Rockchip RK3566_RK3568 Linux SDK Quick Start

ID: RK-FB-YF-942

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

The document presents Rockchip RK3566/RK3568 Linux SDK release notes, aiming to help engineers get started with RK3566/RK3568 Linux SDK development and debugging faster.

Intended Audience

This document (this guide) is mainly intended for:

Technical support engineers

Software development engineers

Chip System Support Status

Chip Name	Uboot Version	Kernel Version	Debian Version	Buildroot Version
RK3566, RK3568	2017.9	5.10, 6.1	11, 12	2021.11, 2024.02

Revision Record

Date	Version	Author	Modification Description
2022-06-20	V1.0.0	Caesar Wang	Initial version.
2022-09-20	V1.0.1	Caesar Wang	Added support for Linux 5.10.
2022-11-20	V1.0.2	Caesar Wang	Updated flashing instructions.
2023-04-20	V1.1.0	Caesar Wang	Adapted to the new version of SDK.
2023-05-20	V1.1.1	Caesar Wang	Updated SDK to V1.1.1.
2023-06-20	V1.2.0	Caesar Wang	Updated SDK to V1.2.0.
2023-09-20	V1.3.0	Caesar Wang	Updated SDK to V1.3.0.
2023-12-20	V1.4.0	Caesar Wang	Updated SDK to V1.4.0.
2024-06-20	V1.5.0	Caesar Wang	Updated SDK to V1.5.0.

Contents

1. Precompiled SDK Images

Developers can bypass the process of compiling the entire operating system from source code by using the precompiled RK3566_RK3568 Linux SDK image. This allows for a quick start to development and related assessments and comparisons, reducing the time and cost wasted due to compilation issues.

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

For Linux 5.10 firmware, the path is: **General Linux SDK Firmware -> Linux 5.10 -> RK3566_RK3568**
For Linux 6.1 firmware, the path is: **General Linux SDK Firmware -> Linux 6.1 -> RK3566_RK3568**

If modifications to the SDK code or a quick start are needed, 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 more than 40GB of hard disk space. If you are performing multiple builds, you will need even more disk space.

Software Requirements: A system with Ubuntu 22.04 or a higher version.

2.2 Installing Libraries and Toolkits

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

Use the following apt-get command 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 applicable for 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. 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.

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 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 for checking and upgrading 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 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
```

3. Docker Environment Setup

To assist developers in quickly completing the complex preparation work for the development environment mentioned above, we also provide a cross-compiler Docker image to enable customers to quickly 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 following systems have been verified:

Distribution Version	Docker Version	Image Load	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 the SDK development and debugging process, the "Rockchip_Developer_Guide_Linux_Software_EN.pdf" is released along with the SDK. It can be obtained in the `docs/en/RK3566_RK3568` 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 RK3566 and RK3568, the chip manuals titled "Rockchip_RK3566_Datasheet_V1.4-20240621.pdf" and "Rockchip_RK3568_Datasheet_V2.1-20240621.pdf" are released alongside the SDK. They can be accessed in the `docs/en/RK3566_RK3568/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 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 becoming proficient with the development and debugging of RK3566 and RK3568 Debian, the "Rockchip_Developer_Guide_Debian_EN.pdf" development guide is released with the SDK. It 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 and familiarizing themselves with the porting and adaptation of third-party operating systems for RK3566 and RK3568, the SDK release includes the "Rockchip_Developer_Guide_Third_Party_System_Adaptation_EN.pdf" development guide, which can be obtained under `docs/en/Linux/System` and will be continuously improved and updated.

4.6 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.

The 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	Supported quantization types: asymmetric i8/fp16 Supports Layer / Channel quantization methods; Normal / KL / MMSE quantization algorithms Supports hybrid 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 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 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 models generated by RKNN-Toolkit2.
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 getting started with DPDK development and debugging, 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 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 using PREEMPT_RT and the Xenomai real-time system.

Below are the latency test results conducted on RK3566_RK3568 with Buildroot, based on both PREEMPT_RT and Xenomai, as follows:

4.8.1 Pressure Testing

```
stress-ng -c 4 --io 2 --vm 1 --vm-bytes 4M --timeout 1000000s
```

4.8.1.1 PREEMPT_RT Patch

