

DevKit Develop Enviroment For Ubuntu16.04

DevKit Develop Enviroment For Ubuntu16.04

1.Install Vivado

- 1.1 Download
- 1.2 Install Main Program
- 1.3 Install cable driver
- 1.4 Add Vivado SDK to Dash

2.Build Minimum System Hardware

- 2.1 Install Board Interface File
- 2.2 Create Project
- 2.3 Create Minimum System Design
- 2.4 Export hdf(hardware definition file) file

3.Run AMP(Asymmetric Multi-Processing System) on Devkit

- 3.1 Principle
- 3.2 Configure SDK
- 3.3 Create FSBL Application
- 3.4 Create Bare-Metal Application For CPU1
- 3.5 u-boot For CPU0
- 3.6 Linux Kernel For CPU0
- 3.7 Linux Devicetree
- 3.8 Linux Filesystem
- 3.9 Create BOOT.bin
- 3.10 Test

1.Install Vivado

1.1 Download

Our Vivado version is 2016.4 (Download here:<https://www.xilinx.com/support/download.html>)

Get Phenix Pro Devkit Firmware from github:

```
git clone https://github.com/RobSenseTech/PhenixPro_Devkit
```

1.2 Install Main Program

First of all, we need to install **the latest version**(8u121) of Java SE Development Kit(Download here <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>), then install it:

```
cd [your download path]
tar xf jdk-8u121-linux-x64.tar.gz
vim ~/.bashrc
```

add these statements:

```
export JAVA_HOME=[your jdk path]
export JRE_HOME=${JAVA_HOME}/jre
export CLASSPATH=.:{JAVA_HOME}/lib:${JRE_HOME}/lib
export PATH=${JAVA_HOME}/bin:$PATH
```

use this command to make it work:

```
source ~/.bashrc
```

to confirm whether jdk is installed properly, use this command:

```
java -version
```

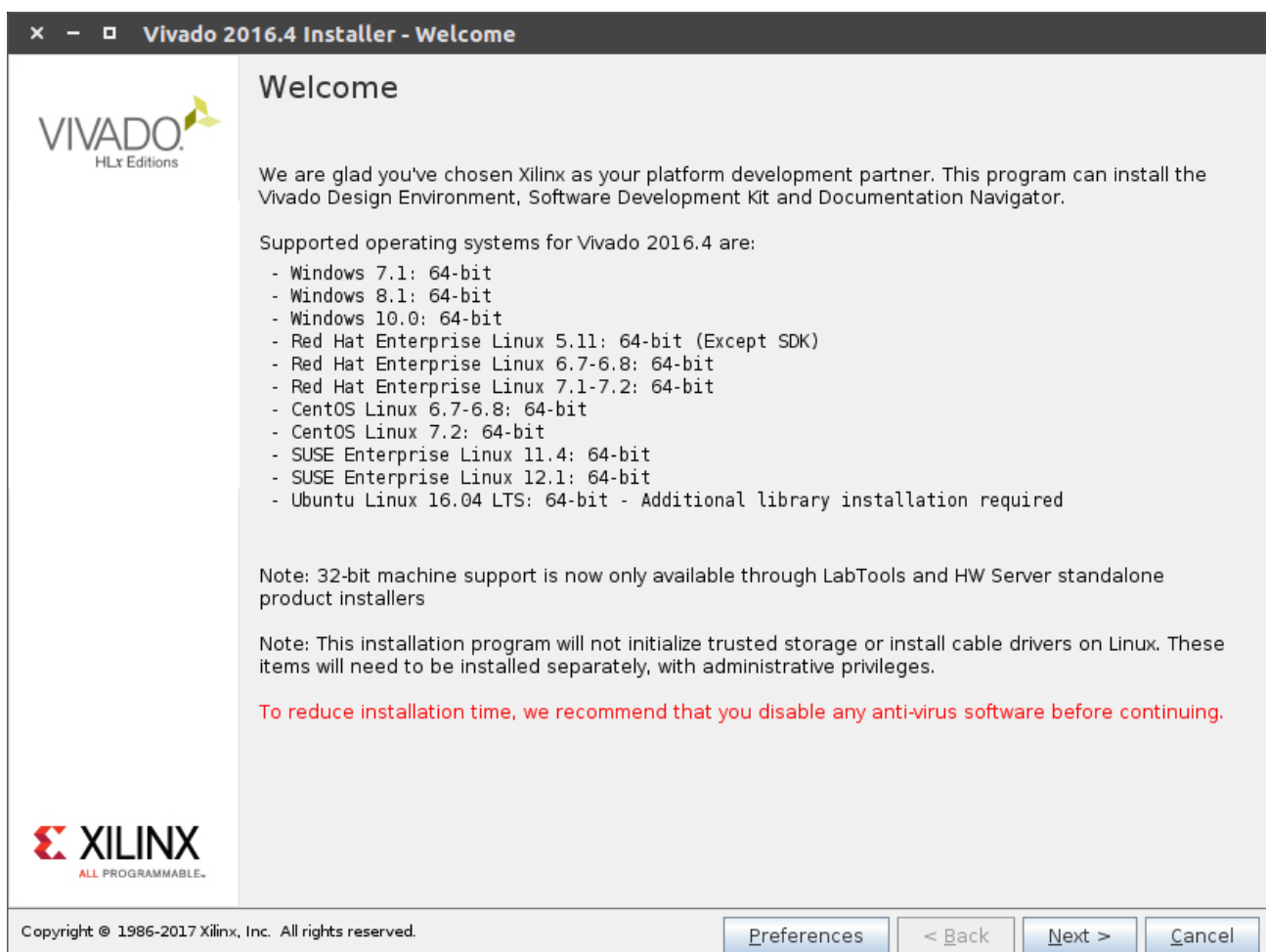
if JDK is installed properly, it will show these information:

```
java version "1.8.0_121"  
Java(TM) SE Runtime Environment (build 1.8.0_121-b13)  
Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed  
mode)
```

now, we can install Vivado:

```
cd [your vivado path]  
tar xf Xilinx_Vivado_SDK_2016.4_0124_1.tar.gz  
cd Xilinx_Vivado_SDK_2016.4_0124_1  
sudo ./xsetup
```

click Next



select three "I Agree" and Next

Vivado 2016.4 Installer - Accept License Agreements

Accept License Agreements

Please read the following terms and conditions and indicate that you agree by checking the I Agree checkboxes.

Xilinx Inc. End User License Agreement

By checking "I AGREE" below, or OTHERWISE ACCESSING, DOWNLOADING, INSTALLING or USING THE SOFTWARE, YOU AGREE on behalf of licensee to be bound by the agreement, which can be viewed by [clicking here](#).

☒ **I Agree**

WebTalk Terms And Conditions

By checking "I AGREE" below, I also confirm that I have read [Section 13 of the terms and conditions](#) above concerning WebTalk and have been afforded the opportunity to read the WebTalk FAQ posted at <http://www.xilinx.com/webtalk>. I understand that I am able to disable WebTalk later if certain criteria described in Section 13(c) apply. If they don't apply, I can disable WebTalk by uninstalling the Software or using the Software on a machine not connected to the internet. If I fail to satisfy the applicable criteria or if I fail to take the applicable steps to prevent such transmission of information, I agree to allow Xilinx to collect the information described in Section 13(a) for the purposes described in Section 13(b).

☒ **I Agree**

Third Party Software End User License Agreement

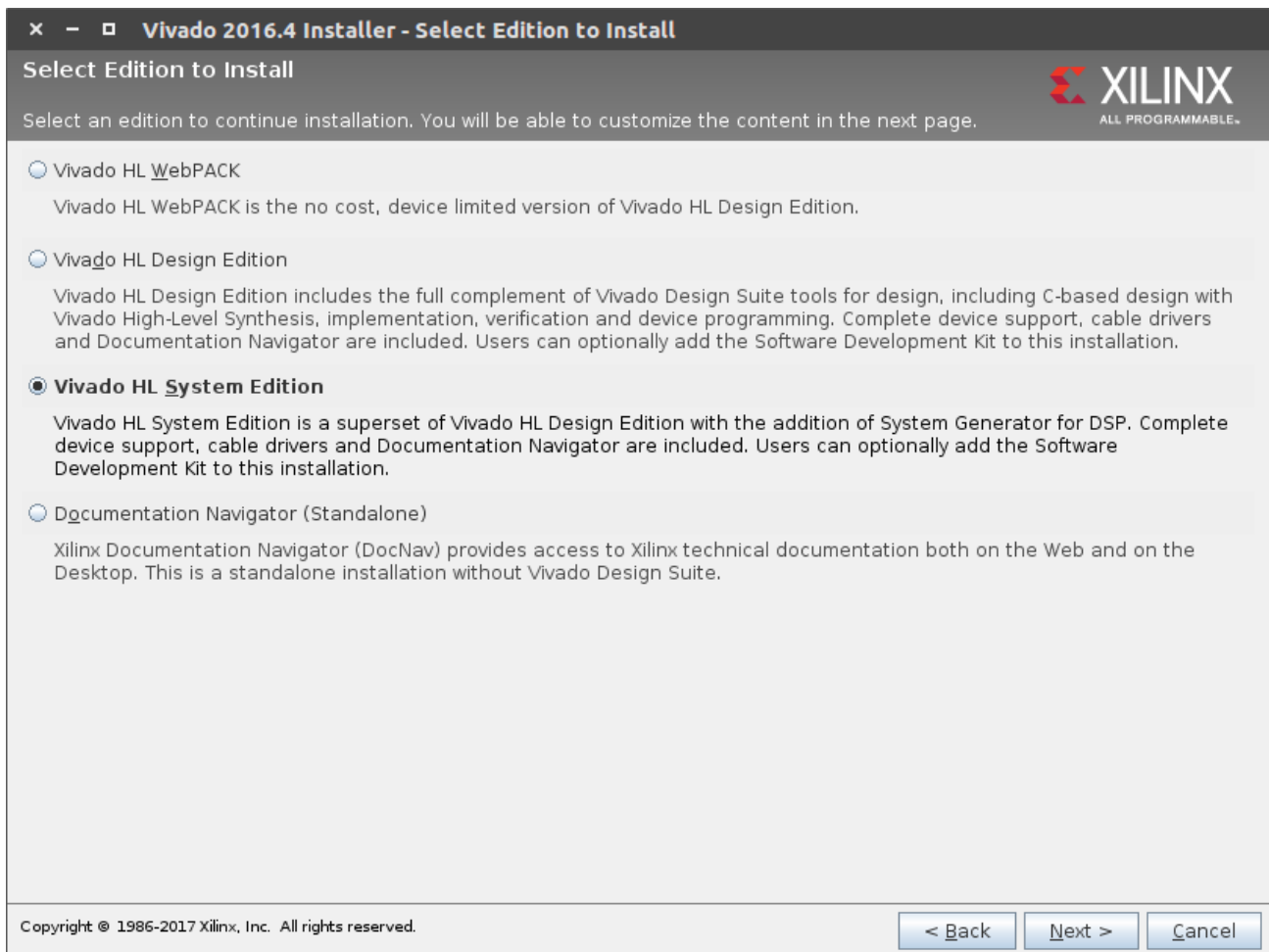
By checking "I AGREE" below, or OTHERWISE ACCESSING, DOWNLOADING, INSTALLING or USING THE SOFTWARE, YOU AGREE on behalf of licensee to be bound by the agreement, which can be viewed by [clicking here](#).

