

# MYD-YT113X Linux Software Development Guide



File status: [ ] Draft [√] Release	<b>FILE ID:</b>	MYIR-MYD-YT113X-SW-DG-EN-L5.4.61
	<b>VERSION:</b>	V2.0[Doc]
	<b>AUTHOR:</b>	MSW0307
	<b>RELEASE:</b>	2024-08-09
	<b>UPDATED:</b>	2024-07-25

# Revision History

VERSION	AUTHOR	PARTICIPANT	DATE	DESCRIPTION
V1.0[Doc]	MSW0202	MSW0019	2023-05-01	Initial version Applicable to the Longan version
V1.1[Doc]	MSW0202	MSW0019	2023-08-30	Added MYD-YT113-I model suitable for Longan version
V2.0[Doc]	MSW0307	MSW0041	2024-08-09	Suitable for Tina 5.0

# CONTENT

Revision History .....	- 2 -
CONTENT .....	- 3 -
1. Overview .....	- 5 -
1.1. Software Resources .....	- 6 -
1.2. Document Resources .....	- 6 -
2. Development environment preparation .....	- 7 -
2.1. Development Host Environment .....	- 7 -
2.2. Software Environment .....	- 9 -
3. Understand the structure of Linux SDK .....	- 12 -
3.1. brief introduction .....	- 12 -
3.2. Get source code .....	- 13 -
3.3. SDK Catalog Introduction .....	- 14 -
4. Building Linux image .....	- 20 -
4.1. Select Configuration .....	- 20 -
4.2. Compile image .....	- 22 -
4.3. Pack Images .....	- 25 -
5. How to burn system image .....	- 26 -
5.1. Making an SD card image .....	- 26 -
6. How to modify the board level support package .....	- 34 -
6.1. Introduction to Board Level Support Packages .....	- 34 -
6.2. Onboard uboot compilation and updates .....	- 35 -
6.3. Onboard Kernel Compilation and Updates .....	- 36 -
6.4. Onboard buildroot compilation and updates .....	- 38 -
7. How to adapt to your hardware platform .....	- 41 -
7.1. How to Create Your Device Tree .....	- 41 -
7.2. Configuring CPU Function Pins Based on Your Hardware .....	- 46 -



7.3. How to use self configured pins.....	- 52 -
8. Application development and startup .....	- 55 -
8.1. Application Auto Startup Configuration .....	- 55 -
8.2. Modification of startup logo .....	- 56 -
9. References .....	- 57 -
Appendix A.....	- 58 -



# 1. Overview

The commonly used methods for system construction and customized development include Buildroot, Yocto, OpenEmbedded, and so on. The buildroot project uses a more lightweight and customized approach to build Linux systems suitable for embedded products. It is not only a tool for creating file systems, but also provides a complete set of Linux based development and maintenance work, enabling underlying embedded developers and upper level application developers to develop under a unified framework, solving the fragmented and unmanaged development form under traditional development methods.

This article mainly introduces the complete process of customizing a complete embedded Linux system based on the Allwinner T113 processor SDK project and MYiR core board, including the preparation of the development environment, code acquisition, and how to port Bootloader and Kernel, customize a root file system rootfs suitable for one's own application needs.

First, we will explain how to build a system image suitable for the MYD-YT113X development board based on the source code provided by MYiR, and how to update and download the built image to the development platform. Secondly, for users who are customizing projects based on the MYC-YT113X core board, we will focus on how to use this SDK to port to the user's hardware platform, highlighting key methods and challenges. Finally, we will also provide some practical examples of driver porting and rootfs customization, enabling users to quickly develop system images that match their hardware.

Please note that this document does not include an introduction to the Buildroot project or basic knowledge related to the Linux system. It is suitable for embedded Linux system developers and embedded Linux BSP developers with some development experience. For specific features that users may need during secondary development, we also provide detailed application notes for developers' reference. For specific information, refer to the document list in Table 2-4 of the "*MYD-YT113X SDK Release Note*".



## 1.1. Software Resources

MYD-YT113X is equipped with an operating system based on the Linux 5.4.61 kernel, providing abundant system resources and other software resources. The development board comes with a cross compilation toolchain, U-boot source code, Linux kernel and driver module source code packages required for embedded Linux system development, as well as various development and debugging tools, application development routines, etc. suitable for Windows desktop environment and PC Linux system. Please refer to Chapter 2 Software Information in the "*MYD-YT113X SDK Release Note*" for specific software information included.

## 1.2. Document Resources

According to the different purposes of users using the development board. We will provide customers with a complete SDK package (Software Development Kit), which includes release notes, evaluation guides, development guides, application notes, commonly used Q&A, and other types of documents and manuals. The specific document list can be found in Table 2-4 of the "*MYD-YT113X SDK Release Note*".

## 2. Development environment preparation

This chapter mainly introduces some software and hardware environments required for the development process of the Tina 5.0 system based on MYD-YT113X development board, including necessary development host environment, essential software tools, code and data acquisition, etc. The specific preparation work will be described in detail below.

### 2.1. Development Host Environment

How to build the development environment applicable to Allwinner T1 series processor platform. By reading this chapter, you will understand the installation and use of related hardware tools, software development and debugging tools. And you will be able to quickly set up the related development environment to prepare for the later development and debugging. Allwinner T1 series processors are SMP multi-core architecture processors, 2 ARM Cortex A7, can run embedded Linux system, using common embedded Linux system development tools can be.

- **Host Hardware**

The construction of the entire SDK package project has high requirements for the development host, requiring the processor to have a dual core or higher CPU, 4GB or more memory, 100GB hard drive or higher configuration. It can be a PC or server with Linux system installed, a virtual machine running Linux system, WSL2 under Windows system, etc.

- **Host Operating System**

Build buildroot project host operating system can have a variety of options, generally choose to install Fedora, openSUSE, Debian, Ubuntu, RHEL or CentOS and other Linux distributions such as the local host for the build, here is recommended for the desktop version of the Ubuntu20.04 64bit system, in the system Here we recommend Ubuntu 20.04 64bit desktop system, in which the

system compilation is more stable, and the subsequent development is also introduced as an example of this system.

- **Install necessary tools**

Install necessary development dependency packages on the host first

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

```
PC$: sudo apt install -y git gnupg flex bison gperf build-essential zip curl  
libc6-dev libncurses5-dev:i386 x11proto-core-dev libx11-dev:i386  
libreadline6-dev:i386 libgl1-mesa-glx:i386 libgl1-mesa-dev g++-multilib tofrodos  
python markdown libxml2-utils xsltproc zlib1g-dev:i386 gawk texinfo gettext
```

```
build-essential gcc libncurses5-dev bison flex zlib1g-dev gettext libssl-dev  
autoconf libtool linux-libc-dev:i386 wget patch dos2unix tree u-boot-tools
```

Other non mandatory configuration packages

```
PC$: sudo dpkg-reconfigure dash #select no
```

```
PC$: sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1 /usr/lib/i386-linux-gnu/libGL.so
```

```
PC$: sudo apt-get install zlib1g-dev # Install when libz.so is missing
```

```
PC$: sudo apt-get install uboot-mkimage # Install or install u-boot tools when missing mkimage
```



## 2.2. Software Environment

This chapter mainly introduces some software and hardware environments required for the development process based on MYD-T113X development board, including necessary development host environment, essential software tools, code and data acquisition, etc. The specific preparation work will be described in detail below.

### 2.2.1. Installation of Cross-Compilation Toolchain

The SDK provided by MYiR not only includes various source codes, but also provides necessary cross compilation toolchains. Users can directly use cross compilation toolchains to establish an independent development environment, which can be directly used to compile applications, etc. The specific process will be detailed in the following chapters. Here are the installation steps for the SDK, as follows:

**Note:** During the process of building this system image using SDK, there is no need to install cross compilation toolchain.

- **Copy SDK to Linux directory and unzip**

Copy the SDK compressed package to the user's working directory under Ubuntu, such as \$HOME/T113, which is defined according to your actual situation. Then unzip the file to obtain the SDK source code file, as follows:

```
PC$ cd $HOME/T113
PC$ tar -zxf MYD-YT113X-Distribution-L5.4.61-V2.0.0.tar.gz
```

- **Create a new toolchain directory**

According to user requirements, unzip the toolchain to the specified directory. The "*03-Tools/Complie Toolchain*" directory in the software documentation directory of the download contains the compilation chain file "*gcc-linaro-5.3.1-2016.05-x86\_64\_arm-linux-gnueabi.tar.gz*". Copy the file to Ubuntu and decompress it. Users can choose the directory to decompress themselves. Here, the "*~/opt*" directory is taken as an example:

- **Decompress the compilation chain**

Extract to the "*~/opt*" directory on the host:

```
PC$ tar -xf gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabi.tar.gz -C ~/opt
```

Note: It is recommended that users unzip to their home directory instead of the root directory to avoid permission issues.

