

MYD-LD25X Linux Software Development Guide



File status: [] Draft [√] Release	FILE ID:	MYIR-MYD-LD25X-SW-DG-EN-L6.1.82
	VERSION:	V1.1[Doc]
	AUTHOR:	MSW0192
	RELEASE:	2024-09-11
	UPDATED:	2024-11-08

Revision History

VERSION	AUTHOR	PARTICIPANT	DATE	DESCRIPTION
V1.0[Doc]	MSW0192	MSW0041	2024-09-15	Official Release
V1.1[Doc]	MSW0192	MSW0041	2024-11-08	The adaptation for the MYD-LD257-8E1D model.



CONTENT

Revision History	- 2 -
CONTENT	- 3 -
1. Overview of System Development	- 5 -
1.1. Software Resources	- 6 -
2. Setting Up the Development Environment	- 7 -
2.1. Host Environment	- 7 -
2.2. Installation of cross-compilation toolchain	- 10 -
3. How to Flash the System Image	- 13 -
4. Separate Compilation and Update of the Board Support Package	- 14 -
4.1. Introduction to the Board Support Package	- 14 -
4.2. Bootloader Compilation and Update	- 16 -
4.3. Onboard Kernel Compilation and Update	- 19 -
5. Setting Up a Basic Yocto Environment	- 23 -
5.1. Introduction	- 23 -
5.2. Obtaining the Source Code	- 24 -
5.3. Quick Compilation of Development Board Image	- 25 -
5.4. Bulid SDK	- 30 -
6. Adapting Hardware to the Baseboard	- 31 -
6.1. Introduction to the meta-bsp Layer	- 31 -
6.2. How to Create Your Own Machine	- 33 -
6.3. How to Create Your Device Tree	- 37 -
6.4. Configuring CPU Function Pins for Your Hardware	- 39 -
6.5. Using Configured Pins	- 41 -
7. Application Development and Deployment	- 42 -
7.1. Applications Based on Makefile	- 42 -



7.2. Qt-Based Applications.....	- 47 -
7.3. Configuring Applications for Automatic Startup.....	- 48 -
8. References	- 53 -
Appendix A.....	- 54 -



1. Overview of System Development

The embedded Linux system is a relatively large software system, including the board support package BSP (Board Support Package), which can be narrowly understood as the system bootloader, drivers, etc.; the Linux system kernel, which includes numerous protocol stacks, resource management, and schedulers; the Rootfs root file system, the application runtime environment, initialization scripts, and service resources, etc. To integrate these resources into a whole that can be well managed, a framework is needed for management.

Embedded Linux system software has many open-source build frameworks, among which the more common ones are Buildroot, Yocto, Busybox and so on.

Yocto uses powerful resource integration methods and high customizable architecture, making it very convenient to build Linux systems suitable for embedded products. At the same time, Yocto is not only a system tool for creating files, but also provides a complete set of Linux-based development and maintenance workflows, allowing lower-level embedded developers and upper-level application developers to work within a unified framework, solving the scattered and unmanaged development forms of traditional development methods.

MYD-LD25X series development platform on, using Yocto to deploy development environments, build systems, and Linux applications integration and image updates, etc. After system developers are familiar with the Yocto development process in Chapter five, they can refer to the porting guide in Chapter six to customize the BSP according to actual project needs, allowing them to quickly port the system to hardware platforms based on the MYD-LD25X core board design.

Note: This document does not include an introduction to the Yocto project and basic knowledge related to Linux systems, and is suitable for embedded Linux system developers with some development experience. For specific functions that users may use during secondary development, we also provide detailed application development manuals.



1.1. Software Resources

The MYD-LD25X board runs an operating system based on Linux kernel version 6.1.82, providing a wealth of system and software resources. The development board comes pre-installed with the cross-compilation toolchain required for embedded Linux system development, ATF source code, U-boot source code, Linux kernel and various driver modules' source codes, as well as various development and debugging tools for Windows and Linux desktop environments, and application development examples. For detailed software information, please refer to Chapter 2 of the "MYD-LD25X SDK Release Note."

2. Setting Up the Development

Environment

This chapter provides guidance on setting up the environment based on the MYD-LD25X development board, including the necessary hardware and software components for the entire development process.

2.1. Host Environment

1) Host Hardware and Software Requirements

- **Host Hardware**

Yocto project builds require a development host with a dual-core CPU or better, at least 8GB of RAM, and a 400GB or larger hard drive. It can be a physical machine or a virtual machine running Linux.

- **Host Operating System**

Various Linux distributions can be used for building Yocto projects. Common choices include Fedora, openSUSE, Debian, Ubuntu, RHEL, or CentOS. This guide recommends Ubuntu 20.04 64-bit for development.

2) Host Environment Configuration

After installing the Ubuntu 20.04 64-bit system, you can make appropriate configurations to prepare for subsequent development.

- **Set Root Password**

For development operations, it is advisable to use a regular user account rather than the root account. This means that the hostname should match your username instead of 'root', in order to avoid potential permission-related issues during the development process.

```
$ sudo passwd root
```

- **Install SSH**

After installing the SSH service, you can connect to Ubuntu for subsequent development using an SSH2 connection with a remote serial debugging tool in a Windows environment. You can search online for tutorials on how to use the serial debugging tool.

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

To generate a key for the user, use the following commands:

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

- **Configure Samba**

Samba allows you to access Ubuntu's contents in folder format directly from Windows, making reading and writing more convenient. To install Samba, enter the following command:

```
$ apt-get install samba
```

In the `/etc/samba/smb.conf` file, add user configuration. The following configuration uses the username "myir" as an example; please adjust it according to your actual username. The following configuration can be directly added to the end of the configuration file:

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

Create an account and set a password:

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

`/etc/init.d/smbd restart` Restart the Samba service:

```
$ /etc/init.d/smbd restart
```


[ok] Restarting smbd (via systemctl): smbd.service.

- **Configure Git**

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

- **Install Necessary Tools**

This section is very important. Please ensure that the host environment correctly executes the following actions, and restart after completing the operations below.

```
$ sudo apt-get update
$ sudo apt-get install -y gawk wget git-core diffstat \
unzip texinfo gcc-multilib build-essential chrpath socat libsdl1.2-dev \
xterm sed cvs 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 cpio \
sudo locales bc libncurses5-dev screen flex bison vim-tiny \
device-tree-compiler xvfb libgtk2.0-dev libssl-dev net-tools libyaml-dev \
rsync liblz4-tool zstd python3-pip git-lfs iputils-ping jq
$ sudo rm -rf /var/lib/apt/lists/*
$ sudo chmod a+x /usr/bin/repo
$ sudo ln -s /usr/bin/python3 /usr/bin/python
$ sudo sed -i -e 's/# en_US.UTF-8 UTF-8/en_US.UTF-8 UTF-8/' /etc/locale.gen
$ echo 'LANG="en_US.UTF-8" | sudo tee /etc/default/locale > /dev/null
$ sudo dpkg-reconfigure --frontend=noninteractive locales
$ sudo update-locale LANG=en_US.UTF-8
$ sudo pip3 install pyusb usb crypto ecdsa crcmod tqdm pycryptodome pycr
yptodomex pyelftools
```

2.2. Installation of cross-compilation toolchain

After using Yocto to build the system image, we can also use Yocto to build an extensible SDK. The materials provided by MYiR contain a pre-compiled SDK package, located at: *03-Tools/Compile Toolchain/SDK*. The functionalities of the SDK files are described in the table below:

Table 2-1. Compilation Toolchain

Toolchain File Name	Description
myir-image-full-openstlinux-weston-myd-ld25x-x86_64-toolchain-4.2.4-snapshot.sh	Includes a standalone cross-development toolchain, also providing qmake, the target platform's sysroot, libraries, and header files required for Qt application development. Users can directly use this SDK to establish an independent development environment.

