MYD-YT113X_Linux 软件开发指南

文件状态： [ ] 草稿 [√] 正式发布	**文件标识：**	MYIR-MYD-YT113X-SW-DG-ZH-L5.4.61
	当前版本：	V1.1[文档]
	作　　者：	Nico
	创建日期：	2023-05-01
	最近更新：	2023-08-30

版 本 历 史

版本	作者	参与者	日期	备注
V1.0[文档]	Nico	Licy	20230501	初始版本，适用于 MYD-YT113X
V1.1[文档]	Nico	Licy	20230830	新增 MYD-YT113-I 型号

深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

目 录

深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

1. 概述

系统的构建和定制化开发，目前比较常见的有 Buildroot, Yocto, OpenEmbedded 等等。其中 buildroot 项目使用更轻便和定制化的方法，来构建出适合嵌入式产品的 Linux 系统。它不仅仅是一个制作文件系统的工具，同时还可以提供整套的基于 Linux 的开发和维护工作，使底层嵌入式开发者和上层应用开发者在统一的框架下开发，解决了传统开发方式下零散和无管理的开发形态。

本文主要介绍基于全志 T113 处理器的 SDK 项目和米尔核心板定制一个完整的嵌入式 Linux 系统的完整流程，其中包括开发环境的准备，代码的获取，以及如何进行 Bootloader, Kernel 的移植，定制适合自身应用需求的根文件系统 rootfs 等。

首先介绍如何基于米尔提供的源代码构建适用于 MYD-YT113X 开发板的系统镜像，以及如何将构建好的镜像更新下载到开发平台。其次针对那些基于 MYC-YT113X 核心板进行项目定制的用户，我们重点将介绍了如何使用这一套 SDK 移植到用户的硬件平台上的方法和重难点解析。最后也会通过一些实际的驱动移植案例和 Rootfs 定制的案例，使用户能够迅速开发符合自己硬件的系统镜像。

请注意本文档并不包含 buildroot 项目以及 Linux 系统相关基础知识的介绍，适合有一定开发经验的嵌入式 Linux 系统开发人员和嵌入式 Linux BSP 开发人员。针对用户在进行二次开发过程中可能会使用到的一些具体功能，我们也提供了详细的应用笔记供发人员参考，具体的信息参见《**MYD-YT113X SDK 发布说明**》表 2-4 中的文档列表。

本文适用开发板和核心板列表：

表 1-1.核心板与开发板列表

核心板	开发板
MYC-YT113-S3	MYD-YT113-S3
MYC-YT113-i	MYD-YT113-i

深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

1.1. 软件资源

MYD-YT113X 搭载基于 Linux 5.4.61 版本内核的操作系统，提供了丰富的系统资源和其他软件资源。开发板出厂附带嵌入式 Linux 系统开发所需要的交叉编译工具链，U-boot 源代码，Linux 内核和各驱动模块的源代码等资料包，以及适用于 Windows 桌面环境和 PC Linux 系统的各种开发和调试工具，应用开发例程等。具体的包含的软件信息请参考《**MYD-YT113X SDK 发布说明**》中第 2 章软件信息中的说明。

1.2. 文档资源

根据用户使用开发板的各个不同用途。将会为客户提供完整的 SDK 包（Software Development Kit），SDK 中包含了发布说明，入门指南，评估指南，开发指南，应用笔记，常用问答等不同类别的文档和手册。具体的文档列表参见《**MYD-YT113X SDK 发布说明**》表 2-4 中的说明。

2. 开发环境准备

本章主要介绍基于 MYD-YT113X 开发板在开发流程所需的一些软硬件环境，包括必要的开发主机环境，必备的软件工具，代码和资料的获取等，具体的准备工作下面将进行详细介绍。

2.1. 开发主机环境

如何搭建适用于全志 T1 系列处理器平台的开发环境。通过阅读本章节，您将了解相关硬件工具，软件开发调试工具的安装和使用。并能快速的搭建相关开发环境，为后面的开发和调试做准备。全志 T1 系列处理器是 SMP 多核架构的处理器，2 个 ARM Cortex A7，可以运行嵌入式 Linux 系统，使用常用的嵌入式 Linux 系统的开发工具即可。

- **主机硬件**

整个 SDK 包项目的构建对开发主机的要求比较高，要求处理器具有双核以上 CPU，4 GB 以上 内存，100GB 硬盘或更高配置。可以是安装 Linux 系统的 PC 或服务器，也可以是运行 Linux 系统的虚拟机，Windows 系统下的 WSL2 等。

- **主机操作系统**

构建 buildroot 项目的主机操作系统可以有很多种选择，一般选择在安装 Fedora, openSUSE, Debian, Ubuntu, RHEL 或者 CentOS 等 Linux 发行版的本地主机上进行构建，这里推荐的是 Ubuntu18.04 64bit 桌面版系统（Ubuntu20.04 64bit 及以上版本也可使用），在该系统环境编译比较稳定，后续开发也是以此系统为例进行介绍。

- **安装必备软件包**

在主机端先安装必要的开发依赖包

```
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
```

其他非必须配置包

深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

```
PC$: sudo dpkg-reconfigure dash #选择 no
PC$: sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1 /usr/lib/i386-linux-gnu/libGL.so
PC$: sudo apt-get install zlib1g-dev # 缺失 libz.so 时安装
PC$: sudo apt-get install uboot-mkimage # 缺失 mkimge 时安装或者安装 u-boot-tools
```

2.2. 软件环境

2.2.1. 资料获取

搭建环境前先下载开发板资料，关于开发资料的详细信息请查阅《**MYD-YT113X SDK 发布说明**》开发板资料下载地址如下（资料会不定期更新，请下载最新版）。

http://down.myir-tech.com/MYD-YT113

2.2.2. 安装交叉编译工具链

在使用 SDK 构建这个系统镜像过程中，还需要安装交叉工具链，米尔提供的这个 SDK 中除了包含各种源代码外还提供了必要的交叉工具链，可以直接用于编译应用程序等。用户可以直接使用次交叉编译工具链来建立一个独立的开发环境，可单独编译 Bootloader，Kernel 或者编译自己的应用程序，具体过程在后面的章节中将会详细介绍。这里先介绍 SDK 的安装步骤，如下：

- **拷贝 SDK 到 Linux 目录并解压**

将 SDK 压缩包拷贝到 Ubuntu 下的用户工作目录，如$HOME/T113X 下，该目录根据自己实际情况定义，然后解压文件，得到 SDK 源码文件，如下：

```
PC$ cd $HOME/T113X
PC$ tar -jxf YT113X-buildroot-T1-5.4.61-X.X.X.tar.bz2
```

注：其中 X.X.X 代表当前版本号

- **查看编译链文件**

进入 SDK 目录，在 build/toolchain 目录下可以找到：

```
PC$ cd $HOME/T113X/auto-t113x-linux/build/toolchain
PC$ $HOME/T113X/auto-t113x-linux/build/toolchain$ ls
gcc-linaro-5.3.1-2016.05-x86_64_aarch64-linux-gnu.tar.xz
gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu.tar.xz
```


深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

```
gcc-linaro-arm.tar.xz
gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabi.tar.xz
gcc-linaro-aarch64.tar.xz
gcc-linaro.tar.bz2
```

标红的压缩包就是 T113 的编译链

- **解压编译链**

将解压到主机的/opt 目录下，用户也可以根据提示自己选择合适的目录

```
PC$ $HOME/T113X/auto-t113x-linux/build/toolchain
tar -xf gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabi.tar.xz -C /opt
```

- **安装、测试编译链**

设置环境变量，并测试安装是否完成。

```
PC$  export PATH=$PATH:/opt/gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabi/bin
PC$ arm-linux-gnueabi-gcc -v
Using built-in specs.
COLLECT_GCC=arm-linux-gnueabi-gcc
COLLECT_LTO_WRAPPER=/home/sur/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-build
```


深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

```
slave/workspace/tcwg-make-release/label/docker-trusty-amd64-tcwg/target/arm
-linux-gnueabi/_build/sysroots/arm-linux-gnueabi --enable-lto --enable-linker-bu
ild-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/lib
c --enable-languages=c,c++,fortran,lto --enable-checking=release --disable-boot
strap --build=x86_64-unknown-linux-gnu --host=x86_64-unknown-linux-gnu --ta
rget=arm-linux-gnueabi --prefix=/home/tcwg-buildslave/workspace/tcwg-make-r
elease/label/docker-trusty-amd64-tcwg/target/arm-linux-gnueabi/_build/builds/d
estdir/x86_64-unknown-linux-gnu
Thread model: posix
gcc version 5.3.1 20160412 (Linaro GCC 5.3-2016.05)
```

可以看到最后一行打印跟安装的版本一致，证明安装成功

深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

3. 使用 SDK 构建开发板镜像

3.1. 简介

Linux SDK 开发包，它集成了 BSP、构建系统、linux 应用、测试系统、独立 IP、工具和文档，既可作为 BSP、IP 的开发、验证和发布平台，也可作为嵌入式 Linux 系统。

是统一使用的 linux 开发平台。它集成了 BSP，构建系统，独立 IP 和测试，既可作为 BSP 开发和 IP 验证平台，也可以作为量产的嵌入式 linux 系统。

Linux SDK 的功能包括以下四部分：

- BSP 开发，包括 bootloader，uboot 和 kernel。
- Linux 文件系统开发，包括量产的嵌入式 linux 系统。
- IP 的验证和发布平台，并且给出 IP 的使用方法和系统集成的 demo 程序，方便第三方快速使用。
- 测试，包括板级测试和系统测试。

米尔提供的光盘镜像中 04_sources 目录下提供了适用于 MYD-YT113X 开发板的 linux SDK 文件和数据，帮助开发者构建出可运行在 MYD-YT113X 开发板上的不同类型 Linux 系统镜像，如下表：

表 3-1. MYD-YT113X 镜像文件说明

镜像文件名称	内容描述
myir-image-yt113s3-emmc-core	以 buildroot 构建不带 QT 图形界面的镜像。
myir-image-yt113s3-emmc-full	以 buildroot 构建带有 QT 图形界面，7 寸 LVDS 显示的镜像。
myir-image-yt113s3-nand	以 buildroot 构建不带 QT 图形界面的镜像。
myir-image-yt113i-emmc-core	以 buildroot 构建不带 QT 图形界面的镜像。
myir-image-yt113i-emmc-full	以 buildroot 构建带有 QT 图形界面，7 寸 LVDS 显示的镜像。

下面以构建 myir-image-yt113s3-emmc-full 镜像为例进行介绍具体的开发流程，为后续定制适合自己的系统镜像打下基础，用户请根据自己开发板型号选择对应的操作，基本操作都与制作 myir-image-yt113s3-emmc-full 镜像一致，如有不同操作本文将会特定指出，请用户仔细观看本文档操作，以免操作不当造成镜像制作失败。

3.2. 获取源码

深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

我们提供两种获取源码的方式，一种是直接从米尔光盘镜像 04_sources 目录中获取压缩包，另外一种是使用 repo 获取位于 github 上实时更新的源码进行构建，请用户根据实际需要选择其中一种进行构建。

3.2.1. 从光盘镜像获取源码压缩包(推荐)

压缩的源码包位于米尔开发包资料 04_Sources/YT113X-buildroot-T1-5.4.61-X.X.X.tar.bz2(X.X.X 代表当前版本号)。拷贝压缩包到用户指定目录，如$HOME/T113X 目录，这个目录将作为后续构建的顶层目录，按照下面的方式解压后出现 SDK 所有内容：

```
PC$ cd $HOME/T113X
PC$ $HOME/T113X$ tar -jxf YT113X-buildroot-T1-5.4.61-X.X.X.tar.bz2
PC$ $HOME/T113X/ auto-t113x-linux$ tree -L 1
.
├── brandy
├── build
├── buildroot
├── build.sh
├── device
├── kernel
├── out
├── platform
├── test
└── tools

9 directories, 1 file
```

3.2.2. 通过 github 获取源码

目前 MYD-YT113X 开发板的 BSP 源代码和 Buildroot 源代码均使用了 github 托管并将保持长期更新，代码仓库地址请查看《**MYD-YT113X_SDK 发布说明**》。用户可以使用 repo 获取和同步 github 上的代码。具体操作方法如下：

```
PC$ mkdir $HOME/T113X
PC$ cd $HOME/T113X
PC$ export REPO_URL='https://mirrors.tuna.tsinghua.edu.cn/git/git-repo/'
```


深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

```
PC$ repo init -u git@github.com:MYIR-ALLWINNER/myir-t1-manifest.git --no-clone-bundle --depth=1 -m myir-t113-5.4.61-1.0.0.xml -b develop-yt113x-manifest
PC$ repo sync
```

代码同步成功之后，同样在$HOME/T113X 目录下得到一个 SDK 文件夹，在里面包含 MYD-YT113X 开发板相关的源码或者源码仓库的路径，目录结构和从压缩包解压出来的一样。

3.3. 认识 linux SDK 结构

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

主要由 brandy、buildroot、 kernel、platform 组成。

- brandy 包含 uboot2018
- buildroot 负责 ARM 工具链、应用程序软件包、Linux 根文件系统生成
- kernel 为 linux 内核
- platform 是平台相关的库和 sdk 应用