- **Install and test cross compilation chain**

For the convenience of setting up the compilation chain, a script T113-env.sh can be created in advance, as follows:

Users can choose their own directory to place scripts in

```
PC$ vi ~/opt/T113-env.sh
```

Then copy the following script to T113-env.sh, and make sure the red section below matches the directory of the decompressed compilation chain.

```
#!/bin/sh
export PATH=~/opt/gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabi/bin/:${P
ATH}
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabi-
export PREFIX=arm-linux-gnueabi-
export CC=arm-linux-gnueabi-gcc
export AR=arm-linux-gnueabi-ar
export AS=arm-linux-gnueabi-as
export LD=arm-linux-gnueabi-ld
export NM=arm-linux-gnueabi-nm
export STRIP=arm-linux-gnueabi-strip
export OBJCOPY=arm-linux-gnueabi-objcopy
export OBJDUMP=arm-linux-gnueabi-objdump
```

Set environment variables and test if the installation is complete.

```
PC$ arm-linux-gnueabi-gcc -v
Using built-in specs.
COLLECT_GCC=arm-linux-gnueabi-gcc
COLLECT_LTO_WRAPPER=~/opt/gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabi/bin/./libexec/gcc/arm-linux-gnueabi/5.3.1/lto-wrapper
Target: arm-linux-gnueabi
Configured with: /home/tcwg-buildslave/workspace/tcwg-make-release/label/docker-trusty-amd64-tcwg/target/arm-linux-gnueabi/snapshots/gcc-linaro-5.3-2016.
```

```
05/configure SHELL=/bin/bash --with-mpc=/home/tcwg-buildslave/workspace/tcwg-make-release/label/docker-trusty-amd64-tcwg/target/arm-linux-gnueabi/_build/builds/destdir/x86_64-unknown-linux-gnu --with-mpfr=/home/tcwg-buildslave/workspace/tcwg-make-release/label/docker-trusty-amd64-tcwg/target/arm-linux-gnueabi/_build/builds/destdir/x86_64-unknown-linux-gnu --with-gmp=/home/tcwg-buildslave/workspace/tcwg-make-release/label/docker-trusty-amd64-tcwg/target/arm-linux-gnueabi/_build/builds/destdir/x86_64-unknown-linux-gnu --with-gnu-as --with-gnu-ld --disable-libstdcxx-pch --disable-libmudflap --with-cloog=no --with-ppl=no --with-isl=no --disable-nls --enable-c99 --with-tune=cortex-a9 --with-arch=armv7-a --with-fpu=vfpv3-d16 --with-float=softfp --with-mode=thumb --disable-multilib --enable-multiarch --with-build-sysroot=/home/tcwg-buildslave/workspace/tcwg-make-release/label/docker-trusty-amd64-tcwg/target/arm-linux-gnueabi/_build/sysroots/arm-linux-gnueabi --enable-lto --enable-linker-build-id --enable-long-long --enable-shared --with-sysroot=/home/tcwg-buildslave/workspace/tcwg-make-release/label/docker-trusty-amd64-tcwg/target/arm-linux-gnueabi/_build/builds/destdir/x86_64-unknown-linux-gnu/arm-linux-gnueabi/libc --enable-languages=c,c++,fortran,lto --enable-checking=release --disable-bootstrap --build=x86_64-unknown-linux-gnu --host=x86_64-unknown-linux-gnu --target=arm-linux-gnueabi --prefix=/home/tcwg-buildslave/workspace/tcwg-make-release/label/docker-trusty-amd64-tcwg/target/arm-linux-gnueabi/_build/builds/destdir/x86_64-unknown-linux-gnu
Thread model: posix
gcc version 5.3.1 20160412 (Linaro GCC 5.3-2016.05)
```

You can see that the last line printed matches the installed version, which proves that the installation was successful.

## 3. Understand the structure of Linux SDK

### 3.1. brief introduction

Tina Linux Unified Platform, also known as Tina 5.0. Intended to integrate Longan and old versions of Tina, creating a standardized, open, efficient, and reusable Linux development platform for unified use. It integrates BSP, build system, independent IP, and testing, and can serve as both a BSP development and IP verification platform, as well as a mass-produced embedded Linux system.

The functions of the Linux SDK include the following four parts:

- BSP development, including bootloader, uboot, and kernel.
- Linux file system development, including mass-produced embedded Linux systems.
- An IP verification and publishing platform, along with instructions for using the IP and a demo program for system integration, to facilitate quick use by third parties.
- Testing, including board level testing and system testing.

The "*04-sources*" directory in the CD image provided by MYiR provides Linux SDK files and data suitable for the MYD-YT113X development board, helping developers build different types of Linux system images that can run on the MYD-YT113X development board, as shown in the table below:

Table 3-1. MYD-YT113X Image File Description

Image file name	description
myir-image-yt113i-core.img	Build an image without QT graphical interface using buildroot.
myir-image-yt113i-full.img	Build an image with QT graphical interface using buildroot.(default to 7-inch LVDS display)

The following to build "*myir-image-yt113i-full.img*" image as an example to introduce the specific development process for the subsequent customization of their own system image to lay the foundation for the user to select the

corresponding operation according to their own development board model, the basic operation with the production of "*myir-image-yt113i-full.img*" image consistent, if there is a difference in the operation of this paper will be specifically pointed out. The basic operation is the same as making "*myir-image-yt113i-full.img*" image, if there is any difference in the operation, this article will point out specifically, please watch the operation of this document carefully, so as not to operate improperly resulting in the failure of image making.

## 3.2. Get source code

Users can directly obtain compressed files from the "*04-Sources*" directory of the MYiR CD image.

### 3.2.1. Obtaining the Source Code Package from Release Materials

The compressed source package is located at MYiR Development Kit Profile "*04-Sources/MYD-YT113X-Distribution-L5.4.61-V2.0.0.tar.gz*". Copy the compressed package to a user-specified directory, such as the "*\$HOME/T113*" directory, which will serve as the top-level directory for subsequent builds, and unzip the package as follows to appear All contents of the SDK:

```
PC$ $HOME$ tar -xzf MYD-YT113X-Distribution-L5.4.61-V2.0.0.tar.gz
PC$ $HOME/T113$ tree -L 1
```

```
.
├── brandy
├── build
├── buildroot
├── build.sh -> build/top_build.sh
├── device
├── kernel
├── openwrt
├── out
├── platform
├── prebuilt
├── rtos
└── tools
```

11 directories, 1 file

### 3.3. SDK Catalog Introduction

The SDK mainly consists of branding, buildroot, kernel, and platform.

- Brand includes u-boot-2018
- Buildroot is responsible for generating ARM toolchains, application software packages, and Linux root file systems
- The kernel is the Linux kernel
- Platform is a platform related library and SDK application

#### 3.3.1. Buildroot Directory

Buildroot is a set of Makefiles and patches that simplify and automate the construction of a complete, bootable Linux environment (including bootloader, Linux kernel, and file system containing various APP) for embedded systems. Buildroot runs on the Linux platform and can use cross compilation tools to build embedded Linux platforms for multiple target boards. Buildroot can automatically build the required cross compilation toolchain, create a root file system, compile Linux kernel images, and generate bootloaders for the target embedded system, or it can perform any independent combination of these steps. For example, the installed cross compilation toolchain can be used separately, while Buildroot only creates the root file system.

The directory structure is as follows:

```
PC$ $HOME/T113/buildroot/buildroot-201902$ tree -L 1
.
├── arch
├── board
├── boot
├── build.sh
├── CHANGES
└── Config.in
```

```
├─ Config.in.legacy
├─ configs
├─ COPYING
├─ DEVELOPERS
├─ dl
├─ docs
├─ fs
├─ linux
├─ Makefile
├─ Makefile.legacy
├─ package
├─ README
├─ scripts
├─ support
├─ system
├─ toolchain
└─ utils
```

The configs directory stores pre-defined configuration files, as shown in the table below:

Table 3-2. MYD-YT113X buildroot configuration file instructions

model	configuration file	describe
MYD-YT113-I	myd_yt113i_br_core_defconfig	emmc core system buildroot configuration
	myd_yt113i_br_full_defconfig	EMMC full system build root configuration

These configuration files have already defined the configuration of different images for different models. The dl directory stores downloaded software packages, and the scripts directory stores scripts compiled by buildroot, such as mkcmd. sh, mkcommon. sh, mkrule, and mksetup. sh. The target directory stores some rule files used to generate the root file system, which is very important for the integration of code and tools. The most important thing for us is the package directory, which contains nearly 3000 software package generation rules. We can add our own software packages or middleware to it.

For more information about buildroot, you can visit the official website of buildroot <http://buildroot.uclibc.org/> obtain.

### 3.3.2. Kernel Directory

Linux kernel source code directory. The currently used kernel version is Linux 5.4.61. Except for the modules directory, the above directory structure is consistent with the standard Linux kernel. The modules directory is where we store external modules that are not integrated with the kernel's menuconfig.

```
PC$ $HOME/T113/kernel$ tree -L 1
```

```
.
└── linux-5.4
```

Note: Since the directory allwinner has been modified, it does not support the execution of make in the current directory for individual compilation, but can be executed in the top-level directory of the source code. /build.sh kernel in the top directory of the source code for individual compilation, the specific steps are explained in subsequent chapters.

### 3.3.3. Brandy Directory

There is a brandy2.0 version in the brandy directory, and currently T113X is using brandy2.0 version. Its directory structure is:

```
PC$ $HOME/T113/brandy$ tree -L 1
```

```
.
└── brandy-2.0
```

Introduction to main directory components:

- brandy-2.0: Uboot source code

### 3.3.4. Platform Directory

Platform private software package directory.

```
PC$ $HOME/T113/platform$ tree -L 1
```

```
.
├── allwinner
├── Makefile -> /home/T113I/build/Makefile
└── thirdparty
```

- Allwinner: Allwinner provides software private package source code
- Third party: directory for storing QT and lvgl source code



### 3.3.5. Tools Directory

```
PC$ $HOME/T113/tools$ tree -L 1
```

```
.
├── build
├── codecheck
├── pack
├── tools_dev
└── tools_win
```

- Codecheck: Code checking tool
- Tools\_win: Windows software tool

### 3.3.6. Build Directory

This directory stores compiled and packaged scripts.

```
PC$ $HOME/T113/build$ tree -L 1
```

```
.
├── bin
├── bsp.sh
├── buildbase.sh
├── createkeys
├── disclaimer
├── envsetup.sh
├── extract_symbols
├── getvmlinux.sh
├── hook
├── Makefile
├── mkcmd.sh
├── mkcommon.sh
├── mkernel.sh
├── pack
├── parser.sh
├── quick.sh
├── scripts
└── shflags
```

- 17 -

```
├─ swpsdc.sh
├─ swupdate
└─ top_build.sh
```

- bin: This directory holds tools for making filesystems.
- bsp.sh: Scripts that are compiled in conjunction with independent repositories.
- Create keys: Tool for creating security scheme keys
- envsetup.sh: script to configure environment variables
- mkcmd.sh: main compilation scripts
- mkcommon.sh: Compilation scripts
- mkernel.sh: kernel compilation related scripts
- pack: Packaging scripts
- top\_build.sh : top-level build scripts

### 3.3.7. The device directory

This is the directory where you can store your configurations.

```
PC$ $HOME/T113/device/config/chips/t113_i$ tree -L 4
```

```
.
├─ config
│   └─ chips
│       └─ t113_i
│           ├── bin
│           ├── boot-resource
│           ├── configs
│           └─ tools
```

- Chips: Directory for storing board level configurations
- T113i: t113i related configuration directory
- Boot resource: Boot resource file, such as bootlogo Startup logo
- Configs: Scheme configuration directory

### 3.3.8. Prebuilt Directory

This directory is used to store various compilation environment libraries.

```
PC$ $HOME/T113/prebuilt$ tree -L 1
```

```
.  
├── hostbuilt  
├── kernelbuilt  
└── rootfsbuilt
```

- Hostbuilt: directory where precompiled commands are stored
- Kernelbuilt: a directory that stores the compiled kernel toolchain
- Rootfsbuilt: The directory where the file system compilation chain is stored

### 3.3.9. Rtos Directory

This directory is where the rtos source code is stored

```
PC$ $HOME/T113/rtos$ tree -L 1  
.  
├── board  
├── envsetup.sh -> tools/scripts/source_envsetup.sh  
├── lichee  
└── tools
```

- board: Directory for storing rtos core configuration
- envsetup.sh: Script for configuring the compilation environment
- lichee: directory for storing rtos source code
- tools: Directory for storing rtos compilation scripts

## 4. Building Linux image

Before building the image, make sure you have installed the necessary packages in section 2.1 to avoid errors when building the image. The following is an example of building a "*myir-image-yt113i-full image*".

