STAMP-RK3506-Kit

User Manual

Version: V1.0

June 2025

Company Profile

Zhejiang Qiyang Intelligent Technology Co., Ltd. was founded in Hangzhou in 2007, is a national high-tech enterprise focusing on the development, production and sales of ARM embedded products. More than 10 years of accumulation and precipitation, successfully built a product development to mass production service chain.

As the core of the company, Qiyang R&D team consists of more than 30 embedded engineers, dedicated to providing users with easy-to-use embedded hardware, software tools and customized product solutions. It has been widely used in industrial control, Internet of Things, new retail, medical, electric power, environmental monitoring, charging pile and other fields.

The production base in Zhuji provides a strong guarantee for Qiyang, covering an area of 5,000 square meters, with 2 SMT production lines, through and strictly follow the ISO9001 quality management system certification to guide production. Relying on the strong production strength, the annual production capacity can reach 1 million sets to ensure the delivery time of users and solve the worries.

Qiyang has a perfect sales and marketing network, professional sales and after-sales team to provide users with a full range of technical support and services. Business has spread to more than 120 countries and regions, successfully helping more than 2000 users to bring their products to market quickly and efficiently.

The combination and extension of R&D, production capacity and market has laid a solid foundation for Qiyang Intelligence to become a professional and global supplier of embedded software and hardware.

We offer:

- **Multi-platform software/hardware products**

NXP, Rockchip, MTK, Renesas, TI, Atmel, Cirrus Logic and other multi-platform ARM development boards/core boards/industrial control boards and peripheral hardware products, as well as supporting tools and software resources to support rapid secondary development of users.

- **Customized Services**

We fully utilize our accumulated technology on ARM platform and Linux, Android, Ubuntu and Debian operating systems to provide customized embedded product services (OEM/ODM).

Thank you for using Qiyang's products, we will do our best to provide you with technical assistance! Wish you good luck in your work!

Technical Support

If you have questions about the documents, you can contact us during office hours (Monday to Friday 8:30-12:00, 13:30-17:30)

Contact us at

Technical Email：supports@qiyangtech.com

Technical Support Phone：+86-0571-87858811-805

Official website：www.qiytech.com(Chinese）/www.qiyangtech.com(English）

Information updates

1. Information updates

Product-related manuals and datasheets are constantly being improved and updated; please ensure that they are up-to-date when you use them.

2. Update notice

Qiyang's newest product information and news updates will be released through the WeChat official account, please pay attention!

3. How to get information

After purchase, please contact the relevant sales staff to obtain the SDK.

4. Provided materials

Software: factory image, related kernel source code, interface test source code, cross compiler

Hardware: corresponding baseboard schematic, PCB source file (Allegro16.6)

Files: hardware manual, test manual, user manual, environment construction manual, IO pin comparison table, core board, baseboard structure dimension drawing (dxf), chip information.

Usage suggestions:

1) before using the development board, be sure to read the hardware manual first;

2) please check the packing list carefully before use and check whether there are any missing files;

3) understand the basic structure and composition of the development kit, including the allocation of hardware resources, the definition of each pin of the core board and the carrier board, and the definition of the expansion pins, etc.;

4) if you need to burn an image, test the development kit functions, you also need to read the user manual and test manual;

5) we accept bulk orders for the STAMP-RK3506-Kit development kit.

Version Record

Version No.	Hardware Platform	Date	Description	Revised by
V1.0	STAMP-RK3506-MB-BETA-V1_00 STAMP-RK3506-CM-BETA-V1_00	2025-06	Initial version	Maoh

Contents

NOTE: The manual mainly introduces the compilation of source code and the burning of images of the STAMP-RK3506-Kit development kit

