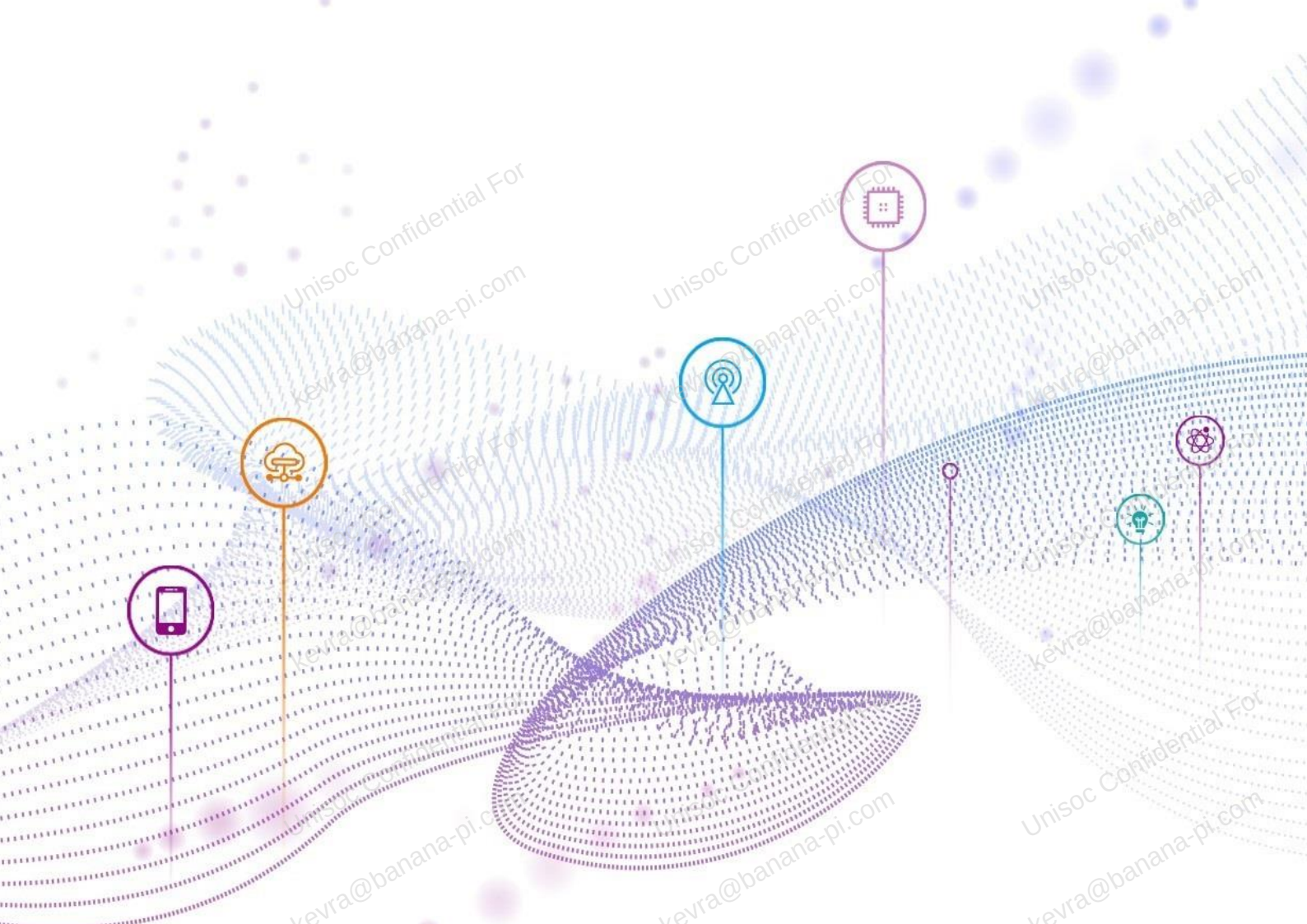




Yocto GLP3.0 软件开发指南

文档版本
发布日期

V1.4
2026-05-20



紫光展锐（上海）科技股份有限公司

版权所有 © 紫光展锐（上海）科技股份有限公司。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）科技股份有限公司（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责任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。

商标声明

紫光展锐、**UNISOC**、、展讯、Spreadtrum、SPRD、锐迪科、RDA 及其他紫光展锐的商标均为紫光展锐（上海）科技股份有限公司及/或其子公司、关联公司所有。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

免责声明

本文档可能包含第三方内容，包括但不限于第三方信息、软件、组件、数据等。紫光展锐不控制且不对第三方内容承担任何责任，包括但不限于准确性、兼容性、可靠性、可用性、合法性、适当性、性能、不侵权、更新状态等，除非本文档另有明确说明。在本文档中提及或引用任何第三方内容不代表紫光展锐对第三方内容的认可、承诺或保证。

用户有义务结合自身情况，检查上述第三方内容的可用性。若需要第三方许可，应通过合法途径获取第三方许可，除非本文档另有明确说明。

紫光展锐（上海）科技股份有限公司



前言

概述

本文档介绍 Yocto GLP3.0 软件开发，内容涵盖 GLP3.0 项目介绍、系统要求、源码目录结构、编译、新增分区、编译后新增目录、只读文件系统配置及使用、下载和开机等。

读者对象

本文档主要适用于 Yocto GLP3.0 项目的软件开发人员。

适用平台

OS 版本	Yocto
Kernel 版本	Kernel 5.15、Kernel 6.1、Kernel 6.6

缩略语

缩略语	英文全称	中文解释
AVL	Approved Vendor List	认可的供应商清单
QVL	Qualified Vendor List	合格供应商列表

符号约定

在本文中可能出现下列符号，每种符号的说明如下。

符号	说明
 说明	用于突出重要或关键信息、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。
 注意	用于突出容易出错的操作。 “注意”不是安全警示信息，不涉及人身、设备及环境伤害。

符号	说明
 警告	用于可能无法恢复的失误操作。 “警告”不是危险警示信息，不涉及人身及环境伤害。

变更信息

文档版本	发布日期	修改说明
V1.4	2026-05-20	新增 4.13 dts/kernel cfg 相关说明。
V1.3	2026-03-16	- 新增 4.11 新建 product。 - 补充 UIS7885 GLP3.0 内容。
V1.2	2025-08-06	更新 9 附录 ，补充 UIS8510 内容。
V1.1	2025-07-07	- 基于最新操作方法，更新 4 编译 相关内容。新建 board 部分，新增 u-boot22、kernel6.1 的介绍。 - 更新 9 附录。
V1.0	2024-08-21	第一次正式发布

关键字

GLP3.0、编译、目录结构、分区、下载、开机

目 录

1 GLP3.0 项目介绍.....	1
2 系统要求.....	2
2.1 操作系统要求.....	2
2.2 工具包安装.....	2
2.3 软件版本要求.....	3
3 源码目录结构.....	4
4 编译.....	5
4.1 编译前准备.....	5
4.2 编译配置信息.....	5
4.3 编译整个工程.....	6
4.3.1 命令行方式配置环境.....	6
4.3.2 图形界面方式配置环境.....	6
4.4 编译单个模块.....	9
4.5 编译 MACHINE 的 Kernel/U-Boot/Chipram.....	9
4.6 生成 pac 包.....	11
4.7 编译结果.....	11
4.8 手动签名方法.....	12
4.9 添加模块.....	13
4.9.1 bb 文件.....	13
4.9.2 添加模块到 IMAGE.....	19
4.10 添加 ko 模块.....	23
4.10.1 添加内部 Kernel ko 到 rootfs.....	23
4.10.2 添加外部 Kernel ko 到 rootfs.....	23
4.11 新建 product.....	27
4.12 新建 board.....	28
4.12.1 新建 Chipram.....	28
4.12.2 新建 U-Boot.....	29
4.12.3 新建 MACHINE.....	32
4.12.4 新建 Kernel.....	36
4.12.5 修改 source/unisoc 目录下 base board 信息.....	39
4.13 dts/kernel cfg 相关说明.....	39
4.13.1 添加对多 dts 的支持.....	39
4.13.2 dts 文件选择.....	41
4.13.3 kernel cfg 说明.....	41
4.14 其他操作及说明.....	42

4.14.1 加快编译速度	42
4.14.2 查看版本号	42
4.14.3 版本区分	43
4.14.4 打开/关闭 MACHINE 中 Kernel 的某个选项	43
4.14.5 修改分区大小	44
4.15 常见问题处理	45
4.15.1 新基线版本合入出错	45
4.15.2 非联网环境编译出现 fetch 错误	45
4.15.3 policy.xml 文件报错	46
4.15.4 pac 包编译失败	47
5 新增分区	49
5.1 构建新分区	49
5.2 编译 pac 包时完成新增分区	51
5.3 自动挂载分区	53
5.4 新分区增加到 researchdownload 工具	56
6 编译后新增目录	57
7 只读文件系统配置及使用	58
7.1 配置方法	58
7.2 使用注意事项	58
8 下载和开机	59
8.1 下载软件流程	59
8.2 下载工具使用	59
8.2.1 工具介绍	59
8.2.2 准备 pac 包	61
8.2.3 下载 pac 包	61
8.3 下载常见问题	63
8.4 开机方式	64
8.5 开机启动流程	64
8.6 Log 抓取和分析	65
8.6.1 工具侧 Log 抓取	65
8.6.2 终端 AP 侧串口 Log 抓取	65
8.6.3 开机 Log 抓取	66
8.6.4 Log 分析	66
9 附录	67
10 参考资料	69

图目录

图 1-1 Yocto 整体结构 ^[1]	1
图 3-1 源码目录结构	4
图 4-1 MACHINE 信息	7
图 4-2 DISTRO 信息	7
图 4-3 选择 product type	8
图 4-4 选择 engineering version	8
图 6-1 编译后工程目录	57
图 8-1 下载软件流程	59
图 8-2 ResearchDownload 主界面	60
图 8-3 ResearchDownload 下载设置	62
图 8-4 下载成功	62
图 8-5 下载失败	63
图 8-6 ResearchDownload 下载失败	63
图 8-7 开机启动流程	64

表目录

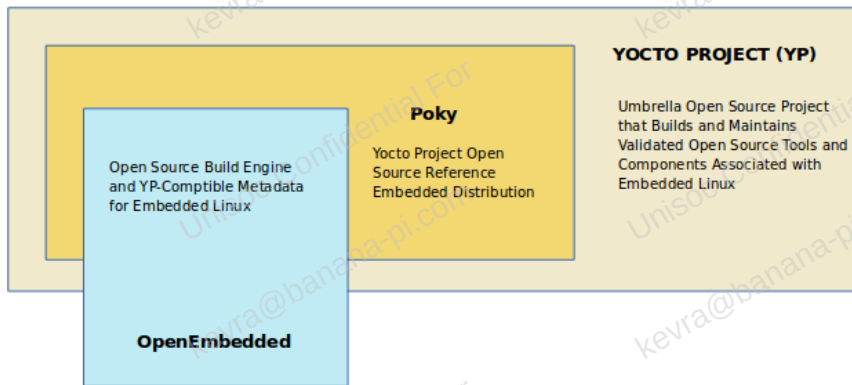
表 4-1 bb 文件基本信息	15
表 4-2 License 相关参数说明	15
表 4-3 依赖相关参数说明	16
表 4-4 打包相关配置信息	19
表 8-1 ResearchDownload 工具栏说明	60
表 8-2 ResearchDownload 状态窗口说明	61
表 9-1 ums9620-2h10 信息	67
表 9-2 ums9620-1h10 信息	67
表 9-3 ums9620-1h10-k515 信息	67
表 9-4 uis8520-1h10-nand 信息	67
表 9-5 uis8910dm-2h10-nand 信息	67
表 9-6 uis8360-1h10-nand 信息	68
表 9-7 uis8510-1h10-emmc 信息	68
表 9-8 uis7885-2h10 信息	68

1 GLP3.0 项目介绍

展锐 GLP3.0 (Generic Linux Platform, 通用 Linux 平台) 是展锐基于 Yocto Project 深度定制的嵌入式 Linux 平台, 为开发多个设备的应用程序提供本地 API, 源码目录参考了 Yocto Project。

Yocto Project 是一个开源协作项目, 为开发人员创建基于 Linux 的自定义系统和嵌入式产品提供了工具和空间。此处 Linux 系统非 Linux 内核, 而是包括了 Bootloader、Linux 内核以及用户空间程序组件。Yocto 官网对 Yocto 的整体描述如图 1-1 所示。

图1-1 Yocto 整体结构^[1]



主要组成部分:

- **Metadata:** 元数据集, 即发行版内各基本元素的描述与来源。
 - **Recipe:** Recipe 是一个以.bb 为扩展名的文件, 是最常见的元数据形式。Recipe 中标明了编译目标的源码、要打哪些 patch (git patch)、编译配置以及编译依赖等信息。
 - **Layer:** Recipe 的集合。Layer 可以自定义构建合并相关的元数据, 并隔离多个体系结构构建的信息。展锐产品中, 对应的目录名称是 meta-unisoc。
 - **Configuration Files:** 包含全局变量的定义、用户定义的变量和硬件配置信息的文件, 作用是告诉构建系统要构建什么, 并放入映像中以支持特定平台。
- **Package:** 用于创建最终映像的构建系统的输出。
- **BitBake:** Yocto 项目的核心组件, 由 OpenEmbedded Core 发布用来构建 Linux, 与 Yocto 分开维护。BitBake 是一个执行工具, 用来执行用 Bash 或者 Python 编写的任务, 同时能解析任务之间的依赖关系。
- **OpenEmbedded Core:** 一个常用的 Metadata Layer, 可以理解为这是 Yocto 项目的基础 Layer, 这个 Layer 是由 Yocto 和 OpenEmbedded 共同维护的。Core 可以理解为这是整个构建系统的核心层。
- **Poky:** Poky 有两个含义, 第一个是用来构建 Linux 的构建系统, 它包含了 BitBake 工具、编译工具链、BSP、诸多程序包或 Layer, 可以认为 Poky 即是 Yocto 的本质。此外, 使用 Poky 系统得到的默认参考 Linux 发行版也叫 Poky。

2 系统要求

2.1 操作系统要求

目前，以下发行版本支持 Yocto GLP3.0 项目的开发：

- Ubuntu 20.04/22.04
- Fedora release 38
- Debian GNU/Linux 11.x (Bullseye)
- AlmaLinux 8

说明

请使用推荐的系统版本，否则可能会出现非预期的编译错误。

2.2 工具包安装

使用的操作系统不同，需要的编译支持工具包也不同。请根据使用需求，安装编译依赖环境的工具包。

- Ubuntu 和 Debian

```
$ sudo apt install gawk wget git diffstat unzip texinfo gcc build-essential chrpath socat cpio python3 python3-pip python3-pexpect xz-utils debianutils iputils-ping python3-git python3-jinja2 python3-subunit zstd liblz4-tool file locales libacl1
```

```
$ sudo locale-gen en_US.UTF-8
```

- Fedora Packages

```
$ sudo dnf install gawk make wget tar bzip2 gzip python3 unzip perl patch diffutils diffstat git cpp gcc gcc-c++ glibc-devel texinfo chrpath ccache perl-Data-Dumper perl-Text-ParseWords perl-Thread-Queue perl-Bignum socat python3-pexpect findutils which file cpio python python3-pip xz python3-GitPython python3-jinja2 rpcgen perl-FindBin perl-File-Compare perl-File-Copy perl-locale zstd lz4 hostname glibc-langpack-en libacl
```

- openSUSE Packages

```
$ sudo zypper install python gcc gcc-c++ git chrpath make wget python-xml diffstat makeinfo python-curses patch socat python3 python3-curses tar python3-pip python3-pexpect xz which python3-Jinja2 rpcgen zstd lz4 bzip2 gzip hostname libacl1
```

```
$ sudo pip3 install GitPython
```

- AlmaLinux Package

```
$ sudo dnf install -y epel-release
```

```
$ sudo yum install dnf-plugins-core
```

```
$ sudo dnf config-manager --set-enabled crb
```

```
$ sudo dnf makecache
```

```
$ sudo dnf install gawk make wget tar bzip2 gzip python3 unzip perl patch diffutils diffstat git cpp gcc gcc-  
c++ glibc-devel texinfo chrpath ccache socat perl-Data-Dumper perl-Text-ParseWords perl-Thread-Queue  
python3-pip python3-GitPython python3-jinja2 python3-pexpect xz which rpcgen zstd lz4 cpio glibc-  
langpack-en libacl
```

2.3 软件版本要求

Git、tar、Python、GCC 和 GNU make 必须满足以下版本要求：

- Git: 1.8.3.1 及以上版本。
- tar: 1.28 及以上版本。
- Python: 3.6.0 及以上版本。
- GCC: 7.5 及以上版本。
- GNU make: 4.0 及以上版本。

3

源码目录结构

源码目录结构如图 3-1 所示，详细介绍参考文档《Yocto 目录结构介绍》。

图3-1 源码目录结构

```
— bitbake
— build.sh -> prebuilts/scripts/build.sh
— layers
— prebuilts
— setup-environment -> prebuilts/scripts/setup-environment
— source
```

4 编译

编译对应项目，具体的 MACHINE/DISTRO，可以参考附录章节所述，需要根据具体的项目选择对应的 board 信息。

说明

本章节描述的 board 信息，仅用于举例说明，实际需要以具体项目 board 信息为准。

4.1 编译前准备

完成代码下载后，将对应工程的属性文件拷贝到代码根目录下：

```
cp -rf ./vendor/sprd/release/IDH/工程名/prebuilts/ unisoc-bin/${MACHINE}-${DISTRO} ./prebuilts/unisoc-bin
```

4.2 编译配置信息

MACHINE

- 路径：
`{RootDir}/layers/meta-unisoc/conf/machine`
- 包含文件，例如：
`uis7885-2h10.conf`
`uis8360-1h10-nand.conf`

说明

去掉扩展名.conf 就是 MACHINE，即
`uis7885-2h10`
`uis8360-1h10-nand`。

DISTRO

- 路径：
`{RootDir}/layers/meta-unisoc/conf/distro`
- 包含文件，例如：
`unisoc-wayland.conf`, `unisoc-initgc.conf`, `unisoc-tiny-image`

说明

去掉扩展名.conf 就是 DISTRO，包含 DISTRO：`unisoc-wayland`, `unisoc-tiny` 和 `unisoc-initgc`。

IMAGE

- 路径：
`{RootDir}/layers/meta-unisoc/recipes-core/images`
- 包含文件，例如：
`unisoc-wayland-image.bb`, `unisoc-console-image.bb`

