Name: _____ ID: _____ Start Date: Thursday, June 13, 2019
Due Date: Tuesday, June 18, 2019

# Software and Hardware Co-Design with Zybo, Spring 2019 HUST
# Lab #11 Embedded Linux Kernel on Zybo

This is an individual lab. Each student must perform it and demonstrate this lab to obtain credit for it. Late lab submission will be accepted with a grade reduction of 20% for each day that it is late. This lab is due by Tuesday, June 18, 2019.

The main ideas of this lab are from "Embedded Linux® Hands-on Tutorial for the ZYBO™",

Revised July 17, 2014, Digilent Inc. and the Zybo Base System Design from Digilent Inc.

## 1 Objectives

1) Create Vivado 2015.2 version of Zybo Base System Design Block Design and bit stream file with Windows-based Vivado version 2015.2. (Section 1 of "Embedded Linux® Hands-on Tutorial")
2) Compile a u-boot.elf file with Xilinx Linux arm compiler on VirtualBox Linux environment. (Section 2 of "Embedded Linux® Hands-on Tutorial")
3) Create a BOOT.BIN file Windows-based Vivado version 2015.4 to include zynq FSBL.elf file, system_wrapper.bit file and u-boot.elf file. (Section 3 of "Embedded Linux® Hands-on Tutorial")
4) Compile a Linux kernel image, uImage, on VirtualBox Linux environment. (Section 4 of "Embedded Linux® Hands-on Tutorial")
5) Compile a disk system in RAM, uramdisk.image.gz, on VirtualBox Linux environment. (Section 5 of "Embedded Linux® Hands-on Tutorial")
6) Compile a device tree blot file, devicetree.dtb file on VirtualBox Linux environment. (Section 5 of "Embedded Linux® Hands-on Tutorial")
7) Create a Linux bootloader on micro SD card with four files: BOOT.bin, devicetree.dtb, uImage, urandisk.image.gz. Boot Linux from micro SD card on Zybo. (Section 5 of "Embedded Linux® Hands-on Tutorial")
8) Modify the device tree, write myled.c driver and the LED application with Linux on Zybo. (Sections 6 and 7 of "Embedded Linux® Hands-on Tutorial")
9) Boot Linux with the led application from micros on Zybo with six files: BOOT.bin, devicetree.dtb, uImage, urandisk.image.gz, led_blink.bin, and myled.ko.

## 2 Source code

zybo_base_system.zip, microSD Example Boot Files, Device driver myled.c and led application code led_blink.c are available from the instructor. Linux-Digilent-Dev-master.zip and u-boot-Digilent-Dev-master.zip are available from github. Xilinx.zip is also available from the instructor.

## 3 Demonstration and Report

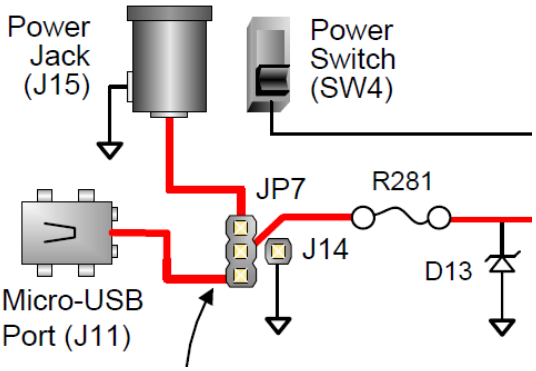
Demonstrate your Linux boot loader and led application by the deadline to get credit for this lab.

# 4 Resources

1) "Embedded Linux® Hands-on Tutorial for the ZYBO™", Revised July 17, 2014, Digilent Inc.
2) zybo_base_system.zip, Zybo Base System Design, https://reference.digilentinc.com/zybo:zybo.
3) *Guidelines on VirtualBox Ubuntu and Xilinx SDK rev4-13-2019, by David Robinson and JianJian Song.*
4) Lab 4 ZYNQ4 Writing Basic Software Application, http://www.xilinx.com/support/university/vivado/vivado-workshops/Vivado-adv-embedded-design-zynq.html.
5) Lab 5 Configuration and Booting, Advanced Embedded System Design on Zynq using Vivado, http://www.xilinx.com/support/university/vivado/vivado-workshops/Vivado-adv-embedded-design-zynq.html.
6) https://github.com/DigilentInc. Linux-Digilent-Dev.git and u-boot-Digilent-Dev.git archives.
7) http://www.wiki.xilinx.com. arm_ramdisk.image.gz in Build and Modify a Rootfs.

# 5 Learn how to Boot Linux from a micro SD Card

This is an exercise for you to learn steps to boot Linux from a micro SD card with example boot loader files.




## 5.1 Format your microSD card

If you need to remove the existing format on your microSD card, you can run Disk Management on your laptop. Delete all formats and make a New Simple Volume on the SD card with all default settings.

## 5.2 Copy Boot Loader files and Test SD card

Copy boot loaders and boot image files from Moodle->Week 8-> micro SD Boot Files folder to your micro SD card in FAT32 format. Insert the micro SD card into the SD slot on your Zybo.

Set your Zybo board to get power from wall plug power supply by connecting power jumper between Wall and VV5V0.

Start a UART terminal with 115200 baud, 8-bit data, no parity, and no flow control. Set comm port to be the comm port of your Zybo board.

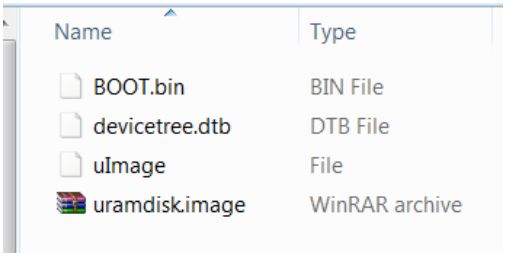
Following these instructions from Zybo Reference Manual from Digilent to Boot Linux from your micro SD card.

## 3.1 microSD Boot Mode

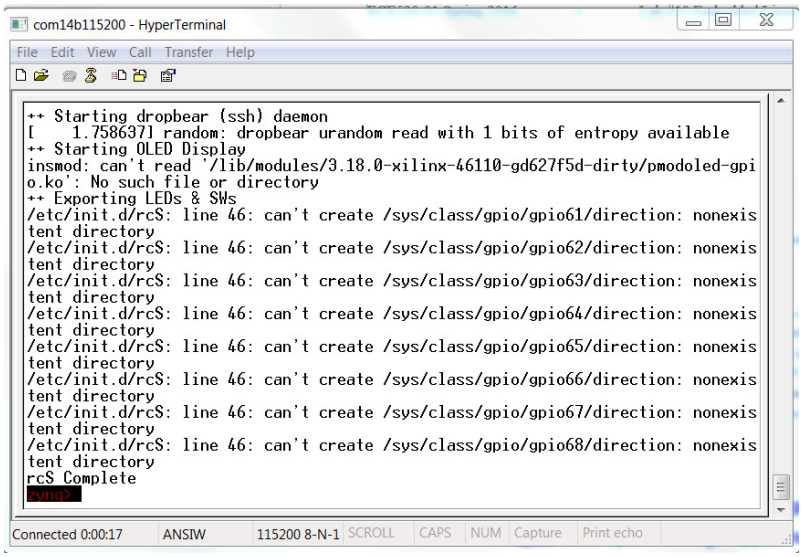
The ZYBO supports booting from a microSD card inserted into connector J4. The following procedure will allow you to boot the Zynq from microSD:

1) Format the microSD card with a FAT32 file system.
2) Copy the Zynq Boot Image created with Xilinx SDK to the microSD card.
3) Rename the Zynq Boot Image on the microSD card to BOOT.bin.
4) Eject the microSD card from your computer and insert it into connector J4 on the ZYBO.
5) Attach a power source to the ZYBO and select it using JP7.
6) Place a single jumper on JP5, shorting the two leftmost pins (labeled "SD").
7) Turn the board on. The board will now boot the image on the microSD card.

This boot loader should have four files on your micro SD card:

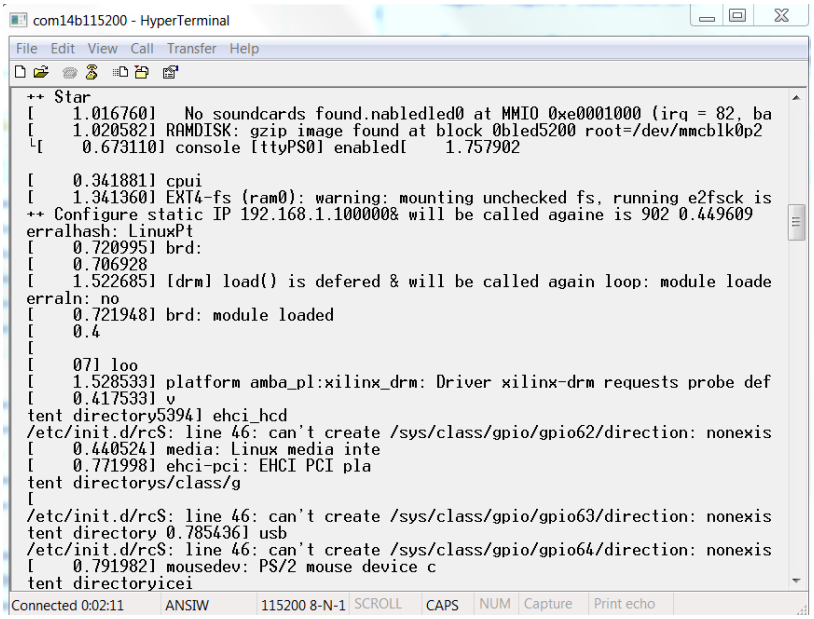



Power up your Zybo board and connect it to your terminal. Press reset button PS-SRST. You will see the following on your terminal.




If you scroll up to see displayed text, you will see warnings and error messages such as "Not sound cards found", "cannot create directories,", "GPIO IRQ not connected", and etc. the reason is because the BOOT.bin is from a Video application which may not match our Zybo board.

We will create our own BOOT.bin in the rest of this lab to remove some of these errors.




At this point, you have completed this part of the lab.

# 6 Update the ZYBO Base System Design to Vivado 2015.2

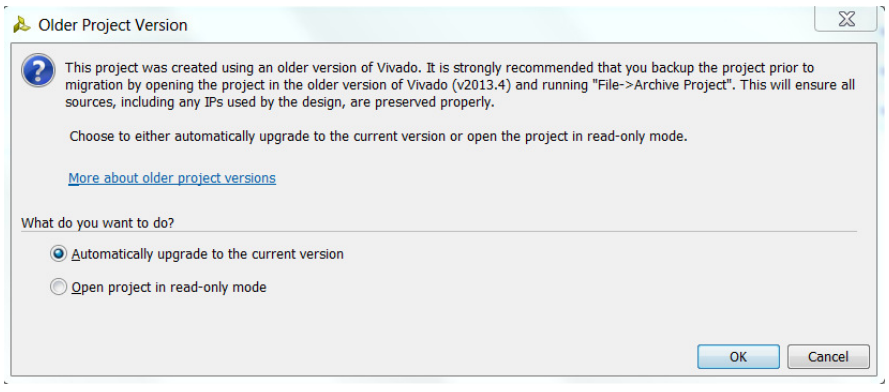
Following “Embedded Linux Hands-on Tutorial for the ZYBO” from Digilent to create the ZYBO Base System Design.

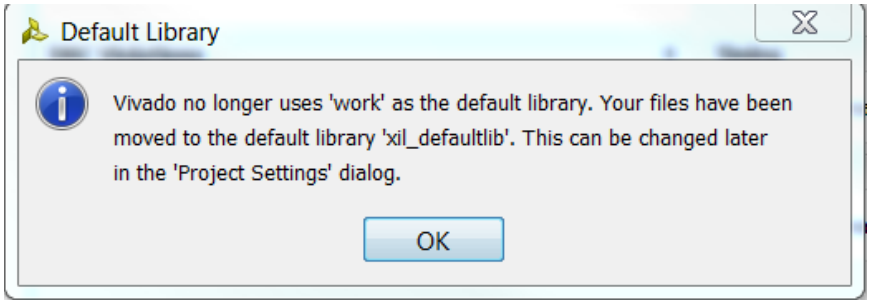
## 6.1 Download Zybo Base System Design zipped file

Download zybo_base_system.zip, from Zybo Base System Design, at https://reference.digilentinc.com/zybo:zybo. Copy unzipped folder zybo_base_system to C:xup/ or your Vivado project directory.

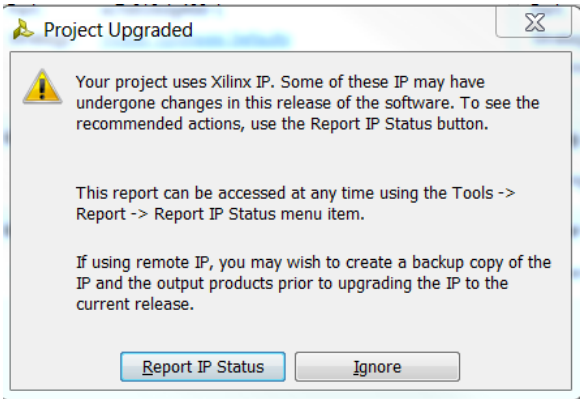
## 6.2 Open Vivado project file zybo_bsd

Open Vivado project file zybo_bsd at C:\xup\zybo_base_system\source\vivado\hw\zybo_bsd. And choose “Automatically upgrade to the current version”. Click “OK” on the default library ‘xil_defaultlib’.

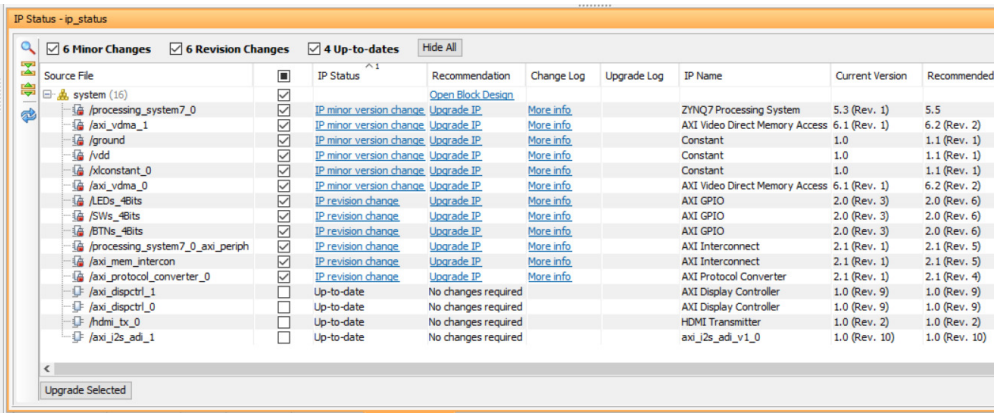


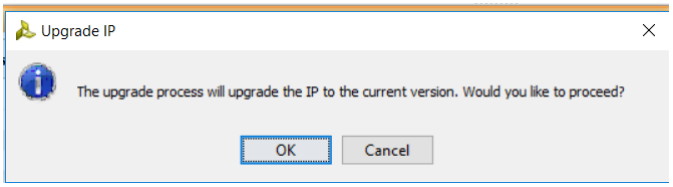



Choose “Report IP Status”.

