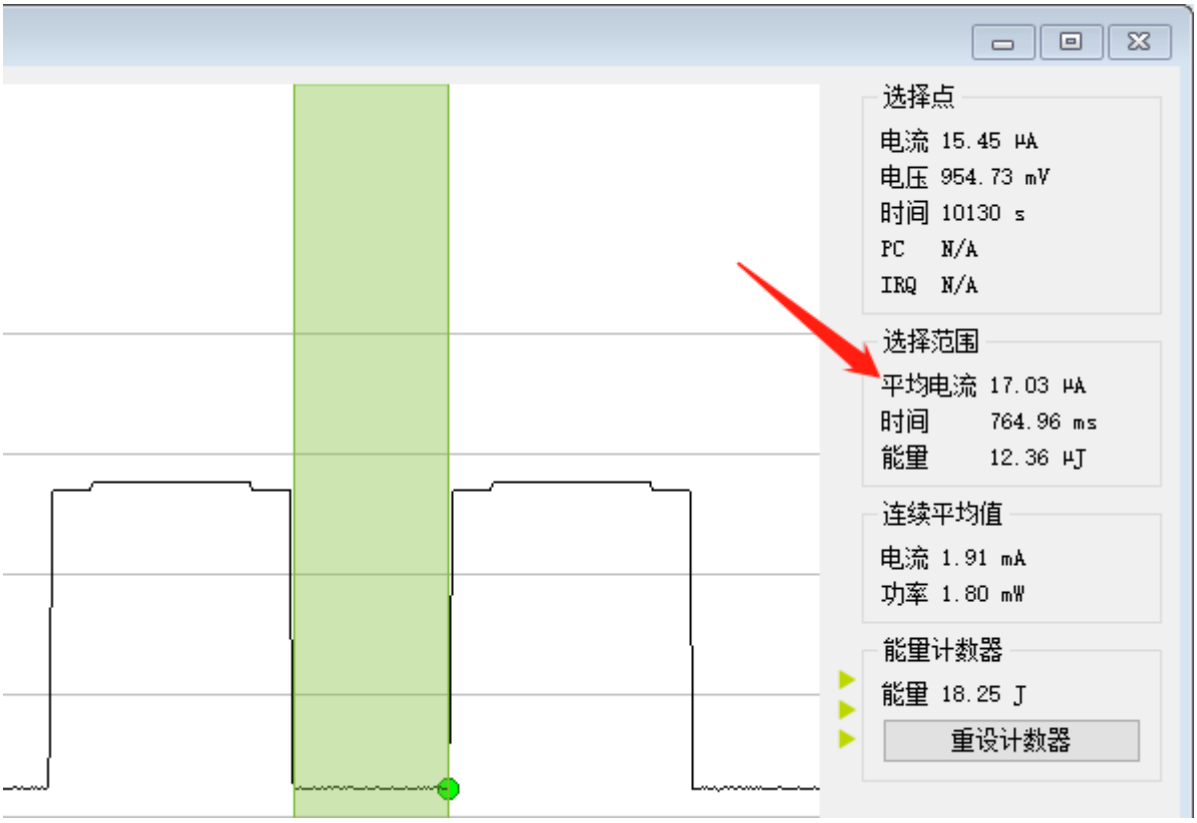
由于是 LSI 作为 RC 时钟源，也会出现唤醒时间有误差的情况。

LSI 作为时钟源，LP0 休眠时的功耗如下：



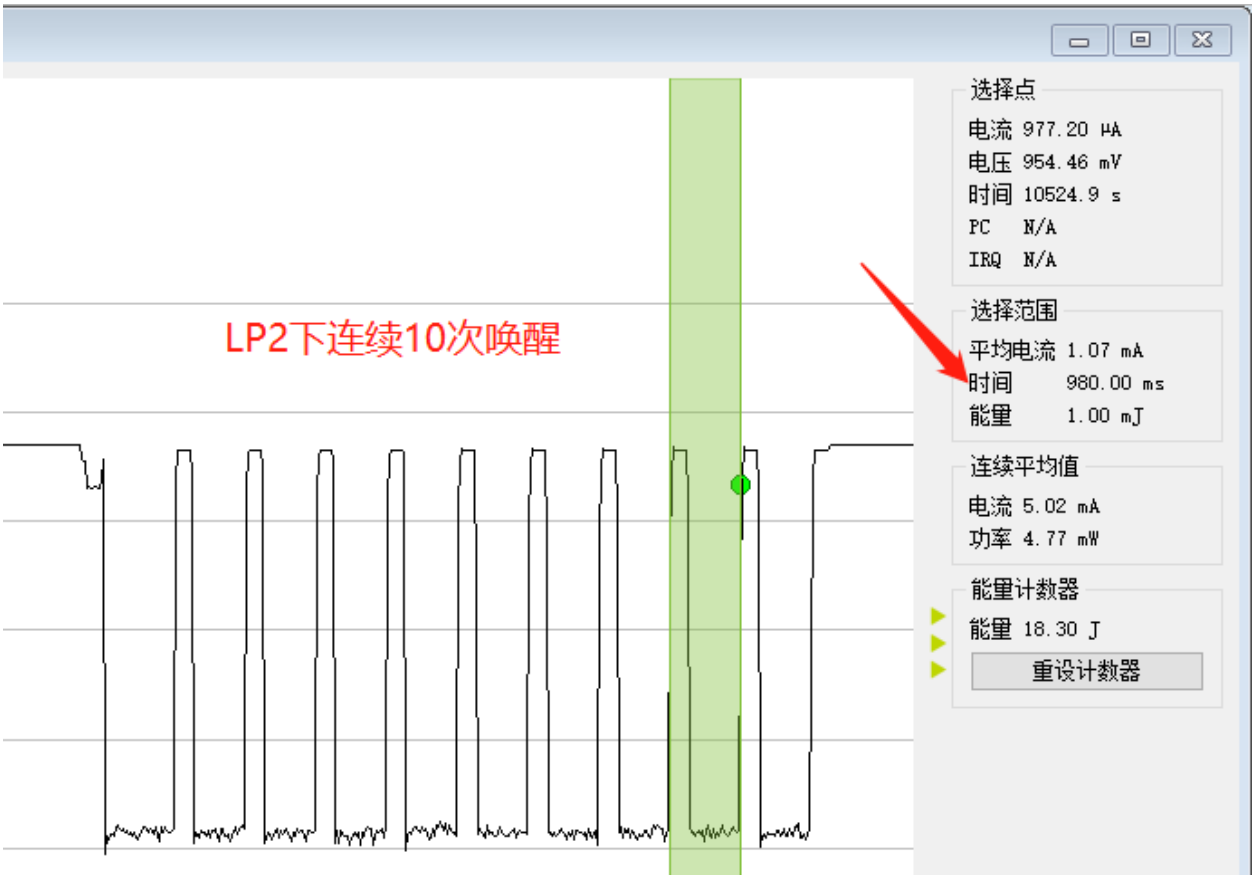
#define SDK_LSI_USED 0

使用外部晶振，LP0 测试的 stage1/stage2/stage3 基本和 LSI 一致，唯一的区别是 LSE 定时精准。另外就是休眠功耗会有差异：

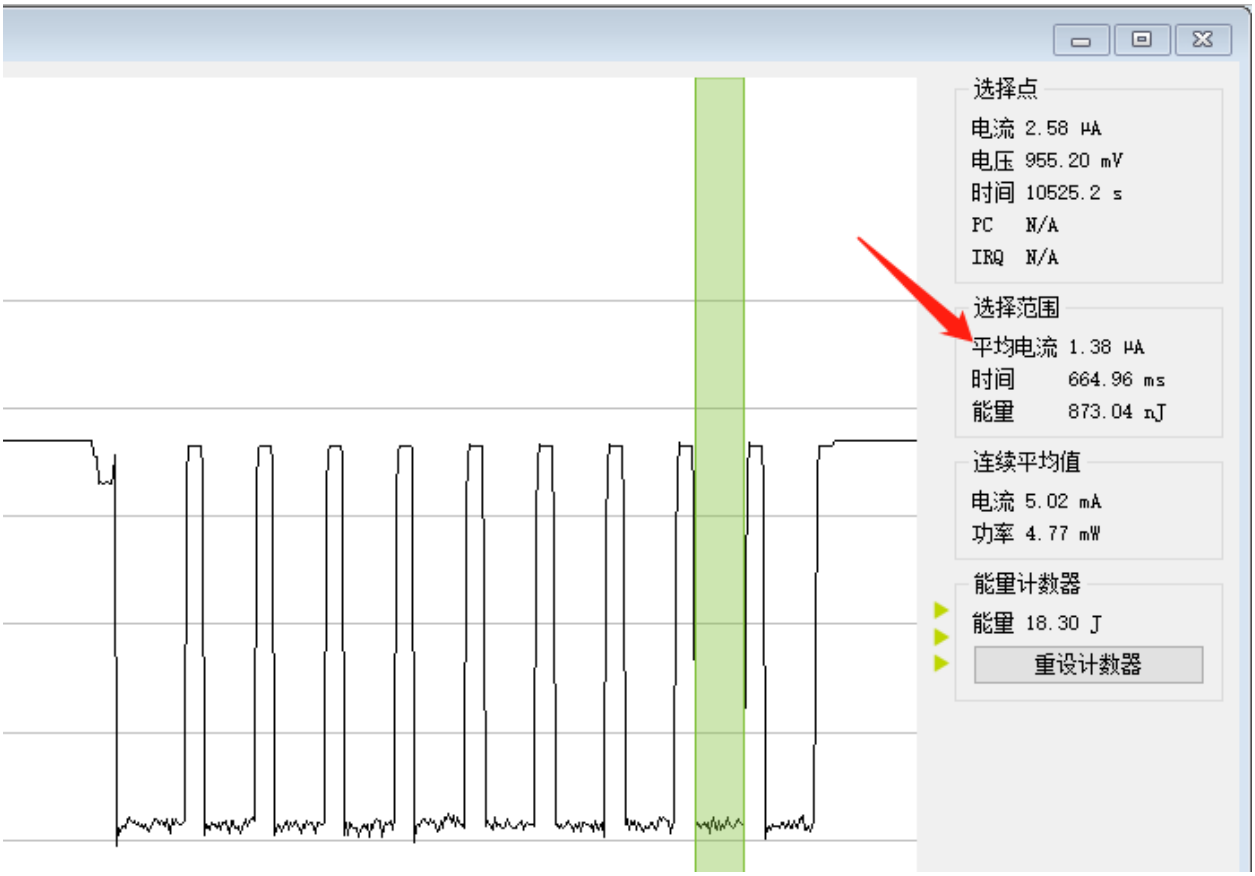


2.2 RTC 唤醒测试（LP2 模式）

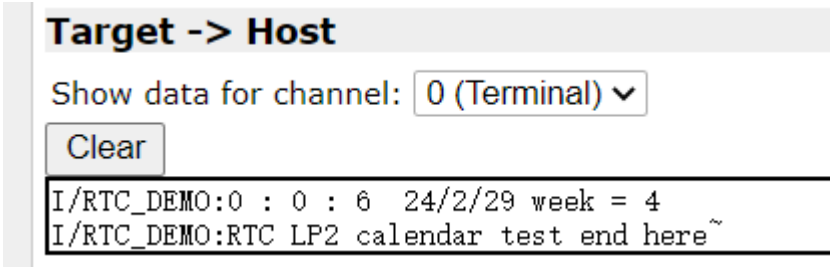
通过监测系统功耗，可以看到系统处于定期唤醒的 LP2 模式（sleep 功耗 **1uA** 左右），总计 SLEEP_CNT_MAX 次，与此同时测试板的 LED 灯会定期闪烁。预期结果如下：


LP2下连续10次唤醒
选择点
电流 977.20 μA
电压 954.46 mV
时间 10524.9 s
PC N/A
IRQ N/A
选择范围
平均电流 1.07 mA
时间 980.00 ms
能量 1.00 mJ
连续平均值
电流 5.02 mA
功率 4.77 mW
能量计数器
能量 18.30 J
重设计数器

休眠功耗如下：



系统休眠唤醒 SLEEP_CNT_MAX 次后，可以连接 RTT 查看 log 输出。预期结果如下：



注意：由于 LP2 模式下系统唤醒实际为系统重启，因此虽然逻辑上来讲测试中会打印 SLEEP_CNT_MAX 次时间输出，但最终看到的只有最后一次

2.6 GPIO 设备使用示例

例程路径：<install_file>/dev/examples/gpio_test

2.6.1 一、使用说明

在测试例程中，使用的 PB09 和 PB10 作为输出的 GPIO 示例。

PB11 作为中断调试的按键，且设置为上拉输入，下降沿触发。

2.6.2 二、操作步骤

2.1: 将编译好的工程文件下载到测试的模块中。

2.2：开始运行调试程序。

2.6.3 三、预期结果

3.1：测试得到 PB09 和 PB10 的电平状态。

3.2：如果给 PB11 一个低电平信号，程序会进入中断，同时改变 PB08 和 PB09 的电平状态。

注: PB09 的电平状态随着 PB11 输入电平状态的改变而改变，而 PB11 每产生一个下降沿中断，PB10 的电平状态就会翻转一次。

2.7 PDM 使用示例

1. PDM（Pulse Density Modulation）是一种用数字信号表示模拟信号的调制方法。PDM 只有 1 位数据输出，要么为 0，要么为 1。

2. PDM 把数字麦克风采样的数据会放置在 PCM 数据寄存器的 0-15 位中，可以通过 DMA 或者 PDM 中断的方式去收集 pdm 采样到的实时数据。

3. 此测试用例是通过 DMA 的乒乓模式来搬运 PDM 数据流并通过 DMA 基本模式把 PDM 数据打印到串口助手上。

2.7.1 一、串口配置

1. 包含 io_config 和 lsuart 头文件。

2. 定义 UART 句柄，并调用 HAL_UART_Init 初始化函数。通过 DMA 搬运数据到串口助手上时，需要在 uart 初始化的时候同时配置 uart 所使用的 dma 对象和 dma 通道。

3. 默认使用 UART1 并把 PB00、PB01 作为通讯串口，其中 PB00 为模块的 TX，PB01 为模块的 RX，默认的 uart 口的配置参数为：波特率 500000、无校验、8 位数据位、1 位停止位。

```
#include "io_config.h"
#include "lsuart.h"
static UART_HandleTypeDef UART_PDM_Config;
    static void ls_pdm_uart_init(void)
    {
            uart1_io_init(PB00, PB01); //uart io 初始化函数
            UART_PDM_Config.UARTX = UART1;
            UART_PDM_Config.DMAC_Instance = &dmac1_inst; //uart 选择的 dma 对象
            UART_PDM_Config.Tx_Env.DMA.DMA_Channel = 3; //uart 选择的 dma 通道号
            UART_PDM_Config.Init.BaudRate = UART_BAUDRATE_500000;
            UART_PDM_Config.Init.MSBEN = 0;
            UART_PDM_Config.Init.Parity = UART_NOPARITY;
            UART_PDM_Config.Init.StopBits = UART_STOPBITS1;
            UART_PDM_Config.Init.WordLength = UART_BYTESIZE8;
            HAL_UART_Init(&UART_PDM_Config);
    }
```

2.7.2 二、PDM 配置

1. 包含 lspdm 头文件。注意：io_config 文件在 UART 配置的时候已经包含。

2. 定义 PDM 句柄，并调用 HAL_PDM_Init 初始化函数。

3. 通过 GPIO 复用模式把 PB10 复用为数字麦克风的 CLOCK 引脚，PB09 复用为数字麦克风的 DATA 引脚。

4. 把 PDM 句柄中的实例对象映射到 PDM 寄存器的基址中，并对 PDM 寄存器进行赋值。

5. 把 PDM 时钟速率设为 1.024MHZ，采样率设为 16KHZ，延时捕捉为 30 个周期，数据增益设为 5，采用单声道模式。

6. 选择 PDM 的 DMA 对象、PDM 的 DMA 通道选择、配置 DMA 乒乓模式下交替存放 FRAME_BUF_SIZE 大小 PDM 数据的两个数组。

```
#include "lspdm.h"
    #define PDM_CLK_KHZ 1024    //默认使用 1.024MHZ 的时钟速率
    #define PDM_SAMPLE_RATE_HZ 16000    //默认使用 16KHZ 采样
    #define FRAME_BUF_SIZE 256
```

(下页继续)

(续上页)

```
    PDM_HandleTypeDef pdm;//pdm 句柄定义
    DMA_RAM_ATTR uint16_t Buf0[FRAME_BUF_SIZE]; //dma 乒乓模式下交替接受 pdm 数据 buf0 的定
义
    DMA_RAM_ATTR uint16_t Buf1[FRAME_BUF_SIZE]; //dma 乒乓模式下交替接受 pdm 数据 buf1 的定
义
    static struct PDM_PingPong_Bufptr pdm_data_receive;
    void pdm_init()
    {
            pdm_clk_io_init(PB10);
            pdm_data0_io_init(PB09);
            pdm.Instance = LSPDM; //LSPDM 表示 pdm 外设的基址，在 lspdm.h 中已经定义
            PDM_Init_TypeDef Init =
            {
                    .fir = PDM_FIR_COEF_16KHZ,
                    .cfg = {
                            .clk_ratio = PDM_CLK_RATIO(PDM_CLK_KHZ), //pdm 时钟配置
                            .sample_rate = PDM_SAMPLE_RATE(PDM_CLK_KHZ,PDM_SAMPLE_
↪RATE_HZ), //pdm 采样速率配置
                            .cap_delay = 30, //延时捕捉
                            .data_gain = 5, //数据增益
                    },
                    .mode = PDM_MODE_Mono, //单声道模式
            };
            pdm.DMAC_Instance = &dmac1_inst; //pdm 选择的 dma 对象
            pdm.Env.DMA.Channel[0] = 1; //dma 通道选择
            pdm_data_receive.Bufptr[0] = Buf0;
            pdm_data_receive.Bufptr[1] = Buf1;
            HAL_PDM_Init(&pdm,&Init);
    }
```

2.7.3 三、其他配置

DMA 对象初始化。

1. 包含 lsdmac.h 头文件。
2. 调用 DEF_DMA_CONTROLLER 函数。
3. 调用 DMA_CONTROLLER_INIT 函数。

```
#include "lsdmac.h"
DEF_DMA_CONTROLLER(dmac1_inst,DMAC1);
DMA_CONTROLLER_INIT(dmac1_inst); //dma 对象初始化
```

2.7.4 四、PDM 配置步骤及流程介绍

1. DMA 对象初始化，调用 DEF_DMA_CONTROLLER 和 DMA_CONTROLLER_INIT 函数

2. UART 初始化及用到的 DMA 通道配置，调用 ls_pdm_uart_init 函数

3. PDM 初始化及用到的 DMA 通道配置，调用 pdm_init 和 pdm_dma_test 函数

4. DMA 乒乓模式的配置，调用 HAL_PDM_PingPong_Transfer_Config_DMA 函数

5. 开始使能 PDM，调用 HAL_PDM_Start 函数

6. 等待 DMA 在乒乓模式下交替接收 pdm 数据的两个数组是否接收满 FRAME_BUF_SIZE 大小的数据，接受完之后会在 DMA 中断处理函数中调用重定义的 HAL_PDM_DMA_CpltCallback 函数

7. 在 HAL_PDM_DMA_CpltCallback 函数中的标志量，用于判断发送哪个用户自定义的数组数据（buf0 或者 buf1）到串口助手上，调用 HAL_UART_Transmit_DMA 函数

8. 当把用户自定义的数组数据通过串口发送完毕之后，会调用用户重定义的发送完成回调函数：HAL_UART_DMA_TxCpltCallback

2.7.5 五、PDM 通过 DMA 和 UART 收发数据的操作如下：

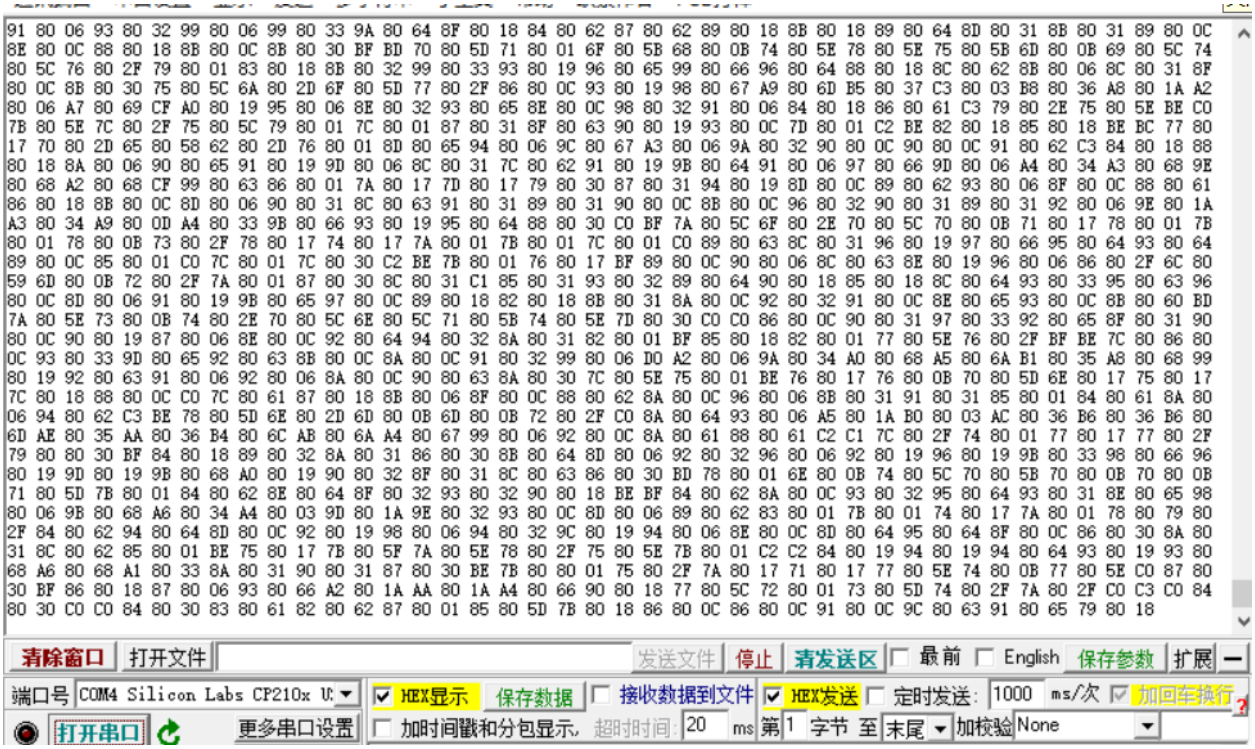
1. 将编译好的程序下载到测试的模块中。

2. 将芯片的 uart 接口（程序中设置的 IO 是 PB00(TX) PB01(RX)）接到串口转接板的 RX/TX 上。同时两者的地线要接到一起。

3. 将 DMIC 的 clock 引脚接 PB10、outdata 引脚接 PB09、LR 引脚接 vdd 或地线、GND 引脚接地线、VDD 引脚接电源 vdd 引脚。

4. 打开电脑端的串口调试工具，设置波特率为：500000 数据位：8 停止位：1 奇偶校验位：null。

5. 打开串口就可以看到串口助手上显示的语音数据，预期结果如下图所示。



2.8 SPI 设备使用示例

根据目前 SDK 所支持的三种通信模式，提供了三种对应的 SPI 示例的路径如下：

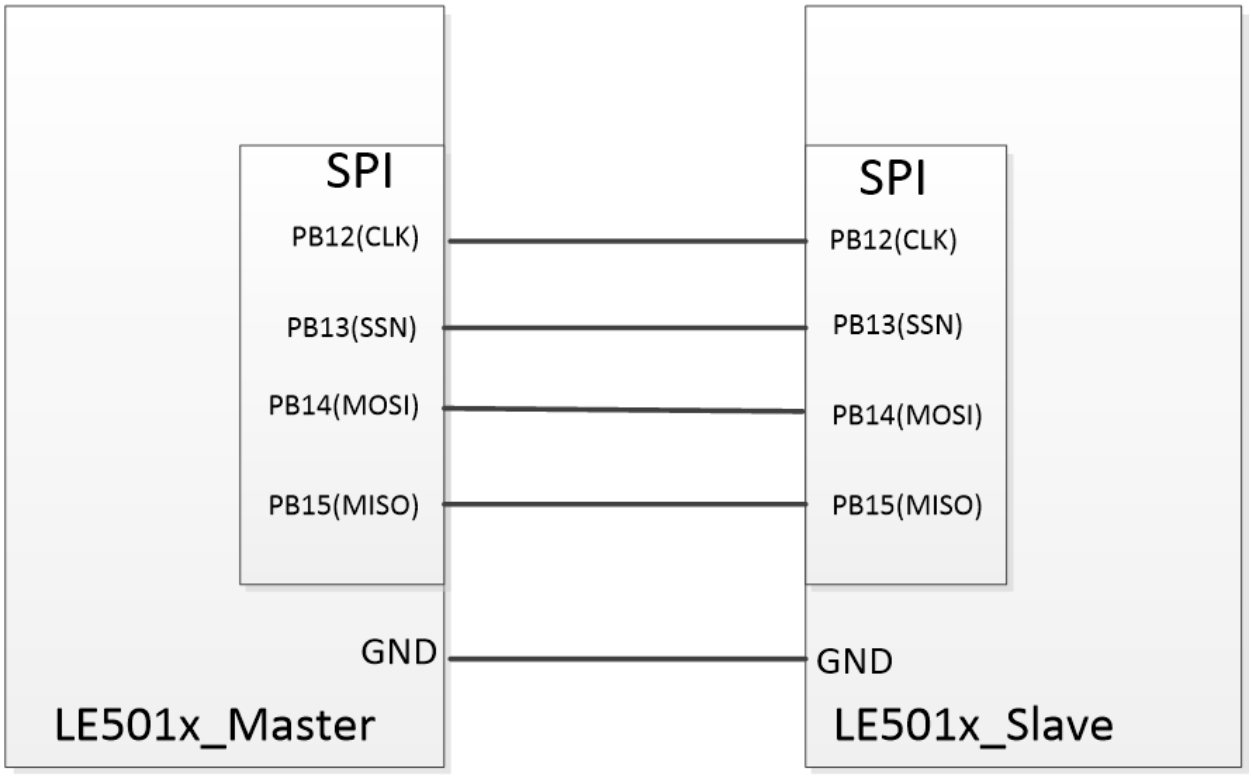
阻塞模式：<install_file>/dev/examples/spi_test/spi_polling

非阻塞模式：<install_file>/dev/examples/spi_test/spi_it

DMA 模式：<install_file>/dev/examples/spi_test/spi_dma

2.8.1 一、硬件外围连接:

SPI 使用示例需要主/从两个模块共同完成，示例演示主机模块和从机模块之间相互通信过程，推荐使用我司官方 LE501x dongle 板加载示例验证。主从硬件连接方式如下:



2.8.2 一、示例介绍:

本示例主要用于演示凌思 SOC 芯片外设 SPI 的主/从应用，示例代码主要由以下部分组成：初始化、SPI 发送/接收、接收数据判断及结果提示。

1)、初始化：IO 初始化、SPI 初始化

IO 初始化：数字复用 IO 支持管脚全映射，在这里我们选择 PB12~PB15 分别复用做为 CLK、SSN、MOSI、MISO。

```
/* Configure the GPIO AF */
spi_clk_io_init(PB12);      /* CLK-------------PB12 */
spi_nss_io_init(PB13);      /* SSN-------------PB13 */
spi_mosi_io_init(PB14);     /* MOSI------------PB14 */
spi_miso_io_init(PB15);     /* MISO------------PB15 */
```

SPI 初始化：用户在使用 SPI 外设前，必须对 SPI 模块参数进行必要的配置，以满足不同 SPI 外围设备的通信时序要求。

```
SpiHandle.Instance               = SPI2;
SpiHandle.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_64;    //通信速率
SpiHandle.Init.Direction         = SPI_DIRECTION_2LINES;        //通信方向
SpiHandle.Init.CLKPhase          = SPI_PHASE_1EDGE;             //时钟相位
SpiHandle.Init.CLKPolarity       = SPI_POLARITY_LOW;            //时钟极性
SpiHandle.Init.DataSize          = SPI_DATASIZE_8BIT;           //数据位宽
SpiHandle.Init.FirstBit          = SPI_FIRSTBIT_MSB;            //数据格式
SpiHandle.Init.TIMode            = SPI_TIMODE_DISABLE;          //TI 模式使能位
SpiHandle.Init.NSS               = SPI_NSS_HARD_OUTPUT;         //CS 脚控制方式
SpiHandle.Init.Mode              = SPI_MODE_MASTER;             //SPI 模式
```

2)、SPI 发送/接收通过调用发送/接收接口 API 函数，并把发送或接收数据 buf 和长度传入接口函数（polling 模式还需要传入超时时间），开始在通信线上发送或接收数据。

> polling 模式：当发送或接收数据个数满足预定长度，即退出 API 函数，可通过函数返回值判断本次通信状态。
>
> IT 模式：使用中断模式发送或接收数据，接口函数调用后会马上退出，可通过函数返回值判断本次通信状态，并在发送或接收数据长度满足预定大小时，驱动会通过回调函数通知应用。
>
> dma 模式：使用 dma 模式发送或接收数据，操作和流程与 IT 模式相同，仅回调函数接口不一样。需要注意：dma 模式使用的 buf 内存必须指定在 dma 特定 RAM 空间

3)、结果判断和提示 SPI 数据发送接收完成后，示例会对当前通信数据进行比对，如果从机接收到的数据和主机发送的数据一致，即在 PB9 IO 口输出 500ms 翻转信号（如使用 dongle 板，可看到蓝色 LED 持续闪烁）。

2.8.3 一、操作步骤:

SPI 三个示例演示功能基本相同，可使用相同硬件和操作顺序，均演示主机发送数据，从机接收数据。这里以 spi_polling 为例。

1. 打开工程，熟悉功能结构和 main.c 文件中各个函数功能。

2. 在 main.c 中找到主从配置宏定义，如下。当主机程序时，使能该宏定义；当从机程序时，不使用该宏定义（双斜杠注释掉）。

```
/* Uncomment this line to use the board as master, if not it is used as slave */
#define MASTER_BOARD
```

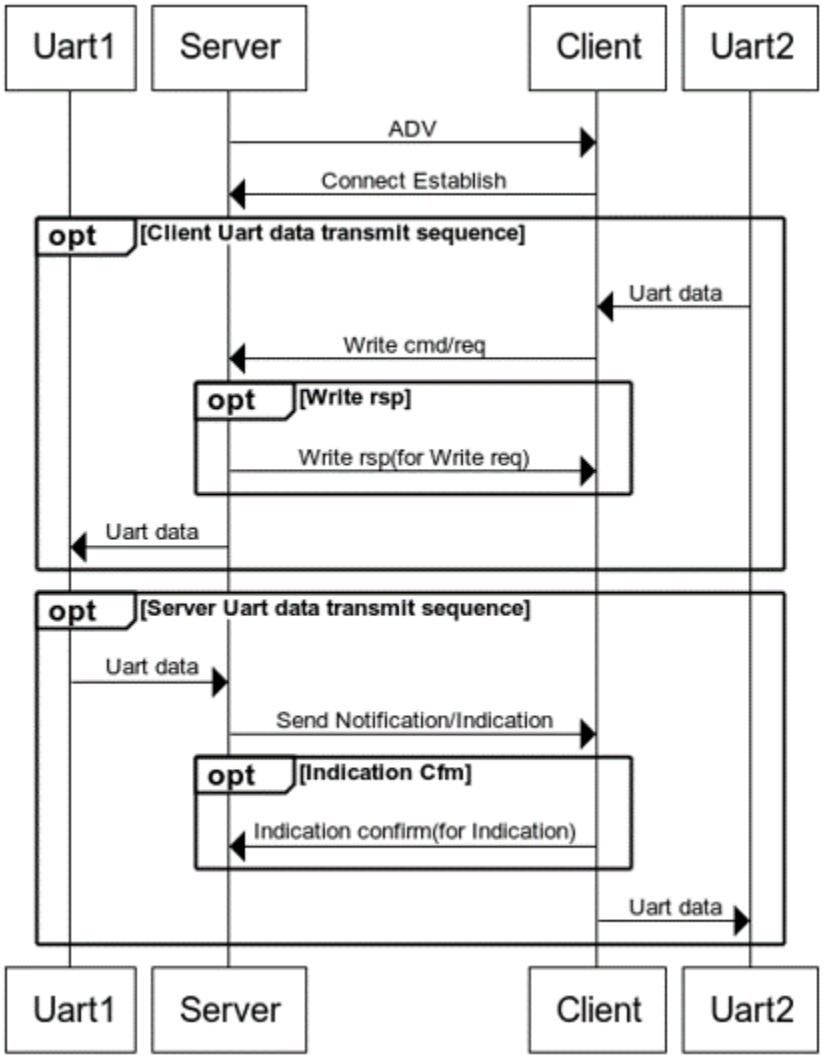
3. 主从程序分别加载到对应硬件板子后，从机上电 3s 后，主机再上电。如果两端通信成功，可在 dongle 板上看到蓝色 LED 灯 500ms 间隔持续闪烁。

2.9 BLE_UART_SERVER（串口透传）示例说明

例程路径：<install_file>/dev/examples/ble/ble_uart_server

2.9.1 一、示例基本配置、流程及说明:

BLE_UART_SERVER（以下简称 uart_server）是具备蓝牙串口透传功能且无安全要求的单连接示例。串口透传，指的是作为无线数据传输通道，蓝牙芯片将 Uart 上收到的数据不经任何处理直接发送给蓝牙对端，同时也将蓝牙收到的数据推送到 Uart 上。



1.1 准备工作之一：串口相关初始化

串口透传，首先需要初始化串口相关的功能模块，主要包括是能串口硬件模块，以及创建一个用于数据查询和处理的软件定时器。调用的相关接口如下：

```
ls_uart_init(); // 串口模块初始化
HAL_UART_Receive_IT(&UART_Server_Config, &uart_server_rx_byte, 1);  // 串口接收使能，每次接收 1byte，存放到 uart_server_rx_byte 内
ls_uart_server_init(); // 初始化软件定时器
```

串口参数默认配置 IO 为 PB00/PB01，波特率为 115200，具体可以参考 ls_uart_init() 的实现。软件定时器周期配置为 50ms。

1.2 准备工作之二：服务添加及注册

串口透传需要添加 uart_server 服务，这里主要包括服务定义和服务内部的特征值/描述符定义。

```
static const struct att_decl ls_uart_server_att_decl[UART_SVC_ATT_NUM] =
{
    [UART_SVC_IDX_RX_CHAR] = {
        .uuid = att_decl_char_array,
        .s.max_len = 0,
        .s.uuid_len = UUID_LEN_16BIT,
        .s.read_indication = 1,
        .char_prop.rd_en = 1,
    },
    [UART_SVC_IDX_RX_VAL] = {
        .uuid = ls_uart_rx_char_uuid_128,
        .s.max_len = UART_SVC_RX_MAX_LEN,
        .s.uuid_len = UUID_LEN_128BIT,
        .s.read_indication = 1,
        .char_prop.wr_cmd = 1,
        .char_prop.wr_req = 1,
    },
    [UART_SVC_IDX_TX_CHAR] = {
        .uuid = att_decl_char_array,
        .s.max_len = 0,
        .s.uuid_len = UUID_LEN_16BIT,
        .s.read_indication = 1,
        .char_prop.rd_en = 1,
    },
    [UART_SVC_IDX_TX_VAL] = {
        .uuid = ls_uart_tx_char_uuid_128,
        .s.max_len = UART_SVC_TX_MAX_LEN,
        .s.uuid_len = UUID_LEN_128BIT,
        .s.read_indication = 1,
        .char_prop.ntf_en = 1,
    },
    [UART_SVC_IDX_TX_NTF_CFG] = {
        .uuid = att_desc_client_char_cfg_array,
        .s.max_len = 0,
        .s.uuid_len = UUID_LEN_16BIT,
        .s.read_indication = 1,
        .char_prop.rd_en = 1,
        .char_prop.wr_req = 1,
    },
};
static const struct svc_decl ls_uart_server_svc =
```

(下页继续)

(续上页)

```
{
    .uuid = ls_uart_svc_uuid_128,
    .att = (struct att_decl*)ls_uart_server_att_decl,
    .nb_att = UART_SVC_ATT_NUM,
    .uuid_len = UUID_LEN_128BIT,
};
```

这里的定义大多符合 SIG 关于服务/特征值/描述符的规范的，具体可参照代码实现及蓝牙官方 core 协议。这里重点提一下一些自定义配置：1、max_len 为该特征值操作的最大长度，单位为 byte，对于改特征值的操作不可以超出 max_len，否则多余的部分会被丢弃 2、read_indication 表示收到对方的读请求时，是否将该请求发送至应用层，通常配置都为 1

添加服务需要首先调用

```
dev_manager_add_service((struct svc_decl *)&ls_uart_server_svc);
```

之后会上 SERVICE_ADDED 消息，再调用

```
gatt_manager_svc_register(evt->service_added.start_hdl, UART_SVC_ATT_NUM, &ls_uart_
↪server_svc_env);
```

1.3 发广播包/建立连接

在 uart_server 服务注册完成后，所有准备工作已经完成，此时可以调用 create_adv_obj() 创建广播对象。函数具体实现为：

```
static void create_adv_obj()
{
    struct legacy_adv_obj_param adv_param = {
        .adv_intv_min = 0x20,
        .adv_intv_max = 0x20,
        .own_addr_type = PUBLIC_OR_RANDOM_STATIC_ADDR,
        .filter_policy = 0,
        .ch_map = 0x7,
        .disc_mode = ADV_MODE_GEN_DISC,
        .prop = {
            .connectable = 1,
            .scannable = 1,
            .directed = 0,
            .high_duty_cycle = 0,
        },
    };
```

(下页继续)

(续上页)

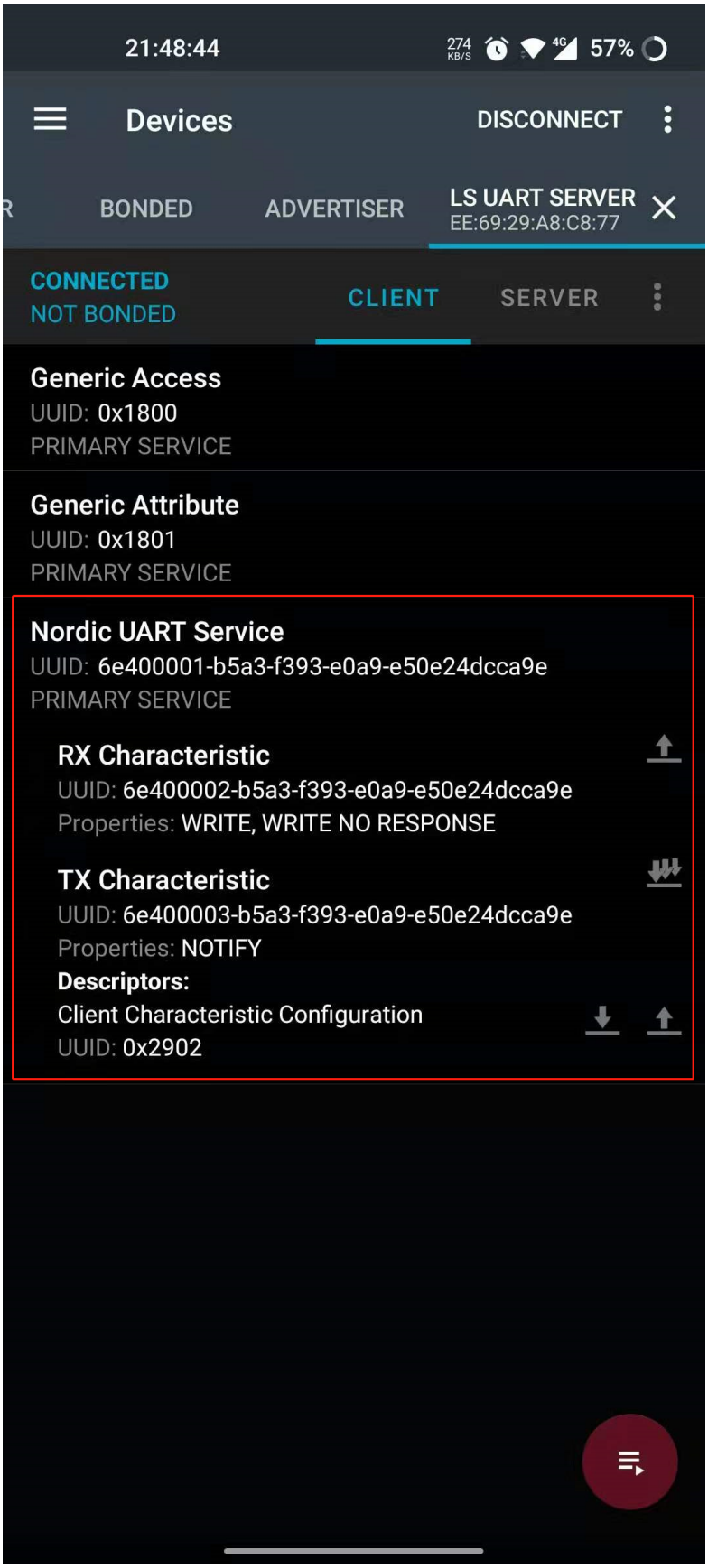
```
    dev_manager_create_legacy_adv_object(&adv_param);
}
```

1、adv_intv_min/adv_intv_max 分别表示广播包的最小和最大周期，单位为 625us，一般配置成同一个值

2、ch_map 表示每组广播包的个数，默认为 7，表示在 37/38/39 这 3 个 channel 上都会发送

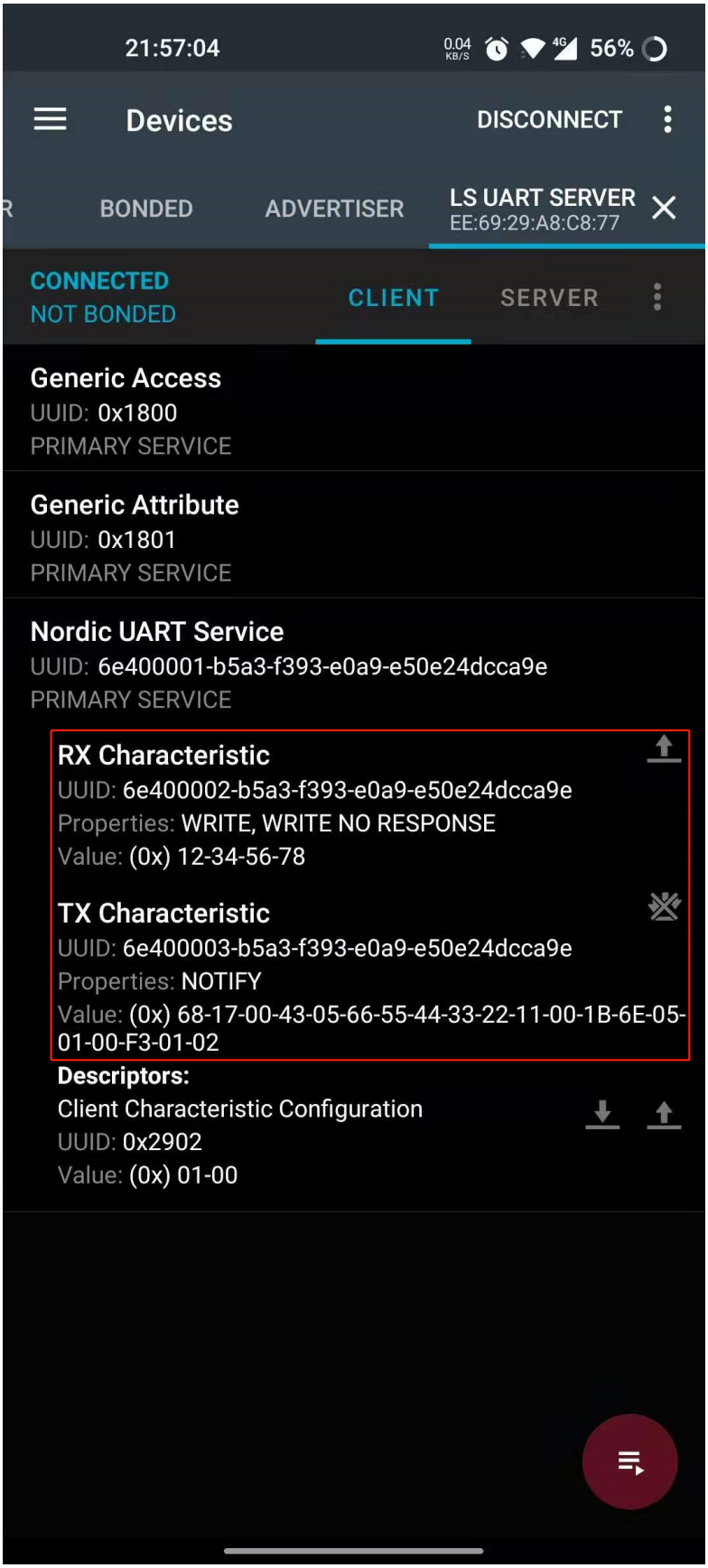
3、uart_server 示例里，connectable 必须为 1，否则为不可连接广播包，后续无法建立连接

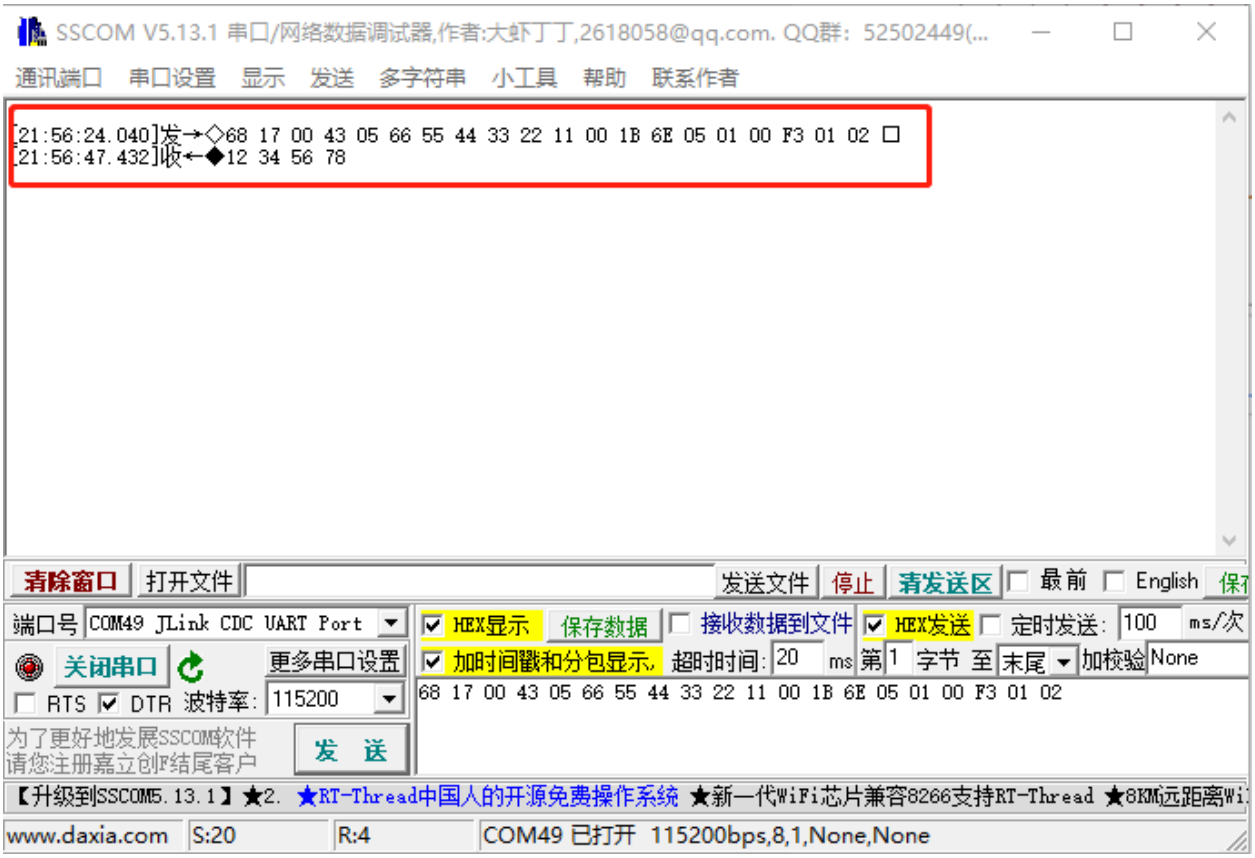
在广播对象创建完成后，调用 start_adv() 开启广播。在这一步需要特别注意，组建 advertising_data 和 scan_response_data 内容需要通过使用宏 ADV_DATA_PACK，返回值为最终的长度，作为参数之一传入 dev_manager_start_adv()。如果没有 advertising_data 或 scan_response_data，对应的 length 需要填 0。**不可以填如与实际内容不匹配的 length，例如 sizeof(advertising_data)，否则协议栈的解析有可能出错！**广播包发出来之后，可以通过手机 APP（例如 nRF Connect）扫描该设备的广播包，并建立连接，成功后本地能在 gap_manager_callback() 里收到 CONNECTED 消息，手机端也会自动进行服务发现流程，通常如下：


