MYD-JX8MPQ Linux 软件开发指南

文件状态: [ ] 草稿 [√] 正式发布	**文件标识:**	MYIR-MYD-JX8MPQ-SW-DG-ZH-L5.10.9
	当前版本:	V1.1[DOC]
	作　　者:	Coin.Du
	创建日期:	2021-10-19
	最近更新:	2022-10-20

版 本 历 史

版本	作者	参与者	日期	备注
V1.0	Coin.Du		20211019	初始版本：uboot 2020，kernel 5.10.9，yocto 3.2.1
V1.1	Coin.Du		20221020	增加 2G/4GDDR 配置

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 Yocto 项目使用更强大和定制化的方法，来构建出适合嵌入式产品的 Linux 系统。Yocto 不仅仅是一个制做文件系统工具，同时提供整套的基于 Linux 的开发和维护工作流程，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。

本文首先讲述在 MYD-JX8MPQ 系列开发板上使用 Yocto 项目安装运行 Linux 系统以及嵌入式 Linux 驱动和应用程序的开发流程，其中包括部署开发环境、构建系统、Linux 应用程序的实例分析、镜像的更新等。系统开发人员熟悉第三章的 Yocto 的开发流程之后，就可以参照第四章的移植指南针对实际项目需求对 BSP 进行差异化的定制，就可以很快的将系统移植到基于 MYD-JX8MPQ 核心板设计的硬件平台之上。

本文档并不包含 Yocto 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。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开发手册。

1.1. 软件资源

MYD-JX8MPQ 搭载基于 Linux 5.10.9 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，ATF 源代码，U-boot 源代码，Linux 内核和各驱动模块的源代码，以及适用于 Windows 桌面环境和 Linux 桌面环境的各种开发调试工具，应用开发样例等。具体的包含的软件信息请参考《MYD-JX8MPQ 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阶段，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-JX8MPQ 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-JX8MPQ 开发板进行系统移植，应用开发，固件烧录等整个开发流程所需的一些软硬件环境，包括必要的硬件配件，软件工具等，具体的准备工作下面将进行详细介绍。

2.1. 开发主机环境

本章节将介绍如何搭建 MYD-JX8MPQ 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。i.MX8MPQ 系列处理器是一个多核异构的处理器，其包含：

- 4 个 ARM Cortex A53 内核，可以运行嵌入式 Linux 系统，使用嵌入式 Linux 系统的开发工具。
- 1 个 ARM Cortex M7 内核，可以运行裸机代码或其他实时操作系统（RTOS），使用 NXP 官方提供的 Cortex M7 软件开发工具。

1）主机软硬件要求

- **主机硬件**

Yocto 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，8GB 以上 内存，500GB 硬盘或更高配置。可以是安装 Linux 系统的主机，也可以是运行 Linux 系统的虚拟机。

- **主机操作系统**

构建 Yocto 项目的主机操作系统可以有很多种选择，详细的信息请参考 Yocto 官方说明 https://docs.yoctoproject.org/current/ref-manual/index.html。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，后续开发也是以此系统为例进行介绍。

2）主机环境配置

当安装完 ubuntu18.04 64bit 桌面版后，可以先进行适当的配置，为后续开发做准备。

- **设置 root 密码**

```
sudo passwd root
```

- **更换清华源**

Ubuntu 安装工具常用命令就是 apt-get，但是安装完系统后，默认的源为国外服务器，下载会很慢，所以需要更换为清华源。打开清华大学开源软件镜像站：https://mirrors.tuna.tsinghua.edu.cn/help/ubuntu/ ，选择你的 ubuntu 版本，如 18.04，Ubuntu 的软件源配置文件是 /etc/apt/sources.list。将系统自带的该文件做个备份，将该文件替换为下面内容，即可使用 TUNA 的软件源镜像。

```
# 默认注释了源码镜像以提高 apt update 速度，如有需要可自行取消注释
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic main restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-updates main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-updates main restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-backports main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-backports main restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-security main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-security main restricted universe multiverse

# 预发布软件源，不建议启用
# deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-proposed main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-proposed main restricted universe multiverse
```

- **安装 ssh**

安装完 ssh 服务后，可以在 window 环境下，用 sercureCRT 工具 ssh2 的方式连接到 ubuntu 进行后续开发。

```
sudo apt-get install openssh-server
```

给用户生成密钥：

```
su user
ssh-keygen -t rsa
```

- **配置 samba**

samba 可以直接在 window 下以文件夹形式访问 ubuntu 的内容，读写更方便。安装 samba：

```
apt-get install samba
```

在/etc/samba/smb.conf 中加入用户配置，如 linux 用户名为“duxy“，如下配置：

```
[duxy]
path = /home/duxy
valid users = duxy
browseable = yes
public = yes
writable = yes
```

创建账号并设置密码：

```
$ sudo smbpasswd -a duxy
New SMB password:
Retype new SMB password:
Added user duxy.
```

/etc/init.d/smbd restart 重启 samba 服务：

```
$ /etc/init.d/smbd restart
[ ok ] Restarting smbd (via systemctl): smbd.service.
```

- **配置 git**

```
git config --global user.name "user"
git config --global user.email "email"
git config --list
```

- **安装 vim**

自带的 vim 工具无法退格，需要重新安装：

```
sudo apt-get remove vim-common
sudo apt-get install vim
```

- **安装 SDK 必要工具**

```
sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib \
build-essential chrpath socat cpio python python3 python3-pip python3-pexpect \
xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev \
pylint3 xterm rsync curl libssl-dev
```

2.2. 安装米尔定制的 SDK

我们在使用 Yocto 构建完系统镜像之后，还可以使用 Yocto 构建一套可扩展的 SDK。在米尔提供的光盘镜像中包含两个编译好的 SDK 包，位于：*03-Tools/Toolchains/*，两个 SDK 文件功能描述如下表：

表 2-1.编译工具链

工具链文件名	描述
fsl-imx-xwayland-glibc-x86_64-meta-toolchain-qt5-cortexa53-crypto-myd-jx8mp-toolchain-5.10-gatesgarth.sh	包含一个独立的交叉开发工具链还提供 qmake, 目标平台的 sysroot, Qt 应用开发所依赖的库和头文件等。用户可以直接使用这个 SDK 来建立一个独立的开发环境
fsl-imx-xwayland-glibc-x86_64-meta-toolchain-cortexa53-crypto-myd-jx8mp-toolchain-5.10-gatesgarth.sh	基础工具链，单独编译 Bootloader，Kernel 或者编译自己的应用程序

这里先介绍 SDK 的安装步骤，如下：

- **拷贝 SDK 到 Linux 目录**

将 SDK 压缩包拷贝到 Ubuntu 下的用户工作目录，得到安装脚本文件，如下：

```
PC$ fsl-imx-xwayland-glibc-x86_64-meta-toolchain-cortexa53-crypto-myd-jx8mp-toolchain-5.10-gatesgarth.sh
PC$ fsl-imx-xwayland-glibc-x86_64-meta-toolchain-qt5-cortexa53-crypto-myd-jx8mp-toolchain-5.10-gatesgarth.sh
```

- **执行安装脚本**

以普通用户权限执行shell 脚本， 运行中会提示安装路径， 默认在/opt目录下。本例程把 meta 基础工具链安装在/home/duxy/opt_5_10/目录，如下：

```
duxy@myir_server:~/tools$ ./fsl-imx-xwayland-glibc-x86_64-meta-toolchain-cortexa53-crypto-myd-jx8mp-toolchain-5.10-gatesgarth.sh
NXP i.MX Release Distro SDK installer version 5.10-gatesgarth
================================================================
```

```
Enter target directory for SDK (default: /opt/fsl-imx-xwayland/5.10-gatesgarth): /home/duxy/opt_5_10
You are about to install the SDK to "/home/duxy/opt_5_10". Proceed [Y/n]? y
Extracting SDK.....................................done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g.
 $ . /home/duxy/opt_5_10/environment-setup-cortexa53-crypto-poky-linux
```

- **测试 SDK**

  安装完成后，使用以下命令加载环境变量到当前 shell，测试 SDK 是否完成：

```
duxy@myir_server:~$ source ~/opt_5_10/environment-setup-cortexa53-crypto-poky-linux
duxy@myir_server:~$ $CC -v
Using built-in specs.
COLLECT_GCC=aarch64-poky-linux-gcc
COLLECT_LTO_WRAPPER=/home/duxy/opt_5_10/sysroots/x86_64-pokysdk-linux/usr/libexec/aarch64-poky-linux/gcc/aarch64-poky-linux/10.2.0/lto-wrapper
Target: aarch64-poky-linux
Configured with: ../../../../../../work-shared/gcc-10.2.0-r0/gcc-10.2.0/configure --build=x86_64-linux --host=x86_64-pokysdk-linux --target=aarch64-poky-linux --prefix=/opt/fsl-imx-xwayland/5.10-gatesgarth/sysroots/x86_64-pokysdk-linux/usr --exec_prefix=/opt/fsl-imx-xwayland/5.10-gatesgarth/sysroots/x86_64-pokysdk-linux/usr --bindir=/opt/fsl-imx-xwayland/5.10-gatesgarth/sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux --sbindir=/opt/fsl-imx-xwayland/5.10-gatesgarth/sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux --libexecdir=/opt/fsl-imx-xwayland/5.10-gatesgarth/sysroots/x86_64-pokysdk-linux/usr/libexec/aarch64-poky-linux --datadir=/opt/fsl-imx-xwayland/5.10-gatesgarth/sysroots/x86_64-pokysdk-linux/usr/share --sysconfdir=/opt/fsl-imx-xwayland/5.10-gatesgarth/sysroots/x86_64-pokysdk-linux/etc --sharedstatedir=/opt/fsl-imx-xwayland/5.10-gatesgarth/sysroots/x86_64-pokysdk-linux/com --localstatedir=/opt/fsl-imx-xwayland/5.10-gatesgarth/sysroots/x86_64-pokysdk-linux/var --libdir=/opt/fsl-imx-xwayland/5.10-gatesgarth/sysroots/x86_64-pokysdk-linux/usr/lib/aarch64-poky-linux --inclu
```

```
dedir=/opt/fsl-imx-xwayland/5.10-gatesgarth/sysroots/x86_64-pokysdk-linux/usr/include --oldincludedir=/opt/fsl-imx-xwayland/5.10-gatesgarth/sysroots/x86_64-pokysdk-linux/usr/include --infodir=/opt/fsl-imx-xwayland/5.10-gatesgarth/sysroots/x86_64-pokysdk-linux/usr/share/info --mandir=/opt/fsl-imx-xwayland/5.10-gatesgarth/sysroots/x86_64-pokysdk-linux/usr/share/man --disable-silent-rules --disable-dependency-tracking --with-libtool-sysroot=/home/duxy/L5.10.9/build-xwayland/tmp/work/x86_64-nativesdk-pokysdk-linux/gcc-cross-canadian-aarch64/10.2.0-r0/recipe-sysroot --with-gnu-ld --enable-shared --enable-languages=c,c++ --enable-threads=posix --enable-multilib --enable-default-pie --enable-c99 --enable-long-long --enable-symvers=gnu --enable-libstdcxx-pch --program-prefix=aarch64-poky-linux- --without-local-prefix --disable-install-libiberty --enable-lto --disable-libssp --enable-libitm --disable-bootstrap --with-system-zlib --with-linker-hash-style=gnu --enable-linker-build-id --with-ppl=no --with-cloog=no --enable-checking=release --enable-cheaders=c_global --without-isl --with-gxx-include-dir=/not/exist/usr/include/c++/10.2.0 --with-build-time-tools=/home/duxy/L5.10.9/build-xwayland/tmp/work/x86_64-nativesdk-pokysdk-linux/gcc-cross-canadian-aarch64/10.2.0-r0/recipe-sysroot-native/usr/aarch64-poky-linux/bin --with-sysroot=/not/exist --with-build-sysroot=/home/duxy/L5.10.9/build-xwayland/tmp/work/x86_64-nativesdk-pokysdk-linux/gcc-cross-canadian-aarch64/10.2.0-r0/recipe-sysroot --enable-poison-system-directories --disable-static --enable-nls --with-glibc-version=2.28 --enable-initfini-array --enable-__cxa_atexit
Thread model: posix
Supported LTO compression algorithms: zlib
gcc version 10.2.0 (GCC)
```