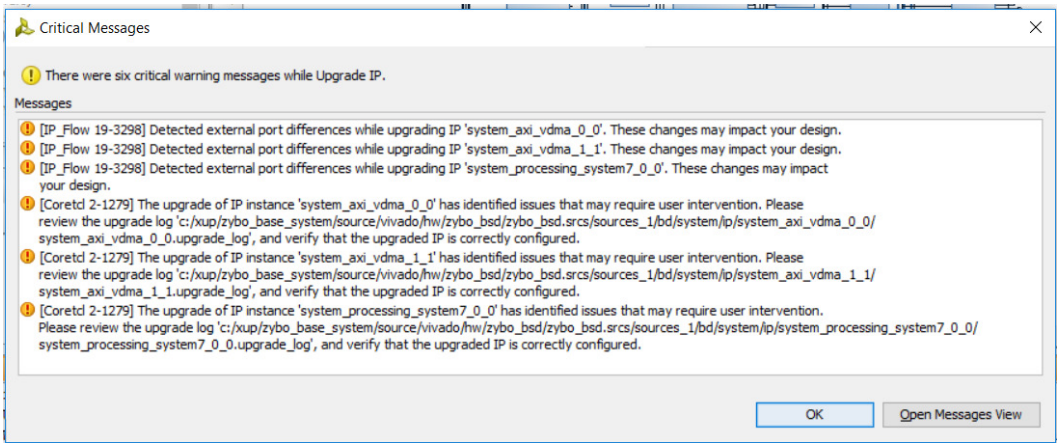



Choose “Update Selected” on the bottom left corner to update all IPs. Check “OK” to update IP.

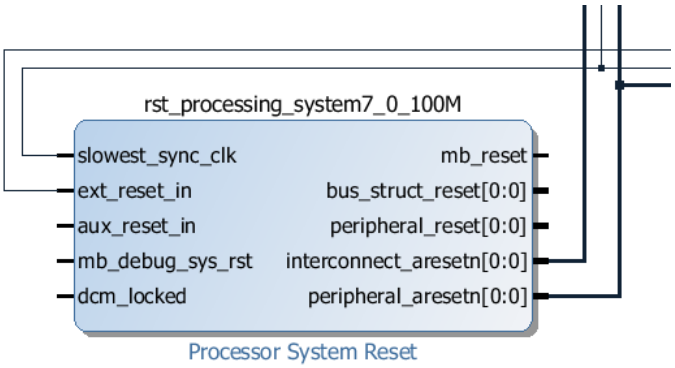



The following warnings will appear. All critical warnings are related to resets because the block design does not have a reset processing system and therefore all reset input pins are connected to Zynq reset outputs. For example, axi_vdma IPs uses FCLK_CLK1 and therefore their reset inputs are connected to FCLK_RESET1.



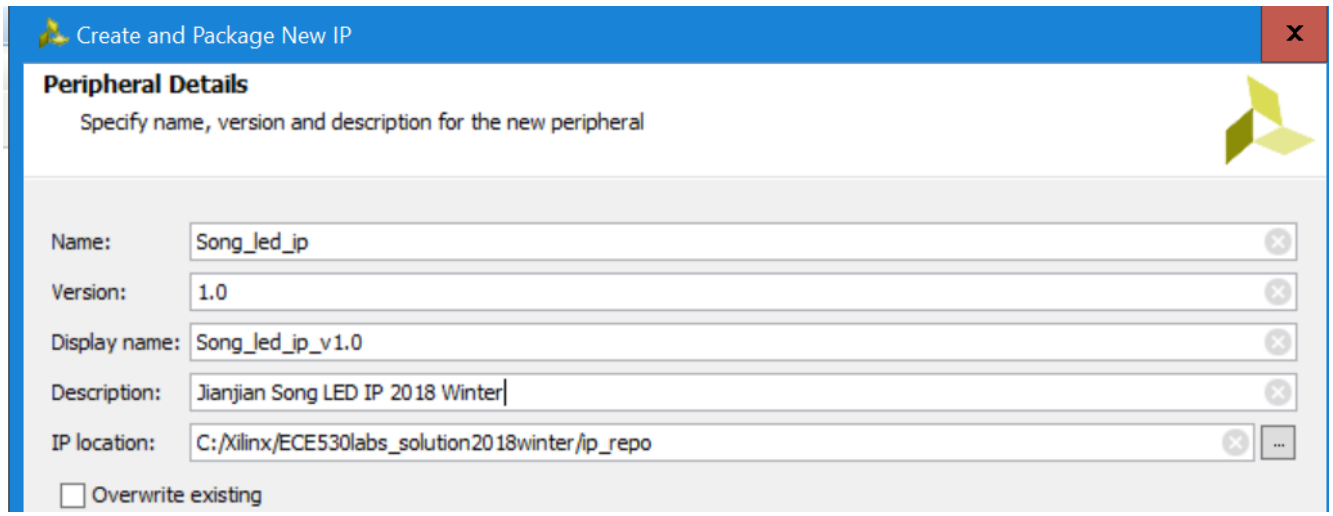

Reset processing system of Vivado 2015.4, which is missing from the Zybo Base System Design from Digilent. One way to remove the above warnings is to add rst_processing_system7_0_100M IP. But it may have unknown issues. Therefore, you may want to keep the original block design.




Save the project as a new project named lab11embeddedLinux.

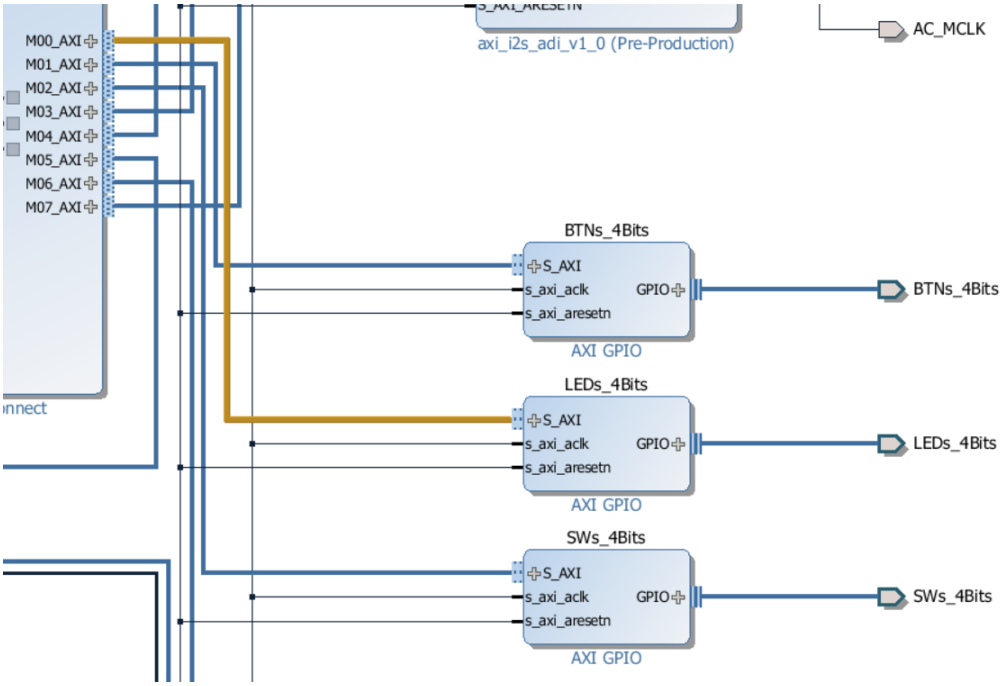
## 6.3 Create a new 4-bit LED IP with your last name

This lab will use an LED IP, which is the same as that from Lab 3 Adding Custom IP to the System of Xilinx Zynq Embedded System Design Tutorials. But you are required to make a new LED IP and name it with your last name_IP, e.g., Song_led_ip as shown below.

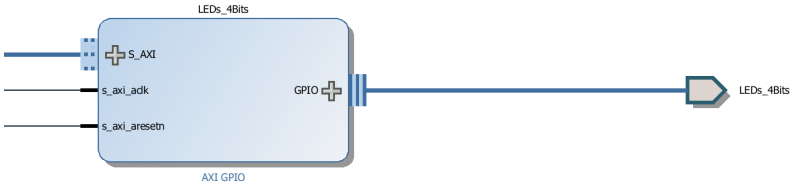


## 6.4 Create a 4-bit LED IP Core to replace the AXI GPIO IP

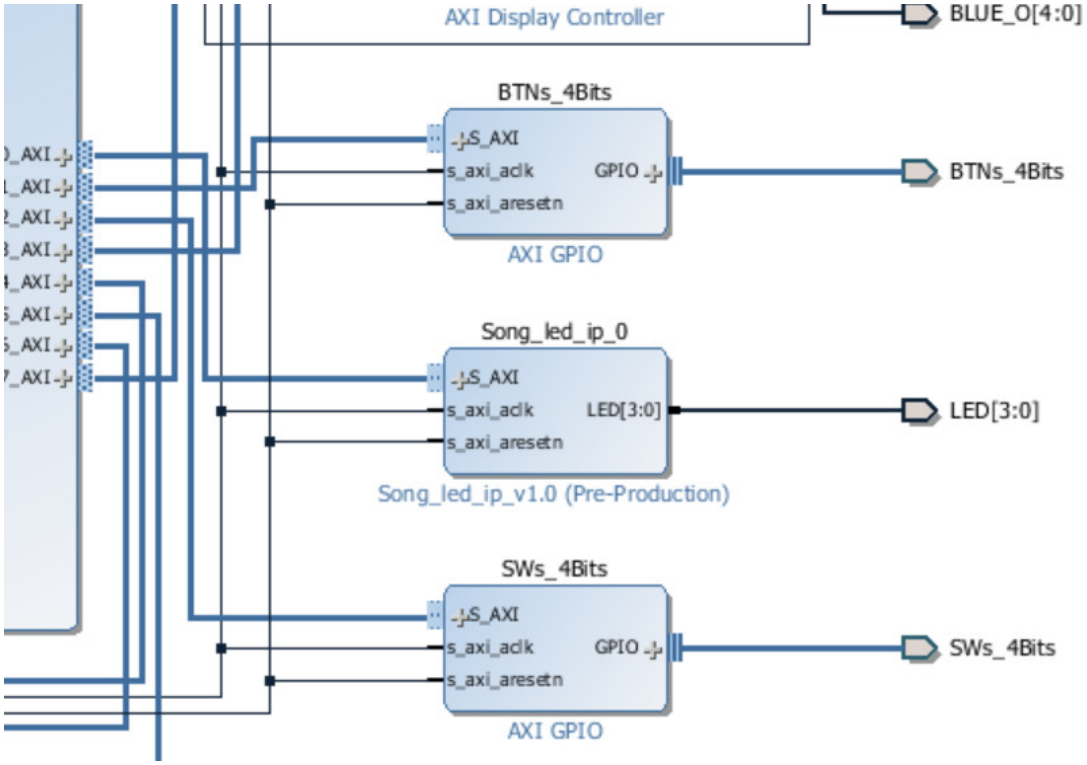
The original Zybo Base System Design uses an AXI GPIO IP to drive the four-bit LEDs. Replace this GPIO interface with your LED IP named your_last_name_led_ip.



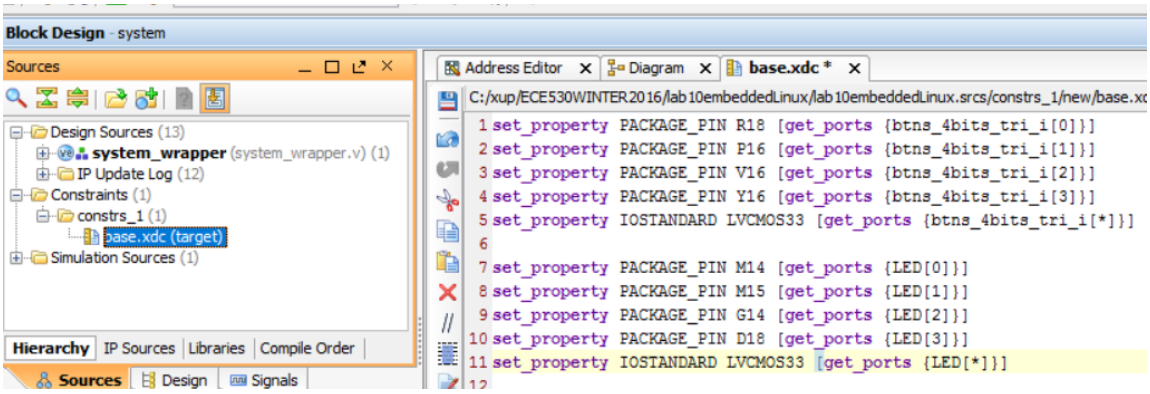

Here is the old link for LEDs_4bits.



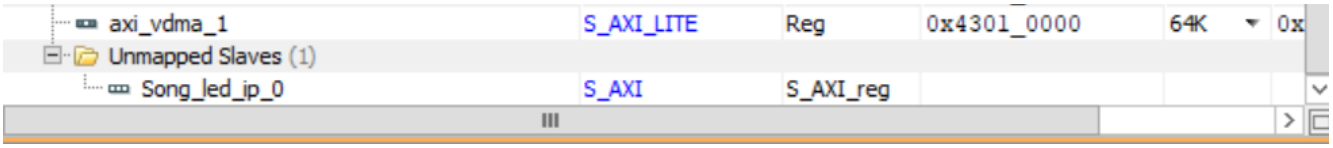

Remove AXI GPIO IP for LEDs and add your LED IP to your design.



