Name: _____ ID: _____ Start Date: Friday, August 20, 2021
Due Date: Sunday, August 22, 2021

Software and Hardware Co-Design with Zybo, Summer 2021 HUST
Lab #10 Embedded Linux Kernel on Zybo
Summary

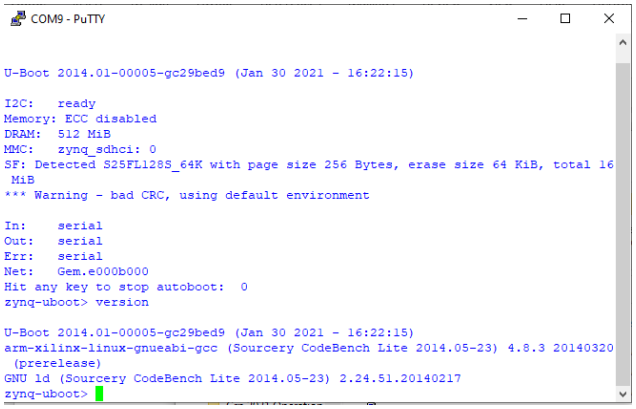
This is an individual lab. Each student must perform it and demonstrate this lab to obtain credit for it. Your LED IP must be named to be your last name with your first name initial such as SongJJ. No Lab report is required. Late lab submission will be accepted with a grade reduction of 10% for each day that it is late.

1 Part 1: to Create and Test the New u-boot File

[1] Compile a u-boot.elf file with Xilinx Linux arm compiler on VirtualBox Linux environment.

[2] Create a BOOT.BIN file with any Vivado block design such as Lab 2 to include zynq FSBL.elf file, system_wrapper.bit file and u-boot.elf file.

[3] Demonstrate your u-boot date is when you created your u-boot as shown below when the booting is stopped by hitting any key within the first 2 seconds. Type version to show your u-boot version and date, etc.



2 Part 2: New kernel, new device tree, device driver and blinking LED

[4] Create Vivado 2016.2 version of Zybo Base System Design Block Design. Add a new LED IP to it. Name your LED IP with your last name and your first name initial, e.g., SongJJ_led_ip. . Generate the bit stream file with Windows-based Vivado version 2016.2.

[5] Compile a Linux kernel image, uImage, on VirtualBox Linux environment.

[6] Compile a disk system in RAM, uramdisk.image.gz, on VirtualBox Linux environment.

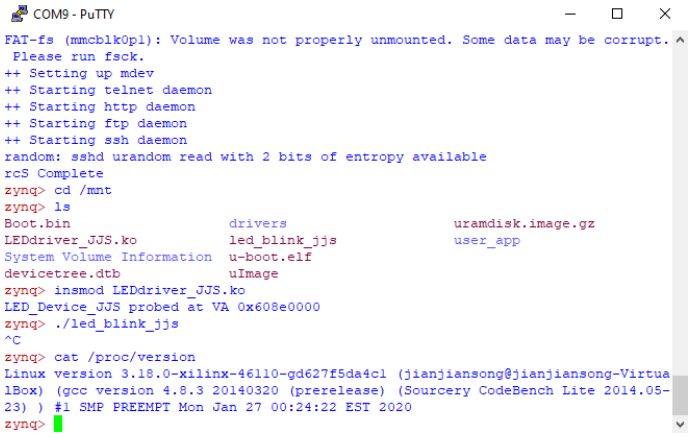
[7] Compile a device tree blot file, devicetree.dtb file on VirtualBox Linux environment.

[8] Modify the device tree, write your LED driver, named LEDdriver_JJS, where JJS is name initials, based on myled.c and write the LED application, named led_blink_jjs, where jjs is your name initials, with Linux on Zybo.

[9] Boot Linux with the led application from microsSD on Zybo with six files: BOOT.bin, devicetree.dtb, uImage, urandisk.image.gz, led_blink_jjs.bin, and LEDdriver_JJS.ko.

[10] Flash the four LEDs and

[11] check kernel creator and date with $cat /proc/version.



Name: _____ ID: _____ Start Date: Friday, August 20, 2021
Due Date: Sunday, August 22, 2021

Software and Hardware Co-Design with Zybo, Summer 2021 HUST
Lab #10 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 10% for each day that it is late.

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. Oracle VM VirtulBox and Ubuntu Linux takes 20 GB of hard drive space.

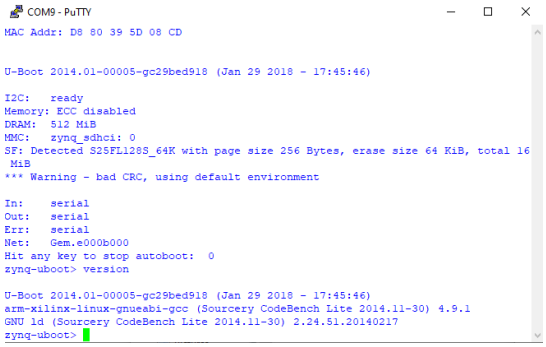
1 Objectives

1.1 Part 1: to Create and Test the New u-boot File

[1] Compile a u-boot.elf file with Xilinx Linux arm compiler on VirtualBox Linux environment. (Section 2 of "Embedded Linux® Hands-on Tutorial")

[2] Create a BOOT.BIN file Windows-based Vivado version 2016.2 to include zynq FSBL.elf file, system_wrapper.bit file and u-boot.elf file. (Section 3 of "Embedded Linux® Hands-on Tutorial")

[3] Check u-boot version with >version.



1.2 Part 2: New kernel, new device tree, device driver and blinking LED with your initials

[1] Create Vivado 2016.2 version of Zybo Base System Design Block Design and bit stream file with Windows-based Vivado version 2016.2. (Section 1 of "Embedded Linux® Hands-on Tutorial")

[2] Compile a Linux kernel image, uImage, on VirtualBox Linux environment. (Section 4 of "Embedded Linux® Hands-on Tutorial")

[3] Compile a disk system in RAM, uramdisk.image.gz, on VirtualBox Linux environment. (Section 5 of "Embedded Linux® Hands-on Tutorial")

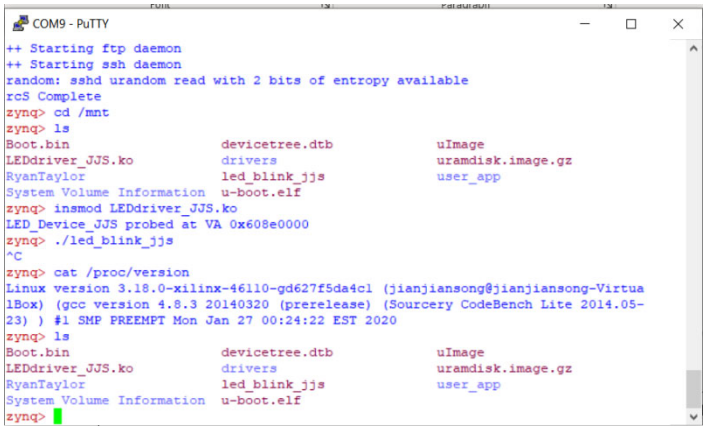
[4] Compile a device tree blot file, devicetree.dtb file on VirtualBox Linux environment. (Section 5 of "Embedded Linux® Hands-on Tutorial")

[5] 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")

[6] Modify the device tree, write your LED driver, named LEDdriver_JJS, where JJS is name initials, based on myled.c and write the LED application, named led_blink_jjs, where jjs is your name initials, with Linux on Zybo.

[7] Boot Linux with the led application from microsSD on Zybo with six files: BOOT.bin, devicetree.dtb, uImage, urandisk.image.gz, led_blink_jjs.bin, and LEDdriver_JJS.ko.

[8] Flash the four LEDs and check kernel date with $cat /proc/version.



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.

3 Demonstration and Report

Demonstrate your Linux boot loader and led application by the deadlines to get credit for this lab. No report is required.

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.

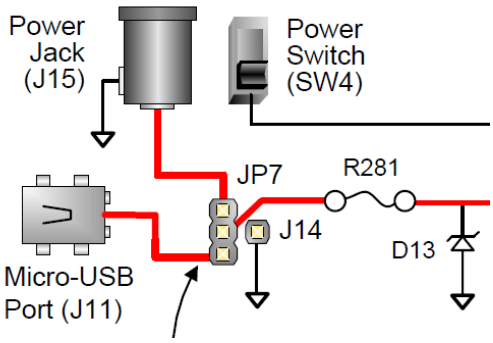
5 Learn how to Boot Linux from a micro SD Card

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 FAT32 New Simple Volume on the SD card with all default settings.

You should not need to power your Zybo board with the wall plug power. If you cannot start your microSD card reader due to power requirement, you can use a 5V wall plug or USB power to drive the SD card reader.

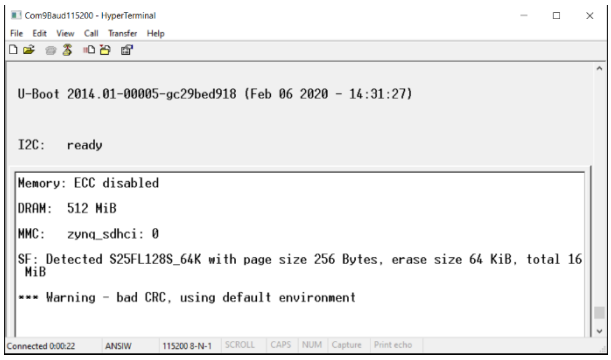


5.2 Copy Boot Loader files and Test SD card

Copy boot loaders and boot image files from Moodle->Week 7-> 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.

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.

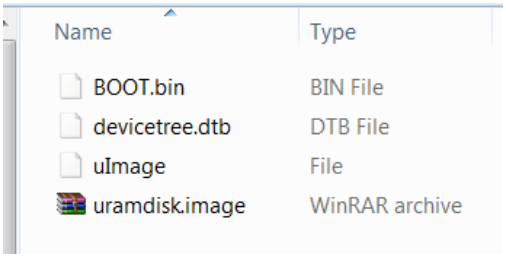


3.1 microSD Boot Mode

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 Putty terminal. To check your U-Boot information, you can hit any key within 3 seconds of booting to start autoboot. Here is the terminal display from the example boot files.

```
U-Boot 2014.01-00005-gc29bed9 (Oct 26 2015 - 19:45:43)

I2C:   ready
Memory: ECC disabled
DRAM:  512 MiB
MMC:   zynq_sdhci: 0
SF: Detected S25FL128S_64K with page size 256 Bytes, erase size 64 KiB, total 16 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   Gem.e000b000
Hit any key to stop autoboot:  0
zynq-uboot>
```

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.

```
COM12 - PuTTY
[    1.922120] export_store: invalid GPIO 63
/etc/init.d/rcS: line 46: can't create /sys/class/gpio/gpio63/dir[    1.928948] export_store: invalid GPIO 72
ection: nonexistent directory
[    1.938472] export_store: invalid GPIO 64
/etc/init.d/rcS: line 46: can't create /sys/class/gpio/gpio64/dir[    1.945286] export_store: invalid GPIO 73
ection: nonexistent directory
[    1.954743] export_store: invalid GPIO 65
/etc/init.d/rcS: line 46: can't create /sys/class/gpio/gpio65/dir[    1.961599] export_store: invalid GPIO 74
ection: nonexistent directory
[    1.971100] export_store: invalid GPIO 66
/etc/init.d/rcS: line 46: can't create /sys/class/gpio/gpio66/dir[    1.977930] export_store: invalid GPIO 75
ection: nonexistent directory
[    1.987435] export_store: invalid GPIO 67
/etc/init.d/rcS: line 46: can't create /sys/class/gpio/gpio67/dir[    1.994243] export_store: invalid GPIO 76
ection: nonexistent directory
[    2.003755] export_store: invalid GPIO 68
/etc/init.d/rcS: line 46: can't create /sys/class/gpio/gpio68/direction: nonexistent directory
rcS Complete
zynq>
```

You can check the dates of files on your MicroSD card as follows.

```
zynq> cd /mnt/
zynq> ls
BOOT.bin                    uImage
System Volume Information   uramdisk.image.gz
devicetree.dtb
zynq> ls -l
total 9332
-rwxr-xr-x    1 root     0          2506864 May  4  2016 BOOT.bin
drwxr-xr-x    2 root     0             4096 Feb 12  2017 System Volume Information
-rwxr-xr-x    1 root     0            11227 Oct 27  2015 devicetree.dtb
-rwxr-xr-x    1 root     0          3332120 Dec 10  2015 uImage
-rwxr-xr-x    1 root     0          3694172 Oct 26  2015 uramdisk.image.gz
zynq>
```

If you cannot find your SD card under /mnt/ try to mount it. 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.

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 Section 5 of this lab.

6 Installing VirtualBox Ubuntu Linux, System Software and Xilinx SDK

You can either install a Linux virtual machine such as Virtual Box or VMware or Linux machine.

If you want to install Oracle VM VirtualBox, you can follow "Guidelines on VirtualBox Ubuntu and Xilinx SDK2014" in pdf from the instructor to install Oracle VM VirtualBox, Ubuntu and Xilinx Linux SDK 2014.4. If you have installed Linux or VirtualBox, follow this section to install other software packages.

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.

After you have installed your Linux, you may want to sudo apt update and sudo apt upgrade to make sure you have the latest versions. However, these upgrades may trigger more upgrades of other packages. Follow the rest of this sections to install necessary software tools.

