MYD-YT507H Linux System Development Guide

File Status : [ ] Draft [√] Release	FILE ID :	MYIR-MYD-YT507H-SW-DG-ZH-L4.9.170
	VERSION :	V1.2(DOC)
	AUTHOR :	Licy
	CREATED :	2022-03-30
	UPDATED :	2022-08-03

Revision History

VERSION	AUTHOR	PARTICIPANT	DATE	DESCRIPTION
V1.0	licy		20220330	Initial Version: u-boot2018.05, Linux Kernel 4.9.170，Buildroot 2019.02
V1.1	licy		20220707	Modified some SDK configurations
V1.2	licy		20220803	Added considerations and problem analysis for setting up a development environment. Delete Section 3.3.

CONTENT

1. Overview

There are many open source system build frameworks on the Linux system platform,these frameworks make it easy for developers to build and customize embedded systems,at present common similar software has Buildroot, buildroot, OpenEmbedded and so on. The buildroot project uses a more powerful and customized approach to build Linux systems that suitable for embedded products. buildroot is not only a file system manufacturing tool, but also provides a complete set of Linux-based development and maintenance workflow, so that the embedded developers of the Underlying Software and the High-Level Application can develop under a unified framework, which solves the fragmented and unmanaged development mode in the traditional development mode.

This document mainly introduces the complete process of customizing a complete embedded Linux system based on buildroot project, including the configuration of development environment, how to get the source code, how to port bootloader and kernel, and how to customize rootfs suitable for their own application requirements. First of all, we will introduce how to build a system image for MYD-YT507H development board based on the source code provided by us, and how to burn the prebuilt image to the development board. Then, we focus on the methods and key points of porting the system to the user's hardware platform. In addition, if you are developing a project based on MYC-YT507H CPU module ,we will also take some actual BSP porting cases and rootfs customization cases as examples to guide users to quickly customize the system image suitable for their own base-board hardware.

This document does not include the introduction of buildroot project and the basic knowledge of Linux system, and the user guide is suitable for embedded Linux development engineers with some development experience. For some specific functions that users may use in the process of secondary development, we also provide detailed application notes for reference,please refer to Table 2-4 of "*MYD-YT507H SDK Release Notes*" for the detailed list of documents.

1.1. Software Resources

MYD-YT507H series development board runs an operating system based on the Linux 4.9.170 kernel,which also provided a wealth of system resources and other software resources. ThThe development board is delivered with the cross-compilation tool chain, U-boot source code, Linux kernel and source code of each driver module and other information packages needed for embedded Linux system development. And for Windows desktop environment and PC Linux system (recommended ubuntu18.04 desktop 64bit or ubuntu18.04 server 64bit) various development and debugging tools, application development routines, etc.For specific software information, ,please refer to Table 2-4 of "*MYD-YT507H SDKRelease Notes*" for the detailed list of documents.

1.2. Document Resources

According to the different stages of using the development board, the SDK contains different types of documents and manuals, such as release notes, introduction guide, evaluation guide, development guide, application notes, frequently asked questions and answers, etc.For detailed document list, please refer to table 2-4 of "*MYD-YT507H SDK Release Notes*".

2. Development Environment

This chapter mainly introduces some software and hardware environment required in the development process, including the necessary development host environment, necessary software tools, code and resource acquisition, etc. the specific preparatory work will be described in detail below.

2.1. Developing Host Environment

This section describes how to deploy the T5 development environment.By reading this section, you will learn about the installation and use of hardware and software tools.And you can quickly deploy the relevant development environment and prepare for subsequent development and debugging.T5 series processor is a multi-core SMP processor.

- **Host Hardware**

To get the buildroot Project expected behavior in a Linux Host Machine, the packages and utilities described below must be installed. An important consideration is the hard disk space required in the host machine. It is recommended that at least 100 GB is provided, which is enough to compile all backends together.In addition, the processor with more than dual core CPU, 8GB memory or higher configuration will better meet the operation requirements. It can be the host with Linux system installed, virtual machine running Linux system or wsl2 under Windows system, etc.

- **Host Operating System**

There are many options for the host operating system used to build the buildroot project. Generally, we choose to build it on the local host with Fedora, openSUSE, Debian, Ubuntu, RHEL or Cent OS Linux distributions. Here, we recommend the Ubuntu 20.04 64bit desktop system, the subsequent development is also based on this system.

- **Prerequisite Package Installation**

```
sudo apt-get update
```

```
sudo apt-get install build-essential gcc libncurses5-dev bison flex texinfo
sudo apt-get install zlib1g-dev gettext libssl-dev autoconf
sudo apt-get install autoconf
sudo apt-get install automake
sudo apt-get install libtool
sudo apt-get install linux-libc-dev:i386
sudo apt-get install git
sudo apt-get install gnupg
sudo apt-get install flex
sudo apt-get install bison
sudo apt-get install gperf
sudo apt-get install build-essential
sudo apt-get install zip
sudo apt-get install curl
sudo apt-get install libc6-dev
sudo apt-get install libncurses5-dev:i386
sudo apt-get install x11proto-core-dev
sudo apt-get install libx11-dev:i386
sudo apt-get install libreadline6-dev:i386
sudo apt-get install libgl1-mesa-glx:i386
sudo apt-get install libgl1-mesa-dev
sudo apt-get install g++-multilib
sudo apt-get install mingw32
sudo apt-get install tofrodos
sudo apt-get install python-markdown
sudo apt-get install libxml2-utils
sudo apt-get install xsltproc
sudo apt-get install zlib1g-dev:i386
sudo apt-get install gawk
sudo apt-get install texinfo
sudo apt-get install gettext
```

The preceding packages need to be manually installed. If automatic installation is required, copy all commands to the sh script and run them in the decompressed package./build/config.sh.

Other configuration packages

```
sudo dpkg-reconfigure dash #slect no
sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1 /usr/lib/i386-linux-gnu/libGL.so
sudo apt-get install zlib1g-dev
sudo apt-get install uboot-mkimage
```

2.2. Introduction of Software Development Tools

In the customized Linux system and debugging process needs to use a lot of debugging, burning Tools, in the cd-rom image directory provided by MYIR 03-tools provides some Tools, a brief introduction as follows

- **Install driver**

In Windows system, USB device drivers will be automatically installed after the target board device is powered on and plugged in with a USB cable. If the installation is successful, Android Phone is displayed in the Windows Manager.

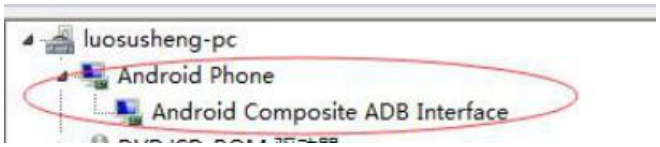

Figure 4-1. Connecting the USB to a PC

- **Burning software installation**

Burn “Phoenixsuit-v1.13” and “PhoenixCard4.2.4”, USB wire brush and SD card, respectively Brush. The installation packages for both software are located at Tools\Tools_win\. After decompression, there will be two versions in Chinese and English, choose one version. Once the SDK is compiled and packaged, it is ready to burn through the PhoenixSuit, as detailed below.

2.3. Install the cross-compile toolchain

In the process of building this system image using the SDK, you also need to install the cross toolchain. In addition to the various source code, the SDK provided by MYIR also provides the necessary cross toolchain, which can be directly used to compile the application, etc. Users can directly use the sub-cross-compilation tool chain to establish an independent development environment, independently compile the Bootloader, Kernel or compile their own applications, the detailed process will be described in the following sections. The installation steps of SDK are introduced as follows:

a. Copy the SDK package to the user's working directory in Host Ubuntu, such as $HOME/work/t507, decompress the file, and obtain the installation script file as follows:

```
PC$ cd $HOME/work/t507
PC$ tar -jxvf YT507H-buildroot-t5-4.9.170-X.X.X.tar.bz2
```

X.X.X Represents the current version number

b. Find build/ Toolchain/in the SDK directory

```
PC$ cd $HOME/work/t507/build/toolchain/
gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu.tar.xz
```

c. Decompress the package to the /opt directory on the host ubuntu

```
PC$ tar -xvf gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu.tar.xz -C /opt
```

d. Set the environment variables and test that the installation is complete

```
PC$ export PATH=$PATH:/opt/gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu/bin
PC$ aarch64-linux-gnu-gcc -v
used specs。
COLLECT_GCC=aarch64-linux-gnu-gcc
COLLECT_LTO_WRAPPER=/home/lcy/t507/out/gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu/bin/../libexec/gcc/aarch64-linux-gnu/7.4.1/lto-wrapper
target : aarch64-linux-gnu
```

```
config : '/home/tcwg-buildslave/workspace/tcwg-make-release_1/snapshots/gcc.git~linaro-7.4-2019.02/configure' SHELL=/bin/bash --with-mpc=/home/tcwg-buildslave/workspace/tcwg-make-release_1/_build/builds/destdir/x86_64-unknown-linux-gnu --with-mpfr=/home/tcwg-buildslave/workspace/tcwg-make-release_1/_build/builds/destdir/x86_64-unknown-linux-gnu --with-gmp=/home/tcwg-buildslave/workspace/tcwg-make-release_1/_build/builds/destdir/x86_64-unknown-linux-gnu --with-gnu-as --with-gnu-ld --disable-libmudflap --enable-lto --enable-shared --without-included-gettext --enable-nls --with-system-zlib --disable-sjlj-exceptions --enable-gnu-unique-object --enable-linker-build-id --disable-libstdcxx-pch --enable-c99 --enable-clocale=gnu --enable-libstdcxx-debug --enable-long-long --with-cloog=no --with-ppl=no --with-isl=no --disable-multilib --enable-fix-cortex-a53-835769 --enable-fix-cortex-a53-843419 --with-arch=armv8-a --enable-threads=posix --enable-multiarch --enable-libstdcxx-time=yes --enable-gnu-indirect-function --with-build-sysroot=/home/tcwg-buildslave/workspace/tcwg-make-release_1/_build/sysroots/aarch64-linux-gnu --with-sysroot=/home/tcwg-buildslave/workspace/tcwg-make-release_1/_build/builds/destdir/x86_64-unknown-linux-gnu/aarch64-linux-gnu/libc --enable-checking=release --disable-bootstrap --enable-languages=c,c++,fortran,lto --build=x86_64-unknown-linux-gnu --host=x86_64-unknown-linux-gnu --target=aarch64-linux-gnu --prefix=/home/tcwg-buildslave/workspace/tcwg-make-release_1/_build/builds/destdir/x86_64-unknown-linux-gnu
Pthread mode : posix
gcc version 7.4.1 20181213 [linaro-7.4-2019.02 revision 56ec6f6b99cc167ff0c2f8e1a2eed33b1edc85d4] (Linaro GCC 7.4-2019.02)
```

3. Build the File System with buildroot

3.1. Introduction Linux SDK

Linux SDK development package, which integrates BSP, build system, Linux application, test system, independent IP, tools and documentation, can be used as BSP, IP development, verification and release platform, as well as embedded Linux system.

Is the unified use of Linux development platform. It integrates BSP, build system, independent IP and test, and can be used as either a BSP development and IP verification platform, or as a mass-production embedded Linux system

The Functionality of the Linux SDK consists of the following four parts :

- BSP , include bootloader , uboot , kernel.
- Linux file system , Includes the embedded Linux root file system.
- IP's authentication and publishing platform includes GPU, Cedarx, Gstreamer, DRM/Weston, Security, and other proprietary packages. And the use of IP and system integration of the demo program, convenient for the third party to use quickly.
- Test, includes board level test and system test.

The 04_sources directory in the image provided by MYIR provides linuxSDK files and data suitable for MYD-YT507H development board, helping developers to build different types of Linux system images that can run on MYD-YT507H development board. For example, myir-image-full system image with Qt5.12.5 graphics library and myir-image-core system image without GUI interface, the following takes the construction of myir-image-full image as an example to introduce the specific development process, so as to lay a foundation for the subsequent customization of their own system image.

3.2. Get the Source Code

We provide two ways to obtain the source code. One is to obtain the compressed package directly from the 04-sources directory of the MYIR image, and the other is to use repo to obtain the source code updated in real time on GitHub for construction. Users can choose one of them.

Note:before building the buildroot system, all software packages in the file system need to be downloaded to the local. In order to build quickly, MYD-YT507H has packaged the relevant software, and users can directly unzip and copy it to the build directory, so as to reduce the repeated download time.

