

**Ubuntu on
Zynq®-7000 All Programmable SoC Tutorial**



**for
Zynq Mini-ITX Development Kits**

**Version 4.0
February 2015**

Table of Contents

Tutorial Prerequisites.....	3
Lab Setup for Xilinx Tools.....	5
Technical Support	6
Tutorial Format	7
Tutorial Overview.....	7
Supplied Files	8
Lab 1 - FPGA Hardware Platform	10
Using the SDK Archive.....	10
Lab 2 - Create the Second Stage Boot Loader (U-boot).....	14
Experiment 1: Clone the Xilinx U-boot Git Repository	15
Experiment 2: Configuring U-boot for the Avnet Zynq Target	20
Experiment 3: Building U-boot from the Configured Source	24
Lab 3 – Partition the SD Card for Booting.....	28
Experiment 1: Format the SD Card for Ubuntu Booting.....	28
Lab 4 - Create the First Stage Boot Loader	40
Experiment 1: Create the First Stage Boot Loader	40
Experiment 2: Create the Boot Image	44
Experiment 3: Prepare the Avnet Target for Booting.....	51
Experiment 4: Boot the Avnet Target with Boot Image	54
Lab 5 – Configure and Build the Linux Kernel.....	57
Experiment 1: Clone the ADI Linux Kernel Git Repository.....	57
Experiment 2: Configure and Build the Linux Kernel.....	59
Lab 6 – Install the Device Tree	64
Experiment 1: Compile the Device Tree Source Files	66
Lab 7 – Obtain and Load the Root File System	69
Experiment 1: Download and Install the Root File System	70
Lab 8 – Boot Ubuntu	74
Experiment 1: Boot the Avnet Target to the Ubuntu Desktop.....	75
Lab 9 – Ubuntu Demo	82
Experiment 1: The Basic Desktop	82
Experiment 2: Audio using the ADAU1761 Codec	94
Appendix I – Build Environment Setup under CentOS for Ubuntu.....	101
Resources	108
Revision History	109

Tutorial Prerequisites

This software tutorial relies on a specific development environment. All the necessary files and documentation for creating this can be found at:

<http://www.zedboard.org/>

The following requirements must be met before the tutorial instructions can be completed:

1. Instructions for software builds under Linux are documented using a CentOS development environment and Code Sourcery toolchain, encapsulated inside an Oracle® VM VirtualBox running on a Windows 7 host. Detailed instructions for obtaining and preparing an appropriate environment are provided in the *VirtualBox and Linux VM Installation Guide* located at:

<http://zedboard.org/support/design/2056/17>


The setup instructions include the following topics:

- Obtaining and installing Oracle VM VirtualBox on Windows 7
 - Creating individual Virtual Machines
 - Installing a Guest OS within a Virtual Machine
 - Tips for Vivado (optional) and SDK (required) installation under Linux
2. The Linux kernel and Ubuntu desktop require a hardware platform targeted to the Zynq Mini-ITX 7z045 or 7z100 development board. Download the Vivado reference design for the appropriate target from:


<http://www.zedboard.org/support/design/2056/17>

HDMI Bare Metal Reference Design Using ADV7511 and ADI IP
This reference design demonstrates how to create a bare metal system to output HDMI video and audio using the ADI ADV7511 HDMI transmitter. The IP used in this design is free and publicly available through ADI.

Zynq Mini-ITX 7Z045 HDMI Bare Metal - Vivado 2014.4

 Download

Zynq Mini-ITX 7Z100 HDMI Bare Metal - Vivado 2014.4

 Download

It is an advantage to have some knowledge of the Xilinx Tools and the general architecture and operation of the Zynq device. A good overview can be found in the Avnet Speedways:

Developing Zynq® - 7000 AP SoC Software

Developing Zynq® - 7000 AP SoC Hardware

These are available for download from:

<http://zedboard.org/support/trainings-and-videos>

Lab Setup for Xilinx Tools

To complete the Tutorial labs, the following setups are recommended.

Software

- [Xilinx® SDK 2014.4](#)
- USB-to-UART Bridge Driver
 - [Zynq Mini-ITX: Silicon Labs CP210x](#) (Setup Guide)
- [Tera Term](#)
- [Oracle® VM VirtualBox for Windows](#) (Tutorial uses v4.3.20 r96997)
- Adobe Reader for viewing PDF content
- [7-zip](#) file archive utility

Hardware

- 64-bit host computer with at least 3/6 GB RAM (typical z7045/z7100) and up to 5/10 GB RAM (peak) available for Windows-7 or Linux operating systems with Vivado Design Suite
 - <http://www.xilinx.com/design-tools/vivado/memory.htm#zynq-7000>
- microSD card slot on PC or external USB-based microSD card reader
- Avnet Zynq Development Kit (choose one)
 - Avnet Mini-ITX 7z045 Kit (**AES-MINI-ITX-7Z045-G**) or Avnet Mini-ITX 7z100 Kit (**AES-MINI-ITX-7Z100-G**)
 - 200W ATX Power Supply - *included in kit*
 - CAT-5 Ethernet cable - *included in kit*
 - 2 USB cables (Type A to Micro-USB Type B) - *included in kit*
 - 4 GB microSD card - *included in kit*
- USB Keyboard
- USB Mouse
- HDMI Cable
- HDMI Monitor with speakers or DVI Monitor with an HDMI-DVI adapter
- Headphones and microphone (or combined headset) with 1/8" audio jacks
- Male-to-male 1/8" audio jack connector cable (ADAU1761 audio jack tests)
- Optional powered external speakers (in place of headphones)

Technical Support

For technical support with any of the instructions, please visit the ZedBoard.org support forum:

<http://www.zedboard.org/forum>

Additional technical support resources are listed below.

Zynq Mini-ITX Kit support page with Documentation and Reference Designs:

<http://www.zedboard.org/support/design/2056/17>

Contact your Avnet/Silica FAE or Avnet Support for any additional questions regarding the reference designs, kit hardware, or if you are interested in designing any of the kit devices into your next product.

<http://www.em.avnet.com/techsupport>

For Xilinx technical support, you may contact your local Avnet/Silica FAE or Xilinx Online Technical Support at www.support.xilinx.com . On this site you will also find the following resources for assistance:

- Software, IP, and Documentation Updates
- Access to Technical Support Web Tools
- Searchable Answer Database with Over 4,000 Solutions
- User Forums
- Training - Select instructor-led classes and recorded e-learning options

Tutorial Format

Each lab in the tutorial begins with a brief description of the objective and a set of general instructions. If these are sufficient you may proceed on your own. For those with less experience, detailed instructions for all steps are provided.

Lab instructions for programming the microSD card assume that the card is connected to the host system via a USB adapter.

Tutorial Overview

This tutorial provides a means to integrate several different technologies on a single platform. Using Avnet Development Boards featuring the Zynq®-7000 All Programmable SoC, we have the power of an 800 MHz Cortex-A9 processor chip with dual 32-bit cores, combined with the unrivalled flexibility of Xilinx Programmable Logic to implement custom hardware systems. We use a Linux kernel as the foundation operating system, but also add a fully featured desktop from Ubuntu, contained in the root file system. The desktop allows the target to function as a personal computer using a USB Keyboard and mouse, along with an HDMI monitor.

With Linux and Ubuntu in place, a vast array of applications can be installed and used, mimicking the environment of a standard desktop PC. This includes the potential to develop applications natively on the embedded ARM system.

Supplied Files

Pre-built binary files for the FAT32 portion of the SD card are included in the event that you encounter problems while performing the tutorial experiments. You may copy these files to your SD/micro SD card to verify that you have your environment is correctly configured.

Using these binaries, you will be able to boot the Linux kernel to the point where it attempts to reference the root file system on the ext4 partition, at which point there will be a kernel panic. You can follow the steps in this tutorial to create an ext4 partition and load the root file system to it, at which point you will have a complete Ubuntu desktop system to test.

The root file system cannot be included due to the size of the system, and further you will need to configure the bootable media with both FAT32 and ext4 partitions before it can be loaded. Detailed steps for obtaining the root file system and extracting it to a properly formatted SD/microSD card are covered in this tutorial document.

adau1761: Contains files used in the audio codec validation tests.

adau1761_golden.state: Advanced Linux Sound Architecture configuration file for the ADAU1761 audio codec.

pipeaudio: Script for testing codec stereo record and playback. Copy this script to the Ubuntu desktop and execute it from a Terminal window. An external audio source is required (see Lab 9, Experiment 2)

tonetest: Script for basic audio tones. Use this script to validate stereo audio playback using the Headphone jack. Copy this script to the Ubuntu desktop and execute it from a Terminal window.

sd_card: Contains the files to run test programs from the SD card:

boot.bin: Boots hw platform and executes the U-boot loader for Ubuntu.

devicetree.dtb: Compiled hardware map for the Ubuntu platform.

ulmage: Compressed image for Linaro kernel used to access the Ubuntu desktop.

Ubuntu_on_Zynq_Mini-ITX_Tutorial_04.pdf: This document.

boot_source: Contains files used in the creation of the sd card images.

u-boot¹: Source files that will eventually be included in the Xilinx GIT repository.

Kconfig: Configuration file adds the Zynq Mini-ITX as a supported board.

zynq_mitx.h: Adds a default u-boot configuration for the Zynq Mini-ITX.

zynq_mitx_defconfig: Adds the Zynq Mini-ITX to u-boot targets.

Zynq_mitx_RSA_defconfig: Adds a default secure boot configuration.

system_top.bit: Bitstream for the hardware platform.

u-boot.elf: Compiled u-boot executable.

zynq_fsbl.elf: Compiled first stage bootloader executable.

zynq_mini-itx-adv7511.dtsi: Device tree source file for the design-specific layer of the Zynq Mini-ITX. This file has been augmented to include entries for the ADAU1761 audio codec.

¹ Use only if the Zynq Mini-ITX is not already included in the U-boot hierarchy.

Lab 1 - FPGA Hardware Platform

Lab Overview

The hardware platform used for Ubuntu is created with Xilinx Vivado Design Suite. In this software tutorial we begin with an SDK project based on the exported hardware design.

1. Download the standalone **HDMI Bare Metal Reference Design using ADV7511 and ADI IP** for your target from the location provided in the **Tutorial Prerequisites** section.
2. There are two ways to proceed:
 - a. Complete the entire standalone reference design to for your selected target to learn how to:
 - Design a hardware system capable of supporting an operating system
 - Synthesize, implement and generate a bitstream to encapsulate the hardware design using Vivado Design Suite
 - Export the relevant design files to a Xilinx SDK project
 - Test the hardware design on your selected target to ensure proper functionality of all system components
 - b. Use the compressed SDK archive supplied with the HDMI reference design. This eliminates any prerequisite for Vivado knowledge.

In either case you may use any path you choose for the SDK workspace, but for the purposes of this tutorial the path will be:

C:\mitx\SDK_Ubuntu

Using the SDK Archive

Follow the steps in this section if you choose to start from the SDK archive, rather than completing all steps in the *HDMI Bare Metal Reference Design*.

Once you have downloaded and decompressed the *HDMI Bare Metal Reference Design*, you will find the SDK project archive in subfolder:

<project_path>\Zynq_Mini_ITX-z7nnn_HDMI_Ref_Des_2014_4\SDK_project_archive

1. Copy the file **zynq_mini-itx_hdmi_sdk_ws.zip** to the **C:\mitx\SDK_Ubuntu** folder.

2. Launch Xilinx Software Development Kit (SDK). **Start → All Programs → Xilinx Design Tools → Vivado 2014.4 → SDK → Xilinx SDK 2014.4.**



Figure 1 – The SDK Application Icon

3. Set the workspace to : **C:\mitx\SDK_Ubuntu**

Click the **OK** button.

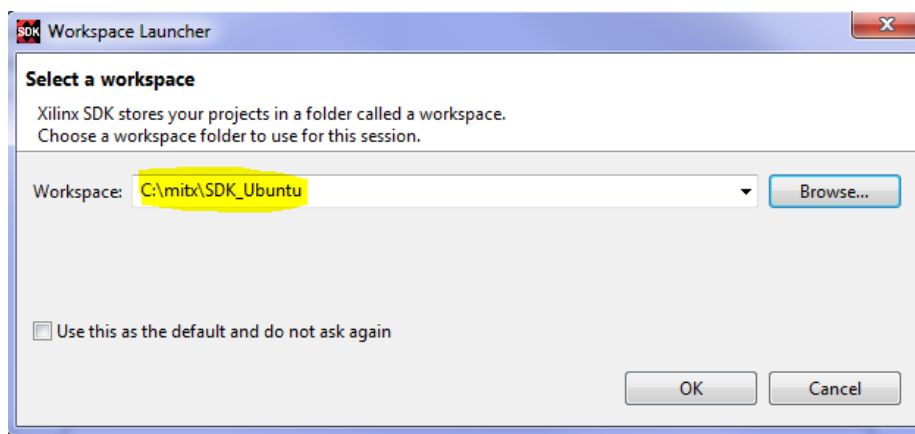


Figure 2 – Select the SDK Workspace

3. Close the *Welcome* tab.

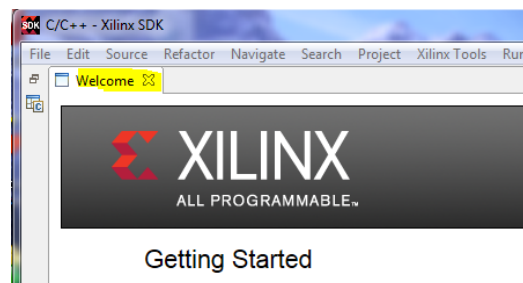


Figure 3 – SDK New Project Welcome Tab

4. From the main SDK menu, select File → Import.

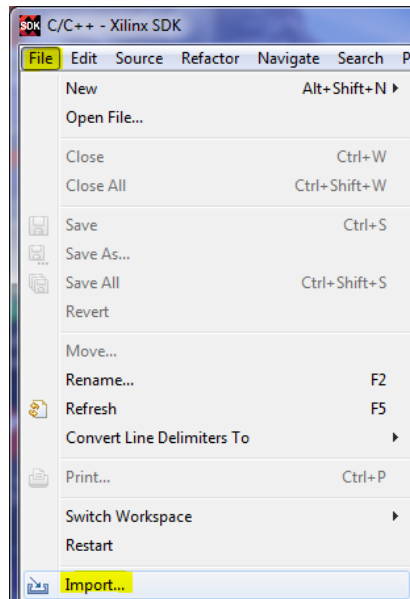


Figure 4 – SDK Import

5. In the *Import* window, expand the **General** entry and click on **Existing Projects into Workspace**. Click the **Next** button.

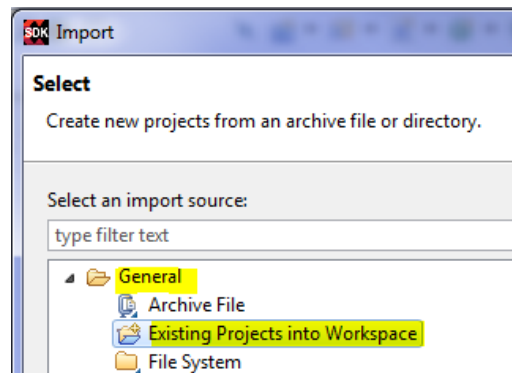


Figure 5 – SDK Projects Import

- Click the **Select archive file** radio button. Click the **Browse** button and select the *zynq_mini-itx_hdmi_sdk_ws.zip* file copied here earlier. Click the **Open** button to close the browser window.

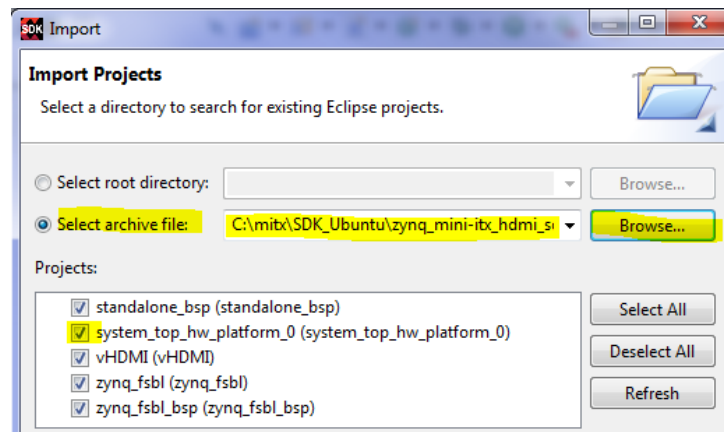


Figure 6 – Archived SDK Projects

- All projects are selected for import by default. You can save steps later by importing all the projects, but at a minimum you must import the *system_top_hw_platform_0* project. If you import only the hardware project, in later steps you will learn to create the other projects from scratch.

Select your desired projects from the checkboxes, and click the **Finish** button to complete the import.

- Use Windows Explorer to copy the hardware bitstream from the SDK project to a new location used to hold all the Zynq boot components.

Create a new folder at:

C:\mitx\Zynq_Ubuntu\boot

Copy the bitstream from:

C:\mitx\SDK_Ubuntu\system_top_hw_platform_0\system_top.bit

to:

C:\mitx\Zynq_Ubuntu\boot\system_top.bit

Lab 2 - Create the Second Stage Boot Loader (U-boot)

Lab Overview

Typically when booting an embedded processing system, there are a series of low-level device-specific operations that are executed to bring the system to a basic operating state. Tasks to be performed may include processor register initialization, memory initialization, peripheral detection and verification, and cache activation. At this stage there are very few resources available, and hence the bootstrap or first-stage loader and its associated software must be as compact as possible.

A second stage loader is generally required to load an operating system. This is beyond the capabilities of the bootstrap loader, so as its final task it loads and passes control to a larger and more capable second stage program. For the purposes of running Linux on our embedded system, U-boot provides all the features needed to act as the second stage.

U-boot is the de-facto loader for embedded Linux because it has been adopted by almost every distribution. It runs on a wide range of processors and has been ported to innumerable boards. Online information is readily available and there are many support forums to aid developers in porting to new architectures. U-boot includes a command line interface and many configuration options, including the ability to pass parameters to the kernel to alter how the boot will proceed. It can load the kernel, the root file system (RFS) and the device tree and perform validation on the installation before passing control to the kernel for execution.

U-boot has already been ported to the Zynq architecture, but not all Zynq targets have been patched back into the mainline distribution. For these targets, you will need to add and modify some files in the u-boot source hierarchy at the locations specified here:

1. **<top_level>/configs/zynq_<target>_defconfig** - each target has a default configuration file. If this file is not part of the distribution, it must be added manually.
2. **<top_level>/include/configs/zynq_<target>.h** contains the target-specific configuration information for u-boot. If the target does not exist in the directory, it must be added manually.
3. **<top_level>/arch/arm/cpu/armv7/zynq/Kconfig** contains the top level configuration information for all Zynq boards. This file sets the parameters used by the default configuration procedure to select the correct *zynq_<target>.h* file.
4. The **zynq_common.h** file specifies u-boot configuration information common to all Zynq targets. You may need to override some defaults, such as extending the default boot delay from 3 seconds, or changing the boot parameters controlled by jumper settings on the board. For example, by default the **sdboot** parameter

assumes the RFS will be loaded to RAM by U-boot. For Ubuntu we chose to boot from an ext4 partition already present on the SD/microSD card. To override the default parameters, do not change the `zynq-common.h` file – instead change the `zynq_<target>.h` file and insert the changes/additions after the `#include` that references `zynq-common.h`. Parameters of the same name will override the parameters in the common file.

For the Zynq Mini-ITX, `zynq_mitx_defconfig`, `zynq_mitx.h` and `Kconfig` files are included in the **Supplied Files**.

Why create the second stage image before we have the first stage loader in place? The initial bootstrap loader image contains the executable for the second stage loader so that it may complete its final task. Therefore, before we can create the first stage image, we must have the second stage boot loader executable available.

When you have completed this lab, you will have learned to:

- Retrieve U-boot source code from the Xilinx repository
- Configure U-boot for the Avnet Zynq target
- Build U-boot for the Avnet Zynq target

Experiment 1: Clone the Xilinx U-boot Git Repository

This experiment shows how to clone the Xilinx U-boot Git repository for Zynq. To successfully complete this lab, you will need Internet access to retrieve the repository information from the Xilinx website.

Experiment 1 General Instruction:

Make a local copy of the Xilinx U-boot Git repository in your home directory, and check out the branch compatible with the Linux kernel we intend to build.

On your Linux host, enter the following commands:

```
$ cd ~  
  
$ git clone git://github.com/Xilinx/u-boot-xlnx.git  
  
$ cd u-boot-xlnx  
  
$ git checkout -b xilinx-v2014.4
```

Experiment 1 Step-by-Step Instruction:

1. If the virtual machine is not already open, launch the VirtualBox application by selecting:

Start → All Programs → Oracle VM VirtualBox → Oracle VM VirtualBox.

If the virtual machine is already open, or you have a native Linux development system, skip ahead to Step 4.



Figure 7 – The Oracle VM VirtualBox Application Icon

2. Select your virtual machine from the choices on the left and click on **Start**.



Figure 8 – The Oracle VM VirtualBox Manager

3. If prompted for login credentials, click on a standard user account and enter your password².

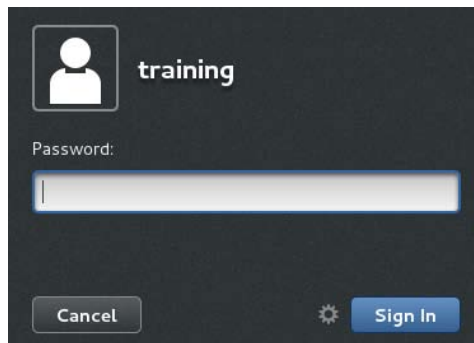


Figure 9 – The CentOS Workstation Login

4. If a terminal is not already open on the desktop of the CentOS guest operating system, open a terminal window from the main menu by selecting:

Applications → Utilities → Terminal

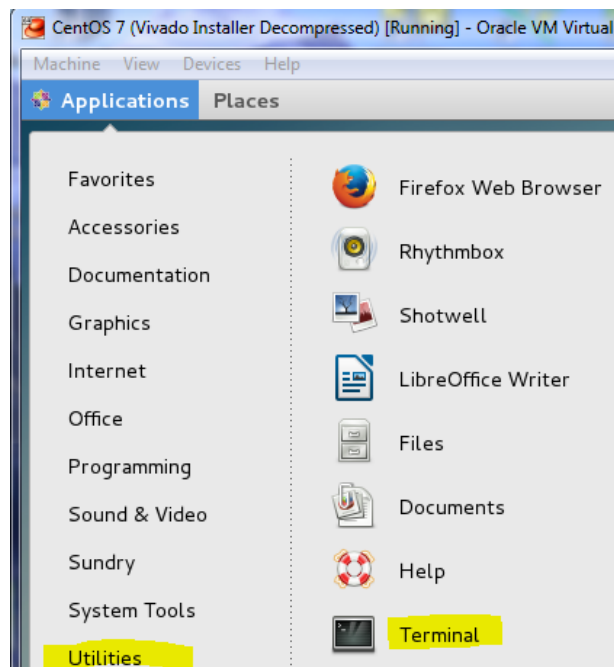


Figure 10 – Launching a CentOS Terminal from the Main Menu

5. The Xilinx U-boot Git repository can be browsed at:

² The username/password from the Avnet VirtualBox setup instructions is **training/Avnet**.

<https://github.com/Xilinx>

To retrieve a working copy of the codebase, clone the remote repository to your local machine. Cloning creates the repository and checks out the latest version, which is referred to as HEAD.

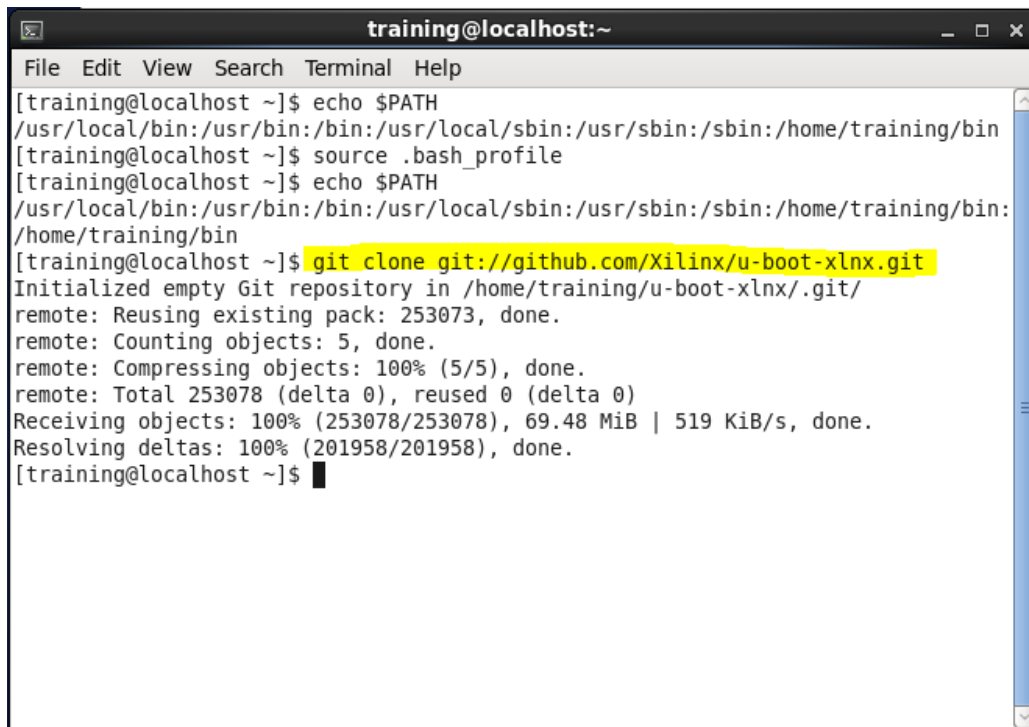
Use the following Git command to clone the repository.

