Rockchip RK3588 Linux SDK Quick Start

ID: RK-JC-YF-915

Release Version: V1.5.0

Release Date: 2024-06-20

Security Level: □Top-Secret □Secret □Internal ■Public

DISCLAIMER

THIS DOCUMENT IS PROVIDED "AS IS". ROCKCHIP ELECTRONICS CO., LTD.("ROCKCHIP")DOES NOT PROVIDE ANY WARRANTY OF ANY KIND, EXPRESSED, IMPLIED OR OTHERWISE, WITH RESPECT TO THE ACCURACY, RELIABILITY, COMPLETENESS,MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE OR NON-INFRINGEMENT OF ANY REPRESENTATION, INFORMATION AND CONTENT IN THIS DOCUMENT. THIS DOCUMENT IS FOR REFERENCE ONLY. THIS DOCUMENT MAY BE UPDATED OR CHANGED WITHOUT ANY NOTICE AT ANY TIME DUE TO THE UPGRADES OF THE PRODUCT OR ANY OTHER REASONS.

Trademark Statement

"Rockchip", "瑞芯微", "瑞芯" shall be Rockchip's registered trademarks and owned by Rockchip. All the other trademarks or registered trademarks mentioned in this document shall be owned by their respective owners.

Rockchip Electronics Co., Ltd.

No.18 Building, A District, No.89, software Boulevard Fuzhou, Fujian,PRC

Website: www.rock-chips.com

Customer service Tel: +86-4007-700-590

Customer service Fax: +86-591-83951833

Customer service e-Mail: fae@rock-chips.com

Preface

Overview

This document primarily describes the basic usage of the RK3588 Linux SDK, aiming to help developers quickly understand and use the RK3588 SDK development package.

Target Audience

This document (this guide) is mainly suitable for the following engineers:

Technical Support Engineers

Software Development Engineers

Chip System Support Status

Chip Name	Uboot Version	Kernel Version	Debian Version	Buildroot Version
RK3588 Series	2017.9	5.10, 6.1	11, 12	2021.11, 2024.02

Revision History

Date	Version	Author	Modification Description
2022-01-15	V0.0.1	Caesar Wang	Initial version
2022-04-14	V0.1.0	Caesar Wang	Update to Beta version
2022-04-21	V0.1.1	Caesar Wang	Update Beta version to 0.1.1
2022-05-20	V1.0.0	Caesar Wang	Update to Release version
2022-06-20	V1.0.1	Caesar Wang	Update SDK to V1.0.1
2022-08-20	V1.0.2	Caesar Wang	Update SDK to V1.0.2
2022-09-20	V1.0.3	Caesar Wang	Update SDK to V1.0.3
2022-11-20	V1.0.4	Caesar Wang	Update flashing instructions
2023-04-20	V1.1.0	Caesar Wang	Update SDK to V1.1.0
2023-05-20	V1.1.1	Caesar Wang	Update SDK to V1.1.1
2023-06-20	V1.2.0	Caesar Wang	Update SDK to V1.2.0
2023-09-20	V1.3.0	Caesar Wang	Update SDK to V1.3.0
2023-12-20	V1.4.0	Caesar Wang	Update SDK to V1.4.0
2024-06-20	V1.5.0	Caesar Wang	Adapt to the new version SDK

Table of Contents

1. Precompiled SDK Image

Developers can bypass the process of compiling the entire operating system from source code by using the precompiled RK3588 Linux SDK image. This allows for a quick start to development and related assessments and comparisons, reducing the waste of development time and costs due to compilation issues.

The firmware can be downloaded from the public address Click here to download SDK firmware.

Firmware path: **General Linux SDK Firmware** -> **Linux5.10** -> **RK3588**

For those who need to modify the SDK code or require a quick start guide, please refer to the following sections.

2. Development Environment Setup

2.1 Preparing the Development Environment

We recommend using a system with Ubuntu 22.04 or a later version for compilation. Other Linux versions may require corresponding adjustments to the software packages. In addition to system requirements, there are other hardware and software requirements.

Hardware Requirements: 64-bit system, with hard disk space greater than 40GB. If you are performing multiple builds, you will need more hard disk space.

Software Requirements: System with Ubuntu 22.04 or a later version.

2.2 Installing Libraries and Toolkits

When developing devices using the command line, you can install the libraries and tools required for compiling the SDK through the following steps.

Use the following apt-get commands to install the libraries and tools needed for subsequent operations:

```
sudo apt-get update && sudo apt-get install git ssh make gcc libssl-dev liblz4-tool expect expect-dev g++ patchelf chrpath gawk texinfo chrpath diffstat binfmt-support qemu-user-static live-build bison flex fakeroot cmake gcc-multilib g++-multilib unzip device-tree-compiler ncurses-dev libgucharmap-2-90-dev bzip2 expat gpgv2 cpp-aarch64-linux-gnu libgmp-dev libmpc-dev bc python-is-python3 python2
```

> **Note:**
> The installation command is suitable for Ubuntu 22.04. For other versions, please use the corresponding installation command based on the package name. If you encounter errors during compilation, you can install the corresponding software package according to the error message. Among them:
>
> - Python requires the installation of version 3.6 or higher, with version 3.6 used as an example here.
> - Make requires the installation of version 4.0 or higher, with version 4.2 used as an example here.
> - LZ4 requires the installation of version 1.7.3 or higher.
> - Compiling Yocto requires a VPN network, and git does not have the CVE-2022-39253 security check patch.

