Geniatech

Spec V1.0

Geniatech Rockchip series

Linux Software Development Guide

Foreword

Overview

This document serves as the geniatech rockchip series of Linux software development guidelines, and aims to help software development engineers and technical support engineers get started with geniatech rockchip Linux development and debugging faster. The product development guidance of geniatech rockchip series will be continuously updated on this document in the future.

Product version

Product name	Kernel version	Buildroot version
XPI-RK3128	Linux4.4	2018.02_rc3

Audience

This document (this guide) is mainly applicable to the following engineers: Software development engineers

Revision History

date	version	author	audit	explanation of the revision
2020-11-23	V1.00	zyy	cf	1. Initialize geniatech rockchip linux development guide structure 2. Support xpi-rk3128

catalog

1. Support list

1.1product list

chip	Model no	Software documentation support	functional specification
Rk3288	DR320 DB5 CBD96 DB288	Does not support	
Rk3399	APC3399	Does not support	
Rk3128	XPI-3128	Does not support	
Rk3399pro		Does not support	

1.2Document development support list

chip	The hardware board type	functional specification
xpi-rk3128	Development board	hdmi out、ent、usbx4、wifi、rtc、Lead out 40pin pin cable

2. Document and tool index

2.1 Document Index

The documents released with Rockchip Linux SDK are designed to help developers quickly get started with development and debugging. The content involved in the documents does not cover all development knowledge and problems. The document list is constantly being updated. If you have any questions or needs on the document, please contact usRockchip Linux SDK comes with Develop reference documents (development guide documents), Platform support lists (support list), RKTools manuals (tool usage documents), SoC platform related (chip platform related documents), Linux reference documents (Linux System Development Guide).

2.2 Tool Index

The tools released with Rockchip Linux SDK are used in the development and debugging phase and mass production phase. The tool version will be updated continuously with the SDK update. If you have any questions or needs on the tool, please contact us. Rockchip Linux SDK comes with linux (tools used in the Linux operating system environment) and windows (tools used in the Windows operating system environment) in the tools directory.

```
tools/
├── linux
│   ├──Linux_Pack_Firmware (Linux firmware packaging tool)
│   ├──Linux_SecureBoot (Linux firmware signature tool)
│   ├──Linux_TA_Sign_Tool.rar
│   ├──Linux_Upgrade_Tool (Linux development tool)
└── windows
    ├── AndroidTool (development tool)
    │   ├── AndroidTool_Release_v2.52
    │   └── rockdev (Firmware packaging tool)
    ├── DriverAssitant_v4.7.zip (Driver installation assistant)
    ├── PCBATool_Setup_V1.0.6_0516_3308.exe (PCBA test and PC tool)
    ├── efuse_v1.37.zip (Efuse burn tool)
    ├── FactoryTool_v1.61.zip (Factory mass production tool)
    ├── SecureBootTool_v1.89.zip (firmware signature tool)
    ├── SpiImageTools_v1.36.zip
    ├── EQ_DRC_TOOL_v1.1.zip (Voice and sound effect real-time tuning tool)
    └── WNpctool_Setup_V1.1.9_180118.zip (manufacturer information burning tool)
```

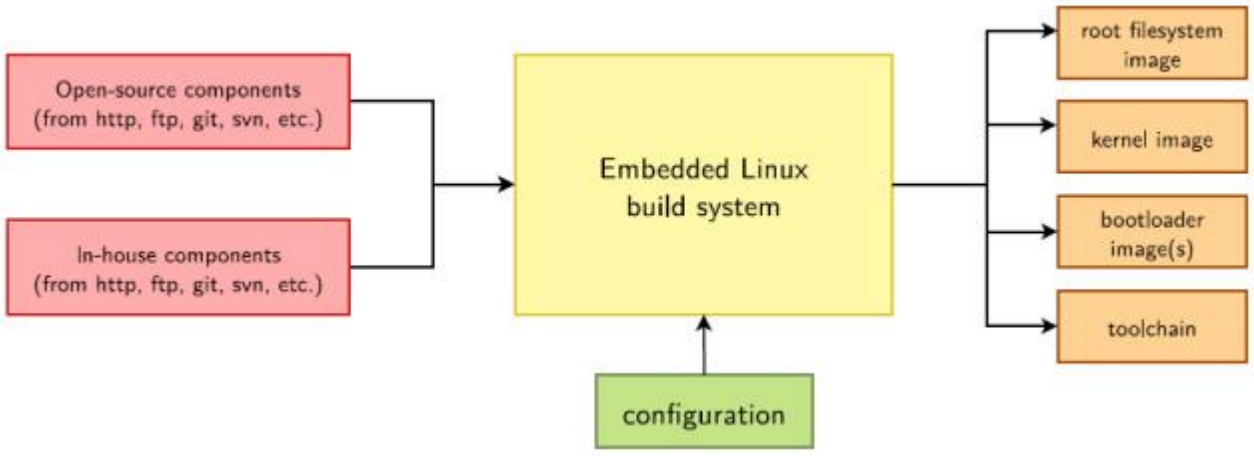
3. SDK software architecture

3.1 SDK overview

3.1.1 buildroot sdk framework

3.1.1 buildroot sdk architectureBuildroot Linux SDK is a software development package based on Buildroot-2018.02, which contains various system source codes, drivers, tools, and application software packages used in Linux-based system development.

Buildroot is an open source embedded Linux system automatic construction framework on the Linux platform. The entire Buildroot is composed of Makefile scripts and Kconfig configuration files. You can configure through Buildroot to compile a complete Linux system software that can be directly burned to the machine to run.



。Buildroot has the following advantages:

1. Build with source code, with great flexibility;
2. Convenient cross-compilation environment for rapid construction;
3. Each system component is easy to configure and convenient for custom development.

3.2 SDK software block diagram

The SDK software framework is divided into four levels: Bootloader, Linux Kernel, and Libraries Applications from bottom to top.The contents of each level are as follows:

- The Bootloader layer mainly provides low-level system support packages, such as Bootloader, U-Boot, ATF related support
- The Kernel layer mainly provides the standard implementation of Linux Kernel, and Linux is also an open operating system. The Linux core of the Rockchip platform is the standard Linux4.4 kernel, which provides basic support for security, memory management, process management, network protocol stacks, etc.; it mainly manages device hardware resources through the Linux kernel, such as CPU scheduling, cache, memory, I/ O etc.
- The Libraries layer corresponds to the general embedded system and is equivalent to the middleware layer. Contains various system basic libraries and third-party open source library support, and provides API interfaces for the application layer. System customizers and application developers can develop new applications based on the Libraries layer API.
- The Applications layer is mainly to implement specific product functions and interaction logic. It needs some system basic libraries and third-party program libraries to support. Developers can develop and implement their own applications and provide various capabilities of the system to end users.

3.3 SDK development process

The Buildroot Linux system is based on Linux Kernel and is an SDK developed for a variety of different product forms. Based on this SDK, system customization and application migration and development can be effectively realized.System development environment and compiled code. The following will briefly introduce the process:

1) Check the system requirements: Before downloading the code and compiling, make sure that the local development equipment can meet the requirements, including the hardware capabilities of the machine, software system, tool chain, etc. Currently, the SDK supports compilation under the Linux operating system environment, and only provides toolchain support under the Linux environment. Other systems such as MacOS and Windows do not currently support it.

2) Build a compilation environment: Introduce various software packages and tools that need to be installed on the development machine, learn about the Linux operating system version that Buildroot Linux has verified, and the library files that it depends on when compiling.

3) Selection of equipment: During the development process, developers are required to select the corresponding hardware board type according to their own needs.

4) Download the source code: After selecting the device type, you need to install the repo tool to download the source code in batches.

5) System customization: Developers can customize U-Boot, Kernel, Buildroot according to the hardware board and product definition used.

6) Compilation and packaging: After having the source code, select the product and initialize the relevant compilation environment, and then execute the compilation commands, including the overall or module compilation and compilation cleanup.