说明

去掉扩展名.bb 就是 IMAGE，包含 IMAGE: `unisoc-wayland-image` 和 `unisoc-console-image`。

4.3 编译整个工程

当前工程支持通过命令行和图形界面这两种方式来配置 MACHINE、DISTRO 等信息。DISTRO 常用的有 `unisoc-initgc` 和 `unisoc-wayland`。

编译前，需要先确定 MACHINE 和 DISTRO 信息，以及编译生成的 IMAGE。

假设 `DISTRO=unisoc-wayland`，`MACHINE=uis8360-1h10-nand`，IMAGE 是 `unisoc-console-image`，两种方式的编译步骤参考 [4.3.1 命令行方式配置环境](#) 和 [4.3.2 图形界面方式配置环境](#)。

4.3.1 命令行方式配置环境

步骤 1 在工程的根目录，运行如下命令：

```
MACHINE= uis8360-1h10-nand DISTRO=unisoc-tiny source setup-environment
```

说明

等号两边不能有空格。

步骤 2 编译 IMAGE，运行如下命令：

```
bitbake unisoc-console-image
```

编译成功后，IMAGE 存放路径：

```
${RootDir}/build-${DISTRO}/tmp-${DISTRO}-glibc(musl)/deploy/images/${MACHINE}/
```

----结束

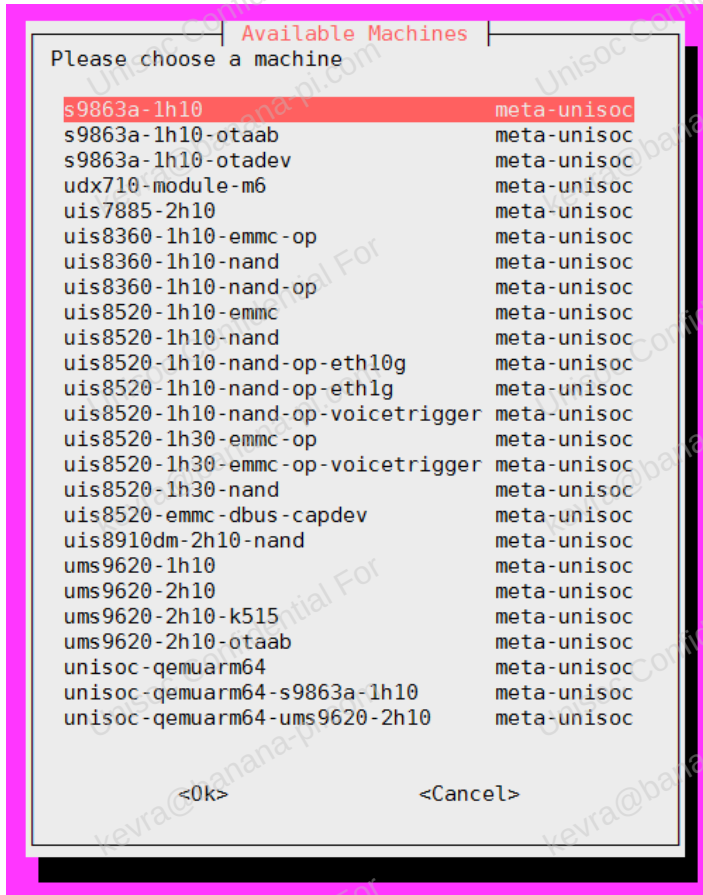
4.3.2 图形界面方式配置环境

步骤 1 在根目录运行如下命令：

```
source setup-environment
```

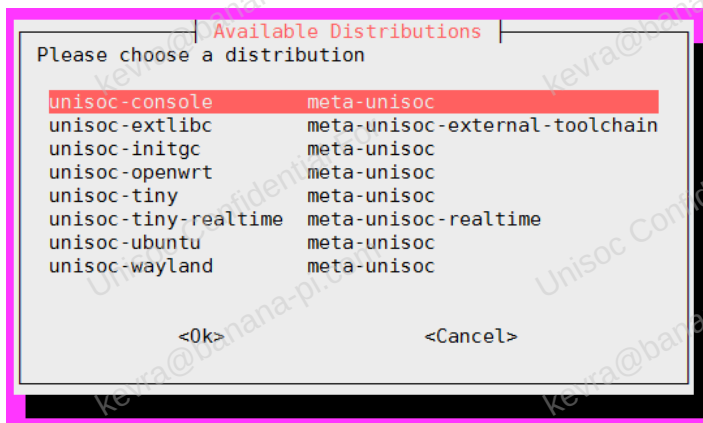
步骤 2 选择 MACHINE，如 [图 4-1](#)，可以使用键盘的上下键选择 MACHINE，选中的 MACHINE 背景为红色，然后按下回车键。

图4-1 MACHINE 信息



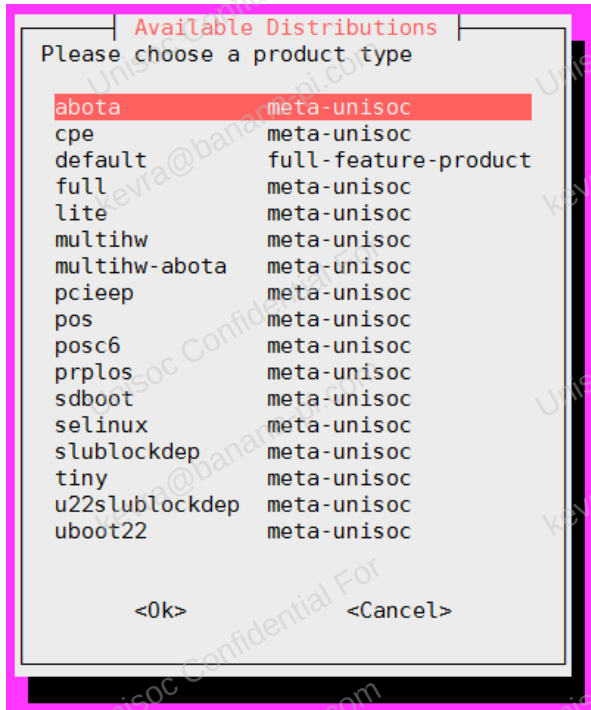
步骤 3 选择 DISTRO，如图 4-2，选中 DISTRO 后按下回车键。

图4-2 DISTRO 信息



步骤 4 选择 product type，根据工程搭配进行选择，如 uis8360 搭配 lite/full 等。如 uis8360-1h10-nand 可搭配 lite/full 等。

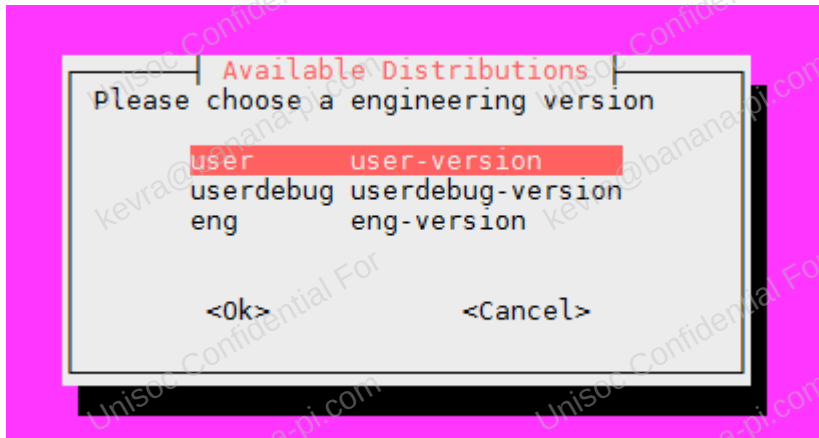
图4-3 选择 product type



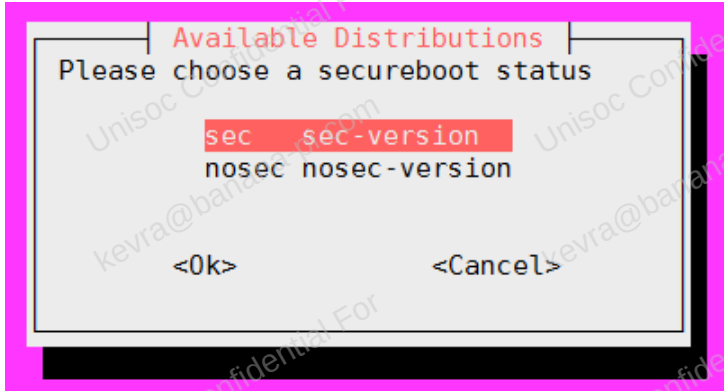
步骤 5 选择 engineering version，根据需求选择对应版本即可。

其中，eng 是把 meta-unisoc-test layer 进行编译，用于测试。

图4-4 选择 engineering version



步骤 6 Secure boot 选择，可以根据工程需要选择是否安全版本。



步骤 7 使用如下命令编译 IMAGE。

```
bitbake unisoc-console-image
```

编译成功后，IMAGE 存放路径：

```
${RootDir}/build-${DISTRO}/tmp-${DISTRO}-glibc/deploy/images/${MACHINE}/*.ext4
```

每个 MACHINE 用到的 IMAGE 类型可能不同。

- 编译生成的 IMAGE 是 ubifs 类型

```
IMAGE_FSTYPES_append = " ubifs"
```

- 编译生成的 IMAGE 是 ext4 类型

```
IMAGE_FSTYPES_append = " ext4"
```

----结束

4.4 编译单个模块

在实际开发过程中，有时候只需要编译单个模块，方法如下：

```
bitbake -C compile moduleName
```

说明

moduleName 对应工程中 moduleName.bb 或 moduleName_1.1.bb，即截取下划线或者点之前的信息。

4.5 编译 MACHINE 的 Kernel/U-Boot/Chipram

单独编译 MACHINE 的 Kernel、U-Boot、Chipram，编译方法如下：

步骤 1 切换到 Yocto 根目录。

步骤 1 以用户选择 DISTRO=unisoc-tiny 为例，运行如下命令：

```
MACHINE= uis8360-1h10-nand DISTRO=unisoc-tiny source setup-environment
```

步骤 2 编译 Kernel。

- 编译 Kernel 命令:

需要先确定对应 machine 依赖 kernel 的信息:

在<machine>.conf 或者*.inc 中查找关键字: PREFERRED_PROVIDER_virtual/kernel

```
bitbake -c cleansstate virtual/kernel && bitbake -C compile virtual/kernel
```

如果能确定对应 kernel 版本, 可执行如下命令:

```
bitbake -c cleansstate linux-unisoc-6.1 && bitbake -C compile linux-unisoc-6.1
```

方法源自 uis7885-2h10.conf 文件内部或者 include 文件, 涉及到的关键字是

PREFERRED_PROVIDER_virtual/kernel="linux-unisoc-6.1"。其他 MACHINE, 根据关键字 PREFERRED_PROVIDER_virtual/kernel, 能找到具体的 recipes。

- Kernel 编译结果:

```
${RootDir}/build-${DISTRO}/tmp-${DISTRO}-glibc/deploy/images/${MACHINE}/Image
```

步骤 3 编译 U-Boot。

- 编译 U-Boot 命令:

```
bitbake -c cleansstate u-boot22 && bitbake -C compile u-boot22
```

方法源自 uis8360-1h10-nand.conf 文件内部或者 include 的文件, 涉及到的关键字是

EXTRA_IMAGEDEPENDS += "u-boot22"。其他的 MACHINE, 方法类似, 都是查找关键字 EXTRA_IMAGEDEPENDS。

- U-Boot 编译结果:

```
${RootDir}/build-${DISTRO}/tmp-${DISTRO}-glibc/deploy/images/${MACHINE}/u-boot-  
${MACHINE}.*.bin
```

步骤 4 编译 Chipram。

- 编译 Chipram 命令:

```
bitbake -c cleansstate chipram && bitbake -C compile chipram
```

方法源自 uis8360-1h10-nand.conf 文件内部或者 include 的文件, 涉及到的关键字是

EXTRA_IMAGEDEPENDS += "chipram"。其他的 MACHINE, 方法类似, 都是查找关键字 EXTRA_IMAGEDEPENDS。

- Chipram 编译结果:

```
${RootDir}/build-${DISTRO}/tmp-${DISTRO}-glibc/deploy/images/${MACHINE}/fdl1.bin  
${RootDir}/build-${DISTRO}/tmp-${DISTRO}-glibc/deploy/images/${MACHINE}/u-boot-spl-  
16k.bin
```

说明

单独编译出来的 Kernel、U-Boot、Chipram 更新到设备中需要先签名, 签名完毕后, 更新签名后的 Kernel、U-Boot、Chipram 到设备中。

判断文件是否需要签名: 使用下载工具加载 pac 包, 加载成功后, pac 包会被解压到一个临时目录中。

- 如果文件是 boot.img, 表示该文件不需要签名。
- 如果文件是 boot-sign.img, 表示该文件是一个被签过名的文件。
- 对应的工程需要对上述参数进行正确的搭配。

----结束

4.6 生成 pac 包

可以使用 build.sh 脚本制作 pac 包，下载 Yocto 源码后可看到 build.sh。正常编译完后，pac 包在 out 目录下，路径为 ./out/target/product/\${MACHINE}/cp_sign/。

说明

生成 pac 包不需要配置 MACHINE 和 DISTRO，直接调用 build.sh 脚本即可。

脚本使用说明

```
./build.sh A+B+C+D+E F
```

- A 表示 MACHINE 类型的关键字，具体信息参考 9 附录。
- B 表示 DISTRO 类型的关键字，具体信息参考 9 附录。
- C 表示 IMAGE 类型的关键字，具体信息参考 9 附录。
- D 表示签名方式，参数有 sec 和 nsec。
 - sec: 使能安全方案。
 - nsec: 关闭安全方案。

具体使用的参数，以 pac 包版本为基础。如果展锐发布的 pac 包名字中含有 sec，表示当前启动了安全方案，因此编译 pac 包时，参数 D 要传入 sec。如果没有输入参数 D，默认编译 sec 的 pac 包。

- E 表示 product 类型的关键字，根据工程自行进行 product 的搭配。具体信息参考 9 附录。
- F 表示版本类型的关键字，userdebug/user/eng。如果此参数省略，默认是 userdebug。
 - userdebug: 含有 debug 功能的 pac 包。
 - user: 含有少量 debug 功能的 pac 包。
 - eng: 包含 debug 功能的基础上会集成 meta-unisoc-test 的 pac 包。

示例

- 生成 userdebug 类型的 pac 包，切换到根目录，运行如下命令（其中 full 为 product）：

```
./build.sh uis8360_1h10_nand+tiny+console+ nsec +full-userdebug userdebug
```

- 生成 user 类型的 pac 包，切换到根目录，运行如下命令：

```
./build.sh uis8360_1h10_nand+tiny+console+ nsec +full-userdebug user
```

- 运行如下命令，参数 E 默认是 userdebug：

```
./build.sh uis8360_1h10_nand+tiny+console+ nsec +full-userdebug
```

4.7 编译结果

编译结果路径：\${ROOTDIR}/build-\${DISTRO}/tmp-\${DISTRO}-glibc/deploy/images/\${MACHINE}/

例如：\${ROOTDIR}/build-unisoc-initgc/tmp-unisoc_initgcglibc/deploy/images/uis8360-1h10-nand

其中，生成的 U-Boot、Kernel 及 rootfs 镜像文件如下：

- uboot image: u-boot-u uis8360-1h10-nand -v2015.07+git999-r0.bin
- kernel image: boot.img

- rootfs: unisoc-wayland-image- uis8360-1h10-nand -20230323031429.rootfs.ext4.gz
不同项目对应的 rootfs 类型不同，查看 MACHINE 的.conf 文件内容可以获取 rootfs 类型。如 uis8360-1h10-nand.conf 定义的 rootfs 类型为 IMAGE_FSTYPES_append = " ext4.gz "。

Kernel、Chipram 及 U-Boot 的符号表路径如下：

- Kernel 符号表路径：\${ROOTDIR}/build-\${DISTRO}/tmp-\${DISTRO}-glibc/work/\${MACHINE}-unisoc-linux/\${KernelName}/*/\${KernelName}-*
 - *表示与 Linux 相关的版本号。
 - KernelName 对应 Kernel 编译时的名称。
 例如：\${ROOTDIR}/build-unisoc-initgc/tmp-unisoc_initgc-glibc/work/ uis8360-1h10-nand -unisoc-linux/linux-unisoc-6.1/6.1-r0/linux-unisoc-6.1-6.1/vmlinux
- Chipram 符号表路径：\${ROOTDIR}/build-\${DISTRO}/tmp-\${DISTRO}-glibc/work/\${MACHINE}-unisoc-linux/\${ChipramName}/*/\${ChipramName}-*
 - *表示与 Chipram 相关的版本号。
 - ChipramName 对应编译时选择的 Chipram 名称。
 例如：\${ROOTDIR}/build-unisoc-initgc/tmp-unisoc_initgc-glibc/work/ uis8360-1h10-nand -unisoc-linux/chipram/0.1-r0/chipram-0.1/nand_spl/u-boot-spl
- U-Boot 符号表路径：\${ROOTDIR}/build-\${DISTRO}/tmp-\${DISTRO}-glibc/work/\${MACHINE}-unisoc-linux/\${UbootName}/*/\${UbootName}-*
 - *表示与 U-Boot 相关的版本号。
 - UbootName 对应编译时选择的 U-Boot 的名称。
 例如：\${ROOTDIR}/build-unisoc-initgc/tmp-unisoc_initgc-glibc/work/ uis8360-1h10-nand -unisoc-linux/u-boot22/v2015.07+git999-r0/u-boot22v2015.07+git999/u-boot

4.8 手动签名方法

可使用 prebuilts/packimage_scripts/manual_sign.sh 脚本，手动签名。不同签名文件路径不同，可以先进行查找。以 UIS7885 平台为例，具体方法如下：

- 对 kernel 签名。

在 kernel 目录下，查找需要签名的 Image 文件，并且重命名为 boot.img。

```
find . -name Image
```

#进入对应的目录下，进行重命名

```
mv Image boot.img
```

将需要签名的 boot.img 文件复制到 prebuilts/packimage_scripts 目录下，使用 manual_sign.sh 脚本进行签名，执行如下命令：

```
./manual_sign.sh uis7885 boot.img
```

- 对 U-Boot 签名。

在 bsp/u-boot 目录下，查找签名文件。

```
find . -name u-boot.bin
```

将需要签名的 u-boot.bin 文件复制到 prebuilts/scripts/packimage_scripts 目录下，使用 manual_sign.sh 脚本进行签名，执行如下命令：

```
./manual_sign.sh uis7885 u-boot.bin
```