2.2.1 Checking and Upgrading the Host's Python Version

The method for checking and upgrading the host's Python version is as follows:

- Check the host's Python version

```
$ python3 --version
Python 3.10.6
```

If the requirement of python>=3.6 is not met, it can be upgraded in the following way:

- Upgrade to the new version of `Python 3.6.15`

```
PYTHON3_VER=3.6.15
echo "wget
https://www.python.org/ftp/python/${PYTHON3_VER}/Python-${PYTHON3_VER}.tgz"
echo "tar xf Python-${PYTHON3_VER}.tgz"
echo "cd Python-${PYTHON3_VER}"
echo "sudo apt-get install libsqlite3-dev"
echo "./configure --enable-optimizations"
echo "sudo make install -j8"
```

2.2.2 Checking and Upgrading the Host's `make` Version

The method to check and upgrade the host's `make` version is as follows:

- Check the host's `make` version

```
$ make -v
GNU Make 4.2
Built for x86_64-pc-linux-gnu
```

- Upgrade to a newer version of `make 4.2`

```
$ sudo apt update && sudo apt install -y autoconf autopoint

git clone https://gitee.com/mirrors/make.git
cd make
git checkout 4.2
git am $BUILDROOT_DIR/package/make/*.patch
autoreconf -f -i
./configure
make make -j8
sudo install -m 0755 make /usr/bin/make
```

2.2.3 Checking and Upgrading the Host's `lz4` Version

The method to check and upgrade the host's `lz4` version is as follows:

- Check the host's `lz4` version

```
$ lz4 -v
*** LZ4 command line interface 64-bits v1.9.3, by Yann Collet ***
```

- Upgrade to a newer version of `lz4`

```
git clone https://gitee.com/mirrors/LZ4_old1.git
cd LZ4_old1

make
sudo make install
sudo install -m 0755 lz4 /usr/bin/lz4
```

2.2.4 Checking and Upgrading the Host's `git` Version

- Check the host's `git` version

```
$ git -v
git version 2.38.0
```

- Upgrade to a newer version of `git`

```
$ sudo apt update && sudo apt install -y libcurl4-gnutls-dev

git clone https://gitee.com/mirrors/git.git --depth 1 -b v2.38.0
cd git
make git -j8
make install
sudo install -m 0755 git /usr/bin/git
```

3. Docker Environment Setup

To assist developers in quickly completing the complex preparation of the development environment mentioned above, we have also provided a cross-compiler Docker image, enabling customers to rapidly verify and thereby reduce the time required to build the compilation environment.

Before using the Docker environment, you may refer to the following document for operations: `<SDK>/docs/en/Linux/Docker/Rockchip_Developer_Guide_Linux_Docker_Deploy_EN.pdf`.

The following systems have been verified:

Distribution Version	Docker Version	Image Loading	Firmware Compilation
Ubuntu 22.04	24.0.5	pass	pass
Ubuntu 21.10	20.10.12	pass	pass
Ubuntu 21.04	20.10.7	pass	pass
Ubuntu 18.04	20.10.7	pass	pass
Fedora 35	20.10.12	pass	NR (not run)

The Docker image can be obtained from the URL Docker Image.

4. Software Development Guide

4.1 Development Guide

To assist development engineers in quickly getting started and becoming proficient with SDK development and debugging, the "Rockchip_Developer_Guide_Linux_Software_CN.pdf" is released along with the SDK. It can be obtained in the `docs/cn/RK3588` directory and will be continuously improved and updated.

4.2 Chip Documentation

To assist development engineers in quickly becoming proficient with the development and debugging of RK3588 and RK3588S, the SDK release includes the chip manuals "Rockchip_RK3588_Datasheet_V1.7-20231117.pdf" and "Rockchip_RK3588S2_Datasheet_V1.0-20231101.pdf". These can be obtained in the `docs/cn/RK3588/Datasheet` directory and will be continuously updated and improved.

4.3 Buildroot Development Guide

To assist development engineers in quickly mastering and familiarizing themselves with the development and debugging of the Buildroot system, the SDK release includes the "Rockchip_Developer_Guide_Buildroot_CN.pdf" development guide, which can be obtained in the `docs/cn/Linux/System` directory and will be continuously improved and updated.

4.4 Debian Development Guide

To assist development engineers in quickly getting started with the development and debugging of the Debian system, the SDK release includes the "Rockchip_Developer_Guide_Debian_CN.pdf" development guide, which can be obtained under `docs/cn/Linux/System` and will be continuously improved and updated.

4.5 Third-Party OS Porting

To assist development engineers in quickly mastering the porting and adaptation of third-party operating systems, the SDK release includes the "Rockchip_Developer_Guide_Third_Party_System_Adaptation_CN.pdf" development guide, which can be obtained in the `docs/cn/Linux/System` directory and will be continuously improved and updated.

4.6 UEFI Development Guide

To assist development engineers in quickly becoming proficient with the RK3588 UEFI development and debugging, the SDK release includes the "Rockchip_Developer_Guide_UEFI_CN.pdf" and "Rockchip_Developer_Guide_Debian_ISO_Install_CN.pdf" development guides, which can be obtained under `docs/cn/RK3588/UEFI` and will be continuously improved and updated.

4.7 RKNPU Development Guide

The SDK provides RKNPU-related development tools, as detailed below:

RKNN-TOOLKIT2:

RKNN-Toolkit2 is a development kit for generating and evaluating RKNN models on a PC:

The development kit is located in the `external/rknn-toolkit2` directory, primarily used to implement a series of functions such as model conversion, optimization, quantization, inference, performance evaluation, and accuracy analysis.

Basic functions are as follows:

Function	Description
Model Conversion	Supports floating-point models of Pytorch / TensorFlow / TFLite / ONNX / Caffe / Darknet Supports quantization-aware models (QAT) of Pytorch / TensorFlow / TFLite Supports dynamic input models (dynamicization/native dynamic) Supports large models
Model Optimization	Constant folding / OP correction / OP Fuse&Convert / Weight sparsification / Model pruning
Model Quantization	Supports quantization types: asymmetric i8/fp16 Supports Layer / Channel quantization methods; Normal / KL/ MMSE quantization algorithms Supports mixed quantization to balance performance and accuracy
Model Inference	Supports model inference on a PC via a simulator Supports model inference on the NPU hardware platform (board-level inference) Supports batch inference, supports multi-input models
Model Evaluation	Supports performance and memory evaluation of models on the NPU hardware platform
Accuracy Analysis	Supports quantization accuracy analysis (simulator/NPU)
Additional Features	Supports version/device query functions, etc.

