**Contents**

[Overview 1](#_Toc481687006)

[Setting up the PetaLinux Environment 3](#_Toc481687007)

[Installing Vivado 8](#_Toc481687008)

[Building PetaLinux project and Applications for Linux Master/FreeRTOS Remote Configuration 9](#_Toc481687009)

[Using RPMSG Framework With Remoteproc 9](#_Toc481687010)

[Using RPMSG Framework Without Remoteproc 21](#_Toc481687011)

[Creating and Building Remote application in XSDK 24](#_Toc481687012)

[Boot the PetaLinux Project 32](#_Toc481687013)

[Boot a PetaLinux Image on QEMU 32](#_Toc481687014)

[Boot a PetaLinux Image on Hardware 33](#_Toc481687015)

[Boot a PetaLinux Image on Hardware with SD Card 36](#_Toc481687016)

[Boot a PetaLinux Image on Hardware with JTAG 41](#_Toc481687017)

[Login Linux with SSH 45](#_Toc481687018)

[Executing Tests for Linux Master/FreeRTOS Remote Comfiguration 49](#_Toc481687019)

[Tests Using RPMSG Framework with Remoteproc 49](#_Toc481687020)

[Tests Using RPMSG Framework without Remoteproc 51](#_Toc481687021)

# Overview

通常在AMP（非对称多处理）配置中，会采用在不同的处理核上运行不同的软件环境并执行各自的代码程序，各核心之间通力合作实现处理器性能的提升。在AMP系统中，所谓的主处理器通常是指最先启动且主要负责管理其他CPU以及这些CPU上软件环境的CPU处理器。而远程处理器简单的讲就是指被主核所支配的CPU。主软件环境通常需要进行远程核心的管理以及任务调度，在需要进行计算任务分配时，将选择性地驱动远程内核上的远程软件环境，并交代任务信息。

OpenAMP软件框架为开发AMP系统提供了必要的API函数。OpenAMP 是Mentor Graphics 与赛灵思公司为了使在AMP系统的设计中开发出的RTOS和裸机程序能够与开源Linux社区提供的接口进行互通讯，而共同通过的一个标准化的嵌入式多核框架。的全称是 ,即开源的非对称多处理框架。OpenAMP框架提供了非对称多处理系统软件开发所需的软件组件。OpenAMP框架是一种软件框架，这种软件框架能够为非对称多处理(AMP)系统开发人员提供三大重要组件：

* 该组件是RPMsg组件的实现基础，在使用Remoteproc和RPMsg进行核间通信时使用到了这个概念。在后面会讲解Virtio的实现原理。
* :该组件用于主机上，实现对远程处理器及其相关软件环境进行生命周期管理(LCM)，说白了就是能够对远程处理器实现启动和关闭、加载远程处理器运用程序到内存适合位置，并完成virtio和RPMSG设备的注册等。

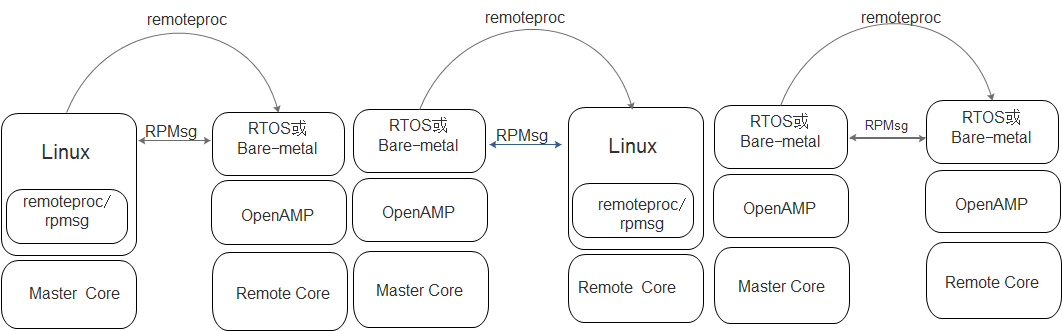
* ：实现多核处理器IPC通信的通道，基于virtio组件实现。

软件框架为开发人员提的标准化接口使得在AMP结构中设计和管理软硬件变得更为简单。和基础架构最初是由德州仪器设计开发的，并专门用在内核。使用该基础架构可以实现在主软件环境下对远程软件环境进行生命周期管理并进行核心间的通讯。但是，所提供的这些基础架构在一定程度上存在着局限性。首先，基础架构通常都是默认将作为其主操作系统，然而并不支持把作为非对称多处理结构配置中的远程软件环境。另外，和API只能从系统的内核空间中获得，由于缺乏相应的等效API或库，这些组件并没能用在其他的一些操作系统上，这造成了适用范围上的局限性。

软件框架很好的解决了以上的问题。该框架是和赛灵思共同通过地一个开源项目，其源代码可以在GitHub（https://github.com/OpenAMP/open-amp）上获得或从该文档相同目录下src文件夹中获得（open-amp-openamp\_zynq\_kernel\_3.17）。

框架能够被很好地使用在实时操作系统或者裸机软件环境中，使得在这些软件环境下能够使用与Linux系统具有相同功能的和，且之间在使用上具有很好的兼容性。框架中的和基础架构使用VirtIO作为传输抽象层。下图显示了框架支持的各种软件环境及配置，由图所示，该框架可以被用在、或者裸机环境下，通过和组件实现不同软件环境之间的通讯。

嵌入式多核框架同样可以作为独立库使用。允许主处理器RTOS或者裸机软件环境中的程序对远程处理器进行生命周期管理并且使用与远程处理器进行通信。



该文档主要基于Linux master/FreeRTOS remote配置，讲解如何使用OpenAMP框架实现多核处理器核间通信，包括开发环境的搭建、工程的建立和编译等，并在需要的时候介绍所涉及的Virtio、RPMsg组件的工作原理。在开始设计开发之前，首先需要搭建相应的开发环境。

# Setting up the PetaLinux Environment

本设计意在实现基于Cortex-A9多核处理器的核间通讯，所采用的硬件开发平台是Xilinx Zynq-7000 All Programmable SoC ZC702 Evaluation Kit(以下统一简称为ZC702EVK)，软件开发环境为petalinux 和 SDK。该平台上的处理器有两个ARM CPU核，在AMP系统设计中通常是在这两个核中运行不同的软件环境。本设计采用一个处理核运行Linux OS，而另一个核运行RTOS或Baremetal来研究多核处理器核间通信过程。本节将主要介绍PetaLinux嵌入式操作系统环境搭建过程和PetaLinux Tools基本使用方法。

在安装PetaLinux之前应确保满足以下要求，这里以v2015.4版本的Petalinux为例进行讲解，不同版本之间可能会有不同，但大致差不多，可以先按照下述要求进行，若还是出现安装失败，可以参考xilinx提供的reference guide(文档编号为UG1144，根据使用的petalinux版本选择版本相近的文档)：

* 个人要求：掌握Linux操作系统基本使用基础，包括熟悉Linux操作环境，基本指令的使用(比如ls,cd,mkdir,chmod,sh,source,软件的安装和卸载指令等等)，主要是要习惯Linux下通过输入命令行完成工作的方式，然后加上多百度多搜索就差不多了。
* 工作站最低配置要求:
* 4 GB RAM
* Pentium 4 2GHz CPU 或更高
* 至少5 GB的空闲硬盘空间
* 支持的OS:
* RHEL 6.5/6.6/7.0(64-bit)
* CentOS 7.0 (64-bit)
* SUSE Enterprise 12.0 (64-bit)
* Ubuntu 14.0.4 (64-bit)
* 在进行一些操作时需要能够取得root最高管理权限
* PetaLinux需要一系列标准开发工具和库，所以在主工作站安装PetaLinux之前应确保安装了表1-1列出的开发工具和库。该表同时也给出了在不同Linux系统下这些工具和库对应的安装包。

表1-1 相关开发工具和库

不同发行版的linux操作系统安装开发工具和库的方式可能不同，以centos环境下安装bison为例:

yum install bison

[yum](http://www.yanghengfei.com/tag/yum/)（全称为 Yellow dog Updater, Modified）是一个在Fedora和RedHat，Centos以及SUSE中的Shell前端软件包管理器。基于RPM包管理，能够从指定的服务器自动下载RPM包并且完成安装，可以自动处理依赖性关系，并且一次安装所有依赖的软体包，无须繁琐地一次次下载、安装。yum提供了查找、安装、删除某一个、一组甚至全部软件包的命令，而且命令简洁而又好记。

具体如何安装每一个开发工具和库，可参考网络搜索结果，需要注意的是尽量采用以上方式进行安装，即通过yum这样的软件包管理器进行安装，这样可以省去很多麻烦，节省开发环境搭建时间。

在确保以上条件满足之后就可以开始安装PetaLinux，在Xilinx官网下载PetaLinux安装包，这里选用petalinux-v2015.4。PetaLinux的安装过程非常简单，在Linux中打开终端（Terminal），在控制台窗口输入以下命令进行安装：





PetaLinux Tools将会被安装在/opt/pkg/petalinux-v2015.4-final目录下，当然也可不附加/opt/pkg这个选项，这样PetaLinux会被默认安装在当前目录下，安装目录可自定义。安装完PetaLinux之后需要执行安装目录下的settings.sh或settings.csh脚本进行工作环境设置。对于默认Shell是bash的执行：



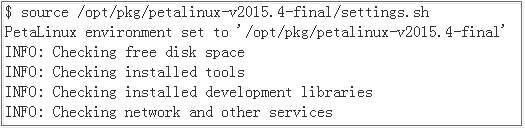
对于默认shell是Cshell的情况执行settings.csh脚本：



如果不知道自己系统使用的是bash还是cshell或者是其他，可以通过指令

echo $SHELL进行查看。

成功执行相应脚本文件后，可以在终端上输出类似以下信息：

可以通过Linux echo命令来确认PetaLinux环境变量是否被正确设置：



若输出PetaLinux的安装路径，则说明PetaLinux工作环境变量已经被设置，且可以开始使用PetaLinux Tools。至此，我们可以使用PetaLinux Tools实现工程创建、内核配置、应用程序开发以及设计仿真等。

**Tips:**每次打开终端时都应该source一次setting.sh(或setting.csh)，因为该设置是暂时，为了避免每次都要重新手动设置，可以编辑.bashrc文件。



通过指令ls –al可以看到.bashrc文件，用编辑器打开后，在文件末尾加上保存即可。这样每次打开终端时，都会自动执行以上语句

PetaLinux SDK为设计者提供了灵活实用的设计工具，设计者可以根据自身需要添加相应的选项，定制符合自身设计要求的工程。这些设计工具主要有以下6种：

* petalinux\_create：
* petalinux\_config
* petalinux\_build
* petalinux\_package
* petalinux\_boot
* petalinux\_util

通常在使用Petalinux tools进行开发的时候有一套常用的设计流程，表2-2中利用petalinux进行设计的工作流程以及相关指令的用途和使用形式。目前不需要太纠结每一个指令的具体使用方法，在实际需要的时候再来学习，petalinux tools的使用可参考ug1144-petalinux-tools-reference-guide（见该文档同等目录下Document文件夹）。

表1-2 petalinux工程设计流程

表1-2列出了petalinux设计步骤流程，从工程建立、系统级配置、Linux内核配置、系统编译到最后的系统测试和部署。当该表所列出的设计步骤并非都是必须的，应根据实际设计要求进行取舍。

# Installing Vivado

# Building PetaLinux project and Applications for Linux Master/FreeRTOS Remote Configuration

这一节主要介绍Linux Master/FreeRTOS Remote配置下的核间通信实例。包括petalinux工程建立，Linux用户空间运用程序、Linux内核空间程序以及FreeRTOS 运用程序的建立和编译。通过之前的介绍我们知道，OpenAMP框架为给实现多核环境下的核间通信提供了三大组件，其中RemoteProc可用来实现master core对remote core进行生命周期管理，这里涉及两种情况，一种是使用remoteproc组件，让master core来完成remote core的启动和将remote core需要执行的运用程序预先加载到合适的内存区域上，然后交由remote core去执行；另一种情况是不使用remoteproc组件，master core和remote core完全相互独立，不需要一个核去控制另一个核。

## Using RPMSG Framework With Remoteproc

1. 在适当目录下创建petalinux工程，假设选择的目录为/home/jdh/project/，工程名字设为demo,则在选定目录下执行以下指令：



Petalinux-create 用于创建工程、用户程序、内核模块等，具体创建的是工程/用户程序/内核模块由-t 选项指定(分别对应为project、apps、modules)，-n 项用于指定工程名字，--template项用于指定硬件平台，这里选择zynq.

