Hi3559A/C V100 开发环境

用户指南

文档版本 00B06

发布日期 2018-10-15

商标声明

 、HISILICON、海思和其他海思商标均为深圳市海思半导体有限公司的商标。

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。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

深圳市海思半导体有限公司

地址：深圳市龙岗区坂田华为基地华为电气生产中心　　邮编：518129

网址：http://www.hisilicon.com

客户服务电话：+86-755-28788858

客户服务传真：+86-755-28357515

客户服务邮箱：support@hisilicon.com

前 言

概述

本文档介绍 Linux 和 Huawei LiteOS 开发环境。Linux 开发环境的搭建、U-boot、Linux 内核、根文件系统以及内核和根文件系统的烧写，以及创建网络开发环境和如何启动 Linux 开发应用程序。Huawei LiteOS 配置以及编译。

本文档主要提供让客户更快地了解 Linux 和 Huawei LiteOS 开发环境指导。

说明
未有特殊说明，Hi3559CV100 与 Hi3559AV100 内容一致。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3559A	V100
Hi3559C	V100

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修订日期	版本	修订说明
2018-10-15	00B06	第 6 次临时版本发布 第 5，9,和 10 章节涉及修改
2018-05-15	00B05	第 5 次临时版本发布 4.3.4 和 4.3.6 小节涉及修改
2018-02-10	00B04	第 4 次临时版本发布 8.5 小节涉及修改 9.1 小节，更换图 9-1
2018-01-23	00B03	第 3 次临时版本发布。 1.3.2、3.3、4.3.5、8.4 和 10.1 小节涉及修改 新增 8.3、8.5 和 8.6 小节
2018-01-10	00B02	第 2 次临时版本发布。 全文均涉及修改，新增第 10 章
2017-11-15	00B01	第 1 次临时版本发布。

目 录

插图目录

表格目录

1 开发环境

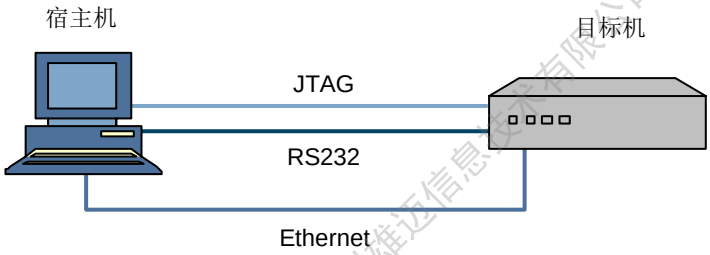
1.1 嵌入式开发环境

由于嵌入式单板的资源有限，不能在单板上运行开发和调试工具，通常需要交叉编译调试的方式进行开发和调试，即“宿主机＋目标机（评估板）”的形式。宿主机和目标机一般采用串口连接，也可同时通过网口或者 JTAG 连接，如图 1-1 所示。

宿主机和目标机的处理器一般不相同。宿主机需要建立适合于目标机的交叉编译环境。程序在宿主机上经过“编译－连接－定位”得到可执行文件。通过一定的方法将可执行文件烧写到目标机中，然后在目标机上运行。

目标机上的 Bootloader 启动后，目标机中的操作信息通过串口或者网口输出到宿主机上显示。在宿主机上的控制台中输入命令，可以控制目标机。

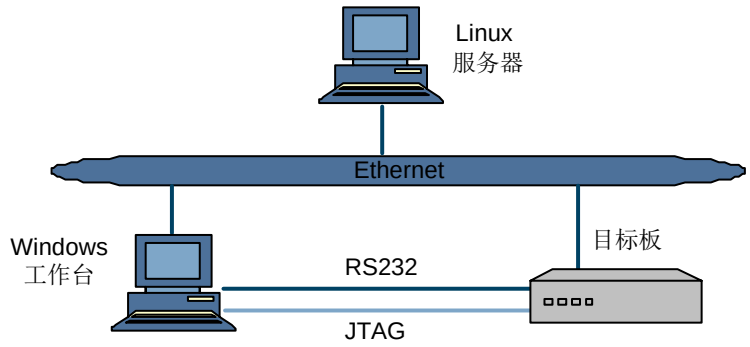
图1-1 嵌入式开发图例



1.2 Hi3559AV100 开发环境

Hi3559AV100 开发环境通常包括 Linux 服务器、Windows 工作台和 Hi3559AV100DMEB（目标板），三者同处于一个网络中，如图 1-2 所示。

图1-2 Hi3559AV100 开发环境



在 Linux 服务器上建立交叉编译环境，Windows 工作台通过串口和网口与 Hi3559AV100 单板连接，开发人员可以在 Windows 工作台中进行程序开发或者远程登录到 Linux 服务器进行程序开发。各部分具体软件介绍如表 1-1 所示。

说明

开发环境中使用了 Windows 工作台，实际上很多工作也可以在 Linux 服务器上完成，如使用 minicom 代替超级终端等，用户可自行选择。

表1-1 Hi3559AV100 开发环境的各部分软件描述

软件		描述
Windows 工作台	操作系统	Windows XP/Windows7/Windows10。
	应用软件	putty、超级终端、tftp 服务器、DS-5 等软件。
Linux 服务器	操作系统	无特别要求，可为 Ubuntu、Redhat、Debian 等。内核版本支持 2.6.18 及以上版本。安装时建议选择完全安装。
	应用软件	NFS、telnetd、samba、vim、arm 交叉编译环境（Gcc 版本 4.9）等。 其他应用软件根据具体开发需要而定，通常系统都已默认安装，只要适当配置即可。
Hi3559AV100	引导程序	U-boot。
	操作系统	Hisilicon Linux、Huawei LiteOS。Linux 内核基于 Linux 标准内核 4.9.y 版本移植开发，根文件系统基于 busybox 1.26.2 版本制作而成。
	应用软件	包含 telnetd、gdb server 等 Linux 常用命令。
	程序开发库	glibc-2.24 版本。

1.3 搭建开发环境

1.3.1 安装 Linux 服务器

建议选择常用的 Linux 发行版，便于寻找各类技术资源。例如：

- RedHat 较新的发行版如 RedHat Fedora Core 系列和 Redhat Enterprise Linux、Red Hat 3.4.4-2。
- RedHat 较老的发行版如 RedHat 9.0 等。

推荐使用较新版本，以方便获取各类资源，如 Fedora Core 系列、SUCE10、Ubuntu10。

Debian 的各类发行版也是常用的。使用 Debian 的好处是各类安装包都可以随时在线更新，各类软件包资源也很丰富。

1.3.2 安装交叉编译工具

注意

使用从网络等渠道得到的交叉编译工具可能存在与使用的内核并不配套，造成开发过程中出现一些不可预料的问题。

发布包提供编译工具链 aarch64-himix100-linux、gcc-arm-none-eabi-4_9-2015q3。aarch64-himix100-linux 为基于 64bit 操作系统 glibc 的工具链，gcc-arm-none-eabi-4_9-2015q3 为 Huawei LiteOS M7 工具链。

安装步骤如下(以 aarch64-himix100-linux 举例)：

步骤 1. 解压工具链。

执行如下命令进行解压：

tar -xvf aarch64-himix100-linux.tgz

步骤 2. 安装工具链。

运行命令 sudo ./aarch64-himix100-linux.install 即可完成此工具链的安装。

其它工具链安装方法与上述描述类似。

----结束

1.3.3 安装 Hi3559AV100 SDK

请参考《Hi3559AV100 SDK 安装及升级使用说明》

注意

通过 Hi3559AV100 DMEB 的软件开发包制作的镜像（包括 u-boot、内核和文件系统），只支持 64 位操作系统。

2 U-boot

关于 U-boot 的介绍与使用请参见《Hi3559A/C V100 U-boot 移植应用 开发指南》。

3 Linux 内核

3.1 内核源代码

成功安装 Hi3559AV100 SDK 后，内核源代码已存放于 SDK 目录下的 osdrv/目录中，用户可直接进入目录进行相关操作。

3.2 配置内核

> **注意**
>
> 如果对内核和 Hi3559AV100 平台没有足够了解，请勿修改默认配置。但可增加需要的模块。

配置内核的操作步骤如下：

步骤 1. 手动拷贝.config 文件：

```
cp arch/arm64/configs/hi3559av100_arm64_xxx_defconfig .config
```