21:48:44
274 KB/S
57%
Devices
DISCONNECT
BONDED
ADVERTISER
LS UART SERVER
EE:69:29:A8:C8:77
CONNECTED
NOT BONDED
CLIENT
SERVER
Generic Access
UUID: 0x1800
PRIMARY SERVICE
Generic Attribute
UUID: 0x1801
PRIMARY SERVICE
Nordic UART Service
UUID: 6e400001-b5a3-f393-e0a9-e50e24dcca9e
PRIMARY SERVICE
RX Characteristic
UUID: 6e400002-b5a3-f393-e0a9-e50e24dcca9e
Properties: WRITE, WRITE NO RESPONSE
TX Characteristic
UUID: 6e400003-b5a3-f393-e0a9-e50e24dcca9e
Properties: NOTIFY
Descriptors:
Client Characteristic Configuration
UUID: 0x2902

2.9.2 二、示例验证步骤及结果:

串口发送 68 17 00 43 05 66 55 44 33 22 11 00 1B 6E 05 01 00 F3 01 02，可以在手机 APP 上收到该数据；同样的，手机 APP 推送 0x12345678，也可以在串口上打印出来，如下图：


21:57:04
0.04 KB/S
56%
Devices
DISCONNECT
BONDED
ADVERTISER
LS UART SERVER
EE:69:29:A8:C8:77
CONNECTED
NOT BONDED
CLIENT
SERVER
Generic Access
UUID: 0x1800
PRIMARY SERVICE
Generic Attribute
UUID: 0x1801
PRIMARY SERVICE
Nordic UART Service
UUID: 6e400001-b5a3-f393-e0a9-e50e24dcca9e
PRIMARY SERVICE
RX Characteristic
UUID: 6e400002-b5a3-f393-e0a9-e50e24dcca9e
Properties: WRITE, WRITE NO RESPONSE
Value: (0x) 12-34-56-78
TX Characteristic
UUID: 6e400003-b5a3-f393-e0a9-e50e24dcca9e
Properties: NOTIFY
Value: (0x) 68-17-00-43-05-66-55-44-33-22-11-00-1B-6E-05-01-00-F3-01-02
Descriptors:
Client Characteristic Configuration
UUID: 0x2902
Value: (0x) 01-00



2.9.3 三、特别说明:

1、关于数据从串口接收到蓝牙发送的处理

为什么使用软件定时器轮询处理数据收发？由于 uart_server 是单连接数据透传，因此串口上的数据是没有固定格式的，无法预知后后续会有多少字节的数据会被收到，因此每次只能接收 1 字节。接收 1 字节完成后，串口接收完成 callback 函数 HAL_UART_RxCpltCallback() 会被调用，将收到的数据保存到全局 buffer 里。而将收到的串口数据推送的蓝牙对端的动作在定时器里做，不在 HAL_UART_RxCpltCallback() 里做的原因，主要是考虑到中断回调函数里不宜有过多逻辑处理，且每次收到 1 字节就启动蓝牙发送会导致数据收发效率低下，因此在软件定时器里周期性检查、处理接收的数据是更为合理的选择。

2、关于 MTU

MTU 是 BLE ATT 的概念，它定义了在 ATT 层 Client 与 Server 之间任意数据包的最大长度。另外，由于 send_notification 和 write cmd 在 GATT 层不需要对端回复 response，这导致调用这两个接口进行数据发送时，一旦传入的 length 超出了一定范围（MTU - 3），超出该范围的数据会被丢弃。所以在 uart server 里，会对串口接收到的数据进行处理，每次调用 send_notification 接口进行数据发送时都会保证传入的数据都会被协议栈接收并处理。至于为何不选择有回复的 send_indication/write request，是因为这两个命令要求数据接收方在 GATT 层必须回复 response（空包，无实际数据内容），这会降低数据通信的效率，因此使用较少。

3、关于通信速率评估

既然是透传，某些特殊场景下可能就需要考虑数据通信速率。uart server 的实际通信速率会受若干因素影响，比如串口波特率、CPU 处理速度、MTU 长度、蓝牙射频性能、设备距离远近以及外部环境干扰等等，因此应用对通信速率如果有要求，需要事先评估。例如，115200 的波特率，串口通信最大速率在 100Kb/s 左右，那么整个系统的透传速率就不可能超过这个理论值；MTU 默认 23 字节时的通信速率，也肯定会低于更大 MTU 配置时的值，因此执行 MTU Exchange 也是提高通信速率的方式之一，等等。

4、关于更新广播间隔

更新广播间隔并非 uart_server 必备的功能，只是放在这个 demo 里作为一个简单的示例。在软件定时器里，会去检查 RTT input，如果检测到有字符输入，且是‘1’-‘9’之间，同时又有广播包在发送，则会调用 ls_uart_server_update_adv_interval() 更新广播包间隔。例如输入字符‘5’，则广播间隔会被更新为 500ms。

2.10 BLE_SINGLE_ROLE（单主/单从/一主一从串口透传）示例说明

例程路径：<install_file>/dev/examples/ble/ble_single_role

2.10.1 一、示例基本配置、流程及说明:

根据不同配置，BLE_SINGLE_ROLE 可以是一个单主机、单从机，或者一主一从的主从一体串口透传实例。从机与主机的功能是相互独立的，而从机的功能和 Uart_Server 基本一致，因此这里重点介绍单主机（MASTER_CLIENT_ROLE=1）功能。

1.1 准备工作之一：初始化

与 Uart_Server 一样，串口透传需要先初始化串口相关功能模块，包括软件定时器。调用的相关接口如下：

```
ls_uart_init(); // 串口模块初始化
ls_app_timer_init(); // 初始化软件定时器
HAL_UART_Receive_IT(&UART_Config, &uart_rx_buf[0], UART_SYNC_BYTE_LEN);  // 串口接收使能，每次接收 1byte，存放到 uart_rx_buf 的最开始
```

串口参数默认配置 IO 为 PB00/PB01，波特率为 115200，具体可以参考 ls_uart_init() 的实现。软件定时器周期配置为 50ms。这些都与 Uart_Server 一致除此之外，与 BLE 连接和 GATT Client 相关的数据结构也需要初始化：

```
ls_uart_client_init();
```

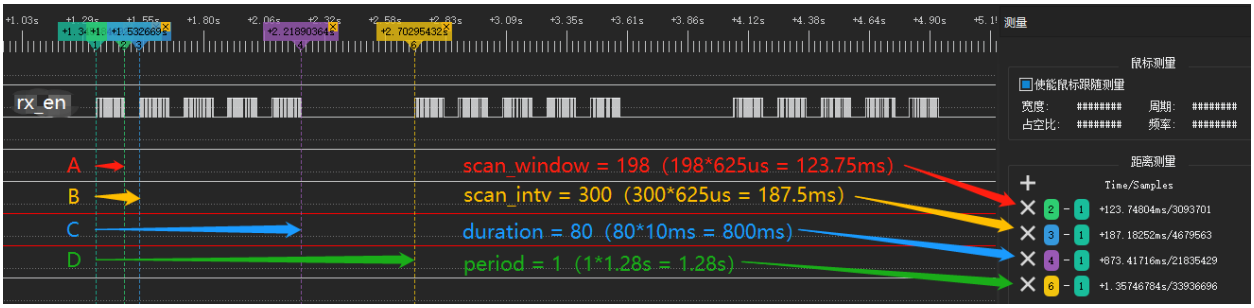
这其中主要包括 con_id/handle/MTU 的初始化等

1.2 打开 Scan/建立连接

BLE 相关的操作，作为主机，需要首先调用 create_scan_obj() 创建扫描对象。当 scan object 创建完成后，协议栈会产生一个 SCAN_OBJ_CREATED 消息，在这个消息的处理函数里，应用需要调用 create_init_obj() 再创建一个发起连接的对象。在两个对象都创建完成后，打开 scan，扫描空中的广播数据。函数具体实现为：

```
static void start_scan(void)
{
    LS_ASSERT(scan_obj_hdl != 0xff);
    struct start_scan_param scan_param = {
        .scan_intv = 100*3,
        .scan_window = 66*3,
        .duration = 80,
        .period = 1,
        .type = OBSERVER,
        .active = 0,
        .filter_duplicates = 0,
    };
    dev_manager_start_scan(scan_obj_hdl, &scan_param);
}
```

scan 的参数比较多，重点解释一下与时间相关的参数，如下图所示：



上图为映射到 IO 的硬件 rx_en 信号，其中高电平表示 RF 处于接收状态。

1、scan_window 表示单次 scan 开启的窗口大小，单位为 625 微秒，如图中 A 所示。

注解： 在单个 scan_window 内，rx_en 会有频繁的拉高拉低，其产生的原因是周边环境里有大量广播设备，RF 多次同步到空中的 ADV Access Address 信号 (0x8E89BED6)，每次 sync 成功都会导致 rx_en 信号的拉低，之后硬件又会自动拉高 rx_en 继续接收

2、scan_intv 表示一个 duration 内，相邻两次 scan 窗口开启间隔，单位为 625 微秒，如图中 B 所示。**scan_intv 必须大于等于 scan_window，否则协议栈会报错**

3、duration 为一次 scan 事件的持续时间，单位为 10ms。如果配置为 0，意味着 scan 事件不会终止，指导收到应用层面的终止命令。由于每次开启 scan 都要持续 scan_window 这么长的时间，因此 duration 的计时不会很准确

4、period 为连续两次 scan 事件之间的间隔，单位为 1.28s

其余的参数解释如下：

5、type 表示 scan 类型，可以配置的选项解释如下：

GENERAL_DISCOVERABLE：表示只能发现 adv flag 里有 general_discovery 或 limited_discovery 标志的广播包，同时该 scan 类型的 duration 实际为 10.24 秒，period 为 0

LIMITED_DISCOVERABLE：表示只能发现 adv flag 里有 limited_discovery 标志的广播包，同时该 scan 类型的 duration 实际为 10.24 秒，period 为 0

OBSERVER：表示最普通的 scan，限制条件最少。除了受 filter_duplicates 的配置影响外，所有的 adv 包都会被接收

OBSERVER_WHITELIST：表示只有在白名单里的设备发送的广播包才会被接收

CONNECTABLE：表示只接收可连接广播包

CONNECTABLE_WHITELIST：表示只接收在白名单里的可连接广播包

6、active 表示是否会发送 scan request

7、filter_duplicates 表示是否对重复地址的广播包进行过滤操作

在 single_role 示例里，scan type 为 OBSERVER。当收到的广播包地址与期望值匹配时，在 ADV_REPORT 消息里，会停止当前的 scan，而在随后的 SCAN_STOPPED 消息里，应用会调用 start_init() 发起连接：

```
static void start_init(uint8_t *peer_addr)
{
    struct dev_addr peer_dev_addr_str;
    memcpy(peer_dev_addr_str.addr, peer_addr, BLE_ADDR_LEN);
    struct start_init_param init_param = {
        .scan_intv = 64,
        .scan_window = 48,
        .conn_to = 0,
        .conn_intv_min = 16,
        .conn_intv_max = 16,
        .conn_latency = 0,
        .supervision_to = 200,

        .peer_addr = &peer_dev_addr_str,
        .peer_addr_type = dev_addr_type,
        .type = DIRECT_CONNECTION,
    };
    dev_manager_start_init(init_obj_hdl,&init_param);
}
```

其中 scan_intv/scan_window 的含义与 scan 的参数含义一致。

conn_to：表示 connection timeout，表示尝试连接的超时时间，以 10ms 为单位。设置为 0 表示没有超时时间。注意该参数只有在 type 为 AUTO_CONNECTION_WHITELIST 时才生效

注解： start_init 里的 type 为 DIRECT_CONNECTION 时，假如需要对 init 行为增加 timeout，需要使用额外的 timer（比如 builtin timer 或外设 timer），在 timeout 触发时调用 dev_manager_stop_init 结束 init 行为

---

conn_intv_min：表示最小连接间隔，以 1.25ms 为单位，容许的范围是 7.5ms 到 4s

conn_intv_max：表示最大连接间隔，以 1.25ms 为单位，容许的范围是 7.5ms 到 4s

conn_latency：表示 slave latency，连接建立之后，slave 可以连续不回应的事件个数

supervision_to：表示 supervision timeout，以 10ms 为单位，容许的范围是 100ms 到 32s

peer_add：表示待连接的设备地址

peer_addr_type：表示待连接的设备地址类型，0 表示 public，1 表示 random

type：表示连接类型。DIRECT_CONNECTION 表示连接 peer_addr 和 peer_addr_type 指定的设备，AUTO_CONNECTION_WHITELIST 表示主机会尝试与所有在白名单里的设备自动建立连接

在建立连接成功之后，协议栈会产生一个 GAP 的 CONNECTED 消息到应用层，demo 里会在这个消息里调用 gatt_manager_client_mtu_exch_send() 进行 MTU 交换，进而触发 GATT 层的后续行为。

1.3 GATT 服务发现流程

single_role demo 的服务发现流程如下图：

服务发现流程主要遵循如下流程：

step 1：主服务发现，调用 gatt_manager_client_svc_discover_by_uuid() 接口，之后应用会收到

CLIENT_PRIMARY_SVC_DIS_IND 消息

step 2: TX 特征值发现：调用 gatt_manager_client_char_discover_by_uuid() 接口，TX UUID 作为参数，之后应用会收到 CLIENT_CHAR_DIS_BY_UUID_IND 消息

step 3: RX 特征值发现：调用 gatt_manager_client_char_discover_by_uuid() 接口，RX UUID 作为参数，之后应用会收到 CLIENT_CHAR_DIS_BY_UUID_IND 消息

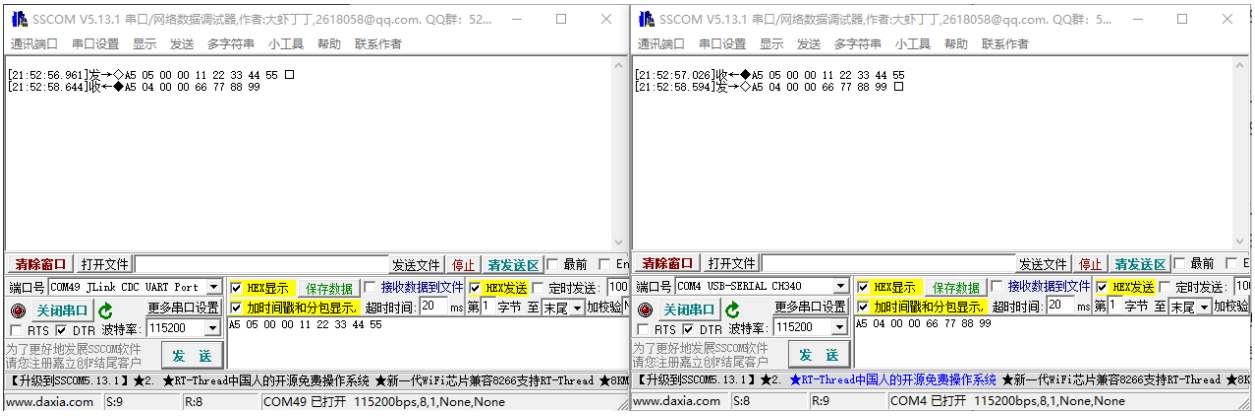
step 4：CCCD 发现：调用 gatt_manager_client_desc_char_discover() 接口，之后会收到 CLIENT_CHAR_DESC_DIS_BY_UUID_IND 消息

step 5：CCCD 使能：调用 gatt_manager_client_cccd_enable() 接口，其中 notification 设置为 enable，indication 设置为 disable。实际为 write no rsp 操作，所以会收到 CLIENT_WRITE_NO_RSP_DONE 消息

至此所有的 Uart Server 服务发现全部完成。

2.10.2 二、示例验证步骤及结果:

single_role 的 master 和 slave 建立连接后，通过 PC 端串口助手连接两个设备的 Uart，分别发送固定格式的数据，可以看到对端能收到相应数据，如下图所示：



2.10.3 三、特别说明:

1、关于串口透传数据格式说明

在 Uart Server demo 里，串口上的数据是完全没有格式的，因此每次只需要接收 1 个 byte，累积的数据在定时器里周期性发送出去。而在 single role demo 里，由于有可能有多连接（一主一从），因此需要通过 connection id 来区分。这就要求串口过来的数据必须指明接收的 connection 对端，进而要求串口传输的数据有一定的数据格式。在 single role demo 里，串口数据格式是 sync_byte(1byte, 默认 0xA5)+length(2bytes)+connection_id(1byte)+data(length bytes)

2.11 CRYPT 设备使用示例

例程路径：<install_file>/dev/examples/peripheral/crypt

2.11.1 操作简介:

crypt 测试文件夹中有两个测试例程，这个两个例程分别为阻塞模式与中断模式的例程，在这两个模式下分别测试了：

1. ECB 模式下，密钥长度分别为 128 位、192 位与 256 位加解密的例程。
2. CBC 模式下，密钥长度分别为 128 位、192 位与 256 位加解密的例程。

所有例程测试的流程均是先将明文进行加密然后与我们的密文进行对比，（住：我们的所有明文与密文均是在文档中进行复制过来的，也就是说这个密文都是经过验证过的，并且是绝对无误的）如果一致则打印加密成功，否则打印加密失败。

2.11.2 操作步骤:

1. 将编译好的程序下载到测试的蓝牙芯片中，注意需要下载 info_sbl.hex, crypt_test.hex 这两个文件，或者只下载 crypt_test_production.hex 这一个文件也可以，（住：以 “_production” 这个文件名结尾的均为合并文件，只用下载这一个文件便可）。
2. 使用编译器仿真可以比较两个数组的值或者使用”LOG_I” 函数在”J-Link RTT Viewer” 软件中打印出来。

2.11.3 预期结果:

在”J-Link RTT Viewer” 软件中打印: I/NO_TAG:CRYPT_AES_ECB_ENCRYPT_128_TEST_SUCCESS!
I/NO_TAG:CRYPT_AES_ECB_DECRYPT_128_TEST_SUCCESS! I/NO_TAG:CRYPT_AES_CBC_ENCRYPT_128_TEST_SUCCE
I/NO_TAG:CRYPT_AES_CBC_DECRYPT_128_TEST_SUCCESS! I/NO_TAG:CRYPT_AES_ECB_ENCRYPT_192_TEST_SUCCE
I/NO_TAG:CRYPT_AES_ECB_DECRYPT_192_TEST_SUCCESS! I/NO_TAG:CRYPT_AES_CBC_ENCRYPT_192_TEST_SUCCE
I/NO_TAG:CRYPT_AES_CBC_DECRYPT_192_TEST_SUCCESS! I/NO_TAG:CRYPT_AES_ECB_ENCRYPT_256_TEST_SUCCE
I/NO_TAG:CRYPT_AES_ECB_DECRYPT_256_TEST_SUCCESS! I/NO_TAG:CRYPT_AES_CBC_ENCRYPT_256_TEST_SUCCE
I/NO_TAG:CRYPT_AES_CBC_DECRYPT_256_TEST_SUCCESS!

2.12 BLE_HID_DMIC（HID 设备间的语音数据交互）示例说明

例程路径：<install_file>/dev/examples/ble/ble_hid_dmic

2.12.1 一、示例基本说明、配置及流程:

关于普通服务的添加可以参考 ble_uart_server 示例说明，关于配对加密绑定流程可以参考 BLE 工作流程中关于 GAP 的介绍，这里重点介绍和语音数据处理相关的功能。

1.1 基本说明

关于 hid： HID 设备是通过报告 (Report) 来给传送数据的，报告有输入特性 (Input)、输出特性 (Output) 和 Feature 特性。输入特性是 HID 设备发送数据到主机，类似 notify 特性；输出特性是主机发送数据到 HID 设备，类似 Write 特性；Feature 特性有输入和输出特性。

关于报告描述符： 报告描述符 (Report Descriptor) 是用来描述 HID 设备中的报告以及报告里面的数据怎么用的。一个报告描述符可以描述多个报告，不同的报告通过 Report ID 来识别。通过报告描述符，主机可以分析出报告里面的数据所表示的意思。

```
const uint8_t hid_report_map[] =
{
    0x05, 0x01, //Usage Page(Generic Desktop) Usage Page 用来指定 HID 设备的某功能项,
↪Usage Page 相当于是 HID 用途的子集合。
    0x09, 0x06, //Usage(Keyboard) Usage 用来指定 Usage Page 的某功能项,Usage 相当于是
Usage Page 的子集合。
    0xA1, 0x01, //Collection(Application)
    0x05, 0x07, //Usage Page(Keyboard)
    0x09, 0x06, //Usage(Keyboard c and C)
    0xA1, 0x01, //Collection(Application) 集合项和 End Collection 搭配使用。
    0x85, 0x01, //Report Id(1) 报告 id 号，由报告 id 生成对应的句柄进行数据交互。
    0x95, 0x08, //Report Count(8) 用来设定告字段的数目，和 Report Size 搭配使用。
    0x75, 0x08, //Report Size(8)  用来设定报告字段的大小，单位是位 (bit)。
    0x15, 0x00, //Logical minimum(0) 报告字段的最小逻辑数值范围，与上面的 Report Size 相对应。
    0x25, 0xFF, //Logical maximum(255) 报告字段的最大逻辑数值范围，即一个报告字段 (8 bits) 能
表示的最大逻辑值为 255。
    0x19, 0x00, //Usage Minimum(No event indicated) 用来表示 Usage(Keyboard) 功能项中使用
Key Codes 的最小键值范围，和 Usage 搭配使用。
    0x29, 0xFF, //Usage Maximum(Reserved (0x00FF)) 用来表示 Usage(Keyboard) 功能项中使用
Key Codes 的最大键值范围，和 Usage 搭配使用。
    0x81, 0x00, //Iuput(Data,Array,Absolute,Bit Field) 表示数据到主机的数据格式。
    0xC0,       //End Collection 集合项的结束标志

    0x05, 0x0C,       //Usage Page(Consumer)
```

(下页继续)

(续上页)

```
    0x09, 0x01,         //Usage(Consumer Control)
    0xA1, 0x01,         //Collection(Application)
    0x85, 0x03,         //Report Id(3)
    0x19, 0x00,         //Usage Minimum(No event indicated)
    0x2A, 0x9D, 0x02,   //Usage Maximum(Reserved (0x029D))
    0x15, 0x00,         //Logical minimum(0)
    0x26, 0x9D, 0x02,   // Logical maximum(669)
    0x75, 0x10,         //Report Size(16)
    0x95, 0x02,         //Report Count(2)
    0x81, 0x00,         //Iuput(Data,Value,Absolute,Bit Field)
    0xC0,               //End Collection

    0x06, 0x00, 0xFF, //Usage Page(Vendor-defined 0xFF00)
    0x09, 0x00, //Usage(Vendor-defined 0x0000)
    0xA1, 0x01,         //Collection(Application)
    0x85, 0x5A,         //Report Id(90)
    0x95, 0xFF, //Report Count(255)
    0x75, 0x08, //Report Size(8)
    0x15, 0x00, //Logical minimum(0)
    0x25, 0xFF, //Logical maximum(255)
    0x19, 0x00, //Usage Minimum(No event indicated)
    0x29, 0xFF, //Usage Maximum(Reserved (0x00FF))
    0x81, 0x00,         //Iuput(Data,Value,Absolute,Bit Field)
    0xC0,               //End Collection
    0xC0,               //End Collection
};
```

注解： 上面的描述符 (descriptor) 指定了以下报告 (report)。

Keyboard Input Report

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Key code							
1	Key code							
2	Key code							
3	Key code							
4	Key code							
5	Key code							
6	Key code							
7	Key code							

Consumer Control Input Report

Byte	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Consumer Control Dtata															
1	Consumer Control Dtata															

Vendor-defined Input Report								
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	data							
1	data							
2	data							
3	data							
...	data							
253	data							
254	data							

1.2 基本配置

关于 hid 的 service 配置和 hid profile 的添加：

```
struct hid_db_cfg db_cfg;///配置 hid 结构体变量
    db_cfg.hids_nb = 1;///默认配置 hid 的个数为 1
    db_cfg.cfg[0].svc_features = HID_PROTO_MODE;///配置为 report 模式
    db_cfg.cfg[0].report_nb = 3;///配置 report 的个数，依据于 report map
    db_cfg.cfg[0].report_id[0] = 1;///配置 report 的索引号与 report id 的关系，发数据时根据
report 的索引号找到相应的 report id
    db_cfg.cfg[0].report_id[1] = 3;
    db_cfg.cfg[0].report_id[2] = 90;

    db_cfg.cfg[0].report_cfg[0] = HID_REPORT_IN;///report 索引号对应的 report id 特性
    db_cfg.cfg[0].report_cfg[1] = HID_REPORT_IN;
    db_cfg.cfg[0].report_cfg[2] = HID_REPORT_FEAT;

    db_cfg.cfg[0].info.bcdHID = 0;///HID 的规格发布号，默认为 0
    db_cfg.cfg[0].info.bCountryCode = 0;///默认为 0
    db_cfg.cfg[0].info.flags = HID_WKUP_FOR_REMOTE;///配置为远程可唤醒设备
    dev_manager_prf_hid_server_add(NO_SEC, &db_cfg, sizeof(db_cfg));///添加 hid profile
的函数调用
```

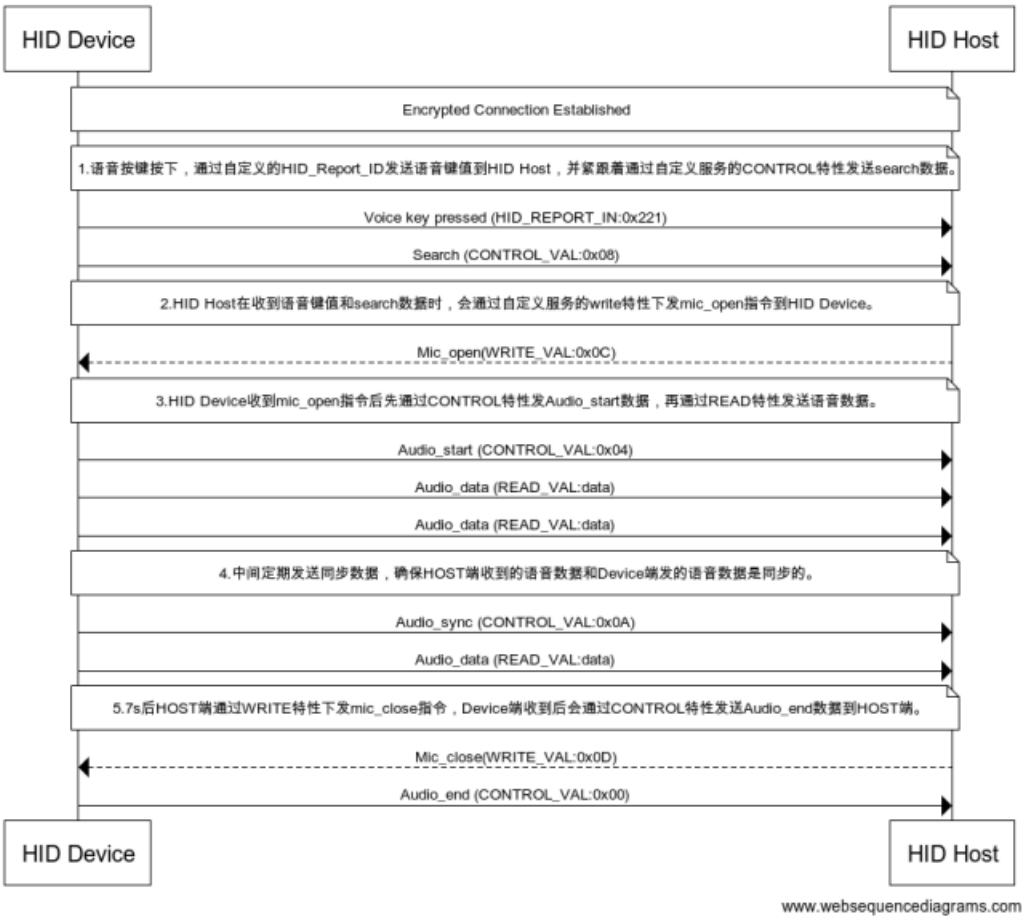
注解： 1.hid 服务添加成功之后会在 dev event:PROFILE_ADDED 中调用 prf_added_handler 函数，然后初始化 hid 事件处理的回调函数：prf_hid_server_callback_init(prf_hid_server_callback)。2. 关于 pdm 的配置参考 dev/examples/peripheral/pdm。

1.3 事件处理

关于 hid 的事件处理回调函数：prf_hid_server_callback

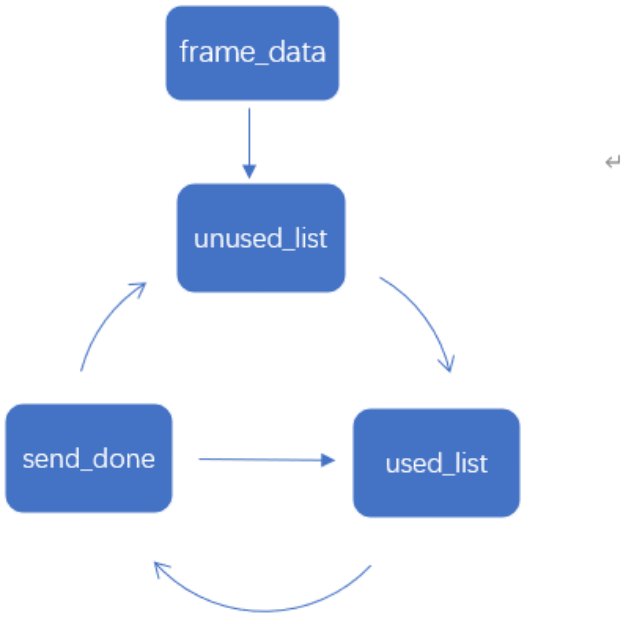
```
static void prf_hid_server_callback(enum hid_evt_type type, union hid_evt_u *evt,␣
↪uint8_t con_idx)
{
    uint16_t ntf_cfg;
    switch (type)
    {
    case HID_REPORT_READ:///回复主机 read hid 设备的特性
        evt->read_report_req.length = 0;
        if(evt->read_report_req.type == APP_HOGPD_REPORT_MAP)
        {
            evt->read_report_req.value = (uint8_t *)hid_report_map;// 主机读取 report␣
↪map 信息
            evt->read_report_req.length = HID_REPORT_MAP_LEN;
        }
        break;
    case HID_NTF_CFG:///回复主机读 hid ntf_cfg 的配置值
        LOG_I("hid_ntf_cfg save flash record_key1 : %x",evt->ntf_cfg.value);
        ntf_cfg = evt->ntf_cfg.value;
        tinyfs_write(hid_dir, RECORD_KEY1, (uint8_t*)&ntf_cfg, sizeof(ntf_cfg));
        tinyfs_write_through();
        break;
    case HID_NTF_DONE:///hid 设备调用 app_hid_send_keyboard_report 函数，发送完数据后的处理
        LOG_I("HID NTF DONE");
        break;
    case HID_REPORT_WRITE:///回复主机 write hid 设备的特性
        LOG_I("HID REPORT WRITE");
        break;
    default:
        break;
    }
}
```

1.4 主从机数据交互流程



1.5 语音数据缓存处理

语音数据缓存：



1.先对 unused_list 初始化，同时申请固定大小的缓存空间(8k 左右)。
2.当一帧语音数据压缩成功后，若 unused_list 的缓存空间够用，则会向 unused_list pop 出固定大小的空间并 push 到 used_list。
3.当 used_list 不为空时，会 pop 出一帧语音数据并调用 send 函数，当数据发送完毕后会把缓存本帧数据的空间 push 到 unused_list 中，并继续检查 used_list 是否为空。

2.12.2 二、特别说明:

通过 PDM 调制输出 PCM 数据流后对压缩算法的要求：

2.1 谷歌 ADPCM 压缩格式要求：

2.1.1 采样数据速率：8Khz/16Khz--16bit，即 1ms 有 8*16bit/16*16bit PCM 的数据。

2.1.2 压缩比：PCM->ADPCM 为 4:1，即 1ms 有 4bytes/8bytes ADPCM 的数据。

2.1.3 Header： Aduio frame Number：2 bytes

remote control id：1 byte

previous predicted ADPCM value：2 bytes

Index into step size table：1 byte

2.1.4 Payload： adpcm data：128 bytes

2.2 三星 MSBC 压缩格式要求：

2.2.1 采样数据速率：16Khz--16bit，即 1ms 有 16*16bit PCM 的数据。

2.2.2 压缩比：PCM->MSBC 为 4:1，即 1ms 有 8bytes MSBC 的数据。

2.2.3 Header： Frame Num：1byte

Frame start flag：3bytes(固定为 0xAD,0x00,0x00)

CRC：1byte

Scaler factor：4bytes

2.2.4 Payload： msbc data：49bytes

CHAPTER 3

API Reference

3.1 BLE API

3.1.1 LSBLE API

Defines

INVALID_CON_IDX
Invalid connection index

INVALID_PEER_ID
Invalid Peer ID

BLE_ADDR_LEN
BLE mac address length

BLE_KEY_LEN
The length of the paired temporary key

ADV_DATA_PACK(*buf*, *field_nums*, ...)

Enums

enum gap_own_addr_type

Own address type.

Values:

enumerator PUBLIC_OR_RANDOM_STATIC_ADDR

Public or random static address

enumerator RESOLVABLE_PRIVATE_ADDR

Resolvable private address

enumerator NON_RESOLVABLE_PRIVATE_ADDR

Non-resolvable private address

enum gap_peer_addr_type

Peer address type.

Values:

enumerator PUBLIC_ADDR

Public address

enumerator RANDOM_ADDR

Random address

enum gap_adv_type

Adv type definition.

Values:

enumerator GAP_ADV_TYPE_FLAGS

Flag

enumerator GAP_ADV_TYPE_MORE_16_BIT_UUID

Use of more than 16 bits UUID

enumerator GAP_ADV_TYPE_COMPLETE_LIST_16_BIT_UUID

Complete list of 16 bit UUID

enumerator GAP_ADV_TYPE_MORE_32_BIT_UUID

Use of more than 32 bit UUD

enumerator GAP_ADV_TYPE_COMPLETE_LIST_32_BIT_UUID

Complete list of 32 bit UUID

enumerator GAP_ADV_TYPE_MORE_128_BIT_UUID

Use of more than 128 bit UUID

enumerator GAP_ADV_TYPE_COMPLETE_LIST_128_BIT_UUID

Complete list of 128 bit UUID

enumerator GAP_ADV_TYPE_SHORTENED_NAME
Shortened device name

enumerator GAP_ADV_TYPE_COMPLETE_NAME
Complete device name

enumerator GAP_ADV_TYPE_TRANSMIT_POWER
Transmit power

enumerator GAP_ADV_TYPE_CLASS_OF_DEVICE
Class of device

enumerator GAP_ADV_TYPE_SP_HASH_C
Simple Pairing Hash C

enumerator GAP_ADV_TYPE_SP_RANDOMIZER_R
Simple Pairing Randomizer

enumerator GAP_ADV_TYPE_TK_VALUE
Temporary key value

enumerator GAP_ADV_TYPE_OOB_FLAGS
Out of Band Flag

enumerator GAP_ADV_TYPE_SLAVE_CONN_INT_RANGE
Slave connection interval range

enumerator GAP_ADV_TYPE_RQRD_16_BIT_SVC_UUID
Require 16 bit service UUID

enumerator GAP_ADV_TYPE_RQRD_32_BIT_SVC_UUID
Require 32 bit service UUID

enumerator GAP_ADV_TYPE_RQRD_128_BIT_SVC_UUID
Require 128 bit service UUID

enumerator GAP_ADV_TYPE_SERVICE_16_BIT_DATA
Service data 16-bit UUID

enumerator GAP_ADV_TYPE_SERVICE_32_BIT_DATA
Service data 32-bit UUID

enumerator GAP_ADV_TYPE_SERVICE_128_BIT_DATA
Service data 128-bit UUID

enumerator GAP_ADV_TYPE_PUB_TGT_ADDR
Public Target Address

enumerator GAP_ADV_TYPE_RAND_TGT_ADDR
Random Target Address

enumerator GAP_ADV_TYPE_APPEARANCE

Appearance

enumerator GAP_ADV_TYPE_ADV_INTV

Advertising Interval

enumerator GAP_ADV_TYPE_LE_BT_ADDR

LE Bluetooth Device Address

enumerator GAP_ADV_TYPE_LE_ROLE

LE Role

enumerator GAP_ADV_TYPE_SPAIR_HASH

Simple Pairing Hash C-256

enumerator GAP_ADV_TYPE_SPAIR_RAND

Simple Pairing Randomizer R-256

enumerator GAP_ADV_TYPE_3D_INFO

3D Information Data

enumerator GAP_ADV_TYPE_MANU_SPECIFIC_DATA

Manufacturer specific data

enum sec_lvl_type

Type of security level.

Values:

enumerator NO_SEC

No security

enumerator UNAUTH_SEC

Unauthenticated security

enumerator AUTH_SEC

Authenticated security

enumerator SEC_CON_SEC

Security connection

enum adv_disc_mode

ADV discovery mode.

Values:

enumerator ADV_MODE_NON_DISC

Mode in non-discoverable

enumerator ADV_MODE_GEN_DISC

Mode in general discoverable

enumerator ADV_MODE_LIM_DISC

Mode in limited discoverable

enumerator ADV_MODE_BEACON

Broadcast mode without presence of AD_TYPE_FLAG in advertising data

enumerator ADV_MODE_MAX

enum phy_type

PHY type.

Values:

enumerator PHY_TYPE_1M

LE 1Mbps phy

enumerator PHY_TYPE_2M

LE 2Mbps phy

enumerator PHY_TYPE_CODED

LE Coded phy

enum scan_type

Scanning type.

Values:

enumerator GENERAL_DISCOVERABLE

General discovery

enumerator LIMITED_DISCOVERABLE

Limited discovery

enumerator OBSERVER

Observer

enumerator OBSERVER_WHITELIST

Selective observer

enumerator CONNECTABLE

Connectable discovery

enumerator CONNECTABLE_WHITELIST

Selective connectable discovery

enum filter_dup_policy

Scan filter duplicates policy.

Values:

enumerator DUP_FILT_DIS

Disable filtering of duplicated packets

enumerator DUP_FILT_EN

Enable filtering of duplicated packets

enumerator DUP_FILT_EN_PERIOD

Enable filtering of duplicated packets, reset for each scan period

enum init_type

Initiating type.

Values:

enumerator DIRECT_CONNECTION

Direct connection establishment, establish a connection with an indicated device

enumerator AUTO_CONNECTION_WHITELIST

Automatic connection establishment, establish a connection with all devices whose address is present in the white list

enum dev_evt_type

Type of events in device manager.

Values:

enumerator STACK_INIT

Stack initialized event

enumerator STACK_READY

Stack ready event

enumerator PROFILE_ADDED

Profile added event

enumerator SERVICE_ADDED

Service added event

enumerator ADV_OBJ_CREATED

Adv object created event

enumerator SCAN_OBJ_CREATED

Scan object created event

enumerator INIT_OBJ_CREATED

Initiate object created event

enumerator ADV_STOPPED

Adv stopped event

enumerator SCAN_STOPPED

Scan stopped event

enumerator INIT_STOPPED

Initiate stopped event

enumerator OBJ_DELETED

Object deleted event

enumerator ADV_REPORT

Receive adv report event

enum prf_id

Profile IDs.

Values:

enumerator PRF_DIS_SERVER

Device information service profile

enumerator PRF_MESH

enumerator PRF_LS_MESH

enumerator PRF_FOTA_SERVER

FOTA server service

enumerator PRF_HID

HID profile

enumerator PRF_BASS

Battery service profile

enum adv_report_type

Advertising report type.

Values:

enumerator REPORT_TYPE_ADV_EXT

Extended advertising report

enumerator REPORT_TYPE_ADV_LEG

Legacy advertising report

enumerator REPORT_TYPE_SCAN_RSP_EXT

Extended scan response report

enumerator REPORT_TYPE_SCAN_RSP_LEG

Legacy scan response report

enumerator REPORT_TYPE_PER_ADV

Periodic advertising report

enum uuid_length

Length of UUID.

Values:

enumerator UUID_LEN_16BIT

16bits UUID

enumerator UUID_LEN_32BIT

32bits UUID

enumerator UUID_LEN_128BIT

128bits UUID

enum svc_att_perm

Service and attribute permissions definition.

Values:

enumerator PERM_NO_AUTH

NON Authenticated

enumerator PERM_UNAUTH

Unauthenticated

enumerator PERM_AUTH

Authenticated

enumerator PERM_SEC_CON

Security connection

enum gap_evt_type

GAP event types enumeration.

Values:

enumerator CONNECTED

Connected event

enumerator DISCONNECTED

Disconnected event

enumerator CONN_PARAM_REQ

Connection parameter request event

enumerator CONN_PARAM_UPDATED

Connection parameter updated event

enumerator MASTER_PAIR_REQ

Master pair request event

enumerator SLAVE_SECURITY_REQ

Slave security request event

enumerator PAIR_DONE

Pair done event

enumerator ENCRYPT_FAIL
Encryption fail event

enumerator ENCRYPT_DONE
Encryption done event

enumerator DISPLAY_PASSKEY
Display passkey event

enumerator REQUEST_PASSKEY
Request passkey event

enumerator NUMERIC_COMPARE
Numeric compare event

enumerator REQUEST_LEGACY_OOB
Request legacy OOB event

enumerator REQUEST_SC_OOB
Request security connection event

enumerator GET_DEV_INFO_DEV_NAME
Get device name of device information

enumerator GET_DEV_INFO_APPEARANCE
Get appearance Icon of device information

enumerator GET_DEV_INFO_SLV_PRE_PARAM
Get slave preferred parameters of device information

enumerator GET_DEV_INFO_PEER_RSSI
Get connection RSSI indication

enum LS_BLE_ROLE
BLE roles enumeration.

Values:

enumerator LS_BLE_ROLE_MASTER
Role of master

enumerator LS_BLE_ROLE_SLAVE
Role of slave

enum gap_io_caps
Defgroup BLE_GAP_IO_CAPS GAP IO Capabilities.

Values:

enumerator BLE_GAP_IO_CAPS_DISPLAY_ONLY
Display Only

enumerator BLE_GAP_IO_CAPS_DISPLAY_YESNO

Display and Yes/No entry

enumerator BLE_GAP_IO_CAPS_KEYBOARD_ONLY

Keyboard Only

enumerator BLE_GAP_IO_CAPS_NONE

No I/O capabilities

enumerator BLE_GAP_IO_CAPS_KEYBOARD_DISPLAY

Keyboard and Display

enum gap_pair_oob

Defgroup SEC_OOB_FLAG OOB Data Flag.

Values:

enumerator BLE_GAP_OOB_DISABLE

OOB Authentication data not present

enumerator BLE_GAP_OOB_ENABLE

OOB Authentication data from remote device present

enum gap_pair_auth

Defgroup SEC_AUTH_FLAG SEC Auth Flag.

Values:

enumerator AUTH_NONE

No auth requirement

enumerator AUTH_BOND

Bond flag

enumerator AUTH_MITM

MITM flag

enumerator AUTH_SEC_CON

Security connection flag

enumerator AUTH_KEY_PRESS_NOTIFY

Key press notify flag

enum gap_key_dist

Defgroup SEC_KEY_DIST_FLAG SEC Key Distribution Flag.

Values:

enumerator KDIST_NONE

No key needs to be distributed

enumerator KDIST_ENCKEY

Distribute encryption and master identification info

enumerator KDIST_IDKEY

Distribute identity and address info

enumerator KDIST_SIGNKEY

Distribute signing info

enum gatt_evt_type

GATT event types.

注解: Event types include request/done/indication or other messages sent to applications by stack.

Values:

enumerator SERVER_READ_REQ

Read request for server

enumerator SERVER_WRITE_REQ

write request for server

enumerator SERVER_NOTIFICATION_DONE

Send notification done for server

enumerator SERVER_INDICATION_DONE

Send indication done for server

enumerator CLIENT_RECV_NOTIFICATION

Receive notification for client

enumerator CLIENT_RECV_INDICATION

Receive indication for client

enumerator CLIENT_PRIMARY_SVC_DIS_IND

Primary service discovery indication for client

enumerator CLIENT_INCL_SVC_DIS_IND

Included service discovery indication for client

enumerator CLIENT_CHAR_DIS_BY_UUID_IND

Characteristic discovery by UUID indication for client

enumerator CLIENT_CHAR_DESC_DIS_BY_UUID_IND

Characteristic descriptor discovery by UUID indication for client

enumerator CLIENT_RD_CHAR_VAL_BY_UUID_IND

Read characteristic value by UUID indication for client

enumerator CLIENT_WRITE_WITH_RSP_DONE

Write response indication for client

enumerator CLIENT_WRITE_NO_RSP_DONE

Write with no response indication for client

enumerator MTU_CHANGED_INDICATION

MTU exchange indication for client & server

enumerator GATT_EVT_MAX

enum svc_set_value_status

Service set values status.

Values:

enumerator SVC_SET_VAL_NO_ERROR

No error

enumerator SVC_SET_VAL_NOT_SUPPORTED

Action not supported

enumerator SVC_SET_VAL_INVALID_HANDLE

Invalid handle

enumerator SVC_SET_VAL_INVALID_OFFSET

Invalid offset

enumerator SVC_SET_VAL_INVALID_LENGTH

Invalid length

enum svc_get_value_status

Service get values status.

Values:

enumerator SVC_GET_VAL_NO_ERROR

No error

enumerator SVC_GET_VAL_NOT_SUPPORTED

Action not supported

enumerator SVC_GET_VAL_INVALID_HANDLE

Invalid handle

enumerator SVC_GET_VAL_APP_ERROR

Application or profile error

Functions

uint8_t *__adv_data_pack__(uint8_t *_buf_, uint8_t _field_nums_, ...)

Packing function for advertising data and scan response. Arguments passed to the function must follow the sequence of adv_type/data_buf/length.

参数

- **buf** -- **[in]** Buffer contains packed data.
- **field_nums** -- **[in]** Number of adv data or scan response types.

返回 Packed data length in bytes.

void __ble_init__(void)

Function for BLE initialization.

void __ble_loop__(void)

Function for BLE event handling with an internal infinite loop. Any function behind ble_loop will never be executed.

void __dev_manager_init__(void (*_cb_)) __enum__ _dev_evt_type_, __union__ _dev_evt_u_*

Initialization of dev_manager.

参数 **cb** -- **[in]** Callback function to handle all the dev_manager messages.

void __dev_manager_stack_init__(__struct__ _ble_stack_cfg_ *_cfg_)

Initialization of dev_manager stack.

参数 **cfg** -- **[in]** BLE stack configuration.

void __dev_manager_get_identity_bdaddr__(uint8_t *_addr_, bool *_random_)

Initialization of dev_manager stack.

参数

- **addr** -- **[out]** Pointer to identity address.
- **random** -- **[out]** Indicate if identity address is a public (False) or static private random (True) address.

void __dev_manager_add_service__(__struct__ _svc_decl_ *_svc_)

Add service.

参数 **svc** -- **[in]** Pointer to service to added.

uint8_t __dev_manager_svc_set_value__(uint16_t _handle_, uint16_t _length_, uint8_t *_value_)

Set value for specified attribute.

参数

- **handle** -- **[in]** Attribute handle.
- **length** -- **[in]** Length of the value.
- **value** -- **[in]** Pointer to value to set.

uint8_t **dev_manager_svc_get_value**(uint16_t *handle*, uint8_t **value*, uint16_t **length*)

Get value for specified attribute.

