MYD-YT507H Linux 系统开发指南

文件状态： [ ] 草稿 [√] 正式发布	**文件标识：**	MYIR-MYD-YT507H-V1-SW-DG-ZH-L4.9.170
	当前版本：	V1.3[文档]
	作　　者：	Licy
	创建日期：	2022-03-22
	最近更新：	2022-08-22

版 本 历 史

版本	作者	参与者	日期	备注
V1.0[文档]	Licy		20220322	初始版本
V1.1[文档]	Licy		20220707	修改部分配置更新 编译器的说明
V1.2[文档]	Licy		20220803	增加搭建开发环境中需要注意的事项和问题分析 删除 3.3 章节
V1.3[文档]	Licy	Nico	20220822	修改显示设备树描述不清问题

目 录

1. 概述

Linux 系统平台上有许多开源的系统构建框架，这些框架方便了开发者进行嵌入式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 T507-H 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 Rootfs 等。我们首先介绍如何基于我们提供的源代码构建适用于 MYD-YT507H 开发板的系统镜像，如何将构建好的镜像烧录到开发板。针对那些基于 MYC-YT507H 核心板进行项目开发的用户，我们重点介绍了将这一套系统移植到用户的硬件平台上的方法和一些要点，并通过一些实际的 BSP 移植案例和 Rootfs 定制的案例，使用户能够迅速定制适合自己硬件的系统镜像。

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-YT507H SDK 发布说明**》表 2-4 中的文档列表。

1.1. 软件资源

MYD-YT507H 搭载基于 Linux 4.9.170 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-YT507H 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-YT507H 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-YT507H 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 T5 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 T5 系列处理器是 SMP 多核架构的处理器，4 个 ARM Cortex A53 核心，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 Ubuntu20.04 64bit 桌面版系统（Ubuntu18.04 64bit 和 Ubuntu21.04 64bitj 均可使用），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

```
sudo apt-get update
sudo apt-get install build-essential gcc libncurses5-dev bison flex texinfo
sudo apt-get install zlib1g-dev gettext libssl-dev autoconf
sudo apt-get install autoconf
sudo apt-get install automake
sudo apt-get install libtool
sudo apt-get install linux-libc-dev:i386
sudo apt-get install git
sudo apt-get install gnupg
```

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

上述的包需要用户手动安装，如需全部自动安装可以将全部的命令复制到一个 sh 脚本里，直接在后续解压的包里执行./build/config.sh。

其他非必须配置包

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

2.2. 软件开发工具介绍

在定制适用 Linux 系统和调试过程需要会用到许多调试，烧写的工具，在米尔提供的光盘镜像目录 03-Tools 下提供了部分工具，简单介绍如下：

驱动软件安装：

在 Windows 的环境下，当目标板设备上电并插上 USB 线之后，会自动安装 USB 设备驱动程序。如果安装成功，则会在 Windows 管理器中出现下图中红色椭圆形标识的设备 Android Phone。

图 2-1.驱动识别

烧录软件安装：

软件 PhoenixSuitv1.13：可用于 USB 在线烧录系统

软件 PhoenixCard4.2.4：可用于 Micro SD 烧录卡制作和 Micro SD 启动卡的制作。刷。这两个软件的安装包位于 tools\tools_win\。解压后会有中英文两个版本，选择一个版本即可。当 sdk 编译打包后，就可以通过 PhoenixSuit 烧录，详细步骤将在后文介绍。

注： T5 需要使用 PhoenixSuit（v1.13）和 PhoenixCard（4.2.4）以上版本，

不然可能会导致无法进行卡升级等问题。

2.3. 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：

a. 将 SDK 压缩包拷贝到 Ubuntu 下的用户工作目录，如$HOME/work 下，解压文件，得到安装脚本文件，如下：

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

注：其中 X.X.X 代表当前版本号

b. 进入 SDK 目录，在 build/toolchain 目录下可以找到

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

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

d. 设置环境变量，并测试安装是否完成。

```
PC$  export PATH=$PATH:/opt/gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu/bin
PC$  aarch64-linux-gnu-gcc -v
使用内建 specs。
COLLECT_GCC=aarch64-linux-gnu-gcc
COLLECT_LTO_WRAPPER=/home/lcy/t507/out/gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu/bin/../libexec/gcc/aarch64-linux-gnu/7.4.1/lto-wrapper
目标：aarch64-linux-gnu
配置为：'/home/tcwg-buildslave/workspace/tcwg-make-release_1/snapshots/gcc.git~linaro-7.4-2019.02/configure' SHELL=/bin/bash --with-mpc=/home/tcwg-buildslave/workspace/tcwg-make-release_1/_build/builds/destdir/x86_64-unknown-linux-gnu --with-mpfr=/home/tcwg-buildslave/workspace/tcwg-make-release_1/_
```

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

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，包括 gpu，cedarx，gstreamer，drm/weston，security 以及其他的私有软件包。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-YT507H 开发板的 linuxSDK 文件和数据，帮助开发者构建出可运行在 MYD-YT507H 开发板上的不同类型 Linux 系统镜像，如带 Qt5.12.5 图形库的 myir-image-full 系统镜像，不带 GUI 界面的 myir-image-core 系统镜像，下面以构建 myir-image-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。

3.2. 获取源码

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/YT507H-buildroot-t5-4.9.170-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/work/t507 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 所有目录：

```
PC$ cd $HOME/work/t507
```

```
PC$ tar -jxvf YT507H-buildroot-t5-4.9.170-X.X.X.tar.bz2
brandy  build  buildroot  build.sh  device  kernel  out  platform  test  tools
```

列出 SDK 目录内容如下：

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

9 directories
```

3.2.2. 通过 github 获取源码

目前 MYD-YT507H 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-YT507H_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  $HOME/work/t507
PC$ cd $HOME/work/t507
PC$ export REPO_URL='https://mirrors.tuna.tsinghua.edu.cn/git/git-repo/'
PC$ repo init -u https://github.com/MYiR-Dev/myir-t5-manifest.git --no-clone-bu
ndle --depth=1 -b master -m myir-t5-4.9.170-1.0.0.xml
PC$ repo sync
```

代码同步成功之后，同样在$HOME/work/t507 目录下得到一个 SDK 文件夹，里面包含 MYD-YT507H 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。