（注：当启动介质是eMMC、UFS、SPI-Nor Flash或SPI-NAND Flash时，使用hi3559av100_arm64_big_little_defconfig；当启动介质是NAND Flash时，使用hi3559av100_arm64_big_little_nand_defconfig）

步骤 2. 用户通过“make menuconfig”进行内核配置：

```
make ARCH=arm64 CROSS_COMPILE=aarch64-himix100-linux- menuconfig
```

步骤 3. 选择需要的模块。

步骤 4. 选择完毕后，保存并退出。

----结束

说明

编译内核时需要在 make 后添加两个参数：ARCH=arm64 CROSS_COMPILE=aarch64-himix100-linux-，其中 CROSS_COMPILE 表示工具链。

3.3 编译内核并生成 ATF+kernel 镜像

步骤 1. 配置保存后，可直接输入

make ARCH=arm64 CROSS_COMPILE=aarch64-himix100-linux- uImage-j 20 命令编译内核生成镜像 uImage，此时需要等待几分钟。

说明

如果编译过程中出现错误，按顺序执行以下命令：

make ARCH=arm64 CROSS_COMPILE=aarch64-himix100-linux- clean

make ARCH=arm64 CROSS_COMPILE=aarch64-himix100-linux- menuconfig

make ARCH=arm64 CROSS_COMPILE=aarch64-himix100-linux- uImage

步骤 2. 进入 osdrv/opensource/arm-trusted-firmware/ arm-trusted-firmware 目录，执行 mk.sh 脚本

```
cd ../../arm-trusted-firmware/arm-trusted-firmware
./mk.sh
```

说明

当 linux 目录有变更时，需要修改 mk.sh 中内核路径，使其指向正确的 linux 目录。

在 osdrv/opensource/ arm-trusted-firmware/arm-trusted-firmware/build/hi3559av100/debug 目录下，生成的 fip.bin 文件就是 ATF+kernel 的镜像 uImage。

----结束

4 根文件系统

4.1 根文件系统简介

Linux 的目录结构的最顶层是一个被称为“/”的根目录。系统加载 Linux 内核之后，就会挂载一个设备到根目录上。存在于这个设备中的文件系统被称为根文件系统。所有的系统命令、系统配置以及其他文件系统的挂载点都位于这个根文件系统中。

根文件系统通常存放于内存和 Flash 中，或是基于网络的文件系统。根文件系统中存放了嵌入式系统使用的所有应用程序、库以及其他需要用到的服务。图 4-1 列出了根文件系统的顶层目录。

图4-1 根文件系统顶层目录结构图

通用的 Linux 系统的根文件系统中会包括根文件系统顶层目录结构图中所有的目录，不过在嵌入式系统中，需要精简根文件系统。部分可以被忽略的目录如表 4-1 所示。

表4-1 嵌入式系统中可忽略的目录说明

目录名称	描述
/home、/mnt、/opt 和/root	所有适合提供给多用户扩展的目录，都可以被忽略。
/var 和/tmp	/var 是存放系统日志或一些服务程序的临时文件。 /tmp 是存放用户的一些临时文件，可以被忽略。
/boot	/boot 目录一般用于存放内核映像，PC 机启动时一般会从该位置加载内核，但在嵌入式系统中，为了节省空间，内核映像存在于 Flash 或网络服务器中，而不是在根文件系统中。因此也可以忽略这个目录。

注：空目录并不会增大文件系统的体积，如果没有特殊原因，建议保留这些目录。

4.2 利用 busybox 制作根文件系统

利用 busybox 制作根文件系统需要先获取 busybox 源代码，然后配置、编译和安装 busybox，操作成功后开始制作根文件系统。

4.2.1 获取 busybox 源代码

成功安装 SDK 后，busybox 完整源代码就存放在 osdrv/目录中。要获取 busybox 源代码也可以从网站 http://www.busybox.net 下载。

4.2.2 配置 busybox

进入 busybox 所在目录，进行配置操作需要输入如下命令：

- cp osdrv/busybox/busybox-1.26.2/config_aarch64_v610_XXX_softfp_neon osdrv/busybox/busybox-1.26.2/.config //指定配置文件

  其中，config_XXX_softfp_neon 代表两种情况：

  - config_aarch64_v610_a53_softfp_neon 对应 64 bit 操作系统单核工具链 aarch64-himix100-linux
  - config_aarch64_v610_a73_a53_softfp_neon 对应 64 bit 操作系统多核工具链 aarch64-himix100-linux
- make menuconfig

  busybox 的配置界面和内核配置相似，其功能选项容易理解，可以根据自己的需求选择配置。在 Busybox Settings ---> Build Options 中注意下面一个选项：

```
[*] Build with Large File Support (for accessing files > 2 GB)
(aarch64-himix100-linux-) Cross Compiler prefix
()  Path to sysroot
```

```
( -mcpu=cortex-a53 -mfloat-abi=softfp -mfpu=neon-vfpv4) Additional
CFLAGS
() Additional LDFLAGS
() Additional LDLIBS
```

其中：

第一个选项是用于选择 SDK 推荐的交叉编译器，配置好后保存并退出。欲了解 busybox 各选项含义请参考 busybox 配置帮助。

4.2.3 编译和安装 busybox

编译和安装 busybox 的具体操作如下：

```
make
make install
```

编译并安装成功后，在 busybox 目录下的_install 目录下生成以下目录及文件：

```
drwxrwxr-x  2 xxx XXX 4096 Feb 13 11:41 bin
lrwxrwxrwx  1 xxx XXX   11 Feb 13 11:41 linuxrc -> bin/busybox
drwxrwxr-x  2 xxx XXX 4096 Feb 13 11:41 sbin
drwxrwxr-x  4 xxx XXX 4096 Feb 13 11:41 usr
```

其中xxx表示用户；XXX表示组。

4.2.4 制作根文件系统

成功安装 SDK 后，在 osdrv/pub/目录中存放已制作好的根文件系统。

用户如有需要可在 busybox 的基础上制作根文件系统。

制作根文件系统的具体操作步骤如下：

步骤 1. mkdir rootbox

cd rootbox

cp –R packet/os/busybox-1.26.2/_intsall/* .

mkdir etc dev lib tmp var mnt home proc

步骤 2. 配置 etc、lib、dev 目录的必需文件。

- etc 目录可参考系统/etc 下的文件。其中最主要的文件包括 inittab、fstab、init.d/rcS 文件等，这些文件最好从 busybox 的 examples 目录下拷贝过来，根据需要自行修改。
- dev 目录下的设备文件，可以直接从系统中拷贝过来或者使用 mknod 命令生成需要的设备文件。拷贝文件时请使用 cp –R file。
- lib 目录是存放应用程序所需要的库文件，请根据应用程序需要拷贝相应的库文件。

----结束

完成以上两个步骤，一个完整的根文件系统就生成了。

说明

SDK 软件包中已经包括配置好的完整的根文件系统，如果无特别需求，可直接使用。要添加自己开发的应用程序，只需将应用程序和相应的库文件拷贝到根文件系统的对应目录即可。

注意

- 为了便于调试，默认发布的版本中没有设置 root 密码；
- 为了保证系统安全，请客户在产品中自行设置 root 密码。

4.3 文件系统简介

嵌入式系统中常用文件系统包括有 cramfs、jffs2、NFS、initrd、yaffs2、ext4 以及 squashfs、ubifs。它们的特点如下：

- cramfs 和 jffs2 具有好的空间特性，很适合嵌入式产品应用。
- cramfs 与 squashfs 为只读文件系统，目前只有 SPI Nor FLASH 支持这两种文件系统。
- squashfs 压缩率最高。
- jffs2 为可读写文件系统。
- NFS 文件系统适用于开发初期的调试阶段。
- yaffs2 文件系统只用于 NAND Flash。
- initrd 采用 cramfs 文件系统，为只读。
- ext4 文件系统用于 eMMC 卡和 UFS。