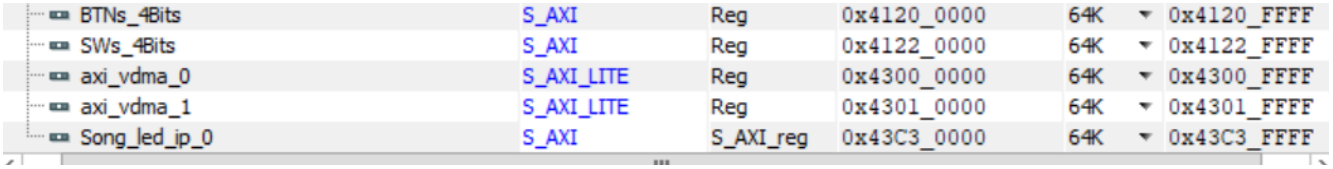

Open constraint file base.xdc under Constraints, change the led pins to LED,



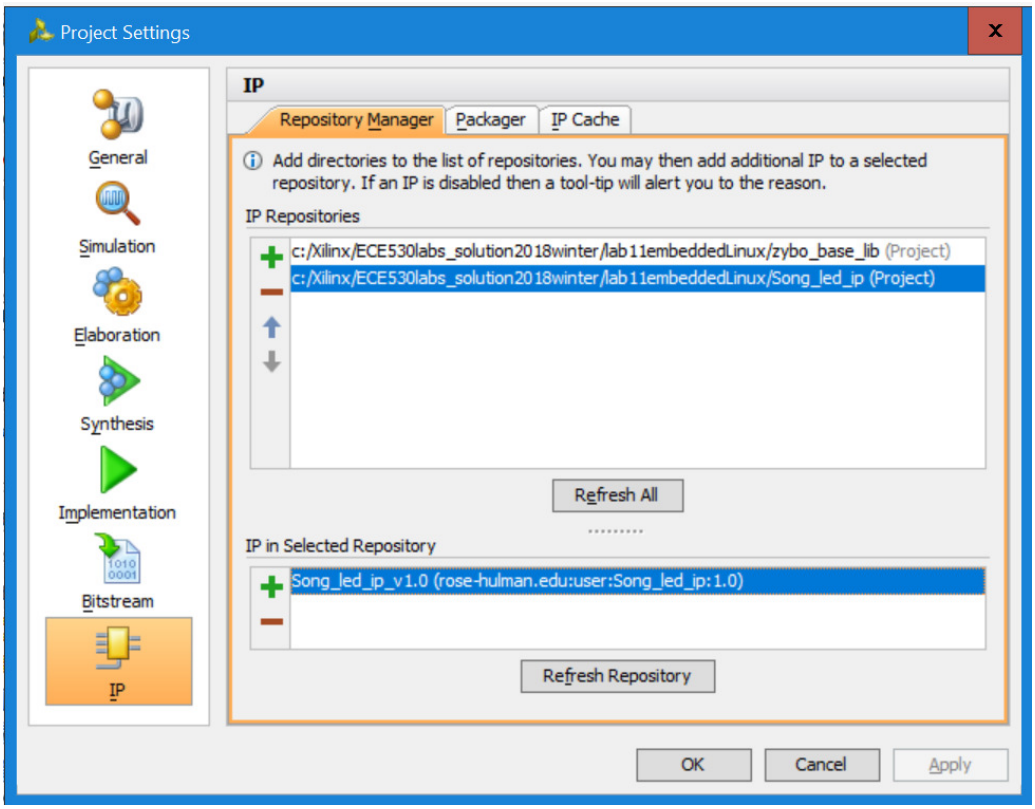

Click on Address Editor to see if your LED IP has been assigned addresses. Your LEDs might not be mapped to any address yet as shown below.




If it does not have address, right click on it and choose “Auto Assign Address”. This is very important. Otherwise, Song_led_ip_0 component will not be included in the driver library.



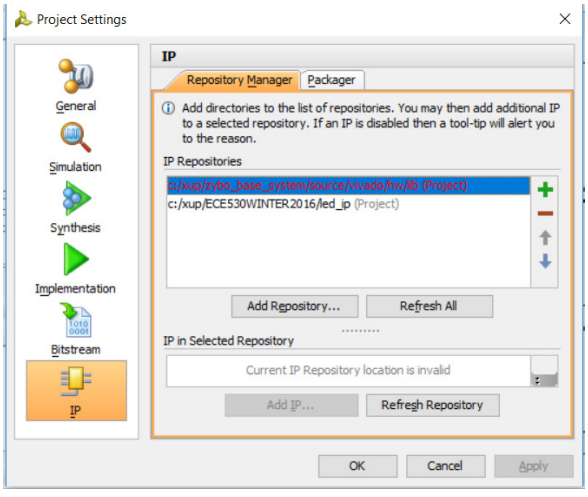

If you do not see your LED IP in your ip catalog, open Project settings->IP to add the IP to your catalog. In the following, the LED IP is called Song_led_ip.

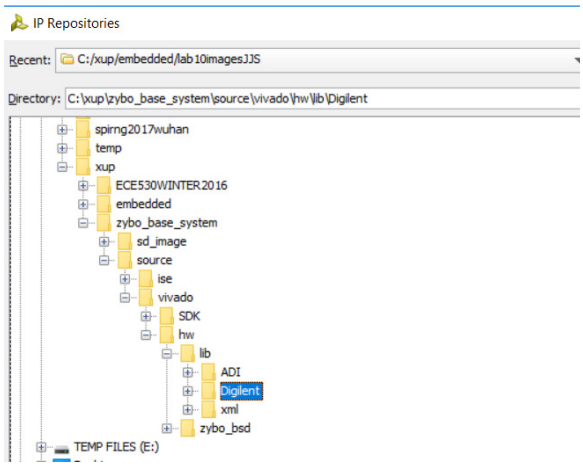



## 6.5 Verification Error

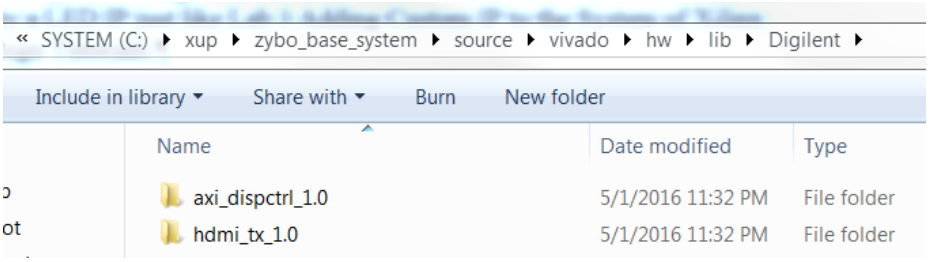
An IP was not found if the error is that it is locked. To fix these errors, make the IP repository links match library folder directories.


Open Project Settings->IP. Make sure the IP repository link is the same as the actual lib directory.

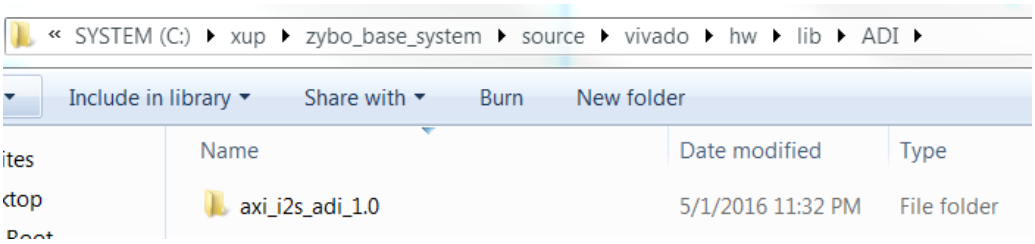


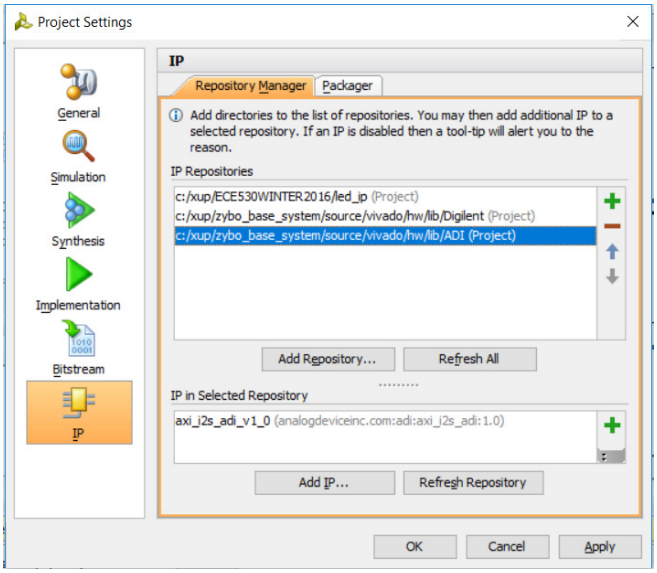
Find where an IP is located and add its repository link accordingly. Here are what folders should be in C:\xup\zybo_base_system\source\vivado\hw\lib\Digilent/.




Here is what should be in C:\xup\zybo_base_system\source\vivado\hw\lib\ADI/.




There could be up to three repositories as shown below so that all IPs are allocated.




## 6.6 Generate the bit stream file for your Zybo Base System Design

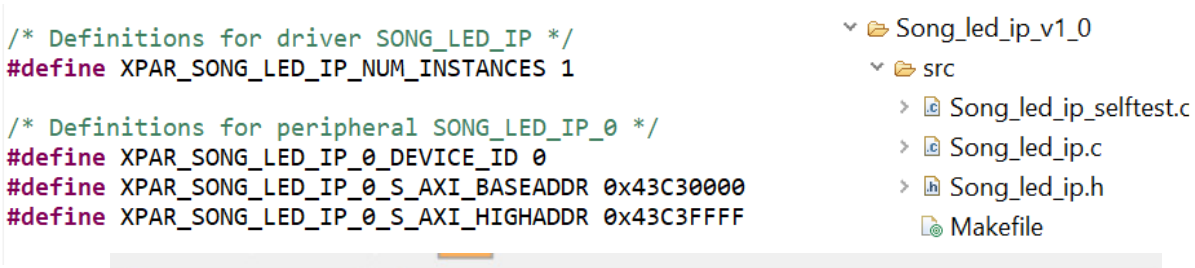
Generate “output products” before starting synthesis. Synthesize Implement your Zybo Base System Design. Check its IO pin assignment. Generate the bit stream file for it. Export it. Launch SDK.

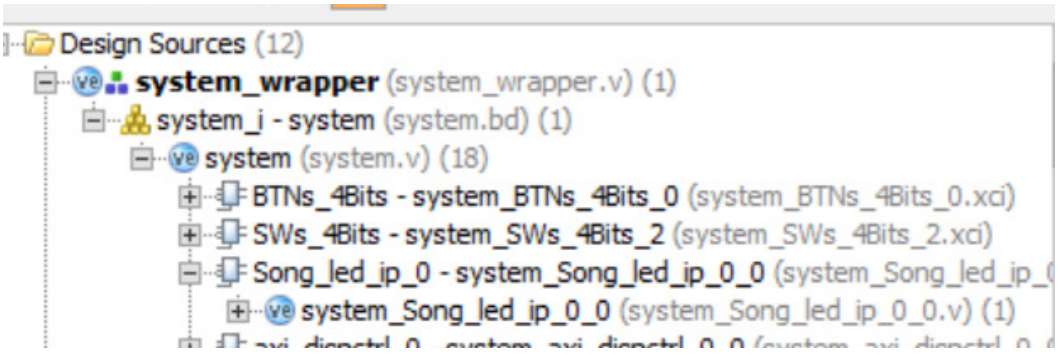
## 6.7 Create an application with Lab #5 code to verify the block design

The LED IP myLED is the same as Lab #5 of this class, add custom IP and write basic software, i.e., Xilinx Lab4 ZYNQ4 Writing basic software application. You can verity it by running Lab 4 code. Make sure to change the device ID in your Lab #5 code to match the current LED IP definition as shown below.

```
/* Definitions for driver SONG_LED_IP */
#define XPAR_SONG_LED_IP_NUM_INSTANCES 1

/* Definitions for peripheral SONG_LED_IP_0 */
#define XPAR_SONG_LED_IP_0_DEVICE_ID 0
#define XPAR_SONG_LED_IP_0_S_AXI_BASEADDR 0x43C30000
#define XPAR_SONG_LED_IP_0_S_AXI_HIGHADDR 0x43C3FFFF
```




```
#include "xparameters.h"
#include "xgpio.h"
#include "Song_led_ip.h"
//====================================================

int main (void)
{

   XGpio dip, push;
   int i, psb_check, dip_check;

   xil_printf("-- Start of the Program --\r\n");

   XGpio_Initialize(&dip, XPAR_SWS_4BITS_DEVICE_ID); // Modify this
   XGpio_SetDataDirection(&dip, 1, 0xffffffff);

   XGpio_Initialize(&push, XPAR_BTNS_4BITS_DEVICE_ID); // Modify this
   XGpio_SetDataDirection(&push, 1, 0xffffffff);


   while (1)
   {
      psb_check = XGpio_DiscreteRead(&push, 1);
      xil_printf("Push Buttons Status %x\r\n", psb_check);
      dip_check = XGpio_DiscreteRead(&dip, 1);
      xil_printf("DIP Switch Status %x\r\n", dip_check);

      // output dip switches value on LED_ip device
      SONG_LED_IP_mWriteReg(XPAR_SONG_LED_IP_0_S_AXI_BASEADDR, 0, dip_check);
      for (i=0; i<9999999; i++);
   }
}
```

## 6.8 (Optional) Create an application with Lab #10 code to verify one of the VGA IPs

It is optional but possible to verify one of the vdma IP and VGA display controller with Lab #10 code for DMA and VGA display. This is one example of using known working software to test new hardware.

# 7 Install Linux, VirtualBox and Xilinx SDK

You can either install a Linux virtual machine such as Virtual Box or VMware or Linux machine. Follow this document: Guidelines on VirtualBox Ubuntu and Xilinx SDK rev4-13-2019, to install a VirtualBox on your laptop.

You can follow a guideline "Guidelines on VirtualBox Ubuntu and Xilinx SDK rev4-13-2019" in pdf from the instructor to install Oracle VirtualBox and Xilinx SDK. If you have followed this guide to install Oracle VirtualBox, you can start your VituralBox and go to Section 8.

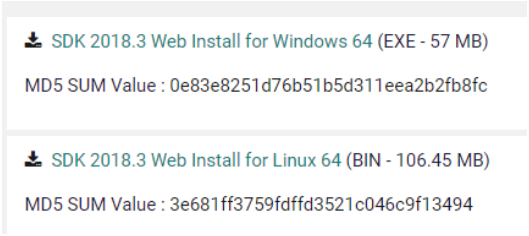
If you install a Linux machine, you will need to install Xilinx SDK for Linux to use Xilinx cross-compile tools. You can also install Linux version of Xilinx Vivado. But it is recommended that stand-alone Windows-based Vivado is used to create your hardware design.

# 8 Install Xilinx Linux SDK 2014.4 (64bit)

Xilinx SDK source files are needed to create tool chain functions for Xilinx Zynq chips.

XilinxSDK2014_fixed_broken_symlink.zip from the class folder on Moodle contains Xilinx 2014 SDK source files for Linux. They need to be installed to directory /opt/Xilinx. Shell settings to be used to compile Linux kernel for Zybo is located at /opt/Xilinx/SDK/2014.4/settings64.sh. This file needs to be sourced: >source /opt/Xilinx/SDK/2014.4/settings64.sh. You can also just install Xilinx SDK under Linux to use its cross-compile tools.