成功执行后可以看到类似如下信息：

在目录/home/jdh/project/下会多出一个名为demo的文件夹。

1. 将路径切换至工程目录下 

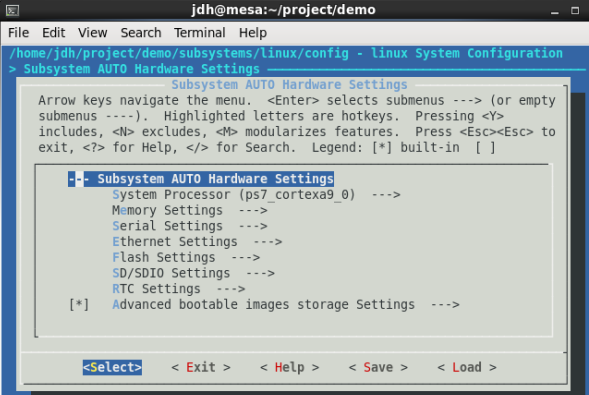
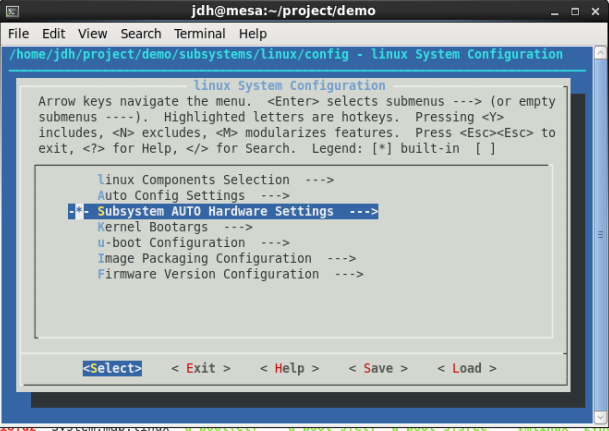
将路径切换到demo文件夹中(之后我们用<proj\_root>代表目录/home/jdh/

project/demo)，列出目录文件如下：

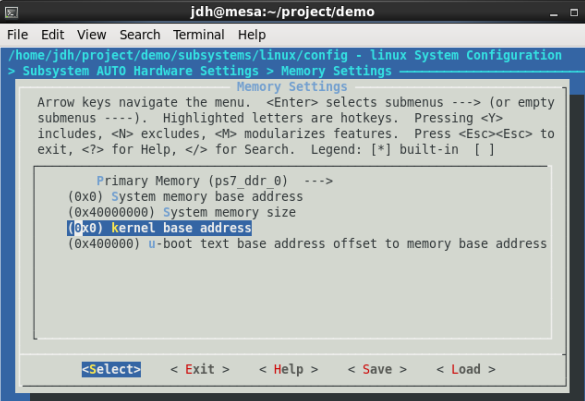
1. 接下来要对工程做相应的配置。首先设置Linux内核基址，用于指定Linux内核被加载到内存的地址。执行命令（该操作必须是在工程根目录下执行，在该例子里为demo文件夹目录下）：



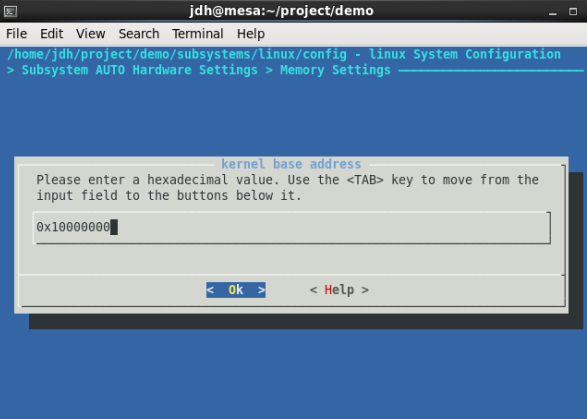
执行该指令后会出现以下配置窗口。

选择Subsystem AUTO Hardware Settings

选择 Memory Settings

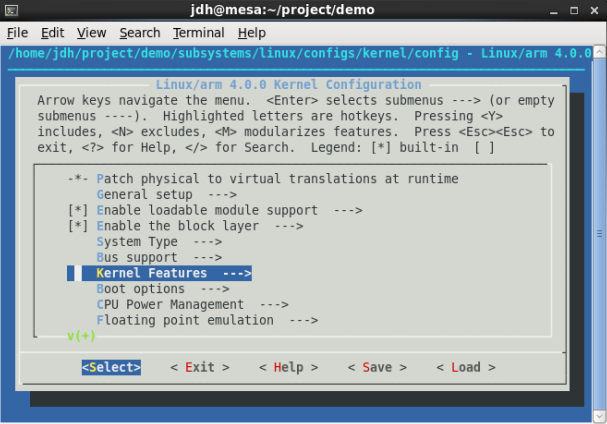
设置kernel base address为0x10000000（理论上该值可以改为其他值，只要

保证运行Linux和FreeRTOS的内存区域不冲突就行）



设置完成后保存退出。

1. 配置内核选项。执行以下指令：



执行成功后出现如下配置窗口

然后做以下设置：

1. 使能loadable module support选项（该项在v2015.4下默认使能）

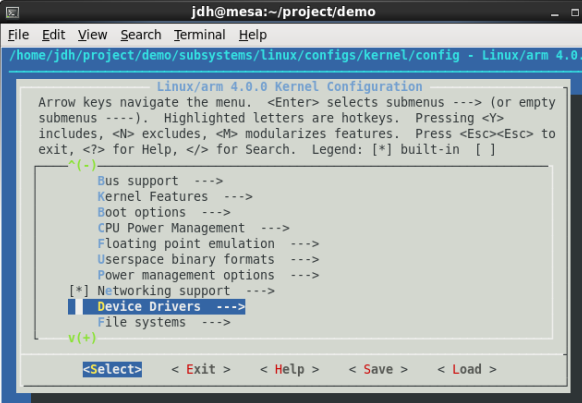
[ \* ] Enable loadable module support

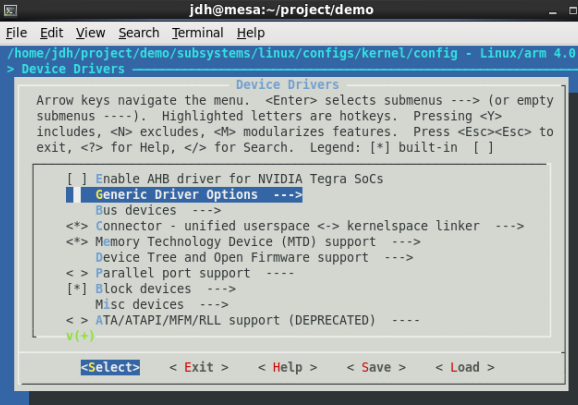
1. 使能user space firmware loading support（v2015.4下默认使能）