4.3.1 cramfs

cramfs 是针对 Linux 内核 2.4 之后的版本所设计的一种新型文件系统，使用简单，加载容易，速度快。

cramfs 的优缺点如下：

- 优点

  将文件数据以压缩形式存储，在需要运行时进行解压缩，能节省 Flash 存储空间。

- 缺点

  由于它存储的文件是压缩的格式，所以文件系统不能直接在 Flash 上运行。同时，文件系统运行时需要解压数据并拷贝至内存中，在一定程度上降低读取效率。另外 cramfs 文件系统是只读的。

如果想要在单板运行的 Linux 中提供 cramfs 的能力，必须要在编译内核时把 cramfs 的选项加入。在 make menuconfig 后，进入“File systems”，选择“Miscellaneous filesystems”，最后选中其中的“Compressed ROM file system support (cramfs) (OBSOLETE)”（SDK 里面提供的内核默认已经选择了该文件系统的支持）。

mkfs.cramfs 是用来制作 cramfs 文件系统映象的工具。通过这个工具处理已经制作好的根文件系统，就可以生成 cramfs 文件系统的映象（这类似于我们把光盘制作成 ISO 文件映像）。具体操作如下所示：

```
mkfs.cramfs  ./rootbox ./cramfs-root.img
```

其中，rootbox 是之前已经制作好的根文件系统，cramfs-root.img 是生成的 cramfs 文件系统映像文件。

4.3.2 jffs2

jffs2 是 RedHat 的 David Woodhouse 在 jffs 基础上改进的文件系统，是用于微型嵌入式设备的原始闪存芯片的实际文件系统。jffs2 文件系统是日志结构化的可读写的文件系统。

jffs2 的优缺点如下：

- 优点

  使用了压缩的文件格式。最重要的特性是可读写操作。

- 缺点

  jffs2 文件系统挂载时需要扫描整个 jffs2 文件系统，因此当 jffs2 文件系统分区增大时，挂载时间也会相应的变长。使用 jffs2 格式可能带来少量的 Flash 空间的浪费。这主要是由于日志文件的过度开销和用于回收系统的无用存储单元，浪费的空间大小大致是若干个数据段。jffs2 的另一缺点是当文件系统已满或接近满时，jffs2 运行速度会迅速降低。这是因为垃圾收集的问题。

加载 jffs2 文件系统时的步骤如下：

步骤 1. 扫描整个芯片，对日志节点进行校验，并且将日志节点全部装入内存缓存。

步骤 2. 对所有日志节点进行整理，抽取有效的节点并整理出文件目录信息。

步骤 3. 找出文件系统中无效节点并且将它们删除。

步骤 4. 最后整理内存中的信息，将加载到缓存中的无效节点释放。

----结束

由此可以看出虽然这样能有效地提高系统的可靠性，但是在一定程度上降低了系统的速度。尤其对于较大的闪存芯片，加载过程会更慢。

为了使内核支持 jffs2 文件系统，必须在编译内核时把 jffs2 的选项加入（我们发布的内核默认已经加入了支持）。在 make menuconfig 后，进入“File systems”，选择“Miscellaneous filesystems”，最后选中其中的“Journalling Flash File System v2 (JFFS2) support”选项（SDK 里面提供的内核默认已经选择了该文件系统的支持）。

jffs2 的制作方法为：

```
mkfs.jffs2 -d ./rootbox -l -e 0x20000 -o  jffs2-root.img
```

其中，mkfs.jffs2 工具可以从互联网中下载，也可以在 SDK 包中找到。rootbox 为之前已经制作好的根文件系统。参数说明如表 4-2 所示。

表4-2 JFFS2 参数表

参数	说明
d	指定根文件系统
l	little-endian 小端模式
e	Flash 的块大小
o	输出映像文件

4.3.3 yaffs2

yaffs2 是专门为 NAND Flash 设计的嵌入式文件系统。它是日志结构的文件系统，提供了损耗平衡和掉电保护，可以有效地避免意外掉电对文件系统一致性和完整性的影响。

yaffs2 的优缺点如下：

- 优点
  - 专门针对 NAND Flash，软件结构得到优化，速度快。
  - 使用硬件的 spare area 区域存储文件组织信息，启动时只需扫描组织信息，启动比较快。
  - 采用多策略垃圾回收算法，能够提高垃圾回收的效率和公平性，达到损耗平衡的目的。
- 缺点

  没有采用压缩的文件格式。当包含的内容相同时，yaffs2 镜像文件要比 jffs2 镜像文件大。

yaffs2 文件系统在 SDK 中作为一个模块提供。只需在 yaffs2 代码中的 Makefile 中加入所依赖的内核代码路径，进行编译，即可生成 yaffs2 文件系统模块。

yaffs2 镜像文件的制作和 cramfs 相同，即通过工具制作，只需简单的几个参数，具体如下：

```
mkyaffs2image  ./rootbox yaffs2-root.img [pagesize] [ecctype]
```

其中，rootbox 是之前已经制作好的根文件系统，yaffs2-root.img 是生成的 yaffs2 文件系统镜像文件，pagesize 是单板上焊接 NAND Flash 器件的页大小，ecctype 是单板上焊接 NAND Flash 器件的 ecc 类型。

4.3.4 initrd

initrd 相当于存储介质，它支持的文件系统格式有 ext2、cramfs 等，因此内核除了支持 initrd 之外，还要支持 cramfs 文件系统。内核需要做如下配置，initrd 才可以正常工作：

- 进入“Device Drivers->Block devices”，选择支持“RAM block device support”。

- 进入“General setup”，选择支持“Initial RAM filesystem and RAM disk (initramfs/initrd) support”。
- 进入“File systems”，选择“Miscellaneous filesystems”，最后选中其中的“Compressed ROM file system support (cramfs) (OBSOLETE)”。

当前 SDK 中默认选中以上两项。

制作initrd的步骤如下：

步骤 1. 制作 cramfs 镜像文件，具体制作方法请参见“4.3.1 cramfs”。

步骤 2. 以步骤 1 制作的镜像文件作为输入，制作 initrd 文件，制作命令为“mkimage -A arm64 -T ramdisk -C none -a 0 -e 0 -n cramfs-initrd -d ./cramfs-image cramfs-initrd”。

----结束

4.3.5 squashfs

squashfs 文件系统是一套基于 Linux 内核使用的压缩只读文件系统，压缩率高。

squashfs 具有如下特点：

- 数据(data),节点(inode)和目录(directories)都被压缩
- 保存了全部的 32 位 UID/GIDS 和文件的创建时间
- 最大支持 4G 文件系统
- 检测并删除重复文件

使用 squashfs 文件系统步骤：

步骤 1. 制作支持 squashfs 的内核镜像。进入 linux-4.9.y 目录下，执行以下命令：

```
cp arch/arm64/configs/hi3559av100_arm64_big_little_defconfig .config
make ARCH=arm64 CROSS_COMPILE=aarch64-himix100-linux- menuconfig （保存退出即可）
make ARCH=arm64 CROSS_COMPILE=aarch64-himix100-linux- uImage
cd ../../arm-trusted-firmware/ arm-trusted-firmware
./mk.sh
```

步骤 2. 制作 squashfs 文件系统镜像。在发布包 SDK/package/osdrv/tools/pc_tools 目录下的 mksquashfs 为制作 squashfs 文件系统工具。使用方法如下：

```
./mksquashfs rootfs ./rootfs.squashfs.img -b 64K -comp xz
```