I. Preparation

- Prepare a PC with Windows 10 installed.
- A virtual machine should be installed on the PC, and a Linux system (Ubuntu or other Linux distribution) should be installed in the virtual machine, or another PC with a Linux system should be installed. If it is not installed, you can refer to the "Guide to Installing Ubuntu 20.04 in a Virtual Machine" provided for installation. This manual takes the operation of Ubuntu 20.04 installed in a virtual machine under Windows 10 as an example.
- Serial port connection: connect the development board's DEBUG serial port (J2) to the PC's USB through a Type-C cable (the Debug serial port requires the PC host to support the CH340 driver).
- Network connection: connect the Ethernet interface (J6) of the development board to the network interface of the PC through a network cable, or connect through a switch.
- USB connection: connect the USB Device (J3) of the development board to the USB port of the PC via a USB cable.
- Serial port settings: open the terminal communication software

under Windows or the SecureCRT serial terminal software provided in the tool directory of the CD-ROM, select the serial port used to connect to the development board and set the following parameters: baud rate (115200), data bit (8 bits), stop bit (1 bit), check bit (none), data flow control (none).

II. System Image Burning (firmware burning)

Note: the product from normal shipments has the image burned according to the order requirements.

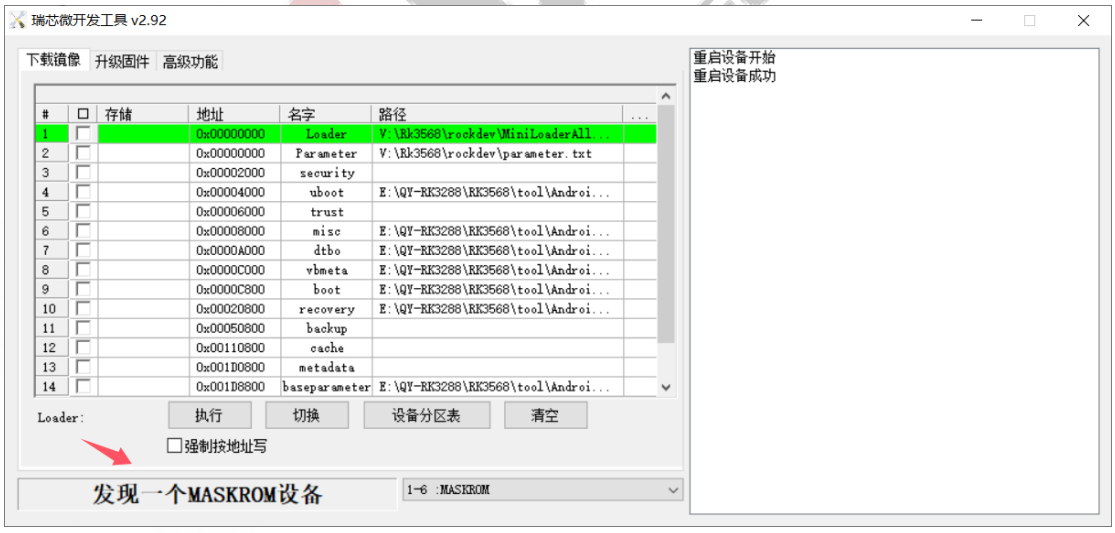
2.1. Working Mode

STAMP-RK3506-Kit has three modes: normal boot mode, Maskrom mode and Loader mode. Maskrom mode and Loader mode are both image burning modes, but their application scenarios are different. The details are as follows:

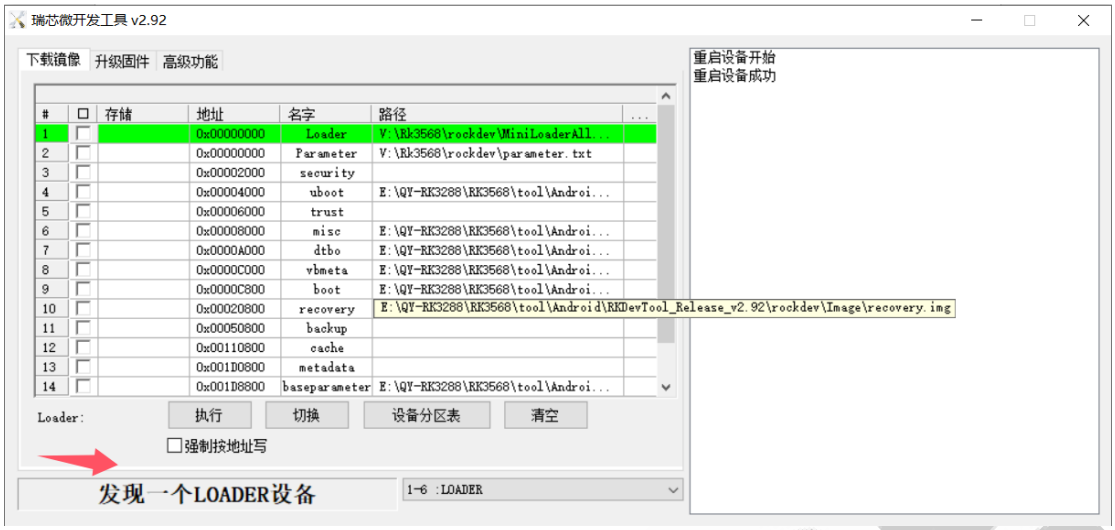
Normal startup mode: the development board starts normally and enters this mode by default. The serial port log is printed normally. The serial port log is as follows:

```
         Starting Advanced IEEE 802_/WPA/WPA2/EAP Authenticator...
[FAILED] Failed to start Advanced I_1X/WPA/WPA2/EAP Authenticator.
See 'systemctl status hostapd.service' for details.
         Starting Bluetooth service...
[  OK  ] Started Bluetooth service.
[   11.626449] ttyFIQ ttyFIQ0: tty_port_close_start: tty->count = 1 port count = 2

Debian GNU/Linux 10 linaro-alip ttyFIQ0

linaro-alip login: root (automatic login)

Last login: Wed Aug 24 08:19:43 UTC 2022 on ttyFIQ0
Linux linaro-alip 4.19.219 #207 SMP Wed Aug 24 10:13:56 CST 2022 aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@linaro-alip:~# [   13.102317] EXT4-fs (mmcblk0p8): mounting ext2 file system using t
he ext4 subsystem
[   13.104008] EXT4-fs (mmcblk0p8): warning: mounting unchecked fs, running e2fsck is rec
ommended
[   13.104751] EXT4-fs (mmcblk0p8): mounted filesystem without journal. Opts: (null)
[   13.281045] EXT4-fs (mmcblk0p7): mounting ext2 file system using the ext4 subsystem
[   13.283373] EXT4-fs (mmcblk0p7): warning: mounting unchecked fs, running e2fsck is rec
ommended
[   13.284119] EXT4-fs (mmcblk0p7): mounted filesystem without journal. Opts: (null)

root@linaro-alip:~#
```

Maskrom mode: if there is no image in the core board, or after erasing the flash, or the development board cannot be started normally and forced to enter, it will enter Maskrom mode, and the burning software will display as follows:



Loader mode: the core board starts normally. When debugging the kernel, re-burn the complete image or partition image to enter this mode. The burning software is displayed as follows:



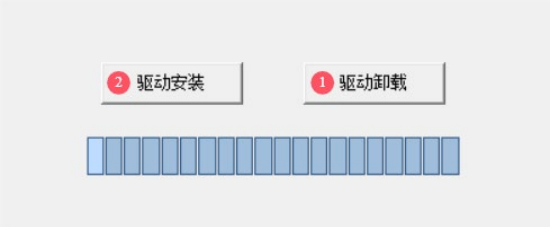
2.2. Image Description

There are two types of images, one is to package all partition images into a complete image package, and the other is a separate image package for burning the corresponding partition image during debugging. Their names and contents are different. The default complete image package is much larger, and the default packaged image package name is **update.img**, while the separate image package is mostly opposite to the corresponding partition image name, such as the kernel image package name is **boot.img**, and the size is relatively small.

