

Hi3798MV100 SDK编译环境搭建全攻略：从Ubuntu配置到固件生成

原创 VIP免费 于 2026-08-05 14:33:09 发布 · 246 阅读 · 👍 10 · 🌟 9

1. 项目概述：从零开始构建Hi3798MV100开发环境

最近在折腾一个老盒子，创维E900V10C，芯片是海思的Hi3798MV100。这玩意儿现在网上资源不少，很多玩家拿来刷机当个电视盒子或者轻量级服务器用。我手头这台是“非高安”版本的，理论上可玩性更高。折腾的第一步，也是最基础、最关键的一步，就是搞定它的SDK编译环境。这就像你要盖房子，总得先有砖头、水泥和图纸，SDK就是这一整套“建筑材料”和“施工手册”。对于Hi3798MV100这类嵌入式芯片，官方SDK里包含了交叉编译工具链、内核源码、驱动、库文件以及构建系统，只有成功编译出完整的固件，后续的定制、调试和应用开发才有根基。

很多新手拿到SDK包，照着网上零散的教程操作，经常会卡在环境依赖、版本冲突或者莫名其妙的编译错误上，一折腾就是好几天。这篇笔记，我就结合自己最近在Ubuntu 20.04 LTS系统上，从零搭建Hi3798MV100 SDK编译环境的全过程，把每一步的细节、踩过的坑和解决方案都捋清楚。目标很明确：让你能在一台干净的Linux系统上，成功跑通 `make all`，生成可以烧录的固件镜像。这不仅适用于E900V10C，对于其他采用同款芯片的盒子（比如文中提到的E900，E900S，E910，E950等）也有参考价值，毕竟底层芯片和SDK框架是相通的。

2. 开发环境准备与SDK解压

编译嵌入式Linux系统，一个稳定、纯净的Linux宿主环境是首要条件。Windows下的WSL或者虚拟机方案也可以，但为了减少路径、权限和性能上的潜在问题，我强烈建议使用物理机安装的Ubuntu系统。我这里用的是Ubuntu 20.04.6 LTS，这是一个长期支持版本，软件源稳定，社区支持好。不建议用太新的发行版，以免遇到一些老版本构建工具（比如 `make 3.8x`）的兼容性问题。

2.1 安装必要的编译工具和依赖库

在开始之前，我们需要安装一整套基础编译工具和针对海思平台可能需要的依赖库。打开终端，一次性安装以下软件包：

bash

AI代码解释 | 复制

```
sudo apt-get update
sudo apt-get install -y build-essential git-core libncurses5-dev u-boot-tools \
    bison flex texinfo zip unzip zlib1g-dev libc6-i386 lib32z1 lib32stdc++6 \
    gcc-multilib g++-multilib cmake pkg-config wget cpio python2.7 python3 \
    python3-pip device-tree-compiler lzop
```

这里有几个关键点需要注意：

- `libc6-i386`，`lib32z1`，`lib32stdc++6`：海思的交叉编译工具链通常是32位的，在64位系统上运行需要这些32位兼容库。缺少它们会导致运行工具链时报“No such file or directory”或“wrong ELF class”的错误，这个坑我踩过。
- `python2.7`：很多老的SDK构建脚本仍然依赖Python 2。尽管Python 2已经停止维护，但为了兼容性必须安装。系统中可以同时存在Python 2和Python 3，通常 `python` 命令默认链接到 `python3`，而 `python2` 命令需要显式指定。有些SDK脚本头一行可能是 `#!/usr/bin/python`，如果它指向了Python 3可能会出错，必要时需要修改脚本或建立软链接。
- `u-boot-tools`，`device-tree-compiler`：用于处理U-Boot镜像和设备树（Device Tree）编译，是生成启动镜像的关键。
- `make` 版本：运行 `make -v` 确认版本。海思SDK通常对 `make` 版本不敏感，但保持3.8x或4.x的稳定版本即可。

注意： 安装完成后，建议重启系统或重新登录，以确保所有新安装的库文件路径被正确加载到当前会话中。

2.2 获取并解压Hi3798MV100 SDK

海思的SDK通常不是开源发布的，需要从芯片供应商或特定的开发者社区获取。对于Hi3798MV100这类消费级芯片的SDK，往往可以在一些智能电视盒子、IPTV机顶盒的开发者论坛或开源项目（如Armbian、CoreELEC针对特定盒子的移植项目）中找到。文件名可能类似于 `Hi3798MV100_SDK_V1.0.0.tgz` 或 `hi3798mv100-xxx.tar.bz2`。