其中，rootfs 是之前已经制作好的根文件系统，rootfs.squashfs.img 是生成的 squashfs 文件系统映像文件。-b 64K 指定 squashfs 文件系统的块大小为 64K（决定于实际 spi flash 块大小）。-comp 指定文件系统压缩方式为 xz。请根据实际情况修改参数。

----结束

4.3.6 ext4

ext4 文件系统是一个高效的、优秀的、可靠的和极具特点的文件系统，相对于 ext3 的的改进是更深层次的，是文件系统数据结构方面的优化。

步骤 1. 制作支持 ext4 的内核镜像。

进入 linux-4.9.y 目录下，执行以下命令：

```
cp arch/arm64/configs/hi3559av100_arm64_xxx_defconfig .config
（注：使用hi3559av100_arm64_big_little_defconfig或
hi3559av100_arm64_big_little_nand_defconfig）
make ARCH=arm64 CROSS_COMPILE=aarch64-himix100-linux- menuconfig（保存退出即可）
make ARCH=arm64 CROSS_COMPILE=aarch64-himix100-linux- uImage
cd ../../arm-trusted-firmware/ arm-trusted-firmware
./mk.sh
```

步骤 2. 制作 ext4 文件系统镜像。

在发布包 osdrv/tools/pc_tools 目录下的 make_ext4fs 为制作 ext4 文件系统工具。使用方法如下：

```
./ make_ext4fs -l 96M -s rootfs.ext4.img rootfs
```

其中，-l 96M 是指定 uboot 中配置 ext4 的文件系统分区大小为 96MB，-s 为使用 gzip 压缩，rootfs.ext4.img 是生成的 Ext4 文件系统映像文件，rootfs 是之前已经制作好的根文件系统。请根据实际情况修改参数。

----结束

5 多核加载启动

5.1 Linux(multi-core)(A53MP+A73MP)+Huawei LiteOS(A53UP)

编译 OSDRV 时，AMP_TYPE 传参为 linux，此参数下生成的 A53UP(Huawei LiteOS)、M7(Huawei LiteOS)镜像支持在 linux 下加载，加载步骤如下：

步骤 1. 系统上电时，A53MP 首先启动，运行 uboot。

步骤 2. Uboot 下使用“bootm 0Xxxxxxxxx” 命令启动 A53MP+A73MP 的 linux multi-core 系统。

步骤 3. Linux 系统上通过 load_liteos 命令启动 A53UP(Huawei LiteOS)、M7(Huawei LiteOS)，命令参考如下：

load_liteos 0 0x131000000 ./sample_liteos_a53.bin

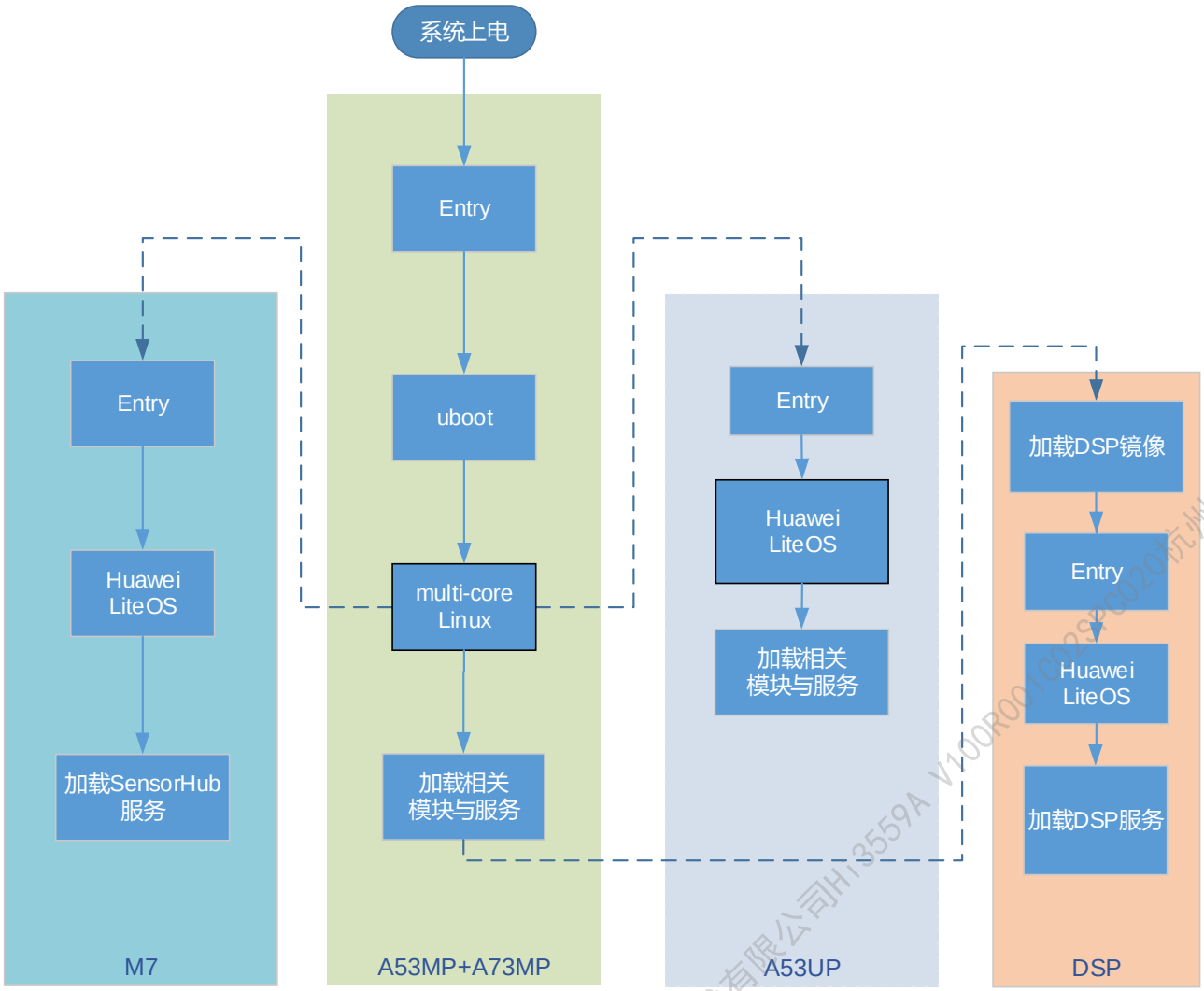
load_liteos 1 0x19000000 ./sample_liteos_m7.bin

步骤 4. linux 系统中启动 DSP(Huawei LiteOS)。

说明

load_liteos 0 0x131000000 ./sample_liteos_a53.bin 中 0x131000000 为默认配置，其值受内存布局影响，如需修改，可参考 9 Hi3559AV100 内存分配.章节介绍。

图5-1 系统启动流程



----结束

5.2 Linux(big-little)(A53MP+A73MP)+Huawei LiteOS(A53UP)

编译 OSDRV 时，AMP_TYPE 传参为 linux_liteos，此参数下生成的生成的 A53UP(Huawei LiteOS)、M7(Huawei LiteOS)镜像支持在 uboot 下加载，加载步骤如下：

步骤 1. 系统上电时，A53MP 首先启动，运行 uboot。

步骤 2. uboot 使用命令“config_m7”配置 M7，再将 M7 镜像写入 0x19000000 地址中，使用命令“go_m7”启动 M7(Huawei LiteOS)（不启动 M7 时可省略）。

