Xilinx Development Environment

Ultrascale+ Linux Development Basic

ALINX

Version Record

Version	Date	Release By	Description
Rev1.01	2022-04-30	Rachel Zhou	First Release

The ALINX official Documents are in Chinese, and the English version was translated by **Shanghai Tianhui** Trading Company. They has **not been officially Review by ALINX** and are for reference only. If there are any errors, please send email feedback to support@aithtech.com for correction.

Amazon Store: https://www.amazon.com/alinx

Aliexpress Store:

https://alinxfpga.aliexpress.com/store/911112202?spm=a2g0o.detail.1000007.1.704e2bedqLBW90

Ebay Store: https://www.ebay.com/str/alinxfpga

Customer Service Information

Skype: rachelhust@163.com

Wechat: rachelhust

Email: support@aithtech.com and rachehust@163.com

Content

Version Record 2
Content 3
Part 1: Customizing Linux with Petalinux 5
Part 1.1: Create a project with Petalinux 5
Part 1.2: Modify the device tree 8
Part 1.3: Generate BOOT file 9
Part 1.4: Run Linux 10
Part 1.5: Petalinux Use Process Summary 12
Part 1.6: Save a Project as a Template Project 13
Part 1.7: Some Official Optimization Suggestions 14
Part 2: Program hello world 15
Part 2.1: Code 15
Part 2.2: Compile the Host Program 15
Part 2.3: Cross Compile Arm Program 16
Part 3: gpio Control LED 18
Part 3.1: Device tree Automatically Generated by Petalinux 18
Part 3.2: Petalinux Project Users Modify the Device Tree 19
Part 3.3: Project creation 19
Part 3.4: Run The Program 22
Part 4: Add Boot Scripts and User Files 24
Part 4.1:Ready to work 24
Part 4.2: Self-Start Script 24
Part 5: SD Card Root File System 27
Part 5.1: Petalinux Configuration 27
Part 5.2: Make SD Card File System 28
Part 5.2.1: SD card modify partition 28

Part 5.2.2: Unzip the root file system to the EXT4 partition of the SD card 33
Part 6: QT and OPENCV Cross-Compilation Environment 34
Part 6.1: Configure the Root File System 34
Part 6.2: Install SDK 38
Part 6.3: QT Creator Cross-compilation Environment Setting40
Part 6.3: Create QT test project 45
Part 6.5:Run the cross-compiled version of the QT program49
Part 7: Use Vitis to Develop Linux Programs 51
Part 7.1: Use Vitis to build Linux applications 51
Part 7.2: Run via NFS share 56
Part 8: Vitis Accelerates Basic Platform Creation 58
Part 8.1: Vivado hardware platform 58
Part 8.2: Export XSA 61
Part 8.3: Vitis Software Platform 64
Part 8.3.1: Petalinux configuration 64
Part 8.3.2: Create vitis platform 68
Part 8.3.3: Test Vitis Acceleration 72
Part 9: NVMe SSD operation under Linux 78
Part 9.1: petalinux Configuration 78
Part 9.2: Set up partition 78
Part 9.3: Read and write speed 80
Part 10: Boot Linux from QSPI Flash 81
Part 11: Boot Linux from EMMC 88
Part 11.1: Format EMMC and Mount 88
Part 11.2: Mount the Virtual Machine Folder using NFS 89
Part 11.3: Make EMMC boot system 90

Part 1: Customizing Linux with Petalinux

Special reminder, the environment construction tutorial used in this tutorial is in the Linux of "Xilinx Development Environment Installation Tutorial", please set up the development environment according to the tutorial first.

This tutorial refers to the official Xilinx tutorial ug1144-petalinux-tools-reference-guide, and uses Petalinux to build a basic Linux system.

Part 1.1: Create a project with Petalinux

1) In the user directory, create a working directory. The "~" in the path represents the user's home path, which is equivalent to /home/alinx2020

```
mkdir  -p  ~/peta_prj/linxPsBase/hardware
```

2) Copy the vivado exported xsa file into the hardware folder. Here I use the xsa compiled by the ps_hello project in course2 (the routines in this document are all 4ev as an example, for other models, please use the vivado project corresponding to the chip)

Note: The xsa file exported by the vivado project of ps_base is used here

3) Enter the project directory, "~" in the path represents the user's

home path, which is equivalent to /home/alinx2020

```
cd ~/peta_prj/linxPsBase/
```

4) Set the petalinux environment variable, run the following command, Environment variables are only valid in the current terminal, so the following petalinux-related commands must be entered in the current terminal and need to be re-run after closing

```
source /opt/pkg/petalinux/settings.sh
```

5) Create a petalinux project with the name "petalinux" and the type "project", using the zynqMP template

```
petalinux-create -t project -n petalinux --template zynqMP
```

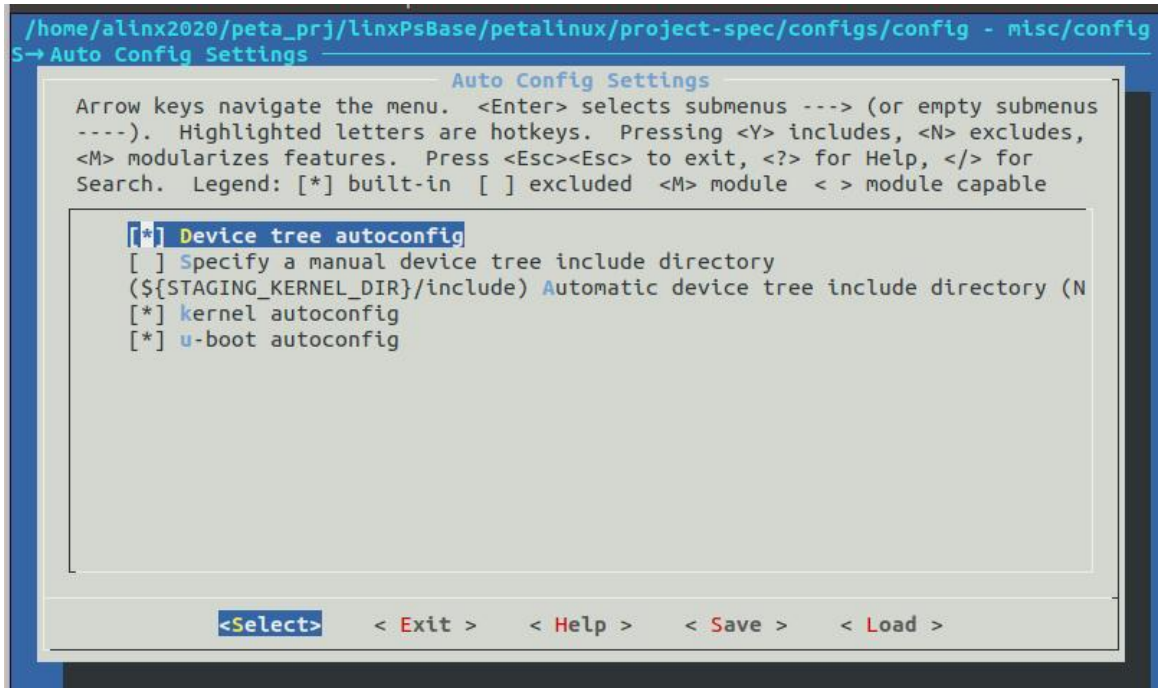
6) Use the following command to enter the petalinux directory, the "~" in the path represents the user's home path, which is equivalent to /home/alinx2020

```
cd ~/peta_prj/linxPsBase/petalinux
```

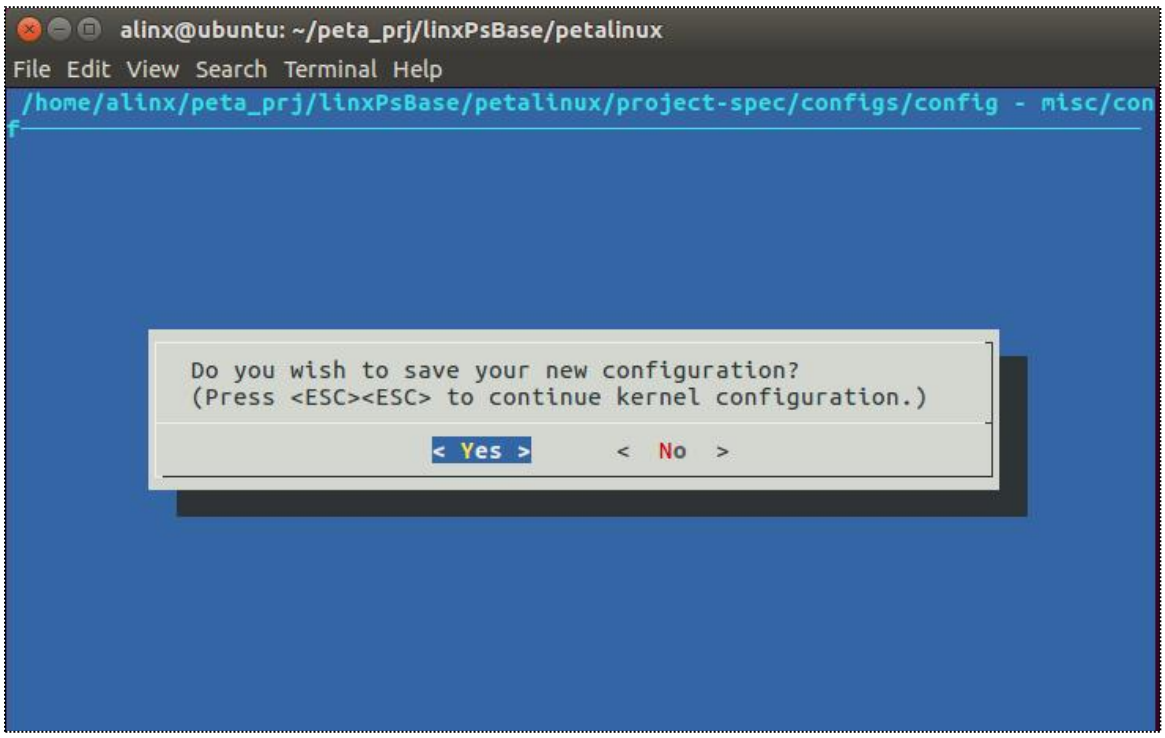
7) Configure the hardware information of the Petalinux project, there can only be one xsa file in the hardware information directory

```
petalinux-config --get-hw-description ../hardware/
```

8) The project configuration item interface pops up, as shown in the figure below. In the "Auto Config Settings" option, check "Device tree autoconfig", "kernel autoconfig" and "u-boot autoconfig", then "Save", then "Exit"



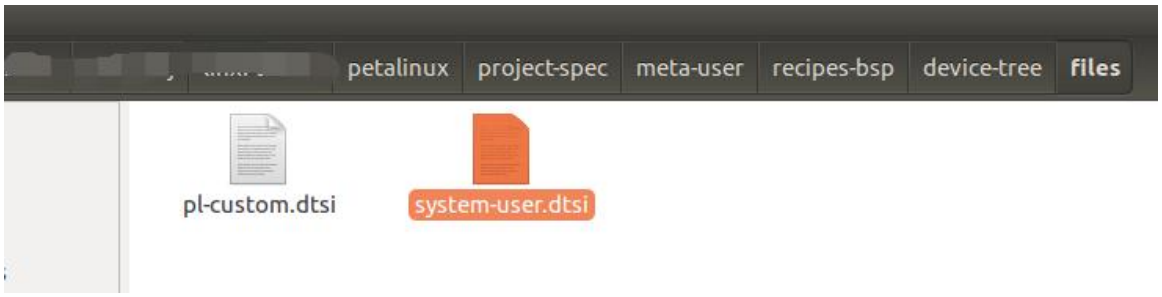
9) If pop up whether to save the configuration interface, select <Yes>, press Enter



10)Wait for the configuration to complete

```
alinx2020@ubuntu:~/peta_prj/linxPsBase/petalinux$ petalinux-config --get-hw-description
../hardware/
INFO: sourcing build tools
INFO: Getting hardware description...
INFO: Rename design_1_wrapper.xsa to system.xsa
[INFO] generating Kconfig for project
[INFO] menuconfig project


*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

[INFO] sourcing build environment
[INFO] generating u-boot configuration files, This will be deprecated in upcoming release
s
[INFO] generating kernel configuration files, This will be deprecated in upcoming release
s
[INFO] generating kconfig for Rootfs
[INFO] silentconfig rootfs
[INFO] generating plnxtool conf
[INFO] generating user layers
[INFO] generating workspace directory
alinx2020@ubuntu:~/peta_prj/linxPsBase/petalinux$
```

Part 1.2: Modify the device tree



1) Modify the content of the " project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi" file in the above figure as follows:

```
/include/ "system-conf.dtsi"
/ {
};

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

/* USB    */
&dwc3_0 {
```

```
        status = "okay";
        dr_mode = "host";
};
```

2) Enter the command to start compiling

```
petalinux-build
```

Note: This step must ensure that the ubuntu system can connect to the network, petalinux needs to download some source code from github

3) Compilation is complete

```
alinx@ubuntu:~/peta_prj/linxPsBase/petalinux$ petalinux-build
[INFO] building project
[INFO] sourcing bitbake
INFO: bitbake petalinux-user-image
Loading cache: 100% |##################################################################| Time: 0:00:00
Loaded 3460 entries from dependency cache.
Parsing recipes: 100% |##################################################################| Time: 0:00:03
Parsing of 2569 .bb files complete (2534 cached, 35 parsed). 3461 targets, 137 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% |###############################################################| Time: 0:00:08
Checking sstate mirror object availability: 100% |#######################################| Time: 0:00:24
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
NOTE: Tasks Summary: Attempted 3132 tasks of which 3132 didn't need to be rerun and all succeeded.
INFO: Copying Images from deploy to images
NOTE: Failed to copy built images to tftp dir:  /tftpboot
[INFO] successfully built project
alinx@ubuntu:~/peta_prj/linxPsBase/petalinux$
```

Part 1.3: Generate BOOT file

1) Enter the file directory

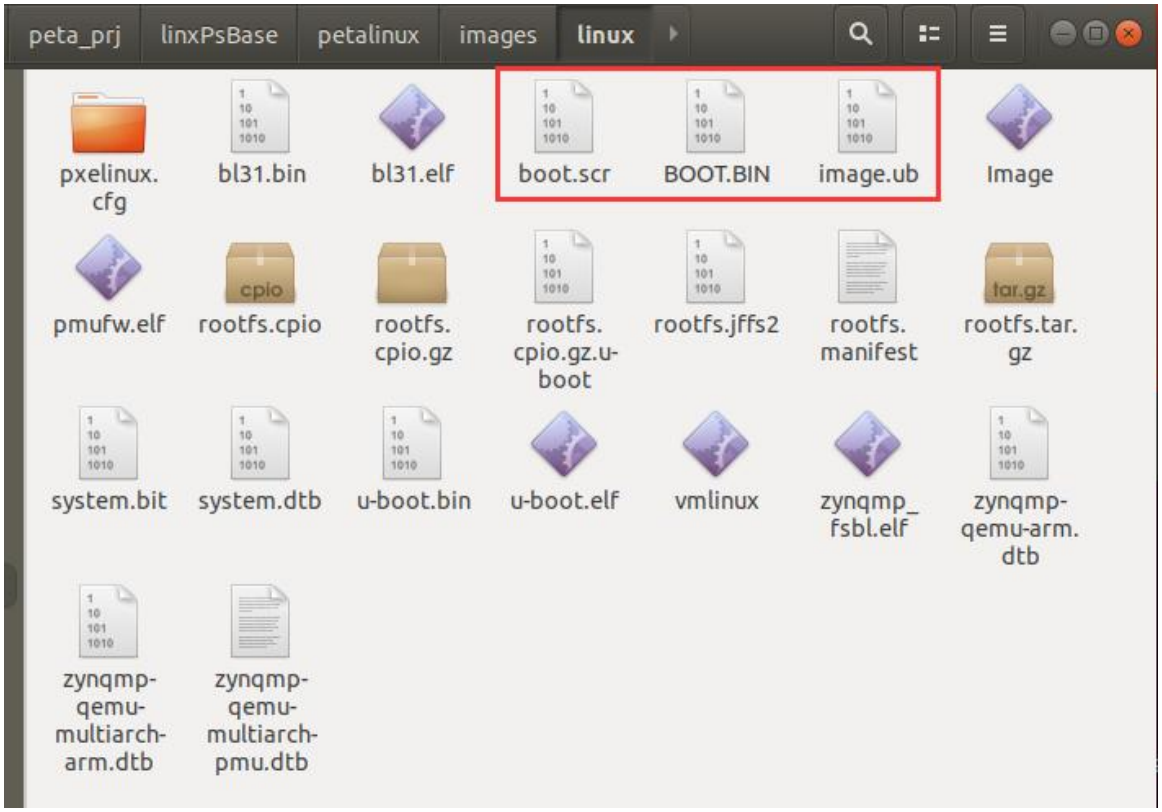
```
cd    ~/peta_prj/linxPsBase/petalinux/images/linux
```

2) Package file, "--u-boot" parameter is the content to be packaged and synthesized, and all u-boot dependencies will be synthesized, "--fpga" refers to the file with bit, and "--force" is forced synthesis, overwriting the previous synthesis BOOT.BIN, notice that there are 2 dashes in the command.

```
petalinux-package –boot    --u-boot    --fpga    --force
```

```
alinx2020@ubuntu:~/peta_prj/linxPsBase/petalinux$ petalinux-package --boot --u-boot  --fpga  --force
INFO: sourcing build tools
INFO: File in BOOT BIN: "/home/alinx2020/peta_prj/linxPsBase/petalinux/images/linux/zynqmp_fsbl.elf"
INFO: File in BOOT BIN: "/home/alinx2020/peta_prj/linxPsBase/petalinux/images/linux/pmufw.elf"
INFO: File in BOOT BIN: "/home/alinx2020/peta_prj/linxPsBase/petalinux/project-spec/hw-description/design_1_wrapper.bit"
INFO: File in BOOT BIN: "/home/alinx2020/peta_prj/linxPsBase/petalinux/images/linux/bl31.elf"
INFO: File in BOOT BIN: "/home/alinx2020/peta_prj/linxPsBase/petalinux/images/linux/system.dtb"
INFO: File in BOOT BIN: "/home/alinx2020/peta_prj/linxPsBase/petalinux/images/linux/u-boot.elf"
INFO: Generating zynqmp binary package BOOT.BIN...


****** Xilinx Bootgen v2020.1
  **** Build date : May 26 2020-14:07:15
    ** Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.


[INFO]   : Bootimage generated successfully

INFO: Binary is ready.
WARNING: Unable to access the TFTPBOOT folder /tftpboot!!!
WARNING: Skip file copy to TFTPBOOT folder!!!
alinx2020@ubuntu:~/peta_prj/linxPsBase/petalinux$
```

3) In the current directory, you can see the newly generated BOOT.bin file



Part 1.4: Run Linux

1) Format the SD card in FAT32 format
2) Copy the files BOOT.bin, boot.scr and image.ub to the SD card

3) SD card inserted into the board
4) The board is set to SD card boot mode
5) Open the serial terminal and power on the board

```
[   16.973660] random: crng init done
Public key portion is:
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCoRAG6wigkCE6nABxQJiTlwhGKluh5Kg
hnMGOD9LQ2xDvpvjFaSXJZPnOu61xpOTTZlWCTsRwpneMFPRCsaVlv9/TCek+zfaK0XLVk
piOF root@petalinux
Fingerprint: sha1!! 66:fd:50:23:04:b6:f2:9b:d7:49:63:e8:9c:bc:07:7a:48
dropbear.
Starting internet superserver: inetd.
Starting syslogd/klogd: done
Starting tcf-agent: OK

PetaLinux 2020.1 petalinux /dev/ttyPS0

petalinux login:
```

6) Log in with the user name root and password root to enter the system. Because the root file system is the default ram type, the file system cannot save files and configurations.

```
[   16.973660] random: crng init done
Public key portion is:
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCoRAG6wigkCE6nABxQJiTlwhGKluh5K
hnMGOD9LQ2xDvpvjFaSXJZPnOu61xpOTTZlWCTsRwpneMFPRCsaVlv9/TCek+zfaK0XLV
piOF root@petalinux
Fingerprint: sha1!! 66:fd:50:23:04:b6:f2:9b:d7:49:63:e8:9c:bc:07:7a:4
dropbear.
Starting internet superserver: inetd.
Starting syslogd/klogd: done
Starting tcf-agent: OK

PetaLinux 2020.1 petalinux /dev/ttyPS0

petalinux login: root
Password:
root@petalinux:~#
```

Part 1.5: Petalinux Use Process Summary

1) Petalinux working environment

✓ Petalinux software installation

✓ "/bin/sh" is bash, the system default may be dash, need to be modified

✓ Cannot create petalinux project in shared folder

✓ Set petalinux environment variables

$ source <path-to-installed-PetaLinux>/settings.sh

2) Create Petalinux project

petalinux-create --type project --template <CPU_TYPE> --name <PROJECT_NAME>

<CPU_TYPE> can be: zynqMP, zynq, microblaze

<PROJECT_NAME> is the project name

3) Import hardware information

Use the cd command to enter the petalinux project directory

Use petalinux-config --get-hw-description command to point to the directory containing *.xsa

4) Configure petalinux

petalinux-config configures the entire petalinux project. If you need to configure uboot, kernel, rootfs, etc., you need to bring the following parameters.

petalinux-config -c u-boot

petalinux-config -c kernel

petalinux-config -c rootfs

5) Compile and create system image

petalinux-build

6) Manually modify the petalinux configuration, these configuration files can be directly copied and used in other projects

Various files that can be manually configured are stored in the

<plnx-proj-root>/project-spec/meta-user directory

recipes-bsp/u-boot/files/platform-top.h modify u-boot configuration

recipes-bsp/device-tree/files/system-user.dtsi,

ecipes-bsp/device-tree/files/pl-custom.dtsi can modify and add device tree

recipes-kernel/linux/linux-xlnx/ saves the kernel configuration

project-spec/meta-user/recipes-apps save app

7) Clean up petalinux

If there are various problems with petalinux compilation, especially if we cannot compile normally after modifying some configurations, or keep getting stuck, we need to use the command petalinux-build -x mrproper -f to clean up the previous compilation results.

Part 1.6: Save a Project as a Template Project

Run the “petalinux-build -x mrproper -f" command to clean up the existing project

Clean up the folder according to the following table

Folder or File	Can be deleted	other
<base folder>	*.xsa .gitignore	config.project must be retained
.petalinux/	Delete everything except "metadata"	
.xil/	delete all	
components/	Should be an empty folder	Empty folder
project-spec/	yocto-layer.log	"attributes" must be retained
project-spec/configs/	config.old, rootfs_config.old	"config" and "rootfs_config" must be retained
project-spec/hw-description/	Delete everything except "metadata"	
project-spec/meta-plnx-generated	delete all	
project-spec/meta-user		Keep all

After saving the template, put the xsa file of the new project in the project directory, and then use the command petalinux-config --get-hw-description to update the hardware information. But if the size of ddr changes, you need to manually modify the definition in the project-spec/configs/config file.

```
#
# Memory Settings
#
CONFIG_SUBSYSTEM_MEMORY_PS7_DDR_0_BANKLESS_SELECT=y
# CONFIG_SUBSYSTEM_MEMORY_SIMPLE_SELECT is not set
# CONFIG_SUBSYSTEM_MEMORY_MANUAL_SELECT is not set
CONFIG_SUBSYSTEM_MEMORY_PS7_DDR_0_BANKLESS_BASEADDR=0x0
CONFIG_SUBSYSTEM_MEMORY_PS7_DDR_0_BANKLESS_SIZE=0x40000000
CONFIG_SUBSYSTEM_MEMORY_PS7_DDR_0_BANKLESS_KERNEL_BASEADDR=0x0
CONFIG_SUBSYSTEM_MEMORY_PS7_DDR_0_BANKLESS_U__BOOT_TEXTBASE_OFFSET=0x100000
CONFIG_SUBSYSTEM_MEMORY_IP_NAME="PS7_DDR_0"

#
```

Part 1.7: Some Official Optimization Suggestions

To reduce the build time by disabling the network state feeds, de-select the

petalinux-config → Yocto Settings → Enable Network state feeds

Part 2: Program hello world

Part 2.1: Code

1) Create an application directory app_helloworld

2) In the app_helloworld folder, create a main.c document

3) Edit the main.c document

```
#include <stdio.h>


int main(void)
{
	printf("hello world\r\n");
	return 0;
}
```

Part 2.2: Compile the Host Program

4) Compile program

```
gcc -o helloworld main.c
```

5) Run the program directly

```
./helloworld
```

6) You can see the printed message "hello world", indicating that the program has been executed correctly on the computer

Part 2.3: Cross Compile Arm Program

The program ultimately needs to run on the FPGA board. We need to select the arm compiler to compile the program for the board. Here, we choose the compiler that comes with Petalinux installation

1) Run the following command to add the arm compiler to the environment variable. The environment variable is only valid in the current terminal, and the following commands must be completed in the current terminal.

```
source    /tools/Xilinx/Vivado/2020.1/settings64.sh
```

2) Cross-compiler

```
aarch64-linux-gnu-gcc -o helloworld main.c
```

3) Generate executable program helloworld on the board
4) Use the linux system customized by petalinux in the previous chapter, and copy the executable program helloworld to the SD card

5) Set the board to SD card startup and power on
6) Enter linux system
7) Enter the directory where helloworld is located (sd-mmcblk1p1 represents the first partition of the sd card, sd-mmcblk0p1

represents the first partition of emmc)

```
cd    /media/sd-mmcblk1p1
```

8) Run the program

```
./helloworld
```

9) You can see the printed message "hello world", indicating that the program has been executed correctly on the board

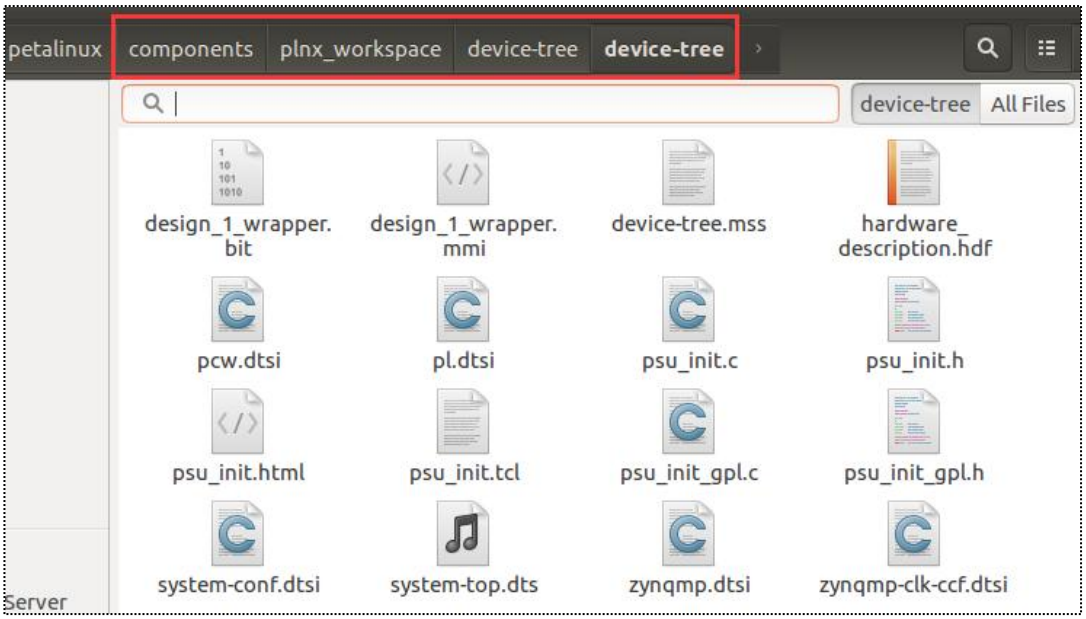
Part 3: gpio Control LED

To write some simple applications under linux, we will find it very simple, such as opening a file, writing a piece of data, and closing the handle that was just opened. This completes the entire writing process. As for the location of this file, whether it is flash, memory, or hard disk, it's not that we don't care about it, but the code just now is fully compatible. But don't you think that everything becomes simpler under linux. This is because we didn't consider this: open a certain file just mentioned, what is this file, and how did it come from?

Part 3.1: Device tree Automatically Generated by Petalinux

In the linux kernel code, many drivers are already included. We need to associate these drivers with the hardware on the FPGA board to generate usable devices under the system.

Here, we can use the device tree to associate our hardware and drivers. Before, when we customized the Linux system, petalinux helped us automatically generate the device tree. The generated device tree directory is as follows:

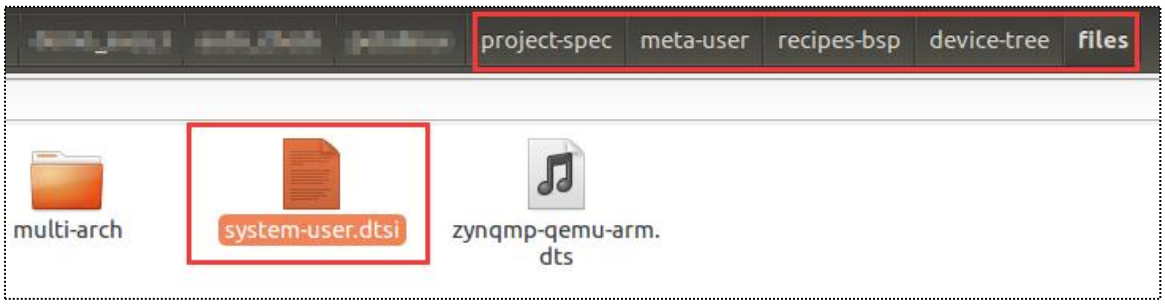


Open the zynqmp.dtsi file, and the gpio device tree on the ps side of the device is described as follows:

```
gpio: gpio@ff0a0000 {
                compatible = "xlnx,zynqmp-gpio-1.0";
                status = "disabled";
                #gpio-cells = <0x2>;
                interrupt-parent = <&gic>;
                interrupts = <0 16 4>;
                interrupt-controller;
                #interrupt-cells = <2>;
                reg = <0x0 0xff0a0000 0x0 0x1000>;
                gpio-controller;
                power-domains = <&pd_gpio>;
        };
```

Part 3.2: Petalinux Project Users Modify the Device Tree

In the directory just now, any modification of the device tree by the user is invalid. The user needs to use the following files to modify or add his own device tree file.



Because the GPIO device tree on the ps side has been correctly generated by petalinux, we don't need to make any changes here.

Part 3.3: Project creation

1) As in the previous Helloworld chapter, create a gpio_test.c function and modify the program code as follows:

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <unistd.h>
#include <fcntl.h>
#include <stdbool.h>

#define PS_LED_BASE          338
#define PS_LED_INDEX  40
#define PS_LED_ADDR          (PS_LED_BASE+PS_LED_INDEX)

void led_ctl_init(void)
{
    int lGpioFd;
    char lCache[100];

    lGpioFd = open("/sys/class/gpio/export", O_WRONLY);
    if(lGpioFd < 0)
    {
        printf("open export fail\n");
        return;
    }
    int len = sprintf(lCache, "%d", PS_LED_ADDR);
    write(lGpioFd, lCache, len+1);
    close(lGpioFd);

    sprintf(lCache, "/sys/class/gpio/gpio%d/direction", PS_LED_ADDR);
    lGpioFd = open(lCache, O_RDWR);
    if (lGpioFd < 0)
    {
        printf("open direction fail\n");
        return;
    }
    write(lGpioFd, "out", 4);
    close(lGpioFd);
}

void led_ctl_on(bool isON)
{
```

```
    int lGpioFd;
    char lCache[100];

    sprintf(lCache, "/sys/class/gpio/gpio%d/value", PS_LED_ADDR);
    lGpioFd = open(lCache, O_RDWR);
    if (lGpioFd < 0)
    {
        printf("open value fail\n");
        return;
    }
    if(isON)
    {
        write(lGpioFd, "1", 2);
    }
    else
    {
        write(lGpioFd, "0", 2);
    }
    close(lGpioFd);
}

int main()
{
    led_ctl_init();
    printf("================start led ctrl==============\n");

    while(1)
    {
        led_ctl_on(true);
        printf("on\r\n");
        usleep(500000);

        led_ctl_on(false);
        printf("off\r\n");
        usleep(500000);
    }
```

```
    return 0;
}
```

In the code, the main function of led_ctl_init is to export gpio pins and convert them to output pins. The led_ctl_on function controls the high and low levels of the pins to realize the on and off of the led light.

2) Compile code

```
alinx2020@ubuntu:~/peta_prj/apps/gpio_test$ aarch64-linux-gnu-gcc -o gpio_test g
pio_test.c
alinx2020@ubuntu:~/peta_prj/apps/gpio_test$
```

Part 3.4: Run The Program

1) Boot and enter the FPGA development board linux system
2) View the device nodes related to gpio as follows

```
ls    /sys/class/gpio/
```

```
root@petalinux:~# ls   /sys/class/gpio/
export       gpiochip338  unexport
root@petalinux:~#
```

3) Here, gpiochipxxx refers to a group of gpio, the number of pins in it and the specific pin pointed to, if the user does not make other changes, it is determined by the vivado project.
4) Check the pin label information to determine the group of the pin

```
root@petalinux:~# ls /sys/class/gpio/
export       gpiochip338  unexport
root@petalinux:~# cat /sys/class/gpio/gpiochip338/label
zynqmp_gpio
root@petalinux:~#
```

5) Through the schematic diagram, we know that the PS_led pin is MIO40.
6) Use the number 338 of the group where PS_led is located, plus

the offset value of the PS_led pin in this group of 40, which is the pin we actually want to operate, as the macro definition in the code:

```
#define PS_LED_BASE        338
#define PS_LED_INDEX  40
#define PS_LED_ADDR        (PS_LED_BASE+PS_LED_INDEX)
```

7) Use nfs mount method, and run the program, the led light flashes once per second

```
root@petalinux:/mnt# ./gpio_test
================start led ctrl==============
on
off
on
off
on
```

Part 4: Add Boot Scripts and User Files

In the previous tutorial, we used petalinux to create Linux. The default root file system is ramdisk type, and the file system is read-only. Any modification to this directory will be lost when the system restarts. This chapter describes how to add the user's application and data files, and start the user's application automatically after booting.

Part 4.1:Ready to work

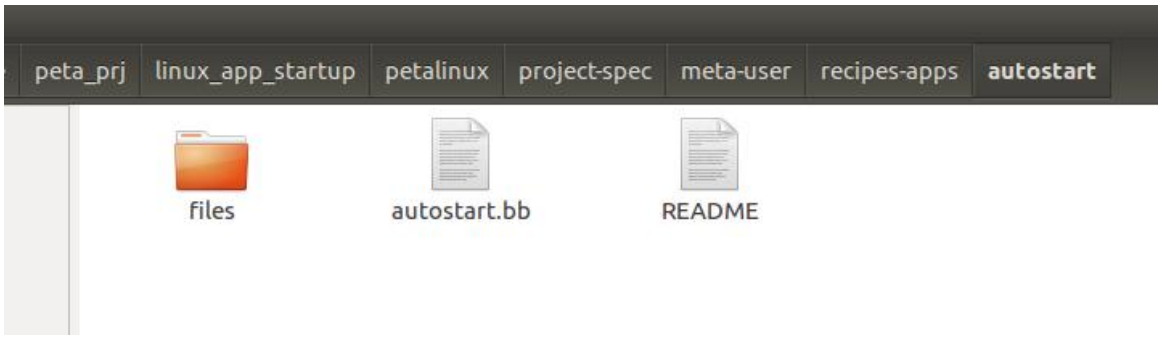
Select the petalinux project created earlier, and enter the directory petalinux in the control terminal

Part 4.2: Self-Start Script

1) Enter the following command to create an application template and create an application named "autostart"

```
petalinux-create -t apps --template install --name autostart --enable
```

2) Enter the template directory



3) Open the autostart file in the files folder, this is a script file, print the string "Hello Petalinux World", we change it to "Hello ALINX World", and then add a paragraph to determine whether there is an autostart.sh script in the first partition of the SD, If it exists, run it, and the user can modify it according to actual needs, such as

launching an application or other initialization operations.