3.3.1. buildroot 介绍

Buildroot 是一组 Makefile 和补丁，可简化并自动化地为嵌入式系统构建完整的、可启动的 Linux 环境(包括 bootloader、Linux 内核、包含各种 APP 的文件系统)。Buildroot 运行于 Linux 平台，可以使用交叉编译工具为多个目标板构建嵌入式 Linux 平台。Buildroot 可以自动构建所需的交叉编译工具链，创建根文件系统，编译 Linux 内核映像，并生成引导加载程序用于目标嵌入式系统，或者它可以执行这些步骤的任何独立组合。例如，可以单独使用已安装的交叉编译工具链，而 Buildroot 仅创建根文件系统。

目录结构如下

```
buildroot-201902/
```


深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

```
├── 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
```

其中 configs 目录里存放预定义好的配置文件，如下表：

表 3-2. MYD-YT113X buildroot 配置文件说明

型号	配置文件	描述
MYD-YT113-S3	myd_yt113s3_emmc_core_br_defconfig	emmc core 系统 buildroot 配置
	myd_yt113s3_emmc_full_br_defconfig	emmc full 系统 buildroot 配置
	myd_yt113s3_nand_br_defconfig	nand 系统 buildroot 配置
MYD-YT113-I	myd_yt113i_emmc_core_br_defconfig	emmc core 系统 buildroot 配置
	myd_yt113i_emmc_full_br_defconfig	emmc full 系统 buildroot 配置

这些配置文件已经定义好了不同型号的不同镜像的配置，dl 目录里存放已经下载好的软件包，scripts 目录里存放 buildroot 编译的脚本，mkcmd.sh，mkcommon.sh，mkrule 和 mksetup.sh 等。target 目录里存放用于生成根文件系统的一些规则文件，该目录，对于代码和工具的集成非常重要。对于我们来说最为重要的是 package 目录，里面存放了将近 3000 个软件包的生成规则，我们可以在里面添加我们自己的软件包或者是中间件。

更多关于 buildroot 的介绍，可以到 buildroot 的官方网站 http://buildroot.uclibc.org/获取。

3.3.2. kernel

linux 内核源码目录。当前使用的内核版本是 linux5.4.61。除了 modules 目录，以上目录结构跟标准的 linux 内核一致。modules 目录是我们用来存放没有跟内核的 menuconfig 集成的外部模块的地方。

3.3.3. brandy

brandy 目录下有 brandy2.0 版本，目前 T113X 使用 brandy2.0 版本，其目录结构为:

```
brandy-2.0/
├── build.sh -> tools/build.sh
├── spl-pub
├── tools
└── u-boot-2018
```

3.3.4. platform

平台私有软件包目录。

```
platform/
├── apps
├── base
├── config
├── core
├── external
├── framework
├── Makefile -> /home/sur/T113-core/auto-t113x-linux/build/Makefile
└── tools
```

其中，framework/auto 内包含了 T1 linux 版本的 SDK 接口和示例。

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

其中 rootfs 会在每次顶层执行 build.sh 的时候强制覆盖到 out 目录相应的 target 下（target 为机器的根文件系统目录）。

framework/qt 内包含了 QT5.12.5 的源代码。

3.3.5. tools

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

3.3.6. 原厂 test 系统

test 是一个测试系统，名叫 dragonboard。dragonboard 提供快速的板级测试。

3.3.7. device

该目录包含了 t113(MYD-YT113-S3)、t113_i(MYD-YT113-I)两个产品，t113 产品目录下包含三种类型镜像配置， t113_i 产品目录下包含两种类型镜像配置。其中 MYD-YT113-S3 和 MYD-YT113-I 的 core 跟 full 系统配置目录结构基本一致，只有内核和 buildroot 配置有差异，具体差异请看《**MYD-YT113X_SDK 发布说明**》2.1 章节。由于型号比较多，而且每个型号的 full 跟 core 系统的目录结构差异不大，所以本文只讲解不同配置的开发方法。

/device/config/chips/t113 是 MYD-YT113-S3 型号芯片配置目录，/device/config/chips/t113_i 是 MYD-YT113-I 型号芯片配置目录，内包含多个板级配置，每个板级配置都有不同的 board.dts，sys_config.fex 等配置文件。

- **MYD-YT113-S3 型号配置讲解**

主要内容如下：

```
device/config/chips/t113
```


深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

```
├── bin
│   ├── boot0_nand_sun8iw20p1.bin          nand 用的 boot0 启动文件
│   ├── boot0_sdcard_sun8iw20p1.bin        emmc 用的 boot0 启动文件
│   ├── dsp0.bin
│   ├── fes1_sun8iw20p1.bin                烧录工具用的初始化文件
│   ├── optee_sun8iw20p1.bin               optee
│   ├── sboot_sun8iw20p1.bin               安全启动的 bin
│   └── u-boot-sun8iw20p1.bin
├── boot-resource
│   ├── boot-resource
│   │   ├── bat
│   │   ├── bootlogo.bmp
│   │   ├── fastbootlogo.bmp
│   └── boot-resource.ini
├── configs
│   ├── default                            正常情况下不生效
│   ├── myir-image-yt113s3-emmc-full       EMMC 板型
│   │   ├── board.dts                      EMMC 板级 dts 配置
│   │   ├── bsp
│   │   │   ├── BoardConfig.mk
│   │   │   ├── BoardConfig_nor.mk
│   │   │   ├── bootlogo.bmp
│   │   │   ├── env.cfg
│   │   │   ├── env_nor.cfg
│   │   │   ├── sys_partition.fex
│   │   │   └── sys_partition_nor.fex
│   │   ├── linux-5.4
│   │   │   ├── board.dts
│   │   │   └── config-5.4
│   │   ├── longan
│   │   │   ├── BoardConfig.mk             内核，buildroot,工具链等配置
│   │   │   ├── BoardConfig_nor.mk
│   │   │   ├── bootlogo.bmp               EMMC 板型 bootlogo 图片
```


深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

```
|   |   |   ├── env_ab.cfg
|   |   |   ├── env.cfg                          EMMC 板型环境变量
|   |   |   ├── env-recovery.cfg
|   |   |   ├── sys_partition.fex                EMMC 板型默认分区文件
|   |   ├── sys_config.fex                       EMMC 板型 sys_config 配置
|   |   └── uboot-board.dts                      EMMC 板型 uboot 使用的 dts 文件
|   ├── myir-image-yt113s3-nand                                SPI NAND 板级
目录
|   |   ├── BoardConfig.mk
|   |   ├── board.dts -> linux-5.4/board.dts     SPI NAND 板级 dts 配置
|   |   ├── bsp
|   |   |   ├── bootlogo.bmp
|   |   |   ├── env.cfg
|   |   |   └── sys_partition.fex
|   |   ├── env.cfg
|   |   ├── linux-5.4
|   |   |   ├── board.dts
|   |   |   └── config-5.4
|   |   ├── longan
|   |   |   ├── BoardConfig.mk                   内核，buildroot,工具链等配置
|   |   |   ├── bootlogo.bmp                     SPI NAND 板型 bootlogo 图片
|   |   |   ├── env.cfg                          SPI NAND 板型环境变量
|   |   |   └── sys_partition.fex                SPI NAND 板型默认分区文件
|   |   ├── sys_config.fex                       SPI NAND 板型 sys_config 配置
|   |   ├── sys_partition.fex
|   |   └── uboot-board.dts                      SPI NAND 板型 uboot 使用的 dts 文件
```

- **MYD-YT113-I 型号配置讲解**

```
device/config/chips/t113_i
├── bin
|   ├── boot0_nand_sun8iw20p1.bin                 nand 用的 boot0 启动文件
|   ├── boot0_sdcard_sun8iw20p1.bin               emmc 用的 boot0 启动文件
|   ├── boot0_spinor_sun8iw20p1.bin
|   ├── dsp0.bin
```

```
|   ├── fes1_sun8iw20p1.bin                烧录工具用的初始化文件
|   ├── optee_sun8iw20p1.bin               optee
|   ├── sboot_sun8iw20p1.bin               安全启动的 bin
|   ├── u-boot-spinor-sun8iw20p1.bin
|   └── u-boot-sun8iw20p1.bin
├── boot-resource
|   ├── boot-resource
|   |   ├── bat
|   |   ├── bootlogo.bmp
|   |   ├── fastbootlogo.bmp
|   |   ├── font24.sft
|   |   ├── font32.sft
|   |   └── wavefile
|   └── boot-resource.ini
├── configs
|   ├── default                                正常情况下不生效
|   └── myir-image-yt113i-full                 EMMC 板型
|       ├── board.dts                          EMMC 板级 dts 配置
|       ├── bsp
|       ├── linux-5.4
|       ├── longan
|   |   |   ├── BoardConfig.mk                 内核，buildroot,工具链等配置
|   |   |   ├── BoardConfig_nor.mk
|   |   |   ├── bootlogo.bmp                   EMMC 板型 bootlogo 图片
|   |   |   ├── env_ab.cfg
|   |   |   ├── env.cfg                        EMMC 板型环境变量
|   |   |   ├── env_nor.cfg
|   |   |   ├── sys_partition_ab.fex
|   |   |   ├── sys_partition.fex              EMMC 板型默认分区文件
|   |   ├── sys_partition.fex                  EMMC 板型 sys_config 配置
|   |   └── uboot-board.dts                    EMMC 板型 uboot 使用的 dts 文件
```

3.4. Linux SDK 的配置与构建

本节介绍全编译和部分编译的详细步骤。编译完成后，通过打包，生成最终的 img。

本章节以生成 myir-image-yt113s3-emmc-full.img 和 myir-image-yt113i-full.img 镜像讲解，执行如下步骤：

首先进入到源码顶层目录：auto-t113x-linux，然后执行下面的命令。

```
PC$: cd $HOME/T113X/auto-t113x-linux
PC$: $HOME/T113X/auto-t113x-linux$ ./build.sh config
执行完之后选择对应开发板型号的配置，详细选择在后面讲解
PC$: $HOME/T113X/auto-t113x-linux$ ./build.sh
============如果不需要 qt 功能，可以跳过这两步命令================
PC$: $HOME/T113/auto-t113x-linux$ ./build.sh qt
PC$: $HOME/T113X/auto-t113x-linux$ ./build.sh
==================================================================
PC$: $HOME/T113X/auto-t113x-linux$ ./build.sh pack
```

上面介绍了整个编译流程所需要的命令，下面会分步讲解每个命令。

1). 第一步 “./build.sh config”

执行 build.sh config，在后续对话中选择配置 。Choice 选择时可以输入数字也可以输入数字对应的字符串。

- **MYD-YT113-S3 型号配置选择**

```
PC$ ./build.sh config
Welcome to mkscript setup progress
All available platform:
  0. linux
Choice [linux]: 0
All available linux_dev:
  0. bsp
  1. dragonboard
  2. longan
  3. tinyos
Choice [longan]: 2
All available kern_ver:
  0. linux-5.4
Choice [linux-5.4]: 0
```

```
All available ic:
   0. t113
   1. t113_i
Choice [t113]: 0
All available board:
   0. myir-image-yt113s3-emmc-core
   1. myir-image-yt113s3-emmc-full
   2. myir-image-yt113s3-nand
Choice [myir-image-yt113s3-emmc-full]: 1

S3 型号 nand 开发板只有 core 系统，只能选择 2
Emmc 开发板可以选择 core，full 系统。

All available flash:
   0. default
   1. nor
Choice [default]: 0
All available gnueabi:
   0. gnueabi
   1. gnueabihf
Choice [gnueabi]: 0
```

- **MYD-YT113-I 型号配置选择**

```
PC$ ./build.sh config
Welcome to mkscript setup progress
All available platform:
   0. linux
Choice [linux]: 0
All available linux_dev:
   0. bsp
   1. dragonboard
   2. longan
   3. tinyos
Choice [longan]: 2
```

```
All available kern_ver:
  0. linux-5.4
Choice [linux-5.4]: 0
All available ic:
  0. t113
  1. t113_i
Choice [t113_i]: 1
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
All available gnueabi:
  0. gnueabi
  1. gnueabihf
Choice [gnueabi]: 0
```

如果在执行完./build.sh config 后，出现下面的错误提示。

```
 File "<string>", line 1
   import os.path; print os.path.relpath('/home/sur/T113-core/auto-t113x-linux/kernel/linux-5.4/arch/arm/configs/sun8iw20p1smp_t113_auto_defconfig', '/home/sur/T113-core/auto-t113x-linux/kernel/linux-5.4/arch/arm/configs')
                       ^
SyntaxError: invalid syntax
ERROR: Can't find kernel defconfig!
```

那么就是 python 版本不对造成，需要 python2 的版本。请检查当前开发环境是否已经安装了 python2。如未安装请查看 2.1 章节或者直接执行下面命令：

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

检查当前版本是否是 python2

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

如果安装了 python2 但是查看版本还是其他版本，那么就是主机环境安装了多个 python 版本，此时需要自行切换 python 版本，执行如下命令进行切换版本：