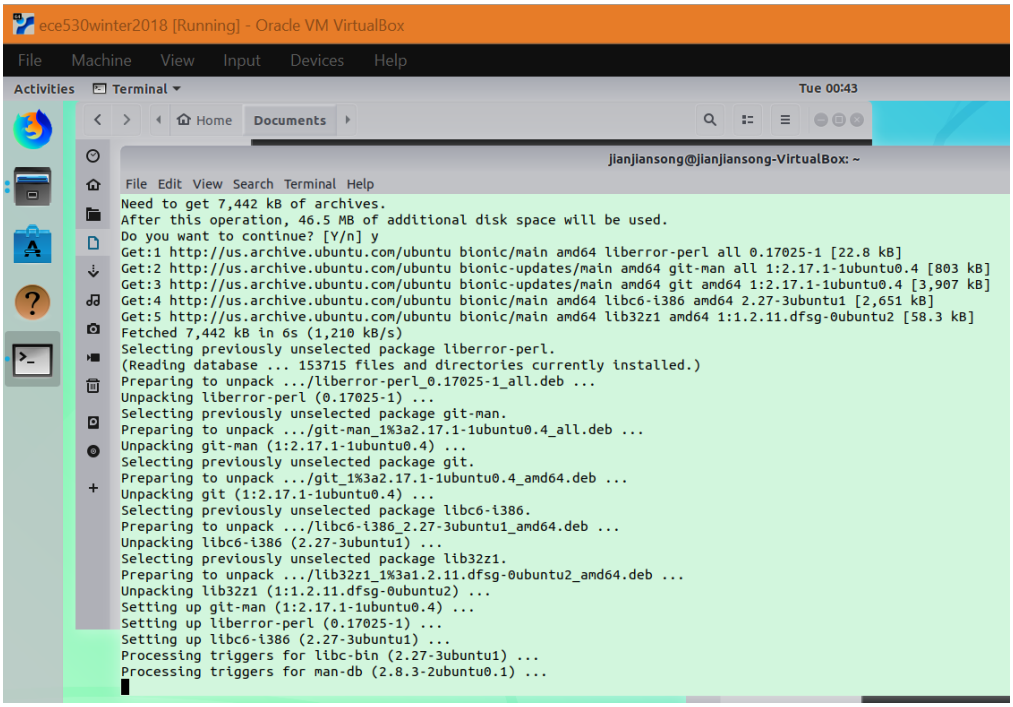
After you have used these guidelines to complete your installation, you can go to Section 7 of this lab handout to start Part 1: compiling the u-boot.

6.1 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. There is no space between “apt-get”.

```
$sudo apt-get install git lib32z1
```

```
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
```



6.2 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-get install gcc.

```
$sudo apt-get install gcc-arm-none-eabi gcc-arm-linux-gnueabi.
$sudo apt-get install lib32z1 lib32ncurses5 libbz2-1.0:i386 lib32stdc++6.
$sudo apt-get 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.

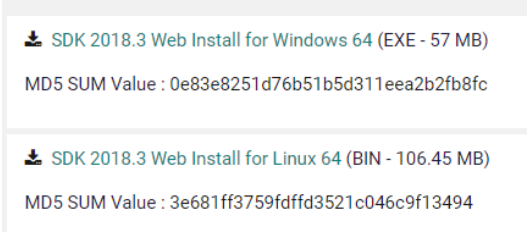
6.3 Install Xilinx Linux SDK 2014.4

Again, you do not need to follow this section if you have following "Guidelines on VirtualBox Ubuntu and Xilinx SDK rev5-1-2020" in pdf to have installed all tools.

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

XilinxSDK2014_fixed_broken_symlink.zip is available from the instructor. It contains Xilinx 2014 SDK source files for Linux. The files 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. One versoion 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.



Upload XilinxSDK2014_fixed_broken_symlink.zip to your Virtual Machine or Linux machine by default to ~/Downloads. Move the file to ~/Documents after it has been downloaded by dragging and dropping the file.

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 can type a few characters and then type Tab key to complete a directory or file name. 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.

```
$unzip ~/Documents/XilinxSDK2014_fixed_broken_symlink.zip -d ~Desktop/
```

```
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/
```

```
$sude chmod -R +x /opt/Xilinx/
```

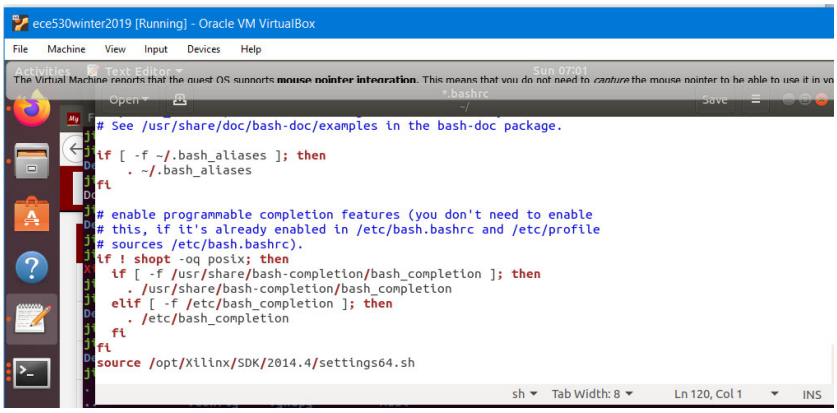
```
jianjiansong@jianjiansong-VirtualBox:~$ ls ~/Desktop/
Xilinx
jianjiansong@jianjiansong-VirtualBox:~$ sudo mv ~/Desktop/Xilinx/ /opt/
[sudo] password for jianjiansong:
jianjiansong@jianjiansong-VirtualBox:~$ sudo chmod -R +x /opt/Xilinx/
jianjiansong@jianjiansong-VirtualBox:~$ cd /
jianjiansong@jianjiansong-VirtualBox:/$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

This SDK needs to be configured by sourcing a shell command file /opt/Xilinx/SDK/2014.4/settings64.sh. This can be done automatically by adding a command in the shell script .bashrc. This script will be executed whenever a terminal or a shell is started. This script will be edited to add a source command to execute a shell script under /opt/Xilinx folder. Source command will load any functions file into the current shell script or a command prompt. Run "echo $PATH" to display the value of PATH variable before adding SDK path. You can also run "ls -a" to see .bashrc file in your home directory.

Next, type the command in your home directory. Type "$cd ~ and Enter" to get to your home directory if you are not in it. gedit is a text editor to add a source 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:~$
```

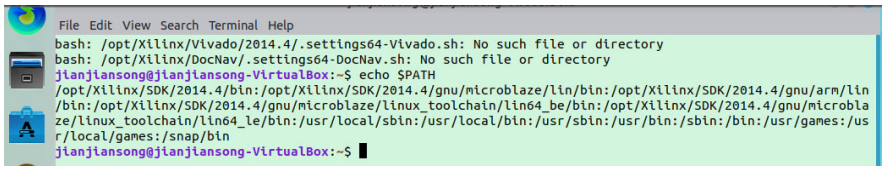
A text editor by gedit will appear and add to the bottom of the file “source /opt/Xilinx/SDK/2014.4/settings64.sh”. Save (Ctrl+s) and close the text editor. Note, in Ubuntu the “X” button to close windows is in normally in the top left corner of the window, but if it’s maximized the “X” button is hidden,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

```
$tar zxvf ~/Download/Xilinx.tar.gz -C ~/Desktop/ && mv ~/Desktop/Xilinx /opt/
```

After existing gedit and exiting the terminal, restart a terminal (by typing Ctrl+Alt+t) and type: echo $PATH to see SDK paths be added to the path variable. Ignore the bash: messages about .settings64-Vivado.sh and seetings64-DocNav.sh as these were not created.



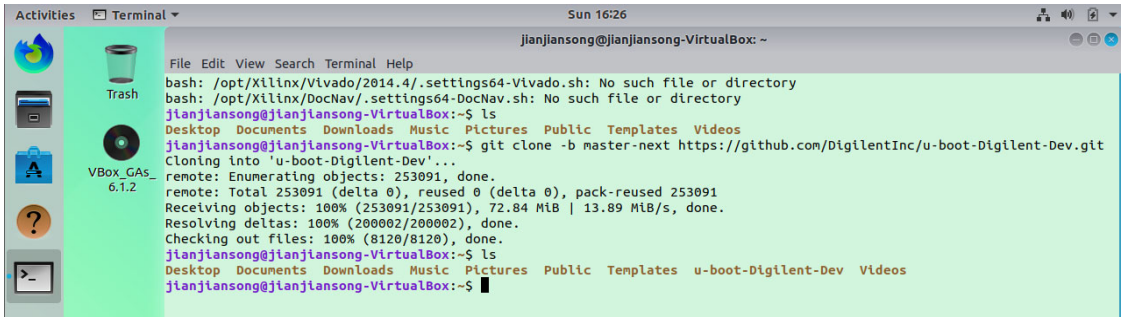
Part 1: to Create and Test the New u-boot File

1 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-Digilent-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. The command is

```
$git clone -b master-next https://github.com/DigilentInc/u-boot-Digilent-Dev.git
```

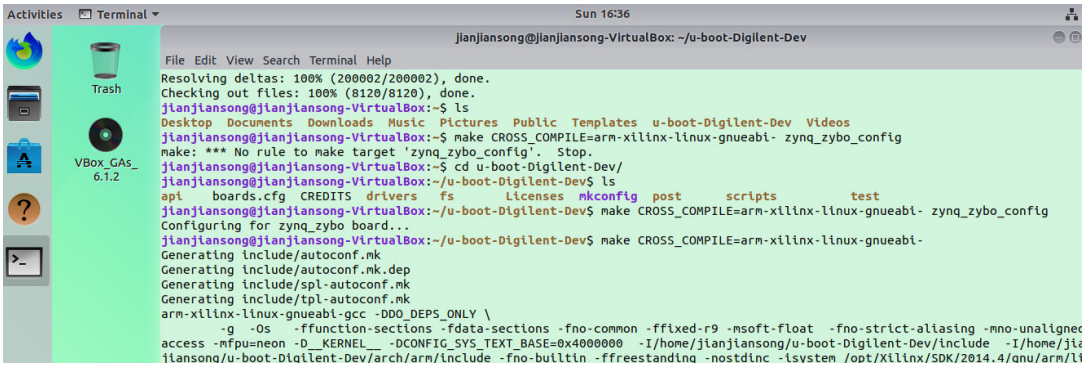


The following command will set up the correct build environment or configuration parameters. Sometimes you may have to quit your VirtualBox and start a new one for some configuration parameters to be set correctly. If you see configuration errors, restarting your VirtualBox may fix them. If make does not work, run: sudo apt install u-boot-tools or add /u-boot-Digilent.Dev/tools/ path to your environment variable PATH, i.e., PATH=$PATH:~/u-boot- Digilent.Dev/tools. Restarting your VirtualBox may help to set some configuration parameters correctly. The commands are as follows. There is a space between gneabi- and zynq_zybo_config.

```
$make CROSS_COMPILE=arm-xilinx-linux-gnueabi- zynq_zybo_config
$ make CROSS_COMPILE=arm-xilinx-linux-gnueabi-
```