Newer versions of Xilinx SDK source files can be found from Xilinx by searching for "Software Development Kit Standalone WebInstall Client" under PetaLinux-Xilinx heading. The latest version is "SDK 2018.3 Web Install for Linux 64 (BIN 106.45MB). However, 2014 version of Xilinx SDK for Linux is used for this lab due to compatibility issues.




Download a copy of XilinxSDK2014_fixed_broken_symlink.zip and copy it to ~/Document. Here are instructions by David Robinson, a student of Rose-Hulman on how to set up Xilinx SDK under Ubuntu Linux.

Type the command below in the terminal. You will need to provide the terminal with your password once it finishes unzipping the Xilinx zip. This command unzips the Xilinx zip, moves it to the correct location, and recursively gives your user the ability execute programs in the Xilinx folder.

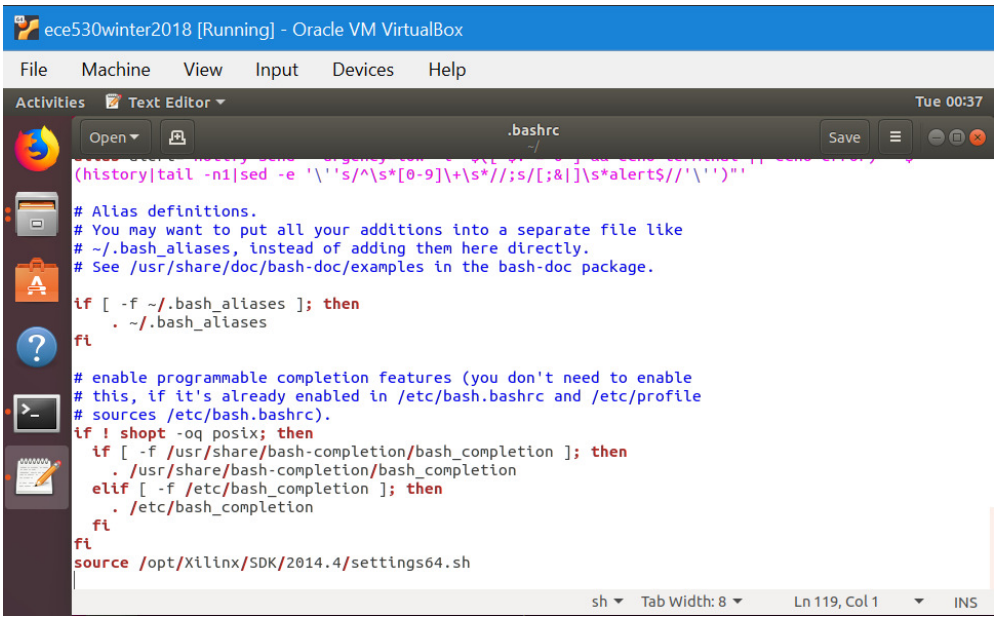
```
jianjiansong@jianjiansong-VirtualBox:~$ ls
Desktop  Documents  Downloads  firefox  Music  Pictures  Public  Templates  Videos
jianjiansong@jianjiansong-VirtualBox:~$ ls Documents/
XilinxSDK2014_fixed_broken_symlink.zip
jianjiansong@jianjiansong-VirtualBox:~$ unzip ~/Documents/XilinxSDK2014_fixed_broken_symlink.zip -d ~/Desktop/
```

```
jianjiansong@jianjiansong-VirtualBox:~$ ls ~/Desktop/
Xilinx
jianjiansong@jianjiansong-VirtualBox:~$ sudo mv ~/Desktop/Xilinx/ /opt/
[sudo] password for jianjiansong:
jianjiansong@jianjiansong-VirtualBox:~$ sudo chmod -R +x /opt/Xilinx/
jianjiansong@jianjiansong-VirtualBox:~$ cd /
jianjiansong@jianjiansong-VirtualBox:/$ ls /opt/Xilinx/
SDK
jianjiansong@jianjiansong-VirtualBox:/$
```

Next, type the command in your home directory. Type cd ~ enter to get to your home directory if you are not in it. gedit is a text editor to add a command to .bashrc file. .bashrc file is a shell script that Bash runs whenever it is started interactively. It initializes an interactive shell session. You can put any command in that file that you could type at the command prompt. On Linux, bash is the standard shell for common users.

```
jianjiansong@jianjiansong-VirtualBox:~$ ls -a
.              .bashrc  Documents  .ICEauthority  Pictures                 Templates                  .vboxclient-seamless.pid
..             .cache   Downloads  .local         .profile                 .vboxclient-clipboard.pid  Videos
.bash_history  .config  firefox    .mozilla       Public                   .vboxclient-display.pid
.bash_logout   Desktop  .gnupg     Music          .sudo_as_admin_successful  .vboxclient-draganddrop.pid
jianjiansong@jianjiansong-VirtualBox:~$ gedit .bashrc
jianjiansong@jianjiansong-VirtualBox:~$
```

A text editor will appear and add to the bottom of the file “source /opt/Xilinx/SDK/2014.4/settings64.sh”. Save (Ctrl+s) and close the text editor. Note, in Ubuntu the “X” button to close windows is in normally in the top left corner of the window, but if it’s maximized the “X” button is hidden, it’s there I promise. Move your mouse to the menu bar and it will appear.




If you have downloaded Xilinx.tar.gz file instead of Xilinx.zip file, you can use the following command to open and install it.

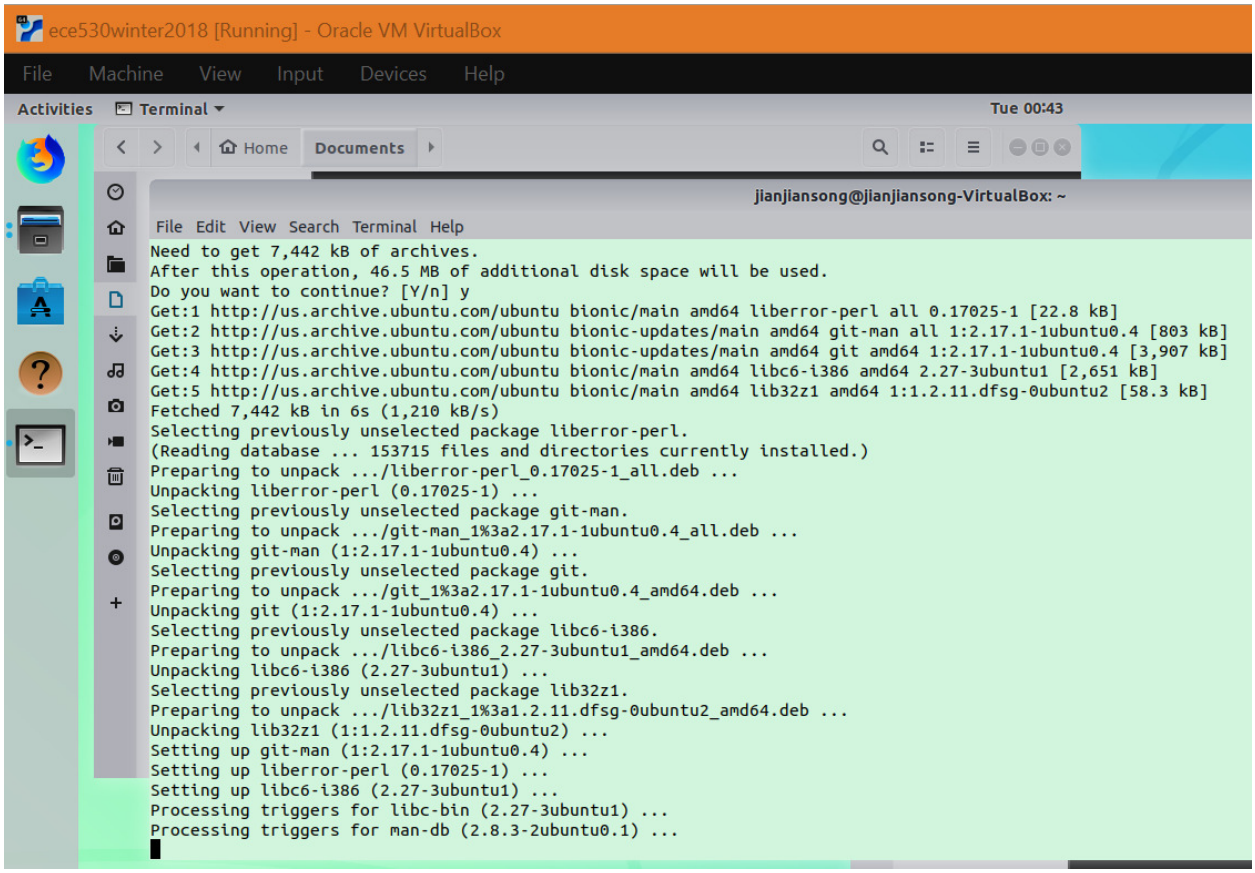
```
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
jianjiansong@jianjiansong-VirtualBox:~$ cd Downloads/
jianjiansong@jianjiansong-VirtualBox:~/Downloads$ ls
Xilinx.tar.gz
jianjiansong@jianjiansong-VirtualBox:~/Downloads$ cd ~
jianjiansong@jianjiansong-VirtualBox:~$ tar zxvf ~/Downloads/Xilinx.tar.gz -C ~/Desktop/ && sudo mv ~/Desktop/Xilinx /opt/
```

## 9 Install Git

Next, we need to install Git and the 32-bit runtime. You may have to run commands with sudo to get root permission.

```
jianjiansong@jianjiansong-VirtualBox:~$ sudo apt-get install git lib32z1
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  git-man libc6-i386 liberror-perl
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui gitk gitweb git-cvs git-mediawiki git-svn
The following NEW packages will be installed:
  git git-man lib32z1 libc6-i386 liberror-perl
0 upgraded, 5 newly installed, 0 to remove and 11 not upgraded.
Need to get 7,442 kB of archives.
After this operation, 46.5 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

There is no space between “apt-get”.




## 10 Install gcc compilers and 32-bit gcc libraries

Install 64-bit gcc: sudo apt-get install gcc-arm-none-eabi gcc-arm-linux-gnueabi.

Install 32-bit gcc library: sudo apt-get install lib32z1 lib32ncurses5 libbz2-1.0:i386 lib32stdc++6.

Install gcc:  sudo apt install gcc.

```
jianjiansong@jianjiansong-VirtualBox:~$ sudo apt-get install gcc-arm-none-eabi gcc-arm-linux-gnueabi
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  binutils-arm-linux-gnueabi binutils-arm-none-eabi cpp-7-arm-linux-gnueabi cpp-arm-linux-gnueabi gcc-7-arm-linux
  gcc-7-arm-linux-gnueabi-base gcc-7-cross-base gcc-8-cross-base libasan4-armel-cross libatomic1-armel-cross libc
  libcilkrts5-armel-cross libgcc-7-dev-armel-cross libgcc1-armel-cross libgomp1-armel-cross libisl15 libnewlib-ar
  libstdc++-arm-none-eabi-newlib libstdc++6-armel-cross libubsan0-armel-cross linux-libc-dev-armel-cross
Suggested packages:
  binutils-doc gcc-7-locales cpp-doc gcc-7-multilib-arm-linux-gnueabi gcc-7-doc libgcc1-dbg-armel-cross libgomp1-
  libitm1-dbg-armel-cross libatomic1-dbg-armel-cross libasan4-dbg-armel-cross liblsan0-dbg-armel-cross libtsan0-d
  libubsan0-dbg-armel-cross libcilkrts5-dbg-armel-cross libmpx2-dbg-armel-cross libquadmath0-dbg-armel-cross auto
  gdb-arm-linux-gnueabi gcc-doc libnewlib-doc
The following NEW packages will be installed:
  binutils-arm-linux-gnueabi binutils-arm-none-eabi cpp-7-arm-linux-gnueabi cpp-arm-linux-gnueabi gcc-7-arm-linux
  gcc-7-arm-linux-gnueabi-base gcc-7-cross-base gcc-8-cross-base gcc-arm-linux-gnueabi gcc-arm-none-eabi libasan4
  libc6-armel-cross libc6-dev-armel-cross libcilkrts5-armel-cross libgcc-7-dev-armel-cross libgcc1-armel-cross li
  libnewlib-arm-none-eabi libnewlib-dev libstdc++-arm-none-eabi-newlib libstdc++6-armel-cross libubsan0-armel-cro
0 upgraded, 25 newly installed, 0 to remove and 11 not upgraded.
Need to get 135 MB of archives.
After this operation, 1,077 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

```
jianjiansong@jianjiansong-VirtualBox:~$ sudo apt-get install lib32z1 lib32ncurses5 libbz2-1.0:i386 lib32stdc++6
Reading package lists... Done
Building dependency tree
Reading state information... Done
lib32z1 is already the newest version (1:1.2.11.dfsg-0ubuntu2).
The following additional packages will be installed:
  gcc-8-base:i386 lib32gcc1 lib32tinfo5 libc6:i386 libgcc1:i386
Suggested packages:
  glibc-doc:i386 locales:i386
The following NEW packages will be installed:
  gcc-8-base:i386 lib32gcc1 lib32ncurses5 lib32stdc++6 lib32tinfo5 libbz2-1.0:i386 libc6:i386 libgcc1:i386
0 upgraded, 8 newly installed, 0 to remove and 11 not upgraded.
Need to get 3,289 kB of archives.
After this operation, 14.0 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

After the installation, you would not see the following error when compiling myled.c

**Re: PC with Ubuntu 14.04 64bit, SDK 2015.1 fails to compile (make error 2) compiler not found** Options

05-15-2015 02:02 PM

This has always been the case with the toolchain on 64-bit Ubuntu. I used to need these to just start Vivado when it was 32-bit, and now after upgrading to the 64-bit 2015.1, I forgot all about it until my system complained that *arm-xilinx-eabi-gcc could not be found*. These are third-party (GNU) projects/programs. Xilinx just packs a stable version in with their installer for convenience, which they probably should have built for amd64 platforms if they're dropping support for 32-bit otherwise, but they're likely just going to push it out later.

If you'd rather use a 64-bit version, you can install it with apt-get or whatever your distro's package manager is, then in the SDK you can change the project settings to use your own toolchain.

```
sudo apt-get install gcc-arm-none-eabi gcc-arm-linux-gnueabi
```

If you think Xilinx is bad, try using TI's Code Composer Studio and the TI-RTOS/SYSBIOS real-time system... I did, and I've cherished every moment I spend with the Xilinx tools ever since, lol, and TI doesn't have to integrate a whole toolchain for FPGA development either.

## 11 Compile U-Boot.elf File (Section 2 of "Embedded Linux® Hands-on Tutorial")

It is necessary to create a new u-boot.elf file. This is done with Linux and the u-boot.elf file needs to be copied to your Windows to be used to create the BOOT.BIN file with SDK under Windows. You can read Section 2, page 16, of "Embedded Linux Hands-on Tutorial for the Zybo" to compile u-boot and generate the u-boot.elf file. You can also just follow this section.

The source files to compile the correct version of u-boot source files is available from GitHub repository: https://github.com/DigilentInc/u-boot-Diglent-Dev.git. Clone this directory on your VirtualBox or Linus Machine with the following command. There is no space between master and next: master-next.