3.2.1. Get Compressed Source Code from CD Image

You can find the buildroot compressed source package in the development kit package *04-sources/*YT507H-buildroot-t5-4.9.170-X.X.X.tar.bz. Copy the compressed package to the user specified directory, such as the *$HOME/work/t507* directory. This directory will be used as the top-level directory for subsequent construction. After decompressing, the layers directory will appear:

```
PC$ cd $HOME/work/t507
PC$ tar -jxvf YT507H-buildroot-t5-4.9.170-X.X.X.tar.bz
brandy build buildroot build.sh device kernel out platform test tools
```

List the layers directory as follows:

```
PC$ tree -d -L 1 t507
t507
├── brandy
├── build
├── buildroot
├── device
├── kernel
├── out
├── platform
├── test
└── tools
```

```
9 directories
```

3.2.2. Get Source Code from GitHub

At present, the BSP source code and buildroot source code of MYD-YT507H development board are managed by GitHub and will be updated for a long time. Please refer to Section 2.2 of "MYD-YT507H SDK1.0.0 Release Notes". Users can use repo to get and synchronize the code on GitHub. The specific operation methods are as follows:

```
PC$ mkdir $HOME/work/t507
PC$ cd $HOME/work/t507
PC$ repo init -u https://github.com/MYiR-Dev/myir-t5-manifest.git --no-clone-bundle --depth=1 -b master -m myir-t5-4.9.170-1.0.0.xml
PC$ repo sync
```

After the synchronization code is completed, you will get a layers folder under the *$home/work/t507* directory, which contains the source code or source repository path related to myd-YT507H development board. The directory structure is the same as that extracted from the compressed package.

3.2.3. Linux SDK structure

```
├── brandy
├── build
├── buildroot
├── device
├── kernel
├── out
├── platform
├── test
└── tools
```

It is mainly composed of Brandy, Buildroot, Kernel and Platform.

- Brandy: include uboot2018 ;

- Buildroot: Responsible for ARM toolchain, application software package, Linux rootfs system generation.
- Kernel: linux kernel ;
- Platform: platform-specific libraries and SDK applications.

3.2.4. Buildroot Introduction

Buildroot is a set of Makefiles and patches that simplify and automate building complete, bootable Linux environments (including bootloaders, Linux kernels, file systems containing various apps) for embedded systems. Buildroot runs on Linux and can be used to build an embedded Linux platform for multiple target boards using cross-compilation tools. Buildroot can automatically build the required cross-compile toolchains, create root file systems, compile Linux kernel images, and generate boot loaders for target embedded systems, or it can perform any independent combination of these steps. For example, you can use an installed cross-compilation tool chain alone, while Buildroot creates only the root file system.

Buildroot https://buildroot.org/downloads/manual/manual.html

Buildroot source https://buildroot.org/downloads/

```
├── arch
├── board
├── boot
├── CHANGES
├── Config.in
├── configs
├── COPYING
├── dl
├── docs
├── external-packages
├── fs
├── linux
├── Makefile
├── support
```

```
├── package
├── README
├── scripts
├── target
├── support
├── system
├── utils
└── toolchain
```

The configs directory stores predefined configuration files, such as sun50iW9p1_longan_defconfig, dl directory stores downloaded software packages, scripts directory stores buildroot compiled scripts, mkcmd.sh, Mkcommon. sh, mkrule and mksetup.sh etc. The target directory, which houses the rule files used to generate the root file system, is important for code and tooling integration. The most important thing for us is the Package directory, which holds the generation rules for nearly 3,000 packages to which we can add our own packages or middleware.

More information about the buildroot, can go to the official website http://buildroot.uclibc.org/ for buildroot.

3.2.5. kernel

Linux kernel source directory. The current kernel version in use is Linux4.9.170. The above directory structure is consistent with the standard Linux kernel, except for the modules directory. The modules directory is where we store external modules that aren't integrated with the kernel's Menuconfig.

3.2.6. brandy

Brandy2.0 is available in the brandy catalog. Currently, T507 uses Brandy2.0

Structure as follows:

```
├── spl-pub
├── tools
```

```
└── u-boot-2018
```

NOTE:The uboot is not compiled in the default code compilation process. If you need to modify the uboot, you need to compile it. The specific compilation method will be introduced later.

3.2.7. platform

Platform private package directory.

```
platform/
├── apps
├── base
├── config
├── core
├── external
├── framework
└── tools
```

Framework/Auto includes SDK interfaces and examples for T5 Linux version.

```
platform/framework/auto/
├── rootfs
├── sdk_demo
└── sdk_lib
```

Rootfs forces the target of the out directory (the root file system directory of the machine) to be overridden each time the top layer executes build.sh.

3.2.8. tools

Compile and package tools

```
tools/
├── build
├── codecheck
├── pack
└── tools_win
```

3.2.9. Test system

Test is a testing system called Dragonboard. Dragonboard provides quick board-level testing.

3.2.10. device

```
├── config
│       ├── chips
│       │   └── t507
│       ├── common
│       │   ├── debug
│       │   ├── dtb
│       │   ├── hdcp
│       │   ├── imagecfg
│       │   ├── partition
│       │   ├── sign_config
│       │   ├── toc
│       │   ├── tools
│       │   └── version
│       └── rootfs_tar
├── bin
│       ├── bl31.bin
│       ├── boot0_nand_sun50iw9p1.bin
│       ├── boot0_sdcard_sun50iw9p1.bin
│           ├── fes1_sun50iw9p1.bin
│       ├── optee_sun50iw9p1.bin
│       ├── sboot_sun50iw9p1.bin
│       ├── u-boot-sun50iw9p1.bin
├── boot-resource
│       ├── boot-resource
│       │   ├── bat
│       │   ├── bootlogo.bmp
│       └── boot-resource.ini
├── configs
```

```
|       ├── default
|       |   ├── BoardConfig.mk
|       |   ├── boot_package.cfg
|       |   ├── diskfs.fex
|       |   ├── dragon_toc_android.cfg
|       |   ├── dragon_toc.cfg
|       |   ├── env_burn.cfg
|       |   ├── env.cfg
|       |   ├── env_dragon.cfg
|       |   ├── image.cfg
|       |   ├── image_dragonboard.cfg
```

4. How to Burn System Image

MYC-YT507H series core board and development board designed by MYIR Electronics Co., LTD is based on T5 series microprocessor of Allwinner Technology, which has various startup modes, so it needs different updating system tools and methods. Users can select different update methods according to their requirements. The update methods are as follows.

- PhoenixSuit burn: Applicable to r&d, commissioning, testing and other scenarios.
- Making SD card initiator: Suitable for r&d, debugging, and quick startup scenarios.
- SD card burner: suitable for mass production of eMMC。

4.1. PhoenixSuit burn

1). preparatory

- Development Board
- USB Type_C
- 12VPower adapter
- PhoenixSuit

2). Setup the development board

Select the startup mode and set the boot mode switch to the Download mode (S0/S1/S2/S3:1 0 1 1). Connect the hardware, connect the J6 OTG interface of the development board to the computer, and insert the power adapter.

3). Burn the system under Windows

Use OTG USB cable to connect the development board and the host. Press and hold FEL button first, then press the power button to reset the system, and release

FEL button about two seconds later. Open Windows Device Manager..as shown in the following page：

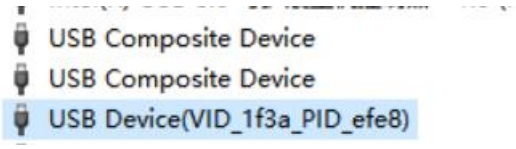

Figure 4-1. Connecting the USB to a PC

If the driver is not automatically installed, you can find the corresponding driver files (64bit usbdriver_64. zip and 32bit usbdriver.rar) in the SDK directory /tools/ Tools_win and decompress them to install the driver.

For the OTG full burn test, double-click the Phoenixsuit_v1.13 phoenixsuit.exe file.

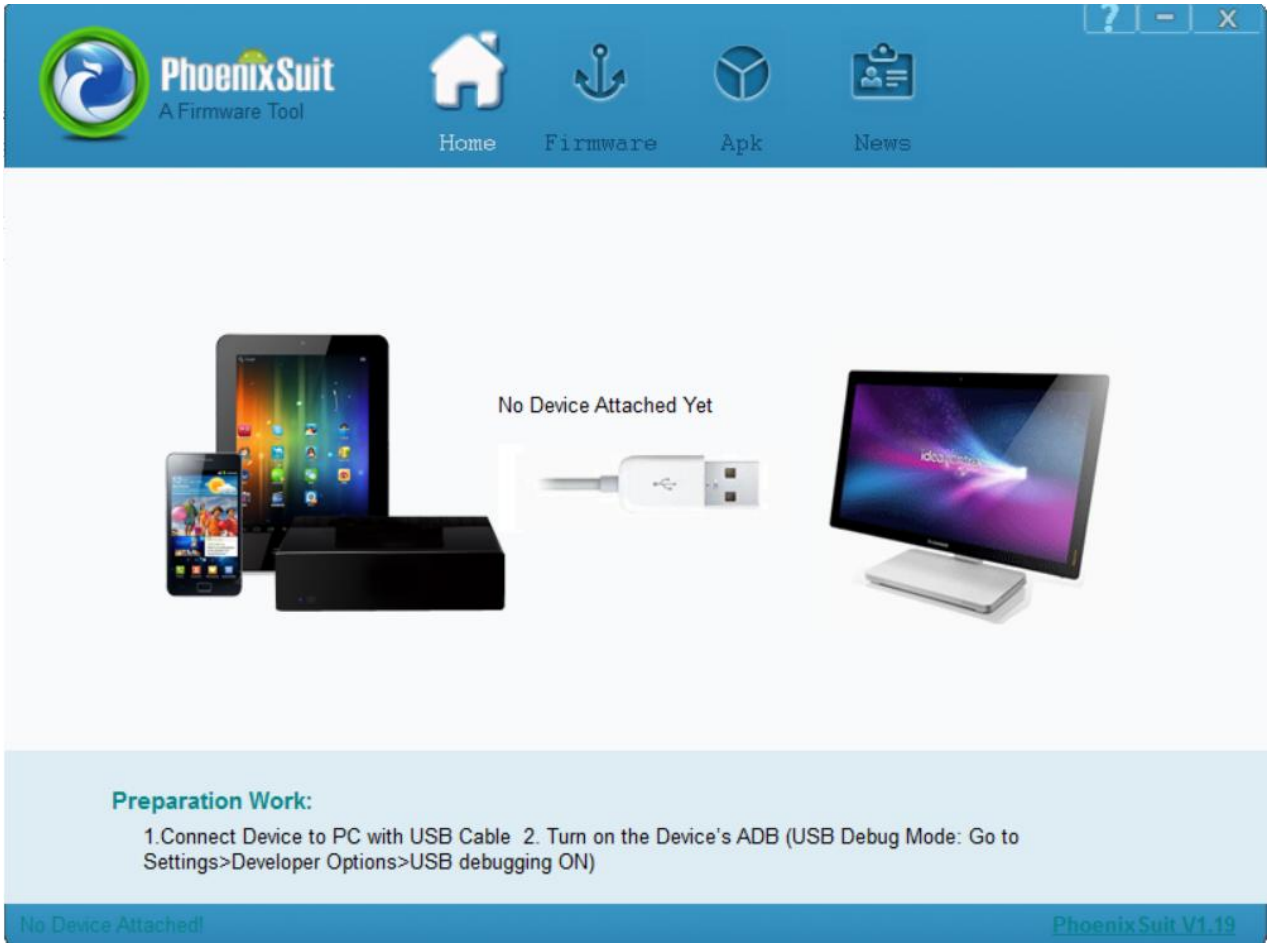

Figure 4-2. PhoenixSuit program

On the following screen, click "One-click refresh" and then click "Browse" to select the firmware image file.

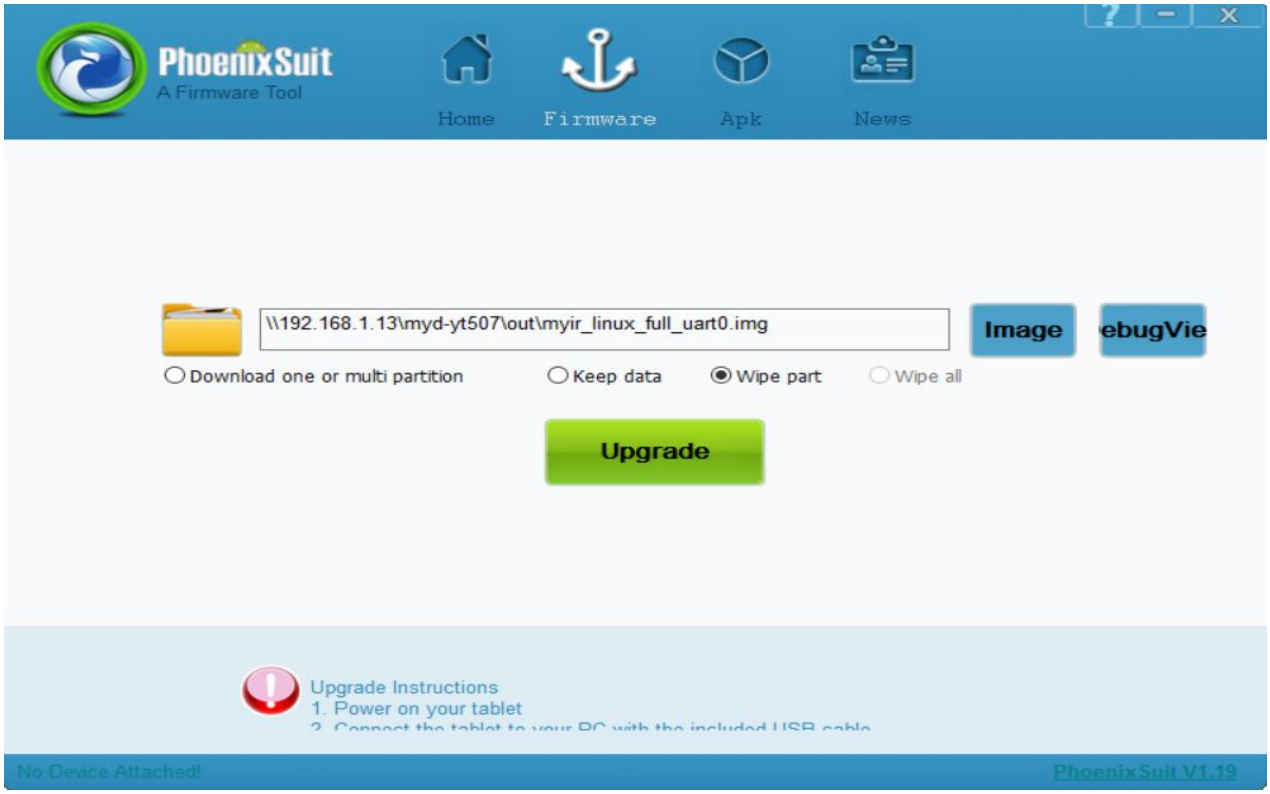

Figure 4-3. One key brush machine

On the following screen.