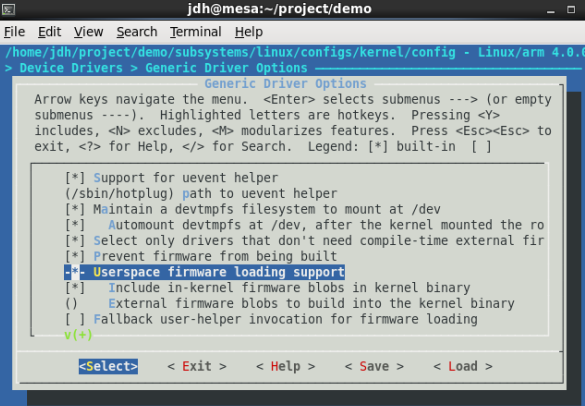
Device Drivers --->

Generic Driver Options --->

< \* > Userspace firmware loading support

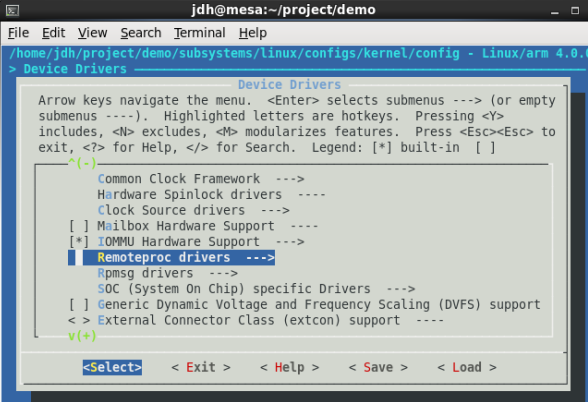


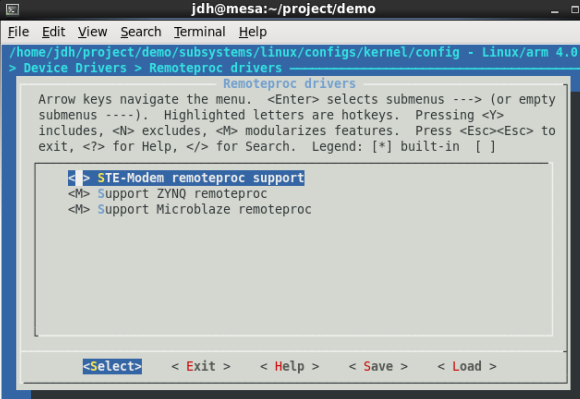


1. 使能remoteproc driver支持（该项在petalinux v2015.4下默认使能）

Device Drivers --->

Remoteproc drivers --->

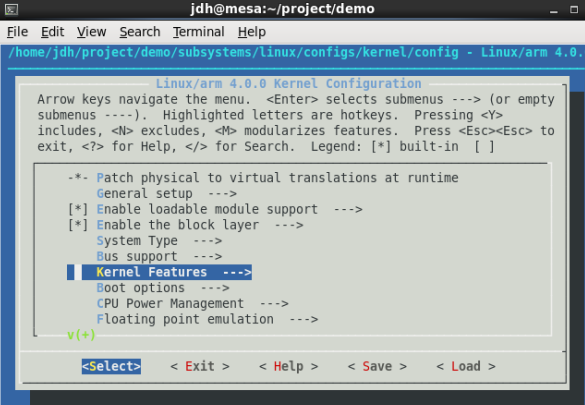
 <M> Support ZYNQ remoteproc

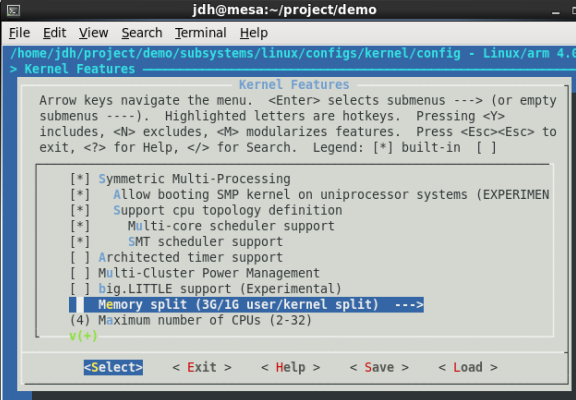


1. 设置内存划分方式。

Kernel Features --->

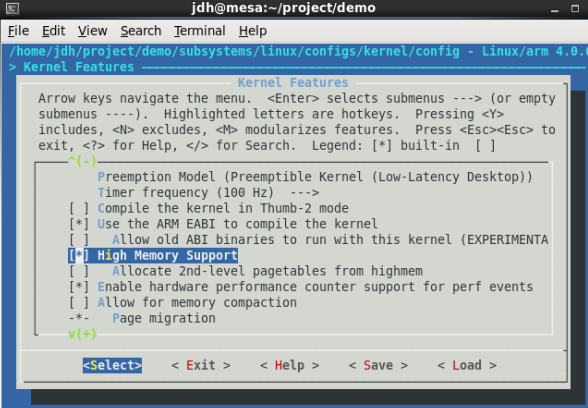
Memory split(...)--->

 (X) 2G/2G user/kernel split



1. 使能High Memory support（该项在petalinux v2015.4下默认使能）

Kernel Features --->

 [ \* ] High Memory Support --->

1. 保存并退出。
2. 添加设备节点（也即编辑设备树文件）

该步骤需要编辑system-top.dts文件（位于subsystems/linux/configs/

device-tree/）,这一步是为了给驱动模块添加设备节点，以便平台驱动能够被probe（可参考Linux驱动开发相应资料）。在System-top.dts中添加以下内容，最终应该长成这样：

/dts-v1/;

/include/ "system-conf.dtsi"

/ { amba {

remoteproc0: remoteproc@0{

compatible = "xlnx,zynq\_remoteproc";

reg = < 0x00000000 0x10000000 >;

firmware = "firmware";

vring0 = <15>; /\* the soft interrupt ID for the master core \*/

vring1 = <14>; /\* the soft interrupt ID for the remote core \*/

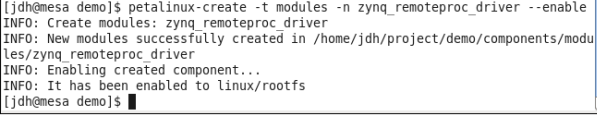
};

};

};

reg指定预留的内存区域，用于remote处理器执行FreeRTOS程序以及两核之间通信所用的共享内存区域，vring0是分配给master core的软中断，当接收到15号中断时意味着共享内存区域有待接收的数据，master core将执行相应的中断处理函数去处理共享内存上的数据；vring1是分配给remote core的软中断，当接收到14号中断时意味着共享内存区域有待接收的数据，remote core将执行相应的中断处理函数去处理共享内存上的数据。当master core往共享内存区域写入数据后会往remote core发送14号中断，通知其去接收数据；相同的，当remote core往共享内存写入数据后，会往master core发送15号中断，通知其去接收数据。

1. 创建内核模块（module）执行指令：





然后将remoteproc\_internal.h和zynq\_remoteproc\_driver.c文件(该文件存在于该文档同等目录src/demo\_using\_remoteproc文件夹下)复制到

<proj\_root>/components/modules/zynq\_remoteproc\_driver/

将rpmsg\_user\_dev\_driver.c文件(该文件存在于该文档同等目录src文件夹下)复制到

<proj\_root>/components/modules/rpmsg\_user\_dev\_driver/

1. 创建Linux用户空间程序



然后将mat\_mul\_demo.c文件(该文件存在于该文档同等目录src/demo\_using\_ remoteproc文件夹下)复制到<proj\_root>/components/apps/mat\_mul\_demo/

之后需要对该目录下的Makefile文件做一定的修改，修改如下：

将原本的

$(APP): $(APP\_OBJS)

$(CC) $(LDFLAGS) -o $@ $(APP\_OBJS) $(LDLIBS)

改为

$(APP): $(APP\_OBJS)

$(CC) $(LDFLAGS) -lpthread -o $@ $(APP\_OBJS) $(LDLIBS)

也就是添加一个-lpthread编译连接选项，这是由于mat\_mul\_demo.c程序中使用到了多线程操作，需要连接相应的库文件。

1. 由于在该例子中remote core的运用程序需要通过master core加载到适当的内存中去，再交由remote core去执行，所以需要预先将remote core的可执行程序安装在Linux的根文件系统下。这里有两种实现方式，其原理是一样的
2. 然后将matrix\_multiply.elf文件（利用XSDK生成的可执行文件，创建过程见Creating and Building Remote application in XSDK一节）放在

<proj\_root>/components/apps/matrix\_multiply/data目录下，更改名字为matrix\_multiply

同时修改文件<proj\_root>/components/apps/matrix\_multiply/Makefile，修改成如下形式

include $(PETALINUX)/components/apps/apps.common.mk

FIRMWARE=matrix\_multiply

install:

$(TARGETINST) -d data/$(FIRMWARE) /lib/firmware/$(FIRMWARE)

**注意**：$(TARGETINST)前必须是一个tab符，不能是空格

1. 

然后在<proj\_root>/components/apps/matrix\_multiply/目录下创建文件夹data，并将matrix\_multiply.elf文件（利用XSDK生成的可执行文件，创建过程见Creating and Building Remote application in XSDK一节）放到data/下，更改名字为matrix\_multiply。

然后删除<proj\_root>/components/apps/matrix\_multiply/目录下的Makefile文件,并重新建立一个Makefile内容如下：

include $(PETALINUX)/components/apps/apps.common.mk

FIRMWARE=matrix\_multiply

install:

$(TARGETINST) -d data/$(FIRMWARE) /lib/firmware/$(FIRMWARE)

**注意**：$(TARGETINST)前必须是一个tab符，不能是空格

以上代码的意思是将可执行文件matrix\_multiply安装在Linux根文件系统/lib/firmware/目录下

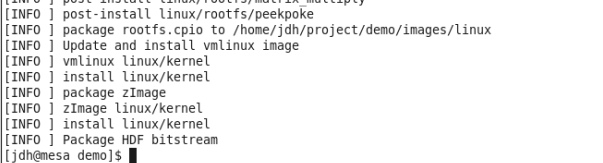
---------------------------------------------------------------------

**Note**：matrix\_multiply.elf的名字也可以不改，如果这样Makefile里的FIRMWARE要相应的修改为FIRMWARE=matrix\_multipl.elf

---------------------------------------------------------------------

1. 编译工程



如果编译过程中出错可以执行：



查看详细的编译过程，并从中查看错误信息。

当执行最后看到如下信息就说明build成功了，当然如果结果最后仅提示有关于tftpboot的这样一个错误的话，可以暂时忽略。

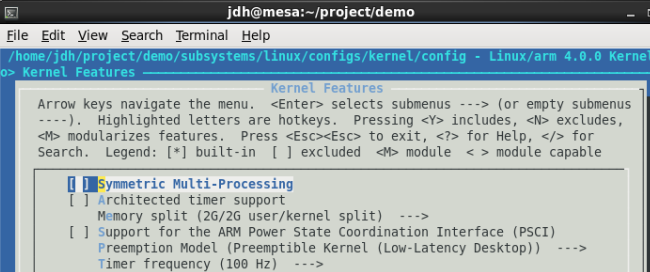
## Using RPMSG Framework Without Remoteproc

PetaLinux工程的创建过程与上一节类似，具体步骤如下：

1. 同上一节step 1~4
2. 取消SMP模式，执行：

Kernel Features --->

[ ]Symmetric Multi-Processing

 保存并退出。

1. 添加设备节点（也即编辑设备树文件）该步骤需要编辑system-top.dts文件(位于subsystems/linux/configs/device-tree/）,这一步是为了给驱动模块添加设备节点，以便驱动能够被probe（可参考Linux驱动开发相应资料）。暂且让System-top.dts包含以下内容：

/dts-v1/;

/include/ "system-conf.dtsi"

/ { amba{

remoteproc0:remoteproc@0{

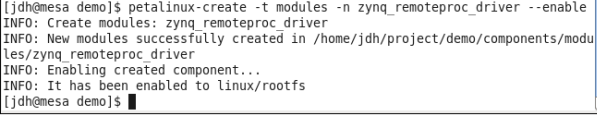
compatible = "xlnx,rpmsg\_virtio\_driver";

};

};

};

1. 创建内核模块（module）执行指令：





然后将源文件virtio\_device\_driver.c文件(该文件存在于该文档同等目录src/demo\_without\_using\_remoteproc文件夹下)复制到

<proj\_root>/components/modules/virtio\_device\_driver/

将rpmsg\_user\_dev\_driver.c文件(该文件存在于该文档同等目录src文件夹下)复制到

<proj\_root>/components/modules/rpmsg\_user\_dev\_driver/

1. 创建Linux用户空间程序



然后将mat\_mul\_demo.c文件(该文件存在于该文档同等目录src/demo\_without\_

using\_remoteproc文件夹下)复制到

<proj\_root>/components/apps/mat\_mul\_demo/

之后需要对该目录下的Makefile文件做一定的修改，修改如下：

将原本的

$(APP): $(APP\_OBJS)

$(CC) $(LDFLAGS) -o $@ $(APP\_OBJS) $(LDLIBS)

改为

$(APP): $(APP\_OBJS)

$(CC) $(LDFLAGS) -lpthread -o $@ $(APP\_OBJS) $(LDLIBS)

也就是添加一个-lpthread编译连接选项，这是由于mat\_mul\_demo.c程序中使用到了多线程操作，需要连接相应的库文件。

1. 将该文档同等目录下，src/patch/irq-gic.patch文件夹复制到$PETALINUX/components/linux-kernel/xlnx-4.0/drivers/irqchip/目录下，路径切换到

