Rockchip RK3399 Linux SDK Quick Start

ID: RK-FB-YF-944

Release Version: V1.3.0

Release Date: 2023-09-20

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 RK3399 Linux SDK release notes, aiming to help engineers get started with RK3399 Linux SDK development and debugging faster.

Intended Audience

This document (this guide) is mainly intended for:

Technical support engineers

Software development engineers

Chipset and System Support

Chipset	Buildroot	Debian	Yocto
RK3399	Y	Y	Y

Revision History

Date	Version	Author	Revision History
2022-06-20	V1.0.0	Caesar Wang	Initial version
2022-09-20	V1.0.1	Caesar Wang	Update it for linux5.10
2022-11-20	V1.0.2	Caesar Wang	Update Linux Upgrade Instruction
2023-04-20	V1.1.0	Caesar Wang	Adapt to the new version of SDK
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

Contents

1. Building a development environment

It is recommended to use Ubuntu 22.04 for compilation. Other Linux versions may need to adjust the software package accordingly. In addition to the system requirements, there are other hardware and software requirements. Hardware requirements: 64-bit system, hard disk space should be greater than 40G. If you do multiple builds, you will need more hard drive space

Considering the time cost of setting up the customer's development environment, we also provide the image mode of cross compiler docker for customer verification, so as to shorten the time-consuming of setting up the compilation environment.

Reference documents `Docker/Rockchip_Developer_Guide_Linux_Docker_Deploy_EN.pdf`.

The compatibility test results of docker compilation image system are as follows:

OS Vesion	Docker Version	Doading	Image build
ubuntu 22.04	20.10.21	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
fedora35	20.10.12	pass	NR (not run)

1.1 Install libraries and toolsets

When using the command line for device development, you can install the libraries and tools required for compiling the SDK by the following steps.

Use the following `apt-get` command to install the libraries and tools required for the following 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
```

> **Description:**
> The installation command is applicable to Ubuntu22.04. For other versions, please use the corresponding installation command according to the name of the installation package. If you encounter an error when compiling, you can install the corresponding software package according to the error message. in:
>
> - If the PC cannot access the Google website while compiling Buildroot, DNS needs to be set up to support downloading DL packages using the domestic image kgithub.com
> - Python 3.6 or later versions is required to be installed, and python 3.6 is used as an example here.
> - make requires make 4.0 and above to be installed, take make 4.2 as an example here.
> - lz4 1.7.3 or later versions is required to be installed.

> - Compiling yocto requires a VPN network, and git does not have the CVE-2022-39253 security detection patch.

1.1.1 Setting up DNS to support for kgithub.com

```
sudo sed -i '$a 43.154.68.204\tkgithub.com' /etc/hosts
sudo sed -i '$a 43.155.83.75\traw.kgithub.com
objects.githubusercontent.kgithub.com' /etc/hosts
```

1.1.2 Check and upgrade the `python` version of the host

The method of checking and upgrading the `python` version of the host is as follows:

- Check host `python` version

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

If you do not meet the requirements of python>=3.6 version, you can upgrade it in the following way:

- Upgrade `python 3.6.15` new version

```
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"
```

1.1.3 Check and Upgrade the `make` Version on the Host

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

- Check the `make` version on the host

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

- Upgrade to the new 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
```

1.1.4 Check and Upgrade the `lz4` Version on the Host

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

- Check the `lz4` version on the host

```
$ lz4 -v
*** LZ4 command line interface 64-bits v1.9.3, by Yann Collet ***
refusing to read from a console
```

- Upgrade to the 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
```

1.1.5 Check and Upgrade the `git` Version on the Host

- Check the `git` version on the host

```
$ /usr/bin/git -v
git version 2.38.0
```

- Upgrade to the new version of `git`

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

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

2. Software Development Guide

2.1 Development Guide

Aiming to help engineers get started with SDK development and debugging faster, We have released “Rockchip_Developer_Guide_Linux_Software_EN.pdf” with the SDK, please refer to the documents under the project's `docs/en/RK3399` directory.

2.2 Chip Datasheet

Aiming to help engineers get started with RK3399 development and debugging faster. We have released "Rockchip_RK3399_Datasheet_V2.1_20200323.pdf" , please refer to the documents under the project's `docs/en/RK3399/Datasheet` directory.

2.3 Debian Development Guide

Aiming to help engineers get started with RK3399 Debian development and debugging faster, “Rockchip_Developer_Guide_Debian_EN.pdf” is released with the SDK, please refer to the documents under the project's `docs/en/Linux/System` directory, which will be continuously improved and updated.

2.4 Third Party System Adaptation

Aiming to help engineers get started with Third Party System Adaptation faster, “Rockchip_Developer_Guide_Third_Party_System_Adaptation_EN.pdf” is released with the SDK, please refer to the documents under the project's `docs/en/Linux/System` directory, which will be continuously improved and updated.

2.5 Software Update History

Software release version upgrade history can be checked through project xml file by the following command:

```
.repo/manifests$ realpath rk3399_linux5.10_release.xml
# e.g.:the printed version is v1.3.0 and the update time is 20230920
# <SDK>/.repo/manifests/rk3399_linux/rk3399_linux5.10_release_v1.3.0_20230920.xml
```

Software release version updated information can be checked through the project text file by the following command:

```
<SDK>/.repo/manifests/rk3399_linux/RK3399_Linux5.10_SDK_Note.md
or
<SDK>/docs/en/RK3399/RK3399_Linux5.10_SDK_Note.md
```

3. Hardware Development Guide

Hardware related development can refer to the user guide, which can be found in the engineering directory:

RK3399 Hardware Design Guidelines:

```
<SDK>/docs/en/RK3399/Hardware/Rockchip_RK3399_Hardware_Design_Guide_V1.2_EN.pdf
```

RK3399 IND Industry Board Hardware Development Guide:

```
<SDK>/docs/en/RK3399/Hardware/Rockchip_RK3399_User_Manual_IND_EVB_V1.1_EN.pdf
```

4. The Precaution of IO Power Design

Please refer to the following documents for details:

```
<SDK>/docs/en/RK3399/Rockchip_RK3399_Introduction_IO_Power_Domains_Configuration.
pdf
<SDK>/docs/en/Common/IO-DOMAIN/Rockchip_Developer_Guide_Linux_IO_DOMAIN_EN.pdf
```

5. SDK Configuration Framework Introduction

5.1 SDK Project Directory Introduction

There are buildroot, debian, recovery, app, kernel, u-boot, device, docs, external and other directories in the project directory. Repositories are managed using manifests, and the repo tool is used to manage each directory or its subdirectory corresponding to a git.

- buildroot: root file system based on Buildroot.
- debian: root file system based on Debian.
- device/rockchip: store board-level configuration for each chip and some scripts and prepared files for building and packaging firmware.
- docs: stores development guides, platform support lists, tool usage, Linux development guides, and so on.
- external: stores some third-party libraries, including audio, video, network, recovery and so on.
- kernel: stores kernel development code.
- output: stores the firmware information, compilation information, XML, host environment, etc. generated each time.
- prebuilts: stores cross-building toolchain.
- rkbin: stores Rockchip Binary and tools.
- rockdev: stores building output firmware.
- tools: stores some commonly used tools under Linux and Windows system.
- u-boot: store U-Boot code developed based on v2017.09 version.
- yocto: stores the root file system developed based on Yocto 4.0.

6. SDK Building Introduction

The SDK can be accessed through `make` or `./build.sh` , add target parameters to configure and compile related functions.

6.1 SDK Compilation Commands

`make help` , e.g:

The SDK can configure and compile relevant features by using `make` or `./build.sh` with target parameters. Please refer to the `device/rockchip/common/README.md` compilation instructions for details.

6.2 SDK Compilation Command View

`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 operation of make is `./build.sh`

You can also run `./build.sh <target>` to compile the relevant functions, which can be done through `./build.sh help` View the specific compilation commands.

```
$ ./build.sh -h
Usage: build.sh [OPTIONS]
Available options:
lunch               - choose defconfig
*_defconfig         - switch to specified defconfig
olddefconfig        - resolve any unresolved symbols in .config
savedefconfig       - save current config to defconfig
menuconfig          - interactive curses-based configurator
kernel-5.10         - build kernel 5.10
kernel              - build kernel
modules             - build kernel modules
loader              - build loader (uboot|spl)
uboot               - build u-boot
spl                 - build spl
uefi                - build uefi
wifibt              - build Wifi/BT
rootfs              - build rootfs (default is buildroot)
buildroot           - build buildroot rootfs
yocto               - build yocto rootfs
debian              - build debian rootfs
recovery            - build recovery
pcba                - build PCBA
security_check      - check contidions for security features
createkeys          - build secureboot root keys
```