### 4.1. Select Configuration

```
PC$ $HOME/T113$ ./build.sh config
=====ACTION List: mk_config ;=====
options :
All available platform:
    0. android
    1. linux
Choice [linux]: 1
All available linux_dev:
    0. bsp
    1. buildroot
    2. openwrt
Choice [buildroot]: 1
All available ic:
    0. t113_i
Choice [t113_i]: 0
All available board:
    0. myir-image-yt113i-core
    1. myir-image-yt113i-full
Choice [myir-image-yt113i-full]: 1
All available flash:
    0. default
    1. nor
Choice [default]: 0
```

If it is completed after execution/ After "*building.sh*" config, the following error message appears.

File "<string>", line 1

```
import os.path; print os.path.relpath('/home/sur/T113-core/kernel/linux-5.4/
arch/arm/configs/sun8iw20p1smp_t113_auto_defconfig', '/home/sur/T113-core/k
ernel/linux-5.4/arch/arm/configs')
```

^

**SyntaxError: invalid syntax**

**ERROR: Can't find kernel defconfig!**

This is caused by an incorrect version of Python, requiring a version of Python 2. Please check if Python 2 is already installed in the current development environment. If not installed, please refer to section 2.1 or execute the following command directly:

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

Check if the current version is Python 2

```
PC$ python --version
```

```
Python 2.7.18
```

If Python 2 is installed but the version is still different, it means that multiple Python versions are installed in the host environment. At this time, you need to switch between Python versions by executing the following command:

**Note: Switch according to your actual host environment**

```
sudo update-alternatives --install /usr/bin/python python /usr/bin/python2.7 2
```

```
sudo update-alternatives --install /usr/bin/python python /usr/bin/python3.6 1
```

```
sudo update-alternatives --config python
```

There are 2 candidate options available to replace Python (providing/usr/bin/Python).

select	route	priority	state
0	/usr/bin/python2.7	2	Automatic mode
* 1	/usr/bin/python2.7	2	Manual mode
2	/usr/bin/python3.6	1	Manual mode

To maintain the current value [\*], press<Enter>or type the selected number:

1

## 4.2. Compile image

After selecting the configuration, execute the following command to start compiling the system, which takes a relatively long time.

```
PC$ $HOME/T113$ ./build.sh
=====ACTION List: build_linuxdev;=====
options :
INFO: -----
INFO: build linuxdev ...
INFO: chip: sun8iw20p1
INFO: platform: linux
INFO: kernel: linux-5.4
INFO: board: myir-image-yt113i-fullINFO: output: /home/szy/T113I/T113/out/t1
13_i/myir-image-yt113i-full/buildroot
INFO: -----
INFO: don't build dtbo ...
INFO: build arisc
find: '/home/szy/T113I/T113/brandy/brandy-2.0/spl': No such file or directory
find: '/home/szy/T113I/T113/brandy/brandy-2.0/u-boot-bsp': No such file or di
rectory
find: '/home/szy/T113I/T113/brandy/dramlib': No such file or directory
INFO: build_bootloader: brandy_path=/home/szy/T113I/T113/brandy/brandy-2.0
INFO: skip build brandy.
INFO: build kernel ...
```

When performing this step, you may encounter the following errors

```
gdbusconnection.c: In function 'check_initialized':
gdbusconnection.c:567:8: warning: unused variable 'flags' [-Wunused-variable]
 567 |     gint flags = g_atomic_int_get (&connection->atomic_flags);
      |         ^~~~~
gdbusmessage.c: In function 'parse_value_from_blob':
gdbusmessage.c:1712:29: warning: variable 'item' set but not used [-Wunused-but-set-variable]
 1712 |         GVariant *item;
      |         ^~~~~
gdbusmessage.c: In function 'append_value_to_blob':
gdbusmessage.c:2326:24: warning: unused variable 'end' [-Wunused-variable]
 2326 |         const gchar *end;
      |         ^~~~~
gdbusauth.c: In function '_g_dbus_auth_run_server':
gdbusauth.c:1302:11: error: '%s' directive argument is null [-Werror=format-overflow=]
 1302 |         debug_print ("SERVER: WaitingForBegin, read '%s'", line);
      |         ^~~~~~
CC      libgio_2.0_la-gdbusinterface.lo
cc1: some warnings being treated as errors
Makefile:3633: recipe for target 'libgio_2.0_la-gdbusauth.lo' failed
make[5]: *** [libgio_2.0_la-gdbusauth.lo] Error 1
make[5]: *** Waiting for unfinished jobs....
gdbusmessage.c: In function 'g_dbus_message_to_blob':
gdbusmessage.c:2702:30: error: '%s' directive argument is null [-Werror=format-overflow=]
 2702 |         tupled_signature_str = g_strdup_printf ("%s", signature_str);
      |         ^~~~~~
gdbusintrospection.c: In function 'g_dbus_interface_info_generate_xml':
gdbusintrospection.c:751:3: warning: 'access_string' may be used uninitialized in this function [-Wmaybe-uninitialized]
 751 |     g_string_append_printf (string_builder, "%s<property type=\"%s\" name=\"%s\" access=\"%s\"\"",
      |     ^~~~~~
```

Figure 4-1. Compiling GDBus failed 1

Modify step 1: (Note that the full name of the path may not be the same, find "*gdbusauth.c*" according to your actual path).

```
PC$ $HOME/T113$ vim ./out/t113_i/myir-image-yt113i-full/buildroot/buildroot/build/host-libglib2-2.56.3/gio/gdbusauth.c
```

Add this judgment code in the following position:

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

Modify step 2: (Note that the path may be different in full, find "*gdbusmessage.c*" according to your actual path).

```
PC$ $HOME/T113$ vim ./out/t113_i/myir-image-yt113i-full/buildroot/buildroot/build/host-libglib2-2.56.3/gio/gdbusmessage.c
```

Add this judgment code in the following position:

```
if (message->body != NULL)
```

```
{
    gchar *tupled_signature_str;
    if (signature != NULL)
        tupled_signature_str = g_strdup_printf("(%s)", signature_str);
    if (signature == NULL)
```

```
gawk -f ./mkerrnos.awk ./errnos.in >code-to-errno.h
gawk -f ./mkerrcodes1.awk ./errnos.in >mkerrcodes.h
gawk -f ./mkstrtable.awk -v textidx=2 -v nogettext=1 \
./err-sources.h.in >err-sources-sym.h
gawk -f ./mkstrtable.awk -v textidx=2 -v nogettext=1 \
./err-codes.h.in >err-codes-sym.h
gawk -f ./mkstrtable.awk -v textidx=2 -v nogettext=1 \
-v prefix=GPG_ERR -v namespace=errnos_ \
./errnos.in >errnos-sym.h
gawk: ./mkerrnos.awk:86: warning: regexp escape sequence `\'#\' is not a known regexp operator
gawk: ./mkerrcodes1.awk:84: warning: regexp escape sequence `\'#\' is not a known regexp operator
gawk: ./mkstrtable.awk:113: warning: regexp escape sequence `\'#\' is not a known regexp operator
/home/cat/T113/autot113-linux/out/t113/evb1_auto/longan/buildroot/host/bin/arm-linux-gnueabi-cpp -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -P _mk
errcodes.h | grep GPG_ERR_ | \
gawk -f ./mkerrcodes.awk >mkerrcodes.h
gawk: ./mkstrtable.awk:113: warning: regexp escape sequence `\'#\' is not a known regexp operator
/usr/bin/gcc -g -O0 -I. -I. -o mkheader ./mkheader.c
gawk: fatal: cannot use gawk builtin 'namespace' as variable name
make[4]: *** [Makefile:1615: errnos-sym.h] Error 2
make[4]: *** Waiting for unfinished jobs....
gawk: ./mkerrcodes.awk:88: warning: regexp escape sequence `\'#\' is not a known regexp operator
rm _mkerrcodes.h
make[3]: *** [Makefile:508: all-recursive] Error 1
make[2]: *** [Makefile:440: all] Error 2
make[1]: *** [package/pkg-generic.mk:241: /home/cat/T113/autot113-linux/out/t113/evb1_auto/longan/buildroot/build/libgpg-error-1.33/.stamp_built] Error 2
make: *** [Makefile:96: _all] Error 2
make: Leaving directory '/home/cat/T113/autot113-linux/buildroot/buildroot-201902'
ERROR: build buildroot Failed
```

Figure 4-2. Compiling GDBus failed 2

```
PC$ cd HOME/T113/out/t113_i/myir-image-yt113i-full/buildroot/buildroot/build
/libgpg-error-1.33/src/
```

Change "*namespace*" to "*pkg\_namespace*" in "*Makefile*", "*Makefile.am*", "*Makefile.in*", and "*mkstrtable.awk*" in this directory, and then re-execute the compile command(./build.sh).



### 4.3. Pack Images

The final step is to package the image generation system and execute the following command:

```
PC$ $HOME/T113$ ./build.sh pack
=====ACTION List: mk_pack ;=====
options :
INFO: packing firmware ...
INFO: /home/szy/T113I/T113/out/t113_i/common/keys
ERROR: something is incorrect:please <./build.sh config>.
copying tools file
copying configs file
copying product configs file
linux copying boardt&linux_kernel_version configs file
ls: cannot access '/home/szy/T113I/T113/device/config/chips/t113_i/configs/myi
r-image-yt113i-full/linux-5.4/env*': No such file or directory
Use u-boot env file:
Warning: u-boot env file " not exist! use file in default directory other than '
myir-image-yt113i-full' directory
.....
510M    /home/szy/T113I/T113/out/myir-image-yt113i-full.img
```

## 5. How to burn system image

MYiR company designed MYC-YT113X series core board and development board is based on Allwinner's T1 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:

- Create SD card bootloader: suitable for R&D debugging, fast boot scenarios, etc.
- Make SD card burner: suitable for mass production burning eMMC

### 5.1. Making an SD card image

The following steps are made under Windows system.