7) Burn and run:

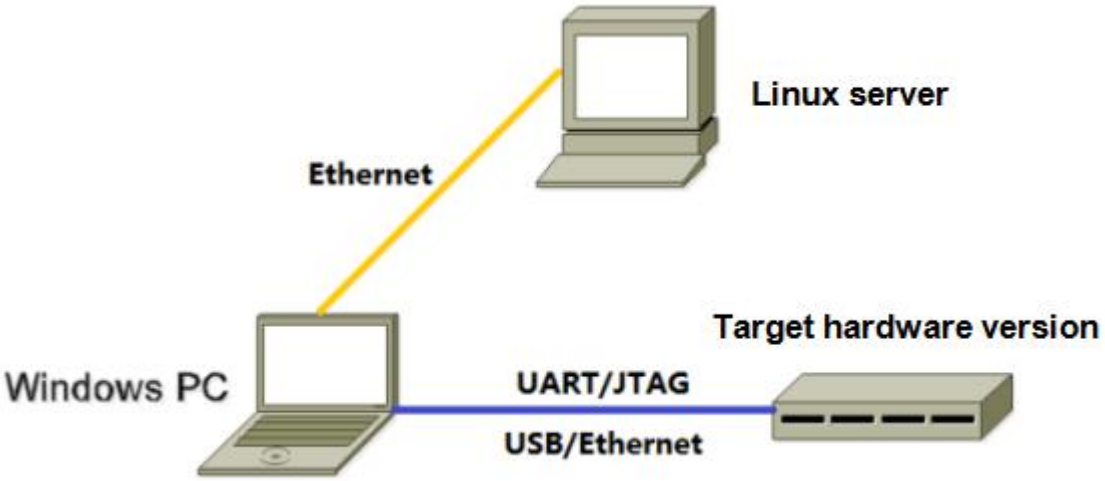
4. Development environment setup

4.1 overview

This section mainly introduces how to build a local compilation environment to compile Rockchip Buildroot Linux SDK source code. The current SDK only supports compilation under Linux and provides a cross-compilation tool chain under Linux.A typical embedded development environment usually includes Linux server, Windows PC and target hardware version. Taking RK3308 as an example, the typical development environment is shown in Figure 4-1.

- A cross-compilation environment is established on the Linux server to provide code update download and code cross-compilation services for software development.

- Windows PC and Linux server share the program, and install SecureCRT or puTTY, remotely log in to the Linux server through the network, perform cross-compilation, and code development and debugging.
- The Windows PC is connected to the target hardware board through the serial port and USB, and the compiled image file can be programmed to the target hardware board, and the system or application program can be debugged.



Note: Windows PC is used in the development environment. In fact, many tasks can also be done on Linux PC, such as using minicom instead of SecureCRT or puTTY. Users can choose by themselves.

4.2 Linux Server development environment setup

The Rockchip Buildroot Linux SDK was developed and tested on Ubuntu 16.04. Therefore, we recommend using Ubuntu 16.04 for compilation. There is no specific test for other versions, and the software package may need to be adjusted accordingly.In addition to system requirements, there are other hardware and software requirements.

- Hardware requirements: 64-bit system, hard disk space greater than 40G. If you do multiple builds, you will need more hard disk space.
- Package dependencies: In addition to python 2.7, make 3.8, and git 1.7, some additional software packages need to be installed, which will be listed in the package installation chapter.

4.2.1 The release package uses the Linux server system version

The SDK development environment installs the following versions of Linux systems, and the SDK is compiled with this Linux system by default:

Ubuntu 16.04.2 LTS

4.2.2 Dependent package installation

After the operating system is installed, and the user has configured the network environment by himself, you can proceed to the following steps to complete the installation of the relevant software packages.

Platform	Install dependent package
xpi-rk3128	1. apt-get update sudo apt-get update sudo apt-get upgrade 2. Install the required dependent software packages when compiling Kernel and U-Boot sudo apt-get install git gnupg flex bison gperf build-essential zip curl zlib1g-dev \ gcc-multilib g++-multilib libc6-dev-i386 lib32ncurses5-dev x11proto-core-dev libx11-dev \ lib32z1-dev ccache libgl1-mesa-dev libxml2-utils xsltproc unzip device-tree-compiler \ liblz4-tool 3. Install the required dependent software packages when compiling Buildroot sudo apt-get install libfile-which-perl sed make binutils gcc g++ bash patch gzip \ bzip2 perl tar cpio python unzip rsync file bc libmpc3 git repo texinfo pkg-config cmake \ tree texinfo If you encounter an error during compilation, you can install the corresponding software package based on the error message.

4.2.3 Introduction to Cross Compiler Toolchain

Since Rockchip Buildroot SDK is currently only compiled under Linux, we also only provide a cross-compilation tool chain under Linux. The prebuilt directory of the compilation tool chain used by U-Boot and Kernel is under prebuilt/gcc, and buildroot uses the tool chain compiled in this open source software.

1) U-Boot and Kernel compilation tool chain

prebuilts/gcc/linux-x86/aarch64/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux -gnu/bin/aarch64-linux-gnu-

Corresponding version

gcc version 6.3.1 20170404 (Linaro GCC 6.3-2017.05)

2)Buildroot Compilation tool chain

buildroot/output/rockchip_rk3308_release/host/bin/aarch64-rockchip-linux-gnu-

Corresponding version

gcc version 6.4.0 (Buildroot 2018.02-rc3-00017-g9c68ede

If you need a toolchain for other platforms or versions, you need to compile it yourself.

After the above environment is prepared, the Linux server development environment has been set up, and the source code can be downloaded and compiled.

4.2.4 Linux programming tool use

The burning tool on Linux system is published in tools\linux\Linux_Upgrade_Tool\

4.3 Windows PC Development environment setup

4.3.1 Development tool installation

Please install editing software such as SourceInsight and Notepad++ by yourself.

4.3.2 usb driver installation

During the development and debugging stage, the device needs to be switched to Loader mode or Maskrom mode, and Rockusb driver needs to be installed to recognize the device normally.

The Rockchip USB driver installation assistant is stored in tools\windows\DriverAssitant_v4.x.zip. Support xp, win7_32, win7_64, win8_32, win8_64 operating systems.

The installation steps are as follows:

4.3.3 Windows burning tool use

The burning tool on the Windows system is released in tools\windows\AndroidTool/AndroidTool_Release, which can be used for development, debugging, and firmware burning in the Windows environment.

4.4 Target hardware board preparation

Please refer to selecting the corresponding hardware board for subsequent development and debugging. The corresponding hardware instruction document will introduce the hardware interface, instruction, and programming operation method.

5. SDK installation preparation

5.1 brief introduction

Rockchip Buildroot Linux SDK

5.2 SDK acquisition

SDK is released through geniatech external server, and customers need to apply for SDK corresponding to our company.

5.2.1 SDK download link

Platform	Download link
XPI-rk3128	http://www.geniatech.net/hefei/sd-release/rk3128-series/SDK-RK3128-Buildroot_RKH190305-HWv1.0-20201020.tar.gz

5.2.2 SDK code compression package

In order to facilitate customers to quickly obtain the SDK source code, geniatech usually provides an init initialization compressed package of the corresponding hardware. Developers can use the following methods.

Take xpi-3288 as an example:

tar xvzf SDK-RK3128-Buildroot_RKH190305-HWv1.0-20201020.tar.gz

cd rk3128linux0227

Platform	FW Package	version
XPI-3288	SDK-RK3128-Buildroot_RKH190305-HWv1.0-20201020.tar.gz	V1.00

5.3 SDK Directory Structure

After the SDK download is complete, you can see the following directory structure in the root directory:

Take xpi-3288 as an example:

```
├─── app
├─── buildroot
├─── build.sh -> device/rockchip/common/build.sh
├─── device
├─── distro
├─── docs
├─── envsetup.sh -> buildroot/build/envsetup.sh
├─── external
├─── IMAGE
├─── init.sh
├─── kernel
├─── Makefile -> buildroot/build/Makefile
├─── mkfirmware.sh -> device/rockchip/common/mkfirmware.sh
├─── npu
```

```
├──── prebuilts
├──── rkbin
├──── rkflash.sh -> device/rockchip/common/rkflash.sh
├──── rockdev
├──── tools
└──── u-boot
14 directories, 6 files
```

- The Buildroot directory stores the buildroot open source project code, and the root file system can be customized
- build.sh is the system compilation script, which can be executed to compile the SDK completely
- The device directory stores board-level configuration and some preset files, boot scripts, etc.
- The docs directory stores relevant documents provided by the SDK
- The external directory stores SDK related libraries and tool source code
- Rockdev stores all image file backups and version compilation information after build.sh is executed
- kernel is part of the source code of the kernel
- The mkfirmware.sh script can package the image files and copy them to the rockdev/ directory
- The prebuilts directory stores the cross-compilation tool chain used for U-Boot and Kernel compilation
- rkbin The directory stores some key binary files of the Rockchip platform, including: ddr.bin,miniloader.bin,bl31.bin, It will be used during U-Boot compilation.
- tools The directory stores development tools, debugging tools, and mass production tools under Windows and Linux environments
- The u-boot directory stores the source code of the U-Boot part

6.SDK Compiling

6.1 Uboot Compiling

- XPI3128

cd rk3128linux0227

./build.sh BoardConfig.mk

./build.sh uboot

After compiling, three image files of trust.img, px30_loader_v1.07.107.bin and uboot.img will be generated.