$PETALINUX/components/linux-kernel/xlnx-4.0/drivers/irqchip/

然后执行以下语句(**执行之前建议把petalinux备份一下，若使用的是实验室服务器上的petalinux，那么必须备份，备份方式为将整个petalinux安装文件夹拷贝到合适的目录下，比如/home/jdh/,然后对备份进行打补丁，source备份中的setting.sh或setting.csh文件**)：



为了使得系统支持非SMP模式下的中断操作，需要相应地修改内核中gic模块的源码，上面的指令是在给相应源码打上所需补丁。如需要撤销补丁只需在该目录下执行以下指令即可：



1. 将该文档同等目录下，src/patch/arm-gic.patch文件夹复制到$PETALINUX/components/linux-kernel/xlnx-4.0/include/linux/irqchip目录下，路径切换到

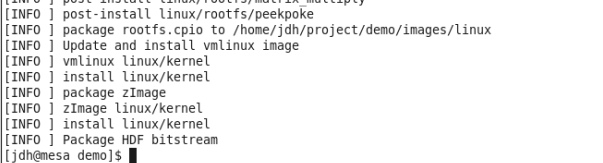
$PETALINUX/components/linux-kernel/xlnx-4.0/include/linux/irqchip/

然后执行以下语句：



1. 编译工程,路径切换回<proj\_root>,然后执行



如果编译过程中出错可以执行：

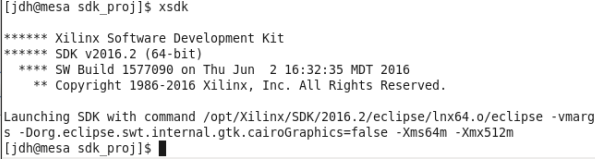


查看详细的编译过程，并从中查看错误信息。

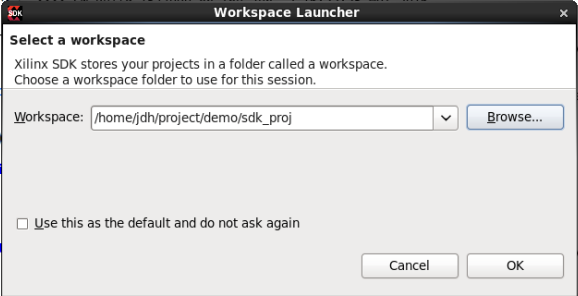
当执行最后看到如下信息就说明build成功了，当然如果结果最后仅提示有关于tftpboot的这样一个错误的话，可以暂时忽略。

# Creating and Building Remote application in XSDK

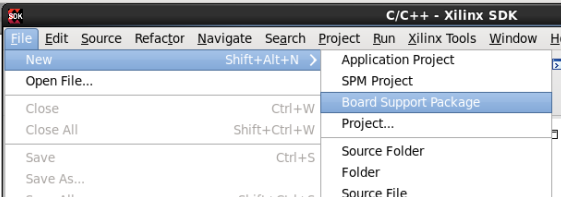
1. 启动xsdk

Linux环境下通过命令行方式启动如下：



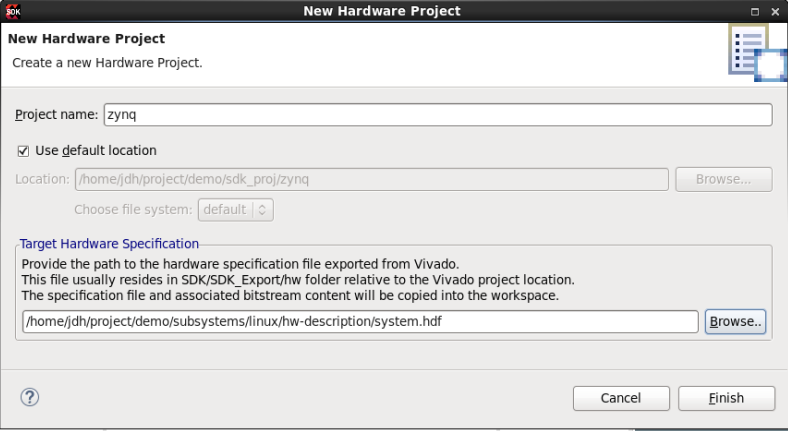
1. 指定工程目录

点击确定。

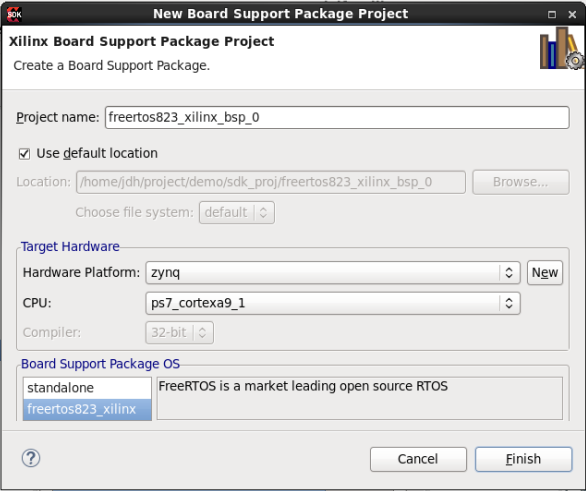
1. 首先需要创建一个新的板级支持包（borad support package(BSP)），在XSDK窗口中选择**File > New > Board Support Package**

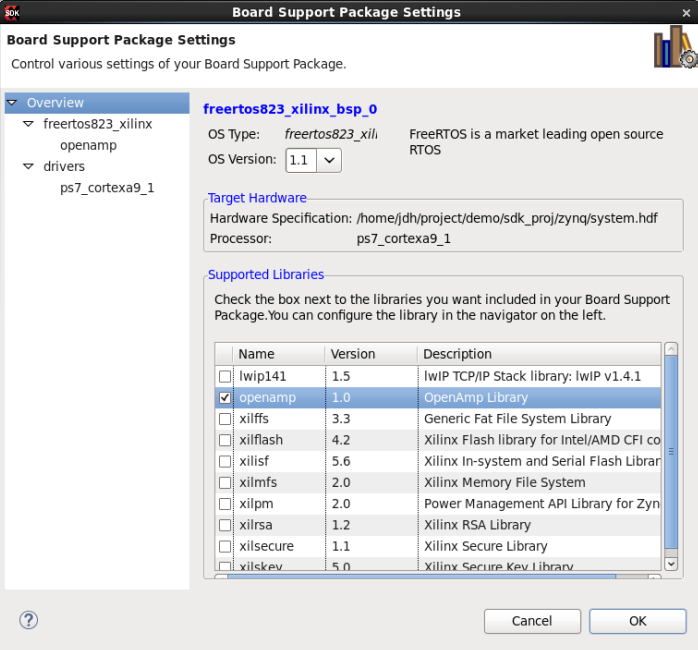
系统提示No Hardware Platform in the workspace,因为我们还没有为工程指定相应的硬件平台，这里需要添加一个.hdf文件，该文件可以从所创建的petalinux工程中获取，具体路径为<proj\_root>/subsystems/linux/hw-description/system.hdf

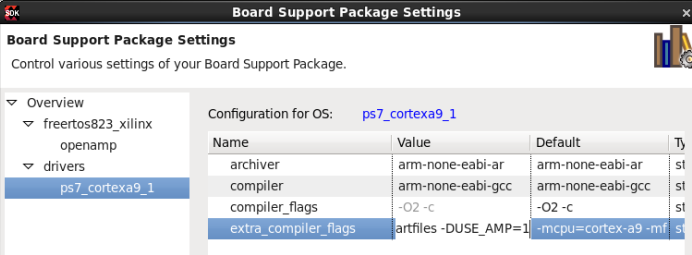
点击specify,并添加system.hdf文件。（所使用的petalinux版本与vivado版本应尽量保持一致，否则可能会出现错误）。



点击finish，紧接着就可以创建BSP

指定CPU：ps7\_cortexa9\_1,Board Support Package OS：freertos823\_xilinx。点击finish。

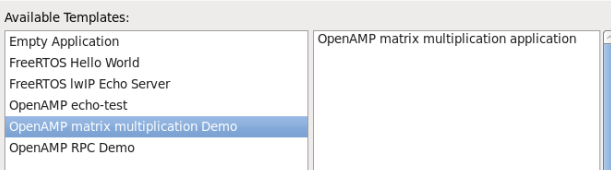
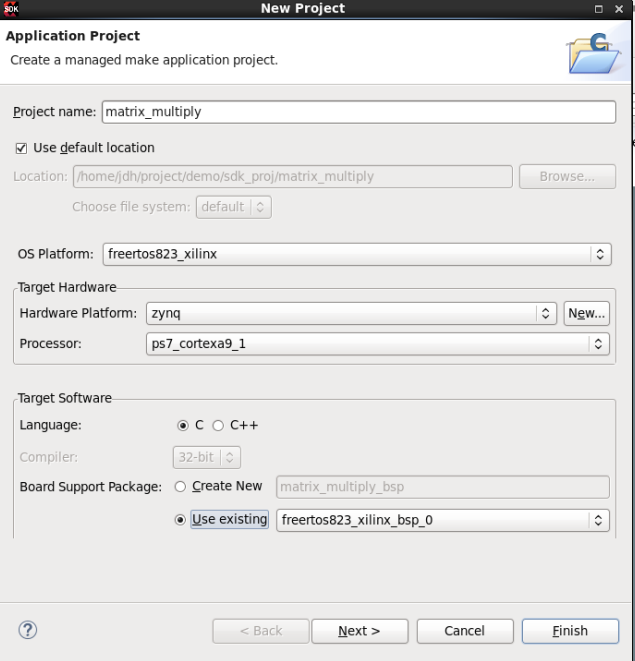
勾选Supported Libraries 中的openamp，然后如下图，在extra\_complier\_flags一栏的value后加上-DUSE\_AMP=1

点击OK

1. 创建运用程序

在XSDK窗口中选择**File>New > Application projects**

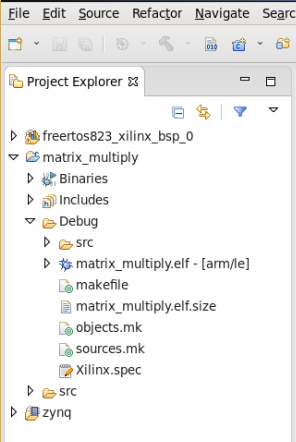
如下图设置OS platform，Hardware Platform，processor以及BSP，