```
jianjiansong@jianjiansong-VirtualBox:~/u-boot-Digilent-Dev$ ls
api    boards.cfg  CREDITS  drivers   fs       Licenses  mkconfig  post      scripts          test
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$
```

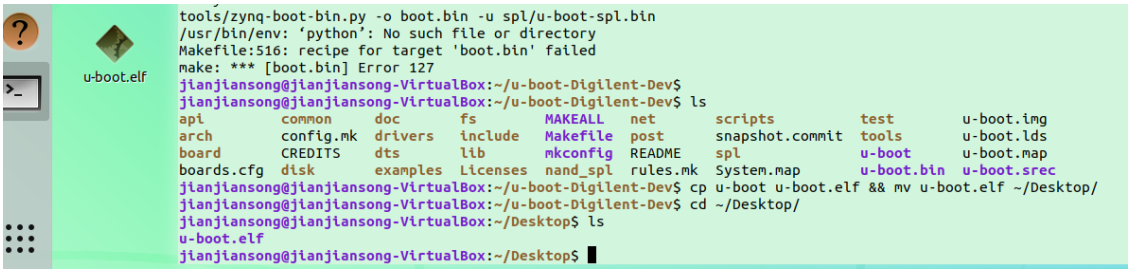
Now you can make the u-boot files by typing PgUp key to recall previous command and remove zynq_zybo_config from it and execute the new make command to make the u-boot. This command takes about four minutes to run.



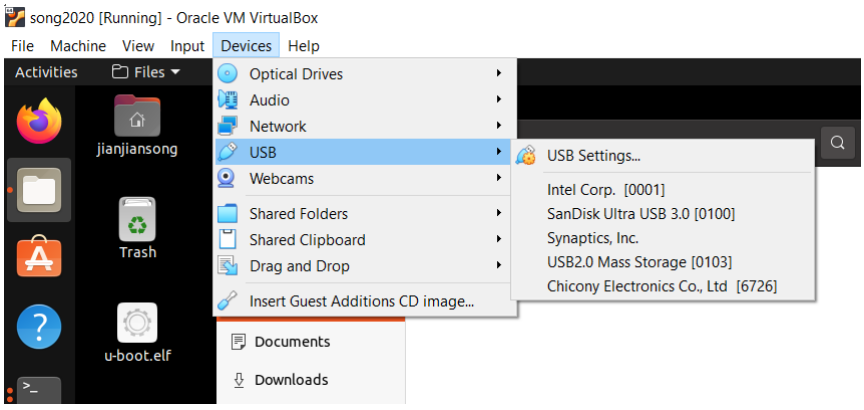
Ignore “make: ***[boot.bin] Error 127”. The compiled u-boot file is simple u-boot. It needs to be changed to u-boot.elf (Executable and Linkable File) to be included into BOOT.BIN for the boot loader.

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

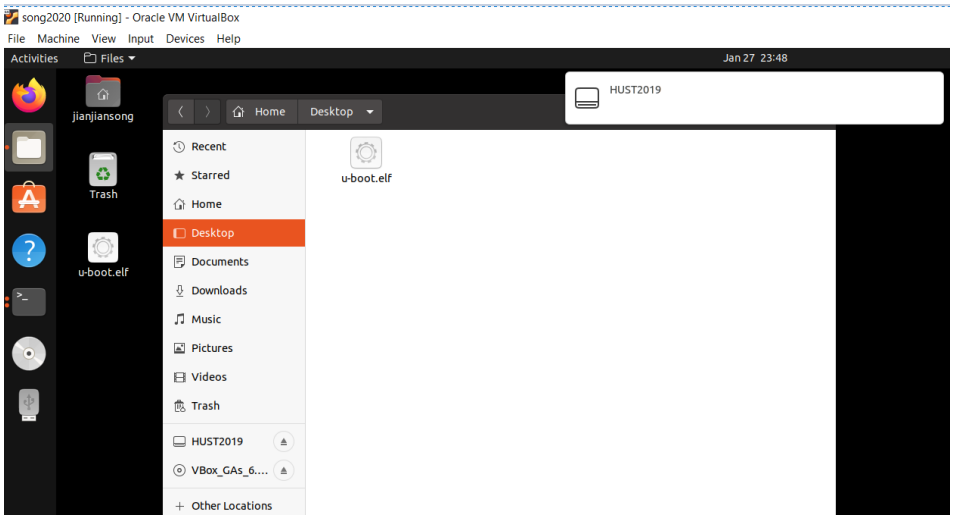
```
$cp u-boot u-boot.elf && mv u-boot.elf ~/Desktop/
```

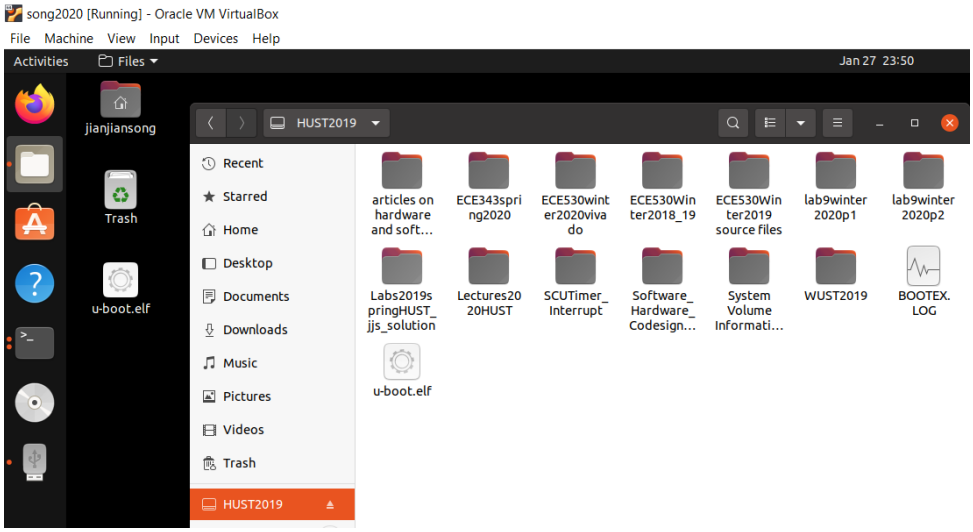


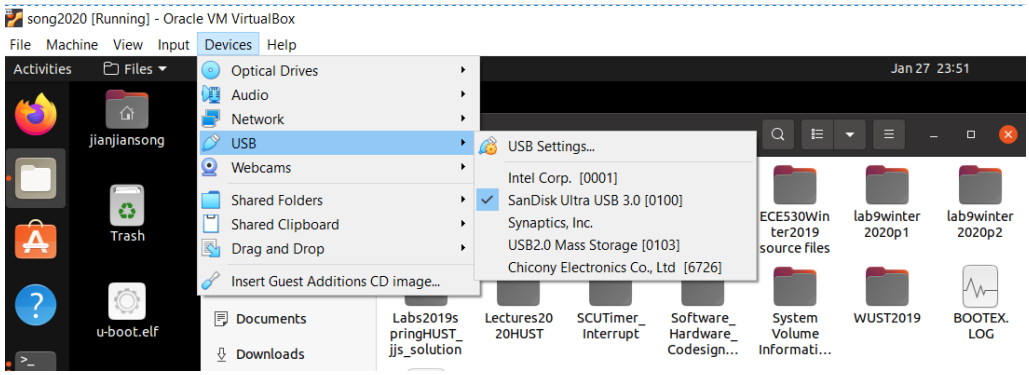
You can now copy your u-boot.elf file to a USB flash drive so that it can be accessible from your Windows. Connect a USB flash drive to your laptop. Under your VirtualBox, choose Devices->USB to see a list of USB devices. Choose the one you want to use, for example, SanDisk ultra USB 3.0[0100]. Click to open it. The actual flash drive is named HUST2019 and is available in the file system of this VirtualBox.



Double click on Files cabinet to open the list of file folders. You can see Desktop folder and HUST2019 flash drive. You can now double click to open Desktop and click and drag u-boot.elf to HUST2019 flash drive. Now you can choose Devices-USB again to release or reject this flash drive so that it can be accessed by Windows again.







2 Generate BOOT.BIN and Demonstrate on Zybo Board

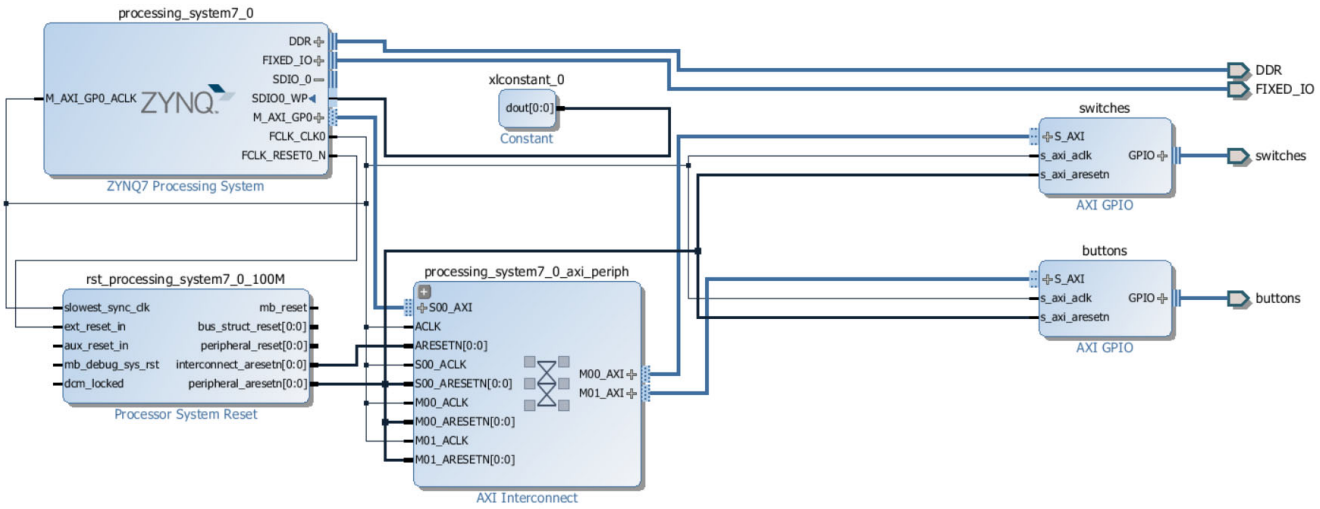
This is the final section of Part 1 of this lab. You will need to demonstrate this part by the deadline to obtain full credit for it. you can use any Vivado block design to generate this boot.bin file. However, you must turn on SD 0 in your Zynq module to make your boot.bin work. Here is a screen shot of a block design with SD 0 being turned on. Notice SD 1 should be connected to MIO pins 40 to

45 and its card detect pin is MIO pin 47. Its Write Protect is connected to EMIO, which is grounded internally by a Constant module as shown in the ZYNQ block schematic.

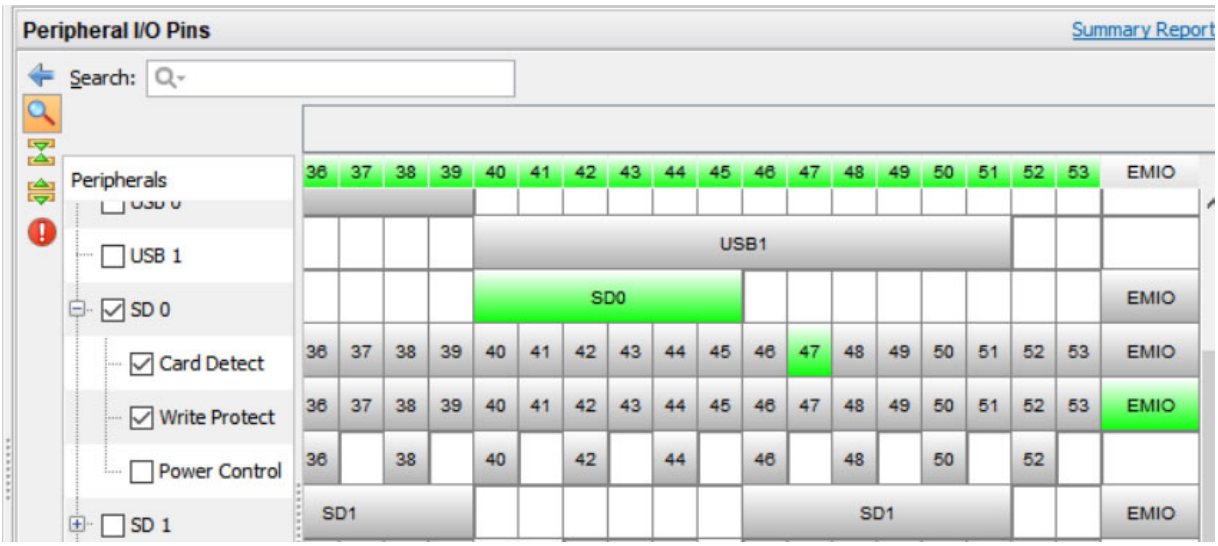
Section 3 on page 16 of "Embedded Linux Hands-on Tutorial for the Zybo" describes this process. You can use the same SDK for your Zybo base system design with the led IP to create a BOOT.BIN file. Your BOOT.BIN file should contain three files: FSBL file, *.bit file and the u-boot.elf file.

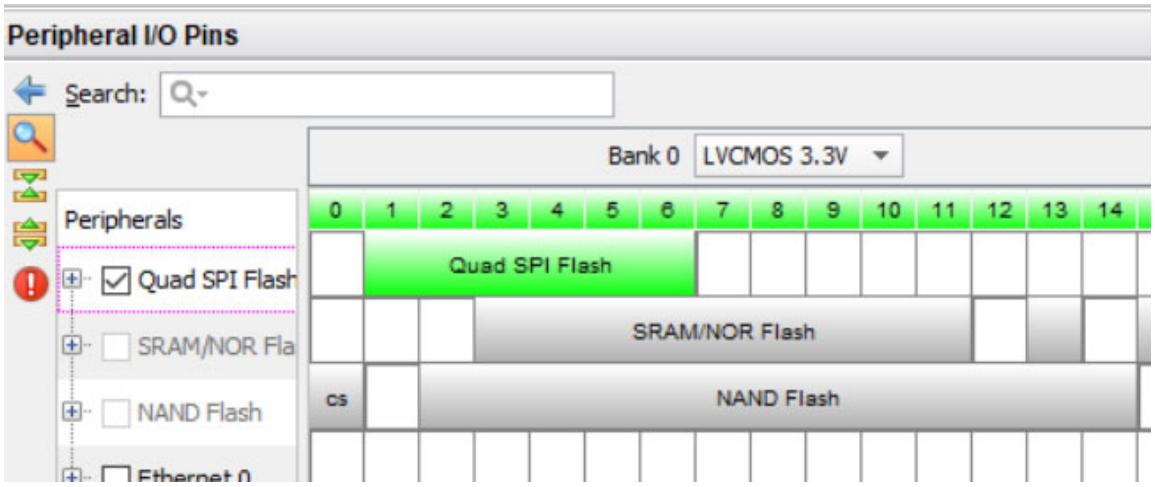
2.1 Add SD 0 and Quad SPI to a block design of yours

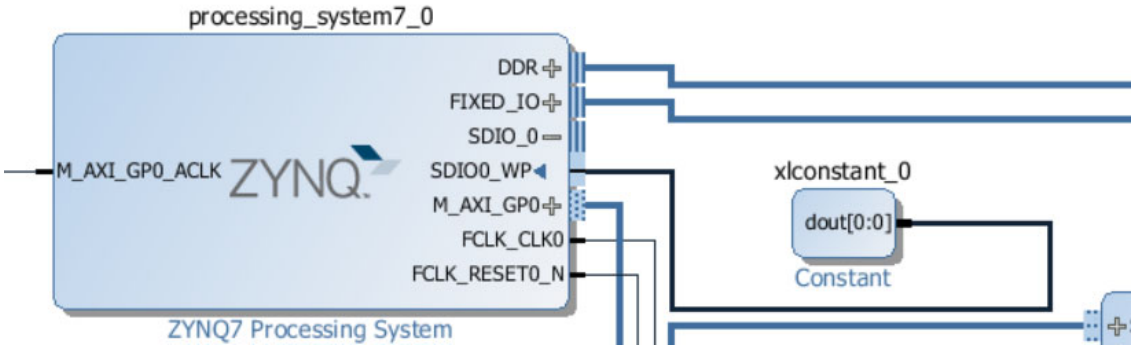
I used Lab 2 as follows to generate my boot.bin file.



Add SD 0 device so that the microSD card can be accessed by the Linux kernel. The SD device general pins are MIO pins from 40 to 45, and 47 for Card Detect. The Write Protect pin is connected to EMIO but is grounded internally with a Constant module initialized to be zero. Quad SPI device may not be needed but adding this may help you to see one less error from your u-boot,







7 MicroSD Slot

The ZYBO provides a microSD slot (J4) for non-volatile external memory storage as well as booting the Zynq. The slot is wired to Bank 1/501 MIO[40-47], including Card Detect. On the PS side peripheral SDIO 0 is mapped out to these pins and controls communication with the SD card. The pinout can be seen in Table 4**Error! Reference ource not found.**. The peripheral controller supports 1-bit and 4-bit SD transfer modes, but does not support SPI mode. Based on the Zynq TRM, SDIO host mode is the only mode supported.

Signal Name	Description	Zynq Pin	SD Slot Pin
SD_D0	Data[0]	MIO42	7
SD_D1	Data[1]	MIO43	8
SD_D2	Data[2]	MIO44	1
SD_D3	Data[3]	MIO45	2
SD_CCLK	Clock	MIO40	5
SD_CMD	Command	MIO41	3
SD_CD	Card Detect	MIO47	9

Table 4. MicroSD pinout.

2.2 Copy u-boot file

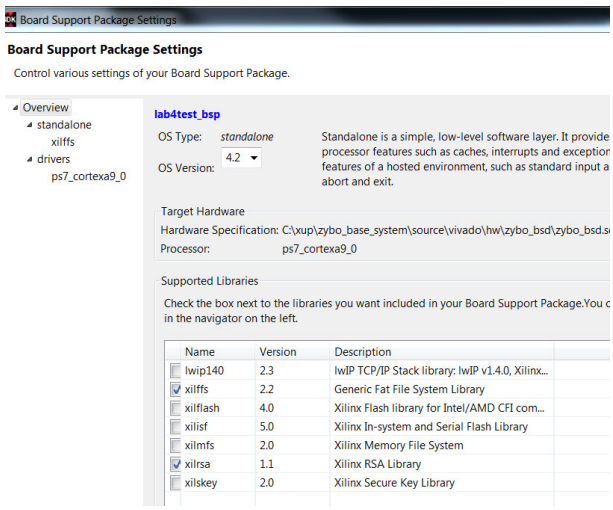
You should have created a new u-boot.elf file and saved it on your image folder or Desktop of your laptop.

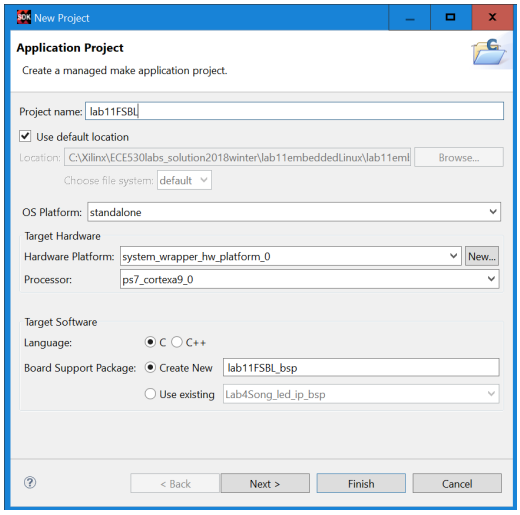
2.3 Add options to Board Support Package Settings

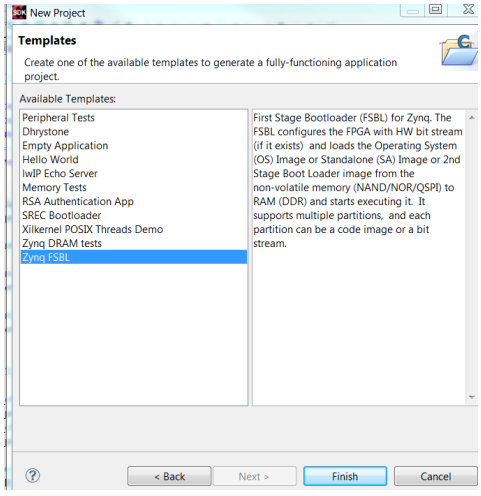
In SDK, after a project has been created and tested, right click on *.bsp and click Board Support Package Settings. Choose xilffs and xilrsa and click OK. (This is required for the next step to create the FSBL.)

2.4 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”.

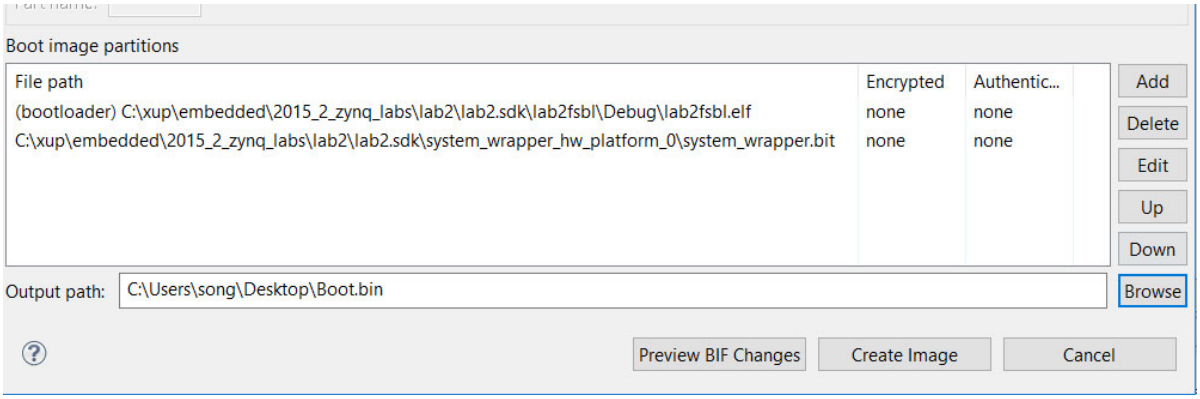




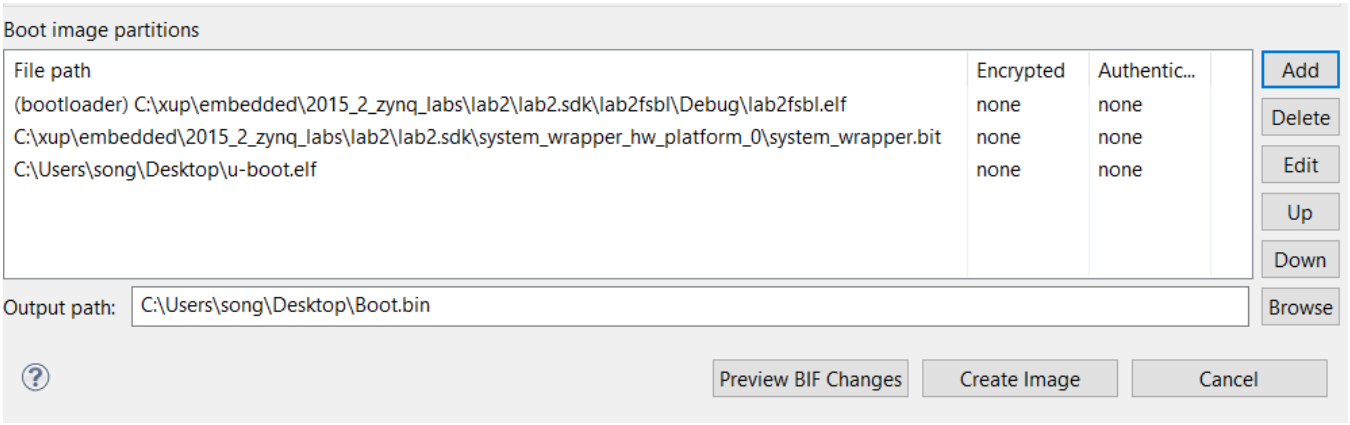


2.5 Create a BOOT.BIN file

In SDK, select Xilinx Tools->Create Zynq Boot Image. Keep the Output BIF field to be at the default location. Two of the three files for boot.bin should have been included to the boot image partitions as follows.

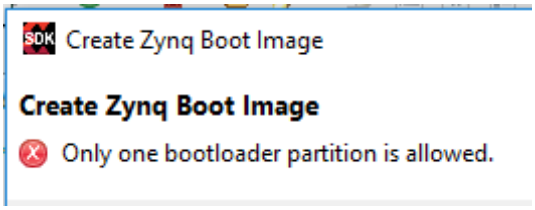


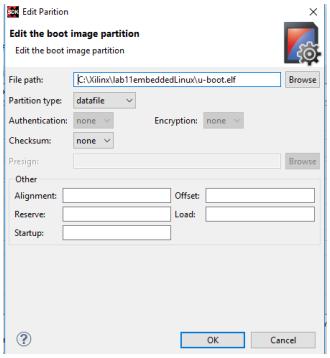
Otherwise, add your FSBL.elf file as boot loader file from the /Debug folder of your FSBL project folder the SDK folder of this lab: *.sdk\lab11FSBL\Debug. Add system_wrapper.bit file as data file to the Boot image partitions from *.sdk\system_wrapper_hw_platform_0. Add u-boot.elf file you have created as the last data file. I have chosen to store boot.bin file on the desktop of my laptop.



Click “Create image” to generate BOOT.bin file.

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.





2.6 Demonstrate your BOOT.bin file on Zybo

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. You can hit any key within 3 seconds to stop the installation so that you can see the U-Boot date and other information. This date should be

when you created your u-boot file. Notice the date must be after August 19, 2021 to show you have compiled this u-boot.

