MYD-YT507H Android System Development Guide

File Status:	FILE ID:	MYIR-MYD-YT507H-SW-DG-EN-A4.9.170
[ ] Draft	VERSION:	V1.0[DOC]
[√] Release	AUTHOR:	Licy
	CREATED:	2022-09-21
	UPDATED:	2022-11-18

Revision History

VERSION	AUTHOR	PARTICIPANT	DATE	DESCRIPTION
V1.0[DOC]	Licy		20220921	Initial version

CONTENT

1. Overview

Android operating system is a free and open source mobile operating system based on Linux kernel developed by Google. This system is mainly used for mobile devices, such as smartphones and tablets.

The hierarchical architecture of the Android system is very clear, and its platform consists of five parts: application, application framework, system library, Android runtime, and Linux kernel. As shown in the figure below:

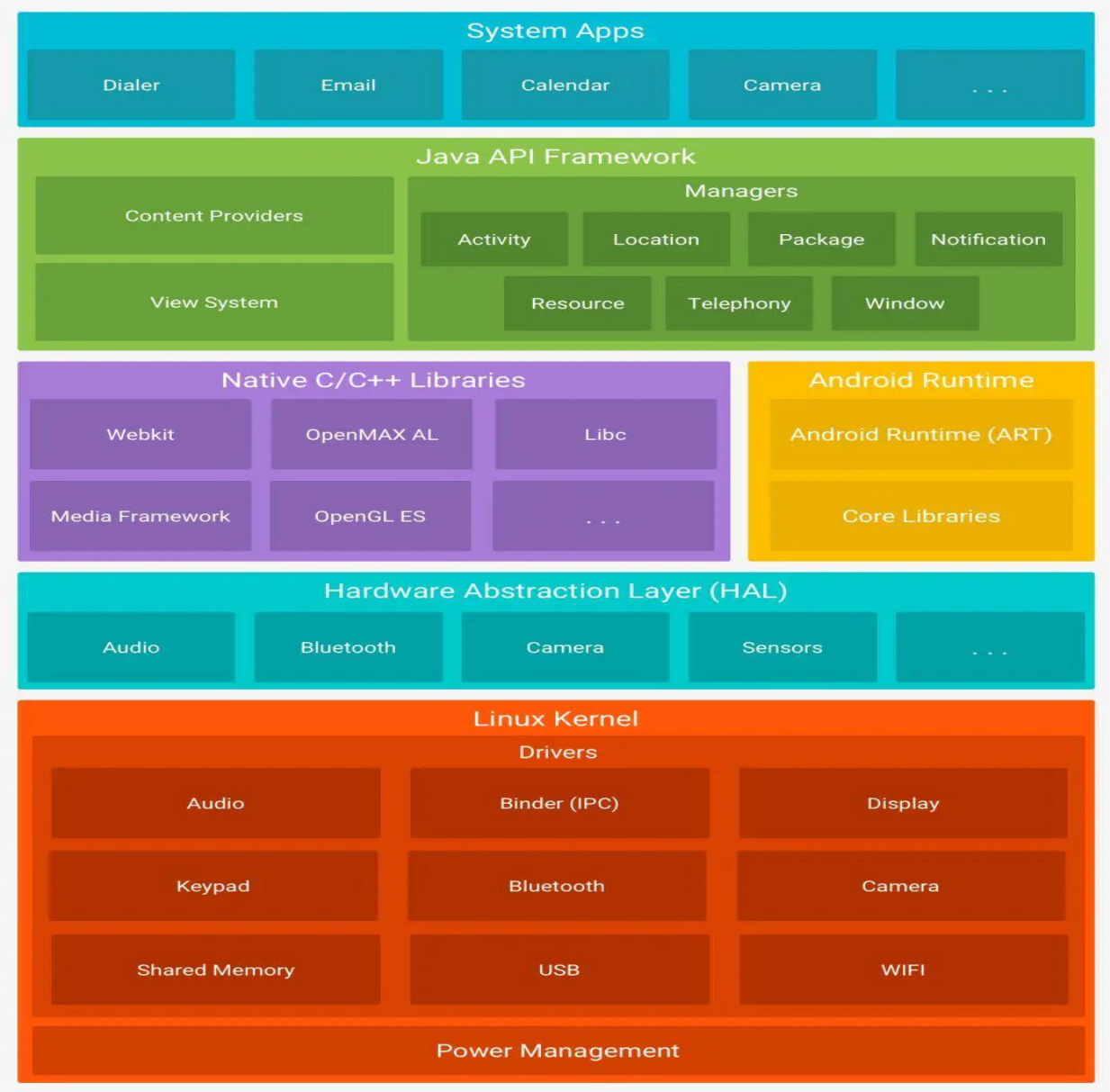

Figure 1-1. Android Framework

- System Apps

The Android platform includes by default the main applications, including email, SMS, calendar, maps, browser, contacts, etc. These programs are written in Java language, but of course you can also write your own software to replace the programs provided by Android.

- API FRAMEWORK

The Android application framework is the basis for developers to develop. It consists of 9 main parts: view system, content provider, window manager, activity manager, notification manager, location manager, resource manager, phone manager and package manager.

- LIBRARIES

It includes two parts, core library and Android runtime.

  - Core library:

Android contains a collection of C/C++ libraries for use by the various components of the Android system. Its available to developers through Android's application framework. It includes system C libraries, media libraries, interface management libraries, graphics libraries, database engine, font libraries, etc.

  - Runtime:

Although Android uses Java language to write applications, it does not use J2ME to execute Java programs, but uses Android runtime which is used by Android itself.Android runtime includes two parts: core library and Dalvik virtual machine.

- HAL（Hardware Abstraction）

HAL is located at the interface layer between the operating system kernel and the hardware circuitry, and its purpose is to abstract the hardware and hide the platform-specific hardware interface details.

- LINUX KERNEL

The operating system in the Android platform uses Linux as the kernel, which includes a display driver, camera driver, Flash memory driver, Binder (IPC) driver, keyboard driver, WIFI driver, Audio driver, and a power management section.

This article introduces the complete process of customizing a complete embedded Android system based on Allwinner T507-H Android Q(10) SDK project

and MYIR Core Board, which includes the preparation of development environment, code acquisition, and how to port Bootloader, Kernel, customize android to suit your application needs. file system for your application. We first introduce how to build the system image for MYD-YT507H development board based on the source code we provide, and how to burn the built image to the development board. For those who are developing projects based on the MYC-YT507H core board, we focus on the method and some key points of porting this set of system to the user's hardware platform, so that users can quickly customize the system image to fit their hardware.

This document does not contain an introduction to the open source Android project and Linux kernel related basics, and is suitable for embedded Linux system developers and embedded Linux BSP developers and Android system engineers who have some development experience. For some specific functions that users may use in the process of secondary development, we will also provide application notes for R&D personnel to refer to as appropriate, see the document list in Table 2-5 of "**MYD-YT507H Android SDK Release Notes**" for specific information.

1.1. Software Resources

MYD-YT507H can be equipped with Android 10 system, which provides rich system resources and other software resources. This includes documentation, code and various development and debugging tools for Windows desktop environment and PC Linux system, application development routines, etc. For specific included software information, please refer to the description in Chapter 2 Software Information in the "**MYD-YT507H Android SDK Release Notes**".

1.2. Document Resources

Depending on the user's use of the development board for each different purpose. Different types of documents and manuals such as Release Notes, Getting Started Guide, Evaluation Guide, Development Guide, Application Notes, and Frequently Asked Questions will be provided to customers. The specific list of documents is described in Table 2-5 of the "**MYD-YT507H Android SDK Release Notes**".

2. Development Environment

This chapter introduces some hardware and software environments required for the development process based on the MYD-YT507H development board, including the necessary development host environment, essential software tools, code and data acquisition, etc. Specific preparations will be described in detail below.

2.1. Developing Host Environment

How to build the development environment for Allwinner T5 series processor platform. By reading this chapter, you will understand the installation and use of related hardware tools, software development and debugging tools. You will also be able to quickly set up the relevant development environment to prepare for the development and debugging later. Allwinner T5 series processors are SMP multi-core architecture processors with 4 ARM Cortex A53 cores, which can run Android system.

- **Host Hardware**

The build of the entire SDK package project has high requirements for the development host, requiring a processor with a dual-core CPU or higher, 8GB or more of memory, and a 200GB hard drive or higher configuration. It can be a PC or server with a Linux system installed, or a virtual machine running Linux.

- **Host Operating System**

There are many choices of host operating system for building Android projects, usually you choose to build on the local host with Fedora, openSUSE, Debian, Ubuntu, RHEL or CentOS Linux distributions installed, here we recommend Ubuntu 20.04 64bit desktop system (Ubuntu18.04 64bit and Ubuntu21.04 64bit are both available), and subsequent development will be introduced with this system as an example.

- **Installation of essential packages**

Install the necessary development dependencies on the host side first

```
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
sudo apt-get install openjdk-8-jdk
```

The above packages need to be installed manually by the user, if you want to install them all automatically you can copy all the commands into a script file.

Other non-required configuration packages

```
sudo dpkg-reconfigure dash #choose 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 # Install if libz.so is missing
sudo apt-get install u-boot-tools
```

2.2. Introduction of Software Development Tools

In the process of customizing the applicable Linux system and debugging will need to use a number of debugging, burning tools, provided under the CD-ROM image directory */03-Tools/* in MYIR provides some of the tools, briefly described as follows:

- **Driver Software Installation：**

In a Windows environment, the USB device driver is automatically installed when the target board device is powered up and the USB cable is plugged in. If the installation is successful, the device Android Phone, identified by the red oval in the figure below, will appear in the Windows Device Manager.

