



Unisoc Confidential For hiar

Android 10.0 IDH 包编译使用指南

文档版本	V1.3
发布日期	2020-10-13

版权所有 © 紫光展锐（上海）科技有限公司。保留一切权利。

本文件所含数据和信息都属于紫光展锐（上海）科技有限公司（以下简称紫光展锐）所有的机密信息，紫光展锐保留所有相关权利。本文件仅为信息参考之目的提供，不包含任何明示或默示的知识产权许可，也不表示有任何明示或默示的保证，包括但不限于满足任何特殊目的、不侵权或性能。当您接受这份文件时，即表示您同意本文件中内容和信息属于紫光展锐机密信息，且同意在未获得紫光展锐书面同意前，不使用或复制本文件的整体或部分，也不向任何其他方披露本文件内容。紫光展锐有权在未经事先通知的情况下，在任何时候对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负任何与本文件相关的直接或间接的、任何伤害或损失。

请参照交付物中说明文档对紫光展锐交付物进行使用，任何人对紫光展锐交付物的修改、定制化或违反说明文档的指引对紫光展锐交付物进行使用造成的任何损失由其自行承担。紫光展锐交付物中的性能指标、测试结果和参数等，均为在紫光展锐内部研发和测试系统中获得的，仅供参考，若任何人需要对交付物进行商用或量产，需要结合自身的软硬件测试环境进行全面的测试和调试。

Unisoc Confidential For hiar

紫光展锐（上海）科技有限公司



前言

概述

本文档主要介绍 Android 10.0 IDH 包的内容、结构、以及如何进行编译并将编译输出下载到手机中。

本文档是基于 Linux njand02 3.19.0-25-generic #26~14.04.1-Ubuntu SMP Fri Jul 24 21:16:20 UTC 2015 x86_64 x86_64 GNU/Linux 的编译环境撰写，故在其他版本的 host 环境上可能存在差异。

读者对象


本文档主要适用于 Android 10.0 平台开发的开发人员。

缩略语

缩略语	英文全名	中文解释
IDH	Independent Design House	独立方案设计公司
PAC	Package	打包 android 镜像一种文件格式
OTA	Over-the-Air Technology	空中下载技术
WCN	Wireless Communication Network	无线通信网络

符号约定

在本文中可能出现下列标志，它所代表的含义如下。

符号	说明
 说明	用于突出重要/关键信息、补充信息和小窍门等。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害。

变更信息

文档版本	发布日期	修改说明
V1.0	2019-09-03	撰写客户指南文档。
V1.1	2019-12-16	添加平台 UIS8581E\SL8541E。
V1.2	2020-03-16	结构调整、内容优化、格式更新等。
V1.3	2020-10-13	更新模板。

关键字

IDH、编译。

Unisoc Confidential For hiar

目 录

1 概述.....	1
1.1 IDH 包组件	1
1.2 IDH 包解压	2
1.2.2 开源源码解压	2
1.2.3 非开源部分解压	3
2 编译环境准备.....	5
2.1 硬件环境要求	5
2.2 软件环境要求	5
2.2.1 操作系统要求	5
2.2.2 工具包安装	5
2.2.3 python 安装	5
2.3 常见问题说明	6
2.3.1 系统库缺失	6
2.3.2 ubuntu 版本较低	6
2.3.3 Java 版本错误	6
3 编译操作指导.....	7
3.1 编译步骤	7
3.1.1 完整编译	7
3.1.2 单独编译	12
3.2 版本编译说明	15
3.2.1 Userdebug 版本编译	15
3.2.2 User 版本编译	15
3.2.3 GMS 版本编译	16
3.2.4 GSI 版本编译	16
3.2.5 运营商版本编译	17
3.3 Android 10.0 编译新特性	17
3.3.1 dynamic partitions 编译	17
3.3.2 Mainline Module 集成	18
3.4 编译结果验证	19
3.4.1 ADB Push 验证	19
3.4.2 镜像烧录验证	20
3.5 常见问题说明	21
3.5.1 非开源 apk 重新签名说明	21
3.5.2 编译报错说明	22
3.5.3 编译选项说明	22

3.5.4 库的依赖关系说明	22
4 OTA 包编译及差分包制作说明	23
4.1 相关文件准备	23
4.2 OTA 包编译	24
4.3 OTA 差分包制作	24
5 pac 制作及下载说明	25
5.1 pac 制作	25
5.1.1 工具制作	25
5.1.2 脚本制作	26
5.2 pac 下载说明	26
6 Android 10.0 编译规则的新变化	28
6.1 新增的规则	28
6.1.1 限制 HOST TOOLS 的使用	28
6.1.2 调用其他编译系统时的.PHONY 规则	28
6.1.3 规范.PHONY 规则的使用	28
6.2 变更的规则	29
6.2.1 规范模块的命名规则	29
6.2.2 `DIST_DIR`, `dist_goal` 变更为 `dist-for-goals`	30
6.2.3 拆分 PRODUCT_PACKAGES	30
6.3 弃用的规则	30
6.3.1 不建议使用隐式 make 规则	30
6.3.2 禁用 make 的 export 和 unexport 关键字	31
6.3.3 从 Android.mk 中移除 BUILD_NUMBER	31
6.3.4 弃用 USER	31
6.3.5 弃用 LOCAL_MODULE_TAGS := eng debug	31
6.3.6 Android.mk 不再支持 windows 交叉编译	32
6.3.7 弃用 `*.c.arm` / `*.cpp.arm`	32
7 参考文档	33

图目录

图 1-1 IDH 包组成	1
图 1-2 非开源代码组成	2
图 1-3 IDH 包结构	2
图 1-4 开源源码解压	3
图 1-5 非开源部分解压	4
图 3-1 初始化环境变量及命令	7
图 3-2 编译工程列表	8
图 3-3 编译参数	9
图 3-4 add_lunch_combo 新增编译工程	9
图 3-5 COMMON_LUNCH_CHOICES 新增编译工程	10
图 3-6 编译输出目录说明	11
图 3-7 项目名目录	12
图 3-8 BSP 独立编译输出 out 目录结构	14
图 3-9 编译类型选择	15
图 3-10 编译工程选择	15
图 3-11 编译版本及编译变量选择	15
图 3-12 GSI 仓库	16
图 3-13 包含 GSI 的 super 镜像	16
图 3-14 GSI 版本编译	16
图 3-15 运营商编译说明	17
图 3-16 super.img	18
图 3-17 Mainline Module 目录结构	18
图 3-18 Mainline Module 发布	19
图 3-19 fastboot 命令	20
图 3-20 烧录 system 镜像	21
图 3-21 烧录物理分区	21

图 3-22 预编译方式自动签名	22
图 4-1 bin 文件	23
图 4-2 OTA 包	24
图 5-1 工具制作 pac.....	25
图 5-2 修改 size 大小	26
图 5-3 ResearchDownload 下载工具使用说明	27

Unisoc Confidential For hiar

表目录

表 3-1 fastbootd/bootloader 模式说明	20
---------------------------------------	----

Unisoc Confidential For hiar

1 概述

1.1 IDH 包组件

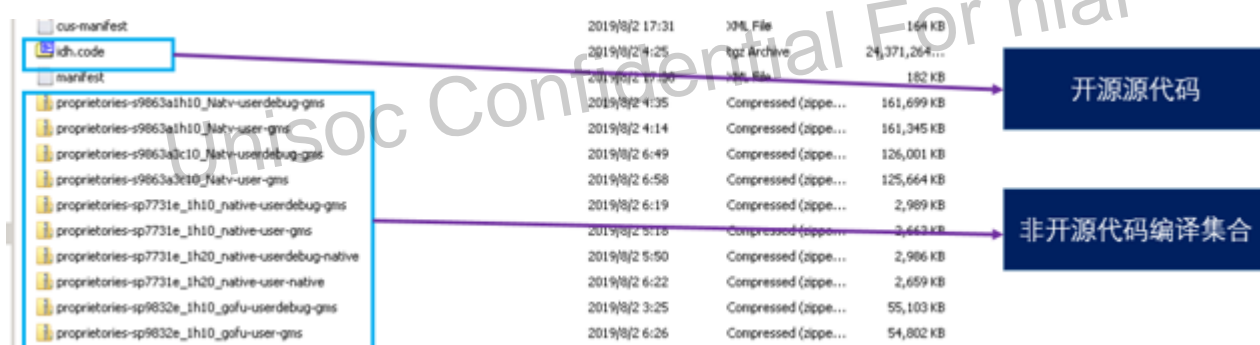
目前 UNISOC 大多采用 git 推送的方式提供 IDH 包。

Android 10.0 UNISOC IDH 包由以下几个部分组成：

- 开源源码部分
如：idh.code.tgz
- 非开源部分
如：proprieties-s9863a1h10_Natv-userdebug-native.zip，其中 s9863a1h10_Natv 为 product name，userdebug 为 build type。