```
U-Boot 2014.01-00005-gc29bed918 (Feb 06 2020 - 14:31:27)

I2C:   ready
Memory: ECC disabled
DRAM:  512 MiB
MMC:   zynq_sdhci: 0
SF: Detected S25FL128S_64K with page size 256 Bytes, erase size 64 KiB, total 16
 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   Gem.e000b000
Hit any key to stop autoboot:  0
zynq-uboot>
```

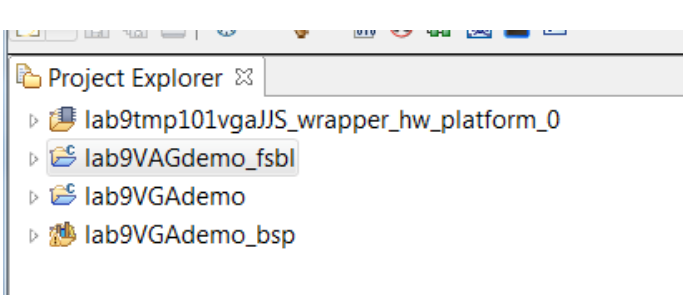
If you press PS-SRST button to reboot and complete the booting sequence to see your Linux kernel, you can run cat /proc/version to see the Linux kernel version as shown below.

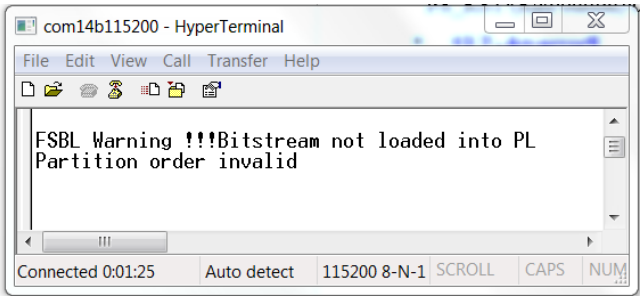
```
zynq> cat /proc/version
Linux version 3.18.0-xilinx-46110-gd627f5da4c1 (jianjiansong@jianjiansong-Virtua
lBox) (gcc version 4.8.3 20140320 (prerelease) (Sourcery CodeBench Lite 2014.05-
23) ) #1 SMP PREEMPT Mon Jan 27 00:24:22 EST 2020
zynq>
```

2.7 Possible errors: wrong file order and no SD detected

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.





Another possible error at booting time is when the u-boot could not find the SD card. The error appears at the beginning of the u-boot startup as shown below. The u-boot was looking for QSPI flash instead of SD card.

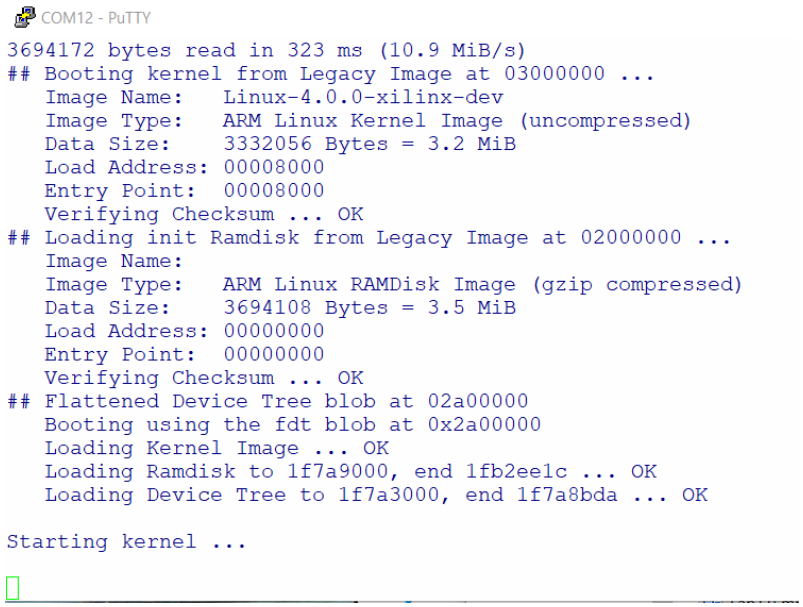
COM12 - PuTTY

```
U-Boot 2014.01-00005-gc29bed918 (Jan 28 2020 - 22:42:28)

I2C:   ready
Memory: ECC disabled
DRAM:  512 MiB
MMC:   zynq_sdhci: 0
spi_setup_slave: No QSPI device detected based on MIO settings
SF: Failed to set up slave
*** Warning - spi_flash_probe() failed, using default environment

