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

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 Images

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

The SDK firmware can be downloaded from the public address SDK Firmware Download Here.

Linux 5.10 firmware path: **Universal Linux SDK Firmware** -> **Linux 5.10** -> **RK3588**
Linux 6.1 firmware path: **Universal Linux SDK Firmware** -> **Linux 6.1** -> **RK3588**

For those who need to modify the SDK code or get started quickly, please refer to the following sections.

2. Setting Up the Development Environment

2.1 Preparing the Development Environment

We recommend using a system with Ubuntu 22.04 or a higher 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: A system with Ubuntu 22.04 or a higher version.

2.2 Installing Libraries and Toolkits

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

Use the following apt-get command to install the libraries and tools required 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 applicable to Ubuntu 22.04. For other versions, please use the corresponding installation commands based on the package names. If you encounter errors during compilation, you can install the corresponding software packages based on the error messages. Specifically:
>
> - 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.

2.2.1 Checking and Upgrading the Host's Python Version

The method to check and upgrade 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, you can upgrade by the following method:

- Upgrade to the new version `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:

- Checking the Host's `make` Version

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

- Upgrading 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 new 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
```

3. Setting Up Docker Environment

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

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

The verified systems are as follows:

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 website Docker Image.

4. Software Development Guide

4.1 Development Guide

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

4.2 Chip Documentation

To assist development engineers in quickly mastering and familiarizing themselves with the development and debugging of RK3588 and RK3588S, the chip manuals "Rockchip_RK3588_Datasheet_V1.7-20231117.pdf" and "Rockchip_RK3588S2_Datasheet_V1.0-20231101.pdf" are released along with the SDK. They can be obtained in the `docs/en/RK3588/Datasheet` directory and will be continuously improved and updated.

4.3 Buildroot Development Guide

To assist development engineers in quickly becoming proficient with the development and debugging of the Buildroot system, the SDK release includes the "Rockchip_Developer_Guide_Buildroot_EN.pdf" development guide, which can be obtained in the `docs/en/Linux/System` directory and will be continuously improved and updated.

4.4 Debian Development Guide

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

4.5 Third-Party OS Porting

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

4.6 UEFI Development Guide

To assist development engineers in quickly getting started with and becoming familiar with the development and debugging of RK3588 UEFI, the SDK release includes the "Rockchip_Developer_Guide_UEFI_EN.pdf" and "Rockchip_Developer_Guide_Debian_ISO_Install_EN.pdf" development guides, which can be obtained under `docs/en/RK3588/UEFI` and will be continuously improved and updated.

4.7 RKNPU Development Guide

The SDK provides RKNPU-related development tools, as follows:

RKNN-TOOLKIT2:

RKNN-Toolkit2 is a development kit for generating and evaluating RKNN models on the 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 features are as follows:

Feature	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	Supported 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 the PC through the 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 function (simulator / NPU)
Additional Features	Supports version/device query functions, etc.

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

```
├── 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 `external/rknpu2` project directory, used for inferring RKNN models generated by RKNN-Toolkit2.
For specific usage instructions, please refer to the documents in the current `doc/` directory:

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

4.8 DPDK Development Guide

To assist development engineers in quickly getting started with the development and debugging of RK3588 DPDK, the SDK release includes the "Rockchip_Developer_Guide_Linux_DPDK_EN.pdf" and "Rockchip_Developer_Guide_Linux_GMAC_DPDK_EN.pdf" development guides, which can be obtained in the `<SDK>/external/dpdk/rk_docs` directory 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 real-time performance, such as using PREEMPT_RT and the Xenomai real-time system.

Below are the latency test results under stress tests based on the RK3588 Buildroot with both PREEMPT_RT and Xenomai, as follows:

Stress Test:

```
stress-ng -c 8 --io 2 --vm 1 --vm-bytes 1024M --timeout 1000000s
```

