# Tina Linux Buildroot
# 使用指南

**版本号: 1.5**

**发布日期: 2025.8.18**

# 版本历史

| 版本号 | 日期 | 制/修订人 | 内容描述 |
| --- | --- | --- | --- |
| 1.0 | 2024.2.21 | AWA1586 | 初始版本 |
| 1.1 | 2025.1.3 | AWA2253 | 增加不同 ubuntu 版本下的 Buildroot 构建章节 |
| 1.2 | 2025.4.16 | AWA0916 | 支持 MR153；修改部分描述错误 |
| 1.3 | 2025.5.6 | AWA0916 | 支持 AIOT Linux A733、T736 |
| 1.4 | 2025.7.11 | AWA1586 | 支持 Tina Linux T153、T536 |
| 1.5 | 2025.8.18 | AWA0916 | 统一格式、专业术语 |

# 目　录

# 1 前言

## 1.1 文档简介

本文档介绍 Tina 系统 (Linux SDK) 下的 Buildroot 文件系统的目录结构、编译和使用，主要目的用于指导用户如何定制和使用文件系统和使用基于 Buildroot 构建自己的应用程序。

## 1.2 目标读者

基于 Tina 系统的 Buildroot 文件系统及应用开发者。

## 1.3 适用范围

全志 T113、T527、A40i、MR153、A733、T736、T153、T536 等芯片。

# 2 Buildroot 介绍

Buildroot，是嵌入式开发领域内一个成套的嵌入式开发环境。它可以用来定制自己的交叉编译器，制作自己的根文件系统。

Buildroot 集成在 Tina 系统里面，在 Tina 平台中，主要使用 Buildroot 定制文件系统。因此，可以在 Tina 里面配置并且编译。

# 3 buildroot 目录结构

## 3.1 buildroot

Tina SDK 下的 buildroot 文件夹主要包含（201902/202205）两个版本的 Buildroot 项目源码、Buildroot 全志自研软件包编译配置 (config) 和 AW Buildroot 方案全志自研软件包 (package)。

**布的 SDK 一般只包含其中一个版本。**

```
# ${SDK_PATH}/buildroot
├── buildroot-201902 # 201902版本
├── buildroot-202205 # 202205版本
   ├── arch       # 存放CPU架构相关的配置脚本，如arm/mips/x86
   ├── board      # 存放了一些默认开发板的配置补丁之类的
   ├── boot       # 引导系统
   ├── build.sh
   ├── CHANGES
   ├── Config.in
   ├── Config.in.legacy
   ├── configs    # 放置开发板的一些配置参数,如sun55iw3p1*_*_defconfig
   ├── COPYING
   ├── DEVELOPERS
   ├── dl         # 存放下载的源码包及应用软件的压缩包
—docs # 存放相关的参考文档
   ├── fs         # 放各种文件系统的源代码
   ├── linux      # 存放着Linux kernel的自动构建脚本
   ├── Makefile
   ├── Makefile.legacy
   ├── package    # 下面放着应用软件的配置文件，包含Config.in和xxx.mk及patch
   ├── README
   ├── scripts    # 存放编译脚本
   ├── support
   ├── system     # 根目录主要骨架和启动初始化配置,然后再安装toolchain的动态库和勾选的package的可执行文件
   │   └── skeleton # 提供rootfs模板目录结构，包含一些最基本的目录及文件。
   ├── toolchain  # 存放编译工具链
   └── utils
├── config                # Buildroot全志自研软件包编译配置
│   └── buildroot
└── package               # Buildroot方案全志自研软件包
   ├── auto               # Buildroot方案模块测试demo
   ├── qt                 # qt-5.12
   ....
```

## 3.2 output

注：buildroot 根目录下的 output 目录，在 Tina 编译系统里面，转移到了 out/{IC}/{board}/buildroot/buildroot 路径下。

```
├──── output   # 输出文件夹，存放解压后的代码和目标文件
│ ├──── build  # 存放解压后的各种软件包和编译完成后的现场
│ ├──── host   # 存放着制作好的编译工具链，如gcc、arm-linux-gcc等工具
│ ├──── images # 存放着编译好的uboot.bin, zImage, rootfs等镜像文件，可烧写到板子里, 让linux系统跑起来。
│ ├──── per-package # 开启BR2_PER_PACKAGE_DIRECTORIES（并行编译）才有的目录，存放每个软件包的编译依赖以及
      编译产物。
│ ├──── staging # 软链接到host/< tuple >/sysroot/ 就是上面说到的文件系统需要的so库、*.h等目录,方便查看
│ ├──── target # 用来制作rootfs文件系统，里面放着Linux系统基本的目录结构，以及编译好的应用库和bin可执行文件(
      Buildroot根据用户配置把.ko .so .bin文件安装到对应的目录下去，根据用户的配置安装指定位置)
```

