PhenixPro Devkit V1.0

Download the codebase

Download code from github:

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

compile code under the guidance of [prj_path]/README.md

Build the Firmware

Here is a simple process for developers to compile the source code.

compile cpu1 BSP(Board Support Package):

```
make bsp-clean;make bsp -j8
```

compile pilot code:

```
make pilot-clean;make pilot -j8
```

compile all:

```
make clean;make -j8
```

Note: Bcause the pilot code depends on cpu1 bsp, please make sure that you have compiled bsp before compiling pilot code.

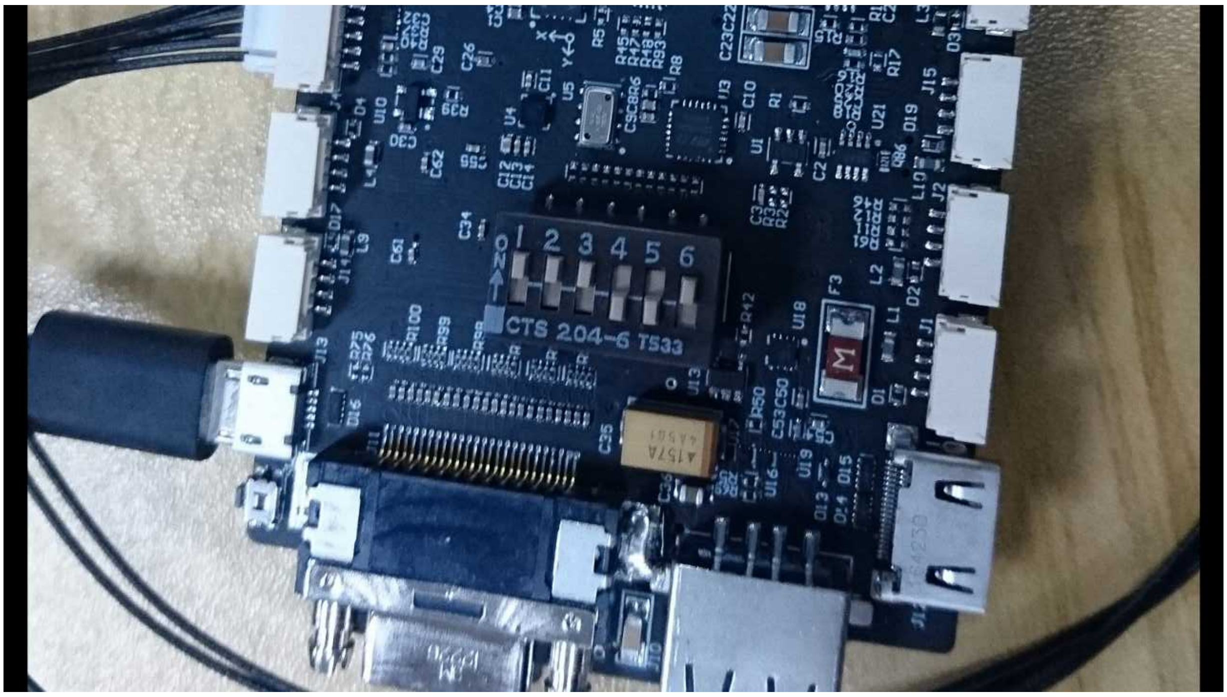
Program the controller

Boot

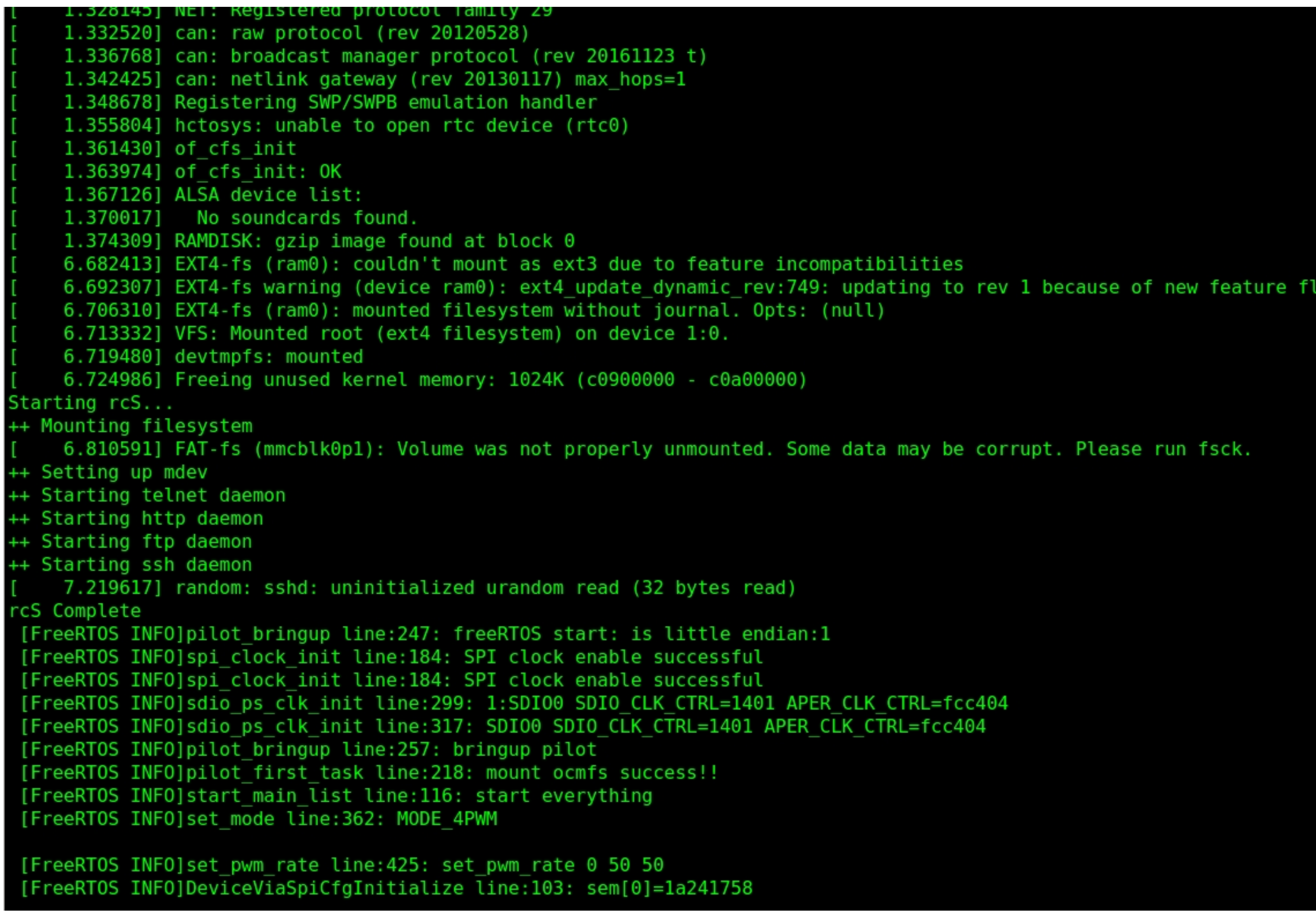
Devkit support boot from sd card, copy these file to sd card:

```
1.PhenixPro_Devkit/BOOT.BIN
2.PhenixPro_Devkit/amp_system/linux_image/devicetree.dtb
3.PhenixPro_Devkit/amp_system/linux_image/uImage
4.PhenixPro_Devkit/amp_system/linux_image/uramdisk.image.gz
```

set DIP switch like the image below(it means boot from sd card):



