SV32WB0x 系列

SDK 入门指南

Nov 2018

关于本手册

本手册介绍了 SV32WB0x 系列的 SDK 入门指南

发布说明:

版本	日期	编辑	描述
V1.0	2021.1	Andy	首次发布。
V1.1	2021.3	Andy	增加 Secure Boot 说明
V1.2	2021.5	Andy	增加 BLE 开发说明
V1.3	2021.8	Andy	说明同步至 WSDK.20Q2.2127.01 版本

- 文档变更通知&证书下载：

本文档更新不会逐一通知，用户需要使用时请自行去南方硅谷官网上下载最新版资料；需要相关证书的用户请联系南方硅谷客服 请知悉！

目录

1. 概述

1.1. 支持功能概述

SV32WB0x SDK 支持以下系列的芯片，并且所有芯片支持 DSP/FPU/BLE 等功能

Table 30: SV32WB0xx Part Number

Part number	Category	Architecutre	Wireless Type	CPU CLKxSRAMxFlashxPackage		Feature	Operating Temp(°C)
SV32WB01	MCU/IoT	32 bit core	WiFi+BLE	320MHz x 384KB	16Mb x QFN32		-40 to +85
SV32WB01-L	MCU/IoT	32 bit core	WiFi+BLE	320MHz x 384KB	16Mb x QFN32	Lower power	-40 to +85
SV32WB01-T	MCU/IoT	32 bit core	WiFi+BLE	480MHz x 512KB	16Mb x QFN32	Turbo	-40 to +85
SV32WB01-H	MCU/IoT	32 bit core	WiFi+BLE	320MHz x 384KB	16Mb x QFN32	High Temp	-40 to +105
SV32WB05	MCU/IoT	32 bit core	WiFi+BLE	320MHz x 384KB	16Mb x QFN40		-40 to +85
SV32WB06	MCU/IoT	32 bit core	WiFi+BLE	480MHz x 512KB	NO x QFN60		-40 to +85
SV32WB07	MCU/IoT	32 bit core	WiFi+BLE	480MHz x 512KB	16Mb x QFN40		-40 to +85

1.2. 流程概述

SDK 的使用流程如图 1-1 所示

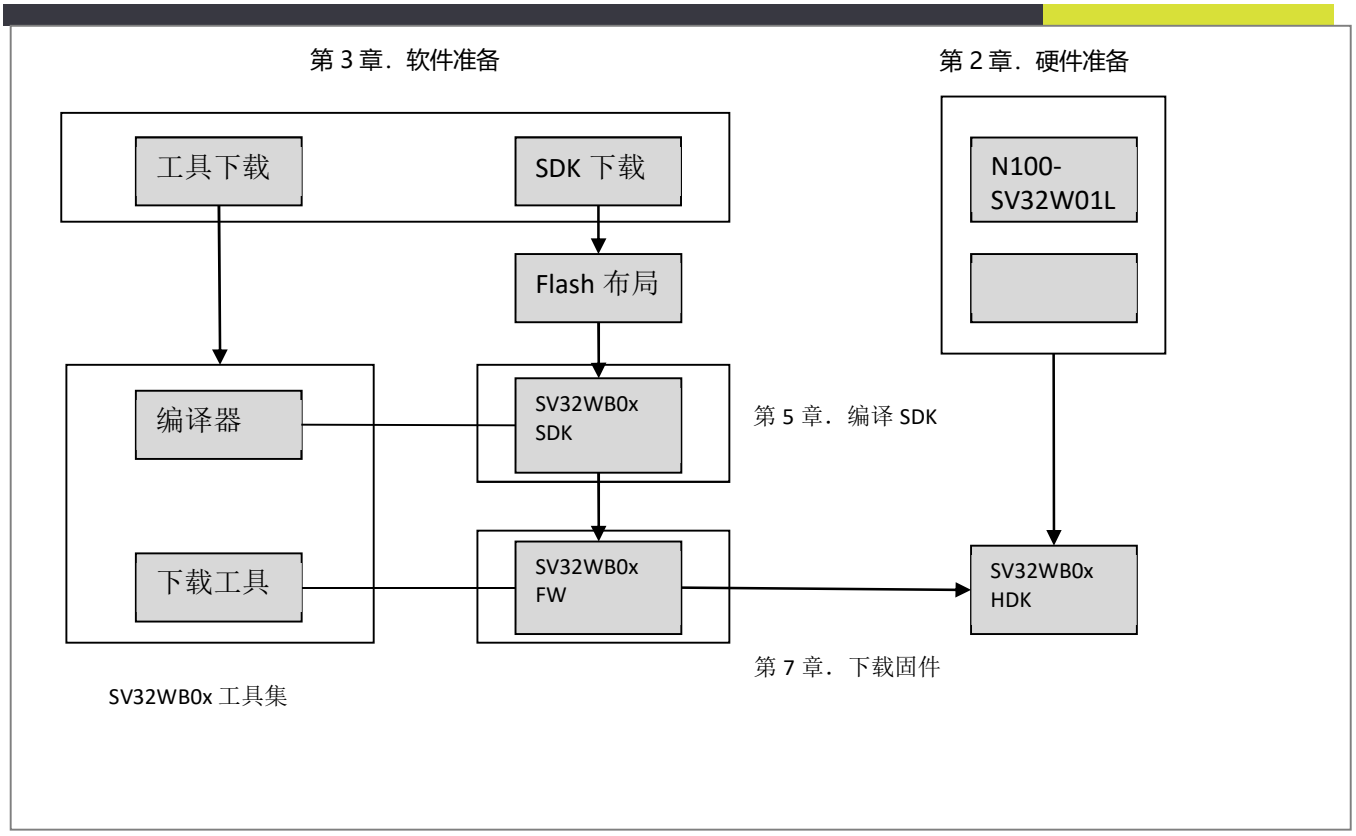


图 1-1

1.3. SV32WB0x SDK

SV32WB0x Software Development Kit (SDK) 是南方硅谷为开发者基于 FreeRTOS 操作系统提供的物联网(IoT) 应用开发平台，包括基础平台以及上层应用开发示例，如智能灯、智能开关等。SV32WB0x SDK 提供以下功能：

- 使用 FreeRTOS 系统，引入 OS 多任务处理的机制，用户可以使用FreeRTOS 标准接口实现资源管理、循环操作、任务内延时、任务间信息传递和同步等面向任务流程的设计方式。具体接口使用方法参考 FreeRTOS 官方网站的使用说明或者 USING THE FreeRTOS REAL TIME KERNEL—A Practical Guide 介绍。
- 提供网络操作接口 lwIP API，同时提供了 BSD Socket API 接口的封装实现，用户可以直接按照 Socket API 的使用方式来开发软件应用，也可以直接编译运行其他平台的标准 Socket 应用，有

效降低平台切换的学习成本。

- 引入了 cJSON 库，使用该库函数可以更加方便的实现对 JSON 数据包的解析。
- 提供 Wi-Fi 接口、 SmartConfig 接口、 Sniffer 相关接口、系统接口、定时器接口、 FOTA 接口和外围驱动接口
- 引入了 mbedtls 库，使用 ECC & SHA & AES 硬件加速引擎
- 加入 BLE 5.0 Master, Slave, Advertiser, Scanner roles 功能

1.4. SV32WB0x 工具集

1.3.1 编译器

编译 SV32WB0x SDK 需要使用 Linux 操作系统，若使用 Windows 操作系统，建议使用 VMware Workstation 作为 SV32WB0x 虚拟机。为了简化编译操作，南方硅谷已将编译 SDK 所需要的工具安装到虚拟机中。用户只需安装 VMware Workstation，并导入南方硅谷提供的镜像文件即可直接编译 SV32WB0x SDK。

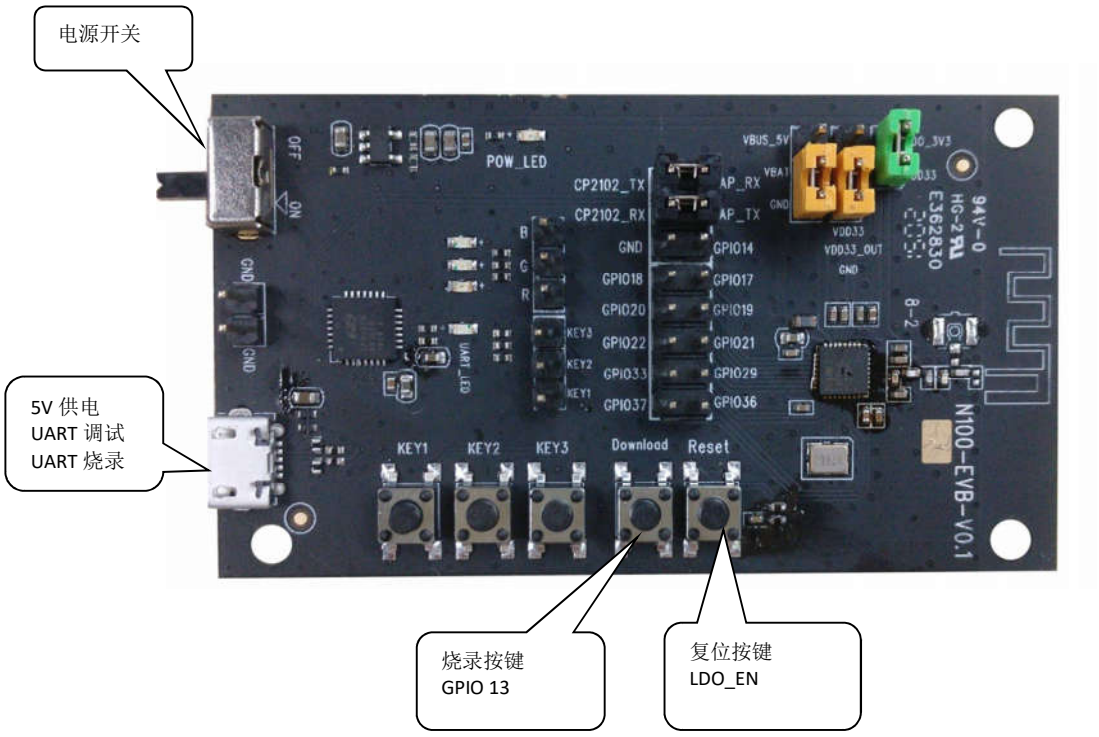
1.3.2 固件下载工具

Download_Tool_UI 工具是由南方硅谷官方开发的固件下载工具，用户可直接通过 Uart 接口，将固件直接烧录到 Flash 中

1.3.3 串口调试工具

串口调试工具可以通过标准 Uart 端口直接与 SV32WB0x 建立通信。用户可以直接在串口终端输入命令和实时查看相关打印信息。

2. 硬件准备



烧录模式

烧录步骤:

- 打开下载工具
  - 选择烧录固件的路径
  - 配置波特率，选择对应的串口
  - 打开串口，选择
  - 点击“开始烧录”开启连续烧录模式
  - 长按“烧录按键”不松开，按一次“复位按键”，MCU 重启后即可进入烧录模式，烧录器进

入烧录过程（超时 2S），此时可以松开“烧录按键“。如果无法进入烧录模式，重复此步骤

- 烧录完成后，停止烧录，关闭串口，按一次“复位按键”重启开发板，使用 SecureCRT 查看 LOG

注意：

当烧录工具不能正常运行时，检查一下烧录工具的路径是否带有中文，尝试放到 D 盘目录再运行

3. 软件准备

3.1. FreeRTOS SDK

请在南方硅谷提供的 FTP 服务器下载

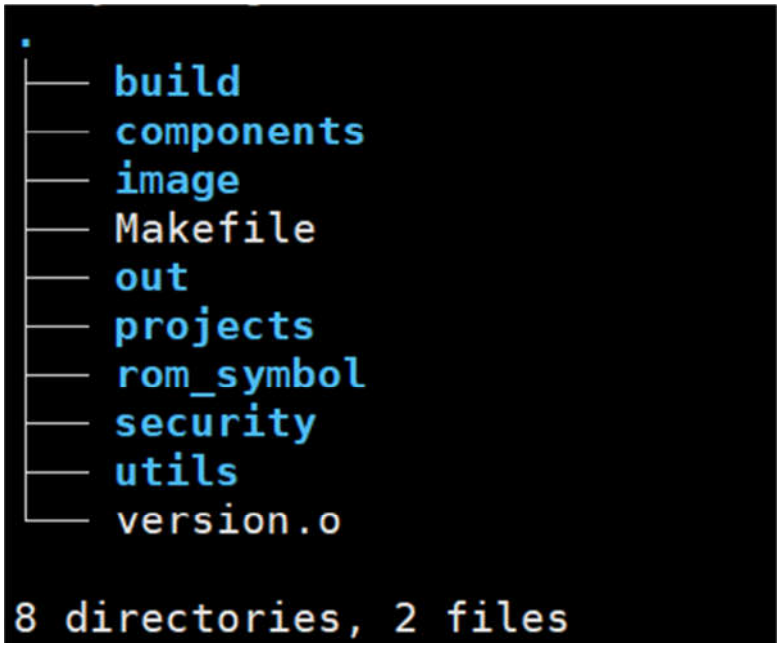


图 3-1 FreeRTOS SDK 内容

- build：编译辅助脚本
- components：组件目录，主要南方硅谷提供的静态库（如 wifi、外围驱动相关），开源库（如 lwip、cjson）
- image：编译生成目录，使用烧录工具烧录 bin 档
- out：编译生成中间文件
- utils:辅助工具
- security: security boot pem 证书
- projects：范例工程文件，目前包含以下 project
  - mac_atcmd 通用开发固件，包含所有测试 AT 命令，可用于功能演示与开发

- ultra_low_power 超低功耗专案，通过配置 feature.mk 中 SRAM--256K、MCU Clk--160MHz、flash Clk--40MHz 以达到最低省电模式。功耗测试一般使用此专案
- ble_mesh 功能演示
- mac_transparent IOT 透传专案

3.2. SV32WB0x 工具集

3.3.1 编译器

- 从南方硅谷提供的 FTP 下载 nds32le-elf-mculib-v3s(ast_324).txz 交叉编译工具
- 复制 nds32le-elf-mculib-v3s(ast_324).txz 到 Linux 系统，并解压
- Linux 中指统编译器的路径，如图 3-3-1 所示
- 验证编译器，如图 3-3-2 所示

```
114 fi
115
116 export PATH=${PATH}:/home/andy.liang/IOT/tool/nds32le-elf-mculib-v3s/bin/
117 #export PATH=${PATH}:/home/andy.liang/IOT/tool/nds32le-elf-mculib-v3m/bin/
```

图 3-3-1

```
andy.liang@ubiserver:andy$ nds32le-elf-gcc -v
Using built-in specs.
COLLECT_GCC=nds32le-elf-gcc
COLLECT_LTO_WRAPPER=/home/andy.liang/iot/tool/nds32le-elf-mculib-v3s/bin/../libexec/gcc/nds32le-elf/4.9.4/lto-wrapper
Target: nds32le-elf
Configured with: /NOBACKUP/sqa15/sqa/build-ast322/V3_rebuild/build-system-3/source-packages/gcc-4.9.4-bsp-v4_2_0-branch/configure --target=nds32l
e-elf --prefix=/NOBACKUP/sqa15/sqa/build-ast322/V3_rebuild/build-system-3/toolchain/nds32le-elf-mculib-v3s --with-pkgversion=2020-11-11_nds32le-e
lf-mculib-v3s-d1e3d7ae2a6 --disable-nls --enable-languages=c,c++ --enable-lto --with-gmp=/NOBACKUP/sqa15/sqa/build-ast322/V3_rebuild/build-system
-3/host-tools --with-mpfr=/NOBACKUP/sqa15/sqa/build-ast322/V3_rebuild/build-system-3/host-tools --with-mpc=/NOBACKUP/sqa15/sqa/build-ast322/V3_re
build/build-system-3/host-tools --with-isl=/NOBACKUP/sqa15/sqa/build-ast322/V3_rebuild/build-system-3/host-tools --with-cloog=/NOBACKUP/sqa15/sqa
/build-ast322/V3_rebuild/build-system-3/host-tools --without-zstd --with-arch=v3s --with-cpu=n10 --enable-default-relax=yes --enable-0s-default-i
fc=yes --enable-0s-default-ex9=yes --with-nds32-lib=mculib --disable-werror --enable-multilib=yes --with-multilib-list=dsp,zol,graywolf --with-ne
wlib --disable-shared --enable-threads=single --with-headers=/NOBACKUP/sqa15/sqa/build-ast322/V3_rebuild/build-system-3/toolchain/nds32le-elf-mcu
lib-v3s/nds32le-elf/include CFLAGS='-O2 -g -Wno-implicit-fallthrough -Wno-int-in-bool-context -Wno-cast-function-type' CXXFLAGS='-O2 -g -Wno-impl
icit-fallthrough -Wno-int-in-bool-context -Wno-cast-function-type' CC=/NOBACKUP/sqa15/sqa/build-ast322/V3_rebuild/build-system-3/host-tools/bin/g
cc CXX=/NOBACKUP/sqa15/sqa/build-ast322/V3_rebuild/build-system-3/host-tools/bin/g++ LD=/NOBACKUP/sqa15/sqa/build-ast322/V3_rebuild/build-system-
3/host-tools/bin/ld AR=/NOBACKUP/sqa15/sqa/build-ast322/V3_rebuild/build-system-3/host-tools/bin/gcc-ar RANLIB=/NOBACKUP/sqa15/sqa/build-ast322/V
3_rebuild/build-system-3/host-tools/bin/gcc-ranlib NM=/NOBACKUP/sqa15/sqa/build-ast322/V3_rebuild/build-system-3/host-tools/bin/nm --enable-check
ing=release LDFLAGS=--static 'CFLAGS_FOR_TARGET=-O2 -g  -mforce-no-ext-zol -mforce-no-ext-dsp' 'CXXFLAGS_FOR_TARGET=-O2 -g  -mforce-no-ext-zol -m
force-no-ext-dsp' LDFLAGS_FOR_TARGET=
Thread model: single
gcc version 4.9.4 (2020-11-11_nds32le-elf-mculib-v3s-d1e3d7ae2a6)
andy.liang@ubiserver:andy$
```