ERROR1: If rk3128 is not selected in ./build.sh BoardConfig.mk, proceed as follows:
cd device/rockchip/
rm .BoardConfig.mk
ls -n rk3128/BoardConfig.mk .BoardConfig.mk

6.2 Kernel Compiling

- XPI3128

./build.sh kernel

After compiling, a boot.img image file will be generated.

Kernel configuration

HW version	The silk print on board	Corresponding configuration file
XPI3128 V1.00	RKH190305 VER1.0	rockchip_linux_defconfig rk3128-fireprime.dts

6.3 Buildroot Compiling

- XPi3128

./build.sh buildroot

After compiling, A rootf.img image file will be generated in rockdev/.

6.4 Automatically compile scripts

In order to improve the efficiency of compilation and reduce the possible misoperation of manual compilation, the SDK integrates fully automated compilation scripts to facilitate firmware compilation and backup.

1) The original file of the fully automated compilation script is stored in:device/rockchip/common/build.sh

2) Modify the specific variables in the device/rockchip/rkxx (chip platform)/BoardConfig.mk script to compile the corresponding product firmware.

Such as xpi3128 platform, you can modify the device/rockchip/rk3128/BoardConfig.mk file:

3) Execute automatic compilation script:

./build.sh

The script will automatically configure environment variables, compile U-Boot, compile Kernel, compile Buildroot, compile Recovery and then generate firmware.

4) Script generated content:

The script will copy the compiled firmware:

Under the Rockdev/ directory, the specific path is subject to actual generation. Every time you compile, a new directory is created and saved, the firmware version of the debugging development process is automatically backed up, and various information about the firmware version is stored.

- XPI-RK3288

HW version	The silk print on board	Compiling
XPI-RK3288 V1.00	RKH190305 VER1.0	1. Automatic compilation cd rk3128linux0227 ./build.sh BoardConfig.mk ./build.sh 2.编译生成 rockdev/下 boot.img misc.img parameter.txt rootfs.ext4 trust.img update.img MiniLoaderAll.bin oem.img recovery.img rootfs.img uboot.img userdata.img

6.5 Partial compilation of modules

In order to facilitate development and debugging, the fully automated compilation script also supports the compilation of individual modules, which is convenient for module debugging. Some modules can be specified and compiled. For partial compilation of the module, see subsequent updates.

7.SDK Image burning

7.1 Overview

This chapter mainly introduces the process of how to program the completed image file (image) and run it on the hardware device. Several image programming tools provided by the Rockchip platform are introduced as shown in the table below. You can choose a suitable programming method for programming. Before programming, you need to install the latest USB driver, see 4.3.2 USB Driver Installation for details.

Tool	Operating system	Description
Rockchip Development Tools	Windows	Discrete upgrade firmware and the entire update upgrade firmware tool

7.2 Introduction to burning mode

Several modes of Rockchip platform hardware operation are shown in the table below, Only when the device is in Maskrom and Loader mode, the firmware can be programmed or the on-board firmware can be updated.

Mode	Tool upgrading	Introduction
Maskrom	Support	When Flash has not burned the firmware, the chip will boot into Maskrom mode, and the firmware can be burned for the first time; during development and debugging, if the Loader fails to start normally, you can also enter Maskrom mode to burn the firmware.
Loader	Support	In Loader mode, firmware can be programmed and upgraded. A certain partition image file can be programmed separately through the tool to facilitate debugging.

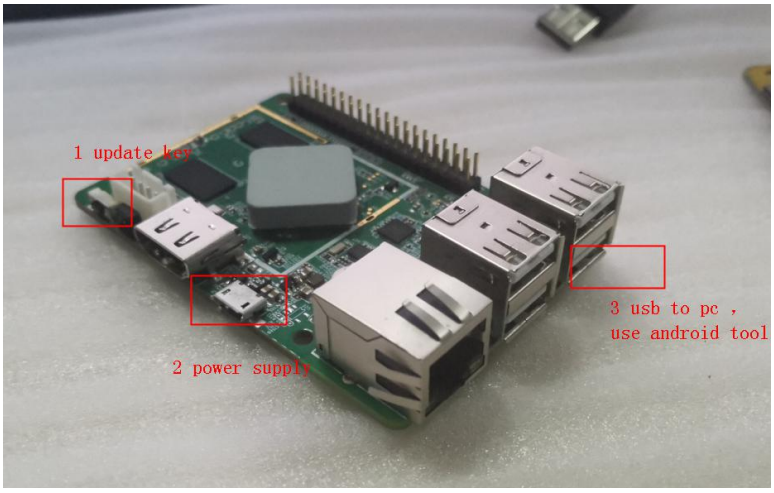
7.3 Rockchip Development Tool Programming

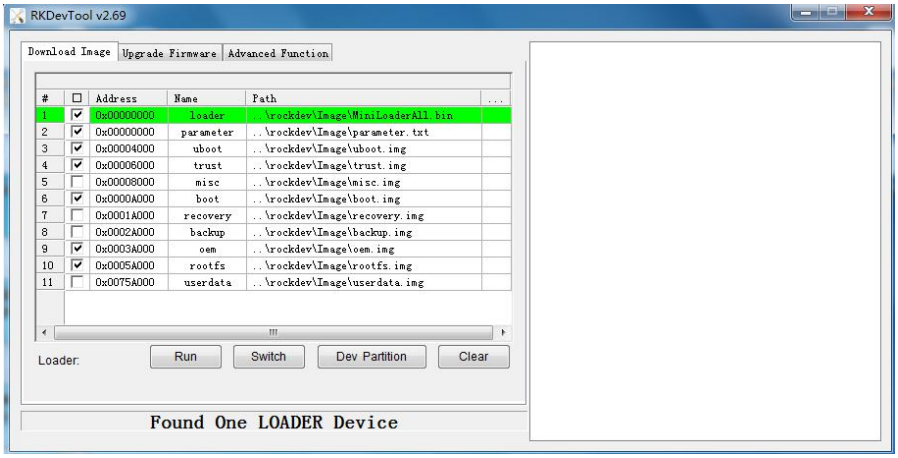
For the programming instructions, please refer to "Rockchip Development Tools Manual.pdf" under the docs\RKTools manuals directory.

The SDK provides programming tools, as shown below. After compiling and generating the corresponding firmware, enter the programming mode and you can flash the machine. For machines that have already burned other firmware, you can choose to burn the firmware again, or choose a low-level device, erase idb, and then flash.

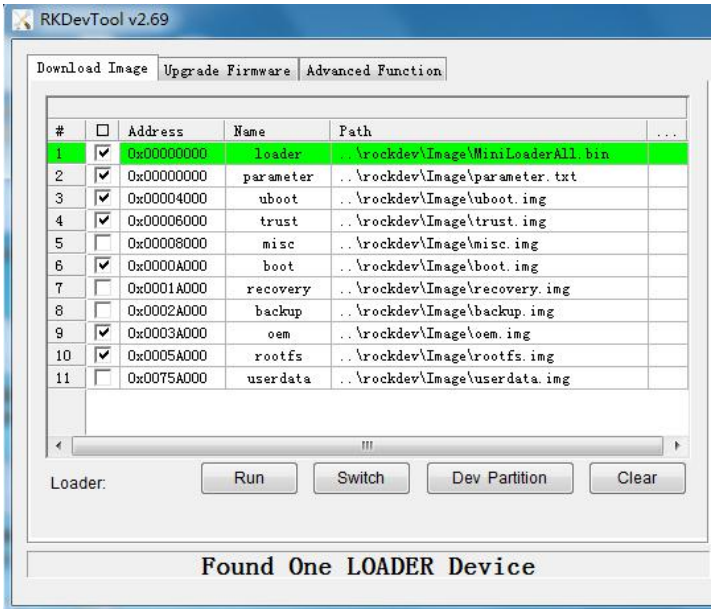
7.3.1 Enter programming mode

Take xpi-rk3128 as an example: first press and hold the button next to the infrared head, power on, and connect to the USB port for programming.