- 对 Chipram 签名。

在 bsp/chipram 目录下，查找签名文件。

```
find . -name u-boot-spl-16k.bin
```

将需要签名的 u-boot-spl-16k.bin 文件复制到 prebuilts/scripts/packimage_scripts 目录下，使用 manual_sign.sh 脚本进行签名，执行以下命令：

```
./manual_sign.sh uis7885 u-boot-spl-16k.bin
```

4.9 添加模块

本节以 pm 模块为例介绍如何添加编译模块，具体方法如下：

- 步骤 1 在 \$(root_dir)/layers/meta-unisoc/recipes-core 中，新建 pm 目录，添加 bb 文件。pm_0.0.1.bb 详情参见 [4.9.1 bb 文件](#)。

说明

bb 文件所在 recipes 的位置，需要与其功能相符合。例如 sml 安全相关的模块，那么对应的 sml.bb 就放置于 recipes-security 目录下。

- 步骤 1 使用 bitbake 工具测试文件是否添加成功。

配置模块的配置文件和编译环境变量后（配置 MACHINE 和 DISTRO），可以通过 bitbake -s 命令查看文件是否成功添加。

```
apadmin@BJ08432PCU1:~/work/unc_linux_trunk/build-unisoc-wayland$ bitbake -s |grep pm
pm                                :0.1-r0
```

- 步骤 2 使用 bitbake 工具进行编译。

编译命令：bitbake 模块名

```
~/work/unc_linux_trunk/build-unisoc-wayland $bitbake pm
```

- 步骤 3 添加模块到 IMAGE。

具体方法参见章节 [4.9.2 添加模块到 IMAGE](#)。

----结束

4.9.1 bb 文件

针对 pm_0.0.1.bb 文件内容进行相关变量和 bb 文件规范问题的说明。pm_0.0.1.bb 的主要内容如下：

```
DESCRIPTION = "Unisoc power manager module"
```

```
HOMEPAGE = "http://www.unisoc.com/"
```

```
LICENSE = "LGPL-3.0"
```

```
SECTION = "lib"
```

```
DEPENDS = ""
LIC_FILES_CHKSUM = "file://COPYING;md5=169502d5dcf12805080eb9ac0af3c8e1 \
file://COPYING.LESSER;md5=334a51ca2c88132a5313e4852ec468b3"
PV = "0.1"
PR = "r0"
PROVIDES = "pm"

SRC_URI = "ssh://gitadmin@gitmirror.unisoc.com/yocto/unisoc/pm;protocol=ssh;branch=unc_linux_trunk"
EXTERNALSRC_BUILD = "${EXTERNALSRC}/src"

do_compile () {
    make clean
    make
}

do_install () {
    install -d ${D}${libdir}/
    install -m 0644 ${B}/libpm.so ${D}${libdir}/
    install -m 0644 ${B}/libinput-uevent.so ${D}${libdir}/
    install -d ${D}${includedir}/
    install -m 0644 ${S}/src/pm.h ${D}${includedir}/${PN}/
    install -m 0644 ${S}/input_uevent/input_uevent.h ${D}${includedir}/${PN}/
}

FILES_${PN} += "${libdir}/libpm.so"
FILES_SOLIBSDEV = ""
INSANE_SKIP_${PN} = "dev-so"
TARGET_CC_ARCH += "${LDFLAGS}"
```

通常 bb 文件可以分成如下几个部分：

- bb 文件基本信息。
- LICENSE 相关信息
- 依赖关系信息
- 源码路径信息
- Task 的定义
- 打包相关配置信息

4.9.1.1 bb 文件基本信息

bb 文件基本信息主要包括模块说明、模块版本号、模块发布号等基本信息，如表 4-1 所示。

表4-1 bb 文件基本信息

参数名称	说明
DESCRIPTION	模块说明。
HOMEPAGE	模块相关公司主页（可省略）。
PV	模块版本号。
PR	模块发布号。
PROVIDES	模块别名。

模块别名通常默认包含 bb 文件的本身名字，例如 pm_0.0.1.bb，那么 PROVIDES 变量就默认包含了 pm 这个别名，方便后续这个模块被别的模块依赖。

4.9.1.2 LICENSE 相关信息

Yocto 通过表 4-2 相关参数指定 License 相关的信息。

表4-2 License 相关参数说明

参数名称	说明
LICENSE	模块所遵循的许可版本。 适用于配方构建的软件包的源码许可列表（可能单个也可能多个）。
LIC_FILES_CHKSUM	License 文件的 Checksum，可以通过 md5sum 命令生成如下： <pre>apadmin@BJ08432PCU1:~/work/unisoc/pm\$ md5sum COPYING 169502d5dcf12805080eb9ac0af3c8e1 COPYING</pre> LIC_FILES_CHKSUM 变量会跟踪许可证文本的更改。如果许可证文本发生变化，则会触发构建失败，这使开发人员有机会查看任何许可证更改。必须为所有配方定义此变量，除非 LICENSE 设置为“CLOSED”。 LIC_FILES_CHKSUM 的设置： <pre>LIC_FILES_CHKSUM = "file://COPYING;md5=bcd3ab47632e78cf1a3581b0b9e52794"</pre>

在 pm 模块的源码目录中，存在“COPYING”文件。许多项目都有一个“COPYING”文件，用于存储所有源代码文件的许可证信息。

```
~/YOCTO/yocto_board/source/unisoc/powermanager/pm$ md5sum COPYING
bcd3ab47632e78cf1a3581b0b9e52794  COPYING
```

LIC_FILES_CHKSUM 变量会跟踪许可证文本的更改。在配置步骤结束时验证校验和，如果校验和不匹配，构建将失败。防止上游项目的许可证将来可能会发生变化，更改被忽视。

如果直接基于开源软件进行开发，可以参考如下表格。

LICENSE	简述	使用要求	商标使用	是否允许附加协议	是否允许专利授权
MIT	宽松简单且精要的一个协议。在适当标明来源及免责的情况下，它允许你对代码进行任何形式的使用。	1)在代码中保留作者提供的协议和版权信息	可以使用作者的姓名、作品的Logo，或商标	允许	不允许
BSD	较为宽松的协议，包含两个变种BSD 2-Clause 和BSD 3-Clause，两者都与MIT协议只存在细微差异。	1)在代码中保留作者提供的协议和版权信息	可以使用作者的姓名、作品的Logo，或商标	允许	不允许
Apache	一个较宽松且简明地指出了专利授权的协议。	1)在代码中保留作者提供的协议和版权信息 2)在代码中声明对原来代码的重大修改及变更	不能使用	允许	允许
MPL	此协议旨在较为宽松的BSD协议和更加互惠的GPL协议中寻找一个折衷点。	1)在代码中保留作者提供的协议和版权信息 2)公开源码，代码必需公开	不能使用	允许	允许
EPL	对商用非常友好的一种协议，可以用于软件的商业授权。包含对专利的优雅授权，并且也可以对相关代码应用商业协议。	1)在代码中保留作者提供的协议和版权信息 2)公开源码，代码必需公开	可以使用作者的姓名、作品的Logo，或商标	允许	允许
LGPL	主要用于一些代码库。衍生代码可以以此协议发布（言下之意你可以用其他协议），但与此协议相关的代码必需遵循此协议。	1)在代码中保留作者提供的协议和版权信息 2)公开源码，代码必需公开 3) 涉及的库可以用于商业软件中	可以使用作者的姓名、作品的Logo，或商标	允许	允许
GPL	此协议是应用最为广泛的开源协议，拥有较强的版权自由(copyleft)要求。衍生代码的分发需开源并且也要遵守此协议。	1)在代码中保留作者提供的协议和版权信息 2)公开源码，代码必需公开 3) 在代码中声明对原来代码的重大修改及变更	可以使用作者的姓名、作品的Logo，或商标	不允许	允许

4.9.1.3 依赖关系信息

通过 RDEPENDS 可以指定运行时的依赖项，通过 DEPENDS 可以指定编译时的依赖项。

表4-3 依赖相关参数说明

参数名称	说明
RDEPENDS	该模块运行时所依赖的其他模块，如果没有就不需要。
DEPENDS	该模块编译时所依赖的其他模块，如果没有就不需要。

注意

这里通过 PROVIDES 这个变量去寻找依赖关系。例如在 pm-demo.bb 中定义了 DEPENDS += “pm”，那么会去找对应的模块作为依赖，被依赖的模块别名为 pm，也就是 PROVIDES 变量中包含 pm 的模块。

4.9.1.4 源码路径信息

在 Yocto 中，通常有如下两种获取源码的方式，这两种方式在指定源码路径时存在一定的差异。

- 从远程仓库获取开源代码，典型的例如 git 仓库。
- 从本地目录获取自研代码。

开源代码，从 git 仓库获取源码

以 recipes-graphics/littlevgl/lvgl_7.4.0.bb 模块为例，在对应的 bb 文件中指定源码的位置：

```
SRC_URI = "https://github.com/lvgl/lvgl/archive/v${PV}.tar.gz;downloadfilename=${PN}-${PV}.tar.gz \
file://0000-add-makefile.patch \
file://0001-add-lvconf.patch \
file://0002-modify-lvconf_internal.patch \
file://0003-modify-lv_font_mk.patch \
file://0004-add-lv_font_chinese_19.patch \
file://0005-modify-lvgl_h.patch \
file://0006-add-lv_multimedia_c.patch \
file://0007-add-lv_multimedia_h.patch \
file://0008-modify-lv_widgets_mk.patch \
file://0009-append-lv_conf.patch \
file://0010-add-lv_draw_blend_c.patch \
file://0011-add-lv_obj_h.patch \
file://0012-modify-lvconf_h.patch \
file://0013-modify-lvconf_internal_h.patch \
"
```

可以看到后面的 patch 是针对开源软件的修改。开源软件在官方的 meta 仓库下通常存在模块的 bb 文件，将 patch 增加到 bb 文件中，可完成功能的定制。

说明

代码中黄色高亮部分需要用户重点关注。

自研模块，从本地仓库获取源码

展锐使用 layers/meta-unisoc/conf/machine/include/unisoc-extern.inc 文件来统一管理本地源码，想要增加自己的仓库，在 layers/meta-unisoc/conf/machine/include/unisoc-extern.inc 文件中指定好路径即可。

```
INHERIT += "unisoc_externsrc"
OEROOT := "${@os.path.abspath(os.path.dirname(d.getVar('FILE', True)))}/../../.."

EXTERNALSRC_pn-linux-unisoc-5.4 = "${OEROOT}/source/kernel5.4/"
EXTERNALSRC_pn-linux-unisoc-5.15 = "${OEROOT}/source/kernel5.15/"
EXTERNALSRC_pn-linux-unisoc-6.1 = "${OEROOT}/source/kernel6.1/"

EXTERNALSRC_pn-u-boot22 = "${OEROOT}/source/bsp/u-boot22"
```

- INHERIT += "unisoc_externsrc" 表明使用本地仓库，bbclass 会自动的把 EXTERNALSRC 变量转换成 S 变量（把源码解压到的位置）。

- OEROOT 代表工程的根目录，这里记录了每个 recipes 对应的 EXTERNALSRC 值，后续即可转换成源码的目录。把这个值直接设置成仓库的路径，也就是源码的位置。

4.9.1.5 Task 的定义

do_compile

确保编译的产物都生成在 $\{\text{WORKDIR}\}$ ，而不是生成在模块仓库内。

do_compile(): 覆盖官方提供的 do_compile 过程进行编译，官方提供的 do_compile 如下：

```
base_do_compile() {
    if [ -e Makefile -o -e makefile -o -e GNUmakefile ]; then
        oe_runmake || die "make failed"
    else
        bbnote "nothing to compile"
    fi
}
```

如果不想覆盖，而是追加定义，那么可以采用 do_compile_prepend 和 do_compile_append 的 task，Yocto 会把这些与 do_compile 组合成最终的 compile task。

以 pm 模块增加如下 do_compile task 为例：

1. 进入 pm 源码目录编译，会把生成的产物传入到 $\{\text{B}\}$ 目录下：

```
do_compile () {
    make libpm -C  $\{\text{S}\}$  OBJ_DIR= $\{\text{B}\}$ 
    make libinput-uevent -C  $\{\text{S}\}$  OBJ_DIR= $\{\text{B}\}$ 
}
```

2. 相应的处理在 Makefile 中控制，如下所示：

```
ifndef OBJ_DIR
    OBJ_DIR = obj
Endif
.....
OBJS = $(patsubst %.c,$(OBJ_DIR)/%.o,$(notdir $(SRCS)))

libpm:dir $(OBJS)
    $(CC) $(CFLAGS) -shared -o $(TARGET) $(OBJS) $(LIBS)

$(OBJS):$(SRCS)
    $(foreach var,$^,\
        $(CC) $(CFLAGS) -c $(var) -o $(patsubst %.c,$(OBJ_DIR)/%.o,$(notdir $(var)));\
    )
```

do_install

对于自研模块的头文件，确保安装到 `usr/include/${PN}/` 目录下，PN 指代模块名。同时控制文件的权限。

例如，对于 pm 模块，在 install task 会将一些头文件安装到指定路径，后续集成到 rootfs，指定的路径通常为 `${D}${includedir}/` 目录（`/usr/include` 目录），直接安装到 `/usr/include/pm/` 目录下。这样可以防止编译 eSDK 时，多个模块安装同名的头文件（`log.h`）到同一目录，造成错误。

规范的 install task 如下：

```
do_install() {
    install -d ${D}${libdir}/
    install -m 0644 ${B}/libpm.so ${D}${libdir}/
    install -m 0644 ${B}/libinput_uevent.so ${D}${libdir}/
    install -d ${D}${includedir}/
    install -m 0644 ${S}/src/pm.h ${D}${includedir}/${PN}/
    install -m 0644 ${S}/input_uevent/input_uevent.h ${D}${includedir}/${PN}/
}
```

其中，`pm.h` 和 `input_uevent.h` 是 pm 模块独有的，很难出现同名，但仍推荐安装到 `${D}${includedir}/${PN}/` 目录下。权限可以是 0644，降低相应的权限。

4.9.1.6 打包相关配置信息

打包相关配置信息如表 4-4 所示。

表4-4 打包相关配置信息

参数名称	说明
FILES_\${PN}	填充相应的包中的文件。
FILES_SOLIBSDEV	解决打包过程中的 QA error。
INSANE_SKIP	跳过打包 QA 检查项，不建议使用这个变量。
TARGET_CC_ARCH	解决打包过程中的 QA error。

4.9.2 添加模块到 IMAGE

模块统一使用 packagegroup 方式管理。packagegroup 相关文件路径：

`$(root_dir)/layers/meta-unisoc/recipes-core/packagegroups`

步骤 1 在 packagegroup-unisoc-base.bb 添加 packagegroup 定义，packagegroup-unisoc-base-pm 相关内容：

```
SUMMARY = "Organize packages to avoid duplication across all images"

PACKAGE_ARCH = "${MACHINE_ARCH}"
```

```
inherit packagegroup

# packages which content depend on MACHINE_FEATURES need to be MACHINE_ARCH
#
# if you want to use a distro feature control a package ,you cam write as this example
# RDEPENDS_packagegroup-unisoc-console += "${@bb.utils.contains('DISTRO_FEATURES',
'distrofeaturename', 'packagename', ",d)}"
# if you want to use a machine feature control a package ,you can write like this
# RDEPENDS_packagegroup-unisoc-console += "${@bb.utils.contains('MACHINE_FEATURES',
'machinefeaturename', 'packagename', ",d)}"

# if you want to add a single package ,this is a example
# SUMMARY_packagegroup-base-wifi = "WiFi support"
# RDEPENDS_packagegroup-base-wifi = "\
# ${VIRTUAL-RUNTIME_wireless-tools} \
# wpa-supPLICANT"
#
# and then add thie package to PACKAGES arg and PROVIDES.

PROVIDES = "${PACKAGES}"

PACKAGES = '\
packagegroup-unisoc-base \
    ${@bb.utils.contains('MACHINE_FEATURES', 'propertys', 'packagegroup-unisoc-base-propertys',
",d)} \
packagegroup-unisoc-base-debug \
packagegroup-unisoc-base-debug-user \
    ${@bb.utils.contains('MACHINE_FEATURES', 'connman', 'packagegroup-unisoc-base-connman',
",d)} \
    ${@bb.utils.contains('MACHINE_FEATURES', 'modem', 'packagegroup-unisoc-base-modem',
",d)} \
    ${@bb.utils.contains('MACHINE_FEATURES', 'radio', 'packagegroup-unisoc-base-radio', ",d)} \
packagegroup-unisoc-base-rgbled \
    ${@bb.utils.contains('MACHINE_FEATURES', 'engpc', 'packagegroup-unisoc-base-engpc', ",d)} \
    ${@bb.utils.contains('MACHINE_FEATURES', 'pm', 'packagegroup-unisoc-base-pm', ",d)} \
,
.....
.....
```

```
.....

SUMMARY_packagegroup-unisoc-base-pm = "Power management"
RDEPENDS_packagegroup-unisoc-base-pm = "\
    input-uevent \
    pm \
"
```

步骤 2 模块添加到需要的 packagegroup 中。

目前有如下 group:

- packagegroup-unisoc-docker
- packagegroup-unisoc-tiny
- packagegroup-unisoc-console
- packagegroup-unisoc-minimal