按照自己实际主机环境进行切换

```
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
有 2 个候选项可用于替换 python (提供 /usr/bin/python)。

  选择       路径            优先级  状态
------------------------------------------------------------
  0            /usr/bin/python2.7   2         自动模式
* 1            /usr/bin/python2.7   2         手动模式
  2            /usr/bin/python3.6   1         手动模式

要维持当前值[*]请按<回车键>，或者键入选择的编号：1
```

2). 第二步 “./build.sh”

配置选择完之后，执行下面命令开始编译系统，此过程需要比较长的时间。

```
PC$: cd $HOME/T113X/auto-t113x-linux
PC$: $HOME/T113X/auto-t113x-linux$ ./build.sh                该命令是进行整体编译
ACTION List: mklichee;========
Execute command: mklichee
INFO: ------------------------------------
INFO: build lichee ...
INFO: chip: sun8iw20p1
INFO: platform: linux
INFO: kernel: linux-5.4
INFO: board: myir-image-yt113s3-emmc-full
INFO: output: /home/nico/T113X/auto-t113x-linux/out/t113/ myir-image-yt113s3
-emmc-full/longan
```

```
INFO: ----------------------------------------
INFO: build buildroot ...
.......
INFO: pack rootfs ok ...
INFO: ----------------------------------------
INFO: build lichee OK.
INFO: ----------------------------------------
```

执行该步骤时可能会遇到下面的错误

```
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\"",
```

图 3-1. 编译 GDbus 失败 1

修改步骤 1：(注意可能路径全名不一样，根据自己的实际路径找到 gdbusauth.c 即可)

```
PC$: $HOME/T113X/auto-t113x-linux$ vim ./out/t113/ myir-image-yt113s3-emmc
-full /longan/buildroot/build/host-libglib2-2.56.3/gio/gdbusauth.c
```

在如下位置添加此判断代码：

```
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)
```

修改步骤 2：(注意可能路径全名不一样，根据自己的实际路径找到 gdbusmessage.c 即可)

```
PC$: $HOME/T113X/auto-t113x-linux$ vim ./out/t113/ myir-image-yt113s3-emmc
-full /longan/buildroot/build/host-libglib2-2.56.3/gio/gdbusmessage.c
```

在如下位置添加此判断代码：

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

```
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/auto-t113-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/auto-t113-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/auto-t113-linux/buildroot/buildroot-201902'
ERROR: build buildroot Failed
```

图 3-2. 编译 GDbus 失败 2

```
PC$ cd $HOME/T113X/auto-t113x-linux/out/t113/ myir-image-yt113s3-emmc-full
 /longan/buildroot/build/libgpg-error-1.33/src
```

将该目录下的 Makefile、Makefile.am、Makefile.in、mkstrtable.awk 里的 namespace 都改为 pkg_namespace，然后重新执行编译命令即可(./build.sh)。

3). ./build.sh qt (编译 qt)

如果不需要 qt 功能则可以跳过这一步骤，执行下面的命令即可编译 qt：

```
PC$: $HOME/T113X/auto-t113x-linux$ ./build.sh qt
ACTION List: mkqt;========
```

```
Execute command: mkqt
INFO: build Qt ...
INFO: build arm-linux-gnueabi version's Qt
$HOME/T113X/auto-t113x-linux/platform/framework/qt/qt-everywhere-src-5.12.5
.......

INFO: build buildroot OK.
INFO: build Qt and buildroot Ok.
```

此时 qt 编译成功，然后一定要在执行一次“./build.sh”命令，这个命令会将刚刚编译 qt 所生成的相关库、demo 和其他文件移到文件系统中，否则文件系统将不会带有 qt。

4). 第三步 “./build.sh pack”

最后一步打包镜像生成系统，执行如下命令：

```
PC$: $HOME/T113X/auto-t113x-linux$ ./build.sh pack
ACTION List: mkpack ;=========
Execute command: mkpack
INFO: packing firmware ...
INFO: Use BIN_PATH: $HOME/T113X/auto-t113x-linux/device/config/chips/t113/bin
......
Dragon execute image.cfg SUCCESS !
----------image is at----------

size:456M  $HOME/T113X/auto-t113x-linux/out/ myir-image-yt113s3-emmc-full.img
```

3.5. 板载 u-boot 编译和更新

U-boot 是一个功能非常丰富的开源启动引导程序，包括内核引导，下载更新等众多方面，在嵌入式领域应用十分广泛，可查看官网获取更多信息 http://www.denx.de/wiki/U-Boot/WebHome

T1 平台也使用 boot chains 做启动引导程序，不同的 boot chains 模式会对应不同启动阶段。

本章节以 myir-image-yt113s3-emmc-full 镜像为例讲解，其他型号步骤基本一致，如有不同之处将会提出，请仔细阅读文档，避免操作不当导致失败。

3.5.1. 在独立的交叉编译环境下编译 u-boot

1). 获取 u-boot 源代码

拷贝开发包 04_Source/YT113X-buildroot-T1-5.4.61-X.X.X.tar.bz(X.X.X 代表当前版本)到指定自定义的 T113X 目录（如$HOME/T113X），解压进入源码目录并查看对应的文件信息，如拷贝到 T113X 目录:

```
PC$ cd $HOME/T113X
PC$ $HOME/T113X$ tar -jxf YT113X-buildroot-T1-5.4.61-X.X.X.tar.bz
```

- 源代码目录：u-boot-2018
- SPL 源代码目录： spl-pub
- 编译脚本：build.sh

```
PC$ $HOME/T113X$ cd auto-t113x-linux/brandy/brandy-2.0$
PC$: $HOME/T113X/auto-t113x-linux/brandy/brandy-2.0$ tree -L 1
├── build.sh -> tools/build.sh
├── spl-pub
├── tools
└── u-boot-2018
```

2). 配置与编译

- **进入源代码目录**

```
PC$: $HOME/T113X/auto-t113x-linux/$ cd brandy/brandy-2.0
```

- **加载 SDK 里的工具链**

```
PC$: $HOME/T113X/auto-t113x-linux/brandy/brandy-2.0$ ./build.sh -t
grep: HOME/T113X/auto-t113x-linux/brandy/brandy-2.0/spl/Makefile: 没有那个文件或目录(这个信息无影响)
Prepare toolchain ...
```

- **加载 defconfig 配置文件**

- MYD-YT113-S3 型号

```
PC$: $HOME/T113X/auto-t113x-linux/brandy/brandy-2.0$ cd u-boot-2018
eMMC 板型:
```


深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

```
PC$: $HOME/T113X/auto-t113x-linux/brandy/brandy-2.0/u-boot-2018$ make sun8iw20p1_auto_defconfig

Nand 板型：
PC$: $HOME/T113X/auto-t113x-linux/brandy/brandy-2.0/u-boot-2018$ make sun8iw20p1_auto_nand_defconfig
```

- MYD-YT113-I 型号

```
PC$: $HOME/T113X/auto-t113x-linux/brandy/brandy-2.0/u-boot-2018$ make sun8iw20p1_auto_t113_i_defconfig
```

注意：一定要根据自己开发板型号加载对应的 defconfig 文件

- **修改 uboot 配置**

  - make menuconfig 图形界面修改

  加载完 defconfig 文件后，执行 make menuconfig 打开 uboot 图形界面配置。

```
PC$: $HOME/T113X/auto-t113x-linux/brandy/brandy-2.0/u-boot-2018$ make menuconfig
```

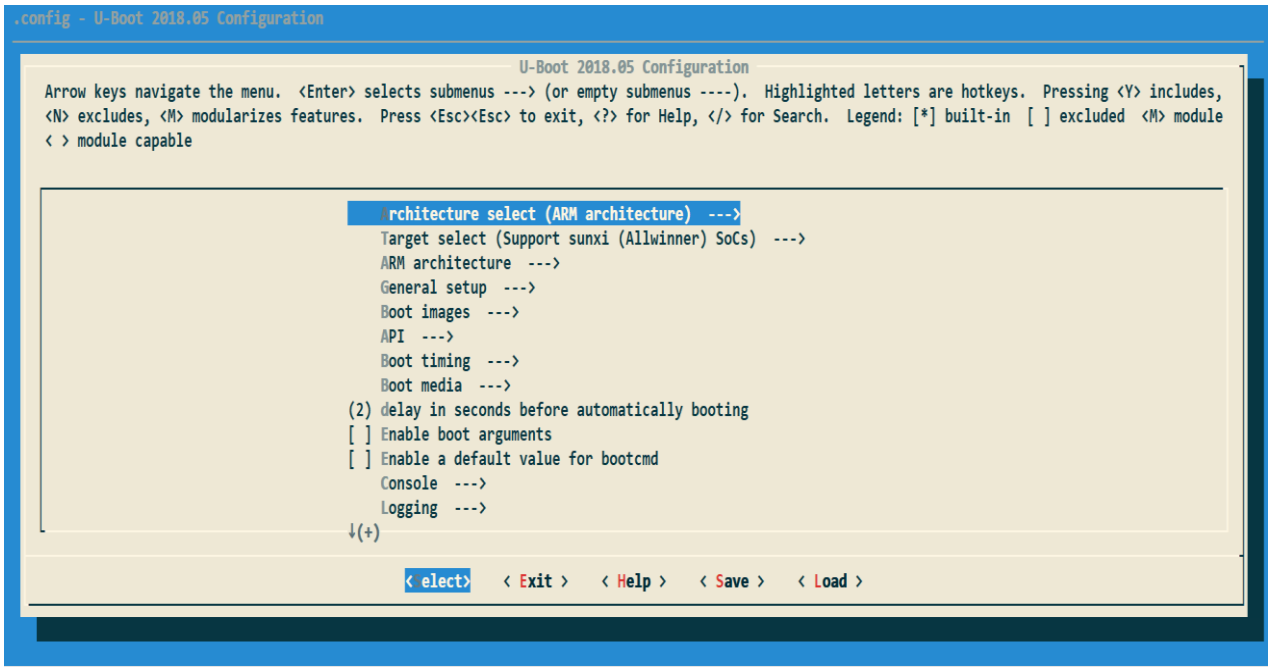

图 3-3. Uboot 图形配置界面

此时用户只需要根据自己实际情况勾选或者取消某个配置选项，即可对 uboot 配置做出对应的修改。

深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

但是此次操作修改的配置只保存在一个中间的临时配置文件中，如果下次加载了其他型号的配置文件，然后又回来加载该配置文件，那么前面做出的修改就会被还原。

➢ 直接修改 defconfig 配置文件（推荐）

用户可以直接修改对应的 defconfig 文件即可对 uboot 配置进行更改，以 MYD-YT113-S3 型号 EMMC 板型为例，需要修改的对应 defconfig 文件为 sun8iw20p1_auto_defconfig，用户只需要打开该文件，根据自己实际情况进行修改即可。该方法可以保存对 uboot 的配置做出的修改，不会因为第二次加载该配置而消失。

MYD-YT113X 所有型号的 defconfig 文件都在下面路径中。

```
PC$: $HOME/T113X/auto-t113x-linux/brandy/brandy-2.0/u-boot-2018/configs$
```

- **编译更新 uboot**

上面已经加载和修改了对应 defconfig 的配置文件，下面就开始编译 uboot

```
PC$: $HOME/T113X/auto-t113x-linux/brandy/brandy-2.0/u-boot-2018$ make
  CHK     include/config/uboot.release
  CHK     include/generated/version_autogenerated.h
  CHK     include/generated/timestamp_autogenerated.h
  CHK     include/generated/generic-asm-offsets.h
  CHK     include/generated/asm-offsets.h
'u-boot.bin' -> 'u-boot-sun8iw20p1.bin'
'u-boot-sun8iw20p1.bin' -> '/media/home/nico/RC/T113-I/SDK/device/config/chips/t113/bin/u-boot-sun8iw20p1.bin'
'u-boot-sun8iw20p1.bin' -> '/media/home/nico/RC/T113-I/SDK/out/t113/myd_yt113_s3_emmc_full/longan/u-boot-sun8iw20p1.bin'
  CHK     include/config.h
  CFG     u-boot.cfg
  CFGCHK  u-boot.cfg
```

编译完成之后需要返回到 SDK 源码顶层目录执行下面命令，将更改后的 uboot 编译文件打包至 out 目录下。

```
PC$: cd $HOME/T113X/auto-t113x-linux$
PC$: $HOME/T113X/auto-t113x-linux$ ./build.sh pack
```


深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

此时在“$HOME/T113X/auto-t113x-linux/out/pack_out”目录下找到 boot_package.fex 文件，将这个文件放入到 U 盘中，将 U 盘插入到开发板中，U 盘会被自动挂载在下面的路径，然后执行下面的命令进行单独更新 uboot。

```
root@myd-yt113-s3:~# cd /mnt/usb/sda1/
root@myd-yt113-s3:/mnt/usb/sda1# ota-burnuboot boot_package.fex
Burn Uboot Success
```

此时 uboot 更新成功，目前单独更新 uboot 方法只适用于 EMMC 板型开发板,nand 开发板暂时不支持单独更新。所以 nand 开发板执行完“./build.sh pack”之后按照 4.1 章节的方法烧录到开发板中，即可更新 uboot。

3.5.2. 在 linux SDK 项目下编译 u-boot(推荐)