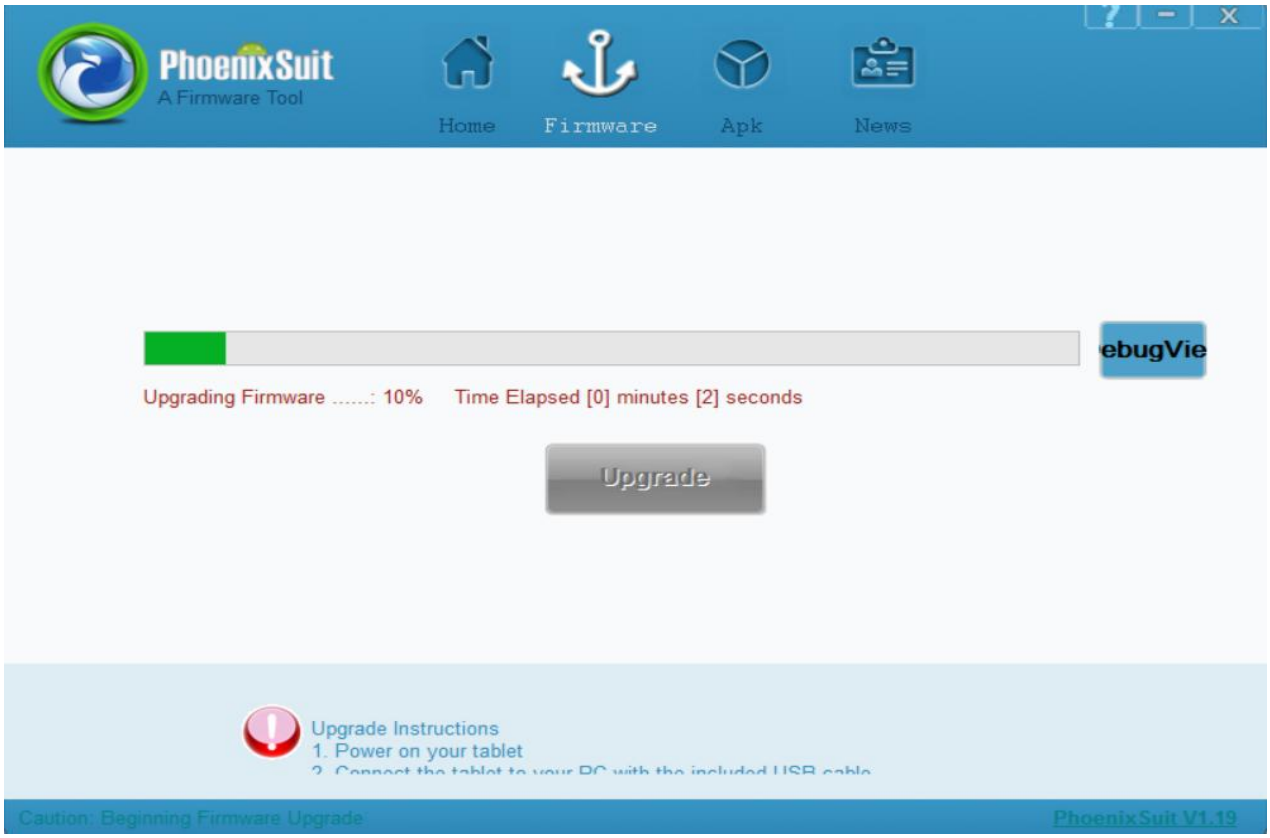

Figure 4-4. Formatting the upgrade

Wait until the burn is complete. The following interface is displayed after the burn is complete.

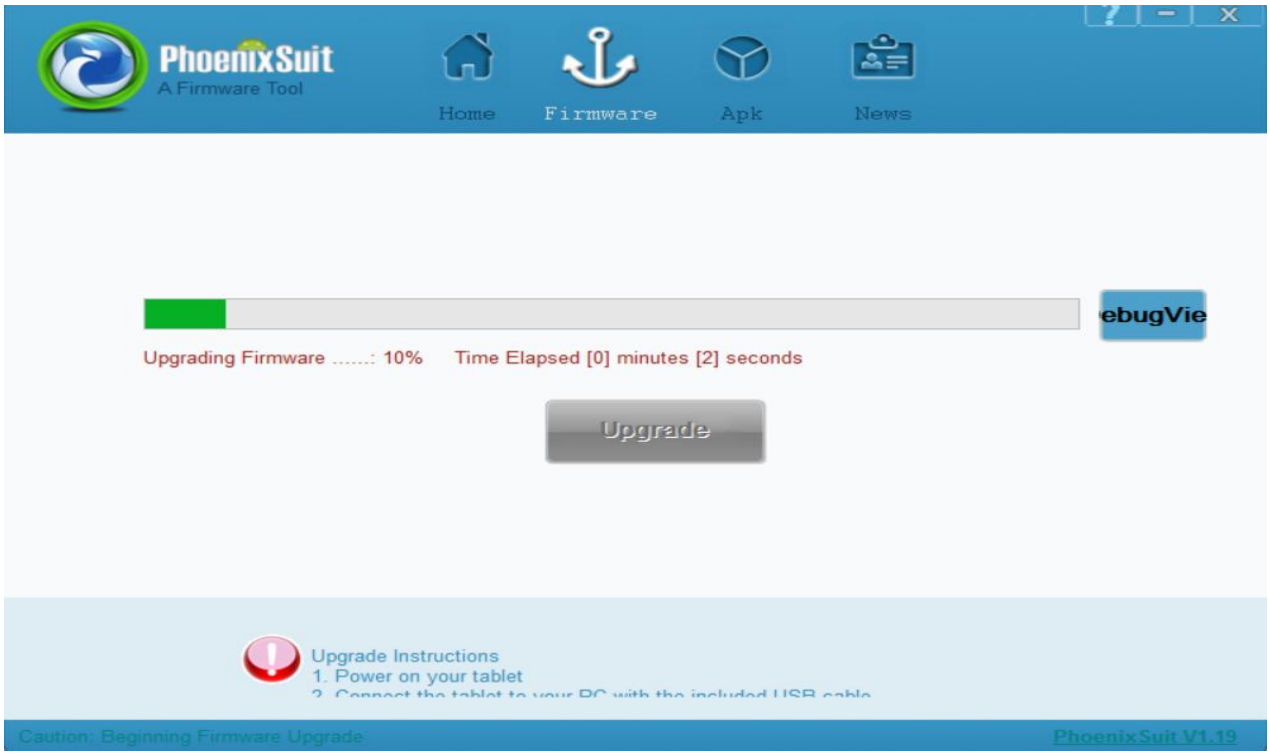

Figure 4-5. Brush finished

4.2. Make an SD card initiator

The following steps are performed on Windows.

1). The preparatory work

- SD card (not less than 8GB)
- MYD-YT507H development board
- Image Preparation Tool PhoenixCard (Path: \03-tools\myir Tools)

Table 4-1. Image package list

Name of the mirror	The package name	Applicable core plate
myir-image-full	myir_linux_full_uart0.img	MYC-YT507H
myir-image-core	myir_linux_core_uart0.img	MYC-YT507H

myir-image-ubuntu	myir_linux_ubuntu_uart0.img	MYC-YT507H
myir-image-android	myir_linux_android_uart0.img	MYC-YT507H

2). Making SD card to start (taking myir-image-full image as an example)

Copy phoenixcard from the tools directory to any directory in Windows. Double-clickPhoenixcard Phoenixcard. exe file in the directory. Insert the 8GB SD card into the Windows USB interface through the SD card reader, as shown in the following figure, and select the "image" path.Select "start up" and click "burn card" button to finish automatically.

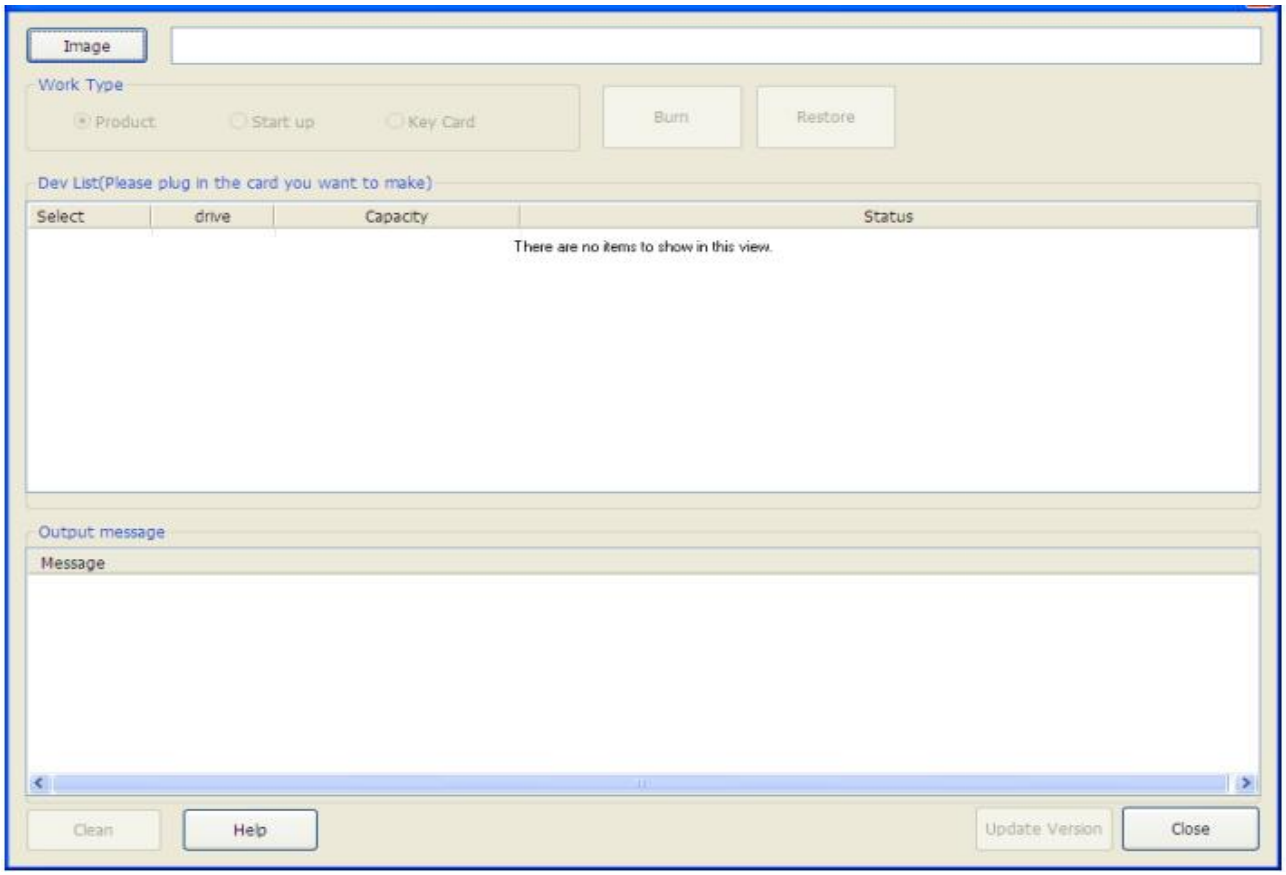

Figure 4-10. Brush program

The card burning process is in progress and is expected to complete in 3-5 minutes (depending on packet size).