In:    serial
Out:   serial
Err:   serial
```

The booting process was hanging after kernel was booted.



Part 2: New kernel, new device tree, device driver and blinking LED

3 Update the ZYBO Base System Design to Vivado 2016.2

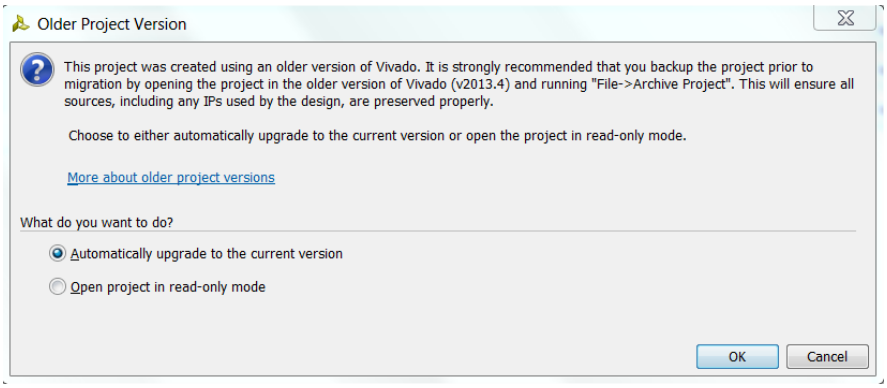
This section explains "Embedded Linux Hands-on Tutorial for the ZYBO" from Digilent to create the ZYBO Base System Design. You can follow this section only to complete this task.

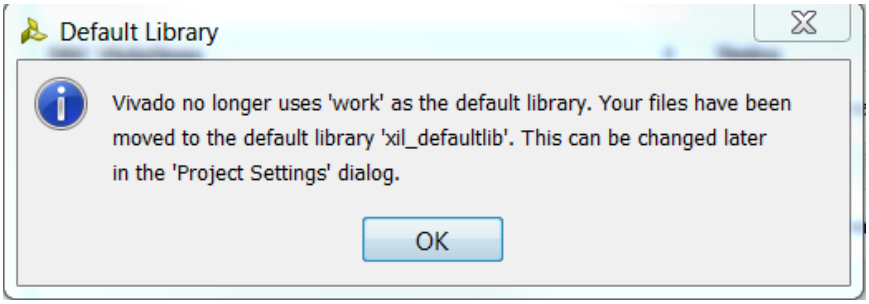
3.1 Download Zybo Base System Design zipped file

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.

3.2 **Open Vivado project file zybo_bsd**

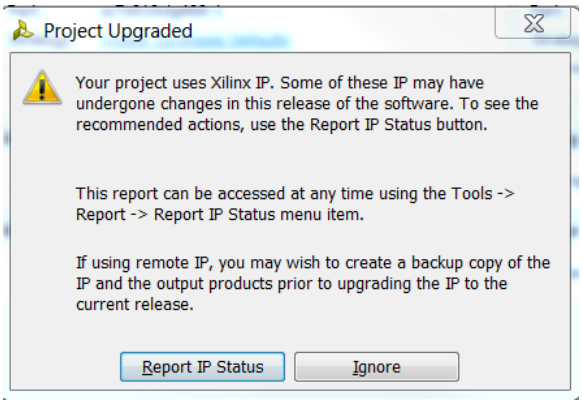
Open Vivado project file zybo_bsd at C:\xup\zybo_base_system\source\vivado\hw\zybo_bsd. And choose "Automatically upgrade to the current version". Click "OK" on the default library 'xil_defaultlib'.

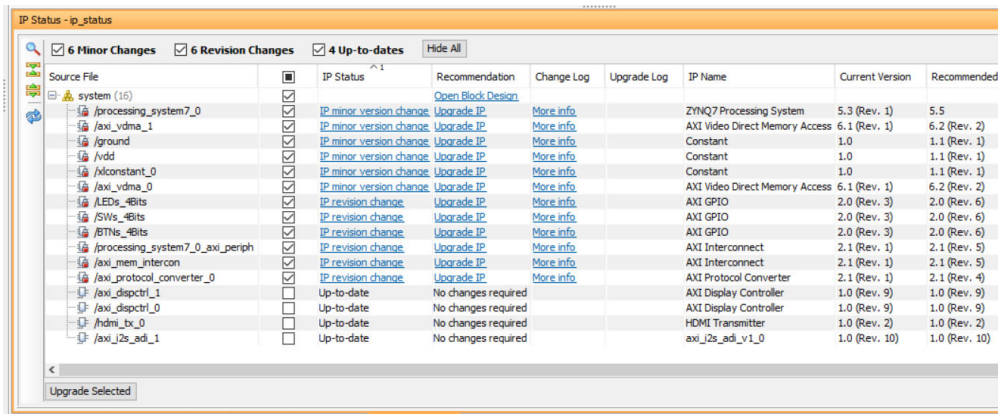


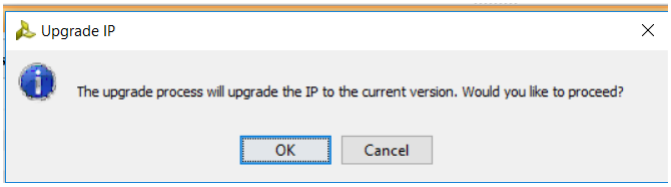


Choose "Report IP Status".

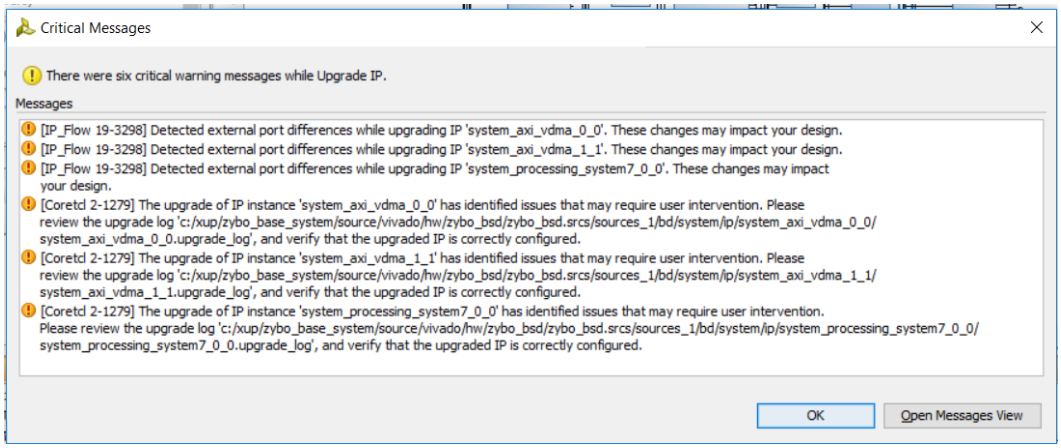
Choose "Update Selected" on the bottom left corner to update all IPs. Check "OK" to update IP.



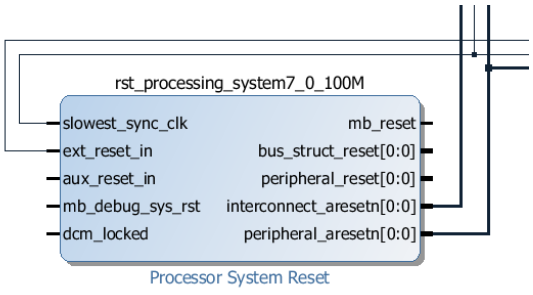




The following warnings will appear. All critical warnings are related to resets because the block design does not have a reset processing system and therefore all reset input pins are connected to Zynq reset outputs. For example, axi_vdma IPs uses FCLK_CLK1 and therefore their reset inputs are connected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



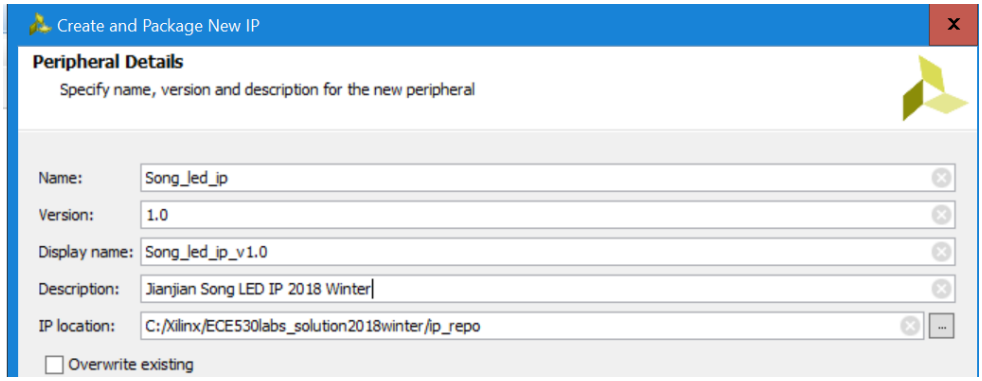
Reset processing system of Vivado 2015.4, which is missing from the Zybo Base System Design from Digilent. One way to remove the above warnings is to add rst_processing_system7_0_100M IP. But it may have unknown issues. Therefore, you may want to keep the original block design.



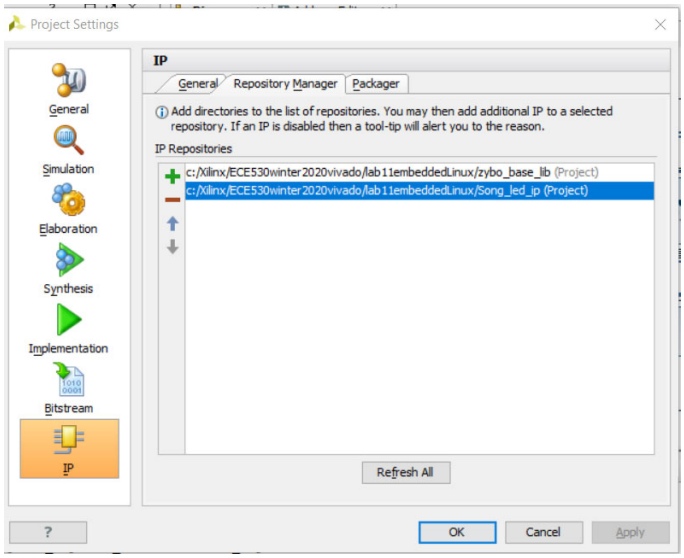
Save the project as a new project named lab11embeddedLinux.

3.3 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. This is an indicator to show you have completed this part independently.



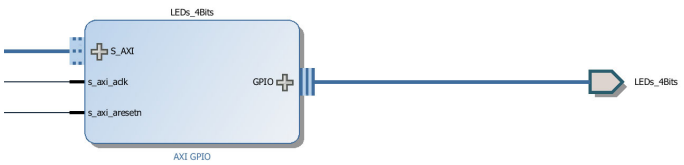
Move this ID folder to your Lab 11 project folder so that this IP is always available. I have moved the zybo base design library to my Lab 11 project folder as shown below.

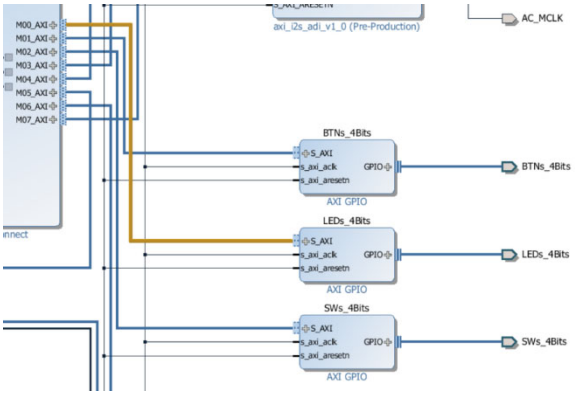


3.4 Create a 4-bit LED IP Core to replace the AXI GPIO IP

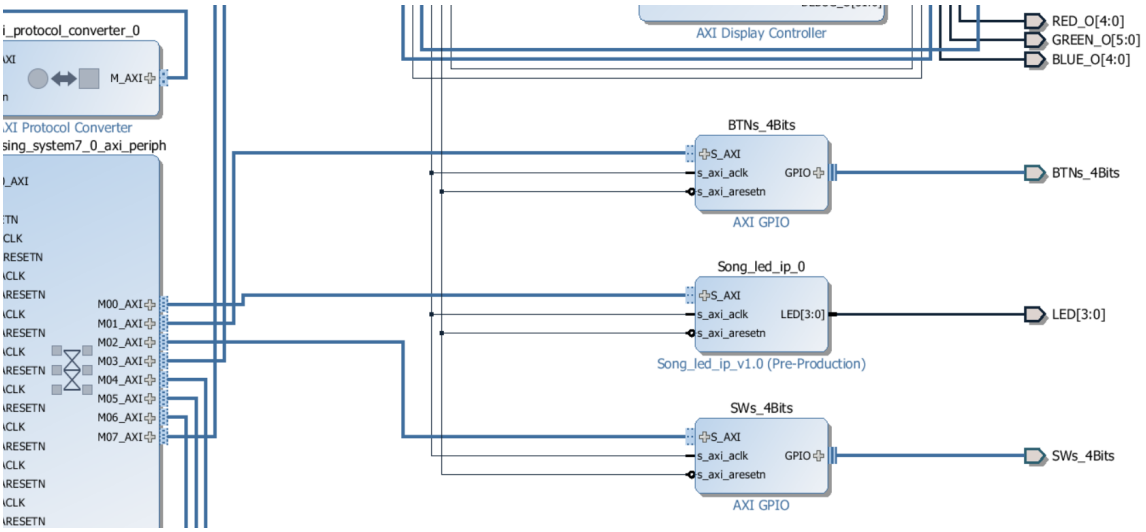
The original Zybo Base System Design uses an AXI GPIO IP to drive the four-bit LEDs. Replace this GPIO interface with your LED IP named your_last_name_led_ip.

Here is the old link for LEDs_4bits.

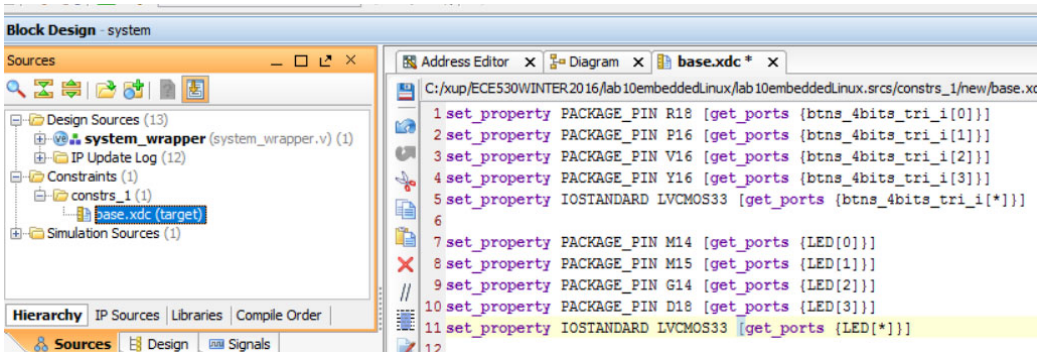




Remove AXI GPIO IP for LEDs and add your LED IP to your design.

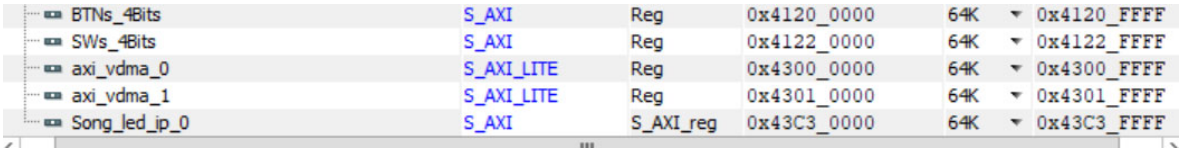


Open constraint file base.xdc under Constraints, change the led pins to LED,

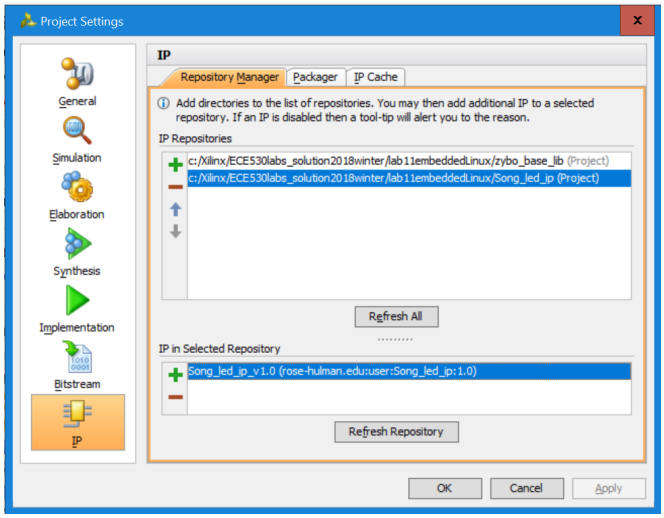


Click on Address Editor to see if your LED IP has been assigned addresses. Your LEDs might not be mapped to any address yet as shown below.

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.

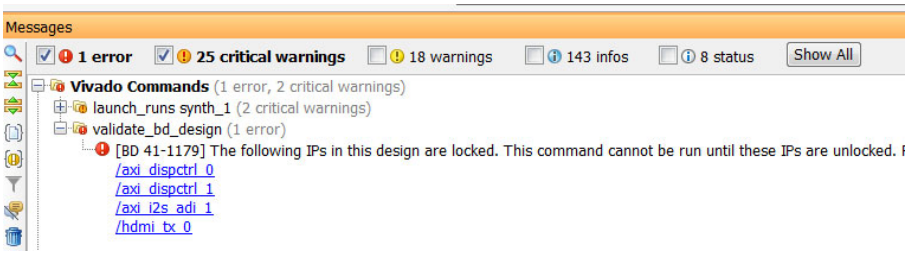


If you do not see your LED IP in your ip catalog, open Project settings->IP to add the IP to your catalog. In the following, the LED IP is called Song_led_ip.

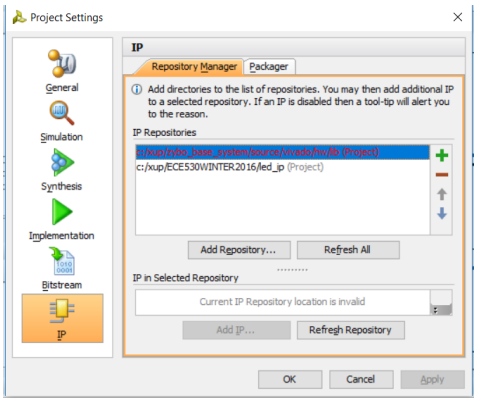


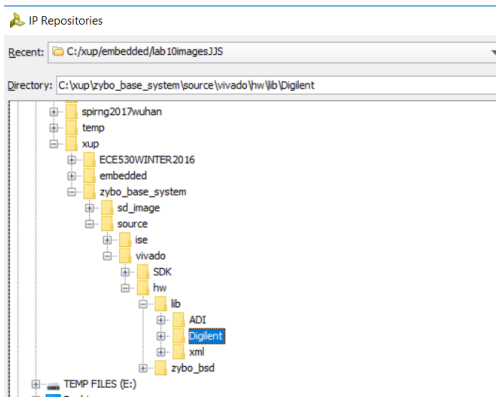
3.5 Verification Error

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.

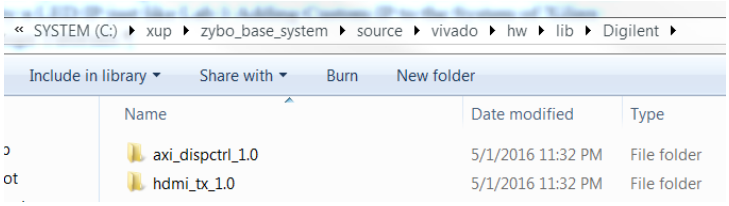


Open Project Settings->IP. Make sure the IP repository link is the same as the actual lib directory.

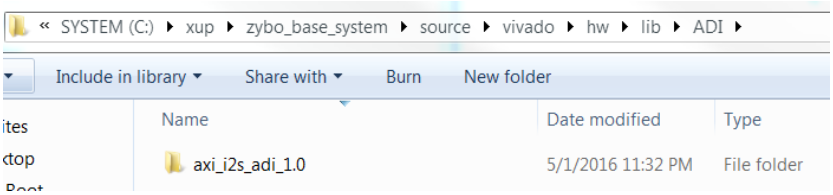




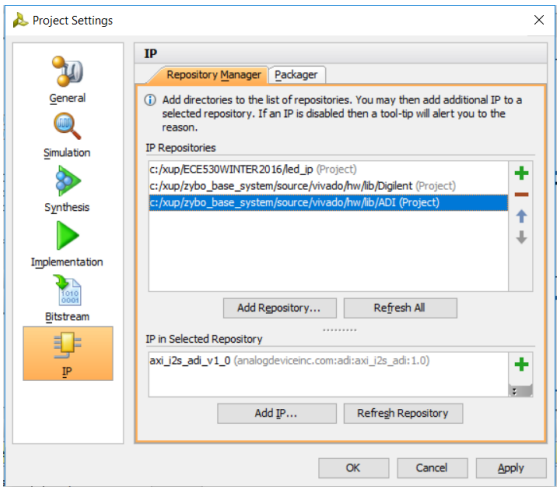
Find where an IP is located and add its repository link accordingly. Here are what folders should be in C:\xup\zybo_base_system\source\vivado\hw\lib\Digilent/.



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

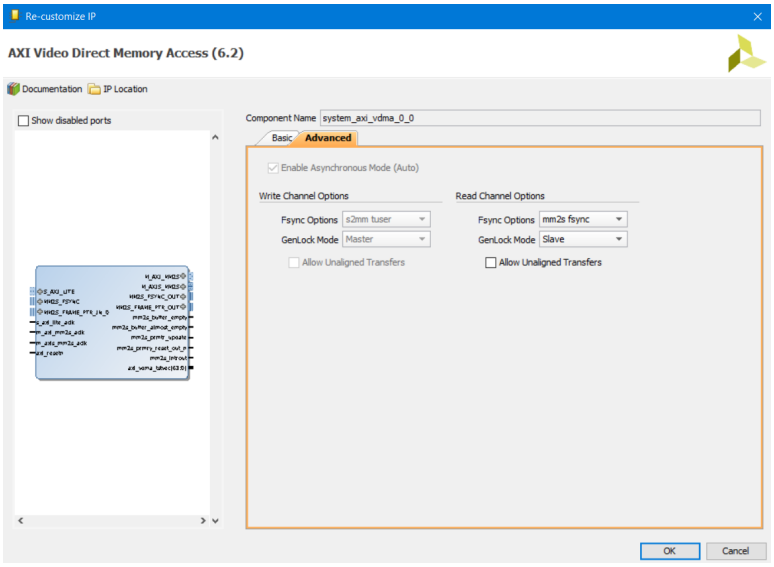


There could be up to three repositories as shown below so that all IPs are allocated.



3.6 Replace VDMA modules with new system_axi_vdma???

This part may not be necessary if you do not see any error messages.



3.7 Generate the bit stream file for your Zybo Base System Design

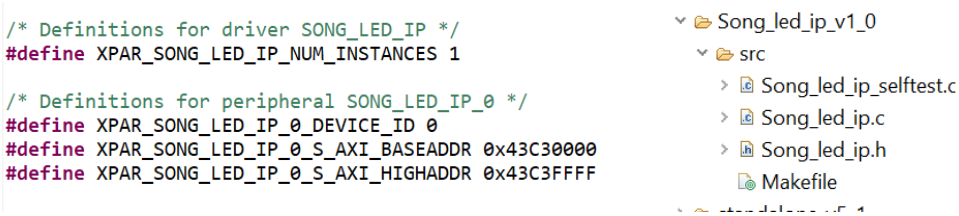
Generate "output products" before starting synthesis. Synthesize Implement your Zybo Base System Design. Check its IO pin assignment. Generate the bit stream file for it. Export it. Launch SDK.

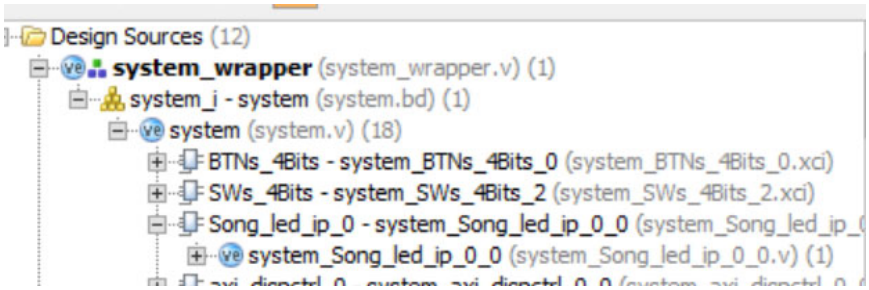
3.8 Create an application with Lab #4 code to verify the block design

The LED IP myLED is the same as Lab #4 "Xilinx Lab4 ZYNQ4 Writing basic software application" to add custom IP and write basic software. You can verity it by running Lab 4 code. Make sure to change the device ID in your Lab #4 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++);
   }
}
```