# 3.3 platform

由于 Tina 平台不单单支持 Buildroot 文件系统，还有 openwrt 等，因此一些共性全志自研软件包
此目录下。下图仅列出部分应用列表。

```
# ${SDK_PATH}/platform
platform
├──── allwinner
│   ├──── display
│   │   ├──── libgpu            # GPU的mali.so库
│   │   └──── libuapi
│   ├──── power
│   │   └──── healthd
│   ├──── system
│   │   ├──── ota-burnboot      # OTA升级相关代码
│   │   └──── rpbuf
│   ├──── usb
│   │   ├──── adbd              # adbd相关代码
│   │   ├──── mtp
│   │   └──── setusbconfig
│   ├──── utils
│   │   ├──── boot-play
│   │   ├──── cpu_monitor       # cpu_monitor相关代码
│   │   ├──── libsocket_db
│   │   ├──── mtop              # mtop相关代码
│   │   └──── uevent-monitor
│   └──── wireless
│       ├──── btmanager         # bt相关代码
│       ├──── common
│       ├──── firmware          # wifi/bt的固件包
│       └──── wifimanager       # wifi相关代码
└──── Makefile -> build/Makefile
```

# 4 构建 Buildroot

## 4.1 配置 repo 环境

说明

**如果没有 repo/git 环境，可参考如下方式准备。**

### 4.1.1 安装必要的软件包

在配置 repo 之前，请确保系统已安装以下必要软件包：

```
sudo apt-get update //更新包索引
sudo apt-get install -y git-core curl //安装Git和Curl工具，这是repo所依赖的基础工具
```

### 4.1.2 下载并配置 repo 工具

#### .1 下载 repo

在用户的 bin 目录中下载 repo 脚本。创建 bin 目录（如果不存在），然后下载 repo 脚本：

```
mkdir -p ~/bin
curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
```

#### 4.1.2.2 设置 repo 脚本的可执行权限

下载完成后，需要为 repo 脚本设置可执行权限：

```
chmod a+x ~/bin/repo //设置repo脚本为可执行文件
```

#### 4.1.2.3 将 repo 工具添加到 PATH 环境变量

为了在终端中能够直接使用 repo 命令，需要将 repo 工具的路径添加到 PATH 环境变量中：

```
export PATH=~/bin:$PATH
```

为了使这个修改在每次登录时自动生效，可以将这条命令添加到 ~/.bashrc或 ~/.profile文件中：

```
echo 'export PATH=~/bin:$PATH' >> ~/.bashrc //将bin文件路径添加命令写入bashrc //
source ~/.bashrc //重新加载.bashrc文件，使修改立即生效
```

#### 4.1.2.4 更换镜像源

由于谷歌服务器位于国外，每次运行时 repo 会检查更新导致下载较慢，甚至直接卡住不动然后失败。国内用户可以配置镜像源，在 repo 的运行过程中会尝试访问官方的 git 源更新自己，提高下载速度。

执行以下命令更换镜像源：

```
#ubuntu20.04/16.04执行下面命令
echo export REPO_URL='https://mirrors.bfsu.edu.cn/git/git-repo' >> ~/.bashrc
# ubuntu14.04执行下面命令
echo export REPO_URL='http://mirrors.bfsu.edu.cn/git/git-repo' >> ~/.bashrc
# 然后source使配置立即生效
source ~/.bashrc
```

#### 4.1.2.5 配置 git 用户与邮箱

git 上传代码需要配置用户名以及邮箱地址，可以通过以下命令进行配置：

```
 git config --global user.email "you@example.com"
nfig --global user.name "Your Name"
```

## 4.2 配置编译环境

**ubuntu 20.04/22.04 版本，执行下面命令安装软件包：**

```
sudo apt-get install build-essential subversion git-core libncurses5-dev zlib1g-dev gawk flex quilt libssl-dev xsltproc
    libxml-parser-perl mercurial bzr ecj cvs unzip lib32z1 lib32z1-dev lib32stdc++6 libstdc++6 libc6:i386 libstdc++6:i386
    lib32ncurses-dev lib32z1 ncurses-term bison libexpat1-dev python-is-python3 -y
```

**ubuntu 16.04/18.04 版本，执行下面命令安装软件包：**

```
sudo apt-get install build-essential subversion git-core libncurses5-dev zlib1g-dev gawk flex quilt libssl-dev xsltproc
    libxml-parser-perl mercurial bzr ecj cvs unzip lib32z1 lib32z1-dev lib32stdc++6 libstdc++6 libc6:i386 libstdc++6:i386
    lib32ncurses5 lib32z1 ncurses-term bison libexpat1-dev -y
```

**ubuntu 14.04 版本可直接执行以下命令安装：**

```
sudo apt-get install build-essential subversion git-core libncurses5-dev zlib1g-dev gawk flex quilt libssl-dev xsltproc
    libxml-parser-perl mercurial bzr ecj cvs unzip lib32z1 lib32ncurses5 lib32z1-dev lib32stdc++6 libstdc++6 ncurses-
    term bison libexpat1-dev -y
```

