

Hi3798MV100 SDK编译环境搭建与嵌入式Linux系统构建全流程指南

原创 👑VIP免费 于 2026-08-05 09:13:24 发布 · 312 阅读 · 👍13 · 🌟5

1. 从零开始的Hi3798MV100 SDK编译环境搭建

最近在折腾一块基于海思Hi3798MV100芯片的机顶盒子板子，型号是创维E900V21C。这玩意儿在海鲜市场很常见，几十块钱就能淘到，硬件配置对于学习嵌入式Linux开发来说性价比极高。我拿到手的第一件事，就是想把官方的SDK编译环境给搭起来，毕竟这是后续所有开发工作的基石。但这个过程，远比想象中要“复古”和“折腾”。如果你也正准备入坑Hi3798MV100，或者任何一款老版本海思芯片的开发，这篇笔记或许能帮你避开我踩过的那些坑。

Hi3798MV100是一颗经典的机顶盒SoC，虽然现在看其CPU性能和外围接口不算突出，但胜在资料相对丰富（以海思的标准而言），社区也有不少玩家在研究。编译它的SDK，核心目的就是为了生成三个关键的东西：U-Boot引导程序、Linux内核以及根文件系统。有了它们，你才能让板子真正跑起来，并在此基础上进行应用开发或系统定制。整个编译过程强烈依赖于一个特定的、老旧的交叉编译工具链和一套定制化的构建脚本，对宿主机的Linux发行版和软件版本有近乎苛刻的要求。网上很多教程直接跳到编译命令，但环境没配好，一切都是徒劳。所以，我们从头说起。

2. 宿主机操作系统与软件版本的精确选择

这是整个过程中最至关重要的一步，选错了系统，后面会寸步难行。根据官方SDK（通常版本号类似Hi3798MV100R001C01SPCxxx）的文档和无数前辈的血泪教训，最兼容、问题最少的宿主机系统是 **Ubuntu 14.04 64位**。是的，你没看错，是2014年发布的那个版本。

为什么必须是它？原因在于SDK内置的交叉编译工具链（通常是 `arm-hisiv300-linux` 或 `arm-hisiv400-linux`）是在特定的glibc库版本下构建的。在新版本的Ubuntu（如18.04, 20.04）上，系统的动态链接库版本过高，会导致工具链无法运行，报各种“`No such file or directory`”或“`version GLIBC_2.xx' not found`”的错误。试图去升级工具链或修改系统库，复杂度极高，极易失败。

那么，有没有其他选择？ 有，但都是妥协方案：

- 使用Docker容器**：这是目前最优雅的解决方案。创建一个基于Ubuntu 14.04镜像的Docker容器，在容器内进行所有编译工作。这样既能保持宿主机系统的洁净和现代，又能满足SDK的古老需求。我强烈推荐这种方式。
- 使用老旧的物理机或虚拟机**：老老实实安装一个Ubuntu 14.04的虚拟机。VMware或VirtualBox都可以。确保分配足够的磁盘空间（建议40GB以上）和内存（4GB以上）。
- 尝试在较新系统上“打补丁”**：有人通过手动安装旧版本的 `libc6:i386` 等32位兼容库，或者使用 `chroot` 构造一个旧环境，让工具链跑起来。这个过程非常繁琐，不推荐新手尝试。

我的选择是Docker方案。接下来，所有步骤都基于在Ubuntu 14.04 Docker容器内操作来展开。如果你用虚拟机，操作完全一致。

3. 获取SDK与交叉编译工具链

海思的SDK通常不是公开下载的，需要一定的渠道获取。对于Hi3798MV100这类消费级芯片，SDK可能会随着某些开发板流出，或者在相关的技术社区找到分享链接（请注意版权风险）。假设你已经拿到了一个名为 `Hi3798MV100R001C01SPCxxx.tgz` 的SDK包。