Here are the steps for installing the SDK:

- **Copy SDK to Linux Directory**

Transfer the SDK compressed package to the user's working directory in Ubuntu. This will give you the installation script file, which looks like this:

```
$ mkdir myd-ld25x-toolchain;cd ~/myd-ld25x-toolchain
$ cp ~/03-Tools/Compile Toolchain/* ./
$ ls ~/myd-ld25x-toolchain
myir-image-full-openstlinux-weston-myd-ld25x-x86_64-toolchain-4.2.4-snapshot.sh
```

- **Execute the Installation Script**

To execute the shell script with regular user permissions, it will prompt for the installation path, which defaults to the /opt directory. In this example, the Qt toolchain is installed in the directory /home/myir/myd-ld25x-toolchain, as shown below:

```
$ ./myir-image-full-openstlinux-weston-myd-ld25x-x86_64-toolchain-4.2.4-snapshot.sh
ST OpenSTLinux - Weston - (A Yocto Project Based Distro) SDK installer version 4.2.4-snapshot
```

```
=====
=====
Enter target directory for SDK (default: /opt/st/myd-ld25x/4.2.4-snapshot): ./
You are about to install the SDK to "/home/myir/myd-ld25x-toolchain". Proceed [Y/n]? y
Extracting SDK
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.
```

Initialize environment variables:

```
$ source ~/myd-ld25x-toolchain/environment-setup-cortexa35-ostl-linux
```

- **Test the SDK**

Use \$CC or \$CXX to check whether the cross-compiled gcc and g++ are installed correctly:

```
$ $CC -v
Using built-in specs.
COLLECT_GCC=aarch64-ostl-linux-gcc
COLLECT_LTO_WRAPPER=/home/myir/myd-ld25x-toolchain/sysroots/x86_64-ostl-linux/usr/libexec/aarch64-ostl-linux/gcc/aarch64-ostl-linux/12.3.0/lto-wrapper
Target: aarch64-ostl-linux
.....
Thread model: posix
Supported LTO compression algorithms: zlib zstd
gcc version 12.3.0 (GCC)
```

Note: The operation to initialize the environment variables will only apply to the currently opened terminal window. Any new window opened in any way will not load the LMA35 SDK cross-compilation toolchain by default. Therefore, if you need to perform Yocto



builds or other development tasks, please open a new window to avoid environmental errors caused by the cross-compilation toolchain.



3. How to Flash the System Image

The MYD-LD25X series development boards, designed by MYIR, are based on ST's STM32MP257 microprocessors and support various boot methods, requiring different update tools and methods. Users can choose from the following update options:

- STM32CubeProgrammer: A Windows software that allows direct programming of the board via Type-C, without needing additional storage media.
- TF Card Launcher: Suitable for development and debugging, it facilitates quick booting but cannot flash images.
- TF Card Programmer: Ideal for mass production, used for flashing eMMC.

For detailed image flashing methods, refer to the "MYD-LD25X Production Guide," which includes comprehensive preparation steps and operational instructions for each model and image type.

4. Separate Compilation and Update of the Board Support Package

4.1. Introduction to the Board Support Package

To adapt to a new hardware platform, you first need to understand the resources provided by MYIR's MYD-LD25X development board. For detailed information, please refer to the "MYD-LD25X SDK Release Note." Additionally, we have compiled a list of files that may need modification within the BSP (Board Support Package) to assist users in locating and making necessary changes. The specific details are outlined in the table below:

Table 4-1. Adding Configuration Information

Project	Device Tree	Description
U-boot	arch/arm/dts/myb-stm32mp257x-2GB.dts	U-Boot device tree definition file for 2GB RAM
	arch/arm/dts/myb-stm32mp257x-2GB-resmem.dtsi	Device tree configuration include file for reserved memory with 2GB RAM
	arch/arm/dts/myb-stm32mp257x-2GB-u-boot.dtsi	U-Boot specific device tree configuration include file
	arch/arm/dts/myb-stm32mp257x-base.dtsi	Basic device tree configuration include file
	arch/arm/dts/myb-stm32mp257x-1GB.dts	U-Boot device tree definition file for 1GB RAM
	arch/arm/dts/myb-stm32mp257x-1GB-resmem.dtsi	Device tree configuration include file for reserved memory with 1GB RAM
	arch/arm/dts/myb-stm32mp257x-1GB-u-boot.dtsi	U-Boot specific device tree configuration include file
	configs/myd_ld25x_2G_defconfig	2GB DDR U-Boot configuration file
	configs/myd_ld25x_1G_defconfig	1GB DDR U-Boot configuration file
Kernel	arch/arm64/configs/myd_stm32mp257x_defconfig	Default kernel configuration file
	arch/arm64/boot/dts/myir/myb-stm32mp257x-2GB.dts	Device tree definition file for 2GB RAM



	arch/arm64/boot/dts/myir/myb-stm32mp257x-2GB-ethswitch.dts	Device tree definition file for Ethernet switch with 2GB RAM
	arch/arm64/boot/dts/myir/myb-stm32mp257x-2GB-resmem.dtsi	Device tree include file for reserved memory with 2GB RAM
	arch/arm64/boot/dts/myir/myb-stm32mp257x-1GB.dts	Device tree definition file for 1GB RAM
	arch/arm64/boot/dts/myir/myb-stm32mp257x-1GB-ethswitch.dts	Device tree definition file for Ethernet switch with 1GB RAM
	arch/arm64/boot/dts/myir/myb-stm32mp257x-1GB-resmem.dtsi	Device tree include file for reserved memory with 1GB RAM
	arch/arm64/boot/dts/myir/myb-stm32mp257x-base.dtsi	Basic device tree configuration include file
	arch/arm64/boot/dts/myir/myb-stm32mp257x-ethswitch.dtsi	Ethernet switch device tree configuration include file

The following section focuses on the processes users undertake based on the Bootloader, Kernel, and Yocto source code and data we provide.

4.2. Bootloader Compilation and Update

U-Boot is a highly versatile open-source bootloader that supports kernel booting, downloading updates, and many other functions, making it widely used in embedded systems. For more information, visit the official website: U-Boot.