步骤 3. uboot 使用命令“go_a53up 0X45000000”启动 A53UP(Huawei LiteOS)（不启动 A53UP 时可省略）。

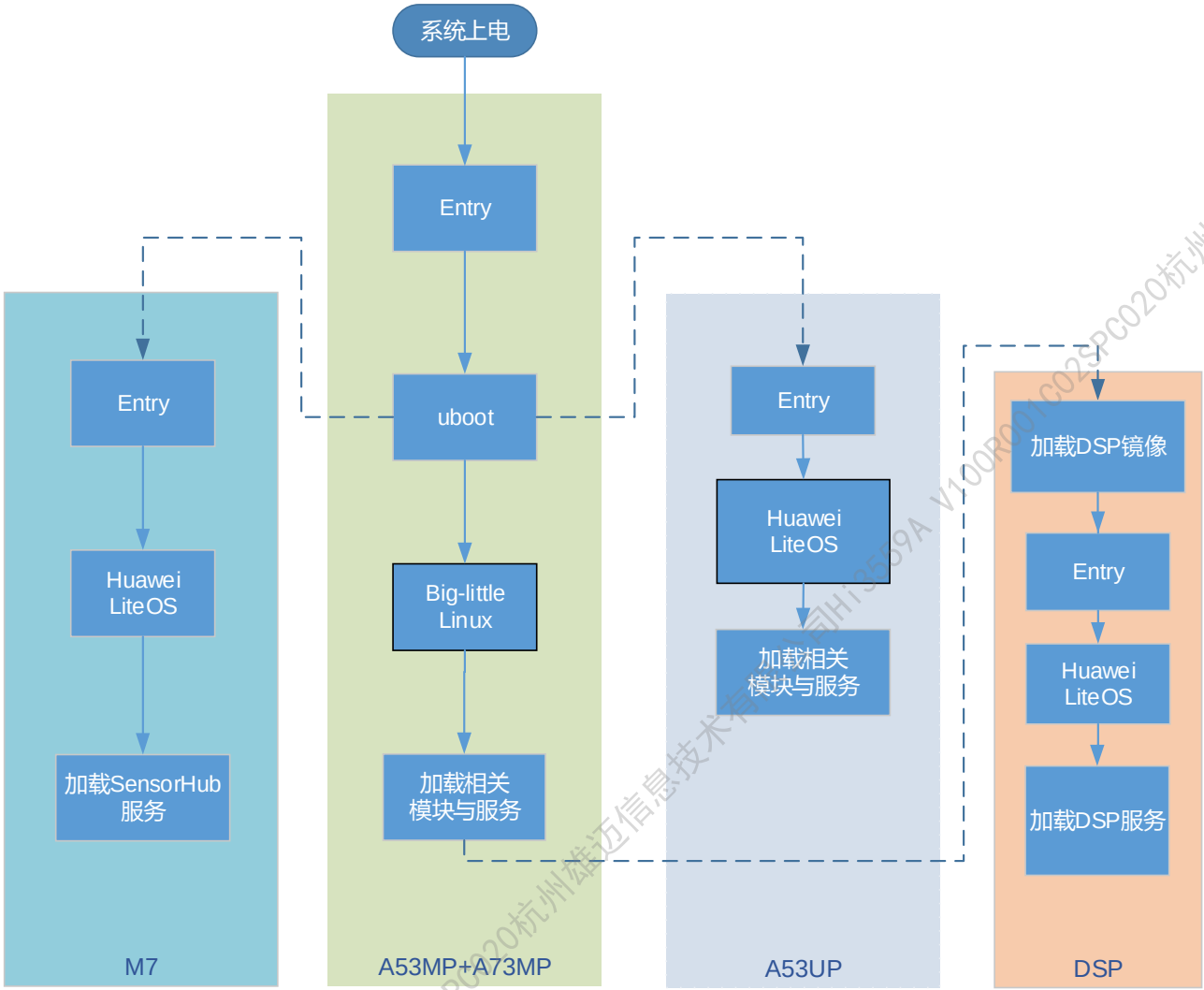
步骤 4. Uboot 下使用“bootm 0Xxxxxxxxx” 命令启动 A53MP+A73MP 的 linux big-little 系统。

步骤 5. linux 系统中启动 DSP(Huawei LiteOS)。

说明

go_a53up 0X45000000 中 0X45000000 为默认配置，其值受内存布局影响，如需修改，可参考 Hi3559AV100 内存分配章节介绍。

图5-2 系统启动流程



----结束

6 Huawei LiteOS

6.1 Huawei LiteOS 配置

Huawei LiteOS 的开发，编译工作首要任务是安装编译工具链。具体安装方法可以参考本文档 1.3.2 安装交叉编译工具。

步骤 1. 切换目录至 Huawei LiteOS 目录下：

cd platform/liteos_a53/liteos

或 cd platform/liteos_m7/liteos

步骤 2. 执行 make menuconfig

配置完成后，保存并退出。

----结束

6.2 Huawei LiteOS 编译

配置保存后，可直接输入“make”命令编译 OS 库文件。

7 应用程序开发简介

7.1 编写代码

用户可根据个人习惯选择代码编写工具。通常在 Windows 环境下使用 Source Insight，在 Linux 环境下使用 Vim+ctags+cscope，功能也相当强大。

7.2 运行应用程序

要运行编译好的应用程序，首先需要将其添加到目标机中，必须完成以下工作：

- 将应用程序和需要的库文件（如果有）等添加到目标机的根文件系统相应的目录中。通常将应用程序放到/bin 目录里，库文件放到/lib 目录里，配置文件则放到/etc 目录里。
- 制作包含新应用程序的根文件系统。

说明

如果执行应用程序，需要读写文件系统操作。请选择 yaffs2、jffs2 文件系统。

如果需要制作 cramfs、yaffs2 或 jffs2 文件系统，制作相应的文件系统（请参见“4.3 文件系统简介”），然后烧写根文件系统到 Flash 指定位置，并设置相应的启动参数。同样，启动 Linux 后便可运行新的应用程序。

说明

如果新添加的应用程序需要系统启动后自动运行，请编辑/etc/init.d/rcS 文件，添加需要启动的应用程序路径。

8 IPCM 模块

8.1 IPCM 简介

IPCM 是 Inter-Processor Communication Module（多核通信模块）的简称。用于实现 A53MP+A73MP、A53UP、DSP、Cortex-M7 等相互之间的通信。

8.2 IPCM 源代码

IPCM 源码路径位于发布包目录下 osdrv/components/ipcm/，用户可直接进入目录进行相关操作。

以下为 IPCM 源代码顶层目录结构：

```
├─── arch
│    └─── hi3559av100
│         └─── configs
├─── class
│    ├─── cdev
│    ├─── net
│    ├─── sharefs
│    └─── virt-tty
├─── include
├─── message
├─── sysdeps
│    ├─── bare
│    ├─── linux
│    └─── liteos
```

```
└──── test
└──── readme_cn.txt
└──── readme_en.txt
└──── Makefile
└──── do_make_module
└──── makeprepare.sh
```

对各个目录或文件的解释如下：

- arch 芯片平台相关
- class ipcm 使用功能组件的封装，包括：cdev（字符设备），sharefs（共享文件系统），virt-tty（虚拟串口终端）
- include 头文件
- message 通信消息层代码
- sysdeps 系统依赖相关，包括：linux，Huawei LiteOS，bare（非操作系统）
- test 测试用例及 sample
- readme 使用说明

8.3 节点分配

IPCM 为每一个需要通信的核分配一个节点，并为之编号，用于建立连接时指定发送消息对端。表 8-1 是 Hi3559AV100 的默认分配方式。

表8-1 IPCM 节点分配

节点号	核	OS
0	A53MP+A73MP	Linux(big-little)
1	A53UP	Huawei LiteOS
2	DSP0	Huawei LiteOS
3	DSP1	Huawei LiteOS
4	DSP2	Huawei LiteOS
5	DSP3	Huawei LiteOS
6	M7	Huawei LiteOS

8.4 IPCM 使用说明

在 IPCM 顶层目录执行：

```
make PLATFORM=hi3559av100 CFG= hi3559av100_xxx_yyyyyy_config
```