```
jianjiansong@jianjiansong-VirtualBox:~$ ls
Desktop  Documents  Downloads  drivers  firefox  Linux-Digilent-Dev  Music  Pictures  Public  Templates  Videos
jianjiansong@jianjiansong-VirtualBox:~$ git clone -b master-next https://github.com/DigilentInc/u-boot-Digilent-Dev.git
Cloning into 'u-boot-Digilent-Dev'...
remote: Enumerating objects: 253091, done.
remote: Total 253091 (delta 0), reused 0 (delta 0), pack-reused 253091
Receiving objects: 100% (253091/253091), 72.68 MiB | 6.96 MiB/s, done.
Resolving deltas: 100% (200085/200085), done.
Checking out files: 100% (8120/8120), done.
jianjiansong@jianjiansong-VirtualBox:~$ ls
Desktop    Downloads  firefox           Music     Public     u-boot-Digilent-Dev
Documents  drivers    Linux-Digilent-Dev  Pictures  Templates  Videos
jianjiansong@jianjiansong-VirtualBox:~$
```

The following command will set up the correct build environment or configuration parameters. Sometimes you may have to quit your VirtualBox and start a new one for some configuration parameters to be set correctly. If you see configuration errors, restarting your VirtualBox may fix them.

```
api    boards.cfg  CREDITS  drivers   fs       Licenses  mkconfig  post      scripts          System.map  u-boot.img  u-boot.srec
arch   common      disk     dts       include  MAKEALL   nand_spl  README    snapshot.commit  test        u-boot.lds
board  config.mk   doc      examples  lib      Makefile  net       rules.mk  spl              tools       u-boot.map
jianjiansong@jianjiansong-VirtualBox:~/u-boot-Digilent-Dev$ make CROSS_COMPILE=arm-xilinx-linux-gnueabi- zynq_zybo_config
Configuring for zynq_zybo board...
jianjiansong@jianjiansong-VirtualBox:~/u-boot-Digilent-Dev$
```

The following command will build the u-boot file as u-boot. The error 127 at the end of this compilation can be ignored as we do not need to generate boot.bin file.

```
board  config.mk   doc      examples  lib      Makefile  net       rules.mk  spl              tools       u-boot.map
jianjiansong@jianjiansong-VirtualBox:~/u-boot-Digilent-Dev$ make CROSS_COMPILE=arm-xilinx-linux-gnueabi-
make -C tools all
make[1]: Entering directory '/home/jianjiansong/u-boot-Digilent-Dev/tools'
gcc -Wall -Wstrict-prototypes -O2 -fomit-frame-pointer -include /home/jianjiansong/u-boot-Digilent-Dev/include/libfdt_env.h -idirafter /home/j
song/u-boot-Digilent-Dev/include -idirafter /home/jianjiansong/u-boot-Digilent-Dev/arch/arm/include -idirafter /home/jianjiansong/u-boot-Digil
/include -I /home/jianjiansong/u-boot-Digilent-Dev/lib/libfdt -I /home/jianjiansong/u-boot-Digilent-Dev/tools -DCONFIG_SYS_TEXT_BASE=0x4000000
HOSTCC -D__KERNEL_STRICT_NAMES -D_GNU_SOURCE -pedantic  -o proftool proftool.o
strip proftool
make[2]: Entering directory '/home/jianjiansong/u-boot-Digilent-Dev/tools/kernel-doc'
make[2]: Nothing to be done for 'all'.
make[2]: Leaving directory '/home/jianjiansong/u-boot-Digilent-Dev/tools/kernel-doc'
make[1]: Leaving directory '/home/jianjiansong/u-boot-Digilent-Dev/tools'
make -f /home/jianjiansong/u-boot-Digilent-Dev/scripts/Makefile.build -C api/
```

The compiled u-boot file is simple u-boot. It needs to be changed to u-boot.elf (Executable and Linkable File) to be included into BOOT.BIN for the boot loader.

Copy u-boot file as u-boot.elf file to your directory for all boot loader files as ~/Desktop/ in the following.

Make

```
Entry Point:  00000000
tools/zynq-boot-bin.py -o boot.bin -u spl/u-boot-spl.bin
/usr/bin/env: ‘python’: No such file or directory
Makefile:516: recipe for target 'boot.bin' failed
make: *** [boot.bin] Error 127
jianjiansong@jianjiansong-VirtualBox:~/u-boot-Digilent-Dev$ ls
api    boards.cfg  CREDITS  drivers   fs       Licenses  mkconfig  post      scripts          System.map  u-boot      u-boot.lds
arch   common      disk     dts       include  MAKEALL   nand_spl  README    snapshot.commit  test        u-boot.bin  u-boot.map
board  config.mk   doc      examples  lib      Makefile  net       rules.mk  spl              tools       u-boot.img  u-boot.srec
jianjiansong@jianjiansong-VirtualBox:~/u-boot-Digilent-Dev$ cp u-boot u-boot.elf && mv u-boot.elf ~/Desktop/
jianjiansong@jianjiansong-VirtualBox:~/u-boot-Digilent-Dev$ cd ~/Desktop/
jianjiansong@jianjiansong-VirtualBox:~/Desktop$ ls
u-boot.elf
jianjiansong@jianjiansong-VirtualBox:~/Desktop$
```

## 12 Generate BOOT.BIN

Section 3 on page 16 of “Embedded Linux Hands-on Tutorial for the Zybo” describes this process.

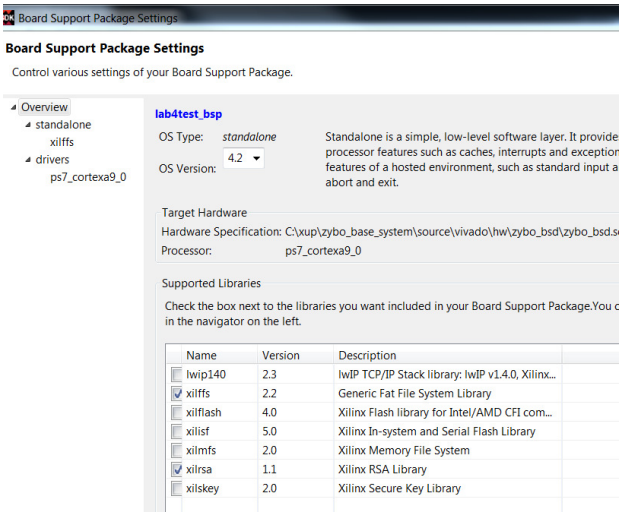
You can use the same SDK for your Zybo base system design with the led IP to create a BOOT.BIN file. Your BOOT.BIN file should contain three files: FSBL file, *.bit file and the u-boot.elf file.

## 12.1 Copy u-boot file from Section 8

You should have created a new u-boot.elf file from Section 8 and now copy it to your SDK folder.

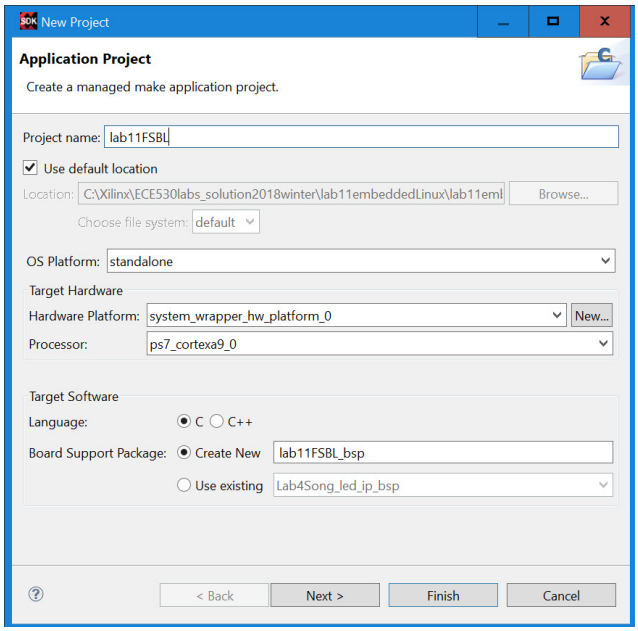
## 12.2 Add options to Board Support Package Settings

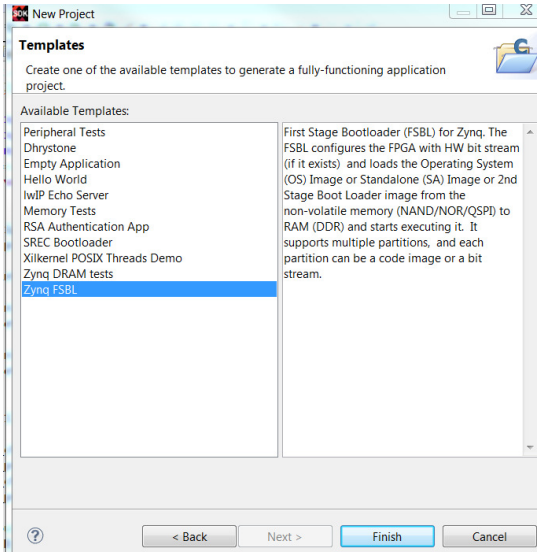
In SDK, after a project has been created and tested, right click on *.bsp and click Board Support Package Settings. Choose xilffs and xilrsa and click OK. (This is required for the next step to create the FSBL.)




## 12.3 Open SDK

Create a new application project for FSBL and choose "Use existing" board support package. Name the project lab11FSBL. Click "Next".




Choose Zybo FSBL. Click "Finish".

## 12.4 Set the mac address for Ethernet by replacing the default fsbl_hooks.c with the one from Zybo base system.

Copy fsbl_hooks.c from folder /zybo_base_system/source/vivado/SDK/fsbl to your FSBL project folder /src folder such as C:\..\lab11embeddedLinux.sdk\lab11EmbeddedLinuxFSBL\src.

## 12.5 Create a BOOT.BIN file

Make a directory your project folder/sd_image/ under your project folder.

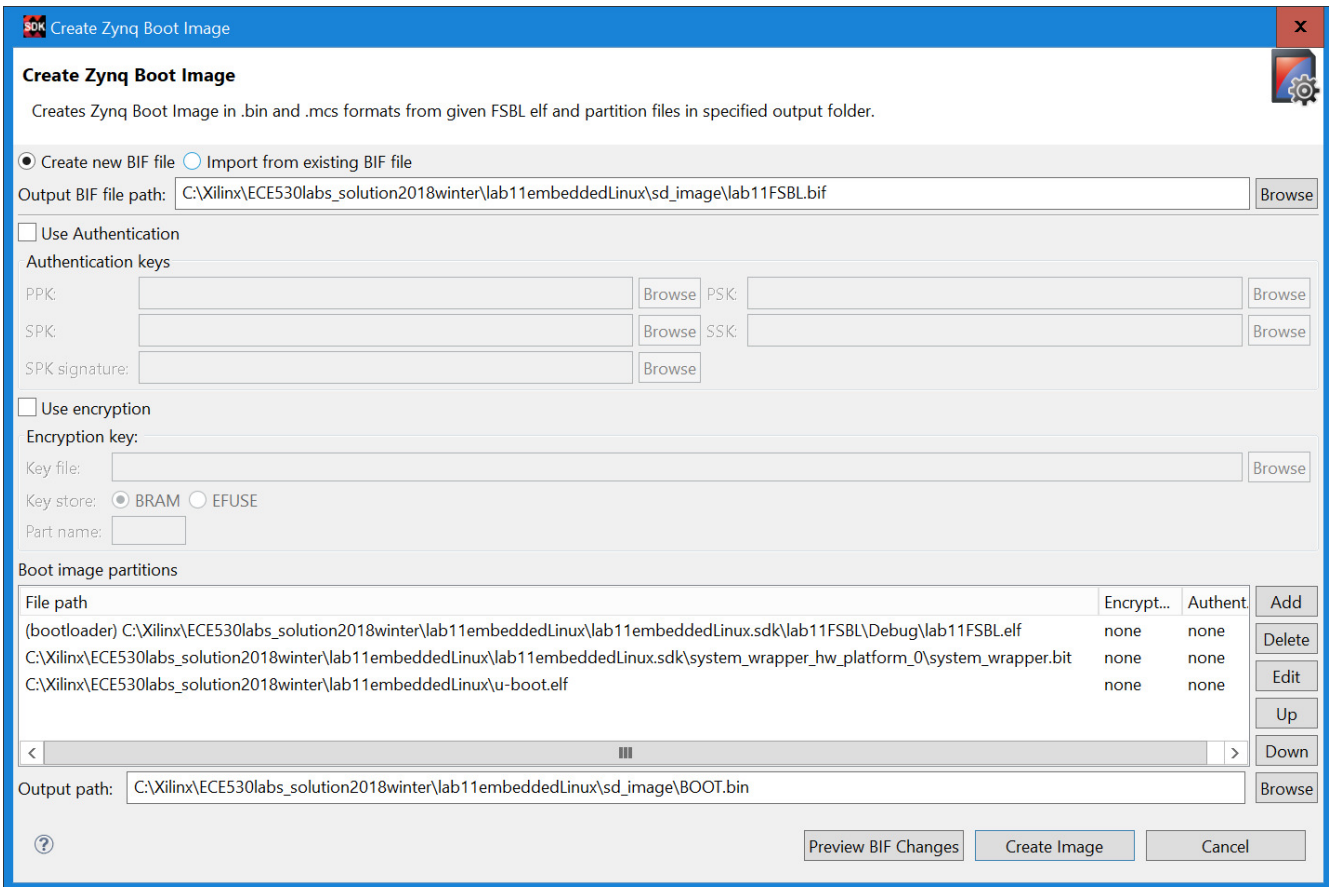
In SDK, select Xilinx Tools->Create Zynq Boot Image. Click on the Browse button of Output

BIF field to browse to your image directory in /sd_image/. Add three files: one boot loader and two data files.

Add your FSBL.elf file as boot loader file from the /Debug folder of your FSBL project folder such as C:\....\lab11embeddedLinux\lab11embeddedLinux.sdk\lab11FSBL\Debug.

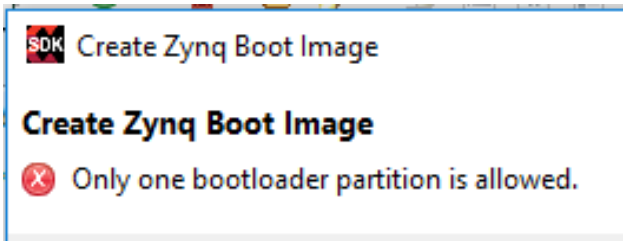
Add system_wrapper.bit file as data file to the Boot image partitions from C:\....\lab11embeddedLinux\lab11embeddedLinux.sdk\system_wrapper_hw_platform_0.

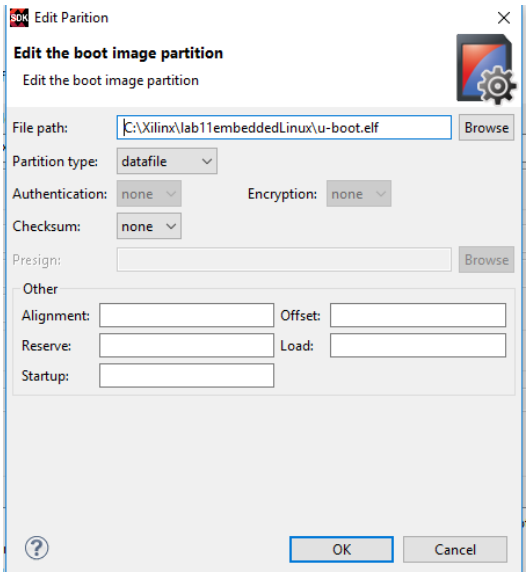
Add u-boot.elf file you have created from Section 8 as the last data file.




Click “Create image” to generate BOOT.bin file to be stored in /sd_image folder.

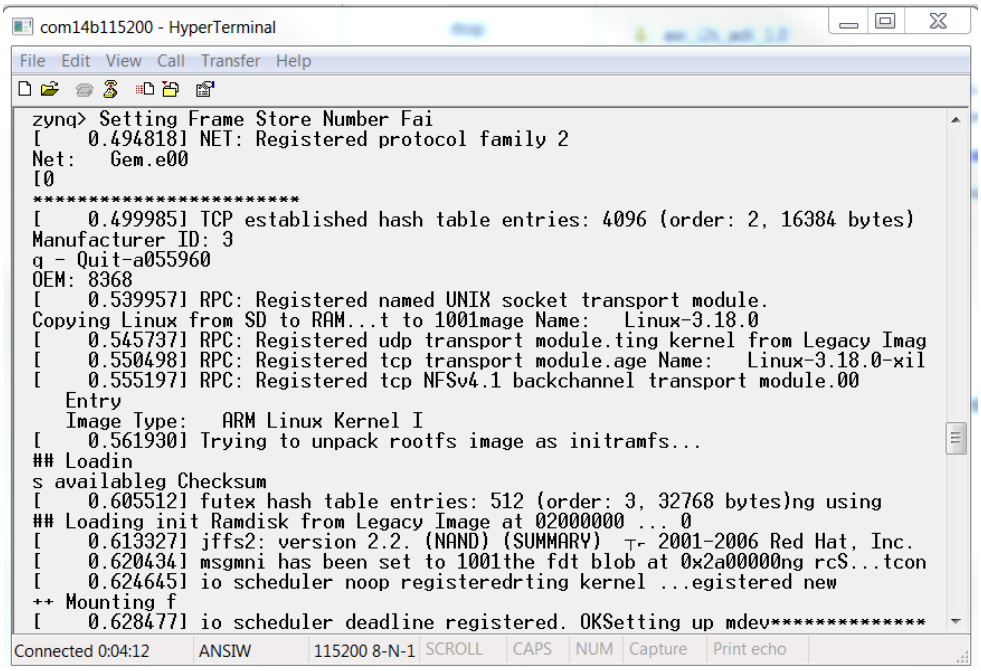
The following error may appear if more than one file has Partition type as bootloader. Only the FSBL file is of type Bootloader. The other files are all of type datafile.



## 12.6 Test your BOOT.bin file

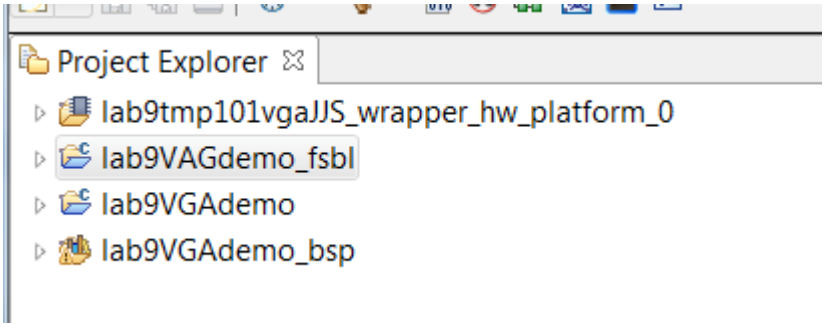
Replace the BOOT.bin file on your micro SD card in Section 4 with your BOOT.bin file. Boot Linux with your Zybo from your micro SD card to see if it works. Notice the Ethernet installation seems to be correct.

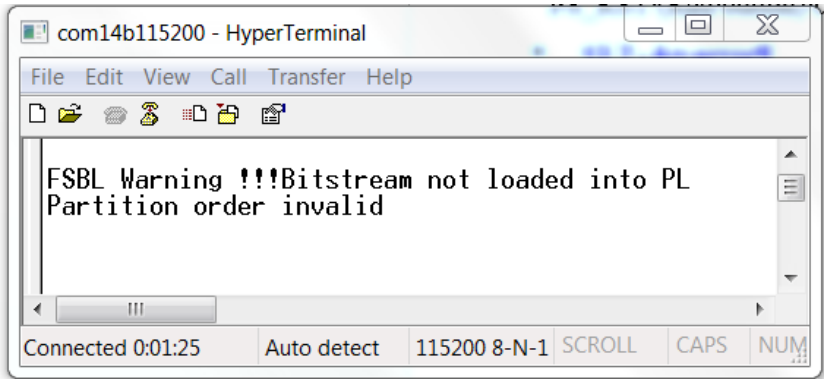



## 12.7 A possible error

The following error occurred because the order of three files was wrong. The bit stream file was placed after the application *.elf file. In the Boot image partitions pane, the bit stream file *.bit must be before the application file *.elf.

You can right click on _fsbl folder under your Project Explorer menu to open Create Boot Image. You can then edit this configuration and create another boot loader image. You may notice that *.bit file is gone. Add it again and make sure *.bit is before *.elf.



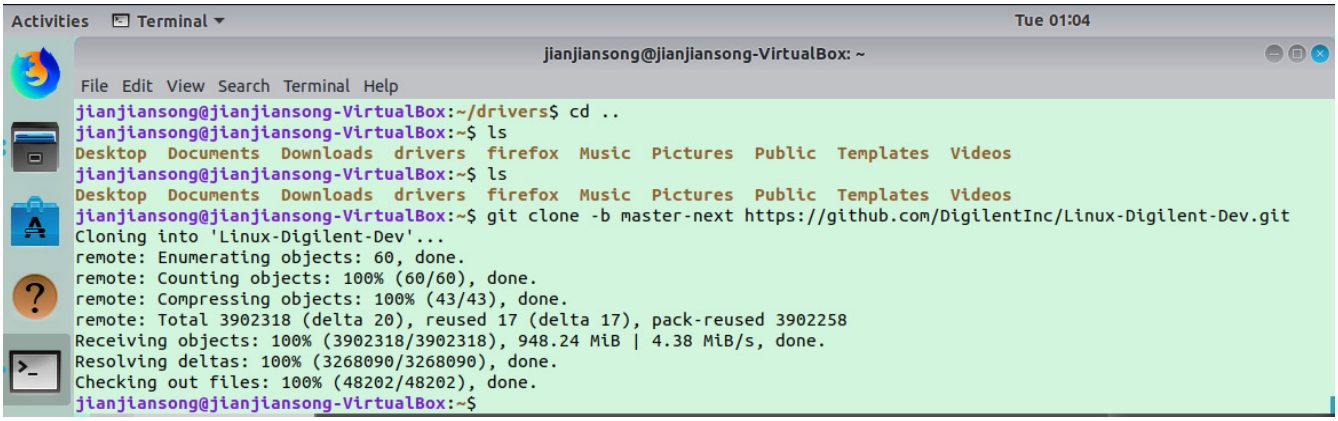



# 13 Compile Linux Kernel (Section 4 of "Embedded Linux® Hands-on Tutorial")

Follow Section 4 on page 24 of "Embedded Linux Hands-on Tutorial for the Zybo". The current directory should be /Linux-Digilent-Dev under your user home directory.

## 13.1 Get Linux kernel source from Digilent Git repository

Run the following command to get correct source files from github. You may need to restart your VirtualBox after cloning to remove some errors.




## 13.2 Configure the Linux kernel for Zybo

Make sure u-boot tools path is added to the path variables. The current directory should be /Linux-Digilent-Dev under your user home directory. If mkimage command is not found, run: sudo apt install u-boot-tools. Restarting your VirtualBox may help to set some configuration parameters correctly. Run the following commands under Linux-Digilent-Dev directory.