参数

- **handle** -- **[in]** Attribute handle.
- **length** -- **[out]** Length of the value.
- **value** -- **[out]** Pointer to value to get.

void **dev_manager_create_legacy_adv_object**(**struct** *legacy_adv_obj_param* **p_param*)

Create legacy adv object.

参数 **p_param** -- **[in]** Parameter for legacy adv object.

void **dev_manager_create_ext_adv_object**(**struct** *ext_adv_obj_param* **p_param*)

Create extended adv object.

参数 **p_param** -- **[in]** Parameter for extended adv object.

void **dev_manager_create_scan_object**(**enum** *gap_own_addr_type own_addr_type*)

Create scan object.

参数 **own_addr_type** -- **[in]** Parameter for scan object.

void **dev_manager_create_init_object**(**enum** *gap_own_addr_type own_addr_type*)

Create initiate object.

参数 **own_addr_type** -- **[in]** Own address type.

void **dev_manager_start_adv**(uint8_t *adv_handle*, uint8_t **adv_data*, uint8_t *adv_data_length*, uint8_t **scan_rsp_data*, uint8_t *scan_rsp_data_length*)

Start advertising.

参数

- **adv_handle** -- **[in]** Handle of adv object.
- **adv_data** -- **[in]** Adv data.
- **adv_data_length** -- **[in]** Length of adv data.
- **scan_rsp_data** -- **[in]** Scan response data.
- **scan_rsp_data_length** -- **[in]** Length of Scan response data.

void **dev_manager_set_adv_duration**(uint16_t *duration*)

Set duration of adv.

参数 **duration** -- **[in]** Duration of adv activity(in unit of 10ms). 0 means that advertising continues until the application disable it.

void **dev_manager_update_adv_data**(uint8_t *adv_handle*, uint8_t **adv_data*, uint8_t *adv_data_length*, uint8_t **scan_rsp_data*, uint8_t *scan_rsp_data_length*)

Update advertising data or scan response data.

参数

- **adv_handle** -- **[in]** Handle of adv object.
- **adv_data** -- **[in]** Adv data.
- **adv_data_length** -- **[in]** Length of adv data.
- **scan_rsp_data** -- **[in]** Scan response data.
- **scan_rsp_data_length** -- **[in]** Length of Scan response data.

void **dev_manager_stop_adv**(uint8_t *adv_handle*)

Stop advertising.

参数 **adv_handle** -- **[in]** Handle of adv object.

void **dev_manager_start_scan**(uint8_t *scan_handle*, **struct** *start_scan_param* **param*)

Start scan.

参数

- **scan_handle** -- **[in]** Handle of scan object.
- **param** -- **[in]** Parameter for scan object.

void **dev_manager_stop_scan**(uint8_t *scan_handle*)

Stop scan.

参数 **scan_handle** -- **[in]** Handle of scan object.

void **dev_manager_start_init**(uint8_t *init_handle*, **struct** *start_init_param* **param*)

Start initiate.

参数

- **init_handle** -- **[in]** Handle of init object.
- **param** -- **[in]** Parameter for init object.

void **dev_manager_stop_init**(uint8_t *init_handle*)

Stop initiate.

参数 **init_handle** -- **[in]** Handle of init object.

void **dev_manager_delete_activity**(uint8_t *obj_hdl*)

Delete activity.

参数 **obj_hdl** -- **[in]** Handle of object to delete.

void **dev_manager_set_mac_addr**(uint8_t **addr*)

Set mac address.

参数 **addr** -- **[in]** Pointer to mac address to set.

uint8_t **dev_manager_update_adv_interval**(uint8_t *adv_handle*, uint32_t *new_intv_min*, uint32_t *new_intv_max*)

Update adv interval. The new intervals will not be applied until the adv activity is restarted.

参数

- **adv_handle** -- **[in]** Handle of the adv activity to be updated.
- **new_intv_min** -- **[in]** Minimum new adv interval.
- **new_intv_max** -- **[in]** Maxmium new adv interval.

返回 Status of updating. 0 means NO_ERROR, otherwise means invalid handle of adv.

void **gap_manager_init**(void (**evt_cb*)) **enum** *gap_evt_type*, **union** *gap_evt_u**, uint8_t

Initialization of gap_manager.

参数 **evt_cb** -- **[in]** Callback function to handle all the gap_manager messages.

void **gap_manager_disconnect**(uint8_t *con_idx*, uint8_t *reason*)

Disconnect specified connection.

参数

- **con_idx** -- **[in]** Connection ID number to disconnect.
- **reason** -- **[in]** Reason to disconnect.

void **gap_manager_master_bond**(uint8_t *con_idx*, **struct** *pair_feature* **pair_feat*)

The master starts the bonding process.

参数

- **con_idx** -- **[in]** Connection ID number.
- **pair_feat** -- **[in]** Pairing parameter setting, This parameter can be a value of *pair_feature*.

void **gap_manager_master_encrypt**(uint8_t *con_idx*)

The master starts the secure connection process.

参数 **con_idx** -- **[in]** Connection ID number.

void **gap_manager_slave_security_req**(uint8_t *con_idx*, uint8_t *auth*)

Initiate an encryption request from the slave.

参数

- **con_idx** -- **[in]** Connection ID number.
- **auth** -- **[in]** SEC Auth param, This parameter can be a value of *gap_pair_auth*

void **gap_manager_slave_pair_response_send**(uint8_t *con_idx*, uint8_t *accept*, **struct** *pair_feature* **feat*)

The slave exchange pairs information.

参数

- **con_idx** -- **[in]** Connection ID number.
- **accept** -- **[in]** Whether to save master pairing information,value(0 or 1).
- **feat** -- **[in]** Pairing parameter setting, This parameter can be a value of *pair_feature*.

void **gap_manager_passkey_input**(uint8_t *con_idx*, **struct** *gap_pin_str* **passkey*)

Master and slave pair key input.

参数

- **con_idx** -- **[in]** Connection ID number.
- **passkey** -- **[in]** Connect the key, This parameter can be a value of *gap_pin_str*.

void **gap_manager_numeric_compare_set**(uint8_t *con_idx*, bool *equal*)

Verify that the key is the same in numerical comparison mode(Only for LE Secure Connections).

参数

- **con_idx** -- **[in]** Connection ID number.
- **equal** -- **[in]** Numeric comparison results.

void **gap_manager_sc_oob_set**(uint8_t *con_idx*, **struct** *gap_sc_oob* **sc_oob*)

Set the security oob of the specified connection.

参数

- **con_idx** -- **[in]** Connection ID number.
- **sc_oob** -- **[in]** SEC OOB value, This parameter can be a value of *gap_sc_oob*.

void **gap_manager_tk_set**(uint8_t *con_idx*, uint8_t *key*[*BLE_KEY_LEN*])

Set the security oob of the specified connection, BLE_KEY_LEN The length of the paired temporary key.

参数

- **con_idx** -- **[in]** Connection ID number.
- **key** -- **[in]** Pairing Temporary Key value.

uint8_t **gap_manager_get_role**(uint8_t *con_idx*)

Gets the role of the specified connection.

参数 **con_idx** -- **[in]** Connection ID number.

返回 Role of the connection. Refer to *LS_BLE_ROLE*.

uint8_t **gap_manager_get_sec_lvl**(uint8_t *con_idx*)

Gets the security level of the specified connection.

参数 **con_idx** -- **[in]** Connection ID number.

返回 The security level of the specified connection.

void **gap_manager_get_peer_addr**(uint8_t *con_idx*, **struct** *ble_addr* **addr*)

Gets the peer device address.

参数

- **con_idx** -- **[in]** Connection ID number.
- **addr** -- **[out]** Pointer to address.

void **gap_manager_get_identity_addr**(uint8_t *peer_id*, **struct** *ble_addr* **addr*)

Gets peer identity device address.

参数

- **peer_id** -- **[in]** Pairing ID number.
- **addr** -- **[out]** Pointer to address.

void **gap_manager_update_conn_param**(uint8_t *con_idx*, **struct** *gap_update_conn_param* **p_param*)

Update parameter for specified connection.

参数

- **con_idx** -- **[in]** Connection ID number.
- **p_param** -- **[in]** Pointer to parameter to update.

void **gap_manager_set_pkt_size**(uint8_t *con_idx*, **struct** *gap_set_pkt_size* **p_param*)

Update packet size in air for specified connection.

参数

- **con_idx** -- **[in]** Connection ID number.
- **p_param** -- **[in]** Pointer to packet size parameter to set.

void **gap_manager_delete_bonding**(uint8_t *peer_id*)

Deletes the bound device information.

参数 **peer_id** -- **[in]** Pairing ID number.

uint8_t **gap_manager_get_bonding_peer_id**(uint8_t *link_id*)

Gets the pairing ID of the bound device.

参数 **link_id** -- **[in]** Connection ID number.

返回 The pairing ID of the bound device.

uint8_t **gap_manager_get_bonded_dev_num**(void)

Gets the number of bound devices.

返回 The number of bound devices.

void **gap_manager_get_peer_rssi**(uint8_t *link_id*)

Gets the RSSI value of the specified connected device.

参数 **link_id** -- **[in]** Connection ID number.

void **gatt_manager_init**(void (*evt_cb)) **enum** *gatt_evt_type*, **union** *gatt_evt_u**, uint8_t

Initialize GATT manager.

参数 **evt_cb** -- **[in]** Callback function for gatt service.

void **gatt_manager_svc_register**(uint16_t *start_hdl*, uint8_t *att_num*, **struct** *gatt_svc_env* **svc*)

Register service in GATT manager.

参数

- **start_hdl** -- **[in]** Start handle of the service to be registered.
- **att_num** -- **[in]** Number of attributes contained in the service.
- **svc** -- **[in]** Pointer of service to be registered.

void **gatt_manager_server_read_req_reply**(uint8_t *con_idx*, uint16_t *handle*, uint8_t *status*, uint8_t *data, uint16_t *length*)

Send reply to read request from GATT client.

参数

- **con_idx** -- **[in]** Connection index.
- **handle** -- **[in]** Handle of attribute to read.
- **status** -- **[in]** Status of read command execution in application.
- **data** -- **[in]** Pointer of data reply.
- **length** -- **[in]** Length of data to send in the units of bytes.

void **gatt_manager_server_send_indication**(uint8_t *con_idx*, uint16_t *handle*, uint8_t *data, uint16_t *length*, uint16_t **transaction_id*)

Send indication to client.

参数

- **con_idx** -- **[in]** Connection index.
- **handle** -- **[in]** Handle of attribute to send indication.
- **data** -- **[in]** Pointer of data reply.
- **length** -- **[in]** Length of data to send in the units of bytes.
- **transaction_id** -- **[in]** Id of transaction between stack with application.

void **gatt_manager_server_send_notification**(uint8_t *con_idx*, uint16_t *handle*, uint8_t *data, uint16_t *length*, uint16_t **transaction_id*)

Send notification to client.

参数

- **con_idx** -- **[in]** Connection index.

- **handle** -- **[in]** Handle of attribute to send notification.
- **data** -- **[in]** Pointer of data reply.
- **length** -- **[in]** Length of data to send in the units of bytes.
- **transaction_id** -- **[in]** Id of transaction between stack with application.

void **gatt_manager_client_indication_confirm**(uint8_t *con_idx*, uint16_t *handle*)

Send indication confirm to server.

参数

- **con_idx** -- **[in]** Connection index.
- **handle** -- **[in]** Handle of attribute to send confirm.

uint16_t **gatt_manager_get_svc_att_handle**(**struct** *gatt_svc_env* **svc*, uint8_t *att_idx*)

Get the handle of attribute in the specified service.

参数

- **svc** -- **[in]** Pointer of the service containing the attribute.
- **att_idx** -- **[in]** Attribute index.

void **gatt_manager_client_write_no_rsp**(uint8_t *con_idx*, uint16_t *handle*, uint8_t **data*, uint16_t *length*)

Send data to GATT server by writing command(write without response).

参数

- **con_idx** -- **[in]** Connection index.
- **handle** -- **[in]** Handle of attribute to write.
- **data** -- **[in]** Pointer of data to send.
- **length** -- **[in]** Length of data to send in the units of bytes.

void **gatt_manager_client_write_with_rsp**(uint8_t *con_idx*, uint16_t *handle*, uint8_t **data*, uint16_t *length*)

Send data to GATT server by writing request(write with response).

参数

- **con_idx** -- **[in]** Connection index.
- **handle** -- **[in]** Handle of attribute to write.
- **data** -- **[in]** Pointer of data to send.
- **length** -- **[in]** Length of data to send in the units of bytes.

void **gatt_manager_client_cccd_enable**(uint8_t *con_idx*, uint16_t *handle*, bool *notification_en*, bool *indication_en*)

Enable cccd(client characteristic configuration descriptor) on GATT server service.

参数

- **con_idx** -- **[in]** Connection index.
- **handle** -- **[in]** Handle of attribute of cccd.
- **notification_en** -- **[in]** 1 = enable notification, 0 = disable notification.
- **indication_en** -- **[in]** 1 = enable indication, 0 = disable indication.

void **gatt_manager_client_svc_discover_by_uuid**(uint8_t *con_idx*, uint8_t **uuid*, **enum** *uuid_length* *uuid_len*, uint16_t *start_hdl*, uint16_t *end_hdl*)

Discovery service by specified uuid.

参数

- **con_idx** -- **[in]** Connection index.
- **uuid** -- **[in]** Pointer of uuid to be discovered.
- **uuid_len** -- **[in]** Length of uuid in units of *uuid_length*.
- **start_hdl** -- **[in]** Start handle of the range to search.
- **end_hdl** -- **[in]** End handle of the range to search.

void **gatt_manager_client_char_discover_by_uuid**(uint8_t *con_idx*, uint8_t **uuid*, **enum** *uuid_length* *uuid_len*, uint16_t *start_hdl*, uint16_t *end_hdl*)

Discovery characteristic by specified uuid.

参数

- **con_idx** -- **[in]** Connection index.
- **uuid** -- **[in]** Pointer of uuid to be discovered.
- **uuid_len** -- **[in]** Length of uuid in units of *uuid_length*.
- **start_hdl** -- **[in]** Start handle of the range to search.
- **end_hdl** -- **[in]** End handle of the range to search.

void **gatt_manager_client_desc_char_discover**(uint8_t *con_idx*, uint16_t *start_hdl*, uint16_t *end_hdl*)

Discovery cccd.

参数

- **con_idx** -- **[in]** Connection index.
- **start_hdl** -- **[in]** Start handle of the range to search.
- **end_hdl** -- **[in]** End handle of the range to search.

void **gatt_manager_client_mtu_exch_send**(uint8_t *con_idx*)

Send MTU exchange to GATT server.

参数 **con_idx** -- **[in]** Connection index.

void **gatt_manager_client_read**(uint8_t *con_idx*, uint16_t *handle*)

Read attribute value by handle.

参数

- **con_idx** -- **[in]** Connection index.
- **handle** -- **[in]** Handle of attribute to read.

struct dev_addr

#include <*ls_ble.h*> Device address.

Public Members

uint8_t **addr**[**BLE_ADDR_LEN**]

Address array

struct ble_addr

#include <*ls_ble.h*> BLE address structure.

Public Members

struct *dev_addr* **addr**

Address value in form of *dev_addr*

uint8_t **type**

Address type

struct legacy_adv_prop

#include <*ls_ble.h*> Legacy adv properities.

Public Members

uint8_t **connectable**

Connectable property

uint8_t **scannable**

Scannable property

uint8_t **directed**

Directed property

uint8_t **high_duty_cycle**

High duty cycle property

struct legacy_adv_obj_param

#include <*ls_ble.h*> Legacy adv object parameters structure.

Public Members

struct *dev_addr* ***peer_addr**

Peer address. Only valid for directed adv

uint16_t **adv_intv_min**

Minimum adv interval, in units of 625us

uint16_t **adv_intv_max**

Maximum adv interval, in units of 625us

enum *gap_own_addr_type* **own_addr_type**

Own address type

enum *gap_peer_addr_type* **peer_addr_type**

Peer address type

uint8_t **filter_policy**

Adv filter policy

uint8_t **ch_map**

Adv channel map. bit0: channel 37 enabled. bit1: channel 38 enabled. bit2: channel 39 enabled.

enum *adv_disc_mode* **disc_mode**

Adv discovery mode

struct *legacy_adv_prop* **prop**

Legacy adv properities

struct ext_adv_obj_param

#include <ls_ble.h> Extended adv object parameters structure.

Public Members

struct *legacy_adv_obj_param* **legacy_adv_obj**

Shared legacy adv parameters

uint8_t **max_skip**

Maximum number of advertising events the controller can skip before sending the AUX_ADV_IND packets.

0 means that AUX_ADV_IND PDUs shall be sent prior each advertising events

uint8_t **phy**

Indicate on which PHY secondary advertising has to be performed. Refer to *phy_type*

uint8_t **adv_sid**

Adv set ID

struct start_scan_param

#include <ls_ble.h> Scanning parameters.

Public Members

uint16_t **scan_intv**

Scan intervals in units of 625us

uint16_t **scan_window**

Scan window in units of 625us

uint16_t **duration**

Scan duration in units of 10ms. 0 means the scan action will run continuously until app stop it

uint16_t **period**

Scan window in units of 625us

enum *scan_type* **type**

Scan type

uint8_t **active**

Active scan

uint8_t **filter_duplicates**

Duplicate packet filtering policy. Refer to *filter_dup_policy*

struct start_init_param

#include <ls_ble.h> Start initiating parameters.

Public Members

struct *dev_addr* ***peer_addr**

Peer device address

uint16_t **scan_intv**

Scan intervals in units of 625us

uint16_t **scan_window**

Scan window in units of 625us

uint16_t **conn_to**

Timeout for automatic connection establishment (in unit of 10ms). Cancel the procedure if not all indicated devices have been connected when the timeout occurs. 0 means there is no timeout

uint16_t **conn_intv_min**

Minimum value for the connection interval (in unit of 1.25ms). Shall be less than or equal to conn_intv_max value. Allowed range is 7.5ms to 4s

uint16_t **conn_intv_max**

Maximum value for the connection interval (in unit of 1.25ms). Shall be greater than or equal to conn_intv_min value. Allowed range is 7.5ms to 4s

uint16_t **conn_latency**

Slave latency. Number of events that can be missed by a connected slave device

uint16_t **supervision_to**

Link supervision timeout (in unit of 10ms). Allowed range is 100ms to 32s

uint8_t **peer_addr_type**

Address type for peer device. 0=public/1=private random

enum *init_type* **type**

Initiating type

struct profile_added_evt

#include <ls_ble.h> Profile added event.

Public Members

uint16_t **start_hdl**

Start handle of the profile

enum *prf_id* **id**

Profile ID

struct service_added_evt

#include <ls_ble.h> Service added event.

Public Members

uint16_t **start_hdl**

Start handle of the Service

uint8_t **status**

Status of the service add action

struct obj_created_evt

#include <ls_ble.h> Object created event.

Public Members

uint8_t **handle**

Handle of the created object

uint8_t **status**

Status of object create action

struct stopped_evt

#include <ls_ble.h> Object created event.

Public Members

uint8_t **handle**

Handle of the stopped event

uint8_t **reason**

Reason for stopped event

struct obj_deleted_evt

#include <ls_ble.h> Object deleted event.

Public Members

uint8_t **handle**

Handle of the deleted object

uint8_t **status**

Status of object delete action

struct adv_report_info

#include <ls_ble.h> Adv report information.

Public Members

uint8_t **evt_type**

Adv report type. Refer to *adv_report_type*

uint8_t **complete**

Report is complete

uint8_t **connectable**

Connectable advertising

uint8_t **scannable**

Scannable advertising

uint8_t **directed**

Directed advertising

struct adv_report_evt

#include <ls_ble.h> Adv report event.

Public Members

uint8_t *__data__

Adv data

__struct__ *dev_addr* *__adv_addr__

Address of the device send the adv

uint16_t __length__

Adv report length

uint8_t __adv_addr_type__

Adv address type

int8_t __rssi__

RSSI

__struct__ *adv_report_info* __info__

Adv report information

__union dev_evt_u__

#include <*ls_ble.h*> Device event union.

Public Members

__struct__ *profile_added_evt* __profile_added__

Profile added event

__struct__ *service_added_evt* __service_added__

Service added event

__struct__ *obj_created_evt* __obj_created__

Object created event

__struct__ *stopped_evt* __stopped__

Stopped event

__struct__ *obj_deleted_evt* __deleted__

Object deleted event

__struct__ *adv_report_evt* __adv_report__

Adv report event

__struct ble_stack_cfg__

#include <*ls_ble.h*> BLE Stack configuration.

Public Members

bool **private_addr**

Identity address type. 0: Public 1: Random

bool **controller_privacy**

Indicate if controller privacy is enabled

struct char_properties

#include <ls_ble.h> Characteristics properities.

Public Members

uint8_t **broadcast**

Broadcast of characteristic value in Server Characteristic Configuration Descriptor enable

uint8_t **rd_en**

Read request enable

uint8_t **wr_cmd**

Write command enable

uint8_t **wr_req**

Write request enable

uint8_t **ntf_en**

Notification enable

uint8_t **ind_en**

Indication enable

uint8_t **wr_signed**

Write signed enable

uint8_t **ext_prop**

Extended properities enable

struct char_permissions

#include <ls_ble.h> Characteristics permissions.

Public Members

uint8_t **rd_perm**

Read permission. Refer to *svc_att_perm*

uint8_t **wr_perm**

Write permission. Refer to *svc_att_perm*

uint8_t **ind_perm**

Indication permission. Refer to *svc_att_perm*

uint8_t **ntf_perm**

Notification permission. Refer to *svc_att_perm*

struct att_decl

#include <ls_ble.h> Attribute declaration.

Public Members

const uint8_t ***uuid**

UUID of the attribute

uint16_t **max_len**

Maximum length supported by the attribute in units of byte

uint16_t **eks**

1 means Encryption key Size must be 16 bytes

uint16_t **uuid_len**

Length of UUID. Refer to *uuid_length*

uint16_t **read_indication**

Trigger Read Indication. 0 means data in database, 1 means read request will be forwarded to application

struct *att_decl*::**[anonymous] s**

struct *char_permissions* **char_perm**

Characteristic permission

struct *char_properties* **char_prop**

Characteristic properties

struct svc_decl

#include <ls_ble.h> Service declaration.

Public Members

const uint8_t ***uuid**

UUID of the service

struct *att_decl* ***att**

Attributes contained in the services

uint8_t **nb_att**

Number of attributes contained in the services

uint8_t **sec_lvl**

Security level. Refer to *svc_att_perm*

uint8_t **uuid_len**

Length of UUID. Refer to *uuid_length*

uint8_t **secondary**

0 = Primary Service, 1 = Secondary Service

struct gap_conn_param

#include <ls_ble.h> Connection parameters structure.

Public Members

uint16_t **intv_min**

Connection interval minimum

uint16_t **intv_max**

Connection interval maximum

uint16_t **latency**

Latency

uint16_t **time_out**

Supervision timeout

struct gap_conn_param_req

#include <ls_ble.h> Connection parameters request.

Public Members

struct *gap_conn_param***const** ***conn_param**

Connection parameters received

bool ***accept**

True = accept, False = reject

struct gap_conn_param_updated

#include <*ls_ble.h*> Connection parameters updated indication.

Public Members

uint16_t **con_interval**

Connection interval value

uint16_t **con_latency**

Connection latency value

uint16_t **sup_to**

Supervision timeout

struct gap_sc_oob

#include <*ls_ble.h*> SEC OOB value.

Public Members

uint8_t **conf**[**BLE_KEY_LEN**]

Confirm Value

uint8_t **rand**[**BLE_KEY_LEN**]

Random Number

struct pair_feature

#include <*ls_ble.h*> Set security parameter.

Public Members

uint8_t **iocap**

Set the IO capability, This parameter can be a value of *gap_io_caps*

uint8_t **oob**

Indicate whether OOB is supported, This parameter can be a value of *gap_pair_oob*

uint8_t **auth**

Set the auth, This parameter can be a value of *gap_pair_auth*

uint8_t **key_size**

Indicate the supported maximum LTK size (range: 7-16), This parameter can be a value of *gap_io_caps*

uint8_t **ikey_dist**

Set the initial key distribution, This parameter can be a value of *gap_key_dist*

uint8_t **rkey_dist**

Set the response key distribution, This parameter can be a value of *gap_key_dist*

struct gap_connected

#include <ls_ble.h> Connected indication event structure.

Public Members

uint16_t **con_interval**

Connection interval

uint16_t **con_latency**

Latency

uint16_t **sup_to**

Supervision timeout

uint8_t **peer_id**

Peer ID

struct gap_disconnected

#include <ls_ble.h> Disconnected indication event structure.

Public Members

uint8_t **reason**

Reason for disconnection

struct gap_master_pair_req

#include <ls_ble.h> Set master security parameter.

Public Members

uint8_t **auth**

Set the auth, This parameter can be a value of *gap_pair_auth*

struct gap_slave_security_req

#include <ls_ble.h> Set slave security parameter.

Public Members

uint8_t **auth**

Set the auth, This parameter can be a value of *gap_pair_auth*

struct gap_pair_done

#include <ls_ble.h> Parameter of pairing completion.

Public Members

bool **succeed**

The value indicates a successful pairing,Successful pairing is ”true” and unsuccessful pairing is ”false”

uint8_t **auth**

Pairing level achieved, This parameter can be a value of *gap_pair_auth*

uint8_t **fail_reason**

The reasons for the failure of the pairing

union *gap_pair_done*::**[anonymous] u**

struct gap_encrypt_fail

#include <ls_ble.h> Failed to encrypt the parameter.

Public Members

uint8_t **reason**

The reason for encryption failure

struct gap_encrypt_done

#include <ls_ble.h> Encryption completed security parameters.

Public Members

uint8_t **auth**

Pairing level achieved, This parameter can be a value of *gap_pair_auth*

struct gap_pin_str

#include <ls_ble.h> Passkey structure.

Public Members

char **pin**[6]

6 decimal numbers as passkey

char **str_pad**

The byte behind pin used to store '\0'

struct gap_display_passkey

#include <*ls_ble.h*> SEC passkey entry value.

Public Members

struct *gap_pin_str* **passkey**

Passkey entry value (000000~999999),This parameter can be a value of *gap_pin_str*

struct gap_numeric_compare

#include <*ls_ble.h*> SEC number comparison value.

Public Members

struct *gap_pin_str* **number**

Number comparison value (000000~999999),This parameter can be a value of *gap_pin_str*

struct gap_dev_info_dev_name

#include <*ls_ble.h*> Get device name.

Public Members

uint16_t **length**

Length of device name

uint8_t ***value**

Pointer to device name

struct gap_dev_info_appearance

#include <*ls_ble.h*> Get appearance.

Public Members

uint16_t **appearance**

Device appearance icon

struct gap_dev_info_slave_pref_param

#include <ls_ble.h> Get slave preferred parameters.

Public Members

uint16_t **con_intv_min**

Minimum connection interval

uint16_t **con_intv_max**

Maximum connection interval

uint16_t **slave_latency**

Slave latency

uint16_t **conn_timeout**

Supervision timeout

struct gap_dev_info_peer_rssi

#include <ls_ble.h> The RSSI value of the current connection.

Public Members

int8_t **rssi**

The RSSI value of the current connection(master or slave)

union gap_evt_u

#include <ls_ble.h> GAP event union definition.

Public Members

struct *gap_connected* **connected**

Connected event

struct *gap_disconnected* **disconnected**

Disconnected event

struct *gap_conn_param_req* **conn_param_req**

Connection parameter request event

struct *gap_conn_param_updated* **conn_param_updated**

Connection parameter updated event

struct *gap_master_pair_req* **master_pair_req**

Master pair request event

struct *gap_slave_security_req* **slave_security_req**

Slave security request event

struct *gap_pair_done* **pair_done**

Pair done event

struct *gap_encrypt_fail* **encrypt_fail**

Encryption fail event

struct *gap_encrypt_done* **encrypt_done**

Encryption done event

struct *gap_display_passkey* **display_passkey**

Display passkey event

struct *gap_numeric_compare* **numeric_compare**

Numeric comparison event

struct *gap_dev_info_dev_name* **get_dev_name**

Get device name

struct *gap_dev_info_appearance* **get_appearance**

Get Get appearance

struct *gap_dev_info_slave_pref_param* **slv_pref_param**

Get slave preferred parameters

struct *gap_dev_info_peer_rssi* **peer_rssi**

Get RSSI value of the current connection

struct gap_update_conn_param

#include <ls_ble.h> Connection parameter update.

Public Members

uint16_t **intv_min**

Mininum connection interval

uint16_t **intv_max**

Maximum connection interval

uint16_t **latency**

Latency

uint16_t **sup_timeout**

Supervision timeout

uint16_t **ce_len_min**

Mininum connection event length

uint16_t **ce_len_max**

Maximum connection event length

struct gap_set_pkt_size

#include <ls_ble.h> Set packet size in air.

Public Members

uint16_t **pkt_size**

Packet size in bytes

struct gatt_svc_env

#include <ls_ble.h> GATT service environment.

Public Members

void ***hdr**

Pointer to next *gatt_svc_env*

uint16_t **start_hdl**

Start handle of the service

uint8_t **att_num**

Attributes number in the service

struct gatt_server_read_req

#include <ls_ble.h> GATT read request.

Public Members

struct *gatt_svc_env***const** ***svc**

Pointer to serice containing the attribute to read

uint8_t **att_idx**

Attribute index

struct gatt_server_write_req

#include <ls_ble.h> GATT write request.

Public Members

struct *gatt_svc_env***const** ***svc**

Pointer to serice containing the attribute to read

uint8_t**const** ***value**

Pointer to value to write

uint8_t ***return_status**

Return status

uint16_t **offset**

Offset at which the data has to be written

uint16_t **length**

Length of data to write

uint8_t **att_idx**

Attribute index

struct gatt_server_notify_indicate_done

#include <*ls_ble.h*> Send notify/indicate done on GATT server.

Public Members

uint16_t **transaction_id**

Index of transaction

uint8_t **status**

Status of notifiy/indicate done

struct gatt_client_recv_notify_indicate

#include <*ls_ble.h*> Received notify/indicate on GATT client.

Public Members

uint16_t **handle**

Handle of notification/indication

uint16_t **length**

Length of notification/indication

uint8_t**const** ***value**

Pointer to value in notification/indication

struct gatt_mtu_changed_indicate

#include <*ls_ble.h*> MTU exchange indicate.

Public Members

uint16_t **mtu**

MTU received

struct gatt_handle_range

#include <ls_ble.h> Range of GATT handles.

Public Members

uint16_t **begin_handle**

Start handle

uint16_t **end_handle**

End handle

struct gatt_client_svc_disc_indicate

#include <ls_ble.h> Service discovery indicate for GATT client.

Public Members

const uint8_t ***uuid**

UUID of the service

struct *gatt_handle_range* **handle_range**

Handle range of the service

enum *uuid_length* **uuid_len**

Length of the service UUID

struct gatt_client_svc_disc_include_indicate

#include <ls_ble.h> Included service discovery indicate for GATT client.

Public Members

const uint8_t ***uuid**

UUID of the service

struct *gatt_handle_range* **handle_range**

Handle range of the service

uint16_t **attr_handle**

Attribute handle of included service

enum *uuid_length* **uuid_len**

Length of the service UUID

struct gatt_client_disc_char_indicate

#include <ls_ble.h> Characteristic discovery indicate for GATT client.

Public Members

const uint8_t ***uuid**

UUID of the characteristic

uint16_t **attr_handle**

Attribute handle of characteristic

uint16_t **pointer_handle**

Pointer attribute handle to UUID

uint8_t **properties**

Properities of the characteristic

enum *uuid_length* **uuid_len**

Length of the service UUID

struct gatt_client_disc_char_desc_indicate

#include <ls_ble.h> Characteristic descriptro discovery indicate for GATT client.

Public Members

const uint8_t ***uuid**

UUID of the descriptor

uint16_t **attr_handle**

Attribute handle of descriptor

enum *uuid_length* **uuid_len**

Length of the service UUID

struct gatt_read_indicate

#include <ls_ble.h> Read indication.

Public Members

uint8_t**const** ***value**

Pointer to the value read

uint16_t **handle**

Attribute handle

uint16_t **offset**

Offset at which the data has to be written

uint16_t **length**

Length of value

struct gatt_write_rsp

#include <ls_ble.h> Response for write request.

Public Members

uint16_t **transaction_id**

Index of transaction

uint8_t **status**

Status of write

struct gatt_write_no_rsp

#include <ls_ble.h> Response for write command.

Public Members

uint16_t **transaction_id**

Index of transaction

uint8_t **status**

Status of write

union gatt_evt_u

#include <ls_ble.h> Union definition for GATT events.

Public Members

struct *gatt_server_read_req* **server_read_req**

GATT server read request.

struct *gatt_server_write_req* **server_write_req**

GATT server write request.

struct *gatt_server_notify_indicate_done* **server_notify_indicate_done**

GATT server send notify/indicate done.

struct *gatt_client_recv_notify_indicate* **client_recv_notify_indicate**

GATT client receive notify/indicate.

struct *gatt_mtu_changed_indicate* **mtu_changed_ind**

MTU exchange indication.

struct *gatt_client_svc_disc_indicate* **client_svc_disc_indicate**

GATT client service discovery indicate.

struct *gatt_client_svc_disc_include_indicate* **client_svc_disc_include_indicate**

GATT client included service discovery indicate.

struct *gatt_client_disc_char_indicate* **client_disc_char_indicate**

GATT client characteristic discovery indicate.

struct *gatt_client_disc_char_desc_indicate* **client_disc_char_desc_indicate**

GATT client characteristic descriptor discovery indicate.

struct *gatt_read_indicate* **client_read_indicate**

GATT client read indicate.

struct *gatt_write_rsp* **client_write_rsp**

GATT client write request response.

struct *gatt_write_no_rsp* **client_write_no_rsp**

GATT client write command response.

3.1.2 PRF_HID API

Defines

HID_NB_ADD_MAX

This macro definition indicates that the maximum number of HID service instances is 2.

HID_NB_REPORT_MAX

This macro definition indicates that the maximal number of Report Characteristics that can be present in a HID Service is 5.

Enums

enum hid_svc_feature

HID device service features.

Values:

enumerator HID_KEYBOARD

The HID device is operating as a keyboard

enumerator HID_MOUSE

The HID device is operating as a mouse

enumerator HID_PROTO_MODE

The HID Device supports the Boot Protocol Mode

enumerator HID_EXT_REF

The Report Map Characteristic value maps information to an external service characteristic

enumerator HID_BOOT_KB_WR

The Boot Keyboard Input Report Characteristic value is writable

enumerator HID_BOOT_MOUSE_WR

The Boot Mouse Input Report Characteristic value is writable

enumerator HID_MASK

enumerator HID_REPORT_NTF_EN

Report Notification Enabled

enum hid_report_cfg

Report characteristic parameter configuration.

Values:

enumerator HID_REPORT_IN

The Report is an Input Report

enumerator HID_REPORT_OUT

The Report is an Output Report

enumerator HID_REPORT_FEAT

The Report is a Feature Report

enumerator HID_REPORT_WR

The Report Characteristic value is writable. Taken in account only if the Report is an Input Report

enum hid_info_flag

Representation of the flags parameter in the HID information.

Values:

enumerator HID_WKUP_FOR_REMOTE

Inform if the HID Device is capable of providing wake-up signal to a HID host

enumerator HID_NORM_CONN

Inform if the HID Device is normally connectable

enum hid_evt_type

Type of operation HID events.

Values:

enumerator HID_REPORT_READ

Read report value configuration

enumerator HID_NTF_CFG

The report notifies configuration values

enumerator HID_NTF_DONE

Report Notification done

enumerator HID_REPORT_WRITE

Modify/Set report value

enum app_hogpd_report_type

Type of reports enumeration.

Values:

enumerator APP_HOGPD_REPORT

The Report characteristic is used to exchange data between a HID Device and a HID Host

enumerator APP_HOGPD_REPORT_MAP

The Report Map characteristic

enumerator APP_HOGPD_BOOT_KEYBOARD_INPUT_REPORT

Boot Keyboard Input Report

enumerator APP_HOGPD_BOOT_KEYBOARD_OUTPUT_REPORT

Boot Keyboard Output Report

enumerator APP_HOGPD_BOOT_MOUSE_INPUT_REPORT

Boot Mouse Input Report

Functions

void **prf_hid_server_callback_init**(void (**evt_cb*)) **enum** *hid_evt_type*, **union** *hid_evt_u**, uint8_t

Initializes the HID events that reports a request to the counterpart device.

参数 **evt_cb** -- **[in]** Callback function for HID events

void **dev_manager_prf_hid_server_add**(uint8_t *sec_lvl*, **struct** *hid_db_cfg* **cfg*, uint16_t *len*)

Add the HID service to the database of the local device.

参数

- **sec_lvl** -- **[in]** Security level eg:default NO_SEC
- **cfg** -- **[in]** Configure structure variables for HID service information *hid_db_cfg*
- **len** -- **[in]** Length of *hid_db_cfg*

void **app_hid_send_keyboard_report**(uint8_t *report_idx*, uint8_t **report_data*, uint8_t *len*, uint8_t *conidx*)

A function interface for the HID device to send data.

参数

- **report_idx** -- **[in]** HID report instance
- **report_data** -- **[in]** Point to the data address to send
- **len** -- **[in]** The length of the data to be sent
- **conidx** -- **[in]** Connect instance

void **hid_ntf_cfg_init**(uint16_t *ntf_cfg*, uint8_t *con_idx*, uint8_t *peer_id*)

Report notification configuration.

参数

- **ntf_cfg** -- **[in]** The report notifies configuration values
- **con_idx** -- **[in]** Connect instance
- **peer_id** -- **[in]** Peer device instance

struct hid_info

#include <prf_hid.h> HID Information structure.

Public Members

uint16_t **bcdHID**

HID Class Specification release number in binarycoded decimal (for example, 1.50 is 0x150)

uint8_t **bCountryCode**

Hardware target country

uint8_t **flags**

Flags *hid_info_flag*

struct hids_cfg

#include <prf_hid.h> HID configuration structure.

Public Members

uint8_t **svc_features**

Features supported in the HID Service *hid_svc_feature*

uint8_t **report_nb**

Value of the HID Information Characteristic

uint8_t **report_cfg**[**HID_NB_REPORT_MAX**]

Features supported by each of the Report Characteristics in the HID Service *hid_report_cfg*

uint8_t **report_id**[**HID_NB_REPORT_MAX**]

Report id number for a given report type, The Report ID is defined in the Report Map

struct *hid_info* **info**

Value of the HID Information Characteristic *hid_info*

struct hid_db_cfg

#include <prf_hid.h> HID database configuration structure.

Public Members

uint8_t **hids_nb**

Number of HID service

struct *hids_cfg* **cfg**[**HID_NB_ADD_MAX**]

HID configuration items *hids_cfg*

struct hid_read_report_req_evt

#include <prf_hid.h> A structure for reporting information about read events.

Public Members

uint16_t **length**

The length of the data to be sent

uint8_t ***value**

Point to the data address to send

uint8_t **hid_idx**

HIDS Instance

uint8_t **type**

type of report *app_hogpd_report_type*

uint8_t **idx**

Report Instance - 0 for boot reports and report map

struct hid_write_report_req_evt

#include <prf_hid.h> A structure for reporting information about write events.

Public Members

uint16_t **length**

The length of the data to be sent

uint8_t ***value**

Point to the data address to send

struct hid_ntf_cfg_evt

#include <prf_hid.h> Structure for the Report configuration events.

Public Members

uint16_t **value**

Notification Configuration Value

union hid_evt_u

#include <prf_hid.h> Information data Union used to read or write information events.

Public Members

struct *hid_read_report_req_evt* **read_report_req**

hid_read_report_req_evt

struct *hid_ntf_cfg_evt* **ntf_cfg**

hid_ntf_cfg_evt

struct *hid_write_report_req_evt* **write_report_req**

hid_write_report_req_evt

3.1.3 LSMESH API

Defines

__LS_MESH_EMPTY

flexible array length

Enums

enum ls_mesh_evt_type

Linkedsemi mesh event type definition.

Values:

enumerator LS_MESH_RX_MSG_EVT

rx message event

enumerator LS_MESH_FINISH_EVT

finished event

Functions

void **prf_ls_mesh_callback_init**(void (*evt_cb)) **enum** *ls_mesh_evt_type*, **union** *ls_mesh_evt_u**

Initialization for rx messages API of linkedsemi mesh.

参数 **evt_cb** -- **[in]** Callback function to handle all the ls mesh messages.

void **ls_mesh_init**(void)

Initialization of linkedsemi mesh.

void **dev_manager_prf_ls_mesh_add**(uint8_t *sec_lvl*, void **param*)

Add profile of linkedsemi mesh to the database.

参数

- **sec_lvl** -- **[in]** no security
- **param** -- **[in]** default NULL.

void **ls_mesh_start**(void)

Active task of linkedsemi mesh.

void **ls_mesh_set_beacon_value_ind**(**const** uint8_t **value*, uint8_t *len*)

API for Tx message of linkedsemi mesh.

参数

- **value** -- **[in]** application data
- **len** -- **[in]** length of application data

struct ls_mesh_rx_msg_evt

#include <ls_mesh.h> Linkedsemi mesh rx message structure.

Public Members

uint8_t **msg_len**

rx message length include adv_type/uuid/handle/version and vlaue

uint8_t **adv_type**

defined 0x16

uint16_t **uuid**

defined 0xFEE4

uint16_t **handle**

rx message handle

uint32_t **version**

Version dependent handle

uint8_t **value**[**__LS_MESH_EMPTY**]

rx message value

union ls_mesh_evt_u

#include <*ls_mesh.h*> Linkedsemi mesh event union.

Public Members

struct *ls_mesh_rx_msg_evt* **ls_mesh_send_msg**

Linkedsemi mesh rx message event

3.1.4 LSSIG_MESH_PROVEE API

Defines

__LSSIGMESH_EMPTY

Flexible array length

UUID_MESH_DEV_LEN

Device UUID length

MESH_AUTH_DATA_LEN

Authentication data length

MAX_MESH_MODEL_NB

Max number of model in a node

MAX_MESH_MODEL_MSG_BUFFER

The buffer of model message

UPADTE_GLP_STOP_TYPE

To be updated the status of stop type by tmall genie GLP function to application

UPADTE_GLP_STOP_TIMEOUT_TYPE

To be updated the status of timeout type by tmall genie GLP function to application

GENERIC_ONOFF_SERVER

SIG Model Index of generic onoff server 0x1000

GENERIC_ONOFF_CLIENT

SIG Model Index of generic onoff client 0x1001

GENERIC_LVL_SERVER

SIG Model Index of generic level server 0x1002

GENERIC_LVL_CLIENT

SIG Model Index of generic level client 0x1003

LIGHTNESS_SERVER

SIG Model Index of light lightness server 0x1300

LIGHTS_CTL_SERVER

SIG Model Index of light control server 0x1303

LIGHTS_HSL_SERVER

SIG Model Index of light HSL server 0x1307

VENDOR_TMALL_SERVER

Vendor Model Index of tmall genie server 0x01A80000

VENDOR_USER_SERVER

Vendor Model Index of linkedsemi server 0x093A0001

VENDOR_USER_CLIENT

Vendor Model Index of linkedsemi client 0x093A0002

GENERIC_ONOFF_GET

Opcdoe of generic onoff get 0x0182

GENERIC_ONOFF_SET

Opcdoe of generic onoff set 0x0282

GENERIC_ONOFF_SET_UNAK

Opcdoe of generic onoff set unacknowleged 0x0382

GENERIC_ONOFF_STATUS

Opcdoe of generic onoff status 0x0482

GENERIC_LVL_GET

Opcdoe of generic level get 0x0582

GENERIC_LVL_SET

Opcdoe of generic level set 0x0682

GENERIC_LVL_SET_UNAK

Opcdoe of generic level set unacknowleged 0x0782

GENERIC_LVL_STATUS

Opcdoe of generic level status 0x0882

LIGHT_LIGHTNESS_SET

Opcdoe of light lightness set 0x4c82

LIGHT_LIGHTNESS_SET_UNAK

Opcdoe of light lightness set unacknowleged 0x4d82

LIGHT_LIGHTNESS_STATUS

Opcdoe of light lightness status 0x4e82

LIGHT_HSL_SET

Opcdoe of light hsl set 0x7682

LIGHT_HSL_SET_UNACK

Opcdoe of light hsl set unacknowleged 0x7782

LIGHT_HSL_STATUS

Opcdoe of light hsl status 0x7882

LIGHT_CTL_SET

Opcdoe of light ctl set 0x5E82

LIGHT_CTL_SET_UNACK

Opcdoe of light ctl set unacknowleged 0x5F82

LIGHT_CTL_STATUS

Opcdoe of light ctl status 0x6082

APP_MESH_VENDOR_SET

Vendor opcdoe of tmall genie set 0x0001A8d1

APP_MESH_VENDOR_SET_UNAK

Vendor opcdoe of tmall genie set unacknowleged 0x0001A8d2

APP_MESH_VENDOR_STATUES

Vendor opcdoe of tmall genie status 0x0001A8d3

APP_MESH_VENDOR_INDICATION

Vendor opcdoe of tmall genie indication 0x0001A8d4

APP_MESH_VENDOR_CONFIRMATION

Vendor opcdoe of tmall genie confirmation 0x0001A8d5

APP_MESH_VENDOR_TRANSPARENT_MSG

Vendor opcdoe of tmall genie transparent message 0x0001A8cf

APP_LS_SIG_MESH_VENDOR_GET

Vendor opcdoe of linkedsemi set 0x00093AD0

APP_LS_SIG_MESH_VENDOR_SET

Vendor opcdoe of linkedsemi set 0x00093AD1

APP_LS_SIG_MESH_VENDOR_SET_UNAK

Vendor opcdoe of linkedsemi set unacknowleged 0x00093AD2

APP_LS_SIG_MESH_VENDOR_STATUS

Vendor opcdoe of linkedsemi status 0x00093AD3

APP_LS_SIG_MESH_VENDOR_INDICATION

Vendor opcdoe of linkedsemi indication 0x00093AD4

APP_LS_SIG_MESH_VENDOR_CONFIRMATION

Vendor opcdoe of linkedsemi confirmation 0x00093AD5

APP_LS_SIG_MESH_VENDOR_HEARTBEAT

Vendor opcdoe of linkedsemi heartbeat 0x00093AD6

APP_LS_SIG_MESH_VENDOR_SCENE_SET

Vendor opcdoe of linkedsemi scene setting 0x00093AD7

APP_LS_SIG_MESH_VENDOR_SCENE_ACK

Vendor opcdoe of linkedsemi scene setting ack 0x00093AD8

VENDOR_OPCODE_LEN

Vendor opcdoe length

VENDOR_OPCODE_MASK

Vendor opcdoe MASK

VENDOR_OPCODE_TYPE

Vendor opcdoe type

Typedefs

typedef uint8_t **SIGMESH_ModelHandle_TypeDef**

typedef uint8_t **SIGMESH_NodeInfo_TypeDef**

Enums

enum mesh_evt_type

SIG mesh event type.

s

Values:

enumerator MESH_ACTIVE_ENABLE

To be reported this event type after SIG mesh was initialized and activated

enumerator MESH_ACTIVE_DISABLE

To be reported this event type after SIG mesh was stopped

enumerator MESH_ACTIVE_REGISTER_MODEL

To be reported this event type after all requested models were registered, you need to save the returned local index

enumerator MESH_ACTIVE_MODEL_PUBLISH

To be reported this event type after client model was enable publish function by provisioner, you need to save the returned information of publish

enumerator MESH_ACTIVE_MODEL_GROUP_MEMBERS

To be reported this event type after all requested models were automatically bound AppKey, only for genie mesh

enumerator MESH_ACTIVE_MODEL_RSP_SENT

To be reported this event type after the protocol stack confirms that the message was successfully sent

enumerator MESH_ACTIVE_LPN_START

To be reported that this event type requests low power properties (such as timeout, interval, previous_address, RX window factor) after the low-power feature node was registered

enumerator MESH_ACTIVE_LPN_OFFER

When the low-power node establishes a friendly relationship, all nearby friend-supporting attributes are reported to the application layer and this event type is activated

enumerator MESH_ACTIVE_LPN_STATUS

To be reported that this event type requests to send a status message of LPN, such as Friendship status/address of friend node

enumerator MESH_ACTIVE_STORAGE_LOAD

To be reported this event type when each time the device was restarted and reported whether the device was a node

enumerator MESH_GET_PROV_INFO

To be reported this event type requesting device properties (such as uuid, urihash, oob) during provisioning

enumerator MESH_GET_PROV_AUTH_INFO

To be reported that this event type requests device authentication values during configuration

enumerator MESH_REPORT_ATTENTION_STATE

To be reported this event type after the attention state had been updated

enumerator MESH_REPOPT_PROV_RESULT

To be reported this event type whether the device would successfully become a node after it was provisioned

enumerator MESH_ACCEPT_MODEL_INFO

To be reported this event type when vendor model recevied messages

enumerator MESH_REPORT_TIMER_STATE

To be reported this event type when the power-on timer timed out to perform power-on count clearing

enumerator MESH_GENIE_PROV_COMP

To be reported this event type after all requested models were automatically bound AppKey, only for genie lower power

enumerator MESH_ADV_REPORT

To be reported this event type when the device scaned an adv message

enumerator MESH_STATE_UPD_IND

To be reported this event type when sig model recevied messages

enumerator MESH_ACTIVE_GLP_START

To be reported this event type when each time the node was restarted and enabled genie lower power

enumerator MESH_ACTIVE_GLP_STOP

To be reported this event type when the node was stopped genie lower power

enumerator MESH_ACTIVE_AUTO_PROV

To be reported this event type when automatic provisioning was enabled on the device

enumerator MESH_EVT_TYPE_MAX

enum mesh_feature

sig mesh feature type definition

Values:

enumerator EN_RELAY_NODE

Relay feature

enumerator EN_PROXY_NODE

Both the Gatt feature and proxy feature are enabled

enumerator EN_FRIEND_NODE

Friend feature

enumerator EN_LOW_POWER_NODE

Lower power feature

enumerator EN_MSG_API

The Message's API feature must be enabled

enumerator EN_PB_GATT

Both the Gatt feature and proxy feature are enabled

enumerator EN_DYN_BCN_INTV

Dynamic beacon interval feature

enumerator EN_PROVER

Provisioner feature

enum mesh_provisioned_state

Device state definition.