3.9 (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

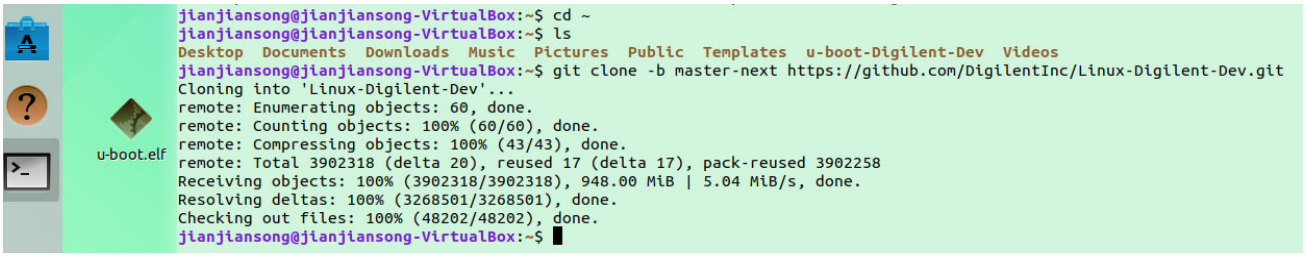
4 Compile Linux Kernel (Section 4 of "Embedded Linux® Hands-on Tutorial")

This section is similar to Section 4 on page 24 of "Embedded Linux Hands-on Tutorial for the Zybo". The current directory should be /Linux-Digilent-Dev under your home directory.

4.1 Get Linux kernel source from Digilent Git repository

Run the following command to get correct source files from github. This operation may take five minutes. The total size of the files is about 4GBs. You may need to restart your VirtualBox after cloning to remove some errors. Notice there is a space between clone and -b.

```
$git clone -b master-next https://github.com/DigilentInc/linux-Digilent-Dev.git
```



4.2 Configure the Linux kernel for Zybo

Make sure u-boot tools path is added to the path variables. Run this command otherwise.

```
$PATH=$PATH:~/u-boot-Digilent-Dev/tools/
$sudo apt install u-boot-tools
```

The current directory should be /Linux-Digilent-Dev under your 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.

```
$PATH=$PATH:~/u-boot-Digilent-Dev/tools/
```

```
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. Run the following command under /linux-Digilent-Dev.

```
$make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi-
xilinx_zynq_defconfig
```

```
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. You can type up/down arrows to recall a command so that you can revise it.

```
$make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- scripts/kconfig/conf -silentoldconfig Kconfig
```

```
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. Set load address to be 0x8000. If you get an error that mkimage command is not found, run "sudo apt install u-boot-tools". This command may take about 5 minutes on my VirtualBox. Type up arrow key to recall a similar previous command so that you can revise it. There is a space between 0x8000 and uImage.