```
jianjiansong@jianjiansong-VirtualBox:~$ ls
Desktop  Documents  Downloads  drivers  firefox  Linux-Digilent-Dev  Music  Pictures  Public  Templates  u-boot-Digilent-Dev  Videos
jianjiansong@jianjiansong-VirtualBox:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
jianjiansong@jianjiansong-VirtualBox:~$ PATH=$PATH:/home/jianjiansong/u-boot-Digilent-Dev/tools/
jianjiansong@jianjiansong-VirtualBox:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/home/jianjiansong/u-boot-Digilent-Dev/too
ls/
jianjiansong@jianjiansong-VirtualBox:~$
```

Compile configuration first and then compile the Linux kernel for this configuration. After these two steps in Figures 47 and 48, follow Figure 49 to make uImage. Run the following command to use default configuration file for zybo board as in Figure 47.

```
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev$ make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- xilinx_zynq_defconfig
#
# configuration written to .config
#
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev$
```

Run the following command to set configure as in Figure 48.

```
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev$ ls
arch     CREDITS        drivers   include  Kbuild   lib          mm             README          scripts   System.map  virt
block    crypto         firmware  init     Kconfig  MAINTAINERS  Module.symvers  REPORTING-BUGS  security  tools       vmlinux
COPYING  Documentation  fs        ipc      kernel   Makefile     net            samples         sound     usr         vmlinux.o
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev$ make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- scripts/kconfig/conf -silentoldconfig Kconfig
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev$
```

Run the following command to make uImage file. This may take about 17 minutes. If you get an error that mkimage command is not found, run "sudo apt install u-boot-tools".

```
COPYING  Documentation  fs        ipc      kernel   Makefile     net            samples         sound     usr         vmlinux.o
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev$ make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- UIMAGE_LOADADDR=0x8000 uImage
  CHK     include/config/kernel.release
  CHK     include/generated/uapi/linux/version.h
  CHK     include/generated/utsrelease.h
make[1]: 'include/generated/mach-types.h' is up to date.
  CALL    scripts/checksyscalls.sh
```

```
Image Name:   Linux-3.18.0-xilinx-46110-gd627f
Created:      Tue Feb  5 12:35:50 2019
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    3459272 Bytes = 3378.20 kB = 3.30 MB
Load Address: 00008000
Entry Point:  00008000
  Image arch/arm/boot/uImage is ready
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev$ cd arch/arm/boot/
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev/arch/arm/boot$ ls
bootp  compressed  dts  Image  install.sh  Makefile  uImage  zImage
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev/arch/arm/boot$ cp uImage ~/Desktop/
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev/arch/arm/boot$ ls
bootp  compressed  dts  Image  install.sh  Makefile  uImage  zImage
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev/arch/arm/boot$
```

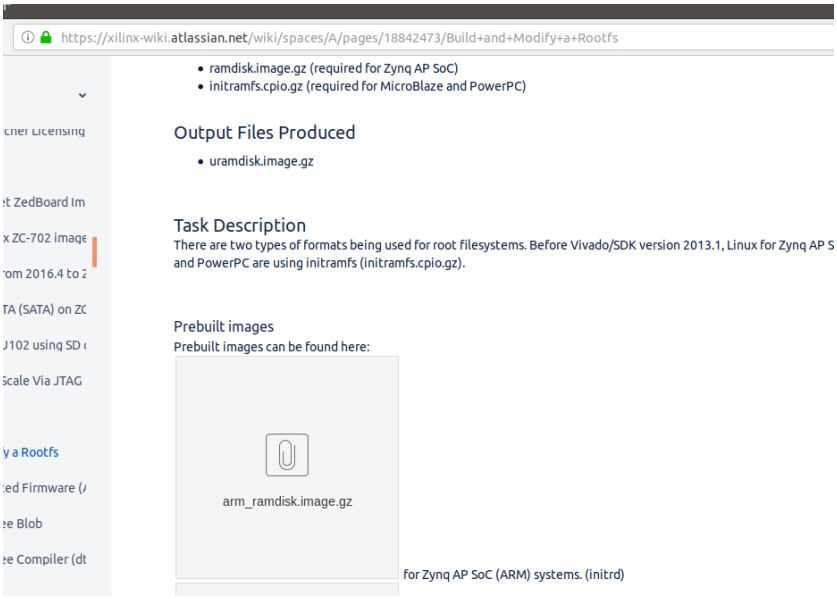
uImage file is ready to be copied to Boot Loader on micro SD card.

# 14 RAMDISK and Device Tree Blob (DTB File)

Follow Section 5 on page 25 of "Embedded Linux Hands-on Tutorial for the Zybo" to make a RAM file system uramdisk.image.gz and a device tree devicetree.dtb. They both reside under user's home directory.

## 14.1 Make RAMDISK

Obtain arm_ramdksi.image.gz http://www.wiki.xilinx.com/Build+and+Modify+a+Rootfs. Use this to make an uramdisk.image.gz file.




```
jianjiansong@jianjiansong-VirtualBox:~$ ls
arm_ramdisk.image.gz  devicetree.dtb  Downloads  lab11solution2018winter  Music     Public     u-boot-Digilent-Dev  Videos
Desktop               Documents       drivers    Linux-Digilent-Dev       Pictures  Templates  user_app
jianjiansong@jianjiansong-VirtualBox:~$ ./u-boot-Digilent-Dev/tools/mkimage -A arm -T ramdisk -c gzip -d arm_ramdisk.image.gz uramdisk.image.gz
Image Name:
Created:      Tue Feb  5 12:43:43 2019
Image Type:   ARM Linux RAMDisk Image (gzip compressed)
Data Size:    5309954 Bytes = 5185.50 kB = 5.06 MB
Load Address: 00000000
Entry Point:  00000000
jianjiansong@jianjiansong-VirtualBox:~$ ls
arm_ramdisk.image.gz  devicetree.dtb  Downloads  lab11solution2018winter  Music     Public     u-boot-Digilent-Dev  user_app
Desktop               Documents       drivers    Linux-Digilent-Dev       Pictures  Templates  uramdisk.image.gz    Videos
jianjiansong@jianjiansong-VirtualBox:~$
```

## 14.2 Make Device Tree Blob (DTB File)

dtc executable in Linux-Digilent-Dev/scripts/dtc/dtc needs to be made by uImage making process. Therefore, the kernel needs to be made before this device tree can be made.

The following screenshot shows how dtc executable is made from the kernel compilation process.