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

6. Wait until the clone operation completes, this could take 5-20 minutes depending upon your connection speed.

The clone command sets up a few convenient items for you by:

- Keeping the address of the original repository
- Aliasing the address of the original repository as origin so that changes can be easily sent back (if you have authorization) to the remote repository



```
training@localhost:~  
File Edit View Search Terminal Help  
[training@localhost ~]$ echo $PATH  
/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/training/bin  
[training@localhost ~]$ source .bash_profile  
[training@localhost ~]$ echo $PATH  
/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/training/bin:/home/training/bin  
[training@localhost ~]$ git clone git://github.com/Xilinx/u-boot-xlnx.git  
Initialized empty Git repository in /home/training/u-boot-xlnx/.git/  
remote: Reusing existing pack: 253073, done.  
remote: Counting objects: 5, done.  
remote: Compressing objects: 100% (5/5), done.  
remote: Total 253078 (delta 0), reused 0 (delta 0)  
Receiving objects: 100% (253078/253078), 69.48 MiB | 519 KiB/s, done.  
Resolving deltas: 100% (201958/201958), done.  
[training@localhost ~]$
```

Figure 11 – Using the Git Clone Command

7. Change into the U-boot source folder.

```
$ cd u-boot-xlnx
```

8. Checkout the code changes related to the selected tools release, which are kept under a designated Git tag. U-boot and Linux kernel versions are closely related, so it is essential that that we match the correct U-boot release to the version of the kernel we intend to use. The Linux kernel that we will be using was ported for Ubuntu by Analog Devices, and at the time of writing was version 3.14.0. This is compatible with the U-boot version in the Xilinx repository from release 2013.4.

```
$ git checkout -b xilinx-v2014.4
```

Note: If you wish to view the available branch tags, use the following command:

```
$ git tag
```

Experiment 2: Configuring U-boot for the Avnet Zynq Target

A working copy of the U-boot repository is now available on the local development machine and can be used to configure the source tree so that it can be built for a Zynq target.

U-Boot, like Linux, is maintained for many different processor systems and must be configured for a designated target before being compiled. The make files included in the U-boot source tree look for a target board name supplied on the command line, and use that to locate the corresponding configuration files in the source tree. This is done using the **make config** command format:

make zynq_<target>_config

where **<target>** for the Zynq Mini-ITX is **mitx**. When this command is executed, a top level configuration file matches the target to a specific configuration file, so all supported Zynq boards must be represented. The top-level configuration file to be modified is found at:

<u-boot top level>/arch/arm/cpu/arm7/zynq/Kconfig

The target-specific configuration files for the Zynq Mini-ITX must be located in the U-boot source hierarchy at:

<u-boot top level>/configs/zynq_mitx_defconfig
<u-boot top level>/include/configs/zynq_mitx.h

Experiment 2 General Instruction:

Configure the parameters for building U-boot source so the executable created is suitable for the Avnet Zynq target. Ensure the Zynq Mini-ITX is part of the u-boot source distribution, and if not, add the **zynq_mitx.h** and **zynq_mitx_defconfig** files from the **Supplied Files**. On your Linux host, enter the following commands:

```
$ cd ~/u-boot-xlnx/  
$ make distclean  
$ make zynq_mitx_defconfig
```

Experiment 2 Step-by-Step Instruction:

1. If the virtual machine is not already open, launch the VMware Player application and open a Terminal window. See the Step-by-Step Instruction 1 through 4 of **Lab 2, Experiment 1**, if you don't remember how to do this.
2. Change from the home directory into the U-boot source directory.

```
$ cd u-boot-xlnx/
```

3. For good measure (sometimes a necessity) run a make distribution clean command against the U-boot source code. This command will remove all intermediary files created by previous configurations as well as any residual files left by prior builds.

```
$ make distclean
```

4. To configure the build for the correct target, the top-level configuration file must have entries for each board to select the target-specific configuration files. The file is located at:

<u-boot top level>/arch/arm/cpu/arm7/zynq/Kconfig

If the target board is not included in the current u-boot source, this file must be modified to add the a new target. First, an entry for a new target must be added into the choice block at the beginning of the file:

```
choice
    prompt "Xilinx Zynq board select"

config TARGET_ZYNQ_ZED
    bool "Zynq ZedBoard"

config TARGET_ZYNQ_MICROZED
    bool "Zynq MicroZed"

config TARGET_ZYNQ_ZC70X
    bool "Zynq ZC702/ZC706 Board"

config TARGET_ZYNQ_ZC770
    bool "Zynq ZC770 Board"

config TARGET_ZYNQ_ZYBO
    bool "Zynq Zybo Board"

config TARGET_ZYNQ_AFX
    bool "Zynq AFX Board"

config TARGET_ZYNQ_CSE
    bool "Zynq CSE Board"

config TARGET_ZYNQ_MINI-ITX
    bool "Zynq Mini-ITX Board"

endchoice
```

Figure 12 – Specify the Zynq Mini-ITX as a Supported U-boot Target

The second change is to include an entry for the new target in the `SYS_CONFIG_NAME` block. This matches the `zynq_<target>` portion of the `zynq_<target>.h` file in the hierarchy.

```
config SYS_CONFIG_NAME
    default "zynq_zed" if TARGET_ZYNQ_ZED
    default "zynq_microzed" if TARGET_ZYNQ_MICROZED
    default "zynq_zc70x" if TARGET_ZYNQ_ZC70X
    default "zynq_zc770" if TARGET_ZYNQ_ZC770
    default "zynq_zybo" if TARGET_ZYNQ_ZYBO
    default "zynq_cse" if TARGET_ZYNQ_CSE
    default "zynq_afx" if TARGET_ZYNQ_AFX
    default "zynq_mitx" if TARGET_ZYNQ_MITX
```

Figure 13 – Designate the Zynq Mini-ITX Configuration File

If the Zynq Mini-ITX is not included in the u-boot source hierarchy, for convenience a modified *Kconfig* file is included in the **Supplied Files**.

5. A board configuration file set is included for each Zynq target. The top level target-specific headers and the default configuration settings can be seen in:

```
<u-boot top level>/configs/zynq_mitx_defconfig3
<u-boot top level>/include/configs/zynq_mitx.h
```

This *zynq_mitx.h* file references a second file where parameters common to all Zynq boards are defined. This file is:

```
/include/configs/zynq-common.h.
```

The *zynq-common.h* file provides default configuration assignments for parameters we may wish to override. For example, U-boot can pass a parameter *sdboot* to the Linux kernel to tell it where to find the root file system. By default, the Root File System is copied to RAM, but for Ubuntu we chose to locate it in an ext4 partition on the microSD card. This conveniently allows all changes made to persist across power-cycles of the target.

³ If the current kernel source does not include these files, you may use the versions from the **Supplied Files** folder included with this reference design package.

To override the default, copy the entire *#define* section for **CONFIG_EXTRA_ENV_SETTINGS**⁴ from *zynq-common.h* to *zynq_mitx.h*, inserting it after the *#include* shown below. You can see this completed by reviewing the *zynq-mitx.h* file in the **Supplied Files** folder included with this reference design.

```
#include <configs/zynq-common.h>

/* INSERT YOUR OVERRIDE PARAMETERS HERE */
```

Figure 14 – Override U-boot Parameters

Edit the *sdboot* "line" as shown in following code snippet. Note there is no *ramdisk* load, since it exists on the *ext4* partition already. And the *bootm* command has been changed to remove the load address used by a *ramdisk*.

Updated:

```
"sdboot=if mmcinfo; then " \
    "run uenvboot; " \
    "echo Copying Linux from SD to RAM...RFS in ext4 && " \
    "fatload mmc 0 0x30000000 ${kernel_image} && " \
    "fatload mmc 0 0x2A000000 ${devicetree_image} && " \
    "bootm 0x30000000 - 0x2A000000; " \
```

Original:

```
"sdboot=if mmcinfo; then " \
    "run uenvboot; " \
    "echo Copying Linux from SD to RAM... && " \
    "fatload mmc 0 0x30000000 ${kernel_image} && " \
    "fatload mmc 0 0x2A000000 ${devicetree_image} && " \
    "fatload mmc 0 0x20000000 ${ramdisk_image} && " \
    "bootm 0x30000000 0x20000000 0x2A000000; " \
```

⁴ Note that the entire section is one long line, with the backslash indicating continuation at the end of each line in the file.

Refining the constant value will cause the compiler to generate numerous warnings. Eliminate the warnings by placing the following code immediately prior to the updated constant:

```
#ifdef CONFIG_EXTRA_ENV_SETTINGS
#undef CONFIG_EXTRA_ENV_SETTINGS
#endif
```

Other parameters you may wish to override are the boot delay and boot prompt.

```
#ifdef CONFIG_SYS_PROMPT
#undef CONFIG_SYS_PROMPT
#endif
#define CONFIG_SYS_PROMPT        "zynq-mitx-uboot >"

#ifdef CONFIG_BOOTDELAY
#undef CONFIG_BOOTDELAY
#endif
#define CONFIG_BOOTDELAY        8
```

You may add these lines with your preferred changes after the CONFIG_EXTRA_ENV_SETTINGS entry. **Save** the file and close your editor.

6. ADD IN STUFF FOR KCONFIG HERE

7. Configure U-boot for the Avnet Zynq target by using the modified configuration.

```
$ make zynq_mitx_defconfig
```

Experiment 3: Building U-boot from the Configured Source

Now that the source tree has been configured for the Avnet Zynq target platform, it can be built using the cross toolchain. The resulting executable file will be created in your working directory with the name **u-boot**. You will need this file later for inclusion in the first-stage loader image created in **Lab 4 - Create the First Stage Boot Loader**.

Experiment 3 General Instruction:

Build U-boot for the Avnet Zynq target platform using the cross toolchain. The source configuration was performed in the previous experiment, so the executable file can be built with a single command. In your Linux system, enter:

```
$ cd ~/u-boot-xlnx
$ make
```

Experiment 3 Step-by-Step Instruction:

1. In the previous experiment, the U-boot source tree was configured for an Avnet Zynq target platform.

Make sure the **/home/training/u-boot-xlnx/** folder is the current working directory in your terminal window. Change folders if necessary.

```
$ pwd
$ cd ~/u-boot-xlnx (if necessary)
```

2. Build the U-boot executable for the Zynq target with the **make** command.

The build process should take anywhere from 5 to 10 minutes to finish and should complete successfully. If the build is successful and the console output looks similar to that shown below, you may continue with the next step.

```
$ make
```



```
training@localhost:~/u-boot-xlnx
File Edit View Search Terminal Help
CC      spl/lib/display_options.o
CC      spl/lib/crc32.o
CC      spl/lib/ctype.o
CC      spl/lib/div64.o
CC      spl/lib/hang.o
CC      spl/lib/linux_compat.o
CC      spl/lib/linux_string.o
CC      spl/lib/string.o
CC      spl/lib/time.o
CC      spl/lib/vsprintf.o
LD      spl/lib/built-in.o
LDS     spl/u-boot-spl.lds
LD      spl/u-boot-spl
OBJCOPY spl/u-boot-spl.bin
MKIMAGE u-boot.img
./tools/zynq-boot-bin.py -o boot.bin -u spl/u-boot-spl.bin
Input file is: spl/u-boot-spl.bin
Output file is: boot.bin
Using /home/training/u-boot-xlnx/spl/u-boot-spl.bin to get image length - it is
45388 (0xb144) bytes
After checksum waddr= 0x13 byte addr= 0x4c
Number of registers to initialize 0
Generating binary output /home/training/u-boot-xlnx/boot.bin
[training@localhost u-boot-xlnx]$
```

Figure 15 – U-boot Build Completed

Note the highlighted line referring to the output binary file from the compiler. This is **NOT** the file for booting Zynq, although it has the same name as a Zynq SD card boot file. The final Zynq *boot.bin* file contains a First Stage Boot Loader image, a bitstream, and a u-boot executable in ELF format. It is the ELF format output that we need to save from the compilation, and that is found at:

/home/training/u-boot-xlnx/uboot

3. When the build has completed, the U-boot hierarchy contains a tool for converting binary files into image files that can be loaded by U-boot. We will

need this tool later when building the Linux kernel, so it is useful to place this tool into a directory that is part of the execution search path.

```
$ sudo cp ~/u-boot-xlnx/tools/mkimage /bin/.
```

- Now that we have a successful build, *double-click* on the home icon on your desktop to open a file browser window.

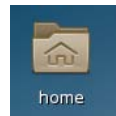
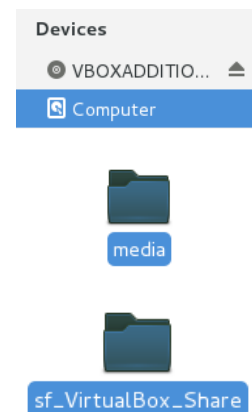


Figure 16 – File Browser Icon

For convenience, you can open a second browser and open it to the shared folder where files can be transferred between the guest and host systems. You can leave this window open to use for drag and drop transfers and expand/minimize it as needed.

- Select **Computer** under *Devices* in the left panel of the browser window.
- Double-click* the **media** folder.
- Double-click* the **sf_VirtualBox_Share** folder.



- In your first browser window, locate the ELF format u-boot image file:

/home/training/u-boot-xlnx/u-boot

Right-click on this file, select the **Copy**⁵ option and **Paste** it into the shared folder browser.

- On the Windows host, use Windows Explorer to access the shared file folder. By default the folder name will be the same as on the VM, except without the *sf_*

⁵ Alternately, you can drag and drop the file from the first browser window to the shared-folder browser window.

prefix. The actual location within your Windows file system was chosen when you established file sharing between the host and guest operating systems.

Copy or move **u-boot** from the shared Windows folder to boot location on your Windows host, created in Lab 1:

C:\mitx\Zynq_Ubuntu\boot

Rename **u-boot** to **u-boot.elf** so that it has the appropriate extension needed for the SDK tools to see it when creating a bootable image.

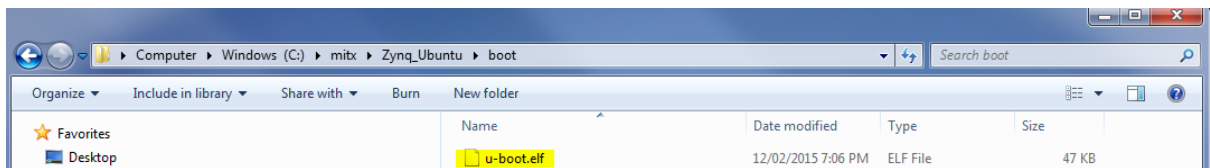


Figure 17 – u-boot ELF Copied/Renamed on the Host OS

Lab 3 – Partition the SD Card for Booting

Lab Overview

The Avnet Zynq target will be booted from data contained on an SD card, and once the Linux environment is operational the root file system will also reside on the SD card. In this lab we will prepare the SD card with the required partition formats.

Experiment 1: Format the SD Card for Ubuntu Booting

In this experiment we will create two partitions on the SD card. The first partition will be in FAT32 format, which can be accessed by either a Windows or Linux operating system. This partition will contain the files used for initial bring-up of the Avnet Zynq target. The second partition will be in ext4 format, usable only by a Linux OS. On this partition we will place the root file system, required by the Linux kernel to complete the OS boot process.

Because the second partition is Linux specific, we will prepare the SD card from the Virtual Machine using a utility called the Gnome Partition Utility (GParted).

Experiment 1 General Instruction:

Use GParted to create two empty partitions on a 4 MB (or larger) SD card. The first partition will be 52MB to 400 MB (depending on the capacity of your media) FAT32 format, and the second partition will use the remaining space in ext4 format.

Note: *All existing data on the SD card will be destroyed. Make sure you back up any data you need to retain prior to the formatting operation.*

It should be noted here that not all SD cards of the same capacity offer the same performance, and even cards of the same advertised capacity may differ between manufacturers. This can create boot issues if you are copying an image from one card to another. If you are using multiple cards, it is best to stick to a single manufacturer to reduce boot problems resulting from bad images.

The data transfer rate of SD cards affects the performance of Linux significantly when the root file system resides on the removable media. Both SD and microSD cards come in different classes, from 4 to 10 at the time of writing, and within a specific manufacturer the transfer rate is generally higher with a higher class. This is not always the case between manufacturers, however. For the best performance, use a card that states its maximum transfer rate. During testing, we used a SanDisk 16 MB class 10 card with a 30 MBps stated transfer rate to create a system with very good response from the Ubuntu Desktop.

Experiment 1 Step-by-Step Instruction:

1. Insert the SD card into a compatible read/write slot on your computer.
2. In the main menu of VirtualBox, select:

Devices → USB Devices → Generic USB Storage

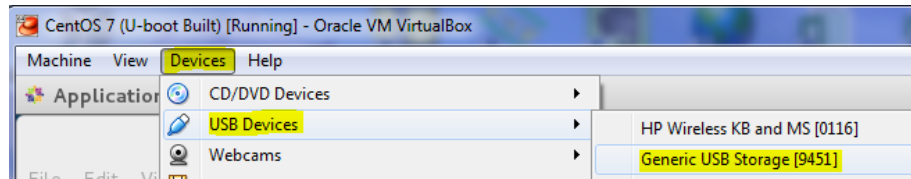


Figure 18 – Select USB Storage Device

3. The first time this is done, virtual device driver software will install. Once the driver is active, the USB device will be captured and a pop-up window appears at the bottom of the guest OS window. Click on **Open with Files** to mount the media and open a browser window.

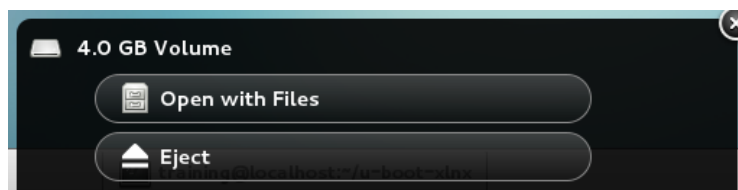


Figure 19 – USB Storage Device Detected by Virtual Machine

4. Close the browser window for the SD card, as we will be repartitioning and formatting the entire card. **Note that any existing data on the card will be LOST.**

5. To run the GParted Partition Editor:

CentOS: From the main menu select:

Applications → System Tools → GParted

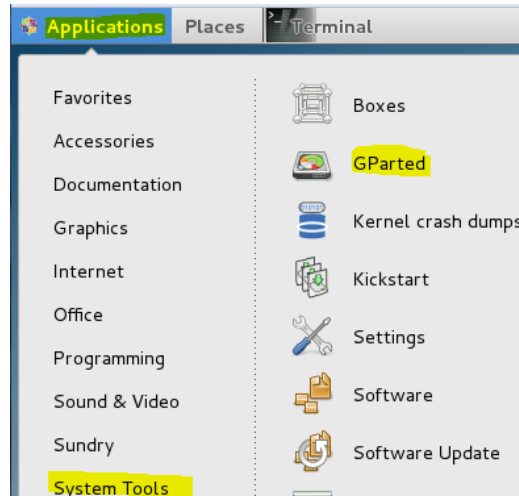


Figure 20 – Launch GParted Partition Editor in CentOS

Ubuntu: Click on the Dashboard icon and start typing the first few characters of gparted into the search field. When the GParted Partition Editor appears, double-click the icon to launch it.

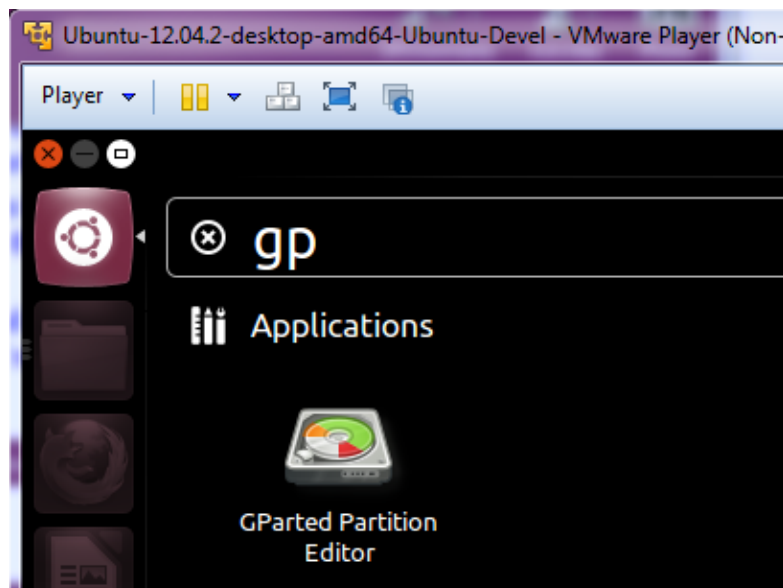


Figure 21 – Launch GParted Partition Editor in Ubuntu

6. Enter the root password when prompted. Click the **Authenticate** button.

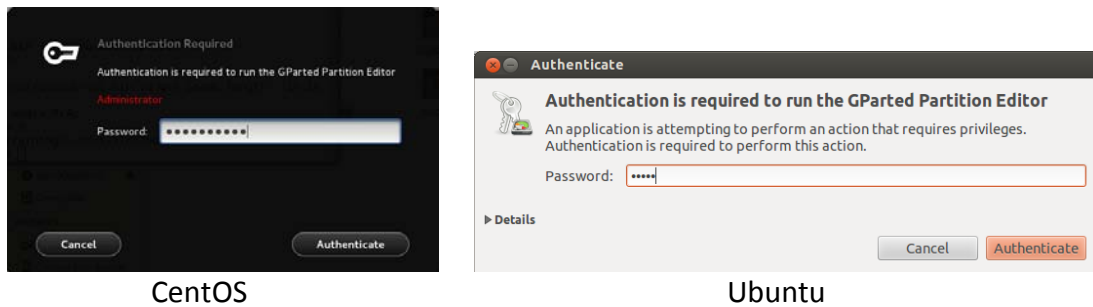


Figure 22 – Enter Root Password

7. GParted will by default display the partitions on your Linux development system. **You DO NOT want to change anything here!** In the upper right, open the drop-down menu to display the available devices, and select the one corresponding to the SD card. This will generally be labelled as **/dev/sdb**, with a size shown that is slightly smaller than the full size of your SD card. Click on this device to select it (This example shows a 4 GB capacity SD/microSD card in the dropdown).

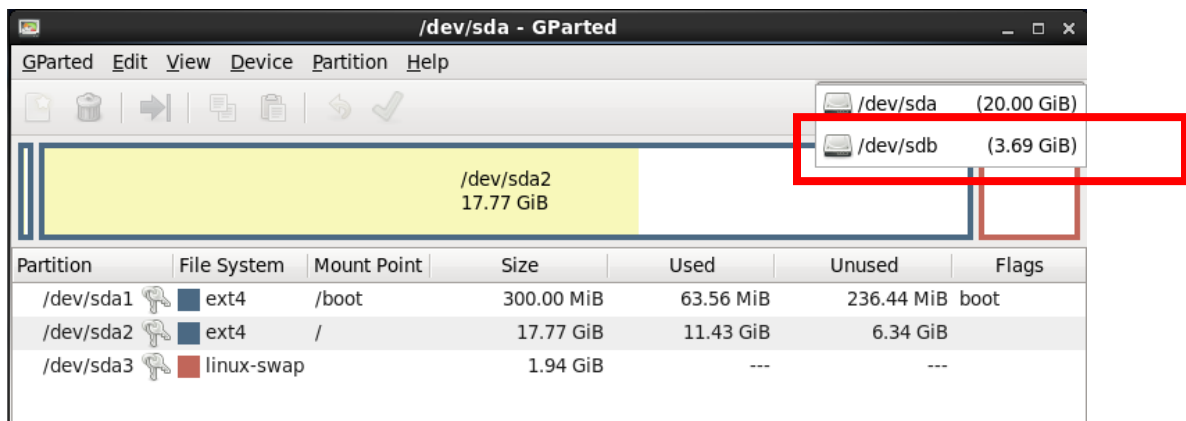


Figure 23 – GParted SD Card Selection

8. If your SD card has been used before, it may contain a file system already. Any existing system not specifically used for the Ubuntu boot process should be removed so we can start with a clean slate. In the example below we have a 4 GB SD card with a FAT32 file system that was created on a Windows 7 system. If the card is empty, you may skip to **step 14**.

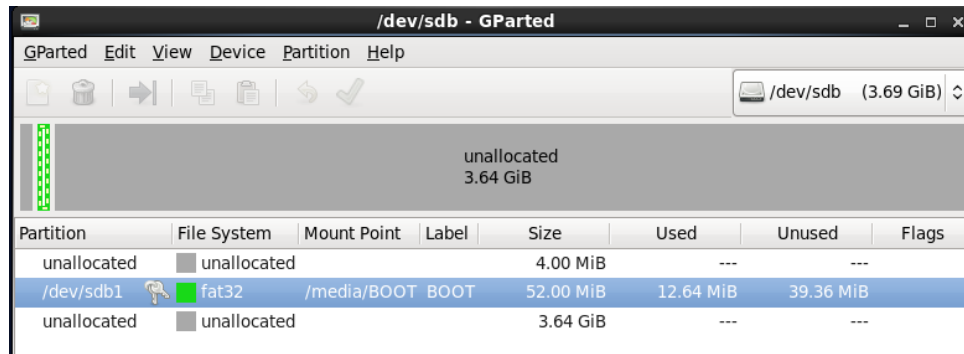


Figure 24 – Existing SD Card Partitions

9. Right-click on the file system to be removed and select **Unmount**.

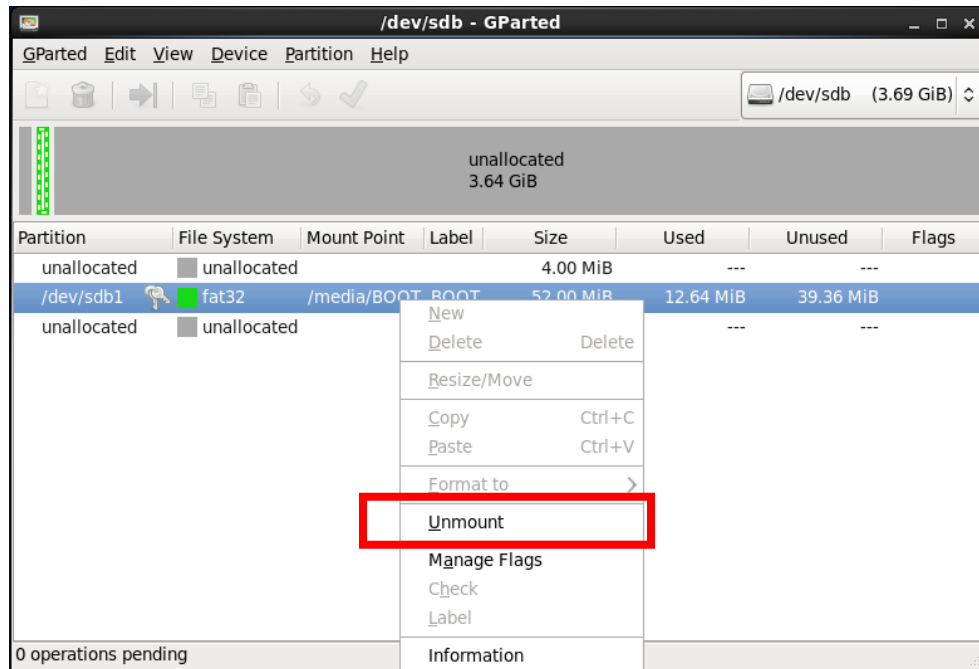


Figure 25 – Unmount an Existing Partition

10. Right click on the unmounted file system and select **Delete**. You will see a pending Delete action appear in a new window at the bottom of GParted. Actions are queued and then executed serially with a single command.