For specific usage instructions, please refer to the documentation in the current `doc/` directory:

```
├── 01_Rockchip_RKNPU_Quick_Start_RKNN_SDK_V2.0.0beta0_CN.pdf
├── 01_Rockchip_RKNPU_Quick_Start_RKNN_SDK_V2.0.0beta0_EN.pdf
...
├── RKNNToolKit2_API_Difference_With_Toolkit1-V2.0.0beta0.md
└── RKNNToolKit2_OP_Support-v2.0.0-beta0.md
```

RKNN API:

The development instructions for RKNN API are located in the project directory `external/rknpu2`, used for inferring RKNN models generated by RKNN-Toolkit2.
For specific usage instructions, please refer to the documentation in the current `doc/` directory:

```
...
├── 02_Rockchip_RKNPU_User_Guide_RKNN_SDK_V2.0.0beta0_CN.pdf
├── 02_Rockchip_RKNPU_User_Guide_RKNN_SDK_V2.0.0beta0_EN.pdf
├── 03_Rockchip_RKNPU_API_Reference_RKNN_Toolkit2_V2.0.0beta0_CN.pdf
├── 03_Rockchip_RKNPU_API_Reference_RKNN_Toolkit2_V2.0.0beta0_EN.pdf
├── 04_Rockchip_RKNPU_API_Reference_RKNNRT_V2.0.0beta0_CN.pdf
└── 04_Rockchip_RKNPU_API_Reference_RKNNRT_V2.0.0beta0_EN.pdf
```

4.8 DPDK Development Guide

To assist development engineers in quickly becoming proficient with the development and debugging of RK3588 DPDK, the "Rockchip_Developer_Guide_Linux_DPDK_CN.pdf" and "Rockchip_Developer_Guide_Linux_GMAC_DPDK_CN.pdf" development guides are released with the SDK. They can be obtained in the directory `<SDK>/external/dpdk/rk_docs` and will be continuously improved and updated.

4.9 Real-Time Linux Development Guide

As the demand for real-time performance in products increases, the real-time capabilities of standard Linux are no longer sufficient for many products. It is necessary to optimize standard Linux to enhance its real-time performance, such as through the use of PREEMPT_RT and the Xenomai real-time system.

For more details, please refer to the development patch package and instructions located at `docs/Patches/Real-Time-Performance/`.

4.10 Software Update Log

- Software release version upgrades can be viewed through the engineering XML. The specific method is as follows:

```
.repo/manifests$ realpath rk3588_linux_release.xml
# For example: the printed version number is v1.5.0, and the update time is
20240620
# <SDK>/.repo/manifests/rk3588_linux/rk3588_linux_release_v1.5.0_20240620.xml
```

- The content of the software release version upgrade can be viewed through the engineering text, refer to the engineering directory:

```
<SDK>/.repo/manifests/rk3588_linux/RK3588_Linux5.10_SDK_Note.md
or
<SDK>/docs/en/RK3588/RK3588_Linux5.10_SDK_Note.md
```

- The board can obtain version information in the following way

```
buildroot:/# cat /etc/os-release
NAME=Buildroot
VERSION=linux-5.10-gen-rkr8
ID=buildroot
VERSION_ID=2021.11
PRETTY_NAME="Buildroot 2021.11"
OS="buildroot"
BUILD_INFO="xxx  Mon Dec 18 23:12:04 CST 2023 - rockchip_rk3588"
KERNEL="5.10 - rockchip_linux_defconfig"
```

5. Hardware Development Guide

Hardware-related development can refer to the user guide in the project directory:

Rockchip RK3588 Hardware Design and User Guide:

```
<SDK>/docs/en/RK3588/Hardware/
├── RK3588_PinOut_V1.1_20220922.xlsx
├── Rockchip_RK3588M_EVB_User_Guide_V1.0_EN.pdf
├── Rockchip_RK3588S_EVB_User_Guide_V1.1_EN.pdf
├── Rockchip_RK3588S_Hardware_Design_Guide_V1.2_EN.pdf
├── Rockchip_RK3588_EVB1_User_Guide_V1.2_EN.pdf
├── Rockchip_RK3588_EVB7_User_Guide_V1.0_EN.pdf
└── Rockchip_RK3588_Hardware_Design_Guide_V1.4_EN.pdf
```

6. SDK Configuration Framework Description

6.1 Introduction to SDK Project Directory

The SDK project directory includes directories such as buildroot, debian, rtos, app, kernel, u-boot, device, docs, external, etc. It uses a manifest to manage the repository and the repo tool to manage each directory or its corresponding git project.

- app: Contains upper-layer application APPs, mainly some application demos.
- buildroot: Root file system developed based on Buildroot (2021 or 2024).
- debian: Root file system developed based on Debian bullseye (11 or 12).
- device/rockchip: Contains chip-level board configurations as well as scripts and files for compiling and packaging firmware.
- docs: Contains development guidance documents, platform support lists, tool usage documents, Linux development guides, etc.
- external: Contains third-party related repositories, including display, audio and video, camera, network, security, etc.
- kernel: Contains the code for kernel development.
- hal: Contains bare-metal development libraries based on RK HAL hardware abstraction layer, used for AMPAK solutions.
- output: Contains firmware information, compilation information, XML, host environment, etc., generated each time.
- prebuilts: Contains cross-compilation toolchains.
- rkbin: Contains Rockchip-related binaries and tools.
- rockdev: Contains compiled output firmware, actually a soft link to `output/firmware`.
- rtos: Root file system developed based on RT-Thread 4.1.
- tools: Contains commonly used tools under Linux and Windows operating systems.
- u-boot: Contains U-Boot code developed based on the v2017.09 version.
- uefi: Contains UEFI code developed based on the edk2 V2.7 version.
- yocto: Root file system developed based on Yocto 4.0 or 5.0.