3.2.3. 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。

3.2.4. 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。

参考网址

Buildroot 用户手册 https://buildroot.org/downloads/manual/manual.html

Buildroot 源码下载位置 https://buildroot.org/downloads/

目录结构如下

```
├── arch
├── board
├── boot
├── CHANGES
├── Config.in
```

```
├── configs
├── COPYING
├── dl
├── docs
├── external-packages
├── fs
├── linux
├── Makefile
├── support
├── package
├── README
├── scripts
├── target
├── support
├── system
├── utils
└── toolchain
```

其中 configs 目录里存放预定义好的配置文件，比如我们的 sun50iw9p1_longan_defconfig，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。

3.2.5. kernel

linux 内核源码目录。当前使用的内核版本是 linux4.9.170。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。

3.2.6. brandy

brandy 目录下有 brandy2.0 版本，目前 T507 使用 brandy2.0 版本，其目录

结构为:

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

*默认的代码编译流程中并不会编译 uboot，用户需要修改 uboot 的时候需要自己编译。具体编译方法后续有介绍。

3.2.7. platform

平台私有软件包目录。

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

其中，framework/auto 内包含了 T5 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。

3.2.8. tools

编译打包工具， tools_win 是 PC 端烧录等工具。

```
tools/
├── build
```

```
├── codecheck
├── pack
└── tools_win
```

3.2.9. 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。

3.2.10. device

```
├── config
│       ├── chips
│       │   └── t507
│       ├── common
│       │   ├── debug
│       │   ├── dtb
│       │   ├── hdcp
│       │   ├── imagecfg
│       │   ├── partition
│       │   ├── sign_config
│       │   ├── toc
│       │   ├── tools
│       │   └── version
│       └── rootfs_tar
├── bin    打包时使用的启动文件
│       ├── bl31.bin
│       ├── boot0_nand_sun50iw9p1.bin      nand 用的 boot0 启动文件
│       ├── boot0_sdcard_sun50iw9p1.bin    emmc 用的 boot0 启动文件
│           ├── fes1_sun50iw9p1.bin 烧录工具用的初始化文件
│       ├── optee_sun50iw9p1.bin       optee
│       ├── sboot_sun50iw9p1.bin       安全启动的 bin，暂不支持
│       ├── u-boot-sun50iw9p1.bin      uboot 的 bin
├── boot-resource
```

```
|       ├── boot-resource
|       |   ├── bat
|       |   ├── bootlogo.bmp    启动画面，更新 bootlogo 可以替换此文件
|       └── boot-resource.ini
├── configs
|       ├── default 默认的平台性配置
|       |   ├── BoardConfig.mk
|       |   ├── boot_package.cfg       boot 引导的 boot 包内容配置
|       |   ├── diskfs.fex
|       |   ├── dragon_toc_android.cfg
|       |   ├── dragon_toc.cfg
|       |   ├── env_burn.cfg
|       |   ├── env.cfg       uboot 传给 linux 的启动参数
|       |   ├── env_dragon.cfg
|       |   ├── image.cfg
|       |   ├── image_dragonboard.cfg
```

4. 如何烧录系统镜像

米尔公司设计的 MYC-YT507H 系列核心板与开发板是基于全志公司的 T5 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：

- PhoenixSuit 烧写: 适用于研发调试，测试等场景。
- 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. PhoenixSuit 烧写

1). 工具需求

- 开发板一块
- USB Type_C 两根
- 12V 电源适配器
- PhoenixSuit 官方软件

2). 设置硬件

连接好硬件，将开发板的 J6 OTG 接口与电脑连接，插入电源适配器。

3). 在 windows 下烧录系统

使用 OTG USB 线连接开发板和主机， 先按住 FEL 键不要松开然后按 开机 键系统复位， 大约两秒后 松开 FEL 键。打开 windows 设备管理器，可发现 USB 设备驱动自动安装与识别。

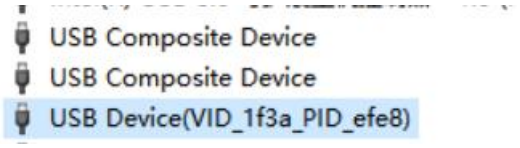

图 4-1.usb 连接 PC

如发现出现黄色感叹号，驱动没有自动安装，可在 SDK 目录 t507/tools/tools_win 下找到对应的驱动文件（64bit USBDriver_64.zip 和 32bit USBDriver.rar）并手动解压安装。

OTG 完全烧写测试，双击 PhoenixSuit 目录里 PhoenixSuit.exe 文件

图 4-2.PhoenixSuit 程序

在如下界面中， 点击”一键刷机” 再点击”浏览” 选择固件镜像文件

图 4-3.一键刷机

在如下界面中，点击”是” 进入格式化升级模式

图 4-4.格式化升级

等待烧写完成，烧写完成弹出如下界面

图 4-5.刷写完成

烧写完成关机，将 BOOT 拨码开关拨成 (S0/S1/S2/S3 : 1 0 1 1)，上电即可启动 MYD-YT507H 板卡。OTG uboot 烧写测试 在如下界面中， 勾选复选框”单或多分区下载(勾选此项，刷机工具下载你选择的分区)”，勾 选”BOOT-RESOURCE”和”ENV” 复选框

图 4-6.部分刷写

使用 OTG USB 线连接开发板和主机，先按住 FEL 键然不要松开然后按开机键系统复位，大约两秒 后松开 FEL 键，等待烧写完成。同理，其他配置项如 BOOT 和 ROOTFS 都可以通过勾选来单独刷机。

4.2. 制作 SD 卡启动器

以下步骤均在 Windows 系统下制作。

1). 准备工作

- SD 卡（不少于 8GB）
- MYD-YT507H 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-full	myir_linux_full_uart0.img	MYC-YT507H
myir-image-core	myir_linux_core_uart0.img	MYC-YT507H
myir-image-ubuntu	myir_linux_ubuntu_uart0.img	MYC-YT507H
myir-image-android	myir_linux_android_uart0.img	MYC-YT507H

2). 制作 SD 卡启动（以 myir-image-full 系统为例）

将用户资料 tools 目录的 PhoenixCard_.zip 拷贝到 windows 任意目录，双击 PhoenixCard 目录里 PhoenixCard.exe 文件。 通过 SD 读卡器把容量大小为 8GB 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。

