MYS-ZU5EV
Linux Development Guide

File Status : [ ] Draft [√] Release	**FILE ID :**	MYIR-MYS-ZU5EV-SW-DG-EN-L5.4.0
	VERSION :	V1.11
	AUTHOR :	Fengyong
	CREATED :	2021-05-25
	UPDATED :	2021-05-25

Revision History

VERSION	AUTHOR	PARTICIPANT	DATE	DESCRIPTION
V1.10	Fengyong		20210525	Initial version for MYS-ZU5EV
V1.11	Fengyong		20210721	Add vcu example

CONTENT

1. Overview

We have many open source system building frameworks on Linux system platform that facilitate the construction and custom development of embedded systems by developers, and on Xilinix's zynqMP platform, there are now more common buildroot, Petalinux and so on. The Petalinux project uses a more powerful and customized approach to build Linux systems for embedded products. It's not just a tool for making file systems, it also provides a complete Linux-based development and maintenance workflow that enables underlying embedded and upper-level application developers to develop within a unified framework.

This article focuses on the complete process of customizing a complete embedded Linux system based on petalinux projects and MYIR Core Boards, including the preparation of the development environment, the acquisition of code, and how to perform Bootloader, Kernel porting, custom rootfs for their application needs, and more. We first looked at how to build a system image for MYS-ZU5EV board based on the source code we provided, and how to burn the built image to the board. For those users who develop projects based on MYS-ZU5EV core boards, we highlight the methods and key points of porting this system to the user's hardware platform, and through some actual BSP migration cases and Rootfs custom cases, users can quickly customize the system image that is appropriate for their hardware.

This document does not cover petalinux projects and the basics of Linux systems, and is intended for embedded Linux system developers with some development experience.

1.1. Software resources

MYS-ZU5EV features a Linux 5.4.0-based operating system with rich system and other software resources. The development board comes with the cross-compilation tool chain, U-boot source code, source code of linux kernel and each drive module, application development samples, etc. needed for embedded Linux system development. For specific software information, please refer to the instructions in Chapter 2 of the “MYS-ZU5EV_SDK Release Notes”.

1.2. Document resources

Depending on the stages of the user's use of the board, the SDK includes different categories of documentation and manuals, such as release instructions, getting started guides, evaluation guides, development guides, application notes, FPGA development manuals, etc. The specific documentation list refers to the instructions in Table 2-4 of the “MYS-ZU5EV_SDK Release Notes”.

2. Development environment

This chapter mainly introduces some hardware and software environments required by MYS-ZU5EV development board in the development process, including the necessary development host environment, necessary software tools, code and data acquisition, etc.

2.1. Development host environment

This section describes how to build a development environment for the zynqMP family of processor platforms. By reading this section, you will learn about the installation and use of relevant hardware tools, software development and debugging tools. And can quickly build the relevant development environment, for the later development and commissioning preparation. The zynqMP family of processors consists of 4 ARM Cortex A53 cores and 2 ARM Cortex R5 cores.

- **Host hardware**

Petalinux construction of the project requires a higher requirement for the development host, requiring the processor to have a 2GHz CPU above Pentium4, more than 8GB of memory, 100GB hard drive or higher configuration. You can be the host of a Linux system or a virtual machine running a Linux system.

- **Host operating system**

Build the host operating system for the Petalinux project, here is recommended for the Ubuntu 16.04 64bit desktop version of the system, the subsequent development is also used as an example of this system.

- **Install necessary software package**

```
# sudo apt-get install tofrodos  iproute2 gawk
# sudo apt-get install gcc git make
# sudo apt-get install xvfb
# sudo apt-get install net-tools  libncurses5-dev  tftpd
# sudo apt-get install zlib1g-dev zlib1g-dev:i386 libssl-dev
# sudo apt-get flex bison libselinux1
# sudo apt-get install gnupgwgetdiffstatchrpathsocatxterm
#sudo apt-get install autoconflibtool tar unzip texinfo
```

```
# sudo apt–get zlib1g-dev gcc-multilibbuild-essential
# sudo apt-get libsdl1.2-dev libglib2.0-dev
# sudo apt-get install screen pax gzip tar
```

- **Confirm sh as bash**

Make sure that the default sh is bash, and if you don't point to bash, some of the actions in the following sections will be affected.

Follow the command below to view the sh

```
# ls -al /bin/sh
/bin/sh -> dash
```

If sh points to dash, you need to change it to bash

```
sudo dpkg-reconfigure dash
# ls -al /bin/sh
/bin/sh ->bash
```

Confirmation is a point to bash before you can proceed to the contents of subsequent chapters.

2.2. Introduction to software development tools

Many debugging, burning tools are used in customizing linux systems for the core of ARM Cortex A53, some of which are available under the CD directory 03-Tools provided by MYIR, in addition to the following tools, which are briefly described below :

petalinux-v2020.1-final-installer.run

This is the installation tool of petalinux, installed petalinux tool, you can build petalinux software system.

Download Link：https://www.xilinx.com/support/download.html

2.3. Install Petalinux tool

Be sure to install the petalinux tool with non-root permissions. Follow the command below to install petalinux2020.1. During installation, there will be prompts such as PetaLinux User License Agreement (EULA) that you need to press the keyboard "q" and then press "y" for agreement license confirmation. The use in this article <WORKDIR>to represent a working directory on the host, such as

"/home/work/", is to guarantee directory access. Copy "petalinux-v2020.1-final-installer.run" into the working directory.

```
# mkdir –p petalinux
#./petalinux-v2020.1-final-installer.run /home/work/petalinux
```

2.4. Install MYIR's custom SDK

After we've built the system image with Petalinux, we can also use Petalinux to build a scalable SDK. The CD provided by MYIR contains a compiled SDK package located at: 03-Tools/sdk-qt .tar.xz, which includes a separate cross-development tool chain that also provides qmake, sysroot of the target platform, libraries and header files on which Qt application development depends, etc. Users can use this SDK directly to create a stand-alone development environment, compile Bootloader, Kernel, or compile their own applications separately, a process that will be detailed in a later section. Here's how to install the SDK, as follows :

- **Copy the SDK to the Linux directory and unzip it**

Copy the SDK compression package to the user working directory under Ubuntu, such as /home/work, unzip the file, and get the installation script file, as follows :

```
# cd /home/work
# tar –jxvf sdk-qt.tar.xz
sdk.sh
```

- **Install SDK**

```
# ./sdk.sh
PetaLinux SDK installer version 2020.1
=======================================
You are about to install the SDK to "/opt/petalinux/2020.1". 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.
```

- **Select the installation directory**

SDK is installed by default in the /opt/petalinux/2020.1/directory, and users can also customize the installation path, such as installing to/home/work/sdk directory :

```
# ./sdk.sh -d /home/work/sdk -y
```

- **Test SDK**

Once the installation is complete, use the following command to set the environment variables to test whether the SDK was installed successfully :

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux

# $CC --version
aarch64-xilinx-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.
```

In addition to the cross-toolchain, MYIR's SDK includes resources for developing Qt applications such as Qt libraries, qmake, and so on, which are the basis for subsequent application development and debugging using QT Creator.

3. Use Petalinux to build images

3.1. Introduction

Petalinux is Xilinx's embedded Linux development kit, which includes source code such as Linux Kernel, u-boot, device-tree, rootfs, and can be easily generated, configured, and compiled for use by MYS-ZU5EV boards. For the basics of Petalinux, please refer to https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug1144-petalinux-tools-reference-guide.pdf.

In the optical image provided by MYIR CD 04_sources a Petalinux BSP package for the MYC-ZU5EV board is available in the catalog to help developers build a Linux system image that can run on the MYS-ZU5EV board. The following is an example of building image to describe the specific development process, for the subsequent customization of their own system image to lay the foundation.

3.2. Get the source code

Get the compression package directly from the MYIR CD 04-sources catalog and ask the user to build on it.

3.2.1. Get the source compression package from the CD

The compressed source package is located at MYIR Development Pack Profile 04-Sources/Petalinux/mys_zu5ev2020_4G_core.bsp. Copy the compression package to a user-specified directory, such as the /home/work/petalinux directory, which will serve as the top-level directory for subsequent builds :

```
#cd home/work/petalinux
```

3.2.2. Get the source code via github

The Petalinux BSP, Kernel, and u-boot source codes of the MYS-ZU5EV development board are currently github and will remain up to date, and the code warehouse address is available in the “MYS-ZU5EV_SDK Release Notes” . Users can use git commands and synchronize code on github. Here's how :

```
# mkdir  /home/work/github
# cd /home/work/github
# git clone https://github.com/MYiR-Dev/myir-zynqMP-uboot.git
# git clone https://github.com/MYiR-Dev/myir-zynqMP-kernel.git
```

3.3. Quickly compile images

You need to set the appropriate environment variables before you can build your system using the Petalinux project.

```
#source /home/work/petalinux/settings.sh
```

- **Building image**

We chose the petalinux bsp package mys_zu5ev2020_4G_core.bsp for mys-zu5ev-core as an example.

```
# petalinux-create -t project -s mys_zu5ev2020_4G_core.bsp
# cd mys_zu5ev
# petalinux-build
```

- **Generate image**

```
# petalinux-package --boot --fsbl images/linux/zynqmp_fsbl.elf --u-boot=images
/linux/u-boot.elf --pmufw --atf --fpga images/linux/system.bit --force
```

Once the system is built, system image in various formats are generated in the images/linux directory, and the following is a list of file information generated after the build：

Table 3-1. Images file description

FILE NAME	DESCRIPTION
BOOT.bin	Contains fsbl, pmufw, bl31, uboot, devicetree, fpgabit burning file
image.ub	Kernel and device tree burn packages generate files
rootfs.tar.gz	Rootfs system

3.4. Build SDK

MYIR already offers a more complete SDK installation package that users can use directly. But when users need to introduce new libraries into the SDK, they need to re-build new SDK tools using Petalinux.

This section simply provides build instructions for the SDK provided by MYIR, using the following build command to generate the SDK package：

```
#petalinux-build --sdk
```

Building an SDK takes a long time, waiting for the build to complete and generating the SDK installation package as "/images/linux/sdk .sh", see section 2.4.

4. How to burn the system image

MYIR designed the MYS-ZU5EV series core board has two start-up methods, so different update system tools and methods are required.

- ➢ Making SD card launchers: suitable for research and development commissioning, quick start and other scenarios.
- ➢ Making SD card burners: suitable for mass production of burn eMMC.

4.1. Make an SD card launcher

The following steps are made under Windows.

1) Preparation

- ➢ SD card (not less than 4GB)
- ➢ MYS-ZU5EV development board
- ➢ Making images tools Win32DiskImager-0.9.5-install.exe （path：\03-Tools\）

Table 4-1. Image package list

Image Name	Package Name
mys-zu5ev-core	mys-zu5ev-core.img.gz
mys-zu5ev-full	mys-zu5ev-full.img.gz
mys-zu5ev-mipi	mys-zu5ev-mipi.img.gz

2) Make an SD card launcher

In the case of mys-zu5ev-core systems, other images burning methods are similar.

- **Unzip the resources below**

```
mys-zu5ev-core.img.gz
```

- **Write the image file to Micro SD Card**

Place the Micro SD Card in the card reader reader, then plug it into the computer, install win32DiskImager by Win32DiskImager-0.9.5-install.exe, then double-click on win32DiskImager.exe read out the USB stick partition, and click the folder icon to load the mirror file.

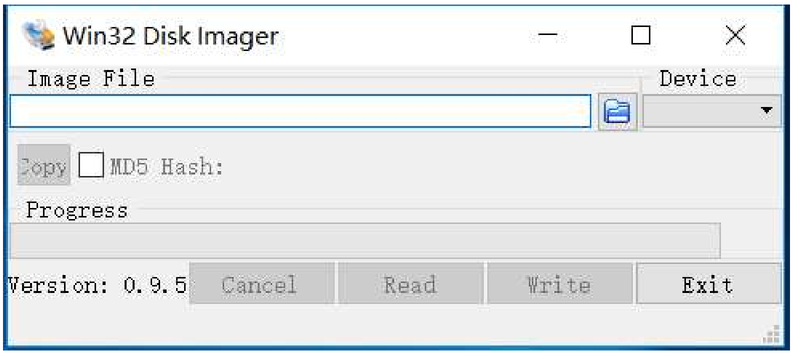
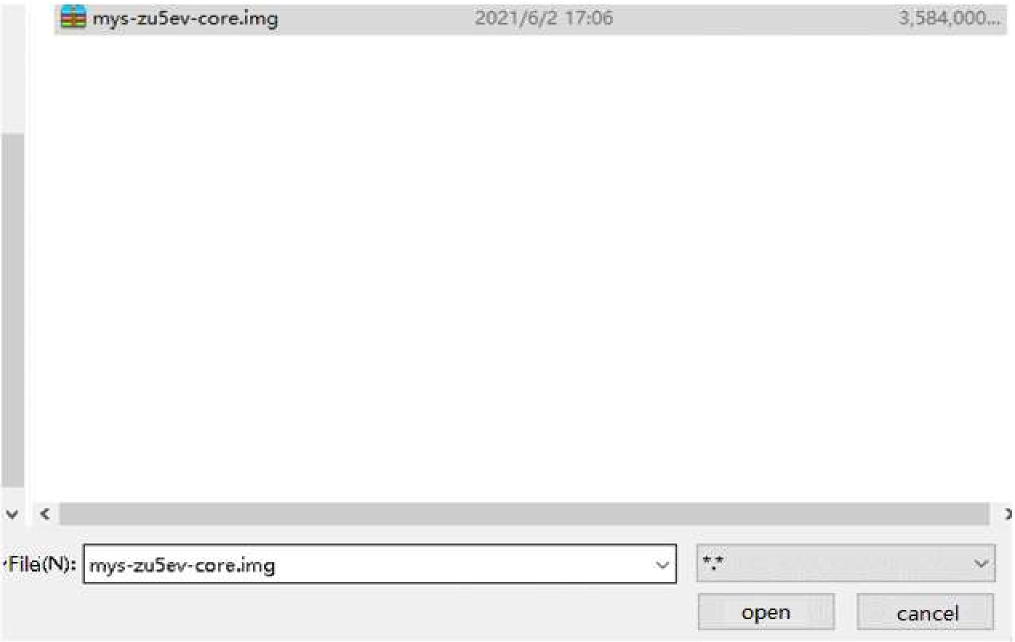


Figure 4-1. Tool configuration

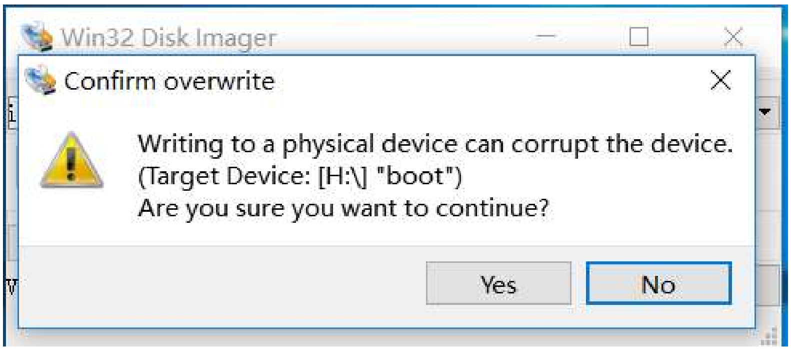
Select the system package and click Open.



File(N): mys-zu5ev-core.img

Figure 4-2. Tool configuration

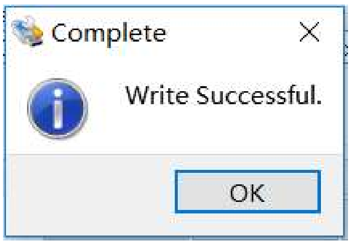
Once the image is loaded, click the "Write" button, a warning pops up, and click "Yes" to wait for the write to complete.



Writing to a physical device can corrupt the device.
(Target Device: [H:\] "boot")
Are you sure you want to continue?

Figure 4-3. Tool configuration

Wait for the write to complete, about a few minutes, depending on the read and write speed of the SD.



Complete

Write Successful.

Figure 4-4. Tool configuration

- **Check that the burn was successful**

When the write is complete, you can use this SD card to start, the SD card into the board SD card slot, and then the board's startup mode switch SW1 1 dial to OFF, 2 dial to ON, 3 dial to OFF, 4 dial to ON, set up as TF card start mode. Then power up and you can start the system with the SD card.

- **The debugging system generated by burning petalinux starts from QSPI flash**

（1）Using the tf startup card made in the previous step, copy the BOT.bin, image.ub, rootfs.tar.gz file generated by petalinux to the first partition of the tf card (boot partition).

（2）The board's start-up mode switch SW1 1 dial to OFF, 2 dial to ON, 3 dial to OFF, 4 dial to ON, and sets up into TF card start mode；

（3）Insert a TF card that has been stored in a burned-out file, connecting serial port rate of 115200, the development board power；

（4）The board will boot into the rootfs file system, into the Linux command line, and the input commands will begin to update：

