

Implementing Linux on the Zynq™-7000 SoC

Lab 2.2

Creating the Basic Root File System



September 2012
Version 05

Table of Contents

Table of Contents	2
Lab 2.2 Overview	3
Lab 2.2 Objectives	3
Experiment 1: Retrieve Source Code for Essential Applications	4
<i>Questions:</i>	9
Experiment 2: Building Applications for a Zynq Target	10
<i>Questions:</i>	24
Experiment 3: Create System Folders and Startup Scripts	25
<i>Questions:</i>	35
Exploring Further	36
Revision History	36
Resources	36
Answers	37
Experiment 1	37
Experiment 2	37
Experiment 3	37

Lab 2.2 Overview

The Xilinx Linux project combines the benefit of open source Linux operating system together with a customized solution geared towards developing software on its processing platform.

In this lab, the process to build a RAM disk image containing a very basic root file system is explored.

Lab 2.2 Objectives

When you have completed Lab 2.2, you will know how to do the following:

- Retrieve source code for BusyBox and Dropbear applications
- Configure the source trees for BusyBox and Dropbear applications
- Build the BusyBox and Dropbear applications
- Create system directories and startup configuration
- Build a RAM disk containing the Root File System

Experiment 1: Retrieve Source Code for Essential Applications

This experiment shows how to obtain the source code for applications that are essential to the root file system on an embedded Linux platform. To successfully complete this lab, you will need Internet access to retrieve the source code.

Experiment 1 General Instruction:

Obtain source code for packages needed to create a basic root file system for Zynq.

Important Note: If performing this lab on an Avnet SpeedWay training laptop, the appropriate source trees have already been obtained for you and Experiment 1 is for reference only.

If the the BusyBox source already exists in the `/home/training/busybox/` folder and the Dropbear source already exists in the `/home/training/dropbear-0.53.1/` folder, Experiment 1 can be skipped and Experiment 2 can be started.

Experiment 1 Step-by-Step Instructions:

1. If the CentOS virtual machine is not already open, launch the VMware Player application from by selecting **Start → All Programs → VMware → VMware Player**. If the CentOS virtual machine is already open, skip ahead to Step 4.



Figure 1 – The VMware Player Application Icon

2. Select the virtual machine named **CentOS-6.3-amd64-ZedBoard-linux** from the selections on the left and then click on the **Play virtual machine** button to the right.

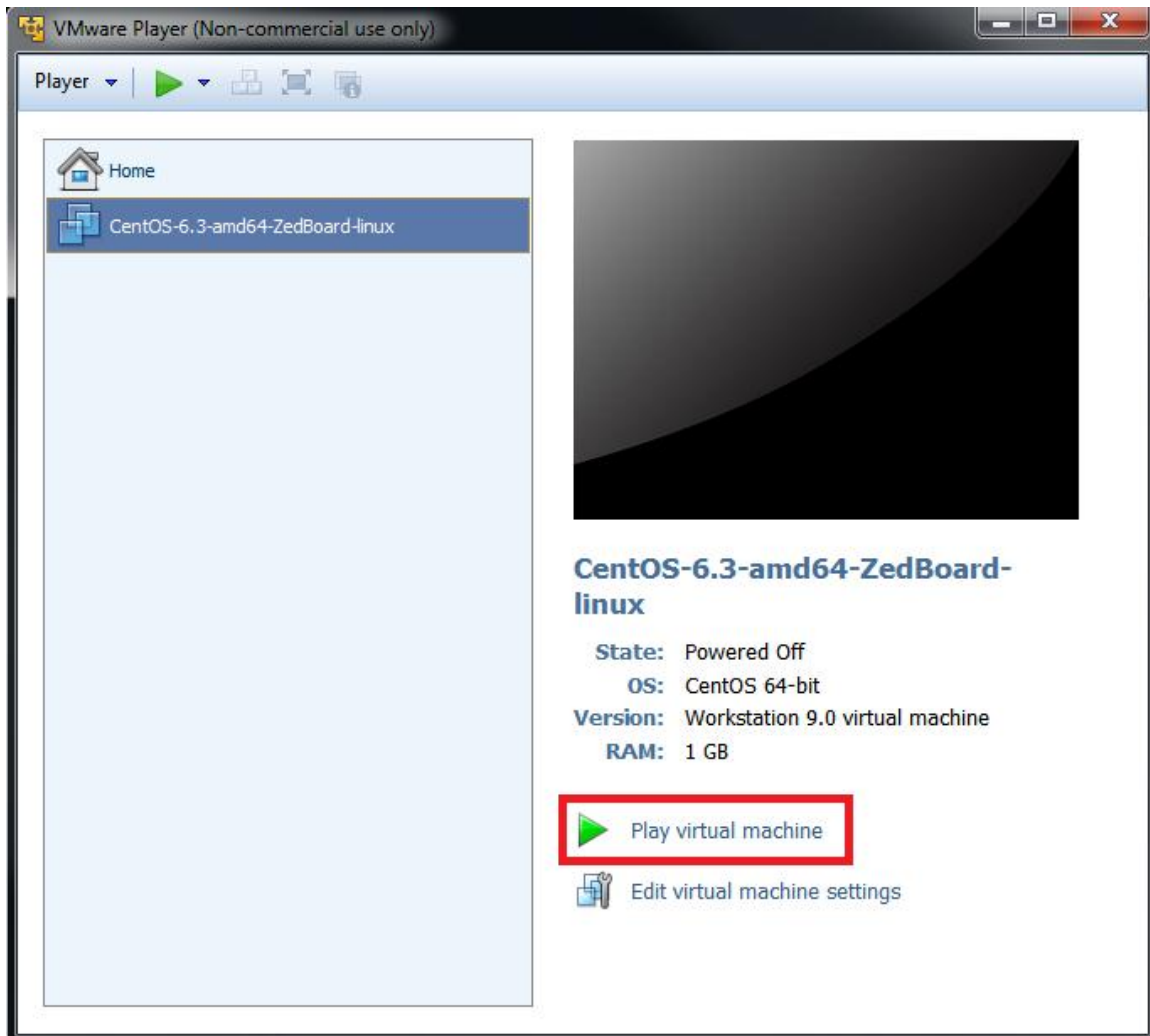


Figure 2 – The VMware Player Application

3. If prompted for a workstation login, click on the user entry **training** and enter the password **Avnet** in order to log into the system.

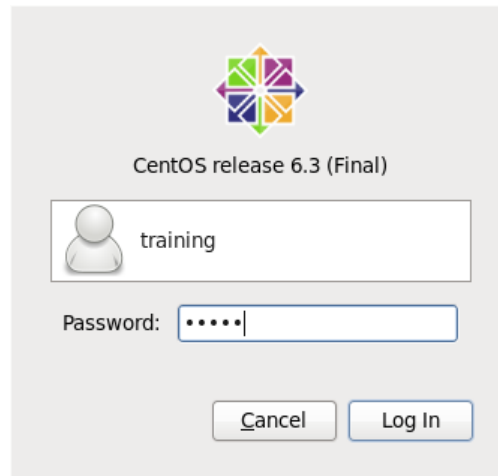


Figure 3 – The CentOS Workstation Login

4. If a terminal is not already open on the desktop of the CentOS guest operating system, open a terminal window through the **Applications→System Tools→Terminal** menu item.

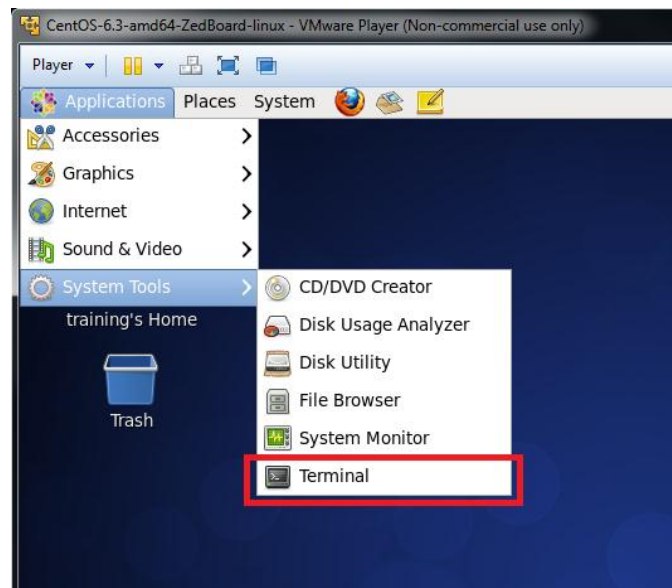


Figure 4 – Launching the CentOS Terminal from the Desktop

5. Change the current working directory to the training user home directory.

```
$ cd ~
```

6. The BusyBox used for Zynq is kept in a Git repository and is located at the following URL:

[git://git.busybox.net/busybox](https://git.busybox.net/busybox)

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.