- **preparation**
  - SD card (not less than 8G)
  - MYD-YT113X Development Board
  - Create image tool PhoenixCard (path:\03-Tools\PhoenixCard)

Table 5-1. list of mirror packages

Image Name	Package Name	Applicable core boards
myir-image-yt113i-core.img	myir-image-yt113i-core.img	MYC-YT113i-4E256D MYC-YT113i-4E512D MYC-YT113i-8E512D MYC-YT113i-8E1D
myir-image-yt113i-full.img	myir-image-yt113i-full.img	MYC-YT113i-4E256D MYC-YT113i-4E512D MYC-YT113i-8E512D MYC-YT113i-8E1D

#### 5.1.1. Making an SD card launcher (using the myrir-image-yt113i-full system as an example)

##### 1). Modify the configuration file to make the SD image

Currently the image generated by the SDK does not support SD card boot, you need to modify the following configuration:

```
PC$ cd $HOME/T113/device/config/chips/t113_i/configs/myir-image-yt113i-full
PC$ $HOME/T113/device/config/chips/t113_i/configs/myir-image-yt113i-full$ vi
m sys_config.fex
.....
;-----
;storage_type    = boot medium, 0-nand, 1-sd, 2-emmc, 3-nor, 4-emmc3, 5-sp
inand -1(default)auto scan
;-----
[target]
storage_type     = 1
burn_key         = 0
;-----

PC$ $HOME/T113/device/config/chips/t113_i/configs/configs/myir-image-yt113i
-full/buildroot$ vim env.cfg

#kernel command arguments
earlycon=uart8250,mmio32,0x02501400
initcall_debug=0
console=ttyS5,115200
nand_root=ubi0_5
mmc_root=/dev/mmcblk1p5
mtd_name=sys
rootfstype=ubifs,rw
init=/init
loglevel=3
.....
```

## 2). SD boot image burning steps

Copy "*PhoenixCard.zip*" from "*03-Tools/PhoenixCard*" directory to any directory in windows, double click "*PhoenixCard.exe*" file in "*PhoenixCard*" directory. Insert the 16GB SD card into the windows USB port via SD card reader, as shown in the picture below, select "*Image*" path; select "*Start up*", click "*Burn*". Click the "*Burn*" button to finish the program automatically.

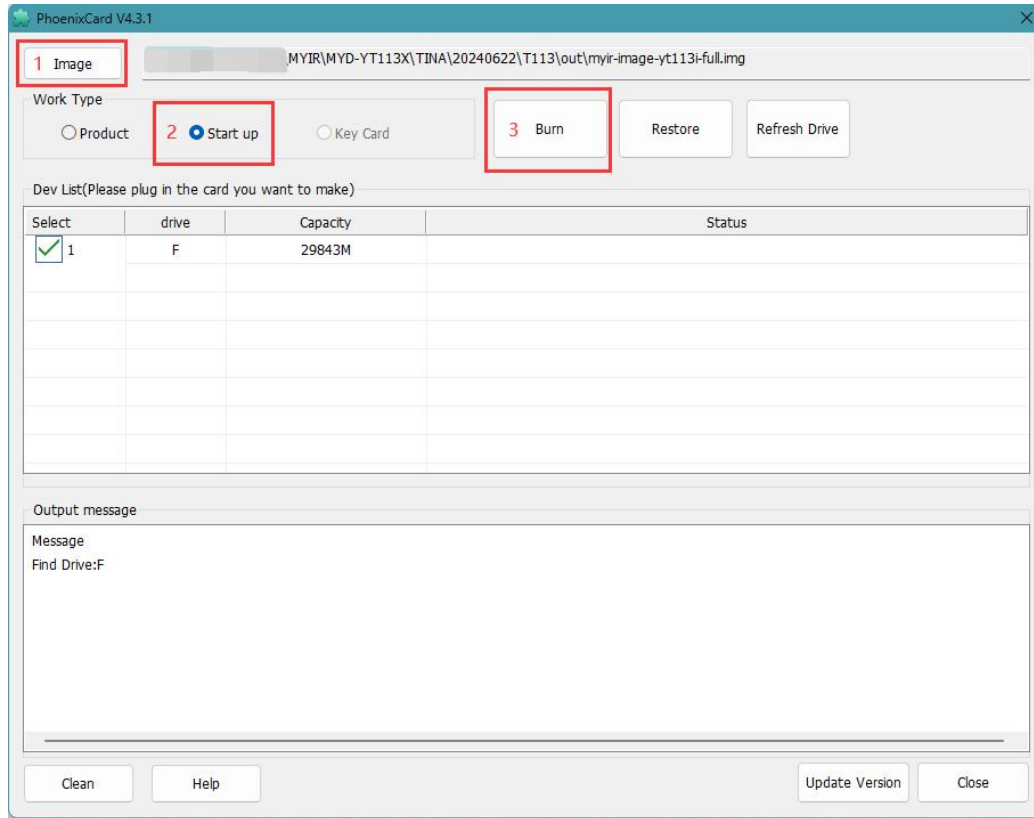


Figure 5-1. Burn writing steps

The figure below (Figure 5-2) shows that the card is being burned and the process is expected to take 3-5 minutes to complete (the time depends on the size of the mirror)

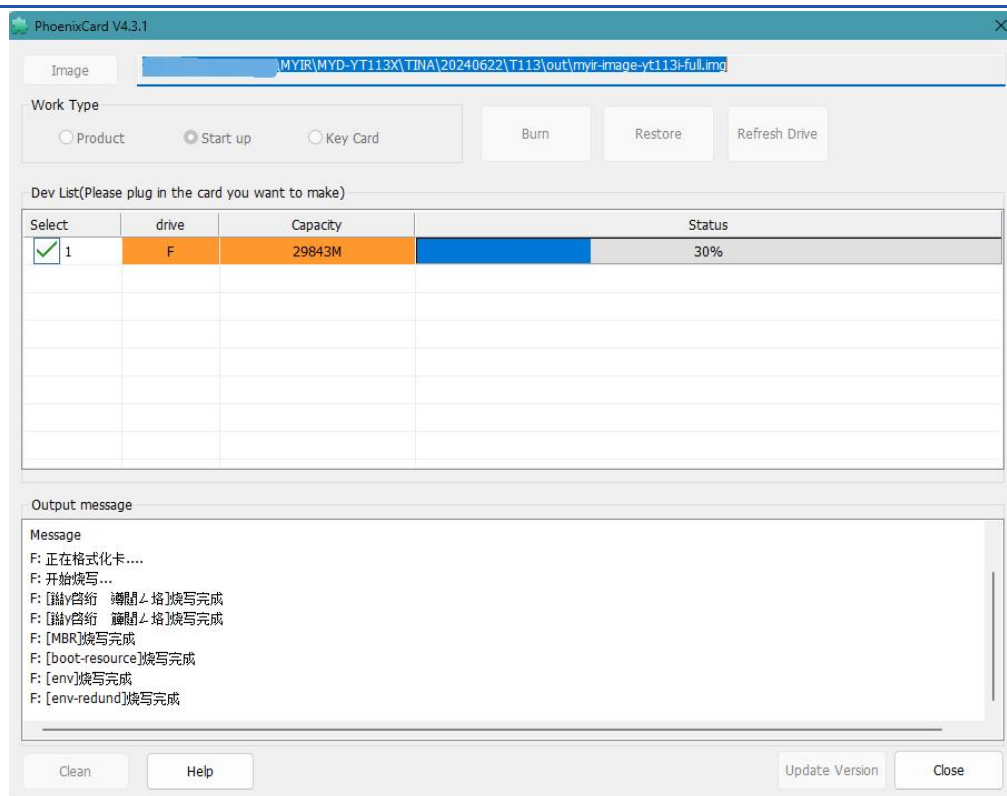


Figure 5-2. Burning process

The following figure (Figure 5-3) shows that the burning of the card is complete, while note that the output message indicates that the burning is complete, at this time the SD card booter is successfully created, insert the SD card into the SD card slot (J5) of the emmc or nand board, and then power on the development board can be started.

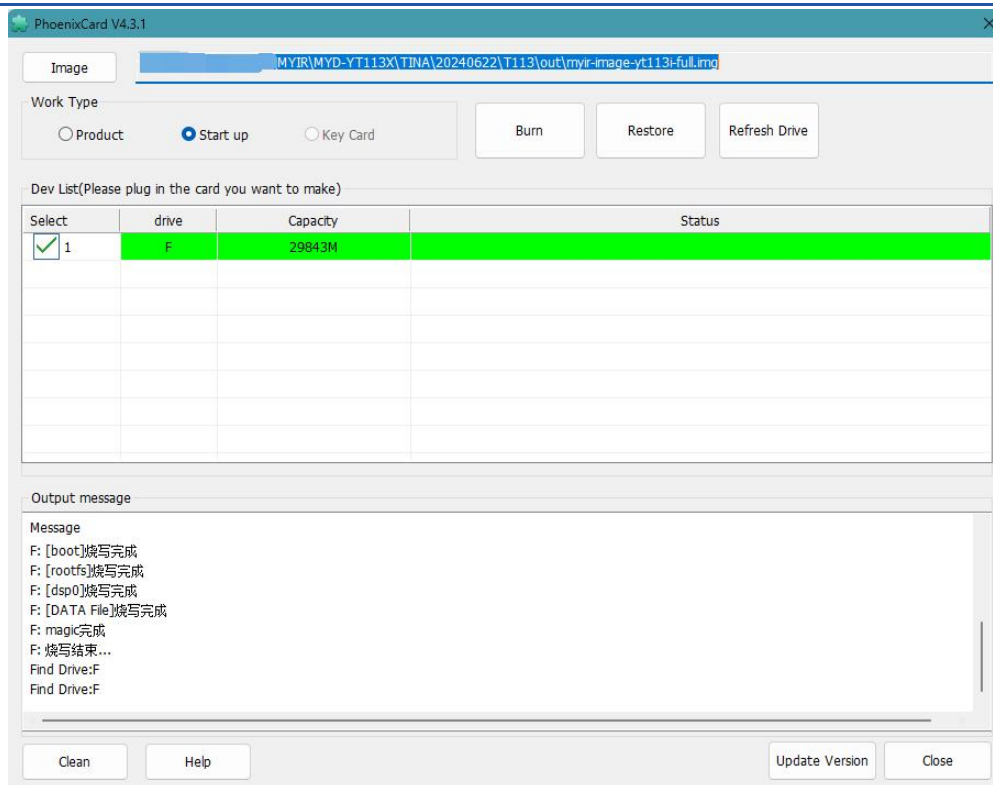


Figure 5-3. Burn successful

## 5.1.2. Making an SD card burner

### 1). Mass production card production process

To meet the needs of production burning, this method is suitable for mass production burning methods. Brush the system that needs to be burned into the onboard eMMC through the system in the SD card. Please follow the following steps to complete the specific production process:

The preparation work is consistent with Chapter 5.1, and the operation steps are similar.