```
rk3568_t:/ # /data/cyclictest -c 0 -m -n -t 4 -p 99
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 9.36 9.67 9.84 8/1152 3678

T: 0 ( 3482) P:99 I:1000 C:5892687 Min:      9 Act:   20 Avg:   25 Max:     123
T: 1 ( 3483) P:99 I:1500 C:3928444 Min:      9 Act:   20 Avg:   25 Max:     116
T: 2 ( 3484) P:99 I:2000 C:2946323 Min:      9 Act:   13 Avg:   26 Max:     120
T: 3 ( 3485) P:99 I:2500 C:2357050 Min:     10 Act:   20 Avg:   26 Max:     126
(2-hour test)
```

4.8.1.2 Xenomai Cobalt Mode

```
rk3568_t:/ # /data/cyclictest -c 0 -m -n -t 4 -p 99
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 7.65 7.80 7.75 8/1118 3998

T: 0 ( 3492) P:99 I:1000 C:21069118 Min:      2 Act:    4 Avg:   13 Max:      85
T: 1 ( 3493) P:99 I:1500 C:14046070 Min:      2 Act:   13 Avg:   15 Max:      76
T: 2 ( 3494) P:99 I:2000 C:10534550 Min:      2 Act:    6 Avg:   16 Max:      70
T: 3 ( 3495) P:99 I:2500 C:8427637 Min:      2 Act:   15 Avg:   15 Max:      96
(Testing for 5 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 engineering XML. The specific method is as follows:

```
.repo/manifests$ realpath rk356x_linux5.10_release.xml
# For example: The printed version number is v1.5.0, and the update time is
20240620
# <SDK>/.repo/manifests/rk356x_linux/rk356x_linux5.10_release_v1.5.0_20240620.xml
```

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

```
<SDK>/.repo/manifests/rk356x_linux/RK3566_RK3568_Linux5.10_SDK_Note.md
or
<SDK>/docs/en/RK3566_RK3568/RK3566_RK3568_Linux5.10_SDK_Note.md
```

- Board-side version information can be obtained 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  Thur Jun 20 23:12:04 CST 2024 - rockchip_rk3568"
KERNEL="5.10 - rockchip_linux_defconfig"
```

5. Hardware Development Guide

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

Rockchip RK3566 and RK3568 Hardware Design Guide:

```
<SDK>/docs/en/RK3566_RK3568/Hardware/Rockchip_RK3566_Hardware_Design_Guide_V1.1_2
0220206_EN.pdf
<SDK>/docs/en/RK3566_RK3568/Hardware/Rockchip_RK3568_Hardware_Design_Guide_V1.2_2
0211107_EN.pdf
```

Rockchip RK3566 and RK3568 EVB Hardware User Guide:

```
<SDK>/docs/en/RK3566_RK3568/Hardware/Rockchip_RK3568_EVB_User_Guide_V1.2_EN.pdf
<SDK>/docs/en/RK3566_RK3568/Hardware/Rockchip_RK3566_EVB2_User_Guide_V1.1_EN.pdf
```

6. IO Power Supply Design Considerations

The IO level of the controller power domain must be consistent with the IO level of the connected peripheral chip, and the voltage configuration of software must be consistent with the voltage of hardware to avoid GPIO damage.

 Note:

About matching of GPIO power domain and IO level:

PMUIO0_VDD, PMUIO1_VDD, VCCIO1_VDD, VCCIO2_VDD, VCCIO3_VDD, VCCIO4_VDD, VCCIO5_VDD, VCCIO6_VDD, VCCIO7_VDD, voltage of these GPIO power domain must be consistent with the IO level voltage of the connected peripheral to avoid GPIO damage.

Also need to note that the voltage configuration of software should be consistent with the voltage of hardware: For example, if hardware IO level is connected to 1.8V, the voltage configuration of software should be configured to 1.8V accordingly; if hardware IO level should be connected to 3.3V, and the voltage configuration of software should also be configured to 3.3V to avoid GPIO damage.

For more information, please refer to:

```
<SDK>/docs/en/RK3566_RK3568/Rockchip_RK356X_Introduction_IO_Power_Domains_Configuration.pdf
<SDK>/docs/en/Common/IO-DOMAIN/Rockchip_Developer_Guide_Linux_IO_DOMAIN_EN.pdf
```

7. SDK Configuration Framework Description

7.1 SDK Project Directory Overview

The SDK project directory includes directories such as buildroot, debian, rtos, app, kernel, u-boot, device, docs, and external. 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 and scripts and files for compiling and packaging firmware.
- docs: Contains development guidance documents, platform support lists, tool usage documents, and Linux development guides.
- external: Contains third-party related repositories, including display, audio and video, camera, network, and security.
- kernel: Contains code for kernel development.
- hal: Contains bare-metal development libraries based on RK HAL hardware abstraction layer for AMPAK solutions.
- output: Contains firmware information, compilation information, XML, and host environment 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 for Linux and Windows operating systems.
- u-boot: Contains U-Boot code based on the v2017.09 version for development.
- uefi: Contains UEFI code based on the edk2 V2.7 version for development.
- yocto: Root file system based on Yocto 4.0 or 5.0 development.

8. Introduction to Cross-Compilation Toolchain for SDK

Considering that the Rockchip Linux SDK is currently compiled only in the Linux PC environment, we have only provided the cross-compilation toolchain for Linux. The prebuilt toolchain in the prebuilt directory is for use with U-Boot and Kernel. Specific Rootfs should be compiled using their respective toolchains or third-party toolchains.

8.1 U-Boot and Kernel Compilation Toolchain

The SDK prebuilts directory contains cross-compiled 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

8.2 Buildroot Toolchain

8.2.1 Configuring the Compilation Environment

To compile individual modules or third-party applications, a cross-compilation environment must be configured. For instance, the cross-compilation tools for RK3568 are located in the `buildroot/output/rockchip_rk3568/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_rk3568
```

Enter the command to check:

```
cd buildroot/output/rockchip_rk3568/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
```

8.2.2 Packaging Toolchain

Buildroot supports the packaging of the built-in toolchain into a compressed package 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 require it. After extraction, the path to `gcc` will be:

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

8.3 Debian Toolchain

Utilizing Docker on the host machine, compile using gcc or `dpkg-buildpackage`.

8.4 Yocto Toolchain

Refer to the following:

Building your own recipes from first principles

New Recipe

9. 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`.

9.1 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 functions, and you can view the specific compilation commands through `./build.sh help`.

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

Manifest: rk356x_linux5.10_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'.
```

9.2 SDK Board-Level Configuration

Navigate to the project directory `<SDK>/device/rockchip/rk3566_rk3568`:

Board-Level Configuration	Description
rockchip_rk3566_evb2_lp4x_v10_32bit_defconfig	Suitable for RK3566 EVB with LPDDR4 development board, running a 32-bit root file system
rockchip_rk3566_evb2_lp4x_v10_defconfig	Suitable for RK3566 EVB with LPDDR4 development board
rockchip_rk3568_evb1_ddr4_v10_32bit_defconfig	Suitable for RK3568 EVB with LPDDR4 development board, running a 32-bit root file system
rockchip_rk3568_evb1_ddr4_v10_defconfig	Suitable for RK3568 EVB with DDR4 development board
rockchip_rk3568_evb8_lp4_v10_32bit_defconfig	Suitable for RK3568 EVB with LPDDR4/RK860X development board, running a 32-bit root file system
rockchip_rk3568_evb8_lp4_v10_defconfig	Suitable for RK3568 EVB8 with LPDDR4/RK860X development board
rockchip_rk3568_pcie_ep_lp4x_v10_defconfig	Suitable for RK3568 PCIe EP standard card development and verification
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 for **RK3566 EVB with LPDDR4 development board running a 32-bit root file system**:

```
./build.sh
device/rockchip/rk3566_rk3568/rockchip_rk3566_evb2_lp4x_v10_32bit_defconfig
```

Select the board-level configuration for **RK3566 EVB with LPDDR4 development board**:

```
./build.sh device/rockchip/rk3566_rk3568/rockchip_rk3566_evb2_lp4x_v10_defconfig
```

Select the board-level configuration for **RK3568 EVB with DDR4 development board running a 32-bit root file system**:

```
./build.sh
device/rockchip/rk3566_rk3568/rockchip_rk3568_evb1_ddr4_v10_32bit_defconfig
```

Select the board-level configuration for **RK3568 EVB with DDR4 development board**:

```
./build.sh device/rockchip/rk3566_rk3568/rockchip_rk3568_evb1_ddr4_v10_defconfig
```

Select the board-level configuration for **RK3568 EVB with LPDDR4/RK860X development board**:

```
./build.sh device/rockchip/rk3566_rk3568/rockchip_rk3568_evb8_lp4_v10_defconfig
```

Select the board-level configuration for **RK3568 EVB with LPDDR4/RK860X development board running a 32-bit root file system**:

```
./build.sh
device/rockchip/rk3566_rk3568/rockchip_rk3568_evb8_lp4_v10_32bit_defconfig
```

Select the board-level configuration for **RK3568 PCIe EP standard card development and verification**:

```
./build.sh
device/rockchip/rk3566_rk3568/rockchip_rk3568_pcie_ep_lp4x_v10_defconfig
```

Method 2

```
rk3566_rk3568$ ./build.sh lunch
Pick a defconfig:

1. rockchip_defconfig
2. rockchip_rk3566_evb2_lp4x_v10_32bit_defconfig
3. rockchip_rk3566_evb2_lp4x_v10_defconfig
4. rockchip_rk3568_evb1_ddr4_v10_32bit_defconfig
5. rockchip_rk3568_evb1_ddr4_v10_defconfig
6. rockchip_rk3568_evb8_lp4_v10_32bit_defconfig
7. rockchip_rk3568_evb8_lp4_v10_defconfig
8. rockchip_rk3568_pcie_ep_lp4x_v10_defconfig
9. rockchip_rk3568_uvc_evb1_ddr4_v10_defconfig
Which would you like? [1]:
```

9.3 Custom SDK Configuration

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

```
(rockchip_rk3568_evb1_ddr4_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 term 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 suit one's own hardware and application needs.
- Boot: Configuration for the boot partition support format is done 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: Security features are enabled, including Secureboot activation methods, Optee storage methods, and writing keys.
- Extra partitions: Used to configure additional partitions.
- Firmware: Firmware-related options are configured here.
- Update (Rockchip update image): Used to configure options for Rockchip's 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, and the customized compilation will be based on these settings.

Through the above configuration, you can choose different Rootfs/Loader/Kernel configurations for various customized compilations, allowing for flexible selection and configuration of system components to meet specific needs.

9.4 Fully Automated Compilation

Navigate to the root directory of the project and execute the following commands to automatically complete all compilations:

```
./build.sh all # Compiles only module code (u-Boot, kernel, Rootfs, Recovery)
               # Requires further execution of `./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 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
or
RK_ROOTFS_SYSTEM=debian ./build.sh
```

> Note:
> It is recommended to clean up previous compilation artifacts with each SDK update by running `./build.sh cleanall`.

9.5 Module Compilation

9.5.1 U-Boot Compilation

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

9.5.2 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

make ARCH=arm64 rk3566-evb2-lp4x-v10-linux.img -j16
or
make ARCH=arm64 rk3568-evb1-ddr4-v10-linux.img -j16
or
make ARCH=arm64 rk3568-evb8-lp4-v10-linux.img -j16
```

- Method Three

```
cd kernel
export CROSS_COMPILE=aarch64-linux-gnu-
make ARCH=arm64 rockchip_linux_defconfig

make ARCH=arm64 rk3566-evb2-lp4x-v10-linux.img -j16
or
make ARCH=arm64 rk3568-evb1-ddr4-v10-linux.img -j16
or
make ARCH=arm64 rk3568-evb8-lp4-v10-linux.img -j16
```

9.5.3 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 will be generated in the Buildroot directory `output/rockchip_rk3568_recovery/images`.

> Note: The recovery.img includes the kernel, so every time the Kernel is modified, 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.

9.5.4 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 formats of images are generated in the Buildroot directory `output/rockchip_rk3568/images`, 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`

9.5.4.1 Building Buildroot Modules

Configuration for different chips and target functionalities can be set using `source buildroot/envsetup.sh`.

```
$ source buildroot/envsetup.sh
Top of tree: rk3566_rk3568

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

57. rockchip_rk3566
58. rockchip_rk3566_32
59. rockchip_rk3566_recovery
60. rockchip_rk3566_rk3568_base
61. rockchip_rk3566_rk3568_ramboot
62. rockchip_rk3568
63. rockchip_rk3568_32
64. rockchip_rk3568_recovery

...