在 packagegroup-unisoc-console.bb 中添加 packagegroup-unisoc-base-pm:

```
SUMMARY = "Organize packages to avoid duplication across all images"

PACKAGE_ARCH = "${MACHINE_ARCH}"
inherit packagegroup

# packages which content depend on MACHINE_FEATURES need to be MACHINE_ARCH
#
# if you want to use a distro feature control a package ,you cam write as this example
# RDEPENDS_packagegroup-unisoc-console += "${@bb.utils.contains('DISTRO_FEATURES',
'distrofeaturename', 'packagename', ",d)}"
# if you want to use a machine feature control a package ,you can write like this
# RDEPENDS_packagegroup-unisoc-console += "${@bb.utils.contains('MACHINE_FEATURES',
'machinefeaturename', 'packagename', ",d)}"

# if you want to add a single package ,this is a example
# SUMMARY_packagegroup-base-wifi = "WiFi support"
# RDEPENDS_packagegroup-base-wifi = "\
${VIRTUAL-RUNTIME_wireless-tools} \
wpa-suplicant"
#
# and then add thie package to PACKAGES arg and PROVIDES.

PROVIDES = "${PACKAGES}"
```

```
PACKAGES = '\
    packagegroup-unisoc-console \
    ,

DEBUG_USER = "${@bb.utils.contains('MACHINE_FEATURES', 'debug-user', 'packagegroup-unisoc-base-debug-user', ",d)}"

RDEPENDS_packagegroup-unisoc-console ??= "\
    kernel-modules \
    os-release \
    ${@bb.utils.contains('MACHINE_FEATURES', 'property', 'packagegroup-unisoc-base-property', ",d)} \
    ${@bb.utils.contains('MACHINE_FEATURES', 'lrzsz', 'lrzsz', ", d)} \
    ${@bb.utils.contains('USERDEBUG', 'userdebug', 'packagegroup-unisoc-base-debug', ", d)} \
    ${@bb.utils.contains('USERDEBUG', 'userdebug', 'packagegroup-unisoc-base-debug-user', DEBUG_USER, d)} \
    ${@bb.utils.contains('MACHINE_FEATURES', 'connman', 'packagegroup-unisoc-base-connman', ",d)} \
    ${@bb.utils.contains('MACHINE_FEATURES', 'modem', 'packagegroup-unisoc-base-modem', ",d)} \
    ${@bb.utils.contains('MACHINE_FEATURES', 'radio', 'packagegroup-unisoc-base-radio', ",d)} \
    ${@bb.utils.contains('DISTRO_FEATURES', 'zram', 'zram', ",d)} \
    ${@bb.utils.contains('MACHINE_FEATURES', 'rgbled', 'packagegroup-unisoc-base-rgbled', ",d)} \
    \
    ${@bb.utils.contains('MACHINE_FEATURES', 'keyevent', 'keyevent', ",d)} \
    ${@bb.utils.contains('MACHINE_FEATURES', 'engpc', 'packagegroup-unisoc-base-engpc', ",d)} \
    ${@bb.utils.contains('MACHINE_FEATURES', 'pm', 'packagegroup-unisoc-base-pm', ",d)} \
    "

```

步骤 3 添加 pm 到<MACHINE>.conf 的 MACHINE_FEATURES。以 uis7885-2h10.conf 为例，格式如下：

```
MACHINE_FEATURES += "trusty usbhost pm fscmsdos production "
```

步骤 4 完整编译，编译整个 IMAGE。

以 uis7885-2h10 为例，命令格式如下：

```
~/work/unc_linux_trunk/build-unisoc-console $bitbake unisoc-wayland-image
```

编译成功后，pm 模块就被添加到 rootfs 中。

----结束

4.10 添加 ko 模块

4.10.1 添加内部 Kernel ko 到 rootfs

当 Kernel 的某个 CONFIG_XXX=m，需要在<machine>.conf（例如 uis7885-2h10.conf）中添加如下信息：

```
IMAGE_INSTALL_append = " kernel-modules"
```

注意

kernel-modules 前面要有空格。

4.10.2 添加外部 Kernel ko 到 rootfs

步骤 1 创建 bb 文件，例如 hello-ko.bb。

步骤 2 配置 hello-ko.bb 的源码路径信息。

在文件 \${RootDir}layers/meta-unisoc/conf/machine/include/unisoc-extern.inc 尾部添加如下信息：

```
EXTERNALSRC_pn-hello-ko = "${OEROOT}/source/unisoc/hello-ko"
```

说明

OEROOT 表示 Yocto 的根目录。

步骤 3 添加 hello-ko.bb 基本信息

```
SUMMARY = "Example of how to build an external Linux kernel module"
LICENSE = "GPLv2"
LIC_FILES_CHKSUM = "file://COPYING;md5=12f884d2ae1ff87c09e5b7ccc2c4ca7e"
EXTERNALSRC_BUILD = "${EXTERNALSRC}"
inherit module

S = "${WORKDIR}"

# The inherit of module.bbclass will automatically name module packages with
# "kernel-module-" prefix as required by the oe-core build environment.
RPROVIDES_${PN} += "kernel-module-hello"
KERNEL_MODULE_AUTOLOAD += "hello"
```

- RPROVIDES_\${PN}：给模块起了一个别名，即生成的 ko 是 hello.ko。
- KERNEL_MODULE_AUTOLOAD：开机自动加载 hello.ko，等号前面的“+”号不要省略。如果开机不自动加载 ko，删除 KERNEL_MODULE_AUTOLOAD 这行信息。

步骤 4 添加源码。

- Makefile 内容如下：

```
obj-m := hello.o
SRC := $(shell pwd)
```

```
all:
    $(MAKE) -C $(KERNEL_SRC) M=$(SRC)
modules_install:
    $(MAKE) -C $(KERNEL_SRC) M=$(SRC) modules_install
clean:
    rm -f *.o *~ core *.depend *.cmd *.ko *.mod.c
    rm -f Module.markers Module.symvers modules.order
    rm -rf .tmp_versions Modules.symvers
```

- hello.c 内容如下：

```
#include <linux/module.h>

int init_module(void)
{
    printk("Hello World!\n");
    return 0;
}

void cleanup_module(void)
{
    printk("Goodbye Cruel World!\n");
}

MODULE_LICENSE("GPL");
```

- COPYING 文件的内容，根据实际情况添加。

以上信息，也可以参考如下地址信息：

`${RootDir}/layers/openembedded-core/meta-skeleton/recipes-kernel/hello-mod`

步骤 5 ko 与 MACHINE_FEATURE 进行绑定。

- 在文件`${RootDir}/layers/meta-unisoc/recipes-core/packagegroups/packagegroup-unisoc-base.bb`，添加如下内容：

```
23 PACKAGES = '\
24     packagegroup-unisoc-base \
25     ${@bb.utils.contains('MACHINE_FEATURES', 'propertys', 'packagegroup-unisoc-base-
propertys', "d")} \
26     packagegroup-unisoc-base-debug \
27     packagegroup-unisoc-base-debug-user \
28     ${@bb.utils.contains('MACHINE_FEATURES', 'connman', 'packagegroup-unisoc-base-
connman', "d")} \
```

```

29     ${@bb.utils.contains('MACHINE_FEATURES', 'modem', 'packagegroup-unisoc-base-
modem', ",d")} \
30     ${@bb.utils.contains('MACHINE_FEATURES', 'trusty', 'packagegroup-unisoc-base-
trusty', ",d")} \
31     ${@bb.utils.contains('MACHINE_FEATURES', 'radio', 'packagegroup-unisoc-base-
radio', ",d")} \
32     ${@bb.utils.contains('MACHINE_FEATURES', 'production', 'packagegroup-unisoc-base-
production', ",d")} \
33     ${@bb.utils.contains('MACHINE_FEATURES', 'aprd', 'packagegroup-unisoc-base-aprd',
",d")} \
34     packagegroup-unisoc-base-rgbled \
35     ${@bb.utils.contains('MACHINE_FEATURES', 'engpc', 'packagegroup-unisoc-base-
engpc', ",d")} \
36     ${@bb.utils.contains('MACHINE_FEATURES', 'pm', 'packagegroup-unisoc-base-pm',
",d")} \
37     ${@bb.utils.contains('MACHINE_FEATURES', 'hello', 'packagegroup-unisoc-base-
hello', ",d")} \
38     ${@bb.utils.contains('MACHINE_FEATURES', 'usbcontrol', 'packagegroup-unisoc-base-
usbcontrol', ",d")} \
39     ${@bb.utils.contains('MACHINE_FEATURES', 'pppdial', 'packagegroup-unisoc-base-
pppdial', ",d")} \
40     ${@bb.utils.contains('MACHINE_FEATURES', 'terminal', 'packagegroup-unisoc-base-
terminal', ",d")} \
41     ${@bb.utils.contains('MACHINE_FEATURES', 'alsa', 'packagegroup-unisoc-base-alsa',
",d")} \
42     ${@bb.utils.contains('MACHINE_FEATURES', 'gstreamer', 'packagegroup-unisoc-base-
gstreamer', ",d")} \
43     ${@bb.utils.contains('MACHINE_FEATURES', 'video', 'packagegroup-unisoc-base-
video', ",d")} \
44     ${@bb.utils.contains('MACHINE_FEATURES', 'memion', 'packagegroup-unisoc-base-
memion', ",d")} \
45 '

146 SUMMARY_packagegroup-unisoc-base-hello = "kernel module"
147 RDEPENDS_packagegroup-unisoc-base-hello = "
148     hello-ko \
149 "

```

说明

代码的 148 行表示，hello-ko（即 hello-ko.bb 文件）关联到 packagegroup-unisoc-base-hello，代码的 37 行通过 MACHINE_FEATURES 的 hello 就能控制 packagegroup-unisoc-base-hello。

- 在文件\${RootDir}/layers/meta-unisoc/recipes-core/packagegroups/packagegroup-unisoc-wayland.bb，添加如下内容：

```
31 RDEPENDS_packagegroup-unisoc-console ??= "\
32     kernel-modules \
33     os-release \
34     ${@bb.utils.contains('ADB_SRC_MODE', 'customer', '', 'adbd', d)} \
35     ${@bb.utils.contains('MACHINE_FEATURES', 'property', 'packagegroup-unisoc-base-
property', d)} \
36     ${@bb.utils.contains('MACHINE_FEATURES', 'lrzsz', 'lrzsz', d)} \
37     ${@bb.utils.contains('USERDEBUG', 'userdebug', 'packagegroup-unisoc-base-debug',
d)} \
38     ${@bb.utils.contains('USERDEBUG', 'userdebug', 'packagegroup-unisoc-base-debug-
user', DEBUG_USER, d)} \
39     ${@bb.utils.contains('MACHINE_FEATURES', 'connman', 'packagegroup-unisoc-base-
connman', d)} \
40     ${@bb.utils.contains('MACHINE_FEATURES', 'modem', 'packagegroup-unisoc-base-
modem', d)} \
41     ${@bb.utils.contains('MACHINE_FEATURES', 'trusty', 'packagegroup-unisoc-base-
trusty', d)} \
42     ${@bb.utils.contains('MACHINE_FEATURES', 'radio', 'packagegroup-unisoc-base-
radio', d)} \
43     ${@bb.utils.contains('DISTRO_FEATURES', 'zram', 'zram', d)} \
44     ${@bb.utils.contains('MACHINE_FEATURES', 'rgbled', 'packagegroup-unisoc-base-
rgbled', d)} \
45     ${@bb.utils.contains('MACHINE_FEATURES', 'keyevent', 'keyevent', d)} \
46     ${@bb.utils.contains('MACHINE_FEATURES', 'autoreboot', 'autoreboot', d)} \
47     ${@bb.utils.contains('MACHINE_FEATURES', 'usbcontrol', 'packagegroup-unisoc-base-
usbcontrol', d)} \
48     ${@bb.utils.contains('MACHINE_FEATURES', 'engpc', 'packagegroup-unisoc-base-
engpc', d)} \
49     ${@bb.utils.contains('MACHINE_FEATURES', 'production', 'packagegroup-unisoc-base-
production', d)} \
50     ${@bb.utils.contains('MACHINE_FEATURES', 'pm', 'packagegroup-unisoc-base-pm',
d)} \
51     ${@bb.utils.contains('MACHINE_FEATURES', 'hello', 'packagegroup-unisoc-base-
hello', d)} \
52     ${@bb.utils.contains('MACHINE_FEATURES', 'powerd', 'powerd', d)} \
53     ${@bb.utils.contains('MACHINE_FEATURES', 'terminal', 'packagegroup-unisoc-base-
terminal', d)} \
54     ${@bb.utils.contains('MACHINE_FEATURES', 'thermald', 'thermald', d)} \
55     ${@bb.utils.contains('MACHINE_FEATURES', 'pppdial', 'packagegroup-unisoc-base-
pppdial', d)} \
56     ${@bb.utils.contains('MACHINE_FEATURES', 'fsckmsdos', 'fsckmsdos', d)} \
```

```

57     ${@bb.utils.contains('MACHINE_FEATURES', 'alsa', 'packagegroup-unisoc-base-alsa',
",d)} \
58     ${@bb.utils.contains('MACHINE_FEATURES', 'gstreamer', 'packagegroup-unisoc-base-
gstreamer', ",d)} \
59     ${@bb.utils.contains('MACHINE_FEATURES', 'apr', 'packagegroup-unisoc-base-apr',
",d)} \
60     ${@bb.utils.contains('MACHINE_FEATURES', 'video', 'packagegroup-unisoc-base-
video', ",d)} \
61     ${@bb.utils.contains('MACHINE_FEATURES', 'memion', 'packagegroup-unisoc-base-
memion', ",d)} \
62 "

```

步骤 6 在<machine>.conf（例如 uis7885-2h10.conf）添加 hello 信息。

```

MACHINE_FEATURES += "trusty usbhost pm hello fskcmsdos production usbcontrol thermald
autoreboot pppdial nvmerge nghttp2 terminal powerd rgbled aprd"

```

把 hello 添加到 MACHINE_FEATURES 列表中。

步骤 7 编译<image>。

image 参考 9 附录 中 IMAGE 列。以 uis7885-2h10 为例，命令如下：

```
bitbake unisoc-wayland-image
```

----结束

4.11 新建 product

当基于某个工程要区分一些配置或者使用不同配置的文件时，可以使用 product 进行区分。

下面介绍新建 product 的方法以及注意事项，以 UIS7885 举例。

如新建 product 名称为 small，基于 uis7885-2h10-uboot22 工程新建，步骤如下：

1. 在 layer/meta-unisoc 目录下新建如下文件：

```
conf/distro/include/product/distro-features-small.inc
```

其中内容为：

```

require distro-features-small.inc
OVERRIDES:append = ":${UNISOC_PRODUCT_CLASS}"
UNISOC_PRODUCT_CLASS = "uboot22"

```

基于 conf/machine/include/xml/uis7885-2h10-uboot22.xml 文件 copy：

```
conf/machine/include/xml/ uis7885-2h10-uboot22.xml
```

2. 检查 layer/meta-unisoc 下面是否有 lite 的目录，如果有，也需要新建新的 product 的目录，需要保证使用的 dts/config 的文件与 base 工程一致。
3. 进入 prebuilts/pac_config 目录下，查看是否存在 base board 的 ini 文件，如果存在，基于此文件完成 new product 的新增工作。

```
cp uis7885-2h10-uboot22.ini uis7885-2h10-small.ini
```

#将新增的 ini 文件完成关键字替换

#运行如下命令完成关键字的替换

```
sed -i "s/ lite /small/g" uis7885-2h10-small.ini
```

4. 新增 ota 相关文件，对应仓库 yocto/unisoc/unisoc_ota_pack

```
find . -name "uis7885-2h10-uboot22"
```

将找到的文件目录进行 copy

```
cp -rf config/ uis7885-2h10-uboot22/pack-config.json config/ uis7885-2h10-small/pack-config.json
```

4.12 新建 board

新建一个 board 需要完成如下操作：

- 新建 Chipram
- 新建 U-Boot
- 新建 MACHINE
- 新建 Kernel

本节以已有的 base board（uis7885-2h10）为基础，介绍基于 base board 新建一个 board（uis7870-test）的详细操作步骤。

为避免遗漏，必须使用以下 find 和 grep 命令在每个目录都进行查找和添加。

```
find . -name "*uis7885_2h10"
```

```
find . -name "*uis7885-2h10"
```

```
grep -inr "uis7885_2h10"
```

```
grep -inr "uis7885-2h10"
```

- 针对 find 查找出的文件，复制并修改文件名和文件内关键字。
- 针对 grep 查找出的内容，添加对应文件中关键字的内容。

4.12.1 新建 Chipram

进入 \${ROOTDIR}/source/bsp/chipram 目录下，执行如下操作：

步骤 1 进入 include/configs/目录下，复制 uis7885_2h10.h 文件并重命名为 uis7870_test.h。

```
cp uis7885_2h10.h uis7870_test.h
```

步骤 1 修改 \${ROOTDIR}/source/bsp/chipram 目录下的 board.def 文件。

```
diff --git a/board.def b/board.def
```

```
index fc6015d..2c0410a 100755
```

```
--- a/board.def
+++ b/board.def
@@ -295,6 +295,9 @@ uis7885_2h10_config : preconfig
uis7885_2h10_config : preconfig
    @$(MKCONFIG) $@ arm armv8 uis7885_2h10 spreadtrum qogirn6pro

+uis7870_test_config : preconfig
+    @$(MKCONFIG) $@ arm armv8 uis7870_test spreadtrum qogirn6pro
+
```

----结束

4.12.2 新建 U-Boot

以下修改针对 u-boot22 新建 board 的 U-Boot 部分。

步骤 1 添加 Kconfig 支持，在 arch/arm/mach-sprd/Kconfig 文件中新增 new board 信息，如下所示，根据 base board 的信息 copy new board 的相关信息。

```
config SYS_BOARD
    string "Board name"
    default "sp9863a_1h10" if TARGET_SP9863A_1H10
    default "uis7885_2h10" if TARGET_UIS7885_2H10
    default "uis7885_2h10_otaab" if TARGET_UIS7885_2H10_OTAAB
    default "uis8520_1h10_nand" if TARGET_UIS8520_1H10_NAND
    default "uis8520_1h10_emmc" if TARGET_UIS8520_1H10_EMMC
    default "uis8520_emmc_dbus_capdev" if TARGET_UIS8520_EMMC_DBUS_CAPDEV
    default "uis8520_1h20_nand" if TARGET_UIS8520_1H20_NAND
    default ""

config SYS_CONFIG_NAME
    string "Board configuration name"
    default "sp9863a_1h10" if TARGET_SP9863A_1H10
    default "uis7885_2h10" if TARGET_UIS7885_2H10
    default "uis7885_2h10_otaab" if TARGET_UIS7885_2H10_OTAAB
    default "uis8520_1h10_nand" if TARGET_UIS8520_1H10_NAND
    default "uis8520_1h10_emmc" if TARGET_UIS8520_1H10_EMMC
    default "uis8520_emmc_dbus_capdev" if TARGET_UIS8520_EMMC_DBUS_CAPDEV
    default "uis8520_1h20_nand" if TARGET_UIS8520_1H20_NAND
    default ""
```

步骤 1 修改 `board/sprd/Kconfig` 文件，新增 new board 相关内容。

```
config TARGET_UIS8520_1H10_NAND
    bool "Spreadtrum uis8520_1h10_nand"
    select ARM64

config TARGET_UIS8520_1H20_NAND
    bool "Spreadtrum uis8520_1h20_nand"
    select ARM64
```

步骤 2 添加 u-boot 的 DTS 配置，在 `arch/arm/dts/Makefile` 中，增加 new board 对应的 DTS 配置。

```
dtb-$(CONFIG_ARCH_SPRD) += \
```

```
sp9863a_1h10.dtb \  
uis7885_2h10.dtb \  
uis7885_2h10_otaab.dtb \  
uis8520_1h10_nand.dtb \  
uis8520_1h10_nand_op_eth1g.dtb \  
uis8520_1h10_nand_op_eth10g.dtb \  
uis8520_1h10_nand_op_voicetrigger.dtb \  
uis8520_1h10_emmc.dtb \  
uis8520_emmc_dbus_capdev.dtb \  
uis8520_1h20_nand.dtb
```