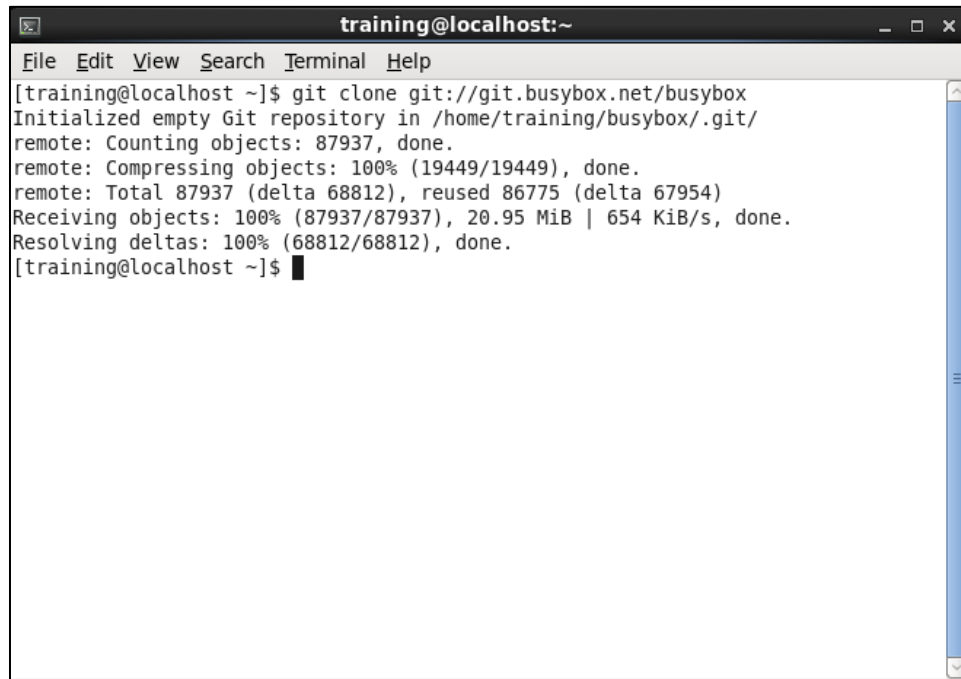
Use the following Git command to clone the repository. Perform this step only if you are not performing this exercise as part of a Live SpeedWay course.

```
$ git clone git://git.busybox.net/busybox
```

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

The clone command sets up a few convenience 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 ~]$ git clone git://git.busybox.net/busybox  
Initialized empty Git repository in /home/training/busybox/.git/  
remote: Counting objects: 87937, done.  
remote: Compressing objects: 100% (19449/19449), done.  
remote: Total 87937 (delta 68812), reused 86775 (delta 67954)  
Receiving objects: 100% (87937/87937), 20.95 MiB | 654 KiB/s, done.  
Resolving deltas: 100% (68812/68812), done.  
[training@localhost ~]$
```

Figure 5 – Using the Git Clone Command

8. The Dropbear used for Zynq can be obtained from the following URL:

<http://matt.ucc.asn.au/dropbear/releases/dropbear-0.53.1.tar.gz>

Perform this step only if you are not performing this exercise as part of a Live SpeedWay course. Use the following command to download the source code:

```
$ wget http://matt.ucc.asn.au/dropbear/releases/dropbear-0.53.1.tar.gz
```


Questions:

Answer the following questions:

- *How is a working copy of the BusyBox code base obtained?*

- *How is a release copy of the Dropbear code base obtained?*

Experiment 2: Building Applications for a Zynq Target

A working copy of the BusyBox repository and a release copy of the Dropbear source code are now available on the local development machine. These local copies can be used to configure the source trees so that they can be built for a Zynq target platform.

Experiment 2 General Instruction:

Configure and build the BusyBox and Dropbear source trees for the Zynq target platform.

Experiment 2 Step-by-Step Instructions:

1. If the CentOS virtual machine is not already open, launch the VMware Player application from by selecting **Start → All Programs → VMware → VMware Player**. If the CentOS virtual machine is already open, skip ahead to Step 4.



Figure 6 – The VMware Player Application Icon

2. Select the virtual machine named **CentOS-6.3-amd64-ZedBoard-linux** from the selections on the left and then click on the **Play virtual machine** button to the right.

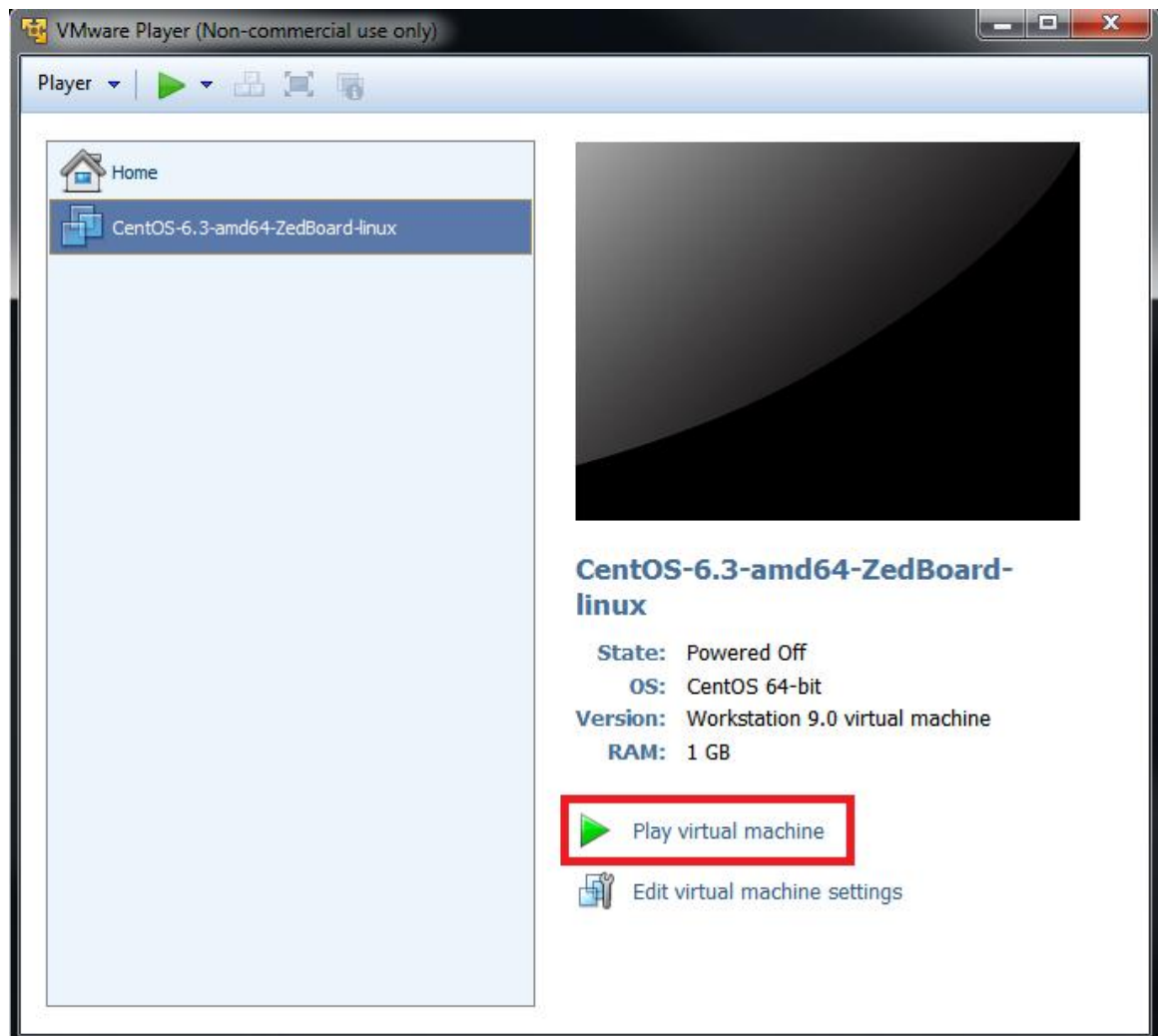


Figure 7 – The VMware Player Application

3. If prompted for a workstation login, click on the user entry **training** and enter the password **Avnet** in order to log into the system.

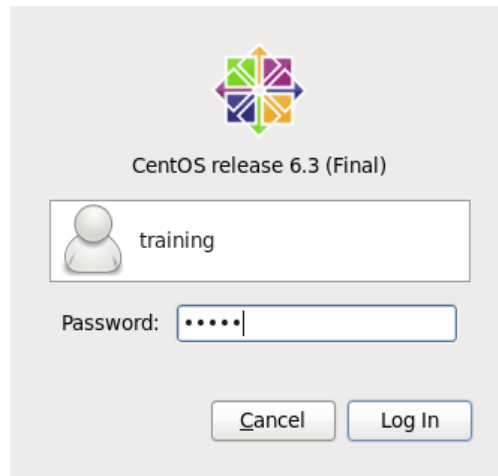


Figure 8 – The CentOS Workstation Login

4. If a terminal is not already open on the desktop of the CentOS guest operating system, open a terminal window through the **Applications→System Tools→Terminal** menu item. If a terminal is already open, bring that terminal session into focus on the desktop.

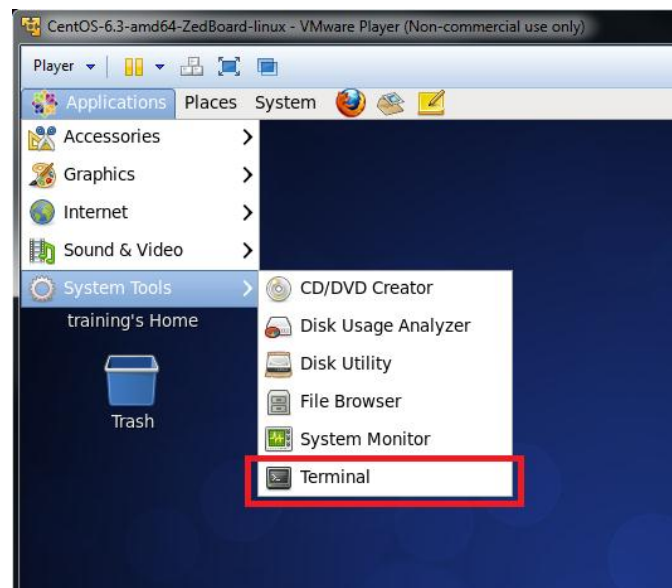


Figure 9 – Launching the CentOS Terminal from the Desktop

5. Create a new folder to act as the starting point for the target root file system staging.