图 3-3-2

- 安装编译器兼容库

Note 1: For the use of Linux toolchains, please install zlib1g:i386 along with BSP v4.2:

```
$ sudo apt-get install zlib1g:i386
```

Note 2: Please install additional packages as follows to ensure all components of BSP v4.2 work successfully on 64-bit version of Ubuntu:

```
$ sudo apt-get install lib32z1
$ sudo apt-get install lib32ncurses5
$ sudo apt-get install e2fslibs:i386
$ sudo apt-get install gcc-multilib
$ sudo apt-get install g++-multilib
```

3.3.2 固件下载工具

- 从南方硅谷提供的 FTP 下载 Download_Tool_UI
- 参考硬件准备

3.3.3 Linux 系统镜像下载

从南方硅谷提供的 FTP 中，下载 tiramisu_linux.part1 tiramisu_linux.part3 tiramisu_linux.part3 三个文件，解压后即可得到 Linux 系统镜像。其中 root 账户密码为 icommiot

4. Flash 布局

Flash 布局如图 4-1 与 4-2 所示

Flash with FileSystem FOTA

bootloader	64KB
FOTA data	4KB
RF config	4KB
MAC config	4KB
PAD config	4KB

Boot Info	4KB
RAW	SETTING_PARTITION_USER_RAW_SIZE Default 8Kb
Main Partition	SETTING_PARTITION_MAIN_SIZE
FileSystem Parition	Remain Size

图 4-1

Flash with A/B FOTA

bootloader	64KB
FOTA data	4KB
RF config	4KB
MAC config	4KB
PAD config	4KB
Boot Info	4KB
RAW	SETTING_PARTITION_USE R_RAW_SIZE Default 8Kb
XIP1 Main Partition	SETTING_PARTITION_MAI N_SIZE
XIP2 Main Partition	SETTING_PARTITION_MAI N_SIZE

FileSystem Parition	Remain Size

图 4-2

- BootLoader

  初始化硬件资源、主分区升级、主分区地址跳转等。

- FOTA data

  保存 OTA 升级所需要的信息

- RF config

  RF 参数区，用于校准 RF，提高 RF 性能

- MAC config

  存放 MAC 地址的分区

- PAD config

  GPIO 配置分区

- BOOT info

  ROM code boot 信息

- RAW

  用户数据区，可存自定义数据，如证书数据等

- XIP1

  代码主分区，程序主分区，BootLoader 运行完后，跳转到此段

- XIP2

  如果使用 AB 升级方式，则存在

- file system

  文件系统区，所 Flash 剩余的空间做为文件系统

串口升级或网络 OTA 升级，只更新程序主分区。USB 烧录则更新整片 Flash。

5. 编译 SDK

5.1 编译准备

5.1.1 可编译的范例

如图 5-1-1-1 所示，projects 目录下包含若干可编译的范例

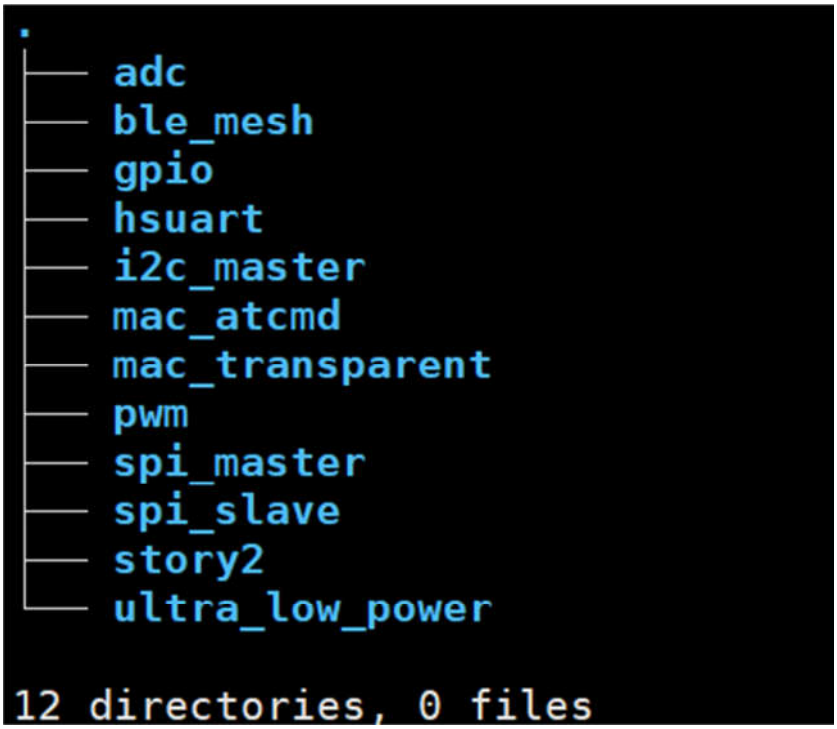


图 5-1-1-1

其中：

- mac_atcmd 通用开发固件，包含所有测试 AT 命令，可用于功能演示与开发。IOT 应用开发优先使用此 project 做为 demo。
- ultra_low_power 超低功耗专案，通过配置 feature.mk 中 SRAM--256K、MCU Clk--160MHz、

flash Clk--40MHz 以达到最低省电模式。功耗测试一般使用此专案

- ble_mesh 功能演示
- mac_transparent IOT 透传专案，IOT 透传开发使用此 demo。
- 其他

驱动测试范例，每个范例中的 src/app/ custom_cmd.c 文件都有说明如何测试驱动

5.1.2 芯片/晶体选择

每个范例中，都存在 mk/board.mk 文件，用于配置芯片型号/晶体参数，如图 5-1-2-1 所示

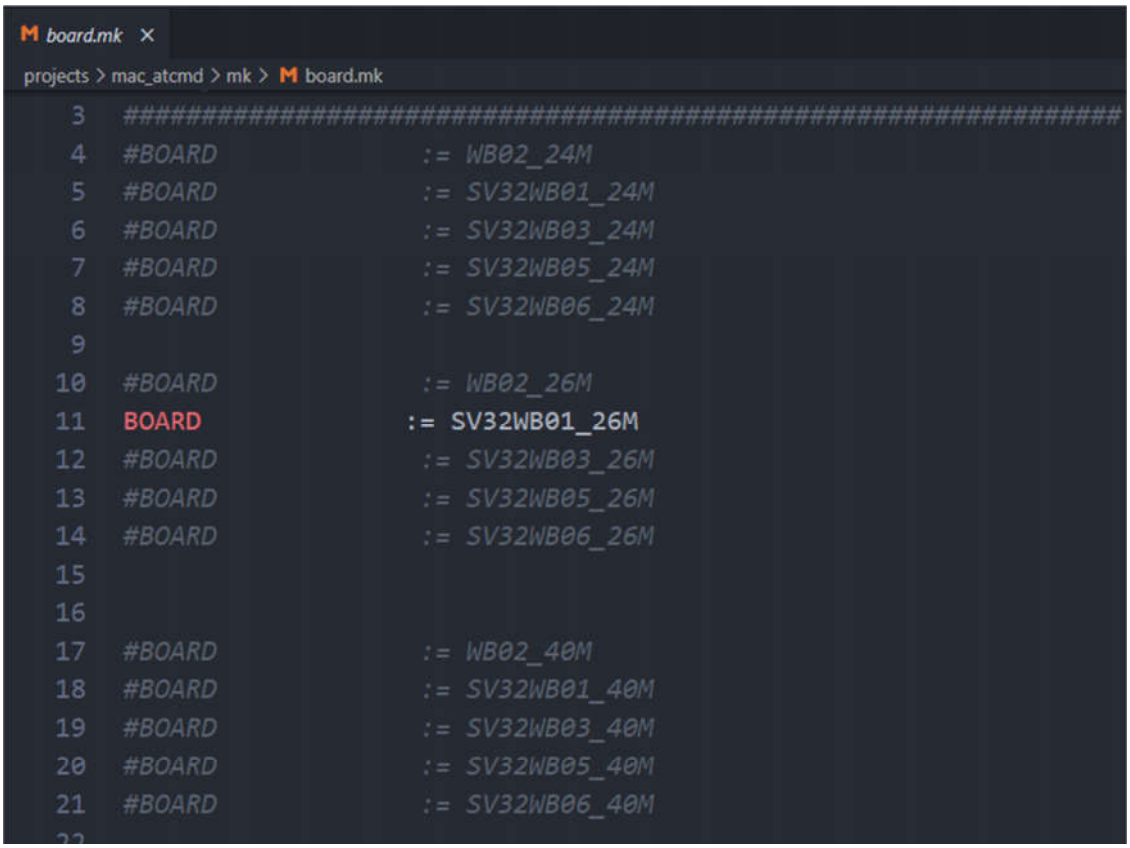


图 5-1-2-1

5.1.3 GPIO 配置

每个范例的中，src/board/inc/custom/ssv6020C/ 目录中包含了 P32，P40，P60 三种封装的芯片 GPIO 配置。通过配置其 custom_io_hal.h 实现所需的配置功能。例如 SV32WB01(L)芯片

则需要配置 P32 中的 custom_io_hal.h

```
//ALT0 : AICE_TMSC  /ALT1 : ADC0          /ALT2 : ANTSW_BT_SW_li      /ALT3 : UART0_RXD       /ALT4 : NONE
#define M_CUSTOM_P00_MODE       M_CUSTOM_ALT3

//ALT0 : AICE_TCKC  /ALT1 : ADC1          /ALT2 : ANTSW_WIFI_TX_SW_li /ALT3 : UART0_TXD       /ALT4 : NONE
#define M_CUSTOM_P01_MODE       M_CUSTOM_ALT3

//ALT0 : SIO13      /ALT1 : NONE          /ALT2 : NONE                /ALT3 : NONE            /ALT4 : NONE
#define M_CUSTOM_P13_MODE       M_CUSTOM_ALT0

//ALT0 : GPIO14     /ALT1 : NONE          /ALT2 : NONE                /ALT3 : PDMTX0_DOUT0    /ALT4 : NONE
#define M_CUSTOM_P14_MODE       M_CUSTOM_ALT0

//ALT0 : GPIO17     /ALT1 : SD_DATA2      /ALT2 : UART2_NCTS          /ALT3 : NONE            /ALT4 : NONE
#define M_CUSTOM_P17_MODE       M_CUSTOM_ALT0

//ALT0 : GPIO18     /ALT1 : SD_DATA3      /ALT2 : NONE                /ALT3 : NONE            /ALT4 : SPISLV1_
#define M_CUSTOM_P18_MODE       M_CUSTOM_ALT0

//ALT0 : GPIO19     /ALT1 : SD_CMD        /ALT2 : NONE                /ALT3 : NONE            /ALT4 : SPISLV1_
#define M_CUSTOM_P19_MODE       M_CUSTOM_ALT0

//ALT0 : GPIO20     /ALT1 : SD_CLK        /ALT2 : NONE                /ALT3 : NONE            /ALT4 : SPISLV1_
#define M_CUSTOM_P20_MODE       M_CUSTOM_ALT0

//ALT0 : GPIO21     /ALT1 : SD_DATA0      /ALT2 : NONE                /ALT3 : NONE            /ALT4 : SPISLV1_
#define M_CUSTOM_P21_MODE       M_CUSTOM_ALT0
```

5.2 开始编译

- 确定编译器安装正确，输入 nds32le-elf-gcc –v 查看编译器版本为：gcc version 4.9.4 (2020-11-11_nds32le-elf-mculib-v3s-d1e3d7ae2a6)
- 输入：make clean 清理 SDK 旧内容
- 输入：make setup p=project_name 选择要编译工范例，如 make setup p=mac_atcmd 则编译 mac_atcmd 此范例
- 输入：make 进行编译
- 编译完成后，image 目录存在输出文件

5.3 添加用户文件

SDK 中的文件组织都以模块化的形式存在，每个模块用 module.mk 来管理（类似于 Android SDK 的 Android.mk 文件）。当需要添加一个 C 文件时，只需修改 module.mk 文件即可。例如在 projects/mac_atcmd/src/app/目录下增加 hello.c 文件，修改

projects/mac_atcmd/src/app/module.mk 文件如下：

```
LIB_SRC := main.c
LIB_SRC += custom_cmd.c
LIB_SRC += hello.c

LIB_ASRC :=
LIBRARY_NAME := application
LOCAL_CFLAGS :=
LOCAL_AFLAGS :=

LOCAL_INC := -I$(TOPDIR)/components/inc/bsp
LOCAL_INC += -I$(TOPDIR)/components/drv
LOCAL_INC += -I$(TOPDIR)/components/softmac
LOCAL_INC += -I$(TOPDIR)/components/iotapi
```

图 5-3-1

5.4 添加用户目录

用户新建的目录，里面必须有 module.mk 文件，可参考

components/third_party/airkiss/module.mk 文件，如图 5-4-1 所示

```
LIB_SRC :=
LIB_SRC += airkiss_main.c

ifeq ($(strip $(AIRKISSAES_EN)), 1)
STATIC_LIB += $(MYDIR)/libairkiss_aes.a $(MYDIR)/libairkiss_aes_log.a
else
STATIC_LIB += $(MYDIR)/libairkiss.a $(MYDIR)/libairkiss_log.a
endif

LIB_ASRC :=
LIBRARY_NAME := airkiss
LOCAL_CFLAGS += -Wno-address
ifeq ($(strip $(AIRKISSAES_EN)), 1)
LOCAL_CFLAGS += -DAIRKISS_AES
endif
LOCAL_AFLAGS +=
LOCAL_INC := -I$(TOPDIR)/components/wifi_pkg_cap
LOCAL_INC += -I$(TOPDIR)/components/third_party/airkiss
LOCAL_INC += -I$(TOPDIR)/components/drv
LOCAL_INC += -I$(TOPDIR)/components/softmac
LOCAL_INC += -I$(TOPDIR)/components/iotapi

RELEASE_SRC := 2

$(eval $(call build-lib,$(LIBRARY_NAME),$(LIB_SRC),$(LIB_ASRC),$(LOCAL_CFLAGS),$(LOCAL_INC),$(LOCAL_AFLAGS),$(MYDIR)))
```

图 5-4-1

- LIB_SRC:指定需要编译的文件
- STATIC_LIB:指定此模块需要链接的库文件

- LIBRARY_NAME:指定此模块的名字
- LOCAL_INC:指定头文件的路径

最后一行直接复制使用即可

最后，在需要编译的工程中，指定该目录的路径即可（即该 module.mk 的路径）。如 projects/mac_atcmd/mk/libraries.mk 文件中，指定 airkiss 的编译如下：

```
# Library link
#########################################################################
STATIC_LIB  +=
IMPORT_DIR  += $(PROJ_DIR)/src/app
IMPORT_DIR  += components/third_party/airkiss

#########################################################################
# object link
#########################################################################
PROJECT_SRC +=
PROJECT_OBJ +=
```

图 5-4-2

5.5 添加用户头文件

头文件可分为局部头文件与全局头文件

局部头文件如图 5-5-1 所示，模块的 LOCAL_INC 指定；全局头文件在 build_opts.mk 中指定，如图所示

```
INCLUDE += -I$(SDKDIR)/components/third_party/matrixssl
INCLUDE += -I$(SDKDIR)/components/third_party/crc16
INCLUDE += -I$(SDKDIR)/components/inc/crypto
INCLUDE += -I$(SDKDIR)/components/softmac
INCLUDE += -I$(SDKDIR)/components/iotapi
INCLUDE += -I$(SDKDIR)/components/netstack_wrapper
```

图 5-5-1

5.6.选择地区码

由于每个地区使用的 WiFi 信道有差异，所以需要根据实际情况设定地区码（开机后呼叫

set_country_code API）。如果在中国地区使用，但设定了台湾地区，则无法连接 2.4G 的 12 与 13 信道。

目前支持的地区码有：TW CN HK US JP 五个地区

可修改 feature.mk 以支持默认地区码

```
DEFAULT_RATE_MASK               := 0x2B1
COUNTRY_CODE            := 0
# DEF=0, TW=1, CN=2, HK=3, US=4, JP=5 Please check the COUNTRY_CODE in wificonf.h
#
#################################################################
```

5.7. 添加数学库 libm.a 支持

当开发需要使用数学库时，可以使用编译器自带的 libm.a 静态库，在 build/compiler.mk 中，LDFLAGS 追加-lm，即可把数学库连接进来使用。在需要用到的.c 中 include <math.h>即可

build > compiler.mk

```
77  DEFAULT_CFLAGS          += -D${TARGET_DEF}
78  DEFAULT_AFLAGS          += $(GLOBAL_DEF) $(DBGFLAGS) $(BUILD_FLAGS)
79  DEFAULT_AFLAGS          += -D${TARGET_DEF}
80
81  LDFLAGS         += -Wl,--gc-sections
82  LDFLAGS         += -nostartfiles -Xlinker -M
83  LDFLAGS         += --specs=nosys.specs -Werror -mcmodel=large -g -lm
84
```