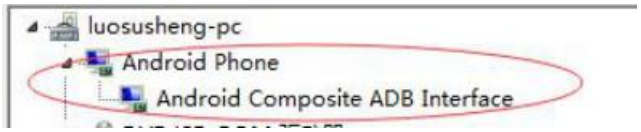

Figure 2-1. Driver Identification

- **Software Burning Installation：**

Software PhoenixSuit(v1.19)：Available for USB online burn-in systems

Software PhoenixCard(4.2.4)：It can be used for Micro SD burn-in card creation and Micro SD boot card creation.

The installation packages for these two programs are located in */tools/tools_win/*, after unpacking there will be two versions in English and Chinese, just choose one. Once the sdk is compiled and packaged, it can be burned by PhoenixSuit, as described in detail later.

Note: T5 requires PhoenixSuit (v1.19) and PhoenixCard (4.2.4) or higher. Otherwise, it may cause problems such as inability to perform card upgrades.

3. The Android SDK Structure

3.1. Introduction

Android SDK development kit, which integrates BSP, android system, standalone IP and test, can be used both as a BSP development and IP verification platform and as an embedded Android system for mass production.

The functionality of the Android SDK includes the following four parts:

- BSP development, including bootloader, uboot and kernel.
- Android file system development, including mass-produced embedded android systems.
- IP verification and distribution platform, including gpu, cedarx, gstreamer, security and other private packages. It also gives the IP usage and system integration demos for quick use by third parties.
- Testing, including board-level testing and system testing.

The android SDK files and data for the MYD-YT507H development board are available in the */04_sources/* directory in the iso image provided by MYIR, to help developers build an android system image that can run on the MYD-YT507H development board.

3.2. Get the Source Code

We provide two ways to get the source code, one is to get the zip package directly from the MYIR CD image */04-sources/* directory, the other is to use repo to get the source code located on github live updates to build, please choose one of them to build according to the actual needs.

3.2.1. Get Compressed Source Code(Preferred)

The compressed source package is located at MYIR Development Kit Profile */04-Sources/YT507H-Android10-t5-A4.9.170-X.X.X.tar.bz2/* (X.X.X stands for the current version number). Copy the tarball to a user-specified directory, such as the "$HOME/work/t507-android" directory, which will be used as the top-level

directory for subsequent builds, and all directories of the SDK appear after unpacking as follows:

```
PC$ cd $HOME/work/t507-android
PC$ tar -jxvf YT507H-Android10-t5-A4.9.170-X.X.X.tar.bz2
android  longan
```

- android Directory for AOSP source and HAL source package.
- longan Directory for BSP source package.

3.2.2. Get Source Code from GitHub

Currently the BSP source code and Buildroot source code of MYD-YT507H development board are hosted on github and will be kept updated for a long time, please see "MYD-YT507H Android SDK Release Notes" for the code repository address. Users can use repo to get and synchronize the code on github. Here's how to do it:

```
PC$ mkdir  $HOME/work/t507-android
PC$ cd $HOME/work/t507-android
PC$ export REPO_URL='https://mirrors.tuna.tsinghua.edu.cn/git/git-repo/'
PC$ repo init -u https://github.com/MYIR-ALLWINNER/myir-t5-android-manifest.
git --no-clone-bundle --depth=1 -b master -m myir-t5-android10-1.0.0.xml
PC$ repo sync
. . .
PC$ android  longan
```

After successful code synchronization, you will also get an SDK folder in the "$HOME/work/t507-android" directory, which contains the path to the source code or source code repository related to the MYD-YT507H development board, with the same directory structure as the one extracted from the zip package.

3.2.3. Understand the BSP Structure

longan is a linux BSP package.

```
├── brandy
├── build
├── build.sh
├── device
```

```
├── kernel
├── out
├── test
└── tools
```

It is mainly composed of brandy, kernel.

- brandy includes uboot2018;
- kernel is linux 4.9.170 kernel;

3.2.4. kernel

The linux kernel source code directory. The current kernel version is linux 4.9.170, and the above directory structure is consistent with the standard linux kernel except for the modules directory, which is where we store external modules that are not integrated with the kernel's menuconfig.

3.2.5. brandy

There is a brandy2.0 version in the brandy directory, currently T507 uses brandy2.0 version and its directory structure is:

```
├── spl-pub
├── tools
└── u-boot-2018
```

Note: The default code compilation process does not compile uboot, and users need to compile it themselves when they need to modify uboot. The specific compilation method is described later.

3.2.6. tools

tools_win is a tool for compiling and packaging, tools_win is a tool for burning on the PC side, etc.

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

3.2.7. test

Dragonboard provides fast board-level testing.

3.2.8. device

```
├── config
│       ├── chips
│       │   └── t507
│       ├── common
│       │   ├── debug
│       │   ├── dtb
│       │   ├── hdcp
│       │   ├── imagecfg
│       │   ├── partition
│       │   ├── sign_config
│       │   ├── toc
│       │   ├── tools
│       │   └── version
│       └── rootfs_tar
├── bin     (Startup files used during packaging)
│       ├── bl31.bin
│       ├── boot0_nand_sun50iw9p1.bin       (boot0 boot file for nand)
│       ├── boot0_sdcard_sun50iw9p1.bin     (boot0 boot file for emmc)
│           ├── fes1_sun50iw9p1.bin          (Initialization file for burn tool)
│       ├── optee_sun50iw9p1.bin     (optee)
│       ├── sboot_sun50iw9p1.bin     (Safe boot bin, not supported at this time)
│       ├── u-boot-sun50iw9p1.bin    (uboot 的 bin)
├── boot-resource
│       ├── boot-resource
│       │   ├── bat
│       │   ├── bootlogo.bmp  (Boot screen, update bootlogo to replace this file)
│       └── boot-resource.ini
├── configs
│       ├── default (Default platform-based configuration)
```

```
|       |   ├── BoardConfig.mk
|       |   ├── boot_package.cfg  (Boot package content configuration for boot)
|       |   ├── diskfs.fex
|       |   ├── dragon_toc_android.cfg
|       |   ├── dragon_toc.cfg
|       |   ├── env_burn.cfg
|       |   ├── env.cfg     (Boot parameters passed from uboot to linux)
|       |   ├── env_dragon.cfg
|       |   ├── image.cfg
|       |   ├── image_dragonboard.cfg
```

4. Configuration and Build of the Android System

The previous chapters have described the complete process of building a system image based on the android SDK project running on the MYD-YT507H development board, and burning the image to the development board. Since many of the pins of the MYC-YT507H core board have the feature of multiple functions, there will always be some differences between the base board designed based on the MYC-YT507H core board and the MYB-YT507H in the actual project. These differences may be the removal of the display, adding more GPIOs, or the need to add more serial ports, and possibly expand some peripherals through SPI, I2C, USB, etc.; this chapter describes the specific process of developing and customizing your own system from a system developer's point of view, laying the foundation for adapting your own hardware later.

4.1. Board Level 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 the hardware features on the device and kernel configuration information, as well as any other hardware drivers required.In some cases, the BSP contains individually licensed intellectual property (IP) for one or more components.

Usually according to the different stages of hardware boot, we divide the BSP into Bootloader part and Kernel part. The hardware BSP code designed with MYC-YT507H core board can be viewed in the part of linux SDK.

Brandy contains only the Bootloader part of the spl and u-boot, this part mainly implements the core hardware, such as DDR, Clock initialization and the kernel boot. Based on the MYC-YT507H core board hardware to modify this part of the content.

```
├── spl-pub
├── tools
```

```
└── u-boot-2018
```

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

```
kernel/
├── linux 4.9
```

When designing a product using MYIR's core board, the bootloader section can be left unmodified if there are no special needs. You need to pay more attention to the development and debugging of the product kernel driver and the design of the application software. Subsequent chapters will describe kernel development and application development in detail.

4.2. Configuration and Build of Board-level Support Packages

If you need to compile the board-level support package, you need to perform the following steps and go to the longan directory.

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

Execute . /build.sh config and select the configuration in the subsequent dialogue . Choice can be selected by either entering a number.

Enter the string corresponding to the number.

```
PC/longan$ ./build.sh config
Welcome to mkscript setup progress
All available platform:
0. android   //Android option
1. linux      //linux option
Choice [linux]:0
All available ic:
  0. t507
Choice [myir]: 0
All available board:
  0. evb
  1. evb2
```

```
Choice [evb]: 0
```

Go to the longan directory and execute the following command.

```
PC/longan$ ./build.sh
```

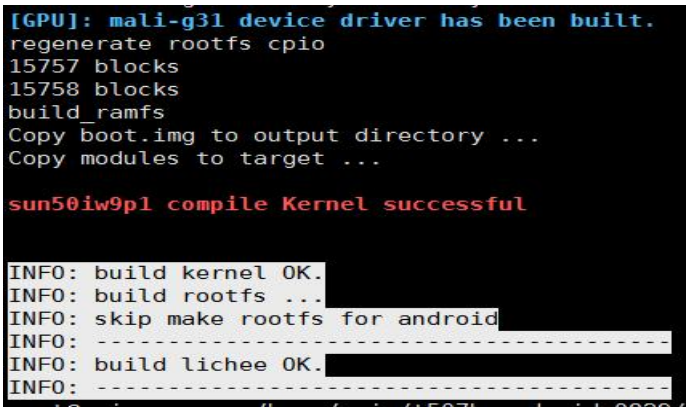

Figure 4-1. BSP Construction Completed

Table 4-1. Description of Parameters

Type	Command	Description
Overall compilation	./build.sh config	Compile configuration, pop-up compile selection
	./build.sh autoconfig	Compile configuration according to the parameters passed in, without popping up the compile selection
	./build.sh	Compile the SDK according to the compilation configuration
	./build.sh clean	Clear process files and target files
	./build.sh distclean	Clear all generated files
Partial compilation	./build.sh brandy	Compile brandy(uboot)
	./build.sh kernel	Compiling the kernel