同样方法请自行安装用于 qt 开发的工具链。 安装两个工具链的时候， 请指定不同目录， 请勿使用相同目录，否则会出现文件相互覆盖情形。

3. 使用 Yocto 构建开发板镜像

3.1. 简介

Yocto 是一个开源的“umbrella”项目，意指它下面有很多个子项目，Yocto 只是把所有的项目整合在一起，同时提供一个参考构建项目 Poky，来指导开发人员如何应用这些项目，构建出嵌入式 Linux 系统。它包含 Bitbake、 OpenEmbedded-Core, 板级支持包，各种软件包的配置文件。

MYD-JX8MPQ 提供了符合 Yocto 的配置文件，帮助开发者构建出可烧写在 MYD-JX8MPQ 板上的 Linux 系统像。Yocto 还提供了丰富的开发文档资源，让开发者学习并定制自己的系统。由于篇幅有限，不能完整介绍 Yocto 的使用，请用户自行上网搜索。

本节适合需要对文件系统进行深度定制的开发者，希望从 Yocto 构建出符合 MYD-JX8MPQ 系列开发板的文件系统，同时基于它的定制化方法。初次体验使用或无特殊需要的开发者可以直接使用 MYD-JX8MPQ 已经提供的文件系统。

注意：构建 Yocto 不需要加载 2.2 节中的 SDK 工具链环境变量，请创建新 shell 或打开新的终端窗口。

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。由于 Yocto 构建前需要下载文件系统中所有软件包到本地，为了快速构建，MYD-JX8MPQ 已经把相关的软件打包好，可以直接解压使用，减少重复下载的时间。

3.2.1. 从光盘镜像获取源码压缩包

源码包在米尔开发包资料 *04-Sources/myd-jx8mp-yocto.tar.gz*。拷贝压缩包到用户指定目录，并解压 Yocto 源码包到工作目录 myd-jx8mp-yocto：

```
PC$:mkdir -p myd-jx8mp-yocto
PC$:tar -zxf myd-jx8mp-yocto.tar.bz2 -C myd-jx8mp-yocto
```

3.2.2. 通过 github 获取

目前 MYD-JX8MPQ 开发板的 BSP 源代码和 Yocto 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-JX8MPQ 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：

把 *03-tools/repo* 文件放到 ~/bin 目录，加上可执行权限，并加到 PATH 变量中：

```
PC$:   mkdir -p myd-jx8mp-yocto
PC$:   cd myd-jx8mp-yocto
PC$:   mkdir -p ~/bin
PC$:   curl https://mirrors.tuna.tsinghua.edu.cn/git/git-repo > ~/bin/repo
PC$:   chmod a+x ~/bin/repo
PC$:   export PATH=~/bin:${PATH}
export REPO_URL='https://mirrors.tuna.tsinghua.edu.cn/git/git-repo/'
repo init -u https://github.com/MYiR-Dev/myir-imx-manifest.git --no-clone-bundl
e --depth=1 -m myir-i.mx8m-5.10.9-1.0.0.xml -b i.MX8M-5.10-gatesgarth
repo sync
```

代码同步成功之后，同样在 myd-jx8mp-yocto 目录下得到 *04-Sources/myd-jx8mp-yocto.tar.gz* 源码包一样的目录内容。

3.3. 快速编译开发板镜像

在使用 Yocto 项目进行系统构建之前都需要先设置相应的环境变量，我们在构建 myir-image-full 之前需要使用米尔提供的 myir-setup-release.sh 脚本进行环境变量的设置，设置过程中会创建一个构建目录（如 build-xwayland）后续构建过程，以及输出文件都包含在这个目录。

1）查看 Yocto 源码包内容

当解压了 myd-jx8mp-yocto.tar.gz 文件，或者通过 repo 下载了文件后，会存生成如下文件，列出 myd-jx8mp-yocto 目录内容如下：

```
PC$:tree -a -L 1 myd-jx8mp-yocto
myd-jx8mp-yocto
├── myir-setup-release.sh -> sources/meta-myir/tools/myir-setup-release.sh
├── README -> sources/base/README
├── README-IMXBSP -> sources/meta-myir/README
├── .repo
├── setup-environment -> sources/base/setup-environment
└── sources
```

一般也建议客户先查看 README-IMXBSP 内容，有详细编译的方法。

2）指定 uboot DDR 的配置文件

核心板支持 2G/3G/4G DDR 配置，需要在 uboot 中指定对应的配置文件。

文件位于 sources/meta-myir/meta-bsp/conf/machine/myd-jx8mp.conf

```
UBOOT_CONFIG ??= "sd"
#UBOOT_CONFIG[sd] = "myd_jx8mp_2g_defconfig,sdcard"
UBOOT_CONFIG[sd] = "myd_jx8mp_defconfig,sdcard"
#UBOOT_CONFIG[sd] = "myd_jx8mp_4g_defconfig,sdcard"
SPL_BINARY = "spl/u-boot-spl.bin"
```

- myd_jx8mp_2g_defconfig： 2G DDR 配置
- myd_jx8mp_defconfig： 3GDDR 配置
- myd_jx8mp_4g_defconfig： 4GDDR 配置

默认编译 3GDDR, 将其他 2 个配置项前面加 # 进行注释，剩下为注释即为将要编译的版本。

3）执行环境变量设置脚本

脚本设置编译环境的语法如下：

```
EULA=1 DISTRO=fsl-imx-xwayland MACHINE=myd-jx8mp source sources/meta-myir/tools/myir-setup-release.sh -b build-xwayland
```

配置脚本执行完成后将进入 *build-xwayland* 目录下，在此目录下就可以开始构建系统。

注：编译 yocto 需要使用普通用户，不能使用 root 用户

4）构建 myir-image-full 镜像

这里编译时会有在网络下载第三方源码的 fetch 过程，如果网络不好经常出现 fetch 错误，所以建议将网盘上的 downloads 文件下载下来，在放置在同级目录，如这里是 myd-jx8mp-yocto 目录。 这里演示编译过程。

- **拷贝 downloads 文件到指定目录**

从网盘下载 downloads 文件，首先需要拷贝到对应目录（如：myd-jx8mp-yocto），如红色文件所示：

```
build-xwayland  myir-setup-release.sh  README-IMXBSP      sources
downloads       README                 setup-environment
```

downloads 文件也可以直接从网上拉取，但是拉取速度取决于用户的网速，所以一般建议还是用网盘的 downloads 文件。一般网上拉取文件的命令如下所示：

```
$: bitbake myir-image-full --runall=fetch
```

- **构建完整镜像**

download 文件拷贝完成之后，接着执行以下命令构建系统：

```
duxy@myir_server:~/myd-jx8mp-yocto/build-xwayland$ bitbake myir-image-full
NOTE: Your conf/bblayers.conf has been automatically updated.
Loading cache: 100% |
                                | ETA:  --:--:--
Loaded 0 entries from dependency cache.
WARNING: /home/duxy/myd-jx8mp-yocto/sources/poky/meta/recipes-graphics/vulkan/vulkan-demos_git.bb: Unable to get checksum for vulkan-demos SRC_URI entry 0001-Modify-parameter-in-vulkan-demo-computenbody.patch: file could not be found
```

```
WARNING: /home/duxy/myd-jx8mp-yocto/sources/poky/meta/recipes-graphics/vulkan/vulkan-demos_git.bb: Unable to get checksum for vulkan-demos SRC_URI entry 0001-Fix-bug-in-computeheadless-and-renderheadless.patch: file could not be found
Parsing recipes: 100% |##########################################################################################################################| Time: 0:01:51
Parsing of 3294 .bb files complete (0 cached, 3294 parsed). 4861 targets, 250 skipped, 1 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION           = "1.48.0"
BUILD_SYS            = "x86_64-linux"
NATIVELSBSTRING      = "ubuntu-18.04"
TARGET_SYS           = "aarch64-poky-linux"
MACHINE              = "myd-jx8mp"
DISTRO               = "fsl-imx-xwayland"
DISTRO_VERSION       = "5.10-gatesgarth"
TUNE_FEATURES        = "aarch64 armv8a crc cortexa53 crypto"
TARGET_FPU           = ""
meta
meta-poky            = "HEAD:943ef2fad8428f002850e3655a3312e13d0dcb2c"
meta-oe
meta-multimedia
meta-python          = "HEAD:ac4ccd2fbbb599d75ca4051911fcbaca39dbe6d7"
meta-freescale       = "HEAD:41d4f625c6db7a778f0f9a735c2cb48e023bc49b"
meta-freescale-3rdparty = "HEAD:b85d08a55cb833bfc4e8b5034ff804286c67620e"
meta-freescale-distro = "HEAD:11be3f01962df8436c5c7b0d61cd3dbd1b872905"
meta-bsp
meta-sdk
meta-ml              = "HEAD:2c4d66839a440c6b76b426dc6e4960a37165541c"
```

```
meta-nxp-demo-experience = "HEAD:c7263d9f3cc7bbf44e7164ffeda494cf283d3d
ec"
meta-browser        = "HEAD:ee3be3b5986a4aa0e73df2204a625ae1fe5df37e"
meta-rust           = "HEAD:53bfa324891966a2daf5d36dc13d4a43725aebed"
meta-clang          = "HEAD:61faae011fb95712064f2c58abe6293f0daeeab5"
meta-gnome
meta-networking
meta-filesystems    = "HEAD:ac4ccd2fbbb599d75ca4051911fcbaca39dbe6d7"
meta-qt5            = "HEAD:8d5672cc6ca327576a814d35dfb5d59ab24043cb"
meta-python2        = "HEAD:c43c29e57f16af4e77441b201855321fbd546661"

Initialising tasks: 100% |#########################################
##################################################################
##################| Time: 0:00:07
Sstate summary: Wanted 4083 Found 0 Missed 4083 Current 0 (0% match, 0% co
mplete)
NOTE: Executing Tasks
Currently  4 running tasks (156 of 9809)   1% |#
                                             |
0: m4-native-1.4.18-r0 do_configure - 19s (pid 26905)
1: binutils-cross-aarch64-2.35-r0 do_unpack - 16s (pid 29474)
```

5）构建过程错误解决

构建过程如果网络不好可能会出现以下错误提示，如下所示：

```
ERROR: Task (/home/duxy/myd-jx8mp-yocto/sources/meta-myir/meta-ml/recipes
-libraries/tensorflow-lite/tensorflow-lite_2.4.0.bb:do_configure) failed with exit co
de '1'
```