4.2.1. Obtaining U-Boot Source Code

You can obtain the U-Boot source code through one of the following methods:

1) From GitHub

Create a directory to store U-Boot and use the following commands to clone the source code from GitHub:

```
$ mkdir bsp;cd bsp
$ git clone https://github.com/MYiR-Dev/myir-st-u-boot.git \
-b develop-ld25x-v2022.10
```

2) From CD Image

After extracting the CD image, go to the 04-Sources directory and extract the source code package MYD-LD25X-Distribution-L6.1.82-V*.tar.gz to your host. Navigate to the extracted directory to access the U-Boot source code:

```
$ mkdir MYD-LD25X
$ tar xvf 04-Sources/MYD-LD25X-Distribution-L6.1.82-V*.tar.gz \
-C MYD-LD25X
$ cd MYD-LD25X/MYD-LD25X-Distribution-L6.1.82-V*/MYD-LD25X-Uboot-L2022.10-V*
```

4.2.2. Compiling the Bootloader

To compile the bootloader, first, check the directory structure of MYD-LD25X-Uboot-L2022.10-V*:

```
$ ls -la
build/ - Directory for compiled files
build-uboot-en.sh - English build script
build-uboot-zh.sh - Chinese build script
deploy/ - Directory for deployment files
```



```
$ ./build-uboot-en.sh
-----Script Start-----
Compilation chain is normal
Please select the DDR size configuration to build (1/2):
1 - 1GB model suitable for MYD-LD257-8E1D
2 - 2GB model suitable for MYD-LD257-8E2D
Please enter your choice (1 or 2): 2
*****
* Configuring U-Boot. *
*****
[ ] 100%
*****
* Compiling U-Boot. *
*****
[ ] 100%
*****
* Configuring Optee. *
*****
[ ] 100%
*****
* Building FIP files. *
```

Tel: +86-755-22984836

4.3. Onboard Kernel Compilation and Update

The kernel is the core of the operating system, responsible for managing hardware resources and system services. It handles critical tasks such as process management, memory allocation, device drivers, and file systems. The kernel is loaded during system startup to ensure smooth coordination between hardware and software. Common open-source kernels like the Linux kernel are widely used in various devices, from embedded systems to servers. For more information, please refer to the kernel's official website or community.

4.3.1. Obtaining Kernel Source Code

1) Getting the Source Code from GitHub

Create a kernel work directory and use the following commands to download the source code:

```
$ mkdir linux;cd linux
$ git clone https://github.com/MYiR-Dev/myir-st-linux.git \
-b develop-ld25x-6.1.82
```

2) Get the source code from the release information

After obtaining the published information, extract the source code compression package MYD-LD25X-Distribution-L6.1.82-V*.tar.gz under 04-Sources to the host for use.

```
$ mkdir MYD-LD25X
$ tar xvf 04-Sources/MYD-LD25X-Distribution-L6.1.82-V*.tar.gz \
-C MYD-LD25X
$ cd MYD-LD25X/MYD-LD25X-Distribution-L6.1.82-V*/MYD-LD25X-Linux-L6.1.82-V*
```

4.3.2. Compiling the Kernel

After extraction, check the directory structure of MYD-LD25X-Linux-L6.1.82-V*:

```
$ ls -la
build-linux-en.sh  build-linux-zh.sh  myir-st-linux  README_en  README_zh
```

➤ myir-st-linux: Linux source code, including the complete Linux source.

- 19 -

- Running the one-click compilation script will automatically decompress the source code package and compile it.

- 20 -

Build completed, output directory is: /media/home/beste/01_Pro/ld25x/bsp_hu/
MYD-LD25X-Linux-L6.1.82-V*/output

The final compiled files will be saved in the output directory at the same level as the script, with the following contents:

```
$ ls output/ -l
total 76040
Image
Image.gz
modules.tar.gz
myb-stm32mp257x-2GB.dtb
myb-stm32mp257x-2GB-ethswitch.dtb
myb-stm32mp257x-1GB.dtb
myb-stm32mp257x-1GB-ethswitch.dtb
```

4.3.3. Updating the Kernel

This section describes the file burning method, either via command on the development board or using the STM32CubeProgrammer tool connected via Type-C from a PC. For the STM32CubeProgrammer burning method, please refer to the "MYD-LD25X Mass Production Guide." In the operations below, mmcblk1 refers to the eMMC partition, and mmcblk0 refers to the TF card partition.

1) Replace the Kernel File Image on the Development Board:

- Replace Image.gz on the eMMC development board or TF card.

Copy the compiled Image.gz to the eMMC development board using methods such as scp or a storage medium like a USB drive. For example, using scp through the development board's serial debugging port, it is recommended to back up before replacing as follows:

```
# cd /boot
# mv Image.gz bak-Image.gz
# scp beste@192.168.40.21:~/scpfiler/Image.gz /boot
# ls /boot
bak-Image.gz
```

Image.gz
.....

Updating the system on a TF card follows a process similar to the one described above.

2) Replace the Device Tree (dtb) on the Development Board:

- **Replace the device tree file on the eMMC development board or TF card.**

Copy the compiled dtb file from the output directory to the eMMC development board. The method for copying all device tree files is similar to the operations described below. Here, we use myb-stm32mp257x-2GB.dtb as an example. Users of the MYD-LD257-8E1D should modify the device tree name to the corresponding 1GB file in the following steps. You can transfer the file to the development board using scp or storage media like a USB drive. In this example, we use scp. When executing from the debug serial port on the development board, it's recommended to back up the original file before replacing it as follows:

```
# cd /boot
# mv myb-stm32mp257x-2GB.dtb bak-myb-stm32mp257x-2GB.dtb
# scp beste@192.168.40.21:~/scpfiler/myb-stm32mp257x-2GB.dtb /boot
# ls /boot
.....
myb-stm32mp257x-2GB.dtb
bak-myb-stm32mp257x-2GB.dtb
.....
```

3) Update Modules on the Development Board

To update the modules, first back up /lib/modules/6.1.82 to linux-bak, then extract the compiled modules.tar.gz to the root directory to complete the update.

```
# cp /lib/modules/6.1.82 ~/linux-bak -rf
# rm /lib/modules/6.1.82 -rf
# tar xf modules.tar.gz -C / > /dev/null 2>&1
```



5. Setting Up a Basic Yocto

Environment

5.1. Introduction

Yocto is an open-source "umbrella" project that encompasses a range of sub-projects. It integrates these projects and provides a reference build project called Poky to guide developers on how to use these projects to build embedded Linux systems. It includes Bitbake, OpenEmbedded-Core, board support packages, and configuration files for various software packages.

MYD-LD25X provides Yocto-compatible configuration files to assist developers in creating a Linux system image that can be flashed onto the MYD-LD25X board. Yocto also offers extensive development documentation resources for developers to learn and customize their systems. Due to space limitations, a complete introduction to Yocto usage cannot be provided here; users are encouraged to search online for additional information.

This section is suitable for developers who need to deeply customize their file systems and wish to build a file system tailored to the MYD-LD25X series development boards using Yocto, as well as for those interested in its customization methods. Developers who are new to Yocto or do not have special requirements may directly use the pre-provided file systems from MYD-LD25X.