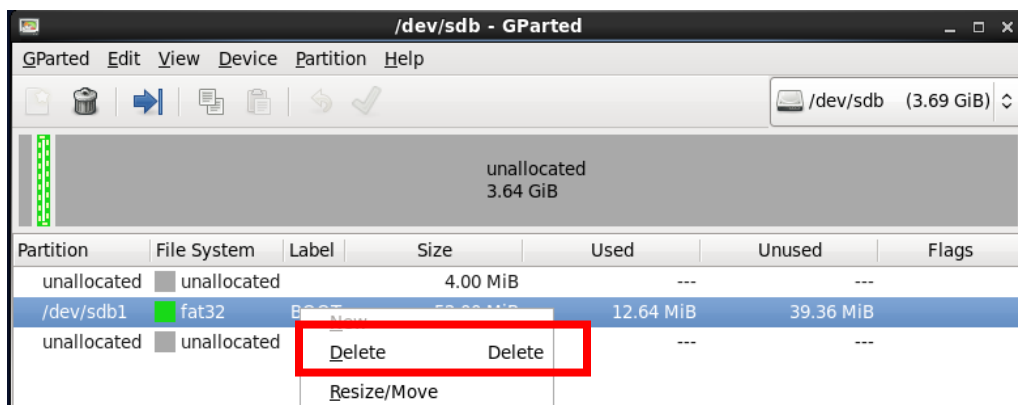


Figure 26 – Delete Existing SD Card Partition

11. You will now see GParted as shown below, with the entire SD card unallocated and one action pending. Click on the Edit menu and select **Apply All Operations** from the dropdown list.

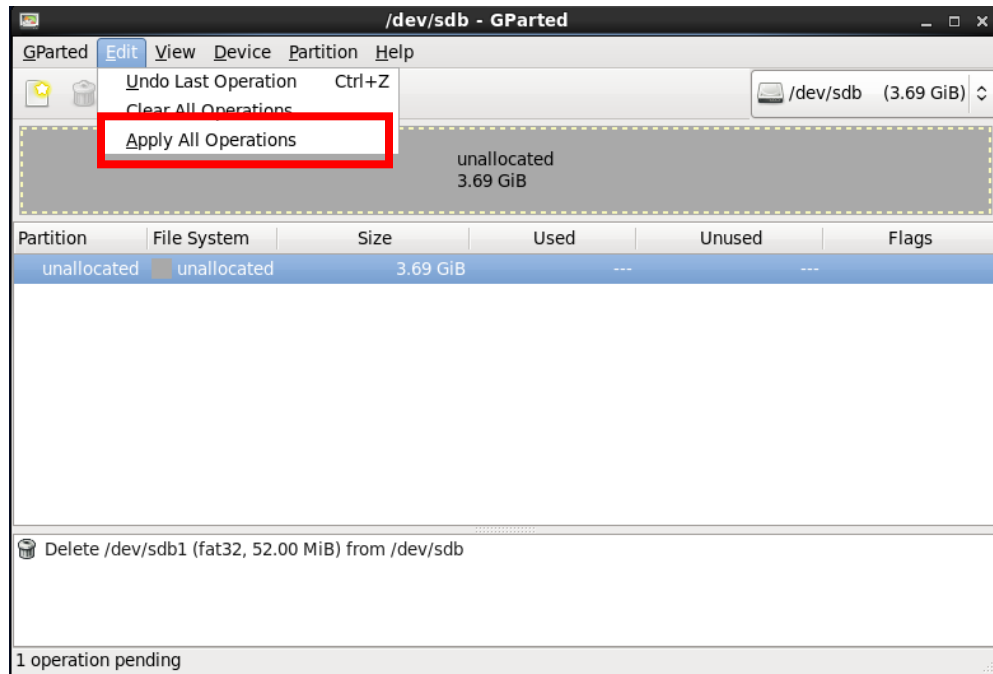


Figure 27 – GParted Perform Pending Operations

12. Click Apply to proceed with the operation. All data on the device will be permanently lost.

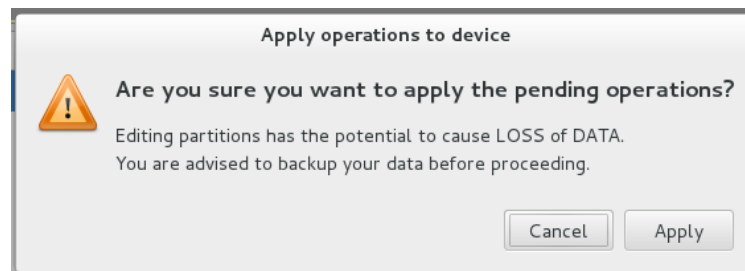


Figure 28 – GParted Verify Partition Operations

NOTE: If you apply operations in GParted and you get back an error saying the resource is busy, make sure you have no open browser windows for the /media directory. Close them if you do, restart GParted, and the actions should work.

13. Once the operations have completed, you will see a message indicating that all operations were successful. Click **Close**.

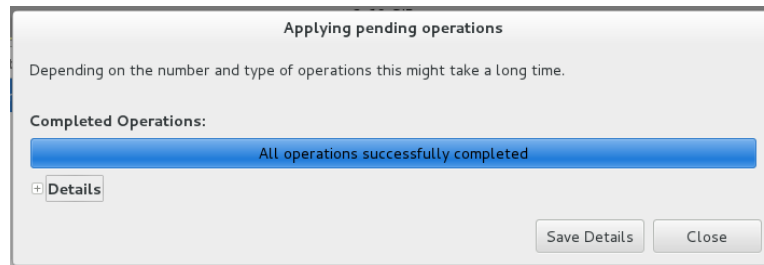


Figure 29 – GParted Operations Successful

14. Your SD card should now appear with the full space completely unallocated. Right click on the unallocated space and select **New**.

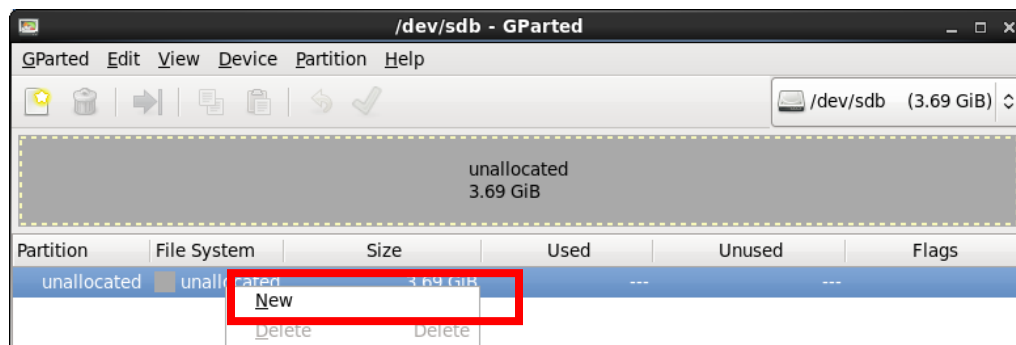


Figure 30 – Create New SD Card Partition

15. In the *Create New Partition* window, enter the parameters shown below⁶ and click the **Add** button. The operation will queue up in the lower window in GParted.

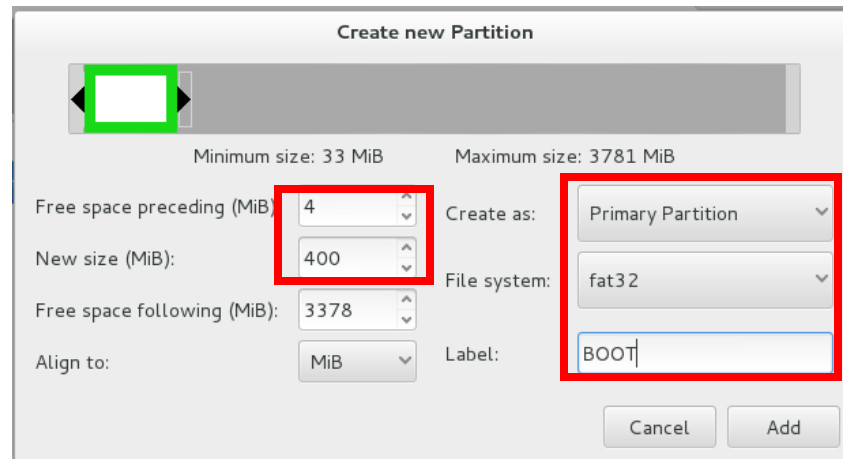


Figure 31 – Create FAT32 Partition

16. Right-click on the unallocated space below your new FAT32 file system and select **New**.

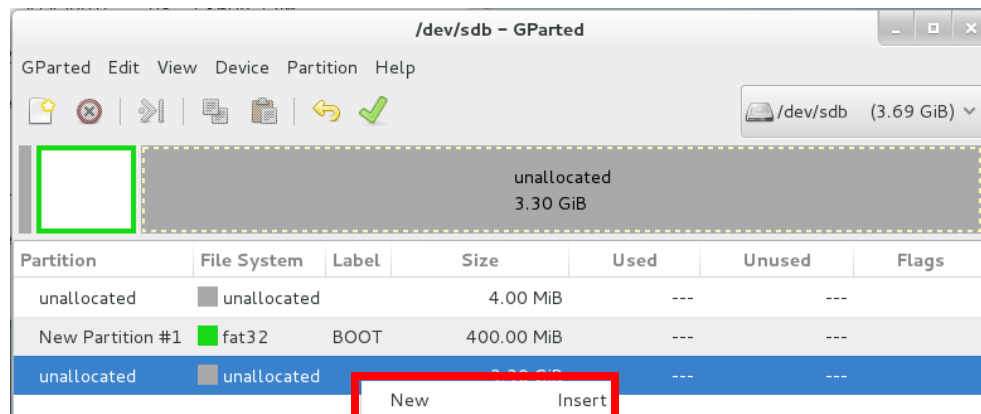


Figure 32 – Create New SD Card Partition

⁶ 52 MB is the minimum recommended FAT32 partition size for the boot images. If you have a higher capacity card, you may want to increase this partition size to allow you more flexibility on the files you can save there. 400 MB is typically large enough to accommodate any extra files you wish to retain.

17. In the *Create New Partition* window, enter the parameters shown below and click the **Add** button. The operation will queue up in the lower window in GParted. The New size will automatically default to the remaining free space on your SD card, which is what we want.

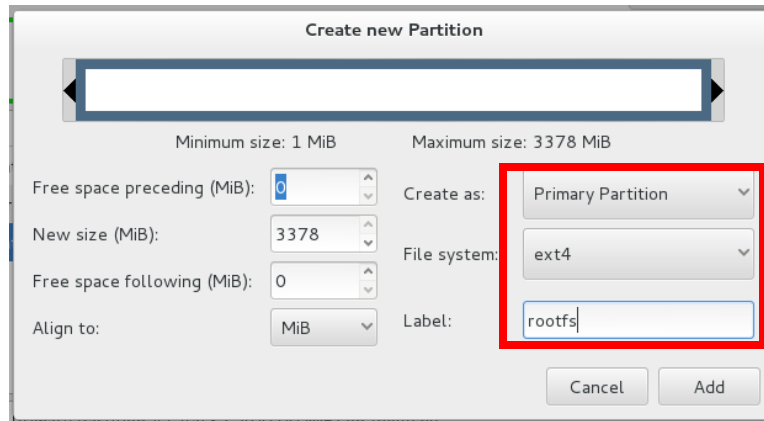


Figure 33 – Create New ext4 Partition

18. The GParted window should now look very similar to the one shown below. If you are satisfied that all is as expected, select **Edit** from the menu and click on **Apply All Operations**.

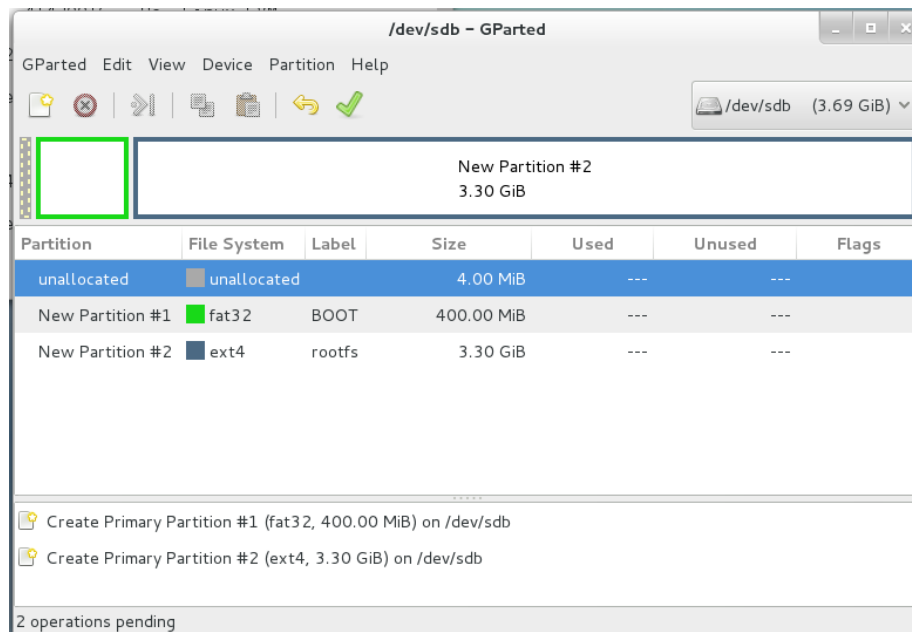


Figure 34 – Partition SD Card with FAT32 and ext4

19. Click **Apply** to proceed with the operation. All data on the device will be permanently lost. Depending on the size of your SD card, the operations may take a few minutes or more to complete.

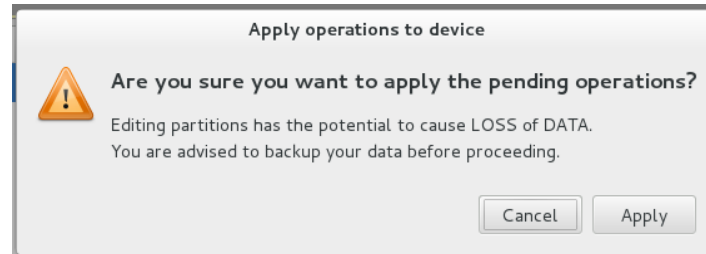


Figure 35 – Verify Partition Creation Operation

20. Once the operations have completed, you will see a message indicating that all operations were successful. Click **Close**.

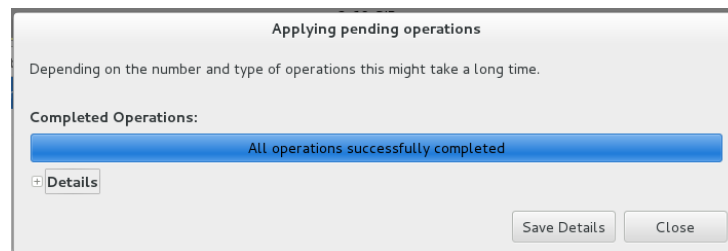


Figure 36 – Partition Creation Successful

21. Your SD card is now ready for the files that will be used to boot Ubuntu on the Zynq target. The partition information should look similar to the figure shown below. Close the *GParted* window.

Partition	File System	Label	Size	Used	Unused	Flags
unallocated	unallocated		4.00 MiB	---	---	
/dev/sdb1	fat32	BOOT	400.00 MiB	820.00 KiB	399.20 MiB	
/dev/sdb2	ext4	rootfs	3.30 GiB	128.67 MiB	3.17 GiB	

Figure 37 – New Partition Table

22. Disconnect the USB device from the virtual machine. In the main menu of VirtualBox select:

Devices → USB Devices → Generic USB Storage

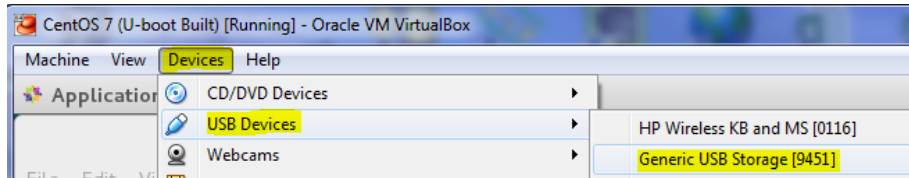


Figure 38 – Disconnect USB Storage Device

23. Back in Windows you can open Windows Explorer to locate the removable drive, which will now appear as a FAT32 drive named *BOOT*. Note that Windows does not recognize the ext4 file system format, so it is not available from the Windows host. *Right-click* on *BOOT*, select **Eject** and you can remove the media. The card is now ready for use in subsequent labs.

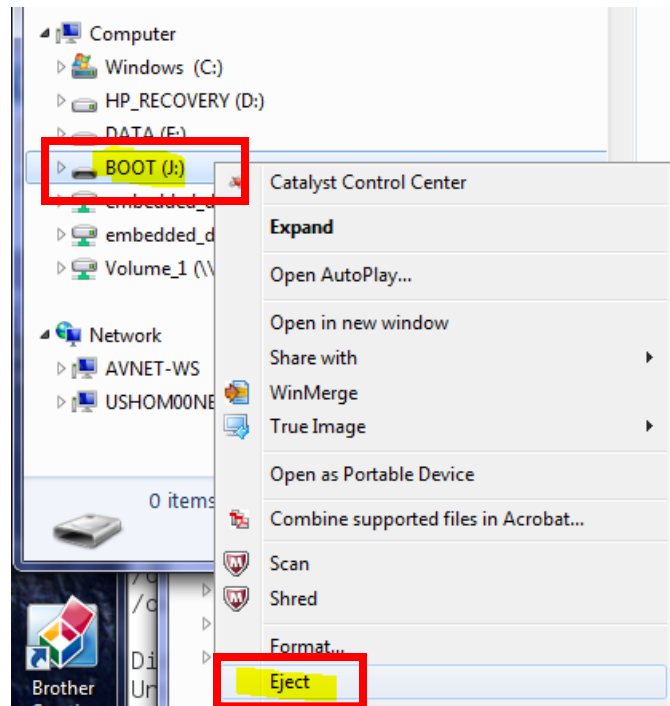


Figure 39 – Remove SD Card from Host

Lab 4 - Create the First Stage Boot Loader

Lab Overview

At this point all the components necessary for initial bring-up of the Avnet Zynq target have been created, except for the First Stage Boot Loader (FSBL). We have the hardware bitstream file from “Lab 1 – FPGA Hardware Platform”, and in the second lab we configured and built U-boot to serve as the second stage loader that will ultimately launch the operating system. In Lab 3 we initialized the boot media.

In this lab we will use the Xilinx SDK in Windows⁷ to create the FSBL and combine it with the other two components into a single boot image, and test that image to verify we can complete a first stage boot and see a U-boot prompt on the console.

When you have completed this lab, you will know how to do the following:

- Build an FSBL for the Avnet target
- Create a boot image
- Boot the Avnet target to a U-boot prompt

Experiment 1: Create the First Stage Boot Loader

Experiment 1 General Instruction:

Launch Xilinx Software Development Kit (SDK) and open the workspace containing the Standalone platform from Lab 1. Create an FSBL project, allow the SDK to generate the default source code, and if necessary, modify the code to customize it for your selected target. An example of required customization is shown in the Step-by-Step Instruction for the Avnet Mini-ITX board (ZedBoard does not require any customization).

Experiment 1 Step-by-Step Instruction:

1. Launch Xilinx Software Development Kit (SDK). **Start → All Programs → Xilinx Design Tools → Vivado 2014.4 → SDK → Xilinx SDK 2014.4.**



Figure 40 – The SDK Application Icon

⁷ If you prefer, the SDK can be run on a Linux host. Operation within the SDK is identical; there will be small differences in procedures for operations performed outside the SDK.

2. Set the workspace to the path for your target. This is the path to the SDK workspace from Lab 1. The example path is:

C:\mitx\SDK_Ubuntu

Click the **OK** button.

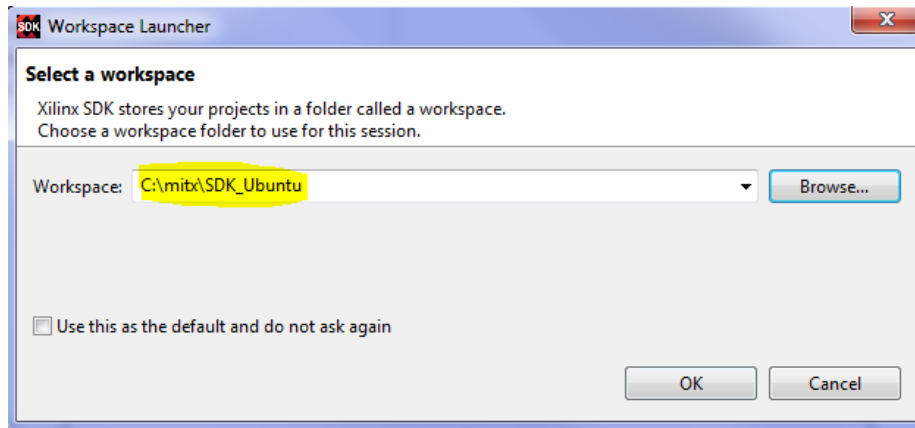


Figure 41 – Select the SDK Workspace

3. When the SDK launch is complete, your SDK project should have, at a minimum, the *system_top_hw_platform_0* project installed. If you have the first stage boot loader project Lab 1, you may skip to Step 6.

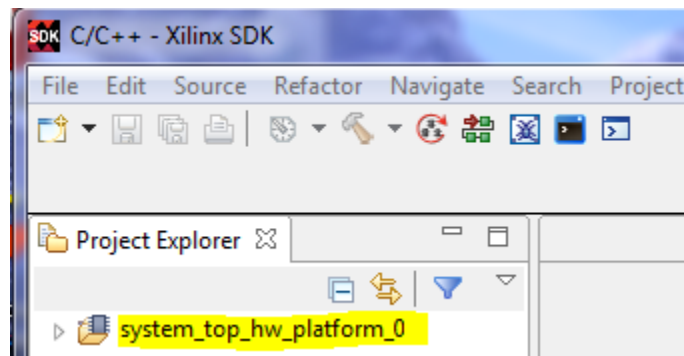


Figure 42 – Hardware Platform Project

4. To boot the Avnet target, we need a bootstrap loader to initialize the processor state. To do that we can create a new Application project for a Zynq 1st stage boot loader.

Select **File -> New -> Application Project** from the menu. Enter **zynq_fsbl_0** for the *Project Name*, accept the remaining default values and click **Next**.