```
echo "Hello ALINX World"
USERHOOK_SD0= /media/sd-mmcblk0p1/autostart.sh
USERHOOK_SD1= /media/sd-mmcblk1p1/autostart.sh
if [ -f $USERHOOK_SD0 ]; then
    sh $USERHOOK_SD0 &
fi
if [ -f $USERHOOK_SD1 ]; then
    sh $USERHOOK_SD1 &
fi
```

4) The content of the edited file autostart.bb is as follows, the main function is to copy autostart to the root file system init.d directory, and then add 0755 permissions. RDEPENDS_${PN}_append += "bash" means this depends on bash

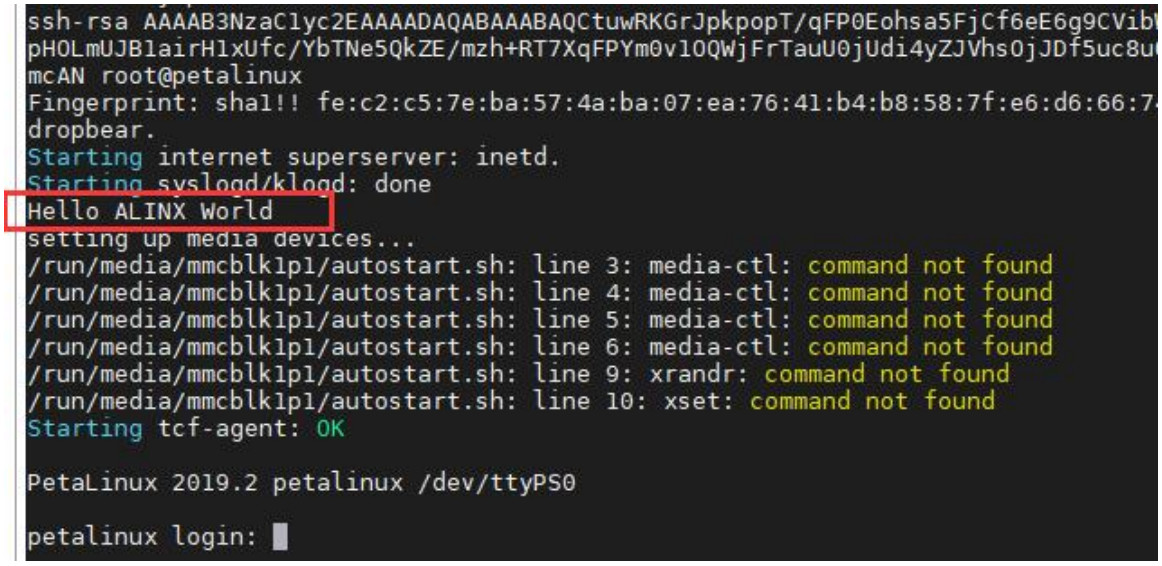
```
#
# This file is the autostart recipe.
#

SUMMARY = "Simple autostart application"
SECTION = "PETALINUX/apps"
LICENSE = "MIT"
LIC_FILES_CHKSUM =
"file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC_URI = "file://autostart \
    "

S = "${WORKDIR}"

inherit update-rc.d

INITSCRIPT_NAME = "autostart"
```

```
INITSCRIPT_PARAMS = "start 99 5 ."


do_install() {
        install -d ${D}${sysconfdir}/init.d
        install -m 0755 ${S}/autostart ${D}${sysconfdir}/init.d/autostart
}


RDEPENDS_${PN}_append += "bash"
```

5) Recompile the program

```
petalinux-build
```

6) Package the application and start it. Here you can see that the added script has been started. You can also put an autostart.sh script in the first partition of the sd card. You can enjoy what you want to add in the script.



Part 5: SD Card Root File System

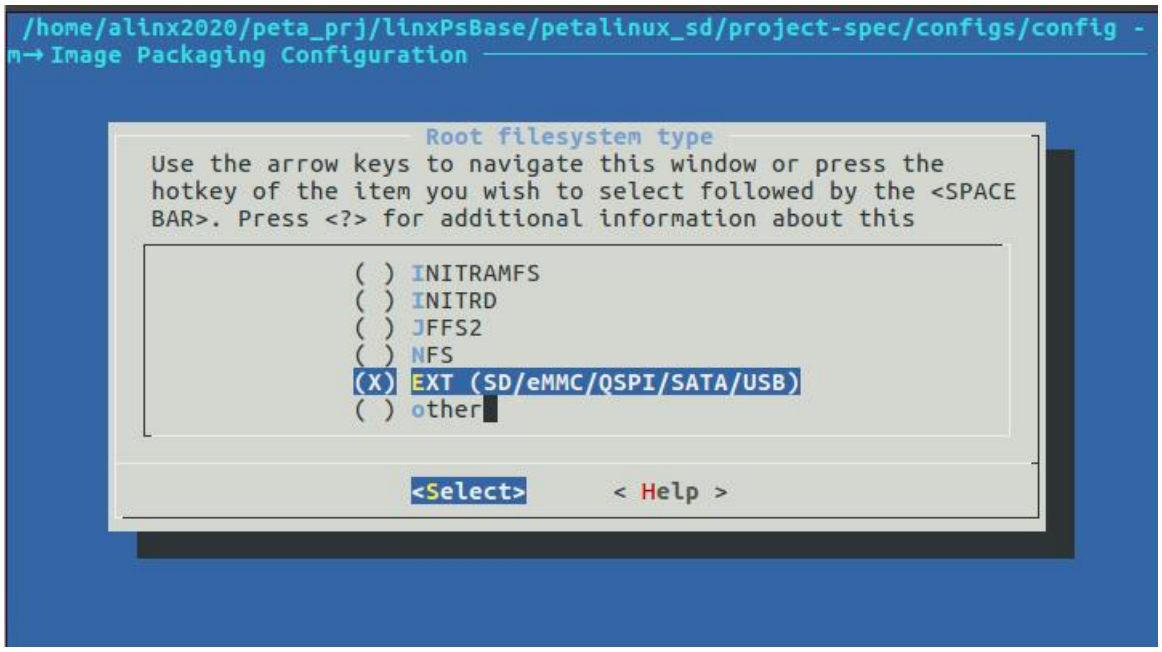
The default root file system type of Petalinux is INITRAMFS, which cannot save files and configurations. This chapter introduces how to use the SD card as the root file system

Part 5.1: Petalinux Configuration

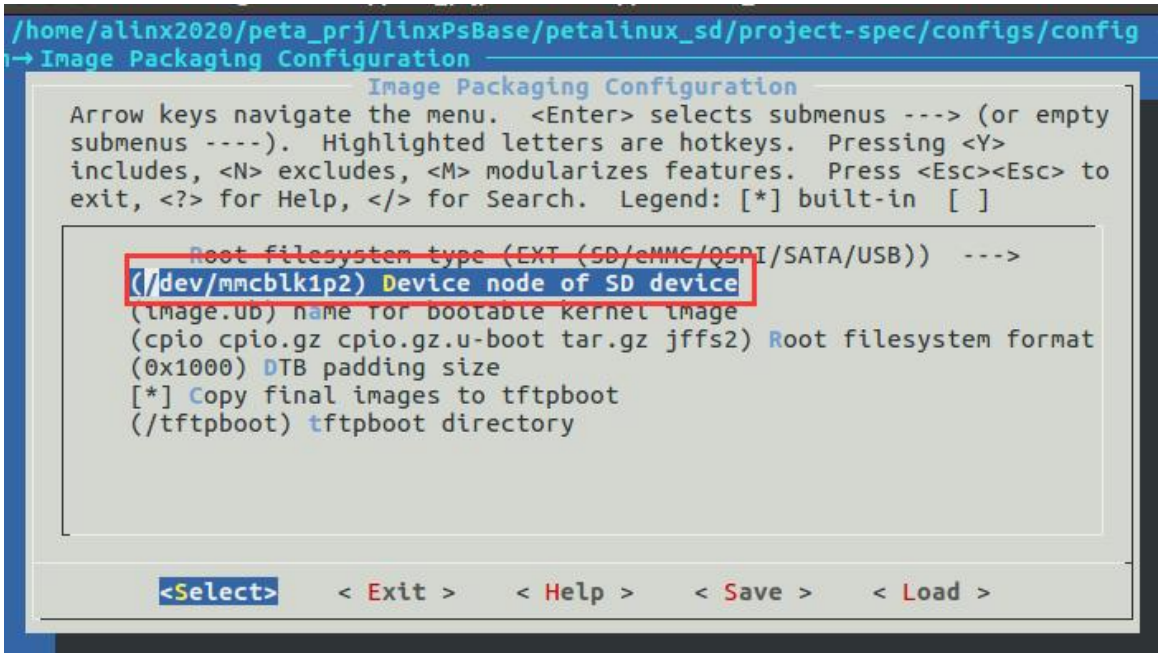
1) Run the command to configure Petalinux

```
petalinux-config
```

2) Image Packaging Configuration -> Root filesystem type, select EXT (SD/eMMC/QSPI/SATA/USB)



3) The root file system path selection is very important. If you fill in the wrong file and you cannot start Linux, mmcblk represents the sd card and emmc, mmcblk0p2 represents the second partition of the first mmc device, if the emmc and sd cards exist at the same time, the sd is mmcblk1, so fill in here /dev/mmcblk1p2



Part 5.2: Make SD Card File System

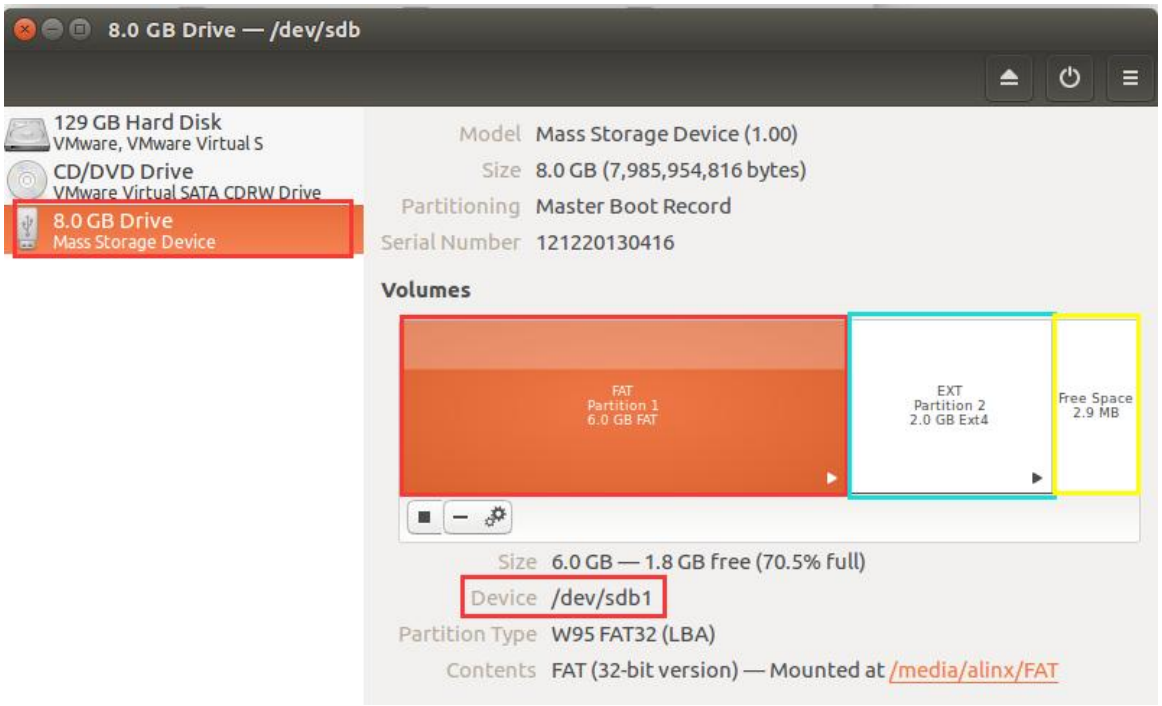
Creating an SD card file system will cause the content in the SD card to be lost, please make a backup first.

Part 5.2.1: SD card modify partition

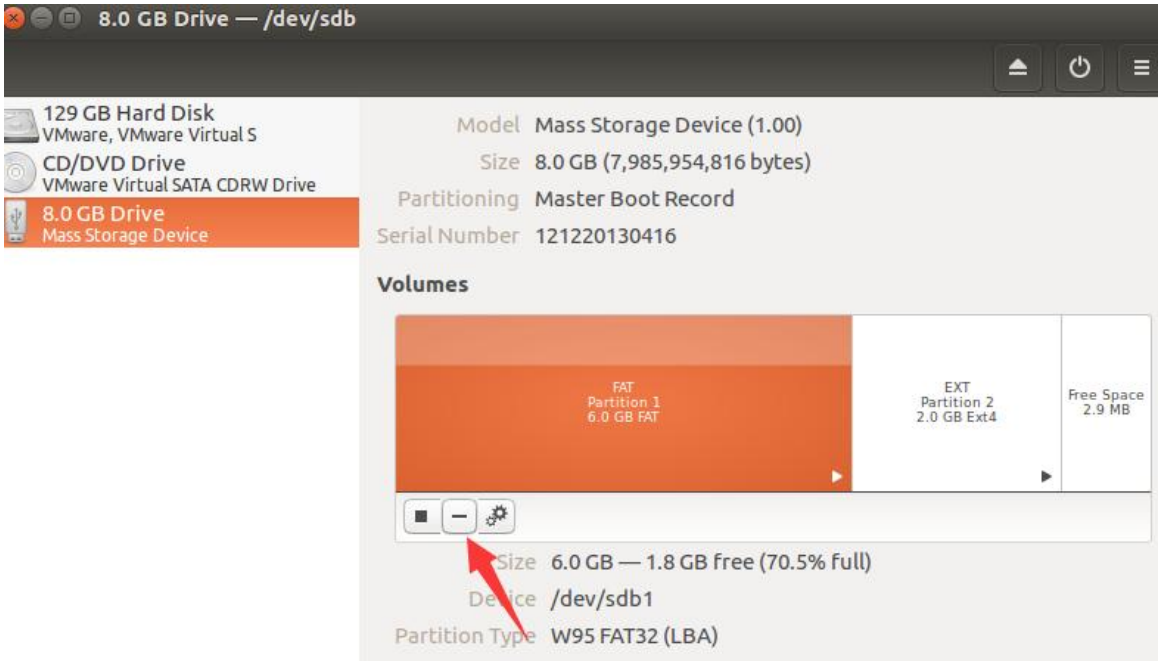
1) Insert the sd card of the FPGA development board into the card reader, and then insert it into the USB port of the computer
2) Connect to the virtual machine Linux

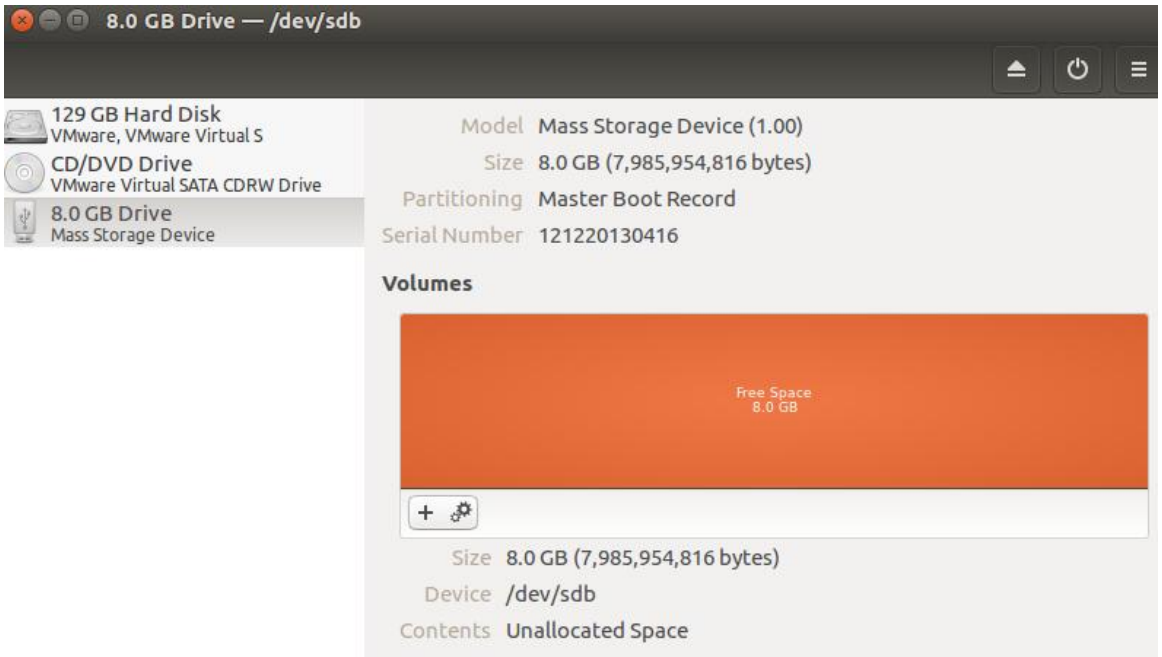
3) In the search path of ubuntu, enter disk, and the Disks icon will appear

4) Click the Disks icon with the mouse, and the "Disks" dialog box appears. In this experiment, the SD card has been divided into 2 partitions, one is named FAT and the other is named EXT. Here we need to repartition.

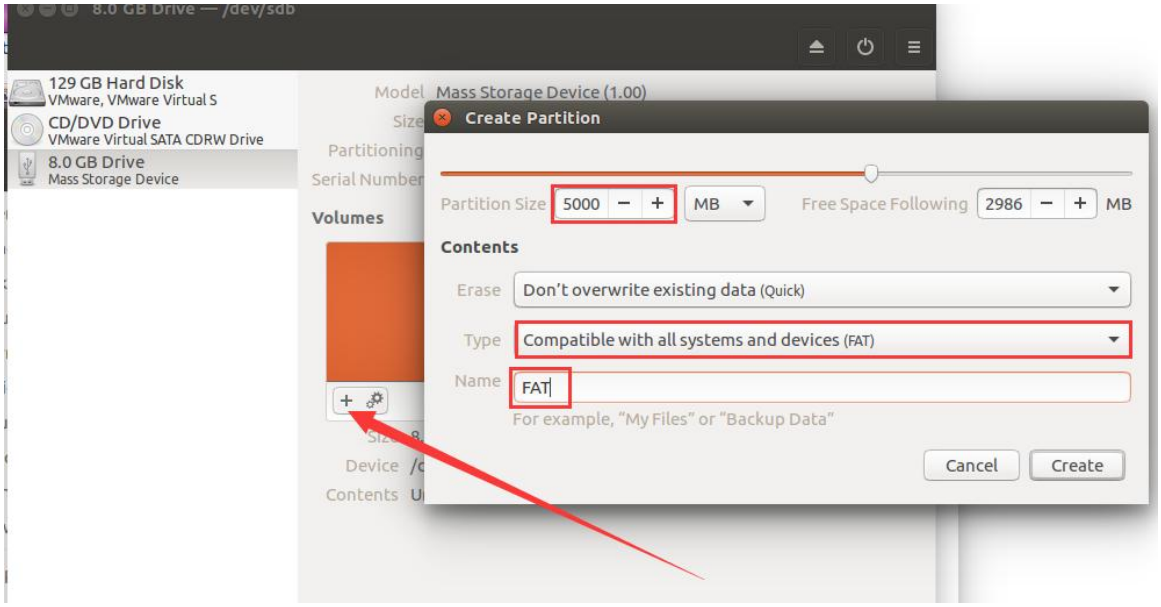


5) Select the delete partition icon under each partition to delete all partitions.

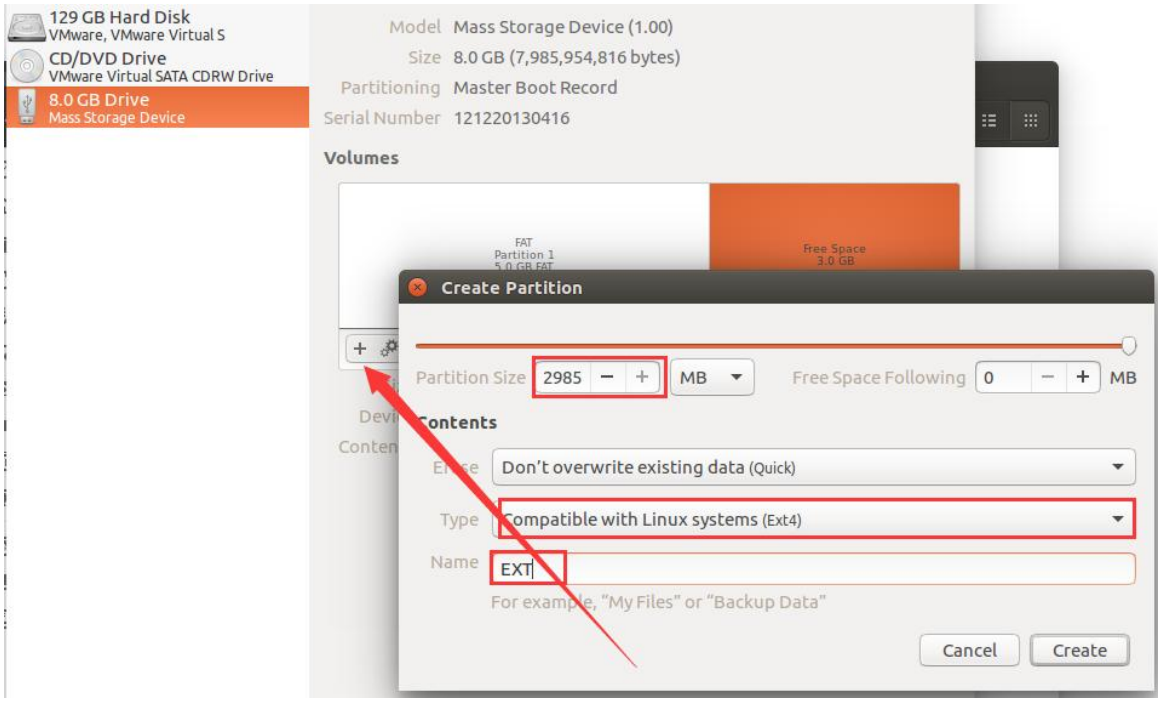




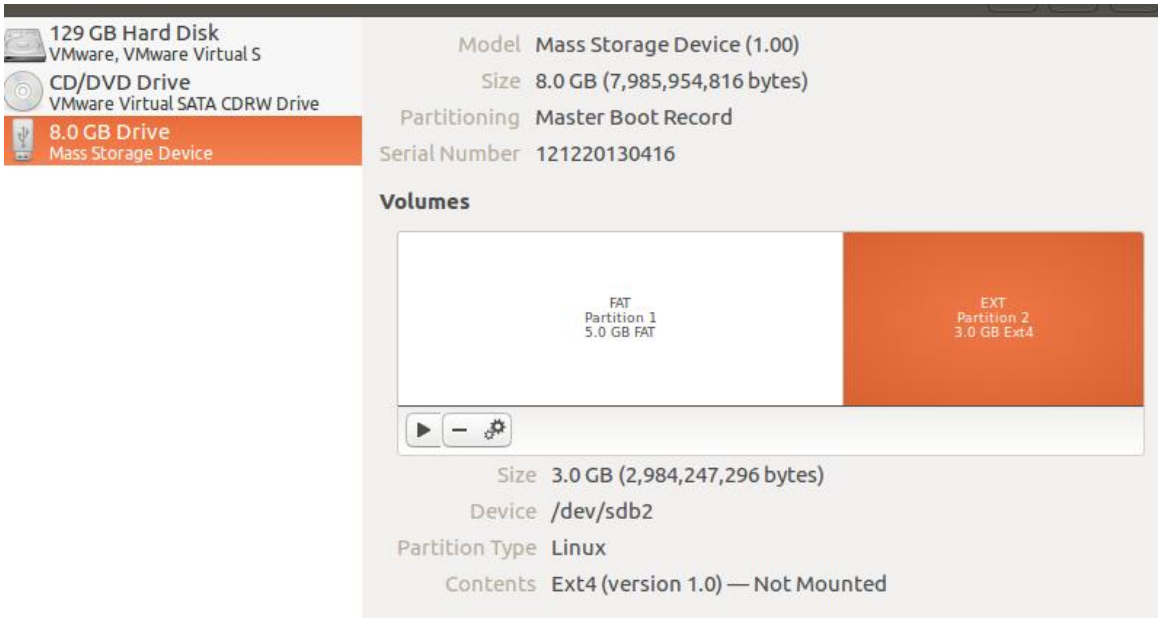
6) Click the icon of adding partition to add the first partition. In this experiment, fill in 5000MB, the format is FAT, which is used to store the boot file BOOT.bin of ZYNQ, the kernel file, and the device tree. The name is FAT



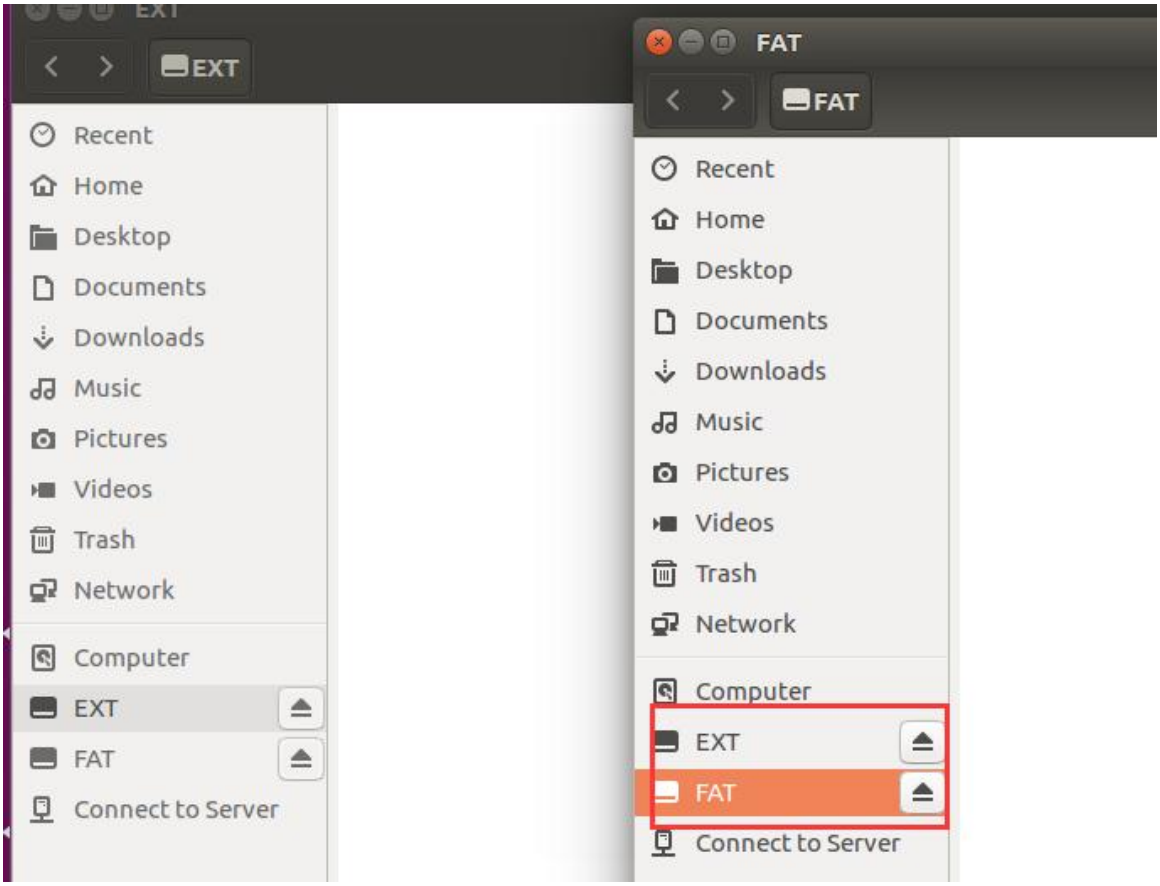
7) Create a second partition to store the root file system, the format is EXT4, and the name is EXT



8) After the partition is created, reinsert the SD card



9) The system will automatically mount the partition and pop up a window



Part 5.2.2: Unzip the root file system to the EXT4 partition of the SD card

1) Enter the decompression command to decompress the file system. It may take a few minutes to decompress. Please note that the decompression method must be consistent with the tutorial, (/media/alinx/EXT is the path of the EXT4 partition of the SD card, which may be different, please modify it according to your actual situation).