Note: Building Yocto does not require loading the SDK toolchain environment variables from section 2.3. Please create a new shell or open a new terminal window.

5.2. Obtaining the Source Code

We offer two methods for obtaining the source code: one is to download the compressed package from the 04-Sources directory of the release materials, and the other is to use repo to fetch the source code from GitHub, which is continuously updated. Users should choose the method that best suits their needs. Since Yocto requires downloading all software packages locally before building, MYD-LD25X has pre-packaged the relevant software to minimize repeated downloads. Release materials can be obtained from the MYiR Developer Center.

5.2.1. Obtaining the Source Code from Release Materials

The source package is available in MYD-LD25X-Distribution-L6.1.82-V*.tar.gz within the MYiR development package materials in the 04-Sources directory. Copy the compressed package to your specified directory, and extract it as follows:

```
$ mkdir -p MYD-LD25X
$ cd MYD-LD25X
$ tar -xvf MYD-LD25X-Distribution-L6.1.82-V*.tar.gz \
-C MYD-LD25X
$ cd MYD-LD25X/MYD-LD25X-Distribution-L6.1.82-V*/MYD-LD25X-Yocto-mickle
dore-V*
```

If you have already obtained the Yocto source code as described in this section, you can skip the following section 5.2.2.

5.2.2. Obtaining from GitHub

Currently, both the BSP source code and Yocto source code for the MYD-LD25X development board are hosted on GitHub and will be kept up-to-date. Please refer to the "MYD-LD25X SDK Release Note" for the repository address. Users can use repo to fetch and synchronize code from GitHub. The specific operation methods are as follows:

```
$ mkdir -p ~/bin
$ curl https://mirrors.tuna.tsinghua.edu.cn/git/git-repo > ~/bin/repo
$ chmod a+x ~/bin/repo
```



```
$ export PATH=~/.bin:${PATH}
$ export REPO_URL='https://mirrors.tuna.tsinghua.edu.cn/git/git-repo/'
$ mkdir -p MYD-LD25X-yocto
$ cd MYD-LD25X-yocto
$ repo init -u https://github.com/MYiR-Dev/myir-st-manifest.git \
--no-clone-bundle --depth=1 -m myir-stm32mp2-6.1.82-1.0.0.xml \
-b myd-ld25x-v24.06.26-mickledore
$ repo sync
$ ls
layers
```

After successfully synchronizing the code, you will find the same directory contents as in MYD-LD25X-Yocto-mickledore-V* within the MYD-LD25X directory.

5.3. Quick Compilation of Development Board Image

5.3.1. Execute Environment Variable Setup Script

Before building the system with the Yocto project, you need to set the appropriate environment variables. Therefore, each time you compile Yocto in a new terminal window, run the envsetup.sh script located in the Yocto source directory at layers/meta-myr/scripts/.

The procedure to set up the build environment is as follows. Depending on your development board model, you need to execute the appropriate command:

- **MYD-LD25X-8E256D model**

```
$ cd MYD-LD25X-yocto
$ DISTRO=openstlinux-weston MACHINE=myd-ld25x source \
layers/meta-myr/scripts/envsetup.sh
```

After running the configuration script, you will automatically enter the generated build-openstlinuxweston-my-d-ld25x directory.

Note: Yocto compilation should be performed as a regular user, not as root.

5.3.2. Build the Image

To understand the differences between the various images provided by MYiR, please refer to the "MYD-LD25X SDK Release Notes."

- **Build the myir-image-full Image**

After completing the environment setup and downloading the necessary files, you can build the myir-image-full image by executing the following command:

```
$ bitbake myir-image-full
NOTE: Started PRServer with DBfile: /media/home/beste/01_Pro/ld25x/build-op
enstlinuxweston-myd-ld25x/cache/prserv.sqlite3, Address: 127.0.0.1:40731, PID:
1971012
Loading cache: 100% |#####|
#####| Time: 0:00:01
Loaded 4749 entries from dependency cache.
Parsing recipes: 100% |#####|
#####| Time: 0:00:01
Parsing of 2997 .bb files complete (2994 cached, 3 parsed). 4754 targets, 515
skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION           = "2.4.0"
BUILD_SYS            = "x86_64-linux"
NATIVELSBSTRING      = "universal"
TARGET_SYS           = "aarch64-ostl-linux"
MACHINE              = "myd-ld25x"
DISTRO               = "openstlinux-weston"
DISTRO_VERSION        = "4.2.4-snapshot-20240903"
TUNE_FEATURES        = "aarch64 armv8a crc cortexa35"
TARGET_FPU           = ""
DISTRO_CODENAME       = "mickledore"
.....
```

- **Build the myir-image-core Image**

To build the myir-image-core image, use the following command, similar to the full image build:

```
$ bitbake myir-image-core
```

- **Build the myir-image-burn Image**

Before building the burn image, the full image needs to be built first, as the burn image requires copying the file system, kernel files, and other elements from the full image into the burn image system for flashing.

The myir-image-burn image comes in two versions: myir-image-burn-1G and myir-image-burn-2G, which are suitable for the MYD-LD257-8E1D and MYD-LD257-8E2D models, respectively. The burn image is used to flash to the TF card, and then after booting from the TF card, it is used to flash the myir-image-full image from the TF card to the eMMC, serving as a production-friendly image package. For specific instructions on how to flash this image, please refer to the "MYD-LD25X Production Guide". Depending on the development board model, you can construct the burn image by executing one of the following commands:

- MYD-LD257-8E1D

```
$ bitbake myir-image-burn-1G
```

- MYD-LD257-8E2D

```
$ bitbake myir-image-burn-2G
```

After the build completes, the compiled image will be located in tmp-glibc/deploy/images/myd-ld25x/. You can update the generated image file using STM32CubeProgrammer as described in the "MYD-LD25X Mass Production Guide."

- **Build the Raw File for TF Card Flashing**

If you need to flash the image onto a TF card, you must use the sdcard-raw-tools. First, build the sdcard-raw-tools by running:

```
$ bitbake sdcard-raw-tools
```

Then, navigate to the following directory:

```
$ cd tmp-glibc/deploy/images/myd-ld25x/scripts/
```

Once in the directory, run the provided script to see how to use it.

```
$ ./create_sdcard_from_flashlayout.sh
[ERROR]: bad number of parameters
```

Help:

```
./create_sdcard_from_flashlayout.sh [-h|--help] [--compress] <FlashLayout file>
```

```
-h      :      this help
--help:      this help
--compress:    compress the raw image generated
--force-rootfs: force to use predefined rootfs size (3906 MB)
```

By setting SDCARD_SIZE on shell environment or calling the script with it you can limit the size of RAW sdcard

SDCARD_SIZE=<value on MB>

ex.: SDCARD_SIZE=2048 ./script/create_sdcard_from_flashlayout.sh <flashlayout.t>

this exemple limit the size of sdcard to 2GB (2048MB)

By setting DEVICE on shell environment or calling the script with it you can customize the command

ex.: DEVICE=sdb ./script/create_sdcard_from_flashlayout.sh <flashlayout>

Next, I'll demonstrate how to use the script to generate a raw.xz compressed file using the burn image as an example. The raw file is used for flashing onto the TF card, and outputting it as an xz compressed file makes it easier to transfer.