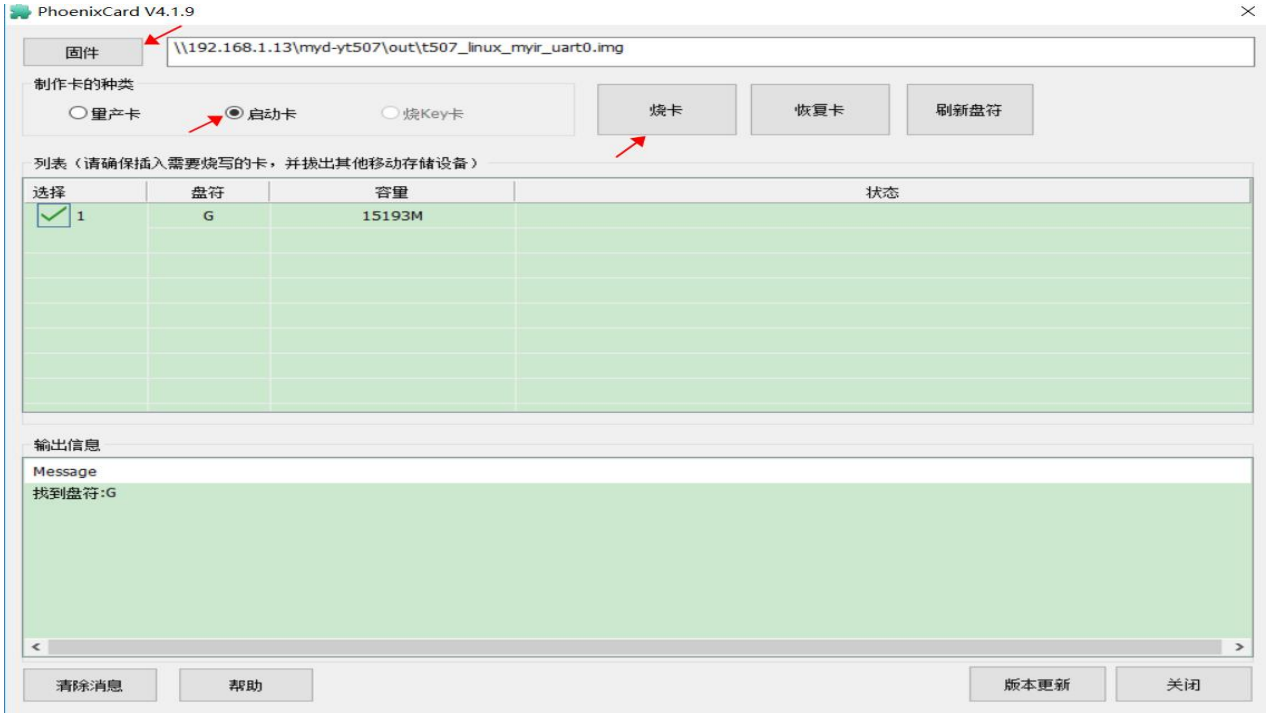

图 4-7.刷写程序

正在烧卡过程，预计 3-5 分钟完成（取决于包的大小）。

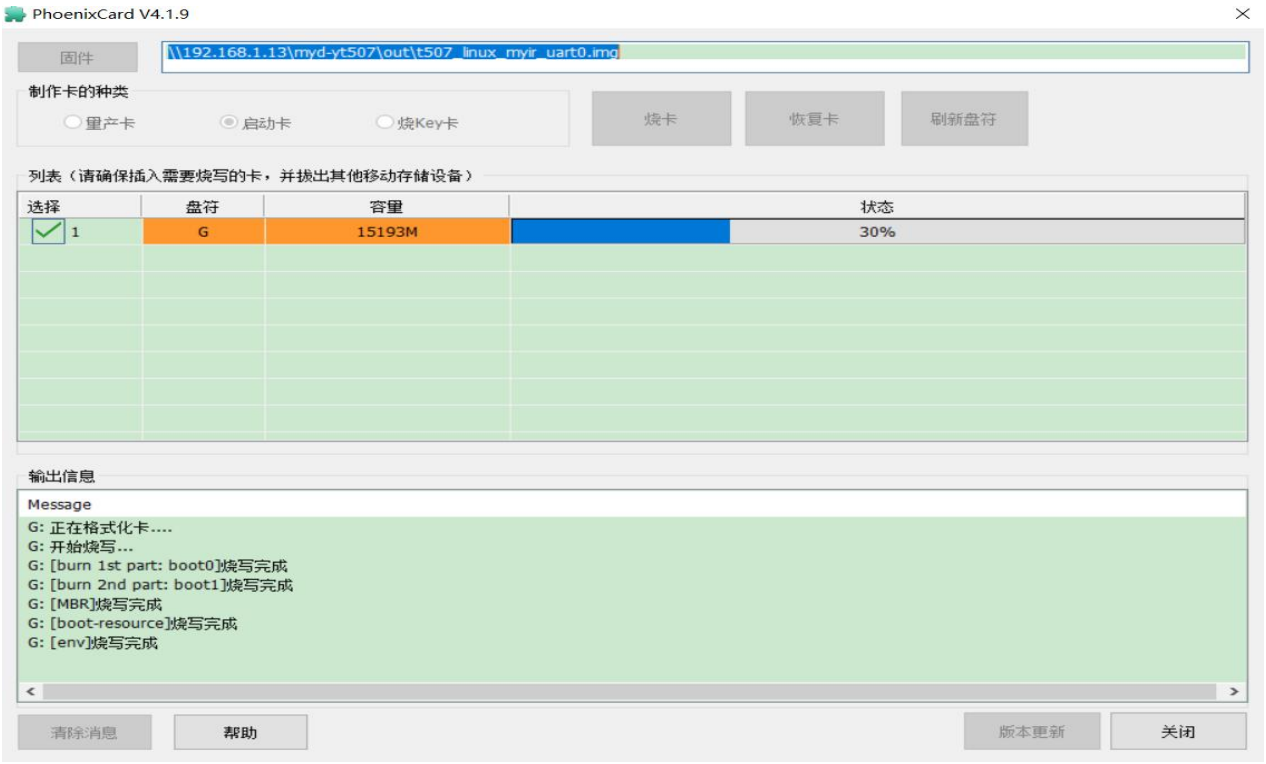

图 4-8.刷机过程

完成烧卡，同时注意输出信息提示烧写完成。

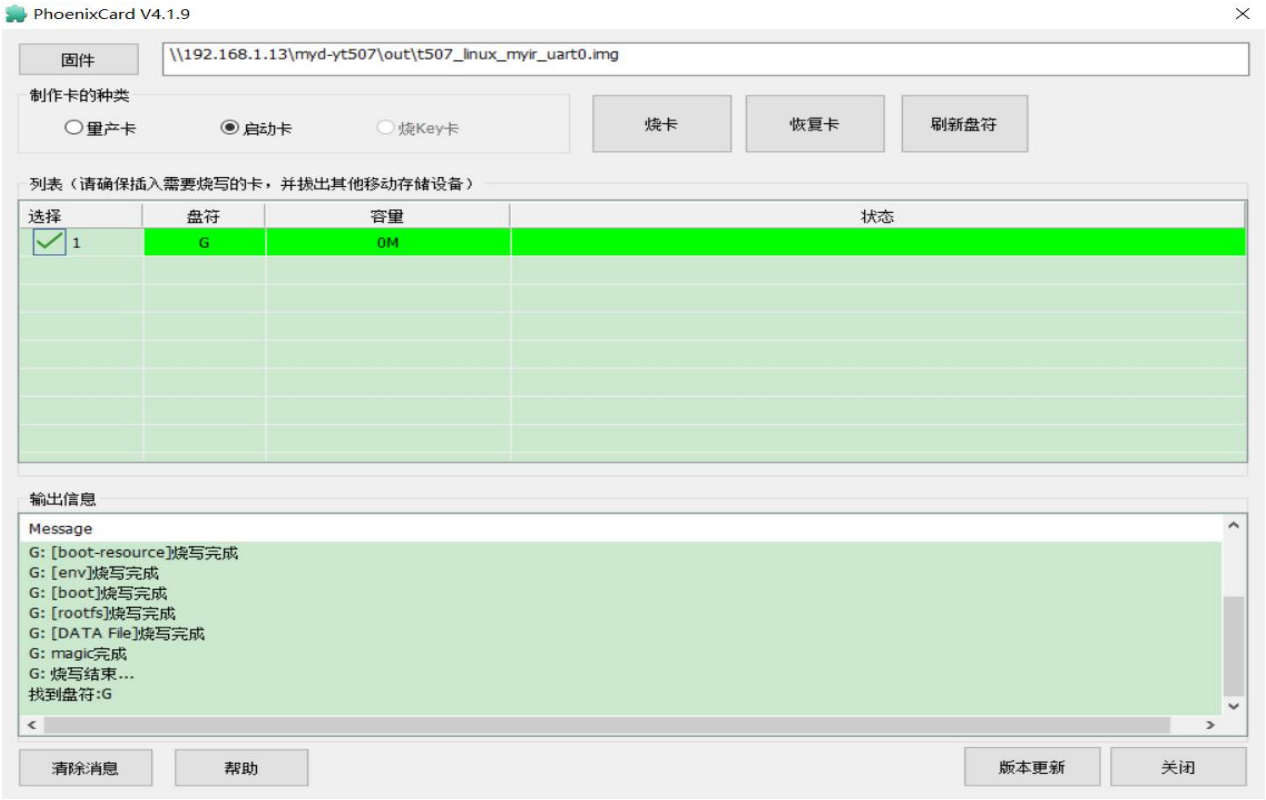

图 4-9.刷机完成

当写入完成后，即可使用此 SD 进行启动，将 SD 插入开发板背面的 SD 卡槽（J8），并将拨码开关调至(S0/S1/S2/S3: 0 1 1 1)，重新上电即可使用 SD 卡启动系统。

注意：当使用 SD 卡做启动卡启动 myir-image-core/full 时，需要在 uboot 中修改启动参数

调试接口打断 uboot 倒计时。在 uboot 终端下修改文件系统挂载位置。输入如下命令：

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

4.3. 制作 SD 卡烧录器

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 中。具体制作过程请按照下列步骤完成。

- **制作 SD 卡启动器**

制作一个 SD 卡启动器目的是使用 SD 作为媒介来刷写 emmc。

图 4-10.量产卡制作

选择“量产卡”来制作，制作方法同 4.2 章，这里就不在赘述。

- **SD 卡烧录 EMMC**

将 SD 插入开发板背面的 SD 卡槽（J8），并将拨码开关调至(S0/S1/S2/S3: 0 1 1 1)启动系统。插上电源，自动启动 SD Card 内的烧写系统，可以使用调试串口查看更新状态，大概 3-5 分钟（取决于包的大小）后完成。

- **验证 eMMC 启动**