2.3. Image Burning

1. Install RK USB Driver:

Unzip the **Rockchip_DriverAssitant** compressed package provided by the network disk to get an **exe** file, double-click to open it. First select Uninstall the driver, then select Install the driver, as shown in the figure below:



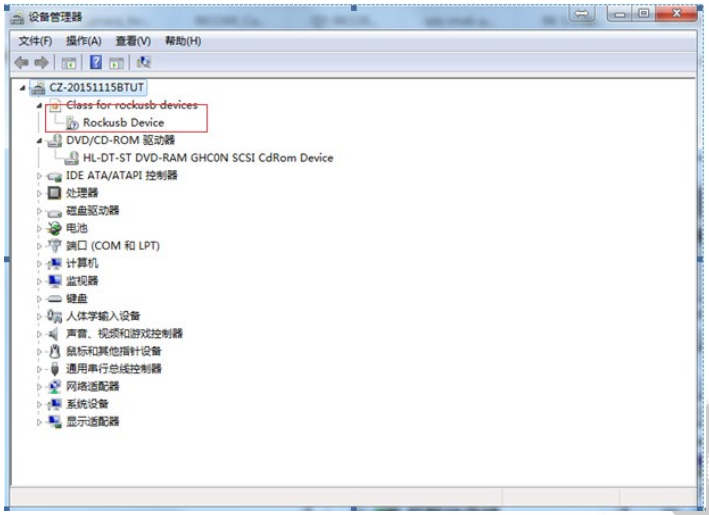
2. Enter Loader Mode：

Make sure that the Type-C data cable is connected to the STAMP-RK3506-Kit programming port (J3).

Method 1 (hardware): press and hold **Recovery** (SW3) and short-press **Reset** (SW1) at the same time. Release **Recovery** (SW3) after about 2 seconds.

The second method (software): Enter the **reboot loader** command in the serial terminal or use the switch button in **RKDevTool** to switch to **loader** mode.

After successfully entering Loader mode, you can see the detected USB device in the PC device manager, as shown below:

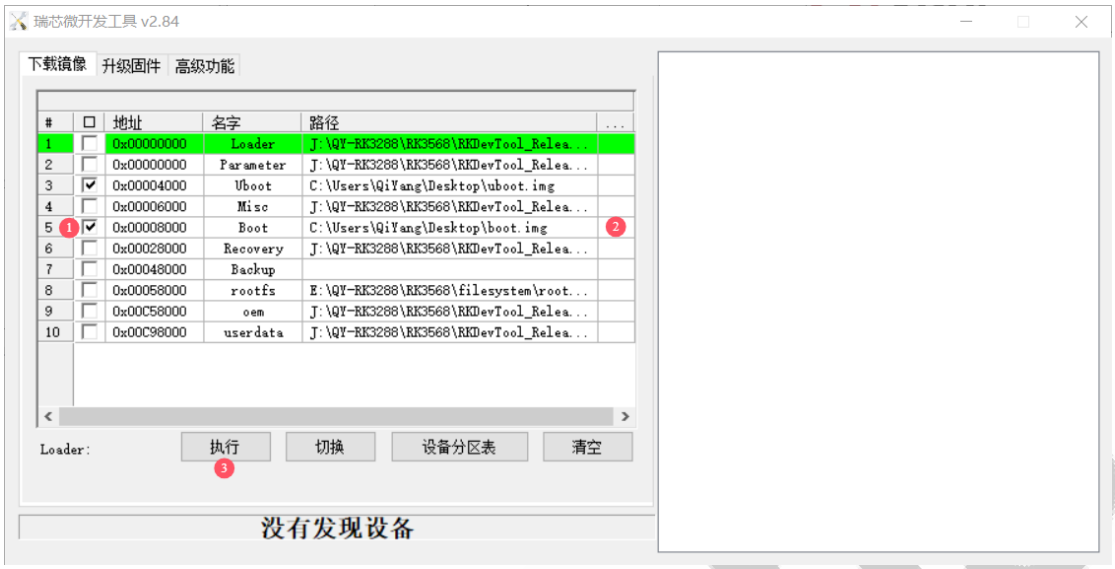


3. Burn the image

Burn the whole package: unzip the **RKDevTool_Release** compressed package on the network disk, double-click the compressed package **RKDevTool.exe** file, and click Upgrade Firmware->Firmware->Select the complete image package->Upgrade.

