Rockchip RK3576 Linux SDK Quick Start Guide

Document Identifier: RK-JC-YF-A20

Release Version: V1.0.0

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 RK3576 Linux SDK, aiming to help developers quickly understand and use the RK3576 SDK development package.

Target Audience

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

Technical Support Engineers

Software Development Engineers

Chip System Support Status

Chip Name	Uboot Version	Kernel Version	Buildroot Version	Yocto Version	Debian Version
RK3576	2017.9	6.1	2024.02	5.0	12

Revision Record

Date	Version	Author	Modification Description
2024-04-20	V0.1.0	Caesar Wang	Initial version
2024-06-20	V1.0.0	Caesar Wang	Release version

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 RK3576 Linux SDK image, which allows for a quick start in development and related assessments and comparisons. This can reduce the waste of development time and cost due to compilation issues.

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

Firmware path: **Universal Linux SDK Firmware** -> **Linux 6.1** -> **RK3576**

For those who need to modify the SDK code or require a quick start, 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 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: System with Ubuntu 22.04 or a higher version.

2.2 Installing Libraries and Toolkits

When developing for devices using the command line, you can install the necessary libraries and tools required for compiling the SDK by following these 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 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 version >= 3.6 is not met, you can upgrade by the following method:

- 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:

- 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. Docker Environment Setup

To assist developers in quickly completing the complex preparation of the development environment mentioned above, we also provide a cross-compiler Docker image for clients to rapidly verify, thereby reducing 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 URL 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/RK3576` directory and will be continuously improved and updated.

4.2 Chip Documentation

To assist development engineers in quickly getting started with the development and debugging of RK3576, the SDK release includes the chip manuals "RK3576_Brief_Datasheet_V1.3-20240315.pdf" and "Rockchip_RK3576_Datasheet_V1.1-20240430.pdf". They can be obtained in the `docs/en/RK3576/Datasheet` directory and will be continuously improved and updated.

4.3 Buildroot Development Guide

To assist development engineers in quickly getting started with the development and debugging of the Buildroot system, the development guide "Rockchip_Developer_Guide_Buildroot_EN.pdf" is released along with the SDK. It 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 becoming proficient 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 mastering the porting and adaptation of third-party operating systems, the SDK release includes the development guide titled "Rockchip_Developer_Guide_Third_Party_System_Adaptation_EN.pdf". It can be obtained in the `docs/en/Linux/System` directory and will be continuously improved and updated.

4.6 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 a PC:

The development kit is located in the `external/rknn-toolkit2` directory, mainly 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 through the simulator on the PC 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 the model 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 current `doc/` directory documentation:

```
├── 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-Toolkit2 generated rknn models.
For specific usage instructions, please refer to the current `doc/` directory documentation:

```
...
├── 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
```

4.7 DPDK Development Guide

To assist development engineers in quickly mastering the development and debugging of RK3576 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.

4.8 Real-Time Linux Development Guide

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

Below are the latency results from stress tests conducted on RK3576 using Buildroot, based on both PREEMPT_RT and Xenomai, as follows:

Stress Test:

```
stress-ng -c8 --io 8 --cpu-load 100 -vm 4 --vm-bytes 512M --timeout 10000000s
```

- PREEMPT_RT

```
root@rk3576-buildroot:/# cyclictest -c0 -m -t -p99 -D8H
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 24.41 23.87 23.55 17/341 2533
T: 0 ( 1425) P:99 I:1000 C:28799993 Min:      3 Act:    6 Avg:    9 Max:      44
T: 1 ( 1426) P:99 I:1500 C:19199994 Min:      3 Act:    7 Avg:    9 Max:      36
T: 2 ( 1427) P:99 I:2000 C:14399992 Min:      3 Act:    7 Avg:    8 Max:      35
T: 3 ( 1428) P:99 I:2500 C:11519992 Min:      3 Act:   11 Avg:   10 Max:      39
T: 4 ( 1429) P:99 I:3000 C:9599985 Min:      2 Act:   10 Avg:    8 Max:      41
T: 5 ( 1430) P:99 I:3500 C:8228558 Min:      2 Act:   12 Avg:    7 Max:      35
T: 6 ( 1431) P:99 I:4000 C:7199986 Min:      2 Act:    8 Avg:    9 Max:      48
T: 7 ( 1432) P:99 I:4500 C:6399988 Min:      2 Act:    6 Avg:    7 Max:      38
(tested for 8 hours)
```

- Xenomai Cobalt Mode