7. Introduction to Cross-Compilation Toolchain for SDK

Given that the Rockchip Linux SDK is currently only compiled in a Linux PC environment, we have provided the cross-compilation toolchain for Linux only. The prebuilt toolchain in the prebuilt directory is intended for use with U-Boot and the Kernel. The specific Rootfs should be compiled using the corresponding toolchain for each or a third-party toolchain.

7.1 U-Boot and Kernel Compilation Toolchain

The SDK prebuilts directory contains cross-compilers for U-Boot and Kernel compilation, as listed below:

Directory	Description
prebuilts/gcc/linux-x86/aarch64/gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu	64-bit toolchain for gcc arm 10.3.1
prebuilts/gcc/linux-x86/arm/gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabihf	32-bit toolchain for gcc arm 10.3.1

You can download the toolchain from the following address:

Click here

7.2 Buildroot Toolchain

7.2.1 Setting Up the Compilation Environment

To compile individual modules or third-party applications, a cross-compilation environment needs to be configured. For instance, the cross-compilation tools for RK3588 are located in the `buildroot/output/rockchip_rk3588/host/usr` directory. It is necessary to set the `bin/` directory of the tools and the `aarch64-buildroot-linux-gnu/bin/` directory as environment variables. Execute the script for automatic configuration of environment variables in the top-level directory:

```
source buildroot/envsetup.sh rockchip_rk3588
```

Enter the command to check:

```
cd buildroot/output/rockchip_rk3588/host/usr/bin
./aarch64-linux-gcc --version
```

At this point, the following information will be printed:

```
aarch64-linux-gcc.br_real (Buildroot -gc307c95550) 12.3.0
```

7.2.2 Packaging Toolchain

Buildroot supports the packaging of its built-in toolchain into a compressed archive for third-party applications to compile independently. For detailed information on how to package the toolchain, please refer to the Buildroot official documentation:

```
buildroot/docs/manual/using-buildroot-toolchain.txt
```

In the SDK, you can directly run the following command to generate the toolchain package:

```
./build.sh bmake:sdk
```

The generated toolchain package is located in the `buildroot/output/*/images/` directory, named `aarch64-buildroot-linux-gnu_sdk-buildroot.tar.gz`, for users who need it. After extraction, the path to `gcc` will be:

```
./aarch64-buildroot-linux-gnu_sdk-buildroot/bin/aarch64-buildroot-linux-gnu-gcc
```

7.3 Debian Toolchain

Utilize Docker on the host machine, GCC, or `dpkg-buildpackage` for related compilation processes.

7.4 Yocto Toolchain

Refer to the following:

[Building your own recipes from first principles]

[New Recipe]

8. SDK Compilation Instructions

The SDK can be configured and compiled for specific functionalities using `make` or `./build.sh` with target parameters.
Refer to the compilation instructions in `device/rockchip/common/README.md`.

8.1 Viewing SDK Compilation Commands

`make help`, for example:

```
$ make help
  menuconfig             - interactive curses-based configurator
  oldconfig              - resolve any unresolved symbols in .config
  syncconfig             - Same as oldconfig, but quietly, additionally update
deps
  olddefconfig           - Same as syncconfig but sets new symbols to their
default value
  savedefconfig          - Save current config to RK_DEFCONFIG (minimal config)
  ...
```

The actual execution of make is `./build.sh`

That is, you can also run `./build.sh <target>` to compile related features, and you can view the specific compilation commands through `./build.sh help`.

```
############### Rockchip Linux SDK ###############

Manifest: rk3588_linux_release_v1.5.0_20240620.xml

Log colors: message notice warning error fatal

Usage: build.sh [OPTIONS]
Available options:
chip[:<chip>[:<config>]]          choose chip
defconfig[:<config>]              choose defconfig
config                            modify SDK defconfig
...
updateimg                         build update image
otapackage                        build A/B OTA update image
all                               build images
release                           release images and build info
save                              alias of release
all-release                       build and release images
allsave                           alias of all-release
shell                             setup a shell for developing
cleanall                          cleanup
clean[:module[:module]]...        cleanup modules
post-rootfs <rootfs dir>          trigger post-rootfs hook scripts
help                              usage

Default option is 'all'.
```

8.2 SDK Board-Level Configuration

Navigate to the project directory `<SDK>/device/rockchip/rk3588`:

Board-Level Configuration	Description
rockchip_rk3588_evb1_lp4_v10_defconfig	Suitable for RK3588 EVB1 development board with LPDDR4
rockchip_rk3588_evb7_v11_defconfig	Suitable for RK3588 EVB7 development board with LPDDR4
rockchip_rk3588s_evb1_lp4x_v10_defconfig	Suitable for RK3588S EVB1 development board with LPDDR4
rockchip_defconfig	Default configuration, which will be symbolically linked to a specific board-level configuration

Method 1
Add the board-level configuration file after `./build.sh`, for example:

Select the board-level configuration **suitable for RK3588 EVB1 with LPDDR4 development board**:

```
./build.sh device/rockchip/rk3588/rockchip_rk3588_evb1_lp4_v10_defconfig
```

Select the board-level configuration **suitable for RK3588 EVB7 with single PMIC development board**:

```
./build.sh device/rockchip/rk3588/rockchip_rk3588_evb7_v11_defconfig
```

Select the board-level configuration **suitable for RK3588S EVB1 with LPDDR4 development board**:

```
./build.sh device/rockchip/rk3588/rockchip_rk3588s_evb1_lp4x_v10_defconfig
```

Method 2

```
rk3588$ ./build.sh lunch
Pick a defconfig:

1. rockchip_defconfig
2. rockchip_rk3588_evb1_lp4_v10_defconfig
3. rockchip_rk3588_evb7_v11_defconfig
4. rockchip_rk3588s_evb1_lp4x_v10_defconfig
Which would you like? [1]:
```

> Note:
> Before April 2023, the default configuration for the RK3588 EVB obtained from Rockchip was EVB1.
> After April 2023, the default configuration for the RK3588 EVB obtained from Rockchip was EVB7.