```
sudo tar xzvf <petalinux path>/images/linux/rootfs.tar.gz    -C    /media/alinx/EXT
```

2) Enter "sudo rsync" in the command window, the synchronization may take more than ten minutes.
3) Copy BOOT.bin and iamge.ub to the FAT32 partition (first partition) of the SD card, set the FPGA development board to start in SD mode, and start the FPGA development board.

Part 6: QT and OPENCV Cross-Compilation Environment

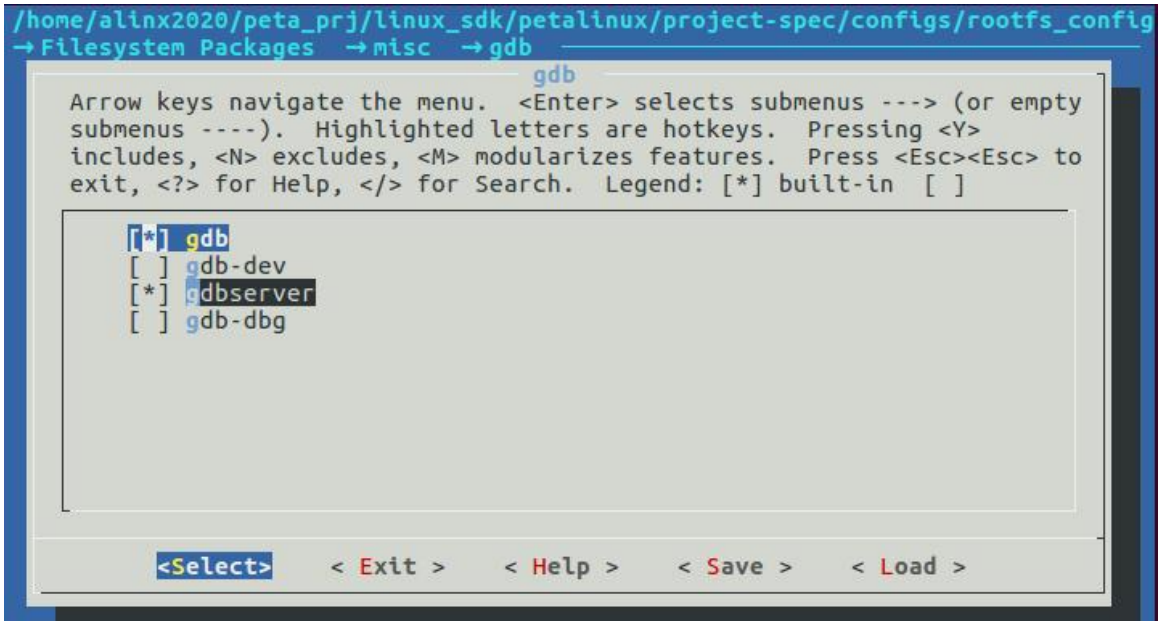
Qt is a cross-platform C++ graphical user interface application development framework. It can be simply understood as a library for graphics development. If we use the Qt library to develop an application with an interface for the board, we need to cross-compile the Qt library. OpenCV is a well-known library in video processing. Many applications need to use the OpenCV library. In the past, the QT and OpenCV libraries had to download the code and compile it by themselves, which was very troublesome. This chapter explains how to use Petalinux to configure the development environment of OpenCV and QT, which once again reflects the advantages of the Petalinux development environment, and only simple configuration can solve very complex problems.

Part 6.1: Configure the Root File System

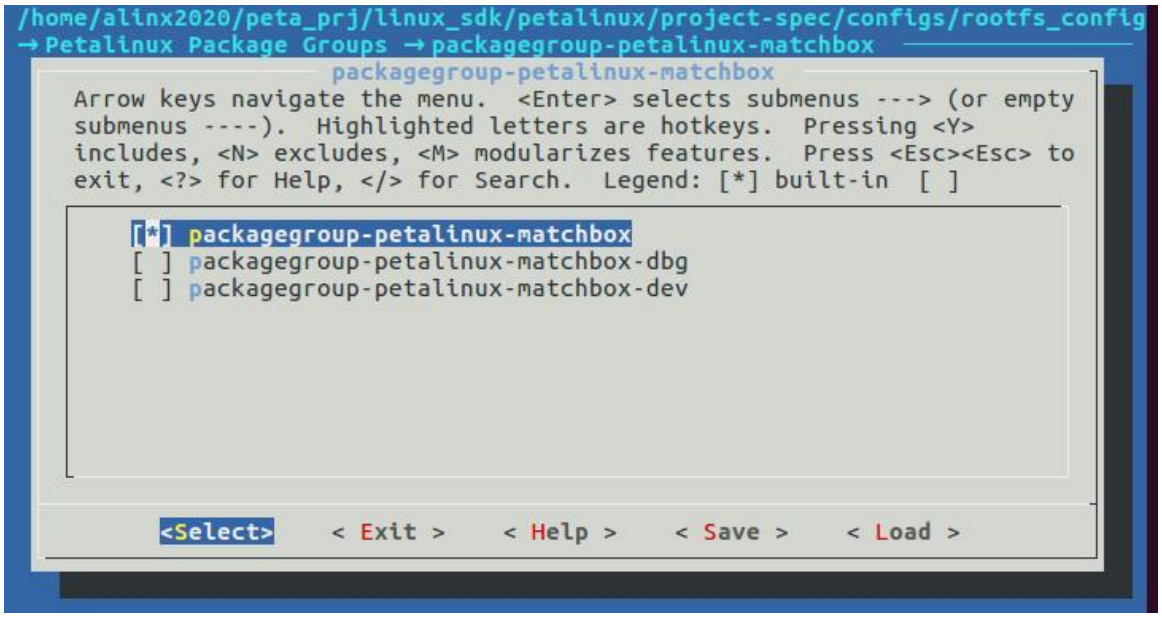
1) Run petalinux command to configure rootfs

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

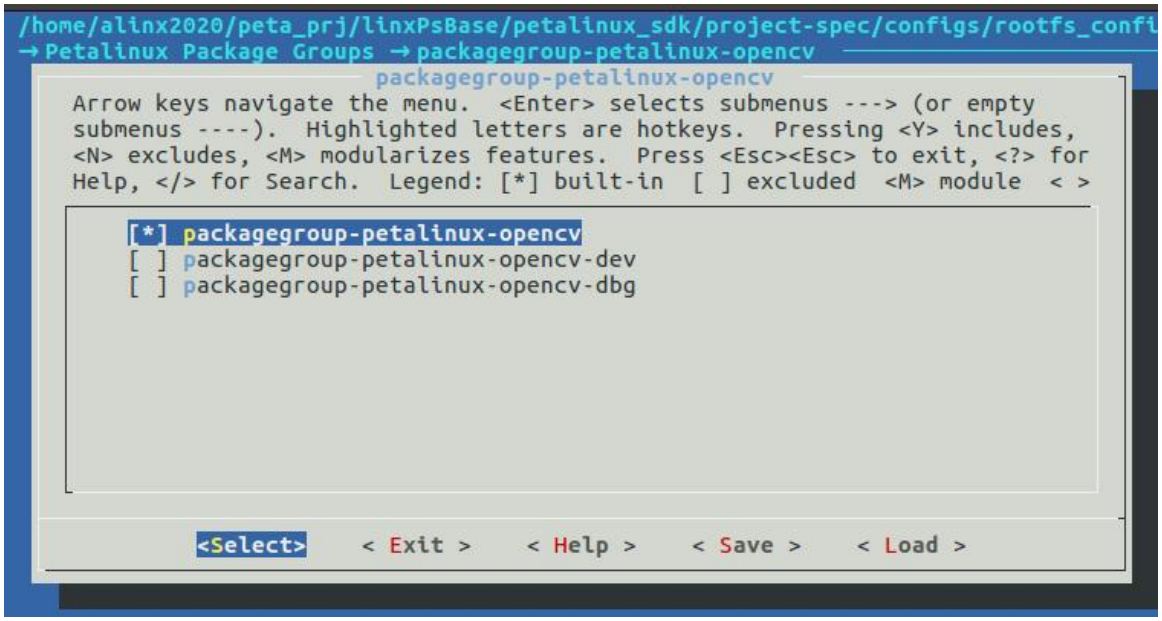
2) Check Filesystem Packages → misc → gdb and gdbserver, gdb can be used to debug applications, and gdbserver can be used to debug applications remotely through the network.



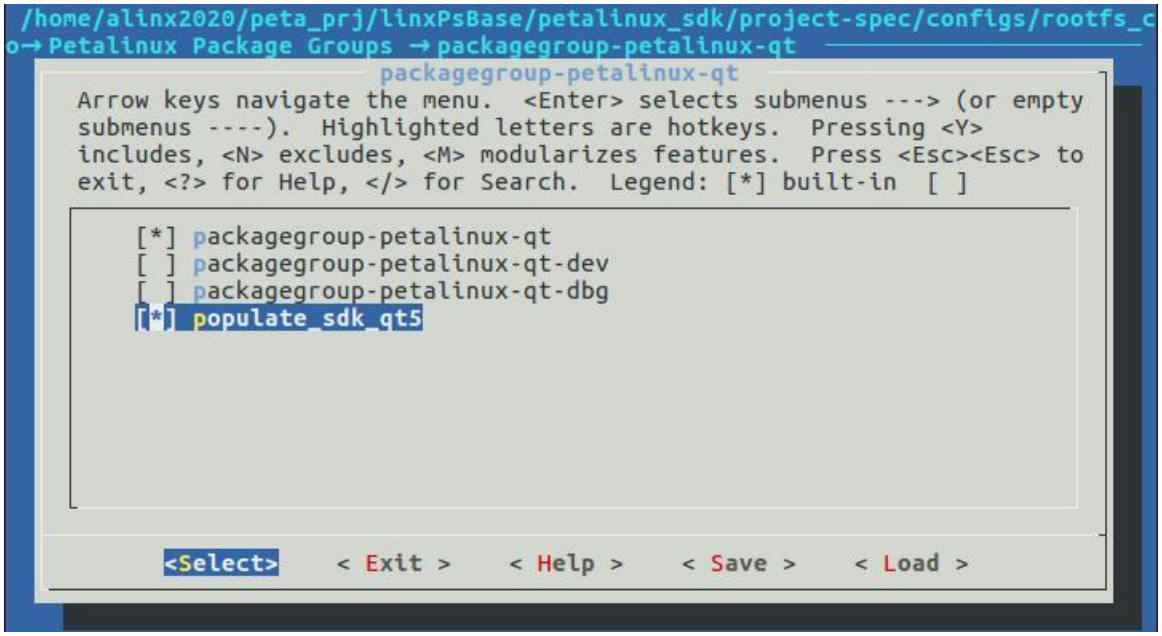
3) Check Petalinux Package Groups → packagegroup-petalinux-matchbox, configure matchbox desktop, matchbox is a simple Linux desktop management system.



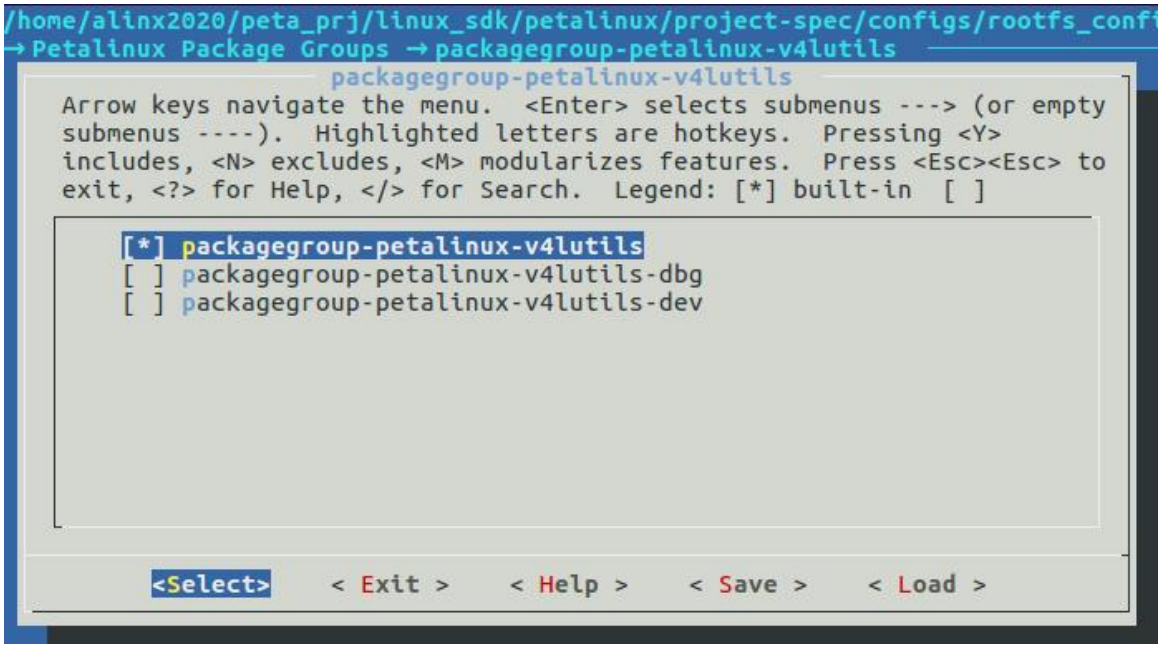
4) Check Petalinux Package Groups → packagegroup-petalinux-opencv to configure the opencv library. The following dev and dbg versions are not checked here and will not affect the operation. After checking, the file system will be very large.



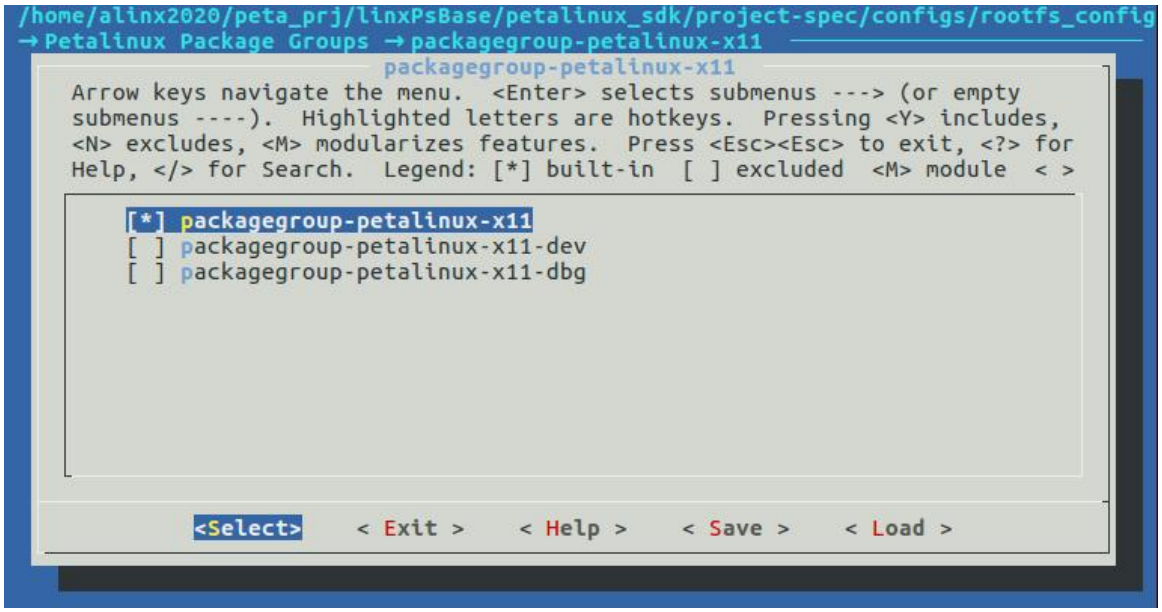
5) Check Petalinux Package Groups → packagegroup-petalinux-qt, configure the qt library



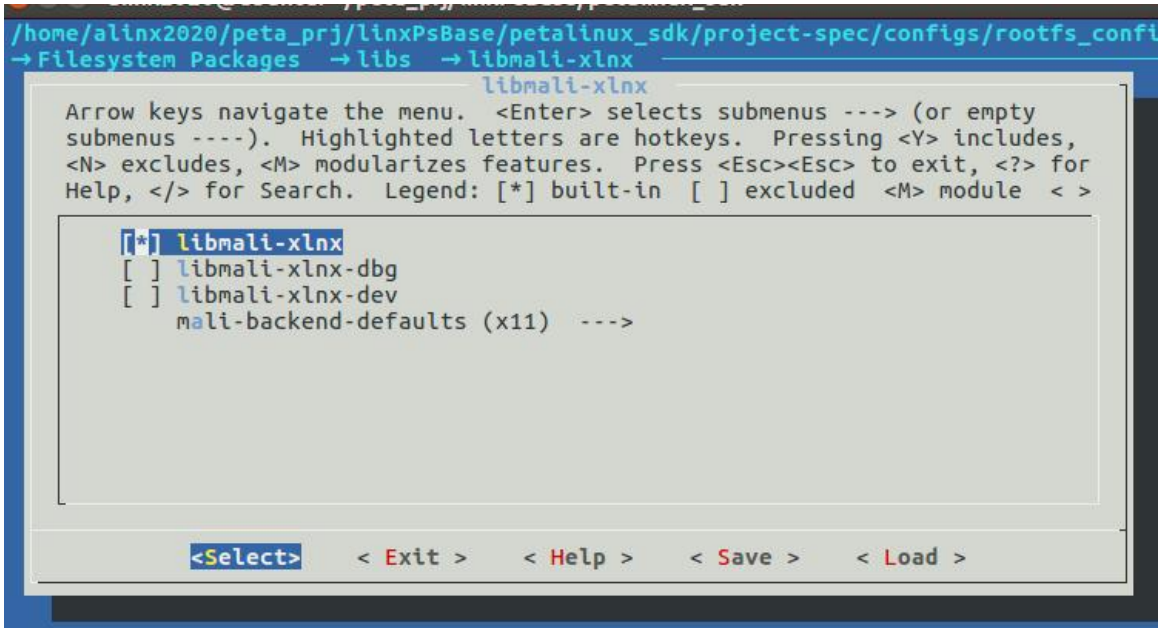
6) Check Petalinux Package Groups → packagegroup-petalinux-v4lutils to configure the v4l tool. This is mainly a video configuration tool under Linux, which may be used for video development.



7) Check Petalinux Package Groups → packagegroup-petalinux-x11 to configure the x11 library



8) Check Filesystem Packages → libs → libmali-xlnx to configure GPU library



9) Compile petalinux project

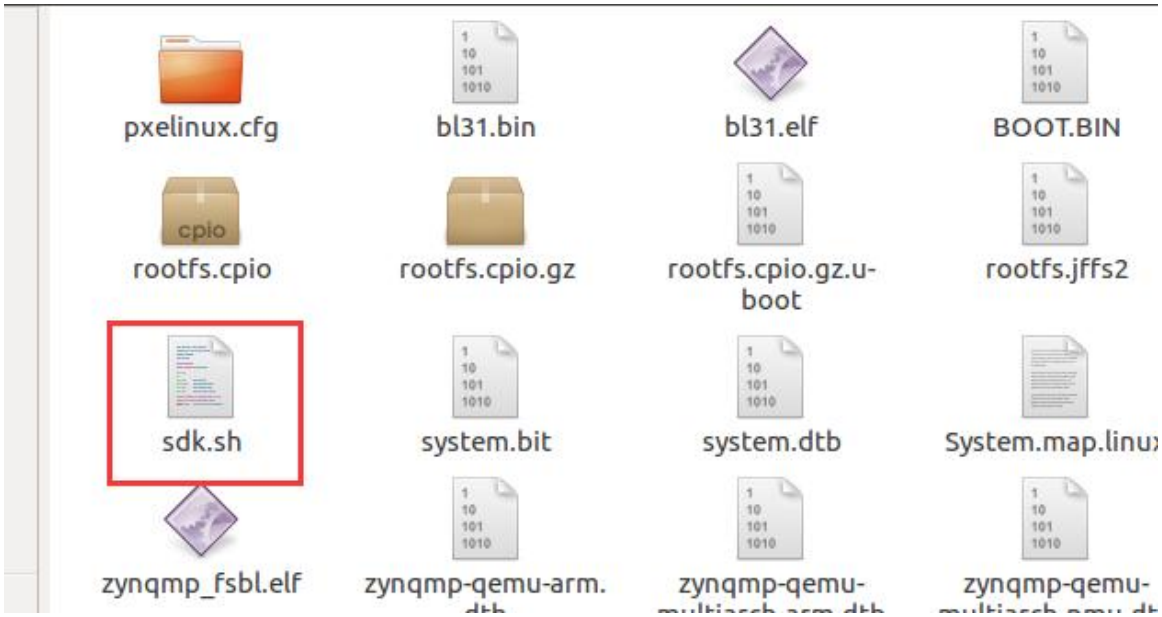
```
petalinux-build
```

10)Build SDK

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

Part 6.2: Install SDK

1) You can find the generated sdk.sh file in images/linux. This file is relatively large and is not a text file. It contains a cross-compiler, root file system, various libraries and header files. It is very useful. If you only do Linux application development , Only one sdk.sh file is needed, which can be separated from Petalinux and Vitis.



2) Run the installation sdk, it prompts us to install in /opt/petalinux/2020.1 by default, press Enter, and then enter Y to install to the default directory. After the installation is complete, it prompts

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

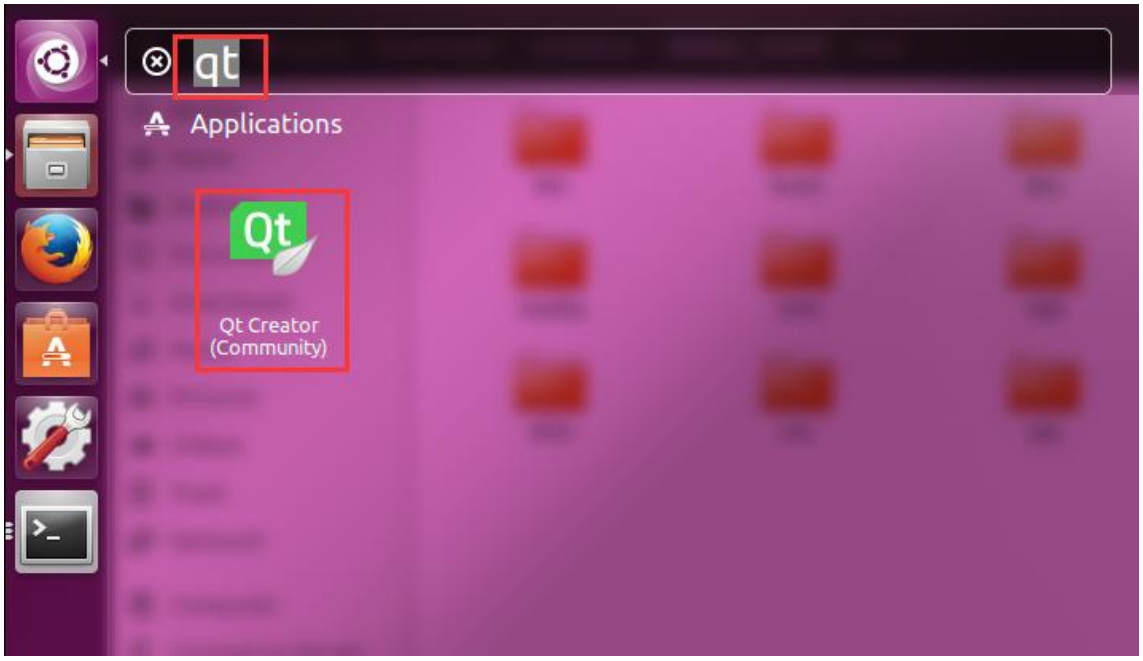
You can use the SDK