```
$ mkdir ~/_rootfs/
```

6. Change from the home directory into the BusyBox source directory.

In case you are not familiar with BusyBox and are wondering what it is needed for, here is an explanation from the BusyBox developers:

“BusyBox combines tiny versions of many common UNIX utilities into a single small executable. It provides replacements for most of the utilities you usually find in GNU fileutils, shellutils, etc. The utilities in BusyBox generally have fewer options than their full-featured GNU cousins; however, the options that are included provide the expected functionality and behave very much like their GNU counterparts. BusyBox provides a fairly complete environment for any small or embedded system. BusyBox has been written with size-optimization and limited resources in mind.”

For the purpose of this exercise, BusyBox will serve as our Linux system command interpreter and is somewhat analogous to the Windows command interpreter **cmd.exe** executable.

```
$ cd ~/busybox/
```

7. For good measure (sometimes a necessity) run a make distribution clean command against the BusyBox 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 distclean
```

8. A default configuration is included for Zynq target. Configure BusyBox for the Zynq target by using the included defconfig.

```
$ make ARCH=arm defconfig
```

9. The build can be customized from the supplied defaults by running the application configuration menu.

```
$ make menuconfig
```

10. In the menu options **BusyBox Settings→Installation Options→BusyBox installation prefix**, set the install location to the **/home/training/_rootfs** folder.

Then select **Ok** to commit the changes to the installation prefix.

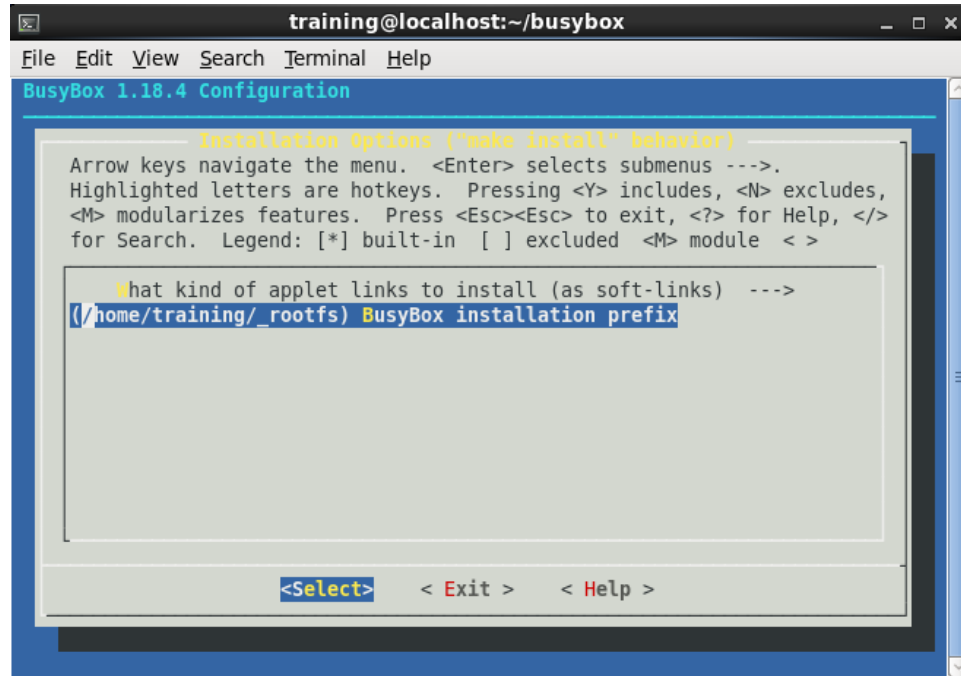


Figure 10 – Configuring BusyBox Using the Configuration Menu

11. Exit from the menu configuration tools and save the updated BusyBox configuration. To do this, perform the following actions:

Press the **<ESC>** key three times to move up the menu tree and exit the menu configuration tool.

Select the **Yes** option to save the updated configuration down to the **.config** file.

12. Build BusyBox using the following command using the updated configuration.

```
$ make ARCH=arm install
```

Once the build has completed, the terminal output should look similar to that shown in Figure 11.

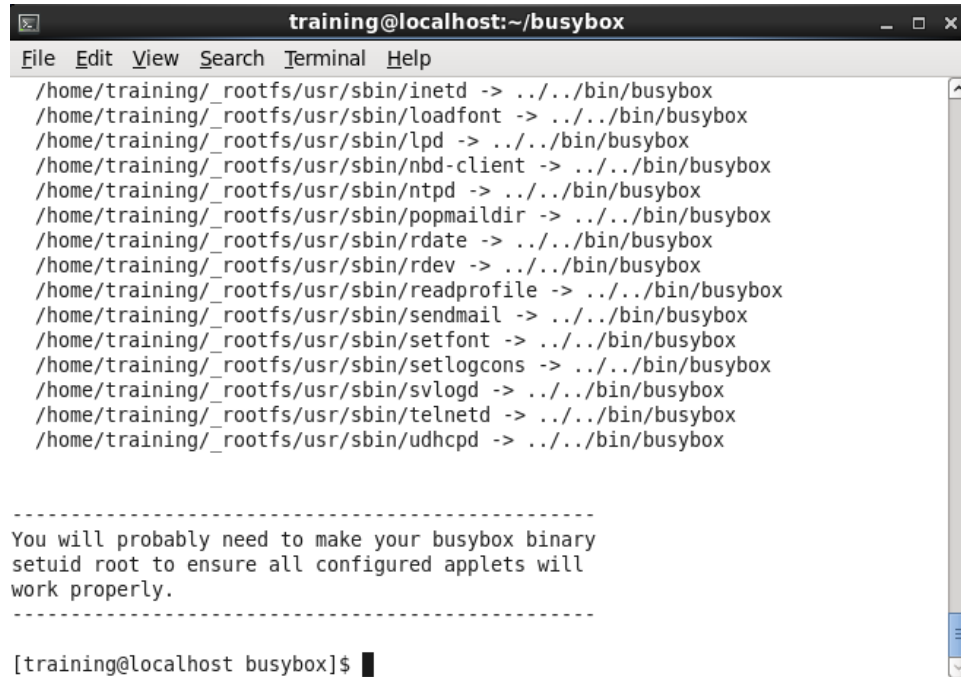
A terminal window titled 'training@localhost:~/busybox' showing the output of the 'make ARCH=arm install' command. The output lists 17 files being installed to '/bin/' and a note about setting permissions. The prompt is '[training@localhost busybox]\$'.