To start creating the raw.xz file for the burn image, use the following command. The process is similar for full and core images; you just need to select the appropriate Flashlayout file:

```
$ ./create_sdcard_from_flashlayout.sh --compress \
../flashlayout_myir-image-burn/optee/FlashLayout_sdcard_myb-stm32mp257x-2GB-optee.tsv
```

```
Create Raw empty image: ../flashlayout_myir-image-burn/optee/../../FlashLayout
_sdcard_myb-stm32mp257x-2GB-optee.raw of 5442MB
```

```
Create partition table:
```

```
[CREATED] part 01:      fsbla1 [partition size 256.0 KiB]
[CREATED] part 02:      fsbla2 [partition size 256.0 KiB]
[CREATED] part 03:  metadata1 [partition size 256.0 KiB]
[CREATED] part 04:  metadata2 [partition size 256.0 KiB]
[CREATED] part 05:      fip-a [partition size 4.0 MiB]
[CREATED] part 06:      fip-b [partition size 4.0 MiB]
[CREATED] part 07: u-boot-env [partition size 512.0 KiB]
[CREATED] part 08:      bootfs [partition size 64.0 MiB]
[CREATED] part 09:  vendorfs [partition size 183.0 MiB]
[CREATED] part 10:      rootfs [partition size 4.0 GiB]
[CREATED] part 11:      userfs [partition size 1.1 GiB]
```

```
.....
```

It is important to note that the tsv file selected must be for the SD card, not for the eMMC. Once the process is complete, the raw.xz file will be output to the parent directory, which is tmp-glibc/deploy/images/myd-ld25x/:

```
$ ls ../*.raw.xz
-rw- 444893344 ../FlashLayout_sdcard_myb-stm32mp257x-2GB-optee.raw.xz
```

After extracting this file, you can use the resulting raw file for flashing, following the instructions in the "MYD-LD25X Mass Production Guide" in the Win32DiskImager burning section.

5.4. Bulid SDK

Yocto provides functionality to build an SDK toolchain, which includes tools, headers, and libraries for both low-level and application developers. This eliminates the need for users to manually create or compile dependency libraries. The SDK is used for compiling U-Boot and Linux kernel code and includes headers and libraries for the target system, making it easier for application developers to port applications to the target device. Here' s how to build the toolchain SDK:

This section provides a brief explanation on building the SDK provided by MYiR. Use the following command to generate the SDK package:

```
$ bitbake -c populate_sdk myir-image-full
```

After the build completes, the SDK installation package will be located in the tmp-glibc/deploy/sdk/ directory. For installation instructions, please refer to section 2.2.

```
myir-image-full-openstlinux-weston-myd-ld25x-x86_64-toolchain-4.2.4-snapshot.  
sh
```

6. Adapting Hardware to the Baseboard

6.1. Introduction to the meta-bsp Layer

The Yocto project's "layer model" is a development model designed for creating embedded and IoT Linux systems. It differentiates the Yocto project from other simpler build systems by supporting both collaboration and customization. A layer is a repository containing related sets of instructions that inform the OpenEmbedded build system on what to do.

The meta-myr layer is built on top of the meta-st layer from STMicroelectronics and is tailored for the MYD-LD25X development board. Within this layer, the meta-bsp layer includes various metadata and recipes for BSP, GUI, distribution configuration, middleware, or applications. Users can adapt their hardware designed for the MYD-LD25X development board based on this "layer model," customize their applications, and build a system image suited to their needs. This section primarily introduces the meta-myr layer, including its specific contents as follows:

```
$ tree -a -L 1 layers/meta-myr/meta-myr-stm32mp
layers/meta-myr/meta-myr-stm32mp
├── classes
├── CODE_OF_CONDUCT.md
├── conf
├── License.md
├── README -> README.md
├── README.md
├── recipes-bsp
├── recipes-connectivity
├── recipes-core
├── recipes-devtools
```

- └─ recipes-extended
- └─ recipes-graphics
- └─ recipes-kernel
- └─ recipes-myrir
- └─ recipes-security
- └─ recipes-st
- └─ recipes-support

Layer Details:

Table 6-1. Description of the meta-myrir Layer Contents

Source Code and Data	Description
conf	Contains current layer path information and machine software configuration.
recipes-bsp	Includes configuration information for ATF, U-Boot, and firmware.
recipes-kernel	Contains resources for the Linux kernel and third-party firmware.
recipes-myrir	Includes MYIR's custom package configuration for the file system.

When performing a system port, it is crucial to focus on the recipes-bsp section, which handles hardware initialization and system boot, and the recipes-kernel section, which is responsible for the implementation of the Linux system's kernel and drivers.

6.2. How to Create Your Own Machine

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

6.2.1. Creating a Board Configuration in Yocto

1) Select a Similar Machine File

Copy a similar machine file and rename it to your board's specific name. For example, a machine file similar to MYD-LD25X can be found in the directory layers/meta-myir/meta-myir-stm32mp/conf/machine. Navigate to this directory and list the files:

```
$ cd layers/meta-myir/meta-myir-stm32mp/conf/machine
$ ls
include  myd-ld25x.conf
```

2) Copy and Rename

Once you find a similar machine file, copy it and rename it to your own machine file. For instance:

```
$ cp myd-ld25x.conf test-myd-ld25x.conf
$ ls
myd-ld25x.conf  test-myd-ld25x.conf
```

3) Machine Configuration File

Some parameters to pay attention to in the MACHINE configuration file include:

```
KERNEL_DEVICETREE += "myir/myb-stm32mp257x-1GB.dtb \
                        myir/myb-stm32mp257x-2GB.dtb \
                        myir/myb-stm32mp257x-2GB-ethswitch.dtb \
                        myir/myb-stm32mp257x-1GB-ethswitch.dtb \
                        "
```

This specifies the device trees included with the kernel. Here, all types of device trees for myb-ld25x are specified. You can refer to section 4.1 for details on

available device trees. Note that this should point to the final compiled .dtb files, not the .dts files.

4) Compile and Test

After creating the machine file, you can compile and test it. Execute the following commands to compile the minimal image for testing. Make sure to modify the configuration command to use your custom MACHINE name and check the flash model type:

```
$ cd MYD-LD25X-yocto
$ DISTRO=openstlinux-weston MACHINE=test-myd-ld25x source layers/meta-myir/scripts/envsetup.sh
$ bitbake core-image-minimal
```

After compiling, the generated image is located at build-openstlinuxweston-test-myd-ld25x/tmp-glibc/deploy/images/test-myd-ld25x/. Copy this image and use STM32CubeProgrammer to flash it onto the development board, following the instructions in the "MYD-LD25X Production Guidance Manual," then start the test:

```
root@test-myd-ld25x:~#
```

6.2.2. Creating Board Configuration Files in U-Boot

In the development process, users typically need to create their own board configuration files according to their board requirements. This section will demonstrate, through a simple example, how to create your own board configuration files step-by-step.