```
#update /mnt/sd-mmcblk1p1
```

The script will burn BOOT.bin, image.ub to QSPI-Flash, rootfs.tar.gz to eMMC.

（5）After burning, the board's start-up mode switch SW1 1 dial to ON, 2 dial to OFF, 3 dial to ON, 4 dial to ON, set to Qspi flash boot mode, power up again, you can enter the burned rootfs file system.

4.2. Make an SD card burner

To meet the needs of production burn, MYIR developed a burning method for mass production. The system in the SD card will need to be burned and written into the on-board Flash. Follow these steps to complete the production process.

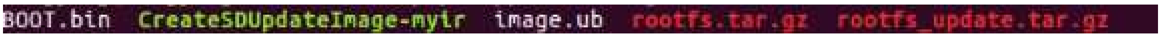
➢ Make update package

To make an installation package using the ubuntu system, the following resources need to be pre-installed：

```
sudo apt-get install kpartxfdisk mount dosfstools e2fsprogs pv
```

Copy 03-Tools/SDCardUpdate-mys-zu5ev.tar.xz resource pack to the ubuntu system from CD and unzip it.

```
# tar -xvf SDCardUpdater-mys-zu5ev.tar.xz

# ls
```



```
BOOT.bin  CreateSDUpdateImage-myir  image.ub  rootfs.tar.gz  rootfs_update.tar.gz
```

Figure 4-5. Include files

➢ BOOT.BIN、image.ub、rootfs.tar.gz：A startup file generated by Petalinux that needs to be burned

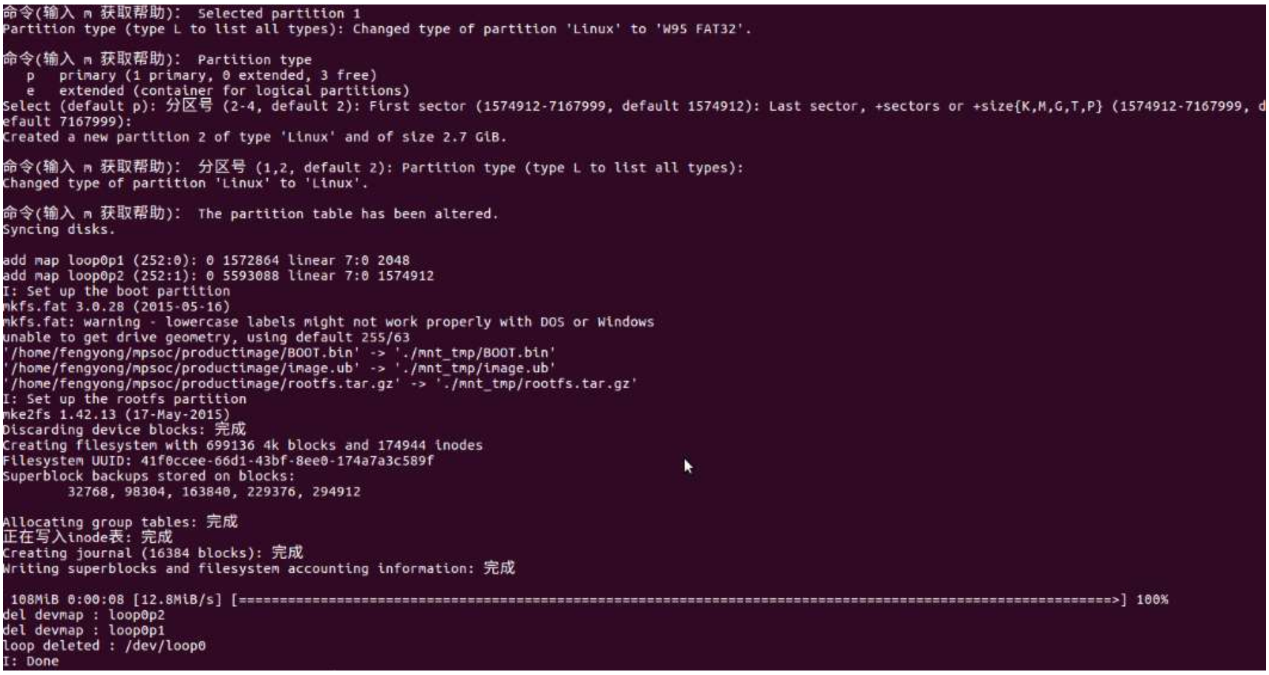
➢ CreateSDUpdateImage-myir：make script

➢ rootfs_update.tar.gz：The root file system that contains the auto-burner.

Burning packages can be done automatically with direct commands.

```
# sudo ./CreateSDUpdateImage-myir
```

Screenshots of the production process are below.



```
命令(输入 m 获取帮助):  Selected partition 1
Partition type (type L to list all types): Changed type of partition 'Linux' to 'W95 FAT32'.

命令(输入 m 获取帮助):  Partition type
   p   primary (1 primary, 0 extended, 3 free)
   e   extended (container for logical partitions)
Select (default p): 分区号 (2-4, default 2): First sector (1574912-7167999, default 1574912): Last sector, +sectors or +size{K,M,G,T,P} (1574912-7167999, d
efault 7167999):
Created a new partition 2 of type 'Linux' and of size 2.7 GiB.

命令(输入 m 获取帮助):  分区号 (1,2, default 2): Partition type (type L to list all types):
Changed type of partition 'Linux' to 'Linux'.

命令(输入 m 获取帮助):  The partition table has been altered.
Syncing disks.

add map loop0p1 (252:0): 0 1572864 linear 7:0 2048
add map loop0p2 (252:1): 0 5593088 linear 7:0 1574912
I: Set up the boot partition
mkfs.fat 3.0.28 (2015-05-16)
mkfs.fat: warning - lowercase labels might not work properly with DOS or Windows
unable to get drive geometry, using default 255/63
'/home/fengyong/mpsoc/productimage/BOOT.bin' -> './mnt_tmp/BOOT.bin'
'/home/fengyong/mpsoc/productimage/image.ub' -> './mnt_tmp/image.ub'
'/home/fengyong/mpsoc/productimage/rootfs.tar.gz' -> './mnt_tmp/rootfs.tar.gz'
I: Set up the rootfs partition
mke2fs 1.42.13 (17-May-2015)
Discarding device blocks: 完成
Creating filesystem with 699136 4k blocks and 174944 inodes
Filesystem UUID: 41f0ccee-66d1-43bf-8ee0-174a7a3c589f
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912

Allocating group tables: 完成
正在写入inode表: 完成
Creating journal (16384 blocks): 完成
Writing superblocks and filesystem accounting information: 完成

 108MiB 0:00:08 [12.8MiB/s] [===============================================================================================================>] 100%
del devmap : loop0p2
del devmap : loop0p1
loop deleted : /dev/loop0
I: Done
```

Figure 4-5.Make Image

After production is complete, a compression package myir-image-burn-mys-zu5ev.img .gz is generated in the same directory, which is a mirror package for SD card burning.

➢ Make an SD burn card

Copy myir-image-burn-mys-zu5ev.img .gz under the windows system and unzip it out.

Use Win32DiskImager.exe burn myir-image-burn-mys-zu5ev.img package to tf card. The method is the same as the "Write image file to Micro SD Card" method in the "4.1 Making SD Card Launcher" section.

➢ SD burn card burning system

The made SD burn card is inserted into the SD card slot of the development board, and then the board's start-up mode switch SW1 1 dial to OFF, 2 dial to ON, 3 dial to OFF, 4 dial to ON, set to TF card start mode, start the system. Plug in the power, automatically start the writing system in the SD Card, you can use the debug serial port to view the update status, or view the led light next to the reset button, the burning process will flash slowly, after the burning led light will flash quickly.

When the burn is complete, the board's starting mode switch SW1 1 dial to ON, 2 dial to OFF, 3 dial to ON, 4 dial to ON, set to Qspi flash start-up mode, power up again, you can start the board from qspiflash mode, into the burned rootfs file system.

5. How to modify the BSP

The previous sections have described more fully the complete process of building a system image running on the MYS-ZU5EV board based on the Petalinux project and burning the image to the board. Because many of the pins of the MYS-ZU5EV core board have multiple functional configurations, there are always some differences in the actual project. In addition to hardware differences, there are some differences in software systems, may need to compare complete graphics systems, QT libraries, etc. , focusing on background management applications, may need more complete network applications. This requires system developers to do some tailoring and porting based on the code we provide. This chapter describes the process of developing and customizing your own system from a system developer's perspective, laying the groundwork for adapting your own hardware later.

5.1. Petalinux bsp introduction

Petalinux bsp contains a variety of metadata and recipes for BSPs, middleware, or applications. Based on this "layer model", users can adapt hardware based on MYS-ZU5EV core board design and customize their applications to build their own system images, which are included in petalinux bsp as follows :

```
mys_zu5ev$ tree -L 3
├── build
|   ├── bitbake-cookerdaemon.log
|   ├── bootgen.bif
|   ├── build.log
|   ├── build.log.old
|   ├── cache
|   |   ├── bb_codeparser.dat
|   |   ├── bb_persist_data.sqlite3
|   |   ├── bb_unihashes.dat
|   |   └── local_file_checksum_cache.dat
|   ├── conf
|   |   ├── bblayers.conf
|   |   ├── devtool.conf
|   |   ├── local.conf
|   |   ├── locked-sigs.inc
|   |   ├── plnxtool.conf
|   |   ├── sanity_info
|   |   ├── sdk-conf-manifest
|   |   ├── site.conf
|   |   ├── templateconf.cfg
|   |   └── unlocked-sigs.inc
|   ├── config.log
|   ├── downloads
|   ├── misc
|   |   ├── config
|   |   └── rootfs_config
|   ├── sstate-cache
|   └── tmp
|       ├── abi_version
|       ├── buildstats
|       ├── cache
|       ├── deploy
|       ├── hosttools
|       ├── log
|       ├── pkgdata
|       ├── saved_tmpdir
|       ├── sstate-control
|       ├── stamps
|       ├── sysroots
|       ├── sysroots-components
|       ├── sysroots-uninative
|       ├── work
|       └── work-shared
├── components
|   ├── plnx_workspace
|    |    └── device-tree
|    └── yocto
|        ├── cache
|        ├── conf
|        ├── downloads
|        ├── environment-setup-aarch64-xilinx-linux
|        ├── layers
|        ├── site-config-aarch64-xilinx-linux
|        ├── sysroots
|        ├── version-aarch64-xilinx-linux
|        └── workspace
├── config.project
├── images
|    └── linux
|        ├── bl31.bin
|        ├── bl31.elf
|        ├── BOOT.BIN
|        ├── boot.scr
|        ├── Image
|        ├── image.ub
|        ├── pmufw.elf
|        ├── pxelinux.cfg
|        ├── rootfs.cpio
|        ├── rootfs.cpio.gz
|        ├── rootfs.cpio.gz.u-boot
|        ├── rootfs.jffs2
|        ├── rootfs.manifest
|        ├── rootfs.tar.gz
|        ├── system.bit
|        ├── system.dtb
|        ├── u-boot.bin
|        ├── u-boot.elf
|        ├── vmlinux
```

```
│       ├── zynqmp_fsbl.elf
│       ├── zynqmp-qemu-arm.dtb
│       ├── zynqmp-qemu-multiarch-arm.dtb
│       └── zynqmp-qemu-multiarch-pmu.dtb
└── project-spec
    ├── attributes
    ├── configs
    │   ├── busybox
    │   ├── config
    │   ├── init-ifupdown
    │   ├── rootfs_config
    │   └── rootfs_config.old
    ├── hw-description
    │   ├── design_1_wrapper.bit
    │   ├── metadata
    │   ├── psu_init.c
    │   ├── psu_init_gpl.c
    │   ├── psu_init_gpl.h
    │   ├── psu_init.h
    │   ├── psu_init.html
    │   ├── psu_init.tcl
    │   └── system.xsa
    └── meta-user
        ├── conf
        ├── COPYING.MIT
        ├── README
        ├── recipes-apps
        ├── recipes-bsp
        └── recipes-kernel
```

Table 5-1. meta-user layer content description

Source code and data	Decription
conf	Includes board software configuration resource information
recipes-app	The included application
recipes-bsp	Includes configuration resources such as uboot and devicetree
recipes-kernel	Resource that contains the linux kernel

When performing system porting, the focus is on the recipes-bsp section, which is responsible for hardware initialization and system booting, the recipes-kernel section of the Linux system and the drive implementation, and the recipes-app section of the application customization.

5.2. Board-level support package introduction

A board-level support package (BSP) is a collection of information that defines how to support a specific hardware device, device set, or hardware platform. The BSP includes information about the hardware characteristics and kernel configuration information on the device, as well as any other hardware drivers required.

Usually, depending on the stage of hardware startup, we divide BSP into Bootloader and Kernel sections, and the hardware BSP code designed with MYS-ZU5EV core board can view the contents of the recipes-bsp and recipes-kernel recipes in meta-user.

Recipes-bsp contains only u-boot and device-tree, which are primarily implemented for core hardware such as DDR, initialization of The Clock, and kernel booting. Based on MYS-ZU5EV core board hardware to modify this part of the content.

```
recipes-bsp
├── device-tree
└── u-boot
```

Recipes-kernel contains the Linux kernel, which is primarily implemented.

```
recipes-kernel/
└── linux
```

When designing products using MYIR's core board, the bootloader section does not have to be modified without special requirements. You need to pay more attention to kernel-driven development, as well as application design. Kernel development and application development will be described in more detail in subsequent sections.

5.3. On-board u-boot compilation and update

This U-boot is a very versatile open source boot program, including kernel boot, download updates and many other aspects, widely used in the embedded field, you can view the official website for more information: http://www.denx.de/wiki/U-Boot/WebHome

5.3.1. Compile u-boot under the petalinux project

When the user modifies the U-boot code, they can also use petalinux to build the entire image. The reference example is as follows.

```
# git add .
# git commit -m "demo"
# git format-patch -1

```

Once the modification is complete, a 0001-demo.patch is generated, copied to the project-spec/meta-user/recipes-bsp/u-boot/files/ directory of petalinux, and then the newly modified code can be compiled by adding the following to the project-spec/meta-user/recipes-bsp/u-boot/u-boot-xlnx_%.bbappend.