当用户按照 3.5.1 中的迭代开发过程改好 U-boot 的代码之后，也可以使用 SDK 进行整个镜像的构建。

不能通过图形界面修改 uboot 配置，只能直接修改 defconfig 文件，然后用此方法编译。

编译 uboot 源码：

```
PC$: $HOME/T113X/auto-t113x-linux$ ./build.sh brandy
```

返回到 SDK 源码顶层目录，打包镜像：

```
PC$: $HOME/T113X/auto-t113x-linux$ ./build.sh pack
```

编译完 uboot 源码，按照 3.5.1 章节最后单独更新 uboot 步骤即可更新 uboot。也可以按照 4.1 章节的方法烧录到到开发板中，更新 uboot。

3.6. 板载 Kernel 编译与更新

Linux kernel 是个十分庞大的开源内核，被应用在各种发行版操作系统上，Linux kernel 以其可移植性，多种网络协议支持，独立的模块机制，MMU 等诸多丰富特性，使 Linux kernel 能在嵌入式系统中被广泛采用。

同时 T1 也支持 Linux 内核，将得到长期稳定的更新，MYD-YT113X 使用 T1 内核移植，最新支持 Linux kernel 5.4.61 版本。

1). 获取 kernel 源代码

深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

拷贝开发包 04_Source/YT113X-buildroot-T1-5.4.61-X.X.X.tar.bz(X.X.X 代表当前版本)到指定自定义的 work 目录（如$HOME/work/T113X），解压进入源码目录并查看对应的文件信息，如拷贝到 work 目录:

```
PC$ cd $HOME/T113X
PC$ $HOME/T113X$ tar -jxf YT113X-buildroot-T1-5.4.61-X.X.X.tar.bz
PC$ $HOME/T113X$ cd auto-t113x-linux/kernel
PC$ $HOME/T113X/auto-t113x-linux/kernel$ tree -L 1
.
└── linux-5.4
```

目录包含：

➢ 源代码：linux-5.4

2). 修改内核配置

米尔已经将大部分功能集成到内核，一般不需要进行配置。如需添加特殊的功能，外设驱动请按照下列方式配置。

- **加载编译链**

按照 2.2 章节配置编译链，然后配置编译链环境。

```
PC$  export PATH=$PATH:/opt/gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabi/bin
```

- **进入内核目录**

```
PC$ cd $HOME/T113X/auto-t113x-linux/kernel/linux-5.4$
```

- **加载 defconfig 配置**

➢ MYD-YT113-S3 型号

```
emmc 板型：
full 系统配置：
PC$ $HOME/T113X/auto-t113x-linux/kernel/linux-5.4$ make myd_yt113s3_emmc_full_defconfig

core 系统配置：
PC$ $HOME/T113X/auto-t113x-linux/kernel/linux-5.4$ make myd_yt113s3_emmc_core_defconfig
```


深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

nand 板型：

PC$ $HOME/T113X/auto-t113x-linux/kernel/linux-5.4$ make myd_yt113s3_nand_defconfig

➢ MYD-YT113-I 型号

full 系统配置：

PC$ $HOME/T113X/auto-t113x-linux/kernel/linux-5.4$ make myd_yt113i_emmc_full_defconfig

core 系统配置：

PC$ $HOME/T113X/auto-t113x-linux/kernel/linux-5.4$ make myd_yt113i_emmc_core_defconfig

注意：加载完对应的 defconfig 文件后会在当前目录生成一个临时的.config 文件，将这个文件复制一份到目录 A 下并命名为 1.con（用户根据自己的情况选择目录），这个后续步骤需要用到。

- **打开内核配置界面**

PC$ $HOME/T113X/auto-t113x-linux/kernel/linux-5.4$ make menuconfig

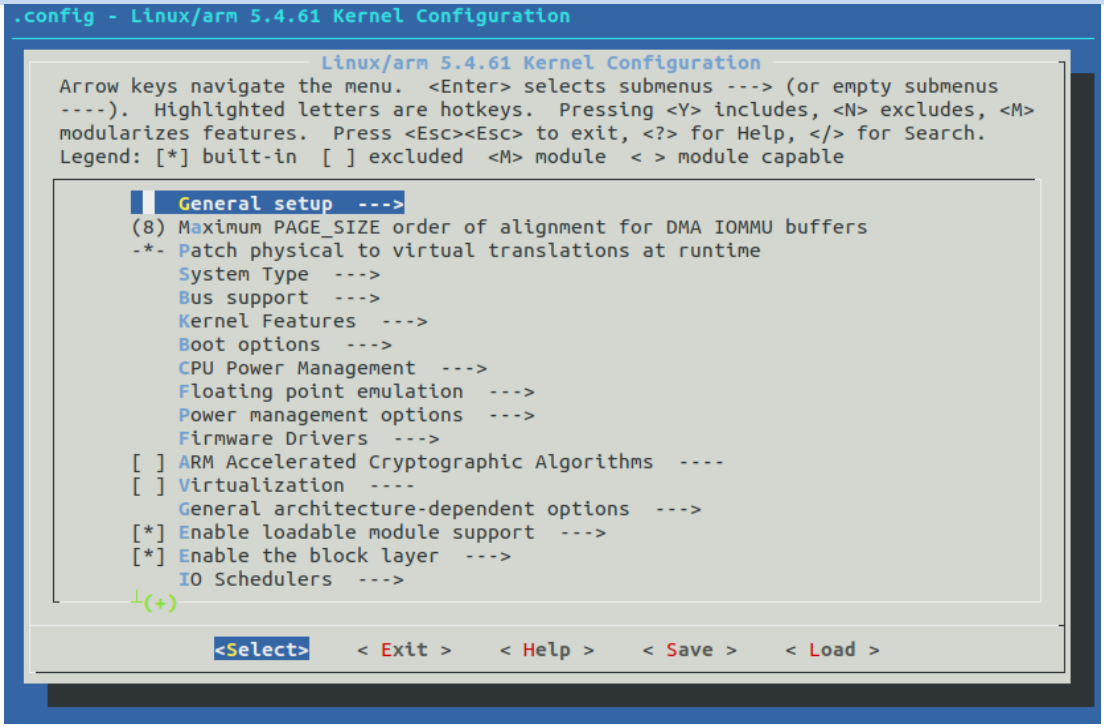

图 3-4. 内核配置界面

打开内核配置界面后，可以添加或某些内核配置即可，本次以添加 System V IPC 为例，该配置在 General setup--->下即可找到，按 y 选中配置，按 n 取消配置，按 m 编译成模块。

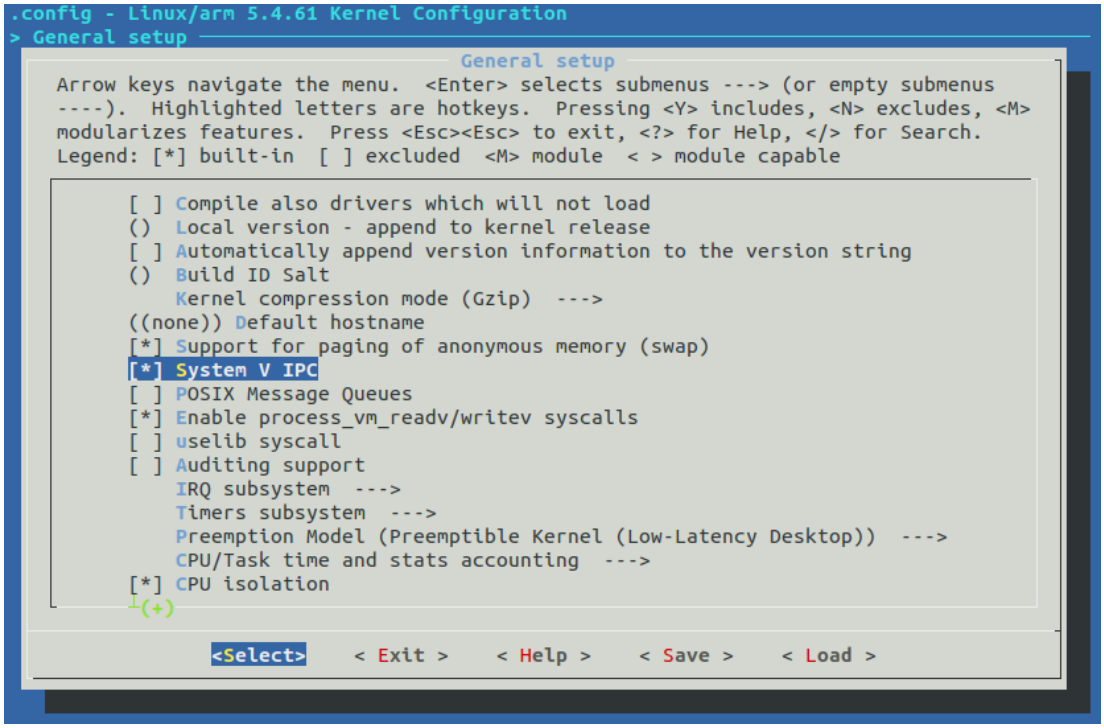

图 3-5. 选中内核配置

注意：添加完配置后，需要在对应的 defconfig 文件做出对应的修改,不然配置无法生效。添加完配置后退出图形界面，此时当前目录下.config 文件保存了刚刚的配置，这个时候将.config 文件也复制到目录 A 下并命名为 2.con，跟上一小节的 1.con 做比较，去修改对应的 defconfig 文件。

MYD-YT113X 内核所有 defconfig 文件都在下面目录

```
PC$: $HOME/T113X/auto-t113x-linux/kernel/linux-5.4/arch/arm/configs$
```

回到 SDK 源码顶层目录，执行下面选择配置、编译、打包镜像命令。

```
PC$: cd $HOME/T113X/auto-t113x-linux$
PC$: $HOME/T113X/auto-t113x-linux$ ./build.sh config（与 3.4 章节第一步配置一致）
PC$: $HOME/T113X/auto-t113x-linux$ ./build.sh kernel
PC$: $HOME/T113X/auto-t113x-linux$ ./build.sh pack
```

此时在“$HOME/T113X/auto-t113x-linux/out/pack_out”目录下找到 boot.fex 文件，将这个文件放入到 U 盘中，将 U 盘插入到开发板中，U 盘会被自动挂载在下面的路径，然后执行下面的命令进行单独更新 kernel。

```
root@myd-yt113-s3:~# dd if=/mnt/usb/sda1/boot.fex of=/dev/mmcblk0p4
41200+0 records in
41200+0 records out
21094400 bytes (21 MB, 20 MiB) copied, 3.51428 s, 6.0 MB/s
```

此时 kernel 更新成功，目前单独更新 kernel 方法只适用于 EMMC 板型开发板,nand 开发板暂时不支持单独更新。所以 nand 开发板执行完“./build.sh pack”之后按照 4.1 章节的方法烧录到开发板中，即可更新 kernel。

3). 更新设备树

板级 dts 与 uboot 使用的 dts 在$HOME/T113X/auto-t113x-linux/device/config/chips/t113/configs/ myir-image-yt113s3-emmc-full 目录下（以 emmc full 镜像配置为例）

```
PC$: cd $HOME/T113X/auto-t113x-linux/device/config/chips/t113/configs/ myir-image-yt113s3-emmc-full$ tree -L 1
.
├── board.dts                    板级 dts
├── bsp
├── linux-5.4
├── longan
├── sys_config.fex
└── uboot-board.dts              emmc 板型 uboot 使用的 dts
```

修改完设备树后，回到 SDK 源码顶层目录，执行下面编译、打包镜像命令。

```
PC$: cd $HOME/T113X/auto-t113x-linux$
PC$: $HOME/T113X/auto-t113x-linux$ ./build.sh kernel
PC$: $HOME/T113X/auto-t113x-linux$ ./build.sh pack
```

目前无法单独更新设备树，所以编译完设备树文件，打包镜像后，按照 4.1 章节的方法烧录到到开发板中，即可更新设备树文件。

4. 如何烧录系统镜像

米尔公司设计的 MYC-YT113X 系列核心板与开发板是基于全志公司的 T1 系列微处理器,其启动方式多样，所以需要不同的更新系统工具与方法。用户可以根据需求选择不同的方式进行更新。更新方式主要有以下几种：

- 制作 SD 卡启动器: 适用于研发调试，快速启动等场景
- 制作 SD 卡烧录器：适用于批量生产烧写 eMMC

4.1. 制作 SD 卡镜像

以下步骤均在 Windows 系统下制作。

- **准备工作**
  - SD 卡 （不少于 8G）
  - MYD-YT113X 开发板
  - 制作镜像工具 PhoenixCard （路径：\03_Tools\myir tools）

表 4-1.镜像包列表

镜像名称	包名	适用核心板
myir-image-yt113s3-emmc-core	myir-image-yt113s3-emmc-core.img	MYC-YT113S3-4E128D
myir-image-yt113s3-emmc-full	myir-image-yt113s3-emmc-full.img	MYC-YT113S3-4E128D
myir-image-yt113s3-nand	myir-image-yt113s3-nand.img	MYC-YT113S3-256N128D
myir-image-yt113i-core	myir-image-yt113i-core.img	MYC-YT113i-4E512D MYC-YT113i-8E1D
myir-image-yt113i-full	myir-image-yt113i-full.img	MYC-YT113i-4E512D MYC-YT113i-8E1D