☒ **I Agree**

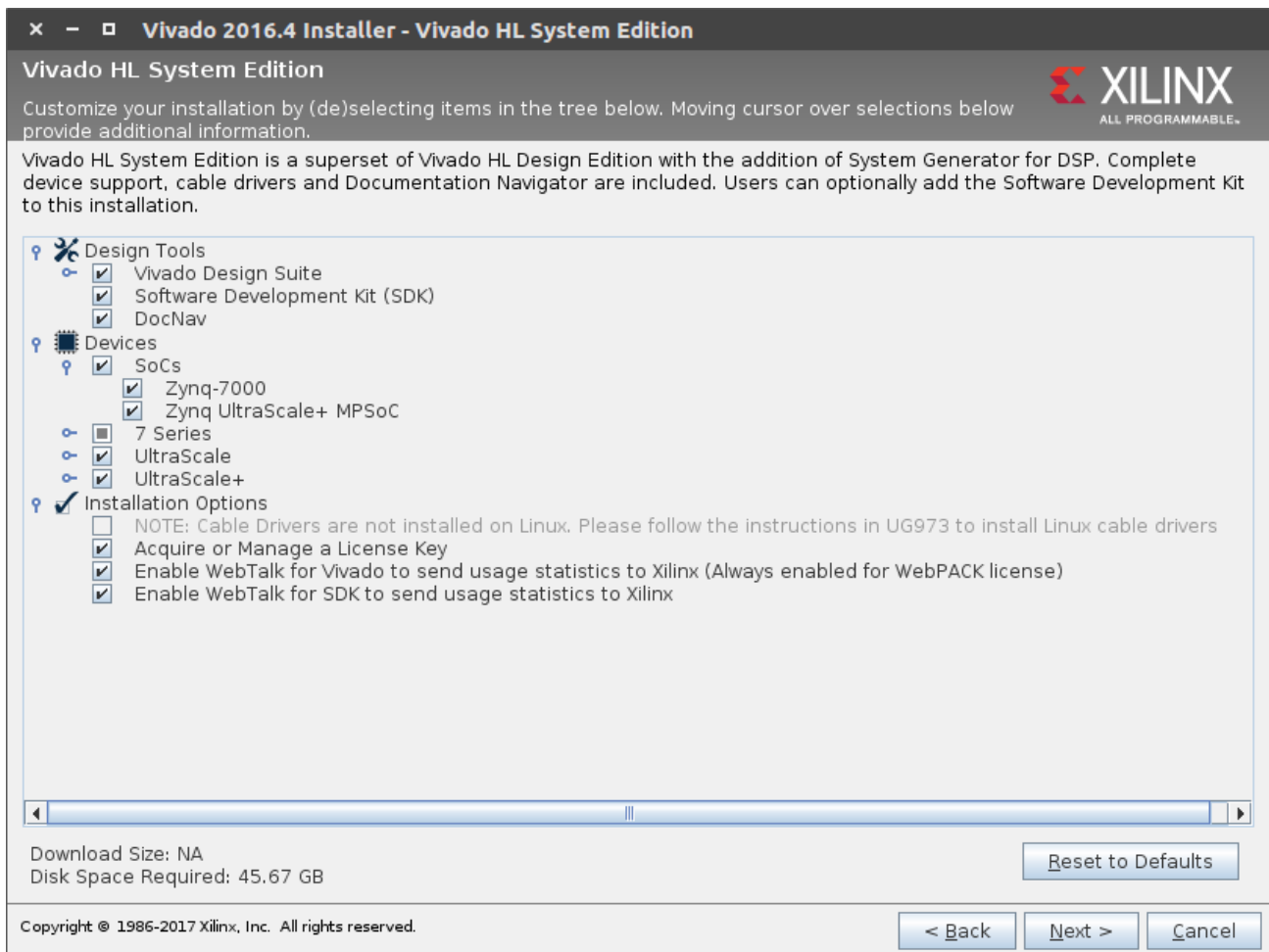
Copyright © 1986-2017 Xilinx, Inc. All rights reserved.

< Back Next > Cancel

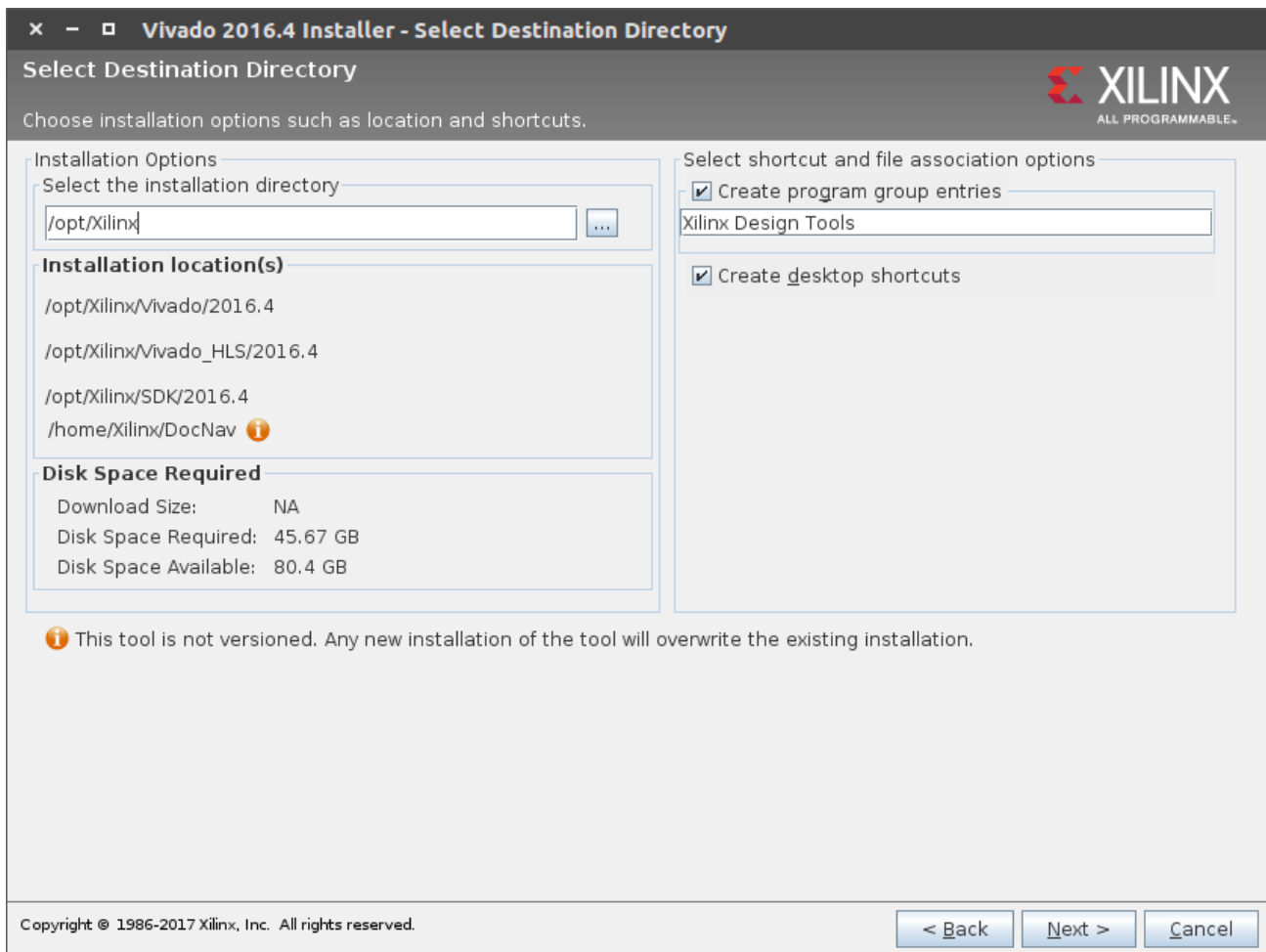
select third one, click Next



select "Software Development Kit (SDK)", click Next



choose installation directory, click Next



click install



Installation Summary

Edition: Vivado HL System Edition

Devices

- SoCs (Zynq-7000, Zynq UltraScale+ MPSoC)
- 7 Series (Artix-7, Kintex-7, Virtex-7)
- UltraScale (Kintex UltraScale, Virtex UltraScale)
- UltraScale+ (Kintex UltraScale+, Virtex UltraScale+)

Design Tools

- Vivado Design Suite (Vivado, System Generator for DSP, Vivado High Level Synthesis)
- Software Development Kit (SDK)
- DocNav

Installation Options

- Enable WebTalk for SDK to send usage statistics to Xilinx
- Enable WebTalk for Vivado to send usage statistics to Xilinx (Always enabled for WebPACK license)
- Acquire or Manage a License Key

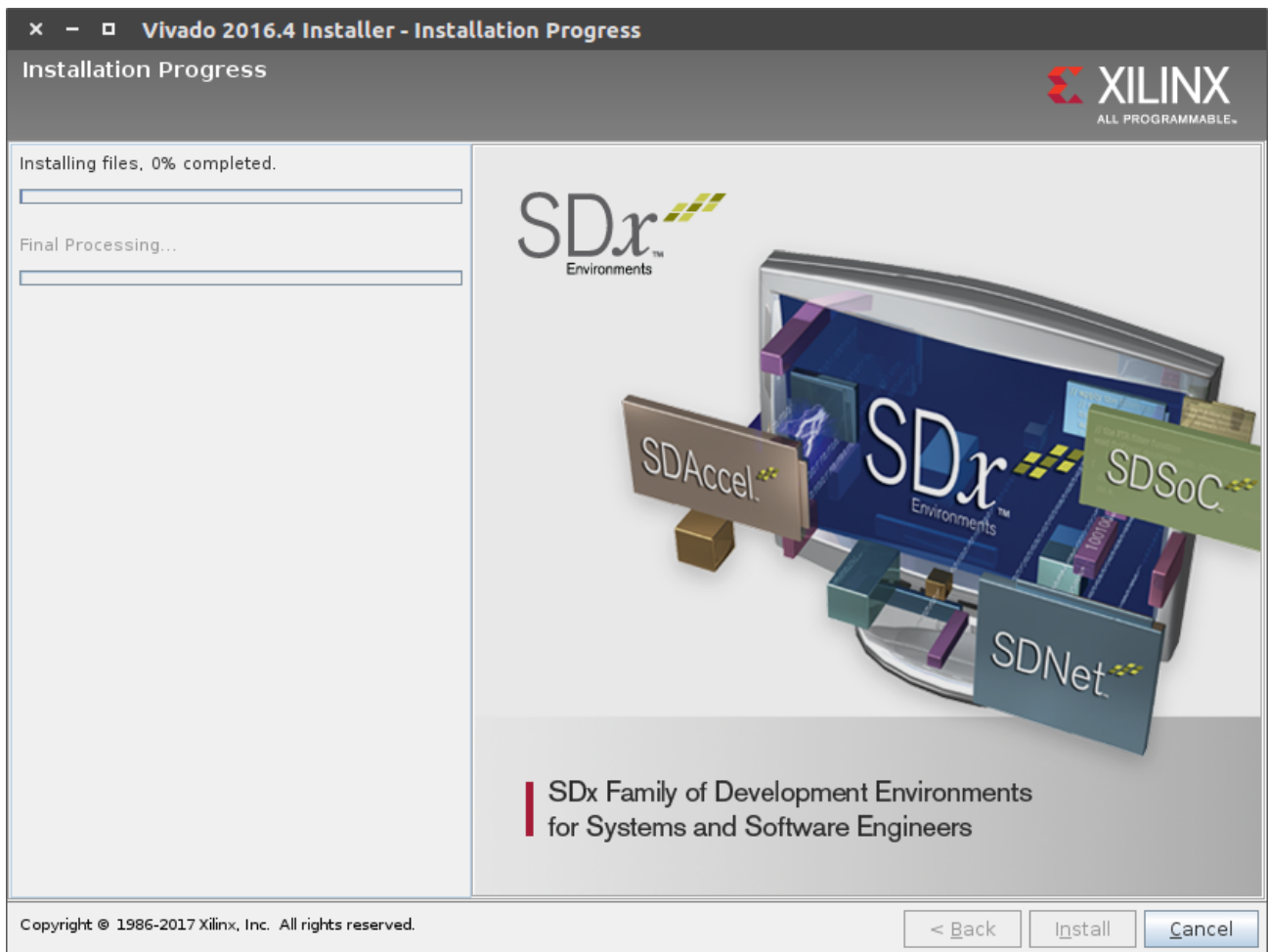
Installation location

- /opt/Xilinx/Vivado/2016.4
- /opt/Xilinx/Vivado_HLS/2016.4
- /opt/Xilinx/SDK/2016.4
- /home/Xilinx/DocNav