信道	频宽(MHz)	中心频率(MHz)	中国	中国台湾	美国	欧洲	日本	韩国	新加坡	加拿大	以色列	澳大利亚	其他大部分国家
1	20	2412	是	是	是	是	是	是	是	是	否	是	是
2	20	2417	是	是	是	是	是	是	是	是	否	是	是
3	20	2422	是	是	是	是	是	是	是	是	是	是	是
4	20	2427	是	是	是	是	是	是	是	是	是	是	是
5	20	2432	是	是	是	是	是	是	是	是	是	是	是
6	20	2437	是	是	是	是	是	是	是	是	是	是	是
7	20	2442	是	是	是	是	是	是	是	是	是	是	是
8	20	2447	是	是	是	是	是	是	是	是	是	是	是
9	20	2452	是	是	是	是	是	是	是	是	是	是	是
10	20	2457	是	是	是	是	是	是	是	是	否	是	是
11	20	2462	是	是	是	是	是	是	是	是	否	是	是
12	20	2467	是	否	否	是	是	是	是	否	否	是	是
13	20	2472	是	否	否	是	是	是	是	否	否	是	是
14	20	2484	否	否	否	否	802.11 b only	否	否	否	否	否	否

世界各个地区 WiFi 2.4G 信道一览表

注：表格仅供参考，以实际信道表为准。

6. SDK 进阶介绍

6.1. feature.mk 配置解析

```
7  ##################################################################
8  # Debugger option
9  ##################################################################
10 #BUILD_OPTION                  := PERFORMANCE
11 #BUILD_OPTION                  := DEBUG
12 BUILD_OPTION                   := RELEASE
13 SUPPORT_EXCEPTION_DUMP        := 1
14 SUPPORT_SRM_TASK_LOG          := 0
15 SUPPORT_OSAL_SYS_INFO         := 1
16 SUPPORT_SRM_DBG_INFO          := 1
17
```

debug 选项

- BUILD_OPTION：当设定为 DEBUG 模式时，自动打开 Freertos StackHighWaterMark 功能，用于检查所有运行中的 task stack 使用情况。

```
?>sysinfo
[System information]
mcu clk 320000000
info ICache Enable, DCache Enable
NTC0 WB NTC1 WB NTC2 NO NTC3 NO
xtal clk 26000000, bus clk 160000000, xip mode 4
[Task information]
Name             Priority Stack_Size(word) Free CPU_Usage
 isr                    H     X/  512        X 0001/1000
*cli                    1   130/ 1024      894 0000/1000
 IDLE                   0    94/  128       34 0998/1000
 Tmr Svc                7    55/  512      457 0000/1000
 Radio_Receive_T        3    84/ 1344     1260 0001/1000
 Radio_Tx_Task          3    70/  768      698 0000/1000
 tcpip_task             3    80/ 2048     1968 0000/1000
sysinfo=OK
```

- SUPPORT_OSAL_SYS_INFO　　支持 sysinfo 查看当前 task 的状态
- SUPPORT_SRM_DBG_INFO　　支持 meminfo 查看当前 memory 的状态
- SUPPORT_SRM_MEM_INFO　　支持 meminfo 命令中打印 memory 使用情况，一般用于内存泄露分析

注意：打开 debug 功能，占用更多的 SRAM 与 CPU 资源。

```
?>meminfo
total SRAM: 384K
============================================
            used
      dynamic+static/total   free  max_free
ILM     27856+ 96696/131072   6520     2680
Bus         0+ 98492/262144 163652   163616
Total     223044    /393216 170172

DLM used list:


ILM used list:
0x000179B8 size 112, caller 00001008:
0x00017A28 size 288, caller 3002EAF4:
0x00017B48 size 288, caller 3002EAF4:
0x00017C68 size 48, caller 3002EAF4:
0x00017C98 size 72, caller 3002FBE8:
0x00017CE0 size 112, caller 3002FC88:
0x00017D50 size 632, caller 3002FCCE:
0x00017FC8 size 1032, caller 3002FD34:
0x000183D0 size 72, caller 000052D4:
```

SUPPORT_SRM_MEM_INFO

```
18  ##################################################################
19  # Partition Setting, influence FOTA
20  ##################################################################
21  SETTING_PARTITION_MAIN_SIZE      := 620K
22  SETTING_FLASH_TOTAL_SIZE         := defined_by_chip
23  SETTING_PARTITION_USER_RAW_SIZE := 8K
24  SETTING_PSRAM_HEAP_SIZE          := 0
```

Flash 配置

- SETTING_PARTITION_MAIN_SIZE　　主分区最大大小
- SETTING_FLASH_TOTAL_SIZE　　Flash 的总大小
- SETTING_PARTITION_USER_RAW_SIZE　　User 用户分区的大小
- SETTING_PSRAM_HEAP_SIZE　　如果外挂 PSRAM，设定 psram 的大小

```
26  ##################################################################
27  # System Setting
28  ##################################################################
29  SUPPORT_LOW_POWER                := 1
30
31  #SYS_BUS_CLK                      := 40M
32  #SYS_BUS_CLK                      := 80M
33  SYS_BUS_CLK                      := 160M
34
35  #SYS_MCU_MAX_CLK                  := 80M
36  #SYS_MCU_MAX_CLK                  := 160M
37  #SYS_MCU_MAX_CLK                  := 240M
38  #SYS_MCU_MAX_CLK                  := 320M
39  #SYS_MCU_MAX_CLK                  := 480M
40  SYS_MCU_MAX_CLK                  := defined_by_chip
41
42  #SYS_FLASH_CLK                    := 40M
43  SYS_FLASH_CLK                    := 80M
44
45  #XIP_BIT                         := 2
46  XIP_BIT                          := 4
47
48  # 0 ==> disable fota, 1==> filesystem fota, 2 ==> ping pong fota.
49  FOTA_OPTION                      := 1
50
51  # SETTING_SRAM_OPTION = 1 ==> SRAM: MAX_SRAM
52  # SETTING_SRAM_OPTION = 0 ==> SRAM: 256KB
53  SETTING_SRAM_OPTION              := 1
54
55  SUPPORT_MCU_UNALIGNMENT          := 0
```

system 设定

- SUPPORT_LOW_POWER　　是否支持低功耗功能，低功耗功能会占用 sram
- SYS_BUS_CLK　　BUS 时钟选择
- SYS_MCU_MAX_CLK　　CPU 主频选择
- SYS_FLASH_CLK　　Flash 读写 clk 主频选择
- XIP_BIT　　FLASH 读写模式，4 为四线模式
- FOTA_OPTION　　FOTA 模式选择，1 为文件系统方式，2 为 A/B 区方式
- SETTING_SRAM_OPTION　　选择 sram 的大小
- SUPPORT_MCU_UNALIGNMENT　　设定 MCU 是否支持非对齐操作

```
#############################################################
# WIFI feature option
#############################################################
SETTING_THROUGHPUT_HIGH         := 1
TCPIPSTACK_EN                   := 1
LWIP_PATH                       := lwip-1.4.0
#LWIP_PATH                      := lwip-2.0.3
LWIP_RESOURCES_PARAMETER        := 1
# 0:PBUF_SIZE=12, 1:PBUF_SIZE=18, 2:PBUF_SIZE=24, 3:Customer modify in lwipopts.h, 4:Dynamic allocate memory
RC_MINSTREL_EN                  := 0
RC_NATIVE_EN                    := 1
EAP_EN                          := 0
WIFI_REPEATER_EN                := 0
SOFTAP_EN                       := 1
SMARTCONFIG_EN                  := 1
DEFAULT_RATE_MASK               := 0x2B1
COUNTRY_CODE                    := 0
# DEF=0, TW=1, CN=2, HK=3, US=4, JP=5 Please check the COUNTRY_CODE in wificonf.h
```

WiFi 设定

- SETTING_THROUGHPUT_HIGH　　高性能，占用更多的 SRAM 空间
- COUNTRY_CODE　　设定默认地区码

```
111 ###########################################################################
112 # Filesystem feature
113 ###########################################################################
114 SUPPORT_FFS                        := 2
115 # SD Card
116 SUPPORT_SDC                        := 0
117
118 ###########################################################################
119 # Misc setting
120 ###########################################################################
121 UART_IO_NUM                        := defined_by_chip
122 # 0 ==> Debug UART, 1 ==> HSUART1, 2==> HSUART2
123 UART_BAUD_RATE                     := 115200
124 BOOTLOADER_ENABLE_QE               := 0
125 SUPPORT_CXX                        := 0
126 ISR_STACK_SIZE                     := 2048
```

其他设定

- SUPPORT_FFS　　支持的文件系统，0 为无文件系统
- SUPPORT_SDC　　支持 SPI 读写 SD 卡（FAT32）功能
- UART_IO_NUM　　设定 debug log 输出口
- UART_BAUD_RATE　　设定 debug log 的波特率
- BOOTLOADER_ENABLE_QE　　开启 bootloader 使能 Flash QE 模式功能

6.2. UART 的配置

SV32WB0x 支持三路 UART，包含一路普通 UART，两路高速 UART

名称	GPIO	波特率范围	是否带流控功能	作用
UART0	GPIO00/GPIO01	1-921600	否	烧录，查看 LOG
HSUART1	GPIO29/GPIO33	1-4800000	否	通信
HSUART2	GPIO36/GPIO37	1-4800000	是	通信

feature.mk 配置文件中，UART_IO_NUM 的配置可以使 LOG 从指定的 UART 口输出，但不能修改烧录口的位置

HSUART 编程需要的头文件为 drv_hsuart.h，包含以下常见函数

- drv_hsuart_init_ex 初始化指定 HSUART 模块
- drv_hsuart_set_format_ex 设定指定 HSUART 模块的参数
- drv_hsuart_set_fifo_ex 设定指定 HSUART 模块的 FIFO 深度
- drv_hsuart_register_isr_ex 注册中断函数
- drv_hsuart_set_hardware_flow_control_ex 设定流控参数，如果无流控或不需要流控，请勿设定
- drv_hsuart_write_fifo_ex 指定 HSUART 发送数据
- drv_hsuart_read_fifo_ex 指定 HSUART 接收数据
- drv_hsuart_deinit_ex 去初始化指定 HSUART 模块

6.3. Wi-Fi API 使用介绍

所有 Wi-Fi API，均在 components/iotapi/wifi_api.h 申明，分为以下几类：

6.3.1. WiFi 工作模式设定

- DUT_wifi_OFF：关闭所有 WiFi 功能，包括 RF/PHY 电路部份，一般在休眠时自动呼叫
- get_DUT_wifi_mode：获取当前工作模式
- DUT_wifi_start：设定当前工作模式，可以为 STA、AP、STA/AP 共存、sinffer 模式
- DUT_wifi_stop：停止当前 WiFi 所有工作

6.3.2. 扫描函数

- scan_AP_xxx 根据需求，选择不同参数的扫描函数，当 ssid 参数不为 NULL 时，可以扫描指定的 AP 名称，包含隐藏与非隐藏 AP

6.3.3. 主动连接函数

- wifi_connect_active_xxx 根据需求，连接指定 AP，可以是隐藏 AP

6.3.4. 被动连接配置参数

- set_wifi_config_xxxx 当呼叫 scan_AP_xxx 后，并且已经扫描到需要连接的 AP，可以呼叫此类函数进行设定。

6.3.5. 被动连接函数

- wifi_connect_xxx 当已经呼叫 set_wifi_config_xxxx 设定参数后，呼叫此类函数进行连接路由器

6.3.6. 断线函数

- wifi_disconnect 断开已经连接的 AP

6.3.7. 获取 wifi 连接情况函数

- get_wifi_status 获取当前是否连接

6.3.8. 设定参数

- set_local_mac 获取当前 MAC 地址
- get_local_mac 设定临时 MAC 地址
- set_country_code 设定临时地区码

6.3.9. softap

- wifi_ap_easy_conf 配置 softap 的参数，配置完成后使用 DUT_wifi_start 进入 softap 模式
- get_connectsta_info 获取当前连接到 softap 的设备信息
- wifi_softap_disconnect_sta 断开指定的设备

6.4. Wi-Fi 配网介绍

6.4.1. 开启/关闭 sinffer 模式

- start_sniffer_mode 开启 sinffer 模式，在 sinffer 配置完成后呼叫
- stop_sniffer_mode 退出 sinffer 模式

6.4.2. sinffer 配置模式

- attach_sniffer_cb 配置 sinffer 参数，其中包含过滤条件，回调函数，回调函数栈大小
- deattach_sniffer_cb 去配置

6.4.3. sinffer 信道

- set_channel 设定当前信道
- get_current_channel 获取当前信道

6.5. 添加自定义国家码

添加自定义国家码，如哥伦比亚（简称 CO）流程如下：

1、 下载最新国家码资料库：git clone git://git.kernel.org/pub/scm/linux/kernel/git/linville/wireless-regdb.git

2、 打开 db.txt 文件，搜索 CO 关键词

```
257 country CO: DFS-FCC
258     (2402 - 2482 @ 40), (20)
259     (5170 - 5250 @ 80), (17), AUTO-BW
260     (5250 - 5330 @ 80), (24), DFS, AUTO-BW
261     (5490 - 5730 @ 160), (24), DFS
262     (5735 - 5835 @ 80), (30)
```

3、 打开 SDK 中 countryinfo.h 配置文件，参考已有的代码与上述的信息填写如下

```
                {0, 0, 0, 0, 0, 0}}
        },
        {JP, {{2402, 2494, 40, 0, 20, 0},
              {5030, 5090, 40, 0, 23, 0},
              {5170, 5250, 80, 0, 20, 0},
              {5250, 5330, 80, 0, 20, NL80211_RRF_DFS},
              {5490, 5710, 160, 0, 23, NL80211_RRF_DFS},
              {0, 0, 0, 0, 0, 0}}
        },
        {CO, {{2402, 2482, 40, 0, 20, 0},
              {5170, 5250, 80, 0, 17, 0},
              {5250, 5330, 80, 0, 24, NL80211_RRF_DFS},
              {5490, 5730, 160, 0, 24, NL80211_RRF_DFS},
              {5735, 5835, 80, 30, 0, 0},
              {0, 0, 0, 0, 0, 0}}
        },
    };
```

Flags: 目前只处理 DFS

max_antenna_gain: 不参考

max_eirp: 不参考

4、 打开 wificonf.h，枚举变量 COUNTRY_CODE 添加 CO 国家码

```
416  typedef enum {
417      DEF=0,
418      TW,
419      CN,
420      HK,
421      US,
422      JP,
423      CO,
424      COUNTRY_MAX,
425  }COUNTRY_CODE;
```

5、在开发过程中，即可使用 set_country_code(CO);设定哥伦比亚国家码

6.6. CLI 指令集开发

SV32WB0x SDK 支持客户自定义 AT 命令，当 feature.mk 中设定了 SUPPORT_CUSTOM_CMD，则工程下面的 custom_cmd.c 会被编译进固件。客户可以基于 custom_cmd.c 添加自定义的 AT 命令。

- 在 gCustomCmdTable 中添加命令

```
61  /* ---------------------- Registered CMDs to CMD Table ---------------------- */
62  const CLICmds gCustomCmdTable[] =
63  {
64      { "meminfo",            Cmd_meminfo,        "meminfo"},
65      { "sysinfo",            Cmd_sysinfo,        "sysinfo"},
66
67      { "cmd_log",            Cmd_log,        "test command"},
68      { (const char *)NULL, (CliCmdFunc)NULL,    (const char *)NULL },
69  };
```

- 实现添加的命令

```
28  int Cmd_log(s32 argc, char *argv[])
29  {
30      int i;
31      char* tag_array[5] = {"app", "driver", "app_iperf", "lwip", "app_main"};
32      for(i=0; i<5; i++)
33      {
34          printf("\n");
35          log_e( tag_array[i], "(%s)this is ERROR message\n", tag_array[i]);
36          log_w( tag_array[i], "(%s)this is WARN message\n", tag_array[i]);
37      }
38      return ERROR_SUCCESS;
39  }
```

6.7. 低功耗编程

6.7.1. 省电模式介绍

SV32WB0X SDK 提供五种省电模式 Power mode(0~4)与一种休眠模式(Dormant)

	MCU	WiFi
Power Mode 0	ON	ON
Power Mode 1	Standby	ON
Power Mode 2	ON	DTIM
Power Mode 3	Standby	DTIM
Power Mode 4	Sleep	DTIM
Dormant	OFF	OFF

说明：

- MCU ON：CPU 一直全速运行
- MCU Standby：当 CPU 空间时进入 clock gating 进行省电
- MCU Sleep：CPU 空间时进入进入 clock gating 进行省电，并且关闭外设 Clock（即此时 HSUART/UART/PWM 等无法正常工作）
- WiFi ON：RX 功能一直打开。
- WiFi ON：DTIM 功能省电，间隔打开 RX
- Dormant 只能定时器或 GPIO 唤醒系统，WiFi 需要重新连接。而 Power Mode 则保持 WiFi 连接.