Partition burning: unzip the **RKDevTool_Release** compressed package on the network disk, double-click the compressed package **RKDevTool.exe** file, click the option box of the partition you want to burn

-> load the partition firmware in the corresponding blank area -> execute.



4. Erase Flash

This function is only used when Linux is used to flash Android, or when the entire flash needs to be erased and the image needs to be re-burned. The steps are as follows:

Enter **Loader** mode, then click **RKDevTool.exe**'s Upgrade Firmware -> Erase Flash. After erasing, it will enter Maskrom mode by default.

III. Linux Source Code Compilation

3.1. Building a PC Host Environment

For this section, please refer to the Ubuntu installation manual provided by the network disk.

3.2. Necessary Libraries Installation

After the PC host environment is built, you need to install the necessary libraries. (The following libraries are some common libraries. You can continue to install other libraries according to the errors reported during the compilation process)

sudo apt-get install repo 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 python-pip ncurses-dev pyelftools

3.3. SDK Compiler Introduction

The SDK directory contains the **u-boot kernel** compiler and th e **buildroot** compiler. The two are compiled using different compile rs. The path of **u-boot** and **kernel** using the same compiler is as f ollows: prebuilts/gcc/linux-x86/arm/gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabihf/bin/arm-none-linux-gnueabihf- . **Buildroot** uses the compiler generated by **buildroot**, so it will be generated after b uildroot is compiled. The path is as follows: buildroot/output/rockchi

p_rk3506/host/bin/arm-buildroot-linux-gnueabihf-gcc

3.4. SDK Compilation (Buildroot is compiled by default)

Simply execute **./build.sh** in the SDK top-level directory (be sure not to use root privileges), and relevant configuration options will pop up (select **eMMC** or **Norflash** according to the startup configuration). The default selection is **rockchip_rk3506_b_qiyang_emmc_defconfig**, as shown in the picture below:

```
chengsj@u20:~/Rk3506$ ./build.sh lunch

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

Manifest: STAMP-RK3506B-V1_00.xml

Log colors: message notice warning error fatal

Log saved at /home/chengsj/Rk3506/output/sessions/2025-06-18_07-58-11
Pick a defconfig:

1. rockchip_rk3502_robot_defconfig
2. rockchip_rk3506_b_evb1_defconfig
3. rockchip_rk3506_b_qiyang_NOR_Flash_defconfig
4. rockchip_rk3506_b_qiyang_emmc_defconfig
5. rockchip_rk3506_g_demo_defconfig
6. rockchip_rk3506_g_evb1_amp_defconfig
7. rockchip_rk3506_g_evb1_defconfig
8. rockchip_rk3506_g_evb1_smp_defconfig
Which would you like? [1]:
```

After the normal compilation is completed, a **rockdev** folder will be generated in the SDK top-level directory, which contains the image files shown in the following picture:

```
chengsj@u20:~/Rk3506/rockdev$ ls
boot.img  MiniLoaderAll.bin  misc.img  oem.img  parameter.txt  recovery.img  rootfs.img  uboot.img  update.img  userdata.img
chengsj@u20:~/Rk3506/rockdev$
```