Insert SD card to devkit, connect micro usb to PC, open uart tools(recommand SecureCRT) with 115200 baudrate, and power on, you will see print information on console:



```
[    1.328145] NET: Registered protocol family 29
[    1.332520] can: raw protocol (rev 20120528)
[    1.336768] can: broadcast manager protocol (rev 20161123 t)
[    1.342425] can: netlink gateway (rev 20130117) max_hops=1
[    1.348678] Registering SWP/SWPB emulation handler
[    1.355804] hctosys: unable to open rtc device (rtc0)
[    1.361430] of_cfs_init
[    1.363974] of_cfs_init: OK
[    1.367126] ALSA device list:
[    1.370017]   No soundcards found.
[    1.374309] RAMDISK: gzip image found at block 0
[    6.682413] EXT4-fs (ram0): couldn't mount as ext3 due to feature incompatibilities
[    6.692307] EXT4-fs warning (device ram0): ext4_update_dynamic_rev:749: updating to rev 1 because of new feature fl
[    6.706310] EXT4-fs (ram0): mounted filesystem without journal. Opts: (null)
[    6.713332] VFS: Mounted root (ext4 filesystem) on device 1:0.
[    6.719480] devtmpfs: mounted
[    6.724986] Freeing unused kernel memory: 1024K (c0900000 - c0a00000)
Starting rcS...
++ Mounting filesystem
[    6.810591] FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.
++ Setting up mdev
++ Starting telnet daemon
++ Starting http daemon
++ Starting ftp daemon
++ Starting ssh daemon
[    7.219617] random: sshd: uninitialized urandom read (32 bytes read)
rcS Complete
 [FreeRTOS INFO]pilot_bringup line:247: freeRTOS start: is little endian:1
 [FreeRTOS INFO]spi_clock_init line:184: SPI clock enable successful
 [FreeRTOS INFO]spi_clock_init line:184: SPI clock enable successful
 [FreeRTOS INFO]sdio_ps_clk_init line:299: 1:SDIO0 SDIO_CLK_CTRL=1401 APER_CLK_CTRL=fcc404
 [FreeRTOS INFO]sdio_ps_clk_init line:317: SDIO0 SDIO_CLK_CTRL=1401 APER_CLK_CTRL=fcc404
 [FreeRTOS INFO]pilot_bringup line:257: bringup pilot
 [FreeRTOS INFO]pilot_first_task line:218: mount ocmfs success!!
 [FreeRTOS INFO]start_main_list line:116: start everything
 [FreeRTOS INFO]set_mode line:362: MODE_4PWM

 [FreeRTOS INFO]set_pwm_rate line:425: set_pwm_rate 0 50 50
 [FreeRTOS INFO]DeviceViaSpiCfgInitialize line:103: sem[0]=1a241758
```

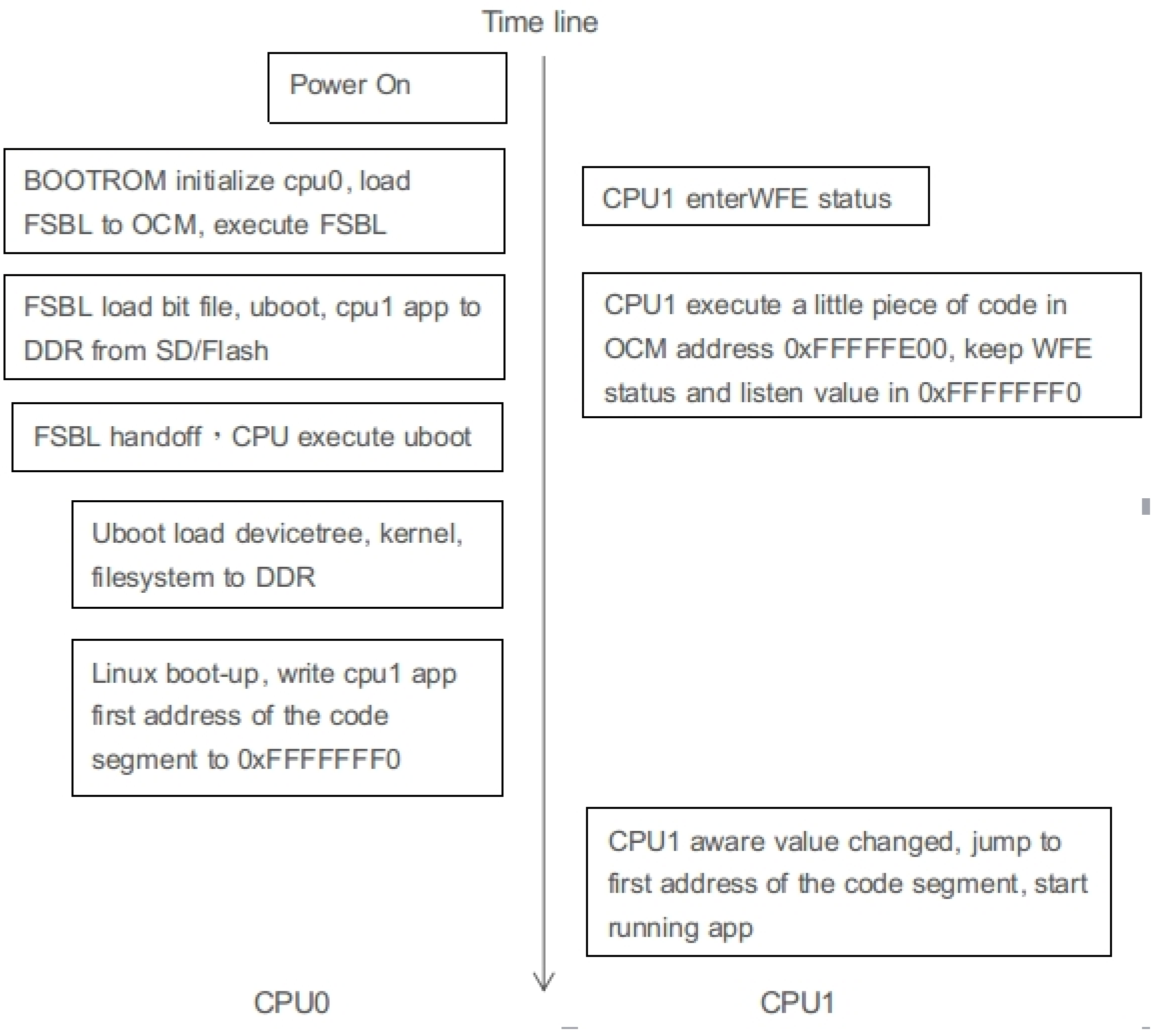
Build Software

This chapter will show you a simple demo about running an AMP system(Asymmetric Multi-Processing 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

2.1.0 Principle

Simply put, the principle of Amp system in zynq is cpu0 wake up cpu1, now, let't see how it works:

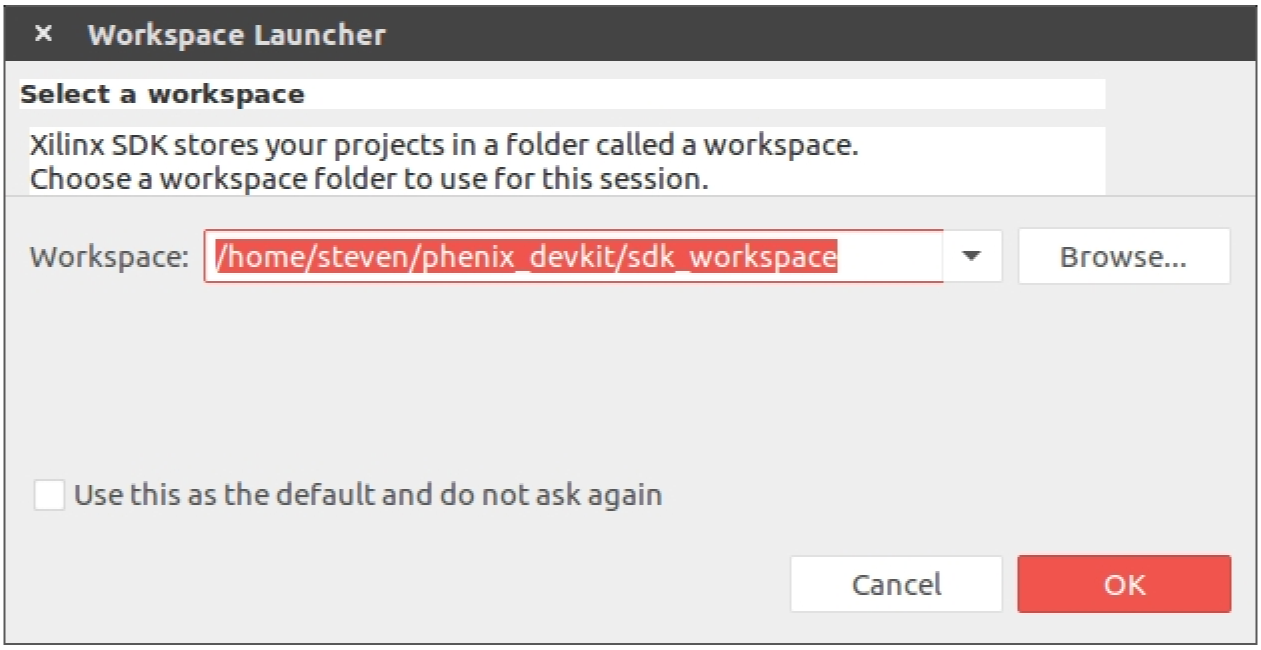

Time line
Power On
BOOTROM initialize cpu0, load FSBL to OCM, execute FSBL
FSBL load bit file, uboot, cpu1 app to DDR from SD/Flash
FSBL handoff，CPU execute uboot
Uboot load devicetree, kernel, filesystem to DDR
Linux boot-up, write cpu1 app first address of the code segment to 0xFFFFFFF0
CPU1 enterWFE status
CPU1 execute a little piece of code in OCM address 0xFFFFFE00, keep WFE status and listen value in 0xFFFFFFF0
CPU1 aware value changed, jump to first address of the code segment, start running app
CPU0
CPU1
Screenshot from 2017-03-20 13-38-38

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 0xfffffff0 adress. after linux startup, it will write the code segment first address of cpu1 app into 0xfffffff0. Once cpu1 realize the value is changed to nonzero, then the pc pointer of cpu1 will jump to the adress which is written at 0xfffffff0, this is how the whole system startup.

2.1.1 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 dirctory as workspace and click OK:



Select a workspace

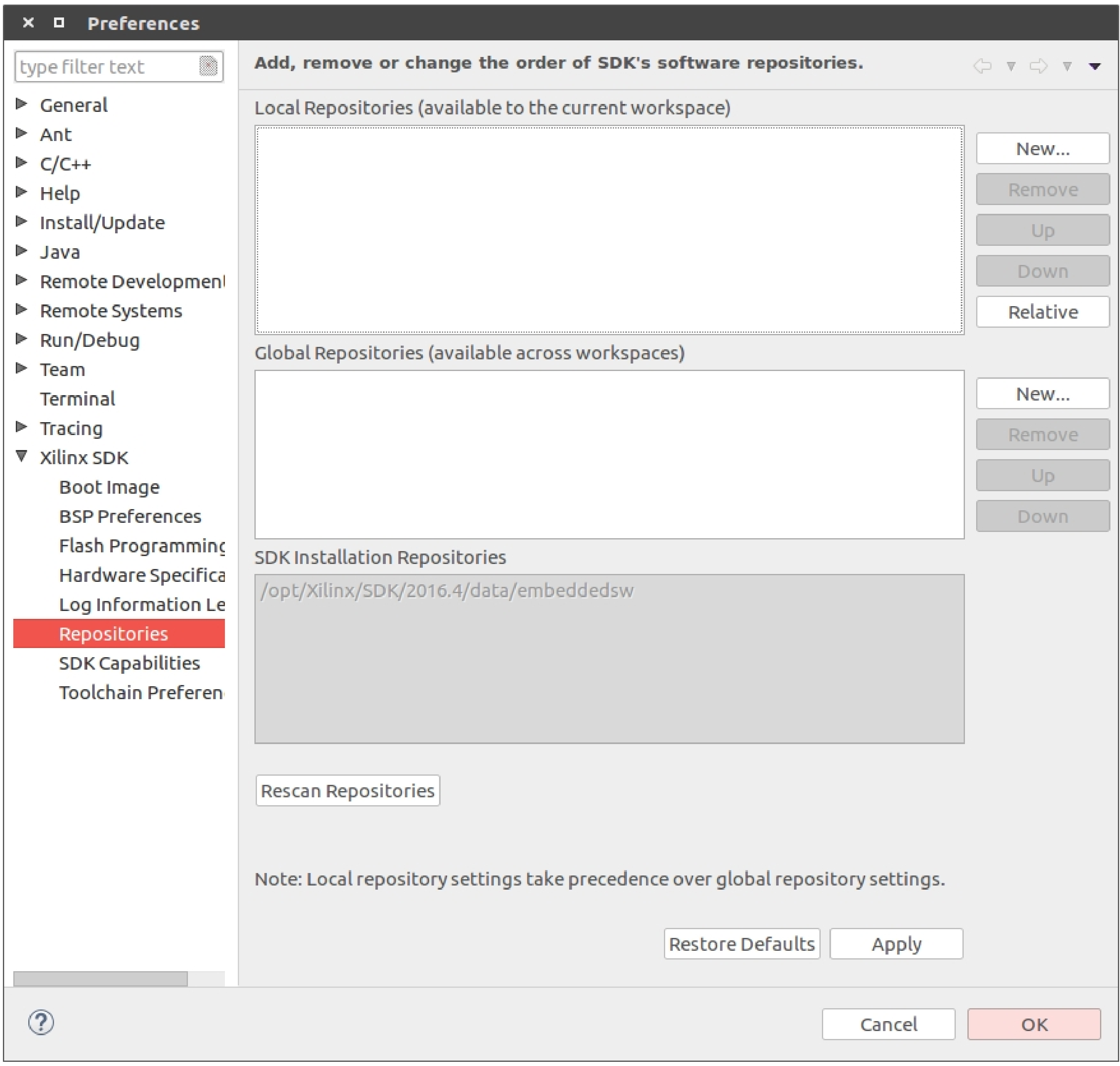
Xilinx SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: /home/steven/phenix_devkit/sdk_workspace

☐ Use this as the default and do not ask again

Screenshot from 2017-03-14 16-48-32

Select **Xilinx_tools > Repositories**



Add, remove or change the order of SDK's software repositories.

Local Repositories (available to the current workspace)

Global Repositories (available across workspaces)