其中：hi3559av100_xxx_yyyyyy_config 为配置文件，在 arch/ hi3559av100/configs 下面。

编译完成后，在 out/node 目录下生成目标文件。Linux 使用 ko 文件，Huawei LiteOS 使用库文件。

IPCM 的配置说明，设备节点操作请查看 ipcm 顶层目录 readme.txt 说明。

IPCM 的编译需要依赖 OSDRV 编译完成。OSDRV 编译完成后，在 IPCM 目录，编译示例如下：

- 对于 A53MP+A73MP，执行：

  make PLATFORM=hi3559av100 CFG=hi3559av100_mp_linux_big-little_config all (Linux+ Huawei LiteOS 方案)

  或

  make PLATFORM=hi3559av100 CFG=hi3559av100_mp_linux_multi-core_config all (单 Linux 方案)

  在 out/node_0 目录下生成

  node_0

  ├── hi_ipcm.ko
  ├── hi_virt-tty.ko
  ├── libsharefs.a
  ├── libsharefs.so
  ├── sharefs
  └── virt-tty

- 对于 A53UP，执行：

  make PLATFORM=hi3559av100 CFG=hi3559av100_a53_liteos_config all

  在 out/node_1 目录下生成：

  node_1

  ├── libipcm.a
  ├── libsharefs.a
  └── libvirt-tty.a

- 对于 Cortex-M7，执行：

  make PLATFORM=hi3559av100 CFG=hi3559av100_m7_liteos_config all

  在 out/node_6 目录下生成：

  node_6

  ├── libipcm.a
  └── libvirt-tty.a

8.5 virt-tty 虚拟串口终端

Hi3559AV100 部署了多个操作系统，开发者需要对每个系统调试并查看打印信息。为每个系统配置一个硬件串口，会增加硬件单板布线及成本。为此，提供一套虚拟终端 virt-tty 的解决方案用于调试每个系统。

Virt-tty 作为 IPCM 提供的一套组件之一，代码目录位于：osdrv/components/ipcm/class/virt-tty。配置好 virt-tty 后，在编译 IPCM 时，会同时编译出 virt-tty 的目标文件。Virt-tty 采用 IPCM 的 5 号端口。

virt-tty 采用 Server/Client 模型，Server 端接收 Client 发送的消息并通过一定的方式抛给用户。同时，Server 端接收用户输入的命令、数据等再发送给 Client。一个 Server，并分配一个硬件调试串口，多个 Client，不需要调试串口。整个硬件解决方案只需要一个调试串口。

Virt-tty 在 Hi3559AV100 的典型使用场景为：A53MP+A73MP 作为 Server，A53UP、Cortex-M7、DSP 等作为 Client。其拓扑如图 8-1。操作步骤如下：

步骤 1. A53UP、Cortex-M7 Huawei LiteOS 链接 libipcm.a，libvirt-tty.a 库。并在 app_init 中执行初始化：

_ipcm_vdd_init();

virt_tty_dev_init();

步骤 2. 参考 5 “多核加载启动”启动多系统。

步骤 3. A53MP+A73MP 加载 hi_ipcm.ko，hi_virt-tty.ko。virt-tty 为其应用程序。

步骤 4. A53MP+A73MP 配置好网络连接到 PC，并启用 telnetd 服务。

步骤 5. 在 PC 终端工具上新建 telnet 连接到 A53MP+A73MP。在 telnet 窗口里执行：

virt-tty a53

即可进入 A53UP 的调试控制台。

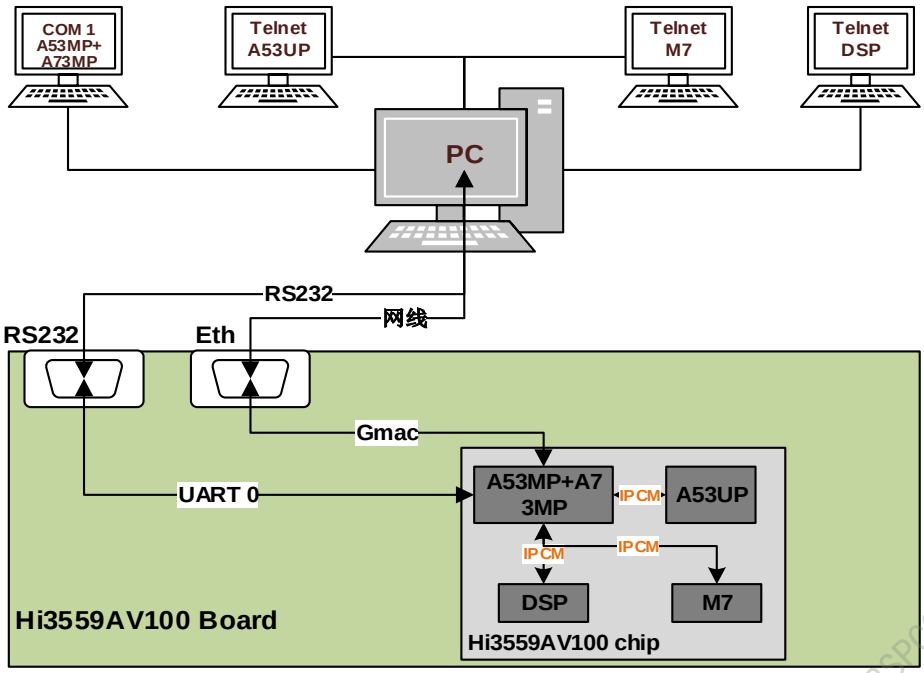
virt-tty dspX （X 为 0、1、2、3）

即可进入 dspX 的调试控制台。

virt-tty m7

即可进入 Cortex-M7 的调试控制台。

图8-1 Hi3559AV100 virt-tty 拓扑



说明

由于 Huawei LiteOS 的 shell 只能接收一个输入控制台，不能共同使用物理串口与 virt-tty 作为 shell 输入。OSDRV 默认配置为 virt-tty，如果需要切换到物理串口（A53UP 默认为 SOC 的 uart 1，M7 默认为 SensorHub 的 uart0），按以下操作：

- 对于 A53UP，打开：

  platform/bsp/board/hi3559av100/cortex-a53_aarch64/include/hisoc/uart.h

- 对于 M7，打开：

  platform/bsp/board/hi3559av100/cortex-m7/include/hisoc/uart.h

  注释掉以下代码：

  ifdef LOSCFG_PLATFORM_HISI_AMP

  undef TTY_DEVICE

  define TTY_DEVICE "/dev/virt-tty"

  endif

  执行：make clean; make 重新编译。

8.6 Sharefs 功能

Sharefs 可使 A53UP Huawei LiteOS 访问 A53MP+A73MP Linux 上目录。其源码目录位于：osdrv/components/ipcm/class/sharefs。Sharefs 采用 IPCM6 号端口。

Sharefs 采用 Server/Client 模型。Server 提供被访问目录，接收 Client 发出的文件访问命令并执行，然后返回结果给 Client。在 Client 端，通过一些基本的文件或目录访问操作（open/read/write/close、cd/ls/stat 等），其实际等同于访问 Server 端对应的目录。

Server 端被指定作为 Sharefs 的访问目录同也可以作为 U 盘、SD 卡、NFS 等的挂载点。这样，Client 可等同地访问 U 盘、SD 卡、NFS 等介质。

操作示例步骤如下：

步骤 1. A53UP Huawei LiteOS 链接 libipcm.a，libsharefs.a 库。并在 app_init 中执行初始化：

_ipcm_vdd_init();

sharefs_client_init("/sharefs");

其入参“/sharefs”即是 Client 端指定要访问 Server 端对应的目录，由用户自定义。

步骤 2. A53MP+A73MP 加载 hi_ipcm.ko，并执行：sharefs &，作为后台程序。也可在用户的程序中链接 libsharefs.a、libsharefs.so 库，并在应用中执行：sharefs_server_init()。

步骤 3. 在 A53UP Huawei LiteOS 可访问 A53MP+A73MP Linux 的目录“/sharefs”。执行：

cd /sharefs 或 ls /sharefs

注意

由 Client 指定的 Sharefs 访问目录（sharefs_client_init 的入参）在 Server 端必须存在，并且能被 Server 端应用 sharefs 访问。不然，Client 会访问失败。