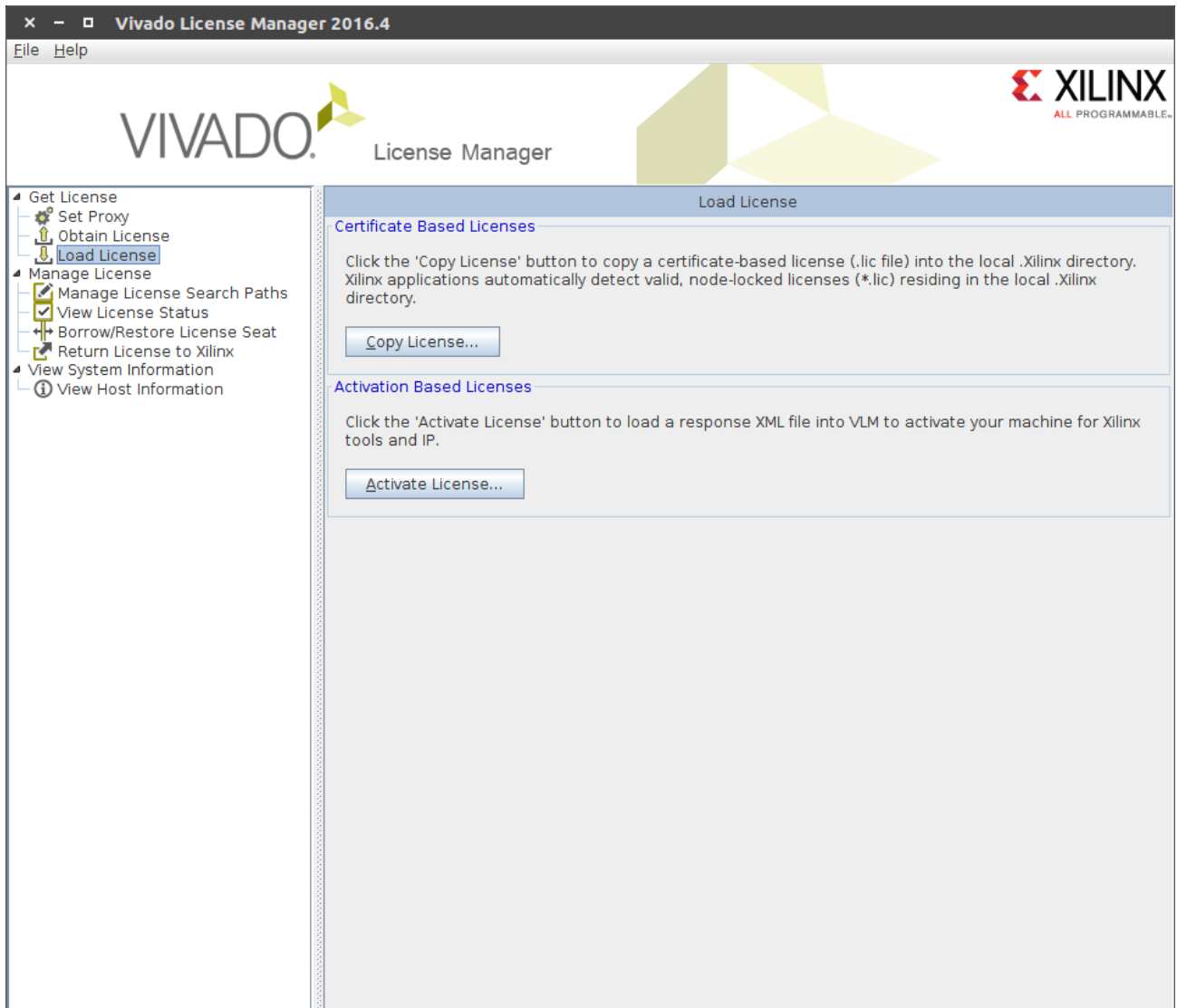
Disk Space Required

- Download Size: NA
- Disk Space Required: 45.67 GB

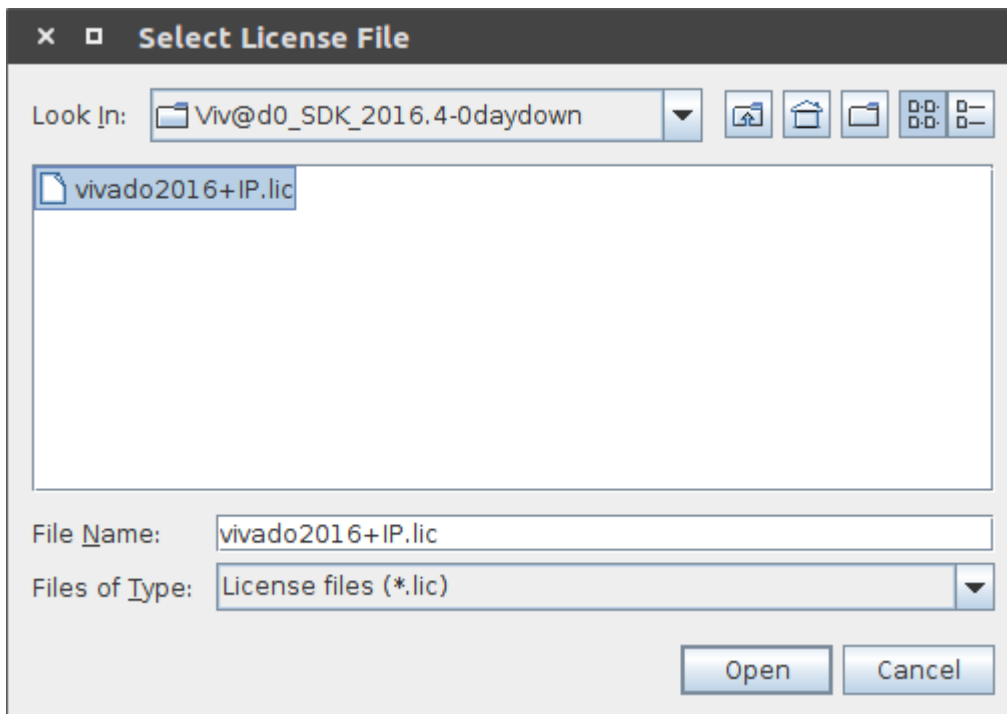




after this, we need to load the license:



click "Load License" and "Copy License" (You can find lic file in firmware path
[phenix_devkit path]/amp_system/)



select your license and click open.

the last step, you need to add some environment parameter in **.bashrc**:

```
source /opt/Xilinx/Vivado/2016.4/settings64.sh
source /opt/Xilinx/SDK/2016.4/settings64.sh
export PATH=$PATH:/opt/Xilinx/SDK/2016.4/bin
```

1.3 Install cable driver

On Windows, Install Cable Drivers is an optional selection in the installer. For Linux, because root or sudo access is required to install drivers, this option has been removed from the Linux installer beginning in Vivado 2015.4. The general Vivado installer can now be run on Linux without root or sudo privileges. To install cable drivers on Linux, there is now a script that must be run as root or sudo post installation.

Script Location: /data/xicom/cable_drivers/lin64/install_script/install_drivers/

Script Name: install_drivers

Reference: ug973-vivado-release-notes-install-license

1.4 Add Vivado SDK to Dash

After steps from 1.1 to 1.3, Vivado SDK has been installed successfully, but in x64 ubuntu, we still cannot find vivado in Dash, so we have to add it by ourselves:

For Vivado:

```
sudo touch /usr/share/applications/vivado.desktop
sudo vim /usr/share/applications/vivado.desktop
```

add these statement:

```
[Desktop Entry]
Encoding=UTF-8
Name=Vivado 2016.4
Exec=[Vivado Install Dir]/Vivado/2016.4/bin/vivado
Terminal=false
Type=Application
Icon=[Vivado Install
Dir]/Vivado/2016.4/doc/images/vivado_logo.png
StartupNotify=true
Categories=System;
```

For SDK:

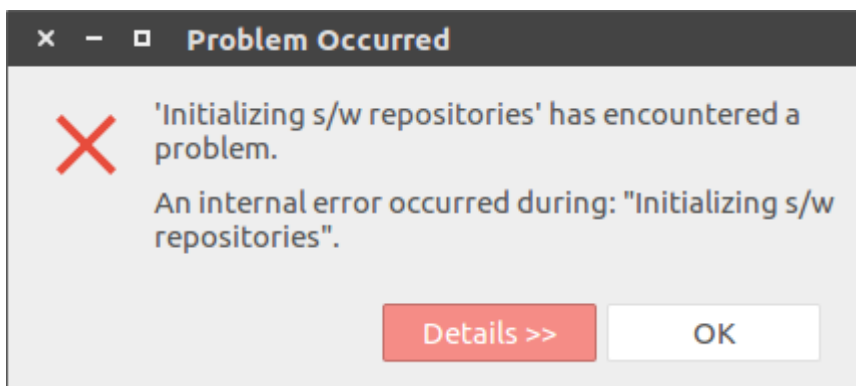
```
cd [Vivado Install Dir]/SDK/2016.4/eclipse/lnx64.o/
sudo mkdir jre
cd jre
sudo ln -s [your jdk path]/jre/bin
sudo touch /usr/share/applications/xsdk.desktop
sudo vim /usr/share/applications/xsdk.desktop
```

add these statement:

```
[Desktop Entry]
Encoding=UTF-8
Name=SDK 2016.4
Exec=xsdk
Terminal=false
Type=Application
Icon=[Vivado Install
Dir]/SDK/2016.4/data/sdk/images/sdk_logo.png
StartupNotify=true
Categories=System;
```

Note:

When launch SDK in Dash, it shows this error, it means you have to launch sdk with shell command "xsdk", otherwise, you may find that you can't close SDK:



If you launch SDK in terminal with command "xsdk", and there is nothing happen, but show information below:

```
***** Xilinx Software Development Kit
***** SDK v2016.4 (64-bit)
**** SW Build 1733598 on Wed Dec 14 22:35:42 MST 2016
** Copyright 1986-2016 Xilinx, Inc. All Rights Reserved.
```

```
Launching SDK with command
/opt/Xilinx/SDK/2016.4/eclipse/lnx64.o/eclipse -vmargs -
Xms64m -Xmx512m -
Dorg.eclipse.swt.internal.gtk.cairoGraphics=false
```

**then, you need to add these 2 lines to [Vivado Install
Dir]/SDK/2016.4/eclipse/lnx64.o/eclipse.ini before statement "-vmargs"**

```
--launcher.GTK_version
2
```

just like this:

```
1 -startup
2 plugins/org.eclipse.equinox.la
3 --launcher.library
4 plugins/org.eclipse.equinox.la
5 -showsplash
6 com.xilinx.sdk.product
7 --launcher.GTK_version
8 2
9 -vmargs
10 -Xms64m
11 -Xmx512m
```

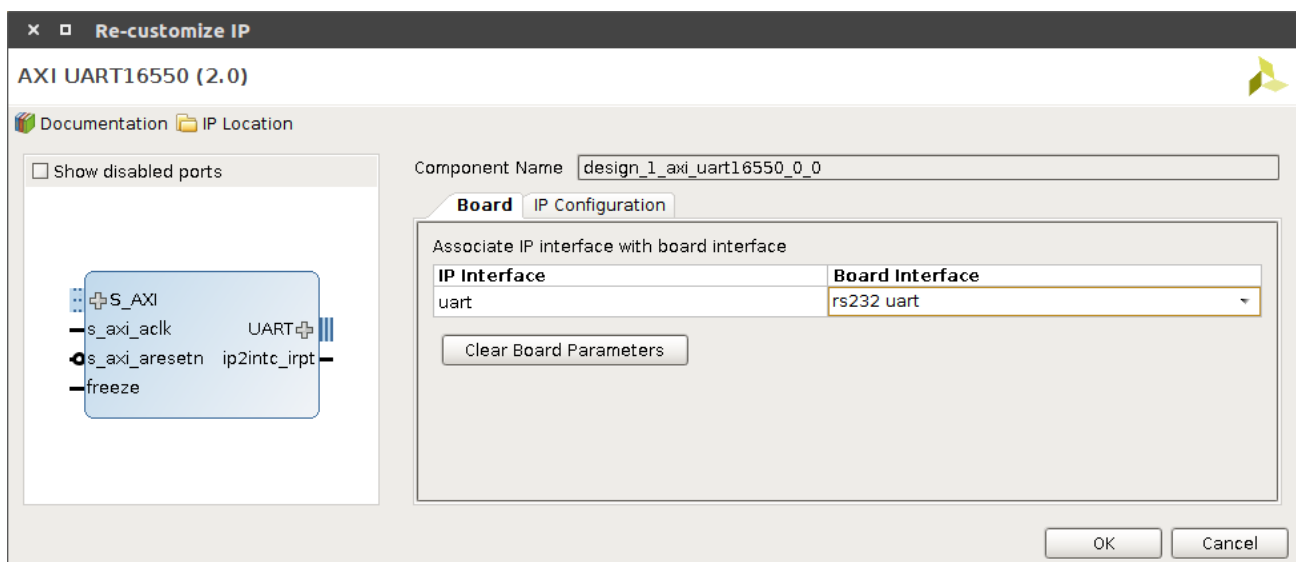
2. Build Minimum System Hardware

2.1 Install Board Interface File

(Reference: ug895-vivado-system-level-design-entry.pdf Appx.A)

After installing Vivado, the installation directory will contain a folder called **board_files**. This folder contains XML files for different FPGA boards manufactured by Xilinx.

XML files define different interfaces on the board. Interfaces such as UART, DDR Memory, Ethernet etc. For example, we use IP core AXI UART16550, but it has 14pins by default, there are only 2 pins in our devkit for uart, so we have to custom the interface of the IP core through XML. After installing board interface file, we will be able to assign different inferences that are available on your selected board to a specific IP block:



Robsense has written these XML files, they are stored in our firmware project, all you need is just get it from github, and copy then into path **board_files**:

```
cd [phenix_devkit path]/amp_system/  
tar xf vivado_prj.tar.gz  
sudo cp vivado_prj/phenix_devkit [Vivado Install  
Dir]/Vivado/2016.4/data/boards/board_files -r
```