```
root@rk3576:/# ./usr/demo/cyclictest -c0 -m -n -t -p99 -D 2H
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 21.61 21.60 21.64 20/241 1759

T: 0 ( 1653) P:99 I:1000 C:7199993 Min:      2 Act:    7 Avg:    7 Max:      43
T: 1 ( 1654) P:99 I:1500 C:4799991 Min:      2 Act:   11 Avg:    6 Max:      40
T: 2 ( 1655) P:99 I:2000 C:3599991 Min:      2 Act:    5 Avg:    6 Max:      43
T: 3 ( 1656) P:99 I:2500 C:2879986 Min:      2 Act:   16 Avg:    6 Max:      39
T: 4 ( 1657) P:99 I:3000 C:2399988 Min:      2 Act:    7 Avg:    7 Max:      42
T: 5 ( 1658) P:99 I:3500 C:2057132 Min:      2 Act:   12 Avg:    6 Max:      45
T: 6 ( 1659) P:99 I:4000 C:1799991 Min:      2 Act:    7 Avg:    8 Max:      44
T: 7 ( 1660) P:99 I:4500 C:1599991 Min:      2 Act:   16 Avg:    8 Max:      47
(tested for 2 hours)
```

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

4.9 Software Update Log

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

```
.repo/manifests$ realpath rk3576_linux6.1_release.xml
# For example: The printed version number is v0.1.0, and the update time is
20240420
# <SDK>/.repo/manifests/release/rk3576_linux_beta_v0.1.0_20240420.xml
```

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

```
<SDK>/.repo/manifests/RK3576_Linux6.1_SDK_Note.md
or
<SDK>/docs/en/RK3576/RK3576_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-rkr2
ID=buildroot
VERSION_ID=2024.02
PRETTY_NAME="Buildroot 2024.02"
OS="buildroot"
RK_BUILD_INFO="xxx Thu Apr 18 17:59:46 CST 2024 - rockchip_rk3576"
```

5. Hardware Development Guide

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

RK3576 Hardware Design Guide:

```
<SDK>/docs/en/RK3576/Hardware/
├── RK3576_Hardware_Design_Guide_V1.0_20240415_EN.pdf
```

RK3576 EVB Hardware User Guide:

```
<SDK>/docs/en/RK3576/Hardware/
└── Rockchip_RK3576_EVB1_User_Guide_V1.0_EN.pdf
```

6. SDK Project Directory Overview

The SDK project directory includes directories such as buildroot, debian, yocto, app, kernel, u-boot, device, docs, external, prebuilts, 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: The root file system developed based on Buildroot (2024).
- debian: The root file system developed based on Debian bookworm (12).
- device/rockchip: Contains chip-level board configurations as well as scripts and files for SDK compilation and firmware packaging.
- docs: Contains general development guidance documents, chip platform-related documents, Linux system development-related documents, and other reference documents.
- external: Contains third-party related repositories, including display, audio and video, camera, network, security, etc.
- kernel: Contains the code for Kernel development.
- output: Contains information about each firmware generation, compilation information, XML, host environment, etc.
- prebuilts: Contains cross-compilation toolchains.
- rkbin: Contains Rockchip-related binaries and tools.

- rockdev: Contains compiled output firmware, actually a soft link to `output/firmware`.
- tools: Contains commonly used tools for Linux and Windows operating systems.
- u-boot: Contains U-Boot code developed based on the v2017.09 version.
- yocto: Contains the root file system developed based on Yocto 5.0.

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 provided the cross-compilation toolchain for Linux only. The prebuilt toolchain in the prebuilt directory is for use with U-Boot and Kernel. Specific Rootfs requires the corresponding toolchain for each, or a third-party toolchain for compilation.

7.1 U-Boot and Kernel Compilation Toolchain

The SDK prebuilts directory contains cross-compilation tools currently used 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 instance, for the RK3576, the cross-compilation tools are located in the `buildroot/output/rockchip_rk3576/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 the environment variables in the top-level directory:

```
source buildroot/envsetup.sh rockchip_rk3576
```

Enter the command to check:

```
cd buildroot/output/rockchip_rk3576/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 standalone compilation by third-party applications. 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

Utilize Docker on the machine side, gcc, or `dpkg-buildpackage` for the relevant compilation.