Compilation problem analysis:

Executing . /build.sh config gives the following problem

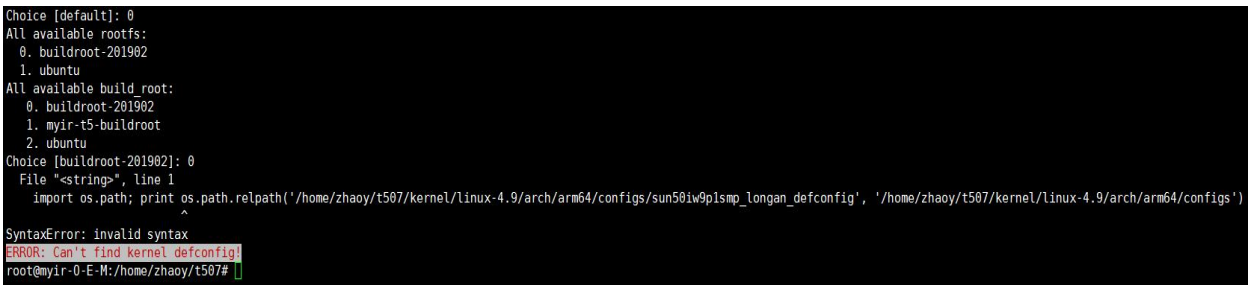

Figure 4-2. Configuration Failure

```
  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 4-3. Failed to Compile the Kernel

Since python2 is required to configure the kernel, please check if python2 is already installed in your development environment; if not, please check section 2.1 or execute the following command：

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

Check if the current version is python2

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

Please switch to python2 version if it is not.

4.3. U-boot Compilation and Update

U-boot is a very feature-rich open source bootloader, including kernel boot, download updates and many other aspects of the embedded field is very widely used, check the official website for more information http://www.denx.de/wiki/U-Boot/WebHome

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

4.3.1. Compile U-Boot Under the Linux Bsp Project

Once the user has modified the U-boot code according to the iterative development process in this document，he entire image can also be built using the SDK.

Compiling uboot source code：

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

Packed files:

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

4.4. Kernel Compilation and Update

Linux kernel is a very large open source kernel that is used in various distributions of operating systems. Linux kernel is widely adopted in embedded systems due to its portability, multiple network protocol support, independent module mechanism, MMU and many other rich features.

At the same time T5 also supports Linux kernel, will get long-term stable updates, MYD-YT507H using T5 kernel port, the latest support for Linux kernel version 4.9.170.

4.4.1. Compile Kernel in a Standalone Cross-compiler Environment

4.4.1.1. Obtaining Kernel Source Code

Copy the development package */04-Source/YT507H-android10-t5-A4.9.170-X.X.X.tar.bz* (X.X.X represents the current version) to the specified custom directory, unzip it into the source directory and check the corresponding file information

```
PC/longan$ cd kernel/
```

The catalog contains:

- Source Code: Soft Links Directory: linux-4.9
- Source Code: myir-t5-kernel

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

4.4.1.2. Configuring the Kernel (Optional)

MYIR has integrated most of the features into the kernel and generally does not require configuration. To add special features, the peripheral drivers should be configured as follows.

- **Go to the Kernel Directory**

```
PC/longan$ cd kernel/linux-4.9
```

- **Create the Output Folder Build**

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

- **Configuring the Kernel**

```
PC/longan$ make ARCH=arm64 O="$PWD/../build" sun50iw9p1smp_longan_defconfig
```

If you need to configure the kernel or want to turn on one of the kernel driver functions, you can also use the following method.

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

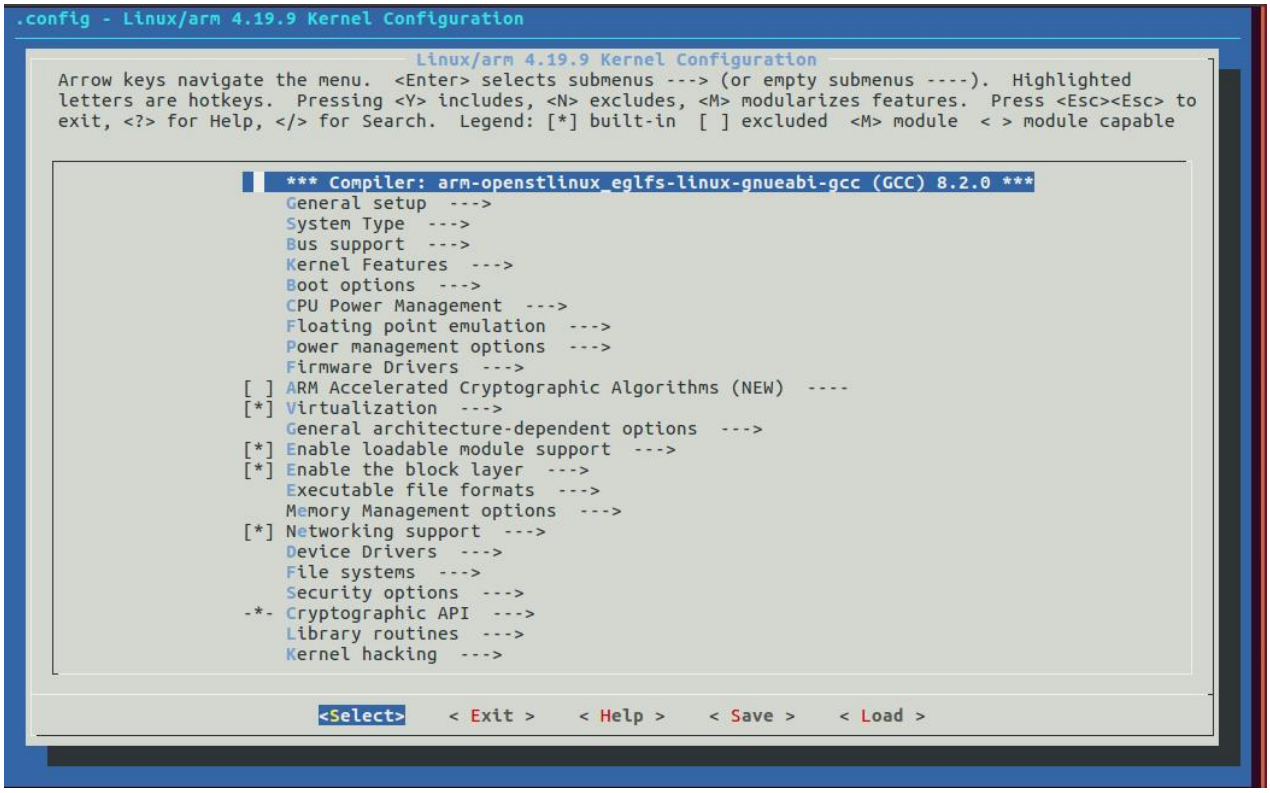

Figure 4-4. Kernel Configuration

4.4.1.3. Compiling with Linux BSP

Compiling kernel source code:

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

Packaged files:

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

4.5. Build of the Android System

Since you need to rely on the BSP file built by longan when building the android file system, you need to build the BSP file first when building the android burner package.

4.5.1. Understand the Source Code Structure of Android

```
├── Android.bp -> build/soong/root.bp
├── art
├── bionic
├── bootable
├── bootstrap.bash -> build/soong/bootstrap.bash
├── build
├── cts
├── dalvik
├── developers
├── development
├── device
├── external
├── frameworks
├── hardware
├── kernel
├── libcore
├── libnativehelper
├── Makefile
├── out
├── packages
├── pdk
├── platform_testing
├── prebuilts
├── repo
```

```
├── sdk
├── system
├── test
├── toolchain
├── tools
└── vendor
```

- art：New ART operating environment
- bionic： Google development system C library, open source under BSD license (c++)
- Bootable：System boot-related code
- build：Storage system compilation rules and configuration of basic development pac kages such as generic
- cts android：Compatibility Test Suite Standards
- dalvik：Dalvik Virtual Machine
- developer：Developer Directory
- development：Related to application development
- device： Equipment-related configuration
- docs：Reference Document Catalog
- external：Open Source Module Related
- frameworks：Application framework, core part of Android system, written in java and c++
- hardware：Mainly the code of the hardware abstraction layer
- libcore：Core library related documents
- libnativehelper：Dynamic libraries to implement the foundations of JNI
- out：The compiled code is entered in this directory
- pdk：Abbreviation for Plug Development Kit, local development kit
- paltform_test：Platform Testing
- prebuilts：Some source code compiled under x86 and ARM architecture
- sdk：SDK and Emulator
- package：Application packages
- system：Strata file system libraries, applications and components

- toolchain：Toolchain files
- tools：Tool files
- makefile：Global Makefile, used to define compilation rules

4.5.2. Configuring the Android Build Environment

You can start to configure and build the system by going to the android directory.

- Configuring the Environment

Use the command：source build/envsetup.sh

```
PC/android$ source build/envsetup.sh
including device/softwinner/common/vendorsetup.sh
```

- Configure the Device

Use the command：lunch