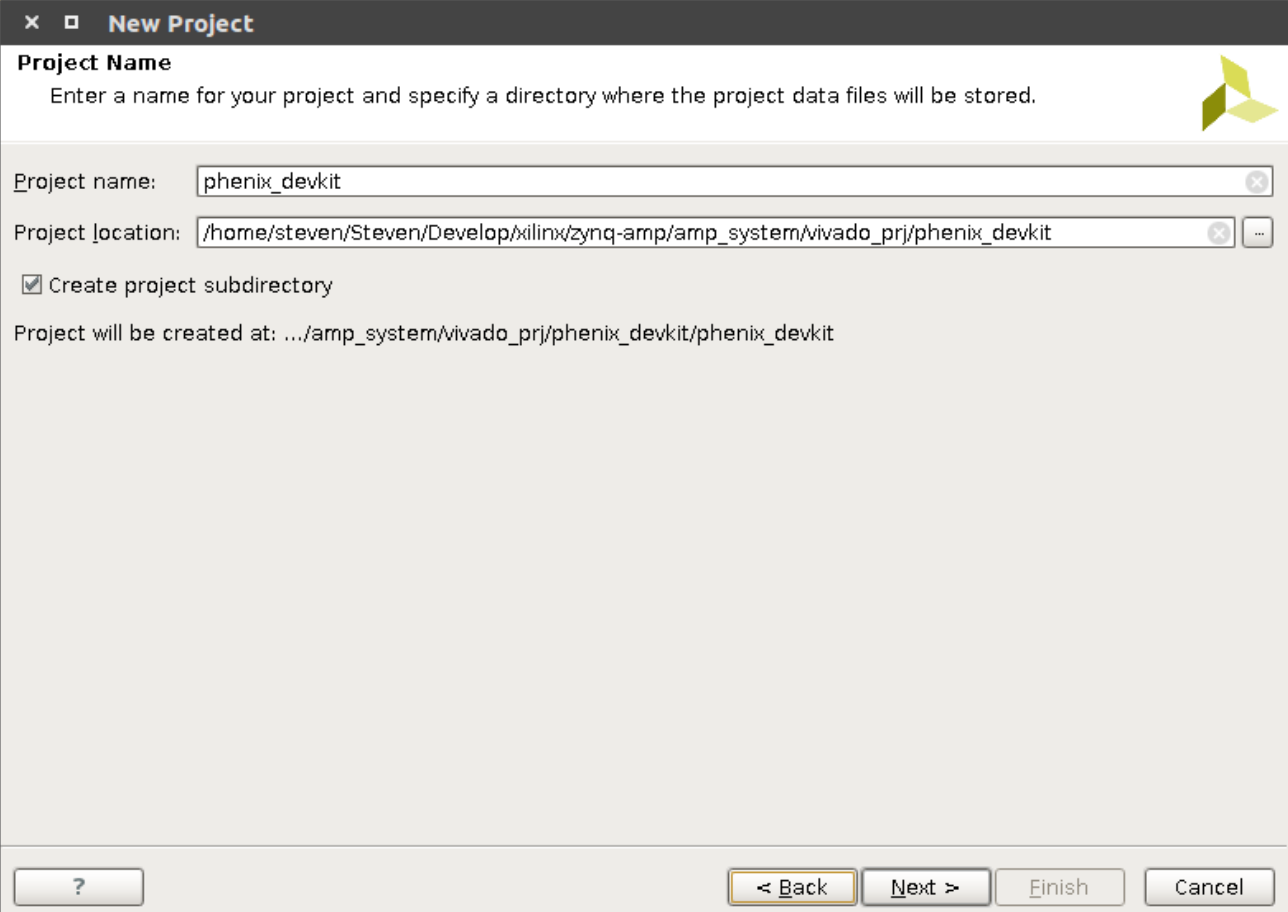
Note: If you are interested in how to write the XML, you can reference Chapter Appx.A in Xilinx official document **UG895**

(https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_4/ug895-vivado-system-level-design-entry.pdf).

2.2 Create Project

(Reference:<http://zedboard.org/zh-hant/node/1454>)

Open Vivadio and click "Create New Project", input project name, and click Next



The image shows the 'New Project' dialog box in Vivado. The title bar is 'New Project'. Below the title bar, the text 'Project Name' is followed by the instruction 'Enter a name for your project and specify a directory where the project data files will be stored.' To the right of this text is the Xilinx logo. Below the instruction, there are two text input fields. The first is labeled 'Project name:' and contains the text 'phenix_devkit'. The second is labeled 'Project location:' and contains the path '/home/steven/Steven/Develop/xilinx/zynq-amp/amp_system/vivado_prj/phenix_devkit'. Below these fields is a checkbox labeled 'Create project subdirectory' which is checked. Below the checkbox, the text 'Project will be created at: .../amp_system/vivado_prj/phenix_devkit/phenix_devkit' is displayed. At the bottom of the dialog box, there are four buttons: a help button with a question mark, a '< Back' button, a 'Next >' button, and a 'Cancel' button.

New Project

Project Name
Enter a name for your project and specify a directory where the project data files will be stored.

Project name: phenix_devkit

Project location: /home/steven/Steven/Develop/xilinx/zynq-amp/amp_system/vivado_prj/phenix_devkit

☒ Create project subdirectory


Project will be created at: .../amp_system/vivado_prj/phenix_devkit/phenix_devkit

? < Back Next > Finish Cancel

select "RTL Project"

New Project

Project Type
Specify the type of project to create.



- ☒ **RTL Project**
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
☐ Do not specify sources at this time
- ☐ **Post-synthesis Project:** You will be able to add sources, view device resources, run design analysis, planning and implementation.
☐ Do not specify sources at this time
- ☐ **I/O Planning Project**
Do not specify design sources. You will be able to view part/package resources.
- ☐ **Imported Project**
Create a Vivado project from a Synplify, XST or ISE Project File.
- ☐ **Example Project**
Create a new Vivado project from a predefined template.

? < Back Next > Finish Cancel

click Next until the UI below

New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Select: ☒ Parts ☐ Boards

Filter

Product category: Speed grade:

Family: Temp grade:

Package:

Search:

Part	I/O Pin Count	Block RAMs	DSPs	FlipFlops	GTPE2 Transceivers	GTXE2 Transceivers	Gb Transceiver
xc7z010iclg485-1L	485	95	100	92400	4	0	4
xc7z020iclg400-3	400	140	220	106400	0	0	0
xc7z020iclg400-1	400	140	220	106400	0	0	0
xc7z020iclg484-3	484	140	220	106400	0	0	0
xc7z020iclg484-2	484	140	220	106400	0	0	0
xc7z020iclg484-1	484	140	220	106400	0	0	0
xc7z020iclg400-1L	400	140	220	106400	0	0	0
xc7z020iclg484-1L	484	140	220	106400	0	0	0
xc7z030fbg484-3	484	265	400	157200	0	4	4
xc7z030fbg484-2	484	265	400	157200	0	4	4

click **Boards** find Phenix Pro Devkit, click Next and then click finish:

Note: If you can't find Phenix Pro Devkit, please make sure that you installed board interface files right in section 2.1

✕
□
New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Select:

Parts

Boards

Filter/ Preview

Vendor:

All

Display Name:

All

Board Rev:

Latest

Reset All Filters

Search:

Q

Display Name	Vendor	Board Rev	Part	I/O Pin Cou
Zybo	digilentinc.com	b	xc7z010clg400-1	400
MicroZed 7020 Board	em.avnet.com	f	xc7z020clg400-1	400
ZedBoard Zynq Evaluation and Development Kit	em.avnet.com	d	xc7z020clg484-1	484
Phoenix Pro DevKit	robsense.com	d	xc7z020clg484-1	484
Artix-7 AC701 Evaluation Platform	xilinx.com	1.1	xc7a200tfbg676-2	676
Kintex-7 KC705 Evaluation Platform	xilinx.com	1.1	xc7k325tffg900-2	900
Kintex-UltraScale KCU105 Evaluation Platform	xilinx.com	1.0	xc7ku040-ffva1156-2-e	1,156
Virtex-7 VC707 Evaluation Platform	xilinx.com	1.1	xc7vx485tffg1761-2	1,761
Virtex-7 VC709 Evaluation Platform	xilinx.com	1.0	xc7vx690tffg1761-2	1,761
Virtex-UltraScale VCU108 Evaluation Platform	xilinx.com	1.0	xc7vu095-ffva2104-2-e	2,104
Virtex-UltraScale VCU110 Evaluation Platform	xilinx.com	1.0	xc7vu180-ffva2104-2-e	2,104

?

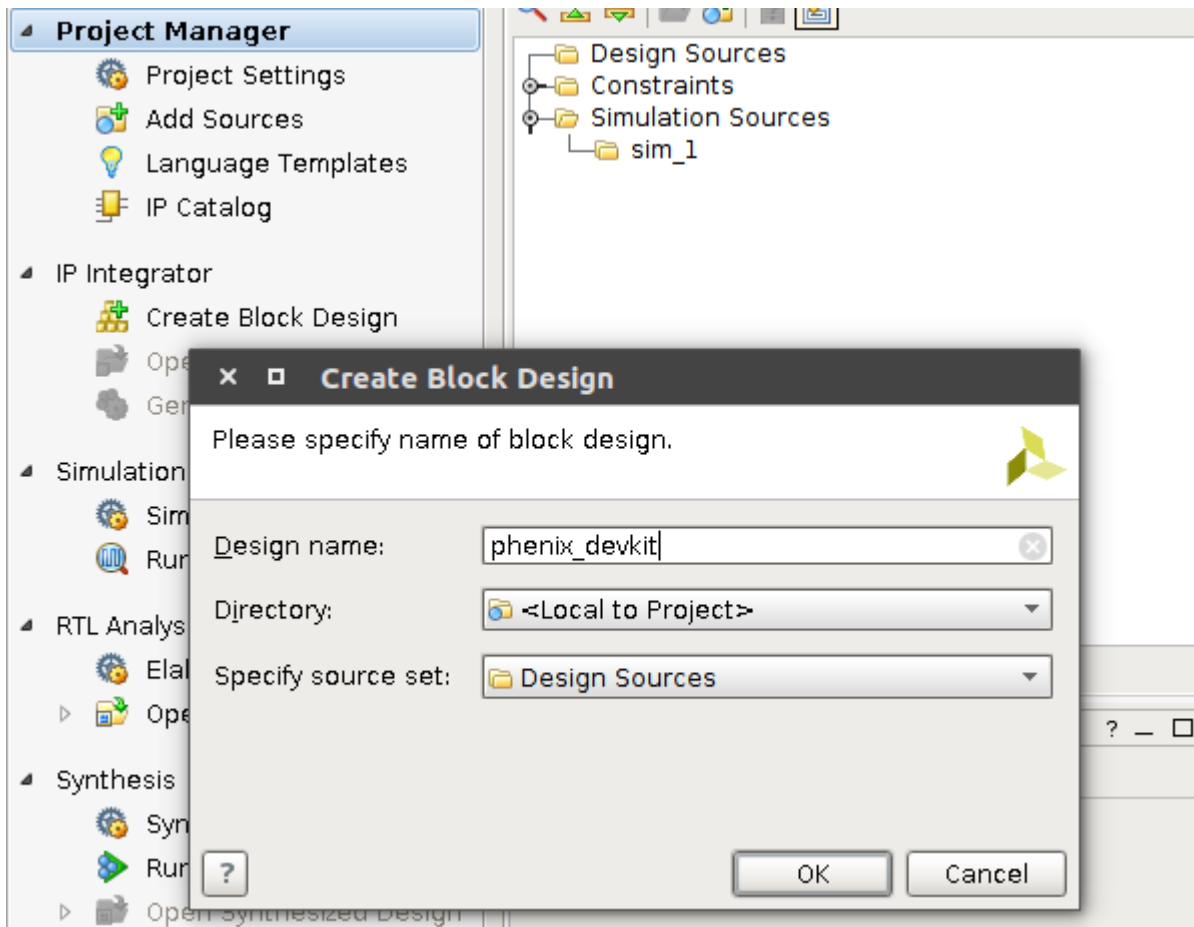
< Back

Next >

Finish

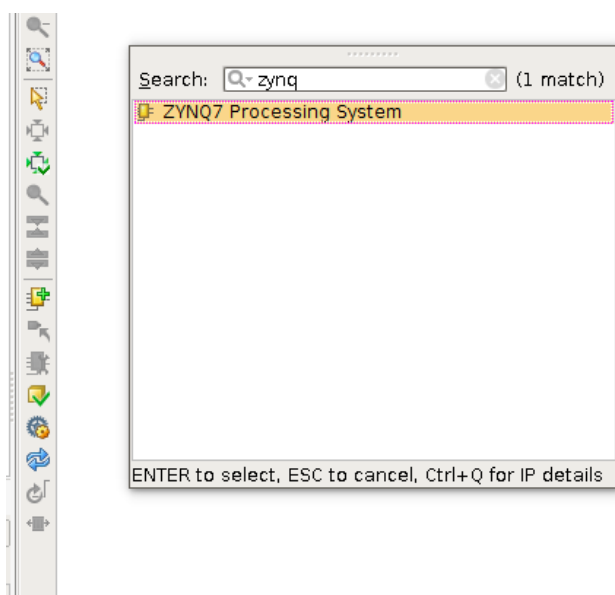
Cancel

click "Create Block Design", input design name, and click "OK"