```
$make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- UIMAGE_LOADADDR=0x8000 uImage
```

```
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev$ make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- scripts/kconfig/conf -silentoldcofig Kconfig
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
  CHK     include/generated/compile.h
  GZIP    kernel/config_data.gz
  CHK     kernel/config_data.h
  Kernel: arch/arm/boot/Image is ready
  Kernel: arch/arm/boot/zImage is ready
  Image arch/arm/boot/uImage is ready
```

```
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
  CHK     include/generated/compile.h
  CHK     kernel/config_data.h
  Kernel: arch/arm/boot/Image is ready
  Kernel: arch/arm/boot/zImage is ready
  UIMAGE  arch/arm/boot/uImage
Image Name:   Linux-3.18.0-xilinx-46110-gd627f
Created:      Sun Feb  2 18:04:08 2020
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    3459280 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$ ls -l | grep uImage
-rw-r--r-- 1 jianjiansong jianjiansong 3459344 Feb  2 18:04 uImage
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev/arch/arm/boot$ cp uImage ~/Desktop/
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev/arch/arm/boot$
```

```
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev/arch/arm/boot$ file uImage
uImage: u-boot legacy uImage, Linux-3.18.0-xilinx-46110-gd627f, Linux/ARM, OS Kernel Image (Not compressed), 3459280 bytes, Sun Feb  2 23:04:08 2020, Load Address: 0x0
00, Entry Point: 0x00008000, Header CRC: 0x3FE1A430, Data CRC: 0x60836D05
jianjiansong@jianjiansong-VirtualBox:~/Linux-Digilent-Dev/arch/arm/boot$
```

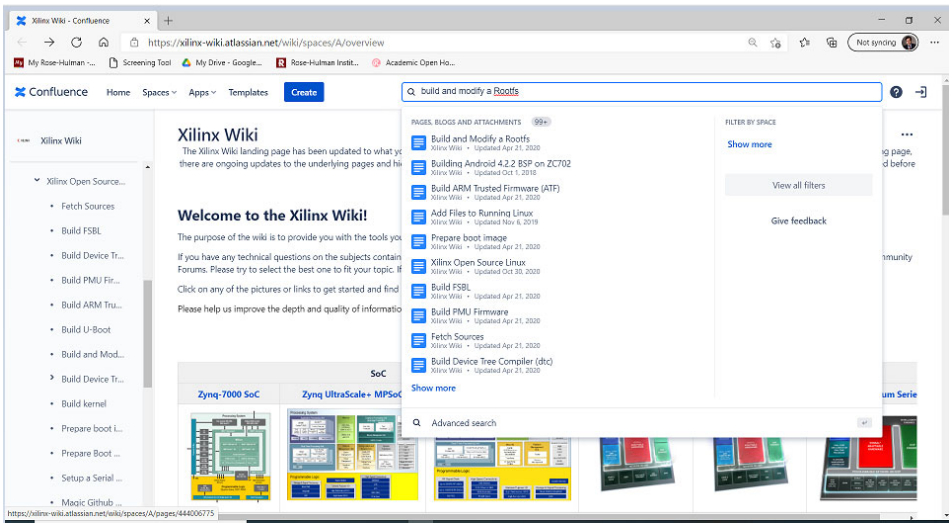
uImage file is ready to be copied to Boot Loader on micro SD card.

5 RAMDISK and Device Tree Blob (DTB File)

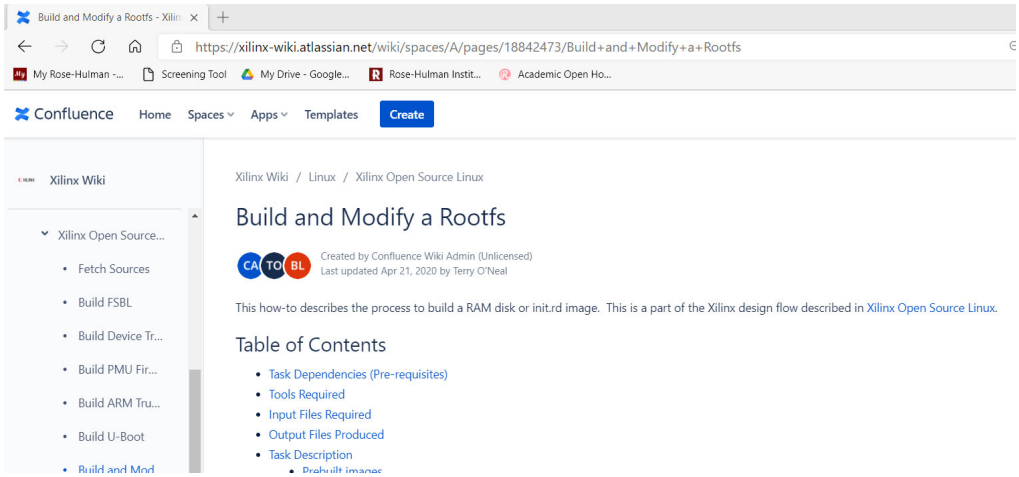
Follow Section 5 on page 25 of "Embedded Linux Hands-on Tutorial for the Zybo" to make a RAM file system uramdisk.image.gz and a device tree devicetree.dtb. They both reside under user's home directory.

5.1 Make RAMDISK

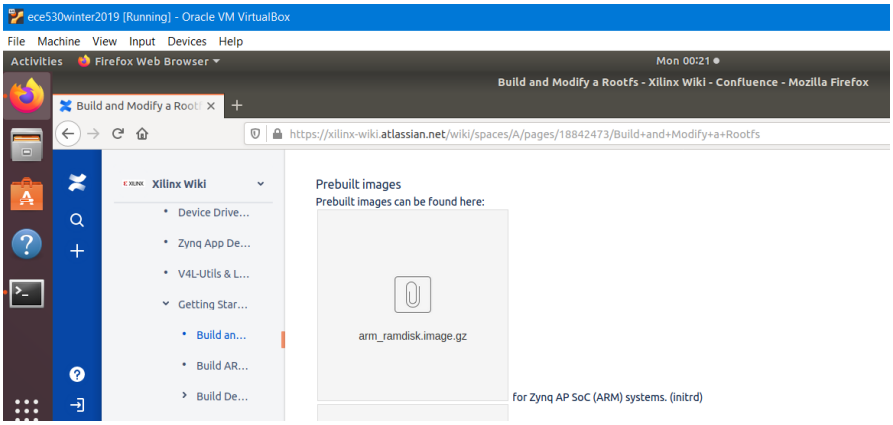
Under your VirtualBox or Linux Window, google Xilinx Wiki to find Xilinx Wiki – Confluence. Open it. Search for Build and Modify Rootfs and open it.

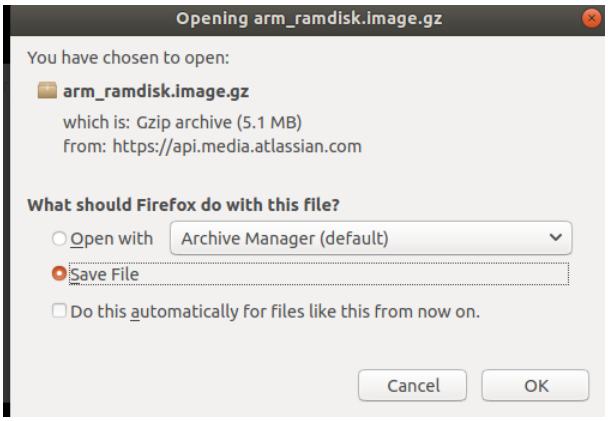


Obtain arm_ramdisk.image.gz from Xilinx Wiki – Confluence at https://xilinx-wiki.atlassian.net/wiki/spaces/A/overview. Go to Getting Started->Build and Modify a Root File System.

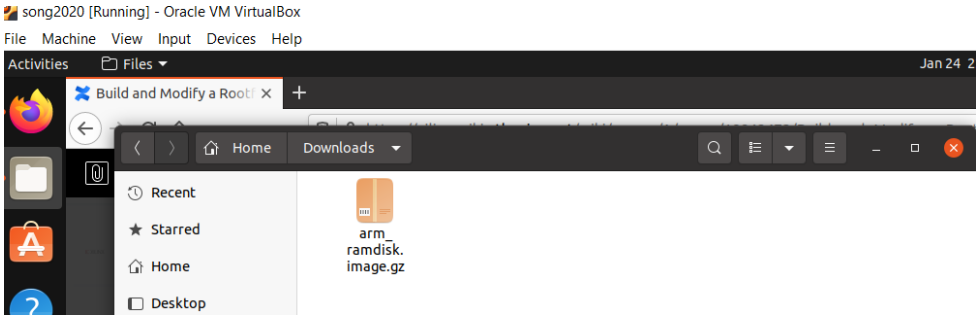


Find Prebuilt images and download arm_rakdisk.image.gz file on this page. This file is used to make an uramdisk.image.gz file.





The file is downloaded to your Download directory.



Open a terminal with Ctrl+Alt+t. Move arm_ramdisk.image.gz to your home directory.

```
$mv ~/Downloads/arm_ramdisk_image.gz  ~/
```

Make a new ram image from it. To complete a directory or file name, type the first few characters and then type Tab key to complete a file name or directory name.

```
$ ./u-boot-Digilent-Dev/tools/mkimage -A arm -T ramdisk -c gzip -d
arm_ramdisk_image.gz uramdisk.image.gz
```

```
jianjiansong@jianjiansong-VirtualBox:~$ ls
arm_ramdisk.image.gz  Desktop  Documents  Downloads  drivers  Linux-Digilent-Dev  Music  Pictures  Public  Templates  u-boot-Digilent-Dev  user_app  Videos
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:      Thu Feb  6 13:57:08 2020
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  Documents  drivers            Music     Public     u-boot-Digilent-Dev  user_app
Desktop               Downloads  Linux-Digilent-Dev  Pictures  Templates  uramdisk.image.gz    Videos
jianjiansong@jianjiansong-VirtualBox:~$
```

5.2 Make a 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
```

Copy the source code zynq-zybo.dts from Linux-Digilent-Dev/arch/arm/boot/dts/zynq-zybo.dts to under drivers/ your home directory ~/drivers/cd zynq-zybo_jjs.dts, where jjs are your name initials.

```
~/drivers$cp ~/Linux-Digilent-Dev/arch/arm/boot/dts/zynq-zybo.dts  zynq-zybo.dts
```

Make a device tree blob named devicetree.dtb with the following command.

```
~/drivers$ ~/Linux-Digilent-Dev/scripts/dtc/dtc -I dts -O dtb -o devicetree.dtb  zynq-zybo.dts
```

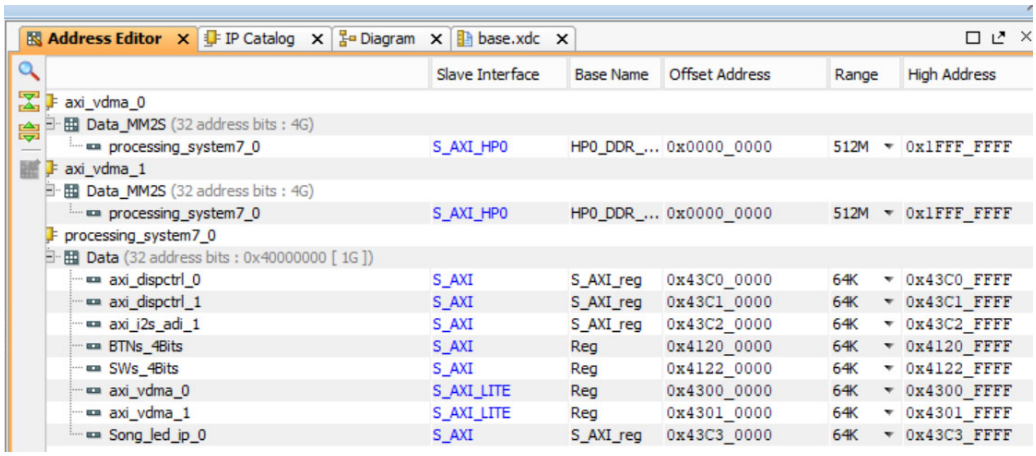
```
jianjiansong@jianjiansong-VirtualBox:~/drivers$ ls
LEDdriver_JJS.c  Makefile
jianjiansong@jianjiansong-VirtualBox:~/drivers$ cp ~/Linux-Digilent-Dev/arch/arm/boot/dts/zynq-zybo.dts zynq-zybo.dts
jianjiansong@jianjiansong-VirtualBox:~/drivers$ ls
LEDdriver_JJS.c  Makefile  zynq-zybo.dts
jianjiansong@jianjiansong-VirtualBox:~/drivers$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

6 Test Your New Boot Loader

Copy your new uImage, uramdisk.image.gz 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.

7 Modify the Device Tree to Add LEDdriver_JJS 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 (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 Song_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 name it zynq-zybo-jjs.dts, where jjs is your name initial.

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

Here is what should be added to the device tree. the LED IP device node name is LED_Device_JJS, where JJS is your name initials. This node name is the name to be used to access this device as a process and file. Make compatible=”Jianjian Song, ECE530 Winter 2021”.

```
LED_Device_JJS {
        compatible = “Jianjian Song, ECE530 Winter 2021”;
        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_Device_JJS {
                        compatible = "JianjianSong, ECE530 winter 2021";
                        reg = <0x43C30000 0x10000>;
                };
        } ;
} ;
```

Make a new device tree blob file, named devicetree.dtb with the following command:

```
~/Linux-Digilent-Dev/scripts/dtc/dtc -I dts -O dtb -o devicetree.dtb zynq-zybo-jjs.dts
```

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

8 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.

8.1 The driver file LEDdriver_JJS.c

Get myled.c file from Moodle and store it in ~/drivers folder. Revise LEDdriver_JJS.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 LEDdriver_JJS.c should be the device name (node name) in the device tree: LED_Device_JJS. Device match table should have the compatible as in the device tree: {.compatible = “Jianjian Song, ECE530 Winter 2021”}. Both name and compatible must be exactly the same as in the device tree. 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. The screen shot also shows when the driver was inserted correctly.