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. For detailed instructions, refer to the compilation guide in `device/Rockchip/common/README.md`.

To ensure a smooth update process for the SDK each time, it is recommended to clean the previous compilation artifacts before updating. This practice helps to avoid potential compatibility issues or compilation errors that may arise from using old compilation artifacts with a new version of the SDK. To clean these artifacts, simply run the command `./build.sh cleanall`.

One-Click Compilation

Chip	Type	Reference Model	One-Click Compilation	Rootfs Compilation	Kernel Compilation	U-Boot Compilation
RK3576	Development Board	RK3576 EVB1	./build.sh lunch:rockchip_rk3576_evb1_v10_defconfig && ./build.sh	./build.sh rootfs	make ARCH=arm64 rockchip_linux_defconfig rk3576.config; make ARCH=arm64 rk3576-evb1-v10-linux.img -j24	./make.sh rk3576 --spl-new
RK3576	Industrial Board	RK3576 INDUSTRY	./build.sh lunch:rockchip_rk3576_industry_evb_v10_defconfig rk3576.config && ./build.sh	./build.sh rootfs	make ARCH=arm64 rockchip_linux_defconfig rk3576.config; make ARCH=arm64 rk3576-evb1-v10-linux.img -j24	./make.sh rk3576 --spl-new
RK3576	IO Test Board	RK3576 IOTEST	./build.sh lunch:rockchip_rk3576_iotest_v10_defconfig rk3576.config && ./build.sh	./build.sh rootfs	make ARCH=arm64 rockchip_linux_defconfig rk3576.config; make ARCH=arm64 rk3576-iotest-v10-linux.img -j24	./make.sh rk3576 --spl-new
RK3576	Single-Eye IPC	RK3576 EVB1	./build.sh lunch:rockchip_rk3576_ipc_evb1_v10_defconfig rk3576.config && ./build.sh	./build.sh rootfs	make ARCH=arm64 rockchip_linux_defconfig rk3576.config; make ARCH=arm64 rk3576-evb1-v10-linux.img -j24	./make.sh rk3576 --spl-new
RK3576	Multi-Eye IPC	RK3576 EVB1	./build.sh lunch:rockchip_rk3576_multi_ipc_evb1_v10_defconfig rk3576.config && ./build.sh	./build.sh rootfs	make ARCH=arm64 rockchip_linux_defconfig rk3576.config; make ARCH=arm64 rk3576-evb1-v10-ipc-4x-linux.img -j24	./make.sh rk3576 --spl-new
RK3576	Test Board 1	RK3576 TEST1	./build.sh lunch:rockchip_rk3576_test1_v10_defconfig rk3576.config && ./build.sh	./build.sh rootfs	make ARCH=arm64 rockchip_linux_defconfig rk3576.config; make ARCH=arm64 rk3576-test1-v10-linux.img -j24	./make.sh rk3576 --spl-new
RK3576	Test Board 2	RK3576 TEST2	./build.sh lunch:rockchip_rk3576_test2_v10_defconfig rk3576.config && ./build.sh	./build.sh rootfs	make ARCH=arm64 rockchip_linux_defconfig rk3576.config; make ARCH=arm64 rk3576-test2-v10-linux.img -j24	./make.sh rk3576 --spl-new
RK3576	Vehicle EVB Board	RK3576 VEHICLE EVB	./build.sh lunch:rockchip_rk3576_vehicle_evb_v10_defconfig rk3576.config rk3576_vehicle.config && ./build.sh	./build.sh rootfs	make ARCH=arm64 rockchip_linux_defconfig rk3576.config; make ARCH=arm64 rk3576-vehicle-evb-v10-linux.img -j24	./make.sh rk3576 --spl-new