通过 Micro SD Card 将系统镜像烧录到 eMMC 之后，需要关机并且拔出刷机的 SD 卡，就可以切换启动模式到 eMMC（S0/S1/S2/S3: 1 0 1 1）进行启动。

5. 如何修改板级支持包

前面的章节已经比较完整的讲述了基于 linux SDK(buildroot)项目构建运行在 MYD-YT507H 开发板上的系统镜像，并且将镜像烧录到开发板上的完整流程。由于 MYC-YT507H 核心板的很多管脚都具有多种功能复用的特性，所以实际项目中基于 MYC-YT507H 核心板设计的底板与 MYB-YT507H 相比总会有一些差异。这些差异可能是去掉显示，增加更多 GPIO, 或者需要增加更多串口，还有可能通过 SPI，I2C，USB 等扩展一些外设等等；除了硬件上的差异，还有一些系统组件上的差异，比如侧重 HMI 应用的，可能需要比较完备的图形系统，QT 库等，侧重后台管理应用的，可能需要更完备的网络应用，Python 运行环境等。这就需要系统开发人员在我们提供的代码基础上做一些裁剪和移植的工作。本章从一个系统开发人员的角度来讲述开发和定制自己系统的具体过程，为后面适配自己的硬件打下基础。

5.1. Buildroot 层介绍

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。

5.2. 板级支持包介绍

板级支持包(BSP)是定义如何支持特定硬件设备、设备集或硬件平台的信息集合。BSP 包括有关设备上的硬件特性的信息和内核配置信息，以及所需的任何其他硬件驱动程序。

在某些情况下，BSP 包含一个或多个组件的单独许可的知识产权(IP)。

通常根据硬件启动的不同阶段，我们将 BSP 分成 Bootloader 部分和 Kernel 部分，采用 MYC-YT507H 核心板设计的硬件 BSP 代码可以查看 linux SDK 的部分内容。