```
security_uboot     - build uboot with security paramter
security_boot      - build boot with security paramter
security_recovery  - build recovery with security paramter
security_rootfs    - build rootfs and some relevant images with security paramter
(just for dm-v)
updateimg          - build update image
otapackage         - build OTA update image
sdpackage          - build SDcard update image
firmware           - generate and check firmwares
all                - build all basic image
save               - save images and build info
allsave            - build all & firmware & updateimg & save
cleanall           - cleanup
post-rootfs        - trigger post-rootfs hook scripts
shell              - setup a shell for developing
help               - usage

Default option is 'allsave'.
```

6.3 SDK board-level configuration

Enter the project `<SDK>/device/rockchip/rk3399` directory:

Board level configuration	Note
rockchip_rk3399_evb_ind_lpddr4_defconfig	Suitable for RK3399 industry development board
rockchip_rk3399_firefly_defconfig	Suitable for RK3399 firefly development boards
rockchip_rk3399_sapphire_excavator_lp4_defconfig	Suitable for RK3399 sapphire excavator with LPDDR4 development board
rockchip_rk3399_sapphire_excavator_defconfig	Suitable for RK3399 sapphire excavator development board

The first way:

Add board configuration file behind `/build.sh` , for example:

Select the board configuration of **RK3399 industry development board**:

```
./build.sh device/rockchip/rk3399/rockchip_rk3399_evb_ind_lpddr4_defconfig
```

Select the board configuration of the **RK3399 firefly development board**:

```
./build.sh device/rockchip/rk3399/rockchip_rk3399_firefly_defconfig
```

Select the board-level configuration of the **RK3399 sapphire excavator with LPDDR4 development board**:

```
./build.sh
device/rockchip/rk3399/rockchip_rk3399_sapphire_excavator_lp4_defconfig
```

Select the board-level configuration of the **RK3399 sapphire excavator development board**:

```
./build.sh device/rockchip/rk3399/rockchip_rk3399_sapphire_excavator_defconfig
```

The second way:

```
rk3399$ ./build.sh lunch
processing option: lunch

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

0. default BoardConfig.mk
1. BoardConfig-rk3399-evb-ind-lpddr4.mk
2. BoardConfig-rk3399-firefly.mk
3. BoardConfig-rk3399-sapphire-excavator-lp4.mk
4. BoardConfig-rk3399-sapphire-excavator.mk
5. BoardConfig.mk
Which would you like? [0]:
...
```

6.4 Configuring Different Components of startup/kernel/system in the SDK

The SDK can be configured for different components using `make menuconfig`, and the currently available components are mainly as follows:

Note that after configuring menuconfig, you need to save the configuration with `make savedefconfig`.

```
(rk3399) SoC
    Rootfs  --->
    Loader (u-boot)  --->
    Kernel  --->
    Boot  --->
    Recovery (buildroot)  --->
    PCBA test (buildroot)  --->
    Security  --->
    Update (OTA and A/B)  --->
    Firmware  --->
    Extra partitions  --->
    Others configurations  --->
```

With the above configuration, different rootfs/loader/kernel configurations can be selected for various customize compilations. It also has a userful command line switching function.

> Note that after configuring menuconfig, you need to save the configuration with `make savedefconfig`

6.5 Automatic Build

Enter root directory of project directory and execute the following commands to automatically complete all build:

```
./build.sh all # Only build module code(u-Boot, kernel, Rootfs, Recovery)
               # Need to execute ./mkfirmware.sh again for firmware package

./build.sh     # Base on ./build.sh all
               # 1. Add firmware package ./mkfirmware.sh
               # 2. update.img package
               # 3. Save the patches of each module to the out directory
               # Note: ./build.sh  and  ./build.sh allsave command are the same
```

It is Buildroot by default, you can specify rootfs by setting the environment variable RK_ROOTFS_SYSTEM. There are two types of system for RK_ROOTFS_SYSTEM: buildroot and debian.

If you need debain, you can generate it with the following command:

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

> Note:
> Every time the SDK is updated, it is recommended to clean up the previous compiled products and run them directly `./build.sh cleanall`

6.6 Build and Package Each Module

6.6.1 U-boot Build

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

6.6.2 Kernel Build

- Method 1

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

- Method 2

```
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 rk3399-evb-ind-lpddr4-linux.img -j
or
make ARCH=arm64 rk3399-evb-ind-lpddr4-linux.img -j
```

- Method 3

```
cd kernel
export CROSS_COMPILE=aarch64-linux-gnu-
make ARCH=arm64 rockchip_linux_defconfig
make ARCH=arm64 rk3399-evb-ind-lpddr4-linux.img -j
or
make ARCH=arm64 rk3399-evb-ind-lpddr4-linux.img -j
```

6.6.3 Recovery Build