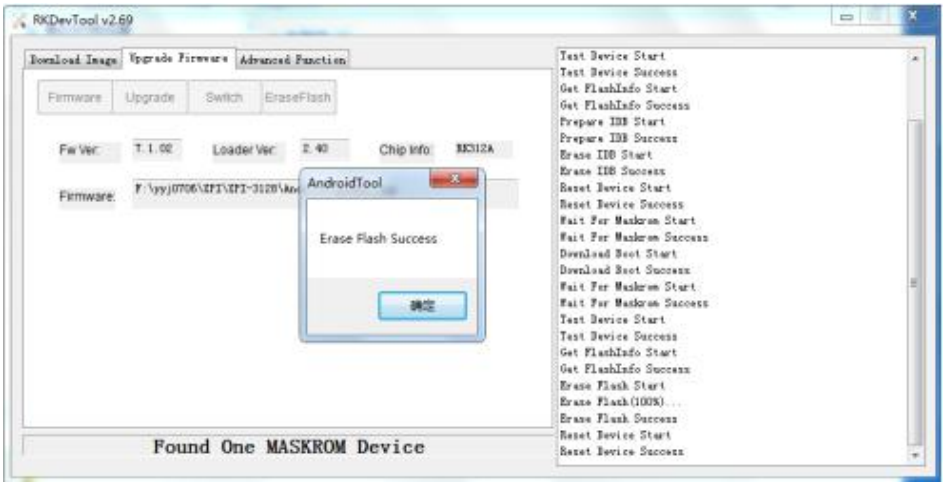
7.3.2 Downloading image



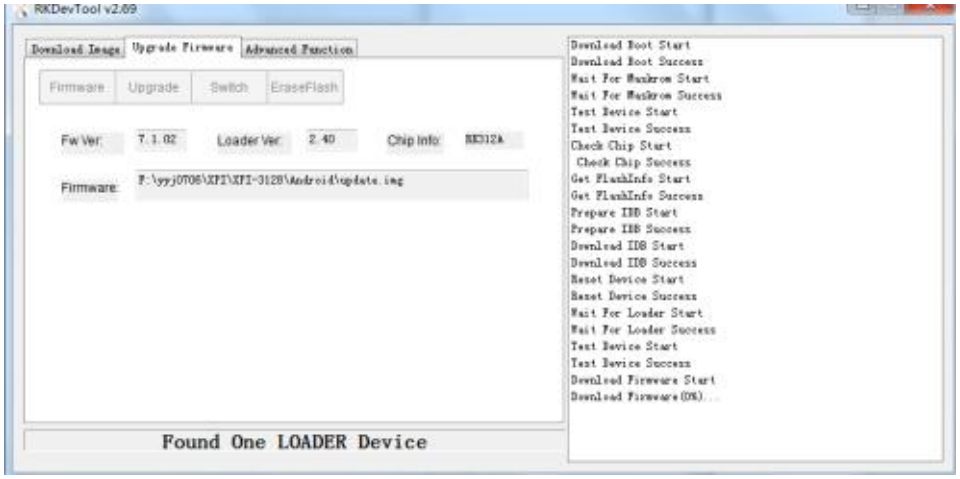
1）The USB connects hardware board and enters the programming mode.

2）Open the tool and click the Download Mirror menu, click… column will pop up a file selection box, you can select the img of the corresponding partition, and configure several other items at once. geniatech rockchip series selection: 1, 2, 3, 4, 6, 9, 10

3）After the configuration is complete, click "Execute" and you can see the blank box on the right to enter the download prompt. The "Low" button is used to erase the device, and the "Clear" button is to clear the text in the edit box.



7.3.3 Upgrade firmware (update.img exists only for buildroot)



1）Click the firmware to select the newly packaged update.img file, and click the upgrade button to download. (Note that the device must be in download mode).

8.U-boot Development

8.1 Rockchip U-Boot introduction

Rockchip U-Boot next-dev branch is the version that Rockchip cut out from the official version of U-Boot v2017.09 for development.

- Support firmware startup of RK Android platform;
- Start with the latest Android AOSP (such as GVA) firmware;
- Support Linux Distro firmware startup;
- Support Rockchip miniloader and SPL/TPL two pre-loader boot;
- Support LVDS, EDP, MIPI, HDMI and other display devices;
- Support Emmc, Nand Flash, SPI NOR flash, SD card, U disk and other storage devices to start;
- Support FAT, EXT2, EXT4 file system;
- Support GPT, RK parameter partition format;
- Support boot logo display, charging animation display, low power management, power management;
- Support I2C, PMIC, CHARGE, GUAGE, USB, GPIO, PWM, GMAC, EMMC, NAND, interrupt and other drivers;
- Support RockUSB and Google Fastboot two USB gadgets to burn EMMC;
- Support Mass storage, ethernet, HID and other USB devices;
- Support dtb using Kernel;
- Support dtbo function;

8.2 Platform compilation

8.2.1 rkbin

The rkbin project mainly stores Rockchip's non-open source bin files (trust, loader, etc.), scripts, packaging tools, etc., so rkbin is just a "toolkit" project.

The rkbin project needs to maintain the same level directory relationship with the U-Boot project, otherwise the rkbin warehouse will not be found during compilation. When compiling the U-Boot project, the compiling script will index the relevant bin files and packaging tools from the rkbin warehouse, and finally generate trust.img, uboot.img, loader and other relevant firmware in the U-Boot root directory.

8.2.2 gcc Tool chain

The default compiler used is version gcc-linaro-6.3.1:

32-bit compiler：gcc - linaro - 6.3.1 - 2017.05 - x86_64_arm - linux - gnueabihf

64-bit compiler：gcc - linaro - 6.3.1 - 2017.05 - x86_64_aarch64 - linux - gnu

The toolkit provided by Rockchip: prebuilts is used by default. Please ensure that it maintains the same level directory relationship with the U-Boot project.

If you need to change the compiler path, you can modify the content in the compilation script ./make.sh:

GCC_ARM32=arm - linux - gnueabihf -
GCC_ARM64=aarch64 - linux - gnu -
TOOLCHAIN_ARM32=../prebuilts/gcc/linux - x86/arm/gcc - linaro - 6.3.1 - 2017.05-X86_64_ arm- linuxgnueabihf/bin
TOOLCHAIN_ARM64=../prebuilts/gcc/linux - x86/aarch64/gcc - linaro - 6.3.1 - 2017.05 - x86_64_aarch64 - linux -gnu/bin

8.2.3 Compiling

Compiling command:

./make.sh [board] - - - - [board] The source of this name is: configs/[board]_defconfig file。

Command example:

./make.sh evb- rk3308 - - - - build for evb- rk3308_defconfig
./make.sh firefly- rk3288 - - - - build for firefly- rk3288_defconfig

9.Kernel Development

9.1.1 Overview

Earlier versions of Linux Kernel configured the board-related information directly in the board-level configuration file, such as IOMUX, the default high/low GPIO, and the client device information under each I2C/SPI bus. In order to abandon this 'hard code' approach, Linux introduces the concept of Device Tree to describe different hardware structures.

Device Tree data is highly readable and follows the DTS specification, which is usually described in .dtsi and .dts source files. In the process of kernel compilation, it is compiled into a binary file of .dtb. During the boot-up phase, dtb will be loaded into an address space of RAM by bootloader (such as U-Boot), and the address will be passed to Kernel space as a parameter. The kernel parses the entire dtb file and extracts the information of each device for initialization.

