

# MYD-LT527 Android System Development Guide



File Status: [ ] Craft [ ✓ ] Release	<b>FILE ID:</b>	MYIR-MYD-LT527-SW-DG-EN-A5.15
	<b>VERSION:</b>	V1.0[doc]
	<b>AUTHOR:</b>	Nico
	<b>CREATED:</b>	2024-01-18
	<b>UPDATED:</b>	2024-02-02

# Revision History

VERSION	AUTHOR	PARTICIPANT	DATE	DESCRIPTION
V1.0	nico		20240202	Initial Version



# CONTENT

Revision History .....	- 2 -
CONTENT .....	- 3 -
1. Overview .....	- 5 -
1.1. Software Resources .....	- 5 -
1.2. Document Resources .....	- 5 -
2. Development Environment .....	- 7 -
2.1. Developing Host Environment .....	- 7 -
2.2. Software Environment .....	- 8 -
2.2.1. Information Acquisition .....	- 8 -
2.2.2. Install cross compilation tool chain .....	- 8 -
3. Android SDK Structure .....	- 11 -
3.1. synopsis .....	- 11 -
3.2. Get source code .....	- 11 -
3.2.1. Getting the source tarball from a CD-ROM image .....	- 11 -
3.3. Recognizing the SDK structure .....	- 13 -
3.3.1. Android SDK main structure .....	- 13 -
3.3.2. "longan" Catalog Structure .....	- 14 -
3.3.3. "bsp" catalog structure .....	- 15 -
4. Build android image .....	- 17 -
4.1. Compile the kernel .....	- 17 -
4.2. Compile android .....	- 20 -
5. How to burn system images .....	- 27 -
5.1. PhoenixSuit Burning .....	- 27 -
5.2. PhoenixCard Burning .....	- 31 -
6. How to Modify Board Support PackageS .....	- 36 -



6.1. Board Support Package Introduction .....	- 36 -
6.2. Onboard uboot compilation and update .....	- 38 -
6.2.1. Compile uboot in SDK separately .....	- 38 -
6.2.2. Update uboot .....	- 38 -
6.3. Onboard kernel compilation and update .....	- 39 -
6.3.1. Compile the kernel separately in the SDK .....	- 39 -
6.3.2. Update kernel .....	- 40 -
6.4. Onboard Android Compilation and Updates .....	- 40 -
6.4.1. Compile Android in SDK separately .....	- 41 -
7. How to adapt your hardware platform .....	- 42 -
7.1. How to create your device tree .....	- 42 -
7.1.1. Onboard Device Tree .....	- 42 -
7.1.2. Adding a Device Tree .....	- 43 -
7.2. How to Configure CPU Function Pins According to Your Hardware... - 44 -	
7.2.1. Methods of GPIO Configuration .....	- 44 -
7.3. How to use your own configured pins .....	- 47 -
7.3.1. Using GPIO pins in kernel drivers .....	- 47 -
7.3.2. Configuration of the HAL layer .....	- 53 -
8. How to add your app .....	- 63 -
8.1. Pre-installed non-uninstallable APK .....	- 63 -
8.2. Preinstalled uninstallable apk .....	- 63 -
9. References .....	- 65 -
Appendix A .....	- 66 -
Warranty & Technical Support Services .....	- 66 -



# 1. Overview

This article mainly introduces the complete process of customizing a complete embedded Android system based on Allwinner T5 series SDK project and MYIR core board, which includes the preparation of the development environment, the acquisition of the code, and how to do Bootloader, Kernel porting, customizing the Android system suitable for their own application needs. We firstly introduce how to build the system image for MYD-LT527 development board based on the source code provided by us, and how to burn the built image to the development board. For those users who carry out project development based on MYC-LT527X core board, we focus on the methods and some key points of porting this set of systems to the user's hardware platform, and through some actual BSP porting cases and Android customization cases, so that the user can quickly customize the system image suitable for their own hardware.

This document does not contain the introduction of Android system related basic knowledge, suitable for embedded Android system developers and embedded Android BSP developers who have some development experience. For some specific functions that users may use in the process of secondary development, we also provide detailed application notes for developers' reference, see the document list in Table 2-4 of "MYD-LT527 Android SDK Release Notes" for specific information.

## 1.1. Software Resources

MYD-LT527 is equipped with Android 13 system, which provides rich system resources and other software resources. Including documentation, code, and various development and debugging tools for Windows desktop environment and PC Linux system, application development routines and so on. Please refer to Chapter 2 Software Information in the MYD-LT527 Android SDK Release Notes for detailed information on the included software.

## 1.2. Document Resources

According to the user to use the development board for various different purposes. Users will be provided with a complete SDK package (Software



Development Kit), which contains different categories of documents and manuals such as Release Notes, Getting Started Guide, Evaluation Guide, Development Guide, Application Notes, Frequently Asked Questions, etc. The specific list of documents can be found in Table 2-4 of the "MYD-LT527 Android SDK Release Notes". For a list of specific documents, please refer to Table 2-4 of "MYD-LT527 Android SDK Release Notes".



## 2. Development Environment

This chapter mainly introduces the development process based on the MYD-LT527 development board in the development of some of the required hardware and software environment, including the necessary development host environment, the necessary software tools, code and data acquisition, etc., specific preparations for the following will be described in detail.

### 2.1. Developing Host Environment

How to build the development environment applicable to 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 related development environment to prepare for the later development and debugging. Allwinner T5 series processors are multi-core processors, equipped with a high-performance eight-core CortexTM-A55.

- **Host Hardware**

The build of the entire Android SDK package project has higher requirements for the development host, requiring a processor with a 4-core CPU or higher, 16GB of RAM or more, and a 500GB hard disk or higher configuration. It can be a PC or server with Linux system installed, or a virtual machine running Linux system, WSL2 under Windows system, etc.

- **Host Operating System**

Build Android project host operating system can have a variety of choices, generally choose to install Fedora, openSUSE, Debian, Ubuntu, RHEL or CentOS and other Linux distributions on the local host to build, here is the recommended Ubuntu20.04 64bit desktop version of the system, the subsequent development is also This system as an example to introduce.

- **Prerequisite Package Installation**

Install the necessary development dependencies on the host side first.

```
PC$: sudo apt-get update
```



```
PC$: sudo apt install -y git gnupg flex bison gperf build-essential zip curl libc6-de
v libncurses5-dev:i386 x11proto-core-dev libx11-dev:i386 libreadline6-dev:i386 lib
gl1-mesa-glx:i386 libgl1-mesa-dev g++-multilib tofrodos python markdown libx
ml2-utils xsltproc zlib1g-dev:i386 gawk texinfo gettext build-essential gcc libncurs
es5-dev bison flex zlib1g-dev gettext libssl-dev autoconf libtool linux-libc-dev:i38
6 wget patch dos2unix tree u-boot-tools libelf-dev libncurses5
```

## 2.2. Software Environment

### 2.2.1. Information Acquisition

Before building the environment to download the development board materials, about the development of detailed information, please refer to the "MYD-LT527 Android SDK Release Notes" development board materials to download the address is as follows (information will be updated from time to time, please download the latest version).

<http://d.myirtech.com/MYD-LT527/>

### 2.2.2. Install cross compilation tool chain

In the process of using the SDK to build this system image, it is also necessary to install the cross-toolchain, MYIR provides this SDK in addition to a variety of source code also provides the necessary cross-toolchain, which can be used directly for compiling applications and so on. Users can directly use the cross-compile toolchain to establish an independent development environment, can compile Bootloader, Kernel, Android or compile their own applications, the specific process will be described in detail in the following sections. The installation of the SDK is described here, as follows:

- **Copy the SDK to the Linux directory**

Copy the SDK package to the user's working directory under Ubuntu, e.g. \$HOME/MYD-LT527, which is defined according to your own situation.

- **View compiled chain file**

According to the user needs to decompress the toolchain to the specified directory, download information in the "03\_Tools/Complie Toolchain" directory stores the compilation chain file "*gcc-arm-10.3-2021.07-x86\_64-aarch64-none-linux-gnu.tar.gz*", copy the file to Ubuntu, and then unpack the file, users can





choose their own unpacking directory, here to create a new "*~/opt*" directory as an example:

```
PC$ mkdir -p ~/opt
```

- **Extracting compilation chain files**

Extract to the host's "*/opt*" directory, or you can choose your own directory if prompted:

```
PC$ tar -xf gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu.tar.gz -C opt/
```

- **Installing and testing the compilation chain**

To set up the compilation chain, you can create a script "LT527X-env.sh" in advance, as follows:

You can choose the directory where you want to place your scripts, here is an example of a new "*~/env*" directory.

```
PC$ mkdir -p ~/env && vi ~/env/LT527X-env.sh
```

Then copy the following script into "LT527X-env.sh", the red fragment below must be the same as the directory of the compiled chain after decompression.

```
#!/bin/sh
export PATH=$PATH:~/opt/gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gn
u/bin
export ARCH=arm64
export CROSS_COMPILE=aarch64-none-linux-gnu-
export PREFIX=aarch64-none-linux-gnu-
export AS=aarch64-none-linux-gnu-as
export LD=aarch64-none-linux-gnu-ld
export CC=aarch64-none-linux-gnu-gcc
export AR=aarch64-none-linux-gnu-ar
export NM=aarch64-none-linux-gnu-nm
export STRIP=aarch64-none-linux-gnu-strip
export OBJCOPY=aarch64-none-linux-gnu-objcopy
export OBJDUMP=aarch64-none-linux-gnu-objdump
```

Setting environment variables and testing if the installation is complete



```
PC$ source ~/env/LT527X-env.sh
PC$ aarch64-none-linux-gnu-gcc -v
Using built-in specs.
COLLECT_GCC=aarch64-none-linux-gnu-gcc
COLLECT_LTO_WRAPPER=/home/myir/opt/gcc-arm-10.3-2021.07-x86_64-aarch64-
none-linux-gnu/bin/./libexec/gcc/aarch64-none-linux-gnu/10.3.1/lto-wrapper
Target: aarch64-none-linux-gnu
Configured with: /data/jenkins/workspace/GNU-toolchain/arm-10/src/gcc/config
ure --target=aarch64-none-linux-gnu --prefix= --with-sysroot=/aarch64-none-lin
ux-gnu/libc --with-build-sysroot=/data/jenkins/workspace/GNU-toolchain/arm-1
0/build-aarch64-none-linux-gnu/install//aarch64-none-linux-gnu/libc --with-bug
url=https://bugs.linaro.org/ --enable-gnu-indirect-function --enable-shared --disa
ble-libssp --disable-libmudflap --enable-checking=release --enable-languages=c,
c++,fortran --with-gmp=/data/jenkins/workspace/GNU-toolchain/arm-10/build-a
arch64-none-linux-gnu/host-tools --with-mpfr=/data/jenkins/workspace/GNU-to
olchain/arm-10/build-aarch64-none-linux-gnu/host-tools --with-mpc=/data/jenki
ns/workspace/GNU-toolchain/arm-10/build-aarch64-none-linux-gnu/host-tools -
-with-isl=/data/jenkins/workspace/GNU-toolchain/arm-10/build-aarch64-none-li
nux-gnu/host-tools --enable-fix-cortex-a53-843419 --with-pkgversion='GNU Tool
chain for the A-profile Architecture 10.3-2021.07 (arm-10.29)'
Thread model: posix
Supported LTO compression algorithms: zlib
gcc version 10.3.1 20210621 (GNU Toolchain for the A-profile Architecture 10.3-2
021.07 (arm-10.29))
```

You can see that the last line prints the same version as the installed version, which proves that the installation was successful.





