MYS-8MMX Linux System Development Guide

File Status： [ ] Draft [√] Release	**FILE ID：**	MYIR-MYS-8MMX-SW-DG-EN-L5.4.3
	VERSION：	V2.0[DOC]
	AUTHOR：	Coin、Miller Wu
	CREATED：	2021-03-15
	UPDATED：	2022-05-09

Revision History

VERSION	AUTHOR	PARTICIPANT	DATE	DESCRIPTION
V1.0[DOC]	Coin		20210315	Initial Version:u-boot 2019，kernel 5.4.3，Yocto zeus
V2.0[DOC]	Coin		20220509	PHY YT8531S instead of 8035; TYPEC PTN5150 instead of STUSB1600,Add SPDIF output with HDMI; delete PCIE clock IC

CONTENT

1. Overview

There are many open source system build frameworks on the Linux system platform,these frameworks make it easy for developers to build and customize embedded systems,at present common similar software has Buildroot, Yocto, OpenEmbedded and so on. The Yocto project uses a more powerful and customized approach to build Linux systems that suitable for embedded products. Yocto is not only a file system manufacturing tool, but also provides a complete set of Linux-based development and maintenance workflow, so that the embedded developers of the Underlying Software and the High-Level Application can develop under a unified framework, which solves the fragmented and unmanaged development mode in the traditional development mode.

This document mainly introduces the complete process of customizing a complete embedded Linux system based on Yocto project, including the configuration of development environment, how to get the source code, how to port bootloader and kernel, and how to customize rootfs suitable for their own application requirements. First of all, we will introduce how to build a system image for MYS-8MMX development board based on the source code provided by us, and how to burn the prebuilt image to the development board. Then, we focus on the methods and key points of porting the system to the user's hardware platform. In addition, if you are developing a project based on MYS-8MMX CPU module ,we will also take some actual BSP porting cases and rootfs customization cases as examples to guide users to quickly customize the system image suitable for their own base-board hardware.

This document does not include the introduction of Yocto project and the basic knowledge of Linux system, and the user guide is suitable for embedded Linux development engineers with some development experience. For some specific functions that users may use in the process of secondary development, we also provide detailed application notes for reference,please refer to Table 2-3 of "MYS-8MMXSDK2.0.0 Release Notes" for the detailed list of documents.

1.1. Software Resources

MYS-8MMX series development board runs an operating system based on the Linux 5.4.3 kernel,which also provided a wealth of system resources and other software resources. The resource packages provided with the development board are as follows:

- ➢ A software development kit (SDK) for cross-development on an host PC.
- ➢ Distribution in source code:ATF,U-boot,Linux Kernel and drivers of each module .
- ➢ Various debugging tools for Windows desktop and Linux desktop environments.
- ➢ Various peripherals application development samples, etc.

For specific software information, ,please refer to Table 2-3 of "MYS-8MMX SDK2.0.0 Release Notes" for the detailed list of documents.

1.2. Document Resources

According to the different stages of using the development board, the SDK contains different types of documents and manuals, such as release notes, introduction guide, evaluation guide, development guide, application notes, frequently asked questions and answers, etc.For detailed document list, please refer to table 2-3 of "MYS-8MMX SDK2.0.0 Release Notes" .

2. Development Environment

This chapter mainly introduces some software and hardware environment required in the development process, including the necessary development host environment, necessary software tools, code and resource acquisition, etc. the specific preparatory work will be described in detail below.

2.1. Developing Host Environment

This section describes how to deploy the MYS-8MMX development environment.By reading this section, you will learn about the installation and use of hardware and software tools.And you can quickly deploy the relevant development environment and prepare for subsequent development and debugging.The MYS-8MMX series processor is a multi-core heterogeneous processor. The specific instructions are as follows:

- Four Arm Cortex-A53 64-bit RISC core operating at up to 1.8 GHz,which can run embedded Linux systems and use common development tools of embedded Linux system.
- Embed a Cortex-M4 32-bit RISC core operating at up to 400 MHz frequency ,which can run bare-metal code or other real-time operating systems(RTOS) and use cortex M4 software development tools provided by NXP official.

1) Hardware and Software Requirements

- **Host Hardware**

To get the Yocto Project expected behavior in a Linux Host Machine, the packages and utilities described below must be installed. An important consideration is the hard disk space required in the host machine. It is recommended that at least 200 GB is provided, which is enough to compile all backends together. In addition, the processor with more than dual core CPU, 8GB memory or higher configuration will better meet the operation requirements. It can be the host with Linux system installed etc.

- **Host Operating System**

There are many options for the host operating system used to build the yocto project. Please refer to the official Yocto instructions for details: https://www.yoctoproject.org/docs/current/mega-manual/mega-manual.html#dev-preparing-the-build-host . Generally, we choose to build it on the local host with Fedora, openSUSE, Debian, Ubuntu, RHEL or Cent OS Linux distributions. Here, we recommend the Ubuntu 16.04 64bit desktop system, the subsequent development is also based on this system.

2) Prerequisite Package Installation

After the installation of the Ubuntu 16.04 64bit desktop system, the appropriate configuration can be carried out to prepare for the subsequent development.

- **Set the root Password**

```
myir@myir-server1:$  sudo passwd root
```

- **Installation of SSH**

After installing the SSH service, you can connect to Ubuntu in the window environment by using the SecureCRT tool SSH2 for subsequent development.

```
myir@myir-server1:$  sudo apt-get install openssh-server
```

Generate key for user

```
myir@myir-server1:$  su user
ssh-keygen -t rsa
```

- **Configuration of samba**

Samba service can directly let users access the content of Ubuntu in the form of file under window, which is more convenient to read and write. To install Samba:

```
myir@myir-server1:$  apt-get install samba
```

Add user configuration to the */etc/samba/smb.conf* configuration file. For example, if the Linux user name is "myir", the configuration is as follows:

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

Create an account and set a password:

```
myir@myir-server1:$   sudo smbpasswd -a myir
New SMB password:
Retype new SMB password:
Added user myir.
```

Restart Samba service:

```
myir@myir-server1:$   /etc/init.d/smbd restart
[ ok ] Restarting smbd (via systemctl): smbd.service.
```

- **Git configuration**

```
myir@myir-server1:$  git config --global user.name "user"
myir@myir-server1:$  git config --global user.email "email"
myir@myir-server1:$  git config --list
```

- **Installation of vim**

The system's own VIM tool cannot be de qualified and needs to be re installed：

```
myir@myir-server1:$  sudo apt-get remove vim-common
myir@myir-server1:$  sudo apt-get install vim
```

- **Installation of necessary tools**

```
myir@myir-server1:$  sudo apt-get install gawk wget git-core diffstat unzip texinf
o gcc-multilib build-essential chrpath socat libsdl1.2-dev libsdl1.2-dev xterm sed c
vs subversion coreutils texi2html docbook-utils python-pysqlite2 help2man make
```

```
gcc g++ desktop-file-utils libgl1-mesa-dev libglu1-mesa-dev mercurial autoconf
automake groff curl lzop asciidoc u-boot-tools python3-pip
```

3) Create Working Directory

You need to create a working directory to set up a unified environment variable path. Then copy the source code from the product CD to the working directory to facilitate the path access of the following steps. The user and owner of the copied code must be a personal account and cannot be operated with the root account.eg:

```
myir@myir-server1:$  mkdir -p ~/MYS-8MMX
myir@myir-server1:$  cp -r <DVDROM>/02-Images  ~/MYS-8MMX
myir@myir-server1:$  cp -r <DVDROM>/03-Tools    ~/MYS-8MMX
myir@myir-server1:$  cp -r <DVDROM>/04-Sources ~/MYS-8MMX
```

2.2. Introduction of Software Development Tools

The Pins Tool for i.MX Application Processors is the successor of Processor Expert? Software for i.MX Processors. The new Pins Tool makes pin configuration easier and faster with an intuitive and easy user interface, which then generates normal C code that can then be used in any C and C++ application. The Pins Tool configures pin signals from multiplexing (muxing) to the electrical properties of pins, and it also creates Device Tree Snippets Include (.dtsi) files and reports in CSV format.

The Pins Tool for i.MX Application Processors is installed as a desktop tool which then loads additional device information through a network connection, but does otherwise not need internet connection. It does not require a project setup, as all the settings are stored in text and generated source files, which then can be easily stored in a version control system or exchanged with other users.

The Pins Tool for i.MX Application Processors makes it easy and fast to configure the hardware and pins of the i.MX processors, without the need for extensive hardware knowledge.Software download address:

https://www.nxp.com/design/designs/pins-tool-for-i-mx-application-processors:PINS-TOOL-IMX

2.3. Install the SDK Customized by MYIR

After using Yocto to build the system image, we can also use Yocto to build a set of extensible SDK.The CD image provided by MYIR contains a compiled SDK package, which is located in the *03_tools/Toolchains/.* It can be used to compile bootloader, kernel or their own applications. The specific function will be described in detail in the following chapters.

Table 2-1.Function description of compiler tool chain

Toolchain File Name	Description
fsl-imx-xwayland-glibc-x86_64-myir-image-full-aarch64-myd-imx8mm-toolchain-5.4-zeus.sh	The SDK not only contains an independent cross development tool chain, but also provides qmake, sysroot of the target platform, libraries and header files that QT application development depends on,etc.Users can directly use this SDK to build an independent development environment
fsl-imx-xwayland-glibc-x86_64-meta-toolchain-aarch64-myd-imx8mm-toolchain-5.4-zeus.sh	Basic tool chain, compile bootloader, kernel separately or compile your own application

Here we will first introduce the installation steps of the SDK,the steps are as follows:

1) Run the SDK Installation Script

Run the shell script with the normal user permissions. The installation path will be raised in the operation. By default, it is in the */opt* directory. This routine installs the meta basic tool chain in *//home/myir/MYS_8MMX_SDK/opt* directory,as shown below:

```
myir@myir-server1:$   cd ~/MYS-8MMX/03-Tools/Toolchain/
myir@myir-server1:$   ./fsl-imx-xwayland-glibc-x86_64-meta-toolchain-aarch64-
myd-imx8mm-toolchain-5.4-zeus.sh
NXP i.MX Release Distro SDK installer version 5.4-zeus
```

```
=================================================================
Enter target directory for SDK (default: /opt/fsl-imx-xwayland/5.4-zeus): /home/m
yir/MYS_8MMX_SDK/opt
You are about to install the SDK to "/home/myir/MYS_8MMX_SDK/opt". Proceed
[Y/n]? y
Extracting SD
K...........................................................................................................................
............................................................................................................................
............................................................................................................................
............................................................................................................................
.....................................................................................................................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/myir/MYS_8MMX_SDK/opt/environment-setup-aarch64-poky-linux
```

2) Test SDK

Initialize cross-compilation via SDK and ensure that the environment is correctly setup:

```
myir@myir-server1:$  cd  ~/MYS_8MMX_SDK
myir@myir-server1:$  source /home/myir/MYS-8MMX_SDK/opt/environment-setu
p-aarch64-poky-linux
myir@myir-server1:$  $CC --version
aarch64-poky-linux-gcc (GCC) 9.2.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPO
SE.
myir@myir-server1:$ ~/MYS-8MMX_SDK$ echo $CC
aarch64-poky-linux-gcc -mcpu=cortex-a53+crc --sysroot=/home/myir/MYS-8MM
X_SDK/opt/sysroots/aarch64-poky-linux
```

In the same way, please install the tool chain for Qt development. When installing two file chains, please specify different directory. Do not make the same directory, otherwise files will overlap each other.