```
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

SRC_URI += "file://platform-top.h"
SRC_URI += "file://0001-add-config.patch"
SRC_URI += "file://0002-modify-config.patch"
SRC_URI += "file://0003-reset-ushhub.patch"
SRC_URI += "file://0001-demo.patch"

do_configure_append () {
	if [ "${U_BOOT_AUTO_CONFIG}" = "1" ]; then
		install ${WORKDIR}/platform-auto.h ${S}/include/configs/
		install ${WORKDIR}/platform-top.h ${S}/include/configs/
	fi
}

do_configure_append_microblaze () {
	if [ "${U_BOOT_AUTO_CONFIG}" = "1" ]; then
		install -d ${B}/source/board/xilinx/microblaze-generic/
		install ${WORKDIR}/config.mk ${B}/source/board/xilinx/microblaze-gen
eric/
	fi
}
```

You can build u-boot as follows.

```
# petalinux-build -c u-boot -x distclean
# petalinux-build -c u-boot
```

5.3.2. How to update the Boot.bin separately

1) Generate boot.bin

Before updating the boot .bin, we need execute the following command to generate boot.bin：

```
# petalinux-package --boot --fsbl images/linux/zynqmp_fsbl.elf --u-boot=images
/linux/u-boot.elf --pmufw --atf --fpga images/linux/system.bit  --force
```

2) Update the boot.bin separately

The core board is equipped with a free space of 32M qspiflash, where the default flash first partition is the boot.bin partition “/dev/mtd0”.

After copying images/linux/boot.bin to the board over the network or USB stick or SD card, here we copy the boot .bin to the user's home directory and burn the boot .bin to qspiflash separately by following the following command.

```
# flashcp -v boot.bin /dev/mtd0
```

5.4. On-board Kernel compilation and update

This Linux kernel is a very large open source kernel, is used in a variety of distribution operating systems, Linux kernel with its portability, a variety of network protocol support, independent module mechanism, MMU and many other rich features, so that Linux kernel can be widely used in embedded systems. The linux version used by MYS-ZU5EV is Linux kernel 5.4.0.

5.4.1. Compile Kernel under the Petalinux project

When the user modifies kernel's code, they can also use petalinux to build the entire image. The reference example is as follows.

```
# git add .
# git commit -m "demo"
# git format-patch -1
```

Once the modification is complete, a 0001-demo.patch is generated, which is copied to the petalinux project-spec/meta-user/recipes-kernel/linux/linux-xlnx directory, and then the newly modified code can be compiled by adding the following to the file of the project-spec/meta-user/recipes-kernel/linux/linux-xlnx_%.bbappend.

```
SRC_URI += "file://bsp.cfg"
SRC_URI += "file://0001-add-watchdog.patch"
SRC_URI += "file://0002-adv7619-driver.patch"
SRC_URI += "file://0003-mplane.patch"
SRC_URI += "file://0001-demo.patch"

KERNEL_FEATURES_append = " bsp.cfg"
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
```

Once the modification is complete, kernel can be built as follows.

```
# petalinux-build -c kernel distclean
# petalinux-build -c kernel
```

5.4.2. How to update Kernel separately

Once the user has compiled successfully, the images/linux/image.ub file can be updated by transferring the transfer media such as Ethernet, USB drives, etc. to the board to execute the following commands.

```
# flashcp -v image.ub /dev/mtd1
```

5.5. Under the petalinux project to build the BSP software of the new FPGA project

In the MYS_ZU5EV development board developed a new FPGA function, in order to make the fpga function to use properly, the corresponding software needs to be developed. We use petalinux to quickly build projects and build a corresponding set of software. The steps are as follows:

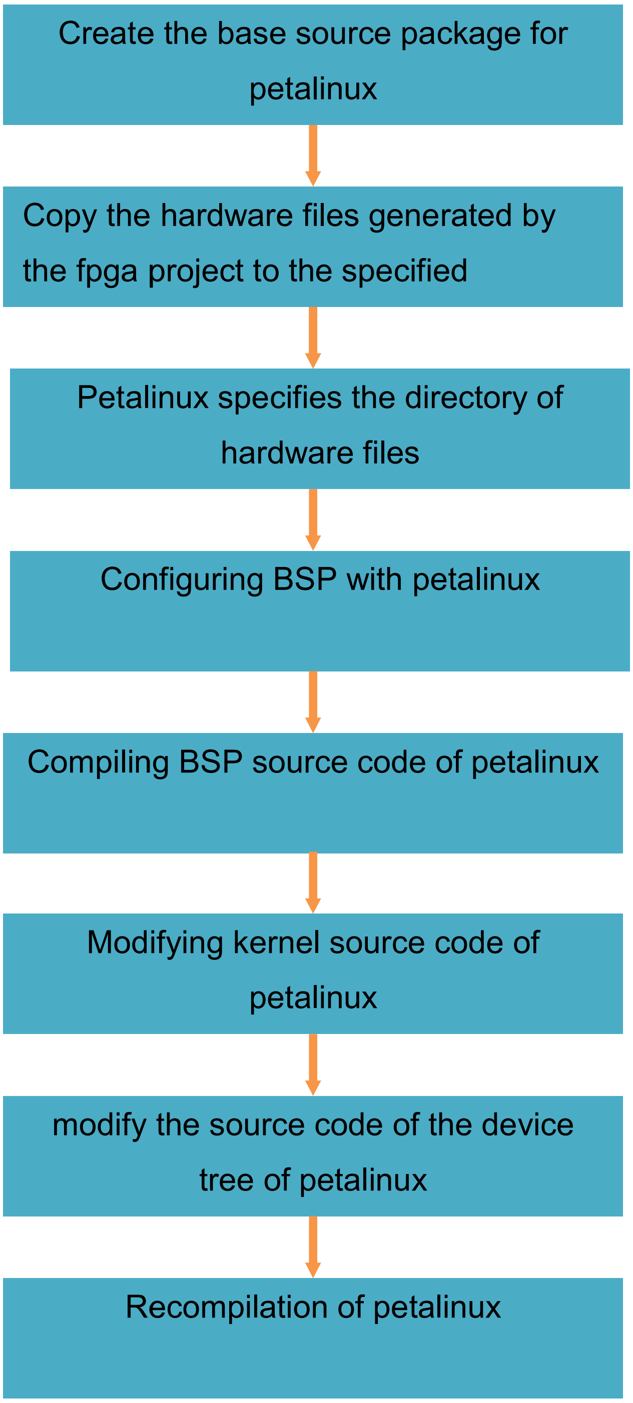

Create the base source package for petalinux
Copy the hardware files generated by the fpga project to the specified
Petalinux specifies the directory of hardware files
Configuring BSP with petalinux
Compiling BSP source code of petalinux
Modifying kernel source code of petalinux
modify the source code of the device tree of petalinux
Recompilation of petalinux

Figure 5-1 Petalinux build project diagram

- **Create the underlying source package for petalinux**

Using bsp on the disc as the base source package, here is an example of 04-Source/Petalinux/mys_zu5ev2020_4G_core.bsp.

```
# petalinux-create -t project -s mys_zu5ev2020_4G_core.bsp
# cd mys_zu5ev
```

- **Copy the hardware files generated by the fpga project to the specified directory**

Once you've created a source package for mys_zu5ev2020_4G_core.bsp, the rest is done on that basis. Copy the hardware profile generated by the fpga project design_1_wrapper.xsa to the specified directory, here in the case of the /home/work/zu5ev directory.

```
# ls /home/work/zu5ev/design_1_wrapper.xsa
/home/ work/zu5ev /design_1_wrapper.xsa
```

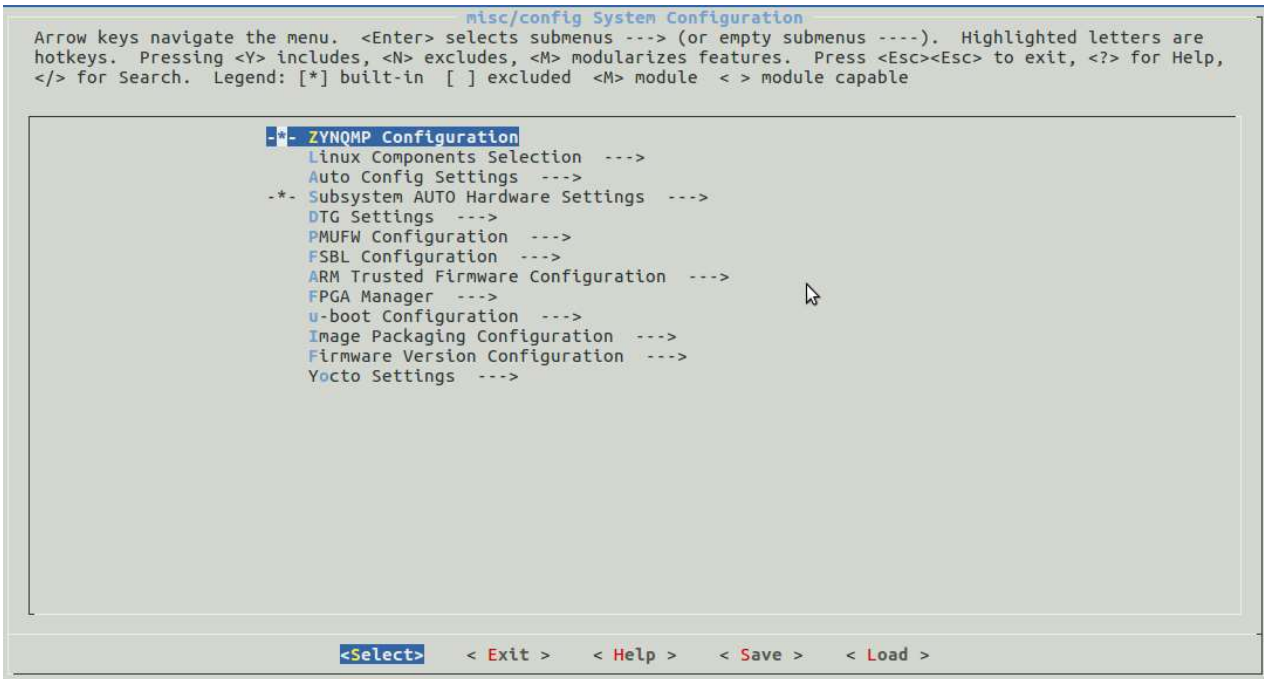
- **Petalinux specifies the directory of hardware files**

This Petalinux specifies the directory of the hardware profile, and petalinux compiles the petalinux project by building the petalinux project with the hardware profile design_1_wrapper.xsa in the directory of the hardware profile. Here's what to do :

```
# petalinux-config --get-hw-description=/home/work/zu5ev
```

- **Configuring BSP with petalinux**

Once the directory of the hardware profile is specified, the petalinux-config directory above automatically enters the petalinux configuration interface. In this interface, you can configure it as needed.



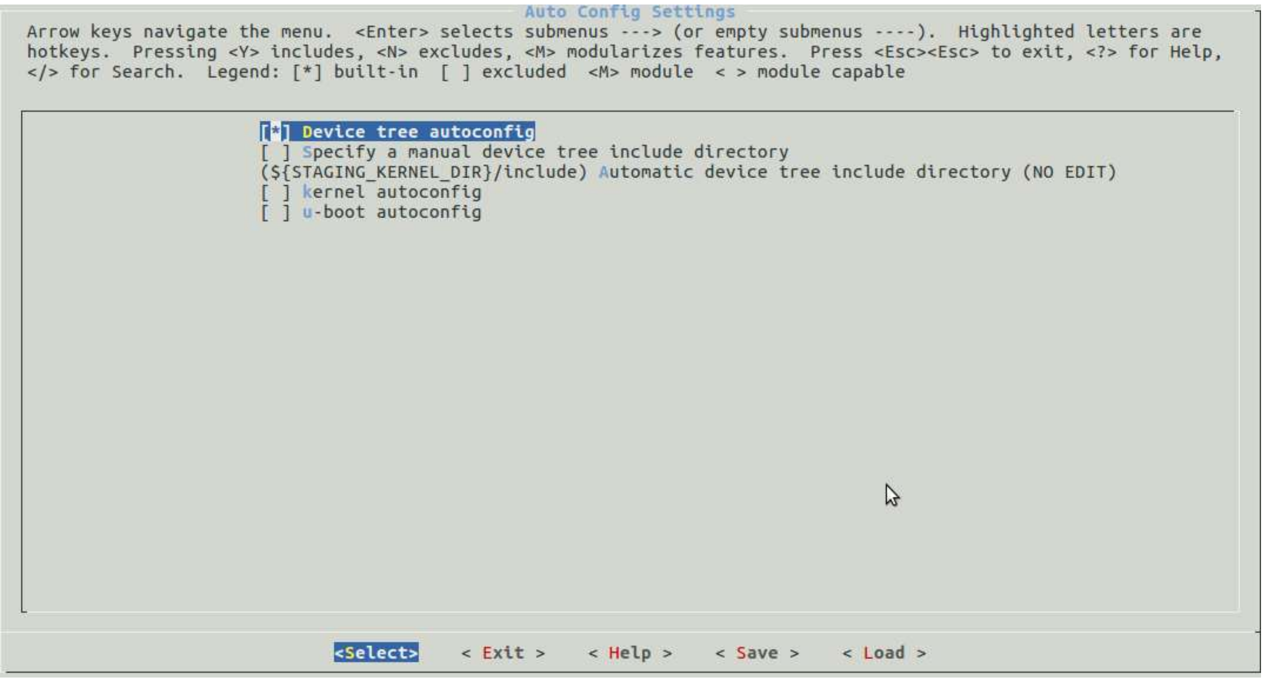
```
misc/config System Configuration
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are
hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help,
</> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

                    -*- ZYNQMP Configuration
                        Linux Components Selection  --->
                        Auto Config Settings  --->
                    -*- Subsystem AUTO Hardware Settings  --->
                        DTG Settings  --->
                        PMUFW Configuration  --->
                        FSBL Configuration  --->
                        ARM Trusted Firmware Configuration  --->
                        FPGA Manager  --->
                        u-boot Configuration  --->
                        Image Packaging Configuration  --->
                        Firmware Version Configuration  --->
                        Yocto Settings  --->

            <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 5-2 Configuration interface

Here are a few important configurations for reference.