## 3. Android SDK Structure

### 3.1. synopsis

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

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

- BSP development including bootloader, uboot and kernel.
- android file system development, including mass-produced embedded android systems.

IP validation and distribution platform, and give the use of IP and system integration demo program, convenient for third parties to use quickly.

- Testing, including board level testing and system testing.

The *"04\_Sources"* directory in the CD-ROM image provided by MYIR provides android SDK files and data for the MYD-LT527 development board to help developers build different types of Android system images that can run on the MYD-LT527 development board.

### 3.2. Get source code

Users can get the package directly from the *"04-sources"* directory of the MYIR CD-ROM image.

#### 3.2.1. Getting the source tarball from a CD-ROM image

The compressed source package is located in the MYIR development kit materials *"04-Sources/MYD-LT527-SDK\_V1.0.0.tar.gz"* and *"04-Sources/MYD-LT527-Android13-SDK\_V1.0.0.tar.gz"*. Copy the compressed package to a user-specified directory, such as \$HOME/MYD-LT527, which will serve as the directory for subsequent builds:

Unzip the "MYD-LT527-SDK\_V1.0.0.tar.gz" zip file, the directory contains the uboot source code, the native kernel source code and allwinner bsp module code, the directory structure is as follows:



```
PC$ cd $HOME/MYD-LT527
PC$ $HOME/MYD-LT527$ tar -xf MYD-LT527-SDK_V1.0.0.tar.gz
PC$ $HOME/MYD-LT527/MYD-LT527-SDK_V1.0.0$ tree -L 1
.
├── bsp
├── linux-5.15
└── u-boot-2018

3 directories, 0 files
```

Extract the *"MYD-LT527-Android13-SDK\_V1.0.0.tar.gz"* zip package, its directory structure is as follows:

```
PC$ $HOME/MYD-LT527$ tar -xf MYD-LT527-Android13-SDK_V1.0.0.tar.gz
PC$ $HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0$ tree -L 1
.
├── Android.bp -> build/soong/root.bp
├── art
├── bionic
├── bootable
├── bootstrap.bash -> build/soong/bootstrap.bash
├── build
├── BUILD -> build/bazel/bazel.BUILD
├── cts
├── dalvik
├── developers
├── development
├── device
├── external
├── frameworks
├── hardware
├── kernel
├── libcore
├── libnativehelper
└── longan
```



```

├── out
├── packages
├── pdk
├── platform_testing
├── prebuilts
├── sdk
├── system
├── test
├── toolchain
├── tools
├── vendor
└── WORKSPACE -> build/bazel/bazel.WORKSPACE

```

27 directories, 4 files

### 3.3. Recognizing the SDK structure

#### 3.3.1. Android SDK main structure

The following mainly introduces android SDK related, android source code in the "MYD-LT527-Android13-SDK\_V1.0.0" directory, its structure is as follows:

```

├── Android.bp -> build/soong/root.bp
├── art
├── bionic
├── bootable
├── bootstrap.bash -> build/soong/bootstrap.bash
├── build
├── BUILD -> build/bazel/bazel.BUILD
├── cts
├── dalvik
├── developers
├── development
├── device
├── external
├── frameworks

```



```

├── hardware
├── kernel
├── libcore
├── libnativehelper
├── longan
├── out
├── packages
├── pdk
├── platform_testing
├── prebuilts
├── sdk
├── system
├── test
├── toolchain
├── tools
├── vendor
└── WORKSPACE -> build/bazel/bazel.WORKSPACE

```

The main directories modified by the user are device, longan, and vendor.

- The "device" directory is for device-related configurations.
- The longan directory is the linux 5.15 kernel code.
- The "vendor" directory is used to add source code and libraries from different vendors.

### 3.3.2. "longan" Catalog Structure

The longan directory is the kernel directory, this directory is used to compile the kernel source code, its structure is as follows:

```

PC$ $HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0/longan$ tree -L 1
├── Android.mk
├── brandy
├── bsp
├── build
├── build.sh
└── device

```



```
├── kernel
├── out
├── prebuilt
├── rtos
├── tee_kit
├── test
└── tools
```

11 directories, 2 files

Main Catalog Introduction:

- The "BSP" directory is used to store allwinner BSP module code and related files.
- "device" directory, new defconfig files for board level and soc level.
- The "kernel" directory, which holds only the native kernel code.

### 3.3.3. "bsp" catalog structure

This directory stores the code related to the allwinner bsp module. Allwinner stores its own driver separately from the native kernel driver on kernel version 5.15, and its directory structure is as follows:

```
PC$ $HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0/longan/bsp$
tree -L 1
```

```
├── configs
├── drivers
├── include
├── Kconfig
├── Makefile
├── modules
├── platform
└── ramfs
```

6 directories, 2 files



- The "configs" directory, mainly stores SoC-level driver-related dtsti files and defconfig files.
- The "drivers" directory, mainly stores the driver code corresponding to AW (except gpu, nand and other modules).
- "include" directory, mainly stores .h files exported by modules.
- The "modules" directory stores the drivers for gpu and nand modules.
- The "platform" directory stores files related to the SoC platform.





## 4. Build android image

Building an android image requires two steps, first compiling the kernel in the longan directory and then compiling the android source code in the source top level directory.

### 4.1. Compile the kernel

First go to the "longan" directory and execute the ". /build.sh config" command:

```
PC$ $HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0/longan$  
./build.sh config
```

```
=====ACTION List: mk_config ;=====
```

options :

All available platform:

- 0. android
- 1. linux

Choice [android]: 0

All available ic:

- 0. a523
- 1. a527
- 2. t527

Choice [t527]: 2

All available board:

- 0. demo
- 1. demo\_battery
- 2. demo\_car
- 3. demo\_fastboot
- 4. demo\_linux\_aiot
- 5. demo\_linux\_car

Choice [demo\_car]: 2

All available flash:

- 0. default
- 1. nor



Choice [default]: 0

Setup BSP files

INFO: Prepare toolchain ...

INFO: longanforandroid project, clang\_top\_path: /home/myir/LT527X/android

INFO: kernel defconfig: generate /home/myir/LT527X/android/longan/out/t527/kernel/build/.config by /home/myir/LT527X/android/longan/device/config/chips/t527/configs/demo\_car/linux-5.15/android13\_arm64\_defconfig

INFO: Prepare toolchain ...

make: Entering directory '/home/myir/LT527X/android/longan/kernel/linux-5.15'

make[1]: Access to the catalog "/home/myir/LT527X/android/longan/out/t527/kernel/build"

GEN Makefile

HOSTCC scripts/basic/fixdep

HOSTCC scripts/kconfig/conf.o

HOSTCC scripts/kconfig/confdata.o

HOSTCC scripts/kconfig/expr.o

LEX scripts/kconfig/lexer.lex.c

YACC scripts/kconfig/parser.tab.[ch]

HOSTCC scripts/kconfig/lexer.lex.o

HOSTCC scripts/kconfig/menu.o

HOSTCC scripts/kconfig/parser.tab.o

HOSTCC scripts/kconfig/preprocess.o

HOSTCC scripts/kconfig/symbol.o

HOSTCC scripts/kconfig/util.o

HOSTLD scripts/kconfig/conf

\*\*\* Default configuration is based on '../..../device/config/chips/t527/configs/demo\_car/linux-5.15/android13\_arm64\_defconfig'

#

# configuration written to .config

#

make[1]: Leave the catalog "/home/myir/LT527X/android/longan/out/t527/kernel/build"

make: Leaving directory '/home/myir/LT527X/android/longan/kernel/linux-5.15'



```
INFO: clean buildserver
INFO: prepare_buildserver
```

Then execute the command ". /build.sh" command, as follows:

```
PC$ $HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0/longan$
./build.sh
=====ACTION List: build_linuxdev;=====
options :
INFO: -----
INFO: build linuxdev ...
INFO: chip: sun55iw3p1
INFO: platform: android
INFO: kernel: linux-5.15
INFO: board: demo_car
INFO: output: /home/myir/LT527X/android/longan/out/t527/demo_car/android
INFO: -----
INFO: build dtbo ...
debug: main,line:309
debug: main,line:311
debug: main,line:309
debug: main,line:311
debug: main,line:309
debug: main,line:311
INFO: build_dtbo: make dtboimg start.

===== (Part of the compilation process is omitted) =====

INFO: Use dtsti: /home/myir/LT527X/android/longan/bsp/configs/linux-5.15/sun55
iw3p1.dtsi
/home/myir/LT527X/android/longan/device/config/chips/t527/configs/demo_car/
board.dts:875.10-881.4: Warning (spi_bus_reg): /soc@3000000/spi@4026000/spid
ev1: SPI bus unit address format error, expected "0"
```



```
/home/myir/LT527X/android/longan/device/config/chips/t527/configs/demo_car/
board.dts:894.10-900.4: Warning (spi_bus_reg): /soc@3000000/spi@4027000/spid
ev2: SPI bus unit address format error, expected "0"
/home/myir/LT527X/android/longan/bsp/configs/linux-5.15/sun55iw3p1.dtsi:30.1
0-70.4: Warning (alias_paths): /aliases: aliases property name must include only lo
wercase and '-'
also defined at /home/myir/LT527X/android/longan/device/config/chips/t527/c
onfigs/demo_car/board.dts:14.10-28.4
Use init ramdisk file: '/home/myir/LT527X/android/longan/kernel/linux-5.15/bsp/r
amfs/rootfs_arm64.cpio.gz'.
```

**sun55iw3p1 compile all(Kernel+modules+boot.img) successful**

```
INFO: build dts ...
INFO: Prepare toolchain ...
Setup BSP files
'/home/myir/LT527X/android/longan/out/t527/kernel/staging/sunxi.dtb' -> '/ho
me/myir/LT527X/android/longan/out/t527/demo_car/android/sunxi.dtb'
INFO: build rootfs ...
INFO: skip make rootfs for android
INFO: -----
INFO: build Tina OK.
INFO: -----
```

This completes the kernel build.