4.9.1 PREEMPT_RT Patch

```
root@rk3588:/# cyclictest -c 0 -m -t 8 -p 99
# /dev/cpu_dma_latency set to 0us

policy: fifo: loadavg: 15.20 15.28 15.29 11/345 10678
T: 0 ( 6379) P:99 I:1000 C:6473077 Min:      3 Act:   13 Avg:    7 Max:      28
T: 1 ( 6380) P:99 I:1500 C:4315388 Min:      4 Act:    7 Avg:    7 Max:      24
T: 2 ( 6381) P:99 I:2000 C:3236536 Min:      3 Act:    4 Avg:    6 Max:      24
T: 3 ( 6382) P:99 I:2500 C:2589231 Min:      4 Act:    8 Avg:    7 Max:      30
T: 4 ( 6383) P:99 I:3000 C:2157683 Min:      1 Act:    1 Avg:    2 Max:      16
T: 5 ( 6384) P:99 I:3500 C:1849443 Min:      1 Act:    2 Avg:    2 Max:      15
T: 6 ( 6385) P:99 I:4000 C:1618261 Min:      1 Act:    2 Avg:    2 Max:      16
T: 7 ( 6386) P:99 I:4500 C:1438455 Min:      1 Act:    2 Avg:    2 Max:      15
(2-hour test)
```

4.9.2 Xenomai Cobalt Mode

```
root@rk3588:/# cyclictest -c 0 -m -n -t 8 -p 99
# /dev/cpu_dma_latency set to 0us

policy: fifo: loadavg: 12.82 12.81 12.81 11/225 1490
T: 0 ( 1179) P:99 I:1000 C:12797171 Min:      1 Act:    4 Avg:    4 Max:      19
T: 1 ( 1180) P:99 I:1500 C:8531447 Min:      1 Act:    4 Avg:    3 Max:      16
T: 2 ( 1181) P:99 I:2000 C:6398585 Min:      1 Act:    4 Avg:    4 Max:      25
T: 3 ( 1182) P:99 I:2500 C:5118868 Min:      1 Act:    3 Avg:    5 Max:      28
T: 4 ( 1183) P:99 I:3000 C:4265723 Min:      1 Act:    5 Avg:    4 Max:      17
T: 5 ( 1184) P:99 I:3500 C:3656334 Min:      1 Act:    4 Avg:    5 Max:      18
T: 6 ( 1185) P:99 I:4000 C:3199292 Min:      2 Act:    4 Avg:    5 Max:      17
T: 7 ( 1186) P:99 I:4500 C:2843815 Min:      1 Act:    5 Avg:    5 Max:      19
(Testing for 4 hours)
```

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

4.10 Software Update Log

- The software release version upgrade can be viewed through the engineering XML file, with the specific method as follows:

```
.repo/manifests$ realpath rk3588_linux6.1_release.xml
# For example: The printed version number is v1.1.0, and the update time is
20240620
# <SDK>/.repo/manifests/release/rk3588_linux6.1_release_v1.0.0_20240620.xml
```

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

```
<SDK>/.repo/manifests/RK3588_Linux6.1_SDK_Note.md
or
<SDK>/docs/en/RK3588/RK3588_Linux6.1_SDK_Note.md
```

- The board can obtain version information in the following way

```
buildroot:/# cat /etc/os-release
NAME=Buildroot
VERSION=linux-6.1-stan-rkr3
ID=buildroot
VERSION_ID=2024.02
PRETTY_NAME="Buildroot 2024.02"
OS="buildroot"
RK_BUILD_INFO="XXX Thu Jun 20 10:51:40 CST 2024 - rockchip_rk3588"
```

5. Hardware Development Guide

For hardware-related development, refer to the user guide in the project directory:

RK3588 Hardware Design and User Guide:

```
<SDK>/docs/en/RK3588/Hardware/
├── Rockchip_RK3588S_EVB_User_Guide_V1.1_EN.pdf
├── Rockchip_RK3588S_Hardware_Design_Guide_V1.0_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 SDK Project Directory Overview

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 projects.

- app: Contains upper-layer application APPs, mainly some application demos.
- buildroot: Root file system based on Buildroot (2021 or 2024) development.
- debian: Root file system based on Debian bullseye (11 or 12) development.
- 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 based on RT-Thread 4.1 development.
- tools: Contains commonly used tools under Linux and Windows operating systems.
- u-boot: Contains U-Boot code based on version v2017.09 development.
- uefi: Contains UEFI code based on edk2 V2.7 version development.
- yocto: Contains root file system based on Yocto 4.0 or 5.0 development.

7. Introduction to Cross-Compilation Toolchain for SDK

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

7.1 U-Boot and Kernel Compilation Toolchain

The SDK prebuilts directory contains pre-configured cross-compilers for U-Boot and Kernel compilation, as follows:

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

You can download the toolchain from the following address:

Click here

7.2 Buildroot Toolchain

7.2.1 Configuring the Compilation Environment

To compile individual modules or third-party applications, a cross-compilation environment must be set up. For example, for the RK3588, the cross-compilation tools are located in the `buildroot/output/rockchip_rk3588/host/usr` directory. The `bin/` directory and the `aarch64-buildroot-linux-gnu/bin/` directory need to be set as environment variables. Execute the script to automatically configure the 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 the 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 have the need. After extraction, the path to `gcc` will be:

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

7.3 Debian Toolchain

Use Docker on the host machine, gcc, or `dpkg-buildpackage` for related compilations.

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 arguments.
Refer to the compilation instructions in `device/rockchip/common/README.md` for details.

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_linux6.1_release_v1.1.0_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 RK3588 EVB obtained from Rockchip was EVB1. After that, the default configuration for RK3588 EVB obtained from Rockchip was EVB7.

8.3 SDK Custom Configuration

The SDK can be configured through the `make menuconfig` command, with the main configurable components currently being as follows:

```
(rockchip_rk3588_evb1_lp4_v10_defconfig) Name of defconfig to save
[*] Rootfs (Buildroot|Debian|Yocto)  --->
[*] 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 hardware and load the main operating system.
- AMPAK: A multi-core heterogeneous boot solution suitable for application scenarios requiring real-time performance.
- Kernel: Here you configure kernel options to customize the Linux kernel according to your hardware and application needs.
- Boot: Here you configure the support format for the Boot partition.
- 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: Enabling security features, including Secureboot methods, Optee storage methods, and writing Keys.

- Extra partitions: Used to configure additional partitions.
- Firmware: Here you configure firmware-related options.
- Update (Rockchip update image): Used to configure options for Rockchip's complete firmware.
- Others configurations: Other 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 configurations so that the customized compilation will be based on 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 a smooth update for each iteration of the Software Development Kit (SDK), it is recommended to clean up the previous compilation artifacts before updating. This practice helps to 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 root directory of the project and execute the following commands to automatically complete all compilations:

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

./build.sh     # Compiles the 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 a different rootfs can be specified by setting the environment variable `RK_ROOTFS_SYSTEM`. `RK_ROOTFS_SYSTEM` currently supports three systems: buildroot, debian, and yocto.

For example, to generate a debain system, the following commands can be used:

```
export RK_ROOTFS_SYSTEM=debian
./build.sh all-release
or
RK_ROOTFS_SYSTEM=debian ./build.sh all-release
```

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, a recovery.img will be generated in the Buildroot directory `output/rockchip_rk3588_recovery/images`.

> Note: The recovery.img includes the kernel, so every time there is a change in the Kernel, Recovery needs to be repackaged. 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 directory, with the default format being rootfs.ext4.

For specific details, refer to the Buildroot development documentation:

`<SDK>/docs/en/Linux/System/Rockchip_Developer_Guide_Buildroot_EN.pdf`

8.5.4.1 Building Buildroot Modules

Set up configurations for different chips and target features by using `source buildroot/envsetup.sh`.

```
$ source buildroot/envsetup.sh
Top of tree: rk3588

You're building on Linux
Lunch menu...pick a combo:

70. rockchip_rk3588
71. rockchip_rk3588_base
72. rockchip_rk3588_ramboot
73. rockchip_rk3588_recovery
...

Which would you like? [1]:
```

The default selection is 70, `rockchip_rk3588`. Then proceed to the RK3588 Buildroot directory to begin compiling the relevant modules.

`rockchip_rk3588_base` is used for compiling the base modules, and `rockchip_rk3588_recovery` is for compiling the Recovery module.

For example, to compile the rockchip-test module, the common compilation commands are as follows:

Navigate to the buildroot directory

```
<SDK># cd buildroot
```

- Delete and recompile rockchip-test

```
buildroot# make rockchip-test-recreate
```

- Rebuild rockchip-test

```
buildroot# make rockchip-test-rebuild
```

- Delete rockchip-test

```
buildroot# make rockchip-test-dirclean
or
buildroot# rm -rf output/rockchip_rk3588/build/rockchip-test-master/
```

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/System/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 the Rootfs:

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

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

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 above:

```
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

- If you encounter git permission issues that cause compilation errors.

Due to the addition of the CVE-2022-39253 security detection patch in the newer version of git, if an older version of Yocto is required, poky should include the following to fix it:

```
commit ac3eb2418aa91e85c39560913c1ddfd2134555ba
Author: Robert Yang <liezhi.yang@windriver.com>
Date:   Fri Mar 24 00:09:02 2023 -0700

    bitbake: fetch/git: Fix local clone url to make it work with repo

    The "git clone /path/to/git/objects_symlink" couldn't work after the
following
    change:

    https://github.com/git/git/commit/6f054f9fb3a501c35b55c65e547a244f14c38d56
```

Alternatively, you can also revert the git version on the PC to V2.38 or an earlier version, such as the following:

```
$ 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
```

8.6 Firmware Packaging

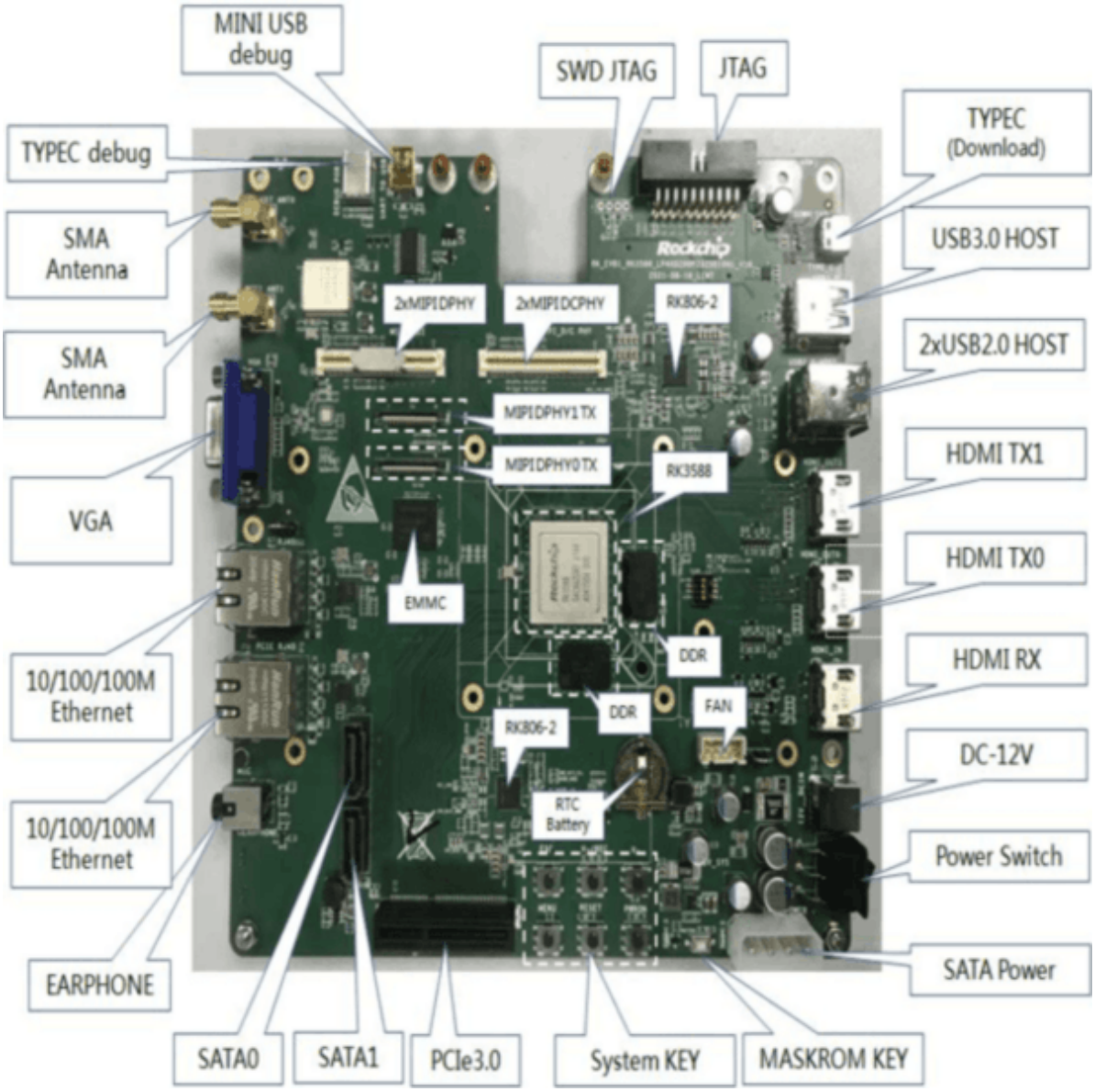
After compiling the various parts such as Kernel, U-Boot, Recovery, and Rootfs, navigate to the root directory of the project and execute the following command to automatically complete the packaging of all firmware to the `output/firmware` directory:

Firmware Generation:

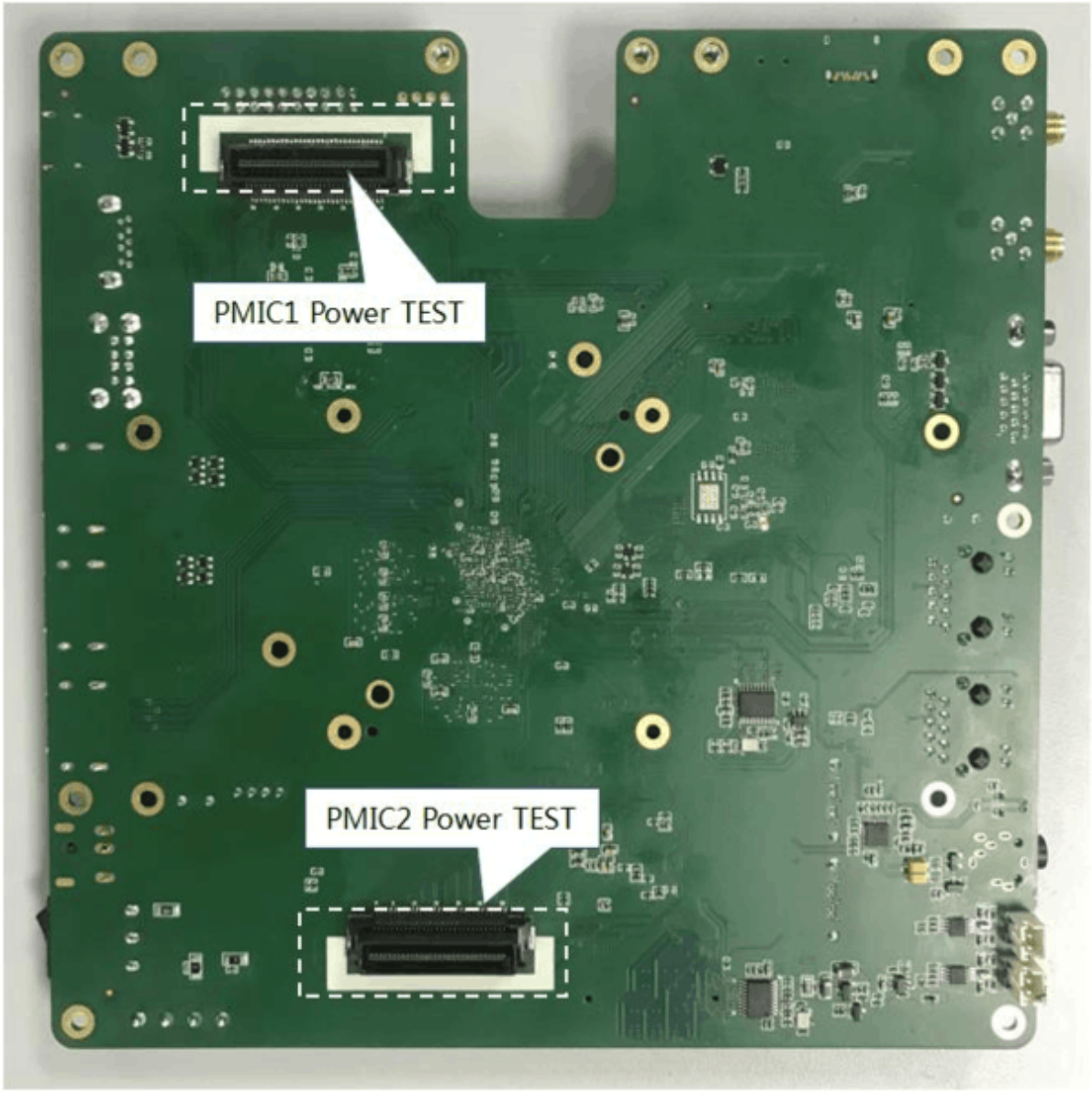
```
./build.sh firmware
```

9. Flashing Instructions

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



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

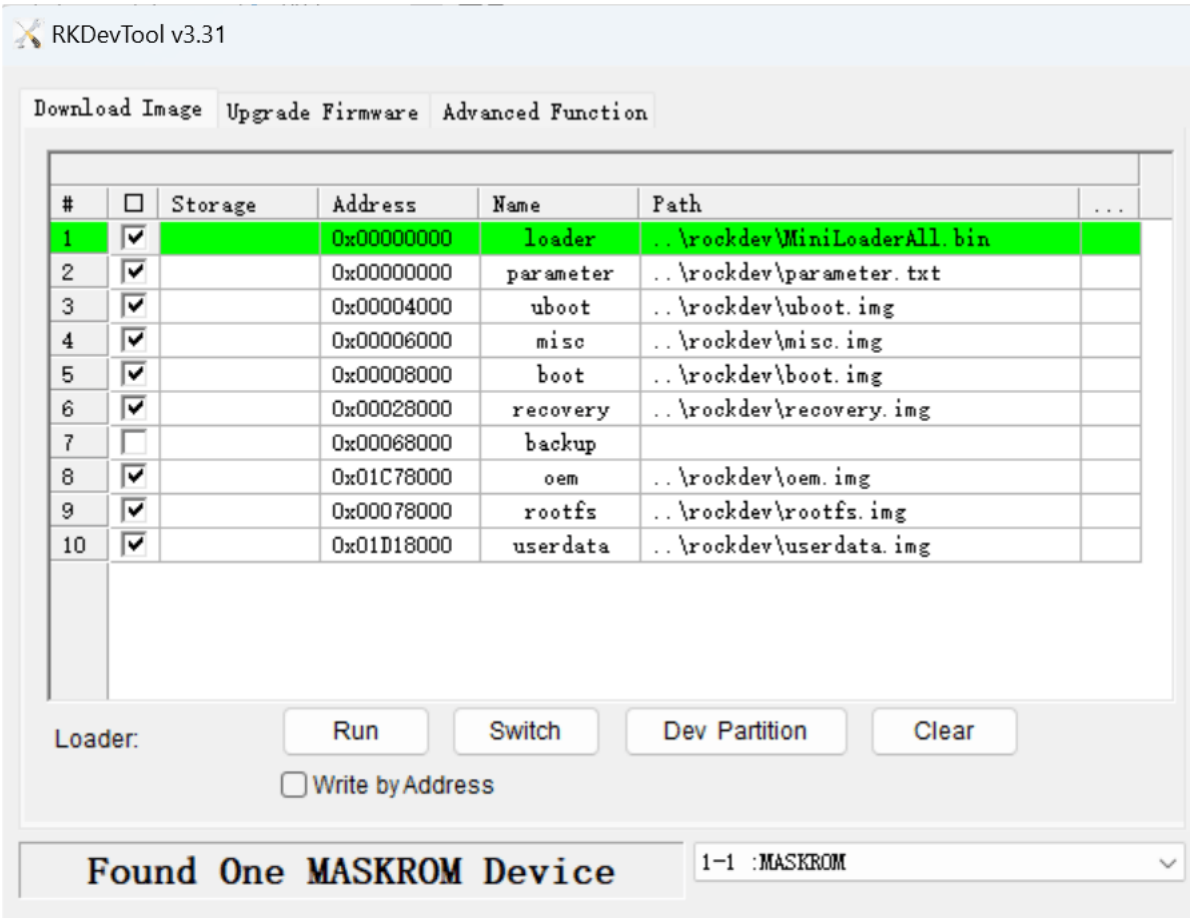


9.1 Windows Flashing Instructions

The SDK provides a Windows flashing tool (the tool version must be V3.31 or higher), 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 and hold the "MASKROM" button without releasing, then press and release the reset button "RST" to enter the MASKROM mode. Load the corresponding path of the compiled firmware and click "Execute" to perform the flashing. Alternatively, hold the "recovery" button without releasing and press the reset button "RST" to enter the loader mode for flashing. Below are the partition offsets for MASKROM mode and the flashing files.
(Note: The Windows PC needs to run the tool with administrator privileges to execute)



Note: Before flashing, the latest USB driver must be installed. For more details on the driver, 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 above). 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 the packaged complete firmware:

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

Or in the root directory, when the machine is running in MASKROM state, 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 boot.img compiled from the kernel.
- The recovery partition: for recovery.img compiled from recovery.
- The backup partition: reserved, not in use at the moment.
- The rootfs partition: for rootfs.img compiled from buildroot, debian, or yocto.
- The oem partition: for the manufacturer's use, storing manufacturer's apps or data. Mounted in the /oem directory.
- The userdata partition: for temporary file generation by apps or for final user use. Mounted in the /userdata directory.