Figure 11 – Building the BusyBox Application

13. Change from the current directory into the home directory.

```
$ cd ~
```

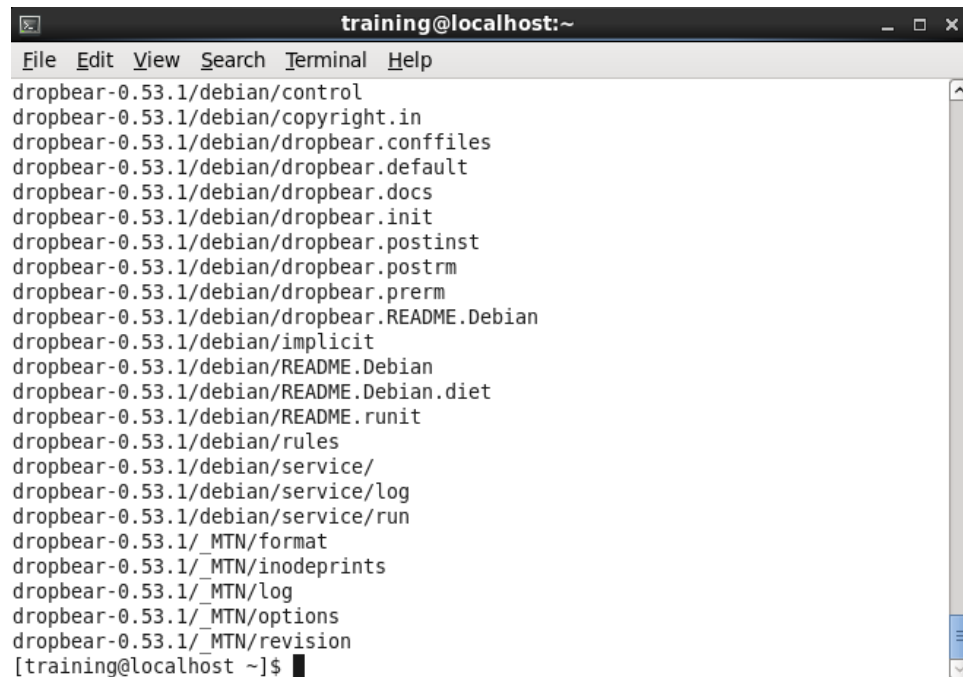
14. Unpack the Dropbear source code archive.

In case you are not familiar with Dropbear and are wondering what it is needed for, here is an explanation from the Dropbear developers:

"Dropbear is a relatively small SSH 2 server and client. It runs on a variety of POSIX-based platforms. Dropbear is open source software, distributed under a MIT-style license. Dropbear is particularly useful for "embedded"-type Linux (or other Unix) systems, such as wireless routers."

For the purpose of this exercise, Dropbear will serve as our remote secure shell server. This is required for later labs to allow the SDK Remote System Explorer to take control of the target system and launch gdb-server for debugging applications.

```
$ tar -zxvf dropbear-0.53.1.tar.gz
```

A screenshot of a terminal window titled 'training@localhost:~'. The window shows the output of the command 'tar -zxvf dropbear-0.53.1.tar.gz'. The output lists the contents of the archive, including files like 'control', 'copyright.in', 'dropbear.conf', 'dropbear.default', 'dropbear.docs', 'dropbear.init', 'dropbear.postinst', 'dropbear.postrm', 'dropbear.prerm', 'dropbear.README.Debian', 'dropbear.implicit', 'dropbear.README.Debian', 'dropbear.README.Debian.diet', 'dropbear.README.runit', 'dropbear.rules', 'dropbear.service', 'dropbear.service/log', 'dropbear.service/run', 'dropbear._MTN/format', 'dropbear._MTN/inodeprints', 'dropbear._MTN/log', 'dropbear._MTN/options', and 'dropbear._MTN/revision'. The prompt '[training@localhost ~]\$' is visible at the bottom.

```
training@localhost:~  
File Edit View Search Terminal Help  
dropbear-0.53.1/debian/control  
dropbear-0.53.1/debian/copyright.in  
dropbear-0.53.1/debian/dropbear.conf  
dropbear-0.53.1/debian/dropbear.default  
dropbear-0.53.1/debian/dropbear.docs  
dropbear-0.53.1/debian/dropbear.init  
dropbear-0.53.1/debian/dropbear.postinst  
dropbear-0.53.1/debian/dropbear.postrm  
dropbear-0.53.1/debian/dropbear.prerm  
dropbear-0.53.1/debian/dropbear.README.Debian  
dropbear-0.53.1/debian/implicit  
dropbear-0.53.1/debian/README.Debian  
dropbear-0.53.1/debian/README.Debian.diet  
dropbear-0.53.1/debian/README.runit  
dropbear-0.53.1/debian/rules  
dropbear-0.53.1/debian/service/  
dropbear-0.53.1/debian/service/log  
dropbear-0.53.1/debian/service/run  
dropbear-0.53.1/_MTN/format  
dropbear-0.53.1/_MTN/inodeprints  
dropbear-0.53.1/_MTN/log  
dropbear-0.53.1/_MTN/options  
dropbear-0.53.1/_MTN/revision  
[training@localhost ~]$
```

Figure 12 – Unpacking the Dropbear Source Code Archive

15. Change from the current directory into the Dropbear source directory.

```
$ cd ~/dropbear-0.53.1/
```

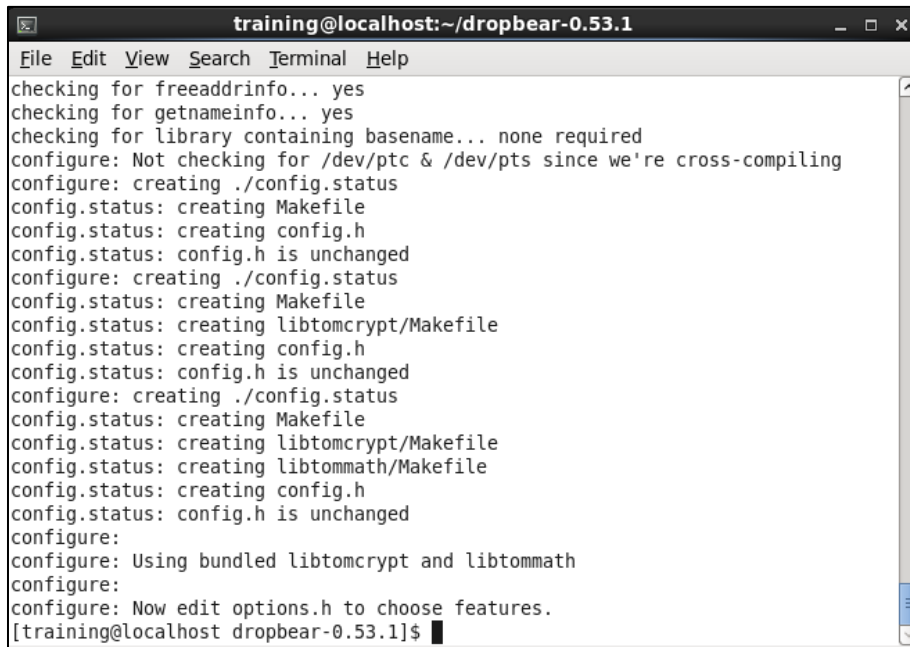

16. Configure Dropbear for the Zynq target by using the configure command.

Hint: Up until this point many of the command line entry instructions have been manageable to enter by hand. Some of the commands in the following steps take up multiple lines and can be difficult to manually enter accurately. One workaround to this challenge is to **Copy** the following single line from this PDF document into the VM clipboard by right-clicking in Adobe Reader and selecting the **Copy** option. Then **Paste** contents from the VM clipboard to the CentOS terminal window by right-clicking in the terminal window and selecting the **Paste** option. The backslash character '\' allows a command entry to be followed by an additional line entry when followed by the **<Enter>** key.

```
$ ./configure --prefix=/home/training/_rootfs \  
--host=arm-xilinx-linux-gnueabi --disable-zlib \  
LDFLAGS="-Wl,--gc-sections" \  
CFLAGS="-ffunction-sections -fdata-sections -Os"
```

Note in the above command the -Wl flag is a **hyphen** followed by uppercase **W** and a lower case **L** character sequence.