3. Build the File System with Yocto

3.1. Introduction

The Yocto Project is an open-source "umbrella" project, meaning it has many sub-projects under it. Yocto just puts all the projects together and provides a reference build project Poky to guide developers on how to apply these projects to build an embedded Linux system,Yocto also Contains Bitbake tool, OpenEmbedded-Core, board level support packages, configuration files of various software,so you can build systems with different requirements.

MYS-8MMX provides yocto compliant configuration files to help developers build linux system images that can be burned on MYS-8MMX board. Yocto also provides rich development document resources for developers to learn and customize their own systems. Due to the limited space,for more information about the Yocto project, please refer to the site: www.yoctoproject.org .

This section is suitable for developers who need to customize the file system in depth, and hope to build a file system from yocto that meets the MYS-8MMX series development board,. If developers use it for the first time without special needs, they can directly use the file system provided by MYS-8MMX.

Note: building yocto does not need to initialize the SDK toolchain environment variables in Section 2.3. Please create a new shell or open a new terminal window.

3.2. Get the Source Code

We provide two ways to obtain the source code. One is to obtain the compressed package directly from the *04-sources* directory of the MYIR CD image, and the other is to use repo to obtain the source code updated in real time on GitHub for construction. Users can choose one of them.

3.2.1 Get Source Code Package from CD Image

You can find the Yocto compressed source package in the development kit package 04-Sources/mys-8mmx-yocto.tar.gz. Copy the compressed package to the user specified directory, such as the $HOME/MYS-8MMX directory. This directory will be used as the top-level directory for subsequent construction. After decompressing, the source directory will appear:

```
PC$:mkdir -p mys-8mmx-yocto
PC$:tar -zxf mys-8mmx-yocto.tar.bz2 -C mys-8mmx-yocto
```

3.2.2 Get Source Code from GitHub

Table 3-1. Source list

	Github	Branch Name
Manifest	https://github.com/MYiR-Dev/myir-imx-manifest	i.MX8M-5.4-zeus
meta-myir-imx	https://github.com/MYiR-Dev/meta-myir-imx	i.MX8M-5.4-zeus
U-Boot	https://github.com/MYiR-Dev/myir-imx-uboot	develop
Kernel	https://github.com/MYiR-Dev/myir-imx-linux	develop

At present, the BSP source code and Yocto source code of MYS-8MMX development board are managed by GitHub and will be updated for a long time. Please refer to Section 2.2 of “MYS-8MMX SDK2.0.0 Release Notes”. Users can use repo to get and synchronize the code on GitHub. The specific operation methods are as follows:

Put the 03-tools/repo file into the $HOME/bin directory, add the executable permission, and add it to the path variable:

```
PC$:   mkdir -p mys-8mmx-yocto
PC$:   cd mys-8mmx-yocto
PC$:   mkdir -p ~/bin
PC$:   cp repo ~/bin/
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-5.4.3-2.0.0.xml -b i.MX8M-5.4-zeus
repo sync
```

After the synchronization code is completed, you will get a folder under the mys-8mmx-yocto directory, which contains the source code or source repository path related to MYS-8MMX development board. The directory structure is the same as that extracted from the compressed package.

3.3. Build Development Board Image

This section provides the detailed information along with the process for building an image. Before using yocto project to build the system, we need to set the corresponding environment variables.MYIR provides a script(myir-setup-release.sh) to simplify the setup for MYIR machines. To use the script, the name of the specific machine to be built for needs to be specified as well as the desired graphical backend. The script sets up a directory and the configuration files for the specified machine and backend(eg:build-xwayland).

The Yocto Project build uses the bitbake command. For example, bitbake <component>.Each component build has multiple tasks, such as fetching, configuration, compilation, packaging, and deploying to the target rootfs. The bitbake image build gathers all the components required by the image and build in order of the dependency per task. The first build is the toolchain along with the tools required for the components to build.After the build is complete, this directory will contain all the output files.

1) View the Contents of Yocto Source Code Package

List the MYS-8MMX-Yocto directory as follows:

```
PC$:tree -a -L 1 mys-8mm
mys-8mmx-yocto
├── myir-setup-release.sh -> sources/meta-myir/tools/myir-setup-release.sh
├── README -> sources/base/README
├── README-IMXBSP -> sources/meta-myir/README
├── setup-environment -> sources/base/setup-environment
└── sources
```

Generally, it is also recommended that customers check the README-IMXBSP content first, and there are detailed compilation methods.

Notice: Do not use root to decompress and compile, do not compile under NTFS format disk

2) Choose to compile V1 or V2 version image

sources/meta-myir/meta-bsp/conf/machine/mys-8mmx.conf decide whether to compile V1 or V2.

```
UBOOT_CONFIG ??= "sd"
#UBOOT_CONFIG[sd] = "mys_iot_imx8mm_ddr4_evk_defconfig,sdcard"
UBOOT_CONFIG[sd] = "mys_iot_imx8mm_v20_ddr4_evk_defconfig,sdcard"
```

Compile the V2 version image by default.

3) Execute Script to Set Environment Variables

The syntax for the envsetup.sh script is shown below:

```
 DISTRO=fsl-imx-xwayland MACHINE=mys-8mmx source sources/meta-myir/tool
s/myir-setup-release.sh -b build-xwayland
```

After the script runs, the working directory is the one just created by the script,eg:build-xwayland, and automatically jump to this directory.In this directory,bitbake <component> builds the named component.

4) Building myir-image-full image

When compiling, there will be a fetch process to download the third-party source code. If the network is unstable, the fetch error often occurs. Therefore, it is recommended to download the downloads file on the disk and place it in the same level directory, such as mys-8mmx-yocto directory. Here is a demonstration of the compilation process.

- **Copy downloads file to the Specified Directory**

Download the downloads file from the web disk and copy it to the corresponding directory (e.g:mys-8mmx-yocto), as shown in the red file:

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

The downloads file can also be pulled directly from the Internet, but the pull speed depends on the network speed of the user, so it is generally recommended to use the downloads file from the network disk. The general commands for pulling files on the Internet are as follows:

```
myir@myir-server1:  bitbake myir-image-full --runall=fetch
```

- **Build a full image**

After copying the download file, execute the following command to build the system:

```
duxy@myir-server1:/media/duxy/wujl/mys-8mm/build-xwayland$ bitbake myir-i
mage-full
NOTE: Your conf/bblayers.conf has been automatically updated.
Parsing recipes: 100% |###########################################| Ti
me: 0:00:41
Parsing of 2905 .bb files complete (0 cached, 2905 parsed). 4109 targets, 244 skip
ped, 2 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION           = "1.44.0"
BUILD_SYS            = "x86_64-linux"
NATIVELSBSTRING      = "ubuntu-16.04"
TARGET_SYS           = "aarch64-poky-linux"
MACHINE              = "mys-8mmx"
DISTRO               = "fsl-imx-xwayland"
DISTRO_VERSION       = "5.4-zeus"
TUNE_FEATURES        = "aarch64 cortexa53 crc crypto"
TARGET_FPU           = ""
meta
meta-poky            = "HEAD:a8f6e31bebc5a551fab1fec8d67489af80878f71"
meta-oe
meta-multimedia
meta-python          = "HEAD:bb65c27a772723dfe2c15b5e1b27bcc1a1ed884c"
```

```
meta-freescale       = "HEAD:94f4f086c6014cbcfd10bda3540d19558c8bf0b0"
meta-freescale-3rdparty = "HEAD:aea3771baa77e74762358ceb673d407e36637e5
f"
meta-freescale-distro = "HEAD:ca27d12e4964d1336e662bcc60184bbff526c857"
meta-bsp
meta-sdk
meta-ml             = "HEAD:a8806096d3f9cc81ad8918e4668d130faffd2614"
meta-browser        = "HEAD:5f365ef0f842ba4651efe88787cf9c63bc8b6cb3"
meta-rust           = "HEAD:d8d77be1292064a02adcb5e72e293604b704f69b"
meta-gnome
meta-networking
meta-filesystems     = "HEAD:bb65c27a772723dfe2c15b5e1b27bcc1a1ed884c"
meta-qt5            = "HEAD:a582fd4c810529e9af0c81700407b1955d1391d2"

Initialising tasks: 100% |############################################| Time:
 0:00:13
Sstate summary: Wanted 3450 Found 0 Missed 3450 Current 0 (0% match, 0% co
mplete)
NOTE: Executing Tasks
NOTE: Setscene tasks completed
Currently 11 running tasks (5560 of 9200)  60% |###################           |
0: qtbase-5.13.2+gitAUTOINC+a7a24784ee-r0 do_configure - 4m15s (pid 21529)
```

5) Fix Error in Build Process

If the network is unstable during the build process, the following error messages may appear, as follows:

```
ERROR: Task (/home/myir/MYS-8MMX/MYS-8MMX-Yocto/sources/meta-myir/me
ta-ml/recipes-libraries/tensorflow-lite/tensorflow-lite_2.1.0.bb:do_configure) failed
 with exit code '1'
```

The above error may be caused by poor network, which may cause some components of this package not to be downloaded completely. Please solve it in the following way.

- **Check the compilation directory of the error package**

Execute the following command to view the compilation directory of the error package:

```
myir@myir-server1： bitbake -e tensorflow-lite | grep ^S=
S="/media/duxy/wujl/mys-8mm/build-xwayland/tmp/work/aarch64-poky-linux/te
nsorflow-lite/2.1.0-r0/git"
```

- **Block Dependent Downloads**

Enter the download directory of the compilation directory, block the dependent download, and modify the following red mark:

```
myir@myir-server1:$  cd /media/duxy/wujl/mys-8mm/build-xwayland/tmp/work/
aarch64-poky-linux/tensorflow-lite/2.1.0-r0/git/tensorflow/lite/tools/make/
myir@myir-server1:$  vi download_dependencies.sh
#download_and_extract "${EIGEN_URL}" "${DOWNLOADS_DIR}/eigen"
#download_and_extract "${GEMMLOWP_URL}" "${DOWNLOADS_DIR}/gemmlowp
"
#download_and_extract "${GOOGLETEST_URL}" "${DOWNLOADS_DIR}/googletest
"
#download_and_extract "${ABSL_URL}" "${DOWNLOADS_DIR}/absl"
#download_and_extract "${NEON_2_SSE_URL}" "${DOWNLOADS_DIR}/neon_2_sse
"
#download_and_extract "${FARMHASH_URL}" "${DOWNLOADS_DIR}/farmhash"
#download_and_extract "${FLATBUFFERS_URL}" "${DOWNLOADS_DIR}/flatbuffers"
#download_and_extract "${FFT2D_URL}" "${DOWNLOADS_DIR}/fft2d"
```

- **Delete Previous Download Directory**

After modifying the download dependency, users need to delete packages that have not been downloaded completely before:

```
myir@myir-server1:  rm downloads -rf
```

- **Copy Download File**

After deleting the download, the user needs to copy the downloaded package to the directory, and then unzip it. The file is in the directory *04-sources/tensorflow_lite-downloads.tar.gz*:

```
myir@myir-server1:  cp 04-Source/tensorflow_lite-downloads.tar.gz  .
myir@myir-server1:  tar -zxf  tensorflow_lite-downloads.tar.gz
```

After modification, rebuild the system. And then you can find the complete image of the compilation in the "*build-xwayland/tmp/deploy/images/mys-8mmx*" directory.