6.7.2. 影响功耗的因素

- 硬件因素
  - DCDC/LDO 设置，DCDC 的使用比 LDO 省电
  - TX 功率设定
- 软件因素
  - Sram 的配置，512K 的 Sram 比 256K Sram 多 37uA
  - IO 配置，GPIO 初始化为上拉，但休眠时把 GPIO 拉低，造成 IO 口漏电
  - MCU 主频，320MHz 主频相对 80MHz 主频多耗电 3.2mA
  - 总线主频，160MHz 比 40MHz 多耗电 4.3mA
- 环境因素
  - WiFi 封包的大小与数量
  - 环境干扰度（重传影响）

6.7.3. GPIO 配置与省电模式

- 芯片所有 Pin（除 Pin 00 01，预设是 AICE 功能, input 模式带上拉）默认都为 GPIO 模式，当

custom_io_hal.h 配置了 GPIO 模式，但驱动没有相关设定时，此时 GPIO 处于无输入无输出状态（GPIO 13 内部 20K？下拉，60 pin 封装芯片 GPIO 28 需要外部下拉 4.7K），类似于 high-z 模式。

- 设定 GPIO 的电平后，在休眠过程中，与休眠唤醒后电平都不会发生改变
- 当 GPIO 测量到 0.6V—2V 的电压时，说明 GPIO 在漏电（约 80uA），需要通过外部上下拉，或都软件设定高低电平，让电平处于 0V 或 3.3V，才能达到省电效果
- bsp.c 中，lowpower_dormant_gpio_hook 函数可以设定休眠期间的 GPIO 行为。例如 IOT 应用中，使用 M_GPIO_DEFAULT 默认值，与休眠前电平保持一致。而 IOT 透传应用中，配置 clk 默认是 pull down，其他 sdio 预设 pull up。

```
226 int lowpower_dormant_gpio_hook() {
227 #if defined(SUPPORT_HOST_IF) && (SUPPORT_HOST_IF == 0)
228     // do your gpio setting
229     //return M_GPIO_USER_DEFINED;
230     // use default gpio setting.
231     return M_GPIO_DEFAULT;
232 #else
233     // host low power.
234     hal_gpio_set_manual_pull_raw(SDIO_MASK, SDIO_PULLUP_MASK, SDIO_PULLDOWN_MASK, 0);
235     hal_gpio_set_pull_raw(SDIO_MASK, SDIO_PULLUP_MASK, SDIO_PULLDOWN_MASK, 0);
236     return M_GPIO_USER_DEFINED;
237 #endif
238 }
```

- 在休眠期间，除了 IOT 透传的 SDIO 或 SPI 接口，其他 peripheral IO（例如 UART/PWM/I2C）都会断电，对应的 IO 处于 GPIO 功能，由于断电前电平的状态未知，所以这里注意容易发生跟对端电位对拉，造成漏电。具体解决可以根据实际电路参考 lowpower_dormant_gpio_hook 函数配置电位。

6.7.4. API 说明

Power Mode 设定，由 lowpower_mode 控制 MCU 与 set_power_mode 控制 WiFi 的组合决定，参考 AT 指令：AT+POWERMODE。对应 5 种省电模式

```
743     switch(param->argv[0][0]) {
744         case '1':
745             lowpower_mode(E_LOW_POWER_STANDBY);
746             set_power_mode(0, DUT_STA);
747             break;
748         case '2':
749             lowpower_mode(E_LOW_POWER_ACTIVE);
750             set_power_mode(1, DUT_STA);
751             break;
752         case '3':
753             lowpower_mode(E_LOW_POWER_STANDBY);
754             set_power_mode(1, DUT_STA);
755             break;
756         case '4':
757             lowpower_mode(E_LOW_POWER_SLEEP);
758             set_power_mode(1, DUT_STA);
759             break;
760         case '0':
761         default:
762             lowpower_mode(E_LOW_POWER_ACTIVE);
763             set_power_mode(0, DUT_STA);
764             break;
765     }
766     return 0;
```

休眠模式，参考 AT 指令：AT+DORMANT。呼叫 sys_dormant 前，有两个前置条件

1、关闭 WiFi：DUT_wifi_start(DUT_NONE); DUT_wifi_OFF();

2、进入临界保护：OS_EnterCritical();

```
711 int At_dormant(stParam *param) ATTRIBUTE_SECTION_FAST;
712 int At_dormant(stParam *param) {
713     if (param->argc != 2) {
714         dormant_usage();
715         return 0;
716     }
717     struct timeval tv;
718     tv.tv_sec = strtol(param->argv[0], NULL, 10);
719     tv.tv_usec = strtol(param->argv[1], NULL, 10);
720     if (tv.tv_sec < 0) {
721         ATCMD_LOG_I("not support little time sleep for %dsec\n", tv.tv_sec);
722         return 0;
723     }
724     if (tv.tv_sec > 134217) {
725         ATCMD_LOG_I("not support super big time sleep for %dsec\n", tv.tv_sec);
726         return 0;
727     }
728     DUT_wifi_start(DUT_NONE);
729     DUT_wifi_OFF();
730     OS_EnterCritical();
731     //sys_rtc_cali();
732     sys_dormant(&tv);
733     OS_ExitCritical();
734     ATCMD_LOG_I("time=%d.%06d\n", tv.tv_sec, tv.tv_usec);
735     return 0;
736 }
```

设定 GPIO 唤醒功能

参考 AT 命令：AT+GPIO_WAKEUP

1、设定 IO 口为 GPIO 模式，并且为输入

2、呼叫 hal_gpio_set_wakeup_enable_with_level(gpio_id, pol)设定唤醒的条件（高低电平）

6.8. 安全启动

6.8.1. 安全启动（Secure Boot）概念

- Secure Boot 采用 ECDSA(Elliptic Curve Digital Signature Algorithm)作为算法。
- 在 Boot-loader 的编译时期，使用 ECDSA 的私钥对其做签名；Secure Boot 开机过程中，使用对应的公钥验证 Boot-loader 的签名：
  - 使用 openssl 工具，在编译时期产生 key pair。
  - 公钥储存在写入操作不可逆的 e-FUSE 中，且公钥不需要隐藏。
  - 私钥只储存在客户端，不储存在芯片开机软硬件系统中。
- 因为公钥被储存在 e-FUSE 中，客户能根据不同的设备类型或特定用例使用不同的公钥。
- 保证 Boot-loader 的完整性，不保证其机密性。

6. 8. 2. 开启安全启动

启动 Secure Boot 功能，必须要在编译时期开启 SECURE_BOOT

projects/mac_atcmd/mk/feature.mk 使能 SECURE_BOOT

```
#################################################################
# Misc setting
#################################################################
UART_IO_NUM                          := 0
# 0 ==> Debug UART, 1==> Data UART
UART_BAUD_RATE                     := 115200
SETTING_UART_UPGRADE_BOOTLOADER := 0
SETTING_UART_UPGRADE_EN          := 1
BOOTLOADER_ENABLE_QE             := 0
SECURE_BOOT                        := 1
```

SDK 预设的 key pair(包含预设公钥和默认私钥)的位置在：跟目录/security/ec_param.pem

关闭 Secure Boot or 换 key 时需要重新编译

6.8.3. 如何换 key

- 产生私钥：

```
# openssl ecparam -genkey -name prime256v1 -out
ec_param.pem -param_enc explicit
```

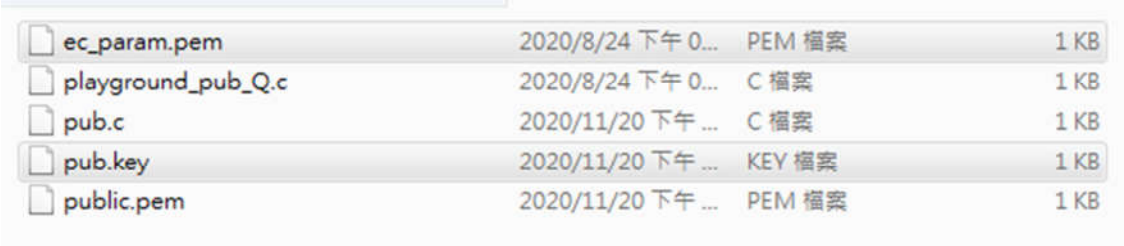
产品密钥须妥善保存，不可外泄！

- 在编译 image 前，更换的私钥的方式：

移除预设的 key pair 档案。

```
# rm ./security/ec_param.pem
```

新私钥 ec_param.pem 放置 security 后编译



- 私钥对一个已经编译好的 image 重新签名：

使用 SDK 中的签名工具 utils/sign_replace_tool.sh

私钥对一个已经编译好的 image 重新签名(无论此 image 在当初编译时是否有启动 Secure Boot

)：

使用 SDK 中的签名工具 utils/sign_replace_tool.sh

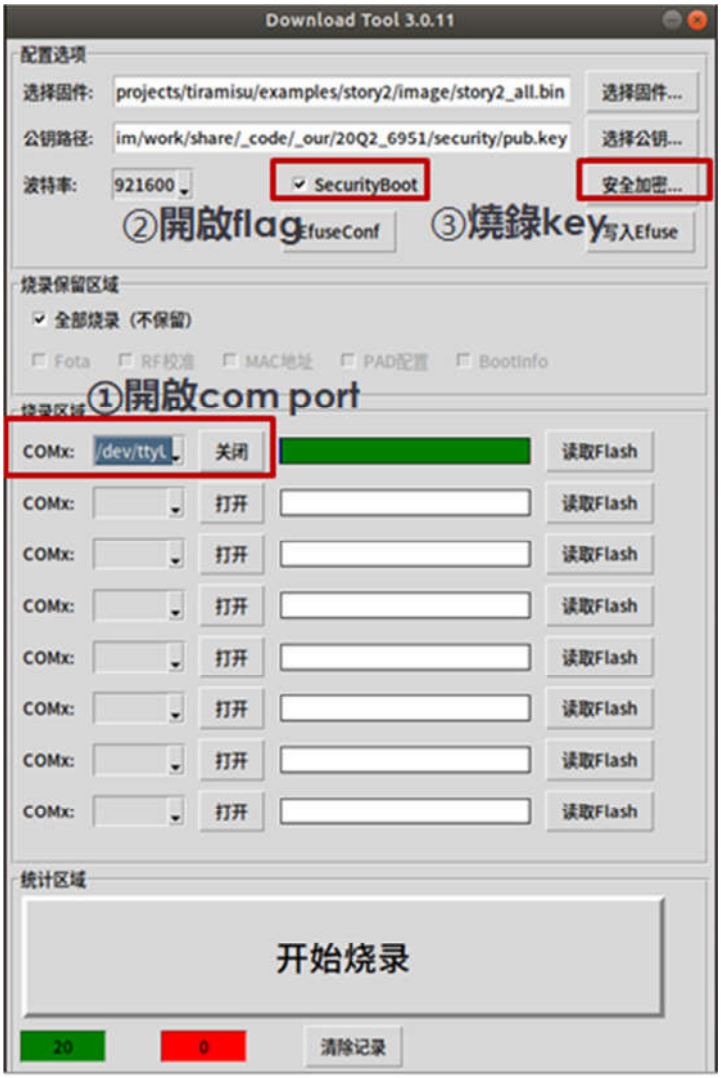
```
# ./utils/sign_replace_tool.sh ./image/secboot_all.bin ./image/secboot_all.new.bin ./security/new_ec_param.pem
```

第一个参数是已经编译好的 image(无论此 image 在当初编译时是否有启动 Secure Boot)。

第二个参数是输出档名，可以是不存在的档案。

第三个参数是指定的私钥

6.8.4. 烧录 KEY

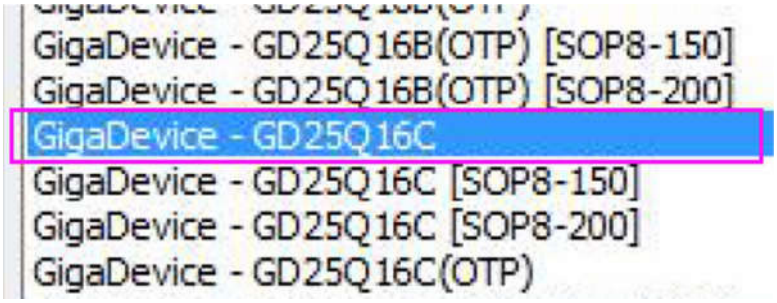


6.9. Bootloader 启动 Flash QE 模式

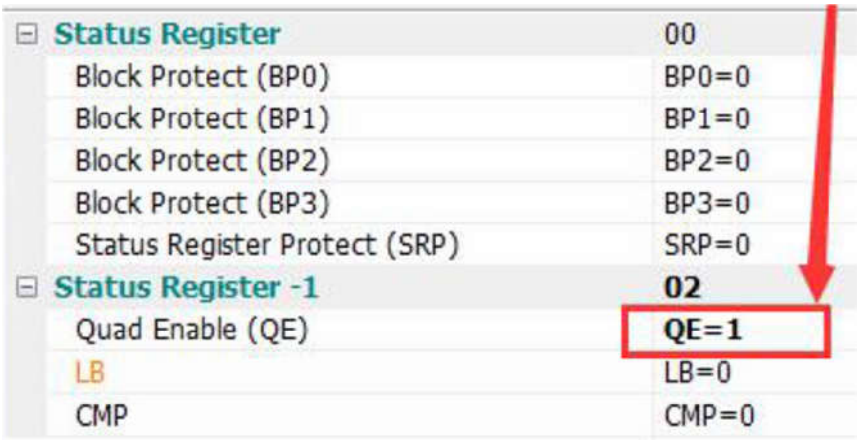
SV32WB0x SDK 默认配置使用 4 线模式（QE 模式）读写 Flash，当使用内置 Flash 芯片时，芯片内部的 rom code 会自动打开 QE 模式。当客户使用 SV32WB06(L)外挂其他品牌的 Flash 时，需要注意开启对应 Flash QE 模式的方式。一般有两种模式打开：

6.9.1. 第三方 Flash 烧录软件开启，例如 FlyPRO 软件

- 芯片选择：厂家 GigaDevice；型号：GD25Q16C；



- 配置选项：QE 选择等于 1；



6.9.2. Bootloader 开启方法

如果第三方烧录软件不支持开启 QE 模式，则可以使能 BOOTLOADER_ENABLE_QE 后，在 bootloader（custom_ota.c）中添加代码。目前 Flash 共有三类 QE type。根据 Flash 对应的 Datasheet 填入即可。

```
75
76      // step 2. switch qe bit method.
77      switch(flash_rx[0]) {
78          case MANUFACTURER_ID_GIGADEVICE:
79              if (flash_rx[2] == 0x17) { // GD25Q64C
80                  flash_set_qe_type2(flash_tx, flash_rx);
81              } else if (flash_rx[2] == 0x15) { //GD25Q16C
82                  flash_set_qe_type1(flash_tx, flash_rx);
83              } else { // GD others.
84                  flash_set_qe_type1(flash_tx, flash_rx);
85              }
86              break;
87          case MANUFACTURER_ID_WINBOND:
88              if (flash_rx[2] == 0x17) { // W25Q64JV
89                  flash_set_qe_type2(flash_tx, flash_rx);
90              } else if (flash_rx[2] == 0x15) { //W25Q16JV
91                  flash_set_qe_type1(flash_tx, flash_rx);
92              } else { // Winbond others.
93                  flash_set_qe_type1(flash_tx, flash_rx);
94              }
95              break;
96          case MANUFACTURER_ID_MXIC:
97              flash_set_qe_type0(flash_tx, flash_rx);
98              break;
99          case MANUFACTURER_ID_EON:
```

三种 type 可以总结如下：

	Read Status cmd	Qe bit	Write Status cmd
QE Type 0	0x05	bit 6	0x01
QE Type 1	0x05 and 0x35	bit 9	0x01
QE Type 2	0x35	bit 1	0x31

举例，GD25Q16C Flash Datasheet 如下，所以对应 QE type1

GigaDevice 3.3V Uniform Sector Dual and Quad Serial Flash GD25Q16C

6. STATUS REGISTER

S15	S14	S13	S12	S11	S10	S9	S8
SUS	CMP	HPM	Reserved	Reserved	LB	QE	SRP1

S7	S6	S5	S4	S3	S2	S1	S0
SRP0	BP4	BP3	BP2	BP1	BP0	WEL	WIP

The status and control bits of the Status Register are as follows:

其中 table of ID 表格如下：

Table of ID Definitions:

GD25Q16C

Operation Code	MID7-MID0	ID15-ID8	ID7-ID0
9FH	C8	40	15
90H	C8		14
ABH			14

对应此处代码

```
176         // step 2. switch qe bit method.
177         switch(flash_rx[0]) {
178             case MANUFACTURER_ID_GIGADEVICE:        C8
179                 if (flash_rx[2] == 0x17) { // GD25Q64C
180                     flash_set_qe_type2(flash_tx, flash_rx);
181                 } else if (flash_rx[2] == 0x15) { //GD25Q16C
182                     flash_set_qe_type1(flash_tx, flash_rx);
183                 } else { // GD others.
184                     flash_set_qe_type1(flash_tx, flash_rx);
185                 }
186                 break;
```

6.10.Padmux GPIO/IO 配置

padmux 配置 SV32WB0x 所有 GPIO 的功能。每个封装对应一个 custom_io_hal.h 文件，例如：

32pin 封装：projects/**<project>**/src/board/inc/custom/ssv6020C/P32/custom_io_hal.h

40pin 封装：projects/**<project>**/src/board/inc/custom/ssv6020C/P40/custom_io_hal.h

60pin 封装：projects/**<project>**/src/board/inc/custom/ssv6020C/P60/custom_io_hal.h

6.10.1. GPIO 统计

32pin 设定表如图所示

PIN	Boot Strapping ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7
0	AICE	ADC0	ANTSW	UART0(RX)				GPIO00
1	AICE	ADC1	ANTSW	UART0(TX)				GPIO01
13	SIO13							SIO13
14	GPIO14			PDMTX0				GPIO14
17	GPIO17	SDIO	UART2(CTS)					GPIO17
18	GPIO18	SDIO		DATASPIS	SPIS1_CSN	SPIM1_CSN		GPIO18
19	GPIO19	SDIO		DATASPIS	SPIS1	SPIM1		GPIO19
20	GPIO20	SDIO		DATASPIS	SPIS1	SPIM1		GPIO20
21	GPIO21	SDIO		DATASPIS	SPIS1	SPIM1		GPIO21
22	GPIO22	SDIO	UART2(RTS)					GPIO22
29	GPIO29	ADC3			UART1(RX)			GPIO29
33	GPIO33				UART1(TX)	BTCX		GPIO33
36	GPIO36	ADC6	I2C0	UART2(RX)		BTCX	ANTSW	GPIO36
37	GPIO37	ADC7	I2C0	UART2(TX)		BTCX	ANTSW	GPIO37