然后点击next ，并选择OpenAMP matrix multiplication Demo,点击finish完成运用程序创建

此时在XSDK窗口左侧可以看到

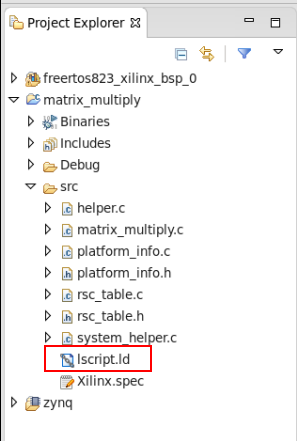
最终我们需要得到的ELF格式可执行文件在matrix\_multiply/Debug/

matrix\_multiply/matrix\_multiply.elf

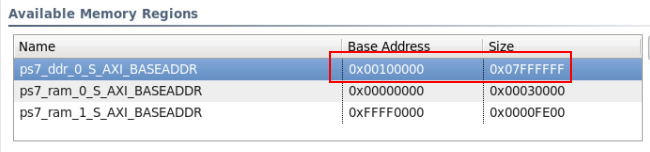


1. 如果是master core和remote core完全互相独立，即master core不使用

remoteproc组件负责remote core的启动的情形下，需要对以上创建的matrix\_multiply工程做以下修改：

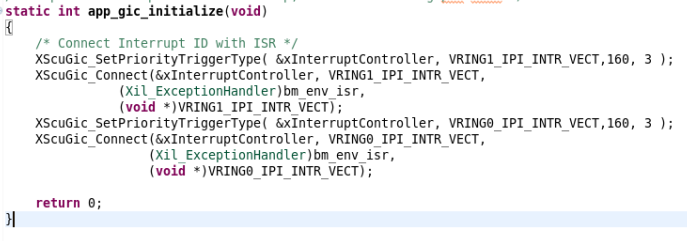
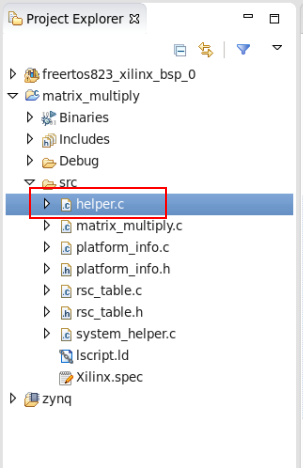
1. 修改FreeRTOS占用的内存区域。打开图中所示文件

修改内容如下图所示：

这么做是为了在master core不负责启动remote core，且不为其运用程序分配运行空间的情况下，Linux和FreeRTOS所占用空间不发生冲突而导致错误。

1. 修改gic的初始化程序。

打开文件helper.c，修改函数app\_gic\_initialize(void)为以下形式：



测试步骤请移步Executing Tests for Linux Master/FreeRTOS Remote Comfiguration一节(without remoteproc)

# Boot the PetaLinux Project

通常可以在QEMU和实际硬件平台式boot petalinux，详细信息可以参考Documents文件夹里的ug1144-petalinux-tools-reference-guide文档。

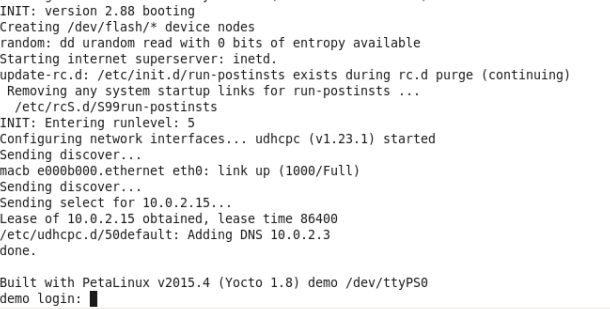
## Boot a PetaLinux Image on QEMU

这种方法可以在不借助实际硬件的情况对工程进行调试，这是利用QEMU这样一个仿真软件实现。Petalinux提供了QEMU支持，使得用户可以在仿真环境下对Petalinux软件镜像进行测试。具体步骤如下：

1. 首先需要确保petalinux-build成功执行，并且在工程目录<proj\_root>/images/linux/下生成了image.ub文件，这是Linux的镜像文件。
2. 在工程根目录下（<proj\_root>/）,执行以下指令：



在控制台窗口应该可以看到以下启动信息

以login:root password:root进行登录

**Tips**:通过先组合按下Ctrl+A,然后松开，再按下X键退出QEMU

实验证明，通过QEMU不一定能够正确验证程序的正确性，有时QEMU仿真时时错误的，但在硬件上运行是正确的。

## Boot a PetaLinux Image on Hardware

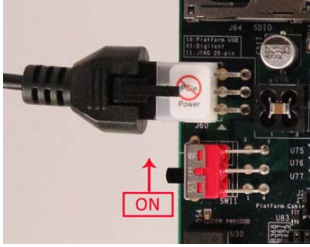
首先需要将开发板与开发主机相连，并配置好相应的环境（安装相关驱动等）

**步骤如下**：

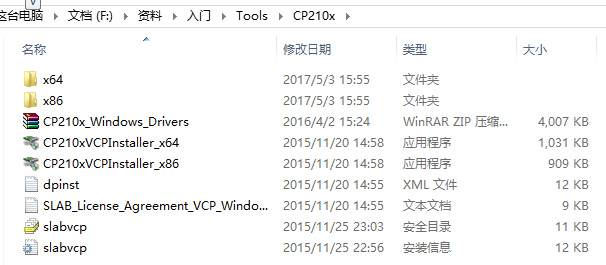
1. 通过一条USB mini-B线将开发板上的串口（USB UART Connector J17）连接到电脑上。如图：

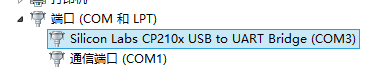


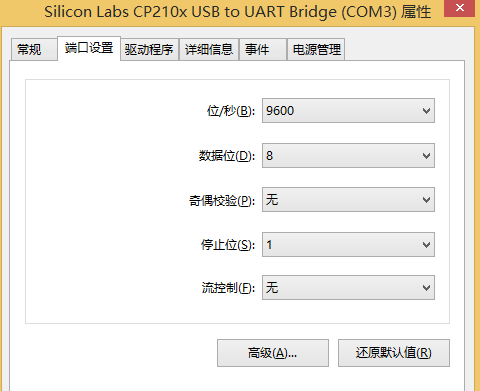
最后连接电源线。

1. 在电脑上安装 USB Uart驱动

获取地址为http://www.silabs.com ，在本文档所在目录下tools文件夹中放有安装包。

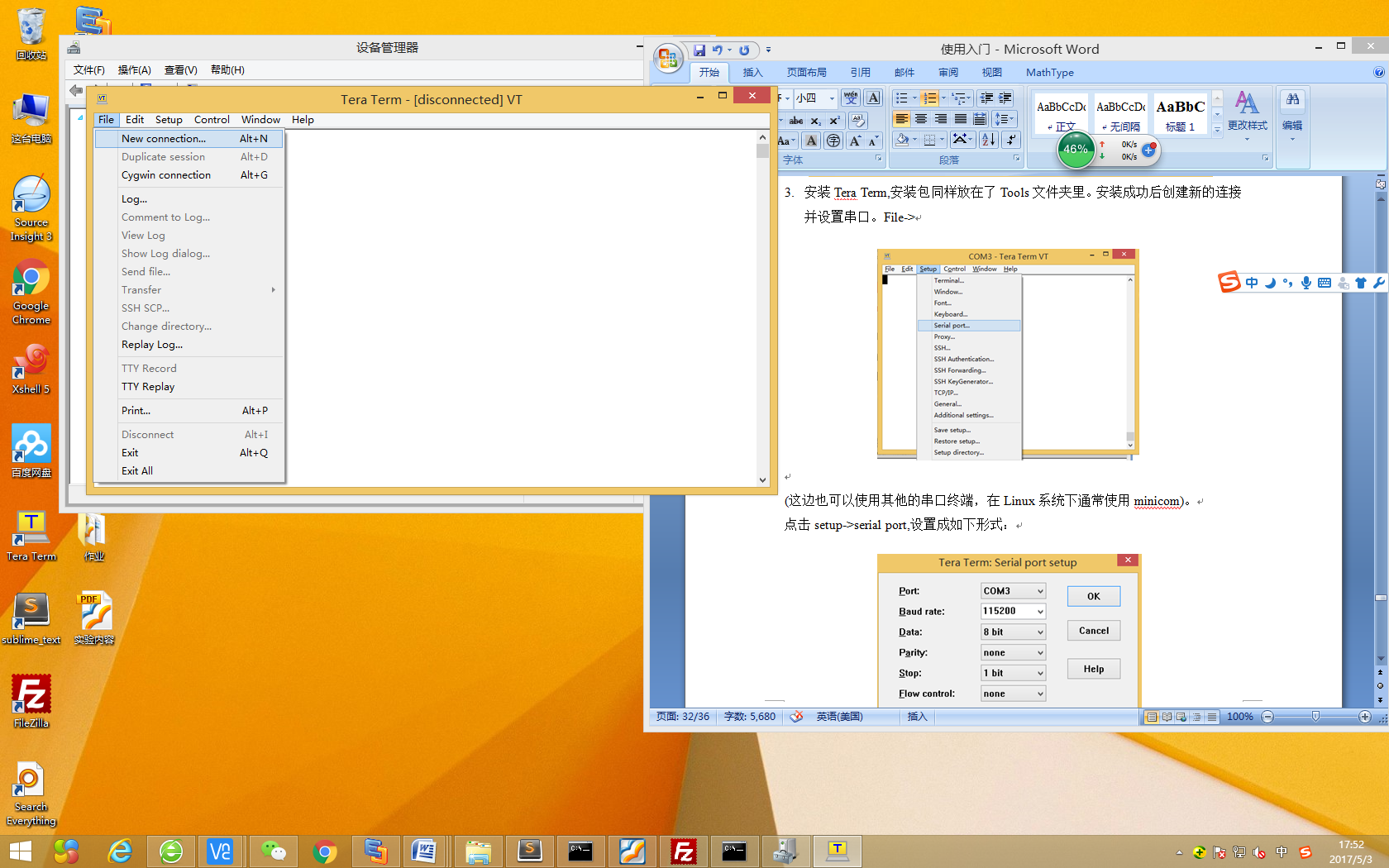
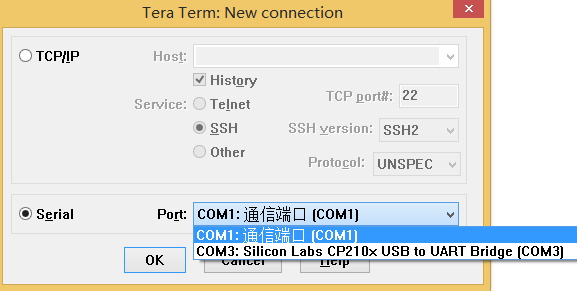
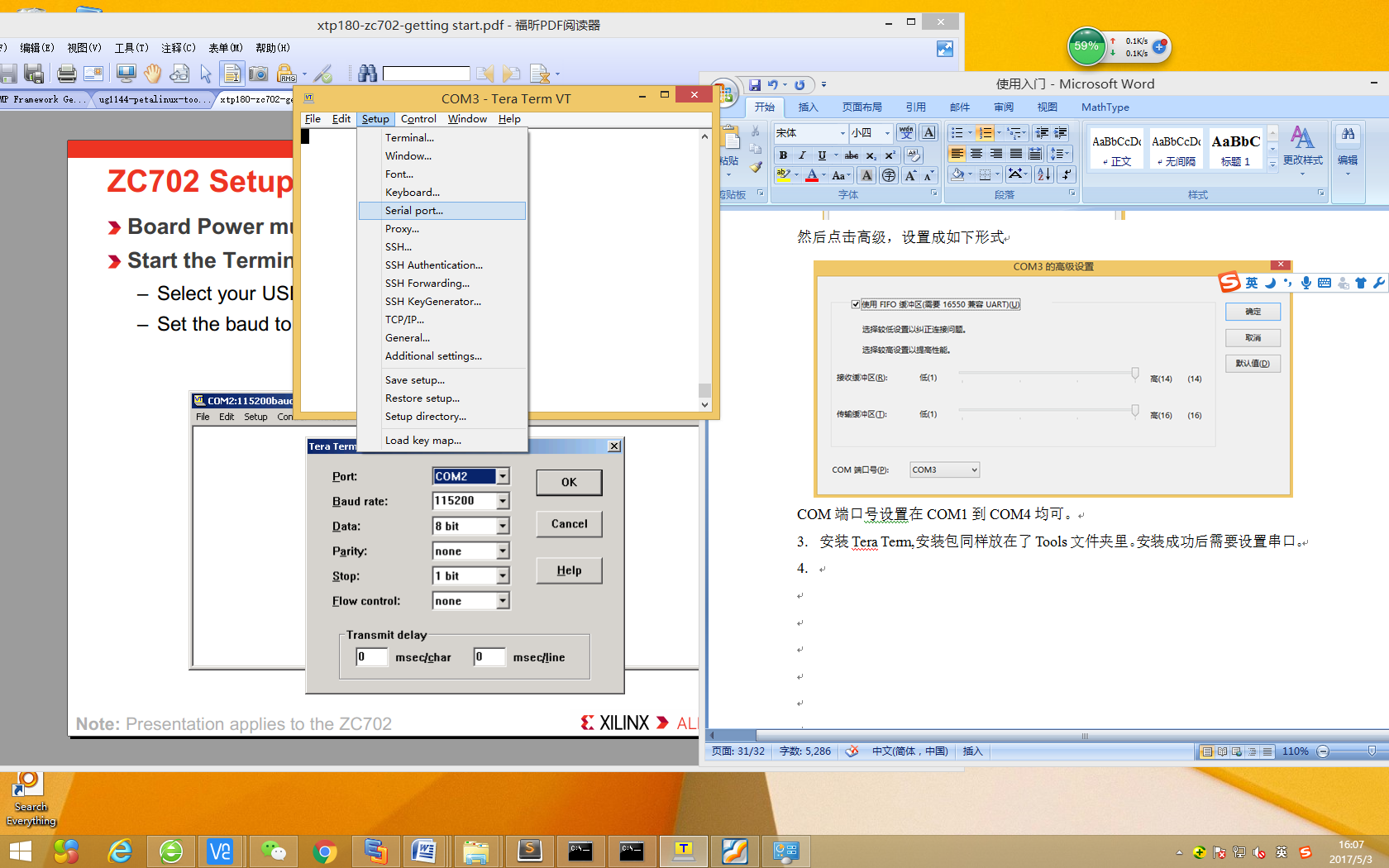
根据系统位数选择相应的安装程序。安装完成后打开电脑上的设备管理器，在端口处可以看到Silicon Labs CP210X USB to UART Bridge一项，右键打开属性。