```
alinx2020@ubuntu:~/peta_prj/linux_sdk/images/linux$ ./sdk.sh
PetaLinux SDK installer version 2020.1
======================================
Enter target directory for SDK (default: /opt/petalinux/2020.1):
You are about to install the SDK to "/opt/petalinux/2020.1". Proceed [Y/n]? y
[sudo] password for alinx2020:
Extracting SDK.....................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.......done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the
environment setup script e.g.
 $ . /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
alinx2020@ubuntu:~/peta_prj/linux_sdk/images/linux$
```

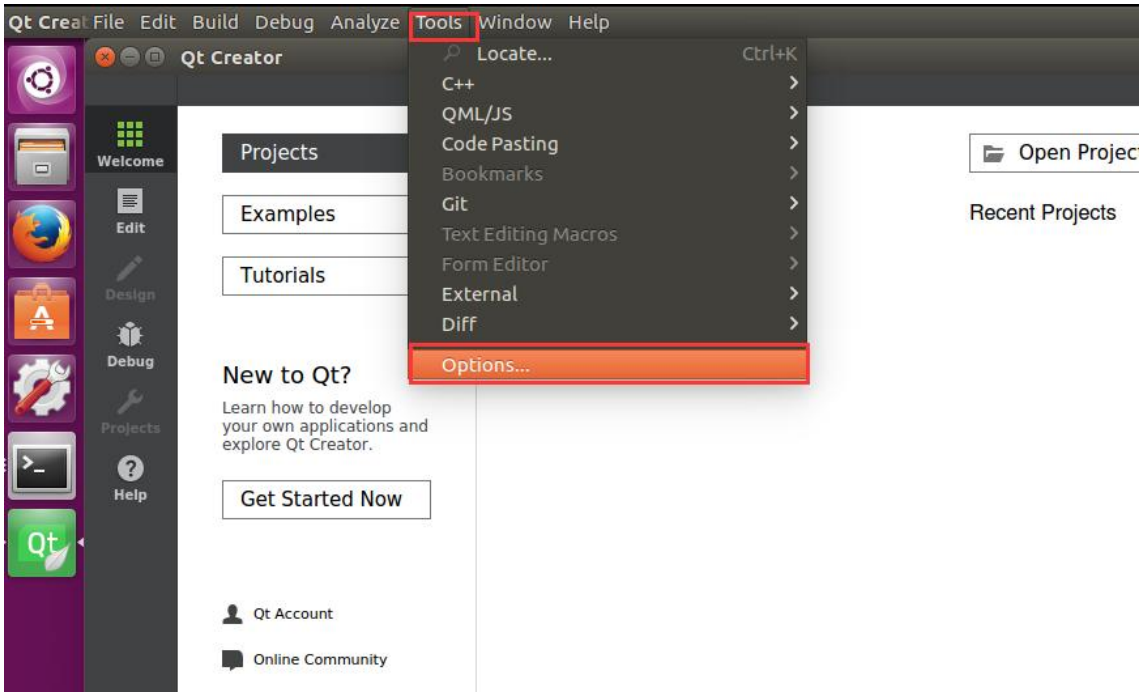
3) If it is configured as an SD card root file system, please refer to "SD Card Root File System Making" to make an SD card

Part 6.3: QT Creator Cross-compilation Environment Setting

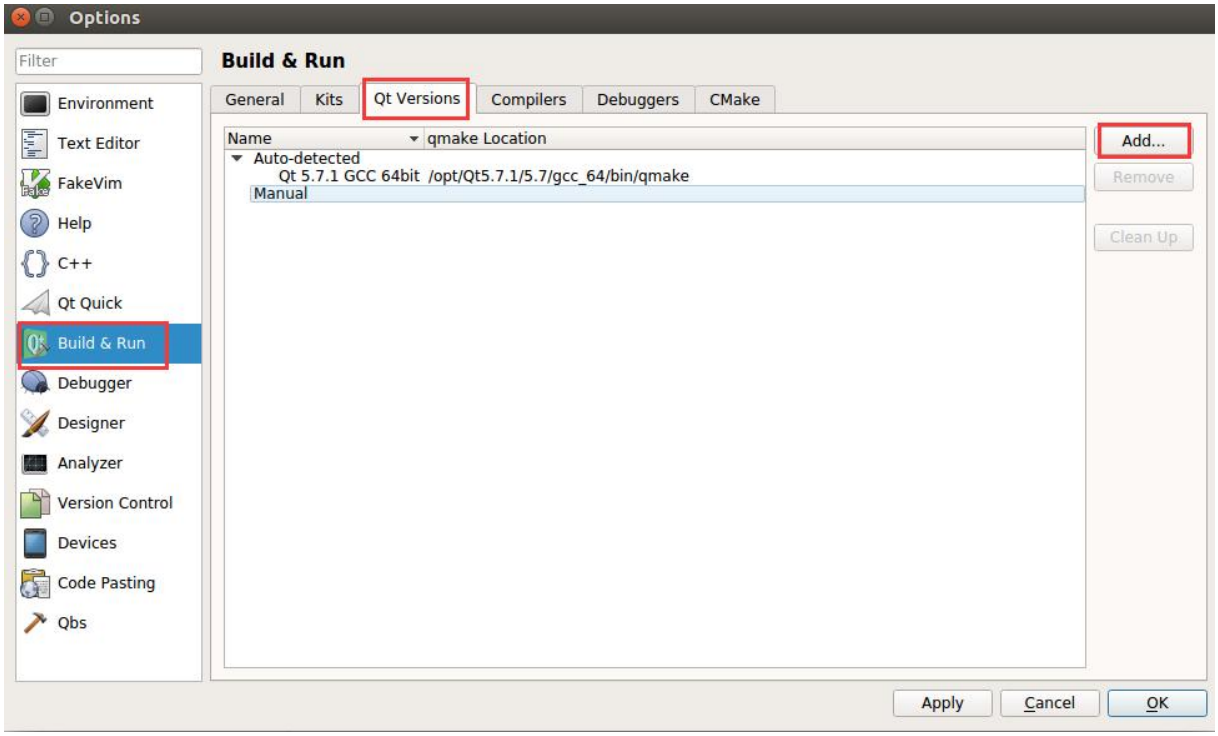
1) Run QT Creator, if you can't find the application shortcut, you can search for it



2) Click Tools -> Options



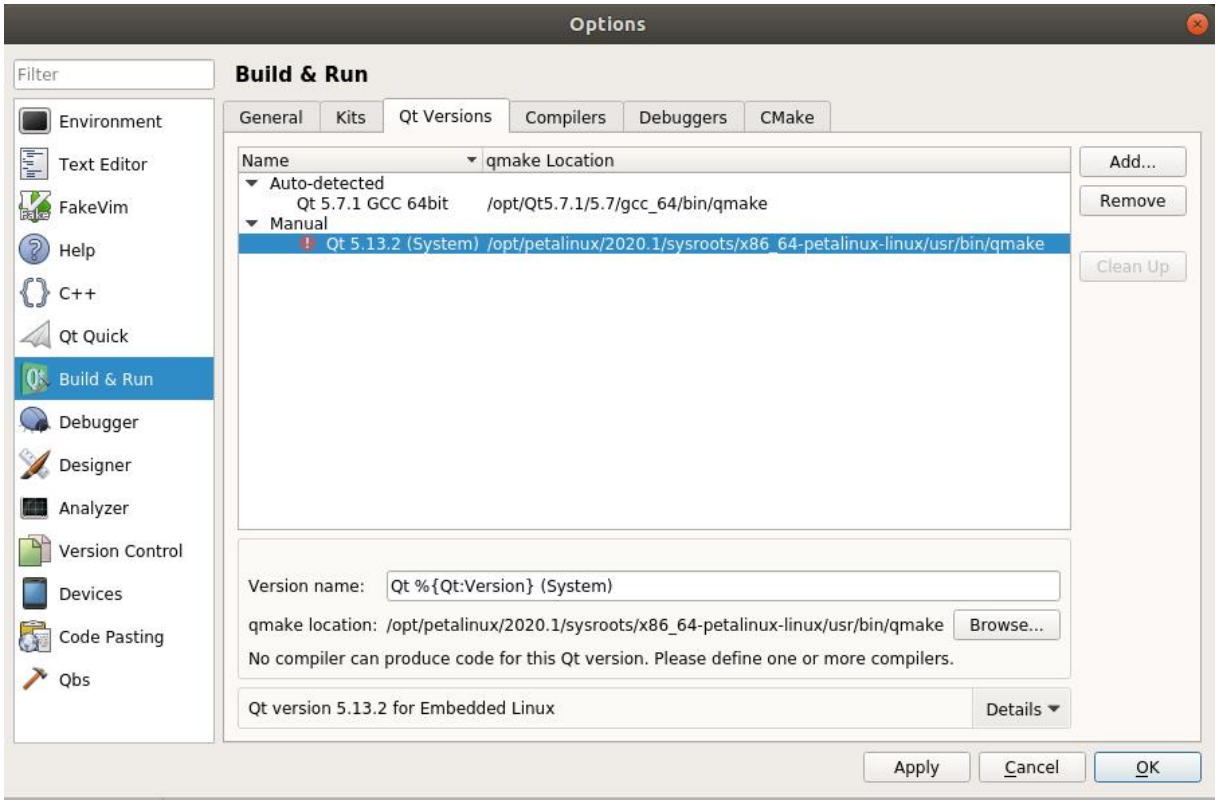
3) Click "Add.." on the Qt Versions page of the Build & Run option.



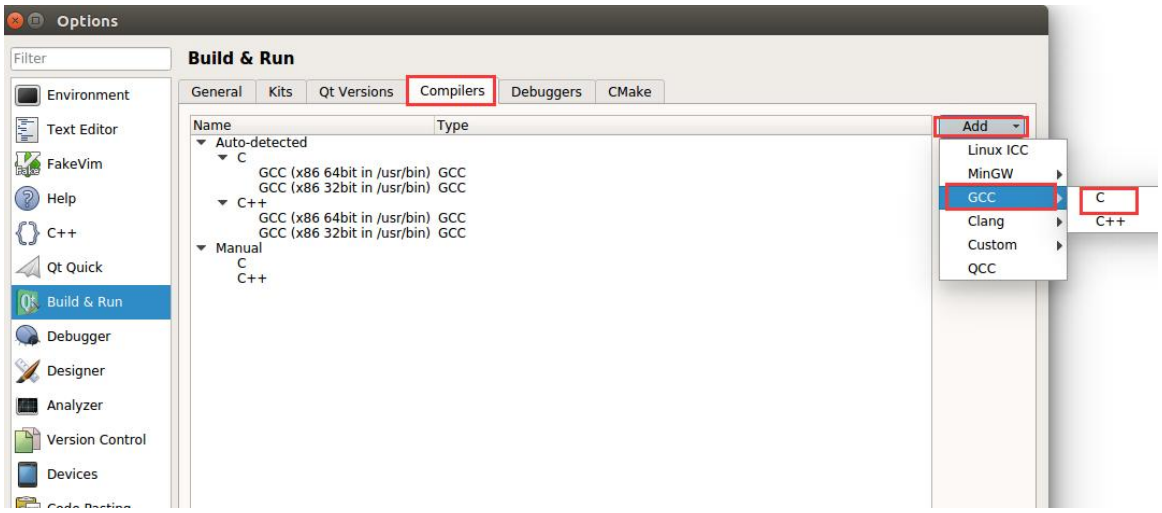
4) Select the qmake file generated by the previous SDK compilation, the path is

/opt/petalinux/2020.1/sysroots/x86_64-petalinux-linux/usr/bin/qmake

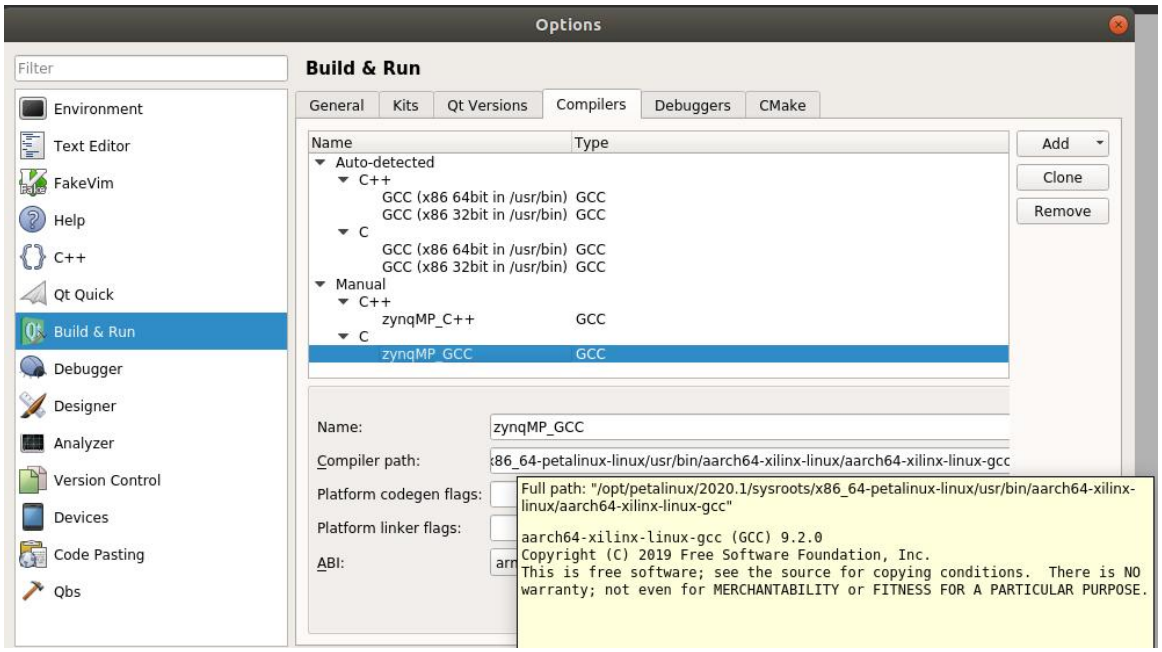
5) After adding qmake, keep the default name, do not modify



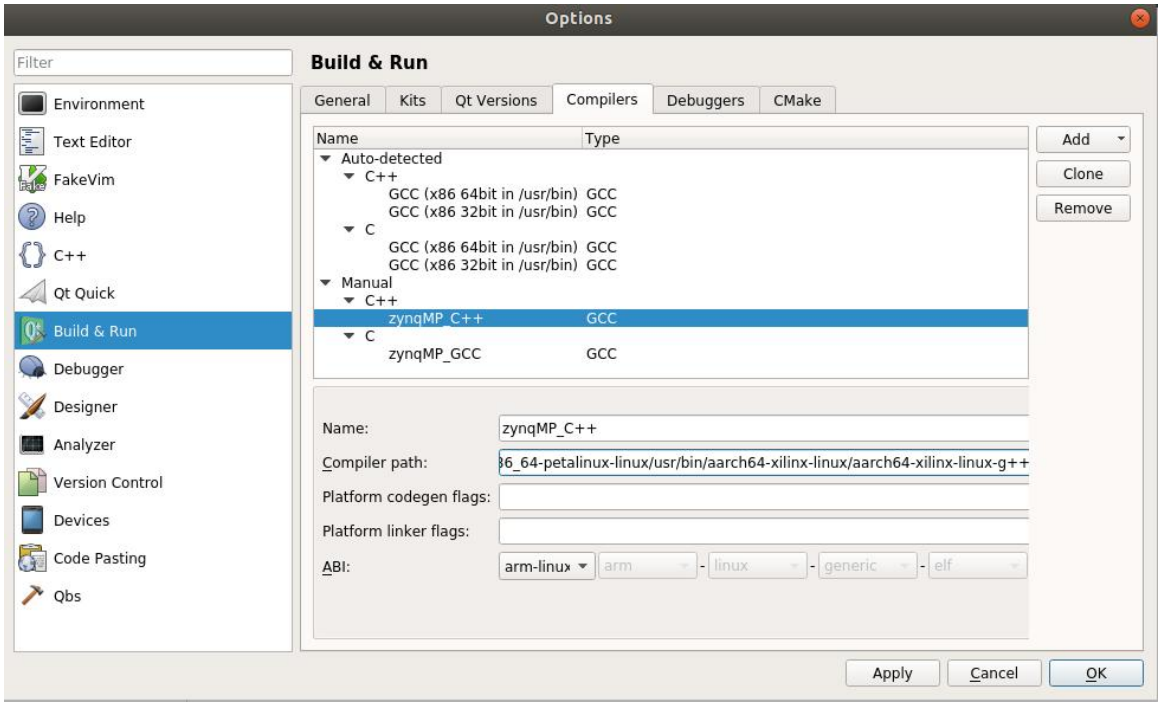
6) On the Compilers page, click "Add..", select GCC -> C



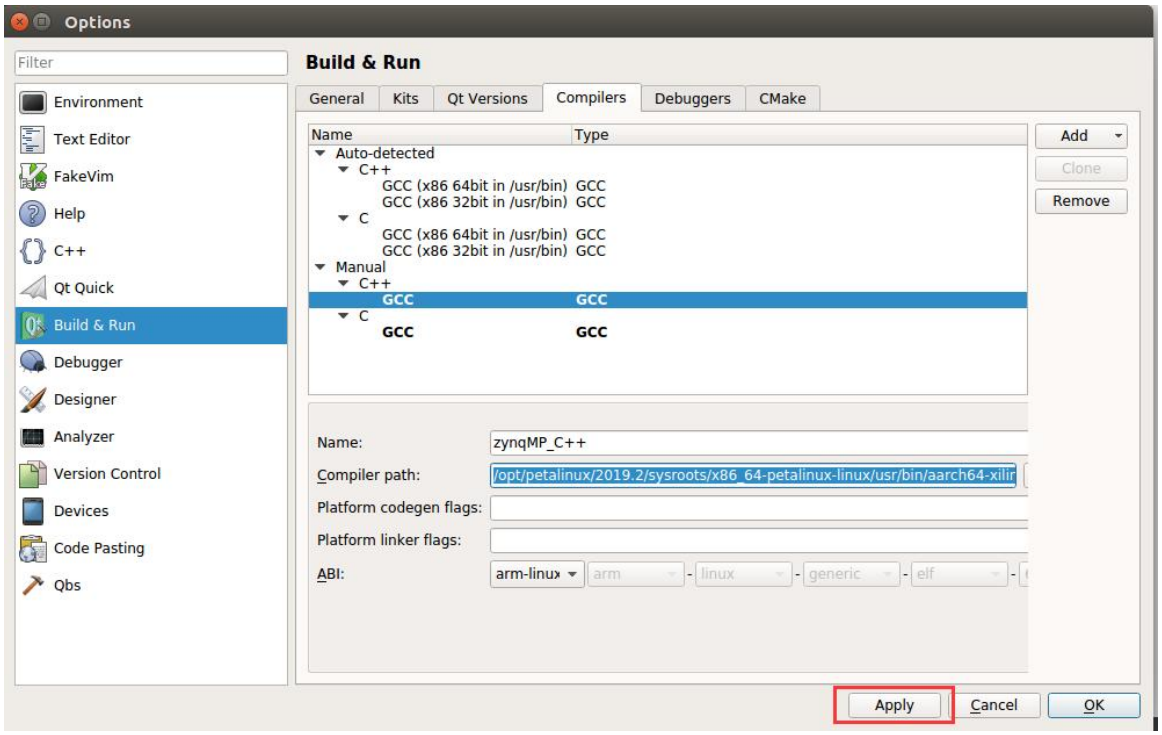
7) The Name is modified to zynqMP_GCC, and the Compiler path is /opt/petalinux/2020.1/sysroots/x86_64-petalinux-linux/usr/bin/aarch64-xilinx-linux/aarch64-xilinx-linux-gcc



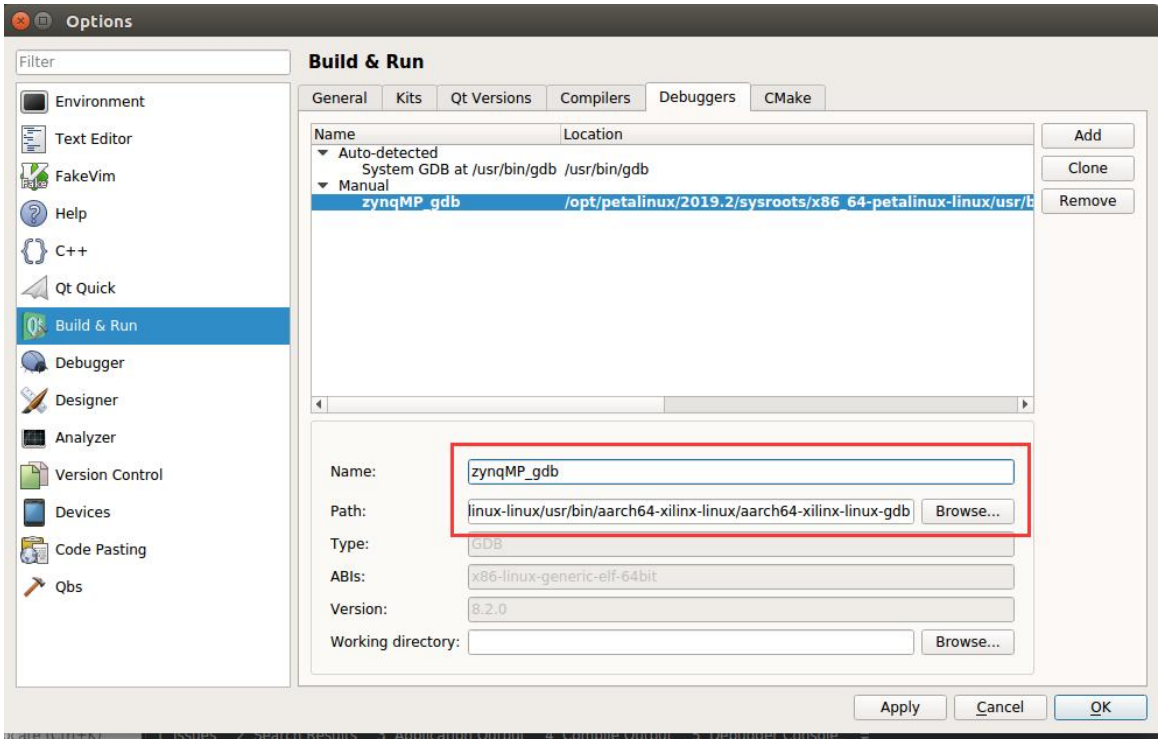
8) On the Compilers page, click Add.., select GCC -> C++, change the Name to zynqMP_C++, and the path selection is /opt/petalinux/2020.1/sysroots/x86_64-petalinux-linux/usr/bin/aarch64-xilinx-linux/aarch64-xilinx-linux-g++



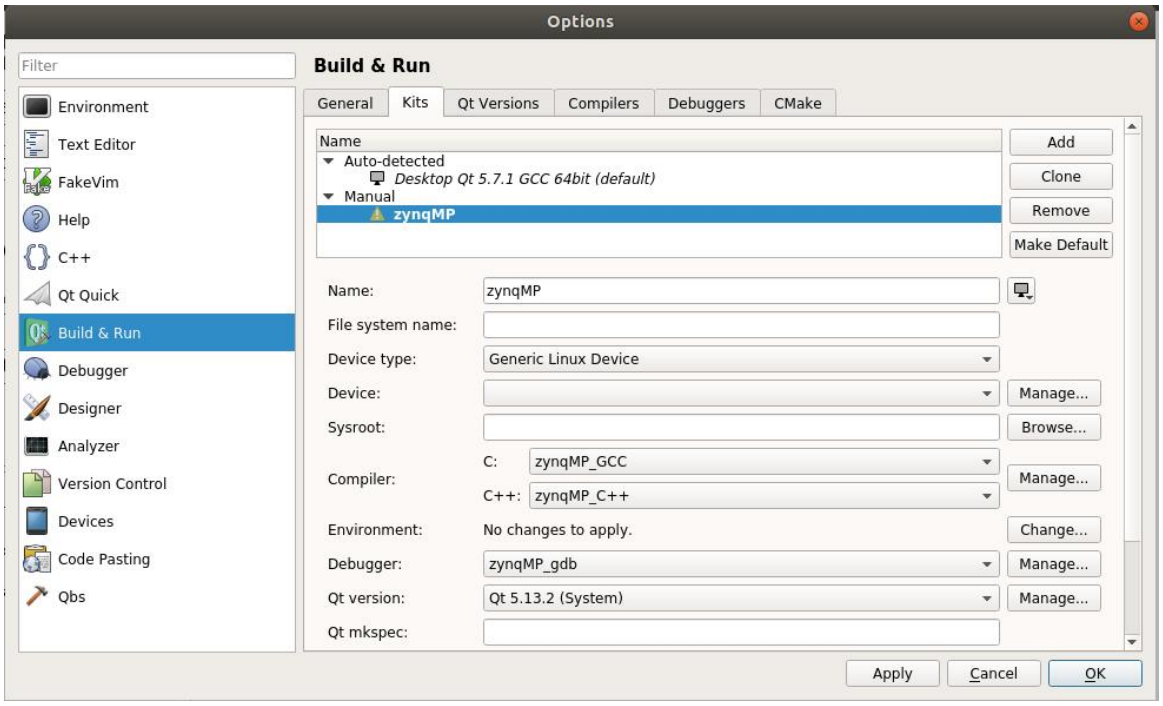
9) Click "Apply" again to update the information



10)Select Debuggers, add one, click Add to add a debugger, change the name to zynqMP_gdb, and the path selection is /opt/petalinux/2020.1/sysroots/x86_64-petalinux-linux/usr/bin/aarch64-xilinx-linux/aarch64-xilinx-linux-gdb, click Apply



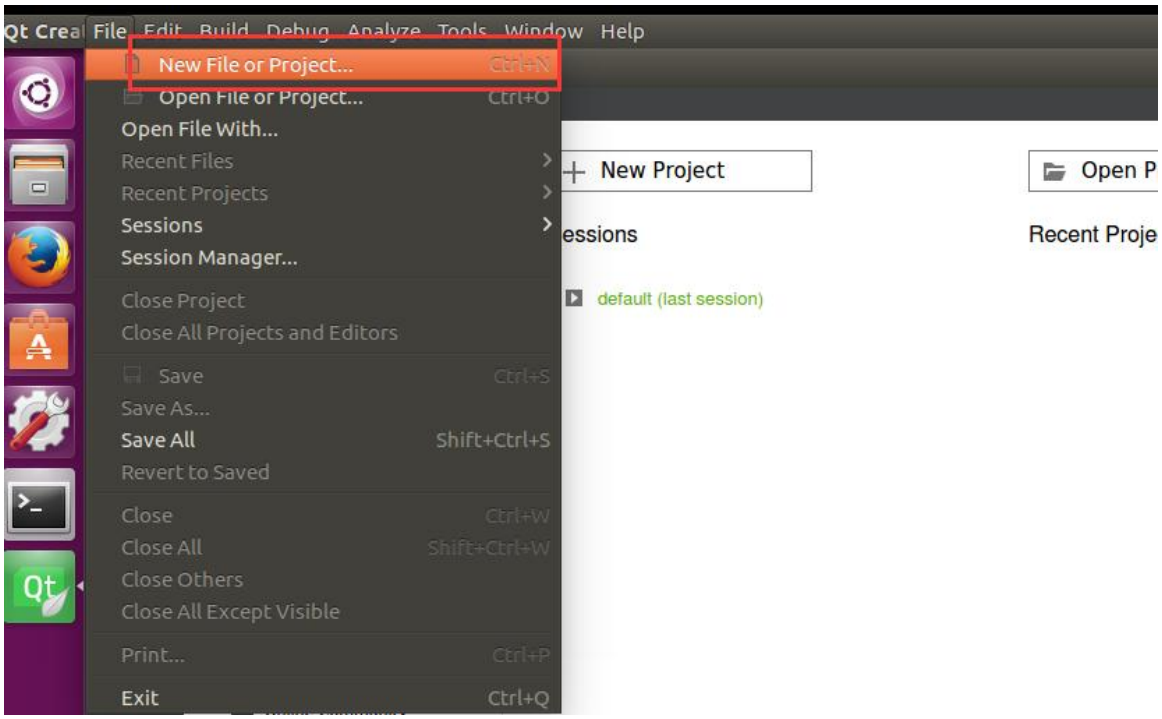
11)On the Kits page, click Add to add a new Kit, change the Name to zynqMP, select Generic Linux Device for Device type, Compiler C: select zynqMP_GCC, Compiler C++ select zynqMP_C++, Debugger select zynqMP_gdb, Qt version select Qt 5.13.2 (System)



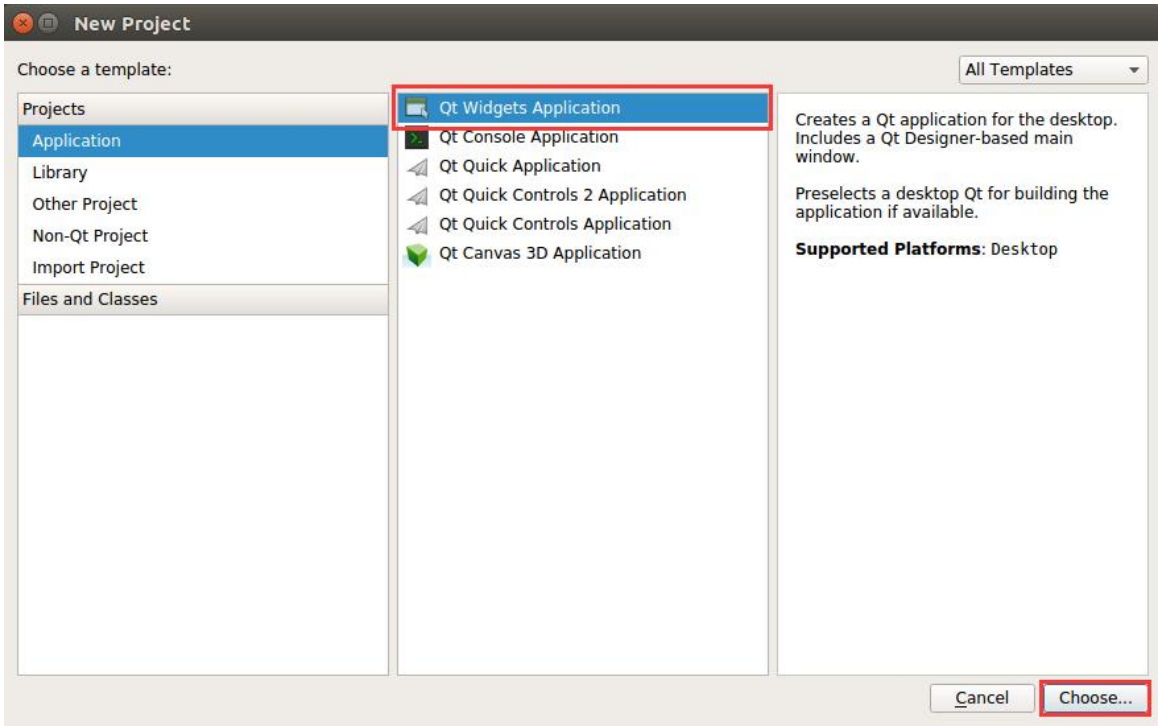
12)Click OK to configure Kits to complete

Part 6.3: Create QT test project

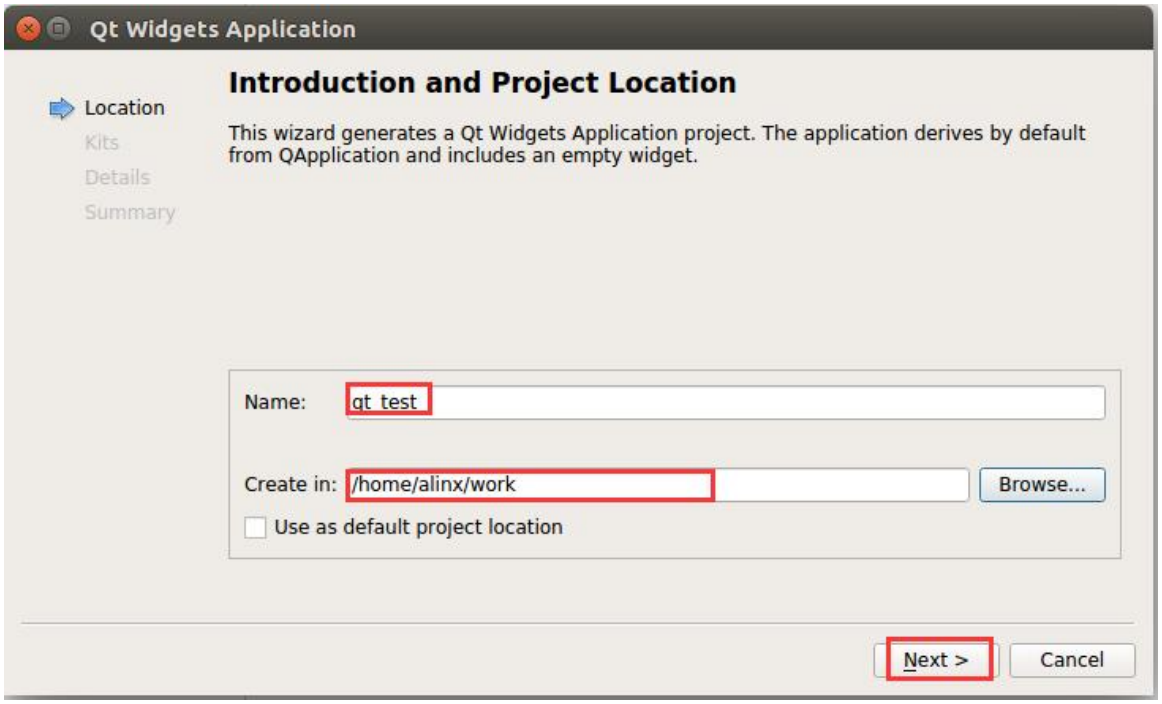
1) Click File -> New File or Project



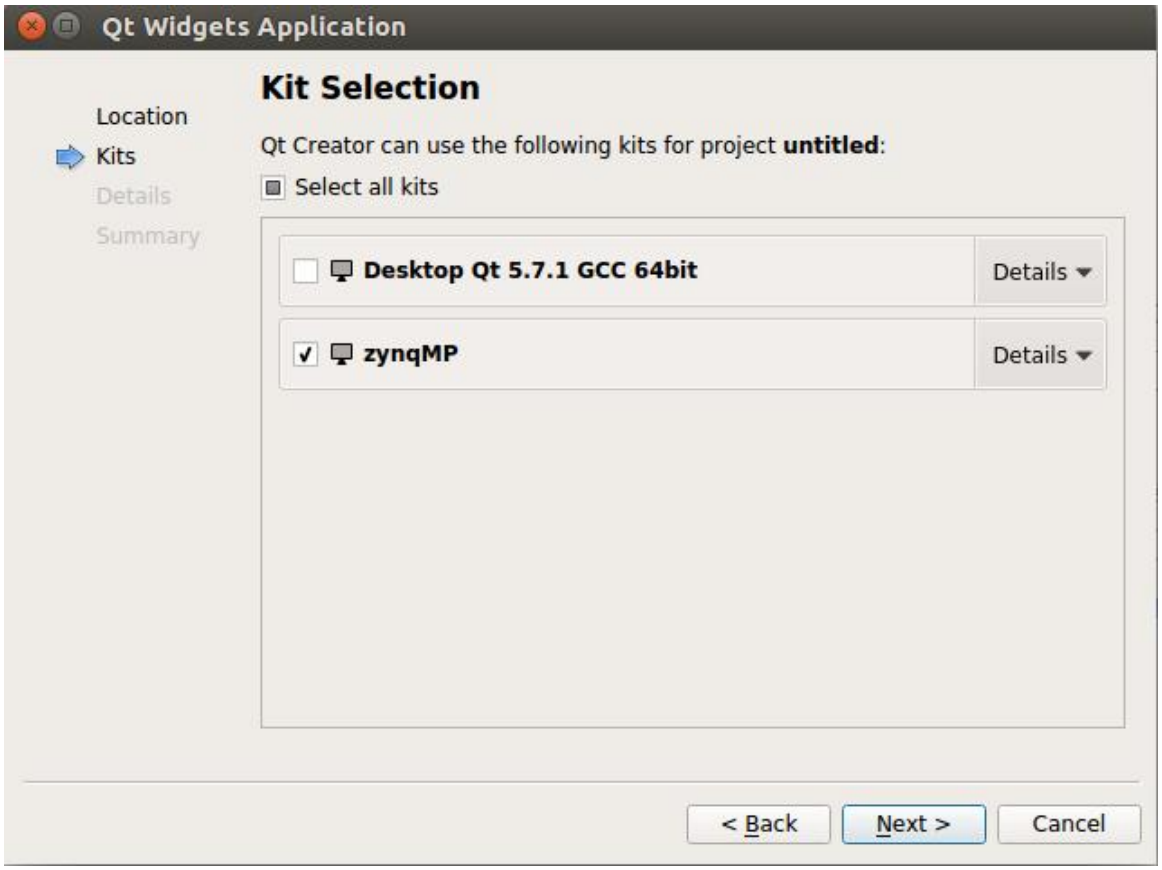
2) Select "Application" "Qt Widgets Application" for the project template, and then click "Choose..."



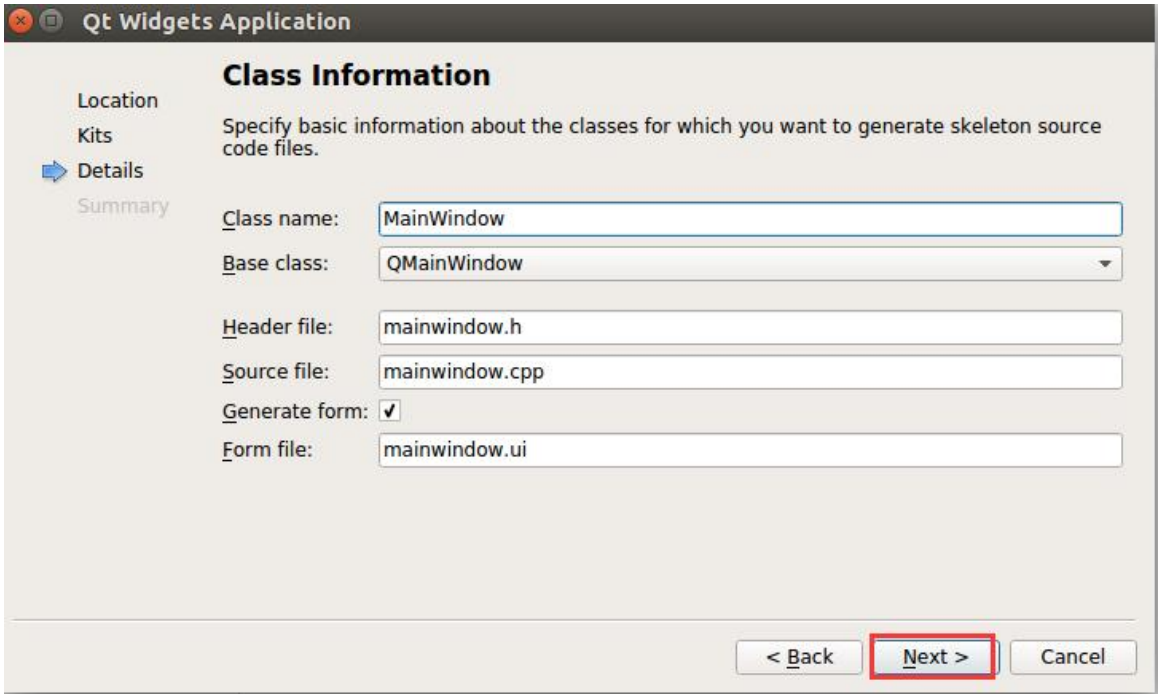
3) Change the Name to qt_test, and select the path /home/alinx/work for this experiment



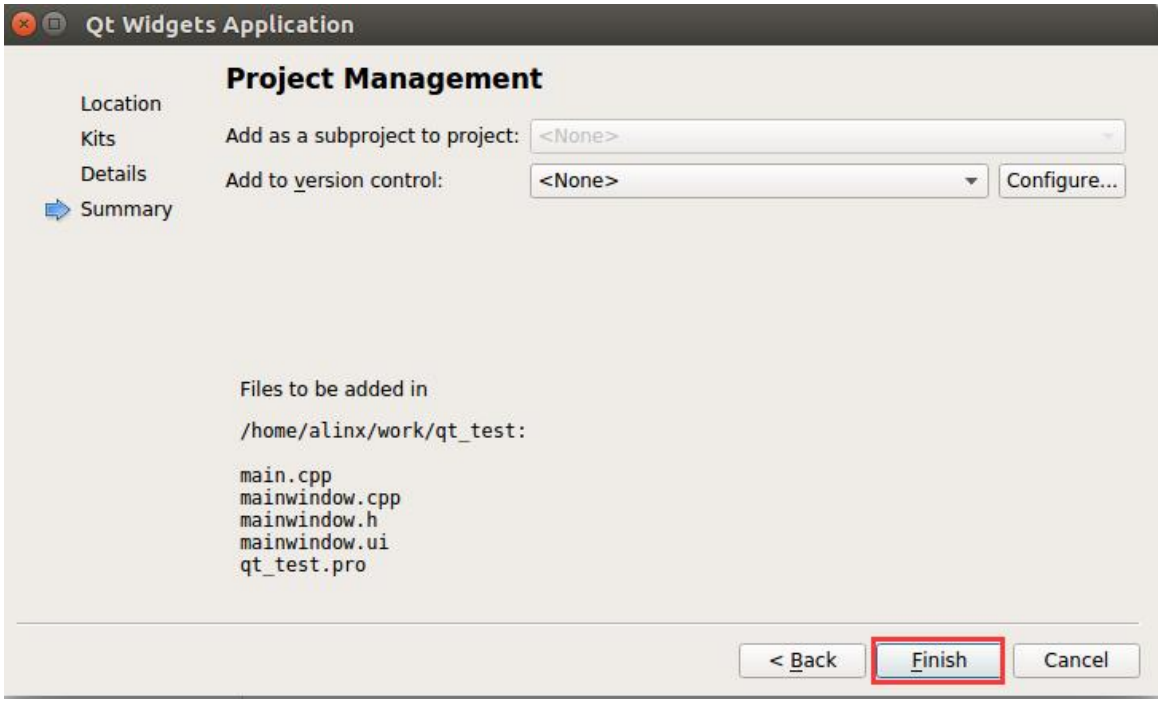
4) Kit selection, choose ZYNQMP version here



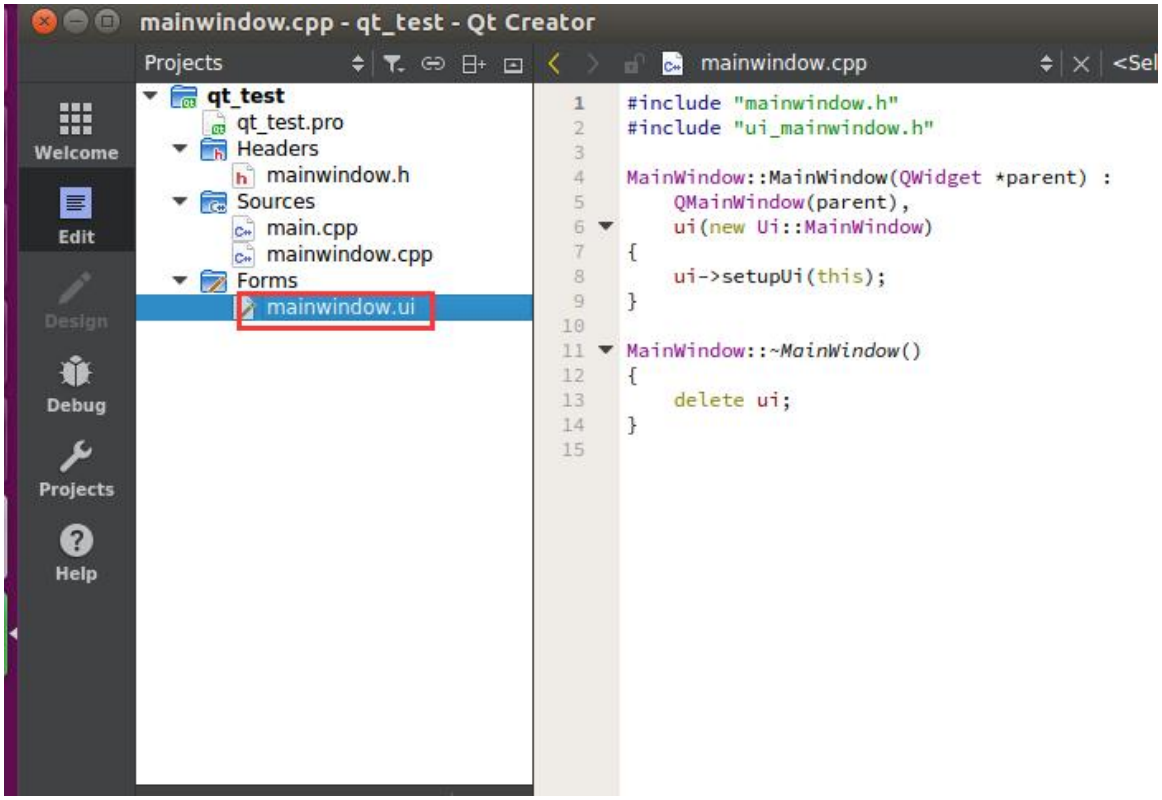
5) Choose Next to continue



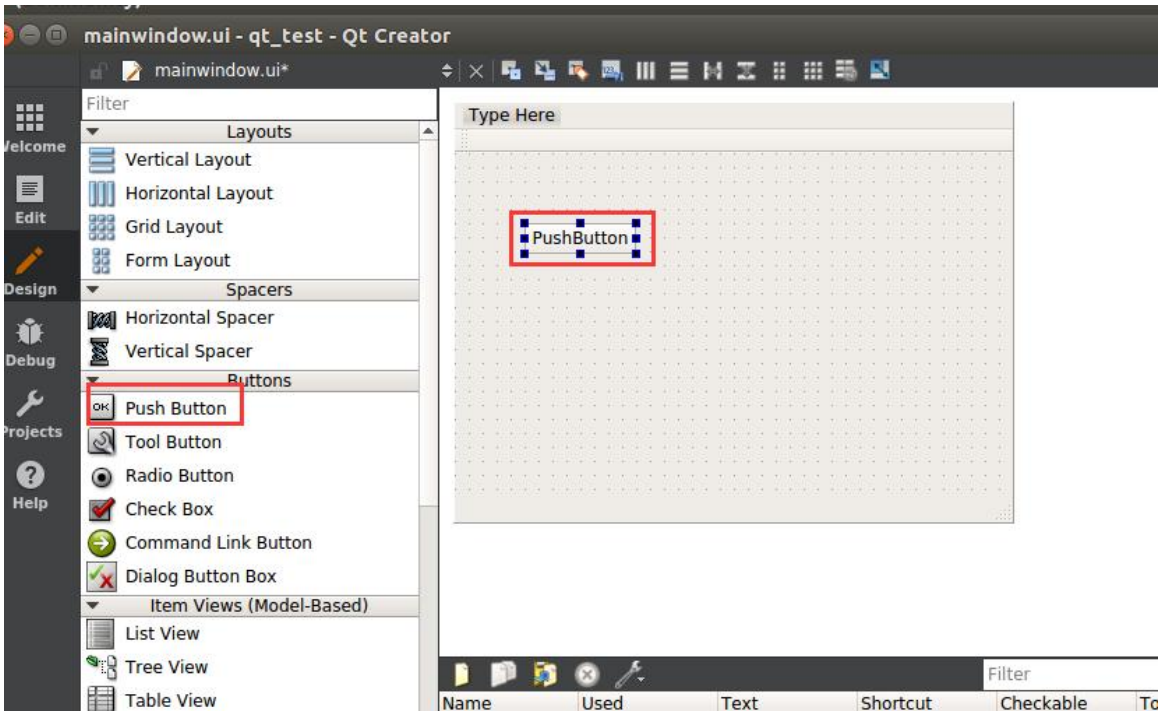
6) Select Finish to end the project creation wizard



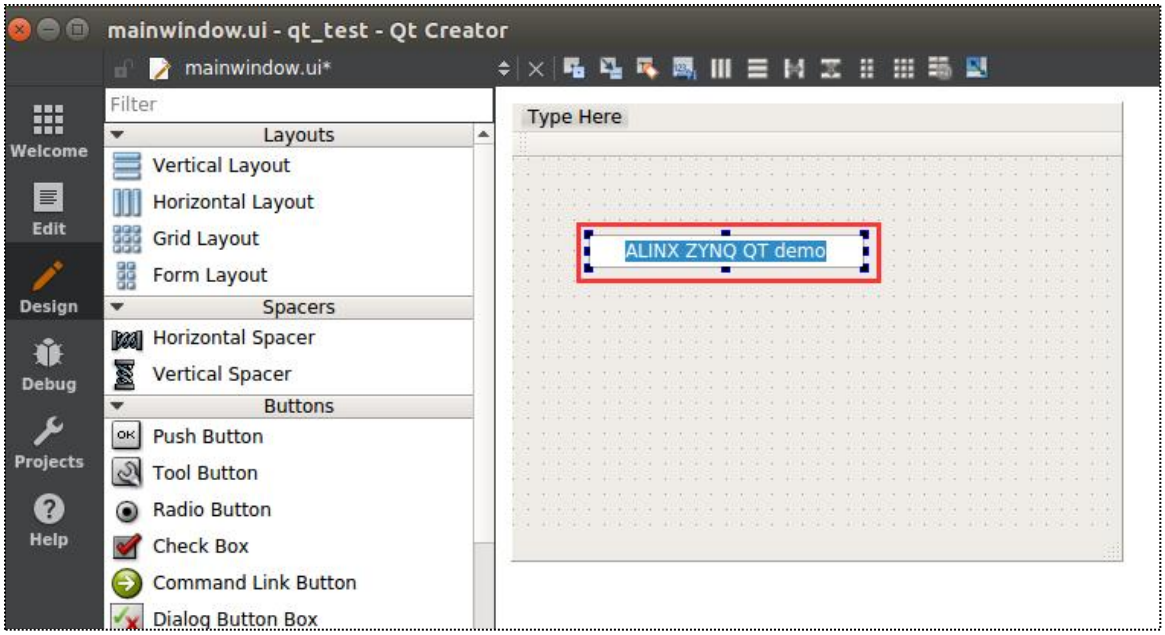
7) Double-click the mainwindow.ui file



8) Drag and drop a Push Button to the main interface

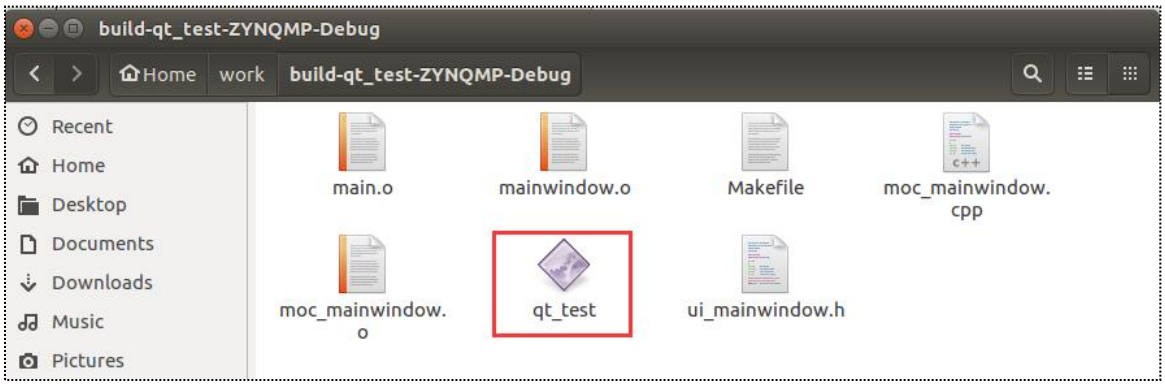


9) Double-click the text part to modify the button text



10)Select ZYNQ Kit and click the hammer icon to compile the program

11)In the build-qt_test-ZYNQMP-Debug directory, you can see that a qt_test file has been generated, and this file needs to be run on the FPGA board.



Part 6.5:Run the cross-compiled version of the QT program

1) Copy the previously made linux files BOOT.BIN and image.ub to the sd card (sd_boot directory)
2) Start the FPGA board and enter the system
3) Set the Environment Variables for Qt library Operation (if X11 is not configured, use export QT_QPA_PLATFORM=linuxfb for environment variables)