假设你已经下载了SDK压缩包 `hi3798mv100_sdk.tar.gz`。我们将其解压到用户目录下的一个工作空间：

bash

AI代码解释 | 复制

```
cd ~
mkdir -p projects/hi3798mv100
cd projects/hi3798mv100
tar -xzvf /path/to/your/hi3798mv100_sdk.tar.gz
```

解压后，你会看到一个包含多个目录的SDK根目录，结构通常如下：

AI代码解释 | 复制

```
hi3798mv100_sdk/
├── build/           # 构建脚本和配置
├── kernel/          # Linux内核源码（可能是打补丁的版本）
├── boot/            # U-Boot引导程序源码
└── tools/           # 交叉编译工具链、打包工具等
```

```
|— app/          # 示例应用程序
|— mpp/          # 媒体处理平台（Media Process Platform）组件
|— osdrv/        # 操作系统驱动相关（可能包含更广泛的开源组件）
|— release/      # 编译输出目录（初始可能是空的）
```

重要步骤：设置环境变量 SDK的编译严重依赖环境变量，尤其是交叉编译工具链的路径。在SDK根目录下，通常存在一个名为 `env_setup.sh` 或 `setenv.sh` 的脚本。在开始编译前，必须先 `source` 这个脚本：

bash

AI代码解释 | 复制

```
cd hi3798mv100_sdk
source ./env_setup.sh
```

执行这个命令后，它会做几件关键事情：

- 将SDK自带的交叉编译工具链路径（如 `tools/linux/toolchain/arm-hisiv500-linux/bin` ）添加到系统的 `PATH` 环境变量最前面。这样，当你运行 `arm-hisiv500-linux-gcc` 时，系统会优先使用SDK提供的版本。
- 设置 `CROSS_COMPILE` 环境变量，例如 `CROSS_COMPILE=arm-hisiv500-linux-` 。内核和U-Boot的Makefile会读取这个变量来调用正确的交叉编译器。
- 可能还会设置 `ARCH=arm` 等架构相关变量。

你可以通过以下命令验证工具链是否生效：

bash

AI代码解释 | 复制

```
echo $CROSS_COMPILE
which arm-hisiv500-linux-gcc
arm-hisiv500-linux-gcc -v
```

如果都能正确输出工具链前缀、路径和版本信息，说明环境变量设置成功。

实操心得：我习惯把 `source /path/to/sdk/env_setup.sh` 这一行添加到我的 `~/.bashrc` 文件末尾。这样每次打开新的终端窗口，都会自动设置好编译环境，非常方便。但要注意，如果你同时开发多个不同架构的平台，这种方式可能会造成冲突，此时更推存在需要时手动 `source` 。

3. SDK编译流程深度解析

环境准备好后，就可以开始核心的编译工作了。海思SDK的编译通常不是简单的一个 `make` 命令，而是一个有顺序的构建流程，一般遵循“U-Boot -> Kernel -> Rootfs -> 打包”的顺序。SDK根目录下通常有一个顶层的 `Makefile` 或 `build.sh` 脚本来管理整个流程。

3.1 编译配置与菜单选择

在开始编译前，通常需要进行一些配置。对于Hi3798MV100，关键的配置项包括：

- 芯片型号：** 确认是Hi3798MV100。
- 安全类型：** 区分“高安”与“非高安”。我们的盒子是“非高安”（Non-Secure Boot），这个选项至关重要，它决定了内核和驱动的编译配置，以及最终固件的签名校验方式。选错会导致编译出的系统无法启动。
- 板级配置：** 虽然芯片相同，但不同盒子（如创维E900V10C、华为EC6108V9）的外设（如Wi-Fi/蓝牙芯片、内存型号、闪存类型、网口PHY）可能不同。SDK中可能提供了多个板级配置（ `boardconfig` ）或设备树（ `.dts` ）文件。

进入SDK的 `build` 或 `osdrv` 目录，查找配置脚本。常见的配置命令是 `make menuconfig` （针对内核）或 `./cfg.sh` （SDK顶层配置）。

bash

AI代码解释 | 复制

```
# 假设在osdrv目录下进行整体配置
cd osdrv
make menuconfig
```

或者使用SDK提供的配置脚本：

bash

AI代码解释 | 复制

```
cd ..
./cfg.sh hi3798mv100 nonsecure
# 参数解释：hi3798mv100 是芯片型号，nonsecure 表示非高安
```