Table 3-1.Image file list

Image File	Description
myir-image-full-mys-8mmx.sdcard.bz2	Full image
myir-image-full-mys-8mmx.tar.bz2	File system
myir-image-full-mys-8mmx.ext4	Ext4 file system
Image	Kernel image
*.dtb	Device Tree
imx-boot	Uboot image

The generated image file can be updated according to Section 4.1.

3.4. Build the Image Burned to EMMC with SD Card

In order to meet the needs of burning image to EMMC, MYIR provides a burning method suitable for mass production.Users can copy the specified system image into the specified partition of EMMC on board through the system in SD card.The entire burner image resource has been integrated into the yocto project, so users only need to use the yocto project to build the burner image.

1) Execute script to set environment variables

The syntax for the envsetup.sh script is shown below:

```
 PC $:DISTRO=fsl-imx-xwayland MACHINE=mys-8mmx source sources/meta-myir
/tools/myir-setup-release.sh -b build-xwayland
```

2) Building myir-image-full image

Since the SD card burner image is to copy the whole system (myir-image-full) to the specified partition directory of eMMC, it is necessary to build the myir-image-full image before building the SD card burner image.

3) Build SD Card Burner Image

The following commands is an example on how to build the burner image:

```
myir@myir-server1:  $ bitbake -c cleansstate fac-burn-emmc-full
myir@myir-server1:  $ bitbake -c cleansstate myir-image-burn
myir@myir-server1:  $ bitbake fac-burn-emmc-full
myir@myir-server1:  $ bitbake myir-image-burn
```

After the system is compiled successfully, the system image files of various formats are generated under the directory "*tmp/deploy/images/mys-8mmx/myir-image-burn-mys-8mmx.sdcard.bz2*".Then update according to section 4.2.

3.5. Build SDK (optional)

MYIR has provided a relatively complete SDK installation package, which can be directly used by users. However, when users need to introduce new libraries into the SDK, they need to reuse yocto to build new SDK tools.

This section simply describes how to build the SDK .The following command is an example on how to build the SDK package with QT:

```
myir@myir-server1:$ bitbake -c populate_sdk meta-toolchain-qt5
```

Or use the following command to generate a toolchain without QT:

```
myir@myir-server1:$ bitbake -c populate_sdk meta-toolchain
```

After building, the SDK installation package will be generated in the path of "*build-xwayland/tmp/deploy/sdk*". Please refer to Section 2.3 for the installation method.

```
myir@myir-server1:$  ls build-xwayland/tmp/deploy/sdk
-rw-r--r-- 2 myir myir       9149 12 月 21 18:04 fsl-imx-xwayland-glibc-x86_64-met
a-toolchain-aarch64-myd-imx8mm-toolchain-5.4-zeus.host.manifest
-rwxr-xr-x 2 myir myir  106846848 12 月 21 18:06 fsl-imx-xwayland-glibc-x86_64-
meta-toolchain-aarch64-myd-imx8mm-toolchain-5.4-zeus.sh
-rw-r--r-- 2 myir myir       1607 12 月 21 18:04 fsl-imx-xwayland-glibc-x86_64-met
a-toolchain-aarch64-myd-imx8mm-toolchain-5.4-zeus.target.manifest
-rw-r--r-- 2 myir myir     450652 12 月 21 18:04 fsl-imx-xwayland-glibc-x86_64-me
ta-toolchain-aarch64-myd-imx8mm-toolchain-5.4-zeus.testdata.json
-rw-r--r-- 2 myir myir      49819 12 月 21 19:18 fsl-imx-xwayland-glibc-x86_64-myir
-image-full-aarch64-myd-imx8mm-toolchain-5.4-zeus.host.manifest
-rwxr-xr-x 2 myir myir 3127407720 12 月 21 19:34 fsl-imx-xwayland-glibc-x86_64-
myir-image-full-aarch64-myd-imx8mm-toolchain-5.4-zeus.sh
-rw-r--r-- 2 myir myir     152135 12 月 21 19:17 fsl-imx-xwayland-glibc-x86_64-myi
r-image-full-aarch64-myd-imx8mm-toolchain-5.4-zeus.target.manifest
```

```
-rw-r--r-- 2 myir myir     487891 12 月 21 19:17 fsl-imx-xwayland-glibc-x86_64-myir-image-full-aarch64-myd-imx8mm-toolchain-5.4-zeus.testdata.json
```

4. How to Burn System Image

MYS-8MMX series CPU Module and development board designed by MYIR company are equipped with i.MX 8M Mini microprocessor of NXP company. MYS-8MMX series products have a variety of start-up methods, so different update system tools and methods are required .Users can choose different platforms to update according to their needs.In addition, since the startup mode needs to be adjusted when burning, the user can select the dial switch according to the table below.

Table 4-1.Select boot mode

Boot Mode	**SW1 (1/2/3/4)**	**SW2 (5/6/7/8)**	**Note**
TF Card	OFF/ON/OFF/ON	OFF/ON/OFF/ON	
eMMC	OFF/ON/ON/OFF	ON/OFF/ON/OFF	
QSPI	OFF/ON/OFF/OFF	OFF/OFF/OFF/ON	
USB Download	ON/OFF/X/X	X/X/X/X	

4.1. How to Flash with UUU Tool

1) Tools requirements

- A development board
- Type_C cable
- Official UUU tool (*03-Tools/tools/*MYS_8MMX_UUU.zip)

Note: UUU tool is not compatible with win7, please use win10 or Linux system.

2) Set up hardware

Select the boot mode and set the dial switch to USB download mode SW1 (on, off, x, x) SW2 (x, x, x), where "X" represents any setting. Then connect the J2 Type-C interface of the development board with the computer. Plug in the 5V/3A power adapter.

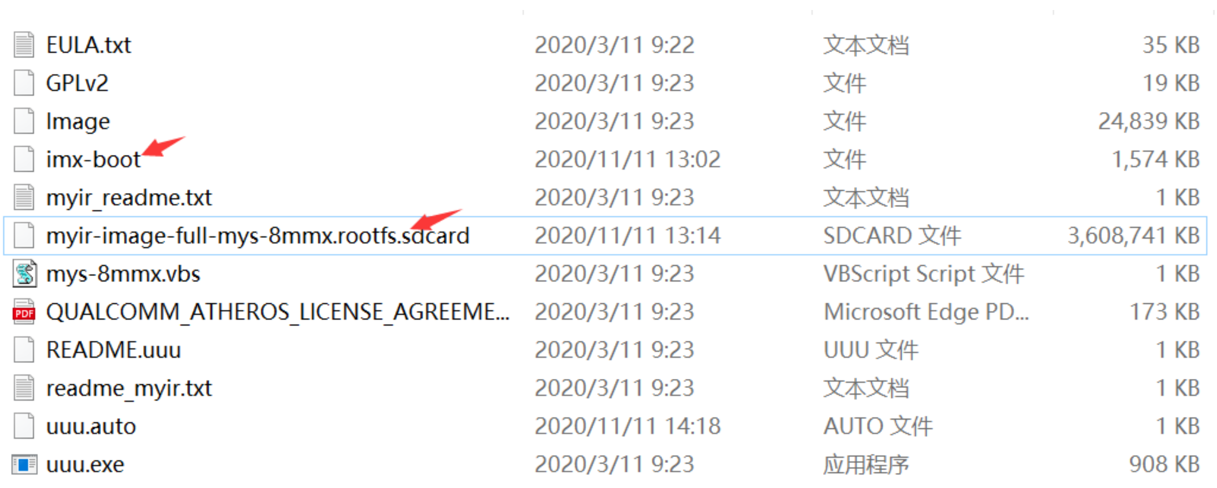
3) Burning System Under Windows

- **Prepare the image to be burned**

After building the image with yocto, please refer to Section 3.3 for a list of images.

- **Copy the compiled image to the uuu tool directory**

Copy *03-Tools/tools /MYS_ 8MMX_ UUU.zip* tools to the windows directory, unzip and copy the corresponding image to MYS_ 8MMX_ UUU directory (By default, the image has been copied to the directory), as shown in the following box:



EULA.txt	2020/3/11 9:22	文本文档	35 KB
GPLv2	2020/3/11 9:23	文件	19 KB
Image	2020/3/11 9:23	文件	24,839 KB
imx-boot	2020/11/11 13:02	文件	1,574 KB
myir_readme.txt	2020/3/11 9:23	文本文档	1 KB
myir-image-full-mys-8mmx.rootfs.sdcard	2020/11/11 13:14	SDCARD 文件	3,608,741 KB
mys-8mmx.vbs	2020/3/11 9:23	VBScript Script 文件	1 KB
QUALCOMM_ATHEROS_LICENSE_AGREEME...	2020/3/11 9:23	Microsoft Edge PD...	173 KB
README.uuu	2020/3/11 9:23	UUU 文件	1 KB
readme_myir.txt	2020/3/11 9:23	文本文档	1 KB
uuu.auto	2020/11/11 14:18	AUTO 文件	1 KB
uuu.exe	2020/3/11 9:23	应用程序	908 KB

Figure 4-1.UUU Tool Content

- **Burn image**

Double click the "MYS-8MMX-myir-image-full.vbs" file in the MYS-8MMX-OTG-DOWNLOAD tool directory to start burning the image, or enter the command to burn.Here are the steps to burn with the command:

First, open the CMD window with administrator privileges,then go to the MYS-8MMX-OTG-DOWNLOAD directory, and execute “uuu.exe uuu.auto” command to burn the system, as shown in the following log:

```
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.2.39-0-gdcc404f

Success 0    Failure 0
```

```
1:22   2/ 3   [=============78%=======>       ] SDPU: write -f imx-boot -offset
 0x57c00:
```

After burning, select eMMC to start.Please refer to table 4.1 for detailed settings on how to boot from eMMC.

4.2. Make TF Card Starter

Make TF card starter under Windows.

1) Preparation

- One TF card (not less than 8GB)
- MYS-8MMX full image
- Flash tool Win32DiskImager-1.0.0-binary.zip(Path: */03-Tools*)

Table 4-2.Image package list

Image Name	Package Name
myir-image-full	myir-image-full-myd-imx8mm.sdcard

2) How to make SD card starter

Take myir-image-full system as an example.

- **Prepare Tools and Image**

Image file：myir-image-full-myd-imx8mm.sdcard

Tool：win32DiskImager-1.0.0-binary.zip

- **Write image file to micro SD card**

Insert the TF card into the computer by using the card reader , and double-click to open Win32 DiskImager.exe and click the arrow to load the image file.