# 4.2.1 各个软件包的说明：

build-essential：
包含编译软件所需的基本工具和库，包括 gcc、g++、make 等。
subversion：
版本控制系统，用于管理源代码的版本，适合团队协作。
git-core（通常只需使用 git）：
分布式版本控制系统，用于源代码管理，支持分支和合并等功能。
libncurses5-dev：
提供终端用户界面（TUI）开发所需的库，支持文本界面的应用程序。
zlib1g-dev：
提供 Zlib 压缩库的开发文件，常用于数据压缩和解压缩。
gawk：
GNU 版本的 AWK，文本处理工具，用于模式扫描和处理。
flex：
分析器生成器，通常与 bison 一起使用。
quilt：
用于管理补丁的工具，方便在源代码上应用和管理多个补丁。
libssl-dev：
OpenSSL 的开发库，提供 SSL/TLS 加密功能，常用于网络安全。
xsltproc：
XSLT 处理器，用于将 XML 文档转换为其他格式（如 HTML）。
libxml-parser-perl：
Perl 的 XML 解析库，提供 XML 文档的解析功能。
mercurial：
分布式版本控制系统，类似于 Git，用于源代码管理。
bzr：
Bazaar 版本控制系统，另一种源代码管理工具。
ecj：
Eclipse Java Compiler，用于编译 Java 代码。
cvs：
控制系统，较旧的源代码管理工具。现已被
unzip：
用于解压缩 ZIP 文件的工具。
lib32z1：
32 位 Zlib 压缩库，提供 32 位应用程序的压缩支持。
lib32z1-dev：
32 位 Zlib 压缩库的开发文件，供 32 位应用程序开发使用。
lib32stdc++6：
32 位 GNU 标准 C++ 库，供 32 位应用程序使用。
libstdc++6：
GNU 标准 C++ 库，供 64 位应用程序使用。
libc6:i386
提供基本的系统调用接口和 C 标准库函数，如字符串处理、内存管理、输入输出等。
libstdc++6:i386
提供 C++ 标准库的功能，包括 STL（标准模板库）、输入输出流、字符串处理等。
lib32ncurses-dev：
32 位 ncurses 开发库，提供终端用户界面开发所需的功能。
ncurses-term：
提供终端类型的描述文件，支持终端用户界面的应用程序。
bison：
语法分析器生成器，用于生成解析器（parser），通常与 flex 一起使用。
libexpat1-dev：
Expat XML 解析库的开发文件，提供 XML 文档的解析功能。
python-is-python3:
创建一个符号链接，使得 python 命令指向 python3。

可通过执行dpkg -l显示所有已安装的软件包及其版本信息。也可以执行apt list --installed命令以更易读的

格式列出所有已安装的软件包。

# 4.3 配置 Buildroot

## 4.3.1 配置 BoardConfig.mk

为了方便文件系统定制，Tina SDK 支持 Buildroot 构建文件系统，为了支持 Buildroot 方案，SDK 需要进行如下的适配。

新增 ${SDK_PATH}/device/config/chips/{ic}/configs/{board}/buildroot 文件夹，新增如下文件，

```
${SDK_PATH}/device/config/chips/{ic}/configs/{board}/buildroot$ tree -L 1
├── BoardConfig.mk    # 配置文件，必需文件，下文说明
├── env_ab.cfg        # ab系统需要
├── env.cfg           # 环境变量配置文件，必需文件
├── env-recovery.cfg  # recovery环境变量配置文件
├── swupdate          # OTA升级相关
├── sys_partition_ab.fex
├── sys_partition.fex # 系统分区表
└── sys_partition-recovery.fex
```

Buildroot 的配置在 BoardConfig*.mk 如下所示：

```
LICHEE_BUILDING_SYSTEM
LICHEE_BR_VER

LICHEE_BR_RAMFS_CONF
```

当需要使用 Buildroot 来构建文件系统时，需要在 BoardConfig*.mk 文件增加这些环境变量的配置，介绍如下：

```
# ${SDK_PATH}/device/config/chips/{ic}/configs/{board}/buildroot/BoardConfig.mk
LICHEE_BUILDING_SYSTEM:=buildroot          # 构建系统Buildroot
LICHEE_BR_VER:=202205                      # Buildroot版本，目前支持201902和202205
LICHEE_BR_DEFCONF:=sun55iw3p1_defconfig    # Buildroot构建根文件系统时使用的配置文件
LICHEE_BR_RAMFS_CONF:=sun55iw3p1_ramfs_defconfig # 编译ramfs时使用的配置文件，没有编译ramfs，可以不配置
```

## 2 配置 defconfig

在编译对应版本的 Buildroot 前，需要提供构建文件系统的 Buildroot 的配置文件，如 sun55iw3p1_defconfig。

```
xxx:${SDK_PATH} ls  buildroot/buildroot-202205/configs/sun55iw3p1_defconfig
buildroot/buildroot-202205/configs/sun55iw3p1_defconfig
```