## 4.2. Compile android

First go to the top-level directory of the source code and execute the following command:

```
$HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0$ source build/envsetup.s
h
including device/softwinner/common/vendorsetup.sh
```



```
=====
=====
Android13 vendorsetup help:
```

```
build          -- build & pack
build_grf_target_files -- build grf target file from sdk
ca             -- goto android top dir
cbb           -- goto longan/device/config/chips/{ic}/bin dir
cbd           -- goto longan/device/config/chips/{ic}/configs/{board} dir
cb            -- goto longan/build dir
cbr           -- goto brandy dir
cbasp         -- goto langan/bsp dir
cdevice       -- goto device dir
ck            -- goto kernel dir
cl            -- goto longan dir
clone         -- create new device
cok           -- goto longan/out/kernel/build dir
com           -- goto common dir
cout          -- goto $OUT dir
cs            -- goto bootloader(spl) dir
ct            -- goto longan/tools dir
cu            -- goto uboot dir
extract-bsp    -- nothing to do
make           -- allwinner make wrapper to support android grf build
pack4dist      -- pack for dist
pack          -- pack image
sdkupdate      -- reset all local changes & update to latest version
update_uboot   -- update uboot into target-files.zip
vendor_help    -- vendorsetup.sh help
=====
=====
```

Execute the "lunch" command.

```
$HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0$ lunch
```



You're building on Linux

Lunch menu .. Here are the common combinations:

1. a523\_af86310vat\_arm64-user
2. a523\_af86310vat\_arm64-userdebug
3. a523\_aiot\_demo\_arm64-user
- .
- .
- .
80. t527\_demo\_car\_arm64-user
81. t527\_demo\_car\_arm64-userdebug
82. uml-userdebug

Which would you like? [aosp\_arm-eng]

Pick from common choices above (e.g. 13) or specify your own (e.g. aosp\_barbet-eng):**81**

Hint: next time you can simply run 'lunch t527\_demo\_car\_arm64-userdebug'

```
=====
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=13
TARGET_PRODUCT=t527_demo_car_arm64
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=generic
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-a
TARGET_2ND_CPU_VARIANT=generic
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
```



```
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-91-generic-x86_64-Ubuntu-20.04.6-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=TQ2A.230405.003.B2
OUT_DIR=out
=====
```

Finally, execute the "make" command and wait for the compilation to finish.

```
$HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0$ make
```

```
device/softwinner/saturn/BoardConfig.mk:149: warning: longan/out/t527/demo_c
ar/android/dist/sunxi_pd_test.ko not found, skip
device/softwinner/saturn/BoardConfig.mk:149: warning: longan/out/t527/demo_c
ar/android/dist/pwm-sunxi.ko not found, skip
device/softwinner/saturn/BoardConfig.mk:149: warning: longan/out/t527/demo_c
ar/android/dist/sunxi-dma.ko not found, skip
device/softwinner/saturn/BoardConfig.mk:149: warning: longan/out/t527/demo_c
ar/android/dist/sunxi-iommu.ko not found, skip
device/softwinner/saturn/BoardConfig.mk:149: warning: longan/out/t527/demo_c
ar/android/dist/fb.ko not found, skip
device/softwinner/saturn/BoardConfig.mk:149: warning: longan/out/t527/demo_c
ar/android/dist/disp.ko not found, skip
device/softwinner/saturn/BoardConfig.mk:149: warning: longan/out/t527/demo_c
ar/android/dist/edp2.ko not found, skip
```

```
===== (Part of the compilation process is omitted) =====
```

```
2023-12-18 18:29:12 - build_super_image.py - INFO : Building super image from
info dict...
```



```
2023-12-18 18:29:12 - common.py - INFO : Running: "/home/myir/LT527X/android/out/host/linux-x86/bin/lpmake --metadata-size 65536 --super-name super --metadata-slots 3 --virtual-ab --device super:3758096384 --group sb_a:3749707776 --group sb_b:3749707776 --partition system_a:readonly:987189248:sb_a --image system_a=out/target/product/t527-demo-car/system.img --partition system_b:readonly:0:sb_b --partition vendor_a:readonly:130641920:sb_a --image vendor_a=out/target/product/t527-demo-car/vendor.img --partition vendor_b:readonly:0:sb_b --partition product_a:readonly:170541056:sb_a --image product_a=out/target/product/t527-demo-car/product.img --partition product_b:readonly:0:sb_b --partition vendor_dkkm_a:readonly:8298496:sb_a --image vendor_dkkm_a=out/target/product/t527-demo-car/vendor_dkkm.img --partition vendor_dkkm_b:readonly:0:sb_b --partition system_dkkm_a:readonly:348160:sb_a --image system_dkkm_a=out/target/product/t527-demo-car/system_dkkm.img --partition system_dkkm_b:readonly:0:sb_b --sparse --output out/target/product/t527-demo-car/super.img"
2023-12-18 18:29:30 - build_super_image.py - INFO : Done writing image out/target/product/t527-demo-car/super.img
[100% 485/485] Target vendor_boot debug image: out/target/product/t527-demo-car/vendor_boot-debug.img
```

#### build completed successfully (09:28 (mm:ss)) ####

#### build completed successfully (09:28 (mm:ss)) ####

The android compilation is now complete

Finally, execute the "pack" command to generate a burnable image file.

```
$HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0$ pack
INFO: /home/myir/LT527X/android/vendor/security/toc_keys
INFO: No kernel param, parse it from .buildconfig
copying tools file
copying configs file
copying product configs file
/home/myir/LT527X/android/longan/out/t527/demo_car/pack_out/aaults32.fex
```





```
/home/myir/LT527X/android/longan/out/t527/demo_car/pack_out/aultools.fex
/home/myir/LT527X/android/longan/out/t527/demo_car/pack_out/boot_package.
cfg
/home/myir/LT527X/android/longan/out/t527/demo_car/pack_out/boot_package.
fex
/home/myir/LT527X/android/longan/out/t527/demo_car/pack_out/boot_package
_nor.cfg
```

===== (Some of the packing process is omitted) =====

```
FilePath: misc.fex
FileLength=1000000Add partion vbmeta.fex VBMETA_FEX000000
Add partion very vbmeta.fex VBMETA_FEX000000
FilePath: vbmeta.fex
FileLength=2000Add partion vbmeta_system.fex VBMETA_SYSTEM_FEX
Add partion very vbmeta_system.fex VBMETA_SYSTEM_FEX
FilePath: vbmeta_system.fex
FileLength=1000Add partion vbmeta_vendor.fex VBMETA_VENDOR_FEX
Add partion very vbmeta_vendor.fex VBMETA_VENDOR_FEX
FilePath: vbmeta_vendor.fex
FileLength=1000Add partion dtbo.fex DTBO_FEX000000000
Add partion very dtbo.fex DTBO_FEX000000000
FilePath: dtbo.fex
FileLength=200000Add partion Reserve0.fex RESERVE0_FEX0000
Add partion very Reserve0.fex RESERVE0_FEX0000
FilePath: Reserve0.fex
FileLength=1000000BuildImg 0
Dragon execute image.cfg SUCCESS !
-----image is at-----

1.6G /home/myir/LT527X/android/longan/out/t527_android13_demo_car_uart0.i
mg
```



At this point the android image build is complete, and the resulting image files will be stored in the out directory under the longan directory, as follows:

```
$HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0/longan/out$ ls  
kernel pack_out serversocket t527 t527_android13_demo_car_uart0.img toolchain
```



## 5. How to burn system images

The MYC-LT527X series of core boards and development boards designed by MYIR are based on Allwinner's T5 series of microprocessors, which have a variety of startup methods, so they require different tools and methods for updating the system. Users can choose different ways to update according to their needs. There are mainly the following update methods:

- USB burning: suitable for R & D debugging, testing and other scenarios.
- Produce SD card starter: suitable for R&D debugging, fast startup and other scenarios.
- Produce SD card burner: suitable for mass production burning eMMC

### 5.1. PhoenixSuit Burning

#### 1) Tool requirements

- Development board MYD-LT527 development board a piece of
- Type\_C one
- 12V power adapter
- PhoenixSuit v1.1.0 (official software) *"/03\_Tools/"* directory get

#### 2) Connecting Hardware

Connect the hardware, connect the USB port of the USB to Type-C cable to the USB port on top of the development board's J5 cradle, and the Type-C end to the computer.

**Note:** Some computers may not have a Type-C port or the power connector belongs to Type-C but can not be used to burn, you need to buy a USB to Type-C adapter, as shown in the following figure:



Figure 5-1. USB to Type-C Adapter Diagram



### 3) Burning a system under windows

- Using the PhoenixSuit tool

First unzip "03-Tools/phoenixsuit.rar" tool to windows, then double click "PhoenixSuit\_EN.msi" to install the English version. Just choose the default configuration during installation and finally open the "PhoenixSuit" software.

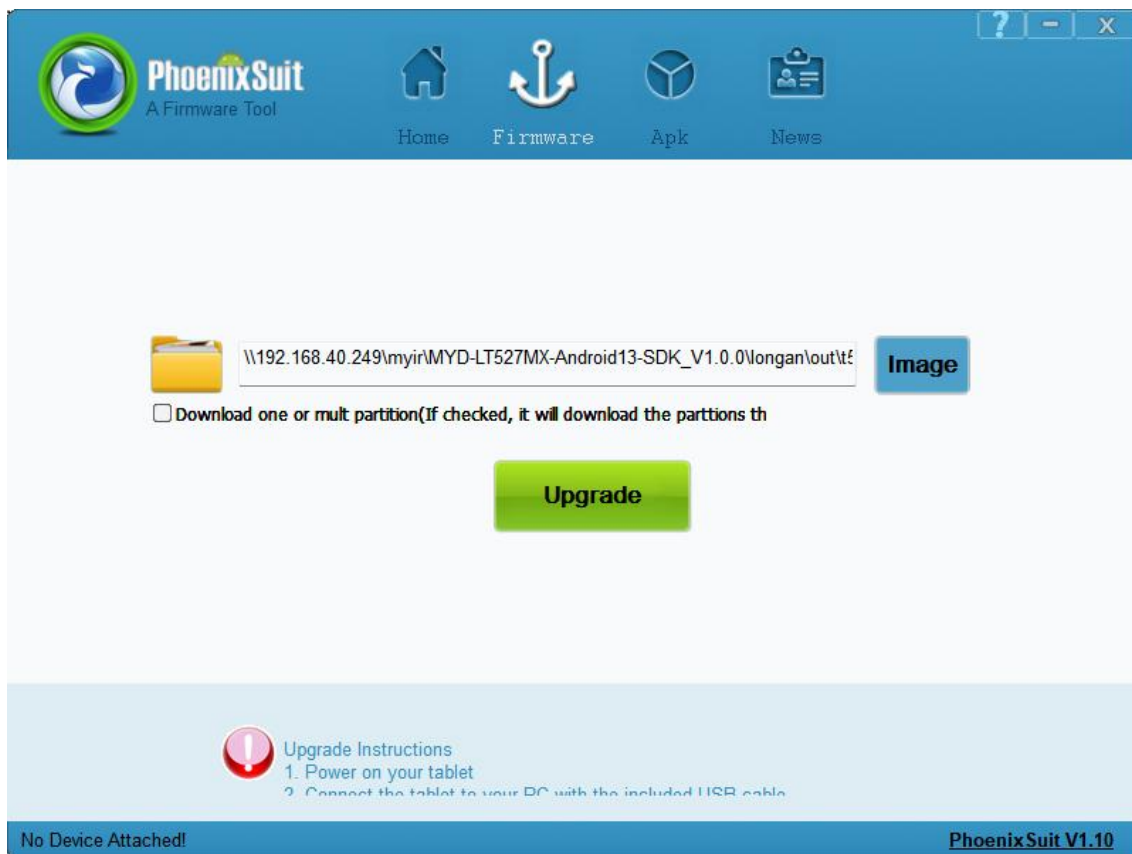


Figure 5-2. PhoenixSuit Burning Tool

First, click on "Image", select the image you want to burn, and then follow the steps in the red numbers.