> **Note**:
>
> - **Rootfs Compilation**:
> The default is to compile the Buildroot system. If other systems are required, set the corresponding environment variables. For example,
> To compile the Buildroot system: RK_ROOTFS_SYSTEM=buildroot ./build.sh
> To compile the Debian system: RK_ROOTFS_SYSTEM=debian ./build.sh
> To compile the Yocto system: RK_ROOTFS_SYSTEM=yocto ./build.sh
> - **Kernel and U-Boot Compilation**: It is necessary to specify the toolchain.
> - For example, the SDK's built-in 32-bit toolchain:
> export CROSS_COMPILE=../prebuilts/gcc/linux-x86/arm/gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabihf/bin/arm-none-linux-gnueabihf-
> - For example, the SDK's built-in 64-bit toolchain:
> 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-

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`

You can also compile related features by running `./build.sh <target>`, and you can view the specific compilation commands through `./build.sh help`.

```
./build.sh -h

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

Manifest: rk3576_linux6.1_release_20240620_v1.0.0.xml

Log colors: message notice warning error fatal

Usage: build.sh [OPTIONS]
Available options:
chip[:<chip>[:<config>]]            select chip
defconfig[:<config>]                select defconfig
 *_defconfig                        switch to specified defconfig
    available defconfigs:
    rockchip_defconfig
    rockchip_rk3576_evb1_v10_defconfig
    rockchip_rk3576_industry_evb_v10_defconfig
    rockchip_rk3576_iotest_v10_defconfig
    rockchip_rk3576_ipc_evb1_v10_defconfig
    rockchip_rk3576_multi_ipc_evb1_v10_defconfig
    rockchip_rk3576_test1_v10_defconfig
    rockchip_rk3576_test2_v10_defconfig
    rockchip_rk3576_vehicle_evb_v10_defconfig
 olddefconfig                       resolve any unresolved symbols in .config
 savedefconfig                      save current config to defconfig
 menuconfig                         interactive curses-based configurator
config                              modify SDK defconfig
print-parts                         print partitions
list-parts                          alias of print-parts
mod-parts                           interactive partition table modify
edit-parts                          edit raw partitions
new-parts:<offset>:<name>:<size>... re-create partitions
insert-part:<idx>:<name>[:<size>]   insert partition
del-part:(<idx>|<name>)             delete partition
move-part:(<idx>|<name>):<idx>      move partition
```

```
rename-part:(<idx>|<name>):<name>   rename partition
resize-part:(<idx>|<name>):<size>   resize partition
misc                                pack misc image
kernel-6.1[:dry-run]                build kernel 6.1
kernel[:dry-run]                    build kernel
recovery-kernel[:dry-run]           build kernel for recovery
modules[:dry-run]                   build kernel modules
linux-headers[:dry-run]             build linux-headers
kernel-config[:dry-run]             modify kernel defconfig
kconfig[:dry-run]                   alias of kernel-config
kernel-make[:<arg1>:<arg2>]         run kernel make
kmake[:<arg1>:<arg2>]               alias of kernel-make
wifibt[:<dst dir>[:<chip>]]         build Wifi/BT
amp                                 build and pack amp system
buildroot-config[:<config>]         modify buildroot defconfig
bconfig[:<config>]                  alias of buildroot-config
buildroot-make[:<arg1>:<arg2>]      run buildroot make
bmake[:<arg1>:<arg2>]               alias of buildroot-make
rootfs[:<rootfs type>]              build default rootfs
buildroot                           build buildroot rootfs
yocto                               build yocto rootfs
debian                              build debian rootfs
recovery                            build recovery
security-createkeys                 create keys for security
security-misc                       build misc with system encryption key
security-ramboot                    build security ramboot
security-system                     build security system
loader[:dry-run]                    build loader (u-boot)
uboot[:dry-run]                     build u-boot
u-boot[:dry-run]                    alias of uboot
uefi[:dry-run]                      build uefi
extra-parts                         pack extra partition images
firmware                            pack and check firmwares
edit-package-file                   edit package-file
edit-ota-package-file               edit package-file for OTA
updateimg                           build update image
ota-updateimg                       build update image for OTA
all                                 build images
release                             release images and build info
all-release                         build and release images
shell                               setup a shell for developing
cleanall                            cleanup
clean[:module[:module]]...          cleanup modules
    available modules:
    all
    amp
    config
    extra-parts
    firmware
    kernel
    loader
    misc
    recovery
    rootfs
    security
    updateimg
```

```
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/rk3576`:

Board-Level Configuration	Description
rockchip_defconfig	Default configuration, which will be symbolically linked to the RK3576 EVB1 development board configuration
rockchip_rk3576_evb1_v10_defconfig	Configuration suitable for RK3576 EVB1 AIOT board
rockchip_rk3576_industry_evb_v10_defconfig	Configuration suitable for RK3576 EVB1 industrial board
rockchip_rk3576_iotest_v10_defconfig	Configuration suitable for RK3576 IOTEST test board
rockchip_rk3576_ipc_evb1_v10_defconfig	Configuration suitable for RK3576 single-eye IPC EVB1 development board
rockchip_rk3576_multi_ipc_evb1_v10_defconfig	Configuration suitable for RK3576 multi-eye IPC EVB1 development board
rockchip_rk3576_test1_v10_defconfig	Configuration suitable for RK3576 TEST1 test board
rockchip_rk3576_test2_v10_defconfig	Configuration suitable for RK3576 TEST2 test board
rockchip_rk3576_vehicle_evb_v10_defconfig	Configuration suitable for RK3576 vehicle EVB development board

Configuration can be performed using `make lunch` or `./build.sh lunch`.