Brandy 中只包含了 Bootloader 部分的 spl 和 u-boot，这一部分主要实现核心硬件，如 DDR，Clock 的初始化及内核的引导。基于 MYC-YT507H 核心板硬件修改这部分的内容。

```
├── spl-pub
├── tools
```

```
└── u-boot-2018
```

kernel 中包含 Linux 内核部分，主要实现内核及外设固件内容。

```
kernel/
├── linux 4.9
```

在使用米尔的核心板设计产品时，如无特殊的需求，bootloader 部分可不必修改。你需要更多的关注产品内核驱动的开发与调试以及应用软件的设计。后续章节将详细描述内核开发与应用开发。

5.3. Linux SDK 的配置与构建

本章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。

如需要生成 myir-image-full 镜像需要执行如下步骤：

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

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。

```
PC$ ./build.sh config
Welcome to mkscript setup progress
All available platform:
0. android   //Android 方案
1. linux     //linux 方案
Choice [linux]:1
All available linux_dev:
0. bsp
1. dragonboard
2. longan   //作为首选配置
3. tinyos
Choice [longan]: 2
```

```
All available ic:
  0. myir
  1. t507
Choice [myir]: 0
All available board:
  0. core  //配置成 myir-image-core
  1. full   //配置成 myir-image-full
  2. myd_test
  3. myt
  4. ubuntu  //配置成 myir-image-ubuntu
Choice [ubuntu]: 1
  0. default
  1. nor  //无 nor flash
Choice [default]: 0
All available rootfs:
 0. buildroot-201902  //linux 文件系统
 1. ubuntu  //ubuntu 文件系统
All available build_root:
  0. buildroot-201902 //linux 文件系统
  1. myir-t5-buildroot  //和 buildroot-201902 是同一个
  2. ubuntu  //ubuntu 文件系统
Choice [ubuntu]: 0
```

进入 顶级目录，执行如下命令即可。

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

图 5-1. 构建完成

表 5-1.参数说明

类型	命令	说明
整体编译	./build.sh config	编译配置，弹出编译选择
	./build.sh autoconfig	根据传入的参数进行编译配置，不弹出编译选择
	./build.sh	根据编译配置，编译 SDK
	./build.sh clean	清除过程文件和目标文件
	./build.sh distclean	清除所有生成的文件
局部编译	./build.sh brandy	编译 brandy(uboot)
	./build.sh kernel	编译 kernel
	./build.sh buildroot	编译 buildroot
	./build.sh qt	编译 qt
	./build.sh dragonboard	编译 dragonboard
	./build.sh sata	编译 sata
打包	./build.sh pack	打包命令，调试串口为 uart0
	./build.sh pack_debug	打包命令，调试串口为 uard0
	./build.sh pack_secure	打包命令，生成 secure 固件，调试串口为 uart0

编译问题分析：

执行./build.sh config 出现如下问题

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

图 5-2. 配置失败

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

图 5-3. 编译内核失败

由于配置内核需要使用 python2，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 执行如下命令：

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

检查当前版本是否是 python

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

如非 python2 版本请自行切换。

GDBUS 编译过程中会出现如下错误：

```
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 5-4. 编译 GDbus 失败

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$ vim ./output/myir/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$ vim ./out/myir/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)
```

5.4. 板载 u-boot 编译与更新

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

T5 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。

5.4.1. 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/YT507H-buildroot-t5-4.9.170-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/t507），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 $HOME/work/t507
PC$ tar -jxvf YT507H-buildroot-t5-4.9.170-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

```
lrwxrwxrwx  1 root root   14 2 月  10 10:44 build.sh -> tools/build.sh
drwxr-xr-x 10 root root 4096 3 月   4 14:59 spl-pub
drwxr-xr-x  5 root root 4096 2 月  10 10:45 tools
drwxr-xr-x 26 root root 4096 3 月   4 14:59 u-boot-2018
```

2). 配置与编译

- **进入源代码目录**

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

- **加载 SDK 里的工具链**

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

- **编译 U-Boot**

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

5.4.2. 在 linux SDK 项目下编译 u-boot(推荐)

当用户按照 5.4.1 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。

编译 uboot 源码：

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

打包文件：

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

5.4.3. 如何单独更新 U-boot

使用 PhoenixSuit 执行单独更新策略即可。详细请查看 4.1 章

图 5-1.更新 uboot

5.5. 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 T5 也支持 Linux 内核，将得到长期稳定的更新，MYD-YT507H 使用 T5 内核移植，最新支持 Linux kernel 4.9.170 版本。