First, open the "PhoenixCard" program under windows (the location of the program is explained in section 5.1 Preparation). Insert a 16GB SD card into the Windows USB interface using an SD card reader, as shown in the following figure, and select the "Image" path; Select "Product" and click the "Burn" button to automatically complete the production.

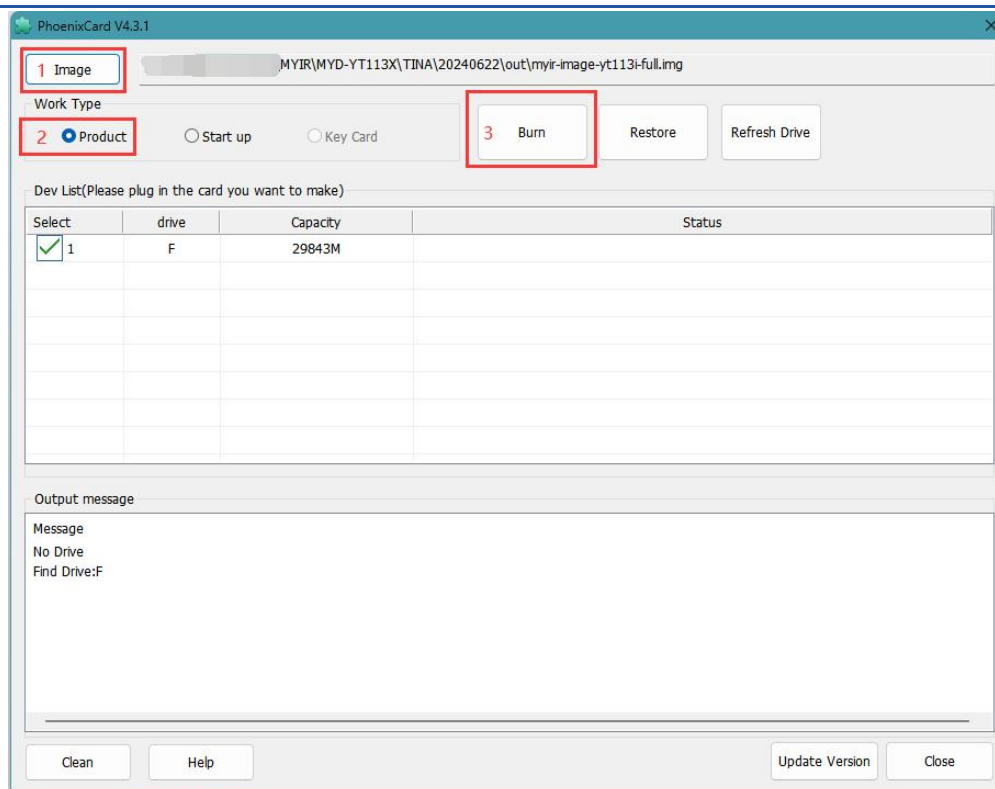


Figure 5-4. Mass Production Card Production

The subsequent method of operation is consistent with section 5.1.1 and will not be repeated here.

**Note:** For specific operations, users can refer to the "*MYD-YT113X Mass Production Guidance Note*".

## 2). Verify eMMC boot

Insert the pre made SD burning card into the SD card slot (J5) of the EMMC board, then power on and start the development board to wait for the burning and printing to be completed. After that, unplug the SD card and restart the development board.

Insert the mass-produced flashcard made above into the SD card slot of the development board (J5), plug in the power, turn the power switch (SW1) to the power on state (the power light is always red), and the system will start automatically burning the image to eMMC after startup. At this time, the user can open the serial port to check the burning information, or observe the blue light status of the development board (flashing after burning is completed) to confirm whether the burning is complete.

Due to the large amount of information burned and printed, only a portion of important data is captured for display, and the final log message indicates that the burning is complete.

```
[05.461]succeeded in download part env-redund
[05.465]begin to download part boot
partdata hi 0x0
partdata lo 0x65d000
sparse: bad magic
[05.917]succeeded in writting part boot
origin_verify value = 1b60c878, active_verify value = 1b60c878
[06.080]succeeded in verify part boot
[06.084]succeeded in download part boot
[06.087]begin to download part rootfs
partdata hi 0x0
partdata lo 0x1b6f2fe8
chunk 0(10577)
chunk 1(10577)
chunk 2(10577)
chunk 3(10577)
chunk 4(10577)
chunk 5(10577)
chunk 6(10577)
chunk 7(10577)
chunk 8(10577)
.....

[93.708]succeeded in downloading boot0
current bitmap buffer size is 0 and new bitmap size is 483.
pitch abs is 21 and glyph rows is 23.
current bitmap buffer size is 483 and new bitmap size is 529.
pitch abs is 23 and glyph rows is 23.
CARD OK
[93.730]sprite success
```



```
sprite_next_work=3
next work 3
SUNXI_UPDATE_NEXT_ACTION_SHUTDOWN
[96.738][mmc]: mmc exit start
[96.756][mmc]: mmc 2 exit ok
*** Flash Success ***
*** Flash Success ***
*** Flash Success ***
```

If the "*Flash Success*" field appears and the blue light on the development board is flashing, it indicates that the burning has been completed. At this time, the development board needs to be powered off, the SD card needs to be unplugged, and finally the system needs to be powered on again. The SD card needs to be unplugged and powered on again, otherwise it will repeatedly enter the burning state.

**Note:** All MYD-YT113X models have the same burning procedure.

## 6. How to modify the board level

### support package

The previous sections have described the complete process of building a system image running on the MYD-YT113X development board based on the Linux SDK project and burning the image to the development board. Since many pins of the MYC-YT113X core board have the feature of multiple function reuse, there will always be some differences between the base board designed based on the MYC-YT113X core board and the MYB-YT113X in the actual project. These differences may be to remove the display, add more GPIO, or the need to add more serial ports, there may be through the SPI, I2C, USB and other extensions of some peripherals, etc.; In addition to hardware differences, there are also some differences in system components, such as focusing on the back-end management applications, 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 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.

#### 6.1. Introduction to Board Level Support Packages

In order to adapt to the new hardware platform of users, it is first necessary to understand what resources MYiR MYD-YT113X development board provides. For specific information, please refer to the "*MYD-YT113X SDK Release Note*". In addition, we have also compiled a list of files that need to be modified in various parts of the key BSP for the convenience of users to search and modify. The specific content is shown in the table below:

Table 6-1. BSP configuration information

Item	Device Tree	Description
U-boot	device/config/chips/t113_i/configs/myir-image-yt113i-xxxx/uboot-board.dts	Board Level Device Tree

- 34 -

	brandy/brandy-2.0/u-boot-2018/configs/sun8iw20p1_auto_t113_i_defconfig	Uboot configuration file
	device/config/chips/t113_i/configs/myir-image-yt113i-xxxx/buildroot/env.cfg	Uboot environment variable configuration
Kernel	device/config/chips/t113_i/configs/myir-image-yt113i-xxxx/linux-5.4/board.dts	Board Level Device Tree
	device/config/chips/t113_i/configs/myir-image-yt113i-xxxx/linux-5.15/myd_yt113i_xxxx_defconfig	kernel configuration file

Note: The "xxxx" in "*myir-image-yt113i-xxxx*" stands for "coer" or "full" system configuration, and users need to modify the corresponding configuration file according to the currently selected system configuration.

## 6.2. Onboard uboot compilation and updates

Uboot is a very feature-rich open source boot loader, including kernel boot, download and update, etc. It is widely used in the embedded field, check the official website for more information <http://www.denx.de/wiki/U-Boot/WebHome>

The T1 platform also uses boot chains as the boot loader, and different boot chain modes correspond to different boot stages.

### 6.2.1. Compile uboot under SDK project

When the user iterative development process to change the U-boot code or modify the uboot device tree, you can use the SDK to build the entire image, you can also compile the uboot source code separately, but not as a separate burning use:

Compile uboot source code:

```
PC$: $HOME/T113$ ./build.sh bootloader
```

At this point, the uboot update and compilation are complete, and then it needs to be packaged into an img file before the update can be burned onto the development board.

```
PC$: $HOME/T113$ ./build.sh pack
```

### 6.2.2. Update uboot

After compiling the uboot source code, burn the generated image to the development board according to the method in section 5.1 and update uboot.

## 6.3. Onboard Kernel Compilation and Updates

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

Meanwhile, T1 also supports Linux kernel, which will be updated stably for a long time. MYD-YT113X is ported with T1 kernel, and the latest support for Linux kernel version 5.4.61.

### 1). Configure kernel (not mandatory)

MYiR has already integrated most of the functions into the kernel, and generally does not require configuration. If you need to add special features, please configure the peripheral driver according to the following method.