步骤 3 增加 new board 的 DTS 文件，在 arch/arm/dts 目录下增加 new board 的 DTS 文件，里面存放的是与外设有关的配置信息，该文件主要是预留内存空间和堆空间。

```
cp uis8520_1h10_nand.dts uis8520_1h20_nand.dts  
# 进入 m710_test.dts 文件完成相应的替换
```

步骤 4 新增对于 new board 的目录文件，进入 board/sprd 下查看 base board 的相关目录，进行 copy。

```
cp -rf uis8520_1h10_nand uis8520_1h20_nand  
# 进入 uis8520_1h20_nand 目录完成相应的替换
```

步骤 5 添加.h 文件，在 include/configs 目录下的.h 文件主要是一些功能宏的定义，需要增加 new board 相关的.h 文件。

```
cp -r uis8520_1h10_nand.h uis8520_1h20_nand.h  
# 进入 uis8520_1h20_nand.h 目录下完成相应的替换
```

步骤 6 增加 new board 的 defconfig 文件，在 configs 目录下新增 new board 的 defconfig 文件，并完成关键字的修改。

```
cp uis8520_1h10_nand_defconfig uis8520_1h20_nand_defconfig  
# 进入 uis8520_1h20_nand_defconfig 文件中完成相应的替换
```

----结束

4.12.3 新建 MACHINE

进入 layers/meta-unisoc/ 目录下，执行如下操作：

步骤 1 查看 base board 对应的文件，复制和重命名找到的文件及目录（文件夹和文件都需要复制）。复制和重命名要求如下：

- 复制到原文件及目录的所在目录。
- 重命名文件及目录的名称时，将 uis7885-2h10 替换为 uis7870-test。

```
$ find . -name "* uis7885-2h10*"
conf/machine/uis7885-2h10.conf
recipes-bsp/formfactor/files/uis7885-2h10/
recipes-bsp/thermal/files/uis7885-2h10/
recipes-bsp/usbscontrol/files/uis7885-2h10/
recipes-connectivity/channel_manager/files/uis7885-2h10/
recipes-connectivity/modem/files/uis7885-2h10/
recipes-connectivity/wcnini/marlin3lite/uis7885-2h10/
recipes-devtools/charge/files/uis7885-2h10/
recipes-devtools/engpc/files/config/dev/uis7885-2h10/
recipes-devtools/engpc/files/uis7885-2h10/
recipes-devtools/nativemmi/files/config/uis7885-2h10/
recipes-multimedia/alsa/alsa-lib/uis7885-2h10/
recipes-multimedia/audio/files/uis7885-2h10/
recipes-multimedia/audio/files/uis7885-2h10/config/
recipes-multimedia/libcamera/files/uis7885-2h10/
recipes-multimedia/pulseaudio/pulseaudio/uis7885-2h10/
recipes-update/libubootenv/libubootenv/uis7885-2h10/
recipes-update/swupdate/swupdate/uis7885-2h10/
recipes-security/trustzone/configs/uis7885-2h10.inc
recipes-security/trustzone/teecfg/uis7885/uis7885-2h10/
#复制命令，以其中一个为例，其他类似
cp -r recipes-bsp/formfactor/files/uis7885-2h10/ recipes-bsp/formfactor/files/uis7870-test/
```

其中，/conf/machine/uis7885-2h10.conf 为 base board 的配置文件，其余为不同 recipes 中涉及到 base board 的文件，进行新增即可，并且将文件名 uis7885-2h10 替换为 uis7870-test。最终目录结构如下：

conf/machine/uis7870-test.conf

```

recipes-bsp/formfactor/files/uis7870-test/machconfig
recipes-bsp/thermal/files/uis7870-test/thermalSensorsConfig.xml
recipes-bsp/usbcontrol/files/uis7870-test/usbenum-user.sh
recipes-bsp/usbcontrol/files/uis7870-test/usbenum.ini
recipes-bsp/usbcontrol/files/uis7870-test/usbenumfirst-user
recipes-bsp/usbcontrol/files/uis7870-test/usbenumfirst.service
recipes-connectivity/channel_manager/files/uis7870-test/ptyconf.ini
recipes-connectivity/channel_manager/files/uis7870-test/system.ini
recipes-connectivity/modem/files/uis7870-test/modem_ch_info.xml
recipes-connectivity/modem/files/uis7870-test/modem_cp_info.xml
recipes-connectivity/modem/files/uis7870-test/modem_sp_info.xml
recipes-connectivity/modem/files/uis7870-test/system_user.ini
recipes-connectivity/modem/files/uis7870-test/system_userdebug.ini

recipes-connectivity/wcnini/marlin3lite/uis7870-test/bt_configure_pskey.ini
recipes-connectivity/wcnini/marlin3lite/uis7870-test/bt_configure_pskey_aa.ini
recipes-connectivity/wcnini/marlin3lite/uis7870-test/bt_configure_rf.ini
recipes-connectivity/wcnini/marlin3lite/uis7870-test/bt_configure_rf_aa.ini
recipes-connectivity/wcnini/marlin3lite/uis7870-test/fm_board_config.ini
recipes-connectivity/wcnini/marlin3lite/uis7870-test/wifi_board_config.ini
recipes-connectivity/wcnini/marlin3lite/uis7870-test/wifi_board_config_aa.ini
recipes-devtools/charge/files/uis7870-test/charge-init.sh
recipes-devtools/charge/files/uis7870-test/charge.service
recipes-devtools/engpc/files/config/dev/uis7870-test/cp.conf
recipes-devtools/engpc/files/uis7870-test/normal_lite_mach.conf
recipes-devtools/engpc/files/uis7870-test/normal_lite_user_mach.conf
recipes-devtools/nativemmi/files/config/uis7870-test/PCBA.conf
recipes-devtools/nativemmi/files/config/uis7870-test/nativemmi.ini

recipes-multimedia/alsa/alsa-lib/uis7870-test/HiFi.conf
recipes-multimedia/alsa/alsa-lib/uis7870-test/VoiceCall.conf
recipes-multimedia/alsa/alsa-lib/uis7870-test/sprphone-sc2730.conf
recipes-multimedia/alsa/alsa-lib/uis7870-test/sprphone.conf

recipes-multimedia/audio/files/uis7870-test/audio_param/audio_pga.xml
recipes-multimedia/audio/files/uis7870-test/audio_param/audio_process.xml
recipes-multimedia/audio/files/uis7870-test/audio_param/audio_structure.xml
recipes-multimedia/audio/files/uis7870-test/audio_param/audioparam_config.xml
recipes-multimedia/audio/files/uis7870-test/audio_param/codec.xml
recipes-multimedia/audio/files/uis7870-test/audio_param/cvs.xml
recipes-multimedia/audio/files/uis7870-test/audio_param/dsp_vbc.xml
recipes-multimedia/audio/files/uis7870-test/config/audio_config.xml
recipes-multimedia/audio/files/uis7870-test/config/audio_pcm.xml
recipes-multimedia/audio/files/uis7870-test/config/audio_route.xml
recipes-multimedia/audio/files/uis7870-test/config/codec_pga.xml
recipes-multimedia/libcamera/files/uis7870-test/camera_property.ini
recipes-multimedia/libcamera/files/uis7870-test/sensor_config.xml
recipes-multimedia/pulseaudio/pulseaudio/uis7870-test/earpiece_output.conf
recipes-multimedia/pulseaudio/pulseaudio/uis7870-test/headphone_output.conf
recipes-multimedia/pulseaudio/pulseaudio/uis7870-test/music-analog-output.conf
recipes-multimedia/pulseaudio/pulseaudio/uis7870-test/pulseaudio-bluetooth.conf
recipes-multimedia/pulseaudio/pulseaudio/uis7870-test/pulseaudio-ufon.conf
recipes-multimedia/pulseaudio/pulseaudio/uis7870-test/speaker_output.conf
recipes-multimedia/pulseaudio/pulseaudio/uis7870-test/sprd-input-headphone-mic.conf
recipes-multimedia/pulseaudio/pulseaudio/uis7870-test/sprd-input-internal-mic.conf
recipes-multimedia/pulseaudio/pulseaudio/uis7870-test/sprd-input-internal-signal-mic.conf
recipes-multimedia/pulseaudio/pulseaudio/uis7870-test/sprd_default.conf
  
```

完成上面的内容后，对如下目录进行检查：

```
$ find . -name "uis7885"
```

```
conf/machine/include/uis7885/
```

```
recipes-kernel/linux/linux-unisoc-5.4/uis7885/
```

```
recipes-security/trustzone/teecfg/uis7885/
```

#复制命令，以其中一个为例，其他类似

```
cp -r recipes-security/trustzone/teecfg/uis7885/ recipes-security/trustzone/teecfg/uis7870/
```

以上文件目录，文件包含 uis7885 的更宏观目录，存放着配置文件，这里需要对应新增 uis7870 目录。

conf/machine/include/xml/uis7870-test.xml

- conf/machine/include/uis7870/目录结构如下：

```
├── camera.inc
├── gpu.inc
├── trusty.inc
├── uis7870.inc
└── video.inc
```

- recipes-kernel/linux/linux-unisoc-5.4/uis7870/目录结构如下：

```
├── 0001-Bug-1982891-change-androidboot-to-sprdboot-in-n6pro.patch
├── 0002-fix-user-compile-error.patch
├── 0003-enable-trusty-vendor-hook.patch
├── add-dtbo-uis7870.patch
├── recovery-uis7870_defconfig
├── recovery-uis7870-test.cfg
├── recovery-uis7870-test.dts
├── sprd_uis7870_defconfig
├── uis7870.dts
├── uis7870-test-audio.cfg
├── uis7870-test.cfg
└── uis7870-test-overlay.dts
```

- recipes-security/trustzone/teeconfig/uis7870/目录结构如下：

```
├── uis7870-test
├── tee.xml
└── wifionly.xml
```

步骤 1 将文件名由 base board 修改为 new board。每个文件下，含有关键字的文件也需同步修改。

将 uis7885-2h10 修改为 uis7870-test，或将 uis7885 修改为 uis7870，需要注意下划线和连字符的区分。

注意事项：

- recipes-kernel/linux/linux-unisoc-5.4/uis7870/路径下，文件中的内容只需要将 uis7885-2h10 修改为 uis7870-test 即可。同样注意大小写、下划线、连字符的区分。
- conf/machine/uis7870-test.conf 文件，需要将含有 uis7885-2h10、uis7885_2h10、uis7885 的内容都进行替换。其中涉及到部分关键字的修改容易忽略，例如：

```
$ cat conf/machine/uis7870-test.conf
require conf/machine/include/uis7870/uis7870.inc
...
KERNEL_BOARD = "uis7870-test"
KERNEL_DT_SUBDIR = "${@bb.utils.contains('DEFAULTTUNE','aarch64','sprd/',",d")}"
UBOOT_BOARD = "uis7870_test"
SENSOR_BOARD="uis7870_test"
```

```
KERNEL_DEFCONFIG = "sprd_uis7870_defconfig"
KERNEL_MACHINE_PATCH = "file://uis7870-test.cfg \
file://uis7870-test-audio.cfg \
```

#以上修改不完全，只是对应的例子，需要将类似的部分全部修改

- 其他 uis7870-test 目录，可以执行如下命令，遍历当前目录及其子目录的所有文件，逐个文件查找 uis7885-2h10 文本并用 uis7870-test 替换。

```
sed -i "s/uis7885-2h10/uis7870-test/g" `grep -rl uis7870-test ./`
```

在其他目录下，重复执行命令，直至遍历完所有 uis7870-test 目录下的文件。

- 步骤 2** 部分特殊的 bbappend 文件中含有 uis7885-2h10，需要根据 base board 增加对应的内容，然后替换关键字。

```
$ cat recipes-graphics/wayland/wayland_%.bbappend
FILESEXPATHS_append :=
"${THISDIR}/${PN}:${THISDIR}/${PN}/default:${THISDIR}/wayland"

SRC_URI += "\
    ${@bb.utils.contains('GPU_BOARD', 'uis7885-2h10', 'file://not_install_libwayland.diff', ", d)} \
    ${@bb.utils.contains('GPU_BOARD', 'uis7885-2h10-otaab', 'file://not_install_libwayland.diff', ",
d)} \
    ${@bb.utils.contains('GPU_BOARD', 'uis7870-test', 'file://not_install_libwayland.diff', ", d)} \
    "

$ cat recipes-graphics/wayland/weston_%.bbappend
# todo need add otaab board
CONTROL_LIBWAYLAND = "${@bb.utils.contains('GPU_BOARD', 'uis7885-2h10', ", '-Dbackend-
wayland=true', d)}"
CONTROL_LIBWAYLAND = "${@bb.utils.contains('GPU_BOARD', 'uis7870-test', ", '-Dbackend-
wayland=true', d)}"
```

```
recipes-graphics/wayland/wayland_%.bbappend
recipes-graphics/wayland/weston_%.bbappend
```

- 步骤 3** 进入 prebuilts/pac-binary 目录，查看是否包含 new board 的目录。例如对于 uis7870-test，由于不存在 uis7870 目录，因此需要新建 uis7870 目录，并把对应 uis7885 目录下文件复制过来，并替换 uis7885 相关关键字。

```
cp -r uis7885 uis7870
```

- 步骤 4** 进入 prebuilts/pac_config 目录下，查看是否存在 base board 的 ini 文件，如果存在，基于此文件完成 new board 的新增工作。

```
cp uis7885-2h10.ini uis7870-test.ini
#将新增的 ini 文件完成关键字替换
#运行如下命令完成关键字的替换
```

```
sed -i "s/uis7885-2h10/uis7870-test/g" uis7870-test.ini
sed -i "s/uis7885_2h10/uis7870_test/g" uis7870-test.ini
```

- 步骤 5 source/unisoc 目录包含了不同的模块，若存在对 base board 的调用，需要在相应位置添加对 new board 的使用。进入/source/unisoc/connectivity/wcn/libbt，添加如下内容：

```
diff --git a/Makefile b/Makefile
index 1e6fe5296bd0..9922457b972a 100644
--- a/Makefile
+++ b/Makefile
@@ -103,10 +103,12 @@ endif
ifeq ($(SPRD_WCNBT_CHISET),uis7885-2h10)
    board = marlin3_lite
    CFLAGS += -DMARLIN3_LITE
endif

+ifeq ($(SPRD_WCNBT_CHISET),uis7870-test)
+    board = marlin3_lite
+    CFLAGS += -DMARLIN3_LITE
+endif
+
```

----结束

4.12.4 新建 Kernel

4.12.4.1 kernel 5.4/kernel 5.15

可以在不更改 Kernel 仓库的前提下进行修改，也可以直接在 Kernel 模块下更改代码合入仓库。

不更改 Kernel 仓库的修改方式

- 步骤 1 arch/arm64/boot/dts/sprd/Makefile 目录下，Makefile 中增加 new board 的 dts 信息和 dtbo 信息，具体添加内容如下（可以根据具体的 base board 进行对应的添加）：

```
diff --git a/arch/arm64/boot/dts/sprd/Makefile b/arch/arm64/boot/dts/sprd/Makefile
index 1e6fe5296bd0..9922457b972a 100644
--- a/arch/arm64/boot/dts/sprd/Makefile
+++ b/arch/arm64/boot/dts/sprd/Makefile
@@ -50,10 +50,12 @@ endif
ifeq ($(BSP_BUILD_FAMILY),qogim6pro)
    dtbo-$(CONFIG_ARCH_SPRD) += \
        uis7885-1h10-overlay.dtbo \
```

```

- uis7885-2h10-overlay.dtbo
+ uis7885-2h10-overlay.dtbo \
+ uis7870-test-overlay.dtbo

uis7885-1h10-overlay.dtbo-base := uis7885.dtb
uis7885-2h10-overlay.dtbo-base := uis7885.dtb
+uis7870-test-overlay.dtbo-base := uis7870.dtb
endif

endif ##BSP_BUILD_ANDROID_OS
@@ -68,7 +70,8 @@ dtb-$(CONFIG_ARCH_SPRD) += \
    ums512.dtb \
    ums312.dtb \
    ums9230.dtb \
- uis7885.dtb
+ uis7885.dtb \
+ uis7870.dtb

dtbo-$(CONFIG_ARCH_SPRD) += \
    sp9863a-1h10-overlay.dtbo \
@@ -83,7 +86,8 @@ dtbo-$(CONFIG_ARCH_SPRD) += \
    ums9230-4h10_go-overlay.dtbo \
    ums9230-4h10-overlay.dtbo \
    uis7885-1h10-overlay.dtbo \
- uis7885-2h10-overlay.dtbo
+ uis7885-2h10-overlay.dtbo \
+ uis7870-test-overlay.dtbo

endif ##BSP_BUILD_DT_OVERLAY

```

步骤 2 完成步骤 1 相关修改后，需要生成 patch，放到 layer/meta-unisoc/recipe-kernel/linux-unisoc-5.4/uis7870 目录下。

#通过 git diff 查看修改是否存在，然后执行如下命令即可生成对应的 patch

```
git diff
git diff Makefile > add-dtbo-uis7870.patch
```

将对应的 patch 放到 layer/meta-unisoc/recipe-kernel/linux-unisoc-5.4/uis7870 目录下：

```
cp add-dtbo-uis7870.patch home/yocto/layer/meta-unisoc/recipe-kernel/linux-unisoc-5.4/uis7870
```

步骤 3 layer/meta-unisoc/conf/machine/uis7870-test.conf 文件中添加生成的 patch，具体的位置和变量如

下：

```
KERNEL_MACHINE_PATCH = "file://uis7870-test.cfg \
    file://uis7870-test-audio.cfg \
    file://0001-Bug-1982891-change-androidboot-to-sprdboot-in-n6pro.patch \
    file://0002-fix-user-compile-error.patch \
    file://0003-enable-trusty-vendor-hook.patch \
    file://add-dtbo-uis7870.patch \
"
```

----结束

更改 Kernel 仓库的修改方式

直接在 Kernel 模块下更改代码合入仓库，具体修改方法参考[不更改 Kernel 仓库的修改方式](#)的步骤 1。

4.12.4.2 kernel 6.1

步骤 1 以 uis8360-1h10-nand board 为例，在 arch/arm/boot/dts/Makefile 目录下，Makefile 中增加 new board 的 DTS 信息，具体添加内容如下所示：

```
dtb-$(CONFIG_ARCH_SPRD) += \
    uis8910dm-2h10-nand.dtb \
    recovery-uis8910dm-2h10-nand.dtb \
    uis8360-1h10-nand.dtb \
    recovery-uis8360-1h10-nand.dtb
```