This article aims to introduce how to add a new board dts configuration and some common dts syntax descriptions. A more detailed introduction to the dts syntax is beyond the scope of this article. If you are interested, please refer to:1. https://www.devicetree.org/specifications/

2. Documentation/devicetree/bindings

9.1.2 Add a product DTS

9.1.2.1 Create dts file

Linux Kernel currently supports the use of dts on multiple platforms. The dts files of the RK platform are stored in:

ARM:arch/arm/boot/dts/

ARM64 :arch/arm64/boot/dts/rockchip

The general naming rule for dts files is "soc-board_name.dts", such as rk3308-evb-dmic-i2s-v10.dts. soc refers to the name of the chip, and board_name is generally named according to the silk screen of the board. If your board is an all-in-one board, you only need a dts file to describe it.

```
rk3308-ai-va-v10.dts
└── rk3308.dtsi
```

If the hardware design is the structure of the core board and the backplane, or the product has multiple product forms, you can put the common hardware description in the dtsi file, and the dts file describes the different hardware modules, and through include "xxx.dtsi" The common hardware description is included.

```
├── rk3308-evb-amic-v10.dts
│   ├── rk3308-evb-ext-v10.dtsi
│   └── rk3308-evb-v10.dtsi
│       └── rk3308.dtsi
└── rk3308-evb-dmic-i2s-v10.dts
    ├── rk3308-evb-ext-v10.dtsi
    └── rk3308-evb-v10.dtsi
        └── rk3308.dtsi
```

9.1.2.2 Modify the Makefile of the directory where dts is located

```
diff --git a/arch/arm64/boot/dts/rockchip/Makefile
b/arch/arm64/boot/dts/rockchip/Makefile
            index 073281d..7c329d4 100644
            --- a/arch/arm64/boot/dts/rockchip/Makefile
            +++ b/arch/arm64/boot/dts/rockchip/Makefile
            @@ - 1,6 +1,9 @@
            dtb-$(CONFIG_ARCH_ROCKCHIP) += px30-evb-ddr3-v10.dtb
            dtb-$(CONFIG_ARCH_ROCKCHIP) += px30-evb-ddr3-lvds-v10.dtb
            dtb-$(CONFIG_ARCH_ROCKCHIP) += px30-evb-ddr4-v10.dtb
            +dtb-$(CONFIG_ARCH_ROCKCHIP) += rk3308-evb-amic-v10.dtb
```

When compiling kenrel, you can directly make dts-name.img (such as rk3308-evb-amic-v10.img) to generate the corresponding resource.img (including dtb data).

9.1.2.3 Several explanations of dts syntax dts syntax can be like c/c++, through #include xxx.dtsi to include other public dts data. The dts file will inherit the attributes and values of all device nodes in the contained dtsi file.

If property is defined in multiple dts/dtsi files, its value will ultimately be the definition of dts. All controller nodes related to the chip will be defined in soc.dtsi. To enable the device function, you need to set its status in the dts file as "okay".

```
/dts-v1/;
#include "rk3308-evb-v10.dtsi"


/ {
    /* rk3308-evb-v10.dtsi also defines this attribute, and the final value is the
defined value of this dts.
       */
    model = "Rockchip RK3308 evb analog mic board";
    compatible = "rockchip,rk3308-evb-amic-v10", "rockchip,rk3308";
    /* The node under the root node
is a new device node */
    sound {
        compatible = "simple-audio-card";
        ...
    };
};
/* The node under the non-root node is the device node described by the reference
included dtsi, and the value can be reset */
&i2s_8ch_2{
    status = "okay";
    #sound-dai-cells = <0>;
};
```

9.2 ARM、GPU、DDR Frequency modification

DVFS （Dynamic Voltage and Frequency Scaling）, this is a real-time voltage and frequency adjustment technology. At present, the modules supporting DVFS in the 4.4 kernel include CPU, GPU, and DDR.

CPUFreq is a set of kernel developers defined to support dynamic adjustment of low CPU power consumption, while taking into account CPU performance. CPUFreq CPU usage, the current kernel version provides the following strategies:

- interactive：Dynamic frequency adjustment according to CPU load;
- conservative：Conservative strategy, adjust frequency and voltage step by step;
- ondemand：Dynamic frequency and voltage adjustment according to CPU load, slower response than interactive strategy;
- userspace：Users set voltage and frequency by themselves, the system will not automatically adjust;
- powersave：Prioritize power consumption, always set the frequency to the lowest value;
- performance：performance first, always set the frequency to the highest value;

For detailed module functions and configuration, please refer to the docs\Develop reference documents\DVFS\ directory. A53/A72/GPU/DDR have corresponding debugging interfaces, which can be operated through ADB commands. The corresponding interface catalog is as follows:

A53 : /sys/devices/system/cpu/cpu0/cpufreq/

A72: /sys/devices/system/cpu/cpu4/cpufreq/

GPU: /sys/class/devfreq/ff9a0000.gpu/

DDR: /sys/class/devfreq/dmc/

这些目录下有如下类似节点：

available_frequencies：Show supported frequencies

available_governors： Display supported frequency conversion strategies

cur_freq：Show current frequency

governor：Display the current frequency conversion strategy

max_freq：Display the current maximum running frequency

min_freq：Display the current minimum running frequency

Take RK3399/RK3399pro GPU as an example for fixed frequency operation, the process is as follows:

See which frequencies are supportedcat /sys/class/devfreq/ff9a0000.gpu/available_frequencies

Switch frequency conversion strategy

echo userspace > /sys/class/devfreq/ff9a0000.gpu/governor

Fixed frequency

echo 400000000 > /sys/class/devfreq/ff9a0000.gpu/userspace/set_freq

cat /sys/class/devfreq/ff9a0000.gpu/cur_freq

10.BuildrootDevelopment

10.1 Buildroot Development basis

This section briefly introduces some common configuration modifications in Buildroot development. Rockchip Buildroot Linux SDK uses Buildroot-2018.02 version.

Buildroot Basic Development Guide, See "The Buildroot User Manual.pdf" in the docs\ Linux reference documents\ directory.

You can also directly visit the official website link to browse:http://buildroot.org/downloads/manual/manual.html

10.1.1 Default configuration selection and compilation

After the customer configures the compilation dependencies according to the actual compilation environment, follow the steps below and execute make.

$ source buildroot/build/envsetup.sh
You're building on Linux
Lunch menu...pick a combo:
1. rockchip_rk3308_release
2. rockchip_rk3308_debug
3. rockchip_rk3308_robot_release
4. rockchip_rk3308_robot_debug
5. rockchip_rk3308_mini_release

Which would you like? [1]
If you select rockchip_rk3308_release, enter the corresponding serial number 1.
$ make
After compiling, execute the mkfirmware.sh script in the SDK root directory to generate firmware
make Perform the following process

- Download source code
- Configure, compile and install cross toolchain
- Configure, compile, and install selected packages
- Install the selected format to generate the root file system

Buildroot output results are saved in the output directory. The specific directory is determined by the configuration file. The above example is saved in the buildroot/output/rockchip_rk3308_release directory. The subsequent compilation can be

Execute in the buildroot/output/rockchip_rk3308_release directory or the project root directory (make menuconfig can also be executed in the project root directory). This directory includes several subdirectories:

- image：Contains the compressed root file system image file
- build/: Contains all the source files, including the host tools and selected packages required by Buildroot. This directory contains the source code of all modules.
- staging/: This directory is similar to the directory structure of the root file system and contains all the header files and libraries generated by compilation, as well as other development files, but they are not tailored and are relatively large, and are not suitable for the target file system.
- target/: Contains a complete root file system. Compared with staging, it has no development files, no header files, and binary files are also strip processed.
- host/: The tools required for host-side compilation include cross-compilation tools.

10.1.2 Module configuration compilation

The above steps have selected a default configuration, but it may not meet our needs. We may need to add some third-party packages or modify the configuration options of the package. Buildroot supports graphical selection and configuration.