- **Modify the kernel**

Execute the following command to open menuconfig configuration

```
PC$ cd $HOME/T113
PC$ $HOME/T113$ ./build.sh menuconfig
```

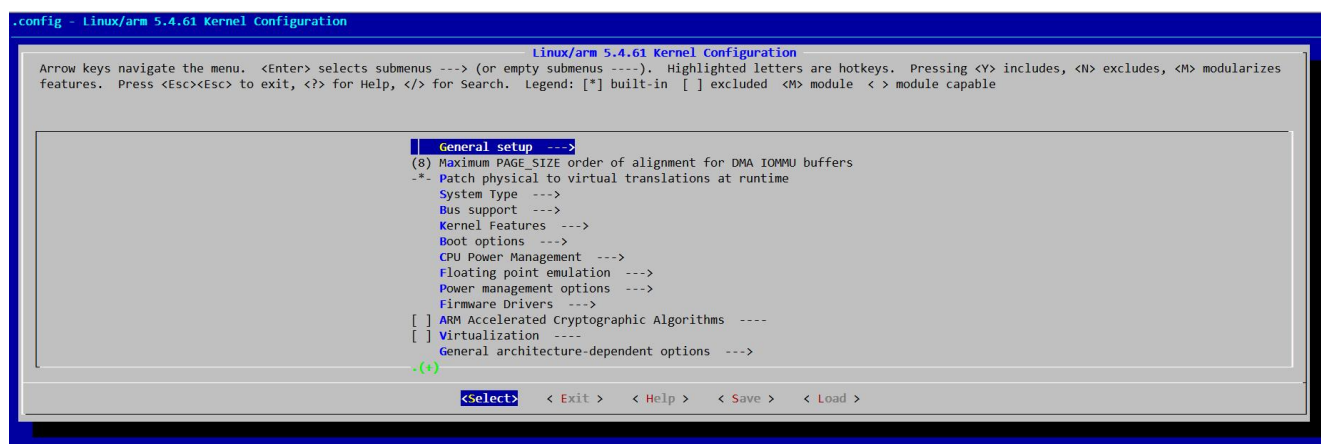


Figure 6-1. Kernel configuration interface

- **Save kernel configuration**

```
PC$ $HOME/T113$ ./build.sh saveconfig
```

### 2). Compile kernel separately

After the user iteratively improves the kernel code during the development process, the SDK can be used to build the entire image. You can also compile the kernel source code separately, but it will not be used for separate burning:

```
PC$: $HOME/T113$ ./build.sh kernel
```

At this point, the kernel update and compilation are complete, and then it needs to be packaged into an img file before it can be burned and updated to the development board

```
PC$: $HOME/T113$ ./build.sh pack
```

### 3). Update device tree

The dts used for board level dts and uboot are located in the \$HOME/T113/device/config/chips/t113\_i/config/myrir-image-yt113i full directory (taking the full image configuration as an example)

```
PC$: cd $HOME/T113/device/config/chips/t113_i/configs/myrir-image-yt113i-full
$ tree -L 1
.
├── bin
├── BoardConfig.mk
├── board.dts -> linux-5.4/board.dts
├── bsp
├── buildroot
├── env.cfg
├── linux-5.4
├── openwrt
├── sys_config.fex
├── sys_partition.fex
├── uboot-board.dts
└── uboot
```

dts used for eMMC board type

After modifying the device tree, return to the top-level directory of the SDK source code and execute the compile and package image command below.

```
PC$: cd $HOME/T113$
```

```
PC$: $HOME/T113$ ./build.sh kernel
```

- 37 -

```
PC$: $HOME/T113$ ./build.sh pack
```

### 6.3.2. Update kernel or device tree

After compiling the kernel or device tree separately, package the image and burn it to the development board according to the method in section 5.1 to update the device tree file.

## 6.4. Onboard buildroot compilation and updates

### 6.4.1. Compile and update buildroot separately from SDK.

#### 1). Configure buildroot

MYiR has integrated most of the tools into buildroot and generally does not require configuration. If you need to add special tools, please configure them in the following way.

- **Modify buildroot**

Execute the following command to open menuconfig configuration.

```
PC$: $HOME/T113$ ./build.sh buildroot_menuconfig
```

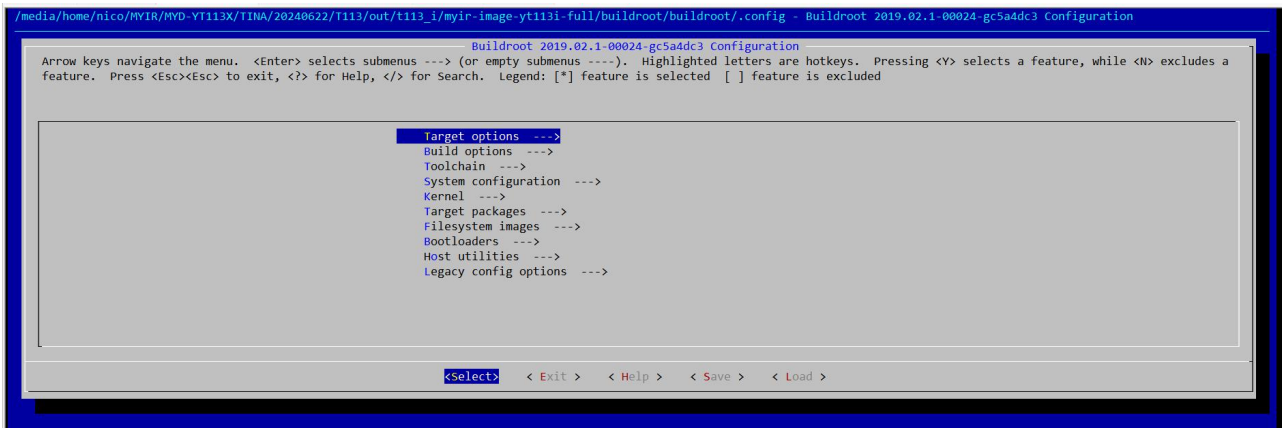


Figure 6-2. Menuconfig configuration

After opening the menuconfig configuration interface, you can modify the configuration of buildroot.

- **Save buildroot configuration**

After you exit the menuconfig configuration interface, execute the following commands to save the operations in menuconfig from the previous step

```
PC$: $HOME/T113$ ./build.sh buildroot_saveconfig
```

- **Compile buildroot separately**

After the user iterates the development process and changes the code for buildroot, they can use the SDK to build the entire image. You can also compile the buildroot source code separately, but it will not be used for separate burning:

```
PC$: $HOME/T113$ ./build.sh
```

At this point, the buildroot update and compilation are complete, and then it needs to be packaged into an img file before burning the update to the development board

```
PC$: $HOME/T113$ ./build.sh pack
```

### 6.4.2. Update buildroot

Buildroot is compiled separately, packaged as an image, and then burned to the development board as described in section 5.1 to update the device tree file.

### 6.4.3. File system customization

Users sometimes need to add their own files to the file system, there is a file system directory under buildroot similar to the corresponding file system directory of the development board, you just need to copy your own files to this directory.

```
PC$: $HOME/T113/buildroot/buildroot-201902/board/myir/t113i-xxxx/rootfs-overlay$ tree -L 1
.
├── etc
├── root
└── usr
```

**Note:** t113i-xxxx, where xxxx represents core or full system configuration, users need to modify the corresponding configuration file based on the currently selected system configuration.

This directory corresponds to the root directory "/" of the development board. Adding files or directories to this directory will also generate corresponding files on the development board.

After adding, go back to the top-level directory and execute  
"./build.sh" , ". /build.sh pack".

```
PC$: $HOME/T113$ ./build.sh
```

```
PC$: $HOME/T113$ ./build.sh pack
```

Then update the file system according to section 6.4.2.



# 7. How to adapt to your hardware platform

In order to adapt to the new hardware platform of users, it is first necessary to understand what resources MYiR MYD-YT113X development board provides. For specific information, please refer to the "*MYD-YT113X SDK Release Note*". In addition, users also need to have a detailed understanding of the CPU chip manual, as well as the product manual and pin definitions of the MYC-YT113X core board, in order to configure and use these pins correctly based on their actual functions.

## 7.1. How to Create Your Device Tree

### 7.1.1. Onboard Equipment 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 make appropriate adjustments to the Linux kernel device tree based on their actual hardware resources. Here is a list of device trees in various parts of the BSP of MYD-YT113X for user development reference. The specific content is shown in the table below:

Table 7-1. MYD-YT113X Device Tree List

Item	Device Tree	Description
U-boot	/device/config/chips/t113_i/configs/myir-image-yt113i-full/u-boot-board.dts	dts used by uboot
	/brandy/brandy-2.0/u-boot-2018/arch/arm/dts/myir-t113-lvds.dtsi	7-inch single channel LVDS screen device tree configuration
	/brandy/brandy-2.0/u-boot-2018/arch/arm/dts/myir-t113-lvds-dual.dtsi	19 inch dual LVDS screen device tree configuration
Kernel	/device/config/chips/t113_i/configs/myir-image-yt113i-full/linux-5.4/board.dts	Bottom board configuration resources, pin resource configuration
	/kernel/linux-5.4/arch/arm/boot/dts/sun8iw20p1.dtsi	Core resource allocation
	/kernel/linux-5.4/arch/arm/boot/dts/myir-t113-lvds.dtsi	7-inch single channel LVDS screen device tree configuration
	/kernel/linux-5.4/arch/arm/boot/dts/myir-t113-lvds-dual.dtsi	19 inch dual LVDS screen device tree configuration

This SDK source code provides two types of image configurations for the t113\_i model, using board.dts and uboot-board.dts at the following paths:

```
PC$: $HOME/T113/device/config/chips/t113_i/configs/myir-image-yt113i-full
```

```
PC$: $HOME/T113/device/config/chips/t113_i/configs/myir-image_core
```

The device tree configuration file for using LVDS under uboot is located at the following path:

```
PC$: $HOME/T113/brandy/brandy-2.0/u-boot-2018/arch/arm/dts
```

The device tree configuration file for using LVDS in the kernel is located at the following path:

```
PC$: $HOME/T113/kernel/linux-5.4/arch/arm/boot/dts
```

## 7.1.2. Adding 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 kernel by BootLoader, and kernel parses it to form the dev structure associated with the driver for the driver code to use. The following is an example of how to increase the device tree with kernel.