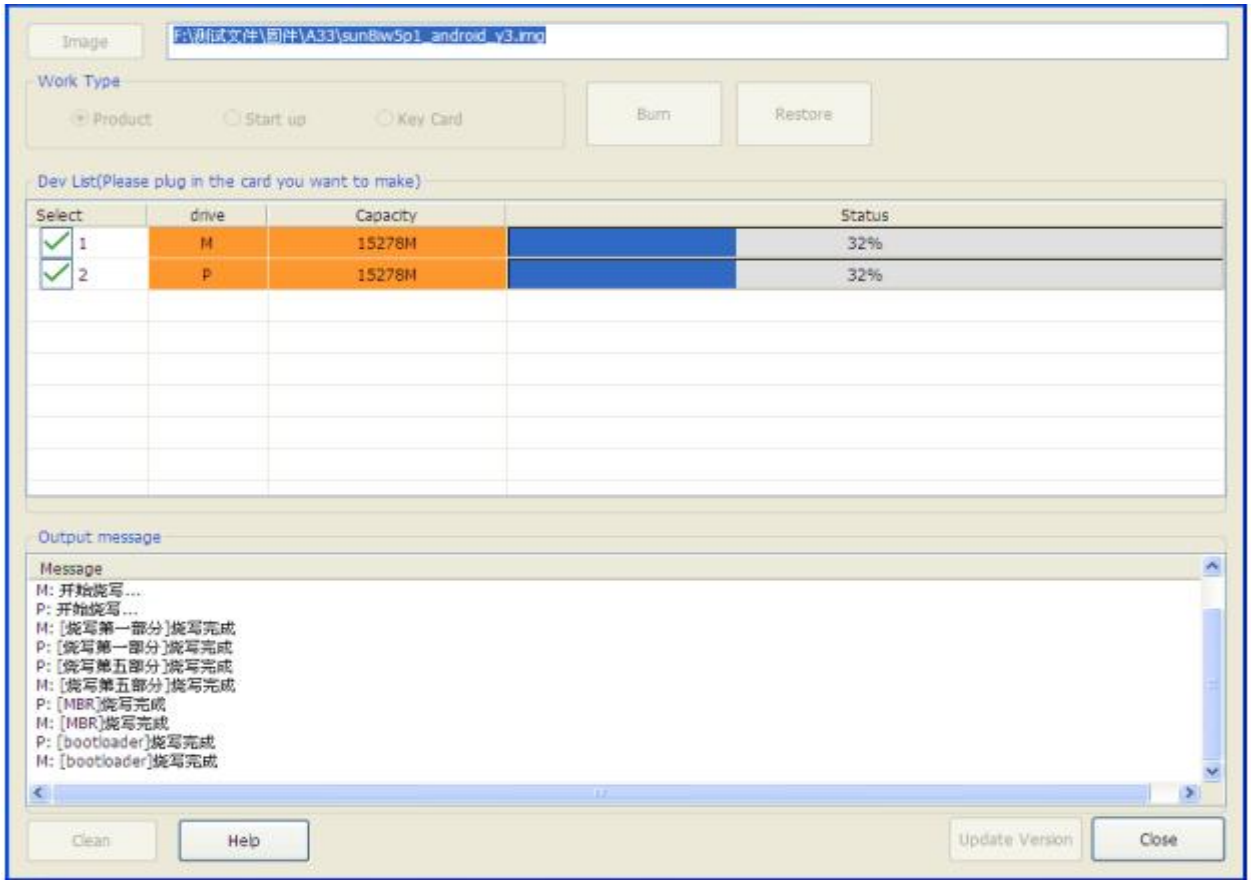

Figure 4-11. Brush process

The card burning is complete, and note that the output indicates that the card burning is complete.

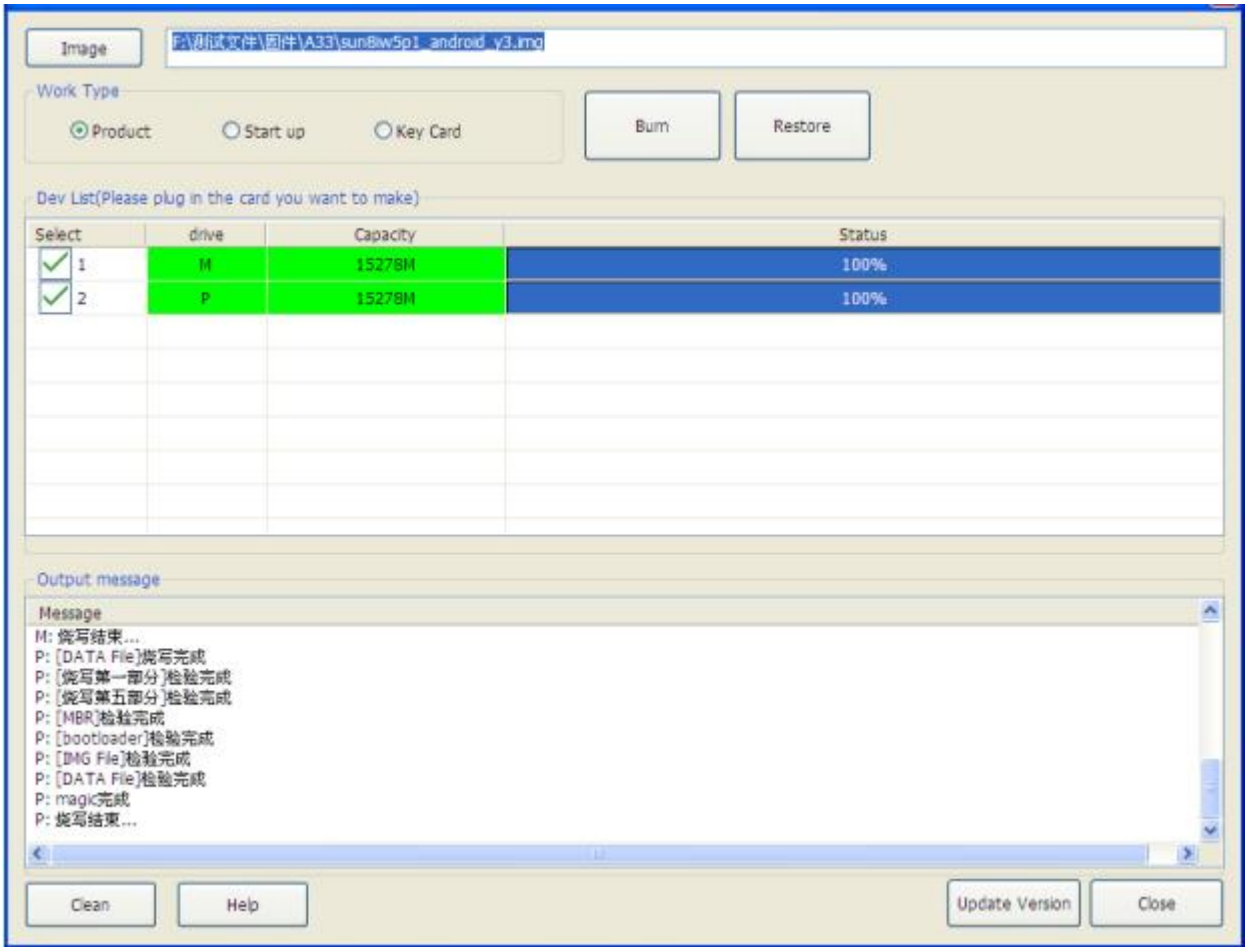

Figure 4-12. Brush finished

After the writing is complete, the SD card can be used to start the system. Insert the SD card into the SD card slot (J8) on the back of the development board, set the boot mode switch to (S0/S1/S2/S3: 0 1 1 1), and power on the system again to start the system with the SD card.

Note: When using the SD card as the boot card to start myir-image-core/full, you need to modify the boot parameters in the Uboot,Procedure On the Uboot terminal, change the rootfs mount location. Enter the following command:

```
=> env set mmc_root /dev/mmcblk1p4
=> saveenv
```

4.3. Make SD card burner

In order to meet the needs of production firing, this method is suitable for mass production firing method.The system that needs to be burned is written into the

onboard eMMC through the system in SD card.Please follow the following steps to complete the specific production process.

- **Make an SD card initiator**

The purpose of making an SD card launcher is to use SD as a medium to write eMMC.

Figure 4-13. Mass production card making

Select "product" to make, and the production method is the same as chapter 4.2, which will not be described here.

- **SD card burns EMMC**

Insert SD into the SD card slot (J8) on the back of the development board, and set the boot mode switch to (S0/S1/S2/S3: 0 1 1 1) to start the system.Plug in the power, automatically start the write system in the SD Card, you can use the

debugging serial port to check the update status, about 3-5 minutes (depending on the size of the package) to complete.

- **Verify eMMC startup**

After the system image is burned to eMMC by Micro SD Card, you need to shut down and pull out the SD Card of eMMC for startup.and set the boot mode switch to (S0/S1/S2/S3: 1 0 1 1) to start the system.

5. How to Modify Board Level Support Package

The previous chapter has described the complete process of building the system image running on MYD-YT507H development board and burning the image to the development board based on Linux SDK(Buildroot) project.Because many pins of MYC-YT507H core board have the characteristics of multiple functions reuse, there are always some differences between the backboard designed based on MYC-YT507H core board and MYB-YT507H in actual projects.These differences may be the removal of display, the addition of more GPIO, the need to add more serial ports, the possibility of extending some peripherals through SPI, I2C, USB, etc.;In addition to the differences in hardware, there are also some differences in system components, such as HMI application, which may need a relatively complete graphics system, QT library, etc., and background management application, which may need a more complete network application, Python running environment, etc.This requires system developers to do some tailoring and porting on top of the code we provide.This chapter describes the specific process of developing and customizing your own system from the perspective of a system developer, laying the foundation for adapting your own hardware.

5.1. Buildroot layer is introduced

Buildroot is a set of Makefiles and patches that simplify and automate building complete, bootable Linux environments (including bootloaders, Linux kernels, file systems containing various apps) for embedded systems.Buildroot runs on Linux and can be used to build an embedded Linux platform for multiple target boards using cross-compilation tools.

5.2. This section describes the board support package

A board-level support package (BSP) is a collection of information that defines how a particular hardware device, device set, or hardware platform is supported.The BSP includes information about hardware features on the device and kernel configuration information, as well as any other hardware drivers required.In some cases, a BSP contains a separately licensed intellectual property (IP) for one or more components.Usually according to the different stages of hardware startup, we divide BSP into Bootloader part and Kernel part. The hardware BSP code designed by MYC-YT507H core board can be viewed in part of Linux SDK.

Brandy only contains SPL and U-boot of Bootloader, which mainly implements core hardware initialization, such as DDR and Clock, and kernel boot.Based on MYC-YT507H core board hardware modification of this part of the content.

```
├ — — SPL - pub
├ — — the tools
└ — — u - the boot - 2018
```

Kernel contains the Linux kernel, which mainly implements the kernel and peripheral firmware content.

```
kernel/
├ — — Linux 4.9
```

When designing products using MYIR's core board, the bootloader part does not need to be modified unless there are special requirements.You need to pay more attention to the development and debugging of product kernel driver and the design of application software.Subsequent chapters will describe kernel development and application development in detail.

5.3. Configuration and build of the Linux SDK

This chapter describes the detailed steps for full and partial compilation.Once compiled, the final .img is generated by packaging.

```
PC$ ./build.sh config
```

```
PC$ ./build.sh
PC$ ./build.sh qt
PC$ ./build.sh
PC$ ./build.sh pack
```

Run build.sh config and select configuration in a subsequent dialog."Choice" can be a number or a number

Enter the string corresponding to the number.

```
PC$ ./build.sh config
Welcome to mkscript setup progress
All available platform:
0. android    // Android
1. linux
Choice [linux]:1
All available linux_dev:
0. bsp
1. dragonboard
2. longan   //default
3. tinyos
Choice [longan]: 2
All available ic:
   0. myir
   1. t507
Choice [myir]: 0
All available board:
   0. core   //myir-image-core
   1. full   //myir-image-full
   2. myd_test
   3. myt
   4. ubuntu //myir-image-ubuntu
Choice [ubuntu]: 1
All available flash:
   0. default
```

```
    1. nor
Choice [default]: 0
All available rootfs:
  0. buildroot-201902   //for linux rootfs
  1. ubuntu    //for ubuntu rootfs
All available build_root:
    0. buildroot-201902 //for linux rootfs
    1. myir-t5-buildroot   //Like buildroot-201902
    2. ubuntu //for ubuntu rootfs
Choice [ubuntu]: 0
```

Go to the top-level directory and run the following command.

```
PC$ ./build.sh
PC$ ./build.sh qt
PC$ ./build.sh
PC$ ./build.sh pack
```

Figure 5-1.Completion of system

Table 5-1. Parameter Description

type	The command	instructions
The overall compilation	./build.sh config	Compile configuration, pop-up compile selection
	./build.sh autoconfig	Compile configuration based on the parameters passed in, no
	./build.sh	Build the SDK according to the build configuration
	./build.sh clean	Clear process files and object files
	./build.sh distclean	Clear all generated files
Local compile	./build.sh brandy	Compile the brandy (uboot)
	./build.sh kernel	Compile the kernel
	./build.sh buildroot	Compile buildroot
	./build.sh qt	Compile the qt
	./build.sh dragonboard	Compile dragonboard

	./build.sh sata	Compile the sata
packaging	./build.sh pack	Package command, debugging serial port is uart0
	./build.sh pack_debug	Package command, debugging serial port is card0
	./build.sh pack_secure	Package command, generate secure firmware, debug serial

Compile problem analysis:

The following problem occurs when you run ./build.sh config.

```
Choice [default]: 0
All available rootfs:
  0. buildroot-201902
  1. ubuntu
All available build_root:
   0. buildroot-201902
   1. myir-t5-buildroot
   2. ubuntu
Choice [buildroot-201902]: 0
  File "<string>", line 1
    import os.path; print os.path.relpath('/home/zhaoy/t507/kernel/linux-4.9/arch/arm64/configs/sun50iw9p1smp_longan_defconfig', '/home/zhaoy/t507/kernel/linux-4.9/arch/arm64/configs')
                    ^
SyntaxError: invalid syntax
ERROR: Can't find kernel defconfig!
root@myir-O-E-M:/home/zhaoy/t507#
```

Figure 5-2. Configuration failed

```
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/basic/bin2c
  HOSTCC  scripts/kconfig/conf.o
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf  --defconfig Kconfig
***
*** Can't find default configuration "arch/arm64/defconfig"!
***
make[1]: *** [scripts/kconfig/Makefile:100：defconfig] 错误 1
make: *** [Makefile:560: defconfig] Error 2
ERROR: build kernel Failed
INFO: mkkernel failed
```

Figure 5-3. Failed to compile the kernel

Since configuring the kernel requires Python2, check to see if Python2 is already installed in your current development environment. If no, see Section 2.1 or run the following command:

```
PC$ sudo apt-get install python
```

Check whether the current version is Python

```
PC$ python --version
Python 2.7.18
```

If not python2, switch by yourself.