具体可参考图 1-1。

图1-1 IDH 包组成



由于 Android 10.0 上 UNISOC 的编译架构发生变化，支持 BSP 独立编译，故非开源代码，生成文件的压缩包，包括两个部分（BSP&&Android）。如图 1-2 所示。

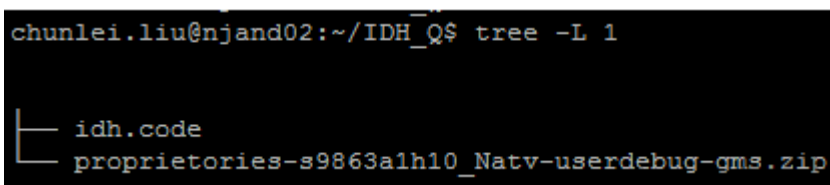
图1-2 非开源代码组成



1.2 IDH 包解压

以 s9863a1h10_Natv 工程为例，获取的 IDH 包结构如图 1-3 所示。

图1-3 IDH 包结构



1.2.2 开源源码解压

将 idh.code.tgz 解压到特定目录，如 D:\IDH_Q\，如图 1-4 所示。

图1-4 开源源码解压

```
chunlei.liu@njand02:~/IDH_Q/idh.code$ tree -L 1
.
├── Android.bp -> build/soong/root.bp
├── art
├── bionic
├── bootable
├── bootstrap.bash -> build/soong/bootstrap.bash
├── bsp
├── build
├── compatibility
├── cts
├── dalvik
├── developers
├── development
├── device
├── external
├── frameworks
├── hardware
├── kernel
├── libcore
├── libnativehelper
├── Makefile
├── out
├── packages
├── pdk
├── platform_testing
├── prebuilts
├── sdk
├── system
├── test
├── toolchain
├── tools
├── vbmeta-gsi.img
└── vendor
```

1.2.3 非开源部分解压

将 proprietaries-s9863a1h10_Natv-userdebug-gms.zip 解压之后的 out 文件夹放到上述目录/IDH_Q/idh.code/out && /IDH_Q/idh.code/bsp，如图 1-5 所示。

图1-5 非开源部分解压

```
chunlei.liu@njand02:~/IDH_Q/proprieties-s9863a1h10_Natv-userdebug-gms$ tree -L 5
.
├── bsp
│   └── out
│       └── s9863a1h10
│           └── dist
│               ├── sml
│               └── trusty
└── out
    └── target
        └── product
            └── s9863a1h10
                ├── obj
                ├── obj_arm
                ├── product
                ├── system
                └── vendor
```

Unisoc Confidential For hiar

2

编译环境准备

2.1 硬件环境要求

虚拟机编译环境硬件要求：

- ROM 至少 250G。
- RAM 至少 16G。否则编译大概率会出现 Exception in thread "main" java.lang.OutOfMemoryError: Java heap space。

2.2 软件环境要求

2.2.1 操作系统要求

建议选择 14.04 版本的 64 位 ubuntu 系统，查看 ubuntu 的具体版本号命令为：

```
lsb_release -a
```

额外需要的软件包主要有：

- python.org 中提供的 Python 2.6 - 2.7。
- gnu.org 中提供的 GNU Make 3.81 - 3.82。
- git-scm.com 中提供的 Git 1.7 或更高版本。

说明

ubuntu10.04~12.0 版本或者更高版本 ubuntu16.04 也支持，但是不同版本依赖的编译支持工具略有差异。

2.2.2 工具包安装

选择 ubuntu 14.04 系统后，可使用如下命令安装编译依赖环境的工具包：

```
$ sudo apt-get install git-core gnupg flex bison gperf build-essential \
zip curl zlib1g-dev gcc-multilib g++-multilib libc6-dev-i386 \
lib32ncurses5-dev x11proto-core-dev libx11-dev lib32z-dev ccache \
libgl1-mesa-dev libxml2-utils xsltproc unzip libssl-dev
```

说明

根据安装的 ubuntu 版本的不同，需要的编译支持工具包不同，完整的工具包在下面的网址可以找到：
<https://source.android.com/setup/build/initializing>。

2.2.3 python 安装

在 IDH 包开源部分的/prebuilts/ python/下有预编译版本的 python，因此无需另行安装。

如果系统已安装其他版本的 python，建议删除，并安装 2.7 或 2.6 版本；查看 python 版本的命令为：

```
python -version
```

📖 说明

可以在 python 官网下载：<https://www.python.org/>。

2.3 常见问题说明

2.3.1 系统库缺失

如：fatal error: openssl/bio.h 等

首先得安装 openssl：

```
sudo apt-get install openssl
```

当 #include<openssl/ssl.h>后编译报错，如果再出现 xxx not found，解决办法为：

```
sudo apt-get install libssl-dev build-essential zlib1g-dev libidn11-dev libidn11
```

2.3.2 ubuntu 版本较低

Google 建议使用 14.04 版本的 ubuntu 系统，当使用低于 14.04 版本的 ubuntu 系统时会报错“GLIBC_2.17/2.18 not found”。

如果坚持使用该版本的 ubuntu 系统，则需要安装对应的依赖库，具体可参考如下命令：

```
sudo dpkg -i libc6_2.17-0ubuntu4_amd64.deb
```

2.3.3 Java 版本错误

如果编译的 Android 版本与 Java 版本不一致，make 将会终止并显示诸如以下消息：

```
*****
You are attempting to build with the incorrect version of java.
Your version is: WRONG_VERSION.
The correct version is: RIGHT_VERSION.
Please follow the machine setup instructions at
  https://source.android.com/source/initializing.html
*****
```

原因是未能安装指定的 JDK。此时需确保已经设置环境变量，将正确的 JDK 附加到路径开头，或者移除有问题的 JDK。

3 编译操作指导

3.1 编译步骤

无论是单编还是全编，都需要完成以下几步：

步骤 1 source build/envsetup.sh，初始化环境变量及命令。

步骤 2 lunch，选择具体的编译工程，指定编译参数。

步骤 3 make XXX，编译特定目标。

---结束

3.1.1 完整编译

3.1.1.1 初始化环境变量及命令

初始化环境变量及命令，搜集编译工程等，如图 3-1 所示。

图3-1 初始化环境变量及命令



3.1.1.2 新增及选择工程

- 可以编译的工程，如图 3-2 所示。选择对应的序号即可编译对应的版本。如 30，编译带 GMS 包的 userdebug 版本。

图3-2 编译工程列表

```

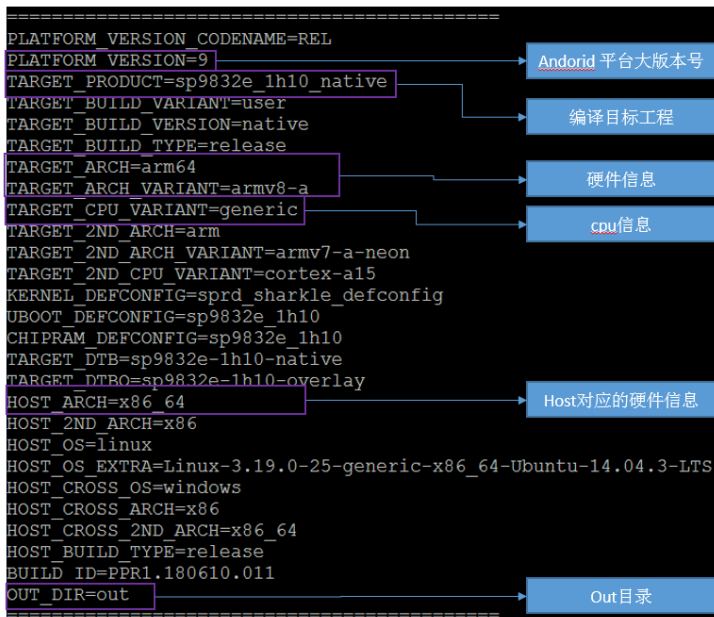
28. s9863a10c10_Natv-userdebug-gms
29. s9863a10c10_Natv-userdebug-native
30. s9863a1h10_Natv-userdebug-gms
31. s9863a1h10_Natv-userdebug-native
32. s9863a1h10_Tsg-userdebug
33. s9863a1h10_go_32b_Natv-userdebug
34. s9863a1h10_go_32b_Natv-userdebug-gms
35. s9863a1h10_go_Natv-userdebug
36. s9863a1h10_nosec-userdebug-native
37. s9863a2h10_Natv-userdebug-gms
38. s9863a2h10_Natv-userdebug-native
39. s9863a3c10_Natv-userdebug-gms
40. s9863a3c10_Natv-userdebug-native
41. s9863a3c10_go_32b_Natv-userdebug-gms
42. s9863a3c10_go_32b_Natv-userdebug-native
43. s9863a3h10_Natv-userdebug-gms
44. s9863a3h10_Natv-userdebug-native
45. s9863a3h10_go_32b_Natv-userdebug-gms
46. s9863a3h10_go_32b_Natv-userdebug-native
47. sp7731e_1h10_native-userdebug-gms
48. sp7731e_1h10_native-userdebug-native
49. sp7731e_1h20_native-userdebug-gms
50. sp7731e_1h20_native-userdebug-native
51. sp9832e_1h10_gofu-userdebug-gms
52. sp9832e_1h10_gofu-userdebug-native
53. uml-userdebug
54. ums512_1h10_Natv-userdebug-native
55. ums512_1h10_nosec-userdebug-native

Which would you like? [aosp_arm-eng]

```

- lunch 选择好编译的工程之后，会输出当前编译工程的一些参数，如图 3-3 所示。

图3-3 编译参数



- Android 10.0 不再使用 add_lunch_combo 来新增编译工程，图 3-4 所示。而是使用宏变量 COMMON_LUNCH_CHOICES 实现。如在 SC9863A 的 AndroidProducts.mk 新增工程，如图 3-5 所示。

图3-4 add_lunch_combo 新增编译工程

```

including device/sprd/sharkl5Pro/ums518_zebu/vendorsetup.sh
device/sprd/sharkl5Pro/ums518_zebu/vendorsetup.sh:17: add_lunch_combo is obsolete. Use COMMON_LUNCH_CHOICES in your AndroidProducts.mk instead.
  