配置界面（如果是 `menuconfig` ）会以字符图形菜单形式呈现。你需要导航到相关选项，确保：

- `CPU` 选择为 `Hi3798MV100` 。
- `Security` 或 `Secure Boot` 选项设置为 `Disable` 或 `Non-Secure` 。
- 找到 `Board` 或 `Product` 选择，如果有类似 `E900V10C` 或 `hi3798mv100-demo` 的选项，就选择它。如果没有，可能需要后续手动替换设备树文件。

配置完成后，选择保存并退出。配置信息通常会保存在一个 `.config` 文件中。

3.2 分步编译：U-Boot、内核与根文件系统

配置保存后，就可以开始分步编译了。我推荐先分步编译，验证每一部分是否正常，最后再尝试一键编译。

3.2.1 编译U-Boot U-Boot是系统的引导程序，负责初始化硬件、加载内核和根文件系统。

bashAI代码解释 | 复制

```
# 通常U-Boot源码在 boot 或 u-boot 目录下
cd boot
# 清理旧编译结果（如果是第一次可跳过）
make distclean
# 应用板级配置，例如 hi3798mv100_defconfig
make hi3798mv100_defconfig
# 或者如果之前menuconfig已配置好，可以直接加载
# make ARCH=arm CROSS_COMPILE=arm-hisiv500-linux- hi3798mv100_defconfig

# 开始编译
make ARCH=arm CROSS_COMPILE=arm-hisiv500-linux- -j$(nproc)
```

-j\$(nproc) 表示使用与CPU核心数相同的线程数进行并行编译，可以显著加快速度。

编译成功后，会在当前目录生成 u-boot.bin （原始的U-Boot镜像）和 u-boot-merged.bin （可能合并了某些头部信息的镜像）。海思平台通常需要一个经过特殊工具处理的 u-boot.bin ，我们稍后在打包环节会用到。

3.2.2 编译Linux内核 内核是操作系统的核心。

bashAI代码解释 | 复制

```
cd ../kernel
# 清理
make ARCH=arm distclean
# 应用配置。如果之前运行过顶层menuconfig，这里可能已经有.config文件了。
# 如果没有，可以拷贝SDK提供的默认配置，例如：
cp arch/arm/configs/hi3798mv100_defconfig .config
# 或者再次使用menuconfig进行微调（可选）
make ARCH=arm CROSS_COMPILE=arm-hisiv500-linux- menuconfig
# 编译内核镜像和模块
make ARCH=arm CROSS_COMPILE=arm-hisiv500-linux- uImage -j$(nproc)
make ARCH=arm CROSS_COMPILE=arm-hisiv500-linux- modules -j$(nproc)
```

编译成功后，关键产出在：

- arch/arm/boot/uImage ：可引导的内核镜像。
- 各种内核模块（.ko 文件）散落在各模块目录中。

3.2.3 构建根文件系统 根文件系统包含了系统运行所需的所有库、工具、配置文件和应用程序。SDK可能提供几种方式：

- 使用预制的BusyBox根文件系统：SDK的 osdrv/rootfs 目录下可能有基于BusyBox的构建脚本。
- 使用Buildroot：更现代和灵活的方式，可以定制软件包。
- 使用现成的文件系统：例如直接使用为Hi3798MV100适配的Armbian或Debian根文件系统镜像。

这里以SDK自带的BusyBox 构建为例：

bashAI代码解释 | 复制

```
cd ../osdrv/rootfs
# 可能有一个 build_rootfs.sh 脚本
./build_rootfs.sh hi3798mv100 nonsecure
```

这个脚本会解压BusyBox，编译生成基本的根文件系统目录树，并安装到类似 rootfs_hi3798mv100 的目录中。

3.2.4 编译媒体处理平台（MPP）和其他组件 如果你需要视频编解码、图形显示等多媒体功能，还需要编译MPP组件。

bashAI代码解释 | 复制

```
cd ../mpp
# 通常也有一个顶层的Makefile
make ARCH=arm CROSS_COMPILE=arm-hisiv500-linux- -j$(nproc)
```

MPP的编译可能会依赖内核头文件，确保你已经编译过内核。

3.3 整体编译与固件打包

在分步编译验证无误后，可以使用SDK提供的一键编译脚本进行整体构建。这通常在SDK根目录或 osdrv 目录下。

bash

AI代码解释 | 复制

```
# 回到SDK根目录或osdrv目录
cd ../../
# 执行一键编译脚本，例如：
make all
# 或者
./build.sh all
```