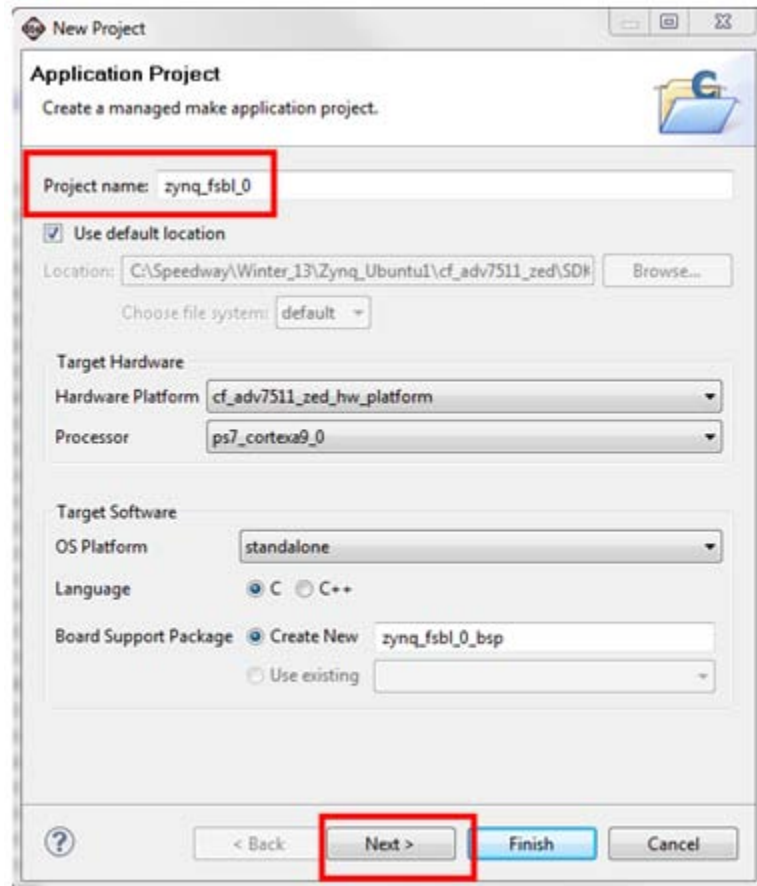


Figure 43 – First Stage Bootloader Project

5. Select **Zynq FSBL** from the Available Templates list and click **Finish**.

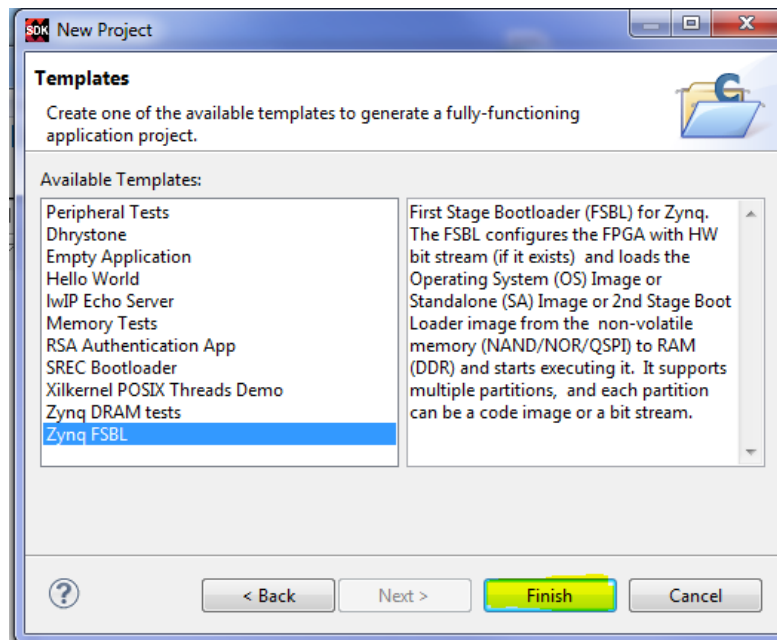


Figure 44 – First Stage Boot Loader Template

The SDK creates a *zynq_fsbl_0* project and a Board Support Package project *zynq_fsbl_0_bsp*, and automatically build the software. Click on the *Console* tab to view the build process.

6. The elf file for the bootstrap loader is created at:

C:\mitx\SDK_Ubuntu\zynq_fsbl_0\Debug\zynq_fsbl_0.elf

Using Windows Explorer, copy the *zynq_fsbl_0.elf* file and browse to the folder created earlier to hold the bitstream file:

C:\mitx\Zynq_Ubuntu\boot

Paste the elf file in at that location. We will be using it in the next Experiment to create a boot image for the Avnet target.

Experiment 2: Create the Boot Image

This experiment shows how to export a boot image using SDK.

Boot Image Format

The Zynq BootROM is capable of booting the processor from several different non-volatile memory types using a file-based data structure called the Boot Image Format (BIF). For the most part, Linux systems accessing Programmable Logic IP require three components within the boot image:

1. FSBL (Stage 1 boot loader)
2. Programmable Logic (PL) Hardware Bitstream
3. U-boot (Stage 2 boot loader)

Refer to Xilinx UG585 document, the Zynq Technical Reference Manual, for further details.

Boot Medium

Although Avnet Zynq Development boards can boot from Quad-SPI Flash, Ethernet or removable SD/microSD media, for the purposes of this lab we will use the SD/microSD card as the boot medium. This gives us the advantage of being able to write directly to the FAT32 partition of the SD card from a PC, and is typical for a development system.

Experiment 2 General Instruction:

Launch Xilinx Software Development Kit (SDK) and open the workspace used in Lab 4, Experiment 1. Use the SDK **Create Zynq Boot Image** tool to create the Zynq boot image file (**boot.bin**).

Experiment 2 Step-by-Step Instruction:

1. If the SDK project is not still open from the previous lab, launch it using steps 1 and 2 from Lab 4, Experiment 1.

2. From the main SDK menu, select **Xilinx Tools** → **Create Zynq Boot Image**.

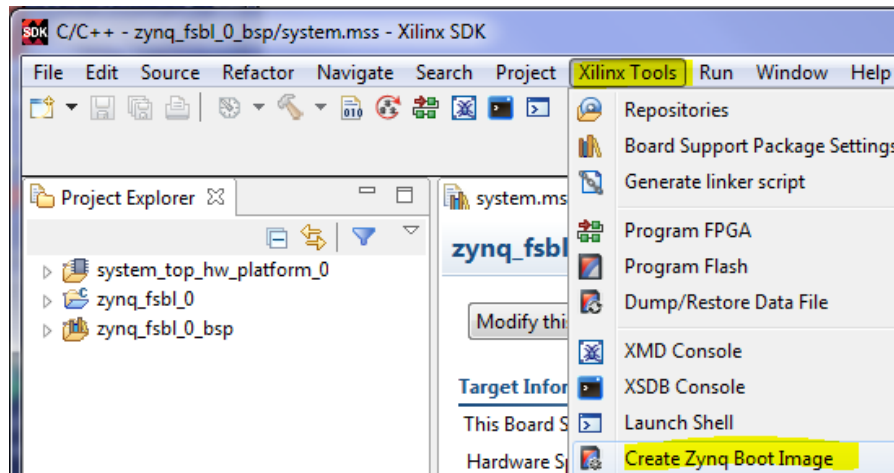


Figure 45 – Launching the Zynq Boot Image Tool

3. A new Boot Image Format (BIF) file must be created. Click the **Create new BIF file** radio button.

Click the **Browse** button to select a location for the new file. A convenient place is within the **boot** folder created earlier. Name the file **ubuntu.bif** to identify its purpose.

C:\mitx\Zynq_Ubuntu\boot\ubuntu.bif

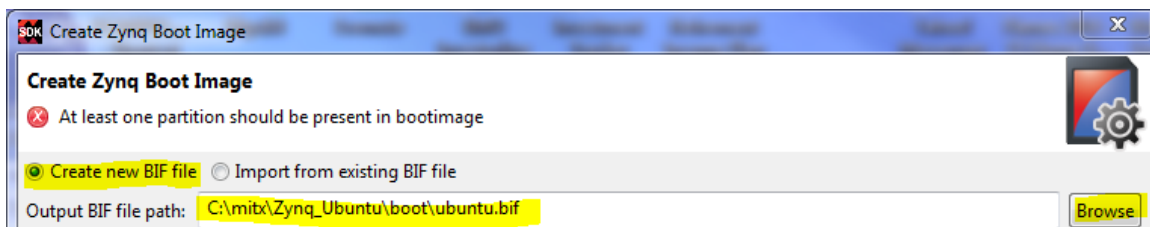


Figure 46 – Create a New BIF File

- The next step is to select the files that will make up our boot image. In the *Boot image partitions* section, click the **Add** button.



Figure 47 – Add Boot Image Partitions

- Click the **Browse** button to select the First Stage Boot Loader file from the saved location in Experiment 1. The Partition type must be set to **bootloader**.

C:\mitx\Zynq_Ubuntu\boot\zynq_fsbl_0.elf

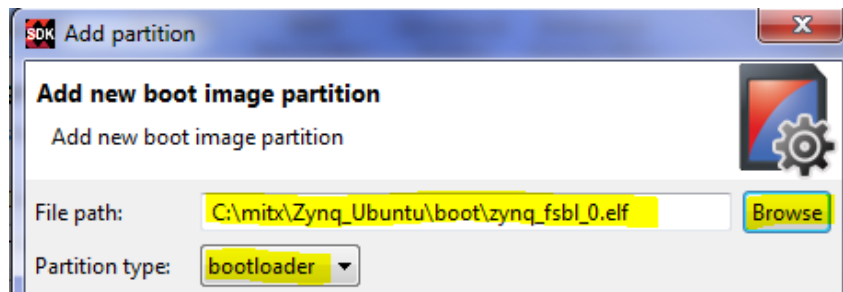


Figure 48 – Add First Stage Boot Loader

Click the **OK** button to accept this file.

- Keep in mind the importance of order for files placed into a boot image. Click the **Add** button to begin selection of the hardware bitstream.

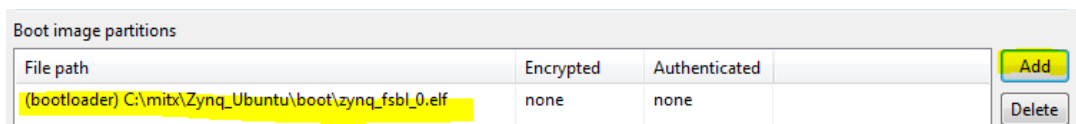


Figure 49 – Add Next Boot Image Partition

7. Ensure the *Partition type* is set to **datafile**. Click the **Browse** button to select the bitstream file from the saved location in Lab 1.

C:\mitx\Zynq_Ubuntu\boot\system_top.bit

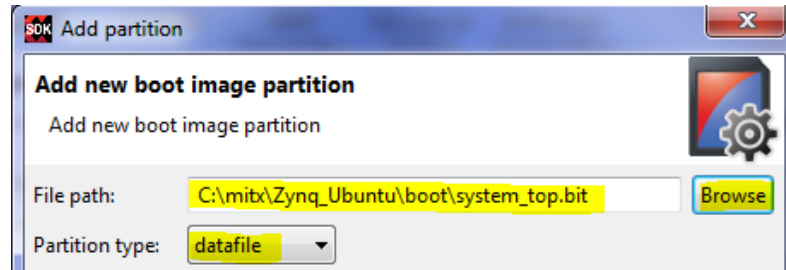


Figure 50 – Add Bitstream

Click the **OK** button to accept this file.

8. Specify the application executable last (in our case it is the second stage boot loader U-boot). Click the **Add** button and again ensure the *Partition type* is set to **datafile**. Click the **Browse** button to select the ELF file from the save location in Lab 2.

C:\mitx\Zynq_Ubuntu\boot\u-boot.elf

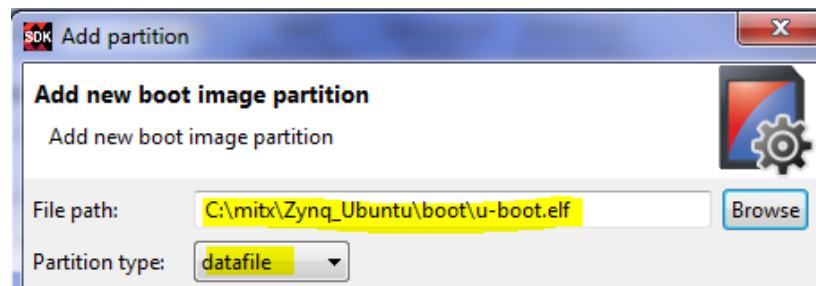


Figure 51 – Add U-Boot

Click the **OK** button to accept this file.

9. Check the *Output path* field and ensure the path is set to:

C:\mitx\Zynq_Ubuntu\boot\BOOT.bin

Use the **Browse** button adjacent to the *Output path* to alter the pathname if necessary.

Click the **Create Image** button to launch *Bootgen*, the tool for building a boot image for the Zynq-7000 All Programmable SoC.

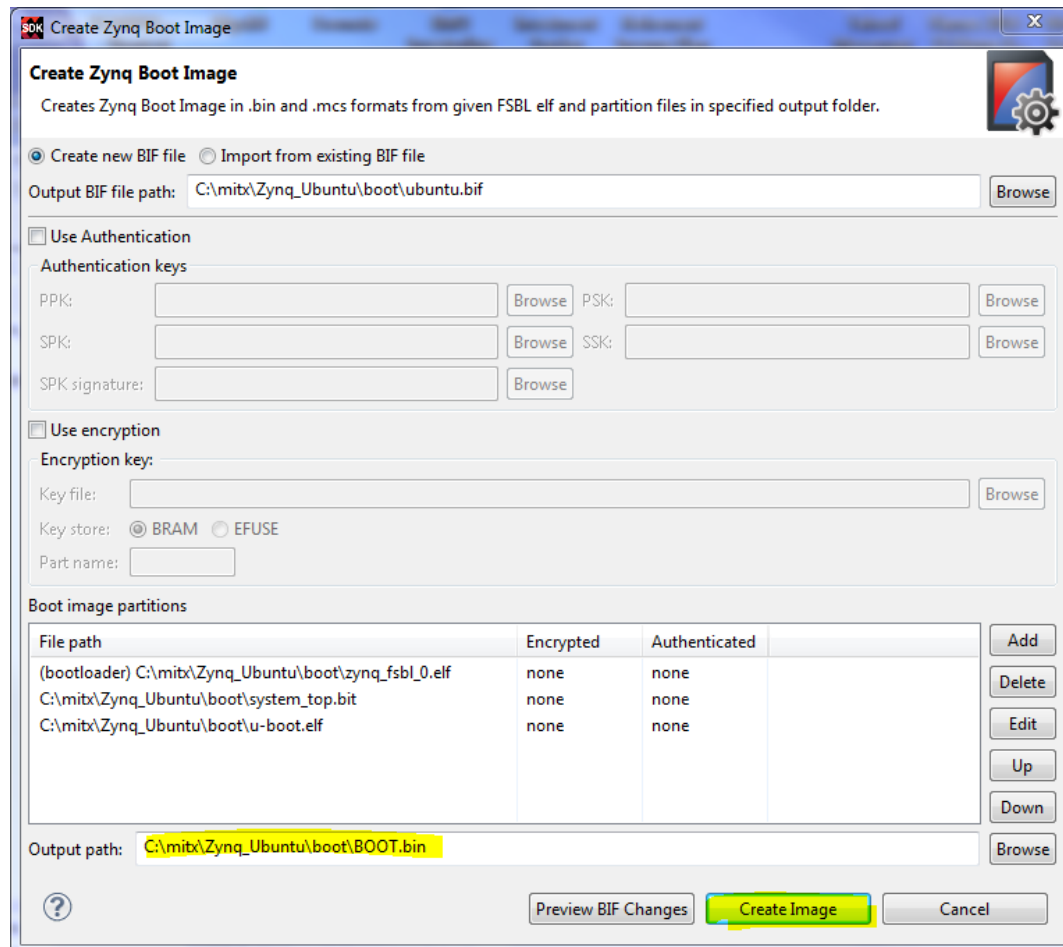
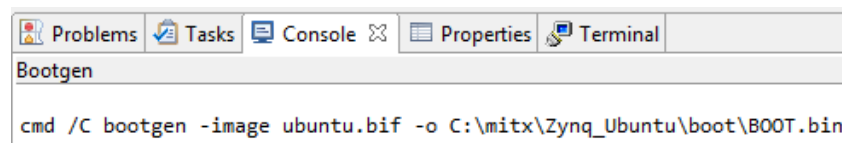


Figure 52 – Create Boot Image

10. Bootgen merges the BIT and ELF files into a single binary boot image using the partition format specified in the Boot Image Format (BIF) file.








```

cmd /C bootgen -image ubuntu.bif -o C:\mitx\Zynq_Ubuntu\boot\BOOT.bin
  
```

Figure 53 – Bootgen Complete

Your boot folder will now contain the following files:

Name	Date modified	Type	Size
 BOOT.bin	15/02/2015 1:06 PM	BIN File	13,495 KB
 system_top.bit	06/01/2015 1:01 PM	BIT File	13,010 KB
 u-boot.elf	12/02/2015 7:06 PM	ELF File	2,188 KB
 ubuntu.bif	14/02/2015 5:33 PM	BIF File	1 KB
 zynq_fsbl_0.elf	14/02/2015 4:57 PM	ELF File	388 KB

Boot.bin: Boot image file used to initialize the Zynq All Programmable SoC and execute u-boot.

system_top.bit: The bitstream used to load the Programmable Logic. This file is contained in a datafile partition in the boot image.

u-boot.elf: The second stage boot loader. This file is contained in a datafile partition in the boot image.

ubuntu.bif: The partition format of the boot image file. This file may be referenced in the Create Zynq Boot Image tool if you wish to recreate the boot image later.

zynq_fsbl_0.elf: The first stage boot loader that initializes the Processor System. This file is contained the bootloader partition in the boot image.

At this point you may close the Xilinx SDK.

11. Insert the SD/microSD card into the PC or card reader and wait for it to enumerate as a Windows drive. If prompted by Windows security software when inserting the card, select the **Continue without scanning** option. Select this option whenever this message appears after inserting your SD/microSD card.

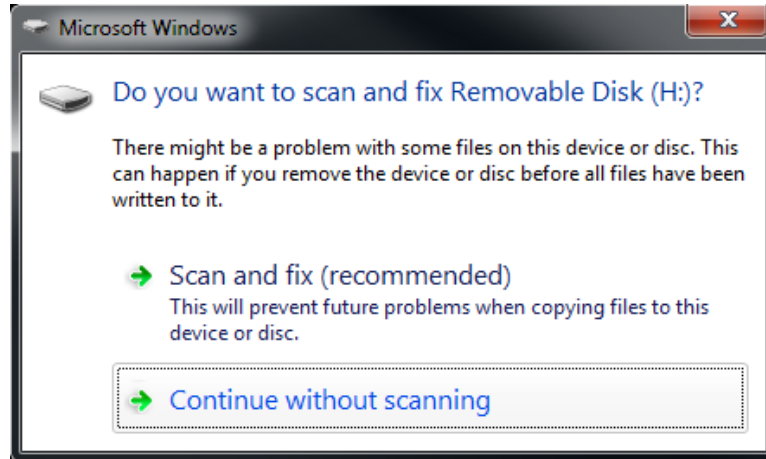


Figure 54 – Windows Prompt to Scan and Fix Removable Media

12. Copy the **Boot.bin** file from the *boot* folder to the top level of the media. Replace any existing versions of the **Boot.bin**⁸ file that may be on the card.

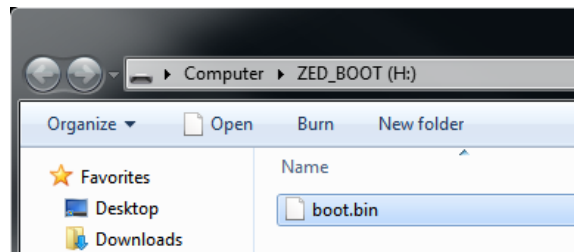


Figure 55 – The Boot Image File Copied to the SD Card

13. With the **boot.bin** file in place on the SD/microSD card, you may eject the device from your Windows host and remove the card.

⁸ Not case-sensitive.

Experiment 3: Prepare the Avnet Target for Booting

In this experiment we will set up the hardware with all connections necessary for booting from the SD/microSD card.

Experiment 3 General Instruction:

Make all necessary connections to the Zynq Mini-ITX in preparation for booting from the microSD card.

Experiment 3 Step-by-Step Instruction:

1. Connect the ATX power supply to the Mini-ITX, as shown in the figure below. The male four-connector cable from the power supply should be inserted first, followed by the larger connector. Apply downward pressure to the large connector until it snaps into place, securing the smaller connector when it does so.

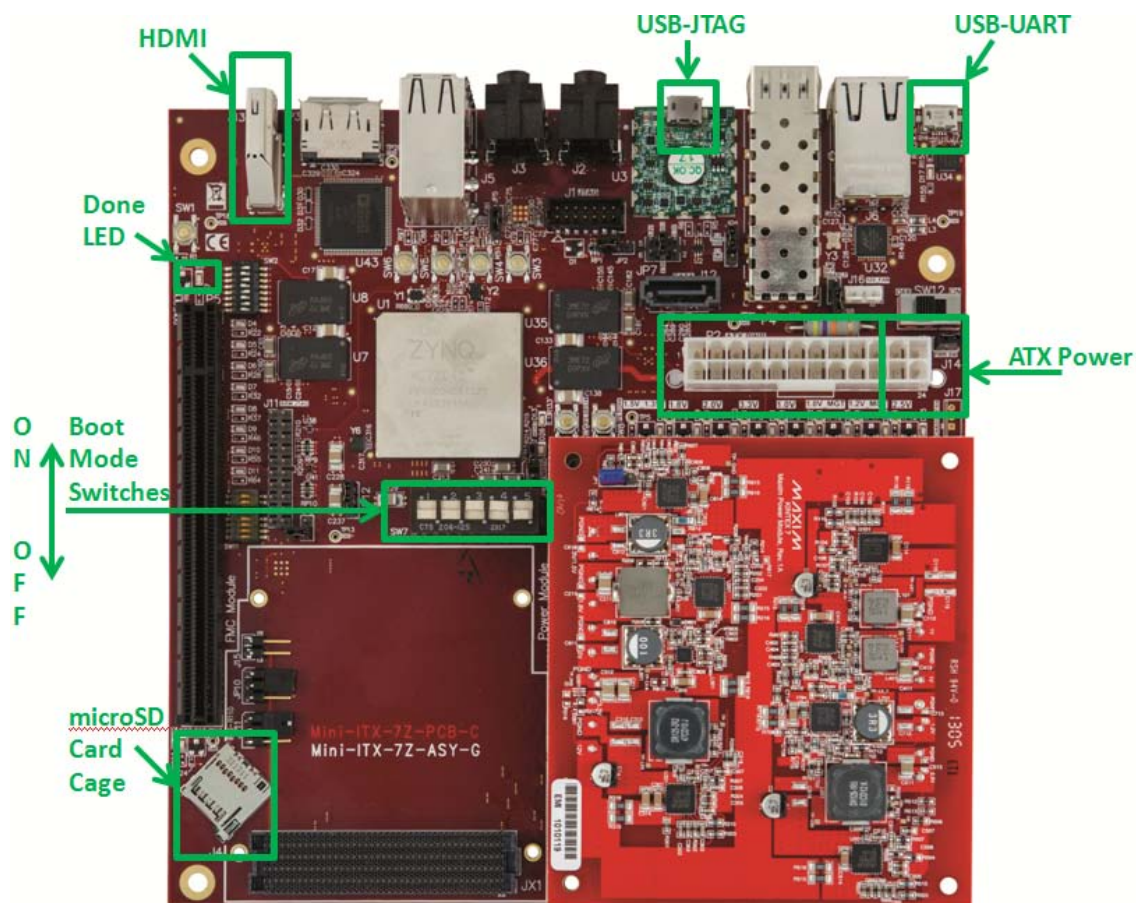


Figure 56 – Zynq Mini-ITX Board Components

2. Connect the USB-UART port of the Zynq Mini-ITX (J7), labeled UART, to a PC using a USB-to-microUSB cable.
3. Connect an HDMI cable between the HDMI monitor and the J13 connector on the Zynq Mini-ITX.
4. Verify the Mini-ITX boot mode switch block (SW7), labeled PS BOOT, are set to JTAG Boot mode as shown in the figure below. All Switches must be in the OFF position (away from the numbers).

microSD Boot Mode	JTAG Boot Mode
1: OFF	1: OFF
2: OFF	2: OFF
3: ON	3: OFF
4: ON	4: OFF
5: OFF	5: OFF

Figure 57 – Mode Switch Block (SW7) Settings

5. Turn power switch (SW12) to the ON position. The Zynq Mini-ITX will power on and the cooling fan on the Zynq device should start running. A row of 8 green LEDs next to the power module and a row of 8 red LEDs next to the PCIe connector will illuminate.
6. The PC may pop-up a dialog box asking for driver installation. The Zynq Mini-ITX has a USB-UART bridge based on the Silicon Labs CP210x chipset. Use of this feature requires that a USB driver be installed on your Host PC.

If Windows recognizes the USB-UART and loads the software driver, go ahead and proceed to the next step. However, if the host PC did not recognize the USB-UART and enumerate it as a COM port device refer to the *Silicon Labs CP210x USB-to-UART Setup Guide* in the link below for instructions on installing this driver. When driver installation is complete, continue to the next step.

<http://www.zedboard.org/support/documentation/2056>

When driver installation is complete, continue to the next step.

7. Use the Windows Device Manager to determine the COM Port.

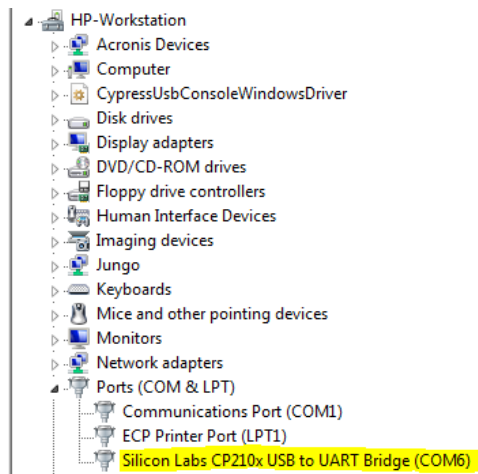


Figure 58 – Device Manager Showing Enumerated USB-UART

Note: Each unique USB-UART device attached will enumerate under the next available COM port. In this example, the Silicon Labs CP210x USB-to-UART device is enumerated as COM6.

Experiment 4: Boot the Avnet Target with Boot Image

The Avnet Zynq target can now be booted using the Boot Image that was copied to the SD/microSD card in the previous exercise.

Experiment 4 General Instruction:

Boot the Avnet target using the removable media card with the Zynq boot image and observe the terminal output on a console (Tera Term or similar) on your host system.

Experiment 4 Step-by-Step Instruction:

1. On the PC, open a serial terminal program. For this demo, Windows 7 was used which does not come with a built in terminal application such as HyperTerm. Tera Term was used in this example which can be downloaded from the Tera Term project on the SourceForge Japan page:

<http://ttssh2.sourceforge.jp>

2. Once Tera Term is installed, Tera Term can be accessed from the desktop or Start menu shortcuts.



Figure 59 – Tera Term Icon

3. To configure baud rate settings, open the Serial Port Setup window from the **Setup→Serial port** menu selection. Select the USB-UART COM port enumeration that matches the listing found in Device Manager.

Also set the Baud rate option to **115200**, the Data width option to **8-bit**, the Parity option to **none**, the Stop bit option to **1 bit**, and the flow control to **none**.

Finally, assign the transmit delay parameters to **10 msec/char** and **100 msec/line**, and then click **OK**. This setting will help with later lab exercises where many lines of text will be sent to the console in rapid succession.