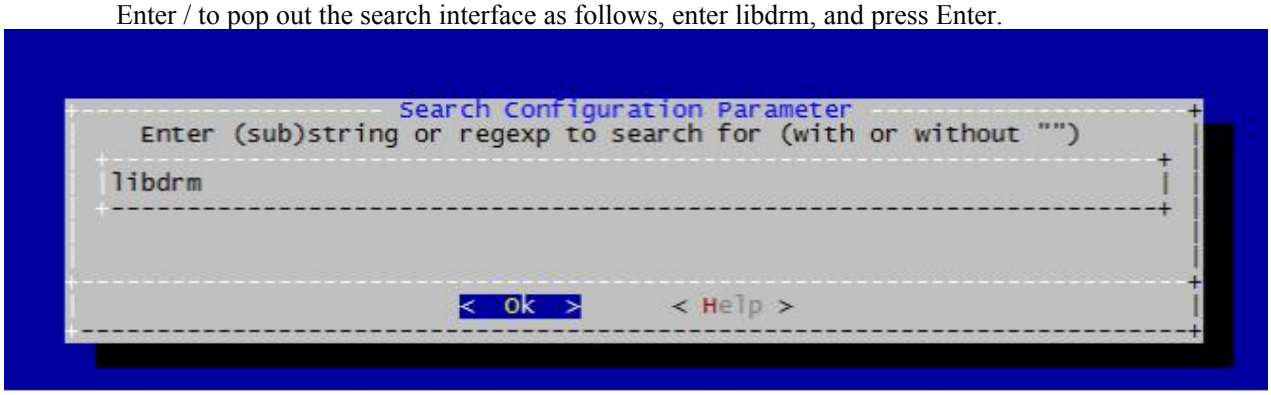
Buildroot supports make menuconfig|nconfig|gconfig|xconfig

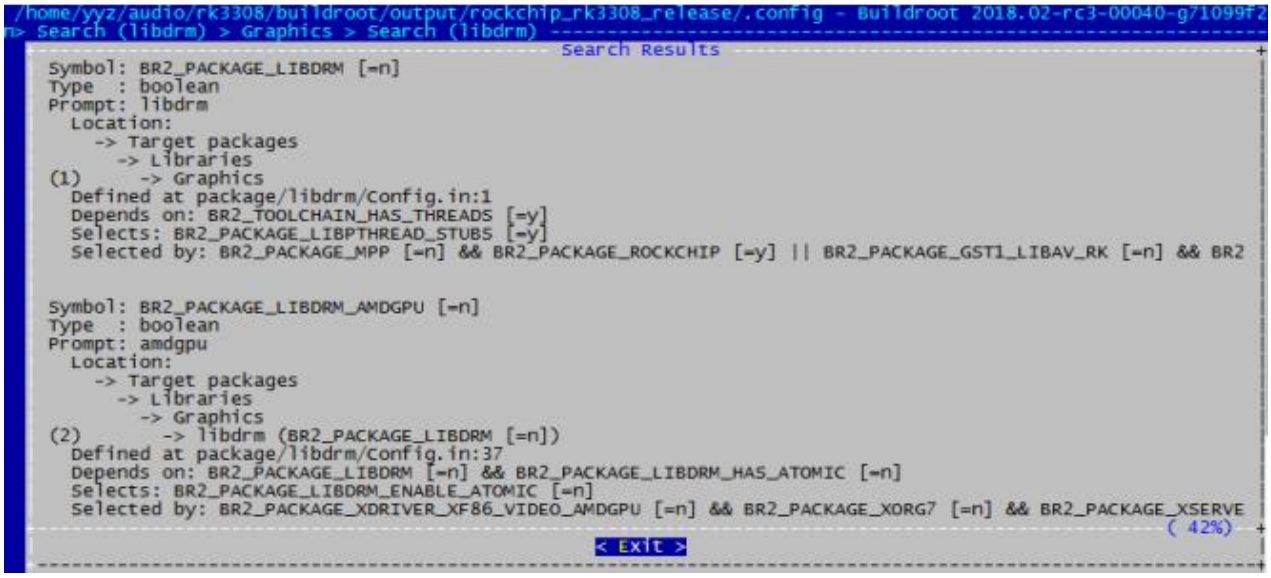
Example, add libdrm support

After executing the source command and configuring the environment variables, you can go to the root directory:

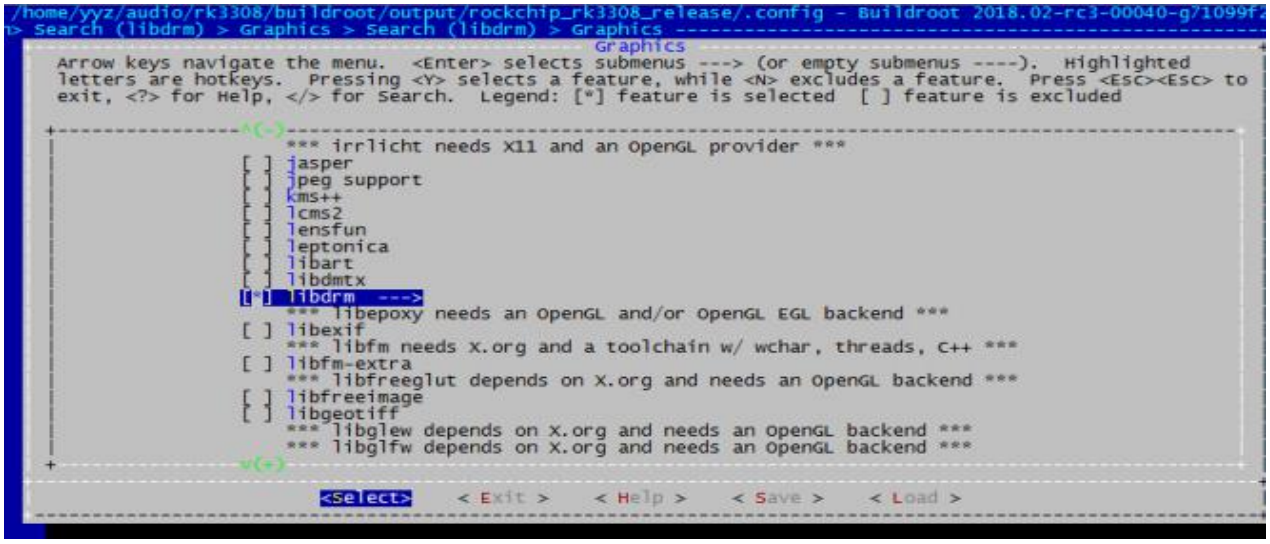
make menuconfig

Enter / to pop out the search interface as follows, enter libdrm, and press Enter.





Select 1, and then press space to select up



Then save and exit, save the configuration command, the configuration file under the buildroot/configs/ directory will be modified;

make savedefconfig

Compile libdrm, generate libdrm

make libdrm

10.1.3 Busybox configuration modification

Configuration command:

make busybox-menuconfig

After the modification is completed, save the configuration through the command:

make busybox-update-config

10.1.4 Cross compilation tool

After Buildroot is compiled, a cross-compilation tool will be generated under the specified output directory host directory, which we can use to compile the target program. The cross compilation tool directory generated by the default configuration is:

buildroot/output/rockchip_rk3308_release/host/usr/bin/output/host/usr/bin/

We can directly compile the program with the cross compilation tool

./buildroot/output/rockchip_rk3308_release/host/usr/bin/output/host/usr/bin/aar ch64-rockchip-linux-gnu-gcc main.c -o test

Floating point support (neon support is turned on in the following configuration), RK3308 supports crc/crypto/fp/simd features, the configuration is as follows:

CFLAGS += -mcpu=cortex-a35+crc+crypto

10.1.5 Add local source code package

The above introduction is in the case that Buildroot already has a source code package, we can choose to open the compilation, if Buildroot does not exist or how to integrate the application written by ourselves into Buildroot?

Buildroot supports a variety of module compilation methods, including generic-package, cmake-package, autotools-package, etc. We take generic-package as an example;

Example: package/rockchip/alsa_capture

1. Create the directory package/rockchip/alsa_capture
2. Create Config.in

```
config BR2_PACKAGE_ALSA_CAPTURE
bool "Simple ALSA Capture Demo"
```

3. Create alsa_capture.mk, where the source directory points to external/alsa_capture/src

```
################################################
###########
 #
 # alsa_capture #
 ################################################
###########
 ifeq ($(BR2_PACKAGE_ALSA_CAPTURE), y)
 ALSA CAPTURE VERSION:=1.0.0
 ALSA_CAPTURE_SITE=$(TOPDIR)/../external/alsa_capture/src
 ALSA_CAPTURE_SITE_METHOD=local

 define ALSA CAPTURE  BUILD CMDS
     $(TARGET_MAKE_ENV) $(MAKE) CC=$(TARGET_CC) CXX=$(TARGET_CXX) -C
$(@D)
 endef
 define ALSA_CAPTURE_CLEAN_CMDS
     $(TARGET_MAKE_ENV) $(MAKE) -C $(@D) clean
 endef
 define ALSA_CAPTURE_INSTALL_TARGET_CMDS
     $(TARGET_MAKE_ENV) $(MAKE) -C $(@D) install
 endef
```

```
 define ALSA_CAPTURE_UNINSTALL_TARGET_CMDS
     $(TARGET_MAKE_ENV) $(MAKE) -C $(@D) uninstall
 endef
 $(eval $(generic-package))
 endif
```