说明

可以根据具体的 base board 进行对应的添加。

然后在 arch/arm/boot/dts 目录下查找 base board 的相关目录，然后每个目录进行 copy，并且对应的文件名修改为 new board 的名称，文件内 base board 相关的关键字进行替换。

步骤 2 在 arch/arm/configs 目录下查找 base board 相关的文件，进行 copy。如：

```
cp sprd_recovery_uis8910dm_1h10_nand_defconfig sprd_recovery_uis8360_1h10_nand_defconfig
cp sprd_uis8910dm_1h10_nand_defconfig sprd_uis8360_1h10_nand_defconfig
```

步骤 3 查找如下目录 drivers/unisoc_platform/sprd_disp/，是否存在 base board 相关目录，然后进行 copy。

```
cp -rf uis8910dm uis8360
```

步骤 4 查找如下目录 include/dt-bindings，是否存在 base board 相关文件，进行 copy，如：

```
cd include/dt-bindings
find . -name "*uis8910dm*"
include/dt-bindings/clock/sprd,uis8360-clk.h
include/dt-bindings/reset/sprd,uis8360-reset.h
include/dt-bindings/soc/sprd,uis8360-mask.h
include/dt-bindings/soc/sprd,uis8360-regs.h
#然后将查找到的文件依次 copy
```

----结束

4.12.5 修改 source/unisoc 目录下 base board 信息

完成新建 board 操作后，为避免 source/unisoc 目录下存在对 base board 的相关配置，可以在 source/unisoc 目录下查找关键字进行修改。

```
find . -name "*uis7885_2h10"
find . -name "*uis7885-2h10"
grep -inr "uis7885_2h10"
grep -inr "uis7885-2h10"
```

- 针对 find 查找出的文件，复制并修改文件名和文件内关键字。
- 针对 grep 查找出的内容，添加对应文件中关键字的内容。

4.13 dts/kernel cfg 相关说明

4.13.1 添加对多 dts 的支持

1. kernel multi dts (layer/meta-unisoc/README)

- UNISOC_SUPPORT_MULTI_DEVICETREE：启用标志，默认禁用。
- UNISOC_MULTI_DEVICETREE_SUFFIX：基于主设备树生成其他设备树名称。

例如：

- 主设备树："uis8520-1h31-nand.dtb"，UNISOC_MULTI_DEVICETREE_SUFFIX = "cp56m cp64m"
- 其他设备树名称：uis8520-1h31-nand-cp56m.dtb；uis8520-1h31-nand-cp64m.dtb
- UNISOC_MULTI_DEVICETREE_OVERLAY_FRAGMENTS：用于覆盖的片段，将生成额外的 dts，并根据主 dtb 替换片段。

例如：

```
UNISOC_MULTI_DEVICETREE_SUFFIX = "cp56m cp64m"
```

```
UNISOC_MULTI_DEVICETREE_OVERLAY_FRAGMENTS = "uis8520-1h31-nand -reserved-memory.dtsi"
```

uis8520-1h31-nand-cp56m.dts 中的内容为: `"#include " uis8520-1h31-nand -reserved-memory.dtsi"`

然后 dts 修改后的内容为:

```
"#include " uis8520-1h31-nand -reserved-memory-cp56m.dtsi"
```

uis8520-1h31-nand-cp64m.dts 中的内容为: `"#include " uis8520-1h31-nand -reserved-memory.dtsi"`

然后 dts 修改后的内容为:

```
"#include " uis8520-1h31-nand -reserved-memory-cp64m.dtsi"
```

- UNISOC_ADDITIONAL_DEVICETREE: 根据如上三点, 将在 UNISOC_ADDITIONAL_DEVICETREE 中添加额外的 dtb 配置, 并将其添加到 KERNEL_DEVICETREE 中。

进入到 layer/meta-unisoc 目录下, 修改如下文件:

```
vi conf/machine/uis8520-1h31-nand.conf
```

```
UNISOC_SUPPORT_MULTI_DEVICETREE = '1'
```

```
NISOC_MULTI_DEVICETREE_SUFFIX = "layoutcp280m"
```

```
UNISOC_MULTI_DEVICETREE_OVERLAY_FRAGMENTS = "uis8520-tiny.dtsi uis8520-1h31-nand.dtsi"
```

```
KERNEL_DEVICETREE:append:uboot22 = "${UNISOC_ADDITIONAL_DEVICETREE} "
```

```
KERNEL_RECOVERY_DEVICETREE:append:uboot22 = "${UNISOC_ADDITIONAL_DEVICETREE} "
```

2. teeconf 支持多 dts

ADDITIONAL_SML_TARGET_DTB_PATH: 在 ADDITIONAL_SML_TARGET_DTB_PATH 中设置额外的 DTS 路径。

```
cd recipes-security/trustzone/configs/uis8520-1h31-nand.inc
```

```
ADDITIONAL_SML_TARGET_DTB_PATH:uboot22 = "${OEROOT}/source/kernel/kernel5.15/arch/arm64/boot/dts/sprd/reserved-memory/uis8520-tiny-layoutcp280m.dtsi"
```

3. 修改打包文件

```
cd prebuilds/pac_config
vi uis8520-1h31-nand-uboot22.ini
BOOT=1@./out/target/product/uis8520-1h31-nand/kernel-uis8520-1h31-nand-layoutcp280m-sign.img
Recovery=1@./out/target/product/uis8520-1h31-nand/recovery-kernel-initramfs-recovery-uis8520-1h31-
nand-layoutcp280m-sign.img
TEECFG=1@./out/target/product/uis8520-1h31-nand/teecfg-uis8520-tiny-layoutcp280m/teecfg-sign.bin
#如上打包的三个分区 bin 需要修改
```

需要注意的是，如果不修改 ini 打包文件，即使增加了前面的配置，多 dts 的功能也不会生效。

4.13.2 dts 文件选择

不同 kernel 版本的源码在 source/kernel 目录下，里面有不同工程的 dts 文件，如果修改了对应的 dts 文件，编译后发现修改没有生效，可以通过如下方法进行排查。

在 layer/meta-unisoc 目录下对 dts 文件进行查找，可以看到多个 dts 文件，如下图所示。其中是针对不同工程的特定功能进行适配的 dts 文件，layer/meta-unisoc 目录下的 dts 文件会对 source/kernel 下面的 dts 文件进行覆盖，所以修改 source/kernel 目录下的 dts 文件在编译后，修改并不会生效。

针对不同的工程，如何选择对应的 dts 文件，说明如下。

```
cd layers/meta-unisoc
find . -name "*.dts"
```

1. 如果路径中包含 **distro**，且后面的路径为 unisoc-tiny，则此目录下的 dts 对 DISTRO=unisoc-tiny 的工程生效。
./recipes-kernel/linux/linux-unisoc-5.15/uis8520/**distro**/unisoc-tiny/uis8520-1h30-nand.dts
2. 如果路径中包含 **product**，且后面的路径为 pcieep，则此目录下的 dts 对 PRODUCT= pcieep 的工程生效。
./recipes-kernel/linux/linux-unisoc-5.15/uis8520/**product**/pcieep/uis8520-1h10-nand.dts
3. 如下路径的 dts 文件会对 uis8520 工程都会生效。
./recipes-kernel/linux/linux-unisoc-5.15/uis8520/uis8520-1h30-nand.dts

📖说明

- 如上三个路径的覆盖优先级为：第一个和第二个不会互相覆盖且不会发生冲突，但是会覆盖第三个路径下的 dts 文件。
- 其中 distro/product 的含义参考 [4.6 生成 pac 包](#)。

4.13.3 kernel cfg 说明

- 如需查看当前 kernel 所有的 config 的信息，可以在编译完 kernel 后查看如下.config 文件。

```
/ yocto/build-unisoc-tiny/tmp-unisoc_tiny-musl/work/uis8360_1h10_nand-unisoc-linux-musleabi/linux-unisoc-6.1/6.1-r0/linux-unisoc-6.1-6.1/.config
```

- 如需新增不同功能的 `cfg`，可以进行如下配置。
以 `uis8360` 工程进行举例：
 - 新建文件 `kernel-module-adc.cfg`，可以在此文件中配置有关 **adc** 功能的 `CONFIG`。
 - 将新建的 `kernel-module-adc.cfg` 文件存放在路径 `layers/meta-unisoc/recipes-kernel/linux/linux-unisoc-6.1/uis8360/kernel-config/` 下。
 - 配置 `layers/meta-unisoc/conf/machine/uis8360-1h10-nand.conf`，将 **adc** 添加到变量 `UNISOC_KERNEL_FEATURES` 下，则对应的 `kernel-module-adc.cfg` 文件中配置的 `config` 也会生效。

4.14 其他操作及说明

4.14.1 加快编译速度

代码调试时可以使用以下两种方法加快编译速度。

- 使用 `pac.sh` 脚本打包。
AP 代码未修改，Modem 代码作了修改并编译生成新的 Modem 镜像，在 `prebuilts/scripts` 目录下执行 `pac.sh` 可快速生成 PAC 包用于代码调试。
`pac.sh` 脚本运行示例：

```
cd prebuilts/scripts
./pac.sh uis7885-2h10+wayland+wayland+sec userdebug
```

说明

必须使用 `build.sh` 完成过一次整个工程的编译并生成所有的分区镜像后，才能使用 `pac.sh` 进行打包。

- 编译生成某个镜像。
以编译生成 `rootfs` 镜像为例，假如修改了 `atrouter` 应用模块的代码和配置文件（如 `system.ini`）。使用以下命令编译 `atrouter` 模块并生成 `rootfs` 镜像。

```
bitbake -c cleansstate atrouter && bitbake -C compile atrouter
bitbake -c cleansstate unisoc-wayland-image && bitbake -C compile unisoc-wayland-image
```

生成新的 `rootfs` 镜像后，可以使用 `researchdownload` 工具替换旧的 `rootfs` 镜像。也可以使用上述的 `pac.sh` 命令直接打成 `pac` 包进行烧录，相比使用 `build.sh` 编译整个工程，减少了编译时间。

4.14.2 查看版本号

4.14.2.1 推送版本号

查看 `${ROOTDIR}/vendor/sprd/release/IDH/CUS_Release_Notes` 目录获取 AP 和 CP 的推送版本号。

4.14.2.2 编译版本号

查看/etc/os-release 获取编译版本号。编译版本号示例如下：

```
root@ uis7885-2h10:/ # cat /etc/os-release
ID=unisoc-wayland
NAME="unisoc-wayland-distro"
VERSION="uis7885-2h10+unisoc-wayland-2.0+W23.13.1-06.30.19+userdebug+native (dunfell)"
VERSION_ID=uis7885-2h10-unisoc-wayland-2.0-w23.13.1-06.30.19-userdebug-native
PRETTY_NAME="unisoc-wayland-distro uis7885-2h10+unisoc-wayland-2.0+W23.13.1-06.30.19+userdebug+native (dunfell)"
DISTRO_CODENAME="dunfell"
```

说明

编译版本号随着编译时间发生变化。

4.14.3 版本区分

同一 bb 文件中如果需实现不同版本（user 或 userdebug 或 eng），安装不同配置文件，可以在 do_install 函数中加入相关的判断语句来区分。各版本的区分如下表所示。

变量名称	user 版本	userdebug 版本	eng 版本
USERDEBUG	user	userdebug	userdebug
ENG_IT	user	userdebug	eng

例如，使用 USERDEBUG 变量来区分 user 或 userdebug 版本，条件判断语句格式：

```
do_install ()
{
    if [ ${USERDEBUG} = "userdebug" ]; then
        install ....
        install ....
    else
        install ....
        install ....
    fi
}
```

4.14.4 打开/关闭 MACHINE 中 Kernel 的某个选项

打开或关闭 MACHINE 中 Kernel 的某个功能选项的操作如下，如果清楚需要打开的选项，可直接跳转至 [步骤 5](#)。

步骤 1 编译 Kernel。

```
bitbake -c cleansstate linux-unisoc-5.4 && bitbake -C compile linux-unisoc-5.4
```

步骤 1 进入内核根目录。

```
cd Rootdir/source/kernel/kernel5.4
```

步骤 2 启动 menuconfig。

```
cp oe-workdir/linux-unisoc-*/.config .config
make ARCH=arm64 menuconfig
```

目录 linux-unisoc-* 包含 Kernel 编译成功后的所有文件，目录具体名字由实际情况决定。根据实际需求打开某个配置项，然后保存为 .config.1。

说明

这里 ARCH 选 arm64 还是 arm 取决于配置的 Kernel 是 64 位还是 32 位。

步骤 3 确定依赖项。

```
diff .config.1 .config | grep "<"
```

根据提示确定需要打开的配置项。

步骤 4 删除 menuconfig 产生的文件。

```
make mrproper
```

说明

必须执行这个步骤，否则编译 Kernel 时会出现错误。

步骤 5 添加配置项到 cfg 文件中。

在文件 MACHINE.cfg（例如 uis7885-2h10.cfg）的尾部添加依赖项目，格式如下：

```
CONFIG_XXX=y //使能某个功能
```

或者

```
# CONFIG_XXX is not set //关闭某个功能
```

步骤 6 再次编译 Kernel。

```
bitbake -c cleansstate linux-unisoc-5.4 && bitbake -C compile linux-unisoc-5.4
```

编译成功后，检查 .config(oe-workdir/linux-unisoc-*/.config) 是否包含打开或者关闭的选项。

----结束

4.14.5 修改分区大小

使用 img_create.sh 制作 ubuntu rootfs，传入参数：

```
BLOCK_SIZE="1M"
```

```
IMG_SIZE="8192"
```

制作的 rootfs 是 8GB，system 分区预留大小不够，需要修改分区大小。

layers/meta-unisoc/conf/machine/include/xml 目录下，找到对应的 machine.xml 进行分区大小的修改，如下图：

```
diff --git a/conf/machine/include/xml/ums9620-2h10.xml b/conf/machine/include/xml/ums9620-2h10.xml
index b28e3068..c9eab7c2 100755
--- a/conf/machine/include/xml/ums9620-2h10.xml
+++ b/conf/machine/include/xml/ums9620-2h10.xml
@@ -2339,7 +2339,7 @@
     <Partition id="boot" size="45"/>
     <Partition id="dtb" size="8"/>
     <Partition id="dtbo" size="8"/>
-    <Partition id="system" size="4100"/>
+    <Partition id="system" size="20000"/>
     <Partition id="sysdumpdb" size="10"/>
     <Partition id="userdata" size="0xFFFFFFFF"/>
 </Partitions>
```

4.15 常见问题处理

4.15.1 新基线版本合入出错

问题现象

合入新基线版本后，编译出现如下错误提示：

```
bb.data_smart.ExpansionError: Failure expanding variable do_compile[file-checksums], expression was
${@srcree_hash_files(d)} which triggered exception CalledProcessError: Command '['git', 'write-tree']' returned
non-zero exit status
```

问题排查

- 检查是否修改 layers/meta-unisoc/conf/machine/include/unisoc-extern.inc 文件，或合入时是否有冲突。
- 检查是否存在未合入文件。如存在，编译时会出现以下错误提示：

```
fatal: git-write-tree: error building trees
```

4.15.2 非联网环境编译出现 fetch 错误

问题现象

非联网环境编译出现 fetch 错误。

问题排查

确认拉取代码的命令是否正确，注意区分拉取 Android 版本和 yocto 版本代码的命令。

1. 确认导致下载问题的 bb 文件。
2. 查看导致下载问题 bb 文件内容，确认下载的 tar 包名称，在\${ROOTDIR}/source/tarballs 目录下搜索该 tar 包。如搜索结果如下两种情形，联系展锐 FAE 重新推送该 tar 包。
 - 没有找到 tar 包。
 - 找到 tar 包，但 tar 包的校验和与展锐生成的校验和不一致。


```
- $(verbose) if test -x $(XMLEINT) && test -f $(xmldtd); then \
-     $(XMLEINT) --noout --path $(dir $(xmldtd)) --dtdvalid $(xmldtd) $@ ;\
- fi

xml: $(polxml)
```

步骤 3 把 Makefile.patch 分别放到如下两个 refpolicy 目录下。

```
/branch/yocto2.0-c/layers/meta-selinux$ git status
Not currently on any branch.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   recipes-security/refpolicy/refpolicy-mcs_git.bb

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        recipes-security/refpolicy/refpolicy-2.20190201/Makefile.patch
        recipes-security/refpolicy/refpolicy_git/Makefile.patch

no changes added to commit (use "git add" and/or "git commit -a")
```

步骤 4 单独编译 refpolicy-mcs 模块。

```
bitbake refpolicy-mcs
```

如果没有问题，就可以执行后续操作，如生成 pac 包。

----结束

4.15.4 pac 包编译失败

问题现象

pac 包编译失败，出现如下报错：

command output:

```
Invalid parameters, file
[/home/data/jenkins_node2/workspace/linux_base//prebuilts/unisoc_bin/5g_modem_v2_22c/uis7885/QogirN6Pro_PS_buildddir/QogirN6Pro_PS_nvitem.bin] don't exist
```

问题排查

QogirN6Pro_PS_nvitem.bin 的路径没有对应上。进入 prebuilts/pac_configs，修改 uis7885-2h10.ini（对应 machine.ini），将其中的 PS nv bin 修改成正确的路径，修改如下：

```

steven.lin@bjand12:~/yocto/qlp2.0_customer/prebuilts/pac_configs$ git diff
diff --git a/ums9620-1h10.ini b/ums9620-1h10.ini
index e0142f3..e856cc3 100755
--- a/ums9620-1h10.ini
+++ b/ums9620-1h10.ini
@@ -32,11 +32,11 @@ ums9620-1h10+wayland+wayland+nsec-userdebug-native=
CONFIGFILE=./out/target/product/ums9620-1h10/ums9620-1h10.xml
FDL1=1@./out/target/product/ums9620-1h10/fdl1-sign.bin
FDL2=1@./out/target/product/ums9620-1h10/fdl2-sign.bin
-NV_NR=1@./prebuilts/unisoc_bin/5g_modem_v2_22c/ums9620/QogirN6Pro_PS_buildddir/QogirN6Pro_PS_nvitem.bin
+N NV_NR=1@./prebuilts/unisoc_bin/5g_modem_v2_22c/ums9620/QogirN6Pro_PS_Customer_buildddir/QogirN6Pro_PS_Customer_nvitem.bin
PhaseCheck=1@
EraseUBOOT=1@
Modem_NR=1@./prebuilts/unisoc_bin/5g_modem_v2_22c/ums9620/QogirN6Pro_PS_buildddir/SC9600_QogirN6Pro_PS_modem.dat
-Modem_NR_DELTANV=1@./prebuilts/unisoc_bin/5g_modem_v2_22c/ums9620/QogirN6Pro_PS_buildddir/QogirN6Pro_PS_deltanv.bin
+Modem_NR_DELTANV=1@./prebuilts/unisoc_bin/5g_modem_v2_22c/ums9620/QogirN6Pro_PS_Customer_buildddir/QogirN6Pro_PS_Customer_deltanv.bin
Modem_NR_PHY=1@./prebuilts/unisoc_bin/5g_modem_v2_22c/ums9620/QogirN6Pro_PHY_buildddir/SC9600_QogirN6Pro_PHY_modem.bin
DFS=1@./prebuilts/unisoc_bin/sp_base_cm4/9620/QogirN6Pro_SP_buildddir/QogirN6Pro_SP.bin
Modem_GNSS=1@./prebuilts/unisoc_bin/gnss_20b_new/marlin3lite/marlin3lite_gnss_cm4_buildddir/gnssmodem.bin