```
./build.sh recovery
```

Note: Recovery is a unnecessary function, some board configuration will not be set

6.6.4 Buildroot Build

Enter project root directory and run the following commands to automatically complete compiling and packaging of Rootfs.

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

After compilations, rootfs.ext4 is generated in Buildroot directory " output/rockchip_rk3399/images".

6.6.5 Debian Build

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

After compilation, generate linaro-rootfs. img in the Debian directory.

> Description：re-install the depend packages
> 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 Debian development documentation reference:

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

6.6.6 Yocto Build

Enter project root directory and execute the following commands to automatically complete compiling and packaging Rootfs.

EVB boards：

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

After compiling, rootfs.img is generated in yocto directory "/build/lastest".
The default login username is root.

Please refer to Rockchip Wiki for more detailed information of Yocto.

FAQ:

- If you encounter the following problem during above compiling:

```
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 tosetup-locale-python3.

6.6.7 Cross-Compilation

6.6.7.1 SDK Directory Built-in Cross-Compilation

The SDK prebuilts directory built-in cross-compilation are as follows:

Contents	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

6.6.7.2 Buildroot Built-in Cross-compilation

The configuration of different chips and target functions can be set through `source buildroot/envsetup.sh`

```
$ source buildroot/envsetup.sh
Top of tree: rk3399

Pick a board:
24. rockchip_rk3399
25. rockchip_rk3399_base
26. rockchip_rk3399_recovery


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

Default selection 24 `rockchip_rk3399`, then enter the Buildroot directory for RK3399 and start compiling the relevant modules.

For example, to compile the rockchip_test module, the commonly used compilation commands are as follows:

SDK#cd buildroot

- Enter buildroot

```
SDK$cd buildroot
```

- To build rockchip-test

```
buildroot$make rockchip-test
```

- Rebuild rockchip-test

```
buildroot$make rockchip-test-rebuild
```

- Remove rockchip-test

```
buildroot$make rockchip-test-dirclean
or
buildroot$rm -rf /buildroot/output/rockchip_rk3399/build/rockchip-test-master/
```

If you need to compile a single module or a third-party application, you need to configure the cross-compilation environment. For example, RK3399 its cross-compilation tool is located in the `buildroot/output/rockchip_rk3399/host/usr` directory, you need to set the bin/ directory of the tool and the `aarch64-buildroot-linux-gnu/bin/` directory as the environment variable, execute the script that automatically configures environment variables in the top-level directory::

```
source buildroot/envsetup.sh rockchip_rk3399
```

Enter the command to view:

```
cd buildroot/output/rockchip_rk3399/host/usr/bin
./aarch64-linux-gcc --version
```

The following information will be printed:

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

> Save to rootfs configuration file
> buildroot$ make update-defconfig

6.6.8 Firmware Package

After compiling various parts of Kernel/U-Boot/Recovery/Rootfs above, enter root directory of project directory and run the following command to automatically complete all firmware packaged into `output/firmware` directory:

Firmware generation:

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

7. Upgrade Introduciton

Interfaces layout of RK3399 excavator are showed as follows:

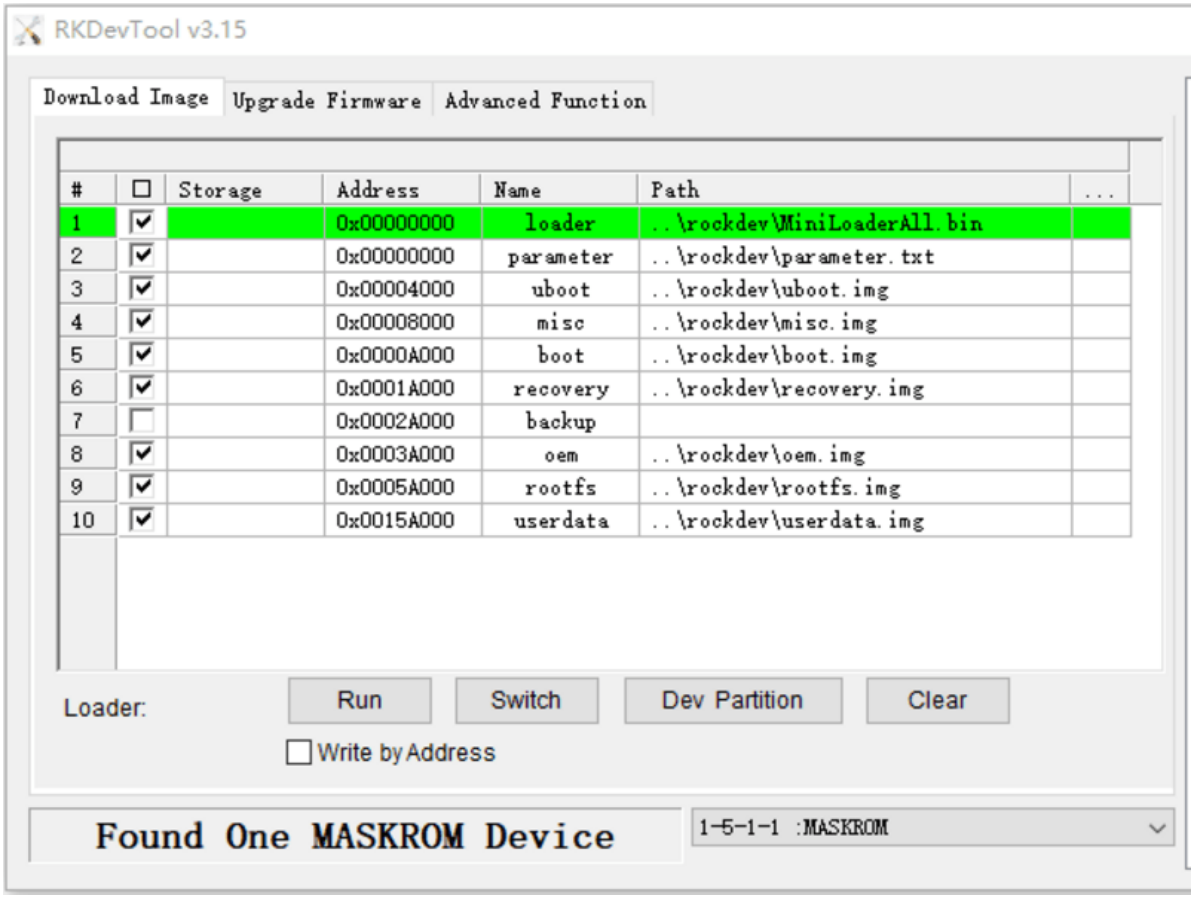
Interfaces layout of RK3399 IND industry board are showed as follows:

7.1 Windows Upgrade Introduction

SDK provides windows upgrade tool (this tool should be V3.15or later version) which is located in project root directory:

```
tools/
├── windows/RKDevTool
```

As shown below, after compiling the corresponding firmware, device should enter MASKROM or BootROM mode for update. After connecting USB cable, long press the button “MASKROM” and press reset button “RST” at the same time and then release, device will enter MASKROM Mode. Then you should load the paths of the corresponding images and click “Run” to start upgrade. You can also press the “recovery” button and press reset button “RST”then release to enter loader mode to upgrade. Partition offset and flashing files of MASKROM Mode are shown as follows (Note: Window PC needs to run the tool as an administrator):



Note：Before upgrade, please install the latest USB driver, which is in the below directory:

```
<SDK>/tools/windows/DriverAssitant_v5.12.zip
```

7.2 Linux Upgrade Instruction

The Linux upgrade tool (Linux_Upgrade_Tool should be v2.17 or later versions) is located in “tools/linux” directory. Please make sure your board is connected to MASKROM/loader rockusb, if the compiled firmware is in rockdev directory, upgrade commands are as below:

```
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 -trust rockdev/trust.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 whole update.img in the firmware

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

Or in root directory, run the following command on the machine to upgrade in MASKROM state:

```
./rkflash.sh
```

7.3 System Partition Introduction

Default partition introduction (below is RK3399 IND reference partition):

Number	Start (sector)	End (sector)	Size	Name
1	16384	24575	4096K	uboot
2	24576	32767	4096K	trust
3	32768	40959	4096K	misc
4	40960	106495	32M	boot
5	106496	303104	32M	recovery
6	172032	237567	32M	bakcup
7	237568	368639	64M	oem
8	368640	12951551	6144M	rootfs
9	12951552	30535646	8585M	userdata

- uboot partition: for uboot.img built from uboot.
- trust partition: for trust.img built from uboot.
- misc partition: for misc.img built from recovery.

- boot partition: for boot.img built from kernel.
- recovery partition: for recovery.img built from recovery.
- backup partition: reserved, temporarily useless. Will be used for backup of recovery as in Android in future.
- oem partition: used by manufactor to store their APP or data, mounted in /oem directory
- rootfs partition: store rootfs.img built from buildroot or debian.
- userdata partition: store files temporarily generated by APP or for users, mounted in /userdata directory

8. RK3399 SDK Firmware

Click here