Values:

enumerator UNPROVISIONED_KO

The device was not a node

enumerator PROVISIONED_OK

The device was a node

enum mesh_provisioning_result

provisioning state of the device definition

Values:

enumerator MESH_PROV_STARTED

The device was provisioning

enumerator MESH_PROV_SUCCEED

The device provisioned and successly to be a node

enumerator MESH_PROV_FAILED

The device provisioned but fail to be a node

enum mesh_timer_state

mesh timer sate of user definition

Values:

enumerator MESH_TIMER_DOING

The timer was working

enumerator MESH_TIMER_DONE

The timer timed out

enum tmall_glp_stop_reason

The reason definition why genie lower power was stoped.

Values:

enumerator NO_STOPPING_GLP_REQ

Normal state for glp

enumerator APPLICATION_USER_STOPPING_GLP_REQ

User requests to turn off GLP

enumerator PROVISIONING_INVITE_SWITCH_GLP_REQ

During provisioning requests to turn off GLP

enumerator PROVISIONING_COMP_SWITCH_GLP_REQ

After provisioning requests to turn on GLP

enum tmall_glp_state

state definition of genie lower power

Values:

enumerator TMALL_GLP_STATE_IDLE

Idle state for glp

enumerator TMALL_GLP_STATE_ACTIVE

Active state for glp

enum mesh_state_idx

State identifier values of sig model.

Values:

enumerator MESH_STATE_GEN_ONOFF
Generic OnOff state

enumerator MESH_STATE_GEN_LVL
Generic Level state

enumerator MESH_STATE_GEN_DTT
Generic Default Transition Time state

enumerator MESH_STATE_GEN_POWER_ACTUAL
Generic Power Actual state

enumerator MESH_STATE_GEN_POWER_LAST
Generic Power Last state

enumerator MESH_STATE_GEN_POWER_DFLT
Generic Power Default state

enumerator MESH_STATE_GEN_POWER_RANGE
Generic Power Range state

enumerator MESH_STATE_GEN_ONPOWERUP
Generic OnPowerUp state

enumerator MESH_STATE_LIGHT_LN
Light Lightness

enumerator MESH_STATE_LIGHT_LN_LIN
Light Lightness Linear

enumerator MESH_STATE_LIGHT_LN_LAST
Light Lightness Last

enumerator MESH_STATE_LIGHT_LN_DFLT
Light Lightness Default

enumerator MESH_STATE_LIGHT_LN_RANGE
Light Lightness Range

enumerator MESH_STATE_LIGHT_LN_RANGE_MIN
Light Lightness Range Min

enumerator MESH_STATE_LIGHT_LN_RANGE_MAX
Light Lightness Range Max

enumerator MESH_STATE_LIGHT_CTL_LN
Light CTL Lightness

enumerator MESH_STATE_LIGHT_CTL_TEMP
Light CTL Temperature

enumerator MESH_STATE_LIGHT_CTL_DELTA_UV
Light CTL Delta UV

enumerator MESH_STATE_LIGHT_CTL_TEMP_DFLT
Light CTL Temperature Default

enumerator MESH_STATE_LIGHT_CTL_TEMP_RANGE
Light CTL Temperature Range

enumerator MESH_STATE_LIGHT_CTL_DELTA_UV_DFLT
Light CTL Delta UV Default

enumerator MESH_STATE_LIGHT_HSL_LN
Light HSL Lightness

enumerator MESH_STATE_LIGHT_HSL_HUE
Light HSL Hue

enumerator MESH_STATE_LIGHT_HSL_SAT
Light HSL Saturation

enumerator MESH_STATE_LIGHT_HSL_TGT
Light HSL Target

enumerator MESH_STATE_LIGHT_HSL_DFLT
Light HSL Default (Lightness + Hue + Saturation)

enumerator MESH_STATE_LIGHT_HSL_DFLT_LN
Light HSL Lightness Default

enumerator MESH_STATE_LIGHT_HSL_DFLT_HUE
Light HSL Hue Default

enumerator MESH_STATE_LIGHT_HSL_DFLT_SAT
Light HSL Saturation Default

enumerator MESH_STATE_LIGHT_HSL_RANGE_HUE
Light HSL Hue Range

enumerator MESH_STATE_LIGHT_HSL_RANGE_SAT
Light HSL Saturation Range

enumerator MESH_STATE_LIGHT_XYL_LN
Light xyL Lightness

enumerator MESH_STATE_LIGHT_XYL_XY
Light xyL x and y

enumerator MESH_STATE_LIGHT_XYL_LN_TGT
Light xyL Lightness Target

enumerator MESH_STATE_LIGHT_XYL_XY_TGT

Light xyL x and y Target

enumerator MESH_STATE_LIGHT_XYL_LN_DFLT

Light xyL Lightness Default

enumerator MESH_STATE_LIGHT_XYL_XY_DFLT

Light xyL x and y Default

enumerator MESH_STATE_LIGHT_XYL_XY_RANGE

Light xyL x and y Range

enum lpn_rx_window_factor

The Receive window factor values are used in the friend offer delay calculation.

Values:

enumerator LPN_RX_WINDOW_FACTOR_1

ReceiveWindowFactor 1

enumerator LPN_RX_WINDOW_FACTOR_1_5

ReceiveWindowFactor 1.5

enumerator LPN_RX_WINDOW_FACTOR_2

ReceiveWindowFactor 2

enumerator LPN_RX_WINDOW_FACTOR_2_5

ReceiveWindowFactor 2.5

enum lpn_rssi_factor

The contribution of the RSSI measured by the Friend node used in Friend Offer Delay calculations.

Values:

enumerator LPN_RSSI_FACTOR_1

RSSIFactor 1

enumerator LPN_RSSI_FACTOR_1_5

RSSIFactor 1.5

enumerator LPN_RSSI_FACTOR_2

RSSIFactor 2

enumerator LPN_RSSI_FACTOR_2_5

RSSIFactor 2.5

enum friend_node_min_queue_size_log

The minimum number of messages that the Friend node can store in its Friend Queue, the min queue size log field is defiendd as log2(Nx).

Values:

enumerator FRIEND_NODE_MIN_QUEUE_SIZE_LOG_PROHIB
Prohibited

enumerator FRIEND_NODE_MIN_QUEUE_SIZE_LOG_N2
At least 1 message is stored in the friend queue

enumerator FRIEND_NODE_MIN_QUEUE_SIZE_LOG_N4
At least 2 messages are stored in the friend queue

enumerator FRIEND_NODE_MIN_QUEUE_SIZE_LOG_N8
At least 3 messages are stored in the friend queue

enumerator FRIEND_NODE_MIN_QUEUE_SIZE_LOG_N16
At least 4 messages are stored in the friend queue

enumerator FRIEND_NODE_MIN_QUEUE_SIZE_LOG_N32
At least 5 messages are stored in the friend queue

enumerator FRIEND_NODE_MIN_QUEUE_SIZE_LOG_N64
At least 6 messages are stored in the friend queue

enumerator FRIEND_NODE_MIN_QUEUE_SIZE_LOG_N128
At least 7 messages are stored in the friend queue

Functions

struct mesh_prov_info __attribute__ ((packed))

void **prf_ls_sig_mesh_callback_init**(void (**evt_cb*)) **enum** *mesh_evt_type*, **union** *ls_sig_mesh_evt_u**
Initialization Event Callback of Sig mesh.

参数 **evt_cb** -- Callback function to handle all sig mesh messages.

void **dev_manager_prf_ls_sig_mesh_add**(uint8_t *sec_lvl*, **struct** *ls_sig_mesh_cfg *cfg*)
Add Sig mesh profile to dev manager.

参数

- **sec_lvl** -- No security
- **cfg** -- Node Suported Feature, struct *ls_sig_mesh_cfg*

void **ls_sig_mesh_init**(**struct** *mesh_model_info *param*)
Initialization of Sig mesh.

参数 **param** -- Register Model informatin ,struct *mesh_model_info*

void **ls_sig_mesh_platform_reset**(void)
Node reset Operation.

void **set_prov_param**(**struct** *mesh_prov_info* *param)

Set the prov parameter.

参数 **param** -- struct *mesh_prov_info*

void **set_prov_auth_info**(**struct** *mesh_prov_auth_info* *param)

Set the prov auth parameter.

参数 **param** -- struct *mesh_prov_auth_info*

void **model_subscribe**(uint8_t**const** *ModelHandle*, uint16_t**const** *Addr*)

Subscribe group address for model.

参数

- **ModelHandle** -- model local id
- **Addr** -- group address

void **model_get_subscribe_list**(uint8_t *ModelHandle*, uint8_t **add_list*, bool *uuid_flag*)

Get all subscript address from subscribe_list.

参数

- **ModelHandle** -- model local id
- **add_list** -- application list
- **uuid_flag** -- get uuid flag

uint16_t **model_get_subscribe_listSize**(uint8_t *ModelHandle*)

Get size from subscribe_list.

参数 **ModelHandle** -- model local id

返回 uint16_t listSize

uint16_t **ls_sig_mesh_get_primary_address**(void)

Get primary address to Node.

返回 uint16_t

void **model_send_info_handler**(**struct** *model_send_info* *param)

User send message to other nodes by Vendor Client Model.

参数 **param** --

void **mesh_model_client_set_state_handler**(**struct** *model_cli_set_state_info* *param)

Set state to other nodes by Sig Client Model.

参数 **param** --

void **mesh_standard_model_publish_message_handler**(**struct** *model_cli_trans_info* *param)

Publish message to other nodes by by Sig Client Model.

参数 **param** --

void **mesh_vendor_model_publish_message_handler**(**struct** *vendor_model_publish_message* **msg*)

Publish message to other nodes by by Vendor Client Model,but Cann't set dest address.

参数 **msg** --

void **TIMER_Set**(uint8_t *TimerID*, uint32_t *DelayMS*)

Set mesh timer.

参数

- **TimerID** --
- **DelayMS** --

void **TIMER_Cancel**(uint8_t *TimerID*)

Delete mesh timer.

参数 **TimerID** --

void **SIGMESH_UnbindAll**(void)

Clear all provisioned information, a node beacoms a device.

void **stop_tx_unprov_beacon**(void)

Device stop unprov beacon.

void **start_tx_unprov_beacon**(**struct** *bcn_start_unprov_param* **param*)

Device start unprov beacon.

参数 **param** --

void **ls_sig_mesh_con_set_scan_rsp_data**(uint8_t **scan_rsp_data*, uint8_t **scan_rsp_data_len*)

Set_scan_rsp_data for genie mesh gatt.

参数

- **scan_rsp_data** --
- **scan_rsp_data_len** --

void **start_ls_sig_mesh_gatt**(void)

Enable Gatt mesh function and Start connectable adv.

void **stop_ls_sig_mesh_gatt**(void)

Disable Gatt mesh function and Stop connectable adv.

void **ls_sig_mesh_proxy_adv_ctl**(uint8_t *enable*)

Enable/Disable Proxy adv.

参数 **enable** --

void **start_glp_handler**(**struct** *start_glp_info* **param*)

Enable lower power for genie mesh.

参数 **param** -- struct *start_glp_info*

void **stop_glp_handler**(void)

Disable lower power for genie mesh.

void **start_lpn_handler**(**struct** *start_lpn_info* *param)

Enable lower power for lpn.

参数 param --

void **stop_lpn_handler**(void)

Disable lower power for lpn.

void **ls_sig_mesh_disable**(void)

Disable Sig mesh function.

void **ls_sig_mesh_enable**(void)

Enable Sig mesh function.

void **lnp_select_friend_handler**(uint16_t *friend_addr*)

Select friend node for LPN.

参数 friend_addr --

void **ls_sig_mesh_auto_prov_handler**(**struct** *mesh_auto_prov_info***const** *param, bool**const** *auto_prov_mesh_flag*)

Enable Auto Provisioning process.

参数

- **param** -- struct *mesh_auto_prov_info*
- **auto_prov_mesh_flag** -- Enable/Disable Flag

void **report_provisioner_unicast_address_ind**(uint16_t *unicast_address*)

Report unicast_address of provisioner.

参数 unicast_address --

void **ls_sig_mesh_set_proxy_con_interval**(uint16_t **interval_ms*)

Set Connectable Adv interval for proxy node.

参数 interval_ms --

void **ls_sig_mesh_set_pb_gatt_con_interval**(uint16_t **interval_slot*)

Set Connectable Adv interval for device.

参数 interval_slot --

Variables

uint8_t **DevUuid**[**UUID_MESH_DEV_LEN**]

device uuid

uuid of device

uint32_t **UriHash**

Supported URI-Hash types

URI-Hash information

uint16_t **OobInfo**

OOB information

uint8_t **PubKeyOob**

Supported public key types

uint8_t **StaticOob**

Supported Static OOB types

uint8_t **OutOobSize**

Maximum size in octets of Output OOB supported

uint8_t **InOobSize**

Maximum size in octets of Input OOB supported

uint16_t **OutOobAction**

Supported Output OOB Actions

uint16_t **InOobAction**

Supported Input OOB Actions

uint8_t **Info**

Bit field providing additional information

struct mesh_prov_auth_info __attribute__

uint8_t **ModelHandle**

Client Model local index

Model local index

uint8_t **TxHandle**

Message transmiter number

uint32_t **MsgOpcode**

Message opcode

uint16_t **MsgLength**

Message length

uint8_t **msg**[**MAX_MESH_MODEL_MSG_BUFFER**]
: Message

uint8_t **RxDelyMs**
: rx delay at least 10ms

uint16_t **SleepIntvlMs**
: rx delay at least 10ms

uint8_t **rx_window_factor**
: enum lpn_rx_window_factor

uint8_t **min_queue_size_log**
: enum enum friend_node_min_queue_size_log

uint8_t **rx_delay_ms**
: rx delay at least 10ms

uint16_t **previous_addr**

uint32_t **poll_timeout_100ms**
: poll timeout at least 1s

uint32_t **poll_intv_ms**
: poll interval

uint8_t **app_key_lid**
: App Key local index

int8_t **rssi**
: rssi of message

uint16_t **source_addr**
: source_addr of message

uint8_t **not_relayed**
: True if message have been received by an immediate peer;False, it can have been relayed.

uint32_t **opcode**
: Operation code

uint16_t **rx_info_len**
: message length

uint8_t **info**[**__LSSIGMESH_EMPTY**]
: message

uint8_t **elmt_idx**
: Element index

uint16_t **state_id**
: State identifier, num mesh_state_idx

uint32_t **state**

New state value or targeted state value depending on transition time

Provisioning state ,enum mesh_provisioning_result

uint32_t **trans_time_ms**

Transition time in milliseconds

uint16_t **status**

Provisioning failed reason

uint8_t **element_id**

Element index

uint8_t **model_lid**

Model local index

uint8_t **vendor_model_role**

Vendor Model Server or Client

uint16_t **sig_model_cfg_idx**

Config Index for Sig Model

uint16_t **vendor_model_cfg_idx**

Config Index for Vendor Model

uint32_t **model_id**

Model Index

uint32_t **period_ms**

Publish period in milliseconds

uint16_t **addr**

Publication address

bool **publish_flag**

Enable Publish function

bool **subs_flag**

Enable Subscript function

uint16_t **unicast_addr**

unicast address

uint8_t **model_nb**

Total registered Model

uint16_t **group_addr**

Group address for publish and subscription

struct *mesh_auto_prov_model_info* **model_info**[**MAX_MESH_MODEL_NB**]

Maximum models in a node struct *mesh_auto_prov_model_info*

uint8_t **app_key**[16]

uint8_t **net_key**[16]

bool **UriHash_Present**

Support or not

struct report_dev_provisioned_state_info

#include <ls_sig_mesh.h> Device state information struct.

Public Members

uint8_t **proved_state**

enum mesh_provisioned_state

uint8_t **proving_success_state**

enum mesh_provisioning_result

struct mesh_prov_info

#include <ls_sig_mesh.h> Authentication values struct.

Public Members

uint8_t **DevUuid**[**UUID_MESH_DEV_LEN**]

device uuid

uint32_t **UriHash**

Supported URI-Hash types

uint16_t **OobInfo**

OOB information

uint8_t **PubKeyOob**

Supported public key types

uint8_t **StaticOob**

Supported Static OOB types

uint8_t **OutOobSize**

Maximum size in octets of Output OOB supported

uint8_t **InOobSize**

Maximum size in octets of Input OOB supported

uint16_t **OutOobAction**

Supported Output OOB Actions

uint16_t **InOobAction**

Supported Input OOB Actions

uint8_t **Info**

Bit field providing additional information

struct mesh_prov_auth_info

#include <ls_sig_mesh.h> Authentication information struct.

Public Members

uint8_t **Adopt**

Accept pairing request, 0 reject

uint8_t **AuthSize**

authentication size

uint8_t **AuthBuffer**[**MESH_AUTH_DATA_LEN**]

authentication value

struct report_mesh_attention_info

#include <ls_sig_mesh.h> Attention state.

Public Members

uint8_t **state**

struct update_state_info

#include <ls_sig_mesh.h> Report updated state struct.

Public Members

uint8_t **upd_type**

The state type of the update

uint8_t **len**

data lenght

uint8_t **data**[**__LSSIGMESH_EMPTY**]

data

struct model_send_info

#include <ls_sig_mesh.h> Send messages struct.

Public Members

uint8_t **ModelHandle**
: Client Model local index

uint8_t **app_key_lid**
: App key local index

uint16_t **dest_addr**
: Destination address

uint32_t **opcode**
: Message opcode

uint16_t **len**
: Message length

uint8_t **info**[**MAX_MESH_MODEL_MSG_BUFFER**]
: Message

struct vendor_model_publish_message

#include <ls_sig_mesh.h> Publish message struct -for Client Vendor model.

Public Members

uint8_t **ModelHandle**
: Client Model local index

uint8_t **TxHandle**
: Message transmiter number

uint32_t **MsgOpcode**
: Message opcode

uint16_t **MsgLength**
: Message length

uint8_t **msg**[**MAX_MESH_MODEL_MSG_BUFFER**]
: Message

struct model_cli_set_state_info

#include <ls_sig_mesh.h> Set state message structure - for Client SIG model.

Public Members

uint32_t **state_1**

state1

uint32_t **state_2**

state1

uint16_t **dest_addr**

Destination address

uint16_t **set_info**

Set information

uint8_t **mdl_lid**

Client Model local index

uint8_t **app_key_lid**

App key local index

struct model_cli_trans_info

#include <ls_sig_mesh.h> Start transition to a new state message structure - for Client SIG model.

Public Members

uint32_t **state_1**

Model state1

uint32_t **state_2**

Model state2

uint32_t **trans_time_ms**

Transition time in milliseconds

uint16_t **trans_info**

Transition information

uint16_t **dest_addr**

Destination address

uint16_t **delay_ms**

Delay in milliseconds

uint8_t **mdl_lid**

Client Model local index

uint8_t **app_key_lid**

App key local index

struct start_glp_info

#include <ls_sig_mesh.h> Request parameter structure - for genie lower power.

Public Members

uint8_t **RxDelyMs**

rx delay at least 10ms

uint16_t **SleepIntvlMs**

rx delay at least 10ms

struct start_lpn_info

#include <ls_sig_mesh.h> Request parameter structure - for lower power node.

Public Members

uint8_t **rx_window_factor**

enum lpn_rx_window_factor

uint8_t **min_queue_size_log**

enum enum friend_node_min_queue_size_log

uint8_t **rx_delay_ms**

rx delay at least 10ms

uint16_t **previous_addr**

uint32_t **poll_timeout_100ms**

poll timeout at least 1s

uint32_t **poll_intv_ms**

poll interval

struct lpn_offer_info

#include <ls_sig_mesh.h> Report friend node parameter structure - for lower power node.

Public Members

uint16_t **friend_addr**

friend node unicast address

uint8_t **friend_rx_window**

friend node rx window

uint8_t **friend_queue_size**

friend queue size,enum friend_node_min_queue_size_log

uint8_t **friend_subs_list_size**

friend subscription list size

int8_t **friend_rssi**

friend node rssi

struct lpn_status_info

#include <ls_sig_mesh.h> Report friendship status structure - for lower power node.

Public Members

uint16_t **lpn_status**

Friendship status

uint16_t **friend_addr**

friend node unicast_addr

struct model_rx_info

#include <ls_sig_mesh.h> Receive message structure from other nodes.

Public Members

uint8_t **ModelHandle**

Model local index

uint8_t **app_key_lid**

App Key local index

int8_t **rssi**

rssi of message

uint16_t **source_addr**

source_addr of message

uint8_t **not_relayed**

True if message have been received by an immediate peer;False, it can have been relayed.

uint32_t **opcode**

Operation code

uint16_t **rx_info_len**

message length

uint8_t **info**[**__LSSIGMESH_EMPTY**]

message

struct model_state_upd

#include <ls_sig_mesh.h> State update indication structure for SIG Model.

Public Members

uint8_t **elmt_idx**

Element index

uint16_t **state_id**

State identifier, num mesh_state_idx

uint32_t **state**

New state value or targeted state value depending on transition time

uint32_t **trans_time_ms**

Transition time in milliseconds

struct report_mesh_prov_result_info

#include <ls_sig_mesh.h> Report provisioning state.

Public Members

uint8_t **state**

Provisioning state ,enum mesh_provisioning_result

uint16_t **status**

Provisioning failed reason

struct report_mesh_timer_state_info

#include <ls_sig_mesh.h> Report mesh timer state.

Public Members

uint8_t **timer_id**

mesh timer index

uint8_t **status**

mesh timer status, enum mesh_timer_state

struct model_id_info

#include <ls_sig_mesh.h> Registered Model inforation structure for model.

Public Members

uint8_t **element_id**

Element index

uint8_t **model_lid**

Model local index

uint8_t **vendor_model_role**

Vendor Model Server or Client

uint16_t **sig_model_cfg_idx**

Config Index for Sig Model

uint16_t **vendor_model_cfg_idx**

Config Index for Vendor Model

uint32_t **model_id**

Model Index

struct mesh_model_info

#include <ls_sig_mesh.h> Registered Model all informations structure for node.

Public Members

uint8_t **nb_model**

Total registered Model

uint8_t **app_key_lid**

App key loacal index

struct *model_id_info* **info**[**MAX_MESH_MODEL_NB**]

Maximum models in a node struct *model_id_info*

struct mesh_publish_info_ind

#include <ls_sig_mesh.h> Report new publication parameters for a model.

Public Members

uint8_t **model_lid**

Model local index

uint32_t **period_ms**

Publish period in milliseconds

uint16_t **addr**

Publication address

struct mesh_auto_prov_model_info

#include <ls_sig_mesh.h> Auto provisioning information for model.

Public Members

bool **publish_flag**

Enable Publish function

bool **subs_flag**

Enable Subscript function

uint32_t **model_id**

Model Index

struct mesh_auto_prov_info

#include <ls_sig_mesh.h> Auto provisioning information for node.

Public Members

uint16_t **unicast_addr**

unicast address

uint8_t **model_nb**

Total registered Model

uint16_t **group_addr**

Group address for publish and subscription

struct *mesh_auto_prov_model_info* **model_info**[**MAX_MESH_MODEL_NB**]

Maximum models in a node struct *mesh_auto_prov_model_info*

uint8_t **app_key**[16]

uint8_t **net_key**[16]

union ls_sig_mesh_evt_u

#include <ls_sig_mesh.h> Sig Mesh event union definition.

Public Members

struct *report_dev_provisioned_state_info* **st_proved**

Device state information, struct *report_dev_provisioned_state_info*

struct *report_mesh_attention_info* **update_attention**

Attention state,struct *report_mesh_attention_info*

struct *report_mesh_prov_result_info* **prov_rslt_sate**

Report provisioning state, struct *report_mesh_prov_result_info*

struct *model_rx_info* **rx_msg**

Receive message structure from other nodes, struct *model_rx_info*

struct *update_state_info* **update_state_param**

Report updated state struct, struct *update_state_info*

struct *report_mesh_timer_state_info* **mesh_timer_state**

Report mesh timer state, struct *report_mesh_timer_state_info*

struct *adv_report_evt* **adv_report**

Report adv information, struct *adv_report_evt*

struct *model_state_upd* **mdl_state_upd_ind**

State update indication structure for SIG Model, struct *model_state_upd*

struct *mesh_model_info* **sig_mdl_info**

Registered Model all informations structure for node, struct *mesh_model_info*

struct *mesh_publish_info_ind* **mesh_publish_info**

Report new publication parameters for a model, struct *mesh_publish_info_ind*

struct *mesh_auto_prov_info* **mesh_auto_prov_param**

Auto provisioning information for node, struct *mesh_auto_prov_info*

struct *lpn_offer_info* **lpn_offer_info**

Report friend node parameter structure - for lower power node, struct *lpn_offer_info*

struct *lpn_status_info* **lpn_status_info**

Report friendship status structure - for lower power node, struct *lpn_status_info*

struct ls_sig_mesh_cfg

#include <ls_sig_mesh.h> Sig Mesh initial parameter.

Public Members

uint32_t **MeshFeatures**

Features supported by nodes,enum mesh_feature

uint16_t **MeshCompanyID**

CompanyID supported by SIG

uint16_t **MeshProID**

Production ID

uint16_t **MeshProVerID**

Production Version ID

uint16_t **MeshLocDesc**

user-defined

uint16_t **NbAddrReplay**

Number of relay node support for replay attrack

uint8_t **NbCompDataPage**

Page number of composition data

uint8_t **FrdRxWindowMS**

Rx Windows of friend node

uint8_t **FrdQueueSize**

QueueCache size of friend node

uint8_t **FrdSubsListSize**

Subscript List size of friend node

struct bcn_start_unprov_param

#include <ls_sig_mesh.h> Restart beacon parameter of device.

Public Members

uint8_t **DevUuid**[**UUID_MESH_DEV_LEN**]

uuid of device

uint16_t **OobInfo**

OOB information

uint32_t **UriHash**

URI-Hash information

bool **UriHash_Present**

Support or not

3.1.5 LSSIG_MESH_PROVISIONER API

Defines

CONNECT_IDX_INVALID

MESH_KEY_LENGTH

Enums

enum config_client_get_type

Values:

enumerator CONFIG_CLIENT_GET_TYPE_BEACON

enumerator CONFIG_CLIENT_GET_TYPE_DFLT_TTL

enumerator CONFIG_CLIENT_GET_TYPE_GATT_PROXY

enumerator CONFIG_CLIENT_GET_TYPE_RELAY

enumerator CONFIG_CLIENT_GET_TYPE_FRIEND

enumerator CONFIG_CLIENT_GET_TYPE_HB_PUBLI

enumerator CONFIG_CLIENT_GET_TYPE_HB_SUBS

enumerator CONFIG_CLIENT_GET_TYPE_NET_TRANSMIT

enumerator CONFIG_CLIENT_GET_TYPE_NET_KEYS

enumerator CONFIG_CLIENT_GET_TYPE_COMPO_DATA

enumerator CONFIG_CLIENT_GET_TYPE_LPN_POLLTIEOUT

enumerator CONFIG_CLIENT_GET_TYPE_MAX

enum config_client_set_type

Values:

enumerator CONFIG_CLIENT_SET_TYPE_BEACON

enumerator CONFIG_CLIENT_SET_TYPE_DFLT_TTL

enumerator CONFIG_CLIENT_SET_TYPE_GATT_PROXY

enumerator CONFIG_CLIENT_SET_TYPE_FRIEND

enumerator CONFIG_CLIENT_SET_TYPE_RESET

enumerator CONFIG_CLIENT_SET_TYPE_NET_TX

enumerator CONFIG_CLIENT_SET_TYPE_RELAY

enumerator CONFIG_CLIENT_SET_TYPE_MAX

enum config_client_value_type

Values:

enumerator CONFIG_CLIENT_GET_VAL_TYPE_BEACON

enumerator CONFIG_CLIENT_GET_VAL_TYPE_COMPO_DATA

enumerator CONFIG_CLIENT_GET_VAL_TYPE_DFLT_TTL

enumerator CONFIG_CLIENT_GET_VAL_TYPE_GATT_PROXY

enumerator CONFIG_CLIENT_GET_VAL_TYPE_RELAY

enumerator CONFIG_CLIENT_GET_VAL_TYPE_MDL_PUBLI

enumerator CONFIG_CLIENT_GET_VAL_TYPE_MDL_SUBS

enumerator CONFIG_CLIENT_GET_VAL_TYPE_MDL_SUBS_LIST

enumerator CONFIG_CLIENT_GET_VAL_TYPE_NETKEY

enumerator CONFIG_CLIENT_GET_VAL_TYPE_NETKEY_LIST

enumerator CONFIG_CLIENT_GET_VAL_TYPE_APPKEY

enumerator CONFIG_CLIENT_GET_VAL_TYPE_APPKEY_LIST

enumerator CONFIG_CLIENT_GET_VAL_TYPE_NODE_IDENTITY

enumerator CONFIG_CLIENT_GET_VAL_TYPE_MDL_APP

enumerator CONFIG_CLIENT_GET_VAL_TYPE_MDL_APP_LIST

enumerator CONFIG_CLIENT_GET_VAL_TYPE_NODE_RESET

enumerator CONFIG_CLIENT_GET_VAL_TYPE_FRIEND

enumerator CONFIG_CLIENT_GET_VAL_TYPE_PHASE

enumerator CONFIG_CLIENT_GET_VAL_TYPE_HB_PUBLI

enumerator CONFIG_CLIENT_GET_VAL_TYPE_HB_SUBS

enumerator CONFIG_CLIENT_GET_VAL_TYPE_LPN_POLLTIMEOUT

enumerator CONFIG_CLIENT_GET_VAL_TYPE_NET_TX

enum config_client_net_action_type

Values:

enumerator CONFIG_CLIEN_ADD_NET_KEY_TYPE

enumerator CONFIG_CLIEN_UPDATE_NET_KEY_TYPE

enumerator CONFIG_CLIEN_DELETE_NET_TYPE

enumerator CONFIG_CLIEN_GET_NET_KEY_BOUND_APP_KEY

enumerator CONFIG_CLIEN_GET_NET_KEY_TYPE_NODE_ID

enumerator CONFIG_CLIEN_GET_NET_KEY_TYPE_PHASE

enumerator CONFIG_CLIEN_SET_NET_KEY_TYPE_PHASE

enumerator CONFIG_CLIEN_SET_NET_KEY_TYPE_NODE_ID

enumerator CONFIG_CLIEN_NET_KEY_TYPE_MAX

enum config_client_app_action_type

Values:

enumerator CONFIG_CLIENT_ADD_APP_KEY_TYPE

enumerator CONFIG_CLIENT_UPDATE_APP_KEY_TYPE

enumerator CONFIG_CLIENT_DELETE_APP_KEY_TYPE

enumerator CONFIG_CLIENT_ACTIVE_APP_TYPE_MAX

enum config_client_mdl_get_type

Values:

enumerator CONFIG_CLIENT_MDL_GET_TYPE_PUBLI

enumerator CONFIG_CLIENT_MDL_GET_TYPE_SUBS

enumerator CONFIG_CLIENT_MDL_GET_TYPE_APP

enumerator CONFIG_CLIENT_MDL_GET_TYPE_MAX

enum config_client_mdl_subs_action_type

Values:

enumerator CONFIG_CLIENT_MDL_SUBS_ACTION_TYPE_ADD

enumerator CONFIG_CLIENT_MDL_SUBS_ACTION_TYPE_DELETE

enumerator CONFIG_CLIENT_MDL_SUBS_ACTION_TYPE_OVERWRITE

enumerator CONFIG_CLIENT_MDL_SUBS_ACTION_TYPE_DELETE_ALL

enumerator CONFIG_CLIENT_MDL_SUBS_ACTION_TYPE_MAX

enum config_client_mdl_app_action_type

Values:

enumerator CONFIG_CLIENT_MDL_APP_ACTION_TYPE_BIND

enumerator CONFIG_CLIENT_MDL_APP_ACTION_TYPE_UNBIND

enumerator CONFIG_CLIENT_MDL_APP_ACTION_TYPE_MAX

enum config_client_mdl_publi_set_type

Values:

enumerator CONFIG_CLIENT_MDL_PUBLI_SET_TYPE_ADDR

enumerator CONFIG_CLIENT_MDL_PUBLI_SET_TYPE_VADDR

enumerator CONFIG_CLIENT_MDL_PUBLI_SET_TYPE_MAX

enum mesh_provisioner_evt_type

Values:

enumerator MESH_PROVER_GET_PROV_AUTH_INFO

enumerator MESH_PROVER_KEY_DEV_ADD_RSP_INFO

enumerator MESH_PROVER_KEY_NET_ADD_IND

enumerator MESH_PROVER_KEY_APP_ADD_IND

enumerator MESH_PROVER_HEALTH_MODEL_RSP_INFO

enumerator MESH_PROVER_SET_DEV_RSP_INFO

enumerator MESH_PROVER_IDENTIFY_REQ_IND_INFO

enumerator MESH_PROVER_EVT_MAX

enum mesh_provisioner_rx_ind_type

Values:

enumerator MESH_PROVER_ACTIVE_NODE_FOUND

enumerator MESH_PROVER_ACTIVE_NODE_GATT

enumerator MESH_PROVER_ACTIVE_NODE_STOPPED

enumerator MESH_PROVER_ACTIVE_PROXY_SVC

enumerator MESH_PROVER_ACTIVE_STATE

enumerator MESH_PROVER_CONFC_GET_BEACON_STATUS

enumerator MESH_PROVER_CONFC_GET_DEFAULT_TTL_STATUS

enumerator MESH_PROVER_CONFC_GET_GATT_PROXY_STATUS

enumerator MESH_PROVER_CONFC_GET_FRIEND_STATUS

enumerator MESH_PROVER_CONFC_MODEL_PUB_STATUS

enumerator MESH_PROVER_CONFC_MODEL_SUBS_STATUS

enumerator MESH_PROVER_CONFC_MODEL_APP_STATUS

enumerator MESH_PROVER_CONFC_COMP_DATA_PAGE0

enumerator MESH_PROVER_CONFC_COMP_DATA_PAGE0_ELEMENT

enumerator MESH_PROVER_CONFC_APP_KEY_STATUS

enumerator MESH_PROVER_CONFC_NET_TX_STATUS

enumerator MESH_PROVER_RX_IND_MAX

enum prover_provisioning_state

Values:

enumerator PROVISIONING_STARTED

enumerator PROVISIONING_SUCCEED

enumerator PROVISIONING_FAILED

Functions

struct prover_active_state_info __attribute__ ((packed))

void **prf_ls_sig_mesh_provisioner_callback_init**(void (**evt_cb*)) **enum** *mesh_provisioner_evt_type*, **union** *ls_sig_mesh_provisioner_evt_u** , void (**rx_ind_handle*)**enum** *mesh_provisioner_rx_ind_type*, **union** *ls_sig_mesh_provisioner_rx_info_u**

参数

- **evt_cb** --
- **rx_ind_handle** --

void **ls_sig_mesh_add_uuid_unicast_addr**(**const** uint16_t *unicast_addr*, **const** uint8_t **dev_uuid*)

参数

- **unicast_addr** --
- **dev_uuid** --

void **ls_sig_mesh_provisioner_init**(uint16_t**const** *unicast_addr*)

参数 **unicast_addr** -- provisioner unicast address

void **ls_sig_mesh_provisioner_add_net_key**(**const** uint16_t *netkey_id*, **const** uint8_t **net_key*)

参数

- **netkey_id** --
- **net_key** --

void **ls_sig_mesh_provisioner_use_new_net_key**(**const** uint8_t *netkey_id*)

参数 **netkey_id** --

void **ls_sig_mesh_provisioner_update_net_key**(**const** uint8_t *netkey_id*, **const** uint8_t **net_key*)

参数

- **netkey_id** --
- **net_key** --

void **ls_sig_mesh_provisioner_revoke_old_net_key**(**const** uint8_t *netkey_id*)

参数 **netkey_id** --

void **ls_sig_mesh_provisioner_delete_net_key**(**const** uint8_t *netkey_id*)

参数 **netkey_id** --

void **ls_sig_mesh_provisioner_add_app_key**(**const** uint8_t *netkey_id*, **const** uint8_t *appkey_id*, **const** uint8_t **app_key*)

参数

- **netkey_id** --
- **appkey_id** --
- **app_key** --

void **ls_sig_mesh_provisioner_update_app_key**(**const** uint8_t *netkey_id*, **const** uint8_t *appkey_id*, **const** uint8_t **app_key*)

参数

- **netkey_id** --
- **appkey_id** --
- **app_key** --

void **ls_sig_mesh_provisioner_delete_app_key**(**const** uint8_t *netkey_id*, **const** uint8_t *appkey_id*)

参数

- **netkey_id** --
- **appkey_id** --

void **ls_sig_mesh_provisioner_scan**(uint16_t *timeout*)

参数 **timeout** --

void **ls_sig_mesh_provisioner_invite**(**const** uint8_t *conidx*, **const** uint8_t **dev_uuid*, **const** uint16_t *unicast_addr*, **const** uint8_t *attention_dur_s*)

参数

- **conidx** --
- **dev_uuid** --
- **unicast_address** --
- **attention_dur_s** --

void **ls_sig_mesh_provisioner_stop**(void)

void **app_mesh_prover_set_prov_auth_info**(**struct** *mesh_prov_auth_info *param*)

参数 **param** --

void **ls_sig_mesh_identify_cfm**(bool *accept*, uint8_t *netkey_lid*, uint8_t *algo*, uint8_t *pub_key*, uint8_t *auth_method*, uint8_t *auth_action*, uint8_t *auth_size*)

参数

- **accept** --

- **netkey_lid** --
- **unicast_addr** --
- **algo** --
- **pub_key** --
- **auth_method** --
- **auth_action** --
- **auth_size** --

void **ls_sig_mesh_prover_config_reg_model**(uint16_t *primary_addr*)

参数 **primary_addr** --

void **ls_sig_mesh_prover_config_set_dev**(uint8_t *dev_key_lid*, uint8_t *net_key_lid*, uint16_t *primary_addr*)

参数

- **dev_key_lid** --
- **net_key_lid** --
- **primary_addr** --

void **ls_sig_mesh_prover_config_client_active_netkey**(uint8_t *act_netkey_type*, uint16_t *net_key_id*, **const** uint8_t **data*, uint8_t *data_len*)

参数

- **act_netkey_type** --
- **net_key_id** --
- **data** --
- **data_len** --

void **ls_sig_mesh_prover_config_client_active_appkey**(uint8_t *act_appkey_type*, uint16_t *net_key_id*, uint16_t *app_key_id*, **const** uint8_t **appkey*)

参数

- **act_appkey_type** --
- **net_key_id** --
- **app_key_id** --
- **appkey** --

void **ls_sig_mesh_prover_config_client_get_model**(uint8_t *get_model_type*, uint16_t *element_address*, uint32_t *model_id*)

参数

- **get_model_type** --
- **element_address** --
- **model_id** --

void **ls_sig_mesh_prover_config_client_act_model_subscript**(uint8_t *model_subscript_act_type*, uint16_t *element_address*, uint32_t *model_id*, bool *address_type*, uint16_t *length*, **const** uint8_t **address_uuid*)

参数

- **model_subscript_act_type** --
- **element_address** --
- **model_id** --
- **address_type** --
- **length** --
- **address_uuid** --

void **ls_sig_mesh_prover_config_client_set_model_publication**(uint8_t *addr_type*, uint16_t *element_addr*, uint16_t *appkey_id*, bool *cred_flag*, uint8_t *pub_ttl*, uint8_t *pub_period*, uint8_t *retx_cnt*, uint8_t *retx_intv_slots*, uint32_t *mdl_id*, uint16_t *length*, **const** uint8_t **val*)

参数

- **addr_type** --
- **element_addr** --
- **appkey_id** --
- **cred_flag** --
- **pub_ttl** --
- **pub_period** --

- **retx_cnt** --
- **retx_intv_slots** --
- **mdl_id** --
- **length** --
- **val** --

void **ls_sig_mesh_prover_config_client_act_model_app**(uint8_t *mdl_app_act_type*, uint16_t *element_addr*, uint16_t *appkey_id*, uint32_t *mdl_id*)

参数

- **mdl_app_act_type** --
- **element_addr** --
- **appkey_id** --
- **mdl_id** --

void **ls_sig_mesh_prover_config_client_set**(uint8_t *set_type*, uint8_t *value*, uint8_t *tx_cnt*, uint8_t *intv_slots*)

参数

- **set_type** --
- **value** --
- **tx_cnt** --
- **intv_slots** --

void **ls_sig_mesh_prover_config_client_get**(uint8_t *get_type*, uint16_t *value*)

参数

- **get_type** --
- **value** --

void **ls_sig_mesh_prover_config_client_set_heartbeat_publication**(uint16_t *dst*, uint16_t *cnt*, uint16_t *period_s*, uint8_t *ttl*, uint16_t *features*, uint16_t *nekkey_id*)

参数

- **dst** --

- **cnt** --
- **period_s** --
- **ttl** --
- **features** --
- **nekkey_id** --

void **ls_sig_mesh_prover_config_client_set_heartbeat_subscription**(uint16_t *src*, uint16_t *dst*, uint16_t *period_s*)

参数

- **src** --
- **dst** --
- **period_s** --

void **ls_sig_mesh_prover_health_client_get**(uint16_t *address*, uint8_t *appkey_lid*, uint8_t *get_type*, uint16_t *company_id*)

参数

- **address** --
- **appkey_lid** --
- **get_type** --
- **company_id** --

void **ls_sig_mesh_prover_health_client_set**(uint16_t *address*, uint8_t *appkey_lid*, uint8_t *set_type*, uint8_t *set_cfg*, uint8_t *val*)

参数

- **address** --
- **appkey_lid** --
- **set_type** --
- **set_cfg** --
- **val** --

void **ls_sig_mesh_prover_health_client_act_fault**(uint16_t *address*, uint8_t *appkey_lid*, uint8_t *act_type*, uint8_t *act_cfg*, uint8_t *test_id*, uint16_t *company_id*)

参数

- **address** --

- **appkey_lid** --
- **act_type** --
- **act_cfg** --
- **test_id** --
- **company_id** --

void **ls_sig_mesh_prover_health_client_reg_model**(void)

Variables

enum config_client_get_type __attribute__

uint8_t **state**

uint16_t **status**

uint16_t **unicast_addr**

uint8_t **dev_nb_elt**

uint16_t **dev_algorithms**

uint8_t **dev_pub_key_type**

uint8_t **dev_static_oob_type**

uint8_t **dev_out_oob_size**

uint16_t **dev_out_oob_action**

uint8_t **dev_in_oob_size**

uint16_t **dev_in_oob_action**

uint16_t **loc_desc**

uint8_t **number_sig_models**

uint8_t **number_vendor_models**

uint32_t **model_info**[**__LSSIGMESH_EMPTY**]

uint8_t **active_status**

uint16_t **net_key_id**

uint16_t **app_key_id**

uint16_t **element_addr**

uint32_t **model_id**

uint16_t **value**

uint16_t **publish_addr**

bool **cred_flag**

uint8_t **publish_ttl**

uint32_t **publish_period_ms**

uint8_t **publish_retx_cont**

uint8_t **publish_retx_intv_step_solution**

struct prover_node_scan_info

Public Members

uint32_t **uri_hash**

uint16_t **oob_info**

int8_t **rssi**

uint8_t **dev_uuid**[**UUID_MESH_DEV_LEN**]

struct prover_active_state_info

Public Members

uint8_t **state**

uint16_t **status**

uint16_t **unicast_addr**

struct prover_add_dev_key_rsp_info

Public Members

uint8_t **dev_key_lid**

struct prover_add_net_key_ind_info

Public Members

uint8_t **net_key_lid**

struct prover_add_app_key_ind_info

Public Members

uint8_t **app_key_lid**

struct prover_health_client_model_rsp_info

Public Members

uint8_t **mdl_lid**

struct prover_identify_req_ind_info

Public Members

uint8_t **dev_nb_elt**

uint16_t **dev_algorithms**

uint8_t **dev_pub_key_type**

uint8_t **dev_static_oob_type**

uint8_t **dev_out_oob_size**

uint16_t **dev_out_oob_action**

uint8_t **dev_in_oob_size**

uint16_t **dev_in_oob_action**

struct prover_confc_get_compo_data_ind_info

Public Members

uint16_t **unicast_addr**

uint16_t **company_id**

uint16_t **product_id**

uint16_t **version_id**

uint16_t **min_num_replay**

uint16_t **support_features**

uint8_t **dev_nb_elements**

struct prover_confc_get_compo_data_element_ind_info

Public Members

uint16_t **unicast_addr**

uint16_t **loc_desc**

uint8_t **number_sig_models**

uint8_t **number_vendor_models**

uint32_t **model_info**[**__LSSIGMESH_EMPTY**]

struct prover_confc_get_default_ttl_ind_info

Public Members

uint16_t **unicast_addr**

uint8_t **default_ttl**

struct prover_confc_get_app_key_status_ind_info

Public Members

uint16_t **unicast_addr**

uint8_t **active_status**

uint16_t **net_key_id**

uint16_t **app_key_id**

struct prover_confc_model_subs_app_status_ind_info

Public Members

uint16_t **unicast_addr**

uint8_t **status**

uint16_t **element_addr**

uint32_t **model_id**

uint16_t **value**

struct prover_confc_model_pubs_status_ind_info

Public Members

uint16_t **unicast_addr**

uint8_t **status**

uint16_t **element_addr**

uint16_t **publish_addr**

uint16_t **app_key_id**

bool **cred_flag**

uint8_t **publish_ttl**

uint32_t **publish_period_ms**

uint8_t **publish_retx_cont**

uint8_t **publish_retx_intv_step_solution**

uint32_t **model_id**

union ls_sig_mesh_provisioner_evt_u

Public Members

struct *prover_add_dev_key_rsp_info* **prover_node_add_dev_key_rsp_info**

struct *prover_health_client_model_rsp_info* **prover_node_health_model_rsp_info**

struct *prover_add_net_key_ind_info* **prover_node_add_net_key_ind_info**

struct *prover_add_app_key_ind_info* **prover_node_add_app_key_ind_info**

struct *prover_identify_req_ind_info* **prover_identify_req_ind_info**

union ls_sig_mesh_provisioner_rx_info_u

Public Members

struct *prover_active_state_info* **prover_node_state_info**

struct *prover_node_scan_info* **prover_node_scan_info**

struct *prover_confc_get_compo_data_ind_info* **confc_get_compo_data_info**

struct *prover_confc_get_compo_data_element_ind_info* **confc_get_compo_data_element_info**

struct *prover_confc_get_default_ttl_ind_info* **confc_get_default_ttl_info**

struct *prover_confc_get_app_key_status_ind_info* **confc_get_app_key_status_info**

struct *prover_confc_model_subs_app_status_ind_info* **confc_model_subs_app_status_info**

struct *prover_confc_model_pubs_status_ind_info* **confc_model_pubs_status_info**

3.2 PERIPHERAL API

3.2.1 SPI FLASH API

Defines

FLASH_PAGE_SIZE

Flash Page Size.