4.1.1. 制作 SD 卡启动器（以 myir-image-yt113s3-emmc-full 系统为例）

1). 修改配置文件，制作 SD 镜像

目前 SDK 生成的镜像不支持 SD 卡启动，需要修改如下配置：

```
PC$ cd $HOME/T113X/auto-t113x-linux/device/config/chips/t113/configs/myir-image-yt113s3-emmc-full
PC$ $HOME/T113X/auto-t113x-linux/device/config/chips/t113/configs/ myir-image-yt113s3-emmc-full$ vim sys_config.fex
```


深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

```
.....
;----------------------------------------------------------------------------------
;storage_type    = boot medium, 0-nand, 1-sd, 2-emmc, 3-nor, 4-emmc3, 5-spinand -1(defualt)auto scan
;----------------------------------------------------------------------------------
[target]
storage_type     = 1
burn_key         = 0
;----------------------------------------------------------------------------------

PC$ $HOME/T113X/auto-t113x-linux/device/config/chips/t113/configs/ myir-image-yt113s3-emmc-full /longan$ vim


#kernel command arguments
earlycon=uart8250,mmio32,0x02501400
initcall_debug=0
console=ttyAS5,115200
nand_root=ubi0_5
mmc_root=/dev/mmcblk1p5
mtd_name=sys
rootfstype=ubifs,rw
init=/init
loglevel=7
.....
```

注意：nand 镜像不支持 SD 卡启动

2). SD 启动镜像烧录步骤

将用户资料 tools 目录的 PhoenixCard_.zip 拷贝到 windows 任意目录，双击 Phoenix Card 目录里 PhoenixCard.exe 文件。 通过 SD 读卡器把容量大小为 16GB SD card 插入 windows USB 接口上,如下图所示，选择“固件”路径；选择“启动卡”,点击“烧卡”按钮即可自动制作完成。

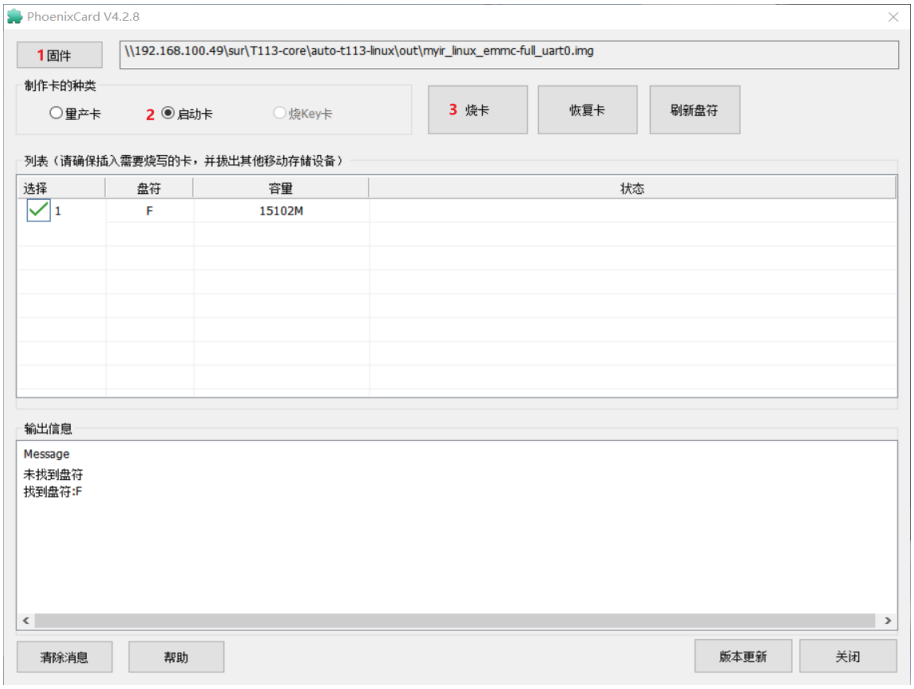

图 4-1. 刷写步骤

下图（图 4-2）显示正在烧卡，预计过程 3-5 分钟完成（时间取决于镜像的大小）

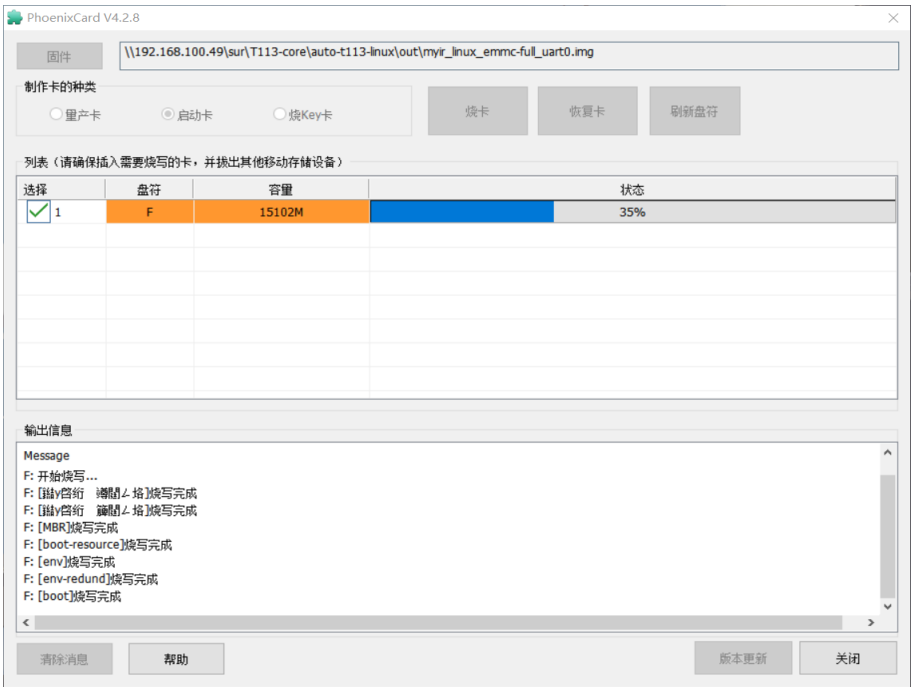

图 4-2. 刷写过程

下图（图 4-3）显示烧卡完成，同时注意输出信息提示烧写完成，此时 SD 卡启动器制作成功，将 SD 卡插入到 emmc 或者 nand 板卡的 SD 卡槽（J5）中，然后上电即可启动开发板。

深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

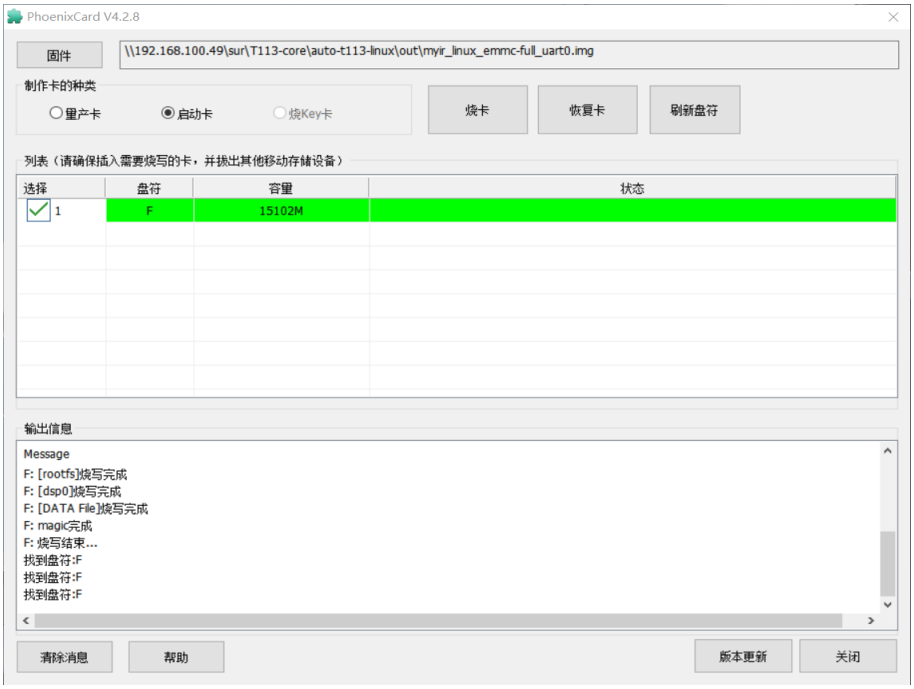

图 4-3. 刷写成功

4.1.2. 制作 SD 卡烧录器

1). 量产卡制作过程

为满足生产烧录的需要，此项方法适用于大批量生产的烧录方法。通过 SD 卡中的系统将需要烧录的系统刷写进板载的 eMMC 或 spi nand 中。具体制作过程请按照下列步骤完成：

准备工作与章节 4.1 一致，操作步骤大同小异。

先在 windowns 下打开 PhoenixCard 程序（程序所在位置在 4.1 章节准备工作有说明）。通过 SD 读卡器把容量大小为 16GB SD card 插入 windows USB 接口上,如下图所示，选择“固件”路径；选择“量产卡”,点击“烧卡”按钮即可自动制作完成。

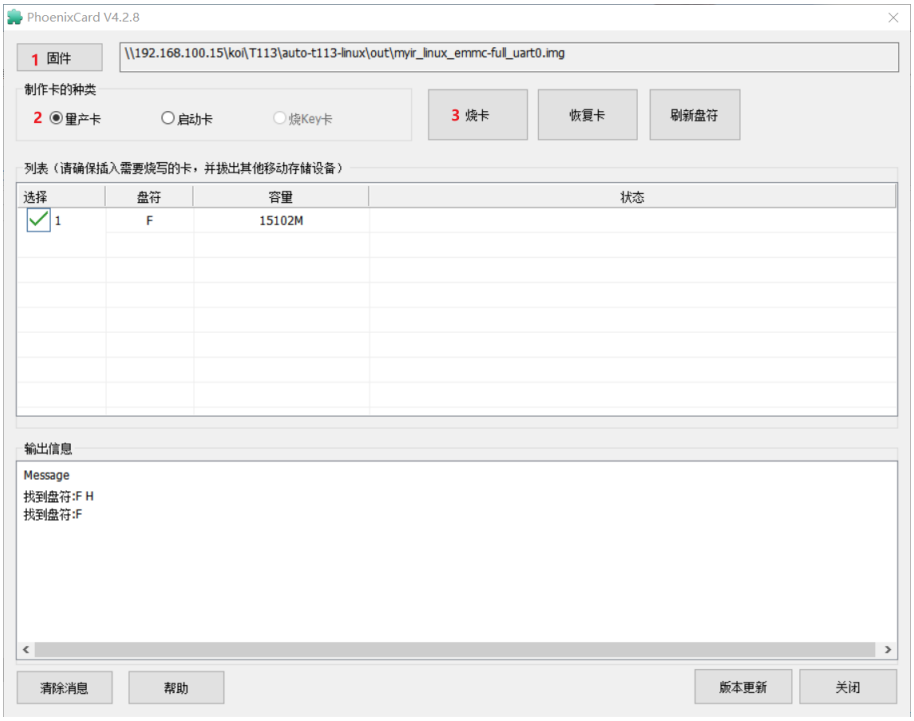

图 4-4. 量产卡制作

后续的制作方法与 4.1.1 章节一致，这里就不在赘述。

2). 验证 eMMC 启动

将制作好的 SD 烧录卡插入到 emmc 或者 nand 板卡的 SD 卡槽（J5）中，然后上电即启动开发板等待烧录打印完成，然后掉电拔下 SD 卡，重新启动开发板即可。

注意：如果烧录完成没有拔下 SD 卡启动了开发板，会重新烧录镜像。

由于烧录打印的信息较多，只截取部分重要数据展示，最后出现 Flash Success 信息表示烧录完成（nand 板卡烧录过程中的信息跟 emmc 板卡不一样，但是最后烧录完成都会出现 Flash Success 信息）。

```
[07.110]begin to download part boot
partdata hi 0x0
partdata lo 0x1423000
sparse: bad magic
[09.469]successed in writting part boot
origin_verify value = 262bacf0, active_verify value = 262bacf0
[09.959]successed in verify part boot
[09.963]successed in download part boot
[09.966]begin to download part rootfs
```

```
partdata hi 0x0
partdata lo 0xf962f40
chunk 0(8515)
chunk 1(8515)
chunk 2(8515)
chunk 3(8515)
chunk 4(8515)
chunk 5(8515)
chunk 6(8515)
chunk 7(8515)
chunk 8(8515)
chunk 9(8515)
chunk 10(8515)
......

[84.409]successed 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
[84.431]sprite success
sprite_next_work=3
next work 3
SUNXI_UPDATE_NEXT_ACTION_SHUTDOWN
sunxi board shutdown
[87.441][mmc]: mmc exit start
[87.459][mmc]: mmc 2 exit ok
*** Flash Success ***
*** Flash Success ***
*** Flash Success ***
```