（1）The device tree is configured as petalinux automatically, kernel and uboot are configured as non-automatic, which we can configure manually. Here are the options:

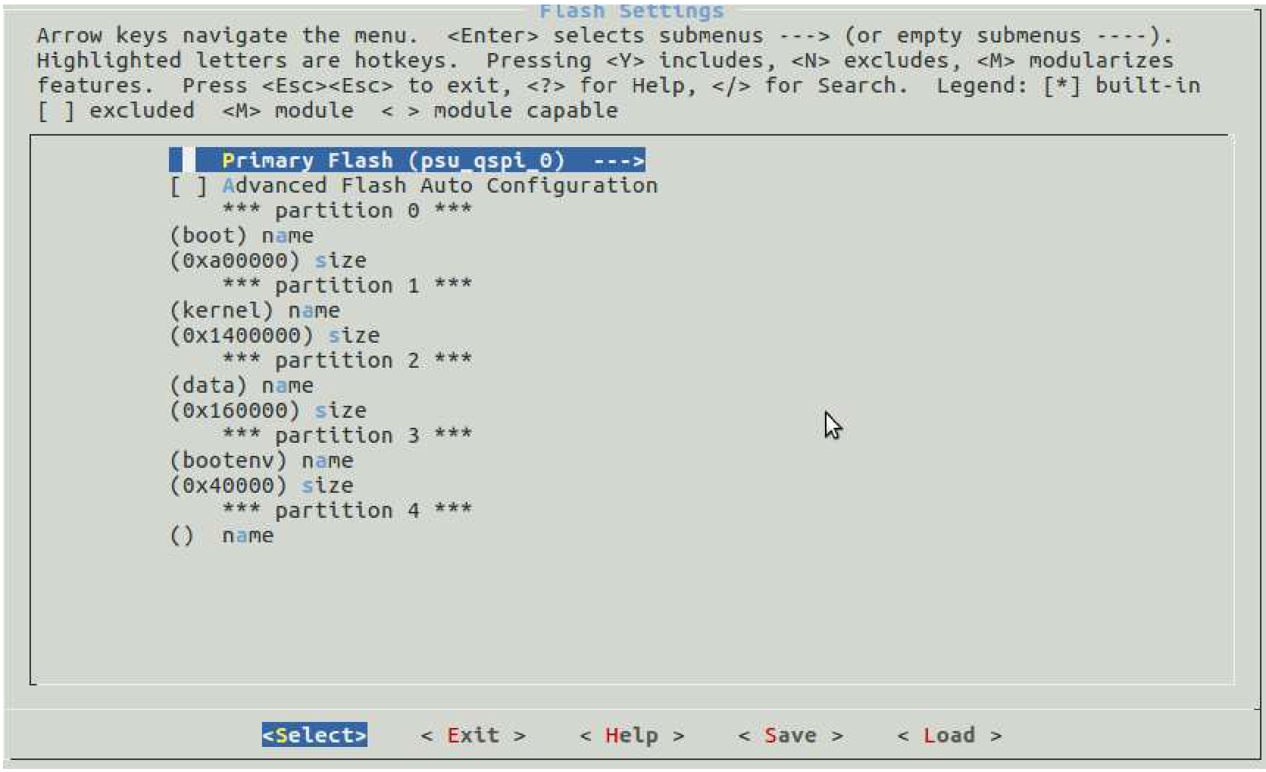
```
[*] Device tree autoconfig
[ ] kernel autoconfig
[ ] u-boot autoconfig
```



```
                          Auto Config Settings
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are
 hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help,
 </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

                    [*] Device tree autoconfig
                    [ ] Specify a manual device tree include directory
                    (${STAGING_KERNEL_DIR}/include) Automatic device tree include directory (NO EDIT)
                    [ ] kernel autoconfig
                    [ ] u-boot autoconfig

                   <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 5-3 Automatic configuration selection

(2)qspi flash configuration

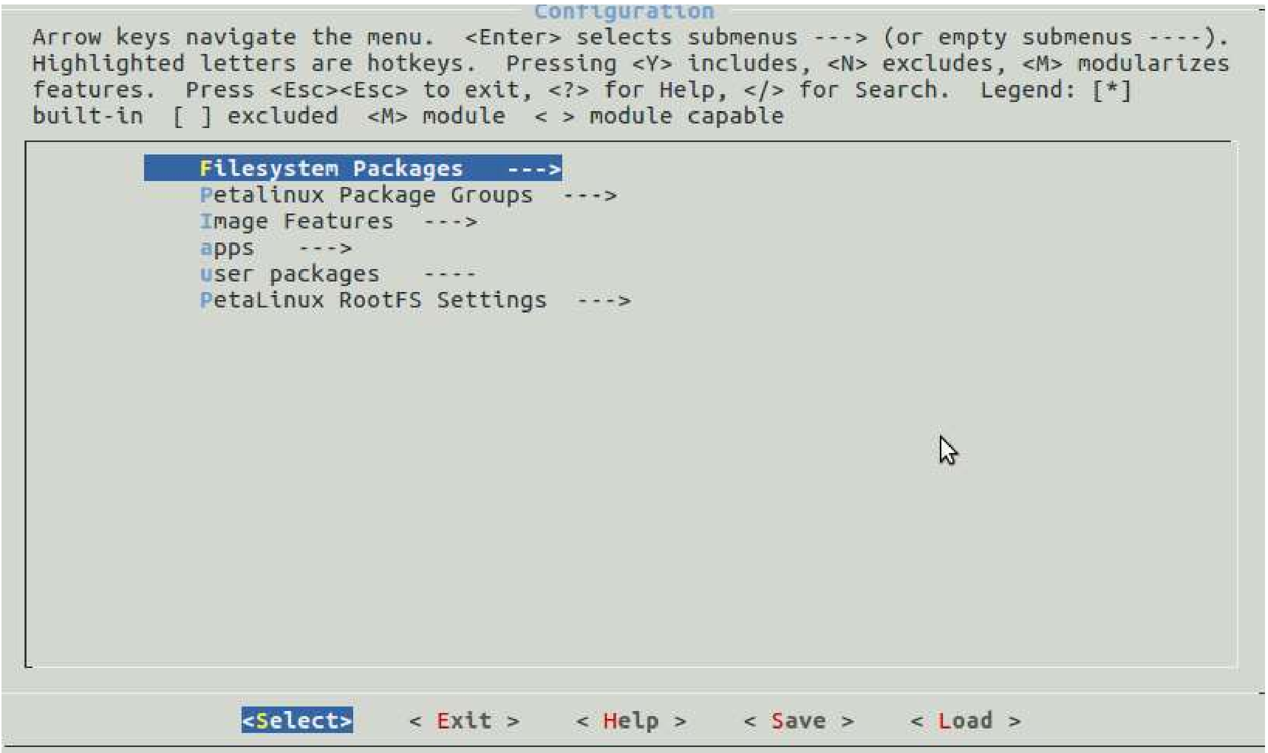


```
                            Flash Settings
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
 features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
 [ ] excluded  <M> module  < > module capable

          Primary Flash (psu_qspi_0)  --->
      [ ] Advanced Flash Auto Configuration
          *** partition 0 ***
      (boot) name
      (0xa00000) size
          *** partition 1 ***
      (kernel) name
      (0x1400000) size
          *** partition 2 ***
      (data) name
      (0x160000) size
          *** partition 3 ***
      (bootenv) name
      (0x40000) size
          *** partition 4 ***
      ()  name

          <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 5-4 qspi configuration

If you need a profile system, you can enter the following command configuration :

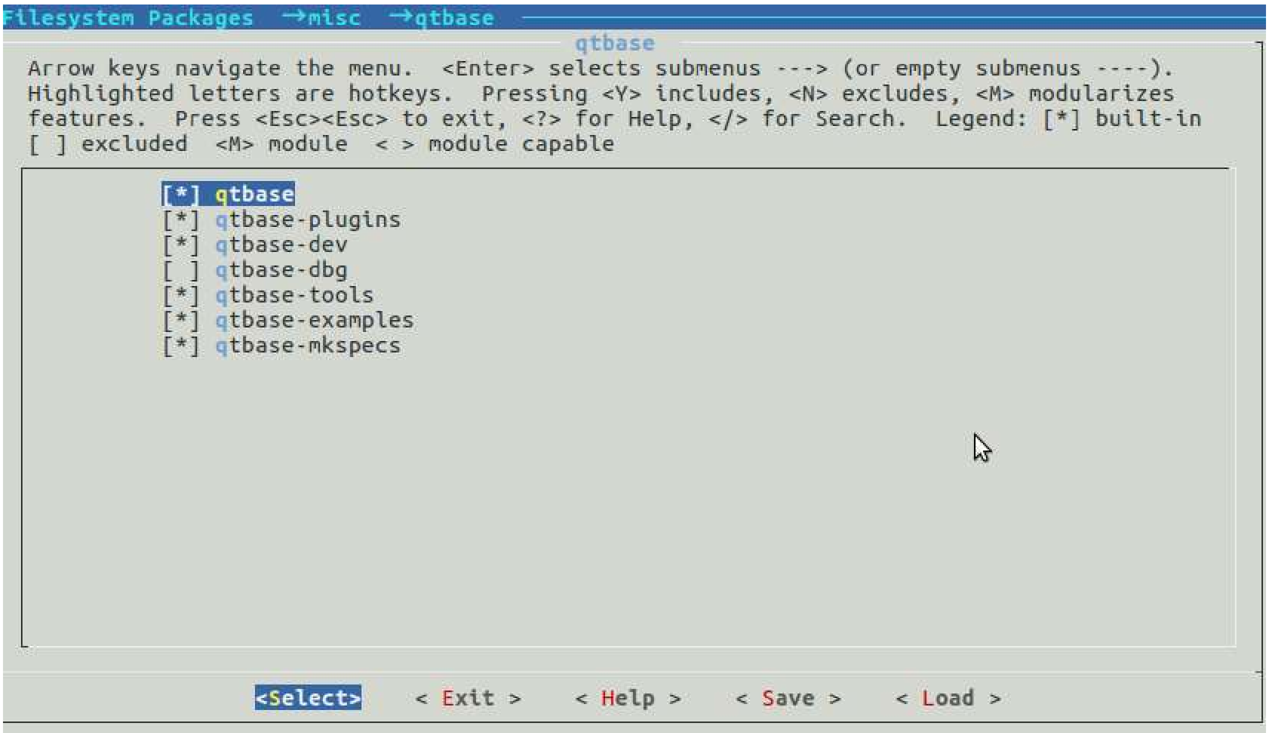
```
#petalinux-config –c rootfs
```



```
Configuration
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*]
built-in  [ ] excluded  <M> module  < > module capable

        Filesystem Packages  --->
        Petalinux Package Groups  --->
        Image Features  --->
        apps  --->
        user packages  ----
        PetaLinux RootFS Settings  --->

      <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 5-5 Configure rootfs

For example, if you need to configure the qt function, you should choose the following configuration :



```
Filesystem Packages  →misc  →qtbase
qtbase
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
[ ] excluded  <M> module  < > module capable

        [*] qtbase
        [*] qtbase-plugins
        [*] qtbase-dev
        [ ] qtbase-dbg
        [*] qtbase-tools
        [*] qtbase-examples
        [*] qtbase-mkspecs

      <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 5-6 Configure qt

Once configured, you can build the petalinux project, which is where petalinux automatically downloads and compiles the petalinux source code.

- **Compiling BSP source code of petalinux**

```
#petalinux-build
```

Once compiled, the kernel source code and the device tree source code are modified as needed, and after adding or modifying the code, the petalinux source code can be recompiled to produce the corresponding petalinux image file.

- **Modifying kernel source code of petalinux**

Modify the kernel source code in petalinux and first enter the linux source directory build/tmp/work/zynqmp_generic-xilinx-linux/linux-xlnx/5.4+gitAUTOINC+22b71b4162-r0/linux-zynqmp_generic-standard-build/ , modify the linux source code. Then follow the steps below to produce a patch file 0001-demo.patch。

```
# git add .
# git commit -m "demo"
# git format-patch -1
```

Add the patch file to the project-spec/meta-user/recipes-kernel/linux/linux-xlnx directory. Then you need to add the following to the project-spec/meta-user/recipes-kernel/linux/linux-xlnx_%.bbappend file, and then recompile kernel to compile the newly modified code.

```
SRC_URI += "file://bsp.cfg"
SRC_URI += "file://0001-add-watchdog.patch"
SRC_URI += "file://0002-adv7619-driver.patch"
SRC_URI += "file://0003-mplane.patch"
SRC_URI += "file://0001-demo.patch"

KERNEL_FEATURES_append = " bsp.cfg"
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
```

- **Modifying kernel source code of petalinux**

Add the required device tree source code in a way similar to the following.

```
/include/ "system-conf.dtsi"

#include <dt-bindings/media/xilinx-vip.h>

/ {
        chosen {
bootargs = "console=ttyPS0,115200 earlyprintk root=/dev/mmcblk1p2 rwrootwai
tclk_ignore_unused";
stdout-path = "serial0:115200n8";
        };

leds {
                compatible = "gpio-leds";
                led1 {
                        label = "rs485_de";
gpios = <&gpio 12 0>;
linux,default-trigger = "gpio";
                };
                led2 {
                        label = "wdt_en";
gpios = <&gpio 33 0>;
linux,default-trigger = "gpio";
                };
                led3 {
                        label = "led_sys";
gpios = <&gpio 43 0>;
linux,default-trigger = "gpio";
                };
        };
        ......
};

&gem3 {
phy-handle = <&phy0>;
phy-mode = "rgmii-id";
        phy0: phy@21 {
                reg = <4>;
        };
```

```
};
……
```

At this point, the entire petalinux build is complete, and the recompiled can generate a mirror file for the petalinux project.

6. How to fit your hardware platform

To adapt to the user's new hardware platform, you first need to understand what resources are available from MYIR's MYS-ZU5EV board, and you can view the “MYS-ZU5EV_SDK Release Notes” . In addition, the user also needs to CPU chip manual, as well as MYS-ZU5EV core board product manual, pin definition has a more detailed understanding, in order to facilitate the actual function of these pins for proper configuration and use.

6.1. How to create your device tree

6.1.1. Onboard device tree

Users can create their own device tree in BSP source, and only need to make appropriate adjustments to the Linux kernel device tree based on actual hardware resources. The list of device trees in the BSP sections of MYS-ZU5EV is listed here for user development reference, as shown in the table below :

Table 6-1 Description of the device tree file

PATH	DEVICE TREE	DESCRIPTION
components/plnx_workspace/device-tree/device-tree/	zynqmp.dtsi	The basic configuration of the zynqmp family of devices is automatically generated by petalinux and should not be modified manually.
	pcw.dtsi	The underlying configuration that contains the PS side is automatically generated by petalinux and should not be modified manually.
	system-conf.dtsi	peripheral configuration, automatically generated by petalinux, do not manually modify.
	system-top.dts	Top-level configuration, automatically generated by

		petalinux, do not modify it manually.
	pl.dtsi	Includes PL-side configuration, automatically generated by petalinux, do not manually modify.
project-spec/meta-user/recipes-bsp/device-tree/files	system-user.dtsi	The configuration of the peripheral resources of the development board can be modified according to the actual hardware situation.
	pl-custom.dtsi	Can be changed according to the actual PL side design.

6.1.2. The addition of the device tree

This Linux kernel device tree is a data structure that describes device information in a unique syntax format. Passed to kernel by BootLoader, kernel parses and forms a dev structure associated with the driver for use by the driver code.
If users want to add a new device tree to their hardware, they just need to project-spec/meta-user/recipes-bsp/device-tree/files directory creates a new dtsi file and then includes it in the system-user.dtsi. The following example is to create a new device tree and add a node that controls the led through mio.

- **Modify device tree**

```
# touch user-define.dtsi
```

Then included in the system-user.dtsi :

```
/include/ "system-conf.dtsi"
/include/ "user-define.dtsi"

/ {
    leds {
            compatible = "gpio-leds";
            led1 {
```

```
                        label = "rs485_de";
                        gpios = <&gpio 12 0>;
                        linux,default-trigger = "gpio";
                };
                led2 {
                        label = "wdt_en";
                        gpios = <&gpio 33 0>;
                        linux,default-trigger = "gpio";
                };
    };
    ........
};
```

- **New led nodes**

Edit the device tree user-define.dtsi to add led nodes, as shown below :

```
/ {
        gpio-leds {
            compatible =  “gpio-leds” ;
            led3 {
                  label = "led_sys";
                  gpios = <&gpio 43 0>;
                  linux,default-trigger = "gpio";
            };
        };
};
```

- **Add a new device tree to the recipe**

Edit “project-spec/meta-user/recipes-bsp/device-tree/device-tree.bbappend” to look like this:

```
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

SRC_URI += "file://system-user.dtsi"
SRC_URI += "file://user-define.dtsi"

python () {
    if d.getVar("CONFIG_DISABLE"):
        d.setVarFlag("do_configure", "noexec", "1")
}

export PETALINUX
do_configure_append () {
        script="${PETALINUX}/etc/hsm/scripts/petalinux_hsm_bridge.tcl"
        data=${PETALINUX}/etc/hsm/data/
        eval xsct -sdx -nodisp ${script} -c ${WORKDIR}/config \
        -hdf ${DT_FILES_PATH}/hardware_description.${HDF_EXT} -repo ${S} \
        -data ${data} -sw ${DT_FILES_PATH} -o ${DT_FILES_PATH} -a "soc_mapping"
}
```

6.2. How to configure CPU function pins based on your hardware

Implementing the control of a function pin is one of the more complex system development processes, which contains the driving development, application implementation and so on, this section does not analyze the development process of each part specifically, but uses examples to explain the control implementation of the functional pin.