The following error occurs during gdbus compilation:

Figure 5-4. Failed to compile gdbus

Modify step 1 (note that the name of the path may be different, just find gdbusauth.c according to your actual path):

```
PC$ vim ./output/myir/full/longan/buildroot/build/host-libglib2-2.56.3/gio/gdbusauth.c
```

Add this code at the following locations:

```
line = _my_g_input_stream_read_line_safe (g_io_stream_get_input_stream (auth->priv->stream),
&line_length,
cancellable,
error);
 if (line != NULL)
        debug_print ("SERVER: WaitingForBegin, read '%s'", line);
if (line == NULL)
```

Modify step 2 (note that the name of the path may be different, just find gdbusmessage.c according to your actual path):

```
PC$ vim ./out/myir/full/longan/buildroot/build/host-libglib2-2.56.3/gio/gdbusmessage.c
```

Add this code at the following locations:

```
signature_str = g_variant_get_string (signature, NULL);
if (message->body != NULL)
{
        gchar *tupled_signature_str;
        if (signature != NULL)
            tupled_signature_str = g_strdup_printf ("(%s)", signature_str);
        if (signature == NULL)
```

5.4. Onboard U-boot compilation and update

Uboot is a very rich open source to start the bootloader, kernel boot, download from many aspects, such as update, they are widely used in embedded field can check the website http://www.denx.de/wiki/U-Boot/WebHome for more information

T5 platform also uses Boot Chains as boot programs, different Boot Chains mode will correspond to different boot stages.

5.4.1. Compile the U-boot in a separate cross-compile environment

1). Obtain u-boot source code

download YT507H-buildroot-t5-4.9.170-X.X.X.tar.bz

```
PC$ cd $HOME/work/t507
PC$ tar -jxvf YT507H-buildroot-t5-4.9.170-X.X.X.tar.bz
```

- Source code directory: U-boot-2018
- SPL source code directory: SPL -pub
- Compile the script: build.sh

```
LRWXRWXRWX 1 root root 14 2月 10 10:44 build.sh -> tools/build.sh
Drwxr-xr-x 10 root root 4096 3月 4 14:59 SPL -pub
Drwxr-xr-x 5 root root 4096 February 10 10:45 tools
Drwxr-xr-x 26 root root 4096 3月 4 14:59 U-boot-2018
```

2). Configuration and Compilation

- **Go to the source directory**

```
PC $cd brandy/brandy-2.0
```

- **Load the toolchain in the SDK**

```
PC$ ./build.sh -t
Prepare toolchain ...
```

- **Compile the Uboot**

```
PC$ cd u-boot-2018
PC$ make sun50iw9p1_defconfig
PC$ make -j
```

5.4.2. Compiling Uboot under Linux SDK projects (recommended)

After the user has modified the U-Boot code according to the iterative development process in 5.4.1, the SDK can also be used to build the entire image.

Compile uboot source code:

```
PC$ ./build.sh brandy
```

Package files:

```
PC$ ./build.sh pack
```

5.4.3. How do I update the Uboot separately

Execute a separate update policy using the PhoenixSuit. See Chapter 4.1 for details.

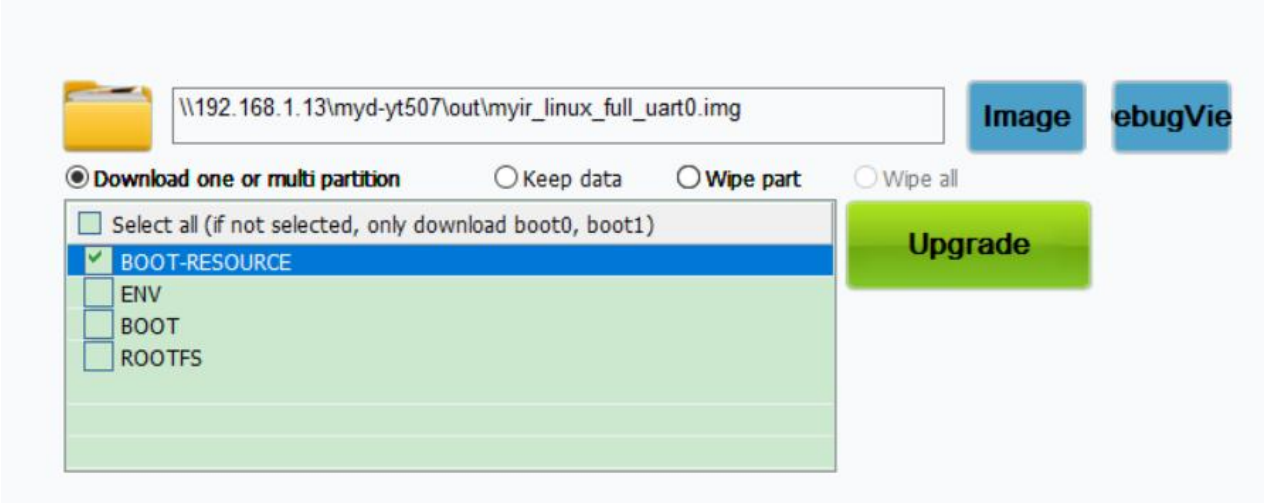

Figure 5-1. Updating the Uboot

5.5. On board Kernel compilation and update

The Linux Kernel is a very large open source kernel, which is used in various distribution operating systems. The Linux Kernel is widely used in embedded systems due to its portability, support of various network protocols, independent module mechanism, MMU and many other rich features.

At the same time, T5 also supports the Linux kernel, which will be updated for a long time. MYD-YT507H uses the T5 kernel transplant, and the latest version supports the Linux kernel 4.9.170.

5.5.1. Compile the Kernel in a separate cross-compile environment

5.5.1.1. Obtaining kernel source code

```
PC$ cd $HOME/work/t507
PC $tar -jxvf YT507H-buildroot-t5-4.9.170-X.X.X.tar.bz
PC$ cd kernel/
```

The directory contains:

- Source code soft link directory: Linux-4.9
- Source code: myir-t5-kernel

```
LRWXRWXRWX 1 lcy root 15 10 月 19 17:47 linux-4.9 -> myir-t5-kernel/
Drwxr-xr-x 29 lcy root 4096 3 月 22 10:21 myir-t5-kernel
```

5.5.1.2. Configuring the kernel (optional)

MYIR has integrated most of the functionality into the kernel and generally does not require configuration.To add special functions, configure peripheral drivers in the following way.

- **Go to the kernel directory**

```
PC$ cd t507/kernel/Linux-4.9
```

- **Create the output folder build**

```
PC$ mkdir -p ../build
```

- **Configuring the kernel**

```
PC$ make ARCH=arm64 O="$PWD/../build" sun50iw9p1_longan_defconfig
```

You can also use the following methods to configure the kernel or enable a driver function of the kernel.

```
PC$ cd ../build
PC$ make menuconfig
```

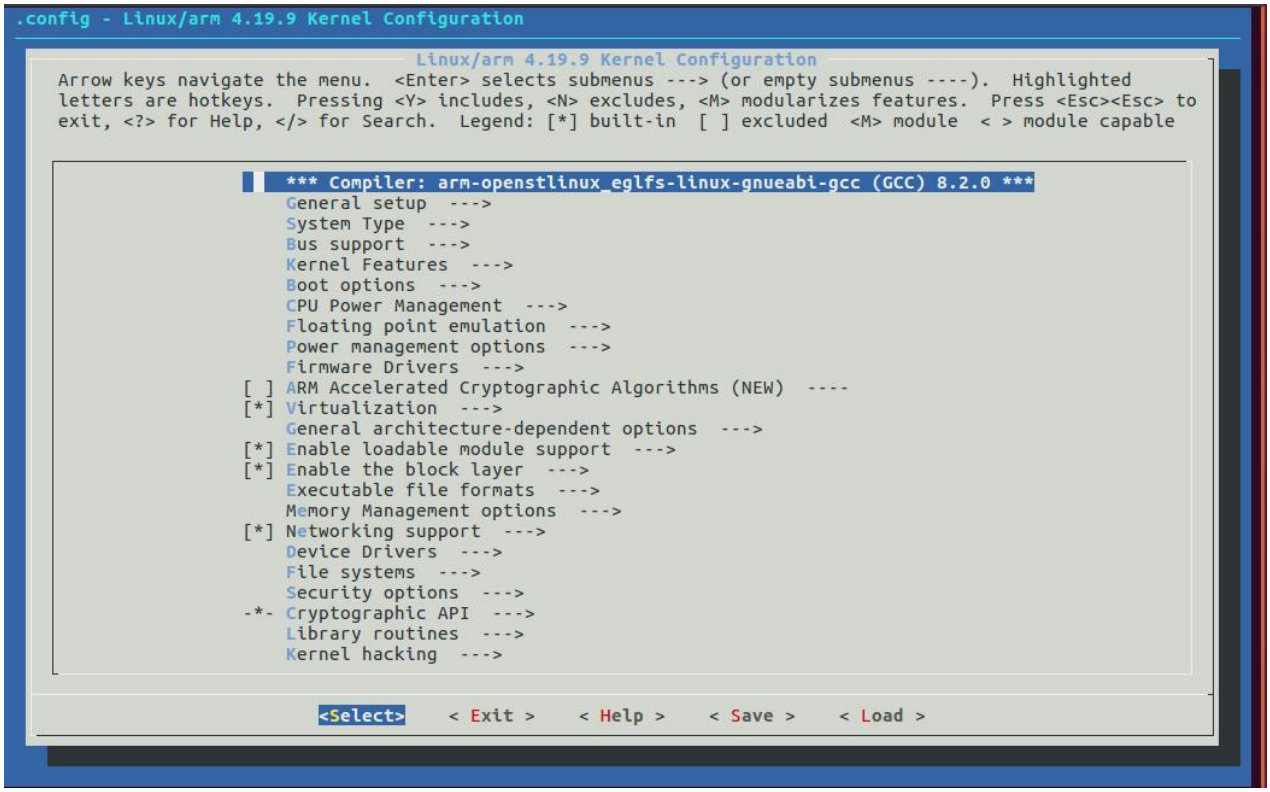

Figure 5-2. Kernel configuration page

5.5.1.3. Compiling the kernel (not recommended)

- **Load SDK environment variables**

```
PC$ export PATH=$PATH:/opt/gcc-linaro-7.4.1-2019.02-x86_64_aarch64-Linux-gnu/bin
```

- **Configuring the kernel**

```
PC$ make ARCH=arm64 O="$PWD/../build" sun50iw9p1_longan_defconfig
```

- **Compile the kernel**

```
PC$ make ARCH=arm64 uImage vmlinux dtbs O="$PWD/../build"
```

```
PC$ make ARCH=arm64 modules O="$PWD/../build"
```

Wait patiently for compilation to complete.

5.5.1.4. Compiling with the Linux SDK (recommended)

Compile kernel source code:

```
PC$ ./build.sh kernel
```

Package files:

```
PC$ ./build.sh pack
```

5.5.2. How to update the Kernel with OTG

Execute a separate update policy using the PhoenixSuit. Select BOOT. For details, see 4.1.