5. 如何适配您的硬件平台

为了适配用户新的硬件平台，首先需要了解米尔的 MYD-YT113X 开发板提供了哪些资源，具体的信息可以查看《**MYD-YT113X SDK 发布说明**》。除此之外用户还需要对 CPU 的芯片手册，以及 MYC-YT113X 核心板的产品手册，管脚定义有比较详细的了解，以便于根据实际的功能对这些管脚进行正确的配置和使用。

5.1. 如何配置您的 sys_config.fex

sys_config.fex 是全志对 T1 定义的一套功能配置文件，此文件可用于定义各个节点的管脚，属性，电源等，使用户可快速配置资源的功能。为了让用户掌握 sys_config.fex 配置和使用方法。本章将讲解使用方法

sys_config.fex 文件路径：

```
PC$: $HOME/T113X/auto-t113x-linux/device/config/chips/t113(t113_i)/configs/myir-image-xxx/sys_config.fex  (xxx 代表不同的配置)
```

定义属性类方式：

```
; version:版本 1.00
; machine:板级文件名
;------------------------------------------------------------------------------
[product]
version = "100"
machine = "emmc"

;------------------------------------------------------------------------------
;debug_mode     = 0-close printf, > 0-open printf
;------------------------------------------------------------------------------
[platform]
eraseflag   = 1
debug_mode  = 8

;------------------------------------------------------------------------------
;storage_type    = boot medium, 0-nand, 1-sd, 2-emmc, 3-nor, 4-emmc3, 5-spinand -1(defualt)auto scan
```

```
;------------------------------------------------------------------------------
[target]
storage_type    = 2
burn_key        = 0
定义管脚类方式:
[card0_boot_para]
card_ctrl       = 0
card_high_speed = 1
card_line       = 4
sdc_d1          = port:PF0<2><1><2><default>
sdc_d0          = port:PF1<2><1><2><default>
sdc_clk         = port:PF2<2><1><2><default>
sdc_cmd         = port:PF3<2><1><2><default>
sdc_d3          = port:PF4<2><1><2><default>
sdc_d2          = port:PF5<2><1><2><default>
bus-width = 4
cap-sd-highspeed =
cap-wait-while-busy =
no-sdio =
no-mmc =
sunxi-power-save-mode =

;------------------------------------------------------------------------------
[card2_boot_para]
card_ctrl       = 2
card_high_speed = 1
card_line       = 4
sdc_clk         = port:PC02<3><1><3><default>
sdc_cmd         = port:PC03<3><1><3><default>
sdc_d0          = port:PC06<3><1><3><default>
sdc_d1          = port:PC05<3><1><3><default>
sdc_d2          = port:PC04<3><1><3><default>
sdc_d3          = port:PC07<3><1><3><default>
```

```
sdc_tm4_hs200_max_freq = 150
sdc_tm4_hs400_max_freq = 100
sdc_ex_dly_used = 2
;sdc_io_1v8     = 1
sdc_tm4_win_th = 8
sdc_dis_host_caps = 0x180
;sdc_erase      = 2
;sdc_boot0_sup_1v8 = 1
;sdc_type       = "tm4"
```

5.2. 如何创建您的设备树

5.2.1. 板载设备树

用户可以在 BSP 源码里创建自己的设备树，一般情况下不需要修改 Bootloader 部分中的代码。用户只需要根据实际的硬件资源对 Linux 内核设备树进行适当的调整即可。在此将 MYD-YT113X 的 BSP 各个部分中的设备树列表罗列出来，方便用户开发参考，具体内容如下表所示：

表 5-1.MYD-YT113X 设备树列表

项目	设备树	说明
U-boot	uboot-board.dts	uboot 使用的 dts
	myir-t113-lvds.dtsi	7 寸单路 LVDS 设备树配置
	myir-t113-lvds-dual.dtsi	19 寸双路 LVDS 设备树配置
Kernel	board.dts	底板配置资源、管脚资源配置
	sun8iw20p1.dtsi	核心资源配置
	myir-t113-lvds.dtsi	7 寸单路 LVDS 设备树配置
	myir-t113-lvds-dual.dtsi	19 寸双路 LVDS 设备树配置

本 SDK 源码提供 t113 和 t113_i 两个型号 5 种镜像类型配置，它们的使用的 board.dts、uboot-board.dts 分别在以下路径：

```
PC$: $HOME/T113X/auto-t113x-linux/device/config/chips/t113/configs/ myir-image-yt113s3-emmc-core
PC$: $HOME/T113X/auto-t113x-linux/device/config/chips/t113/configs/ myir-image-yt113s3-emmc-full
```


深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

PC$: $HOME/T113X/auto-t113x-linux/device/config/chips/t113/configs/ myir-image-yt113s3-nand

PC$: $HOME/T113X/auto-t113x-linux/device/config/chips/t113_i/configs/ myir-image-yt113i-core

PC$: $HOME/T113X/auto-t113x-linux/device/config/chips/t113_i/configs/ myir-image-yt113i-full

uboot 下使用 LVDS 的设备树配置文件在以下路径：

PC$: $HOME/T113X/auto-t113x-linux/brandy/brandy-2.0/u-boot-2018/arch/arm/dts

kernel 下使用 LVDS 的设备树配置文件在以下路径：

PC$: $HOME/T113X/auto-t113x-linux/kernel/linux-5.4/arch/arm/boot/dts

5.2.2. 设备树的添加

Linux 内核设备树是一种数据结构，它通过特有的语法格式描述片上片外的设备信息。由 BootLoader 传递给 kernel，kernel 进行解析后形成和驱动程序关联的 dev 结构供驱动代码使用，下面将以 kernel 为例介绍如何增加设备树。

在内核源码下$HOME/T113X/auto-t113x-linux/kernel/linux-5.4/arch/arm/boot/dts 下可以看到大量的平台设备树。如适合 MYD-YT113X 的设备树，可在当前路径下增加自定义设备树,如：

PC$: $HOME/T113X/auto-t113x-linux/kernel/linux-5.4/arch/arm/boot/dts

MYC-YT113X 核心板相关的资源编写进 sun8iw20p1.dtsi 以及 board.dts。其它扩展的接口和设备可以对它们进行引用，如下所示（仅供参考）：

如需修改不同显示器配置需修改如下文件(由于只有 full 镜像提供图形显示，因此只修改如下路径的 board.dts 文件)

$HOME/T113X/auto-t113x-linux/device/config/chips/t113/configs/ myir-image-yt113s3-emmc-full/board.dts

board.dts 配置如下：

下面例子是 7 寸 LVDS 显示配置，如果需要更改为 19 寸双路 LVDS 显示，将对应的配置注释打开，然后注释上原来显示的配置，因为不支持同时显示。

```
// $HOME/T113X/auto-t113x-linux/device/config/chips/t113/configs/ myir-image-yt113s3-emmc-full/board.dts

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

/{
	model = "sun8iw20";
	compatible = "allwinner,r528", "arm,sun8iw20p1";

	reg_vdd_cpu: vdd-cpu {
		compatible = "pwm-regulator";
		pwms = <&pwm 3 5000 0>;
		regulator-name = "vdd_cpu";
		regulator-min-microvolt = <810000>;
		regulator-max-microvolt = <1160000>;
		regulator-settling-time-us = <4000>;
		regulator-always-on;
		regulator-boot-on;
		status = "okay";
	};
```

board.dts 修改之后还需要修改同级目录中的 uboot-board.dts 文件，修改方法跟 board.dts 一致。

```
$HOME/T113X/auto-t113x-linux/device/config/chips/t113/configs/ myir-image-yt113s3-emmc-full/uboot-board.dts

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

```
/dts-v1/;

/* optee used 7MB: SHM 2M: OS: 1M: TA:4M*/
/*/memreserve/ 0x41900000 0x00100000;*/
/* DSP used 1MB */
/* /memreserve/ 0x42000000 0x00100000; */

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

最后双路 LVDS 显示还需要修改 env.cfg 文件，在 bootcmd 添加 lvds_if_reg。

```
$HOME/T113X/auto-t113x-linux/device/config/chips/t113/configs/ myir-image-yt113s3-emmc-full/longan/env.cfg

……
boot_dsp0=sunxi_flash read 43000000 ${dsp0_partition};bootr 43000000 0 0
boot_normal=sunxi_flash read 43000000 boot;bootm 43000000
boot_recovery=sunxi_flash read 43000000 recovery;bootm 43000000
boot_fastboot=fastboot
lvds_if_reg=mw.l 0x05461084 0xE0100000

#uboot system env config
bootdelay=3
#default bootcmd, will change at runtime according to key press
#default nand boot
bootcmd=run lvds_if_reg setargs_mmc boot_dsp0 boot_normal
```

5.3. 如何根据您的硬件配置 CPU 功能管脚

实现一个功能引脚的控制是一个较为复杂的系统开发过程之一，其中包含了引脚的配置，驱动的开发，应用的实现等等步骤，本节不具体分析每个部分的开发过程，而是以实例来讲解功能管脚的控制实现。

5.3.1. GPIO 管脚复用

GPIO: General-purpose input/output，通用的输入输出口，在嵌入式设备中是一个十分重要的资源，可以通过它们输出高低电平或者通过它们读入引脚的状态是高电平或是低电平，MYD-YT113X 封装有外设控制器，这些外设控制器与外部设备交户一般是通过控制 GPIO 来实现，而将 GPIO 被外设控制器使用我们称为复用（Alternate Function），给它们赋予了更多复杂的功能，如用户可以通过 GPIO 口和外部硬件进行数据交互(如 UART)，控制硬件工作(如 LED、蜂鸣器等),读取硬件的工作状态信号（如中断信号）等，所以 GPIO 口的使用非常广泛。

1). GPIO 引脚复用 uart4 功能

配置 uart4 功能，需要先找到哪些引脚可以复用成 uart4 功能，引脚之间的复用关系可参考《**MYC-YT113X-PinList**》核心板 PinList 列表，下面将以 uart4 为例介绍如何配置 gpio 管脚。(以 myir-image-yt113s3-emmc-full 镜像配置讲解)

- **引用 uart4 节点**

```
PC$: $HOME/T113X/auto-t113x-linux/device/config/chips/t113/configs/ myir-image-yt113s3-emmc-full/board.dts

&uart4 {
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&uart4_pins_a>;
        pinctrl-1 = <&uart4_pins_b>;
        status = "okay";
};
```

- **查看引脚原理图的连接**

查看底板原理图可知对应核心模组的 109 和 107 脚，如图 5-1：

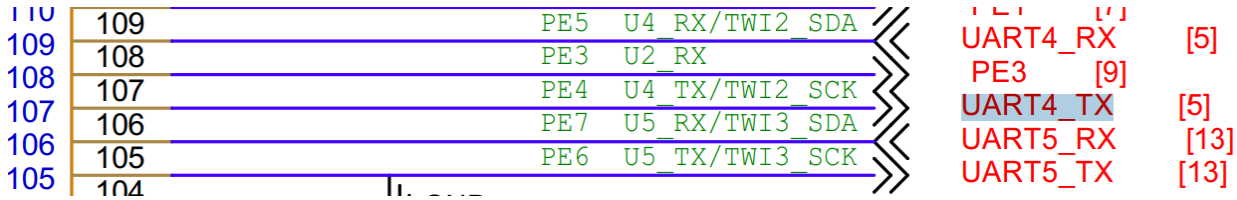

图 5-1. 引脚原理图

- **查看 uart4 核心模组引脚对应关系**

查看核心模组 PinList 的 109 和 107 脚可知对应 PE4 和 PE5，如图 5-2：

深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

MYC Module Pin No	MYC Module Pin Name	PIn Is GPIO	T113-S3 Pin Name	Default function	IO Voltage
97	GND	/	/	/	0V
98	PE2	PE2	PE2/NCSI0_PCLK/UART2_TX/TWI0_SCK/CLK_FANOUT0/UART0_TX/RGMII_RXD1/RMII_RXD1/PE_EINT2		0/3.3V
99	PE12	PE12	PE12/TWI2_SCK/NCSI0_FIELD/I2S0_DOUT2/I2S0_DIN2/RGMII_TXD3/PE_EINT12		0/3.3V
100	NC	/	/	/	3.3V
101	PE8	PE8	PE8/NCSI0_D4/UART1_RTS/PWM2/UART3_TX/JTAG_MS/MDC/PE_EINT8		0/3.3V
102	PE9	PE9	PE9/NCSI0_D5/UART1_CTS/PWM3/UART3_RX/JTAG_DI/MDIO/PE_EINT9		0/3.3V
103	NC	/	/	/	/
104	GND	/	/	/	0V
105	PE6	PE6	PE6/NCSI0_D2/UART5_TX/TWI3_SCK/SPDIF_IN/D_JTAG_DO/R_JTAG_DO/RGMII_TXCTRL/RMII_TXEN/PE_EINT6		0/3.3V
106	PE7	PE7	PE7/NCSI0_D3/UART5_RX/TWI3_SDA/SPDIF_OUT/D_JTAG_CK/R_JTAG_CK/RGMII_CLKIN/RMII_RXER/PE_EINT7		0/3.3V
107	PE4	PE4	PE4/NCSI0_D0/UART4_TX/TWI2_SCK/CLK_FANOUT2/D_JTAG_MS/R_JTAG_MS/RGMII_TXD0/RMII_TXD0/PE_EINT4		0/3.3V
108	PE3	PE3	PE3/NCSI0_MCLK/UART2_RX/TWI0_SDA/CLK_FANOUT1/UART0_RX/RGMII_TXCK/RMII_TXCK/PE_EINT3		0/3.3V
109	PE5	PE5	PE5/NCSI0_D1/UART4_RX/TWI2_SDA/LEDC_DO/D_JTAG_DI/R_JTAG_DI/RGMII_TXD1/RMII_TXD1/PE_EINT5		0/3.3V
110	PE1	PE1	PE1/NCSI0_VSYNC/UART2_CTS/TWI1_SDA/LCD0_VSYNC/RGMII_RXD0/RMII_RXD0/PE_EINT1		0/3.3V
111	PE0	PE0	PE0/NCSI0_HSYNC/UART2_RTS/TWI1_SCK/LCD0_HSYNC/RGMII_RXCTRL/RMII_CRS_DV/PE_EINT0		0/3.3V