6.2.1. The method of the GPIO pin configuration

GPIO: General-purpose input/output，in embedded devices, is a very important resource through which high and low levels can be output or read into the pin state through them - high or low.

GPIO configuration method

The configuration of this GPIO can be found in the description file (01-Document\Datasheet\CPU\ug1085-zynq-ultrascale-trm.pdf) and the schematics of the MYS_ZU5EV board (01-Document\HardwareFiles\Schematic\MYS-ZU5EV-32E4D-EDGE_V11.pdf), as follows:
GPIO is divided into 78 MIOs and 96 EMIO, specifically refer to the "General PurposeI/O" section of the CD-ROM 01-Document\Datasheet\CPU\ug1085-zynq-ultrascale-trm.pdf document with detailed instructions.

- **General calculation method**

MIO GPIO number=MIO port number，MIO0 GPIO number is GPIO0. EMIO GPIO number = EMIO port number + 78，EMIO0 GPIO number is GPIO78.

6.2.2. GPIO is defined in the device tree

1) Configure the function pin as an instance of the GPIO function

This instance uses PS_MIO43 as a test GPIO. Describes how to configure GPIO in the device tree and provides kernel-driven use for later chapters.
Simply add nodes in device tree project-spec/meta-user/recipes-bsp/device-tree/file/system-user.dtsi.

```
……
      leds {
```

```
                compatible = "gpio-leds";
                led1 {
                        label = "rs485_de";
                        gpios = <&gpio 12 0>;
                        linux,default-trigger = "gpio";
                };
                led2 {
                        label = "wdt_en";
                        gpios = <&gpio 33 0>;
                        linux,default-trigger = "gpio";
                };
                led3 {
                        label = "led_sys";
                        gpios = <&gpio 43 0>;
                        linux,default-trigger = "gpio";
                };
        };
......
```

6.3. How to use your own configured pins

The pins we have configured in Kernel's equipment tree can be used in Kernel for control of the pins.

6.3.1. User space uses GPIO pins

The architecture of Linux operating system is divided into user and kernel states (or user space and kernel). User state is the active space of the upper-level application, and the execution of the application must rely on the resources provided by the kernel, including CPU resources, storage resources, I/O resources, and so on. In order for an upper-level app to have access to these resources, the kernel must provide an interface for the upper-level app to access: a system call. This shell is a special application, commonly known as the command line, essentially a command interpreter, which is called down the system, up and down a variety of applications. With Shell scripts, a very large functionality can usually be implemented with a few lines of Shell scripts, because these Shell statements are usually encapsulated in a layer of system calls. For user and system interaction. This section describes three basic ways to control how to use GPIO pins in the user state.

- Shell command
- System call
- Library function

1) Pin control with shell

Pin control with shell is essentially implemented by calling the file operation interface provided by Linux, and this section does not provide detailed instructions to view the description in Section 3.1 of the “MYS-ZU5EV_Linux Evaluation Guide” .

2) Library functions implement pin control

Starting with Linux 4.8, Linux introduced a new gpio operating method, GPIO character device. Based on the "file descriptor" character device, each GPIO group has a corresponding gpiochip file under "/dev", such as "/dev/gpiochip0 for GPIOA, /dev/gpiochip1 for GPIOB" and so on.

The Libgpiod library function is implemented because of the gpiochip approach, based on the C language, so the developer implemented Libgpiod, providing some tools and a simpler C API interface. Libgpiod provides a complete API for developers, as well as some apps in user space to operate GPIO.

Libgpiod basic interface descriptions :

- gpiodetect - List all gpiochip that appear in the system, their names, labels, and the number of GPIO rows.
- gpioinfo - Lists all the rows of the specified gpiochips, their names, consumers, directions, activity status, and additional flags.
- gpioget - Read the specified GPIO row value.
- gpioset - Set the specified GPIO row values, potentially keeping them exported and waiting for timeouts, user input, or signals.
- gpiofind - Look for the gpiochip name and row offset for a given row name.
- gpiomon - Wait for the events on the GPIO line, specify which events to observe, how many events to handle before exiting, or whether the events should be reported to the console.

For more descriptions, check out the libgpiod source code https://git.kernel.org/pub/scm/libs/libgpiod/libgpiod.git/.

The following will be MIO43 as the operation of GPIO pins to achieve the C language code control examples (alternately high and low).

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

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

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

	/* Open device: gpiochip0 for mio43*/
	fd = open(chrdev_name, 0);
	if (fd == -1) {
		ret = -errno;
		fprintf(stderr, "Failed to open %s\n", chrdev_name);

	return ret;
	}

	/* request GPIO line: mio43 */
	req.lineoffsets[0] = 43;
	req.flags = GPIOHANDLE_REQUEST_OUTPUT;
	memcpy(req.default_values, &data, sizeof(req.default_values));
	strcpy(req.consumer_label, "gpio_43");
	req.lines = 1;

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

	/* Start GPIO ctr*/
	while(1) {
```

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

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

Copy the above code under a gpioex.c file and load the SDK environment variable to the current shell:

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
```

Use the compilation command $CC to generate an executable gpioex.

```
#$CC -o gpioex gpioex.c
```

Mask block led3 in device tree project-spec/meta-user/recipes-bsp/device-tree/file/system-user.dtsi.

```
……
        leds {
                compatible = "gpio-leds";
                led1 {
                        label = "rs485_de";
                        gpios = <&gpio 12 0>;
                        linux,default-trigger = "gpio";
                };
                led2 {
                        label = "wdt_en";
                        gpios = <&gpio 33 0>;
                        linux,default-trigger = "gpio";
                };
                /*
                led3 {
                        label = "led_sys";
                        gpios = <&gpio 43 0>;
                        linux,default-trigger = "gpio";
                };
                */
        };
......
```

Compile to generate image.ub, replace the image.ub in the tf card, tf card start-up mode to start the system.

Copy executables via the network, u disk or tf card to the board's directory, you can enter commands under the terminal to run directly, and you can see the led light flashing next to the reset button.

```
# ./gpioex
```

3) The system calls for pin control

A set of "special" interfaces that the operating system provides to user programs to call. User programs can obtain services from the operating system kernel through this set of "special" interfaces, such as users opening files, closing files, or reading and writing files through file system-related calls, and clock-related system calls to obtain system time or set timers.

When MIO43 functions as a gpio-leds, it can be controlled by system calls via the pins controlled by the driver.

```
/******************************************************************************
 * Copyright (c) 2014-2017 MYIR Tech Ltd.
 *         File: led-test.c
 *         Date: 2014/11/3
 *       Author: Kevin Su
 * Description: A demo program to show how to control leds from user-space.
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <signal.h>
#include <linux/input.h>

#define DEBUG  1

#define ERR_MSG(fmt, args...)      fprintf(stderr, fmt, ##args)
#ifdef DEBUG
	#define DBG_MSG(fmt, args...)  fprintf(stdout, fmt, ##args)
#else
	#define DBG_MSG(fmt, args...)
#endif

#define LED_DIR          "/sys/class/leds/led_sys/"
#define NAME_MAX_LENGTH              64
#define LED_DELAY_US        (100*1000)

#define ARRAY_SIZE(x)       (sizeof(x)/sizeof(x[0]))

#define BITS_MASK(num)   ((1<<num) - 1)

typedef struct led_ctrl_s {
	char name[NAME_MAX_LENGTH];
	char brightness[NAME_MAX_LENGTH];
	char trigger[NAME_MAX_LENGTH];
	char trigger_backup[NAME_MAX_LENGTH];
```

```
        int state;
        int initialized;
} led_ctrl_t;

static led_ctrl_tleds[] = {
        /* name, brightness, trigger, trigger_str,  state, initialized */
        {"led_sys", LED_DIR "brightness", LED_DIR "trigger", "", 0, 0},
};

static int led_set_trigger(led_ctrl_t *led, const char *trigger)
{
        int ret;
        int fd = open(led->trigger, O_WRONLY);

        if (fd< 0) {
             ERR_MSG("Open %s failed!\n", led->trigger);
             return -1;
        }

        ret = write(fd, trigger, strlen(trigger));
        if (ret != strlen(trigger)) {
             ERR_MSG("Write %s failed!\n", led->trigger);
             close(fd);
             return -1;
        }
        close(fd);
        DBG_MSG("[%8s] Set trigger to '%s'\n",
             led->name,
             trigger);

        return 0;
}
```

```
static int led_get_trigger(led_ctrl_t *led, char *trigger)
{
        int ret;
        char tmp[1000] = {0};
        char *ptr[2] = {NULL};
        int fd = open(led->trigger, O_RDONLY);

        if (fd< 0) {
            ERR_MSG("Open %s failed!\n", led->trigger);
            return -1;
        }

        /* read back string with format like: "none cpu1 cpu2 [heartbeat] nand" */
        ret = read(fd, tmp, sizeof(tmp));
        if (ret <= 0) {
            ERR_MSG("Read %s failed!\n", led->trigger);
            close(fd);
            return -1;
        }
        close(fd);

        /* find trigger from read back string, which is inside "[]" */
        ptr[0] = strchr(tmp, '[');
        if (ptr[0]) {
            ptr[0] += 1;
            ptr[1] = strchr(ptr[0], ']');
            if (ptr[1]) {
                *ptr[1] = '\0';
            } else {
                ERR_MSG("[%s] Can not find trigger in %s!\n", led->name, tmp);
                return -1;
            }
            strcpy(trigger, ptr[0]);
```

```
    } else {
        ERR_MSG("[%s] Can not find trigger in %s!\n", led->name, tmp);
        return -1;
    }

    DBG_MSG("[%8s] Get trigger: '%s'\n",
        led->name,
        trigger);

    return 0;
}

static int led_set_brightness(led_ctrl_t * led, int brightness)
{
    int ret;
    int fd = open(led->brightness, O_WRONLY);
    char br_str[2] = {0};

    br_str[0] = '0' + brightness;

    if (fd< 0) {
        ERR_MSG("Open %s failed!\n", led->brightness);
        return -1;
    }

    ret = write(fd, br_str, sizeof(br_str));
    if (ret != sizeof(br_str)) {
        ERR_MSG("Write %s failed!\n", led->brightness);
        close(fd);
        return -1;
    }
    close(fd);
    // DBG_MSG("[%s] Set brightness to %s successfully!\n",
        // led->name,
        // br_str);

    return 0;
}

static int led_init(void)
{
    int i;
    char tmp[NAME_MAX_LENGTH];


    for (i=0; i<ARRAY_SIZE(leds); i++) {
        memset(tmp, 0, sizeof(tmp));

        /* Backup all led triggers */
        if (led_get_trigger(&leds[i], tmp)) {
            return -1;
        }
        strcpy(leds[i].trigger_backup, tmp);

        /* Set all led triggers to 'none' */
        if (led_set_trigger(&leds[i], "none")) {
            return -1;
        }

        /* Set all brightness to 0 */
        if (led_set_brightness(&leds[i], 0)) {
            return -1;
        }
        leds[i].state = 0;
        leds[i].initialized = 1;
    }
    return 0;
}

static void led_restore(void)
{
    int i;

    /* set all brightness to '0', and restore triggers */
    for (i=0; i<ARRAY_SIZE(leds); i++) {
        if (leds[i].initialized) {
              led_set_brightness(&leds[i], 0);
              led_set_trigger(&leds[i], leds[i].trigger_backup);
        }
    }
}

/* Will be called if SIGINT(Ctrl+C) and SIGTERM(simple kill) signal is received */
static void signal_callback(int num)
{
    led_restore();
    exit(num);
}

int main(int argc, const char *argv[])
{
    /* Open button device */
    if(led_init()) {
        led_restore();
        return -1;
    }

    /* Register SIGINT(Ctrl+C) and SIGTERM(simple kill) signal and signal handl
er */
```

```
        signal(SIGINT, signal_callback);
        signal(SIGTERM, signal_callback);

        for (;;) {
            if (leds[0].state == 0) { /* if already ON, do nothing */
                leds[0].state = 1;
                led_set_brightness(&leds[0], leds[0].state);
            } else { /* this led should be turn OFF */
                leds[0].state = 0;
                led_set_brightness(&leds[0], leds[0].state);
            }
            usleep(LED_DELAY_US);
        }

        led_restore();

        return 0;
}
```

Copy the above code under a gpiotest.c file and load the SDK environment variable to the current shell:

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
```

Use the compilation command $CC to generate an executable led-test.

```
# $CC -o led-test led-test.c
```

Configure led3 in device tree project-spec/meta-user/recipes-bsp/device-tree/file/system-user.dtsi.

```
......
    leds {
        compatible = "gpio-leds";
        led1 {
            label = "rs485_de";
            gpios = <&gpio 12 0>;
            linux,default-trigger = "gpio";
```

```
                };
                led2 {
                        label = "wdt_en";
                        gpios = <&gpio 33 0>;
                        linux,default-trigger = "gpio";
                };
                led3 {
                        label = "led_sys";
                        gpios = <&gpio 43 0>;
                        linux,default-trigger = "gpio";
                };
        };
......
```

Compile to generate image.ub, replace the image.ub in the tf card, tf card start-up mode to start the system.

Copy executables via network, u disks or tf card to the board's directory, and enter commands under the terminal to run directly.