第一步：准备Docker环境（宿主机为现代Linux或Windows WSL2） 在宿主机上，拉取Ubuntu 14.04镜像并启动一个容器。

```
bash
# 拉取镜像
docker pull ubuntu:14.04

# 运行容器，将本地SDK目录挂载到容器内，并创建一个用于编译的用户
docker run -it --name hi3798mv100-build \
-v /path/to/your/sdk:/home/sdk \
-v /path/to/your/output:/home/output \
ubuntu:14.04 /bin/bash
```

进入容器后，首先更新软件源并安装一些基础工具，比如 `vim`，`make`，`gcc`，`g++`（注意这是宿主机的gcc，用于编译一些本地工具），以及 `git` 等。

```
bash
apt-get update
apt-get install -y vim make gcc g++ git wget cpio python
```

第二步：解压SDK 在容器内，进入挂载的目录。

```
bash
cd /home/sdk
tar -xzf Hi3798MV100R001C01SPCxxx.tgz
```

解压后，你会看到一个复杂的目录结构，通常包含以下关键部分：

- `pub/`：存放公版镜像、文档等。
- `source/`：最核心的源码目录，包含 `boot`（U-Boot）、`kernel`（Linux内核）和 `rootfs`（根文件系统构建脚本和配置）。

- `tools/`：包含交叉编译工具链、PC端工具（如 `mkimage`）。
- `build/`：存放编译脚本，如 `build.sh`。

第三步：安装交叉编译工具链 工具链通常在 `tools/linux/toolchain/` 目录下。对于Hi3798MV100，常见的是 `arm-hisiv300-linux` 或 `arm-hisiv400-linux`。

bashAI写代码 | 复制

```
cd /home/sdk/tools/linux/toolchain/  
# 假设工具链包是 arm-hisiv300-linux.tar.gz  
tar -xzf arm-hisiv300-linux.tar.gz -C /opt/
```

解压到 `/opt` 目录后，需要将工具链路径加入到系统的 `PATH` 环境变量中。通常SDK的 `build` 目录下会有一个 `env_setup.sh` 之类的脚本。如果没有，需要手动设置。

bashAI写代码 | 复制

```
# 编辑当前用户的bashrc文件  
vim ~/.bashrc  
# 在文件末尾添加  
export PATH="/opt/arm-hisiv300-linux/bin:$PATH"  
export ARCH=arm  
export CROSS_COMPILE=arm-hisiv300-linux-  
# 使环境变量生效  
source ~/.bashrc  
# 验证工具链是否安装成功  
arm-hisiv300-linux-gcc -v
```

如果成功输出gcc版本信息，恭喜你，最困难的环境部分已经完成了一大半。

4. 编译U-Boot引导程序

U-Boot相当于嵌入式板子的BIOS，负责初始化硬件、加载内核。海思的U-Boot通常经过了深度定制。

第一步：进入U-Boot源码目录并配置

bashAI写代码 | 复制

```
cd /home/sdk/source/boot/u-boot-2010.06  
# 先执行一次清理  
make distclean  
# 应用默认配置文件，不同板子配置不同，常见 hi3798mv100_defconfig  
make hi3798mv100_defconfig
```

这里的 `defconfig` 文件定义了该板型的所有编译选项。如果你是从其他板型移植，可能需要基于一个相近的配置进行修改。关键配置位于 `.config` 文件中，但通常通过 `make menuconfig` 来调整。不过海思的U-Boot菜单配置可能被简化，很多关键参数（如DDR初始化参数、寄存器配置）直接写死在板级头文件或 `boards.cfg` 里。

第二步：编译生成U-Boot镜像

bashAI写代码 | 复制

```
make -j4
```

`-j4` 表示使用4个线程并行编译，可以加快速度。编译成功后，在源码目录下会生成关键的镜像文件，通常名为 `u-boot.bin`。但海思平台通常还需要一个步骤，就是用 `tools` 目录下的 `mkboot.sh` 或 `sign` 工具对这个 `u-boot.bin` 进行打包和签名（特别是高安芯片），生成最终的 `u-boot-xxx.bin`。**这里有一个大坑：** Hi3798MV100有高安（Secure Boot）和非高安（Non-Secure Boot）版本。你手里的板子如果是运营商的定制版，很可能是高安版，直接用普通编译的U-Boot是启动不了的。创维E900V21C这类盒子，很多是非高安版，相对简单。务必确认你的芯片版本。