```
  SHIPPED scripts/dtc/dtc-parser.tab.h
  HOSTCC  scripts/dtc/dtc-lexer.lex.o
  SHIPPED scripts/dtc/dtc-parser.tab.c
  HOSTCC  scripts/dtc/dtc-parser.tab.o
  HOSTLD  scripts/dtc/dtc
  HOSTCC  scripts/genksyms/genksyms.o
  SHIPPED scripts/genksyms/parse.tab.c
  HOSTCC  scripts/genksyms/parse.tab.o
  SHIPPED scripts/genksyms/lex.lex.c
  SHIPPED scripts/genksyms/keywords.hash.c
  SHIPPED scripts/genksyms/parse.tab.h
  HOSTCC  scripts/genksyms/lex.lex.o
```

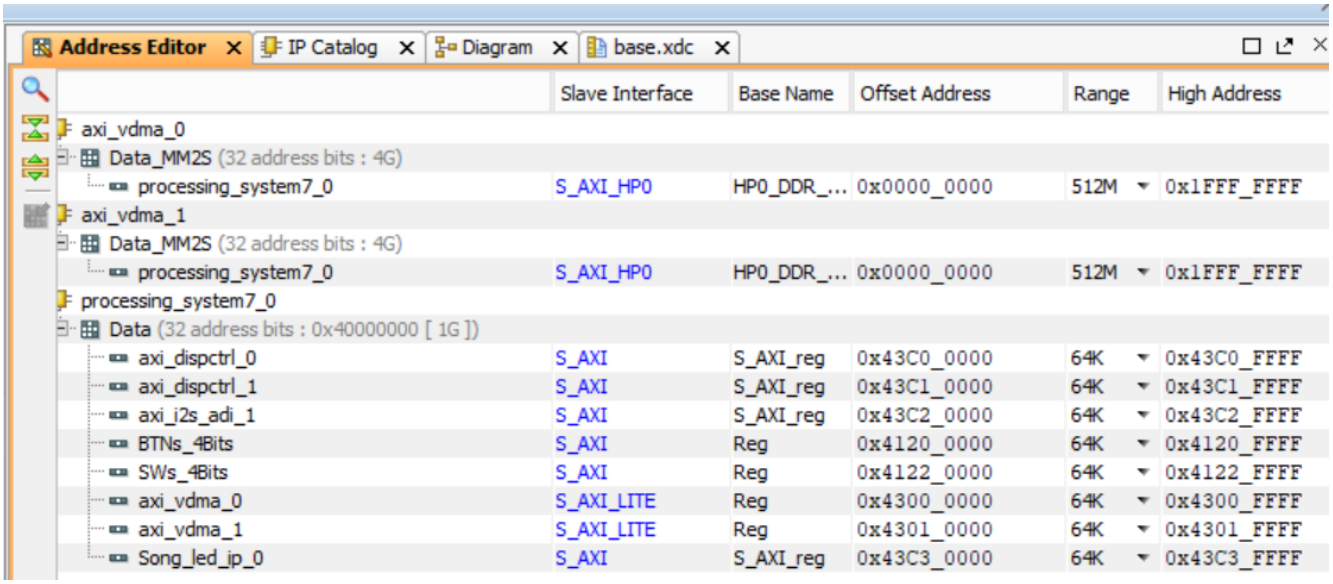
```
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev$ ls
arch      CREDITS        drivers   include  Kbuild   lib          mm              README          scripts   System.map  virt
block     crypto         firmware  init     Kconfig  MAINTAINERS  Module.symvers  REPORTING-BUGS  security  tools       vmlinux
COPYING   Documentation  fs        ipc      kernel   Makefile     net             samples         sound     usr         vmlinux.o
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev$ ./scripts/dtc/dtc -I dts -O dtb -o ../devicetree.dtb arch/arm/boot/dts/zynq-zybo.dts
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev$ cd ..
jianjiansong@jianjiansong-VirtualBox:~$ ls
arm_ramdisk.image.gz  devicetree.dtb  Downloads  lab11solution2018winter  Music     Public     u-boot-Digilent-Dev  user_app
Desktop               Documents       drivers    Linux-Digilent-Dev       Pictures  Templates  uramdisk.image.gz    Videos
jianjiansong@jianjiansong-VirtualBox:~$ cp devicetree.dtb ~/Desktop/
jianjiansong@jianjiansong-VirtualBox:~$
```

## 15 Test Your New Boot Loader

Copy your new uImage and devicetree.dtb file to your micro SD card and start your Zybo Linux kernel on your Zybo board to test the files are correct.

## 16 Modify the Device Tree to Add myled Device Node

Follow Section 6 on page 25 of "Embedded Linux Hands-on Tutorial for the Zybo" to add your LED IP device to the revised device tree. In the following address map, the LED IP is called led_ip (or your LED IP device name) and its offset address is 0x43c30000 (or the correct address for your LED IP). This needs to be added under Linux in the device tree source file zynq-zybo.dts to be compiled again to generate a new device tree blob file *.dtb. You can find the base address of your led IP device as shown below for led_ip_0 at 0x43C3_0000.




| | Slave Interface | Base Name | Offset Address | Range | High Address |
|---|---|---|---|---|---|
| axi_vdma_0 | | | | | |
| Data_MM2S (32 address bits : 4G) | | | | | |
| processing_system7_0 | S_AXI_HP0 | HP0_DDR_... | 0x0000_0000 | 512M | 0x1FFF_FFFF |
| axi_vdma_1 | | | | | |
| Data_MM2S (32 address bits : 4G) | | | | | |
| processing_system7_0 | S_AXI_HP0 | HP0_DDR_... | 0x0000_0000 | 512M | 0x1FFF_FFFF |
| processing_system7_0 | | | | | |
| Data (32 address bits : 0x40000000 [ 1G ]) | | | | | |
| axi_dispctrl_0 | S_AXI | S_AXI_reg | 0x43C0_0000 | 64K | 0x43C0_FFFF |
| axi_dispctrl_1 | S_AXI | S_AXI_reg | 0x43C1_0000 | 64K | 0x43C1_FFFF |
| axi_i2s_adi_1 | S_AXI | S_AXI_reg | 0x43C2_0000 | 64K | 0x43C2_FFFF |
| BTNs_4Bits | S_AXI | Reg | 0x4120_0000 | 64K | 0x4120_FFFF |
| SWs_4Bits | S_AXI | Reg | 0x4122_0000 | 64K | 0x4122_FFFF |
| axi_vdma_0 | S_AXI_LITE | Reg | 0x4300_0000 | 64K | 0x4300_FFFF |
| axi_vdma_1 | S_AXI_LITE | Reg | 0x4301_0000 | 64K | 0x4301_FFFF |
| Song_led_ip_0 | S_AXI | S_AXI_reg | 0x43C3_0000 | 64K | 0x43C3_FFFF |

Make a directory called drivers under your Linux home directory. Copy device tree source file zynq-zybo.dts from Linux-Digilent-Dev/arch/arm/boot/dts/zynq-zybo.dts to your driver's directory and edit this copy of zynq-zybo.dts.

```
jianjiansong@jianjiansong-VirtualBox:~/drivers$ cp ~/Linux-Digilent-Dev/arch/arm/boot/dts/zynq-zybo.dts zynq-zybo.dts
jianjiansong@jianjiansong-VirtualBox:~/drivers$ ls
zynq-zybo.dts
jianjiansong@jianjiansong-VirtualBox:~/drivers$ gedit zynq-zybo.dts
```

Here is what should be added to the device tree. After adding your LED IP to the device tree source, make a new device tree blob file. the LED IP device node name is led_ip. This is the name to be used to access this device as a process and file. There is a space before led_ip_0.

Change compatible to your name and your device name to replace the following text.

```
led_ip {
        compatible = “Jianjiansong, led_ip_0”;
        reg = <0x43c30000 0x10000>;
};
```

```
                } ;
                ps7_xadc: ps7-xadc@f8007100 {
                        clocks = <&clkc 12>;
                        compatible = "xlnx,zynq-xadc-1.00.a";
                        interrupt-parent = <&ps7_scugic_0>;
                        interrupts = <0 7 4>;
                        reg = <0xf8007100 0x20>;
                } ;
                led_ip {
                        compatible = "JianjianSong, led_ip_0";
                        reg = <0x43c30000 0x10000>;
                };
        } ;
} ;
```

Plain Text  Tab Width: 8  Ln 332, Col 23  INS

```
jianjiansong@jianjiansong-VirtualBox:~/drivers$ ls
Makefile  myled.c
jianjiansong@jianjiansong-VirtualBox:~/drivers$ cp ~/Linux-Digilent-Dev/arch/arm/boot/dts/zynq-zybo.dts zynq-zybo.dts
jianjiansong@jianjiansong-VirtualBox:~/drivers$ ls
Makefile  myled.c  zynq-zybo.dts
jianjiansong@jianjiansong-VirtualBox:~/drivers$ gedit zynq-zybo.dts
jianjiansong@jianjiansong-VirtualBox:~/drivers$ ../Linux-Digilent-Dev/scripts/dtc/dtc -I dts -O dtb -odevicetreee.dtb zynq-zybo.dts
jianjiansong@jianjiansong-VirtualBox:~/drivers$
```

The following will generate the same device tree.

```
jianjiansong@jianjiansong-VirtualBox:~/drivers$ ls
Makefile  myled.c  zynq-zybo.dts
jianjiansong@jianjiansong-VirtualBox:~/drivers$ gedit zynq-zybo.dts
jianjiansong@jianjiansong-VirtualBox:~/drivers$ ../Linux-Digilent-Dev/scripts/dtc/dtc -I dts -O dtb -o devicetree.dtb zynq-zybo.dts
jianjiansong@jianjiansong-VirtualBox:~/drivers$ ls
devicetree.dtb  Makefile  myled.c  zynq-zybo.dts
jianjiansong@jianjiansong-VirtualBox:~/drivers$
```

Copy this device tree to your micro SD card for Linux boot loader to be used to start your LED IP device under Linux kernel.

## 17 Create a Driver for myled IP

Following Sections 6 and 7 of “Embedded Linux® Hands-on Tutorial for the ZYBO™” to create a driver for myled IP under Linux kernel.

### 17.1 The driver file myled.c

Get myled.c file from Moodle and store it in ~/drivers folder. Revise myled.c to add your LED IP device as shown on page 29 of Sections 6 of “Embedded Linux® Hands-on Tutorial for the ZYBO™”. DRIVER_NAME in myled.c should be the device name in the device tree: led_ip. Device match table should have the compatible as in the device tree: {.compatible = “Jianjian Song, led_ip_0”} (There is a space before led_ip_0). Both name and compatible must be exactly the same. Otherwise, the driver cannot be inserted into kernel correctly. Here is an example when the driver was not inserted successfully because the compatible texts in the device tree and driver were different.

```
zynq> cd /mnt/
zynq> ls
BOOT.bin                    myled.ko
System Volume Information   uImage
devicetree.dtb              uramdisk.image.gz
led_blink
zynq> insmod myled.ko
zynq> _
```

Here is when the driver was inserted correctly.

```
zynq> cd /mnt/
zynq> ls
BOOT.bin                    myled.ko
System Volume Information   uImage
devicetree.dtb              uramdisk.image.gz
led_blink
zynq> insmod myled.ko
led_ip probed at VA 0x608e0000
zynq> _
```

```
#include <linux/platform_device.h>  /* Needed for Platform Driver
Functions */
#include <linux/slab.h>

/* Define Driver Name as the same as in the device tree*/
#define DRIVER_NAME "led_ip"
```

```
/* device match table to match with device node in device tree */
static const struct of_device_id myled_of_match[] = {
    {.compatible = "JianjianSong, led_ip_0"},
    {},
};

MODULE_DEVICE_TABLE(of, myled_of_match);

/* platform driver structure for myled driver */
static struct platform_driver myled_driver = {
    .driver = {
            .name = DRIVER_NAME,
            .owner = THIS_MODULE,
            .of_match_table = myled_of_match},
    .probe = myled_probe,
    .remove = myled_remove,
    .shutdown = myled_shutdown
};

/* Register myled platform driver */
module_platform_driver(myled_driver);

/* Module Informations */
MODULE_AUTHOR("Digilent, Inc.");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION(DRIVER_NAME ": MYLED driver (Simple Version)");
MODULE_ALIAS(DRIVER_NAME);
```

Create a Makefile with: gedit Makefile. Compile myled.c with gedit.

Makefile ~/drivers

```
obj-m   := myled.o

all:
        make -C ../Linux-Digilent-Dev/ M=$(PWD) modules

clear:
        make -C ../Linux-Digilent-Dev/ M=$(PWD) clean
```

Run the following command to make the driver from myled.c: make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi-. It will create myled.ko file as the Linux kernel driver for your LED IP device.

```
jianjiansong@jianjiansong-VirtualBox:~/drivers$ ls
devicetreee.dtb  Makefile  myled.c  zynq-zybo.dts
jianjiansong@jianjiansong-VirtualBox:~/drivers$ gedit Makefile
jianjiansong@jianjiansong-VirtualBox:~/drivers$ make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi-
make -C ../Linux-Digilent-Dev/ M=/home/jianjiansong/drivers modules
make[1]: Entering directory '/home/jianjiansong/Linux-Digilent-Dev'
  CC [M]  /home/jianjiansong/drivers/myled.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/jianjiansong/drivers/myled.mod.o
  LD [M]  /home/jianjiansong/drivers/myled.ko
make[1]: Leaving directory '/home/jianjiansong/Linux-Digilent-Dev'
jianjiansong@jianjiansong-VirtualBox:~/drivers$ ls
devicetreee.dtb  Makefile  modules.order  Module.symvers  myled.c  myled.ko  myled.mod.c  myled.mod.o  myled.o  zynq-zybo.dts
jianjiansong@jianjiansong-VirtualBox:~/drivers$
```

Copy myled.ko file and new devicetree.dtb, device tree blot to our micro SD card for boot loader. Start Linux kernel on your Zybo with these new files. Once your Linux kernel has started, you can find your micro SD card as a device with command “ls /dev”, which will show some devices as follows. The micro SD card device is called mmcplk0p1.