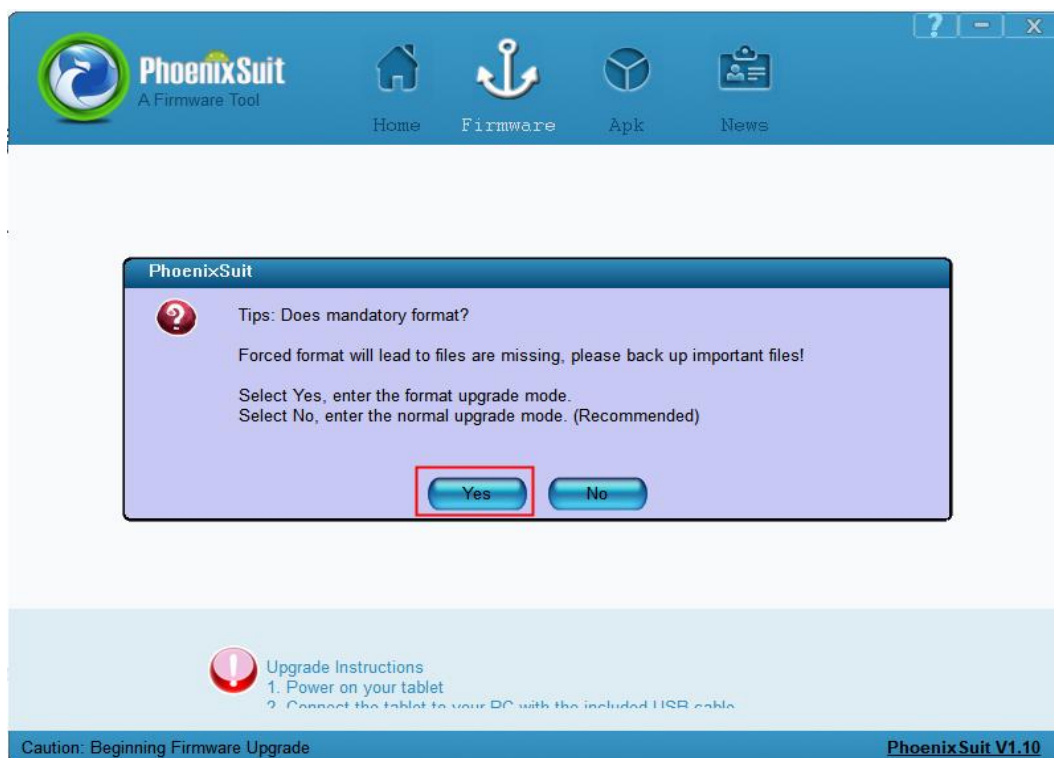


Figure 5-4. Burning Tips

Select "Yes" at this point and wait for the burn to complete.

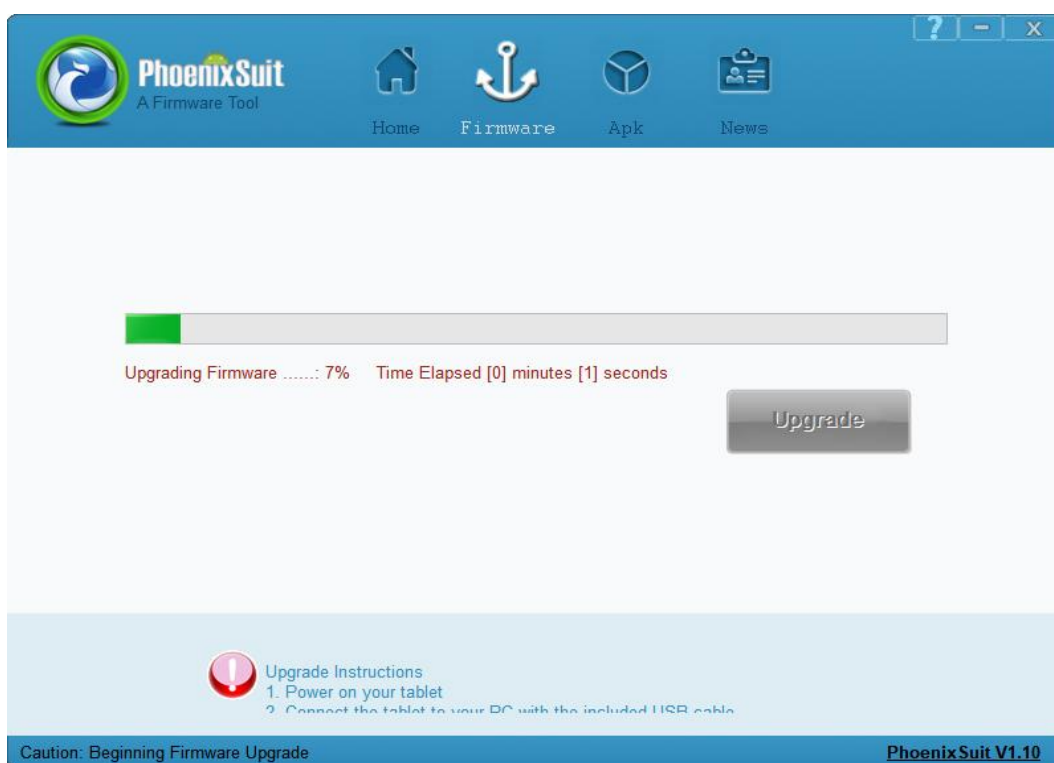


Figure 5-5. 镜像烧录

If the above steps can not be burned, may be the USB device driver abnormal, open windows device manager, found that the USB Device (VID\_1f3a\_PID\_efe8)



for the yellow exclamation mark, this time you need to update the USB driver, in the download resource file to find "03\_Tools Find the "03\_Tools" directory in the download resource file, unzip the "USBDriver.zip" compressed package, and install it manually.

## 5.2. PhoenixCard Burning

PhoenixCard can be used to create startup cards for quick testing and debugging, as well as mass production cards for mass production. The steps for creating these two types of cards are completed as follows.

### 1) Creating an SD boot card

First unzip "PhoenixCard\_V4.2.8\_EN.zip" in the "03\_Tools" directory, then find the "PhoenixCard.exe" executable program. Then find the "PhoenixCard.exe" executable program and double click to open it.

Insert the 16GB SD card into the windows USB port via SD card reader, as shown below, select "Image" path; select "start up" and click "Burn" button. "button to automatically complete the production.

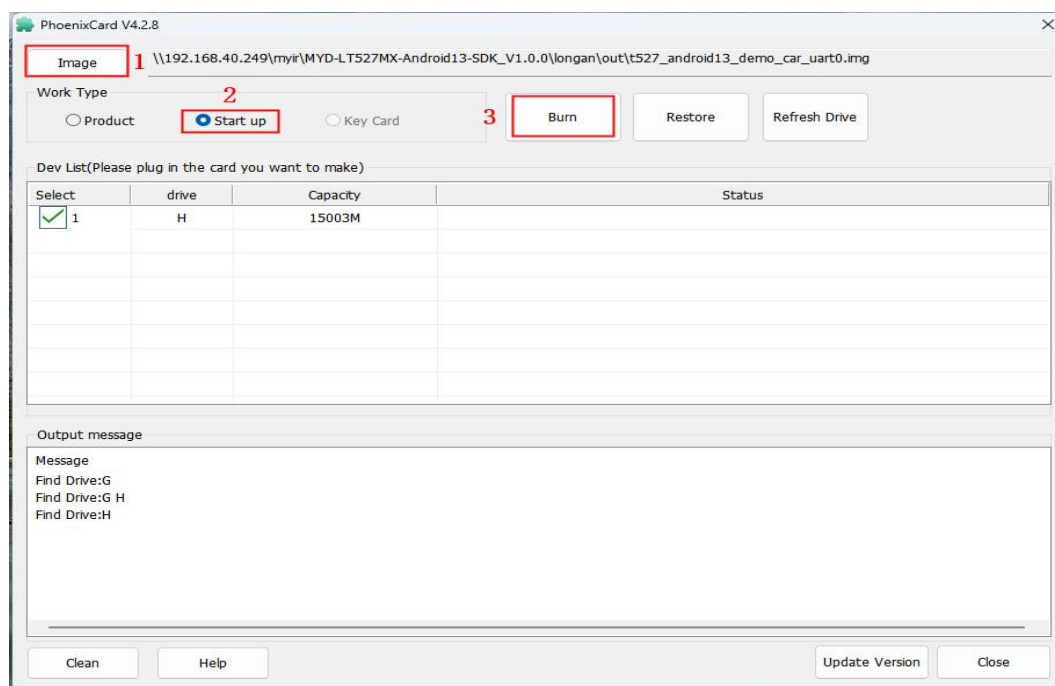


Figure 5-6. PhoenixCard Burning Boot Card

The burning process takes a few minutes to complete, and the final burn is shown in the following figure:



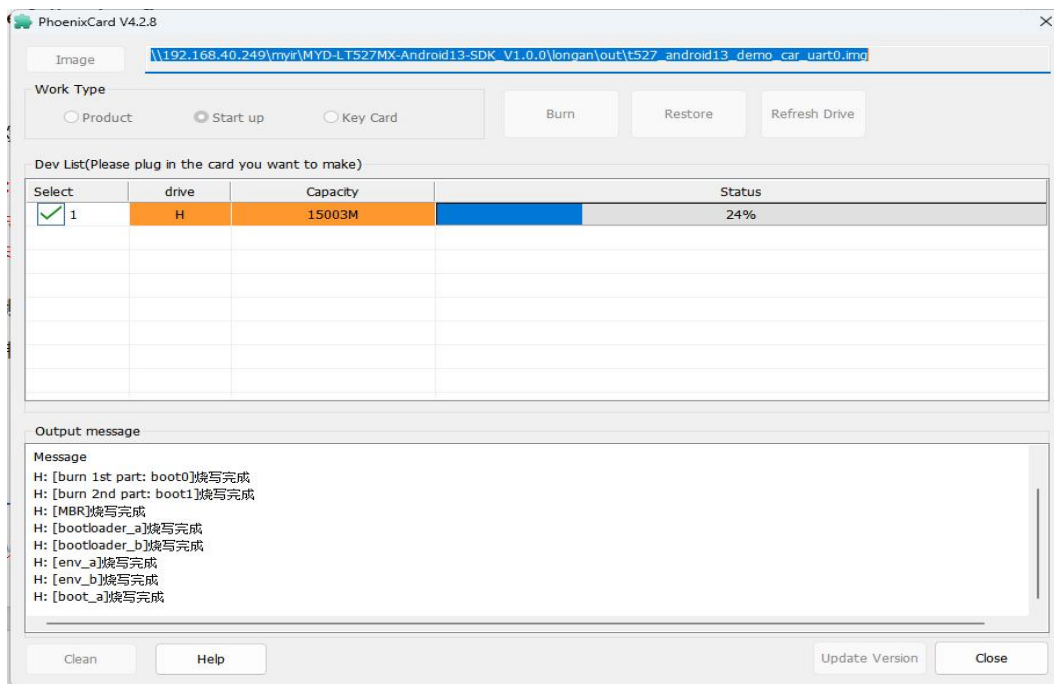


Figure 5-7. Burning in the image

After burning is complete, just insert the SD card into the SD card slot of the development board (J6).