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

The default selection is 57, `rockchip_rk3566`. Then proceed to the RK3566's Buildroot directory to start compiling the relevant modules.

The `rockchip_rk3566_32` is used for compiling a 32-bit root file system, while `rockchip_rk3566_recovery` is for compiling the Recovery module.

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

Navigate to the buildroot directory

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

- Delete and recompile the rockchip-test

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

- Rebuild the rockchip-test

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

- Delete the rockchip-test

```
buildroot# make rockchip-test-dirclean
or
buildroot# rm -rf output/rockchip_rk3566/build/rockchip-test-master/
```

9.5.5 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`

9.5.6 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 compilation, a rootfs.img is generated in the `yocto` directory under `build/latest`.

The default username for login is `root`. For more information about 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
```

Alternatively, 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 used, the poky needs to 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
```

Or you can also revert the git version on the PC 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
```

9.6 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 into the `output/firmware` directory:

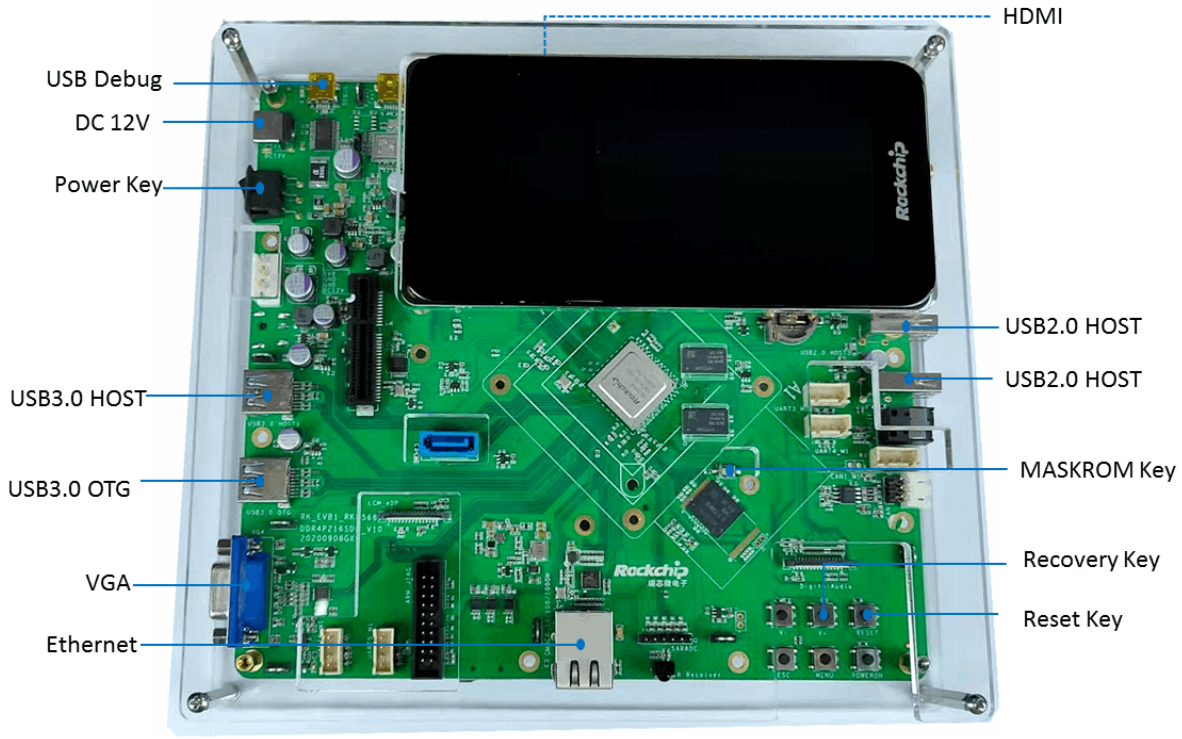