5.5.1. 在独立的交叉编译环境下编译 Kernel

5.5.1.1. 获取 kernel 源代码

拷贝开发包 04-Source/YT507H-buildroot-t5-4.9.170-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/t507），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 $HOME/work/t507
PC$ tar -jxvf YT507H-buildroot-t5-4.9.170-X.X.X.tar.bz
PC$ cd kernel/
```

目录包含：

- 源代码软链接目录：linux-4.9
- 源代码：myir-t5-kernel

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

5.5.1.2. 配置内核（非必须）

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。

- **进入内核目录**

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

- **创建输出文件夹 build**

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

- **配置内核**

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

如需配置内核或者想打开内核某一个驱动功能也可使用如下方式。

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

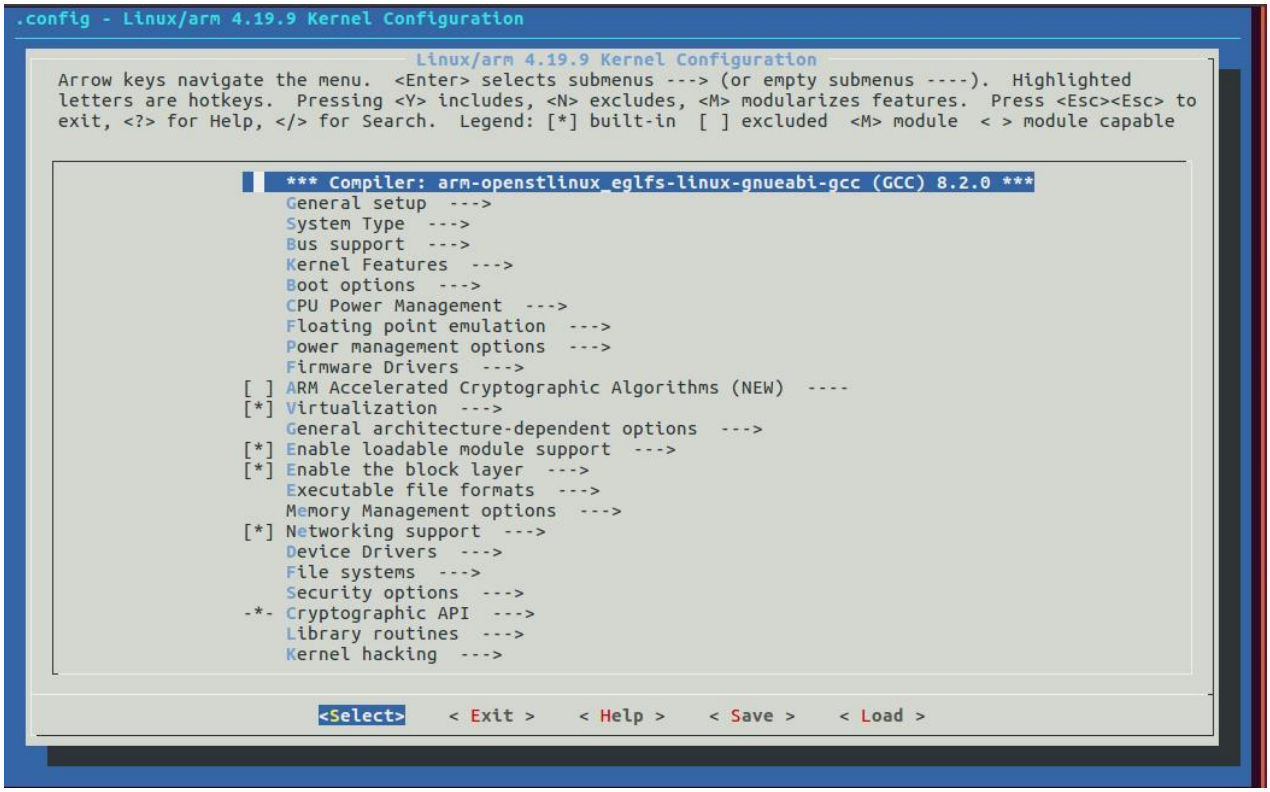

图 5-2. 内核配置界面

5.5.1.3. 编译内核(不推荐)

- **加载 SDK 环境变量**

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

- **配置内核**

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

- **编译内核**

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

耐心等待编译完成.

5.5.1.4. 使用 linux SDK 编译(推荐)

编译 kernel 源码：

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

打包文件：

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

5.5.2. 如何 OTG 更新 Kernel

使用 PhoenixSuit 执行单独更新策略即可。勾选“BOOT”进行更新，详细制作过程请参考 4.1 章。

图 5-3.刷写系统

6. 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-YT507H 开发板提供了哪些资源，具体的信息可以查看《**MYD-YT507H SDK1.0.0** 发布说明》。除此之外用户还需要对 CPU 的芯片手册，以及 MYC-YT507H 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。

6.1. 如何配置您的 sys_config.fex

sys_config.fex 是全志对 T5 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$: device/config/chips/myir/configs/xxx/sys_config.fex   (xxx 代表不同的配置)
```

定义属性类方式：

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

[platform]
eraseflag   = 1
debug_mode  = 0
;------------------------------------------------------------------------------------
;[target]  system bootup configuration
;boot_clock     = CPU boot frequency, Unit: MHz
;storage_type   = boot medium, 0-nand, 1-card0, 2-card2, -1(defualt)auto scan
;advert_enable  = 0-close advert logo    1-open advert logo     (只有多核启动下有效)
;------------------------------------------------------------------------------------
[target]
boot_clock      = 1008
storage_type    = -1
advert_enable   = 0
```

```
burn_key        = 1
```

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><3><default>
sdc_d0          = port:PF1<2><1><3><default>
sdc_clk         = port:PF2<2><1><3><default>
sdc_cmd         = port:PF3<2><1><3><default>
sdc_d3          = port:PF4<2><1><3><default>
sdc_d2          = port:PF5<2><1><3><default>
;sdc_type       = "tm1"
```

*由于全志资料需获得相关的授权，以上两类的配置定义详细含义请联系米尔的技术支持获取文档《**T507_sys_config.fex 使用配置说明.pdf**》

6.2. 如何创建您的设备树

6.2.1. 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-YT507H 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 6-1.MYD-YT507H 设备树列表

项目	设备树	说明
U-boot	sys_config.fex	sys_config.fex 配置（见 6.1）
Kernel	sys_config.fex	sys_config.fex 配置（见 6.1）
	board.dts	底板配置资源
	myir-yt507.dtsi	核心板内部配置资源
	sun50iw9p1-myir.dtsi	核心资源配置
	sun50iw9p1-myir-pinctrl.dtsi	管脚配置
	display/myir-hdmi-1920x1080-1lvds-7-1024x600.dtsi	HDMI 和 LVDS 双显设备树配置
	display/myir-lcd-1lvds-7-1024-600.dtsi	7 寸单路 LVDS 设备树配置
	display/myir-lcd-2lvds-21-1920-1080.dtsi	21 寸双路 LVDS 设备树配置
	display/myir-tv.dtsi	CVBS-OUT 设备树配置
	Display/myir-hdmi.dtsi	HDMI 单独输出设备树配置

注：以上文件路径如下：

device/config/chips/myir/configs/xxx/sys_config.fex（xxx 代表不同的配置,如 full)

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

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

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

6.2.2. 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。

在内核源码下 arch/arm64/boot/dts 下可以看到大量的平台设备树。如适合 MYD-YT507H 的设备树，可在当前路径下增加自定义设备树,如：

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

我们将 MYC-YT507H 核心板相关的资源编写进 sun50iw9p1-myir.dtsi 以及 myir-yt507.dtsi 和 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

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

board.dts 配置如下

```
//device/config/chips/myir/configs/full/board.dts
/*
 * myir-YT507H support.
 */
/dts-v1/;

#include "myir-yt507.dtsi"
#include "display/myir-hdmi-1920x1080-1lvds-7-1024x600.dtsi"
//#include "display/myir-lcd-1lvds-7-1024-600.dtsi"
//#include "display/myir-lcd-lvds-10.1-1280-800.dtsi"
//#include "display/myir-lcd-2lvds-7-1024-600.dtsi"
//#include "display/myir-lcd-2lvds-21-1920-1080.dtsi"
//#include "display/myir-hdmi.dtsi"
//#include "display/myir-tv.dtsi"

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

          aliases {
                pmu0 = &pmu0;
                standby_param = &standby_param;
```

```
        };

        soc@03000000 {

            twi2: twi@0x05002800{
                status = "okay";

            };
        };
} ;
```