## 2) Creating an SD burner card

The steps are almost the same as for the startup card, except that you need to change the "start up" option to "product", as shown in the following figure:

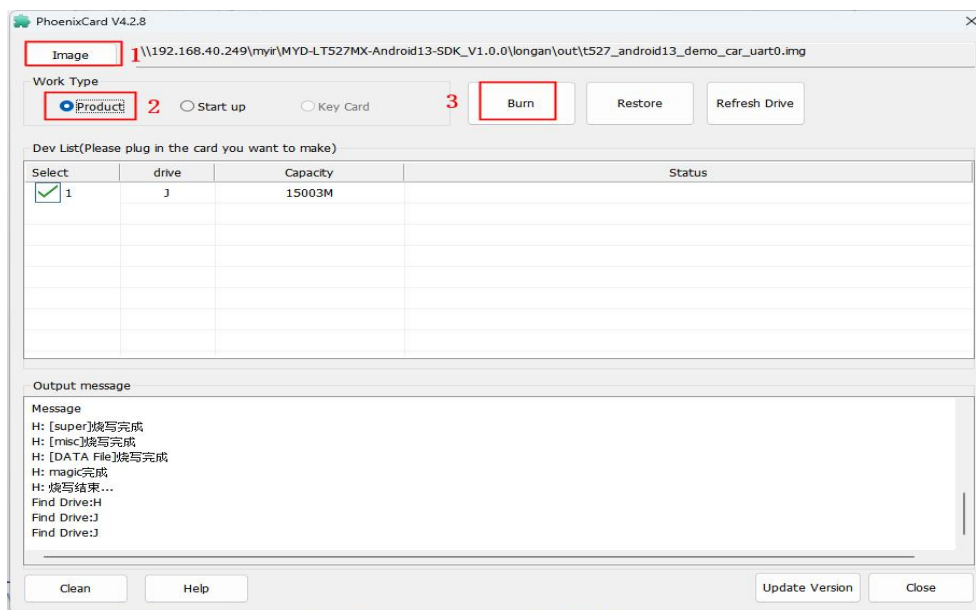


Figure 5-8. production card





The burning process takes a few minutes to complete, and the final burn is shown in the following figure

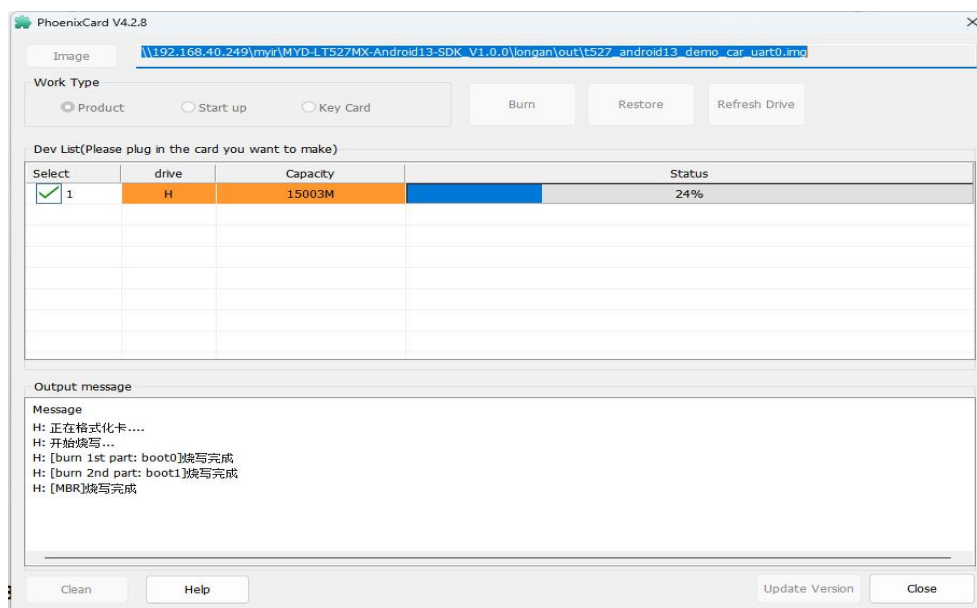


Figure 5-9. Burning in the image

### 3) Burn image to eMMC

The development board is first powered off, and then the SD burner card is inserted into the SD card slot (J6) of the development board, and finally powered on and waited for the completion of the burner, and the burner prints the information as follows:

```

===== (Part of the burning process is omitted) =====
[21.802]succeeded in writting part vendor_boot_a
origin_verify value = e858d530, active_verify value = e858d530
[21.973]succeeded in verify part vendor_boot_a
[21.977]succeeded in download part vendor_boot_a
[21.981]begin to download part init_boot_a
partdata hi 0x0
partdata lo 0x800000
sparse: bad magic
[22.410]succeeded in writting part init_boot_a
origin_verify value = 7c682b4c, active_verify value = 7c682b4c
[22.461]succeeded in verify part init_boot_a
[22.465]succeeded in download part init_boot_a
    
```



```
[22.469]begin to download part super
partdata hi 0x0
partdata lo 0x4d0de24c
chunk 0(44)
chunk 1(44)
chunk 2(44)
chunk 3(44)
chunk 4(44)
```

===== (Part of the burning process is omitted) =====

```
[133.408]succeeded in writting part Reserve0
origin_verify value = 0, active_verify value = 0
[133.500]succeeded in verify part Reserve0
[133.503]succeeded in download part Reserve0
[133.511]succeeded in downloading part
uboot size = 0x150000
storage type = 2
sunxi_sprite_deal_uboot ok
[133.649]succeeded in downloading uboot
[133.657][mmc]: write mmc 2 info ok
storage type = 2
[133.669]succeeded in downloading boot0
CARD OK
[133.673]sprite success
>>>[144>{"step":"firmware","PN":"xxx","SN":"xxx","CN":"xxx","result":{"module":"i
mage","progress":"20","state":"0","message":"write image succuess"}}
sprite_next_work=3
next work 3
SUNXI_UPDATE_NEXT_ACTION_SHUTDOWN
[136.695][mmc]: mmc exit start
[136.713][mmc]: mmc 2 exit ok
```



The last red character appears, indicating that the burning is complete, at this time disconnect the power supply, and then be sure to remove the SD card, and then re-powered can be.



## 6. How to Modify Board Support Packages

The previous chapters have been more complete description of the android SDK project based on the construction of the system image running on the MYD-LT527 development board, and burn the image to the development board of the complete process. Because many pins of the MYD-LT527 core board have the characteristic of multiple function reuse, there are always some differences between the base board designed based on the MYC-LT527X core board and the MYB-LT527X in the actual project.

These differences may be to remove the display, add more GPIOs, or need to add more serial ports, there may be through the SPI, I2C, USB, etc. to extend some peripherals, etc.; In addition to hardware differences, there are also some system component differences, such as focusing on the back-end management of the application, you may need to be more complete network applications, and so on. This requires the system developer to do some trimming and porting work on the basis of the code we provide. This chapter describes the process of developing and customizing your own system from a system developer's point of view, and lays the foundation for adapting your own hardware later on.

### 6.1. Board Support Package Introduction

In order to adapt to the user's new hardware platform, you first need to understand what resources are provided by MYIR's MYD-LT527 development board, and you can check the "MYD-LT527 Android SDK Release Notes" for specific information. In addition, we also focus on the various parts of the BSP need to change some of the file list is organized to facilitate the user to find and modify the specific contents of the following table:

Table 6-1.BSPconfiguration information

Projects	Device Tree	Description
U-boot	longan\device\config\chips\t527\configs\demo_car\uboot-board.dts	Board Level Equipment Tree





	device/softwinner/saturn/common/system/env.cfg	Uboot Environment Variable Configuration
Kernel	longan\device\config\chips\t527\configs\demo_car\board.dts	Board Level Device Tree
bsp	longan\bsp\configs\linux-5.15\sun55iw3p1.dtsi	Soc device tree
	longan\bsp\configs\linux-5.15\panel-atk-10-1.dtsi	10" LVDS display configuration
	longan\bsp\configs\linux-5.15\panel-hontron-7.dtsi	7" LVDS display configuration

A board support package (BSP) is a collection of information that defines how a particular hardware device, set of devices, or hardware platform is to be 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 separately 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, using the MYC-LT527X core board design of the hardware BSP code can be viewed in the linux SDK section.

Brandy contains only Bootloader part of spl and u-boot, this part mainly implements the core hardware, such as DDR, Clock initialization and kernel boot. Based on the MYC-LT527X kernel board hardware modification of this part of the content, so this part of the content usually do not need to modify, has been adapted to complete.

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

The kernel contains the Linux kernel part, which mainly implements the kernel and peripheral firmware contents (only the native kernel is included).

kernel/



## └─ linux-5.15

bsp in the storage allwinner bsp module related code, allwinner in the kernel version 5.15 will be their own driver and the native kernel driver separate storage.

```
└─ configs
└─ drivers
└─ include
└─ Kconfig
└─ Makefile
└─ modules
└─ platform
└─ ramfs
```

When designing a product using MYIR's core board, the bootloader part may not need to be modified if there is no special need. 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.

## 6.2. Onboard uboot compilation and update

U-boot is a very feature-rich open source boot loader, including kernel boot, download updates and many other aspects of the embedded field is very widely used, you can check the official website to get more information <http://www.denx.de/wiki/U-Boot/WebHome>. The T5 platform also uses uboot as the boot loader.

### 6.2.1. Compile uboot in SDK separately

Once the user has iterated the development process to change the U-boot code, the SDK can be used to build the entire image. It is also possible to compile the uboot source code separately, but it is not used as a separate brush write:

```
$HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0/longan$  
./build.sh bootloader
```

### 6.2.2. Update uboot



After u-boot is compiled separately, you need to go back to the top level directory of the source code and compile the Android part according to section 3.4.2, and finally generate the .img image file.

Just update it at the end according to chapter 5.1.

## 6.3. Onboard kernel compilation and update

Linux kernel is a very large open source kernel used in various distributions of operating systems, Linux kernel with its portability, multiple network protocols support, independent module mechanism, MMU and many other rich features, so that the Linux kernel can be widely used in embedded systems.

Meanwhile, T5 also supports Linux kernel and will get long-term stable update, MYD-LT527 uses T5 kernel porting, the latest support for Linux kernel version 5.15.

### 6.3.1. Compile the kernel separately in the SDK

#### 1). Configure the kernel (not required)

MYIR has already integrated most of the functionality into the kernel and generally does not need to be configured. If you need to add special features, the peripheral drivers should be configured as follows.