# 编译 Buildroot

满足这些配置后，按照 Tina 的编译过程，就可以对 Buildroot 进行编译了。

## 4.4.1 Buildroot 完整编译

1. 方案选择：

```
./build.sh config
# 根据需要选择配置，例如依次选择linux,buildroot,t527,demo_linux_car,default,linux-5.15
```

成功后得到如下log信息

```
make: Entering directory `/home/xxx/workspace/t527_linux_aiot/buildroot/buildroot-202205'
  GEN    /home/xxx/workspace/t527_linux_aiot/out/t527/demo_linux_car/buildroot/buildroot/Makefile
#
# configuration written to /home/xxx/workspace/t527_linux_aiot/out/t527/demo_linux_car/buildroot/buildroot/.config
#
make: Leaving directory `/home/xxx/workspace/t527_linux_aiot/buildroot/buildroot-202205'
INFO: buildroot defconfig is sun55iw3p1_defconfig

# Tina 系统已经选择了对应的Buildroot配置
```

2. 编译

Tina 编译过程，会根据选择的 Buildroot 配置进行 Buildroot 文件系统的全部编译。

```
./build.sh  # 该命令会在buildroot目录执行 make all,编译Buildroot所有选中的程序
```

3. 打包

```
./build.sh pack
```

三个步骤即可生成具有 Buildroot 文件系统的 Linux 固件镜像。

## 4.4.2 单独编译软件包

有时候，在开发调试的过程中，需要单独编译一个软件包。以
准编译方式进行模块编译。

```
# (1) 进入buildroot目录,如下
$ cd buildroot/buildroot-202205
# (2) 删除上次mtop编译生成的中间文件 （每次编译，最稳妥就是先clean）
$ make mtop-dirclean
# (3) 强制重新编译mtop
$ make mtop-rebuild
# 编译完成后会自动把生成的bin文件拷贝到out目录下的rootfs中
# 此时可以手动push进小机端进行测试，或者重新执行/build.sh即可更新rootfs包
```

说明

**top 的源码位置：platform/allwinner/utils/mtop/***

**mtop 的 Buildroot 编译配置位置：buildroot/config/buildroot/allwinner/utils/mtop/***

**mtop 编译生成的中间文件位置：out/{ic}/{board}/buildroot/buildroot/build/mtop/***

**mtop 编译最终生成的 rootfs 下的 bin 位置：out/{ic}/{board}/buildroot/buildroot/target/bin/mtop**

说明

**Buildroot 的全志自研软件包每次修改源码后不会自动编译，所以最好手动 dirclean/rebuild 模块一次。**

# 5 Buildroot 常用命令

## 5.1 Tina SDK 支持命令

说明

**以下的命令执行路径为 SDK 的根目录。**

| 命令 | 用法说明 |
| --- | --- |
| ./build.sh config | 选择 Buildroot 文件系统配置 |
| ./build.sh buildroot_rootfs | 单独编译 Buildroot rootfs |
| ./build.sh buildroot_menuconfig | 打开 Buildroot 配置文件 |
| ./build.sh buildroot_saveconfig | 保存 Buildroot 配置文件 |

## 5.2 Buildroot 常用命令

说明

**以下的命令执行路径为 buildroot/buildroot-20xxxx。**

| | 命令 | 用法说明 |
| --- | --- | --- |
| 整体编译 | make xxx_defconfig | 基于 Configs/xxx_defconfig 生成.config |
| 整体编译 | make menuconfig | 界面配置 |
| 整体编译 | 保存到 Configs/xxx_defconfig | make savedefconfig |
| 整体编译 | make | 整体编译 (由于集成了 aw 私有包，不建议使用) |
| 整体编译 | make clean | 清除过程文件和目标文件 |
| 整体编译 | make distclean | 清除所有的生成文件 |
| 软件包 | make xxx-build | 编译名为 xxx 的软件包 |
| 软件包 | make xxx-dirclean | 清除名为 xxx 的软件包 |

| | 命令用法说明 | |
|---|---|---|
| 软件包 | make xxx-rebuild | 重新编译名为 xxx 的软件包 |
| 软件包 | make xxx-reconfigure | 重新配置 xxx 软件包 |

# 6 与标准 Buildroot 的不同

由于 Tina 需要根据不同板型来配置并编译 Buildroot，这就决定了 Tina 的 Buildroot 和标准 Buildroot 编译有些不同。

Tina 系统中的 Buildroot 和标准 Buildroot 的不同主要在.config 和编译目标输出。位置和说明情况如下所示：

- buildroot/buildroot-xxx/.config：Tina 系统编译时不会生成或者使用到此文件，属于无效文件
- out/{IC}/{board}/buildroot/buildroot/.config：Buildroot 编译时使用到的.config
- out/{IC}/{board}/buildroot/ramfs/.config：ramfs 编译时使用到的.config
- buildroot/buildroot-xxx/output：Tina 系统编译时不会生成或者使用到此目录，属于无效目录
- out/{IC}/{board}/buildroot/buildroot/：同一个芯片的不同板型有不同的 Buildroot 输出目录
- out/{IC}/{board}/buildroot/ramfs/：ramfs 输出目录