```

图3-5 COMMON_LUNCH_CHOICES 新增编译工程

```
PRODUCT_MAKEFILES += \
    s9863a1h10_go_Natv:$(LOCAL_DIR)/s9863a1h10_go/s9863a1h10_go_Natv.mk \
    s9863a3h10_Natv:$(LOCAL_DIR)/s9863a3h10/s9863a3h10_Natv.mk \
    s9863a1h10_Natv:$(LOCAL_DIR)/s9863a1h10/s9863a1h10_Natv.mk \
    s9863a1h10_Tsg:$(LOCAL_DIR)/s9863a1h10/s9863a1h10_Tsg.mk \
    s9863a1h10_nosec:$(LOCAL_DIR)/s9863a1h10/s9863a1h10_nosec.mk \
    s9863a2h10_Natv:$(LOCAL_DIR)/s9863a2h10/s9863a2h10_Natv.mk \
    s9863a3h10_go_32b_Natv:$(LOCAL_DIR)/s9863a3h10_go/s9863a3h10_go_32b_Natv.mk \
    s9863a1h10_go_32b_Natv:$(LOCAL_DIR)/s9863a1h10_go_32b/s9863a1h10_go_32b_Natv.mk \
    s9863a10c10_Natv:$(LOCAL_DIR)/s9863a10c10/s9863a10c10_Natv.mk \
    s9863a3c10_go_32b_Natv:$(LOCAL_DIR)/s9863a3c10_go/s9863a3c10_go_32b_Natv.mk \
    s9863a3c10_Natv:$(LOCAL_DIR)/s9863a3c10/s9863a3c10_Natv.mk

COMMON_LUNCH_CHOICES := \
    s9863a1h10_go_Natv-userdebug \
    s9863a3h10_Natv-userdebug-native \
    s9863a3h10_Natv-userdebug-gms \
    s9863a1h10_Natv-userdebug-native \
    s9863a1h10_Natv-userdebug-gms \
    s9863a1h10_Tsg-userdebug \
    s9863a1h10_nosec-userdebug-native \
    s9863a2h10_Natv-userdebug-native \
    s9863a2h10_Natv-userdebug-gms \
    s9863a3h10_go_32b_Natv-userdebug-native \
    s9863a3h10_go_32b_Natv-userdebug-gms \
    s9863a1h10_go_32b_Natv-userdebug \
    s9863a1h10_go_32b_Natv-userdebug-gms \
    s9863a10c10_Natv-userdebug-native \
    s9863a10c10_Natv-userdebug-gms \
    s9863a3c10_go_32b_Natv-userdebug-native \
    s9863a3c10_go_32b_Natv-userdebug-gms \
    s9863a3c10_Natv-userdebug-native \
    s9863a3c10_Natv-userdebug-gms
```

3.1.1.3 执行编译及输出 out 目录

- 通常使用 make 命令来进行全编译。一次全新的编译根据编译服务器的性能不同需要几十分钟到几个小时不等。
- 如果编译使用的机器是支持多线程编译的，则可以使用-j 参数来加快编译的速度，-j 之后的数值表示多线程并行编译，建议数值是编译服务器核数的两倍，主要依赖于编译器是否支持多线程并行编译，同时跟 cpu 有关。比如：

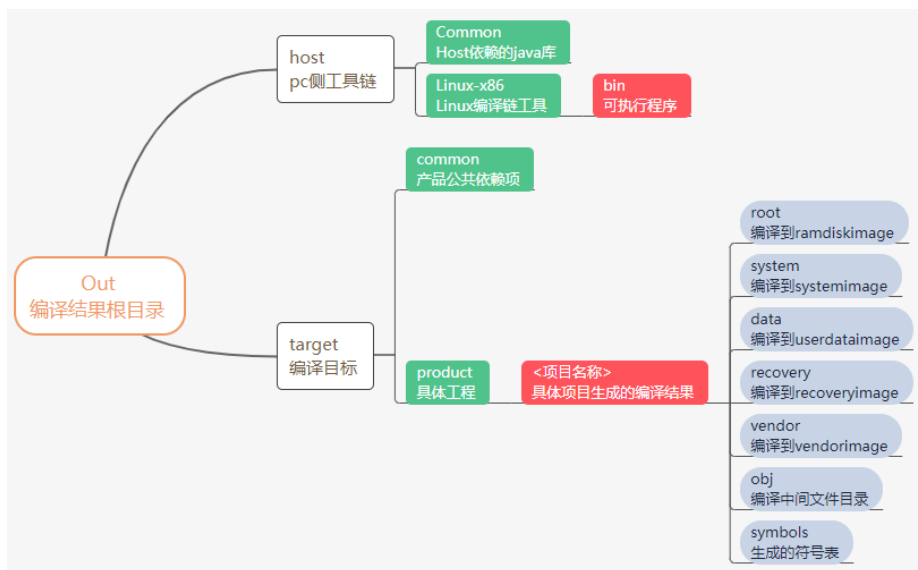
```
make -j8
```

说明

-j 不建议使用较大的数值，防止由于 Makefile 依赖书写不规范导致的编译错误，线程数越大，出现错误的概率越大。

- Android 的编译输出路径为 out，其目录结构如图 3-6 所以。

图3-6 编译输出目录说明



- 其中最重要的目录就是 out/target/product/<项目名>，这里存放着用于下载的所有 bin 和 image 文件，包括 fdl1-sign.bin、fdl2-sign.bin、u-boot-spl-16k-sign.bin、u-boot-sign.bin、boot.img、system.img、userdata.img、recovery.img、cache.img、vendor.img super.img 等。如图 3-7 所示。

Unisoc Confidential For hiar

图3-7 项目名目录

```
chunlei.liu@njand02:~/IDH_Q/idh.code/out/target/product/s9863a1h10$
— boot-debug.img
— boot.img
— cache.img
— dtb.img
— dtbo.img
— odmko.img
— persist.img
— prodnv.img
— product.img
— ramdisk-debug.img
— ramdisk.img
— ramdisk-recovery.img
— recovery.img
— socko.img
— super_empty.img
— super_gsi32.img
— super_gsi64.img
— super.img
— system.img
— userdata.img
— vbmeta-gsi.img
— vbmeta-sign.img
— vbmeta_system.img
— vbmeta_vendor.img
— vendor.img
— ddr_scan-sign.bin
— fdl1-sign.bin
— fdl2-sign.bin
— sml-sign.bin
— tos-sign.bin
— u-boot_autopoweron-sign.bin
— u-boot-sign.bin
```

说明

- 并非所有的下载用文件都是编译生成的，比如 cp 侧的 bin 就是在版本发布中直接提供。
- out/target/product/<项目名>/目录下的 root、system、data、recovery 目录，分别是 boot、system、userdata 和 recovery image 中的直接内容，其中的文件和手机运行后各个对应分区中的内容是一一对应的，当我们通过 mm 指令来编译某个特定的 Android 模块时，更新的也是这些目录中的文件。
- 编译的符号表在很多调试和 bug 解决中是非常重要的，可以在 out/target/product/<项目名>/symbols 目录下找到，我们也可以在 out/target/product/<项目名>/obj 目录下找到同样的内容，不同的是 obj 目录更加具体，不仅仅有符号表，而且有所有 c/c++ java 文件的中间编译结果。同时，kernel 的符号表 vmlinux 也可以在 out/target/product/<项目名>/obj/KERNEL 目录下找到。
- 在 out/target/host/linux-x86/bin 目录下有一些常用的 pc 侧工具，包括 fastboot、mkbootimg、adb 等。

3.1.2 单独编译

在完成一次全编之后，在不改变当前编译项目的前提下，修改代码后可以使用单项的编译来编译对应的部分，加快开发的速度。

3.1.2.1 伪目标镜像单独编译

Android 10.0 上新增了 Super 逻辑分区，引入 Super 镜像，UNISOC 平台默认开启了 Super 逻辑分区，以便 OTA 升级等。通常全编译时，out 目录默认会生成 super 镜像。