- **Enter the kernel directory**

Execute the following command to load the specified defconfig configuration

```
$HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0/longan$  
./build.sh loadconfig android13_arm64_defconfig
```

- **Modify the kernel**

Execute the following command to open the menuconfig configuration

```
$HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0/longan$  
./build.sh menuconfig
```



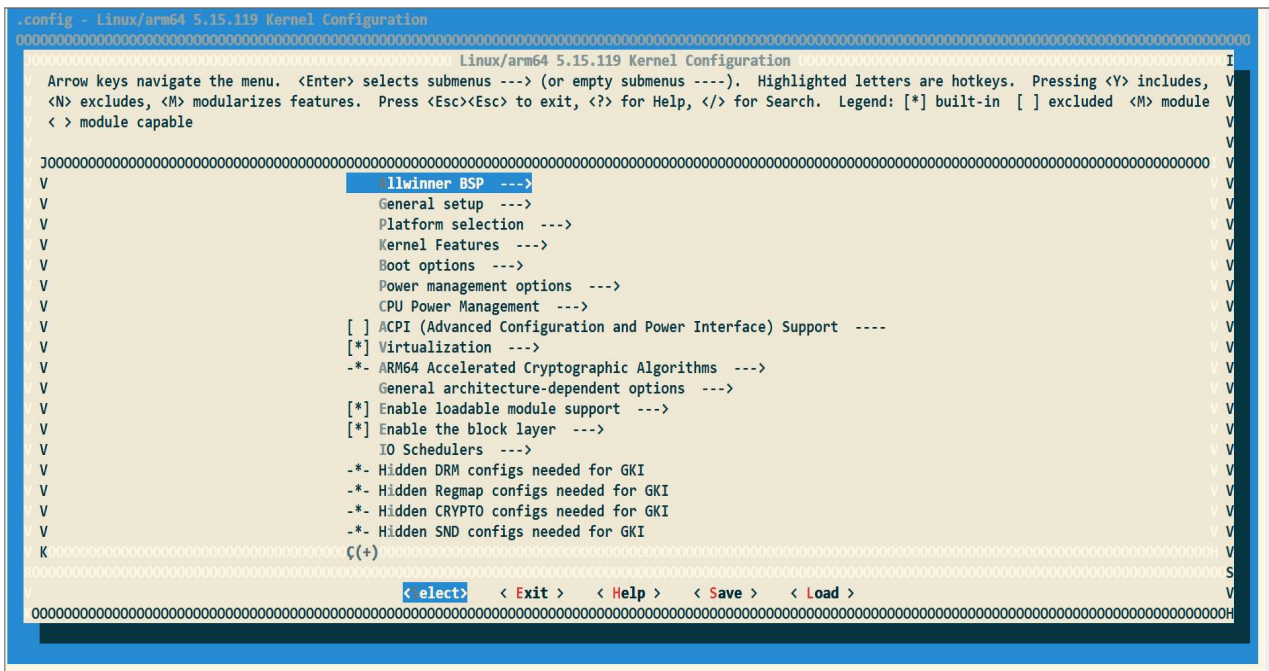


Figure 6-1. menuconfig Configuration interface

- **Saving the kernel configuration**

Execute the following command to save the previous step in menuconfig

```
$HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0/longan$  
./build.sh saveconfig
```

## 2). Compile kernel separately

After the user iterates the development process to change the kernel code or modify the device tree, the SDK can be used to build the entire image. It is also possible to compile the kernel source code separately, but it is not used as a separate brush write:

```
$HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0/longan$  
./build.sh kernel
```

### 6.3.2. Update kernel

After the kernel is compiled separately, you need to go back to the top level directory of the source code and compile the Android part according to section 4.2, and finally generate the .img image file

Just update it at the end according to chapter 5.1

## 6.4. Onboard Android Compilation and Updates





### 6.4.1. Compile Android in SDK separately

When the user iterative development process to change the code of android, you can use the SDK to build the entire image. But not as a separate brush write use:

First go to the top-level directory of the source code and execute the following command:

```
$HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0$  
make
```

This step is compiling the android separately.

Note: If you are in this directory for the first time or re-enter it, you need to execute the "source build/envsetup.sh" and "launch" commands, please refer to section 4.2 for details, if you have been in this directory all along, you don't need to execute these steps. If you have always been in the same directory, you do not need to perform these steps.



# 7. How to adapt your hardware platform

In order to adapt to the user's new hardware platform, the first thing you need to know is what resources are provided by MYIR's MYD-LT527 development board, and the specific information can be viewed in the "MYD-LT527 Android SDK Release Notes". In addition, users also need to have a detailed understanding of the CPU chip manual, as well as the MYC-LT527X core board's product manual, pin definition, in order to facilitate the correct configuration and use of these pins according to the actual function.

## 7.1. How to create your device tree

### 7.1.1. Onboard Device Tree

Users can create their own device tree in the BSP source code, and generally do not need to modify the code in the Bootloader section. Users only need to adjust the Linux kernel device tree according to the actual hardware resources. The device trees in each part of the BSP of MYD-LT527 are listed here for users' reference, as shown in the table below:

Table 7-1.Device Tree Configuration Information

Projects	Device Tree	Description
U-boot	longan\device\config\chips\t527\configs\demo_car\uboot-board.dts	Board Level Device Tree
Kernel	longan\device\config\chips\t527\configs\demo_car\board.dts	Board Level Device Tree
bsp	longan\bsp\configs\linux-5.15\sun55iw3p1.dtsi	Soc device tree
	longan\bsp\configs\linux-5.15\panel-atk-10-1.dtsi	10" LVDS display configuration



	longan\bsp\configs\linux-5.15\panel-hontron-7.dtsi	7" LVDS display configuration
--	--	-------------------------------

## 7.1.2. Adding a Device Tree

The Linux kernel device tree is a data structure that describes on-chip and off-chip device information in a unique syntax format. It is passed to the kernel by the BootLoader, which parses it to form a dev structure associated with the driver for use by the driver code. The following is an example of how to add a device tree to the kernel.

### 1) Kernel adds device tree

In the kernel source code *"longan\bsp\configs\linux-5.15\"* directory you can see a number of device tree, you can add a customized device tree in the current path:

名称	修改日期	类型	大小
debug_defconfig.fragment	2023/9/21 10:04	FRAGMENT 文件	1 KB
fttrace_defconfig.fragment	2023/9/21 10:04	FRAGMENT 文件	1 KB
lock_defconfig.fragment	2023/9/21 10:04	FRAGMENT 文件	1 KB
mem_defconfig.fragment	2023/9/21 10:04	FRAGMENT 文件	1 KB
myir-lt527-lvds-7-inch.dtsi	2023/12/14 21:42	DTSI 文件	6 KB
myir-lt527-lvds-10-inch.dtsi	2023/12/14 16:05	DTSI 文件	2 KB
sun8iw20p1.dtsi	2023/9/21 10:04	DTSI 文件	25 KB
sun8iw20p1smp_min_defconfig	2023/9/21 10:04	文件	3 KB
sun50iw10p1.dtsi	2023/12/4 17:51	DTSI 文件	83 KB
sun50iw10p1_min_defconfig	2023/9/21 10:04	文件	3 KB
sun50iw10p1_min_standby_defconf...	2023/9/21 10:04	文件	3 KB
sun50iw10p1-clk.dtsi	2023/9/21 10:04	DTSI 文件	21 KB
sun55iw3p1.dtsi	2023/12/8 9:40	DTSI 文件	128 KB
sun55iw3p1_min_defconfig	2023/9/21 10:04	文件	3 KB
sun55iw3p1_standby_min_defconfig	2023/9/21 10:04	文件	4 KB
sun55iw5_fpga_min_defconfig	2023/9/21 10:04	文件	4 KB
sun55iw5p1.dtsi	2023/9/21 10:04	DTSI 文件	16 KB
sun55iw6p1.dtsi	2023/9/21 10:04	DTSI 文件	22 KB
sun55iw6p1_fpga_min_defconfig	2023/9/21 10:04	文件	4 KB
sun60iw1p1.dtsi	2023/9/21 10:04	DTSI 文件	56 KB
sun60iw1smp_min_defconfig	2023/9/21 10:04	文件	3 KB

Figure 7-1. Customizing the Device Tree Catalog

Among them, "myir-lt527-lvds-10-inch.dtsi" and "myir-lt527-lvds-7-inch.dtsi" are device trees added by MYIR itself, you can refer to them by References

To change the configuration of a different monitor, you need to modify the following files :



PATH: `longan\device\config\chips\t527\configs\demo_car\board.dts`

At the end of the document:

Select 7" LVDS Solution

```
#if 0
#include "myir-lt527-lvds-10-inch.dtsi "
#else
#include "myir-lt527-lvds-7-inch.dtsi"
#endif
```

Select 10" LVDS Solution

```
#if 1
#include "myir-lt527-lvds-10-inch.dtsi "
#else
#include "myir-lt527-lvds-7-inch.dtsi"
#endif
```

## 7.2. How to Configure CPU Function Pins According to Your Hardware

The realization of a functional pin control is one of the more complex system development process, which contains the configuration of the pin, the development of the driver, the implementation of the application and other steps, this section does not specifically analyze each part of the development process, but rather an example to explain the implementation of the control of the functional pins.

### 7.2.1. Methods of GPIO Configuration

GPIO: General-purpose input/output, general-purpose input/output port, in embedded devices is a very important resource, you can output high and low levels through them or through them to read the state of the pin is high or low,



MYD-LT527 package with peripheral controllers, these peripheral controllers and external devices are generally handled Through the control of the GPIO to realize, and will be GPIO peripheral controller we call the use of multiplexing (Alternate Function), to give them more complex functions, such as the user can be through the GPIO port and external hardware data interaction (such as UART), control hardware work (such as LEDs, buzzers, etc.), read the hardware status signals (such as interrupt), etc., so the GPIO port can be used to control the hardware work (such as interrupt signals), so the GPIO port can be used for data interaction (such as UART). signals), etc., so the GPIO port is very widely used.

## 1) Query the manual to get the reuse relationship

### ● Determine the pins of the core board

All available GPIOs of the core board can be found through the pinlist file "MYC-LT527\_V01-PinList-20231115.xlsx" organized by MYIR, and the defaults used by the current SDK can also be determined.

This will be illustrated with an actual GPIO pin as a case study: the PB7, and here are all the multiplexing relationships for the PB7.

MYIR 米尔电子 Make Your Idea Real											
Ver: 1.0 2023.11.15											
Num	Pin Name	BGA664 Ball	MODE							Power Rail	Comment
			MUX0	MUX2	MUX3	MUX4	MUX5	MUX6	MUX7		
1	PB0		PB0	UART2-TX	SPI2-CS0	JTAG-MS	LCD0-D0	PWM6	PB-EINT0	3V3	
2	PB1		PB1	UART2-RX	SPI2-CLK	JTAG-CK	LCD0-D1	PWM7	PB-EINT1	3V3	
3	PB2		PB2	UART2-RTS	SPI2-MOSI	JTAG-DO	LCD0-D8	CAN-TX0	PB-EINT2	3V3	
4	PB3		PB3	UART2-CTS	SPI2-MISO	JTAG-DI	LCD0-D9	CAN-RX0	PB-EINT3	3V3	
5	PB4		PB4	TWI1-SCK	I2S0-MCLK		PWM8	HDMI-SCL	PB-EINT4	3V3	
6	PB5		PB5	TWI1-SDA	I2S0-BCLK		PWM9	HDMI-SDA	PB-EINT5	3V3	
7	PB6		PB6		I2S0-LRCK		PWM10	HDMI-CEC	PB-EINT6	3V3	
8	PB7		PB7	OWA-IN	I2S0-DOUT0	I2S0-DIN1	LCD0-D16	PWM11	PB-EINT7	3V3	
9	PB8		PB8	OWA-OUT	I2S0-DIN0	I2S0-DOUT1	LCD0-D17	PWM0	PB-EINT8	3V3	

Figure 7-2. GPIO Pin List

## 2). Referencing GPIO Methods in the Device Tree

### ● Configure pins for GPIO functionality

A GPIO-controlled node can be designed in the device tree, and a specific driver can be used to access this node and control the gpio. Subsequent chapters will show how the driver references the device tree node.