Once the Dropbear configuration has completed, the terminal output should look similar to that shown in Figure 13.



```
training@localhost:~/dropbear-0.53.1  
File Edit View Search Terminal Help  
checking for freeaddrinfo... yes  
checking for getnameinfo... yes  
checking for library containing basename... none required  
configure: Not checking for /dev/ptc & /dev/pts since we're cross-compiling  
configure: creating ./config.status  
config.status: creating Makefile  
config.status: creating config.h  
config.status: config.h is unchanged  
configure: creating ./config.status  
config.status: creating Makefile  
config.status: creating libtomcrypt/Makefile  
config.status: creating config.h  
config.status: config.h is unchanged  
configure: creating ./config.status  
config.status: creating Makefile  
config.status: creating libtomcrypt/Makefile  
config.status: creating libtommath/Makefile  
config.status: creating config.h  
config.status: config.h is unchanged  
configure:  
configure: Using bundled libtomcrypt and libtommath  
configure:  
configure: Now edit options.h to choose features.  
[training@localhost dropbear-0.53.1]$
```

Figure 13 – Configuring Dropbear

17. Build Dropbear using the following command using the configuration that was just created:

```
$ make PROGRAMS="dropbear dbclient dropbearkey scp \
dropbearconvert" MULTI=1 strip
```

The **MULTI=1** argument shown in the command above is used for systems without much space where a single binary is created to serve multiple purposes. This will save disk space by avoiding repeated executable code between different executable files. If you are familiar with the operation of BusyBox, it is the same principle. The strip argument shown in the command above is used to discard the symbols from the resulting object files during the build process

Once the build has completed, the terminal output should look similar to that shown in Figure 14.

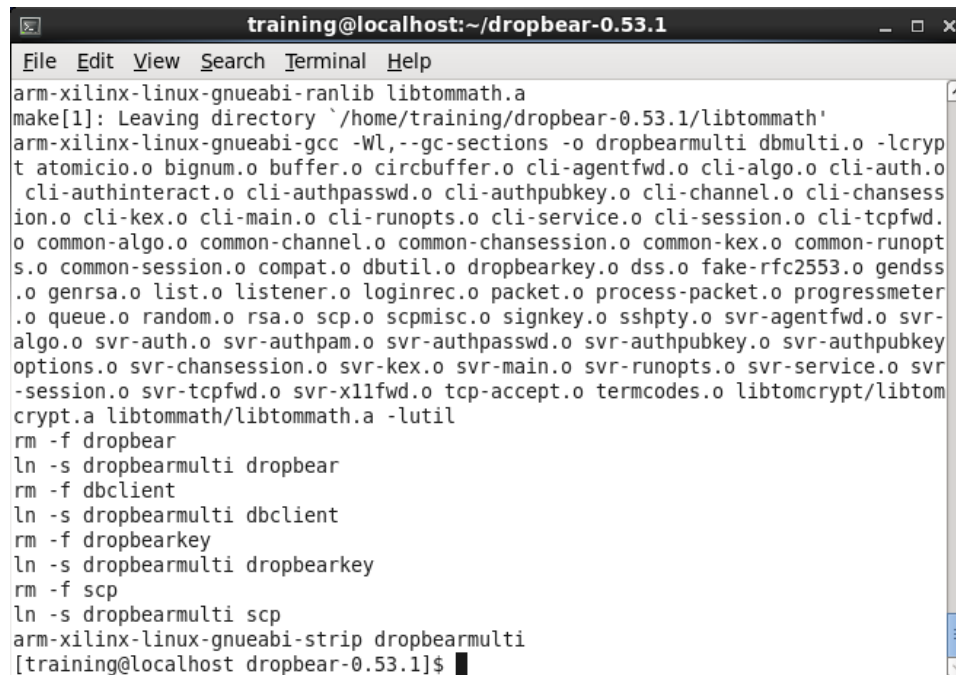
A terminal window titled 'training@localhost: ~/dropbear-0.53.1' showing the output of the 'make' command. The output lists various object files being built, such as libtommath.a, dropbearmulti, dbclient, dropbearkey, scp, and dropbearconvert. It also shows the removal of old files and the creation of new symlinks. The process concludes with the command 'arm-xilinx-linux-gnueabi-strip dropbearmulti' and the prompt '[training@localhost dropbear-0.53.1]\$'.

Figure 14 – Building Dropbear

18. Install Dropbear into the root file system staging folder. For SpeedWay event laptops, the superuser password is **Avnet** and is case sensitive.

```
$ sudo make install
```

19. Change to the root file system staging folder and create a symbolic link to scp.

```
$ cd ~/_rootfs/  
$ ln -s sbin/dropbear /home/training/_rootfs/usr/bin/scp
```

20. Create the base configuration directory and the Dropbear configuration directory.

```
$ mkdir etc etc/dropbear
```

21. Dropbear operates as a secure shell server on Zedboard in order to facilitate secure network communications with remote hosts. Dropbear requires two private key files to be placed in the target system etc/dropbear folder. The two key files have been generated ahead of time and are provided in the Lab 2.2 folder on the host operating system.

WARNING: These Dropbear private keys are provided as an example for this SpeedWay and are intended for experimental purposes only. Misuse of these keys could result in compromising the security of your Zynq based design. For any designs based upon this SpeedWay course material, your own unique set of private keys is strongly recommended. These unique key files can be generated later on with a new unique seed on the Zynq target using the following commands:

```
dropbearkey -t rsa -f dropbear_rsa_host_key
```

```
dropbearkey -t dss -f dropbear_dss_host_key
```

Locate the **dropbear_dss_host_key** and **dropbear_rsa_host_key** files in the Lab 2.2 folder on the host operating system by using Windows explorer to navigate to the following folder:

C:\Speedway\Fall_12\Zynq_Linux\lab2_2

Right click on these files and select the **Copy** option which will place the selected files in the Virtual Machine clipboard.

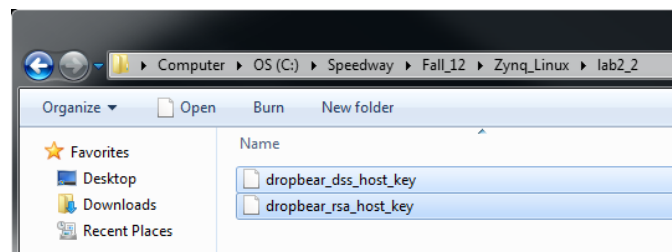


Figure 15 – Copying the Dropbear Key Files to Virtual Machine Clipboard

22. If a file browser window is not already open from a previous exercise, open a file browser window through the **Applications→System Tools→File Browser** menu item.

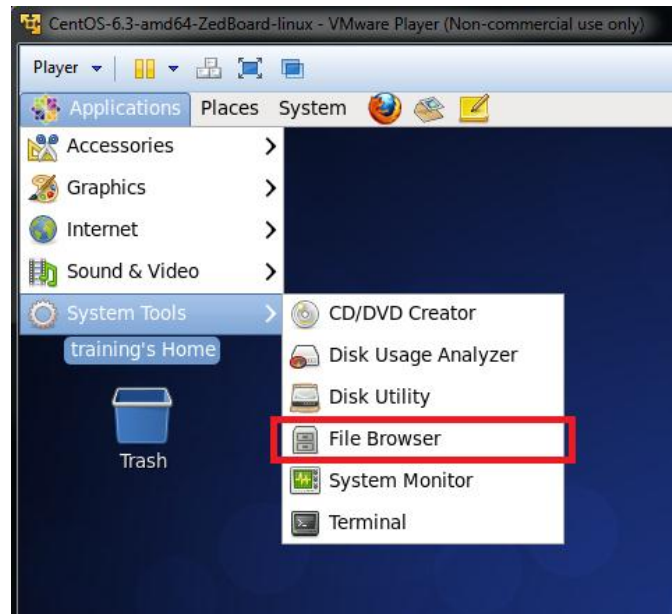


Figure 16 – CentOS File Browser

23. Paste the key files into the root file system staging folder **/home/training/_rootfs/etc/dropbear/** on the guest operating system by using the CentOS File Browser.

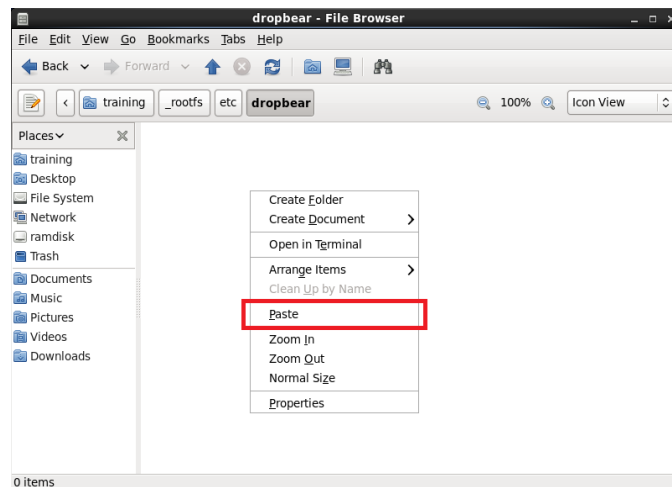


Figure 17 – Dropbear Keys Pasted to the Guest Machine

24. For the target Zynq Linux system to be used with the SDK Remote System Explorer in later labs, the **sftp-server** application must be installed. This tool allows SDK to copy files to the Root File System across the network during application development. These files are precompiled for the convenience of this lab since they are derived from the **zlib** library, the **openssl** package, and the **openssh** package.