----结束

9 Hi3559AV100 内存分配

9.1 内存分配说明

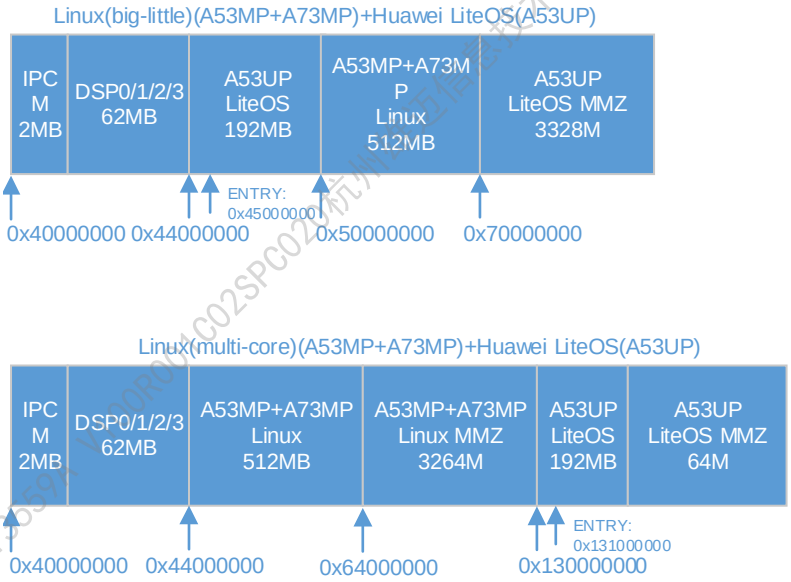
Hi3559AV100 的 DDR 地址空间从 0x40000000 起始。

Hi3559AV100 的 DDR 主要划分为如下几个部分：

- 核间通信共享内存（A53MP+A73MP、A53UP、Cortex-M7、DSP 之间）
- DSP0/1/2/3 Huawei LiteOS 系统内存
- A53MP+A73MP Linux 系统内存
- A53UP Huawei LiteOS 系统内存
- A53MP+A73MP 使用的 MMZ 区域
- A53UP 使用的 MMZ 区域。

图 9-1 是开发包给出的默认参考配置。

图9-1 内存分配配置



说明

- DSP 指令跳转区间必须在 1G 范围之内，由于 DSP 片内 RAM 的物理地址在（0~1GB）范围内，所以 DSP Huawei LiteOS 系统内存需尽可能靠近前面 1GB 空间。
- 可以在 osdrv/osdrv_mem_cfg.sh 文件中配置内存分配，然后编译整个 osdrv。
- 可以通过 osdrv_mem_cfg.sh 中的 DSP_MEM_SIZE、LINUX_MEM_SIZE、LINUX_MMZ_SIZE、LITEOS_TEXT_OFFSET、LITEOS_SYS_MEM_SIZE、LITEOS_MMZ_MEM_BASE、LITEOS_MMZ_MEM_LEN 修改内存布局。
- LITEOS 的启动地址（ENTRY）等于 SYS_MEM_BASE+TEXT_OFFSET，也是启动 A53UP LiteOS 的入口地址（go_a53up 0Xxxxxxxx 或 load_liteos 0 0Xxxxxxxxx ./sample_liteos_a53.bin）。

10 中断分配

10.1 中断配置与分配

说明

- 在 Linux(multi-core)(A53MP+A73MP)+Huawei LiteOS(A53UP)系统布局下，timer6/7、timer8/9、timer10/11、uart1 的中断分配给 A53UP，其余中断分配给 A53MP+A73MP，并默认绑定到 A53MP-0，中断分配对应的配置文件为：linux 目录下 drivers/irqchip/irq-map-hi3559av100.h 文件，用户可更改 irq_map 表，将中断分配给 A53MP0、A53MP1、A73MP0、A73MP1 或 A53UP。
- 在 Linux(big-little)(A53MP+A73MP)+Huawei LiteOS(A53UP)系统布局下，中断分配参考下表，分配给 A53MP+A73MP 的中断，默认绑定到 A53MP-0，中断分配对应的配置文件为：“platform/bsp/board/hi3559av100/cortex-a53_aarch64/include/irq_map.h”。用户可更改 irq_map 表，将中断分配给 A53MP0、A53MP1、A73MP0、A73MP1 或 A53UP。配置完成后，执行：make clean; make 重新编译。

表10-1 Linux(big-little)(A53MP+A73MP)+Huawei LiteOS(A53UP)中断分配表

中断号	中断源	CPU	中断号	中断源	CPU
0~31	CPU 内部中断	-	145	CPU_EXT_TIMER0	A53MP+A73MP
32	Timer0/Timer1	A53MP+A73MP	146	CPU_EXT_TIMER1	A53MP+A73MP
33	Timer2/Timer3	A53MP+A73MP	147	CPU_EXT_TIMER2	A53MP+A73MP
34	Timer4/Timer5	A53MP+A73MP	148	CPU_EXT_TIMER3	A53MP+A73MP
35	Timer6/Timer7	A53UP	149	A53MP_PMU0	A53MP+A73MP
36	Timer8/Timer9	A53UP	150	A53MP_PMU1	A53MP+A73MP
37	Timer10/Timer11	A53UP	151	A53MP_CTIIRQ0	A53MP+A73MP
38	Uart0	A53MP+A73MP	152	A53MP_CTIIRQ1	A53MP+A73MP
39	Uart1	A53UP	153	A53MP_COMMRX0	A53MP+A73MP
40	Uart2	A53MP+A73MP	154	A53MP_COMMRX1	A53MP+A73MP
41	Uart3	A53MP+A73MP	155	A53MP_COMMTX0	A53MP+A73MP
42	Uart4	A53MP+A73MP	156	A53MP_COMMTX1	A53MP+A73MP

中断号	中断源	CPU	中断号	中断源	CPU
43	RTC	A53MP+A73MP	157	A53MP_NCOMMIRQ0	A53MP+A73MP
44	I2C0	A53UP	158	A53MP_NCOMMIRQ1	A53MP+A73MP
45	I2C1	A53UP	159	A73MP_PMU0	A53MP+A73MP
46	I2C2	A53UP	160	A73MP_PMU1	A53MP+A73MP
47	I2C3	A53UP	161	A73MP_CTIIRQ0	A53MP+A73MP
48	I2C4	A53UP	162	A73MP_CTIIRQ1	A53MP+A73MP
49	I2C5	A53UP	163	A73MP_COMMRX0	A53MP+A73MP
50	I2C6	A53UP	164	A73MP_COMMRX1	A53MP+A73MP
51	I2C7	A53UP	165	A73MP_COMMTX0	A53MP+A73MP
52	I2C8	A53MP+A73MP	166	A73MP_COMMTX1	A53MP+A73MP
53	I2C9	A53MP+A73MP	167	A73MP_NCOMMIRQ0	A53MP+A73MP
54	I2C10	A53MP+A73MP	168	A73MP_NCOMMIRQ1	A53MP+A73MP
55	I2C11	A53UP	169	A53UP_PMU	A53UP
56	IR	A53MP+A73MP	170	A53UP_CTIIRQ0	A53UP
57	FMC	A53MP+A73MP	171	A53UP_COMMRX0	A53UP
58	eMMC	A53MP+A73MP	172	A53UP_COMMTX0	A53UP
59	VDMA	A53UP	173	A53UP_NCOMMIRQ	A53UP
60	保留	-	174	PCIE_PM_INT	A53MP+A73MP
61	保留	-	175	PCIE_INTA	A53MP+A73MP
62	SPACC	A53MP+A73MP	176	PCIE_INTB	A53MP+A73MP
63	SSP0	A53UP	177	PCIE_INTC	A53MP+A73MP
64	SSP1	A53UP	178	PCIE_INTD	A53MP+A73MP
65	SSP2	A53UP	179	PCIE_EDMA_INT	A53MP+A73MP
66	SSP3	A53UP	180	PCIE_MSI_INT	A53MP+A73MP
67	SSP4	A53UP	181	PCIE_LINK_DOWN_INT	A53MP+A73MP
68	GMAC0	A53MP+A73MP	182	PCIE_CFG_LINK_AUTO_BW_INT	A53MP+A73MP
69	GMAC1	A53UP	183	PCIE_CFG_BW_MGT_INT	A53MP+A73MP
70	SOFTWARE	A53MP+A73MP	184	AVS_INT0	A53UP
71	VEDU0	A53UP	185	AVS_INT1	A53UP