```
PC/android$ lunch

You're building on Linux

Lunch menu... pick a combo:
    1. aosp_arm-eng
    2. aosp_arm64-eng
    3. aosp_blueline-userdebug
    4. aosp_bonito-userdebug
    5. aosp_car_arm-userdebug
    6. aosp_car_arm64-userdebug
    7. aosp_car_x86-userdebug
    8. aosp_car_x86_64-userdebug
    9. aosp_cf_arm64_phone-userdebug
    10. aosp_cf_x86_64_phone-userdebug
     ...
    32. mercury_evb-user
    33. mercury_evb-userdebug
    34. mercury_evb2-user
    35. mercury_evb2-userdebug
```

```
    36. mini_emulator_arm64-userdebug
    37. mini_emulator_x86-userdebug
    38. mini_emulator_x86_64-userdebug
    39. poplar-eng
    40. poplar-user
    41. poplar-userdebug
    42. qemu_trusty_arm64-userdebug
    43. uml-userdebug

Which would you like? [aosp_arm-eng] 33
```

Select 33 .mercury_evb-userdebug mode

```
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=10
TARGET_PRODUCT=mercury_evb
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm
TARGET_ARCH_VARIANT=armv7-a-neon
TARGET_CPU_VARIANT=cortex-a7
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-43-generic-x86_64-Ubuntu-20.04.5-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=QP1A.191105.004
OUT_DIR=out
============================================
```

- Get BSP Package

Use the command：extract-bsp

```
PC/android$ extract-bsp
Copy /home/myir/t507h-android-0829/longan/out/t507/evb/android/bImage to /home/myir/t507h-android-0829/android/device/softwinner/mercury-evb/kernel
Copy /home/myir/t507h-android-0829/longan/out/t507/evb/android/dtbo.img to /home/myir/t507h-android-0829/android/device/softwinner/mercury-evb/dtbo.img
Copy /home/myir/t507h-android-0829/longan/out/t507/evb/android/lib/modules/*/* to /home/myir/t507h-android-0829/android/device/softwinner/mercury-evb/modules!
Copy /home/myir/t507h-android-0829/longan/out/t507/evb/android/sunxi.dtb to /home/myir/t507h-android-0829/android/device/softwinner/mercury-evb/sunxi.dtb
```

- Building Systems

Use make command to compile Android file system (make -jx where x represents the number of compilation threads, according to the actual situation of the server to configure or not), the initial compilation time of android source code will be long, need to wait patiently：

```
PC/android$ make
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=10
TARGET_PRODUCT=mercury_evb
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm
TARGET_ARCH_VARIANT=armv7-a-neon
TARGET_CPU_VARIANT=cortex-a7
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-43-generic-x86_64-Ubuntu-20.04.5-LTS
HOST_CROSS_OS=windows
```

```
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=QP1A.191105.004
OUT_DIR=out
============================================
Starting ninja...
[ 96% 53/55] build check-all-partition-sizes
The sum of sizes of [system vendor product] is within BOARD_SUPER_PARTITION_
SIZE:
630833152+89174016+182136832 == 902144000 <=  1610612736 == 16106127
36
The sum of sizes of [system vendor product] is within BOARD_SB_SIZE:
630833152+89174016+182136832 == 902144000 <= 1602224128 == 160222412
8
The sum of sizes of [sb] is within BOARD_SUPER_PARTITION_SIZE:
1602224128 == 1602224128 <= 1610612736 == 1610612736
[100% 55/55] Target super fs image for debug: out/target/product/mercury-evb/s
uper.img
2022-09-21 18:20:26 - build_super_image.py - INFO    : Building super image from
 info dict...
2022-09-21 18:20:26 - sparse_img.py - INFO    : Total of 154012 4096-byte output
blocks in 13 input chunks.
2022-09-21 18:20:26 - sparse_img.py - INFO    : Total of 21771 4096-byte output b
locks in 7 input chunks.
2022-09-21 18:20:26 - sparse_img.py - INFO    : Total of 44467 4096-byte output b
locks in 9 input chunks.
2022-09-21 18:20:26 - common.py - INFO    :   Running: "lpmake --metadata-size
65536 --super-name super --metadata-slots 2 --device super:1610612736 --group
 sb:1602224128 --partition system:readonly:630833152:sb --image system=out/ta
rget/product/mercury-evb/system.img --partition vendor:readonly:89174016:sb --
image vendor=out/target/product/mercury-evb/vendor.img --partition product:re
```

```
adonly:182136832:sb --image product=out/target/product/mercury-evb/product.i
mg --sparse --output out/target/product/mercury-evb/super.img"
2022-09-21 18:20:36 - common.py - INFO    : lpmake I 09-21 18:20:26   713   713 b
uilder.cpp:937] [liblp]Partition system will resize from 0 bytes to 630833152 bytes
lpmake I 09-21 18:20:26   713   713 builder.cpp:937] [liblp]Partition vendor will resi
ze from 0 bytes to 89174016 bytes
lpmake I 09-21 18:20:26   713   713 builder.cpp:937] [liblp]Partition product will re
size from 0 bytes to 182136832 bytes
2022-09-21 18:20:36 - build_super_image.py - INFO    : Done writing image out/ta
rget/product/mercury-evb/super.img
[W][2022-09-21T18:18:24+0800][3240596] void cmdline::logParams(nsjconf_t *)():
260 Process will be GID/EGID=0 in the global user namespace, and will have grou
p root-level access to files

#### build completed successfully (02:30 (mm:ss)) ####
```

- Package the Image

Use the command：pack

Generate a burnable image package under the specified path /work/t507h-android/longan/out.

```
PC/android$ pack

boot-resource.fex Len: 0x753400
Vboot-resource.fex Len: 0x4
env.fex Len: 0x20000
Venv.fex Len: 0x4
boot.fex Len: 0x2000000
Vboot.fex Len: 0x4
super.fex Len: 0x35252ff4
Vsuper.fex Len: 0x4
recovery.fex Len: 0x2000000
Vrecovery.fex Len: 0x4
```

```
vbmeta.fex Len: 0x2000
Vvbmeta.fex Len: 0x4
vbmeta_system.fex Len: 0x1000
Vvbmeta_system.fex Len: 0x4
vbmeta_vendor.fex Len: 0x1000
Vvbmeta_vendor.fex Len: 0x4
dtbo.fex Len: 0x200000
Vdtbo.fex Len: 0x4
BuildImg 0
Dragon execute image.cfg SUCCESS !
----------image is at----------