上面可能出错的原因一般是由于客户网络差，导致这个包的一些组件没下载完整，请按照以下方式解决。

- **查看出错包的编译目录**

执行以下命令可以查看出错包的编译目录：

```
bitbake -e tensorflow-lite | grep ^S=
```

```
S="/home/duxy/myd-jx8mp-yocto/build-xwayland/tmp/work/cortexa53-crypto-poky-linux/tensorflow-lite/2.4.0-r0/git "
```

- **屏蔽掉依赖下载**

进入该编译目录的下载目录，屏蔽掉依赖下载，修改如下红色标记：

```
PC$ cd /home/duxy/myd-jx8mp-yocto/build-xwayland/tmp/work/cortexa53-crypto-poky-linux/tensorflow-lite/2.4.0-r0/git/tensorflow/lite/tools/make /
PC$ vi download_dependencies.sh
#download_and_extract "${EIGEN_URL}" "${DOWNLOADS_DIR}/eigen" "${EIGEN_SHA}"
#download_and_extract "${GEMMLOWP_URL}" "${DOWNLOADS_DIR}/gemmlowp" "${GEMMLOWP_SHA}"
#download_and_extract "${RUY_URL}" "${DOWNLOADS_DIR}/ruy" "${RUY_SHA}"
#download_and_extract "${GOOGLETEST_URL}" "${DOWNLOADS_DIR}/googletest" "${GOOGLETEST_SHA}"
#download_and_extract "${ABSL_URL}" "${DOWNLOADS_DIR}/absl" "${ABSL_SHA}"
#download_and_extract "${NEON_2_SSE_URL}" "${DOWNLOADS_DIR}/neon_2_sse"
#download_and_extract "${FARMHASH_URL}" "${DOWNLOADS_DIR}/farmhash" "${FARMHASH_SHA}"
#download_and_extract "${FLATBUFFERS_URL}" "${DOWNLOADS_DIR}/flatbuffers" "${FLATBUFFERS_SHA}"
#download_and_extract "${FFT2D_URL}" "${DOWNLOADS_DIR}/fft2d" "${FFT2D_SHA}"
#download_and_extract "${FP16_URL}" "${DOWNLOADS_DIR}/fp16" "${FP16_SHA}"
#download_and_extract "${CPUINFO_URL}" "${DOWNLOADS_DIR}/cpuinfo" "${CPUINFO_SHA}"
#download_and_extract "${RE2_URL}" "${DOWNLOADS_DIR}/re2" ${RE2_SHA}
```

- **删除先前下载目录**

修改完下载依赖，接着用户可以删除先前还没下载完整的包：

```
PC$ rm downloads -rf
```

- **拷贝 download**

删除 download 之后，用户需要拷贝已经下载好的 download 压缩包到该目录，然后解压，文件在目录 *04-Source/tensorflow_lite-downloads.tar.gz*：

```
PC$: $ cp 04-Source/tensorflow_lite-downloads.tar.gz .
PC$: $ tar -zxf   tensorflow_lite-downloads.tar.gz
```

修改完成之后，重新构建系统，构建完成之后，编译出来的镜像在“*build-xwayland/tmp/deploy/images/myd-jx8mp* ”目录。

表 3-1.镜像文件列表

编译生成文件	描述
myir-image-full-myd-jx8mp.wic.bz2	完整镜像
myir-image-full-myd-jx8mp.tar.bz2	文件系统
myir-image-full-myd-jx8mp.ext4	EXT4 格式文件系统
Image	内核镜像
myd-jx8mp-base.dtb	设备树文件
imx-boot	uboot 镜像

生成的镜像文件按照 4.1 节更新即可。

3.4. 构建 SD 卡烧录器镜像

为满足生产烧录镜像到 eMMC 的需要，米尔开发了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。这里已经把烧录器镜像资源整合到 Yocto 项目中，所以用户可以利用 yocto 项目构建出烧录器镜像。

1）执行环境变量设置脚本

脚本设置编译环境的语法如下：

```
EULA=1 DISTRO=fsl-imx-xwayland MACHINE=myd-jx8mp source sources/meta-myir/tools/myir-setup-release.sh -b build-xwayland
```

2）构建 myir-image-full 镜像

由于 sd 卡烧录器镜像就是把完整系统（myir-image-full），拷贝到某些目录，所以构建 sd 卡烧录器镜像之前需要先构建 myir-image-full 镜像，构建详情请参考 3.3 章节。

3）构建 sd 卡烧录器镜像

- **修改版本号**

修改 *sources/meta-myir/meta-bsp/recipes-myir/tf-upgrade/tf-upgrade/info.txt* 文件 BOARD_VERDINON 版本，如下：

```
BOARD_MANUFACTURER=MYIR
BOARD_VERSION=1.2
```

注意：BOARD_VERSION 值必须大于 1 的任意值。

- **打开自动重启**

修改 sources/meta-myir/meta-bsp/recipes-myir/fac-burn-emmc-full/fac-burn-emmc-full/home/root/burn_emmc.sh

```
if [ x"$HOSTNAME" == x"$MYS_NAME" ];then
  reboot
fi
```

去掉 if fi,保留 reboot

- **构建镜像**

执行以下命令，构建 sd 卡烧录器镜像：

```
bitbake -c cleansstate fac-burn-emmc-full
bitbake myir-image-full
```

```
bitbake myir-image-burn
```

生成镜像文件为 *tmp/deloy/images/myd-jx8mp/myir-image-burn-myd-jx8mp.wic.bz2*，然后按照 4.2 节更新即可。

- **查看版本**

按照 4.2 节更新完成后，开发板会自动启动，开发板查看版本，如果 BOARD_VERDINON 为修改后的版本号即更新成功。

```
root@myd-jx8mp:~# cat /upgrade/info.txt
BOARD_MANUFACTURER=MYIR
BOARD_VERSION=1.2
```

注意：系统挂载 sd 卡中待升级的镜像分区，然后每 3 秒检测一次该镜像的 BOARD_VERSION 的值，如果低于或等于当前系统/upgrade/info.txt 文件的 BOARD_VERSION 值，则不会重启升级，如果大于当前系统 BOARD_VERSION 值则会重启升级，所以用户如果需要重新升级，只需删除当前系统/upgrade/info.txt 文件。

3.5. 构建 SDK

Yocto 提供可构建出 SDK 工具链功能，用于底层或上层应用开发者使用的工具链和相关的头文件或库文件，免去用户手动制作或编译依赖库。用于编译 u-boot 和 linux 内核代码，附带目标系统的头文件和库文件，方便应用开发者移植应用在目标设备上。SDK 工具都是 shell 自解压文件，执行后，默认安装在/opt 目录下。下面讲解如何构建工具链 SDK。

本节只简单对米尔提供的 SDK 做构建说明，使用如下构建命令生成带 QT 的 SDK 包：

```
PC$ bitbake -c populate_sdk meta-toolchain-qt5
```

或者使用下面命令生成不带 QT 的工具链：

```
PC$ bitbake -c populate_sdk meta-toolchain
```

等待构建完成后，将在“*build-xwayland/tmp/deploy/sdk*”路径下生成 SDK 安装包，安装方法请查看 2.2 节。

```
duxy@myir-server1:~/L5.10.9/imx-yocto-bsp/build-xwayland$ ls tmp/deploy/sdk/
fsl-imx-xwayland-glibc-x86_64-meta-toolchain-cortexa53-crypto-myd-jx8mp-toolchain-5.10-gatesgarth.host.manifest
fsl-imx-xwayland-glibc-x86_64-meta-toolchain-cortexa53-crypto-myd-jx8mp-toolchain-5.10-gatesgarth.target.manifest
fsl-imx-xwayland-glibc-x86_64-meta-toolchain-cortexa53-crypto-myd-jx8mp-toolchain-5.10-gatesgarth.testdata.json
fsl-imx-xwayland-glibc-x86_64-meta-toolchain-cortexa53-crypto-myd-jx8mp-toolchain-5.10-gatesgarth.sh
fsl-imx-xwayland-glibc-x86_64-meta-toolchain-qt5-cortexa53-crypto-myd-jx8mp-toolchain-5.10-gatesgarth.host.manifest
fsl-imx-xwayland-glibc-x86_64-meta-toolchain-qt5-cortexa53-crypto-myd-jx8mp-toolchain-5.10-gatesgarth.target.manifest
fsl-imx-xwayland-glibc-x86_64-meta-toolchain-qt5-cortexa53-crypto-myd-jx8mp-toolchain-5.10-gatesgarth.testdata.json
```

4. 如何烧录系统镜像

米尔公司设计的 MYD-JX8MPQ 系列开发板是基于 NXP 公司 的 i.MX 8M Plus 系列微处理器，其启动方式多样，所以需要不同的更新工具与方法。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 TF 卡启动器：适用于研发调试，快速启动等场景。
- 制作 TF 卡烧录器：适用于批量生产烧写 emmc。
- UUU 烧录：适用于研发调试，测试等场景。

另外由于烧写时需要调整启动方式，用户根据下表选择配置拨码开关。

表 4-1.拨码启动方式

启动模式	SW1 拨码（1/2/3/4）	备注
TF Card 启动	OFF/OFF/ON/ON	
eMMC 启动	OFF/OFF/ON/OFF	
USB Download	OFF/OFF/OFF/ON	

4.1. UUU 烧录

1）工具需求

- 开发板一块
- Type_C 一根
- UUU 烧录文件包（*03-Tools/tools/MYD_JX8MP*_UUU.zip），UUU 工具无法兼容 win7，请使用 win10 系统。

2）设置硬件

选择启动模式,将拨码开关拨到 USB 下载模式 SW1(OFF,OFF,OFF,ON)。然后 连接好硬件，将开发板的 J11 TYPEC 接口与电脑连接。插入 12V 电源适配器。

3）在 windows 下烧录系统

- **准备烧录镜像**

Yocto 构建完成后，生成镜像列表请参考 表 3-1.镜像文件列表。

- **拷贝编译的镜像到 UUU 工具目录**

首先拷贝 03-Tools/tools/MYD_JX8MP_UUU.zip 工具到 windows 目录，解压，并拷贝相应镜像到 MYD_JX8MP_UUU 目录（出厂默认已经拷贝）：

名称	修改日期	类型	大小
8E2D	2022/10/20 16:47	文件夹	
8E3D	2022/10/20 16:47	文件夹	
8E4D	2022/10/20 16:49	文件夹	
EULA.txt	2020/12/21 9:51	文本文档	37 KB
GPLv2	2020/12/21 9:51	文件	19 KB
myd-jx8mp_8E2D.vbs	2022/10/20 16:52	VBScript Script ...	1 KB
myd-jx8mp_8E3D.vbs	2022/10/20 16:52	VBScript Script ...	1 KB
myd-jx8mp_8E4D.vbs	2022/10/20 16:52	VBScript Script ...	1 KB
myir_readme.txt	2021/10/21 15:25	文本文档	1 KB
myir-image-full-myd-jx8mp.wic	2021/11/30 9:20	WIC 文件	5,252,834...
QUALCOMM_ATHEROS_LICENSE_AG...	2020/12/21 9:51	Microsoft Edge ...	173 KB
README.uuu	2020/12/21 9:55	UUU 文件	1 KB
uuu.exe	2021/1/14 9:02	应用程序	1,271 KB
uuu_8E2D.auto	2022/10/20 16:50	AUTO 文件	1 KB
uuu_8E3D.auto	2022/10/20 16:51	AUTO 文件	1 KB
uuu_8E4D.auto	2022/10/20 16:51	AUTO 文件	1 KB