4. Create a directory external/alsa_capture/src， write alsa_capture.c

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
printf(“hello world\n”);
return 0;
}
```

write Makefile 文件

```
DEPS =
OBJ = alsa_capture.o
CFLAGS = -std=c++11 -lasound
%.o: %.cpp $(DEPS)
  $(CC) -c -o $@ $< $(CFLAGS)
alsa_capture: $(OBJ)
  $(CXX) -o $@ $^ $(CFLAGS)
.PHONY: clean
clean:
rm -f *.o *~ alsa_capture
.PHONY: install
install:
cp -f alsa_capture $(TARGET_DIR)/usr/bin/
.PHONY: uninstall
uninstall:
      rm -f $(TARGET_DIR)/usr/bin/alsa_capture
```

6. Add the new package to the Buildroot compilation system;
7. Modify package/rockchip/Config.in and add the following line

```
source "package/rockchip/alsa_capture/Config.in"
```

8. Configuration selection package

make menuconfig and select the alsa_capture package;

9. Compile

```
make alsa_capture
```

10. Pack into the file system

```
make
```

11. Recompile the package after modifying the source code

```
make alsa_capture-rebuild
```

10.1.6 fs-overlay

The root file system is compiled by default. Some configuration files may not meet the customization requirements. At this time, fs-overlay can be used. The fs-overlay directory will be replaced to the file system directory at the final stage of compilation and packaged into the root file system. . The fs-overlay path is specified by the default configuration file:

```
BR2_ROOTFS_OVERLAY="board/rockchip/rk3308/fs-overlay"
```

For example, we will modify the fstab file and add debugfs and pstore:

```
vi board/rockchip/rk3308/fs-overlay/etc/fstab
# <file system> <mount pt>      <type>  <options>        <dump>  <pass>
/dev/root       /               ext2    rw,noauto       0       1
proc            /proc           proc    defaults        0       0
devpts          /dev/pts        devpts  defaults,gid=5,mode=620 0       0
tmpfs           /dev/shm        tmpfs   mode=0777        0       0
tmpfs           /tmp            tmpfs   mode=1777       0       0
tmpfs           /run            tmpfs   mode=0755,nosuid,nodev  0       0
sysfs           /sys            sysfs   defaults        0       0
debug           /sys/kernel/debug       debugfs defaults        0       0
pstore          /sys/fs/pstore  pstore  defaults        0       0
```

10.1.7 SDK Common configuration modification

10.1.7.1 Flash Type modification

Configuration file: device/rockchip/rk3308/BoardConfig.mk, the default configuration is nand device.

```
# Set flash type.
# support <emmc, nand, spi_nand, spi_nor>
FLASH_TYPE := emmc
```

Configuration instructions:

- EMMC device

```
FLASH_TYPE=emmc
```

- NAND device

```
FLASH_TYPE=nand
```

10.1.7.2 Rootfs switch to ext2

Rootfs can be configured to read and write ext2 file system, which is convenient for system debugging.

1. Modify the bootargs configuration in Kernel:

```
diff --git a/arch/arm64/boot/dts/rockchip/rk3308-evb-v10.dtsi
b/arch/arm64/boot/dts/rockchip/rk3308-evb-v10.dtsi
index d39faf8..549af2e 100644
--- a/arch/arm64/boot/dts/rockchip/rk3308-evb-v10.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3308-evb-v10.dtsi
@ -12,7 +12,7 @@
compatible = "rockchip,rk3308-evb", "rockchip,rk3308";
chosen {
-               bootargs = "earlycon=uart8250,mmio32,0xff0c0000 swiotlb=1
console=ttyFIQ0 root=PARTUUID=614e0000-0000 rootfstype=squashfs rootwait";
+               bootargs = "earlycon=uart8250,mmio32,0xff0c0000 swiotlb=1
console=ttyFIQ0 root=PARTUUID=614e0000-0000 rootfstype=ext2 rootwait";
        };
```

2.

1. Modify the parameter file corresponding to device\rockchip\rk3308\rockimg\ to ensure that the rootfs partition is large enough to store the partition image.

2. Modify the rootfs file system type in device\rockchip\rk3308\BoardConfig.mk:

```
diff --git a/BoardConfig.mk b/BoardConfig.mk
index 8adbcdd..3a2be4c 100755
--- a/BoardConfig.mk
+++ b/BoardConfig.mk
@@ -2,8 +2,8 @@
 SDK_ROOT := $(PWD)/../../..

 # Set rootfs type, see buildroot.
-# ext4 squashfs
-ROOTFS_TYPE := squashfs
+# ext4 ext2 squashfs
+ROOTFS_TYPE := ext2
```

4. The rootfs partition ext2 file system image will be automatically packaged and generated, or it can be obtained directly from the following path:

10.2 Buildroot RK Package introduction

Rockchip platform has added support for many software packages on Buildroot, which are unified under the directory of buildroot/package/rockchip. The following briefly introduces the software packages on the SDK:

buildroot/package/rockchip

- adbd----------adb daemon
- alexaClientSDK---------- alexa voice SDK
- alsa_capture----------alsa Equipment recording demo program
- cypress_bsa---------- Cypress Bluetooth protocol stack and demo
- DuerClientSDK---------- Baidu Voice DuerOS SDK
- gst1-libav-rk---------- gstreamer libav plugin
- gstreamer1-iep----------gstreamer iep plugin
- gstreamer1-rockchip------gstreamer rk plugin
- libcutils---------- Cutils library ported from Android
- libiep----------iep library
- libion----------ion library
- liblog---------- Log library ported from Android
- LocalPlayer---------- Local non-screen music player
- mdev_mount----------mdev Automount script
- mpp----------mpp library
- pcba----------pcba test program
- pm-suspend-api---------- Sleep wake up multi-process api and demoo program
- rkwifibt---------- wifi, bt firmware and chip configuration
- rockchip_modules---------- Kernel modules driver
- rockchip_test---------- Test script
- rockchip_utils---------- Some general programs, such as the io command
- rv1108-firmware----------rv1108 firmware
- softap----------wifi Start ap mode
- softapServer----------wifi Network configuration service
- wakeWordAgent----------alexa Wake word agent
- wifiAutoSetup----------wifi smart config Network distribution program
- wpa_supplicant_realtek-----wpa_supplicant realtek version

10.3 Pre-built system applications or data

10.3.1 oem.img description

oem.img ： oem 分 The zone image file, which is in the ext2 file system format by default, is used to preset the customer's application; directly execute the packaging command to package the oem partition image.

./mkfirmware.sh

generate rockdev/oem.img;

10.3.2 oem Mirror packaging directory modification

The oem partition packs the dueros directory device/rockchip/rk3308/oem by default. You can modify the packaging directory and oem partition file system by modifying the following configuration files.

oem path currently has three configurations:

- oem：No voice algorithm program is packaged, only basic functions
- dueros：Baidu Voice SDK, default configuration
- aispeech：Spitz Voice SDK
- iflytekSDK：HKUST iFLYTEK Voice SDK

Configuration file：device/rockchip/rk3308/BoardConfig.mk