同时，为了支持 AW 自研的软件包，新增了如下的目录。

| 目录 | 说明 |
| --- | --- |
| buildroot/config | Buildroot 全志自研软件包编译配置 |
| buildroot/package | Buildroot 方案全志自研软件包 |
| platform | 共性全志自研软件包源码 |

# 7 Buildroot 软件包修改

## 7.1 添加 Buildroot 全志自研软件包

以 t527 芯片平台新增 helloworld 为例进行描述。

写 app 源码和 Makefile

```
xxx@GZExdroid04:~/workspace/t527_linux_aiot/platform/allwinner/helloworld$ tree
├── helloworld.c
└── Makefile
```

```
/* helloworld.c */
#include <stdio.h>
void main(void) { printf("Hello world.\n");}
```

```
TARGET = helloworld
SRCS = $(wildcard *.c)

all: $(TARGET)

$(TARGET): $(SRCS)
	$(CC) $^ $(CFLAGS) $(INCLUDES) $(LDFLAGS) -o $@

clean:
	rm -rf $(TARGET)

.PHONY: all clean install
```

2. helloworld 增加编译规则

需要新建buildroot/config/buildroot/helloworld目录，并创建 helloworld.mk 与 Config.in 文件。

通过 make menuconfig 进行选择。

```
config BR2_PACKAGE_HELLOWORLD
	bool "helloworld"
	help
	  This is a demo to add local app.
```

Buildroot 编译 helloworld 所需要的设置 helloworld.mk，包括源码位置、编译规格、安装目录等。

```
HELLOWORLD_VERSION:= 1.0.0
HELLOWORLD_SITE = $(PLATFORM_PATH)/../../../platform/allwinner/helloworld
```

```
HELLOWORLD_SITE_METHOD:=local
HELLOWORLD_INSTALL_TARGET:=YES

define HELLOWORLD_BUILD_CMDS
   $(MAKE) CC="$(TARGET_CC)" LD="$(TARGET_LD)" -C $(@D) all
endef

define HELLOWORLD_INSTALL_TARGET_CMDS
   $(INSTALL) -D -m 0755 $(@D)/helloworld $(TARGET_DIR)/bin
endef

$(eval $(generic-package))
```

3. 把软件包添加到 Buildroot 编译脚本中。

uildroot/config/增加如下信息Config.in

```
xxx@GZExdroid04:~/workspace/t527_linux_aiot/buildroot/config$ git diff buildroot/Config.in
diff --git a/buildroot/Config.in b/buildroot/Config.in
index f5bf03a..816b819 100755
--- a/buildroot/Config.in
+++ b/buildroot/Config.in
@@ -10,6 +10,7 @@ endmenu

 menu "system"
 source "../config/buildroot/allwinner/system/ota/ota-burnboot/Config.in"
+source "../config/buildroot/helloworld/Config.in"
 endmenu

 menu "usb"
```

uildroot/config/build增加如下信息m.mk

```
xxx@GZExdroid04:~/workspace/t527_linux_aiot/buildroot/config$ git diff buildroot/platform.mk
diff --git a/buildroot/platform.mk b/buildroot/platform.mk
index 488090d..0e0e24a 100755
--- a/buildroot/platform.mk
+++ b/buildroot/platform.mk
@@ -34,3 +34,4 @@ include ${PLATFORM_PATH}/rpbuf/rpbuf.mk
 include ${PLATFORM_PATH}/allwinner/wireless/wireless_common/wireless_common.mk
 include ${PLATFORM_PATH}/allwinner/wireless/btmanager/btmanager.mk
 include ${PLATFORM_PATH}/ai-sdk/ai-sdk.mk
+include ${PLATFORM_PATH}/helloworld/helloworld.mk
```

保存onfig 选中

通过 menuconfig 选中 APP，然后 make savedefconfig，helloworld 的配置（BR2_PACKAGE_HELLOWORLD）就会保存到 sun55iw3p1_defconfig 中。

修改 Buildroot 的 defconfig 后，开始编译即可。

5. 编译 app

可以和整个平台一起编译 APP；或者 make helloworld 单独编译。

文件都会拷贝到 output/build/helloworld-1.0.0

然后生成的 bin 文件拷贝到 output/target/bin/helloworld，这个文件会打包到文件系统中。

说明

**如遇到较为复杂的程序，可以参考 AW 其他自研的类似 app，适配操作都是类似的。**

# 7.2 配置 Buildroot 开机自启动

模块位置：buildroot/allwinner/system/busybox-init-base-files。

d 开机自启动为例：

1. 新建 S50adb_start 文件，并实现 start 与 stop 方法，分别对应开机自启动运行，以及关机退出进程。

```
xxx:~/workspace/t527_linux_aiot/buildroot/config/buildroot/allwinner/system/busybox-init-base-files/etc/init.d$ tree
.
├── S20mdev
├── S50adb_start # 新增的开机启动脚本
...
```