Locate the **lib** folder in the Lab 2.2 folder under the **dev-tools** subfolder on the host operating system by using Windows explorer to navigate to the following folder:

C:\Speedway\Fall_12\Zynq_Linux\lab2_2\dev-tools

Right click on the **lib** folder and select the **Copy** option which will place the selected folder contents in the Virtual Machine clipboard.

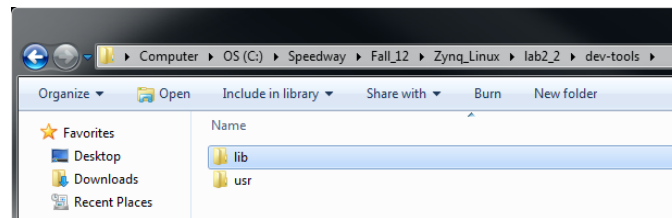


Figure 18 – Copying the System lib Folder to Virtual Machine Clipboard

25. Paste the system **lib** folder into the root file system staging folder **/home/training/_rootfs/** on the guest operating system by using the CentOS File Browser.

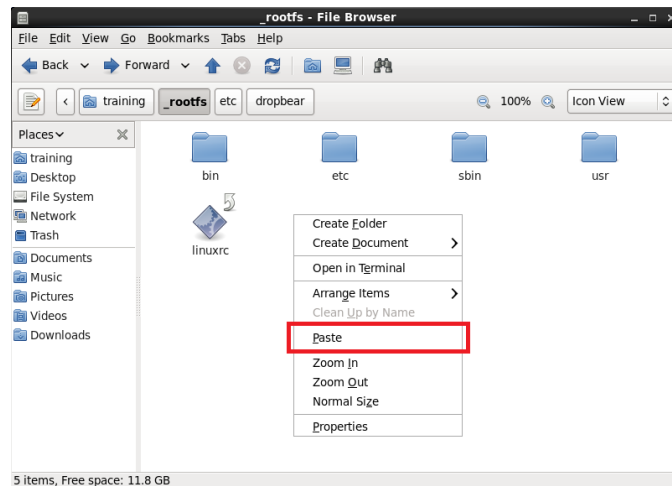


Figure 19 – The System lib Folder Pasted to the Guest Machine

26. Locate the **lib** and **libexec** folders in the Lab 2.2 folder under the **dev-tools\usr** subfolder on the host operating system by using Windows explorer to navigate to the following folder:

C:\Speedway\Fall_12\Zynq_Linux\lab2_2\dev-tools\usr

Select both the **lib** and **libexec** folders, one way to do this is to right-click each file while holding down the **<CTRL>** key on the keyboard for multiple file selections. Right click on the **lib** and **libexec** folders and select the **Copy** option which will place the selected folder contents in the Virtual Machine clipboard.

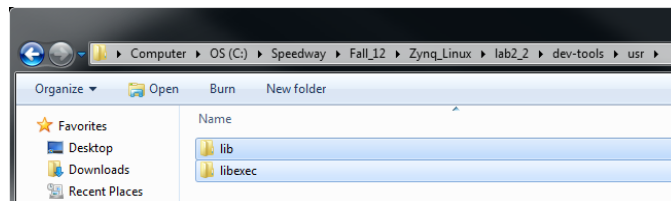


Figure 20 – Copying the User lib and libexec folders to Virtual Machine Clipboard

27. Paste the user **lib** and **libexec** folders into the root file system staging folder **/home/training/_rootfs/usr/** on the guest operating system by using the CentOS File Browser.

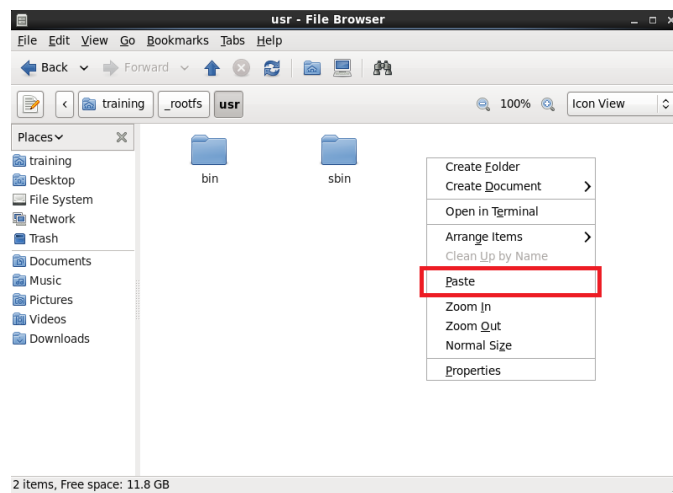


Figure 21 – The User lib and libexec Folders Pasted to the Guest Machine

Questions:

Answer the following questions:

- *Which folder contains the root file system that is being staged for the target platform?*

Experiment 3: Create System Folders and Startup Scripts

So far, the essential applications have been built configured and installed to the root file system staging area for the Zynq target platform.

The CodeSourcery tool-chain includes a pre-built standard C library along with some helper applications like gdb-server that should be copied to the target root file system. The C standard library provides macros, type definitions, and functions for tasks like string handling, mathematical computations, input/output processing, memory allocation and several other operating system services. You may already be familiar with some of these functions such as `printf()`, `atoi()`, `strlen()`, or `malloc()`.

In this next experiment, the system folders and files will be installed and the startup scripts created and placed in the staged root file system.

Experiment 3 General Instruction:

Create startup scripts and install the system folders and files into the staged root file system.

Experiment 3 Step-by-Step Instructions:

1. First, make sure the `/home/training/_rootfs/` folder is the current working directory by using the `pwd` command. If the working directory shown is not the `/home/training/_rootfs/` folder change directories to that folder.

```
$ pwd
```

2. Create additional file links to the `libz` library.

```
$ cd lib
$ ln -s libz.so.1.2.7 libz.so
$ ln -s libz.so.1.2.7 libz.so.1
$ cd ..
```

3. Create additional file links to the `libcrypto` library.

```
$ cd usr/lib
$ ln -s libcrypto.so.1.0.0 libcrypto.so
$ cd ../..
```

4. Copy in the libraries supplied with the CodeSourcery toolchain using the following commands.

Note that notifications about omitted directories are expected behaviors of these two commands.

```
$ cp ../CodeSourcery/Sourcery_CodeBench_Lite_for_Xilinx_\
GNU_Linux/arm-xilinx-linux-gnueabi/libc/lib/* lib
$ cp ../CodeSourcery/Sourcery_CodeBench_Lite_for_Xilinx_\
GNU_Linux/arm-xilinx-linux-gnueabi/libc/usr/lib/* usr/lib
```

5. Strip the libraries of debug symbols to reduce the amount of room that they take up on disk.

Note that notifications about unrecognized file formats are expected behaviors of these two commands.

```
$ arm-xilinx-linux-gnueabi-strip lib/*
$ arm-xilinx-linux-gnueabi-strip usr/lib/*
```

6. Copy in the supplied CodeSourcery system tools.

```
$ cp ../CodeSourcery/Sourcery_CodeBench_Lite_for_Xilinx_\
GNU_Linux/arm-xilinx-linux-gnueabi/libc/sbin/* sbin
$ cp ../CodeSourcery/Sourcery_CodeBench_Lite_for_Xilinx_\
GNU_Linux/arm-xilinx-linux-gnueabi/libc/usr/bin/* usr/bin
```

7. Create the remaining needed root file system directory structure.

```
$ mkdir dev etc/init.d mnt opt proc root sys \
tmp var var/log var/www
```

- Each Linux system requires a file systems table file which lists all available disks and disk partitions, and indicates how they are to be initialized or otherwise integrated into the overall system's file system. The **etc/fstab** file serves this purpose.

Create the **etc/fstab** file within the root file system staging folder by opening the file with the gedit text editor. This is easily done from the command line using the following command line:

```
$ gedit etc/fstab
```

Copy the following lines from this document into the VM clipboard by right-clicking in Adobe Reader and selecting the **Copy** option:

LABEL=/	/	tmpfs	defaults	0 0
none	/dev/pts	devpts	gid=5,mode=620	0 0
none	/proc	proc	defaults	0 0
none	/sys	sysfs	defaults	0 0
none	/tmp	tmpfs	defaults	0 0

Paste contents from the VM clipboard to the new **etc/fstab** file by right-clicking in the gedit editor window and selecting the **Paste** option.