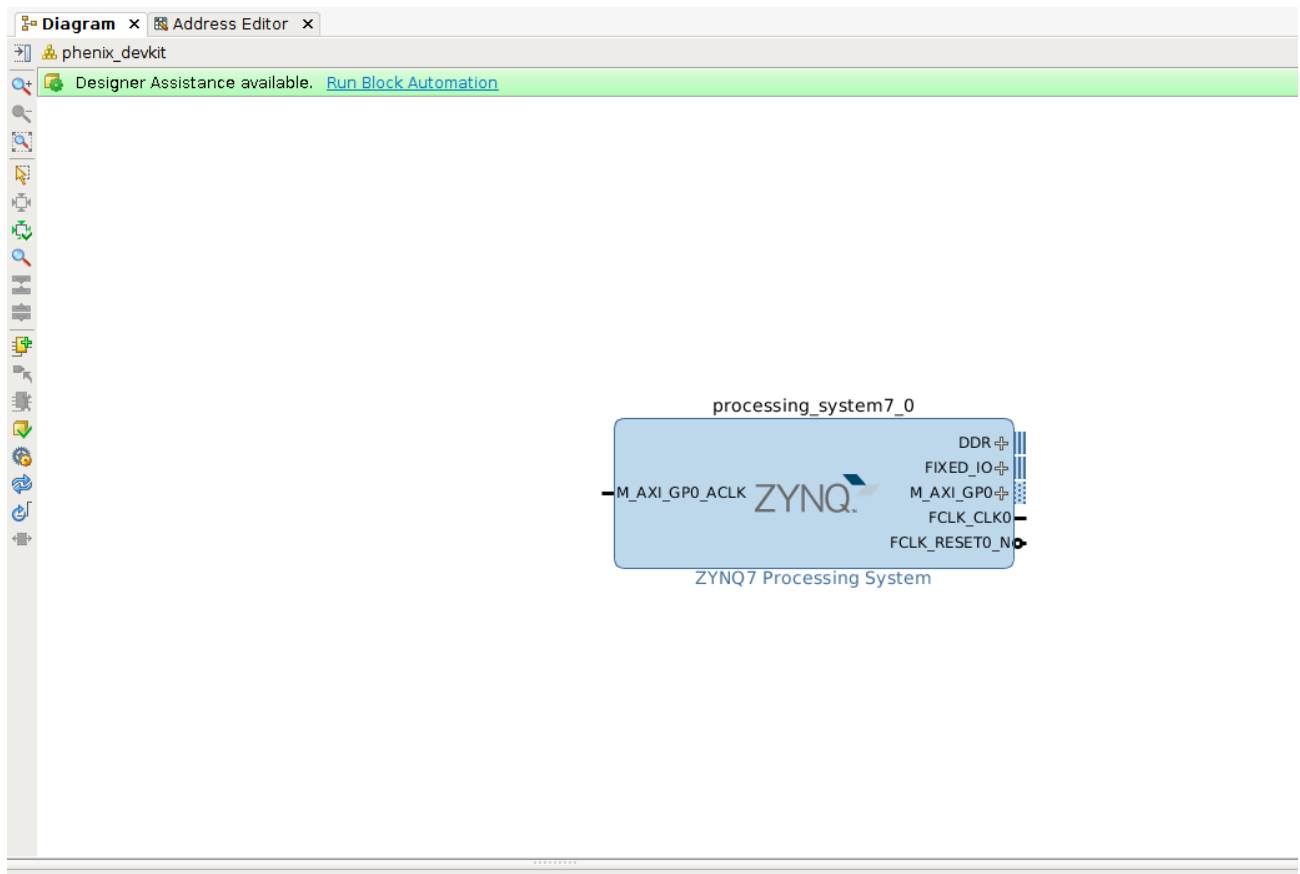
2.3 Create Minimum System Design

Now we have a empty design, let's add IP to it

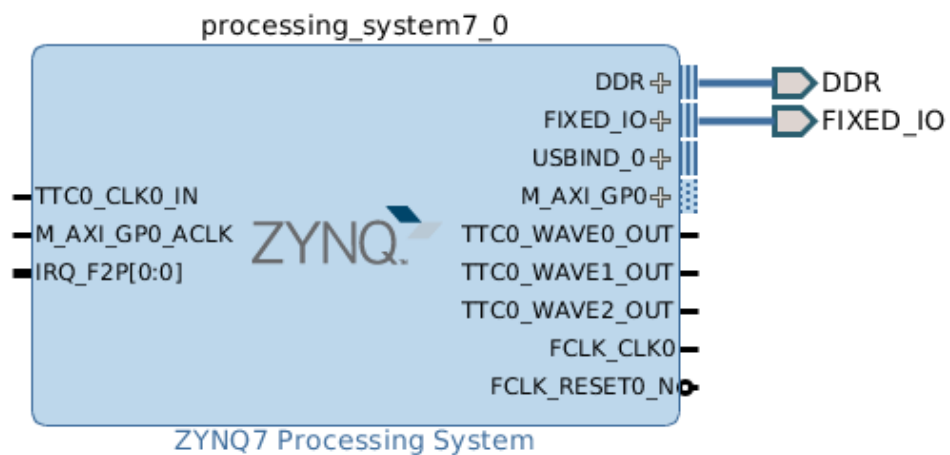


This design is empty. Press the  button to add IP.

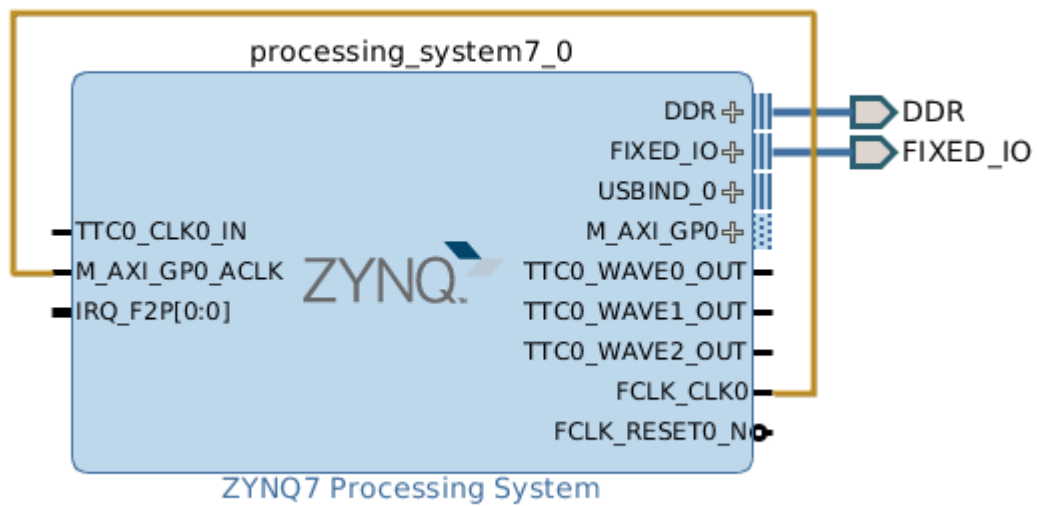
click  and input zynq to search "ZYNQ Processing System", double click to add it to our design, and click **Run Block Automation**, Vivado will configure PS core with board interface files which we installed in 2.1



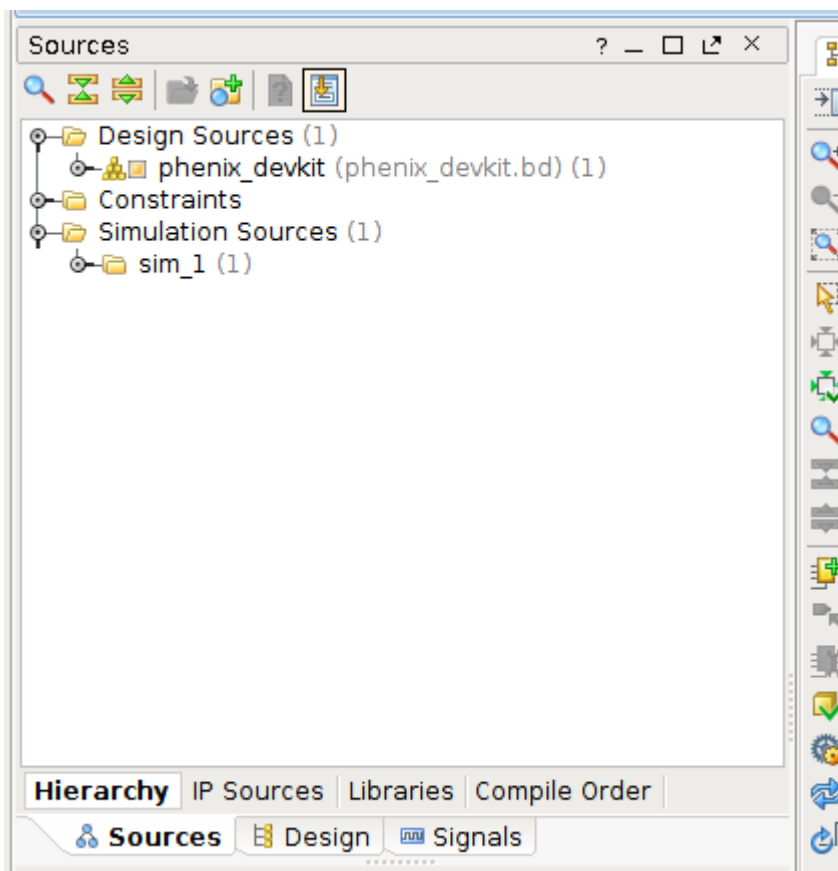
now, the PS core should be like this :




connect an input signal for AXI, like the image below:

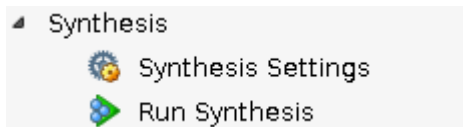


Now, we've created our PS within the IPI block, next, we need to create an HDL wrapper for the Vivado synthesizer knows what to do with our IP block

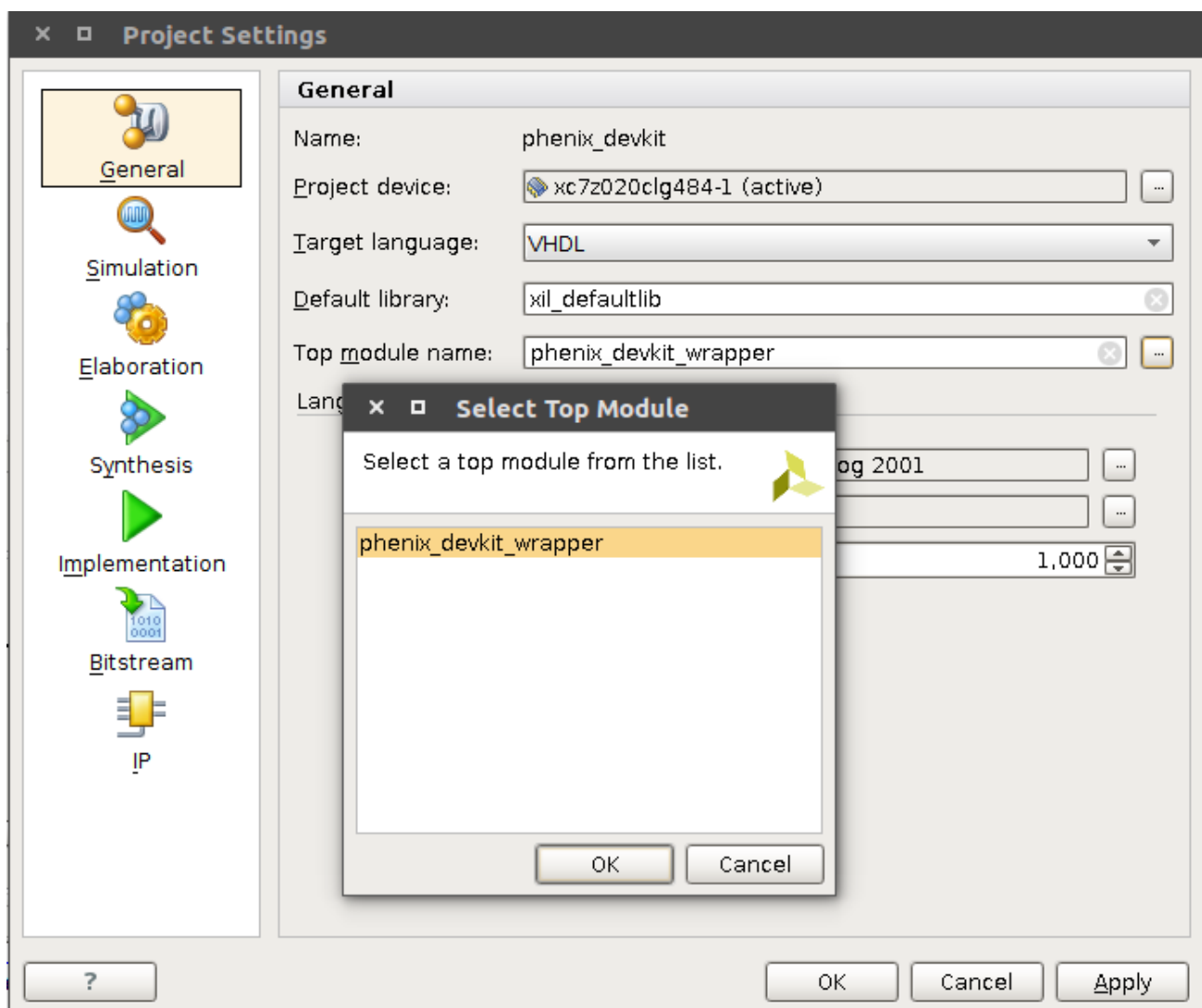


right mouse click  phenix_devkit (phenix_devkit.bd) (1) and select "Create HDL Wrapper", This will generate a HDL wrapper that the Vivado synthesizer understands. Once this happens, we are ready to generate our bitfile. This might sound like a large jump, but there isn't anything else in our design - it's almost entirely PS (the only PL portion is that AXI port support logic).

next, we need to set "Top module name", click "Synthesis Settings"