32pin 支持外设

名称	数量
UART	3
I2C	1
SPI Master	1
SPI Slave	1
PWM	8
ADC	5
SDIO	1
PWM Audio Tx	1(单声道 mono)

60pin 设定表如图所示

PIN	Boot Strapping ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7
0	AICE	ADC0	ANTSW	UART0	SPIS0	SPIM0		GPIO00
1	AICE	ADC1	ANTSW	UART0	SPIS0	SPIM0		GPIO01
2	GPIO02	ADC2			SPIS0	SPIM0		GPIO02
4	GPIO04	I2C0						GPIO04
5	GPIO05				SPIM0_CSN	SPIS0_CSN		GPIO05
6	GPIO06	I2C0	PSRAM_SPI					GPIO06
7		FLASH						
8		FLASH						
9		FLASH						
10		FLASH						
11		FLASH						
12		FLASH						
13	SIO13							SIO13
14	GPIO14			PDMTX0	PDMRX0	I2S0		GPIO14
15	GPIO15			PDMTX0	PDMRX0	I2S0		GPIO15
16	GPIO16				PDMRX1	I2S0		GPIO16
17	GPIO17	SDIO	UART2		PDMRX1	I2S0		GPIO17
18	GPIO18	SDIO		DATASPIS	SPIS1_CSN	SPIM1_CSN		GPIO18
19	GPIO19	SDIO		DATASPIS	SPIS1	SPIM1		GPIO19
20	GPIO20	SDIO		DATASPIS	SPIS1	SPIM1		GPIO20
21	GPIO21	SDIO		DATASPIS	SPIS1	SPIM1		GPIO21
22	GPIO22	SDIO	UART2		I2S0_MCLK			GPIO22
23	GPIO23		UART2	I2C1				GPIO23
24	GPIO24		UART2	I2C1				GPIO24
25	GPIO25	BTCX			UART2			GPIO25
26	GPIO26	BTCX		I2S0_MCLK	UART2	SPIM2_CSN		GPIO26
27	GPIO27		UART1	PDMRX0		I2S1		GPIO27
28	SIO28					I2S0_MCLK		SIO28
29	GPIO29	ADC3	UART1	PDMRX0	UART1	I2S1		GPIO29
30	GPIO30	ADC4	UART1	PDMRX1	PDMTX0	I2S1		GPIO30
31	GPIO31	ADC5	UART1	PDMRX1	PDMTX0	I2S1		GPIO31
32	GPIO32	BTCX		SPIM2				GPIO32
33	GPIO33			SPIM2	UART1	BTCX		GPIO33
35	GPIO35			SPIM2				GPIO35
36	GPIO36	ADC6	I2C0	UART2	SPIM2_CSN	BTCX	ANTSW	GPIO36
37	GPIO37	ADC7	I2C0	UART2		BTCX	ANTSW	GPIO37

60pin 支持外设

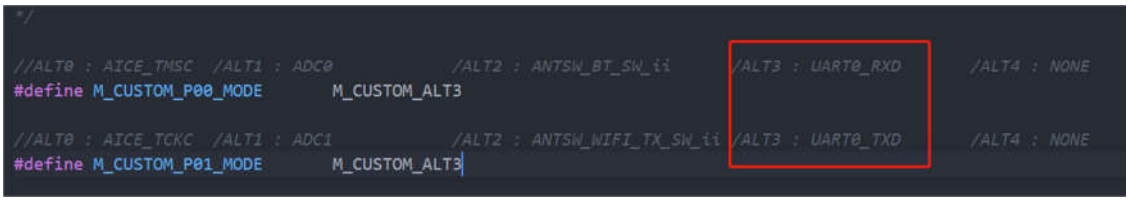
名称	数量
UART	3
I2C	2
SPI Master	3
SPI Slave	2

PWM	8
ADC	8
I2S	2
SDIO	1
PDM RX	2
PWM Audio Tx	1(双声道)

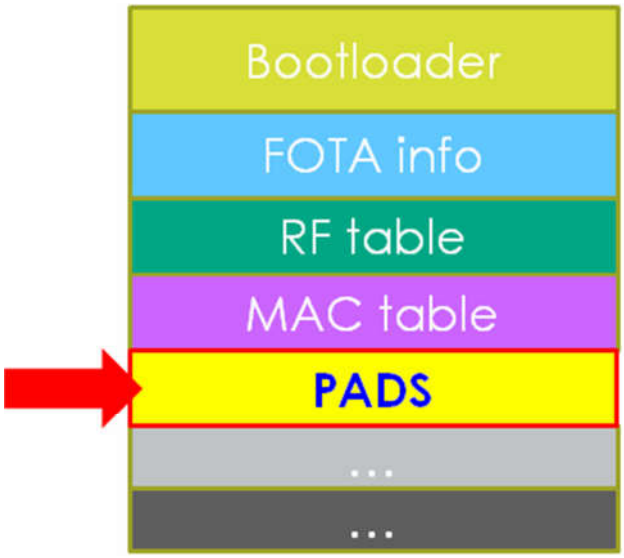
6.10.2. 设定规则

- 每根 PIN 脚都可以强制转换为 GPIO/PWM 功能，若非 GPIO/PWM，仍有整组要求。

  例如 GPIO 00 01 默认为 debug log 输入输出功能，现在只需要 debug tx 查看 LOG，不需要输入功能，即 GPIO 00 可以设定为 GPIO 或 PWM，但不能设定为 AICE 或者其功能



- custom_io_hal.h 在编译时会 mac_atcmd_pad.bin 文件，烧录时被写到 Flash pads 分区，而 FOTA 不会更新此分区



6. 10. 3. 检查机制

- 互斥检查

使用者若是在设置 IO 时有误选或是产生互斥现象，在编译过程中会有报错的机制。例如 Pin04 与 Pin06 同时设定 PWM0 功能

```
//ALT0 : GPIO04      /ALT1 : I2C0_SCL        /ALT2 : NONE
#define M_CUSTOM_P04_MODE          M_CUSTOM_PWM0

//ALT0 : GPIO06      /ALT1 : I2C0_SDA        /ALT2 : NONE
#define M_CUSTOM_P06_MODE          M_CUSTOM_PWM0
```

编译结果

```
/home/jim.wang/work/20Q2.0000.01/components/bsp/soc/ssv6020C/custom_io_chk.h:540:5
: error: #error M_CUSTOM_P06_MODE define PWM0 collision to other pin.
    #error M_CUSTOM_P06_MODE define PWM0 collision to other pin.
     ^
/home/jim.wang/work/20Q2.0000.01/components/bsp/soc/ssv6020C/custom_io_chk.h:542:0
: error: "M_CUSTOM_PWM0_PIN" redefined [-Werror]
  #define M_CUSTOM_PWM0_PIN           (GPIO_06)
 ^
/home/jim.wang/work/20Q2.0000.01/components/bsp/soc/ssv6020C/custom_io_chk.h:393:0
: note: this is the location of the previous definition
  #define M_CUSTOM_PWM0_PIN           (GPIO_04)
 ^
cc1: all warnings being treated as errors
```

- PWM 编号检查

  例如在 PIN06 设定为 PWM9（实际范围是 0-8）。

  注：Pin 编号与 PWM 编号没有强制关联。例如 Pin06 可以设定为 PWM0-8 任一个编号

6.11.GPIO 应用

GPIO 配置参考 6.10 章节，驱动头文件位于 drv_gpio.h/drv_pinmux.h。需要注意以下几个事项。

- custom_io_hal.h 配置文件中，ALT0 代表 GPIO 默认状态，ALT7 代表设定为 GPIO 模式，可以看到芯片 IO 状态，大部份为 GPIO 状态，此类 GPIO 设定 ALT0 或 ALT7 效果一致
- SIO 为 Strapping IO 缩写，表代此 IO 为 Strapping pin，在上电时有电平需求。上电后可以当作普通 GPIO 读写。例如 SIO13 拉高开机，进入烧录模式，拉低开机进入正常模式
- GPIO 默认状态参考 6.7.3 章节

PIN	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7
0	AICE	ADC0	ANTSW	UART0				GPIO00
1	AICE	ADC1	ANTSW	UART0				GPIO01
13	SIO13							SIO13
14	GPIO14			PDMTX0				GPIO14
17	GPIO17	SDIO	UART2					GPIO17
18	GPIO18	SDIO			SPIS1_CSN	SPIM1_CSN		GPIO18
19	GPIO19	SDIO			SPIS1	SPIM1		GPIO19
20	GPIO20	SDIO			SPIS1	SPIM1		GPIO20
21	GPIO21	SDIO			SPIS1	SPIM1		GPIO21
22	GPIO22	SDIO	UART2					GPIO22
29	GPIO29	ADC3			UART1			GPIO29
33	GPIO33				UART1	BTCX		GPIO33
36	GPIO36	ADC6	I2C0	UART2		BTCX	ANTSW	GPIO36
37	GPIO37	ADC7	I2C0	UART2		BTCX	ANTSW	GPIO37

- GPIO 中断硬件不支持双边沿触发功能，只能软件应用层去适配。例如设定为上升沿中断后，在中断中设定为下降沿中断等
- 所有的 IO pin，可以在运行过程中设定为 GPIO 或者 PWM，参考 drv_pinmux.h 文件。例如指 Pin 00 设定为 GPIO 功能，则可 drv_pinmux_force_gpio(0,1)，取消则使用 drv_pinmux_force_gpio(0,0)

```
24 /**
25  * @brief  force digit pinout to GPIO mode.
26  *
27  * @param  pin [IN] select pin to GPIO mode.
28  * @param  isForceGPIO  1 to Force GPIO.
29  *                      0 remove Force GPIO.
30  */
31 void drv_pinmux_force_gpio(uint32_t pin, int isForceGPIO);
32 void drv_pinmux_force_pwm(uint32_t pin, uint32_t pwm_id, int isForcePWM);
```

6.12.PWM 应用

SV32WB0x 支持 8 路 PWM，当 Bus Clock 为 160M 时，PWM 输入频率范围 10Hz---80MHz，4096 阶占空比可调。PWM 可以映射在所有 GPIO 口上使用，例如 projects/pwm/src/board/inc/ssv6020C/P32/custom_io_hal.h 中，把 8 路 PWM 映射到如下 GPIO 口

```
29 #define M_CUSTOM_P00_MODE       M_CUSTOM_ALT3
30
31 //ALT0 : AICE_TCKC  /ALT1 : ADC1          /ALT2 : ANTSW_WIFI_TX_SW_ii /ALT3 : UART0_TXD      /ALT4 : NONE
32 #define M_CUSTOM_P01_MODE       M_CUSTOM_ALT3
33
34 //ALT0 : SIO13      /ALT1 : NONE          /ALT2 : NONE                /ALT3 : NONE           /ALT4 : NONE
35 #define M_CUSTOM_P13_MODE       M_CUSTOM_PWM0
36
37 //ALT0 : GPIO14     /ALT1 : NONE          /ALT2 : NONE                /ALT3 : PDMTX0_DOUT0   /ALT4 : NONE
38 #define M_CUSTOM_P14_MODE       M_CUSTOM_PWM1
39
40 //ALT0 : GPIO17     /ALT1 : SD_DATA2      /ALT2 : UART2_NCTS          /ALT3 : NONE           /ALT4 : NONE
41 #define M_CUSTOM_P17_MODE       M_CUSTOM_PWM2
42
43 //ALT0 : GPIO18     /ALT1 : SD_DATA3      /ALT2 : NONE                /ALT3 : DATASPIS_CSN   /ALT4 : SPISLV
44 #define M_CUSTOM_P18_MODE       M_CUSTOM_PWM3
45
46 //ALT0 : GPIO19     /ALT1 : SD_CMD        /ALT2 : NONE                /ALT3 : DATASPIS_MOSI  /ALT4 : SPISLV
47 #define M_CUSTOM_P19_MODE       M_CUSTOM_PWM4
48
49 //ALT0 : GPIO20     /ALT1 : SD_CLK        /ALT2 : NONE                /ALT3 : DATASPIS_SCLK  /ALT4 : SPISLV
50 #define M_CUSTOM_P20_MODE       M_CUSTOM_PWM5
51
52 //ALT0 : GPIO21     /ALT1 : SD_DATA0      /ALT2 : NONE                /ALT3 : DATASPIS_MISO  /ALT4 : SPISLV
53 #define M_CUSTOM_P21_MODE       M_CUSTOM_PWM6
54
55 //ALT0 : GPIO22     /ALT1 : SD_DATA1      /ALT2 : UART2_NRTS          /ALT3 : NONE           /ALT4 : NONE
56 #define M_CUSTOM_P22_MODE       M_CUSTOM_PWM7
57
```

io 配置文件

```
38  * @param   pwm_id
39  *              0x00 - PWM_0.
40  *              0x01 - PWM_1.
41  *              0x02 - PWM_2.
42  *              0x03 - PWM_3.
43  *              0x04 - PWM_4.
44  *              0x05 - PWM_5.
45  *              0x06 - PWM_6.
46  *              0x07 - PWM_7.
47  * @param   freq_hz         freq_hz range (10 ~ 80000000 - PWM_CLK=160MHz).
48  * @param   duty            range 0 ~ 4096
49  * @param   is_invert       invert the waveform
50  *
51  * @return  -1              The operation error.
52  * @return   0              The operation completed successfully.
53  */
54 int8_t drv_pwm_config(uint8_t pwm_id, uint32_t freq_hz, uint32_t duty, uint8_t is_invert);
```

PWM 配置参数

由于 PWM 为数位电路，其输出频率由 Bus Clock 使用除法得到，所以当 PWM 输出的频率越高，其步进范围越大。

周期 = Bus Clock /设定频率

占空比 = 周期*设定空占比/4096

Bus Clock = 160M		
周期（必须为整数）	频率	占空比精度

2	80MHz	3 阶（0，2048，4096）
3	53.333MHz	4 阶
4	40 MHz	5 阶（0，1024，2048，3072，4096）
5	32MHz	6 阶
…	…	…
4095	39.07KHz	4096 阶（1，2，3，4…4096）
4096	39.0625KHz	4096 阶

例如，当开发者设定 PWM 频率为 45MHz，占空比设定为 1800 时，计算周期数为 4，即实际输出频率为 40MHz，duty 为 1024。当设定的 PWM 频率小于 39.07KHz 时，占空比可以达到最高精度

6.13.I2C 应用

SV32WB0x 支持 I2C 主/从模式，参考 projects/i2c_master/src/app/custom_cmd.c 文件。其中 I2C 从模式格式如下：

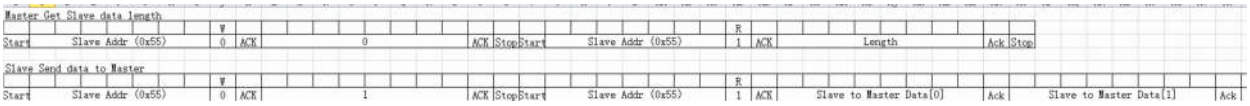
1、 设定从模式参数，其中 0x55 为从机地址，设定回调函数

```
int Cmd_i2c_slave_init(s32 argc, char *argv[])
{
    printf("run i2C Slave init example start...%s %s\n" , __DATE__ , __TIME__);

    drv_i2c_slv_init(DRV_I2C_DEFAULT, 0x55);
    drv_i2c_slv_register_recv_isr(i2c_rx_get_callback);
    drv_i2c_slv_register_tx_done_isr(i2c_tx_done_callback);

    return 0;
}
```

2、 I2C 从读时序

a) 主机读取从机 REG = 0x00 的数据，长度为 1 个 Bytes。代表设备有多少数据需要上报，最长 255。

b) 主机读取从机 REG = 0X01 的数度，长度为第一步中读取的长度，代码实际数据。



3、 I2C 从写

a) 主机向从机 REG = 0X02 写数据

6.14.Flash 读写应用

SV32WB0x SDK 提供 Flash API 对 Flash 进行读/写/擦除操作，根据 NorFlash 的特性，Flash 写的最小单位为一个 page，每个 page 256 字节，则写地址需要 256 字节对齐。擦除的最小单位为一个 sector，每个 sector 4096 字节，则擦除地址需要 4096 字节对齐。flash_sector_erase 与 flash_page_program 的操作，需要使用 OS_EnterCritical()与 OS_ExitCritical()保护起来，防止操作过程任务调度。

- flash_init　　初始 Flash 接口
- flash_sector_erase　　擦除一个 sector，地址为 Flash 的绝对地址
- flash_page_program　　写入一个 page，地址为 Flash 的绝对地址
- memcpy　　从 Flash 中的绝对地址+0x30000000 拷贝数据出来

例如需要读写 0x9000 的 Flash 4K 大小数据，则可以如下：

```
flash_init()
OS_EnterCritical();
flash_sector_erase(0x9000)
for(i=0;i<size;i+=256)
{
  flash_page_program(0x9000+i, &write_buffer[i],256);
}
OS_ExitCritical();
memcpy(&read_buffer[0], 0x30000000+0x9000, 4096);
```

6.15.FOTA 应用

6.15.1.　　FOTA 介绍

FOTA 是模块透过 Wi-Fi 下载并更新固件的方法。

SV32WB0x 提供两种方式更新固件

- File system method

  透过将固件下载 file system 后，重新启动后透过 bootloader 刻录至代码主分区
- Ping pong method

Flash 切成 A/B 区， A 区开机的状况下固件会刻录到 B 区，断电上电会直由 B 区开机。 B 区开机的状况下固件会刻录到 A 区，断电上电后会由 A 区开机。