端口设置如下图：

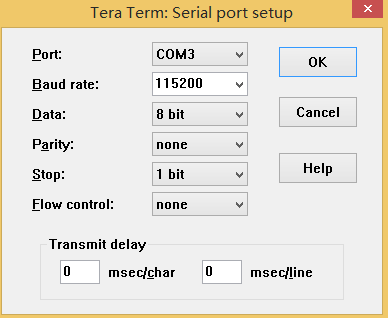


然后点击高级，设置成如下形式，COM端口号设置在COM1到COM4均可。

1. 安装Tera Term,安装包同样放在了Tools文件夹里。安装成功后创建新的连接并设置串口。



(这边也可以使用其他的串口终端，在Linux系统下通常使用minicom)。

点击setup->serial port,设置成如下形式：

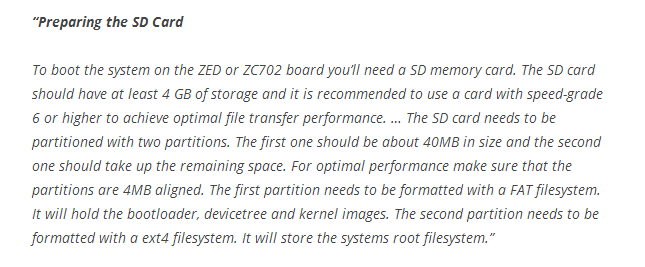
Port项与第二步中设置的保持一致。

**Note**：如果板子上的开关和跳接帽被改动过，请参考**ug850-zc702-eval-bd**文档关于Default Switch and Jumper Settings部分说明，获取开关和跳帽的默认状态。

### Boot a PetaLinux Image on Hardware with SD Card

**步骤如下：**

1. 准备一张SD卡(需要外加一个SD适配器)，并将其格式化为FAT32格式（以下实验用的是Kingston 8G class 4的SD卡）。根据一些资料显示建议使容量大于等于4G的SD卡，读写速率越快越好。

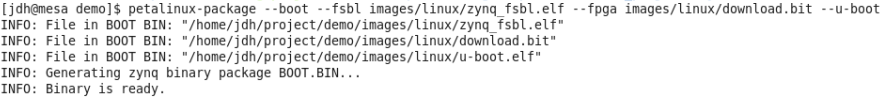
该demo中只给SD制作一个FAT格式的分区就行。

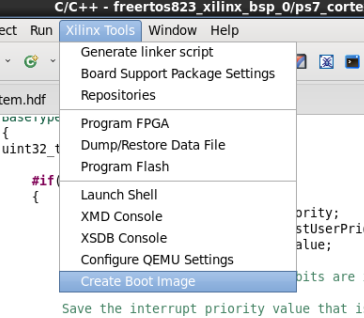
1. 将image.ub(<proj\_root/image/linux/image.ub>)和BOOT.BIN加到制作好的SD卡中。image.ub在petalinux-build成功后可获得，BOOT.BIN制作方式：
2. 在目录<proj\_root>/下执行：

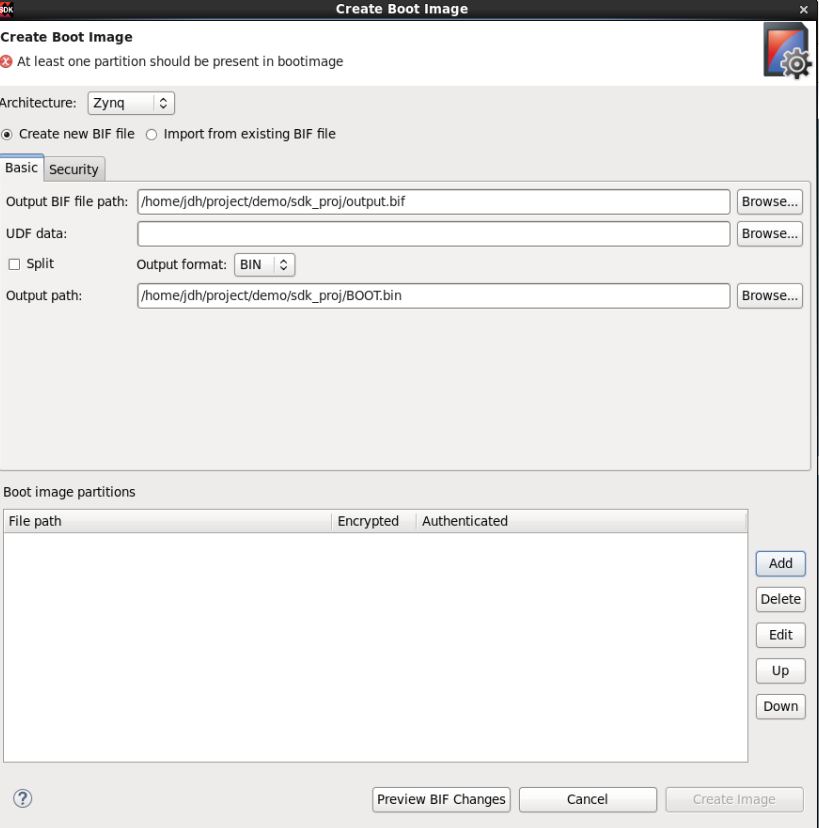


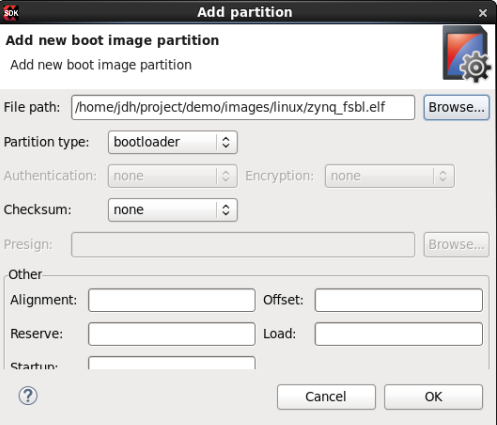


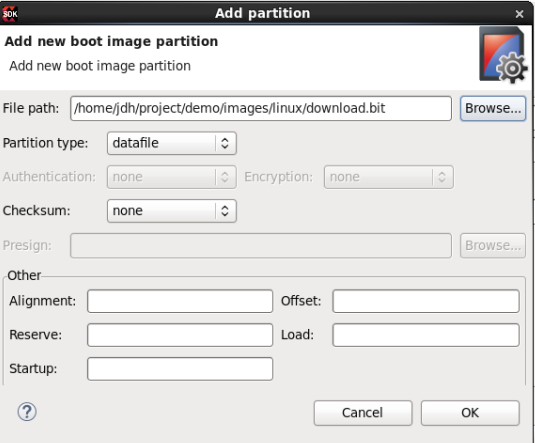
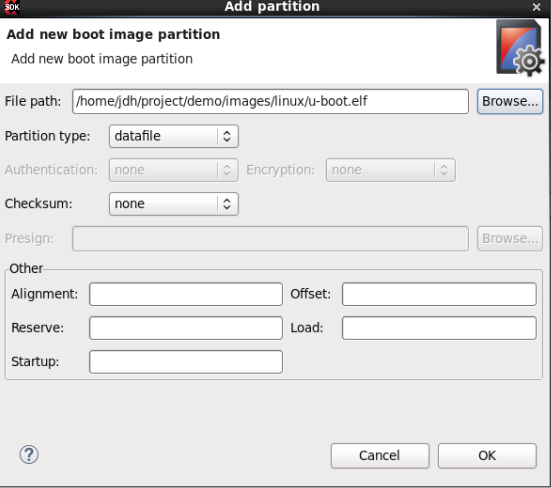
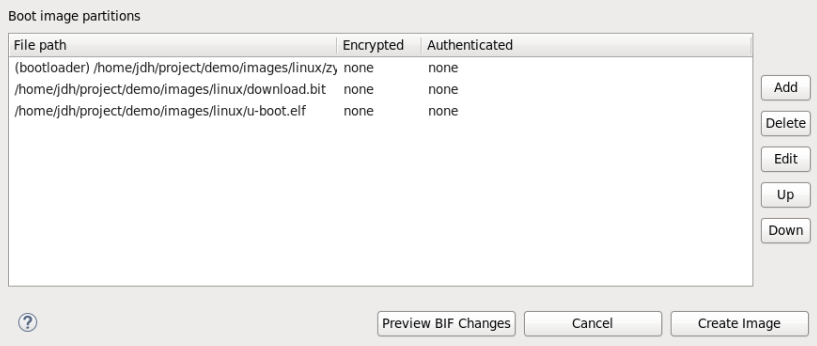
执行结果如下：