```

5 新增分区

5.1 构建新分区

需要完成新分区的构建，创建对应的 `partion.bb` 文件。相应文件存放于 `yocto/layers/meta-unisoc/recipes-core/images/partion.bb`。bb 文件的创建可以参考已有的分区 bb 文件，例如 `prodnv.bb`。

```
SUMMARY = "partion image"
DESCRIPTION = "null partion \
"
LICENSE = "CLOSED"

IMAGE_INSTALL = ""
IMAGE_LINGUAS = ""
PACKAGE_INSTALL = ""

ROOTFS_POSTPROCESS_COMMAND_remove += "
zap_empty_rootfs_password rootfs_update_timestamp read_only_rootfs_hoook; write_image_test_data
empty_var_volatile ; update_timestamp;rootfs_reproducible; ; "
SORT_PASSWD_POSTPROCESS_COMMAND = ""
ROOTFS_POSTPROCESS_COMMAND_remove = " set_user_group; "

inherit image
do_build[noexec] = "1"
do_build[depends] = ""
KERNELDEPMODDEPEND = ""
LDCONFIGDEPEND = ""

do_image() {
    rm -rf ${IMAGE_ROOTFS}/*

    # 接口
    # 可以用如下命令拷贝数据到新的分区中
    #install -m 0777 ${OEROOT}/source/app/hello ${IMAGE_ROOTFS}
```

```

}

# This variable is triggered to check if sysvinit must be overwritten by a single rcS
export SYSVINIT = "no"

PARTION_MKUBIFS_ARGS ?= ""
PARTION_FSTYPES ?= ""
PARTION_SIZE ?= ""

IMAGE_ROOTFS_ALIGNMENT = "2048"

MKUBIFS_ARGS = "${PARTION_MKUBIFS_ARGS}"

IMAGE_FSTYPES = "${PARTION_FSTYPES}"
EXTRA_IMAGEDEPENDS = ""
IMAGE_POSTPROCESS_COMMAND = ""

EXTRA_USERS_PARAMS_remove = "${EXTRA_USERS_PARAMS}"
EXTRA_USERS_PARAMS_remove += "${EXTRA_USERS_PARAMS_append}"

IMAGE_ROOTFS_SIZE = "${PARTION_SIZE}"

```

说明

黄色部分为区别于别的分区的信息。

在对应的 board.conf 文件中增加分区相关信息。其中，PARTION_MKUBIFS_ARGS 针对 fstype 为 ubifs 的情况，如果新增分区类型为 ext4，则不需要设置该变量。

以 uis7885-2h10 为例，新增分区类型为 ext4，分区大小为 1.5GB。在 uis7885-2h10.conf 中增加如下信息，其中 PARTION_SIZE 的单位为 KB：

```

PARTION_FSTYPES = "ext4"
PARTION_SIZE = "1572864"

```

此时可以单独完成新分区的构建。

```

MACHINE= uis7885-2h10 DISTRO=unisoc-wayland source setup-environment
bitbake partion

```

编译完成后会生成相应的新分区，位于如下目录：yocto/build-unisoc-wayland/tmp-unisoc_wayland-glibc/deploy/images/uis7885-2h10。

具体文件名：partion-\${MACHINE}-\${timedate}.rootfs.ext4

5.2 编译 pac 包时完成新增分区

在单独构建新分区通过后，需要在 build.sh 等脚本增加构建新分区的内容，实现生成 pac 包的同时，构建出新分区。

1. 在 build.sh 文件中，对应的 MACHINE 下，新增高亮部分代码，完成新分区的编译。

```
...
building_image $LINENO prodnv
echo "Build prodnv: $(expr $(date +%s) - $btime)" >> ${timedir}/${timename} && btime=`date +%s`

building_image $LINENO userdata
echo "Build userdata: $(expr $(date +%s) - $btime)" >> ${timedir}/${timename} && btime=`date +%s`

building_image $LINENO partion
echo "Build partion: $(expr $(date +%s) - $btime)" >> ${timedir}/${timename} && btime=`date +%s`

building_image $LINENO swupdate-image
echo "Build swupdate-image: $(expr $(date +%s) - $btime)" >> ${timedir}/${timename} &&
btime=`date +%s`

building_image $LINENO recovery-img
echo "Build recovery-img: $(expr $(date +%s) - $btime)" >> ${timedir}/${timename} && btime=`date +%s`
...
```

2. 在 pac.sh 文件中，新增如下内容，完成新分区的打包及重命名工作。例如，将新生成的 partion 分区重命名为 aisdk.img:

```
...
if [ -e $IMAGEPATH/partion-${MACHINENAME}.ext4.gz ] || [ -e $IMAGEPATH/partion-
${MACHINENAME}.ubifs ]; then
    if [ -e $IMAGEPATH/partion-${MACHINENAME}.ubifs ]; then
        cp $IMAGEPATH/partion-${MACHINENAME}.ubifs ${OUTPUT_PATH}/aisdk.img
    fi

    if [ -e $IMAGEPATH/partion-${MACHINENAME}.ext4.gz ]; then
        cp $IMAGEPATH/partion-${MACHINENAME}.ext4.gz ${OUTPUT_PATH}/aisdk.ext4.gz
        cd ${OUTPUT_PATH} && gunzip aisdk.ext4.gz && mv aisdk.ext4 aisdk.img
    fi
fi
...
```

完成上述修改后，在编译 pac 过程能够顺利生成指定名字的分区 aisdk.img。

3. 修改对应的 board.xml 文件，新增 aisdk 分区，并添加分区大小（单位 MB）。

说明

新增分区在 system 分区之后，userdata 分区之前。如果在 system 之前，需要修改 kernel dts，明确 root=xxx 的新信息。

uis7885-2h10.conf 文件中，PARTION_SIZE = "1572864"（单位 KB），可以确定对应的分区大小为 1536MB。在 layer/meta-unisoc/conf/machine/include/xml/uis7885-2h10.xml 文件中新增如下高亮部分代码：

```
...
    <Partition id="metadata" size="16"/>
    <Partition id="vbmeta_system" size="1"/>
    <Partition id="vbmeta_vendor" size="1"/>
    <Partition id="aisdk" size="1536"/>
    <Partition id="userdata" size="0xFFFFFFFF"/>
  </Partitions>
  <ProductComment/>
</Product>
</ProductList>
...

  <File>
    <ID>Partion</ID>
    <IDAlias> Partion </IDAlias>
    <Type>YAFFS_IMG2</Type>
    <Block id=" partion ">
      <Base>0x0</Base>
      <Size>0x0</Size>
    </Block>
    <Flag>1</Flag>
    <CheckFlag>0</CheckFlag>
    <Description> Partion image file</Description>
  </File>
  <File>
    <ID>BootLogo</ID>
    <IDAlias>BootLogo</IDAlias>
    <Type>CODE2</Type>
    <Block id="logo">
      <Base>0x0</Base>
      <Size>0x0</Size>
    </Block>
```

```
<Flag>1</Flag>
<CheckFlag>0</CheckFlag>
<Description>BootLoader boot logo</Description>
</File>
...
```

5.3 自动挂载分区

完成新分区的构建后，需要将相应的分区挂载到 userdata 目录下。

1. 在 board.conf 文件中新增 partition-init feature。此处不使用 fstab 自动挂载，而是通过 partition-init feature 选择是否挂载新分区。

在 layer/meta-unisoc/conf/machine/uis7885-2h10.conf 文件中新增如下高亮部分代码：

```
...
MACHINE_FEATURES += "${@bb.utils.contains('USERDEBUG','userdebug','networkutils','d')}}"
MACHINE_FEATURES += " partition-init"
...
```

2. 通过 packagegroup 将对应的 feature 和模块 bb 文件相关联，实现 image 的定制。

在 yocto/layer/meta-unisoc/recipes-core/packagegroups/packagegroup-unisoc-wayland.bb 文件中，新增如下高亮部分代码：

```
.....
${@bb.utils.contains('MACHINE_FEATURES','vdsp','packagegroup-unisoc-base-vdsp','d')} \
${@bb.utils.contains('MACHINE_FEATURES','partition-init','packagegroup-unisoc-base-partition-init',
",d")} \
.....
```

在 yocto/layer/meta-unisoc/recipes-core/packagegroups/packagegroup-unisoc-base.bb 文件中，新增如下高亮部分代码：

```
.....
${@bb.utils.contains('MACHINE_FEATURES','userdata-resize2fs','packagegroup-unisoc-base-
userdata-resize2fs','d')} \
${@bb.utils.contains('MACHINE_FEATURES','partition-init','packagegroup-unisoc-base-partition-init',
",d")} \
.....
SUMMARY_packagegroup-unisoc-base-userdata-resize2fs = "auto resize userdata space "
RDEPENDS_packagegroup-unisoc-base-userdata-resize2fs = " \
resize2fs-script \
"
SUMMARY_packagegroup-unisoc-base-partition-init = "mount partition and run app "
RDEPENDS_packagegroup-unisoc-base-partition-init = " \
partition-mount \
```

```
"
.....
```

3. 完成 partion-mount 功能的实现。

新建目录 yocto/layer/meta-unisoc/recipes-devtools/partion， partion 的目录结构如下。

```
files
├── COPYING
├── partion-mount.sh
└── partion-mount.bb
```

- partion-mount.bb 为对应的 bb 文件，内容如下：

```
DESCRIPTION = "mount partion"
HOMEPAGE = "http://www.unisoc.com/"
LICENSE = "UNISOC"
SECTION = "bin"
LIC_FILES_CHKSUM =
"file://${THISDIR}/files/COPYING;md5=dce8ff4bf27149d72c9ef7cdbf0c346a"
PV = "0.1"
PR = "r0"
inherit update-rc.d

PROVIDES = "partion-mount"
SRC_URI = " \
    file://partion-mount.sh \
"

INITSCRIPT_NAME = "partion-mount"
INITSCRIPT_PARAMS = "defaults 30"

#新分区的分区号根据对应的 xml 文件中的分区序号来确定，也就是从第一个分区开始计数。
PARTION_NUMBER = "39"

do_compile() {
}

do_install() {

    install -d ${D}${sysconfdir}/init.d
    install -m 0755 ${WORKDIR}/partion-mount.sh ${D}/${sysconfdir}/init.d/partion-mount
```

```
sed -i "s/NUMBER/${PARTION_NUMBER}/" ${D}/${sysconfdir}/init.d/partion-mount

install -d ${D}/${sbindir}
install -m 0755 ${WORKDIR}/partion-mount.sh ${D}/${sbindir}/partion-mount.sh

sed -i "s/NUMBER/${PARTION_NUMBER}/" ${D}/${sbindir}/partion-mount.sh
}
```

TARGET_CC_ARCH += "\${LDFLAGS}"

主要工作是将对应的 `partion-mount.sh` 安装到生成的 `image` 的 `init.d` 目录中作为服务脚本，实现开机过程的启动。将对应的 `partion-mount.sh` 安装到生成的 `image` 的 `sbin` 目录下，可以直接进行脚本的调用。

具体 `partion-mount.sh` 脚本的内容可以根据实际需要进行编写，但需要完成对新分区的挂载工作，也就是将挂载和加载程序等操作结合起来，方便进一步的维护。`partion-mount.sh` 的内容如下：

```
#!/bin/sh

df -h | grep userdata

if [ $? = 0 ]; then
    mkdir -p /mnt/userdata/aisdk
    mount -t ext4 -o ro /dev/mmcblk0pNUMBER /mnt/userdata/aisdk

    #run customer app eg 这里可以加载用户的脚本或者应用程序
fi
```

- COPYING 文件为 `partion-mount.bb` 的 `LIC_FILES_CHKSUM`。可以从其他模块复制，COPYING 相应内容如下：

Copyright (c) 2016-2020 by UNISOC Technology(Shanghai), Inc. ALL RIGHTS RESERVED.

These coded instructions, statements, and computer programs are the copyrighted works and confidential proprietary information of UNISOC Technology(Shanghai), Inc. They may not be modified, copied, reproduced, distributed, or disclosed to third parties in any manner, medium, or form, in whole or in part, without the prior written consent of UNISOC Technology(Shanghai), Inc

执行如下命令，确定 COPYING 文件的 md5sum 值，添加到 `partion-mount.bb` 文件中对应位置。

```
#在 files 目录下
$ md5sum COPYING
```

```
d41d8cd98f00b204e9800998ecf8427e  COPYING

# 对应的 partition-mount.bb 中，修改 LIC_FILES_CHKSUM 的值。
.....
LIC_FILES_CHKSUM = "file://${THISDIR}/files/COPYING;md5=
d41d8cd98f00b204e9800998ecf8427e"
.....
```

完成上述操作后，可以通过 conf 文件中的 machine-feature 实现新增分区的自动挂载和分区程序的加载。

5.4 新分区增加到 researchdownload 工具

将新增的分区自动加入到 researchdownload 工具中，修改对应的 board.ini 文件。

在 yocto/prebuilts/pac_config/uis7885-2h10.ini 文件中，新增如下高亮部分代码：

```
[uis7885_board]
...
UserData=1@./out/target/product/uis7885-2h10/userdata.img
Aisdk=1@./out/target/product/uis7885-2h10/aisdk.img
BootLogo=1@./out/target/product/uis7885-2h10/unisoc_FHD_1080_1920.bmp
...
[uis7885_sec_board]
...
UserData=1@./out/target/product/uis7885-2h10/userdata.img
Aisdk=1@./out/target/product/uis7885-2h10/aisdk.img
BootLogo=1@./out/target/product/uis7885-2h10/unisoc_FHD_1080_1920.bmp
...
[uis7885_ubuntu_sec_board]
...
UserData=1@./out/target/product/uis7885-2h10/userdata.img
Aisdk=1@./out/target/product/uis7885-2h10/aisdk.img
BootLogo=1@./out/target/product/uis7885-2h10/unisoc_FHD_1080_1920.bmp
```

6 编译后新增目录

完成 GLP3.0 项目编译后会新增一些目录，用于存放相应的编译结果等信息。

利用 build.sh 直接编译整个项目，会生成对应的 pac 包，此时项目主目录如图 6-1 所示。

图6-1 编译后工程目录

```
~/yocto_pac$ tree -L 1 .
.
├── bitbake
├── build.sh -> prebuilts/scripts/build.sh
├── build-unisoc-wayland
├── layers
├── out
├── prebuilts
├── setup-environment -> prebuilts/scripts/setup-environment
├── source
└── sstate-cache
```

编译后新增目录为 build-unisoc-wayland、out 及 sstate-cache，其中需重点关注 build-unisoc-wayland 目录。
详细介绍参考文档《Yocto 目录结构介绍》。

7 只读文件系统配置及使用

7.1 配置方法

Yocto 编译生成 rootfs 时，可以指定 image feature 生成只读的 rootfs。在 image 的 bb 文件中指定如下：

```
EXTRA_IMAGE_FEATURES += "read-only-rootfs"
```

7.2 使用注意事项

当生成只读 rootfs 时，只有 /var、/home、/mnt/data、/mnt/userdata、/etc/productinfo 目录可写，这几个可写目录通过挂载 data 分区和 product 分区实现可写。恢复出厂设置时，会清除 /mnt/data/ 目录下的内容，所以应用在选择保存数据的位置时需要谨慎考虑。

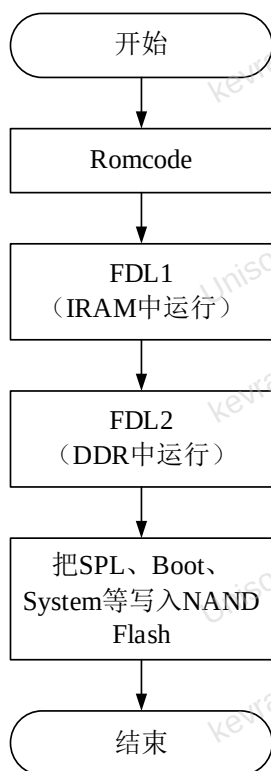
```
root@ums9620-2h10:/mnt# mount
/dev/mmcblk0p34 on / type ext4 (ro,relatime,seclabel,commit=600)
devtmpfs on /dev type devtmpfs (rw,relatime,seclabel,size=1534820k,nr_inodes=383705,mode=755)
sysfs on /sys type sysfs (rw,relatime,seclabel)
selinuxfs on /sys/fs/selinux type selinuxfs (rw,relatime)
proc on /proc type proc (rw,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,relatime,seclabel)
configfs on /sys/kernel/config type configfs (rw,relatime)
tmpfs on /run type tmpfs (rw,nosuid,nodev,seclabel,size=1746436k,nr_inodes=436609,mode=755)
/dev/mmcblk0p36 on /mnt/userdata type ext4 (rw,relatime,seclabel,commit=600)
/dev/mmcblk0p1 on /etc/productinfo type ext4 (rw,relatime,seclabel,nodelalloc,commit=600,data=journal)
/dev/mmcblk0p36 on /mnt/data type ext4 (rw,relatime,seclabel,commit=600)
/dev/mmcblk0p36 on /home type ext4 (rw,relatime,seclabel,commit=600)
/dev/mmcblk0p36 on /var type ext4 (rw,relatime,seclabel,commit=600)
tmpfs on /var/volatile type tmpfs (rw,relatime,seclabel,size=1746436k,nr_inodes=436609)
devpts on /dev/pts type devpts (rw,relatime,seclabel,gid=5,mode=620,ptmxmode=000)
```