.d/文件夹下。

```
26
27 #config boot scripts for audio
28 ifeq ($(LICHEE_IC), $(filter $(LICHEE_IC), t113_s3p t113_s4 t113_s4p))
29         ASOUND_BOOT_SCRIPTS=$(@D)/audio-config/S80bootplay-$(LICHEE_IC)
30 else
31         ASOUND_BOOT_SCRIPTS=$(@D)/audio-config/S80bootplay
32 endif
33
34 define BUSYBOX_INIT_BASE_FILES_BUILD_CMDS
35         cp -rf $(PLATFORM_PATH)/allwinner/system/busybox-init-base-files/etc/* $(@D)/etc/
36         cp -rf $(PLATFORM_PATH)/allwinner/system/busybox-init-base-files/ota-config $(@D)/
37         cp -rf $(PLATFORM_PATH)/allwinner/system/busybox-init-base-files/audio-config $(@D)/
38 endef
39
40 define BUSYBOX_INIT_BASE_FILES_INSTALL_TARGET_CMDS
41         cp -rf $(@D)/etc/* $(TARGET_DIR)/etc/
42         cp -rf $(OTA_FILES_PATH)/* $(TARGET_DIR)/etc/
43         cp -rf $(ASOUND_CONF_FILES) $(TARGET_DIR)/etc/asound.conf
44         cp -rf $(ASOUND_BOOT_SCRIPTS) $(TARGET_DIR)/etc/init.d/S80bootplay
45 endef
```

图 7-1: etc_install

说明

**原理介绍：**
**Buildroot 原生开机运行的模块为 package/initscripts，加载 rootfs 后首先会运行/etc/init.d/rcS 脚本，这个脚本会遍历/etc/init.d/文件夹下的 S 开头的文件，并依次执行他们。**
**所以如果期望开机自启动其他进程，可新建 Sxx 的文件，拷贝至/etc/init.d/即可实现。**

# 8 Q&A

## 8.1 ubuntu20.04 显示无法识别 python

**问题原因：** Ubuntu 从 20.04 开始不再将 python 加入 PATH 环境变量，在编译安装一些软件会提示无法运行并提示找不到 python，然而 python3 已安装，需要额外重定向。

**解决方法：**[illegible]较繁琐。可以通过安装
软件包 python-is-python3 创建软连接，将 python 指向 python3，即可解决问题。

```
# 安装python-is-python3软件包，创建软连接
sudo apt-get install python-is-python3
```

## 8.2 ubuntu20.04 显示编译错误

如下图所示，编译时显示预处理指令存在语法解释错误：

```
12-30 19:09:19.190   53577 D mkcommon  : CAT     u-boot-dtb.bin
12-30 19:09:19.217   53577 D mkcommon  : COPY    u-boot.bin
12-30 19:09:19.251   53577 D mkcommon  : 'u-boot.bin' -> 'u-boot-sun8iw20p1.bin'
12-30 19:09:19.254   53577 D mkcommon  : 'u-boot-sun8iw20p1.bin' -> '/home/                    /device/config/chips/t113//bin/u-boot-sun
8iw20p1.bin'
12-30 19:09:19.256   53577 D mkcommon  : 'u-boot-sun8iw20p1.bin' -> '/home/                    :/out/t113/evb1_auto/buildroot/u-boot-sun8
iw20p1.bin'
12-30 19:09:19.283   53577 D mkcommon  : CFGCHK  u-boot.cfg
12-30 19:09:19.357   53577 D mkcommon  : --------build for mode:all board:t113------------------
12-30 19:09:19.421   53577 D mkcommon  : platform set to sun8iw20p1
12-30 19:09:19.453   53577 D mkcommon  : CHK                                  /brandy/brandy-2.0/spl-pub/include/config.h
12-30 19:09:19.454   53577 D mkcommon  : UPD                                  /brandy/brandy-2.0/spl-pub/include/config.h
12-30 19:09:19.458   53577 D mkcommon  : CHK                                  /brandy/brandy-2.0/spl-pub/autoconf.mk
12-30 19:09:19.459   53577 D mkcommon  : UPD                                  /brandy/brandy-2.0/spl-pub/autoconf.mk
12-30 19:09:19.460   53577 D mkcommon  :                                     /brandy/brandy-2.0/tools/make_dir/make4.1/bin/make -C /home/
                                       /brandy/brandy-2.0/spl-pub/arch/arm/cpu/armv7/
12-30 19:09:19.483   53577 D mkcommon  : In file included from sboot_entry.S:3:0:
12-30 19:09:19.486   53577 D mkcommon  :                                     :/brandy/brandy-2.0/spl-pub/include/config.h:1:23: error: extra tokens
 at end of #ifndef directive [-Werror]
12-30 19:09:19.487   53577 D mkcommon  : #ifndef __home_                      _brandy_brandy_2_0_spl_pub_include_config_h_
12-30 19:09:19.488   53577 D mkcommon  : ^
12-30 19:09:19.489   53577 D mkcommon  : /home/                              :/brandy/brandy-2.0/spl-pub/include/config.h:2:23: error: missing whit
espace after the macro name [-Werror]
12-30 19:09:19.490   53577 D mkcommon  : #define __                         _brandy_brandy_2_0_spl_pub_include_config_h_
12-30 19:09:19.491   53577 D mkcommon  : ^
12-30 19:09:19.491   53577 D mkcommon  : cc1: all warnings being treated as errors
12-30 19:09:19.492   53577 D mkcommon  : /home/                               /brandy/brandy-2.0/spl-pub/mk/config.mk:148: recipe for target 'sboot
_entry.o' failed
12-30 19:09:19.493   53577 D mkcommon  : make[2]: *** [sboot_entry.o] Error 1
12-30 19:09:19.494   53577 D mkcommon  : Makefile:134: recipe for target '/home/                    :/brandy/brandy-2.0/spl-pub/arch/arm/
cpu/armv7/libarch.o' failed
12-30 19:09:19.495   53577 D mkcommon  : make[1]: *** [/home/                                      /brandy/brandy-2.0/spl-pub/arch/arm/cpu/armv7/libarch.o
] Error 2
12-30 19:09:19.495   53577 D mkcommon  : Makefile:148: recipe for target 'all' failed
12-30 19:09:19.496   53577 D mkcommon  : make: *** [all] Error 2
12-30 19:09:19.497   53577 E mkcommon  : build brandy Failed
12-30 19:09:19.498   53577 F mkcommon  : build_bootloader failed
```