Here is a screenshot of a successful compilation:

```
Packing /home/chengsj/Rk3506/output/firmware/update.img for update...
Android Firmware Package Tool v2.27
------ PACKAGE ------
Add file: ./package-file
package-file,Add file: ./package-file done,offset=0x800,size=0xe1,userspace=0x1
Add file: ./parameter.txt
parameter,Add file: ./parameter.txt done,offset=0x1000,size=0x1e9,userspace=0x1,flash_address=0x00000000
Add file: ./MiniLoaderAll.bin
bootloader,Add file: ./MiniLoaderAll.bin done,offset=0x1800,size=0x429c0,userspace=0x86
Add file: ./uboot.img
uboot,Add file: ./uboot.img done,offset=0x44800,size=0x400000,userspace=0x800,flash_address=0x00002000
Add file: ./misc.img
misc,Add file: ./misc.img done,offset=0x444800,size=0xc000,userspace=0x18,flash_address=0x00004000
Add file: ./recovery.img
recovery,Add file: ./recovery.img done,offset=0x450800,size=0xd72a00,userspace=0x1ae6,flash_address=0x00004800
Add file: ./boot.img
boot,Add file: ./boot.img done,offset=0x11c3800,size=0x4a6600,userspace=0x94d,flash_address=0x0002e800
Add file: ./oem.img
oem,Add file: ./oem.img done,offset=0x166a000,size=0x11ef000,userspace=0x23de,flash_address=0x0004e800
Add file: ./rootfs.img
rootfs,Add file: ./rootfs.img done,offset=0x2859000,size=0x8bf7000,userspace=0x117ee,flash_address=0x00058800
Add file: ./userdata.img
userdata,Add file: ./userdata.img done,offset=0xb450000,size=0x800000,userspace=0x1000,flash_address=0x00121200
Add CRC...
Make firmware OK!
------ OK ------
********rkImageMaker ver 2.23********
Generating new image, please wait...
Writing head info...
Writing boot file...
Writing firmware...
Generating MD5 data...
MD5 data generated successfully!
New image generated successfully!

Run 'make edit-package-file' if you want to change the package-file.

Running mk-updateimg.sh - build_updateimg succeeded.
Images under /home/chengsj/Rk3506/output/firmware/ are ready!
Running mk-firmware.sh - build_firmware succeeded.
Running 99-all.sh - build_all succeeded.
```

IV. Cross-compiling

4.1. Cross compiler Installation

Transfer the network disk\cross compiler to Ubuntu through **sftp** and other tools. After making sure the transmission is normal, decompress it through the **tar** command, as shown in the following picture:

```
ylook@ubuntu:~/Crosscompile$ tar -xvf arm-buildroot-linux-gnueabihf_sdk-buildroot.tar.gz
arm-buildroot-linux-gnueabihf_sdk-buildroot/
arm-buildroot-linux-gnueabihf_sdk-buildroot/lib64
arm-buildroot-linux-gnueabihf_sdk-buildroot/arm-buildroot-linux-gnueabihf/
arm-buildroot-linux-gnueabihf_sdk-buildroot/arm-buildroot-linux-gnueabihf/include/
arm-buildroot-linux-gnueabihf_sdk-buildroot/arm-buildroot-linux-gnueabihf/include/c++/
arm-buildroot-linux-gnueabihf_sdk-buildroot/arm-buildroot-linux-gnueabihf/include/c++/12.4.0/
arm-buildroot-linux-gnueabihf_sdk-buildroot/arm-buildroot-linux-gnueabihf/include/c++/12.4.0/latch
arm-buildroot-linux-gnueabihf_sdk-buildroot/arm-buildroot-linux-gnueabihf/include/c++/12.4.0/string
arm-buildroot-linux-gnueabihf_sdk-buildroot/arm-buildroot-linux-gnueabihf/include/c++/12.4.0/expected
```

After decompression, enter the decompressed directory and execute the **relocate-sdk.sh** script to convert the paths of the library files in the compiler to local paths, as shown in the following picture:

```
ylook@ubuntu:~/Crosscompile/arm-buildroot-linux-gnueabihf_sdk-buildroot$ ./relocate-sdk.sh
Relocating the buildroot SDK from /home/chengsj/Rk3506/buildroot/output/rockchip_rk3506/host to /home/ylook/Crosscompile/arm-buildroot-linux-gnueabihf_sdk-buildroot ...
ylook@ubuntu:~/Crosscompile/arm-buildroot-linux-gnueabihf_sdk-buildroot$
```

After waiting for a while, the path conversion is completed and the cross compiler is installed successfully.