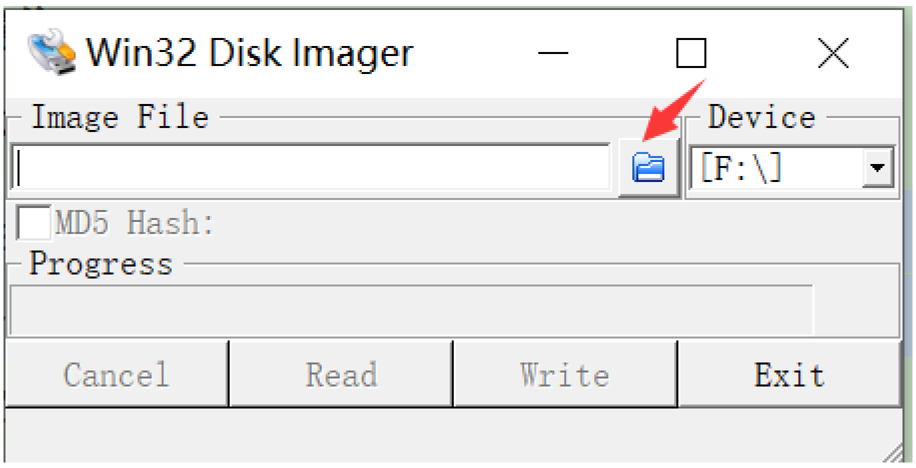
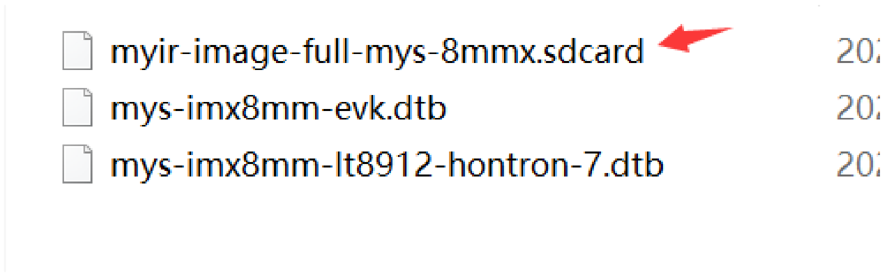


Figure 4-2.Tool configuration

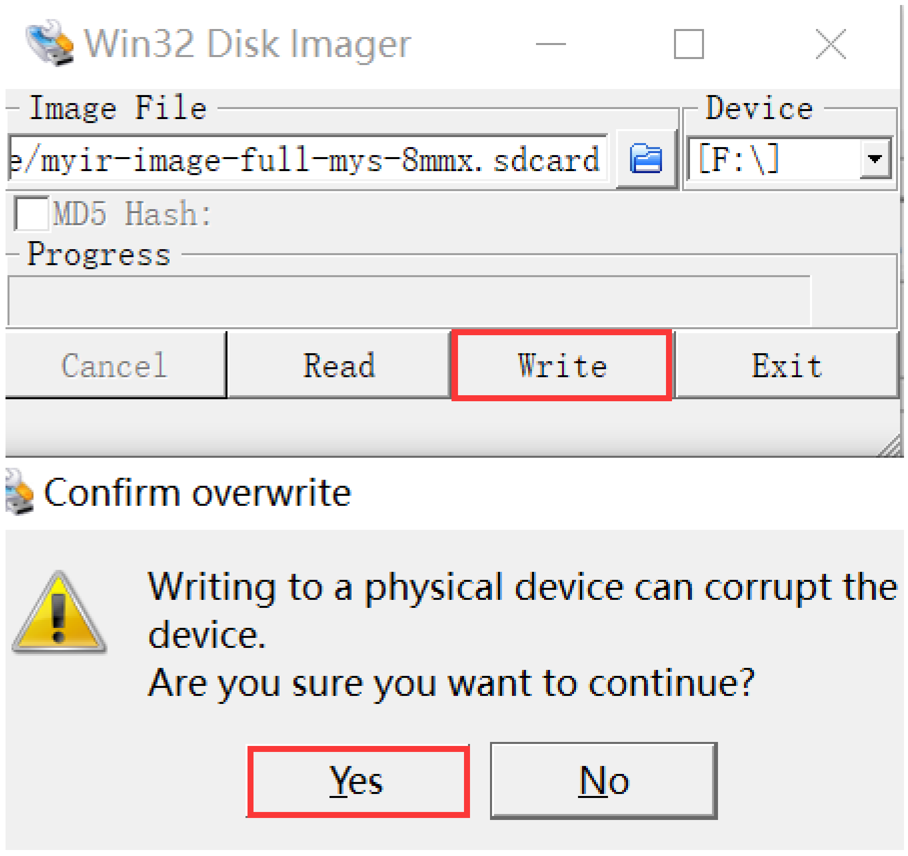
After selecting the system package, click open, as shown in the red arrow below:



myir-image-full-mys-8mmx.sdcard 20:
mys-imx8mm-evk.dtb 20:
mys-imx8mm-lt8912-hontron-7.dtb 20:

Figure 4-3.Select image

After loading the image, click the "write" button, and a warning will pop up. Click "yes" to wait for the completion of writing.



e/myir-image-full-mys-8mmx.sdcard

[F:\]

Writing to a physical device can corrupt the device.
Are you sure you want to continue?

Figure 4-4.Burning instructions

Wait for the write to complete, approximately 3-4 minutes, depending on the TF read/write speed :

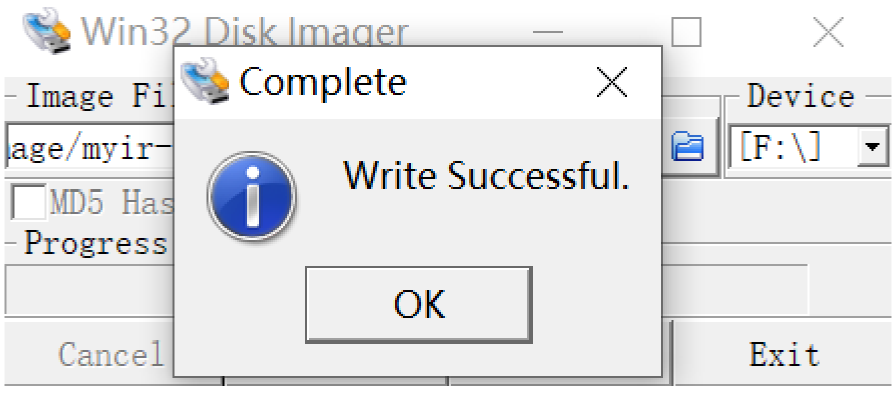

Win32 Disk Imager

Image Fi

age/myir-

MD5 Has

Progress

Cancel

Device

[F:\]

Exit

Complete

Write Successful.

OK

Figure 4-5.Write Successful

- **Check for success**

After burning, select SD Card to boot.Please refer to table 4.1 for detailed settings on how to boot from SD Card.

4.3. Make TF Card Burner

In order to meet the needs of burning image to eMMC, MYIR provides a burning method suitable for mass production.Here are the specific steps.

1) Execute script to set environment variables

The syntax for the envsetup.sh script is shown below:

```
EULA=1 DISTRO=fsl-imx-xwayland MACHINE=mys-8mmx source sources/meta-myir/tools/myir-setup-release.sh -b build-xwayland
```

2) Building myir-image-full image

Since the SD card burner image is to copy the whole system (myir-image-full) to the specified partition directory of eMMC, it is necessary to build the myir-image-full image before building the SD card burner image.

3) Build SD Card Burner Image

- Build the image

The following commands is an example on how to build the burner image:

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

bitbake myir-image-full

bitbake myir-image-burn
```

After the system is compiled successfully, the system image files of various formats are generated under the directory " *tmp/deloy/images/mys-8mmx/myir-image-burn-mys-8mmx.wic.bz2*".Then update according to section 4.2.

5. How to Modify Board Level Support Package

The previous chapter has described how to build a system image that can run on MYS-8MMX development board based on Yocto project. In addition, it also describes in detail the process of burning the image to the development board.This section describes the BSP layer created by myir, and also describes how to build the kernel, u-boot through a stand-alone environment and yocto project.

5.1. Introduction to meta-bsp

The "layer model" of Yocto project is a development model created for embedded and Internet of things Linux, which distinguishes Yocto project from other simple build systems. The layer model supports both collaboration and customization. A layer is a repository of related instruction sets that tell openembedded what to do with the build system.

The Yocto Project makes it easy to create and share a BSP for a new NXP based board.For example,the meta-bsp layer is based on the NXP official meta-imx layer ,which is suitable for MYS-8MMX development board,and the layer contains BSP, GUI, release configuration, middleware or application metadata and recipes.The contents of meta-bsp layer are as follows:

```
myir@myir-server1:  tree -a -L 1 meta-bsp/
meta-bsp/
├── classes
├── conf
├── recipes-bsp
├── recipes-connectivity
├── recipes-core
├── recipes-daemons
├── recipes-devtools
├── recipes-graphics
├── recipes-kernel
├── recipes-multimedia
├── recipes-myir
├── recipes-security
├── recipes-support
└── recipes-utils
14 directories, 0 files
```

Description of some layers:

Table 5-1.meta-myir-bsp layer content description

Source Code and Data	Description
conf	Including development board software configuration resource information
recipes-bsp	Contains configuration information such as atf、uboot and firmware
recipes-kernel	Contains Linux kernel resources and third-party firmware resources
recipes-myir	Contains configuration information for the file system

In the process of system transplantation, we should focus on the recipes-bsp part which is responsible for hardware initialization and system boot, and the recipes kernel which is responsible for Linux kernel and driver implementation.

5.2. Introduction to Board Level Support Package

In order to adapt to the new hardware platform of users, it is necessary to know what resources are provided by MYS-8MMX development board. For specific information, please refer to the MYS-8MMX SDK release notes. In addition, we also list some files that may need to be changed to facilitate users to search and modify. The specific contents are as follows:

Table 5-2.Add Configuration Information

Project	Device Tree	Description
U-boot	arch\arm\dts\mys-iot-imx8mm-ddr4-evk.dts arch\arm\dts\myir-imx8mm.dtsi	Device tree containing MYS-8MMX peripheral resources
Kernel	arch\arm64\configs\mys_iot_defconfig	Include board system configuration
	arch\arm64\boot\dts\myir\myb-imx8mm.dtsi	Including all peripheral drivers. Because customers mainly customize the board level driver, they need to modify the device tree to fit their own board
	arch\arm64\boot\dts\myir\mys-imx8mm-evk.dts	No display configuration
	arch\arm64\boot\dts\myir mys-imx8mm-evk-rpmsg.dts	Device tree of M4
	arch\arm64\boot\dts\myir\mys-imx8mm-lt8912-hontron-7.dts	Device tree for 7-inch screen display
	arch\arm64\boot\dts\myir\mys-imx8mm-lt8912-atk-10-1.dts	Device tree for 10-inch screen display

The normal startup process of MYS-8MMX is as follows:

➢ Bootrom：This is a tiny ROM or write protected flash memory embedded in the processor chip. It contains the first code that the processor executes on power up or reset. Depending on the configuration of some tape pins or internal fuses, it can decide where to load the next part of the code to be executed and how to verify its correctness or validity.

- ATF:Arm trusted firmware provides a reference implementation of safe world software for armv8-a, including security monitors executed at exception Level 3 (EL3). Various arm interface standards are implemented, such as power status coordination interface (PSCI), trusted board boot requirement (TBBR, ARM DEN0006C-1), SMC call, system control and management interface.
- UBOOT SPL：SPL's main function is to initialize external memory, and then copy uboot to external memory to run, so it runs in internal memory.
- UBOOT：The uboot initializes the appropriate system hardware and software environment to prepare for calling the operating system kernel.

The following chapters will describe the kernel development and application development in detail.

5.3. Compilation and Update of Bootloader

U-Boot ("the Universal Boot Loader" or U-Boot) is an open-source bootloader that can be used on ST boards to initialize the platform and load the Linux kernel. it is widely used in embedded system,do go further, please refer to the official site: http://www.denx.de/wiki/U-Boot/WebHome .

Since the i.MX8M/Mini bootloader consists of several parts, it is necessary to compile multiple pieces and then generate the final boot image. It consists of the following files:

- imx-atf：ATF is mainly responsible for switching between non secure environment and secure environment.
- myir-imx-uboot：i.MX8M/Mini platform boot from SPL, and different boot chains modes will correspond to different boot stages.You should pay attention to the difference between SPL compilation process and uboot main body compilation process.Normally, although the compilation command is the same, the main body uboot will be compiled first to generate uboot.bin Next, uboot SPL is compiled to generate uboot- spl.bin.
- firmware-8.7：Bin format file of firmware initialization, which is called by SPL or uboot to initialize DDR.
- imx-mkimage：Tool for generating the last burned image.

In addition, you can also use the following method to compile bootloader source code separately：

5.3.1. How to build and load U-Boot in Yocto Project

If you want to set up git repository to manage the code and build a new image completely according to the process in Chapter 3, first,you need to submit the modified U-boot source code to git repository,and then modify the corresponding source pull address(SRC_URI) and source revision(SRCREV). Take GitHub repository as an example:

1) View commit value

Go to the source code,the following commands is an example on how to get the latest commit value in the u-boot source code:

```
myir@myir-server1:$  cd myir-imx-uboot/
myir@myir-server1:$  git log
commit e584c34bf1fa25e50e74a3d5fcd88e13ac0d7739
Merge: 0169c66 ed3fb8d
Author: coin-haha <68100225+coin-haha@users.noreply.github.com>
Date:   Tue Oct 20 09:22:10 2020 +0800
    Merge pull request #4 from coin-haha/develop
    FEAT: add myir logo
commit ed3fb8daa45163db434c43d2c18a377f80e0f4c5
Date:   Tue Oct 20 09:20:20 2020 +0800

    FEAT: add myir logo
```

2) Modify source location

Go to the *sources/meta-myir/meta-bsp/recipes-bsp/u-boot* directory and find u-boot-imx_ 2019.04.bb file, modify the corresponding source location, as shown in the red mark below:

```
#UBOOT_SRC ?= "git://github.com/MYiR-Dev/myir-imx-uboot.git;protocol=https"
#SRCBRANCH = "develop"
#SRC_URI = "${UBOOT_SRC};branch=${SRCBRANCH} \
#"
#SRCREV = "e584c34bf1fa25e50e74a3d5fcd88e13ac0d7739"
UBOOT_SRC ?= "git:////${HOME}/MYS-8MMX/04-Source//myir-imx-uboot;protocol=file"
SRCBRANCH = "develop"
SRC_URI = "${UBOOT_SRC};branch=${SRCBRANCH}"
SRCREV = "e584c34bf1fa25e50e74a3d5fcd88e13ac0d7739"
```

- UBOOT_SRC:uboot code download address
- SRCBRANCH: Branch name

➢ SRCREV: commit value

3) Execute Script to Set Environment Variables

```
 myir@myir-server1:$    DISTRO=fsl-imx-xwayland MACHINE=myd-imx8mm sourc
emyir-setup-release.sh    -b build-xwayland
```

4) Build uboot

The following commands is an example on how to build U-Boot in Yocto Project :

```
myir@myir-server1:$ bitbake -c cleansstate u-boot
myir@myir-server1:$ bitbake -c cleanstate imx-boot
myir@myir-server1:$ bitbake imx-boot
```

Go to the *./tmp/deploy/images/myd-imx8mm* directory,and you will find the target image imx-boot.

5) Verify that uboot compiles successfully

After modifying and compiling uboot, you need to confirm whether uboot has been updated, and execute the following command to view the information:

```
myir@myir-server1:$ strings build-xwayland/tmp/deploy/images/myd-imx8mm/i
mx-boot | grep 2020
U-Boot SPL 2019.04-5.4.3-2.0.0+ge584c34 (Nov 12 2020 - 10:37:21 +0000)
11/12/2020
U-Boot 2019.04-5.4.3-2.0.0+ge584c34 (Nov 12 2020 - 10:37:21 +0000)
Built : 01:20:15, Nov 11 2020
```

It can be seen from the above "+ ge584c34" string that it is consistent with the previous character of the modified SRCREV value, that is, uboot has been updated.

5.3.2. How to update U-Boot separately

1) Burn to the specified partition in SD card

Insert the SD card into the computer with the card reader of a PC, read and take out all partitions, the USB card reader is /dev/sdb, partition is /dev/sdb,as shown in the figure below:

```
myir@myir-server1:$   cat /proc/partitions
```

```
major minor  #blocks  name
.......
 8       48   15273984 sdd
  8       49      85196 sdd1
  8       50    3510437 sdd2
```

To burn the boot image to the SD card, execute the following command:

```
 sudo dd if=imx-boot of=/dev/sdd bs=1k seek=33
sync
```

2) Burn to specified partition in eMMC

Copy the image(imx-boot) to the development board and view the corresponding partition:

```
root@myd-imx8mm:~# cat /proc/partitions
major minor  #blocks  name
.....
 179        0    7636800 mmcblk2
 179        1      85196 mmcblk2p1
 179        2    3510437 mmcblk2p2
```

The following command is an example on how to write to eMMC target partition via dd command:

```
root@myd-imx8mm:~# echo 0 > /sys/block/mmcblk2boot0/force_ro
root@myd-imx8mm:~# dd if=imx-boot of=/dev/mmcblk2boot0 bs=1k seek=33
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. Kernel Compilation and Update

Linux is a clone of the operating system Unix, written from scratch by Linus Torvalds with assistance from a loosely-knit team of hackers across the Net. It aims towards POSIX and Single UNIX Specification compliance.

It has all the features you would expect in a modern fully-fledged Unix, including true multitasking, virtual memory, shared libraries, demand loading, shared copy-on-write executables, proper memory management, and multistack networking including IPv4 and IPv6.

It is distributed under the GNU General Public License v2 - see the accompanying COPYING file for more details.

Linux kernel is also a very large open source kernel, which is applied to various distribution operating systems. Linux kernel is widely used in embedded system with its portability, network protocol support, independent module mechanism, MMU and other rich characteristics.Linux kernel.

MYS-8MMX adapts to NXP open source community version kernel version, and currently supports the latest version of Linux kernel 5.4.

5.4.1. How to build Kernel in standalone environment

1) Get the Linux source code

Get myir-imx-linux source code from github. Use the following command to download the latest source code：

```
PC$:cd mys-8mmx-yocto
PC$:git clone https://github.com/MYiR-Dev/myir-imx-linux.git -b develop
```

Or Get the source code from yocto project.After the user builds the system (please refer to Section 3.3 for details), this process will pull the uboot source code from GitHub to the local, and the user only needs to copy it to the same level directory, and then duplicate the name, as shown below:

```
PC$:cd build-xwayland/tmp/work-shared/mys-8mmx/
PC$:cp -r   kernel-source/  <PATH>/mys-8mmx-yocto/
PC$:mv  kernel-source myir-imx-linux
```

Where "path" is the user's own working directory.

2) **Configure Kernel Source Code**

- **Initialize cross-compilation via SDK**

Before compiling the u-boot separately, you need to initialize the SDK. Please refer to Section 2.3 for details.

- **Configure and Compile source code**

Go to the source code,execute the following command to configure and compile the source code

```
PC$ make distclean
PC$:make mys_iot_defconfig
PC$ LDFLAGS="" CC="$CC" make -j16

PC$ mkdir test

PC $ make modules_install INSTALL_MOD_PATH=./test
```

After compiling, the file generated in the source directory is as follows:

Table 5-3.Image File

Image FIle	Directory
Image	arch/arm64/boot/Image
DTB FIles	arch/arm64/boot/dts/myir /mys*.dtb

5.4.2. How to build Kernel in Yocto Project

You need to enter the Kernel source code directory to submit the modified Kernel source code, and then find the latest commit value in GitHub repository and copy the value,the following command is an example on how to get and modify the latest commit value in the Kernel source code:

1) **View commit value**

Go to the source code,the following commands is an example on how to get the latest commit value in the u-boot source code:

```
myir@myir-server1:$  cd myir-imx-linux/
myir@myir-server1:$  git log
commit 4023ef20e040f79b2be4b0ab5d664b0889d9701a
Merge: a1fc13d c768b9c
Author: coin-haha <68100225+coin-haha@users.noreply.github.com>
Date:   Mon Nov 9 16:05:07 2020 +0800
    Merge pull request #10 from coin-haha/develop
    FIX: configure J24 expand pin to gpio mode
commit c768b9c1afe60e2c8005d8748921ef8d91b96bc5
Merge: 1df160d a1fc13d
Date:   Mon Nov 9 16:01:07 2020 +0800

    Merge remote-tracking branch 'upstream/develop' into develop
commit 1df160d50142b521ab9e6467b58b11790dda3240
Date:   Mon Nov 9 15:55:36 2020 +0800
    FEAT: add myir logo
```

2) Modify source location

Go to the *sources/meta-myir/meta-bsp/recipes-kernel/linux/* directory and find *linux-imx_5.4.bb* file, modify the corresponding source location, as shown in the red mark below:

```
#KERNEL_BRANCH ?= "develop"
#LOCALVERSION = "-2.0.0"
#KERNEL_SRC ?= "git://github.com/MYiR-Dev/myir-imx-linux.git;protocol=https"
#SRC_URI = "${KERNEL_SRC};branch=${KERNEL_BRANCH}"
#SRCREV = "4023ef20e040f79b2be4b0ab5d664b0889d9701a"

KERNEL_SRC ?= "git:////media/duxy/wujl/mys-8mm/myir-imx-linux;protocol=file"
SRCBRANCH = "develop"
SRC_URI = "${KERNEL_SRC};branch=${SRCBRANCH}"
SRCREV = "4023ef20e040f79b2be4b0ab5d664b0889d9701a"
```

- KERNEL_SRC:Kernel code download address
- SRCBRANCH: Branch name
- SRCREV: commit value

3) Execute Script to Set Environment Variables

```
myir@myir-server1:$  DISTRO=fsl-imx-xwayland MACHINE=myd-imx8mm source
   myir-setup-release.sh   -b build-xwayland
```

4) Build Kernel

The following commands is an example on how to build U-Boot in Yocto Project :

```
myir@myir-server1:$  bitbake -c cleansstate virtual/kernel
myir@myir-server1:$  bitbake virtual/kernel
```

Go to the */tmp/deploy/images/myd-imx8mm* directory,and you will find the target image(Image).