1) Create Board

When creating your own board configuration, you can establish it by copying and renaming an existing board configuration file. These files are generally located in the board directory of the U-Boot source code. Navigate to the myir subdirectory under the U-Boot source board directory, then copy the myd_ld25x folder to create test_ld25x, as shown below:

```
$ cd MYD-LD25X-Uboot-L2022.10-V*/board/myir/
$ cp myd_ld25x/ test_ld25x -rf
$ ls
```

myd_ld25x test_ld25x

Enter the test_ld25x directory, rename myd-ld25x.c to test-ld25x.c, and modify the Makefile to change myd-ld25x.o to test_ld25x.o.

```
$ mv myd-ld25x.c test-ld25x.c
$ vi Makefile
$ cat Makefile
# SPDX-License-Identifier: GPL-2.0

obj-y += test-ld25x.o
```

2) Create the Board .config File

● Modify the New Board Kconfig

In the test_ld25x directory, modify the Kconfig file to match the following (with the red text indicating changes):

```
if ld25x

config SYS_BOARD
    default "test_ld25x"

config SYS_VENDOR
    default "myir"

config SYS_CONFIG_NAME
    default "test_ld25x"

endif
```

Next, add the following content (with the red text indicating changes) to arch/arm/mach-stm32mp/Kconfig.25x (this will affect make menuconfig):

```
source "board/st/stm32mp2/Kconfig"
source "board/myir/test_ld25x/Kconfig"
```

● Create the New Board Header File

From the root of the source directory, navigate to include/configs and copy myd_ld25x.h to test_ld25x.h:

```
$ cd include/configs
$ cp myd_ld25x.h test_ld25x.h
$ ls -l test_ld25x.h
-rw-rw-r-- 1 myir myir 1620 Sep  5 11:49 test_ld25x.h
```

Note: Since the `SYS_CONFIG_NAME` has been changed to `test_ld25x.h`, the header file should be renamed to `test_ld25x.h`.

● Customize the System Board Configuration File

Enter the configs directory from the root of the source code, and choose the appropriate defconfig based on the Flash model to copy as test-defconfig. Here, we use the 2GB version as an example.

```
$ cd configs
$ cp myd_ld25x_2G_defconfig test_ld25x_2G_defconfig
$ ls test_ld25x_2G_defconfig -la
-rw-rw-r-- 1 beste beste 1963 8  1 15:23 test_ld25x_2G_defconfig
```

After these steps, the board should be mostly customized. If you need to compile with Yocto, you will also need to adjust the U-Boot defconfig settings in the layers accordingly.

```
$ vi layers/meta-myir/meta-myir-stm32mp/recipes-bsp/u-boot/u-boot-stm32mp-
config.inc
...
UBOOT_CONFIG[default_stm32mp25] ?= "test_ld25x_2G_defconfig,,u-boot.dtb"
...
```

3) Compile

After submitting the modified U-Boot to Git, you need to compile it using Yocto. Please refer to section 4.2.2 for the details on how to perform this compilation.

6.3. How to Create Your Device Tree

Introduction to the Device Tree Hierarchy

A device tree is a data structure that describes the hardware layout of a system using a specific syntax. It stores information about both on-chip and off-chip devices. The device tree file is compiled into a Device Tree Blob (DTB) file by U-Boot or the kernel. During operation, this DTB is parsed to retrieve board-level device information. The DTB used by U-Boot and the kernel are consistent with each other.

1) Device Tree for MYIR-ST-U-Boot

Below is a list of the device tree files used in U-Boot for the MYD-LD25X board, for user reference:

Table 6-2. MYD-LD25X U-Boot Device Tree List

Component	Device Tree File	Description
U-boot	arch/arm/dts/myb-stm32mp257x-2GB.dts	Device tree definition file for 2GB RAM in U-Boot
	arch/arm/dts/myb-stm32mp257x-2GB-resmem.dtsi	Include file for reserved memory device tree configuration for 2GB RAM
	arch/arm/dts/myb-stm32mp257x-2GB-u-boot.dtsi	U-Boot specific device tree configuration include file
	arch/arm/dts/myb-stm32mp257x-1GB.dts	Device tree definition file for 1GB RAM in U-Boot
	arch/arm/dts/myb-stm32mp257x-1GB-resmem.dtsi	Include file for reserved memory device tree configuration for 1GB RAM
	arch/arm/dts/myb-stm32mp257x-1GB-u-boot.dtsi	U-Boot specific device tree configuration include file
	arch/arm/dts/myb-stm32mp257x-base.dtsi	Base device tree configuration include file

When compiling the U-Boot source code for MYD-LD25X, all related .dts and .dtsi files are merged to generate the default myb-stm32mp257x-2GB.dtb used in the U-Boot stage.

2) Device Tree for MYIR-ST-Linux

The device tree hierarchy for MYIR-ST-Linux is as follows: myb-stm32mp257x-base.dtsi + myb-stm32mp257x-ethswitch.dtsi + myb-stm32mp257x-2GB-resmem.dtsi -> myb-stm32mp257x-2GB-ethswitch.dts and myb-stm32mp257x-

2GB.dts. Below is a detailed list of the device tree files for MYD-LD25X for user development reference:

Table 6-3. MYD-LD25X Linux Device Tree List

Component	Device Tree File	Description
Kernel	arch/arm64/boot/dts/myir/myb-stm32mp257x-2GB.dts	Device tree definition file for 2GB RAM
	arch/arm64/boot/dts/myir/myb-stm32mp257x-2GB-ethswitch.dts	Device tree definition file for 2GB RAM with Ethernet switch
	arch/arm64/boot/dts/myir/myb-stm32mp257x-2GB-resmem.dtsi	Include file for reserved memory device tree configuration for 2GB RAM
	arch/arm64/boot/dts/myir/myb-stm32mp257x-1GB.dts	Device tree definition file for 1GB RAM
	arch/arm64/boot/dts/myir/myb-stm32mp257x-1GB-ethswitch.dts	Device tree definition file for 1GB RAM with Ethernet switch
	arch/arm64/boot/dts/myir/myb-stm32mp257x-1GB-resmem.dtsi	Include file for reserved memory device tree configuration for 1GB RAM
	arch/arm64/boot/dts/myir/myb-stm32mp257x-base.dtsi	Base device tree configuration include file
	arch/arm64/boot/dts/myir/myb-stm32mp257x-ethswitch.dtsi	Ethernet switch device tree configuration include file

6.3.1. Adding Device Trees

1) Uboot create a device tree

- **Creating Device Trees in U-Boot**

Enter the arch/arm/dts directory from the root of the source code, copy myb-stm32mp257x-2GB.dts to test-stm32mp257x-2GB.dts. The 1GB configuration is similar, just select the device tree corresponding to 1GB. For 2GB, refer to the following:

```
$ cp myb-stm32mp257x-2GB.dts test-stm32mp257x-2GB.dts
```

- **Modify the Device Tree Makefile**

Edit the Makefile in the same directory to include the new device tree. Add the following content:

```
...
dtb-$(CONFIG_STM32MP25X) += \
    stm32mp257f-dk.dtb \
    test-stm32mp257x-2GB.dts \
```

- 36 -

- **Modify the Board Configuration File**

Next, go to the configs directory in the source directory and update the default device tree in the test_ld25x_2G_defconfig configuration file:

```
...
CONFIG_ENV_SECT_SIZE=0x80000
CONFIG_DEFAULT_DEVICE_TREE="test-stm32mp257x-2GB"
...
```

2) Creating Device Trees in the Kernel

For MYiR series device trees located in arch/arm64/boot/dts/myir, you can create a new device tree by copying an existing MYiR device tree and renaming it. Follow a similar process to that used for U-Boot. Please refer to the steps mentioned above.