4.2. How To Use the Cross Compiler

After the cross compiler is correctly installed, enable the cross compiler by declaring its environment variable **source enviroment-setup**, as shown in the following picture:

```
ylook@ubuntu:~/Crosscompile/arm-buildroot-linux-gnueabihf_sdk-buildroot$ source environment-setup
 _                 _ _     _                 _
| |__  _   _(_) | __| |_ __ ___   ___ | |_
| '_ \| | | | | |/ _` | '__/ _ \ / _ \| __|
| |_) | |_| | | | (_| | | | (_) | (_) | |_
|_.__/ \__,_|_|_|\__,_|_|  \___/ \___/ \__|

      Making embedded Linux easy!

Some tips:
* PATH now contains the SDK utilities
* Standard autotools variables (CC, LD, CFLAGS) are exported
* Kernel compilation variables (ARCH, CROSS_COMPILE, KERNELDIR) are exported
* To configure do "./configure $CONFIGURE_FLAGS" or use
  the "configure" alias
* To build CMake-based projects, use the "cmake" alias

ylook@ubuntu:~/Crosscompile/arm-buildroot-linux-gnueabihf_sdk-buildroot$
```

After the declaration is successful, use the **arm-buildroot-linux-gnueabihf-gcc -v** command to check whether the environment variables are normal, as shown in the following picture:

```
ylook@ubuntu:~/Crosscompile/arm-buildroot-linux-gnueabihf_sdk-buildroot$ arm-buildroot-linux-gnueabihf-gcc -v
Using built-in specs.
COLLECT_GCC=/home/ylook/Crosscompile/arm-buildroot-linux-gnueabihf_sdk-buildroot/bin/arm-buildroot-linux-gnueabihf-gcc.br_real
COLLECT_LTO_WRAPPER=/home/ylook/Crosscompile/arm-buildroot-linux-gnueabihf_sdk-buildroot/bin/../libexec/gcc/arm-buildroot-linux-gnueabihf/12
Target: arm-buildroot-linux-gnueabihf
Configured with: ./configure --prefix=/home/chengsj/Rk3506/buildroot/output/rockchip_rk3506/host --sysconfdir=/home/chengsj/Rk3506/buildroot
engsj/Rk3506/buildroot/output/rockchip_rk3506/host/arm-buildroot-linux-gnueabihf/sysroot --enable-__cxa_atexit --with-gnu-ld --disable-libss
droot/output/rockchip_rk3506/host --with-mpc=/home/chengsj/Rk3506/buildroot/output/rockchip_rk3506/host --with-mpfr=/home/chengsj/Rk3506/bui
buildroot.org/buildroot/-/issues --without-zstd --disable-libquadmath --disable-libquadmath-support --enable-tls --enable-threads --without-
mode=arm --enable-languages=c,c++ --with-build-time-tools=/home/chengsj/Rk3506/buildroot/output/rockchip_rk3506/host/arm-buildroot-linux-gnu
Thread model: posix
Supported LTO compression algorithms: zlib
gcc version 12.4.0 (Buildroot -g146c158ee8)
ylook@ubuntu:~/Crosscompile/arm-buildroot-linux-gnueabihf_sdk-buildroot$
```

At this point, the cross-compiler environment is normal, and you can use this compiler to cross-compile C applications and cross-compile third-party libraries, etc.

Zhejiang Qiyang Intelligent Technology Co., Ltd

Tel: +86-571-87858811 / 87858822

Fax: +86-571-89935912

Technical Support: +86-571-87858811 ext. 805

E-MAIL： supports@qiyangtech.com

Website: http://www.qiytech.com

ADD: 3rd Floor, Building A, WSCG Building, NO.6 Xiyuan 8th Road, Sandun Town, Xihu District, Hangzhou City, Zhejiang China

Postal Code: 310030