```
$ ./build.sh lunch

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

1. rockchip_defconfig
2. rockchip_rk3576_evb1_v10_defconfig
3. rockchip_rk3576_industry_evb_v10_defconfig
4. rockchip_rk3576_iotest_v10_defconfig
5. rockchip_rk3576_ipc_evb1_v10_defconfig
6. rockchip_rk3576_multi_ipc_evb1_v10_defconfig
7. rockchip_rk3576_test1_v10_defconfig
```

```
8. rockchip_rk3576_test2_v10_defconfig
9. rockchip_rk3576_vehicle_evb_v10_defconfig
Which would you like? [1]:
```

> Most customers receive the EVB1 board, which is suitable for general AIOT application scenarios.

Other feature configurations can be set through `make menuconfig` to configure related properties.

8.3 SDK Custom Configuration

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

```
(rockchip_rk3576_evb1_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 you configure the 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: Security features are enabled, including Secureboot methods, Optee storage methods, and burn-in keys.
- Extra partitions: Used to configure additional partitions.
- Firmware: Here you configure firmware-related options.
- Update (Rockchip update image): Used to configure options for the Rockchip complete firmware.

- Others configurations: Other additional configuration options.

The `make menuconfig` configuration interface provides a text-based user interface for selecting and configuring various options.
After the configuration is complete, use the `make savedefconfig` command to save these configurations, 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 Automatic 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 can prevent 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 # Compiles only module code (u-Boot, kernel, Rootfs, Recovery)
               # Further execute `./build.sh ./mkfirmware.sh` 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 rk3576.config
make ARCH=arm64 rk3576-evb1-v10-linux.img -j4

Note: `rk3576-evb1-v10-linux` can be replaced with the specific hardware DTS
```

- Method Three

```
cd kernel
export CROSS_COMPILE=aarch64-linux-gnu-
make ARCH=arm64 rockchip_linux_defconfig rk3576.config
make ARCH=arm64 rk3576-evb1-v10-linux.img -j4

Note: `rk3576-evb1-v10-linux` can be replaced with the specific hardware DTS
```

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_rk3576_recovery/images`.

> Note: The recovery.img includes the kernel, so every time the Kernel is modified, 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 the compilation, different formats of the image are generated in the `output/rockchip_rk3576/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 Buildroot Module Compilation

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

```
$ source buildroot/envsetup.sh
Top of tree: rk3576

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

65. rockchip_rk3576
66. rockchip_rk3576_recovery
...

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

The default selection is 66, `rockchip_rk3576`. Then proceed to the RK3576's Buildroot directory to start compiling the relevant modules.