5) Verify that uboot has compiled successfully

After modifying and compiling Kernel, you need to confirm whether kernel has been updated, and execute the following command to view the information:

```
myir@myir-server1:/media/duxy/wujl/mys-8mm/build-xwayland$ strings tmp/de
ploy/images/mys-8mmx/Image | grep 2020
Linux version 5.4.3+g4023ef2 (oe-user@oe-host) (gcc version 9.2.0 (GCC)) #1 SMP
 PREEMPT Fri Nov 13 02:23:27 UTC 2020
ENS2020
fsl,p2020-guts
CTION!22020
Option nocheck/check=none is deprecated and will be removed in June 2020.
BT.2020 Constant Luminance
BT.2020
BT2020_CYCC
BT2020_RGB
BT2020_YCC
ITU-R BT.2020 YCbCr
fsl,p2020-rev1-esdhc
```

```
fsl,p2020-esdhc
#1 SMP PREEMPT Fri Nov 13 02:23:27 UTC 2020
```

It can be seen from the above "+g4023ef2" string that it is consistent with the previous character of the modified SRCREV value, that is, uboot has been updated.

5.4.3. How to update Kernel separately

1) Burn to the specified partition in SD card

Insert the sd card into the computer with the card reader of a PC, read and take out all partitions, the USB card reader is /dev/sdb, partition is /dev/sdb,as shown in the figure below:

Insert the TF card into the host with a card reader, and the boot partition will be automatically mounted to the host /media/$user directory. Copy the Image and device tree to this directory.

```
 myir@myir-server1:$  cp -rf  *.dtb Image  /media/my-linux/boot/
```

2) Burn to specified partition in eMMC

Copy the Image and device tree file (*dtb) to the boot partition of the development board. Since the boot partition will mount automatically after starting the development board, execute the following command to view the boot mount point:

```
root@mys-8mmx:~# 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
```

It can be seen from the above that the boot partition will be automatically mounted in the /run/media/mmcblk2p1 directory, and then we can directly copy the image to this directory:

```
root@mys-8mmx:~#  cd /run/media/mmcblk2p1/
```

```
 root@mys-8mmx:~# cp -rf  *.dtb  Image ./
root@mys-8mmx:~#   sync
```

6. How to Fit Your Hardware Platform

6.1. How to create your own machine

In the development process, users sometimes need to customize the board configuration. This section will explain how to create your own machine through an example.

6.1.1. Create a board configuration under yocto project

1) Copy similar machine files

Copy a similar machine file and rename it to the specified name of your board, For example, the machine file(mys-8mmx.conf) similar to MYS-8MMX stores in the *sources/meta-myir /meta-bsp/conf/machine/* directory.The machine file is as follows:

```
PC$ cd sources/meta-myir/meta-bsp/conf/machine/
PC$ ls
include  mys-8mmx.conf
```

2) Copy and rename

After finding a similar machine file, copy and rename it as your own machine file, such as:

```
PC$ cp mys-8mmx.conf mys-test.conf
PC$ ls
include  mys-8mmx.conf  mys-test.conf
```

3) Add related compilation description in Readme

After creating the machine file, you need to add relevant compilation instructions to the readme file, enter the yocto working directory mys-8mmx-yocto, view and modify the REDME-IMXBSP file, as shown in the following red string:

```
duxy@myir-server1:/media/duxy/wujl/mys-8mm$ ls -l
Total 32
```

```
. . . . . . .
lrwxrwxrwx  1 duxy duxy   19 11 月 10 20:51 README -> sources/base/README
lrwxrwxrwx  1 duxy duxy   24 11 月 10 20:51 README-IMXBSP -> sources/meta-myir/README
. . . . . . . . . .
duxy@myir-server1:/media/duxy/wujl/mys-8mm$
 * MYIR i.MX 8MM IOT (mys-test)
Quick Start Guide
-----------------
Build images
---------------------
Building XWayland
---------------------------
 DISTRO=fsl-imx-xwayland MACHINE=mys-8mmx source sources/meta-myir/tools/myir-setup-release.sh -b build-xwayland
 DISTRO=fsl-imx-xwayland MACHINE=mys-test source sources/meta-myir/tools/myir-setup-release.sh -b build-xwayland
```

4) Compile and test

After creating the machine file, you can compile and test it. For example, execute the following command to compile the minimum image, and then install the image to the board for testing

```
 myir@myir-server1:$  DISTRO=fsl-imx-xwayland MACHINE=mys-test source sources/meta-myir/tools/myir-setup-release.sh -b build-xwayland
 myir@myir-server1:$  bitbake core-image-minimal
```

Go to the build-xwayland-test/*tmp/deploy/images/myd-test/* directory,and you will find the target image *core-image-minimal-myd-test.sdcard.bz2* .Then copy and unzip it, and write it to the TF card according to section 4.2,then boot from SD:

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

6.1.2. Creating board configuration file in uboot

In the development process, users generally need to create their own board configuration file. This section describes how to create their own board configuration file step by step through a simple example.

1) Create a board

Copy a similar board file and rename it to the specified name of your board, For example, the machine file similar to MYS-8MMX is in the uboot sources *board/myir* directory.The machine file is as follows:

```
myir@myir-server1: ls myir-imx-uboot/board/myir
common  myd_imx6ul_14x14  myd_imx6ull_14x14  mys_iot_imx8mm
```

2) Copy and rename

Go to the uboot source code *board/myir* directory, copy myd_imx8mm folder and rename it to mys_iot_imx8mm_test, as shown below:

```
 myir@myir-server1:$ cd   myir-imx-uboot
 myir@myir-server1:$  ls board/myir/
common  myd_imx6ul_14x14  myd_imx6ull_14x14  myd_imx8mm  mys_iot_imx8m
m mys_iot_imx8mm_test
```

Then go to mys_iot_imx8mm_test directory, rename “mys_iot_imx8mm.c” to “mys_test_imx8mm.c” ,next, modify makefile and replace “mys_iot_imx8mm.o” with “mys_test_imx8mm.o” ,as shown below:

```
myir@myir-server1:~/myir-imx-uboot/board/myir/mys_iot_imx8mm_test$ ls
built-in.o    Kconfig   lpddr4_timing.c  mys_iot_imx8mm.o   mys_test_imx8mm.c
ddr4_timing.c  lpddr4_timing_4g.c  Makefile
..................
myir@myir-server1:~/myir-imx-uboot/board/myir/mys_iot_imx8mm_test$ vi Mak
efile
.......
#
obj-y += mys_test_imx8mm.o
........
```

3) Make system configuration file

- **Modify the kconfig file of the new board**

Go to mys_iot_imx8mm_test directory, modify the kconfig file as shown in the following red mark:

```
 if  TARGET_MYS_IOT_IMX8MM_TEST_DDR4_EVK

config SYS_BOARD
        default "mys_iot_imx8mm_test"

config SYS_VENDOR
        default "myir"

config SYS_CONFIG_NAME
        default "mys_iot_imx8mm_test"

source "board/myir/common/Kconfig"

endif
 endif
```

Next, add the following red mark content to the *arch/arm/mach-imx/imx8m/Kconfig* file :

Note:This modification will affect the “make menuconfig” configuration information

```
config TARGET_MYD_IMX8MM_TEST
        bool "myd test iot ddr4 evk board"
        select IMX8MM
        select IMX8M_DDR4

  endchoice
  ……………
source "board/myir/myd_imx8mm_test/Kconfig"
```

- **Create the header file of the new board**

Go to the uboot source code *include/configs* directory, copy "mys_iot_imx8mm.h" file and rename it to "mys_iot_imx8mm_test.h", as shown below:

```
myir@myir-server1:$ ls include/configs/myd_imx8mm*
mys_iot_imx8mm.h   mys_iot_imx8mm_test.h
```

Note: due to SYS_CONFIG_NAME has been changed to mys_iot_imx8mm_test, so this header file needs to be changed to mys_iot_imx8mm_test.h.

- **Customize the configuration file of the system board**

Go to the uboot source code *configs* directory, copy "mys_iot_imx8mm_ddr4_evk_defconfig" file and rename it to "mys_test_imx8mm_ddr4_evk_defconfig", as shown below:

```
myir@myir-server1:$ ls configs/mys_*
mys_iot_imx8mm_ddr4_evk_defconfig   mys_test_imx8mm_ddr4_evk_defconfig
```

Then modify "mys_test_imx8mm_ddr4_evk_defconfig" file, as follows:

```
CONFIG_TARGET_MYS_IOT_IMX8MM_TEST_DDR4_EVK=y
#CONFIG_TARGET_MYS_IOT_IMX8MM_DDR4_EVK=y
```

After these steps, you have basically customized the board. If you need to compile with yocto, you need to modify the device tree configuration information in the machine according to section 6.1.1.

- **Compilation**

The following commands is an example on how to compile new board:

```
myir@myir-server1:$ make distclean
myir@myir-server1:$ make mys_test_imx8mm_ddr4_evk_defconfig
myir@myir-server1:$ make
```

6.2. How to Create Your Device Tree

Follow the sequences described in the below chapters to create the device-tree on your board.

6.2.1. Board Level Device Tree

A device tree is a tree data structure with nodes that describe the devices in a system. Each node has property/value pairs that describe the characteristics of the device being represented. Each node has exactly one parent except for the root node, which has no parent. … Rather than hard coding every detail of a device into an operating system, many aspect of the hardware can be described in a data structure that is passed to the operating system at boot time.

1) Introduction of imx-myir-uboot device tree

The following table lists various key device trees of MYS-8MMX board,which is very helpful for the development reference of users.

Table 6-1.MYS-8MMX U-Boot Device-tree List

Project	Device Tree	Description
U-boot	mys-iot-imx8mm-ddr4-evk.dts	DTB file containing MYS-8MMX peripherals
	myir-imx8mm.dtsi	Board level main DTB files, including all peripheral drivers
	fsl-imx8-ca53.dtsi	A53 cluster part of SOC: including CPU, PMU and PSCI (power state coordination interface

When compiling uboot source code of MYS-8MMX board, all relevant dts/dtsi files will be merged to generate the default “mys-iot-imx8mm-ddr4-evk.dtb” files used in uboot stage.

2) Introduction of imx-myir-linux device tree

The following table lists various key device trees of MYS-8MMX board,which is very helpful for the development reference of users.