Firmware Generation:

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

10. Flashing Instructions

The interface distribution diagram for RK3566 EVB2 is as follows:

The interface distribution diagram for RK3568 EVB1 development board is as follows:



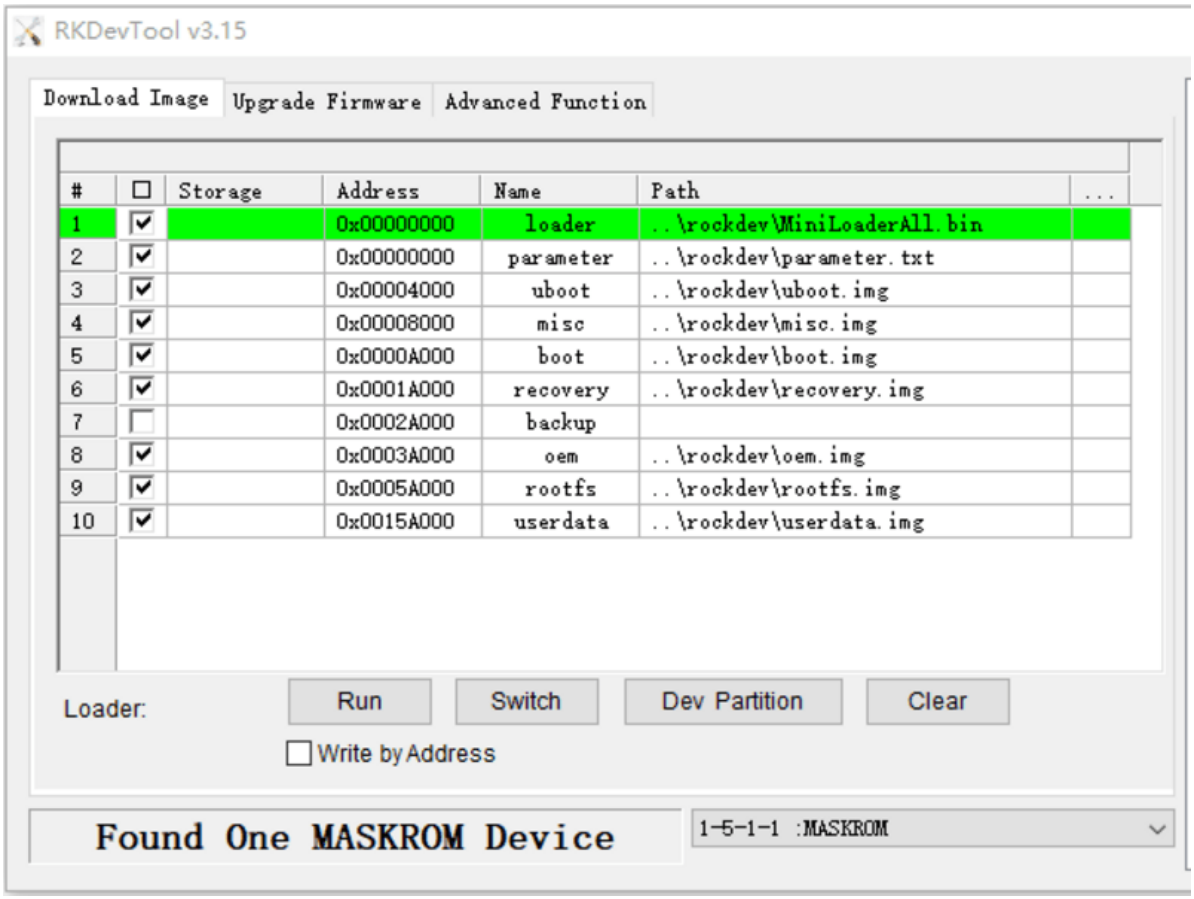
10.1 Windows Flashing Instructions

The SDK provides a Windows flashing tool (the tool version must be V3.31 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 must enter the MASKROM or BootROM flashing mode. Connect the USB download cable, hold the "MASKROM" button without releasing and press the reset button "RST", then release to enter the MASKROM mode. After loading the path of the compiled firmware, click "Execute" to perform the flashing. 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 drivers must be installed. For more details on the drivers, see:

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

10.2 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 in the root directory, when the machine is running in MASKROM state, run the following upgrade:

```
./rkflash.sh
```

10.3 System Partition Description

Default Partition Description (Below is the RK3568 EVB partition reference)

Number	Start (sector)	End (sector)	Size	Name
1	16384	24575	4M	uboot
2	24576	32767	4M	misc
3	32768	163839	64M	boot
4	163840	425983	128M	recovery
5	425984	491519	32M	backup
6	491520	13074431	6144M	rootfs
7	13074432	13336575	128M	oem
8	13336576	61120478	22.7G	userdata

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