The `rockchip_rk3576_recovery` is used 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_rk3576/build/rockchip-test-master/
```

8.5.5 Debian Compilation

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

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

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

For specific 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 directory yocto/build/latest.

The default user 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

- 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 must include the following to be fixed:

```
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 downgrade the PC's git version to V2.38 or an earlier version, for example:

```
$ 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

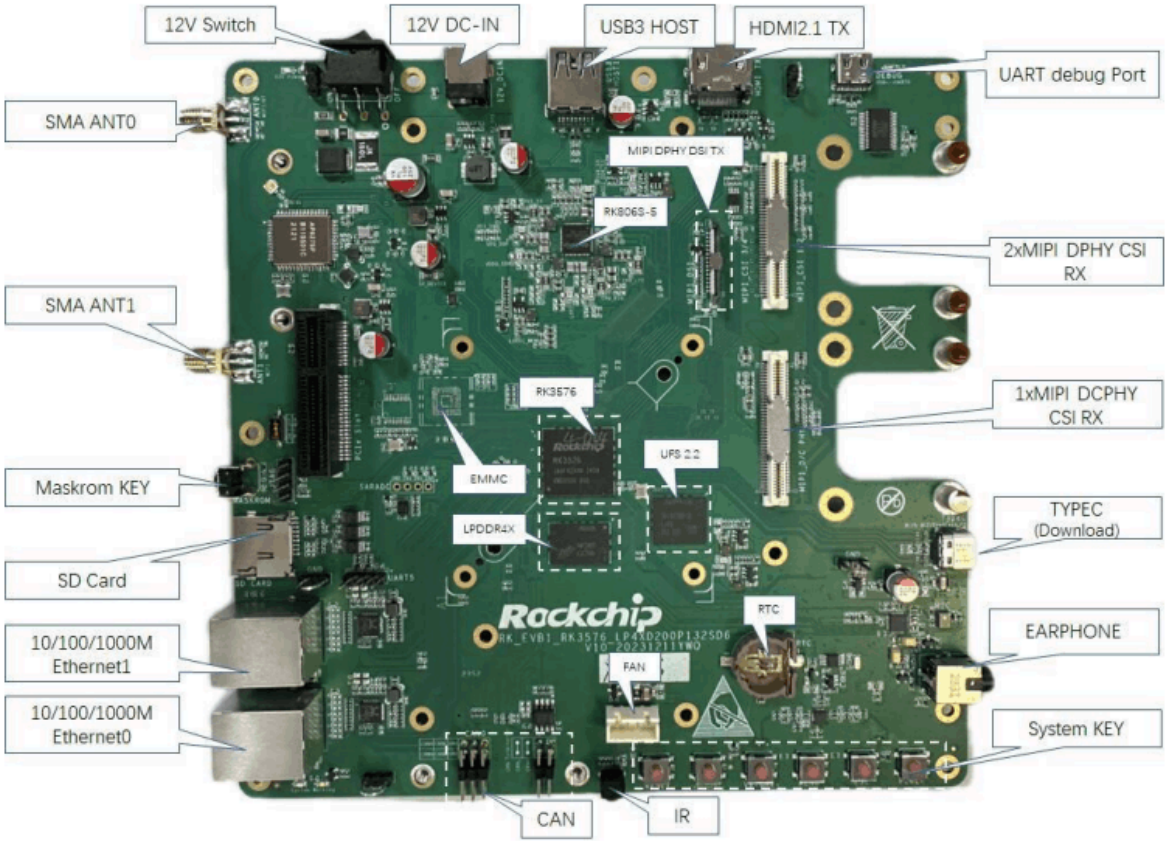