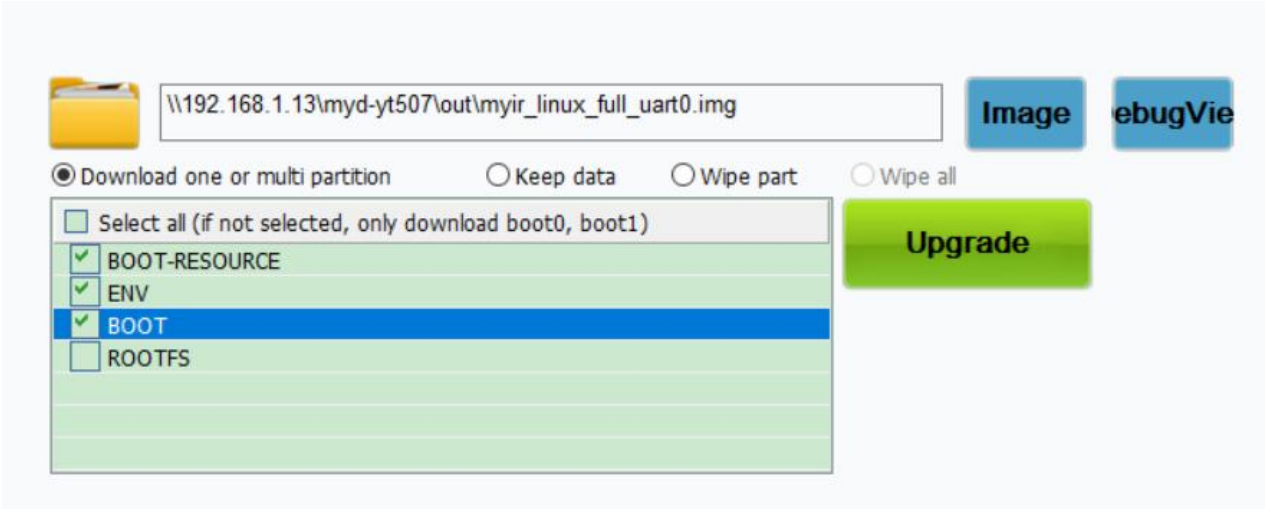

Figure 5-3. Brush system

6. How to Fit Your Hardware Platform

In order to adapt to your new hardware platform, you need to know what resources mill's MYD-YT507H development board provides. For details, see MYD-YT507H SDK1.0.0 Release Notes.In addition, users also need to have a detailed understanding of the CPU chip manual, as well as the product manual of MYC-YT507H core board, pin definition, in order to facilitate the correct configuration and use of these pins according to the actual function.

6.1. How do I configure your sys_config.fex

Sys_config. fex is a set of function configuration files defined by T5. This file can be used to define pins, properties, power supply of each node, so that users can quickly configure the function of resources.To enable users to master sys_config.fex configuration and usage.This chapter will explain how to use it

sys_config. fex file path:

```
PC$: device/config/chips/t507/configs/myir/sys_config.fex
```

Define attribute class methods:

```
[product]
version = "100"
machine = "demo2"

[platform]
eraseflag    = 1
debug_mode   = 0
;------------------------------------------------------------------------------
;[target]   system bootup configuration
;boot_clock       = CPU boot frequency, Unit: MHz
;storage_type     = boot medium, 0-nand, 1-card0, 2-card2, -1(defualt)auto sca
n
```

```
;Advert_enable = 0-close advert logo 1-open advert logo (only valid under m
ulti-core startup)
;-------------------------------------------------------------------------------
[target]
boot_clock          = 1008
storage_type        = -1
advert_enable       = 0
burn_key            = 1
```

Define pin mode:

```
[card0_boot_para]
card_ctrl           = 0
card_high_speed  = 1
card_line           = 4
sdc_d1              = port:PF0<2><1><3><default>
sdc_d0              = port:PF1<2><1><3><default>
sdc_clk             = port:PF2<2><1><3><default>
sdc_cmd             = port:PF3<2><1><3><default>
sdc_d3              = port:PF4<2><1><3><default>
sdc_d2              = port:PF5<2><1><3><default>
;sdc_type           = "tm1"
```

* Due to the relevant authorization, please contact The technical support of MYIR to obtain the details of the above two configuration definitions in the document t507_sys_config.fex Usage Configuration Description files.

6.2. How do I create your device tree

6.2.1. Onboard device tree

Users can create their own device trees in the BSP source code, generally without modifying the code in the Bootloader section.You only need to adjust the Linux kernel device tree based on actual hardware resources.The device tree list in each part of BSP of MYD-YT507H is listed here for user development reference. The specific content is shown in the following table:

Table 6-1. MYD-YT507H device tree list

project	Device tree	instructions
U-boot	sys_config.fex	Sys_config.fex configuration (see 6.1)
Kernel	sys_config.fex	Sys_config.fex configuration (see 6.1)
	board.dts	Backplane Configuration Resources
	myir-YT507H.dtsi	Resources are configured internally on the core board
	sun50iw9p1-myir.dtsi	Core Resource Allocation
	sun50iw9p1-myir-pinctrl.dtsi	Pin configuration
	display/myir-hdmi-1920x1080-1lvds-7-1024x600.dtsi	HDMI and LVDS dual display device tree configuration
	display/myir-lcd-1lvds-7-1024-600.dtsi	7 inch single channel LVDS device tree configuration
	display/myir-lcd-2lvds-21-1920-1080.dtsi	21 inch dual LVDS device tree configuration
	display/myir-tv.dtsi	CVBS-OUT Device tree configuration
	Display/myir-hdmi.dtsi	HDMI Device tree configuration

DTS path:

device/config/chips/myir/configs/xxx/sys_config.fex (xxx indicates configuration.)

device/config/chips/myir/configs/xxx/board.dts

kernel/linux-4.9/arch/arm64/boot/dts/sunxi/

kernel/linux-4.9/arch/arm64/boot/dts/sunxi/display/

6.2.2. Add a device tree

Linux kernel device tree is a data structure that describes on-chip and off-chip device information in a unique syntactic format.The BootLoader passes it to the kernel, which forms the dev structure associated with the driver for the driver code to use after parsing.

In the kernel source arch/arm64/boot/dts/sunxi can see a large number of platform device trees.If a device tree is suitable for MYD-YT507H, you can add a custom device tree to the current path, for example:

PATH: linux-4.9/arch/arm64/boot/dts/sunxi

We write the resources related to MYD-YT507H core board into sun50iw9p1-myir.dtsi, myir-yt507.dtsi and board.dts.Other extended interfaces and devices can reference them, as shown below (for reference only) :

PATH: device/config/chips/myir/configs/full/board.dts

```
/ *
 * myir-YT507H support.
 * /
/dts-v1/;

#include "myir-yt507.dtsi"
#include "display/myir-hdmi-1920x1080-1lvds-7-1024x600.dtsi"
```

```
#include "display/myir-hdmi-1920x1080-1lvds-7-1024x600.dtsi"
//#include "display/myir-lcd-1lvds-7-1024-600.dtsi"
//#include "display/myir-lcd-lvds-10.1-1280-800.dtsi"
//#include "display/myir-lcd-2lvds-7-1024-600.dtsi"
//#include "display/myir-lcd-2lvds-21-1920-1080.dtsi"
//#include "display/myir-hdmi.dtsi"
//#include "display/myir-tv.dtsi"

/{
        model = "myir-yt507h-full";
      compatible = "allwinner,t507", "arm,sun50iw9p1";

      aliases {
            pmu0 = &pmu0;
            standby_param = &standby_param;
      };

      soc@03000000 {
            twi2: twi@0x05002800{
                  status = "okay";
            };
      };
};
```

6.3. How to configure CPU function pins based on your hardware

Realizing the control of a function pin is one of the more complex system development process, which includes the configuration of the pin, the development of the drive, the implementation of the application and so on. This section does not analyze the development process of each part in detail, but explains the control implementation of the function pin by example.

6.3.1. GPIO pin configuration method

GPIO: general-purpose input/output is a very important resource in embedded devices. You can output high and low levels through them or read pin states through them - high or low levels.

T5 encapsulates a large number of peripheral controllers. The communication between these peripheral controllers and external devices is generally realized by controlling GPIO, and the GPIO is used by peripheral controllers as Alternate Function, which endodes them with more complex functions. For example, users can use GPIO port to interact with external hardware (such as UART), control hardware work (such as LED, buzzer, etc.), and read hardware working status signals (such as interrupt signals).Therefore, GPIO port is widely used.

1). Manual Query

The configuration of GPIO can be found in the description Document (01-Document Datasheet) and the list of core board pins (01-Document Hardware\) compiled by Mill as follows:

- **General calculation method**

  ((port * 16 + line) << 8) | function

- **Configuration pinmux**

The ncSI0-D15 of Blue on the LED interface is taken as an example. Check the hardware manual to obtain this pin reuse function

PE19 / NCSI0 - D15 / PE - EINT20

Table 6-2. Pin pin definitions

Function	pin Name	IO Type	IO State	up/down	Multi2(DDR4)	Multi3(DDR3)	Multi4(LPDDR3)	Multi5(LPDDR4)	Multi6	PIN power
PE	PE19	I/O	DIS		NCSI-D15				PE-EINT20	VCC-PE

6.3.2. GPIO is referenced in device tree

1). Configure function pins as GPIO function instances

This example uses PE19 as the GPIO test.This section describes how to configure device nodes in the device tree for use by kernel drivers in subsequent sections.This example can also be used to control the reset of external devices, power supply and other control functions for reference.

Simply add nodes to the device tree.

```
//device/config/chips/myir/configs/full/board.dts
gpioctr_device {
    compatible = "myir,gpioctr";
    status = "okay";
    gpioctr-gpios = <&gpioe 19 0>;
};
```

2). Development board LCD resource reallocation example

MYD-YT507H development board defines and realizes many rich functions, but also occupies a large number of pin resources, such as users directly use MYD-YT507H based on the design and development, will need to redefine and configure the pin.LCD reuse pin (LCD-D0) function, you can first check the table to understand the reuse pin function.

Table 6-3. Reuse list of pins (partial description)

Function	pin Name	IO State	up/down	Multi2(DDR4)	Multi3(DDR3)	Multi4(LPDDR3)	Multi5(LPDDR4)	Multi6	PIN power

PD	PD0	I/O	DIS	LCD-D0	LVDF0-V0P	TS0-CLK		PD-EINT0	VCC-PD

MYD-YT507H development board has used PD0 two pins as LVDS data signal pins, the pin configuration is as follows:

```
//kernel/Linux-4.9/arch/arm64 /boot/DTS/sunxi/sun50iw9p1-myir-pinctrl.dtsi
 lvds0_pins_a: lvds0@0 {
                        allwinner,pins = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5
", "PD8", "PD9", "PD6", "PD7";
                        allwinner,pname = "PD0", "PD1", "PD2", "PD3", "PD4", "P
D5", "PD8", "PD9", "PD6", "PD7";
                        allwinner,function = "lvds0";
                        allwinner,muxsel = <3>;
                        allwinner,drive = <3>;
                        allwinner,pull = <0>;
                };
                lvds0_pins_b: lvds0@1 {
                        allwinner,pins = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5
", "PD8", "PD9", "PD6", "PD7";
                        allwinner,pname = "PD0", "PD1", "PD2", "PD3", "PD4", "P
D5", "PD8", "PD9", "PD6", "PD7";
                        allwinner,function = "lvds0_suspend";
                        allwinner,muxsel = <7>;
                        allwinner,drive = <3>;
                        allwinner,pull = <0>;
                };
```

Refer to the manual to reconfigure pins for i2C1 and assign them to PD0 (Multi2(DDR4)).

```
//Linux-4.9/arch/arm64/boot/dts/sunxi/display/myir-lvds-lcd-1-7-1024-600.dtsi
```

```
&lcd0 {
        lcd_used            = <1>;
        lcd_driver_name     = "default_lcd";
        lcd_backlight       = <200>;
        lcd_if              = <3>;
        lcd_x               = <1024>;
        lcd_y               = <600>;
        lcd_width           = <150>;
        lcd_height          = <94>;
        lcd_dclk_freq       = <50>;    / / < 70 >
        lcd_pwm_used        = <1>;
        lcd_pwm_ch          = <0>;
        lcd_pwm_freq        = <50000>;
        lcd_pwm_pol         = <1>;
        lcd_pwm_max_limit   = <255>;
        lcd_hbp             = <160>;
        lcd_ht              = <1324>;
        lcd_hspw            = <116>;
        lcd_vbp             = <24>;
        lcd_vt              = <629>;
        lcd_vspw            = <3>;
        lcd_lvds_if         = <0>;
        lcd_lvds_colordepth = <0>;
        lcd_lvds_mode       = <0>;
        lcd_frm             = <0>;
        lcd_hv_clk_phase    = <0>;
        lcd_hv_sync_polarity= <0>;
        lcd_gamma_en        = <0>;
        lcd_bright_curve_en = <0>;
        lcd_cmap_en         = <0>;
        lcd_fsync_en        = <0>;
        lcd_fsync_act_time  = <1000>;
        lcd_fsync_dis_time  = <1000>;
```

```
                lcd_fsync_pol         = <0>;
                deu_mode              = <0>;
                lcdgamma4iep          = <22>;
                smart_color           = <90>;
                lcd_pin_power = "bldo1";
                lcd_power = "dc1sw";
                //lcd_bl_en           = <&pio PD 28 1 0 3 1>;
                //lcd_gpio_0          = <&pio PH 4 1 0 3 1>;

                pinctrl-0 = <&lvds0_pins_a>;
                pinctrl-1 = <&lvds0_pins_b>;
```

If the lcd0 interface is not used, you can set the node status of the lcd device tree to "disabled".

```
&lcd0 {
            status = "disabled";
        };
```

6.4. How to use self-configured pins

The pins configured in the device tree of u-Boot or Kernel can be used in u-Boot or Kernel to control the pins.