Configuration method can be done by adding nodes to the device tree

PATH: [longan\device\config\chips\t527\configs\demo\\_car\board.dts](#)



```
gpioctr_device {
compatible = "myir,gpioctr";
status = "okay";
gpioctr-gpios = <&gpio 7 0>;
};
```

- **Configure pins for other specific functions**

MYD-LT527 development board defines and implements a number of rich features, but at the same time also occupies a large number of pin resources, here on the LVDS display control pins (LVDS0-D0P) as an example, to illustrate how to use the GPIO multiplexing function, the same as the first to understand the pin multiplexing relationships.

Pin Name	IO Type	Function2	Function3	Function4	Function5	Function6	Function14
PD0	I/O	LCD0-D2	LVDS0-D0P	DSIO-D0P	PWM0-0		PD-EINT0
PD1	I/O	LCD0-D3	LVDS0-D0N	DSIO-D0N	PWM0-1		PD-EINT1
PD2	I/O	LCD0-D4	LVDS0-D1P	DSIO-D1P	PWM0-2		PD-EINT2
PD3	I/O	LCD0-D5	LVDS0-D1N	DSIO-D1N	PWM0-3		PD-EINT3
PD4	I/O	LCD0-D6	LVDS0-D2P	DSIO-CKP	PWM0-4		PD-EINT4
PD5	I/O	LCD0-D7	LVDS0-D2N	DSIO-CKN	PWM0-5		PD-EINT5
PD6	I/O	LCD0-D10	LVDS0-CKP	DSIO-D2P	PWM0-6		PD-EINT6
PD7	I/O	LCD0-D11	LVDS0-CKN	DSIO-D2N	PWM0-7		PD-EINT7
PD8	I/O	LCD0-D12	LVDS0-D3P	DSIO-D3P	PWM0-8		PD-EINT8
PD9	I/O	LCD0-D13	LVDS0-D3N	DSIO-D3N	PWM0-9		PD-EINT9
PD10	I/O	LCD0-D14	LVDS1-D0P	DSI1-D0P	PWM0-10	SPI1-CS0/DBI-CSX	PD-EINT10
PD11	I/O	LCD0-D15	LVDS1-D0N	DSI1-D0N	PWM0-11	SPI1-CLK/DBI-SCLK	PD-EINT11
PD12	I/O	LCD0-D18	LVDS1-D1P	DSI1-D1P	PWM0-12	SPI1-MOSI/DBI-SDO	PD-EINT12

Figure 7-3. GPIO Multiplexing Relationship Table

From the above figure, we can see that the function3 multiplexing relationship of PD0 is used as the data signal of LVDS.

The following pin configurations can be made:

PATH: longan\device\config\chips\t527\configs\demo\_car\board.dts

```
lvds0_pins_a: lvds0@0 {
    pins = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5", "PD6", "PD7", "PD8", "PD9";
    function = "lvds0";
    drive-strength = <30>;
};

lvds0_pins_b: lvds0@1 {
    pins = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5", "PD6", "PD7", "PD8", "PD9";
};
```



```
function = "gpio_in";  
};
```

## 7.3. How to use your own configured pins

The pins that we have configured in u-boot or Kernel's device tree can be used in Kernel to enable control of the pins.

### 7.3.1. Using GPIO pins in kernel drivers

- Use of standalone IO drivers

In the second point of the 6.2.1 chapter in the device tree example, has been defined to complete the gpio node information, the following will use the kernel driver to realize the control of the GPIO (PB7 pin 1 and 0, such as the need to detect the need to use a multimeter to test the change in pin level)

```
//gpiocr.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>
```



```
static int major = 0;
static struct class *gpiocr_class;
static struct gpio_desc *gpiocr_gpio;

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(gpiocr_gpio, status);

    return 1;
}

static int gpio_drv_open (struct inode *node, struct file *file)
{
    gpiod_direction_output(gpiocr_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;
}

static struct file_operations gpiocr_drv = {
    .owner    = THIS_MODULE,
    .open     = gpio_drv_open,
    .read     = gpio_drv_read,
    .write    = gpio_drv_write,
    .release  = gpio_drv_close,
};

static int chip_demo_gpio_probe(struct platform_device *pdev)
{
    gpiocr_gpio = gpiod_get(&pdev->dev, "gpiocr", 0);
    if (IS_ERR(gpiocr_gpio)) {
        dev_err(&pdev->dev, "Failed to get GPIO for led\n");
        return PTR_ERR(gpiocr_gpio);
    }

    major = register_chrdev(0, "myir_gpiocr", &gpiocr_drv); /* /dev/gpiocr */

    gpiocr_class = class_create(THIS_MODULE, "myir_gpiocr_class");
    if (IS_ERR(gpiocr_class)) {
        printk("%s %s line %d\n", __FILE__, __FUNCTION__, __LINE__);
        unregister_chrdev(major, "gpiocr");
        gpiod_put(gpiocr_gpio);
        return PTR_ERR(gpiocr_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" },
    {},
};

/* 定义 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,
    },
};

static int __init gpio_init(void)

```



```
{
    int err;
    err = platform_driver_register(&chip_demo_gpio_driver);

    return err;
}

static void __exit gpio_exit(void)
{
    platform_driver_unregister(&chip_demo_gpio_driver);
}

module_init(gpio_init);
module_exit(gpio_exit);

MODULE_LICENSE("GPL");
```

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

- **The driver example will be configured directly into the kernel**

Create a new file "gpioctr.c" in the */drivers/char* folder of the kernel source code, copy the above driver code into it, and modify the Kconfig, Makefile and defconfig.

In the */drivers/char/Kconfig* file add the

```
config SAMPLE_GPIO
    tristate "this is a gpio test driver"
```

In the */drivers/char/Makefile* file add the:

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

Add to the file *longan/device/config/chips/t527/configs/demo\_car/linux-5.15/android13\_arm64\_defconfig*:

```
CONFIG_SAMPLE_GPIO=y
```



Update the kernel according to chapter 5.3 and then burn it to the development board

- **Driver examples compiled into separate modules**

Add gpioctr.c to the working directory and copy the above driver code, and write a separate Makefile program in the same directory.

```
PC$: $HOME/gpioctr$ ls
gpioctr.c Makefile
PC$: $HOME/gpioctr$ vi Makefile
# modifications KERN_DIR
#KERN_DIR = # Directory of kernel source code used by the board
KERN_DIR = $HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0/longan/kernel/linux-5.15

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
```

Load SDK environment variables to the current shell. please refer to section 2.2.2 Setting up the compilation chain.

Make sure the kernel is compiled before executing the command, if not, please refer to section 4.1 to compile the kernel once first.

Execute the make command after compiling the kernel to generate the gpioctr.ko driver module file.

```
PC$: cd /$HOME/gpioctr
PC$: $HOME/gpioctr$ make
make -C $HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0/kernel M=`pwd` modules
```



```
make[1]: Entering directory '$HOME/MYD-LT527/MYD-LT527-Android13-SDK_V1.0.0/kernel'
```

```
CC [M] /home/sur/gpioctr/gpioctr.o
```

```
MODPOST /home/sur/gpioctr/Module.symvers
```

```
CC [M] /home/sur/gpioctr/gpioctr.mod.o
```

```
LD [M] $Home/gpioctr/gpioctr.ko
```

```
make[1]: Leaving directory '$HOME/ MYD-LT527-Android13-SDK_V1.0.0/kernel'
```

```
PC$: $HOME/gpioctr$ ls
```

```
gpioctr.c gpioctr.ko gpioctr.mod gpioctr.mod.c gpioctr.mod.o gpioctr.o Makefile modules.order Module.symvers
```

After successful compilation, the gpioctr.ko file can be transferred to the development board via adb command, Ethernet, USB flash drive and other transfer media to load the driver using the insmod command.

Different external devices have their own independent driver code and architecture implementation, in the different peripheral driver modification, debugging, you need to comply with the respective driver framework. Such as touch screen, keyboard, etc. need to use the input driver architecture; ADC and DAC using the IIO architecture, the display device using the DRM driver architecture, etc., this section does not do all the driver development of a specific explanation

### 7.3.2. Configuration of the HAL layer

Continuing from the previous section on GPIO control, we will introduce how the gpio is called and controlled by the HAL layer. the HAL layer needs to write a JNI interface to call the underlying layer, and the following APK written in JAVA will be used to realize the GPIO control (on the MYD-LT527's backplane, we design PB7 to control a red LED, as shown in the figure below. When GPIO (PB7) is configured to 0, the light will be lit. (The opposite will be off)