8.3 SDK Customization Configuration

The SDK can be configured through `make menuconfig`, and the main configurable components are as follows:

```
(rockchip_rk3588_evb1_lp4_v10_defconfig) Name of defconfig to save
[*] Rootfs (Buildroot|Yocto|Debian)  --->
[*] Loader (U-Boot)  --->
[ ] AMPAK (Asymmetric Multi-Processing System)
[*] Kernel (Embedded in an Android-style boot image)  --->
    Boot (Android-style boot image)  --->
[*] Recovery (based on Buildroot)  --->
    *** Security feature depends on buildroot rootfs ***
    Extra partitions (oem, userdata, etc.)  --->
    Firmware (partition table, misc image, etc.)  --->
[*] Update (Rockchip update image)  --->
    Others configurations  --->
```

- Rootfs: The Rootfs here represents the "root file system," where you can choose different root file system configurations such as Buildroot, Yocto, Debian, etc.
- Loader (U-Boot): This is the configuration for the bootloader, typically U-Boot, which is used to initialize the hardware and load the main operating system.
- AMPAK: A multi-core heterogeneous boot solution suitable for application scenarios requiring real-time performance.
- Kernel: Here, kernel options are configured to customize the Linux kernel to fit your hardware and application needs.
- Boot: Configure the supported format for the Boot partition here.
- Recovery (based on Buildroot): This is the configuration for the recovery environment based on Buildroot, used for system recovery and upgrades.

- PCBA test (based on Buildroot): This is the configuration for a PCBA (Printed Circuit Board Assembly) test environment based on Buildroot.
- Security: Enable security features, including Secureboot methods, Optee storage methods, and burn-in keys.
- Extra partitions: Used to configure additional partitions.
- Firmware: Configure firmware-related options here.
- Update (Rockchip update image): Used to configure options for Rockchip complete firmware.
- Others configurations: Additional configuration options.

The `make menuconfig` configuration interface provides a text-based user interface to select and configure various options. After the configuration is complete, use the `make savedefconfig` command to save these settings, so that the customized compilation will proceed according to these settings.

Through the above config, you can choose different Rootfs/Loader/Kernel configurations for various customized compilations, allowing you to flexibly select and configure system components to meet specific needs.

8.4 Fully Automated Compilation

To ensure that each update of the Software Development Kit (SDK) proceeds smoothly, it is recommended to clean up the previous compilation artifacts before updating. This practice helps avoid potential compatibility issues or compilation errors, as old compilation artifacts may not be suitable for the new version of the SDK. To clean these artifacts, you can simply run the command `./build.sh cleanall`.

Navigate to the project's root directory and execute the following command to automatically complete all compilations:

```
./build.sh all # Only compiles module code (u-Boot, kernel, Rootfs, Recovery)
               # Further execution of `./build.sh ./mkfirmware.sh` is required
for firmware packaging

./build.sh     # Compiles module code (u-Boot, kernel, Rootfs, Recovery)
               # Packages into a complete update.img upgrade package
               # All compilation information is copied and generated in the out
directory
```

The default is Buildroot, but you can specify a different rootfs by setting the environment variable `RK_ROOTFS_SYSTEM`. `RK_ROOTFS_SYSTEM` currently supports three systems: buildroot, debian, and yocto.

For example, to generate a debian system, you can use the following commands:

```
export RK_ROOTFS_SYSTEM=debian
./build.sh
or
RK_ROOTFS_SYSTEM=debian ./build.sh
```

8.5 Module Compilation

8.5.1 U-Boot Compilation

```
./build.sh uboot
```

8.5.2 Kernel Compilation

- Method One

```
./build.sh kernel
```

- Method Two

```
cd kernel
export CROSS_COMPILE=../prebuilts/gcc/linux-x86/aarch64/gcc-arm-10.3-2021.07-
x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu-
make ARCH=arm64 rockchip_linux_defconfig rk3588_linux.config
make ARCH=arm64 rk3588-evb1-lp4-v10-linux.img -j
or
make ARCH=arm64 rk3588-evb7-v11-linux.img -j
```

- Method Three

```
cd kernel
export CROSS_COMPILE=aarch64-linux-gnu-
make ARCH=arm64 rockchip_linux_defconfig rk3588_linux.config
make ARCH=arm64 rk3588-evb1-lp4-v10-linux.img -j
or
make ARCH=arm64 rk3588-evb7-v11-linux.img -j
```

8.5.3 Recovery Compilation

Navigate to the root directory of the project and execute the following command to automatically complete the compilation and packaging of Recovery.

```
<SDK>#./build.sh recovery
```

After compilation, the `recovery.img` is generated in the Buildroot directory `output/rockchip_rk3588_recovery/images`. Note: `recovery.img` includes the kernel, so every time there is a Kernel change, Recovery needs to be repackaged and generated. The method to repackage Recovery is as follows:

```
<SDK>#source buildroot/envsetup.sh
<SDK>#cd buildroot
<SDK>#make recovery-reconfigure
<SDK>#cd -
<SDK>#./build.sh recovery
```

> Note: Recovery is a non-essential feature, and some board-level configurations may not set it up.