实操心得： 编译U-Boot时最常见的错误是找不到交叉编译工具链。请反复检查 `CROSS_COMPILE` 环境变量是否正确，以及工具链的 `bin` 目录是否在 `PATH` 中。另一个常见错误是缺少某些头文件，比如 `asm/arch/xxx.h`，这通常是因为源码目录结构不对，或者没有正确应用海思提供的补丁。确保你是从海思提供的完整 `source` 目录开始编译，而不是从官方的U-Boot官网下载代码。

5. 编译Linux内核

内核编译的步骤相对标准，但海思的[内核源码](#)树里集成了大量的海思平台专用驱动和模块。

第一步：内核配置

bashAI写代码 | 复制

```
cd /home/sdk/source/kernel/linux-3.18.y  
# 清理  
make distclean  
# 导入默认配置，海思通常提供 .config 文件，或者通过 defconfig  
cp arch/arm/configs/hi3798mv100_defconfig .config  
# 启动图形化配置界面（需要容器内安装 libncurses5-dev）
```

```
apt-get install libncurses5-dev
make ARCH=arm CROSS_COMPILE=arm-hisiv300-linux- menuconfig
```

在 `menuconfig` 中，你需要根据你的板子硬件情况调整配置。关键点包括：

- **系统类型 (System Type)** ：确保选中 `Hisilicon Hi3798MV100` 系列。
- **设备驱动 (Device Drivers)** ：重点关注 `MMC/SD/SDIO card support` 、 `USB support` 、 `Network device support` （特别是以太网PHY驱动，如 `Hisilicon Gigabit Ethernet` ）、 `Graphics support` （GPU驱动，如 `Mali` 或 `PowerVR` ，但海思可能用自家的 `hi_fb` ）。
- **文件系统 (File systems)** ：确保支持你根文件系统将要使用的格式，如 `ext2/3/4` ， `squashfs` ， `jffs2` 等。
- 海思特有的驱动，如 `HISILICON VPU`（视频编解码）、 `HISILICON GPIO` 等，通常已经以模块（ `M` ）或内置（ `*` ）形式配置好。

对于初次编译，如果你不确定，可以直接保存默认配置退出。

第二步：编译内核与模块

bashAI写代码 | 复制

```
# 编译内核镜像和模块
make ARCH=arm CROSS_COMPILE=arm-hisiv300-linux- uImage -j4
# 编译设备树文件（如果使用）
make ARCH=arm CROSS_COMPILE=arm-hisiv300-linux- dtbs -j4
# 编译所有模块
make ARCH=arm CROSS_COMPILE=arm-hisiv300-linux- modules -j4
```

编译成功后，关键产出在：

- `arch/arm/boot/uImage` ：压缩的内核镜像，U-Boot可以直接引导。
- `arch/arm/boot/dts/hi3798mv100-xxx.dtb` ：设备树二进制文件（如果使用设备树）。
- 各个模块文件（ `.ko` ）散落在各驱动目录。

第三步：安装模块到临时目录 为了后续打包进根文件系统，需要将编译好的模块安装到一个临时目录。

bashAI写代码 | 复制

```
# 创建一个临时目录
mkdir -p /home/output/rootfs_modules
# 安装模块，INSTALL_MOD_PATH指定安装路径
make ARCH=arm CROSS_COMPILE=arm-hisiv300-linux- modules_install INSTALL_MOD_PATH=/home/output/rootfs_modules
```

执行后，模块会被安装到 `/home/output/rootfs_modules/lib/modules/3.18.xx/` 目录下。

踩坑记录：内核版本与工具链的匹配 我遇到过最诡异的问题是，编译内核时一切顺利，但编译某些外设驱动（特别是第三方驱动）时，提示函数未定义或内核头文件不匹配。这往往是因为内核源码版本与交叉编译工具链所依赖的内核头文件版本不一致。海思SDK提供的工具链和内核源码是匹配的，但如果你尝试升级内核版本，就必须同步使用匹配的新工具链，或者手动指定工具链内的 `-I` 头文件路径。对于新手，强烈建议 **不要轻易修改内核版本**，就用SDK里提供的那个。