```
export DISPLAY=:0.0
```

4) Run the qt_test program

```
/media/sd-mmcblk1p1/qt_test
```

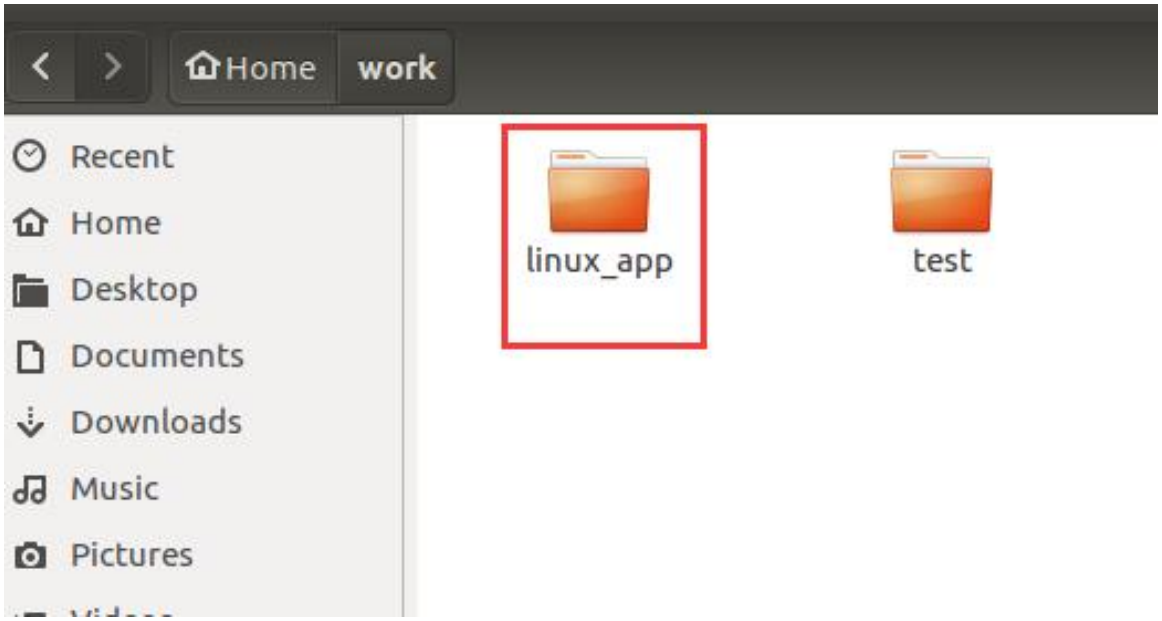
5) Press Ctrl + C to end the program

Part 7: Use Vitis to Develop Linux Programs

In the previous tutorial, we used petalinux to make an embedded Linux system. In this experiment, we will make a Linux application and run it on the FPGA development board. This experiment needs to use the Linux operating environment prepared in the above experiment.

Part 7.1: Use Vitis to build Linux applications

1) Create a directory "linux_app" in "/home/alinx/work" for the workspace of the SDK

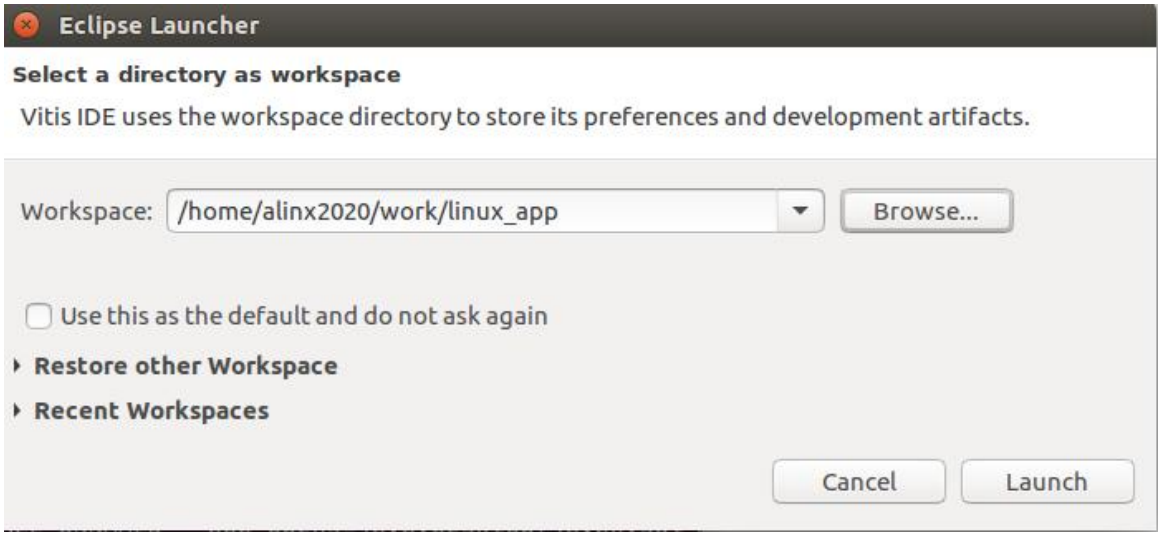


2) Set Vivado environment variables and run vitis

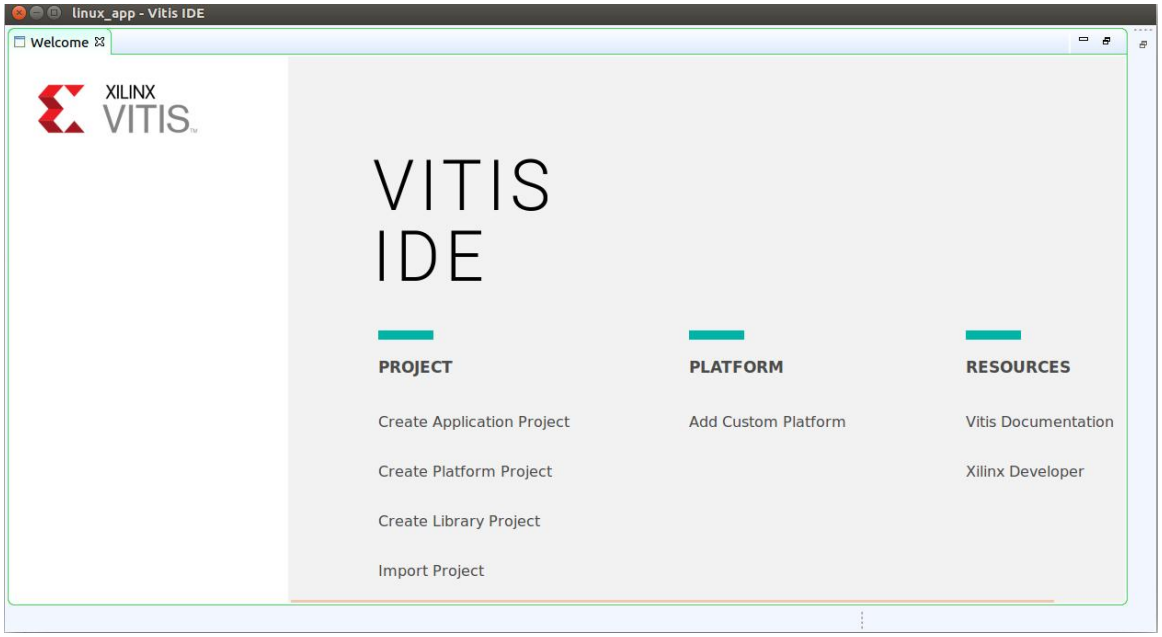
```
source /tools/Xilinx/Vivado/2020.1/settings64.sh
vitis&
```

```
alinx2020@ubuntu: ~/Downloads/course_s4_linux/01_ax_peta/ax_peta

INFO: Binary is ready.
WARNING: Unable to access the TFTPBOOT folder /tftpboot!!!
WARNING: Skip file copy to TFTPBOOT folder!!!
alinx2020@ubuntu:~/Downloads/course_s4_linux/01_ax_peta/ax_peta$ source /tools/X
ilinx/Vivado/2019.2/settings64.sh
alinx2020@ubuntu:~/Downloads/course_s4_linux/01_ax_peta/ax_peta$ vitis&
[1] 127408
alinx2020@ubuntu:~/Downloads/course_s4_linux/01_ax_peta/ax_peta$
****** Xilinx Vitis Development Environment
****** Vitis v2019.2 (64-bit)
  **** SW Build 2708876 on Wed Nov  6 21:40:25 MST 2019
    ** Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

Launching Vitis with command /tools/Xilinx/Vitis/2019.2/eclipse/lnx64.o/eclipse
-vmargs -Xms64m -Xmx4G -Dorg.eclipse.swt.internal.gtk.cairoGraphics=false -Dosgi
.configuration.area=@user.home/.Xilinx/Vitis/2019.2 --add-modules=ALL-SYSTEM --a
dd-modules=jdk.incubator.httpclient --add-opens=java.base/java.nio=ALL-UNNAMED -
-add-opens=java.desktop/sun.swing=ALL-UNNAMED --add-opens=java.desktop/javax.swi
ng=ALL-UNNAMED --add-opens=java.desktop/javax.swing.tree=ALL-UNNAMED --add-opens
=java.desktop/javax.swing.plaf.basic=ALL-UNNAMED --add-opens=java.desktop/javax.
swing.plaf.synth=ALL-UNNAMED --add-opens=java.desktop/com.sun.awt=ALL-UNNAMED --
add-opens=java.desktop/sun.awt.X11=ALL-UNNAMED &
```

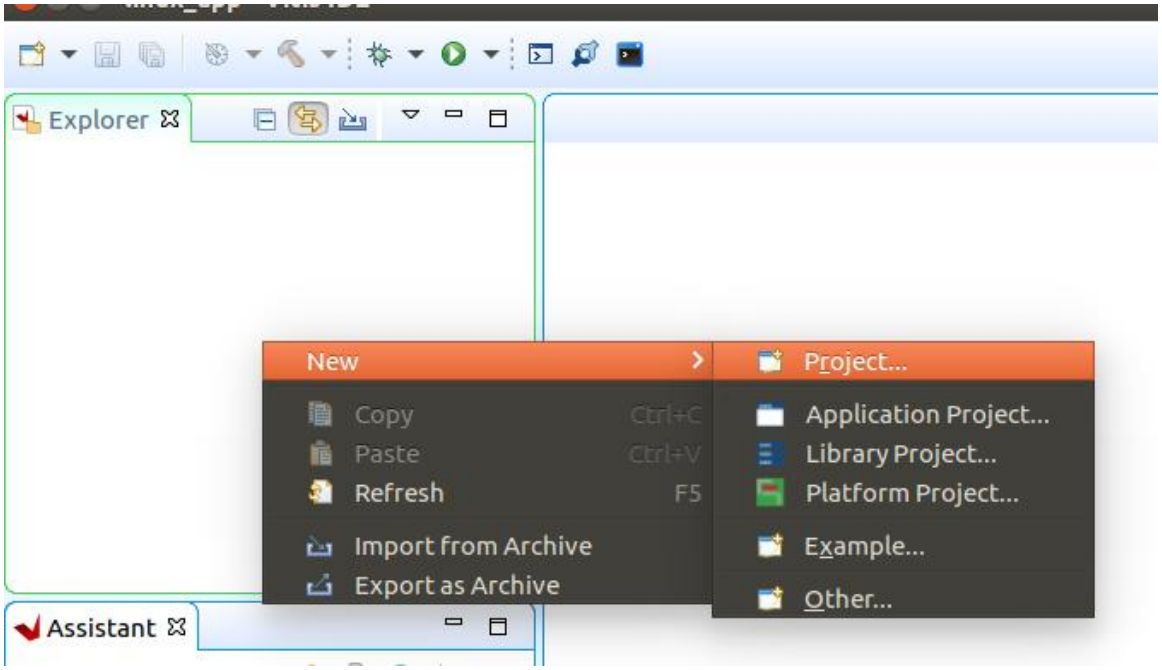
3) Work space select "/home/alinx/work/linux_app"



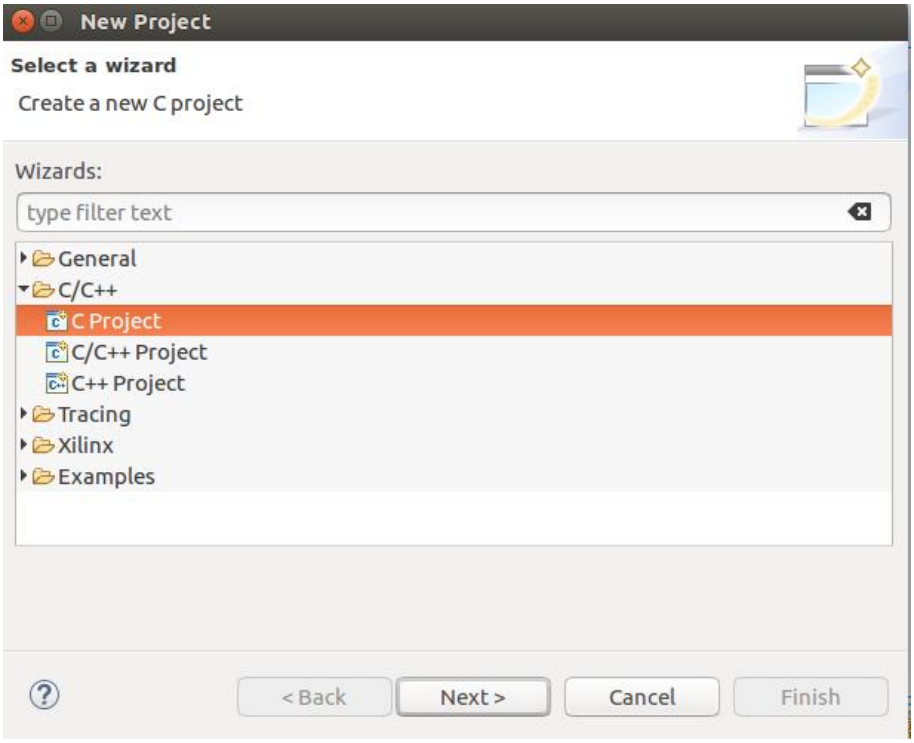
4) Click the X next to "Welcome" to close the welcome screen



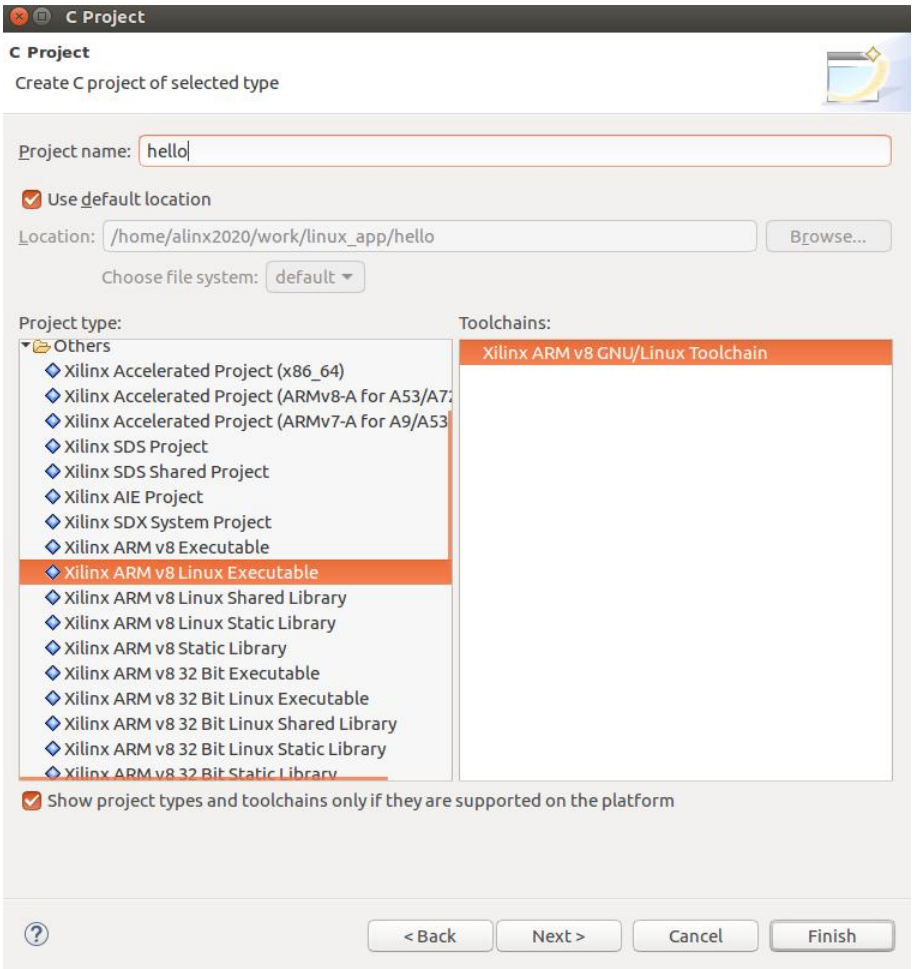
5) Right-click "New -> Project..." in the blank space of Explorer



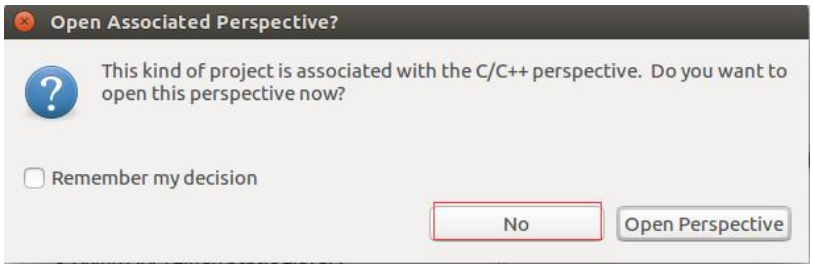
6) Select "C Project", which is a project in C language, click "Next"

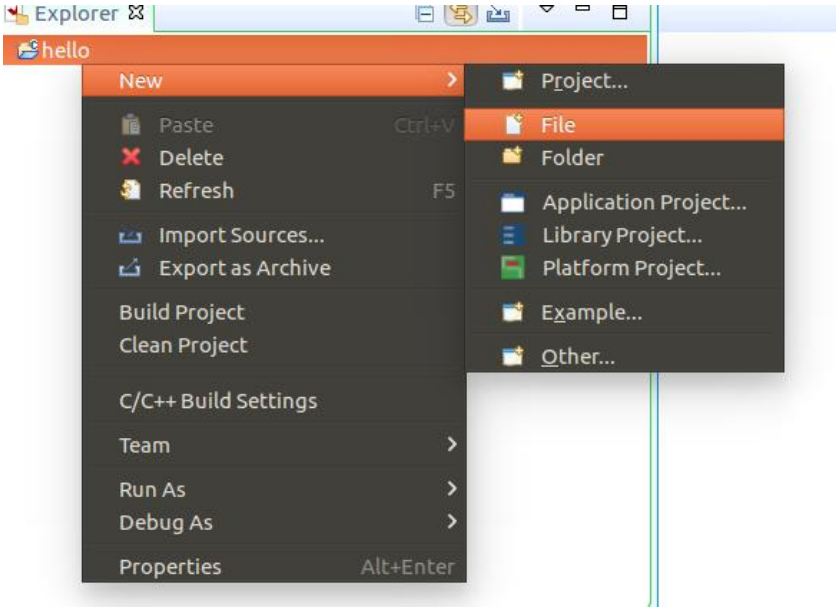


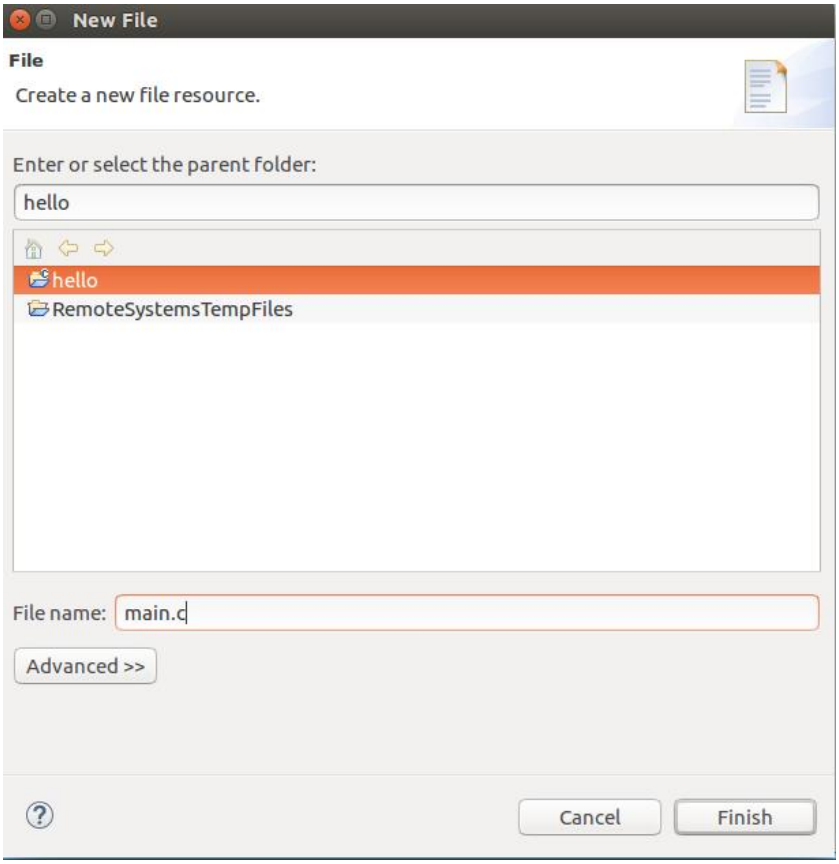
7) Fill in the project name "hello", select the project type "Xilinx ARM v8 Linux Executable" in the Others folder, and then click "Finish"



8) If the following message pops up, click "No"



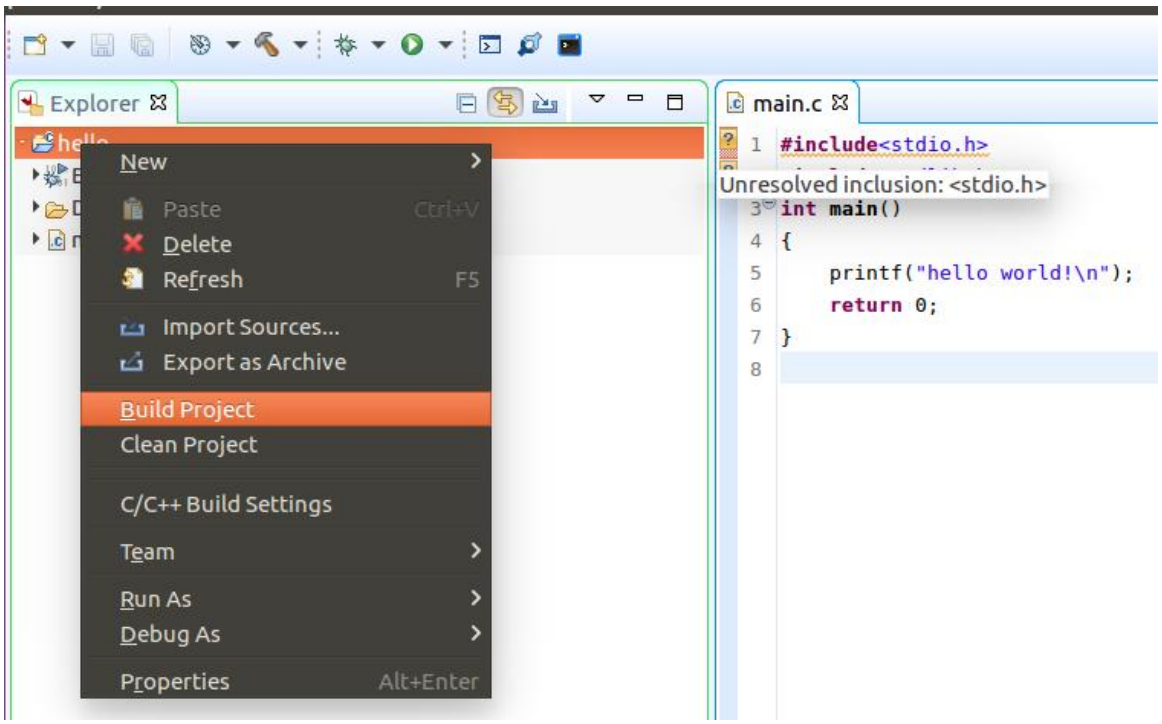




9) Edit the main.c file and write a simple print

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    printf("hello world!\n");
    return 0;
}
```

10)Build Project



Part 7.2: Run via NFS share

1) The FPGA development board is plugged into the Internet cable (PS, ETH1, the router needs to support automatic IP acquisition), power on, and mount the Linux host NFS, where 192.168.1.115 is the host address, and /home/alinx2020/work is the NFS directory, /mnt is the mount point of the FPGA development board. Here, the host and FPGA development board are required to be in the same

network segment (the red part must be consistent with your actual situation).

```
mount -t nfs -o nolock 192.168.1.115:/home/alinx2020/work /mnt
```

```
ax_peta login: root
Password:
root@ax_peta:~# mount -t nfs -o nolock 192.168.1.115:/home/alinx2020/work /mnt
root@ax_peta:~#
```

2) Enter /mnt/linux_app/hello/Debug directory, run hello.elf, we can see Hello World printed out.

```
cd /mnt/linux_app/hello/Debug
./ hello.elf
```

```
Password:
root@ax_peta:~# mount -t nfs -o nolock 192.168.1.115:/home/alinx2020/work /mnt
root@ax_peta:~# cd /mnt/linux_app/hello/Debug
root@ax_peta:/mnt/linux_app/hello/Debug# ./hello.elf
hello world!
root@ax_peta:/mnt/linux_app/hello/Debug#
```

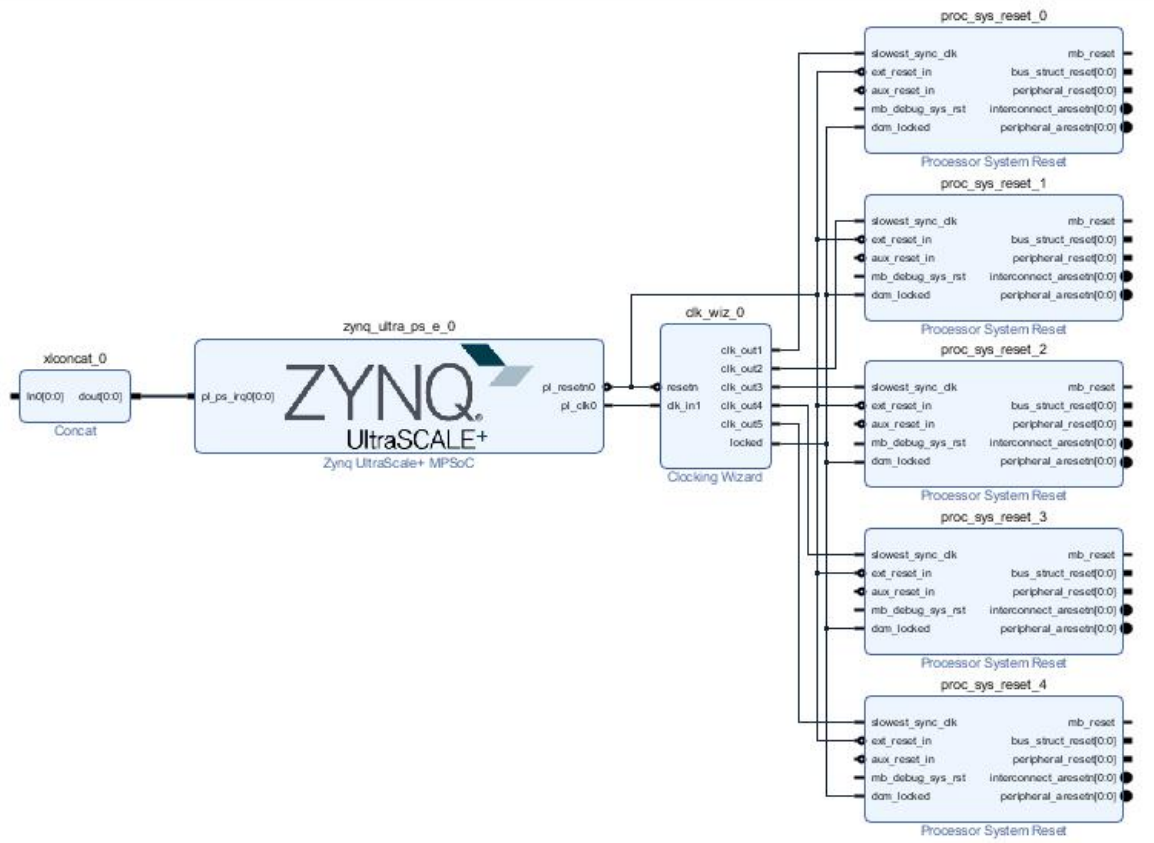
Part 8: Vitis Accelerates Basic Platform Creation

Note: This tutorial requires proficient use of vivado and petalinux, and it is recommended to try Vitis acceleration after learning OpenCL and HLS.

This chapter mainly introduces how to customize the hardware and software platform that uses PL-side FPGA acceleration, which requires special processing in the VIvado project to be used for acceleration. Acceleration requires knowledge of OpenCL and HLS.

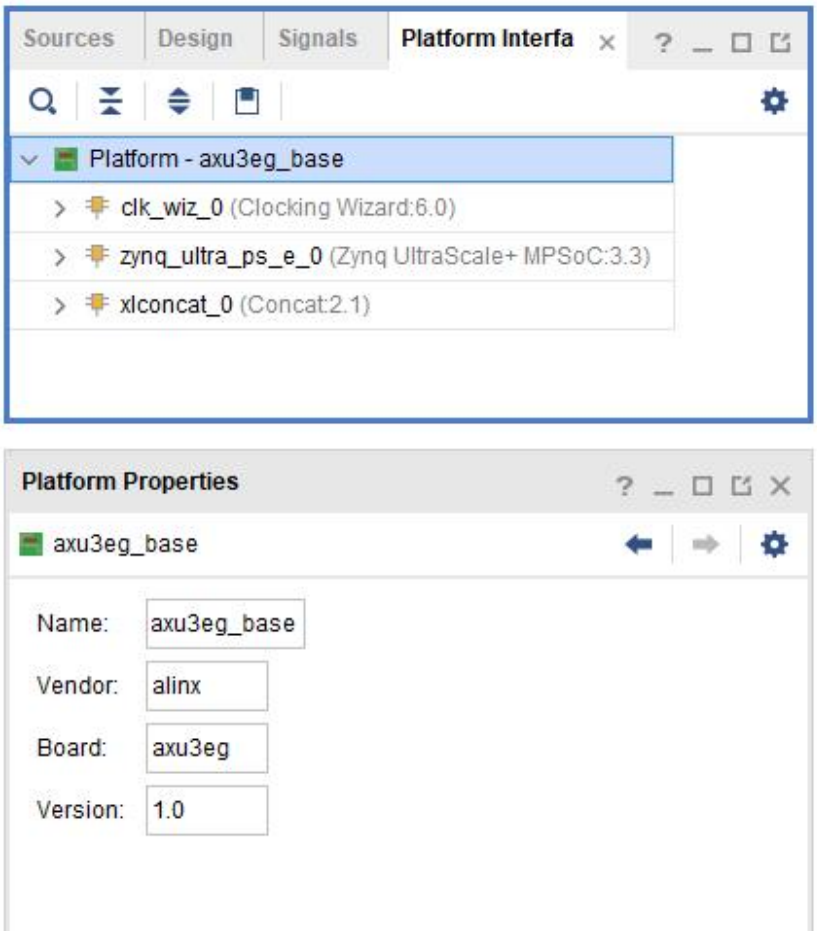
Part 8.1: Vivado hardware platform

1) The specific steps of the Vivado project will not be explained in detail, as shown below



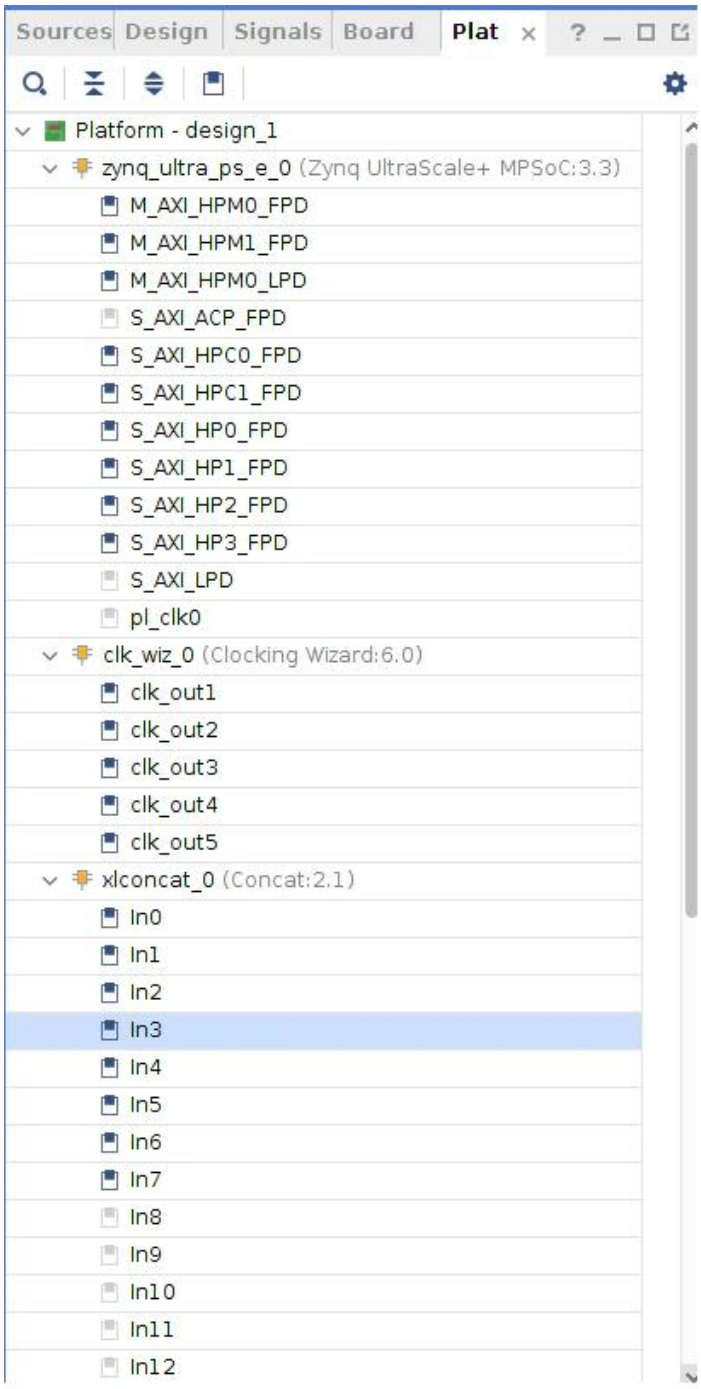
2) Define platform (PFM) interfaces and attributes. Define the interface used by Vitis here. Click Window→Platform Interface on the Vivado menu to set the platform. Click Enable platform Interfaces to set the platform.