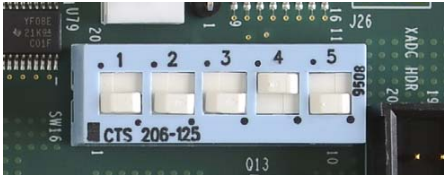
生成的BOOT.BIN在<proj\_root>和images/linux/目录下均存在。

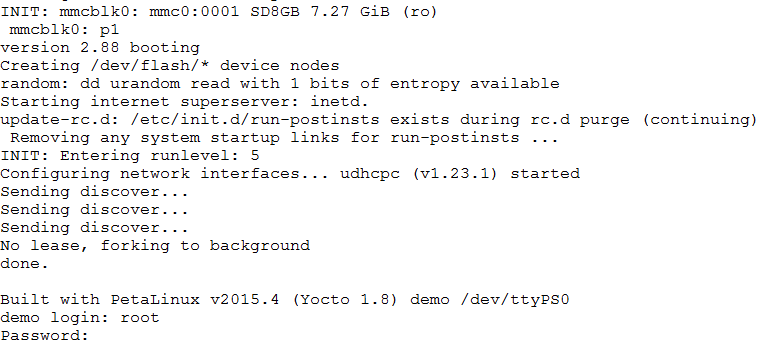
1. 通过XSDK制作BOOT.BIN，XSDK窗口点击Xilinx Tools--->Create Boot Image

选择Architecture:zynq,指定Output BIF file path

然后在BOOT image partitions中依次将zynq\_fsbl.elf、download.bit和u-boot.elf文件添加进去，**要注意添加顺序**。最后点击Create Image就可以了。

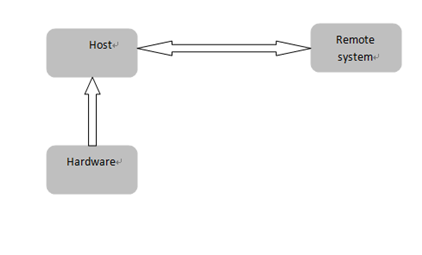


1. 把包含image.ub和BOOT.BIN(或者BOOT.bin)的SD卡插到板子上的卡槽中（J64 slot）
2. 将SW16配置成**00110**，即将下图中的**3,4往上拨（图中所示模式是不对的，需要更改）**。这是为了让板子选择从SD卡启动。
3. 打开Tera Term并设置好串口（详情见Boot a PetaLinux Image on Hardware部分）
4. 给板子上电，等待系统启动。

系统启动之后可以在窗口调试窗口看到如下启动信息（启动信息太长只截取部分）

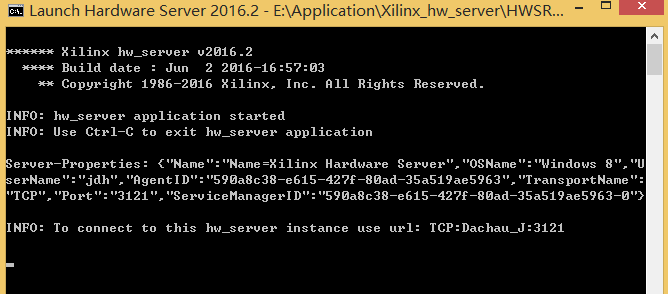
以login:root password:root登录

### Boot a PetaLinux Image on Hardware with JTAG

详情请参考ug1144-petalinux-tools-reference-guide文档。这边讲解以下情形下的Boot方式。（非该情景，详情请直接参考ug1144文档，此处略）。

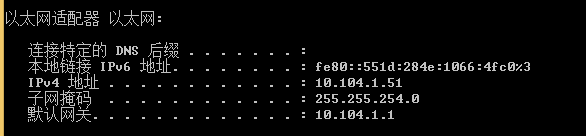
假设Host的系统为Windows，Remote system为运载在远程服务器上的Linux系统。Hardware为zc702开发板。因为通常是通过远程登录到服务器上进行工作，而开发板是连接到本地PC机上，同时为了解决在测试时频繁拔插SD的繁琐和低效率，所以可采用以下方法。

**具体实现步骤如下**：

1. 在Host端运行hw\_server.bat(vivado开发套件提供了这个文件) 打开DOS窗口---->cmd----->输入hw\_server（若找不到指令，请在path路径中添加hw\_server的存放路径,hw\_server程序存在于Vivado或SDK安装目录下的bin文件夹中）

**或**直接点击运行

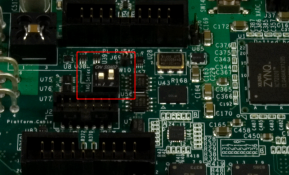
**注意**：运行hw\_server后应保持窗口开启，不要关闭。

1. 另外利用ipconfig（假设本地PC上是Windows系统，若是Linux请执行ifconfig）指令查看Host的IP，假设为10.104.x.xxx
2. 在Remote端，编译petalinux project，之后使用petalinux-package --prebuilt将petalinux image.ub镜像、u-boot.elf,BOOT.bin,system.dtb等文件放在生成的pre-built目录下，然后将download.bit文件拷贝pre-built/linux/implementation/目录下。

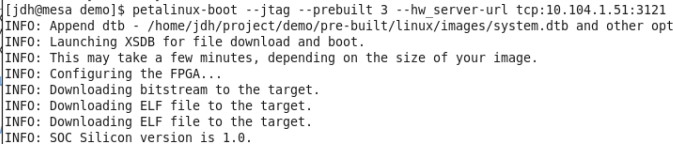
**Note：**如果工程修改过，需要重新编译，然后执行



进行强制更新。

1. 将开发板上的SW10设置成如下图形式，SW10=01；设置为JTAG mode
2. 使用USB Type-A to Micro-B线连接开发板JTAG接口和本地PC机。
3. 通过一条USB mini-B线连接开发板（USB UART Connector J17）和本地PC机。
4. 在服务器上，工程根目录（<proj\_root>）下执行

petalinux-boot --prebuilt 3 --jtag --hw\_server-url TCP:10.104.x.xxx:3121即可。

成功后有以下信息

同时可在串口调试窗口看到系统的启动信息。系统启动成功后同样以login:root password:root进行登录。

**Tips：**更多关于这一部分的内容可以参考以下文档：

<https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ug873_zynq_ctt.pdf>

<https://www.xilinx.com/support/documentation/boards_and_kits/UG926_Z7_ZC702_Eval_Kit.pdf>

<https://www.xilinx.com/support/documentation/boards_and_kits/zc702_zvik/ug850-zc702-eval-bd.pdf>

相关文档也可在该文档所在目录下Documents文件夹下获得

# Login Linux with SSH

PetaLinux提供了TCF(Target Communication Framework)Agent来协助zynq用户空间程序的开发。利用这一点，我们便可以在不使用串口的情况下来实现调试。我们在使用SSH来远程连接设备这本身是很简单的，但是如果我们不知道设备的IP那就比较麻烦了，在Linux中我们可以通过ifconfig指令查看设备IP，或者利用该指令指定IP，但是考虑一种情况，我们已经让Linux在开发板上跑起来了，可现在开发板上的串口坏了或者被占用了，我们看不到系统输出的信息，也没能输入任何指令，更不用说输入ifconfig指令来查看了。所以这里为了解决这种问题可以采用下面的方法。

**首先需要使能TCF支持。**

步骤：

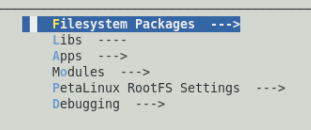
1. 切换到petalinux工程目录下：

$ cd <plnx-proj-root>

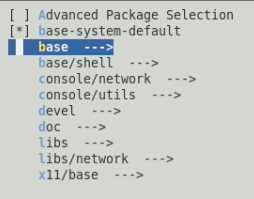
1. 配置文件系统，执行：

$ petalinux-config -c rootfs

执行后跳出窗口如下图



1. 选择Filesystem Packages



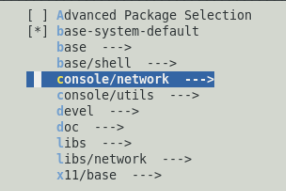
1. 选择base

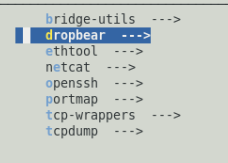


1. 选择tcf-agent,并如下图将其勾选上。

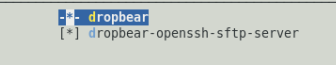


1. 然后返回base选项所在目录，选择console/network



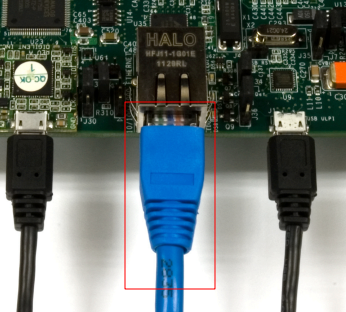


1. 选择dropbear
2. 将dropbear-openssh-sftp-server勾选上



完成以上几步之后先编译一次

$ petalinux-build

1. 给开发板接上网线。

现在我们已经使能了系统的ssh server功能，要通过ssh连接还需要知道对应的ip地址，由于不知道系统启动后被分配的IP是多少，所以我们索性给它设置一个。

**所以然后设置IP**

***方法肯定不止一种***。

将目录切换到<plnx-proj-root>/build/linux/rootfs/targetroot/etc/rc5.d

***插一句***：这里使用的petalinux的是v2015.4版本，不同版本之间的目录结构可能不一样，但道理是一样的。

在该目录下创建脚本文件，比如叫做myfoot.sh

编辑文件内容如下

#! /bin/sh

echo "setting eth0"

ifconfig eth0 10.104.1.55 ##这里IP地址自己设定，设定和自己的pc机网络在一个网段就行。如果本机和开发板同时连接到同一个路由器上，将板子通过网线连接到路由器上，设置成同网段IP就行，比如本地IP为10.104.1.51，那么可以将eth0 ip配置为10.104.1.xxx,只要该IP未被占用；如果没有路由器，可以将板子和本地PC机直接通过网线相连，拿window下来举例，打开网络和共享中心🡪更改适配器设置🡪右键以太网（或本地连接），选择属性🡪双击TCP/IPv4选项🡪设置IP地址和掩码🡪确定保存即可，然后设置eth0的IP使其与本地PC机的IPV4地址在同一网段就可以了。

