Rockchip RK3399 Linux SDK Quick Start

ID: RK-FB-YF-944

Release Version: V1.0.2

Release Date: 2022-11-20

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

Contents

1. Set up an Development Environment

It is recommended to use Ubuntu 20.04 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

Software requirements: Ubuntu 20.04 system:

Please install software packages with below commands to setup SDK compiling environment:

```
sudo apt-get install git ssh make gcc libssl-dev liblz4-tool expect \
     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 g++-aarch64-linux-gnu
```

It is recommended to use Ubuntu 20.04 system or higher version for development. If you encounter an error during compilation, you can check the error message and install the corresponding software packages accordingly.

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 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）

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_CN.pdf” with the SDK, please refer to the documents under the project's docs/ 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".

2.3 Debian Development Guide

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

2.5 Software Update History

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

```
.repo/manifests$ realpath rk3399_linux_release.xml
# e.g.:printf version v2.9.0, update time on 20220620
# <SDK>/.repo/manifests/rk3399_linux_release_v2.9.0_20220620.xml
```

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

```
.repo/manifests/rk3399_linux$ cat RK3399_Linux5.10_SDK_Note.md
```

Or refer to the project directory:

```
<SDK>/docs/RK3399/RK3399_Linux5.10_SDK_Note.md
```

3. Hardware Development Guide

Please refer to user guides in the project directory for hardware development:

RK3399 excavator hardware development guide:

```
<SDK>/docs/RK3399/Rockchip_RK3399_User_Manual_Sapphire_EVB_V3.0_CN.pdf
```

RK3399 IND industry board hardware development guide:

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

4. The Precaution of IO Power Design

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.

Please refer to the following documents for details:

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

5. SDK Project Directory Introduction

There are buildroot, debian, recovery, app, kernel, u-boot, device, docs, external and other directories in the project directory. Each directory or its sub-directories will correspond to a git project, and the commit should be done in the respective directory.

- app: store application APPs with Demo.
- 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.
- IMAGE: stores building time, XML, patch and firmware directory for each building.
- external: stores some third-party libraries, including audio, video, network, recovery and so on.
- kernel: stores kernel4.4/4.19/5.10 development code.
- 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 4.1.

6. SDK Building Introduction

6.1 SDK Board Level Configuration

Enter the project SDK/device/rockchip/rk3399 directory:

Board level configuration	Note
BoardConfig-rk3399-evb-ind-lpddr4.mk	Suitable for RK3399 industry development board
BoardConfig-rk3399-firefly.mk	Suitable for RK3399 firefly development boards
BoardConfig-rk3399-sapphire-excavator-lp4.mk	Suitable for RK3399 sapphire excavator with LPDDR4 development board
BoardConfig-rk3399-sapphire-excavator.mk	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/BoardConfig-rk3399-evb-ind-lpddr4.mk
```

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

```
./build.sh device/rockchip/rk3399/BoardConfig-rk3399-firefly.mk
```

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

```
./build.sh device/rockchip/rk3399/BoardConfig-rk3399-sapphire-excavator-lp4.mk
```

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

```
./build.sh device/rockchip/rk3399/BoardConfig-rk3399-sapphire-excavator.mk
```

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]:
...
```

6.2 Compilation Commands

Execute the command in the root directory: `./build.sh -h|help`

```
rk3399$ ./build.sh -h
sage: build.sh [OPTIONS]
Available options:
BoardConfig*.mk    -switch to specified board config
lunch              -list current SDK boards and switch to specified board config
wifibt             -build wifibt
uboot              -build uboot
uefi               -build uefi
spl                -build spl
loader             -build loader
kernel             -build kernel
modules            -build kernel modules
toolchain          -build toolchain
rootfs             -build default rootfs, currently build buildroot as default
buildroot          -build buildroot rootfs
ramboot            -build ramboot image
multi-npu_boot     -build boot image for multi-npu board
yocto              -build yocto rootfs
debian             -build debian rootfs
pcba               -build pcba
recovery           -build recovery
all                -build uboot, kernel, rootfs, recovery image
cleanall           -clean uboot, kernel, rootfs, recovery
firmware           -pack all the image we need to boot up system
updateimg          -pack update image
otapackage         -pack ab update otapackage image (update_ota.img)
sdpackage          -pack update sdcard package image (update_sdcard.img)
save               -save images, patches, commands used to debug
allsave            -build all & firmware & updateimg & save
check              -check the environment of building
info               -see the current board building information
app/<pkg>          -build packages in the dir of app/*
external/<pkg>     -build packages in the dir of external/*

createkeys         -create secureboot root keys
security_rootfs    -build rootfs and some relevant images with security paramter
(just for dm-v)
security_boot      -build boot with security paramter
security_uboot     -build uboot with security paramter
security_recovery  -build recovery with security paramter
security_check     -check security paramter if it's good
```

```
Default option is 'allsave'.
```