Table 6-2.MYS-8MMX LinuxDevice-tree List

Project	Device Tree	Description
Kernel	myb-imx8mm.dtsi	Board level main DTB files, including all peripheral drivers
	mys-imx8mm-evk.dts	DTB file containing MYS-8MMX peripherals
	mys-imx8mm-lt8912-hontron-7.dts	Device tree configuration corresponding to 7-inch LVDS
	mys-imx8mm-lt8912-atk-10-1.dts	Device tree configuration corresponding to 10.1-inch LVDS

6.2.2. Add your board level device tree

1) Add board level device tree under the uboot

- **Select a similar device tree file**

Go to the uboot source code *arch/arm/dts* directory, copy “myb-fsl-imx8mm-ddr4-evk.dts” file and rename it to “ myb-test-imx8mm-ddr4-evk.dts” , as shown below:

```
PC$ cp mys-iot-imx8mm-ddr4-evk.dts mys-test-imx8mm-ddr4-evk.dts
```

- **Modify makefile of device tree**

Next, modify the makefile of the device tree in the directory, and add the following contents:

```
 .........
fsl-imx8mp-evk.dtb \
mys-iot-imx8mm-ddr4-evk.dtb \
mys-test-imx8mm-ddr4-evk.dtb
.......
```

- **Modify the board configuration file**

Modify the default device tree of the configuration file(mys_test_imx8mm_ddr4_evk_defconfig), as shown below:

```
CONFIG_CMD_EXT4_WRITE=y
CONFIG_CMD_FAT=y
CONFIG_DEFAULT_DEVICE_TREE="mys-test-imx8mm-ddr4-evk"
```

2) Add board level device tree under the Kernel

MYIR series device tree is in *arch/arm64/boot/dts/myir* directory. Users can also copy myir device tree and rename it in this directory. The method is similar to uboot. Please refer to the above steps.

6.3. How to configure function pins according to your hardware

Realizing the control of a function pin is one of the more complex system development processes, including pin configuration, driver development, application realization and other steps. This section does not specifically analyze the development process of each part, but explains the control implementation of function pin with examples.

6.3.1. GPIO pin configuration

The IO pins of MYS-8MMX board are generally defined in *arch/arm64/boot/dts/myir/mys-imx8mm-evk.dts* device tree file. each entry consists of 6 integers and represents the mux and config setting for one pin. The first 5 integers <mux_reg conf_reg input_reg mux_val input_val> are specified using a PIN_FUNC_ID macro, which can be found in pins-imx8mm.h under device tree source folder.Please refer to the i.MX8MMRM.pdf datasheet for details. An example of configuration is described below.

1) View pinctrl configuration rules

NXP pinctrl configuration format：<mux_reg conf_reg input_reg mux_val input_va
l pad_val >
Parameter Description:

- mux_reg：Multiplex register offset address
- conf_reg：Configure register offset address
- input_reg：Input register offset address
- mux_val: Multiplex register value
- input_val:Input register value
- pad_val:Configuration of pin speed, up and down, etc

For example, the pin configuration is as follows:

```
MX8MM_IOMUXC_GPIO1_IO09_GPIO1_IO9  0x19
```

It can be found in imx8mm-pinfunc.h under device tree source folder(*/arch/arm64/boot/dts/myir/*):

```
#define MX8MM_IOMUXC_GPIO1_IO09_GPIO1_IO9
 0x04C 0x2B4 0x000 0x0 0x0
```

You can check from the datasheet manual that the example is to set GPIO1_IO9 to fast and the driving ability to x1. Therefore, if you want to configure a pinctrl, it involves three registers, namely, multiplexing register, configuration register and input register.

2) Configure GPIO on the device tree

The following is an example of how to apply for and allocate device hardware resources in the DTS file. For example, users can define LED device nodes in the “mys-imx8mm-evk.dts ” DTS file, as follows:

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

		status = "okay";
		gpioctr-gpios = <&gpio3 16 0>;
	};
```

6.4. How to Use Your Own Configured Pins

The pin we configured in the u-boot or kernel device tree can be used in u-boot or kernel, so as to realize the control of the pins.

6.4.1. How to use GPIO in uboot

1) GPIO control through uboot command

In the uboot shell, you can directly use the command to control GPIO. For example, to set up GPIO5_IO4, use the following command in uboot shell.

```
u-boot=> gpio set 132
gpio: pin 132 (gpio 132) value is 1
u-boot=> gpio clear 132
gpio: pin 132 (gpio 132) value is 0
u-boot=>
```

You can see that the blue light(D27) is on or off.

2) GPIO control through device tree

You can also use the device tree to define the node of IO resources, and then implement the IO function in the code, such as the power reset control of PHY.

- **Configuration of device tree under uboot**

It can be found in *mys-iot-imx8mm-ddr4-evk.dts* under uboot device tree source folder(*arch/arm/dts/*), We use gpio4_22 as the reset pin of phy, and the configuration of device tree is as follows:

```
&fec1 {
   ..............
      phy-reset-gpios = <&gpio4 22 GPIO_ACTIVE_LOW>;
      status = "okay";
  ................
};
```

- **How to use reset pin in uboot**

After the reset pin is configured in the device tree, it needs to be used in uboot, as shown in the red mark below:

```
#ifdef CONFIG_FEC_MXC
#define FEC_RST_PAD IMX_GPIO_NR(4, 22)
static iomux_v3_cfg_t const fec1_rst_pads[] = {
	IMX8MM_PAD_SAI2_RXC_GPIO4_IO22 | 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);
	udelay(500);
	gpio_direction_output(FEC_RST_PAD, 1);
}
..........................
#endif
```

6.4.2. How to use GPIO in Kernel driver

1) How to use independent GPIO driver

In section 6.3.1, the GPIO device node information has been defined in the GPIO sample device tree. Next, we will use the kernel driver to realize the control of GPIO (set the GOIO5_IO4 pin to 1or 0, and use a multimeter to test the change of pin level if necessary).

```
//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.Determine the master device number  */
static int major = 0;
static struct class *gpioctr_class;
static struct gpio_desc *gpioctr_gpio;


/* 2. Implement the corresponding open/read/write functions and fill in the file_ o
perations structure*/
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;
}

/* Define your own file_ operations structure*/
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,
};
```

```
/* get GPIO resources from platform_ Device  and  register driver */
static int chip_demo_gpio_probe(struct platform_device *pdev)
{
        /*   Defined in device tree: 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);
        }

        /*  Register 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" },
    { },
};

/* define 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,
    },
};

/* Register platform_ driver in entry function*/
static int __init gpio_init(void)
{
    int err;
    err = platform_driver_register(&chip_demo_gpio_driver);

        return err;
}

/* If there is an entry function, there should be an exit function: when the driver is
unregister, the exit function will be called
    unregister platform_driver
```

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

/* Other improvements: provide equipment information and automatically create
device nodes */
module_init(gpio_init);
module_exit(gpio_exit);

MODULE_LICENSE("GPL");
```

The driver code can be compiled into a module using a separate Makefile, or it can be directly configured into the kernel.

2) How to compile the driver into kernel

Create a new gpioctr. c file in the *drivers/char/* folder of the kernel source code, then copy the above driver code, and modify kconfig, makefile and mys_iot_defconfig.

Add configuration in kconfig file：

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

Edit *drivers/char/Makefile*:

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

Add the configuration item in mys_iot_defconfig file:

```
CONFIG_SAMPLE_GPIO=y
```

Then compile and update the kernel according to section 5.4.

3) How to compile driver outside source tree

Add the gpioctr.c file in the working directory and copy the above driver code to the file, and write the independent Makefile program in the same directory.As shown below:

```
# Modify KERN_DIR
#KERN_DIR = # The directory of the kernel source code used by the board
KERN_DIR = /home/wujl/MYS-8MMX/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
#
If you want to compile a.c, b.c into ab.ko,To specify:# ab-y := a.o b.o
# # obj-m += ab.o
```

Then set up the host terminal window toolchain environment:

```
myir@myir-server1:~/MYS-8MMX/gpio$ source /home/wujl/MYS_SDK/opt/enviro
nment-setup-aarch64-poky-linux
```

Next, execute the make command to generate gpioctr.ko driver module file:

```
myir@myir-server1:~/MYS-8MMX/gpio$ make
make -C /home/wujl/MYS-8MMX/myir-imx-linux M=`pwd` modules
make[1]: Entering directory '/home/wujl/MYS-8MMX/myir-imx-linux'
  CC [M]  /home/wujl/MYS-8MMX/gpio/gpioctr.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/wujl/MYS-8MMX/gpio/gpioctr.mod.o
  LD [M]  /home/wujl/MYS-8MMX/gpio/gpioctr.ko
```

```
make[1]: Leaving directory '/home/wujl/MYS-8MMX/myir-imx-linux'
```

Finally, copy gpioctr.ko File to the */lib/modules* directory of the development board, and then use the insmod command to load the driver.

6.4.3. How to control a GPIO in Userspace

The architecture of Linux operating system is divided into user mode and kernel mode (or user space and kernel). User mode is the active space of the upper application. The execution of the application must rely on the resources provided by the kernel, including CPU resources, storage resources, I/O resources, etc. In order to enable the upper application to access these resources, the kernel must provide the access interface for the upper application: system call.

However, shell is a special application program, commonly known as the command line. It is a command interpreter in essence. It passes through system calls and various applications. With shell scripts, a very large function can be realized in a few short shell scripts, because these shell statements usually encapsulate the system calls. In order to facilitate the interaction between users and the system.

This article shows two ways to control a GPIO in userspace:

- Shell command
- System call

1) Realize pin control through shell command

Shell control pins are essentially implemented by calling the file operation interface provided by Linux. This section does not give a detailed description. Please refer to “MYS-8MMX_Linux_Linux_Software_Evaluation_Guide”, Section 3.1.

2) System call to realize pin control

A set of "special" interfaces provided by an operating system to a user program. The user program can obtain the services provided by the operating system kernel through this group of "special" interfaces, such as applying to open files, closing files or reading and writing files, and obtaining system time or setting timers through clock related system call.

At the same time, the pin is also a resource and can be controlled by system call. In section 6.4.2, we have completed the implementation of the pin driver. Now we can call and control the pin controlled by the driver.

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

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

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

	close(fd);

	return 0;
}
```

Copy the above code to an example-gpio.c file and Initialize cross-compilation via SDK:

```
myir@myir-server1:$  source /home/myir/MYS-8MMX_SDK/opt/environment-setu
p-aarch64-poky-linux
```

Use the compile command $CC to generate the executable file gpiotest:

```
myir@myir-server1:$  $CC gpiotest.c -o gpiotest
```

Finally,copy the executable file to the board directory(*/usr/bin/*),the following command is an example on how to run directly(“on” means set high, “off” means set low):

```
root@mys-8mmx:~#  gpiotest /dev/myir_gpioctr0 on
root@mys-8mmx:~#  gpiotest /dev/myir_gpioctr0 off
```

7. How to Add an Application

The porting of Linux applications is usually divided into two stages: development debugging and production deployment.

In the development and debugging stage,thanks to the SDK Customized by MYIR(Please refer to Section 2.3 for details), it is easy to develop and debug a application in standalone environment.However, in the production deployment stage, thanks to the Yocto project, users only need to write the recipe file for the tested application and put the source code in the corresponding directory. Then you can use bitbake command to rebuild the image and automatically package the application into the system.

7.1. Makefile-based project

Makefile is actually a file, which defines a series of Compilation Rules to guide the compilation of source code. After defining the Compilation Rules in Makefile, users only need one make command, and the whole project will be compiled automatically, which greatly improves the efficiency of software development. In the development of Linux programs, no matter the kernel, driver, application, makefile has been widely used.

However, make is a command tool to explain the rules of makefile file. When the make command is executed, the make command will search for makefile (or makefile) in the current directory, and then execute the operations defined in makefile. It can not only simplify the command line of the compiler terminal, but also automatically judge whether the original file has been changed, so as to automatically recompile the changed source code.

The following will take an example (to realize the key control LED light on and off) to describe the preparation of makfile and the execution process of make.The makefile rules are as follows:

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

- target:"target" can be an object file, an execution file, or a label.。
- prerequisites:It is the file needed to generate target.
- command:This is the command that make needs to execute.