中断号	中断源	CPU	中断号	中断源	CPU
72	VEDU1	A53UP	186	AVS_INT2	A53UP
73	VEDU2	A53UP	187	AVS_INT3	A53UP
74	保留	-	188	AVS_INT4	A53UP
75	VGS0	A53UP	189	AVS_INT5	A53UP
76	VGS1	A53UP	190	AVS_INT6	A53UP
77	VPSS0	A53UP	191	AVS_INT7	A53UP
78	VPSS1	A53UP	192	GPIO0	A53MP+A73MP
79	GDC0	A53UP	193	GPIO1	A53MP+A73MP
80	GDC1	A53UP	194	GPIO2	A53MP+A73MP
81	JPGE	A53UP	195	GPIO3	A53MP+A73MP
82	保留	-	196	GPIO4	A53MP+A73MP
83	GME	A53UP	197	GPIO5	A53MP+A73MP
84	JPGD	A53UP	198	GPIO6	A53MP+A73MP
85	TDE	A53MP+A73MP	199	GPIO7	A53MP+A73MP
86	GZIP	A53MP+A73MP	200	GPIO8	A53MP+A73MP
87	PGD	A53MP+A73MP	201	GPIO9	A53MP+A73MP
88	IVE	A53MP+A73MP	202	GPIO10	A53MP+A73MP
89	保留	-	203	GPIO11	A53MP+A73MP
90	NNIE0	A53MP+A73MP	204	GPIO12	A53MP+A73MP
91	NNIE1	A53MP+A73MP	205	GPIO13	A53MP+A73MP
92	VICAP	A53UP	206	GPIO14	A53MP+A73MP
93	VIPROC0	A53UP	207	GPIO15	A53MP+A73MP
94	VIPROC1	A53UP	208	GPIO16	A53MP+A73MP
95	MIPI_TX_INT	A53UP	209	GPIO17	A53MP+A73MP
96	HDMI_TX_AON	A53UP	210	GPIO18	A53MP+A73MP
97	HDMI_TX_PWD	A53UP	211	SHUB_PWR_START	A53MP+A73MP
98	HDMI_TX_SEC	A53UP	212	SHUB_GPIO0_INT	A53MP+A73MP
99	VDP0	A53UP	213	SHUB_GPIO1_INT	A53MP+A73MP
100	VDP1	A53MP+A73MP	214	SHUB_GPIO2_INT	A53MP+A73MP
101	AIAO	A53UP	215	SHUB_GPIO3_INT	A53MP+A73MP

中断号	中断源	CPU	中断号	中断源	CPU
102	DSP0	A53MP+A73MP	216	SHUB_GPIO4_INT	A53MP+A73MP
103	DSP1	A53MP+A73MP	217	SHUB_UART0_INT	A53MP+A73MP
104	DSP2	A53MP+A73MP	218	SHUB_UART1_INT	A53MP+A73MP
105	DSP3	A53MP+A73MP	219	SHUB_UART2_INT	A53MP+A73MP
106	SDIO0	A53MP+A73MP	220	SHUB_UART3_INT	A53MP+A73MP
107	SDIO1	A53MP+A73MP	221	SHUB_UART4_INT	A53MP+A73MP
108	SDIO2	A53MP+A73MP	222	SHUB_I2C0_INT	A53MP+A73MP
109	SDIO0_WAKE	A53MP+A73MP	223	SHUB_I2C1_INT	A53MP+A73MP
110	SDIO1_WAKE	A53MP+A73MP	224	SHUB_I2C2_INT	A53MP+A73MP
111	SDIO2_WAKE	A53MP+A73MP	225	SHUB_I2C3_INT	A53MP+A73MP
112	DDRPHY0|DDRPHY1|DDR_ERR_INT	A53MP+A73MP	226	SHUB_I2C4_INT	A53MP+A73MP
113	DMAC0	A53UP	227	SHUB_I2C5_INT	A53MP+A73MP
114	DMAC1	A53MP+A73MP	228	SHUB_I2C6_INT	A53MP+A73MP
115	UFS	A53MP+A73MP	229	SHUB_I2C7_INT	A53MP+A73MP
116	USB_P0	A53MP+A73MP	230	SHUB_SPI0_INT	A53MP+A73MP
117	USB_P1	A53MP+A73MP	231	SHUB_SPI1_INT	A53MP+A73MP
118	SLVS-EC	A53UP	232	SHUB_SPI2_INT	A53MP+A73MP
119	保留	-	233	SHUB_CAN_INT	A53MP+A73MP
120	MIPI_RX	A53UP	234	SHUB_DMAC_INT	A53MP+A73MP
121	DDRT0	A53MP+A73MP	235	保留	-
122	DDRT1	A53MP+A73MP	236	保留	-
123	VDH_OLP	A53UP	237	保留	-
124	VDH_ILP	A53UP	238	SHUB_LSADC	A53MP+A73MP
125	VDH_SCD_SAFE	A53UP	239	TRNG	A53MP+A73MP
126	VDH_SCD_NORMAL	A53UP	240	DPU_RECT	A53UP
127	VDH_MDMA	A53UP	241	DPU_MATCH	A53UP
128	SSP5	A53MP+A73MP	242	DPU_POSTPROC	A53UP
129	SSP6	A53MP+A73MP	243	IPC_INT0	A53MP+A73MP
130	A53UP_CCI_INT	A53UP	244	IPC_INT1	A53MP+A73MP

中断号	中断源	CPU	中断号	中断源	CPU
131	BL_CCI_INT	A53MP+A73MP	245	IPC_INT2	A53UP
132	GPU_IRQEVENT	A53MP+A73MP	246	保留	-
133	GPU_IRQGPU	A53MP+A73MP	247	保留	-
134	GPU_IRQJOB	A53MP+A73MP	248	保留	-
135	GPU_IRQMMU	A53MP+A73MP	249	保留	-
136	RSA	A53MP+A73MP	250	保留	-
137	WDG0|WDG1|WDG2	A53MP+A73MP	251	保留	-
138	SAR_ADC_INT	A53MP+A73MP	252	保留	-
139	CAN0	A53MP+A73MP	253	保留	-
140	CAN1	A53MP+A73MP	254	保留	-
141	保留	-	255	保留	-
142	保留	-	246	保留	-
143	保留	-	247	保留	-
144	保留	-	248	保留	-

A 缩略语

A

ARM	Advanced RISC Machine	ARM 公司指令集

C

CRAMFS	Compressed RAM file system	压缩 RAM 文件系统

D

DMS	Digital Media Solution	媒体解决方案平台

E

ELF	Executable and Linkable Format	可执行连接格式文件

G

GCC	GNU Complier Collection	GNU 编译器集合
GNU	GNU’s Not UNIX	GNU

I

IP	Internet Porotocol	Internet 协议

J

jffs2	Journalling FLASH File System v2	一种 Flash 文件系统
JTAG	Joint Test Action Group	联合测试行动组

P

PC	Personal Computer	个人计算机

S

SDRAM	Synchronous Dynamic Random Access Memory	同步动态随机存储器
SDK	Software Development Kit	软件开发工具集

Y

yaffs2	Yet Another Flash File System	一直专门针对 NAND 闪存设计的文件系统