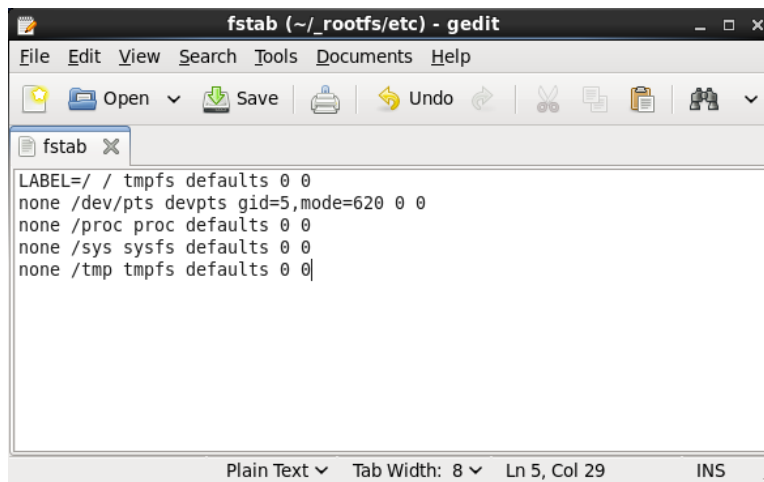


Figure 22 – Using gedit Text Editor to Create etc/fstab File

Exit the gedit text editor using the **File→Quit** menu option and save changes to the **etc/fstab** file.

9. Each Linux system also requires a startup configuration file for the init process. The **/etc/inittab** file contains directions for init on what programs and scripts to run when entering a specific runlevel. The **etc/inittab** file serves this purpose.

Create the **etc/inittab** file within the root file system staging folder by opening the file with the gedit text editor. This is easily done from the command line using the following command line:

```
$ gedit etc/inittab
```

Copy the following lines from this document into the VM clipboard by right-clicking in Adobe Reader and selecting the **Copy** option:

```
::sysinit:/etc/init.d/rcS
# /bin/ash
#
# Start an askfirst shell on the serial ports
ttyPS0::respawn:-/bin/ash

# What to do when restarting the init process
::restart:/sbin/init

# What to do before rebooting
::shutdown:/bin/umount -a -r
```

Paste contents from the VM clipboard to the new **etc/inittab** file by right-clicking in the gedit editor window and selecting the **Paste** option.

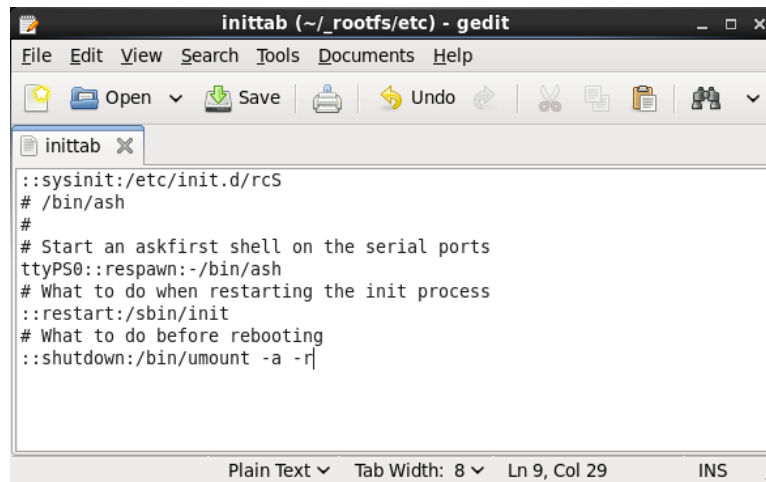


Figure 23 – Using gedit Text Editor to Create etc/inittab File

Exit the gedit text editor using the **File→Quit** menu option and save changes to the **etc/inittab** file.

10. Each Linux system also requires a password file describes user login accounts for the system. The **etc/passwd** file serves this purpose.

Create the **etc/passwd** file within the root file system staging folder by opening the file with the gedit text editor. This is easily done from the command line using the following command line:

```
$ gedit etc/passwd
```

Some Linux systems store the password as an encrypted string which is generated by the **passwd** command. Copy the following single line (which also contains the encrypted root password) from this document into the VM clipboard by right-clicking in Adobe Reader and selecting the **Copy** option:

```
root:$1$qC.CEbjC$SVJyqm.IG.gkElhaeM.FD0:0:0:root:/root:/bin/sh
```

Paste contents from the VM clipboard to the new **etc/passwd** file by right-clicking in the gedit editor window and selecting the **Paste** option.

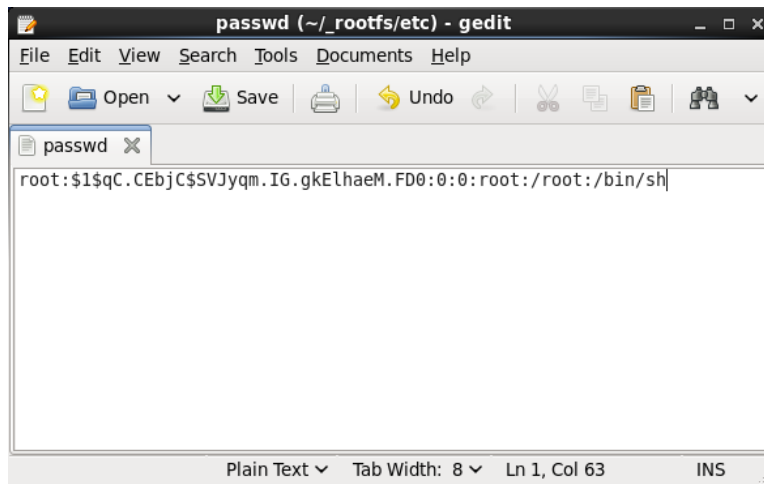


Figure 24 – Using gedit Text Editor to Create etc/passwd File

Exit the gedit text editor using the **File→Quit** menu option and save changes to the **etc/passwd** file.

11. Each Linux system also requires a system startup script and the **etc/init.d/rcS** script is used when bringing the system up in single user mode. The **etc/init.d/rcS** file serves this purpose.

Create the **etc/init.d/rcS** file within the root file system staging folder by opening the file with the gedit text editor. This is easily done from the command line using the following command line (note that the capital "S" character in the **rcS** file is significant):

```
$ gedit etc/init.d/rcS
```

Copy the following lines from this document into the VM clipboard by right-clicking in Acrobat Reader and selecting the **Copy** option:

```
#!/bin/sh

echo "Starting rcS..."

echo "++ Mounting filesystem"
mount -t proc none /proc
mount -t sysfs none /sys
mount -t tmpfs none /tmp

echo "++ Setting up mdev"

echo /sbin/mdev > /proc/sys/kernel/hotplug
mdev -s

mkdir -p /dev/pts
mkdir -p /dev/i2c
mount -t devpts devpts /dev/pts

echo "++ Starting telnet daemon"
telnetd -l /bin/sh

echo "++ Starting http daemon"
httpd -h /var/www

echo "++ Starting ftp daemon"
tcpsvd 0:21 ftpd ftpd -w /&

echo "++ Starting dropbear (ssh) daemon"
dropbear

echo "rcS Complete"
```

Paste contents from the VM clipboard to the new **etc/init.d/rcS** file by right-clicking in the gedit editor window and selecting the **Paste** option.

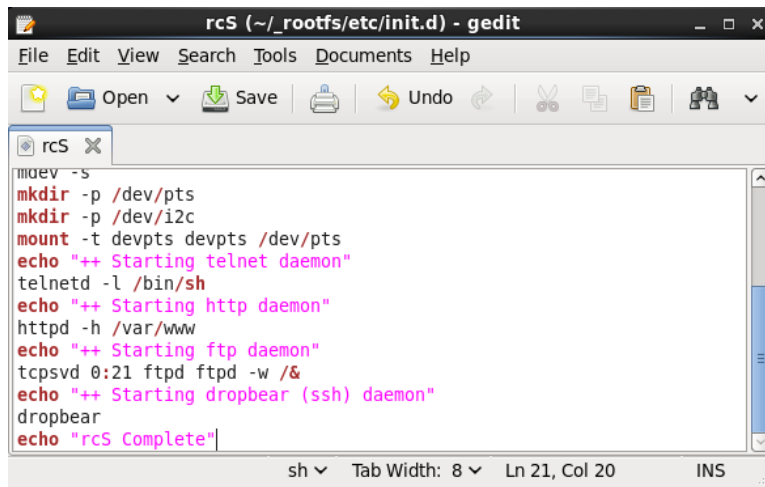


Figure 25 – Using gedit Text Editor to Create etc/init.d/rcS File

Exit the gedit text editor using the **File→Quit** menu option and save changes to the **etc/init.d/rcS** file.