/home/myir/t507h-android-0829/longan/out/t507_android10_evb_uart0.img

pack finish
use pack4dist for release
```

5. How to Burn System Image

The core board and development board of MYC-YT507H series designed by MYIR is based on Allwinner's T5 series microprocessor, which has various boot methods, so different tools and methods are needed to update the system. Users can choose different ways to update according to their needs. The update methods are mainly as follows:

- PhoenixSuit burn-in: Suitable for R&D debugging, testing and other scenarios.
- Make an SD card launcher: Suitable for R&D debugging, quick start-up and other scenarios.
- Create SD card burner: Suitable for mass production burn-in of eMMC.

5.1. How to Flash with PhoenixSuit

1). Tool Requirements

- One development board
- Two USB to Type_C cables
- 12V power adapter
- PhoenixSuit Official Software

2). Setting Up the Hardware

Connect the hardware, connect the J6 OTG interface of the development board to the computer, and plug in the power adapter.

3). Burning the System Under Windows

Use the OTG USB cable to connect the development board and the host computer, first press and hold the FEL key and then press the power on key to reset the system, release the FEL key after about two seconds. Open the windows device manager, you can find the USB device driver automatically installed and recognized.

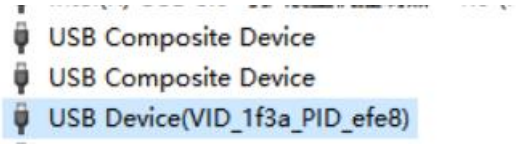

Figure 5-1. USB Connection to PC

If you find a yellow exclamation mark, the driver is not automatically installed, you can find the corresponding driver file (64bit USBDriver_64.zip and 32bit USBDriver.rar) under the SDK directory */t507/tools/tools_win/* and unzip it manually to install it.

For OTG full burn test, double-click the PhoenixSuit.exe file in the PhoenixSuit directory.

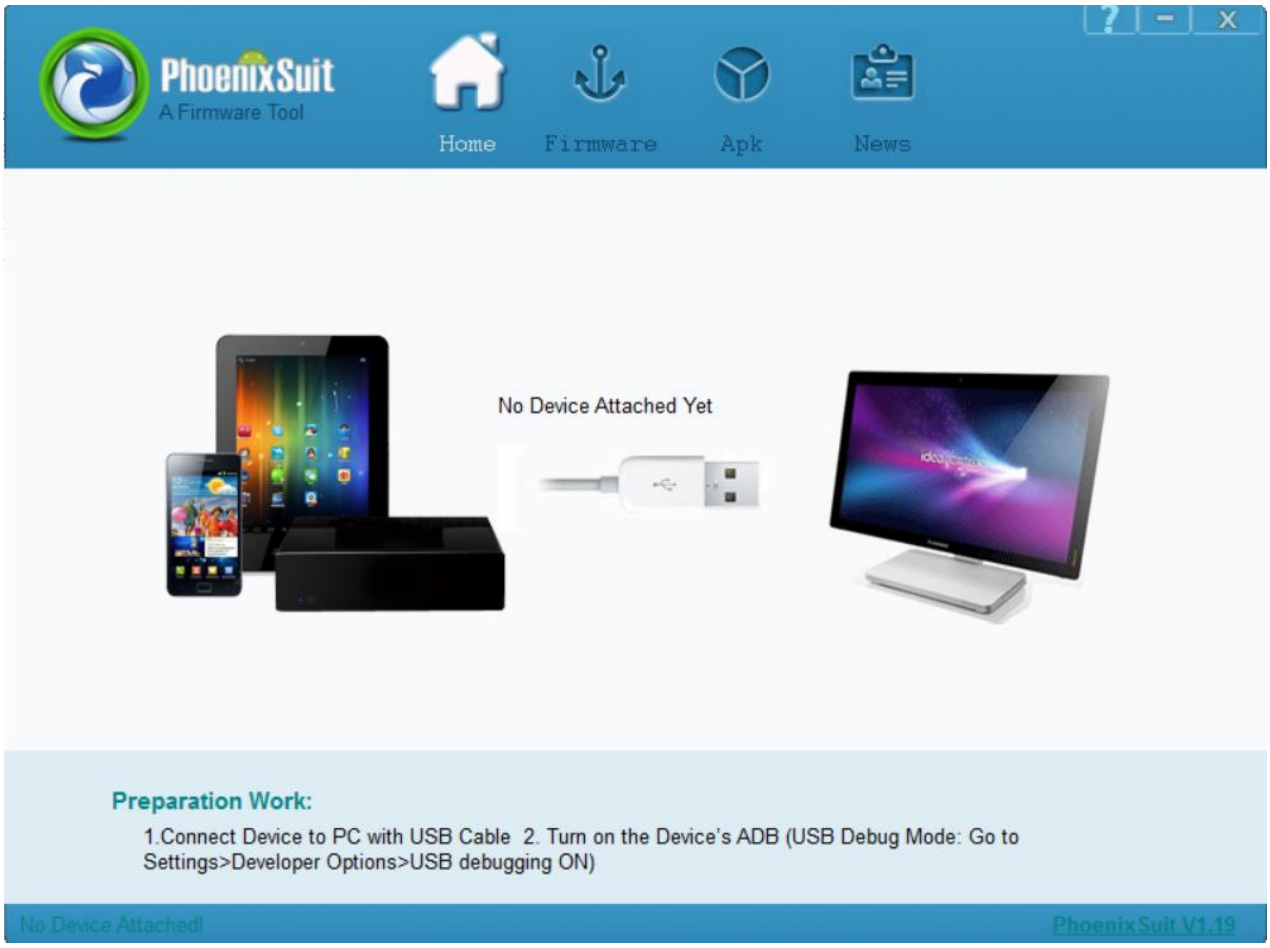

Figure 5-2. PhoenixSuit Program

In the following screen, click "One Click" and then click "Browse" to select the firmware image file

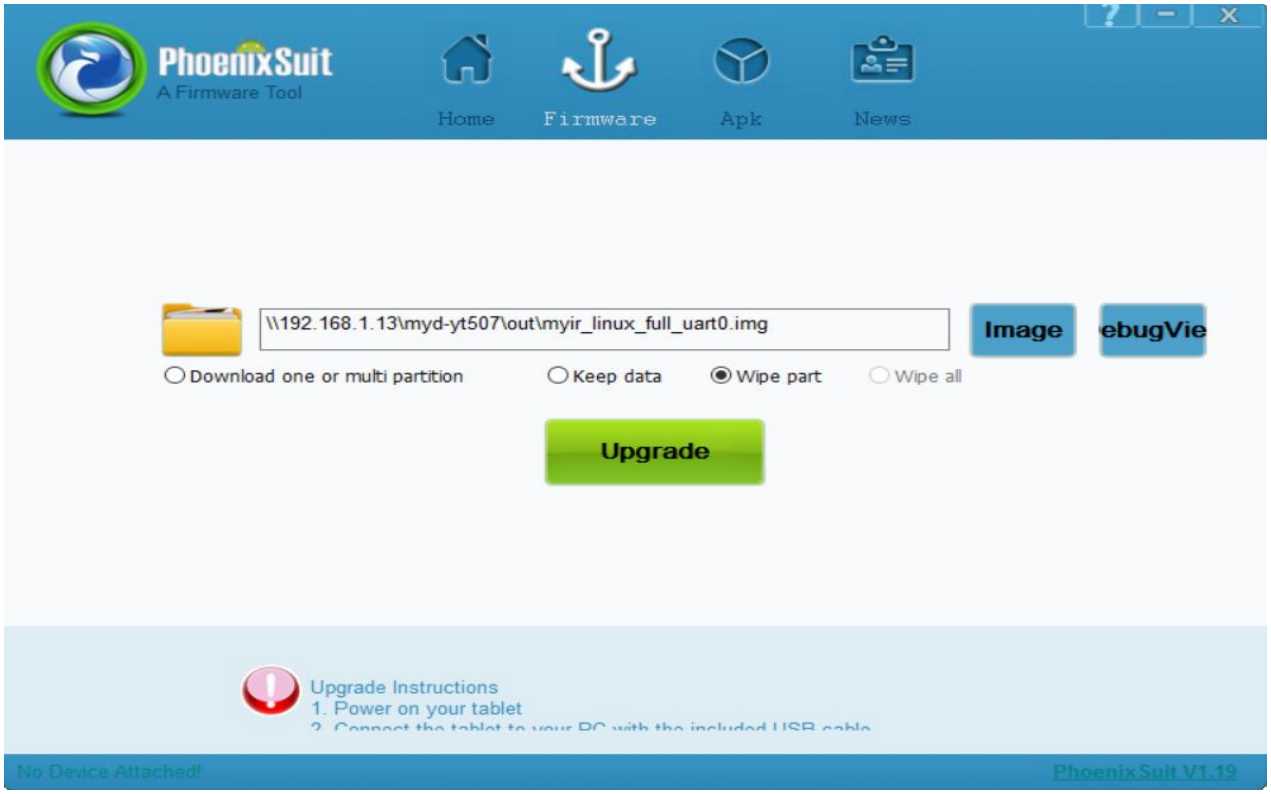

Figure 5-3. Burning Image

In the following screen, click "Yes" to enter the formatting upgrade mode

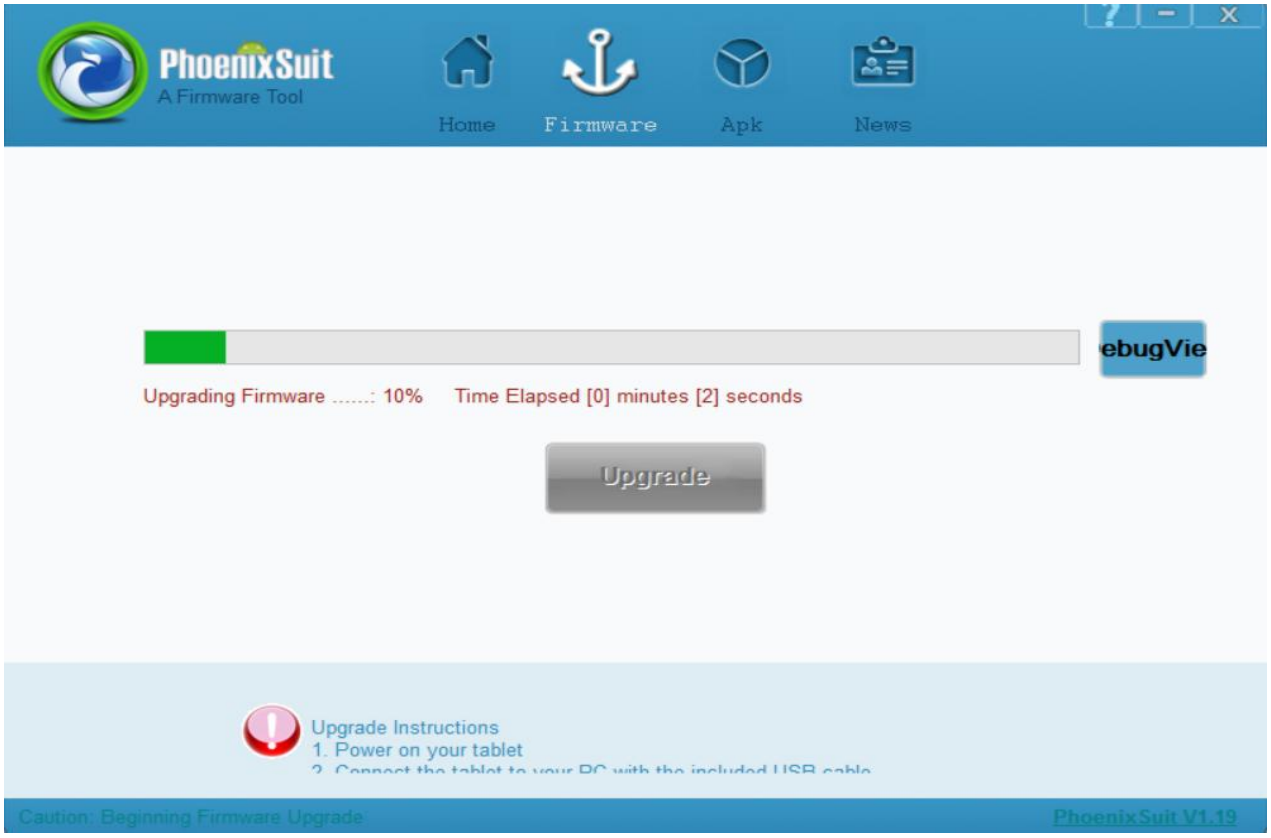

Figure 5-4. Upgrading Firmware

Wait for the burn-in to complete, the following interface pops up when the burn-in is complete

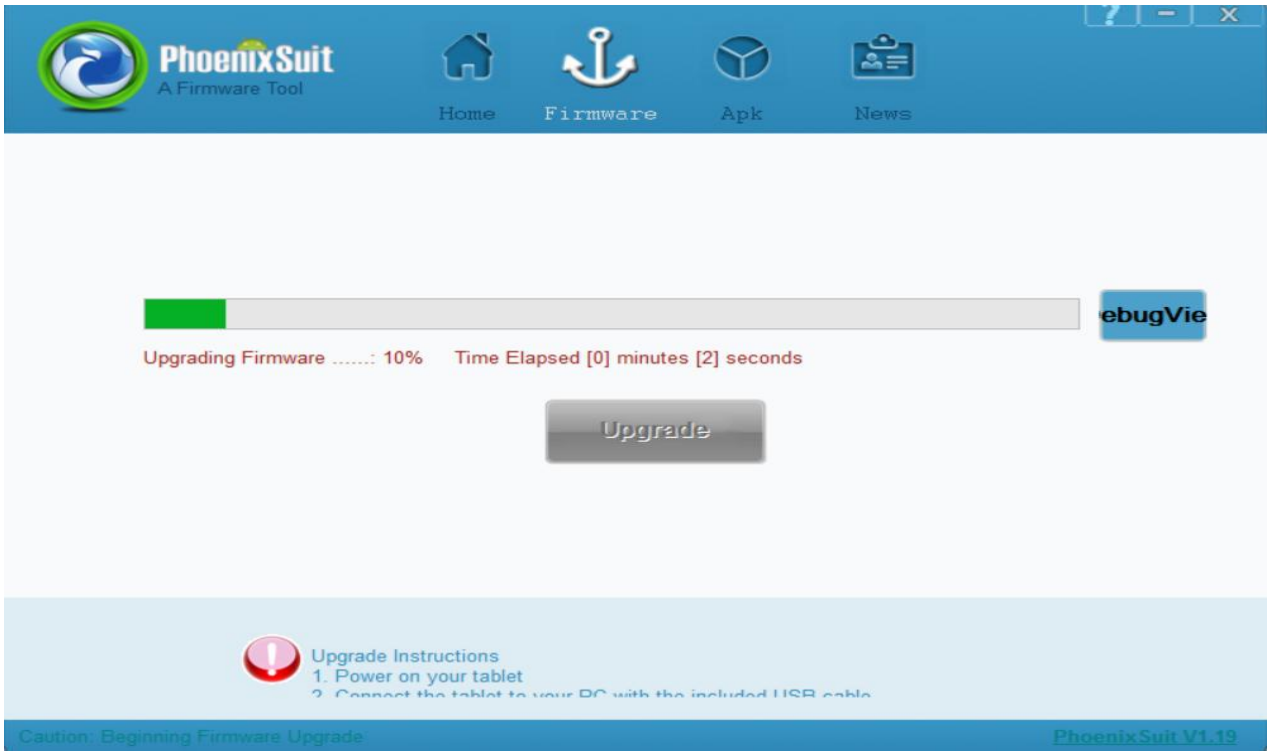

Figure 5-5. Burn-in completed

Use the OTG USB cable to connect the development board and the host computer, first press and hold the FEL key and then press the power on button to reset the system, release the FEL key after about two seconds and wait for the burn-in to complete. In the same way, other configuration items such as BOOT and ROOTFS can be checked to brush separately.

5.2. Make TF Card Starter

The following steps are all made under Windows system.

1). Preparation

- SD card (no less than 8 GB)
- MYD-YT507H Development Board
- Create image tool PhoenixCard (path: \03-Tools\myir tools)

Table 5-1. Image Package List

Image Name	Package Name	Applicable Core Boards
myir-image-android	t507_android10_evb_uart0.img	MYC-YT507H

2). Make SD Card Boot