FLASH_SECTOR_SIZE

Flash Sector Size.

STATUS_REG1_SUS1_MASK

Status Register 1 SUS1 Mask.

STATUS_REG1_SUS2_MASK

Status Register 1 SUS2 Mask.

Functions

void **spi_flash_dual_mode_set**(bool *dual*)

bool **spi_flash_dual_mode_get**(void)

void **spi_flash_xip_status_set**(bool *xip*)

Set the status variable indicating whether the Flash is in XIP mode.

void **spi_flash_writing_status_set**(bool *writing*)

Set the status variable indicating whether the Flash is programming or erasing.

void **spi_flash_drv_var_init**(bool *xip*, bool *writing*)

Initialize variables of the SPI Flash driver.

void **spi_flash_init**(void)

Initialize SPI Flash driver.

void **spi_flash_xip_start**(void)

Enter SPI Flash XIP mode.

void **spi_flash_xip_stop**(void)

Exit SPI Flash XIP mode.

void **spi_flash_read_status_register_0**(uint8_t **status_reg_0*)

Read Status Register 0.

参数 **status_reg_0** -- **[out]**

void **spi_flash_read_status_register_1**(uint8_t *status_reg_1)

Read Status Register 1.

参数 **status_reg_1** -- **[out]**

bool **spi_flash_write_in_process**(void)

Check WIP Status.

返回 WIP status

void **spi_flash_write_status_register**(uint16_t *status*)

Write Status Register.

参数 **status** -- **[in]** The value to write to Status Register

void **spi_flash_dual_page_program**(uint32_t *offset*, uint8_t *data, uint16_t *length*)

void **spi_flash_quad_page_program**(uint32_t *offset*, uint8_t *data, uint16_t *length*)

Quad Page Program.

参数

- **offset** -- **[in]** Offset to FLASH_BASE_ADDR
- **data** -- **[in]** The pointer of the data to program into Flash
- **length** -- **[in]** Data length in bytes, must be less than or equal to FLASH_PAGE_SIZE

void **spi_flash_page_program**(uint32_t *offset*, uint8_t *data, uint16_t *length*)

Page Program.

参数

- **offset** -- **[in]** Offset to FLASH_BASE_ADDR
- **data** -- **[in]** The pointer of the data to program into Flash
- **length** -- **[in]** Data length in bytes, must be less than or equal to FLASH_PAGE_SIZE

void **spi_flash_page_erase**(uint32_t *offset*)

Page Erase.

参数 **offset** -- **[in]** Offset to FLASH_BASE_ADDR

void **spi_flash_sector_erase**(uint32_t *offset*)

Sector Erase.

参数 **offset** -- **[in]** Offset to FLASH_BASE_ADDR

void **spi_flash_chip_erase**(void)

Chip Erase.

void **spi_flash_multi_io_read**(uint32_t *offset*, uint8_t *data, uint16_t *length*)

void **spi_flash_dual_io_read**(uint32_t *offset*, uint8_t *data, uint16_t *length*)

void **spi_flash_quad_io_read**(uint32_t *offset*, uint8_t **data*, uint16_t *length*)

Quad IO Read.

参数

- **offset** -- **[in]** Offset to FLASH_BASE_ADDR
- **data** -- **[out]** The pointer of the data buffer
- **length** -- **[in]** Data length in bytes

void **spi_flash_fast_read**(uint32_t *offset*, uint8_t **data*, uint16_t *length*)

Fast Read.

参数

- **offset** -- **[in]** Offset to FLASH_BASE_ADDR
- **data** -- **[out]** The pointer of the data buffer
- **length** -- **[in]** Data length in bytes

void **spi_flash_deep_power_down**(void)

Deep Power Down.

void **spi_flash_release_from_deep_power_down**(void)

Release From Deep Power Down.

void **spi_flash_read_id**(uint8_t *jedec_id*[3])

Read ID.

参数 **jedec_id** -- **[out]** The buffer for JEDEC ID

void **spi_flash_read_unique_id**(uint8_t *unique_serial_id*[16])

Read Unique ID.

参数 **unique_serial_id** -- **[out]** The buffer for unique serial ID

void **spi_flash_erase_security_area**(uint8_t *idx*)

Erase Security Area.

参数 **idx** -- **[in]** The index of security area

void **spi_flash_program_security_area**(uint8_t *idx*, uint16_t *addr*, uint8_t **data*, uint16_t *length*)

Program Security Area.

参数

- **idx** -- **[in]** The index of security area
- **addr** -- **[in]** The address of security area
- **data** -- **[in]** The pointer of the data to program into security area
- **length** -- **[in]** Data length in bytes, must be less than or equal to the size of one security area

void **spi_flash_read_security_area**(uint8_t *idx*, uint16_t *addr*, uint8_t **data*, uint16_t *length*)

Read Security Area.

参数

- **idx** -- **[in]** The index of security area
- **addr** -- **[in]** The address of security area
- **data** -- **[out]** The pointer of the data buffer
- **length** -- **[in]** Data length in bytes

void **spi_flash_software_reset**(void)

Software Reset.

void **spi_flash_qe_status_read_and_set**(void)

Check and Set Quad Enable Status.

void **spi_flash_prog_erase_suspend**(void)

Suspend Program or Erase Operation.

void **spi_flash_prog_erase_resume**(void)

Resume Program or Erase Operation.

bool **spi_flash_writing_busy**(void)

Check if Flash is writing.

返回 Programming or Erasing status

bool **spi_flash_xip_status_get**(void)

Check if Flash is in XIP mode.

返回 XIP status

3.2.2 GPIO API

Defines

PA00

GPIOA00 selected

PA01

GPIOA01 selected

PA02

GPIOA02 selected

PA03

GPIOA03 selected

PA04
GPIOA04 selected

PA05
GPIOA05 selected

PA06
GPIOA06 selected

PA07
GPIOA07 selected

PA08
GPIOA08 selected

PA09
GPIOA09 selected

PA10
GPIOA10 selected

PA11
GPIOA11 selected

PA12
GPIOA12 selected

PA13
GPIOA13 selected

PA14
GPIOA14 selected

PA15
GPIOA15 selected

PB00
GPIOB00 selected

PB01
GPIOB01 selected

PB02
GPIOB02 selected

PB03
GPIOB03 selected

PB04
GPIOB04 selected

PB05
GPIOB05 selected

PB06
GPIOB06 selected

PB07
GPIOB07 selected

PB08
GPIOB08 selected

PB09
GPIOB09 selected

PB10
GPIOB10 selected

PB11
GPIOB11 selected

PB12
GPIOB12 selected

PB13
GPIOB13 selected

PB14
GPIOB14 selected

PB15
GPIOB15 selected

PC00
GPIOC00 selected

PC01
GPIOC01 selected

PC02
GPIOC02 selected

PC03
GPIOC03 selected

PC04
GPIOC04 selected

PC05
GPIOC05 selected

PC06
GPIOC06 selected

PC07
GPIOC07 selected

PC08
GPIOC08 selected

PC09
GPIOC09 selected

PC10
GPIOC10 selected

PC11
GPIOC11 selected

PC12
GPIOC12 selected

PC13
GPIOC13 selected

PC14
GPIOC14 selected

PC15
GPIOC15 selected

PD00
GPIOD00 selected

PD01
GPIOD01 selected

PD02
GPIOD02 selected

PD03
GPIOD03 selected

PD04
GPIOD04 selected

PD05
GPIOD05 selected

PD06
GPIOD06 selected

PD07
GPIOD07 selected

PD08
GPIOD08 selected

PD09
GPIOD09 selected

PD10
GPIOD10 selected

PD11
GPIOD11 selected

PD12
GPIOD12 selected

PD13
GPIOD13 selected

PD14
GPIOD14 selected

PD15
GPIOD15 selected

Enums

enum io_pull_type_t
IO pull type.

Values:

enumerator IO_PULL_DISABLE

enumerator IO_PULL_UP

enumerator IO_PULL_DOWN

enum exti_edge_t
Values:

enumerator INT_EDGE_FALLING

enumerator INT_EDGE_RISING

Functions

void **io_init**(void)

GPIO Init.

void **io_cfg_output**(uint8_t *pin*)

GPIO config output.

参数 **pin** -- Specific GPIO pin

void **io_cfg_opendrain**(uint8_t *pin*)

GPIO config open drain output mode.

参数 **pin** -- Specific GPIO pin

void **io_cfg_pushpull**(uint8_t *pin*)

GPIO config pushpull output mode.

参数 **pin** -- Specific GPIO pin

void **io_cfg_input**(uint8_t *pin*)

GPIO config input.

参数 **pin** -- Specific GPIO pin

void **io_cfg_disable**(uint8_t *pin*)

GPIO input disable.

参数 **pin** -- Specific GPIO pin

void **io_write_pin**(uint8_t *pin*, uint8_t *val*)

GPIO config output.

参数

- **pin** -- Specific GPIO pin
- **val** -- GPIO level status 0 means low level 1 means high level

void **io_set_pin**(uint8_t *pin*)

set GPIO high level

参数 **pin** -- Specific GPIO pin

void **io_clr_pin**(uint8_t *pin*)

set GPIO low level

参数 **pin** -- Specific GPIO pin

void **io_toggle_pin**(uint8_t *pin*)

toggle GPIO

参数 **pin** -- Specific GPIO pin

uint8_t **io_get_output_val**(uint8_t *pin*)

get GPIO output level

参数 **pin** -- Specific GPIO pin

返回 **GPIO** -- output level 0 means low level 1 means high level

uint8_t **io_read_pin**(uint8_t *pin*)

get GPIO input level

参数 **pin** -- Specific GPIO pin

返回 **GPIO** -- output level 0 means low level 1 means high level

void **io_pull_write**(uint8_t *pin*, *io_pull_type_t* *pull*)

set GPIO pullup or pulldwon or nullpull

参数

- **pin** -- Specific GPIO pin
- **pull** -- Configure the GPIO pull up and down ,This parameter can be a value of *io_pull_type_t*

io_pull_type_t **io_pull_read**(uint8_t *pin*)

read GPIO pull states

参数 **pin** -- Specific GPIO pin

返回 **GPIO** -- pullup and pulldown state This parameter can be a value of *io_pull_type_t*

void **io_ext_intrp_enable**(uint8_t *pin*)

GPIO external interrupt enable.

参数 **pin** -- Specific GPIO pin

void **io_ext_intrp_disable**(uint8_t *pin*)

GPIO external interrupt disable.

参数 **pin** -- Specific GPIO pin

void **io_exti_config**(uint8_t *pin*, *exti_edge_t* *edge*)

Sets the trigger edge for IO interrupt.

参数

- **pin** -- Specific GPIO pin
- **edge** -- edges for IO interrupts

void **io_exti_enable**(uint8_t *pin*, bool *enable*)

GPIO external interrupt enable or disable.

参数

- **pin** -- Specific GPIO pin

- **enable** --

void **io_exti_callback**(uint8_t *pin*)

GPIO external interrupt callback.

参数 **pin** -- specific GPIO pin

void **set_gpio_mode**(*gpio_pin_t *pin*)

Set GPIO mode.

参数 **pin** -- specific GPIO pin

void **spi_flash_io_init**(void)

Init IO for SPI Flash (CS CLK DQ0 DQ1)

void **spi_flash_io_deinit**(void)

DeInit SPI Flash IO.

void **qspi_flash_io_init**(void)

Init IO for QSPI Flash (CS CLK DQ0 DQ1 DQ2 DQ3)

void **qspi_flash_io_deinit**(void)

DeInit QSPI Flash IO.

void **ssi_clk_io_init**(uint8_t *clk*)

GPIO initilization.

参数 **clk** -- Specific GPIO pin

void **ssi_nss0_io_init**(uint8_t *nss0*)

GPIO initilization as ssi nss0.

参数 **nss0** -- Specific GPIO pin

void **ssi_nss1_io_init**(uint8_t *nss1*)

Set pin mux function to ssi nss1.

参数 **nss1** -- Specific GPIO pin

void **ssi_dq0_io_init**(uint8_t *dq0*)

config GPIO to ssi dq0

参数 **dq0** -- Specific GPIO pin

void **ssi_dq1_io_init**(uint8_t *dq1*)

Set pin mux function to ssi dq1.

参数 **dq1** -- Specific GPIO pin

void **ssi_dq2_io_init**(uint8_t *dq2*)

Set pin mux function to ssi dq2.

参数 **dq2** -- Specific GPIO pin

void **ssi_dq3_io_init**(uint8_t *dq3*)

Set pin mux function to ssi dq3.

参数 dq3 -- Specific GPIO pin

void **spi2_clk_io_init**(uint8_t *clk*)

Set pin mux function to spi2 clk.

参数 clk -- Specific GPIO pin

void **spi2_nss_io_init**(uint8_t *nss*)

Set pin mux function to spi2 nss.

参数 nss -- Specific GPIO pin

void **spi2_mosi_io_init**(uint8_t *mosi*)

Set pin mux function to spi2 mosi.

参数 mosi -- Specific GPIO pin

void **spi2_miso_io_init**(uint8_t *miso*)

Set pin mux function to spi2 mosi.

参数 miso -- Specific GPIO pin

void **spi2_clk_io_deinit**(void)

clear pin mux function of spi2_clk

void **spi2_nss_io_deinit**(void)

clear pin mux functions of spi2_nss

void **spi2_mosi_io_deinit**(void)

clear pin mux function of spi2_mosi

void **spi2_miso_io_deinit**(void)

clear pin mux function of spi2_miso

void **iic1_io_init**(uint8_t *scl*, uint8_t *sda*)

Set pin mux function as iic1_io_init.

参数

- **scl** -- Specific GPIO pin
- **sda** -- Specific GPIO pin

void **iic1_io_deinit**(void)

clean pin mux function of iic1

void **iic2_io_init**(uint8_t *scl*, uint8_t *sda*)

Set pin mux function as iic2_io_init.

参数

- **scl** -- Specific GPIO pin
- **sda** -- Specific GPIO pin

void **iic2_io_deinit**(void)

clean pin mux function of iic2

void **uart1_io_init**(uint8_t *txd*, uint8_t *rxd*)

Set pin mux function to uart1 (config of gpio)

参数

- **txd** -- Specific GPIO pin
- **rxd** -- Specific GPIO pin

void **uart1_io_deinit**(void)

clear pin mux function of uart1

void **uart1_7816_io_init**(uint8_t *txd*, uint8_t *ck*)

Set pin mux function to uart1_7816.

参数

- **txd** -- Specific GPIO pin
- **ck** -- Specific GPIO pin

void **uart1_7816_io_deinit**(void)

clear pin mux function of uart1_7816

void **uart2_io_init**(uint8_t *txd*, uint8_t *rxd*)

Set pin mux function to uart2.

参数

- **txd** -- Specific GPIO pin
- **rxd** -- Specific GPIO pin

void **uart2_io_deinit**(void)

clear pin mux function of uart2

void **uart3_io_init**(uint8_t *txd*, uint8_t *rxd*)

Set pin mux function to uart3.

参数

- **txd** -- Specific GPIO pin
- **rxd** -- Specific GPIO pin

void **uart3_io_deinit**(void)

clear pin mux function of uart3

void **adtim1_ch1_io_init**(uint8_t *pin*, bool *output*, uint8_t *default_val*)

Set pin mux function to adtim1.

参数

- **pin** -- Specific GPIO pin
- **output** -- config pin output status
- **default_val** -- config pin output high level or low level

void **adtim1_ch1_io_deinit**(void)

clear mux function of pin adtim1

void **adtim1_ch2_io_init**(uint8_t *pin*, bool *output*, uint8_t *default_val*)

Set mux function of pin adtmi1 of channel 2.

参数

- **pin** -- Specific GPIO pin
- **output** -- config pin output status
- **default_val** -- config pin output high level or low level

void **adtim1_ch2_io_deinit**(void)

clear mux function of pin adtim1

void **adtim1_ch3_io_init**(uint8_t *pin*, bool *output*, uint8_t *default_val*)

Set mux function of pin adtmi1 of channel 3.

参数

- **pin** -- Specific GPIO pin
- **output** -- config pin output status
- **default_val** -- config pin output high level or low level

void **adtim1_ch3_io_deinit**(void)

Set pin mux function.

void **adtim1_ch4_io_init**(uint8_t *pin*, bool *output*, uint8_t *default_val*)

Set mux function of pin adtmi1 of channel 4.

参数

- **pin** -- Specific GPIO pin
- **output** -- config pin output status
- **default_val** -- config pin output high level or low level

void **adtim1_ch4_io_deinit**(void)

clear mux function of pin adtim1 of channel 4

void **adtim1_ch1n_io_init**(uint8_t *pin*)

Set mux function of pin adtim1 of ch1n as input and output.

参数 pin -- Specific GPIO pin

void **adtim1_ch1n_io_deinit**(void)

clear mux function of pin adtim1 of ch1n

void **adtim1_ch2n_io_init**(uint8_t *pin*)

Set mux function of pin adtim1 of ch2n.

参数 pin -- Specific GPIO pin

void **adtim1_ch2n_io_deinit**(void)

clear mux function of pin adtim1 of ch2n

void **adtim1_ch3n_io_init**(uint8_t *pin*)

Set mux function of pin adtim1 of ch3n.

参数 pin -- Specific GPIO pin

void **adtim1_ch3n_io_deinit**(void)

clear mux function of pin adtim1 of ch3n

void **adtim1_etr_io_init**(uint8_t *pin*)

Set mux function of pin adtim1 of etr.

参数 pin -- Specific GPIO pin

void **adtim1_etr_io_deinit**(void)

clear mux function of pin adtim1 of etr

void **adtim1_bk_io_init**(uint8_t *pin*)

configure gpio pin as adtim1_bk

参数 pin -- Specific GPIO pin

void **adtim1_bk_io_deinit**(void)

clear mux function of pin adtim1_bk

void **gptima1_ch1_io_init**(uint8_t *pin*, bool *output*, uint8_t *default_val*)

Set mux function of pin gptima1 of channel 1.

参数

- **pin** -- Specific GPIO pin
- **output** -- config pin output status
- **default_val** -- config pin output high level or low level

void **gptima1_ch1_io_deinit**(void)

clear mux function of pin gptima1 of channel 1

void **gptima1_ch2_io_init**(uint8_t *pin*, bool *output*, uint8_t *default_val*)

Set mux function of pin gptima1 of channel 2.

参数

- **pin** -- Specific GPIO pin
- **output** -- config pin output status
- **default_val** -- config pin output high level or low level

void **gptima1_ch2_io_deinit**(void)

clear mux function of pin gptima1 of channel 2

void **gptima1_ch3_io_init**(uint8_t *pin*, bool *output*, uint8_t *default_val*)

Set mux function of pin gptima1 of channel 3.

参数

- **pin** -- Specific GPIO pin
- **output** -- config pin output status
- **default_val** -- config pin output high level or low level

void **gptima1_ch3_io_deinit**(void)

clear mux function of pin gptima1 of channel 3

void **gptima1_ch4_io_init**(uint8_t *pin*, bool *output*, uint8_t *default_val*)

Set mux function of pin gptima1 of channel 4.

参数

- **pin** -- Specific GPIO pin
- **output** -- config pin output status
- **default_val** -- config pin output high level or low level

void **gptima1_ch4_io_deinit**(void)

clear mux function of pin gptima1 of channel 4

void **gptima1_etr_io_init**(uint8_t *pin*)

Set mux function of pin gptima1 of channel etr.

参数 **pin** -- Specific GPIO pin

void **gptima1_etr_io_deinit**(void)

clear mux function of pin gptima1 of etr

void **gptimb1_ch1_io_init**(uint8_t *pin*, bool *output*, uint8_t *default_val*)

Set mux function of pin gptima1 of channel 1.

参数

- **pin** -- Specific GPIO pin

- **output** -- config pin output status
- **default_val** -- config pin output high level or low level

void **gptimb1_ch1_io_deinit**(void)

clear mux function of pin gptimb1 of channel 1

void **gptimb1_ch2_io_init**(uint8_t *pin*, bool *output*, uint8_t *default_val*)

Set mux function of pin gptima1 of channel 2.

参数

- **pin** -- Specific GPIO pin
- **output** -- config pin output status
- **default_val** -- config pin output high level or low level

void **gptimb1_ch2_io_deinit**(void)

clear mux function of pin gptimb1 of channel 2

void **gptimb1_ch3_io_init**(uint8_t *pin*, bool *output*, uint8_t *default_val*)

Set mux function of pin gptima1 of channel 3.

参数

- **pin** -- Specific GPIO pin
- **output** -- config pin output status
- **default_val** -- config pin output high level or low level

void **gptimb1_ch3_io_deinit**(void)

clear mux function of pin gptimb1 of channel 3

void **gptimb1_ch4_io_init**(uint8_t *pin*, bool *output*, uint8_t *default_val*)

Set mux function of pin gptima1 of channel 4.

参数

- **pin** -- Specific GPIO pin
- **output** -- config pin output status
- **default_val** -- config pin output high level or low level

void **gptimb1_ch4_io_deinit**(void)

clear mux function of pin gptimb1 of channel 4

void **gptimb1_etr_io_init**(uint8_t *pin*)

Set mux function of pin gptimb1 of etr.

参数 **pin** -- Specific GPIO pin

void **gptimb1_etr_io_deinit**(void)

clear mux function of pin gptimb1 of etr

void **gptimc1_ch1_io_init**(uint8_t *pin*, bool *output*, uint8_t *default_val*)

Set mux function of pin gptima1 of channel 1.

参数

- **pin** -- Specific GPIO pin
- **output** -- config pin output status
- **default_val** -- config pin output high level or low level

void **gptimc1_ch1_io_deinit**(void)

clear mux function of pin gptimc1 of channel 1

void **gptimc1_ch1n_io_init**(uint8_t *pin*)

Set mux function of pin gptimc1 of channel 1.

参数 **pin** -- Specific GPIO pin

void **gptimc1_ch1n_io_deinit**(void)

clear mux function of pin gptimc1 of channel 1

void **gptimc1_ch2_io_init**(uint8_t *pin*, bool *output*, uint8_t *default_val*)

Set mux function of pin gptimc1 of channel 2.

参数

- **pin** -- Specific GPIO pin
- **output** -- config pin output status
- **default_val** -- config pin output high level or low level

void **gptimc1_ch2_io_deinit**(void)

clear mux function of pin gptimc1 of channel 2

void **gptimc1_bk_io_init**(uint8_t *pin*)

Set mux function of pin gptimc1 of bk.

参数 **pin** -- Specific GPIO pin

void **gptimc1_bk_io_deinit**(void)

clear mux function of pin gptimc1 of bk

void **adc12b_in0_io_init**(void)

Set mux function of pin gadc12b of in0.

void **adc12b_in0_io_deinit**(void)

clear mux function of pin gadc12b of in0

void **adc12b_in1_io_init**(void)

Set mux function of pin adc12b of in1.

void **adc12b_in1_io_deinit**(void)
clear mux function of pin adc12b of in1

void **adc12b_in2_io_init**(void)
Set mux function of pin adc12b of in2.

void **adc12b_in2_io_deinit**(void)
clear mux function of pin adc12b of in2

void **adc12b_in3_io_init**(void)
Set mux function of pin adc12b of in3.

void **adc12b_in3_io_deinit**(void)
clear mux function of pin adc12b of in3

void **adc12b_in4_io_init**(void)
Set mux function of pin adc12b of in4.

void **adc12b_in4_io_deinit**(void)
clear mux function of pin adc12b of in4

void **adc12b_in5_io_init**(void)
Set mux function of pin adc12b of in5.

void **adc12b_in5_io_deinit**(void)
clear mux function of pin adc12b of in5

void **adc12b_in6_io_init**(void)
Set mux function of pin adc12b of in6.

void **adc12b_in6_io_deinit**(void)
clear mux function of pin adc12b of in6

void **adc12b_in7_io_init**(void)
Set mux function of pin adc12b of in7.

void **adc12b_in7_io_deinit**(void)
clear mux function of pin adc12b of in7

void **adc12b_in8_io_init**(void)
Set mux function of pin adc12b of in8.

void **adc12b_in8_io_deinit**(void)
clear mux function of pin adc12b of in8

void **pdm_clk_io_init**(uint8_t *pin*)
configure gpio pin as pdm_clk

参数 **pin** -- Specific GPIO pin

void **pdm_clk_io_deinit**(void)
clear gpio of pdm_clk

void **pdm_data0_io_init**(uint8_t *pin*)
configure gpio of pdm_data0

参数 **pin** -- Specific GPIO pin

void **pdm_data0_io_deinit**(void)
clear gpio pin of pdm_data0

void **pdm_data1_io_init**(uint8_t *pin*)
configure gpio pin as pdm_data1

参数 **pin** -- Specific GPIO pin

void **pdm_data1_io_deinit**(void)
disable pin mux function from pdm_data1

struct gpio_pin_t

Public Members

uint8_t **num**
gpio pin

uint8_t **port**
gpio port

3.2.3 LSCRYPT API

Enums

enum aes_key_type
AES Key Size.

Values:

enumerator AES_KEY_128

enumerator AES_KEY_192

enumerator AES_KEY_256

Functions

HAL_StatusTypeDef **HAL_LSCRYPT_Init**(void)

LSCRYPT Initialize.

返回 status

HAL_StatusTypeDef **HAL_LSCRYPT_DeInit**(void)

LSCRYPT De-Initialize.

返回 status

HAL_StatusTypeDef **HAL_LSCRYPT_AES_Key_Config**(**const** uint32_t **key*, **enum** *aes_key_type key_size*)

LSCRYPT AES Key Config.

参数

- **key** -- **[in]** Buffer pointer containing the key, must be 4 bytes aligned
- **key_size** -- **[in]** Key Size

返回 status

HAL_StatusTypeDef **HAL_LSCRYPT_AES_ECB_Encrypt**(**const** uint8_t **plaintext*, uint32_t *length*, uint8_t **ciphertext*)

LSCRYPT AES ECB Encrypt (Block Mode)

参数

- **plaintext** -- **[in]** Input data
- **length** -- **[in]** Input data Length
- **ciphertext** -- **[out]** Output data

返回 status

HAL_StatusTypeDef **HAL_LSCRYPT_AES_ECB_Decrypt**(**const** uint8_t **ciphertext*, uint32_t *length*, uint8_t **plaintext*)

LSCRYPT AES ECB Decrypt (Block Mode)

参数

- **ciphertext** -- **[in]** Input data
- **length** -- **[in]** Input data Length
- **plaintext** -- **[out]** Output data

返回 status

HAL_StatusTypeDef **HAL_LSCRYPT_AES_CBC_Encrypt**(**const** uint32_t *iv*[4], **const** uint8_t **plaintext*, uint32_t *length*, uint8_t **ciphertext*)

LSCRYPT AES CBC Encrypt (Block Mode)

参数

- **iv** -- **[in]** Initial vector
- **plaintext** -- **[in]** Input data
- **length** -- **[in]** Input data Length
- **ciphertext** -- **[out]** Output data

返回 status

HAL_StatusTypeDef **HAL_LSCRYPT_AES_CBC_Decrypt**(**const** uint32_t *iv*[4], **const** uint8_t **ciphertext*, uint32_t *length*, uint8_t **plaintext*)

LSCRYPT AES CBC Decrypt (Block Mode)

参数

- **iv** -- **[in]** Initial vector
- **ciphertext** -- **[in]** Input data
- **length** -- **[in]** Input data Length
- **plaintext** -- **[out]** Output data

返回 status

HAL_StatusTypeDef **HAL_LSCRYPT_AES_ECB_Encrypt_IT**(**const** uint8_t **plaintext*, uint32_t *length*, uint8_t **ciphertext*)

LSCRYPT AES ECB Encrypt (Interrupt Mode)

参数

- **plaintext** -- **[in]** Input data
- **length** -- **[in]** Input data Length
- **ciphertext** -- **[out]** Output data

返回 status

HAL_StatusTypeDef **HAL_LSCRYPT_AES_ECB_Decrypt_IT**(**const** uint8_t **ciphertext*, uint32_t *length*, uint8_t **plaintext*)

LSCRYPT AES ECB Decrypt (Interrupt Mode)

参数

- **ciphertext** -- **[in]** Input data
- **length** -- **[in]** Input data Length
- **plaintext** -- **[out]** Output data

返回 status

HAL_StatusTypeDef **HAL_LSCRYPT_AES_CBC_Encrypt_IT**(**const** uint32_t *iv*[4], **const** uint8_t **plaintext*, uint32_t *length*, uint8_t **ciphertext*)

LSCRYPT AES CBC Encrypt (Interrupt Mode)

参数

- **iv** -- **[in]** Initial vector
- **plaintext** -- **[in]** Input data
- **length** -- **[in]** Input data Length
- **ciphertext** -- **[out]** Output data

返回 status

HAL_StatusTypeDef **HAL_LSCRYPT_AES_CBC_Decrypt_IT**(**const** uint32_t *iv*[4], **const** uint8_t **ciphertext*, uint32_t *length*, uint8_t **plaintext*)

LSCRYPT AES CBC Decrypt (Interrupt Mode)

参数

- **iv** -- **[in]** Initial vector
- **ciphertext** -- **[in]** Input data
- **length** -- **[in]** Input data Length
- **plaintext** -- **[out]** Output data

返回 status

void **HAL_LSCRYPT_AES_Complete_Callback**(bool *Encrypt*, bool *CBC*)

Callback function that will be invoked in the interrupt context when AES operation is complete. Overwrite this function to get notification of completion of AES operation.

参数

- **Encrypt** -- the complete operation is Encryption if ture else Decryption
- **CBC** -- the block cipher mode of the complete operation is CBC if true else ECB

void **HAL_LSCRYPT_IRQHandler**(void)

3.2.4 LSSSI API

Typedefs

typedef struct *__SSI_InitTypeDef* **SSI_InitTypeDef**

SSI Initialization Parameters Typedef.

typedef struct *__SSI_HandleTypeDef* **SSI_HandleTypeDef**

SSI Handle Typedef.

Enums

enum Data_Frame_Size

Data Frame Size enumeration.

Values:

enumerator DFS_32_4_bits

enumerator DFS_32_5_bits

enumerator DFS_32_6_bits

enumerator DFS_32_7_bits

enumerator DFS_32_8_bits

enumerator DFS_32_9_bits

enumerator DFS_32_10_bits

enumerator DFS_32_11_bits

enumerator DFS_32_12_bits

enumerator DFS_32_13_bits

enumerator DFS_32_14_bits

enumerator DFS_32_15_bits

enumerator DFS_32_16_bits

enumerator DFS_32_17_bits

enumerator DFS_32_18_bits

enumerator DFS_32_19_bits

enumerator DFS_32_20_bits

enumerator DFS_32_21_bits

enumerator DFS_32_22_bits

enumerator DFS_32_23_bits

enumerator DFS_32_24_bits

enumerator DFS_32_25_bits

enumerator DFS_32_26_bits

enumerator DFS_32_27_bits

enumerator DFS_32_28_bits

enumerator DFS_32_29_bits

enumerator DFS_32_30_bits

enumerator DFS_32_31_bits

enumerator DFS_32_32_bits

enum Control_Frame_Size

Control Frame Size enumeration.

Values:

enumerator Control_Word_1_bit

enumerator Control_Word_2_bit

enumerator Control_Word_3_bit

enumerator Control_Word_4_bit

enumerator Control_Word_5_bit

enumerator Control_Word_6_bit

enumerator Control_Word_7_bit

enumerator Control_Word_8_bit

enumerator Control_Word_9_bit

enumerator Control_Word_10_bit

enumerator Control_Word_11_bit

enumerator Control_Word_12_bit

enumerator Control_Word_13_bit

enumerator Control_Word_14_bit

enumerator Control_Word_15_bit

enumerator Control_Word_16_bit

enum Clock_Phase

Clock Phase enumeration.

Values:

enumerator Inactive_Low

enumerator Inactive_High

enum Clock_Polarity

Clock Polarity enumeration.

Values:

enumerator SCLK_Toggle_In_Middle

enumerator SCLK_Toggle_At_Start

enum Frame_Format

Frame Format enumeration.

Values:

enumerator Motorola_SPI

enumerator Texas_Instruments_SSP

enumerator National_Semiconductors_Microwire

Functions

HAL_StatusTypeDef **HAL_SSI_Init**(*SSI_HandleTypeDef* *hssi)

SSI Initialize.

返回 Status

HAL_StatusTypeDef **HAL_SSI_Deinit**(*SSI_HandleTypeDef* *hssi)

SSI De-Initialize.

返回 Status

HAL_StatusTypeDef **HAL_SSI_Transmit_IT**(*SSI_HandleTypeDef* *hssi, void *Data, uint16_t *Count*)

SSI Transmit (Interrupt Mode)

参数

- **hssi** -- **[in]** Handle of SSI
- **Data** -- **[in]** Buffer pointer for TX
- **Count** -- **[in]** Number of data frame in units of *Data_Frame_Size*

返回 Status

void **HAL_SSI_TxCpltCallback**(*SSI_HandleTypeDef* *hssi)

Callback function that will be invoked in the interrupt context when SSI TX (Interrupt Mode) is complete Overwrite this function to get notification of completion of SSI TX.

参数 **hssi** -- Handle of SSI

HAL_StatusTypeDef **HAL_SSI_Receive_IT**(*SSI_HandleTypeDef* *hssi, void *Data, uint16_t *Count*)

SSI Receive (Interrupt Mode)

参数

- **hssi** -- **[in]** Handle of SSI
- **Data** -- **[out]** Buffer pointer for RX
- **Count** -- **[in]** Number of data frame in units of *Data_Frame_Size*

返回 Status

void **HAL_SSI_RxCpltCallback**(*SSI_HandleTypeDef *hssi*)

Callback function that will be invoked in the interrupt context when SSI RX (Interrupt Mode) is complete Overwrite this function to get notification of completion of SSI RX.

参数 **hssi** -- Handle of SSI

HAL_StatusTypeDef **HAL_SSI_TransmitReceive_IT**(*SSI_HandleTypeDef *hssi*, void *TX_Data, void *RX_Data, uint16_t *Count*)

SSI TransmitReceive (Interrupt Mode)

参数

- **hssi** -- **[in]** Handle of SSI
- **TX_Data** -- **[in]** Buffer pointer for TX
- **RX_Data** -- **[out]** Buffer pointer for RX
- **Count** -- **[in]** Number of data frame in units of *Data_Frame_Size*

返回 Status

void **HAL_SSI_TxRxCpltCallback**(*SSI_HandleTypeDef *hssi*)

Callback function that will be invoked in the interrupt context when SSI TXRX (Interrupt Mode) is complete Overwrite this function to get notification of completion of SSI TXRX.

参数 **hssi** -- Handle of SSI

HAL_StatusTypeDef **HAL_SSI_Transmit_DMA**(*SSI_HandleTypeDef *hssi*, void *Data, uint16_t *Count*)

SSI Transmit (DMA Mode)

参数

- **hssi** -- **[in]** Handle of SSI
- **Data** -- **[in]** Buffer pointer for TX
- **Count** -- **[in]** Number of data frame in units of *Data_Frame_Size*

返回 Status

void **HAL_SSI_TxDMACpltCallback**(*SSI_HandleTypeDef *hssi*)

Callback function that will be invoked in the interrupt context when SSI TX (DMA Mode) is complete Overwrite this function to get notification of completion of SSI TX.

参数 **hssi** --

HAL_StatusTypeDef **HAL_SSI_Receive_DMA**(*SSI_HandleTypeDef *hssi*, void *Data, uint16_t *Count*)

SSI Receive (DMA Mode)

参数

- **hssi** -- **[in]** Handle of SSI

- **Data** -- **[out]** Buffer pointer for RX
- **Count** -- **[in]** Number of data frame in units of *Data_Frame_Size*

返回 Status

void **HAL_SSI_RxDMACpltCallback**(*SSI_HandleTypeDef* *hssi)

Callback function that will be invoked in the interrupt context when SSI RX (DMA Mode) is complete Overwrite this function to get notification of completion of SSI RX.

参数 **hssi** -- Handle of SSI

HAL_StatusTypeDef **HAL_SSI_TransmitReceive_DMA**(*SSI_HandleTypeDef* *hssi, void *TX_Data, void *RX_Data, uint16_t *Count*)

SSI TransmitReceive (Interrupt Mode)

参数

- **hssi** -- **[in]** Handle of SSI
- **TX_Data** -- **[in]** Buffer pointer for TX
- **RX_Data** -- **[out]** Buffer pointer for RX
- **Count** -- **[in]** Number of data frame in units of *Data_Frame_Size*

返回 Status

void **HAL_SSI_TxRxDMACpltCallback**(*SSI_HandleTypeDef* *hssi)

Callback function that will be invoked in the interrupt context when SSI TXRX (DMA Mode) is complete Overwrite this function to get notification of completion of SSI TXRX.

参数 **hssi** -- Handle of SSI

void **HAL_SSI_IRQHandler**(*SSI_HandleTypeDef* *hssi)

struct SSI_DMA_Env

#include <lsssi.h> SSI DMA TX RX Environment.

Public Members

uint8_t **DMA_Channel**

struct SSI_Interrupt_Env

#include <lsssi.h> SSI Interrupt TX RX Environment.

Public Members

uint8_t *__Data__

uint16_t __Count__

__struct ssi_ctrl__

#include <lsssi.h> SSI Control Structure.

Public Members

uint32_t __reserved0__

uint32_t __frame_format__

Frame_Format

uint32_t __cph__

Clock_Phase

uint32_t __cpol__

Clock_Polarity

uint32_t __reserved1__

uint32_t __control_frame_size__

Control_Frame_Size

uint32_t __data_frame_size__

Data_Frame_Size

__struct __SSI_InitTypeDef__

#include <lsssi.h> SSI Initialization Parameters Typedef.

Public Members

__struct__ *ssi_ctrl* __ctrl__

Control Parameters

uint16_t __clk_div__

Clock Dividing Coefficient

uint8_t __rxsample_dly__

RX Sampling Delay

__struct __SSI_HandleTypeDef__

#include <lsssi.h> SSI Handle Typedef.

Public Members

reg_ssi_t *__REG__

Register Base Pointer

SSI_InitTypeDef __Init__

Initialization Parameters

void *__DMAC_Instance__

DMA Controller Handle Pointer

__struct__ *SSI_DMA_Env* __DMA__

DMA Env

__struct__ *SSI_Interrupt_Env* __Interrupt__

Interrupt Env

__union__ *__SSI_HandleTypeDef*::**[anonymous]** __Tx_Env__

__union__ *__SSI_HandleTypeDef*::**[anonymous]** __Rx_Env__

Tx Rx Environment

uint8_t __Hardware_CS_Mask__

Hardware CS Bit Mask, 0 for Software CS

3.2.5 LSPDM API

Defines

__LSPDM__

LSPDM Macro for Register Access.

__PDM_FIR_COEF_8KHZ__

Select the PDM filter controller at a sampling rate of 8kHz.

__PDM_FIR_COEF_16KHZ__

Select the PDM filter controller at a sampling rate of 16kHz.

__PDM_CLK_RATIO__(*kHz*)

Calculate the PDM clock rate value.

注解： eg: if pdm clock is 1.024MHz, configure khz is 1024.

参数

- __kHz__ -- **[in]** Set the PDM clock rate value Unit: khz

PDM_SAMPLE_RATE(*Clk_kHz*, *Sample_Rate_Hz*)

Calculate the PDM sample rate value.

注解: eg: if pdm sample rare is 16KHz and pdm clock is 1.024MHz, configure the Clk_kHz is 1024, configure the Sample_Rate_Hz is 16000.

参数

- **Clk_kHz** -- **[in]** Set the PDM clock rate value Unit: khz
- **Sample_Rate_Hz** -- **[in]** Set the PDM sample rate value Unit: hz

Typedefs

typedef enum *__PDM_MODE_TypeDef* **PDM_MODE_TypeDef**

PDM mode selection enumeration typedef.

typedef struct *__PDM_CFG_TypeDef* **PDM_CFG_TypeDef**

PDM Configuration Parameters Typedef.

typedef struct *__PDM_Init_TypeDef* **PDM_Init_TypeDef**

PDM Initialization Parameters Typedef.

typedef struct *__PDM_HandleTypeDef* **PDM_HandleTypeDef**

PDM Handle Typedef.

Enums

enum __PDM_MODE_TypeDef

PDM mode selection enumeration typedef.

Values:

enumerator PDM_MODE_Mono

enumerator PDM_MODE_Stereo

Functions

HAL_StatusTypeDef **HAL_PDM_Init**(*PDM_HandleTypeDef* **hpdm*, *PDM_Init_TypeDef* **Init*)

PDM Initialize.

参数

- **hpdm** -- **[in]** Handle of PDM
- **Init** -- **[in]** Initialization Parameters

返回 Status

HAL_StatusTypeDef **HAL_PDM_DeInit**(*PDM_HandleTypeDef* *hpdm)

PDM DeInitialize.

参数 **hpdm** -- **[in]** Handle of PDM

返回 Status

HAL_StatusTypeDef **HAL_PDM_Transfer_Config_IT**(*PDM_HandleTypeDef* *hpdm, uint16_t *pFrameBuffer0, uint16_t *pFrameBuffer1, uint16_t FrameNum)

PDM Transfer Data Configuration (Interrupt Mode)

参数

- **hpdm** -- **[in]** Handle of PDM
- **pFrameBuffer0** -- **[in]** The Buffer pointer to accept PDM data
- **pFrameBuffer1** -- **[in]** The Buffer pointer to accept PDM data in stereo mode. Other mode is NULL.
- **FrameNum** -- **[in]** Accept the size of PDM data

返回 Status

void **HAL_PDM_CpltCallback**(*PDM_HandleTypeDef* *hpdm)

Callback function that will be invoked in the interrupt context when PDM Transfer (Interrupt Mode) is complete

参数 **hpdm** -- Handle of PDM

HAL_StatusTypeDef **HAL_PDM_Transfer_Config_DMA**(*PDM_HandleTypeDef* *hpdm, uint16_t *pFrameBuffer0, uint16_t *pFrameBuffer1, uint16_t FrameNum)

PDM Transfer Data Configuration (DMA Mode)

参数

- **hpdm** -- **[in]** Handle of PDM
- **pFrameBuffer0** -- **[in]** The Buffer pointer to accept PDM data
- **pFrameBuffer1** -- **[in]** The Buffer pointer to accept PDM data in stereo mode. Other mode is NULL.
- **FrameNum** -- **[in]** Accept the size of PDM data

返回 Status

HAL_StatusTypeDef **HAL_PDM_PingPong_Transfer_Config_DMA**(*PDM_HandleTypeDef* *hpdm, **struct** *PDM_PingPong_Bufptr* *CH0_Buf, **struct** *PDM_PingPong_Bufptr* *CH1_Buf, uint16_t FrameNum)

PDM Transfer Data Configuration (DMA PingPong Mode)

参数

- **hpdm** -- **[in]** Handle of PDM
- **CH0_Buf** -- **[in]** The Buffer pointer to accept PDM data *PDM_PingPong_Bufptr*
- **CH1_Buf** -- **[in]** The Buffer pointer to accept PDM data in stereo mode. Other mode is NULL. *PDM_PingPong_Bufptr*
- **FrameNum** -- **[in]** Accept the size of PDM data

返回 Status

void **HAL_PDM_DMA_CpltCallback**(*PDM_HandleTypeDef* *hpdm, uint8_t *buf_idx*)

Callback function that will be invoked in the interrupt context when PDM Transfer (DMA Mode) is complete

参数

- **hpdm** -- Handle of PDM
- **buf_idx** -- Transfer data to complete IDX

HAL_StatusTypeDef **HAL_PDM_Start**(*PDM_HandleTypeDef* *hpdm)

PDM peripheral start working

参数 **hpdm** -- **[in]** Handle of PDM

HAL_StatusTypeDef **HAL_PDM_Stop**(*PDM_HandleTypeDef* *hpdm)

PDM peripheral stop working

参数 **hpdm** -- **[in]** Handle of PDM

void **HAL_PDM_IRQHandler**(*PDM_HandleTypeDef* *hpdm)

PDM interrupt handler function

参数 **hpdm** -- **[in]** Handle of PDM

Variables

const uint32_t **fir_coef_8khz**[64]

const uint32_t **fir_coef_16khz**[64]

struct __PDM_CFG_TypeDef

#include <*lspdm.h*> PDM Configuration Parameters Typedef.

Public Members

uint8_t **clk_ratio**

PDM_CLK_RATIO

uint8_t **cap_delay**

Configure the value of the PDM capture delay. data capture after this value cycles of 128Mhz

uint8_t **sample_rate**

PDM_SAMPLE_RATE

uint8_t **data_gain**

Configuration the value of the PDM gain adjustment

struct __PDM_Init_TypeDef

#include <lspdm.h> PDM Initialization Parameters Typedef.

Public Members

const struct pdm_fir ***fir**

Select the PDM filter controller

PDM_CFG_TypeDef **cfg**

PDM_CFG_TypeDef

PDM_MODE_TypeDef **mode**

PDM_MODE_TypeDef

struct PDM_DMA_Env

#include <lspdm.h> PDM DMA Environment.

Public Members

uint32_t **PingPong_Ctrl_Data**

Configuration the data control register for DMA ping-pong mode

uint8_t **Channel**[2]

Configure the number of DMA channels

bool **Channel_Done**[2]

DMA channel completion flag

struct PDM_Interrupt_Env

#include <lspdm.h> PDM Interrupt Environment.

Public Members

uint16_t *__pFrameBuffer__[2]

An pointer array to accept PDM data

uint16_t __FrameNum__

Configure the size to accept PDM data

__struct PDM_PingPong_Bufptr__

#include <*lspdm.h*> PDM PingPong Buffer Pointer Structure.

Public Members

uint16_t *__Bufptr__[2]

Configure the pointer array to accept PDM data

__struct __PDM_HandleTypeDef__

#include <*lspdm.h*> PDM Handle Typedef.

Public Members

reg_pdm_t *__Instance__

PDM registers base address

void *__DMAC_Instance__

Select the initialized DMA object

__struct__ *PDM_DMA_Env* __DMA__

DMA Env

__struct__ *PDM_Interrupt_Env* __Interrupt__

Interrupt Env

__union__ *__PDM_HandleTypeDef*::**[anonymous]** __Env__

Select Environment: DMA or Interrupt Environment

3.2.6 LSRTC API

Enums

__enum RTC_CLK_SEL__

RTC clock source enumeration definition.

Values:

__enumerator RTC_CKSEL_LSE__

enumerator RTC_CKSEL_LSI

Functions

void **HAL_RTC_IRQHandler**(void)

RTC IRQ handler.

void **HAL_RTC_Init**(uint8_t *cksel*)

RTC initialization function.

参数 **cksel** -- **[in]** clock source selection

void **HAL_RTC_DeInit**(void)

RTC de-initialization function.

void **RTC_CalendarSet**(*calendar_cal_t* *calendar_cal, *calendar_time_t* *calendar_time)

RTC calendar set function.

参数

- **calendar_cal** -- **[in]** parameter containing date/month/year infor to set
- **calendar_time** -- **[in]** parameter containing second/minute/hour/week infor to set

HAL_StatusTypeDef **RTC_CalendarGet**(*calendar_cal_t* *calendar_cal, *calendar_time_t* *calendar_time)

RTC calendar get function.

参数

- **calendar_cal** -- **[out]** returned date/month/year information
- **calendar_time** -- **[out]** returned second/minute/hour/week information

返回 status 0: success | others: error

void **RTC_wkuptime_set**(uint32_t *t_ms*)

RTC wakeup time set function.

参数 **t_ms** -- **[in]** wakeup time(the unit is millisecond). The accuracy depends on the real RC clock.

void **rtc_wkup_callback**(void)

RTC wakeup callback function(in LP0 mode)

struct calendar_cal_t

#include <lsrtc.h> structure definition of calendar date & month & year.