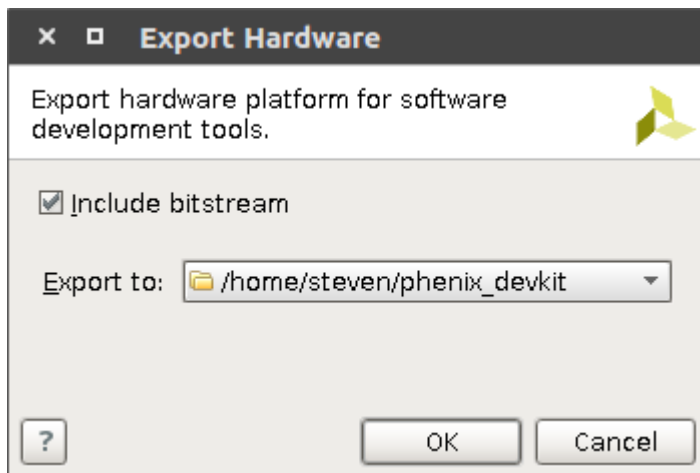
and "..." button, select phenix_devkit_wrapper:



At last, click "Run Synthesis", "Run Implementation" and "Generate Bitstream" to generate PL program

2.4 Export hdf(hardware definition file) file

If we want to develop on PS with Xilinx SDK, we need to tell SDK how our hardware looks like, which is the usage of the hdf file. Exporting hdf file is quite simple, just click **File->Export->Export Hardware**, and choose the path you want to store the hdf file(don't forget include bitstream file):



now, we can launch SDK to generate FSBL and BSP(chapter 3.3,3.4)

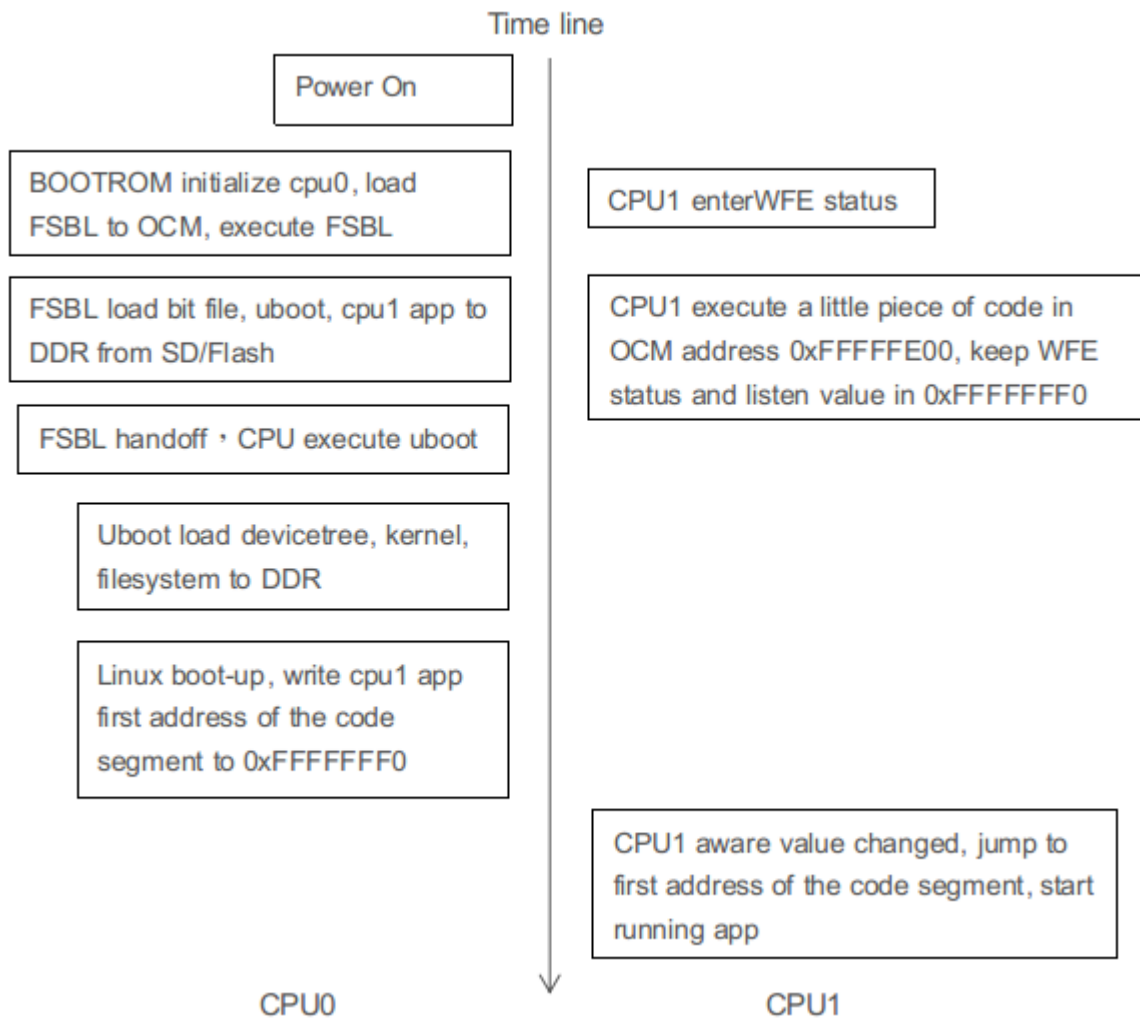
3.Run AMP(Asymmetric Multi-Processing System) on Devkit

This chapter will show you a simple demo about running an AMP system on Devkit, with cpu0 running linux and cpu1 running bare-metal program.(Reference: Xilinx Official Doc **xapp1078-amp-linux-bare-metal.pdf**).

The most important thing is that two cpu should run in different DDR physical address space. Devkit has 512MB DDR, whose physical address range is 0x0 - 0x20000000. In this case, we set 0x0 - 0x19FFFFFF(416MB) as linux space, and 0x1A000000 - 0x1FFFFFFF(96MB) as bare-metal space

3.1 Principle

Simply put, the principle of Amp system in zynq is cpu0 wake up cpu1, now, let's see how it works:

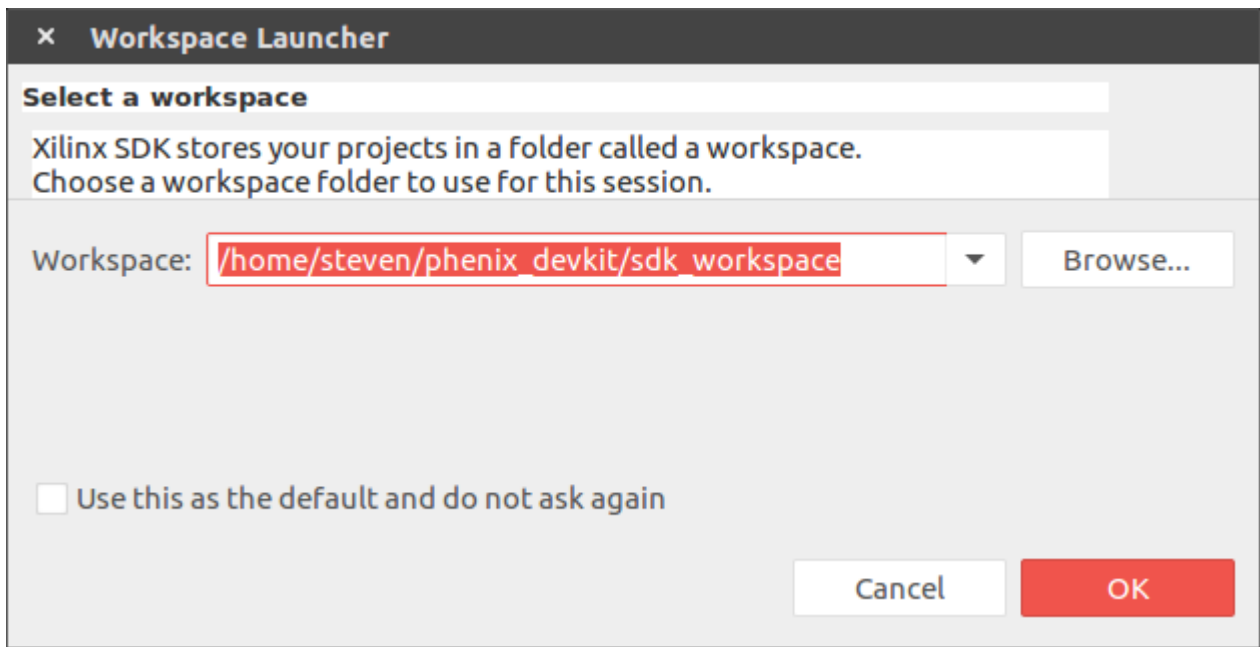


Cpu0 start first, at the same time, cpu1 running a little piece of code which have been loaded from bootrom to ocm(on chip memory), it watching the value of 0xffffffff0 adress. after linux startup, it will write the code segment first address of cpu1 app into 0xffffffff0. Once cpu1 realize the value is changed to nonzero, then the pc pointer of cpu1 will jump to the adress which is written at 0xffffffff0, this is how the whole system startup.

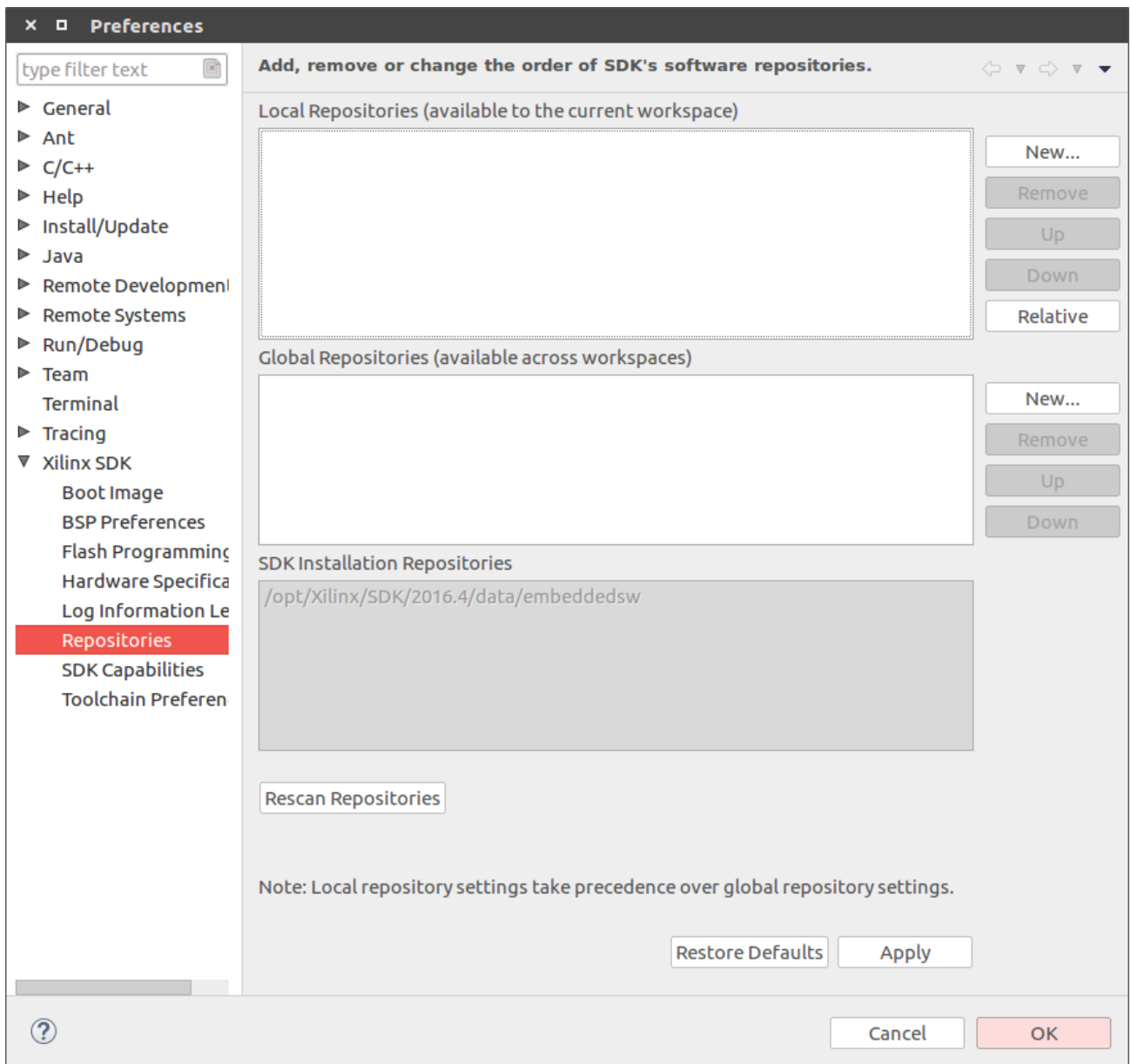
3.2 Configure SDK

First of all, download xapp1078.zip from <https://www.xilinx.com/search/site-keyword-search.html?searchKeywords=xapp1078> and unzip it. This zip file include standalone BSP files (used by the bare-metal application) and modified FSBL files. To give SDK knowledge of these files, SDK needs to be configured to have knowledge of the new repository.