8 下载和开机

8.1 下载软件流程

下载软件流程如图 8-1 所示。

图8-1 下载软件流程



启动之后，启动 Romcode，在 IRAM 中运行 FDL1，在 DDR 中运行 FDL2，然后把 SPL、Boot、System 等写入 NAND Flash 中。

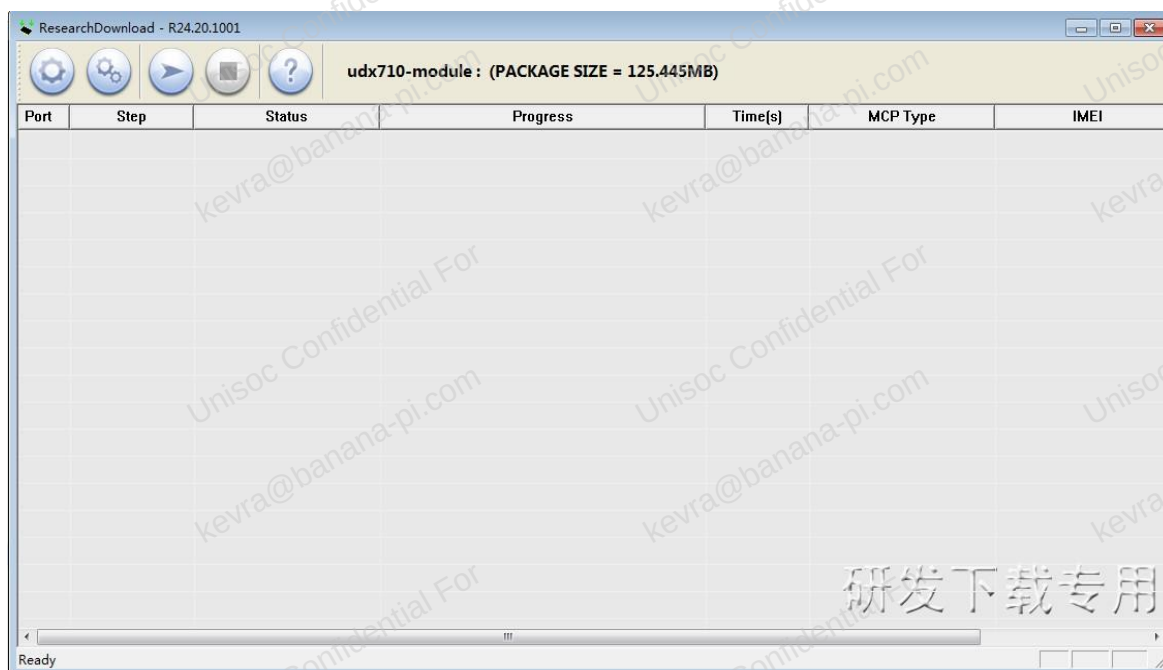
8.2 下载工具使用

8.2.1 工具介绍

ResearchDownload 工具可从 Uni-Support 工具管理模块获取。

运行 ResearchDownload.exe，工具主界面如图 8-2 所示。其中，左上角的“R24.20.1001”是工具的版本号，版本号会不断升级，具体以实际工具界面为准。

图8-2 ResearchDownload 主界面



8.2.1.1 工具栏

工具栏各快捷图标功能说明如表 8-1 所示。

表8-1 ResearchDownload 工具栏说明

工具栏图标	说明
	加载按钮。选择待下载 pac 包文件。
	设置按钮。设置 ResearchDownload 参数，程序将保存设置的结果。使用中若环境无变化，无需再设置。开始下载后，程序禁用此按钮，停止下载后恢复为可用状态。
	开始按钮。单击此按钮后，程序检测并打开可用端口，准备开始下载过程。如果有模块连接到指定的端口，程序检测到模块则开始下载过程。 第一次运行程序时，此按钮处于禁用状态，只有先加载 pac 包文件后，才可以开始下载过程。
	停止按钮。单击此按钮停止下载过程。只有当用户单击开始按钮，程序打开可用端口后此按钮才变为可用状态。 只有停止下载过程，才能够退出程序。
	版本信息按钮。单击此按钮显示程序版本和版权信息。

8.2.1.2 状态窗口

工具栏下方区域为状态窗口，显示端口的状态、下载的状态、结果等。

Port	Step	Status	Progress	Time[s]	MCP Type	IMEI	Rate[MB/s]
3	System	Downloading...		22	—	867400020316612	Avg:4.82, Peak:11.50

状态窗口说明如表 8-2 所示。

表8-2 ResearchDownload 状态窗口说明

Item	说明
Port	当前下载模块的端口号。
Step	当前下载操作的步骤。
Status	当前的下载操作状态。
Progress	进度条显示当前下载的完成情况。 - 显示“Waiting”表示没有检测到模块。 - 显示“Inprogress”表示正在进行某种操作。 - 下载成功后此栏将变为绿色，如下载出错，此栏将变为红色。
Time(s)	单次下载计时，单位为秒。
MCP Type	显示当前下载模块的 MCP Type 信息。
IMEI	显示当前下载模块的 IMEI 信息。
Rate[MB/s]	显示下载速率 - Avg 表示平均速率。 - Peak 表示最高速率。

8.2.2 准备 pac 包

生成 pac 包，请参考 4.6 生成 pac 包。

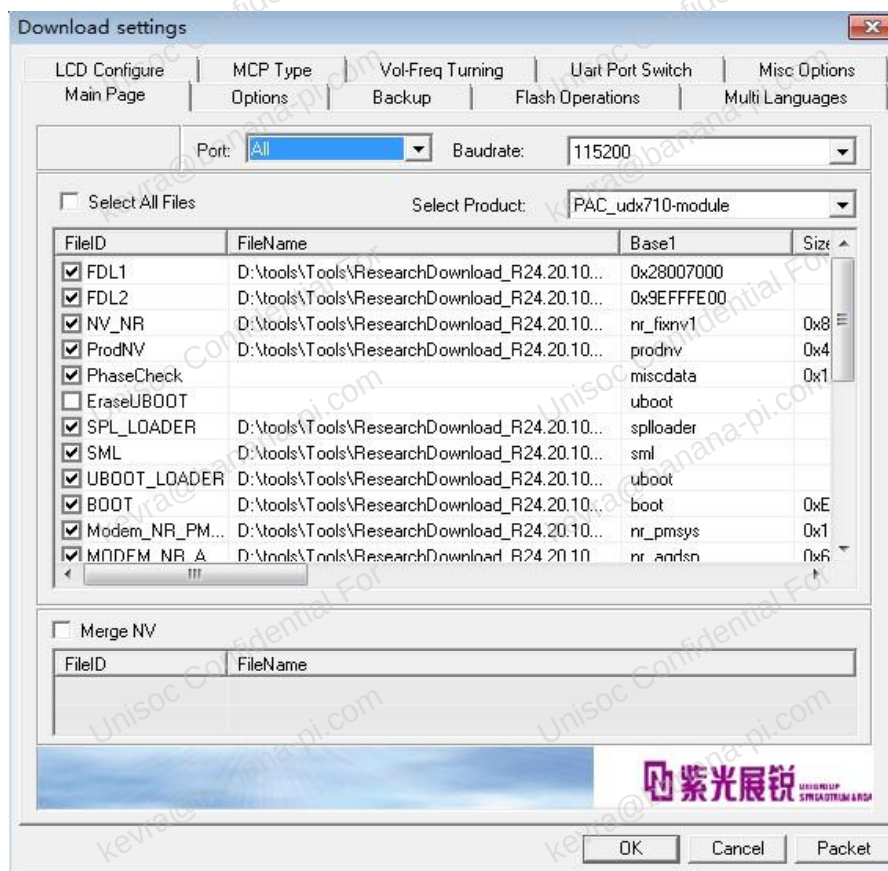
8.2.3 下载 pac 包

步骤 1 单击加载按钮 ，加载选定的 pac 包。

步骤 1 单击设置按钮 ，设置下载参数。

设置界面如图 8-3 所示，pac 包加载完成后，建议使用默认配置下载整个 pac 包文件。后续根据调试需要可以单独下载需要更新的文件。

图8-3 ResearchDownload 下载设置



步骤 2 单击开始按钮 ，开始下载。

程序会自动检测可用的下载端口，并打开端口准备下载。

步骤 3 手动进入下载模式后，正确连接到端口。

程序自动检测到终端后开始下载。

Port	Step	Status	Progress	Time[s]	MCP...	IMEI	Rate[MB/s]
31	System	Downloading...		39	—	867400020316612	Avg:8.72, Peak:14.05

状态窗口将显示下载的进度。下载完成后对应端口的 Progress 栏以不同的颜色显示下载结果，绿色“Passed”表示成功，红色“Failed”表示失败。

图8-4 下载成功

Port	Step	Status	Progress	Time[s]	MCP...	IMEI	Rate[MB/s]
31	UBOOTLoader	Finish	Passed	59s	—	867400020316612	Avg:10.15, Peak:14.14

图8-5 下载失败

Port	Step	Status	Progress	Time[s]	MCP...	IMEI	Rate[MB/s]
31	FDL2	Unplugged	Failed: User cancel	3s	—	—	Avg:0.00, Peak:0.00

步骤 4 将终端与端口断开连接。

步骤 5 单击停止按钮，停止下载。

说明

- 如果所有待调试终端已经下载完成，则可以停止下载。
- 如果还有其他待调试终端需要下载，将该终端连接到端口，程序将自动开始下载。

----结束

更详细的指导请参考 ResearchDownload 工具包中的 User Guide 文档。

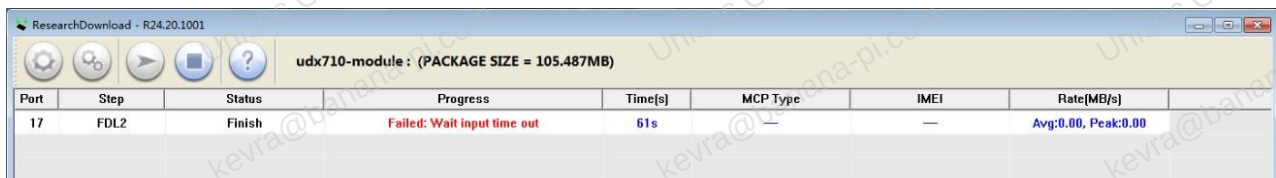
8.3 下载常见问题

NAND Flash 没有配置

问题现象

下载失败，工具状态窗口显示如图 8-6 所示。

图8-6 ResearchDownload 下载失败



问题分析

终端 AP 侧串口 Log 中，如果有如下红色信息，表示工程文件（source\devtools\mcp_gen\mcp.xls）的 NAND Flash 支持列表中没有配置该 NAND Flash。

```
UUUUUNIF
NIF
ROM
RBS
...
SprdGetNandParam
```

id[0] is ef id[1] is aa id[2] is 90 id[3] is 15

id[4] is 4

...

Nand params unconfigured, please check it. Halt on booting!!!

解决方案

通过 Uni-Support 的 QVL（Qualified Vendor List，合格供应商列表）模块查询使用的 NAND Flash 是否在本平台的 QVL 中。

- 如果在 QVL 中，请提 CQ 更新该 mcp.xls。
- 如果不在 QVL 中，请启动 AVL（Approved Vendor List，认可的供应商清单）认证流程。

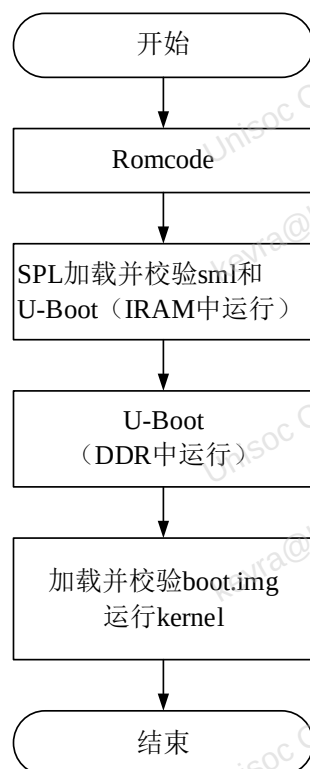
8.4 开机方式

具体开发板的开机方式，请参考具体开发板的说明。

8.5 开机启动流程

开机启动流程如图 8-7 所示。

图8-7 开机启动流程



启动之后，启动 Romcede，在 IRAM 中 SPL 加载并校验 SML 和 U-Boot，在 DDR 中运行 U-Boot，然后加载并校验 boot.img，运行 Kernel，最后机器启动成功。

8.6 Log 抓取和分析

8.6.1 工具侧 Log 抓取

把 ResearchDownload_R24.20.1001\Bin\Channel.ini 文件中的 Level 设置为 5。

[Log]

; Text log level

; 0, No text log

; 1, Log errors,default value

; 2, Log warnings

; 3, Log runtime information

; 4, Log data only

; 5, Log everything

Level = 5

Log 保存路径：ResearchDownload_R24.20.1001\Bin\Log\2020_08_12

其中，2020_08_12 表示 ResearchDownload 保存 Log 时，获取的本 PC 日期。

8.6.2 终端 AP 侧串口 Log 抓取

抓取 AP 侧串口 Log 的步骤如下：

步骤 1 设置日志等级。

- 方法一：在 ResearchDownload 工具设置。



单击 ，在 Download settings 界面的 Options 页签，设置 Debug Level 为 7。

说明

- 开启 Kernel Log 功能：设置 Debug Level 为 7，并且完成 pac 包下载。软件会将 kernel log level 设置为 7。
- 关闭 Kernel Log 功能：设置 Debug Level 为 1，同时 Main Page 页签中勾选 PhaseCheck 下载项，重新下载 pac 包。
- 方法二：修改源代码。
例如，修改 source/kernel/kernel5.4/arch/arm64/boot/dts/sprd/uis7885.dts 文件中的 loglevel 值为 7，重新编译 Kernel，下载新 boot.img 文件。

步骤 2 连接目标单板。

- 展锐参考板，将 USB 线一端连接到单板的 USB 接口，另一端连接到 PC 的 USB 接口。
- 非展锐参考板，将 USB 转串口线一端连接到单板的串口，另一端连接到 PC 的 USB 接口。

说明

- 展锐参考板有一个 USB 转串口芯片，这个 USB 转串口芯片一端连接到 UIS7885 的串口，另一端连接到 USB 接口，所以能直接用 USB 线连接展锐参考板的 USB 接口。
- 非展锐参考板，一般不会有这个 USB 转串口芯片，需要使用 USB 转串口线连接到单板的串口。串口 Log 从 UIS7885 芯片的串口输出。

步骤 3 待 PC 识别端口后，打开串口工具，选择端口。

- 展锐参考板，大部分选择第一个端口。
- 非展锐参考板，需根据单板设计选择，也可以依次尝试。

步骤 4 单板上电，系统启动，串口工具抓取 Kernel Log。

说明

- 抓取下载流程 Log，可以使用默认日志等级。
- 抓取开机启动流程 Log，建议设置日志等级为 7，可以获取 Kernel Log。

----结束

8.6.3 开机 Log 抓取

成功烧录 pac 包后，正确连接串口和电源，打开串口吐出的 Log 即为开机 Log，可以根据 Log 进行分析。

8.6.4 Log 分析

终端 AP 侧串口 Log 分析：

RBS //下载直接以RBS开始

U-Boot 2015.07 (Jul 27 2020 - 18:56:56 +0000)orca

U-Boot code: 9F000000 -> 9F09D208 BSS: -> 9F156698

...

SprdGetNandParam //在source\devtools\mcp_gen\mcp.xls中查询NAND Flash ID，若未查询到下载会失败
 id[0] is ef id[1] is aa id[2] is 90 id[3] is 15 id[4] is 4

__PrintNandInfo

...

power_button_pressed(): power_button_pressed status 2 //按键信息

... do_role(): Get chipram env mode 102,go download //通过获取Chipram配置信息，进入下载模式

...

main_loop: bootcmd="download" Hit any key to stop autoboot: 0

do_download:enter // U-Boot中进入下载流程

now time sec 0 now alarm sec 0 set_buf_base_size(): download buffer base 0x82000000, size 0x2000000

dl_cmd_handler:enter

9 附录

MACHINE: ums9620-2h10 (相关文件 uis7885-2h10.conf)

表9-1 ums9620-2h10 信息

Arch	Kernel	DISTRO	IMAGE
Arm64	5.4	unisoc-wayland	unisoc-wayland-image

MACHINE: ums9620-1h10 (相关文件 ums9620-1h10.conf)

表9-2 ums9620-1h10 信息

Arch	Kernel	DISTRO	IMAGE
Arm64	5.4	unisoc-wayland	unisoc-wayland-image

MACHINE: ums9620-1h10-k515 (相关文件 ums9620-1h10-k515.conf)

表9-3 ums9620-1h10-k515 信息

Arch	Kernel	DISTRO	IMAGE
Arm64	5.15	unisoc-wayland	unisoc-wayland-image

MACHINE: uis8520-1h10-nand (相关文件 uis8520-1h10-nand.conf)

表9-4 uis8520-1h10-nand 信息

Arch	Kernel	DISTRO	IMAGE	Product
Arm64	5.15	unisoc-wayland	unisoc-wayland-image	Uboot22/abota 等

MACHINE: uis8910dm-2h10-nand (相关文件 uis8910dm-2h10-nand.conf)

表9-5 uis8910dm-2h10-nand 信息

Arch	Kernel	DISTRO	IMAGE	Product
Arm	6.1	unisoc-initgc	unisoc-console-image	pos/posc6

MACHINE: uis8360-1h10-nand (相关文件 uis8360-1h10-nand.conf)

表9-6 uis8360-1h10-nand 信息

Arch	Kernel	DISTRO	IMAGE	Product
Arm	6.1	unisoc-tiny	unisoc-console-image	lite/full

MACHINE: uis8510-1h10-emmc (相关文件 uis8510-1h10-emmc.conf)

表9-7 uis8510-1h10-emmc 信息

Arch	Kernel	DISTRO	IMAGE	Product
Arm	6.6	unisoc-tiny	unisoc-console-image	cpe/mifi

MACHINE: uis7885-2h10 (相关文件 uis7885-2h10.conf)

表9-8 uis7885-2h10 信息

Arch	Kernel	DISTRO	IMAGE
Arm64	5.4	unisoc-wayland	unisoc-wayland-image

10

参考资料

1. [1] <https://docs.yoctoproject.org/overview-manual/yp-intro.html>
2. <https://www.yoctoproject.org/docs/3.1/dev-manual/dev-manual.html>
3. <https://www.yoctoproject.org/docs/3.1/ref-manual/ref-manual.html#detailed-supported-distros>
4. 《Yocto 目录结构介绍》