Public Members

uint32_t **date**

date

uint32_t **mon**

month

uint32_t **year**

year

struct calendar_time_t

#include <lsrtc.h> structure definition of calendar second & minute & hour & week.

Public Members

uint32_t **sec**

second

uint32_t **min**

minute

uint32_t **hour**

hour

uint32_t **week**

week

3.2.7 UART API

Defines

UART_CLOCK

UART_BUADRATE_ENUM_GEN(*BAUD*)

Baud rate calculation.

Typedefs

typedef struct *__UART_HandleTypeDef* **UART_HandleTypeDef**

UART handle Structure definition.

Enums

enum app_uart_paritytype

UART parity type.

Values:

enumerator UART_NOPARITY

Parity diable

enumerator UART_ODDPARITY

Parity Odd

enumerator UART_EVENPARITY

Parity Even

enum app_uart_bytesizetype

UART bytesize type.

Values:

enumerator UART_BYTESIZE5

Byte size 5 bits

enumerator UART_BYTESIZE6

Byte size 6 bits

enumerator UART_BYTESIZE7

Byte size 7 bits

enumerator UART_BYTESIZE8

Byte size 8 bits

enum app_uart_stopbittype

UART stopbit type.

Values:

enumerator UART_STOPBITS1

Stop 1 bits

enumerator UART_STOPBITS2

Stop 2 bits

enum app_uart_baudrate_t

Baud rate setting.

Values:

enumerator UART_BAUDRATE_1200

enumerator UART_BAUDRATE_2400

enumerator UART_BAUDRATE_4800

enumerator UART_BAUDRATE_9600

enumerator UART_BAUDRATE_14400

enumerator UART_BAUDRATE_19200

enumerator UART_BAUDRATE_28800

enumerator UART_BAUDRATE_38400

enumerator UART_BAUDRATE_57600

enumerator UART_BAUDRATE_76800

enumerator UART_BAUDRATE_115200

enumerator UART_BAUDRATE_230400

enumerator UART_BAUDRATE_250000

enumerator UART_BAUDRATE_500000

enumerator UART_BAUDRATE_460800

enumerator UART_BAUDRATE_750000

enumerator UART_BAUDRATE_921600

enumerator UART_BAUDRATE_1000000

enumerator UART_BAUDRATE_2000000

enum HAL_UART_StateTypeDef

HAL UART State structures definition.

注解： HAL UART State value is a combination of 2 different substates: gState and RxState.

- gState contains UART state information related to global Handle management and also information related to Tx operations.

Values:

enumerator HAL_UART_STATE_RESET

Peripheral is not yet Initialized Value is allowed for gState and RxState

enumerator HAL_UART_STATE_READY

Peripheral Initialized and ready for use Value is allowed for gState and RxState

enumerator HAL_UART_STATE_BUSY

an internal process is ongoing Value is allowed for gState only

enumerator HAL_UART_STATE_BUSY_TX

Data Transmission process is ongoing Value is allowed for gState only

enumerator HAL_UART_STATE_BUSY_RX

Data Reception process is ongoing Value is allowed for RxState only

Functions

HAL_StatusTypeDef **HAL_UART_AutoBaudRate_Detect**(*UART_HandleTypeDef* *huart, uint8_t *mode*, uint8_t **detect_byte*)

void **HAL_UART_BaudRateDetectCpltCallback**(*UART_HandleTypeDef* *huart, uint8_t *detect_byte*)

HAL_StatusTypeDef **HAL_UART_AutoBaudRate_Detect_IT**(*UART_HandleTypeDef* *huart, uint8_t *mode*)

HAL_StatusTypeDef **HAL_UART_Transmit**(*UART_HandleTypeDef* *huart, uint8_t *pData, uint16_t *Size*, uint32_t *Timeout*)

Sends an amount of data in blocking mode.

注解:

参数

- **huart** -- Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **pData** -- Pointer to data buffer (uint8_t data elements).
- **Size** -- Amount of data elements (uint8_t) to be sent
- **Timeout** -- Timeout duration

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_UART_Receive**(*UART_HandleTypeDef* *huart, uint8_t *pData, uint16_t *Size*, uint32_t *Timeout*)

Receives an amount of data in blocking mode.

注解:

参数

- **huart** -- Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **pData** -- Pointer to data buffer (uint8_t data elements).

- **Size** -- Amount of data elements (uint8_t) to be received.
- **Timeout** -- Timeout duration

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_UART_Transmit_IT**(*UART_HandleTypeDef* *huart, uint8_t *pData, uint16_t *Size*)

Sends an amount of data in non blocking mode.

参数

- **huart** -- Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.int_t
- **pData** -- Pointer to data buffer (uint8_t data elements).
- **Size** -- Amount of data elements (uint8_t) to be sent

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_UART_Receive_IT**(*UART_HandleTypeDef* *huart, uint8_t *pData, uint16_t *Size*)

Receives an amount of data in non blocking mode.

参数

- **huart** -- Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **pData** -- Pointer to data buffer (uint8_t data elements).
- **Size** -- Amount of data elements (uint8_t) to be received.

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_UART_Transmit_DMA**(*UART_HandleTypeDef* *huart, uint8_t *pData, uint16_t *Size*)

Sends an amount of data in DMA mode.

参数

- **huart** -- Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **pData** -- Pointer to data buffer (uint_8 data elements).
- **Size** -- Amount of data elements (uint_8) to be sent.

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_UART_Receive_DMA**(*UART_HandleTypeDef* *huart, uint8_t *pData, uint16_t *Size*)

Receives an amount of data in DMA mode.

注解: When the UART parity is enabled (PCE = 1) the received data contains the parity bit.

参数

- **huart** -- Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **pData** -- Pointer to data buffer (uint_8 data elements).
- **Size** -- Amount of data elements (uint_8) to be received.

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_UART_Init**(*UART_HandleTypeDef* *huart)

Initializes the UART mode according to the specified parameters in the *UART_InitTypeDef* and create the associated handle.

参数 **huart** -- Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_UART_DeInit**(*UART_HandleTypeDef* *huart)

DeInitializes the UART peripheral.

参数 **huart** -- Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

返回 **HAL** -- status

void **HAL_UARTx_IRQHandler**(*UART_HandleTypeDef* *huart)

This function handles UARTx interrupt request.

参数 **huart** -- Pointer to a UARTx_HandleTypeDef structure that contains the configuration information for the specified UART module.

返回 **None** --

void **HAL_UART_TxCpltCallback**(*UART_HandleTypeDef* *huart)

Tx Transfer completed callbacks.

注解: This function needs to be implemented when the callback is needed, the HAL_UART_TxCpltCallback could be implemented in the user file

---

参数 **huart** -- Pointer to a UARTx_HandleTypeDef structure that contains the configuration information for the specified UART module.

返回 None --

void **HAL_UART_RxCpltCallback**(*UART_HandleTypeDef* *huart)

Rx Transfer completed callbacks.

注解: This function needs to be implemented when the callback is needed, the HAL_UART_RxCpltCallback could be implemented in the user file

---

参数 **huart** -- Pointer to a UARTx_HandleTypeDef structure that contains the configuration information for the specified UART module.

返回 None --

void **HAL_UART_DMA_TxCpltCallback**(*UART_HandleTypeDef* *huart)

DMA UART Tx Transfer completed callbacks.

注 解: This function needs to be implemented when the callback is needed, the HAL_UART_DMA_TxCpltCallback could be implemented in the user file

---

参数 **huart** -- Pointer to a UARTx_HandleTypeDef structure that contains the configuration information for the specified UART module.

返回 None --

void **HAL_UART_DMA_RxCpltCallback**(*UART_HandleTypeDef* *huart)

DMA UART Rx Transfer completed callbacks.

注 解: This function needs to be implemented when the callback is needed, the HAL_UART_DMA_RxCpltCallback could be implemented in the user file

---

参数 **huart** -- Pointer to a UARTx_HandleTypeDef structure that contains the configuration information for the specified UART module.

返回 None --

struct UART_InitTypeDef

Public Members

app_uart_baudrate_t **BaudRate**

uint8_t **WordLength**

config uart byte size This parameter can be a value of *app_uart_bytesizetype*

uint8_t **StopBits**

config uart stop bits. This parameter can be a value of *app_uart_stopbittype*

uint8_t **Parity**

config uart parity type. This parameter can be a value of *app_uart_paritytype*

uint8_t **MSBEN**

config uart msb or lsb

uint8_t **HwFlowCtl**

Specifies whether the hardware flow control mode is enabled or disabled.

struct UartDMAEnv

#include <*lsuart.h*> UART DMA TX RX Environment.

Public Members

uint8_t **DMA_Channel**

struct UartInterruptEnv

#include <*lsuart.h*> UART Interrupt TX Environment.

Public Members

uint8_t ***pBuffPtr**

Pointer to UART Tx transfer Buffer

uint16_t **XferCount**

UART Tx Transfer Counter

struct __UART_HandleTypeDef

#include <*lsuart.h*> UART handle Structure definition.

Public Members

reg_uart_t *__UARTX__

UART registers base address

UART_InitTypeDef __Init__

UART communication parameters

void *__DMAC_Instance__

__struct__ *UartDMAEnv* __DMA__

__struct__ *UartInterruptEnv* __Interrupt__

__union__ *__UART_HandleTypeDef*::__[anonymous]__ __Tx_Env__

__union__ *__UART_HandleTypeDef*::__[anonymous]__ __Rx_Env__

HAL_UART_StateTypeDef __gState__

UART state information related to global Handle management and also related to Tx operations. This parameter can be a value of *HAL_UART_StateTypeDef*

HAL_UART_StateTypeDef __RxState__

UART state information related to Rx operations.

uint32_t __ErrorCode__

UART Error code

3.2.8 LSSPI API

Defines

__SPI2__

LSSPI Macro for Register Access.

__SPI_CLOCK__

SPI CLOCK.

__HAL_SPI_ERROR_NONE__

No error

__HAL_SPI_ERROR_MODF__

MODF error

__HAL_SPI_ERROR_CRC__

CRC error

__HAL_SPI_ERROR_OVR__

OVR error

HAL_SPI_ERROR_DMA
DMA transfer error

HAL_SPI_ERROR_FLAG
Error on RXNE/TXE/BSY Flag

HAL_SPI_ERROR_ABORT
Error during SPI Abort procedure

SPI_MODE_SLAVE
The mode is slave

SPI_MODE_MASTER
The mode is master

SPI_DIRECTION_2LINES
The direction mode is 2 LINES

SPI_DIRECTION_2LINES_RXONLY
The direction mode is 2 LINES and RXONLY

SPI_DIRECTION_1LINE
The direction mode is 1 LINE

SPI_DATASIZE_8BIT
The SPI Data Size is 8BIT

SPI_DATASIZE_16BIT
The SPI Data Size is 16BIT

SPI_POLARITY_LOW
The SPI Clock Polarity is LOW

SPI_POLARITY_HIGH
The SPI Clock Polarity is HIGH

SPI_PHASE_1EDGE
The SPI Clock Phase is 1 edge

SPI_PHASE_2EDGE
The SPI Clock Phase is 2 edge

SPI_NSS_SOFT
Software select the NSS pin

SPI_NSS_HARD_INPUT
Hard select the NSS as input pin

SPI_NSS_HARD_OUTPUT
Hard select the NSS as output pin

SPI_BAUDRATEPRESCALER_8
The BaudRate is 8 frequency of the SPI clock

SPI_BAUDRATEPRESCALER_16
The BaudRate is 16 frequency of the SPI clock

SPI_BAUDRATEPRESCALER_32
The BaudRate is 32 frequency of the SPI clock

SPI_BAUDRATEPRESCALER_64
The BaudRate is 64 frequency of the SPI clock

SPI_BAUDRATEPRESCALER_128
The BaudRate is 128 frequency of the SPI clock

SPI_BAUDRATEPRESCALER_256
The BaudRate is 256 frequency of the SPI clock

SPI_FIRSTBIT_MSB
SPI MSB Transmission

SPI_FIRSTBIT_LSB
SPI LSB Transmission

SPI_TIMODE_DISABLE

SPI_CRCCALCULATION_DISABLE

SPI_CRCCALCULATION_ENABLE

SPI_IT_TXE
SPI TX Interrupt Definition

SPI_IT_RXNE
SPI RX Interrupt Definition

SPI_IT_ERR
SPI ERR Interrupt Definition

SPI_FLAG_RXNE

SPI_FLAG_TXE

SPI_FLAG_BSY

SPI_FLAG_CRCERR

SPI_FLAG_MODF

SPI_FLAG_OVR

SPI_FLAG_MASK

__HAL_SPI_RESET_HANDLE_STATE(*__HANDLE__*)

Reset SPI handle state.

__HAL_SPI_ENABLE_IT(*__HANDLE__*, *__INTERRUPT__*)

Enable the specified SPI interrupts.

__HAL_SPI_DISABLE_IT(*__HANDLE__*, *__INTERRUPT__*)

Disable the specified SPI interrupts.

__HAL_SPI_CLEAR_IF(*__HANDLE__*, *__INTERRUPT__*)

Clear the specified SPI interrupts IF.

__HAL_SPI_GET_IT_SOURCE(*__HANDLE__*, *__INTERRUPT__*)

Check whether the specified SPI interrupt source is enabled or not.

__HAL_SPI_GET_FLAG(*__HANDLE__*, *__FLAG__*)

Check whether the specified SPI flag is set or not.

__HAL_SPI_CLEAR_CRCERRFLAG(*__HANDLE__*)

Clear the SPI CRCERR pending flag.

__HAL_SPI_CLEAR_MODFFLAG(*__HANDLE__*)

Clear the SPI MODF pending flag.

__HAL_SPI_CLEAR_OVRFLAG(*__HANDLE__*)

Clear the SPI OVR pending flag.

__HAL_SPI_ENABLE(*__HANDLE__*)

Clear the SPI OVR pending flag.

Enable the SPI peripheral.

__HAL_SPI_DISABLE(*__HANDLE__*)

Disable the SPI peripheral.

SPI_INVALID_CRC_ERROR

SPI_VALID_CRC_ERROR

SPI_1LINE_TX(*__HANDLE__*)

Set the SPI transmit-only mode.

SPI_1LINE_RX(*__HANDLE__*)

Set the SPI receive-only mode.

SPI_RESET_CRC(*__HANDLE__*)

Reset the CRC calculation of the SPI.

SPI_CHECK_FLAG(*__SR__*, *__FLAG__*)

Check whether the specified SPI flag is set or not.

SPI_CHECK_IT_SOURCE(*__IVS__*, *__INTERRUPT__*)

Check whether the specified SPI Interrupt is set or not.

IS_SPI_MODE(*__MODE__*)

Checks if SPI Mode parameter is in allowed range.

IS_SPI_DIRECTION(*__MODE__*)

Checks if SPI Direction Mode parameter is in allowed range.

IS_SPI_DIRECTION_2LINES(*__MODE__*)

Checks if SPI Direction Mode parameter is 2 lines.

IS_SPI_DIRECTION_2LINES_OR_1LINE(*__MODE__*)

Checks if SPI Direction Mode parameter is 1 or 2 lines.

IS_SPI_DATASIZE(*__DATASIZE__*)

Checks if SPI Data Size parameter is in allowed range.

IS_SPI_CPOL(*__CPOL__*)

Checks if SPI Serial clock steady state parameter is in allowed range.

IS_SPI_CPHA(*__CPHA__*)

Checks if SPI Clock Phase parameter is in allowed range.

IS_SPI_NSS(*__NSS__*)

Checks if SPI Slave Select parameter is in allowed range.

IS_SPI_BAUDRATE_PRESCALER(*__PRESCALER__*)

Checks if SPI Baudrate prescaler parameter is in allowed range.

IS_SPI_FIRST_BIT(*__BIT__*)

Checks if SPI MSB LSB transmission parameter is in allowed range.

IS_SPI_TIMODE(*__MODE__*)

Checks if SPI TI mode parameter is disabled.

IS_SPI_CRC_CALCULATION(*__CALCULATION__*)

Checks if SPI CRC calculation enabled state is in allowed range.

IS_SPI_CRC_POLYNOMIAL(*__POLYNOMIAL__*)

Checks if SPI polynomial value to be used for the CRC calculation, is in allowed range.

IS_SPI_DMA_HANDLE(*__HANDLE__*)

Checks if DMA handle is valid.

Typedefs

typedef struct *__SPI_HandleTypeDef* **SPI_HandleTypeDef**

SPI handle Structure definition.

Enums

enum HAL_SPI_StateTypeDef

HAL SPI State structure definition.

Values:

enumerator HAL_SPI_STATE_RESET

Peripheral not Initialized

enumerator HAL_SPI_STATE_READY

Peripheral Initialized and ready for use

enumerator HAL_SPI_STATE_BUSY

an internal process is ongoing

enumerator HAL_SPI_STATE_BUSY_TX

Data Transmission process is ongoing

enumerator HAL_SPI_STATE_BUSY_RX

Data Reception process is ongoing

enumerator HAL_SPI_STATE_BUSY_TX_RX

Data Transmission and Reception process is ongoing

enumerator HAL_SPI_STATE_ERROR

SPI error state

enumerator HAL_SPI_STATE_ABORT

SPI abort is ongoing

Functions

HAL_StatusTypeDef **HAL_SPI_Init**(*SPI_HandleTypeDef* **hspi*)

Initialize the SPI according to the specified parameters in the *SPI_InitTypeDef* and initialize the associated handle.

参数 **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_SPI_DeInit**(*SPI_HandleTypeDef* **hspi*)

De-Initialize the SPI peripheral.

参数 **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

返回 **HAL** -- status

void **HAL_SPI_MspInit**(*SPI_HandleTypeDef *hspi*)

Initialize the SPI MSP.

参数 **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

返回 **None** --

void **HAL_SPI_MspDeInit**(*SPI_HandleTypeDef *hspi*)

De-Initialize the SPI MSP.

参数 **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

返回 **None** --

HAL_StatusTypeDef **HAL_SPI_Transmit**(*SPI_HandleTypeDef *hspi*, uint8_t *pData, uint16_t *Size*, uint32_t *Timeout*)

Transmit an amount of data in blocking mode.

参数

- **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData** -- pointer to data buffer
- **Size** -- amount of data to be sent
- **Timeout** -- Timeout duration

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_SPI_Receive**(*SPI_HandleTypeDef *hspi*, uint8_t *pData, uint16_t *Size*, uint32_t *Timeout*)

Receive an amount of data in blocking mode.

参数

- **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData** -- pointer to data buffer
- **Size** -- amount of data to be received
- **Timeout** -- Timeout duration

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_SPI_TransmitReceive**(*SPI_HandleTypeDef* *hspi, uint8_t *pTxData, uint8_t *pRxData, uint16_t Size, uint32_t Timeout)

Transmit and Receive an amount of data in blocking mode.

参数

- **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData** -- pointer to transmission data buffer
- **pRxData** -- pointer to reception data buffer
- **Size** -- amount of data to be sent and received
- **Timeout** -- Timeout duration

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_SPI_Transmit_IT**(*SPI_HandleTypeDef* *hspi, uint8_t *pData, uint16_t Size)

Transmit an amount of data in non-blocking mode with Interrupt.

参数

- **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData** -- pointer to data buffer
- **Size** -- amount of data to be sent

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_SPI_Receive_IT**(*SPI_HandleTypeDef* *hspi, uint8_t *pData, uint16_t Size)

Receive an amount of data in non-blocking mode with Interrupt.

参数

- **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData** -- pointer to data buffer
- **Size** -- amount of data to be sent

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_SPI_TransmitReceive_IT**(*SPI_HandleTypeDef* *hspi, uint8_t *pTxData, uint8_t *pRxData, uint16_t Size)

Transmit and Receive an amount of data in non-blocking mode with Interrupt.

参数

- **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

- **pTxData** -- pointer to transmission data buffer
- **pRxData** -- pointer to reception data buffer
- **Size** -- amount of data to be sent and received

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_SPI_Transmit_DMA**(*SPI_HandleTypeDef* *hspi, void *Data, uint16_t Count)

Transmit an amount of data in DMA mode.

参数

- **hspi** -- pointer to a SPI_HandleTypeDef structure that contains. the configuration information for SPI module.
- **Data** -- pointer to data buffer.
- **Count** -- amount of data to be sent

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_SPI_Receive_DMA**(*SPI_HandleTypeDef* *hspi, void *Data, uint16_t Count)

Receive an amount of data in DMA mode.

参数

- **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **Data** -- pointer to data buffer.
- **Count** -- amount of data to be received.

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_SPI_TransmitReceive_DMA**(*SPI_HandleTypeDef* *hspi, void *TX_Data, void *RX_Data, uint16_t Count)

Transmit and Receive an amount of data in DMA mode.

参数

- **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **TX_Data** -- pointer to transmission data buffer.
- **RX_Data** -- pointer to reception data buffer.
- **Count** -- amount of data to be sent and received.

返回 **HAL** -- status

void **HAL_SPI_IRQHandler**(*SPI_HandleTypeDef* *hspi)

This function handles SPI interrupt request.

参数 **hspi** -- Pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

返回 **None** --

void **HAL_SPI_TxCpltCallback**(*SPI_HandleTypeDef *hspi*)

Tx Transfer completed callback.

参数 **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

返回 **None** --

void **HAL_SPI_RxCpltCallback**(*SPI_HandleTypeDef *hspi*)

Rx Transfer completed callback.

参数 **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

返回 **None** --

void **HAL_SPI_TxRxCpltCallback**(*SPI_HandleTypeDef *hspi*)

Tx and Rx Transfer completed callback.

参数 **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

返回 **None** --

void **HAL_SPI_ErrorCallback**(*SPI_HandleTypeDef *hspi*)

SPI error callback.

参数 **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

返回 **None** --

void **HAL_SPI_TxDMACpltCallback**(*SPI_HandleTypeDef *hspi*)

Tx Transfer completed callback in DMA.

参数 **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

返回 **None** --

void **HAL_SPI_RxDMACpltCallback**(*SPI_HandleTypeDef *hspi*)

Rx Transfer completed callback in DMA.

参数 **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

返回 **None** --

void **HAL_SPI_TxRxDMACpltCallback**(*SPI_HandleTypeDef* *hspi)

Tx and Rx Transfer completed callback in DMA.

参数 **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

返回 **None** --

HAL_SPI_StateTypeDef **HAL_SPI_GetState**(*SPI_HandleTypeDef* *hspi)

Return the SPI handle state.

参数 **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

返回 **SPI** -- state

uint32_t **HAL_SPI_GetError**(*SPI_HandleTypeDef* *hspi)

Return the SPI error code.

参数 **hspi** -- pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

返回 **SPI** -- error code in bitmap format

struct SPI_InitTypeDef

#include <lsspi.h> SPI Configuration Structure definition.

Public Members

uint32_t **Mode**

Specifies the SPI operating mode. This parameter can be a value of SPI Mode

uint32_t **Direction**

Specifies the SPI bidirectional mode state. This parameter can be a value of SPI Direction Mode

uint32_t **DataSize**

Specifies the SPI data size. This parameter can be a value of SPI Data Size

uint32_t **CLKPolarity**

Specifies the serial clock steady state. This parameter can be a value of SPI Clock Polarity

uint32_t **CLKPhase**

Specifies the clock active edge for the bit capture. This parameter can be a value of SPI Clock Phase

uint32_t **NSS**

Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of SPI Slave Select Management

uint32_t **BaudRatePrescaler**

Specifies the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of SPI BaudRate Prescaler

注解: The communication clock is derived from the master clock. The slave clock does not need to be set.

uint32_t **FirstBit**

Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of SPI MSB LSB Transmission

uint32_t **TIMode**

Specifies if the TI mode is enabled or not. This parameter can be a value of SPI TI Mode

uint32_t **CRCCalculation**

Specifies if the CRC calculation is enabled or not. This parameter can be a value of SPI CRC Calculation

struct SPI_DMA_Env

#include <*lsspi.h*> SPI DMA Environment.

Public Members

uint8_t **DMA_Channel**

bool **DMA_EN**

struct __SPI_HandleTypeDef

#include <*lsspi.h*> SPI handle Structure definition.

Public Members

reg_spi_t ***Instance**

SPI registers base address

SPI_InitTypeDef **Init**

SPI communication parameters

uint8_t ***pTxBuffPtr**

Pointer to SPI Tx transfer Buffer

uint16_t **TxXferSize**

SPI Tx Transfer size

uint16_t **TxXferCount**

SPI Tx Transfer Counter

uint8_t ***pRxBuffPtr**

Pointer to SPI Rx transfer Buffer

uint16_t **RxXferSize**

SPI Rx Transfer size

uint16_t **RxXferCount**
SPI Rx Transfer Counter

void (***RxISR**) (**struct** *__SPI_HandleTypeDef* *hspi)
function pointer on Rx ISR

void (***TxISR**) (**struct** *__SPI_HandleTypeDef* *hspi)
function pointer on Tx ISR

void ***DMAC_Instance**

struct *SPI_DMA_Env* **DMA**

union *__SPI_HandleTypeDef*::**[anonymous]** **Tx_Env**

union *__SPI_HandleTypeDef*::**[anonymous]** **Rx_Env**

HAL_LockTypeDef **Lock**
Locking object

HAL_SPI_StateTypeDef **State**
SPI communication state

uint32_t **ErrorCode**
SPI Error code

3.2.9 LSI2C API

Defines

HAL_I2C_ERROR_NONE
No error

HAL_I2C_ERROR_BERR
BERR error

HAL_I2C_ERROR_ARLO
ARLO error

HAL_I2C_ERROR_NACKF
NACK error

HAL_I2C_ERROR_OVR
OVR error

HAL_I2C_ERROR_DMA
DMA transfer error

HAL_I2C_ERROR_TIMEOUT
Timeout Error

HAL_I2C_ERROR_SIZE

Size Management error

HAL_I2C_ERROR_DMA_PARAM

DMA Parameter Error

I2C_ADDRESSINGMODE_7BIT

7bits addressing mode

I2C_ADDRESSINGMODE_10BIT

10bits addressing mode

I2C_DUALADDRESS_DISABLE

Disable I2C dual addressing mode

I2C_DUALADDRESS_ENABLE

Enable I2C dual addressing mode

I2C_GENERALCALL_DISABLE

Disable I2C general call addressing mode

I2C_GENERALCALL_ENABLE

Enable I2C general call addressing mode

I2C_NOSTRETCH_DISABLE

Disable I2C nostretch mode.

I2C_NOSTRETCH_ENABLE

Disable I2C nostretch mode.

I2C_MEMADD_SIZE_8BIT

I2C_MEMADD_SIZE_16BIT

Typedefs

typedef struct *__I2C_HandleTypeDef* **I2C_HandleTypeDef**

I2C_handle_Structure_definition I2C handle Structure definition.

Functions

HAL_StatusTypeDef **HAL_I2C_Init**(*I2C_HandleTypeDef* *hi2c)

Initializes the I2C according to the specified parameters in the *I2C_InitTypeDef* and initialize the associated handle.

参数 **hi2c** -- Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

返回 HAL status returned HAL_OK/HAL_ERROR information

HAL_StatusTypeDef **HAL_I2C_DeInit**(*I2C_HandleTypeDef* *hi2c)

DeInitialize the I2C peripheral.

参数 **hi2c** -- Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

返回 HAL status returned HAL_OK/HAL_ERROR information

HAL_StatusTypeDef **HAL_I2C_Master_Transmit**(*I2C_HandleTypeDef* *hi2c, uint16_t *DevAddress*, uint8_t *pData, uint16_t *Size*, uint32_t *Timeout*)

Transmits in master mode an amount of data in blocking mode.(Blocking mode: Polling)

参数

- **hi2c** -- Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress** -- Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData** -- Pointer to data buffer
- **Size** -- Amount of data to be sent
- **Timeout** -- Timeout duration.note： when Timeout = HAL_MAX_DELAY，that means the timeout is not valid.

返回 HAL status returned HAL_BUSY/HAL_OK/HAL_ERROR information

HAL_StatusTypeDef **HAL_I2C_Master_Receive**(*I2C_HandleTypeDef* *hi2c, uint16_t *DevAddress*, uint8_t *pData, uint16_t *Size*, uint32_t *Timeout*)

Receives in master mode an amount of data in blocking mode.(Blocking mode: Polling)

参数

- **hi2c** -- Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress** -- Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData** -- Pointer to data buffer
- **Size** -- Amount of data to be sent
- **Timeout** -- Timeout duration。note： when Timeout = HAL_MAX_DELAY，that means the timeout is not valid.

返回 HAL status returned HAL_BUSY/HAL_OK/HAL_ERROR information

HAL_StatusTypeDef **HAL_I2C_Mem_Write**(*I2C_HandleTypeDef* *hi2c, uint16_t *DevAddress*, uint16_t *MemAddress*, uint16_t *MemAddSize*, uint8_t *pData, uint16_t *Size*, uint32_t *Timeout*)

Write an amount of data in blocking mode to a specific memory address.

参数

- **hi2c** -- Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress** -- Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress** -- Internal memory address
- **MemAddSize** -- Size of internal memory address
- **pData** -- Pointer to data buffer
- **Size** -- Amount of data to be sent
- **Timeout** -- Timeout duration

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_I2C_Mem_Read**(*I2C_HandleTypeDef* *hi2c, uint16_t *DevAddress*, uint16_t *MemAddress*, uint16_t *MemAddSize*, uint8_t *pData, uint16_t *Size*, uint32_t *Timeout*)

Read an amount of data in blocking mode from a specific memory address.

参数

- **hi2c** -- Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress** -- Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress** -- Internal memory address
- **MemAddSize** -- Size of internal memory address
- **pData** -- Pointer to data buffer
- **Size** -- Amount of data to be sent
- **Timeout** -- Timeout duration

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_I2C_IsDeviceReady**(*I2C_HandleTypeDef* *hi2c, uint16_t *DevAddress*, uint32_t *Trials*, uint32_t *Timeout*)

Checks if target device is ready for communication.

注解: This function is used with Memory devices

参数

- **hi2c** -- Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress** -- Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **Trials** -- Number of trials
- **Timeout** -- Timeout duration

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_I2C_Master_Transmit_IT**(*I2C_HandleTypeDef* *hi2c, uint16_t *DevAddress*, uint8_t *pData, uint16_t *Size*)

Transmit in master mode an amount of data in non-blocking mode with Interrupt(Non-Blocking mode: Interrupt)

参数

- **hi2c** -- Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress** -- Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData** -- Pointer to data buffer
- **Size** -- Amount of data to be sent

返回 HAL status returned HAL_BUSY/HAL_OK/HAL_ERROR information

HAL_StatusTypeDef **HAL_I2C_Master_Receive_IT**(*I2C_HandleTypeDef* *hi2c, uint16_t *DevAddress*, uint8_t *pData, uint16_t *Size*)

Receive in master mode an amount of data in non-blocking mode with Interrupt(Non-Blocking mode: Interrupt)

参数

- **hi2c** -- Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress** -- Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData** -- Pointer to data buffer
- **Size** -- Amount of data to be sent

返回 HAL status returned HAL_BUSY/HAL_OK/HAL_ERROR information

void **HAL_I2C_IRQHandler**(*I2C_HandleTypeDef* *hi2c)

This function handles I2C event interrupt request.

参数 **hi2c** -- Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

void **HAL_I2C_MasterTxCpltCallback**(*I2C_HandleTypeDef *hi2c*)

Master Tx Transfer completed callback.

参数 **hi2c** -- Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

void **HAL_I2C_MasterRxCpltCallback**(*I2C_HandleTypeDef *hi2c*)

Master Rx Transfer completed callback.

参数 **hi2c** -- Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

void **HAL_I2C_ErrorCallback**(*I2C_HandleTypeDef *hi2c*)

I2C error callback.

参数 **hi2c** -- Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

void **HAL_I2C_AbortCpltCallback**(*I2C_HandleTypeDef *hi2c*)

I2C abort callback.

参数 **hi2c** -- Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

HAL_I2C_StateTypeDef **HAL_I2C_GetState**(*I2C_HandleTypeDef *hi2c*)

Return the I2C handle state.

参数 **hi2c** -- Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

返回 **HAL** -- state

HAL_I2C_ModeTypeDef **HAL_I2C_GetMode**(*I2C_HandleTypeDef *hi2c*)

Returns the I2C Master, Slave, Memory or no mode.

参数 **hi2c** -- Pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module

返回 **HAL** -- mode

uint32_t **HAL_I2C_GetError**(*I2C_HandleTypeDef *hi2c*)

Return the I2C error code.

参数 **hi2c** -- Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

返回 **I2C** -- Error Code

struct I2C_InitTypeDef

#include <lsi2c.h> SPI Configuration Structure definition.

Public Members

uint32_t **ClockSpeed**

Specifies the clock frequency. This parameter must be set to a value lower than 1MHz

uint32_t **OwnAddress1**

Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.

uint32_t **AddressingMode**

Specifies if 7-bit or 10-bit addressing mode is selected. This parameter can be a value of I2C addressing mode.

uint32_t **DualAddressMode**

Specifies if dual addressing mode is selected. This parameter can be a value of I2C dual addressing mode.

uint32_t **OwnAddress2**

Specifies the second device own address if dual addressing mode is selected This parameter can be a 7-bit address.

uint32_t **GeneralCallMode**

Specifies if general call mode is selected. This parameter can be a value of I2C general call addressing mode.

uint32_t **NoStretchMode**

Specifies if nostretch mode is selected. This parameter can be a value of I2C nostretch mode.

struct I2cDMAEnv

#include <*lsi2c.h*> I2C DMA Environment.

Public Members

void (*__Callback__) ( )

uint8_t **DMA_Channel**

struct I2cInterruptEnv

#include <*lsi2c.h*> I2C Interrupt Environment.

Public Members

uint8_t *__pBuffPtr__

Pointer to I2C transfer Buffer

uint16_t **XferCount**

I2C Transfer Counter

struct __I2C_HandleTypeDef

#include <*lsi2c.h*> I2C_handle_Structure_definition I2C handle Structure definition.

Public Members

reg_i2c_t *__Instance__

I2C registers base address

I2C_InitTypeDef __Init__

I2C communication parameters

uint8_t *__pBuffPtr__

Pointer to I2C transfer buffer

uint16_t __XferSize__

I2C transfer size

uint16_t __XferCount__

I2C transfer counter

uint32_t __XferOptions__

I2C transfer options

uint32_t __PreviousState__

I2C communication Previous state and mode context for internal usage

void *__DMAC_Instance__

__struct__ *I2cDMAEnv* __DMA__

__struct__ *I2cInterruptEnv* __Interrupt__

__union__ *__I2C_HandleTypeDef*::**[anonymous]** __Tx_Env__

__union__ *__I2C_HandleTypeDef*::**[anonymous]** __Rx_Env__

I2C Tx/Rx DMA handle parameters

HAL_LockTypeDef __Lock__

I2C locking object

HAL_I2C_StateTypeDef __State__

I2C communication state

HAL_I2C_ModeTypeDef __Mode__

I2C communication mode

uint32_t __ErrorCode__

I2C Error code

uint32_t __Devaddress__

I2C Target device address

uint32_t __EventCount__

I2C Event counter

3.2.10 LSTRNG API

Functions

HAL_StatusTypeDef **HAL_TRNG_Init**(void)

LSTRNG Initialize.

HAL_StatusTypeDef **HAL_TRNG_DeInit**(void)

LSTRNG De-initialization.

返回 status

HAL_StatusTypeDef **HAL_TRNG_GenerateRandomNumber**(uint32_t **random32bit*)

Gets the true random number (Block Mode).

参数 **random32bit** -- **[in]** The value of a truly random number.

返回 status

HAL_StatusTypeDef **HAL_TRNG_GenerateRandomNumber_IT**(void)

Gets the true random number (Interrupt Mode).

返回 status

void **HAL_TRNG_IRQHandler**(void)

void **HAL_TRNG_ReadyDataCallback**(uint32_t *random32bit*)

Callback function that will be invoked in the interrupt context when true random number operation is complete. Overwrite this function to get notification of completion of true random number operation.

参数 **random32bit** -- The value of a truly random number.

3.2.11 LSIWDG API

Functions

HAL_StatusTypeDef **HAL_IWDG_Init**(uint32_t *LoadValue*)

LSIWDG Initialize.

参数 **LoadValue** -- **[in]** Counter load value.

返回 status

HAL_StatusTypeDef **HAL_IWDG_Refresh**(void)

The watchdog counter reloads the value. If the dog is not fed within the WDT timeout time, the WDT timeout behavior occurs.

返回 status

3.2.12 LSTIMER API

Defines

TIM_CLEARINPUTSOURCE_NONE

OCREF_CLR is disabled

TIM_CLEARINPUTSOURCE_ETR

OCREF_CLR is connected to ETRF input

TIM_EVENTSOURCE_UPDATE

Reinitialize the counter and generates an update of the registers

TIM_EVENTSOURCE_CC1

A capture/compare event is generated on channel 1

TIM_EVENTSOURCE_CC2

A capture/compare event is generated on channel 2

TIM_EVENTSOURCE_CC3

A capture/compare event is generated on channel 3

TIM_EVENTSOURCE_CC4

A capture/compare event is generated on channel 4

TIM_EVENTSOURCE_COM

A commutation event is generated

TIM_EVENTSOURCE_TRIGGER

A trigger event is generated

TIM_EVENTSOURCE_BREAK

A break event is generated

TIM_INPUTCHANNELPOLARITY_RISING

Polarity for TIx source

TIM_INPUTCHANNELPOLARITY_FALLING

Polarity for TIx source

TIM_INPUTCHANNELPOLARITY_BOTHEDGE

Polarity for TIx source

TIM_ETRPOLARITY_INVERTED

Polarity for ETR source

TIM_ETRPOLARITY_NONINVERTED

Polarity for ETR source

TIM_ETRPRESCALER_DIV1

No prescaler is used

TIM_ETRPRESCALER_DIV2
ETR input source is divided by 2

TIM_ETRPRESCALER_DIV4
ETR input source is divided by 4

TIM_ETRPRESCALER_DIV8
ETR input source is divided by 8

TIM_COUNTERMODE_UP
Counter used as up-counter

TIM_COUNTERMODE_DOWN
Counter used as down-counter

TIM_COUNTERMODE_CENTERALIGNED1
Center-aligned mode 1

TIM_COUNTERMODE_CENTERALIGNED2
Center-aligned mode 2

TIM_COUNTERMODE_CENTERALIGNED3
Center-aligned mode 3

TIM_CLOCKDIVISION_DIV1
Clock division: tDTS=tCK_INT

TIM_CLOCKDIVISION_DIV2
Clock division: tDTS=2*tCK_INT

TIM_CLOCKDIVISION_DIV4
Clock division: tDTS=4*tCK_INT

TIM_OUTPUTSTATE_DISABLE
Capture/Compare 1 output disabled

TIM_OUTPUTSTATE_ENABLE
Capture/Compare 1 output enabled

TIM_AUTORELOAD_PRELOAD_DISABLE
TIMx_ARR register is not buffered

TIM_AUTORELOAD_PRELOAD_ENABLE
TIMx_ARR register is buffered

TIM_OCFAST_DISABLE
Output Compare fast disable

TIM_OCFAST_ENABLE
Output Compare fast enable

TIM_OUTPUTNSTATE_DISABLE

OCxN is disabled

TIM_OUTPUTNSTATE_ENABLE

OCxN is enabled

TIM_OCPOLARITY_HIGH

Capture/Compare output polarity

TIM_OCPOLARITY_LOW

Capture/Compare output polarity

TIM_OCNPOLARITY_HIGH

Capture/Compare complementary output polarity

TIM_OCNPOLARITY_LOW

Capture/Compare complementary output polarity

TIM_OCIDLESTATE_SET

Output Idle state: OCx=1 when MOE=0

TIM_OCIDLESTATE_RESET

Output Idle state: OCx=0 when MOE=0

TIM_OCNIDLESTATE_SET

Complementary output Idle state: OCxN=1 when MOE=0

TIM_OCNIDLESTATE_RESET

Complementary output Idle state: OCxN=0 when MOE=0

TIM_ICPOLARITY_RISING

Capture triggered by rising edge on timer input

TIM_ICPOLARITY_FALLING

Capture triggered by falling edge on timer input

TIM_ICPOLARITY_BOTHEDGE

Capture triggered by both rising and falling edges on timer input

TIM_ICSELECTION_DIRECTTI

TIM Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively

TIM_ICSELECTION_INDIRECTTI

TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively

TIM_ICSELECTION_TRC

TIM Input 1, 2, 3 or 4 is selected to be connected to TRC

TIM_ICPSC_DIV1

Capture performed each time an edge is detected on the capture input

TIM_ICPSC_DIV2
Capture performed once every 2 events

TIM_ICPSC_DIV4
Capture performed once every 4 events

TIM_ICPSC_DIV8
Capture performed once every 8 events

TIM_OPMODE_SINGLE
Counter stops counting at the next update event

TIM_OPMODE_REPETITIVE
Counter is not stopped at update event

TIM_IT_UPDATE
Update interrupt

TIM_IT_CC1
Capture/Compare 1 interrupt

TIM_IT_CC2
Capture/Compare 2 interrupt

TIM_IT_CC3
Capture/Compare 3 interrupt

TIM_IT_CC4
Capture/Compare 4 interrupt

TIM_IT_COM
Commutation interrupt

TIM_IT_TRIGGER
Trigger interrupt

TIM_IT_BREAK
Break interrupt

TIM_IT_CC1O
Capture/compare 1 overcap interrupt

TIM_IT_CC2O
Break interrupt 2 overcap interrupt

TIM_IT_CC3O
Break interrupt 3 overcap interrupt

TIM_IT_CC4O
Break interrupt 4 overcap interrupt

TIM_FLAG_UPDATE

Update interrupt flag

TIM_FLAG_CC1

Capture/Compare 1 interrupt flag

TIM_FLAG_CC2

Capture/Compare 2 interrupt flag

TIM_FLAG_CC3

Capture/Compare 3 interrupt flag

TIM_FLAG_CC4

Capture/Compare 4 interrupt flag

TIM_FLAG_COM

Commutation interrupt flag

TIM_FLAG_TRIGGER

Trigger interrupt flag

TIM_FLAG_BREAK

Break interrupt flag

TIM_FLAG_CC1OF

Capture 1 overcapture flag

TIM_FLAG_CC2OF

Capture 2 overcapture flag

TIM_FLAG_CC3OF

Capture 3 overcapture flag

TIM_FLAG_CC4OF

Capture 4 overcapture flag

TIM_CHANNEL_1

Capture/compare channel 1 identifier

TIM_CHANNEL_2

Capture/compare channel 2 identifier

TIM_CHANNEL_3

Capture/compare channel 3 identifier

TIM_CHANNEL_4

Capture/compare channel 4 identifier

TIM_CHANNEL_ALL

Global Capture/compare channel identifier

TIM_CLOCKSOURCE_ETRMODE2
External clock source mode 2

TIM_CLOCKSOURCE_INTERNAL
Internal clock source

TIM_CLOCKSOURCE_ITR0
External clock source mode 1 (ITR0)

TIM_CLOCKSOURCE_ITR1
External clock source mode 1 (ITR1)

TIM_CLOCKSOURCE_ITR2
External clock source mode 1 (ITR2)

TIM_CLOCKSOURCE_ITR3
External clock source mode 1 (ITR3)

TIM_CLOCKSOURCE_TI1ED
External clock source mode 1 (TTI1FP1 + edge detect.)

TIM_CLOCKSOURCE_TI1
External clock source mode 1 (TTI1FP1)

TIM_CLOCKSOURCE_TI2
External clock source mode 1 (TTI2FP2)

TIM_CLOCKSOURCE_ETRMODE1
External clock source mode 1 (ETRF)

TIM_CLOCKPOLARITY_INVERTED
Polarity for ETRx clock sources

TIM_CLOCKPOLARITY_NONINVERTED
Polarity for ETRx clock sources

TIM_CLOCKPOLARITY_RISING
Polarity for TIx clock sources

TIM_CLOCKPOLARITY_FALLING
Polarity for TIx clock sources

TIM_CLOCKPOLARITY_BOTHEDGE
Polarity for TIx clock sources

TIM_CLOCKPRESCALER_DIV1
No prescaler is used

TIM_CLOCKPRESCALER_DIV2
Prescaler for External ETR Clock: Capture performed once every 2 events.

TIM_CLOCKPRESCALER_DIV4

Prescaler for External ETR Clock: Capture performed once every 4 events.

TIM_CLOCKPRESCALER_DIV8

Prescaler for External ETR Clock: Capture performed once every 8 events.

TIM_CLEARINPUTPOLARITY_INVERTED

Polarity for ETRx pin

TIM_CLEARINPUTPOLARITY_NONINVERTED

Polarity for ETRx pin

TIM_CLEARINPUTPRESCALER_DIV1

No prescaler is used

TIM_CLEARINPUTPRESCALER_DIV2

Prescaler for External ETR pin: Capture performed once every 2 events.

TIM_CLEARINPUTPRESCALER_DIV4

Prescaler for External ETR pin: Capture performed once every 4 events.

TIM_CLEARINPUTPRESCALER_DIV8

Prescaler for External ETR pin: Capture performed once every 8 events.

TIM_OSSR_ENABLE

When inactive, OC/OCN outputs are enabled (still controlled by the timer)

TIM_OSSR_DISABLE

When inactive, OC/OCN outputs are disabled (not controlled any longer by the timer)

TIM_OSSI_ENABLE

When inactive, OC/OCN outputs are enabled (still controlled by the timer)

TIM_OSSI_DISABLE

When inactive, OC/OCN outputs are disabled (not controlled any longer by the timer)

TIM_LOCKLEVEL_OFF

LOCK OFF

TIM_LOCKLEVEL_1

LOCK Level 1

TIM_LOCKLEVEL_2

LOCK Level 2

TIM_LOCKLEVEL_3

LOCK Level 3

TIM_BREAK_ENABLE

Break input BRK is enabled

TIM_BREAK_DISABLE

Break input BRK is disabled

TIM_BREAKPOLARITY_LOW

Break input BRK is active low

TIM_BREAKPOLARITY_HIGH

Break input BRK is active high

TIM_AUTOMATICOUTPUT_DISABLE

MOE can be set only by software

TIM_AUTOMATICOUTPUT_ENABLE

MOE can be set by software or automatically at the next update event (if none of the break inputs BRK and BRK2 is active)

TIM_TRGO_RESET

TIMx_EGR.UG bit is used as trigger output (TRGO)

TIM_TRGO_ENABLE

TIMx_CR1.CEN bit is used as trigger output (TRGO)

TIM_TRGO_UPDATE

Update event is used as trigger output (TRGO)

TIM_TRGO_OC1

Capture or a compare match 1 is used as trigger output (TRGO)

TIM_TRGO_OC1REF

OC1REF signal is used as trigger output (TRGO)

TIM_TRGO_OC2REF

OC2REF signal is used as trigger output(TRGO)

TIM_TRGO_OC3REF

OC3REF signal is used as trigger output(TRGO)

TIM_TRGO_OC4REF

OC4REF signal is used as trigger output(TRGO)

TIM_MASTERSLAVEMODE_ENABLE

No action

TIM_MASTERSLAVEMODE_DISABLE

Master/slave mode is selected

TIM_SLAVEMODE_DISABLE

Slave mode disabled

TIM_SLAVEMODE_RESET

Reset Mode

TIM_SLAVEMODE_GATED
Gated Mode

TIM_SLAVEMODE_TRIGGER
Trigger Mode

TIM_SLAVEMODE_EXTERNAL1
External Clock Mode 1

TIM_OCMODE_TIMING
Frozen

TIM_OCMODE_ACTIVE
Set channel to active level on match

TIM_OCMODE_INACTIVE
Set channel to inactive level on match

TIM_OCMODE_TOGGLE
Toggle

TIM_OCMODE_PWM1
PWM mode 1

TIM_OCMODE_PWM2
PWM mode 2

TIM_OCMODE_FORCED_ACTIVE
Force active level

TIM_OCMODE_FORCED_INACTIVE
Force inactive level

TIM_TS_ITR0
Internal Trigger 0 (ITR0)

TIM_TS_ITR1
Internal Trigger 1 (ITR1)

TIM_TS_ITR2
Internal Trigger 2 (ITR2)

TIM_TS_ITR3
Internal Trigger 3 (ITR3)

TIM_TS_TI1F_ED
TI1 Edge Detector (TI1F_ED)

TIM_TS_TI1FP1
Filtered Timer Input 1 (TI1FP1)

TIM_TS_TI2FP2

Filtered Timer Input 2 (TI2FP2)

TIM_TS_ETRF

Filtered External Trigger input (ETRF)

TIM_TS_NONE

No trigger selected

TIM_TRIGGERPOLARITY_INVERTED

Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_NONINVERTED

Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_RISING

Polarity for TIxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_FALLING

Polarity for TIxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_BOTHEDGE

Polarity for TIxFPx or TI1_ED trigger sources

TIM_TRIGGERPRESCALER_DIV1

No prescaler is used

TIM_TRIGGERPRESCALER_DIV2

Prescaler for External ETR Trigger: Capture performed once every 2 events.