### 1). Add device tree to the kernel

A large number of platform device trees can be seen under the kernel source code under "*\$HOME/T113/kernel/linux-5.4/arch/arm/boot/dts*". If the device tree is suitable for MYD-YT113X, you can add a custom device tree under the current path:

mt8135-evbp1.dts	2024/6/21 17:09
mt8135-pinctrl.h	2024/6/21 17:09
mvebu-linkstation-fan.dtsi	2024/6/21 17:09
mvebu-linkstation-gpio-simple.dtsi	2024/6/21 17:09
mxs-pinctrl.h	2024/6/21 17:09
myir-t113-lvds.dtsi	2024/6/27 11:03
myir-t113-lvds-dual.dtsi	2024/6/27 11:03
nspire.dtsi	2024/6/21 17:09
nspire-classic.dtsi	2024/6/21 17:09
nspire-clp.dts	2024/6/21 17:09
nspire-cx.dts	2024/6/21 17:09
nspire-tp.dts	2024/6/21 17:09
nuvoton-common-npcm7xx.dtsi	2024/6/21 17:09
nuvoton-npcm750.dtsi	2024/6/21 17:09
nuvoton-npcm750-evb.dts	2024/6/21 17:09
omap2.dtsi	2024/6/21 17:09
omap3.dtsi	2024/6/21 17:09

Figure 7-1. Kernel custom device tree directory

Among them, "*myir-t113-lvds.dtsi*" and "*myir-t113-lvds-dual.dtsi*" are device trees added by MYiR itself and can be referenced.

## 2). Add device tree to the uboot

In the kernel source code "*branding/brandy-2.0/u-bot-2018/arch/arm/dts*" directory, you can see many device trees, and you can add custom device trees in the current path:

meson-gxbb-odroidc2.dts	2024/7/23 14:07
meson-gxl.dtsi	2024/7/23 14:07
meson-gxl-mali.dtsi	2024/7/23 14:07
meson-gxl-s905x.dtsi	2024/7/23 14:07
meson-gxl-s905x-khadas-vim.dts	2024/7/23 14:07
meson-gxl-s905x-libretech-cc.dts	2024/7/23 14:07
meson-gxl-s905x-p212.dts	2024/7/23 14:07
meson-gxl-s905x-p212.dtsi	2024/7/23 14:07
myir-t113-lvds.dtsi	2024/7/23 14:07
myir-t113-lvds-dual.dtsi	2024/7/23 14:07
omap3.dtsi	2024/7/23 14:07
omap3-beagle.dts	2024/7/23 14:07
omap3-beagle-u-boot.dtsi	2024/7/23 14:07
omap3-beagle-xm.dts	2024/7/23 14:07
omap3-beagle-xm-ab.dts	2024/7/23 14:07
omap3-beagle-xm-ab-u-boot.dtsi	2024/7/23 14:07

Figure 7-2. uboot custom device tree directory

Among them, "*myir-t113-lvds.dtsi*" and "*myir-t113-lvds-dual.dtsi*" are device trees added by MYiR itself and can be referenced.

### 3). Modify display scheme

The resources related to the MYC-YT113X core board are written in sun8i w20p1.dtsi and board.dts. Other extended interfaces and devices can be referenced to them as shown below (for reference only):

If you need to modify the configuration of different displays, you need to modify the following files (kernel and uboot device tree need to be modified, because only the full image to provide graphical display, so only modify the following path to the board.dts file)

`$HOME/T113/device/config/chips/t113_i/configs/myir-image-yt113i-fulll/board.dts`

- **kernel**

The following example is a 7-inch LVDS screen display configuration. If you need to change to a 19 inch dual channel LVDS screen display, open the corresponding

configuration annotation and annotate the original displayed configuration, as the MYD-YT113X development board does not support simultaneous display.

```
// $HOME/T113/device/config/chips/t113_i/configs/myir-image-yt113i-full/linux-5.4/board.dts

#include "sun8iw20p1.dtsi"
#include "myir/myir-t113-lvds.dtsi"
//#include "myir/myir-t113-lvds-dual.dtsi"

/{
    board = "t113_i", "t113_i-evb1_auto";
    model = "sun8iw20";
    compatible = "allwinner,t113_i", "arm,sun8iw20p1";
.....
```

#### ● uboot

After board.dts is modified, you also need to modify the uboot-board.dts file in the same directory, the modification method is the same as board.dts.

```
$HOME/T113/device/config/chips/t113_i/configs/myir-image-yt113i-full/uboot-board.dts

/*
 * Allwinner Technology CO., Ltd. sun8iw20p1 soc board.
 *
 * soc board support.
 */

#include "myir-t113-lvds.dtsi"
//#include "myir-t113-lvds-dual.dtsi"

.....
```

## 7.2. Configuring CPU Function Pins Based on Your Hardware

Implementing the control of a functional pin is one of the more complex system development processes, which includes the configuration of pins, development of drivers, implementation of applications, and other steps. This section does not specifically analyze the development process of each part, but uses examples to explain the control implementation of functional pins.

### 7.2.1. GPIO Pin Multiplexing Methods

GPIO: General-purpose input/output, general-purpose input/output port, in embedded devices is a very important resource, you can use them to output high and low levels or through them to read the state of the pin is high or low, MYD-YT113X package has a peripheral controller, these peripheral controllers with the external device handover is usually realized by controlling the GPIO, and the GPIO is used by the peripheral controller we call multiplexing (Alternate Function), to give them more complex functions, such as the user can use the GPIO port and the external device with more complex functions. is realized by controlling the GPIO, and the GPIO is used by peripheral controllers we call 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), so the GPIO is not the same as the GPIO. interrupt signal), etc., so the GPIO port is very widely used.

#### 1). GPIO pin multiplexing for uart4 function

To configure uart4 function, you need to find out which pins can be reused as uart4 function, the reuse relationship between pins can be referred to "MYC-YT113X-PinList" core board PinList list, the following will be an example of how to configure the gpio pins with uart4. (Explained with "myir-image-yt113i-full.img" image configuration)

- references the uart4 node

```
PC$: $HOME/T113/device/configs/myir-image-yt113i-full/board.dts
```

```
&uart4 {
```

```
pinctrl-names = "default", "sleep";
pinctrl-0 = <&uart4_pins_a>;
pinctrl-1 = <&uart4_pins_b>;
status = "okay";
};
```

- to view the pin schematic connections

Viewing the baseboard schematic shows pins 109 and 107 corresponding to the core module, as shown in Figure 7-3:

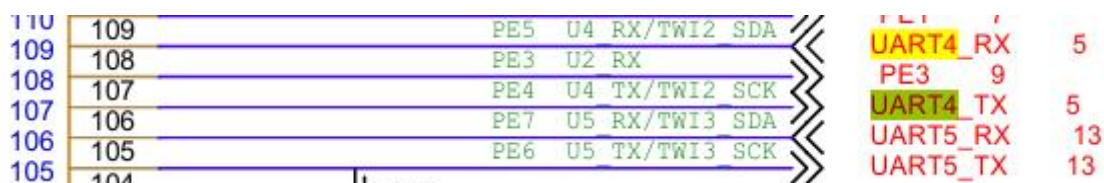


Figure 7-3. Pin schematic diagram

- to check the pin correspondences of the uart4 core module.

By looking at pin 109 and 107 of the core module PinList, you can know the corresponding PE4 and PE5, as shown in Figure 7-4:

Num	Pin Name	BGA625 Ball	MUX0	MUX2	MUX3	MUX4	MUX5	MUX6	MUX7	MUX8	MUX9	Power Rail
97	GND											
98	UART2_TX	L2	PE2	NCSIO-PCLK	UART2-TX	TWB-SCK	CLK-FANOUT0	UART0-TX		RGMI-RXD1/RMIL-RXD1	PE-ENT2	3V3
99	USB_OTG1_ID	R6	PE12	TW2-SCK	NCSIO-FIELD	Q2S0-DOUT2	Q2S0-DIN2			RGMI-TXD3	PE-ENT12	3V3
100	UART0_RX	D7	PG18	UART2-RX	TW3-SDA	PWM6	CLK-FANOUT1	SPDIF-OUT	UART0-RX		PG-ENT18	3V3
101	UART1_RTS	R3	PE8	NCSIO-D4	UART1-RTS	PWM2	UART3-TX	JTAG-MS		MDC	PE-ENT8	3V3
102												
103	UART0_TX	E7	PG17	UART2-TX	TWB-SCK	PWM7	CLK-FANOUT0	IR-TX	UART0-TX		PG-ENT17	3V3
104	GND											
105	UART5_TX	R1	PE6	NCSIO-D2	UART5-TX	TW3-SCK	SPDIF-IN	D-JTAG-DO	R-JTAG-DO	RGMI-TX/CTRL/RMIL-TXEN	PE-ENT6	3V3
106	UART5_RX	R2	PE7	NCSIO-D3	UART5-RX	TW3-SDA	SPDIF-OUT	D-JTAG-CK	R-JTAG-CK	RGMI-CLKIN/RMIL-RXER	PE-ENT7	3V3
107	UART4_TX	T2	PE4	NCSIO-D0	UART4-TX	TW2-SCK	CLK-FANOUT2	D-JTAG-MS	R-JTAG-MS	RGMI-TXD0/RMIL-TXD0	PE-ENT4	3V3
108	UART2_RX	U5	PE3	NCSIO-MCLK	UART2-RX	TW3-SDA	CLK-FANOUT1	UART0-RX		RGMI-TXCK/RMIL-TXCK	PE-ENT3	3V3
109	UART4_RX	T3	PE5	NCSIO-D1	UART4-RX	TW2-SDA	LED0-DO	D-JTAG-DI	R-JTAG-DI	RGMI-TXD1/RMIL-TXD1	PE-ENT5	3V3
110	ETH_RESETn	U1	PE1	NCSIO-VSYNC	UART2-CTS	TW1-SDA	LCDD-VSYNC			RGMI-RXD0/RMIL-RXD0	PE-ENT1	3V3
111	UART2_RTS	V1	PE0	NCSIO-HSYNC	UART2-RTS	TW1-SCK	LCDD-HSYNC			RGMI-RX/CTRL/RMIL-CRS-DV	PE-ENT0	3V3

Figure 7-4. Pin correspondence relationship

- **Configure GPIO reuse relationship**

From Figure 7-4, it can be seen that pins PE4 and PE5 can be configured as UART4\_TX and UART4-RX.

```
PC$: $HOME/T113/device/config/chips/t113_i/configs/myir-image-yt113i-full/bo
ard.dts
```

```
uart4_pins_a: uart4_pins@0 {
    pins = "PE4", "PE5";
    function = "uart4";
    drive-strength = <10>;
    bias-pull-up;
};
```

```
uart4_pins_b: uart4_pins@1 {
    pins = "PE4", "PE5";
    function = "gpio_in";
};
```

- **Add serial port alias**

At this point, you need to go to the following path to add a serial port alias

```
PC$: $HOME/T113/kernel/linux-5.4/arch/arm/boot/dts/sun8iw20p1.dtsi
```

```
/ {
    model = "sun8iw20";
    compatible = "allwinner,sun8iw20p1";
    interrupt-parent = <&gic>;
    #address-cells = <2>;
    #size-cells = <2>;

    aliases {
        serial0 = &uart0;
        serial1 = &uart1;
        serial2 = &uart2;
        serial3 = &uart1;
        serial4 = &uart4;
```



```
serial5 = &uart5;  
};
```

Finally, update the device tree according to section 7.1.2 and burn the new device tree onto the development board.