```
# Set oem partition type.
# ext2 squashfs
OEM_PARTITION_TYPE=ext2

#OEM config: oem/dueros/aispeech/iflytekSDK
OEM_PATH=oem
```

10.3.3 userdata.img description

userdata.img: userdata partition image file, the default is ext2 file system format, used to store runtime data, generally a readable and writable file; directly execute the packaging command to package the userdata partition image.

mkfirmware.sh

generate rockdev/userdata.img;

10.4 New partition configuration

Please refer to \RKDocs\RKTools manuals\Android to add a partition configuration guide V1.00.pdf

10.5 OTA upgrade

10.5.1 Compile dependency

Rootfs System dependent configuration：

BR2_PACKAGE_RECOVERYSYSTEM=y

Recovery Compile command：

./build.sh recovery

./mkfirmware.sh

After compilation, misc.img and recovery.img will be generated in the rockdev directory

10.5.2 Upgrade firmware production

Upgrade firmware can support complete partition partition upgrade, or specify partition upgrade. The package-file file can be modified to remove the partitions that are not to be upgraded, which can reduce the size of the upgrade package (update.img).

Note: recovery.img does not support upgrades temporarily and cannot be packaged into it.

10.5.3 upgrade

Under rootfs, run from the command line:

recoverySystem ota /xxx/update.img

The machine will reboot into recovery and upgrade, if the upgrade package is placed in the USB flash drive, execute the following command:

recoverySystem ota /mnt/usb_storage/update.img

Available path

Mount path of the U disk: /mnt/usb_storage/

Mount path of sdcard: /mnt/external_sd/

Mount path of flash: /userdata/

10.6 Reset

We save the configuration files that can be read and written in the userdata partition. The factory firmware will default some configuration parameters. The user will generate or modify the configuration file after a period of use. Sometimes the user needs to clear the data, and we need to restore to the factory configuration.

● Solution: When the firmware is packaged, userdata.img is generated, and when the user requests to restore the factory configuration, the userdata partition is formatted.

● SDK implementation: Function keys RECOVERY + VOLUMEUP trigger to restore factory configuration, please refer to the code:

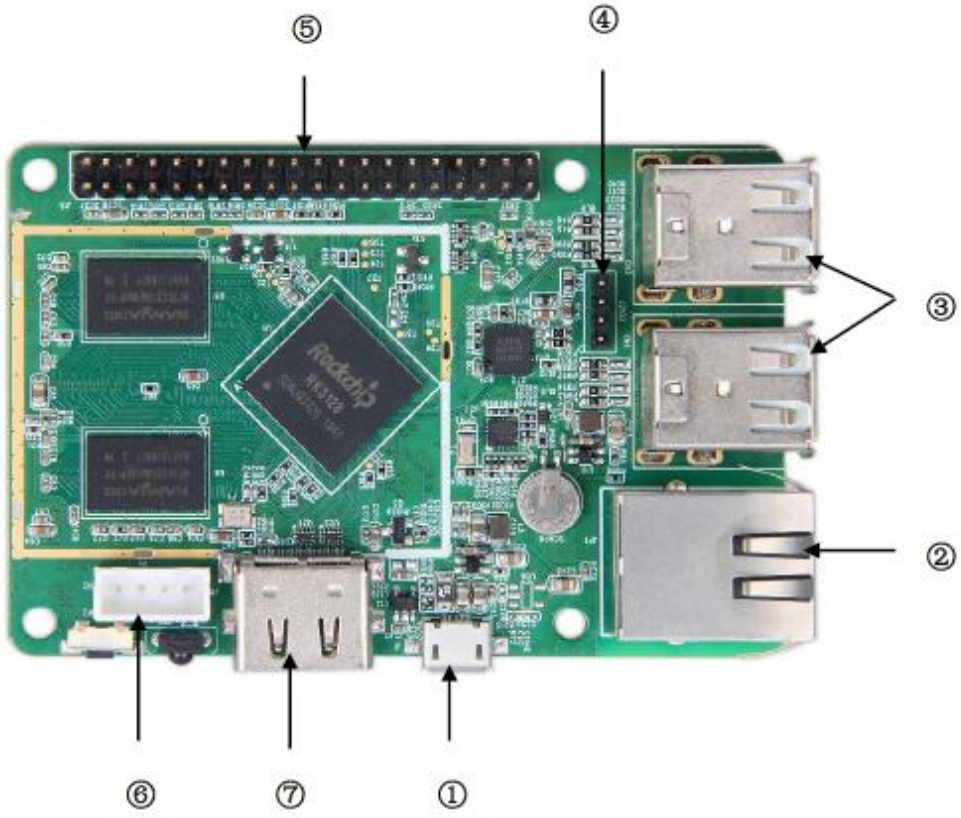
buildroot/board/rockchip/rk3308/fs-overlay/etc/input-event-daemon.conf

board/rockchip/rk3308/fs-overlay/usr/sbin/factory_reset_cfg

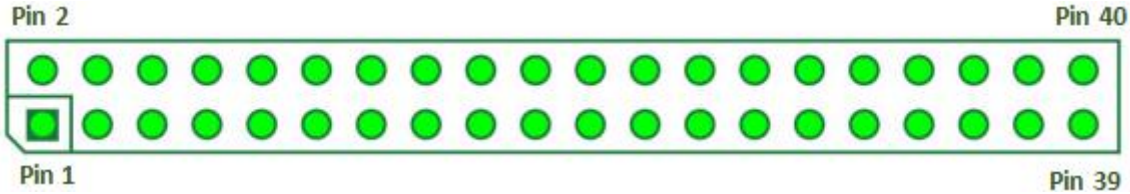
11.hardware information

11.1 Interface definition

11.1.1 Frontal perspective

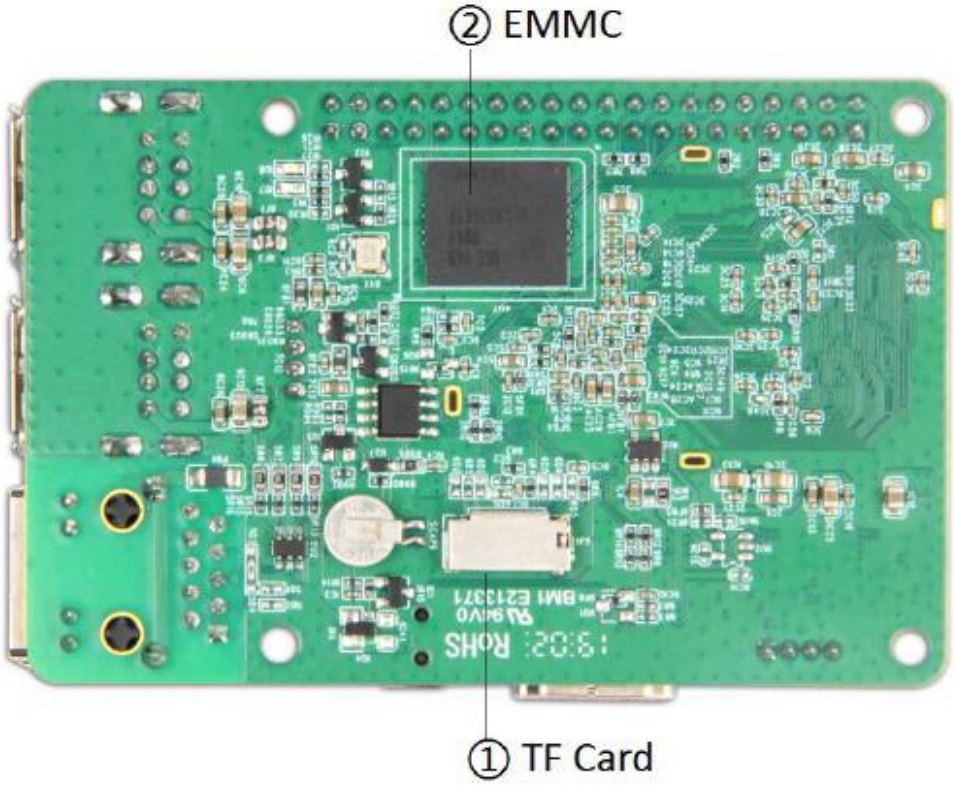


No.	Name	Description
1	DC IN	5V/2A
2	LAN	Ethernet 10/100M
3	USB HOST	USB HOST 2.0*4(support USB upgrade)
4	WIFI Jack	WIFI Jack*1
5	Extension GPIO	Extension GPIO
6	UART	UART*1
7	HDMI OUT	HDMI out*1



Extension GPIO			
PIN	**Default function**	**PIN**	**Default function**
1	VCC_IO_3V3	2	VCC_5V
3	I2C_SDA_A	4	VCC_5V
5	I2C_SCK_A	6	GND
7	GPIO3_C1	8	UART_TX_B
9	GND	10	UART_RX_B
11	UART_RX_A	12	PWM0
13	GPIO0_D6	14	GND
15	SPDIF_OUT	16	I2S_SDI
17	VCC_IO_3V3	18	I2S_SDO
19	PCM_OUT	20	GND
21	PCM_IN	22	I2S_IN
23	PCM_FS	24	PCM_CLK
25	GND	26	UART_TX_A
27	I2C_SDA_B	28	I2C_SCK_B
29	GPIO3_C7	30	GND
31	GPIO0_B4	32	IR
33	PWM1	34	GND
35	SYS_LED	36	I2S_MCLK
37	PWM2	38	I2S_SCLK
39	GND	40	I2S_LRCLK

11.1.2 Rear view



No.	Name	Description
1	TF Card Slot	TF card *1(max. 64GB)
2	EMMC	8G EMMC