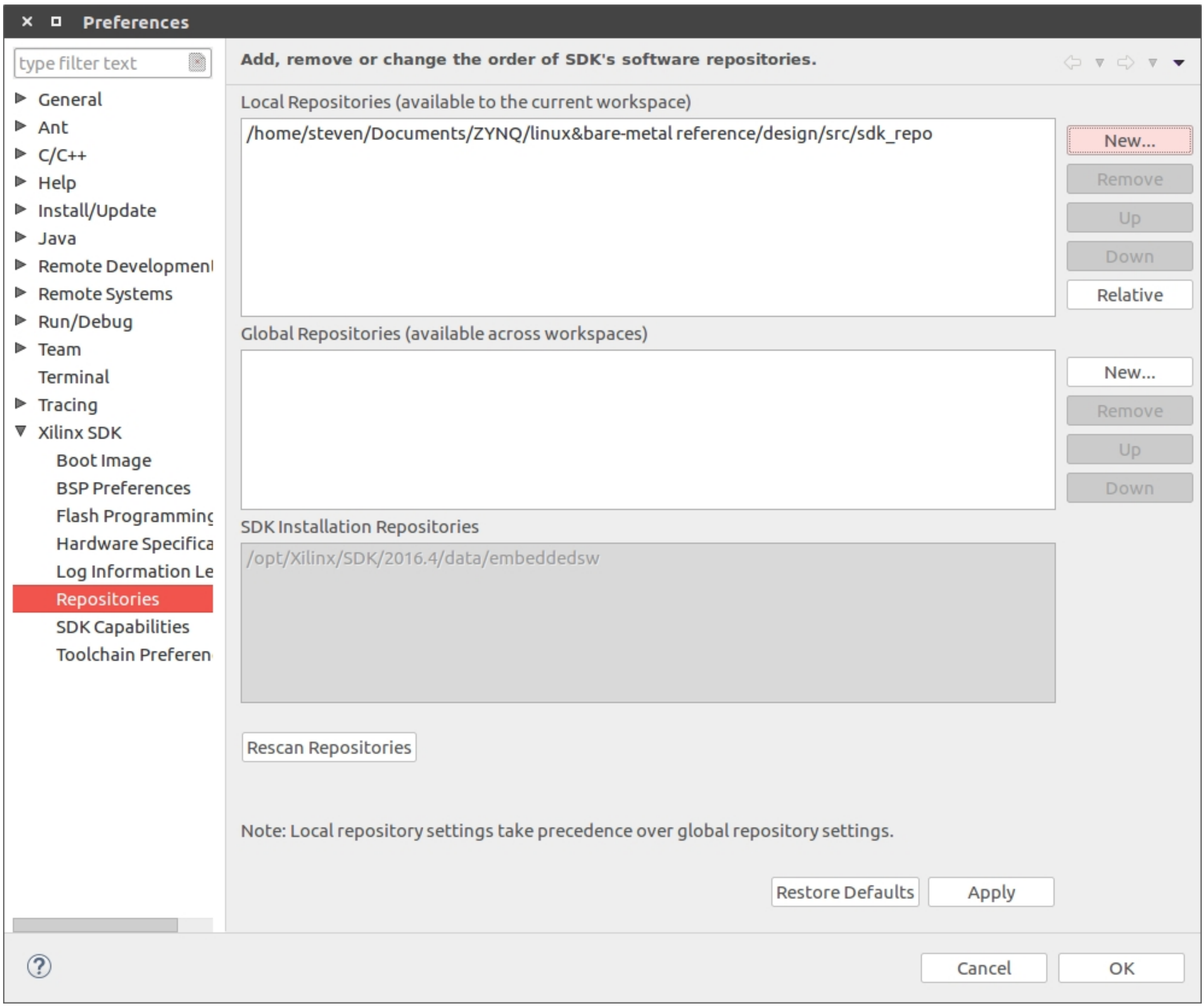
SDK Installation Repositories

/opt/Xilinx/SDK/2016.4/data/embeddedsw

Note: Local repository settings take precedence over global repository settings.

Screenshot from 2017-03-15 10-14-40

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



Add, remove or change the order of SDK's software repositories.

Local Repositories (available to the current workspace)

/home/steven/Documents/ZYNQ/linux&bare-metal reference/design/src/sdk_repo

Global Repositories (available across workspaces)

SDK Installation Repositories

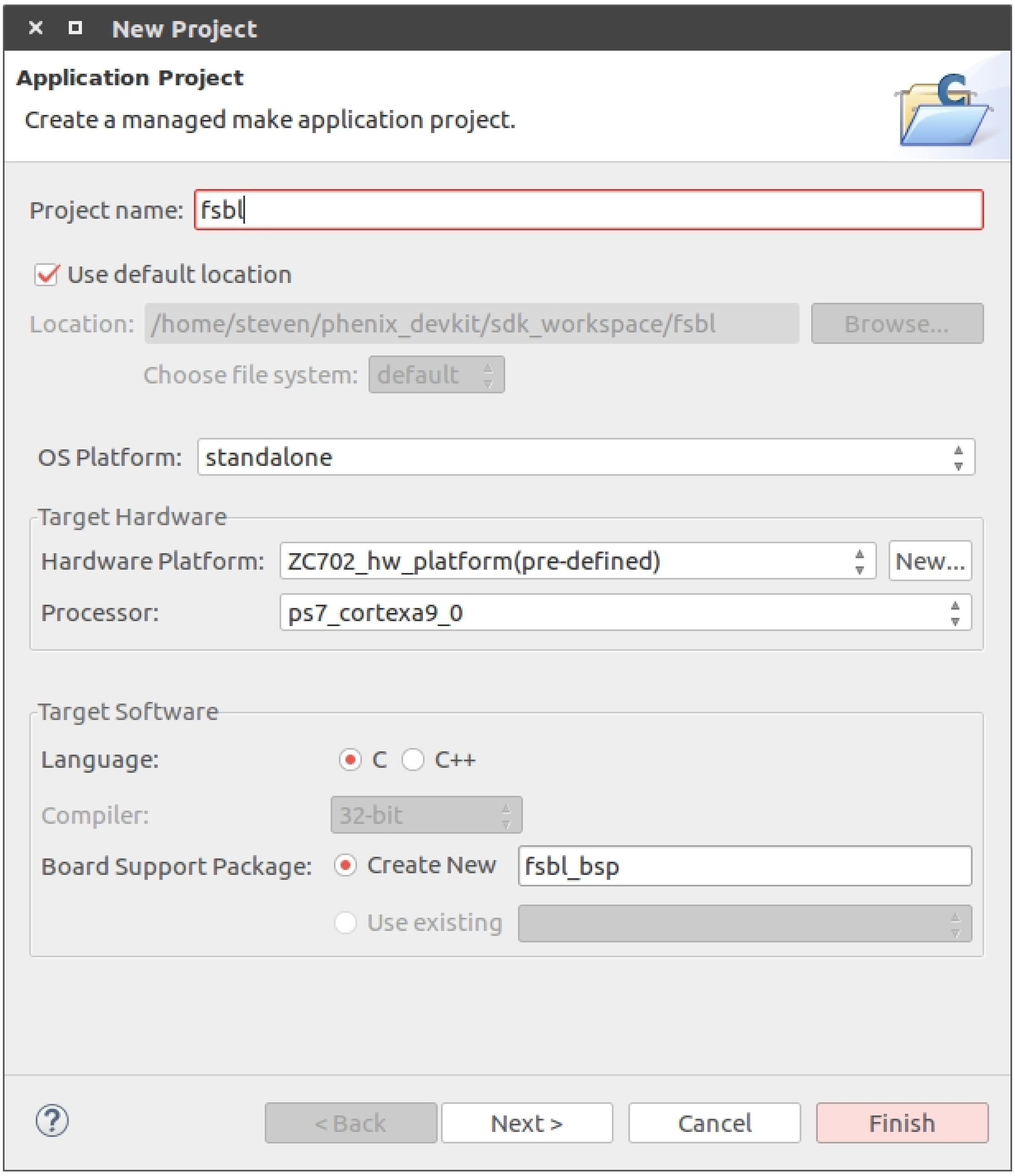
/opt/Xilinx/SDK/2016.4/data/embeddedsw

Note: Local repository settings take precedence over global repository settings.

Screenshot from 2017-03-15 10-16-55

2.1.2 Create FSBL Application

Select **File > New > Application_Project**


New Project

Application Project

Create a managed make application project.