### 7.2.2. Example of Configuring Function Pins as GPIO Functions

This example uses the PD20 as a test GPIO to introduce how to configure a device node in the device tree and for use by the kernel driver in later chapters. This example can also be used as a reference for controlling reset, power and other control functions of external devices. (Take "*myir-image-yt113i-full.img*" image as an example).

Simply add nodes to the device tree.

```
PC$: $HOME/T113/device/config/chips/t113_i/configs/myir-image-yt113i-full/bo  
ard.dts
```

```
gpioctr_device {  
    compatible = "myir,gpioctr";  
    status = "okay";  
    gpioctr-gpios = <&pio PD 20 GPIO_ACTIVE_HIGH>;  
  
};
```

### 7.2.3. Development Board LCD Resource Reallocation Example

MYD-YT113X development board to define and implement the many rich features, but also occupies a large number of pin resources, such as the user directly using the MYD-YT113X based on the design and development, will need to redefine and configure the pin. The following LCD reuse pin function as an example, reuse relationship to view the "*MYC-YT113X Pin list*" document.

See the following directory dts file, you can know lvds0 function occupies the PD0

~ 10 pins, to be able to freely allocate the use of these pins, first of all, these pins will be released.

PC\$: \$HOME/T113/device/config/chips/t113\_i/configs/myir-image-yt113i-full/board.dts

Before modification:

/\*

\* Allwinner Technology CO., Ltd.

\*/

/dts-v1/;

#include "sun8iw20p1.dtsi"

#include "myir/myir-t113-lvds.dtsi"

//#include "myir/myir-t113-lvds-dual.dtsi"

.....

lvds0\_pins\_a: lvds0@0 {

allwinner,pins = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5", "PD6", "PD7", "PD8", "PD9";

allwinner,pname = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5", "PD6", "PD7", "PD8", "PD9";

allwinner,function = "lvds0";

allwinner,muxsel = <3>;

allwinner,drive = <3>;

allwinner,pull = <0>;

};

lvds0\_pins\_b: lvds0@1 {

allwinner,pins = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5", "PD6", "PD7", "PD8", "PD9";

```

        allwinner,pname = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5",
        "PD6", "PD7", "PD8", "PD9";
        allwinner,function = "lvds0_suspend";//io_disabled
        allwinner,muxsel = <7>;
        allwinner,drive = <3>;
        allwinner,pull = <0>;

};

```

The following is the modified display. It should be noted that the displayed references should be commented out (# include "*myrir-t113-lvds.dtsi*").

Note: These pins are also referenced in sun8iw20p1.dtsi and uboot-board.dts in the same directory, so they need to be modified in this way and will not be show here.

After modification:

```

/*
 * Allwinner Technology CO., Ltd.
 */

/dts-v1/

#include "sun8iw20p1.dtsi"
//#include "myir/myir-t113-lvds.dtsi"
//#include "myir/myir-t113-lvds-dual.dtsi"

.....

    lvds0_pins_a: lvds0@0 {
        allwinner,pins = " ";
        allwinner,pname = " ";
        allwinner,function = "lvds0";
        allwinner,muxsel = <3>;
        allwinner,drive = <3>;
        allwinner,pull = <0>;

};

```

```
lvds0_pins_b: lvds0@1 {
    allwinner,pins = " ";
    allwinner,pname = " ";
    allwinner,function = "lvds0_suspend";//io_disabled
    allwinner,muxsel = <7>;
    allwinner,drive = <3>;
    allwinner,pull = <0>;
};
```

### 7.3. How to use self configured pins

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

#### 7.3.1. Using GPIO pins in u-boot

uboot can directly use commands to control the GPIO settings. The following content uses PD20 as an example to illustrate the process of using GPIO.

Calculate the value of the pin corresponding to  $gpio=(n-1) * 32 + x$  (assuming A is 1, B is 2, and so on, D corresponds to 4) (x in PD20 represents 20) as follows:

Set 1 and set 0 for the PD20 pin, if you need to detect the need to use a multimeter to test the change in the level of the pin (value of  $(4-1) * 32 + 20 = 116$ ), you can use the following commands.

```
=> gpio clear 116
gpio: pin 116 (gpio 116) value is 0
=> gpio set 116
gpio: pin 116 (gpio 116) value is 1
```

#### 7.3.2. Using GPIO Pins in User Space

The Linux operating system architecture is divided into user space and kernel space (or user mode and kernel mode). User space is the activity space for higher-level applications, and their execution relies on resources provided by the kernel, including CPU resources, storage resources, I/O resources, etc. In order to allow

upper-layer applications to access these resources, the kernel must provide an interface for access: namely, system calls.

The shell is a special application, commonly referred to as the command line. It essentially serves as a command interpreter, issuing system calls while communicating with various applications. Using shell scripts, a few lines of shell code can achieve significant functionality because these shell statements typically encapsulate system calls. This facilitates interaction between the user and the system.

This section will explain how to use Shell to control GPIO pins in user mode.

- **Controlling Pins with Shell**

Shell control pins are essentially implemented by calling the file manipulation interface provided by Linux. Configured using the debugging interface, each gpio PIN has four attributes, namely, multiplexing (function), data, drive capability (dlevel), and pull-up/down state (pull). The operation is as follows.

```
root@myd-yt113-i:~# mount -t debugfs none /sys/kernel/debug
root@myd-yt113-i:~# cd /sys/kernel/debug/sunxi_pinctrl
```

View pin configuration:

```
root@myd-yt113-i:/sys/kernel/debug/sunxi_pinctrl# echo PD20 > sunxi_pin
root@myd-yt113-i:/sys/kernel/debug/sunxi_pinctrl# cat sunxi_pin_configure
pin[PD20] funciton: f
pin[PD20] data: 0
pin[PD20] dlevel: 20mA
pin[PD20] pull up: 0xffffffff
pin[PD20] pull down: 0xffffffff
pin[PD20] pull disable: 0x0
```

Modify pin attribute:

```
root@myd-yt113-i:/sys/kernel/debug/sunxi_pinctrl# echo PD20 1 > pull
root@myd-yt113-i:/sys/kernel/debug/sunxi_pinctrl# cat sunxi_pin_configure
pin[PD20] funciton: f
pin[PD20] data: 0
pin[PD20] dlevel: 20mA
```

```
pin[PD20] pull up: 0x1  
pin[PD20] pull down: 0xfffff  
pin[PD20] pull disable: 0xfffff
```

## 8. Application development and startup

A Makefile is essentially a document that defines a series of compilation rules. It records the detailed information on how the source code is compiled. Once a Makefile is written, you only need to run the make command, and the entire project will be compiled automatically, greatly improving the efficiency of software development. Makefiles are widely used in the development of Linux programs, whether for kernels, drivers, or applications.

### 8.1. Application Auto Startup Configuration

MYiR has added the S99autorun.sh user auto start service in the SDK, which is located at the SDK path of "*buildroot/buildroot-201902/board/myir/t113i full/rootfs overflow/etc/init.d*" and on the development board path of "*/etc/init.d*".

The content of the script is as follows:

```
root@myd-yt113-i:/# cat /etc/init.d/S99autorun.sh
#!/bin/sh
echo "user startup xxx.sh"
```

Users only need to put their application into this script, and the application will start and run automatically.

## 8.2. Modification of startup logo

The logo resource file is stored in the "*device/config/chips/t113\_i/boot-resource/boot-resource*" directory, if users need to change the logo to their own, they just need to replace the "*bootlogo.bmp*" file and replace it with your own logo, and the name should be *bootlogo.bmp*, and the image format must be .bmp.

After the replacement is completed, you need to recompile the whole image and update the image to replace it successfully.

**Note:** The format of the logo image must be converted to BMP format through tools. If it is in another format, manually changing the file format to BMP may cause it to not display. The resolution of the logo must be consistent with the current display screen resolution, otherwise it may also cause it to not display.



## 9. References

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

# Appendix A

## Warranty & Technical Support Services

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

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

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

### Service Guarantee

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

### Price

MYIR insists on providing customers with the most valuable products. We do not pursue excess profits which we think only for short-time cooperation. Instead, we hope to establish long-term cooperation and win-win business with customers. So we will offer reasonable prices in the hope of making the business greater with the customers together hand in hand.

### Delivery Time

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

### Technical Support

MYIR has a professional technical support team. Customer can contact us by email

(support@myirtech.com), we will try to reply you within 48 hours. For mass production and customized products, we will specify person to follow the case and ensure the smooth production.

### **After-sale Service**

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

The service covers:

#### **Technical support service**

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

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

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

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

#### **After-sales maintenance service**

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

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

- Due to unauthorized installation of the software, system or incorrect configuration or computer virus which has caused the damage of products.

**Warm tips**

1. MYIR does not supply maintenance service to LCD. We suggest the customer first check the LCD when receiving the goods. In case the LCD cannot run or no display, customer should contact MYIR within 7 business days from the moment get the goods.
2. Please do not use finger nails or hard sharp object to touch the surface of the LCD.
3. MYIR suggests user purchasing a piece of special wiper to wipe the LCD after long time use, please avoid clean the surface with fingers or hands to leave fingerprint.
4. Do not clean the surface of the screen with chemicals.
5. Please read through the product user manual before you using MYIR's products.
6. For any maintenance service, customers should communicate with MYIR to confirm the issue first. MYIR's support team will judge the failure to see if the goods need to be returned for repair service, we will issue you RMA number for return maintenance service after confirmation.

**Maintenance period and charges**

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

**Shipping cost**

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

**Products Life Cycle**

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

**Value-added Services**

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

2. MYIR provides the services of OS porting, BSP drivers' development, API software development, etc.
3. MYIR provides other products supporting services like power adapter, LCD panel, etc.
4. ODM/OEM services.

**MYIR Electronics Limited**

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

Yunli 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)