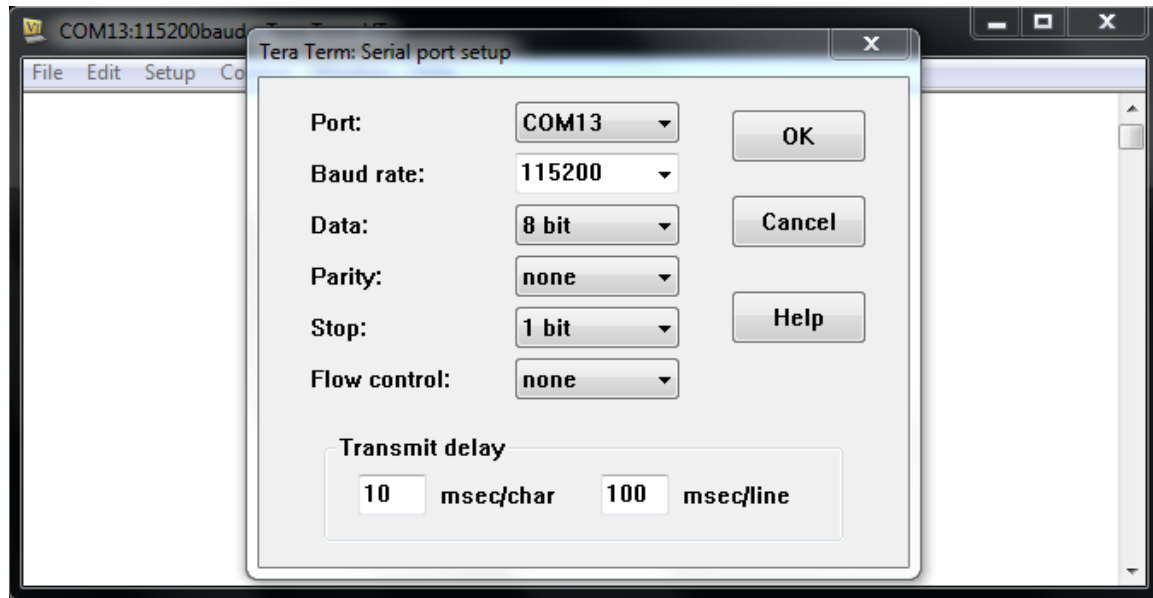


Figure 60 – Tera Term Serial Port Setup Page

4. Optionally, the terminal settings can be saved for later use. To do this, use the **Setup→Save** setup menu selection and overwrite the existing TERATERM.INI file.
5. Power cycle the Avnet target and monitor the Tera Term window carefully.

When the terminal output from U-boot and a countdown is observed, **press any of the keyboard keys to interrupt the boot process** and stop at the U-boot command prompt.

If you fail to interrupt the U-boot countdown, you will see that U-boot dutifully tries to look for a Linux kernel image and its associated device file, but is unable to locate it. At this point you will see the error:

**** Unable to read file ulmage ****

U-boot will return to its command prompt.

If the amber USB-Link Status (LD11) does not flicker to indicate activity, and no output is displayed on the terminal after 10 seconds, check the driver installation to determine if the device driver is recognized and enumerated successfully and that there are no errors reported by Windows.

Take a few minutes to explore the U-boot environment and command line options. Use the `?` command entry to display a listing of the supported U-boot commands. Use any of the listed commands to explore U-boot's capabilities.

```
COM3:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help

U-Boot 2014.01-dirty (May 27 2014 - 15:01:57)

I2C:   ready
Memory: ECC disabled
DRAM:  512 MiB
MMC:   zynq_sdhci: 0
SF: Detected S25FL256S_64K with page size 256 Bytes, erase size 64 KiB, total 32
MiB
In:     serial
Out:    serial
Err:    serial
Net:    Gem.e000b000
Hit any key to stop autoboot:  0
zed-uboot >
```

Figure 61 – Zynq U-boot Prompt

Note: If you have run U-boot on the Avnet target previously, and you saved the environment variables using the **saveenv** command, you may wish to update the non-volatile memory at this time so the new **sdboot** command will be used. When we changed the U-boot source code, we updated the default environment, but U-boot only uses that when it cannot locate environment variables in non-volatile memory.

At the zynq-uboot prompt, enter:

```
env default -a -f
printenv sdboot
```

Check to make sure the sdboot command reflects our source changes. If you are sure it is correct, save the environment by entering:

```
saveenv
```

```
COM3:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help

Net:   zynq_gem
Hit any key to stop autoboot:  0
zynq-uboot> env default -a -f
## Resetting to default environment
zynq-uboot> printenv sdboot
sdboot=echo Copying Linux kernel from SD to RAM...RFS in ext4:mmcinfo;fatload mmc
c 0 0x30000000 ${kernel_image};fatload mmc 0 0x2A000000 ${devicetree_image};bootm
0x30000000 - 0x2A000000
zynq-uboot> saveenv
Saving Environment to SPI Flash...
SF: Detected S25FL256S with page size 64 KiB, total 32 MiB
SF: Warning - Only lower 16MB is accessible in 3 byte addressing mode
Erasing SPI flash...SF: Successfully erased 131072 bytes @ 0xe00000
Writing to SPI flash...SF: program success 131072 bytes @ 0xe00000
done
zynq-uboot>
```


Lab 5 – Configure and Build the Linux Kernel

Lab Overview

Analog Devices (ADI) maintains a freely downloadable Linux kernel that has been tested on Xilinx Zynq-7000 Programmable SoC development boards, and is suitable for hosting the Ubuntu Desktop. The source files are hosted on an open source repository site called GitHub.

In this lab, the process to rebuild the kernel from the source repository is explored.

When you have completed this lab, you will learn to do the following:

- Retrieve Linux kernel source code from the GitHub repository
- Configure the kernel for the Avnet target
- Build the kernel for the Avnet target

Experiment 1: Clone the ADI Linux Kernel Git Repository

This experiment shows how to make a local copy of the ADI Linux kernel Git repository for Zynq. To successfully complete this lab, you will need Internet access to retrieve the repository information from the GitHub website.

Experiment 1 General Instruction:

Make a local copy of the ADI Linux kernel Git repository in your home directory, and check out the branch we intend to build.

On your Linux host, enter the following commands:

```
$ cd ~  
  
$ git clone \  
https://github.com/analogdevicesinc/linux.git ubuntu  
  
$ cd ubuntu  
  
$ git checkout xcomm_zynq
```

Experiment 1 Step-by-Step Instruction:

1. If the virtual machine is not already open, launch the Oracle VM VirtualBox application by following Steps 1 through 4 in Lab 2, Experiment 1.
2. The Linux Git repository is located at the following URL:

`git://github.com/analogdevicesinc/linux.git`

To get a working copy of the codebase, clone the remote repository to your local machine. Cloning creates the repository and checks out the latest version, which is referred to as HEAD.

Use the following commands to clone the repository. Make sure you use the **https:** directive instead of the old **git:** string. Failure to do this will cause the process to hang without any indication as to the cause.

```
$ cd ~  
  
$ git clone https://github.com/analogdevicesinc/linux.git ubuntu  
  
[training@localhost ~]$ git clone https://github.com/analogdevicesinc/linux.git  
ubuntu  
Cloning into 'ubuntu'...  
remote: Counting objects: 3922649, done.  
remote: Compressing objects: 100% (69/69), done.  
Receiving objects: 0% (19194/3922649), 7.07 MiB | 843.00 KiB/s
```

3. Wait until the clone operation completes, this could take an hour or more depending upon your connection speed.

The clone command sets up a few convenient items for you by:

- Keeping the address of the original repository
- Aliasing the address of the original repository as origin so that changes can be easily sent back (if you have authorization) to the remote repository

This copies the repository to a new directory at **/home/training/ubuntu**.

4. Change into the top level source folder.

```
$ cd ubuntu
```

5. Check out the code branch compatible with the FPGA platform⁹. This sets up the **xcomm_zynq** branch for remote tracking.

```
$ git checkout xcomm_zynq
```

Experiment 2: Configure and Build the Linux Kernel

This experiment shows how to configure the source branch to target the Xilinx Zynq SoC, including extensions for the Analog Devices ADV7511 HDMI transmitter and ADAU1761 Audio Codec, and to build an executable file.

Experiment 2 General Instruction:

Clean, configure and build the Linux kernel for the ARM architecture of the Zynq SoC.

On your Linux host, enter the following commands:

```
$ make ARCH=arm distclean  
$ make ARCH=arm zynq_xcomm_adv7511_defconfig  
$ make ARCH=arm uImage LOADADDR=0x8000
```

Experiment 2 Step-by-Step Instruction:

1. Change from the home directory into the ADI Linux kernel source directory.

```
$ cd ~/ubuntu/
```

2. For good measure (sometimes a necessity), run a make distribution clean command against the kernel source code. This command will remove all intermediary files created by config as well as any intermediary files created by make and it is a good way to clean up any stale configurations.

```
$ make ARCH=arm distclean
```

⁹ The kernel branch referenced here is compatible with Xilinx tools release 14.2 and higher.

3. A Zynq configuration file is included for Zynq targets using the ADI adv7511 HDMI video transmitter on the board. The file is located at:

`/arch/arm/configs/zynq_xcomm_adv7511_defconfig`

This file defines a common configuration for a number of ADI reference designs that may or may not include HDMI video and wireless communication. Our design includes HDMI video, but no wireless comms. Configure the Linux Kernel for the Zynq target by using this default configuration file.

```
$ make ARCH=arm zynq_xcomm_adv7511_defconfig
```

```
[training@localhost ubuntu]$ make ARCH=arm zynq_xcomm_adv7511_defconfig
HOSTCC  scripts/basic/fixdep
HOSTCC  scripts/kconfig/conf.o
SHIPPED scripts/kconfig/zconf.tab.c
SHIPPED scripts/kconfig/zconf.lex.c
SHIPPED scripts/kconfig/zconf.hash.c
HOSTCC  scripts/kconfig/zconf.tab.o
HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
[training@localhost ubuntu]$
```

Figure 62 – Kernel Configuration

4. Build the kernel source and generate the compressed image file expected by our version of U-boot with the make command.

The build process should take about 15 to 25 minutes to complete. If the build is successful and the console output looks similar to that shown in the Figure below, proceed to the next step.

```
$ make ARCH=arm uImage LOADADDR=0x8000
```

```
training@localhost:~/ubuntu
File Edit View Search Terminal Help
CC      arch/arm/boot/compressed/misc.o
CC      arch/arm/boot/compressed/decompress.o
AS      arch/arm/boot/compressed/debug.o
CC      arch/arm/boot/compressed/string.o
SHIPPED arch/arm/boot/compressed/hyp-stub.S
AS      arch/arm/boot/compressed/hyp-stub.o
SHIPPED arch/arm/boot/compressed/lib1funcs.S
AS      arch/arm/boot/compressed/lib1funcs.o
SHIPPED arch/arm/boot/compressed/ashldi3.S
AS      arch/arm/boot/compressed/ashldi3.o
SHIPPED arch/arm/boot/compressed/bswapsdi2.S
AS      arch/arm/boot/compressed/bswapsdi2.o
LD      arch/arm/boot/compressed/vmlinux
OBJCOPY arch/arm/boot/zImage
Kernel: arch/arm/boot/zImage is ready
UIMAGE arch/arm/boot/uImage
Image Name:   Linux-3.18.0-gf4fce61e
Created:      Tue Feb 17 15:09:56 2015
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    3225968 Bytes = 3150.36 kB = 3.08 MB
Load Address: 00008000
Entry Point:  00008000
Image arch/arm/boot/uImage is ready
[training@localhost ubuntu]$
```

Figure 63 – Linux Kernel Build Completed

If your build fails to locate the *mkimage* tool, please see *Lab 2, Experiment 3, Step 3* to correct the problem. You may rebuild the entire kernel again, or you can manually create the image with the following commands:

```
cd arch/arm/boot
gzip -9 zImage
mkimage -A arm -a 0x8000 -e 0x8000 -n 'Linux kernel' \
-T kernel -d zImage.gz uImage
```

7. If a file browser window is not already open from a previous exercise, *double-click* on the home icon on your desktop to open one.

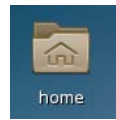


Figure 64 – File Browser Icon

5. Locate the file **/home/training/ubuntu/arch/arm/boot/ulmage**, which is the target executable image needed by U-boot on Zynq. Copy this file into the Virtual Machine shared folder to make it accessible from the Windows host.

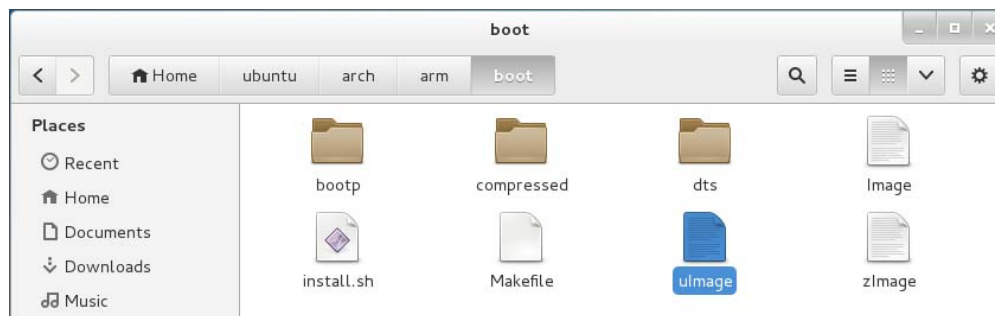


Figure 65 – Kernel ulmage

6. Use Windows Explorer on the host system to copy the kernel image from the shared directory to the following folder:

C:\mitx\Zynq_Ubuntu\boot

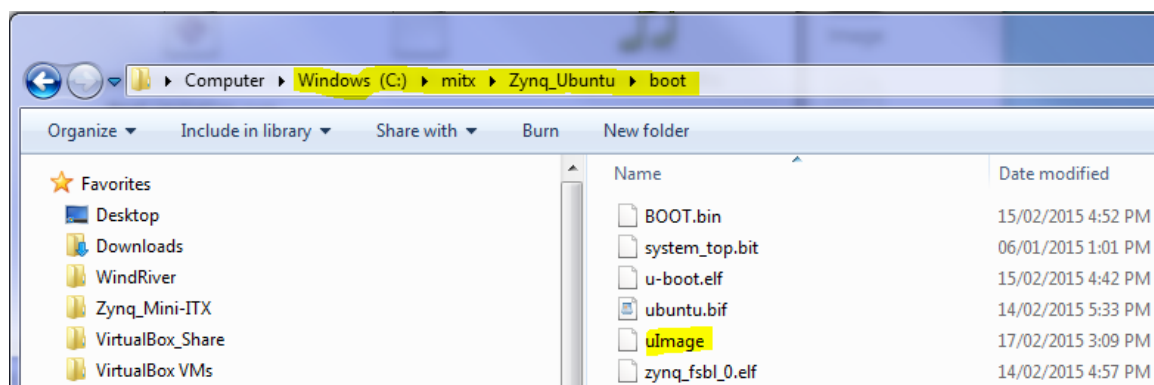
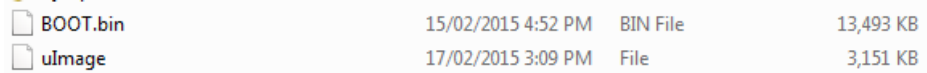


Figure 66 – Kernel ulmage Copied to the Host Machine

7. Insert the microSD card into the PC or SD card reader and wait for it to enumerate as a Windows drive.
8. Copy the **ulmage** file from the **boot** folder to the top level of the SD card. Replace any existing versions of the *ulmage* file that may be on the card.





 BOOT.bin	15/02/2015 4:52 PM	BIN File	13,493 KB
 ulmage	17/02/2015 3:09 PM	File	3,151 KB

Figure 67 – microSD Card Contents

Lab 6 – Install the Device Tree

Lab Overview

The Device Tree is a file which contains a data structure describing the hardware system. The information is used by Linux during the kernel boot process to map device parameters such as device type, memory location and interrupt signals. Although we have already initialized and booted the ARM processor cores into a standalone running state, Linux has no knowledge of this and by default assumes that it must perform all initialization on its own. To do this it needs to set up virtual memory, print to the console, and locate all of the installed hardware in the system and load software drivers.

These operations are carried out by writing to registers, but the Linux kernel needs a method to discover the parameters associated with the current hardware system. In a fixed system this could be handled with static header files or kernel configuration at build time, but for many dynamic devices it is much more desirable to obtain the configuration information at run time. This is further complicated by the endless customization available on an FPGA, which would quickly result in an unmanageable number of possible kernel header and configuration combinations.

On a PC, the device information is supplied by the BIOS, but the ARM processors don't have anything comparable.

So the chosen solution is a **device tree**, also referred to as **Open Firmware** (abbreviated OF) or **Flattened Device Tree** (FDT). This is a text file which contains all the hardware information about the system, such as device addresses, interrupt vectors and bus addresses, compiled into byte code format. U-boot copies the compiled data into a known address in RAM before jumping to the kernel's entry point.

The Ubuntu top level device tree source file for the Avnet Zynq Mini-ITX targets is located in the Git kernel repository at:

```
~/ubuntu/arch/arm/boot/dts/zynq-mini-itx-adv7511.dts
```

In the 2014.4 release, this file is simply used to add two lower level files to the devicetree hierarchy. The entire file contents are shown here:

```
/dts-v1/;  
  
/include/ "zynq-mini-itx.dtsi"  
/include/ "zynq-mini-itx-adv7511.dtsi"
```

Figure 68 – Top Level zynq-mini-itx-adv7511.dts

The *zynq-mini-itx.dtsi* file contains information specific to the Zynq Mini-ITX board, such as the amount of DDR, the make and model of Ethernet PHY and the location of the USB reset line. This file does not typically require modification once a board is in production.

The Zynq Mini-ITX specific device tree file is found at:

`~/ubuntu/arch/arm/boot/dts/zynq-mini-itx.dtsi`

The elements common to all Zynq devices are represented by the *zynq.dtsi* file, which is included as the first line of the *zynq-mini-itx.dtsi*. This makes sense as the internal makeup of the Zynq device does not change, so it becomes a static part of the device tree that should not change for a production board.

The Zynq common layer device tree file is found at:

`~/ubuntu/arch/arm/boot/dts/zynq.dtsi`

Finally, *zynq-mini-itx-adv7511.dtsi* holds details specific to the Programmable Logic design loaded to the device, and also information for on-board components that are connected via the PL fabric. This file changes if any IP addressable by Linux is added or removed from the design, or if existing external IP interfaces is altered.

The design-specific device tree file for the Zynq Mini-ITX is located at:

`~/ubuntu/arch/arm/boot/dts/zynq-mini-itx-adv7511.dtsi`

When you have completed this lab, you will have learned to:

- Compile the device tree source files for use by the Linux kernel at boot time

Note: At the time this documentation was updated, the *zynq-mini-itx-adv7511.dtsi* file included in the Linux source hierarchy does not include entries for the ADAU1761 audio codec. To allow this device to operate correctly, substitute the device tree file included in **Supplied Files**.

Experiment 1: Compile the Device Tree Source Files

This experiment shows how to compile the device tree source files to a single binary output product and copy it to your host system for inclusion in the boot directory.

Experiment 1 General Instruction:

Use the makefile in the local *ubuntu* repository to compile the device tree source file to its binary equivalent.

On your Linux host, enter the following commands:

```
$ cd ~/ubuntu
```

```
$ make ARCH=arm zynq-mini-itx-adv7511.dtb
```

Rename the result file to **devicetree.dtb** and copy it from the output directory to the boot file directory on the host machine.

Experiment 1 Step-by-Step Instruction:

1. The location of the device tree source is hard-coded into the makefile in the root directory of your local *ubuntu* repository. You can compile the device tree by providing an output file name in the root directory that matches the name of the top level source file (the only difference is the .dtb versus .dts endings).

```
$ cd ~/ubuntu
```

```
$ make ARCH=arm zynq-mini-itx-adv7511.dtb
```

```
[training@localhost ubuntu]$ make ARCH=arm zynq-mini-itx-adv7511.dtb
DTC      arch/arm/boot/dts/zynq-mini-itx-adv7511.dtb
```

The binary (*.dtb*) is created in the same directory as the source (*.dts*) file.

Output:

```
DTC arch/arm/boot/dts/zynq-mini-itx-adv7511.dtb
```

2. Rename the compiled device tree to the name expected by U-boot when it copies the file to RAM at load time.

```
$ cd arch/arm/boot/dts
$ mv zynq-mitx-adv7511.dtb devicetree.dtb
```

```
[training@localhost ubuntu]$ cd arch/arm/boot/dts
[training@localhost dts]$ mv zynq-mini-itx-adv7511.dtb devicetree.dtb
```

8. If a file browser window is not already open from a previous exercise, *double-click* on the home icon on your desktop to open one.

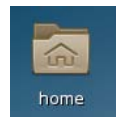


Figure 69 – File Browser Icon

9. Locate the file **/home/training/ubuntu/arch/arm/boot/dts/devicetree.dtb**. Copy this file into the Virtual Machine shared folder to make it accessible from the Windows host.

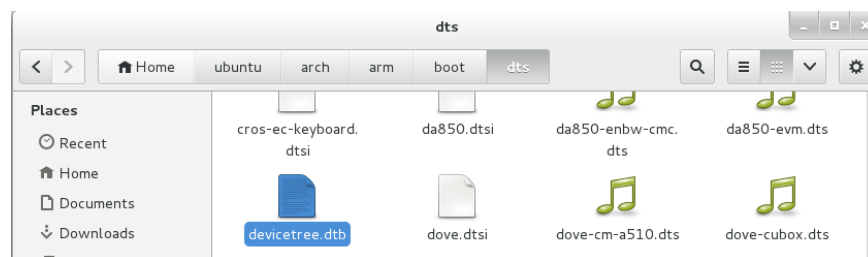


Figure 70 – Compiled devicetree

10. Use Windows Explorer on the host system to copy *devicetree.dtb* from the shared directory to the following folder:

C:\mitx\Zynq_Ubuntu\boot

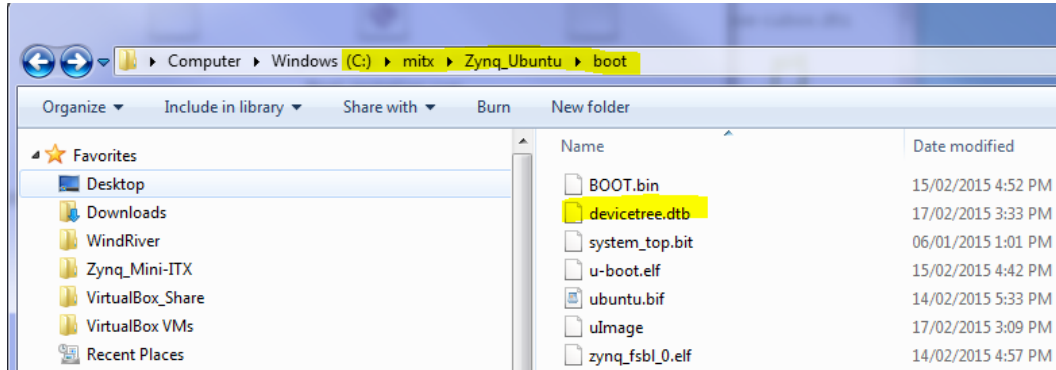


Figure 71 – Compiled devicetree Copied to the Host Machine

11. Insert the microSD card into the PC or SD card reader and wait for it to enumerate as a Windows drive.
12. Copy the **devicetree.dtb** file from the **boot** folder to the top level of the SD card. Replace any existing versions of the *devicetree.dtb* file that may be on the card.

	BOOT.bin	15/02/2015 4:52 PM	BIN File	13,493 KB
	uImage	17/02/2015 3:09 PM	File	3,151 KB
	devicetree.dtb	17/02/2015 3:33 PM	DTB File	11 KB

Figure 72 – microSD Card Contents

Lab 7 – Obtain and Load the Root File System

Lab Overview

The Root File System is contained on the same partition as the root directory in the Linux kernel, and it is the file system on which all other file systems are mounted. The RFS is separate from the Linux kernel, but is required by the kernel to complete the boot process. All file data used by Linux and any user accounts is contained in the Root File System.

The separation of kernel and file system adds additional flexibility to Linux distributions. Distributions largely use a common Linux kernel (with slight variations to support specific elements of local importance), but the large part of what makes distributions unique is determined by the makeup of the RFS. In the case of Ubuntu and Android, for example, both use a common kernel but include feature differences at the root file system level. For Ubuntu, RFS elements provide a desktop-style graphical UI, while Android relies on a Java Virtual Machine to allow for ease of application development. Both systems can co-exist on the same hardware platform because they make use of a common Linux kernel.

The creation of a complete Root File System from scratch is a complex procedure, and beyond the scope of this tutorial. However, if you wish to understand how an RFS is constructed, refer to the Avnet Speedway [Implementing Linux on the Zynq-7000 SoC](#), Lab 2.2, *Creating a Basic Root File System*. The Speedway Lab provides step by step instructions for creating a Root File System from scratch, including creating a complete development environment. While the content of Ubuntu and Android may differ, the process for importing the required features is the same.

In general, it is best to start with a Root File System that contains much of what you already need for your implementation, and then customize it to your specific requirements. Linaro provides a rich root file system that will coexist with our kernel to provide a desktop experience, which serves the purposes of this tutorial.

When you have completed this lab, you will know how to do the following:

- Obtain the Root File System image from the Linaro website
- Install the Root File System to an SD card

You will need Internet access from your Linux host to retrieve the RFS.

Experiment 1: Download and Install the Root File System

Experiment 1 General Instruction:

Download the RFS image from the Linaro website, and install it on a pre-formatted SD card that will be used to boot Ubuntu on the ZedBoard.