6.3. 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。

6.3.1. GPIO 管脚配置的方法

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。

T5 封装大量的外设控制器，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). 查询手册方式

GPIO 的配置可通过米尔整理的 Datasheet 找到描述文件（01-Document\Datasheet\）与核心板引脚清单（01-Document\Hardware\），示例如下：

- **通用计算方法**

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

- **配置 pinmux**

  下面 LED 接口上 Blue 的 NCSI0-D15 为例，查看硬件手册获得此管脚复用功能

  PE19/NCSI0-D15/PE-EINT20；

表 6-2.NCSI0-D15 管脚复用列表

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

6.3.2. 设备树中引用 GPIO

1). 配置功能管脚为 GPIO 功能实例

此实例使用 PE19 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。

只需在设备树里增加节点即可。

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

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

2). 开发板 LCD 资源重新分配实例

MYD-YT507H 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-YT507H 基础上进行设计开发，将需要对管脚进行重新定义和配置。下列就以将 LCD 复用管脚（LCD-D0）功能，可先查表了解管脚的复用功能。

表 6-3.LCD-D0 管脚复用列表

Function	pinName	IO State	up/down	Multi2(DDR4)	Multi3(DDR3)	Multi4(LPDDR3)	Multi5(LPDDR4)	Multi6	PIN power
PD	PD0	I/O	DIS	LCD-D0	LVDF0-V0P	TS0-CLK		PD-EINT0	VCC-PD

MYD-YT507H 开发板已经使用 PD0 两个管脚作为 LVDS 的数据信号管脚，管脚配置如下：

```
//kernel/linux-4.9/arch/arm64/boot/dts/sunxi/sun50iw9p1-myir-pinctrl.dtsi
 lvds0_pins_a: lvds0@0 {
                        allwinner,pins = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5", "
PD8", "PD9", "PD6", "PD7";
                        allwinner,pname = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5",
 "PD8", "PD9", "PD6", "PD7";
                        allwinner,function = "lvds0";
                        allwinner,muxsel = <3>;
```

```
                        allwinner,drive = <3>;
                        allwinner,pull = <0>;
                    };
                    lvds0_pins_b: lvds0@1 {
                        allwinner,pins = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5", "
PD8", "PD9", "PD6", "PD7";
                        allwinner,pname = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5",
 "PD8", "PD9", "PD6", "PD7";
                        allwinner,function = "lvds0_suspend";
                        allwinner,muxsel = <7>;
                        allwinner,drive = <3>;
                        allwinner,pull = <0>;
                    };
```

通过查手册可重新配置进行管脚给 i2C1 使用分配为 PD0（Multi2(DDR4)）。

```
//linux-4.9/arch/arm64/boot/dts/sunxi/display/myir-lcd-1lvds-7-1024-600.dtsi
. . .
        &lcd0 {
                    lcd_used            = <1>;
                    lcd_driver_name     = "default_lcd";
                    lcd_backlight       = <200>;
                    lcd_if              = <3>;
                    lcd_x               = <1024>;
                    lcd_y               = <600>;
                    lcd_width           = <150>;
                    lcd_height          = <94>;
                    lcd_dclk_freq       = <50>;   // <70>
                    lcd_pwm_used        = <1>;
                    lcd_pwm_ch          = <0>;
                    lcd_pwm_freq        = <50000>;
                    lcd_pwm_pol         = <1>;
                    lcd_pwm_max_limit   = <255>;
                    lcd_hbp             = <160>;
```

```
                lcd_ht          = <1324>;
                lcd_hspw        = <116>;
                lcd_vbp         = <24>;
                lcd_vt          = <629>;
                lcd_vspw        = <3>;
                lcd_lvds_if     = <0>;
                lcd_lvds_colordepth = <0>;
                lcd_lvds_mode   = <0>;
                lcd_frm         = <0>;
                lcd_hv_clk_phase  = <0>;
                lcd_hv_sync_polarity= <0>;
                lcd_gamma_en    = <0>;
                lcd_bright_curve_en = <0>;
                lcd_cmap_en     = <0>;
                lcd_fsync_en    = <0>;
                lcd_fsync_act_time  = <1000>;
                lcd_fsync_dis_time  = <1000>;
                lcd_fsync_pol   = <0>;
                deu_mode        = <0>;
                lcdgamma4iep    = <22>;
                smart_color     = <90>;
                lcd_pin_power = "bldo1";
                lcd_power = "dc1sw";
                //lcd_bl_en     = <&pio PD 28 1 0 3 1>;
                //lcd_gpio_0    = <&pio PH 4 1 0 3 1>;

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

当不使用此 lcd0 接口时，也可以将 lcd 设备树节点状态配置为禁用态“disabled”。

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

6.4. 如何使用自己配置的管脚

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

6.4.1. U-boot 中使用 GPIO 管脚

1). 终端命令控制

uboot 可以直接使用命令来控制 GPIO 的设置。如设置 GPIOF14，可使用下列命令。

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

6.4.2. 内核驱动中使用 GPIO 管脚

1). 独立 IO 驱动的使用

在 6.2.3 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 PF14 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。

2). 驱动示例将直接配置进内核

在内核源代码的 sample 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 sun50iw9p1_myir_defconfig。

添加 Kconfig：

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

添加 Makefile：

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

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

添加 sun50iw9p1_myir_defconfig：

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

按照 5.5.3 节编译与更新内核即可。

3). 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。

```
# 修改 KERN_DIR
#KERN_DIR = # 板子所用内核源码的目录
KERN_DIR = $HOME/work/t507/kernel/linux-4.9/

obj-m += gpioctr.o
```

```
all:
	make -C $(KERN_DIR) M=`pwd` modules

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

# 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。

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

执行 make 命令，即可生成 gpioctr.ko 驱动模块文件。

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

编译成功之后，将 gpioctr.ko 文件可通过以太网，WIFI, USB otg, U 盘等传输介质传输到开发板的/lib/modules 目录下即可使用 insmod 命令加载驱动。