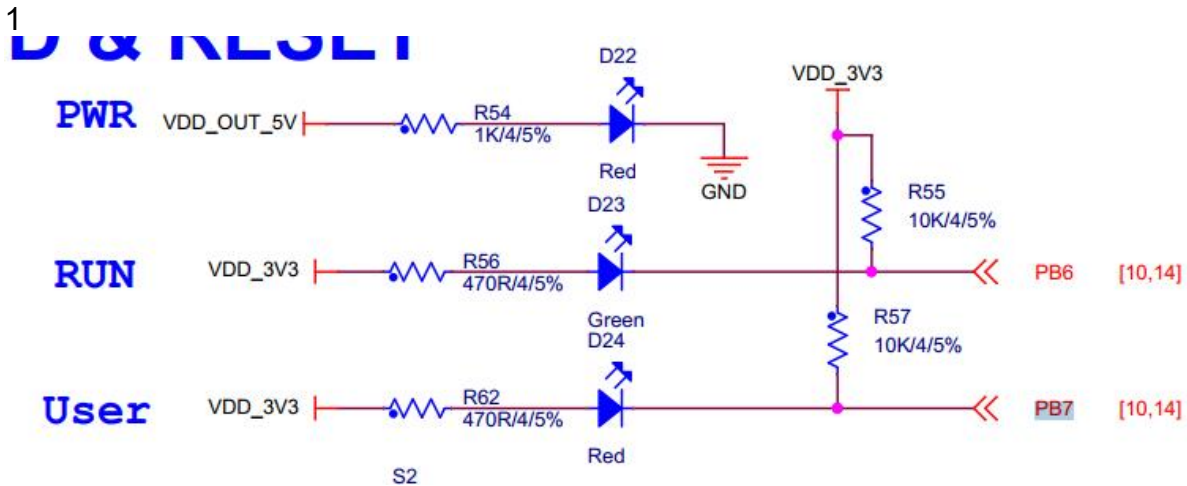


Figure 7-4. LED Schematic

The JNI interfaces are as follows:

```
#include <stdio.h>
#include "com_example_myapplication_Gpio.h"
#include <android/log.h>

#define TAG "jni_test"

JNIEXPORT jint JNICALL Java_com_example_myapplication_Gpio_nativeWriteGpio
(JNIEnv *env, jclass cls, jstring j_path, jstring j_value)
{
    __android_log_print(ANDROID_LOG_DEBUG, TAG, "Java_com_example_myapplic
ation_Gpio_nativeWriteGpio");
    const char* path = (*env)->GetStringUTFChars(env, j_path, NULL);
    if (path == NULL)
    {
        __android_log_print(ANDROID_LOG_DEBUG, TAG, "out of memory");
        return -1;
    }

    const char* value = (*env)->GetStringUTFChars(env, j_value, NULL);
    if (value == NULL) {
        (*env)->ReleaseStringUTFChars(env, j_path, path);
        __android_log_print(ANDROID_LOG_DEBUG, TAG, "ReleaseStringUTFChars");
        return -1;
    }
}
```



```

}

    FILE* file = fopen(path, "w");
    if (file != NULL) {
        fputs(value, file);
        fclose(file);
        __android_log_print(ANDROID_LOG_DEBUG, TAG, "Value written to file successfully");
        (*env)->ReleaseStringUTFChars(env, j_path, path);
        (*env)->ReleaseStringUTFChars(env, j_value, value);
        return 0;
    } else {
        (*env)->ReleaseStringUTFChars(env, j_path, path);
        (*env)->ReleaseStringUTFChars(env, j_value, value);
        __android_log_print(ANDROID_LOG_DEBUG, TAG, "Failed to open file: %s", path);
        return -1;
    }
}

JNIEXPORT jint JNICALL Java_com_example_myapplication_Gpio_nativeReadGpio(JNIEnv *env, jclass cls, jstring j_path)
{
    #if 1
        const char* path = (*env)->GetStringUTFChars(env, j_path, NULL);
        if (path == NULL)
        {
            //printf("out of memory.\n");
            return -1;
        }

        FILE *file = fopen(path, "r");
        if (file != NULL) {

```



```

        int value;
        fgets(&value, sizeof(value), file);
        //printf("File read value: %s", value);
        fclose(file);
        jstring j_value = (*env)->NewStringUTF(env, value);
        return j_value;
    } else {
        //printf("Failed to open file: %s\n", path);
        return NULL;
    }
#endif
    return 0;
}

```

After the APK calls the JNI interface, you can control the brightness of D24. the APK control code is as follows:

MainActivity.java:

```

package com.example.myapplication;

import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;

import com.example.myapplication.databinding.ActivityMainBinding;

import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

public class MainActivity extends AppCompatActivity {

```





```
// Used to load the 'myapplication' library on application startup.
static {
    System.loadLibrary("myapplication");
}

private static final ScheduledExecutorService scheduler = Executors.newScheduledThreadPool(1);
private static boolean isLedOn = false;
private static final String TAG = "MainActivity";

public static void led_main() {
    long interval = 1000;
    Log.d(TAG, "led_main 1");
    scheduler.scheduleAtFixedRate(() -> {
        Log.d(TAG, "led_main 2");
        if (isLedOn) {
            Log.d(TAG, "led_main 4");
            int result = Gpio.setStandbyLedOn(false);
            Log.d(TAG, "led_main 5");
            if (result == 0) {
                Log.d(TAG, "Turn off LED light successfully");
            } else {
                Log.d(TAG, "Failed to turn off LEDs");
            }
            isLedOn = false;
        } else {
            Log.d(TAG, "led_main 6");
            int result = Gpio.setStandbyLedOn(true);
            Log.d(TAG, "led_main 7");
            if (result == 0) {
                Log.d(TAG, "Turn on the LED light successfully");
            } else {
```



```

        Log.d(TAG, "Failed to turn on LEDs");
    }
    isLedOn = true;
}
}, 0, interval, TimeUnit.MILLISECONDS);
Log.d(TAG, "led_main 3");
}

private ActivityMainBinding binding;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    binding = ActivityMainBinding.inflate(getLayoutInflater());
    setContentView(binding.getRoot());

    // Example of a call to a native method
    TextView tv = binding.sampleText;
    tv.setText(stringFromJNI());

    Log.d(TAG, "led_main go!");
    led_main();
}

/**
 * A native method that is implemented by the 'myapplication' native library,
 * which is packaged with this application.
 */
public native String stringFromJNI();
}

```

Gpio.java:



```

ackage com.example.myapplication;

import android.util.Log;

import java.io.File;

/**
 * Class that provides access to some of the gpio management functions.
 *
 * {@hide}
 */
public final class Gpio {
    public static final String TAG = "GPIO";
    private static boolean ServiceReady = false;
    private static boolean mHasNormalLed = false;
    private static boolean mHasStandbyLed = false;
    private static boolean mHasNetworkLed = false;
    private static boolean DEBUG = false;

    // can't instantiate this class
    private Gpio() {
    }

    static {
        String lib_name = "libmyapplication.so";
        String lib_path[] = {"/system/lib/", "/system/lib64/", "/vendor/lib/", "/vendor/lib64/"};
        Log.d(TAG, "find jniso");
        for (int i = 0; i < lib_path.length; i++) {
            File file = new File(lib_path[i], lib_name);
            if (file.exists()) {
                Log.d(TAG, "find jniso1");
            }
        }
    }
}

```



```

        System.loadLibrary("myapplication");
        break;
    }
}
if (checkLed()){
    ServiceReady = true;
}
}

private static native int nativeWriteGpio(String path, String value);
private static native int nativeReadGpio(String path);

private static final String PATHSTR    = "/sys/class/leds";
private static final String LIGHT_NAME = "/brightness";

private static final String NORMAL_LED_PATH  = "/sys/class/leds/run_led";
private static final String STANDBY_LED_PATH = "/sys/class/leds/err_led";
private static final String NETWORK_LED_PATH = "/sys/class/leds/debug_led";

private static boolean checkLed() {
    String led[] = {NORMAL_LED_PATH, STANDBY_LED_PATH};
    File normal = new File(NORMAL_LED_PATH);
    if (normal.exists()) {
        mHasNormalLed = true;
    }
    File standby = new File(STANDBY_LED_PATH);
    if (standby.exists()) {
        mHasStandbyLed = true;
    }
    File network = new File(NETWORK_LED_PATH);
    if (network.exists()) {
        mHasNetworkLed = true;
    }
}

```



```

    }
    return mHasNormalLed || mHasStandbyLed || mHasNetworkLed;
}

public static int setNormalLedOn(boolean on) {
    if (!ServiceReady || !mHasNormalLed) {
        //if (DEBUG) Log.d(TAG, "GpioService is not exists or normal led is not exist
s");
        return -1;
    }
    String dataPath = NORMAL_LED_PATH + LIGHT_NAME;
    return nativeWriteGpio(dataPath, on ? "1" : "0");
}

public static int setStandbyLedOn(boolean on) {
    Log.d(TAG, "setStandbyLedOn go!");
    if (!ServiceReady || !mHasStandbyLed) {
        //if (DEBUG) Log.d(TAG, "GpioService is not exists or standby led is not exi
sts");
        Log.d(TAG, "ServiceReady: "+ServiceReady+",mHasStandbyLed: "+mHasSt
andbyLed);
        return -1;
    }
    Log.d(TAG, "setStandbyLedOn 2");
    String dataPath = STANDBY_LED_PATH + LIGHT_NAME;
    Log.d(TAG, "setStandbyLedOn 3");

    return nativeWriteGpio(dataPath, on ? "1" : "0");
}

public static int setNetworkLedOn(boolean on) {
    if (!ServiceReady || !mHasNetworkLed) {

```



```
//if (DEBUG) Log.d(TAG, "GpioService is not exists or network led is not exists");
return -1;
}
String dataPath = NETWORK_LED_PATH + LIGHT_NAME;
return nativeWriteGpio(dataPath, on ? "1" : "0");
}

private static String composePinPath(char group, int num) {
    String numstr;
    String groupstr;

    groupstr = String.valueOf(group).toUpperCase();
    numstr = Integer.toString(num);
    return PATHSTR.concat(groupstr).concat(numstr);
}
}
```



## 8. How to add your app

The porting of Android applications is usually divided into two phases, the development and debugging phase and the production deployment phase. In the development and debugging phase, we can use the SDK built by MYIR to cross-compile our written application and then remotely copy it to the target host for testing. In the production deployment phase, we need to write the recipe file for the application and use the build production image to build the application.

### 8.1. Pre-installed non-uninstallable APK

**Note:** The name of the apk should not contain special characters such as Chinese characters and spaces.

Create a directory in the *vendor/aw/public/package/apk* directory to store the corresponding APK. place the apk in this directory. Create an Android.mk file in that directory and edit the:

```
# Example
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)

LOCAL_MODULE := APK_MODULE_NAME
LOCAL_MODULE_CLASS := APPS
LOCAL_MODULE_TAGS := optional
LOCAL_MODULE_SUFFIX := $(COMMON_ANDROID_PACKAGE_SUFFIX)
LOCAL_CERTIFICATE := PRESIGNED
LOCAL_SRC_FILES := $(LOCAL_MODULE).apk

include $(BUILD_PREBUILT)
```

In the program mk file (device/softwinner/saturn/t527-demo-car/device-common.mk)

PRODUCT\_PACKAGES:

```
PRODUCT_PACKAGES += APK_MODULE_NAME
```

### 8.2. Preinstalled uninstallable apk



Create a directory in the *vendor/semidrive/modules* directory to store the corresponding APK. place the apk in this directory. Create an Android.mk file in that directory and edit the:

```
LOCAL_PATH := $(call my-dir)

LOCAL_MODULE := Test
LOCAL_SRC_FILES := $(LOCAL_MODULE).apk

$(shell rm -rf $(TARGET_OUT_VENDOR)/preinstall/$(LOCAL_MODULE))
$(shell mkdir -p $(TARGET_OUT_VENDOR)/preinstall/$(LOCAL_MODULE))
$(shell cp $(LOCAL_PATH)/$(LOCAL_SRC_FILES) $(TARGET_OUT_VENDOR)/preinstall/$(LOCAL_MODULE))
```





## 9. References

- **U-boot**  
<https://www.denx.de/wiki/U-Boot>
- **Linux kernel**  
<https://www.kernel.org/>
- **Android**  
<https://developer.android.com/>
- **Android source code**  
<http://aospref.com>
- **Android open source project**  
<https://github.com/aosp-mirror/>



# 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 Intelligent Park, Bantian, Longgang District.

Support Email: [support@myirtech.com](mailto:support@myirtech.com)

Sales Email: [sales@myirtech.com](mailto:sales@myirtech.com)

Phone: +86-755-22984836

Fax: +86-755-25532724

Website: [www.myirtech.com](http://www.myirtech.com)