On your Linux host, enter the following commands:

```
$ cd ~

$ wget http://releases.linaro.org/12.11/ubuntu/precise-
images/ubuntu-desktop/linaro-precise-ubuntu-desktop-
20121124-560.tar.gz

$ sudo tar --strip-components=3 -C /media/rootfs -xzf \
  linaro-precise-ubuntu-desktop-20121124-560.tar.gz \
  binary/boot/filesystem.dir
```

Experiment 2 Step-by-Step Instruction:

1. Download a prebuilt root file system image from Linaro to your local user top-level directory in the virtual machine. The suggested version (known working) is at the URL below:

```
$ cd ~

$ wget
http://releases.linaro.org/12.11/ubuntu/precise-
images/ubuntu-desktop/linaro-precise-ubuntu-desktop-
20121124-560.tar.gz
```

Depending on your connection speed, this download may take 10-30 minutes.

```
[training@localhost dts]$ cd ~
[training@localhost ~]$ wget http://releases.linaro.org/12.11/ubuntu/precise-image
s/ubuntu-desktop/linaro-precise-ubuntu-desktop-20121124-560.tar.gz
--2015-02-17 16:08:22-- http://releases.linaro.org/12.11/ubuntu/precise-images/ub
untu-desktop/linaro-precise-ubuntu-desktop-20121124-560.tar.gz
Resolving releases.linaro.org (releases.linaro.org)... 107.20.81.227
Connecting to releases.linaro.org (releases.linaro.org)|107.20.81.227|:80... conne
cted.
HTTP request sent, awaiting response... 200 OK
Length: 495927469 (473M) [application/x-tar]
Saving to: 'linaro-precise-ubuntu-desktop-20121124-560.tar.gz'

100%[=====>] 495,927,469 818KB/s in 7m 32s

2015-02-17 16:15:54 (1.05 MB/s) - 'linaro-precise-ubuntu-desktop-20121124-560.tar.
gz' saved [495927469/495927469]
```

1. If not already connected, insert the SD card into a compatible read/write slot on your computer.
2. In the main menu of VirtualBox, select:

Devices → USB Devices → Generic USB Storage

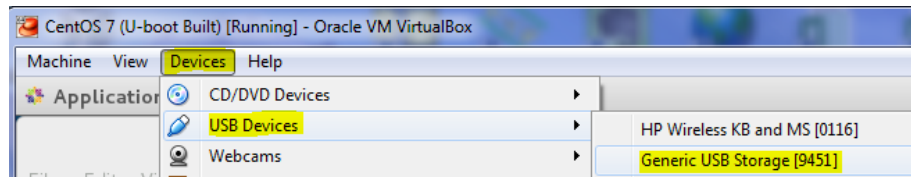


Figure 73 – Select USB Storage Device

3. Click on **Open with Files** to mount the media and open a browser window.

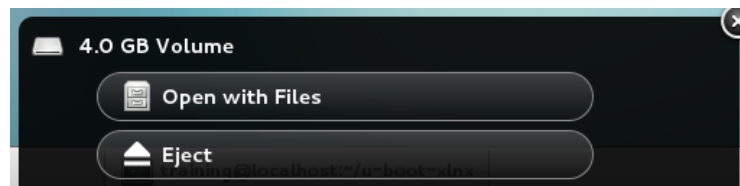


Figure 74 – USB Storage Device Detected by Virtual Machine

Partitions on the media will mount in the directory:

`/run/media/training/`

4. In a Terminal window, extract the root file system onto the SD card. This process will take 5 to 15 minutes to complete. Note that there is no visual progress indicator.

```
$ sudo tar --strip-components=3 -C
/run/media/training/rootfs -xzip linaro-precise-
ubuntu-desktop-20121124-560.tar.gz
binary/boot/filesystem.dir
```

```
[training@localhost ~]$ sudo tar --strip-components=3 -C /run/media/training/rootf
s -xzip linaro-precise-ubuntu-desktop-20121124-560.tar.gz binary/boot/filesystem.d
ir
[sudo] password for training:
[training@localhost ~]$
```

5. When the root file installation is complete, you can view the contents of the ext4 partition by selecting **rootfs** from the Devices area in the browser window that was opened during the mount.

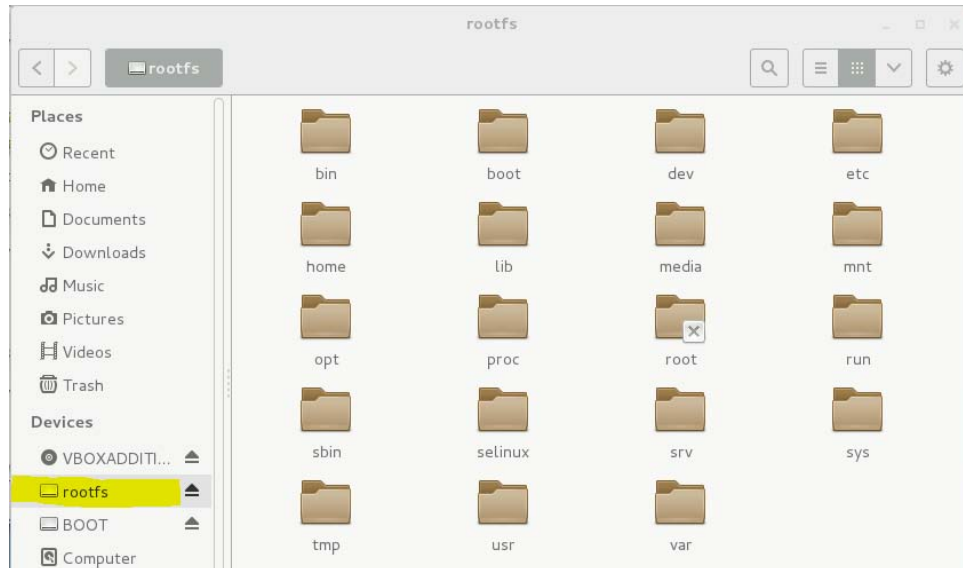


Figure 75 – rootfs Installed on SD Card

6. Disconnect the USB device from the virtual machine. In the main menu of VirtualBox select:

Devices → USB Devices → Generic USB Storage

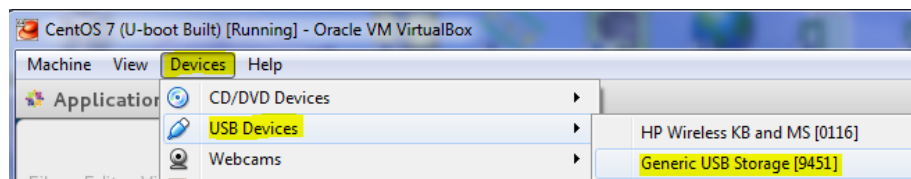


Figure 76 – Disconnect USB Storage Device from VM

7. Back in Windows you can open Windows Explorer to locate the removable drive, which will now appear as a FAT32 drive named *BOOT*. *Right-click* on *BOOT*, select **Eject** and you can remove the media.

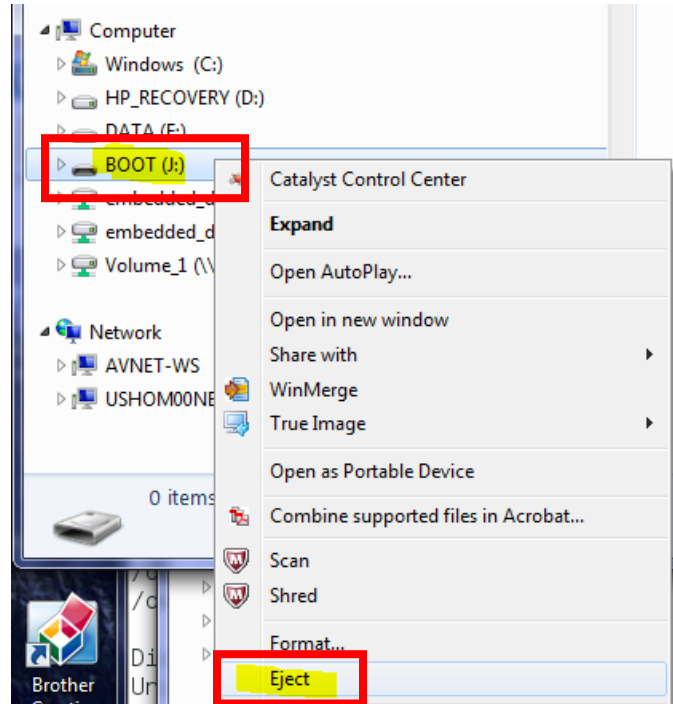


Figure 77 – Remove SD Card from Host

Lab 8 – Boot Ubuntu

Lab Overview

If you have successfully completed all the preceding labs, you will have an SD/microSD card configured with 2 partitions. The first partition is in FAT32 format, accessible from a Windows or Linux host, and it contains the files required to initialize the board and start the Linux kernel booting. The second partition is in ext4 format, invisible to Windows but used by Linux, and it holds the root file system created by Linaro.

The root file system contains the Ubuntu desktop environment. This higher level software uses a common Linux kernel, and can therefore coexist on the same processor system with other higher level packages, such as an Android Java VM. In fact, this is the basis for much of the current development in the mobile marketplace; the initiative to replace all existing computer systems with mobile devices has begun. If a mobile phone can have a sufficiently powerful processor system, there is no reason it cannot act as a computer in its own right. An Android OS provides the app-rich environment currently familiar to high-end smartphone owners, while the Ubuntu desktop allows access to applications normally seen only on PCs or tablets. This merging of the mainstream computing with the mobile world is the next step forward in portable computing.

When you have completed Lab 8, you will have learned to:

- Configure the Avnet target to boot from an SD/microSD card
- Connect all the external devices necessary to allow the Avnet target to function as a desktop computer.
- Boot the Ubuntu desktop to provide a graphical desktop environment
- Make post-installation changes to the environment to add an updated video driver, and to correct an issue with the audio codec

Experiment 1: Boot the Avnet Target to the Ubuntu Desktop

This experiment describes the device connections required for a desktop boot. Initial Board setup will be the same as in Lab 4 - Create the First Stage Boot Loader. You will need the following additional equipment for this lab:

- HDMI or DVI capable monitor capable of 1600 X 1200 resolution
- USB 2.0 Powered Hub (ZedBoard only)
- USB Keyboard
- USB Mouse
- HDMI cable, with HDMI to DVI adapter if a DVI monitor is used

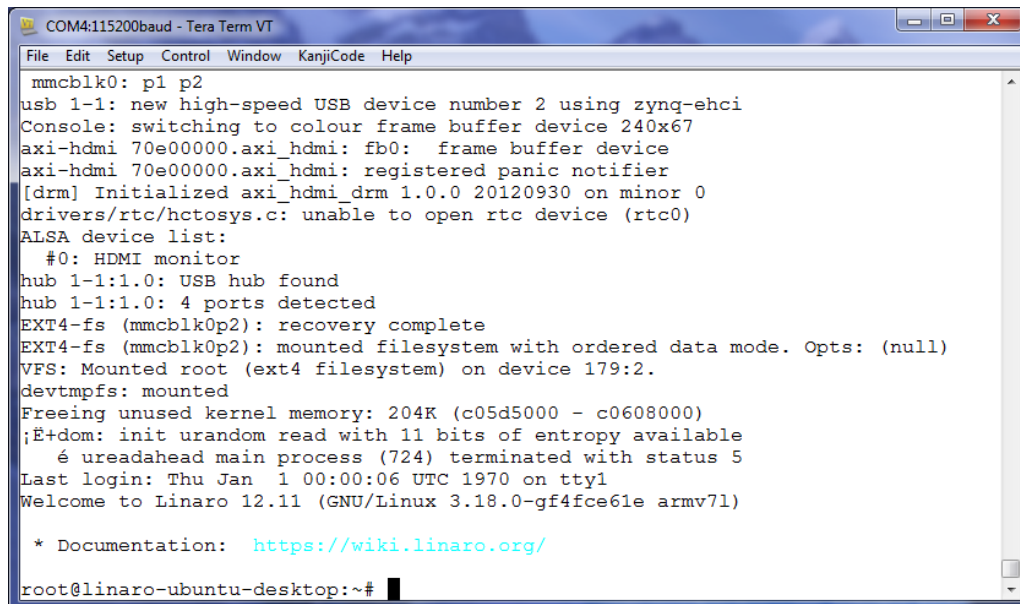
Experiment 1 General Instruction:

Configure the target for booting from the SD card. Connect the external peripherals necessary to use the system as a desktop computer. At a minimum you need a keyboard, a mouse, an HDMI monitor and a serial connection for a console. Apply power to the board to boot the desktop.

Experiment 1 Step-by-Step Instruction:

1. Configure the Avnet target for booting from the SD/microSD card, as described in **Lab 4 - Create the First Stage Boot Loader**.
2. Connect the HDMI/DVI monitor to the HDMI output on the target using the HDMI cable, and adapter if required. Turn the monitor on.
3. Connect the USB keyboard and USB mouse to two of the USB ports in the USB stack on the Zynq Mini-ITX board.
4. If you have not done so as part of Step 1, connect the USB serial port to your host system for use with Tera Term (or equivalent) as the boot console and insert the microSD card into the cage on the Zynq Mini-ITX.

5. Apply power to the target, and wait for the blue configuration LED to illuminate. Connect Tera Term to the USB-UART COM port. You will see U-boot load the images from the SD/microSD card and transfer control to the Linux kernel. The remainder of the console output comes from the kernel boot process, which will end with a command prompt.



```
COM4:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help
mmcblk0: p1 p2
usb 1-1: new high-speed USB device number 2 using zynq-ehci
Console: switching to colour frame buffer device 240x67
axi-hdmi 70e00000.axi_hdmi: fb0: frame buffer device
axi-hdmi 70e00000.axi_hdmi: registered panic notifier
[drm] Initialized axi_hdmi_drm 1.0.0 20120930 on minor 0
drivers/rtc/rtc-sys.c: unable to open rtc device (rtc0)
ALSA device list:
#0: HDMI monitor
hub 1-1:1.0: USB hub found
hub 1-1:1.0: 4 ports detected
EXT4-fs (mmcblk0p2): recovery complete
EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null)
VFS: Mounted root (ext4 filesystem) on device 179:2.
devtmpfs: mounted
Freeing unused kernel memory: 204K (c05d5000 - c0608000)
;E+dom: init urandom read with 11 bits of entropy available
     é ureadahead main process (724) terminated with status 5
Last login: Thu Jan  1 00:00:06 UTC 1970 on tty1
Welcome to Linaro 12.11 (GNU/Linux 3.18.0-gf4fce61e armv7l)

* Documentation: https://wiki.linaro.org/

root@linaro-ubuntu-desktop:~#
```

Figure 78 – Linaro Console Boot

6. Once the command prompt is displayed, you will see the Ubuntu desktop display on the HDMI monitor. You can interact with the desktop via the USB mouse and keyboard. Note that the screen resolution may not be optimal – this is addressed in the next section.

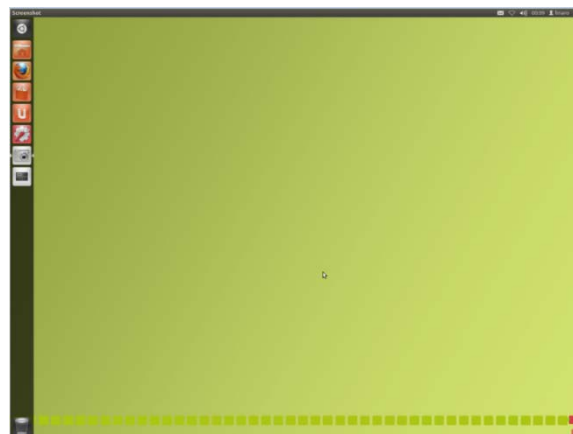


Figure 79 – Ubuntu Default Desktop

7. There are a couple of deficiencies in this installation, as documented on the ADI wiki site:

<http://wiki.analog.com/resources/tools-software/linux-drivers/platforms/zynq>

You may check this URL for the latest information, but at the time of writing the following information was provided for video and audio modifications:

Post-installation tweaks

After the system has been installed it is time to do some post-installation tweaks to the system. None of them are required to get a basic working system, but they improve the overall video and audio experience quite a bit.

Set the System Clock

The video configuration in the following section may fail if the system clock is not current, as new files will be created with a timestamp that is older than the files they will replace. To set the system clock, click on the time displayed in the upper right of the monitor.

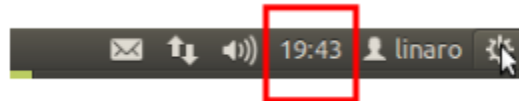


Figure 80 – Ubuntu Desktop Clock

A calendar will be displayed. At the bottom, click on the **Time and Date Settings**.

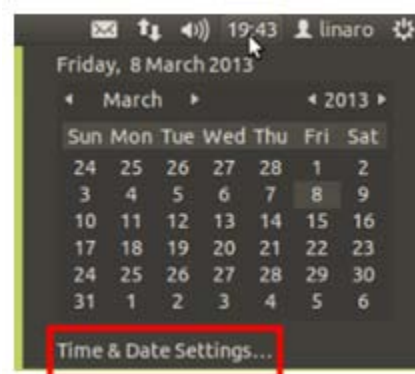


Figure 81 – Ubuntu Desktop Time and Date Settings

Use the *Time & Date* panel to set the system time. If your target has an Internet connection, instead of setting the time manually you may select the radio button to set the time *Automatically from the internet*.

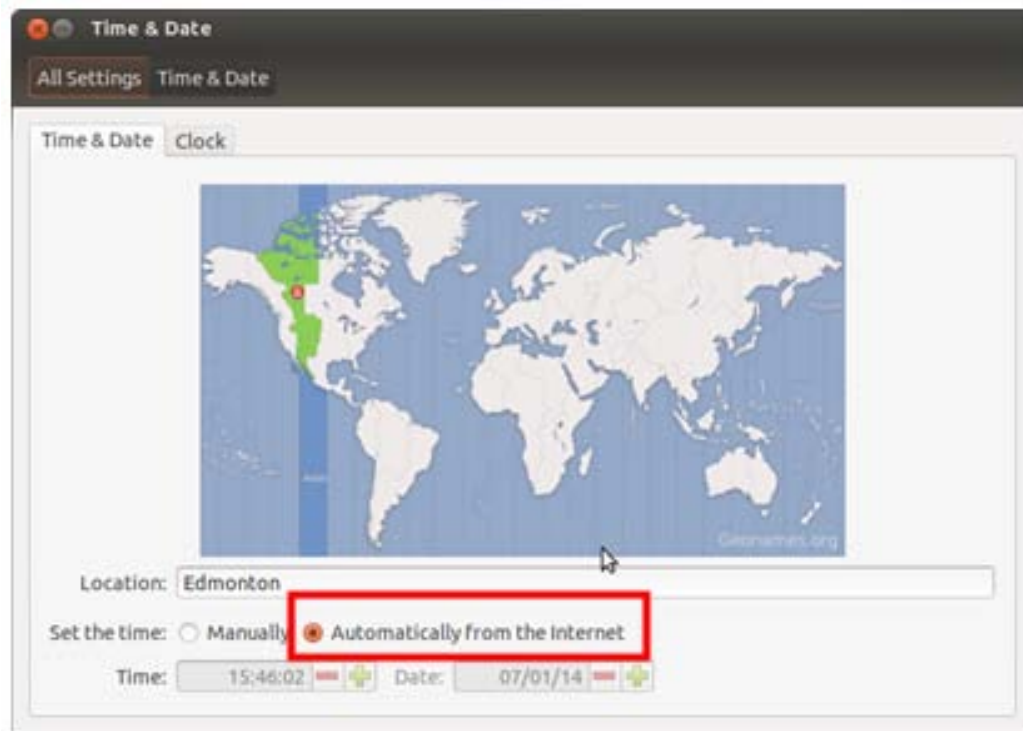



Figure 82 – Set Ubuntu Current Time

Once the time is current, close the *Time & Date* panel by clicking on the  icon at the upper left.

Enable xf86-video-modesetting Xorg driver

In the default installation, the Ubuntu desktop may not be able to take advantage of the full resolution of your monitor. The xf86-video-modesetting driver has been written to take advantage of the new Kernel Mode Setting (KMS) API of the DRM layer. This allows a user to switch between different screen resolutions at runtime (using the Xservers xrandr interface) and adds plug-and-play support for monitors.

Unfortunately the current Linaro Ubuntu distribution does not contain a package for xf86-video-modesetting driver. So it is necessary to manually download and build it.

Open up a terminal on the target system (use the Dashboard and search for “terminal” to locate a suitable program) and run the following commands:

Download and install xf86-video-modesetting

```
> sudo apt-get install xserver-xorg-dev libdrm-dev xutils-dev
> wget http://xorg.freedesktop.org/archive/individual/driver/xf86-video-modesetting-0.5.0.tar.bz2
> tar -xjf xf86-video-modesetting-0.5.0.tar.bz2
> cd xf86-video-modesetting-0.5.0
> ./configure --prefix=/usr
> make
> sudo make install
```

To enable the modesetting driver, you will need root access to use the vi editor to create **/etc/X11/xorg.conf** and add following lines. The easiest way to accomplish this is to use the Tera Term console, which is already logged in as root.

Section "Device"

Identifier "ADV7511 HDMI"

Driver "modesetting"

EndSection

If you would like to use the Ubuntu console, you will find the root user is disabled by default in Ubuntu. To enable the root user, enter:

sudo passwd

- Enter new root password
- Confirm new root password

To become the root user, enter **su** and the root password at the prompt. Type **exit** at the prompt to return to the linaro user.

Once this file is created¹⁰, the driver changes will take effect on the **next boot**. You will then be able to click on the System Operations icon in the upper right of the display:



Figure 83 – Click on the System Operations Icon

¹⁰ To exit vi, hit the <ESC> key, type a colon <:> and enter **wq** on the command line.

Select **Displays** from the drop-down menu to access resolution settings for the monitor. In the example below, an HP LP2065 20" monitor was used:

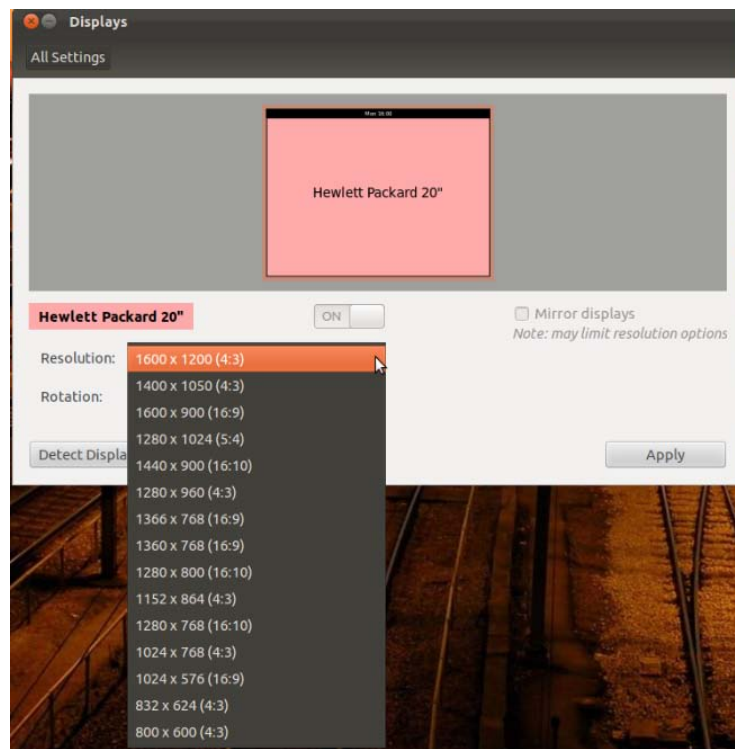


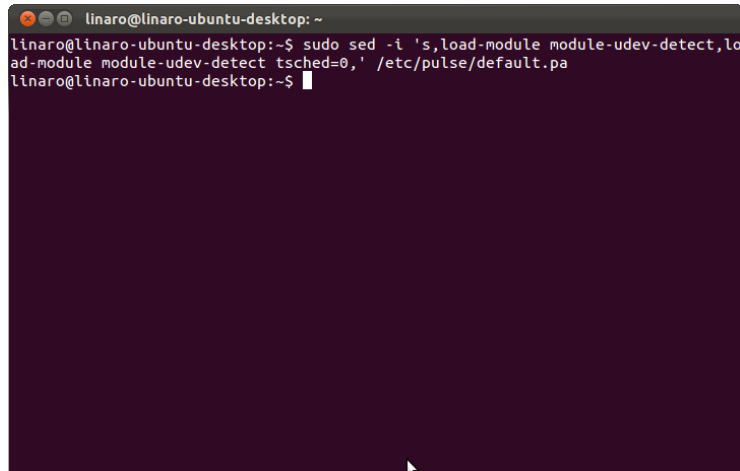
Figure 84 – Display Settings after Driver Update

The root file system exists on the SD card, so changes made will persist across subsequent boots.

Fixing issues with Pulse Audio

PulseAudio is the audio daemon used by default on the Linaro Ubuntu installation. Unfortunately PulseAudio's 'glitch-free' algorithm seems to cause audio glitches on this particular platform. To get seamless audio experience it is necessary to disable the glitch-free feature. To disable the 'glitch-free' feature of pulse audio open up a terminal on the target system and run the following command (all on one line):

```
> sudo sed -i 's,load-module module-udev-detect,load-module module-udev-detect  
tsched=0,' /etc/pulse/default.pa
```


A terminal window titled 'linaro@linaro-ubuntu-desktop: ~' with a dark background. The prompt is 'linaro@linaro-ubuntu-desktop:~\$'. The command entered is 'sudo sed -i 's,load-module module-udev-detect,load-module module-udev-detect tsched=0,' /etc/pulse/default.pa'. The cursor is at the end of the command line.

```
linaro@linaro-ubuntu-desktop:~$ sudo sed -i 's,load-module module-udev-detect,load-module module-udev-detect tsched=0,' /etc/pulse/default.pa
linaro@linaro-ubuntu-desktop:~$
```

Figure 85 – Update Pulse Audio in Terminal Window

System Shutdown

This is now a desktop computer environment, so you should follow a standard shutdown procedure before removing power from the Avnet target. To shut down the graphics system, select **Shutdown** from the drop down menu off the System (gear) icon at the upper right in the Ubuntu desktop.