View detailed build commands for some modules, for example: `./build.sh -h kernel`

```
rk3399$ ./build.sh -h kernel
###Current SDK Default [ kernel ] Build Command###
cd kernel
make ARCH=arm64 rockchip_linux_defconfig
make ARCH=arm64 rk3399-evb-ind-lpddr4-linux.img -j12
```

6.3 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. Copy the firmware in the rockdev directory to the
IMAGE/***_RELEASE_TEST/IMAGES directory
               # 4. Save the patches of each module to the
IMAGE/***_RELEASE_TEST/PATCHES 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 three types of system for RK_ROOTFS_SYSTEM: buildroot, Debian, and yocto.

For example, if you need debain, you can generate it with the following command:

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

6.4 Build and package each module

6.4.1 U-boot Build

```
### U-Boot build command
./build.sh uboot

### To view the detailed U-Boot build command
./build.sh -h uboot
```

6.4.2 Kernel Build

```
### Kernel build command
./build.sh kernel

### To view the detailed Kernel build command
./build.sh -h kernel
```

6.4.3 Recovery Build

```
### Recovery build command
./build.sh recovery

### To view the detailed Recovery build command
./build.sh -h recovery
```

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

6.4.4 Buildroot Build

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

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

After build, rootfs.ext4 is generated in Buildroot directory “output/rockchip_chipset/images”.

6.4.4.1 Buildroot Cross Compilation

If you need to build a single module or a third-party application, you need to setup the cross compilation environment. Such as RK3399 EVB, Cross compilation tool is located in “buildroot/output/rockchip_rk3399/host/usr” directory. You need to set bin/ directory of tools and aarch64-buildroot-linux-gnu/bin/ directory to environment variables, and execute auto-configuration environment variable script in the top-level directory (only valid for current console):

```
source envsetup.sh
```

Enter the command to check:

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

Then the following logs are printed:

```
aarch64-linux-gcc.br_real (Buildroot linux-5.10-gen-rkr2-297-ge5de4b00a7) 11.3.0
```

6.4.4.2 Build Modules in Buildroot

For example, for the rockchip-test module, commonly used build commands are as follows:

- Build ockchip-test

```
SDK$make rockchip-test
```

- Rebuild rockchip_test

```
SDK$make rockchip-test-rebuild
```

- delete rockchip_test

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

6.4.5 Debian Building

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

Or enter debian/ directory:

```
cd debian/
```

Please refer to the readme.md in the directory for further building and Debian firmware generation.

(1) Building base Debian system

```
sudo apt-get install binfmt-support qemu-user-static live-build
sudo dpkg -i ubuntu-build-service/packages/*
sudo apt-get install -f
```

Build 64 bit Debian:

```
RELEASE=buster TARGET=desktop ARCH=arm64 ./mk-base-debian.sh
```

After building, linaro-buster-alip-xxxxx-1.tar.gz (xxxxx is timestamp generated) will be generated in “debian/”:

FAQ:

- If you encounter the following problem during above building:

```
noexec or nodev issue /usr/share/debootstrap/functions: line 1450:
..../rootfs/ubuntu-build-service/buster-desktop-arm64/chroot/test-dev-null:
Permission denied E: Cannot install into target '/rootfs/ubuntu-build-
service/buster-desktop-arm64/chroot' mounted with noexec or nodev
```

Solution：

```
mount -o remount,exec,dev xxx (project directory) yyy(mount point), and then
rebuild
```

In addition, if there are other building issues, please check firstly that the building system is not ext2/ext4.