提供三种 protocol（升级固件的数据来源）

- HTTP
- TFTP
- RAW

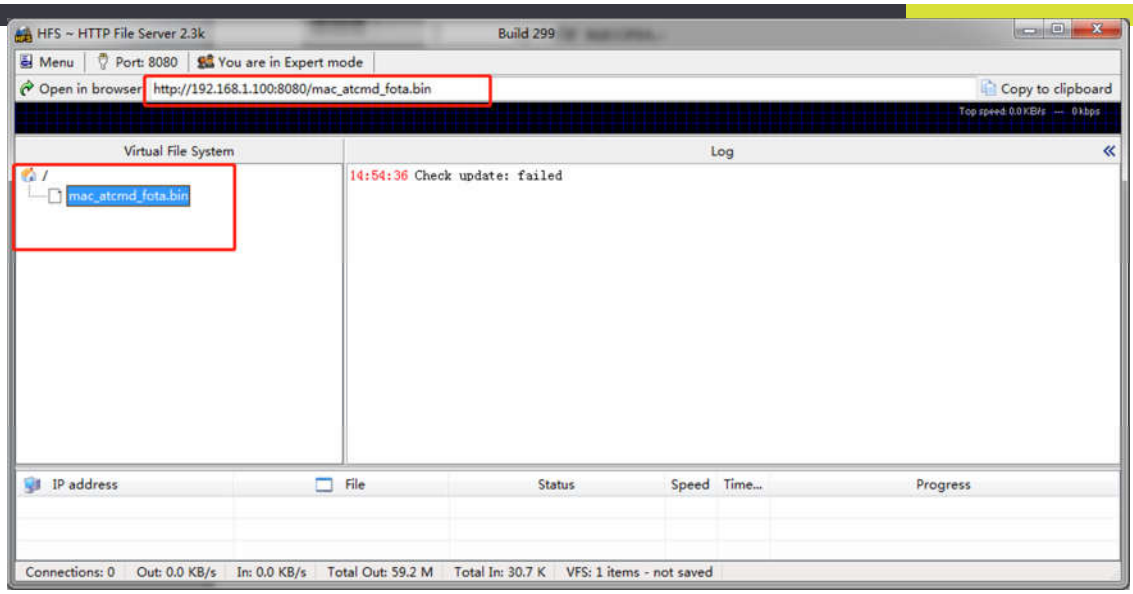
其中 HTTP/TFTP 是设备主动下载数据，而 RAW 是被动接收数据，适合对接云端 SDK 的 FOTA 部份。通过设定 feature.mk 配置使用文件系统还是 Ping pong 升级方式

```
48  # 0 ==> disable fota, 1==> filesystem fota, 2 ==> ping pong fota.
49  FOTA_OPTION                    := 2
```

6.15.2. FOTA API（HTTP 方式）

ota_set_server(0,”192.168.123.58”,8000)	设定 protocol，IP 地址，端口
ota_set_parameter(“mac_atcmd_fota”,NULL);	设定下载的文件信息（不需要加.bin 后缀）
ota_update();	开始升级
ota_reboot();	重启，更新固件

PS：HTTP 服务器可以使用 HFS 软件搭建



6.15.3. FOTA API（raw 方式）

```
00316:    ota_set_server(RAW,"xxx",0);
00317:    ota_set_parameter("xxx","xxx");
00318:    ota_register_state_callback(tuya_ota_state_cb);//checkout ota if init
00319:
00320:    ota_register_status_callback(tuya_ota_status_cb);//checkout ota if success
00321:
00322:
00323:
00324:    ota_update();
00325:
00326:    while(tuya_ota_state == OTA_STATE_NONE)
00327:    {
00328:        OS_MsDelay(500);
00329:    }
```

```
00299: static void tuya_ota_state_cb(OTA_STATE_TYPE event)
00300: {
00301:     tuya_ota_state = event;
00302: }
00303:
00304: static void tuya_ota_status_cb(OTA_STATUS event)
00305: {
00306:     tuya_ota_status = event;
00307: }
```

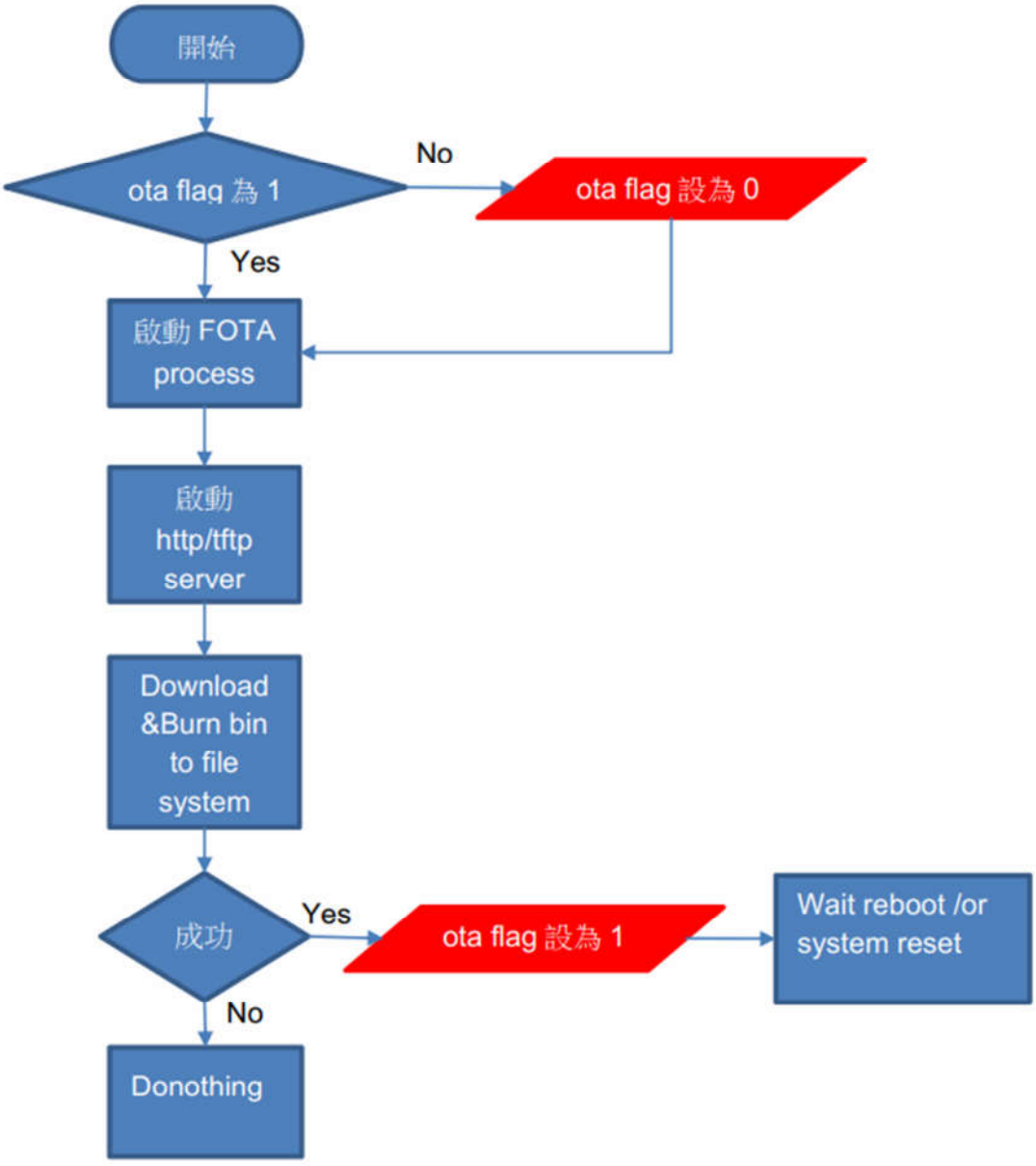
其中“xxx”表示任意字串，注册两个回调，当 tuya_ota_state == OTA_STATE_NONE 时，说明还未初始化完毕

```
/* gateway upgrade result notify */
STATIC VOID __gw_upgrade_notify_cb(IN CONST FW_UG_S *fw, IN
{

    if(tuya_ota_status == OTA_SUCCESS)
    {
        printf("FOTA success.Reboot Now\n");
        ota_reboot();
    }

}

/* gateway upgrade process */
STATIC OPERATE_RET __gw_upgrage_process_cb(IN CONST FW_UG_
                                           IN CONST
{
    OPERATE_RET ret = OPRT_OK;
    uint32_t _off_set = 0;


    PR_DEBUG("%s total_len %d  offset %d current len %d\n",\
        __func__,total_len,offset,len);


    input_raw_data((char *)data,len);
    *remain_len = 0;


     return OPRT_OK;
}
```

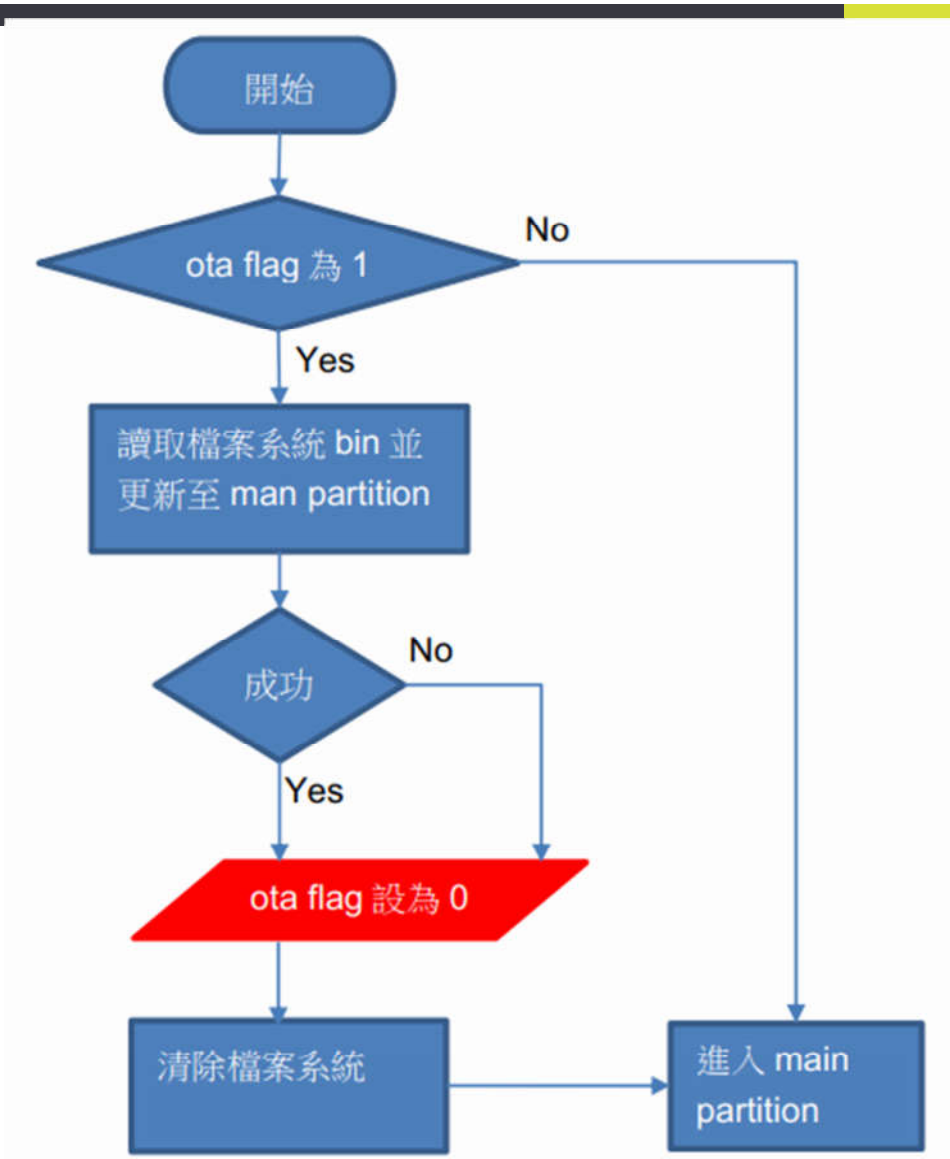
当通知下载数据会，呼叫 input_raw_data，把数据交给内部处理。当 tuya_ota_state == OTA_SUCCESS 后，呼叫 ota_reboot()更新固件

6.15.4. FOTA 原理介绍

6.15.4.1. 使用文件系统

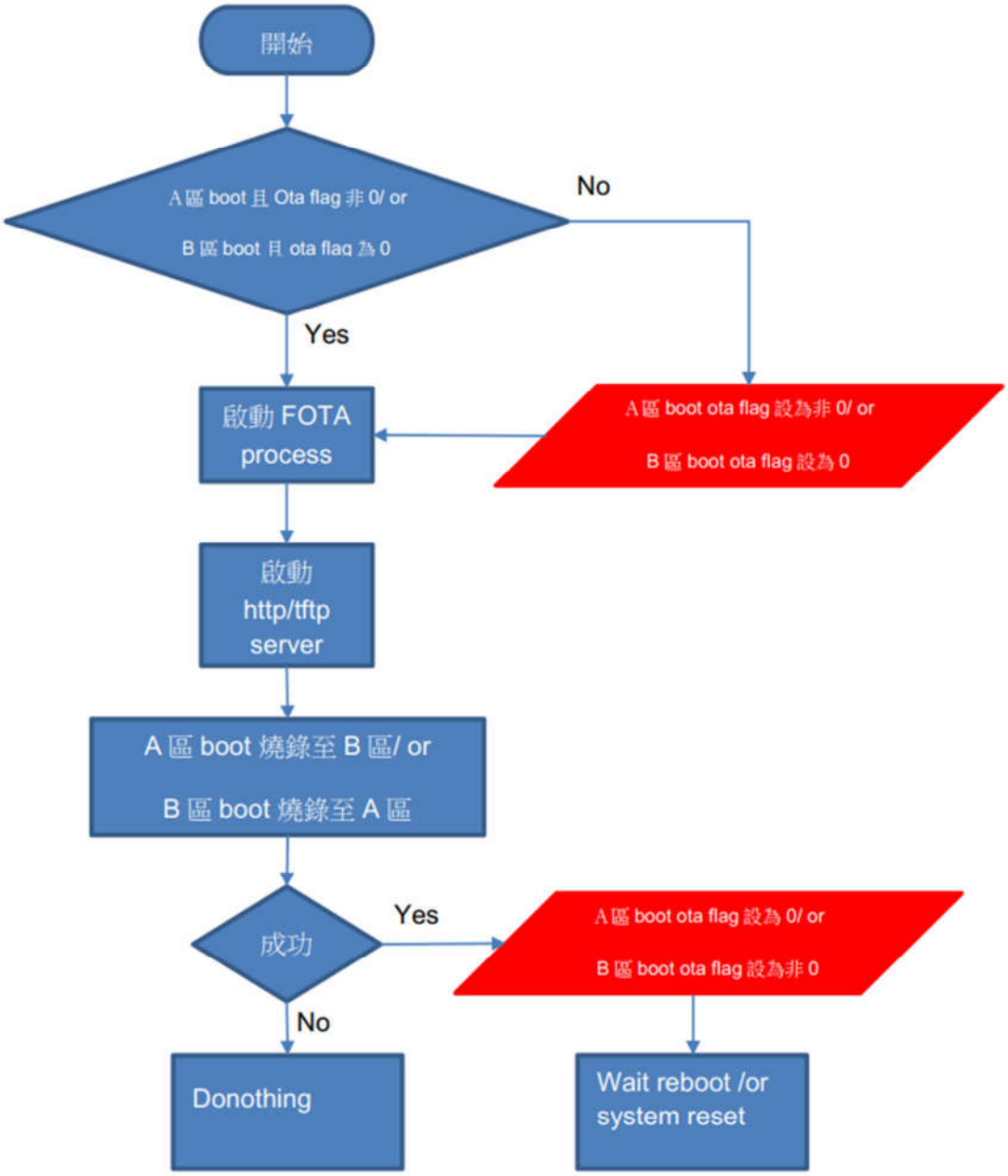


下载过程

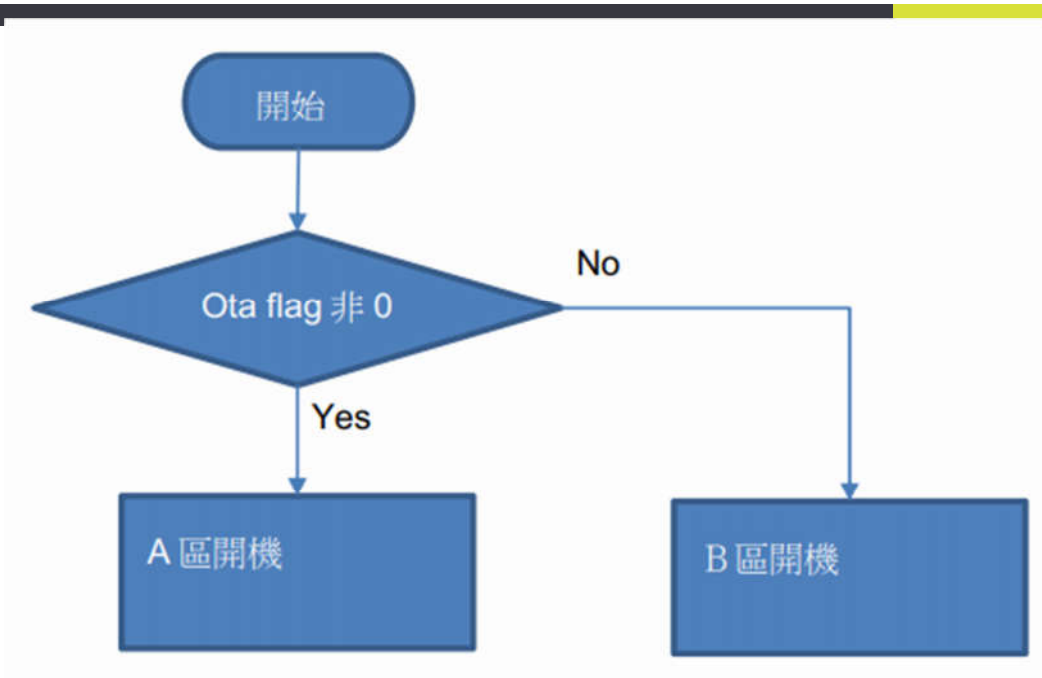


更新过程

6.15.4.2. 使用 A/B 区方式



下载过程



升级过程

注意

file system method or ping pong method 选定一种方式做 FOTA 后，之后都只能使用选定的方式做 FOTA，必须重新刻录 bootloader 才能替换另一种方式