After compiling the various components such as Kernel, U-Boot, Recovery, and Rootfs, navigate to the root directory of the project and execute the following command to automatically complete the packaging of all firmware into the `output/firmware` directory:

Firmware Generation:

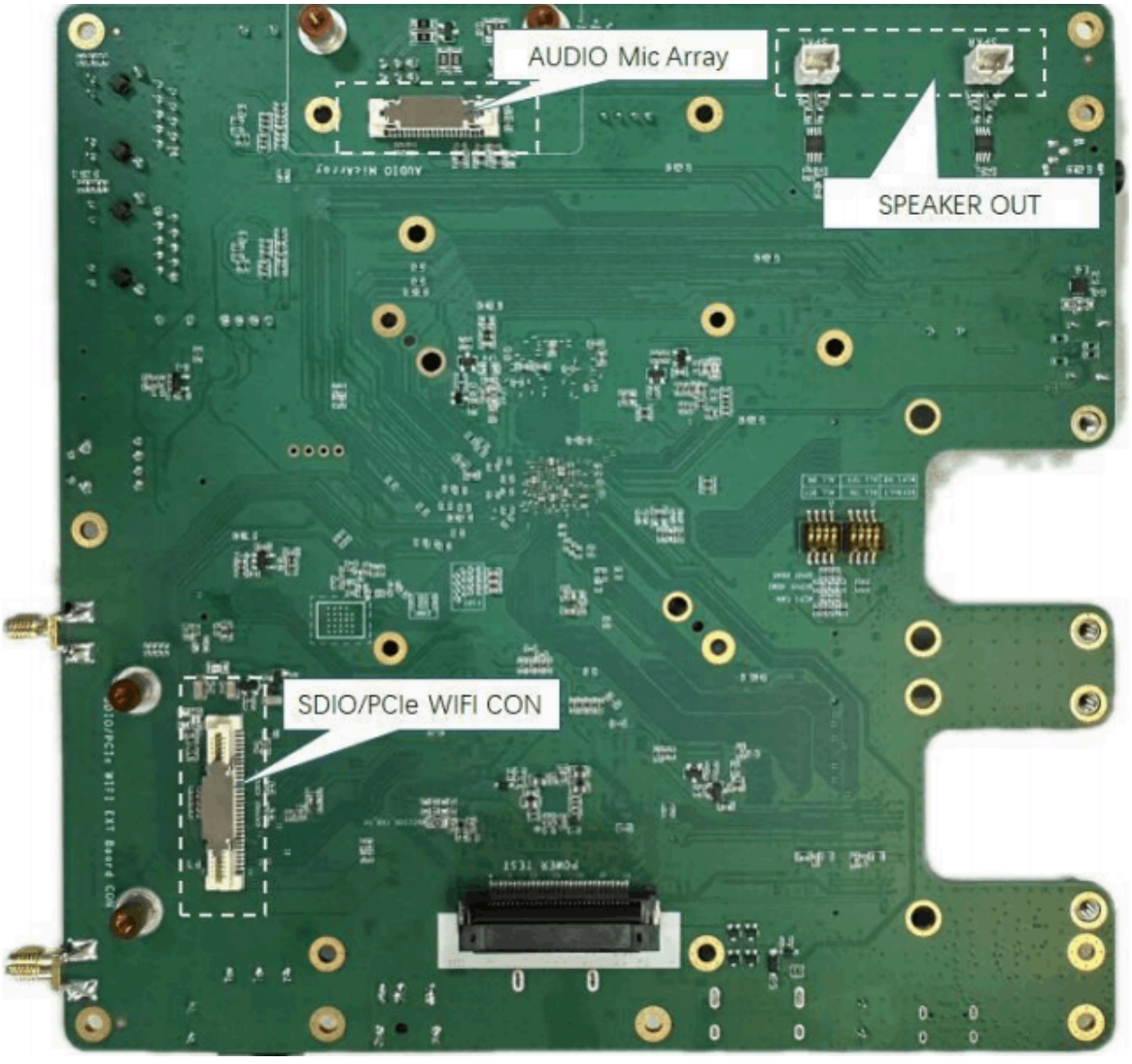
```
./build.sh firmware
```

9. Flashing Instructions

The front interface distribution of the RK3576 EVB 1 development board is as follows:



The back interface distribution of the RK3576 EVB1 development board is as follows:

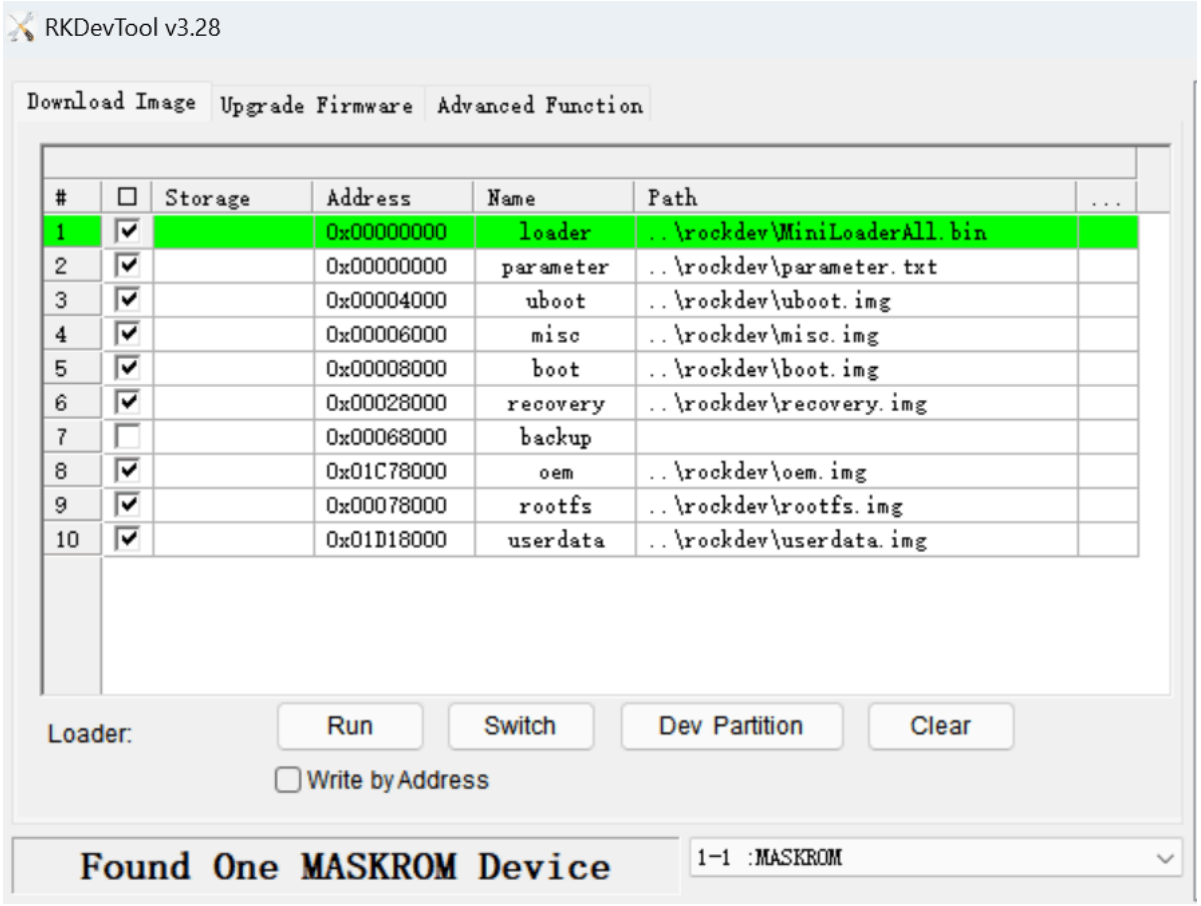


9.1 Windows Flashing Instructions

The SDK provides a Windows flashing tool (the tool version must be V3.28 or above), 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, then release to enter the MASKROM mode. Load the corresponding path of the compiled firmware and click "Execute" to flash. Alternatively, hold the "recovery" button without releasing and press the reset button "RST", then release 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 complete firmware after packaging:

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

Or upgrade in the root directory when the machine is running in MASKROM state:

```
./rkflash.sh
```

9.3 System Partition Description

Default Partition Description (Below is the RK3576 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	256GB	240GB	userdata

- uboot Partition: For the uboot.img compiled from uboot.
- misc Partition: For misc.img, used by recovery.

- boot Partition: For boot.img compiled from the kernel.
- recovery Partition: For recovery.img compiled by recovery.
- backup Partition: Reserved, not in use at the moment.
- rootfs Partition: For rootfs.img compiled by buildroot, debian, or yocto.
- oem Partition: For manufacturers to use, storing manufacturer's apps or data. Mounted at the /oem directory.
- userdata Partition: For apps to temporarily generate files or for end users, mounted under the /userdata directory.