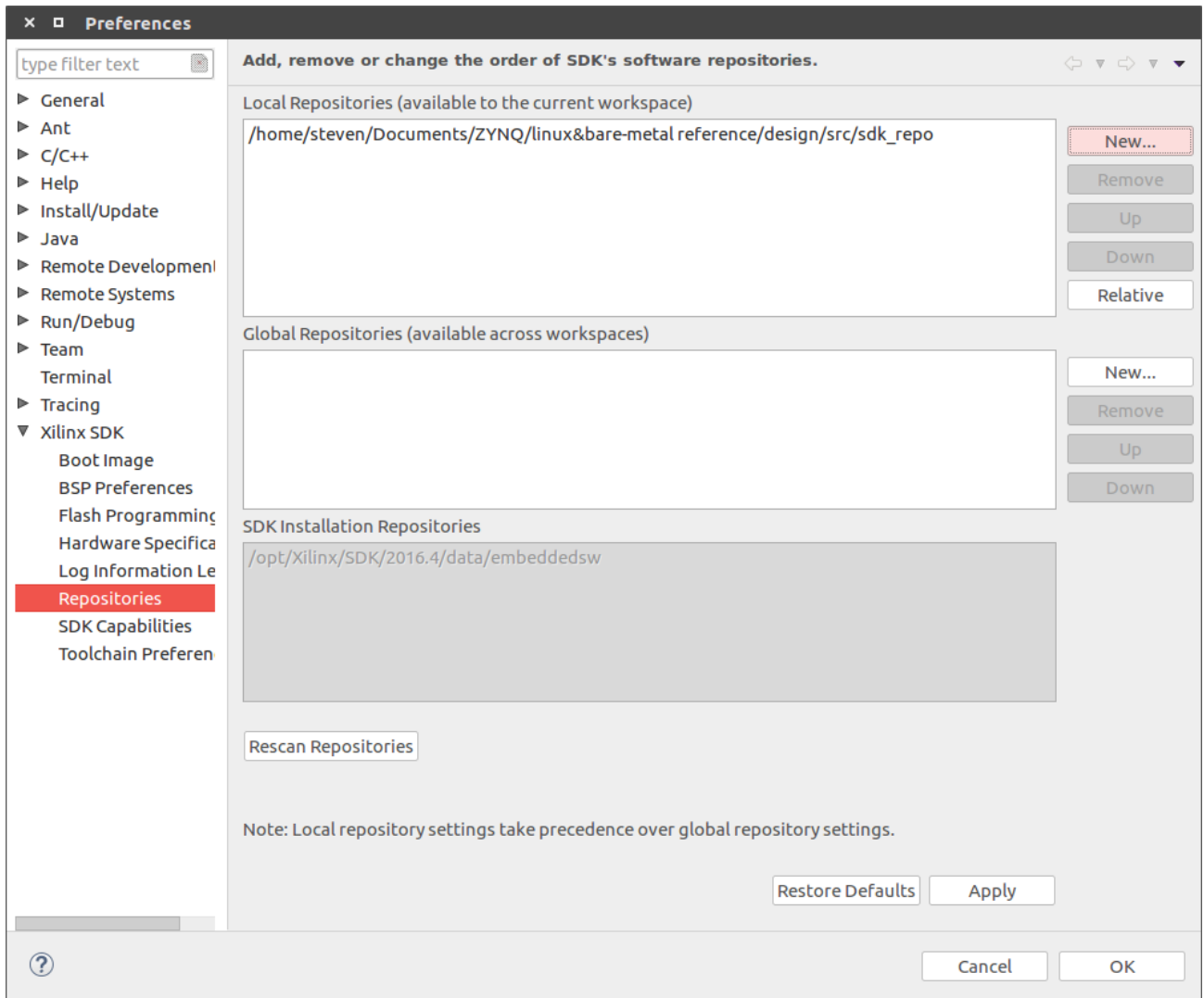
Now, open Xilinx SDK, indicate a directory as workspace and click OK:



Select **Xilinx_tools** > **Repositories**



select **New** and Browse to and select the directory design\src\sdk_repo, select **OK**



3.3 Create FSBL Application

Select **File > New > Application_Project**

New Project

Application Project
Create a managed make application project.

Project name:

☒ Use default location

Location:

Choose file system:

OS Platform:

Target Hardware

Hardware Platform:

Processor:

Target Software

Language: ☒ C ☐ C++

Compiler:

Board Support Package: ☒ Create New ☐ Use existing

select **New...** button to indicate the hdf file, click **Finish** and **Next**

New Hardware Project

Create a new Hardware Project.

Project name:

☒ Use default location

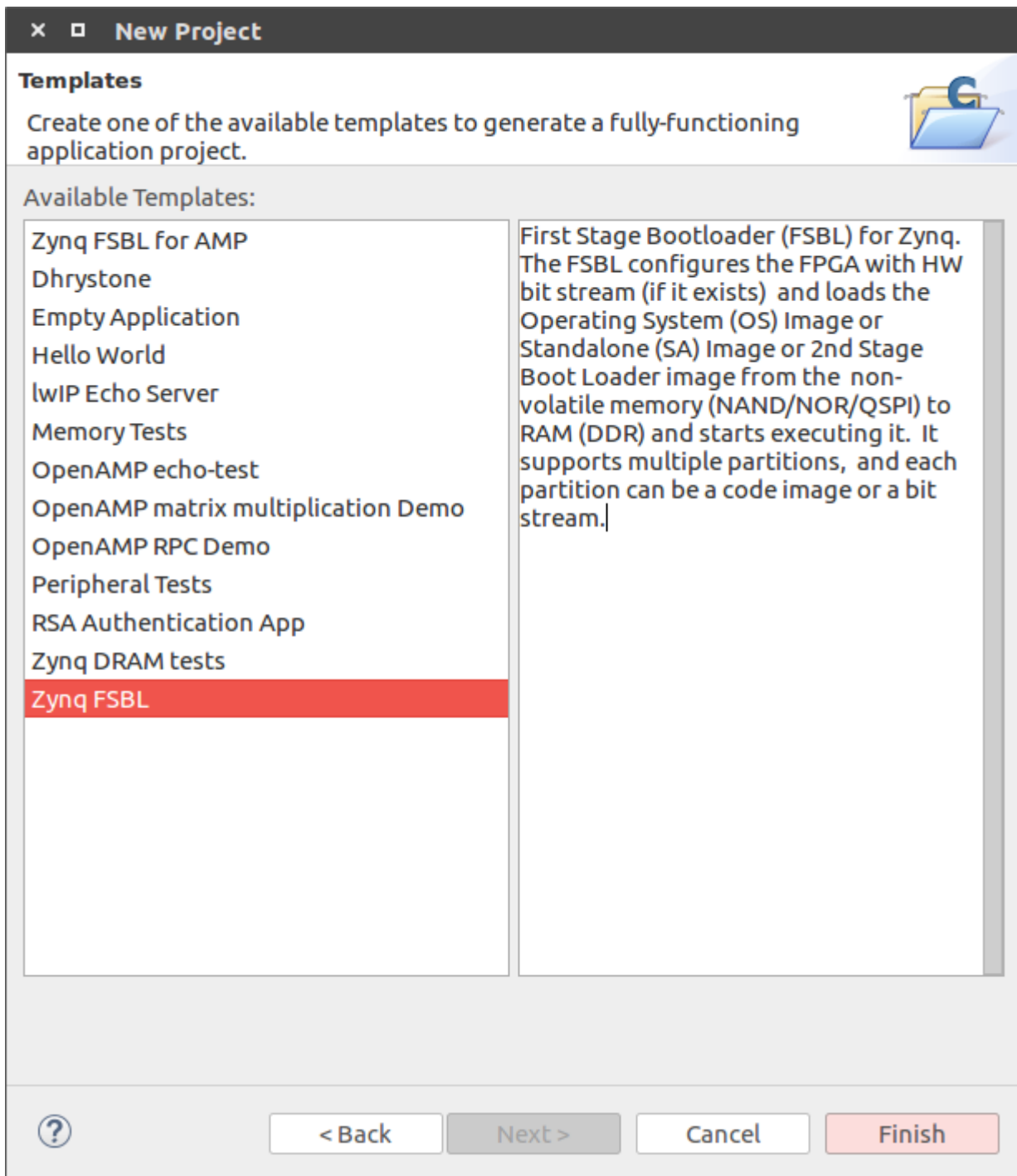
Location:

Choose file system:

Target Hardware Specification

Provide the path to the hardware specification file exported from Vivado.
This file usually resides in SDK/SDK_Export/hw folder relative to the Vivado project location.
The specification file and associated bitstream content will be copied into the workspace.

select **Zynq FSBL** and click **Finish**



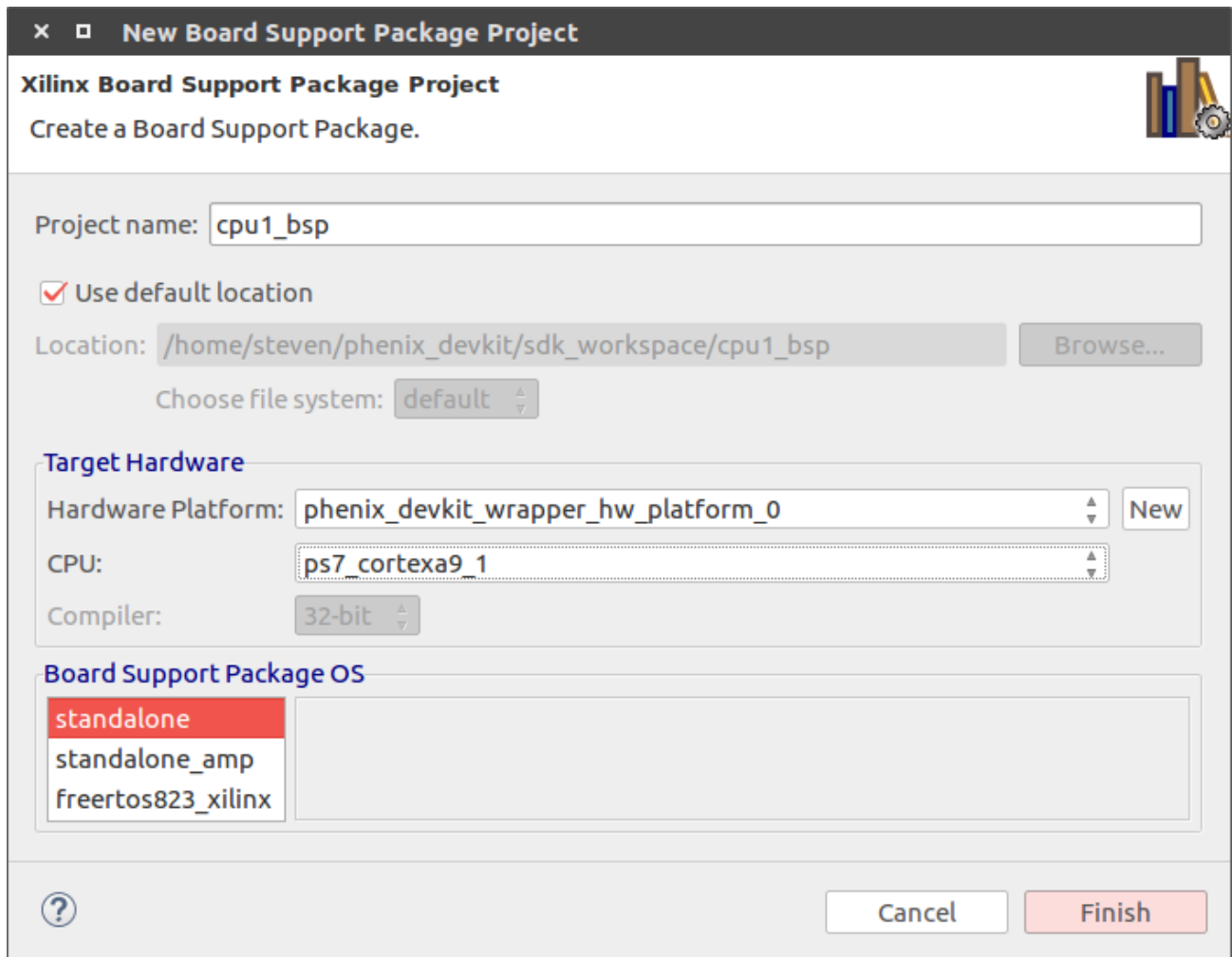
Note: xapp1078-amp-linux-bare-metal.pdf write that select Zynq FSBL for AMP, because FSBL generated by old version SDK don't support amp system

3.4 Create Bare-Metal Application For CPU1

The instructions in this section create the application ELF that runs on CPU1 after the FSBL

loads the applications to DDR memory.

First we need to create the BSP for cpu1, select **File > New > Board_Support_Package**. Change CPU to **ps7_cortexa9_1** and click **Finish**



New Board Support Package Project

Xilinx Board Support Package Project
Create a Board Support Package.