图 5-2. 引脚对应关系

- **配置复用关系**

由图 5-2 可知，可以看到 PE4、PE5 引脚可以配置成 UART4_TX、UART4_RX。

```
PC$: $HOME/T113X/auto-t113x-linux/device/config/chips/t113/configs/ myir-image-yt113s3-emmc-full/board.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";
    };
```

- **新增串口别名**

此时需要去下面的路径添加串口别名

```
PC$: $HOME/T113X/auto-t113x-linux/kernel/linux-5.4/arch/arm/boot/dts/sun8iw20p1.dtsi
/ {
    model = "sun8iw20";
```


深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

```
        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;
};
```

最后按照 3.6 章第三小节更新设备树，将新的设备树烧录到开发板即可。

5.3.2. 配置功能管脚为 GPIO 功能实例

此实例使用 PD20 作为测试 GPIO。介绍如何在设备树里配置设备节点，并为后面章节供内核驱动使用。此示例还可以给控制外部设备的复位，电源等控制功能提供参考。(以 emmc full 镜像为例)

只需在设备树里增加节点即可。

```
PC$: $HOME/T113X/auto-t113x-linux/device/config/chips/t113/configs/ myir-image-yt113s3-emmc-full/board.dts

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

                };
```

5.3.3. 开发板 LCD 资源重新分配实例

MYD-YT113X 开发板定义和实现的众多丰富的功能，但同时也占有了大量的管脚资源，如用户直接使用 MYD-YT113X 基础上进行设计开发，将需要对管脚进行重新定义和配置。下列就以 LCD 复用管脚功能为例,复用关系查看《**MYC-YT113X-PinList**》文档。

看到下面目录的 dts 文件中，可以知道 lvds0 功能占用了 PD0~10 的引脚，要想自由分配使用这些引脚，首先将这些引脚释放。

```
PC$: $HOME/T113X/auto-t113x-linux/device/config/chips/t113/configs/ myir-image-yt113s3-emmc-full /board.dts
修改前:
/*
 * Allwinner Technology CO., Ltd.
 */

/dts-v1/;

/* optee used 7MB: SHM 2M: OS: 1M: TA:4M*/
/*/memreserve/ 0x41900000 0x00100000;*/
/* DSP used 1MB */
/* /memreserve/ 0x42000000 0x00100000; */

#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", "PD
7", "PD8", "PD9";
            allwinner,function = "lvds0_suspend";//io_disabled
            allwinner,muxsel = <7>;
            allwinner,drive = <3>;
            allwinner,pull = <0>;
        };
```

以下是修改后展示,这里需要注意的是，要将显示的引用注释掉（#include "myir-t113-lvds.dtsi"）。

注意：在 sun8iw20p1.dtsi 和同级目录下的 uboot-board.dts 也引用了这些引脚，所以也要将它们这样修改，这里就不展示了。

修改后：

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

/dts-v1/;

/* optee used 7MB: SHM 2M: OS: 1M: TA:4M*/
/*/memreserve/ 0x41900000 0x00100000;*/
/* DSP used 1MB */
/* /memreserve/ 0x42000000 0x00100000; */
```

```
#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>;
};
```

5.4. 如何使用自己配置的管脚

在 u-boot 或 Kernel 的设备树中配置后的管脚，可以在 Kernel 中进行使用，从而实现对管脚的控制。

5.4.1. 内核驱动中使用 GPIO 管脚

- **独立 IO 驱动的使用**

在 5.3.2 节中的第一个设备树示例中，已经定义完成了 gpio 节点信息，下面将使用内核驱动来实现 GPIO 的控制（对 PD20 管脚进行置 1 与置 0，如需检测需使用万用表测试管脚电平的变化）。

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

/* 1. 确定主设备号  */
static int major = 0;
static struct class *gpioctr_class;
static struct gpio_desc *gpioctr_gpio;


/* 2. 实现对应的 open/read/write 等函数，填入 file_operations 结构体*/
static ssize_t gpio_drv_read (struct file *file, char __user *buf, size_t size, loff_t *offs
et)
{
        printk("%s %s line %d\n", __FILE__, __FUNCTION__, __LINE__);
        return 0;
}
```

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

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

        gpiod_set_value(gpioctr_gpio, status);

        return 1;
}

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

        return 0;
}

static int gpio_drv_close (struct inode *node, struct file *file)
{
        printk("%s %s line %d\n", __FILE__, __FUNCTION__, __LINE__);
        return 0;
}

/* 定义自己的 file_operations 结构体*/
static struct file_operations gpioctr_drv = {
        .owner      = THIS_MODULE,
        .open    = gpio_drv_open,
        .read    = gpio_drv_read,
```

```
        .write   = gpio_drv_write,
        .release = gpio_drv_close,
};

/* 从 platform_device 获得 GPIO
 *   把 file_operations 结构体告诉内核：注册驱动程序
 */
static int chip_demo_gpio_probe(struct platform_device *pdev)
{
        /*  设备树中定义有: gpioctr-gpios=<...>;     */
        gpioctr_gpio = gpiod_get(&pdev->dev, "gpioctr", 0);
        if (IS_ERR(gpioctr_gpio)) {
                dev_err(&pdev->dev, "Failed to get GPIO for led\n");
                return PTR_ERR(gpioctr_gpio);
        }

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

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

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

    return 0;

}
```

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

    return 0;
}

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

/* 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,
    },
};

/* 在入口函数注册 platform_driver */
static int __init gpio_init(void)
{
    int err;
    err = platform_driver_register(&chip_demo_gpio_driver);

        return err;
```

```
}

/* 有入口函数就应该有出口函数：卸载驱动程序时，就会去调用这个出口函数
 *     卸载 platform_driver
 */
static void __exit gpio_exit(void)
{
    platform_driver_unregister(&chip_demo_gpio_driver);
}

/* 其他完善：提供设备信息，自动创建设备节点 */
module_init(gpio_init);
module_exit(gpio_exit);

MODULE_LICENSE("GPL");
```

将驱动程序代码使用单独的 Makefile 编译成模块也可以直接配置进内核。

- **驱动示例将直接配置进内核**

在内核源代码的/drivers/char 文件夹下新建 gpioctr.c 文件，将上述驱动代码拷贝进去，并修改 Kconfig 与 Makefile 及 defconfig。

在/drivers/char/Kconfig 文件中添加：

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

在/drivers/char/Makefile 文件中添加：

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

在 arch/arm/configs/myd_yt113s3_emmc_full_defconfig 文件中添加：

```
CONFIG_SAMPLE_GPIO=y
```

按照 3.6 章节更新内核，然后烧录到开发板。

- **驱动示例编译成单独模块**

在工作目录下增加 gpioctr.c 并拷贝上述驱动代码，同目录下编写独立 Makefile 程序。

深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

```
PC$: $HOME/gpioctr$ ls
gpioctr.c  Makefile
PC$: $HOME/gpioctr$ vi Makefile
# 修改 KERN_DIR
#KERN_DIR = # 板子所用内核源码的目录
KERN_DIR = $HOME/T113X/auto-t113x-linux/kernel/linux-5.4/

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

# 要想把 a.c, b.c 编译成 ab.ko, 可以这样指定:
# ab-y := a.o b.o
# obj-m += ab.o
```

加载 SDK 环境变量到当前 shell。(根据自己实际安装编译链目录加载)

```
PC$ export PATH=$PATH:/opt/gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabi/bin
```

执行 make 命令，即可生成 gpioctr.ko 驱动模块文件。

```
PC$: $HOME/gpioctr$ make
make -C $HOME/T113X/auto-t113x-linux/kernel/linux-5.4/ M=`pwd` modules
make[1]: Entering directory '$HOME/T113X/auto-t113x-linux/kernel/linux-5.4/'
  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/T113X/auto-t113x-linux/kernel/linux-5.4/'
PC$: $HOME/gpioctr$ ls
```

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

编译成功之后，将 gpioctr.ko 文件可通过以太网，WIFI, U 盘等传输介质传输到开发板即可使用 insmod 命令加载驱动。

不同的外部设备各自具有独立的驱动代码和架构实现，在对不同外设驱动修改，调试时，需要遵守各自的驱动框架。如触摸屏，键盘等需要使用 input 驱动架构；ADC 与 DAC 使用 IIO 架构，显示设备使用 DRM 驱动架构等等，本节不对所有驱动开发做具体的讲解。

5.4.2. 用户空间使用 GPIO 管脚

Linux 操作系统的体系架构分为用户态和内核态（或者用户空间和内核）。用户态即上层应用程序的活动空间，应用程序的执行必须依托于内核提供的资源，包括 CPU 资源、存储资源、I/O 资源等。为了使上层应用能够访问到这些资源，内核必须为上层应用提供访问的接口：即系统调用。

Shell 是一个特殊的应用程序，俗称命令行，本质上是一个命令解释器，它下通系统调用，上通各种应用。使用 Shell 脚本，通常短短的几行 Shell 脚本就可以实现一个非常大的功能，原因就是这些 Shell 语句通常都对系统调用做了一层封装。为了方便用户和系统交互。

本节将讲述在用户态如何使用 GPIO 管脚的控制三种基本方式。

- Shell 命令
- 系统调用
- 库函数

- **Shell 实现管脚控制**

Shell 控制管脚实质上是调用 Linux 提供的文件操作接口俩实现的。使用调试接口来配置，每个 gpio 的 PIN 都有四个属性，分别是复用（function），数据（data），驱动能力（dlevel），上下拉状态（pull）。操作方法如下:

```
[root@myir:/]# mount -t debugfs none /sys/kernel/debug
[root@myir:/]# cd /sys/kernel/debug/sunxi_pinctrl
```

查看 pin 的配置:

```
[root@myir:/]# echo PD20 > sunxi_pin
[root@myir:/]# cat sunxi_pin_configure
```

```
pin[PD20] funciton: f
pin[PD20] data: 0
pin[PD20] dlevel: 20mA
pin[PD20] pull up: 0xffffff
pin[PD20] pull down: 0xffffff
pin[PD20] pull disable: 0x0
```

修改 pin 属性：

```
[root@myir:/]# echo PD20 1 > pull
[root@myir:/]# 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: 0xffffff
pin[PD20] pull disable: 0xffffff
```

- **库函数实现管脚控制**

从 Linux 4.8 版本开始，Linux 引入了新的 gpio 操作方式，GPIO 字符设备。不再使用以前 SYSFS 方式在"/sys/class/gpio"目录下来操作 GPIO，而是，基于"文件描述符"的字符设备，每个 GPIO 组在"/dev"下有一个对应的 gpiochip 文件，例如"/dev/gpiochip0 对应 GPIOA, /dev/gpiochip1 对应 GPIOB"等等。

Libgpiod 库函数实现由于 gpiochip 的方式，基于 C 语言，所以开发者实现了 Libgpiod，提供了一些工具和更简易的 C API 接口。Libgpiod (Library General Purpose Input/Output device) 提供了完整的 API 给开发者，同时还提供了一些用户空间下的应用来操作 GPIO。

Libgpiod 常用基本接口描述：

- gpiodetect - 列出系统中出现的所有 gpiochip，它们的名称，标签和 GPIO 行数。
- gpioinfo - 列出指定的 gpiochips 的所有行、它们的名称、使用者、方向、活动状态和附加标志。
- gpioget -读取指定的 GPIO 行值。

深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

➢ gpioset - 设置指定的 GPIO 行值，潜在地保持这些行导出并等待超时、用户输入或信号。

➢ gpiofind - 查找给定行名称的 gpiochip 名称和行偏移量。

➢ gpiomon - 等待 GPIO 行上的事件，指定要观察哪些事件，退出前要处理多少事件，或者是否应该将事件报告到控制台。

更多描述，可查看 libgpiod 源代码 https://git.kernel.org/pub/scm/libs/libgpiod/libgpiod.git/。

下列将以 PD20 做为操作 GPIO 管脚来实现 C 语言的代码控制实例(交替置高置低)。

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

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

        strcpy(chrdev_name, "/dev/gpiochip0");