图 8-1: 20_04 版本编译报错

**问题原因：**编译路径中存在非 ASCII 字符，比如中文字符，会导致编译器出现问题。

**解决方法：**移动 SDK 包到纯英文路径下，确保路径中不存在非 ASCLL 字符。

# repo 初始化仓库报错

```
     :~/workspace/     $ repo init -u http://gerrit.allwinnertech.com:8
081/product/tina/manifest.git -b tina-dev -m t113/product-da-t113-tina5.0-
stable.xml
  File "/home/     i/bin/repo", line 88
    reexec(f"python{min_major}.{min_minor + inc}")
                                                ^
SyntaxError: invalid syntax
```

图 8-2: repo 初始化仓库报错

**问题原因**：Python 版本问题。f-string（格式化字符串字面量）是从 Python 3.6 开始引入的。如果使用的 Python 版本低于 3.6，就会出现这个语法错误。可以通过在终端中运行 python --version 或 python3 --version 来检查当前使用的 Python 版本。

当前 python3 版本为 3.5.22，因此不能支持以上的语法，需要安装 python3.6 版本。

**解决方法：**手动安装 python3.6 以上的版本。

1. 下载 python3.6

```
网址：Python Source Releases | Python.org
选择Gzipped source tarball格式文件下载
```

2. 解压压缩包

```
tar xfz Python-3.6.5.tgz
# 这里使用xfz命令，而不建议使用-xvzf命令，因为其释放的文件夹需要root权限才可以更改或者删除。
```

3. 进入到解压好的文件夹内配置

```
cd Python-3.6.5/
# 指定python3.6的安装路径为/usr/bin/python3.6
./configure "--prefix=/usr/bin/python3.6"
```

4. 编译并安装

```
sudo make
sudo make install
# 进入python3.6文件夹内，有bin include lib share 文件夹
```

5. 建立指向 Python3.6 的链接，权限不够的话 sudo

```
sudo rm -rf /usr/bin/python3
sudo ln -s /usr/bin/python3.6/bin/python3.6 /usr/bin/python3
python3# 如需要修改默认指向
# 先删除默认的Python软链接：
sudo rm -rf /usr/bin/python
# 然后创建一个新的软链接指向需要的Python版本：
sudo ln -s /usr/bin/python3 /usr/bin/python
# 如果想还原回原python2.7，只需
sudo rm -rf /usr/bin/python
sudo ln -s /usr/bin/ptyhon2.7 /usr/bin/python
```