- Because building Base Debian requires to access to foreign websites, and when domestic networks access foreign websites, download failures often occur:

The live build is used in Debian10, you can configure like below to change the image source to domestic:

```
+++ b/ubuntu-build-service/buster-desktop-arm64/configure
@@ -11,6 +11,11 @@ set -e
 echo "I: create configuration"
 export LB_BOOTSTRAP_INCLUDE="apt-transport-https gnupg"
 lb config \
+ --mirror-bootstrap "https://mirrors.tuna.tsinghua.edu.cn/debian" \
+ --mirror-chroot "https://mirrors.tuna.tsinghua.edu.cn/debian" \
+ --mirror-chroot-security "https://mirrors.tuna.tsinghua.edu.cn/debian-security"
\
+ --mirror-binary "https://mirrors.tuna.tsinghua.edu.cn/debian" \
+ --mirror-binary-security "https://mirrors.tuna.tsinghua.edu.cn/debian-security"
   --apt-indices false \
   --apt-recommends false \
   --apt-secure false \
```

If the package cannot be downloaded for other network reasons, there are pre-build packages shared on Baidu Cloud Disk, put it in the current directory, and then do the next step directly.

(2) Building rk-debian rootfs

Build 64bit Debian：

```
VERSION=debug ARCH=arm64 ./mk-rootfs-buster.sh
```

(3) Creating the ext4 image(linaro-rootfs.img)

```
./mk-image.sh
```

The linaro-rootfs.img will be generated.

6.4.6 Yocto Build

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

RK3399 EVB boards：

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

After compiling, rootfs.img is generated in yocto directory “/build/lastest”.

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.The image generated after compiling is in “yocto/build/lastest/rootfs.img”. The default login username is root.

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

6.4.7 Firmware Package

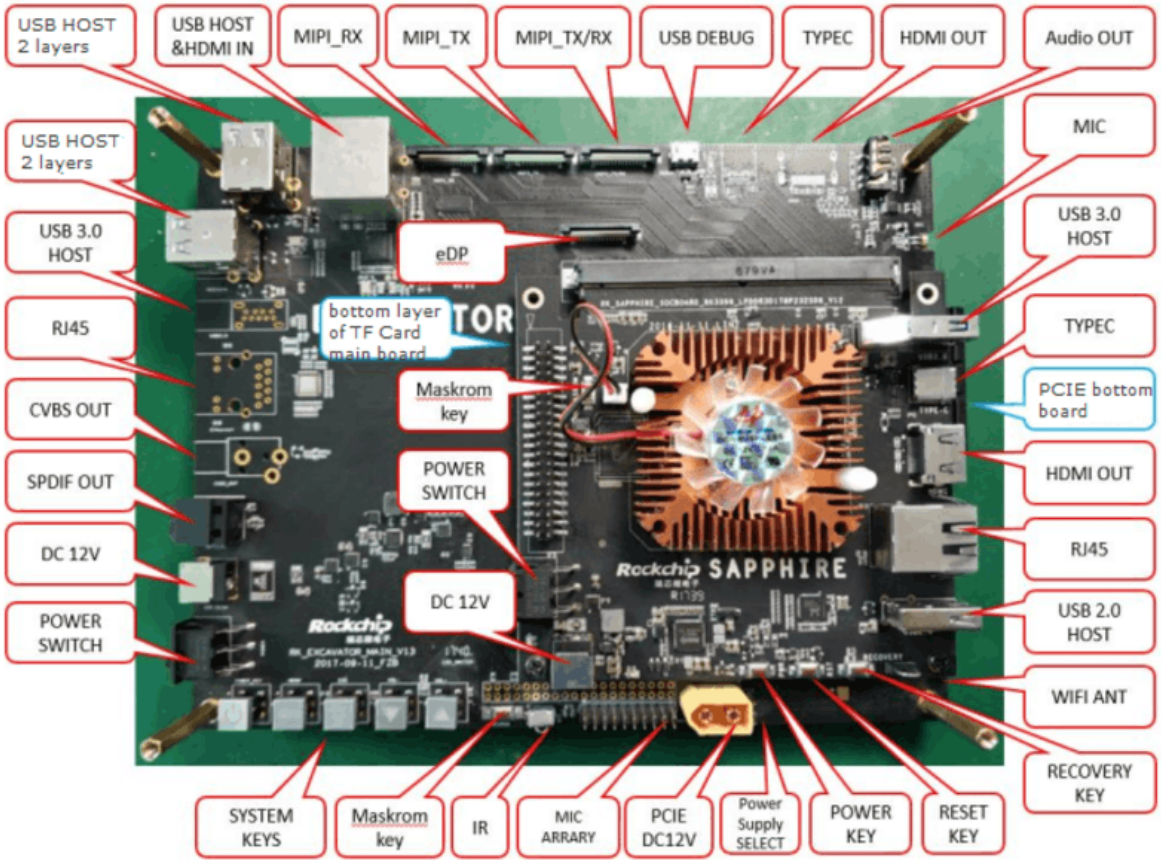
After compiling various parts of Kernel/U-Boot/Recovery/Rootfs above, enter root directory of project directory and run the following command to automatically complete all firmware packaged into rockdev directory:

Firmware generation:

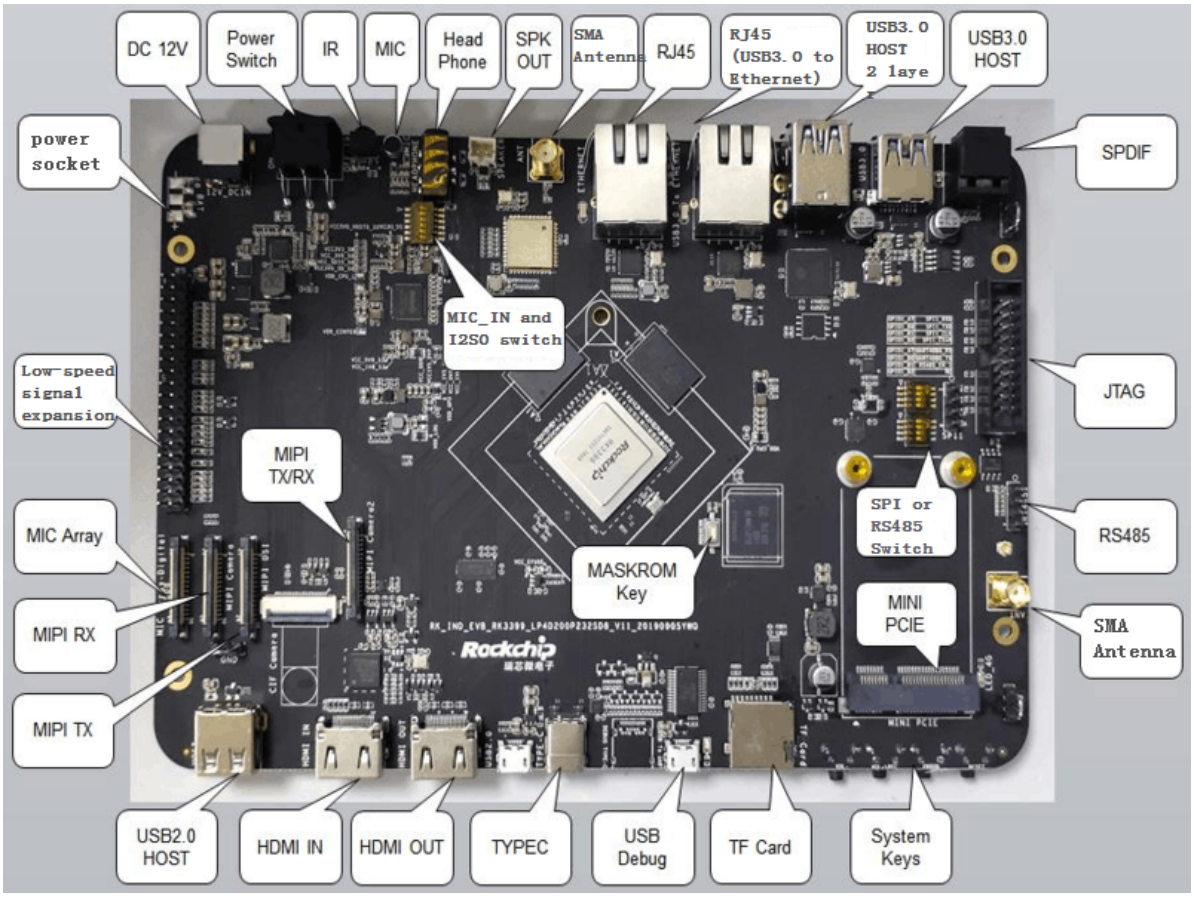
```
./mkfirmware.sh
```

7. Upgrade Introduciton

Interfaces layout of RK3399 excavator are showed as follows:



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

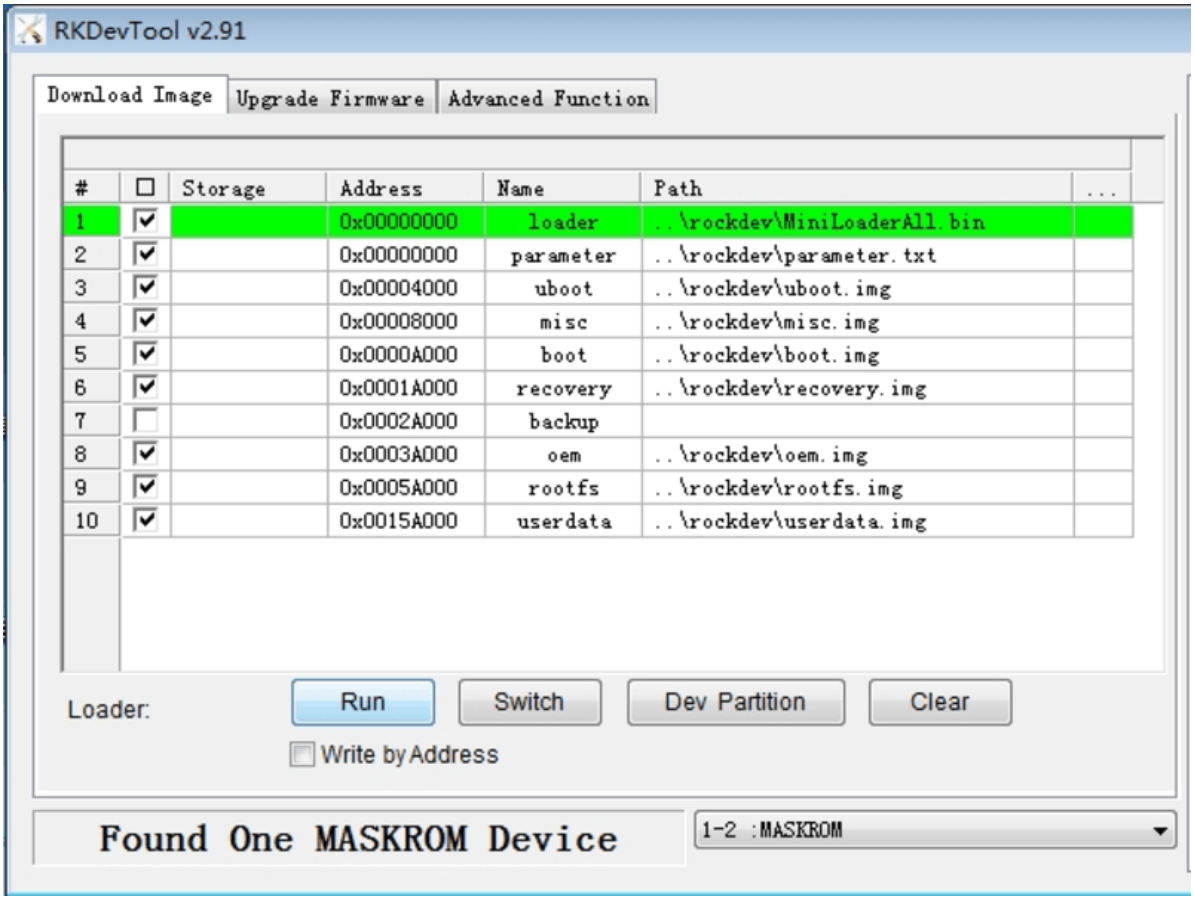


7.1 Windows Upgrade Introduction

SDK provides windows upgrade tool (this tool should be V2.91 or 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_v5.11.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 -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

- Baidu Cloud Disk

Buildroot

Debian

Yocto

- Microsoft OneDriver

Buildroot

Debian

Yocto