6.4.1. GPIO pins are used in u-boot

1). Terminal command control

Uboot can control GPIO Settings directly using commands.To set GPIOF14, run the following command.

```
=> gpio set GPIOF14
gpio: pin GPIOF14 (gpio 94) value is 1
=> gpio clear GPIOF14
gpio: pin GPIOF14 (gpio 94) value is 0
```

6.4.2. GPIO pins are used in kernel drivers

1). Use of independent I/O drivers

In the first device tree example in Section 6.2.3, the GPIO node information has been defined, and the kernel driver will be used to control THE GPIO (set the PF14 pin to 1 and 0, if necessary, use a multimeter to test the pin level change).

```
//gpioctr.c
#include <linux/module.h>
#include <linux/of_device.h>
#include <linux/fs.h>
#include <linux/errno.h>
#include <linux/miscdevice.h>
#include <linux/kernel.h>
#include <linux/major.h>
#include <linux/mutex.h>
#include <linux/proc_fs.h>
#include <linux/seq_file.h>
#include <linux/stat.h>
#include <linux/init.h>
#include <linux/device.h>
#include <linux/tty.h>
#include <linux/kmod.h>
#include <linux/gfp.h>
#include <linux/gpio/consumer.h>
#include <linux/platform_device.h>

/* 1. Determine the number of the main device */
static int major = 0;
static struct class *gpioctr_class;
static struct gpio_desc *gpioctr_gpio;
```

```
/* 2. Implement the corresponding open/read/write functions and fill in the fi
le_operations structure */
static ssize_t gpio_drv_read (struct file *file, char __user *buf, size_t size, loff_t
*offset)
{
        printk("%s %s line %d\n", __FILE__, __FUNCTION__, __LINE__);
        return 0;
}

static ssize_t gpio_drv_write (struct file *file, const char __user *buf, size_t size,
 loff_t *offset)
{
        int err;
        char status;

        printk("%s %s line %d\n", __FILE__, __FUNCTION__, __LINE__);
        err = copy_from_user(&status, buf, 1);

        gpiod_set_value(gpioctr_gpio, status);

        return 1;
}

static int gpio_drv_open (struct inode *node, struct file *file)
{
        gpiod_direction_output(gpioctr_gpio, 0);

        return 0;
}

static int gpio_drv_close (struct inode *node, struct file *file)
{
        printk("%s %s line %d\n", __FILE__, __FUNCTION__, __LINE__);
```

```
        return 0;
}

/* Define your own file_operations structure */
static struct file_operations gpioctr_drv = {
        .owner      = THIS_MODULE,
        .open       = gpio_drv_open,
        .read       = gpio_drv_read,
        .write      = gpio_drv_write,
        .release  = gpio_drv_close,
};

/* Obtain GPIO from platform_device
 * Tell the kernel the file_operations structure: register the driver
 * /
static int chip_demo_gpio_probe(struct platform_device *pdev)
{
        /* Gpioctr - gPIOS =<...>;      * /
        gpioctr_gpio = gpiod_get(&pdev->dev, "gpioctr", 0);
        if (IS_ERR(gpioctr_gpio)) {
             dev_err(&pdev->dev, "Failed to get GPIO for led\n");
             return PTR_ERR(gpioctr_gpio);
        }

        / * registered file_operations  * /
        major = register_chrdev(0, "myir_gpioctr", &gpioctr_drv);  /* /dev/gpioct
r */

        gpioctr_class = class_create(THIS_MODULE, "myir_gpioctr_class");
        if (IS_ERR(gpioctr_class)) {
             printk("%s %s line %d\n", __FILE__, __FUNCTION__, __LINE__);
             unregister_chrdev(major, "gpioctr");
             gpiod_put(gpioctr_gpio);
```

```
            return PTR_ERR(gpioctr_class);
        }

        device_create(gpioctr_class, NULL, MKDEV(major, 0), NULL, "myir_gpioct
r%d", 0);

    return 0;

}

static int chip_demo_gpio_remove(struct platform_device *pdev)
{
        device_destroy(gpioctr_class, MKDEV(major, 0));
        class_destroy(gpioctr_class);
        unregister_chrdev(major, "myir_gpioctr");
        gpiod_put(gpioctr_gpio);

    return 0;
}

static const struct of_device_id myir_gpioctr[] = {
    { .compatible = "myir,gpioctr" },
    {},
};

/* Define platform_driver */
static struct platform_driver chip_demo_gpio_driver = {
    .probe        = chip_demo_gpio_probe,
    .remove       = chip_demo_gpio_remove,
    .driver       = {
        .name     = "myir_gpioctr",
        .of_match_table = myir_gpioctr,
    },
```

```
};

/* Register platform_driver */ in the entry function
static int __init gpio_init(void)
{
    int err;
    err = platform_driver_register(&chip_demo_gpio_driver);

        return err;
}

/* Where there is an entry function, there should be an exit function: this exi
t function is called when the driver is uninstalled
 * uninstall platform_driver
 * /
static void __exit gpio_exit(void)
{
    platform_driver_unregister(&chip_demo_gpio_driver);
}

/* Other improvements: provide device information, automatically create devic
e nodes */
module_init(gpio_init);
module_exit(gpio_exit);

MODULE_LICENSE("GPL");
```

Compiling driver code into modules using separate makefiles can also be configured directly into the kernel.

2). The driver sample will be configured directly into the kernel

Create a gpioctr.c file under the sample folder of the kernel source code, copy the above driver code into it, and modify Kconfig and Makefile and sun50iw9p1_myir_defconfig.

Add Kconfig:

```
//linux/sample/Kconfig
config SAMPLE_GPIO
        tristate "this is a gpio test driver"
        depends on CONFIG_GPIOLIB
```

Add a Makefile:

```
//linux/sample/Makefile
# SPDX - License - Identifier: GPL - 2.0
# Makefile for Linux samples code

obj-$(CONFIG_SAMPLE_ANDROID_BINDERFS)    += binderfs/
.
obj-$(CONFIG_SAMPLE_GPIO)                    += gpioctr.o
```

Add sun50iw9p1_myir_defconfig:

```
/ / Linux-4.9/arch/arm64/configs/sun50iw9p1_myir_defconfig
CONFIG_SAMPLES=y
CONFIG_SAMPLE_GPIO=y
CONFIG_SAMPLE_RPMSG_CLIENT=m
```

Follow section 5.5.3 to compile and update the kernel.

3). The driver sample compiles to a separate module

Add gpioctr.c to your working directory and copy the driver code above. Write a separate Makefile in the same directory.

```
# modified KERN_DIR
#KERN_DIR =#the directory where the kernel source is used by the board
KERN_DIR = $HOME/work/t507/kernel/Linux-4.9/

obj-m += gpioctr.o
```

```
all:
	make -C $(KERN_DIR) M=`pwd` modules
clean:
	make -C $(KERN_DIR) M=`pwd` modules clean
	rm -rf modules.order

#To compile a.c, b.c into ab.ko, you can specify:
# ab-y := a.o b.o
# obj-m += ab.o
```

Load SDK environment variables into the current shell.

```
PC$ export PATH=$PATH:/opt/gcc-linaro-7.4.1-2019.02-x86_64_aarch64-Linux-gnu/bin
```

Run the make command to generate the gpioctr.ko driver module file.

```
root@ubuntu:/home/myir# make
The make -C/home/lcy/work/t507/kernel/linux-4.9/ M =` PWD `modules
Make [1] : if the directory '/ home/lcy/work/t507/kernel/linux-4.9 /'
  CC [M]  /home/myir/gpioctr.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/myir/gpioctr.mod.o
  LD [M]  /home/myir/gpioctr.ko
Make [1] : brigade directory '/home/lcy/work/t507/kernel/linux-4.9/'
```

After successful compilation, the gpioctr.ko file can be transferred to the /lib/modules directory of the development board through Ethernet, WIFI, USB OTG, USB disk and other transmission media, and then the driver can be loaded using the insmod command.

4). Use of peripheral controllers

Different peripheral devices have their own independent driver code and architecture implementation. When modifying and debugging different peripheral

drivers, they need to comply with their respective driver frameworks.For example, touch screen and keyboard need to use INPUT driver architecture;ADC and DAC use IIO architecture, display devices use DRM driver architecture and so on. This section does not explain driver development in detail.

6.4.3. User space uses GPIO pins

The Linux operating system architecture is divided into user-mode and kernel-mode (or user-space and kernel).User mode is the activity space of the upper application program. The execution of the application program must rely on the resources provided by the kernel, including CPU resources, storage resources, I/O resources, etc.In order for upper-layer applications to access these resources, the kernel must provide an interface for upper-layer applications to access them: system calls.

Shell is a special application program, commonly known as the command line, is essentially a command interpreter, it calls the system, through a variety of applications.With Shell scripts, a few lines of Shell scripts can do a lot of work, because Shell statements often encapsulate system calls.To facilitate user interaction with the system.

This section describes three basic ways to control GPIO pins in user mode.

- Shell command
- The system calls
- Library function

1). Shell implements pin control

Shell control pins are essentially realized by calling the file operation int erfaces provided by Linux. This section does not give detailed explanation. Pl ease refer to section 3.1 of MYD-YT507H_Linux Software Evaluation Guide for description.

2). Library functions implement pin control

Starting with Linux version 4.8, Linux introduced a new GPIO operation, the GPIO character device.Instead of using the old /sysfs method of operating GPIO in the

"/sys/class/gpio "directory, a character device based on the" file descriptor "has a corresponding gpiochip file under "/dev" for each GPIO group. For example, "/dev/gpiochip0 corresponds to GPIOA, /dev/gpiochip1 corresponds to GPIOB", etc.

Libgpiod library function implementation due to the gpiochip way, based on the C language, so the developer implemented Libgpiod, provides some tools and a simpler C API interface.Libgpiod (Library General Purpose Input/Output Device) provides a full API for developers, as well as user-space applications to manipulate GPIO.

Libgpiod interface description:

- gpiodetect - Lists all GPIochips present in the system, their name, label, and GPIO line count.
- gpioinfo - Lists all rows of the specified gPIochips, their names, users, directions, active status, and additional flags.
- gpioget - Reads the specified GPIO line value.
- gpioset - Sets the specified GPIO line values, potentially holding those lines exported and waiting for timeout, user input, or signals.
- gpiofind - Finds the gpiochip name and row offset for the given row name.
- gpiomon - Waits for events on the GPIO line, specifying which events to watch, how many events to process before exiting, or whether events should be reported to the console.

More description to view libgpiod source https://git.kernel.org/pub/scm/libs/libgpiod/libgpiod.git/.

The following code control examples of C language will be implemented using PF14 as operation GPIO pins (alternating high and low).

```
//example-gpio.c
#include  <errno.h>
```

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include <linux/gpio.h>

int main(int argc, char **argv)
{
        struct gpiohandle_request req;
        struct gpiohandle_data data;
        char chrdev_name[20];
        int fd, ret;

        strcpy(chrdev_name, "/dev/gpiochip5");

        /* Open device: GPIochip5 for GPIO bank F */
        fd = open(chrdev_name, 0);
        if (fd == -1) {
             ret = -errno;
             fprintf(stderr, "Failed to open %s\n", chrdev_name);

        return ret;
        }

        /* request GPIO line: GPIO_F_14 */
        req.lineoffsets[0] = 14;
        req.flags = GPIOHANDLE_REQUEST_OUTPUT;
        memcpy(req.default_values, &data, sizeof(req.default_values));
        strcpy(req.consumer_label, "gpio_f_14");
        req.lines = 1;
```

```
        ret = ioctl(fd, GPIO_GET_LINEHANDLE_IOCTL, &req);
        if (ret == -1) {
            ret = -errno;
            fprintf(stderr,"Failed to issue GET LINEHANDLE IOCTL (%d)\n",ret);
        }
        if (close(fd) == -1)
            perror("Failed to close GPIO character device file");

        /* Start GPIO ctr*/
        while(1) {
            data.values[0] = !data.values[0];
            ret = ioctl(req.fd, GPIOHANDLE_SET_LINE_VALUES_IOCTL, &data);
            if (ret == -1) {
                ret = -errno;
                fprintf(stderr,"Failed to issue %s (%d)\n", ret);
            }
            sleep(1);
        }

        /* release line */
        ret = close(req.fd);
        if (ret == -1) {
            perror("Failed to close GPIO LINEHANDLE device file");
            ret = -errno;
        }
        return ret;
}
```

Copy the above code into an example-gpio.c file and load the SDK environment variables into the current shell:

```
PC $export PATH= $PATH:/opt/gcc-linaro-7.4.1-2019.02-x86_64_aarch64-Linux-gnu/bin
```

The executable example-gpio can be generated using the compile command $CC.