- 单独编译 u-boot:

```
make bootloader
```

生成: fdl2-sign.bin、u-boot-sign.bin、u-boot_autopoweron-sign.bin

- 单独编译 fdl1 和 uboot-16k:

```
make chipram
```

生成: fdl1-sign.bin u-boot-spl-16k-sign.bin

- 单独编译 boot image:

```
make bootimage
```

生成: boot.img dt.img kernel ramdisk.img

- 单独编译 boot-debug Image:

```
make bootimage_debug
```

生成 boot-debug.img，用于 XTS 测试。使用 user 版本搭配 boot-debug，可进行 root 操作。

- 单独编译 Super Image:

```
make superimage
```

生成: super.img、super_gsi32.img、super_gsi64.img，GSI 相关 image 用于制作 GSI 版本。

- 单独编译 system image:

```
make systemimage
```

生成: system.img，需要使用 vbmeta_system.img 进行校验。

- 单独编译 vendor image:

```
make vendorimage
```

生成: vendor.img，需要使用 vbmeta_vendor.img 进行校验。

- 单独编译 product image:

```
make productimage
```

生成: product.img

- 单独编译 userdata image:

```
make userdataimage
```

生成: userdata.img

3.1.2.2 特定 module 单独编译

如果单独编译 Settings 应用，可使用如下集中方式编译：

在根目录下：

```
make Settings -j8
```

在根目录下：

```
mmm packages/apps/Settings/
```

工程整编过或通过 make 命令编译后，可在 Settings 的 Android.mk 所在目录执行 mm 或 mma 命令来编译。

说明

- m 在源码树的根目录执行 make，相当于全编译。
- mm 编译当前目录下的模块，不会编译依赖。
- mmm 编译指定目录下的模块，不会编译依赖。
- mma 编译当前目录下的模块，会编译相关依赖。
- mmma 编译指定目录下的模块，会编译相关依赖。

3.1.2.3 BSP 独立编译

Android 10.0 上 BSP 支持独立编译，如果单独编译 Kernel，可使用 BSP 独立编译，在源码目录 bsp 下面进行 source/lunch/make。

具体操作请参见 UNISOC 释放的正式文档《Android 10.0 BSP 独立编译系统使用指南》。

输出的 out 目录结构如图 3-8 所示。

图3-8 BSP 独立编译输出 out 目录结构

```
chunlei.liu@nand02:~/IDH_Q/ldh.code/bsp/out/s9863alh10/dist$
├── chipram
│   ├── ddr_scan.bin
│   ├── ddr_scan.map
│   ├── ddr_scan-sign.bin
│   ├── fd11.axf
│   ├── fd11.bin
│   ├── fd11.map
│   ├── fd11-sign.bin
│   ├── u-boot-spl
│   ├── u-boot-spl-16k.bin
│   ├── u-boot-spl-16k-sign.bin
│   └── u-boot-spl.map
├── kernel
│   ├── dtbo.img
│   ├── Image
│   ├── kernel-uapi-headers.tar.gz
│   ├── sp9863a-1h10.dtb
│   ├── sp9863a-1h10-overlay.dtb
│   ├── System.map
│   ├── usr
│   │   └── include
│   └── vmlinux
├── modules
│   ├── flash_ic_otp8137.ko
│   ├── gator.ko
│   ├── microarray_fp.ko
│   ├── pvrsvkm.ko
│   ├── sprdbt_tty.ko
│   ├── sprd_camera.ko
│   ├── sprd_cpp.ko
│   ├── sprd_flash_drv.ko
│   ├── sprd_fm.ko
│   ├── sprd_sensor.ko
│   ├── sprdwl_ng.ko
│   ├── stmvl5310.ko
│   └── tcs3430.ko
├── sml
│   └── sml.bin
├── trusty
│   └── tos.bin
├── u-boot15
│   ├── fd12.bin
│   ├── fd12-sign.bin
│   ├── System_autopoweron.map
│   ├── System.map
│   ├── u-boot
│   ├── u-boot_autopoweron
│   ├── u-boot_autopoweron.bin
│   ├── u-boot_autopoweron.cfg
│   ├── u-boot_autopoweron-sign.bin
│   ├── u-boot.bin
│   ├── u-boot.cfg
│   └── u-boot-sign.bin
```

3.2 版本编译说明

3.2.1 Userdebug 版本编译

Userdebug 版本编译步骤参照 3.1.1 完整编译。

3.2.2 User 版本编译

一般 lunch 不会列出编译 user 版本的选项，可以输入完整的工程名将其中的 userdebug 替换为 user，如：sp9832e_1h10_native_user-native。

也可以使用 choosecombo 命令来选择具体工程：

步骤 1 选择 Build Type，如图 3-9 所示。

图3-9 编译类型选择

```
chunlei.liu@NJand01:~/AndroidP/idh.code$ choosecombo
Build type choices are:
  1. release
  2. debug
Which would you like? [1] 1
```

步骤 2 输入 Product name，可参考 lunch 的列表输入特定的工程名称，如图 3-10 所示。

图3-10 编译工程选择

```
Which product would you like? [aosp_arm] sp9832e_1h10_oversea
```

步骤 3 选择 Build variant && Build version，如图 3-11 所示。

图3-11 编译版本及编译变量选择

```
Variant choices are:
  1. user
  2. userdebug
  3. eng
Which would you like? [eng] 1

Version choices are:
  1. native
  2. gms
Which would you like? [native] 1
```

----结束

3.2.3 GMS 版本编译

Android 10.0 编译 GMS 版本时，需要将 GMS 包相关的应用放到指定目录：vendor/partner_gms 仓库下。

说明

由于 Google GMS 维护策略的变化，具体使用如下命令获取 GMS 包，UNISOC 释放的 IDH 不再包含 GMS 包。

```
git clone https://partner-android.googlesource.com/platform/vendor/partner_gms -b qt-gms-dev
```

3.2.4 GSI 版本编译

为了更好的兼容 OTA 升级，平台版本默认开启了 Super 的逻辑分区，如果 GSI 采用维护 package 的方式，则需要将 GSI 打包进 super.img。目前平台上已支持将 GSI 打包进 super.img，制作打包 pac 时，可以根据系统的 abi，确定对应的 GSI 版本（super_gsi32.img/ super_gsi64.img）。

- GSI 仓库：/vendor/sprd/partner/aosp-images，如图 3-12 所示。

图3-12 GSI 仓库

```
chunlei.liu@njand02:~/IDH_Q/idh.code/vendor/sprd/partner/aosp-images$ ls
aosp_arm64-user-system.img  aosp_arm-user-system.img
```

- 在执行 make / make superimage 时会自动生成包含的 GSI 的 super 镜像，如图 3-13 所示。

图3-13 包含 GSI 的 super 镜像

```
chunlei.liu@njand02:~/IDH_Q/idh.code/out/target/product/s9863a1h10$ tree -L 1|grep "super"
├── super_empty.img
├── super_gsi32.img
├── super_gsi64.img
└── super.img
```

- 具体修改如下，如图 3-14 所示：

build/tools/releasetools/build_super_image.py

图3-14 GSI 版本编译

```
209 # for ARM 32-bit user GSI
210 image_path = "vendor/sprd/partner/aosp-images/aosp_arm-user-system.img"
211 info_dict["system_image"] = image_path
212 out = os.path.join(os.path.dirname(out), "super_gsi32.img")
213 BuildSuperImageFromDict(info_dict, out)
214 # for ARM 64-bit user GSI
215 image_path = "vendor/sprd/partner/aosp-images/aosp_arm64-user-system.img"
216 info_dict["system_image"] = image_path
217 out = os.path.join(os.path.dirname(out), "super_gsi64.img")
```

关于 GSI 版本维护与调试，请参见 UNISOC 释放的正式文档《Android 10.0 GSI 客户调试指导手册》。

3.2.5 运营商版本编译

部分运营商没有对应的工程，故需要使用如下方式编译，如有对应的运营商工程（CTCC），编译对应的工程即可。

- 单个运营商的编译：

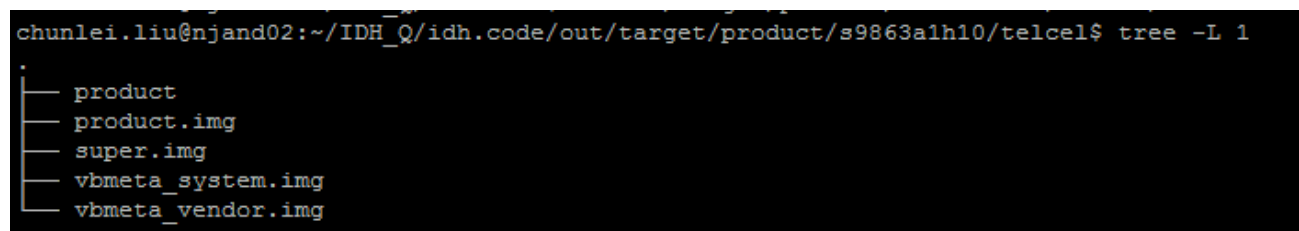
```
make --product_carrier telcel -j8
```