6.15.5. FOTA 文件解析

FOTA.bin 由 hdr 与 playload 组成，其中 hdr 长度 84bytes

```
typedef union {
    char    raw[OTA_HDR_SIZE];
    struct {
        unsigned int magic;
        unsigned int a_size;
        unsigned int b_size;
        unsigned int major_ver;
        unsigned int minor_ver;
        char a_md5[32];
        char b_md5[32];
    } hdr;
} ota_hdr_st;
```

- magic　　表示当前文件升级类型。文件系统方式为 0x84848711，A/B 区方式为 0x73737600
- a_size　　表示 FOTA.bin 中，XIP1 的固件大小

- b_size　　表示 FOTA.bin 中，XIP2 的固件大小
- major_ver　主版本号
- minor_ver　次版本号
- a_md5[32]　XIP1 的 md5sum 值
- b_md5[32]　XIP2 的 md5sum 值
- 使用文件系统文件升级

hdr(84Bytes)	playload xip1

- 使用 A/B 区升级

Hdr(84Bytes)	playload xip1	playload xip2

6.14.6. 自定义升级过程

6.14.6.1. 非 AB 区方式

文件系统

- 下载 hdr，提取中 a_size 与 a_md5[32]的值
- 下载 playload，保存文件名为 ota.bin
- 计算 ota.bin 的 md5 值
- 比较 md5 的值与 a_md5[32]，如果一致，则说明下载正确
- 呼叫 ota_update_flag()，重启后 bootloader 会自动更新固件

非文件系统

- 下载 hdr，提取中 a_size 与 a_md5[32]的值
- 下载 playload 到自定义的 Flash 位置
- 计算 ota.bin 的 md5 值
- 比较 md5 的值与 a_md5[32]，如果一致，则说明下载正确
- 标识某个 Flash 地址，bootloader 中确认是否需要升级，然后搬运数据到 xip1 分区

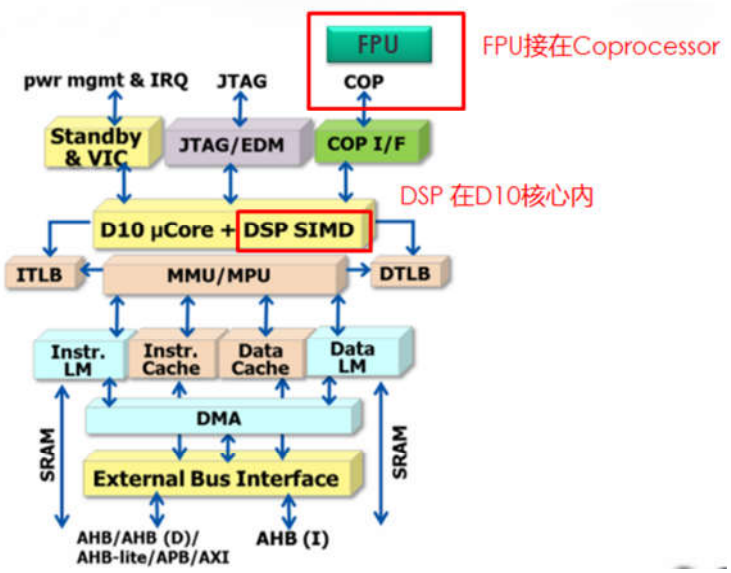
6.15.6. 镜像压缩升级

为了高效使用 Flash，SDK 中提供镜像压缩升级的功能。Fota 升级固件，经过 lzma 算法压缩后，通过 HTTP 或者 FTP 下载至文件系统中，在 Bootloader 升级阶段，解压 Fota 升级固件，并拷贝到主分区，完成 FOTA 升级。lzma 压缩比约为 50%，则一个 500KB 左右的 FOTA 固件，压缩后约 250KB。

Feature.mk 中设定 LZMA_EN := 0 使用 lzma 功能

6.16.DSP/FPU 应用

SV30WB0x MCU 内置 DSP 与 FPU 处理单元，Block Diagram 如图如示：



6.16.1. FPU 的开启以及注意点

当使用我司提供的编译器 nds32le-mculib-v3s 时，默认打开 FPU 功能（单精度浮点数）。当遇到单精度浮点运行时，自动使用 FPU 寄存器与浮点指令参与运算。当运到整型或双精度浮点数时，使用软计算方法。

当移植例如 Speex 语音算法库时，会内置定点或浮点的算法。这个需要开发者实际测量，选择不同的算法，对效能上有多大的差异，再决定使用定点或浮点的算法。

6.16.2. DSP 的开启与使用

SV32WB0x SDK 中，build/compiler.mk 文件默认开启 DSP 硬件加速功能，其中：

- -ldsp 链接编译器提供的 dsp 库，但不一定是硬件加速
- -mext-dsp 启动硬件 DSP 加速功能

```
22  ifeq ($(strip $(MCU_DEF)), ANDES_D10F)
23  DEFAULT_CFLAGS          += -mext-dsp \
24                             -mcmodel=large
25  DEFAULT_AFLAGS          += -mext-dsp \
26                             -mcmodel=large
27  #LDFLAGS                 += -ldsp -nostdlib
28  LDFLAGS                 += -ldsp
29  else
30  ifeq ($(strip $(MCU_DEF)), ANDES_D10)
31  DEFAULT_CFLAGS          += -mcpu=d1088 -march=v3 -mext-dsp -ldsp\
32                             -mcmodel=large
33  DEFAULT_AFLAGS          += -mcpu=d1088 -march=v3 -mext-dsp -ldsp\
34                             -mcmodel=large
35  LDFLAGS                 += -ldsp
```

DSP 库可以分为八类，如下图所示

- nds32_basic_math.h
- nds32_complex_math.h
- nds32_controller_math.h
- nds32_filtering_math.h
- nds32_matrix_math.h
- nds32_statistics_math.h
- nds32_transform_math.h
- nds32_utils_math.h

Description	Function name	Supported since	Page
	nds32_cfft_rd4_q15	1	252
	nds32_cifft_rd4_f32	1	249
	nds32_cifft_rd4_q31	1	251
	nds32_cifft_rd4_q15	1	253
Complex FFT Functions	nds32_cfft_f32	3	255
	nds32_cfft_q31	3	257
	nds32_cfft_q15	3	259
	nds32_cifft_f32	3	256
	nds32_cifft_q31	3	258
	nds32_cifft_q15	3	260
DCT Type II Functions	nds32_dct_f32	1	262
	nds32_dct_q31	1	264
	nds32_dct_q15	1	266
	nds32_idct_f32	1	263
	nds32_idct_q31	1	265
	nds32_idct_q15	1	267
DCT Type IV Functions	nds32_dct4_f32	1	269
	nds32_dct4_q31	1	271
	nds32_dct4_q15	1	273
	nds32_idct4_f32	1	270
	nds32_idct4_q31	1	272
	nds32_idct4_q15	1	274
Real FFT Functions	nds32_rfft_f32	1	276
	nds32_rfft_q31	1	278
	nds32_rfft_q15	1	280
	nds32_rifft_f32	1	277
	nds32_rifft_q31	1	279
	nds32_rifft_q15	1	281

详细 API 请参考《Andes_DSP_Library_V3_User_Manual_UM119_V2.2.pdf》

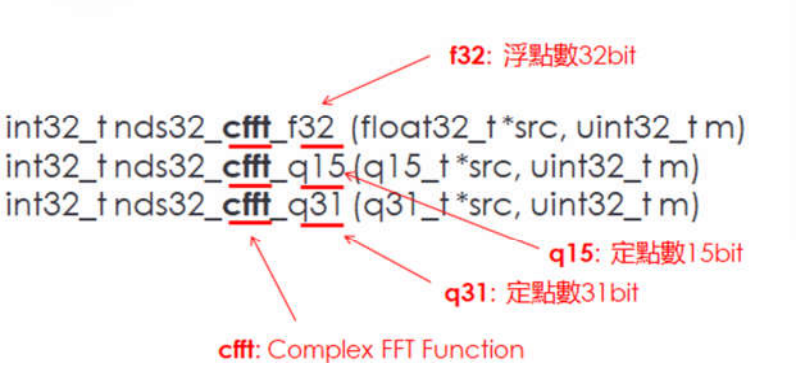
例如，使用 DSP 函数，计算 CFFT，参考代码如下：

```
#include "math.h"
//DSP Header
#include "nds32_math_types.h"
#include "nds32_transform_math.h"
#include "nds32_filtering_math.h"

int Cmd_dsp_fft_f32_test(s32 argc, char *argv[])
{
    int i = 0;
    uint32_t sample_num = 1024;

    float32_t *data_of_f32;
    data_of_f32 = (float32_t*)OS_MemAlloc(2*1024*sizeof(float32_t));

    for(i=0; i<1024; i++)
    {
        data_of_f32[i*2] = cos(2*PI*i/1024);
        data_of_f32[i*2+1] = 0;
    }

    start_tick = OS_GetUsSysTick();

    nds32_cfft_f32(data_of_f32, 10);

    spend_tick = OS_GetUsSysTick() -start_tick;

    printf("\nDSP FFT f32 Test (%d points) spend %d us\n", sample_num, spend_tick);

    OS_MemFree(data_of_f32);

    return 0;
}
```

include DSP header

可替換成FFT其他function
nds32_cfft_q15
nds32_cfft_q31

其中



6.16.3. DSP 与 FPU 对效能影响

	Disable All	Enable DSP Only	Enable FPU Only	Enable DSP + FPU
nds32_cfft_f32	3151	3149	232	228
nds32_cfft_q15	227	141	240	145
nds32_cfft_q31	306	186	336	188

*FFT 跑一次所需的时间(us) (数字愈小愈好)

*nds32_cfft_f32: Complex FFT, 1024 point float32_t

*nds32_cfft_q15: Complex FFT, 1024 point q15_t

*nds32_cfft_q31: Complex FFT, 1024 point q31_t

注意点：

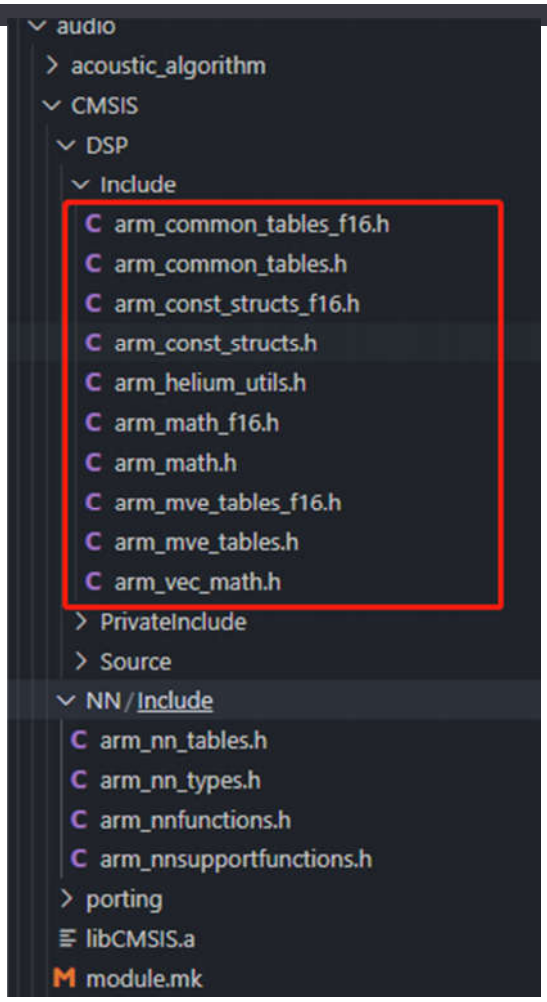
- Enable DSP 只对使用整数型式的 Function 有加速，若参数使用浮点数则无效
- 需要 Enable FPU 功能才能加速浮点数的运算
- Enable FPU 时，每个 Task 的 Content Switch 都会多消耗 128 Byte
- Enable FPU 时，code size 会稍大

6.16.4. CMSIS DSP Library

为了方便客户使用 DSP 库，移植 DSP 应用，SV32WB0x SDK 中移植了 CMSIS 5.7.0，使用方法如下：

- feature.mk 使能 CMSIS_EN 功能

- 添加 CMSIS 头文件路径：components/audio/CMSIS



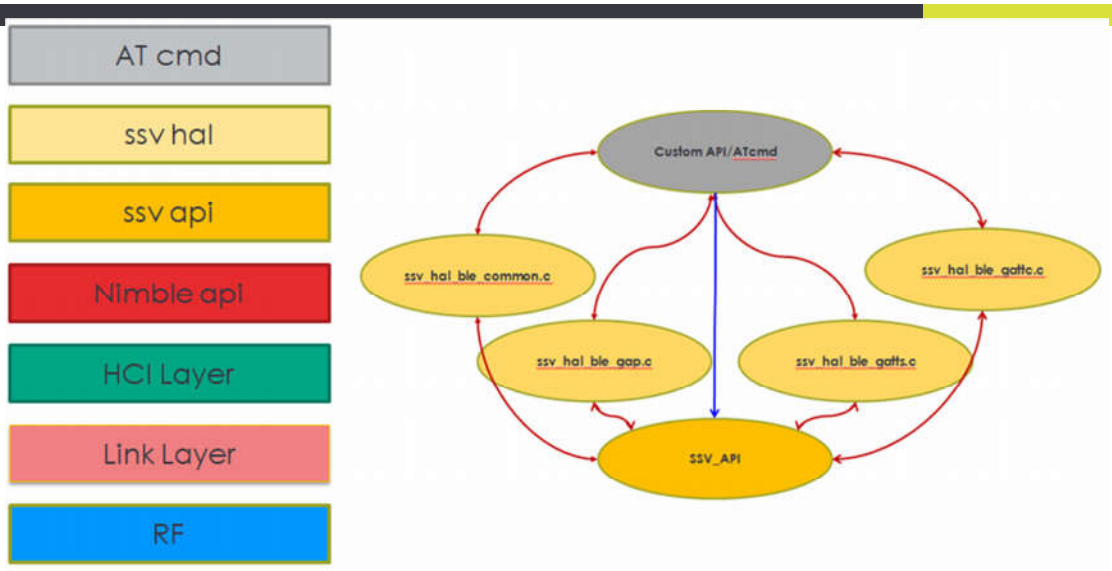
- 使用 CMSIS 标准 API 编程

6.17.BLE 应用

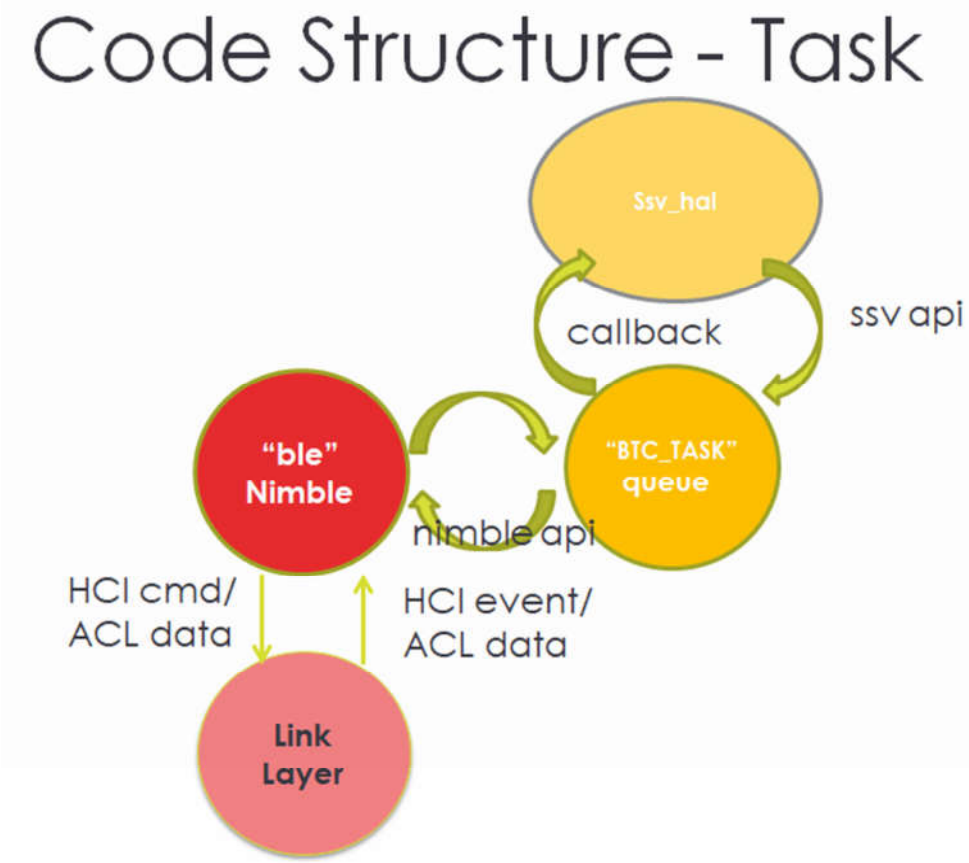
SV32WBOX 内置 BLE 5.0 功能，与 WiFi 共用天线。支持主/从模式、支持扫描与广播者角色、支持 GATT 与 Mesh profile 功能、支持 BLE 标准 HCI 命令。