这个脚本会按照正确的顺序调用上述各个部分的编译，并在最后进行打包。打包是生成可烧录固件（通常是 `.img` 文件）的关键步骤，它会将U-Boot、内核、设备树和根文件系统按照芯片要求的格式组合在一起，有时还会加入引导头、签名（高安情况下）等信息。

对于Hi3798MV100非高安版本，最终生成的固件可能是一个名为 `Hi3798MV100-xxx.img` 或 `update.zip` 的文件，位于 `release/` 或 `image/` 目录下。这个文件就可以通过USB烧录工具（如Hitool）或者卡刷的方式刷入机顶盒。

4. 编译过程中的常见问题与解决方案

编译过程很少一帆风顺，尤其是第一次搭建环境。下面是我遇到的一些典型问题及解决方法。

4.1 工具链相关错误

问题1: `arm-hisiv500-linux-gcc: command not found`

- 原因：环境变量未正确设置，或者工具链路径未加入 `PATH`。
- 解决：
 - 确认已执行 `source env_setup.sh`。
 - 检查该脚本内容，看它是否正确地导出了工具链路径。你可以手动添加：

bash

AI代码解释 | 复制

```
export PATH=/path/to/your/sdk/tools/linux/toolchain/arm-hisiv500-linux/bin:$PATH
export CROSS_COMPILE=arm-hisiv500-linux-
export ARCH=arm
```

- 使用 `which arm-hisiv500-linux-gcc` 确认命令是否可找到。

问题2: `/lib/ld-linux.so.2: bad ELF interpreter: No such file or directory`

- 原因：在64位系统上运行32位的工具链，缺少32位运行库。
- 解决：安装我们之前在环境准备环节提到的32位兼容库：`libc6-i386`，`lib32z1`，`lib32stdc++6`。安装后即可解决。

问题3: 工具链版本过旧导致的语法错误

- 现象：编译较新版本的内核或软件时，工具链报错，提示某些C语言特性不支持。
- 解决：海思官方工具链可能比较老。可以尝试：
 - 使用更新的第三方工具链，如Linaro或Arm官方提供的GCC工具链。但需要测试与海思内核、驱动的兼容性。
 - 如果错误不多，可以尝试修改源码，规避工具链不支持的语法。

4.2 内核编译错误

问题4: `error: unknown type name ‘xxx’` 或头文件缺失

- 原因：内核配置不完整，或者依赖的路径不对。
- 解决：
 - 确保执行了 `make menuconfig` 并保存了配置。
 - 在编译内核前，先编译生成必要的头文件：`make ARCH=arm CROSS_COMPILE=arm-hisiv500-linux- prepare`。
 - 检查错误信息中缺失的头文件，看它属于哪个内核模块，在 `menuconfig` 中确保该模块被启用（`=y` 或 `=m`）。

问题5: 驱动编译错误，提示海思特定函数未定义

- 原因：驱动代码依赖海思提供的内部头文件或符号，这些可能不在标准内核源码树中，而是SDK以补丁或独立模块形式提供。
- 解决：
 - 确认你是否完整地应用了SDK提供的内核补丁。有些SDK需要手动打补丁：

bash

AI代码解释 | 复制

```
cd kernel
patch -p1 < ../patches/kernel/*.patch
```

- 检查驱动代码的Makefile，看它是否正确地引用了海思SDK的include路径（如 `-I$(SDK_PATH)/mpp/include`）。

4.3 根文件系统与打包错误

问题6：根文件系统过大，导致打包失败

- 原因：生成的根文件系统体积超过了Flash分区规划的大小。
- 解决：
 - 裁剪根文件系统，移除不必要的调试工具、文档和库。
 - 使用 `mksquashfs` 等工具将根文件系统制作成压缩的squashfs镜像，可以极大减少体积。
 - 修改分区表配置（如果有权限），调整 `rootfs` 分区的大小。这通常需要修改U-Boot或内核中的分区定义。

问题7：打包工具执行失败

- 现象：在最后执行 `mkimage` 或海思专用的 `pack_tool` 时失败。
- 解决：
 - 确认打包工具是否有可执行权限：`chmod +x tools/pack_tool`。
 - 打包工具可能也是32位程序，同样需要安装32位库。
 - 检查打包工具的配置文件（如 `config.ini` 或 `partition.xml`），确认其中指定的U-Boot、内核、根文件系统等输入文件的路径是否正确。