8.5.4 Buildroot Compilation

Navigate to the root directory of the project and execute the following command to automatically complete the compilation and packaging of the Rootfs:

```
./build.sh rootfs
```

After compilation, different format images are generated in the `output/rockchip_rk3588/images` directory under the Buildroot, with the default format being rootfs.ext4.

8.5.5 Debian Compilation

```
./build.sh debian
```

After compilation, a `linaro-rootfs.img` file is generated in the `debian` directory.

> **Note: It is necessary to install the relevant dependency packages beforehand.**
> sudo apt-get install binfmt-support qemu-user-static live-build
> sudo dpkg -i ubuntu-build-service/packages/*
> sudo apt-get install -f

For more details, please refer to the Debian development documentation:

`<SDK>/docs/en/Linux/ApplicationNote/Rockchip_Developer_Guide_Debian_EN.pdf`

8.5.6 Yocto Compilation

Navigate to the root directory of the project and execute the following command to automatically complete the compilation and packaging of Rootfs:

```
./build.sh yocto
```

After the compilation, the rootfs.img is generated in the yocto directory under build/latest.

The default username for login is root. For more information on Yocto, please refer to the Rockchip Wiki.

FAQ:

- If you encounter the following issue during the compilation:

```
Please use a locale setting which supports UTF-8 (such as LANG=en_US.UTF-8).
Python can't change the filesystem locale after loading so we need a UTF-8
when Python starts or things won't work.
```

Solution:

```
locale-gen en_US.UTF-8
export LANG=en_US.UTF-8 LANGUAGE=en_US.en LC_ALL=en_US.UTF-8
```

Or refer to setup-locale-python3.

8.5.7 Firmware Packaging

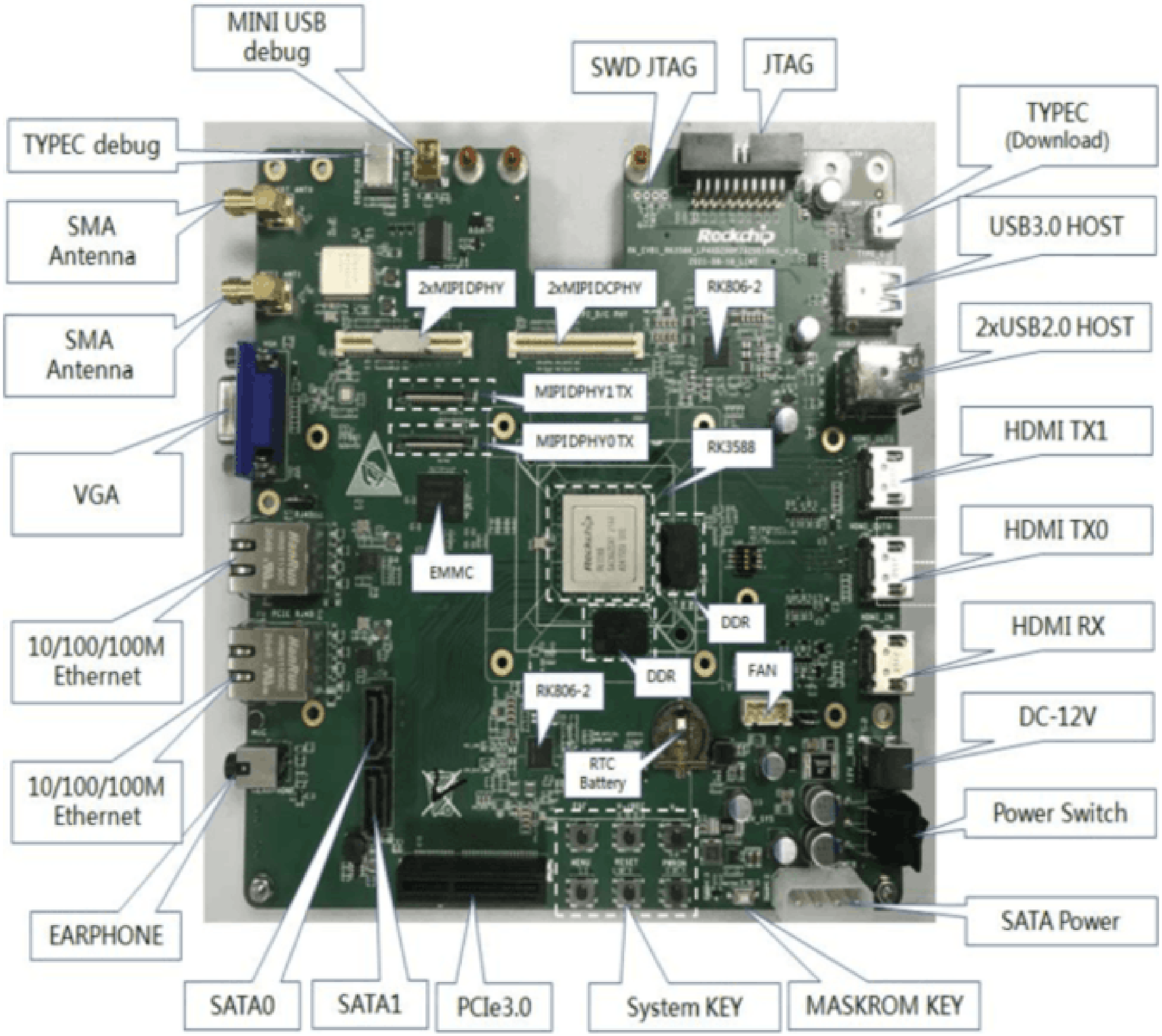
After compiling the various parts such as Kernel, U-Boot, Recovery, and Rootfs, navigate to the root directory of the project and execute the following command to automatically complete the packaging of all firmware into the `output/firmware` directory:

Firmware Generation:

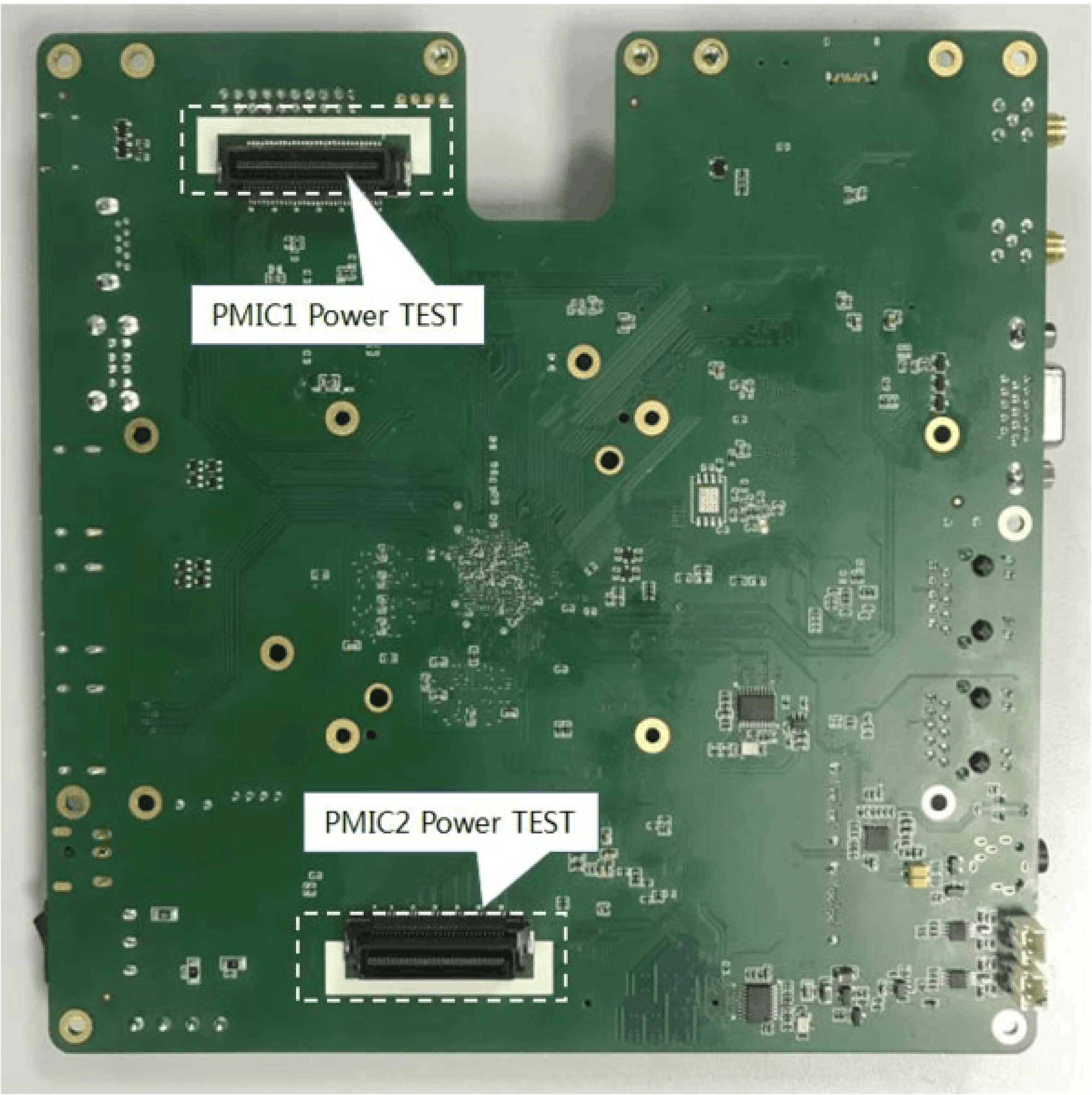
```
./build.sh firmware
```

9. Flashing Instructions

The front interface layout of the RK3588 EVB development board is as follows:



The back interface layout of the RK3588 EVB1 development board is as follows:

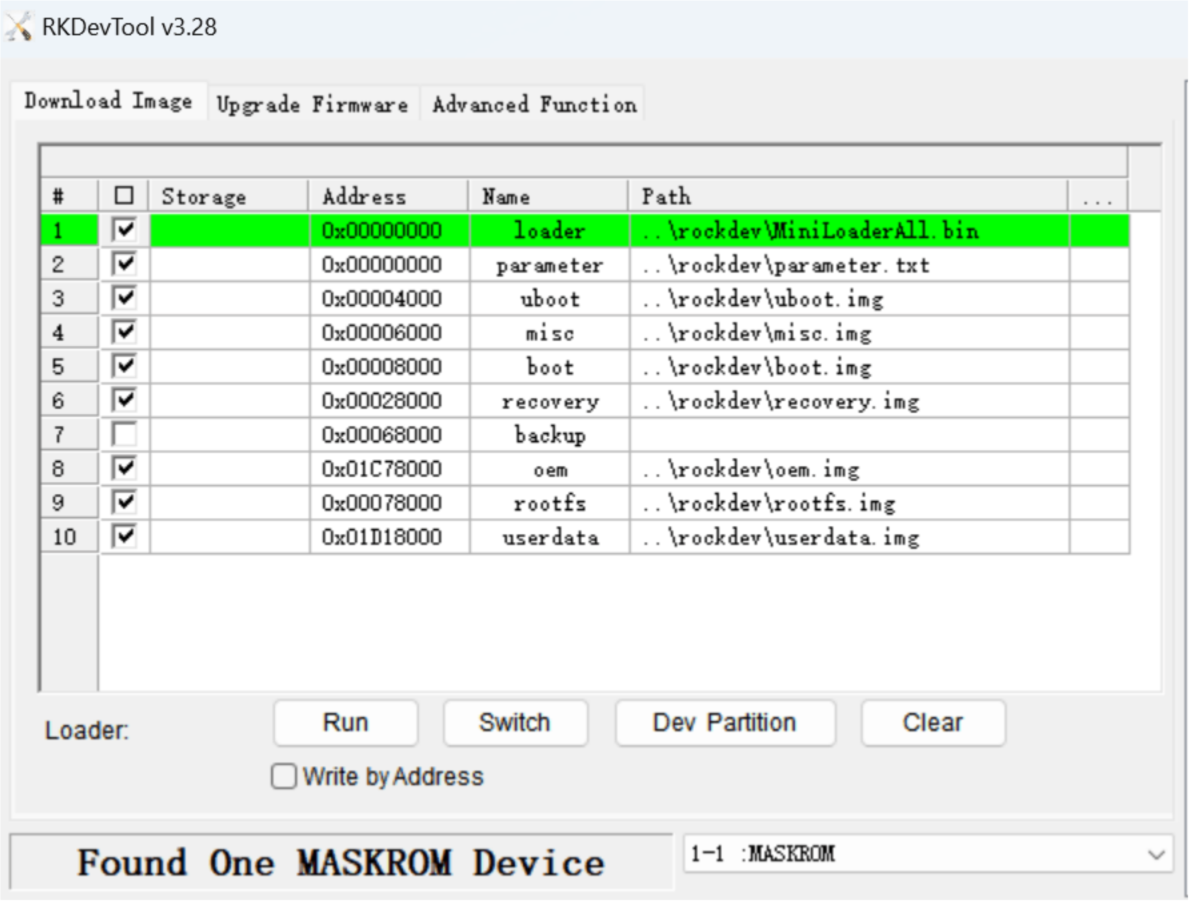


9.1 Windows Flashing Guide

The SDK provides a Windows flashing tool (the tool version must be V3.28 or above), which is located in the root directory of the project:

```
tools/
├── windows/RKDevTool
```

As shown in the figure below, after compiling the corresponding firmware, the device needs to enter the MASKROM or BootROM flashing mode. Connect the USB download cable, hold the "MASKROM" button and press the reset button "RST" without releasing it, then release to enter the MASKROM mode. Load the path of the compiled firmware and click "Execute" to perform the flashing. Alternatively, hold the "recovery" button and press the reset button "RST" without releasing to enter the loader mode for flashing. Below are the partition offsets and flashing files for the MASKROM mode.
(Note: The Windows PC needs to run the tool with administrator privileges to execute)



#	☐	Storage	Address	Name	Path	...
1	☑		0x00000000	loader	..\rockdev\MiniLoaderAll.bin	
2	☑		0x00000000	parameter	..\rockdev\parameter.txt	
3	☑		0x00004000	uboot	..\rockdev\uboot.img	
4	☑		0x00006000	misc	..\rockdev\misc.img	
5	☑		0x00008000	boot	..\rockdev\boot.img	
6	☑		0x00028000	recovery	..\rockdev\recovery.img	
7	☐		0x00068000	backup		
8	☑		0x01C78000	oem	..\rockdev\oem.img	
9	☑		0x00078000	rootfs	..\rockdev\rootfs.img	
10	☑		0x01D18000	userdata	..\rockdev\userdata.img	

Found One MASKROM Device

1-1 :MASKROM

Note: Before flashing, it is necessary to install the latest USB drivers. For more details on the drivers, see:

```
<SDK>/tools/windows/DriverAssitant_v5.13.zip
```

9.2 Linux Flashing Instructions

The flashing tool for Linux is located in the `tools/linux` directory (the Linux_Upgrade_Tool version must be V2.36 or higher). Please ensure that your board is connected to MASKROM/loader rockusb. For example, if the firmware compiled is in the `rockdev` directory, the upgrade commands are as follows:

```
sudo ./upgrade_tool ul rockdev/MiniLoaderAll.bin -noreset
sudo ./upgrade_tool di -p rockdev/parameter.txt
sudo ./upgrade_tool di -u rockdev/uboot.img
sudo ./upgrade_tool di -misc rockdev/misc.img
sudo ./upgrade_tool di -b rockdev/boot.img
sudo ./upgrade_tool di -recovery rockdev/recovery.img
sudo ./upgrade_tool di -oem rockdev/oem.img
sudo ./upgrade_tool di -rootfs rocdev/rootfs.img
sudo ./upgrade_tool di -userdata rockdev/userdata.img
sudo ./upgrade_tool rd
```

Or upgrade with the complete firmware package:

```
sudo ./upgrade_tool uf rockdev/update.img
```

Or in the root directory, when the machine is running in MASKROM status, run the following upgrade:

```
./rkflash.sh
```

9.3 System Partition Description

Default Partition Description (Below is the RK3588 EVB Partition Reference)

Number	Start (sector)	End (sector)	Size	Name
1	8389kB	12.6MB	4194kB	uboot
2	12.6MB	16.8MB	4194kB	misc
3	16.8MB	83.9MB	67.1MB	boot
4	83.9MB	218MB	134MB	recovery
5	218MB	252MB	33.6MB	backup
6	252MB	15.3GB	15.0GB	rootfs
7	15.3GB	15.4GB	134MB	oem
8	15.6GB	31.3GB	15.6GB	userdata

- The uboot partition: For the uboot.img compiled from uboot.
- The misc partition: For misc.img, used by recovery.
- The boot partition: For the boot.img compiled from the kernel.
- The recovery partition: For the recovery.img compiled from recovery.
- The backup partition: Reserved, not in use at the moment.
- The rootfs partition: For the rootfs.img compiled from buildroot, debian, or yocto.
- The oem partition: For the manufacturer's use, storing the manufacturer's APP or data. Mounted in the /oem directory.
- The userdata partition: For APPs to temporarily generate files or for the end user, mounted in the /userdata directory.