```
 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
```

Description of some parameters:

- $(CURDIR)：Indicates the full path of the current directory of Makfile
- $(notdir $(path))：Remove the path name from the path directory and leave only the current directory name. For example, the current makefile directory *"/home/myir/MYS-8MMX/key_ led"*， Then target = key_ led
- $(patsubst pattern, replacement,text)： Replace the "pattern" characters in the text with replacement, such as $(patternsubst% C,% O, $(shell ls *. C)), which means to list the files with suffix(. c) in the current directory first, and then replace them with suffix(. o)
- CC：Name of C compiler
- CXX: Name of C++ compiler
- clean: It's an agreed goal

The detailed codes are as follows:

```
//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/event1 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/user/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/user/brightness"); //led on
                                else
                                        system("echo 0 > /sys/class/leds/user/brightness");//le
d off
                        }

                }

        }
      return 0;
```

```
}
```

Then execute the make command to compile and generate the executable file target on the target machine.

Set up the host terminal window toolchain environment:

```
myir@myir-server1:$  source /media/duxy/wujl/meta-SDK/environment-setup-aarch64-poky-linux
```

Execute make command to generate executable file:

```
myir@myir-server1:$   make
```

Finally,copy the executable file(target_bin) to the board directory(/usr/bin/),the following command is an example on how to run directly:

```
root@mys-8mmx:~# key_led_test /dev/input/event1 noblock
key test
key test
key test
```

7.2. Application based on QT

Qt is a cross-platform graphics application development framework that is used on different sizes of devices and platforms and offers different copyright versions for users to choose from. MYS-8MMX uses Qt version 5.13 for application development.In Qt application development, it is recommended to use QtCreator integrated development environment. Qt application can be developed under Linux PC, which can be automatically cross-compiled into the ARM architecture of development board.

The QtCreator installation package is a binary program that can be installed by executing it directly: *./qt-creator-opensource-linux-x86_64-4.1.0-rc1.run*,for details of installation and configuration, please refer to "MYS-8MMX QT application development notes" or get more development guidance from qtcreator official website: https://www.qt.io/product/development-tools .

7.3. How to start an application automatically

1) Application configuration in Yocto

If you want to use yocto Project to build the image file in the production deployment phase and include the application, you need to create a recipe for the application. Please refer to the site:

https://www.yoctoproject.org/docs/3.1.1/mega-manual/mega-manual.html#hello-world-example

Usually, our application also needs to realize self running after startup, which can also be realized in the recipes. Take a slightly more complex FTP service application as an example to illustrate how to use Yocto to build a production image containing specific applications.The FTP service program described in this section adopts open source proftpd, and the source codes of each version are located in ftp://ftp.proftpd.org/distrib/source/.

Before we start to write a recipe, we can find out whether the application, or a similar application's recipe, already exists in the current source code repository. The search method is as follows:

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

Note: before executing the bitmake command, make sure you have executed the environment variable settings script that builds the yocto project.Please refer to Chapter 3 for details.

You can also find recipes for the same or similar applications in openembedded's official website layer index:

http://layers.openembedded.org/layerindex/branch/master/layers/ .

For the method of writing new recipes, please refer to the new recipes section of yocto project complete manual:

https://www.yoctoproject.org/docs/3.1.1/mega-manual/mega-manual.html#new-recipe-writing-a-new-recipe .

This section focuses on how to port FTP services to the target machine. By searching the current source code repository, it is found that the recipe of proftpd already exists in the yocto project, but it is not added to the system image. The specific porting process is described in detail below.

- **Find proftpd recipe of yocto project**

```
myir@myir-server1:$   cd  build-xwayland
myir@myir-server1:$   bitbake -s | grep proftpd
proftpd                              :1.3.6-r0
```

Note: it can be seen that the proftpd recipe, version 1.3.6-r0, already exists in yocto project.

- **Separate compilation of proftpd**

```
myir@myir-server1:$  bitbake proftpd
```

- **Package proftpd into the file system**

Add a line in *conf/ local.conf* As follows:

```
IMAGE_INSTALL_append = "proftpd"
```

- **Rebuild file system**

```
myir@myir-server1:$  bitbake core-image-minimal
```

- **View services**

Check whether the service is running after burning the new image,the following command is an example on how to check the proftpd service:

```
# 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
```

Here is a supplementary explanation of FTP account settings. FTP client has three types of login accounts, which are anonymous account, normal account and root account.

- **Anonymous account settings**

The user name is FTP, and there is no need to set a password. After logging in, the user can view the contents in the system */var/lib/ ftp* directory, and has no write permission by default. Since the */var/lib/ftp* directory does not exist by default, users need to create a directory */var/lib/ftp* on the target machine.To avoid modifying the meta openembedded layer,this can be done by using a bbappend recipe. For example, Folder creation and permission changes provided in a "proftpd_1%.append" with these lines.

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

After editing proftpd_ 1%. append, place it in the */sources/meta-myir/meta-bsp/recipes-daemons/proftpd/* directory under the meta-myir-st layer. Then rebuild the image file for testing.

- **Ordinary account settings**

Using the commands of useradd and passwd on the target machine, you can create an ordinary user, and after setting the user password, the client can also log in to the user's home directory with the common account. If you need to include ordinary users when packaging images, you can refer to the site(https://www.yoctoproject.org/docs/3.1.1/mega-manual/mega-manual.html#ref-classes-useradd) ,Then rebuild the image. The specific method will not be discussed here.

- **Root account settings**

If you need to log in to the FTP server with root account, you need to modify "/etc/proftpd.conf" file, adding a row of configuration "RootLogin on" to the file. At the same time, you need to set the password for the root account. After the proftpd service is restarted, the client can log in to the target machine using the root account.

```
# systemctl restart proftpd
```

Note: in order to enable the root account to log in, the user generally needs to modify the "/etc/proftpd.conf " file configuration, which is only used for testing. For more configuration of this file, please refer to the site:http://www.proftpd.org/docs/example-conf.html.

2) How to start service automatically at boot time

This section will take proftpd recipe as an example to introduce how to add the application recipe and realize the startup of the program. Proftpd recipes are located in the source code repository layers(/meta-openembedded /meta-networking/recipes-daemons/proftpd). The directory structure is as follows:

```
myir@myir-server1:$ $  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 :Recipe for building proftpd service
- proftpd.service :Auto start service at boot time
- proftpd-basic.init:Start script for proftpd

The "proftpd_ 1.3.6.bb" recipe file specifies the source code path to obtain proftpd service program and some patch files for this version of source code:

```
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 \
"
```

In addition, the configuration(do_configure) and installation process(do_install) of proftpd are also specified in the recipe:

```

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
}
```

For more information about how to install tasks, refer to：
https://www.yoctoproject.org/docs/3.1.1/mega-manual/mega-manual.html#ref-tasks.

"proftpd_ 1.3.6.bb" recipe inheritance systemd.class class(Please refer to：layers/openembedded-core/meta/classes/systemd.bbclass). If you want to run the application service in the boot phase, you need to use the default enable variable(SYSTEMD_AUTO_ENABLE). For example, the user can set the variable(SYSTEMD_AUTO_ENABLE) to start the application service. The example is as follows:

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

At present, the target machine uses systemd tool as the initialization management subsystem. Systemd tool is a collection of basic components of Linux system, which provides a system and service manager. The running process number is PID 1 and is responsible for starting other programs. For an example of how to configure systemd under yocto project, please refer to the site:

https://www.yoctoproject.org/docs/3.1.1/mega-manual/mega-manual.html#selecting-an-initialization-manager .

The contents of the proftpd service file are as follows:

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

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

[Install]
WantedBy=default.target
```

- After:Indicates that the service will run after the network service is started.
- Type:systemd considers the service started up once the process forks and the parent has exited. For classic daemons use this type unless you know that it is not necessary. You should specify PIDFile= as well so systemd can keep track of the main process.
- ExecStart:Indicates the program to be started and its parameters.

For more information about systemd, please check this website:https://wiki.archlinux.org/index.php/systemd .

If you are adding your own application, you can also refer to the above example to create a recipe,enable a unit to start automatically at boot, and package it into the system image.

8. Reference

- **Linux kernel open source community**

  https://www.kernel.org/

- **Yocto Project BSP Development Guide**

  https://www.yoctoproject.org/docs/3.1.1/bsp-guide/bsp-guide.html

- **Yocto project Linux Kernel Development Manual**

  https://www.yoctoproject.org/docs/3.1.1/kernel-dev/kernel-dev.html

- **Yocto Development Guide**

  https://www.yoctoproject.org/

- **NXP Development Communities**

  https://community.nxp.com/

Appendix A

Warranty & Technical Support Services

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

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

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

Service Guarantee

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

Price

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

Delivery Time

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

Technical Support

MYIR has a professional technical support team. Customer can contact us by email (support@myirtech.com), we will try to reply you within 48 hours. For mass production and customized products, we will specify person to follow the case and ensure the smooth production.

After-sale Service

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

Technical support service

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

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

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

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

After-sales maintenance service

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

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

Warm tips

1. MYIR does not supply maintenance service to LCD. We suggest the customer first check the LCD when receiving the goods. In case the LCD cannot run or no display, customer should contact MYIR within 7 business days from the moment get the goods.
2. Please do not use finger nails or hard sharp object to touch the surface of the LCD.

3. MYIR suggests user purchasing a piece of special wiper to wipe the LCD after long time use, please avoid clean the surface with fingers or hands to leave fingerprint.

4. Do not clean the surface of the screen with chemicals.

5. Please read through the product user manual before you using MYIR' s products.

6. For any maintenance service, customers should communicate with MYIR to confirm the issue first. MYIR' s support team will judge the failure to see if the goods need to be returned for repair service, we will issue you RMA number for return maintenance service after confirmation.

Maintenance period and charges

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

Shipping cost

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

Products Life Cycle

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

Value-added Services

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

2. MYIR provides the services of OS porting, BSP drivers’ development, API software development, etc.

3. MYIR provides other products supporting services like power adapter, LCD panel, etc.

4. ODM/OEM services.

MYIR Electronics Limited

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

Yunli Inteiligent Park, Bantian, Longgang District.

Support Email: support@myirtech.com

Sales Email: sales@myirtech.com

Phone: +86-755-22984836

Fax: +86-755-25532724

Website: www.myirtech.com