6. 构建根文件系统

根文件系统包含了Linux启动后所有的用户态程序、库文件和配置文件。海思SDK通常提供两种构建方式：一种是基于 `busybox` 制作最小根文件系统，另一种是使用 `buildroot` 或 `yocto` 构建更复杂的系统。这里以最常见的 `busybox` 方式为例。

第一步：编译BusyBox `BusyBox` 是一个集成了上百个常用Linux命令的单一可执行文件，是嵌入式根文件系统的核心。

bashAI写代码 | 复制

```
cd /home/sdk/source/rootfs/busybox-1.xx.x
# 清理
make distclean
# 导入海思提供的默认配置，通常叫 hi3798mv100_defconfig
make hi3798mv100_defconfig
# 可以进行自定义配置
make ARCH=arm CROSS_COMPILE=arm-hisiv300-linux- menuconfig
```

在 `menuconfig` 中，主要关注：

- `Settings -> Build Options -> Build BusyBox as a static binary` ：是否编译为静态链接。静态链接的BusyBox不依赖动态库，更简单，但体积稍大。动态链接则需要将工具链的动态库复制到根文件系统。
- 在 `Settings -> Installation Options` 中，确认 `Don't use /usr` 被选中，这样安装时会直接安装到 `_install` 目录，而不是系统的 `/usr` 下。配置完成后，开始编译和安装：

bashAI写代码 | 复制

```
make ARCH=arm CROSS_COMPILE=arm-hisiv300-linux- -j4
make ARCH=arm CROSS_COMPILE=arm-hisiv300-linux- install
```

编译安装后，当前目录下会生成一个 `_install` 目录，里面就是BusyBox生成的所有命令的链接。

第二步：组装根文件系统骨架 我们需要创建一个目录作为根文件系统的根。

bashAI写代码 | 复制

```
mkdir -p /home/output/rootfs
cd /home/output/rootfs
# 复制BusyBox的内容
cp -r /home/sdk/source/rootfs/busybox-1.xx.x/_install/* .
# 创建Linux系统必需的目录
mkdir -p proc sys dev etc/init.d tmp var/lib mnt home root lib
```

第三步：复制动态库 如果你的BusyBox是动态链接的，或者你后续要运行动态链接的应用程序，就必须将交叉工具链里的动态库复制过来。

bashAI写代码 | 复制

```
# 找到工具链的动态库路径，通常在 /opt/arm-hisiv300-linux/arm-hisiv300-linux/libc/lib
cp -d /opt/arm-hisiv300-linux/arm-hisiv300-linux/libc/lib/*.so* ./lib/
# -d 参数保持软链接
```

第四步：添加内核模块 将之前编译好的内核模块复制过来。

bashAI写代码 | 复制

```
cp -r /home/output/rootfs_modules/lib/modules ./lib/
```

第五步：创建基本的配置文件 最重要的文件是 `etc/inittab` 和 `etc/init.d/rcS` 。

- `etc/inittab`：定义系统启动后要运行的进程。

bashAI写代码 | 复制

```
# 内容示例
::sysinit:/etc/init.d/rcS
::respawn:-/bin/sh
ttyAMA0::askfirst:-/bin/sh
::ctrlaltdel:/sbin/reboot
```

- `etc/init.d/rcS`：系统初始化脚本，需要赋予可执行权限。

bashAI写代码 | 复制

```
#!/bin/sh
# 内容示例
mount -t proc none /proc
mount -t sysfs none /sys
mount -t tmpfs none /tmp
echo /sbin/mdev > /proc/sys/kernel/hotplug
mdev -s
# 设置网络（如果需要）
ifconfig lo 127.0.0.1 up
# 加载必要的内核模块
insmod /lib/modules/3.18.xx/kernel/drivers/xxx.ko
```