12. Set permissions which will allow the **etc/init.d/rcS** script to execute. Change file ownership properties of the files in the staged file system to the system user **root**. For SpeedWay event laptops, the superuser password is **Avnet** and is case sensitive.

The first command that follows is **chmod** which is used to modify file permissions for user read, write, and execute properties. The 755 number passed as an argument is the octal representation of the file permissions which makes this file readable by anyone and writable by the owner only.

```
$ sudo chmod 755 etc/init.d/rcS
$ sudo chown -R root *
$ sudo chgrp -R root *
```

13. The root file system is completely staged at this point. An image file will be created next which can be used as the RAM disk image on the target platform. Change the current working directory back to the user home directory.

```
$ cd ~
```

14. Create a disk image that is large enough to contain the root file system contents.

The **dd** utility converts and copies the specified input file and outputs the contents to a file named **ramdisk32M.image** containing the specified number of bytes (1K) for a specified number of iterations.

In this case is the **/dev/zero** device is used as the data input. This is simply a special Unix system device which returns 0x00 data each time that it is read from it.

In this case, 32,768 1K byte blocks of 0x00 data bytes are copied into the specified image file essentially creating a blank image file.

```
$ dd if=/dev/zero of=ramdisk32M.image bs=1024 count=32768
```

15. Format the disk image using the **mke2fs** tool to make an EXT2 file system.

Use the force option **-F** to create a file system within the specified image file **ramdisk32M.image** and label the disk **ramdisk**.

The block size is specified as **1024** bytes using the **-b** option. The **-m 0** option specifies that no file system blocks are reserved for the super-user.

```
$ mke2fs -F ramdisk32M.image -L "ramdisk" -b 1024 -m 0
```

16. Use the **tune2fs** utility to change the file system settings using the **-i** flag such that periodic file system integrity checks are disabled.

```
$ tune2fs ramdisk32M.image -i 0
```

17. Mount the disk image to a folder and copy the **~/_rootfs/** staging folder contents to the mounted image. For the **sudo** entry, you may be required to enter the superuser password. On SpeedWay event laptops, the superuser password is **Avnet** and is case sensitive.

For the mount command, use the **-o loop** option to mount via the loop device which is a pseudo-device which makes a file accessible as a block device.

```
$ mkdir ramdisk
```

```
$ sudo mount -o loop ramdisk32M.image ramdisk/
```


18. Copy the staged root file system contents to the mounted RAM disk image. For the **sudo** entry, you may be required to enter the superuser password. On SpeedWay event laptops, the superuser password is **Avnet** and is case sensitive.

The -R option specifies a recursive copy through the subfolders.

```
$ sudo cp -R _rootfs/* ramdisk
```

19. Unmount the RAM disk image from the mount point using the **umount** command. For the **sudo** entry, you may be required to enter the superuser password. On SpeedWay event laptops, the superuser password is **Avnet** and is case sensitive.

```
$ sudo umount ramdisk/
```

20. Compress the RAM disk image using the **gzip** tool.

Specify the verbose command option -v so that the percentage reduction can be observed. Also, use the compression level 9 which is the slowest compression method which yields the best size file output.

```
$ gzip -v9 ramdisk32M.image
```

21. If a file browser window is not already open from a previous exercise, open a file browser window through the **Applications→System Tools→File Browser** menu item.

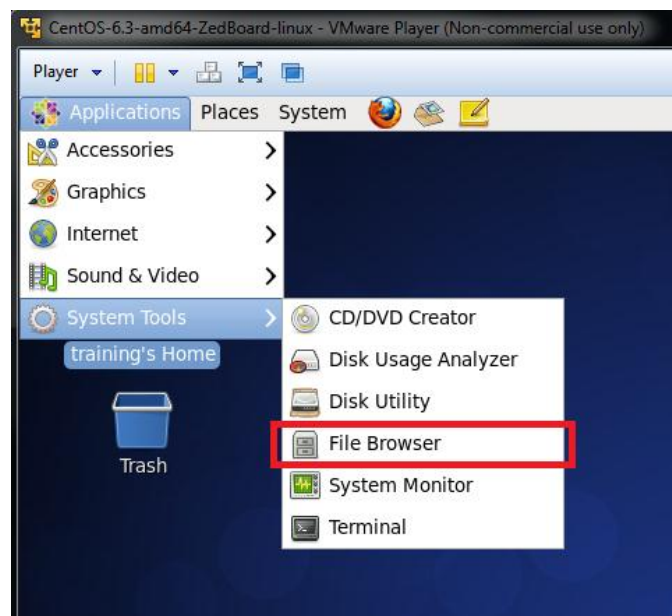


Figure 26 – CentOS File Browser

22. Locate the file **/home/training/ramdisk32M.image.gz** which is the root file system RAM disk image needed to run Linux on Zynq. Right click on this file and select the **Copy** option which will place the selected file in the Virtual Machine clipboard.

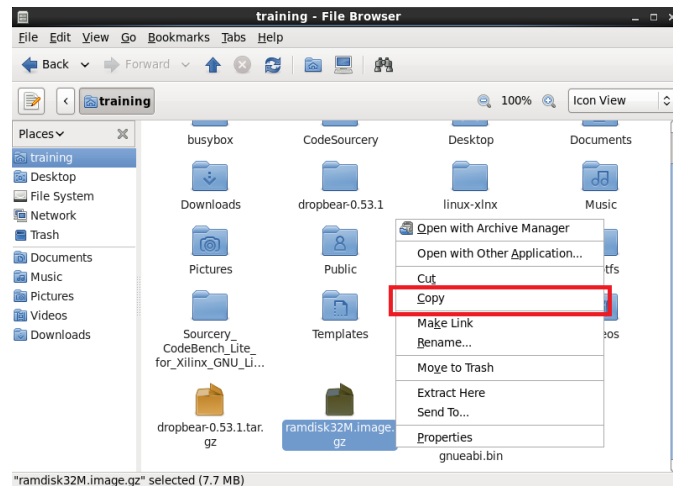


Figure 27 – Copying RAM Disk Image to Virtual Machine Clipboard

23. Paste the compressed RAM disk image into the Lab 2.3 folder on the host operating system by using Windows Explorer to navigate to the following folder:

C:\Speedway\Fall_12\Zynq_Linux\lab2_3

This RAM disk image file will be used in the next lab session. You may have also noticed a **devicetree.dtb** file in this folder. This file serves as the platform description used by the kernel to initialize the hardware and related drivers. We will discuss this file in further detail later in Part 3 and Lab 3.3.

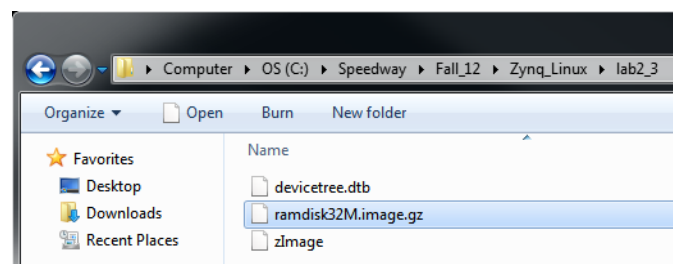


Figure 28 – RAM disk Image Copied to the Host Machine

A basic root file system directory structure has been created with minimal startup scripts. Busybox, Dropbear, base toolchain utilities, and basic libraries were installed to the RAM disk image to complete the base root file system. The RAM disk image is now built and ready to be used on the target ZedBoard.

Questions:

Answer the following questions:

- *Which file contains the compressed root file system RAM disk image format that is needed for Linux to run on the Zynq platform?*

Exploring Further

If you have additional time and would like to investigate more...

- Take a look at installing another useful application such as iPerf and repeat the necessary exercises to rebuild the RAM disk image containing the new application.

This concludes Lab 2.2.

Revision History

Date	Version	Revision
12 Sep 12	00	Initial Draft
01 Oct 12	01	Revised Draft
18 Oct 12	02	Course Release
14 Jan 13	05	ZedBoard.org Training Course Release

Resources

<http://www.zedboard.org>

<http://www.xilinx.com/zynq>

<http://www.xilinx.com/planahead>

<http://git-scm.com/>

<http://www.kernel.org>

Answers

Experiment 1

- *How is a working copy of the BusyBox code base obtained?*

A working copy is obtained using the Git clone command against the repository URL.

- *How is a release copy of the Dropbear code base obtained?*

A release copy is obtained by downloading the source code archive from a known good URL.

Experiment 2

- *Which folder contains the root file system that is being staged for the target platform?*

/home/training/_rootfs/

Experiment 3

- *Which file contains the compressed root file system RAM disk image format that is needed for Linux to run on the Zynq platform?*

ramdisk32M.image.gz