6. 最后使用 python 查看版本

```
python --version
Python 3.6.5
```

# 8.4 ubuntu16.04 编译出现段错误（Segmentation Fault)

**问题现象：**

如图所示：

```
[ 68%] Building CXX object Source/CMakeFiles/CMakeLib.dir/cmMSVC60LinkLineComputer.cxx.o
[ 68%] Building CXX object Source/CMakeFiles/CMakeLib.dir/cmOSXBundleGenerator.cxx.o
[ 68%] Building CXX object Source/CMakeFiles/CMakeLib.dir/cmOutputConverter.cxx.o
[ 68%] Building CXX object Source/CMakeFiles/CMakeLib.dir/cmNewLineStyle.cxx.o
[ 68%] Building CXX object Source/CMakeFiles/CMakeLib.dir/cmOrderDirectories.cxx.o
[ 68%] Building CXX object Source/CMakeFiles/CMakeLib.dir/cmPolicies.cxx.o
g++: internal compiler error: Segmentation fault (program cc1plus)
Please submit a full bug report,
with preprocessed source if appropriate.
See <file:///usr/share/doc/gcc-5/README.Bugs> for instructions.
Source/CMakeFiles/CMakeLib.dir/build.make:2342: recipe for target 'Source/CMakeFiles/CMakeLib.dir/cmPolicies.cx
x.o' failed
make[4]: *** [Source/CMakeFiles/CMakeLib.dir/cmPolicies.cxx.o] Error 4
make[4]: *** Waiting for unfinished jobs....
virtual memory exhausted: Cannot allocate memory
Source/CMakeFiles/CMakeLib.dir/build.make:1598: recipe for target 'Source/CMakeFiles/CMakeLib.dir/cmGeneratorTa
rget.cxx.o' failed
make[4]: *** [Source/CMakeFiles/CMakeLib.dir/cmGeneratorTarget.cxx.o] Error 1
virtual memory exhausted: Cannot allocate memory
Source/CMakeFiles/CMakeLib.dir/build.make:2174: recipe for target 'Source/CMakeFiles/CMakeLib.dir/cmMakefileUti
lityTargetGenerator.cxx.o' failed
make[4]: *** [Source/CMakeFiles/CMakeLib.dir/cmMakefileUtilityTargetGenerator.cxx.o] Error 1
CMakeFiles/Makefile2:2064: recipe for target 'Source/CMakeFiles/CMakeLib.dir/all' failed
make[3]: *** [Source/CMakeFiles/CMakeLib.dir/all] Error 2
Makefile:162: recipe for target 'all' failed
make[2]: *** [all] Error 2
package/pkg-generic.mk:238: recipe for target '/home/ [redacted] /workspace/t113/out/t113/evb1_auto/buildroot/buildro
ot/build/host-cmake-3.8.2/.stamp_built' failed
make[1]: *** [/home/ [redacted] i/out/t113/evb1_auto/buildroot/buildroot/build/host-cmake-3.8.2/.stamp_
built] Error 2
Makefile:96: recipe for target '_all' failed
make: *** [_all] Error 2
make: Leaving directory '/ [redacted] i/workspace/t113/buildroot/buildroot-201902'
12-30 11:02:27.505   27078 E mkcommon  : build buildroot Failed
12-30 11:02:27.520   27078 E mkcommon  : build buildroot rootfs failed
```

图 8-3: Ubuntu 16_04 编译错误示例

**问题原因：**内存不足，需要增加虚拟内存。

**解决方法：**

下文件：

```
sudo gedit /etc/security/limits.conf
```

在 # End of file 下添加以下内容：

```
nvidia hard nofile 65535
nvidia soft nofile 65535
root hard nofile 65535
root soft nofile 65535
ubuntu hard nofile 65535
ubuntu soft nofile 65535

ubuntu hard stack 2024
ubuntu soft stack 2024
nvidia hard stack 2024
nvidia soft stack 2024
root hard stack 2024
root soft stack 2024
```

保存退出，需要重启系统生效。

```
sudo reboot
```

# 8.5 ubuntu14.04 连接 git 服务器报错

[illegible]时，出现网络连接错误：

该报错问题仅出现在 ubuntu14.04 版本上，在 ubuntu20.04 和 ubuntu16.04 上没有出现该错误。

```
                    $ repo init -u http://gerrit.allwinnertech.com:8081/product/tina/manifest
.git -b tina-dev -m t113/product-da-t113-tina5.0-stable.xml
Downloading Repo source from https://mirrors.bfsu.edu.cn/git/git-repo
fatal: unable to access 'https://mirrors.bfsu.edu.cn/git/git-repo/': gnutls_handshake() failed: Han
dshake failed
repo: error: "git" failed with exit status 128
  cwd: /home/[illegible]/workspace/t113/.repo/repo.tmp
  cmd: ['git', 'fetch', '--quiet', '--progress', 'origin', '+refs/heads/*:refs/remotes/origin/*', '
+refs/tags/*:refs/tags/*']
fatal: double check your --repo-rev setting.
fatal: cloning the git-repo repository failed, will remove '.repo/repo'
```

图 8-4: 14_04 版本拉取 git 服务器报错.png

**问题原因：**在前面拉取 repo 仓库时，为了提高速度更换了镜像源：

```
echo export REPO_URL='https://mirrors.bfsu.edu.cn/git/git-repo' >> ~/.bashrc
source ~/.bashrc
```

原因很大可能是因为 https 和 http 的 proxy 的对应的分别是 https 和 http 开头的 proxy server，而 https 的 proxy server 可能无法正常工作。因此，一个解决方法就是把 https 的 proxy server 换成 http 的 proxy server:

```
echo export REPO_URL='http://mirrors.bfsu.edu.cn/git/git-repo' >> ~/.bashrc
source ~/.bashrc
```

重新执行 repo init 操作，可以解决上面报错问题。

## 声明



**商标声明**



**免责声明**