```
PC$ aarch64-linux-gnu-gcc example-gpio.c -o example-gpio
```

Copy the executable file to the /usr/sbin directory on the development board over the network (such as scp) or usb flash drive. You can run the command directly on the terminal.

```
root@myir:~# example-gpio
```

3). System calls implement pin control

The operating system provides a set of "special" interfaces to be invoked by user programs.The user program can obtain the services provided by the operating system kernel through this set of "special" interfaces. For example, the user can request the system to open, close, or read files through file system-related calls, and obtain system time or set timers through clock-related system calls.

Pins are also resources and can be controlled through system calls.In 6.3.2, we have completed the implementation of the pin driver, which can control the pin controlled by the driver.

```
//gpiotest.c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

/ *
 * ./gpiotest /dev/myir_gpioctr on
 * ./gpiotest /dev/myir_gpioctr off
 * /
int main(int argc, char **argv)
{
```

```
int fd;
char status;

/* 1. Check parameters */
if (argc != 3)
{
    printf("Usage: %s <dev> <on | off>\n", argv[0]);
    return -1;
}

/* 2. Open file */
fd = open(argv[1], O_RDWR);
if (fd == -1)
{
    printf("can not open file %s\n", argv[1]);
    return -1;
}

/* 3. Write files */
if (0 == strcmp(argv[2], "on"))
{
    status = 1;
    write(fd, &status, 1);
}
else
{
    status = 0;
    write(fd, &status, 1);
}

close(fd);

return 0;
```

```
}
```

Copy the above code to a gpiotest.c file and load the SDK environment variables into the current shell:

```
PC$ export PATH=$PATH:/opt/gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu/bin
```

Use the compile command aarch64-linux-gnU-gcc to generate the executable file gpiotest.

```
PC$ aarch64-linux-gnu-gcc gpiotest.c -o gpiotest
```

Copy the executable file to the /usr/sbin directory on the development board over the network (SUCH as SCP) or usb flash drive. You can run the command directly on the terminal (on indicates high, off indicates low).

```
root@myir:~# gpiotest /dev/myir_gpioctr on
root@myir:~# gpiotest /dev/myir_gpioctr off
```

7. How to add your application

The migration of Linux applications is usually divided into two phases, the development debugging phase and the production deployment phase.During the development and debugging phase, we can use the SDK built by MYIR to cross-compile the application we have written and then remotely copy it to the target host for testing.The production deployment phase involves writing a recipe file for the application and building a production image using Buildroot.

7.1. Makefile-based applications

A Makefile is simply a document that defines a set of compilation rules that record the details of how the source code is compiled! Once the Makefile is written, only one make command is required, and the entire project is completely compiled automatically, greatly improving the efficiency of software development.Makefiles are widely used in developing Linux programs, whether kernel, driver, or application.

Make is a command tool that interprets instructions in makefiles.It simplifies compilation. When make is executed, make searches the current directory for the Makefile (or Makefile) text file and performs the corresponding operation.Make automatically checks to see if the source code has changed and automatically updates the executable.

The following will describe the writing of Makfile and the execution of make using a practical example (implementing a key-controlled LED light switch on the MYD-YT507H development board).Makefiles have their own set of rules.

```
target ...: prerequisites...
        command
```

- A target can be an object file, an execution file, or a label.
- Prerequisites is the file or target that is required to generate the target.
- Command is the command that make needs to execute.

```
 key_led.o: key_led.c
        ${CC} -I . -c key_led.c
all: key_led.o
        ${CC} key_led.c -o target_bin
clean:
          rm -rf *.o
          rm target_bin
```

- CC: the name of the C compiler
- CXX: name of the C++ compiler
- All: usually the default target, performing the default compilation
- Clean: Is an agreed goal
- Key_led implementation code is as follows:

```
//File: Key_led.c
#include <linux/input.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

/* ./key_led /dev/input/event0 noblock */
int main(int argc, char **argv)
{
        int fd,bg_fd;
        int err, len, i;
        unsigned char flag;
        unsigned int data[1];
```

```
char *bg = "/sys/class/leds/heartbeat/brightness";

struct input_event event;

if (argc < 2)
{
    printf("Usage: %s <dev> [noblock]\n", argv[0]);
    return -1;
}

if (argc == 3 && !strcmp(argv[2], "noblock"))
{
    fd = open(argv[1], O_RDWR | O_NONBLOCK);
}
else
{
    fd = open(argv[1], O_RDWR);
}
if (fd < 0)
{
    printf("open %s err\n", argv[1]);
    return -1;
}

while (1)
{
    len = read(fd, &event, sizeof(event));
    if (event.type == EV_KEY)
    {
            if (event.value == 1)//key down and up
            {

                printf("key test \n");
```

```
                        bg_fd = open(bg, O_RDWR);
                        if (bg_fd < 0)
                            {
                                printf("open %d err\n", bg_fd);
                                return -1;
                        }
                        read(bg_fd,&flag,1);
                        if(flag == '0')
                                system("echo 1 > /sys/class/leds/heartbeat/brig
htness"); //led off
                        else
                              system("echo 0 > /sys/class/leds/heartbeat/brightnes
s");//led on
                        }

                    }

            }
              return 0;
}
```

Compile and generate the executable target_bin on the target machine using the make command.

Load and compile the cross toolchain environment variable to the current shell:

Perform the make:

```
PC$ make
```

As you can see from the results of the previous command, the compiler used is the one created by setting CC variables defined in the script.

Copy the target_bin executable file to /usr/sbin on the development board via network (scp, etc.) or usb disk:

```
root@myir:~#  target_bin  /dev/input/event0  noblock
```

Note: If the target_bin is built using the cross-toolchain compiler, and the architecture of the build host is different from that of the target machine, you need to run the project on the target device.

7.2. Qt-based applications

Qt is a cross-platform graphics application development framework that can be used on different sizes of devices and platforms. Qt also provides users with different copyright versions to choose from. MYD-YT507H uses Qt 5.12 for application development.In Qt application development, it is recommended to use QtCreator integrated development environment, which can develop Qt applications under Linux PC and automatically cross-compile them into ARM architecture programs of development board.

1). QtCreator installation and configuration

From QT website qtcreator installation package download QT's official website: http://download.qt.io/development_releases/qtcreator/4.1/4.1.0-rc1/.

Qt-creator-openSource-linux-x86_64-4.1.0-rc1.run /qt-creator-opensource-linux-x86_64-4.1.0-rc1. If you want to get installation and configuration details, please see the MYD-YT507H QT application development notebook or QTCreator official website to get more development guide https://www.qt.io/product/development-tools.

2). MEasy HMI2.0 is compiled and running

MEasy HMI 2.0 is a QT5-based HMI framework developed by MYIR Technology Co., LTD.The project uses a mixture of QML and C++ programming to efficiently and conveniently build the UI, while C++ is used to implement business logic and complex algorithms.

The MEasy HMI2.0 project source code "MYD-YT507H-2022xxx\04-sources \mxapp2.tar.gz" is available in mill's software distribution package.It can be loaded and compiled via QT Creator, remote debugging, etc. Please refer to *"MYD-YT507H QT Application Development Notes"*.

Reference

- **Linux Kernel open source community**
  https://www.kernel.org/
- **Buildroot website**
  https://buildroot.org/

Appendix A

Warranty & Technical Support Services

MYIR Electronics Limited is a global provider of ARM hardware and software tools, design solutions for embedded applications. We support our customers in a wide range of services to accelerate your time to market.

MYIR is an ARM Connected Community Member and work closely with ARM and many semiconductor vendors. We sell products ranging from board level products such as development boards, single board computers and CPU modules to help with your evaluation, prototype, and system integration or creating your own applications. Our products are used widely in industrial control, medical devices, consumer electronic, telecommunication systems, Human Machine Interface (HMI) and more other embedded applications. MYIR has an experienced team and provides custom design services based on ARM processors to help customers make your idea a reality.

The contents below introduce to customers the warranty and technical support services provided by MYIR as well as the matters needing attention in using MYIR’ s products.

Service Guarantee

MYIR regards the product quality as the life of an enterprise. We strictly check and control the core board design, the procurement of components, production control, product testing, packaging, shipping and other aspects and strive to provide products with best quality to customers. We believe that only quality products and excellent services can ensure the long-term cooperation and mutual benefit.

Price

MYIR insists on providing customers with the most valuable products. We do not pursue excess profits which we think only for short-time cooperation. Instead, we hope to establish

long-term cooperation and win-win business with customers. So we will offer reasonable prices in the hope of making the business greater with the customers together hand in hand.

Delivery Time

MYIR will always keep a certain stock for its regular products. If your order quantity is less than the amount of inventory, the delivery time would be within three days; if your order quantity is greater than the number of inventory, the delivery time would be always four to six weeks. If for any urgent delivery, we can negotiate with customer and try to supply the goods in advance.

Technical Support

MYIR has a professional technical support team. Customer can contact us by email (support@myirtech.com), we will try to reply you within 48 hours. For mass production and customized products, we will specify person to follow the case and ensure the smooth production.

After-sale Service

MYIR offers one year free technical support and after-sales maintenance service from the purchase date. The service covers:

Technical support service

MYIR offers technical support for the hardware and software materials which have provided to customers:

- To help customers compile and run the source code we offer;
- To help customers solve problems occurred during operations if users follow the user manual documents;
- To judge whether the failure exists;
- To provide free software upgrading service.

However, the following situations are not included in the scope of our free technical support service:

- Hardware or software problems occurred during customers' own development;
- Problems occurred when customers compile or run the OS which is tailored by themselves;
- Problems occurred during customers' own applications development;
- Problems occurred during the modification of MYIR's software source code.

After-sales maintenance service

The products except LCD, which are not used properly, will take the twelve months free maintenance service since the purchase date. But following situations are not included in the scope of our free maintenance service:

- The warranty period is expired;
- The customer cannot provide proof-of-purchase or the product has no serial number;
- The customer has not followed the instruction of the manual which has caused the damage the product;
- Due to the natural disasters (unexpected matters), or natural attrition of the components, or unexpected matters leads the defects of appearance/function;
- Due to the power supply, bump, leaking of the roof, pets, moist, impurities into the boards, all those reasons which have caused the damage of the products or defects of appearance;
- Due to unauthorized weld or dismantle parts or repair the products which has caused the damage of the products or defects of appearance;
- Due to unauthorized installation of the software, system or incorrect configuration or computer virus which has caused the damage of products.

Warm tips

1. MYIR does not supply maintenance service to LCD. We suggest the customer first check the LCD when receiving the goods. In case the LCD cannot run or no display, customer should contact MYIR within 7 business days from the moment get the goods.

2. Please do not use finger nails or hard sharp object to touch the surface of the LCD.

3. MYIR suggests user purchasing a piece of special wiper to wipe the LCD after long time use, please avoid clean the surface with fingers or hands to leave fingerprint.

4. Do not clean the surface of the screen with chemicals.

5. Please read through the product user manual before you using MYIR’ s products.

6. For any maintenance service, customers should communicate with MYIR to confirm the issue first. MYIR’ s support team will judge the failure to see if the goods need to be returned for repair service, we will issue you RMA number for return maintenance service after confirmation.

Maintenance period and charges

- MYIR will test the products within three days after receipt of the returned goods and inform customer the testing result. Then we will arrange shipment within one week for the repaired goods to the customer. For any special failure, we will negotiate with customers to confirm the maintenance period.
- For products within warranty period and caused by quality problem, MYIR offers free maintenance service; for products within warranty period but out of free maintenance service scope, MYIR provides maintenance service but shall charge some basic material cost; for products out of warranty period, MYIR provides maintenance service but shall charge some basic material cost and handling fee.

Shipping cost

During the warranty period, the shipping cost which delivered to MYIR should be responsible by user; MYIR will pay for the return shipping cost to users when the product is repaired. If the warranty period is expired, all the shipping cost will be responsible by users.

Products Life Cycle

MYIR will always select mainstream chips for our design, thus to ensure at least ten years continuous supply; if meeting some main chip stopping production, we will inform customers in time and assist customers with products updating and upgrading.

Value-added Services

1. MYIR provides services of driver development base on MYIR’ s products, like serial port, USB, Ethernet, LCD, etc.

2. MYIR provides the services of OS porting, BSP drivers’ development, API software development, etc.

3. MYIR provides other products supporting services like power adapter, LCD panel, etc.

4. ODM/OEM services.

MYIR Electronics Limited

Room 04, 6th Floor, Building No.2, Fada Road,

Yunli Inteiligent Park, Bantian, Longgang District.

Support Email: support@myirtech.com

Sales Email: sales@myirtech.com

Phone: +86-755-22984836

Fax: +86-755-25532724

Website: www.myirtech.com