- 多个运营商的编译：

```
make --product_carrier telcel;reliance;cmcc -j8
```

编译完成后可在 out 目录对应的工程下找到对应的运营商目录，针对运行商的编译最终被打包到 product image 中。编译生成文件如图 3-15 所示。

图3-15 运营商编译说明



3.3 Android 10.0 编译新特性

3.3.1 dynamic partitions 编译

dynamic partitions 引入了一种 userspace 分区系统，将 system, vendor 和 product 等 sub-partitions 合并成一个 super partition, sub-partitions 可以共享 super partition 中的预留 OTA 升级空间。

3.3.1.1 dynamic partitions 编译配置

- 开启 dynamic partitions 功能
PRODUCT_USE_DYNAMIC_PARTITIONS := true
- 配置 super partition size 和 group 信息
BOARD_SUPER_PARTITION_SIZE := 4587520000
BOARD_SUPER_PARTITION_GROUPS := group_oem
BOARD_GROUP_OEM_SIZE := 4587520000
BOARD_GROUP_OEM_PARTITION_LIST := system vendor product

3.3.1.2 super partition 编译

super partition 对应的编译目标是 superimage, 执行 make superimage/make, 会生成目标文件

out/target/product/s9863a1h10/super.img。

平台默认将 product.img system.img vendor.img 打包到 super.img。具体如图 3-16 所示。

图3-16 super.img

```
chunlei.liu@njand02:~/IDH_Q/idh.code/out/target/product/s9863a1h10$ tree -L 1 |grep img
├── boot-debug.img
├── boot.img
├── cache.img
├── dtb.img
├── dtbo.img
├── odmko.img
├── persist.img
├── prodnv.img
├── product.img
├── ramdisk-debug.img
├── ramdisk.img
├── ramdisk-recovery.img
├── recovery.img
├── socko.img
├── super_empty.img
├── super_gsi32.img
├── super_gsi64.img
├── super.img
├── system.img
├── userdata.img
├── vbmeta-gsi.img
├── vbmeta-sign.img
├── vbmeta_system.img
├── vbmeta_vendor.img
└── vendor.img
```

3.3.2 Mainline Module 集成

Mainline Module 仓库路径: vendor/sprd/partner/google_mainline/。

Google 释放的 Mainline Module 目录结构如图 3-17 所示。

图3-17 Mainline Module 目录结构

```
chunlei.liu@njand02:~/IDH_Q/idh.code/vendor/sprd/partner/google_mainline$
├── ANGLEPrebuilt
├── CaptivePortalLoginPrebuilt
├── ConscryptPrebuilt
├── DnsResolverPrebuilt
├── DocumentsUiPrebuilt
├── ExtServicesPrebuilt
├── MediaFrameworkPrebuilt
├── MediaSwCodecPrebuilt
├── ModuleMetadataGooglePrebuilt
├── NetworkPermissionConfigPrebuilt
├── NetworkStackPrebuilt
├── PermissionControllerPrebuilt
└── TimeZoneDataPrebuilt
```





关于 Mainline Module 集成的详情请参见 UNISOC 释放的正式文档《Android 10.0 Mainline Module 集成介绍》。

3.3.2.2 Mainline Module 发布策略

Google 如果有新的 Mainline Module 要发布时，会发邮件通知 Partners。在 Partners 邮件群组里的人员会收到邮件。邮件中会附上 Mainline Module 的存放地址 Google Drive folder（Google 云端硬盘）。如图 3-18 所示。

图3-18 Mainline Module 发布

与我共享 > Google-signed Mainline Partner Release Packages > 7/19/2019 FRC 291601500 ▾

名称	所有者
 repo init -u persistent-https://partner-android.googleusercontent.com/platform/manifest -b q-aml-prebuilt-release	Chuljin Ahn
 git tag: mainline_q_291601500	Chuljin Ahn
 Google Play system update - Commands to run module-relevant tests_v0.2.pdf	Chuljin Ahn
 Google Play system update - Module Release Note - 7_19_2019.pdf	Chuljin Ahn

Google 硬盘地址：

https://drive.google.com/corp/drive/folders/1iQWs_J7OcZEgC23wmh1dtcRENKagt2n7

3.3.2.3 Mainline Module 下载命令

目前最新的 Mainline FRC module 在 google 硬盘的存放形式，只是提供了 repo sync 的命令，需要自己下载：

```
$ repo init -u https://partner-android.googleusercontent.com/platform/manifest -b
q-aml-prebuilt-release
$ repo sync -c -j8
5. Mainline Modules Source Code
```

3.4 编译结果验证

当完成特定目标的编译后，验证方式有 ADB push 验证和镜像烧录验证两种。

3.4.1 ADB Push 验证

在电脑上安装 adb 后，可以使用 adb push 命令，将目标 push 到手机对应目录。待手机启动完成后即可验证对应功能。

如验证 settings 应用：

```
adb root
adb remount
adb push xxx/Settings.apk product/priv-app/Settings
adb push xxx/oat/arm64/arm/Settings.vdex product/priv-app/Settings/oat/arm64/arm)
```

```
adb push xxx/oat/arm64/arm)/Settings.odex product /priv-app/Settings/oat/arm64/arm)
adb reboot
```

3.4.2 镜像烧录验证

借助于 ResearchDownload 工具，加载对应的 pac 包，替换编译生成的 img 文件，烧录、重启、验证。

如果烧录 super 镜像，同时需要替换 vbmeta_system 镜像跟 vbmeta_vendor 镜像。

说明

ResearchDownload 工具获取地址：Modem release 的 Tools 压缩包中\Tools\DEBUG_TOOL\ResearchDownload。

镜像烧录验证分为 fastbootd 模式下载镜像和 bootloader 模式下载镜像，说明如表 3-1 所示。

表3-1 fastbootd/bootloader 模式说明

模式	作用
Fastbootd 模式	用于烧录和管理动态分区，adb reboot fastboot 进入。
Bootloader 模式	Android 10.0 原 bootloader 一分为二，adb reboot bootloader 进入。用于管理和烧录物理分区。

3.4.2.2 fastbootd 模式下载镜像

Fastbootd 模式下烧录逻辑分区 system vendor product;

步骤 1 解压后进入 fastboot 命令所在目录，如图 3-19 所示。

图3-19 fastboot 命令

```
D:\platform-tools_r28.0.1-windows\platform-tools>fastboot --version
fastboot version 0.0.0-28209
Installed as D:\platform-tools_r28.0.1-windows\platform-tools\fastboot.exe
```

步骤 2 adb reboot fastboot 进入 fastbootd mode。

步骤 3 烧录动态分区（注意 selinux 权限问题，必须先解锁 bootloader），命令格式如下：

```
fastboot flash <partition name> <filename>
```

步骤 4 烧录 system 镜像，如图 3-20 所示。

图3-20 烧录 system 镜像

```
C:\Users\chunlei.liu>fastboot flash system C:\Users\chunlei.liu\Downloads\signed_signed-aosp_arm64-img-5649316\system.img
Invalid sparse file format at header magic
Resizing 'system' OKAY [ 0.034s]
Sending sparse 'system' 1/3 (523608 KB) OKAY [ 24.448s]
Writing 'system' OKAY [ 13.458s]
Sending sparse 'system' 2/3 (523632 KB) OKAY [ 27.135s]
Writing 'system' OKAY [ 12.974s]
Sending sparse 'system' 3/3 (130296 KB) OKAY [ 6.239s]
Writing 'system' OKAY [ 5.680s]
Finished. Total time: 102.395s
```

----结束

说明

如果烧录逻辑分区，对应的校验镜像也需要重新烧录；如果烧录 system，需要同时烧录 vbmeta_system；如果烧录 vendor，需要同时烧录 vbmeta_vendor；如果烧录 super，则需要同时重新烧录 vbmeta_system、vbmeta_vendor。

3.4.2.3 bootloader 模式下载镜像

bootloader 模式下烧录物理分区：

步骤 1 adb reboot bootloader，进入 bootloader 模式。

步骤 2 烧录物理分区（注意 selinux 权限问题，必须先解锁），Bootloader 模式下烧录 bootimage，如图 3-21 所示。

图3-21 烧录物理分区

```
C:\Users\chunlei.liu>fastboot flash boot D:\8.1\boot_debug\boot-debug.img
< waiting for any device >
Sending 'boot' (35840 KB) OKAY [ 1.605s]
Writing 'boot' OKAY [ 1.143s]
Finished. Total time: 3.062s
```