6.4. Configuring CPU Function Pins for Your Hardware

Implementing pin functionality control is a complex system development process that involves pin configuration, driver development, and application implementation. This section will explain pin control implementation through an example.

6.4.1. GPIO Pin Configuration Methods

On the MYD-LD25X board, most IO pin definitions are in the arch/arm64/boot/dts/myir/myb-stm32mp257x-base.dtsi device tree file. The ST official documentation provides IO mux functionalities and corresponding IOPAD attributes for each IO pin, so users need only include the macros for the desired IO pins and functionalities in the array.

1) View the Pinctrl Configuration Rules

st pinctrl configuration format: `#define STM32_PINMUX(port, line, mode)`
`((PIN_NO(port, line)) < 8) | (mode))`

Where:

- Port: Represents the GPIO port, such as the character 'D' in pin PD11.
- Line: Indicates the pin number within its port, starting from 0. For example, pin 11 in PD11.
- Mode: Represents the mode value for the pin's operational mode.

For mode values, refer to the parameter definitions in the file `include/dt-bindings/pinctrl/stm32-pinfunc.h`.

```
/* define PIN modes */
#define GPIO      0x0
#define AF0       0x1
.....
#define AF13      0xe
#define AF14      0xf
#define AF15      0x10
#define ANALOG    0x11
#define RSVD      0x12
```

2) Configure GPIO in the Device Tree

To request and allocate hardware resources using the DTS file, you can configure the GPIO in the `myb-stm32mp257x-base.dts` file. Define the LED device node as follows:

```
gpio_leds_test {
    compatible = "gpio-leds";

    led0 {
        label = "LED-Blue";
        gpios = <&gpioz 5 GPIO_ACTIVE_HIGH>;
        default-state = "on";
        linux,default-trigger = "heartbeat";
    };
};
```


6.5. Using Configured Pins

Once you have configured the pins in U-Boot or the kernel device tree, you can use these pins within U-Boot or the kernel to control them.

6.5.1. Using GPIO Pins in U-Boot

1) Controlling GPIO via U-Boot Command Line

U-Boot allows direct control of GPIO pins through commands. For instance, to control a User LED on the development board with GPIO pin PH4, you can use the following commands. The pin number calculation for GPIO PH4 is as follows:

Note: 72 is the ASCII value of 'H', and 65 is the ASCII value of 'A'.

```
STM32MP> gpio clear 116
gpio: pin 116 (gpio 116) value is 0
STM32MP> gpio set 116
gpio: pin 116 (gpio 116) value is 1
```

This will allow you to see the User LED (D8) on the development board turn on and off.

2) Controlling GPIO via U-Boot Code

You can also control GPIO values within U-Boot code. For example, to control the power reset of a PHY, you might implement it in the U-Boot code as follows:

- **Using the Reset Pin in U-Boot**

To use a reset pin in U-Boot, you can modify the file `board/myir/myd-ld25x/myd-ld25x.c`. Here's an example of how to use the reset pin in your U-Boot code:

```
static void led_gpio(void)
{
    unsigned int gpio_led=116;
    gpio_request(gpio_led, "led-gpio");
    gpio_direction_output(gpio_led, 0);
    gpio_set_value(gpio_led, 0);
    gpio_free(gpio_led);
}
```



7. Application Development and Deployment

Porting Linux applications generally involves two phases: development and debugging, and production deployment. During the development and debugging phase, you can use the SDK provided by your hardware vendor to cross-compile your applications and then transfer them to the target machine for testing. For the production deployment phase, you need to write recipe files for your application and use Bitbake to build the production image.

7.1. Applications Based on Makefile

Porting Linux applications generally involves two phases: development and debugging, and production deployment. During the development and debugging phase, you can use the SDK provided by your hardware vendor to cross-compile your applications and then transfer them to the target machine for testing. For the production deployment phase, you need to write recipe files for your application and use Bitbake to build the production image.

7.1. Applications Based on Makefile

A Makefile is essentially a document that defines a set of build rules and contains detailed information on how to compile the source code. Once a Makefile is written, you can use a single make command to automate the entire build process, significantly improving development efficiency. Makefiles are widely used in the development of Linux programs, including kernels, drivers, and applications.

make is a command-line tool that interprets the instructions in the Makefile. It simplifies the compilation process by executing the commands specified in the Makefile. When you run make, it looks for a file named Makefile (or makefile) in

the current directory and executes the corresponding commands. make automatically determines if source files have changed and recompiles only the modified files.

Here is an example illustrating how to write and execute a Makefile for a simple application that controls a button on the MYD-LD25X development board.

A Makefile follows a specific structure:

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

- target: This can be an object file, an executable, or a label.
- prerequisites: These are the files or targets needed to generate the target.
- command: These are the commands 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
```

Parameter Descriptions:

- \$(CURDIR): Represents the full path of the current directory where the Makefile is located.
- \$(notdir \$(path)): Removes the path from \$(path) and keeps only the current directory name. For example, if the current Makefile directory is /home/myir/MYD-LD25X/key_led, it will become TARGET = key_led.
- \$(patsubst pattern, replacement, text): Replaces occurrences of pattern in text with replacement. For example, \$(patsubst %c, %o, \$(shell ls *.c)) lists all .c files in the current directory and replaces their extension with .o.
- CC: The name of the C compiler.
- CXX: The name of the C++ compiler.

- clean: A conventional target used to clean up the build directory by removing generated files.

Key Implementation Code:

```
#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/LED-Blue/brightness";

    struct input_event event;

    system("echo none > /sys/class/leds/LED-Blue/trigger");

    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/LED-Blue/brigh
tness"); //led on
            else
```

```

system("echo 0 > /sys/class/leds/LED-Blue/brightness"); //led off
    }

}

}

return 0;
}

```

To compile and generate the executable file `target_bin` on the target machine using the `make` command, follow these steps:

```
$ source ~/ld25x_toolchain/environment-setup-aarch64-poky-linux
```

Execute `make`:

```
$ make
```

From the output of this command, you can see that the compiler used is the one specified by the `CC` variable defined in the setup script. Transfer the `key_led_test` executable file to the target board's `/usr/sbin` directory using a transfer method such as SCP or a USB drive. Then, execute the following command on the target board and press the USER button to observe the blue LED turning on and off:

```

# key_led_test /dev/input/event0
key test
key test
key test
key test

```



7.2. Qt-Based Applications

Qt is a cross-platform framework for developing graphical applications, used across various device sizes and platforms. It offers multiple licensing options for users. The MYD-LD25X uses Qt version 5.15.13 for application development. For Qt application development, it's recommended to use the Qt Creator integrated development environment (IDE). This allows for developing Qt applications on a Linux PC and automatically cross-compiling them for the ARM architecture of the development board. For more details, refer to the "MYD-LD25X QT Application Note."

7.3. Configuring Applications for Automatic Startup

1) Automatic Startup Service

The development board system typically includes a default autorun service, which automatically executes the `/usr/bin/autorun.sh` script at startup. The `autorun.service` file is located in `/lib/systemd/system/autorun.service`.

To check the status of the autorun service, use the following command:

```
# systemctl status autorun
● autorun.service - auto run hmi after weston
   Loaded: loaded (/lib/systemd/system/autorun.service; enabled; preset: enabled)
   Active: active (running) since Fri 2023-03-03 10:08:52 UTC; 542ms ago
   Process: 3912 ExecStart=/usr/bin/autorun.sh (code=exited, status=0/SUCCESS)
   Main PID: 3919 (mxapp2)
     Tasks: 6 (limit: 2032)
    Memory: 17.5M
    CGroup: /system.slice/autorun.service
            └─3919 /usr/sbin/mxapp2

3 03 10:08:53 myd-ld25x autorun.sh[3919]: libpng warning: iCCP: known incorrect sRGB profile
3 03 10:08:53 myd-ld25x autorun.sh[3919]: libpng warning: iCCP: known incorrect sRGB profile
```

To add commands that should be executed at startup, you can add them to the `/usr/bin/autorun.sh` script. This script is automatically run by the `autorun` service at boot time. Simply append your desired commands to this script to have them executed on startup.

```
# vi /usr/bin/autorun.sh
source /etc/profile.d/weston_profile.sh
source /etc/profile.d/pulse_profile.sh
/usr/sbin/mxapp2 &
```



```
echo "This is a test log" > /dev/ttySTM0
exit 0
```

The above operation will print a test log to the debug serial port, which can be observed after a reboot. Note that the logs from the autorun service will appear in the output of `systemctl status`, and you can only see these logs through this command, unless you configure the logs to be redirected to the `/dev/ttySTM0` debug serial port.

To restart and verify:

```
# systemctl restart autorun
This is a test log
```

2) Configuring an Application for Automatic Startup

To have your own application automatically start at boot in Yocto, you can create a recipe similar to the `autorun` example found in `meta-myr/meta-myr-stm32mp/recipes-myr/autorun`.

Navigate to the `autorun` package directory:

```
$ cd sources/meta-myr/meta-myr-stm32mp/recipes-myr/autorun
$ ls
autorun  autorun.bb
```

You will see that the `autorun` directory contains two parts: a `.bb` file and a `resources` directory with files used in the `autorun.bb` recipe.

You can follow this example to create a new package called `test-app`, add it to the `myir-image-core` image, and configure it for automatic startup.

Copy the `autorun` directory and rename it to `test-app`:

```
$ cd meta-myr/recipes-myr
$ cp autorun test-app -rf
$ ls
... autorun test-app ...
```

Next, change all file names within the `test-app` directory to `test-app`:

```
$ cd test-app
```

```
$ mv autorun.bb test-app.bb
$ mv autorun/ test-app/
$ cd test-app
$ mv autorun.service test-app.service
```

In the test-app directory, replace the original autorun application with a simple test shell script. The content is as follows:

```
$ ls
licenses autorun test-app.service
$ rm autorun
$ vi test-app.sh    #Create a script and add the following content, then save
and close
#!/bin/sh
echo "====This is a test shell script====" > /dev/ttyS0
```

This script is used to print a test message to the debug serial port. Next, add this script to the service configuration so that it runs when the test-app.service service is executed at boot time. The script is assumed to be located in the /usr/bin/ directory of the filesystem.

```
$ vi test-app.service    #Open the service file and modify it to match the f
ollowing content
[Unit]
Description=auto run hmi after weston
After=weston-graphical-session.service

[Service]
#Type=oneshot
Type=forking
#ExecCondition=/usr/bin/pgrep -x weston
ExecStart=/usr/bin/test-app.sh

[Install]
WantedBy=multi-user.target
```

After making the changes, go up one level to modify the `.bb` file. Open the `.bb` file and update it to match the content below. It is recommended to copy and paste the content directly:

```
$ cd ../
$ vi test-app.bb
SUMMARY = "Test app"
DESCRIPTION = "test application"
LICENSE = "GPL-2"
LIC_FILES_CHKSUM = "file://licenses/GPL-2;md5=94d55d512a9ba36caa9b7df079
bae19f"

S = "${WORKDIR}"

SRC_URI = " \
    file://licenses/GPL-2 \
    file://test-app.sh \
    file://test-app.service \
    "

inherit systemd

do_install() {
    install -d -m 755 ${D}/etc
    install -d -m 755 ${D}${systemd_system_unitdir}

    install ${WORKDIR}/test-app.sh ${D}/usr/bin/
    install -m 644 ${WORKDIR}/test-app.service ${D}${systemd_system_unit
dir}/test-app.service
}

SYSTEMD_PACKAGES = "${PN}"
SYSTEMD_SERVICE_${PN} = "test-app.service"
SYSTEMD_AUTO_ENABLE_hmi = "enable"
```

```
FILES_${PN} += "/"
```

After saving, the test-app package is created. Next, add test-app to myir-image-core by opening the core image's .bb file and including test-app:

```
$ vi meta-myir/recipes-images/images/myir-image-core.bb
....
IMAGE_INSTALL_append = "\
    test-app \
```

Save the changes and build the core image:

```
$ bitbake myir-image-core
```

Then, update the built image to the development board and start it to observe the changes.



8. References

- **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/>



Appendix A

Warranty & Technical Support Services

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

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

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

Service Guarantee

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

Price

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

Delivery Time

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

Technical Support



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

After-sale Service

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

The service covers:

Technical support service

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

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

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

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

After-sales maintenance service

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

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

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

Warm tips

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

Maintenance period and charges

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

Shipping cost

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

Products Life Cycle



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

Value-added Services

1. MYIR provides services of driver development base on MYIR's products, like serial port, USB, Ethernet, LCD, etc.
2. MYIR provides the services of OS porting, BSP drivers' development, API software development, etc.
3. MYIR provides other products supporting services like power adapter, LCD panel, etc.
4. ODM/OEM services.

MYIR Electronics Limited

Room 04, 6th Floor, Building No.2, Fada Road,
Yunli Intelligent Park, Bantian, Longgang District.

Support Email: support@myirtech.com

Sales Email: sales@myirtech.com

Phone: +86-755-22984836

Fax: +86-755-25532724

Website: www.myirtech.com