```
# ./led-test
```

You can see the led light flashing next to the reset button.

7. Linux application example of FPGA PL function

Usually the PL function we implement in fpga requires software support to use the PL function properly. This chapter takes the axi-uartlite ip core as an example and describes the linux application process for FPGA PL features.

7.1. Implementation process of petalinux software in Axi uartlite

- **The steps for implementing the functionality are as follows:**

（1）Establish the basic bsp package

（2）Configure the kernel device tree

（3）Configure the kernel driver

（4）Petalinux compile

（5）Test the module functionality of the PL

- **Establish the basic bsp package**

Copy the design_1_wrapper.xsa generated by 05-ProgrammableLogic_Source/axi_uartlite.rar sample project on the CD to the directory /home/work/vivadoxsa/

```
# petalinux-create -t project -s mys_zu5ev2020_4G_ core.bsp
# petalinux-config --get-hw-description=/home/work/vivadoxsa/
```

- **Configure the kernel device tree**

Configure the petalinux project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi device tree file as follows:

```
/include/ "system-conf.dtsi"

#include <dt-bindings/media/xilinx-vip.h>

/ {
	chosen {
		bootargs = "console=ttyPS0,115200 earlyprintk root=/dev/mmcblk1p2 rw rootwait clk_ignore_unused";
		stdout-path = "serial0:115200n8";
	};
};

&sdhci0 {
	no-1-8-v;
};

&sdhci1 {
	disable-wp;
	no-1-8-v;
};

&qspi {
	flash@0 {
		compatible = "m25p80";
		#address-cells = <1>;
		#size-cells = <1>;
		reg = <0x0>;
		spi-tx-bus-width = <1>;
		spi-rx-bus-width = <4>;
		spi-max-frequency = <54000000>;
	};
};
```

Petalinux automatically generates axi uartlite's device tree profile components/plnx_workspace/device-tree/device-tree/pl.dtsi as follows:

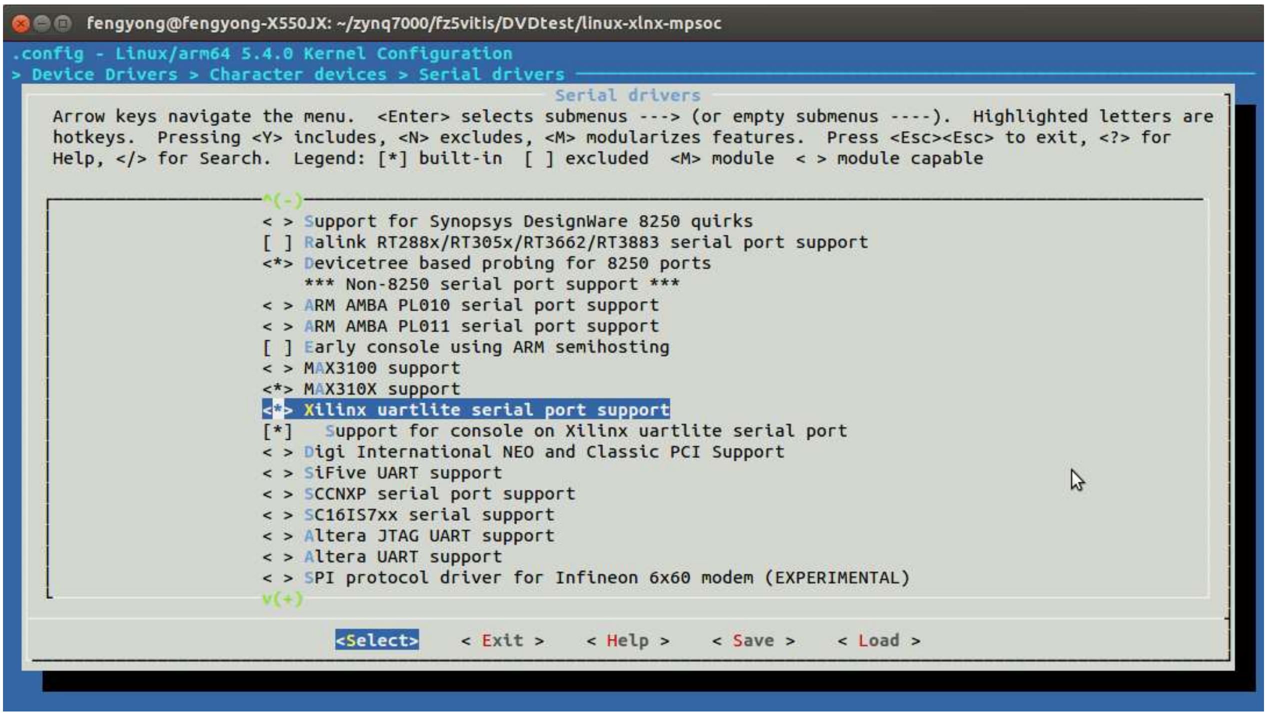
```
/*
 * CAUTION: This file is automatically generated by Xilinx.
 * Version:
 * Today is: Fri Jun 11 08:34:40 2021
 */

/ {
	amba_pl: amba_pl@0 {
		#address-cells = <2>;
		#size-cells = <2>;
		compatible = "simple-bus";
		ranges ;
		axi_uartlite_0: serial@80000000 {
			clock-names = "s_axi_aclk";
			clocks = <&zynqmp_clk 71>;
			compatible = "xlnx,axi-uartlite-2.0", "xlnx,xps-uartlite-1.00.a";
			current-speed = <115200>;
			device_type = "serial";
			interrupt-names = "interrupt";
			interrupt-parent = <&gic>;
			interrupts = <0 89 1>;
			port-number = <1>;
			reg = <0x0 0x80000000 0x0 0x10000>;
			xlnx,baudrate = <0x1c200>;
			xlnx,data-bits = <0x8>;
			xlnx,odd-parity = <0x0>;
			xlnx,s-axi-aclk-freq-hz-d = "99.999001";
			xlnx,use-parity = <0x0>;
		};
	};
};
```

The kernel device tree is configured.

- **Configure the kernel driver**

Look at the kernel device tree, which is already equipped with an axi uartlite driver.

```
# petalinux-config -c kernel
```



```
fengyong@fengyong-X550JX: ~/zynq7000/fz5vitis/DVDtest/linux-xlnx-mpsoc
.config - Linux/arm64 5.4.0 Kernel Configuration
> Device Drivers > Character devices > Serial drivers
                              Serial drivers
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are
  hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for
  Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

                      ^(-)
                      < > Support for Synopsys DesignWare 8250 quirks
                      [ ] Ralink RT288x/RT305x/RT3662/RT3883 serial port support
                      <*> Devicetree based probing for 8250 ports
                          *** Non-8250 serial port support ***
                      < > ARM AMBA PL010 serial port support
                      < > ARM AMBA PL011 serial port support
                      [ ] Early console using ARM semihosting
                      < > MAX3100 support
                      <*> MAX310X support
                      <*> Xilinx uartlite serial port support
                      [*]   Support for console on Xilinx uartlite serial port
                      < > Digi International NEO and Classic PCI Support
                      < > SiFive UART support
                      < > SCCNXP serial port support
                      < > SC16IS7xx serial support
                      < > Altera JTAG UART support
                      < > Altera UART support
                      < > SPI protocol driver for Infineon 6x60 modem (EXPERIMENTAL)
                      v(+)

                <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 8-1 kernel configuration

- **Petalinux compile**

```
# petalinux-build
```

- **Test the module functionality of the PL**

Test PL module function, in hardware, is RS232, that is, the J12 interface of the MYS_ZU5EV development board, testing can be directly J12 interface 1 foot and 3 feet short, you can be on a board RS232 test.

The tester code for Rs232 is as follows:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <errno.h>
#include <getopt.h>
#include <string.h>

#define FALSE 1
```

```
#define TRUE 0

char *recchr="We received:\"";

void print_usage();

int speed_arr[] = {
        B921600, B460800, B230400, B115200, B57600, B38400, B19200,
        B9600, B4800, B2400, B1200, B300,
};

int name_arr[] = {
        921600, 460800, 230400, 115200, 57600, 38400,  19200,
        9600,  4800,  2400,  1200,  300,
};

void set_speed(int fd, int speed)
{
        int   i;
        int   status;
        struct termiosOpt;
        tcgetattr(fd, &Opt);

        for ( i= 0;  i<sizeof(speed_arr) / sizeof(int);  i++) {
                if  (speed == name_arr[i])   {
                        tcflush(fd, TCIOFLUSH);
                        cfsetispeed(&Opt, speed_arr[i]);
                        cfsetospeed(&Opt, speed_arr[i]);
                        status = tcsetattr(fd, TCSANOW, &Opt);
                        if  (status != 0)
                                perror("tcsetattr fd1");
                                return;
                }
                tcflush(fd,TCIOFLUSH);
         }

        if (i == 12){
                printf("\tSorry, please set the correct baud rate!\n\n");
                print_usage(stderr, 1);
        }
}
/*
        *@brief  set serial data bit, stop bit and check bit

*/
int set_Parity(int fd,intdatabits,intstopbits,int parity)
{
        struct termios options;
        if  (tcgetattr( fd,&options)  !=  0) {
                perror("SetupSerial 1");
                return(FALSE);
        }
        options.c_cflag&= ~CSIZE ;
        switch (databits) {
        case 7:
                options.c_cflag |= CS7;
        break;
        case 8:
                options.c_cflag |= CS8;
        break;
        default:
                fprintf(stderr,"Unsupported data size\n");
                return (FALSE);
        }

        switch (parity) {
```

```
    case 'n':
    case 'N':
        options.c_cflag&= ~PARENB;   /* Clear parity enable */
        options.c_iflag&= ~INPCK;     /* Enable parity checking */
    break;
    case 'o':
    case 'O':
        options.c_cflag |= (PARODD | PARENB);
        options.c_iflag |= INPCK;             /* Disnable parity checking */
    break;
    case 'e':
    case 'E':
        options.c_cflag |= PARENB;     /* Enable parity */
        options.c_cflag&= ~PARODD;
        options.c_iflag |= INPCK;       /* Disnable parity checking */
    break;
    case 'S':
    case 's':  /*as no parity*/
        options.c_cflag&= ~PARENB;
        options.c_cflag&= ~CSTOPB;
    break;
    default:
        fprintf(stderr,"Unsupported parity\n");
        return (FALSE);
    }

    switch (stopbits) {
    case 1:
    options.c_cflag&= ~CSTOPB;
    break;
    case 2:
        options.c_cflag |= CSTOPB;
    break;
```

```
        default:
              fprintf(stderr,"Unsupported stop bits\n");
              return (FALSE);
        }
        /* Set input parity option */
        if (parity != 'n')
        options.c_iflag |= INPCK;
        options.c_cc[VTIME] = 150; // 15 seconds
        options.c_cc[VMIN] = 0;

        options.c_lflag&= ~(ECHO | ICANON);

        tcflush(fd,TCIFLUSH); /* Update the options and do it NOW */
        if (tcsetattr(fd,TCSANOW,&options) != 0) {
        perror("SetupSerial 3");
              return (FALSE);
        }
        return (TRUE);
}

/**
        *@breif  open serial port
*/
int OpenDev(char *Dev)
{
        int fd = open( Dev, O_RDWR );         //| O_NOCTTY | O_NDELAY
        if (-1 == fd) {
              perror("Can't Open Serial Port");
              return -1;
        } else
              return fd;
}
```

```
/* The name of this program */
const char * program_name;

/* Prints usage information for this program to STREAM (typically
 * stdout or stderr), and exit the program with EXIT_CODE. Does not
 * return.
 */

void print_usage (FILE *stream, int exit_code)
{
fprintf(stream, "Usage: %s option [ dev... ] \n", program_name);
fprintf(stream,
            "\t-h  --help     Display this usage information.\n"
            "\t-d  --device   The device ttyS[0-3] or ttySCMA[0-1]\n"
            "\t-b  --baudrate Set the baud rate you can select\n"
            "\t               [230400, 115200, 57600, 38400, 19200, 9600, 4800, 2400, 1200, 300]\n"
            "\t-s  --string   Write the device data\n");
    exit(exit_code);
}




/*
       *@breif  main()
 */
int main(int argc, char *argv[])
{
       int  fd, next_option, havearg = 0;
       char *device;
       int i=0,j=0;
       int nread;                  /* Read the counts of data */
       char buff[512];         /* Recvice data buffer */
       pid_tpid;
       char *xmit = "1234567890"; /* Default send data */
       int speed, send_mode = 0;
       const char *const short_options = "hd:s:b:m:";

       const struct option long_options[] = {
           { "help",   0, NULL, 'h'},
           { "device", 1, NULL, 'd'},
           { "string", 1, NULL, 's'},
           { "baudrate", 1, NULL, 'b'},
           { "send/recv mode", 0, NULL, 'm'},
           { NULL,     0, NULL, 0  }
       };

       program_name = argv[0];

       do {
           next_option = getopt_long (argc, argv, short_options, long_options, NU
   LL);
           switch (next_option) {
               case 'h':
                   print_usage (stdout, 0);
               case 'd':
                   device = optarg;
                   havearg = 1;
                   break;
               case 'b':
                   speed = atoi(optarg);
                   break;
               case 's':
                   xmit = optarg;
                   havearg = 1;
                   break;
               case 'm':
                   send_mode = atoi(optarg);
                   break;
               case -1:
                   if (havearg)  break;
               case '?':
                   print_usage (stderr, 1);
               default:
                   abort ();
           }
       }while(next_option != -1);

       sleep(1);
       fd = OpenDev(device);

       if (fd> 0) {
           set_speed(fd, speed);
       } else {
           fprintf(stderr, "Error opening %s: %s\n", device, strerror(errno));
           exit(1);
       }

       if (set_Parity(fd,8,1,'N')== FALSE) {
           fprintf(stderr, "Set Parity Error\n");
           close(fd);
           exit(1);
       }
   #if 0
       pid = fork();

       if (pid< 0) {
           fprintf(stderr, "Error in fork!\n");
        } else if (pid == 0){
    #endif
        if (send_mode){
                while(1) {
                        printf("%s SEND: %s\n",device, xmit);
                        write(fd, xmit, strlen(xmit));
                        sleep(1);
                        i++;
                }
    }else {
                while(1) {
                        nread = read(fd, buff, sizeof(buff));
                        if (nread> 0) {
                                buff[nread] = '\0';
                                printf("%s RECV[%d]: %s\n", device, nread, buff);
                        }
                }
        }
        close(fd);
        exit(0);
    }
```

Set up the cross-compilation tool

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
```

Compile the test program for RS232

```
# $CC -o uart_test uart_test.c
```

Use the compiled executable uart_test to test the functionality of axiuartlite.

```
# ./uart_test -d /dev/ttyUL1 -b 115200 -m 0 &


# ./uart_test -d /dev/ttyUL1 -b 115200 -m 1
/dev/ttyUL2 SEND: 1234567890
/dev/ttyUL2 RECV[1]: 1
/dev/ttyUL2 RECV[1]: 2
```

```
/dev/ttyUL2 RECV[1]: 3
/dev/ttyUL2 RECV[1]: 4
/dev/ttyUL2 RECV[1]: 5
/dev/ttyUL2 RECV[1]: 6
/dev/ttyUL2 RECV[1]: 7
/dev/ttyUL2 RECV[1]: 8
/dev/ttyUL2 RECV[1]: 9
/dev/ttyUL2 RECV[1]: 0
/dev/ttyUL2 SEND: 1234567890
/dev/ttyUL2 RECV[1]: 1
/dev/ttyUL2 RECV[1]: 2
/dev/ttyUL2 RECV[1]: 3
/dev/ttyUL2 RECV[1]: 4
/dev/ttyUL2 RECV[1]: 5
/dev/ttyUL2 RECV[1]: 6
/dev/ttyUL2 RECV[1]: 7
/dev/ttyUL2 RECV[1]: 8
/dev/ttyUL2 RECV[1]: 9
/dev/ttyUL2 RECV[1]: 0
```

The information tested can be seen that the axi uartlite serial communication is normal and the data sent is received normally.

8. How to add your app

The porting of Linux application is typically divided into two phases, the development debugging phase and the production deployment phase. During the development debugging phase, we can cross-compile our well-written application using the MYIR-built SDK and then remotely copy it to the target host for testing. The production deployment phase requires writing recipe files for your app and building production images using BitBake.

8.1. Makefile based applications

This Makefile is actually a document that defines a series of compilation rules that record the details of how the original code is compiled! Once Makefile is written, it only requires a make command, and the entire project is fully automatically compiled, greatly improving the efficiency of software development. When developing Linux programs, Makefile was widely used in both kernels, drivers, and applications.

Make is a command tool, a command tool that interprets instructions in makefile. It simplifies the instructions issued in the compilation process, and when mak is executed, make searches the text file makefile (or makefile) in the current directory to perform the corresponding operations. make automatically determines whether the original file has been changed, automatically recompiling the changed source code.