图 4-1.UUU 工具内容

- **烧录镜像**

根据 DDR 大小，双击 UUU 工具目录下的 myd-jx8mp_8END.vbs 文件开始烧录镜像，如下显示：

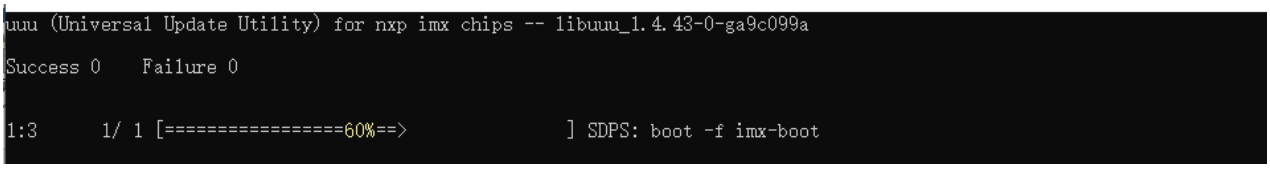

图 4-2.UUU 烧录中

烧录完成后将拨码开关拨到 eMMC 启动模式 SW1(OFF,OFF,ON,OFF) 即可 eMMC 启动。

4.2. 制作 TF 卡启动器

1）准备工作

此步骤均在 Windows 系统下制作。

- TF 卡（不少于 8GB）
- MYD-JX8MP 完整镜像
- 制作镜像工具 Win32DiskImager-1.0.0-binary.zip （路径：*/03-Tools/tools*）

表 4-2.镜像包列表

镜像名	包名
myir-image-full	myir-image-full-myd-jx8mp.wic

2）制作 TF 卡启动（以 myir-image-full 系统为例）

- **解压完整镜像**

window 直接右键解压出来。

Win32DiskImager-1.0.0-binary.zip
myir-image-full-myd-jx8mp.wic.bz2

- **将镜像文件写入 Micro SD Card**

将 TF 卡使用读卡器插入电脑，双击打开 Win32DiskImager.exe 读出 U 盘分区，点击箭头方向加载镜像文件。

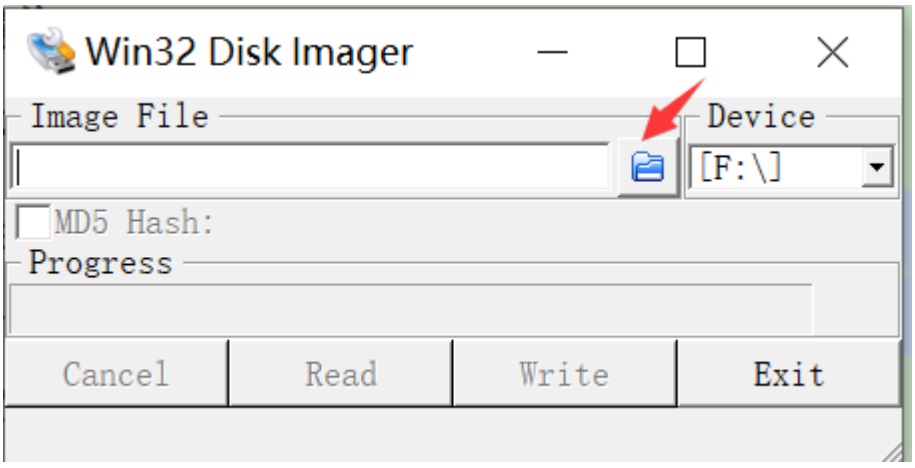

图 4-3.工具配置

选择好系统包后点击打开即可，如下图红色箭头所示。

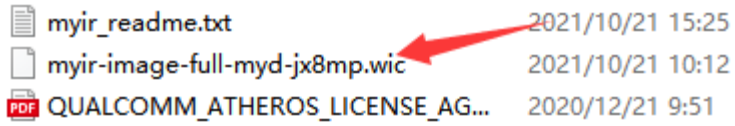

图 4-4.选择镜像

加载完镜像后点击“Write”按钮即可，会弹出警告，点击“Yes”等待写入完成。

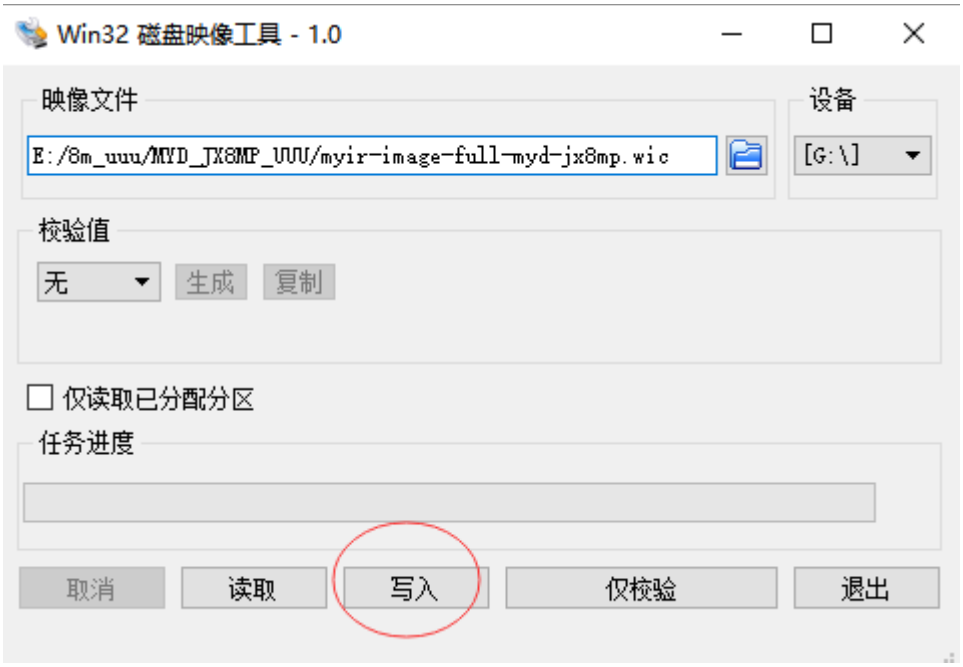

图 4-5. 烧写指示

等待写入完成，大约 3-4 分钟完成，此速度取决于 TF 的读写速度。

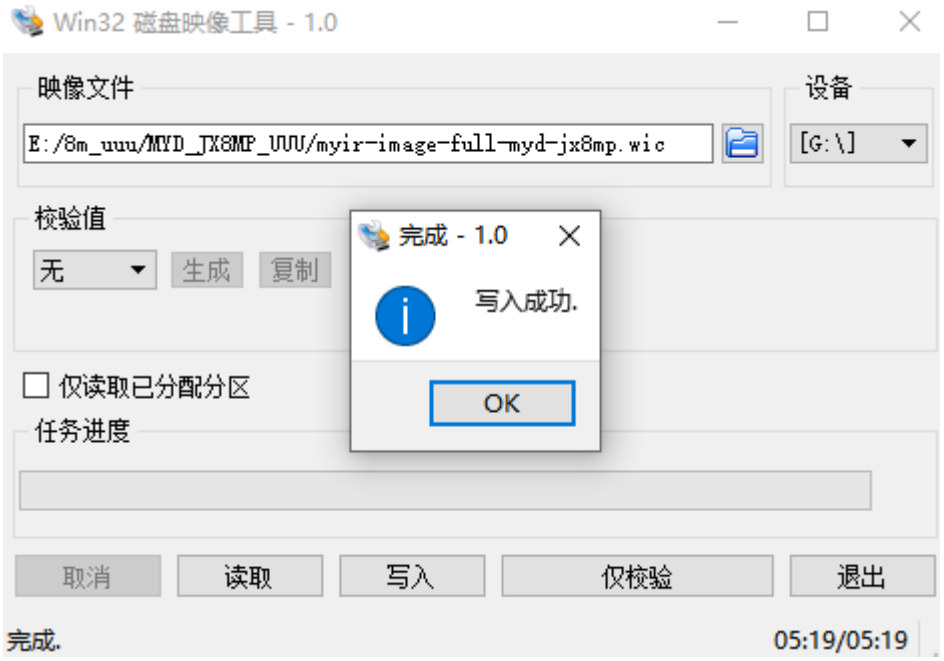

图 4-5. 烧写完成

- **检查是否烧写成功。**

当写入完成后，即可使用此 TF 进行启动，将 TF 插入开发板 TF 卡槽；并将拨码开关调至 SW1(OFF/OFF/ON/ON)即可启动系统。

4.3. 制作 TF 卡烧录器

为满足生产烧录的需要，米尔开发了适用于大批量生产的烧录方法。具体制作过程请按照下列步骤完成。

1）制作 sd 卡烧录器镜像

制作 TF 卡烧录器镜像(myir-image-burn-myd-jx8mp.wic)，详情请查看 3.4 节。

2）将烧录器镜像烧写进 TF 卡

制作 TF 卡启动器即将 sd 卡烧录镜像写进 TF 卡，请参考 4.2 节。

3）烧录镜像到 eMMC

设置 eMMC 启动 SW1(OFF,OFF,ON,OFF)，将 TF 插入开发板的 TF 卡槽，插上电源。利用里面得 tf-upgrade 服务，只要 TF 卡内部版本大于 EMMC 版本就会自动开始升级 SD 卡中/home/root/mfgimage 中得镜像到 EMMC。

如果 EMMC 中没有开启 tf-upgrade 服务，还可以用以下命令破坏 eMMC 的内容：

```
dd if=/dev/zero of=/dev/mmcblk2 bs=1M count=5;mmc bootpart enable 0 1 /dev/mmcblk2
```

然后按复位键或者重新上电，由于此时 eMMC 内容已经破坏，将会自动从 TF 卡启动，然后自动烧写镜像到 eMMC。

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 Yocto 项目构建运行在 MYD-JX8MPQ 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。本节将对 MYIR 创建的层级 BSP 进行说明，而且还描述如何通过独立环境和 Yocto 项目编译内核、U-Boot，以及如何下载内核和 uboot。

5.1. meta-bsp 层介绍

Yocto 项目的“层模型”是一种用于嵌入式和物联网 Linux 创建的开发模型，它将 Yocto 项目与其他简单的构建系统区别开来。层模型同时支持协作和定制。层是包含相关指令集的存储库，这些指令集告诉 OpenEmbedded 构建系统应该做什么。

meta-myir 层基于在 NXP 官方的 meta-imx 层建立的适用于 MYD-JX8MPQ 开发板的层，其下面的 meta-bsp 层中包含 BSP、GUI、发行版配置、中间件或应用程序的各种元数据和配方。用户可以在这个“层模型”的基础上适配基于 MYD-JX8MPQ 开发板设计的硬件，定制自己的应用，从而构建适合自己的系统镜像，本节主要介绍 meta-myir 层下面的 meta-bsp 层，其包含的具体内容如下：

```
PC$ tree -a -L 1 meta-bsp/
meta-bsp
├── classes
├── conf
├── recipes-bsp
├── recipes-connectivity
├── recipes-core
├── recipes-daemons
├── recipes-devtools
├── recipes-fsl
├── recipes-graphics
├── recipes-kernel
├── recipes-multimedia
├── recipes-myir
├── recipes-security
├── recipes-support
```

```
├── recipes-test
└── recipes-utils

16 directories, 0 files
```