TIM_TRIGGERPRESCALER_DIV4

Prescaler for External ETR Trigger: Capture performed once every 4 events.

TIM_TRIGGERPRESCALER_DIV8

Prescaler for External ETR Trigger: Capture performed once every 8 events.

TIM_TI1SELECTION_CH1

The TIMx_CH1 pin is connected to TI1 input

TIM_TI1SELECTION_XORCOMBINATION

The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

TIM_CCx_ENABLE

Input or output channel is enabled

TIM_CCx_DISABLE

Input or output channel is disabled

TIM_CCxN_ENABLE

Complementary output channel is enabled

TIM_CCxN_DISABLE

Complementary output channel is enabled

__HAL_TIM_RESET_HANDLE_STATE(*__HANDLE__*)

Reset TIM handle state.

__HAL_TIM_ENABLE(*__HANDLE__*)

Enable the TIM peripheral.

__HAL_TIM_MOE_ENABLE(*__HANDLE__*)

Enable the TIM main Output.

__HAL_TIM_DISABLE(*__HANDLE__*)

Disable the TIM peripheral.

__HAL_TIM_MOE_DISABLE(*__HANDLE__*)

Disable the TIM main Output.

注解: The Main Output Enable of a timer instance is disabled only if all the CCx and CCxN channels have been disabled

---

__HAL_TIM_MOE_DISABLE_UNCONDITIONALLY(*__HANDLE__*)

Disable the TIM main Output.

注解: The Main Output Enable of a timer instance is disabled unconditionally

---

__HAL_TIM_ENABLE_IT(*__HANDLE__*, *__INTERRUPT__*)

Enable the specified TIM interrupt.

__HAL_TIM_DISABLE_IT(*__HANDLE__*, *__INTERRUPT__*)

Disable the specified TIM interrupt.

__HAL_TIM_GET_FLAG(*__HANDLE__*, *__FLAG__*)

Check whether the specified TIM interrupt flag is set or not.

__HAL_TIM_CLEAR_FLAG(*__HANDLE__*, *__FLAG__*)

Clear the specified TIM interrupt flag.

__HAL_TIM_GET_IT_SOURCE(*__HANDLE__*, *__INTERRUPT__*)

Check whether the specified TIM interrupt source is enabled or not.

__HAL_TIM_CLEAR_IT(*__HANDLE__*, *__INTERRUPT__*)

Clear the TIM interrupt pending bits.

__HAL_TIM_IS_TIM_COUNTING_DOWN(*__HANDLE__*)

Indicates whether or not the TIM Counter is used as downcounter.

注解: This macro is particularly useful to get the counting mode when the timer operates in Center-aligned mode or Encoder mode.

返回 **False** -- (Counter used as upcounter) or True (Counter used as downcounter)

__HAL_TIM_SET_PRESCALER(*__HANDLE__*, *__PRESC__*)

Set the TIM Prescaler on runtime.

__HAL_TIM_SET_COUNTER(*__HANDLE__*, *__COUNTER__*)

Set the TIM Counter Register value on runtime.

__HAL_TIM_GET_COUNTER(*__HANDLE__*)

Get the TIM Counter Register value on runtime.

返回 **16-bit** -- or 32-bit value of the timer counter register (TIMx_CNT)

__HAL_TIM_SET_AUTORELOAD(*__HANDLE__*, *__AUTORELOAD__*)

Set the TIM Autoreload Register value on runtime without calling another time any Init function.

__HAL_TIM_GET_AUTORELOAD(*__HANDLE__*)

Get the TIM Autoreload Register value on runtime.

返回 **16-bit** -- or 32-bit value of the timer auto-reload register(TIMx_ARR)

__HAL_TIM_SET_CLOCKDIVISION(*__HANDLE__*, *__CKD__*)

Set the TIM Clock Division value on runtime without calling another time any Init function.

__HAL_TIM_GET_CLOCKDIVISION(*__HANDLE__*)

Get the TIM Clock Division value on runtime.

__HAL_TIM_SET_ICPRESCALER(*__HANDLE__*, *__CHANNEL__*, *__ICPSC__*)

Set the TIM Input Capture prescaler on runtime without calling another time HAL_TIM_IC_ConfigChannel() function.

__HAL_TIM_GET_ICPRESCALER(*__HANDLE__*, *__CHANNEL__*)

Get the TIM Input Capture prescaler on runtime.

__HAL_TIM_SET_COMPARE(*__HANDLE__*, *__CHANNEL__*, *__COMPARE__*)

Set the TIM Capture Compare Register value on runtime without calling another time ConfigChannel function.

__HAL_TIM_GET_COMPARE(*__HANDLE__*, *__CHANNEL__*)

Get the TIM Capture Compare Register value on runtime.

返回 **16-bit** -- or 32-bit value of the capture/compare register (TIMx_CCRy)

__HAL_TIM_ENABLE_OCxPRELOAD(*__HANDLE__*, *__CHANNEL__*)

Set the TIM Output compare preload.

__HAL_TIM_DISABLE_OCxPRELOAD(*__HANDLE__*, *__CHANNEL__*)

Reset the TIM Output compare preload.

__HAL_TIM_ENABLE_OCxFAST(*__HANDLE__*, *__CHANNEL__*)

Enable fast mode for a given channel.

注解: When fast mode is enabled an active edge on the trigger input acts like a compare match on CCx output. Delay to sample the trigger input and to activate CCx output is reduced to 3 clock cycles.

注解: Fast mode acts only if the channel is configured in PWM1 or PWM2 mode.

__HAL_TIM_DISABLE_OCxFAST(*__HANDLE__*, *__CHANNEL__*)

Disable fast mode for a given channel.

注解: When fast mode is disabled CCx output behaves normally depending on counter and CCRx values even when the trigger is ON. The minimum delay to activate CCx output when an active edge occurs on the trigger input is 5 clock cycles.

__HAL_TIM_URS_ENABLE(*__HANDLE__*)

Set the Update Request Source (URS) bit of the TIMx_CR1 register.

注解: When the URS bit of the TIMx_CR1 register is set, only counter overflow/underflow generates an update interrupt or DMA request (if enabled)

__HAL_TIM_URS_DISABLE(*__HANDLE__*)

Reset the Update Request Source (URS) bit of the TIMx_CR1 register.

注解: When the URS bit of the TIMx_CR1 register is reset, any of the following events generate an update interrupt or DMA request (if enabled): _ Counter overflow underflow _ Setting the UG bit _ Update generation through the slave mode controller

---

__HAL_TIM_SET_CAPTUREPOLARITY(*__HANDLE__*, *__CHANNEL__*, *__POLARITY__*)

Set the TIM Capture x input polarity on runtime.

TIM_CCER_CCxE_MASK

TIM_CCER_CCxNE_MASK

TIM_SET_ICPRESCALERVALUE(*__HANDLE__*, *__CHANNEL__*, *__ICPSC__*)

TIM_RESET_ICPRESCALERVALUE(*__HANDLE__*, *__CHANNEL__*)

TIM_SET_CAPTUREPOLARITY(*__HANDLE__*, *__CHANNEL__*, *__POLARITY__*)

TIM_RESET_CAPTUREPOLARITY(*__HANDLE__*, *__CHANNEL__*)

Enums

enum HAL_TIM_StateTypeDef

HAL State structures definition.

Values:

enumerator HAL_TIM_STATE_RESET

Peripheral not yet initialized or disabled

enumerator HAL_TIM_STATE_READY

Peripheral Initialized and ready for use

enumerator HAL_TIM_STATE_BUSY

An internal process is ongoing

enumerator HAL_TIM_STATE_TIMEOUT

Timeout state

enumerator HAL_TIM_STATE_ERROR

Reception process is ongoing

enum HAL_TIM_ActiveChannel

HAL Active channel structures definition.

Values:

enumerator HAL_TIM_ACTIVE_CHANNEL_1

The active channel is 1

enumerator HAL_TIM_ACTIVE_CHANNEL_2

The active channel is 2

enumerator HAL_TIM_ACTIVE_CHANNEL_3

The active channel is 3

enumerator HAL_TIM_ACTIVE_CHANNEL_4

The active channel is 4

enumerator HAL_TIM_ACTIVE_CHANNEL_CLEARED

All active channels cleared

Functions

HAL_StatusTypeDef **HAL_TIM_Init**(*TIM_HandleTypeDef* *htim)

Initializes the TIM Time base Unit according to the specified parameters in the *TIM_HandleTypeDef* and initialize the associated handle.

注解： Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode.

参数 **htim** -- TIM Base handle

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_DeInit**(*TIM_HandleTypeDef* *htim)

DeInitializes the TIM peripheral.

参数 **htim** -- TIM Base handle

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_Base_Start**(*TIM_HandleTypeDef* *htim)

Starts the TIM Base generation.

参数 **htim** -- TIM Base handle

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_Base_Stop**(*TIM_HandleTypeDef* *htim)

Stops the TIM Base generation.

参数 **htim** -- TIM Base handle

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_Base_Start_IT**(*TIM_HandleTypeDef* *htim)

Starts the TIM Base generation in interrupt mode.

参数 **htim** -- TIM Base handle

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_Base_Stop_IT**(*TIM_HandleTypeDef* *htim)

Stops the TIM Base generation in interrupt mode.

参数 **htim** -- TIM Base handle

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_OC_Start**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Starts the TIM Output Compare signal generation.

参数

- **htim** -- TIM Output Compare handle
- **Channel** -- TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_OC_Stop**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Stops the TIM Output Compare signal generation.

参数

- **htim** -- TIM Output Compare handle
- **Channel** -- TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_OC_Start_IT**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Starts the TIM Output Compare signal generation in interrupt mode.

参数

- **htim** -- TIM Output Compare handle
- **Channel** -- TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_OC_Stop_IT**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Stops the TIM Output Compare signal generation in interrupt mode.

参数

- **htim** -- TIM Output Compare handle
- **Channel** -- TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_PWM_Start**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Starts the PWM signal generation.

参数

- **htim** -- TIM handle
- **Channel** -- TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_PWM_Stop**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Stops the PWM signal generation.

参数

- **htim** -- TIM PWM handle
- **Channel** -- TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_PWM_Start_IT**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Starts the PWM signal generation in interrupt mode.

参数

- **htim** -- TIM PWM handle

- **Channel** -- TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_PWM_Stop_IT**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Stops the PWM signal generation in interrupt mode.

参数

- **htim** -- TIM PWM handle
- **Channel** -- TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_IC_Start**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Starts the TIM Input Capture measurement.

参数

- **htim** -- TIM Input Capture handle
- **Channel** -- TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_IC_Stop**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Stops the TIM Input Capture measurement.

参数

- **htim** -- TIM Input Capture handle
- **Channel** -- TIM Channels to be disabled This parameter can be one of the following values:

- TIM_CHANNEL_1: TIM Channel 1 selected
- TIM_CHANNEL_2: TIM Channel 2 selected
- TIM_CHANNEL_3: TIM Channel 3 selected
- TIM_CHANNEL_4: TIM Channel 4 selected

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_IC_Start_IT**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Starts the TIM Input Capture measurement in interrupt mode.

参数

- **htim** -- TIM Input Capture handle
- **Channel** -- TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_IC_Stop_IT**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Stops the TIM Input Capture measurement in interrupt mode.

参数

- **htim** -- TIM Input Capture handle
- **Channel** -- TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_OnePulse_Init**(*TIM_HandleTypeDef* *htim, uint32_t *OnePulseMode*)

Initializes the TIM One Pulse Time Base according to the specified parameters in the *TIM_HandleTypeDef* and initializes the associated handle.

注解: Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode.

参数

- **htim** -- TIM One Pulse handle
- **OnePulseMode** -- Select the One pulse mode. This parameter can be one of the following values:
  - TIM_OPMODE_SINGLE: Only one pulse will be generated.
  - TIM_OPMODE_REPETITIVE: Repetitive pulses will be generated.

返回 **HAL** -- status

void **HAL_TIM_IRQHandler**(*TIM_HandleTypeDef* *htim)

This function handles TIM interrupts requests.

参数 **htim** -- TIM handle

返回 **None** --

HAL_StatusTypeDef **HAL_TIM_OC_ConfigChannel**(*TIM_HandleTypeDef* *htim, *TIM_OC_InitTypeDef* *sConfig, uint32_t *Channel*)

Initializes the TIM Output Compare Channels according to the specified parameters in the *TIM_OC_InitTypeDef*.

参数

- **htim** -- TIM Output Compare handle
- **sConfig** -- TIM Output Compare configuration structure
- **Channel** -- TIM Channels to configure This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_PWM_ConfigChannel**(*TIM_HandleTypeDef* *htim, *TIM_OC_InitTypeDef* *sConfig, uint32_t *Channel*)

Initializes the TIM PWM channels according to the specified parameters in the *TIM_OC_InitTypeDef*.

参数

- **htim** -- TIM PWM handle
- **sConfig** -- TIM PWM configuration structure
- **Channel** -- TIM Channels to be configured This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

- TIM_CHANNEL_3: TIM Channel 3 selected
- TIM_CHANNEL_4: TIM Channel 4 selected

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_IC_ConfigChannel**(*TIM_HandleTypeDef* *htim, *TIM_IC_InitTypeDef* *sConfig, uint32_t *Channel*)

Initializes the TIM Input Capture Channels according to the specified parameters in the *TIM_IC_InitTypeDef*.

参数

- **htim** -- TIM IC handle
- **sConfig** -- TIM Input Capture configuration structure
- **Channel** -- TIM Channel to configure This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_ConfigOCrefClear**(*TIM_HandleTypeDef* *htim, *TIM_ClearInputConfigTypeDef* *sClearInputConfig, uint32_t *Channel*)

Configures the OCRef clear feature.

参数

- **htim** -- TIM handle
- **sClearInputConfig** -- pointer to a *TIM_ClearInputConfigTypeDef* structure that contains the OCREF clear feature and parameters for the TIM peripheral.
- **Channel** -- specifies the TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_ConfigClockSource**(*TIM_HandleTypeDef* *htim, *TIM_ClockConfigTypeDef* *sClockSourceConfig)

Configures the clock source to be used.

参数

- **htim** -- TIM handle
- **sClockSourceConfig** -- pointer to a *TIM_ClockConfigTypeDef* structure that contains the clock source information for the TIM peripheral.

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_ConfigTI1Input**(*TIM_HandleTypeDef* *htim, uint32_t *TI1_Selection*)

Selects the signal connected to the TI1 input: direct from CH1_input or a XOR combination between CH1_input, CH2_input & CH3_input.

参数

- **htim** -- TIM handle.
- **TI1_Selection** -- Indicate whether or not channel 1 is connected to the output of a XOR gate. This parameter can be one of the following values:
  - TIM_TI1SELECTION_CH1: The TIMx_CH1 pin is connected to TI1 input
  - TIM_TI1SELECTION_XORCOMBINATION: The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIM_GenerateEvent**(*TIM_HandleTypeDef* *htim, uint32_t *EventSource*)

Generate a software event.

注解: Basic timers can only generate an update event.

注解: TIM_EVENTSOURCE_COM is relevant only with advanced timer instances.

注解: TIM_EVENTSOURCE_BREAK are relevant only for timer instances supporting a break input.

参数

- **htim** -- TIM handle
- **EventSource** -- specifies the event source. This parameter can be one of the following values:
  - TIM_EVENTSOURCE_UPDATE: Timer update Event source
  - TIM_EVENTSOURCE_CC1: Timer Capture Compare 1 Event source
  - TIM_EVENTSOURCE_CC2: Timer Capture Compare 2 Event source

- TIM_EVENTSOURCE_CC3: Timer Capture Compare 3 Event source
- TIM_EVENTSOURCE_CC4: Timer Capture Compare 4 Event source
- TIM_EVENTSOURCE_COM: Timer COM event source
- TIM_EVENTSOURCE_TRIGGER: Timer Trigger Event source
- TIM_EVENTSOURCE_BREAK: Timer Break event source

返回 **HAL** -- status

uint32_t **HAL_TIM_ReadCapturedValue**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Read the captured value from Capture Compare unit.

参数

- **htim** -- TIM handle.
- **Channel** -- TIM Channels to be enabled

返回 **Captured** -- value

HAL_TIM_StateTypeDef **HAL_TIM_GetState**(*TIM_HandleTypeDef* *htim)

Return the TIM Base handle state.

参数 **htim** -- TIM Base handle

返回 **HAL** -- state

HAL_StatusTypeDef **HAL_TIMEx_OCN_Start**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Starts the TIM Output Compare signal generation on the complementary output.

参数

- **htim** -- TIM Output Compare handle
- **Channel** -- TIM Channel to be enabled

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIMEx_OCN_Stop**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Stops the TIM Output Compare signal generation on the complementary output.

参数

- **htim** -- TIM handle
- **Channel** -- TIM Channel to be disabled

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIMEx_OCN_Start_IT**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Starts the TIM Output Compare signal generation in interrupt mode on the complementary output.

参数

- **htim** -- TIM OC handle
- **Channel** -- TIM Channel to be enabled

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIMEx_OCN_Stop_IT**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Stops the TIM Output Compare signal generation in interrupt mode on the complementary output.

参数

- **htim** -- TIM Output Compare handle
- **Channel** -- TIM Channel to be disabled

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIMEx_PWMN_Start**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Starts the PWM signal generation on the complementary output.

参数

- **htim** -- TIM handle
- **Channel** -- TIM Channel to be enabled

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIMEx_PWMN_Stop**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Stops the PWM signal generation on the complementary output.

参数

- **htim** -- TIM handle
- **Channel** -- TIM Channel to be disabled

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIMEx_PWMN_Start_IT**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Starts the PWM signal generation in interrupt mode on the complementary output.

参数

- **htim** -- TIM handle
- **Channel** -- TIM Channel to be disabled

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIMEx_PWMN_Stop_IT**(*TIM_HandleTypeDef* *htim, uint32_t *Channel*)

Stops the PWM signal generation in interrupt mode on the complementary output.

参数

- **htim** -- TIM handle
- **Channel** -- TIM Channel to be disabled

返回 **HAL** -- status

HAL_StatusTypeDef **HAL_TIMEx_ConfigBreakDeadTime**(*TIM_HandleTypeDef *htim*, *TIM_BreakDeadTimeConfigTypeDef *sBreakDeadTimeConfig*)

Configures the Break feature, dead time, Lock level, OSSI/OSSR State and the AOE(automatic output enable).

注解: Interrupts can be generated when an active level is detected on the break input, the break 2 input or the system break input. Break interrupt can be enabled by calling the __HAL_TIM_ENABLE_IT macro.

参数

- **htim** -- TIM handle
- **sBreakDeadTimeConfig** -- pointer to a TIM_ConfigBreakDeadConfigTypeDef structure that contains the BDTR Register configuration information for the TIM peripheral.

返回 **HAL** -- status

void **HAL_TIMEx_CommutCallback**(*TIM_HandleTypeDef *htim*)

Hall commutation changed callback in non-blocking mode.

参数 **htim** -- TIM handle

返回 **None** --

void **HAL_TIMEx_CommutHalfCpltCallback**(*TIM_HandleTypeDef *htim*)

Hall commutation changed half complete callback in non-blocking mode.

参数 **htim** -- TIM handle

返回 **None** --

void **HAL_TIMEx_BreakCallback**(*TIM_HandleTypeDef *htim*)

Hall Break detection callback in non-blocking mode.

参数 **htim** -- TIM handle

返回 **None** --

void **HAL_TIM_PeriodElapsedCallback**(*TIM_HandleTypeDef *htim*)

Period elapsed callback in non-blocking mode.

参数 **htim** -- TIM handle

返回 **None** --

void **HAL_TIM_PeriodElapsedHalfCpltCallback**(*TIM_HandleTypeDef *htim*)

Period elapsed half complete callback in non-blocking mode.

参数 **htim** -- TIM handle

返回 None --

void **HAL_TIM_OC_DelayElapsedCallback**(*TIM_HandleTypeDef *htim*)

Output Compare callback in non-blocking mode.

参数 **htim** -- TIM OC handle

返回 None --

void **HAL_TIM_IC_CaptureCallback**(*TIM_HandleTypeDef *htim*)

Input Capture callback in non-blocking mode.

参数 **htim** -- TIM IC handle

返回 None --

void **HAL_TIM_IC_CaptureHalfCpltCallback**(*TIM_HandleTypeDef *htim*)

Input Capture half complete callback in non-blocking mode.

参数 **htim** -- TIM IC handle

返回 None --

void **HAL_TIM_PWM_PulseFinishedCallback**(*TIM_HandleTypeDef *htim*)

PWM Pulse finished callback in non-blocking mode.

参数 **htim** -- TIM handle

返回 None --

void **HAL_TIM_PWM_PulseFinishedHalfCpltCallback**(*TIM_HandleTypeDef *htim*)

PWM Pulse finished half complete callback in non-blocking mode.

参数 **htim** -- TIM handle

返回 None --

void **HAL_TIM_TriggerCallback**(*TIM_HandleTypeDef *htim*)

Hall Trigger detection callback in non-blocking mode.

参数 **htim** -- TIM handle

返回 None --

void **HAL_TIM_TriggerHalfCpltCallback**(*TIM_HandleTypeDef *htim*)

Hall Trigger detection half complete callback in non-blocking mode.

参数 **htim** -- TIM handle

返回 None --

void **HAL_TIM_ErrorCallback**(*TIM_HandleTypeDef *htim*)

Timer error callback in non-blocking mode.

参数 **htim** -- TIM handle

返回 None --

struct TIM_Base_InitTypeDef

#include <lstimer.h> LSTIMER Macro for Register Access.

TIM Time base Configuration Structure definition

Public Members

uint32_t **Prescaler**

Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF

uint32_t **CounterMode**

Specifies the counter mode. This parameter can be a value of TIM Counter Mode

uint32_t **Period**

Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.

uint32_t **ClockDivision**

Specifies the clock division. This parameter can be a value of TIM Clock Division

uint32_t **RepetitionCounter**

Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to:

- the number of PWM periods in edge-aligned mode
- the number of half PWM period in center-aligned mode GP timers: this parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF. Advanced timers: this parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.

uint32_t **AutoReloadPreload**

Specifies the auto-reload preload. This parameter can be a value of TIM Auto-Reload Preload

struct TIM_OC_InitTypeDef

#include <lstimer.h> TIM Output Compare Configuration Structure definition.

Public Members

uint32_t **OCMode**

Specifies the TIM mode. This parameter can be a value of TIM Output Compare and PWM Modes

uint32_t **Pulse**

Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF

uint32_t **OCPolarity**

Specifies the output polarity. This parameter can be a value of TIM Output Compare Polarity

uint32_t **OCNPolarity**

Specifies the complementary output polarity. This parameter can be a value of TIM Complementary Output Compare Polarity

---

注解： This parameter is valid only for timer instances supporting break feature.

---

uint32_t **OCFastMode**

Specifies the Fast mode state. This parameter can be a value of TIM Output Fast State

---

注解： This parameter is valid only in PWM1 and PWM2 mode.

---

uint32_t **OCIdleState**

Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of TIM Output Compare Idle State

---

注解： This parameter is valid only for timer instances supporting break feature.

---

uint32_t **OCNIdleState**

Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of TIM Complementary Output Compare Idle State

---

注解： This parameter is valid only for timer instances supporting break feature.

---

struct TIM_IC_InitTypeDef

#include <lstimer.h> TIM Input Capture Configuration Structure definition.

Public Members

uint32_t **ICPolarity**

Specifies the active edge of the input signal. This parameter can be a value of TIM Input Capture Polarity

uint32_t **ICSelection**

Specifies the input. This parameter can be a value of TIM Input Capture Selection

uint32_t **ICPrescaler**

Specifies the Input Capture Prescaler. This parameter can be a value of TIM Input Capture Prescaler

uint32_t **ICFilter**

Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

struct TIM_ClockConfigTypeDef

#include <lstimer.h> Clock Configuration Handle Structure definition.

Public Members

uint32_t **ClockSource**

TIM clock sources This parameter can be a value of TIM Clock Source

uint32_t **ClockPolarity**

TIM clock polarity This parameter can be a value of TIM Clock Polarity

uint32_t **ClockPrescaler**

TIM clock prescaler This parameter can be a value of TIM Clock Prescaler

uint32_t **ClockFilter**

TIM clock filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

struct TIM_ClearInputConfigTypeDef

#include <lstimer.h> TIM Clear Input Configuration Handle Structure definition.

Public Members

uint32_t **ClearInputState**

TIM clear Input state This parameter can be ENABLE or DISABLE

uint32_t **ClearInputSource**

TIM clear Input sources This parameter can be a value of TIM Clear Input Source

uint32_t **ClearInputPolarity**

TIM Clear Input polarity This parameter can be a value of TIM Clear Input Polarity

uint32_t **ClearInputPrescaler**

TIM Clear Input prescaler This parameter must be 0: When OCRef clear feature is used with ETR source, ETR prescaler must be off

uint32_t **ClearInputFilter**

TIM Clear Input filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

struct TIM_BreakDeadTimeConfigTypeDef

#include <lstimer.h> TIM Break input(s) and Dead time configuration Structure definition.

注解： 2 break inputs can be configured (BKIN and BKIN2) with configurable filter and polarity.

Public Members

uint32_t **OffStateRunMode**

TIM off state in run mode This parameter can be a value of TIM OSSR OffState Selection for Run mode state

uint32_t **OffStateIDLEMode**

TIM off state in IDLE mode This parameter can be a value of TIM OSSI OffState Selection for Idle mode state

uint32_t **LockLevel**

TIM Lock level This parameter can be a value of TIM Lock level

uint32_t **DeadTime**

TIM dead Time This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF

uint32_t **BreakState**

TIM Break State This parameter can be a value of TIM Break Input Enable

uint32_t **BreakPolarity**

TIM Break input polarity This parameter can be a value of TIM Break Input Polarity

uint32_t **BreakFilter**

Specifies the break input filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

uint32_t **AutomaticOutput**

TIM Automatic Output Enable state This parameter can be a value of TIM Automatic Output Enable

struct TIM_HandleTypeDef

#include <lstimer.h> TIM Time Base Handle Structure definition.

Public Members

reg_timer_t *__Instance__

Register base address

TIM_Base_InitTypeDef __Init__

TIM Time Base required parameters

HAL_TIM_ActiveChannel __Channel__

Active channel

HAL_LockTypeDef __Lock__

Locking object

HAL_TIM_StateTypeDef __State__

TIM operation state

3.2.13 LSPIS API

Defines

LSPIS

LSPIS Macro for Register Access.

Enums

enum pis_sync_mode

PIS Sync Mode.

Values:

enumerator PIS_SYNC_DIRECT

enumerator PIS_SYNC_SRC_LEVEL

enumerator PIS_SYNC_SRC_PULSE

enum pis_edge_sel

PIS Edge Select.

Values:

enumerator PIS_EDGE_NONE

enumerator PIS_POS_EDGE

enumerator PIS_NEG_EDGE

enumerator PIS_BOTH_EDGES

Functions

HAL_StatusTypeDef **HAL_PIS_Init**(void)

LSPIS Initialize.

返回 status

HAL_StatusTypeDef **HAL_PIS_DeInit**(void)

LSPIS De-Initialize.

返回 status

HAL_StatusTypeDef **HAL_PIS_Config**(uint8_t *channel*, **enum** pis_src *src*, **enum** pis_dst *dst*, **enum** *pis_sync_mode* *sync*, **enum** *pis_edge_sel* *edge*)

LSPIS Channel Configuration.

参数

- **channel** --
- **src** -- enum pis_src in pis_config.h
- **dst** -- enum pis_dst in pis_config.h
- **sync** -- *pis_sync_mode*
- **edge** -- *pis_edge_sel*

返回 status

HAL_StatusTypeDef **HAL_PIS_Output**(uint8_t *channel*, bool *enable*)

LSPIS Channel Output Enable.

参数

- **channel** --
- **enable** --

返回 status

CHAPTER 4

硬件开发使用手册

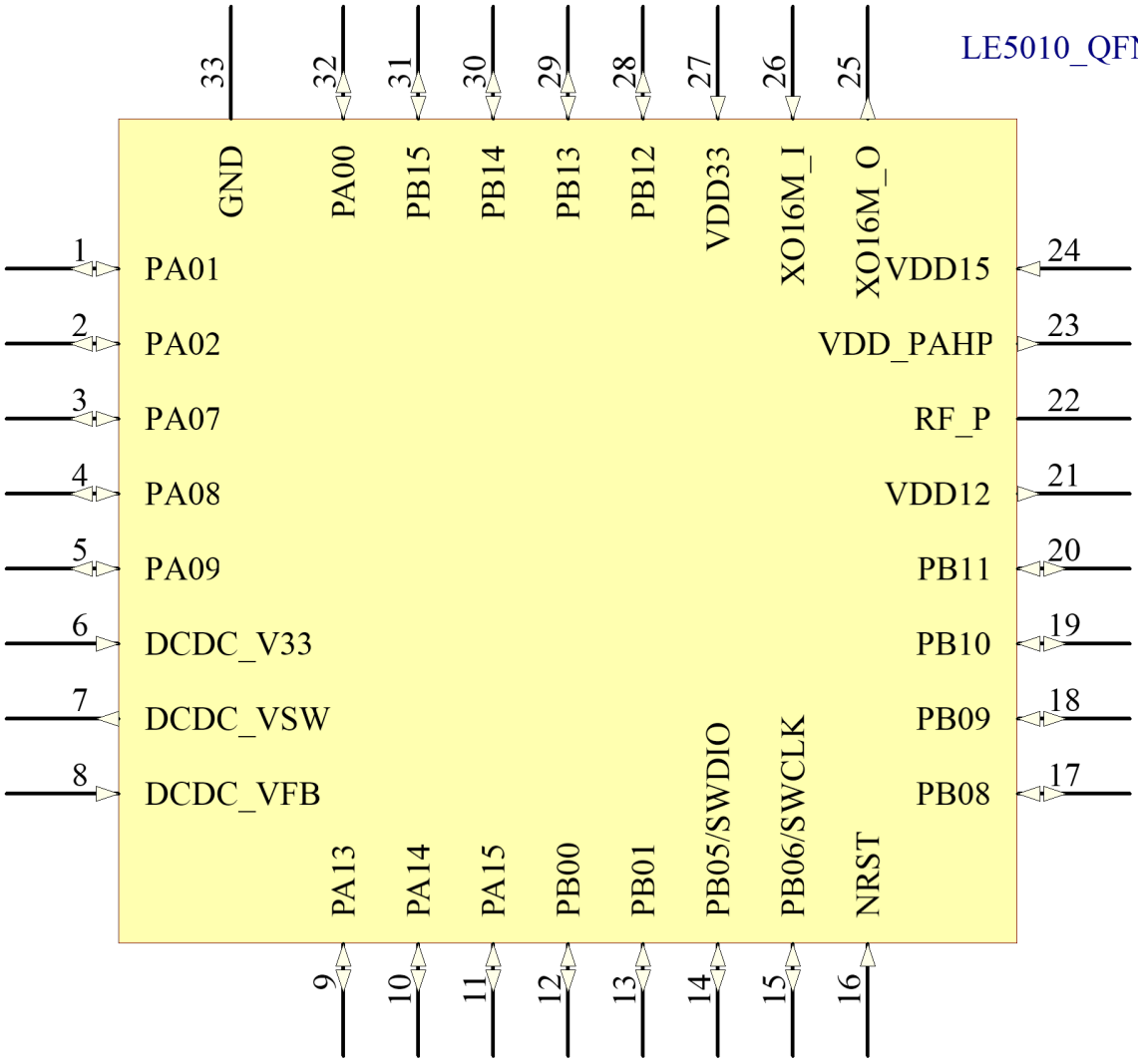
4.1 一、芯片简介

4.1.1 1.1 功能介绍

LE5010 芯片支持 SIG MESH，天猫 MESH 和私有 MESH，供电电压在 1.8V-3.6V，可以使用干电池或者对应电压的锂电池供电。

4.1.2 1.2 引脚定义图

QFN32



GPIO 具有全功能映射，且映射成数字功能的电平跟随系统输入电压。

QFN32 管脚定义：

引脚编号	名称	功能
1	PA01	IO /ADC5
2	PA02	IO /ADC6
3	PA07	IO /WKUP
4	PA08	IO
5	PA09	IO
6	DCDC_V33	Buck 3.3V 电源输入
7	DCDC_VSW	Buck SW 输出
8	DCDC_VFB	Buck 反馈电压

下页继续

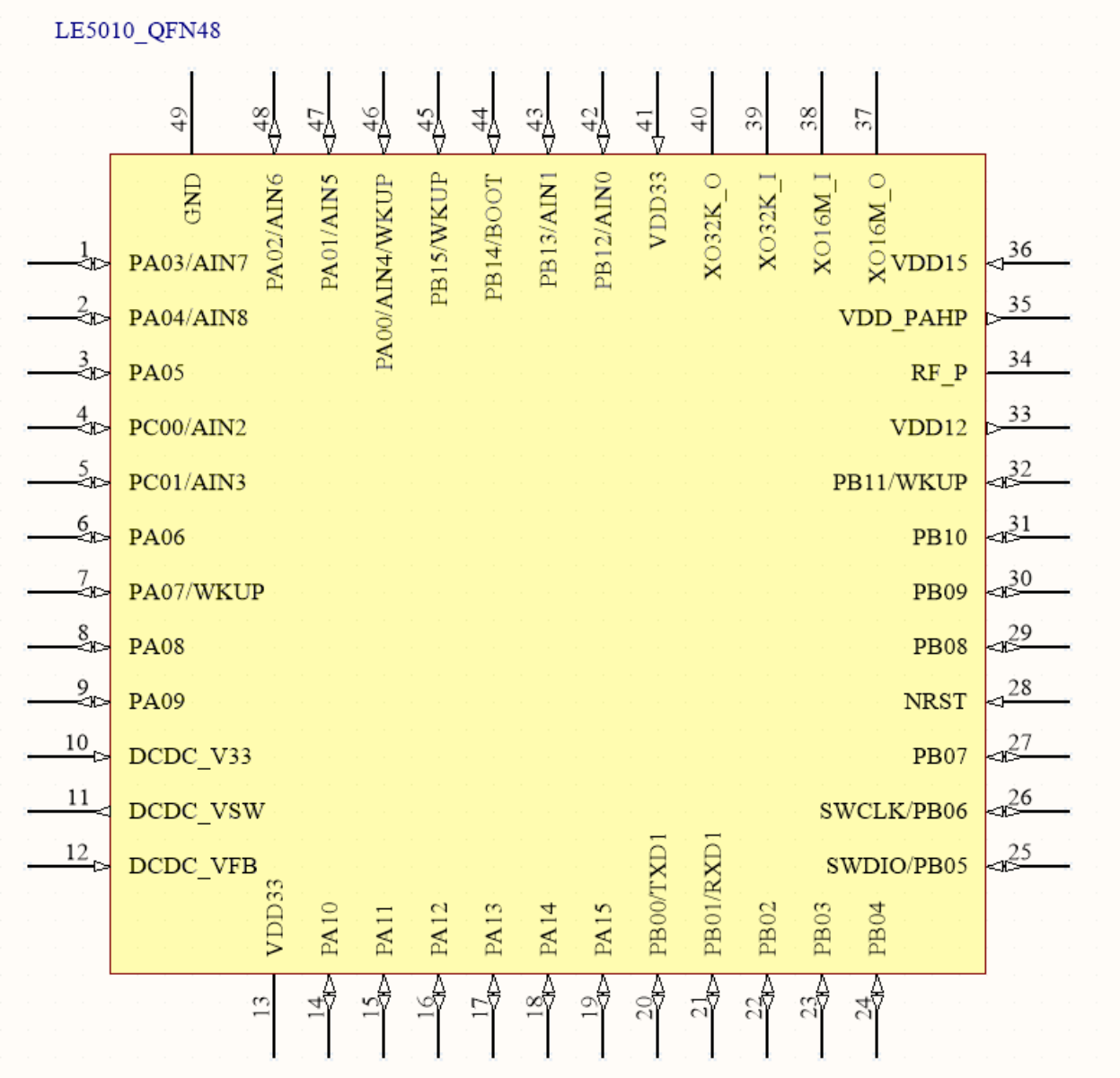
表 1 – 续上页

引脚编号	名称	功能
9	PA13	IO
10	PA14	IO
11	PA15	IO
12	PB00	IO /UART1_TX
13	PB01	IO /UART1_RX
14	PB05	IO /SWDIO
15	PB06	IO /SWCLK
16	NRST	复位引脚
17	PB08	IO
18	PB09	IO
19	PB10	IO
20	PB11	IO /WKUP
21	VDD12	1.2V 电源
22	RF_P	射频引脚
23	VDD_PAHP	PAHP 电源*1*
24	VDD15	1.5V 电源输入
25	XO16M_O	16M 晶振输出
26	XO16M_I	16M 晶振输入
27	VDD33	3.3V 供电电源
28	PB12	IO /ADC0
29	PB13	IO /ADC1
30	PB14	IO
31	PB15	IO /WKUP
32	PA00	IO /ADC4 /WKUP
33	GND	地

注：

1、PAHP 电源，TX 功率大于 10dBm 时，需要在 VDD_PAHP(PIN23) 外部添加一个 1uF 的滤波电容

QFN48



GPIO 具有全功能映射。

QFN48 管脚定义：

引脚编号	名称	功能
1	PA03	IO /ADC7
2	PA04	IO /ADC8
3	PA05	IO
4	PC00	IO /ADC2
5	PC01	IO /ADC3
6	PA06	IO

下页继续

表 2 – 续上页

引脚编号	名称	功能
7	PA07	IO /WKUP
8	PA08	IO
9	PA09	IO
10	DCDC_V33	Buck 3.3V 电源输入
11	DCDC_VSW	Buck SW 输出
12	DCDC_VFB	Buck 反馈电压
13	VDD33	3.3V 电源输入
14	PA10	IO
15	PA11	IO
16	PA12	IO
17	PA13	IO
18	PA14	IO
19	PA15	IO
20	PB00	IO /UART1_TX
21	PB01	IO /UART1_RX
22	PB02	IO
23	PB03	IO
24	PB04	IO
25	PB05	IO /SWDIO
26	PB06	IO /SWCLK
27	PB07	IO
28	NRST	芯片复位引脚
29	PB08	IO
30	PB09	IO
31	PB10	IO
32	PB11	IO /WKUP
33	VDD12	1.2V 电源
34	RF_P	射频引脚
35	VDD_PAHP	PAHP 电源*2*
36	VDD15	1.5V 电源
37	XO16M_O	16M 晶振输出
38	XO16M_I	16M 晶振输入
39	XO32K_I	32.768K 晶振输入
40	XO32K_O	32.768K 晶振输出
41	VDD33	3.3V 电源输入
42	PB12	IO /ADC0
43	PB13	IO /ADC1

下页继续

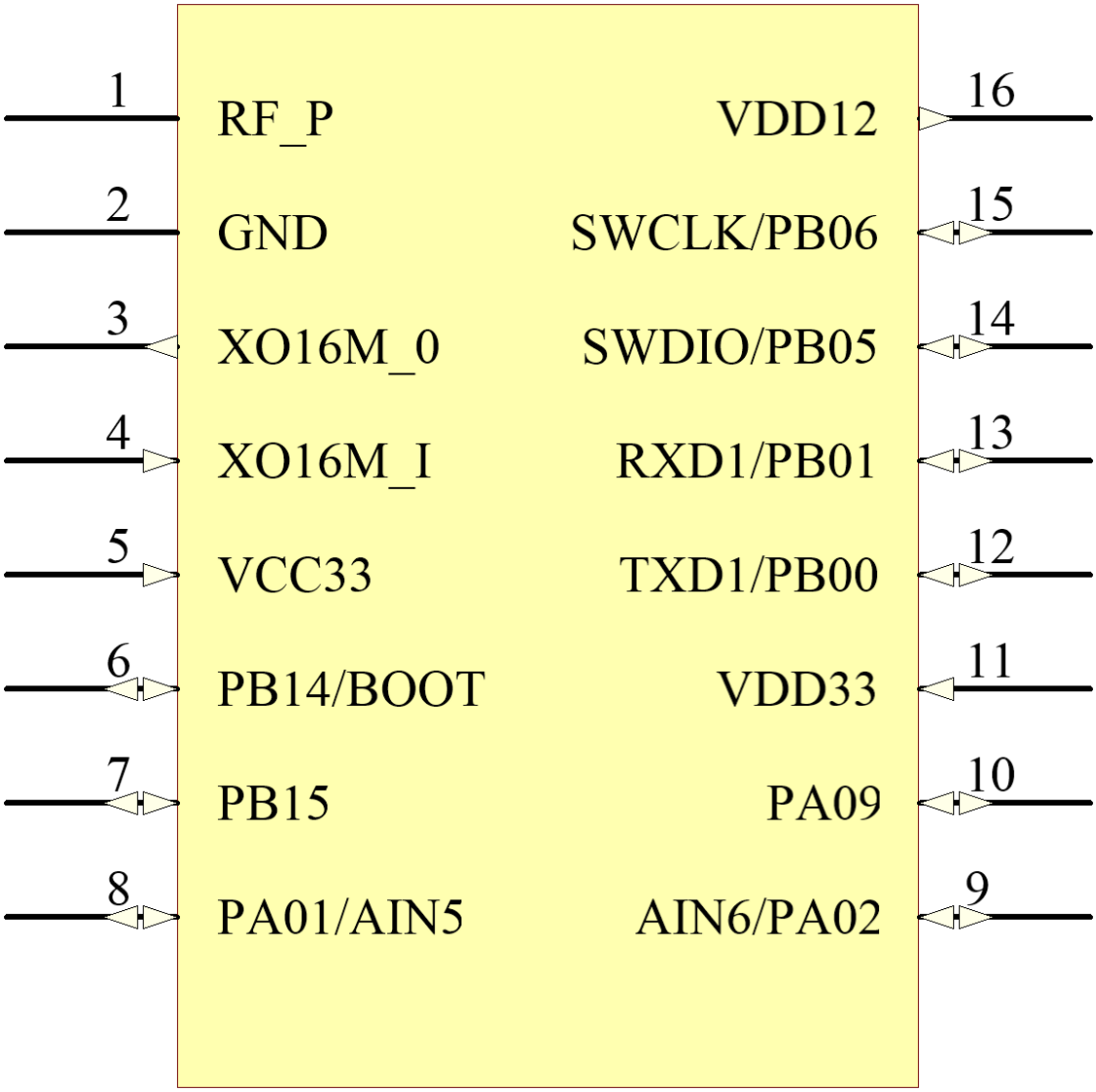
表 2 – 续上页

引脚编号	名称	功能
44	PB14	IO /BOOT 控制
45	PB15	IO /WKUP
46	PA00	IO /WKUP /ADC4
47	PA01	IO /ADC5
48	PA02	IO /ADC6
49	GND	地

注：

2、PAHP 电源，TX 功率大于 10dBm 时，需要在 VDD_PAHP(PIN23) 外部添加一个 1uF 的滤波电容

SOP16



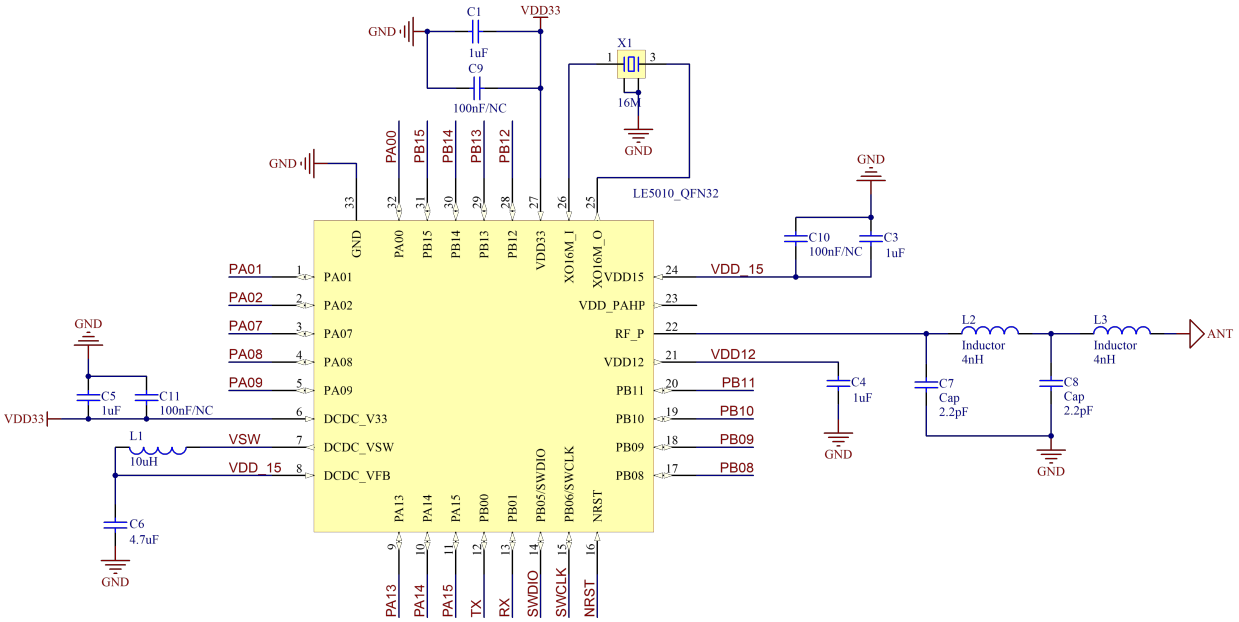
GPIO 具有全功能映射。

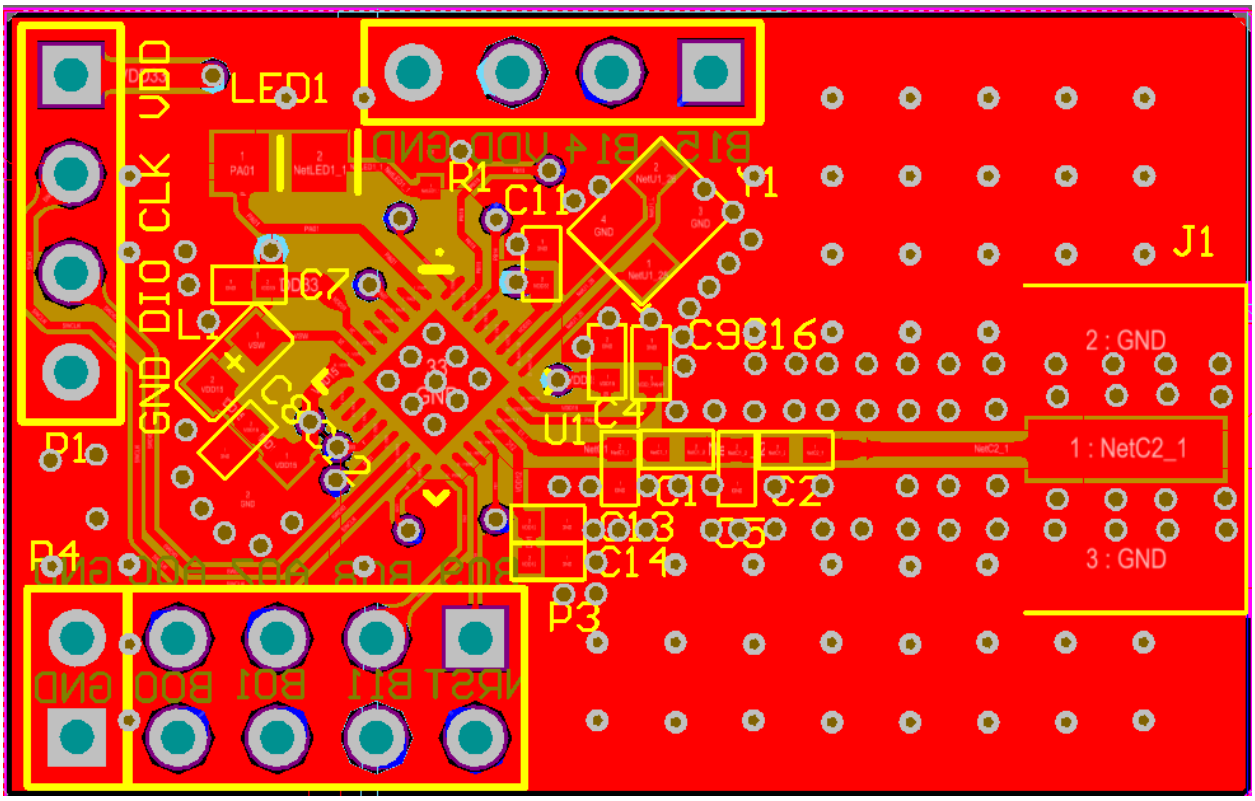
SOP16 管脚定义：

引脚编号	名称	功能
1	RF_P	射频引脚
2	GND	地
3	XO16M_O	晶振接口
4	XO16M_I	晶振接口
5	VDD33	3.3V 电源输入
6	PB14	IO
7	PB15	IO /WKUP
8	PA01	IO /ADC5
9	PA02	IO /ADC6
10	PA09	IO
11	VDD33	3.3V 电源输入
12	PB00	IO /UART1_TX
13	PB01	IO /UART1_RX
14	PB05	IO /SWDIO
15	PB06	IO /SWCLK
16	VDD12	1.2V 电源