----结束

3.5 常见问题说明

3.5.1 非开源 apk 重新签名说明

如果需要更换签名文件，则非开源库应用等需要重新签名。平台支持的方式有如下两种。

- 手动签名替换

```
java -Xmx2048m -Djava.library.path="out/host/linux-x86/lib64" -jar out/host/linux-x86/framework/signapk.jar --min-sdk-version 23
build/target/product/security/release/platform.x509.pem
build/target/product/security/release/platform.pk8 ~/源.apk ~/签名后的.apk
```

- 预编译方式自动签名

```
vendor/sprd/proprieties-source/proprieties-prebuilt/Android.mk
```

新增需要预编译签名的 module 配置。如图 3-22 所示。

图3-22 预编译方式自动签名

```
#####  
#customized properties  
#####  
#prebuild for USCPHOTOSProvider  
include $(CLEAR_VARS)  
LOCAL_MODULE := USCPHOTOSProvider  
LOCAL_MODULE_TAGS := optional  
LOCAL_MODULE_CLASS := APPS  
LOCAL_CERTIFICATE := media  
LOCAL_DEX_PREOPT := false  
LOCAL_PRIVILEGED_MODULE := true  
MODULE_DIR := ../platform/packages/providers/USCPHOTOSProvider/prebuilt  
$(call proprietary-prebuild,$(MODULE_DIR))  
  
MY_TARGET_MODULE :=  
MY_MODULE_SOURCE_DIR :=
```

3.5.2 编译报错说明

正常编译时，如果看到编译报错信息如 No command 'mmm' found 等信息时，首先需要确认是否有发起 source build/envsetup.sh，要想编译首先必须执行上述步骤。

3.5.3 编译选项说明

平台默认开启 LOCAL_DEX_PREOPT，打包 package 时会生成 apk 及 dex 文件，验证的时候需要同时 push。

3.5.4 库的依赖关系说明

使用如下命令查看对应的 so 文件，然后将编译环境下的这些 so 拷贝到目录/lib 下。

```
$ readelf -d helloworld | grep NEEDED  
0x00000001 (NEEDED)             Shared library: [libstdc++.so.6]  
0x00000001 (NEEDED)             Shared library: [libm.so.6]  
0x00000001 (NEEDED)             Shared library: [libgcc_s.so.1]  
0x00000001 (NEEDED)             Shared library: [libc.so.6]
```

4 OTA 包编译及差分包制作说明

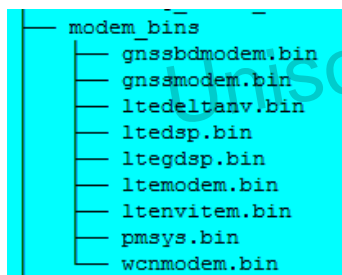
4.1 相关文件准备

- 确认 modem 包 bin 相关文件和编译脚本文件的名称对应关系，以 sp9832e_1h10/AndroidBoard.mk 为例，需要将 release 的文件重命名为 board 里面的对应名称。

gnssbdmodem.bin	——	gnssbdmodem.bin
gnssmodem.bin	——	gnssmodem.bin
RM_sharkle_cm4.bin	——	wcnmodem.bin
SC9600_sharkle_pubcp_Smart_Phone_Uncache_modem.dat	——	ltemodem.bin
SHARKLE2_DM_DSP.bin	——	ltegdsp.bin
sharkle_cm4.bin	——	pmsys.bin
SharkLE_LTEA_DSP_evs_on.bin	——	ltedsp.bin
sharkle_pubcp_Smart_Phone_deltanv.bin	——	ltedeltanv.bin
sharkel_pubcp_Smart_Phone_nvitem.bin	——	ltenvitem.bin

- 确认 cp 相关的 bin 文件已经拷贝到 device/sprd/sharkle/ sp9832e_1h10_test/modem_bins/，如图 4-1 所示。

图4-1 bin 文件



说明

lte 平台包括 ltemodem.bin, ltegdsp.bin 等相关 bin，非 lte 平台命名是不相同的，需要仔细确认。

由于平台众多，modem bins 的多少和名字也都有一些细微区别，如果不确定如何改名字，可以联系展锐 FAE，展锐 FAE 会给出指导。

modem 改名映射规则举例如下：

- SC9832E

SC9600_sharkle_pubcp_Smart_Phone_modem.dat	->	ltemodem.bin
SharkLE_LTEA_DSP.bin	->	ltedsp.bin
SHARKLE2_DM_DSP.bin	->	ltegdsp.bin
PM_sharkle_cm4.bin	->	wcnmodem.bin
sharkle_cm4.bin	->	pmsys.bin
sharkle_pubcp_Smart_Phone_nvitem.bin	->	ltenvitem.bin
sharkle_pubcp_Smart_Phone_deltanv.bin	->	ltedeltanv.bin


```
gnssbdmodem.bin -> gnssbdmodem.bin
```

```
gnssmodem.bin -> gnssmodem.bin
```

● SC9863A

```
SC9600_sharkl3_pubcp_modem.dat -> ltemodem.bin
```

```
sharkl3_pubcp_LTEA_DSP.bin -> ltedsp.bin
```

```
sharkl3_pubcp_DM_DSP.bin -> ltegdsp.bin
```

```
sharkl3_cm4.bin -> pmsys.bin
```

```
gnssbdmodem.bin -> gnssbdmodem.bin
```

```
gnssmodem.bin -> gnssmodem.bin
```

```
PM_sharkl3_cm4.bin -> wcnmodem.bin
```

```
sharkl3_pubcp_nvitem.bin -> ltenvitem.bin
```

```
sharkl3_pubcp_Smart_Phone_deltanv.bin -> ltedeltanv.bin
```

● SC7731E

```
SC9600_pike2_pubcp_uncache_modem.dat --> wmodem.bin
```

```
PIKE2_DM_DSP.bin --> wgdsp.bin
```

```
pike2_pubcp_v1_nvitem.bin --> wnvitem.bin
```

```
pike2_cm4.bin --> pmsys.bin
```

```
PM_pike2_cm4.bin --> wcnmodem.bin
```

```
gnssmodem.bin --> gnssmodem.bin
```

```
gnssbdmodem.bin --> gnssbdmodem.bin
```

4.2 OTA 包编译

注意编译 OTA 之前需要完成一次全编译。

编译命令：

```
make otapackage
```

使用上述命令生成 OTA 包，命名格式：<产品名>-ota-<序列号>.username.zip。如图 4-2 所示。

图4-2 OTA 包

```
chunlei.liu@njand02:~/IDH_Q/idh.code/out/target/product/s9863a1h10$ tree -L 1|grep ota
├── ota_metadata
└── s9863a1h10_Natv-ota-eng.chunlei.liu.zip
```

4.3 OTA 差分包制作

生成新旧两个版本的 OTA 包之后，可以制作对应的升级差分包，命令为：

```
./build/make/tools/releasetools/ota_from_target_files
```

```
-i ota_old.zip ota_new.zip update.zip
```

ota_old.zip 与 ota_new.zip 分别是升级前和要升级到版本的 OTA 包。

📖 说明

为了以后在版本升级时可以使用差分升级，同时保留此版本对应的 target 文件。路径为：

out/target/product/spXXXX/obj/PACKAGING/target_files_intermediates/*-target_files-*.zip

5

pac 制作及下载说明

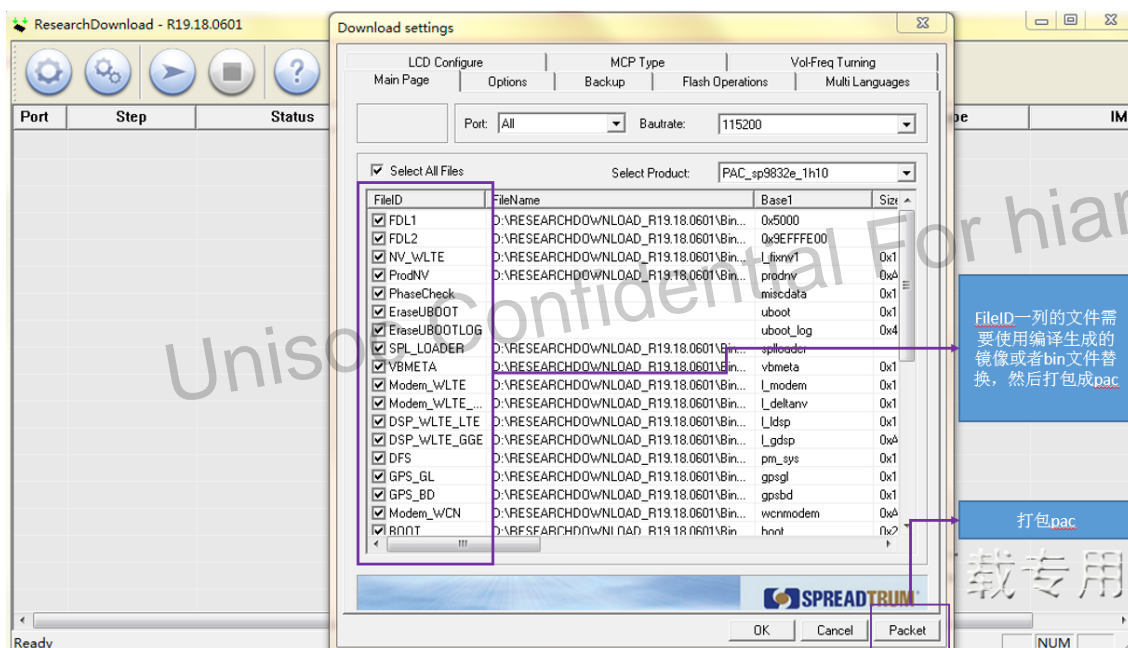
5.1 pac 制作

pac 制作有两种方式，借助于工具制作和脚本打包制作。

5.1.1 工具制作

工具制作方式如图 5-1 所示。

图5-1 工具制作 pac



不足：需要先 load 一个 pac，然后替换，而且替换前后的生成文件大小差异不能太大，太大可能因分配的分区分大小 size 不匹配，导致烧录失败。

当分区 size 大小不匹配时，需要修改配置文件，修改分区大小。如：

RESEARCHDOWNLOAD_R19.18.0601\Bin\ImageFiles\DownloadFiles460679189: s9832e1h10.xml 文件，找到对应的分区大小设置的地方，修改 size 大小。如图 5-2 所示。