3) Change the name of the board to make it easier to understand on the Vitis platform. Choose Platform, change the following Board to axu3eg

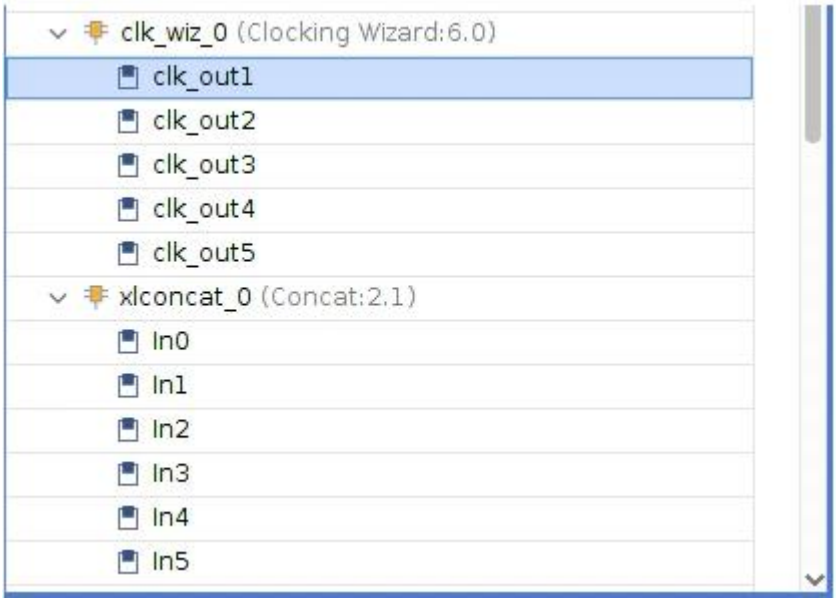


4) Define the available interfaces and ports. Right-click on each port,

and the menu will come out. You can also switch between Enable and Disenable by double-clicking. In xynq_ultra_ps_e_0, set the next interface and port to Enable. (Other than Dienable) M AXI HPM0 FPD, M AXI HPM1 FPD, M AXI HPM0 LPD, S AXI HPC0 FPD, S AXI HPC1 FPD, S AXI HP0 FPD, S AXI HP1 FPD, S AXI HP2 FPD, S AXI HP3 FPD; for clk wiz 0 all use Enable; for xlconcet 0, set In 0～In 7 to Enable.



5) Clock default value. A certain clock must be set to the default value. After selecting clk out1, click the Options tab in Platform Interface Properties and select "is default". In addition, the id must be set to 0. The default clock that can be used in Vitis. The other clocks are arranged in sequence starting from 1, and after inputting the serial number, press the Enter key to confirm.

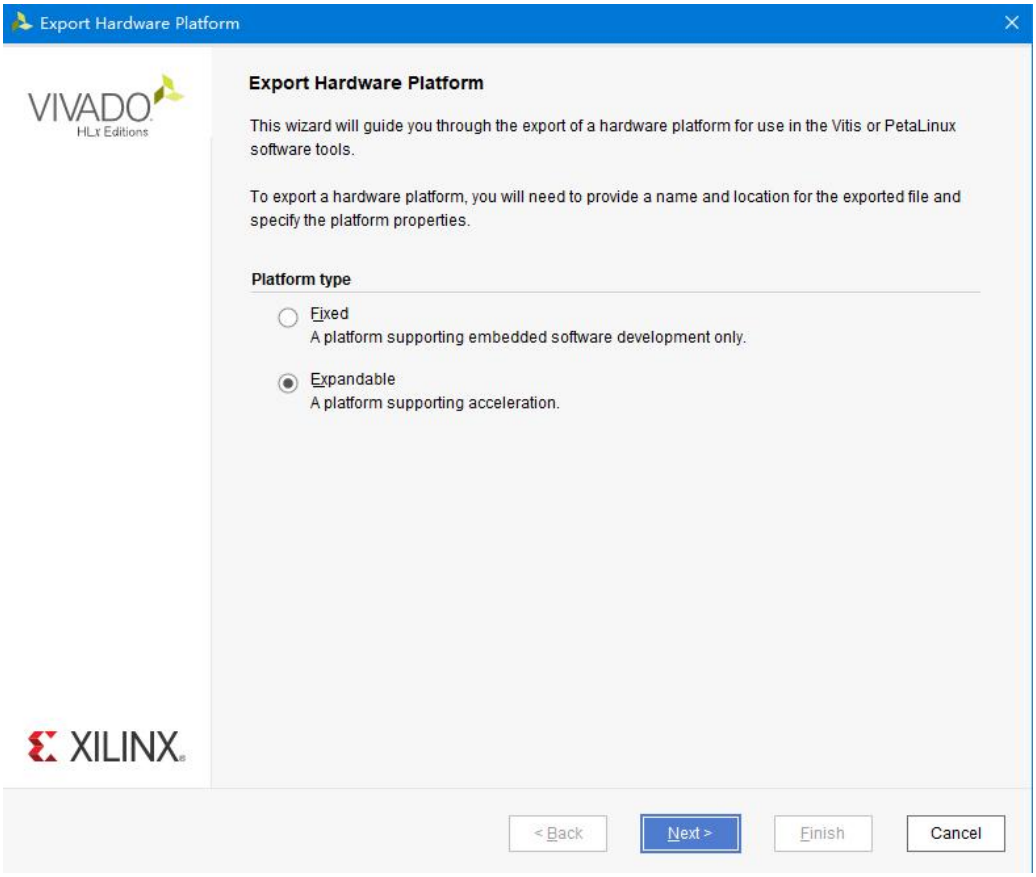


6) Then compile and generate bit file
7) XRT environment settings, enter the following command on the Tcl console

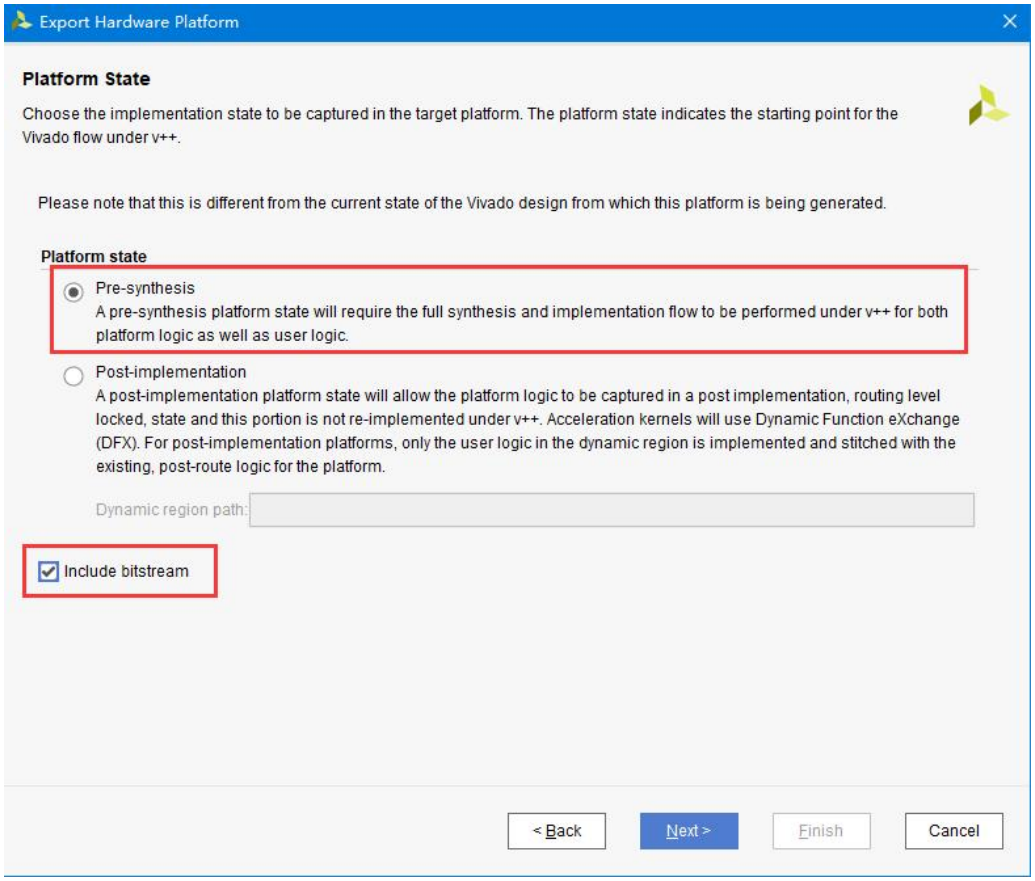
```
set_property platform.design_intent.embedded true [current_project]
set_property platform.design_intent.server_managed false [current_project]
set_property platform.design_intent.external_host false [current_project]
set_property platform.design_intent.datacenter false [current_project]
set_property platform.default_output_type "sd_card" [current_project]
```

Part 8.2: Export XSA

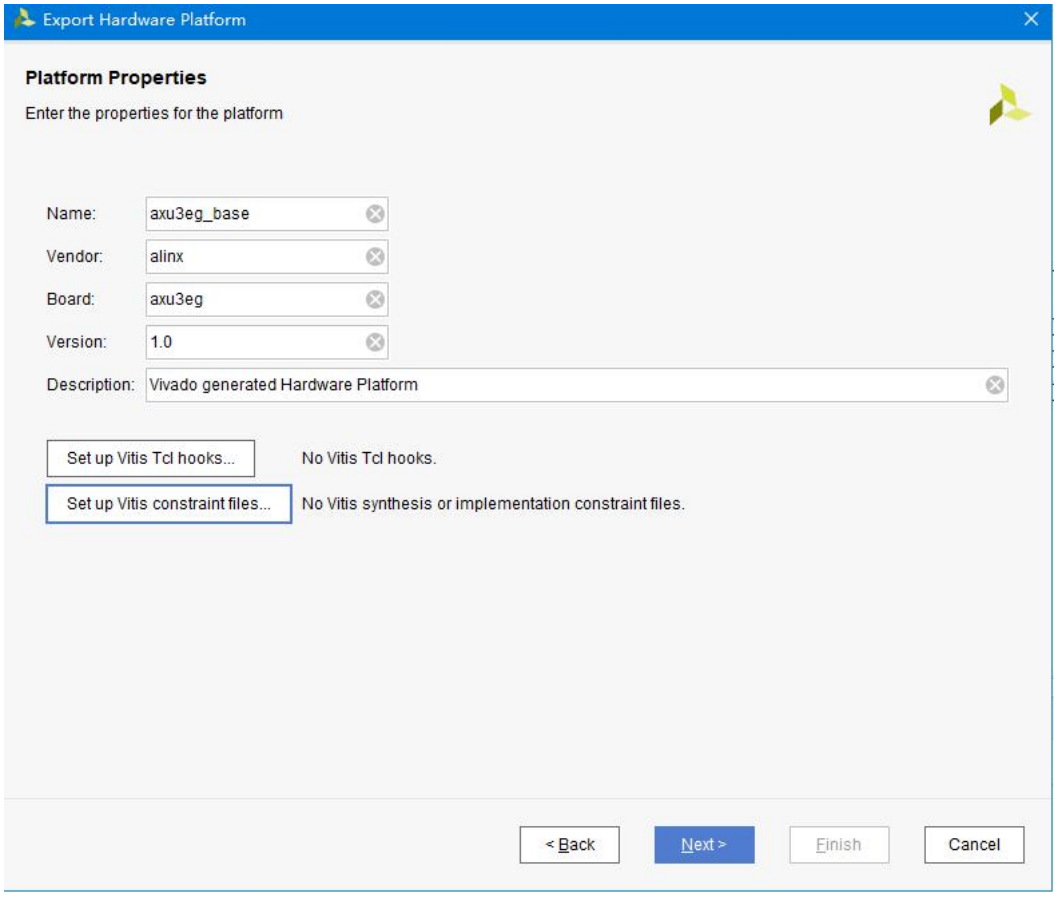
8) Export the xsa file and select "Expandable", so that you can use vitis for acceleration.



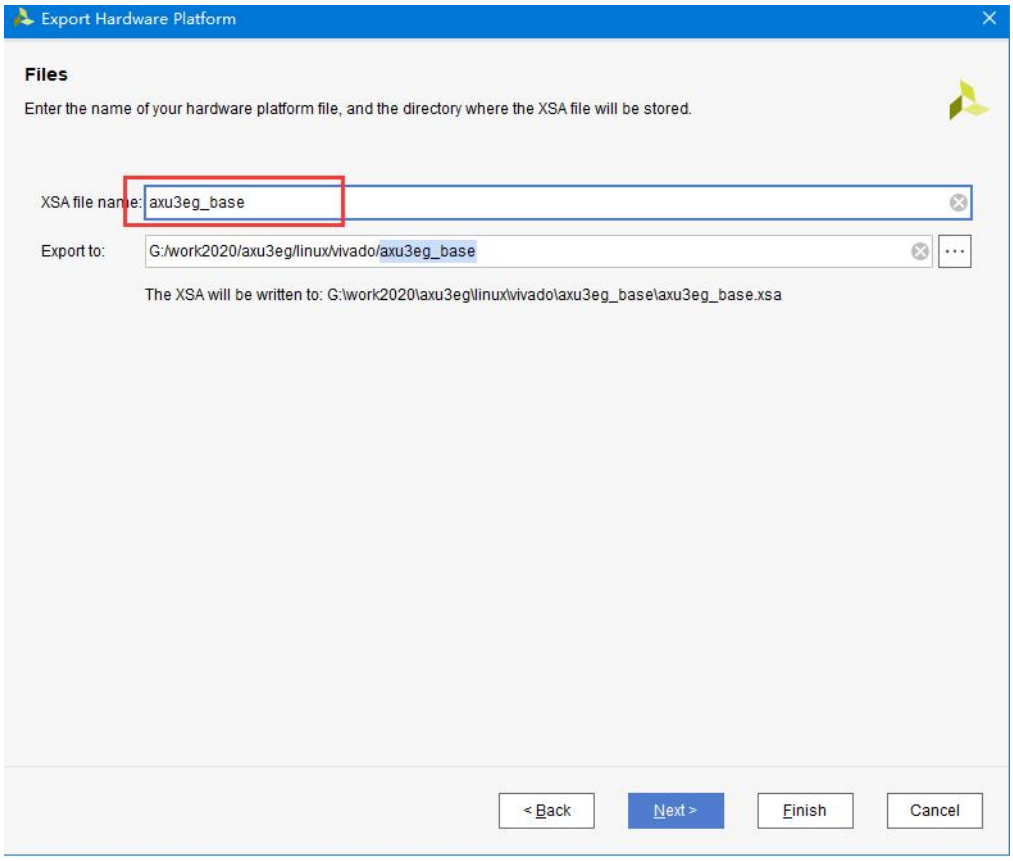
9) Platform state select "Pre-synthesis", then select "Include bitstream"



10)Fill in the platform properties as shown below



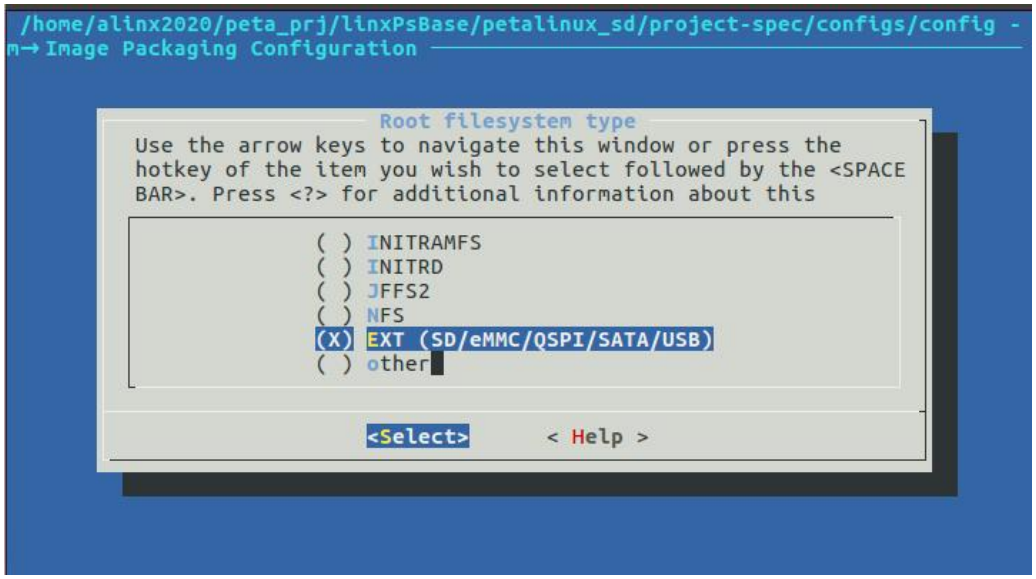
11) Fill in the xsa file name with "axu3eg_base"



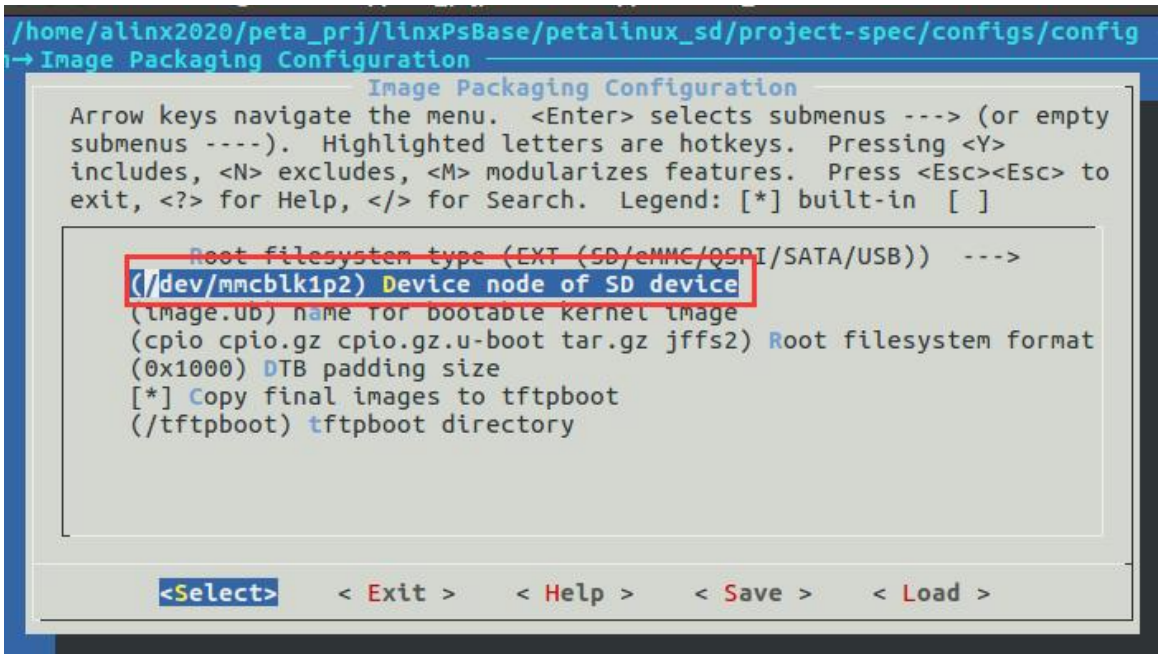
Part 8.3: Vitis Software Platform

Part 8.3.1: Petalinux configuration

1) Image Packaging Configuration -> Root filesystem type select EXT (SD/eMMC/QSPI/SATA/USB)



2) The root file system path selection is very important. If you fill in the wrong file and you cannot start Linux, mmcblk represents the sd card and emmc, mmcblk0p2 represents the second partition of the first mmc device, if the emmc and sd cards exist at the same time, the sd is mmcblk1, so fill in here /dev/mmcblk1p2



3) Modify ./ project-spec/meta-user/conf/user-rootfsconfig as follows

```
#Note: Mention Each package in individual line
#These packages will get added into rootfs menu entry


CONFIG_gpio-demo
CONFIG_peekpoke
CONFIG_xrt
CONFIG_xrt-dev
CONFIG_zocl
CONFIG_opencl-clhpp-dev
CONFIG_opencl-headers-dev
CONFIG_packagegroup-petalinux-opencv
```

4) Modify the device tree file

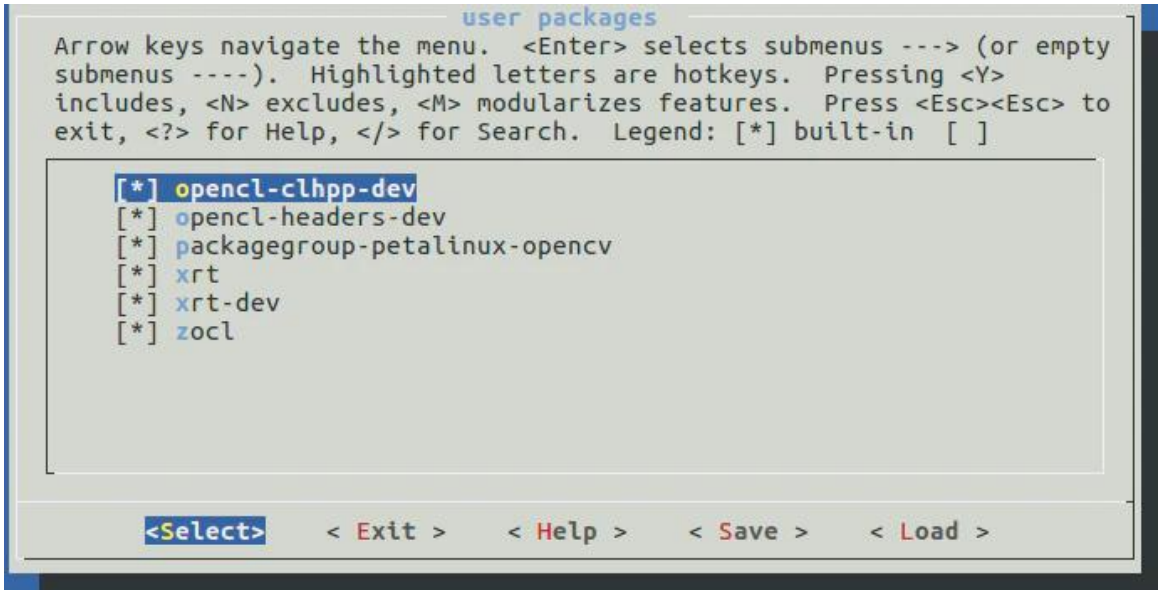
./project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi

```
/include/ "system-conf.dtsi"
/ {
};

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

/* USB	*/
&dwc3_0 {
	status = "okay";
	dr_mode = "host";
};

&amba {
	zyxclmm_drm {
		compatible = "xlnx,zocl";
		status = "okay";
	};
};
```

5) Configure the root file system

```
petalinux-cofing -c rootfs
```

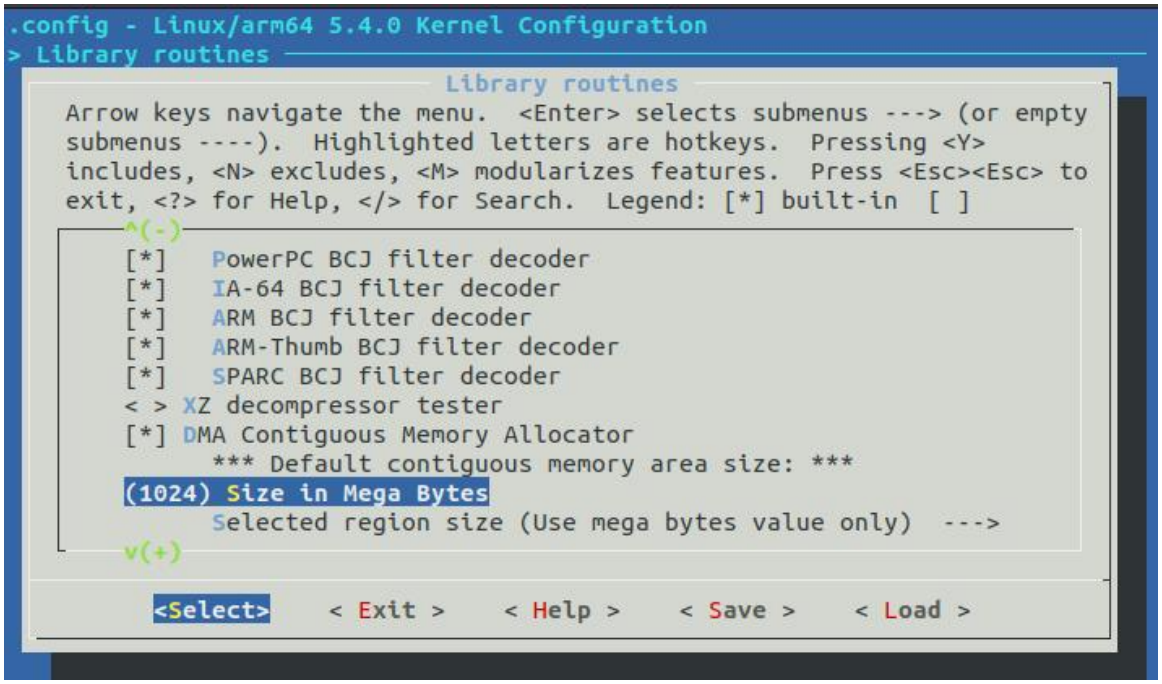
6) check all user packages



7) Configure the kernel

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

8) Library routines →Size in Mega Bytes changed to 1024



9) Compile petalinux project

```
petalinux-build
```

10)Build SDK

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

11)Create a new pfm folder to build the Vitis platform later

```
mkdir pfm
```

12)Install petalinux's sdk to the pfm directory. The sdk is easy to install. The previous tutorial explains

13)Create a boot folder, then copy image.ub zynqmp_fsbl.elf pmufw.elf bl31.elf u-boot.elf generated by petalinux to the boot folder

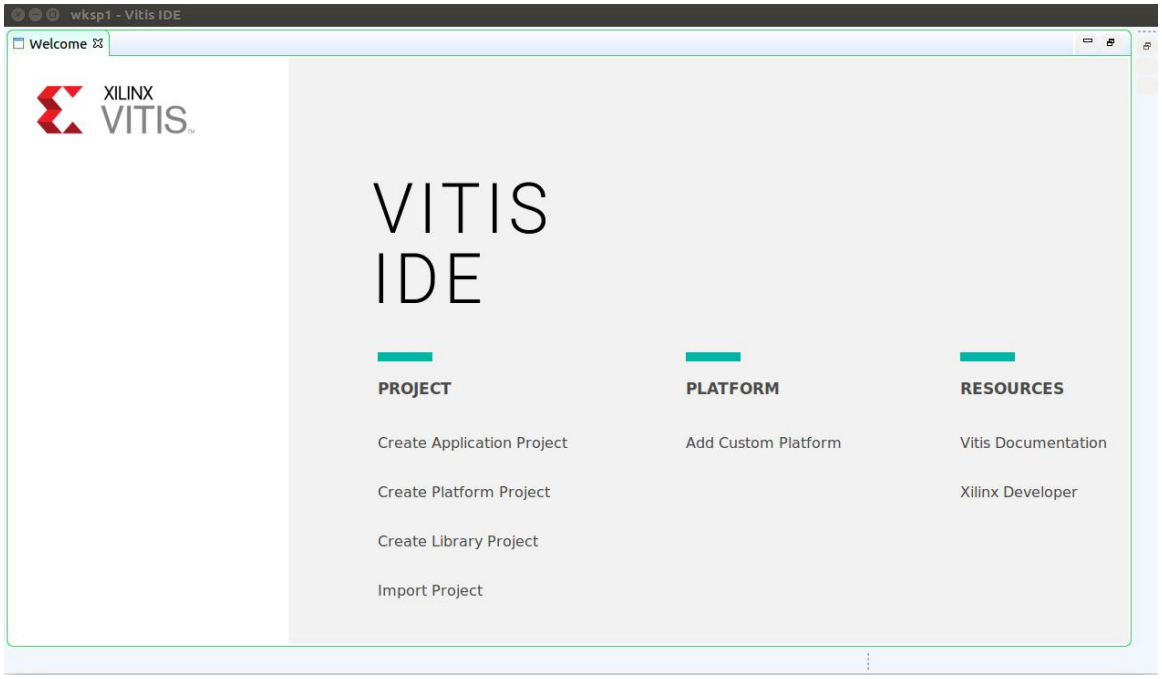
14)Create a linux.bif file in the boot folder, the content is as follows

```
/* linux */
the_ROM_image:
{
	[fsbl_config] a53_x64
	[bootloader] <zynqmp_fsbl.elf>
	[pmufw_image] <pmufw.elf>
	[destination_device=pl] <bitstream>
	[destination_cpu=a53-0, exception_level=el-3, trustzone] <bl31.elf>
	[destination_cpu=a53-0, exception_level=el-2] <u-boot.elf>
}
```

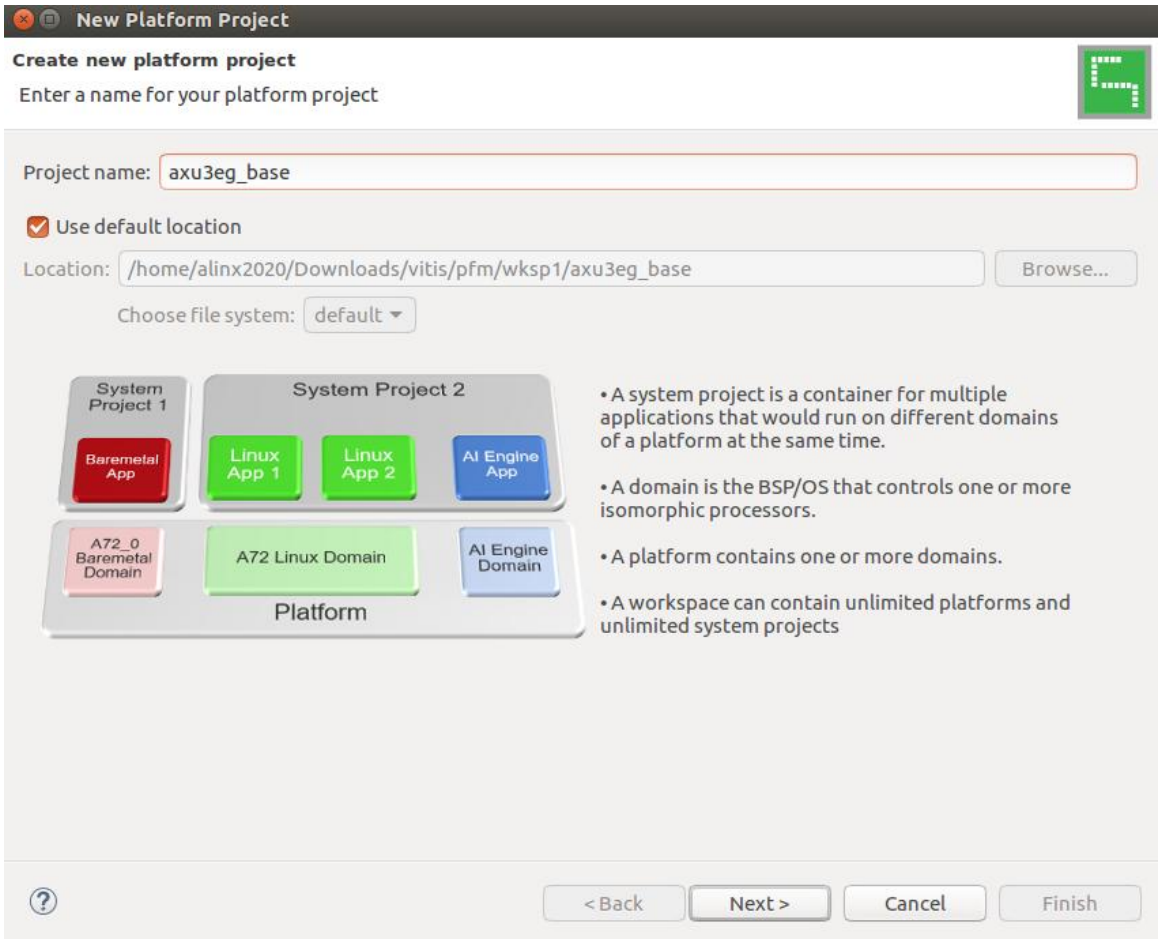
Part 8.3.2: Create vitis platform

15)Enter the pfm folder and start the vitis software

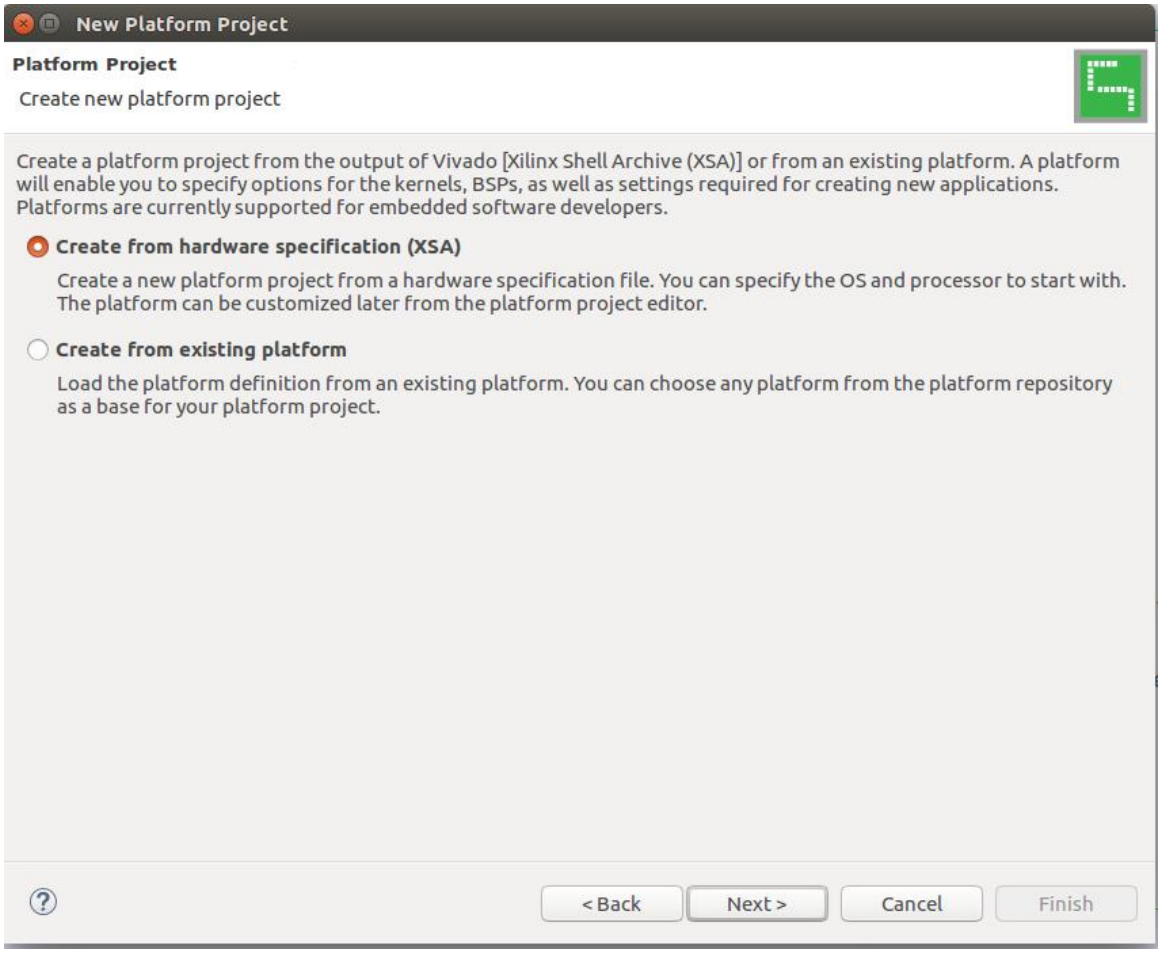
```
cd ./pfm
vitis -workspace wksp1
```



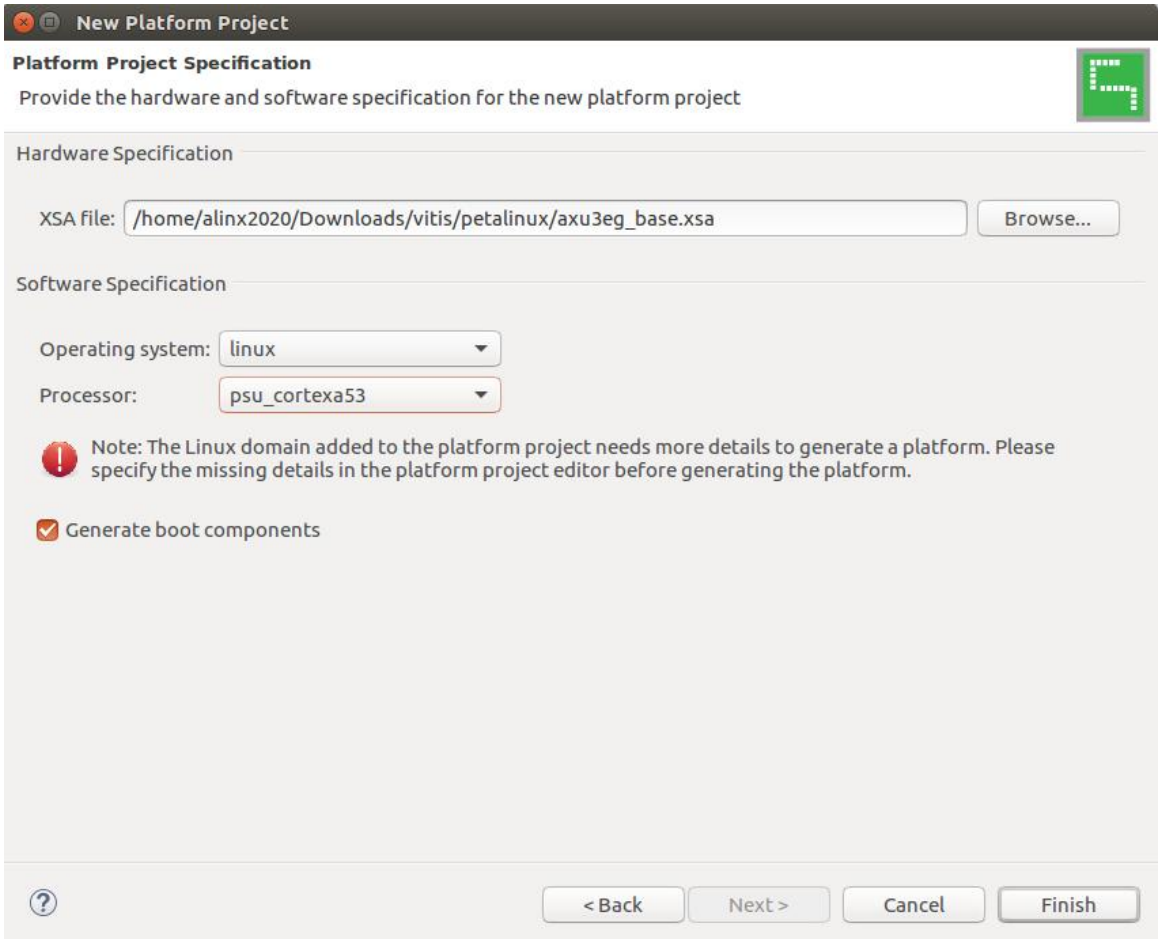
16)After Vitis starts, select Create Platform Project. (Or select File→New→Platform Project from the menu), fill in axu3eg_base in Project name, which is the same as xsa name