Project name: fsbl

[✓] Use default location

Location: /home/steven/phenix_devkit/sdk_workspace/fsbl

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: ZC702_hw_platform(pre-defined)

Processor: ps7_cortexa9_0

Target Software

Language: [✓ C] C++

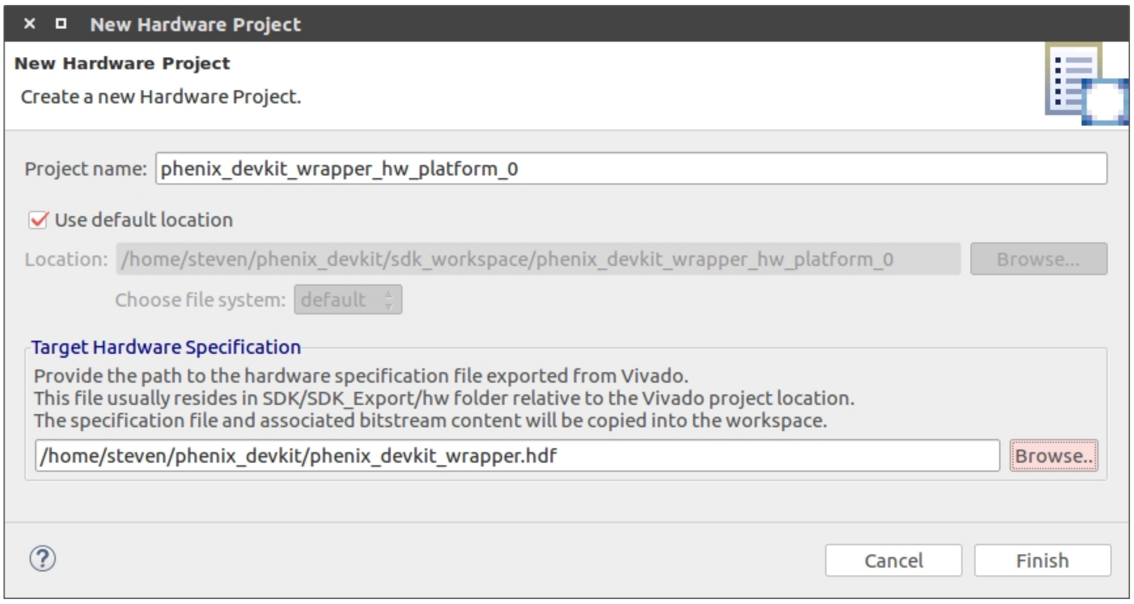
Compiler: 32-bit

Board Support Package: [✓ Create New] fsbl_bsp

Use existing

Screenshot from 2017-03-15 10-30-11

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



New Hardware Project

New Hardware Project

Create a new Hardware Project.

Project name: phenix_devkit_wrapper_hw_platform_0

[✓] Use default location

Location: /home/steven/phenix_devkit/sdk_workspace/phenix_devkit_wrapper_hw_platform_0

Choose file system: default

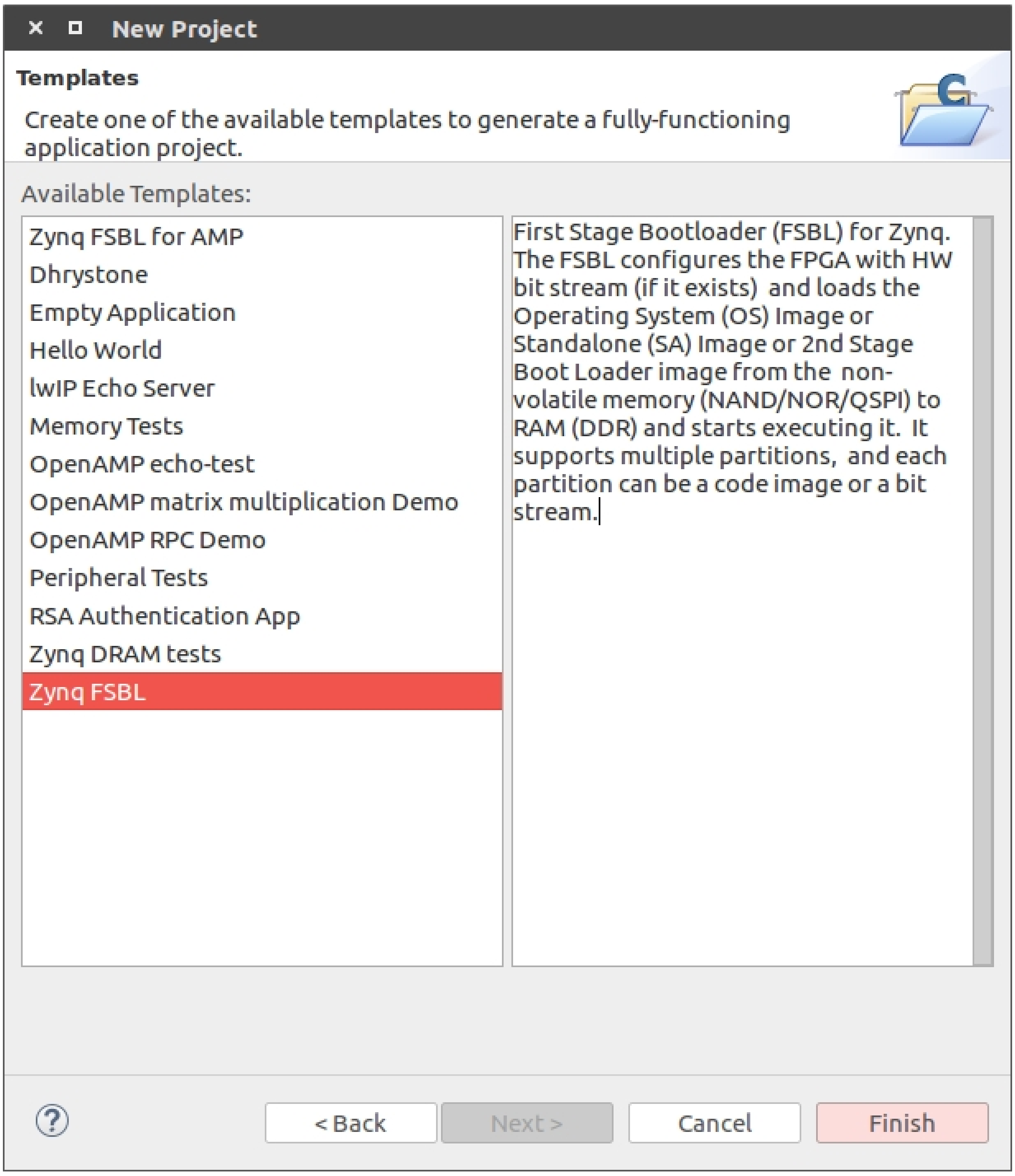
Target Hardware Specification

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.

/home/steven/phenix_devkit/phenix_devkit_wrapper.hdf

Screenshot from 2017-03-15 10-31-51

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



Create one of the available templates to generate a fully-functioning application project.

Available Templates:

- Zynq FSBL for AMP
- Dhrystone
- Empty Application
- Hello World
- lwIP Echo Server
- Memory Tests
- OpenAMP echo-test
- OpenAMP matrix multiplication Demo
- OpenAMP RPC Demo
- Peripheral Tests
- RSA Authentication App
- Zynq DRAM tests
- Zynq FSBL

First Stage Bootloader (FSBL) for Zynq. The FSBL configures the FPGA with HW bit stream (if it exists) and loads the Operating System (OS) Image or Standalone (SA) Image or 2nd Stage Boot Loader image from the non-volatile memory (NAND/NOR/QSPI) to RAM (DDR) and starts executing it. It supports multiple partitions, and each partition can be a code image or a bit stream.