图5-2 修改 size 大小

```
<Partition id="pm_sys" size="1"/>
<Partition id="boot" size="35"/>
<Partition id="system" size="2150"/>
<Partition id="cache" size="150"/>
<Partition id="vendor" size="400"/>
<Partition id="vbmeta" size="1"/>
<Partition id="vbmeta_bak" size="1"/>
<Partition id="userdata" size="0xFFFFFFFF"/>
```

5.1.2 脚本制作

UNISOC 的打包脚本由版本编译与集成团队统一维护，如果没有打包脚本，可以向展锐 FAE 申请获取。

- 脚本需要在./vendor/sprd/release/IDH/Script 目录下执行，并加执行权限。
- 脚本执行方式：

```
./build_pac.sh
```

- a "参数 a" -b "参数 b" -c "参数 c"; -a & -b 是必填项，-c 可选。
- 参数-a 是./vendor/sprd/release/IDH 目录下面 "sp*" 目录名称。
- 参数-c 只对编译有影响，对 pac 没有影响。

举例 sp9832e_1h10_reliance-user-native，调用方式：

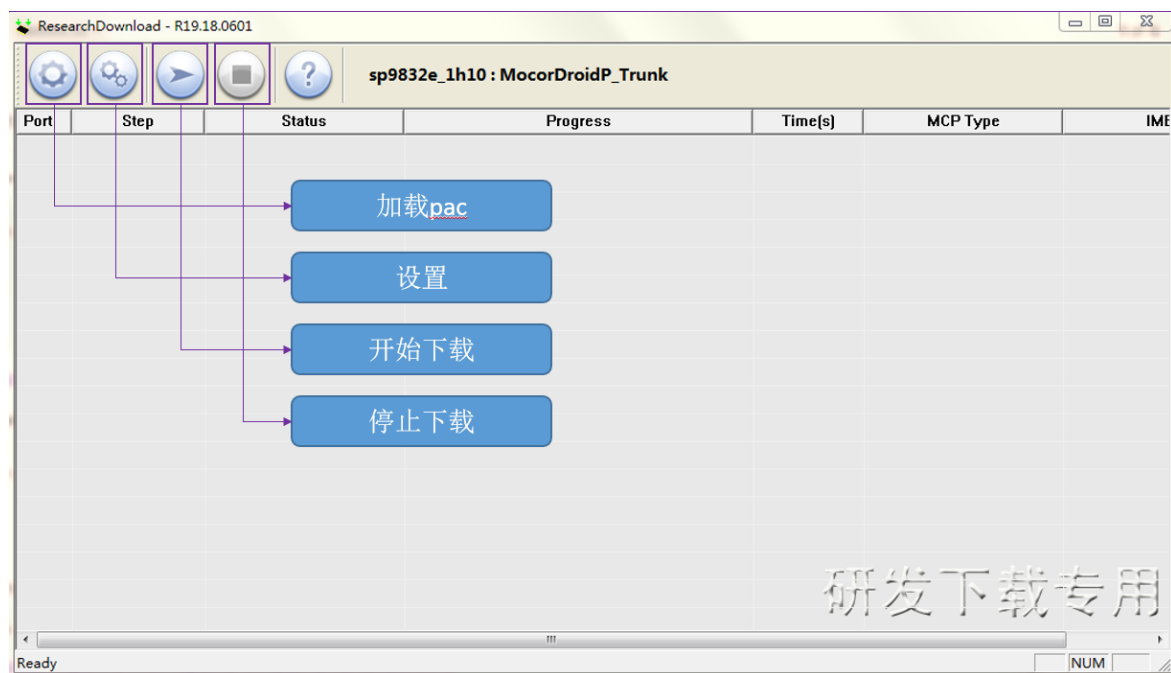
```
./build_pac.sh -a sp9832e_1h10_reliance-user-native -b BUILD -c ***
```

- 参数-b BUILD|build:编译；PAC|pac:打 pac 包。
 - 参数-c 是编译运营商工程，如果有多个用英文","隔开。
- 生成的 pac 包在“参数-a”目录内除 out 目录外的子目录下。

5.2 pac 下载说明

ResearchDownload 下载工具使用说明如图 5-3 所示。

图5-3 ResearchDownload 下载工具使用说明



工具下载的具体步骤：

步骤 1 在 PC 端安装手机驱动。

步骤 2 在 ResearchDownload 工具上加载 pac，并点击开始下载按钮。

步骤 3 按手机音量下键，同时将 PC 端的 USB 线连接手机，并插入电池。

步骤 4 完成下载后插拔电池，并重启手机。

----结束

6

Android 10.0 编译规则的新变化

Android 10.0 上谷歌对编译系统做了较大的升级修改。对照 build/make/Changes.md 结合 bring up 过程中遇到以及解决的各种编译问题，现整理出此章节，以期能够全面描述 Android 10.0 上编译系统的最新要求，以便研发过程中遇到编译问题时能够得以参考。

6.1 新增的规则

6.1.1 限制 HOST TOOLS 的使用

编译系统开始限制 host 操作系统中各种工具在编译过程中的使用。这将有助于我们在不同的机器上编译出同样的目标文件，进而尽可能消除隐藏的问题。

故需将系统默认 PATH 替换为指定的目录列表，任何不在这些目录列表中的工具在使用时将会导致错误，除非配置为允许使用（随着时间的推移，将会越来越严格）。

默认配置位于 build/soong/ui/build/paths/config.go，其中包含了大多数编译过程中所使用的通用工具。

Android 10.0 之前，编译系统从环境变量 PATH 中查找并定位工具所在的位置，Android 10.0 上编译系统编译时，临时修改了环境变量 PATH，因此导致编译时无法查找到部分工具，从而导致了编译错误。主要涉及到工具有：

- prebuilts/build-tools/linux-x86/bin/目录下的工具，如：repo、printenv、pkg-config、ccache、perl 等。
- prebuilts/gcc/linux-x86/host/x86_64-linux-glibc2.15-4.8/目录下的 host gcc。

上述这些工具 Android 10.0 之前直接从 HOST 操作系统安装目录查找，现在移动到 prebuilts 下的相关目录，与此同时，编译系统也需要做对应的修改，以便支持这些增加到 PATH 中的目录列表。

另外，Android 10.0 不建议客户再使用 gcc，目前推荐使用 clang。

6.1.2 调用其他编译系统时的.PHONY 规则

Android 编译过程中调用其它编译系统，例如 bootloader、kernel 等时，通常将编译目标标识为.PHONY，这是因为 Android 编译系统增量编译时，缺少足够的依赖来判断是否需要重新构建目标，所以索性标识为.PHONY 从而每次都重新构建。这种情形，Google 建议将其剥离出 Android 编译系统，以 prebuilts 的方式整合进来，从而改善 Android 增量编译时的速度和可靠性。

如果不想把相关模块剥离出 Android 编译系统，而又想要更快的增量编译速度，详尽的描述依赖关系是一个比.PHONY 更好的方案。

6.1.3 规范.PHONY 规则的使用

为了改善 Android 增量编译的速度和可靠性，Google 规范了.PHONY 规则的一些用法：

.PHONY 标识的目标通常作为实际目标的 `shortcuts`，以便为目标提供一个更加友好的名字，但是，这同时也意味着该目标始终被编译系统认为是 `dirty` 的，从而每次编译时都进行重新构建。对于 `aliases` 或者单次用户请求操作来说，这不是什么大问题，但如果一个真实的编译步骤依赖于 .PHONY 目标，对于一次小型的编译来说，这可能变得非常昂贵。