部分层说明：

表 5-1.meta-bsp 层内容说明

源代码与数据	描述
conf	包含当前层路径信息和机器软件配置
recipes-bsp	包含 atf、uboot 以及固件等配置信息
recipes-kernel	包含 linux 内核的资源与第三方固件资源
recipes-myir	包含文件系统的配置信息
recipes-daemons	包含 proftpd 应用服务信息

在进行系统移植时，需要重点关注的是负责硬件初始化和系统引导的 recipes-bsp 部分，负责 Linux 系统的内核和驱动实现的 recipes-kernel 部分。

5.2. 板级支持包介绍

为了适配用户新的硬件平台，首先需要了解米尔的 MYD-JX8MPQ 开发板提供了哪些资源，具体的信息可以查看《MYD-JX8MPQ SDK 发布说明》。除此之外我们也对 BSP 各个部分中需要改动的一些文件列表整理出来，方便用户查找和修改，具体内容如下表所示：

表 5-2.添加配置信息

项目	设备树	说明
U-boot	arch\arm\dts\myd-jx8mp-base.dts arch\arm\dts\myd-jx8mp-base-u-boot.dtsi arch\arm\dts\myd-jx8mp.dtsi	资源设备树
Kernel	arch\arm64\configs\myd_jx8mp_defconfig	内核配置表
	arch\arm64\boot\dts\myir\myd-jx8mp.dtsi	资源设备树
	arch\arm64\boot\dts\myir\myd-jx8mp-base.dts	HDMI 显示
	arch\arm64\boot\dts\myir\myd-jx8mp-rpmsg.dts	M7 设备树
	arch\arm64\boot\dts\myir\myd-jx8mp-lt8912.dts	MIPI-DSI 转 HDMI
	arch\arm64\boot\dts\myir\myd-jx8mp-hontron-7.dts	7 寸屏显示

MYD-JX8MPQ 的正常启动过程如下：

- Bootrom：是嵌入处理器芯片内的一小块 ROM 或写保护闪存。它包含处理器在上电或复位时执行的第一个代码。根据某些带式引脚或内部保险丝的配置，它可以决定从哪里加载要执行的代码的下一部分以及如何或是否验证其正确性或有效性。
- ATF:ARM 可信固件为 ARMv8-a 提供了安全世界软件的参考实现，包括在异常级别 3（EL3）执行的安全监视器。实现了各种 ARM 接口标准，如：电源状态协调接口（PSCI）受信任的板引导要求（TBBR，ARM DEN0006C-1）SMC 调用、系统控制和管理接口。
- UBOOT SPL：SPL 的主要作用是初始化外部内存，然后把 Uboot 拷贝到外部内存中运行，所以它自己是运行在内部内存中的 。
- UBOOT：建立了适当的系统软硬件环境,为最终调用操作系统内核做好准备。

下面章节重点讲述用户基于我们提供的 Bootloader, Kernel 和 Yocto 源代码和数据进行的流程。

5.3. 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

i.MX8M Plus 平台使用系列使用 SPL 启动，不同的 boot chains 模式会对应不同启动阶段。spl 的编译是编译 uboot 的一部分，和 uboot.bin 走的是两条编译流程，这个要重点注意。正常来说，会先编译主体 uboot，也就是 uboot.bin 再编译 uboot-spl，也就是 uboot-spl.bin,虽然编译命令是一起的，但是编译流程是分开的。下面就介绍整个 u-boot 的编译。

5.3.1. 获取 u-boot 源代码

1）通过 github 获取源码

进入 yocto 工作目录 myd-jx8mp-yocto，然后使用如下命令下载源码，即放置在 myd-jx8mp-yocto.tar.gz 解压后的同级目录：

```
PC$:cd myd-jx8mp-yocto
PC$:git clone https://github.com/MYiR-Dev/myir-imx-uboot.git -b develop_2020.04
```

2）拷贝 Yocto 项目下的源码

用户构建完系统后(详情请参考 3.3 节)，会从 github 拉取 uboot 源码到本地，用户只需拷贝到 MYD-JX8MP-yocto.tar.gz 解压后的同级目录,然后重名，如下所示：

```
PC$:cd build-xwayland/tmp/work/myd_jx8mp-poky-linux/u-boot-imx/1_2020.04-r0/
PC$:cp -r  git/  <PATH>/MYD-JX8MP-yocto/
PC$:mv git myir-imx-uboot
```

其中“PATH”为用户自己的工作目录。

5.3.2. 在 Yocto 项目下编译 u-boot

当用户修改 U-boot 的代码之后，也可以使用 Yocto 进行整个镜像的构建。此时需要将修改后的源代码提交到本地 git 仓库，同时修改元层的对应源代码的拉取地址（SRC_UR）和 commit 值（SRCREV）。以便配方能够找到并获取本地 uboot 源码，参考示例如下。

1）查看 commit 值

进入 uboot 源码修改 uboot 源码，并查看 commit 值，如红色字符串所示：

```
PC$ cd myir-imx-uboot/
PC$ git log
commit 628a6631f8f876e50a4d3b9c6ae583aac4ec9a7c (HEAD -> develop_2020.0
4, origin/develop_2020.04)
Author: duxy <568988005@qq.com>
Date:   Tue Sep 28 08:48:43 2021 +0800

    FIX: uboot mac size from 12 -> 6

commit b48bbb090a8044ca65adac6d87d351f75d6a203b
Author: duxy <568988005@qq.com>
Date:   Mon Sep 27 18:37:18 2021 +0800

    FEAT: read the offset 31M info
```

2）修改源码位置

yocto 中指定 uboot 源码的位置在 *sources/meta-myir/meta-bsp/recipes-bsp/u-boot/ u-boot-imx_2020.04.bb* 文件，修改如下红色显示：

```
#UBOOT_SRC ?= "git://github.com/MYiR-Dev/myir-imx-uboot.git;protocol=https"
UBOOT_SRC ?= "git:///${PWD}/../myir-imx-uboot;protocol=file"
SRCBRANCH = "develop_2020.04"
SRC_URI = "${UBOOT_SRC};branch=${SRCBRANCH} \
"
SRCREV = "628a6631f8f876e50a4d3b9c6ae583aac4ec9a7c"
```

- UBOOT_SRC: uboot 代码下载位置
- SRCBRANCH: 分支名称
- SRCREV: commit 对应值

3）初始化 SDK 环境变量

```
EULA=1 DISTRO=fsl-imx-xwayland MACHINE=myd-jx8mp source sources/meta-myir/tools/myir-setup-release.sh -b build-xwayland
```

4）编译 uboot

执行下面的命令即编译 uboot：

```
bitbake -c cleansstate u-boot
bitbake -c cleanstate imx-boot
bitbake imx-boot
```

生成镜像文件为 *tmp/deloy/images/MYD-JX8MP/imx-boot*。

5）确认 uboot 是否更新

当修改完 uboot 之后，需要确认 uboot 是否已经更新，执行以下命令：

```
strings tmp/deploy/images/myd-jx8mp/imx-boot | grep 2021
U-Boot SPL 2020.04-5.10.9-1.0.0+g628a6631f8 (Sep 28 2021 - 00:48:43 +0000)
09/28/2021
U-Boot 2020.04-5.10.9-1.0.0+g628a6631f8 (Sep 28 2021 - 00:48:43 +0000)
Built : 08:27:48, Mar  1 2021
```

由上面“+g628a6631f8”字符串可知，修改后的 uboot 版本与 SRCREV 前面字符一致，即 uboot 已经更新。

5.3.3. 如何单独更新 U-boot

1）烧录镜像到 sdcard

将 TF 卡用读卡器插入主机，查看对应分区：

```
PC$ cat /proc/partitions
major minor  #blocks  name
.......
 8       48   15273984 sdd
 8       49      85196 sdd1
 8       50    3510437 sdd2
```

执行下面命令，把 uboot 镜像烧录到指定分区：

```
sudo dd if=imx-boot of=/dev/sdd bs=1k seek=32
sync
```

2）烧录镜像到 eMMC

拷贝编译的镜像（imx-boot）到开发板，查看对应分区：

```
root@myd-jx8mp:~# cat /proc/partitions
major minor  #blocks  name
.....
 179       0    7636800 mmcblk2
 179       1      85196 mmcblk2p1
 179       2    3510437 mmcblk2p2
```

执行下面命令，把镜像拷贝到对应磁盘：

```
root@myd-jx8mp:~# echo 0 > /sys/block/mmcblk2boot0/force_ro
root@myd-jx8mp:~# dd if=imx-boot of=/dev/mmcblk2boot0
1573+1 records in
1573+1 records out
1610920 bytes (1.6 MB, 1.5 MiB) copied, 0.0733266 s, 22.0 MB/s
```

5.4. 板载 Kernel 编译与更新

5.4.1. 编译 Linux

1）通过 github 获取源码

进入 yocto 工作目录 myd-jx8mp-yocto，然后使用如下命令下载源码，即放置在 myd-jx8mp-yocto.tar.gz 解压后的同级目录：

```
PC$:cd myd-jx8mp-yocto
PC$:git clone https://github.com/MYiR-Dev/myir-imx-linux.git -b develop_lf-5.10.y
```

2）拷贝 Yocto 项目下的源码

用户构建完系统后(详情请参考 3.3 节)，会从 github 拉取 Linux 源码到本地，用户只需拷贝到 myd-jx8mp-yocto.tar.gz 解压后的同级目录,然后重命名，如下所示：

```
PC$:cd build-xwayland/tmp/work-shared/myd-jx8mp/
PC$:cp -r   kernel-source/  <PATH>/myd-jx8mp-yocto/
PC$:mv  kernel-source myir-imx-linux
```

其中“PATH”为用户自己的工作目录。

5.4.2. 在独立的交叉编译环境下编译 Kernel

1）加载 SDK 环境变量到当前 shell

单独编译 Kernel 之前需要先申明编译链，具体方法请参考 2.2 节。

```
PC$ source ~/opt_5_10/environment-setup-cortexa53-crypto-poky-linux
```

2）进入 Kernel 源码，配置并编译

- **整体编译**

进入 Kernel 源码，使用下面命令配置并编译源码：

```
PC$ make distclean
PC$ make myd_jx8mp_defconfig
PC$ LDFLAGS="" CC="$CC" make -j16
PC$ mkdir test
PC $ make modules_install INSTALL_MOD_PATH=./test
```

生成的 Image 在 *arch/arm64/boot/Image*；生成的 dtb 文件在 *arch/arm64/boot/dts/myir /myd-jx8mp*.dtb*。生成得模块在 test 目录，里面有一个对 kernel 源码链接需要删掉后才能拷贝到开发板。

- **单独编译设备树**

```
make myd_jx8mp_defconfig
make dtbs
```

生成的 dtb 文件在 arch/arm64/boot/dts/myir/myd-jx8mp*.dtb。

5.4.3. 在 Yocto 项目下编译 Kernel

当用户修改 Kernel 代码之后，也可以使用 Yocto 进行整个镜像的构建。此时修改元层的对应源代码的拉取地址（SRC_URI）和 commit 值（SRCREV）。以便配方能够找到并获取本地 Linux 源码，参考示例如下。