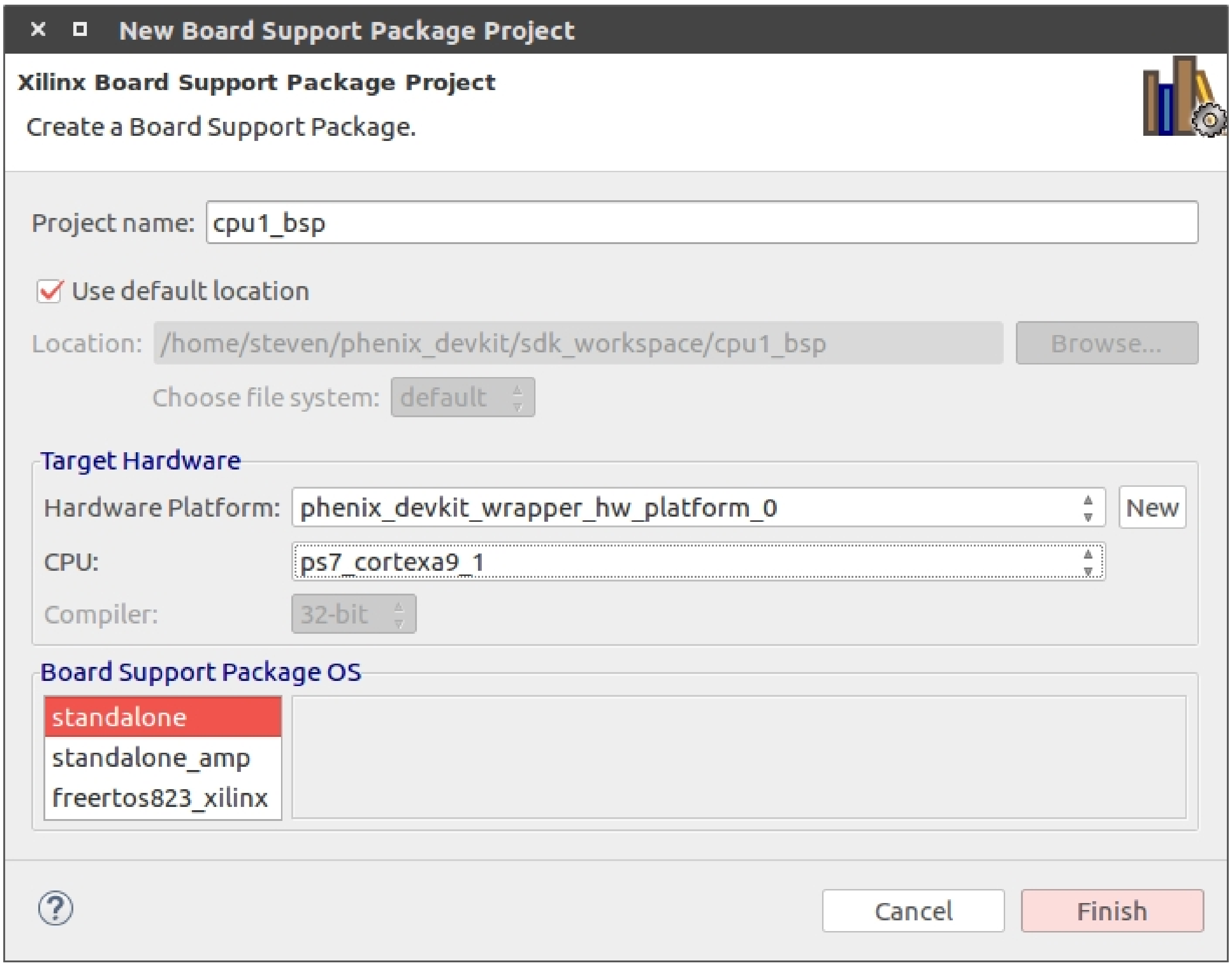
Screenshot from 2017-03-15 11-13-50

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

2.1.3 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
Browse...
Target Hardware
New
Board Support Package OS
Cancel
Finish

Create a Board Support Package.

Project name: cpu1_bsp

✓ Use default location

Location: /home/steven/phenix_devkit/sdk_workspace/cpu1_bsp

Choose file system: default

Hardware Platform: phenix_devkit_wrapper_hw_platform_0

CPU: ps7_cortexa9_1

Compiler: 32-bit

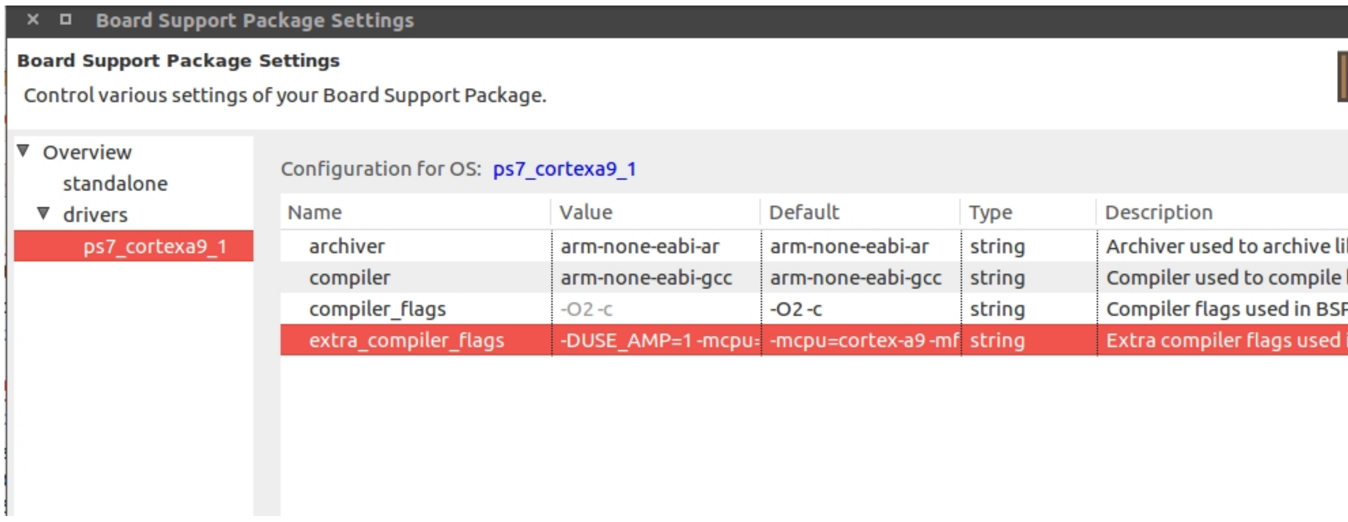
standalone
standalone_amp
freertos823_xilinx