4.2 二、参考系统设计

QFN32 原理图 (1)





注:

1、*NRST* 为复位引脚，低电平复位，复位后需要将其释放，芯片才能正常工作

2、*PB00* 和 *PB01* 上电后默认为 *UART1* 的 *TX*、*RX* 接口，可在程序中更改功能

3、在使用 *UART* 烧录时，需要将 *PB14* 强制拉高

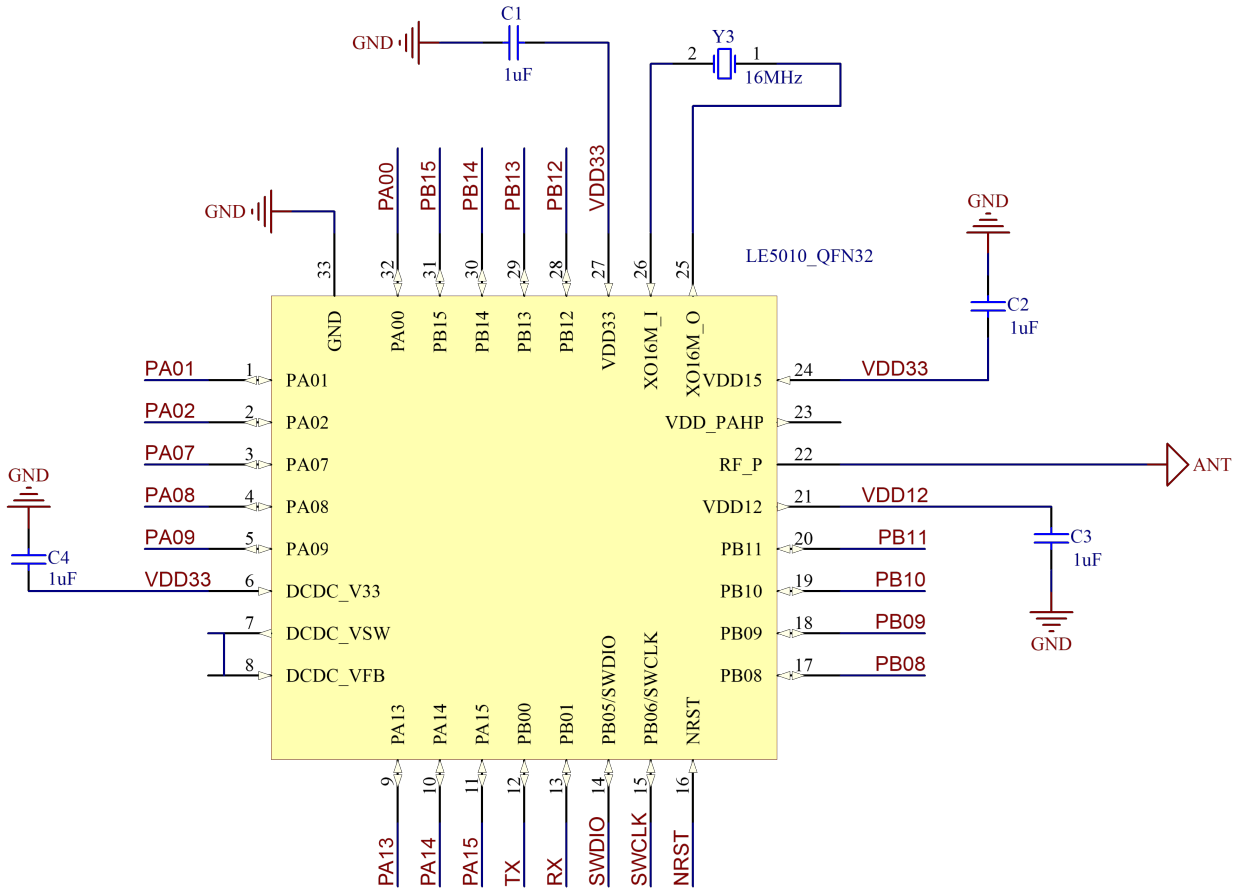
4、匹配电路数值仅供参考，需针对不同的 *PCB* 进行微调

5、容值较小的电容需要更靠近芯片的 *PIN* 脚，同时电源线需要尽量避开 *PWM* 信号线

BOM 表:

位号	封装规格	数量	备注
U1	LE5010(QFN32 4*4)	1	LE5010 凌思微电子
Y1	16MHZ 10PPM 9pF/3225	1	（推荐）SX32Y016000L91T-UZ 泰晶
C6	(±10%)/10V/4.7uF/0402	1	
C1、C2、C3、C4、C6	(±10%)/10V/1uF/0402	4	
C9、C10、C11	(±10%)/10V/100nF/0402	3	
L1	(±10%)/100mA/10uH/0603	1	
C7、C8	(±10%)/10V/2.2pF/0402	2	
L2、L3	(±0.3nH)/1A/4nH/0402	2	

QFN32 原理图 (2)



注：

1、*NRST* 为复位引脚，低电平复位，复位后需要将其释放，芯片才能正常工作

2、*PB00* 和 *PB01* 上电后默认为 *UART1* 的 *TX*、*RX* 接口，可在程序中更改功能

3、在使用 *UART* 烧录时，需要将 *PB14* 强制拉高

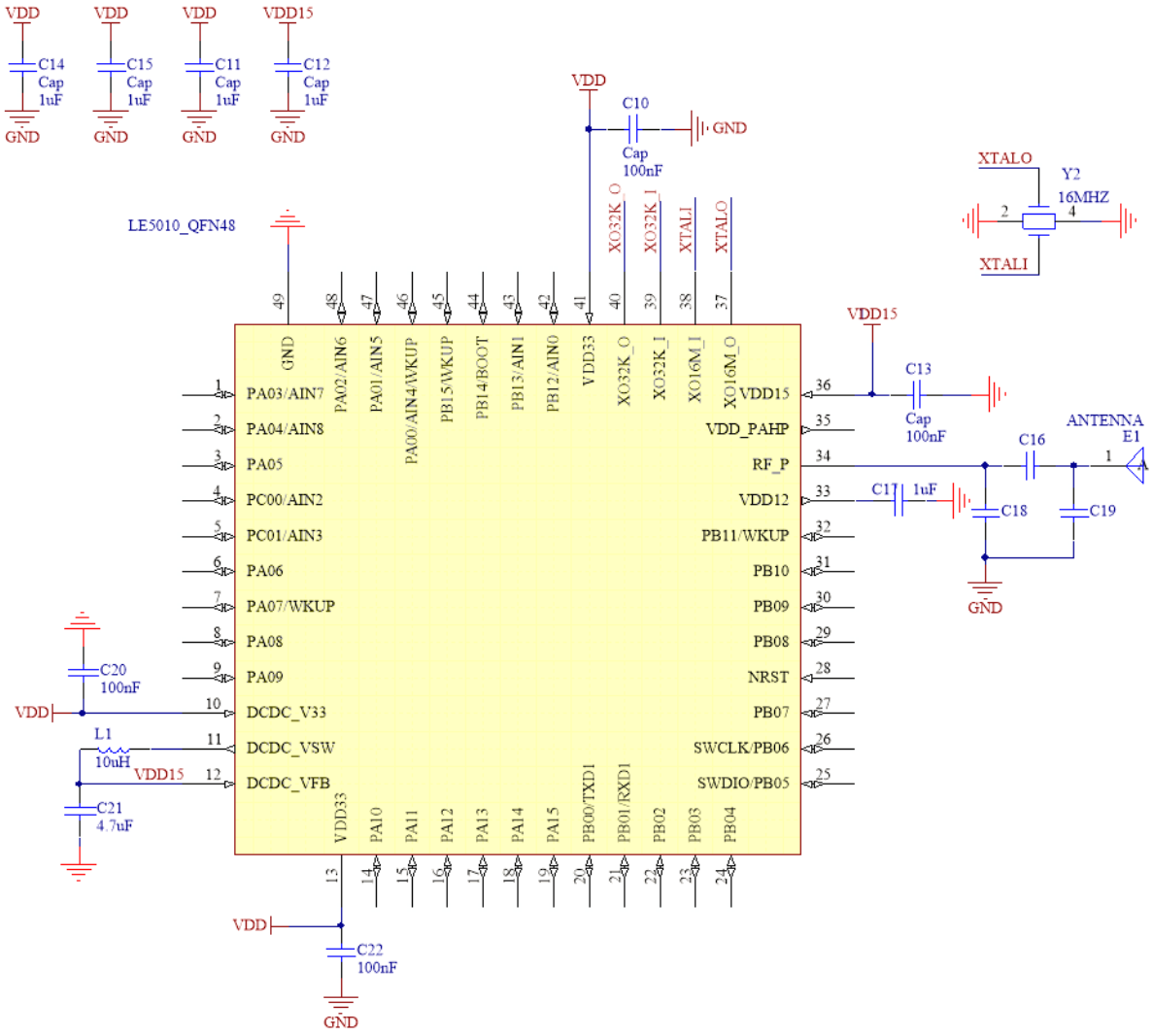
4、匹配电路数值仅供参考，需针对不同的 *PCB* 进行匹配

5、容值较小的电容需要更靠近芯片的 *PIN* 脚，同时电源线需要尽量避开高频信号线

BOM 表：

位号	封装规格	数量	备注
U1	LE5010(QFN32 4*4)	1	LE5010 凌思微电子
Y1	16MHZ 10PPM 9pF/3225	1	（推荐）SX32Y016000L91T-UZ 泰晶
C1、C2、C3、C4	(±10%)/10V/1uF/0402	4	

QFN48 原理图



注：

1、*NRST* 为复位引脚，低电平复位，复位后需要将其释放，芯片才能正常工作

2、*PB00* 和 *PB01* 上电后默认为 *UART1* 的 *TX*、*RX* 接口，可在程序中更改功能

3、在使用 *UART* 烧录时，需要将 *PB14* 强制拉高

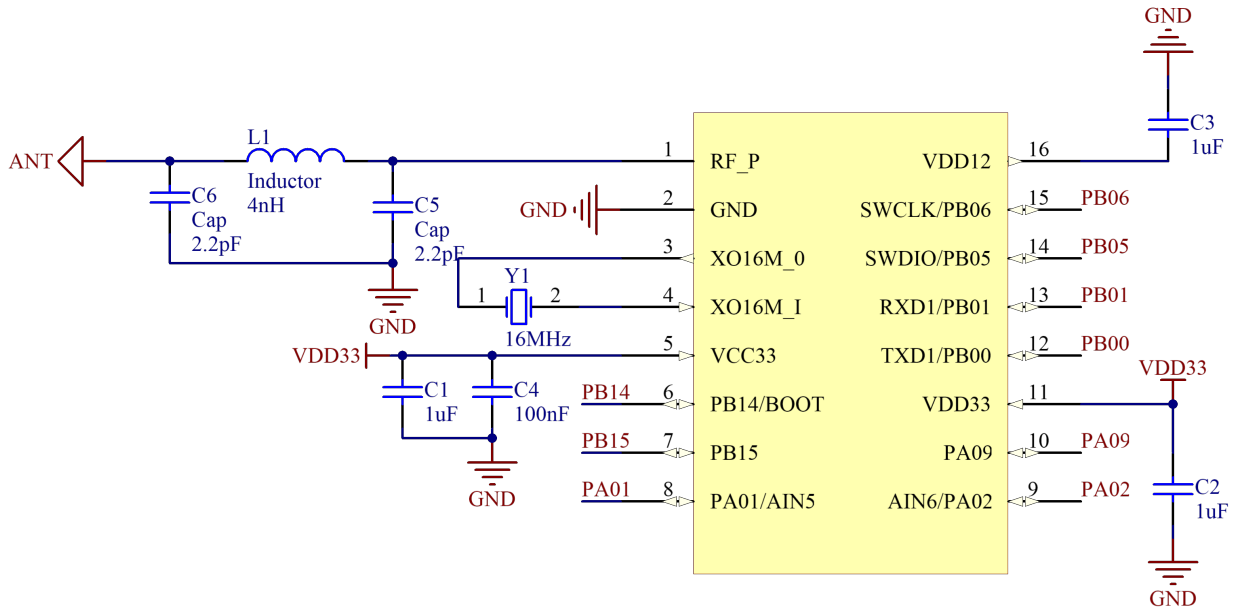
4、匹配电路数值仅供参考，需针对不同的 *PCB* 进行微调

5、容值较小的电容需要更靠近芯片的 *PIN* 脚，同时电源线需要尽量避开 *PWM* 信号线

BOM 表：

位号	封装规格	数量	备注
U1	LE5010(QFN32 4*4)	1	LE5010 凌思微电子
Y2	16MHZ 10PPM 9pF/3225	1	（推荐）SX32Y016000L91T-UZ 泰晶
Y1	32KHZ 10PPM 9pF/3225	1	
C5	(±10%)/10V/4.7uF/0402	1	
C10、C13、C20、C22	(±10%)/10V/100nF/0402	4	
C11、C12、C14、C15	(±10%)/10V/1uF/0402	4	
L1	(±10%)/100mA/10uH/0603	1	

SOP16 原理图



注：

1、PB00 和 PB01 上电后默认为 UART1 的 TX、RX 接口，可在程序中更改功能

2、在使用 UART 烧录时，需要将 PB14 强制拉高

3、PIN5 和 PIN11 都必须外接 3.3v 电源，但在接线时不能直接短接，中间的线路必须先经过电容，在接入 PIN 脚

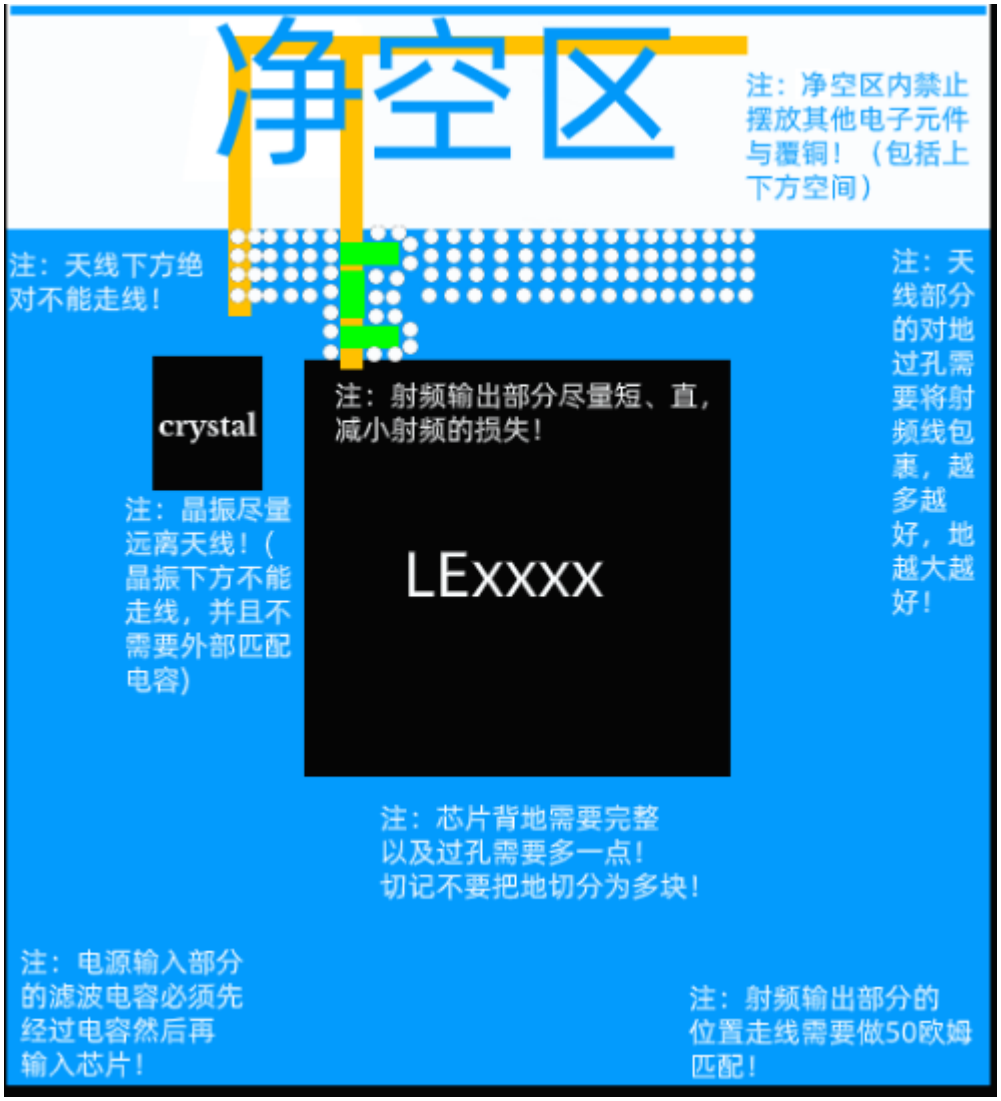
4、匹配电路的数值仅供参考，具体数值需要根据不同的 PCB 进行匹配

BOM 表：

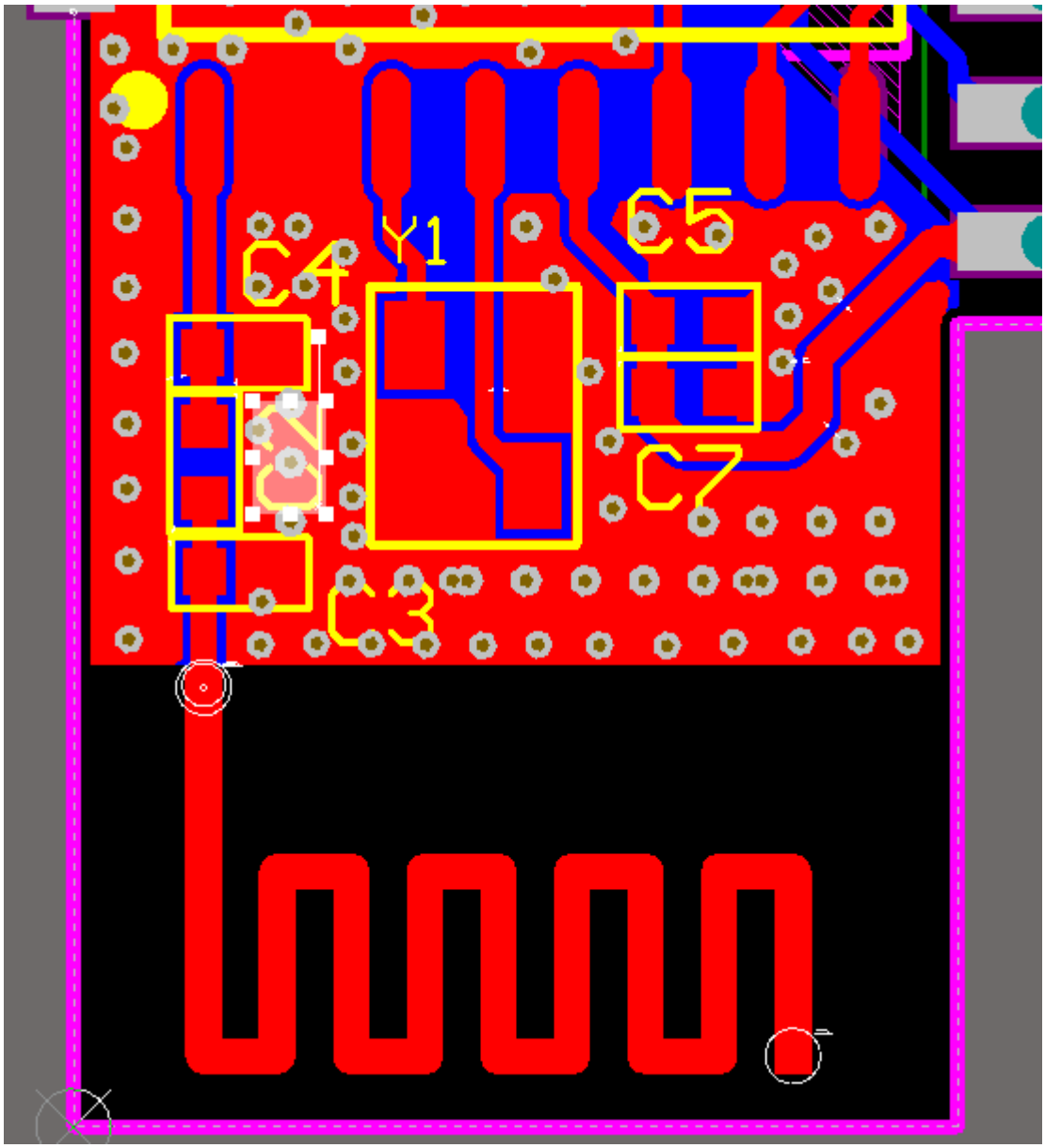
位号	封装规格	数量	备注
U1	LE5010(SOP16)	1	LE5010 凌思微电子
Y1	16MHZ 10PPM 9pF/3225	1	（推荐）SX32Y016000L91T-UZ 泰晶
C1、C2、C3	(±10%)/10V/1uF/0402	3	
C4	(±10%)/10V/100nF/0402	1	
L1	(±0.3nH)/1A/4nH/0402	1	
C5、C6	(±10%)/10V/2.2pF/0402	2	

4.3 三、LE5010/5110 PCB 注意事项

4.3.1 LE5010/5110 PCB 注意事项

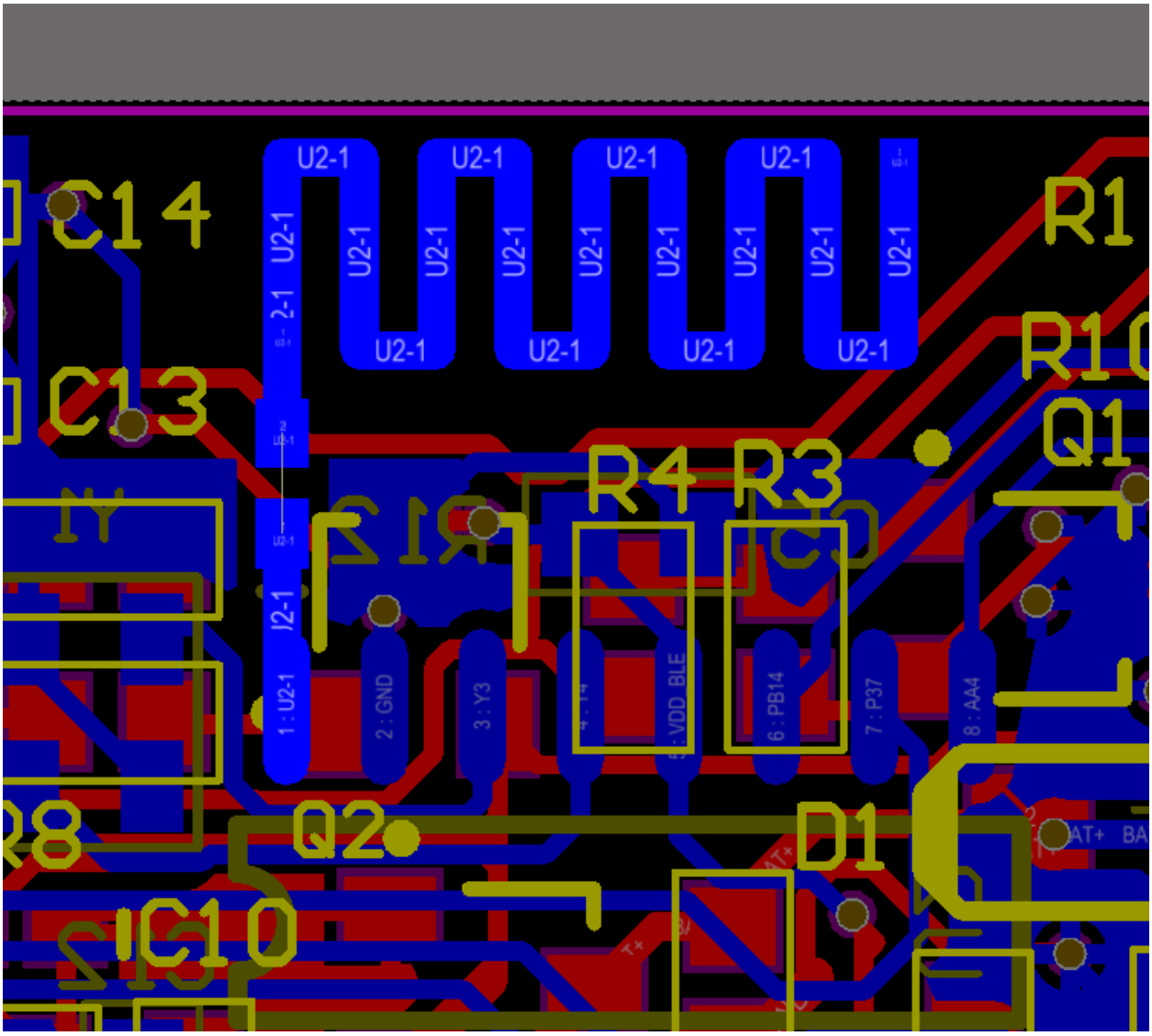


一、天线注意事项



注解： 1、使用板载天线的时候，必须要求留足够的净空区，且在净空区中不能有其他的元器件或者覆铜。

2、设计匹配电路的时候，尽量紧靠，天线与芯片的直接距离尽量近，且走线直、短。



1.1 上图的错误点：

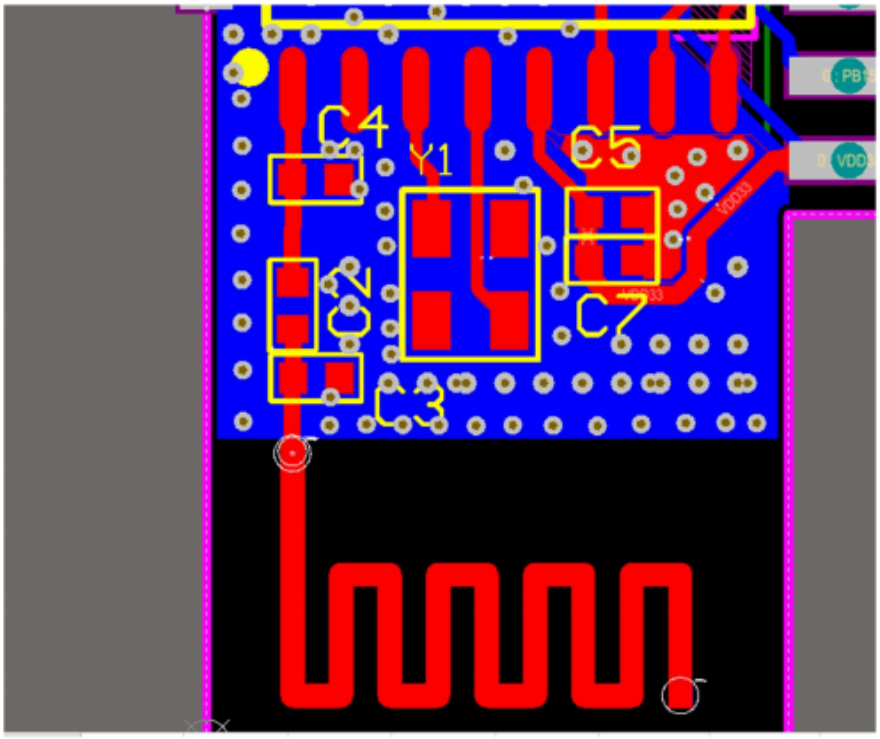
(1)：天线的净空区不够，且有走线，走线会吸收掉部分辐射出来的信号，导致天线的辐射效率变低。

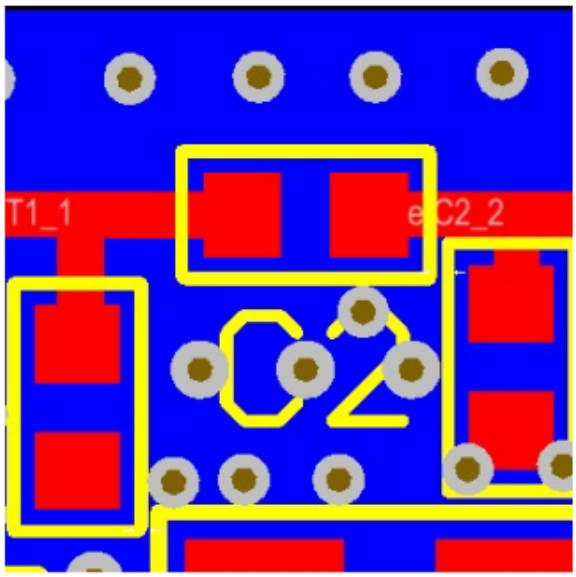
(2)：天线到芯片的 PIN 走线下方有许多杂线，会干扰射频的工作，导致发射出来无线信号质量差。

(3)：没有做匹配电路，只预留了一个电阻为作为测试点，最小的匹配电路为 L 型的电路，单个器件调节阻抗能力有限，很难达到理想的效果。

(4)：由于空间问题，没有留下足够的地，可能导致驻波比很差。

1.2 匹配电路布局注意事项





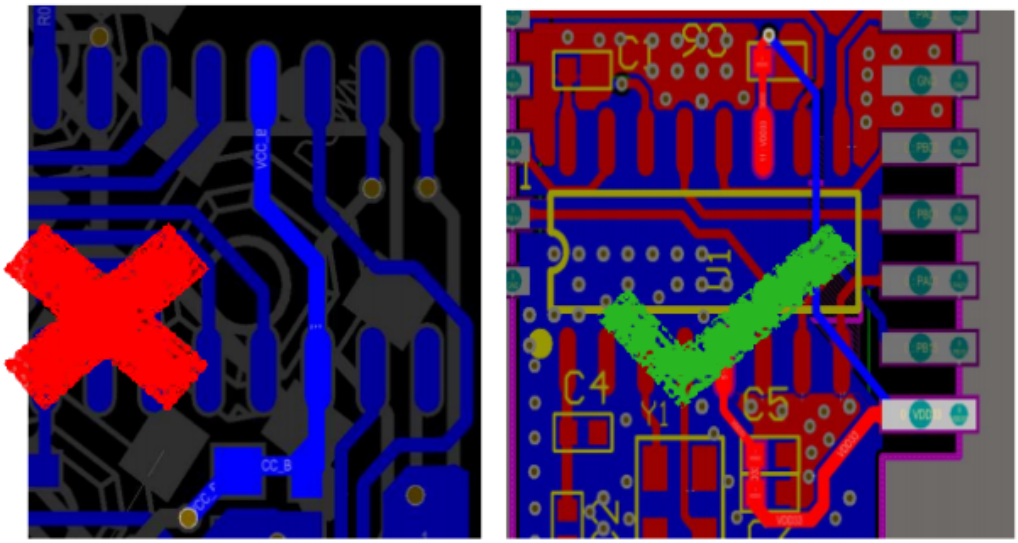
上述两幅图的主要问题是在匹配电路上面，图 1 匹配不够紧靠，会导致走线的阻抗不连续，增加了后续匹配的复杂度。图 2 出现了如果树木枝桠的走线，称为 stub，一旦形成 stub，阻抗一定是不连续，无论是天线或者其他场景的布局走线中，一定要避免出现这种情况。

二、电源的注意事项

2.1 QFN32 和 SOP16 的封装都有两个 3.3V 的电源输入，在靠近电源 pin 脚的位置上面，需要各自放一个对地电容，再接线进入 PIN 脚。如果有过孔，要尽量避免电容出现 stub 的情况，中间需要经过对地的电容，进行滤波处理，再进入目标的电源 PIN 脚，防止产生串扰影响。

2.2 电源在进入 PIN 脚之前，一定要有对地的电容滤波，且经过电容后，直接进入对应的 PIN，如果是 2 层板，有高频信号线与电源线相交，需再增加一个对地电容，然后再接入对应的电源引脚。

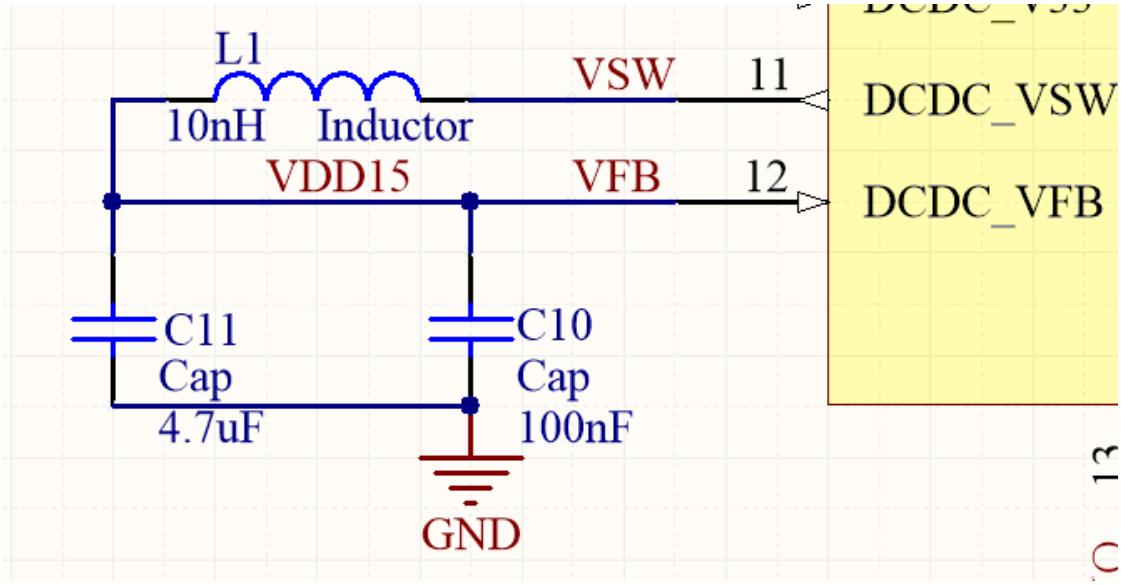
2.3 QFN48 上面有 3 个电源引脚，处理时，电源网络尽量星型分布，同时产生的环路尽量小。



图（5）：左 NG，因为直接将两个 PIN 短接在一起
右 OK,电源先经过背地电容，在进入 PIN 脚。

三、DCDC 电路

3.1 DCDC 电路的主要结构如下所示：SW 为方波输出信号，经过 LC 滤波之后，稳定在 1.5V 左右，给 VDD15 供电。



3.2):SOP16 封装默认没有 DCDC 的 PIN 脚，QFN32 和 QFN48 封装的 DCDC 电路可以忽略，只要将对应的 VFB 和 SW 短接，同时 VDD15 接 3.3V 电源输入即可。

四、地的覆铜

(4.1): 针对 QFN32 和 QFN48 的封装的背面，一定要和 PCB 的 GND 连接在一起，形成回路。

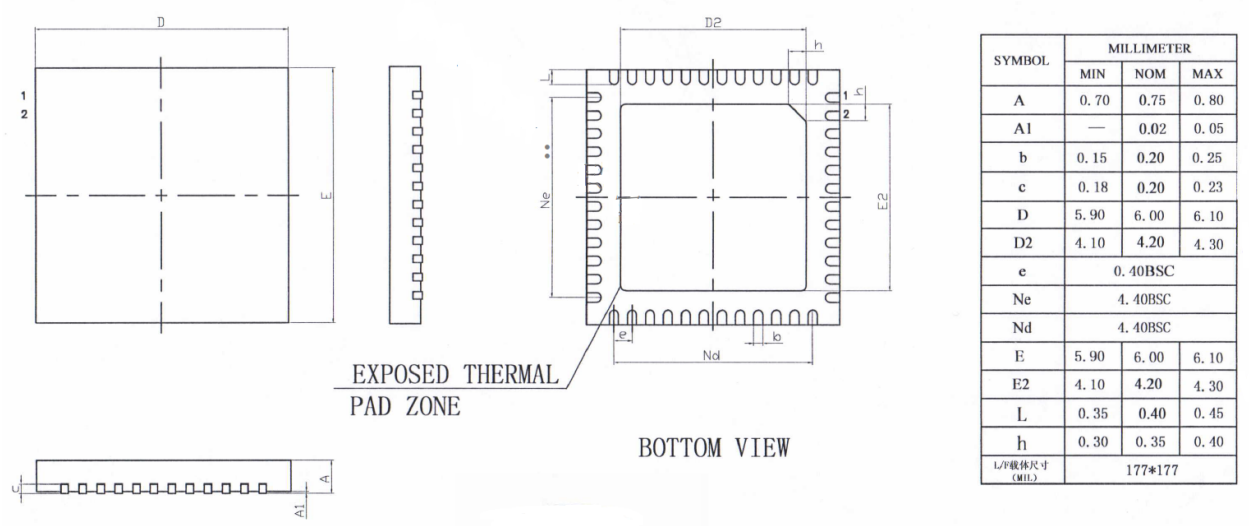
(4.2):PCB 布局的背面，尽量保证完整的地，不能出现死铜。

(4.3): 天线与芯片的 PIN 脚连接下方的地必须完整，不能有走线，或者其他东西，连接线周围也是同理，而且面积越大越好。

4.4 四、封装尺寸

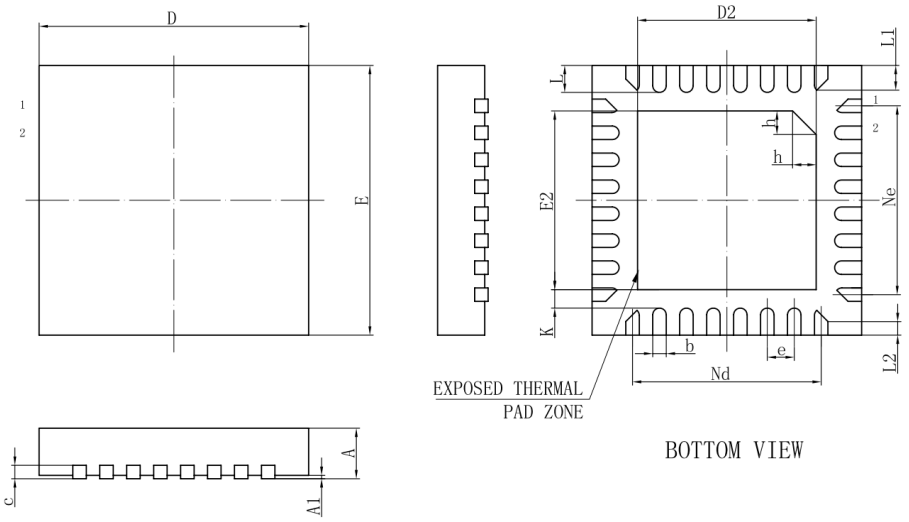
外形尺寸:

QFN48 尺寸图



SYMBOL	MILLIMETER		
	MIN	NOM	MAX
A	0.70	0.75	0.80
A1	—	0.02	0.05
b	0.15	0.20	0.25
c	0.18	0.20	0.23
D	5.90	6.00	6.10
D2	4.10	4.20	4.30
e	0.40BSC		
Ne	4.40BSC		
Nd	4.40BSC		
E	5.90	6.00	6.10
E2	4.10	4.20	4.30
L	0.35	0.40	0.45
h	0.30	0.35	0.40
L/F载体尺寸 (MIL)	177*177		

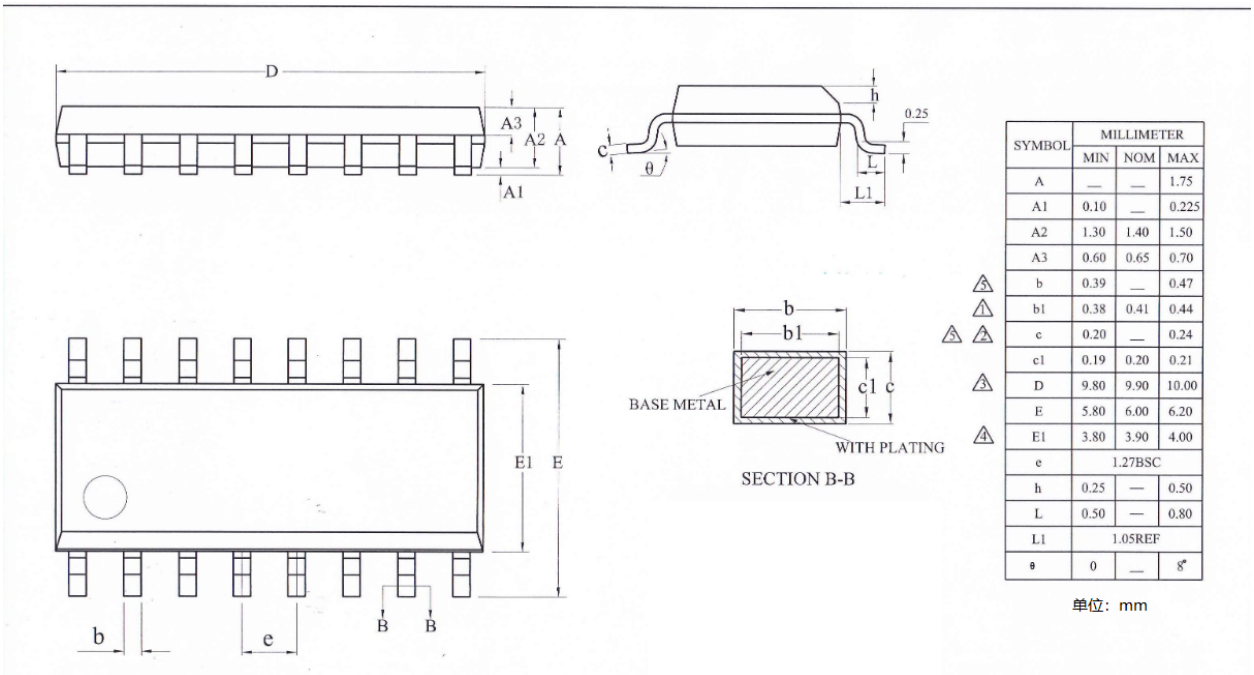
QFN32 尺寸图



SYMBOL	MILLIMETER		
	MIN	NOM	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
b	0.15	0.20	0.25
c	0.18	0.20	0.25
D	3.90	4.00	4.10
D2	2.60	2.65	2.70
e	0.40BSC		
Nd	2.80BSC		
E	3.90	4.00	4.10
E2	2.60	2.65	2.70
Ne	2.80BSC		
K	0.20	–	–
L	0.35	0.40	0.45
L1	0.30	0.35	0.40
L2	0.15	0.20	0.25
h	0.30	0.35	0.40
L/F载体尺寸 (MIL)	112*112		

单位：mm

SOP16 尺寸图



SYMBOL	MILLIMETER		
	MIN	NOM	MAX
A	—	—	1.75
A1	0.10	—	0.225
A2	1.30	1.40	1.50
A3	0.60	0.65	0.70
b	0.39	—	0.47
b1	0.38	0.41	0.44
c	0.20	—	0.24
c1	0.19	0.20	0.21
D	9.80	9.90	10.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e	1.27BSC		
h	0.25	—	0.50
L	0.50	—	0.80
L1	1.05REF		
θ	0	—	8°

单位：mm

各种封装原理图和 PCB 参考示例可以从百度网盘里面下载：

链接：https://pan.baidu.com/s/1iPL2JWaDdYJRqNdGMwiK6w

提取码：ijvd

CHAPTER 5

在线烧录工具使用

5.1 一、烧录工具获取

上位机烧录软件可以从百度网盘里面下载：

链接：https://pan.baidu.com/s/1-K3I05Q1nqbUYAt-7_443Q

提取码：3we6

5.1.1 1.1 文件说明

压缩包内包含两个文件夹，`DownloadTool` 内工具用于下载用户 Image 到芯片内，`Excel2BinTool` 内工具将天猫精灵工程所用的三元组的 Excel 文件转化成二进制文件，打开对应的.exe 文件即可使用相关工具。

5.2 二、操作说明

5.2.1 2.1 config.ini 文件配置

`DownloadTool\config.ini` 文件主要用于下载之前的一些配置

2.1.1 蓝牙地址生成模式

[ADDRESS]: 配置蓝牙的设备地址，烧录到 flash 的地址可以通过 WRITE_FLASH_ADDR 配置，`RANDOM=0` 时，MAC 地址可以滚码烧录，起始地址为 START_ADDRESS 的值，`RANDOM=1` 时，烧录的地址为随机地址。只有在烧录界面勾选了蓝牙地址选项该项配置才会生效。第一次打开烧录软件时，会在 program 和 FlashDownload_Addr 目录下生成一个保存有 1600 个 MAC 地址的 address.csv 文件。如果修改了 config.ini 文件中地址生成方式，需要关闭烧录软件，将这两个目录删除，重新打开才可生成新的地址文件。如果不选择烧录 MAC 地址，代码中也会分配一个随机地址。

2.1.2 串口黑名单

[COM]: 配置过滤串口选项，当电脑中存在不是用来烧录的串口时可将其配置到黑名单中，这样烧录软件扫描串口时则会自动忽略该串口，不会在烧录界面占用一个位置。例如：把 COM8 和 COM10 禁掉，COM_BLACKLIST=8,10

2.1.3 下载配置

[DOWNLOAD]:

配置默认波特率和是否需要下载 MAC 地址，`BAUDRATE=6` 表示默认波特率为 460800;

`BLE_ADDR=0` 时，表示启动烧录软件默认不烧录 MAC 地址，`BLE_ADDR=1` 时，默认选择烧录 MAC 地址;

`TG7121B_TRIPLE_STORAGE = 0` 时，表示为 LE5010 芯片烧录，`TG7121B_TRIPLE_STORAGE = 1` 时，表示 TG7121B 芯片烧录。

5.2.2 2.2 下载操作

1. 断开所有连线，将 uart 串口工具的 RXD Pin 接芯片的 PB00,TXD Pin 接芯片的 PB01,3.3V 接芯片的 VDD，GND 接 GND，再将芯片的 PB14 拉高；
2. 将串口工具连接电脑给芯片上电，必须在第一步的接线完成之后再给芯片上电；
3. 打开 download.exe，选择固件，导入要下载的 hex 文件，如果不是天猫精灵的工程，则不需要导入三元码文件，对于天猫精灵的工程需要先将三元组的 Excel 文件通过 Excel2Bin 工具转换成 bin 文件，然后导入到三元码对应的选项框；
4. 下载工具界面选择相应的 COM 口，默认波特率为 460800，也可改成其它，如果下载过程中一直出现校验出错的情况，可以降低波特率重新烧录；
5. 点击打开按钮，打开相应的 COM 口；
6. 烧录成功之后，下载的进度条为 100%，下载状态显示下载完成；

7. 如果烧录失败，在串口工具不掉电的情况下，重新给芯片上电，或者通过拉低 NRST pin 复位芯片，复位后一定要将 NRST 释放，否则无法正常工作。上位机烧录软件不需要操作；在烧录软件打开串口后，串口工具意外断电，烧录软件上需要关闭串口后再重新打开串口；

8. 去掉 PB14 的拉高，重新上电或复位便可运行烧录的程序。

注：芯片刚启动的时候 BootRom 中会去检测 PB14 的状态，如果处于拉高，则会进入到 UART Download Mode，否则会去运行用户程序；下载的时候会首先对 flash 做一次全擦除操作，每烧录 256 字节都会做 CRC 校验，只有检验通过才会继续烧入，直到烧录完成.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

索引

符号

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V