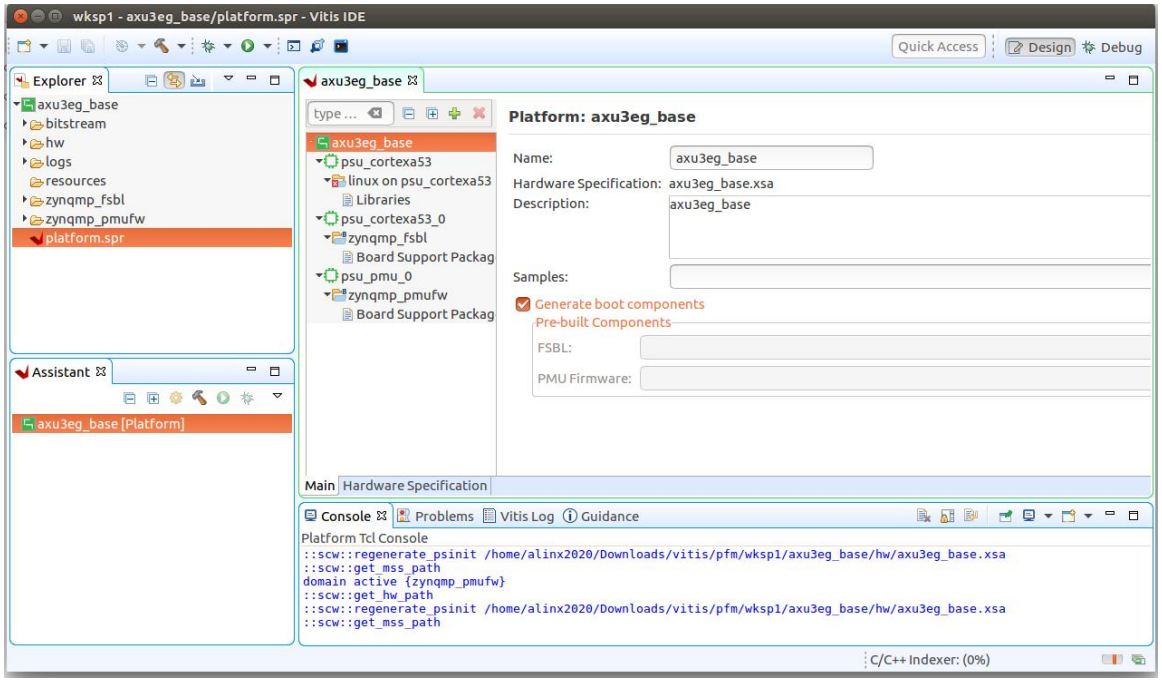
17)Next choose where to start making the platform. Because it is made from XSA, select Create from hardware specification (XSA), and then press Next



18)Specify the xsa file created with Vivado, the Operatings system is Linux, and the Processor selects psu_cortexa53. One of the signs is red, it doesn't matter. After setting, click Finish.

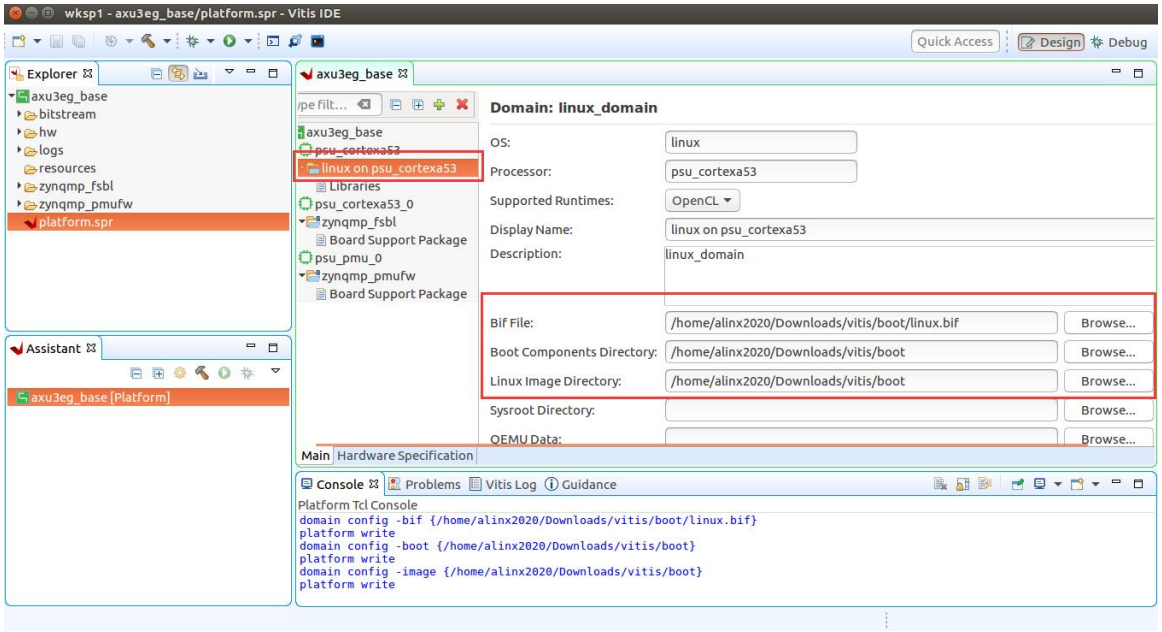


19)There are linux , zynqmp fsbl and zynqmp pmufw in the window. For linux, it needs to be set later.



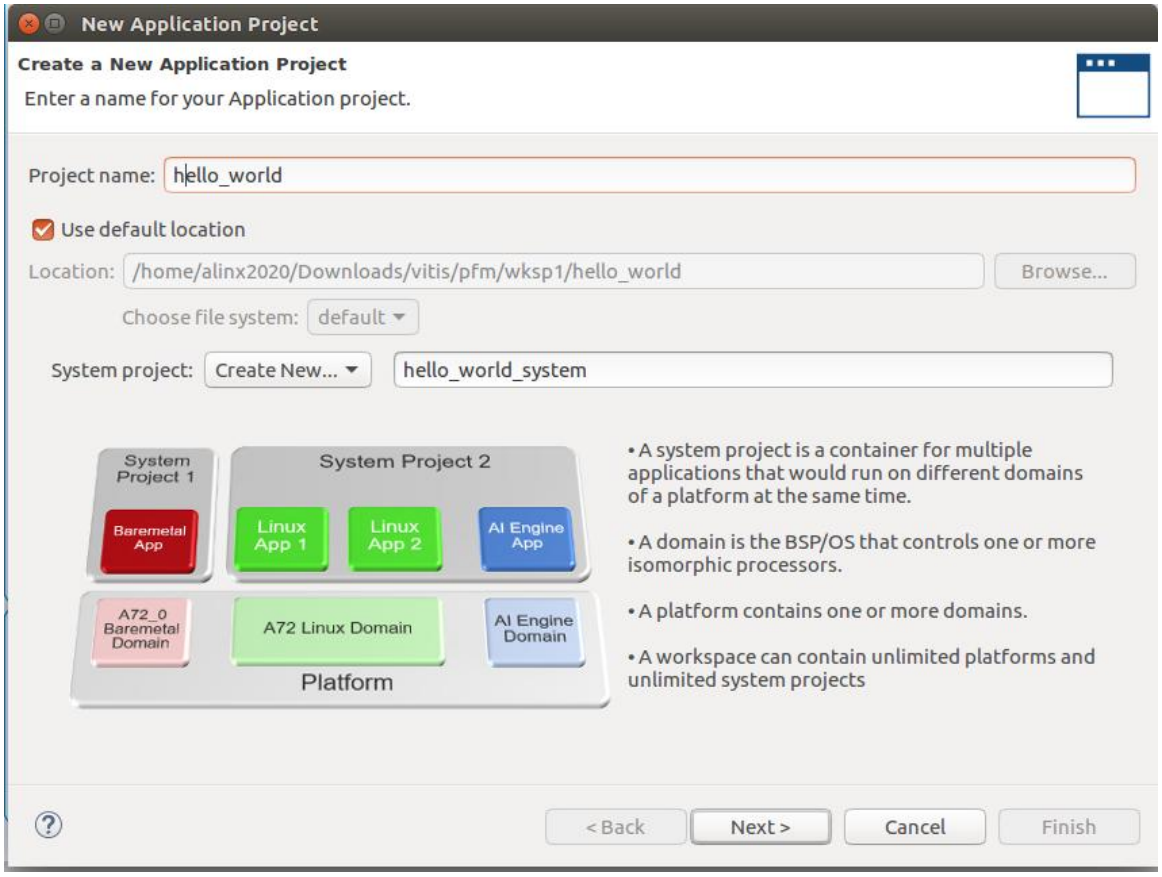
20)The bif file is specified in linux on psu_cortexa53, Boot Component

Directory and Linux image Directory are specified as the boot directory created above, Sysroot can be temporarily unspecified, if you want to specify, you can choose the petalinux sdk sysroots/aarch64-xilinx-linux folder after installation . Click the hammer at the top right to start the compilation and wait for the compilation to complete.

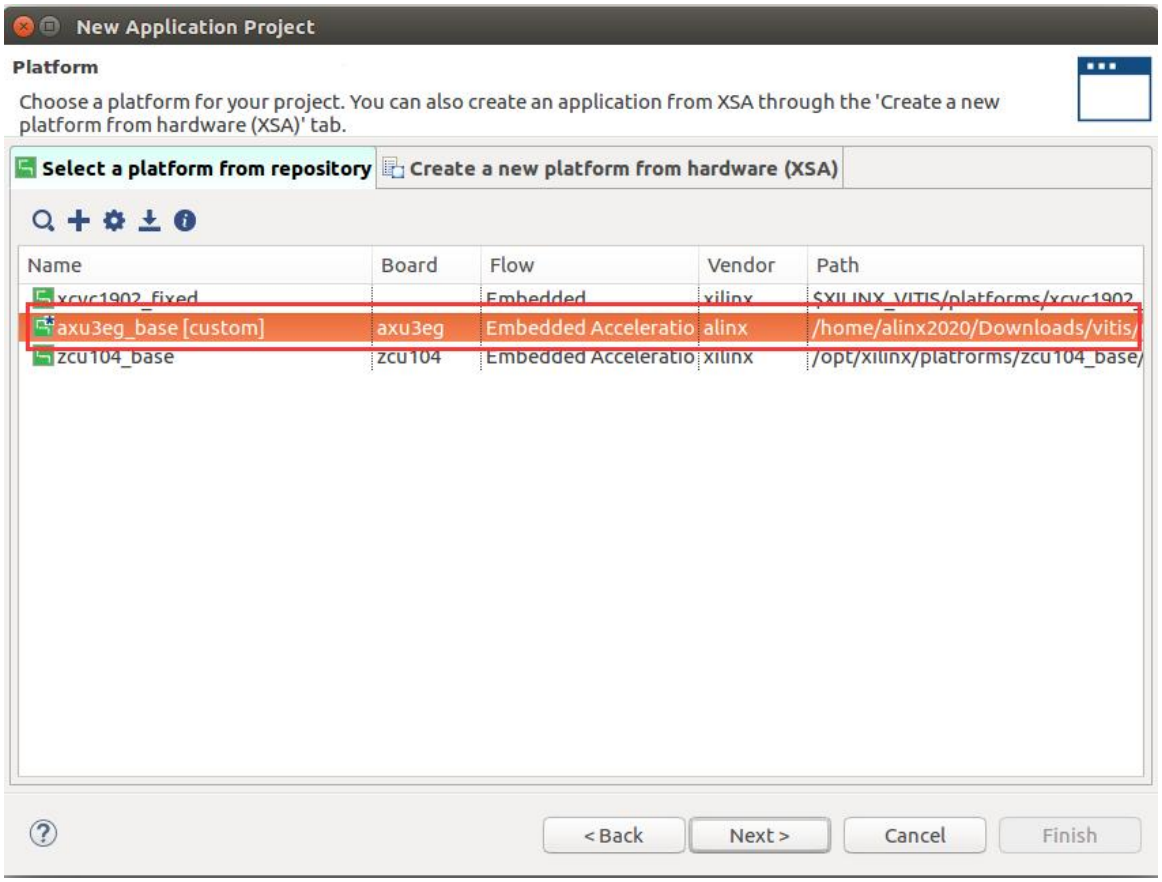


Part 8.3.3: Test Vitis Acceleration

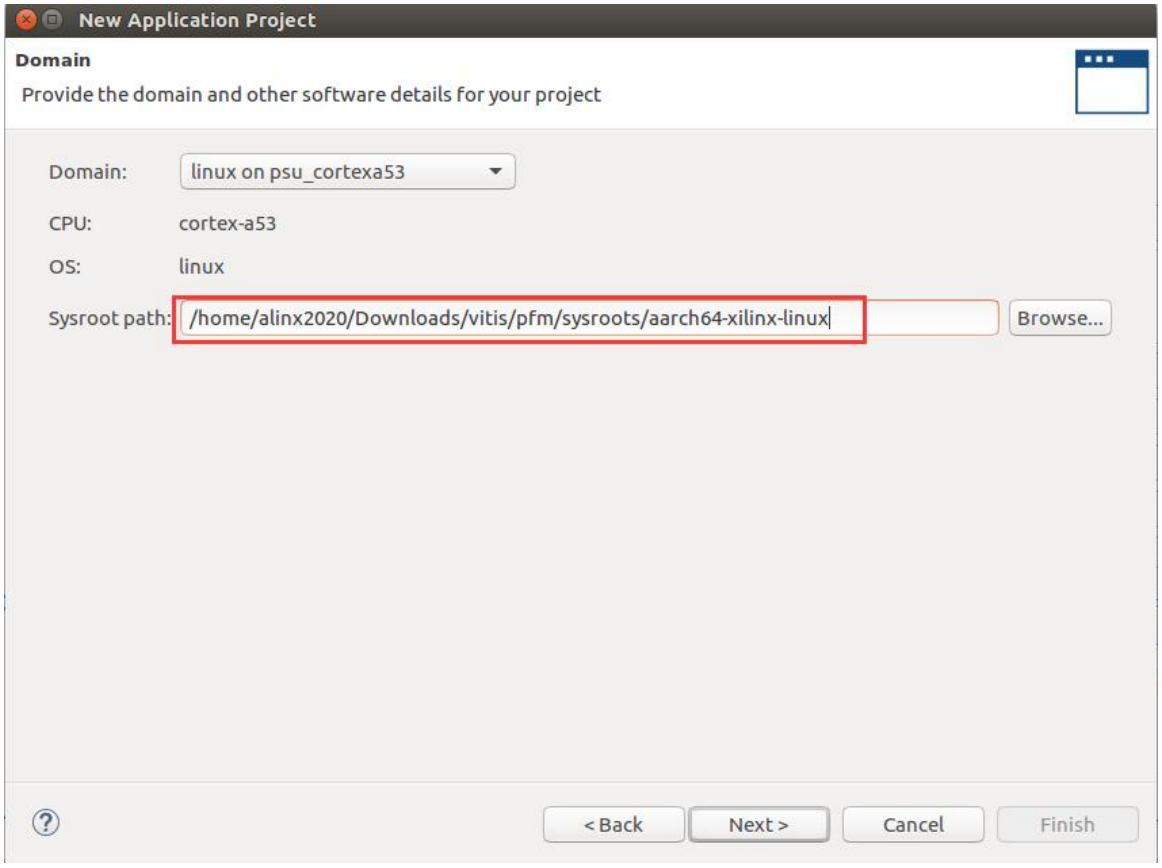
21)Select File→New→Application Project from the Vitis menu. Enter the project name hello_world and click Next



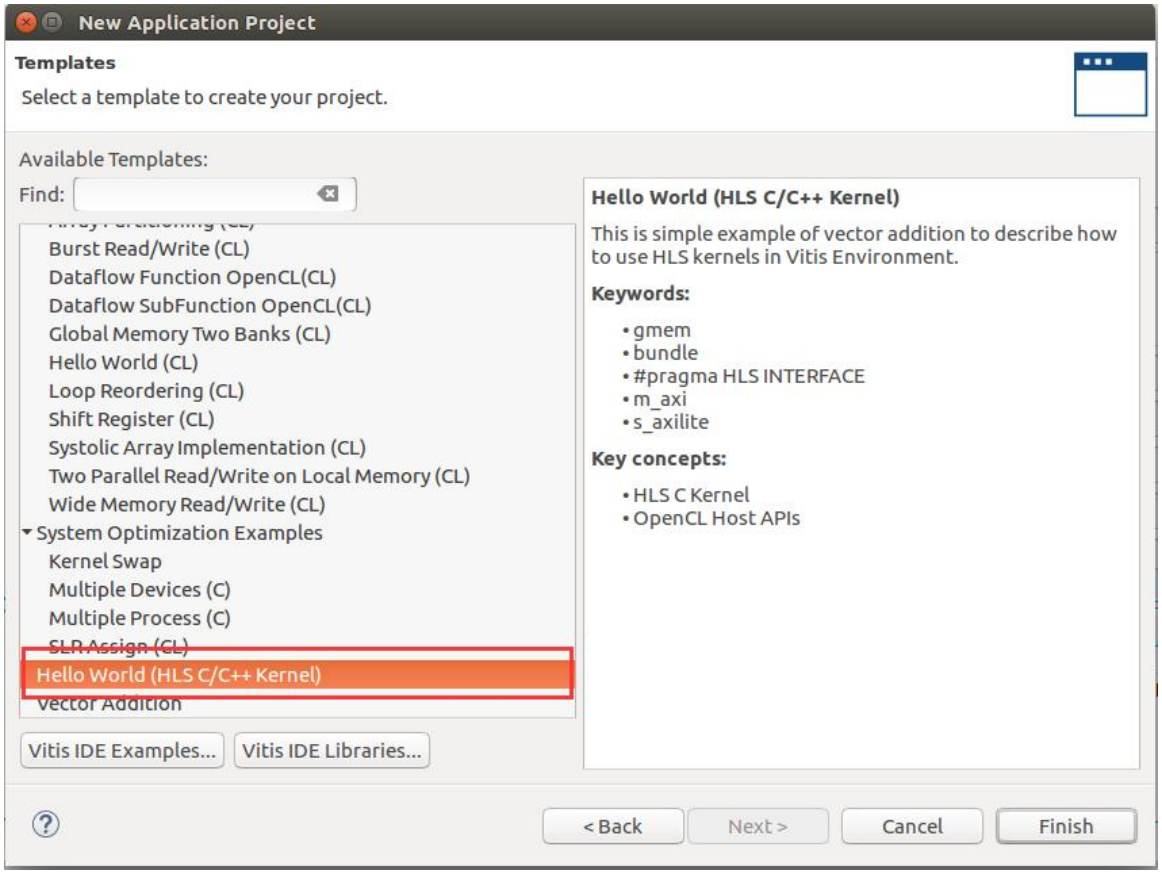
22)The platform selects the axu3eg_base just established



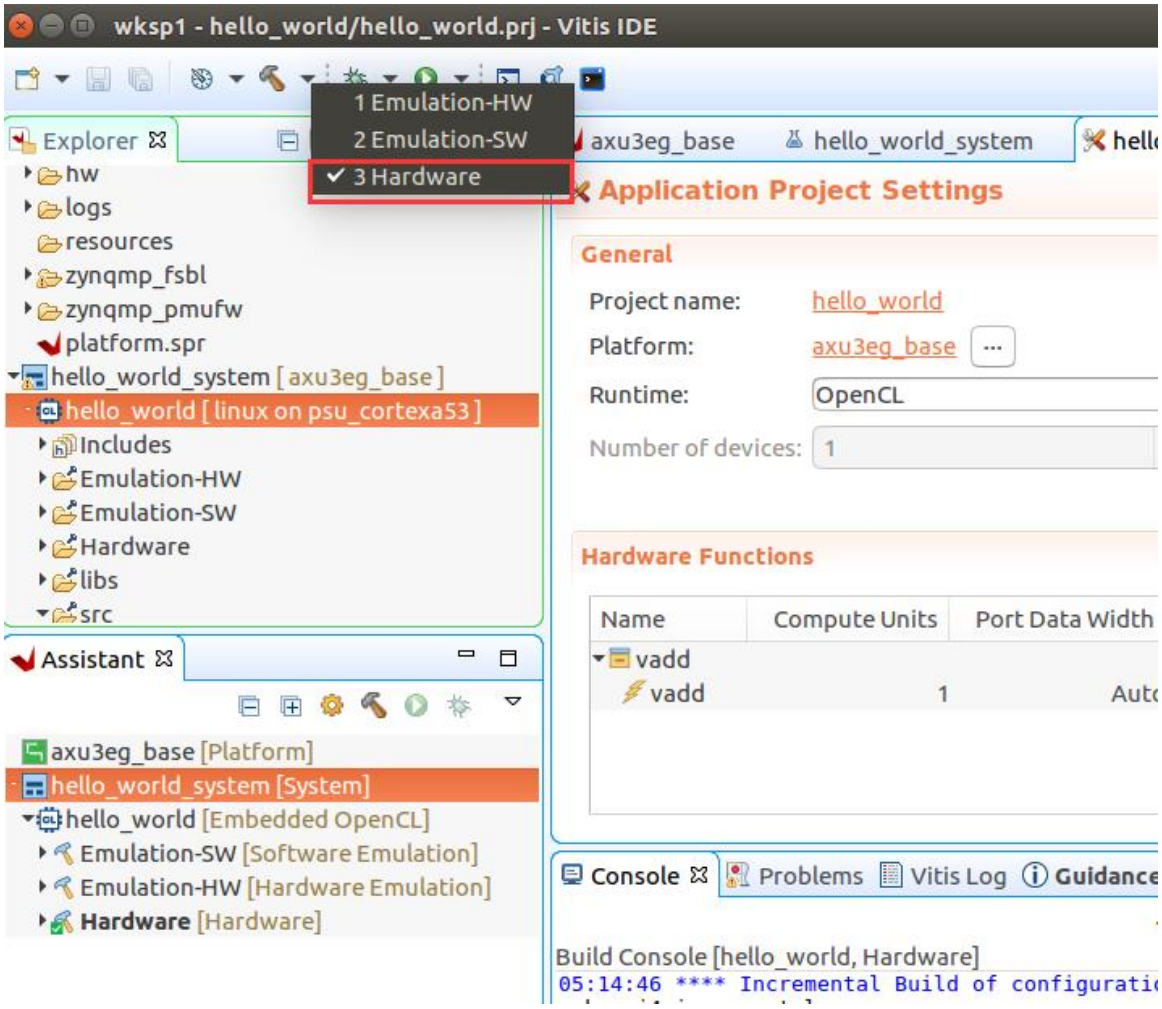
23)File system select the file system after petalinux sdk is installed, click Next



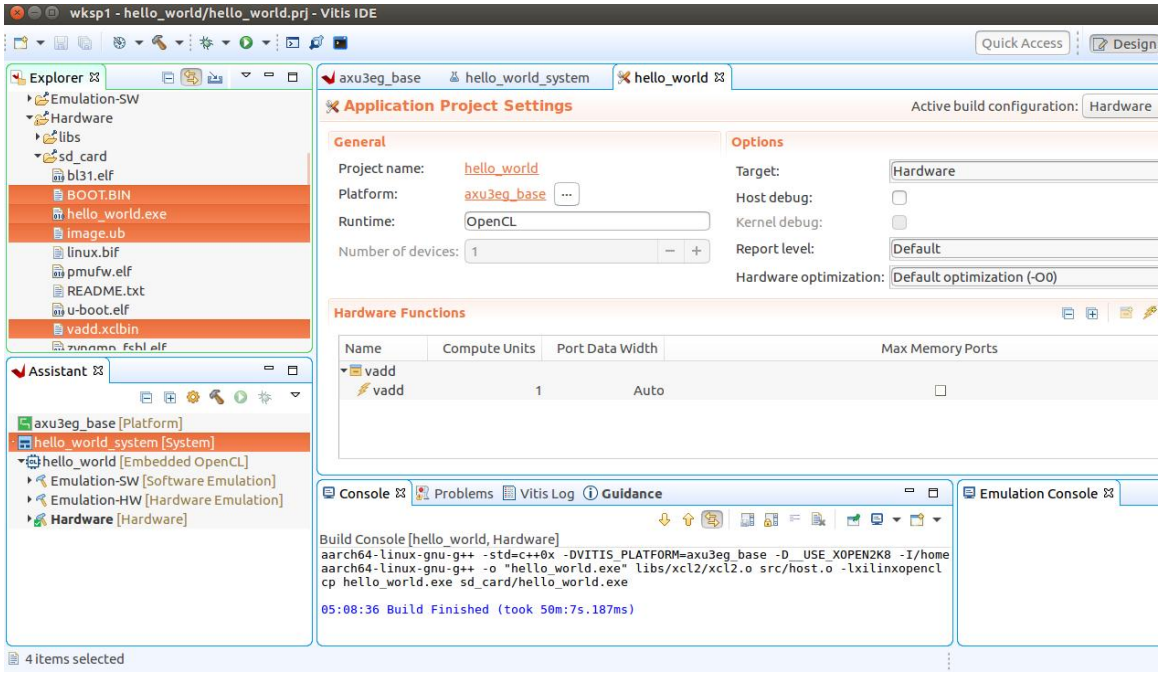
24)Choose Hello world as the template. Other templates may not be accelerated normally and need to be modified.



25) Select Hardware in the drop-down menu of the hammer, just to generate the file to run on the board, and then click the hammer to compile. It takes a long time and requires more than 16G of virtual machine memory, otherwise there will be an error without a reason.



26)After the compilation is complete, copy all the files in the sd_card folder to the first partition of the SD card. If there is no partition, please refer to "SD Card Root File System Making"



27) Delete all the contents in the second partition of the SD card, and then copy the root file system generated by petalinux, and replace the path with your own path

```
sudo tar xzvf <petalinux path>/images/linux/rootfs.tar.gz -C /media/$USER/rootfs
```

28) Insert the Sd card into the FPGA board, log in to the system using the serial port terminal, and run the test program ,export XILINX_XRT=/usr must be run

```
cd /media/sd-mmcblk1p1
export XILINX_XRT=/usr
./hello_world.exe vadd.xclbin
```

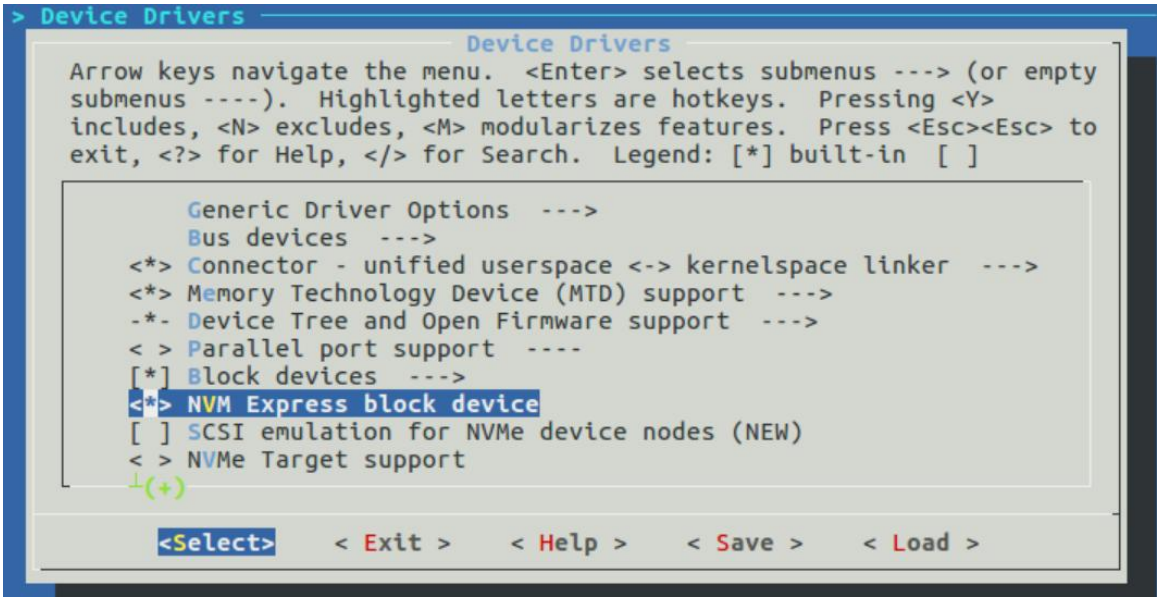
Part 9: NVMe SSD operation under Linux

Part 9.1: petalinux Configuration

1) The PCIe driver is configured by default, and you only need to configure the NVMe driver. Enter in the petalinux project directory.

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

2) Select <*> NVM Express block device in Device Drivers.



3) Recompile the petalinux project after saving and exiting.

Part 9.2: Set up partition

1) Plug in the SSD and start the board, run the following command to check if the SSD device is loaded normally.

```
ls /dev/nvme*
```

```
root@petalinux:~# ls /dev/nvme*
/dev/nvme0     /dev/nvme0n1
```

2) Operate /dev/nvme0n1 to set the partition for the SSD.

fdisk /dev/nvme0n1

```
root@petalinux:~# fdisk /dev/nvme0n1

The number of cylinders for this disk is set to 122104.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help):
```

Enter n to create a new partition, and then enter p, 1, 32, and +80G to create a primary partition.

```
Command (m for help): n
Partition type
   p   primary partition (1-4)
   e   extended
p
Partition number (1-4): 1
First sector (32-250069679, default 32):
Using default value 32
Last sector or +size{,K,M,G,T} (32-250069679, default 250069679): +80G
```

3) Enter wq to save the settings and exit.

```
Command (m for help): wq
The partition table has been altered.
Calling ioctl() to re-read partition table
[ 1357.268018]  nvme0n1: p1
```

4) Use the following command to make a partition file system (format partition).

mkfs -t ext4 /dev/nvme0n1p1

```
root@petalinux:~# mkfs -t ext4 /dev/nvme0n1p1
mke2fs 1.45.3 (14-Jul-2019)
/dev/nvme0n1p1 contains a vfat file system
Proceed anyway? (y,N) y
Discarding device blocks: done
Creating filesystem with 20971520 4k blocks and 5242880 inodes
Filesystem UUID: 0bd321dc-75db-447b-8b96-b987ab2993cc
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
        4096000, 7962624, 11239424, 20480000

Allocating group tables: done
Writing inode tables: done
Creating journal (131072 blocks): done
Writing superblocks and filesystem accounting information: done

root@petalinux:~#
```

5) Use the mount command to mount the partition.

```
mount /dev/nvme0n1p1 /run/media/nvme0n1p1
```

```
root@petalinux:~# mount /dev/nvme0n1p1 /run/media/nvme0n1p1
[  322.273973] EXT4-fs (nvme0n1p1): mounted filesystem with ordered data mode. Opts: (null)
root@petalinux:~#
```

Part 9.3: Read and write speed

After the SSD is formatted and mounted, you can use the dd command to read and write speed measurement.