- .PHONY 标识的目标，其名称不得包含 `/`，例如：

```
test: foo/bar foo/baz
foo/bar: .KATI_IMPLICIT_OUTPUTS := foo/baz
foo/bar:
    @echo "END"
.PHONY: test foo/bar
```

Makefile:4: *** PHONY target "foo/bar" looks like a real file (contains a "/")

- 目标是一个实际存在的文件，但位于 `out` 目录之外。Google 要求编译系统的所有输出都在 `out` 目录中，否则 `m clean` 将无法彻底清除到之前编译的中间文件，进而可能对之后的编译造成不可预知的影响。

...mk:42: warning: writing to readonly directory: "kernel-modules"

- `out/foo` 看起来像一个真实的文件，因此，它不可以依赖于其它伪目标，例如：

```
.PHONY: test
test: out/foo
out/foo: bar
```

Makefile:4: *** real file "out/foo" depends on PHONY target "bar"

6.2 变更的规则

6.2.1 规范模块的命名规则

模块名称命名时只能使用 ``a-z'``, ``A-Z'``, ``0-9'`` 以及特殊符号 ``_+ -=, @ ~``，当前适用于 `LOCAL_PACKAGE_NAME`、`LOCAL_MODULE`、`LOCAL_MODULE_SUFFIX` 以及 `LOCAL_MODULE_STEM`。

需要注意的是，模块命名时不允许使用 `/`。以前 Android 编译系统在很多地方使用带路径的模块名称，这种情况下，建议采用 ``LOCAL_MODULE_RELATIVE_PATH`` 或者 ``LOCAL_MODULE_PATH`` 而不是直接把路径前缀插入到模块名之前，如果这会导致模块重名的文件，请注意使用唯一的模块名或者添加 `LOCAL_MODULE_STEM` 来加以区分。例如：

旧的写法：

```
include $(CLEAR_VARS)
LOCAL_MODULE := ver1/code.bin
LOCAL_MODULE_PATH := $(TARGET_OUT_ETC)/firmware
...
include $(BUILD_PREBUILT)

include $(CLEAR_VARS)
LOCAL_MODULE := ver2/code.bin
LOCAL_MODULE_PATH := $(TARGET_OUT_ETC)/firmware
...
include $(BUILD_PREBUILT)
```

新的写法:

```
include $(CLEAR_VARS)
LOCAL_MODULE := ver1_code.bin
LOCAL_MODULE_STEM := code.bin
LOCAL_MODULE_PATH := $(TARGET_OUT_VENDOR)/firmware/ver1
...
include $(BUILD_PREBUILT)

include $(CLEAR_VARS)
LOCAL_MODULE := ver2_code.bin
LOCAL_MODULE_STEM := code.bin
LOCAL_MODULE_PATH := $(TARGET_OUT_VENDOR)/firmware/ver2
...
include $(BUILD_PREBUILT)
```

6.2.2 `DIST_DIR`, `dist_goal` 变更为 `dist-for-goals`

`DIST_DIR` 和 `dist_goal` 目前已经不用了, 使用 `dist-for-goals` 替代。它接受一个伪目标, 以及一个待拷贝到 `$DIST_DIR` 目录的文件列表。 `$(call dist-for-goals,foo,bar/baz)`

在执行 `m foo dist` 时, 文件 `bar/baz` 将被拷贝到 `$DIST_DIR/baz`。如果需要重命名, `$(call dist-for-goals,foo,bar/baz:logs/foo.log)` 在执行 `m foo dist` 时, 文件 `bar/baz` 将被拷贝到 `$DIST_DIR/logs/foo.log`。

6.2.3 拆分 `PRODUCT_PACKAGES`

以前添加一个同时支持 `host` 和 `target` 的模块到 `PRODUCT_PACKAGES` 会导致两种类型都被编译和安装。很多情况下, 可能只希望默认编译/安装一种类型。因此, Android 10.0 中将 `PRODUCT_PACKAGES` 做了一定的拆分, 即 `PRODUCT_HOST_PACKAGES` 只影响 `host` 端, 而 `PRODUCT_PACKAGES` 只影响 `target` 端。

`PRODUCT_HOST_PACKAGES` 不支持 `‘_ENG/_DEBUG’` 选项, 因为这是 `target` 的属性而不是 `Host` 的; 不支持 `LOCAL_MODULE_OVERRIDES`; 要求所列出的 `host` 模块必须存在, 除非设置了 `ALLOW_MISSING_DEPENDENCIES`。

目前这仍由用户决定是否迁移, 因此, `PRODUCT_PACKAGE` 仍然可以使用。

6.3 弃用的规则

6.3.1 不建议使用隐式 `make` 规则

所谓隐式 `make` 规则是类似如下这样的写法:

```
$(TARGET_OUT_SHARED_LIBRARIES)/%_vendor.so: $(TARGET_OUT_SHARED_LIBRARIES)/%.so
...

%.o: %.foo
...
```

这些隐式 `make` 规则影响范围极大, 甚至包括完全不相干的模块, 所以它们现在已经过时(`obsolete`)了。建议使用静态模式规则, 可以实现与隐式规则一样的效果, 但不至于影响到无关的模块:

```
libs := $(foreach lib,libfoo libbar,$(TARGET_OUT_SHARED_LIBRARIES)/$(lib)_vendor.so)
```



```
$(libs): %_vendor.so: %.so
...

files := $(wildcard $(LOCAL_PATH)/*.foo)
gen := $(patsubst $(LOCAL_PATH)/%.foo,$(intermediates)/%.o,$(files))
$(gen): %.o : %.foo
...
```

6.3.2 禁用 make 的 export 和 unexport 关键字

Android 10.0 编译系统中，`export` 和 `unexport` 已被禁用，根据使用场景将触发错误或者警告。具体来说，在编译的早期阶段，即产品配置以及读取 `BoardConfig.mk` 时，将会弹出警告（将来会变成错误）。而在解析 `Android.mk` 以及后续处理时，将会弹出错误。

Google 将持续限制 Android 核心编译系统对环境变量的依赖，因此不建议通过修改环境变量来影响或者控制编译结果。如果实在需要使用 `export`，建议采用如下方式（尽可能减小影响范围）：

```
$(intermediates)/generated_output.img:
rm -rf $@
export MY_ENV_A="$(MY_A)"; make ...

envsh := $(intermediates)/env.sh
$(envsh):
    rm -rf $@
    echo 'export MY_ENV_A="$(MY_A)'" >$@
    echo 'export MY_ENV_B="$(MY_B)'" >>$@
$(intermediates)/generated_output.img: PRIVATE_ENV := $(envsh)
$(intermediates)/generated_output.img: $(envsh) a/b/c/package.sh
    rm -rf $@
    source $(PRIVATE_ENV); make ...
    source $(PRIVATE_ENV); a/b/c/package.sh ...
```

6.3.3 从 Android.mk 中移除 BUILD_NUMBER

`Android.mk` 文件不得直接使用 `BUILD_NUMBER`，因为每次 `BUILD_NUMBER` 发生变化时都需要重新读取。如果可以的话，最好直接去除，确实需要使用的話，可用 `BUILD_NUMBER_FROM_FILE` 替代。例如：

```
$(LOCAL_BUILT_MODULE):
mytool --build_number $(BUILD_NUMBER_FROM_FILE)
```

`-o $@BUILD_NUMBER_FROM_FILE` 的影响范围限制在子 shell 内。

6.3.4 弃用 USER

随着 Android 编译逐步沙盒化，`USER` 很快将意味着 `nobody`。在大多数情况下，没必要知道谁在执行编译，如果需要，可以使用 `make` 变量 `BUILD_USERNAME`。类似的，`hostname` 工具在 Android 编译时将返回同样的值，真正的值可以通过 `BUILD_HOSTNAME` 获取。

6.3.5 弃用 LOCAL_MODULE_TAGS := eng debug

`LOCAL_MODULE_TAGS` 的值 `eng` 和 `debug` 正在废弃中。他们允许模块自行决定是否安装在 `eng/userdebug` 类型的编译中。这与产品指定安装哪些模块的设定产生了冲突，使得精简产品配置变得困难。

使用 `PRODUCT_PACKAGES_ENG` 以及 `PRODUCT_PACKAGES_DEBUG` 可以达到同样的目的。

6.3.6 Android.mk 不再支持 windows 交叉编译

为 windows 编译的模块，目前仅支持在 `Android.bp` 中定义。

6.3.7 弃用 `*.c.arm` / *.cpp.arm``

以前在 `Android.mk` 文件中可以通过为原文件添加 `.arm` 后缀来指定 `LOCAL_ARM_MODE`。Soong 不支持这种不常见的行为，因而优先考虑使用 `arm` 而不是 `thumb` 来编译整个库。目前，`Android.mk` 文件中也进行类似要求。

Unisoc Confidential For hiar

7

参考文档

1. 《Android 10.0 BSP 独立编译系统使用指南》
2. 《Android 10.0 GSI 客户调试指导手册》
3. 《Android 10.0 Mainline Module 集成介绍》

Unisoc Confidential For hiar