保存后，将该文件的权限改为777

chmod 777 myfoot.sh

在同样目录下，执行以下指令：

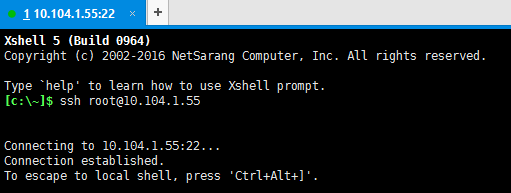
ln -s myfoot.sh S100myfoot.sh

创建一个软连接，S表示开机执行，100是执行顺序

**最后，编译**

$ petalinux-build

将系统的启动镜像拷贝到SD卡中，设置板子从SD卡启动(或者采用JTAG、TFTP等方式启动)，一段时间后就可以利用xshell，putty等软件连接了。



# Executing Tests for Linux Master/FreeRTOS Remote Comfiguration

该测试部分统一通过一个矩阵相乘的例子来进行，预期过程应该是在master和remote之间建立起核间通信通道后，master随机产生两个6x6的矩阵，并通过通信通道发送给remote端，remote端在收到数据后将两个矩阵进行相乘，最后把运算结果发送还给master。

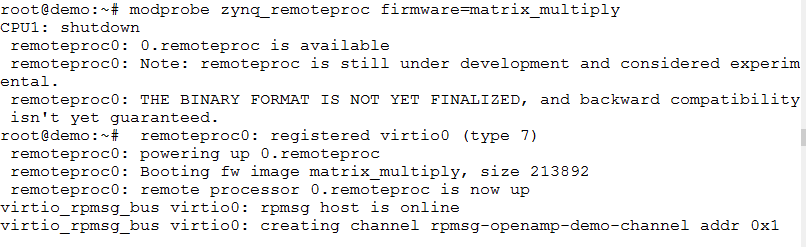
## Tests Using RPMSG Framework with Remoteproc

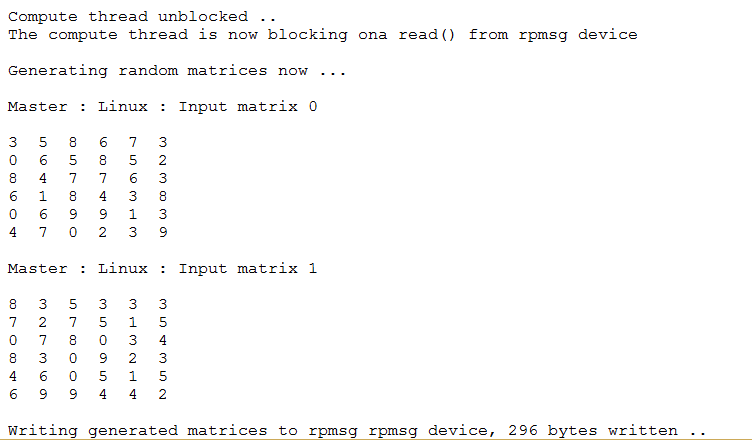
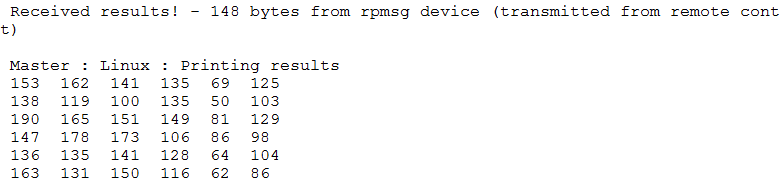
在进行测试之前，应按照前面所说的，完成工程的建立和编译，并把最终在硬件平台上启动整个系统（以login：root password：root登录）

**注意：**目前如果在QEMU环境下进行一下错误会有问题，建议在硬件平台上进行测试。

**测试步骤如下**：

Linux启动后，登录系统，然后依次输入以下命令，运行结果紧接在指令后面：

1. 
2. 
3. 
4. 输入1，结果如下：

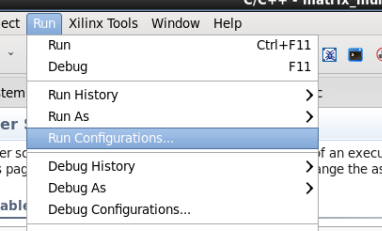
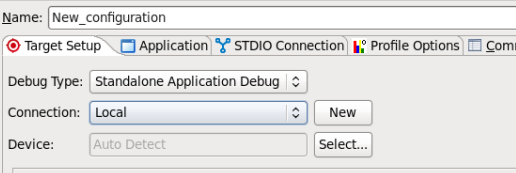


## Tests Using RPMSG Framework without Remoteproc

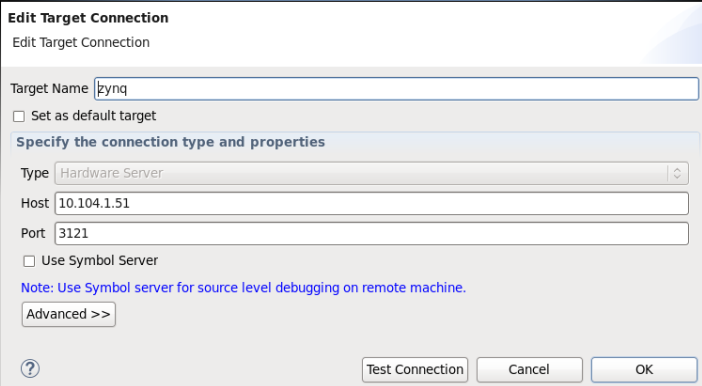
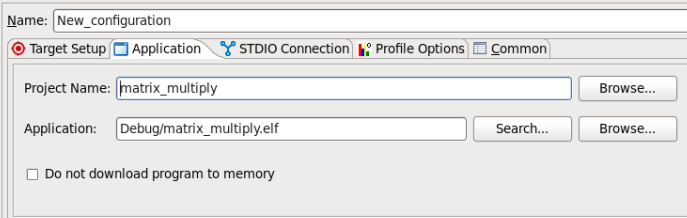
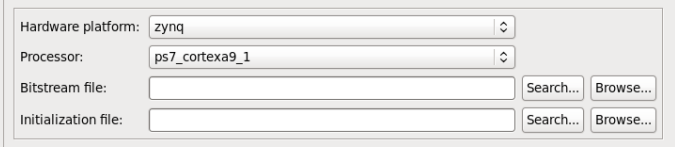
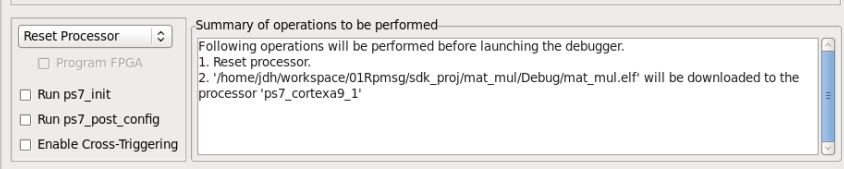
在这种情景下master端和remote端需要分别进行操作。Master端PetaLinux镜像的启动过程如Boot the PetaLinux Project一节所述（不使用QEMU方式）。等到Linux系统启动成功后以login：root password：root登录。

然后在XSDK上进行以下操作：

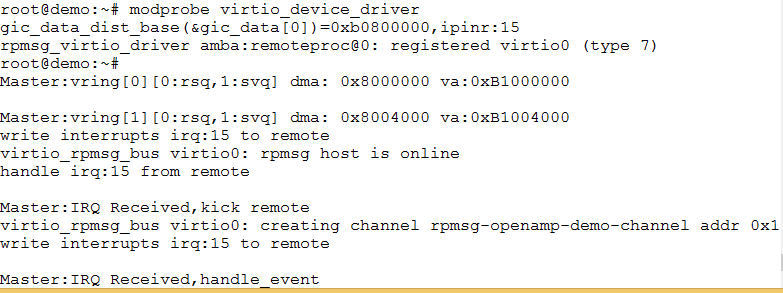
1. 如图在XSDK窗口点击Run🡪Run Configuration

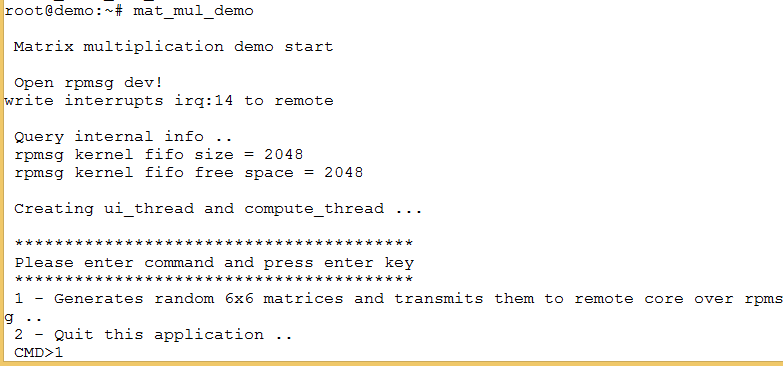
打开如图界面(这里采用的是GDB调试工具)

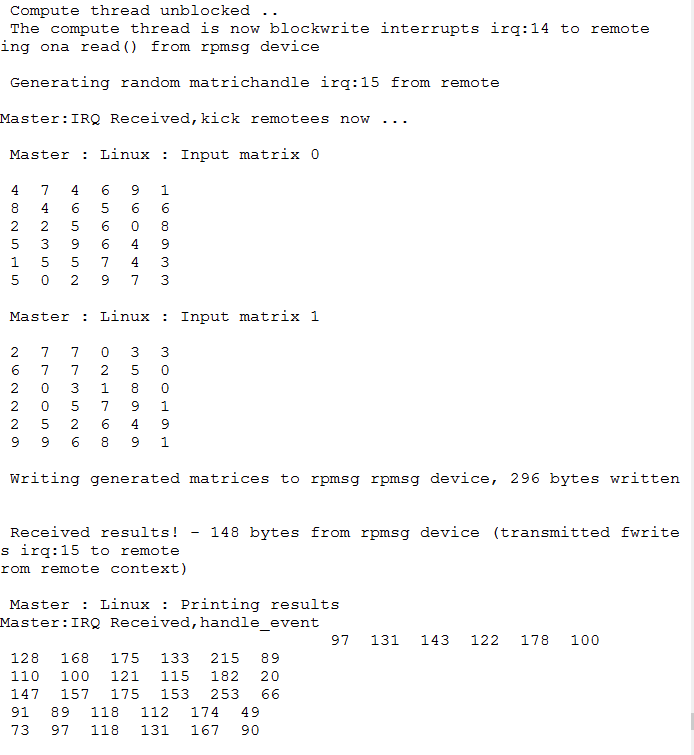
如果zc702evk硬件平台直接连接到本地PC机上，而XSDK是运行在远程服务器上，则点击connection一项的New按钮，添加连接，在此之前应确保完成本文档中Boot a PetaLinux Image on Hardware with JTAG一节提及的设置，图中Host一项是本地PC机的IP；否则如果板子连接和使用的XSDK均在本地PC机上，则保持Local即可。

1. 添加需要运行的程序。
2. Processor选择ps7\_cortexa9\_1
3. 只选择Reset processor就好
4. 点击apply 和Run
5. 在Linux部分，依次执行以下指令(应在前4步完成后再进行这一步操作)：







输入1，结果如下：