1）查看 commit 值

进入源码目录，修改 Kernel 源码，并查看 commit 值，如红色字符串所示：

```
PC$ cd myir-imx-linux/
PC$ git log
commit 4fef21b3cb2bf3cc476578bc5f3d8ede295f02e9 (HEAD -> develop_lf-5.10.
y, origin/develop_lf-5.10.y)
    Author: duxy <568988005@qq.com>
    Date:   Tue Oct 12 16:54:05 2021 +0800

        FIX: set usb0 default mode for host

    commit c490fb475331979715f7f8a5f10d04969bbf5d73
    Author: duxy <568988005@qq.com>
    Date:   Fri Oct 8 17:33:50 2021 +0800

        FEAT: for factory , set spi gpio to general mode

    commit 09c9e24452c029a922864a1012a30f7e63f7e2fa
    Author: duxy <568988005@qq.com>
    Date:   Wed Sep 29 09:35:50 2021 +0800

        FEAT: add RFKILL support
```

2）修改源码位置

yocto 中指定 kernel 源码的位置在 *sources/meta-myir/meta-bsp/recipes-kernel/linux/ linux-imx_5.10.bb* 文件，然后修改如下红色显示：

```
KERNEL_SRC ?= "git://// /home/duxy/myd-jx8mp-yocto/myir-imx-linux;protocol=
file"
```

```
SRCBRANCH = " develop_lf-5.10.y "
SRC_URI = "${KERNEL_SRC};branch=${SRCBRANCH}"
SRCREV = "4fef21b3cb2bf3cc476578bc5f3d8ede295f02e9 "
```

- KERNEL_SRC:Kernel 代码下载位置
- SRCBRANCH: 分支名称
- SRCREV: commit 对应值

3) 初始化 SDK 环境变量

如果是初次编译需要执行如下命令初始化环境：

```
DISTRO=fsl-imx-xwayland MACHINE=myd-jx8mp source sources/meta-myir/tools/myir-setup-release.sh -b build-xwayland
```

4) 编译 Kernel

执行下面的命令即编译 kernel

```
bitbake -c cleansstate virtual/kernel
bitbake virtual/kernel
```

生成镜像文件为 *build-xwayland/tmp/deploy/images/myd-jx8mp/Image*。

5) 确认 Kernel 是否更新

当修改完 Kernel 编译之后，需要确认 kernel 是否已经更新，执行以下命令：

```
duxy@myir_server:~/myd-jx8mp-yocto/build-xwayland$ strings tmp/deploy/images/myd-jx8mp/Image | grep 2021
Linux version 5.10.9-1.0.0+g4fef21b3cb2b (oe-user@oe-host) (aarch64-poky-linux-gcc (GCC) 10.2.0, GNU ld (GNU Binutils) 2.35.0.20200730) #1 SMP PREEMPT Tue Oct 12 08:54:05 UTC 2021
#1 SMP PREEMPT Tue Oct 12 08:54:05 UTC 2021
```

由上面“+g4fef21b3cb2b ”字符串可知，修改后的 Kernel 版本与 SRCREV 前面字符一致，即 Kernel 已经更新。

5.4.4. 如何单独更新 Kernel 和设备树

1）烧录镜像到 TF 卡

将 TF 卡用读卡器插入主机，boot 分区会自动挂载到主机(fat 格式)，拷贝内核和设备树到此目录即可：

```
PC$ cp -rf  *.dtb Image  <boot_partition>
```

2）烧录镜像到 eMMC

拷贝编译的镜像（Image）和设备树文件（*.dtb）到开发板 boot 分区，可以用 scp 或者 U 盘拷贝。由于启动开发板后 boot 会自动挂载，执行以下命令查看 boot 挂载分区：

```
root@myd-jx8mp:~# df -h
.......
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           195M  4.0K  195M   1% /run/user/0
/dev/mmcblk2p1   84M   35M   49M  42% /run/media/mmcblk2p1
```

可知，boot 分区会自动挂载在 /run/media/mmcblk2p1 目录，直接把镜像拷贝到该目录：

```
root@myd-jx8mp:~#  cd /run/media/mmcblk2p1/
root@myd-jx8mp:~# cp -rf  *.dtb  Image ./
root@myd-jx8mp:~#   sync
```

6. 如何适配您的硬件平台

6.1. 如何创建自己的 machine

在开发过程中，用户有时需要创建自定义板配置。本节将通过一个实例讲解用户如何创建属于自己的 machine。

6.1.1. 在 Yocto 创建一个板子配置

1）选择类似 machine 文件

复制一个类似的 machine 文件，并重命名为一个你板子的指定名字，如和 myd-jx8mp 类似的 machine 文件（myd-jx8mp.conf）在 *sources/meta-myir/meta-bsp/conf/machine/* 这个目录，进入这个目录，machine 文件如下：

```
PC$ cd sources/meta-myir/meta-bsp/conf/machine/
PC$ ls
myd-jx8mp.conf
```

2）复制并重命名

找到类似的 machine 文件后，复制并重名为你自己的 machine 文件，如：

```
PC$ cp myd-jx8mp.conf test-jx8mp.conf
PC$ ls
myd-jx8mp.conf  test-jx8mp.conf
```

3）在 READEME 添加相关编译描述

创建好 machine 文件后，您需要在 README 文件添加相关的编译指导，进入 Yocto 工作目录 myd-jx8mp-yocto，查看并修改 REDME-IMXBSP 文件，如下红色字符串所示：

```
duxy@myir_server:~/myd-jx8mp-yocto$ ls -l
总用量 32
. . . . . .
lrwxrwxrwx  1 duxy duxy   19 10 月 20 13:03 README -> sources/base/README
lrwxrwxrwx  1 duxy duxy   24 10 月 20 13:03 README-IMXBSP -> sources/meta-m
yir/README
. . . . . .
```

```
duxy@myir_server:~/myd-jx8mp-yocto$cat README-IMXBSP

Quick Start Guide
-----------------
Build images
--------------------
Building XWayland
--------------------------
 DISTRO=fsl-imx-xwayland MACHINE=myd-jx8mp source sources/meta-myir/tools/myir-setup-release.sh -b build-xwayland
 DISTRO=fsl-imx-xwayland MACHINE=test-jx8mp source sources/meta-myir/tools/myir-setup-release.sh -b build-xwayland
```

4）编译并测试

当 machine 文件创建完成之后，您可以通过编译并下载测试，执行如下命令编译最小镜像测试：

```
 PC$ DISTRO=fsl-imx-xwayland MACHINE=test-jx8mp source sources/meta-myir/tools/myir-setup-release.sh -b build-xwayland
PC$ bitbake core-image-minimal
```

编译完成后生成的镜像 build-xwayland*/tmp/deploy/images/test-jx8mp/core-image-minimal-test-jx8mp.sdcard.wic.bz2*，拷贝并解压按照 4.2 节烧写到 TF 卡，启动测试：

```
root@test-jx8mp:~#
root@test-jx8mp:~#
```

6.1.2. 在 uboot 创建板子配置文件

在开发过程中，用户一般需要根据自己的板子要求创建属于自己的板子配置文件，本小节将通过一个简单实例演示如何一步步创建自己的板子配置文件。

1）创建 board

在创建自己的 board 配置时，用户可以在相对应的板子目录通过复制并重命名建立自己的板子配置文件，一般板子配置文件都在 uboot 源码的 board 目录。进入进入 uboot 源码 board 子目录下的 myir 子目录，复制 myd_jx8mp 文件夹为 test_jx8mp，如下所示：

```
duxy@myir_server:~/myd-jx8mp-yocto/myir-imx-uboot/board/myir$ ls
common  myd_jx8mp  test_jx8mp
```

进入 test_jx8mp 目录，修改 myd_jx8mp.c 为 test_jx8mp.c;修改 Makefile，将 myd_jx8mp.o 修改为 test_jx8mp.o；

2）制作板子.config 文件

- **修改新创建的板子 Kconfig**

进入 mys_iot_imx8mm_test 目录，修改 Kconfig 文件如下红色标记：

```
if  TARGET_TEST_JX8MP_LPDDR4_EVK

config SYS_BOARD
	default "test_jx8mp"

config SYS_VENDOR
	default "myir"

config SYS_CONFIG_NAME
	default "test_jx8mp"

source "board/myir/common/Kconfig"

endif
```

接着在 *arch/arm/mach-imx/imx8m/Kconfig* 文件增加如下红色标记内容（这个会影响 make menuconfig 配置）：

```
config TARGET_TEST_JX8MP_LPDDR4_EVK
        bool "test jx8mp lpddr4 evk board"
        select IMX8MP
        select SUPPORT_SPL
        select IMX8M_LPDDR4
 endchoice
 ..............
source "board/myir/test_jx8mp/Kconfig"
```

- **创建新板子的头文件**

从源码根目录下进入 include/configs 目录，复制 myd_jx8mp.h 为 test_jx8mp.h，如下：

```
duxy@myir_server:~/myd-jx8mp-yocto/myir-imx-uboot/include/configs$ ls -l myd_jx8mp.h
-rwxr-xr-x 1 duxy duxy 8269 10 月 22 11:16 myd_jx8mp.h
duxy@myir_server:~/myd-jx8mp-yocto/myir-imx-uboot/include/configs$ ls -l test_jx8mp.h
-rwxr--r-- 1 duxy duxy 8271 10 月 25 10:01 test_jx8mp.h
```

注意：由于 SYS_CONFIG_NAME 的名字改为了 test_jx8mp,所以此头文件改为 test_jx8mp.h。

- **定制系统板子配置文件**

从源码根目录下进入 configs 目录，复制 myd_jx8mp_defconfig 为 test_jx8mp_defconfig，如下：

```
duxy@myir_server:~/myd-jx8mp-yocto/myir-imx-uboot/configs$ cp myd_jx8mp_defconfig test_jx8mp_defconfig
duxy@myir_server:~/myd-jx8mp-yocto/myir-imx-uboot/configs$ ls -l myd_jx8mp_defconfig
-rwxr-xr-x 1 duxy duxy 3619 10 月 22 11:16 myd_jx8mp_defconfig
duxy@myir_server:~/myd-jx8mp-yocto/myir-imx-uboot/configs$ ls -l test_jx8mp_defconfig
-rwxr-xr-x 1 duxy duxy 3619 10 月 25 10:04 test_jx8mp_defconfig
```

然后修改 test_jx8mp_defconfig 文件，如下：

```
CONFIG_TARGET_TEST_JX8MP_LPDDR4_EVK=y
# CONFIG_TARGET_MYD_JX8MP_LPDDR4_EVK=y
```

经过这几个步骤基本上是已经定制好了板子，如果用户需要用 yocto 编译，那么还需要根据 6.1.1 节修改 machine 里面的设备树配置信息。

3）编译

执行以下命令，编译

```
make distclean
make test_jx8mp_defconfig
make
```

6.2. 如何创建您的设备树

6.2.1. 板载设备树层级的介绍

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。如将部分设备信息结构存放到 device tree 文件中。 uboot 或内核最终将其 device tree 编译成 dtb 文件，使用过程中通过解析该 dtb 来获取板级设备信息。uboot 的 dtb 和 kernel 中的 dtb 是一致的。

1）imx-myir-uboot 设备树介绍

下面 MYD-JX8MPQ 的 uboot 各部分设备树的信息列表，以便用户参考：

表 6-1. MYD-JX8MPQ U-boot 设备树列表

项目	设备树	描述
U-boot	myd-jx8mp-base.dts	MYD 板的 DTB 文件，直接包含 myd-jx8mp.dtsi
	myd-jx8mp.dtsi	板级主要文件，包括所有外设驱动部分
	myd-jx8mp-base-u-boot.dtsi	针对 myd-jxmp-base 得 dtsi 做专用修正