```
zynq-ocm f800c000.ps7-ocmc:
mem                 snd                 tty34               tty62 to ur

mmcblk0p1           tty1                tty38               tty9
mmcblk0p2           tty10               tty39               ttyPS0
network_latency     tty11               tty4                urandom
network_throughput  tty12               tty40               vcs
```

To mount the SD card, run command: mount /dev/mmcblk0p1 /mnt/, where /mnt/ is the directory for SD card. You will not need to mount your SD card if your SD card has only one partition. The kernel will mount it to /mnt/ automatically. If your SD card has two partitions: mmcblk0p1 and mmcblkp2 as shown above, you will need to mount it.

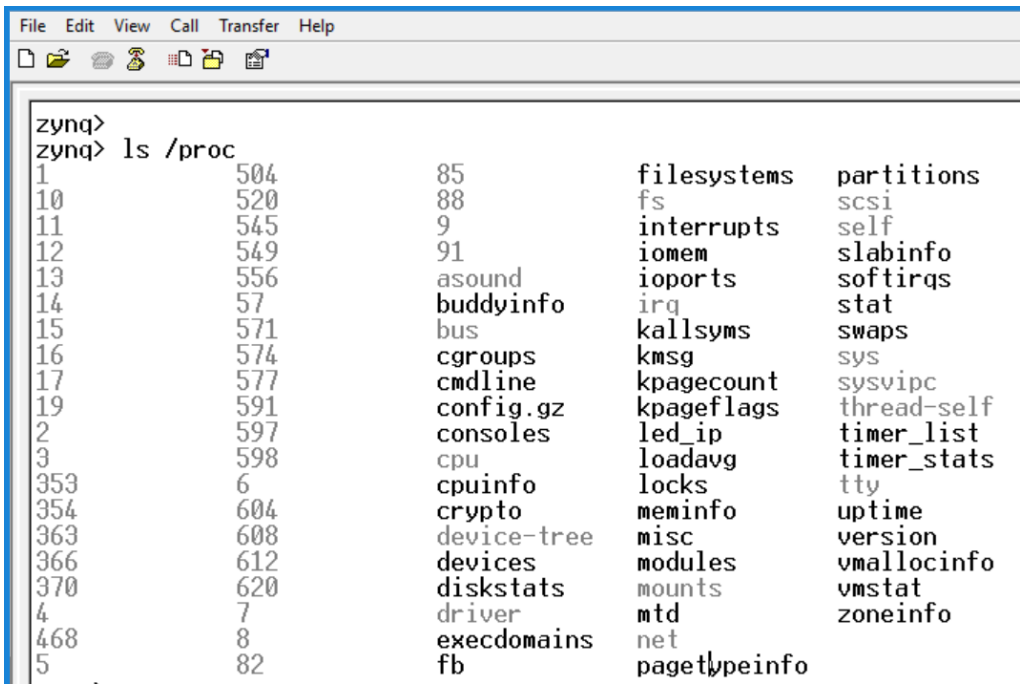
Go to /mnt/ and run “insert module into kernel”: insmod myled.ko.

```
zynq> mount /dev/mmcblk0p1 /mnt/
FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt.
zynq> cd /mnt/
zynq> ls
BOOT.bin                    myled.ko
System Volume Information   uImage
devicetree.dtb              uramdisk.image.gz
zynq> insmod myled.ko
myled probed at VA 0x608e0000
zynq> _
```

If you do not see myled probed at VA 0608e0000 or something similar and only see the following, check to make sure the device names and compatible texts for both your device tree and your driver myled.c are exactly the same.

```
zynq> cd /mnt/
zynq> ls
BOOT.bin                    myled.ko
System Volume Information   uImage
devicetree.dtb              uramdisk.image.gz
led_blink
zynq> insmod myled.ko
zynq> _
```

Run “ls /proc” to see led_ip driver process has been started.




You can then turn on and off the four LEDs by sending four-bit values to them as follows.

```
4               8               execdomains     net
468             82              fb              pagetypeinfo
5               85              filesystems     partitions
504             88              fs              scsi
zynq> echo 0x0f > /proc/led_ip
zynq> echo 0x00 > /proc/led_ip
zynq> _
```

## 17.2 Compile Application file led_blink.c

Following Section 7 on page 36 of “Embedded Linux® Hands-on Tutorial for the ZYBO™” to write a user application to make the LEDs blink under Linux kernel. Download led_blink.c file from Moodle. Change the process to the correct device name in the device tree such as /proc/led_ip.

led_blink.c
~/user_app

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

int main() {
        FILE* fp;
        while(1) {
                fp = fopen("/proc/led_ip", "w");
                if(fp == NULL) {
                        printf("Cannot open /proc/myled for write\n");
                        return -1;
                }

                fputs("0x0F\n", fp);
                fclose(fp);
                sleep(1);
                fp = fopen("/proc/led_ip", "w");
                if(fp == NULL) {
                        printf("Cannot open /proc/myled for write\n");
                        return -1;
                }

                fputs("0x00\n", fp);
                fclose(fp);
                sleep(1);
        }
        return 0;
}
```

C  Tab Width: 8  Ln 17, Col 41  INS

Make a directory user_app under your home directory: mkdir user_app. Copy led_blink.c to this directory. Create a Makefile for led_blink.c. run "make" to create led_blink.bin file. Copy this file to your micro SD card for Linux boot loader.

Makefile
~/user_app

```
CC      =       arm-xilinx-linux-gnueabi-gcc
CFLAGS  =       -g

all:    led_blink

led_blink:      led_blink.o
                $(CC) $(CFLAGS) $^ -o $@

clean:
        rm -rfv *.o
        rm -rfv led_blink

.PHONY: clean
```

```
jianjiansong@jianjiansong-VirtualBox:~/user_app$ ls
led_blink.c  Makefile
jianjiansong@jianjiansong-VirtualBox:~/user_app$ gedit Makefile
jianjiansong@jianjiansong-VirtualBox:~/user_app$ make
arm-xilinx-linux-gnueabi-gcc -g   -c -o led_blink.o led_blink.c
arm-xilinx-linux-gnueabi-gcc -g led_blink.o -o led_blink
jianjiansong@jianjiansong-VirtualBox:~/user_app$ ls
led_blink  led_blink.c  led_blink.o  Makefile
jianjiansong@jianjiansong-VirtualBox:~/user_app$ file led_blink
led_blink: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib
/ld-linux.so.3, for GNU/Linux 2.6.16, with debug_info, not stripped
jianjiansong@jianjiansong-VirtualBox:~/user_app$
```

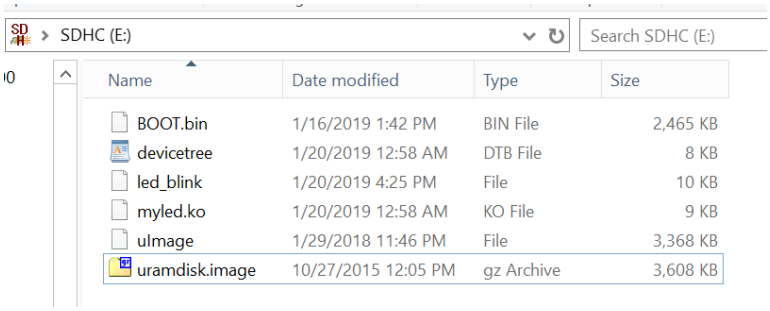
### Boot from SD Card Partition

To boot a filesystem loaded on an SD card requires at least two partitions be present on the SD card. The first partition of the SD card should be formatted into FAT to hold design files (`BOOT.BIN`), the DTB file (`devicetree.dtb`) and the kernel image (`zImage`). Format the second partition on the SD into an `ext` file system (ext4 is recommended) to host the root file system. Most Linux distributions provide tools like `parted` and `fdisk` to create a partition table on the SD card. Refer to *Getting Started with Embedded Linux* found at the Embedded Linux page on the Digilent website for step-by-step instructions on how to partition an SD card to host the root file system.

## 18 To Make a Boot Loader with led_blink.bin

Four files are needed to make a boot-up system to start Linux: BOOT.BIN, a Linux kernel image file uImage, a device tree blob devicetree.dtb (DTB file) and a file system uramdisk.image.gz. Copy these files to your FAT partition of your micro SD card.

Also copy myled.ko and led_blink binary to your micro SD card. Try to start Linux from this SD card. You can now start your Linux kernel and start led_blink to see the four LEDs on your Zybo flashing.




'./" means to run a code from the current directory.

```
zynq> mount /dev/mmcblk0p1 /mnt/
FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt.
zynq> cd /mnt/
zynq> ls
BOOT.bin                    myled.ko
System Volume Information   uImage
devicetree.dtb              uramdisk.image.gz
led_blink
zynq> insmod myled.ko
myled probed at VA 0x608e0000
zynq> ./led_blink
```

## 19 Commands with Driver Modules and Device Tree

lsmod to shows which loadable kernel modules are currently loaded.

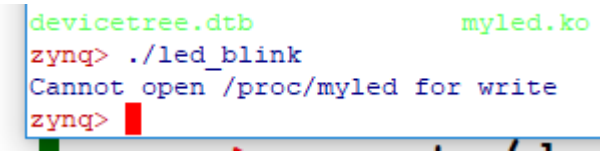
modprobe to add and remove modules from the Linux Kernel.

modinfo to show information about a Linux Kernel module.

## 20 Possible Errors

### 20.1 Cannot Open /proc/myled

Check if the device is mounted by ls /proc. Check of the device name is the same device node name in devicetree and device name in myled.c driver and process name in led_blink.c.



```
zynq> mount /dev/mmcblk0p1 /mnt/
FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.
zynq> cd /mnt/
zynq> ls
BOOT.bin                    myled.ko
System Volume Information   uImage
devicetree.dtb              uramdisk.image.gz
led_blink
zynq> insmod myled.ko
myled probed at VA 0x608e0000
zynq> ls /proc
1             5             81            filesystems   partitions
10            504           83            fs            scsi
103           520           86            interrupts    self
11            545           9             iomem         slabinfo
12            549           asound        ioports       softirqs
13            556           buddyinfo     irq           stat
14            569           bus           kallsyms      swaps
15            57            cgroups       kmsg          sys
16            571           cmdline       kpagecount    sysvipc
17            575           config.gz     kpageflags    thread-self
19            591           consoles      loadavg       timer_list
2             597           cpu           locks         timer_stats
3             598           cpuinfo       meminfo       tty
353           6             crypto        misc          uptime
354           604           device-tree   modules       version
367           608           devices       mounts        vmallocinfo
369           610           diskstats     mtd           vmstat
370           615           driver        myled         zoneinfo
4             7             execdomains   net
468           8             fb            pagetypeinfo
zynq>
```

## 20.2 File format not recognized

Here is a problem and solution from Karl Reese.
I was working on Lab 11 and ran into a strange problem while using the provided Makefile for the user_app led_blink build.
When compiling the led_blink.o object into a the led_blink binary, it threw an error from libc.so.6, saying “file format not recognized: treating as linker script” and then “syntax error”.
Upon examining /opt/Xilinx/SDK/2014.4/gnu/arm/lin/arm-xilinx-linux-gnueabi/libc/lib, I found that libc.so.6 is supposed to link to libc-2.18.so. However, as extracted from the zip file, it only holds the text ‘libc-2.18.so’ – it is not an actual symlink.
This problem also exists for ld-linux.so.3.
I was able to get the make command to finish by running the following commands:

```
cd /opt/Xilinx/SDK/2014.4/gnu/arm/lin/arm-xilinx-linux-
gnueabi/libc/lib
rm libc.so.6
ln -s libc-2.18.so libc.so.6
rm ld-linux.so.3
ln -s ld-2.18.so ld-linux.so.3
```

## 20.3 cc1: error: code model kernel does not support PIC mode

```
jianjiansong@jianjiansong-VirtualBox:~/drivers$ ls
Makefile  myled.c  myled_dallas.c  zynq-zybo.dts
jianjiansong@jianjiansong-VirtualBox:~/drivers$ make
make -C ../Linux-Digilent-Dev/ M=/home/jianjiansong/drivers modules
make[1]: Entering directory '/home/jianjiansong/Linux-Digilent-Dev'
  CC [M]  /home/jianjiansong/drivers/myled.o
cc1: error: code model kernel does not support PIC mode
scripts/Makefile.build:263: recipe for target '/home/jianjiansong/drivers/myled.o' failed
make[2]: *** [/home/jianjiansong/drivers/myled.o] Error 1
Makefile:1381: recipe for target '_module_/home/jianjiansong/drivers' failed
make[1]: *** [_module_/home/jianjiansong/drivers] Error 2
make[1]: Leaving directory '/home/jianjiansong/Linux-Digilent-Dev'
Makefile:4: recipe for target 'all' failed
make: *** [all] Error 2
jianjiansong@jianjiansong-VirtualBox:~/drivers$ ^C
jianjiansong@jianjiansong-VirtualBox:~/drivers$
```

# 21 Appendix B Basic Linux Commands

Some useful Linux commands. Prefix sudo may need to be precede a command to force root permission.

1. sudo – prefix to elevated privileges to root level.
2. man COMMAND to display manual or help page of a command.
3. ls to display files in this directory.
4. ls –a to list hidden files.
5. ls –l to display details of files.
6. ls /proc to display all processes
7. ls /dev to display all devices
8. cd directory name to go to a directory. cd ~ to go to home directory. cd / to go to root directory.
9. mkdir name and rmdir name to create or remove a directory.
10. rm FILENAME to remove a file or directory. rm –fr to remove forcibly and recursively a directory and all files in its sub directory.
11. cp to copy a file. cp file new_file to copy file to a copy called new_file.
12. mv to move a file from another directory or rename a file
13. gedit is a text edit
14. chmod to change permissions of a file. chmod –x file to make the file executable. chmod 755 file to assign root permissions to the file.
15. insmod FILENAME [options] to insert a module into the Linux kernel.
16. mount to mount a file system
17. locate to find location of a file
18. df to find disk space usage
19. zip, unzip to compress a file into a zip archive or extract a zipped file
20. tar to work with tarball archive files such as .tar, .tar.gz, .tar.bz2, etc.
21. tar –xvf to untar a tar archive.
22. source FILENAME to read and execute commands from filename.

# 22 Appendix A: How to Compile Linux Bootloader Files

The best website for Zynq Linux information is Xilinx Wiki at http://www.wiki.xilinx.com/. Under Linux Zynq AP SoC, you can find a number of Linux release images as well as how to build U-Boot, 4.0 Linux kernel, and Device-tree Generator.

# 23 References

1. http://www.wiki.xilinx.com/Zynq+2015.4+Release.
2. Embedded Linux® Hands-on Tutorial for the ZYBO™ Revised July 17, 2014, Digilent Inc.
3. Embedded Linux Development Guide Revision: January 14, 2013, Digilent Inc.
4. Getting Started with Embedded Linux – ZedBoard Revision: January 13, 2013, Digilent Inc.
5. ZYBO Reference Manual Revised February 14, 2014 for the ZYBO rev. B, Digilent Inc.
6. Zybo_base_design.zip from https://reference.digilentinc.com/zybo:zybo.
7. u-boot-Digilent-Dev.git at https://github.com/DigilentInc/u-boot-Digilent-Dev.git.
8. Linux-Digilent-Dev.git at https://github.com/DigilentInc/Linux-Digilent-Dev.git.
9. Ram_ramdisk_image.gz at http://www.wiki.xilinx.com/Build+and+Modify+a+Rootfs.