Copy PhoenixCard.zip from the user profile tools directory to any directory in windows, double click PhoenixCard.exe file in PhoenixCard directory. Insert the 8GB SD card into the windows USB port via SD card reader, as shown below, select "Firmware" path; select "Boot Card" and click "Burn Card" button. Click the "Burn Card" button to finish automatically.

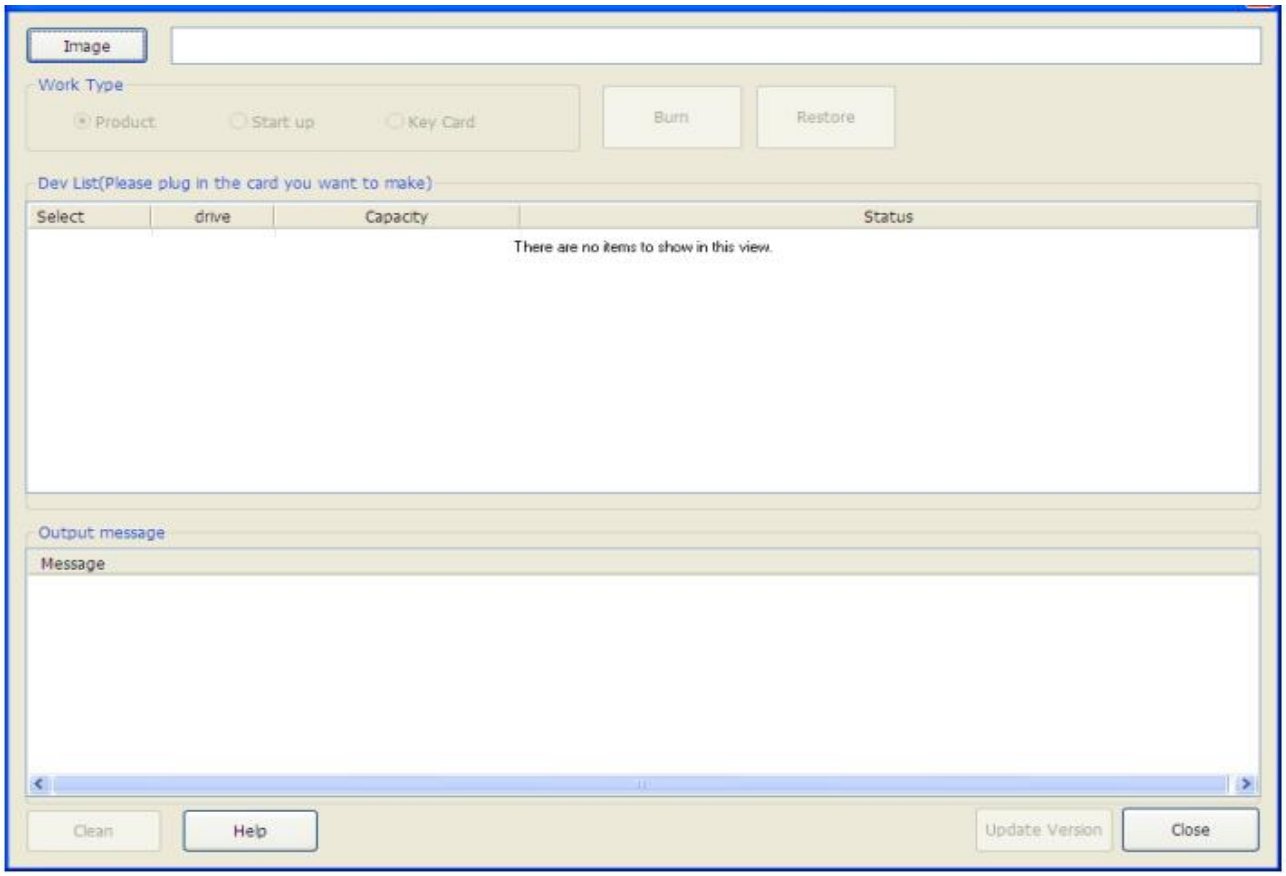

Figure 5-6. Burning Program

The process of burning the card is underway and is expected to take 3-5 minutes to complete (depending on the size of the package).

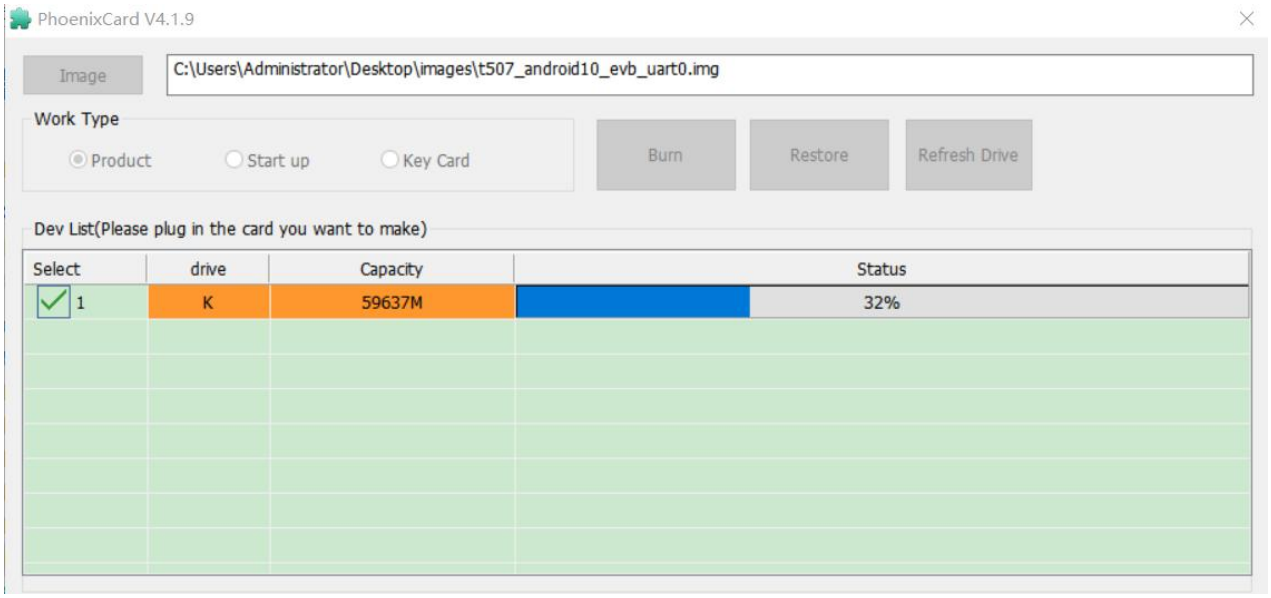

Figure 5-7. Burning Image

Finish burning the card, and also note the output message indicating that the burning is complete.

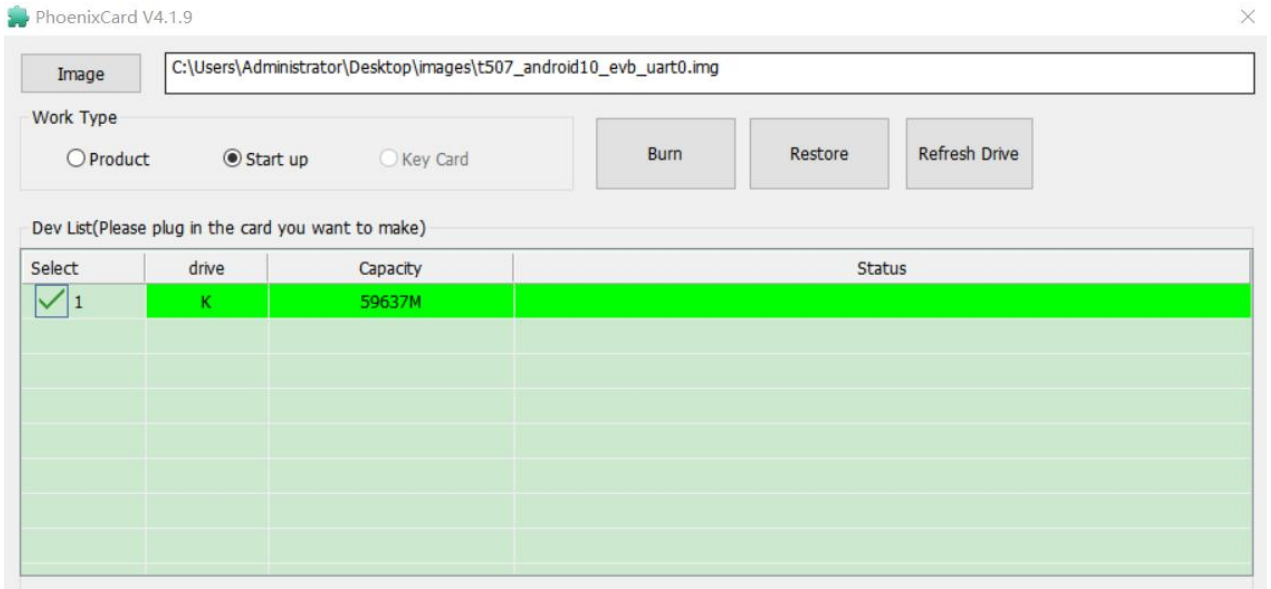

Figure 5-8. Burning Completed

When the writing is finished, you can use this micro SD for booting, insert the card slot (J8) on the back of the development board and set the dip switch to (S0/S1/S2/S3: 0 1 1 1) and power up again to boot the system with the micro SD card.

Note: When using a micro sd card as a boot card to boot myir-image-android, you need to modify the boot parameters debug interface in uboot to interrupt

the uboot countdown. Modify the file system mount location under the uboot terminal. Enter the following command:

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

/dev/mmcblk1p4 represents the fourth partition of the micro sd card.

5.3. Make TF Card Burner

In order to meet the needs of production burn-in, this method is suitable for mass production burn-in method. The system to be burned is written to the onboard eMMC through the system in the SD card. Please follow the steps below to complete the process.

- **Create SD Card Bootloader**

The purpose of making an SD card bootloader is to use SD as a medium to flush eMMC.

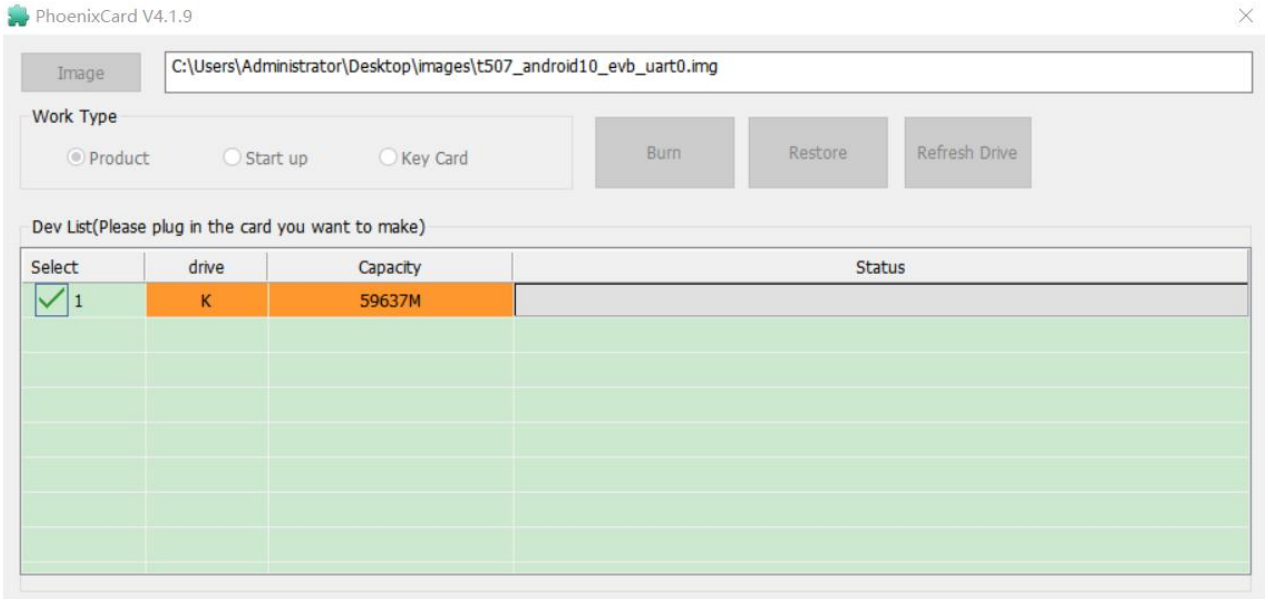

Figure 5-9. Mass Production Card Creation

Choose "mass production card" to create, the production method is the same as chapter 5.2, here will not be repeated.

- **SD Card Burning eMMC**

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

debug serial port to check the update status, it will finish after about 3-5 minutes (depending on the package size).

- **Verify eMMC boot**

After burning the system image to eMMC via Micro SD Card, you need to turn off the computer and pull out the SD card to switch the boot mode to eMMC (S0/S1/S2/S3: 1 0 1 1) for booting.

6. Porting to Fit Your Hardware Platform

In order to adapt to your new hardware platform, you need to know what resources MYIR's MYD-YT507H development board provides. For details, see MYD-YT507H Android SDK 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 to 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/evb/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 scan
;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 Android_10_sys_config.fex Usage Configuration Description files.

6.2. How to 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-yt507.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 configuratio
	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:

longan/device/config/chips/t507/configs/evb/sys_config.fex