1) cd into the mounted path

```
cd /run/media/nvme0n1p1
```

2) Use the dd command to copy the file under the path, and the time-consuming and average read and write speed will be printed out after completion.

```
dd if=./testfile1 of=./testfile2 bs=100k
```

```
root@petalinux:/run/media/nvme0n1p1# dd if=./testfile1 of=./testfile2 bs=100k
51+1 records in
51+1 records out
5308914 bytes (5.3 MB, 5.1 MiB) copied, 0.0204968 s, 259 MB/s
```

3) Sequential write test

```
dd if=/dev/zero of=./testfile1 bs=100k count=1310
```

```
root@petalinux:/run/media/nvme0n1p1# dd if=/dev/zero of=./testfile1 bs=100k count=1310
1310+0 records in
1310+0 records out
134144000 bytes (134 MB, 128 MiB) copied, 0.470858 s, 285 MB/s
root@petalinux:/run/media/nvme0n1p1#
```

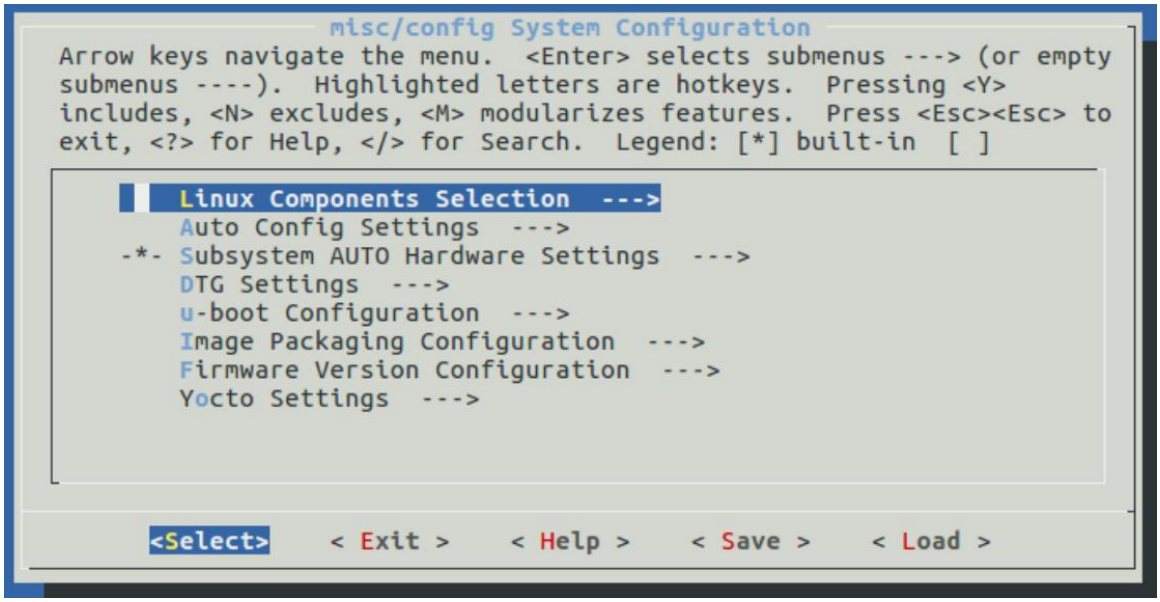
4) Sequential reading test

```
dd if=./testfile1 of=/dev/null bs=100k
```

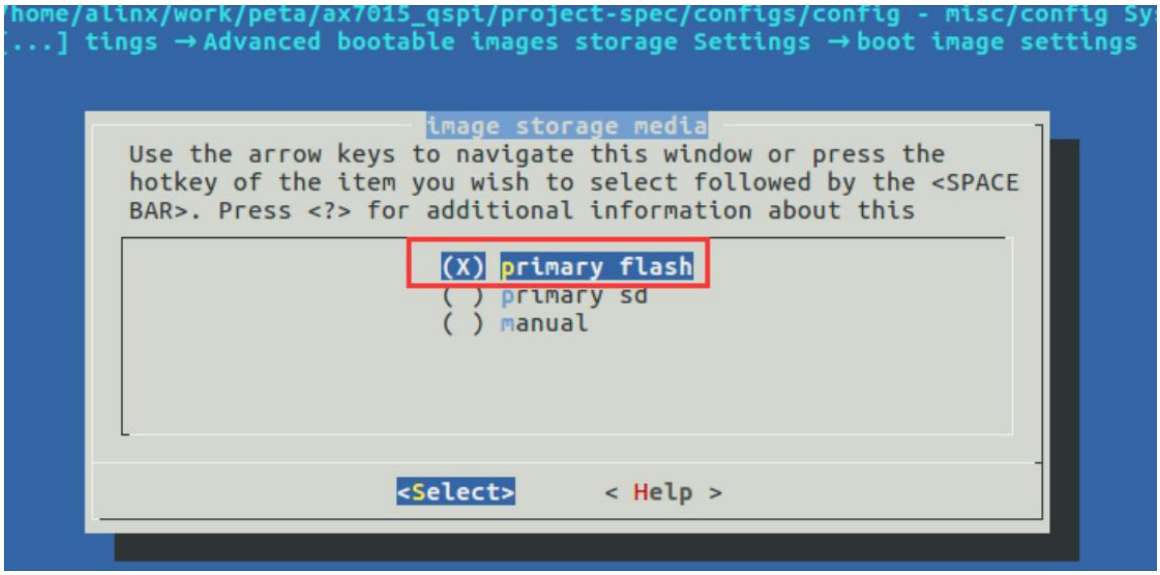
```
root@petalinux:/run/media/nvme0n1p1# dd if=./testfile1 of=/dev/null bs=100k
51+1 records in
51+1 records out
5308914 bytes (5.3 MB, 5.1 MiB) copied, 0.00316914 s, 1.7 GB/s
root@petalinux:/run/media/nvme0n1p1#
```

Part 10: Boot Linux from QSPI Flash

1) Configure “Petalinux” with the “petalinux-config” command



2) Select “primary flash” in “Subsystem AUTO Hardware Settings --->Advanced bootable images storage Settings --->boot image settings --->image storage media” option



3) Select “primary flash” in “Subsystem AUTO Hardware Settings ---> Advanced bootable images storage Settings ---> kernel image settings ---> image storage media” option

```
/home/alinx/work/peta/ax7015_qspi/project-spec/configs/config - misc/config Sys
[...] ngs → Advanced bootable images storage Settings → kernel image settings
                          kernel image 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  [ ]

      image storage media (primary flash)  --->
      (kernel) flash partition name
      (image.ub) image name

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

4) Subsystem AUTO Hardware Settings → Flash Settings" can modify the "QSPI flash" partition.

```
/home/alinx/Downloads/linuxBase4ev_qsf/petalinux/project-spec/configs/config - misc/config System Configurat
i→ Subsystem AUTO Hardware Settings → Flash Settings
                          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 ***
            (bootenv) name
            (0x10000) size
                *** partition 2 ***
            (kernel) name
            (0x1500000) size
                *** partition 3 ***
            (bootscr) name
            (0x10000) size
                *** partition 4 ***
            ()  name

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

The boot partition stores .bit files describing FPGA, u-boot, fsbl, device tree .dtb and other files. The size of the boot partition only needs to be larger than the sum of the sizes of these files, and it needs to be modified according to the actual situation.

The bootenv partition stores u-boot environment variables, which

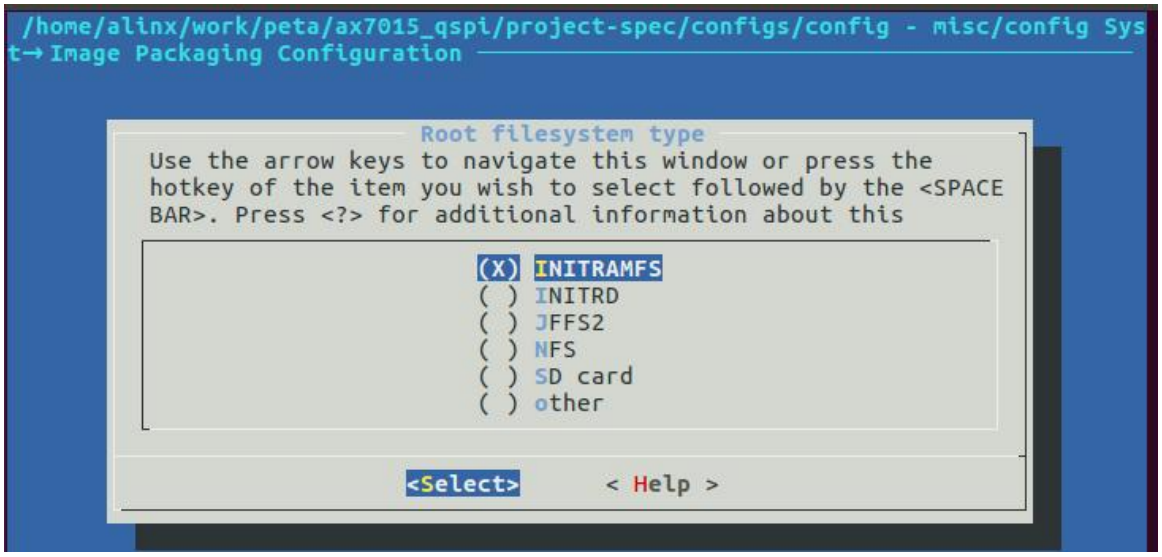
are modified according to actual conditions.

The kernel partition stores image.ub. The partition is larger than image.ub and needs to be modified according to the actual situation.

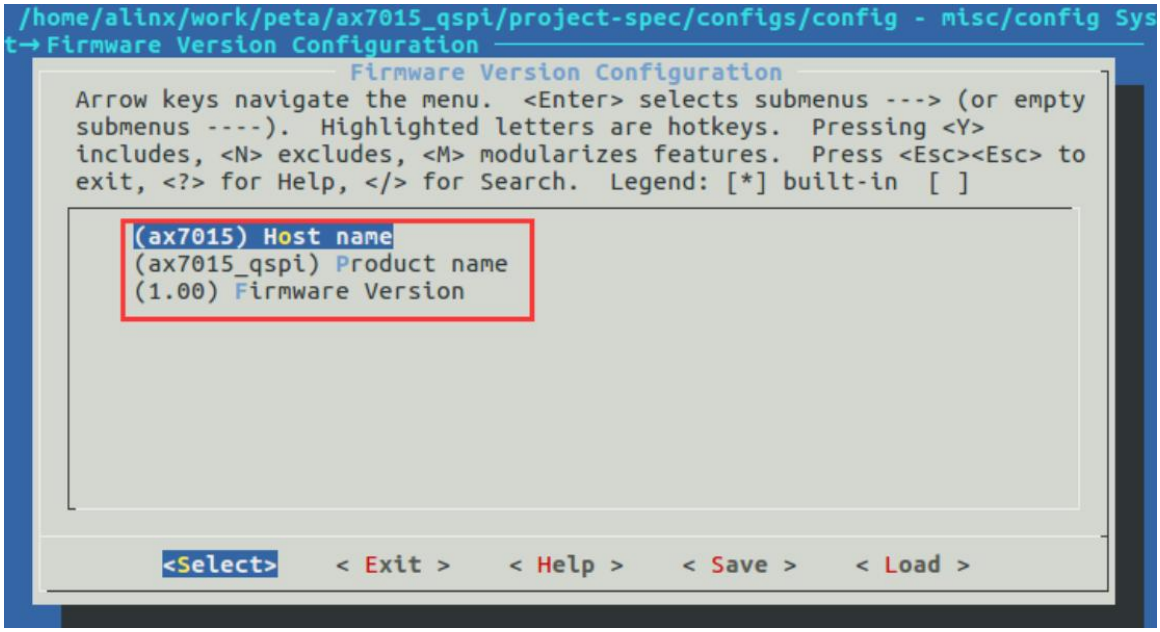
The boot.scr is stored in the bootscr partition. The partition is larger than boot.scr and needs to be modified according to the actual situation.

Also note that the total size of all packaged files cannot exceed the size of flash.

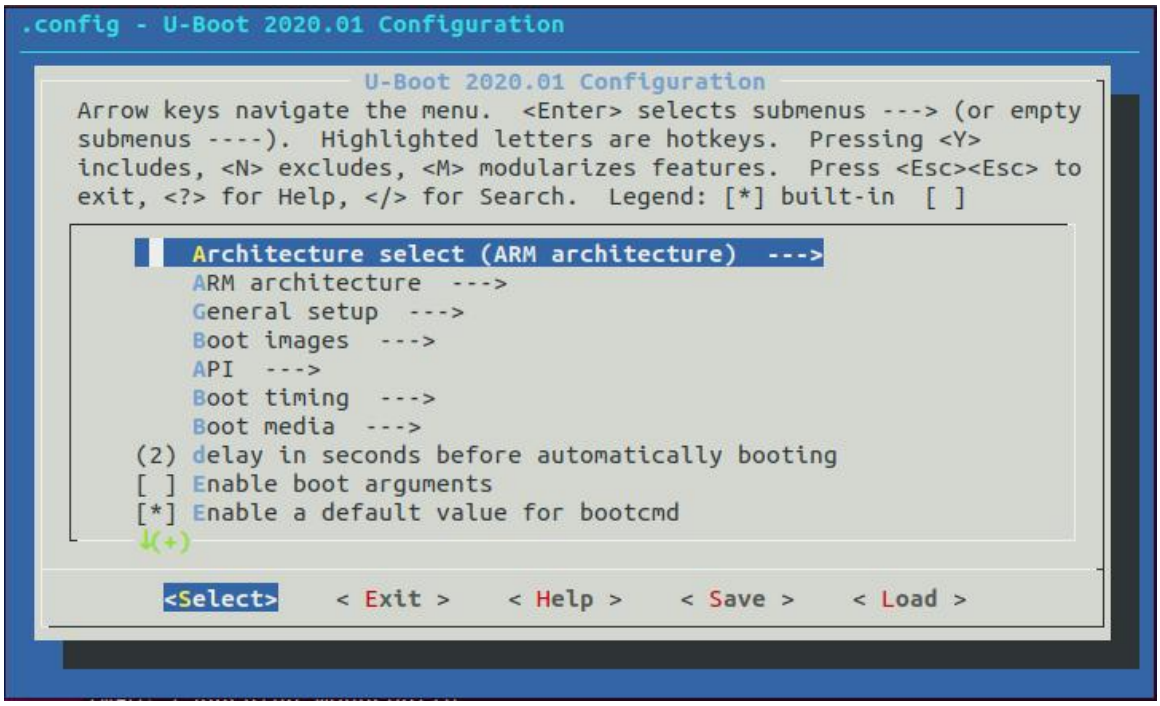
5) In the “Image Packaging Configuration ---> Root filesystem type” selection “INITRAMFS”, use “RAM” type of root file system, so you can easily pack programmed into the “QSPI Flash”



6) You can modify the “Host name” in “Firmware Version Configuration --->”



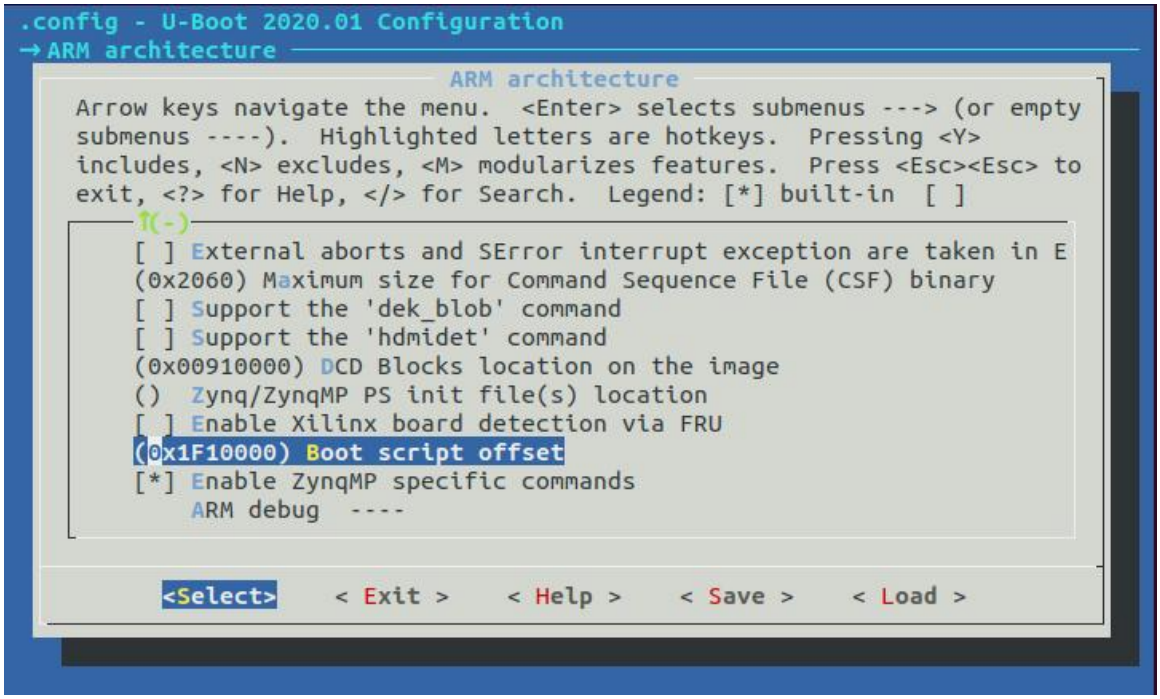
7) Save the configuration and exit. Then use petalinux-config -c u-boot command to configure u-boot



8) Enter ARM architecture ---> Boot script offset and set the offset address of the bootscr partition.

Offset address = boot + bootenv + kernel

0x1F10000 = 0xA00000 + 0x10000 + 0x1500000.



9) Open project-spec/meta-user/recipes-bsp/u-boot/u-boot-zynq-scr.bbappend under the petalinux project path. Modify the kernel offset address and kernel size to correspond to the settings in step 4.

```
u-boot-zynq-scr.bbappend
~/Downloads/linuxBase4ev_qsf/petalinu...ject-spec/meta-user/recipes-bsp/u-boot
QSPI_RAMDISK_SIZE_zynq = "0xA00000"

NAND_KERNEL_SIZE = "0x3200000"
NAND_RAMDISK_SIZE = "0x3200000"

## Below offsets and sizes are based on 128MB QSPI Memory for zynqmp/versal
## For zynqMP
## Load boot.scr at 0x3E80000 -> 62MB of QSPI/NAND Memory
QSPI_KERNEL_OFFSET = "0xA10000"
QSPI_KERNEL_OFFSET_zynqmpdr = "0x3F00000"
QSPI_RAMDISK_OFFSET = "0x4000000"
QSPI_RAMDISK_OFFSET_zynqmpdr = "0x5D00000"

NAND_KERNEL_OFFSET_zynqmp = "0x4100000"
NAND_RAMDISK_OFFSET_zynqmp = "0x7800000"

QSPI_KERNEL_SIZE_zynqmp = "0x1D00000"
QSPI_RAMDISK_SIZE = "0x4000000"
QSPI_RAMDISK_SIZE_zynqmpdr = "0x1D00000"

## For versal
## Load boot.scr at 0x7F80000 -> 127MB of QSPI/NAND Memory
QSPI_KERNEL_OFFSET_versal = "0xF00000"
QSPI_RAMDISK_OFFSET_versal = "0x2E00000"

NAND_KERNEL_OFFSET_versal = "0x4100000"
NAND_RAMDISK_OFFSET_versal = "0x8200000"

QSPI_KERNEL_SIZE_versal = "0x1D00000"
QSPI_RAMDISK_SIZE_versal = "0x4000000"

QSPI_KERNEL_IMAGE_zynq = "image.ub"
QSPI_KERNEL_IMAGE_zynqmp = "image.ub"
QSPI_KERNEL_IMAGE_versal = "image.ub"

NAND_KERNEL_IMAGE = "image.ub"

FIT_IMAGE_LOAD_ADDRESS = "${@append_baseaddr(d,"0x10000000")}"

QSPI_FIT_IMAGE_LOAD_ADDRESS = "${@append_baseaddr(d,"0x10000000")}"
QSPI_FIT_IMAGE_SIZE = "0x1500000"
QSPI_FIT_IMAGE_SIZE_zynqmpdr = "0x3F00000"
QSPI_FIT_IMAGE_SIZE_zynq = "0xF00000"

NAND_FIT_IMAGE_LOAD_ADDRESS = "${@append_baseaddr(d,"0x10000000")}"
Plain Text   Tab Width: 8   Ln 53, Col 32   INS
```

10)After saving, use petalinux-build command to compile the project.

11)After the compilation is successful, use the following command to synthesize BOOT, and use the --kernel option to compile the kernel to BOOT.bin. Use the --add option to compile boot.scr into the kernel. Note that the offset address needs to be set through the --offset option after boot.scr, which corresponds to the setting in step 8.

```
petalinux-package --boot --fsbl ./images/linux/zynqmp_fsbl.elf --fpga --u-boot --kernel
--add ./images/linux/boot.scr --offset 0x01F10000    --force
```

```
alinx@ubuntu:~/Downloads/linuxBase4ev_qsf/petalinux$ petalinux-package --boot --fsbl ./images/linux/zynqmp_fs
bl.elf --fpga --u-boot --kernel --add ./images/linux/boot.scr --offset 0x01F10000  --force
INFO: sourcing build tools
INFO: Getting system flash information...
INFO: File in BOOT BIN: "/home/alinx/Downloads/linuxBase4ev_qsf/petalinux/images/linux/zynqmp_fsbl.elf"
INFO: File in BOOT BIN: "/home/alinx/Downloads/linuxBase4ev_qsf/petalinux/images/linux/pmufw.elf"
INFO: File in BOOT BIN: "/home/alinx/Downloads/linuxBase4ev_qsf/petalinux/project-spec/hw-description/design_
1_wrapper.bit"
INFO: File in BOOT BIN: "/home/alinx/Downloads/linuxBase4ev_qsf/petalinux/images/linux/bl31.elf"
INFO: File in BOOT BIN: "/home/alinx/Downloads/linuxBase4ev_qsf/petalinux/images/linux/system.dtb"
INFO: File in BOOT BIN: "/home/alinx/Downloads/linuxBase4ev_qsf/petalinux/images/linux/u-boot.elf"
INFO: File in BOOT BIN: "/home/alinx/Downloads/linuxBase4ev_qsf/petalinux/images/linux/image.ub"
INFO: File in BOOT BIN: "/home/alinx/Downloads/linuxBase4ev_qsf/petalinux/images/linux/boot.scr"
INFO: Generating zynqmp binary package BOOT.BIN...


****** Xilinx Bootgen v2020.1
  **** Build date : May 26 2020-14:07:15
    ** Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.


[INFO]   : Bootimage generated successfully

INFO: Binary is ready.
WARNING: Unable to access the TFTPBOOT folder /tftpboot!!!
WARNING: Skip file copy to TFTPBOOT folder!!!
```

12) Refer to the method in the vitis tutorial to burn BOOT.bin into the FPGA development board.

13) Set the DIP switch on the FPGA development board to QSPI startup, and then power on the FPGA development board.

Part 11: Boot Linux from EMMC

EMMC and SD card are similar, but unlike SD card, EMMC can be directly formatted on the host computer and copied into the system. So we need to use other methods to start a system (such as booting from SD card, booting from Qflash), format and copy system related files in this system, and then adjust the DIP switch to set the startup mode, restart the FPGA development board .

Use the SD card to make the system refer to Part1 or Part 5.

Use Qflash to start the system refer to Part 10.

Part 11.1: Format EMMC and Mount

To format EMMC, refer to the SSD operation in Part 9,

The mmc device is generally "/dev/mmcblk*", which has the same format as the device name of the SD card. As for which is the SD card, you need to identify it yourself.

The difference with SSD is that after the partition is completed, the SSD uses the mkfs command to format the partition into EXT4 format, but here we need to format the EMMC into FAT32. After the partition is completed, do the following operations

```
root@petalinux:~# fdisk /dev/mmcblk0

Welcome to fdisk (util-linux 2.34).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.


Command (m for help): t
Selected partition 1
Hex code (type L to list all codes): L

 0  Empty           24  NEC DOS         81  Minix / old Lin bf  Solaris
 1  FAT12           27  Hidden NTFS Win 82  Linux swap / So c1  DRDOS/sec (FAT-
 2  XENIX root      39  Plan 9          83  Linux           c4  DRDOS/sec (FAT-
 3  XENIX usr       3c  PartitionMagic  84  OS/2 hidden or  c6  DRDOS/sec (FAT-
 4  FAT16 <32M      40  Venix 80286     85  Linux extended  c7  Syrinx
 5  Extended        41  PPC PReP Boot   86  NTFS volume set da  Non-FS data
 6  FAT16           42  SFS             87  NTFS volume set db  CP/M / CTOS / .
 7  HPFS/NTFS/exFAT 4d  QNX4.x          88  Linux plaintext de  Dell Utility
 8  AIX             4e  QNX4.x 2nd part 8e  Linux LVM       df  BootIt
 9  AIX bootable    4f  QNX4.x 3rd part 93  Amoeba          e1  DOS access
 a  OS/2 Boot Manag 50  OnTrack DM      94  Amoeba BBT      e3  DOS R/O
 b  W95 FAT32       51  OnTrack DM6 Aux 9f  BSD/OS          e4  SpeedStor
 c  W95 FAT32 (LBA) 52  CP/M            a0  IBM Thinkpad hi ea  Rufus alignment
 e  W95 FAT16 (LBA) 53  OnTrack DM6 Aux a5  FreeBSD         eb  BeOS fs
 f  W95 Ext'd (LBA) 54  OnTrackDM6      a6  OpenBSD         ee  GPT
10  OPUS            55  EZ-Drive        a7  NeXTSTEP        ef  EFI (FAT-12/16/
11  Hidden FAT12    56  Golden Bow      a8  Darwin UFS      f0  Linux/PA-RISC b
12  Compaq diagnost 5c  Priam Edisk     a9  NetBSD          f1  SpeedStor
14  Hidden FAT16 <3 61  SpeedStor       ab  Darwin boot     f4  SpeedStor
16  Hidden FAT16    63  GNU HURD or Sys af  HFS / HFS+      f2  DOS secondary
17  Hidden HPFS/NTF 64  Novell Netware  b7  BSDI fs         fb  VMware VMFS
18  AST SmartSleep  65  Novell Netware  b8  BSDI swap       fc  VMware VMKCORE
1b  Hidden W95 FAT3 70  DiskSecure Mult bb  Boot Wizard hid fd  Linux raid auto
1c  Hidden W95 FAT3 75  PC/IX           bc  Acronis FAT32 L fe  LANstep
1e  Hidden W95 FAT1 80  Old Minix       be  Solaris boot    ff  BBT
Hex code (type L to list all codes): b
Changed type of partition 'W95 FAT32' to 'W95 FAT32'.

Command (m for help): wq
The partition table has been altered.
Syncing disks.

root@petalinux:~#
```

If you get an error when you select W95 FAT32, you can first try to use the mkfs command to format the partition to EXT4 and then perform the above operation

After formatting is complete, enter the mount command to view the status of EMMC hanging. If the EMMC partition is not mounted, follow the method in Part 9 to mount EMMC to a folder.

Part 11.2: Mount the Virtual Machine Folder using NFS

We need to make the system started by EMMC on the virtual machine. In order to facilitate the copying of the system files required by EMMC, first use NFS to hang a folder of the virtual machine to the

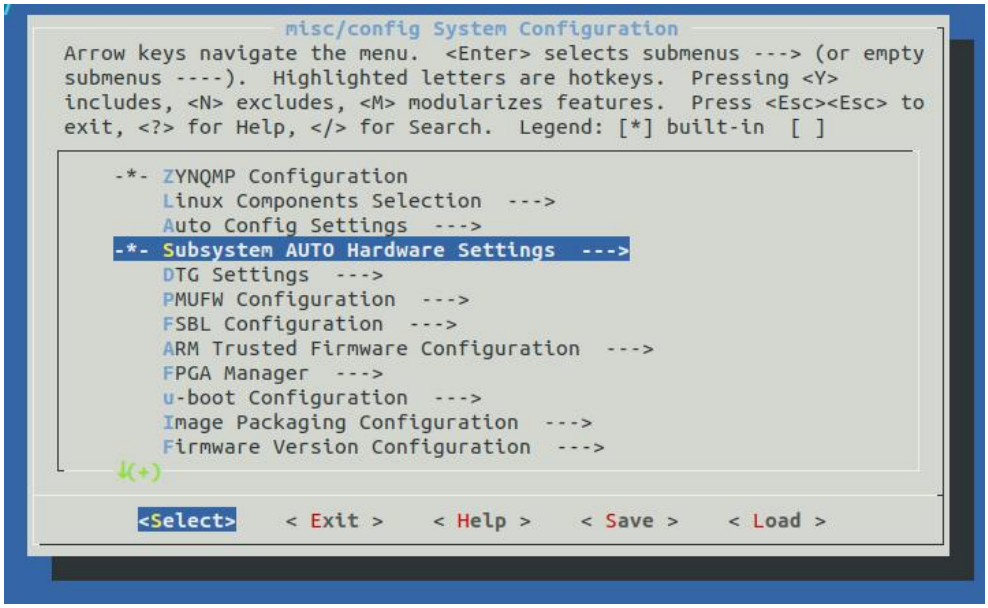
system of the FPGA development board.

The NFS Operation refer to Part 7.2.

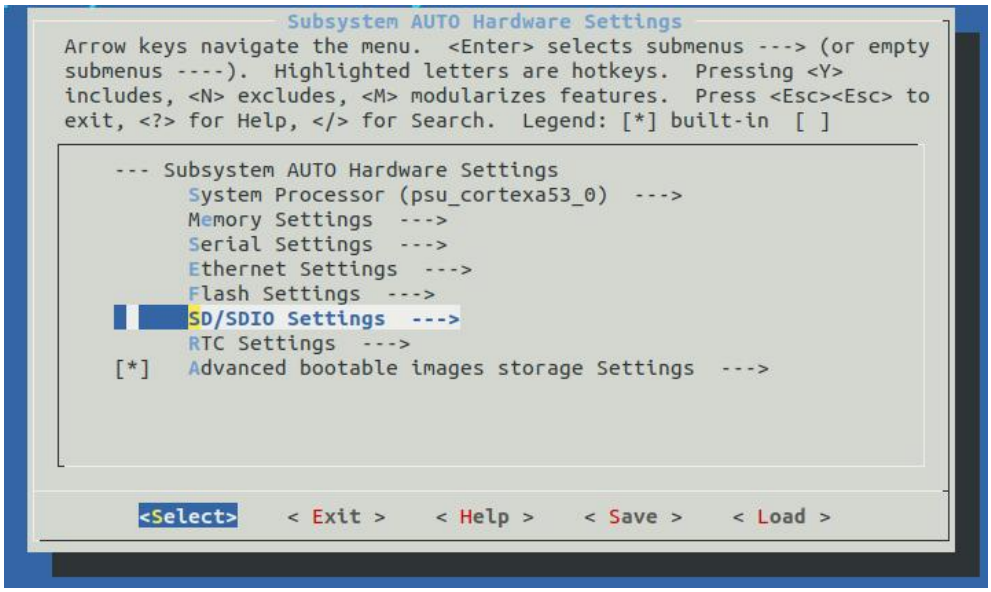
Part 11.3: Make EMMC boot system

EMMC boot and SD card boot are almost the same. Only one setting in petalinux needs to be modified based on the SD boot system.

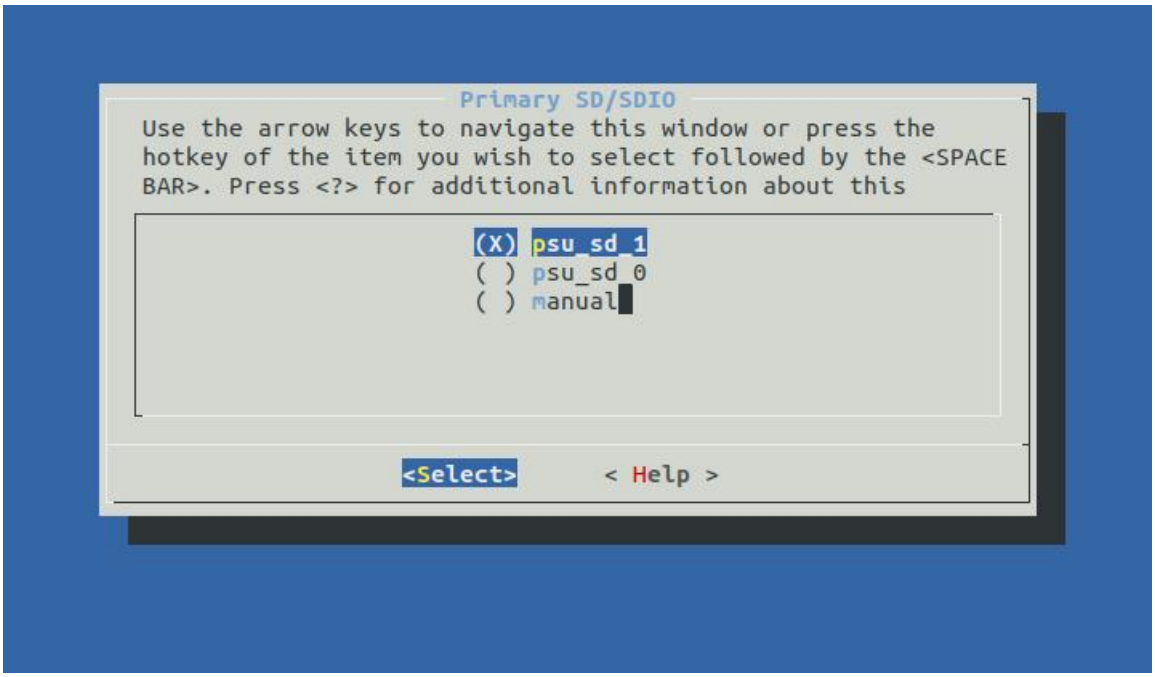
1) Enter petalinux-config
2) Select Subsystem AUTO Hardware Settings --->



3) Select SD/SDIO Settings --->



4) Select the SDIO corresponding to EMMC (try it if you are not sure)



5) Save and exit the configuration interface, and recompile the system.

6) Copy the three files in the following figure obtained by compilation to the folder where the NFS of the development board is hung

7) Copy these three files in the system of the FPGA development board to the directory mounted by EMMC.

8) After the copy is complete, use the “umount” command to unmount the EMMC.

9) Disconnect the power supply of the FPGA development board, adjust the DIP switch to the state of EMMC startup, and then power on again.