To stop the command line console and shut down Linux, enter:

shutdown now -h¹¹

at the command prompt in the console window (Tera Term). You can now switch off power to the target board.

¹¹ From a standard account in the Ubuntu console enter: **sudo shutdown now -h**

Lab 9 – Ubuntu Demo

Lab Overview

The Ubuntu desktop is one of the most popular user interfaces for Linux PCs. In the lab you will become familiar with the basics of the Ubuntu GUI.

When you have completed Lab 9, you will know how to do the following:

- Navigate the Ubuntu desktop
- Locate the fundamental features of Ubuntu

Experiment 1: The Basic Desktop

This experiment introduces the Ubuntu desktop, which uses similar concepts and constructs common to most mouse-based GUIs, including the traditional Windows desktop.

Experiment 1 General Instruction:

Explore the Ubuntu desktop. If your display device has HDMI audio capabilities, use it to demonstrate HDMI audio from Ubuntu applications.

Experiment 1 Step-by-Step Instruction:

1. If you have not already done so, boot your Ubuntu desktop on the Avnet target using the SD/microSD card files created in the previous labs. You should have the following hardware connected to your system to make full use of the desktop:
 - a. HDMI Monitor (or DVI Monitor with HDMI-to-DVI adapter)
 - i. (Optional) HDMI Monitor with Speakers
 - b. USB powered hub (ZedBoard only)
 - c. USB Mouse
 - d. USB Keyboard
 - e. (Optional) Ethernet cable to connect to a DHCP router (or similar) for Internet access via Firefox

Although audio is available over HDMI by default, there are some additional configuration steps required to employ audio over the Analog Devices ADAU1761 codec (used with the audio jacks). Audio coverage with the codec deferred to Experiment 2 in this lab.

Once booted, your basic unmodified desktop should appear as shown in the figure below. You are automatically logged in as the default user, with password “*linaro*”. You will need to enter this password if you leave your desktop unattended, as by default it will automatically lock after a few minutes.

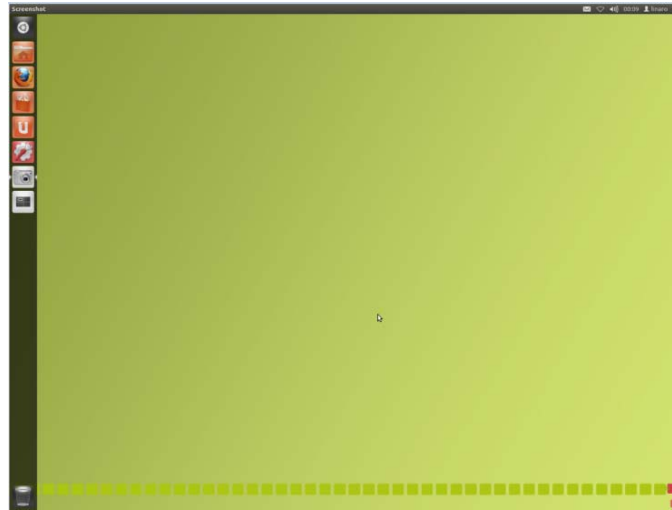


Figure 86 – Ubuntu Default Desktop

There are two control areas visible on the desktop, one at the upper right corner and a series of icons down the left hand side.

2. The control icons at the upper left are quick links to basic OS functions that should be familiar to everyone.

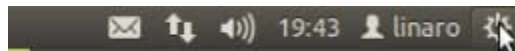

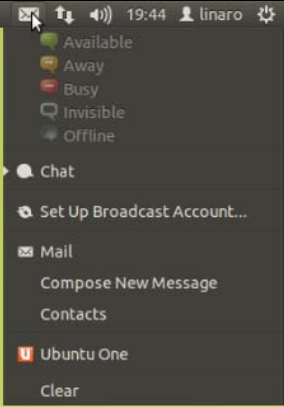

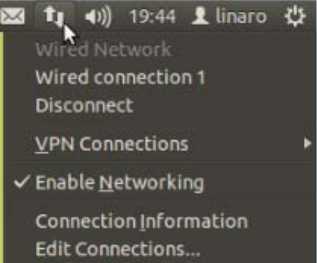

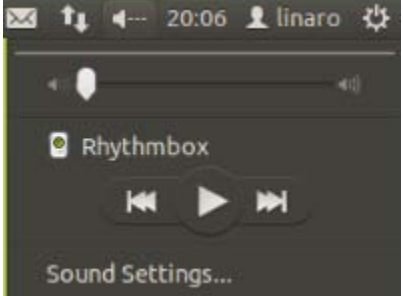

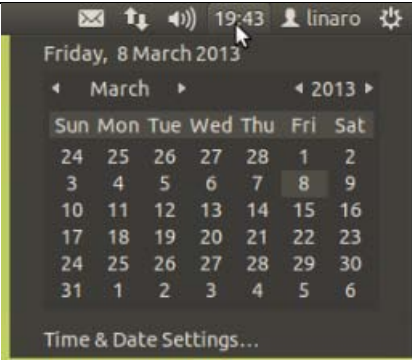
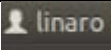
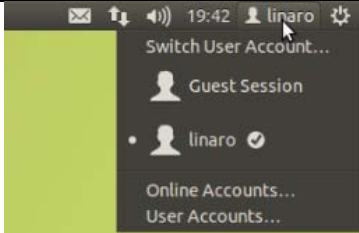

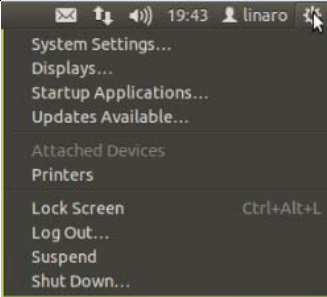


Figure 87 – Ubuntu Control Icons








Viewed from left to right, clicking on each icon in turn will produce a drop-down menu to access functions for:


Email and Web services		
Networking		
Sound Controls		
System clock and calendar functions		

User Account information		
System operations, including Log Off and Shut Down		

3. The primary area to access desktop applications is the **launchpad** at the left of the display. There is a set of default icons, plus each application you launch will appear in this area while it is running. You may right click on an application icon to either lock it or remove from the launchpad.

The applications represented by each icon are shown below:

	Dashboard – quick access to any applications not present in the launchpad
	File Browser – Equivalent to Windows Explorer
	Firefox – the default web browser
	Ubuntu Software Center – manage application installation for your Ubuntu system.
	Ubuntu One – access to the online Ubuntu community
	System Settings – Equivalent to Windows Control Panel
	Snapshot – Application used to take the screenshots for this manual. This is not in the launchpad by default, but was locked in place as described in the Dashboard section

	below.
	Workspaces – divides the screen into 4 separate work areas that can be individually populated.

4. Dashboard

The Ubuntu Dashboard is your central location for locating and running applications. You can select one of the applications groups on the top level to display installed software by group, but it is generally easier to locate applications by simply typing a name into the search box at the top.

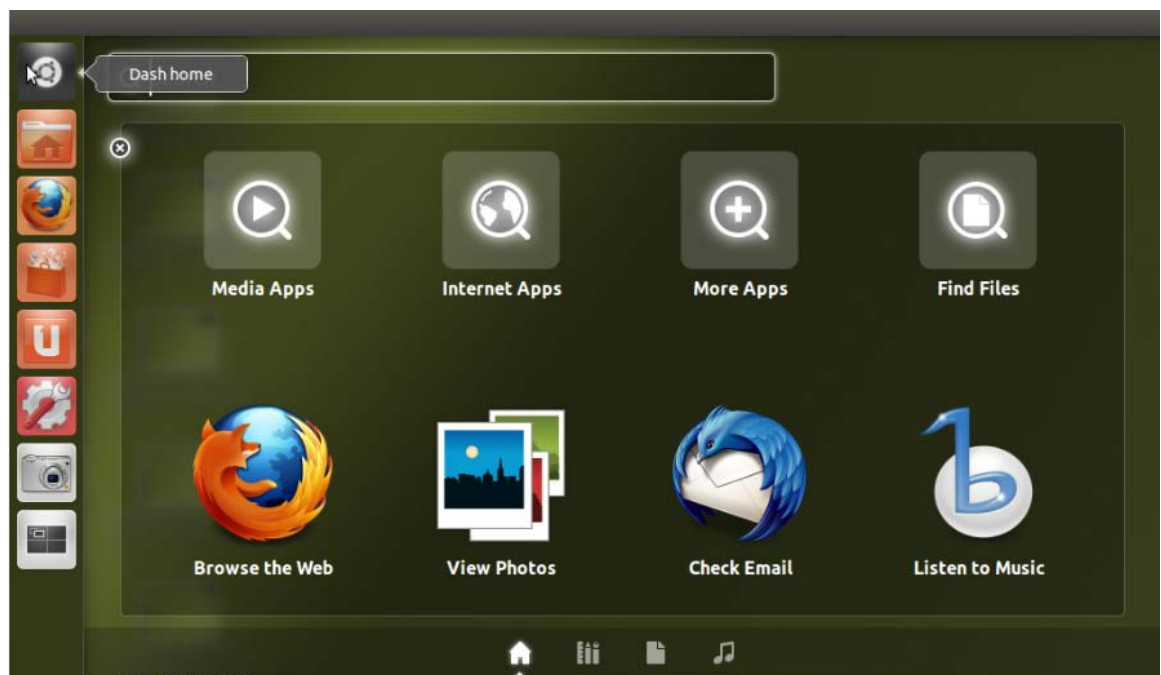


Figure 88 – Ubuntu Dashboard

For example, if you wish to locate a screen capture program, you might begin by typing “screen” into the search area. As you type, matching application names are displayed, so that after a few characters, we find what we are looking for.



Figure 89 – Locate Applications with Dashboard

5. Ubuntu Software Center

From this panel you can view and manage the software application portfolio on your Ubuntu system. You can view installed software, see the installation history, and locate new applications on the Internet which may be downloaded.

Note that not all applications will install on the Ubuntu system running on the ZedBoard, due to the restrictions imposed on the OS by the embedded system.

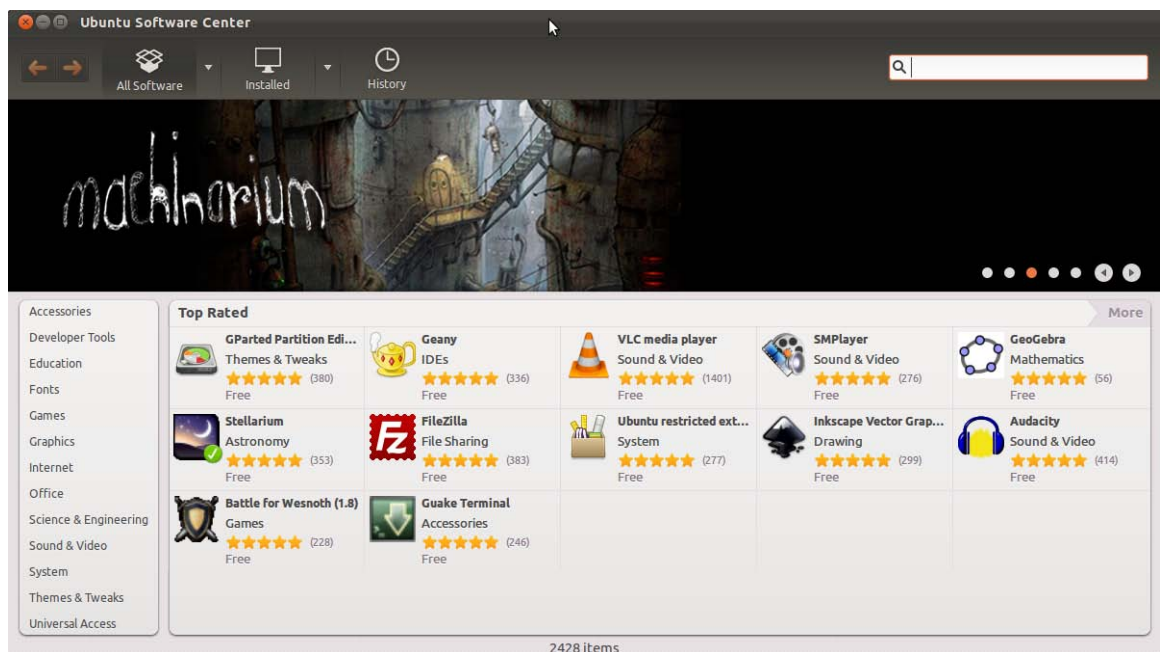


Figure 90 – Ubuntu Software Center

6. Ubuntu One

Ubuntu is the top Linux choice for desktop environments, and as such there is an active online community. If you are interested in exploring Ubuntu further, you can join and participate online by clicking the **Learn More** or **Join Now** buttons. You will need Internet access to participate.



Figure 91 – Ubuntu One

7. Customization

Of course, standard personalization familiar to desktop users is available in Ubuntu. For a simple example, right-click on the desktop to produce the pop-up menu below:

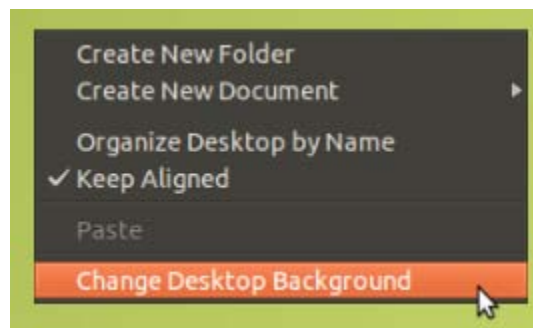


Figure 92 – Ubuntu Desktop Pop-up Menu

Select **Change Desktop Background** and you can pick from a variety of themes, as well as adding your own. An example of a customized desktop background for Ubuntu is shown below.

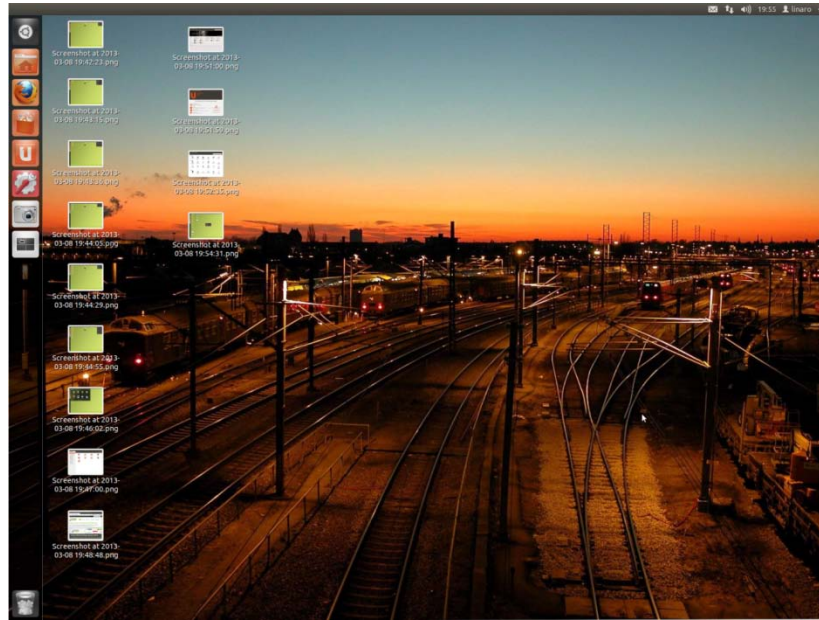


Figure 93 – Ubuntu Sample Custom Background

8. Games

In the default installation, there are a number of well-known games, plus you can use the Ubuntu Software Center to locate and download many more from the Internet. You can use the Dashboard filter on the right to see only the application types you are looking for – in this example, Games.

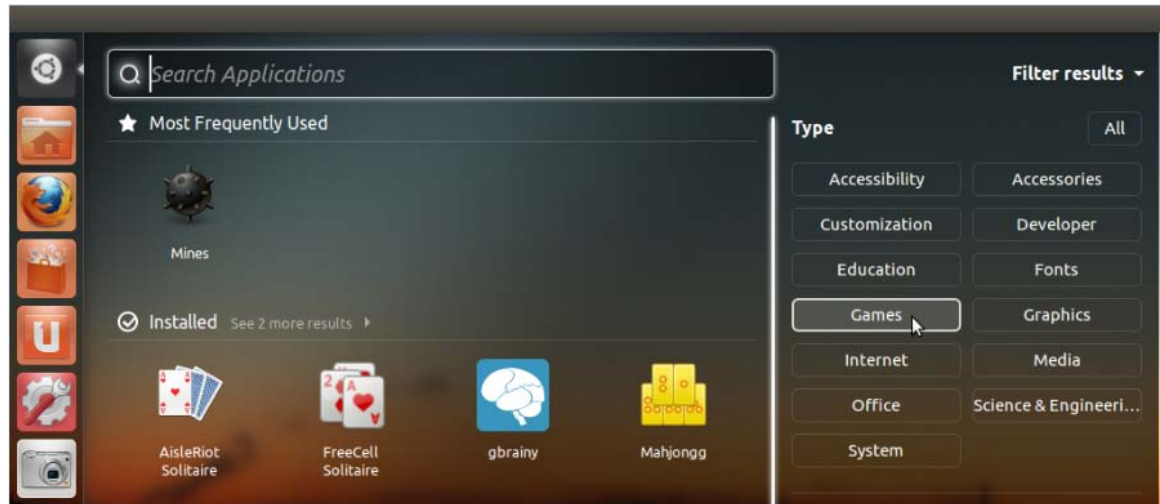


Figure 94 – Ubuntu Games

9. HDMI Audio

HDMI audio is provided by the Analog Devices ADV7511 transmitter using time division multiplexing to “mix” the video and audio signals on the HDMI cable. If you are using an HDMI device capable of processing audio signals received over the HDMI cable, then you will be able to hear audio on that device in the default configuration. You can test this by using the built-in Sound panel available in the Ubuntu desktop.

1. At the top right of the Ubuntu desktop, locate the **Sound** icon:

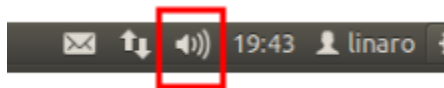


Figure 95 – Ubuntu Sound Icon

- Click the **Sound icon** to expand the selections, and click on **Sound Settings**.



Figure 96 – Ubuntu Sound Settings

- In the Sound panel, select the Output tab and ensure the default sound device selected is the **HDMI monitor**. As you can see in the display, you may also select the Headphones to play sound through that ADAU1761 audio codec. However, until we configure the codec parameters in the following experiment, the results may be less than optimal, so we will defer this until Experiment 2 is completed.¹²

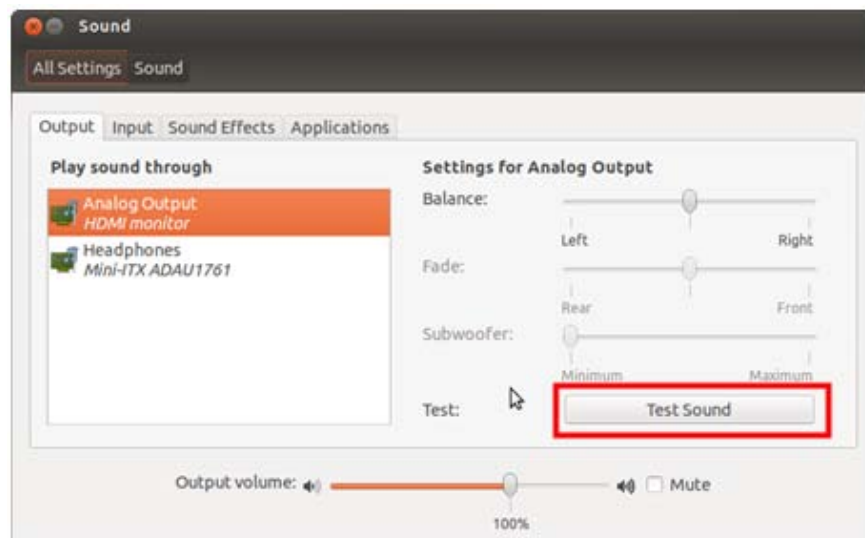


Figure 97 – Ubuntu Sound Panel, Output Tab

¹² In some cases you may find there are no devices listed in the Sound window, even though the monitor and ADAU1761 were detected by ALSA in the boot log. This appears to be an intermittent problem with PulseAudio. It may be possible to correct the issue by uninstalling and reinstalling PulseAudio, but this is not guaranteed to resolve the issue. In any event, the command line audio tests (later) will still function correctly, and this problem should be corrected in a new PulseAudio release in the future.

4. To hear audio from the HDMI device, click on the **Test Sound** button (see previous Figure) to open the Speaker Testing panel.



Figure 98 – Ubuntu Speaker Testing Panel

Alternately click the Test buttons for “Front Left” and “Front Right”. You should hear sound¹³ coming from the associated HDMI speaker.

5. While not required for this audio test, it is instructive to view the Input tab while we have the Sound panel open. Click on the Input tab, and note that by default the audio input device selected is the Analog Devices ADAU1761.

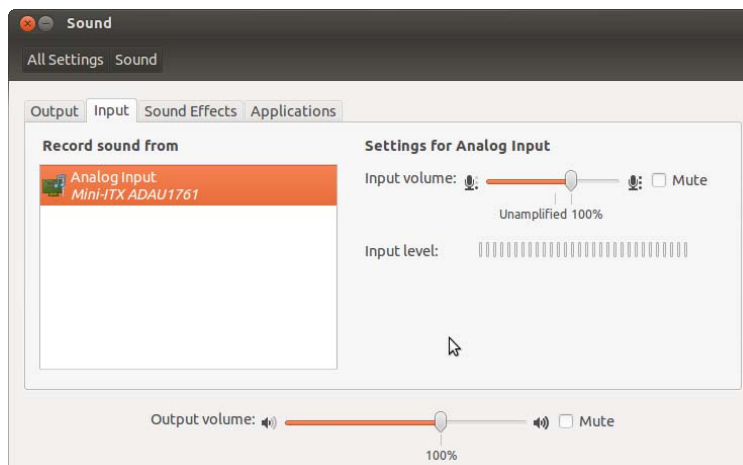




Figure 99 – Ubuntu Sound Panel, Input Tab

¹³ The actual sound may vary depending on the specific release of the Root File System, but at a minimum you should hear the default system sound – a water drop, or sonar ping, or something similar – clearly audible from the selected HDMI speaker.

You may close the Sound panel at this point by clicking on the **Close** icon at the upper left. 

6. With the HDMI audio output verified, we can now try a more prolonged test by playing a Waveform Audio (.wav) file using one of the Ubuntu Desktop applications. There are many applications for playing audio files on Ubuntu, some installed by default and many more available from the Internet. For simplicity, in this instance we will simply use whatever default audio application is currently in place to process the .wav format.

To prevent any possibility of copyright violation, you will need to supply your own .wav file to play. Again, these can be downloaded from the Internet, created on any Windows platform with Windows Media Player installed, or generated on a host Linux system.

Place your .wav file on the Ubuntu Desktop using the File Browser icon from the Launchpad on the left of the display. 

7. Double-click on your .wav file. The default application will launch, and the audio will start playing over the HDMI speakers. In this instance, the default player was the Movie application.

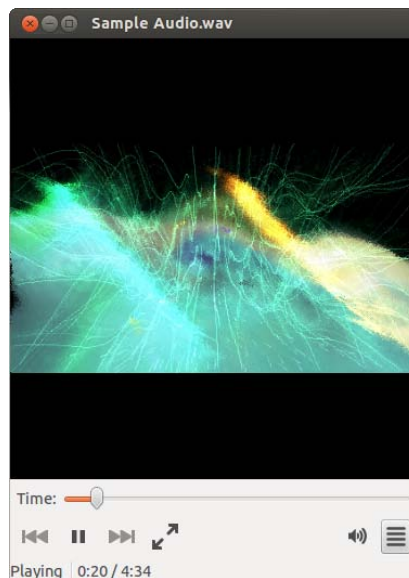


Figure 100 – Playing Audio with Ubuntu Movie Player

This completes the HDMI audio section. You may close the default audio player.

Experiment 2: Audio using the ADAU1761 Codec

The Analog Devices ADAU1761 audio codec is capable of processing stereo input and output using the four audio jacks located on the perimeter of the Avnet targets. If you wish to use headphones, external speakers, or process incoming audio on the Ubuntu system, you will need to use this codec, and the associated software drivers.

There are four jacks that are used with standard 1/8" audio plugs on the Avnet targets:

1. **Headphone out:** used for output to standard unpowered headphones, earbuds or headsets.
2. **Microphone in:** used with a standard unpowered microphone or headset to capture audio input.
3. **Line out:** used with powered external speakers, or as input to an external amplifier.
4. **Line in:** used to receive audio input from another device, such as the headphone output from a PC, or line out from standard audio equipment.

Audio using the jacks is routed through the Analog Devices ADAU1761 audio codec. The Linux drivers supplied in the Linaro kernel make use of the Advanced Linux Sound Architecture (ALSA) to produce high quality stereo output and receive stereo input for recording or playback. The ALSA architecture includes a software mixer which must be configured using an initialization (state) file that creates a path in the codec from the audio jacks to the processing system, and controls various settings such as channels, volume, voltage and signal mixing. Once the configuration is complete, the initialization file can be saved to a default location so the system will be automatically configured at boot time.

The test procedure to validate the audio jacks requires the use of headphones, a microphone (or a headset with audio output and input jacks) and a male-to-male 1/8" audio cable. This cable is used to connect a PC headphone jack to the input jacks on the Avnet target.

Experiment 2 General Instruction:

Use built-in ALSA command line applications for playback and recording to demonstrate the use of the ADAU1761 audio codec, via the audio jacks on the Avnet target.