4). 外设控制器的使用

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，本节不对驱动开发做具体的讲解。

6.4.3. 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

1). Shell 实现管脚控制

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 PF14 > sunxi_pin
[root@myir:/]# cat sunxi_pin_configure
```

结果如下图所示：

```
/sys/kernel/debug # cd sunxi_pinctrl/
/sys/kernel/debug/sunxi_pinctrl # ls
data                function            sunxi_pin
device              platform            sunxi_pin_configur
dlevel              pull
/sys/kernel/debug/sunxi_pinctrl # echo PC2 > sunxi_pin
/sys/kernel/debug/sunxi_pinctrl # cat sunxi_pin_configure
pin[PC2] funciton: 4
pin[PC2] data: 0
pin[PC2] dlevel: 1
pin[PC2] pull: 0
```

图 6-1.GPIO 属性

修改 pin 属性：

```
[root@myir:/]# echo PC2 1 > pull
[root@myir:/]# cat sunxi_pin_configure
```

修改后结果如下：

```
/sys/kernel/debug/sunxi_pinctrl # echo PC2 > sunxi_pin
/sys/kernel/debug/sunxi_pinctrl # cat sunxi_pin_configure
pin[PC2] funciton: 4
pin[PC2] data: 0
pin[PC2] dlevel: 1
pin[PC2] pull: 0
/sys/kernel/debug/sunxi_pinctrl # echo PC2 1 > pull
/sys/kernel/debug/sunxi_pinctrl # cat sunxi_pin_configure
pin[PC2] funciton: 4
pin[PC2] data: 0
pin[PC2] dlevel: 1
pin[PC2] pull: 1
```

图 6-2.GPIO 属性修改

更多关于 GPIO 的开发方法请查看**《MYD-YT507H_GPIO_开发笔记》**。

2). 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。
- 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 PF14 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)。

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

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

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

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

        return ret;
        }
```

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

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

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

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

```
        return ret;
}
```

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

```
PC$  export PATH=$PATH:/opt/gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu/bin
PC$ export CC=aarch64-linux-gnu-gcc
PC$ export CROSS_COMPILE=aarch64-linux-gnu-
```

使用编译命令$CC 可生成可执行文件 example-gpio。

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

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。

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

3). 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 6.3.2 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_gpioctr on
```

```
 * ./gpiotest /dev/myir_gpioctr 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);

        return 0;
}
```

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

```
PC$  export PATH=$PATH:/opt/gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu/bin
PC$ export CC=aarch64-linux-gnu-gcc
PC$ export CROSS_COMPILE=aarch64-linux-gnu-
```

使用编译命令 aarch64-linux-gnu-gcc 可生成可执行文件 gpiotest。

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

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

7. 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。

7.1. 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 会自动的判别原始码是否经过变动了，而自动更新执行档。

下列将以一个实际的示例（在 MYD-YT507H 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。

```
key_led.o: key_led.c
      ${CC} -I . -c key_led.c
all: key_led.o
      ${CC} key_led.c -o target_bin

clean:
        rm -rf *.o
        rm target_bin
```

- CC：C 编译器的名称

- CXX: C++编译器的名称
- all：通常为缺省的目标，执行缺省的编译工作
- clean: 是一个约定的目标

Key_led 实现代码如下：

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

/* ./key_led /dev/input/event0 noblock */
int main(int argc, char **argv)
{
        int fd,bg_fd;
        int err, len, i;
        unsigned char flag;
        unsigned int data[1];
        char *bg = "/sys/class/leds/heartbeat/brightness";

        struct input_event event;

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

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

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

                            printf("key test \n");
                            bg_fd = open(bg, O_RDWR);
                            if (bg_fd < 0)
                           {
                                    printf("open %d err\n", bg_fd);
                            return -1;
                    }
                            read(bg_fd,&flag,1);
                            if(flag == '0')
                            system("echo 1 > /sys/class/leds/heartbeat/brightness"); //l
ed off
```

```
                        else
                                system("echo 0 > /sys/class/leds/heartbeat/brightness
");//led on
                    }


                }


        }
    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 target_bin。

加载编译交叉工具链环境变量到当前 shell:

执行 make:

```
PC$ make
```

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 target_bin 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:~# target_bin /dev/input/event0 noblock
```

说明：如果使用交叉工具链编译器构建 target_bin，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。

7.2. 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-YT507H 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。

1). QtCreator 安装与配置

从 QT 官网获得 qtcreator 安装包 QT 官网下载：http://download.qt.io/development_releases/qtcreator/4.1/4.1.0-rc1/。

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_64-4.1.0-rc1.run 即可，如需获得安装与配置详情请查看《**MYD-YT507H QT** 应用开发笔记》或从 QTCreator 官方网站获得更多开发指导 https://www.qt.io/product/development-tools。

2). MEasy HMI2.0 编译和运行

MEasy HMI 2.0 是深圳市米尔科技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 HMI2.0 项目源代码“MYD-YT507H-2022xxx\04-Sources\mxapp2.tar.gz”。可通过 Qtcreator 进行加载编译，远程调试等，可参看《**MYD-YT507H QT** 应用开发笔记》。

8. 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

负责区域：广东 / 四川 / 重庆 / 湖南 / 广西 / 云南 / 贵州 / 海南 / 香港 / 澳门
电话：0755-25622735 0755-22929657
传真：0755-25532724
邮编：518020
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

生产基地

地址：深圳市龙华区观澜街道大富工业区圣建利工业园 C 栋厂房 2 楼
电话：0755-21015844

上海办事处

负责区域：上海 / 湖北 / 江苏 / 浙江 / 安徽 / 福建 / 江西
电话：021-60317628 15901764611
传真：021-60317630
邮编：200062
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

北京办事处

负责区域：北京/天津/陕西/辽宁/山东/河南/河北/黑龙江/吉林/山西/甘肃/内蒙古/宁夏
电话：010-84675491 13269791724
传真：010-84675491
邮编：102218
地址：北京市大兴区荣华中路 8 号院力宝广场 10 号楼 901 室

销售联系方式

网址：www.myir-tech.com
邮箱：sales.cn@myirtech.com

技术支持联系方式

电话：0755-25622735
邮箱：support.cn@myirtech.com

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。

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，以及米尔科技开发的部分软件源代码
6、可直接从米尔科技购买主要芯片样品，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，即成为米尔科技永久客户，享有再次购买米尔科技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。