第六步：制作根文件系统镜像 最后，需要将这一堆目录和文件打包成一个镜像文件，方便烧录。常见的格式有 `ext4` 和 `squashfs` 。

bashAI写代码 | 复制

```
# 制作ext4格式镜像（可读写）
cd /home/output
dd if=/dev/zero of=rootfs.ext4 bs=1M count=64 # 创建一个64MB的空文件
mkfs.ext4 rootfs.ext4
mkdir -p mnt_point
sudo mount -o loop rootfs.ext4 mnt_point
sudo cp -r rootfs/* mnt_point/
sudo umount mnt_point
# 制作squashfs格式镜像（只读，压缩率高）
mksquashfs rootfs rootfs.squashfs -comp xz
```

`squashfs` 镜像体积更小，适合存储空间有限的SPI NAND Flash。`ext4` 则适合eMMC或SD卡。

7. 整合与烧录：生成最终固件

现在，我们有了三个核心部件：`u-boot.bin`、`uImage`（内核）和 `rootfs.img`（根文件系统镜像）。但通常机顶盒的烧录文件是一个完整的 `.img` 包，它包含了分区表以及各个分区（bootloader、kernel、rootfs等）的映像。海思SDK的 `tools` 目录下通常提供了打包工具，比如 `mkimage` 或专门的 `pack` 脚本。

第一步：了解分区表 Hi3798MV100的典型NAND Flash分区可能如下：

- `bootargs`：启动参数分区。
- `boot`：存放U-Boot的分区。
- `kernel`：存放Linux内核镜像的分区。
- `rootfs`：存放根文件系统的分区。
- `misc`、`recovery` 等。

你需要一个描述分区大小和布局的配置文件，有时叫 `partition.xml` 或直接在打包脚本里定义。

第二步：使用打包脚本 进入SDK的 `build` 或 `tools/pack` 目录，通常能找到 `build.sh` 或 `mkimage.sh` 。

bashAI写代码 | 复制

```
cd /home/sdk/build
# 修改脚本中的路径，指向你编译好的文件
# 通常需要设置环境变量或修改脚本里的路径变量，例如：
# export UB00T_PATH=/home/output/u-boot.bin
# export KERNEL_PATH=/home/output/uImage
# export R00TFS_PATH=/home/output/rootfs.squashfs
./build.sh
```

执行成功后，会在 `output` 或 `image` 目录下生成最终的 `Hi3798MV100R001C01SPCxxx.img` 文件。

第三步：烧录到板子 烧录方式取决于板子的启动模式和接口：

- 串口烧录（最常用）**：板子通过串口连接电脑，进入U-Boot命令行。使用 `tftp` 或 `xmodem` 协议将镜像下载到内存，然后使用 `nand write` 或 `mmc write` 命令写入存储。
- USB烧录**：海思提供了 `Hitool` 工具，通过USB OTG口进行烧录。你需要将板子设置为 `USB Burning` 模式（通常通过短接Flash引脚或按特定按键上电），然后在Windows电脑上用Hitool选择镜像文件进行烧写。
- SD卡烧录**：将镜像写入SD卡，插入板子，设置从SD卡启动，板子会自动将SD卡中的镜像烧写到内部Flash。

最后的忠告：备份原机固件 在第一次烧录你自己编译的固件前，**务必** 先通过Hitool或其他方式，完整地读取并备份原机的整个Flash内容。这是你救砖的最后保障。一旦新固件无法启动，你可以通过短接Flash进入强制烧录模式，将原厂固件写回去。

整个Hi3798MV100的SDK编译流程，就像在操作一个精密的古董仪器，每一步都需要严格按照“手册”来，对版本极度敏感。但一旦走通这个流程，你对嵌入式Linux从源码到镜像的构建全过程，会有一个非常扎实的理解。这不仅仅是敲几个命令，更是对工具链、内核配置、根文件系统组织、固件打包这一整套工具链的熟悉。

标签 [#Hi3798MV100](#) [#SDK编译](#) [#交叉编译工具链](#)