The following is an actual example of Makefile writing and making execution (implementing the control of LED light switch on the MYS-ZU5EV board). Makefile has its own set of rules.

```
target … : prerequisites …
        command
```

- target can be an object file, an execution file, or a label.
- prerequisites is to generate the file or target that target needs.
- command is the command that make needs to be executed.

```
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
	${CC} -I . -c gpioctrl.c
```

- $(notdir $(path))： Indicates that the path directory is removed from the path name, leaving only the current directory name, such as the current Makefile directory as /home/examples/gpioapp, and executing as TARGET=gpioapp
- $(patsubst pattern, replacement,text) ：Replace the formatted "pattern" characters in the text with reset, such as $(patsubst %c, %o, $(shell ls *.c), which means that files with the current directory suffix of .c are listed first, and then replaced with a suffix of .o
- CC：C The name of the compiler
- CXX: C++ The name of the compiler
- clean: a agreed goal

gpioctrl.c is as follows：

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

/*
* ./gpioctrl /sys/class/leds/led_sys on
* ./gpioctrl /sys/class/leds/led_sys off
 */
int main(int argc, char **argv)
{
    int fd;
    char status;

    /* 1. Check argc */
    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 = '0';
        write(fd, &status, 1);
    }
    else
    {
        status = '1';
        write(fd, &status, 1);
    }

    close(fd);

    return 0;
```

```
}
```

Use the make command to compile and generate executable files on the target machine gpioapp_test.

Load SDK environment variables to the current shell:

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
```

Execute make:

```
# make CC=aarch64-xilinx-linux-gcc
```

As you can see from the results of the previous command, the compiler used is the compiler established by setting up the CC variables defined in the script.

Copy gpioapp_test executables to the board's catalog via transport media such as networks, u disks or tf card:

```
# ./gpioapp_test /sys/class/leds/led_sys/brightness on
# ./gpioapp_test /sys/class/leds/led_sys/brightness off

```

The MYS-ZU5EV board controls the LED light switch next to the reset button.

8.2. Qt-based apps

Qt is a cross-platform graphics application development framework that is used on devices and platforms of different sizes, while providing different copyright versions for users to choose from. MYS-ZU5EV uses Qt 5.13.2 for application development. In Qt application development, it is recommended to use QtCreator integrated development environment, Qt application can be developed under Linux PC, automated cross-compiled as a board arm architecture program.

1) QtCreator install and configure

Get the qtcreator installation package QT website for download from the QT official website or MYIR official package : http://download.qt.io/development_releases/qtcreator/4.1/4.1.0-rc1/. QtCreator installation package is a binary program that can be executed directly ./qt-creator-opensource-linux-x86_64-4.1.0-rc1.run, please refer to the documentation "MYS-ZU5EV_QT Application Development Notes" on CD for installation and configuration details.

8.3. The application starts from boot

1) The configuration of the application in Petalinux

Often our applications also need to be boot-on and self-started, which can also be done in recipes. Here's an example of how to use Petalinux to build a production image that contains a specific application, where the FTP service program uses open source Proftpd, where each version of the source code is located ftp://ftp.proftpd.org/distrib/source/.

Before we write a recipe from scratch, we can find out if there are ready-made or similar recipes in Petalinux's existing layers of components/yocto/layers, as follows:

```
# find components/yocto/layers –name proftpd
```

We can also index it at openEmbedded's official website （http://layers.openembedded.org/layerindex/branch/master/layers/）,find out if there are recipes for the same or similar app.

This section focuses on how to port FTP services to the target machine. A search of the Petalinux layer revealed that a recipe for proftpd already existed, but it was not added to the system image. The specific migration process is described in detail below.

- **Find Petalinux's proftpd recipe**

```
# find components/yocto/layers-name proftpd
components/yocto/layers/meta-openembedded/meta-networking/recipes-daemons/proftpd
```

Note: You can see that the propftpd recipe already exists in the Petalinux project.

- **Package proftpd to the file system**

Add a line to <plnx-proj-root>/project-spec/meta-user/conf/user-rootfsconfig:

```
CONFIG_proftpd
```

Then enable proftpd in the configuration of rootfs:

```
# petalinux-config -c rootfs
```

Select user packages ->proftpd.

- **Rebuild the image**

```
# petalinux-build
```

- **Burn the new image**

Once the system is built, re-burn the image and see if the proftpd service is running:

```
# ps -axu | grep proftpd
nobody    673  0.0  0.1   3256  1336 ?       Ss   07:49   0:00 proftpd: (accepting con
nections)
root      741  0.0  0.0   2020   424 ttyPS0   S+   07:50   0:00 grep proftpd
```

Here's a look at FTP account settings. FTP clients have three types of login accounts: anonymous, regular and root.

- **An anonymous account**

The user name is ftp, no password needs to be set, the user can view the contents of the system/var/lib/ftp directory after logging in, by default there is no write permission. Because the system does not have a /var/lib/ftp directory by default, the user is required to create a directory/var/lib/ftp on the target machine. In order to minimize the modification of meta-openembbed, we can do this by adding the append file "proftpd_%.bbappend" to the proftpd recipe/var/lib/ftp directory creation.

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

The edited proftpd_%.bbappend" needs to be placed in the recipes-daemons\proftpd directory below meta-user.

- **General account**

After using the useradd and passswd commands on the target machine to create a regular user and set the user password, the client can also log on to that user's HOME directory using that regular account. Petalinux creates a normal account with a username and password for petalinux by default.

- **Root account**

If you need an open root account to log on to the FTP server, you need to modify the /etc/proftpd.conf file and add a line of configuration "RootLogin on" to the file. At the same time, you also need to set a password for the root account, after restarting the proftpd service, the client can also use the root account to login to the target machine.

```
# /etc/init.d/proftpd restart
```

Note: Modify /etc/proftpd.conf enable root account login for testing purposes only, for more configurations of /etc/proftpd.conf, please refer to http://www.proftpd.org/docs/example-conf.html。

2) Implement self-starting of the application

This section will take the proftpd recipe as an example to show you how to add an application recipe and implement a boot-up self-start of the program from the recipe source level. The proftpd formula is located in the petalinux project components/yocto/layers/meta-openembedded/meta-networking/recipes-daemons/proftpd, as follows.

```
├── 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 Recipes to build proftpd services
- proftpd.service Start the service for boot-on
- proftpd-basic.init The startup script for proftpd

The proftpd_1.3.6.bb recipe specifies the source code path to get the proftpd service program and some patch files for that version of the 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 \
"
```

The recipe also specifies the configuration (do_configure) of the proftpd and the installation process (do_install):

```
# proftpd uses libltdl which currently makes configuring using
# autotools.bbclass a pain...
do_configure () {
    install -m 0755 ${STAGING_DATADIR_NATIVE}/gnu-config/config.guess ${S}
    install -m 0755 ${STAGING_DATADIR_NATIVE}/gnu-config/config.sub ${S}
    oe_runconf
    cp ${STAGING_BINDIR_CROSS}/${HOST_SYS}-libtool ${S}/libtool
}

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
        # installproftpd 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\nAuthPAMConfigproft
pd' \
            ${D}${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

    # ftpmailperl script, which reads the proftpd log file and sends
    # automatic email notifications once an upload finishs,
    # depends on an old perlMail::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
}
```

These two functions correspond to the config and install tasks of the BitBake build process (for more information about the task, please refer to https://docs.yoctoproject.org/ref-manual/tasks.html)。

The current target machine uses init as an initial management subsystem, a collection of linux system infrastructure components that provide a system and service manager that runs as PID 1 and is responsible for starting other programs. Proftpd service boot-start service file proftpd-basic.init reads as follows:

```
#!/bin/sh

### BEGIN INIT INFO
# Provides:          proftpd
# Required-Start:    $remote_fs $syslog $local_fs $network
# Required-Stop:     $remote_fs $syslog $local_fs $network
# Should-Start:      $named
# Should-Stop:       $named
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Starts ProFTPD daemon
# Description:       This script runs the FTP service offered
#                    by the ProFTPD daemon
### END INIT INFO

# Start the proftpd FTP daemon.

PATH=/bin:/usr/bin:/sbin:/usr/sbin
DAEMON=/usr/sbin/proftpd
```

```
NAME=proftpd

# Defaults
RUN="no"
OPTIONS=""
CONFIG_FILE=/etc/proftpd.conf

PIDFILE=`grep -i '^pidfile' $CONFIG_FILE|awk '{ print $2 }'`
if [ "x$PIDFILE" = "x" ];
then
    PIDFILE=/var/run/proftpd.pid
fi

# Read config (will override defaults)
[ -r /etc/default/proftpd ] && . /etc/default/proftpd

trap "" 1
trap "" 15

test -f $DAEMON || exit 0

. /etc/init.d/functions

#
# Servertype could be inetd|standalone|none.
# In all cases check against inetd and xinetd support.
#
if ! egrep -qi "^[[:space:]]*ServerType.*standalone" $CONFIG_FILE
then
    if egrep -qi "server[[:space:]]*=[[:space:]]*/usr/sbin/proftpd" /etc/xinetd.conf 2>
/dev/null || \
       egrep -qi "server[[:space:]]*=[[:space:]]*/usr/sbin/proftpd" /etc/xinetd.d/* 2>/
dev/null || \
       egrep -qi "^ftp.*/usr/sbin/proftpd" /etc/inetd.conf 2>/dev/null
    then
        RUN="no"
        INETD="yes"
    else
        if ! egrep -qi "^[[:space:]]*ServerType.*inetd" $CONFIG_FILE
        then
            RUN="yes"
            INETD="no"
        else
            RUN="no"
            INETD="no"
        fi
    fi
fi

# /var/run could be on a tmpfs

[ ! -d /var/run/proftpd ] && mkdir /var/run/proftpd

inetd_check()
{
    if [ ! -x /usr/sbin/inetd -a ! -x /usr/sbin/xinetd ]; then
        echo "Neither inetd nor xinetd appears installed: check your configuration."
    fi
}

start()
{
    set -e
    echo -n "Starting ftp server $NAME... "
    start-stop-daemon --start --quiet --pidfile "$PIDFILE" --oknodo --exec $DAEMON -- -c $CONFIG_FILE $OPTIONS
    echo "done."
}

signal()
{

    if [ "$1" = "stop" ]; then
        SIGNAL="TERM"
        echo -n "Stopping ftp server $NAME... "
    else
        if [ "$1" = "reload" ]; then
            SIGNAL="HUP"
            echo -n "Reloading ftp server $NAME... "
        else
            echo "ERR: wrong parameter given to signal()"
            exit 1
        fi
    fi
    if [ -f "$PIDFILE" ]; then
        start-stop-daemon --stop --signal $SIGNAL --quiet --pidfile "$PIDFILE"
        if [ $? = 0 ]; then
            echo "done."
            return
        else
            SIGNAL="KILL"
            start-stop-daemon --stop --signal $SIGNAL --quiet --pidfile "$PIDFILE"
            if [ $? != 0 ]; then
                echo
                [ $2 != 0 ] || exit 0
            else
                echo "done."
                return
            fi
        fi
        if [ "$SIGNAL" = "KILL" ]; then
            rm -f "$PIDFILE"
        fi
    else
        echo "done."
        return
    fi
}

case "$1" in
    start)
        if [ "x$RUN" = "xyes" ] ; then
            start
        else
            if [ "x$INETD" = "xyes" ] ; then
                echo "ProFTPD is started from inetd/xinetd."
                inetd_check
            else
                echo "ProFTPD warning: cannot start neither in standalone nor in inetd/xinetd mode. Check your configuration."
            fi
        fi
        ;;

    force-start)
        if [ "x$INETD" = "xyes" ] ; then
            echo "Warning: ProFTPD is started from inetd/xinetd (trying to start anyway)."
            inetd_check
        fi
        start
        ;;
    stop)
        if [ "x$RUN" = "xyes" ] ; then
            signal stop 0
        else
            if [ "x$INETD" = "xyes" ] ; then
                echo "ProFTPD is started from inetd/xinetd."
                inetd_check
            else
                echo "ProFTPD warning: cannot start neither in standalone nor in inetd/xinetd mode. Check your configuration."
            fi
        fi
        ;;

    force-stop)
        if [ "x$INETD" = "xyes" ] ; then
            echo "Warning: ProFTPD is started from inetd/xinetd (trying to kill anyway)."
            inetd_check
        fi
        signal stop 0
        ;;

    reload)
        signal reload 0
        ;;

    force-reload|restart)
        if [ "x$RUN" = "xyes" ] ; then
            signal stop 1
            sleep 2
            start
        else
            if [ "x$INETD" = "xyes" ] ; then
                echo "ProFTPD is started from inetd/xinetd."
                inetd_check
            else
                echo "ProFTPD warning: cannot start neither in standalone nor in inetd/xinetd mode. Check your configuration."
            fi
        fi
        ;;

    status)
        if [ "x$INETD" = "xyes" ] ; then
            echo "ProFTPD is started from inetd/xinetd."
            inetd_check
            exit 0
        else
            if [ -f "$PIDFILE" ]; then
                pid=$(cat $PIDFILE)
            else
                pid="x"
            fi
            if [ `pidof proftpd|grep "$pid"|wc -l` -ne 0 ] ; then
                echo "ProFTPD is started in standalone mode, currently running."
                exit 0
            else
                echo "ProFTPD is started in standalone mode, currently not running."
                exit 3
            fi
        fi
        ;;

    check-config)
```

```
        $DAEMON -t >/dev/null && echo "ProFTPD configuration OK" && exit 0
        exit 1
        ;;

    *)
        echo "Usage: /etc/init.d/$NAME {start|status|force-start|stop|force-stop|reloa
d|restart|force-reload|check-config}"
        exit 1
        ;;
esac

exit 0
```

- start() is the function that is executed when the service is started.
- stop() is a function executed at the end of the service.
- restart() is a function executed when the service is restarted.
- status() is a function that is executed when the state of the process is queried.

When adding an app that they have written, they can refer to the "Application Auto Run at Startup" section of Chapter 8 of the UG1144 manual.

8.4. Application examples

Common demo programs, as well as source code, are available on the CD in the 04_Sources\Example directory.

8.4.1. CAN application example

CAN test requires two MYS_ZU5EV boards to be tested.
Go to the 04_Sources\Example\can directory and compile the code directly to produce executables.

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
# make
```

The compiled executable files are can_receive, can_send copied into two tf cards, two tf cards are inserted into two boards, and the two boards are powered on to test the can communication function :
One of the boards operates as the receiving end :

```
# ip link set can0 up type can bitrate 500000
# ./can_receive
```

The other board operates as the sending end :

```
# ip link set can0 up type can bitrate 500000
# ./can_send -d can0 -i 123 33 44 55
```

8.4.2. I2C application example

I2C test is tested as follows.
Go to the 04_Sources\Example\i2c directory and compile the code directly to produce an executable file.

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
# make
```

Copy the compiled executable i2c_flash into the tf card and start the board test :

```
# ./i2c_flash /dev/i2c-1
```

8.4.3. Network application example

Network communication can be tested with board and computer.
Go to the 04_Sources\Example\network directory and compile the code directly to produce an executable file.

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
# make
```

Copy the compiled executable file arm_server into the tf card, and operate as follows：

```
# ./arm_server
```

Enter into the ubuntu system on the computer side, test operation is as follows, 192.168.xxx.xxx is the ip address of the board：

```
$ ./pc_client 192.168.xxx.xxx
```

8.4.4. Uart application example

The 1st and 3th feet of the board's serial J12 interface can be connected for testing.

Go to the 04_Sources\Example\uart directory and compile the code directly to produce an executable file.

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
# make
```

Copy the compiled executable file uart_test into the tf card, and operate as follows :

```
# ./uart_test -d /dev/ttyUL1 -b 115200 -m 0 &
# ./uart_test -d /dev/ttyUL1 -b 115200 -m 1
/dev/ttyUL1 SEND: 1234567890
/dev/ttyUL1 RECV[1]: 1
/dev/ttyUL1 RECV[1]: 2
/dev/ttyUL1 RECV[1]: 3
/dev/ttyUL1 RECV[1]: 4
/dev/ttyUL1 RECV[1]: 5
/dev/ttyUL1 RECV[1]: 6
/dev/ttyUL1 RECV[1]: 7
/dev/ttyUL1 RECV[1]: 8
/dev/ttyUL1 RECV[1]: 9
/dev/ttyUL1 RECV[1]: 0
```

8.4.5. Framebuffer application example

Test the DP display through the framebuffer program.