```
rcS Complete
zynq> cd /mnt
zynq> ls
Boot.bin                    devicetree.dtb              uImage
LEDdriver_JJS.ko            led_blink_jjs               uramdisk.image.gz
System Volume Information   u-boot.elf
zynq> insmod LEDdriver_JJS.ko
LED_Device_JJS probed at VA 0x608e0000
zynq> ./led_blink_jjs
```

Notice slab.h is included below because kmalloc and kfree are defined in this header file.

LEDdriver_JJS - Notepad

File Edit Format View Help

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <asm/uaccess.h>    /* Needed for copy_from_user */
#include <asm/io.h>         /* Needed for IO Read/Write Functions */
#include <linux/proc_fs.h>  /* Needed for Proc File System Functions */
#include <linux/seq_file.h> /* Needed for Sequence File Operations */
#include <linux/platform_device.h>  /* Needed for Platform Driver Functions */
#include <linux/slab.h>

/* Define device Name same as that in the device tree*/
#define DRIVER_NAME "LED_Device_JJS"


unsigned long *base_addr;   /* Vitual Base Address */
struct resource *res;       /* Device Resource Structure */
unsigned long remap_size;   /* Device Memory Size */
```

*LEDdriver_JJS - Notepad

File Edit Format View Help

```
    return ret;
}

/* device match table to match with device node in device tree */
static const struct of_device_id myled_of_match[] = {
    {.compatible = "JianjianSong, ECE530 winter 2021"},
    {},
};

MODULE_DEVICE_TABLE(of, myled_of_match);

/* platform driver structure for myled driver */
static struct platform_driver myled_driver = {
```

Create a Makefile with: gedit Makefile as follows. This Makefile states to use a makefile in .~/Linux-Digilent-Dev/ to make M=$(PWD). The main source code in the following example is LEDdriver_JJS.c, which is the same name as the object file: LEDdriver_JJS.o. Make sure the spacing in the Makefile is made up of tabs, not spaces, where necessary.

```
1
2 obj-m   := LEDdriver_JJS.o
3
4 all:
5         make -C ~/linux-Digilent-Dev/ M=$(PWD) modules
6
7 clean:
8         make -C ~/linux-Digilent-Dev/ M=$(PWD) clean
9
```

Run the following command to make the driver from LEDdriver_JJS.c:

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

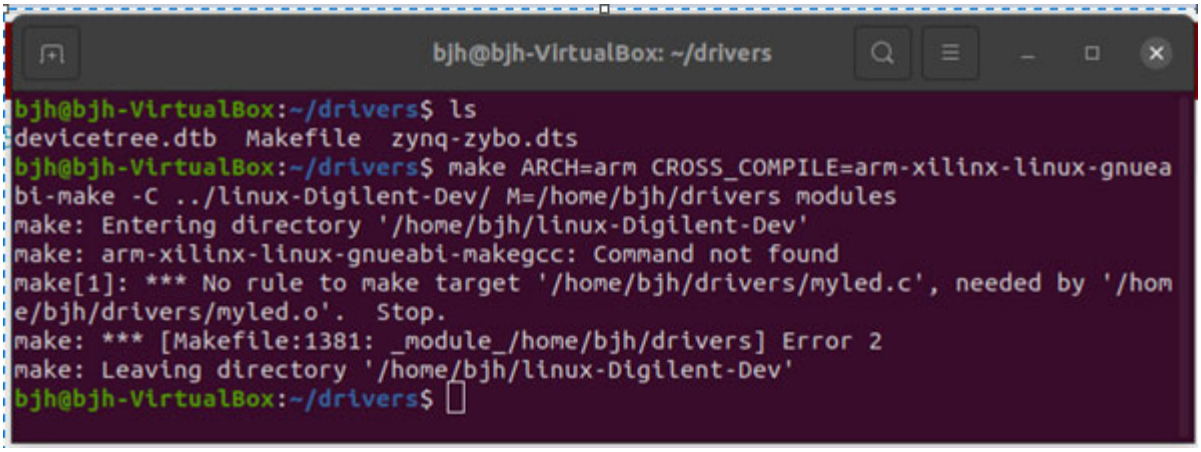
It will create LEDdriver_JJS.ko file as the Linux kernel driver for your LED IP device.

```
jianjiansong@jianjiansong-VirtualBox:~/drivers$ ls
devicetree.dtb  LEDdriver_JJS.c  Makefile  zynq-zybo-jjs.dts
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/LEDdriver_JJS.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/jianjiansong/drivers/LEDdriver_JJS.mod.o
  LD [M]  /home/jianjiansong/drivers/LEDdriver_JJS.ko
make[1]: Leaving directory '/home/jianjiansong/Linux-Digilent-Dev'
jianjiansong@jianjiansong-VirtualBox:~/drivers$ ls
devicetree.dtb   LEDdriver_JJS.ko     LEDdriver_JJS.mod.o  Makefile       Module.symvers
LEDdriver_JJS.c  LEDdriver_JJS.mod.c  LEDdriver_JJS.o      modules.order  zynq-zybo-jjs.dts
jianjiansong@jianjiansong-VirtualBox:~/drivers$
```

8.2 An Error Message: make: arm-xilinx-linux-gnueabi-makegcc: Command not found

If you see the above error, it is possible that you have not installed all required gcc software as follows.

```
$sudo apt-get update
$sudo apt-get upgrade
$sudo apt-get install build-essential
sudo apt-get install gcc-arm-none-eabi gcc-arm-linux-gnueabi
$sudo apt-get install lib32z1 lib32ncurses5 libbz2-1.0:i386 lib32stdc++6.
$sudo apt-get install gcc
```



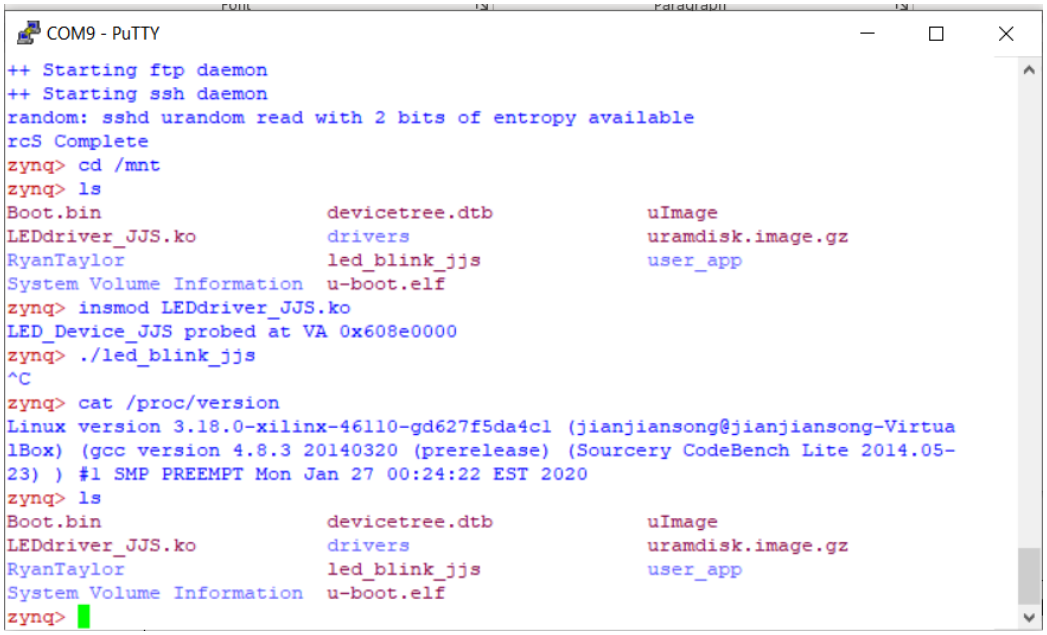
Copy your driver file LEDdriver_JJS.ko file and new devicetree.dtb, device tree blot to your micro SD card for boot loader. Start Linux kernel on your Zybo with these new files. Once your Linux kernel has started, you can find your micro SD card as a device with command “ls /dev”, which will show some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

```
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
```

Otherwise, mount the SD card by 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 LEDdriver_JJS.ko.



If you do not see LED_Device_JJS probed at VA 0608e0000, your LED device driver is installed correctly. If you only see the following, check to make sure the device names and compatible texts for both your device tree and your driver 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_Device_JJS driver process has been started as seen below

```
zynq> ls /proc
1               383             631             fb              partitions
10              4               7               filesystems     scsi
11              483             8               fs              self
12              5               9               interrupts      slabinfo
13              519             LED_Device_JJS  iomem           softirqs
14              535             asound          ioports         stat
15              560             buddyinfo       irq             swaps
16              564             bus             kallsyms        sys
17              57              cgroups         kmsg            sysvipc
171             573             cmdline         kpagecount      thread-self
172             586             config.gz       kpageflags      timer_list
174             589             consoles        loadavg         timer_stats
19              597             cpu             locks           tty
2               6               cpuinfo         meminfo         uptime
274             604             crypto          misc            version
```

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

```
382             620             execdomains     pagetypeinfo
zynq> echo 0x0f > /proc/LED_Device_JJS
zynq> echo 0x00 > /proc/LED_Device_JJS
zynq>
```

8.3 Compile Application file led_blink_jjs.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 name of this file to led_blink_jjs.c, where jjs is your name initials. Change the process to the correct device name in the device tree such as /proc/LED_Device_JJS, where JJS is your name initials.

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_jjs.c, where jjs is your name initials. run "make" to create led_blink_jjs.bin file. Copy this file to your micro SD card for Linux boot loader.

```
//led_blink_jjs.c, revised by Jianjian Song 2019
//led ip application program from Digilent
//Embedded_Linux_hands-on_tutorials_for_the_Zybo, Digilent, 2014
//change driver to LED_Device_JJS -JJS 1-23-2019
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
        FILE* fp;
        while(1) {
        fp = fopen("/proc/LED_Device_JJS", "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_Device_JJS", "w");
        if(fp == NULL) {
                printf("Cannot open /proc/myled for write\n");
                return -1;
        }
        fputs("0x00\n", fp);
        fclose(fp);
        sleep(1);
        }
        return 0;
}
//end led_blink_jjs.c
```

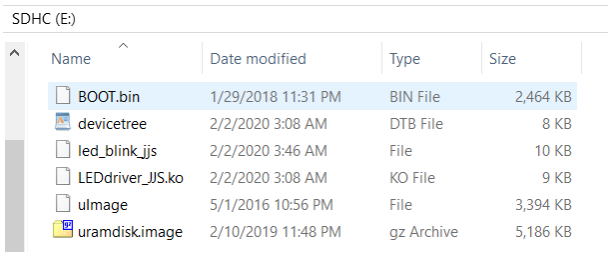
```
Open
1 CC      =       arm-xilinx-linux-gnueabi-gcc
2 CFLAGS  =       -g
3
4 all:    led_blink_jjs
5
6 led_blink_jjs:  led_blink_jjs.o
7                 $(CC) $(CFLAGS) $^ -o $@
8
9 clean:
10        rm -rfv *.o
11        rm -rfv led_blink_jjs
12
13 .PHONY:        clean
14
```

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

9 To Make a Boot Loader with led_blink_jjs.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 LEDdriver_JJS.ko and led_blink_jjs 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_jjs to see the four LEDs on your Zybo flashing. ‘./” means to run a code from the current directory. Kernel date can be checked by command $cat /proc/version to show who made it and when it was made.



```
rcS Complete
zynq> cd /mnt
zynq> ls
Boot.bin                    drivers                     uramdisk.image.gz
LEDdriver_JJS.ko            led_blink_jjs               user_app
System Volume Information   u-boot.elf
devicetree.dtb              uImage
zynq> insmod LEDdriver_JJS.ko
LED_Device_JJS probed at VA 0x608e0000
zynq> echo 0x0f > /proc/LED_Device_JJS
zynq> ./led_blink_jjs
^C
zynq> cat /proc/version
Linux version 3.18.0-xilinx-46110-gd627f5da4c1 (jianjiansong@jianjiansong-Virtua
lBox) (gcc version 4.8.3 20140320 (prerelease) (Sourcery CodeBench Lite 2014.05-
23) ) #1 SMP PREEMPT Mon Jan 27 00:24:22 EST 2020
zynq>
```

10 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.

11 Appendix A: 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.

12 Appendix B: 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.

13 Possible Errors

13.1 arm-xilinx-linux-gnueabi-gcc error: unrecognized command line option: '-m64'

this is because the architecture was not specified and therefore the compiler assumed x86 instead of arm and invoked gcc compiler instead of arm cross-compiler.

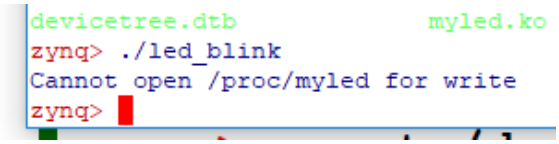
```
jianjiansong@jianjiansong-VirtualBox:~/drivers$ gedit myled_jjs.c
jianjiansong@jianjiansong-VirtualBox:~/drivers$ make CROSS_COMPILE=arm-xilinx-linux-gnueabi-
make -C ../Linux-Digilent-Dev/ M=/home/jianjiansong/drivers modules
make[1]: Entering directory '/home/jianjiansong/Linux-Digilent-Dev'
  CC [M]  /home/jianjiansong/drivers/myled_jjs.o
arm-xilinx-linux-gnueabi-gcc: error: unrecognized command line option '-m64'
arm-xilinx-linux-gnueabi-gcc: error: unrecognized command line option '-mno-red-zone'
arm-xilinx-linux-gnueabi-gcc: error: unrecognized command line option '-mcmodel=kernel'
arm-xilinx-linux-gnueabi-gcc: error: unrecognized command line option '-mno-sse'
arm-xilinx-linux-gnueabi-gcc: error: unrecognized command line option '-mno-mmx'
arm-xilinx-linux-gnueabi-gcc: error: unrecognized command line option '-mno-sse2'
arm-xilinx-linux-gnueabi-gcc: error: unrecognized command line option '-mno-3dnow'
scripts/Makefile.build:263: recipe for target '/home/jianjiansong/drivers/myled_jjs.o' failed
make[2]: *** [/home/jianjiansong/drivers/myled_jjs.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$
```

Here is one solution: add ARCH=arm.

```
jianjiansong@jianjiansong-VirtualBox:~$ cd drivers/
jianjiansong@jianjiansong-VirtualBox:~/drivers$ make CROSS_COMPILE=arm-xilinx-linux-gnueabi- ARCH=arm
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_jjs.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/jianjiansong/drivers/myled_jjs.mod.o
  LD [M]  /home/jianjiansong/drivers/myled_jjs.ko
make[1]: Leaving directory '/home/jianjiansong/Linux-Digilent-Dev'
jianjiansong@jianjiansong-VirtualBox:~/drivers$
```

13.2 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.



13.3 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
```

14 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.