Screenshot from 2017-03-15 11-26-33

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

and stdout to **ps7_uart_1**

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


Board Support Package Settings

Board Support Package Settings

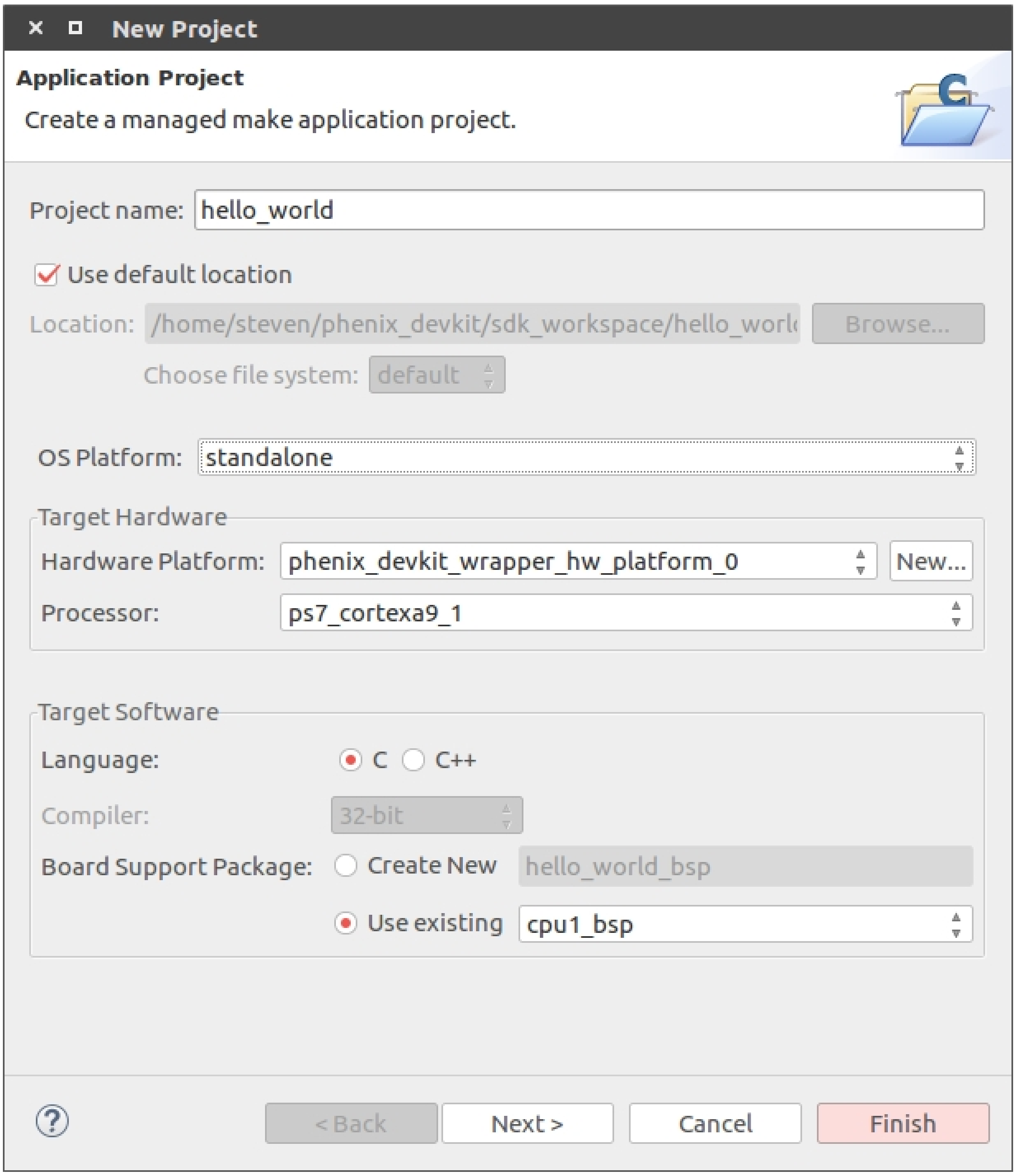
Control various settings of your Board Support Package.

Configuration for OS: ps7_cortexa9_1

Name	Value	Default	Type	Description
archiver	arm-none-eabi-ar	arm-none-eabi-ar	string	Archiver used to archive li
compiler	arm-none-eabi-gcc	arm-none-eabi-gcc	string	Compiler used to compile
compiler_flags	-O2 -c	-O2 -c	string	Compiler flags used in BSP
extra_compiler_flags	-DUSE_AMP=1 -mcpu=	-mcpu=cortex-a9 -mf	string	Extra compiler flags used

Screenshot from 2017-03-15 13-28-40

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: hello_world

[✓] Use default location

Location: /home/steven/phenix_devkit/sdk_workspace/hello_worl

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: phenix_devkit_wrapper_hw_platform_0

Processor: ps7_cortexa9_1

Target Software

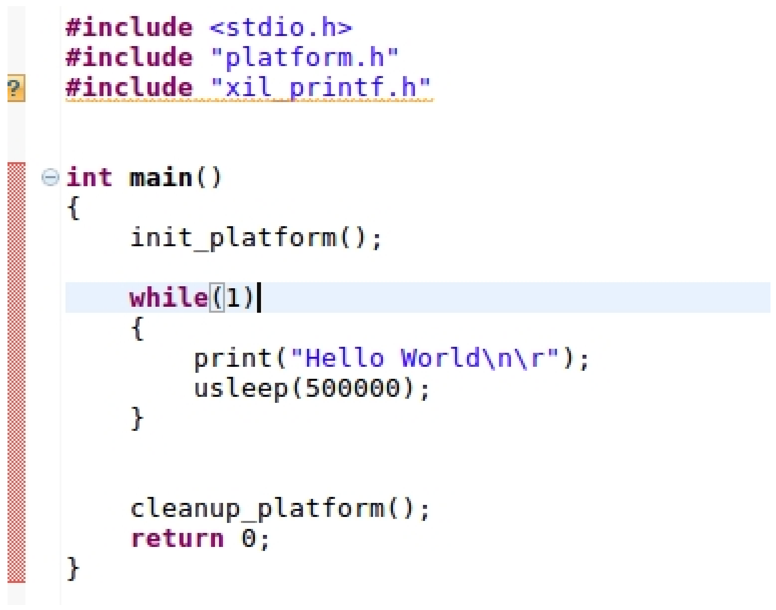
Language: [✓ C] C++

Compiler: 32-bit

Board Support Package: Create New hello_world_bsp [✓ Use existing] cpu1_bsp

Screenshot from 2017-03-15 13-35-45

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;
}
```

Screenshot from 2017-03-15 13-46-22

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_ddr_0_S_AXI_BASEADDR** as 0x1A000000, **Size** as 0x1FF00000, **Stack Size** as 0x800000(8MB), **Heap Size** as 0x1000000(16MB), now, we can complie our code.

2.1.4 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):

make the configuration effective:

```
make zynq_zed_config
```

complie u-booot:

```
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
  ```

2.1.5 Linux Kernel For CPU0

Get kernel code from github:

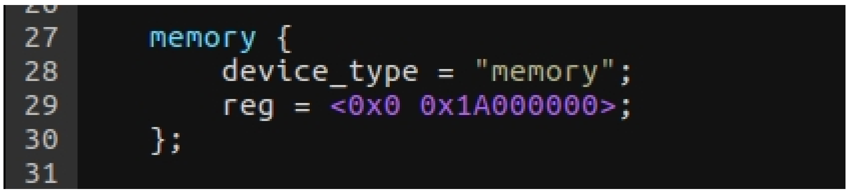
```
git clone -b master --single-branch https://github.com/Xilinx/linux-xlnx.git
```

compile kernel:

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

2.1.6 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:



```
    memory {
        device_type = "memory";
        reg = <0x0 0x1A000000>;
    };
```

Screenshot from 2017-03-15 14-40-18

generate dtb file with command

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

2.1.7 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:

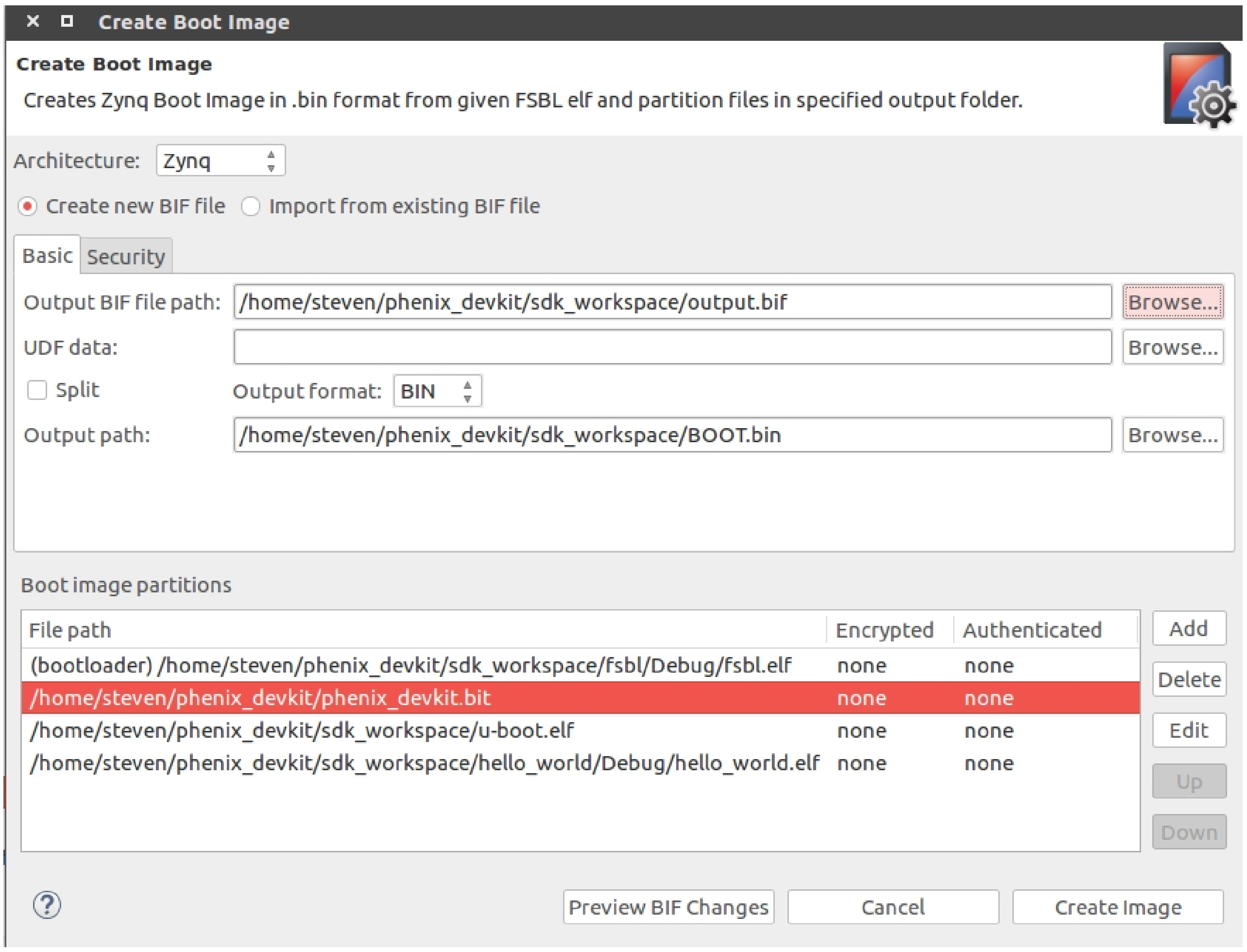
```
gunzip arm_ramdisk.image.gz
chmod u+rwx arm_ramdisk.image
mkdir tmpmnt
sudo mount -o loop arm_ramdisk.image tmpmnt/
```

As we discussed in chapter 3.1, we need **rwmem.elf** to write address 0xfffffff0 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
```

2.1.8 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

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

UDF data:

☐ Split   Output format: BIN

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

Boot image partitions

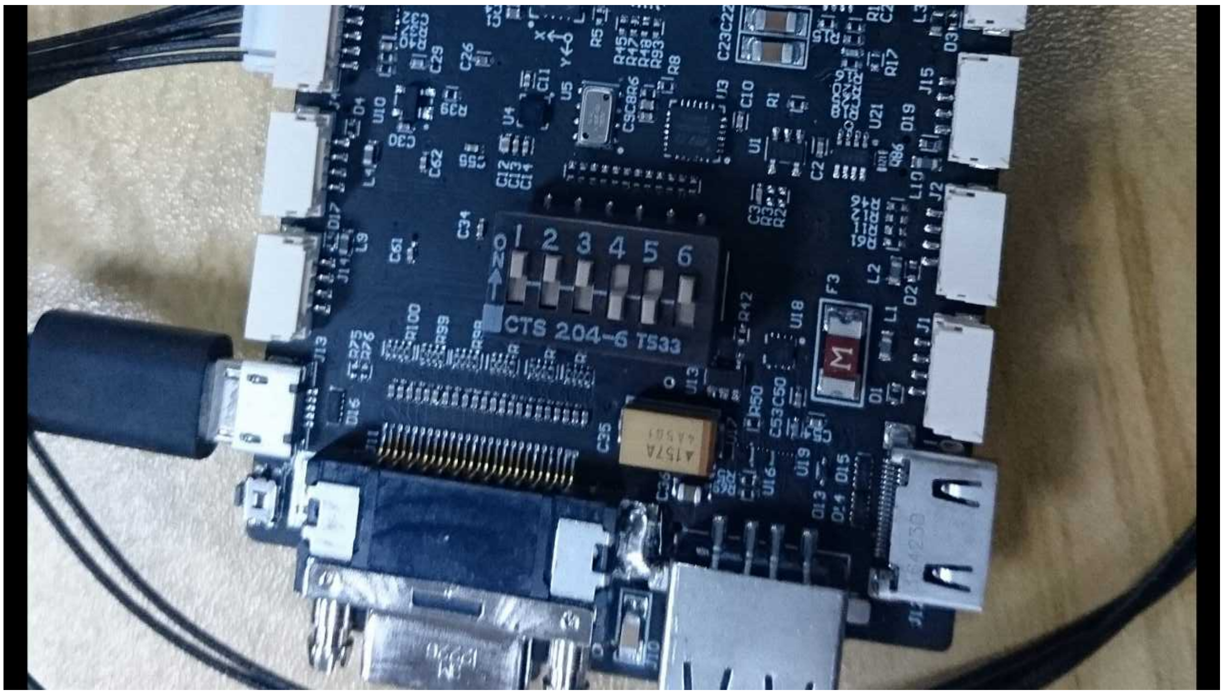
File path	Encrypted	Authenticated
(bootloader) /home/steven/phenix_devkit/sdk_workspace/fsbl/Debug/fsbl.elf	none	none
/home/steven/phenix_devkit/phenix_devkit.bit	none	none
/home/steven/phenix_devkit/sdk_workspace/u-boot.elf	none	none
/home/steven/phenix_devkit/sdk_workspace/hello_world/Debug/hello_world.elf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

Screenshot from 2017-03-15 17-03-51

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

2.1.9 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:

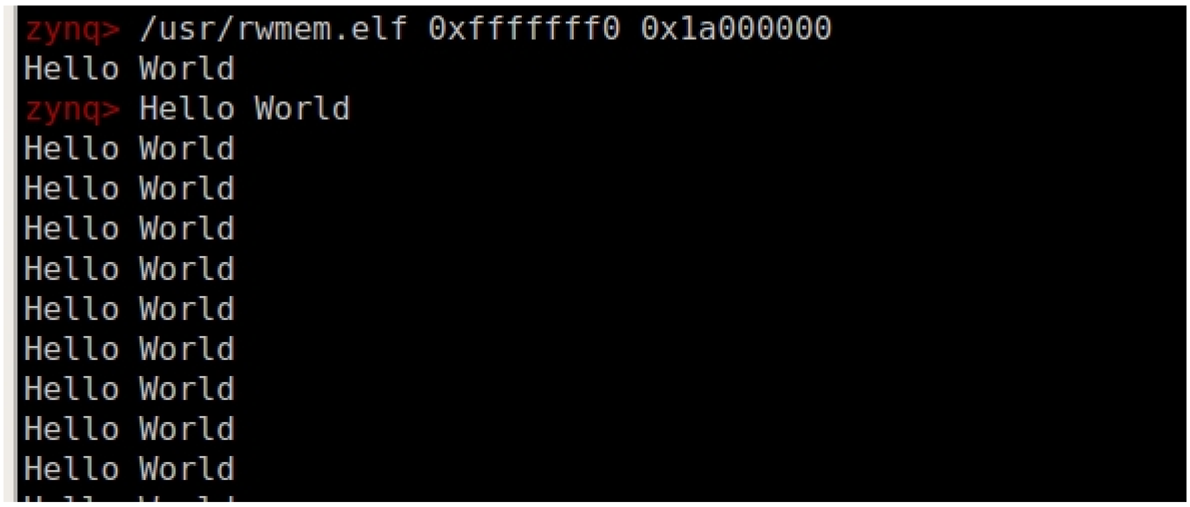


power-on devkit.

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 0xfffffff0 0x1a000000
Hello World
zynq> Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

Screenshot from 2017-03-15 18-20-28