4.4 环境与路径问题

问题8：编译中途失败，重新开始编译时出现奇怪错误

- 原因：中间文件（`.o`，`.d`）状态不一致。
- 解决：彻底清理后再重新编译。

bashAI代码解释 | 复制

```
# 在内核或U-Boot目录下
make distclean # 或 make mrproper（内核清理更彻底）
# 然后重新执行配置和编译步骤。
```

对于SDK整体，查看顶层Makefile是否有 `clean` 或 `distclean` 目标。

问题9：Python脚本语法错误（IndentationError, print语法错误）

- 原因：SDK中的老旧Python脚本与当前系统的Python 3不兼容。
- 解决：
 - 尝试使用Python 2运行：`python2 script.py`。
 - 修改脚本首行，将 `#!/usr/bin/python` 改为 `#!/usr/bin/python2`。
 - 对于简单的语法不兼容（如print语句），可以手动修改脚本，将 `print “xxx”` 改为 `print(“xxx”)`。但需谨慎，可能引入其他问题。

为了便于查阅，我将上述常见问题及解决方法汇总成下表：

问题类别	典型错误信息	可能原因	解决方案
工具链	<code>command not found</code>	PATH环境变量未设置	<code>source env_setup.sh</code> 或手动导出PATH
工具链	<code>bad ELF interpreter</code>	缺少32位运行库	安装 <code>libc6-i386</code> 等32位兼容库
内核	<code>unknown type name</code>	头文件缺失/配置不全	执行 <code>make prepare</code> ；检查menuconfig配置
内核/驱动	海思API未定义	内核补丁未打或路径错误	检查并应用SDK内核补丁；核对Makefile包含路径
根文件系统	打包失败：空间不足	rootfs体积超出分区限制	裁剪文件系统；制作squashfs镜像；调整分区表
打包工具	打包工具执行失败	权限不足或缺少库	<code>chmod +x</code> ；安装32位库；检查配置文件路径
环境	二次编译失败	中间文件状态不一致	执行 <code>make distclean</code> 后重试
脚本	<code>SyntaxError</code> (Python)	Python 2/3语法不兼容	使用 <code>python2</code> 运行；或修改脚本适配Python 3

5. 编译后的验证与后续步骤

成功编译出 `*.img` 文件只是第一步。如何验证它的有效性呢？

- 文件检查：使用 `file` 命令检查生成的镜像文件类型。使用 `ls -lh` 查看文件大小是否合理（通常从几十MB到几百MB）。
- 模拟测试（可选）：使用QEMU等模拟器运行编译出的内核和根文件系统，进行基础功能测试。但对于海思这种包含大量专用外设的芯片，模拟器通常只能测试到内核启动阶段。
- 实机烧录：这是最直接的验证方式。将生成的 `*.img` 文件通过Hitool工具（海思专用烧录工具）的USB烧写功能，或者制作成SD卡启动盘（如果盒子支持），烧录到机顶盒中。观察串口调试输出（需要连接TTL转USB线到盒子的UART引脚），这是诊断启动问题的“眼睛”。正常的启动日志会依次显示U-

Boot、内核加载、文件系统挂载、系统初始化的信息。

如果系统成功启动，并出现了登录提示符（如 `root@Hi3798MV100:~#` ），那么恭喜你，SDK编译环境搭建和基础固件构建就成功了！接下来，你可以基于此进行更多探索：

- **定制根文件系统**：用Buildroot或Yocto构建一个包含更多软件（如Python、Node.js、Docker等）的根文件系统，打造一个功能更强的迷你服务器。
- **驱动开发与调试**：如果你有外设需要驱动，或者发现某些硬件（如Wi-Fi、蓝牙）工作不正常，就需要深入研究内核驱动，进行修改和调试。
- **应用开发**：利用海思的MPP库，开发视频播放、图形界面（如Qt）等应用程序。
- **系统优化**：调整内核参数、裁剪不需要的模块、优化启动速度等。

整个Hi3798MV100的SDK编译过程，就像是在组装一台精密的仪器。环境准备是备齐零件和工具，配置是阅读图纸，分步编译是组装各个子系统，打包是总装，而问题排查则是解决组装过程中遇到的卡顿和异响。这个过程需要耐心和细心，尤其是面对那些晦涩的错误日志时。但每一次成功的编译和启动，都是对系统底层理解的一次深化。我自己的经验是，一定要善用串口调试输出，它是最忠实的问题报告者；另外，做好每一步的笔记，记录下成功的配置和命令，这会在你下次搭建环境或者帮助他人时节省大量时间。

标签 [#Hi3798MV100](#) [#SDK编译](#) [#交叉编译](#)