编译 myd-jx8mp-base.板 uboot 源码时会合并相关的所有 dts/dtsi 文件，生成 uboot 阶段默认使用的 myd-jx8mp-base.dtb 文件。

2）imx-myir-linux 设备树介绍

imx-myir-linux 的设备树层级关系为：myd-jx8mp.dtsi-> myd-jx8mp-base.dts-> myd-jx8mp-lt8912.dts 或者 myd-jx8mp-hontron-7.dts。下面是 myd-jx8mp 的各部分设备树的详细信息列表，以便用户开发参考：

表 6-2.myd-jx8mp Linux 设备树列表

项目	设备树	描述
Kernel	myd-jx8mp-hontron-7.dts	7 寸 LVDS 显示，包含 7 寸 LVDS 的配置信息以及触摸配置,J18

	myd-jx8mp-m190etn01-19.dts	19 寸 1280x1024 屏显示
	myd-jx8mp-base.dts	HDMI 显示
	myd-jx8mp-lt8912.dts	MIPI-DSI 转 HDMI 显示
	myd-jx8mp.dtsi	板级主要文件，包括所有外设驱动部分

6.2.2. 设备树的添加

1）Uboot 创建设备树

- **选择类似的设备树文件**

从源码根目录下进入 *arch/arm/dts* 目录，复制 myd-jx8mp-base.dts 为 test-jx8mp-base.dts，如：

```
PC$ cp myd-jx8mp-base.dts test-jx8mp-base.dts
PC$ cp myd-jx8mp-base-u-boot.dtsi test-jx8mp-base-u-boot.dtsi
```

- **修改设备树 Makefile**

接着修改改目录下的设备树 Makefile，增加内容如下：

```
.....
imx8mn-ab2.dtb \
myd-jx8mp-base.dtb \
test-jx8mp-base.dtb
.....
```

- **修改板子配置文件**

然后修改 test_jx8mp_defconfig 配置文件默认的设备树，如下：

```
CONFIG_CMD_LED=y
CONFIG_OF_CONTROL=y
CONFIG_DEFAULT_DEVICE_TREE="test-jx8mp-base"
```

2）Kernel 创建设备树

MYIR 系列设备树在 *arch/arm64/boot/dts/myir* 目录，用户在该目录也可以通过复制 MYIR 的设备树然重命名，方法过程如 uboot 类似，请参考以上步骤。

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，本节将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

MYD-JX8MPQ 板的 IO 管脚大部分定义在 arch/arm64/boot/dts/myir/myd-jx8mp-base.dts 设备树文件中。由于 NXP 已经把每一个 IO 管脚可能使用到的 IOmux 功能，以及与此功能相对应的 IOPAD 属性都已经准备好，定义在文件/arch/arm64/boot/dts/myir/ imx8mp-pinfunc.h 中，所以用户在配置的时候只需要直接在数组中包括我们想使用的 IO 管脚及功能的宏就可以了。每个引脚配置条目由 6 个整数组成，代表一个管脚的 mux 和 config 设置。前 5 个整数使用 NXP 已经在 imx8mp-pinfunc.h 定义好的宏指定，最后一个整数配置是此引脚上的 pad 设置值如上拉。请参考 iMX_8M_Plus_RM_RevC.pdf 芯片配置设置手册。下面将介绍一个配置的实例。

1）查看 pinctrl 配置规则

NXP pinctrl 配置格式：<mux_reg conf_reg input_reg mux_val input_val pad_val >

其中：

- mux_reg：复用寄存器偏移地址
- conf_reg：配置寄存器偏移地址
- input_reg：输入寄存器偏移地址
- mux_val: 复用寄存器值
- input_val:输入寄存器值
- pad_val:管脚速率、上下拉等的配置

例如引脚配置如下：

```
MX8MP_IOMUXC_SAI1_TXD7__GPIO4_IO19 0x1c4 /*VFAN_EN*/
```

在文件/arch/arm64/boot/dts/myir/ imx8mp-pinfunc.h 中找这个宏的定义如下：

```
#define MX8MP_IOMUXC_SAI1_TXD7__GPIO4_IO19                0x194 0x3F4 0x000 0x5 0x0
```

通过 datasheet 手册可知设置 GPIO4_IO19 为快速，驱动能力选择为 X2。综上所述，配置一个 pinctrl 涉及到 3 个寄存器，分别为复用寄存器、配置寄存器、输入寄存器。

2）在设备树上配置 gpio

下面将使用 DTS 文件来进行设备硬件资源的申请及分配，用户可以在 myd-jx8mp-base.dts 文件下操作 DTS，定义 led 设备节点如下：

```
gpioctr_device {
        compatible = "myir,gpioctr";
        #pinctrl-names = "default";
        #pinctrl-0 = <&pinctrl_gpio_blue>;

        status = "okay";
        gpioctr-gpios = <&gpio5 11 0>;
};
```

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）uboot 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 GPIO5_IO11，可使用下列命令。

```
u-boot=> gpio set 139
gpio: pin 139 (gpio 139) value is 1
u-boot=> gpio clear 139
gpio: pin 139 (gpio 139) value is 0
u-boot=>
```

可以用万用表量到 J25 上 19 脚电压拉高/拉低。

2）设备树控制

用户可以在 uboot 阶段使用设备树定义 IO 操作的节点，在代码中实现 IO 的功能，如下列 phy 的电源复位控制。

- **在 uboot 使用复位引脚**

在设备树配置好后需要在 uboot 使用该引脚,如下红色引脚设置：

```
#ifdef CONFIG_FEC_MXC
#define FEC_RST_PAD IMX_GPIO_NR(4, 2)
static iomux_v3_cfg_t const fec1_rst_pads[] = {
	MX8MP_PAD_SAI1_RXD0__GPIO4_IO02 | MUX_PAD_CTRL(NO_PAD_CTRL),
};
static void setup_iomux_fec(void)
{
	imx_iomux_v3_setup_multiple_pads(fec1_rst_pads,
					 ARRAY_SIZE(fec1_rst_pads));

	gpio_request(FEC_RST_PAD, "fec1_rst");
```

```
        gpio_direction_output(FEC_RST_PAD, 0);
        mdelay(15);
        gpio_direction_output(FEC_RST_PAD, 1);
        mdelay(100);
}.....
#endif
```

6.4.2. 内核驱动中使用 GPIO 管脚

1）独立 IO 驱动的使用

在 6.3.1 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 GOIO5_IO11 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 *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 myd_jx8mp_defconfig。

修改 drivers/char/Kconfig 中添加如下代码：

```
config SAMPLE_GPIO
      tristate "this is a gpio test driver"
      depends on CONFIG_GPIOLIB
```

修改 drivers/char/Makefile：

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

修改 myd_jx8mp_defconfig：

```
CONFIG_SAMPLE_GPIO=m
```

最后按照 5.4 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，同时编写独立 Makefile 程序：

```
# 修改 KERN_DIR
# #KERN_DIR = # 板子所用内核源码的目录
KERN_DIR = /home/duxy/myd-jx8mp-yocto/myir-imx-linux
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：

```
duxy@myir_server:~/myd-jx8mp-yocto $ source ~/opt_5_10/environment-setup-cortexa53-crypto-poky-linux
```

执行 make 命令，即可生成 gpioctr.ko 驱动模块文件：

```
duxy@myir_server:~/myd-jx8mp-yocto/gpio $ make
make -C /home/duxy/myd-jx8mp-yocto/myir-imx-linux M=`pwd` modules
make[1]: Entering directory '/home/ duxy/myd-jx8mp-yocto/myir-imx-linux'
  CC [M]  /home/duxy/myd-jx8mp-yocto/gpio/gpioctr.o
```

```
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/duxy/myd-jx8mp-yocto/gpio/gpioctr.mod.o
  LD [M]  /home/duxy/myd-jx8mp-yocto/gpio/gpioctr.ko
make[1]: Leaving directory '/home/duxy/myd-jx8mp-yocto/myir-imx-linux'
```

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 管脚的控制两种基本方式。

- Shell 命令
- 系统调用

1）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实现的，本节不做详细的说明，可查看**《MYD-JX8MPQ Linux 软件评估指南》**第 3.1 节描述。

2）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 6.4.2 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);

        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$ duxy@myir-server1:~/myd-jx8mp-yocto/gpio$ source ~/opt_5_10/environment-setup-cortexa53-crypto-poky-linux
```

使用编译命令$CC 可生成可执行文件 gpiotest。

```
$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@myd-jx8mp:~#  gpiotest /dev/myir_gpioctr0 on
root@ myd-jx8mp:~#  gpiotest /dev/myir_gpioctr0 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 Bitbake 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，它记录了原始代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-JX8MPQ 开发板上实现按键控制）来讲述 Makefile 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。

```
 TARGET = $(notdir $(CURDIR))
objs := $(patsubst %c, %o, $(shell ls *.c))
$(TARGET)_test:$(objs)
        $(CC) -o $@ $^
%.o:%.c
        $(CC) -c -o $@ $<
clean:
        rm -f  $(TARGET)_test *.all *.o
```

部分参数说明：

- $(CURDIR)：表示 Makfile 当前目录全路径
- $(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/wujl/MYD-JX8MP/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 实现代码如下：

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/ioctl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<sys/select.h>
#include<sys/time.h>
#include<fcntl.h>
#include<errno.h>
#include<linux/input.h>

int main(int argc, char *argv[])
{
        int ledn;
        char* tmp;
        int keys_fd;
        struct input_event t;
```

```
        keys_fd = open(argv[1], O_RDONLY);
        if (keys_fd <= 0)
        {
      printf ("open %s device error!\n",argv[1]);
            return 0;
        }

        printf("Hit any key on board ......\n");

   while(1)
        {
      if (read(keys_fd, &t, sizeof(t)) == sizeof(t))
            {
         if (t.type == EV_KEY)
               if (t.value == 0 || t.value == 1)
               {
                  printf ("key %d %s\n", t.code,(t.value) ? "Pressed" : "Released");
                  if(t.code==KEY_ESC)
                     break;
               }
            }
   }

   close (keys_fd);
   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 SDK 环境变量到当前 shell:

```
PC$ source ~/opt_5_10/environment-setup-cortexa53-crypto-poky-linux
```

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 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，执行以下命令，然后按 USER 按键，可以看到蓝灯亮灭的情况:

```
root@myd-jx8mp:~# key_led_test /dev/input/event2 noblock
Hit any key on board ......
key 2 Pressed
key 2 Released
key 2 Pressed
key 2 Released
key 2 Pressed
```

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-JX8MPQ 使用 Qt 5.15 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。

如需获得安装与配置详情请从 QTCreator 官方网站获得更多开发指导 https://www.qt.io/product/development-tools。

7.3. 应用程序开机自启动

1）应用程序在 Yocto 的配置

要使用 Yocto 构建生产部署阶段的镜像文件并且包含我们编写的应用，就需要为我们编写的应用创建一个配方（Recipe），helloworld 不具备开机自启动的功能，它配方的编写参见 https://www.yoctoproject.org/docs/3.1.1/mega-manual/mega-manual.html#hello-world-example。

通常我们的应用还需要实现开机自启动，这些也可以在配方中实现。下面以一个稍微复杂一点的 FTP 服务应用为例说明如何使用 Yocto 构建包含特定应用的生产镜像，这里的 FTP 服务程序采用的是开源的 Proftpd，各个版本源码位于 ftp://ftp.proftpd.org/distrib/source/。

在我们从头开始写一个配方之前，我们可以在当前源码仓库中查找一下是否已经存在该应用，或者类似应用的配方，查找方法如下：