longan/device/config/chips/t507/configs/evb/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: longan/kernel/linux-4.9/arch/arm64/boot/dts/sunxi

We write the resources related to MYC-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: longan/device/config/chips/t507/configs/evb/board.dts

The board.dts configuration is as follows:

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

#include "myir-yt507.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 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 Pins 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 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.

6.3.1.1. Understand the Pin Alternate Function Through the Manual

1). Determine the Pins of the Core Board

All the available GPIOs of the core board can be found through the pin list file MYC-YT507H Pin List V1.0 organized by MYIR. At the same time, you can determine the default conditions used by the current SDK.

Here we will take an actual GPIO pin as an example: PE19

MYC-YT507H Pin List V1.0						
MYC Module Pin	MYC Module Pin Name	T507-H Pin	T507-H Pin Name	Default function	Voltage	Output or Input
52	NCSI0-D4	D3	PE8 / NCSI0-D4 / PE-EINT9	/ SI0-D4	3.3V	Input
53	NCSI0-D0	E2	PE4 / NCSI0-D0 / PE-EINT5	/ SI0-D0	3.3V	Input
54	NCSI0-D1	E3	PE5 / NCSI0-D1 / PE-EINT6	/ SI0-D1	3.3V	Input
55	NCSI0-VSYNC	D5	PE3 / NCSI0-VSY / NCPE-EINT4	/ SI0-VSY /	3.3V	Input
56	NCSI0-D15	D4	PE19 / NCSI0-D15 / PE-EINT20	/ SI0-D15	3.3V	Input
57	CSI-FSIN0	F6	PE22 / CSI-FSIN0 / TCON-TRIG0 / PE-EINT23	CSI-FSIN0	3.3V	Input
58	NCSI0-D14	E4	PE18 / NCSI0-D14 / PE-EINT19	/ SI0-D14	3.3V	Input

Figure 6-1. Gpio Pin List

2). View the Alternate Function Relationship of GPIOs

Check the hardware manual or T5 pin V0.9 file to obtain the PE19 pin Alternate Function relationship.

Table 6-2. PE19 Pin Alternate Function list(Reference)

Function	PinName	IOType	IO State	up/down	Multi2(DDR4)	Multi3(DDR3)	Multi4(LPDDR3)	Multi5(LPDDR4)	Multi 6	PIN power
PE	PE19	I/O	DIS		NCSI-D15				PE-EINT20	VCC-PE

6.3.1.2. GPIO is Referenced in Device Tree

1). Configure Function Pins as GPIO Function Instances

You can design a GPIO control node in the device tree to access and control the node with specific drivers. The following chapters will specifically implement how the driver refers to the device tree node.

The configuration method is to add nodes in the device tree.

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

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

2). Configure Pins for Other Specific Functions

The MYD-YT507H development board defines and implements many rich functions, but it also occupies a large number of pin resources. Here, the LVDS display control pin (LVDF0-V0P) is taken as an example to illustrate how to use the GPIO Alternate Function function. As above, learn about the Alternate Function relationship of pins first.

Table 6-3. LCD-D0 Pin Alternate Function List

Function	pinName	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

It can be seen from the above table that the Multi3 Alternate Function relationship of PD0 is used as the data signal of LVDS.

The following pin configurations can be performed:

```
//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", "PD5",
 "PD8", "PD9", "PD6", "PD7";
                        allwinner,function = "lvds0";
                        allwinner,muxsel = <3>; //3 stands for Multi3
                        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", "PD5",
 "PD8", "PD9", "PD6", "PD7";
```

```
                allwinner,function = "lvds0_suspend";
                allwinner,muxsel = <7>;
                allwinner,drive = <3>;
                allwinner,pull = <0>;
        };
```

6.4. How to Use Configured Pins

The pins configured in the device tree of Kernel can be used in Kernel to control the pins.

6.4.1. How to Use GPIO Pins in Kernel Driver

6.4.1.1. Use of Standalone IO Drivers

In the first device tree example in Section 6.3.1, the gpio node information has been defined. Next, the kernel driver will be used to implement GPIO control (set PE19 pin to 1 and 0, and use a multimeter to test the change of pin level if necessary).

```
//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 master device number  */
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 file_o
perations structure */
static ssize_t gpio_drv_read (struct file *file, char __user *buf, size_t size, loff_t *offs
et)
{
        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,
};

/* Get GPIO from platform_device
 * Tell the kernel about the file_operations structure: register the driver
 */
static int chip_demo_gpio_probe(struct platform_device *pdev)
{
        /*  Defined in the device tree: 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);
        }
```

```
        /*  Register file_operations */
        major = register_chrdev(0, "myir_gpioctr", &gpioctr_drv);  /* /dev/gpioctr */

        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_gpioctr%
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;
}

/* If you have an entry function, you should have an exit function: when you unins
      tall the driver, you will call this exit function */
// Uninstall platform_driver
static void __exit gpio_exit(void)
{
    platform_driver_unregister(&chip_demo_gpio_driver);
}

/* Other refinements: provide device information and automatically create device
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.

6.4.1.2. Example Drivers 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 "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_longan_defconfig":

```
//linux-4.9/arch/arm64/configs/sun50iw9p1_longan_defconfig
CONFIG_SAMPLES=y
CONFIG_SAMPLE_GPIO=y
CONFIG_SAMPLE_RPMSG_CLIENT=m
```

Follow section 4.4.1.3 to compile and update the kernel.

6.4.1.3. The Driver Sample Compiled as a Separate Module

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

```
# Change KERN_DIR
#KERN_DIR = # Directory of development board kernel source code
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 this:
# ab-y := a.o b.o
# obj-m += ab.o
```

Load SDK environment variables to 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
make -C /home/lcy/work/t507/kernel/linux-4.9/ M=`pwd` modules
make[1]: Entering 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]: Leaving directory '/home/lcy/work/t507/kernel/linux-4.9/'
```

After successful compilation, the “gpioctr.ko” file can be transferred to the directory of the development board via Ethernet, WIFI, USB otg, USB disk, etc. to load the driver using the “insmod” command.

6.4.2. Configuration of HAL Layer

Continuing the GPIO control in the previous section, we will introduce how gpio is called and controlled by the HAL layer. Let's take GPIO (PE19) control LED as an example. PE19 is designed on the bottom plate of MYD-YT507H to control a blue LED, as shown in the following figure. When GPIO (PE19) is configured as 0, the lamp will be on. Otherwise, it will be extinguished.

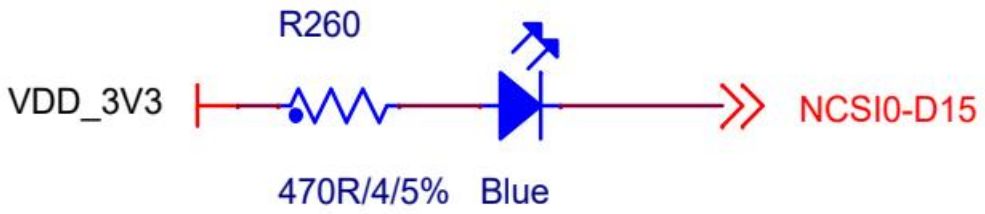

Figure 6-2. Gpio Control led

6.4.2.1. Controllable GPIO in the System

After adding the GPIO configuration, nodes will be generated under the sys file system.

```
# /sys/class/gpio_sw # ls
PE19 normal_led
# /sys/class/gpio_sw # echo 1 > normal_led
```

Writing 0 to the normal_led node in the directory will cause the output to be low, and writing 1 will cause the output to be high. To facilitate the operation in the code, there are java and C++interfaces that provide FRAMEWORK

6.4.2.2. Interface of Java Layer

The interface for java to control GPIO is defined in the file gpio.java.The path is:*/platform/framework* “setNormalLedOn(bool)” “setStandbyLedOn(bool)”,These interface files facilitate the control of Led opening and closing.The interfaces provided are as follows:

```
public static int setNormalLedOn(boolean on);
```

```
public static int setStandbyLedOn(boolean on);
public static int setNetworkLedOn(boolean on) ;
public static int writeGpio(char group, int num, int value);
public static int readGpio(char group, int num) ;
public static int setPull(char group, int num, int value);
public static int getPull(char group, int num);
public static int setDrvLevel(char group, int num, int value);
public static int getDrvLevel(char group, int num);
public static int setMulSel(char group, int num, int value);
public static int getMulSel(char group, int num);
```

6.4.2.3. Interface of C++ Layer

The operation functions of the C++layer are simple encapsulation of the kernel interface. The specific interfaces are as follows:

```
int readData(const char * filePath);
int writeData(const char *data, int count, const char *filePath);
cfg：  Setting/reading functions of gpio
0x00：input
0x01：output
pull：Set/read gpio resistance pull-up or pull-down
0x00：Close pull-up/pull-down
0x01：pull-up
0x02：pull-down
0x03：retain
drv：Set/read the drive level of gpio
0x00：level 00x01：level 10x02：level 20x03：level 3
data：Set/read the level status of gpio
0x00：Low level
0x01：High level
```

In C++language, you can use read and write functions to directly operate these four files.For specific examples, refer to the file */vendor/aw/homelet/framework/gpio/libgpio/GPIOService*

7. How to Add Your Application

The porting of Android applications is usually divided into two phases, the development and debugging phase and the production deployment phase. The development debugging phase allows us to cross-compile our written application using the MYIR build SDK and then remotely copy it to the target host for testing. The production deployment phase involves writing recipe files for the application and using the build production image.

7.1. Pre Installed APK

There are two ways to install the preinstalled apk, either to the */system/app* directory or to the */system/preinstall* directory.

Note: The apk name cannot contain Chinese, spaces or other special characters.

Due to the copyright issue, it is recommended not to install the GAPP application. If you pass the GMS certification you need to install the genuine GAPP application provided by Google.

7.1.1. PreInstall to the system/app Directory

Create a directory in */vendor/aw/public/prebuild/apk/* to store the corresponding APK. Create the Android.mk file in that directory and edit it to:

```
# Example
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := APK_MODULE_NAME  (Unique name of the module)
LOCAL_MODULE_CLASS := APPS
LOCAL_MODULE_TAGS := optional
LOCAL_BUILT_MODULE_STEM := package.apk
LOCAL_MODULE_SUFFIX := $(COMMON_ANDROID_PACKAGE_SUFFIX)
LOCAL_CERTIFICATE := PRESIGNED  (Signature Method)
#LOCAL_OVERRIDES_PACKAGES := OVERRIDES_MODULE  (Modules to be replace
d)  LOCAL_SRC_FILES := name.apk  (The file name of the apk, usually the same na
me as MODULE)
include $(BUILD_PREBUILT)
```

In the solution mk file (device/{vendor-name}/{device-name}/{product-name}.mk)

PRODUCT_PACKAGES item, add:

```
PRODUCT_PACKAGES += APK_MODULE_NAME  (apk module name, pre-installed multiple apk separated by spaces)
```

Pre install to the */system/preinstall* directory

Pre-install in the */system/app* directory and complete all the above steps. Modify Android.mk in the apk directory and add the line:

```
LOCAL_MODULE_PATH := $(TARGET_OUT)/preinstall
```

7.1.2. Preinstall to the System/preinstall Directory

Complete all the above steps with the pre installation to the */system/app* directory. Modify Android.mk in the apk directory and add a line

```
LOCAL_MODULE_PATH := $(TARGET_OUT)/preinstall
```

8. Resources

- **Linux kernel open source community**
  https://www.kernel.org/
- **Android Developer Center**
  https://developer.android.com/

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