        /* Open device: gpiochip5 for GPIO bank F */
        fd = open(chrdev_name, 0);
        if (fd == -1) {
                ret = -errno;
                fprintf(stderr, "Failed to open %s\n", chrdev_name);
```

```
return ret;
}

/* request GPIO line: P4_1 */
req.lineoffsets[0] = 33;
req.flags = GPIOHANDLE_REQUEST_OUTPUT;
memcpy(req.default_values, &data, sizeof(req.default_values));
strcpy(req.consumer_label, "P4_1");
req.lines = 1;

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

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

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

```
            ret = -errno;
        }
        return ret;
```

将上述代码拷贝到一个 gpioctr-test.c 文件下，加载 SDK 环境变量到当前 shell:

```
PC$ export PATH=$PATH:/opt/gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabi/bin
```

使用编译命令$CC 可生成可执行文件 gpioctr。

```
PC$ $HOME/gpioctr$ $CC gpioctr-test.c -o gpioctr-test
```

将可执行文件通过网络（scp 等），u 盘等传输介质拷贝到开发板的/usr/sbin 目录下，即可在终端下输入命令可直接运行，用万用表可以看到 P4_1 引脚高低电平交替变化。

```
[root@myir:/]# ./gpioctr-test
```

- **系统调用实现管脚控制**

操作系统提供给用户程序调用的一组“特殊”接口。用户程序可以通过这组“特殊”接口来获得操作系统内核提供的服务，比如用户可以通过文件系统相关的调用请求系统打开文件、关闭文件或读写文件，可以通过时钟相关的系统调用获得系统时间或设置定时器等。

同时管脚也是资源，也可以通过系统调用的方式实现控制。在 5.3.2 中我们已经完成了管脚的驱动的实现，即可对该驱动程序所控制的管脚进行系统调用控制。

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

/*
 * ./gpiotest /dev/myir_gpioctr0 on
 * ./gpiotest /dev/myir_gpioctr0 off
 */
```

```
int main(int argc, char **argv)
{
	int fd;
	char status;

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

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

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

	close(fd);
```


深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

```
        return 0;
}
```

将上述代码拷贝到一个 gpiotest-API.c 文件下，加载 SDK 环境变量到当前 shell:

```
PC$ export PATH=$PATH:/opt/gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabi/bin
```

使用编译命令$CC 可生成可执行文件 gpiotest。

```
PC$ $HOME/gpioctr$ $CC gpiotest-API.c -o gpiotest-API
```

将可执行文件通过网络（scp 等），u 盘等传输介质拷贝到开发板的/usr/sbin 目录下，即可在终端下输入命令可直接运行(on 表示置高，off 表示置低)。

```
[root@myir:/]# ./gpiotest-API /dev/myir_gpioctr0 on
[root@myir:/]# ./gpiotest-API /dev/myir_gpioctr0 off
```

6. 如何添加您的应用

Linux 应用的移植通常分为两个阶段，开发调试阶段和生产部署阶段。开发调试阶段我们可以使用米尔构建好的 SDK 对我们编写好的应用进行交叉编译然后远程拷贝到目标主机进行测试，生产部署阶段需要为应用编写配方文件，并使用 Buildroot 构建生产镜像。

6.1. 基于 Makefile 的应用

Makefile 其实就是一个文档，里面定义了一系列的编译规则，它记录了原始码如何编译的详细信息！ Makefile 一旦写好，只需要一个 make 命令，整个工程完全自动编译，极大的提高了软件开发的效率。在开发 Linux 程序时，不管是内核，驱动，应用，Makefile 得到了普遍的应用。

make 是一个命令工具，是一个解释 makefile 中指令的命令工具。它可以简化编译过程里面所下达的指令，当执行 make 时，make 会在当前的目录下搜寻 Makefile (or makefile) 这个文本文件，执行对应的操作。make 会自动的判别原始文件是否经过了变动，从而自动重新编译更改的源代码。

下列将以一个实际的示例（在 MYD-YT113X 开发板上实现按键控制 LED 灯开关）来讲述 Makfile 的编写与 make 的执行过程。Makefile 有其自身的一套规则。

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

- target 可以是一个 object file(目标文件)，也可以是一个执行文件，还可以是一个标签（label）。
- prerequisites 就是要生成那个 target 所需要的文件或是目标。
- command 也就是 make 需要执行的命令。

```
sur@ubuntu:~/key-led$ vi Makefile
TARGET = $(notdir $(CURDIR))
objs := $(patsubst %c, %o, $(shell ls *.c))
$(TARGET)_test:$(objs)
        $(CC) -o $@ $^
%.o:%.c
        $(CC) -c -o $@ $<
clean:
```


深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

```
	rm -f  $(TARGET)_test *.all *.o
	${CC} -I . -c key_led.c
```

- $(notdir $(path))： 表示把 path 目录去掉路径名，只留当前目录名，比如当前 Makefile 目录为*/home/sur/key_led*，执行为就变为 TARGET = key_led
- $(patsubst pattern, replacement,text) ： 用 replacement 替换 text 中符合格式"pattern" 的字符，如$(patsubst %c, %o, $(shell ls *.c))，表示先列出当前目录后缀为.c 的文件，然后换成后缀为.o
- CC：C 编译器的名称
- CXX: C++编译器的名称
- clean: 是一个约定的目标

Key_led 实现代码如下：

```
sur@ubuntu:~/key-led$ vi key_led.c
/File: Key_led.c
#include <linux/input.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
/* ./key_led /dev/input/event0 noblock */
int main(int argc, char **argv)
{
	int fd,bg_fd;
	int err, len, i;
	unsigned char flag;
	unsigned int data[1];
	char *bg = "/sys/class/leds/LED0/trigger";
	struct input_event event;
```

```
if (argc < 2)
{
        printf("Usage: %s <dev> [noblock]\n", argv[0]);
        return -1;
}
if (argc == 3 && !strcmp(argv[2], "noblock"))
{
        fd = open(argv[1], O_RDWR | O_NONBLOCK);
}
else
{
        fd = open(argv[1], O_RDWR);
}
if (fd < 0)
{
        printf("open %s err\n", argv[1]);
        return -1;
}
while (1)
{
        len = read(fd, &event, sizeof(event));
        if (event.type == EV_KEY)
        {
                if (event.value == 1)//key down and up
                {
                        printf("key test \n");
                        bg_fd = open(bg, O_RDWR);
                        if (bg_fd < 0)
                        {
                                printf("open %d err\n", bg_fd);
                                return -1;
                        }
                        read(bg_fd,&flag,1);
```

```
                    printf("flag =%d\n",flag);
                    if(flag == '0')
                    {
                        system("echo heartbeat > /sys/class/leds/LED0/trigger");
//led off
                        //system("echo 0 > /sys/class/leds/LED0/brightness"); //l
ed off
                    }
                    else
                    {
                        system("echo none > /sys/class/leds/LED0/trigger"); //led
 off
                        sleep(3);
                        system("echo heartbeat > /sys/class/leds/LED0/trigger");
                    }
                }
            }
        }
        return 0;
```

使用 make 命令进行编译并生成目标机器上的可执行文件。

加载 SDK 环境变量到当前 shell:

```
PC$ export PATH=$PATH:/opt/gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueab
i/bin
```

执行 make:

```
PC$ $HOME/key-led$ make
PC$ $HOME/key-led$ ls
key_led.c  key_led.o  key-led_test  Makefile
```

从上一个命令的结果可以看到，使用的编译器是通过设置脚本中定义的 CC 变量建立的编译器,将 key-led_test 可执行文件通过网络（scp 等），u 盘等传输介质拷贝到开发板的/usr/sbin 目录下:

```
[root@myir:/]# key-led_test /dev/input/event0
```

说明：如果使用交叉工具链编译器构建目标可执行文件，并且构建主机的体系结构与目标机器的体系结构不同，则需要在目标设备上运行项目。

6.2. 基于 Qt 的应用

Qt 是一个跨平台的图形应用开发框架，被应用在不同尺寸设备和平台上，同时提供不同版权版本供用户选择。 MYD-YT113X 使用 Qt 5.12 版本进行应用开发。在 Qt 应用开发中，推荐使用 QtCreator 集成开发环境，可以在 Linux PC 下开发 Qt 应用，自动化地交叉编译为开发板的 ARM 架构程序。

6.2.1. QtCreator 安装与配置

从 QT 官网或 MYIR 官方包获得 qtcreator 安装包 QT 官网下载：https://download.qt.io/development_releases/qtcreator/。

QtCreator 安装包是一个二进制程序，直接执行就可以完成安装 ./qt-creator-opensource-linux-x86_64-5.0.0-rc1.run 即可，如需获得安装与配置详情请查看《**MYD-YT113X QT** 应用笔记开发》或从 QTCreator 官方网站获得更多开发指导 https://www.qt.io/product/development-tools。

6.2.2. MEasy HMI2.x 编译和运行

MEasy HMI 2.x 是深圳市米尔电子有限公司开发的一套基于 QT5 的人机界面框架。项目采用 QML 与 C++混合编程，使用 QML 高效便捷地构建 UI，而 C++则用来实现业务逻辑和复杂算法。

在米尔的软件发布包里可获得 MEasy HMI2.x 项目源代码“MYD-YT113X-2023xxx\04_Sources\mxapp2.tar.gz”。可通过 Qtcreator 进行加载编译，远程调试等，可参看《**MYD-YT113X QT** 应用笔记开发》。

深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

7. 参考资料

- **Linux kernel 开源社区**

  https://www.kernel.org/

- **Buildroot 官网**

  https://buildroot.org/

附录一 联系我们

深圳总部

地址：深圳市龙岗区坂田街道发达路云里智能园 2 栋 6 楼 04 室

负责区域：广东、广西、海南、重庆、云南、贵州、四川、西藏、香港、澳门

传真：0755-25532724

电话：0755-25622735

武汉研发中心

地址：武汉东湖新技术开发区关南园一路 20 号当代科技园 4 号楼 1601 号

电话：027-59621648

华东地区

地址：上海市浦东新区金吉路 778 号浦发江程广场 1 号楼 805 室

负责区域：上海、福建、浙江、江苏、安徽、山东

传真：021-62087085

电话：021-62087019

华北地区

地址：北京市大兴区荣华中路 8 号院力宝广场 10 号楼 901 室

负责区域：辽宁、吉林、黑龙江、北京、天津、河北、山西、内蒙古、湖北、湖南、江西、河南、陕西、甘肃、宁夏、青海、新疆

传真：010-64125474
电话：010-84675491

销售联系方式

网址：www.myir.cn
邮箱：sales.cn@myir.cn

技术支持联系方式

邮箱：support.cn@myir.cn
武汉研发中心电话：027-59621648
深圳总部技术电话：0755-22316235

如果您通过邮件获取帮助时，请使用以下格式书写邮件标题：

[公司名称/个人--开发板型号] 问题概述

这样可以使我们更快速跟进您的问题，以便相应开发组可以处理您的问题。

深圳市米尔电子有限公司 MYIR Electronics Limited
Web: www.myir.cn Mail: sales.cn@myir.cn Tel: 0755-22316235

附录二 售后服务与技术支持

凡是通过米尔电子直接购买或经米尔电子授权的正规代理商处购买的米尔电子全系列产品，均可享受以下权益：

1、6 个月免费保修服务周期
2、终身免费技术支持服务
3、终身维修服务
4、免费享有所购买产品配套的软件升级服务
5、免费享有所购买产品配套的软件源代码，以及米尔电子开发的部分软件源代码
6、可直接从米尔电子购买主要芯片样品，简单、方便、快速；免去从代理商处购买时，漫长的等待周期
7、自购买之日起，即成为米尔电子永久客户，享有再次购买米尔电子任何一款软硬件产品的优惠政策
8、OEM/ODM 服务

如有以下情况之一，则不享有免费保修服务：
1、超过免费保修服务周期
2、无产品序列号或无产品有效购买单据
3、进液、受潮、发霉或腐蚀
4、受撞击、挤压、摔落、刮伤等非产品本身质量问题引起的故障和损坏
5、擅自改造硬件、错误上电、错误操作造成的故障和损坏
6、由不可抗拒自然因素引起的故障和损坏

产品返修：

用户在使用过程中由于产品故障、损坏或其他异常现象，在寄回维修之前，请先致电米尔科技客服部，与工程师进行沟通以确认问题，避免故障判断错误造成不必要的运费损失及周期的耽误。

维修周期：

收到返修产品后，我们将即日安排工程师进行检测，我们将在最短的时间内维修或更换并寄回。一般的故障维修周期为 3 个工作日（自我司收到物品之日起，不计运输过程时间），由于特殊故障导致无法短期内维修的产品，我们会与用户另行沟通并确认维修周期。

维修费用：

在免费保修期内的产品，由于产品质量问题引起的故障，不收任何维修费用；不属于免费保修范围内的故障或损坏，在检测确认问题后，我们将与客户沟通并确认维修费用，我们仅收取元器件

材料费，不收取维修服务费；超过保修期限的产品，根据实际损坏的程度来确定收取的元器件材料费和维修服务费。

运输费用：

产品正常保修时，用户寄回的运费由用户承担，维修后寄回给用户的费用由我司承担。非正常保修产品来回运费均由用户承担。