```
PC $ bitbake -s | grep proftpd
proftpd                                 :1.3.6-r0
```

注意：执行 bitbake 命令之前，确保您已经执行了构建 Yocto 项目的环境变量设置脚本,详情请参考第 3 章。

我们也可以在 OpenEmbedded 的官方网站层索引（http://layers.openembedded.org/layerindex/branch/master/layers/）中查找是否有同样或者类似应用的配方。

编写新配方的方法参见 Yocto 项目完全手册编写新的配方章节 https://www.yoctoproject.org/docs/3.1.1/mega-manual/mega-manual.html#new-recipe-writing-a-new-recipe。

本节重点描述如何移植 FTP 服务到目标机器中的方法。通过搜索当前源代码仓库发现 Yocto 项目中已经存在 proftpd 的配方，只是没有添加的系统镜像中。下面详细描述具体的移植过程。

- **查找 Yocto 的 proftpd 配方**

```
duxy@myir_server:~/myd-jx8mp-yocto/build-xwayland$ bitbake -s | grep proftpd
proftpd                                 :1.3.6-r0
```

注：这里可以看到 Yocto 项目中已经存在 proftpd 配方，版本为 1.3.6-r0。

- **单独编译 proftpd**

```
PC $ bitbake proftpd
```

- **打包 proftpd 到文件系统**

在 conf/local.conf 中增加一行语句：

```
IMAGE_INSTALL_append = "proftpd"
```

- **重新构建镜像**

```
PC $ bitbake core-image-minimal
```

- **烧录新镜像**

系统构建完成之后，需重新烧录镜像并查看 proftpd 服务是否运行：

```
# ps -axu | grep proftpd
nobody    584  0.0  0.3   3032  1344 ?        Ss   01:51   0:00 proftpd: (accepting con
nections)
root     1713  0.0  0.0   1776   336 pts/0    S+   01:59   0:00 grep proftpd
```

这里补充说明一下 FTP 的账户设置。FTP 客户端有三种类型登录账户，分别为匿名账户，普通账户和 root 账户。

- **匿名账户**

用户名为 ftp ，不需要设置密码，用户登录后可以查看系统*/var/lib/ftp* 目录下的内容，默认没有写权限。由于系统默认不存在*/var/lib/ftp* 目录，所以需要用户在目标机器上创建一个目录*/var/lib/ftp*。为了尽量不修改 meta-openembbed, 我们可以通过为 proftpd 配方添加 Append 文件“proftpd_1%.append”来实现/var/lib/ftp 目录的创建。

```
do_install_append() {
        install -m 755 -d ${D}/var/lib/${FTPUSER}
        chown ftp:ftp ${D}/var/lib/${FTPUSER}
}
```

编辑好的 proftpd_1%.append”需要放置到 meta-myir/meta-bsp 下面 recipes-daemons/proftpd 目录。然后重复上面添加应用的步骤，重新构建镜像文件进行测试。

- **普通账户**

在目标机器上使用 useradd 和 passwd 命令可以创建普通用户，并设置用户密码之后，客户端也可以使用该普通账户登录到该用户的 HOME 目录。如果需要在编译镜像时包含普通用户，可以参照 https://www.yoctoproject.org/docs/3.1.1/mega-manual/mega-manual.html#ref-classes-useradd 添加普通用户，然后重新构建镜像文件，具体方法这里不再赘述。

- **root 账户**

如果需要开放 root 账户登录 FTP 服务器，需要先修改/etc/proftpd.conf 文件，在文件中增加一行配置 "RootLogin on"。与此同时，也需要为 root 账户设置密码，重启 proftpd 服务之后，客户端也可以使用 root 账户登录到目标机器上。

```
# systemctl restart proftpd
```

注意：修改/etc/proftpd.conf 使能 root 账户登录仅用于测试目的，关于/etc/proftpd.conf 的更多配置，参见 http://www.proftpd.org/docs/example-conf.html。

2）开机自启动应用

本节还是以 proftpd 配方为例从配方源码的层面介绍如何添加应用程序配方并实现程序的开机自启动。proftpd 配方位于源代码仓库 sources/meta-openembedded/meta-networking/recipes-daemons/proftpd 目录，目录结构如下。

```
PC$ tree
.
├── files
│   ├── basic.conf.patch
│   ├── build_fixup.patch
│   ├── close-RequireValidShell-check.patch
│   ├── contrib.patch
│   ├── default
│   ├── proftpd-basic.init
│   └── proftpd.service
└── proftpd_1.3.6.bb

1 directory, 8 files
```

- proftpd_1.3.6.bb 为构建 proftpd 服务的配方
- proftpd.service 为开机自启动服务

➢ proftpd-basic.init 为 proftpd 的启动脚本

proftpd_1.3.6.bb 配方中指定了获取 proftpd 服务程序的源代码路径以及针对该版本源码的一些补丁文件：

```
SRC_URI = "ftp://ftp.proftpd.org/distrib/source/${BPN}-${PV}.tar.gz \
           file://basic.conf.patch \
           file://proftpd-basic.init \
           file://default \
           file://close-RequireValidShell-check.patch \
           file://contrib.patch  \
           file://build_fixup.patch \
           file://proftpd.service \
"
```

配方中还指定了 proftpd 的配置（do_configure)和安装过程（do_install）：

```
FTPUSER = "ftp"
FTPGROUP = "ftp"

do_install () {
    oe_runmake DESTDIR=${D} install
    rmdir ${D}${libdir}/proftpd ${D}${datadir}/locale
    [ -d ${D}${libexecdir} ] && rmdir ${D}${libexecdir}
    sed -i '/ *User[ \t]*/s/ftp/${FTPUSER}/' ${D}${sysconfdir}/proftpd.conf
    sed -i '/ *Group[ \t]*/s/ftp/${FTPGROUP}/' ${D}${sysconfdir}/proftpd.conf
    install -d ${D}${sysconfdir}/init.d
    install -m 0755 ${WORKDIR}/proftpd-basic.init ${D}${sysconfdir}/init.d/proftpd
    sed -i 's!/usr/sbin/!${sbindir}/!g' ${D}${sysconfdir}/init.d/proftpd
    sed -i 's!/etc/!${sysconfdir}/!g' ${D}${sysconfdir}/init.d/proftpd
    sed -i 's!/var/!${localstatedir}/!g' ${D}${sysconfdir}/init.d/proftpd
    sed -i 's!^PATH=.*!PATH=${base_sbindir}:${base_bindir}:${sbindir}:${bindir}!'
${D}${sysconfdir}/init.d/proftpd

    install -d ${D}${sysconfdir}/default
    install -m 0755 ${WORKDIR}/default ${D}${sysconfdir}/default/proftpd
```

```
    # create the pub directory
    mkdir -p ${D}/home/${FTPUSER}/pub/
    chown -R ${FTPUSER}:${FTPGROUP} ${D}/home/${FTPUSER}/pub
    if ${@bb.utils.contains('DISTRO_FEATURES', 'pam', 'true', 'false', d)}; then
        # install proftpd pam configuration
        install -d ${D}${sysconfdir}/pam.d
        install -m 644 ${S}/contrib/dist/rpm/ftp.pamd ${D}${sysconfdir}/pam.d/proft
pd
        sed -i '/ftpusers/d' ${D}${sysconfdir}/pam.d/proftpd
        # specify the user Authentication config
        sed -i '/^MaxInstances/a\AuthPAM                    on\nAuthPAMConfig
          proftpd' \
            ${D}${sysconfdir}/proftpd.conf
    fi

    install -d ${D}/${systemd_unitdir}/system
    install -m 644 ${WORKDIR}/proftpd.service ${D}/${systemd_unitdir}/system
    sed -e 's,@BASE_SBINDIR@,${base_sbindir},g' \
        -e 's,@SYSCONFDIR@,${sysconfdir},g' \
        -e 's,@SBINDIR@,${sbindir},g' \
        -i ${D}${systemd_unitdir}/system/*.service

    sed -e 's|--sysroot=${STAGING_DIR_HOST}||g' \
        -e 's|${STAGING_DIR_NATIVE}||g' \
        -e 's|-fdebug-prefix-map=[^ ]*||g' \
        -e 's|-fmacro-prefix-map=[^ ]*||g' \
        -i ${D}/${bindir}/prxs

    # ftpmail perl script, which reads the proftpd log file and sends
    # automatic email notifications once an upload finishs,
    # depends on an old perl Mail::Sendmail
    # The Mail::Sendmail has not been maintained for almost 10 years
```

```
    # Other distribution not ship with ftpmail, so do the same to
    # avoid confusion about having it fails to run
    rm -rf ${D}${bindir}/ftpmail
    rm -rf ${D}${mandir}/man1/ftpmail.1
}
```

这两个函数对应 BitBake 构建过程的 config 和 install 任务(关于任务的更多信息，参见 https://www.yoctoproject.org/docs/3.1.1/mega-manual/mega-manual.html#ref-tasks)。

proftpd_1.3.6.bb 配方通过继承 systemd.class（具体内容查看 layers/openembedded-core/meta/classes/systemd.bbclass）默认使能了 SYSTEMD_AUTO_ENABLE 变量并实现开机自启动，用户自己编写的配方也可以通过设置变量 SYSTEMD_AUTO_ENABLE 实现开机自启动，示例如下：

```
SYSTEMD_AUTO_ENABLE_${PN} = "enable"
```

当前目标机器采用 systemd 作为初始化管理子系统，systemd 是一个 Linux 系统基础组件的集合，提供了一个系统和服务管理器，运行为 PID 1 并负责启动其它程序。Yocto 项目下使用 systemd 的配置参见 https://www.yoctoproject.org/docs/3.1.1/mega-manual/mega-manual.html#selecting-an-initialization-manager。

Proftpd 服务的开机自启动服务文件 proftpd.service 内容如下：

```
[Unit]
Description=proftpd Daemon
After=network.target

[Service]
Type=forking
ExecStart=@SBINDIR@/proftpd -c @SYSCONFDIR@/proftpd.conf
StandardError=syslog

[Install]
WantedBy=default.target
```

- After 表示此服务在 network 启动后再启动。
- Type 表示启动的方式为 forking。
- ExecStart 表示需要启动的程序，及对应的参数。

如需了解更多关于 systemd 的信息请查看此网站 https://wiki.archlinux.org/index.php/systemd。

用户在添加自己编写的应用时，也可以参照上面的示例创建配方，设置开机自启动，并打包进系统镜像。自己编写的配方建议放置到 *sources/meta-myir/meta-bsp* 目录。

8. 参考资料

- **Linux kernel 开源社区**
  https://www.kernel.org/
- **Yoto 项目 BSP 开发指南**
  https://www.yoctoproject.org/docs/3.1.1/bsp-guide/bsp-guide.html
- **Yocto 项目 Linux 内核开发手册**
  https://www.yoctoproject.org/docs/3.1.1/kernel-dev/kernel-dev.html
- **Yocto 开发指导**
  https://www.yoctoproject.org/
- **NXP 开发社区**
  https://community.nxp.com/

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

武汉分公司

负责区域：武汉

电话：027-59621648

传真：Na

邮编：430000

地址：湖北省武汉市洪山区关南园一路 20 号当代科技园 7 号楼 1903 号

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：027-59621648

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

凡是通过米尔科技直接购买或经米尔科技授权的正规代理商处购买的米尔科技全系列产品，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。