Project name:

☒ Use default location

Location:

Choose file system:

Target Hardware

Hardware Platform:

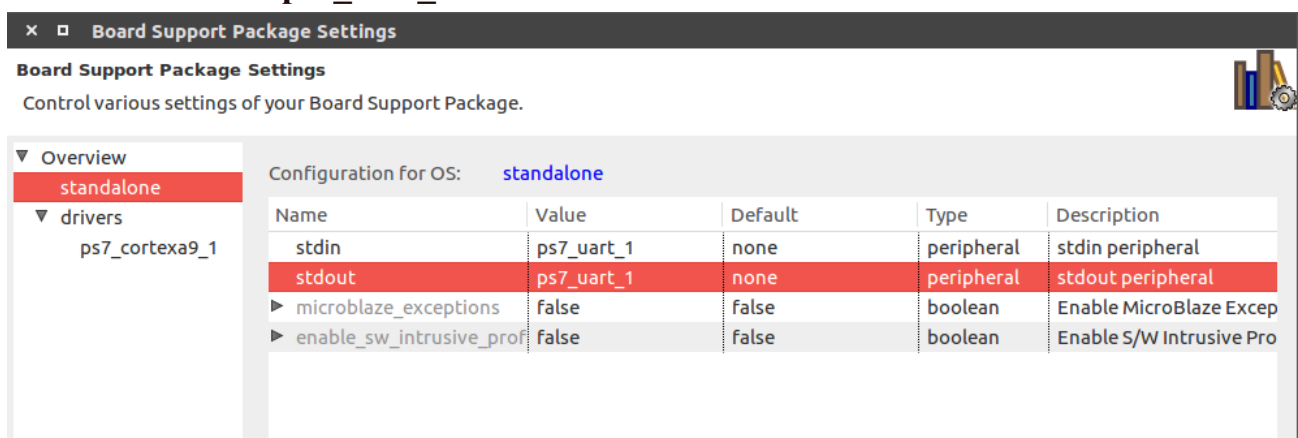
CPU:

Compiler:

Board Support Package OS

- standalone**
- standalone_amp
- freertos823_xilinx

in the Board Support Package settings, select **overview > standalone** and change both stdin and stdout to **ps7_uart_1**



Board Support Package Settings

Control various settings of your Board Support Package.

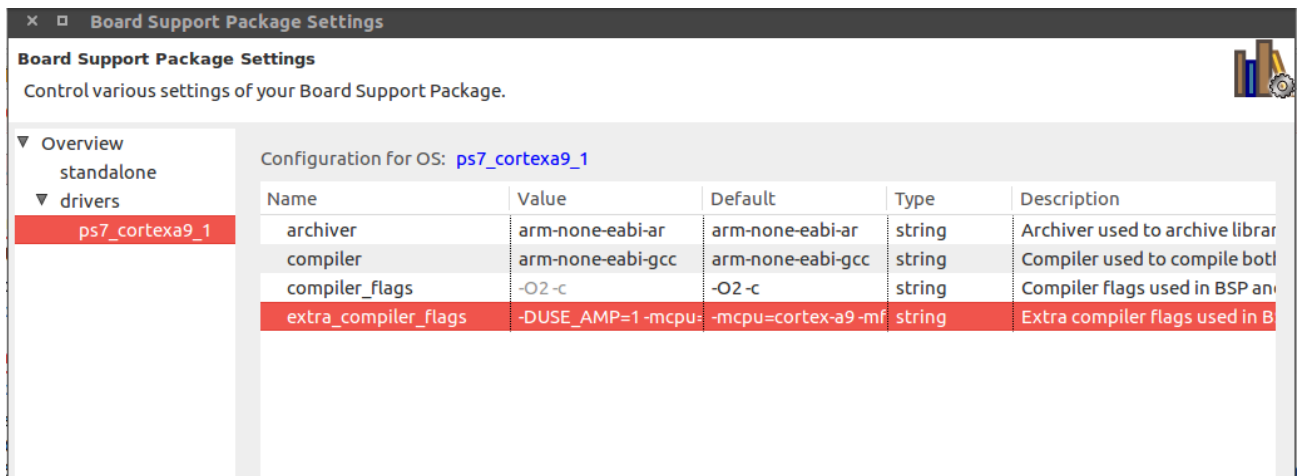
▼ Overview

- standalone**
- ▼ drivers
 - ps7_cortexa9_1

Configuration for OS: **standalone**

Name	Value	Default	Type	Description
stdin	ps7_uart_1	none	peripheral	stdin peripheral
stdout	ps7_uart_1	none	peripheral	stdout peripheral
▶ microblaze_exceptions	false	false	boolean	Enable MicroBlaze Excep
▶ enable_sw_intrusive_prof	false	false	boolean	Enable S/W Intrusive Pro

select **Overview > drivers > cpu_cortexa9_1** and add **-DUSE_AMP=1** to extra_compiler_flags



Now, we can create bare_metal application, select **File > New > Application Project**, set like image below, and click **Finish**:

New Project

Application Project
Create a managed make application project.

Project name:

☒ Use default location

Location:

Choose file system:

OS Platform:

Target Hardware

Hardware Platform:

Processor:

Target Software

Language: ☒ C ☐ C++

Compiler:

Board Support Package: ☐ Create New

☒ Use existing

change the code to a while cycle:

```

#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"

int main()
{
    init_platform();

    while(1)
    {
        print("Hello World\n\r");
        usleep(500000);
    }

    cleanup_platform();
    return 0;
}

```

here comes the most important part: modify **ld.scrip** to tell gcc the first address of the code segment 0x1A000000, which we have discussed at the beginning of this chapter. So let's open **ld.script**, and change **ps7_dds_0_S_AXI_BASEADDR** as 0x1A000000, **Size** as 0x1FF00000, **Stack Size** as 0x800000(8MB), **Heap Size** as 0x1000000(16MB), now, we can compile our code.

3.5 u-boot For CPU0

Get u-boot source code for github:

```
git clone git://github.com/Xilinx/u-boot-xlnx.git
```

As the same with bare-metal application on cpu1, u-boot also need to configure DDR address range(we use zedboard as our default configuration):

- modify CONFIG_SYS_SDRAM_SIZE in zynq_zed.h to (416 * 1024 * 1024)
- modify memory section in devicetree (u-boot-xlnx/arch/arm/dts/zynq-zed.dts):

```

23     memory {
24         device_type = "memory";
25         reg = <0x0 0x1A000000>;
26     };
27

```

make the configuration effective:

```
make zynq_zed_config
```

compile u-boot:

```
make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- -j8
```

last step, rename u-boot to u-boot.elf

Note: compile error handle:

- fatal error: openssl/evp.h
install openssl:

```
sudo apt-get install libssl-dev
```

- ./bin/sh: 1: dtc: not found
install dtc:

```
sudo apt-get install device-tree-compiler
```

3.6 Linux Kernel For CPU0

Get kernel code from github:

```
git clone -b master --single-branch  
https://github.com/Xilinx/linux-xlnx.git
```

compile kernel:

```
make ARCH=arm xilinx_zynq_defconfig  
make ARCH=arm UIMAGE_LOADADDR=0x8000 uImage -j8
```

3.7 Linux Devicetree

We use zedboard devicetree as our default(linux-xlnx/arch/arm/boot/dts/zynq-zed.dts), like steps in u-boot, modify memory section:

```
26
27     memory {
28         device_type = "memory";
29         reg = <0x0 0x1A000000>;
30     };
31
```

generate dtb file with command

```
./scripts/dtc/dtc -I dts -O dtb -o devicetree.dtb
arch/arm/boot/dts/zynq-zed.dts
```

3.8 Linux Filesystem

Note: We use **genext2fs** to generate ramdisk, make sure your computer has installed this software(download here:<https://sourceforge.net/projects/genext2fs/files/>).

Fisrt, download arm_ramdisk.image.gz from

<http://www.wiki.xilinx.com/Build+and+Modify+a+Rootfs>, with these command below, you will have root filesystem in "tmpmnt" directory:

```
gunzip arm_ramdisk.image.gz
chmod u+rw arm_ramdisk.image
mkdir tmpmnt
sudo mount -o loop arm_ramdisk.image tmpmnt/
```

As we discussed in chapter 3.1, we need **rwmem.elf** to write address 0xffffffff0 in ram, so, let's put it into directory "tmpmnt/usr", and regenerate **.image.gz** file:

```

cp xapp1078/design/generated_files/SDK_apps/rwmem.elf
tmpmnt/usr
sudo genext2fs -b 15360 -N 1000 -d tmpmnt ramdisk.image
gzip -9 ramdisk.image
mkimage -A arm -T ramdisk -C gzip -n Ramdisk -d
ramdisk.image.gz uramdisk.image.gz

```

3.9 Create BOOT.bin

This is the last step, let's go back to SDK, select **Xilinx Tools->Create Zynq Boot Image**, add fsbl.elf(chapter 3.3), bit file(chapter 2.3), u-boot.elf(chapter 3.5), hello_world.elf(chapter 3.4), click **Create Image**:

Create Boot Image
Creates Zynq Boot Image in .bin format from given FSBL elf and partition files in specified output folder.

Architecture: Zynq

☒ Create new BIF file ☐ Import from existing BIF file

Basic **Security**

Output BIF file path: /home/steven/phenix_devkit/sdk_workspace/output.bif Browse...

UDF data: Browse...

☐ Split Output format: BIN

Output path: /home/steven/phenix_devkit/sdk_workspace/BOOT.bin Browse...

Boot image partitions

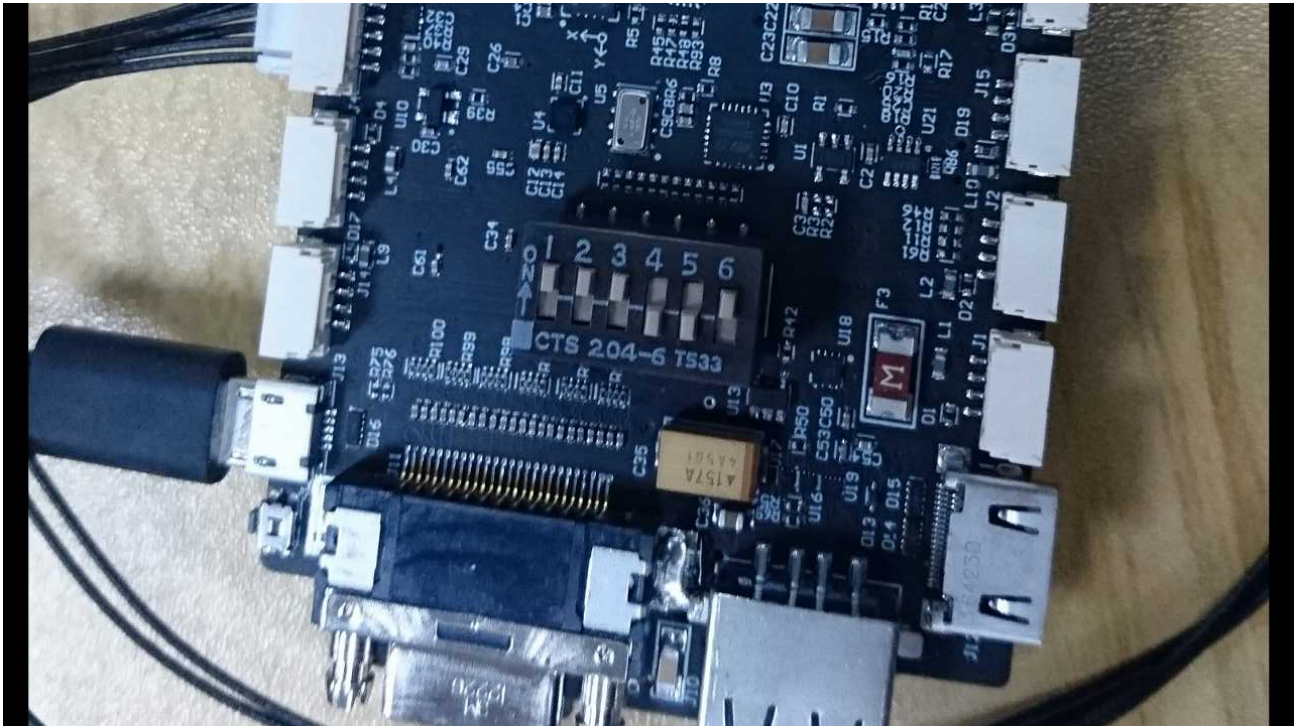
File path	Encrypted	Authenticated
(bootloader) /home/steven/phenix_devkit/sdk_workspace/fsbl/Debug/fsbl.elf	none	none
/home/steven/phenix_devkit/phenix_devkit.bit	none	none
/home/steven/phenix_devkit/sdk_workspace/u-boot.elf	none	none
/home/steven/phenix_devkit/sdk_workspace/hello_world/Debug/hello_world.elf	none	none

? Preview BIF Changes Cancel Create Image

now, you can find BOOT.bin file in your workspace directory

3.10 Test

Copy BOOT.bin, uImage, devicetree.dtb, and arm_ramdisk.image.gz to SD card, connect usb console, baudrate 115200, set devkit as SD card boot with DIP switch:



power-on devkit.

At first boot, we need to set uboot argument, which let linux only be aware of one cpu:

```
set bootargs 'console=ttyPS0,115200 maxcpus=1 root=/dev/ram
rw earlyprintk'
```

After linux boot-up, input command below to terminal, you will see "hello world" print:

```
zynq> /usr/rwmem.elf 0xffffffff 0x1a000000
```

```
Hello World
```

```
zynq> Hello World
```

```
Hello World
```

```
Hello World
```

```
Hello World
```

```
Hello World
```

```
Hello World
```

```
Hello World
```

```
Hello World
```

```
Hello World
```

```
Hello World
```

```
Hello World
```