Go to the 04_Sources\Example\framebuffer directory and compile the code directly to produce an executable.

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
# make
```

Copy the compiled executable framebuffer_test into the tf card, and operate as follows :

```
# ./framebuffer_test
```

8.4.6. HDMIin application example

Through the qt-hdmi program, the test hdmi input image is displayed by DP monitor.

Go to the 04_Sources\Example\qt-hdmi directory and compile the code directly to produce an executable file.

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
# /opt/petalinux/2020.1/sysroots/x86_64-petalinux-linux/usr/bin/qmake
# make
```

Follow "4.1. Make an SD card launcher" chapter, burned 02_Images\mys-zu5ev-full.img.gz into the tf card, start board by tf card mode, copy the compiled executable qt-hdmi into the tf card, and operate as follows :

```
# media-ctl -v --set-format '"a0010000.v_tpg":0 [RBG24 1920x1080 field:none]'
# export DISPLAY=:0.0
# ./qt-hdmi
```

8.4.7. MIPI camera application example

MIPI camera applications require an IMX334 mipi camera to be tested.

Through the qt-camera program, the test IMX334 mipi camera input image is displayed by DP monitor.

Go to the 04_Sources\Example\qt-camera directory and compile the code directly to produce an executable file.

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
# /opt/petalinux/2020.1/sysroots/x86_64-petalinux-linux/usr/bin/qmake
# make
```

Follow "4.1. Make an SD card launcher" chapter, burned 02_Images\mys-zu5ev-mipi.img.gz into the tf card, start board by tf card mode, copy the compiled executable qt-camera into the tf card, and operate as follows :

```
# export DISPLAY=:0.0
# ./qt-camera
```

8.4.8. VCU application example

ZU5EV chip's VCU supports multi-standard video encoding and decoding, including the High Efficiency Video Coding (HEVC) and Advanced Video Coding (AVC) H.264 standards. The unit includes encoding (compression) and decoding (unzipping) functions.

To implement the logical function of VCU in FPGAs, refer to the "6.2 VCU" section of “MYS-ZUEV_FPGA Development Manual” on CD-ROM. Software functionality is then implemented through petalinux.

- **Implementation process of petalinux software in VCU**

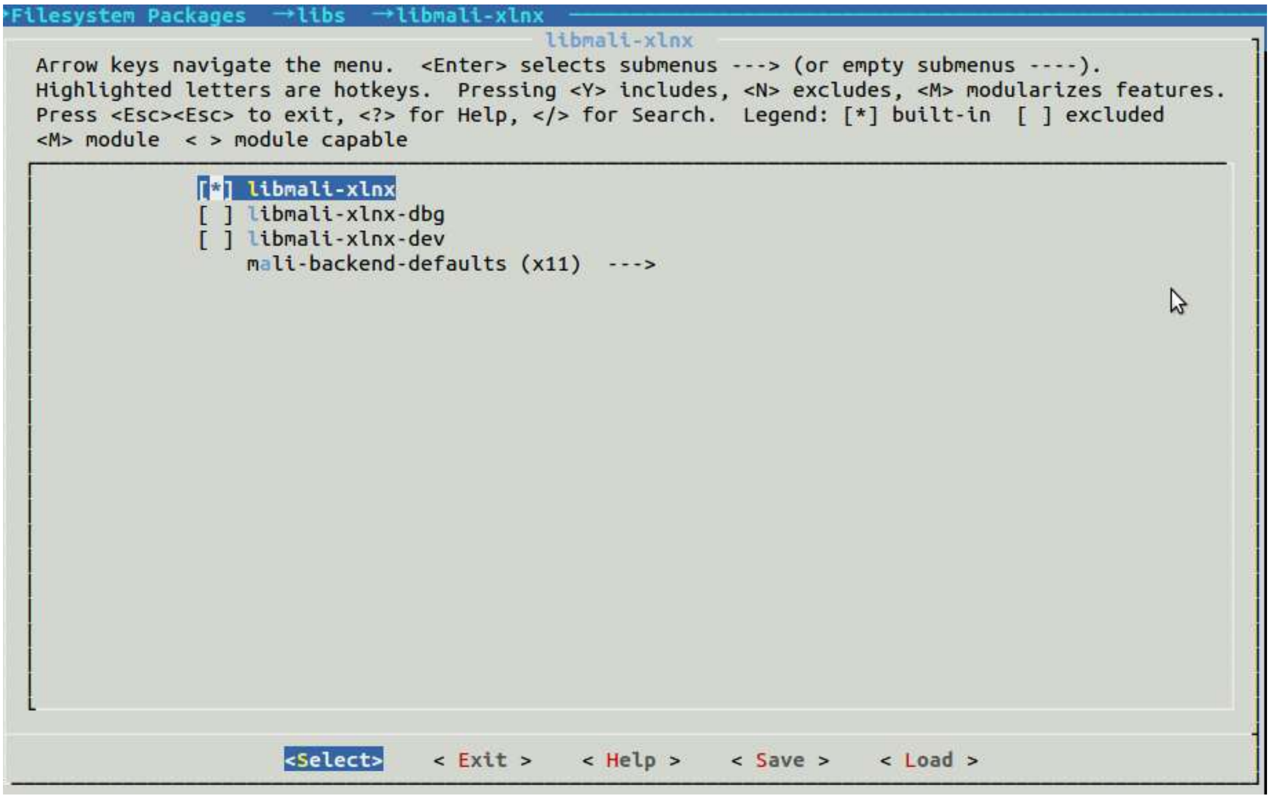
（1）Copy the design_1_wrapper.xsa generated by the sample project 05-ProgrammableLogic_Source/project_fz5_vcu_707.rar on CD to the directory/home/work/vivadoxsa/.

```
# petalinux-create -t project -s mys_zu5ev2020_4G_full.bsp
# petalinux-config --get-hw-description=/home/work/vivadoxsa/
```

（2）Configure the rootfs system as follows:

```
# petalinux-config –c rootfs
```

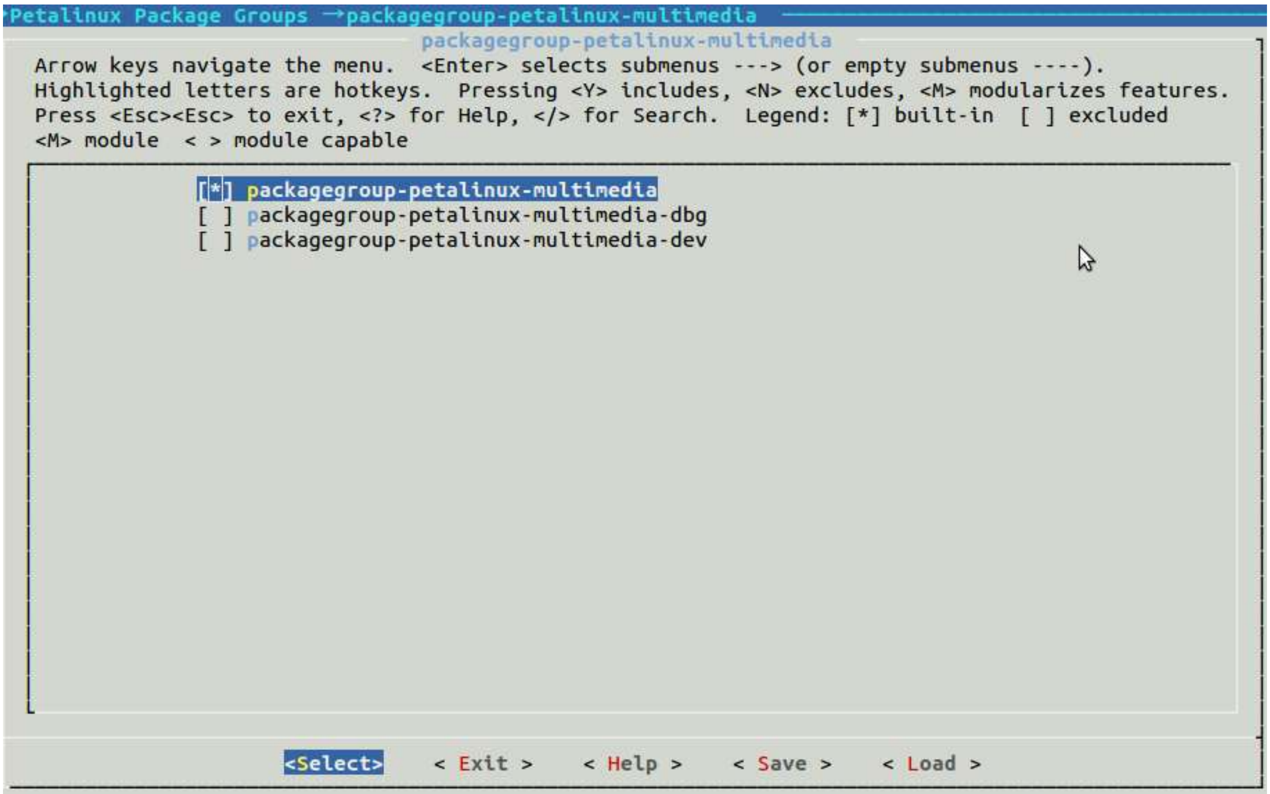
Configure libmali；



```
Filesystem Packages  →libs  →libmali-xlnx
                           libmali-xlnx
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.
  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded
  <M> module  < > module capable

              [*] libmali-xlnx
              [ ] libmali-xlnx-dbg
              [ ] libmali-xlnx-dev
                  mali-backend-defaults (x11)  --->

          <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 8-1 Configure libmaili

Configure multimedia ;



```
Petalinux Package Groups →packagegroup-petalinux-multimedia
                         packagegroup-petalinux-multimedia
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.
  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded
  <M> module  < > module capable

             [*] packagegroup-petalinux-multimedia
             [ ] packagegroup-petalinux-multimedia-dbg
             [ ] packagegroup-petalinux-multimedia-dev

           <Select>    < Exit >    < Help >    < Save >    < Load >
```

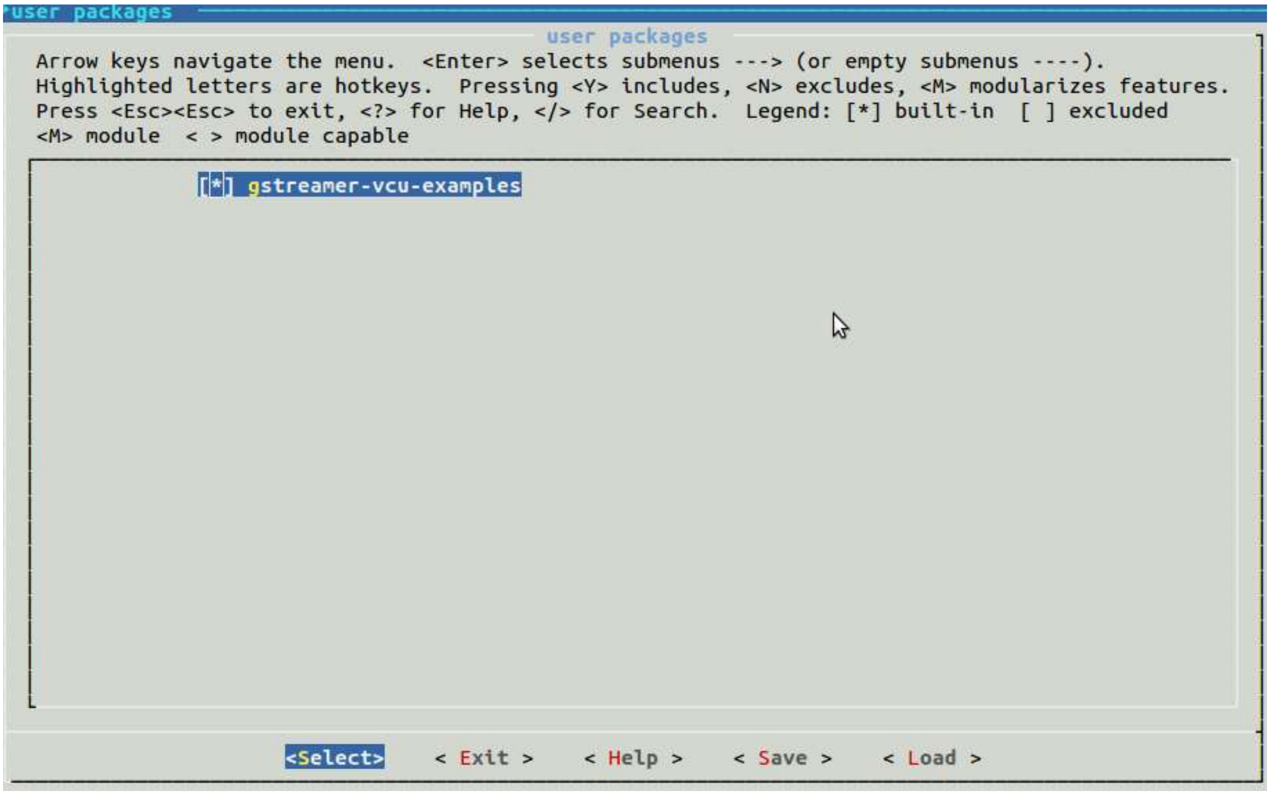
Figure 8-2 Configure multimedia

Configure gstreamer-vcu-examples ;

Add a line to <plnx-proj-root>/project-spec/meta-user/conf/user-rootfsconfig:

```
CONFIG_gstreamer-vcu-examples
```

Then enable gstreamer-vcu-examples in the rootfs configuration.



```
user packages
                                   user packages
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.
  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded
  <M> module  < > module capable

              [*] gstreamer-vcu-examples

                 <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 8-3 Configure gstreamer-vcu-examples

（3）Configure the petalinux project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi device tree file as follows:

```
/include/ "system-conf.dtsi"

/ {
	chosen {
		bootargs = "console=ttyPS0,115200 earlyprintk root=/dev/mmcblk1p2 rw rootwait clk_ignore_unused";
		stdout-path = "serial0:115200n8";
	};

	leds {
		compatible = "gpio-leds";
		led1 {
			label = "rs485_de";
			gpios = <&gpio 12 0>;
			linux,default-trigger = "gpio";
		};
		led2 {
			label = "wdt_en";
			gpios = <&gpio 33 0>;
			linux,default-trigger = "gpio";
		};
		led3 {
			label = "led_sys";
			gpios = <&gpio 43 0>;
			linux,default-trigger = "gpio";
		};
	};

	watchdog {
		compatible = "gpio-watchdog";
	};
};

&gem3 {
	phy-handle = <&phy0>;
	phy-mode = "rgmii-id";
	phy0: phy@21 {
		reg = <4>;
	};
};

&sdhci0 {
	no-1-8-v;
};

&sdhci1 {
	disable-wp;
	no-1-8-v;
};
```

```
&qspi {
	flash@0 {
		compatible = "m25p80";
		#address-cells = <1>;
		#size-cells = <1>;
		reg = <0x0>;
		spi-tx-bus-width = <1>;
		spi-rx-bus-width = <4>;
		spi-max-frequency = <54000000>;
	};
};
```

（4）Compiling petalinux bsp can generate vcu petalinux software:

```
# petalinux-build
```

- **VCU test**

Follow "4.1. Make an SD card launcher" chapter, burned 04_Sources\Example\vcu\mys-zu5ev-vcu.img.gz into the tf card, start board by tf card mode, enter the system operation as follows, you can use the vcu decoding function to play mp4 video.

```
# vcu-demo-decode-display.sh -i /mnt/sd-mmcblk1p1/test.mp4
```

9. Reference

https://www.kernel.org/

https://www.xilinx.com/

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 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