Experiment 2 Step-by-Step Instruction:

1. Before we can use the audio codec, it is necessary to initialize the device via the device drivers, to establish a circuit from the audio jacks to the SoC, and to initialize the audio parameters for playback and recording. There are many separate parameters associated with this task, so the easiest way to do this is to use an initial configuration file. The file is included in **Supplied Files** as:

adau1761_golden.state

Copy the initialization file to the Ubuntu Desktop using any convenient method¹⁴.

2. The configuration file is a simple hierarchical text file, with each control identified and initial parameters provided in separate blocks. Consider the example snippets below:

```
state.monitor {
    control {
    }
}
```

The first block identifies the HDMI monitor as a sound device, but there are no ALSA configuration parameters associated with it. As we have seen, an audio-capable HDMI monitor will produce audio by default under Ubuntu.

```
state.ADAU1761 {
    control.1 {
        iface MIXER
        name 'Digital Capture Volume'
        value.0 255
        value.1 255
        comment {
            access 'read write'
            type INTEGER
            count 2
            range '0 - 255'
```

¹⁴ For example, you can access the file on your local network directly from the Ubuntu desktop over the Ethernet connection, or you can copy the file to a USB thumb drive and plug it into one of the open USB ports on your embedded system.

```

        dbmin -9563
        dbmax 0
        dbvalue.0 0
        dbvalue.1 0
    }
}

```

The next block introduces the ADAU1761 as the second audio device. There are 38 separate control parameters associated with the device, from Digital Playback Volume (shown below) to Lineout Playback Switch for selection of specific audio jacks, through mixer controls that can be used to mix audio tracks from one input to another. There are typically two values for each control, corresponding to the left and right stereo settings, and the initial value is provided next to the identifiers. In the comment block, the ranges and parameters for various audio configuration elements are provided, and these are specific to the ADAU1761 and can be procured from the Hardware manual for the device.

```

control.2 {
    iface MIXER
    name 'Digital Playback Volume'
    value.0 255
    value.1 255
    comment {
        access 'read write'
        type INTEGER
        count 2
        range '0 - 255'
        dbmin -9563
        dbmax 0
        dbvalue.0 0
        dbvalue.1 0
    }
}

```

To illustrate these points, the control block labeled 2 is identified as an interface to the ALSA mixer device for Digital Playback Volume.

```

iface MIXER
name 'Digital Playback Volume'

```

In the comment block, we can see the values can be seen and changed by the driver:

```

access 'read write'

```

The number of channels for the control is 2, indicating stereo:

```

count 2

```


The valid values for the volume are 0 (mute) through 255:

```
range '0 - 255'
```

Referring back to the initialization section above the comment block, we can see that by default the Digital Playback Volume is set to the maximum allowed for each channel:

```
value.0 255      (Left channel)
value.1 255      (Right channel)
```

Further down in the file initial volume controls can be set independently for headphones, microphone and line in, as well as dynamic volume control during operation through the Ubuntu Desktop. You may wish to explore this file using any text editor to experiment with the effect different values have on the input and output audio.

3. To initialize the audio codec for the first time, we can use a terminal window to send command line instructions to the software. Using the launchpad at the left of the Ubuntu Desktop, open a terminal window.
4. Change directories to the desktop, and issue the command:

```
alsactl restore -f adau1761_golden.state
```

This instructs the ALSA architecture to perform the device initialization specified in the named file. The values supplied in the file can be changed using a text editor, and you may issue the command above each time you change the file in order to adopt your updated audio configuration.

Next, we will validate the operation from each of the audio jacks, and once everything is tested the final step will be to save the initial configuration permanently, so it will be the new boot default.

5. In this step, you will connect the headphone output of your host PC to the audio input jacks on the board. The audio will be captured and re-routed out the playback jacks to verify that all four connections are operational.

- a. Connect the 1/8" male-to-male audio plug to the headphone output of your host PC.



Figure 101 – Male Audio Jack in Host PC Headphone Output

- b. Connect the other end of the male-to-male audio plug to the microphone jack of the Zynq Mini-ITX.

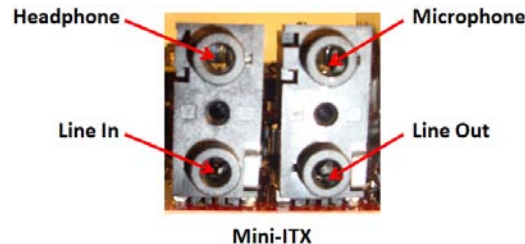


Figure 102 – Male Audio Jack in Microphone Input

- c. Plug your headphones into the headphone jack of the Zynq Mini-ITX.
- d. On the host PC, select any audio file for playback through the headphone output. An .mp3 or .wav music file is a good choice for this test, as it provides persistent audio at various ranges throughout. Double-click on the audio file on the host PC to initiate playback using Windows Media Player.
- e. In the terminal window of the Ubuntu Desktop, we will use the ALSA command line audio recorder and audio player together to send the incoming audio to the ADAU1761, and route it back out through your headphones. This will validate the microphone and headphone jacks.

In the Terminal window, enter:

```
arecord -Dplughw:ADAU1761 -f S32 -r 48000 | aplay -Dplughw:ADAU1761
```

This command selects the ADAU1761 as the input device for recording, and the output device for playback. You should hear the audio

originating on your host PC through the headphones connected to the Avnet target.

- f. While the audio is playing, unplug the microphone jack from the Avnet target, and insert it into the **Line In** receptacle. Once again, you should hear the audio from your host PC through the headphones.

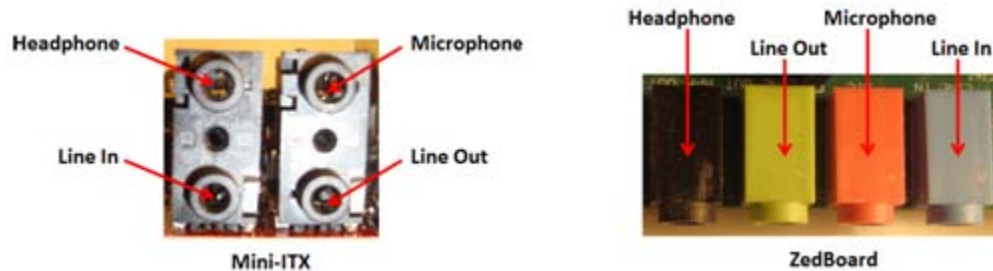


Figure 103 – Male Audio Jack to Line In

- g. While the audio continues to play, unplug the headphone jack and insert it into the Line Out receptacle. You will hear the audio from your host PC, but this time at a lower volume.

If all of these tests functioned as described, you have validated the hardware and software connections on your Avnet target for the audio codec.

6. If you used a .wav file to test the HDMI audio in Experiment 1, you can also play that same file through the audio codec. You can do this from the command line using the ALSA player with the following command. You may have the headphones plugged into the **Line Out** or the **Headphone** receptacle, with superior volume in the latter.

In the terminal window, enter:

```
▪ aplay -Dplughw:ADAU1761 -r 48000 -f S32 <audio_file>.wav
```

7. As described earlier, you may use a text editor to experiment with the audio configurations in the adau1761_golden.state file. Once you have a set of parameters you would like to save as the default, you may do this from the terminal window, using the following command:

```
sudo alsactl store
```

This command saves the current audio parameters to the default configuration file at:

`/var/lib/alsa/asound.state`

This file will be used by the Linux kernel at boot time to configure the ADAU1761 audio codec.

This completes the Ubuntu tutorial.

Appendix I – Build Environment Setup under CentOS for Ubuntu

The Sourcery CodeBench cross toolchain for Zynq targets is installed with the Xilinx SDK. However, downloading from open source repositories and creating Linux partitions¹⁵ for Ubuntu development require additional steps to:

- Allow sudo (root privilege) access from a standard account
- Add Git SCM tools
- Install support libraries for the Gnome Partition Editor (GParted)

General Instruction:

Install the Git SCM tools and support libraries for GParted.

Step-by-Step Instructions:

1. If a terminal window is not already open on the desktop of the CentOS guest operating system, open one from the main menu by selecting:

Applications → Utilities → Terminal

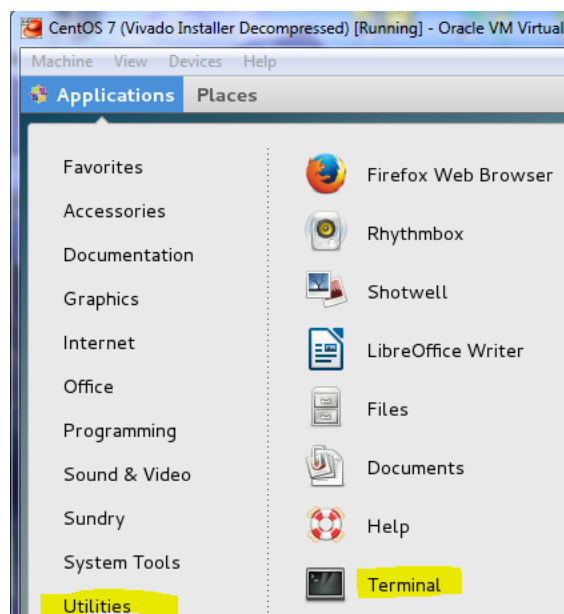


Figure 104 – Launching a CentOS Terminal from the Main Menu

¹⁵ Detailed instructions for open source download and creating ext4 partitions are part of this reference design document.

2. Take on root privileges by running the superuser elevation command **su** and entering your login password. The prompt changes from \$ to # to indicate the privilege elevation succeeded.

```
$ su
```

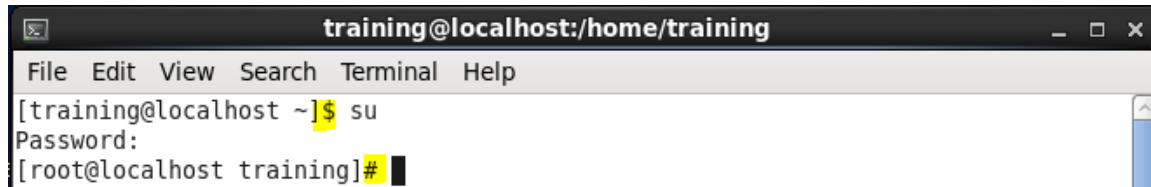


Figure 105 – Elevating to Superuser Privileges

3. Use visudo text editor to edit the */etc/sudoers* file.

```
# visudo
```

4. Add the **training** user to the sudoers list by inserting the following line to the users section of the sudoers file as shown in the Figure below. The users section is located towards the end of the file.

```
training    ALL=(ALL) ALL
```

Press the “i” key on the keyboard to enter *vi insert* mode.

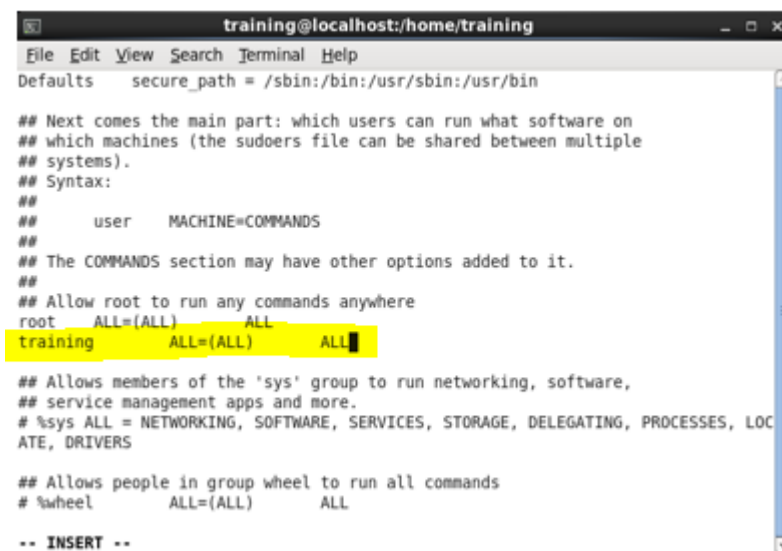


Figure 106 – Adding training Account to */etc/sudoers* File

- Exit the vi editor and save changes to the sudoers file by using the write-quit key sequence:

```
<ESC> :wq
```

- The **training** user will now have sufficient privileges to do important systems tasks using the sudo command. Exit superuser mode.

```
# exit
```

- Next, the system should be revised with the latest updates. Use the yum package manager to install the system updates from the appropriate mirrors and when prompted accept the download and installation of all recommended packages. These updates can take several minutes and may present several user prompts before completion.

If prompted to allow download of packages, accept the download by pressing the **Y** key followed by the **Enter** key.

If prompted for the import of the GPG key accept the import by pressing the **Y** key followed by the **Enter** key.

```
$ sudo yum update
```

```
training@localhost:~
File Edit View Search Terminal Help
(42/55): pulseaudio-libs-glib2-0.9.21-14.el6_3.x86_64.rpm | 23 kB | 00:00
(43/55): pulseaudio-module-bluetooth-0.9.21-14.el6_3.x86_64.rpm | 69 kB | 00:00
(44/55): pulseaudio-module-gconf-0.9.21-14.el6_3.x86_64.rpm | 24 kB | 00:00
(45/55): pulseaudio-module-x11-0.9.21-14.el6_3.x86_64.rpm | 30 kB | 00:00
(46/55): pulseaudio-utils-0.9.21-14.el6_3.x86_64.rpm | 70 kB | 00:00
(47/55): python-2.6.6-29.el6_3.x86_64.rpm | 4.8 MB | 00:04
(48/55): python-libs-2.6.6-29.el6_3.x86_64.rpm | 623 kB | 00:00
(49/55): redhat-logos-60.0.14-12.el6.centos.noarch.rpm | 15 MB | 00:12
(50/55): selinux-policy-3.7.19-155.el6_3.4.noarch.rpm | 1.3 MB | 00:01
(51/55): selinux-policy-targeted-3.7.19-155.el6_3.4.noarch.rpm | 2.6 MB | 00:02
(52/55): sudo-1.7.4p5-13.el6_3.x86_64.rpm | 423 kB | 00:00
(53/55): tzdata-2012c-3.el6.noarch.rpm | 441 kB | 00:00
(54/55): udev-147-2.42.el6.x86_64.rpm | 361 kB | 00:00
(55/55): xulrunner-10.0.7-1.el6.centos.x86_64.rpm | 12 MB | 00:09
-----
Total | 1.0 MB/s | 139 MB | 02:15
warning: rpmts_HdrFromFdno: Header V3 RSA/SHA1 Signature, key ID c105b9de: NOKEY
Retrieving key from file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-6
Importing GPG key 0xC105B9DE:
  Userid : CentOS-6 Key (CentOS 6 Official Signing Key) <centos-6-key@centos.org>
  Package: centos-release-6-3.el6.centos.9.x86_64 (@anaconda-CentOS-201207061011.x86_64/6.3)
  From : /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-6
Is this ok [y/N]:
```

Figure 107 – Yum Prompt to Import GPG Key

- The ncurses-devel package and Git SCM tool will be installed next using the package manager and when prompted accept the download and installation of

all recommended packages. When prompted, accept the defaults to install all packages.

```
$ sudo yum install ncurses-devel git
```

9. If you plan on using this as your own development machine to submit patches, configure Git with your contact info and customize your preferences. By setting your name and email address, they will be embedded into any of the commits that are generated from this system.

Note: Setting the identity only needs to be done once and Git will always use this information unless overridden with a different name or e-mail address for specific projects. To perform the override, repeat these same commands but remember to omit the `--global` flags when in the target project.

```
$ git config --global user.name "<your name>"
```

```
$ git config --global user.email "<your email>"
```

10. Set your editor preference (default is vi or vim) if desired. On this reference system, the vi editor will be used.

```
$ git config --global core.editor vi
```

11. If there is a preference for a particular diff tool used to resolve merge conflicts, this should also be set here. On this reference system, the vimdiff tool will be used.

```
$ git config --global merge.tool vimdiff
```

12. In order to use Sourcery CodeBench on an x86 64-bit Linux host system, the 32-bit system libraries must be installed. The 32-bit libraries are available as a series of packages that can be installed using yum on the command line. When prompted, accept the defaults to install all packages.

```
$ sudo yum install glibc-devel.i686 gtk2-devel.i686 \  
libcanberra.i686 \  
libcanberra-gtk2.i686 PackageKit-gtk3-module.i686 \  
GConf2.i686 ncurses-libs.i686 xulrunner.i686
```


13. When the Xilinx SDK was installed, you were required to execute the following script (assuming the default installation path was chosen) to add toolchain executables to the \$PATH variable.

```
$ source /opt/Xilinx/SDK/2014.4/settings64.sh
```

If you would like this command to execute each time you open a new terminal window, you can place it in your hidden **.bashrc** file in the **/home/training** directory. However, it must be placed in a conditional statement, because if **.bashrc** is referenced during the login shell (as is often the case), it will prevent X11 applications such as File Manager from executing. You may use gedit to add the following lines to your **.bashrc** file.

echo 'Adding SDK paths...'

shopt -q login_shell && echo 'Login shell' || source /opt/Xilinx/SDK/2014.4/settings64.sh

```
$ cd ~  
  
$ gedit .bashrc
```

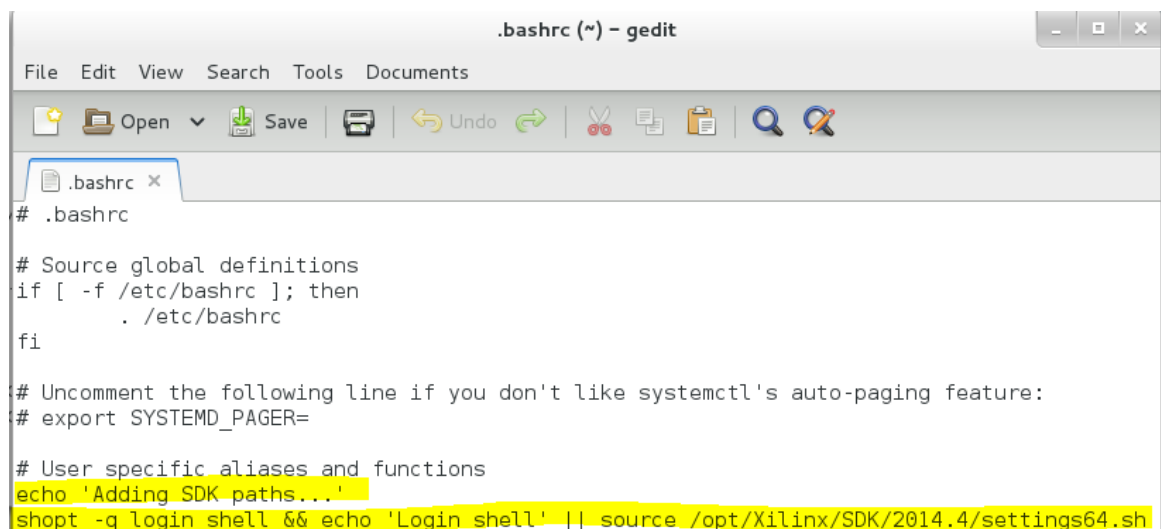


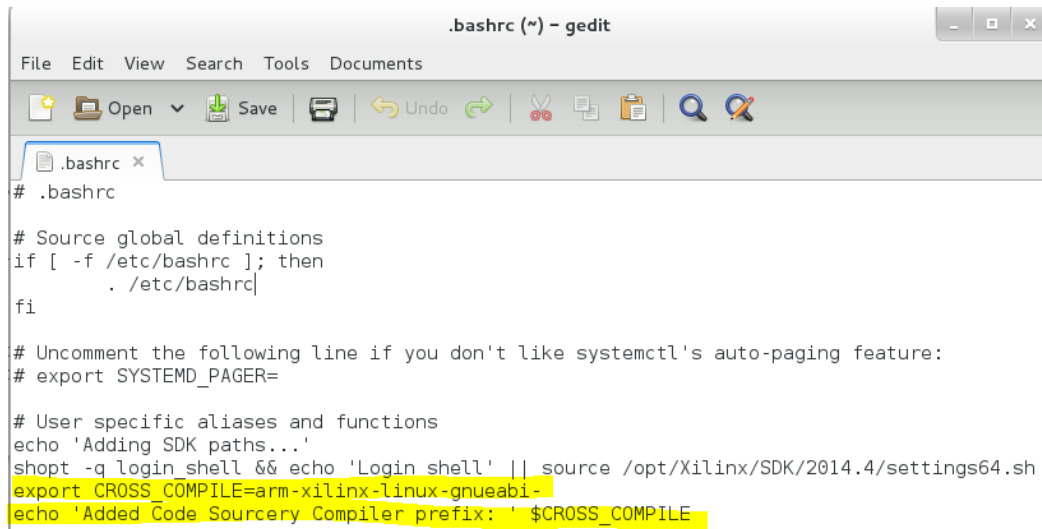
Figure 108 – Add SDK Paths to .bashrc

Do not close the **.bashrc** file yet.

14. Many programs refer to environment variable `CROSS_COMPILE` to invoke the GNU tools that are used to build the software package for an embedded target. You can modify the `.bashrc` file to set this variable each time a terminal window is opened.

Add the following lines to the `.bashrc` file.


```
export CROSS_COMPILE=arm-xilinx-linux-gnueabi-  
echo `Added Code Sourcery Compiler Prefix:`  
$CROSS_COMPILE
```



```
.bashrc (~) - gedit  
File Edit View Search Tools Documents  
Open Save Undo  
# .bashrc  
# Source global definitions  
if [ -f /etc/bashrc ]; then  
    . /etc/bashrc  
fi  
# Uncomment the following line if you don't like systemctl's auto-paging feature:  
# export SYSTEMD_PAGER=  
# User specific aliases and functions  
echo 'Adding SDK paths...'  
shopt -q login shell && echo 'Login shell' || source /opt/Xilinx/SDK/2014.4/settings64.sh  
export CROSS_COMPILE=arm-xilinx-linux-gnueabi-  
echo 'Added Code Sourcery Compiler prefix: ' $CROSS_COMPILE
```

Figure 109 – Add Code Sourcery Compiler String to `.bashrc`

Save the changes and exit **gedit** using the **File→Quit** menu option. Close the terminal window and open a new terminal window. The `.bashrc` file executes each time a new terminal window is opened so the environment is automatically established. The echo commands you included will remind you that the operations executed.

15. This document uses the Gnome Partition Utility to prepare the SD/microSD for booting. The utility is part of the *Extra Packages for Enterprise Linux* (epel) repository, which must be installed first. Enter the following at your terminal prompt:
- a. `sudo yum install epel-release -y`
 - b. `sudo yum install gparted -y`
16. This completes the setup of the virtual machine. If you wish to save your virtual machine work at any point in time, the session can be suspended and the virtual machine window closed. To do this, click on the Window Close  icon in the upper right hand corner of the virtual machine window. Choose **Save the machine state** and click the **OK** button. The next time this VM is opened, your session state will be restored precisely as you left it.

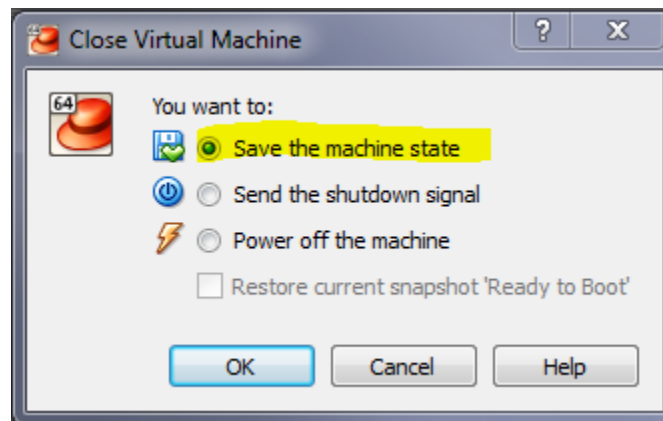


Figure 110 – Suspending the Virtual Machine

Resources

ZedBoard.org

ZedBoard documentation page

<http://www.zedboard.org/documentation/1521>

ZedBoard reference designs and tutorials

<http://www.zedboard.org/design/1521/11>

Zynq Mini-ITX documentation page

<http://www.zedboard.org/documentation/2056>

Zynq Mini-ITX reference designs and tutorials

<http://www.zedboard.org/design/2056/17>

ZedBoard Forum Support (English)

<http://www.zedboard.org/forums/zed-english-forum>

ADI

ADV7511 Product Page

<http://www.analog.com/en/audiovideo-products/analoghdmidvi-interfaces/adv7511/products/product.html>

ADAU1761 Product Page

<http://www.analog.com/en/audiovideo-products/audio-signal-processors/adau1761/products/product.html>

Wiki Instruction page

<http://wiki.analog.com/resources/fpga/docs/hdl>

<http://wiki.analog.com/resources/fpga/docs/hdl/vivado>

<http://wiki.analog.com/resources/fpga/xilinx/kc705/adv7511> (standalone software)

ADI HDL Reference Designs HDL User Guide

<http://wiki.analog.com/resources/fpga/docs/hdl/github>

ADI Support

<http://ez.analog.com/community/fpga>

ADV7511 Xilinx Evaluation Boards Reference Designs (ZedBoard)

<http://wiki.analog.com/resources/fpga/xilinx/kc705/adv7511>

Xilinx Website

Xilinx Design Tools Installation and Licensing Guide

http://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_4/ug973-vivado-release-notes-install-license.pdf

Zynq

<http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/index.htm>

Xilinx Silicon and Tools Forum Support

<http://forums.xilinx.com/>

Other Internet Resources

<http://www.hdmi.org/>

<http://en.wikipedia.org/wiki/HDMI>

<http://www.linaro.org/>

<http://www.ubuntu.com/>

Revision History

Date	Version	Revision
16 Apr 13	01	Initial Draft (EDK/SDK 14.4)
24 May 13	02	Updated for kernel ulmage & compatible U-boot
10 July 14	03	Added Zynq Mini-ITX + Updated to Vivado/SDK 2013.4
19 Feb 2014	04	Added git tag example + Updated to Vivado/SDK 2014.4 + Zynq Mini-ITX Specific