SV32WBOX SDK 提供 ble app uart 范例，让开发者参考，快速上手 BLE 功能。

6.17.1. BLE 软件架构



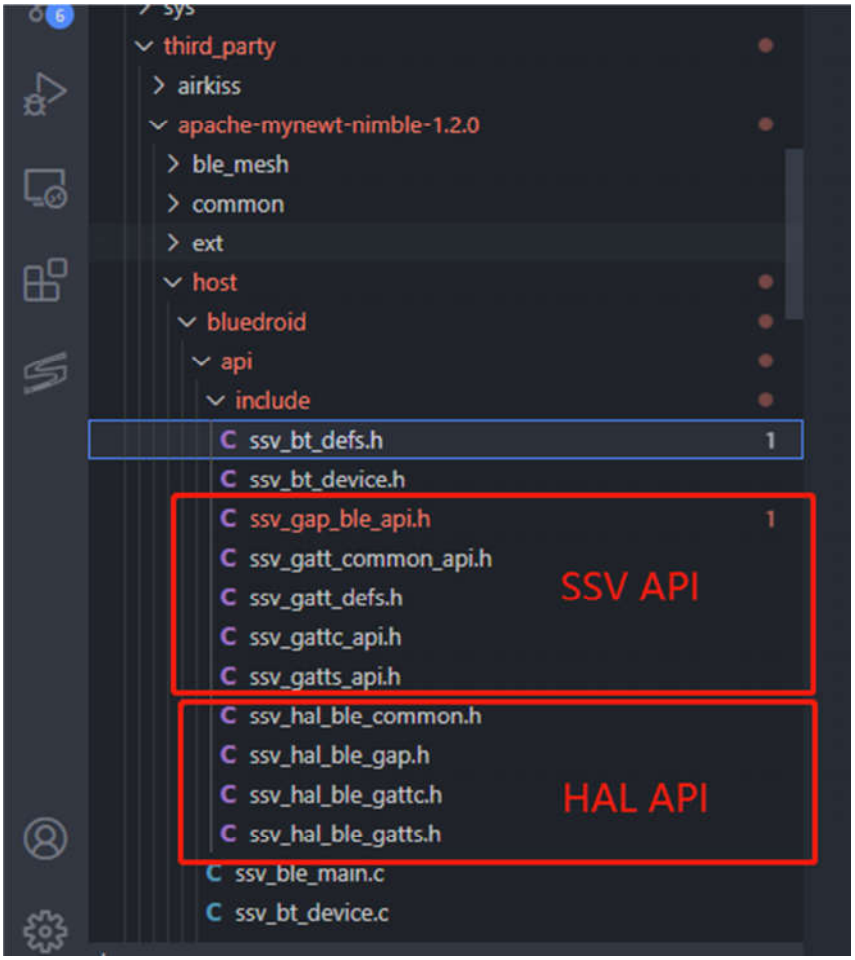
代码层次图



任务层次图

- nimble：BLE 协议栈，通过标准 HCL cmd/ACL data 与 LinkLayer 通信，向上提供 nimble api 功能接口。

- BTC_TASK：对 nimble api 进行封装，提供 ssv api，客户可基于此层 API 做开发
- SSV_HAL：对 SSV API 进一步封装，客户也可基于此层 API 做开发



6.17.2. ble app uart 使用说明

一、编译固件

a) 修改 projects\mac_atcmd\mk\feature.mk 配置文件，打开 ble 功能。其中 ble 静态内存占用 75K 左右（不可释放），动态内存使用 35K 左右（可释放）

```
19 ##############################################################
20 # Partition Setting, influence FOTA
21 ##############################################################
22 SETTING_PARTITION_MAIN_SIZE      := 860K
23 SETTING_FLASH_TOTAL_SIZE         := defined_by_chip
24 SETTING_PARTITION_USER_RAW_SIZE := 8K
25 SETTING_PSRAM_HEAP_SIZE          := 0
```

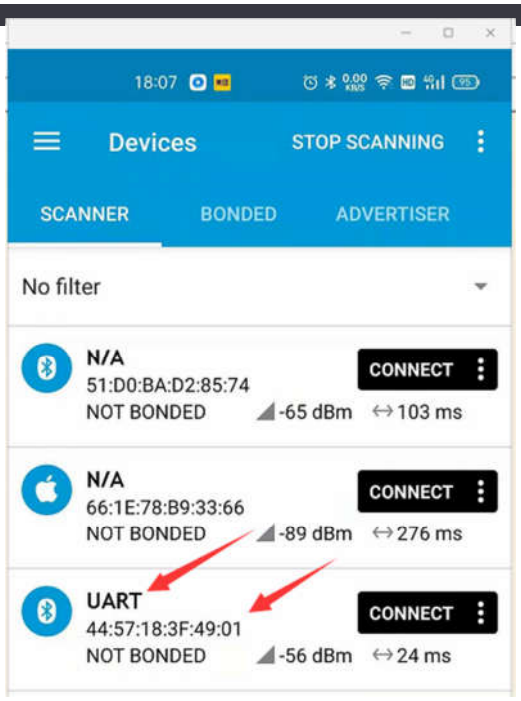
```
66  ##################################################################
67  # BLE option
68  ##################################################################
69  BLE_EN                          := 1
70  BLE_DEBUG_ONLY                  := 0
71  BLE_SLAVE_EN                    := 1
72  BLE_SCAN_EN                     := 0
73  ##if enable BLE_MASTER_EN, BLE_SCAN_EN must be enabled
74  BLE_MASTER_EN                   := 0
75  BLE_DATA_LENGTH_EXTENSION_EN    := 0
76
77  # Thirdparty
78  MESH_BLE_EN                     := 0
79  ##if enable GATTS or GATTC, GAP must be enabled
80  BLE_GAP_EN                      := 1
81  BLE_GATTS_EN                    := 1
82  BLE_GATTC_EN                    := 0
83  BLE_SM_LEGACY_EN                := 0
84  BLE_SM_SC_EN                    := 0
85  BLE_GATTS_API_TEST_EN           := 0
86  BLE_GATTC_API_TEST_EN           := 0
87  BLE_GAP_ATCMD_EN                := 1
88  BLE_GATTS_ATCMD_EN              := 1
89  BLE_GATTC_ATCMD_EN              := 0
90  SETTING_BLE_UART                := 1
91  BLE_GPIO_PROFILE                := 0
92  MESH_BLE_OTHER_MODEL_EN         := 0
93  MESH_BLE_AUTO_RUN_EN            := 0
94  #BLE_AGGR_ADV_REPORT ONLY support MESH(HCI_RAM_MODE)
95  BLE_AGGR_ADV_REPORT             := 0
```

b) 编译生产 mac_atcmd_all.bin 进行烧录(波特率默认 115200)，开发板上电打开串口工具发送 AT+ble_uart_start 开启蓝牙，出现如下 log 即蓝牙开启正常。

```
AT+ble_uart_start=OK

?>conn complete
btc_gap connection established; status=0
btc_gap slave
[gatts_ble_uart_profile_event_handler] SSV_GATTS_CONNECT_EVT: 1
SSV_GATTS_MTU_EVT, MTU 256**evt_len = 0, loop_evt 806038128, msgh_called 1**
ble_hci_rx_ssv invalid input len 0
```

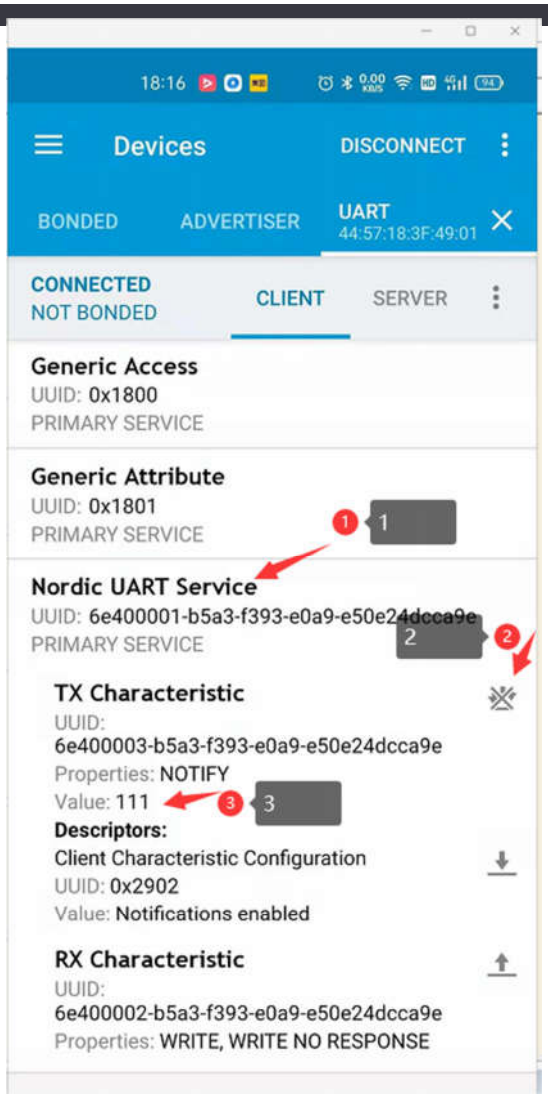
```
Send commands to active session
AT+ble_uart_start
```

c) 手机安装 nrf connect 软件，连接 UART 设备



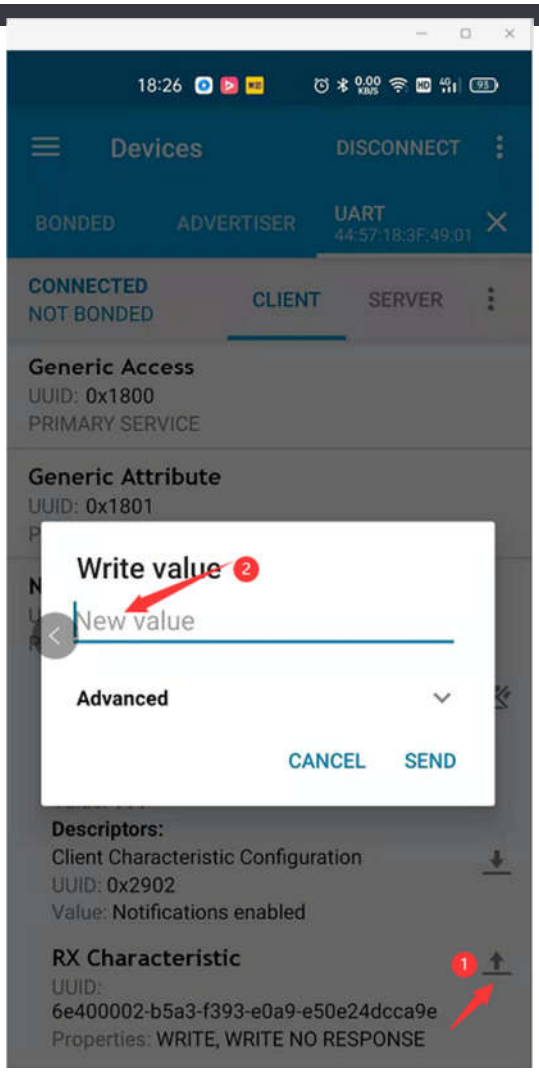
d) SV32WB0X 通过蓝牙给手机发送数据

手机 nRF connex 工具第一步点击 1 处的 Nordic UART service 会弹出下列选择框，第二步点击 2 处的图标直到显示如图所示，串口工具输入 AT+ble_uart_send=111，既可在 3 处看到手机收到的数据。



e) 手机通过蓝牙给 SV32WB0X 发送数据

点击 1 处会弹出下图输入框，以发送 222 为例，Write value 框中输入 222 点击 SEND 发送。



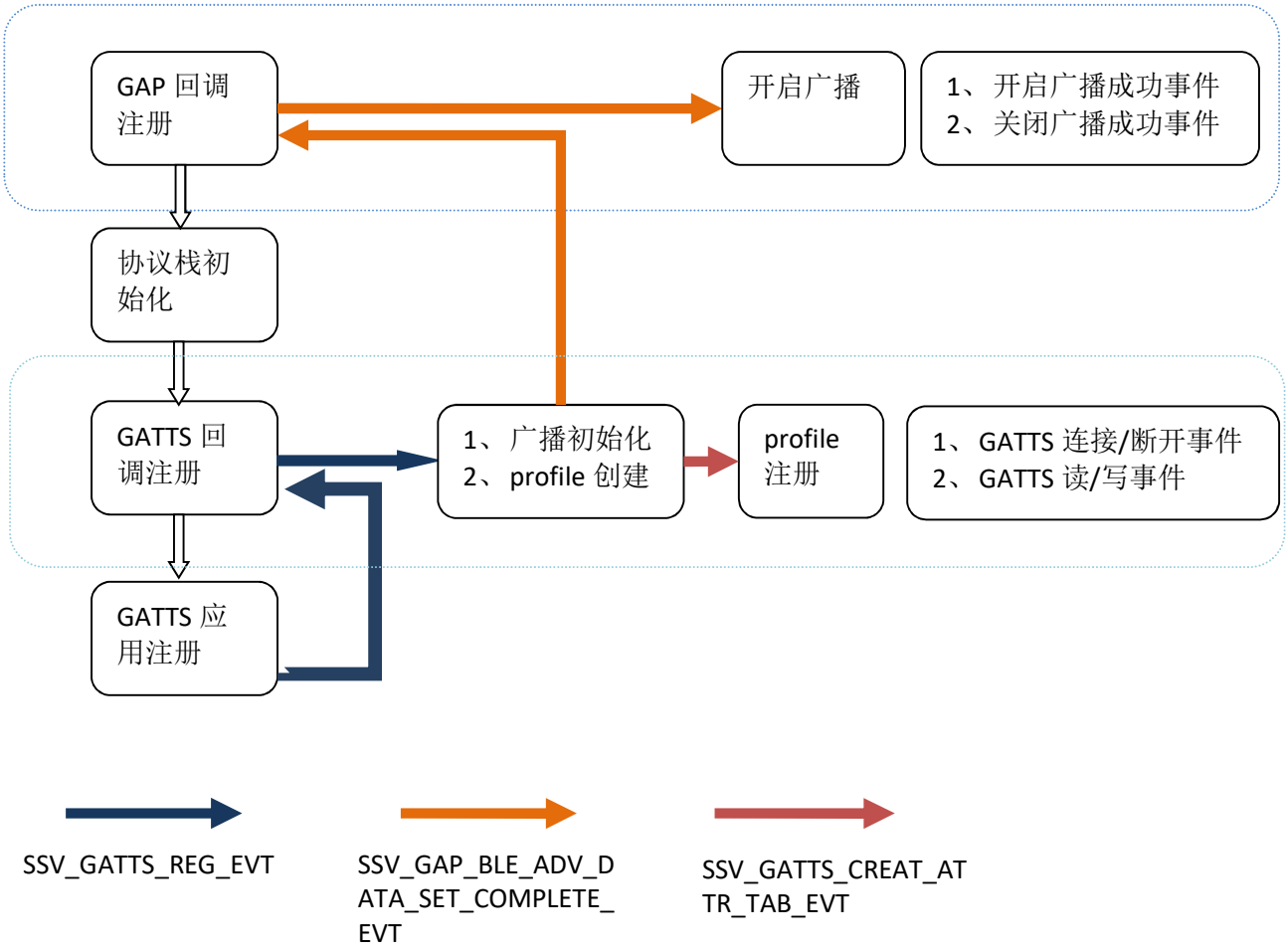
◆

6.17.3. ble app uart 代码分析

```
int At_ble_uart_init(stParam *param)
{
    int ret = 0;
    if (param->argc == 1) {
        strncpy(sg_ble_uart_device_name, param->argv[0], MAX_DEVNAME_LEN);
    } else {
        strncpy(sg_ble_uart_device_name, BLE_UART_DEFAULT_DEVICE_NAME, MAX_DEVNAME_LEN);
    }
    //注册GAP事件，例如设定adv参数，断线，连线等事件
    ssv_ble_gap_register_callback(ssv_ble_uart_gap_event_cb);

    do {
        //nimble协议栈与BTC task初始化
        ret = ssv_hal_ble_common_init();
        if (ret) {
            printf("[%s] failed in line %d\n", __func__, __LINE__);
            break;
        }
        //注册GATTS事件，例如GATT读写/断线/连线/MUTE请求
        ret = ssv_ble_gatts_register_callback(gatts_event_handler);
        if (ret){
```

```
    //固定写法
    ret = ssv_ble_gatts_app_register(SSV_BLE_UART_APP_ID);
    if (ret){
        printf("[%s] failed in line %d gatts app register error, error code = %x\n", __fu
        break;
    }
} while(0);
if (ret) {
    return -1;
}

ssv_err_t ssv_ret = ssv_ble_gatt_set_local_mtu(BLE_UART_GATTC_MTU_SIZE);
if (ssv_ret){
    printf("[%s] line %d, set local MTU failed, error code = %x", __func__, __LINE__, ssv_
}
```

- ssv_ble_gap_register_callback
  - 注册 GAP 相关事件回调函数，例如设定广播参数完成，开始广播完成，停止广播完成等事件
- ssv_hal_ble_common_init
  - 初始化 ble 协议栈与开启 BTC 任务
- ssv_ble_gatts_register_callback
  - 注册 GATTS 相关事件回调函数，例如 probfile 读写事件，GATTS 断线连接事件等
- ssv_ble_gatts_app_register
  - 注册应用标号，固定写法，可触发 SSV_GATTS_REG_EVT 事件，触发 GATTS 与 GAP 回调事侦探



事件流程图

7. mac_atcmd 测试使用范例

参考《SV32WB0x CLI 指令集》